

# **Självständigt arbete på avancerad nivå**

*Independent degree project – second cycle*

*Image Processing*

**Pedestrian Detection on FPGA**

**Supervisor: Benny Thornberg**

**Kamran Qureshi**



**Mittuniversitetet**

MID SWEDEN UNIVERSITY

Campus Härnösand Universitetsbacken 1, SE-871 88. Campus Sundsvall Holmgatan 10, SE-851 70 Sundsvall.

Campus Östersund Kunskapens väg 8, SE-831 25 Östersund.

Phone: +46 (0)771 97 50 00, Fax: +46 (0)771 97 50 01.

**Mid Sweden University**

The Department of Information Technology and Media (ITM)

Author: Kamran Qureshi

E-mail address: [kamiqash@gmail.com](mailto:kamiqash@gmail.com)

Study programme: Master in Electronics Design, 120 credits

Examiner: Professor Mattias O'Nils, [Mattias.Onils@miun.se](mailto:Mattias.Onils@miun.se)

Supervisor: Dr. Benny Thörnberg, [Benny.Thornberg@miun.se](mailto:Benny.Thornberg@miun.se)

Scope: 11068 words

Date: 12/28/2013

# Abstract

Image processing emerges from the curiosity of human vision. To translate, what we see in everyday life and how we differentiate between objects, to robotic vision is a challenging and modern research topic. This thesis focuses on detecting a pedestrian within a standard format of an image. The efficiency of the algorithm is observed after its implementation in FPGA. The algorithm for pedestrian detection was developed using MATLAB as a base. To detect a pedestrian, a histogram of oriented gradient (HOG) of an image was computed. Study indicates that HOG is unique for different objects within an image. The HOG of a series of images was computed to train a binary classifier. A new image was then fed to the classifier in order to test its efficiency. Within the time frame of the thesis, the algorithm was partially translated to a hardware description using VHDL as a base descriptor. The proficiency of the hardware implementation was noted and the result exported to MATLAB for further processing. A hybrid model was created, in which the pre-processing steps were computed in FPGA and a classification performed in MATLAB. The outcome of the thesis shows that HOG is a very efficient and effective way to classify and differentiate different objects within an image. Given its efficiency, this algorithm may even be extended to video.



# Acknowledgement

First and foremost praise to Allah, the beneficent, the merciful, there is no power and strength except with Allah. Secondly, I am sincerely grateful to my supervisor Dr Benny Thornberg, for his kind advice and his great enthusiasm throughout the thesis work. His attitude motivated and encouraged me to freely experiment on my thesis work. I am also thankful to him for his careful reading and constructive comments on my report, as they helped me finalize this thesis.

I am thankful to my parents who prayed for my success, and encouraged me to obtain a higher education. I am thankful to them for morally supporting me in all the capacity that they could.

I would like to thank all my friends, for all the smiles and good times they provided whenever the situation became tough, and kept me motivated to move forward.

To all the people who are involved in the finalizing and proof-reading of this thesis report, kindly accept my appreciation.



## Contents

1	Introduction .....	1
1.1	Background .....	2
1.1	Overall Aim.....	3
1.2	Concrete Verifiable Goals.....	3
2	Theory.....	5
2.1	Hog Descriptor Computation in General: .....	5
3	Related Work .....	7
3.1	Dalal and Triggs work.....	7
3.2	Hardware architecture proposals .....	7
3.3	Other related work.....	7
4	Methodology.....	9
4.1	Creation of Training Dataset .....	9
4.1.1	Creation of training dataset with resized negative images .....	9
4.1.2	Creation of training dataset with cropped negative images .....	9
4.1.3	Creation of large training dataset .....	9
4.2	Generation of HOG for Training SVM.....	9
4.2.1	Histogram of HOG generated.....	10
4.2.2	HOG descriptor generated as a vector of 1368 elements .....	10
4.2.3	HOG reduced to 9 values.....	10
4.3	Testing.....	11
4.4	MATLAB-VHDL Hybrid .....	11
5	Implementation .....	15
5.1	Simulation of Algorithm in MATLAB.....	15
5.2	Hardware Architecture Overview .....	18
5.3	TOP_MODULE:.....	19
5.4	Sliding Window:.....	21
5.4.1	Overview of architecture .....	22
5.4.2	Working of the architecture:.....	22

5.5	Gradient Computation .....	25
5.6	Magnitude & Theta Computation: .....	29
5.7	Histogram per Cell Computation .....	32
5.8	Re-Simulation of MATLAB code .....	34
6	Results and Discussions .....	35
6.1	Results based on MATLAB simulation.....	35
6.1.1	Small Training Dataset and Analysis.....	36
6.1.2	Large Training Dataset and Analysis.....	41
6.1.3	Comparison and Discussion of Result with Dalal and Triggs Research Paper	49
6.2	Results based on VHDL-MATLAB hybrid Model .....	49
6.3	Results using VHDL Model.....	51
7	Conclusion .....	53
7.1	MATLAB Simulation: .....	53
7.2	VHDL Simulation.....	53
8	Future Work.....	55
9	References .....	57



## TABLE OF FIGURES

FIGURE 1: FUNDAMENTAL STEPS OF MACHINE VISION SYSTEM. ....	1
FIGURE 2: MATLAB-VHDL HYBRID MODEL FLOW DIAGRAMS .....	12
FIGURE 3: CELLS FOR A 64x128 IMAGE. ....	16
FIGURE 4: FLOW CHART FOR FPGA IMPLEMENTATION. ....	18
FIGURE 5: TOP MODULE INTERFACE USED IN PROJECT .....	20
FIGURE 6: WORKING OF A SLIDING WINDOW .....	21
FIGURE 7: ARCHITECTURE TO ATTAIN NEIGHBORHOOD PIXEL VALUES. ....	22
FIGURE 8: FILLING OF NINE NEIGHBORHOOD VALUES. ....	23
FIGURE 9: EXTENDED ARCHITECTURE FOR RGB IMAGE .....	25
FIGURE 10: SIMULATION RESULT AFTER CALCULATING GRADIENT .....	28
FIGURE 11: QUIVER PLOTS COMPARISON .....	28
FIGURE 12: QUANTIZATION OF DIVIDED ANGLES. ....	30
FIGURE 13: SIMULATION RESULT AFTER CALCULATING MAGNITUDE AND $\Theta_D$ .....	32
FIGURE 14: HISTOGRAM CREATION MODULE INTERFACE .....	32
FIGURE 15: ARCHITECTURE TO ATTAIN 5x5 PIXEL CELL .....	33
FIGURE 16: MODIFIED MATLAB CODE TESTING WITH FULL VECTOR OF HOG. ....	49
FIGURE 17: MODIFIED MATLAB CODE TESTING WITH HOG REDUCED TO 9 BINNED VALUES. ....	50
FIGURE 18: RESULT AFTER CLASSIFICATION USING HYBRID MODEL .....	51
FIGURE 19: MAGNITUDE AND THETA VALUES AS SIMULATED IN VHDL.....	51
FIGURE 20: HISTOGRAM VALUES AFTER VHDL SIMULATION. ....	52
FIGURE 21: TIMING SUMMARY VHDL MODEL. ....	52



# List of Tables

TABLE 1: SUMMARY OF ACCUMULATION OF RESULTS.....	35
TABLE 2: VARIOUS DATABASES TESTED AFTER TAKING HISTOGRAM OF HOG, USING SMALL DATASET FOR TRAINING. THE TRAINING IS DONE USING RE-SIZED NEGATIVE IMAGES. ....	36
TABLE 3: VARIOUS DATABASES TESTED AFTER TAKING HISTOGRAM OF HOG, USING SMALL DATASET FOR TRAINING. THE TRAINING IS DONE USING CROPPED NEGATIVE IMAGES. ....	37
TABLE 4: VARIOUS DATABASES TESTED AFTER GENERATING FULL VECTOR OF HOG, USING SMALL DATASET FOR TRAINING. THE TRAINING IS DONE USING RE-SIZED NEGATIVE IMAGES. ....	38
TABLE 5: VARIOUS DATABASES TESTED AFTER GENERATING FULL VECTOR OF HOG, USING SMALL DATASET FOR TRAINING. THE TRAINING IS DONE USING CROPPED NEGATIVE IMAGES. ....	39
TABLE 6: VARIOUS DATABASES TESTED AFTER REDUCING THE HOG TO 9 BINNED VALUES, USING SMALL DATASET FOR TRAINING. THE TRAINING IS DONE USING CROPPED NEGATIVE IMAGES. ....	40
TABLE 7: VARIOUS DATABASES TESTED AFTER TAKING HISTOGRAM OF HOG, USING LARGE DATASET FOR TRAINING. THE TRAINING IS DONE USING CROPPED NEGATIVE IMAGES. ....	41
TABLE 8: VARIOUS DATABASES TESTED AFTER GENERATING FULL VECTOR OF HOG, USING LARGE DATASET FOR TRAINING. THE TRAINING IS DONE USING CROPPED NEGATIVE IMAGES. ....	42
TABLE 9: DETECTION WINDOW OF 128x64 SWEEP ON TRAINED IMAGES AFTER GENERATING FULL VECTOR OF HOG, USING LARGE DATASET FOR TRAINING. ....	43
TABLE 10: DETECTION WINDOW OF 128x64 SWEEP ON UNTRAINED IMAGES AFTER GENERATING FULL VECTOR OF HOG, USING LARGE DATASET FOR TRAINING. ....	44
TABLE 11: VARIOUS DATABASES TESTED AFTER REDUCING THE HOG TO 9 BINNED VALUES, USING LARGE DATASET FOR TRAINING. THE TRAINING IS DONE USING CROPPED NEGATIVE IMAGES. ....	45
TABLE 12: DETECTION WINDOW OF 128x64 SWEEP ON TRAINED IMAGES AFTER REDUCING THE HOG TO 9 BINNED VALUES, USING LARGE DATASET FOR TRAINING. ....	46
TABLE 13: DETECTION WINDOW OF 128x64 SWEEP ON UNTRAINED IMAGES AFTER REDUCING THE HOG TO 9 BINNED VALUES, USING LARGE DATASET FOR TRAINING. ....	47
TABLE 14: COMPARISON WITH DALAL AND TRIGGS RESULTS .....	48



# Terminology

HOG: Histogram of Oriented Gradients

False Positive: An image incorrectly termed as Human when it is not.

False Negative: An image incorrectly termed as no Human when it contains human.

SVM: State Vector machine

FPGA: Field Programmable Gate Arrays



# 1 Introduction

Image processing is a highly computational demanding field. Recently there has been a great deal of development within this field of study. The main focus of image processing lies in the efficient extraction of image features and classification. Inspired by the human vision and how a person sees, learns and identifies different objects around them is a fascinating aspect to work on. This has resulted in a translation of image recognition algorithms, developed over the years, to machine vision.

The very basics of machine vision include certain fundamental steps that act as the guideline in relation to correct image recognition. These fundamental steps are shown in figure 1.

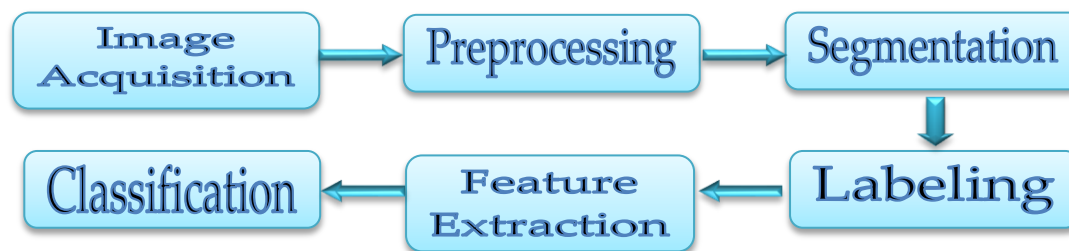


Figure 1: Fundamental Steps of Machine Vision System.

Using these fundamental steps research has been able to identify different objects within an image. One such attempt was to extract a person from an image containing a complex background. However, the segmentation process is the point at which, the contours of the image are usually defined and the picture is simplified. The uniqueness of humans as compared to the background then diminishes. The labeling conducted on the connected components is thus not possible at this stage. Therefore deviation from the basic guideline is necessary.

This thesis adopts a different approach to that of the general model depicted in Figure 1. An image of standard size is taken and gradients of that image are calculated. The gradients which are perpendicular to the tangent lines would have a magnitude (strength) and orientation (angle). This may be considered as image acquisition and pre-processing. In pre-processing, the image is usually normalized for equalizing color ratios. However, this information is taken advantage of in the coming steps and the normalization is left for later steps. A portion of the image pixels would be selected one at a time, known as a cell in this thesis, and the strengths of the same orientations

would be added up. Until this point, there would be an image which is constituted of cells. Overlapping blocks containing cells would be contrast normalized. The values would be saved for each normalized block. The stream of values, generated after all the blocks are contrast normalized, is called a Histogram of Oriented Gradients (HOG). This process is a substitute for segmentation.

