



Michigan State University College of Engineering

ECE 480

Design Team 4

Apr. 26th 2011

FPGA Image Processing for Driver Assistance Camera

Final Report

Design Team:

Buether , John Management

Frankfurth, Josh Documentation

Lee, Meng-Chiao Web Development

Xie, Kan Presentation Design



Executive Summary:

Passenger safety is the primary concern and focus of automobile manufacturers today. In addition to the passive safety equipment, such as seatbelts and airbags, technology based active safety mechanisms are being developed and incorporated into all types of commercial and industrial vehicles and may soon be required by law. Current trends are requiring automobile manufacturers to include a multitude of technology based safety equipment including ultrasonic sensors and back-up cameras and even forward facing cameras. Historically, cameras placed in vehicles give the driver an unaltered view from behind the vehicle; however, with the sponsorship of Xilinx, Michigan State University's ECE 480 Team 4 has designed and implemented algorithms that will detect and classify objects, allowing the driver to be alerted.

This system draws the driver's attention to objects either behind or in front of the vehicle, by marking them with targets. In doing so, the driver will be less likely to overlook objects that may create a safety hazard. The team has combined the techniques of Histogram of Oriented Gradients (HOG) and Support Vector Machine (SVM) to create a system that will both accurately and efficiently detect and visually alert the driver to hazardous objects. Implementation of the algorithm utilizes Xilinx's Spartan-3A Field Programmable Gate Array (FPGA) development board and Xilinx's System Generator tools.

Acknowledgement

Michigan State University's ECE 480 Design Team 4 would like to give a special thanks to those who helped contribute to the success of the driver assistance camera algorithm:

Xilinx: Mr. Paul K. Zoratti,

for his support and the founding of project

Michigan State University, ECE Department: Mr. Michael Shanblatt,

for his direction and his sharing of lifelong career skills

Michigan State University, ECE Department: Dr. Tongtong Li,

for her weekly advice and support of the project

Michigan State University, ECE Department: Dr. Anil K. Jain's Research Group, with special acknowledgement to Serhat Selcuk Bucak

for their advices on object detection implementation and explanation of machine learning algorithms

Michigan State University, ECE Department: Dr. William Punch

for his advice on using the Michigan State University High Performance Computing Center to accomplish goals that seem impossible for ordinary machines.

Table of Contents

Design Team:.....	1
Executive Summary:.....	3
Acknowledgement	4
Chapter 1: Introduction and Background	8
1.1 Introduction	8
1.2 Background	9
Chapter 2: Exploring the solution space and selecting a specific approach	10
2.1 Design Specifications	10
2.2 Fast Diagram	11
2.3 Conceptual Design	11
2.3.1 Histogram of Oriented Gradients.....	11
2.3.2 Support Vector Machine	11
2.4 Proposed Design Solution	12
2.5 Risk Analysis	12
2.6 Budget	12
Chapter 3: Technical Description of Work Performed	13
3.1 Software Design	13
3.1.1 Histogram of Oriented Gradients.....	14
3.1.2 Support Vector Machine	14
3.1.3 Normalization in Software	15
3.1.4 Matlab Process Flow Compilation Diagram	16
3.1.5 Matlab Development Environment	17
3.2 Hardware Design.....	17
3.2.1 Histogram Orientated Gradients	23
3.2.1.1 Cell Control Signals.....	23
3.2.1.2 Bin Selection.....	24
3.2.1.3 Memory Bank.....	26
3.2.2 Sliding Window	27
3.2.3 Normalization.....	31
Chapter 4: Test Date with Proof of Functional Design	32

4.1	Testing Histogram of Oriented Gradient	32
4.2	Testing Support Vector Machine	37
Chapter 5: Final Cost, Schedule, Summary and Conclusion		38
5.1	Final Cost	38
5.2	Schedule	39
5.3	Summary	40
5.4	Conclusion	41
Appendix 1: Technical Roles, Responsibilities, and Work Accomplished		42
A1.1	Meng-Chiao, Lee – Web Development	42
A1.2	John Buether – Manager	43
A1.3	Kan Xie – Presentation Design	45
A1.4	Josh Frankfurth – Documentation	46
Appendix 2: Literature and Website Reference.....		47
Appendix 3: Detailed Technical Attachments		48
A3.1	Histogram of Orientated Gradients in Matlab.....	48
A3.2	TestClassifier.m Matlab Function	49
Appendix 4: Gantt Chart		50

Chapter 1: Introduction and Background

1.1 Introduction

Back-Over crashes have become one of the main car accidents which cause fatalities globally. According to National Highway transportation (NHTSA), around 292 fatalities and 18,000 injuries occur each year as a result of back-over accident involving all types of vehicles, and children and elder are the most common victims. In order to prevent back-over crashes, the U.S. Department of Transportation proposed that automakers to install back-up camera in all new vehicles in 2014 to help drivers see into the blind zones directly behind vehicles. The proposal was designed to provide rear view when the vehicles are in reverse to keep drivers from running over pedestrians who might be crossing behind their vehicles. Even the proposal has not been passed by Congress, OEM and aftermarket has started to increase the placement of back-up camera in vehicles.

From the study of ISuppli, one of the leading market research firms, it showed that the selling rate of aftermarket back-up camera are growing at a steady pace with estimated sale of 182,000 units in 2010 from 125,000 in 2009. The current trend will not only affect the sales of aftermarket back-up camera, but also the sales of Original Equipment Manufacturers (OEM). ISuppli forecasts that the growth will continue until 2015 to reach 813,000 units for aftermarket and 3,352,000 units for OEM. From the facts that the increasing placements of the product in new cars and the noticeable selling rate, it indicated that the customer awareness is already gaining every year. Therefore, in order to improve the drivers' comfort and young drivers' confidence, Xilinx, a leader in programmable logic products, decided to update the current back-up camera into a next stage. With the help from Xilinx, Michigan State University's ECE 480 Team 4 was assigned to create an algorithm to visually alert the driver of pedestrians seen within the back-up camera and approximate the distance in between using Xilinx's Xtreme Spartan-3A board. It is a continuous project from the previous team, and some of the tasks had been done such as edge detection and certain level of human detection. With the sponsorship by Xilinx, the team will be provided with the Xtreme Spartan-3A development, camera, and the company's copy written System Generator tools to develop a prototype. The ECE 480 Team 4 will optimize the algorithms which were done by previous team and move the design to the next level by creating new algorithms such as distance measurement and vehicle detection.

1.2 Background

Automotive backup cameras are becoming a must have in today's automotive world. When used properly backup cameras will not only improve driver awareness by providing them with a video of what's behind them, but they will also monitor what is happening behind the vehicle at all times. Having camera's in the car will allow the vehicle to alert the driver of dangers they may not be aware of or in extreme situations stop the car if danger is detected. Such dangers include lane departure warning, as well as near object collision.

The use of cameras is likely becoming a safety requirement by the NHTSA (to be determined by ruling early 2011) and will likely require all passenger vehicles to be equipped with rear-view cameras.

Design Team 4 will be continuing research and algorithm development started by Design Team 3 from Fall Semester 2010. Design Team 3 was able to implement an edge detection algorithm as well as a skin detection algorithm using a combination of matlab/Simulink interface with Xilinx ISE Design Suite and Xilinx Video Starter Kit. Upon inspection of Team 3's design for algorithms Design Team 4 will increase the effectiveness of these algorithms while also creating new algorithms to advance the options available to Xilinx. Team 4 will also undergo the task of attempting to finish team 3's object detection from last semester.

Team 4 will also undergo research into adding an additional camera (inferred or depth cameras) to our current hardware spec. Adding a camera would allow for more information for the algorithms and help with object detection and monocular ranging.

Chapter 2: Exploring the solution space and selecting a specific approach

2.1 Design Specifications

The goal of this project is to develop an FPGA driver assistance rear view camera algorithm by creating a Histogram of Oriented Gradients algorithm and Support Vector Machine. The team is not using the top of the line performing Field Programmable Gate Arrays (FPGAs) and the performance of the product will be mainly base on the algorithms created by the design team. Therefore, there are several design specifications to be met in the implementation of this project as follow.

