Temporal Unpredictability Detection of Real-time Video Sequence

Yang Liu

A thesis submitted for the degree of

Doctor of Philosophy of the University of London

and for the Diploma of Membership of the

Imperial College

Department of Electrical and Electronic Engineering

Imperial College of Science, Technology and Medicine

University of London

July 2007

Abstract

Human has an unmatched ability to rapidly direct attention to potential objects of interest in a dynamic environment. Given the vast amount of visual data that has to be processed, there is often no time for detailed visual analysis over the entire scene. Instead, attention is directed to the "salient" regions. In this thesis, a spatiotemporal saliency framework is developed to emulate the attentive analysis function of the human visual system. This allows more complex and higher level analysis to be performed on the most salient regions of the dynamic scene in the video sequence.

There are both software and hardware contributions: the software framework that detects spatiotemporal saliency in video sequences and the efficient hardware architectures for the building blocks of the framework. The novelty of our saliency framework is that the predictability of the motion of spatial features is employed to determine temporal saliency. Moreover, the proposed framework is flexible and offers opportunities for hardware design space exploration. Two of the important algorithms of the framework are identified to have room for potential improvement in terms of hardware efficiency. These are the eigenvalue computation and the Kalman filter computation. The investigations of these two algorithms in search for efficient hardware architectures lead to significant performance improvement over previous approaches. These two pieces of hardware work greatly facilitate the hardware implementation of

the proposed framework to detect spatiotemporal saliency in real-time video sequence.

List of Publications

The following publications have been published or prepared for publication.

- Yang Liu, Christos-Savvas Bouganis, and Peter Y. K. Cheung. "A spatiotemporal saliency framework." *In Proceedings of IEEE International Conference on Image Processing (ICIP 2006)*,pages 437-440, Atlanta, GA, Oct. 2006.

- Yang Liu, Christos-Savvas Bouganis, Peter Y. K. Cheung, Philip H. W. Leong, and Stephen J. Motley. "Hardware efficient architectures for eigenvalue computation." *In Proceedings of the conference on Design, automation and test in Europe*, pages 953-958, Munich, Germany, 2006.

- Yang Liu, Christos-Savvas Bouganis, and Peter Y. K. Cheung. "Efficient mapping of a Kalman filter into an FPGA using Taylor expansion." *accepted and to appear in Proceedings of the 17th International Conference on Field Programmable Logic and Applications (FPL)*, Amsterdam, Netherlands, 2007.

- Yang Liu, Christos-Savvas Bouganis, and Peter Y. K. Cheung. "Real-time Spatiotemporal Saliency" Book Chapter 12 of A. Bharath and

M. Petrou (editors) "Reverse Engineering the Human Vision System", Artech Publishers, to appear in 2008.

- Yang Liu, Christos-Savvas Bouganis, and Peter Y. K. Cheung. "A Method of Identifying a Measure of Feature Saliency in a Sequence of Images" U.K. Patent P41663GB Oct 2006 (filed).

Acknowledgements

A few lines are too short to make a complete account of my deep appreciation for my supervisor Professor Peter Y. K. Cheung, who is the most important person for my Ph.D. work. Peter was my interviewer when I was an undergraduate candidate for Imperial College and three years later I became one of his Ph.D. students. I wish to thank Peter for who he is as a professor: an enthusiastic, sagacious, honest minded individual with broad interests who provides steady support in the most difficult times. Peter has helped me to gain experiences at international conferences, by generously giving me the funding to travel to numerous places all over the world and to meet other researchers. This experience has strengthened my Ph.D. with advices and suggestions from a wide range of people and further motivated my research with fresh ideas. I wish to thank Peter for his trust and constant encouragements which have been essential throughout the last three years.

My apprentice pursuit into the world of science would not have been the same without Dr. Christos Bouganis's approach to research. I would also like to thank Christos for the countless hours I have spent with him discussing

research, proof reading papers and this dissertation. I want to extend my appreciation for his patience, for his full devotion for others and for his great understanding.

It has been a distinct privilege for me to work with Dr. George Constantinides. For his great insight and sharp comments, discussions with George have always been very fruitful.

I am thankful to the people in the Circuits and Systems lab for the friendly and sociable working atmosphere. I have been fortunate enough to have had the support from so many intelligent people and without it this would not have been possible. For their technical support, I would like to thank Sutjipto Arifin, Terrence Mak, Chun Te Ewe, Qiang Liu, Edward Ang, and Alastair Smith. Thank you for various helpful discussions.

My deepest gratitude is to my family, I would like to thank my mum, my dad, and my grandparents for their love, their support, and their confidence throughout the past twenty-six years. Thanks also go to my girlfriend for providing me constant motivation. They have always taken care of me and encouraged me in all my decisions. My parents have always put education as a first priority, it is through their encouragement and care that I have made it through all the steps to reach this point in life, and I could not have done it without them.

For everyone listed here I am eternally grateful for their help.

To my grandfather

Zhi Zhong Liu

who sadly passed away during my Ph.D. study

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Objectives

One of the most important features of the human visual system is that it has multiple resolutions, which allows efficient visual data acquisition and processing. The selection of the location to which the high resolution vision is placed is controlled by a visual attention model. In a dynamic environment, this visual attention model can rapidly direct attention to potential objects of interest.

Several modern electronic visual systems [WLB03, GP98, AUCS98, IKN98] adopt multi-resolution sensors inspired by the human visual system. These systems successfully reduce data throughput. However, such systems lack the control unit that directs its highest resolution region to capture the most interesting part in the scene, which imposes serious restrictions on the applicability of the system. Saliency detection algorithms can bestow visual systems with

the intelligence to know where to look. These algorithms serve similar purpose as the human attention model. In this thesis, a novel spatiotemporal saliency framework is developed to emulate the attentive analysis function of the human visual system. More complex and high level analysis of a video can therefore focus on the most salient regions detected by our framework. Practical applications of the saliency framework also include surveillance, compression and prediction of viewers' attention in advertisement.

The proposed saliency framework is design to detect saliency in real-time video sequences. With this design scenario in mind, through the use of different techniques, the storage requirement is greatly reduced. For real-time applications, the computation can be too intensive for software implementation and hardware acceleration is required. In terms of hardware mapping, the most challenging parts within the framework are identified. Efficient hardware architectures are proposed and compared against existing ones in the literature. The proposed hardware architectures demonstrate significant performance improvement supported with solid implementation results.

1.2 Organization of the thesis

Chapter 2 first describes three aspects of human visual system: the retina, the multi-resolution vision, and the visual attention model. The importance of the visual attention model is then highlighted. A number of saliency detection algorithms are described. The advantages and disadvantages of different

approaches are discussed. The distinctive nature of our proposed algorithm is also introduced in this chapter. A summary and a comparison of the saliency detection algorithms is provided at the end of the chapter.

Chapter 3 gives an overview of the proposed spatiotemporal saliency framework. This framework consists of a number of modules. The mechanism and justification of joining all the modules to detect saliency is explained. Performance evaluation of the saliency framework using video sequences is also provided. Chapter 3 shows how the framework can successfully detect salient movement in the video without the involvement of object detection and segmentation even in a complex scene.

In Chapter 4, three major modules of the proposed framework are described in more detail. These are the **2D Feature Detection** module, the **Feature Tracking** module, and the **Adaptive Decision** module. The second section of this chapter investigates the hardware architectures for eigenvalue computation, which is an important part of the proposed saliency framework. In terms of area and throughput, hardware experimental results show that the proposed *Approximate Jacobi Method* in systolic architecture is the most efficient system to compute eigenvalues for $n \times n$ symmetric matrices. For high speed application, a pipelined architecture resorting to the *Algebraic Method* to compute the eigenvalues for 3×3 symmetric matrices is proposed. It successfully achieves very high throughput using only 22% of the area compared to that of the *Approximate Jacobi Method.*

In Chapter 5, the **Temporal Motion Prediction** module and **Suppres-**

sion module are elaborated. Then a new hardware architecture for the Kalman filter is proposed, which reduces the computational complexity of a conventional Kalman filter by approximating the matrix inversion function using Taylor expansion. Hardware implementation results demonstrate that the proposed Approximate Kalman filter implementation has significantly larger throughput using less hardware resources, achieving at the same time similar accuracy and convergence rate.

Chapter 6 concludes this thesis.

1.3 Statement of Original Contributions

The work in this thesis contributes both in the software and the hardware field and has resulted in three papers at internationally renowned conferences.

- From the software perspective, our contribution is the proposition of a temporal saliency detection framework that is suitable for real-time video sequences [LBC06a]. This proposed framework consists of several computational blocks.

- For embedded system design, hardware implementation is needed for real-time performance of the framework. We identify two of the computational blocks that draw considerable computational power and have potential for efficient hardware implementations. The first hardware contribution is an efficient hardware architecture for eigenvalue computation [LBC+06b].

- The second hardware contribution is the Kalman filter implementation. [LBC07]

Chapter 2

Literature Review

This chapter briefly describes the aspects of the human visual system that are pertinent to the topic of this thesis. A wide range of electronic visual systems are inspired by the human visual system. Some representative ones are described in this chapter. The lack of "intelligence" is the common limitation in these electronic systems. Several characteristics of human visual attention and how they inspire artificial attention algorithms are discussed. Such algorithms can improve the performance of the electronic visual systems. A set of well-known attention algorithms proposed in the literature are then described.

2.1 Retina

The human retina is the light-sensitive layer of tissue that lies at the back of the eyeball, sending visual impulses through the optic nerve to the brain. As illustrated in Figure 2.1, it consists of photoreceptor cells connected to

other layers of neuron cells to accomplish various tasks of the information pre-processing of a received image before transmission to the brain.

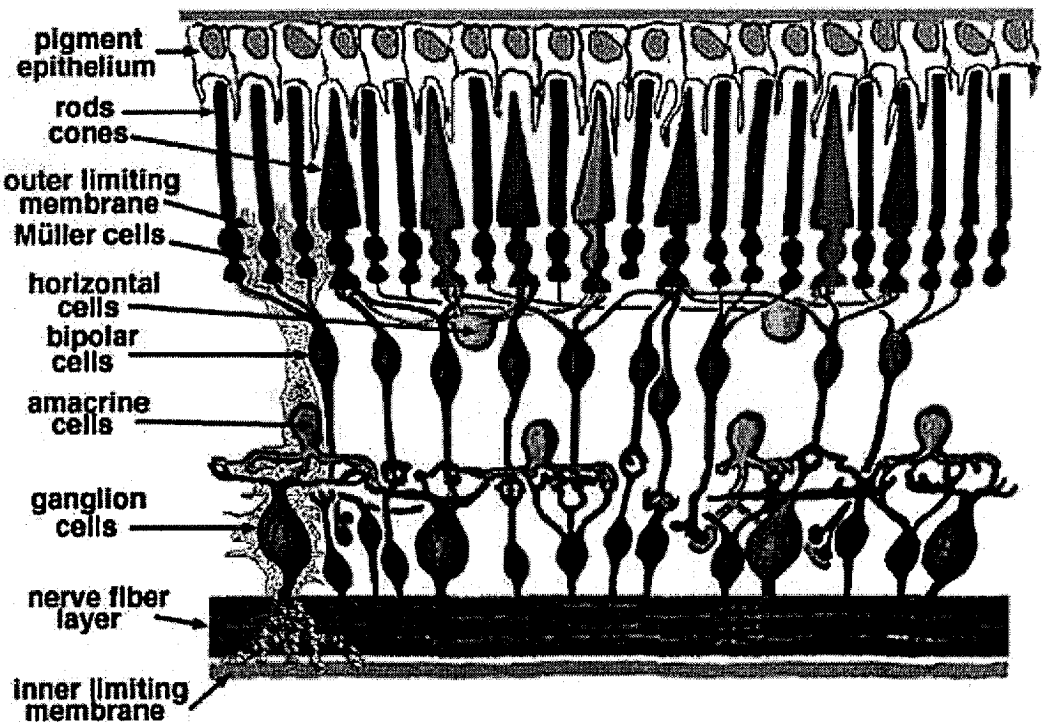


Figure 2.1: Photoreceptors and various cells in the retina[ret]

Each type of neuron cell has its specific function within the retina. The function fulfilled by each type of neuron from the angle of image processing is summarized in Table 2.1.

Neurons	General Function
Photoreceptor Cells	Image Sensor Array(CMOS,CCD)
Horizontal Cells	connectivity
Bipolar Cells	edge extraction computation
Amacrine Cells	connectivity
Ganglion Cells	object detection

Table 2.1: Neurons and their image processing functionalities

2.1.1 Silicon Retina

Mead [Mea90] suggested that biological systems have similar power, space and computational constraints to compact VLSI electronics. Researchers have been motivated to develop electronic visual systems resembling parts of biological retina. Such biologically inspired visual systems, often referred to as silicon retina, have similar data flow as the human vision system. The advancing silicon fabrication technology facilitates the development of different designs of image sensor chips inspired by the human retina [Del93, DM94, Del00, ZLD03, BWHD02, Boa96, CBM+02, KY01, BM96].

Bourbakis and Mertoguno [BM96] present a digital system named KYDON that resembles a human retina. Functionalities such as edge detection and line recognition are distributed among several layers. The lower-level layers resemble the human retina. The functions of lower-level layers are summarized in Table 2.2. By comparing Table 2.1 and Table 2.2, it is evident that the functional structure of the lower-level layers of KYDON resembles that of the human retina.

Layer	Function
0 - Photoreceptor	Convert intensity of light into 8-bit binary value
1 - Edge detector	Perform image segmentation based on edge information
2 - Line recognizer	Break each contour into lines, each represented by one of its two ends
3 - Graph generator	Provide graphical representation for regions segmented by Layer 1

Table 2.2: The lower-level layer of KYDON [BM96]

2.2 Multi-resolution vision

Mapping the functions of the human retina onto silicon chips is one topic of research on silicon retina. Another topic is to take advantage of the topology of the retina, which has multiple resolution.

Photoreceptor cells in the retina are not distributed uniformly. The fovea is at the centre of the macula region of the retina. The density of photoreceptor cells is the highest at the fovea and decreases rapidly away from it. As a result, the spatial resolution of the human visual system declines dramatically and smoothly away from the point of gaze. Psychological experiments have been conducted to measure the contrast threshold[1] CT as a function of spatial frequency f (cycles per degree) and retinal eccentricity[2] e (degree). A model [GP98] that fits the experimental data is given by

$$CT(f,e) = \frac{1}{64}\exp\left(0.106\frac{e+2.3}{2.3}f\right) \tag{2.1}$$

(2.1) has an intuitive interpretation. If f and e are both large, then CT will be large. The implication is that a feature, which has fine texture and is distant from the fovea, is less visible to human because the threshold CT is high. Psychological experimental results demonstrate that the human retina has multiple resolution, which is high at the fovea and low at the periphery.

To imitate the topology of the human retina, multi-resolution electronic

[1]Contrast threshold is the minimal amount of contrast, or differences between two shades of objects, needed in order to detect a pattern.

[2]Retinal eccentricity is the angular distance from the point of fixation.

visual systems have been developed. They represent a compromise between spatial resolution and field of view. Assigning higher resolution to a certain area allows that area to be investigated in greater detail. At the same time, by assigning lower resolution to other area, a larger field of view can be obtained without increasing the amount of data excessively. Computational power can also be saved when the visual information in the field of view is processed to acquire the contextual information so that the scene can be understood.

In an electronic multi-resolution system, two topologies are most widely adopted to resemble the multi-resolution characteristic of the human retina. These are the Log-polar and the Cartesian topologies.

2.2.1 Log-Polar electronic multi-resolution system

Sandini *et al.* presented the implementation in silicon of a very faithful retina-like sensor [SQS+00]. The variation of spatial resolution of the sensor is in accord with that of the human retina, which is reflected by the variation of contrast threshold of human vision in (2.1). The layout of the sensor is illustrated in Figure 2.2. Although this topology imitates that of a human retina, it is very difficult to fabricate image sensor arrays arranged in log-polar topology due to manufacturing limitation. A more feasible alternative is to use a uniform sensor and transform the acquired image to log-polar topology in software. An example using such a software transformation technique for video coding is described below.

One of the main objectives in video coding is to eliminate redundant infor-

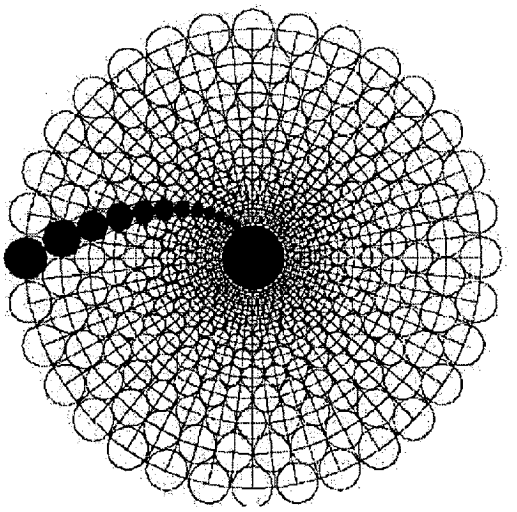
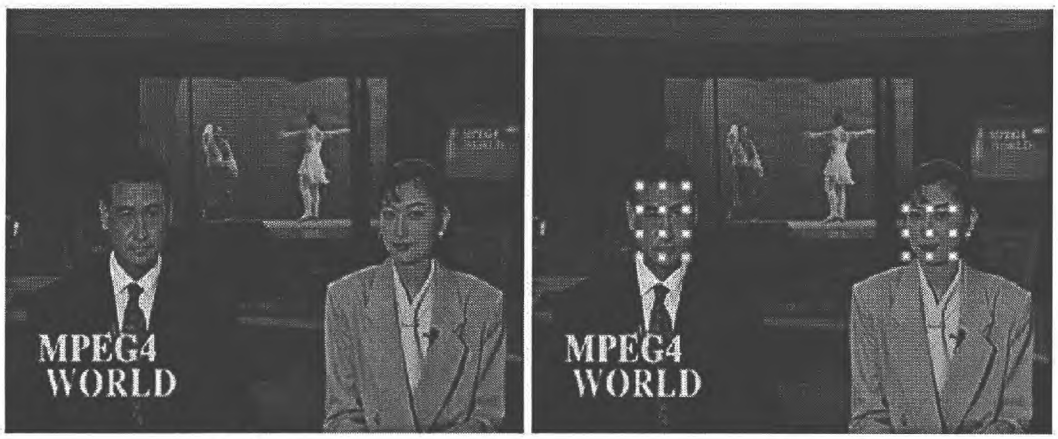


Figure 2.2: Layout of receptors' placing for a log-polar structure[SQS+00]

mation to achieve the compression of a video stream. Wang *et al.* [WLB03] proposed Foveation Scalable Video Coding (FSVC) that takes into account the human visual system. The concept is that provided the point at which the user looks on the screen is known, the spatial resolution around that point can be reduced according to the contrast threshold in (2.1). In this way, the user's eye contrast threshold can match the resolution of the video. As a result, video is compressed without user's notice.

Wang *et al.* [WLB03] assume that human faces are the most interesting regions within the video. Human faces are detected first using skin colour information as illustrated in Figure 2.3. Each frame of the video is transformed into log-polar representation. Then the video is compressed using a wavelet-based method [TRK98, CW99] so that the spatial resolution around faces is reduced. The compressed frame using a uniform resolution compression scheme and the FSVC scheme are shown in Figure 2.4(a) and Figure 2.4(b),

respectively. Human faces, the assumed gaze points of users, have the highest resolution and the surrounding regions have decreasing resolution as shown in Figure 2.4(b).



(a) (b)

Figure 2.3: (a) Input image; (b) human faces selection [WLB03]

(a) (b)

Figure 2.4: (a) Compressed image with uniform resolution scalable coding; (b) Compressed image with FSVC [WLB03]

2.2.2 Cartesian electronic multi-resolution system

In contrast to log-polar topologies, Cartesian topologies have rectangular geometries for all resolution levels and therefore are simpler to be manufactured. The additional advantage of Cartesian topologies over log-polar ones is that most of existing algorithms, cameras, Digital Signal Processing theory, and development tools have been developed for Cartesian image format.

Bandera and Scott [BS89], proposed Cartesian topologies with a high pixel density central area surrounded by concentric rings as shown in Figure 2.5. The size of pixels increases exponentially with the distance from the centre. This results in a high resolution at the centre and dramatically decreasing resolution towards the circumference.

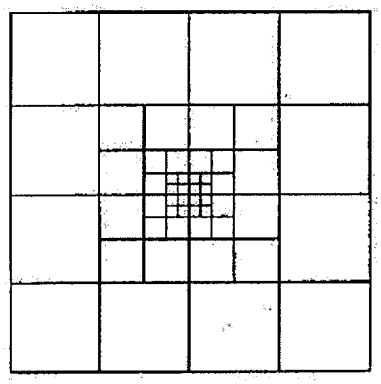


Figure 2.5: Root Lattice [BS89]

Camacho and Arrebola [AUCS98, CAS98] proposed a Cartesian-based multi-resolution system called Shifted Fovea Multi-resolution Geometries (SFMG). They used Field-Programmable Gate Arrays (FPGA) and Charge-Coupled Devices (CCD) to develop *electronically* reconfigurable structures for rectangular Field of View. The advantage of using the *electronically* reconfigurable gaze

fixation mechanisms is the lack of any mechanical movement of the sensor. It uses a mono-resolution CCD device whose output can be electronically configured to the desired geometries. It allows shifting of the highest resolution fovea to any region in the Field of View while selecting a resolution ratio at the surrounding regions of the fovea. The shifted geometries are illustrated in Figure 2.6.

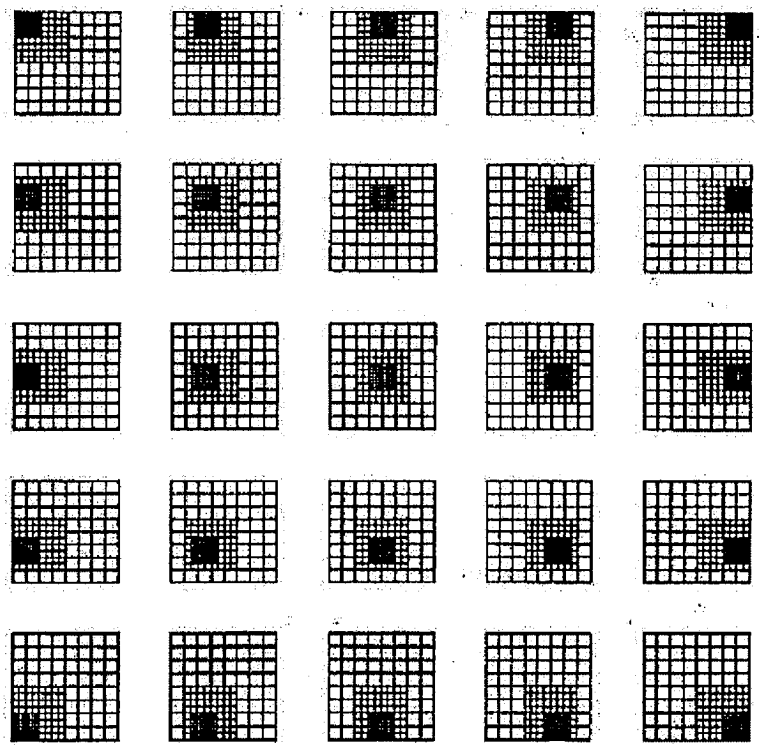


Figure 2.6: Shifted Geometries ($m = 2, d = 2$) [AUCS98]

However, a question naturally arises: how do we know where to place the high resolution fovea?

In [WLB03], the region of interest is assumed to be a human face. However, it is also more pertinent to assume that the region of interest changes from time to time because the user is not likely to fix the attention to one point

of the screen all the time. To acquire real-time accurate measurement of the position of the fixation point, infrared eye tracker has to be used. This would make the system quite expensive. In [GP98], the region of interest need to be manually specified by the user. These approaches impose serious restrictions on the applicability of the system.

One of the most important features of the human visual system is that it has multiple resolutions, which allows efficient visual data acquisition and processing. The selection of the location to which the high resolution vision is placed is controlled by a visual attention model. The electronic multi-resolution systems described previously in this section lack such a control unit. In the next section, the characteristics of human visual attention and some biologically-inspired visual attention models are described.

2.3 Visual attention

In complex dynamic environments, the overwhelming amount of visual data prevents detailed visual analysis over the entire scene. The attention has to be directed to the "salient" regions and humans have the unmatched ability to do so [ROC96]. Visual saliency refers to the idea that certain parts of a scene are pre-attentively distinctive and create some form of immediate significant visual arousal. In everyday life, visual attention is controlled by both bottom-up factors and top-down processes [Jam90, TG80]. These processes are closely linked to the controlling of eye movements so that gaze is guided to the salient

locations in the visual scene [MA03].

Bottom-up process is stimuli-driven. It is based on the human reaction to external stimuli, such as bright colour, distinctive shape or unusually motion. At the front end of the visual system, the retina, the saliency detection appears to be formed entirely on the basis of bottom-up stimulus aspects. Retinal ganglion cells are particularly responsive to edges in the luminance profile while they are unresponsive to uniform illuminations. The pixel-by-pixel representation of the visual environment created by photoreceptor cells is compressed into a representation emphasizing discontinuities.

Top-down process is task driven, where *a priori* knowledge of the target or current goals are known before the detection process. There are several electrophysiological [Mot93, RLS98, MM99] and many functional imaging [BDY99, GHB99, SDST99, RBH00] studies that demonstrate top-down attentional influences in the primary visual cortex. Knowledge and expectations allow us to focus on elements, parts or details of a visual scene that we might otherwise have missed [CS02]. Top-down process aids vision by enabling the brain to create, maintain and change a representation of what is important while we scan a visual scene.

Current psychological evidence supports the idea that orienting to saliency regions is modulated by both bottom-up and top-down signals. Corbetta *et al.* [CS02] propose that visual attention is controlled by two partially segregated neural systems. One system is involved in the cognitive selection of sensory information and responses. The second system is used during the detection of

behaviourally relevant sensory events, particularly when they are salient and unattended.

The first system implies that behavioural relevance (top-down) of objects that reflect our expectations might influence the sensory salience (bottom-up) of the objects in the visual system. For example, if we are searching for a red bird in a crowd of birds, the sensory (or bottom-up) distinctiveness of red object interacts with the ongoing cognitive (top-down) goal of finding a red object. Uninformative sensory stimuli are not effective in drawing attention when we are carefully attending to a specific location rather than diffusely attending over a broad spatial extent [YJ90]. This is called inattentional Blindness [MR98]. Also, distinctive sensory stimuli attract attention more effectively when they are relevant to the task, as when they share attributes or features with the stimuli for which subjects are actively looking. [FRJ92]. In this case, a red flower would draw attention when we are searching for a red bird.

The second system implies that highly salient stimuli need little attentional resource allocation [RD03, Not00]. Strong sensory stimuli orient attention towards unexpected events of potentially high behavioural significance. For example, when we are looking for crabs on the beach, a hole in the ground draws our attention, which has very strong sensory salience. Although it shares none of the features or attributes with a crab, it has the potential of finding one because it can be an entrance to a crab dwelling.

In order to model human visual attention, the concept of saliency map is introduced. A saliency map is a topographic representation of relative stimulus

strength and behavioural relevance across visual space [Tre03]. Activities in the saliency map determine which objects are selected for recognition and action [WOL94]. Therefore, the performance of the multi-resolution visual system described in Section 2.2 can be greatly improved by having a visual attention module to guide the multi-resolution sensor.

2.3.1 Spatial Saliency Detection

Spatial saliency detection finds the salient parts in a static image, which draw our attention. There are numerous methods of detecting spatial saliency. Kadir *et al.* [KB01] resort to information theory and the saliency measure is the entropy of the grey-value distribution of a local region. Regions corresponding to high signal complexity tend to have flatter distributions hence higher entropy. Oliva *et al.* [OTCH03] analyzed the global distribution of low-level spatial features to render a saliency map. A biologically-plausible model proposed by Itti *et al.* [IKN98] and an effective model proposed by Ma *et al.* [MZ03] are described in more detail in the remaining of this subsection. Both models are widely recognized.

Spatial Saliency Model by Itti *et al.* [IKN98]

Itti *et al.* [IKN98] propose the following algorithm to generate the saliency map of an image. Firstly, they convert the input image to a pyramidal image data structure. This is created by reproducing a copy of the image at progressively lower resolutions. Input colour image is progressively low-pass filtered and

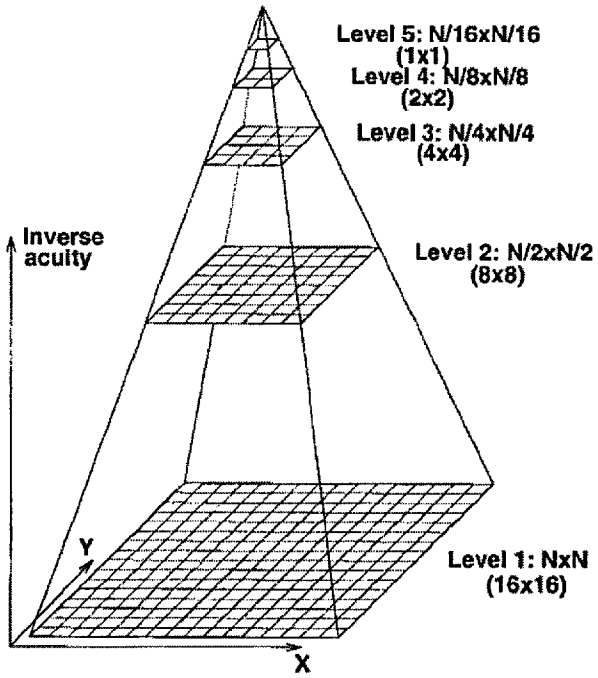


Figure 2.7: Representation of a pyramidal image data structure

subsampled to form such a pyramidal image data structure [GBG+94]. Imagine a pyramid with the image at the lowest resolution at the top and the highest (original) resolution at the bottom such as the one shown in Figure 2.7. Within the pyramid, the image at each level has the same content as the original image. Figure 2.8 shows an example of a pyramidal image data structure with 9 levels of resolution.

After the pyramidal image data structure is constructed, the saliency map can be built according to the work flow shown in Figure 2.9. Intensity, colour and orientation are the three features whose local spatial discontinuities are extracted by "centre-surround" operations. The "centre-surround" operation is inspired by biological visual receptive fields. Typical visual neurons are most sensitive in a small region of the visual space (the centre), while stimuli

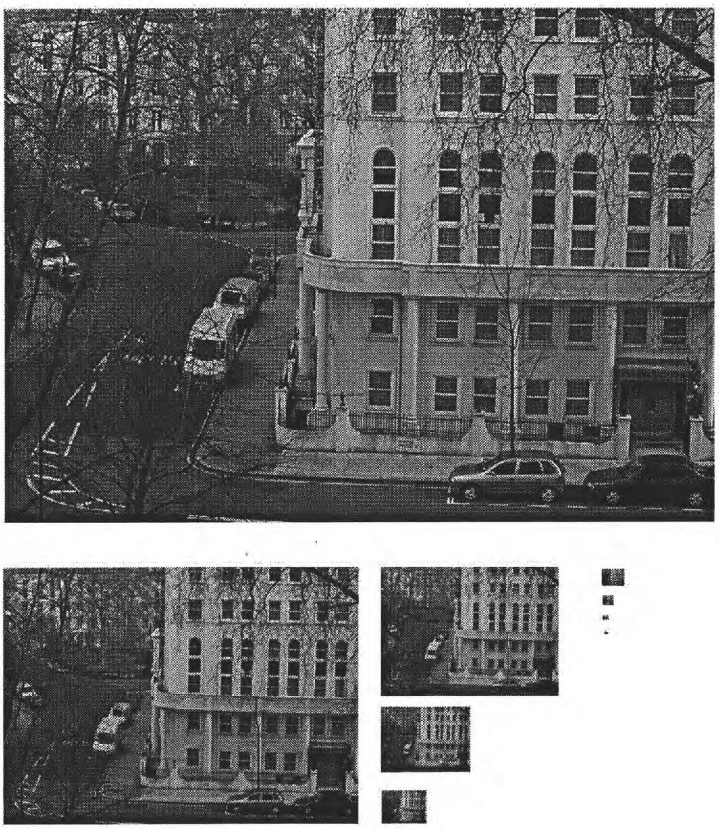


Figure 2.8: Example of pyramidal image data structure

presented in a broader, weaker antagonistic region concentric with the centre (the surround) inhibit the neuronal response [Lev91]. It is particularly suitable for detecting locations which stand out from their surround. Itti *et al.* implemented the centre-surround operation as the difference between fine and coarse levels within the pyramidal image data structure. The across-level difference is obtained by interpolation to the finer level and point-by-point subtraction. Using various combinations of fine and coarse levels for the centre-surround operation, a pyramidal set of feature maps are computed respectively for intensity contrast, colour contrast, and orientation information as shown in Figure 2.9. Local orientation information is acquired using Gabor filters[3].

Normalization is then carried out to globally promote feature maps in which a small number of strong peaks of activity is present, and to globally suppress feature maps which contain several comparable peak responses. For each feature type, the resolutions of the feature maps within the set are reduced to Level 4 of the pyramid and point-by-point addition is applied to render a "conspicuity map". The intensity, colour, and orientation conspicuity maps are normalized and then summed into the final saliency map.

In order to model the human visual attention, the saliency map is fed into a "winner-take-all" (WTA) neural network [KU85] to ensure the most active location remains, while all other locations are suppressed. The global maximum in the salience map is detected and declared as the winner. The

[3]Gabor filters are the product of cosine grating and a 2D Gaussian envelope. They approximate the receptive field sensitivity profile (impulse response) of orientation-selective neurons in the primary visual cortex [Lev91].

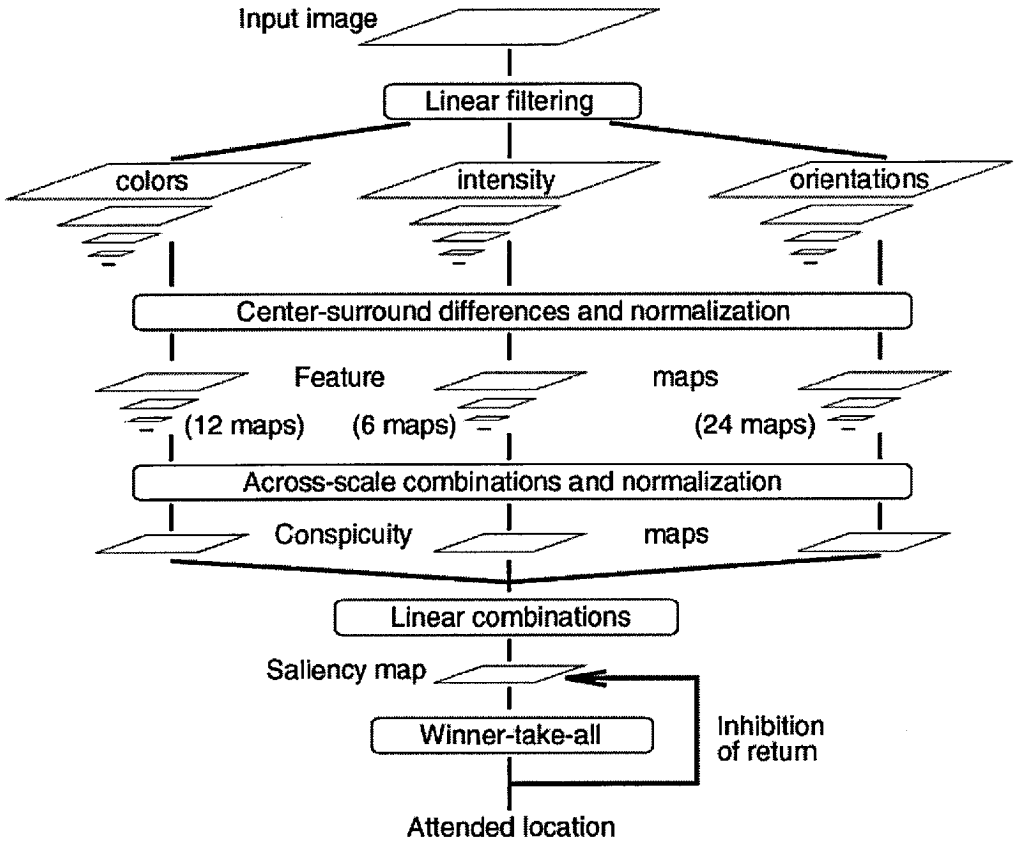


Figure 2.9: General architecture of the saliency-based visual attention model [IKN98]

focus of attention should be directed to this winner. It remains the winner for a prescribed period of time, after which global inhibition of the WTA is triggered. The winner is reset to allow the next most salient point to subsequently rise to become the winner to which the focus of attention is shifted. A demonstration of the model proposed by Itti *et al.* is shown in Figure 2.10.

The main criticism of Itti's model is its high computational cost, which makes it infeasible to be implemented on personal computers in real-time. If only high level applications such as region-based image retrieval and browsing are concerned, it is not necessary to strictly reproduce biological structures by

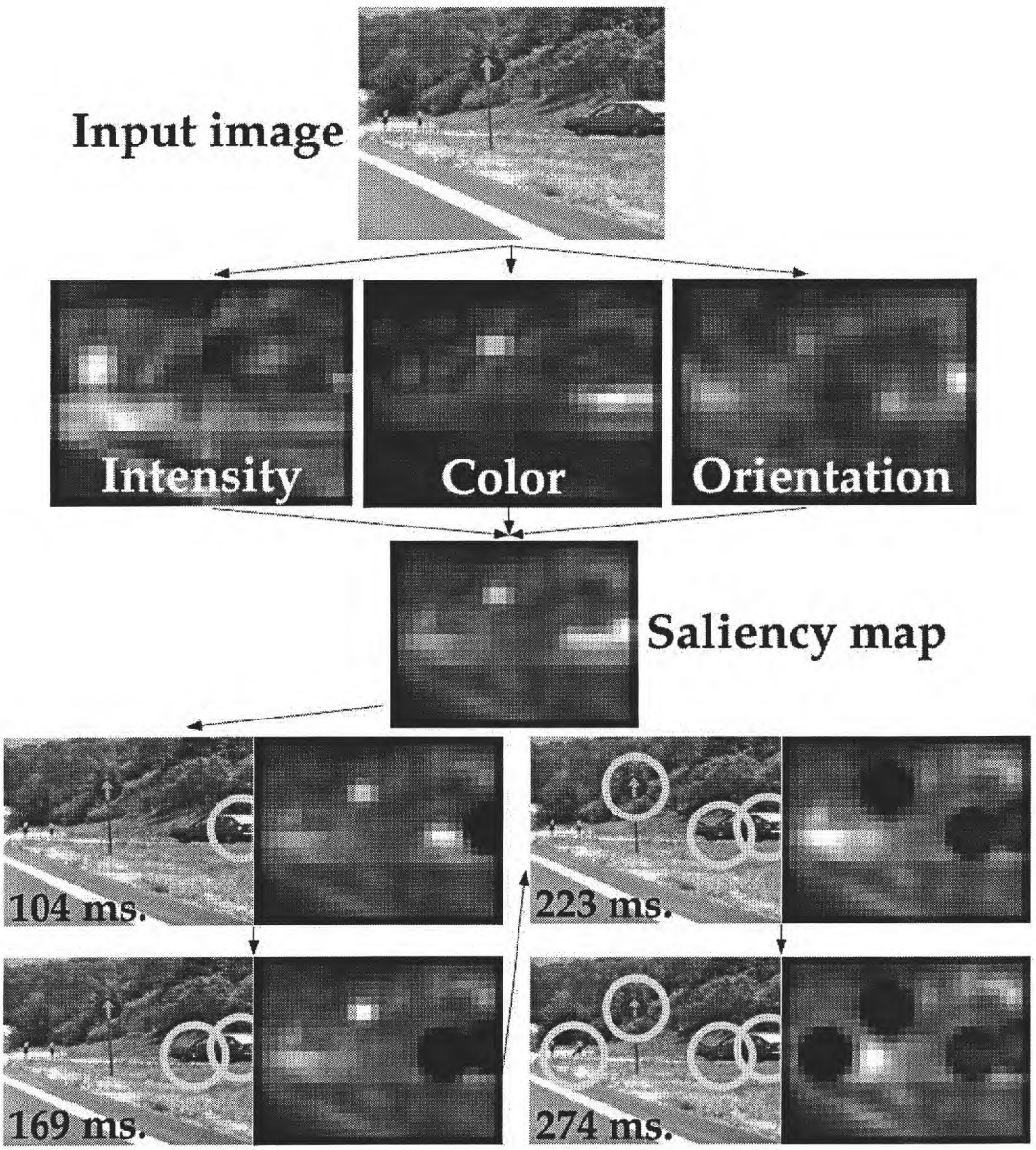


Figure 2.10: Demonstration of Itti *et al.*'s spatial saliency model. Feature maps are extracted from the input image at several spatial resolutions, and are combined into three separate conspicuity maps: Intensity, Color and Orientation. The three conspicuity maps are combined into the single saliency map. A neural winner-take-all network then successively selects, in order of decreasing saliency, the focus of attention. Once a location has been attended to for some brief interval, it is transiently suppressed in the saliency map (dark round areas). As a result, the focus of attention can be shifted to the next most salient point in the saliency map. [Itt00]

computer algorithms [MZ03]. It is observed that in some cases, an algorithm that uses only one feature [MZ03] yields salient regions that are similar to, if no better than an algorithm that uses three features [IKN98].