An input image containing humans would generate a different HOG than one containing no humans. HOGs are unique for each class of humans and non-humans. Labeling using the uniqueness of an HOG on known images, also known as the training data, would thus be accomplished.

For an unknown image feature extraction i.e. the HOG of an image is calculated. The calculated HOG is fed into a binary classifier so as to classify the picture as one containing either a human or a non-human.

There are various algorithms used for a human detection in an image. The modeling of such at the hardware synthesizable RTL level using a Xilinx ISE development environment is to be attempted in this thesis. Analysis of performance parameters such as FPGA's computational resources, simulated speed and power consumptions are the desired parameters. FPGA being a real-time solution can have many applications such as pedestrian detection at pedestrian crossings, surveillance systems and many more.

## 1.1 Background

There are various publications with regards to Human Detection in images. In this thesis Dalal and Triggs Histogram of Oriented Gradient for Human Detection [1] is used as the basis for this research.

Implementation using an FPGA requires an in-depth study with regards to how the algorithm is modeled in MATLAB. The first goal is to attempt to reproduce the algorithm and the results on the datasets that are provided by the MIT pedestrian database [2].

Following this, a part of this algorithm should be modeled at the RTL-level using VHDL as the preferred language. The research papers studied in this regard are "Deep pipelined one-chip FPGA implementation of a real-time image-based human detection algorithm by Negi.k et al." [3] and "Hardware architecture for HOG feature extraction by Kadota et al." [4]



## 1.1 Overall Aim

The aim of the thesis is to learn how to compute HOG and to use it to detect pedestrians within a given image. Experiments are to be conducted regarding how to reduce the size of a long HOG vector for faster hardware implementation. Comparisons will finally be made with the original HOG model.

## 1.2 Concrete Verifiable Goals

The following series of investigations are planned:

- ⊕ Model the algorithm for pedestrian detection in accordance with Dalal and Triggs [1],
- ⊕ Compare the results from a classification of pedestrian and non-pedestrian images with the results previously published by Dalal and Triggs,
- ⊕ Investigate the effect of reducing the HOG vector length,
- ⊕ Model the computation of an HOG-descriptor as much as possible within the given timeframe at the RTL-level. Compare its function with the Matlab model,
- ⊕ Analyze the hardware performance of the partially implemented VHDL model



## 2 Theory

### 2.1 Hog Descriptor Computation in General:

According to Dalal and Triggs pre-processing is used for normalized color and gamma values. Pre-processing has little effect as the descriptor normalization is performed at a later stage, which essentially achieves the same result. Therefore the first step is to compute the gradient values.

They also found that the HOG descriptor has a reduced performance if smoothing filters are applied as smoothing causes loses in the edge information.

They suggested creating a cell histogram in which each pixel casts weighted votes based on the values found in the gradient computation for an orientation-based histogram channel. These cells can be of different sizes. A cell may contain a variety of combinations of pixels e.g 1x1 pixels per cell, 2x2 pixels per cell, 6x6 pixels per cell etc.

A 1-D histogram channel is evenly spread over 0-180 degrees or 0-360 degrees.

In relation to vote weight, the pixel contribution can be

- ⊕ Gradient magnitude
- ⊕ Some function of magnitude
- ⊕ Square of gradient
- ⊕ Square root of gradient.

Now an image has an obvious factor of changes in illumination and contrast. To account for these Dalal and Triggs[1] suggested that gradient strengths must be locally normalized. This is achieved through block normalization where the blocks overlap i.e each cell contributes more than once to the final descriptor.

There are different schemes that can be used for block normalization. Such as the L2-norm

$$f = \frac{v}{\sqrt{\|v\|_2^2 + e^2}}$$

etc.

Once the HOG has been computed a Support Vector Machine Classifier is trained to react to the presence of the object, such as a human. Support Vector Machine is a binary classifier that looks for an optimal hyper plane as a decision function. It is a binary classifier and since only two classes exist, namely human and non-human SVM appears to be an obvious choice.

## 3 Related Work

Active research is being conducted within the field of human detection. Various methods have been proposed to recognize humans. The basis of human detection lies in object recognition. There is extensive literature available on object recognition, however, this research starts from Dalal and Triggs [1] work.

### 3.1 Dalal and Triggs work

A considerable amount of work has been conducted using their research as the basis for human detection. In their research they were able to show experimentally that grids of Histograms of Oriented Gradient (HOG) descriptors can outperform a previously existing feature set for human detection. They studied how the computation at each stage affected the overall performance and concluded that fine-scale gradients, fine orientation binning, relatively coarse spatial binning, and high-quality local contrast normalization in overlapping descriptor blocks are all important in relation to achieving good results.

### 3.2 Hardware architecture proposals

In this regard, two papers Negi et al [3] and Kadota et al [4] were reviewed. In [4] Kadota et al realized that pedestrian detection on an embedded system is a computationally extensive task. They proposed methods to efficiently reduce the complexity of the computation. They showed that by reducing the complexities of the computation the detector performance is not degraded.

Negi et al used the same complexity reduction methods as in the pre-processing steps. In addition they used the color information. Then using binary patterned HOG features, AdaBoost classifiers generated by offline training, and some approximation arithmetic strategies, their architecture can work on a low-end FPGA without external memory.

### 3.3 Other related work

HOG has been combined with multiple other techniques in an attempt to reduce computational complexity, speed up performance and accuracy. In [11], Wojek et.al. used motion information to improve detection performance. Furthermore, they showed that a combination of multiple and complementary feature types can assist in performance improvement. In [12], Geismann et.al. used a two staged approach, in which the first involved a rapid search mechanism based on simple features to detect a

region of interest and, the second involved the use of a computationally more expensive, but also more accurate set of features on these regions to classify them into pedestrian and non-pedestrian.

In [13], Junfeng et.al. combined multiple techniques to detect pedestrians at night. They used near infra-red camera (NIR) and integrated three modules (region-of-interest (ROI) generation, object classification, and tracking) in a cascade. Taking advantage of NIR, as the pedestrian appears to be brighter than the background, they generated an efficient ROI. Then the classifier is trained using Haar-like and a histogram of oriented gradients (HOG) features.

In [14], Mizuno et.al. extracted HOG features for HDTV resolution. They simplified the HOG algorithm with cell-based scanning and simultaneous Support Vector Machine (SVM) calculation, cell-based pipeline architecture, and parallelized modules. Their results showed that the proposed architecture can generate HOG features and detect objects with 40 MHz for SVGA resolution video (800 ~ 600 pixels) at 72 frames per second (fps). Previously [3],[4],[15],[16],[17],[18] have published their respective results on lower resolutions but [3],[4],[15] and [16] achieved a better frame rate.

## 4 Methodology

### 4.1 Creation of Training Dataset

At the stage of training a classifier, initially, a rather small dataset of positive and negative images was used. Positive images constitute images containing humans and negative images constitute images with no humans within it.

#### 4.1.1 Creation of training dataset with resized negative images

At first the classifier is trained using 24 positive images and 24 negative images. Here as the negative images were larger than the standard format, they were all resized to a standard size. The 24 positive images were taken from the MIT database.

#### 4.1.2 Creation of training dataset with cropped negative images

The second attempt is made using 24 positive images and 24 negative images. Here, the negative images were cropped into various standard sized images. Only 24 of the cropped images were used for training the classifier. The 24 positive images remain the same.

#### 4.1.3 Creation of large training dataset

After evaluating the performance using this small dataset, the training set was increased to 2292 images. This constituted 924 positive images and 1368 negative images. The negative images used were generated using the previously large negative images by cropping them to a standard size. The entire MIT database was taken as the positive images.

### 4.2 Generation of HOG for Training SVM

To train the classifier, a HOG was generated using the following methods

- ⊕ Histogram of HOG generated
- ⊕ HOG descriptor generated as a vector of 1368 elements
- ⊕ HOG reduced to 9 values

#### 4.2.1 Histogram of HOG generated

At the stage during which a long HOG vector is generated, the histogram function was executed with  $nbin = 9$ . This command sorts data into 9 bins. Each bin indicates the number of occurrences of a value in the bin.

For experimentations SVM was trained using different datasets.

1. The dataset mentioned in 4.1.1 is used to train the SVM.
2. The dataset mentioned in 4.1.2 is used to train the SVM.

The HOGs of the datasets were calculated using histogram of HOG.

#### 4.2.2 HOG descriptor generated as a vector of 1368 elements

During the next stage, a HOG descriptor is generated as a vector of 1368 elements.

For experimentation, SVM was trained using different datasets.

1. The dataset mentioned in 4.1.1 is used to train the SVM.
2. The dataset mentioned in 4.1.2 is used to train the SVM.
3. The dataset mentioned in 4.1.3 is used to train the SVM.

HOGs are generated as a vector of 1368 elements.

#### 4.2.3 HOG reduced to 9 values

Another experiment is made in an attempt to reduce the size of the output HOG vector. This time it is at the stage of block normalization. The block normalization is calculating and storing 9 values/block. These values are stored in a large vector with the first 9 values preceding the next 9 values. To reduce the size and to conserve the spatial information, the values calculated per block are superimposed by addition. The result would be an output HOG vector with a total of 9 values.

For experimentation, the SVM was trained using different datasets.

1. The dataset mentioned in 4.1.2 is used to train the SVM.
2. The dataset mentioned in 4.1.3 is used to train the SVM.

The HOGs of the datasets were reduced to 9 values using the method defined.



### 4.3 Testing

The SVM is trained using various methods mentioned in 4.2. After training SVM the results were classified using different datasets mentioned as follows

- ⊕ MIT dataset (924 positive images)
- ⊕ Negative images dataset generated by the author (1368 Images)
- ⊕ Inria pedestrian dataset (1126 positive images)
- ⊕ Trained large images
- ⊕ Untrained large images.

**Trained large images:** These are images which were cropped to generate a negative dataset. They were used in the process of training the SVM.

**Untrained large images:** These are those images which did not contribute to the negative dataset. No part of the image was used in the process of training the SVM.

### 4.4 MATLAB-VHDL Hybrid

To relate the two models, one had to be modified in order to work with the other. The MATLAB model was modified and used for training the dataset. An image is input into the VHDL model and the magnitude and theta are obtained as a result. Processing is then continued in the MATLAB model and classification is made in relation to the trained data.

The block diagram in figure 2 shows the footsteps of the procedure.

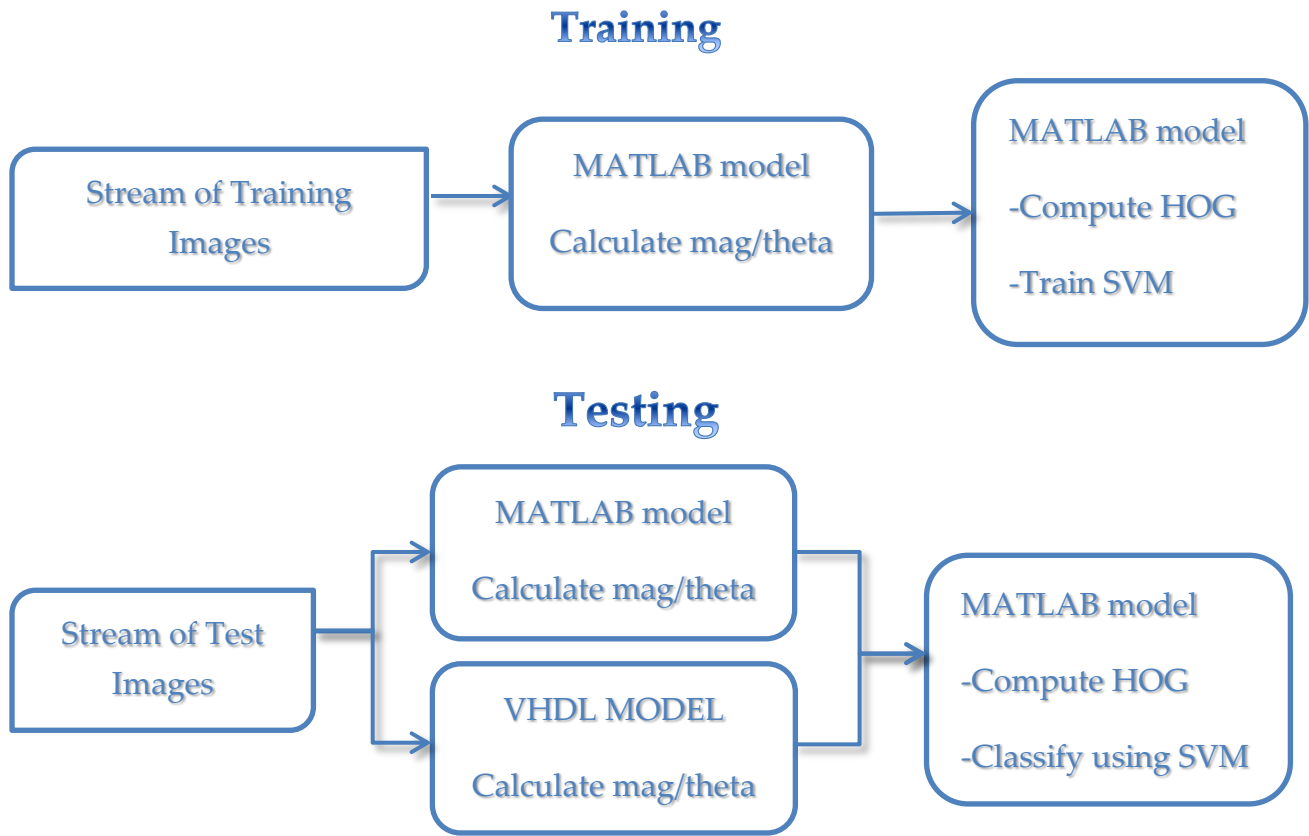


Figure 2: MATLAB-VHDL Hybrid Model flow diagrams

A stream of input images is sent to the MATLAB model. In the first part of the MATLAB model, the magnitudes and thetas of the images are calculated. In the second part, the HOGs are computed and a classifier is trained using the HOGs computed.