- **Functionality**
 - Clearly indicate detect the object of interest in the driver's camera display
 - Able to operate in noisy condition
 - Minimal errors
- **Speed**
 - Real-time detection (minimal 20 frames per second)
 - Continuous seamless buffer
- **Low Maintenance**
 - Easily accessible for future programmers to encourage future development

2.2 Fast Diagram

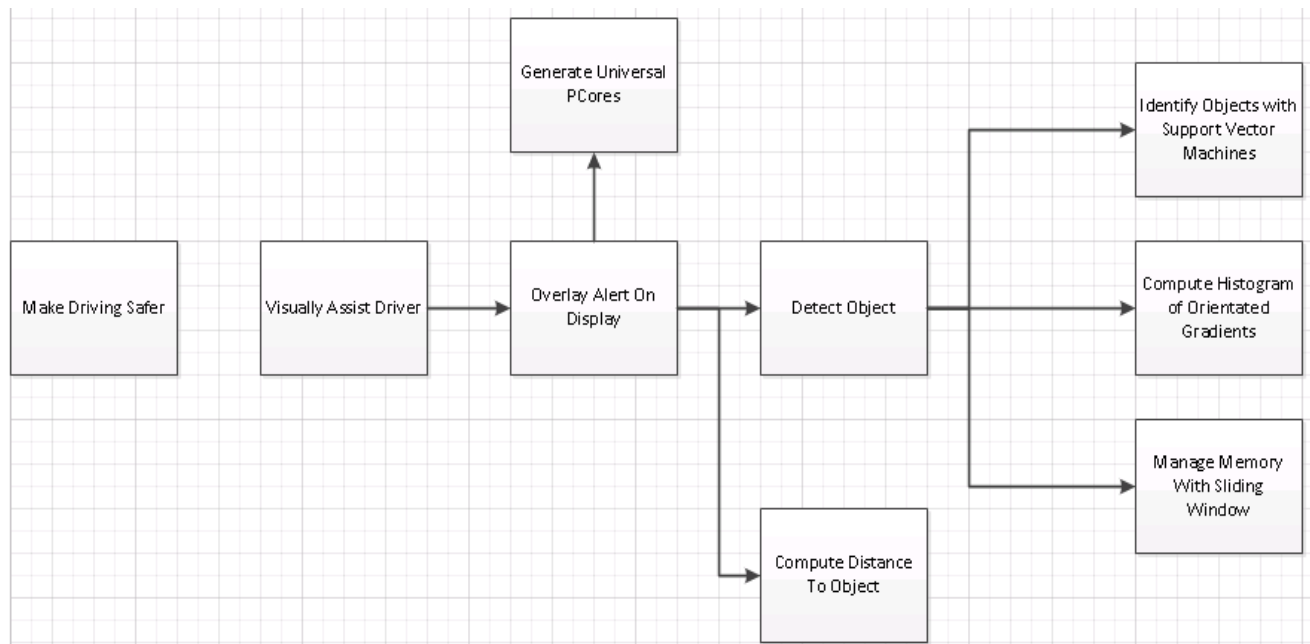


Figure 1: Fast Diagram

2.3 Conceptual Design

2.3.1 Histogram of Oriented Gradients

A histogram of oriented gradients is a measurement of the direction of fine-grain gradients in an area. This representation of data is meant to allow the classification of specific types of objects, particularly when used as data for a support vector machine. It can also be used to identify image features, or non-specific objects.

To compute a histogram of oriented gradients, you first calculate x and y gradients for each pixel in an image. Then you group the pixels into 'cells' of some particular shape, and have each pixel contribute to the histogram of oriented gradients for that cell by 'voting' for the orientation which is centered on that pixel. Typically the pixels' votes are weighted with a function of the position of the individual pixel, and the magnitude of the orientation centered on it.

2.3.2 Support Vector Machine

A support vector machine is fundamentally a binary classifier. You feed it data of interest, and it uses that data to answer a yes-or-no question relevant to the data. Each element of the data vector is used in a series of computations which have a scalar value as a result. These computations are called kernels of the support vector machine. The scalar values are then added together, and their sum is

evaluated. If the sum is above a certain threshold, the answer to the question is presumed to be yes. If it is below this threshold value, the answer is no.

Support vector machines can be 'trained' to identify arbitrary data sets, without any presumed knowledge of the data set. This makes them excellent at tackling problems with large quantities of data, and little theoretical knowledge linking the data to make any sense of it, such as image recognition. They are a near universal component of most classifier systems.

2.4 Proposed Design Solution

Design Team4 proposes that using Matlab and Simulink utilizing SVM toolboxes will allow the team to create a histogram of orientated gradients and a support vector machine is the best solution to the problem of detecting and classifying important objects. The team believes that using the Matlab built in SVM is the most suitable way to complete the design because the hardest part of the task will be handled by Matlab. The Xilinx suite allows for configuration and hardware design of the Xilinx board and therefore must be used. Matlab is a known solution and works well with ISE Design Suite and has the ability to generate the models for the Xilinx hardware. The team is designing the Matlab and Xilinx software to work together so that transitioning from Matlab to Xilinx models is possible.

2.5 Risk Analysis

Our project is based on an development board that is pre-build by Xilinx, according to the report from the last semester, the voltage on the development board has never exceed 5 volts, the risk of hazard by the board is very low.

However, our main concern to our project is the malfunction caused by our software algorithm. From the last semester's report, there were little time for the team to do failure testing and redirection, this will be one of the major tasks our team will be working on this semester to insure that real-time proper detection is achieved.

On the other hand, our team is still working on using new types of camera on the development board, we expect new compatibility problem will be encountered but we will be doing a lot of testing and new coding for the new hardware to make sure it works within the range designed.

2.6 Budget

All parts are donated from Xilinx and there is no need to spend any additional money. The purpose of the project is to use hardware available from Xilinx.

Chapter 3: Technical Description of Work Performed

3.1 Software Design

Software is a key element of Design Team Four's working environment. Our design team had to choose software at the beginning of the semester to develop our software on. Currently options for development for our Xilinx Spartan-3a are limited but they do exist. Many individuals choose to only use the Xilinx Tool Set also known as the Xilinx ISE Design Suite. Xilinx ISE Design Suite is a standalone development environment that contains all the tools needed for designing layouts for our Field Programmable Gate Array (FPGA). One of these tools is known as the Embedded Development Kit, or EDK. EDK is a separate environment from ISE design suite but often works with it for synthesis of a project to our embedded system (E.G. Xilinx FPGA). The synthesis process takes in all our files, and creates an optimized layout for speed and size. These files or designs are written usually in the form of a HDL (Hardware Descriptor Language).

HDL's such as Verilog and VHDL are not the easiest methods to develop hardware and gate level logic on. HDL's are often used to describe finished hardware so that it can be ported but because many organizations have spent lots of time, effort, and resources creating these HDL standards, they are often pushed as development languages. Instead of working with lots of files with inputs and output descriptors our team is taking a different approach to developing our algorithms on the FPGA. Our team is developing our algorithms using a plugin to Matlab known as Simulink. Simulink is a development environment that is very different from the Xilinx environment. Simulink allows for a graphical interface for the developers (Design Team 04) to work in. Our design team can drag and drop logic blocks into our development environment and work in a slightly higher level than using VHDL files. Using a graphical interface is important to us as only half of us have experience with working with VHDL, and being able to see the layout in gate logic is easier than creating layouts in files and attempting to visualize connections on our own. The other advantage of using Simulink is that it is able to access the Xilinx System Generator. Xilinx System Generator has the ability to take in a Matlab Model (our project) and then synthesize an optimized VHDL version of our project for programming on the board. This saves us a lot of time in some areas but at the cost of having to wait up to 45 minutes to generate the design layout for the board. Simulink also has the ability to do a full system simulation so we do not always have to synthesize a board layout using Xilinx System Generator.

