The University of Western Australia
Faculty of Engineering, Computing and Mathematics
School of Electrical, Electronic and Computer Engineering
Centre for Intelligent Information Processing Systems

FPGA Implementation of the Lane Detection and Tracking Algorithm

ROOZBEH ANVARI

Supervised by Professor Dr Thomas Braunl

Master of Engineering Thesis

Roozbeh Anvari

42 Loftus St

Nedlands WA 6009

28^{th} MAY 2010

The Dean,
Faculty of Engineering, Computing and Mathematics
The University of Western Australia
35 Stirling Highway
CRAWLEY WA 6009

Dear Sir,

I submit to you this dissertation entitled "FPGA Implementation of the Lane Detection and Tracking Algorithm" in partial fulfillment of the requirement of the award of Master of Engineering in "Information and Communication Technology".

Yours faithfully,
Roozbeh Anvari.

Abstract

The application of the Image Processing to Autonomous Drive has drawn a significant attention in the literature and research. However the demanding nature of the image processing algorithms conveys a considerable burden to any conventional real-time implementation. Meanwhile the emergence of FPGAs has brought numerous facilities toward fast prototyping and implementation of ASICs so that an image processing algorithm can be designed, tested and synthesized in a relatively short period of time in comparison to traditional approaches. This thesis investigates the best combination of required algorithms to reach an optimum solution to the problem of lane detection and tracking while is aiming to fit the design to a minimal system. The proposed structure realizes three algorithms namely Steerable Filter, Hough Transform and Kalman Filter. For each module the theoretical background is investigated and a detailed description of the realization is given followed by an analysis of both achievements and shortages of the design.

Acknowledgments

Many thanks to my supervisor Professor Dr Thomas Braunl for his invaluable support and guidance throughout the course of this project.

I would like to gratefully express my thanks and love to my family Vida, Javad and Hormoz who have always been supporting me unconditionally.

Contents

List of the Tables

Chapter 1

Introduction

Nowadays the advent of fast platforms has given a more significant role to image processing. There is no wonder any more that there are image processing units embedded in such small platforms like cellphones and cameras, however the demanding nature of the image processing algorithms is still putting a barrier to realize a wide range of existing advancements in theory into the world of real time implementation. On the other hand the great amount of resources available on the FPGAs besides the flexibility to test and prototyping an ASIC that it offers, has made FPGA an ideal choice for realizing image processing algorithms in the real time and autonomous drive is no exception. A brief literature survey agrees that although there are numerous advancements in trying to either developing new algorithms or to refining existing ones, only a little attention is drawn to the realization.

1.1 Thesis scope

This thesis aims to design the required modules of a real-time platform capable of distinguishing between desired lane roads and the rest of the peripherals. Indeed there are many issues that have to be addressed. Firstly the image must be preprocessed so that all the data irrelevant to the goal of the algorithm are distinguished. The level of noise, undesired lanes parallel to the desired ones and the light under which the experiment is conducting are all determining factors that must be considered into account. The next step is to obtain a formal description of the present lanes. Hence it must be decided which algorithm is an optimum choice so that it gives a more accurate description while demanding less resources. Finally it comes to the step where the platform must come to a decision what to do when the outcome is deviated due to either noise or the lack of visual information.

1.2 Eyebot M6

It is desired to tailor the design to fit the resources available on the EyeBot-M6 which is a mobile robot designed in the University of Western Australia capable of image processing tasks. There is a small sized FPGA of the *Xilinx's Spartan-3E family*[1] embedded on the board that is intended to be the host platform for this design. One major challenge of this project would be on how to cope with limited resources on the *Spartan-3E.* The image stream is provided by two color cameras named OV6630 from *OmniVision* [2]. The embedded cameras have a maximum resolution of 352 * 288 pixels and can reach a maximum frame rate of 50 fps. In order to enforce the cameras to generate a 8 bit stream, they are required to be fed a frequency of 18 Mhz. In order to save the on-chip resources a 18 Mbit SRAM is interfaced to the FPGA that has the capacity to hold 10 frames simultaneously. The FPGA platform is connected to the CPU with an asynchronous bus interface called Variable Latency I/O. The main processor on the EyeBot-M6 is a PXA-255 [3] running at 400MHz embedded in a *Gumstix* Board [4] in which there are 64MB of SDRAM, 16MB of flash and a bluetooth module embedded on the board.

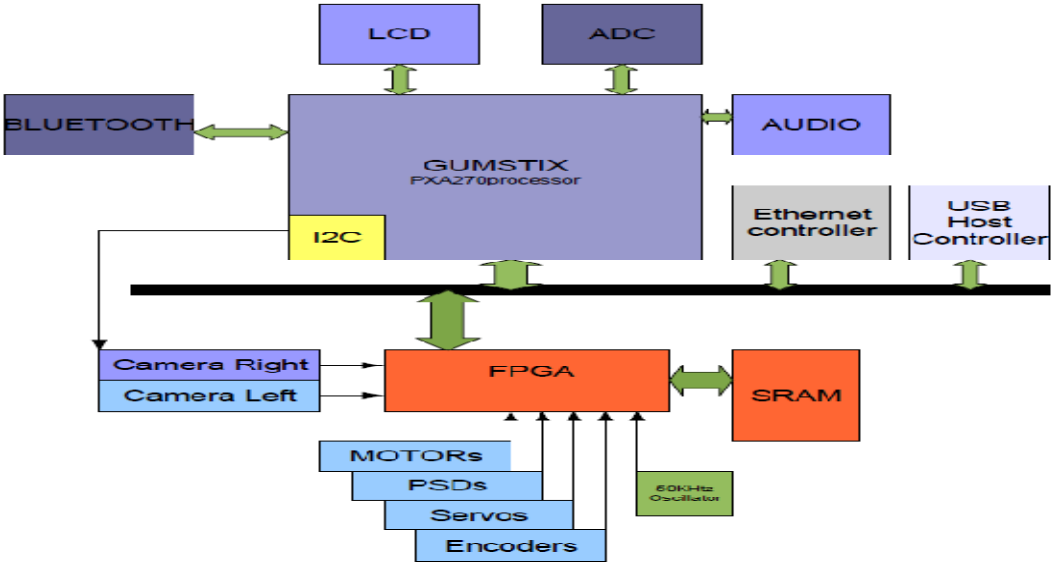


Figure 1.1 : An overview of the EyeBot-M6

1.3 Thesis outline

chapter 2 conveys the results of the conducted survey on the conventional approaches already practiced by the expert toward implementing lane detection on the FPGA

chapter 3 expands the theory of the Steerable filter used for edge detection and image refinement. Then explains the advantageous and disadvantageous brought to the design by applying this approach. Finally the implemented structure on the FPGA is explained, following by an analysis of the performance of the implemented hardware.

chapter 4 investigates two conventional approaches in the literature for a hardware implementation of lane detection followed by the theory of both approaches. The theory behind the Hough transform and its implementation are given in detail. An analysis of the performance of the design follows.

chapter 5 discusses the need for a tracking module to be added to the design. The theory of the Kalman filter and the practiced approach toward transferring the matrix calculations to the hardware is given following by a discussion on the difficulties faced in the implementation and an analysis on the results.

chapter 6 summarizes the implemented design and its achievements. Explains both achievements and shortages of the design and gives suggestions for a future work.

Chapter 2

Literature Survey

This thesis aims to implement the consisting modules of a lane detection and tracking system on the FPGA. However a survey on the literature proved that this topic is a very active and significant area of research and numerous papers are investigating the boarders of knowledge in this domain. Since the thesis aims to implement the design on the FPGA platform, firstly a survey on the conventional FPGA structures that are related to the topic of this thesis is conducted. In the following a brief summary of all the different approaches toward lane detection and tracking follows. Furthermore the literature review relevant to the three main topics of this thesis namely Steerable Filters, Hough Transform and Kalman Filter will be investigated in the corresponding chapters.

2.1 Literature Survey on the Conventional Design Methodologies on the FPGA for Image Processing

The first paper[5] that was found to be very related to the topic of this thesis is investigating the implementation of the lane detection problem on the FPGA. Indeed this paper is proposing a solution for two out of the three concerns of this thesis. This paper is applying the Canny filter for edge detection and is suggesting the Hough Transform as a solution for lane detection.

The canny edge detector is implemented on the FPGA using 2D convolution combined with a moving window structure. The following steps are taken to detect the edges, the image is smoothed by Gaussian convolution, then in order to obtain gradient information the image's derivatives in both directions is calculated using the Prewitt operator. Once the image is edge detected, the Hough transform is applied to approximate the present lines in the image.

Unfortunately the article describes the implementation procedure in a very brief and inadequate approach. Actually the focus of the implementation was to apply pipelining as much as possible in such way that one output pixel per clock edge is obtained. Cascaded FIFOs are used to implement the moving window. A $5 * 5$ window structure for smoothing and two $3 * 3$ windows

for derivation are applied. The result of the canny edge detector on each pixel is presented as a one bit structure where 1 represents an edge and 0 represents a none-edge. By applying this approach the authors are trying to map the whole image to a considerably smaller space so that the whole image is available at once on the FPGA. Calculations of the sine and cosine functions are implemented using look up tables. The whole project is synthesized on a *Xilinx's virtexII* [1] and the exact number of the consumed logic resources is given for each processing block. This implementation has achieved a speed of 44MHz. In this project the whole code is written in *Matlab* [13] and is simulated on the *MODELSIM* [16] prior to synthesis.

This paper contributed a lot to the advancement of the thesis. Even though the article is just briefing about their applied algorithm and reveals no information about implementation, the general idea of possibility of implementing the Hough transform on the FPGA was adopted from this paper. On the contrary this paper is applying a very demanding algorithm for edge detection i.e. Cany filter that is relatively too much more complex in comparison to the concept of using the Steerable Filter applied to this thesis. Finally the idea of implementing sin and cos values is derived from this paper as will be discussed in chapter 4.

2.1.2 General Purpose Image Processing System on the FPGA

Even though it is out the scope of this thesis to implement a general purpose image processing system, it is worth to investigate the expert attitude toward satisfying the requirements of such design. The following two papers are investigating such generic solution that can fit any image processing system. By applying such approaches, the problem of ASIC design reduces to implementation of high level languages like C and assembly on the FPGA.

The second paper[6] that is being discussed here is suggesting a totally different methodology in comparison to those of the first paper. This paper is implementing a Sobel Filter as a 2D moving window structure. The main idea of this design is to make the modules as generic as possible so that the camera and RAM interface modules are independent of the image processor unit's structure. This design is using the embedded multipliers available on the FPGA. Unfortunately it became apparent within this thesis that utilizing the embedded modules is not always possible and sometimes it is required to re-implement some existing modules. The image processing module on this design is interfaced to the outer world by means of handshaking with both the camera and memory interface. It is also remarkable that this design is applying truncation to

adjust the numbers. Although this idea is accepted and applied to this thesis but finally it became apparent that truncation noise can cause great distortion as will be discussed in the following chapter.

The third investigated design[7] is aiming to realize an Embedded Image Processing System on the FPGA by utilizing the Microblaze [8] soft processor as the main approach suggested by Xilinx. Indeed the importance of this article is that it represents a shortcut towards designing the whole embedded system on the FPGA using the automation facilities embedded in the XILINX development studio. Microblaze is the soft core processor that can be adjusted to meet the design's requirements. Since this software is professionally designed and tested by the manufacturer and since according to the Xilinx it is the best approach towards utilizing the Xilinx FPGA's resources, this article is highlighted here. The Microblaze soft-processor is a 32 bit Harvard RISC architecture. It originally includes a 3stage pipeline and 32 general purposed registers besides an ALU, shift unit, and 2 levels of interrupt. Based on the requirements of the project extra modules like barrel shifter, floating point unit, cashes, exception handling facilities, debug logic, and many other blocks can be added to the soft core processor. In this paper different kinds of interface between modules on the FPGA as a pretested freeware are explained. Microblaze is not only offering a professional interface between modules but also between the FPGA and the PC if required. Although this article only introduces the structure itself and does not explain how filters are implemented on the Microblaze, the results of implementing Soble and Wavelet filters on the image are presented.

2.1.3 Hardware/Software Co-design Approaches for Line, Detection

This paper [9] represents another approach toward designing ASICs on the FPGA. The whole procedure is written in ImpulseC[10] which enables one to prototype the hardware as fast as possible, since once the algorithm is written in ImpulseC, the compiler automatically generates the FPGA's structure, interface between the FPGA and peripherals and the software operating at the host processor. Although ImpulseC has many advantages like automating the hardware design procedure and generating parallel structures, it is not a free ware. Instead systemC[11] can

be applied that offers the same facilities but a less convenient compiler. In this article the Robert kernel is applied in order to derivate the image and then Hough transform is used to detect the lines. The most significant contribute of this paper addresses the amplitude of the gradient. Although the gradient magnitude must be calculated as

$$\sqrt{(\partial I/\partial x)^2 + (\partial I/\partial y)^2} \quad (1)$$

in practice the following equation is being used as a proper approximation

$$|G| = |G_x| + |G_y| \quad (2)$$

Equation (2) was found to be very useful and is applied to this thesis in the steerable module

2.1.4 FPGA design using Matlab/Simulink

This article [12] highlights the roll of high level design by introducing the Matlab/Simulink based approaches. Although there are obvious advantages using this technique but the throughput of the design is dependent to the sophistication of the libraries at hand, and the cost and availability of development tools. Recently both major FPGA vendors i.e. Altera[15] and Xilinx, have begun to support Matlab development environment since it is the main platform for DSP development. To use the benefits of high level design the free web based version is not adequate and the full tool subscription is required to support DSP builder by Altera or System-Generator by XILINX. This article investigates the efficiency of this approach when applied to a Xilinx or an Altera board.

When compared in size and efficiency there are no advantages in using one library over another to choose between Altera or Xilinx. In either case there are enough building blocks to design almost any DSP system without generating any custom block. In the following the article explains the advantages and disadvantages of using either Xilinx or Altera when compared for cost, design flow, Simulink support and design flow implementation. It is concluded that both vendor's are almost identical except when it comes to porting from Simulink to the physical

layer where Altera is significantly superior since all the design procedure must be done once again for Xilinx while it can be generated for Altera directly from the Simulink environment.

2.2 A Literature Survey on the Lane Detection Problem

According to the comprehensive survey conducted by Joel C. McCall and Mohan M. Trived [17], all the driver assistance system design literature 1984–2006, follow a very similar design flow. First a model for the road and vehicle is proposed. This model varies between a simple straight line, Clothoid or Spline. Next a set of sensors are used to gather the environmental information used for extracting features like motion flow, edge, texture etc. These extracted features in combination with the actual road model are used to estimate the lane's position. Finally a model is required for the moving vehicle to refine these estimates. Only a few design have been excepted from this methodology. Two cases are cited as an exception, first one is a combinatory control strategy used by Taylor et al[18] in which various control strategies are coupled together, second is the ALVINN [20], autonomous land vehicle in a neural network, in which the neural network "directly incorporates the feature extraction into the control system with no tracking feedback".

Road Modeling is necessary for "eliminating the false positives via outlier removal". Parallel lines, piecewise constant parallel lines, curvatures like splines and even maps generated by dGPS are examples are applied road models. The applied model is determined based on the expected degree of sophistication e.g. a spline model is too much complex for a system intended for only highways.

Road marking extraction seems to be the most determining phase. Since the road and lane marking vary greatly, applying only a single feature extractor is challenging. Edge based techniques work properly with solid and segmented lines. On the contrary if there are many extraneous lines this approach is very likely to fail. In this case i.e. extraneous lines, the frequency domain methods like what is used in LANA [24] is more effective. On the other hand the frequency based system is limited to the diagonal lanes. In addition there are cases in which

the lane position is based on an adaptive road model e.g. RALPH system [25]. This approach can fail in case the road texture is not constant.

In order to improve the extracted features and estimates post processing is mandatory. There are various approaches towards post processing namely Hogh Transform [26],[27], attenuation or enhancement of features using orientation [23] or likelihood [21],[24], culling features using stereo vision [22], dynamic programming [28], and finally cue scheduling [29]. In all these approaches some features are amplified and are chosen to be fed into position tracking module while some extraneous features are attenuated or eliminated due the system's structure.

The last phase is tracking. There are two common tracking techniques namely Kalman filtering [18],[19] and particle filtering [29],[30]. There are combinatory structures with a more complex structure like [31] as well. In all these approaches feature extraction and position tracking are combined in a closed feedback loop.

Chapter 3

Edge Detection

3.1 Edge Detection theory

As a formal definition, any step discontinuity is regarded as an edge. Hence, traditional approaches of edge detection are simply a process of finding the local maxima in the first derivative or the zero crossings in the second derivative by convolving the image by some form of linear filter that approximates either first or second derivatives [1]. While an odd symmetric function can approximate the first derivative, the second derivative is approximated by an even symmetric function.