Following this another stream of images is sent to the MATLAB model, which calculates the values with regards to the magnitude and theta for each image and sends the results to the second block. At this point, the images are classified as pedestrians/no-pedestrians after their respective HOGs have been computed.

In the third step an input image is provided to the VHDL model. The VHDL model generates both the magnitude and theta of the image. This result is then extracted in MATLAB, and remainder of the HOG of the image is computed and the result is classified.

In addition, the VHDL model is simulated to show the results of both the magnitude and theta. Similarly the values of the 8 bins, to be used to create the histograms per cell, were also simulated.

The whole model is then synthesized to analyze the frequency and speed of the system developed.



## 5 Implementation

### 5.1 Simulation of Algorithm in MATLAB

Dalal and Triggs experimented on various cell size and block size parameters. They found that the best performance was from a 3x3 block of 6x6 cells. Therefore, for this implementation, these values are used to compute the HOG and test them in MATLAB.

The first step was to compute the gradients using filter kernels

$[-1 \ 0 \ 1]$  and  $[-1 \ 0 \ 1]^T$ .

This was achieved through convolving these filters in both the horizontal and vertical directions.

Now, considering that there is a standard cell size, it is always possible to find the image column and row ends by dividing the maximum value of the image row/column by the cell row/column. If, for example, there is an image with 64 columns and where the cell column width is 6 then there is a  $\text{floor}(64/6) = 10$ , and if there are 128 rows and a cell row width of 6, then the  $\text{floor}(128/6) = 21$ .

Bases on this calculation, the total no.of cells that the used image is divided into are known, namely  $10 \times 21 = 210$ . Each cell is 6x6 and thus there are 36pixels per cell. The aim is to generate cell histograms using the gradients as the weighted votes. As suggested by Dalal and Triggs, the magnitude of the gradient itself gives the best result, thus, in this case magnitude of the gradient is used as the weighted votes.

By performing some index calculations it is easy to go six columns forward and six rows downwards, thus covering an entire image. During iterations of the loop each pixel's angle theta of the gradient is calculated. The magnitude is calculated using the horizontal and vertical gradients calculated at an earlier step.

9 bins of  $20^\circ$  are selected for calculations. Dalal and Triggs found that increasing the number of bins improves performance but, only up to 9 bins. They found this for bins spaced over  $0^\circ - 180^\circ$ .

In the simulation setup a bin is selected according to the value of the angle and the magnitude is added in the bin, for example if the angle theta is " $35^\circ$ " it lies in the bin between  $20^\circ - 40^\circ$ . The magnitude of the gradient is simply added for all the angles

contained in the particular bin. Finally, 9 values would be obtained from each cell. These 9 values from each cell are then stored in a structure with each location having 9 values, as illustrated in fig.3. Here Cell 1 hist represents 9 bin values calculated for first the cell.

After all the histograms of the cells have been obtained, a block normalization scheme is applied.

Cell 1 hist	Cell 2 hist	.....	.....	.....	Cell 10 hist
Cell 11 hist	.....	.....	.....	.....	.....
.....	.....	.....	.....	.....	.....
Cell 21 hist	.....	.....	.....	.....	Cell 210 hist

Figure 3: Cells for a 64x128 image.

All the cells have 9 values. In block normalization, 3x3 blocks are used. These blocks would overlap each other so that each cell contributes more than once to the normalized descriptor vector.

In fig.3 the cells in the light green color are shown to be overlapping between two 3x3 descriptor blocks.

In this case by incrementing columns it is possible to move forward until the end of “columns – 2” i.e until cell 8 has been reached and by incrementing the rows, the move is downwards until “row – 2” has been reached i.e row 19.

For each block, the corresponding 9 bin values of cells are added. As a result 9 new values are generated. These values are then normalized using the L2-norm.

L2-norm is slightly modified for simplicity of calculation, as where the small constant ‘e’ is neglected. The equation then becomes

$$f = \frac{V}{\sqrt{\sum_i |V_i|^2}}$$

At the end, these 9 normalized values are stored in a vector. The block normalization process starts again until the image has been completed. The vector length would then be the no. of columns i.e 10 for 64x128 image minus 2. In this case, the preceding two cells are always being calculated, the last cell index required is the total number of columns – 2. Similarly for rows, it is the no. of rows i.e 21 for 64x128 image, minus 2.

The total number of values stored in this vector for 64x128 image is

$$(\text{No. of col} - 2) \times (\text{No. of rows} - 2) \times 9 = 8 \times 19 \times 9 = 1368.$$

These are the HOG of the image.

The last step involved in the training of the images was conducted by means of `svmtrain`, an inbuilt function in MATLAB. This function, for a small dataset, only requires two parameters, namely, the structure that is being trained and a corresponding group vector for two different classes.

One row of the structure corresponds to a class in the group vectors column. One row represents the HOG of one image. If the image in training consisted of a human, then the group vector at that column would have a string 'human' written in there, otherwise it would be a string 'no human'.

For larger datasets, the maximum iterations to converge the algorithm must be increased. In this case it was increased to 200,000. The kernel function used is Linear and the method used to find the separating hyper plane is SMO (Sequential minimal Optimization).

## 5.2 Hardware Architecture Overview

It is always good practice to develop a solid block diagram for the FPGA implementation, if possible. The flow chart for the implementation of this algorithm is given in figure 4

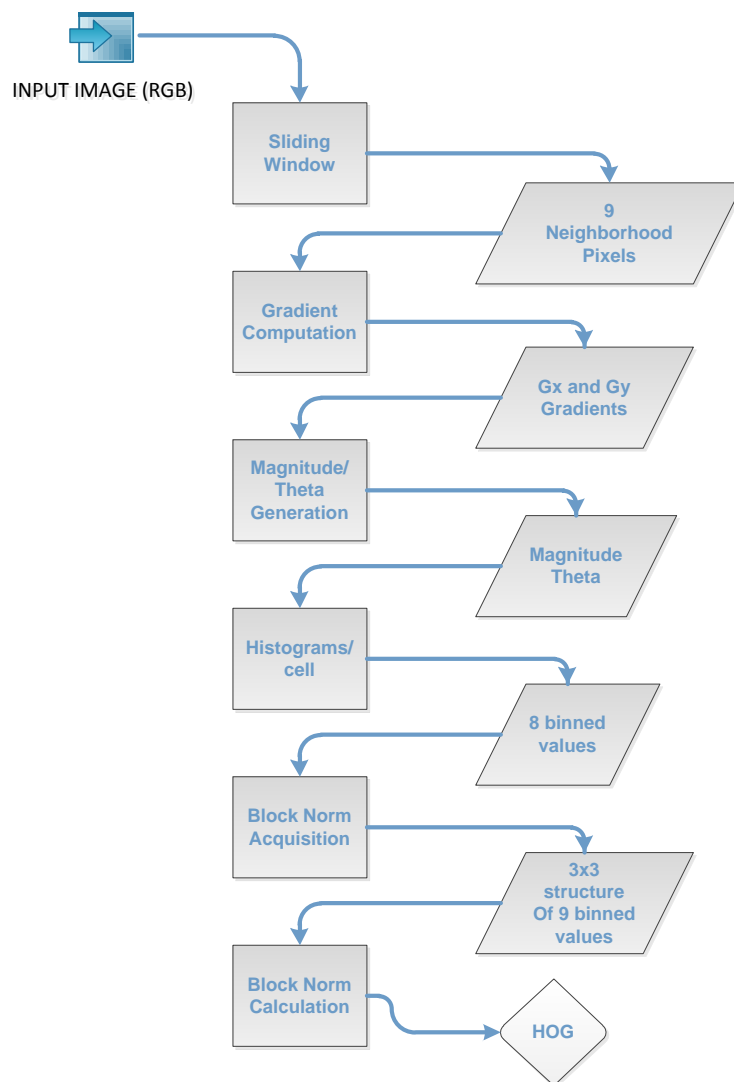


Figure 4: Flow Chart for FPGA implementation.



Inputs are fed into the model pixelwise, through a sliding window on each clock cycle. The sliding window outputs nine pixels at a time, called the nine neighborhood values, using the line buffers. These nine neighborhood values are then used as the input for the gradient computation, where the gradient of the middle pixel i.e P5 is calculated. This would inevitably mean that the borders for any image are not going to be calculated. The border conditions in this development of the algorithm are not considered as they make either very minimal or no difference in the final outcome.

The two gradients,  $G_x$  and  $G_y$ , are forwarded to the next module magnitude/theta generation, where the magnitude and theta using gradients are calculated. The values of the magnitude and theta are sent from this module to the next where these values are saved in array buffers until 25 values for both magnitude and theta are available. These 25 values are used in the histogram/cell to return the 8 bin values used to create one cell.

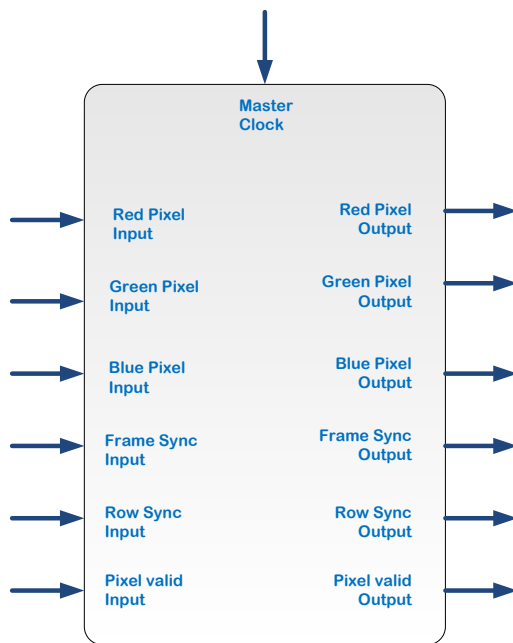
In this thesis, the design has been developed up to the histogram/cell module. The remaining blocks are suggestions for future work and possibly to provide an idea of the end product.

A detailed module working is discussed in the forthcoming topics.

### 5.3 TOP\_MODULE:

Ideally, what is required from the top module is to input a colored image. Each color is represented by 8 bits. Master clock, frame sync and row sync signals are fed from the camera. A pixel valid signal representing when the pixels of interest are starting is also sent from the camera. In return, the RGB data was used and the HOG of the image was calculated and, in the case of a video, the HOG of one frame of video was used. The image, or the frame of video used, should be the same size as that used for training the data.

The Interface of the top module used in this project is shown in figure 5.



**Figure 5: Top Module Interface used in Project**

The design of this top module is because it is a partial implementation of the algorithm. It also depended on the testbench available. The provided test benches were of two types, namely, one for a monochrome image and the other for the RGB image. The first testbench read the pixel values disregarding the color. The output pixel vector was 8 bits long. Thus, it outputted values between 0-255. When simulated, it saved the output as a “.bmp” format image in the folder specified. The output was a black and white image.

The other test bench took an RGB image. Separately, it then read and saved the RGB values and outputted an RGB image. Each color vector is 8 bits long. Therefore, there is finally a colored image. Each pixel has three values conserved inside it. These three 8 bit long vectors per pixel would be used to store the required outputs.

The rest of the ports such as Master Clock, Frame sync, Row sync, pixel valid input and pixel valid output are common in all the top module designs. The beginning of the implementation in VHDL was initially prepared using a monochrome test bench. Later, it was modified for the RGB test bench. Therefore, the workings and procedures of both the monochrome implementation and the RGB implementation are mentioned.

#### 5.4 Sliding Window:

Monochrome development simplified the implementation, which in return assisted in a better understanding of the algorithm and thus designing the more complex code for RGB image.