The final decision to use Matlab and Simulink of simply using Xilinx only software was the ability to use the Matlab built in Bioinformatics Toolbox. The Bioinformatics tool box is a giant library of complicated resources on resources that we can only begin to comprehend. This library has the ability to implement algorithms used in genome and proteome folding and analysis, build applications for drug discovery, but most important to our design team, build a Machine Learning Algorithm profile known as a Support Vector Machine. The team at Matlab has given users the ability to create a SVM system in a matter of weeks instead of a matter of months. A support vector machine is the last stage in doing object detection for application and because Matlab has a base algorithm to work from we will be using

their tool kit to optimize our time and give Design Team 04 the ability to focus more on the Xilinx Product work rather than dedicate a lot of resources to an algorithm that be trained separately from the Xilinx software.

3.1.1 Histogram of Oriented Gradients

Instead of creating our own Histogram of Orientated Gradients algorithm we found an example and modified it to suit our needs. Modifications were made to change how many cell windows we were using in both directions of our input images. It should also be noted that this algorithm does not work for all images in Matlab, so they have to be preprocessed before computing the HOG algorithm. The HOG algorithm also does basic block normalization to improve accuracy of the descriptor.

The histogram of orientated gradients algorithm is appended in the section “Histogram of Orientated Gradients in Matlab”

3.1.2 Support Vector Machine

A Support Vector Machine is a Machine Learning Algorithm that creates a compact model of a specially designed training set. The model consists of a varying number of “Support Vectors” that are used to compare new sets of input data with the model. A support vector machine works by constructing what is known as a hyperplane between the trained data support vectors in the model. Design Team Four implemented a support vector machine that creates what is known as a linear maximum margin support vector machine. In the below image the green dots can represent the support vectors that are generated for our pedestrian images. And the blue vectors are generated for, all images that do not contain pedestrians. The hyperplane is the line that divides them with the most amount of margin as shown in the below image.

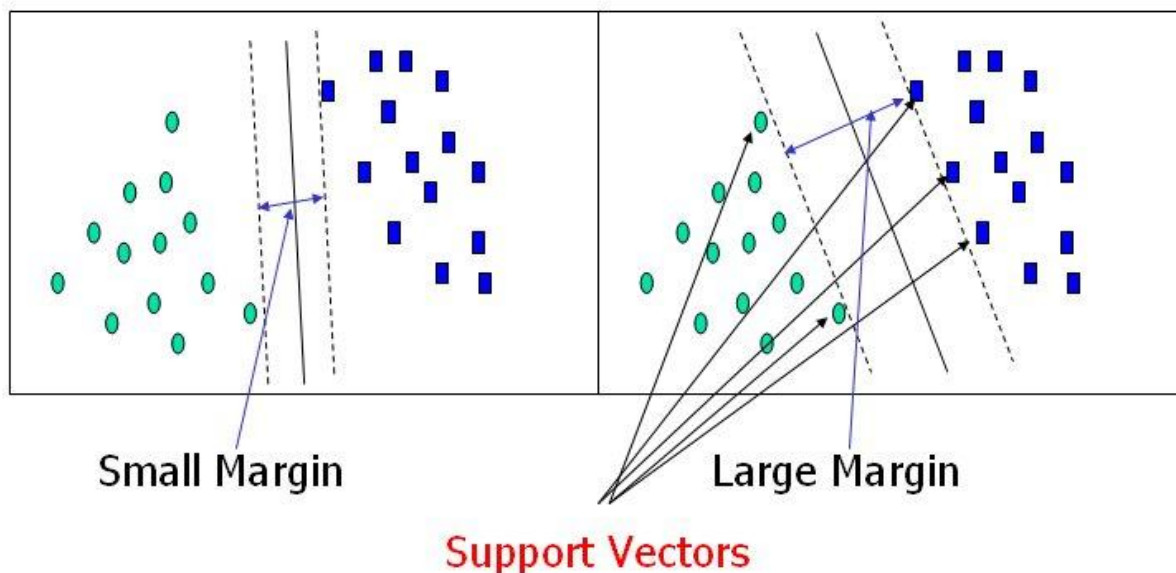


Figure 2: SVM Margin Differences

Design team four's support vector machine was developed by generating a histogram of orientated gradients for all of our training data and storing whether or not the image contained a pedestrian or not. We used a simplified database from MIT and the INRIA for positive images containing pedestrians and several databases for negative images. We found a few resources for creating support vector machines in Matlab but none of them implemented a histogram of orientated gradients to do so. We based the function call to the Matlab SVM Toolbox from an example we found on mathworks written by Omid Sakhi. The below text block takes in an image database and creates the model for our support vector machine. For a small version of our database the algorithm takes approximately 10 hours of cpu time to run. This was one of the reasons our design team had to get approval to access the High Performance Computing Center.

```
function NET = trainnet(IMGDB)

%~~~~~
options = optimset('maxiter',100000);
%~~~~~
T = cell2mat(IMGDB(2,:));
P = cell2mat(IMGDB(3,:));
net =
svmtrain(P',T', 'Kernel_Function','linear','Polyorder',2,'quadprog_opts',options,
'showplot',true);
fprintf('Number of Support Vectors: %d\n',size(net.SupportVectors,1));
classes = svmclassify(net,P');
fprintf('done. %d \n',sum(abs(classes-T')));
save net net
NET = net;
```

Figure 3: Train SVM Code Block

3.1.3 Normalization in Software

Normalization is the process of taking all the localized cells around a bin and computing a final descriptor based on the surrounding values. This provides for a more accurate gradient calculation and gives us better accuracy in both the support vector machine. All normalization in software is done in the Histogram of Orientated Gradients after the image has been properly binned.