In fact, in the discrete domain, the gradient of the image can be simply calculated by taking the difference of the gray values in the image. This procedure is equal to convolving the image by the mask$[-1,1]$. The obvious disadvantage of this simplification is that it is not clear which pixel the result is associated to. There are various approaches to this issue among which the following filter masks offer a first derivative solution

	$\partial I/\partial x$	$\partial I/\partial y$
Robert	$\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$
Prewitt	$\begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{pmatrix}$
Sobel	$\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{pmatrix}$

Figure 3.1 : First Derivative Filter Masks

To point the gradient map of the frame, in each case the magnitude of the gradient map is calculated by

$$\sqrt{(\partial I/\partial x)^2 + (\partial I/\partial y)^2} \quad (1)$$

The main concern in derivative edge detection is the effect of the noise since the local maxima due to the white noise can mask the real gradient maxima due to an edge. That is why it is required to convolve the image with a smoothing function e.g. Gaussian function, so that the effect of the white noise is minimized. Both of the mentioned operators namely the gradient and the Gaussian are linear and it is computationally more efficient to combine them. Therefore if the image and its gradient are indicated by I and G then

$$(I \otimes G)' = I \otimes G' \quad (2)$$

3.2 Steerable Filter Theory

The input image stream usually contains lanes in various directions which are redundant to the problem of autonomous drive. Therefore it is required to apply oriented edge detectors to different parts of the image such that the unwanted lanes are suppressed. One approach to do so is to apply many versions of an edge detector each of which differing from others in the angle, to different parts of the image. This approach obviously consumes huge amount of extra logic and is not reasonable to implement, although is quite fast. A more efficient approach is proposed by [33],[34] in which required filters of arbitrary orientation can be expressed as a linear combination of a set of basis filters. One then only needs to know how many filters are required and what interpolation function satisfies the requirements. This class of oriented filters is referred to as Steerable Filters. A function $g(x, y)$ is steerable if it can be written as a linear sum of rotated versions of itself. This constraint can be expressed as

$$f^{\theta}(x, y) = \sum_{i=1}^{M} k_i(\theta) f^{\theta_i}(x, y) \quad (3)$$

where $f(x,y)$ is an arbitrary isotropic window function, $k_i(\theta)$ are the interpolation functions and M is the number of basis functions required to steer a function $g^{\theta}(x,y)$. In order to investigate what functions are steerable, function $f(x,y)$ is expressed in polar coordinates where $r=\sqrt{x^2+y^2}$, $\varphi=\arg(x,y)$. If f is expandable in a Fourier series in polar angle φ we have

$$f(r,\varphi)=\sum_{n=-N}^{N} a_n(r)e^{in\varphi} \tag{4}$$

The number of the required basis functions and the basis functions themselves are determined as a results of the following theorems [33]

***Theorem* 1**: the steering condition (1) holds for functions expandable in the form of (2) if and only if the interpolation functions $k_i(\theta)$ are solutions of

$$\begin{pmatrix} 1 \\ e^{i\theta} \\ \vdots \\ e^{iN\theta} \end{pmatrix} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ e^{i\theta_1} & e^{i\theta_2} & \cdots & e^{i\theta_M} \\ \vdots & \vdots & \vdots & \vdots \\ e^{iN\theta_1} & e^{iN\theta_2} & \cdots & e^{iN\theta_M} \end{pmatrix} \begin{pmatrix} k_1(\theta) \\ k_2(\theta) \\ \vdots \\ k_M(\theta) \end{pmatrix} \tag{5}$$

***Theorem* 2**: Let T be the number of nonzero coefficients $a_n(r)$ for a function $f(r,\varphi)$ expandable in the form of (5). Then, the minimum number of basis functions sufficient to steer $f(r,\varphi)$ by (1) is T.

Therefore if a function $f(r,\varphi)$ is expandable in Fourier series in polar angle, it is steerable. The number of the none zero coefficients determines the minimum number of the required basis functions required to steer a function. Finally in order to obtain the basis functions equation (5) must be solved.

For instance the two dimensional symmetric Gaussian function G that is used in edge detection is described as

$$G(x,y)=e^{-(x^2+y^2)} \tag{6}$$

by expressing the first derivative of $G(x,y)$ in polar coordinates we have

$$G_1^{0^0}(r,\varphi) = -2re^{-r^2}\cos(\varphi) = -re^{-r^2}(e^{i\varphi} + e^{-i\varphi}) \quad (7)$$

Obviously $G_1^{0^0}(r,\varphi)$ has two none zero coefficients in its Fourier decomposition in polar angle φ, therefore only two basis functions would suffice to synthesize G_1^{θ} out of its basis functions. Now, one needs to obtain the interpolation functions by solving the equation (5) for two basis functions that results to

$$\left(e^{i\theta}\right) = \begin{pmatrix} e^{i\theta_1} & e^{i\theta_2} \end{pmatrix} \begin{pmatrix} k_1(\theta) \\ k_2(\theta) \end{pmatrix} \quad (8)$$

Solving equation (6) is straight forward and if we pick $\theta_1 = 0^{\circ}$ and $\theta_2 = 90^{\circ}$ then the interpolation functions are obtained as

$$k_1(\theta) = \cos(\theta) \quad (9)$$

$$k_2(\theta) = \sin(\theta) \quad (10)$$

And finally the first derivative of the two dimensional symmetric Gaussian function G that is widely used in image processing is expressed in terms of its basis functions as

$$G_1^{\theta} = \sum_{j=1}^{2} k_j(\theta) G_1^{\theta_j} = \cos(\theta)\, G_1^{0^{\circ}} + \sin(\theta)\, G_1^{90^{\circ}} \quad (11)$$

Now it comes to the point to combine the concept of a separable and steerable filter. Let the $\boldsymbol{n}$th derivative of a Gaussian in the x direction to be written as G_n and let $(\ldots)^{\theta}$ represent the rotation operator. The first x derivative of a Gaussian is

$$G_1^{0^{\circ}} = \frac{\partial}{\partial x} e^{-(x^2+y^2)} = -2xe^{-(x^2+y^2)} \quad (12)$$

The same function when rotated 90° is

$$G_1^{90^{\circ}} = \frac{\partial}{\partial y} e^{-(x^2+y^2)} = -2ye^{-(x^2+y^2)} \quad (13)$$

and $G_1^{90^{\circ}}$ are separable and can be described as

$$G_1^{0^\circ} = f_1(x) \cdot f_2(y) \tag{14}$$

$$G_1^{90^\circ} = f_1(y) \cdot f_2(x) \tag{15}$$

$$f_1(x) = -2xe^{-x^2}\ ,\ f_2(x) = e^{-x^2} \tag{16),(17}$$

Freeman[33] suggests a sample spacing of 0.67 in the range which leads to the sampled 9 taps of table 1.

tap#	*f1*	*f2*
0	*0.0*	1
1	*-0.5445*	*0.6383*
2	*-0.2833*	*0.1660*
3	*-0.0450*	*0.0176*
4	*-0.0026*	*0.0008*

Table 3.1 : Filter taps corresponding to the basis functions of the Separable Gaussian filter

	filter in x	*filter in y*
G_1^{90}	*f1*	*f2*
G_1^{0}	*f2*	*f1*

Table 3.2 basis functions of the Separable Gaussian filter in each direction

3.3 FPGA implementation of the Steerable Filter

Prior to combining the filter with the rest of the image processing modules it is required to implement the filter structure itself. According to the fact that filter coefficients are none integers and all have a fraction part, it was required to apply fixed point arithmetic. Indeed a more precise design needed a floating point module to take care of the fractions but in order to keep the design concise it was decided to apply the Fixed Point logic.

3.3.1 Fraction Length

In order to implement a $7 * 7$ filter it was required to differentiate between filter taps 2 and 3, that is equivalent to realizing fixed point numbers with a precision that can differentiate a decimal value of 0.01 i.e. the third tap of f_2. Hence, considering the fact that $2^{-7} + 2^{-8} \approx 0.01$, at least 8 bits are required to be considered for the fraction part. According to the fact that there are 11 signed bits considered to keep the gradient value of the edge detected pixels, it was required to implement multipliers with a word length of $8 + 11 + 1 = 20$ bits, such implementation enforced pipelining and more strict timing constraints. Therefore it was decided to realize the steerable filter as a $5 * 5$ window and to realize a larger window only if the precision of design were not satisfying enough. However, the results of the physical implementation proved that no more precision was required. Moreover, after analyzing the truncation error, as is depicted in table 3.3, it became apparent that by the current constants taps, there is no difference between implementing a fraction length of 5 or 6. That is why a fraction length of 5 is adopted for the filter taps.

tap#	f_1 required	Fraction length	f_1 Binary implementation	f_1 Truncation error	error %	f_2 required	f_2 Binary implementation	f_2 Truncation error	error %
0	*0.0*	*5*	*.00000*	*0*	0	1	*.11111*	*.03125*	*3*
1	*0.5445*	*5*	*.10001*	*.01325*	*2.43*	*0.6383*	*.10100*	*.0133*	*2*
2	*0.2833*	*5*	*.01001*	*.00205*	*0.72*	*0.1660*	*.00101*	*.00975*	*5*
0	*0.0*	*6*	*.000000*	*0*	0	1	*.111111*	*.0156*	*1.5*
1	*0.5445*	*6*	*.100010*	*.01325*	*2.43*	*0.6383*	.101000	*.0133*	*2*
2	*0.2833*	*6*	*.010010*	*.002050*	*0.72*	*0.1660*	*.001010*	*.00975*	*5*
0	*0.0*	*7*	*.0000000*	*0*	0	1	*.1111111*	*.0079*	*0.7*
1	*0.5445*	*7*	*.1000101*	*.00543*	*0.99*	*0.6383*	.1010001	*.00175*	*2*
2	*0.2833*	*7*	*.0100100*	*.00205*	*0.72*	*0.1660*	*.0010101*	*.0019*	*1*
0	*0.0*	*8*	*.00000000*	*0*	0	1	*.11111111*	*.0039*	*0.3*
1	*0.5445*	*8*	*.10001011*	*.00153*	*0.28*	*0.6383*	.10100011	*.0015*	*2*
2	*0.2833*	*8*	*.01001000*	*.00205*	*0.72*	*0.1660*	*.00101010*	*.0019*	*1*

Table 3.3 : A comparison between different values of truncation noise

3.3.2 Implementing the Filter Structure

In order to implement the steerable filter, equation (11) must be realized. There are various remarkable points regarding to realizing this equation that will be addressed in the following. Firstly it is required to access the values of $G_1^{0^\circ}$ and $G_1^{90^\circ}$ for each pixel. There are two possible approaches to address this issue. First one is to implement a two dimensional moving window that is moving across the five rows. The second approach is to decompose $G_1^{0^\circ}$ and $G_1^{90^\circ}$ to their basis filters. So that a two dimensional convolution is decomposed to two one dimensional convolutions. Indeed both approaches are equivalent in terms of time requirements, but the second approach demands smaller number of logic gates. The importance of this approach might not be obvious for smaller filters like the Sobel filter, but for a larger filter with a large number of none zero taps such conversion means huge amount of savings in terms of logic gates. It is worth to note that in Soble filter, the process of multiplying each tap involves a single shift, while in a filter whose taps are none integer values it is a must to separate basis functions, otherwise unreasonable amount of multipliers are required to implement the Fixed Point

multiplications. A comparison between the separable and non-separable moving windows depicted in figures 3.2 and 3.3 gives a clear perception of the need for designing a moving filter as a separable structure if applicable.

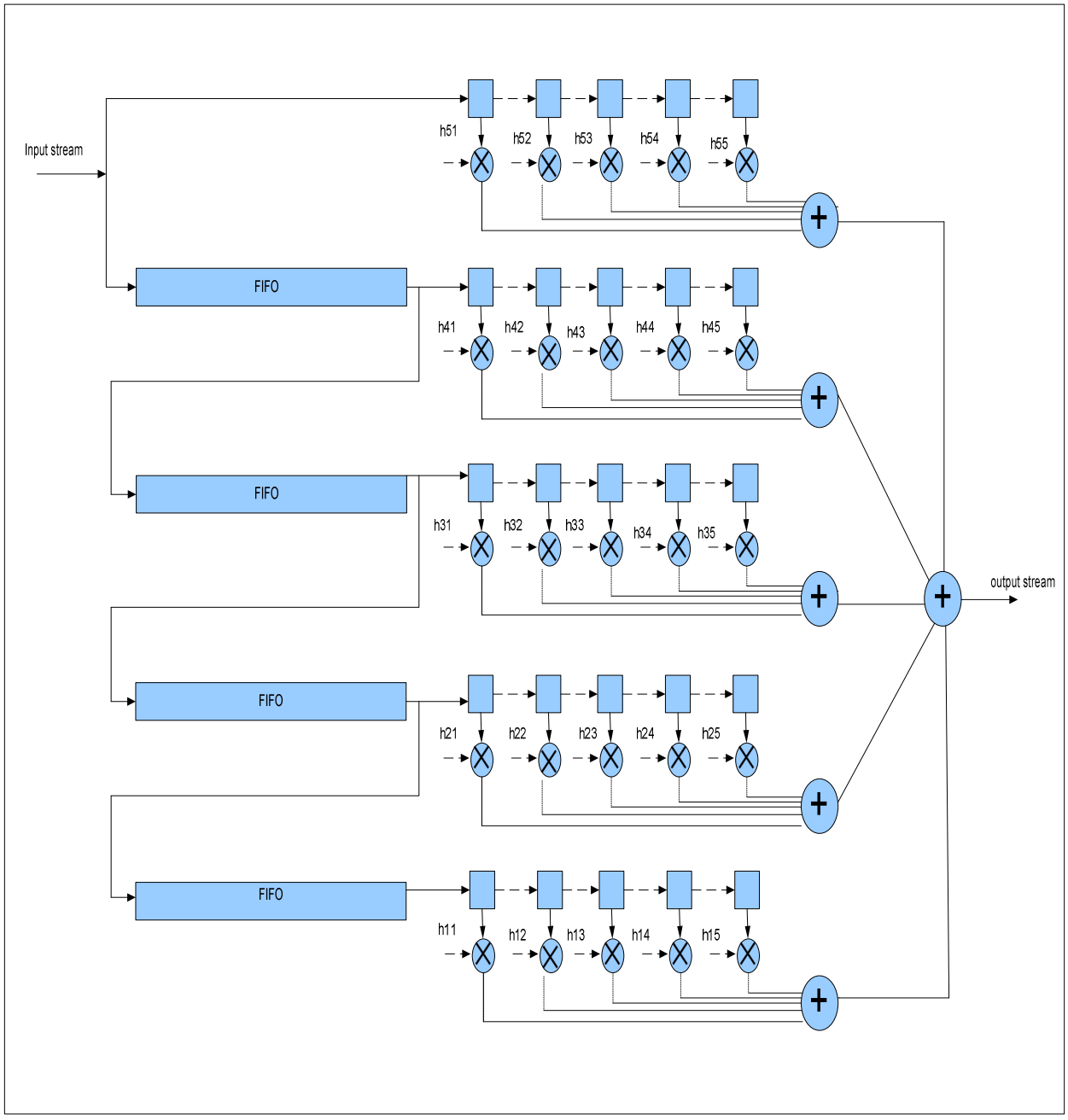


Figure 3.2 : 2D structure for the non-separable moving window

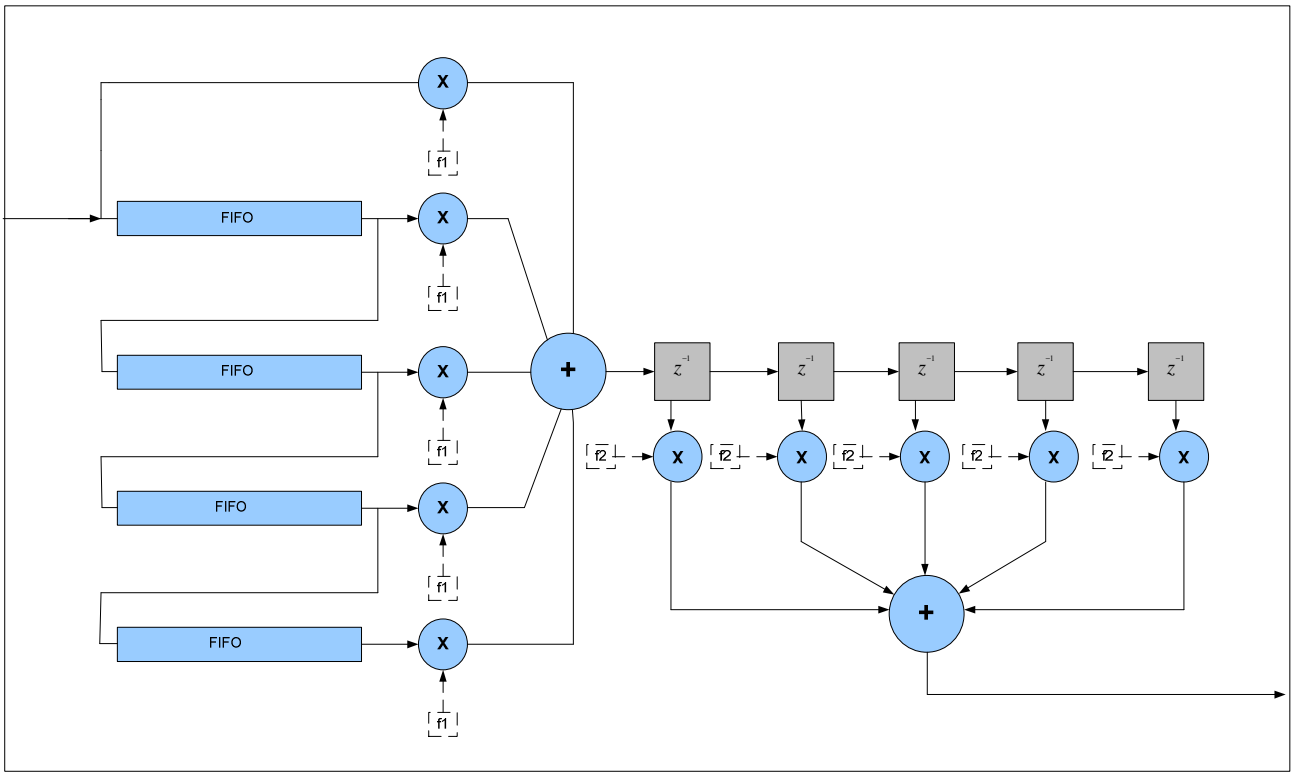


Figure 3.3 : 2D structure for the separable moving window

The second issue addresses the demand for a structure able of keeping the value of 5 rows. Obviously the number 5 is related to the window size of the implemented steerable filter. Hence if it was opted to implement a $6 * 6$ window, it was required to simultaneously access the values of the 6 rows. This adds one more reason to the previous discussion on the size of the filter, justifying why only 5 taps out of 9 are implemented i.e. if another 2 taps were applied, it was required to keep another two rows on the FPGA that would waste a lot of resources. Since there are negative filter taps, it is required to convert the unsigned bit stream coming out of the camera interface to signed values. Therefore prior to any further calculation, the input data stream coming from camera interface is extended to 9 signed bits.