Sliding window is not just the name of this module. It is also the concept that it works on. Consider a picture as a set of blocks as shown in figure 6

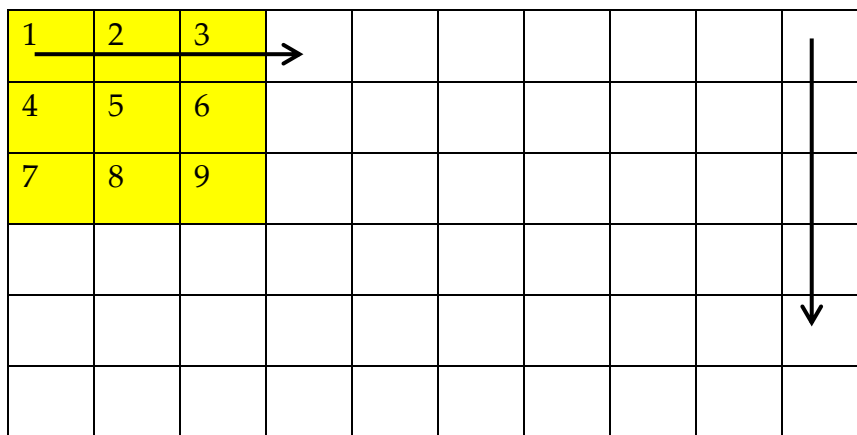


Figure 6: Working of a sliding window

Each block in this figure represents one pixel. A window of 3x3 is created at the beginning of the figure. This window has 9 values which are called neighborhood values. These values are used to calculate the gradients of pixel no 5. After calculating the gradient through this window, the same window is moved one pixel step forward columnwise. At the end of each column, this same window moves one pixel step down row wise. The process follows until the image ends. The window is then reset to its initial location.

In the process, each time the window moves, a new value for pixel no 5 is calculated. Hence the whole process works as if a window is practically slid over the image and the values are thus calculated.

To achieve this task in hardware description requires an architecture that has memory to save the previous values. In the hardware, an image is read one pixel at a time from left to right and top to bottom as shown in the figure above. This means that the pixel value read at the first clock cycle would no longer be available after having been read.

The model of the architecture used to accomplish this task is given in figure 7.

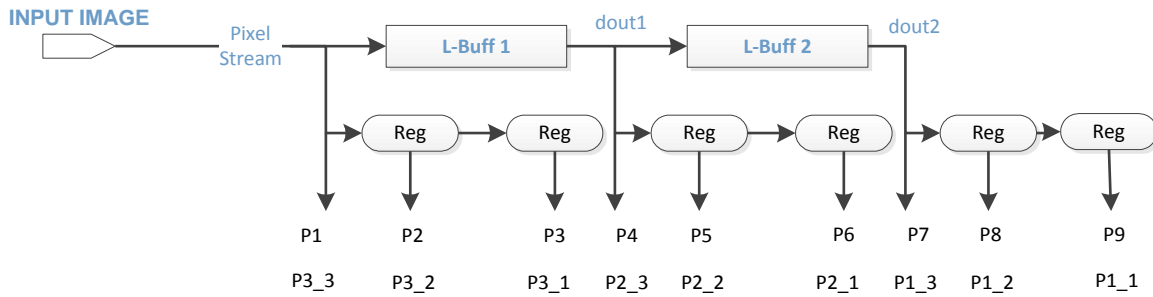


Figure 7: Architecture to attain neighborhood pixel values.

#### 5.4.1 Overview of architecture

In this architecture, L-buff 1 and L-buff 2 are two first in first out (FIFO) buffers implemented in block RAMs. Their size depends on the no.of cols to be read in the image. Each line buffer is capable of storing a whole row in it. The two line buffers are filled, simultaneously, using the pointer to obtain the desired location. The pointer is common for both line buffers.

In figure 7, the caches are represented as a bunch of registers. Three caches were used with a length of 16 bits. Each cache comprises of two 8 bit registers. Finally, the 9 neighborhood values are extracted and sent to ports P1 to P9.

#### 5.4.2 Working of the architecture:

Input pixels are coming in as a stream of pixels. The pointer is presently pointing at the start of both line buffers. Both line buffers and caches are initialized with zeros. The first pixel is stored in the first location of the first line buffer. Consequently, the first value i.e 0 of the first line buffer is stored in the second line buffer's first location. This value is also stored in the higher bits of cache 2. Following a similar pattern cache 3 would end up having the first value of the second line buffer.

The same input pixel's value, at the same clock cycle is also stored in cache 1's higher 8 bits. The architecture works with the concurrent processing of the hardware description where the first pixel is also assigned to the first port P1. P2 is assigned the higher 8 bits of cache 1. This value, as cache 1 was initialized as zeros, would be 0. Cache 1's lower 8 bits are assigned to P3. Then again, in a similar fashion, the first value to line buffer one is termed as dout1 in the figure above is assigned to P4, cache 2's higher 8 bits to P5 and cache 2's lower 8 bits to P6, dout2 the first value of line buffer two assigned to P7, cache 3's higher 8 bits to P8 and, lastly, cache 3's lower 8 bits to P9.

At the next clock cycle, the second pixel of the image would be stored in the second location of the first line buffer, similarly, in the second line buffer, the second value of the first line buffer, which was again 0, would be written, and this would continue until both line buffers are full with the first and second row values of an image.

After reading the first three values of the third line of the image, the 9 neighborhood values would be successfully obtained which can then be used to calculate the gradients of an image.

A summary of how the values are filled in presented in figure 8.

First	Second	Third	Rows of image
P1_1	P2_1	P3_1	First line (row)
0	0	0	
0	0	0	
P1_2	P2_2	P3_2	Second line (row)
P1_1	P2_1	P3_1	First line (row)
0	0	0	
P1_3	P2_3	P3_3	Third line (row)
P1_2	P2_2	P3_2	Second line (row)
P1_1	P2_1	P3_1	First line (row)

Figure 8: Filling of nine neighborhood values.

In this figure P1\_1 denotes Pixel one of first line, P2\_1 is second pixel of first line and so on and so forth till P3\_3 i.e third pixel of third line.

The assignment of caches is as follows

Cache1(15 dt 8)  $\leftrightarrow$  dout1,

Cache1(7 dt 0)  $\leftrightarrow$  Cache1(15 dt 8),

Cache2(15 dt 8)  $\leftrightarrow$  dout2,

Cache2(7 dt 0)  $\leftrightarrow$  Cache2(15 dt 8),

Cache3(15 dt 8)  $\leftrightarrow$  dout3,

Cache3(7 dt 0)  $\leftrightarrow$  Cache3(15 dt 8),

As seen in the above figure, the values of the first row of image are moving downwards. Caches are storing the last two previous values. The ports would attain the values given below:

P1  $\leftrightarrow$  dout1,

P4  $\leftrightarrow$  dout2,

P7  $\leftrightarrow$  dout3,

P2  $\leftrightarrow$  cache1 (15 dt 8),

P5  $\leftrightarrow$  cache2 (15 dt 8),

P8  $\leftrightarrow$  cache3 (15 dt 8),

P3  $\leftrightarrow$  cache1 (7 dt 0),

P6  $\leftrightarrow$  cache2 (7 dt 0),

P9  $\leftrightarrow$  cache3 (7 dt 0),

As the third row of image is filling in the line buffer one, the first line of the image now stored in line buffer two, is being overwritten by values of line buffer one. Line buffer one, at this moment, contains the values of the second row of the image. After the third row values are completely written into line buffer one, the first row of the image would have been used and is thus wiped from the memory. Thus, at any given time, two rows of the image are stored in the memory.

After successfully achieving the results with the line reading architecture using a monochrome image, this technique is extended for an RGB image. Each color is 8 bits long, thus considered as one monochrome pixel. Separate line buffers are used to store the values of each color. As two line buffers had been used in the previous architecture,

in the extended architecture two line buffers are used per color. In fact, separate copy of the architecture per color is used, as shown below. The pointer, however remains common for all the architectures, which in return means that the process works concurrently.

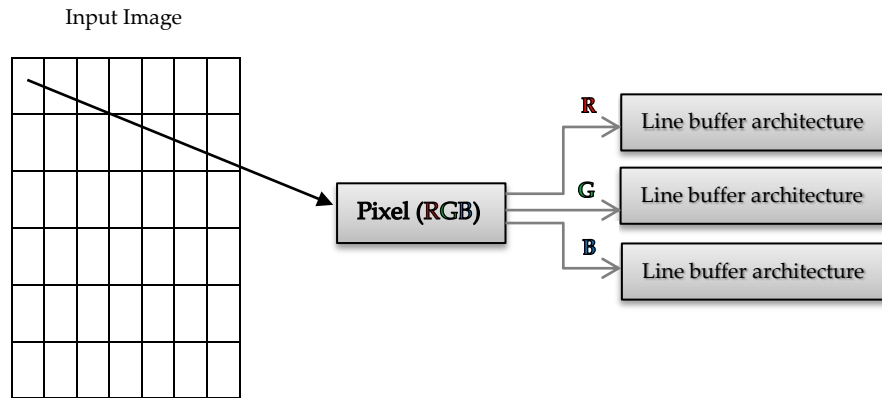


Figure 9: Extended architecture for RGB image

## 5.5 Gradient Computation

In this module the nine neighborhood pixel values are obtained for the monochrome image. To find the gradient of the image, the formula suggested in “Deep pipelined one-chip FPGA implementation of a real-time image-based human detection algorithm by Negi.K et al”[3] is used.

$$\Delta L_x = L(x+1,y) - L(x-1,y)$$

$$\Delta L_y = L(x,y+1) - L(x,y-1)$$

To translate this equation according to this architecture, the end table entries of Figure 8 are revisited.

<b>P1_3</b>	<b>P2_3</b>	<b>P3_3</b>	<b>Third line (row)</b>
<b>P1_2</b>	<b>P2_2</b>	<b>P3_2</b>	<b>Second line (row)</b>
<b>P1_1</b>	<b>P2_1</b>	<b>P3_1</b>	<b>First line (row)</b>

Using this table it is formulated that the middle pixel, which is P5, is P2\_2. This particular pixel is identified because this is the pixel whose gradients in the x and y directions are to be calculated. Therefore, the  $\Delta L_x$  would be  $P2_2 = d_{in5\_x}$  and  $\Delta L_y$  would be again  $P2_2 = d_{in5\_y}$ . The means of separating the two values will be discussed when implementing the RGB image.

According to the table, then

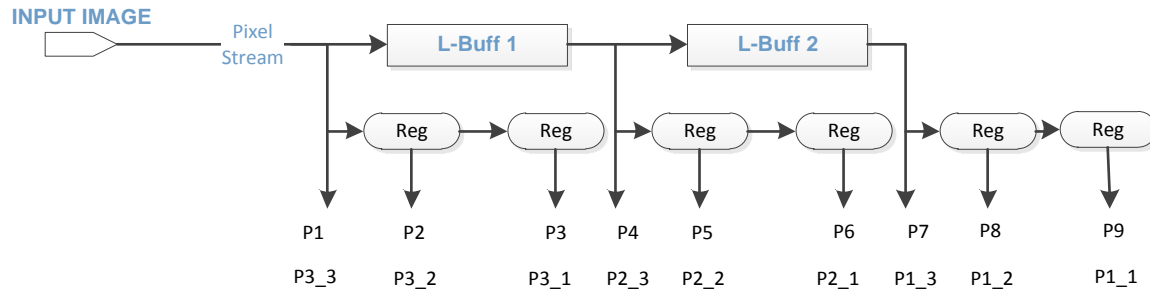
$$L(x+1,y) = P2_3$$

$$L(x-1,y) = P2_1$$

$$L(x,y+1) = P3_2$$

$$L(x,y-1) = P1_2$$

Now referring back to Figure 7 these values were easily translated to pixel values as shown below



The final equation results in

$$d_{in5\_x} = P4 - P6$$

$$d_{in5\_y} = P2 - P8.$$

The nine neighborhood values coming in are 8 bit vector values. To subtract one from another, the 2's complement of the latter is taken. Therefore the 2's complement of P6 and P8 were taken and then simply added together to obtain the two gradient values.



Considering the fact that gradients can be negative, the range of d\_in5\_x and d\_in5\_y becomes  $-128 \leftarrow \rightarrow 127$ .

The magnitude is calculated using an approximation of the formula

$$m(x,y) = \sqrt{\Delta L_x(x,y)^2 + \Delta L_y(x,y)^2}$$

to

$$m(x,y) = |\Delta L_x| + |\Delta L_y|$$

This is a typical approximation of the above equation as it works much faster in hardware.

For this, the highest bits of d\_in5\_x and d\_in5\_y are checked. If they are '1' then simply take the 2's complement else allow the number to stay the same. The whole system of the hardware is based on the 2's complement therefore for

$$d\_in5\_x = -4 = 10000100$$

$$d\_in5\_y = -4 = 10000100$$

After the 2's complement, 01111100 is obtained. Adding both the values together results in 11111000. This number is translated as 8, as the result is also shown after the 2's complement. The result would be determined as being 00001000.

This value was being sent out on data out, which was the 8 bit output port.