3.1.4 Matlab Process Flow Compilation Diagram

Matlab Process Flow Diagram

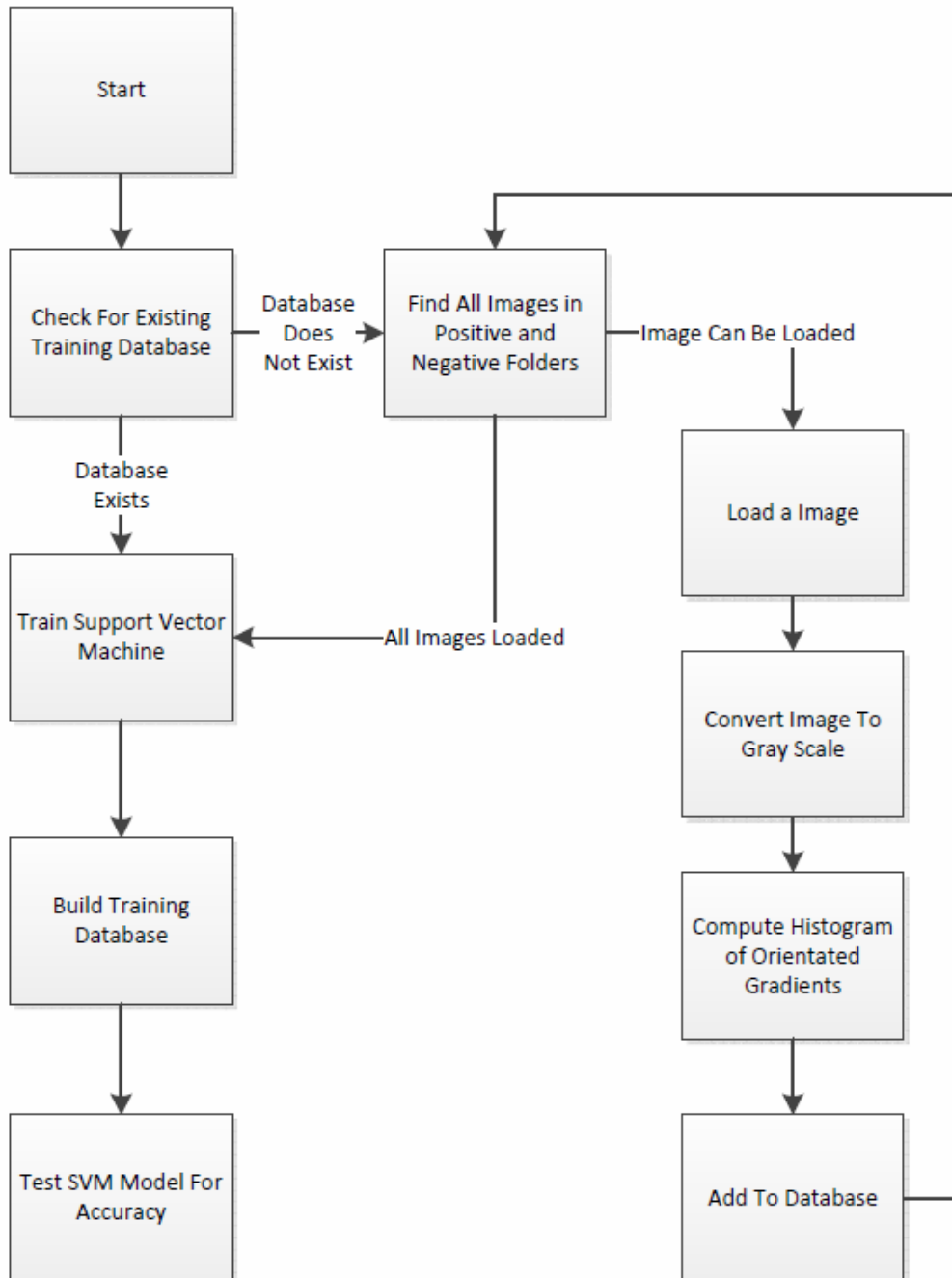


Figure 4: Matlab Flow Diagram

Note this process can take 10 to 62 hours to complete depending on image database size

3.1.5 Matlab Development Environment

All of our functions for Matlab were developed using the Michigan State University High Performance Computing Center. All of our machine learning algorithms, such as Support Vector Machine and Histogram of Orientated Gradients, take several hours to run. We are allowed up to 7 days of runtime per process to create our SVM Model. The Simulink environment was used on both laptops and desktop computers. This compatibility was convenient for our design group and another reason for using Matlab.

3.2 Hardware Design

Hardware design this semester got off to a slow start. The materials we were provided with did not contain a license for Xilinx software, and we spent a lot of time just getting the software required to design the hardware working. Once we had working software, we were able to view the project's previous team's work. File this team left is a modified version of a Xilinx sample Simulink project. It adds the green edge detection block to the project seen in figure 1.

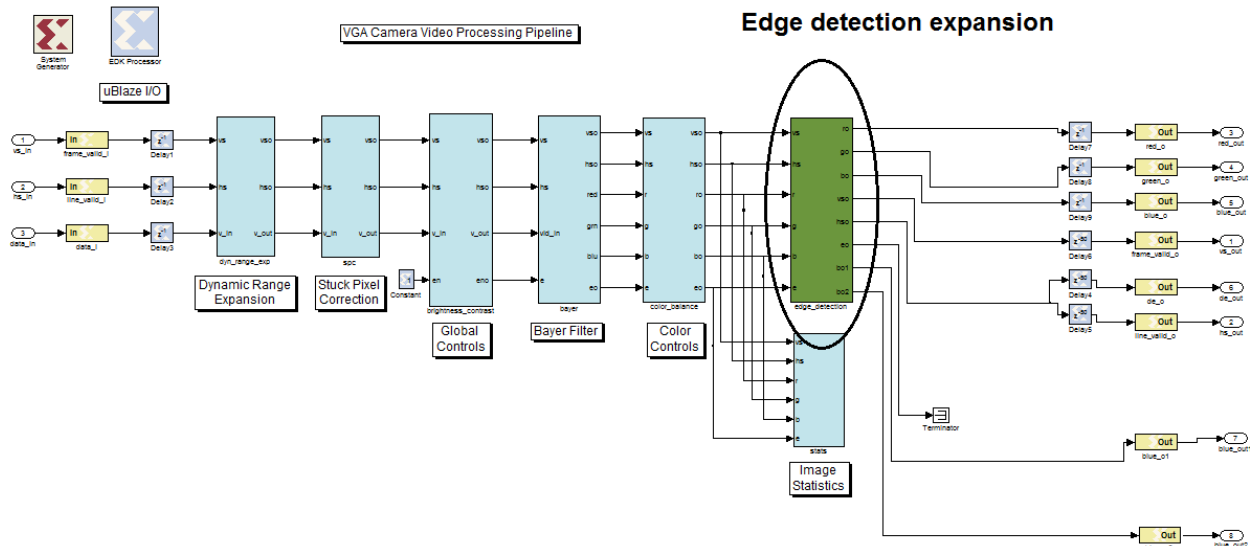


Figure 5: VGA Camera Video Processing Pipeline

The previous team's project performed edge detection based on sobel filters. The block they added can be seen in figure 2 below. It has three main sections: the control signal delay seen in yellow, a section of 5x5 filters that are not used at the bottom, and their x and y detection pipeline in the center.

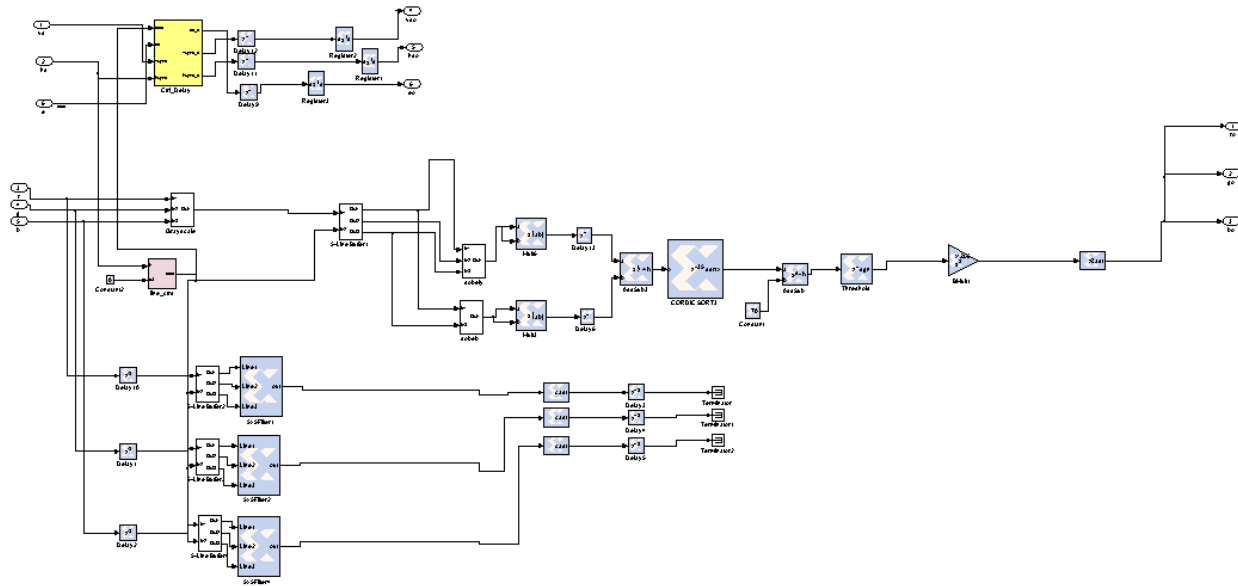


Figure 6: Edge Detection Project From Previous Team

Data is received from the camera as a stream of pixels. A grayscale conversion makes the image easier to work with by consolidating the color signals. The grayscale intensity values travel into line buffer which outputs pixels neighboring lines by introducing some timed delays in the signal. Most image processing algorithms require several pixel data from neighboring regions to perform operations on. The neighboring pixel values then travel into sobel edge detectors, which perform some mathematical operations to compute an edge value in the x or y direction. These edge values are then combined using a simple vector magnitude calculation, and compared to a threshold value to see if the pixel corresponding to the values is an edge pixel.

Once we understood the pipeline, we were able to begin making changes. The first changes we made were to remove the section of 5x5 filters near the bottom of figure 1 to reduce the complexity of the design – they didn't do anything. Once we had tested the project, we broke our much more ambitious possible functionality into components which could be assigned to team members. We quickly went through several design iterations as we gained an understanding of Simulink, and the capabilities of the Xilinx block library, until finally arriving at an abstract design seen in figures 3, 4 and 5.