Spatial Saliency model by Ma *et al.* [MZ03]

Ma *et al.* propose a framework [MZ03] that extracts three attention levels from an image, namely, attended view, attended area, and attended points. The *attended view* means the main part of the image with balance composition and rich information. This can effectively accelerate feature extraction during image retrieval by extracting a sub-image with the most important information. The *attended areas* are those regions which represent important semantics and most possibly attract the human attention. It can provide more detail about important areas for region-based image retrieval. The *attended points* are those locations which perceive local maximum stimuli. They can provide possible search paths on images, which can be used to determine the browsing sequence of image regions.

Ma *et al.* start with finding the attended points. For each pixel in an image, an $M \times N$ window is applied around it. The difference in colour of each pixel from the rest of the pixels in the window is computed to create the saliency map. The local maxima in the saliency map are detected as the attended points. Fuzzy theory [KY95] is used to devise a region growing process that grows some of the attended points to attended area. The fuzzy growing threshold parameter is obtained by minimizing the difference of entropy as

in the image segmentation algorithm [SSA97]. The attended view is defined as a rectangle parameterized by its centre, width and height. The centre of the attention view is the centroid of saliency map. The width and height of the attention view are calculated from the 0^{th} and 1^{st} central moment of the saliency map.

Ma *et al.* make a supposition that the regions with high contrast have rich information and are most likely to attract human attention. Colour contrast is solely employed for the purpose of saliency map generation. This mono-feature model has a reduced computational cost compared to the multi-features saliency model proposed by Itti *et al.* [IKN98]. However, it will not be robust for the cases where colour is not the most useful feature for saliency detection.

2.3.2 Spatiotemporal Saliency Detection

Psychophysical studies have provided strong evidence for an attentional system that is not restricted to the spatial location or to any other single stimulus feature, but rather spreads across multiple dimensions of the same object or surfaces [BPH00]. Spatiotemporal saliency can be thought of as saliency in video. Spatiotemporal saliency detection involves finding regions that correspond to both interesting objects and actions in video sequences. It requires the incorporation of the temporal information in addition to the spatial information.

Laptev *et al.* [LL03] proposed the extension of the Harris corner detector [HS88] from the spatial domain to spatiotemporal domain. According to this

approach, saliency occurs where the image values have significant local variations in both space and time. Zhong *et al.* [ZSV04] divided the video into equal length segments and classified the extracted features into prototypes. The segments of the video that can not be matched with any of the computed prototypes are considered as salient.

Spatiotemporal Saliency model by Cheng *et al.* [CCKW05]

Cheng *et al.* [CCKW05] proposed an algorithm to generate saliency maps for *expert-produced* videos, such as TV shows, movies and commercials. An input video is divided into short video clip called frame-segments. Each video clips consists of a number of frames. For each video frame within a frame-segment, five feature maps are created as in Figure 2.11. These feature maps are based on intensity contrast, red-green colour contrast, blue-yellow colour contrast, x-motion, and y-motion. x-motion and y-motion refer to the horizontal and vertical movements of a specific pixel within a frame, respectively.

A temporal median filter is then applied to find the filtered feature map of each specific feature. In this way, both the spatial saliency distribution and the temporal saliency variation of a video are considered for the final spatiotemporal saliency map. The generation of the filtered intensity feature map is shown in Figure 2.12. The temporal median filter is applied similarly to get the filtered feature maps for the rest of the features. Each filtered feature map represents the general characteristics of a specific feature in a frame-segment.

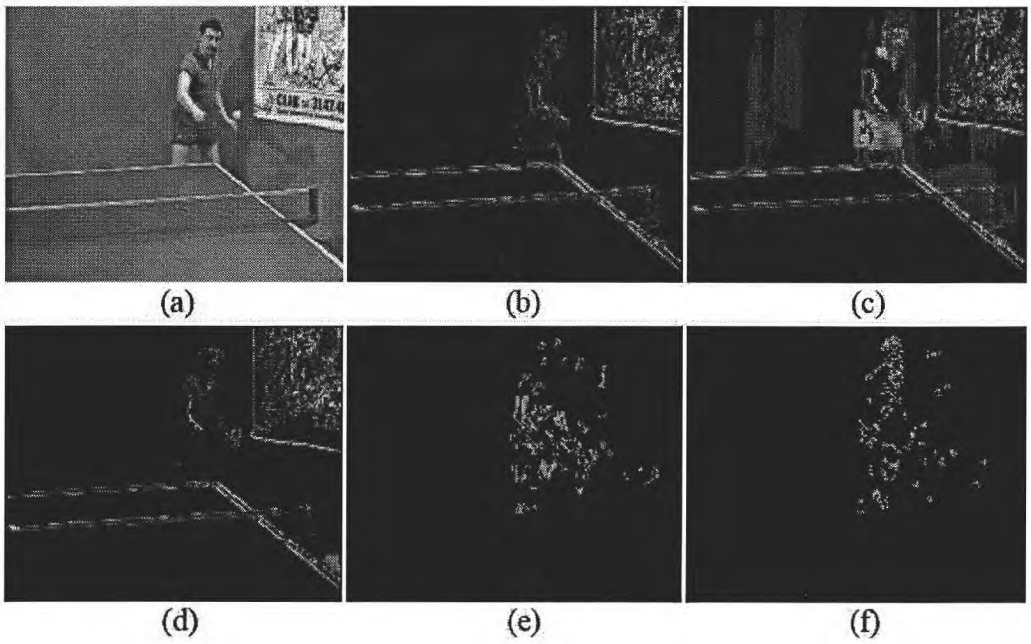


Figure 2.11: Feature maps. (a) original video frame, (b) intensity, (c) red-green colour, (d) blue-yellow colour, (e) x-motion, and (f) y-motion feature maps. [CCKW05]

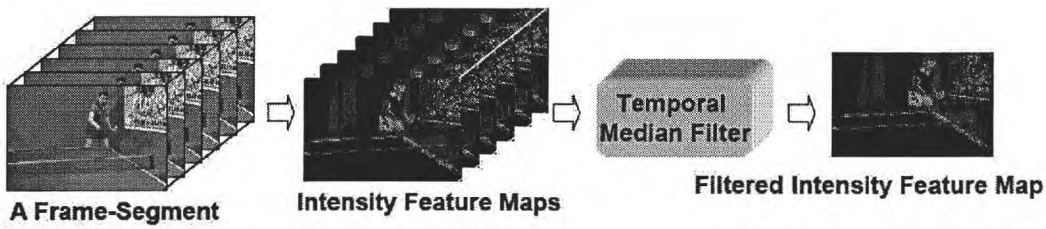


Figure 2.12: The procedure of generating the filtered intensity feature map [CCKW05]

The spatiotemporal saliency map is a weighted sum of the filtered feature maps. Cheng *et al.* specifically chose *expert-produced* videos to take advantage of the different types of camera motions such as zoom, left-pan, right-pan etc. Camera motions in *expert-produced* video reveal what and where the video-maker wants the viewers to see. Weights for filtered feature maps are determined according to the type of camera motion. For example, when camera panning occurs, the horizontal motion should be emphasized. The implication

is that the weight of the filtered x-motion feature map is large as shown in Table 2.3.

	Intensity	RG colour	BY colour	X-motion	Y-motion
Zoom	0.2	0.2	0.2	0.2	0.2
Left/Right-Pan	0.05	0.05	0.05	0.75	0.1
Static	0.15	0.075	0.075	0.35	0.35

Table 2.3: Weights for filtered feature maps under different camera motion types [CCKW05]

Spatiotemporal Saliency model by Zhai *et al.* [ZS06]

Zhai *et al.* [ZS06] proposed to generate both the spatial saliency map and the temporal saliency map independently, which are then combined to form the final spatiotemporal saliency map. The work flow of their saliency algorithm is shown in Figure 2.13.

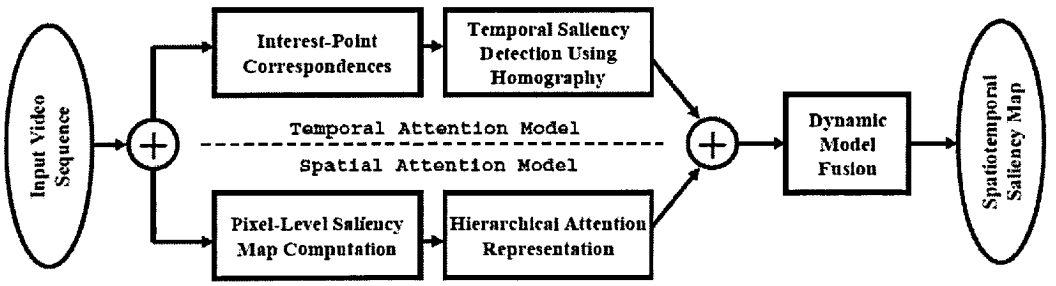


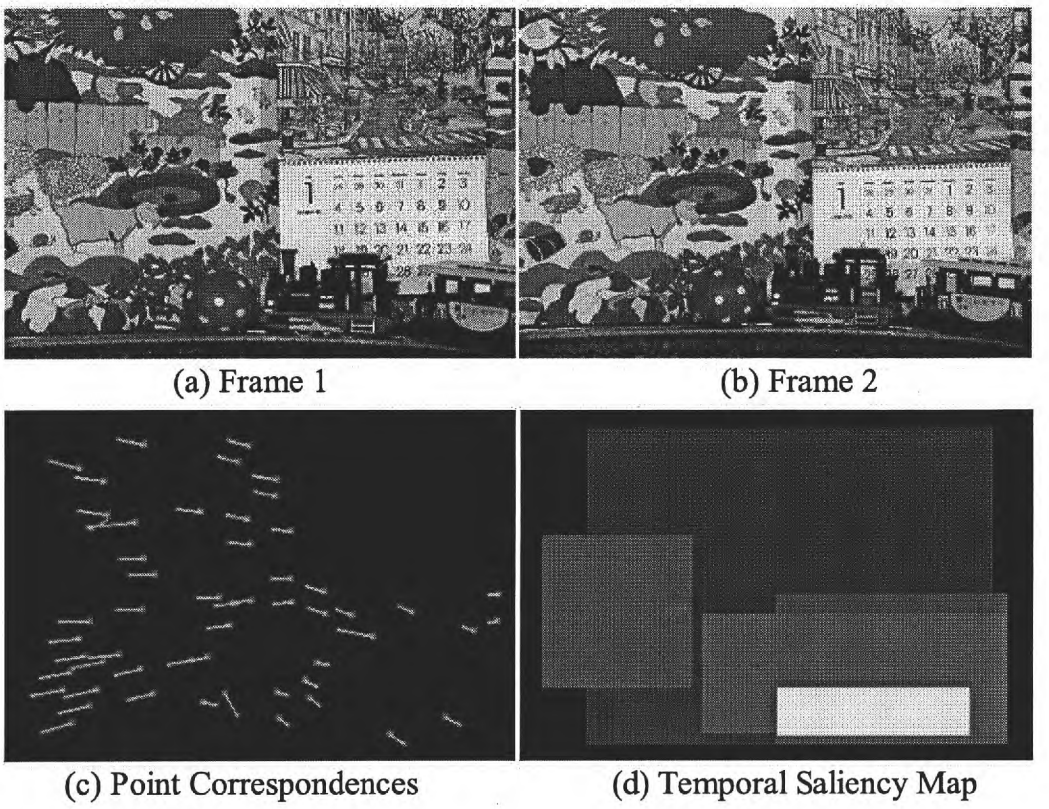
Figure 2.13: Work flow of the spatiotemporal saliency detection algorithm proposed by Zhai *et al.* [ZS06]

The spatial saliency map is generated using the colour histogram of one frame in the video. The authors argued that higher frequency of a colour level in the histogram indicates repeating colour information in the frame, and therefore, it is relatively un-salient. Firstly, for each pixel in the frame, its

colour level is compared with the colour histogram of the frame. The comparisons between colour levels are carried out by looking up a pre-computed distance map. A pixel-level saliency map is thus obtained. A hierarchical spatial attention representation is established to grow the salient points to salient regions. More specifically, the points with local maxima saliency values from the pixel-level saliency map are found. These points become the seed and salient regions grow from these seeds. The growing criteria are designed in such a way, that the salient regions continue to grow if and only if both the inner sides and outer sides of the region have high salient values. The growing stops at the boundary between the high value regions representing the interesting objects and the low value regions for the background. Rectangular attended regions are finally achieved.

The temporal saliency map is computed based on the planar motions between two consecutive video frames. Figure 2.14 illustrates the construction of such a temporal saliency map. Given two consecutive video frames such as in Figure 2.14(a) and Figure 2.14(b), feature points are firstly detected. Correspondences are established between the feature points using the Scale Invariance Feature Transformation (SIFT) [Low04] as in Figure 2.14(c). By applying RANSAC [FB81] on the point correspondences, homographies are obtained, one for each feature point. The homography describes the geometric transformation of a feature point between the consecutive frames. RANSAC associates feature points with rectangular motion segments according to their homographies. Thus, each feature point becomes a member of a rectangular

motion segment. The motion differences between the motion segment and its associated feature points are summed to represent the saliency value of the motion segment. To compensate the non-uniformity of the spatial distribution of interest-points, spanning areas of motion segments are taken into account and the temporal saliency map is computed as in Figure 2.14(d).



(a) Frame 1 (b) Frame 2

(c) Point Correspondences (d) Temporal Saliency Map

Figure 2.14: The construction of a temporal saliency map proposed by Zhai *et al.* [ZS06]

The spatial saliency map and the temporal saliency map are fused to produce the overall spatiotemporal saliency map. The weights are determined by the variance-like measure of the temporal saliency statistics. Higher weights are assigned to the temporal model if large motion contrast is present in the temporal saliency map. Otherwise, higher weights are assigned to the spatial

model if less motion exists.

2.4 Summary

This chapter describes three aspects of human visual system: the retina, the multi-resolution vision, and the visual attention model. The importance of the visual attention model is highlighted. The main characteristic of the visual attention is that it is affected by both the bottom-up and the top-down process. The bottom-up process is stimuli-driven and the top-down process requires *a priori* knowledge. The biological attention model has inspired a wide range of saliency detection algorithms. This chapter described the spatial saliency detection algorithms proposed by Itti *et al.* [IKN98] and Ma *et al.* [MZ03] respectively. These algorithms have been applied to a large number of images and satisfactory results have been achieved. However, spatial saliency detection algorithms are only applicable to static images. To detect saliency in video sequences, temporal information needs to be taken into account. Spatiotemporal saliency detection algorithms proposed by Cheng *et al.* [CCKW05] and Zhai *et al.* [ZS06] were described in this chapter. Moreover, there is a lot of research done by computer graphics community on visual attention [PS02, HYG01, KL99]. A summary of the saliency detection algorithms that appeared in this chapter is shown in Table 2.4.

All the saliency algorithms mentioned in this chapter undergo the bottom-up process. The benefit of this is that the algorithms are generic and au-

First author	Spatial Saliency Detection	Temporal Saliency Detection	Input data type
Itti [IKN98]	Intensity contrast, Colour contrast, Orientation contrast	N/A	One static image
Ma [MZ03]	Colour Contrast	N/A	One static image
Kadir [KB01]	Entropy of the grey-value distribution of a local region	N/A	One static image
Oliva [OTCH03]	Global distribution of low-level spatial features	N/A	One static image
Laptev [LL03]	Local maximum variation in both space and time		Video segment
Zhong [ZSV04]	Classify video segments into prototypes and compare each video segment with obtained prototypes		Entire video sequence
Cheng [CCKW05]	Intensity contrast, red-green colour contrast, blue-yellow colour contrast	x-motion, y-motion, temporal median filter	Video segment
Zhai [ZS06]	Colour histogram	Clustering motion of feature point into motion segment	Two consecutive frames in video

Table 2.4: Summary of the saliency detection algorithms

tonomous without the need for user intervention. However, the top-down process is equally important. How *a priori* information can be incorporated from a top-down process into these algorithms is unclear. Motion holds significant amount of information in video sequences. The temporal saliency detection framework proposed in this thesis define saliency as the following: "If the motion of an entity is NOT predictable, it is salient". A certain assumptions must be made on the motion model to measure predictability. Such an assumption is precisely the *a priori* knowledge the top-down process requires. Our tempo-

ral saliency detection model successfully incorporates the top-down process to the bottom-up process. More details will be given in the following chapters.

Chapter 3

The Proposed New Framework for Spatiotemporal Saliency Detection

This chapter presents an adaptive spatiotemporal saliency detection framework, which is an original contribution [LBC06a]. The framework incorporates spatial feature detection, feature tracking, and motion prediction in order to generate a spatiotemporal saliency map. The details on the interactions between these processes would be elaborated later in the chapter. The purpose of saliency detection is that more complex and high level scene analysis can focus on the most salient regions detected by the framework. Experimental results demonstrated its ability and robustness to produce saliency responses to motion pop-up phenomena that are in line with human responses. Furthermore, the framework is adaptive so that it can handle motions of a wide range

of speed.

3.1 The framework overview

We begin the description of the framework with introducing the underlying idea that governs the saliency detection. The idea is that it uses the unpredictability in the motion of spatial features as a measure of spatiotemporal saliency. In other words, when the motion path of a spatial feature in the image is predictable under a specific model, the feature is classified as non-salient, otherwise it is classified as salient. The above definition of spatiotemporal saliency distinguishes this framework from the previous approaches in the literature [LL03, ZSV04, CCKW05, ZS06], since it is based on the prediction of the feature's motion. As a result, this approach can be thought of as working at a higher conceptual level than previous approaches since statistical inference, a high level process, would be taken into account. Based on the above new definition of spatiotemporal saliency, a generic framework is conceived that can be built up from numerous established techniques. It is an unsupervised process due to its low-level nature and it operates without any knowledge of the shape or size of the object(s) that may be of interest. An overview of the framework is shown in Figure 3.1.

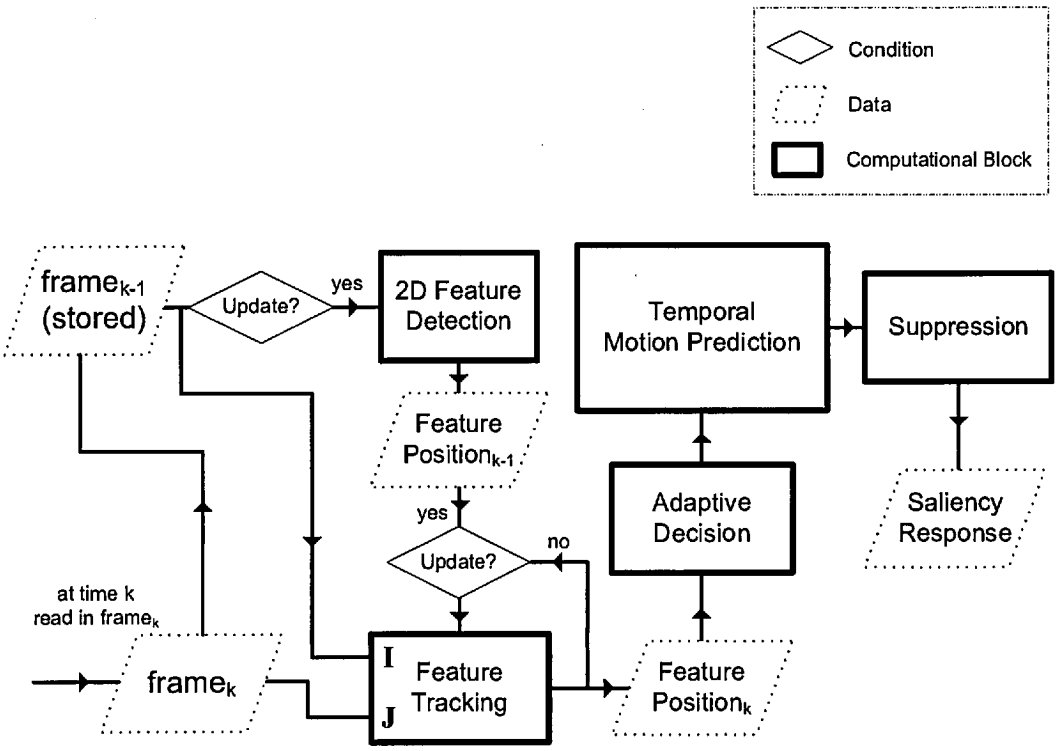


Figure 3.1: Overview of the proposed spatiotemporal saliency framework

2D Feature Detection module

2D features refer to spatial features within a video frame. They are detected by the **2D Feature Detection** module as in Figure 3.1. The detection is carried out in the initialization step of the algorithm. As time elapses, some existing 2D features may exit the field of view of the camera hence the scene, while some new 2D feature may enter the scene. Therefore, it is necessary to refresh the 2D features every n frames to reflect the up-to-date presence of 2D features, where n is set according to a prior knowledge of the scene's activity.

These 2D features are the cues to be both tracked and predicted between consecutive frames and the discrepancies between the tracking and predicting results contribute to the temporal saliency map. Features can be obtained

by any point-based spatial saliency algorithm. The justification of involving a spatial saliency algorithm is that in order for the features to be qualified as spatiotemporally salient, they must first be spatially salient. From those features that satisfy the spatial criterion, the temporally salient ones are screened out as spatiotemporal salient points.

Feature Tracking module

The 2D features are then tracked by the **Feature Tracking** module. Let **u** be a feature in frame **I**. The purpose of feature tracking is to find its corresponding location in frame **J**. Therefore, by feeding in two consecutive frames to the feature trackers, the relative displacement of each feature from one frame to the next can be found.

Adaptive Decision module

A number of previous locations of 2D features are stored in the memory, from which the average speed of each 2D feature in the past few frames can be estimated. A very small average speed implies that the current frame rate of the camera is too high to observe any significant movement of the 2D feature or that the projected speed of the object to the image plane is small. This is problematic because it is difficult to determine unpredictability out of a sequence of insignificant movements. To rectify this, the coordinates of individual 2D features are ignored by the **Adaptive Decision** module according to the average speed of the corresponding 2D feature.

Temporal Motion Prediction module

The coordinates of the 2D features are then fed into the **Temporal Motion Prediction** module to determine how predictable their motions are. Each individual 2D feature is assigned a predictor and treated independently from the rest of the 2D features. The number of predictors needed is therefore equal to the number of 2D features. The **Temporal Motion Prediction** module is formed from an array of **Predictors** as shown in Figure 3.2. Each **Predictor** computes the motion predictability of its corresponding 2D feature. The predictability measure consists of two parts: the long-term predictability and the short-term predictability. Thus, each **Predictor** performs both the long-term and short-term predictability computations as shown in Figure 3.3.

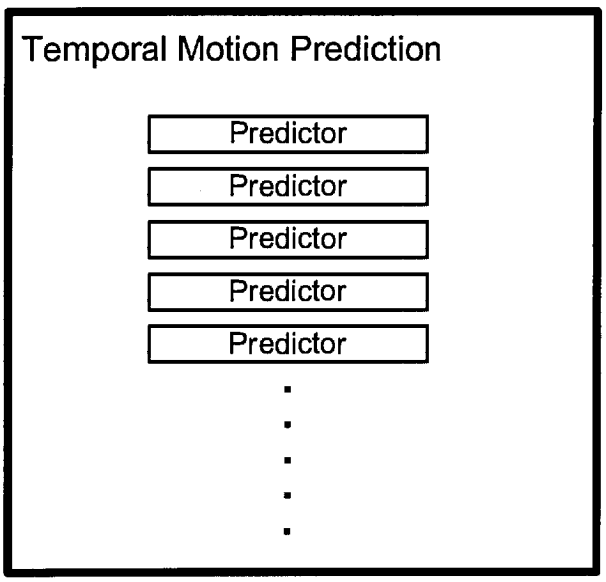


Figure 3.2: The **Temporal Motion Prediction** is formed from an array of **Predictors**

The long-term predictability determines whether the motion of a 2D feature is periodic in a relatively long interval of time. A periodic motion of a 2D

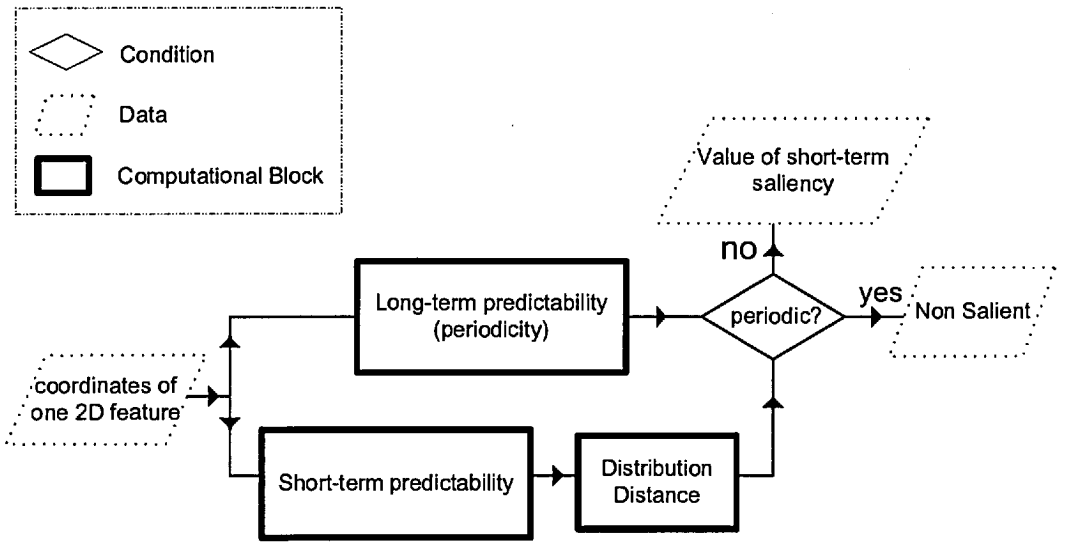


Figure 3.3: The long-term predictability and short-term predictability computation in a **Predictor**

feature implies that the 2D feature follows the same motion path over and over again, thus its motion is predictable.

Short-term predictability refers to predicting the coordinates of a 2D feature in the next frame given a specific motion model. The coordinates are the measurement inputs of the short-term predictors. Each short-term predictor *updates* its model on receiving new measurements and *predicts* the coordinates of the feature in the upcoming frame according to the specific motion model. It must be noticed that both the *updated* and *predicted* coordinates from the short-term predictor are probabilistic rather than deterministic. They can be represented by a set of coordinates each with different probabilities. At time k, each feature has a set of *predicted* coordinates with different probabilities computed at time $k-1$ by its short-term predictor. Each feature also has a set of *updated* coordinates each with different probabilities by updating the short-term predictor with the new measurement at time k. The distance between

the two probability density functions of the *predicted* and *updated* coordinates determines the short-term predictability of the motion behaviour of the 2D feature.

If the motion of a 2D feature is found by the long-term predictor as periodic, the result from the short-term predictor is ignored and the 2D feature is temporally non-salient. If the motion of a 2D feature is not periodic, then its temporal saliency value is computed from the short-term predictor and the distribution distance measure. This interaction between short-term and long-term predictability is shown in Figure 3.3.

Suppression module

Several temporally salient 2D features that are close to each other should make a bigger impact on the spatiotemporal saliency responses than one single temporally salient 2D feature relatively distant away from other 2D features. This is achieved by the **Suppression** module. The module promotes clusters of salient features which are excited by one another whilst playing down singular features whose non-salient neighbors inhibit its saliency response. The result from the **Suppression** module is the spatiotemporal saliency response.

3.2 Realization of the framework

Given the overview of the framework structure in Section 3.1, algorithms that realize the functions of the modules have to be applied. The description on the

algorithms is kept concise and more details will be provided in later chapters. In this section, the realization of the framework is based on the assumption that features with linear motions are classified as non-salient, and features with more complicated motions are classified as salient. It should be noted that this assumption is made to provide an intuitive example of one possible realization of the framework. However, the framework itself is not restricted to this assumptions.

The Shi and Tomasi algorithm [ST94] has been adopted for the **2D Feature Detection** module. The choice of this algorithm lies in the fact that corners can be detected with robustness and that it has been developed to maximize the quality of tracking, which is important for the subsequent steps of the spatiotemporal saliency framework. Each pixel is associated with a spatial gradient matrix obtained from its neighbourhood. The eigenvalues of this matrix are computed and the smallest of them is assigned to the pixel as a measure of its cornerness. Pixels with cornerness less than a predefined threshold are rejected. Finally, it ensures that all the detected corners are far enough apart from each other by removing corners that are close to strong corners, i.e. corners of high minimum eigenvalues.

The pyramidal implementation of the iterative Lucas-Kanade feature tracker [Bou00] is adopted as the **Feature Tracking** module in this particular realization of the saliency framework. This algorithm takes advantage of the pyramidal image data structure illustrated in Figure 2.7 previously mentioned in Section 2.3.1. The goal of a feature tracker is to locate the corresponding fea-

tures between frames. A feature pixel is tracked in the lowest-resolution level L_m within the pyramid and the result is propagated to the next finer level until the original resolution L_0 is reached, where $m+1$ is the total number of levels in the pyramid.

The **Adaptive Decision** module stores the coordinates of the 2D features for the past few frames and estimates their average speed. Depending on the average speed, it then decimates coordinates of the 2D features with slow motion.

The **Temporal Motion Prediction** module consists of an array of long-term predictors and an array of short-term predictors. The long-term predictor is based on autocorrelation and power spectrum [Pri82], which is useful for finding repeated patterns in a signal. A set of previous coordinates of each 2D feature is autocorrelated to a larger set of its historic coordinates. Then a threshold is applied to determine whether the motion of the 2D feature is periodic. The algorithm used for short-term prediction is the Kalman filter [Kal60, KB61], which is a recursive filter that can update and predict the states of the system from noisy measurements. The Kalman filter associates a Gaussian distribution for each state. The Kalman filter is a good instrument in predicting *linear* motion. Based on the non-saliency assumption of *linearity*, the motion that can be predicted by the Kalman filter is non-salient. Otherwise, the motion is salient. The Bhattacharyya distance [Kai67] is used as the distribution distance to quantify saliency from the probability distribution constructed by the short-term predictor. It is a symmetric measure of the

distance between two probability distributions.

The **Suppression** module post-processes the preliminary saliency map rendered by the **Temporal Motion Prediction** module. The features are subject to a threshold process and those whose distribution distance values being less than a threshold are disregarded. The neighboring pixels of each feature are then investigated. A feature only remains salient and advances into the final saliency map if there are sufficient number of other features surviving the threshold process within the neighborhood, otherwise the feature is disqualified. Under this suppression scheme, only clusters of salient features remain and outliers are removed. The suppressor has the effect of disentangling noise from genuine salient movement by relying on multiple saliency responses.

3.3 Performance Evaluation

Experiments were conducted to demonstrate the viability and robustness of the spatiotemporal framework using the realization described in Section 3.2. In all the experiments, the maximum number of features was limited to 1000. The suppressor threshold was set to 2. Each feature had to have at least one neighboring feature within a circle of three pixel radius in order to be qualified as a salient feature.

3.3.1 Adaptive saliency responses

The experiment illustrated in Figure 3.4 shows how the movement of the car triggers the adaptive spatiotemporal saliency responses. For presentation purpose, the centers of the circles are the coordinates of the 2D features that give rise to the saliency responses and the radiuses of the circles indicate the magnitude of saliency.

From frame **F1** to **F3**, the car moves at a constant velocity, therefore no saliency was detected by the model. In order to adapt to the low speed of the car, feature coordinates in frame **F2** are not used. Please notice that it is the feature coordinates that are decimated as opposed to the frame itself. If there were two cars, a fast moving car and a slow one, the coordinates of feature on the fast moving car would not be decimated whereas the coordinates of the features on the slow one would be as in this experiment. This is very beneficial because the framework is capable of analyzing dynamic scenes where motions of objects can differ in speed.

The car changes velocity from frame **F3** and the speed is still low so the feature coordinates in frame **F4** are decimated. The salient responses are picked up by 2D features automatically placed on the car and they are shown as circles in frame **F5**. The absolute velocity change of the car is small, so without the decimation, the small change would be undetected. However, relative to the slow motion of the car, the velocity change is significant and this adaptive mechanism allows us to capture it.

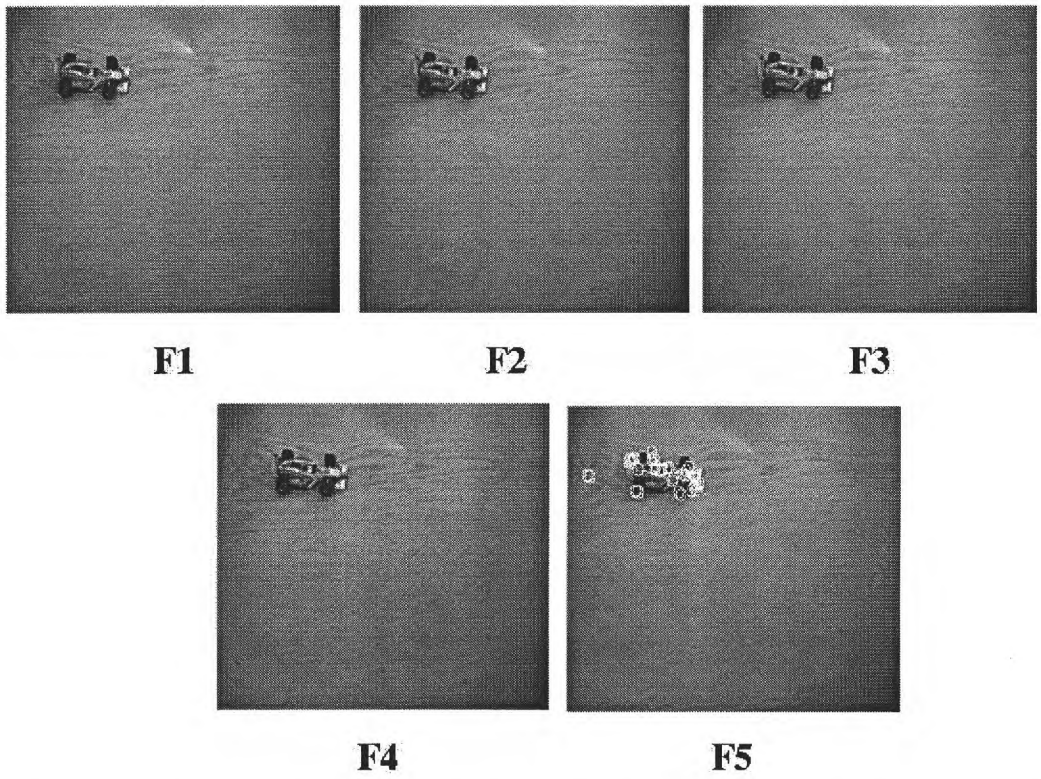


Figure 3.4: Spatiotemporal saliency results: The car moves slowly at a constant velocity before frame **F3**, to adapt to the slow motion, feature coordinates in one frame are ignored out of every two frames. Frames **F2** and **F4** are disregarded by the predictor stage. The car changes velocity after frame **F3**. The salient responses are picked up by 2D features automatically placed on the car and they are shown as circles in frame **F5**.

This experiment illustrates that the proposed spatiotemporal framework detects motion pop-up phenomena successfully.

3.3.2 Complex scene saliency analysis

This experiment demonstrates the robustness of the spatiotemporal saliency model in a complex scene. In Figure 3.5, a massive number of fish flock towards one spot where food is supplied. In the region around that spot, the fish movements are especially vigourous and irregular, hence these points are the

most unpredictable in the scene.

Figure 3.5 (a) shows spatiotemporal saliency detection results at the beginning of the video clip and only the very first two frames of the video are known to the framework. The saliency model suggests a majority of 2D features as salient due to the lack of sufficient temporal information. Hence, there are salient responses all over the place.

Figure 3.5 (b) shows the saliency detection results at a later time. Although only two consecutive frames are considered in the saliency detection process, the framework has implicitly picked up earlier temporal information through parameter correction of the Kalman filters. Evidently in Figure 3.5 (b), the saliency model indeed emphasizes the region around the feeding spot with large saliency responses and smaller responses in other regions.

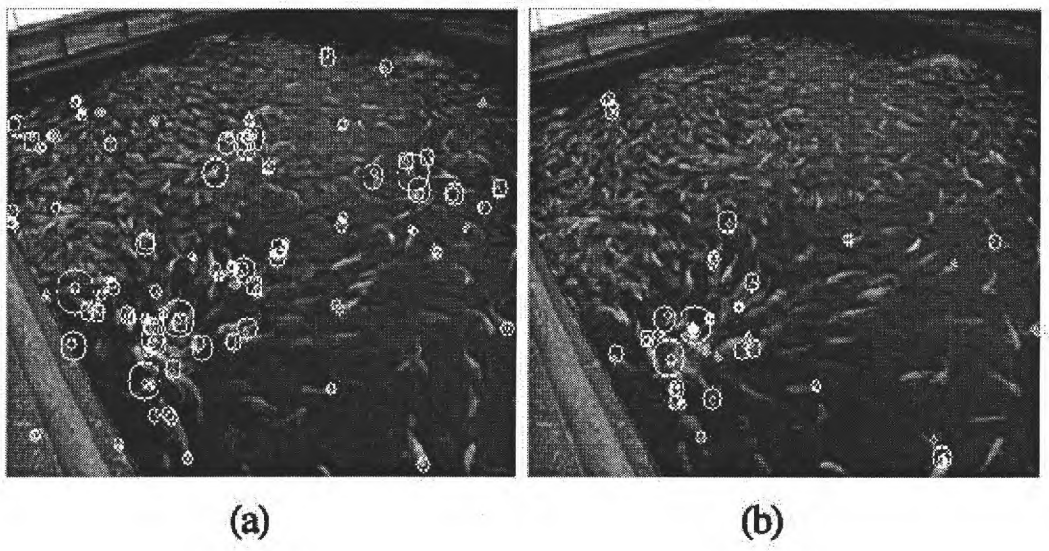


Figure 3.5: Complex scene: a massive number of fish moving towards fish feed. (a) saliency result from the very first two frames only; (b) saliency result at a later time.