The proper structure for realizing time delay is a FIFO. Each FIFO must be of the length of $352 * 9\ bits$ so that it can hold the value of the whole row. Although it is straight forward to implement a FIFO as a large shift register, Xilinx strongly suggests the use of the IP cores for

implementing such large FIFO structures on its FPGAs. These FIFOs are generated using the internal Block RAMs available on the FPGA. The number of words in each FIFO is an integer multiplicand of 2. Therefore in order to implement a FIFO of the length of 352 it is required to choose a FIFO of the length of 512. The logic of these FIFOs is programmable so that after 352 pixels the first output is generated. However an area equivalent to $160 * 9\ bits$ will be wasted.

There is another remarkable issue regarding to the borders of the frame. As mentioned earlier, it is a requirement for a window structure to access all the pixel values within a window. Otherwise an irrelevant value is obtained that is useless. Hence it is required to exclude the borders. This issue is very easily handled by implementing counters for tracking the number of pixels. Therefore the filtering process only starts after 4 rows and 3 pixels are entered the steerable filter. Figure 3.4 depicts this issue. Once end of the frame is reached the remainder of the data residing within the FIFOs is redundant and must be cleared. This problem is easily solved by keeping the track of the filtered pixels by updating the row and column indexes. Furthermore a comparison between figures 3.4 and 3.7 confirms that the boundary rows and columns of the figure 3.7 are distorted as in confirmation with the area indicated in figure 3.4.

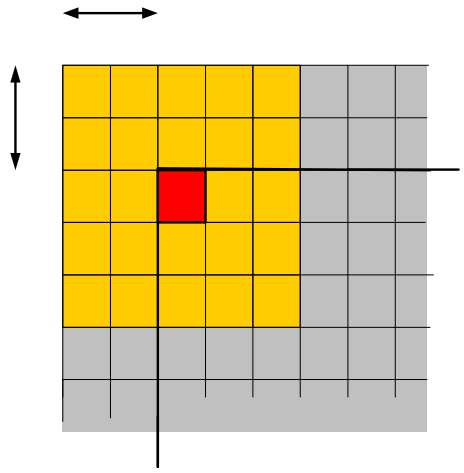


Figure 3.4 Dealing with the boarder pixels

3.3.3 Applying Orientations and Calculating Trigonometric Values

Now that the separable moving window structure is implemented, one more step is required to generate the output. Indeed at this stage it is required to alter the orientation of the filtered window by applying the proper basis functions. In fact if the design was tailored to reach the best possible efficiency for the autonomous drive problem, it is recommended that processing half of the image is always redundant since it contains no relevant information about the road lanes. However in order to keep the design as generic as possible, as is the goal of this thesis while emphasizing the autonomous drive problem, the whole pixels of the frame are processed. Figure 3.4 depicts the desired orientations that are applied to different districts of the image.

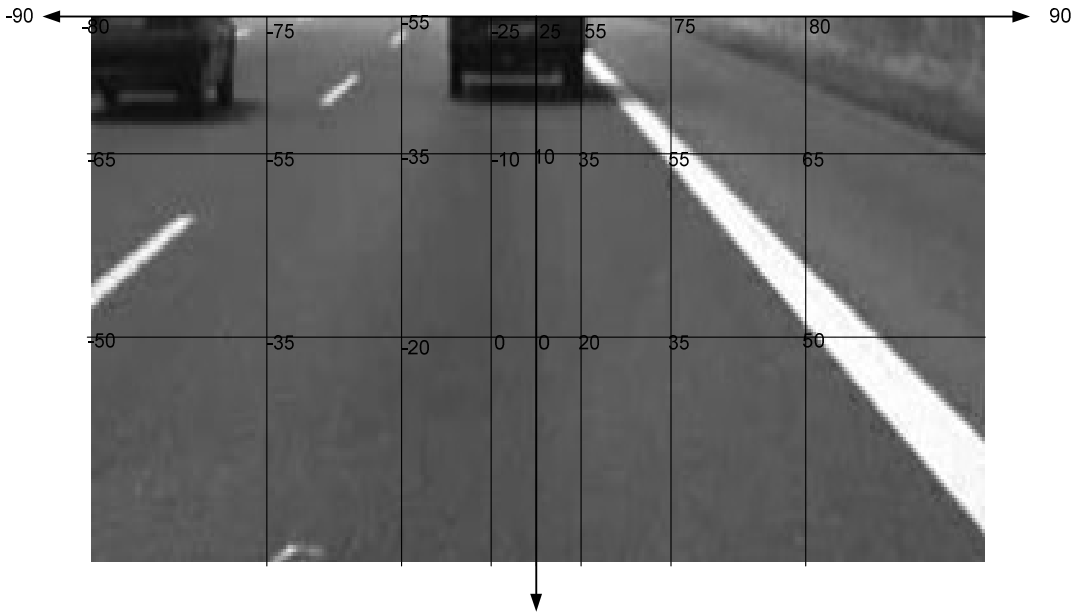


Figure 3.5 : applied orientations to the Steerable Filter

By altering the orientation of the edge filters to the directions depicted in figure 3.5, one makes sure that the edges in the desired directions are strengthened while the others are weakened. The need for this practice becomes more clear when noticing the fact that in a non ideal environment too many edges are present due to the adjacent vehicles, multi parallel lanes in a highway or the obstacles around the road, that are all redundant to the problem of autonomous drive. As

discussed earlier in the chapter the main goal of implementing steerable filters was to address this problem since the main advantage of the steerable filter is just to steer the filter to the desired orientation.

In order to steer the filters, it was required to implement $\sin(\theta)$ and $\cos(\theta)$ functions. According to the figure 3.5 there are only 8 angles for which the values of $\sin(\theta)$ and $\cos(\theta)$ are required to be implemented so that a generic function is redundant to the design. Thereby the values of the required functions are pre-calculated and saved into a look up table as is summarized in table 3.4. In fact for the Hough transform module implemented in the next chapter another look up table for trigonometric functions were required, but in order to make the modules generic and independent it was decided to opt for redundancy. Once again it is easy to refer to the required sin and cos values for a certain range of pixels by applying the proper control logic to check the row and column indexes.

degree	$\cos(\theta)$		$\sin(\theta)$	
80	0.1736	00101	0.9848	11111
75	0.2588	01000	0.9659	11110
55	0.5736	10010	0.8192	11010
25	0.9063	11101	0.4226	01101
65	0.4226	01101	0.9063	11101
35	0.8192	11010	0.5736	10010
10	0.9848	11111	.01736	00101
50	0.6428	10100	0.7660	11000

Table 3.4 a fixed point representation of the required trigonometric values

In order to implement the required multiplication in $\cos(\theta) * G_1^{0^\circ}$ and $\sin(\theta) * G_1^{90^\circ}$, three approaches were investigated. The best approach suggested by *Xilinx* is to apply the embedded fast multipliers on the chip itself. This approach is applied to the Hough module in the next

chapter. However it became apparent that due to the small size of the *Spartan 3E,* the adjacent Block RAMs and multipliers are accessible only through the same predefined fixed route on the chip. According to the fact that there are only 20 multipliers available on the chip among which 2 are used for other components on the chip, and considering the fact that there are 18 multiplications required per steerable filter and that none of the remainder multipliers adjacent to one of the used Block RAMs for implementing the required FIFOs can be used, it became apparent that fixed point multiplications must be implemented manually.

There are two approaches investigated towards implementing the fixed point multiplications. Firstly it was decided to implement a fully pipelined multiplier by applying the shift and sign extension structure. Once the final value is obtained the fraction part is truncated. The reason is that it was decided to limit the steerable filter to the integer numbers. This simplification proved quite efficient for image processing purposes where truncation noise is only affecting the numerator. In case the truncation is applied to the denominator, as is the case in the Kalman Filter design in chapter 5, a small amount of truncation would make a huge error in the result. That is why there is a very large fixed point representation i.e. 19bits, chosen for the Kalman filter registers to decrease the truncation noise as much as possible. The fixed point multiplier is tested with a wide range of positive and negative numbers and it proved to operate accurately.

However after more investigation it became apparent that it is possible to drop the multipliers by applying simple shifts. The reason behind that is the fact that firstly the filter taps are fixed, and secondly there are only a few none zero bits present in each filter tap, otherwise this approach would dramatically decrease the efficiency. Having calculated the values of $\cos(\theta)\, G_1^{0^\circ}$ and $\sin(\theta)\, G_1^{90^\circ}$ there is only one more step left to generate the gradient value of the pixel by adding these two values.

3.3.4 Determining the Register Sizes

The final remark is about the need to determine the proper word lengths. Indeed there are two approaches in a more comprehensive design toward designating the proper word length to the applied registers when the design is larger than to be measured and estimated manually. The more professional solution advised by Xilinx is to apply the design to the AccelDSP[35] in order to calculate the required bit length of each register present in the design by considering the range of applied data input and the operation that the register is involved in. Unfortunately this software is not a freeware and must be purchased. This solution requires the whole design to be ported and simulated in MATLAB. This approach would be a perfect solution for any design if the scheme is originally designed in SIMULINK. Since in this thesis all the modules are developed straight away out of the scratch, it did not worth to redesign the whole procedure in MATLAB due to the time constraints.

Hence it was required to intake extra bits to make sure that no overflow is occurring. First of all, all the unsigned 8-bit input data stream coming from camera interface is casted to 9-bit signed values so that the pixel values are capable of keeping a sign without loosing a bit. Furthermore it was opted to extend the data length to 11-bits prior to multiplying by filter taps. The reason is that after multiplying by either 4 taps of f_1 or 5 taps of f_2 , all the 4 or 5 resultants are summed together to form the required input to the next stage. Indeed all these filter taps are less than 1, but there is a probability that their sum can exceed the 10 bits. When it comes to fixed point multiplications at the steering part, the 11-bit integer operand is multiplied by a number consisted of only a fraction of length 5 bits. The fixed point result is truncated afterwards to fit the 11-bit as an integer number. At the final stage when all the calculations are done, the values corresponding to $\cos(\theta)\, G_1^{0^\circ}$ and $\sin(\theta)\, G_1^{90^\circ}$ are each 11-bit long, therefore it is required to truncate the values to 8 bits. It was observed that there are some negative values present as the output of either $\cos(\theta)\, G_1^{0^\circ}$ or $\sin(\theta)\, G_1^{90^\circ}$, therefore in try to scale back the pixel values to the gray scale it was required to add a positive value so that all the negative values are shifted to the range of gray scale. Finally it is unavoidable to truncate the extra 2 least significant bits. Figure 3.7 depicts the diagram summarizing the overall implemented structure for steerable filter.

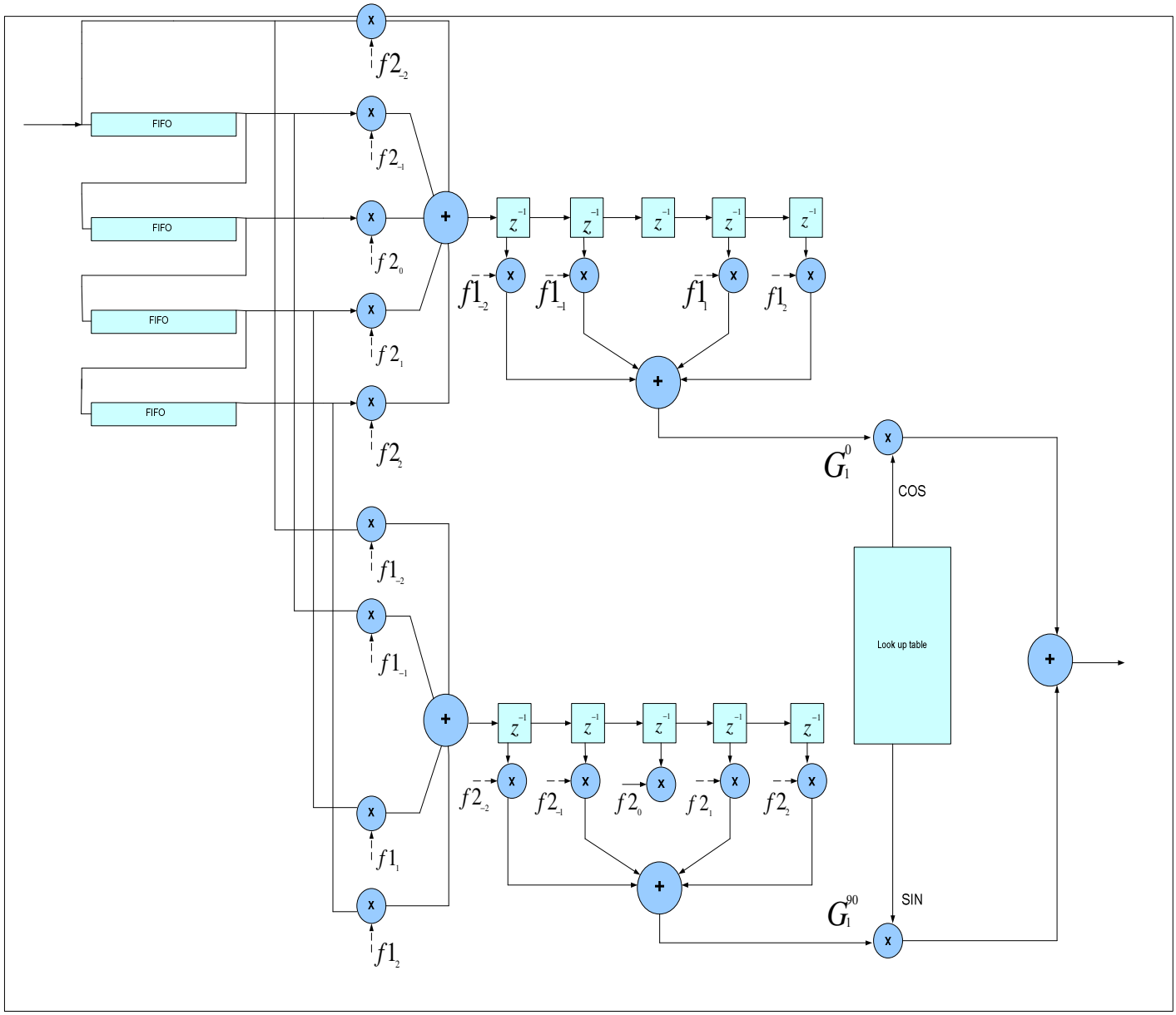


Figure 3.7 : The block diagram of the implemented steerable filter

3.4 Results and Analysis of the Design

The designed steerable filter is fully synthesized and routed on the *FPGA Spartan 3E* present on the Eyebot-M6 and proved to work properly. The following figures are obtained by simulating the proposed structure for steerable filter on the eyebotM6.

a)

b)

c)

Figure 3.8 a) the original image b) image representing the $\cos(\theta)\, G_1^{0^\circ}$ component c) image representing the $\sin(\theta)\, G_1^{90^\circ}$ component

suppressed. It is apparent that the peripheral lanes in the other bands of the highway are not detected any more. In addition lanes due to the other obstacles i.e. other vehicles and other

obstacles are dramatically suppressed. On the other hand the lane in the desired direction is heavily empowered.

On the other hand it was observed that some distortion is introduced to the boundaries of the image. Although the road lane detection is the beneficiary of this issue, since all the distortion occurs at boundaries where no lane is expected, but it is not acceptable for a generic design. After further investigation it is suggested that the introduced distortion is related to the truncation noise due to the short size of the fraction length. Such issue did not occur in designing the other two modules since a relatively large fraction length were introduced to the design.