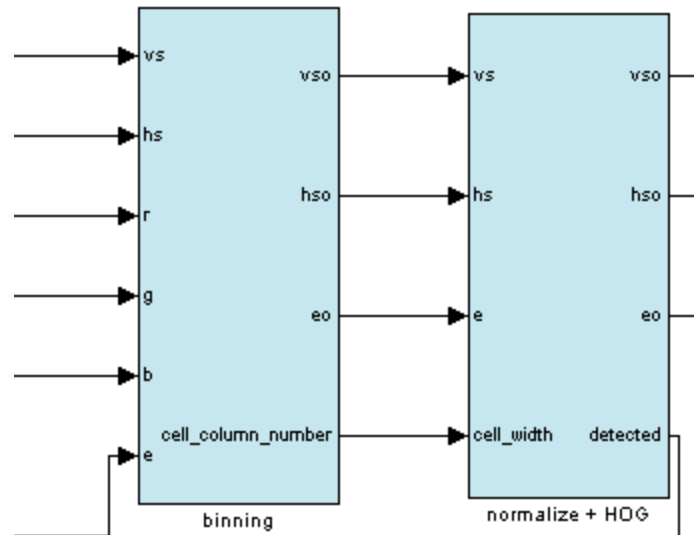


Figure 7: General idea for new blocks

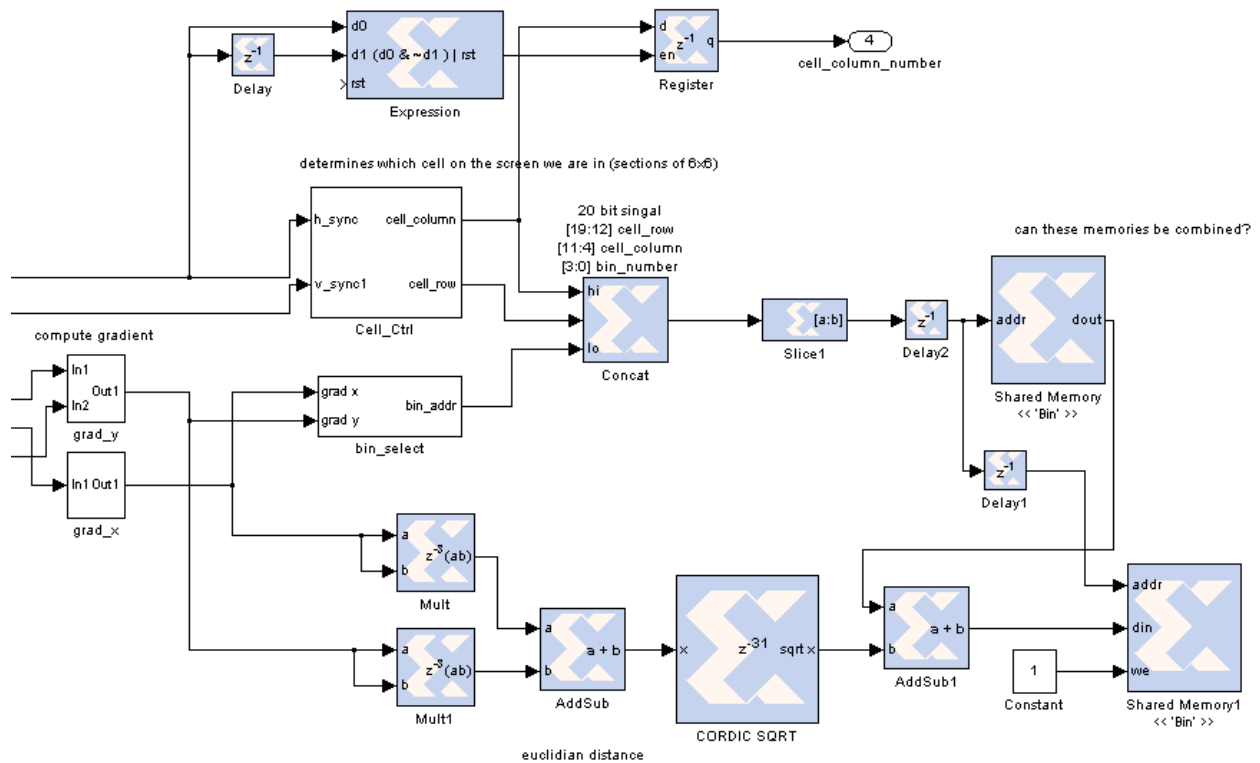


Figure 8: Modified Pipeline to Perform Spatial Binning Instead of Edge Detection

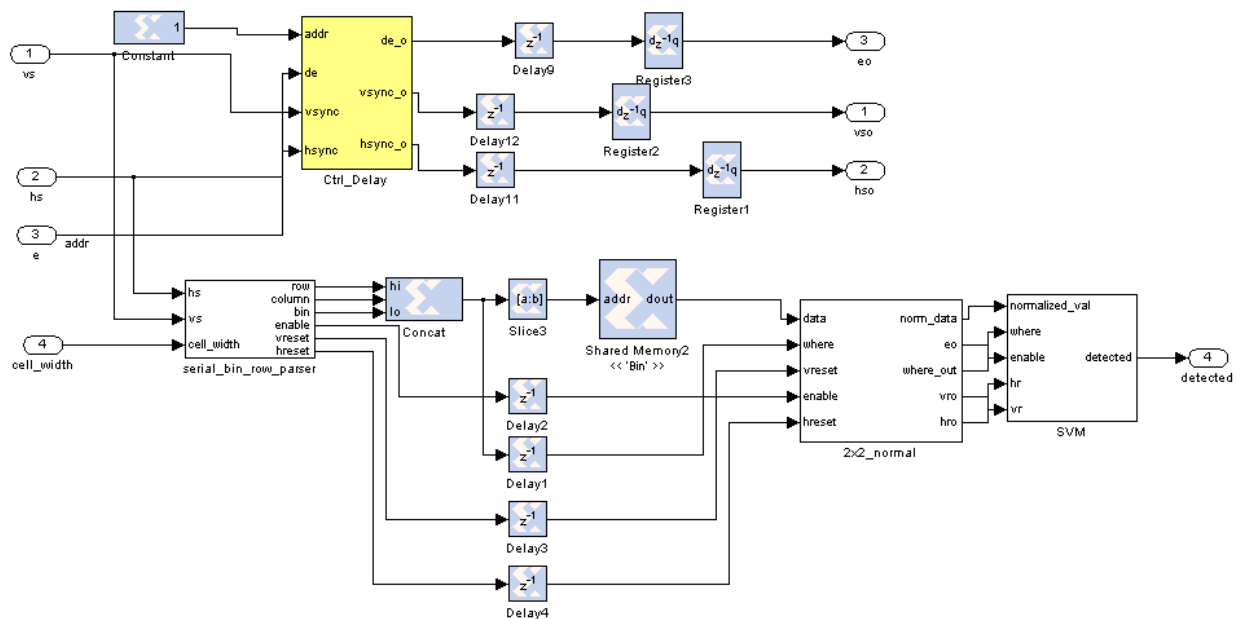


Figure 9: Pipeline to Normalize Histograms and Perform Object Detection/Recognition Using SVM Classification

Work began on the objects shown in the design, and continued up until a point where we wanted to test the design on the development board to make sure we were headed in the right direction. The objective was to generate white noise from the blocks we had filled in so far. During this process it was discovered that the Xilinx shared memory blocks support only two ports per block of shared memory. It was also discovered that none of our group members had been paying attention to the stringent casting and bit width constraints present in the Xilinx block library. We moved on to our second design iteration consisting of only a single top-level block, like the team the previous semester had done, dropping normalization and SVM classification functionality in the process.

We had to come up with a new memory system to replace our naive conception of the Shared Memory block. The resulting system was complex enough to warrant its own subsystem block, and we generate many new signals to control this structure. This left our project very messy, although functionally very similar to the final version of our project.