3.4 Computational complexity of the framework

The proposed saliency framework consists of a number of interacting modules as described in Section 3.1. In Section 3.2, these modules are realized with various algorithms. The matching of the modules with the corresponding algorithm is summarized in Table 3.1.

Module		Algorithm
2D Feature Detection		Shi and Tomasi algorithm [ST94]
Feature Tracking		Pyramidal implementation of the iterative Lucas-Kanade feature tracker [Bou00]
Adaptive Decision		Average speed of 2D feature
Temporal Motion Prediction	Long-term predictor	Autocorrelation [Pri82] and power spectrum
	Short-term predictor	Kalman filter [Kal60, KB61]
	Distribution Distance	Bhattacharyya distance [Kai67]
Suppression		Thresholding and grouping

Table 3.1: The matching of the module with the algorithm

To estimate the computational complexity of the algorithms, a C++ performance profiling tool, AQtime [AQt], is applied to the C++ implementation of the framework. Computation time is used as the complexity metric, which implies that a more complex block requires longer time to process. In a typical application, the resolution of the video sequence is 320×240 and 1000 2D features are used. Running on a Pentium 4 PC with 2.4GHz CPU and 1GB of RAM, the computation time for each frame is 0.083 second. The computation time of the entire framework is broken down into time spent executing each computational block. Such execution time break down is shown in Figure 3.6.

The pie chart estimates the relative computational complexity of algorithms within the framework.

In the pie chart, Shi and Tomasi algorithm, Kalman filter, and pyramidal Lucas-Kanade algorithm occupy a significant portion each. They would be the potential challenges in the hardware implementation of the framework when a real-time performance is required. The execution of Shi and Tomasi algorithm is predominately eigenvalue computation. In order to accelerate the proposed framework, an efficient architecture for eigenvalue computation is proposed in Chapter 4. Moreover, an efficient mapping of Kalman Filter into hardware is also proposed in Chapter 5. Both works demonstrate significant performance improvement and they are not only useful and pertinent to our saliency framework, but also contributive to any other areas that require eigenvalue computation or Kalman filter application. Pyramidal Lucas-Kanade algorithm is not implemented because the algorithm is too specific and many hardware approaches achieving good results can be found in the literature [DRP+06, FDC98] and no potential modification can be spotted to greatly outperform them.

3.5 Summary

This chapter presented an unsupervised adaptive spatiotemporal saliency framework. Our original contribution is the detection of temporal saliency and the manner in which the temporal saliency is built upon the spatial saliency to

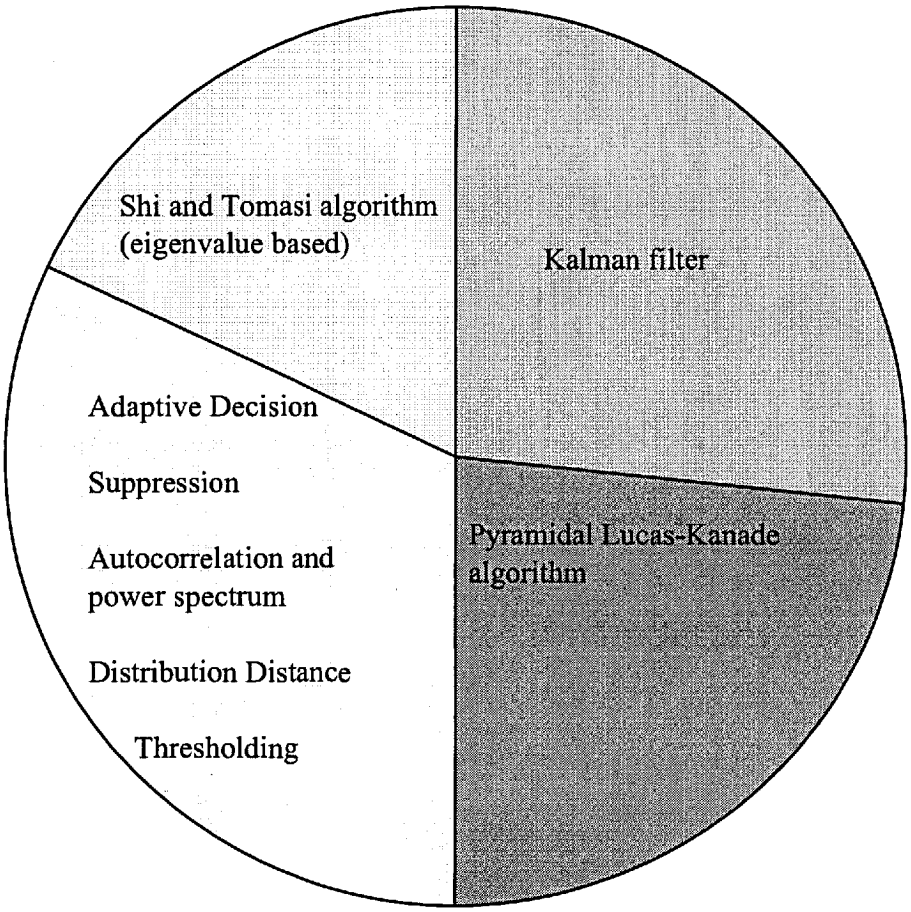


Figure 3.6: Estimation of relative computational complexity of algorithms within the framework

meaningfully represent spatiotemporal saliency. Experimental results showed that the model using this framework has the capability of distinguishing between *predictable* and *unpredictable* motions hence generating a *spatiotemporal saliency* without the involvement of object detection and segmentation even in a complex scene. The framework was shown to be adaptive to the speed of 2D features in order to increase the range of speed it can cope with. Moreover, the described framework has the flexibility to be tuned to specific motion patterns depending on the requirements. Under a linear motion assumption, the Kalman filter is employed as the short-term predictor. The framework

is totally flexible to target other motion models and consequently estimates temporal saliency maps to different models.

Our framework is in line with the interaction between bottom-up processes and top-down processes in the human visual attention model mentioned in Section 2.3. The detection and tracking of spatial 2D features belong to the stimulus bottom-up process. The computation of predictability of the features is a way to analyze behavioural relevance in the scene and hence belongs to the top-down process.

In order to facilitate comparison with saliency algorithms reviewed in Section 2.3.1 and Section 2.3.2, Table 3.2 summarizes these algorithms in the literature as well as our proposed framework. From the storage requirement point of view, all the spatial saliency algorithms [IKN98, MZ03, KB01, OTCH03] need to store one static image whereas the spatiotemporal saliency algorithms can be divided into three categories. In the first category, the algorithm proposed by Zhong *et al.* [ZSV04] requires access to the entire video sequence and hence it is not applicable for real-time streaming video input. The second category includes the algorithms proposed by Laptev *et al.* [LL03] and Cheng *et al.* [CCKW05]. They require storing a video segment before saliency detection can be executed. This not only needs considerable amount of storage, but also unavoidably incurs latency. The third category needs only two frames from the video. It is therefore efficient in terms of storage cost. Zhai *et al.* [ZS06] algorithm and our proposed framework belong to the third category. In our proposed framework, only the current frame and the previous frame

have to be stored for 2D feature tracking. The remaining storage is linearly proportional to the number of features, which is much less than the number of pixels in a frame. Nevertheless, using the appropriate predictor, statistical distributions of the motions of 2D features can be recursively built over the past frames. This gives our proposed framework an extra edge over Zhai *et al.* algorithm because information from earlier frames is implicitly taken into account. Effectively, in the proposed framework, the spatiotemporal saliency map has temporal support of more than just two frames due to the encoding provided by the Kalman filter.

Another advantage of our proposed spatiotemporal saliency detection framework is flexibility. Although the realization of the framework in Section 3.2 uses Shi and Tomasi algorithm to detect spatial 2D features and employs Kalman filter as the short-term predictor, the framework itself is not restricted by these specific algorithms. In future work, it is feasible to use established spatial saliency algorithm to detect 2D features. It is also possible to use particle filters [IB98] as short-term predictor so that a more complicated motion can be modeled as *a priori* knowledge, the predictable motion.

As mentioned before, the description on the modules in this chapter is intentionally kept brief to emphasize the presentation of the entire spatiotemporal saliency framework itself. In Chapter 4, the **2D Feature Detection** module, the **Feature Tracking** module, and the **Adaptive Decision** module are described in more detail. In Chapter 5, the **Temporal Motion Prediction** module and **Suppression** module are elaborated.

First author	Spatial Saliency Detection	Temporal Saliency Detection	Input data type
Itti [IKN98]	Intensity contrast, Colour contrast, Orientation contrast	N/A	One static image
Ma [MZ03]	Colour Contrast	N/A	One static image
Kadir [KB01]	Entropy of the grey-value distribution of a local region	N/A	One static image
Oliva [OTCH03]	Global distribution of low-level spatial features	N/A	One static image
Laptev [LL03]	Local maximum variation in both space and time		Video segment
Zhong [ZSV04]	Classify video segments into prototypes and compare each video segment with obtained prototypes		Entire video sequence
Cheng [CCKW05]	Intensity contrast, red-green colour contrast, blue-yellow colour contrast	x-motion, y-motion, temporal median filter	Video segment
Zhai [ZS06]	Colour histogram	Clustering motion of feature point into motion segment	Two consecutive frames in video
Our proposed framework	Eigenvalue-based feature detector, feature tracker	Kalman filter, autocorrelation, power spectrum, predictability	Two consecutive frames in video

Table 3.2: Summary of the saliency detection algorithms including our proposed framework

Chapter 4

Software and hardware concerns on the 2D feature

The first section of this chapter describes in detail the software algorithms that handle 2D feature detection and tracking. The used 2D feature detection algorithm is based on eigenvalue computation, which is singled out among the software algorithms as the most challenging part and therefore would benefit from hardware acceleration. The second section of this chapter presents our proposed efficient architecture for eigenvalue computation, which constitutes original contribution [LBC+06b].

4.1 Software algorithms for the modules

This section provides the details on the algorithms that realize the **2D Feature Detection** module, the **Feature Tracking** module, and the **Adaptive**

Decision module. In Figure 4.1, these modules are highlighted in grey.

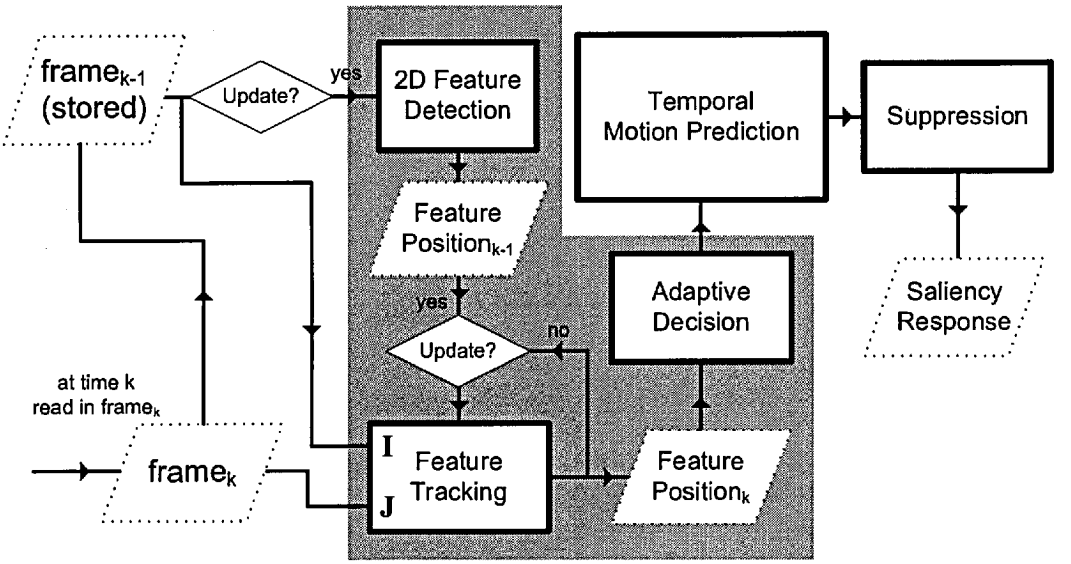


Figure 4.1: Modules in the framework relevant to Chapter 4 are highlighted in grey

4.1.1 Algorithm for 2D Feature Detection module

The Shi and Tomasi algorithm [ST94] has been adopted for the **2D Feature Detection** module. With respect to the entire framework, three important characteristics of the 2D features are desirable. First, the 2D features must be suitable for tracking purposes for the subsequent module in the saliency framework. The Shi and Tomasi algorithm is developed to detect features which maximize the quality of tracking and hence it fits this purpose. Second, it is beneficial that the 2D feature is spatially salient. Because by combining the spatial saliency with the temporal saliency computed by the **Temporal Motion Prediction** module, the framework is capable to detect spatiotemporal saliency. This makes the spatial saliency algorithms described in Section

2.3.1 good candidates. For example, the saliency detection algorithm proposed by Itti *et al.* [IKN98] is appropriate for the **2D Feature Detection** module from the spatial saliency point of view. However, the computational complexity of **2D Feature Detection** module should be moderate, which is the third desirable characteristic. It is not affordable to allocate all the computational power to this single module. In terms of computational cost, Itti *et al.*'s algorithm too expensive to achieve real-time performance for the **2D Feature Detection** module. Furthermore, a salient point detected by a state-of-the-art spatial saliency algorithm is not necessarily a good feature to track, which is of paramount importance for the rest of the framework. Having considered all three desirable characteristics, the Shi and Tomasi algorithm is chosen.

Feature points detected by the Shi and Tomasi algorithm are optimal for tracking because the algorithm itself is designed based on how the tracker works. Given an image $\mathbf{I}$, $\mathbf{u}$ is a pixel with coordinates $[u_x, u_y]^T$, and $\mathbf{w}$ is the neighbourhood $[u_x - w_x, u_x + w_x] \times [u_y - w_y, u_y + w_y]$ around pixel $\mathbf{u}$. The spatial gradient matrix $\mathbf{G_u}$ for pixel $\mathbf{u}$ is defined in (4.1).

$$\mathbf{G_u} = \sum_{x=u_x-w_x}^{u_x+w_x} \sum_{y=u_y-w_y}^{u_y+w_y} \begin{bmatrix} \mathbf{I}_x^2(x,y) & \mathbf{I}_x(x,y)\mathbf{I}_y(x,y) \\ \mathbf{I}_x(x,y)\mathbf{I}_y(x,y) & \mathbf{I}_y^2(x,y) \end{bmatrix} \tag{4.1}$$

where

$$\mathbf{I}_x(x,y) = \frac{\mathbf{I}(x+1,y) - \mathbf{I}(x-1,y)}{2}$$
$$\mathbf{I}_y(x,y) = \frac{\mathbf{I}(x,y+1) - \mathbf{I}(x,y-1)}{2}$$

The eigenvalues of matrix $\mathbf{G}$ are the keys to determine how well the pixel $\mathbf{u}$ can be tracked. Shi and Tomasi prove that if the minimum eigenvalue of $\mathbf{G}$ is large enough, then $\mathbf{u}$ can be corners, salt-and-pepper textures, or any other pattern that can be tracked reliably.

Based on the eigenvalue criterion, the first step of Shi and Tomasi algorithm is to find the minimum eigenvalue of spatial gradient matrix $\mathbf{G}$ defined in (4.1) for each pixel in the image. Each pixel is associated with one corresponding eigenvalue. Pixels with eigenvalues above the predefined threshold remain and the rest are rejected. The pixels with eigenvalues that are local maxima in their respective 3×3 neighbourhood are selected as features. Finally, the algorithm ensures that all the features are far enough apart from each other by removing features that are close to a feature with a higher eigenvalue. Figure 4.2 shows the input image to Shi and Tomasi algorithm and the detected features are represented by crosses in Figure 4.3.

Figure 4.2: Input image for Shi and Tomasi algorithm

4.1.2 Algorithm for Feature Tracking module

Given a feature $\mathbf{u}$ with coordinates $[u_x, u_y]^T$ in the first image $\mathbf{I}$, the goal of feature tracking is to find the coordinates $\mathbf{s} = \mathbf{u} + \mathbf{d} = [u_x + d_x, u_y + d_y]^T$ on the second image $\mathbf{J}$ such that $\mathbf{I}(\mathbf{u})$ and $\mathbf{J}(\mathbf{v})$ are *similar*. $\mathbf{d}$ is called the displacement vector. *Similarity* is defined in a neighbourhood sense. All the pixels in the surrounding region around the feature $\mathbf{u}$, known as the integration window, are taken into account and analyzed for the purpose of tracking $\mathbf{u}$. The integration window $\mathbf{w}$ is of size $(2w_x + 1) \times (2w_y + 1)$. To maximize *similarity* is to minimize the residual function ϵ defined in (4.2).

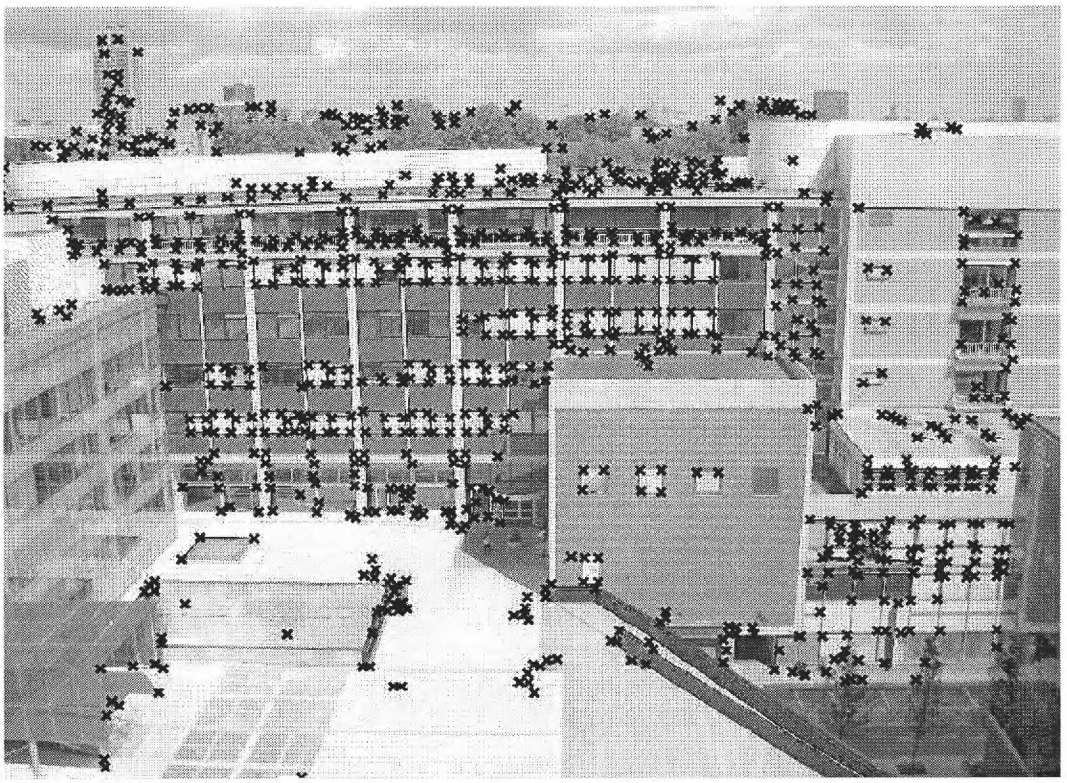


Figure 4.3: Features detected by Shi and Tomasi algorithm. Locations of the features are indicated by the crosses

$$\epsilon(\mathbf{d}) = \epsilon(d_x, d_y) = \sum_{x=u_x-w_x}^{u_x+w_x} \sum_{y=u_y-w_y}^{u_y+w_y} \Big(\mathbf{I}(x,y) - \mathbf{J}(x+d_x, y+d_y)\Big)^2 \qquad (4.2)$$

The criteria of a good feature tracker are accuracy and robustness. A small integration window is preferred to avoid smoothing out the details in order to achieve high accuracy. However, the robustness in handling large motion requires having a large integration window. The pyramidal Lucas-Kanade feature tracker [Bou00] facilitates a trade-off between robustness and accuracy because it takes advantage of the pyramidal image data structure. This is the reason that the pyramidal Lucas-Kanade feature tracker is chosen as the

Feature Tracking module. Recall from Section 2.3.1, the pyramidal image data structure is created by reproducing a copy of the image at progressively lower resolutions. A pyramid is illustrated in Figure 2.7 with the image at the lowest resolution at the top, and the highest (original) resolution at the bottom. The resolution varies by a factor of 2 between adjacent levels. The pyramidal Lucas-Kanade feature tracker tracks in the lowest-resolution level L_m within the pyramid and the result is propagated to the next finer level until the original resolution level L_0 is reached, where $m+1$ is the total number of levels in the pyramid.

At each level L, the corresponding coordinates of feature $\mathbf{u}^L$ in image $\mathbf{I}^L$ can be found by (4.3).

$$\mathbf{u}^L = [u_x^L, u_y^L] = \frac{\mathbf{u}^{L_0}}{2^L} \tag{4.3}$$

$\mathbf{u}^{L_0}$ represents the coordinates of feature $\mathbf{u}$ in the original resolution image $\mathbf{I}^{L_0}$, which lies at level 0 in the pyramid. The spatial gradient matrix $\mathbf{G}$ with respect to the integration window of $\mathbf{u}^L$ is computed by (4.4).

$$\mathbf{G}_{\mathbf{u}^L} = \sum_{x=u_x^L-w_x^L}^{u_x^L+w_x^L} \sum_{y=u_y^L-w_y^L}^{u_y^L+w_y^L} \begin{bmatrix} \mathbf{I}_x^2(x,y) & \mathbf{I}_x(x,y)\mathbf{I}_y(x,y) \\ \mathbf{I}_x(x,y)\mathbf{I}_y(x,y) & \mathbf{I}_y^2(x,y) \end{bmatrix} \tag{4.4}$$

where $\mathbf{I}_x(x,y)$ and $\mathbf{I}_y(x,y)$ are defined as before. Then, the feature tracking process at level L is carried out as follows.

Initialization:

$$\mathbf{g}^L = 2(\mathbf{g}^{L+1} + \mathbf{v}_K^{L+1}), \quad \text{with } \mathbf{g}^{L_m} = [0\ 0]^T \tag{4.5}$$

$$\mathbf{v}_0 = [0\ 0]^T \tag{4.6}$$

Lucas-Kanade iterations: for $k = 1$ to K, incrementing by 1.

$$\begin{aligned} \delta\mathbf{I}_k(x,y) &= \mathbf{I}(x,y) - \mathbf{J}(x + g_x^L + v_x^{k-1}, y + g_y^L + v_y^{k-1}) \\ \mathbf{b}_k &\equiv \sum_{x=u_x^L - w_x^L}^{u_x^L + w_x^L} \sum_{y=u_y^L - w_y^L}^{u_y^L + w_y^L} \begin{bmatrix} \mathbf{I}_x(x,y)\delta\mathbf{I}_k(x,y) \\ \mathbf{I}_y(x,y)\delta\mathbf{I}_k(x,y) \end{bmatrix} \\ \mathbf{v}_k &= \mathbf{v}_{k-1} + (\mathbf{G}_{\mathbf{u}^L})^{-1}\mathbf{b}_k \end{aligned} \tag{4.7}$$

In (4.5), $\mathbf{v}_K^{L+1}$ is the tracking result after all K iterations of Lucas-Kanade process at level $L+1$, which is the immediate previous level to the current level L. $\mathbf{g}$ is an accumulator of tracking results. The implication of (4.5) is that $\mathbf{g}^L$ contains the accumulative tracking result obtained from all levels that have been processed. $\mathbf{g}^L$ provides the initial "guess" for the Lucas-Kanade iteration at the current level L. At the end of the Lucas-Kanade iteration, $\mathbf{v}_K$ represents the displacement vector of the feature $\mathbf{u}$ from image $\mathbf{I}^L$ to image $\mathbf{J}^L$. $\mathbf{v}_K$ is then used to update the accumulator $\mathbf{g}$ for the next level of finer image.

After level 0 is reached, the displacement vector $\mathbf{d} = \mathbf{v}_K + \mathbf{g}^{L_0}$ at the original image resolution is found. Thus, the corresponding coordinates $\mathbf{s} = \mathbf{u} + \mathbf{d}$ in

the image **J** are obtained for the feature **u** in image **I**.

A demonstration of pyramidal Lucas-Kanade feature tracker is shown as the following. Figure 4.4(a) and Figure 4.4(b) show the first input images **I** and the second input image **J** for the feature tracker, respectively. Relative to Figure 4.4(a), the car in Figure 4.4(b) moves to the right while everything else remains static. 2D Features in Figure 4.4(a) are detected by Shi and Tomasi algorithm described in Section 4.1.1. These same features are tracked in Figure 4.4(b) using the pyramidal Lucas-Kanade feature tracker. The tracking result is shown in Figure 4.5, where the displacement vectors for the features are represented by arrows.

4.1.3 Algorithm for Adaptive Decision module

In the example shown in Figure 4.4 and Figure 4.5, the input to the feature tracker are two images. The displacement vectors **d** between image **I** and image **J** are computed by the feature tacker. The new coordinates **s** on image **J** of a 2D feature **u** on image **I** can be found by $\mathbf{s} = \mathbf{u}+\mathbf{d}$. When the feature tracker is applied to a video sequence, displacement vectors between consecutive frames are computed. Therefore, after a few frames, a set of displacement vectors are obtained for each 2D feature. The set of displacement vectors indicates how many pixels a 2D feature has moved between consecutive frames throughout the video sequence.

By storing a number of previous displacement vectors for each 2D feature, the average speed of each 2D feature in the past few frames can be estimated.

(a)

(b)

Figure 4.4: The input images for the pyramidal Lucas-Kanade feature tracker. (a) is the first image **I**; (b) is the second image **J**

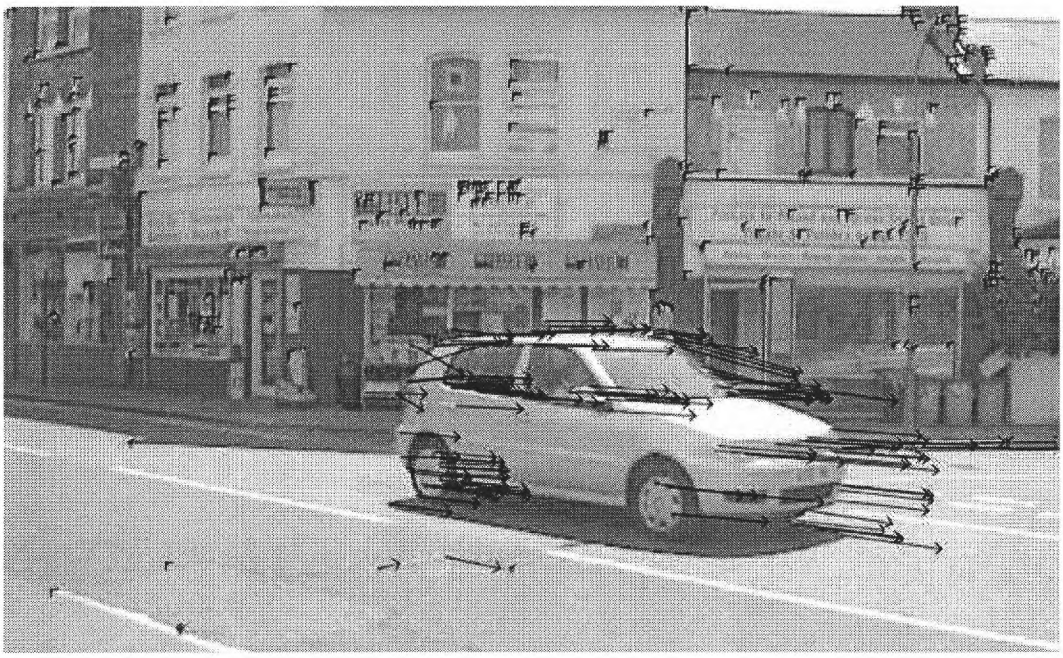


Figure 4.5: The output of the pyramidal Lucas-Kanade feature tracker. Arrows indicate displacement vector.

A very small average speed implies that the current frame rate of the camera is too high to observe any significant movement of the 2D feature or that the projected speed of the object to the image plane is small. This is problematic because it is difficult to determine unpredictability out of a sequence of insignificant movements since all these movements appear to be linear. The **Adaptive Decision** is introduced to rectify this problem.

The operation of the **Adaptive Decision** module is explained in Figure 4.6. The squares represent pixels on the image plane and the size of one pixel limits the precision that the feature tracker can locate the 2D features. The dot represents the actual location of one 2D feature projected on the image plane at a particular frame. It should be noted that the system can only detect the location of the feature at pixel level accuracy. It does not have sub-pixel

accuracy. The decimation process is based on the average displacement of the feature per frame. The numbers above the dot indicate the frame number. Each of the four rows shows that one 2D feature moves to the right, albeit at a different speed.

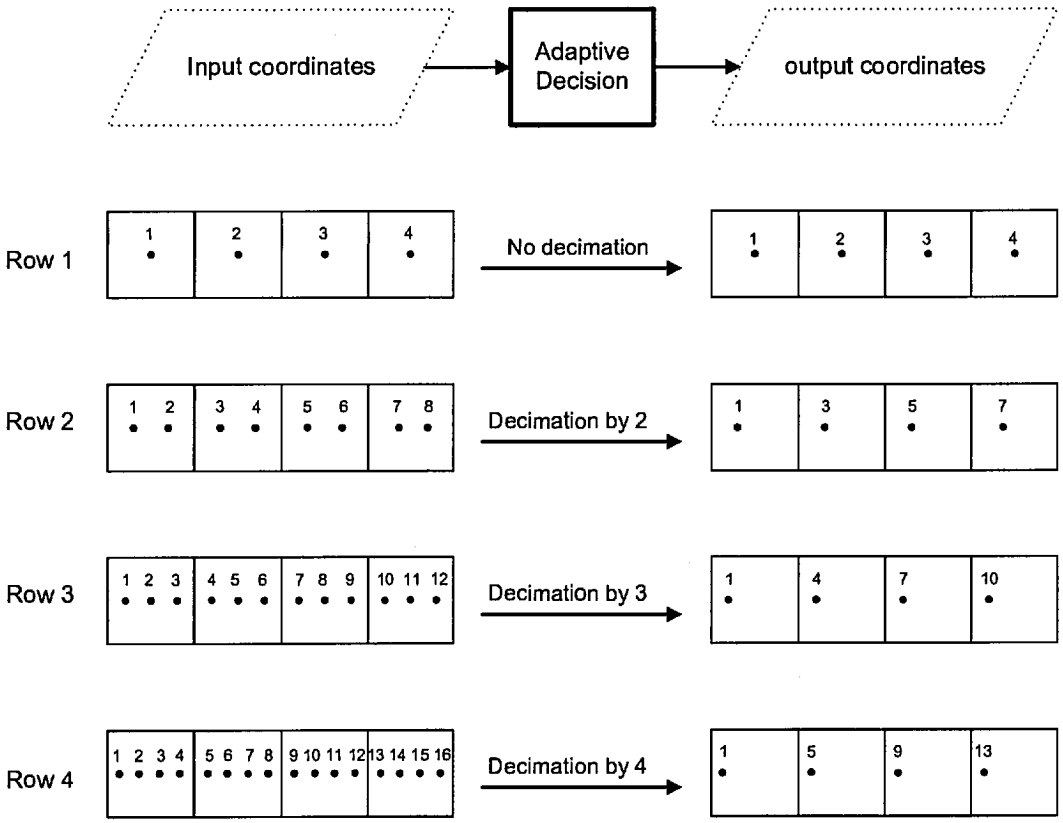


Figure 4.6: The adaptive decision process by decimation operation

The 2D feature in Row 1 moves across one pixel every frame, and all the coordinates are passed through the module to the output. In Row 2, the 2D feature moves across one pixel every two frames. From frame 1 to frame 2, the 2D feature stays on the same pixel, which implies stationary due to the precision of the feature tracker. Every second coordinates of the 2D feature are discarded and the rest are passed through the module to the output. This

process is called decimation by 2. Therefore, the output coordinates of the 2D feature in Row 2 span across different pixels. This provides better temporal information to the next module in the spatiotemporal saliency framework. According to the average speed of the 2D feature, the coordinates are decimated by varying degree. The coordinates of 2D feature with even slower motion need to be decimated more frequently. According to the motion of each 2D feature, this process allows the framework to adapt to slow motion that cannot otherwise be analyzed properly at the current frame rate.

The adaptive decision is made on individual 2D feature independent to the others, thus the coordinates of the slow-moving 2D features are decimated. In a dynamic scene where fast-moving features and slow-moving features co-exist, this feature-wised process is capable of decimating the coordinates of the slow-moving features while the fast-moving features are unaffected. This must be distinguished from the operation that discards the whole frame.

4.2 Hardware efficient architectures for eigenvalue computation

In the previous section, algorithms for the realization of the **2D Feature Detection** module, the **Feature Tracking** module, and the **Adaptive Decision** module are described. The eigenvalue computation used in Shi and Tomasi algorithm [ST94] in the **2D Feature Detection** module is singled out as the most challenging task for implementation in hardware. The rest of the algo-

rithms only consists of common operations such as addition, multiplication, etc. An improved architecture on eigenvalue computation not only benefits our framework, but is also contributive to many areas due to its wide applicability. To name a few in the field of video processing, eigenvalue computation plays an important role in a variety of algorithms such as optical flow algorithm [Far00] and motion analysis [WA93]. The inevitable trend of escalating number of pixels within images and videos is clear in commercial cameras and televisions. As the number of pixels increases, it is critical to have a dedicated piece of hardware that can achieve high throughput of eigenvalues efficiently in real-time.

In this section, we focus on two methods. The first one is based on the approximate Jacobi method, which calculates the eigenvalues for general $n \times n$ symmetric matrices. The second one resorts to the algebraic method for the special case of 3×3 symmetric matrices. The novel contributions of this work are:

- A detailed hardware implementation of the *Approximate Jacobi Method.*
- To demonstrate the efficiency of the proposed architecture, various other architectures have also been implemented and area-throughput design space comparison has been conducted among them.
- A high throughput architecture is developed based on the *Algebraic Method* which is area efficient.
- On implementing the *Algebraic Method,* we propose a hybrid algorithm

to efficiently solve the $\arcsin(q/p)$ subproblem.

4.2.1 Background

For eigenvalue computation, Jacobi-like methods have inherent parallelism and this underlying property makes them particularly suitable for distributed resource system. Systolic array of processors [BLVL85, Kun82, Sch82] was originally proposed for VLSI and the advent of Field Programmable Gate Array (FPGA) popularized this architecture [KIA02, AAB03]. While previous implementations employed the *Exact Jacobi Method*, Gotze *et al* [GPS93] proposed an approximation to the *Exact Jacobi method* and claimed that it is theoretically superior to the exact counterpart without experimental evidence. Besides the Jacobi-based method, the *Algebraic Method* [Wvd91] offers an alternative approach that may be more efficient for some restricted classes of eigenvalue problems. All of these reported hardware architectures lack solid comparative data to allow a system designer to choose between them with confidence. In particular, modern FPGAs have rich hardware resources that are particularly suitable for high throughput eigenvalue computation. This section provides the important comparative evaluation of these methods for computing eigenvalues in terms of resource utilization and throughput.

4.2.2 Exact Jacobi Method

To find the eigenvalues of a symmetric matrix $\mathbf{A}$, there is a framework that takes advantage of the following property. If $\mathbf{A}$ is a symmetric matrix, then

there exists a real orthogonal matrix $\mathbf{Q}$ such that

$$\mathbf{Q}^{\mathrm{T}}\mathbf{A}\mathbf{Q} = \mathbf{D} = \mathrm{diag}(\lambda_1, \ldots, \lambda_n) \tag{4.8}$$

The framework has the objective to render the diagonal matrix $\mathbf{D}$, whose diagonal elements are the eigenvalues of matrix $\mathbf{A}$. Two of the most widely used methods that belong to this framework are: QR methods and Jacobi methods.

QR methods rely on QR decomposition

$$\mathbf{A} = \mathbf{Q}\mathbf{R} \tag{4.9}$$

where $\mathbf{Q}$ is an orthogonal matrix and $\mathbf{R}$ is an upper triangular matrix. Efficient QR methods firstly use a finite sequence of Householder transformations to reduce $\mathbf{A}$ to tridiagonal form, and then apply a version of the QR decomposition to obtain all the eigenvalues and eigenvectors of the tridiagonal matrix [GVL96].

We choose Jacobi methods rather than QR methods, because Jacobi methods are inherently more parallel, which is a valuable merit in distributed resource system. As will be shown later, operations in Jacobi methods are very hardware friendly in terms of resource usage, i.e. no square root or division is involved.

Methodology of Exact Jacobi Method

Given a symmetric $n \times n$ matrix $\mathbf{A}$, Jacobi methods [GVL96] work by systematically reducing the "norm" of the off-diagonal elements as defined in (4.10).

$$\mathcal{F}(\mathbf{A}) = \sqrt{\sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} a_{ij}^2} \tag{4.10}$$

This is accomplished by performing a sequence of orthogonal similarity transformations as in (4.11).

$$\begin{aligned} \mathbf{A}^{(k+1)} &= \mathbf{J}_{pq}^T \mathbf{A}^{(k)} \mathbf{J}_{pq}, \quad k = 0, 1, 2, ... \\ \text{with} \quad \mathbf{A}^{(0)} &= \mathbf{A} \end{aligned} \tag{4.11}$$

$\mathbf{J}_{pq}$ is the transformation matrix governing the Jacobi rotation and is defined by the parameters $(c, s, -s, c)$ in the (pp, pq, qp, qq) entries of a $n \times n$ identity matrix, where $p < q$, $c = \cos(\theta)$, $s = \sin(\theta)$, and θ is the rotation angle. By applying (4.11) iteratively to make off-diagonal element zero, the diagonal elements converge to the eigenvalues of $\mathbf{A}$, denoted as λ_i.