Extending the whole concept of computing gradients to color image had its advantages. In this module, there are 9 neighborhood pixels for each color. Each color's gradient is computed using the method previously discussed. At this point there are three x-coordinate gradients and three y-coordinate gradients.

As convolution was applied in the MATLAB model and the strongest gradient was then taken based on comparison statements, comparison statements are similarly applied in order to extract the strongest gradients from the color information of an image. This greatly increases the gradient strength and in finally producing a solid result.

Here, for the purpose of testing and comparing, the red pixel of the output was assigned the x-coordinate gradients. The green pixel of the output was assigned the y-coordinate gradients and the blue pixel was given a constant value.

The resultant image produced after applying the changes is shown below.

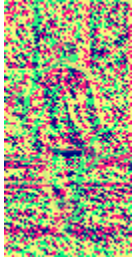


Figure 10: Simulation result after calculating gradient

Figure 10 was exported to MATLAB and decoded in the same pattern, namely, the red pixel produced  $G_x$  gradients, the green pixel produced  $G_y$  gradients and the blue pixels were all constants, hence could be ignored. A quiver plot was then generated to determine the gradient strengths visually using the original pictures gradients found using the MATLAB model when compared with the gradients found using the VHDL model.


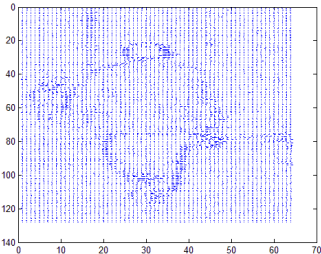
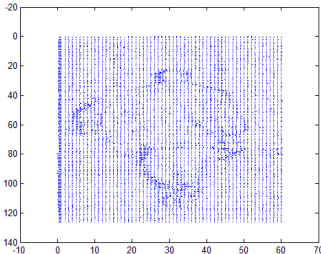
Original Picture	Quiver VHDL model	Quiver MATLAB model
		

Figure 11: Quiver Plots Comparison

After analyzing the two images and finding them to be similar, specifically regarding the outline of a person, whose head and legs are present and, whose contours show strong gradients and thus assist in identifying unique HOGs for humans, the next step of the preprocessing was carried out.

## 5.6 Magnitude & Theta Computation:

A separate module to calculate the magnitude and theta of the gradients was compiled for more clarity in the coding structure. As soon as the two gradients of one pixel were calculated, they are sent to the “Generation magnitude/theta” module.

In this module the strongest value of the two gradients is extracted from the previous module. Using the previously used approximation of the Pythagorean formula, the magnitude was calculated. However, this time the result was first saved in “variable bit vector” of width 8. Then the port “mag” was simply updated by adding the two “variable bit vectors”. These “variable bit vectors” namely `std_Gx_mag` & `std_Gy_mag` were saved specifically to assist in the calculation of the angle theta.

In MATLAB it is very easy to calculate trigonometric functions. To perform the inverse of a trigonometric function takes the same amount of time. However, in the hardware this becomes a matter of estimating the functions rather than simply implementing them, as they take up numerous resources and are very slow to execute. Therefore, a typical means of performing these functions is to develop a look-up table.

The look-up table to calculate theta is extracted from ““Deep pipelined one-chip FPGA implementation of a real-time image-based human detection algorithm by Negi.k et al.”[3]

The development of the look-up table is based on the fact that

$$\text{Equation 1: } \tan\theta = \frac{\Delta L_y(x,y)}{\Delta L_x(x,y)}$$

From equation 1 an expression can be derived that would eliminate the need to divide, in terms of the hardware, since division is also a resource consuming computation.

The previous algorithm developed in MATLAB had 9 bins. Here as suggested in the reference 3, 8 bins are used.

Negi et al used 8 quantization orientations as shown in the figure below.

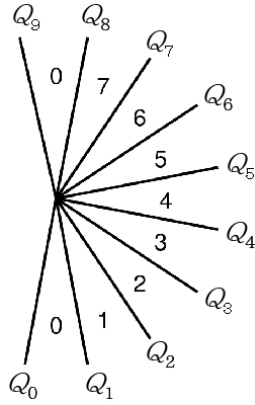


Figure 12: Quantization of divided angles.

This figure is directly taken from their published paper and named the same. This illustrates how they have quantized the orientations. They account for angle between  $-\frac{\pi}{2}$  to  $\frac{\pi}{2}$  for  $\theta$ . For the opposite directions, the same orientation is intended. As an example, it can be seen in the figure above that the range  $Q_8$  to  $Q_9$  is opposite to the range  $Q_0$  to  $Q_1$  and thus marked as the same orientation i.e 0. These values from  $Q_0$  to  $Q_9$  are called the threshold values and can be pre-calculated using the equation suggested below.

$$\tan Q_d = (7-2d) \times \frac{\pi}{16}$$

where  $d$  varies from 0 to 9. If  $\theta$  lies between  $Q_1$  and  $Q_0$  the following equation is obtained.

$$Q_1 \leq \theta < Q_0 .$$

Here, it is possible to generalize the above equation and start to derive an expression for the look-up table. Therefore

$$Q_{d+1} \leq \theta < Q_d$$

Applying trigonometric function

$$\tan Q_{d+1} \leq \tan \theta < \tan Q_d \text{ is obtained}$$

and using equation 1 the following is obtained,

$$\tan Q_{d+1} \leq \frac{\Delta L_y(x,y)}{\Delta L_x(x,y)} < \tan Q_d$$

Cross-multiplying the above equation gives

$$\Delta L_x(x, y) \tan Q_{d+1} \leq \Delta L_y(x, y) < \Delta L_x(x, y) \tan Q_d$$

They employed fixed point arithmetic with a 10 bit fractional part. This was achieved by multiplying each value calculated by 1024. This, finally, provided the lower and upper ranges necessary to calculate  $\theta_d$ .

How they arranged the quadrants for comparison operations is now investigated.

Second Quadrant	First Quadrant
$f_x < 0$ $f_y < 0$	$f_x > 0$ $f_y < 0$
Third Quadrant	Fourth Quadrant
$f_x > 0$ $f_y < 0$	$f_x > 0$ $f_y > 0$

For the first and third quadrants the sign of  $f_x$  and  $f_y$  are different. Similarly for the second and fourth quadrants the signs are the same. As mentioned earlier, the orientations are the same for opposite directions, therefore, two main comparisons remain. One is when they have different signs and the other when  $f_x$  and  $f_y$  have the same signs.

In this VHDL model, the highest bits of "Gx\_mag" and "Gy\_mag" are checked. If they have the same signs the comparisons for  $\theta_d = 0, 1, 2, 3, 4$  are made. Otherwise comparisons for  $\theta_d = 0, 7, 6, 5, 4$  are made.

After finding the magnitude and  $\theta_d$  values, a test image is simulated. The 8 bit RGB vectors are again used to encode the result. A magnitude is assigned to R,  $\theta_d$  is assigned to G, and B remains constant. The resultant picture after simulation is shown in figure 13.

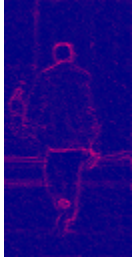


Figure 13: Simulation result after calculating magnitude and  $\theta_d$ .

The new specification of a 5x5 cell and 8 bins meant the MATLAB model had to be modified. This modification is discussed at a later stage in the thesis. Having the magnitude and theta\_d values, it then becomes possible to create histograms per cell by using the weighted votes of magnitude.

## 5.7 Histogram per Cell Computation

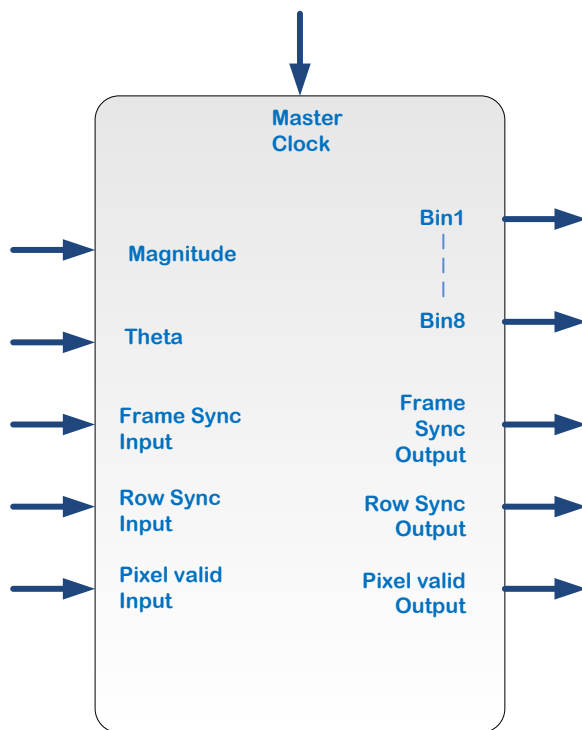


Figure 14: Histogram Creation Module Interface

The whole testing of this thesis is hybrid as a part of the VHDL model and part a MATLAB model is used to compute the final outcome. However, this module was not

included in the hybrid testing, rather it is simulated and synthesized only in the VHDL model. The results of its simulation are shown in the results section.

As 5x5 pixels per cell are now being taken. Each pixel thus has a magnitude and orientation. The magnitudes of the pixels with the same orientation would add up. The development of an algorithm is necessary in order to capture the first 25 values of both the magnitude and theta. The cells cannot be overlapping. A module to add bins is created within this main module. This module, more importantly, is triggered when a bin enable is '1'. To extract the 25 values a concept similar to the sliding window is used. Instead of line buffers, array buffers are used which store two values, magnitude and theta at one location and are first in first out (FIFO) buffers implemented in block RAM.

Columns and rows of the image are counted to accurately enable the addition of the bins. Integer variables flag\_row and flag\_col are defined and initialized to '5'. When the row count reaches the flag\_row and the column count reaches the flag\_col, the first group of values has been successfully extracted. Bin addition is enabled, otherwise always disabled. When the two corresponding conditions are true, flag\_col is also incremented to '5' to obtain the other 25 values. As stated earlier, this process is similar to the sliding window, in which the desire was to extract 9 values, and in which, 2 buffers were used for this purpose. In this case, 5 array buffers are actually used as the previous values are required.

At the end of the row, flag\_row would become greater than '5', therefore a check for this condition increments '5' in flag\_row and resets flag\_col to '5'. This would essentially mean that 4 full rows and 5 pixels of the next 5 rows of image are going to be read. In this way, 25 values have been successfully extracted and sent for weighted voting using magnitude.

The architecture of this algorithm is shown in figure 15

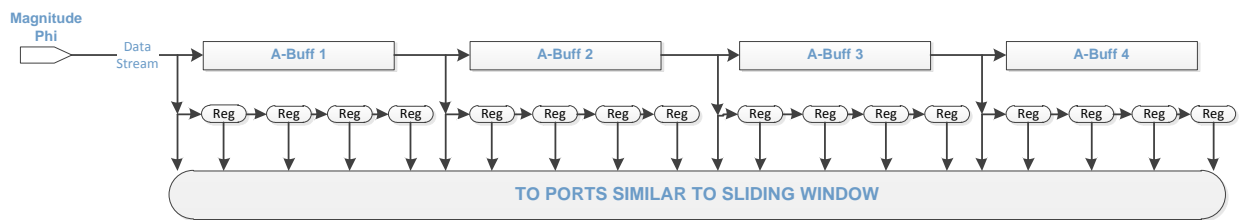


Figure 15: Architecture to attain 5x5 pixel cell

## 5.8 Re-Simulation of MATLAB code

To make the project a working hybrid, some changes were required to be made to the MATLAB model. As a comparison, gradients were computed through the convolution formula in previous MATLAB model. In the modified version, an equation similar to the VHDL model is used. After calculation of the gradients, the highest gradient is extracted using comparison equations. The magnitude is then calculated using the equation given below

$$mag = |G_x| + |G_y|$$

Orientations were found in the previous model using the “atan2” trigonometric function. In this modified version, the same look-up table used in VHDL model is implemented, since it is a hybrid and both models are required to work closely, it was necessary to implement the two models in a similar fashion to obtain closely related results.

These new changes are implemented as a function that returns the magnitude matrix and the theta matrix. The main function “ComputeHOG” earlier took one input that was the image whose HOG was intended. In the new model it is changed to take three inputs, namely, magnitude, theta and the image. Here the image provides the dimension information, and the magnitude and theta are used to calculate the HOG.



## 6 Results and Discussions

### 6.1 Results based on MATLAB simulation

After some experimentations a variety of results have been accumulated. The whole experiment was conducted with the view that the algorithm would also be implemented on hardware. In hardware resources are scarce. Before proceeding to the suggested method used by Dalal and Triggs an attempt was made to reduce the size of the HOG before training.

There were also different ways that the experiments are conducted. The testing of the algorithm was mainly performed on

- ⊕ MIT Pedestrian dataset
- ⊕ Non-pedestrian dataset created by the author using arbitrary images with no humans
- ⊕ Inria Pedestrian dataset[7]
- ⊕ Trained Image
- ⊕ Untrained Image

Table 1 defines the summary of how results are accumulated.