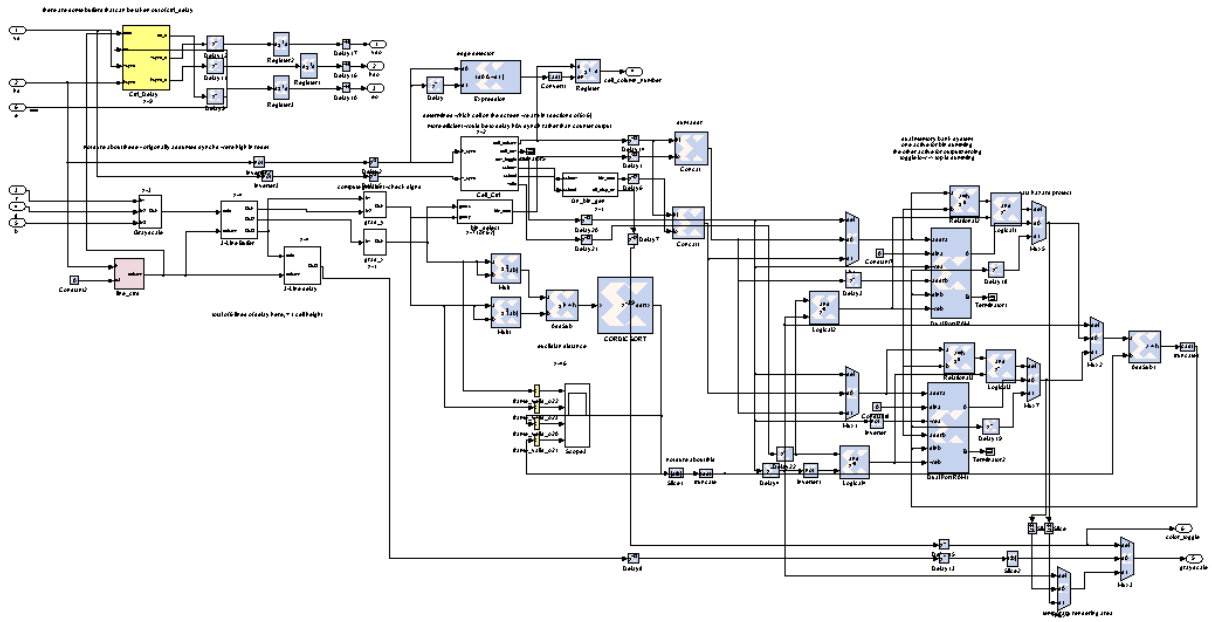


Figure 10: Image of Binning Block Mid-Project

The first part of our pipeline is very similar to that of the previous team, and has all of the same basic components: gray scale conversion, line-buffers, spatial derivatives, and a control signal delay line. Our gradient blocks are merely simpler versions of the previous team's sobel filters, and our magnitude function is virtually identical to theirs – two multipliers, an adder, and a square root function. The second half of our pipeline is where we have added new functionality.