Since $\mathbf{J}_{pq}$ is an orthogonal transformation, (4.12) holds,

$$\|\mathbf{A}^{(k+1)}\|_F = \|\mathbf{A}^{(k)}\|_F \tag{4.12}$$

where $||...||_F$ denotes the Frobenius norm[1]. Each update in (4.11) affects only

[1]Frobenius norm of an $m \times n$ matrix C is defined as the square root of the sum of the absolute squares of its elements, $\|C\|_F \equiv \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |c_{ij}|^2}$

the p and q rows and columns of $\mathbf{A}^{(k)}$ [GVL96], thus

$$[\mathcal{F}(\mathbf{A}^{(k+1)})]^2 = [\mathcal{F}(\mathbf{A}^{(k)})]^2 - 2[(a_{pq}^{(k)})^2 - (a_{pq}^{(k+1)})^2] \tag{4.13}$$

With the exact Jacobi rotation, the coefficients c and s at each iteration are computed using (4.14) and (4.15) so that $a_{pq}^{(k+1)}$ becomes zero and hence maximal reduction of $\mathcal{F}(\mathbf{A}^{(k+1)})$ is achieved.

$$\theta = \frac{1}{2}\tan^{-1}\left[\frac{2a_{pq}^{(k)}}{a_{qq}^{(k)} - a_{pp}^{(k)}}\right] \tag{4.14}$$

$$c = \cos(\theta), \quad s = \sin(\theta) \tag{4.15}$$

Hardware Consideration of Exact Jacobi Method

The *Exact Jacobi Method* comprises the calculation of angle θ from (4.14) followed by the orthogonal similarity transformations in (4.11). The latter process is equivalent to the multiplications of elements of matrix $\mathbf{A}$ with c and s in (4.15). These trigonometric operations can be performed with only addition, shifting and multiplication via the CORDIC (COordinate Rotation DIgital Computer) algorithm [Del89]. The use of CORDIC in the context of eigenvalue computation is described in [CL88]. In fact, all the eigenvalue computation methods depicted in this section involve CORDIC algorithm to a certain extent, so it is reasonable to devote the following paragraphs to the explanation of the algorithm.

The CORDIC algorithm rotates a vector $[x, y]$ by an arbitrary angle θ in

a hardware friendly way. The idea revolves around the use of the elementary angles θ_i

$$\theta_i = \pm \arctan 2^{-i} \quad \text{with} \quad i = 0, 1, ...b-1 \tag{4.16}$$

where b is the wordlength of the hardware system. They all have the important property that the rotation by any of these elementary angles requires only shifts, additions and multiplications by a scaling factor. Since any arbitrary angle can be composed by a combination of the elementary angles, the rotation by any arbitrary angle can be obtained by performing successive rotations of elementary angles. More importantly, the individual scaling can be removed from each iteration and the accumulative scaling factor is applied only once after the last iteration. As a result, only one multiplication is involved regardless of the number of the iterations. Each CORDIC iteration corresponds to the rotation by an elementary angle and the equations are shown in (4.17) to (4.20).

$$x_{i+1} = x_i - d_i \cdot y_i \cdot 2^{-i} \tag{4.17}$$

$$y_{i+1} = y_i + d_i \cdot x_i \cdot 2^{-i} \tag{4.18}$$

$$z_{i+1} = z_i + d_i \cdot \arctan(2^{-i}) \tag{4.19}$$

$$\text{where} \qquad d_i = \text{sign}(z_i) \tag{4.20}$$

As i increases, the elementary angle $\pm \arctan 2^{-i}$ decreases and the net

rotations converge to the desired rotation of the target arbitrary angle. The accumulative scaling factor which need to be multiplied with the resulting vector after b iterations [Del89] is

$$\frac{1}{K_b} = \prod_{i=0}^{b-1} \frac{1}{\sqrt{1+2^{-2i}}} \tag{4.21}$$

By convention, a **"step"** of Jacobi methods consists of finding the angle θ which satisfies (4.14), followed by a rotation by θ. A more thorough implementation of the *Exact Jacobi Method* can be found in [KIA02, AAB03]. The angle θ can be calculated without division operation by setting the initial CORDIC condition as

$$x_0 = a_{qq} - a_{pp} \tag{4.22}$$

$$y_0 = 2a_{pq} \tag{4.23}$$

$$z_0 = 0 \tag{4.24}$$

The initial condition can be visualized as having a vector on the Cartesian plane, whose angle is equal to θ. Hence, by rotating this vector to x-axis, the angle accumulator z would be equal to θ. To be more specific, the CORDIC rotation equations (4.17), (4.18) and (4.19) need to be applied. The rotation direction in each CORDIC iteration is controlled by d_i in (4.25).

$$d_i = -\text{sign}(y_i) \tag{4.25}$$

The angle θ is obtained after b iterations, where b is the wordlength of the system.

After θ is obtained, the rotation by θ is the classical application of the CORDIC algorithm by using rotation equations (4.17), (4.18), (4.19) and (4.20). It should be noted that the value of the scaling factor $1/K_b$ in (4.21) depends only on the wordlength b and can be pre-computed. This scaling can be executed with approximately $b/4$ shifts and additions [AAB03].

4.2.3 Window Jacobi Method

Although each step of the *Exact Jacobi Method* causes one off-diagonal element to be annihilated, i.e. made zero, this zero is destroyed in the next step. Hence the effort of previous step is wasted to some extent and precise annihilation of elements is an overkill but for the very last steps [MP85, Del92]. Thus, the accuracy of annihilation may be coarse in the early steps and refined as the algorithm proceeds. This can be regarded as having a window of iteration indexes chosen for the CORDIC algorithm, so we call this method the *Window Jacobi Method.*

Hardware Consideration of Window Jacobi

The methodology of the *Window Jacobi Method* is the same as the *Exact Jacobi Method.* They both compute θ from (4.14) and rotate the matrix with c and s from (4.15) in each step. The difference of the *window* from *exact* Jacobi method is that the CORDIC algorithms applied for these operations

have variable resolution. Instead of having CORDIC rotations with the index in $\theta_i = \pm \arctan 2^{-i}$ running from 0 to $b-1$, the index can run from m to n, where $m \geq 0$ and $n \leq b$. Experiments in [Arj92] and [Sch91] indicate that, with a fixed window width $n-m+1$ equal to 3 or 4, the total number of steps barely exceeds the number of steps in the *Exact Jacobi Method.* Consequently the total number of CORDIC iterations in the *Window Jacobi Method* can be significantly less than that of the *Exact Jacobi Method.* However, as the *Exact Jacobi Method* is designed to make a particular matrix element zero at every step, one element can be simply overwritten by zero in the *exact* method. Whereas this element must be computed via CORDIC in the *window* method, which incurs hardware overhead. More control logic is also inevitable to select the CORDIC indexes window.

4.2.4 Approximate Jacobi Method

Abundance of work, both software and hardware concerned, have been done on the *Exact* and *Window* Jacobi Method. On the other hand, information on the hardware implementation of the *Approximate* Jacobi Method lacks in the literature. For this reason, more details will be elaborated on this method in this paper. The *Approximate Jacobi Method* takes a further step forward than the *Window Jacobi Method* to target more efficient use of hardware resources. Taking into account the computational effort, the best way to reduce $\mathcal{F}(\mathbf{A})$ in (4.10) is not necessarily finding the $\mathbf{J}_{pq}$ in (4.11), since the annihilation effort is undermined in the next step. It is sufficient to approximate the $\mathbf{J}_{pq}$ as long

as the orthogonality is preserved [GPS93]. While the *Window Jacobi Method* performs a "window" of CORDIC iterations whose resultant angle is close to the θ in (4.14), the *Approximate Jacobi Method* finds the CORDIC elementary angle in (4.16) that is closest to the θ. Rotating by elementary angle requires one single CORDIC iteration, whereas rotation by the "exact" θ requires b number of iterations, where b is the wordlength of the system. As a result, the redundant operations pursuing the $\mathbf{J}_{pq}$ is eliminated.

Methodology of the Approximate Jacobi Method

The *Approximate Jacobi Method* does not annihilate the off-diagonal element $a_{pq}^{(k+1)}$, but reduces it by a factor d as in (4.26).

$$a_{pq}^{(k+1)} = d a_{pq}^{(k)} \ , \ \text{where} \quad d = \frac{1 - 2\tau t - t^2}{1 + t^2} \tag{4.26}$$

The maximal value of $|d|$, denoted as $|d|_{\max}$, is a measure for the quality of the approximation since a smaller $|d|_{\max}$ gives rise to a bigger reduction of $a_{pq}^{(k+1)}$. The *Exact Jacobi Method*, with t computed as in (4.14), yields $d = 0$ at each update. On the contrary, *Approximate Jacobi Method* approximates t with hardware efficient operators notably adders and shifters. The approximated $\tilde{t}$ is shown in (4.27),

$$\tilde{t} = \text{sign}(\tau) \cdot 2^{-l} \approx t = \tan(\theta), \quad l \in \{0, 1, 2, ..., b-1\} \tag{4.27}$$

where b denotes the wordlength of the hardware system. The rotation that *approximates* the Jacobi rotation is described by $\tilde{\mathbf{J}}_{pq}(l)$, which is a $n \times n$ identity matrix with its four entries at (pp, pq, qp, qq) replaced by (4.28).

$$\tilde{\mathbf{J}}_{pq}(l) = \frac{1}{\sqrt{1+2^{-2l}}} \begin{bmatrix} 1 & \text{sign}(\tau)2^{-l} \\ -\text{sign}(\tau)2^{-l} & 1 \end{bmatrix} \tag{4.28}$$

It is essential for the *Approximate Jacobi Method* to choose l such that the approximate angle $\tilde{\theta} = \arctan(2^{-l})$ is the closest to the exact rotation angle θ and $|d|_{\max}$ is minimized. Gotze *et al* [GPS93] derived the following formulae which requires at most three comparisons to find l while guaranteeing $|d|_{\max} \leq 1/3$.

- Compute k according to (4.29), where $a_D = a_{qq} - a_{pp}$ and exp denotes the exponent of a number. In binary number representation, exp means finding the position of the most significant "1" bit in an expression.

$$k = \exp(|a_D|) - \exp(|a_{pq}|) \tag{4.29}$$

- Determine a set of possible l using Table 4.1.

k	selection of l
$k \leq -2$	0
$-2 < k \leq 0$	$\{0, 1\}$
$k > 0$	$\{k-1, k, k+1\}$

Table 4.1: Selection of l depending on k

- When $k > -2$, l is chosen by satisfying the following comparison.

$$(2^l - 2^{-l+1})|a_{pq}| \leq |a_D| + 2^{-1}|a_D| < (2^{l+1} - 2^{-l})|a_{pq}|$$

The core idea that contributes to the efficiency of the *Approximate Jacobi Method* is that the angle of rotation is designed to be equal to a CORDIC elementary angle. This is evident by comparing the approximate Jacobi angle (4.27) with the CORDIC elementary angle (4.16). Hence, the *approximate* Jacobi rotation is equivalent to just one iteration of the CORDIC algorithm as opposed to b iterations in the *exact* Jacobi counterpart. However, the scaling factor varies with l, which determines the choice of the CORDIC elementary angle. The corresponding scaling factor is given in (4.30).

$$\frac{1}{K_l} = \frac{1}{\sqrt{1 + 2^{-2l}}} \tag{4.30}$$

As l increases, the scaling factor converges to unity. The implication is that $1/K_l$ with large l need not to be stored. The number of $1/K_l$ needs to be stored is around $b/2$.

Systolic Architecture of the Approximate Jacobi Method

The systolic array architecture [Kun82, Sch82], which offers a high degree of parallelism, is proposed for the hardware implementation of the *Approximate Jacobi Method*. The architecture is in line with previous work in the literature for the implementation of the *Exact Jacobi Method* [BLVL85, KIA02, AAB03]. Each processor within the systolic array has bilateral communication with its

nearest neighbors to avoid broadcasting of signals. One of the benefits of this short distance communication is that as the size of the systolic array increases, the complexity of routing between processors does not increase. To further take advantage of the symmetry of the matrix, we adopted an upper triangular array as suggested in [GPS93], rather than a full square array of processors. This is illustrated in Figure 4.7.

Since the $\tilde{\mathbf{J}}_{pq}(l)$ in (4.28) differs from the identity matrix at four locations (pp, pq, qp, qq), the approximate Jacobi rotation affects only the p and q columns and rows of the matrix. This underlying characteristic ensures that different pq pairs have independent operations, thus allowing the *sequential* process of orthogonal transformation in (4.11) to be mapped into the *parallel* architecture. In our implementation, p are odd integers and q are even integers. For example, the pair $p = 1, q = 2$ involves the first column, second column, first row, and second row. a_{pp}, a_{pq} and a_{pq} are fed into the Diagonal processor where approximate Jacobi rotation is executed and give rise to the reduction of a_{pq} according to (4.26). This reduction eventually leads to the resultant matrix to be diagonal with the eigenvalues on the diagonal. The orthogonal transformation affects other elements on the same rows and columns. These elements are fed into the off-diagonal processors for corresponding updating of the matrix. The input matrix can be divided into 2×2 blocks and elements in each block is mapped to a processor. Due to symmetry of the input matrix, each diagonal processor processes three elements of the matrix and each off-diagonal processor processes four.

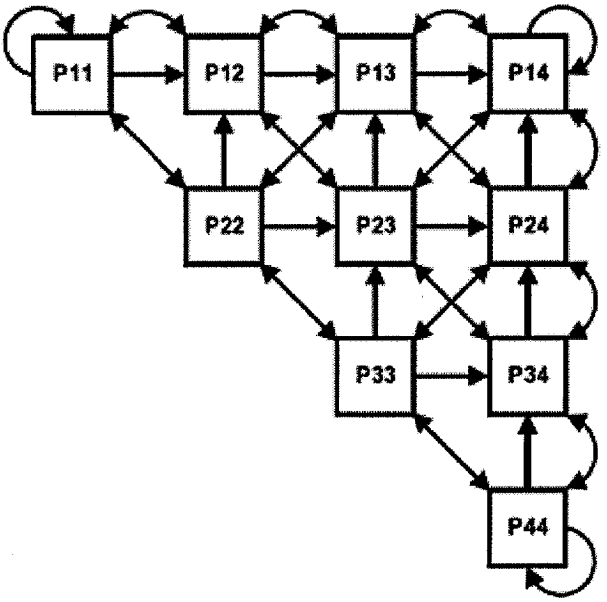


Figure 4.7: Upper triangular array of processors for 8×8 symmetric matrix eigenvalue computation

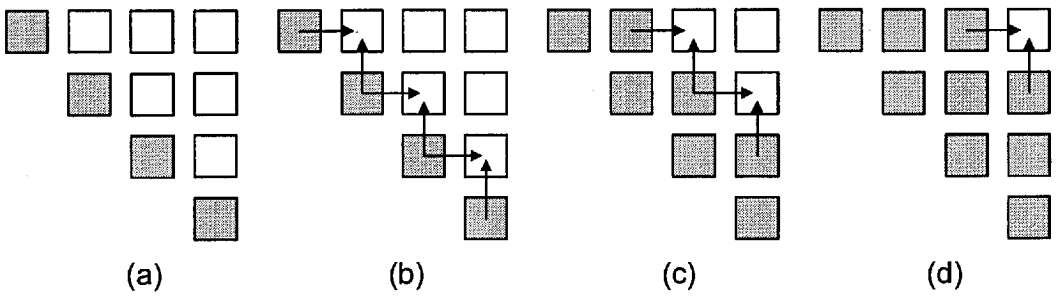


Figure 4.8: Dataflow of the systolic array, engaging processors are shown in grey and arrows indicate the transmission of l and sign(τ)

Diagonal Processor

The block diagram of a diagonal processor is shown in Figure 4.9. Each diagonal processor accomplishes two tasks: computation of l and updating elements of matrix through the approximate Jacobi rotation. On receiving an input matrix, all the diagonal processors compute the quantity l in parallel as illustrated in Figure 4.8(a). l computed by each diagonal processor is transmitted to its neighboring off-diagonal processors in both horizontal and

vertical directions shown in Figure 4.8(b). sign(τ) is transmitted in the identical way. Following analogous derivation to the update equations of *Exact Jacobi Method* [BLVL85], the approximate rotation blocks execute the update operations shown below.

$$\begin{aligned} a_{pp}^{(k+1)} &= a_{pp}^{(k)} - \text{sign}(\tau)2^{-l+1}a_{pq}^{(k)} + 2^{-2l}a_{qq}^{(k)} \\ a_{pq}^{(k+1)} &= 2^{-2l}a_{pp}^{(k)} + \text{sign}(\tau)2^{-l+1}a_{pq}^{(k)} + a_{qq}^{(k)} \\ a_{qq}^{(k+1)} &= \text{sign}(\tau)2^{-l}(a_{pp}^{(k)} - a_{qq}^{(k)}) - 2^{-2l}a_{pq}^{(k)} + a_{pq}^{(k)} \end{aligned}$$

After the updating, every diagonal processor reduces its individual a_{pq} element, contributing to the progressive creation of the diagonal matrix that contains eigenvalues. Therefore, four reductions happen in parallel in the case illustrated in Figure 4.8 where an 8×8 symmetric matrix is assumed.

In the diagonal processor, the scaling factor $1/K_l$ has to be applied twice [GPS93] and hence the square root in (4.30) is removed. Therefore, we can execute scaling recursively using (4.31) with only shifts and additions [Del89].

$$1/K_{i+1}^2 = K_i^2(1 + 2^{-2^{i+1}l}), \quad K_1^2 = 1 - 2^{-l} \tag{4.31}$$

Given b as the wordlength of the diagonal processor, the recursion terminates when $2^{i+1}l \geq b$, i.e. after $\log_2 \lceil \frac{b}{2l} \rceil$ recursions, because it reaches the accuracy of the system. After the scaling, the matrix elements are updated and exchanged with those in the adjacent processors.

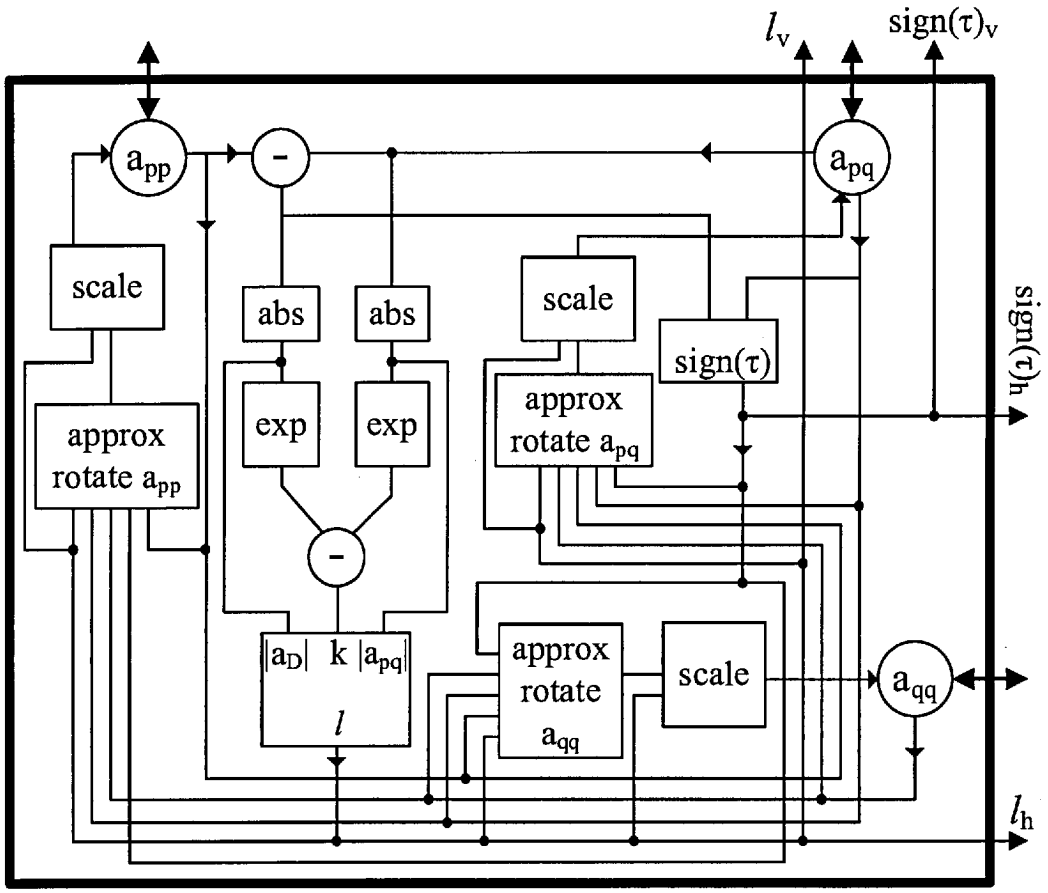


Figure 4.9: Block diagram of a Diagonal Processor

Off-diagonal Processor

The block diagram of an off-diagonal processor is shown in Figure 4.10. Each off-diagonal processor has to wait for the arrival of l and sign(τ) signals from the adjacent processors in both horizontal and vertical directions. In Figure 4.8(b), the off-diagonal processors closest to the diagonal processors receive the l and sign(τ) signals. In the next clock cycle shown in Figure 4.8(c), they pass on the signals to their vertical and horizontal neighboring off-diagonal processors. In the meanwhile, having received the signals in the previous clock cycle, the off-diagonal processors closest to the diagonal processors start

executing the updating.

Firstly, input matrix elements are subject to approximate rotation determined by the horizontally transmitted l_h and $\text{sign}(\tau)_h$ signals, which have been generated by the diagonal processor on the same row. The update operations consist of right-shifts and additions/subtrations are governed by the equations shown below.

$$\begin{aligned} a_{11h} &= a_{11} - \text{sign}(\tau)_h 2^{-l_h} a_{21} \\ a_{12h} &= a_{12} - \text{sign}(\tau)_h 2^{-l_h} a_{22} \\ a_{21h} &= a_{21} + \text{sign}(\tau)_h 2^{-l_h} a_{11} \\ a_{22h} &= a_{22} + \text{sign}(\tau)_h 2^{-l_h} a_{12} \end{aligned}$$

The scaling is carried out by multiplying one of the pre-computed $1/K_l$ as in (4.30) selected by l_h via a multiplexer.

Secondly, the scaled results are subject to similar operations controlled by vertically transmitted signals to complete the matrix elements update.

Finally, the matrix is updated only after the updating operations are completed in the entire array of processors including both diagonal and off-diagonal processors. In the following clock cycle, the updated matrix elements are exchanged among adjacent processors so that other elements in the matrix can be sent to the diagonal processors for reduction.

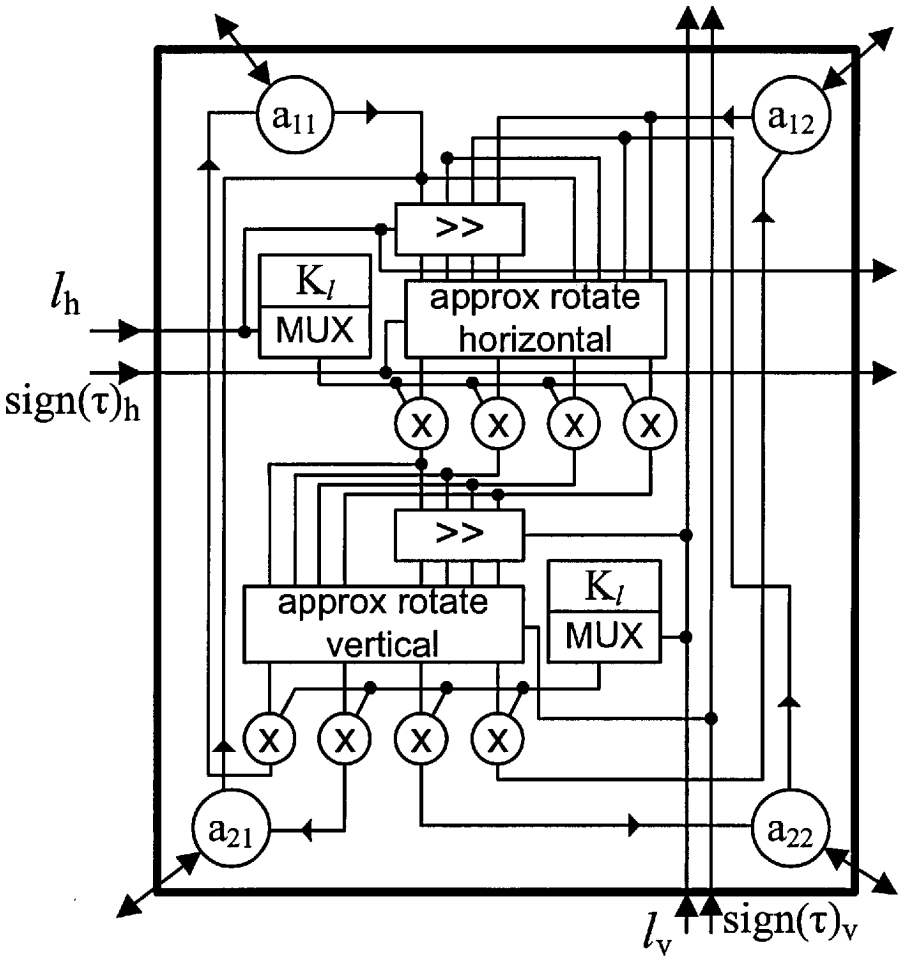


Figure 4.10: Block diagram of an Off-diagonal Processor

Dual Processor Architecture of Approximate Jacobi Method

The systolic architecture is speed-conscious and embodies massive parallelism. On the other end of the spectrum, it is possible to use just one diagonal processor and one off-diagonal processor to carry out the *Approximate Jacobi Method.* Since it only requires two processors to compute eigenvalues regardless the matrix size, we call this architecture dual processor architecture in this section. It is resource-conscious and this alternative architecture is suitable for applications that can only spare limited hardware resource to dedicate to

eigenvalue computation.

In short, the one diagonal processor in the dual processor architecture sequentially performs all the operations done in parallel by the multiple diagonal processors in the systolic architecture, likewise for the off-diagonal processor.

The dual processor architecture is illustrated in Figure 4.11. The matrix elements are stored in the registers and the appropriate elements are multiplexed to the two processors. Both processors are identical to those in the systolic architecture and they update the matrix elements and eventually render the eigenvalues. The updated elements are stored back to the registers. While the diagonal processor computes l and sign(τ) signals, the off-diagonal processor stays idle and it commences computation as soon as the signals are generated and received from the diagonal processor. The control unit oversees the registers, the multiplexer and the two processors.

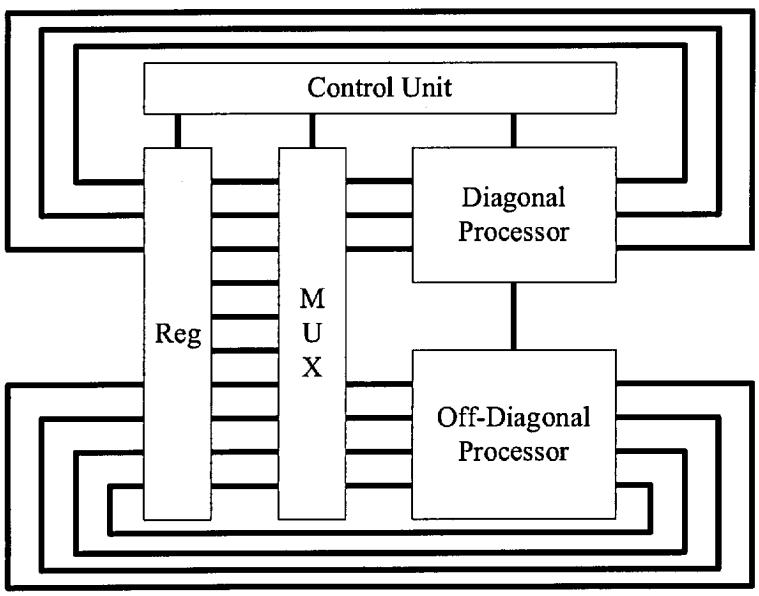


Figure 4.11: Dual Processor architecture

Discussion on Fixed-point System

In our hardware implementation, the elements of the input matrix are assumed to be in the range of -1 to $1 - 2^{-b+1}$, where b is the wordlength of the input. Hence, the most significant bit is the sign and the rest of bits represent the fractional part of the number. If the range assumption is violated, i.e. any element in the input matrix exceed the range, the matrix must be normalized to the range through dividing every elements in the matrix by a constant. So that the largest positive number is less than or equal to $1 - 2^{-b+1}$ and the smallest negative number is greater than or equal to -1. The computed eigenvalues shall then be scaled back by multiplying by the same constant to retrieve the eigenvalues of the original input matrix.

In [HJ90], it shows that the absolute value of the eigenvalue λ of matrix $\mathbf{A}$ is bounded by its elements in the manner as in (4.32).

$$|\lambda| \leq \max(\text{SumOfRow}(\text{ABS}(\mathbf{A}))) \tag{4.32}$$

where ABS takes absolute value of every element in the matrix. SumOfRow returns a vector formed by summing the *row* of the matrix. The input matrix element range of -1 to $1-2^{-b+1}$ guarantees the upper bound of the eigenvalue. For example, for an input matrix 4×4, the absolute value of its eigenvalue is less than or equal to 4. Thus, by assigning two extra bits to the most significant part of the signals, overflow and underflow is avoided.

To clarify notation, the operations involved in reducing the entire set of

off-diagonal matrix elements is collectively known as one *sweep*. With finite fixed-point accuracy, we discovered that although $\mathcal{F}(\mathbf{A})$ in (4.10) decreases to a certain minimum with approximate Jacobi rotations, the eigenvalues computed with excessive sweeps depart from the actual eigenvalues. This is due to truncation error propagating to the next sweep. The eigenvalue instability can be rectified by executing rounding instead of executing truncation. This improvement can be seen from Figure 4.12, where the maximum percentage error is calculated from (4.33).

$$\text{error}_{\max\%} = \frac{|\tilde{\lambda} - \lambda|_{\max}}{2^b} \tag{4.33}$$

$\tilde{\lambda}$ is the computed eigenvalue, λ is the quantized true eigenvalue and b is the input wordlength.

For 3000 random 4×4 matrices in 16-bits system, the maximum percentage errors are shown in Figure 4.12 for the proposed architecture. It is clear that the rounding stabilizes the eigenvalue result.

However, the overhead in terms of area imposed by the rounding operation can not be ignored. For 4×4 16-bits systems, the area increases by 27% for the systolic architecture.

For some hardware architectures, e.g. FPGAs, the available embedded multipliers can be exploited. This is beneficial since the scaling in the off-diagonal processors is achieved through multiplication. In a 4×4 16-bits system, two 18×18 embedded multipliers trade for 1343 slices.

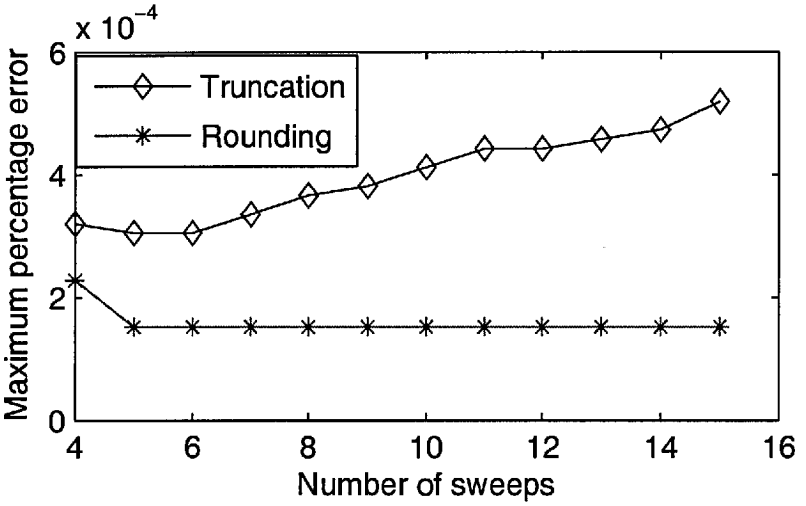


Figure 4.12: Max percentage error in the eigenvalues using rounding and truncation schemes.

4.2.5 Algebraic Method

Methodology of Algebraic Method for 3×3 symmetric matrices

For the special case of a 3×3 symmetric matrix $\mathbf{M}$, the eigenvalues can be calculated algebraically. Given

$$\mathbf{M} = \begin{pmatrix} a & d & e \\ d & b & f \\ e & f & c \end{pmatrix}$$

the eigenvalues can be expressed as roots of a third degree characteristic polynomial $p(\lambda) = \det(\mathbf{M} - \lambda\mathbf{I})$ [GVL96]. Let this characteristic equation be

$$\lambda^3 + k\lambda^2 + l\lambda + m = 0 \tag{4.34}$$

where

$$\begin{aligned} k &= -(a+b+c) \\ l &= ab+bc+ac-f^2-e^2-d^2 \\ m &= af^2-abc-2fde+be^2+cd^2 \end{aligned} \tag{4.35}$$

By substituting $\lambda = x - k/3$, we remove the λ^2 term in (4.34).

$$x^3 + px + q = 0 \tag{4.36}$$

where

$$p = -\frac{1}{3}k^2 + l, \;\; q = \frac{2}{27}k^3 - \frac{1}{3}lk + m \tag{4.37}$$

The condition of symmetric matrix $\mathbf{M}$ assures that all eigenvalues are real numbers [GVL96]. Following [Wvd91], we substitute $x = \sqrt{-\frac{4p}{3}}y$, yielding

$$4y^3 - 3y = \frac{3q}{p\sqrt{-\frac{4p}{3}}}$$

Using the trigonometric relation $\cos 3\psi = 4\cos^3\psi - 3\cos\psi$ and scaling $y = \cos\psi$, we have

$$\cos 3\psi = \frac{3q}{p\sqrt{-\frac{4p}{3}}}$$

The three real solutions to (4.36) are computed and the eigenvalues of **M** are

$$\begin{aligned}
\lambda_1 &= x_1 - \frac{k}{3} = \beta\cos\psi - \frac{k}{3} \\
\lambda_2 &= x_2 - \frac{k}{3} = \beta\cos\left(\psi - \frac{2\pi}{3}\right) - \frac{k}{3} \\
\lambda_3 &= x_3 - \frac{k}{3} = \beta\cos\left(\psi + \frac{2\pi}{3}\right) - \frac{k}{3}
\end{aligned}$$

where

$$\beta = \sqrt{-\frac{4p}{3}}, \quad \psi = \frac{1}{3}\left(\frac{\pi}{2} - \arcsin\frac{3q}{p\beta}\right) \tag{4.38}$$

The range of ψ, $\psi \in (0, \pi/3)$, imposes an ordering to the acquired eigenvalues which is $\lambda_1 \geq \lambda_2 \geq \lambda_3$. This provides the designer of the system the flexibility to select only the desired eigenvalues.

Hybrid arcsin CORDIC algorithm

The *Algebraic Method* for eigenvalue computation requires the computation of a wide division followed by an arcsin function shown in (4.38). CORDIC algorithm can be used to effectively avoid the division and solve this problem. Specifically, α in (4.39) need to be computed.

$$\alpha = \arcsin\left(\frac{q}{p}\right) \tag{4.39}$$

The algorithm begins with placing a vector of magnitude p on the x-axis as shown in Figure 4.13. This vector is rotated according to the rules governed by the CORDIC iteration equations (4.17), (4.18) and (4.19). The rotations

finish until the vector's y component is equal to c, the choice of which will be determined conveniently to solve the problem laid in (4.39). We introduce r_i to denote the magnitude of the vector $[x_i, y_i]$

$$r_i = \sqrt{(x_i^2 + y_i^2)}$$

The initial conditions depicting the vector on x-axis are as follow.

$$x_0 = p \tag{4.40}$$

$$y_0 = 0 \tag{4.41}$$

$$z_0 = 0 \tag{4.42}$$

$$r_0 = \sqrt{(x_0^2 + y_0^2)} = p \tag{4.43}$$

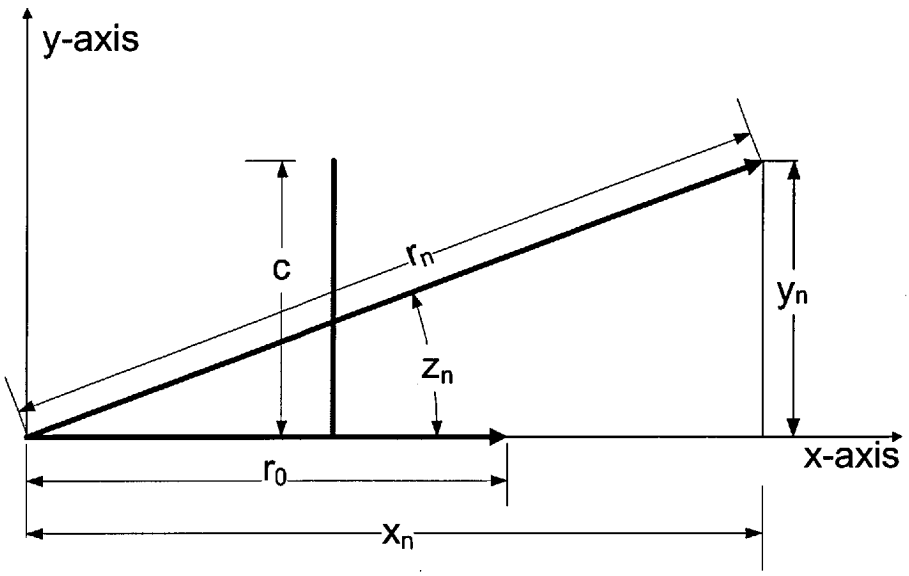


Figure 4.13: arcsin CORDIC rotations after n iterations

The direction to which the vector is rotated in the next iteration is decided by d_i. It is the result of a comparison between c and the y component of the rotated vector at the current iteration

$$d_i = \text{sign}(c - y_i) \tag{4.44}$$

After n iterations, r_0 has been scaled by A_n and the arcsin CORDIC algorithm produces

$$\begin{aligned} r_n &= A_n \cdot r_0 = A_n \cdot p \\ x_n &= \sqrt{(A_n \cdot p)^2 - c^2} \\ y_n &= c \\ z_n &= \arcsin\left(\frac{c}{A_n \cdot p}\right) \\ A_n &= \prod_{i=n} \frac{1}{K_i} = \prod_{i=n} \sqrt{1 + 2^{-2i}} \end{aligned}$$

This is illustrated in Figure 4.13 and if we choose c to be

$$c = A_n \cdot q \tag{4.45}$$

then z_n becomes

$$z_n = \arcsin\left(\frac{A_n \cdot q}{A_n \cdot p}\right) = \arcsin\left(\frac{q}{p}\right)$$

which is the desired function.

Effectively, each CORDIC iteration rotates the vector by angle θ_i and

scales the magnitude of the vector r_i by K_i, where $\theta_i = \arctan 2^{-i}$ and $K_i = \sqrt{1+2^{-2i}}$ respectively. Because of the scaling in magnitude, this computation can nonetheless cause incorrect decision making, i.e. the choice of d_i. Consequently, the vector rotates away from desirable α and the arcsin result in angle accumulation in (4.19) would be incorrect, which is illustrated in Figure 4.15.

If the scaling is compensated in each iteration, the aforementioned decision error can be resolved. This is essentially a general rotation, which is defined as a vector rotation without changing the vector's magnitude. In terms of computation, general rotations are more expensive, since each general rotation is equivalent to carrying out a conventional shift-and-add CORDIC rotation as well as a multiplication for the scaling.

It is observed that the resultant angle computed by the conventional arcsin CORDIC algorithm departs from the true angle more significantly when the incorrect decision is made earlier. Recall from (4.16), θ_i are getting smaller with an increasing i.

If we ensure correct decision making for the first few rotations by resorting to general rotation and apply the conventional CORDIC algorithm afterwards, we achieve a reasonable trade-off between computational complexity and accuracy. The average error in the obtained arcsin angle declines with increasing number of general rotations, which is illustrated in Figure 4.14.

In principle, accuracy is attained from the general rotation and low computation cost is acquired from CORDIC algorithm. Comparison within Figure 4.15 demonstrates the greatly reduced error achieved by the hybrid algorithm.