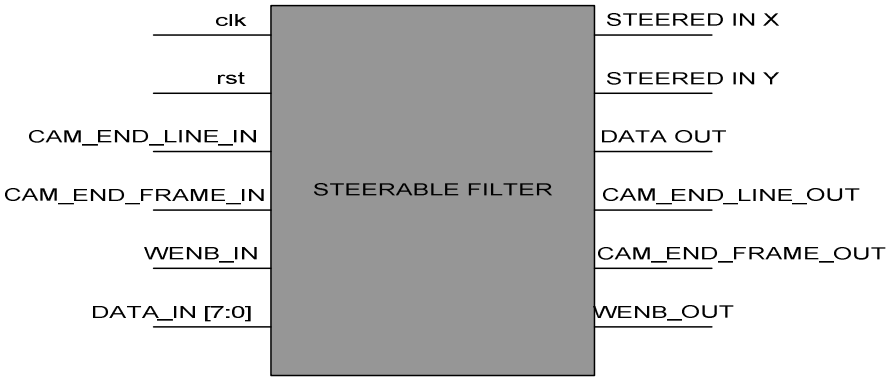


Figure 3.9: Steerable Filter Module

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Total Number Slice Registers	553	1,920	28%	
Number used as Flip Flops	535			
Number used as Latches	18			
Number of 4 input LUTs	1,445	1,920	75%	
Number of occupied Slices	958	960	99%	
Number of Slices containing only related logic	958	958	100%	
Number of Slices containing unrelated logic	0	958	0%	
Total Number of 4 input LUTs	1,567	1,920	81%	
Number used as logic	1,445			
Number used as a route-thru	122			
Number of bonded IOBs	42	66	63%	
Number of RAMB16s	4	4	100%	
Number of BUFGMUXs	1	24	4%	
Average Fanout of Non-Clock Nets	2.92			

Figure 3.10: summary of the utilized resources on the FPGA

Chapter 4

Lane Detection

4.1 Literature Survey

There are four approaches widely used in the literature to identify lines in the image. Hough Transform[36],[37] has been used for a long time and is applied to a wide variety of applications. In fact, the Hough Transform is a very demanding algorithm in terms of computational requirements and in most of the cases either search space is limited to certain parts of the image or is combined with some heuristic approaches to reduce the calculations. In the second approach [38] the image is divided to certain number of tiles and each tile is searched for the lane passing the center of mass in that tile. Even though this approach is not used widely in the literature and is applied to a few number of problems, the implementation of the algorithm in C language on the Eyebot-M6 proved that it is a more proper choice for a Floating Point implementation. However as will be discussed after later in the chapter, this thesis recognized the Hough transform as a better choice for hardware implementation. The third approach that is widely used in the literature aims to find the present lanes in the image by interpolating a variety of different Splines to the interest points in the image. Among all the investigated Splines B-snakes [39] found to be more accurate and popular. Finally the statistical approaches [40] have drawn a lot of attention among the literature. The last two approaches have gained a lot more in comparison to the first two, but obviously are very expensive in terms of computational demands and have never been the subject of a hardware implementation.

4.2 Hough Transform Theory

An object in the image can be described by various number of mathematical functions describing its boundary. Although applying complex functions is theoretically possible, the demanding computational requirements make the physical implementation impossible. The original Hough transform is patented after Paul Hough in 1962 and suggests an efficient approach to describe the boundaries of the object of interest. In this approach much of the information present in the image is not used since the edge image is firstly converted to a bi-valued space by applying a threshold value to the pixel gradients. Therefore a pixel is regarded as a potential boundary pixel only if its associated gradient is higher than a certain value. In fact the need for determining such threshold values is the main disadvantage of the Hough transform as will be addressed in the analysis of the implemented structure. A single lane in the Cartesian space is formulated by

$$y - mx - c = 0 \tag{1}$$

where m and c are corresponding to the slope and intercept of the lane. One single point in the Cartesian space can be considered to belong to a whole family of lines with different m and c values. A single line in the m-c space can represents all the possible m and c values corresponding to all the lines that can pass that point. Therefore a set of points in the Cartesian space can be mapped to a set of lines in the m-c space and that forms the main idea behind the Hough approach towards extracting the present lines in the image. Figure 4.1 depicts the concept of the Hough Transform.

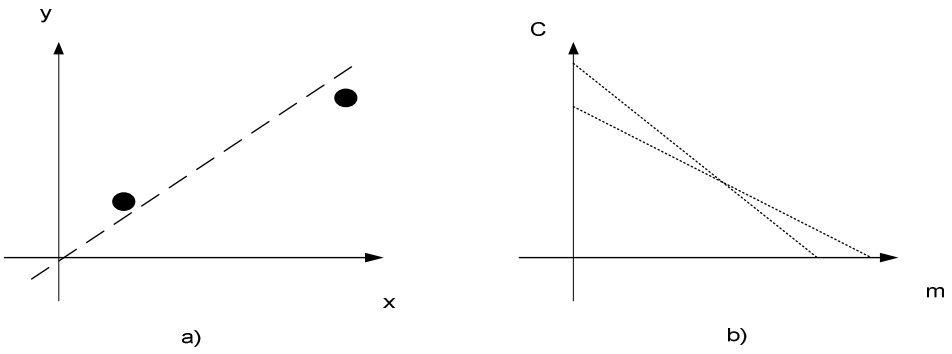
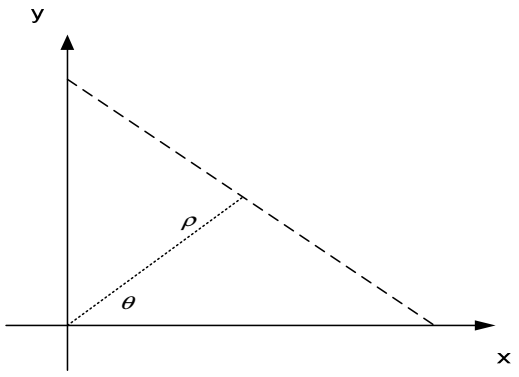


Figure 4.1: Map between Cartesian and Hough space

At this point all the edge points located on a single straight line in the x-y space will intersect on a single point in the m-c space. Therefore the problem of finding the straight lines in the x-y space reduces to finding a single point in the m-c space. The presented formulation of a straight line does not satisfy computational requirements since the values of m can tend to infinity. A line can be represented by its shortest distance and its orientation by



$$\rho = x\,cos(\theta) - \; ysin(\theta) \tag{2}$$

Hence for each single edge point whose gradient is above the threshold a procedure is invoked in which firstly the corresponding shortest distance i.e. ρ_n associated which each orientation θ_n is found, secondly the index number representing the calculated distance is obtained and finally the ρ-θ space associated with ρ_n and θ_n is incremented by one. Substituting the values of ρ and θ in (2) indicates that a line in the x-y space is mapped to a sinusoid in the ρ-θ space.

$$(x_i, y_i) \rightarrow \; M = \sqrt[2]{{x_i}^2 + {y_i}^2}\,, \cos(\varphi) = {x_i}/{M}\,, \sin(\varphi) = {y_i}/{M} \tag{3}$$

$$\rho = Mcos(\varphi)\cos(\theta) - \; M\sin(\varphi)\sin(\theta) = \; M\cos(\varphi - \theta) \tag{4}$$

In practice, there are numerous number of lines passing each point, hence the ρ-θ space is divided to discrete points each representing a range of points in its neighborhood. For instance if 100 points are determined to represented the orientation, each represents a vicinity of 3.6° degrees. Once all the edge points in the x-y space are mapped to their corresponding sinusoids in

the ρ-θ space, the problem of the lane detection is equivalent to finding the local maximums in the ρ-θ space.

4.3 Lane Detection: Second Approach

In this approach the present lines in the image are obtained by calculating the first and second moments of the image [37],[38]. In fact this approach is looking for the minor and major principal axis of an object. That is why in this approach it is required to isolate the objects prior to lane detection otherwise the moments of different object will affect each other. It is easy to deal with this problem in a high level language like C by preprocessing and segmenting the objects. But for an embedded realization such preprocessing is not reasonable and usually the image is segmented to equally distributed tiles. Finally a clustering method is required to combine all the locally found coordinates. The algorithm can be summarized as follows. Firstly Area of the object i.e. the whole tile in practice, is obtained by the 0^{th} moment of the object as

$$A = \sum\sum b(x,y) \tag{5}$$

where $b(x,y)$ equals to one for pixels whose gradient is above threshold. The center of mass, denoted by $(\bar{x},\bar{y})$ represents the center of the desired object e.g. line and is calculated by

$$\bar{x} = \frac{\sum\sum xb(x,y)}{\sum\sum b(x,y)} \tag{6}$$

$$\bar{y} = \frac{\sum\sum yb(x,y)}{\sum\sum b(x,y)} \tag{7}$$

At this step the axis of minimum inertia of the object passing from the center of mass is required to indicate the orientation. This is the axis of least 2^{nd}moment [37]. This aim is satisfied by finding a line for which the following integral is a minimum

$$I = \sum\sum r^2\, b(x,y) \tag{8}$$

where r is the perpendicular distance from (x, y) to the desired line, but prior to this step it is needed to confirm a proper representation to parameterize a straight line. For the same reasons discussed in the previous entry, the line is represented by equation (2). The equation (8) is solved in [37] in detail and the orientation corresponding to the major principal axis i.e. θ_1 and minor principal axis i.e. θ_2 are obtained by

$$\theta_1 = \frac{1}{2}\, atan2\left(\frac{b}{\sqrt{b^2+(a-c)^2}}\ ,\frac{a-c}{\sqrt{b^2+(a-c)^2}}\ \right) \tag{9}$$

$$\theta_2 = \frac{1}{2}\, atan2\left(\frac{-b}{\sqrt{b^2+(a-c)^2}}\ ,\frac{c-a}{\sqrt{b^2+(a-c)^2}}\ \right) \tag{10}$$

$$a = \sum\sum \acute{x}^2 b(\acute{x}\,, \acute{y}) \tag{11}$$

$$b = 2\sum\sum \acute{x}\acute{y} b(\acute{x}, \acute{y}) \tag{12}$$

$$c = \sum\sum \acute{y}^2 b(\acute{x}, \acute{y}) \tag{13}$$

$$\acute{x} = x - \bar{x} \tag{14}$$

$$\acute{y} = y - \bar{y} \tag{15}$$

the constants $a\,, b\,, c$ are called the second moments and θ_1 represents the direction of the line that we are looking for. Both approaches are implemented by C language on the Eyebot-M6 and the results will be compared together at the end of the chapter.

4.4 Comparison between two algorithms

In this part it is justified why Hough Transform is opted as a better choice for hardware implementation by giving three reasons against the second approach. The first limitation for a fixed point implementation is due to the truncation error. It is easy to count that there are four divisions involved in the second approach. According to the fact that divisions are more sensitive to small modifications, a small amount of truncation error in the denominator introduces a considerable amount of truncation noise that will propagate to the next stages. The second reason

is that although Hough Transform is a very demanding algorithm in terms of computational demands, the second approach is more demanding. In case parallelism is properly applied to the problem, equations $(6), (7)$ can be calculated in one trace and equations $(11), (12), (13)$ can be calculated in the second trace. Therefore in case the Hough Transform is implemented properly so that it searches only certain parts of the image, Hough Transform is expected to perform faster. The third reason is related to the need for calculating tan^{-1}. Indeed there is no fixed point solution toward obtaining trigonometric calculations unless expanding them to their equivalent series that makes the design so complex for a fixed point realization.

4.5 Proposed methodology

The standard Hough Transform is a demanding algorithm in terms of both time and search space. However by restricting the range of the lanes for which the algorithm is either updating the Hough Matrix or is searching for a lane, the algorithm could be fitted to the real time implementation requirements. A line is parameterized by equation (2). There are two choices for the range of possible orientations. First choice is to have $\pi \leq \theta < 0$ and ρ can hold both positive and negative values. It is also possible to limit ρ to the positive values but the orientations vary in the range $0 \leq \theta < 2\pi$. Here it was opted to choose the first approach. for a certain reason that will be justified by following the fixed point implementation of the Hough Transform in the next part.

Two FSMs are implemented for each window to realize the Hough Transform in order to detect the road lanes. Once the whole image is edge detected by the steerable filter and is written to the SRAM, the lane detector module is triggered by a signal to detect the local lanes corresponding to the bottom left and bottom right windows of the image. The first FSM in each module fetches the values corresponding to the gradient of each pixel by putting the address of each pixel on the address bus. On the current design, each word in the

SRAM simultaneously holds the values of $\cos(\theta)\, G_1^{0^\circ}$ and $\sin(\theta)\, G_1^{90^\circ}$. Once a word is read to the Hough module sum of the $cos(\theta)\, G_1^{0^\circ}$ and $sin(\theta)\, G_1^{90^\circ}$ is compared to a threshold value in order to distinguish between an edge and a non edge. Indeed by comparing the pixel values with

a threshold only when it is required, there is no need to pre-threshold the image. by applying the Hough Transform to various video streams it became clear that the distribution of the road lanes in the left and right windows are not identical and two different threshold values are required to be applied to the design. Overall, a value of 50 for distracted lines and a value of 80 for continuous lanes were obtained. For the rest of the rest of the cases the proper value seems to always sit somewhere in this range.

Since the design aims to address the road lane detection, it is important to note that no lane above the horizon is required to be searched. In addition the lanes that has a slope larger than $160^0$are not desired and are usually due to either the noise or the other obstacles in the image frame. According to the fact that the pixel values located at the side bars of the image are always very likely to be affected by noise and that the steerable filter has had a destructive effect to this area, this first and last 10 pixels of each row are excluded for lane detection.

Finally it was observed that extremely vertical lanes are never likely to occur due to the conventional camera angles. Therefore some more clock would be saved by excluding the pixels corresponding to the vertical lanes. The remained area is indicated in Fig 4.2 By applying the mentioned hypothesis to all the videos available at hand in the lab, it become clear that this assumption always holds true as long as the road is not curved, and all the desired lanes are always located within this area. Therefore all the desired lines for the left window are located in the range $90 < \theta < 150$ and all the desired lanes in the right window are located in the range $30 < \theta < 90$.

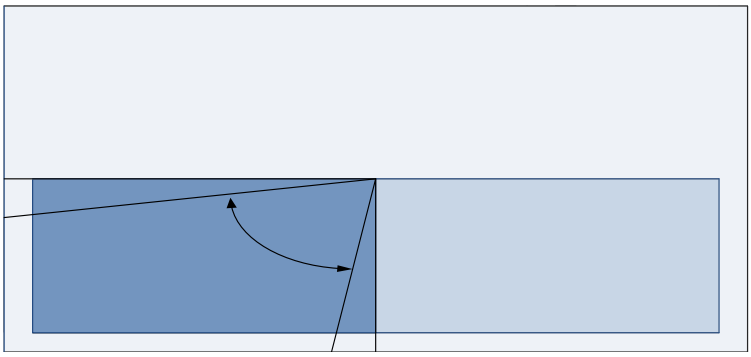


Figure 4.2: The area required to be searched for road lanes

4.6 FPGA implementation of the Hough Transform

According to the complexity of the required logic for the Hough transform, it was concluded that this module cannot be designed as a concurrent module as the steerable filter and a FSM structure would better fit the design. Therefore each module i.e. Hough transform for the left and right windows will be associated two FSMs, one to create the Hough matrix and the second to extract the desired line describing the major orientation axis of the lane. Figure 4.3 depicts the top view of the implemented Hough transform module

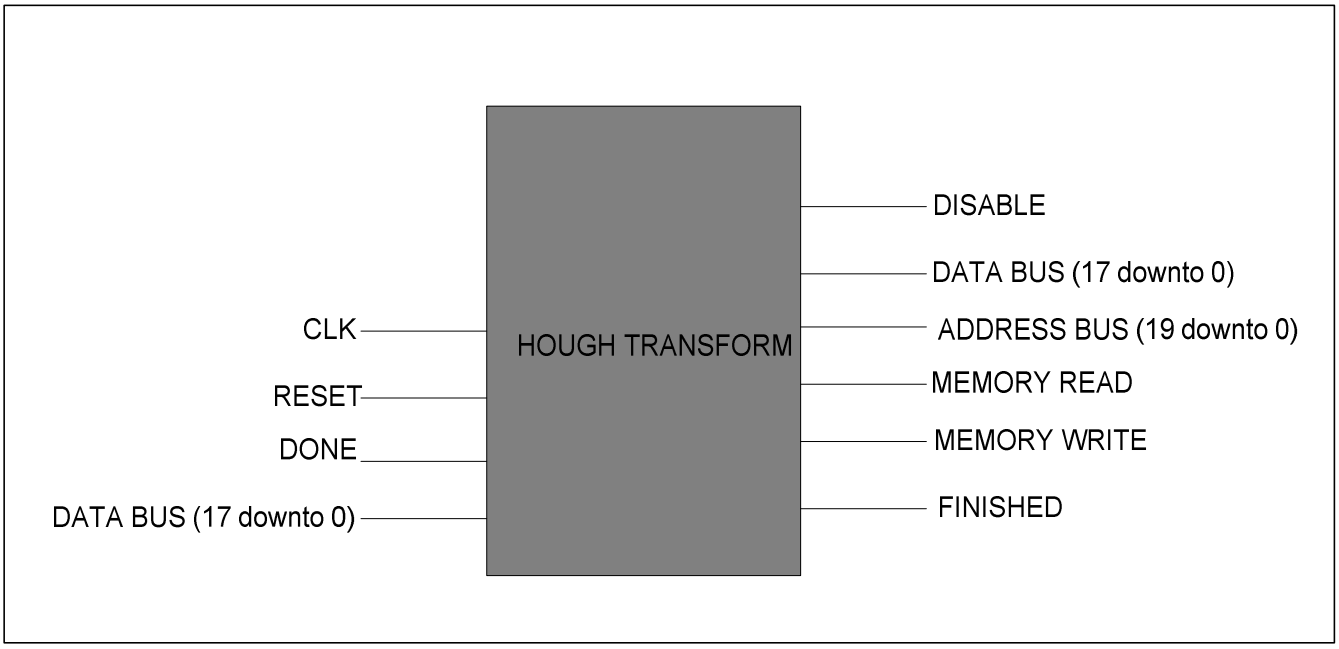


Figure 4.3: Top View of the implemented Hough Transform

As mentioned earlier for the ideal case only the indicated area in Figure 4.2 is required to be searched and updated but in order to make the design quite generic, it was decided to apply the Hough Transform to all the left and right window areas that can easily be refined in a future implementation.