Generation of HOG	Training		Testing
Histogram of HOG generated	Small Data Set	Large Data Set	MIT pedestrian Database
			Negative Images Database
Original Full Vector of HOG generated	Negative Images re-sized	N/A	Inria Pedestrian Database
			Large image used in creating negative database
HOG reduced to 9 binned values.	Negative Images Cropped	Negative Images Cropped	Arbitrary Large Image

Table 1: Summary of accumulation of results

In the above table, a small data set for training consisted of 24 positive images taken from the MIT database and 24 negative images. A large dataset of training consists of 924 positive images taken from the MIT database and 1368 negative images.

### 6.1.1 Small Training Dataset and Analysis

A small data set for training is constituted from 24 positive images taken from the MIT database and 24 negative images.

#### Training Dataset:[4.1.1]

- ⊕ Small Dataset,
- ⊕ Negative: 24 re-sized images used,
- ⊕ Positive: 24 MIT pedestrian images used.

**HOG:** Generated using histogram of HOG [4.2.1]

#### Testing:

MIT Pedestrian database (Total positive images : 924)	
Human	564
No human	360 (0.38) false negative
Negative Images Database (Total negative images : 1368)	
Human	217 (0.15) false positive
No human	1151
Inria Pedestrian Database (Total positive images : 1132)	
Human	283
No human	849 (0.75) false negative

**Table 2:** various databases tested after taking histogram of HOG, using small dataset for training. The training is done using re-sized negative images.

**Training Dataset: [4.1.2]**

- ⊕ Small Dataset,
- ⊕ Negative: 24 cropped images used,
- ⊕ Positive: 24 MIT pedestrian images used.

**HOG:** Generated using histogram of HOG [4.2.1]

**Test:**

<b>MIT Pedestrian Database (Total positive images : 924)</b>	
Human	514
No human	410 (0.44) false negative
<b>Negative Images Database (Total negative images : 1368)</b>	
Human	209 (0.15) false positive
No human	1159
<b>Inria Pedestrian Database (Total positive images : 1132)</b>	
Human	229
No human	903 (0.79) false negative

**Table 3: various databases tested after taking histogram of HOG, using small dataset for training. The training is done using cropped negative images.**

**Analysis:**

Up to this point, the resized non-human images showed a better performance than the cropped non-human images. However, attention should be given to the way that the histogram is being calculated. A histogram, for example, counts the occurrence of different values. This loses the spatial information gathered while block normalizing. In addition to this loss, it feeds on a number of values i.e. the more the values the better it performs. Since the resized image would have significantly more gradients than the cropped image, it can be seen that the training conducted using resized negative images performs better as compared to the training conducted using cropped negative images.

The same procedure is applied to the full output HOG vector to evaluate what the result yields.

#### Training Dataset: [4.1.1]

- ⊕ Small Dataset,
- ⊕ Negative: 24 re-sized images used,
- ⊕ Positive: 24 MIT pedestrian images used.

**HOG:** HOG descriptor generated as a vector of 1368 elements [4.2.2]

**Test:**

<b>MIT Pedestrian Database (Total positive images : 924)</b>	
Human	804
No human	120 (0.12) false negative
<b>Negative Images Database (Total negative images : 1368)</b>	
Human	593 (0.43) false positive
No human	775
<b>Inria Pedestrian Database (Total positive images : 1132)</b>	
Human	380
No human	752 (0.66) false negative

**Table 4: various databases tested after generating full vector of HOG, using small dataset for training. The training is done using re-sized negative images.**

**Training Dataset: [4.1.2]**

- ⊕ Small Dataset,
- ⊕ Negative: 24 cropped images used,
- ⊕ Positive: 24 MIT pedestrian images used.

**HOG:** HOG descriptor generated as a vector of 1368 elements [4.2.2]

**Test:**

<b>MIT Pedestrian Database (Total positive images : 924)</b>	
Human	821
No human	103 (0.11) false negative
<b>Negative Images Database (Total negative images : 1368)</b>	
Human	308 (0.22) false positive
No human	1060
<b>Inria Pedestrian Database (Total positive images : 1132)</b>	
Human	650
No human	482 (0.42) false negative

**Table 5: various databases tested after generating full vector of HOG, using small dataset for training. The training is done using cropped negative images.**

**Analysis:**

To prove the above point that a loss in spatial information affects the detection of a pedestrian within a standard image window of 128x64, the output HOG vector was fed, without reducing its size, to the training classifier.

When the HOG descriptor, generated as a vector of 1368 elements, was applied it preserves the spatial information and thus, the result generated after training the classifier using cropped negative images, naturally performs better.

In the next experiment, an attempt to contract the size of a very long HOG vector is made and the test vector of 9 values was used to evaluate its performance.

### Training Dataset: [4.1.2]

- ⊕ Small Dataset,
- ⊕ Negative: 24 cropped images used,
- ⊕ Positive: 24 MIT pedestrian images used.

**HOG:** HOG reduced to 9 values [4.2.3]

**Test:**

<b>MIT Pedestrian Database (Total positive images : 924)</b>	
Human	823
No human	101 (0.10) false negative
<b>Negative Images Database (Total negative images : 1368)</b>	
Human	307 (0.22) false positive
No human	1061
<b>Inria Pedestrian Database (Total positive images : 1132)</b>	
Human	597
No human	535 (0.47) false negative

**Table 6:** various databases tested after reducing the HOG to 9 binned values, using small dataset for training. The training is done using cropped negative images.

### Analysis:

The performance of the newly designed 9 valued HOG vector appears to be closely related to the HOG descriptor generated as a vector of 1368 elements. However, it may be too soon, when using a small training set to evaluate the performance, to draw conclusions.

### 6.1.2 Large Training Dataset and Analysis

Therefore, a large dataset of training images consisted of 924 positive images taken from the MIT database and 1368 negative images was used.

#### Training Dataset: [4.1.3]

- ⊕ Large Dataset,
- ⊕ Negative: 1368 cropped images used,
- ⊕ Positive: 924 MIT pedestrian images used.

**HOG:** Generated using histogram of HOG [4.2.1]

**Test:**

MIT Pedestrian Database (Total positive images : 924)	
Human	700
No human	224 (0.24) false negative
Negative Images Database (Total negative images : 1368)	
Human	143 (0.10) false positive
No human	1125
Inria Pedestrian Database (Total positive images : 1132)	
Human	666
No human	466 (0.41) false negative

Table 7: various databases tested after taking histogram of HOG, using large dataset for training. The training is done using cropped negative images.

**Training Dataset: [4.1.3]**

- ⊕ Large Dataset,
- ⊕ Negative: 1368 cropped images used,
- ⊕ Positive: 924 MIT pedestrian images used.

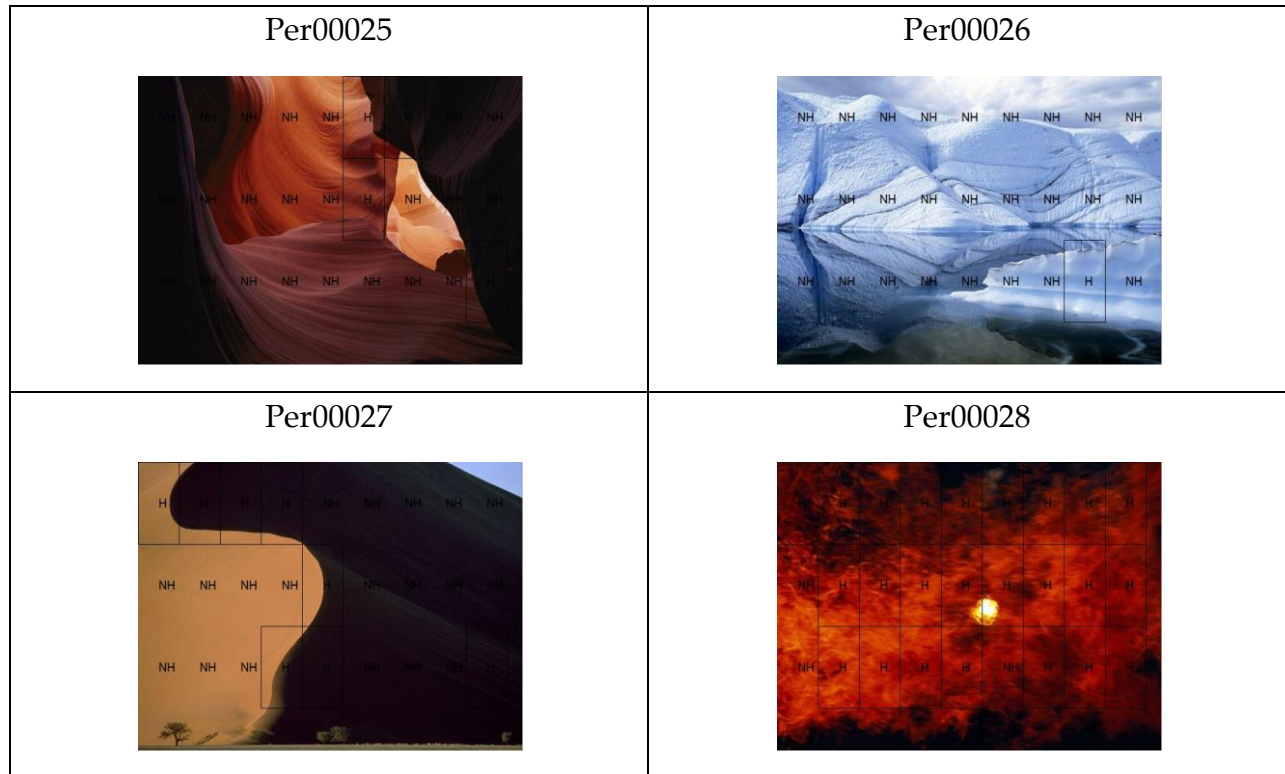
**HOG:** HOG descriptor generated as a vector of 1368 elements [4.2.2]

**Test:**

<b>MIT Pedestrian Database (Total positive images : 924)</b>	
Human	910
No human	14 (0.015) false negative
<b>Negative Images Database (Total negative images : 1368)</b>	
Human	12 (0.0088) false positive
No human	1356
<b>Inria Pedestrian Database (Total positive images : 1132)</b>	
Human	808
No human	324 (0.28) false negative

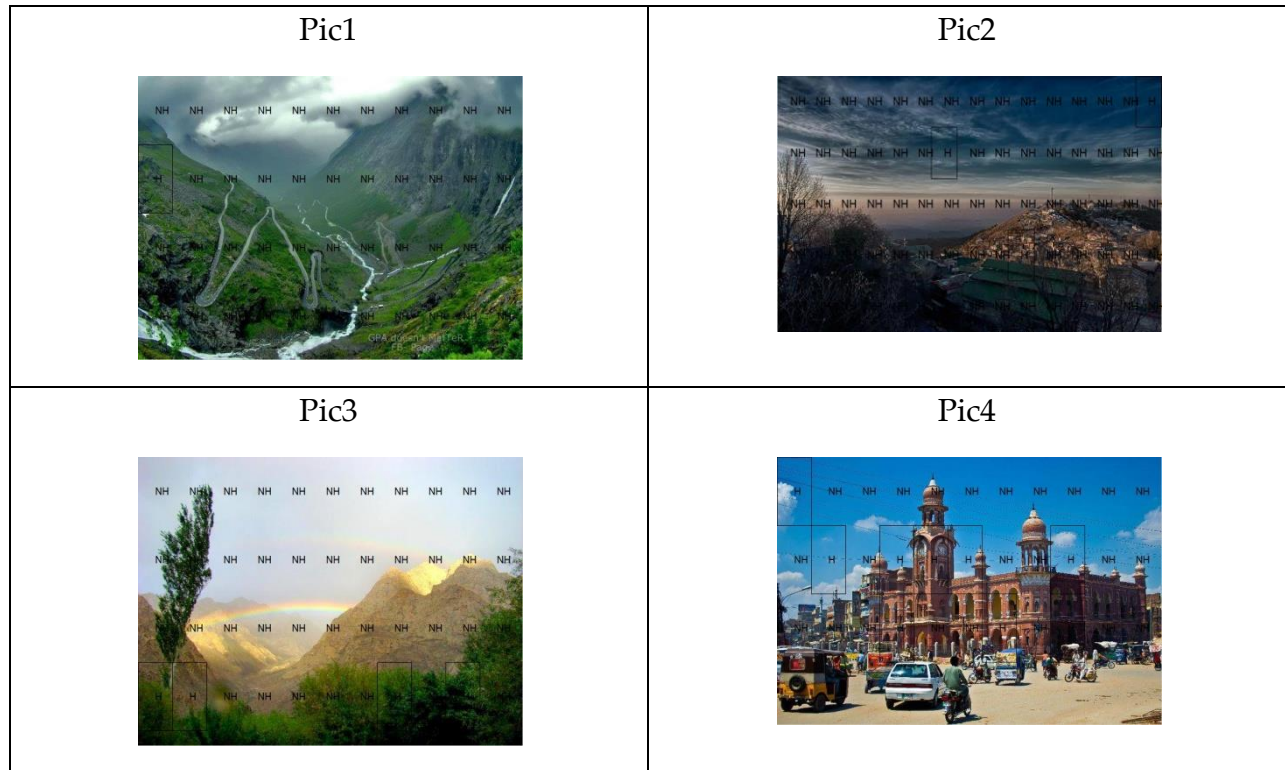
**Table 8: various databases tested after generating full vector of HOG, using large dataset for training. The training is done using cropped negative images.**



**Test on Trained[4.3] Image[8]:****Computation Results**

Per00025		Per00026	
Human	4 (0.14) false pos.	Human	1 (0.03) false pos.
No human	23	No human	26
Per00027		Per00028	
Human	8 (0.29) false pos.	Human	24 (0.88) false pos.
No human	19	No human	3

**Table 9: Detection window of 128x64 swept on trained images after generating full vector of HOG, using large dataset for training.**

**Test on Untrained[4.3] Image[9][10]:****Computation Results:**

Pic1		Pic2	
Human	1 (0.02) false pos.	Human	4 (0.05) false pos.
No human	43	No human	71
Pic3		Pic4	
Human	4 (0.09) false pos.	Human	9 (0.27) false pos.
No human	40	No human	24

Table 10: Detection window of 128x64 swept on untrained images after generating full vector of HOG, using large dataset for training.