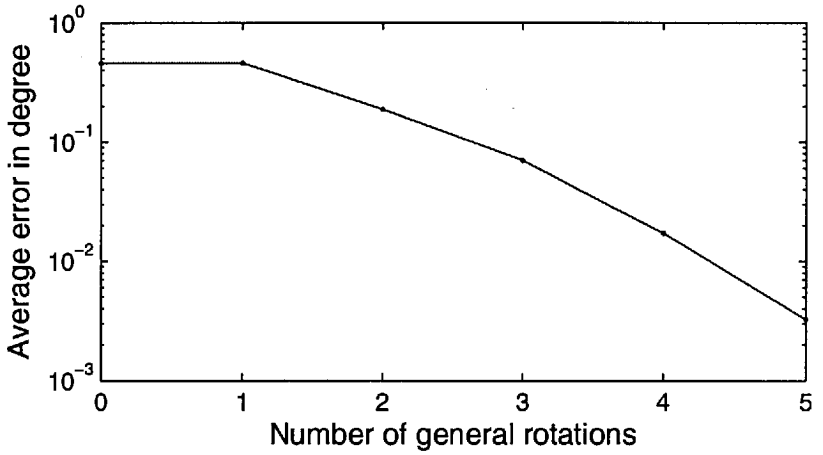


Figure 4.14: Average error in degree of the hybrid arcsin algorithm with respect to the number of general rotations

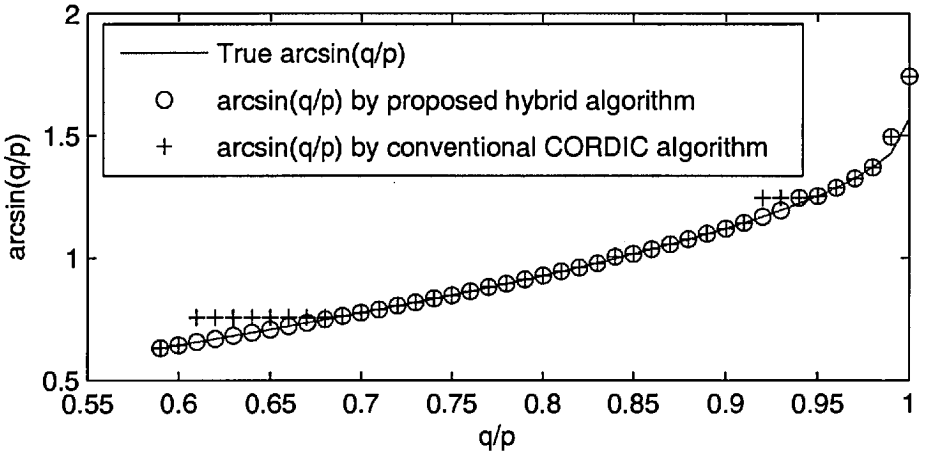


Figure 4.15: arcsin by conventional CORDIC rotations and hybrid algorithm

Architecture of the Algebraic Method

A pipelined architecture is developed based on the feedforward nature of the *Algebraic Method*. Without loss of generality, the design computes the smallest eigenvalue of a 3×3 matrix. The overview of the design is shown in Figure 4.16. The ALU module computes the values of p, q and $k/3$ from (4.37) and (4.35) respectively. This module is the very first stage of the system so the wordlengths of output variables are determined such that early stage trunca-

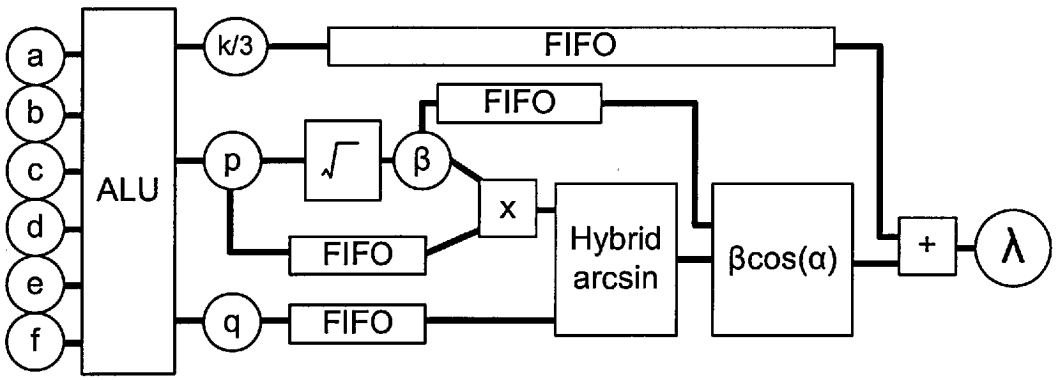


Figure 4.16: Overview of the eigenvalue computation system using the Algebraic Method.

tion error is avoided. To be specific, the wordlength is doubled when there is multiplication and extended one bit when there is addition and subtraction. Existence of common subexpressions has been explored to reduce the number of operators in the ALU.

The square root module adopts the successive approximation algorithm [Man93], which is an iterative process that achieves one bit of accuracy per iteration. The algorithm works by repeatedly halving the range of values, in which the square root of a value is known to exist. For example, to find the square root of N, the initial range is N and it starts with the mid point $\frac{N}{2}$. It then compares $\left(\frac{N}{2}\right)^2$ to N and the result decides which half the square root lies. The search range is halved after each iteration and eventually the square root of desired precision is acquired. In our implementation, the square root module is completely unrolled so that it can be pipelined to provide one square root result every clock cycle.

The hybrid arcsin module implements the **hybrid** arcsin **CORDIC algorithm** described earlier.

4.2.6 Performance Evaluation of Architectures

In this subsection, all the methods described in the previous subsections are compared, namely the *Exact Jacobi Method*, the *Window Jacobi Method* and the *Approximate Jacobi Method* implemented in both *systolic* and *dual processor* architecture. We implemented everything ourselves to ensure that the result is not affected by inconsistent programming style of different authors. Moreover, we compare pipelined architectures of the *Approximate Jacobi Method* and the *Algebraic Method* for 3×3 matrix. The implementation language is Handel-C [celom], which is a high level language that compiles programs into hardware images. The synthesis tool used is the Xilinx® Integrated Software Environment (ISE) software. Target system is the Xilinx® XC2V6000-6 Virtex-II FPGA chip [xil]. In the rest of this section, N-bits system means that the inputs are N-bits fixed-point numbers. Unless stated otherwise, the input matrix is assumed to be 4×4 with 16-bits accuracy.

Both the *Exact Jacobi Method* and *Window Jacobi Method* require a series of CORDIC rotations that have no data dependency on each other. Thus, they can be executed in parallel and each rotation can have a dedicated CORDIC module to fully take advantage of the situation so as to maximize speed. On the other hand, these CORDIC modules can be shared within the same processor to save hardware resource (area). To investigate the impact of CORDIC module sharing, two configurations are applied to the *Exact* and *Window* methods: dedicated CORDIC modules and shared CORDIC modules. Notice for the *Ap-*

proximate Jacobi Method, the rotations are performed with only one CORDIC iteration and thus CORDIC modules sharing is irrelevant.

Area

Table 4.2 summarizes the area (in FPGA slices) of the hardware implementation of the various methods under both architectures. The result demonstrates that the *Approximate Jacobi* hardware is the smallest and the *Window Jacobi* hardware is larger than the *Exact Jacobi* in systolic architecture as expected in Section 4.2.3.

Method	Architecture	Dedicated CORDIC modules	Shared CORDIC modules
Exact Jacobi	systolic	4852	3489
	dual processor	3872	2498
Window Jacobi	systolic	6459	3723
	dual processor	3627	2222
Approximate Jacobi	systolic	4540	N/A
	dual processor	3382	N/A

Table 4.2: Area comparison (in FPGA slices) for 16-bits 4×4 matrix

With respect to the *Approximate Jacobi* systolic architecture, Table 4.3 provides a summary of how the area changes with wordlength as well as matrix size. Further experimental data shows that the area of the *Approximate Jacobi* systolic architecture scales linearly with the wordlength of the input variables.

For the *Approximate Jacobi Method*, a diagonal processor occupies 1235 slices and an off-diagonal processor uses 2255 slices in 16-bits system. Therefore, given a matrix size of $n \times n$, we can predict the total area of the systolic

Matrix size	wordlength	Approximate Jacobi
4×4	15	4135
4×4	16	4474
4×4	17	4698
6×6	16	10006
8×8	16	17735

Table 4.3: Area (in FPGA slices) Variation of the approximate Jacobi systolic architecture

eigenvalue system by summing the processors area according to (4.46),

$$\text{Area}(n \times n) = n \cdot \text{Area}_{\text{D}} + \frac{n(n-1)}{2} \cdot \text{Area}_{\text{O}} \tag{4.46}$$

where Area_{D} is the area of a diagonal processor and Area_{O} is the area of an off-diagonal processor.

Maximum Frequency

The maximum frequencies were obtained by Xilinx® ISE Post-Place & Route Static Timing Analysis [xil] and they are shown in Table 4.4. The dual processor architectures have slightly lower maximum frequencies than the systolic architecture because the routing is more complex. The architectures based on the *Approximate Jacobi Method* have slightly lower frequencies as they require multipliers while others don't.

Number of Clock Cycles per step

In our implementations, the number of clock cycles needed per step in the fastest architecture, systolic architecture with dedicated CORDIC modules,

Method	Architecture	Dedicated CORDIC modules	Shared CORDIC modules
Exact Jacobi	systolic	88.6 MHz	85.8 MHz
	dual processor	84.7 MHz	77.0 MHz
Window Jacobi	systolic	85.0 MHz	86.3 MHz
	dual processor	85.0 MHz	73.0 MHz
Approximate Jacobi	systolic	73.1 MHz	N/A
	dual processor	74.1 MHz	N/A

Table 4.4: Maximum Frequency comparison for 16-bits 4×4 matrix

corresponding to each method is shown in (4.47), (4.48) and (4.49).

$$\mathrm{T}_{\mathrm{exact}} = 15 + 3\frac{1}{2}b \tag{4.47}$$

$$\mathrm{T}_{\mathrm{window}} = 16 + 3W + \frac{1}{2}b \tag{4.48}$$

$$\mathrm{T}_{\mathrm{approximate}} = 15 + \log_2\left(\frac{1}{2}b\right) \tag{4.49}$$

where b indicates the input wordlength, and W denotes the average window size for the *Window Jacobi Method* throughout the eigenvalue computation process. The *Exact Jacobi Method* takes b CORDIC iterations to compute the cosine-sine pair in (4.15) and $2b$ CORDIC rotations to update the matrix. The *Approximate Jacobi Method* needs significantly less clock cycles. It takes at most 3 comparisons to compute l, which is the cosine-sine counterpart. Only one iteration of CORDIC is needed for every approximate rotation. Therefore, while the number of clock cycles per step of the *Exact Jacobi Method* scales linearly with wordlength b as in (4.47), it scales only logarithmically for the *Approximate Jacobi Method* (4.49). In the *Window Jacobi Method*, each step can have different window size, which results in different number of CORDIC

iterations. For the 16-bits 4×4 matrix, where the window is chosen to be 8 for the first 6 steps and 16 for another 3 steps, the average window size W would be 10.67. Table 4.5 summarizes the clock cycles needed per step for this particular case.

Method	Architecture	Dedicated CORDIC modules	Shared CORDIC modules
Exact Jacobi	systolic	71	116
	dual processor	94	182
Window Jacobi	systolic	57	107
	dual processor	103	211
Approximate Jacobi	systolic	18	N/A
	dual processor	36	N/A

Table 4.5: Number of clock cycles in each step for 16-bits 4×4 matrix

Area-Throughput Design space

To render eigenvalues of the same accuracy, the *Approximate Jacobi Method* [GPS93] demands more steps than the *Exact Jacobi Method* and the *Window Jacobi Method*. For 16-bits 4×4 matrix, the *approximate* Jacobi method requires 12 steps against 9 for the *exact* and *window* Jacobi methods. Nevertheless, as shown in Table 4.5, the time saving per step of the Approximate Jacobi architecture is so significant that it outweighs this drawback of slow convergence.

We adopt "throughput" as the speed metric, which indicates the number of eigenvalues generated per second. This metric unifies the maximum frequency, the number of clock cycle per step, and the number of steps per eigenvalue. The Area-Throughput design space for all three methods in both architectures is

illustrated in Figure 4.17. Provided a limited amount of area, the *Approximate Jacobi Method* can generate over two times more eigenvalues per second than other methods. It unambiguously surpasses the rest of the methods, which have comparable performance.

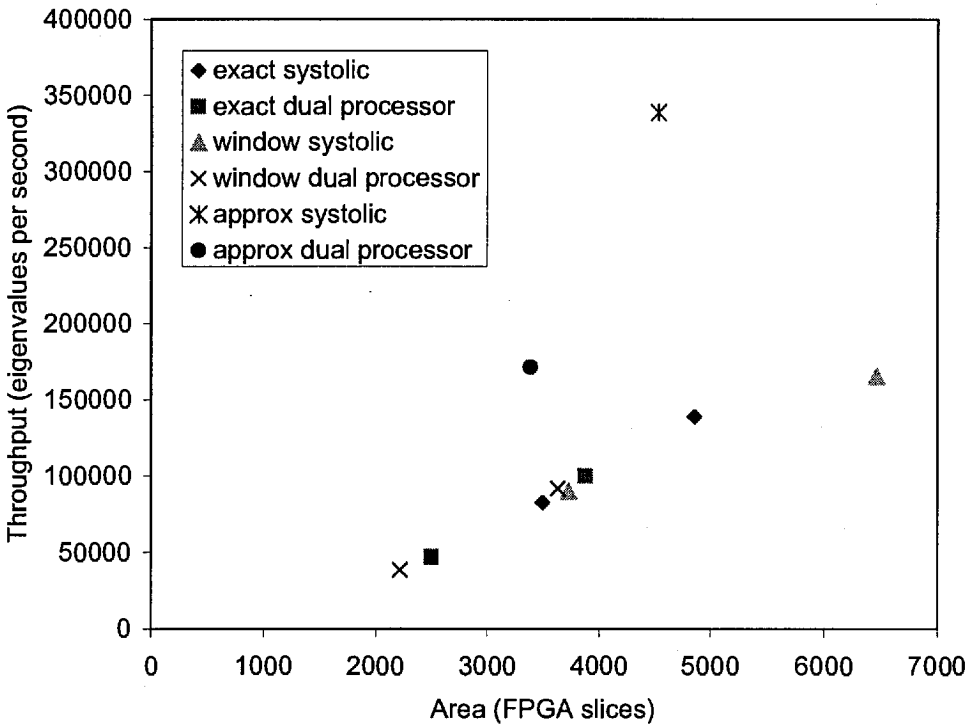


Figure 4.17: Area-Throughput Design Space for 16-bits 4×4 Matrix

To grasp a clear overview of the efficiency of each hardware, we employ an intuitive efficiency measure defined in (4.50). It shows the number of eigenvalue generated every second, given one unit of hardware (one FPGA slice in our implementation).

$$\text{Efficiency} = \frac{\text{Throughput}}{\text{Area}} \tag{4.50}$$

From the efficiency point of view, the *Approximate Jacobi Method* in systolic

architecture has a clear advantage over other methods as demonstrated in Figure 4.18. In contrast to the theory, the *Window Jacobi Method* does not show superiority over the *Exact Jacobi Method* because the speed gained is not substantial enough to justify the extra hardware overhead incurred. In terms of resource sharing, the dual processor architecture offers a better platform than the systolic architecture. This is because for both the *Exact Jacobi Method* and the *Approximate Jacobi Method*, the "dual processor architecture with dedicated CORDIC modules" is slightly more efficient than the "systolic architecture with shared CORDIC modules". The "dual processor architecture with shared CORDIC module" uses less area as shown in Table 4.2, but it is less efficient than others due to a bigger proportion of hardware resource is used for auxiliary function rather than the core eigenvalue computation.

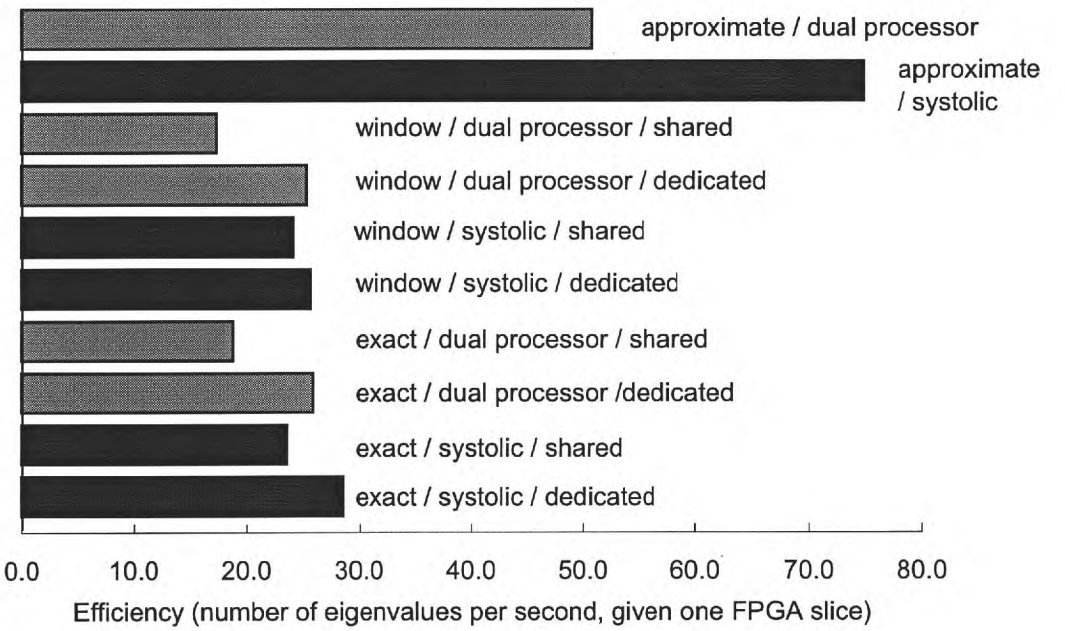


Figure 4.18: Hardware Efficiency comparison for 16-bits 4×4 Matrix

Pipelined architectures for 3×3 symmetric matrices

In very high speed applications, such as real time optical flow computation [Far00], massive throughput of the smallest eigenvalues of 3×3 matrices is required. Since the *Approximate Jacobi Method* is the most efficient one out of the Jacobi eigenvalue framework, we use it to compare with the *Algebraic Method.* Both architectures have been pipelined and designed to return the smallest eigenvalue of a matrix at every clock cycle.

The comparison between the *Algebraic Method* and the *Approximate Jacobi Method* for 13-bits systems is summarized in Table 4.6. The *Approximate Jacobi Method* is unrolled[2] and pipelined, and its behaviour is equivalent to 3 sweeps of its un-pipelined version.

Scheme	Algebraic Method	Approximate Jacobi Method
Area (slices)	4556	20591
Throughput (millions of eigenvalues per second)	62	70
Max % error	13.8×10^{-3}	2.44×10^{-3}

Table 4.6: Comparison of two pipelined architectures in 13-bits accuracy

The results show that the *Algebraic Method* implementation is considerably smaller than the *Approximate Jacobi Method.* However, due to its "open loop" characteristic, its maximum percentage error is 5.65 times that of the *Approximate Jacobi Method.* The pipelined architecture of the *Algebraic Method* is very efficient.

[2]Instead of having one hardware module that computes multiple iterations, multiple hardware modules can be cascaded together, each of which computes one iteration.

4.3 Summary

The first part of the chapter describes Shi and Tomasi algorithm [ST94] for the **2D Feature Detection** module, the pyramidal Lucas-Kanade feature tracker [Bou00] for the **Feature Tracking** module, and the decimation process for the **Adaptive Decision** module.

The second part of the chapter explores the systolic and dual processor architectures for eigenvalue computation based on the *Approximate Jacobi Method*. For comparison purpose, the *Exact Jacobi Method* and *Window Jacobi Method* are also implemented in the two architectures. In terms of area and throughput, it is shown with hardware experimental results that the *Approximate Jacobi Method* in systolic architecture is the most efficient system to compute eigenvalues for $n \times n$ symmetric matrices. For high speed application, a pipelined architecture is proposed resorting to the *Algebraic Method* to compute the eigenvalues for 3×3 symmetric matrices, and successfully achieve very high throughput using only 22% of the area compared to that of the *Approximate Jacobi Method*. A software version of the eigenvalue computation was implemented in C++ on a Pentium 4 PC with 2.4GHz CPU and 1GB of RAM. It had double floating point precision and it used LAPACK library [lap]. Compared to the software version, the throughput of the proposed pipelined architecture is two order of magnitude higher.

Chapter 5

Temporal motion prediction

The first section of this chapter describes the algorithms that participate in the computation of temporal saliency. Out of these algorithms, the Kalman filter is identified as the most computationally intensive one and thus suitable for hardware acceleration. The second part of the chapter concentrates on the hardware exploration of the Kalman filter, which achieves significant performance improvement and constitutes original contribution.

5.1 Algorithms for the modules

This section provides a detailed description of the algorithms that collectively form the **Temporal Motion Prediction** module. The **Temporal Motion Prediction** module consists of an array of **Predictors**. Figure 5.1 shows that one **Predictor** is constructed from three computational blocks, which respectively compute long-term predictability, short-term predictability, and

distribution distance. The operation in the **Suppression** module is also described in this section. In Figure 5.2, the modules within the framework that are relevant to this chapter are highlighted in grey.

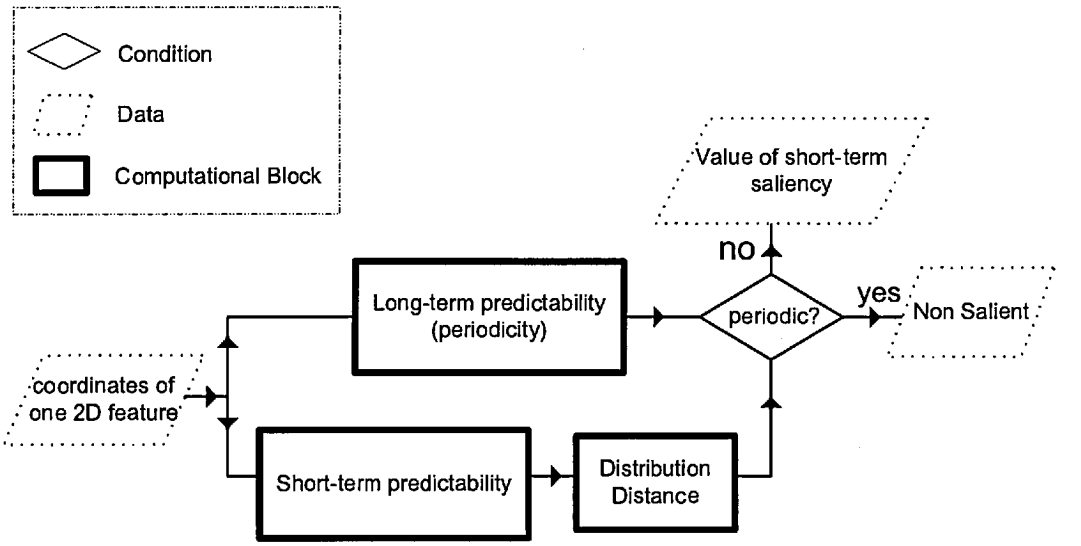


Figure 5.1: The long-term predictability and short-term predictability computation in a **Predictor**

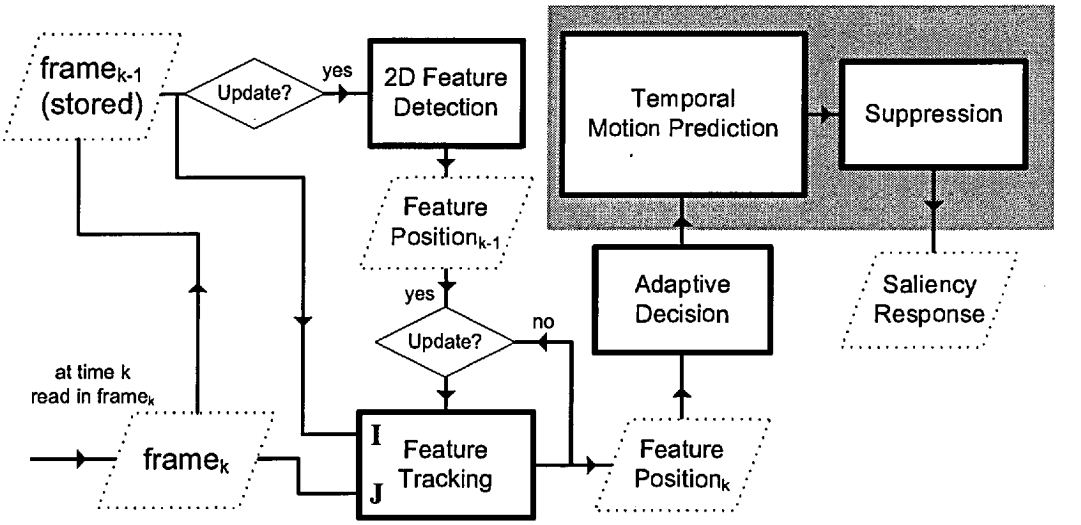


Figure 5.2: Modules in the framework relevant to Chapter 5 are highlighted in grey

5.1.1 Algorithm for a long-term predictor

The long-term predictor determines whether the motion of a 2D feature is periodic in a relatively long interval of time. A periodic motion implies that the movement of the 2D feature has a repetitive pattern, thus its motion is predictable. The long-term predictor is based on autocorrelation coupled with power spectrum analysis [Pri82], which is useful for finding repeated patterns in a signal. The procedure of deciding whether the motion of one 2D feature is periodic is described in this subsection. This procedure is applied to each individual 2D feature independently.

The set of coordinates of a 2D feature in the past N frames is denoted as $\mathbf{S} = [\mathbf{S}_x, \mathbf{S}_y]^T$, where $\mathbf{S}_x$ is the set of x-coordinates and $\mathbf{S}_y$ is the set of y-coordinates. N is the total number of frames to be considered in the periodicity analysis. The frame index f ranges from 1 to N. Hence the coordinates of the 2D feature at frame f are $\mathbf{S}(f) = [\mathbf{S}_x(f), \mathbf{S}_y(f)]^T$. The coordinates of the 2D feature in the first W frames in $\mathbf{S}$ are used as the window $\mathbf{W} = \mathbf{S}(f)$ where f in $[1, W]$. This window $\mathbf{W}$ is shifted by a variable df through $\mathbf{S}$ to generate the autocorrelation profile $\mathbf{A}$ for the 2D feature.

A distance function $Dist$ is defined in (5.1) to measure the distance between two feature coordinates $\mathbf{a} = [a_x, a_y]^T$ and $\mathbf{b} = [b_x, b_y]^T$.

$$Dist(\mathbf{a}, \mathbf{b}) = \text{sign}(a_y - b_y)\,(|a_x - b_x| + |a_y - b_y|) \tag{5.1}$$

$$\text{sign}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$

The distance function in (5.1) is similar to the Manhattan distance except that this distance can be negative depending on the y coordinates of **a** and **b**. This distance function is used in the autocorrelation definition equation in (5.2).

$$\mathbf{A}(df) = \frac{\sum_{f=1}^{W} \left[Dist\big(\mathbf{S}(f), \bar{\mathbf{w}}\big) \cdot Dist\big(\mathbf{S}(f+df), \bar{\mathbf{s}}(df)\big) \right]}{\sqrt{\sum_{f=1}^{W} \left[Dist(\mathbf{S}(f), \bar{\mathbf{w}}) \right]^2 \cdot \sum_{f=1}^{W} \left[Dist\big(\mathbf{S}(f+df), \bar{\mathbf{s}}(df)\big) \right]^2}} \tag{5.2}$$

where

$$\begin{aligned} \bar{\mathbf{w}} &= [\bar{w}_x, \bar{w}_y]^T \\ \bar{w}_x &= \frac{1}{W} \sum_{f=1}^{W} \mathbf{S}_x(f) \\ \bar{w}_y &= \frac{1}{W} \sum_{f=1}^{W} \mathbf{S}_y(f) \end{aligned}$$

$$\begin{aligned} \bar{\mathbf{s}}(df) &= \left[\bar{s}_x(df), \bar{s}_y(df) \right]^T \\ \bar{s}_x(df) &= \frac{1}{W} \sum_{f=1}^{W} \mathbf{S}_x(f+df) \\ \bar{s}_y(df) &= \frac{1}{W} \sum_{f=1}^{W} \mathbf{S}_y(f+df) \end{aligned}$$

Since **S** contains N pairs of x-y coordinates, $f + df$ must be equal to or smaller than N for $\mathbf{S}(f + df)$ to be valid. The maximum value of f is W, thus the maximum allowable df is $N - W$. Therefore, there are $N - W$ autocorrelation valid results.

The power spectrum of $\mathbf{A}$ contains information on the periodicity of the set of coordinates $\mathbf{S}$ of the 2D feature. By employing the Fast Fourier Transform (FFT[1]), the power spectrum $\mathbf{P}$ of $\mathbf{A}$ can be computed as in (5.4) and (5.5).

$$\mathbf{F} = \mathrm{FFT}(\mathbf{A}) \tag{5.4}$$

$$\mathbf{P} = \mathbf{F} \cdot conj(\mathbf{F}) \tag{5.5}$$

where $conj(\mathbf{F})$ denotes the complex conjugate of $\mathbf{F}$.

The maximum value, or the peak, of $\mathbf{P}$ is computed and denoted as $\mathbf{P}_{\text{peak}}$. The mean $\bar{\mathbf{P}}$ of $\mathbf{P}$ is also computed. The peak to mean power ratio $\mathbf{P}_{\text{peak}}/\bar{\mathbf{P}}$ is compared to a threshold P_{threhold}. If the power ratio is less than the threshold, the motion is non-periodic. Since this implies there is no principal frequency that governs the motion of the 2D feature.

On the other hand, if the peak to mean power ratio $\mathbf{P}_{\text{peak}}/\bar{\mathbf{P}}$ is greater than or equal to the threshold P_{threhold}, the frequency $freq_{\text{peak}}$ at which the peak $\mathbf{P}_{\text{peak}}$ resides has to be found. This frequency $freq_{\text{peak}}$ is compared to a threshold $freq_{\text{threshold}}$. If $freq_{\text{peak}}$ is less than the threshold $freq_{\text{threshold}}$, then the motion of the 2D feature is rejected as non-periodic. Otherwise, the motion is periodic. The implication of a small peak frequency is that the principal motion of the 2D feature has a large period, which is out of the detection range.

[1]FFT [Bri88] is an efficient algorithm to compute the Discrete Fourier Transform, which is defined as

$$\mathbf{X}_k = \sum_{n=0}^{N-1} \mathbf{x}_n e^{-\frac{2\pi i}{N} nk} \quad k = 0, ..., N \tag{5.3}$$

Given the frame rate of the video, the biggest period that can be detected is limited by the number of coordinates stored, which is equal to N. Motion with longer period has too many coordinates during one period and only N coordinates are available to the long-term predictor, which are not enough to determine whether the pattern will repeat again or not, i.e. periodic.

The explanation of the periodicity detection process of the long-term predictor can be made clearer with the examples described below. In these examples, the frame rate is assumed to be 30 frames/second. N is set to be 150 and W is set to be 30. This implies that the coordinates in the past 5 seconds are stored and the process is tuned to detect motion with period of 1 second. The peak to mean power ratio threshold $P_{\text{threshold}} = 20$ and the frequency threshold $freq_{\text{theshold}} = 0.5\text{Hz}$. A zero mean Gaussian noise with variance of one is added to the coordinates of the 2D feature to check the robustness. These assumptions are made for proof of concept and do not pose limitations on the long-term predictor.

Periodic circular motion

In this example, the 2D feature undergoes a **periodic circular** motion. To represent the path of the 2D feature, the coordinates of the 2D feature in the past N frames are linked as shown in Figure 5.3. It takes the 2D feature 1 second to complete one circular motion, i.e. the frequency is 1Hz. Hence in 5 seconds, the 2D feature completes the circular motion 5 times albeit following a slightly different path each time due to the noise added.

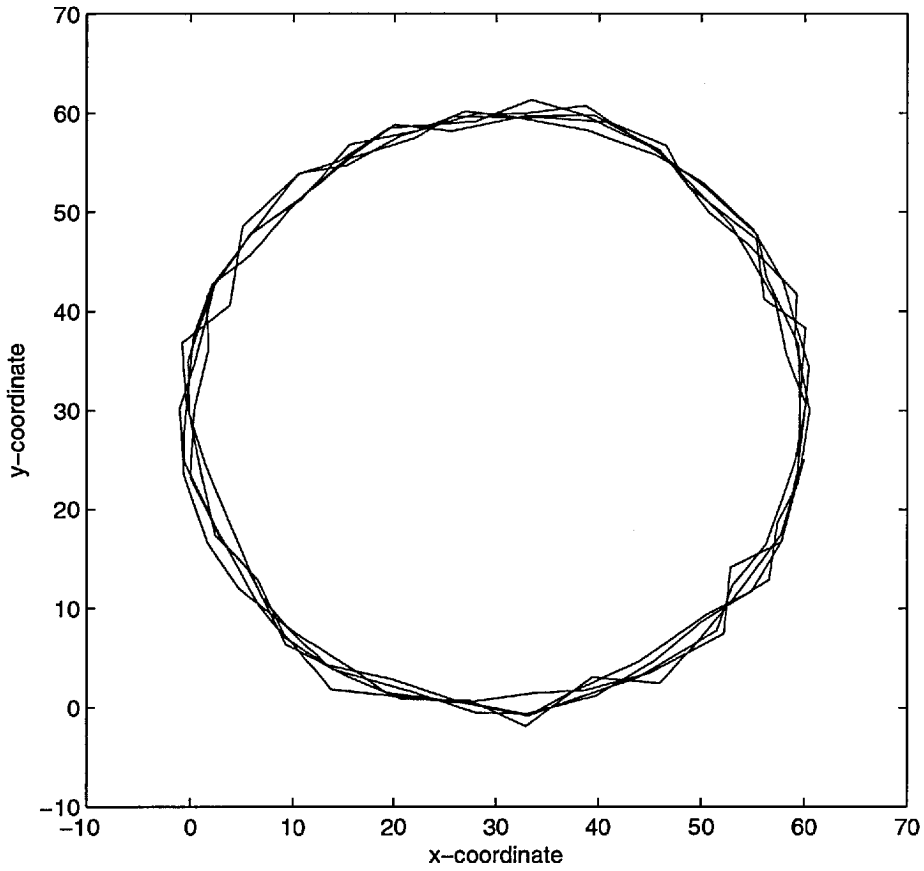


Figure 5.3: The path of 2D feature in the **periodic circular** motion

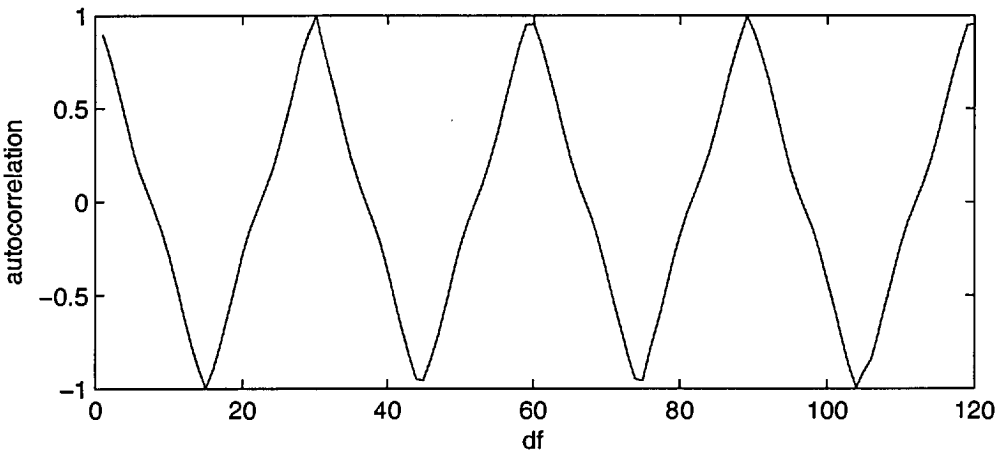


Figure 5.4: The autocorrelation of the coordinates of the 2D feature in the **periodic circular** motion

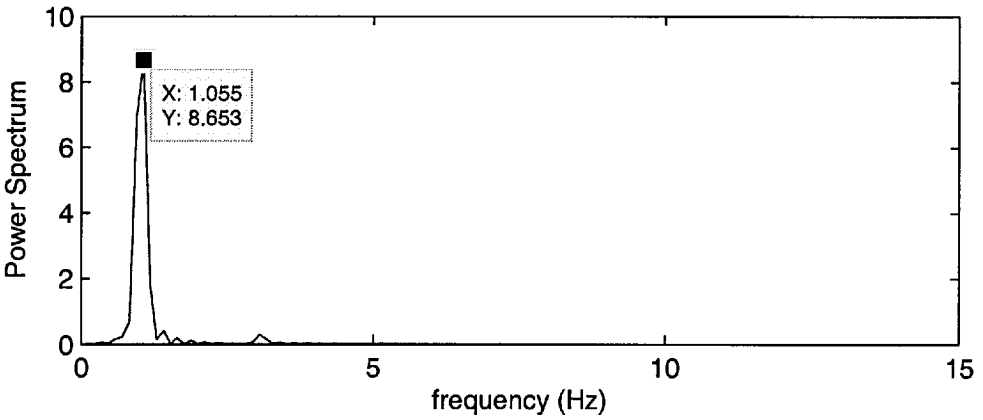


Figure 5.5: The power spectrum of the autocorrelation for the 2D feature in the **periodic circular** motion

The autocorrelation of the N coordinates are computed according to the definition in (5.2). The autocorrelation results are shown in Figure 5.4. There are $N - W = 120$ results for the reason explained after the definition of the autocorrelation.

The power spectrum of the autocorrelation is shown in Figure 5.5. The peak to mean power ratio $\mathbf{P}_{\text{peak}}/\bar{\mathbf{P}}$ is computed to be 53.9, which is greater than the threshold of 20. The high peak to mean power ratio indicates the motion has a dominant frequency. Moreover, the frequency at which the power peak resides is detected to be 1.055Hz, which is also greater than the frequency threshold 0.5Hz. Therefore, the motion of the 2D feature within the past N frames is determined to be periodic. Notice the detected peak frequency, 1.055Hz, is in line with the actual frequency of the motion, which is equal to 1Hz. This demonstrates that the long-term predictor successfully detects the dominant frequency of the motion in this example.