Figure 4.4 depicts the first FSM that was designed to create the Hough transform out of the image gradient pixels in the SRAM. The FSM stays in the initial state as long as no Done signal is detected. Actually this signal is generated by the module that writes the edge detected pixels to the memory. So that once the whole image is written to the SRAM this signal is designated. As soon as the FSM detects this signal it enters the phase of calculating the address of the first pixel to be thresholded. Prior to this step it is required to stop the steerable filter from altering the image residing in the SRAM. Therefore an output *disable signal* is devised to put the other modules on hold during the lane detection phase. As the first step of the algorithm, it is required to calculate the address of the pixels located in the desired area as is indicated in figure 4.5, and to put the address on the address bus to be read from SRAM. As mentioned earlier it was decided to combine the thresholding and lane detection together so that a considerable amount of clock is saved. In the case of an edge the distance ρ of all the lanes passing from the edge-detected coordinates corresponding to all the θ values in the range 90° and 150° for the left window and all the θ values in the range 30° and 90° for the right window are incremented by one which forms the Hough Matrix in the SRAM.

The second FSM is triggered after the last pixel of the window is processed. The second FSM is following a quite simple logic in comparison to the first FSM. By conducting various experiments, It became clear that the Hough Matrix must be searched *column by column* so that the first local maximum would represent the *closest* lanes in the right and left side of the vehicle. According to the fact that each column is representing one single orientation, by searching the Hough matrix corresponding to the left window from the first column to the last column, the corresponding orientation is changing from 90° to 150°. Therefore only the first lane next to the vehicle will be detected. For the right window the search direction is in the opposite way and columns must be searched from end to fist, hence the range of values will alter from 90° to 20° and the outer lines will not be detected. This simple role made a huge advancement to the algorithm by excluding the outer lanes that are always present in the environment. . Figure 4.5 totally describes the applied logic to the second FSM

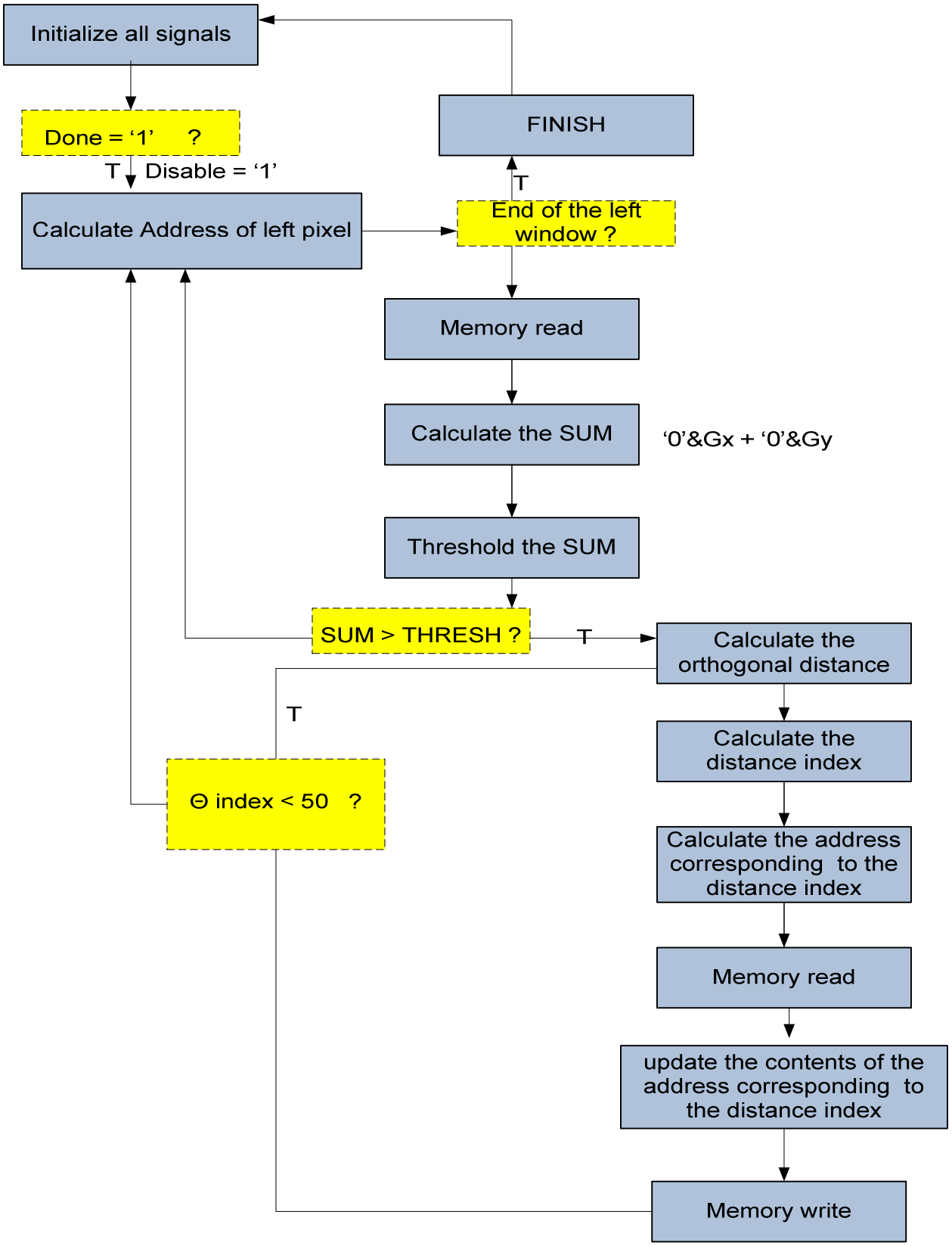


Figure 4.4: Top View of the implemented Hough Transform

According to the figure 4.7 there are 12 states implemented in the first FSM, but in order to put some more safety margins for the memory interface it was decided to include one more dummy state after each memory access that has increased the number of states to 16. At this stage this scheme was not required but it was devised to enable the module to work in higher frequencies when is used on a more robust platform like VirtexII. Indeed it was decided to make all the designed modules in this thesis as generic as possible so that they can be used as a black box in all the future projects. Otherwise each time the design must be changed regarding to the applied clock rate.

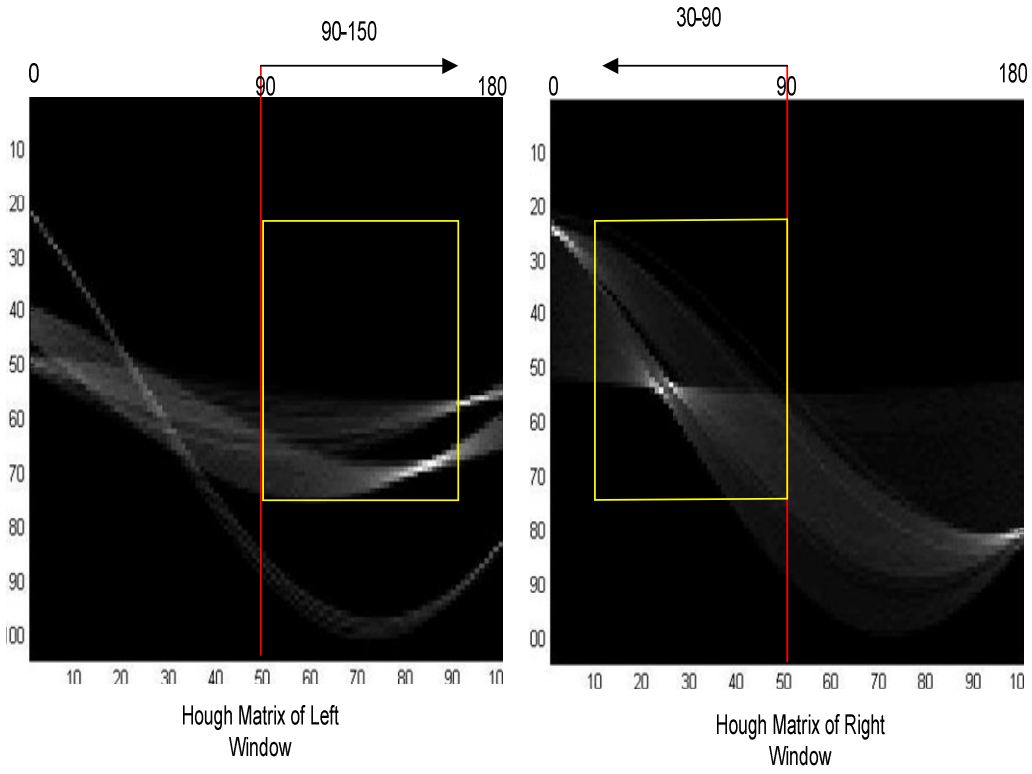


Figure 4.5: The implemented logic in the second FSM of the Hough Transform only searches the indicated area and in the specified direction in a column by column fashion

4.7 Achievements and Analysis of the Results

In order to verify the proposed structure and to emphasize the fact that the implemented algorithm is quite less demanding when implemented on the FPGA, the proposed Hough structure is implemented in MATLAB, C and VHDL. Results are presented and discussed in the following.

Floating point implementation is used as the first step to verify the algorithm itself. Proper values for thresholding both the edge detected frame and the Hough matrix are obtained through applying various video streams to the simulation on MATLAB.

Through different experiments it became clear that the Hough transform is very sensitive to this threshold values and even a very small variation from the proper value will lead to a wrong detection. This issue makes the design quite sensitive to noise and is regarded as the main disadvantage of the Hough transform. In addition it became clear that the speed of this algorithm is a variable of the light, noise and threshold. As presented results in table 4.1 confirm, the speed of the algorithm is heavily dependent to the determined threshold value. Indeed choosing a higher threshold will decrease the number of edge points therefore fewer iterations per edge is required, but this approach usually led to a deviation in the detected lane.

Figure 4.6.a depicts the Hough Transform as is present on the SRAM. There are some points remarkable here. Firstly as was desired the Hough space is not complete in each case and only is updated for half of the possible orientations. Secondly it is obvious that the obtained transformation is not as smooth as a floating point transformation. Indeed the Hough space seems to be jagged that is a direct effect of the truncation. As mentioned earlier, truncation has been applied to the design very often and in a next implementation it is suggested that truncation must be replaced by rounding. Figure 4.6.b depicts the Hough space on the SRAM after being thresholded in which the depicted point is corresponding to the detected lane.

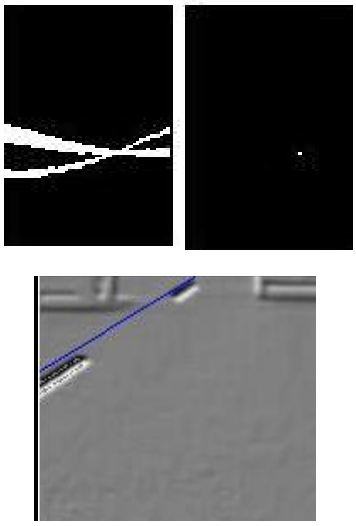


Fig.4.6 (a)Hough Matrix on the SRAM(b) Thresholded Hough Matrix on the SRAM(c) the left lane as detected on the FPGA

In order to get a measure of the efficiency of the hardware design the proposed algorithm is implemented and tested on both C and VHDL. The lane detector module is tested and verified by applying the edge detected frames out of both Sobel and Steerable filters and is simulated offline. Various experiments confirmed that the module has a very high accuracy unless the threshold value is not chosen properly. Indeed it was required to alter the threshold value even in different times of the day due to the variation of the light.

Edge detection	**Freq**	**Lane detection Algorithm**	**SPEED WHEN A LANE IS DETECTED**	**SPEED WHEN NO LANE IS DETECTED**	**Thresh Value for Image**	**Thresh Value for Hough**
Sobel FPGA	50 MHz	Hough C	2.1 fps	6.8 fps	130	80
Steerable FPGA	30 MHz	Hough C	1.3 fps	5.6 fps	130	80
Sobel A FPGA	50 MHz	Hough Day	3.8 fps	8.2 fps	140	100
Steerable FPGA	30 MHz	Hough Day	1.3 fps	5.6 fps	140	80
Steerable FPGA	50 MHz	Hough Night	3.3	4.4	140	60
Steerable C	50 MHz	Zeisl C	2.1 fps	2.1 fps	X	X

table 4.1: An empirical comparison between the speed of the lane detection algorithms in different light conditions

Figure 4.7 depicts the utilized logic per Hough Transform Module. In the case of a real time implementation where two cameras are running simultaneously and lane detection must be done prior to writing the frame to the SRAM, two Hough modules are required per camera. The limiting factor here is the number of utilized multipliers that can be manually implemented to fit Spartan 3E or preferably be ported to a larger FPGA where embedded fast multipliers will speed up the design.

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	253	9,312	2%
Number of 4 input LUTs	578	9,312	6%
Number of occupied Slices	378	4,656	8%
Number of Slices containing only related logic	378	378	100%
Number of Slices containing unrelated logic	0	378	0%
Total Number of 4 input LUTs	711	9,312	7%
Number used as logic	578		
Number used as a route-thru	133		
Number of bonded IOBs	45	158	28%
IOB Latches	18		
Number of BUFGMUXs	1	24	4%
Number of MULT18X18SIOs	7	20	35%
Average Fanout of Non-Clock Nets	2.92		

Figure 4.7: summary of the utilized resources on the FPGA

Chapter 5

Tracking and Kalman Filter

There are strong reasons behind the need to apply a proper tracking module to any image processing system. First of all most of the image processing operations are time demanding which can cause uncertainty due to the miss identification. Second of all it reduces the computational cost by reducing the search area and hence the corresponding pixel operations. Finally it heavily cancels out the noise by discarding other parts of the image therefore the accumulative effect of the noise is reduced [41].

5.1 Literature Survey

According to the conducted literature survey the Kalman filter is broadly applied to the lane tracking problem. All of the considered papers in this field have implemented the Kalman filter in software [43],[17]. There are a few papers that address the hardware implementation of Kalman filter but each of which is considering the implementation of a specialized Kalman filter [44],[45]. Unfortunately all the considered hardware implementations are using a high level implementation by applying the Xilinx's code generator to the Simulink, hence lack a detailed realization. Therefore the whole design for this module is devised and implemented from scratch. Another fact is that almost all the related papers keep silent about the initialization phase and there are only two papers addressing this issue [42],[43].

5.2 Kalman Filter Theory and Initialization

Kalman filter provides "a recursive solution to the linear optimal filtering problem"[42]. There are many advantageous associated with the Kalman filter. It is applicable to both stationary and nonstationary systems on the contrary to the Wiener Filter which only fits stationary systems,

besides the fact that each updated state is computable from the previous estimate and the new measurement so that applying this approach dramatically eliminates the need for data storage as required in some other approaches[42].

A linear, discrete time dynamical system can be described by the *process equation*

$$x_{k+1} = F_{k+1,k}x_k + w_k \tag{1}$$

$$E[w_n w_k^T] = \begin{cases} Q_k \; for \; n = k \\ 0 \quad for \; n \neq k \end{cases} \tag{2}$$

Where x indicates the *state* of the process and $F_{k+1,k}$ is the *transition matrix* of the system that transfers state x_k to x_{k+1}. w_k represents the *process noise* that is assumed to be additive, white ,Gaussian and zero mean[41],[42],[43]. The *measurement equation* can be described by

$$y_k = H_k x_k + v_k \tag{3}$$

$$E[v_n v_k^T] = \begin{cases} R_k \; for \; n = k \\ 0 \quad for \; n \neq k \end{cases} \tag{4}$$

where y_k represents the *observable* at time k and H_k is the *measurement matrix*. v_k represents the *measurement noise* and holds similar specifications as w_k where the covariance matrix of v_k is described by (4). It is assumed that w_k and v_k are uncorrelated[41]. Goal of the Kalman filtering is to gain the unknown posteriori state x_k from the already known priori state $\acute{x_k^-}$ and the new measurement. Derivation of the Kalman filter equations is straight forward and can be found in[41],[42]. Equations (5) to (6) summarize the Kalman filter's behavior.

$$\acute{x_0} = E[x_0] \tag{5}$$

$$P_0 = E[(x_0 - E[x_0])(x_0 - E[x_0]^T)] \tag{6}$$

$$\acute{x_k^-} = F_{k,k-1}\, \acute{x_{k-1}} \tag{7}$$

$$P_k^- = F_{k,k-1} P_{k-1} F_{k,k-1}^T + Q_k \tag{8}$$

$$G_k = \frac{P_k^- H_K^T}{H_K P_k^- H_K^T + R_K} \tag{9}$$

$$\hat{x_K} = x_k^- + G_k(y_k - H_k \hat{x_k^-}) \tag{10}$$

$$P_k = (I - G_k H_k) P_k^- \tag{11}$$

Equation (5) determines the initial state while the initial covariance matrix is estimated by equation (6). Equations (7) and (8) estimate the priori state and its corresponding error covariance matrix. Gain of the filter is represented by G_k while P indicates the *error covariance matrix*. If the displacement of the object under consideration can be modeled as a uniform linear movement, then the state vector, process equation and the measurement equation can be described by equations (11),(12).