**Training Dataset: [4.1.3]**

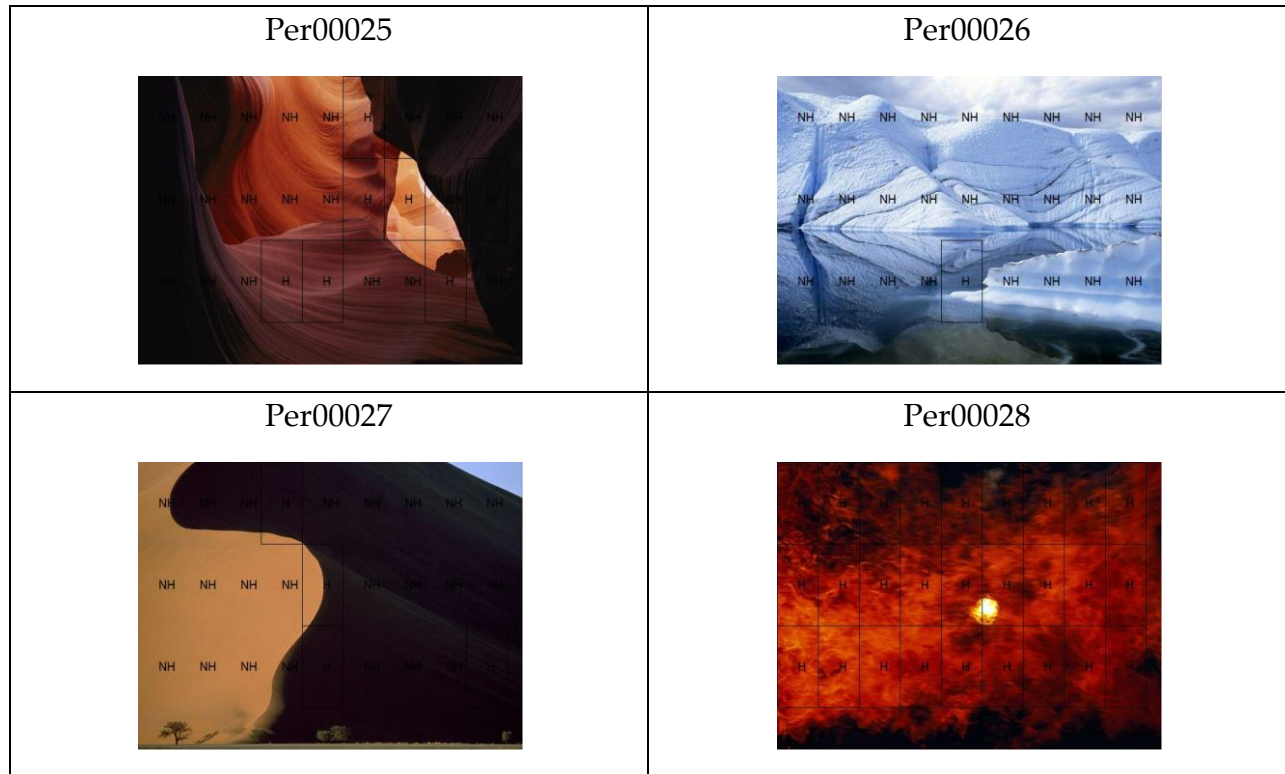
- ⊕ Large Dataset,
- ⊕ Negative: 1368 cropped images used,
- ⊕ Positive: 924 MIT pedestrian images used.

**HOG:** HOG reduced to 9 values [4.2.3]

**Test:**

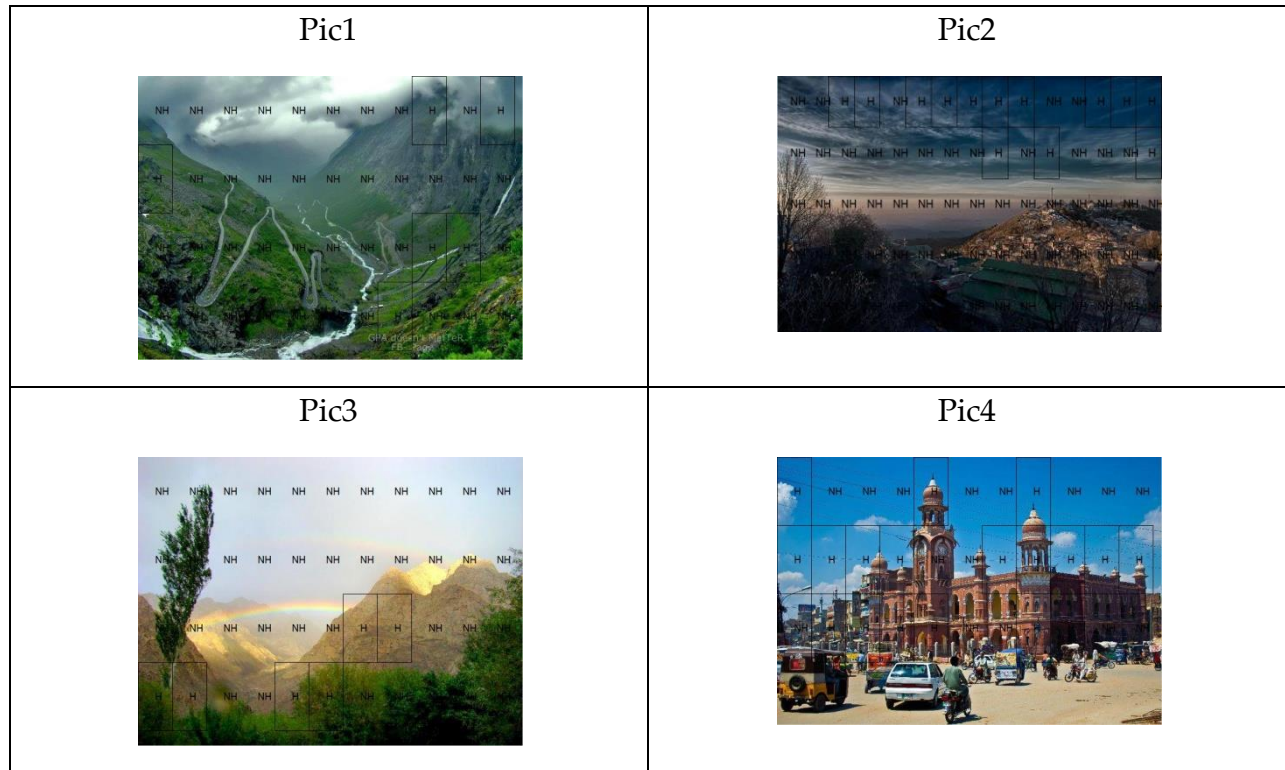
<b>MIT Pedestrian Database (Total positive images : 924)</b>	
Human	837
No human	87 (0.094) false negative
<b>Negative Images Database (Total negative images : 1368)</b>	
Human	95 (0.069) false positive
No human	1273
<b>Inria Pedestrian Database (Total positive images : 1132)</b>	
Human	582
No human	550 (0.48) false negative

**Table 11:** various databases tested after reducing the HOG to 9 binned values, using large dataset for training. The training is done using cropped negative images.

**Test on Trained [4.3] Large Images:****Computation Result:**

Per00025		Per00026	
Human	7 (0.25) false pos.	Human	1 (0.03) false pos.
No human	20	No human	26
Per00027		Per00028	
Human	4 (0.14) false pos.	Human	27 (1.00) false pos.
No human	23	No human	0

**Table 12:** Detection window of 128x64 swept on trained images after reducing the HOG to 9 binned values, using large dataset for training.

**Test on Untrained [4.3] Large Image:****Computation Result**

<b>Pic1</b>		<b>Pic2</b>	
Human	6 (0.13) false pos.	Human	13 (0.17) false pos.
No human	38	No human	62
<b>Pic3</b>		<b>Pic4</b>	
Human	6 (0.13) false pos.	Human	18 (0.54) false pos.
No human	38	No human	15

Table 13: Detection window of 128x64 swept on untrained images after reducing the HOG to 9 binned values, using large dataset for training.

**Analysis:**

The results assist in assessing the increase in performance of the HOG descriptor, when a large dataset is used for training the classifier.

A change from a smaller training set to a larger training set, while using the histogram of HOG output vector, certainly increases the efficiency. It is, however, still not close to the desired result as spatial information is being lost.

However, on applying the same change of larger training data set on the full HOG output vector, a visible increase in efficiency is observed. There are certain difficult examples, where the algorithm fails to produce satisfactory results, such as that shown in “per00028 computation results” where the false positive is as high as 88%. For the MIT pedestrian database the improvement is from 11% (0.11) false negative to 1%(0.01) false negative.

The attempt to reduce the HOG vector did not yield a significant improvement in efficiency when the training dataset was increased. It improved from 10.93% (0.109) false negative to 9%(0.094) false negative.

The results of the MIT pedestrian database using large dataset set for training the SVM are given in table 14

	MIT Pedestrian Dataset (false Negative)
HOG descriptor generated as a vector of 1368 elements [4.2.2]	1%
HOG reduced to 9 values [4.2.3]	9.4%
Dalal and Triggs Result	1%

**Table 14: Comparison with Dalal and Triggs Result**



### 6.1.3 Comparison and Discussion of Result with Dalal and Triggs Research Paper

Dalal and Triggs achieved a near perfect result on the MIT pedestrian dataset. After the application of the full HOG vector, a miss rate of 1% was achieved, this appeared to be close to Dalal and Triggs HOG performance. However, they then created a dataset of 1800 pedestrian images with a large range of poses and backgrounds. The miss rate reported with this dataset is 10.4%. As their dataset was not available, an exact comparison is not possible but, on applying this HOG to the Inria pedestrian dataset, which has a very wide range of poses and backgrounds, it is proved to be possible to generate a miss rate as low as 28%. The difference may be on the basis of the training set used, the dataset on which the HOG is tested and also on the way the training was concluded.

## 6.2 Results based on VHDL-MATLAB hybrid Model

After changing the MATLAB model, a new analysis is required in order to gain an understanding of its behavior.

### Training:

- ⊕ Large Dataset,
- ⊕ Negative: 1368 cropped images used,
- ⊕ Positive: 924 MIT pedestrian images used.

**HOG:** Original full vector of HOG generated

### Test:

MIT Pedestrian Database (Total positive images : 924)	
human	912
nohuman	12 (0.012) false negative
Negative Images Database (Total negative images : 1368)	
human	16 (0.011) false positive
Nohuman	1352

Figure 16: Modified MATLAB code testing with full vector of HOG.

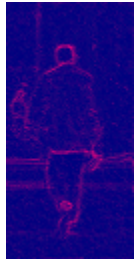
**Training:** Large Dataset, Negative cropped images

**HOG:** HOG reduced to 9 binned values


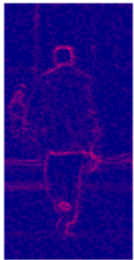
MIT Pedestrian Database (Total positive images : 924)	
human	816
nohuman	108 (0.11) false negative.

Figure 17: Modified MATLAB code testing with HOG reduced to 9 binned values.

The conclusion based on the analysis above is that the results are similar to those for the previous code. This provides confirmation that this modified code can be safely used for classifying data. In the report it has been shown how the image of mag/theta appear after the simulation. The image is shown below as a reference.



The result obtained after classifying human and non-human images is as follows.