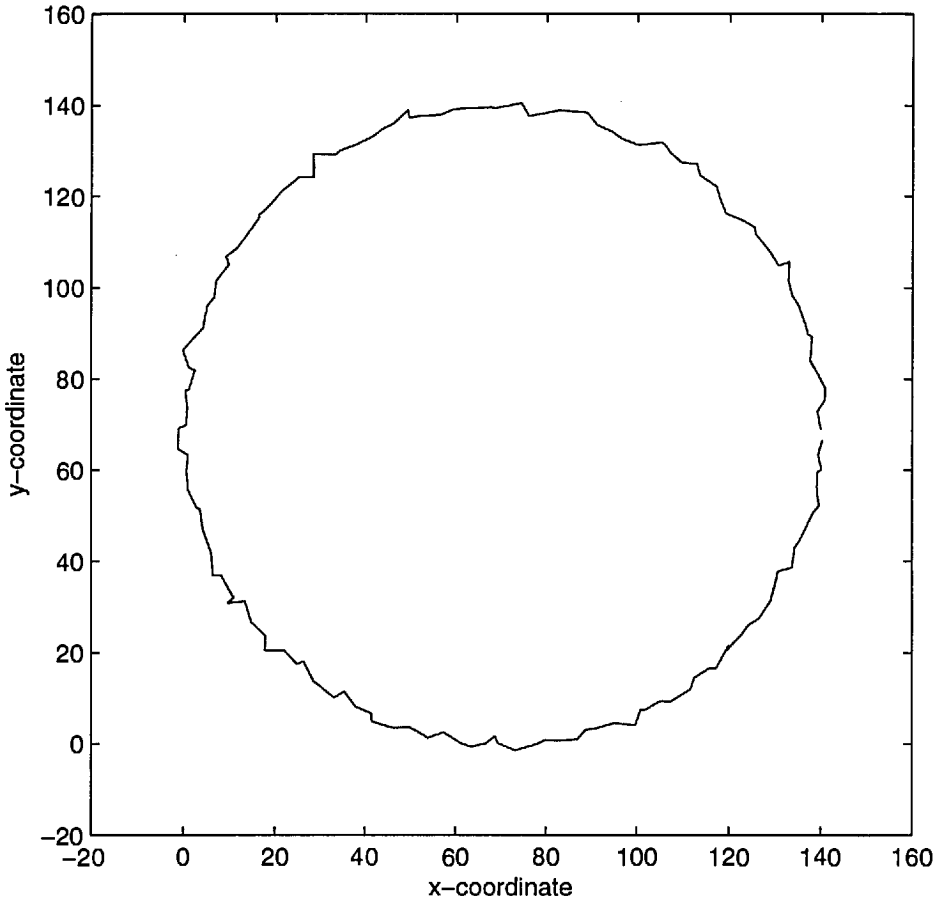


Figure 5.6: The path of 2D feature in the **non-periodic circular** motion

Non-periodic circular motion

In this example, the 2D feature undergoes a **non-periodic circular** motion. It takes the 2D feature 5 seconds to complete one circular motion and the path is shown in Figure 5.6. Since the long-term predictor takes into account the 2D feature's coordinates in the past 5 seconds, given the available information, one circular motion should be determined to be non-periodic.

The autocorrelation is shown in Figure 5.7. The power spectrum of the autocorrelation is shown in Figure 5.8. The peak to mean power ratio $\mathbf{P}_{\text{peak}}/\bar{\mathbf{P}}$

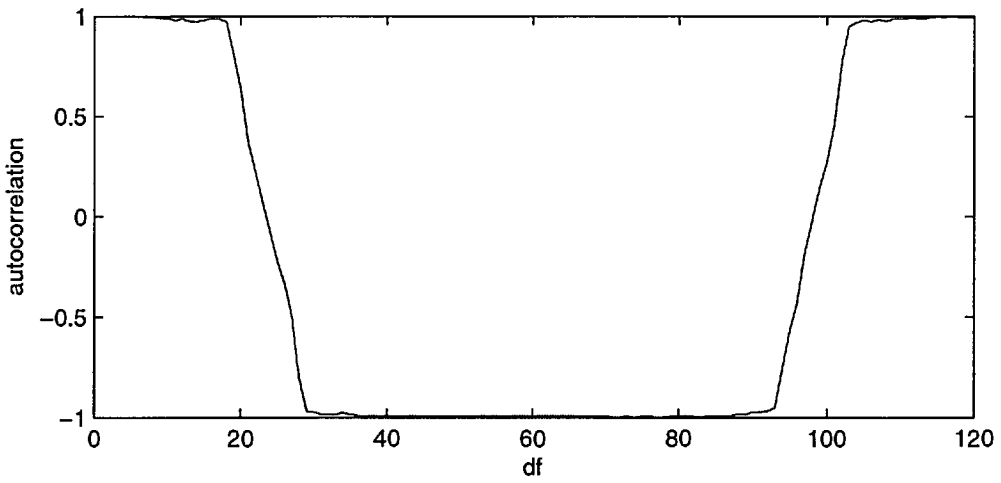


Figure 5.7: The autocorrelation of the coordinates of the 2D feature in the **non-periodic circular** motion

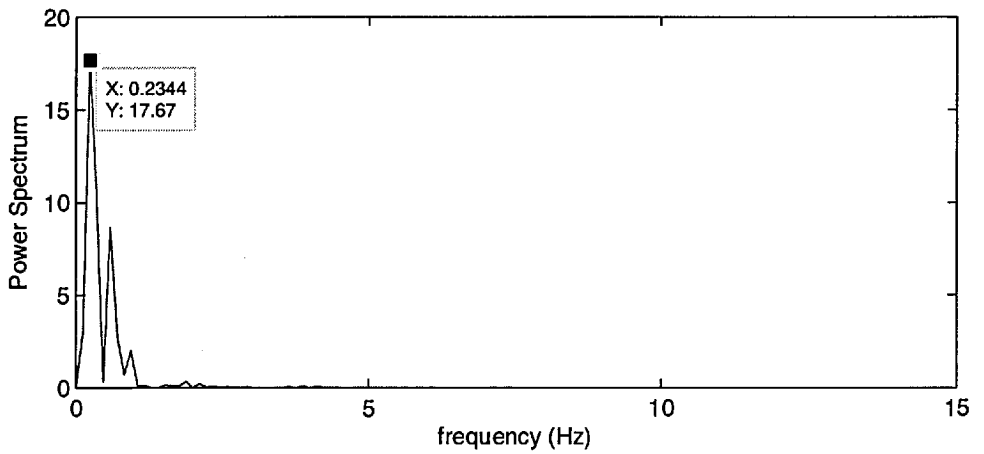


Figure 5.8: The power spectrum of the autocorrelation for the 2D feature in the **non-periodic circular** motion

is computed to be 47.15, which is greater than the threshold of 20. However, the frequency at which the power peak resides is detected to be 0.23Hz, which is smaller than the frequency threshold 0.5Hz. Therefore, the motion of the 2D feature within the past N frames is determined to be non-periodic.

Non-periodic random motion

In this example, a 2D feature undergoes a **non-periodic random** motion. The random path is shown in Figure 5.9. The autocorrelation is shown in Figure 5.10. The power spectrum of the autocorrelation is shown in Figure 5.11. The peak to mean power ratio $\mathbf{P}_{\text{peak}}/\bar{\mathbf{P}}$ is computed to be 13.16, which is less than the threshold of 20. This indicates the motion does not have a dominant period. Therefore, the motion of the 2D feature within the past N frames is determined to be non-periodic.

Periodic triangular motion

In this example, a 2D feature undergoes a **periodic triangular** motion. The triangular path is shown in Figure 5.12. It takes the 2D feature 1 second to complete one circular motion, i.e. the frequency is 1Hz. Hence in 5 seconds, the 2D feature completes the triangular motion 5 times albeit following a slightly different path each time due to the noise added. The autocorrelation is shown in Figure 5.13. The power spectrum of the autocorrelation is shown in Figure 5.14. The peak to mean power ratio $\mathbf{P}_{\text{peak}}/\bar{\mathbf{P}}$ is computed to be 51.38, which is greater than the threshold of 20. Moreover, the frequency at which the power

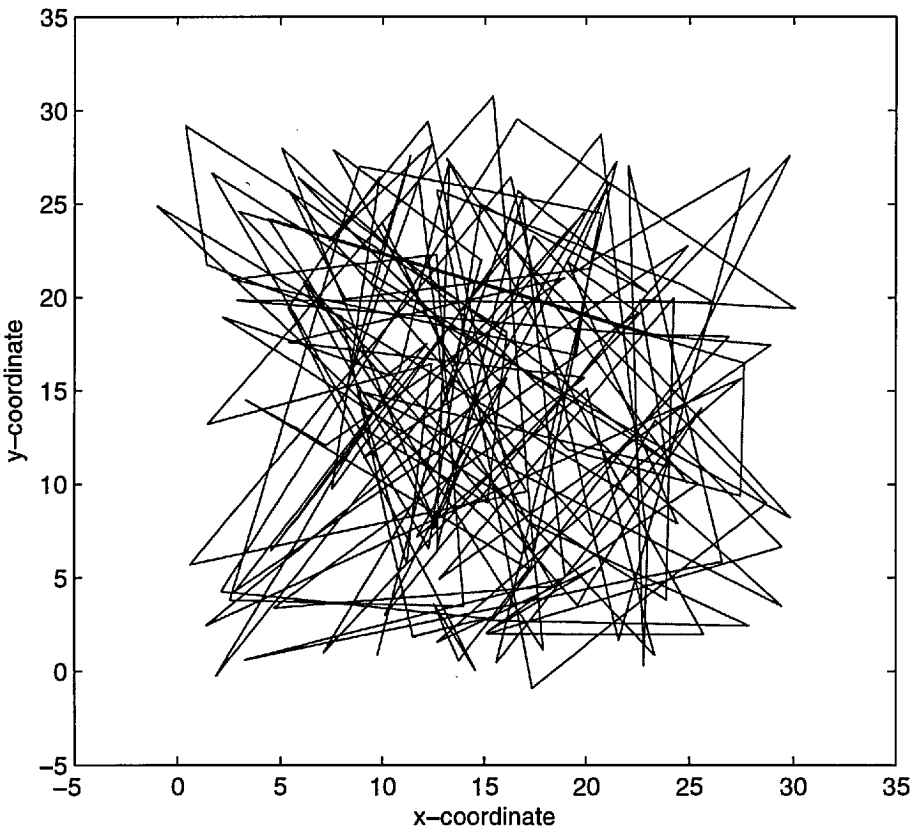


Figure 5.9: The path of 2D feature in the **non-periodic random** motion

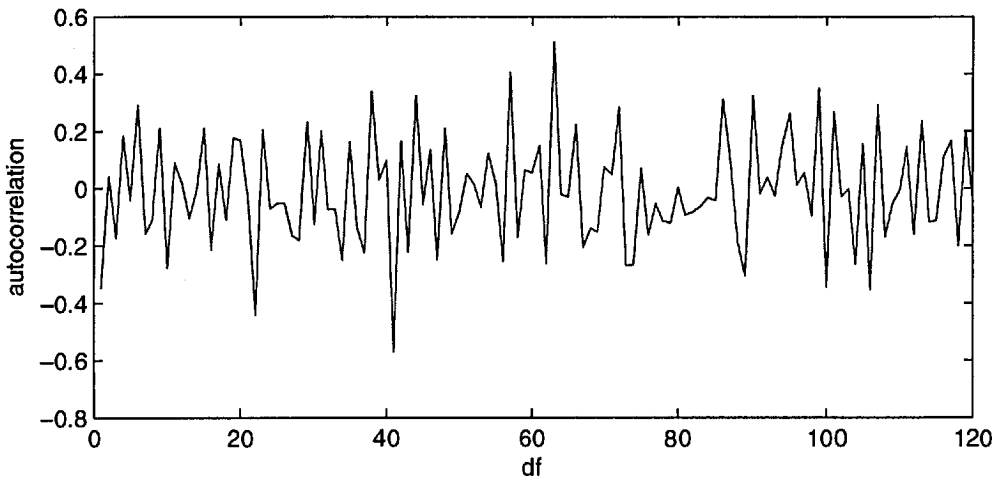


Figure 5.10: The autocorrelation of the coordinates of the 2D feature in the **non-periodic random** motion

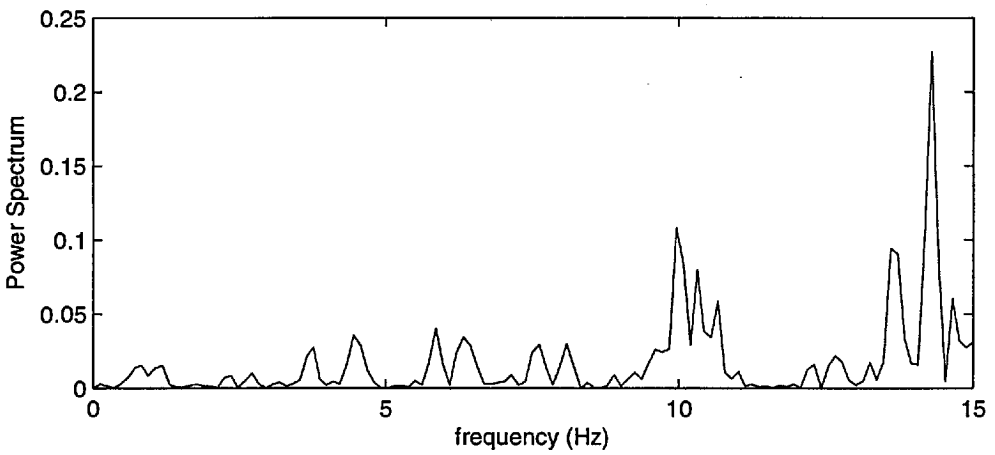


Figure 5.11: The power spectrum of the autocorrelation for the 2D feature in the **non-periodic random** motion

peak resides is detected to be 1.055Hz, which is also greater than the frequency threshold 0.5Hz. Therefor, the motion of the 2D feature within the past N frames is determined to be periodic.

Periodic back-and-forth motion

In this example, the 2D feature undergoes a **periodic back-and-forth** motion. The 2D feature travels between two points back and forth. The path is shown in Figure 5.15. It takes the 2D feature 1 second to complete one back-and-forth turn, i.e. the frequency is 1Hz. Hence in 5 seconds, the 2D feature completes 5 turns albeit following a slightly different path each time due to the noise added. The autocorrelation is shown in Figure 5.16. The power spectrum of the autocorrelation is shown in Figure 5.17. The peak to mean power ratio $\mathbf{P}_{\text{peak}}/\bar{\mathbf{P}}$ is computed to be 53.6, which is greater than the threshold of 20. Moreover, the frequency at which the power peak resides is detected to be 1.055Hz, which is also greater than the frequency threshold

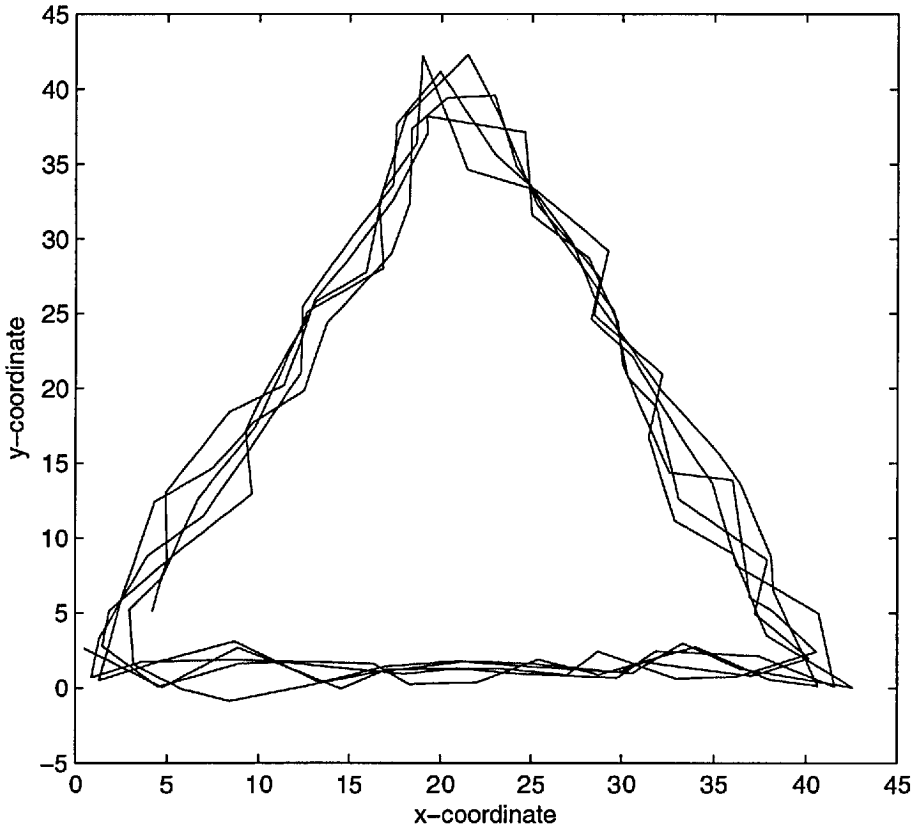


Figure 5.12: The path of 2D feature in the **periodic triangular** motion

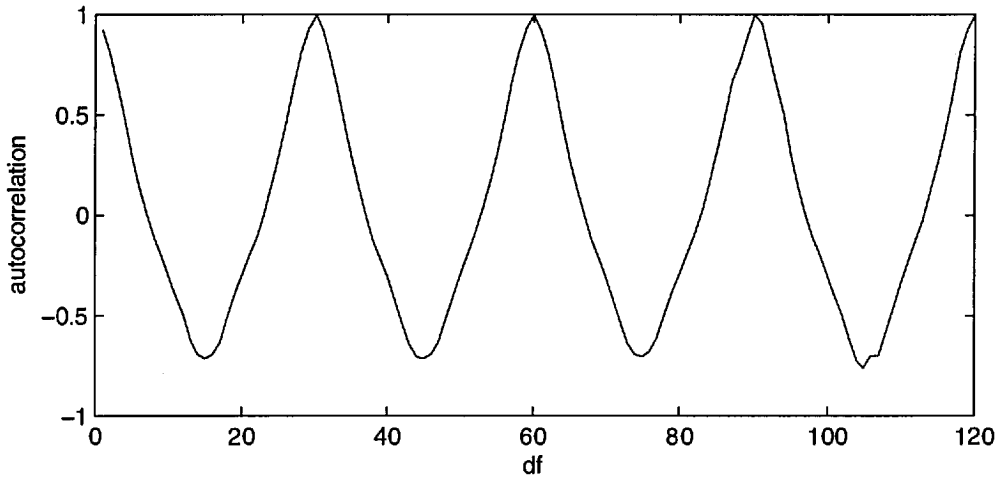


Figure 5.13: The autocorrelation of the coordinates of the 2D feature in the **periodic triangular** motion

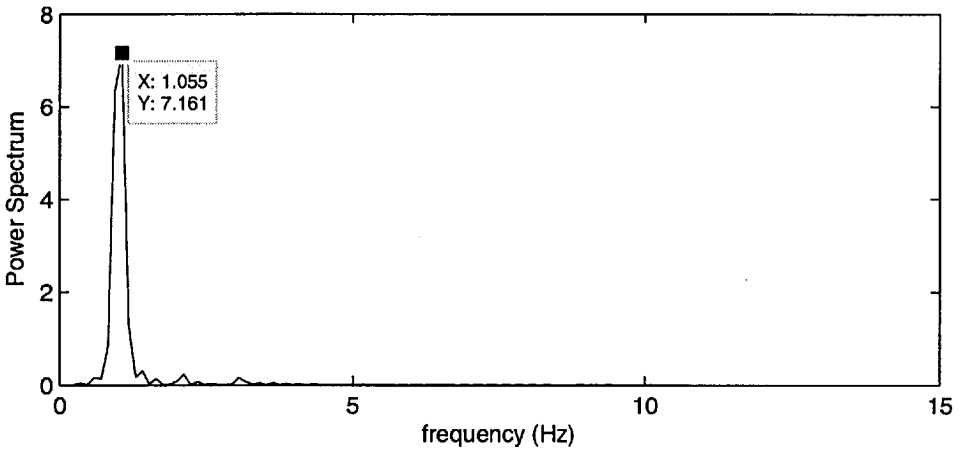


Figure 5.14: The power spectrum of the autocorrelation for the 2D feature in the **periodic triangular** motion

0.5Hz. Therefor, the motion of the 2D feature within the past N frames is determined to be periodic.

Non-periodic linear motion

In this example, the 2D feature undergoes a **non-periodic linear** motion. It takes 5 seconds for the 2D feature travels from one point to another on a linear path. The path is shown in Figure 5.18. Since the long-term predictor takes into account the 2D feature's coordinates in the past 5 seconds, given the available information, this motion should be determined to be non-periodic.

The autocorrelation is shown in Figure 5.19. All results are very close to 1. The power spectrum of the autocorrelation is shown in Figure 5.20. The peak to mean power ratio $\mathbf{P}_{\text{peak}}/\bar{\mathbf{P}}$ is computed to be 120, which is greater than the threshold of 20. However, the frequency at which the power peak resides is detected to be 0Hz, which is smaller than the frequency threshold 0.5Hz. Therefore, the motion of the 2D feature within the past N frames is

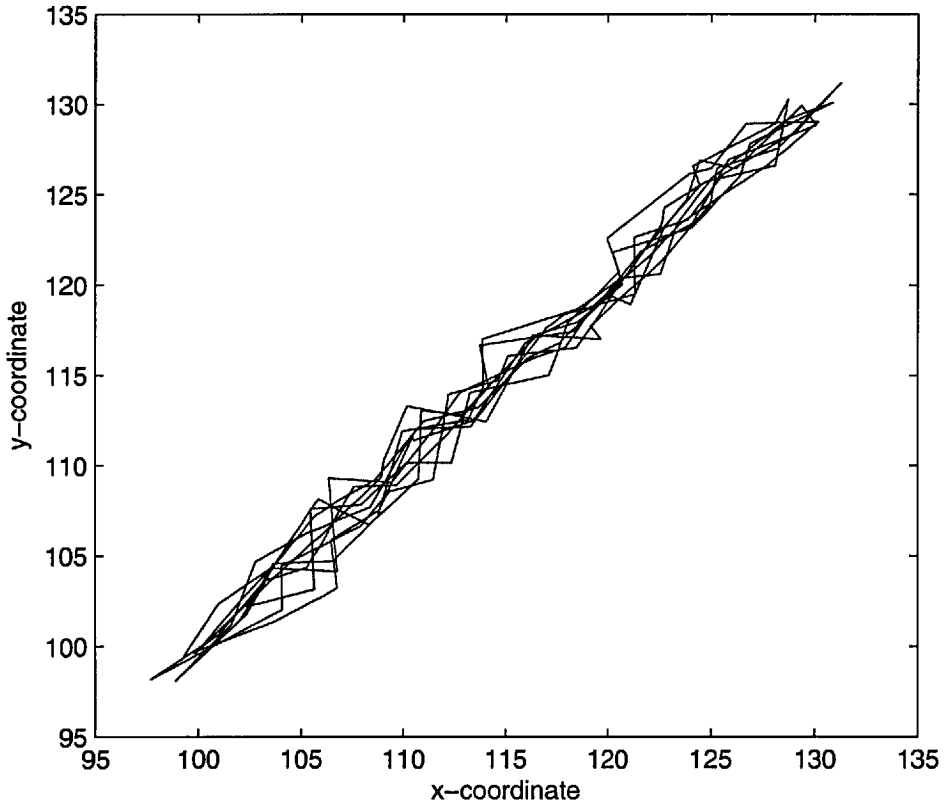


Figure 5.15: The path of 2D feature in the **periodic back-and-forth** motion

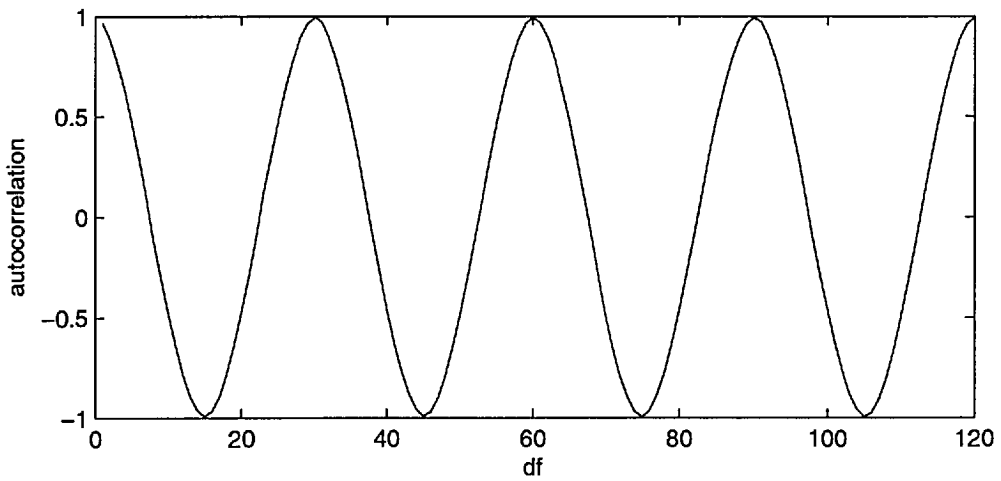


Figure 5.16: The autocorrelation of the coordinates of the 2D feature in the **periodic back-and-forth** motion

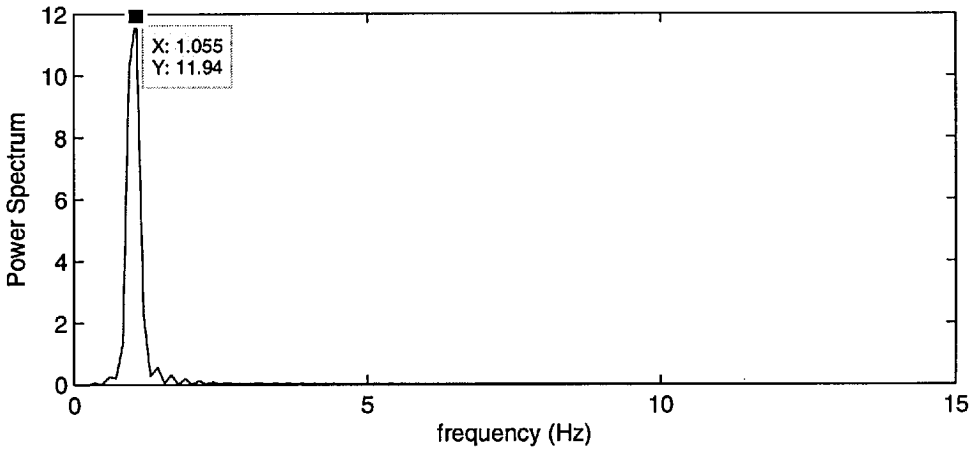


Figure 5.17: The power spectrum of the autocorrelation for the 2D feature in the **periodic back-and-forth** motion

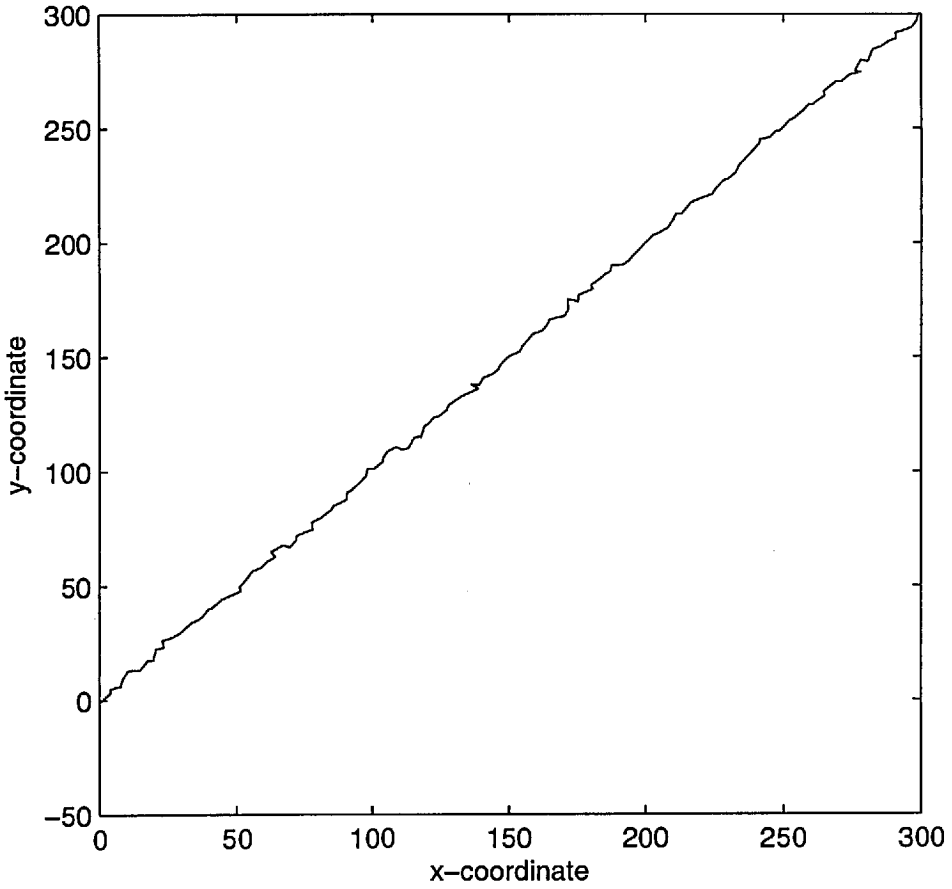


Figure 5.18: The path of 2D feature in the **non-periodic linear** motion

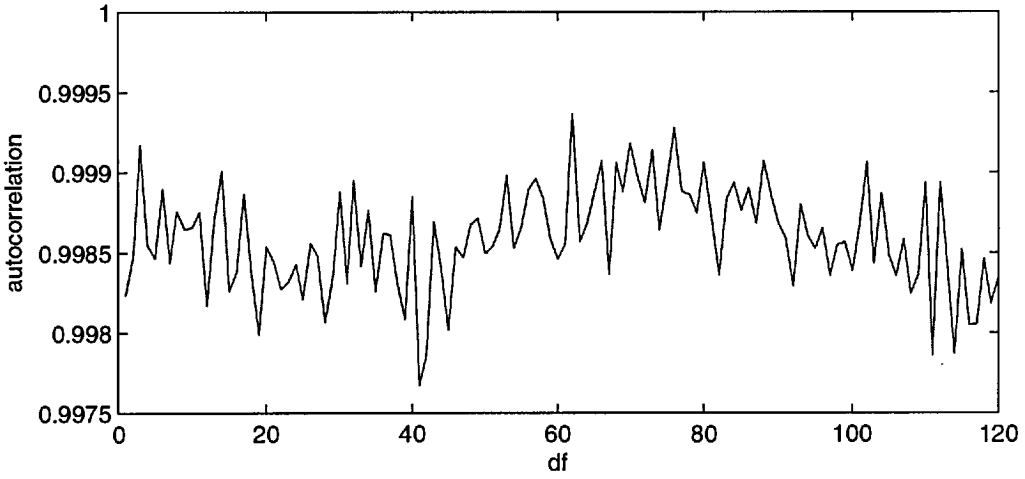


Figure 5.19: The autocorrelation of the coordinates of the 2D feature in the **non-periodic linear** motion

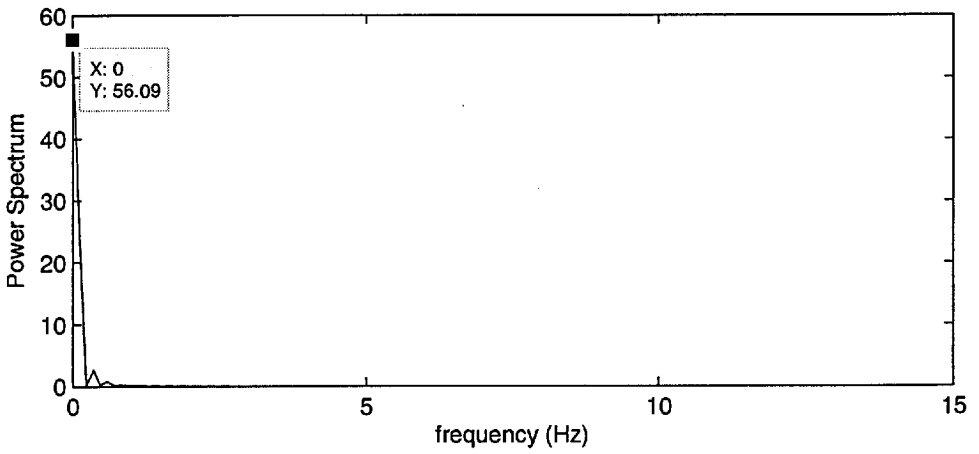


Figure 5.20: The power spectrum of the autocorrelation for the 2D feature in the **non-periodic linear** motion

determined to be non-periodic.

The purpose of the long-term predictor is to determine whether the motion of a 2D feature is periodic in a relatively long interval of time. As opposed to the long-term predictor, the short-term predictor quantifies how well the coordinates of the 2D feature in the upcoming frame can be predicted.

5.1.2 Algorithm for short-term predictor and distribution distance

The algorithm used for the short-term predictor is the Kalman filter. The Kalman filter is a recursive estimator that provides the best estimation, in a least-square error sense, of the "hidden" information of interest from noisy measurements. It was firstly introduced in the seminal papers [Kal60, KB61]. One of the first public known applications was made at NASA to guide the Apollo 11 lunar module to the moon's surface. Since then, the Kalman Filter has found its way to become an indispensable part of many applications ranging from signal processing to communication and control.

Basics of the Kalman filter

It is not always feasible to measure all the variables of a dynamic system. Kalman filter provides a way to infer the missing variables, or states, from noisy measurements. It can also be used to predict future states taking into account all the measurements in the past. It must be noticed that Kalman filter is a *recursive* filter. The model is updated when a new measurement

is available. However, it does not need to store all the measurements. A formulation of the Kalman filter is shown in (5.6).

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{F}\mathbf{x}_k + \mathbf{w}_k \\ \mathbf{z}_k &= \mathbf{H}\mathbf{x}_k + \mathbf{v}_k \end{aligned} \tag{5.6}$$

$\mathbf{x}_k$ is the n-dimensional state vector and $\mathbf{z}_k$ is the m-dimensional measurement vector at time k. The state $\mathbf{x}_k$ evolves in time and the transition is governed by the transition matrix $\mathbf{F}$, which is $n \times n$ in size. The $m \times n$ matrix $\mathbf{H}$ is called the measurement matrix and relates the measurements to the states. The system is corrupted by two types of noise. $\mathbf{w}$ is the n-dimensional process noise vector and $\mathbf{v}$ is the m-dimensional measurement noise vector. The noise vectors are assumed to be independent Gaussian processes with zero mean and covariance matrices $\mathbf{Q}$ and $\mathbf{R}$, respectively.

The Kalman filter assumes that the probability density of the states at every time step is Gaussian and hence exactly and completely characterized by two parameters, its mean and covariance [AMGC02]. The means are represented by the state vector $\mathbf{x}_k$ and the variances are represented by the estimation error covariance $\mathbf{P}_k$. The task of a Kalman filter is to estimate or predict the state vector $\mathbf{x}_k$ from noise-perturbed measurements $\mathbf{z}_k$. At time k, given a new measurement vector $\mathbf{z}_k$, a Kalman filter computes the optimal prediction of the state vector $\hat{\mathbf{x}}_{k+1} = \hat{\mathbf{x}}_{k+1|k}$ based on known measurement vectors $\{\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_k\}$.

Kalman filter can be interpreted as having two stages: *measurement update* and *prediction*. In the *measurement update* stage, the parameters of the Kalman filter are updated using (5.7) to (5.9).

$$\mathbf{K}_k = \mathbf{P}_k^- \mathbf{H}^T [\mathbf{H}\mathbf{P}_k^- \mathbf{H}^T + \mathbf{R}]^{-1} \tag{5.7}$$

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + \mathbf{K}_k(\mathbf{z}_k - \mathbf{H}\hat{\mathbf{x}}_k^-) \tag{5.8}$$

$$\mathbf{P}_k = [\mathbf{I} - \mathbf{K}_k\mathbf{H}]\mathbf{P}_k^-[\mathbf{I} - \mathbf{K}_k\mathbf{H}]^T + \mathbf{K}_k\mathbf{R}_k\mathbf{K}_k^T \tag{5.9}$$

$\mathbf{K}_k$ is the Kalman gain, $\hat{\mathbf{x}}_k^-$ is the *a priori* state vector, $\hat{\mathbf{x}}_k$ is the *a posteriori* state vector, $\mathbf{P}_k^-$ is the *a priori* state estimation error covariance matrix and $\mathbf{P}_k$ is the *a posteriori* state estimation error covariance matrix. The Kalman gain in (5.7) is the statically optimal gain. This Kalman gain minimizes $E\{\|\mathbf{x}_k - \hat{\mathbf{x}}_k\|^2\}$, which is the expected value of the square of the magnitude of the error in the posterior state estimation. By using the optimal gain in (5.7), the update of the state estimation error covariance matrix in (5.9) can be simplified as in (5.10).

$$\mathbf{P}_k = [\mathbf{I} - \mathbf{K}_k\mathbf{H}]\mathbf{P}_k^- \tag{5.10}$$

In the *prediction* stage, the future states are predicted by using the projection equations:

$$\text{Project state:} \qquad \hat{\mathbf{x}}_k^- = \mathbf{F}\hat{\mathbf{x}}_{k-1} \tag{5.11}$$

$$\text{Project covariance:} \quad \mathbf{P}_k^- = \mathbf{F}\mathbf{P}_{k-1}\mathbf{F}^T + \mathbf{Q} \tag{5.12}$$

More details on the derivations of the Kalman filter equations (5.7) to (5.12) can be found in [GA01]. The definitions of the symbols used in these equations are summarized as follow:

$\hat{\mathbf{x}}_k^-$: *a priori* state vector

$\hat{\mathbf{x}}_k$: *a posteriori* state vector

$\mathbf{P}_k^-$: *a priori* estimation error covariance

$\mathbf{P}_k$: *a posteriori* estimation error covariance

$\mathbf{F}$: transition matrix

$\mathbf{H}$: measurement matrix

$\mathbf{Q}$: process noise covariance

$\mathbf{R}$: measurement noise covariance

Kalman filter as the short-term predictor

In our proposed saliency framework, the Kalman filter serves as the short-term predictor. The coordinates of a 2D feature from the feature tracker are the measurements and the probability distributions of possible coordinates and velocities are the states. A *linear* motion model is assumed, thus *linear* motions are predictable and temporally non-salient. On the other hand, *nonlinear* motions that cannot be predicted are temporally salient. The transition matrix $\mathbf{F}$, state vector $\mathbf{x}$ and measurement matrix $\mathbf{H}$ are set to reflect these assumptions.

$$\mathbf{F} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x \\ y \\ x' \\ y' \end{bmatrix}$$

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

x and y are the means of the Gaussian probability density of the coordinates in x-direction and y-direction respectively. x' and y' are the means of the Gaussian probability density of the velocity components in x and y direction respectively. The Kalman filter noise covariance matrices $\mathbf{Q}$ and $\mathbf{R}$ are both set to be identity matrices, which implies that the position and speed variables are treated as uncorrelated, and the process noise variances and the measurement noise variances are assumed to be 1.

At time k, there is the *a priori* state vector $\hat{\mathbf{x}}_k^-$ and the *a priori* estimation error covariance $\mathbf{P}_k^-$ that are both "predicted" from the previous time step $k-1$ using (5.11) and (5.12). A measurement vector $\mathbf{z}_k$ that contains new coordinates of the 2D feature is available at time k. This is used to update the Kalman filter parameters using (5.7), (5.8) and (5.10). After the update, the *a posteriori* state vector $\hat{\mathbf{x}}_k$ and the *a posteriori* estimation error covariance $\mathbf{P}_k$ are obtained. These *a posteriori* statistics incorporate the new measurement vector $\mathbf{z}_k$, whereas the *a priori* statistics are "predicted" without the knowledge

of $\mathbf{z}_k$. Therefore, the distance between *a priori* and *a posteriori* statistics quantify how accurate the prediction is.