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \Delta x_{k+1} \\ \Delta y_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_k \\ y_k \\ \Delta x_k \\ \Delta y_k \end{bmatrix} + w_k \tag{11}$$

$$\begin{bmatrix} xm_k \\ ym_k \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_k \\ y_k \\ \Delta x_k \\ \Delta y_k \end{bmatrix} \tag{12}$$

Otherwise a more sophisticated model i.e. *Extended Kalman Filter* [41] is required. Initialization is the most significant step in order to stabilizing the filter without which the filter response will fluctuate dramatically. The covariance matrix of the *measurement error* is indicated by

$$R_k = \begin{pmatrix} \delta_{e,x}^2 & 0 \\ 0 & \delta_{e,y}^2 \end{pmatrix} \tag{13}$$

where σ denotes the standard deviation and can be estimated by measuring the probability of displacement between the actual position of the object and the measured position of the object. It is assumed that the random variables x and y are uncorrelated indicating that the error in measuring x has no effect on y. For instance if a system requires a measurement accuracy below one pixel, the standard deviation for each coordinate is determined by $\frac{1}{3}\left(3((-1)^2 + (0)^2 + (1)^2)\right) = 2/3$,[42]. Therefore the error measurement covariance matrix of the described system is presented by

$$R = \begin{pmatrix} 2/3 & 0 \\ 0 & 2/3 \end{pmatrix} \tag{14}$$

Covariance Matrix of the Estimation Error is denoted by P_0^- and is described by

$$P_0^- = \begin{pmatrix} \delta_{s,s}^2 & \delta_{s,v}^2 \\ \delta_{s,v}^2 & \delta_{v,v}^2 \end{pmatrix} \tag{15}$$

where s represents the position vector and v indicates the velocity vector. The summarized values in table 5.1 are obtained according to the research conducted by [42] and properly describe the initial values of the elements of P_0^- i.e. the initial values of the covariance matrix of the estimation error which are corresponding to a translational motion in two dimension with time-varying acceleration as the additive white noise

$$P_0^- = 1/16 \begin{pmatrix} s^2 & 0 \\ 0 & v^2 \end{pmatrix} \tag{16}$$

S	25	pixel	at 5 frame /sec
S	6	pixel	at 10 frame /sec
S	3	pixel	at 15 frame /sec
V	49	pixel/frame	at 5 frame /sec
V	12	pixel/frame	at 10 frame /sec
V	5	pixel/frame	at 15 frame /sec

Table 5.1. Initial values for the covariance matrix of estimation error corresponding to two dimensional motion

In the current design a safe value of $10\ frame/sec$ is chosen since it generated a more robust estimation among the three in the simulation phase on MATLAB. Finally it comes to initiating the covariance matrix of the white noise (acceleration). A robust model is generated in [42] to describe any translational system with a constant velocity and random acceleration as

$$\mathrm{Q_k} = a^2 \Delta \mathrm{t}/6 \begin{pmatrix} 2I(\Delta t)^2 & 3I\Delta t \\ 3I\Delta t & 6I \end{pmatrix},\ a \approx 1225\ pix/s^2 \tag{17}$$

Even though all the values suggested by[42] were tested in simulation, applying the mentioned model caused dramatic overshoots in the simulation phase. Therefore the following simpler model is adopted from [43] which proved to generate a more stable behavior as is depicted in fig2.

$$Q = \begin{pmatrix} (x(0) * .2)^2 & 0 & 0 & 0 \\ 0 & (x(0) * .2)^2 & 0 & 0 \\ 0 & 0 & (x(0) * .2)^2 & 0 \\ 0 & 0 & 0 & (x(0) * .2)^2 \end{pmatrix} \tag{18}$$

5.3 Floating Point Implementation

In order to verify the accuracy of the initialized model the filter is first simulated in MATLAB. Since each detected lane by the Hough transform is expressed in the polar coordinates and hence the state vector under observation, it was firstly opted to transform the Kalman filter's equations in the polar coordinates. However it became clear that not only the complexity and the required hardware resources present on the FPGA would increase dramatically due to the introduction of trigonometric functions but also the required truncation for the trigonometric functions for a Fixed Point implementation would contribute to more inaccuracy to the design. Although there are different approaches in the literature towards choosing proper coordinates associated with a lane in a tracking system, a new solution is proposed in this thesis to be measured for efficiency. It was decided to track the intersection of the orthogonal to the lane from the origin as the state vector of the dynamic system under tracking. The coordinates of the intersection are imaged to the Cartesian plane prior to feeding to the Kalman filter by equations (19), (20). Equations (21), (22) determine the inverse transform from Cartesian to Polar as must be applied to the results of the tracked point after tracking to represent the current position of the lane.

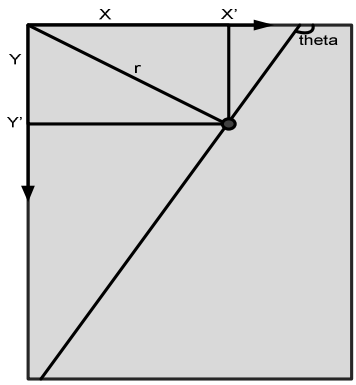


Fig 5.1 image of the tracked point on to the Cartesian coordinate

$$x' = \rho\cos(\theta - 90) = \rho\sin(\theta) \tag{19}$$

$$y' = -\rho\cos(\theta) \tag{20}$$

$$\rho = (x^2 + y^2)^{1/2} \quad (21)$$

$$\theta = \pi - \tan^{-1}(x/y) \quad (22)$$

In Equation (20) the negative sign is added to enforce the positive coordinate to keep the later hardware implementation as simple as possible. This sign modification is not required when the right window is being processed. As mentioned earlier initialization plays a significant role in stability of the filter. According to the several observations on the driving sequences, the left and the right lanes in different frames are usually located in a specific vicinity of the middle of the image hence the state vectors are initialized as

Fig 5.2 initialization for the kalman filter in the left and right windows

$$\rho_L^\circ = 70\ ,\ \theta_L^\circ = 150^\circ \approx x_L^\circ = 35\ , y_L^\circ = 60, \quad (23)$$

$$\rho_R^\circ = 5\ ,\ \theta_R^\circ = 42^\circ \approx x_R^\circ = 3.71\ , y_R^\circ = 3.71, \quad (24)$$

The equations (5)..(11) are not handling a proper representation for the hardware implementation. Therefore the mentioned equations were decided to be spread into a linear representation as is represented in ***Appendix A.*** In simulating the Floating point design it was chosen to apply the achieved spread form of the equations so that a measure of the efficiency of the approach would be achieved. In consistence to the general concept of the Kalman filter, module is staying in the initial state as long as no object i.e. lane, is detected. Once a lane is detected the Kalman filter firstly enters the project-ahead phase to estimates the values of vectors x° i.e. state vector and P° i.e. covariance matrix of estimation error. In the next phase the estimated value of P° contributes to calculation of the filter gain K. Finally in the last phase all the four values of x°, P°, the current measured vector and the calculated gain i.e. K are feed to

the correction equations to form the requirements for the next estimation. At the end the Kalman filter awaits until the next measurement is achieved and this loop goes on as long as a reset signal is designated.

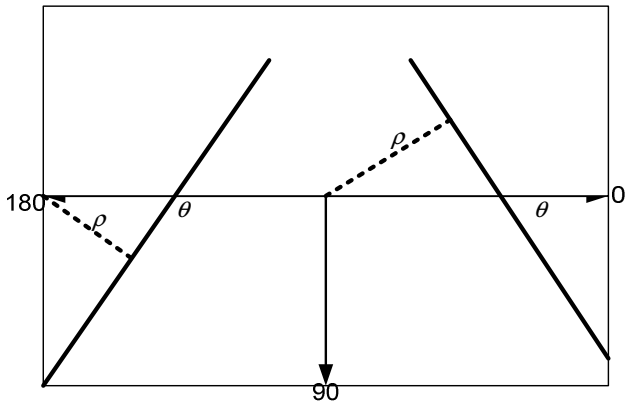


Fig 5.3 proposed structure of tracking the intersection of the prependicular

5.3.2 Analysis of the Results of Floating point implementation

Figures 5.4, 5.5 clearly emphasize the efficiency and importance of the applied Kalman filter. Firstly it is apparent that there are numerous overshoots present in the amplitude of the measured state vector i.e. x and y. This issue plays a significant role in the stability of the system if it is eventually applied to a real driving system. By considering the results of the output of the Kalman filter it is apparent that all the overshoots are suppressed and are not present any more. By considering frames 14 till 19 of figure 5.6 the second achievement of the implemented Kalman filter becomes clear. It is apparent that for various reasons like improper threshoding of the image and most importantly the lack of edge pixels due to the size of the road lane, no lane is recognizable by the Hough transform within a certain range of frames. This issue is observable in figure 5.6. where during frames 1..8 and 14..20 no lane in the left boundary is detectable. It can be seen that the first lane is detected at frame 8 and then after the Kalman filter is correcting the wrong measurement in frame 13 in which the measurement is obviously wrong. The most significant advantage of applying the Kalman filter is observable during frames 14..20 where no lane is detected but the kalman filter still can predict the position of the lane correctly.

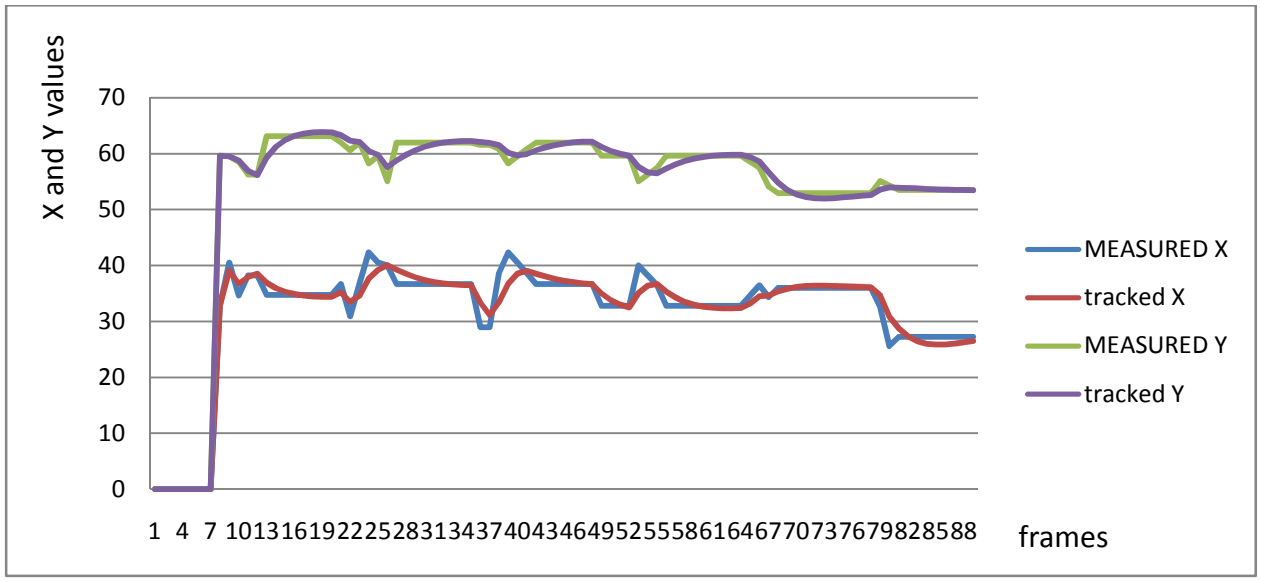


Fig 5.4 the efficiency of the implemented Floating point Kalman Filter to track the measured values in terms of the image of the observable

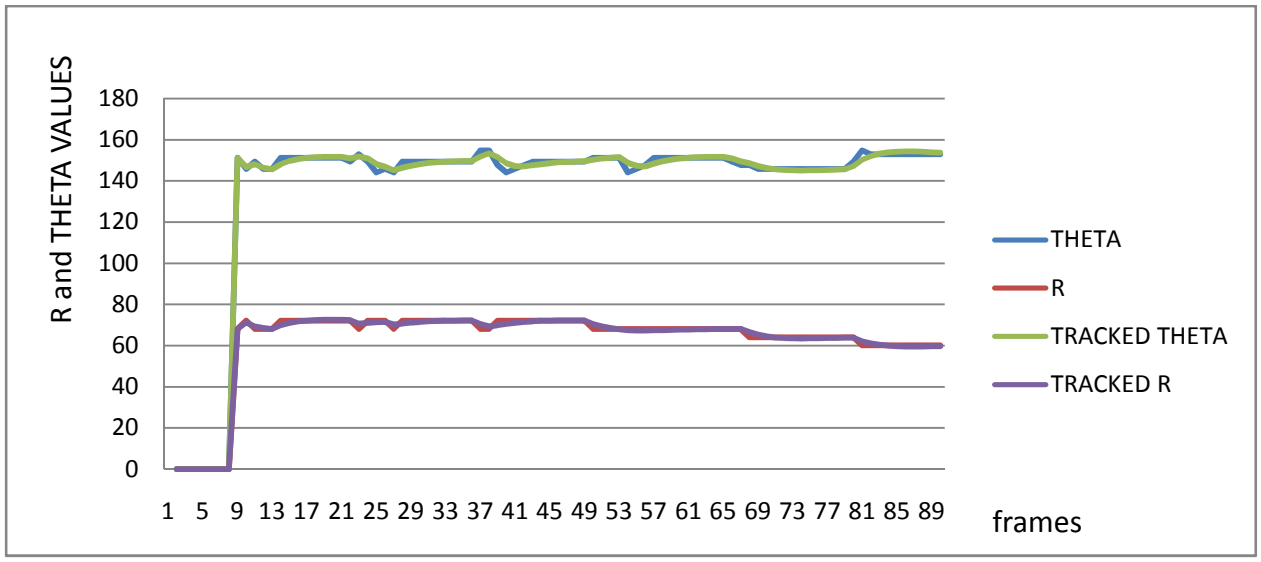


Fig 5.5 the efficiency of the implemented Floating point Kalman Filter to track the measured values in terms of the position of the lane

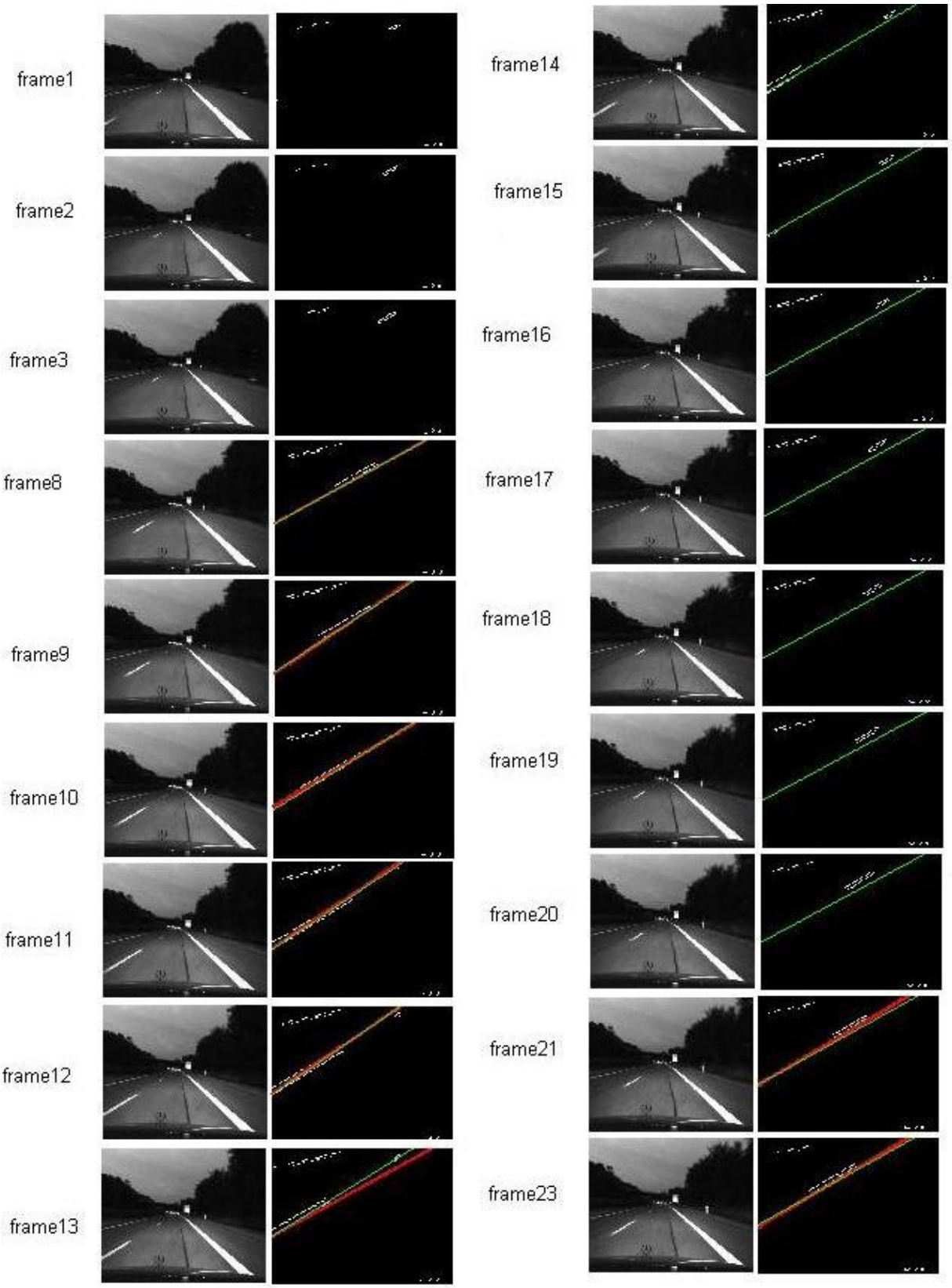


Fig 5.6: lane tracking in the left window, red lanes represent detected lane while the green lane represents the tracked lane

5.4 Fixed Point Implementation and Behavioral Modeling

As was mentioned earlier Kalman filter consists of three phases namely prediction, gain calculation and correction. Hence the hardware design is divided into three blocks to perform each task. The following structure in figure 5.7 is implemented in VHDL to meet the requirements of the Kalman filter.