Original pic	Classified pic
	<p>human</p> 



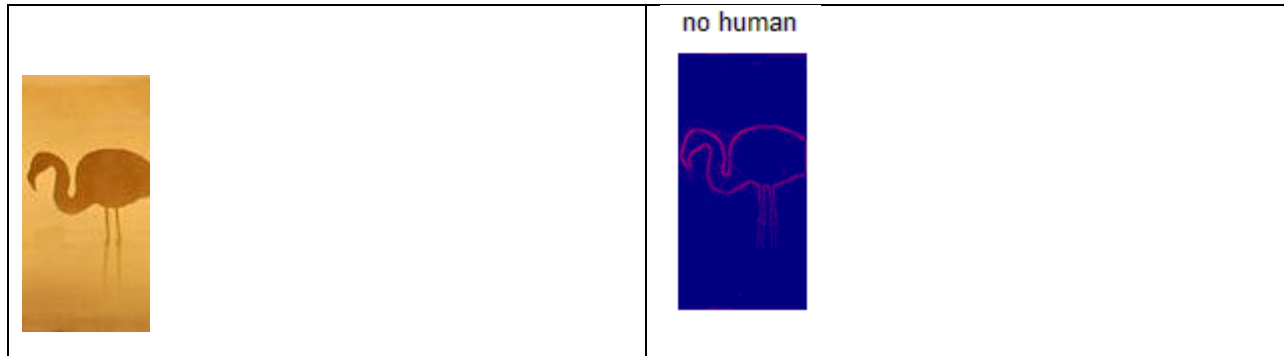


Figure 18: Result after classification using Hybrid Model

### 6.3 Results using VHDL Model

Magnitude and Theta values simulated in VHDL

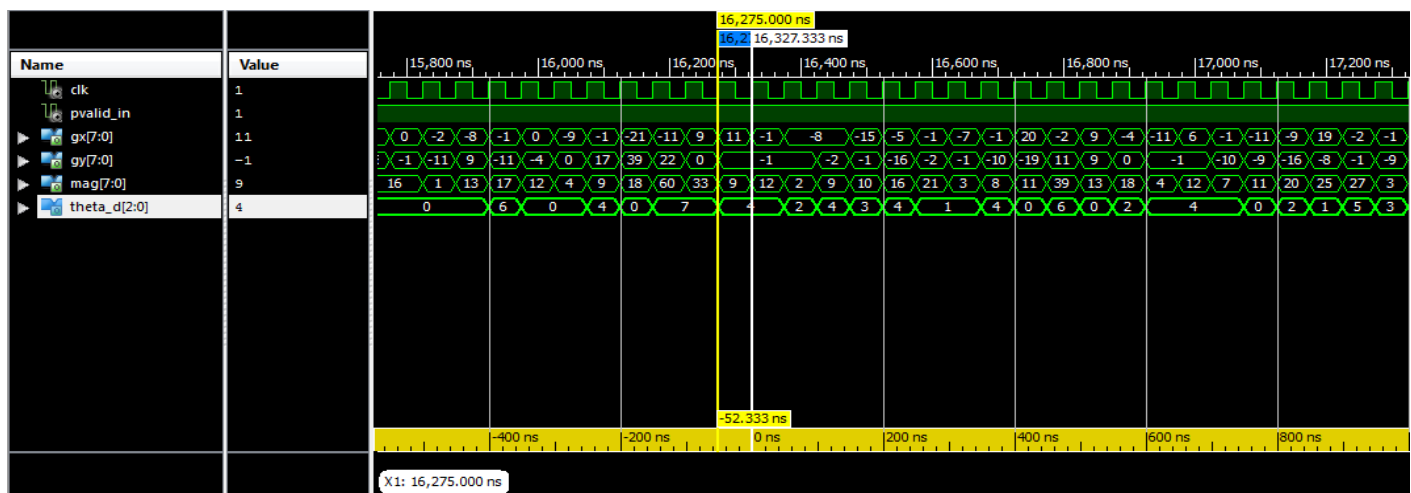


Figure 19: Magnitude and theta values as simulated in VHDL

#### Analysis:

The first marker in yellow shows the two values of  $G_x = 11$  and  $G_y = -1$ . A simplified formula was used for calculating magnitude, by taking the absolute of both values and adding them. In the next clock cycle, identified by the white marker, the magnitude is correctly determined as 12.

## Histogram values simulated in VHDL

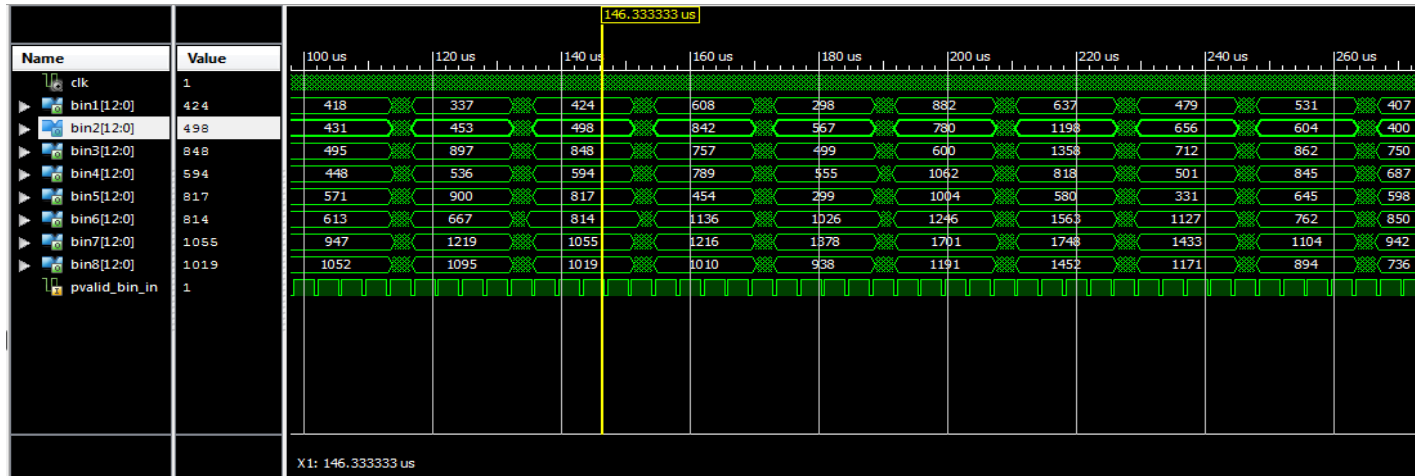


Figure 20: Histogram Values after VHDL simulation.

## The timing summary of VHDL model synthesized till histogram generation

## Timing Summary:

Speed Grade: -3

Minimum period: 9.860ns (Maximum Frequency: 101.418MHz)  
 Minimum input arrival time before clock: 5.534ns  
 Maximum output required time after clock: 4.022ns  
 Maximum combinational path delay: No path found

Figure 21: Timing Summary VHDL Model.

## 7 Conclusion

### 7.1 MATLAB Simulation:

Extraction of a human from an image with a complex background is not possible with the conventional image processing steps. The fact that the segmentation of a human is very difficult necessitates an alternative solution. The HOG descriptor emerges as an excellent technique for this purpose.

Various HOG descriptors were used in this thesis to evaluate the performance of classification. When taking the histogram of the HOG using  $n_{\text{Bin}} = 9$ , important spatial information was lost which caused a decrease in performance. When the HOG was binned into 9 values, the descriptor appeared to be initially promising, but, as the training set increased and the algorithm was more rigorously tested, the full length HOG descriptor was found to be the best solution.

### 7.2 VHDL Simulation

The VHDL model created up to the histogram, is fully synthesizable. It has a minimum period of 9.86ns and the whole process runs at a frequency of 101.418 Mhz, which is a very high speed considering the complexity of the hardware. However this speed will reduce as a more resource consuming block normalization module is yet to be developed. The fsync and rsync ports are given due diligence in each module. These two signals assist in finally attaining a stable video. As the architecture developed works only on one image at a time, these signals were not fully tested.



## 8 Future Work

There is still a great deal to be conducted in terms of the development of the final hardware architecture. Many resources, specifically those mentioned in the references, must be studied for further ideas. Cell histograms are 13 bit long vectors, thus cannot be transferred using the current test bench, which only allows a transfer of an 8 bit port. Also the current test bench generates an image as an output, and the RGB values of that image are used to store the port values. It is thus necessary to develop a specific testbench to export the 8 13-bit ports to MATLAB for testing. A better flow of work would be to first run and analyze the whole code using the histograms generated via VHDL and then move forward to block normalization.

In block normalization it would be an option to test the code after calculating the 9 value HOG, as conducted in the thesis. This would provide an understanding of the behavior of the HOG vector as it would take less resources since it is small and, in addition, the difference of human/nonhuman plots can be seen visually.

In the suggested “Block diagram for FPGA implementation” two array buffers were made with the block normalization module. This is suggested as block normalization follows the same pattern as that for the sliding Window. One block contains 3x3 cells, therefore one location of an array would contain 8 values each of which are 13 bits long. Other techniques which may reduce the complexity of the algorithm are encouraged to be researched.

Block norm calculation is a separate module. It again follows a similar pattern to that for the sliding Window. The values are extracted using the sliding window technique, and then they are used in other blocks.

In the Dalal and Triggs paper, they mentioned the 16 pixel separation of person of all sides of the border. This same rule also applies in the hardware. However, if the pixels decrease, which means that the person is being zoomed into, the results are affected. Similarly, if the pixels are increased, which is the person being zoomed out, the results vary. This process questions the scale invariability of the algorithm. Further work can be conducted to incorporate scale invariability. One technique suggested by Negi et al. is to use many weak classifiers to obtain a strong classifier, which then would be able to detect the person. Another issue is, if a person comes so close to the camera that it would be termed as non-human. In this case, one solution would be to classify the person when in frame and then mark them, or track them until they move totally out of frame.

## 9 References

1. Navneet Dalal , Bill Triggs, "Histograms of Oriented Gradients for Human Detection," in Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)
2. <http://cbcl.mit.edu/software-datasets/PedestrianData.html>
3. Negi,K.,Dohi,K.,Shibata,Y.and Oguri,K.: Deep pipelined one-chip FPGA implementation of a real-time image-based human detection algorithm, IEEE FPT (2011)
4. Kadota, R., Sugano, H., Hiromoto, M., Ochi, H., Moyamoto, R. and Nakamura, Y.: Hardware Architecture for HOG Feature Extraction, Proc. 2009 International Conference on Intelligent Information Hiding and Multimedia Signal Processing, pp.1330-1333, Washington DC, USA: IEEE Computer Society (2009)
5. D. G. Lowe. Distinctive image features from scale-invariant key points. IJCV, 60(2):91–110, 2004
6. T. Joachims. Making large-scale svm learning practical. In B. Schlkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*. The MIT Press,Cambridge, MA, USA, 1999.)
7. <http://lear.inrialpes.fr/data>
8. <http://photography.nationalgeographic.com>
9. <http://hdw.eweb4.com/out/846511.html>
10. <http://www.panoramio.com/photo/55272777>
11. Wojek, C.; Walk, S.; Schiele, B., "Multi-cue onboard pedestrian detection," Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on , vol., no., pp.794,801, 20-25 June 2009  
doi: 10.1109/CVPR.2009.5206638
12. Geismann, P.; Schneider, G., "A two-staged approach to vision-based pedestrian recognition using Haar and HOG features," Intelligent Vehicles Symposium, 2008 IEEE , vol., no., pp.554,559, 4-6 June 2008
13. Junfeng Ge; Yupin Luo; Gyomei Tei, "Real-Time Pedestrian Detection and Tracking at Nighttime for Driver-Assistance Systems," Intelligent Transportation Systems, IEEE Transactions on , vol.10, no.2, pp.283,298, June 2009.
14. Mizuno, K.; Terachi, Y.; Takagi, K.; Izumi, S.; Kawaguchi, H.; Yoshimoto, M., "Architectural Study of HOG Feature Extraction Processor for Real-Time Object

- Detection," Signal Processing Systems (SiPS), 2012 IEEE Workshop on , vol., no., pp.197,202, 17-19 Oct. 2012
15. T. P. Cao, et al., "Real-Time Vision-Based Stop Sign Detection System on FPGA," in Proceedings of Digital Image Computing: Techniques and Applications. Los Alamitos, CA, USA: IEEE Computer Society, 2008, pp.465–471.
  16. M. Hiromoto, et al., "Hardware Architecture for High-Accuracy Real-Time Pedestrian Detection with CoHOG Features," IEEE ICCVW 2009.
  17. Bauer, S.; Kohler, S.; Doll, K.; Brunsmann, U., "FPGA-GPU architecture for kernel SVM pedestrian detection," Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on , vol., no., pp.61,68, 13-18 June 2010.
  18. S. Bauer, et al., "FPGA Implementation of a HOGbased Pedestrian Recognition System," MPC-Workshop, July, 2009.