Bhattacharyya distribution distance

Since the comparison of *a priori* and *a posteriori* statistics is made between two probability distributions, a distribution distance measure is required. Bhattacharyya distance [Kai67] is used as the distribution distance to quantify saliency from the probability distributions constructed by the short-term predictor. It is a symmetric measure of the distance between two probability distributions $p_p(x)$ and $p_q(x)$ and it is defined as:

$$B = -\log\left(\int_{-\infty}^{+\infty}\sqrt{p_p(x)p_q(x)}dx\right) \tag{5.13}$$

The Kalman Filter, as the short-term predictor, assumes that the probability density function of the 2D feature coordinates follows a Gaussian distribution. The Bhattacharyya distance of two univariate Gaussian distributions p and q is given by

$$B_{Gaussian}(q;p) = \frac{1}{4}\frac{(\mu_p-\mu_q)^2}{\sigma_p^2+\sigma_q^2} + \frac{1}{2}\log\left(\frac{\sigma_p^2+\sigma_q^2}{2\sigma_p\sigma_q}\right) \tag{5.14}$$

where μ_p and μ_q are the mean values and σ_p and σ_q are the variances of the distributions p and q, respectively. The first term in (5.14) accounts for the difference due to the mean, and the second term gives the separability caused by the variance. The Bhattacharyya distance (5.14) for Gaussian distribution

can be employed to compute the distance between the *a priori* and *a posteriori* distributions at time k. The distances are computed in x and y directions separately as in (5.15) and (5.16), then the distances in two directions are combined in (5.17).

$$B_k^x = \frac{1}{4}\frac{(\hat{x}_k^- - \hat{x}_k)^2}{(p_k^{x-})^2 + (p_k^x)^2} + \frac{1}{2}\log\left(\frac{(p_k^{x-})^2 + (p_k^x)^2}{2p_k^{x-}p_k^x}\right) \quad (5.15)$$

$$B_k^y = \frac{1}{4}\frac{(\hat{y}_k^- - \hat{y}_k)^2}{(p_k^{y-})^2 + (p_k^y)^2} + \frac{1}{2}\log\left(\frac{(p_k^{y-})^2 + (p_k^y)^2}{2p_k^{y-}p_k^y}\right) \quad (5.16)$$

$$B_k = \sqrt{(B_k^x)^2 + (B_k^y)^2} \quad (5.17)$$

where

$\hat{x}_k^-$ the x-coordinate mean in the *a priori* state vector $\hat{\mathbf{x}}_k^-$

$\hat{y}_k^-$ the y-coordinate mean in the *a priori* state vector $\hat{\mathbf{x}}_k^-$

$\hat{x}_k$ the x-coordinate mean in the *a posteriori* state vector $\hat{\mathbf{x}}_k$

$\hat{y}_k$ the y-coordinate mean in the *a posteriori* state vector $\hat{\mathbf{x}}_k$

$(p_k^{x-})^2$ the variances of x-coordinates in the *a priori* estimation error covariance $\mathbf{P}_k^-$

$(p_k^{y-})^2$ the variances of y-coordinates in the *a priori* estimation error covariance $\mathbf{P}_k^-$

$(p_k^x)^2$ the variances of x-coordinates in the *a posteriori* estimation error covariance $\mathbf{P}_k$

$(p_k^y)^2$ the variances of y-coordinates in the *a posteriori* estimation error covariance $\mathbf{P}_k$

B_k in (5.17) is used to deduce the short-term unpredictability of a 2D

feature. Therefore, one scalar value is generated for one 2D feature at time k as its short-term temporal saliency value. The combination of the Kalman filter and the Bhattacharyya distance to compute the short-term temporal saliency is demonstrated with the following examples.

Examples of the short-term predictor

In Figure 5.21, the measurements of the Kalman filter are the 2D feature coordinates obtained by the feature tracker. The 2D feature travels from bottom left to top right approximately in a straight line at constant velocity. The *a priori* means of coordinates are represented in crosses in Figure 5.21. It can be seen that these predicted values are close to the measurements. It should be noted that the measurements are in future time steps with respect to the *a priori* means. Hence, the *a priori* means are the predicted values by the Kalman filter. The *a posteriori* means of coordinates are rendered after the Kalman filter is updated with new measurements.

The corresponding Bhattacharyya distance between the *a priori* and the *a posteriori* coordinate distributions are shown in Figure 5.22. This distance metric is used to represent the temporal saliency of the 2D feature. The x-axis of Figure 5.22 is chosen to be the same as that of Figure 5.21 in order to facilitate correspondence. The second coordinates have large Bhattacharyya distance value. This is because the Kalman filter is initialized without any knowledge of the velocity of the 2D feature, and it needs a few measurements to infer these missing states.

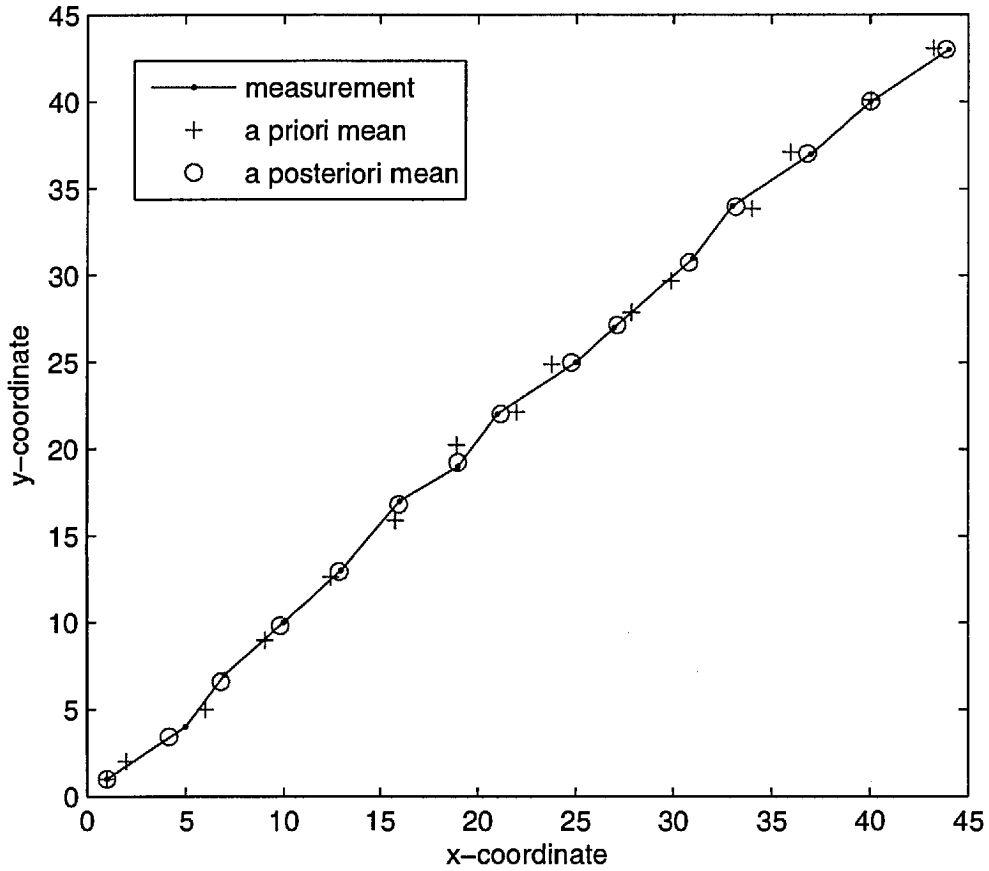


Figure 5.21: The Kalman filter variables for the 2D feature in a linear motion

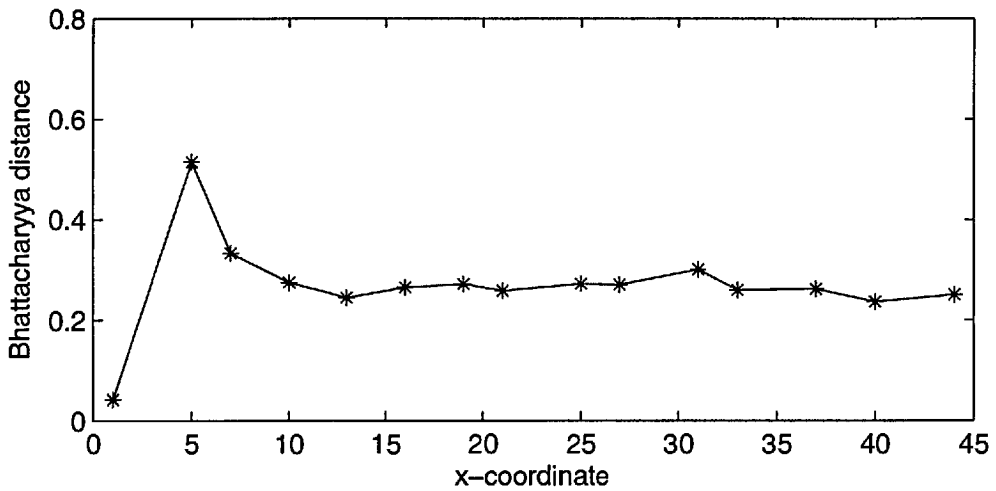


Figure 5.22: The Bhattacharyya distance for the 2D feature in a linear motion

In contrast to the last example, Figure 5.23 shows a 2D feature travels in an unpredictable fashion, which changes course several times. As a result, the *a priori* means of coordinates are less able to predict the upcoming measurements. The Bhattacharyya distance between the *a priori* and the *a posteriori* coordinate distributions are shown in Figure 5.24. The peaks of the Bhattacharyya distance can be associated with the unpredictable change of movement direction of the 2D feature. Such response to unpredictability in motion demonstrates the capability of the Bhattacharyya distance to represent the temporal saliency of the 2D feature.

5.1.3 Algorithm for suppression

The proposed framework renders a sparse saliency map. Only pixels that are associated with 2D features have saliency value, while the rest of the pixels do not. However, not all the 2D features will be present in the saliency map. The short-term temporal saliency values of the 2D features are passed to the **Suppression** module, where selection is carried out.

The calculated saliency values are firstly subject to a threshold process. The 2D features with saliency values less than the threshold are screened out. The neighboring pixels of the remaining 2D features are then investigated. A salient 2D feature remains valid and advances into the final saliency map if there is sufficient number of other 2D features within a predefined circular radius. If this number is set to 2, one single 2D feature isolated from the rest is disqualified. It is likely that this 2D feature is caused by "salt and pepper"

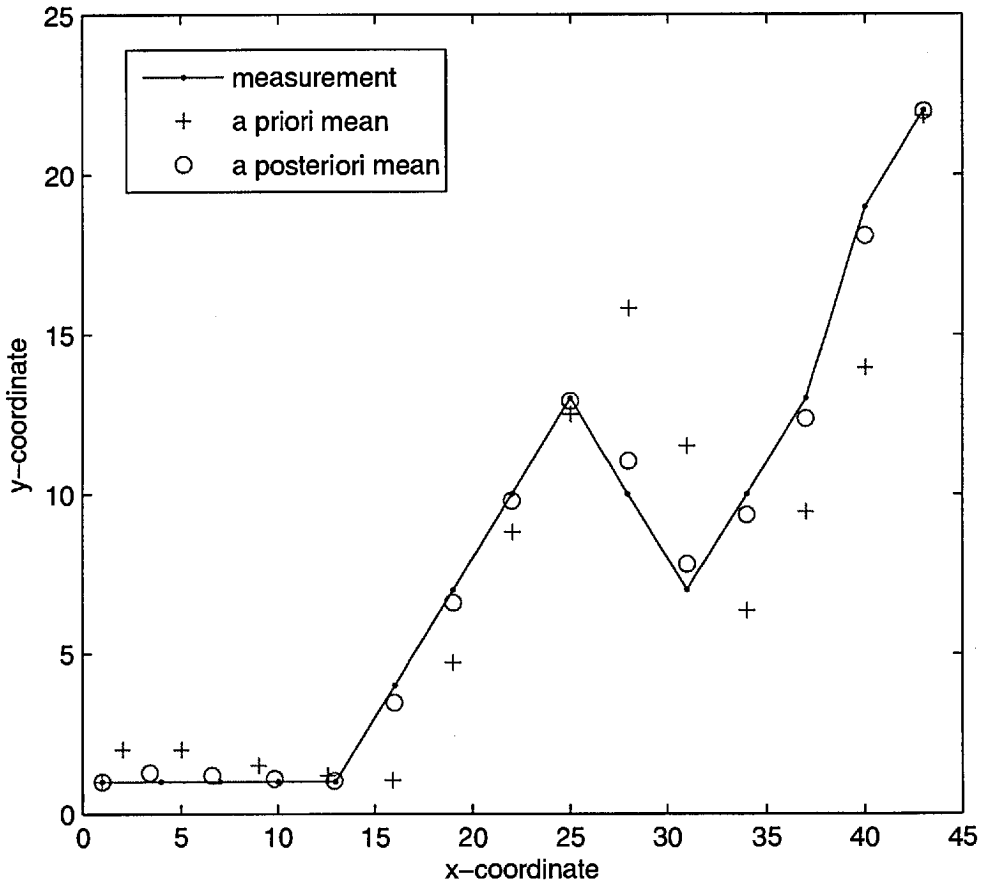


Figure 5.23: The Kalman filter variables for the 2D feature in an unpredictable motion

type of noise. Under this suppression scheme, only clusters of salient features remain and outliers are removed. The suppressor has the effect of disentangling noise from genuine salient movement by relying on multiple saliency responses in a region. Figure 5.25 gives an illustration of the suppression process.

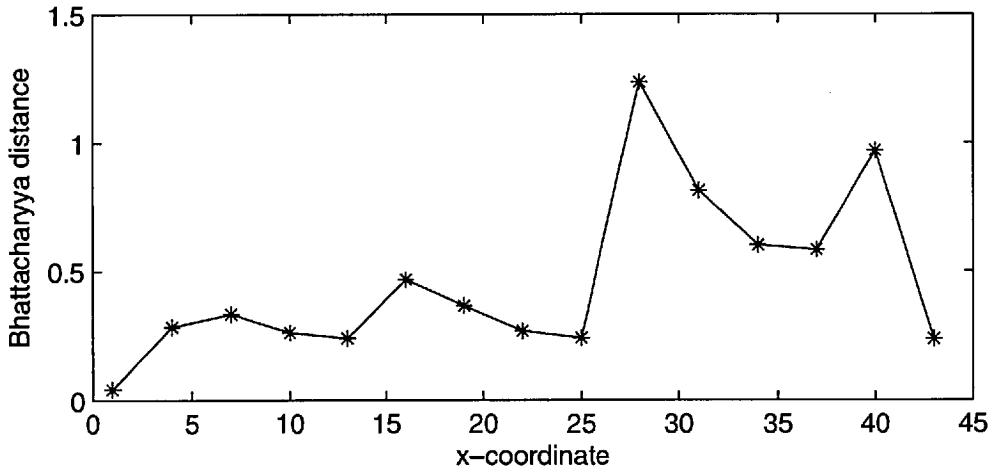


Figure 5.24: The Bhattacharyya distance corresponding to a 2D feature in an unpredictable motion

5.2 Efficient hardware mapping of Kalman filter

The previous section described the algorithms for the long-term predictor, the short-term predictor, the distribution distance, and the suppression module. Both the long-term predictor and the short-term predictor require intensive computations. However, due to the "long-term" nature of the long-term predictor, only one computation is needed for multiple frames. On the other hand, one computation of the short-term predictor is required for each 2D feature at every one frame. More specifically, the computation refers to the calculations of Kalman filter equations ranging from (5.18) to (5.23). Within the proposed saliency framework, the Kalman filter consumes a large portion of the computational power. It is therefore very beneficiary to have an efficient hardware mapping of the Kalman filter.

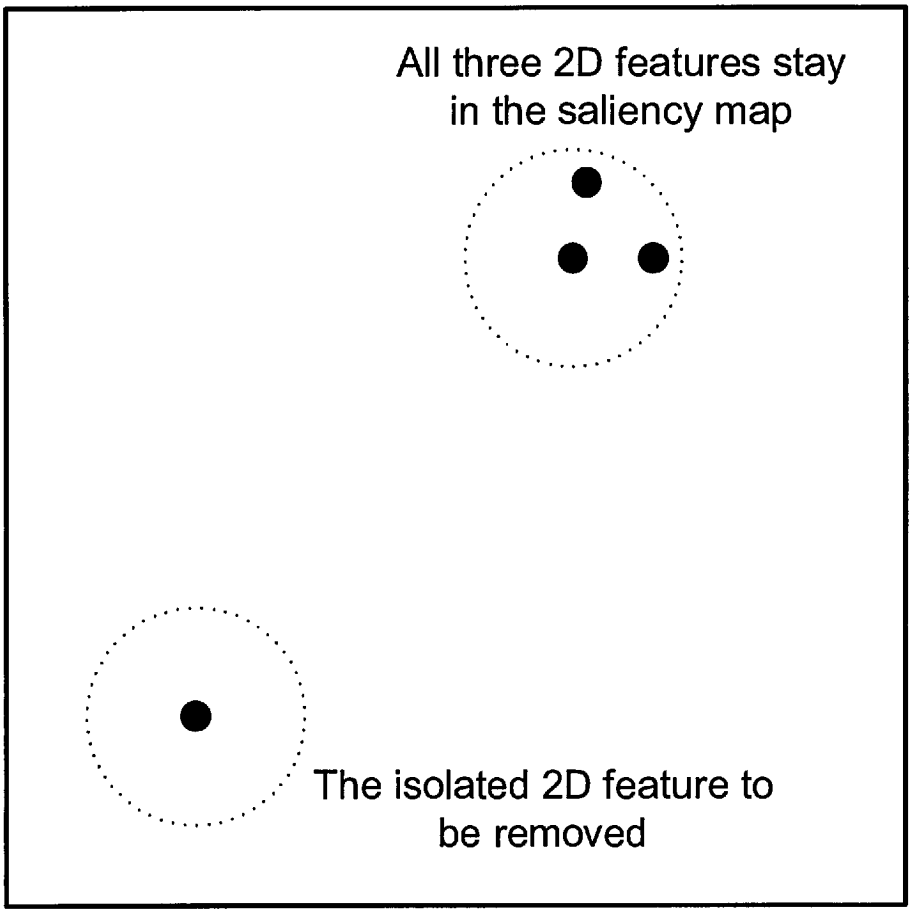


Figure 5.25: The suppression process: after the 2D features are put through the threshold screening, the neighbourhood of each 2D features is investigated. The black dot represents the 2D feature and the dotted circle represents the radius of the investigated neighbourhood. The 2D feature on the bottom left does not have any other 2D feature in its neighbourhood. It is thus removed from the saliency map. The 2D features on the top right are in close proximity to each other and hence they all remain in the saliency map

This section presents an approximation of the conventional Kalman filter by using Taylor expansion and matrix calculus in order to remove the hardware expensive part of the algorithm. The Bierman-Thornton algorithm, as the exact counterpart of the proposed Approximate Kalman filter algorithm, is also implemented for comparison purposes. Comparing to the Bierman-Thornton algorithm, the FPGA implementation results demonstrate that the proposed

Approximate Kalman filter implementation achieves one order of magnitude higher throughput using less hardware resources, obtaining similar convergence rate and accuracy.

5.2.1 Kalman filter implementations in the literature

A number of Kalman filter implementations have been proposed in the literature from the perspective of VLSI design [Yeh88, SH87, KH91, IG89]. Yeh [Yeh88] implemented the conventional Kalman filter on a trapezoidal array and used the Faddeev algorithm [FFW63] to execute matrix operations including matrix multiplication, matrix addition and matrix inversion. Other works resorted to the alternative least squares formulation and belong to the class of square-root Kalman filters. Also, systolic architectures have been proposed and a variety of implementations [SH87, KH91, IG89] are distinguished by the difference in data flow and the order of computations. In [SH87, KH91, IG89], the computational intensive matrix inversion from the conventional Kalman filter is replaced by an orthogonal transformation such as Given transformation for QR triangularization. Recent FPGA implementations [LS97, TRD99] are based on the conventional formulation of the Kalman filter. They are restricted to limited small size problems that either have one state or use a specific type of matrix whose inverse can be expressed in a simple form. However, these approaches have a limited range of applications.

All of the aforementioned work focus on the theoretical proposition of the algorithm and architecture and do not give enough information on the ac-

tual implementation in hardware, let alone in a modern FPGA. The major contributions of the work described in this section are: 1) The use of Taylor expansion and matrix calculus is proposed to derive an approximate formulation of the Kalman filter. By approximating the matrix inversion, which is the most computationally demanding part in a Kalman filter, the computational complexity is reduced significantly; 2) The Bierman-Thornton [Bie77, Tho76] Kalman filter is also implemented for the first time in an FPGA and compared to the proposed algorithm.

5.2.2 Conventional Kalman Filter

The Kalman filter described earlier in this chapter in Subsection 5.1.2 is the conventional Kalman filter. In the *measurement update* stage, the parameters of the conventional Kalman filter are updated using (5.18) to (5.20).

$$\mathbf{K}_k = \mathbf{P}_k^- \mathbf{H}^T [\mathbf{H}\mathbf{P}_k^- \mathbf{H}^T + \mathbf{R}]^{-1} \tag{5.18}$$

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + \mathbf{K}_k(\mathbf{z}_k - \mathbf{H}\hat{\mathbf{x}}_k^-) \tag{5.19}$$

$$\mathbf{P}_k = [\mathbf{I} - \mathbf{K}_k\mathbf{H}]\mathbf{P}_k^-[\mathbf{I} - \mathbf{K}_k\mathbf{H}]^T + \mathbf{K}_k\mathbf{R}_k\mathbf{K}_k^T \tag{5.20}$$

By using the optimal gain in (5.18), the update of the state estimation error covariance matrix in (5.20) can be simplified as in (5.21).

$$\mathbf{P}_k = [\mathbf{I} - \mathbf{K}_k\mathbf{H}]\mathbf{P}_k^- \tag{5.21}$$

In the *prediction* stage, the future states are predicted by using the projection equations:

$$\text{Project state:} \quad \hat{\mathbf{x}}_k^- = \mathbf{F}\hat{\mathbf{x}}_{k-1} \tag{5.22}$$

$$\text{Project covariance:} \quad \mathbf{P}_k^- = \mathbf{F}\mathbf{P}_{k-1}\mathbf{F}^T + \mathbf{Q} \tag{5.23}$$

5.2.3 Approximate Kalman Filter

To implement the conventional Kalman filter described in Section 5.2.2, one has to compute the matrix inversion in (5.18) in order to calculate the optimal gain. Matrix inversion constitutes the biggest obstacle in the integration of the Kalman filter on a silicon chip. That is because it is considerably more expensive than any other operation in the conventional Kalman filter. Hence, arises the motivation to avoid the matrix inversion with the main objective to decrease the implementation requirements. In [KD86, Mas98], the matrix inversion process is also avoided by manually computing the Kalman gain in advance. During the Kalman filtering operations, the pre-computed Kalman gains are fed into the process. However, this approach disregards all the information contained in the new measurement with respect to the gain.

In this subsection, we derive a solution that approximates the optimal Kalman gain in (5.18) by resorting to the Taylor expansion and matrix calculus. This solution significantly reduces the computational complexity by relieving us from performing the demanding matrix inversion. The drawback is a slower

convergence rate than the conventional algorithm since the Kalman gain is not optimal anymore. However, the error due to the approximation is not accumulated since the Kalman filter parameters are properly updated.

Derivation

The matrix function we want to approximate in the conventional Kalman filter is $[\mathbf{H}\mathbf{P}_k^-\mathbf{H}^T + \mathbf{R}]^{-1}$ in (5.18). It has only one variable matrix $\mathbf{P}_k^-$, which is updated at each time step, whereas the other matrices are constant. Our goal is to approximate this matrix function using Taylor expansion. The Taylor series of a differentiable matrix-valued function $g(\mathbf{Y})$ around value $\mathbf{X}$ is given in (5.24), where $\overset{\rightarrow \mathbf{Y}-\mathbf{X}}{dg(\mathbf{X})}$ is the directional derivative [Dat05].

$$g(\mathbf{Y}) = g(\mathbf{X}) + \overset{\rightarrow \mathbf{Y}-\mathbf{X}}{dg(\mathbf{X})} + \frac{1}{2!}\overset{\rightarrow \mathbf{Y}-\mathbf{X}}{dg^2(\mathbf{X})} + \ldots \tag{5.24}$$

The first order approximation of (5.24) is given in (5.25).

$$g(\mathbf{Y}) \approx g(\mathbf{X}) + \overset{\rightarrow \mathbf{Y}-\mathbf{X}}{dg(\mathbf{X})} \tag{5.25}$$

The directional derivative can be calculated from (5.26)

$$\overset{\rightarrow \mathbf{Y}-\mathbf{X}}{dg(\mathbf{X})} = \sum_{u,l} \frac{dg(\mathbf{X})}{d\mathbf{X}_{u,l}} (\mathbf{Y} - \mathbf{X})_{u,l} \tag{5.26}$$

where $\mathbf{X}_{u,l}$ denotes the scalar element on the u^{th} row and l^{th} column of matrix $\mathbf{X}$. Let $g(\mathbf{X}) = [\mathbf{H}\mathbf{X}\mathbf{H}^T + \mathbf{R}]^{-1}$ where $\mathbf{X}$ is the variable matrix and $\mathbf{H}$ and $\mathbf{R}$

are both constant matrices. Thus,

$$\begin{aligned}\frac{dg(\mathbf{X})}{d\mathbf{X}_{u,l}} &= \frac{d([\mathbf{HXH}^T+\mathbf{R}]^{-1})}{d\mathbf{X}_{u,l}} \\ &= -[\mathbf{HXH}^T+\mathbf{R}]^{-1}\frac{d(\mathbf{HXH}^T+\mathbf{R})}{d\mathbf{X}_{u,l}}[\mathbf{HXH}^T+\mathbf{R}]^{-1} \\ &= -[\mathbf{HXH}^T+\mathbf{R}]^{-1}\mathbf{H}\mathbf{E}_{u,l}\mathbf{H}^T[\mathbf{HXH}^T+\mathbf{R}]^{-1} \qquad (5.27)\end{aligned}$$

where

$$\mathbf{E}_{u,l} = \frac{d\mathbf{X}}{d\mathbf{X}_{u,l}} = \begin{pmatrix} 0 & 0 & 0 & \dots \\ 0 & 1 & 0 & \dots \\ 0 & 0 & 0 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \begin{matrix} \\ u \\ \\ \\ \\ l \end{matrix}$$

is a zero matrix with 1 on the u^{th} row and l^{th} column. For $\mathbf{X}=\mathbf{I}$, where $\mathbf{I}$ is an identity matrix, (5.27) becomes

$$\left.\frac{dg(\mathbf{X})}{d\mathbf{X}_{u,l}}\right|_{\mathbf{X}=\mathbf{I}} = -[\mathbf{HH}^T+\mathbf{R}]^{-1}\mathbf{H}\mathbf{E}_{u,l}\mathbf{H}^T[\mathbf{HH}^T+\mathbf{R}]^{-1} \qquad (5.28)$$

Let $\mathbf{C} = [\mathbf{HH}^T+\mathbf{R}]^{-1} = g(\mathbf{X})|_{\mathbf{X}=\mathbf{I}}$. By substituting (5.28) in (5.26), we get (5.29).

$$\overset{\rightarrow \mathbf{Y}-\mathbf{X}}{dg(\mathbf{X})} = -\sum_{u,l}\left[(\mathbf{CH}\mathbf{E}_{u,l}\mathbf{H}^T\mathbf{C})(\mathbf{Y}-\mathbf{X})_{u,l}\right] \qquad (5.29)$$

which can be interpreted as a weighted sum of matrices. The scalar weights are $(\mathbf{Y}-\mathbf{X})_{u,l}$ and the matrices are $\mathbf{CHE}_{u,l}\mathbf{H}^T\mathbf{C}$. Hence, the first-order Taylor approximation in (5.25) is given by (5.30).

$$g(\mathbf{Y}) = \mathbf{C} - \sum_{u,l} \Big[(\mathbf{CHE}_{u,l}\mathbf{H}^T\mathbf{C})(\mathbf{Y}-\mathbf{I})_{u,l} \Big] \tag{5.30}$$

Thus, the approximate Kalman gain in (5.18) is given in (5.31).

$$\mathbf{K}_k^{\text{approx}} = \mathbf{P}_k^- \mathbf{H}^T \Big(\mathbf{C} - \sum_{u,l} \Big[(\mathbf{CHE}_{u,l}\mathbf{H}^T\mathbf{C})(\mathbf{P}_k^- - \mathbf{I})_{u,l} \Big] \Big) \tag{5.31}$$

It is important to point out that due to the approximation, the state estimation error covariance matrix must be updated using (5.20) instead of the simplified (5.21). The derivation from (5.28) to (5.31) takes the Taylor expansion around $\mathbf{I}$. In practice, we can insert any *a priori* information matrix. As can be seen from the approximate Kalman gain in (5.31), we eliminate the need for matrix inversion that has to be computed at each step. $\mathbf{C} = [\mathbf{H}\mathbf{H}^T + \mathbf{R}]^{-1}$ is constant and can be precomputed, as well as the terms $\mathbf{CHE}_{u,l}\mathbf{H}^T\mathbf{C}$ in (5.31).

Hardware Implementation

Essentially, the Approximate Kalman filter system consists of the implementation of equations (5.31), (5.19), (5.20), (5.22) and (5.23). The high-level overview of the Approximate Kalman filter implementation is shown in Figure 5.26. It comprises the Matrix Multiplication module, the Matrix Addition/Subtraction module and the Weighted Sum of Matrices module. At the

initialization stage, all the parameter matrices such as $\mathbf{F}$ and $\mathbf{H}$ are read from the blockRAM and written to a register array. Also, the matrix $\mathbf{C}$ and the constant terms in (5.31) are downloaded. This increases flexibility because the hardware does not need to be rebuilt for a different problem. Instead, new parameter matrices can be updated in the blockRAM and re-initialize the system for the corresponding register array to reflect the update. During one operation of the Approximate Kalman filter block, the measurement vector is fed into the block and the state vector is returned.

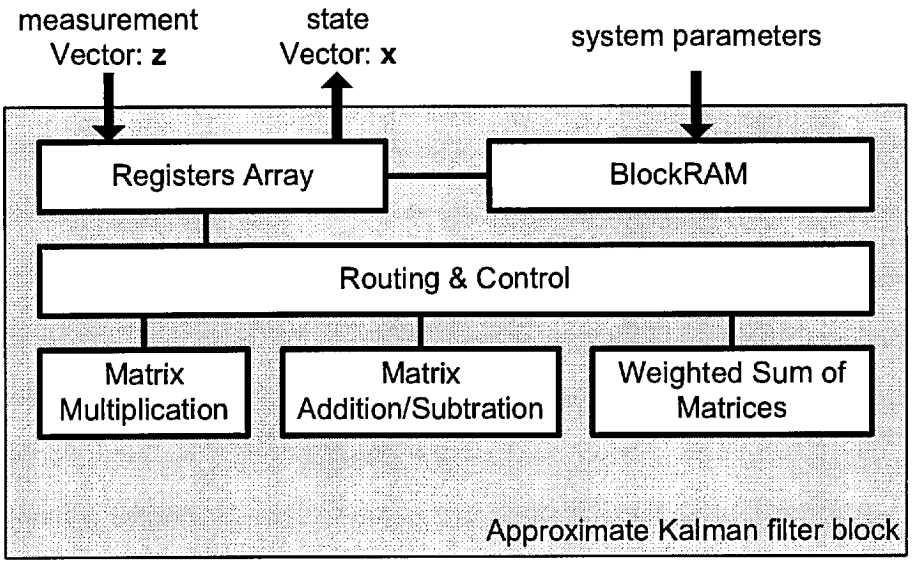


Figure 5.26: Overview of the Approximate Kalman filter

The architecture of a Vector Multiplication module, which is the building block of the Matrix Multiplication module, is described below. To multiply a row vector $\mathbf{a} = [a_1, a_2, a_3, a_4]$ and a row vector $\mathbf{b} = [b_1, b_2, b_3, b_4]$, a Vector Multiplication module that compute the scalar $c = \mathbf{a}\mathbf{b}^T$ is shown in Figure 5.27.

The Matrix Multiplication module consists of many Vector Multiplication

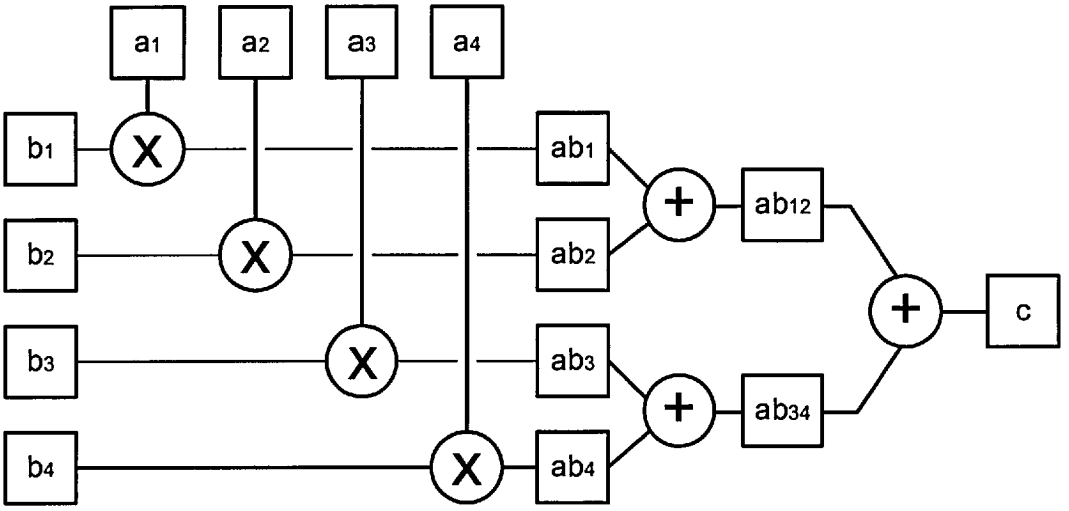


Figure 5.27: Vector Multiplication module

(VM) modules. It is important to execute matrix multiplication rapidly because this operation is predominant in the Approximate Kalman filter. All the vector multiplications are executed in parallel in order to maximize speed as shown in Figure 5.28. Similarly, the Matrix Addition/Subtraction module is formed by an array of adders/subtractors. Each adder/subtractor performs matrix element addition/subtraction in parallel to the others.

The Weighted Sum of Matrices module is shown in Figure 5.29 and computes $\sum_{u,l}\left[(\mathbf{CHE}_{u,l}\mathbf{H}^T\mathbf{C})(\mathbf{P}_k^- - \mathbf{I})_{u,l}\right]$ in (5.31). The rest of the operations in (5.31) are performed by the matrix multiplication module and matrix addition/subtraction module. In Figure 5.29, $\mathbf{T}_{u,l}$ represents $\mathbf{CHE}_{u,l}\mathbf{H}^T\mathbf{C}$, which is an array of constant matrices. These matrices are precomputed and loaded to registers from blockRAM at initialization. $\mathbf{t}_{u,l}$ represents the variable scalar weight $(\mathbf{P}_k^- - \mathbf{I})_{u,l}$. The $\mathbf{T}_{u,l}$ and the corresponding $\mathbf{t}_{u,l}$ are multiplexed to the Matrix scalar multiplication module, which multiplies the $\mathbf{t}_{u,l}$ to every element

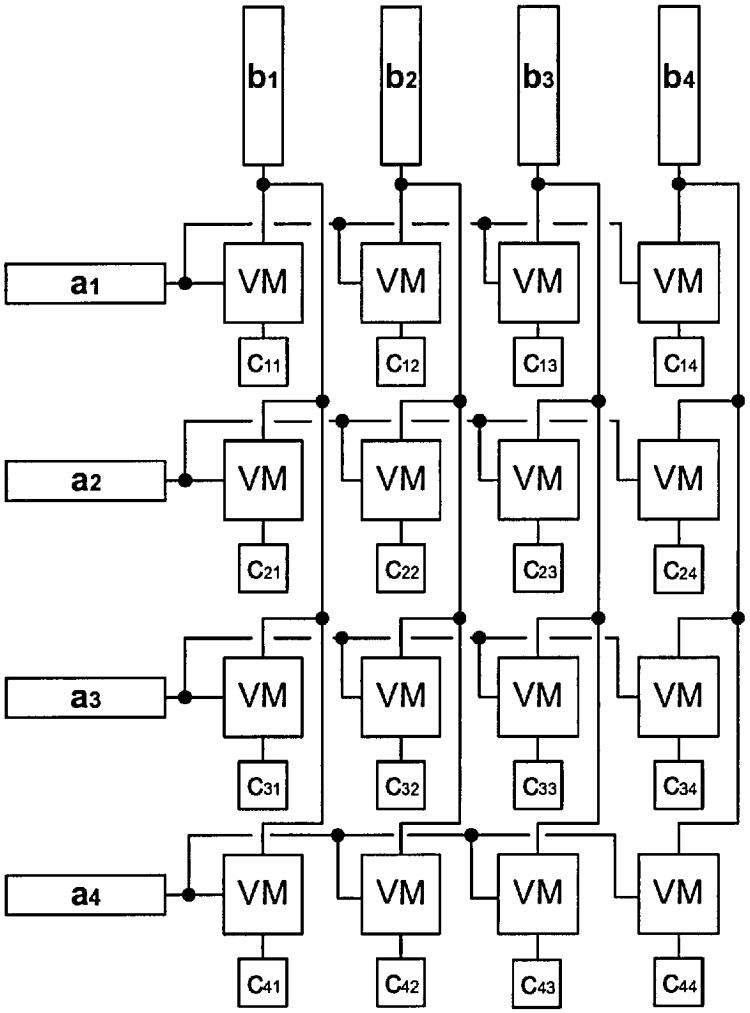


Figure 5.28: Matrix Multiplication module

of $\mathbf{T}_{u,l}$. The summation is achieved through accumulating the result of matrix additions to variable $\mathbf{T}_{\text{out}}$, which eventually contains the weighted sum of the matrices.