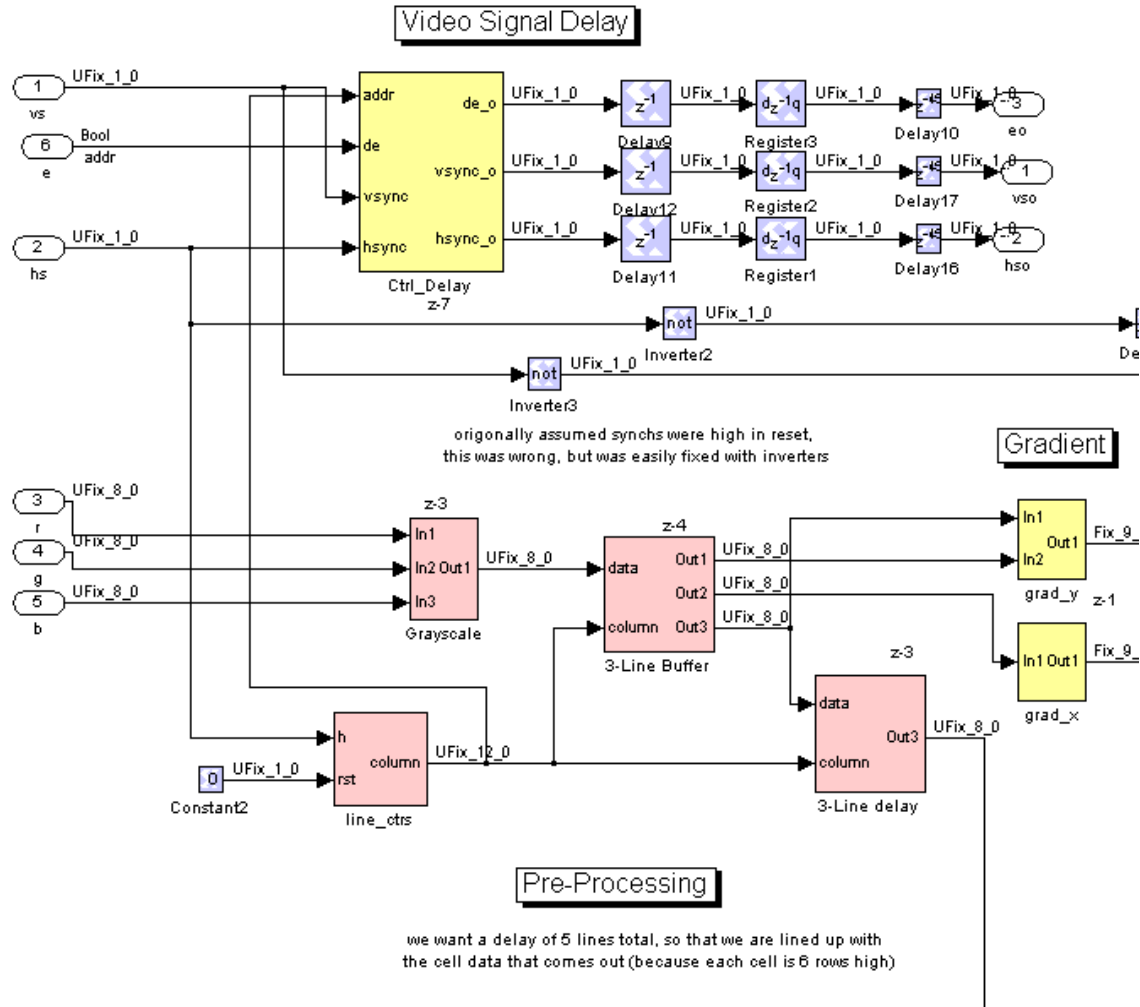


Figure 11: First Part of Project Pipeline - Compare to Previous Team's

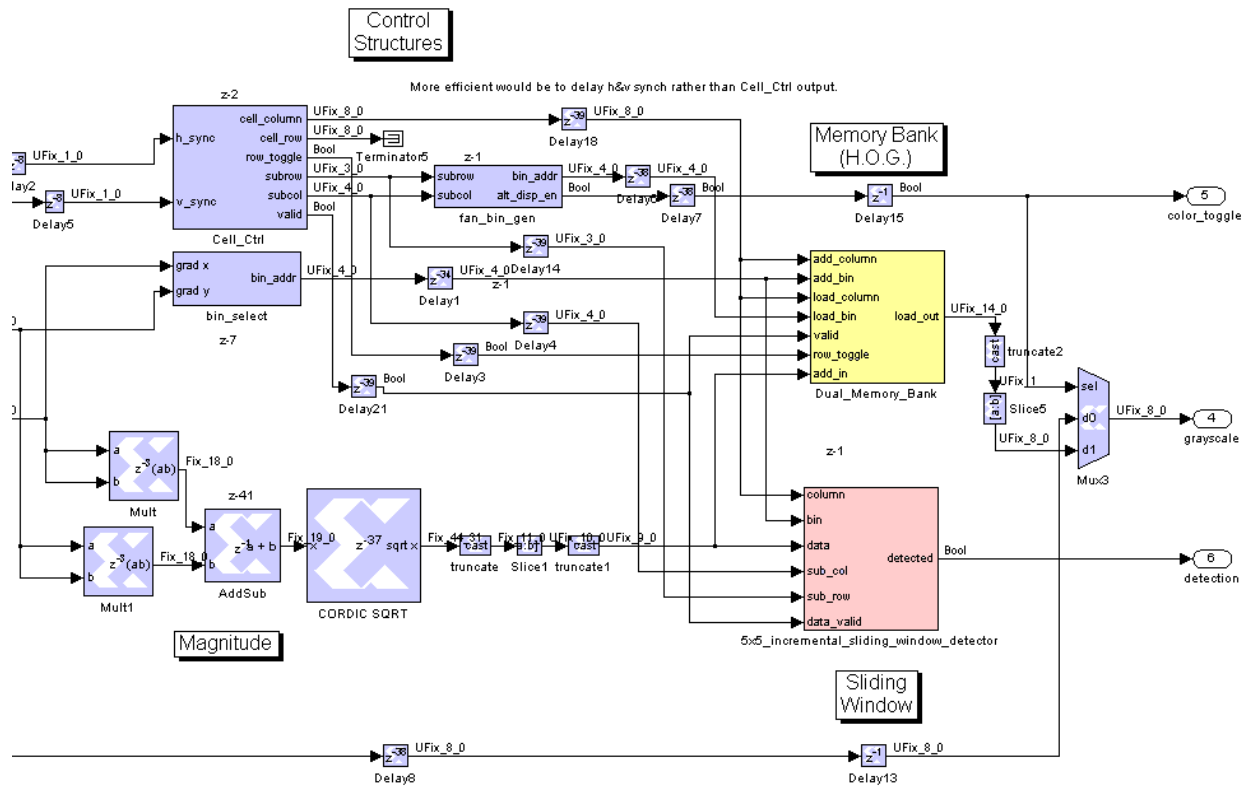


Figure 12: Second Half of Project Pipeline, Demonstrating New Functionality

3.2.1 Histogram Orientated Gradients

3.2.1.1 Cell Control Signals

The first step in for our HOG implementation is to determine which Cell the current pixel in the image stream belongs to. This is done with the Cell_Ctrl block. We went through several iterations of this block with several different signals until finally deciding upon the following.

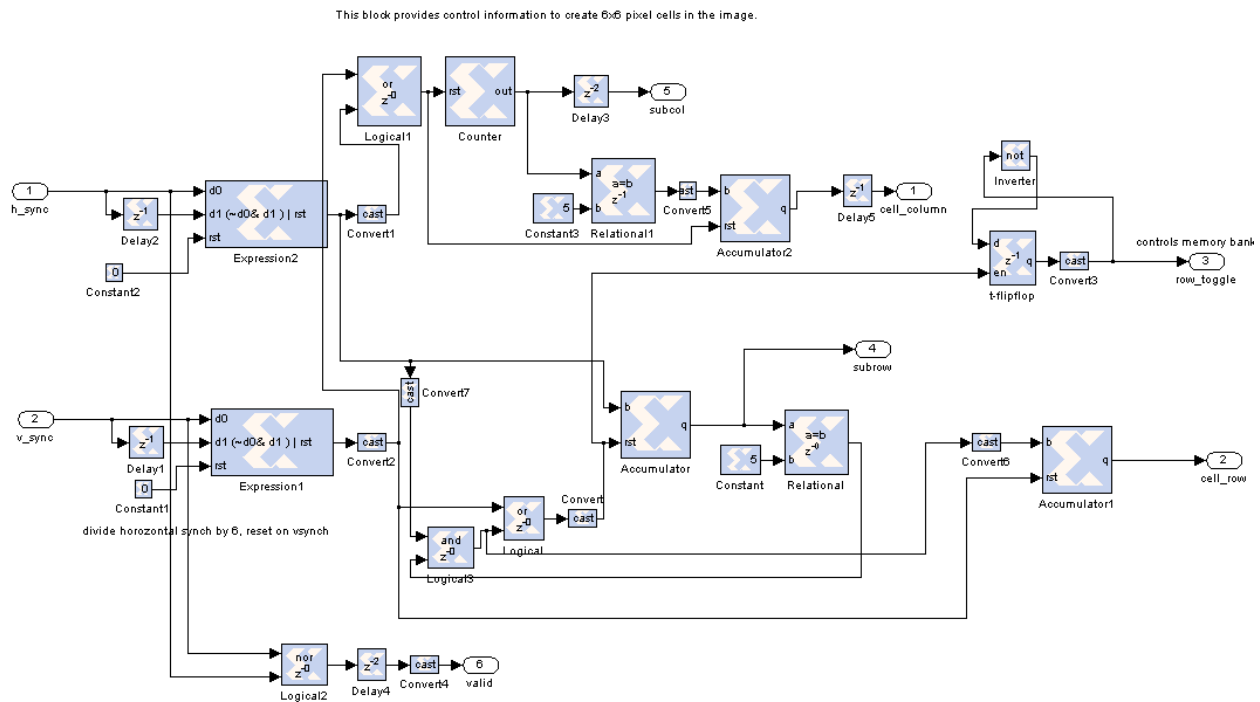


Figure 13: Cell Control Logic Block

This block detects the falling edges of the (inverted) horizontal and vertical synch video control signals in order to reset counters and accumulators, in order to segment the data stream into six pixels by six pixels. The block provides two 8 bit values representing row and column cell indexes into the image. These are accompanied by sub-row and sub-column index data which index individual pixels within a cell and are represented as three bit values ranging from zero to five which. There is a valid bit which indicates whether the values coming out of the block are to be used – if the image is in the process of a horizontal or vertical synch, it may be harmful for other blocks to perform operations on cells that are not part of the image. Finally there is a row toggle Boolean signal that is generated from a T flip-flop which is triggered each time the row value changes, which is used for memory management.

3.2.1.2 Bin Selection

The next step in orientation binning is selecting the appropriate orientation bin within a cell for the current pixel. This can be done trivially using simple trigonometry, but we have implemented a system where no trigonometry functions evaluated after compilation. This reduces the hardware usage of the block, and its latency.