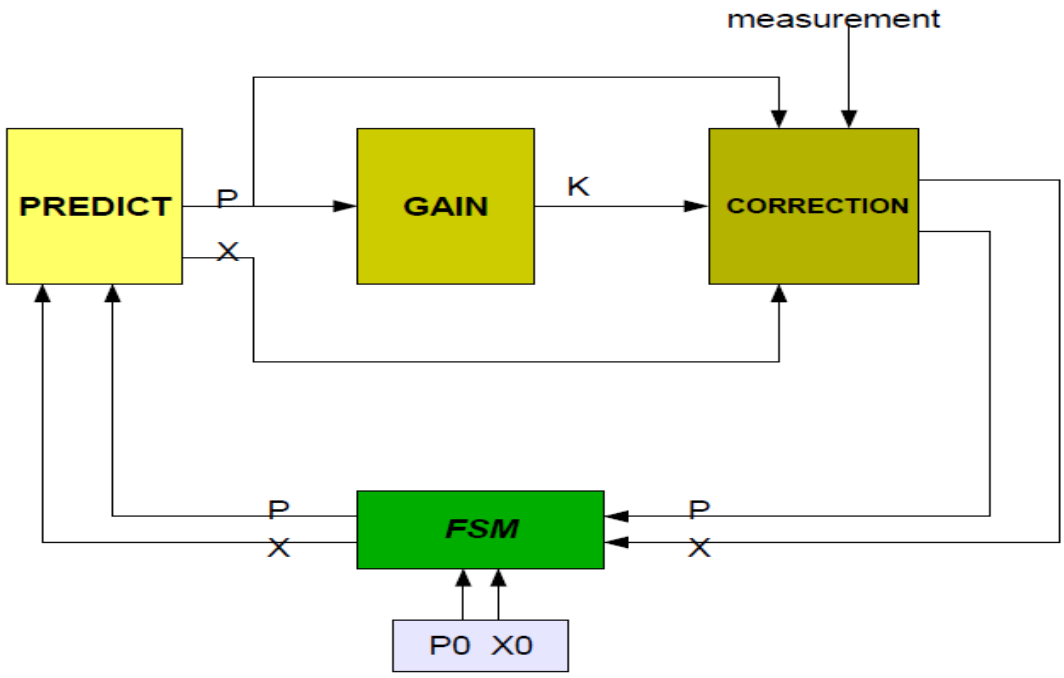


Fig 5.7 consisting modules of the Kalman filter

According to the various experiments conducted on the Kalman filter in the simulation phase it became clear that it is quite sensitive to error propagation. On the other hand a fixed point implementation will always enforce part of the data to be lost. It was observed that at least four digits in the fraction part of a decimal radix representation are required to handle a stable tracking system. Otherwise dramatic overshoots were introduced to the response of the Kalman filter. A precision of four digits in a decimal radix required 10 bits in a binary implementation. Meanwhile 8 bits are reserved for the integer part and 1 bit is required to represent the sign. Therefore a total number of 19 bits are required to represent the signals that interface the three modules of the Kalman filter. The following table is cited from [46] and summarizes only the fixed point sizing rules that are applied to the current design. In the adopted notation a fixed

point number is represented by $[a, b]$ where there are ($a + b + 1$) bits present in the number among which a bits represent the integer part, b bits represent the fraction part and 1 bit is reserved for representing the sign.

operation	*result range*
A + B , A – B	Max(A′left, B′left) + 1 **downto** Min(A′right, B′right)
A ∗ B	A′left + B′left + 1 **downto** A′right + B′right
Signed /, divide	A′left – B′right + 1 **downto** A′right – B′left
Signed (– A)	A'left + 1 **downto** A'right

Table 5.2 Fixed Point sizing rules that are applied to the current design

Once all the modules are tested individually it is required to put them together. Since the Kalman filter is implying a feedback to the design, it was required to meet the timing requirements of the feedback loop. Two solutions were investigated to address this issue. The more sophisticated implementation required the exact amount of delays for all the logic gates through the forward critical path be calculated, so that a feedback path with a proper delay could be designed to meet the timing constraints of the forward path. The mentioned approach would better fit an ASIC rather than a FPGA and requires special tools. However a second solution was devised and implemented by applying a finite state machine. The implemented FSM simply passes the signals through and between modules and isolates them when it is required so that all the timing constraints are met. Figure 5.8 depicts the logic enforced by the finite state machine. There are additional signals implemented to latch and hold the input and output to each module and these signals are assigned by the FSM. Otherwise all the critical paths and timing constraints must be considered into design.

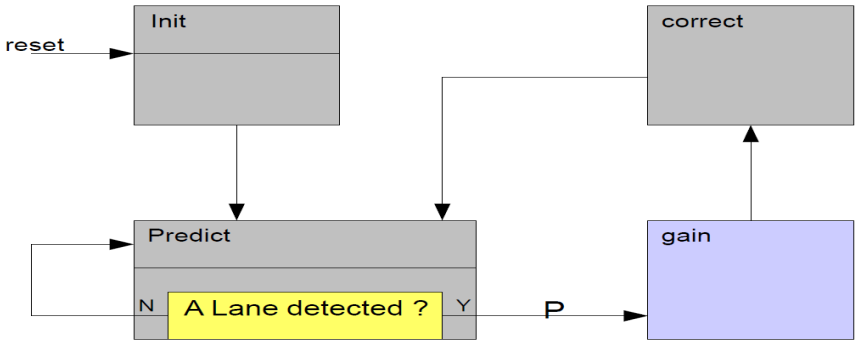


Fig 5.8 Kalman Filter's FSM

The implemented logic will enforce the FSM to stay in the Predict state as long as no object is detected. Only for the first iteration the Predict module is fed by the initial values of state vector and the covariance matrix of estimation error i.e. x_0° , P_0°, while within the next iterations the corrected outputs of the Correct module are forming the x° and P°. The implemented logic in will generate an accurate estimate of the current expected position of the object. By comparing the values of the covariance matrix of error obtained from fixed point and floating point, it was verified that this module is precise up to three significant decimal figures in the fraction part. On the second clock after the pulse indicating the object detection the FSM will feed the outputs of the Predict module to the Gain module. In fact this module is the main source of the error propagation since there is a fixed point division required to be implemented within this module. Therefore even a small amount of truncation error is propagated to the Kalman Gain. However the most significant feature of the Kalman filter is the noise cancelation itself and it cancels the mentioned noise in the next iteration so that the system follows the object as precise as a floating point implementation.

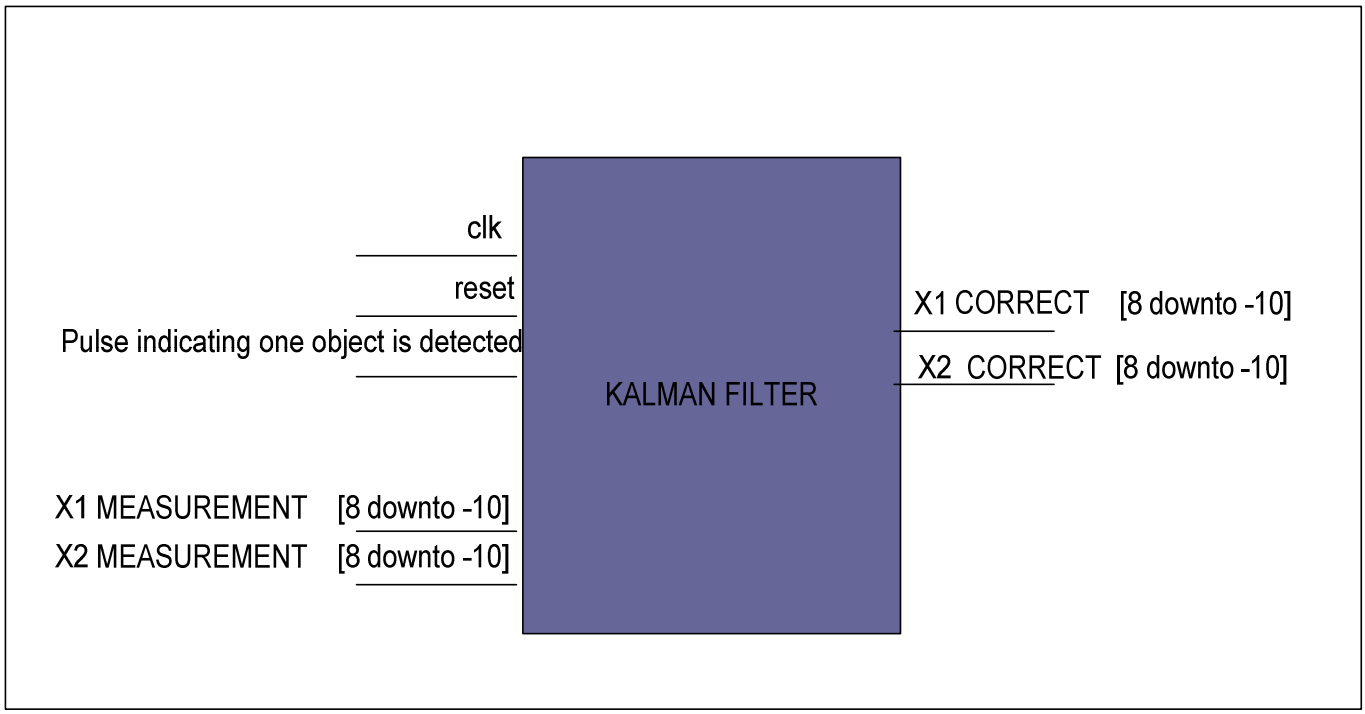


Fig 5.9 Top view of the implemented Kalman Filter

After the third clock FSM enters the Correct state in which both x and P vectors of the filter are altered. In the next clock the FSM enters the predict state and remains in that state until a new pulse indicates a new object is detected. Obviously filter only enters the Init state when a reset signal is issued that will enforce the filter to the initial state. As discussed in the previous chapter, in some frames the detected edges associated with a lane are not sufficient for a lane to be detected. In order to solve this issue it was decided to keep the last measurement as the current position of the detected lane and the achieved results approves that this solution is precise enough for conventional frame rates. Despite the accuracy of the implemented approach, it is suggested that for extreme cases where either the velocity of the observer or the frame rate is very high there might be overshoots in the response of the filter that must be handled.

5.4.2 Analysis of the Results of the Fixed point implementation

The module in a higher level of abstraction only needs to know the current postion of the object and the inerr state of the filter is not required to be public. Therefore the inputs to the filter only include the ***current measurement***, a pulse indicating the ***next object is detected*** besides ***CLK*** and ***RESET***. For the same reason the ***state vector*** $\boldsymbol{x}$ is the only output signal. Figure 7 depicts the input and output signals to the Kalman filter and their corresponding values when simulated on Modelsim. It is important to note that the estimated values have converged to the actual position of the lane and that is a representative of the efficiency of the filter.

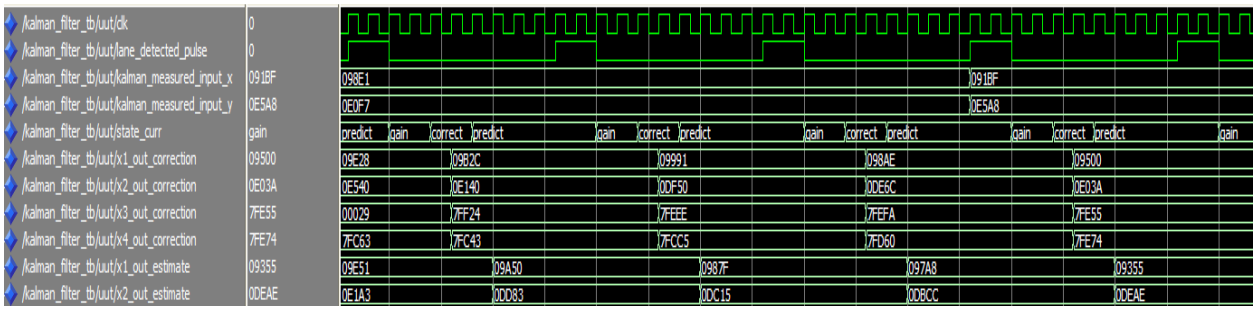


Fig 5.13 inputs and outputs to the Kalman Filter during the four phases of object detection

In order to verify the design a series off the empirical values representing the coordinates of the lane detected by the Hough module are fed to the design and the outputs are compared against each other. Table 5.3 represents three sets of data representing the measured coordinates of the detected lane and the corresponding tracked points calculated by fixed point and floating point implementations. According to the results there is a maximum of 0.4 difference between two implementations in the worst case that will not cause to an error more than one pixel.

Tracked in Floating Point,MATLAB		
frame	X	Y
1	40.1628	59.5751
2	41.0332	59.5034
3	41.0764	59.4999
4	39.5470	57.3548
5	38.7779	56.3419
6	38.3653	55.8447
7	38.1386	55.6112
8	37.2232	56.0497
9	36.6380	56.0497
10	36.2788	56.4051

Tracked in Fixed Point,MODELSIM		
frame	X	Y
1	40.0527	59.5752
2	40.9971	59.4961
3	41.0693	59.4902
4	39.5391	57.3125
5	38.7930	56.3125
6	38.3916	55.8281
7	38.1699	55.6055
8	37.2500	56.0566
9	36.6631	56.4180
10	36.3018	56.7100

DIFFERENCE		
frame	X	Y
1	0.1101	0.0001
2	0.0361	0.0073
3	0.0071	0.0097
4	0.0079	0.0423
5	0.0151	0.0294
6	0.0263	0.0166
7	0.0313	0.0057
8	0.0268	0.0069
9	0.0251	0.3683
10	0.0230	0.3049

Table 5.3 empirical comparison between the Floating point and Fixed Point implementations of the Kalman FIlter

Although introducing longer word length could enhance the precision, the corresponding implementation will require pipelining. The current implementation satisfies the requirements of lane tracking but for a more precise applications like channel tracking in a telecommunication system both longer word length and pipelining are unavoidable. On the other hand the current design is using numerous number of multipliers that needs to be dealt with in a physical realization. Indeed there are two solutions to handle this issue. The fastest approach recommended by the industry is to applying a larger FPGA which contains enough number of fast multipliers. Porting the design to the platform *Xilinx Virtex II* rather than the current *Spartan 3E* would solve this issue. Second possible solution is to implement and embed all the multipliers individually by logic gates that do not seem to be a good solution on a FPGA with a limited

number of gates. Although the implemented structure satisfies the requirements of lane detection, one solution to make the design more accurate is to apply rounding rather than truncation. Since the implemented design is larger than to fit any small FPGA and that rounding will introduce some extra logic for each multiplier, this solution is postponed to a future time.

Chapter 6

Conclusion

This thesis aimed to design and implement a lane detection and tracking system on the FPGA. It was shown that all the three modules meet the required functionalities. The Steerable Filter module satisfies the requirements of the edge detection problem so that not only the frame is edge detected but also all the redundant lanes in the undesired directions are suppressed. However there is some distortion in the edge detected frame due to the short length of the fraction and the applied truncation policy introduced to the design. Furthermore the need to implement the multipliers contributed to a considerable drop in the operation frequency and it became clear that either using a larger platform or implementing the fast multipliers is unavoidable in a future work.