5.2.4 Bierman-Thornton UD Kalman filter

Although the conventional Kalman filter in Section 5.2.2 can be implemented directly in hardware, it is clear that using an iterative method for matrix inversion is particularly sensitive to round-off errors and likely to cause numerical

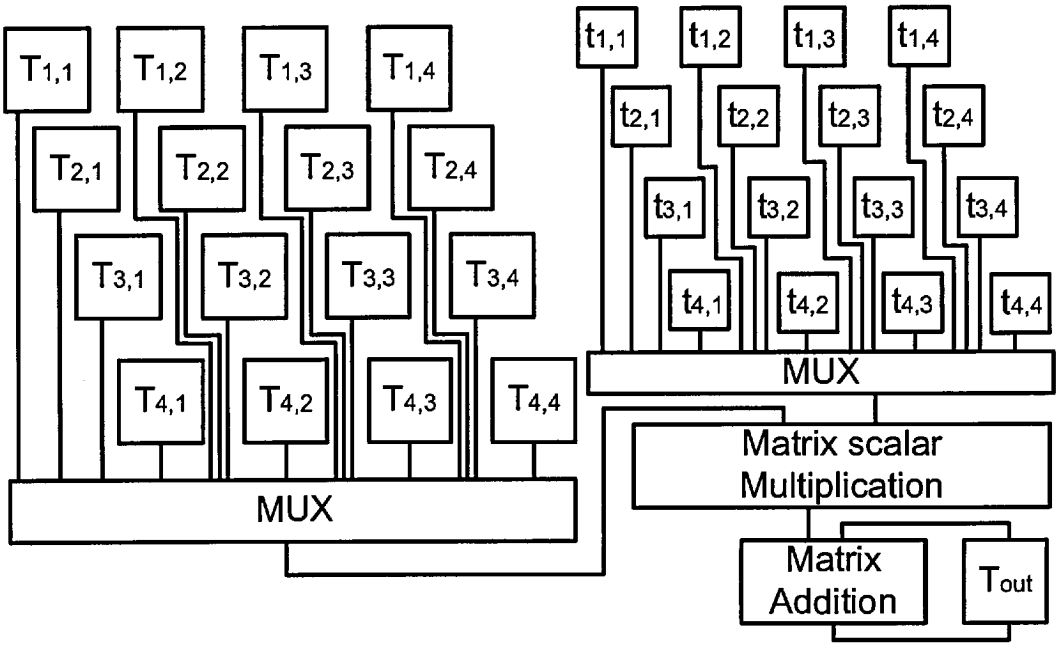


Figure 5.29: The Weighted Sum of Matrices module

problems. Square-root forms of the Kalman filter are more robust numerically because the triangular square roots are propagated to preserve the symmetry of the covariance matrices in the presence of round-off errors [GA01]. A square-root Kalman filter takes the "square-root", the Cholesky factor, of the state estimation error covariance matrix $\mathbf{P} = \mathbf{L}\mathbf{L}^T$. A set of equations is derived to update the factor $\mathbf{L}$ instead of the covariance matrix $\mathbf{P}$ itself. In practice, mathematically equivalent implementation methods can have very different numerical stability at the same precision. Bierman-Thornton UD Kalman filter [Bie77, Tho76] belongs to the square-root Kalman filter framework and it consists of a pair of algorithms, the Bierman algorithm and the Thornton algorithm. It should be noted that the Bierman-Thornton algorithm computes the optimal Kalman gain, thus it is not an approximation to the conventional Kalman filter.

Bierman-Thornton algorithm

The Bierman algorithm executes the *measurement update* of the modified Cholesky factor $\mathbf{U}$ and $\mathbf{D}$ of the covariance matrix $\mathbf{P} = \mathbf{U}\mathbf{D}\mathbf{U}^T$, where $\mathbf{U}$ is a unit[2] upper triangular matrix and $\mathbf{D}$ is a diagonal matrix. It takes in a scalar element of m-dimensional measurement vector $\mathbf{z}$ one at a time, and uses the corresponding vector from the measurement matrix $\mathbf{H}$ to update the state vector $\mathbf{x}$, as well as $\mathbf{U}$ and $\mathbf{D}$. Therefore, it needs m iterations to execute the measurement update.

The Thornton algorithm [Tho76] is responsible for the *prediction* of $\mathbf{x}$, $\mathbf{U}$ and $\mathbf{D}$, which is equivalent to the *prediction* stage of conventional Kalman filter. A detailed description of the Bierman-Thornton algorithm can be found in [GA01].

Hardware implementation

The interaction between the Bierman module and the Thornton module is shown in Figure 5.30. The scalar values from the measurement vector $\mathbf{z}$, along with the associated system parameters from $\mathbf{H}$ and measurement error covariance matrix $\mathbf{R}$, are multiplexed into the Bierman module. After the Bierman module processes the entire measurement vector, the results are ready to be read by the Thornton module. These results are $\mathbf{x}$, $\mathbf{U}$ and $\mathbf{D}$. The Thornton algorithm returns the *predicted* $\mathbf{x}$,$\mathbf{U}$,$\mathbf{D}$, which would be the input for the Bierman module at the next time step.

[2]A unit triangular matrix is a triangular matrix with 1 on the diagonal

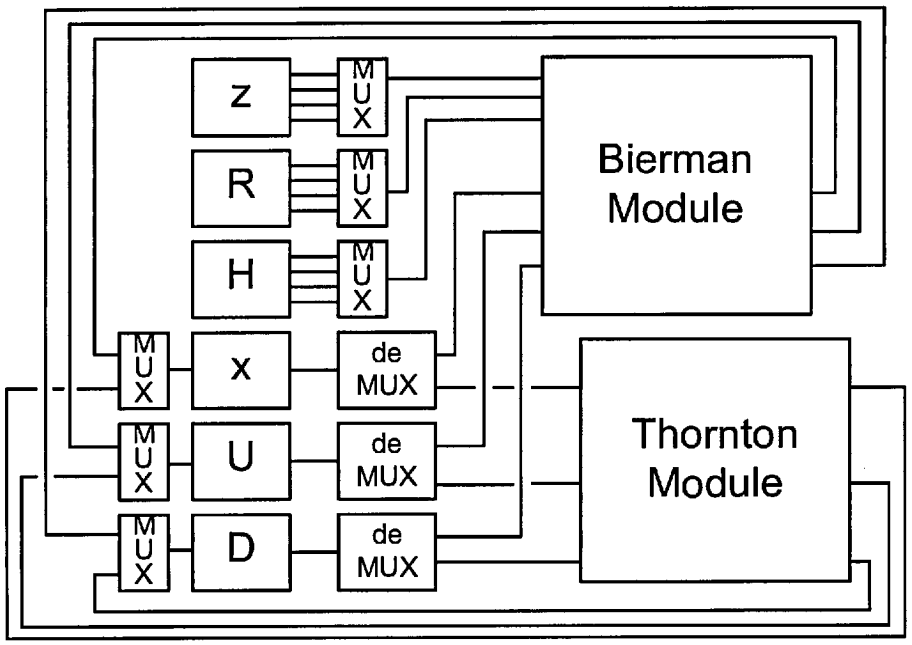


Figure 5.30: Overview of the Bierman-Thronton Kalman filter system

5.2.5 Performance Evaluation

The proposed Approximate Kalman filter and the Bierman-Thornton Kalman filter are implemented and their performance is compared. The target device is the Xilinx® Virtex-4 FPGA chip XC4VFX140 with package FF1517 and speed grade 11. The hardware is designed using Handel-C by Celoxica®. The dimension of the state vector is referred to as n and the dimension of the measurement vector is referred to as m. Unless stated otherwise, $n = 4$ and $m = 4$ throughout the experiment section.

Convergence Rate

To test the convergence rate of the two Kalman filter implementations, a Kalman filter system with a constant state is used. This setup can show how fast the *estimated* states from the two Kalman filters converge to the

true state from the noisy measurements. Figure 5.31 shows that the Approximate Kalman filter does converge to the true value but not as fast as the Bierman-Thornton algorithm. This is expected as the Kalman gain in our algorithm is not optimal due to the approximation. However, it follows the Bierman-Thornton algorithm quite closely.

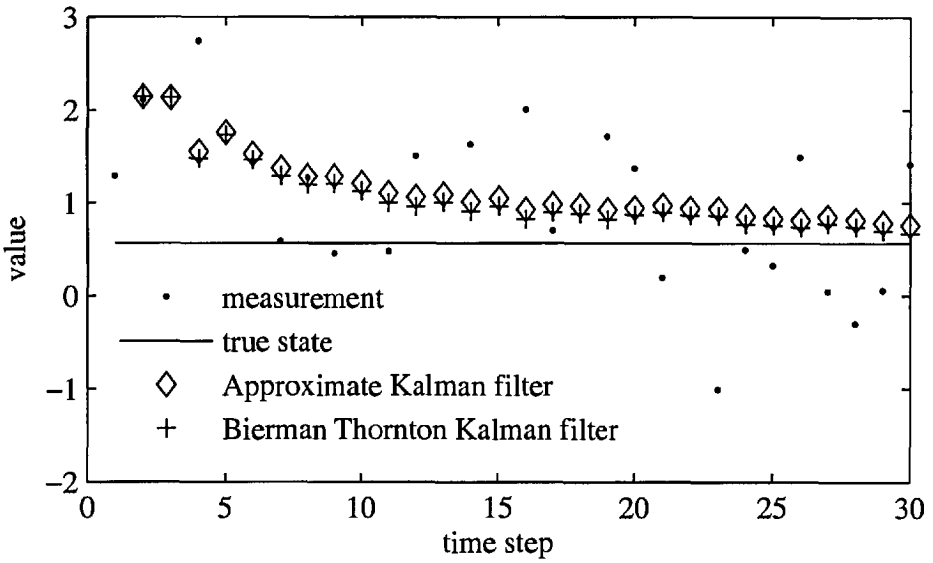


Figure 5.31: The convergence rate of the two Kalman filters

Accuracy

To test the accuracy of the two implementations, a dynamic system is used, in which the state vector changes with time. A uniform wordlength is used throughout the system. Without loss of generality, half of the bits were used for the sign and integer part, and the other half were used for the fractional part. Figure 5.32 shows the impact on the average square error of the estimated state vectors by varying the wordlength. The square error of a state vector is

defined as in (5.32).

$$\frac{1}{n}\sum_{i=1}^{n}\left(\mathbf{x}_i^{\text{true}} - \mathbf{x}_i^{\text{estimated}}\right)^2 \tag{5.32}$$

where $\mathbf{x}_i^{\text{true}}$ and $\mathbf{x}_i^{\text{estimated}}$ are the i^{th} scalar element in the *true* and *estimated* state vectors. For each wordlength value, both Kalman filter implementations were run with 1000 measurement vectors. As reference, the error from a conventional Kalman filter algorithm implemented in software with double floating point precision is also shown in Figure 5.32. The large error of the Bierman-Thornton implementation below 18-bits is due to overflow/underflow. The input data are the same for both implementations. It can be concluded that the approximation Kalman filter implementation requires less wordlength to avoid overflow/underflow than the Bierman-Thornton algorithm.

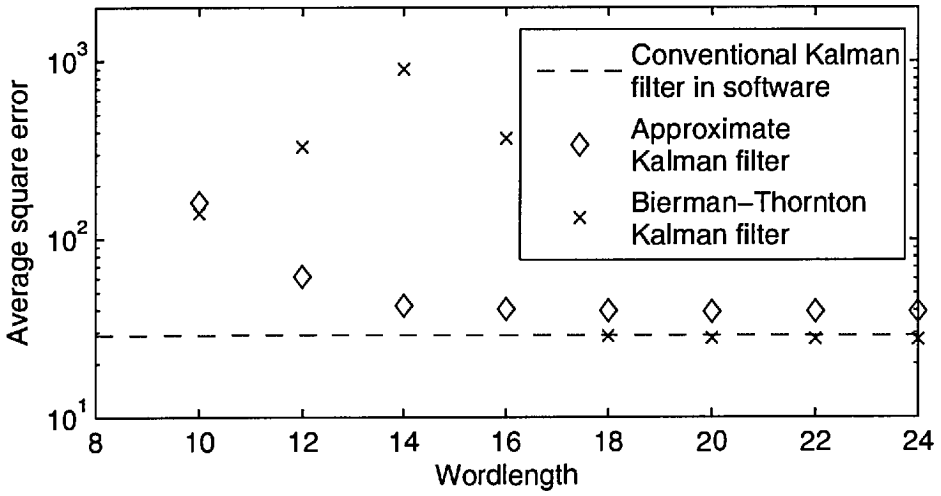


Figure 5.32: Accuracy versus wordlength

Area

Figure 5.33 provides the information on how the area, in terms of FPGA slices, varies with wordlength. In this comparison, the DSP48 blocks are not used. Both the Bierman-Thornton and the approximate Kalman filter implementations use 9 blockRAMs in the FPGA. As illustrated in Figure 5.33, both implementations scale linearly with wordlength.

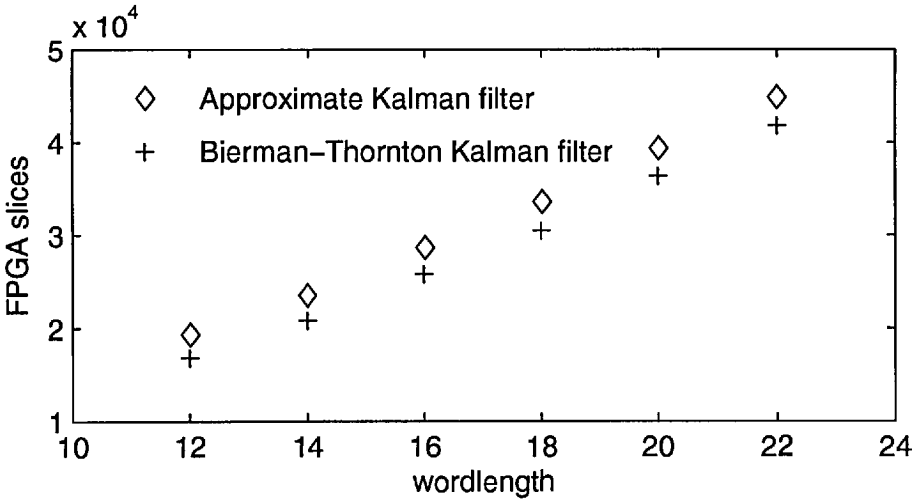


Figure 5.33: Area comparison in FPGA slices

Table 5.1 shows the hardware resource usages for Kalman filters for different problem sizes, to which both n and m are equal. The "no DSP" column corresponds to the number of occupied slices when the DSP48 blocks in Virtex-4 were not used. Only at the problem size of 4, the approximate Kalman filter implementation is slightly larger than the Bierman-Thornton one, which is also reflected in Figure 5.33. It can be concluded that the Approximate Kalman filter implementation scales better than the Bierman-Thornton Kalman filter implementation as the dimension of the measurement and state vector

Problem	Approximate			Bierman-Thornton		
	with DSP48		no DSP	with DSP48		no DSP
size	slices	DSP	slices	slices	DSP	slices
3	3848	24	5848	4728	40	6854
4	12302	96	19323	9019	110	16783
5	13668	150	25972	18357	190	32575
6	18707	162	34583	32897	192	47407

Table 5.1: Hardware resource usage for 12-bits systems. Available resources: 192 DSP48s, 63168 slices.

increases.

Throughput

Throughput is a metric that indicates how fast the Kalman filter hardware can process measurements and return state vectors. Here, throughput is defined as the number of state vectors estimated per second. The FPGA implementations use 16-bits in this subsection. For comparison, a software Kalman filter employing the conventional formulation described in Section 5.2.2 is included. The conventional Kalman filter was implemented in C++ at double floating point precision using OpenCV library [opeet] and was run on a Pentium 4 PC with 2.4GHz CPU and 1GB of RAM. The results are summarized in Table 5.2. It can be concluded that the Approximate Kalman filter FPGA implementation has around 10 times larger throughput than the Bierman-Thronton Kalman filter FPGA implementation which is in turn faster than the software implementation by one order of magnitude. Since the Approximate Kalman filter uses less hardware resources as demonstrated in Section 5.2.5, it is evident that our proposed algorithm is a better design choice.

Kalman Filter Algorithm	Latency (Clock cycle)	Maximum frequency	Through-put
Approximate	78	41.6MHz	532966
Bierman-Thornton	578	39.1 MHz	63087
Conventional in C++	N/A	N/A	2567

Table 5.2: Throughput comparison for 16-bits implementations

5.3 Summary

In this chapter, Section 5.1 describes software algorithms and Section 5.2 focuses on hardware exploration. Specifically, Subsection 5.1.1 describes the use of autocorrelation and power spectrum to determine periodicity for the **long-term predictor**. Subsection 5.1.2 describes the utility of the Kalman filter and the Bhattacharyya distance to quantify temporal saliency for the **short-term predictor**. Subsection 5.1.3 describes the operations for the **Suppression** module.

In Section 5.2, we propose a new architecture for the Kalman filter, which reduces the computational complexity of a conventional Kalman filter by approximating the matrix inversion function using Taylor expansion. We implemented both the proposed Approximate Kalman filter and the Bierman-Thornton Kalman filter in a modern FPGA. The results demonstrate that the proposed Approximate Kalman filter implementation has significantly larger throughput using less hardware resources, achieving at the same time similar accuracy and convergence rate.

Chapter 6

Conclusions

6.1 Summary

This thesis presents a novel temporal unpredictability detection framework. Combined with appropriate spatial saliency algorithms, the proposed framework is capable of effectively detecting temporal unpredictability. The novelty of our saliency framework is that the unpredictability of the 2D features' motion is employed to determine temporal saliency. In other words, unpredictable motions can be temporally salient.

Chapter 3 gives an overview of the proposed spatiotemporal saliency framework. This framework consists of a number of modules. The mechanism and justification of joining all the modules to form saliency is explained. Performance evaluation of the saliency framework using video sequences is also provided. Experiment with real-data show that the proposed framework can successfully detect salient movement in the video.

In Chapter 4, the **2D Feature Detection** module, the **Feature Tracking** module, and the **Adaptive Decision** module are described in more detail. In Chapter 5, the **Temporal Motion Prediction** module and **Suppression** module are elaborated to explain the method of quantifying unpredictability. All of these aforementioned descriptions allow a thorough understanding of the proposed framework.

Moreover, being modules based, the proposed framework is flexible and poses opportunities for hardware space exploration. Two of the important algorithms out of the framework are identified as suitable for hardware acceleration. These are the eigenvalue computation and the Kalman filter computation.

The investigations of these two algorithms in search for efficient hardware architectures lead to significant performance improvement over previous approaches. These two pieces of hardware work greatly facilitate the hardware implementation of the proposed framework to detect spatiotemporal saliency in real-time video sequence.

Section 4.2 explores the systolic and dual processor architectures for eigenvalue computation based on the *Approximate Jacobi Method.* For comparison purpose, the *Exact Jacobi Method* and *Window Jacobi Method* are also implemented in the two architectures. In terms of area and throughput, we show with hardware experimental results that the *Approximate Jacobi Method* in systolic architecture is the most efficient system to compute eigenvalues for $n \times n$ symmetric matrices. For a high speed application, we also propose

a pipelined architecture resorting to the *Algebraic Method* to compute the eigenvalues for 3×3 symmetric matrices, and successfully achieve very high throughput using only 22% of the area compared to that of the *Approximate Jacobi Method.*

In Section 5.2, a new architecture for the Kalman filter is described, which reduces the computational complexity of a conventional Kalman filter by approximating the matrix inversion function using Taylor expansion. The implementation of both the proposed Approximate Kalman filter and the Bierman-Thornton Kalman filter are accomplished in a modern FPGA. The results demonstrate that the proposed Approximate Kalman filter implementation has significantly larger throughput using less hardware resources, achieving at the same time similar accuracy and convergence rate.

In this thesis, we demonstrated that temporal unpredictability can be detected in real-time, which can be a measure of temporal saliency.

6.2 Future work

As mentioned in Section 3.1, 2D features need to be refreshed every n frames to reflect the up-to-date presence of 2D features. n is set according to the prior knowledge of the scene's activity. More research can be done on how to automatically determine n by analyzing the scene's activity.

In the realization of the saliency framework described in this thesis, the short-term predictor assumes that *linear* motion is predictable and hence tem-

porally non-salient. As a result, the Kalman filter is employed as the short-term predictor because it is optimal in predicting linear motion. The framework itself is not limited to linear motion. Users may find certain non-linear motion predictable and change the criterion of temporal saliency. In this situation, the Kalman filter is not suitable for predicting these non-linear motion. It is worthwhile to investigate how the use of particle filter [IB98] as the short-term predictor affects the performance of the saliency framework under non-linear saliency criterion.

From the hardware perspective, the rest of the modules can be implemented in hardware. More importantly, the mapping of the entire framework into hardware is interesting. This enables the system-on-chip solution to temporal saliency for real-time video sequence.

Bibliography

[AAB03] A. Ahmedsaid, A. Amira, and A. Bouridane. Improved SVD systolic array and implementation on FPGA. *FPT Proceedings*, pages 35 – 42, 2003.

[AMGC02] M.S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174 – 88, 2002.

[AQt] http://www.automatedqa.com/products/aqtime/.

[Arj92] A. Arjomand. *Fast Parallel Computation of the Singular Value Decomposition of Real Matrices Using CORDIC Arithmetic.* PhD thesis, Department of Electrical Engineering, Yale University, 1992.

[AUCS98] F. Arrebola, C. Urdiales, P. Camacho, and F. Sandoval. Vision system based on shifted fovea multiresolution retinotopologies. In *Industrial Electronics Society, 1998. IECON '98. Proceedings of*

the 24th Annual Conference of the IEEE, volume 3, pages 1357–1361 vol.3, 1998.

[BDY99] J.A. Brefczynski and E.A. De Yoe. A physiological correlate of the'spotlight'of visual attention. *Nature neuroscience*, 2:370–374, 1999.

[Bie77] G.J. Bierman. *Factorization methods for discrete sequential estimation.* Academic Press, 1977.

[BLVL85] Richard P. Brent, Franklin T. Luk, and Charles Van Loan. Computation of the singular value decomposition using mesh-connected processors. *Journal of VLSI and Computer Systems*, 1(3):242 – 270, 1985.

[BM96] N. G. Bourbakis and J. S. Mertoguno. Kydon: An autonomous, multi-layer image-understanding system: Lower layers. *Engineering Applications of Artificial Intelligence*, 9(1):43–52, 1996.

[Boa96] K.A. Boahen. A retinomorphic vision system. *Micro, IEEE*, 16(5):30–39, 1996.

[Bou00] Jean-Yves Bouguet. Pyramidal Implementation of the Lucas Kanade Feature Tracker Description of the algorithm. *Microprocessor Research Labs, Intel Corporation*, 2000.

[BPH00] E. Blaser, Z.W. Pylyshyn, and A.O. Holcombe. Tracking an object through feature space. *Nature*, 408(6809):196–199, 2000.

[Bri88] E. Oran Brigham. *The fast Fourier transform and its applications.* Prentice-Hall signal processing series. Prentice Hall, Englewood Cliffs, N.J., 1988.

[BS89] C. Bandera and P.D. Scott. Foveal machine vision systems. In *Systems, Man and Cybernetics, 1989. Conference Proceedings., IEEE International Conference on*, pages 596–599 vol.2, 1989.

[BWHD02] R.A. Blum, C.S. Wilson, P.E. Hasler, and S.P. DeWeerth. A CMOS imager with real-time frame differencing and centroid computation. In *Circuits and Systems, 2002. ISCAS 2002. IEEE International Symposium on*, volume 3, pages 329–332, 2002.

[CAS98] P. Camacho, F. Arrebola, and F. Sandoval. Multiresolution sensors with adaptive structure. In *Industrial Electronics Society, 1998. IECON '98. Proceedings of the 24th Annual Conference of the IEEE*, volume 2, pages 1230–1235, 1998.

[CBM+02] G. Chapinal, S.A. Bota, M. Moreno, J. Palacin, and A. Herms. A 128×128 cmos image sensor with analog memory for synchronous image capture. *Sensors Journal, IEEE*, 2(2):120–127, 2002.

[CCKW05] Wen-Huang Cheng, Wei-Ta Chu, Jin-Hau Kuo, and Ja-Ling Wu. Automatic video region-of-interest determination based on user attention model. *IEEE International Symposium on Circuits and*

Systems (ISCAS) (IEEE Cat. No. 05CH37618), Vol. 4:3219 – 3222, 2005.

[celom] http://www.celoxica.com/.

[CL88] J.R. Cavallaro and F.T. Luk. CORDIC arithmetic for an SVD processor. *Journal of Parallel and Distributed Computing*, 5(3):271 – 90, 1988.

[CS02] M. Corbetta and Gordon L. Shulman. Control of goal-directed and stimulus-driven attention in the brain. *Nature neuroscience*, 3:201–215, 2002.

[CW99] Seung-Jong Choi and J.W. Woods. Motion-compensated 3-d subband coding of video. *Image Processing, IEEE Transactions on*, 8(2):155–167, 1999.

[Dat05] Jon Dattorro. *Convex Optimization & Euclidean Distance Geometry.* Meboo Publishing USA, 2005.

[Del89] J.-M. Delosme. CORDIC algorithms: theory and extensions. *Proceedings of the SPIE - The International Society for Optical Engineering*, 1152:131 – 45, 1989.

[Del92] Jean-Marc Delosme. Bit-level systolic algorithms for real symmetric and Hermitian eigenvalue problems. *Journal of VLSI Signal Processing*, 4(1):69 – 88, 1992.

[Del93] T. Delbruck. Silicon retina with correlation-based, velocity-tuned pixels. *Neural Networks, IEEE Transactions on*, 4(3):529–541, 1993.

[Del00] T. Delbruck. Silicon retina for autofocus. In *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on*, volume 4, pages 393–396 vol.4, 2000.

[DM94] T. Delbruck and C.A. Mead. Adaptive photoreceptor with wide dynamic range. In *Circuits and Systems, 1994. ISCAS '94., 1994 IEEE International Symposium on*, volume 4, pages 339–342 vol.4, 1994.

[DRP+06] J. Diaz, E. Ros, F. Pelayo, E.M. Ortigosa, and S. Mota. Fpga-based real-time optical-flow system. *IEEE Transactions on Circuits and Systems for Video Technology*, 16(2):274 – 279, 2006.

[Far00] G. Farneback. Fast and accurate motion estimation using orientation tensors and parametric motion models. *Proceedings 15th International Conference on Pattern Recognition*, 1:135 – 139, 2000.

[FB81] M.A. Fischler and R.C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381 – 95, 1981.

[FDC98] M. Fleury, A.C. Downton, and A.F. Clark. Co-design by parallel prototyping: optical-flow detection case study. In *High Performance Architectures for Real-Time Image Processing (Ref. No. 1998/197), IEE Colloquium on*, pages 8/1–8/13, 1998. TY - CONF.

[FFW63] Dmitrii Konstantinovich Faddeev, Vera Nikolaevna Faddeeva, and Robert Chadwell Williams. *Computational methods of linear algebra.* Freeman, 1963.

[FRJ92] C.L. Folk, R.W. Remington, and J.C. Johnston. Involuntary covert orienting is contingent on attentional control settings. *Journal of experimental psychology. Human perception and performance*, 18(4):1030–1044, 1992.

[GA01] Mohinder S. Grewal and Angus P. Andrews. *Kalman filtering : theory and practice using MATLAB.* Wiley, New York ; Chichester, 2nd edition, 2001.

[GBG+94] H. Greenspan, S. Belongie, R. Goodman, P. Perona, S. Rakshit, and C.H. Anderson. Overcomplete steerable pyramid filters and rotation invariance. *Proceedings 1994 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No.94CH3405-8)*, pages 222 – 228, 1994.

[GHB99] S.P. Gandhi, D.J. Heeger, and G.M. Boynton. Spatial attention affects brain activity in human primary visual cortex. *Proceedings of the National Academy of Sciences of the United States of America*, 96(6):3314–3319, 1999.

[GP98] W.S. Geisler and J.S. Perry. A real-time foveated multiresolution system for low-bandwidth video communication. *Proceedings of the SPIE - The International Society for Optical Engineering Human Vision and Electronic Imaging III, 26-29 Jan.*, 1998.

[GPS93] J. Gotze, S. Paul, and M. Sauer. An efficient Jacobi-like algorithm for parallel eigenvalue computation. *IEEE Transactions on Computers*, 42(9):1058 – 65, Sept. 1993.

[GVL96] Gene H. Golub and Charles F. Van Loan. *Matrix computations.* Johns Hopkins University Press, Baltimore; London, 3rd edition, 1996.

[HJ90] R. A. Horn and C. R. Johnson. *Matrix Analysis.* Cambridge University Press, Cambridge, 1990.

[HS88] C. Harris and M. Stephens. A combined corner and edge detector. *Proceedings of the 4th Alvey Vision Conference*, pages 147 – 151, 1988.

[HYG01] S. Pattanaik H. Yee and D. Greenberg. Spatiotemporal sensitivity and visual attention for efficient rendering of dynamic environments. *ACM Transactions on Graphics*, 20(1):39–65, 2001.

[IB98] Michael Isard and Andrew Blake. CONDENSATION Conditional Density Propagation forVisual Tracking. *International Journal of Computer Vision*, 29(1):5–28, 1998.

[IG89] G.W. Irwin and F.M.F. Gaston. A systolic architecture for square root covariance Kalman filtering. *Proceedings of the International Conference on Systolic Arrays*, pages 255 – 263, 1989.

[IKN98] Laurent Itti, Christof Koch, and Ernst Niebur. Model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(11):1254 – 1259, 1998.

[Itt00] L. Itti. *Models of Bottom-Up and Top-Down Visual Attention.* PhD thesis, California Institute of Technology, 2000.

[Jam90] W. James. *The Principles of Psychology.* Harvard University Press, 1890.

[Kai67] T. Kailath. The Divergence and Bhattacharyya Distance Measures in Signal Selection. *IEEE Trans. on Comm. Technology*, 15:52–60, 1967.

[Kal60] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME - Journal of Basic Engineering*, 82:35–45, 1960.

[KB61] R. E. Kalman and R. S. Bucy. New results in linear filtering and prediction theory. *Transactions of the ASME - Journal of Basic Engineering*, 83:95–107, 1961.

[KB01] T. Kadir and M. Brady. Saliency, scale and image description. *International Journal of Computer Vision*, 45(2):83 – 105, 2001.

[KD86] Sayfollah Kiaei and Uday B. Desai. Independent data flow wavefront array processors for recursive equations. In *Proceedings of the Twentieth Conference on Information Sciences and Systems.*, pages 429–435, Princeton, NJ, USA, 1986.

[KH91] S.-Y. Kung and J.-N. Hwang. Systolic array designs for Kalman filtering. *IEEE Transactions on Signal Processing*, 39(1):171 – 182, 1991.

[KIA02] Minseok Kim, K. Ichige, and H. Arai. Design of Jacobi EVD processor based on CORDIC for DOA estimation with MUSIC algorithm. *PIMRC Proceedings*, 1:120 – 4, 2002.

[KL99] J. Kuffner and J.C. Latombe. Perception-based navigation for animated characters in real-time virtual environments. *The Visual Computer: Real-time Virtual Worlds*, 1999.

[KU85] C Koch and S Ullman. Shifts in selective visual attention: towards the underlying neural circuitry. *Human neurobiology*, 4(4):219–227, 1985.

[Kun82] H. T. Kung. Why systolic architectures? *Computer*, 15(1):37 – 46, 1982.

[KY95] George J. Klir and Bo Yuan. *Fuzzy sets and fuzzy logic : theory and applications.* Prentice Hall, Upper Saddle River, N.J. ; London, 1995.

[KY01] S. Kameda and T. Yagi. A silicon retina calculating high-precision spatial and temporal derivatives. In *Neural Networks, 2001. Proceedings. IJCNN '01. International Joint Conference on*, volume 1, pages 201–205 vol.1, 2001.

[lap] http://www.netlib.org/lapack/.

[LBC06a] Yang Liu, Christos-Savvas Bouganis, and Peter Y. K. Cheung. A spatiotemporal saliency framework. In *Proceedings of IEEE International Conference on Image Processing (ICIP 2006)*, Atlanta, GA, Oct. 2006.

[LBC+06b] Yang Liu, Christos-Savvas Bouganis, Peter Y. K. Cheung, Philip H. W. Leong, and Stephen J. Motley. Hardware efficient architectures for eigenvalue computation. In *Proceedings of the conference*

on Design, automation and test in Europe, pages 953–958, Munich, Germany, 2006.

[LBC07] Yang Liu, Christos-Savvas Bouganis, and Peter Y. K. Cheung. Efficient mapping of a Kalman filter into an FPGA using Taylor expansion. *to appear in Proceedings of the 17th International Conference on Field Programmable Logic and Applications (FPL)*, Amsterdam, Netherlands, 2007.

[Lev91] Audie G. Leventhal. *The Neural Basis of Visual Function: Vision and Visual Dysfunction*, volume 4. Macmillan Scientific & Medical, Basingstoke, 1991.

[LL03] I. Laptev and T. Lindeberg. Space-time interest points. *Proceedings Ninth IEEE International Conference on Computer Vision*, 1:432 – 9, 2003.

[Low04] D.G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91 – 110, 2004.

[LS97] C.R. Lee and Z. Salcic. High-performance FPGA-based implementation of Kalman filter. *Microprocessors and Microsystems*, 21(4):257 – 265, 1997.

[MA03] T Moore and KM Armstrong. Selective gating of visual signals by microstimulation of frontal cortex. *Nature*, 421(6921):370–373, 2003.

[Man93] David M. Mandelbaum. Method for calculation of the square root using combinatorial logic. *Journal of VLSI Signal Processing*, 6(3):233 – 242, 1993.

[Mas98] D. Massicotte. A systolic VLSI implementation of Kalman-filter-based algorithms for signal reconstruction. *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing*, 5:3029 – 3032, 1998.

[Mea90] C. Mead. Neuromorphic electronic systems. *Proceedings of the IEEE*, 78(10):1629–1636, 1990.

[MM99] CJ McAdams and JHR Maunsell. Effects of attention on orientation-tuning functions of single neurons in macaque cortical area v4. *The journal of neuroscience*, 19(1):431–441, 1999.

[Mot93] BC Motter. Focal attention produces spatially selective processing in visual cortical areas v1, v2, and v4 in the presence of competing stimuli. *Journal of neurophysiology*, 70(3):909–919, 1993.

[MP85] J.J. Modi and J.D. Pryce. Efficient implementation of Jacobi's diagonalization method on the DAP. *Numerische Mathematik*, 46(3):443 – 54, 1985.

[MR98] Arien Mack and Irvin Rock. *Inattentional Blindness.* MIT Press, 1998.

[MZ03] Yu-Fei Ma and Hong-Jiang Zhang. Contrast-based image attention analysis by using fuzzy growing. *Proceedings of the ACM International Multimedia Conference and Exhibition*, pages 374 – 381, 2003.

[Not00] HC Nothdurft. Salience from feature contrast: additivity across dimensions. *Vision Research*, 40(10-12):1183–1201, 2000.

[opeet] http://opencvlibrary.sourceforge.net.

[OTCH03] A. Oliva, A. Torralba, M.S. Castelhano, and J.M. Henderson. Top-down control of visual attention in object detection. In *Proceedings. International Conference on Image Processing*, volume 1, pages 253–256, 2003.

[Pri82] M. B. Priestley. *Spectral analysis and time series.* Probability and mathematical statistics. Academic Press, London, 1982.

[PS02] C. Peters and C. O Sullivan. A memory model for autonomous virtual humans. In *Proc. Eurographics Ireland*, pages 21–26, 2002.

[RBH00] D. Ress, B.T. Backus, and D.J. Heeger. Activity in primary visual cortex predicts performance in a visual detection task. *Nature neuroscience*, 3:940–945, 2000.

[RD03] J.H. Reynolds and R. Desimone. Interacting roles of attention and visual salience in v4. *Neuron*, 37(5):853, 2003.

[ret] http://webvision.med.utah.edu/imageswv/schem.jpeg.

[RLS98] P.R. Roelfsema, V.A. Lamme, and H. Spekreijse. Object-based attention in the primary visual cortex of the macaque monkey. *Nature*, 395(6700):376–381, 1998.

[ROC96] R.A. Rensink, J.K. O'Regan, and J.J. Clark. To see or not to see: The need for attention to perceive changes in scenes. *International Journal of Psychology*, 31:257, 1996.

[Sch82] Robert Schreiber. Systolic arrays for eigenvalue computation. *Proceedings of SPIE - The International Society for Optical Engineering*, 341:27 – 34, 1982.

[Sch91] D.E. Schimmel. *Bit-level Jacobi-like Algorithms for Eigenvalue and Singular Value Decompositions.* PhD thesis, Department of Electrical Engineering, Cornell University, 1991.

[SDST99] D.C. Somers, A.M. Dale, A.E. Seiffert, and R.B.H. Tootell. Functional mri reveals spatially specific attentional modulation in human primary visual cortex. *Proceedings of the National Academy of Sciences of the United States of America*, 96(4):1663–1668, 1999.

[SH87] T. Sung and Y. Hu. Parallel VLSI implementation of the Kalman filter. *IEEE Transactions on Aerospace and Electronic Systems*, AES-23(2):215 – 224, 1987.

[SQS+00] G. Sandini, P. Questa, D. Scheffer, B. Dierickx, and A. Mannucci. A retina-like cmos sensor and its applications. In *Sensor Array and Multichannel Signal Processing Workshop. 2000. Proceedings of the 2000 IEEE*, pages 514–519, 2000. TY - CONF.

[SSA97] P.K. Sahoo, D.W. Slaaf, and T.A. Albert. Threshold selection using a minimal histogram entropy difference. *Optical Engineering*, 36(7):1976 – 1981, 1997.

[ST94] Jianbo Shi and C. Tomasi. Good features to track. *Proceedings 1994 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 593 – 600, 1994.

[TG80] Anne M. Treisman and Garry Gelade. A feature-integration theory of attention. *Cognitive Psychology*, 12(1):97–136, 1980.

[Tho76] C. L. Thornton. *Triangular Covariance Factorizations for Kalman Filtering.* PhD thesis, University of California at Los Angeles, School of Engineering, 1976.

[TRD99] R.D. Turney, A.M. Reza, and J.G.R. Delva. FPGA implementation of adaptive temporal Kalman filter for real time video filter-

ing. *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, 4:2231 – 2234, 1999.

[Tre03] Stefan Treue. Visual attention: the where, what, how and why of saliency. *Current opinion in neurobiology*, 13(4):428, 2003.

[TRK98] Jo Yew Tham, S. Ranganath, and A.A. Kassim. Highly scalable wavelet-based video codec for very low bit-rate environment. *Selected Areas in Communications, IEEE Journal on*, 16(1):12–27, 1998.

[WA93] J.Y.A. Wang and E.H. Adelson. Layered representation for motion analysis. *Proceedings. 1993 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 361 – 6, 1993.

[WLB03] Zhou Wang, Ligang Lu, and A.C. Bovik. Foveation scalable video coding with automatic fixation selection. *Image Processing, IEEE Transactions on*, 12(2):243–254, 2003.

[WOL94] JM WOLFE. Guided search 2.0: A revised model of visual search. *Psychonomic bulletin review*, 1(2):202–238, 1994.

[Wvd91] Waerden and B. L. van der. *Algebra.* Springer-Verlag, New York; London, 1991. translated by Fred Blum and John R. Schulenberger. Vol.1.

[xil] *http://www.xilinx.com.*

[Yeh88] H.-G. Yeh. Systolic implementation on Kalman filters. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 36(9):1514 – 1517, Sept. 1988.

[YJ90] S. Yantis and J. Jonides. Abrupt visual onsets and selective attention: voluntary versus automatic allocation. *Journal of experimental psychology. human perception and performance*, 16(1):121–134, 1990.

[ZLD03] S. Zahnd, P. Lichisteiner, and T. Delbruck. Integrated vision sensor for detecting boundary crossings. In *Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium on*, volume 2, pages 376–379, 2003.

[ZS06] Yun Zhai and Mubarak Shah. Visual attention detection in video sequences using spatiotemporal cues. In *Proceedings of the 14th annual ACM international conference on Multimedia*, pages 815–824, New York, NY, USA, 2006.

[ZSV04] Hua Zhong, Jianbo Shi, and M. Visontai. Detecting unusual activity in video. *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2:819 – 26, 2004.