Although the implemented Hough transform performs precisely with adjusted threshold values, it is required to be calibrated regularly. It was observed that this implementation is heavily depended to the light conditions in an environment. The second disadvantage of the implemented logic is that the speed of frame processing is not fixed and depends to the light conditions as well, that makes the analysis troublesome. However the detected lane shows no difference to the results of a floating point implementation on C.

So far the implemented Kalman filter was the best achievement of this thesis. According to various experiments with noisy data the accuracy of the module was verified. In fact it was observed that this module not only cancels out the dramatic measurement noise in some of the frames but also gives a very good estimation for the following frames so that in some frames that no lane was detected the Kalman module estimated the exact location of the lane precisely. Overall the combination of three modules proved to be able to accurately distinguish, detect and track the desired road lanes.

Chapter 7

Future Work

This thesis did not utilize any automation tools like AccelDSP or SystemGenerator and all the modules are designed manually. However it became clear that it is not a proper practice when dealing with a complex design. There are lots of pitfalls that could be easily treated by using such facilities. The main concern of the current design is the introduced distortion to the Steerable Filter's response besides the ambiguity of determining the required fraction length of the signals in each frequency. Both of this issues can be addressed automatically by applying a range of input samples to this tools. In addition these tools offer a fabulous environment for visual design that not only speeds up the design but also leads to a more robust system.

Implemented Hough transform is the most error prone part of this thesis. As discussed earlier this module is very dependent to the determined threshold value. Here I would like to suggest two other approaches that can solve this issue. The easier approach would be to segment the frame and to apply the transform to each segment manually. Hence the probability of a wrong detection drops. Although this approach improves the design, it will not generate a robust system. The best solution is to design an adaptive filter that determines the threshold values due to the level of light and noise.

However there is no guarantee that this module can operate properly if intended to be used in high frequencies. This issue can be explained by noticing the fact that the embedded constants inside the Kalman filter are obtained for conventional speeds of image processing and for other applications like telecommunication it is required to redefine all the embedded error constants. The other concern is the huge size of this module that with any modification still cannot fit *Spartan 3E*. The easier approach, meanwhile the common practice in the industry and literature, is to simply port the design to a larger FPGA. This solution has the additional advantage of using fast multipliers embedded with recent FPGAs. On the hand there is a more sophisticated approach that demands a multi-rate processing with various frequencies on the board.

Appendix A. Required Equations to Port the Kalman Filter to Hardware

Prediction Phase

$$x_1 = x_1 + x_3$$
$$x_2 = x_2 + x_4$$
$$x_3 = x_3$$
$$x_4 = x_4$$

$$p'_{11} = p_{11} + p_{13} + p_{31} + p_{33} + Q_{11}$$
$$p'_{12} = p_{12} + p_{14} + p_{32} + p_{34} + Q_{12}$$
$$p'_{13} = p_{13} + p_{33} + Q_{13}$$
$$p'_{14} = p_{14} + p_{34} + Q_{14}$$

$$p'_{21} = p_{21} + p_{23} + p_{41} + p_{43} + Q_{21}$$
$$p'_{22} = p_{22} + p_{24} + p_{42} + p_{44} + Q_{22}$$
$$p'_{23} = p_{23} + p_{43} + Q_{23}$$
$$p'_{24} = p_{24} + p_{44} + Q_{24}$$

$$p'_{3,1} = p_{31} + p_{33} + Q_{31}$$
$$p'_{3,2} = p_{32} + p_{34} + Q_{32}$$
$$p'_{3,3} = p_{33} + Q_{33}$$
$$p'_{3,4} = p_{34} + Q_{34}$$

$$p'_{4,1} = p_{41} + p_{43} + Q_{41}$$
$$p'_{4,2} = p_{42} + p_{44} + Q_{42}$$
$$p'_{4,3} = p_{43} + Q_{43}$$
$$p'_{4,4} = p_{44} + Q_{44}$$

Blending Factor Calculations

$$\Delta = (p'_{11} + R_{11}) * (p'_{22} + R_{22}) - (p'_{12} + R_{12}) * (p'_{21} + R_{21})$$

$$k_{11} = \frac{1}{\Delta} (p'_{11} * (p'_{22} + R_{22}) - p'_{12} * (p'_{21} + R_{21}))$$

$$k_{12} = \frac{1}{\Delta} (-p'_{11} * (p'_{12} + R_{12}) + p'_{12} * (p'_{11} + R_{11}))$$

$$k_{21} = \frac{1}{\Delta} (p'_{21} * (p'_{22} + R_{22}) - p'_{22} * (p'_{21} + R_{21}))$$

$$k_{22} = \frac{1}{\Delta} (-p'_{21} * (p'_{12} + R_{12}) + p'_{12} * (p'_{11} + R_{11}))$$

$$k_{31} = \frac{1}{\Delta} (p'_{31} * (p'_{22} + R_{22}) - p'_{32} * (p'_{21} + R_{21}))$$

$$k_{32} = \frac{1}{\Delta} (-p'_{31} * (p'_{12} + R_{12}) + p'_{32} * (p'_{11} + R_{11}))$$

$$k_{41} = \frac{1}{\Delta} (p'_{41} * (p'_{22} + R_{22}) - p'_{42} * (p'_{21} + R_{21}))$$

$$k_{42} = \frac{1}{\Delta} (-p'_{41} * (p'_{12} + R_{12}) + p'_{42} * (p'_{11} + R_{11}))$$

$$x_1' = x_1 + (k_{11} * (x_{measured} - x_1) + k_{12} * (y_{measured} - x_2))$$
$$x_2' = x_2 + (k_{21} * (x_{measured} - x_1) + k_{22} * (y_{measured} - x_2))$$
$$x_3' = x_3 + (k_{31} * (x_{measured} - x_1) + k_{32} * (y_{measured} - x_2))$$
$$x_4' = x_4 + (k_{41} * (x_{measured} - x_1) + k_{42} * (y_{measured} - x_2))$$

Error Estimation Correction

$$p_{11}' = p_{11} - (k_{11} * p_{11}) - (k_{12} * p_{21})$$
$$p_{12}' = p_{12} - (k_{11} * p_{12}) - (k_{12} * p_{22})$$
$$p_{13}' = p_{13} - (k_{11} * p_{13}) - (k_{12} * p_{23})$$
$$p_{14}' = p_{14} - (k_{11} * p_{14}) - (k_{12} * p_{24})$$

$$p_{31}' = p_{31} - (k_{31} * p_{11}) - (k_{32} * p_{21})$$
$$p_{32}' = p_{32} - (k_{31} * p_{12}) - (k_{32} * p_{22})$$
$$p_{33}' = p_{33} - (k_{31} * p_{13}) - (k_{32} * p_{23})$$
$$p_{34}' = p_{34} - (k_{31} * p_{14}) - (k_{32} * p_{24})$$

$$p_{21}' = p_{21} - (k_{21} * p_{11}) - (k_{22} * p_{21})$$
$$p_{22}' = p_{22} - (k_{21} * p_{12}) - (k_{22} * p_{22})$$
$$p_{23}' = p_{23} - (k_{21} * p_{13}) - (k_{22} * p_{23})$$
$$p_{24}' = p_{24} - (k_{21} * p_{14}) - (k_{22} * p_{24})$$

$$p_{41}' = p_{41} - (k_{41} * p_{11}) - (k_{42} * p_{21})$$
$$p_{42}' = p_{42} - (k_{41} * p_{12}) - (k_{42} * p_{22})$$
$$p_{43}' = p_{43} - (k_{41} * p_{13}) - (k_{42} * p_{23})$$
$$p_{44}' = p_{44} - (k_{41} * p_{14}) - (k_{42} * p_{24})$$

Appendix B

Computer Listings

1. VHDL Code for

   Steerable Filter

   Hough Transform

   Kalman Filter

2. Matlab Code for simulating

   Steerable Filter

   Hough Transform

   Kalman Filter

3. C/C++ Code for

   Hough Transform on the EyBot M6

   Steerable Edge detection on the Eyebot

References

[1] Xilinx, “FPGA & CPLD Solutions from Xilinx”, [Online] Available: http://www.xilinx.com, [Accessed: Nov, 2009]

[2] OmniVision ”OmniVision”. [Online] Available: http://www.ovt.com/ [Accessed: Feb, 2010]

[3] Intel “Intel Processor PXA270processor for Embedded Computing”, [Online], Available: http://www.intel.com/design/embeddedpca/applicationsprocessors/302302.htm, [Accessed: Oct, 2009]

[4] Gumstix Inc. “Gumstix” [Online] Available: http://www.gumstix.com/ [Accessed: Nov, 2009]

[5] PANKIEWICZ, W. POWIERTOWSKI, G. ROSZAK ,” Implementation of the Lane Detection Algorithm”, 15th International Conference IEEE, 2008, Poland.

[6] Mahmoodi A, Kraut J, Jahns M, Mahmood S, “Hardware Edge Detection using an Altera Stratix Nios2 Development Kit”, Digital ASIC Design Dec 2005, available online at http://www.ualberta.ca/~mahmoodi/Thesis/Edgedetection.pdf

[7] Vo Ky Chau and Truong Quang Vinh ,“EMBEDDED IMAGE PROCESSING SYSTEM ON FPGA”, Available: http://gralib.hcmuns.edu.vn/gsdl/collect/hnkhbk/index/assoc/HASH6088.dir/doc.pdf [Accessed: Jan, 2010]

[8] Xilinx, “Microblaze: Soft Processor Core”. [Online], Available: http://www.xilinx.com/tools/microblaze.htm [Accessed: Jan,2010]

[9] W. Kayankit W. Suntiamorntut ,“Hardware/Software Co-design for Line, Detection Algorithm on FPGA”, 6th International Conference on Electrical Engineering/Electronics, Computer,

[10] ImpulseC, ”Impulse Accelerated Technologies”, [Online]. Available: http://www.impulseaccelerated.com/ , [Accessed: Mar, 2010]

[11] SystemC,,”Open SystemC Initiave” [Online] Available: http://www.systemc.org/home/ [Accessed: Jan 2010]

[12] Discrete Wavelet Transform FPGA Design using MatLab/Simulink, Uwe Meyer-Baese, A. Vera, A. Meyer-Baese, M. Pattichis, R. Perry,University of New Mexico,

[13] Matlab, “MathWork” [Online] Available: http://www.mathworks.com/. [Accessed: Nov, 2009]

[14] Matlab, “Simulink - Simulation and Model-Based Design” [Online]. Available: http://www.mathworks.com/products/simulink/ [Accessed: Nov, 2009]

[15] Altera.”FPGA CPLD & ASIC” [Online], Available: http://www.altera.com/, [Accessed: Jan 2010]

[16] Mentor Graphics, “ModelSim: “Adavanced Simulation & Debugging” [Online], Available: http://model.com/ [Accessed: Mar, 2010].

[17] Video-Based Lane Estimation and Tracking for Driver Assistance: Survey, System, and Evaluation, Joel C Mc.Call and M. Trived, IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, VOL. 7, NO. 1, MARCH 2006

[18] C. Taylor, J. Košecká, R. Blasi, and J. Malik, “A comparative study of vision-based lateral control strategies for autonomous highway driving,” Int. J. Robot. Res., vol. 18, no. 5, pp. 442–453, May 1999.

[19] E. D. Dickmanns and B. D. Mysliwetz, “Recursive 3-D road and relative ego-state recognition,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 14,no. 2, pp. 199–213, Feb. 1992.

[20] D. Pomerleau, “Neural network vision for robot driving,” in The Handbook of Brain Theory and Neural Networks, M. Arbib, Ed. Cambridge, MA: MIT Press, 1995.

[21] K. Kluge and S. Lakshmanan, “A deformable template approach to lane detection,” in Proc. IEEE Intelligent Vehicles Symp., Detroit, MI, 1995, pp. 54–59.

[22] S. Nedevschi, R. Schmidt, T. Graf, R. Danescu, D. Frentiu, T. Marita, F. Oniga, and C. Pocol, “3D lane detection system based on stereovision,” in Proc. IEEE Intelligent Transportation Systems Conf., Washington, DC, Oct. 2004, pp. 161–166.

[23] Y. Otsuka, S. Muramatsu, H. Takenaga, Y. Kobayashi, and T. Monj, “Multitype lane markers recognition using local edge direction,” in Proc. IEEE Intelligent Vehicles Symp., Versailles, France, Jun. 2002, vol. 2, pp. 604–609.

[24] C. Kreucher and S. Lakshmanan, “LANA: A lane extraction algorithm that uses frequency domain features,” IEEE Trans. Robot. Autom., vol. 15, no. 2, pp. 343–350, Apr. 1999.

[25] D. Pomerleau and T. Jochem, “Rapidly adapting machine vision for automated vehicle steering,” IEEE Expert—Special Issue on Intelligent System and Their Applications, vol. 11, no. 2, pp. 19–27, Apr. 1996.

[26] Q. Li, N. Zheng, and H. Cheng, “Springrobot: A prototype autonomous vehicle and its algorithms for lane detection,” IEEE Trans. Intell. Transp. Syst., vol. 5, no. 4, pp. 300–308, Dec. 2004.

[27] J. B. McDonald, “Detecting and tracking road markings using the Hough transform,” in Proc. Irish Machine Vision and Image Processing Conf., Maynooth, Ireland, 2001, pp. 1–9.

[28] D.-J. Kang and M.-H. Jung, “Road lane segmentation using dynamic programming for active safety vehicles,” Pattern Recognit. Lett., vol. 24, no. 16, pp. 3177–3185, Dec. 2003.

[29] N. Apostoloff and A. Zelinsky, “Robust vision based lane tracking using multiple cues and particle filtering,” in Proc. IEEE Intelligent Vehicles Symp., Columbus, OH, Jun. 2003, pp. 558–563.

[30] B. Southall and C. J. Taylor, “Stochastic road shape estimation,” international proceedings of Computer Vision, Vancouver, BC, Canada, 2001, pp. 205–212.

[31] S. Lee and W. Kwon, “Robust lane keeping from novel sensor fusion,” in Proc. IEEE Int. Conf. Robotics and Automation, Seoul, Korea, 2001,vol. 4, pp. 3704–3709.

[32] The University of Western Australia; Computer Science & Software Engineering,”Edge Detection: Classical Method” [Online], Available: http://undergraduate.csse.uwa.edu.au/units/CITS4240/ [Accessed: Feb 2010]

[33] William T. Freeman and Edward H. Adelson, The Design and Use of Steerable Filters, IEEE RANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. 13, NO. 9, SEPTEMBER 1991

[34] Yanxia Wang and Quiqi Ruan, “ Palm-line Extraction Using Steerable Filters”, ICSP 2006 proceedings, IEEE

[35] Xilinx, © “AccelDSP Synthesis Tool” [Online], Available: http://www.xilinx.com/tools/acceldsp.htm [Accessed: March, 2010]

[36] GZ Yang and DF Gillies, “Hough Transform”, Department of Computing, Imperial College

[37] Binary Images, School of Computer Science & Software Engineering, The University of Western Australia

[38] Bernhard Zeisl, Robot Control and Lane Detection with Mobile Phones, Bachelor's Thesis, INST I TUTE FOR REAL - TIME COMPUTER SYSTEMS, Technical University of Munchen

[39] Yue Wang, Eam Khwang Teoh, Dinggang Shen, "Lane detection and tracking using B-Snake", Image and Vision Computing 22 (2004) 269–280

[40] Yue Wang, Earn Khwang Teo and Dinggang Shen, "A B-Snake Model Using Statistical and Geometric Information -Applications to Medical Images", Seventh lnternational Conference on Control, Automation, Robotics And Vision (ICARCV'O2), Dec 2002, Singapore

[41] Erik Cuevas, Daniel Zaldivar, and Raul Rojas, "Kalman filter for vision tracking", 10th August 2005, Technical Report B 05-12

[42] Markus Kohler, Using the Kalman Filter to track Human Interactive Motion , Modelling and Initialization of the Kalman Filter for Translational Motion, Universitat Dortmund, Informatik VII, D-44221 Dortmund, Germany

[43] You Feng, Wang Rongben and Zhang Ronghui,"Algorithm on Lane Changing and Tracking Control Technology for Intelligent Vehicle", Proceedings of the 2007 IEEE, International Conference on Robotics and Biomimetics, December 15 -18, 2007, Sanya, China

[44] Saman S. Abeysekera and Charayaphan Charoensak, "Efficient Realization of Sigma-Delta (Σ-Δ) Kalman, Lowpass Filter in Hardware Using FPGA", EURASIP Journal on Applied Signal Processing, Volume 2006, Article ID 52736, Pages 1–11,DOI 10.1155/ASP/2006/52736

[45] Ruchi Pasricha1, Sanjay Sharma, "An FPGA-Based Design of Fixed-Point Kalman Filter", DSP Journal, Volume 9, Issue 1, June, 2009

[46] Jim Lewis, David Bishop, Kodak, “Fixed and Floating Point Packages”, SynthWorks VHDL Training