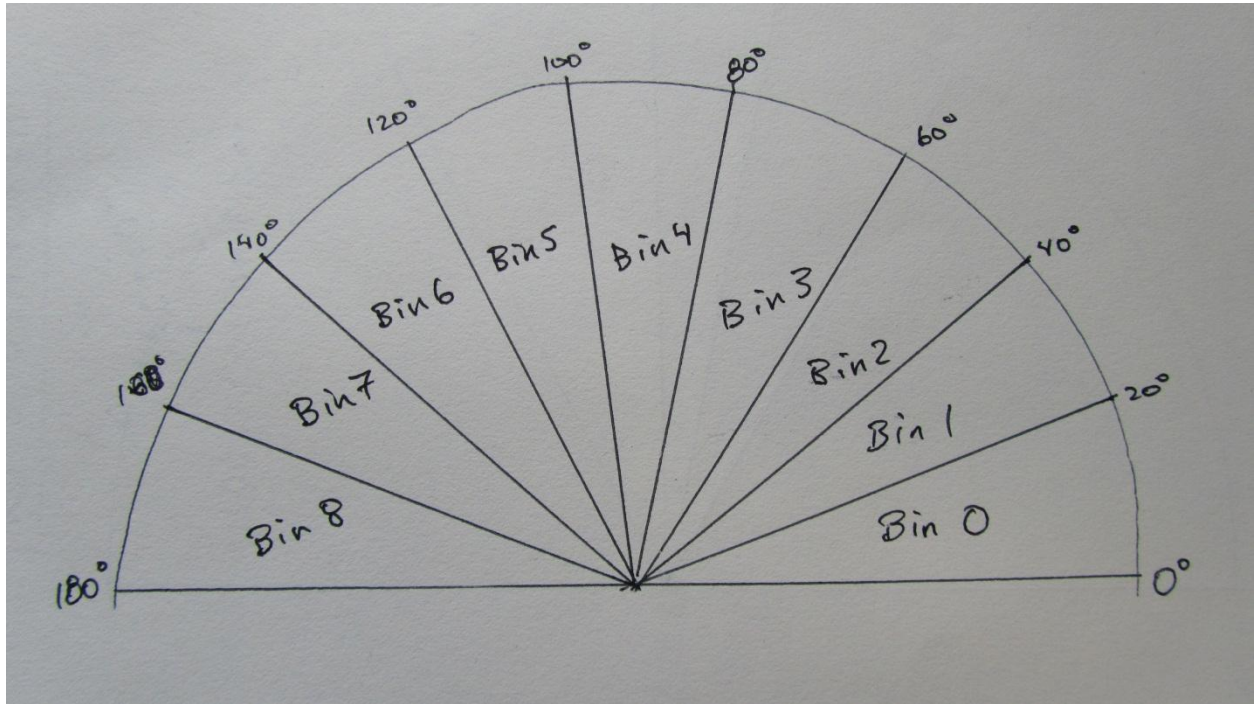


Figure 14: Image of Bin Orientation Utilization

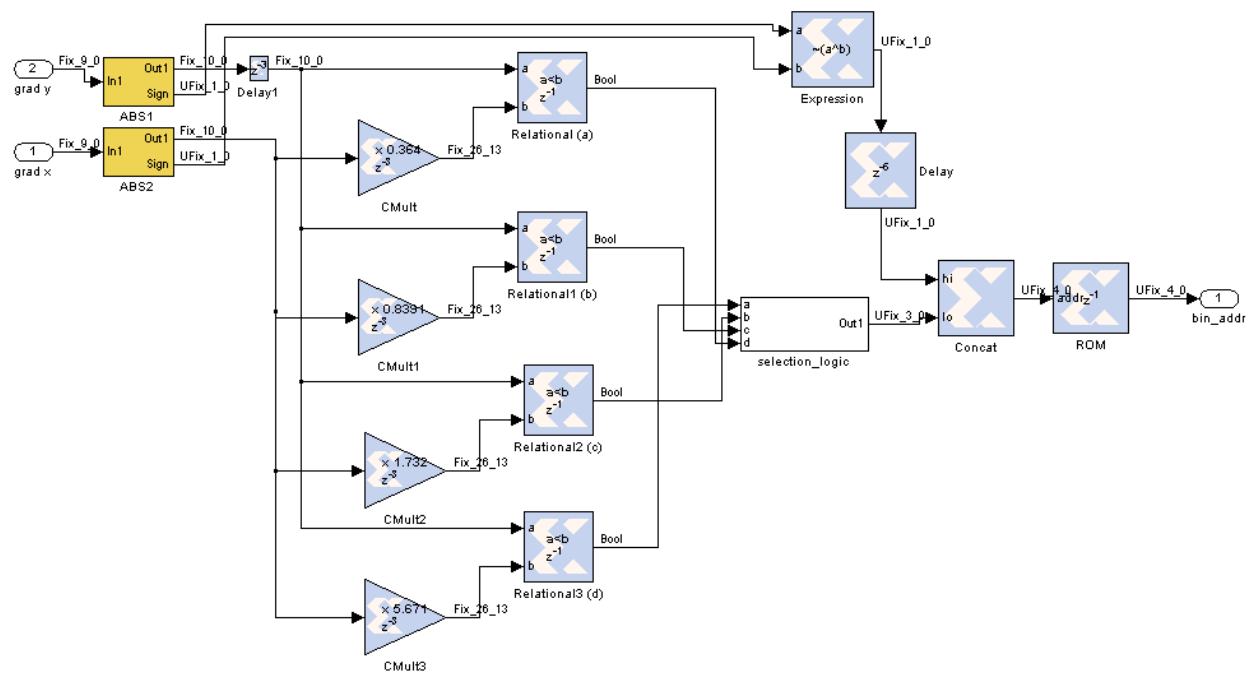


Figure 15: Bin Selection Block

The block projects the x and y gradient values into the first quadrant, and records their sign bits. It then compares y them to several constantly scaled versions of x to generate a unique bit pattern for each bin. This bit pattern is reduced in length by the selection logic, and then recombined with the exclusive or of the sign bits to form an address that indexes into a lookup table of bin values. There may be more efficient ways of doing this than a lookup table, but it is in parallel with a square root function, which is very slow, and so we chose to save our efforts rather than refine the design.

3.2.1.3 Memory Bank

Two things remain to be done once the Cell, Bin, and Magnitude are known. The Magnitudes must be added together for each bin, and the values must be made available for later use. The Dual Memory Bank block in our design makes both of these things Possible.

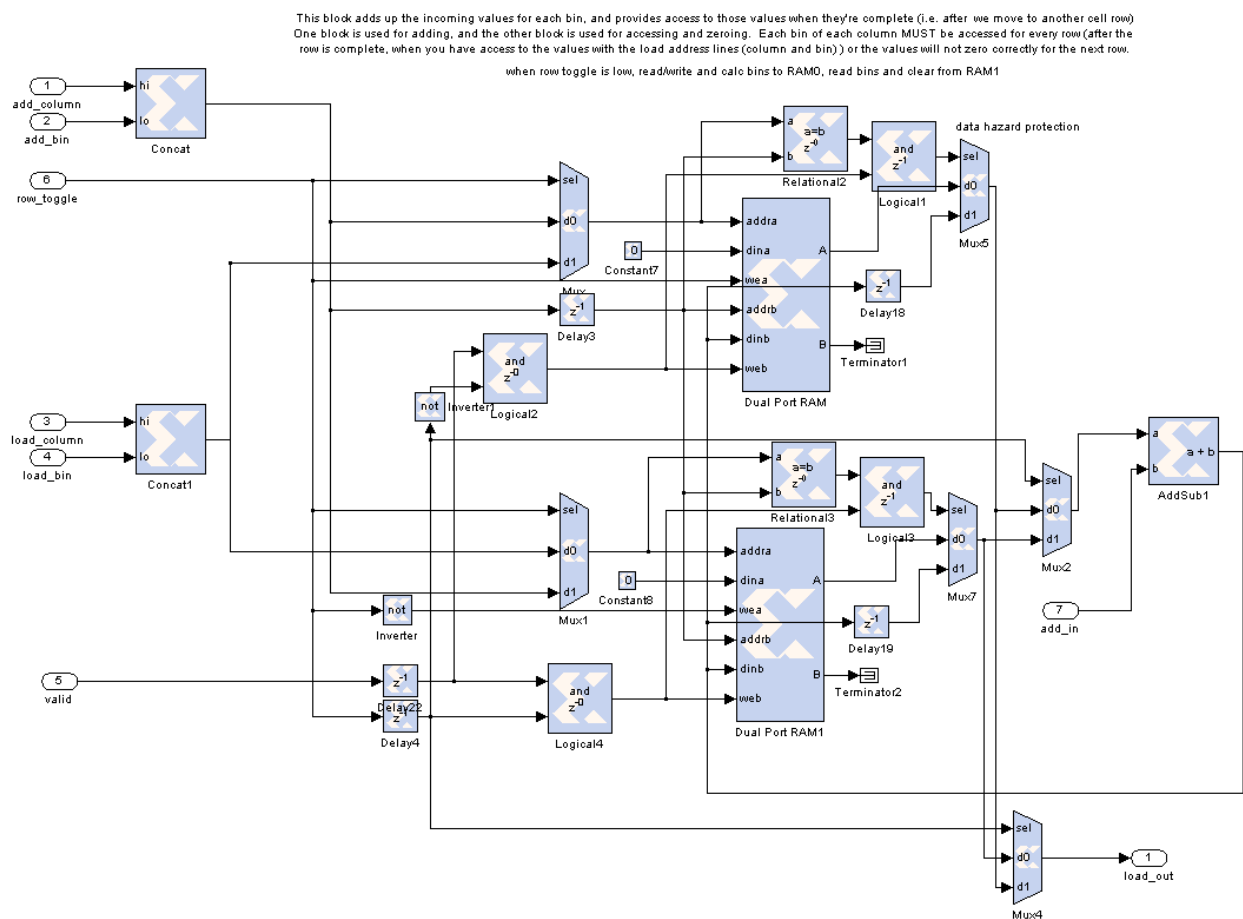


Figure 16: Switched Memory Banks with Dual Port Ram for Summing Magnitudes

At any given time, four operations need to be performed simultaneously in the memory bank. A stored but incomplete bin magnitude value is being loaded for addition to an incoming magnitude; a new sum is being stored back to the memory where it was loaded from; a completed bin magnitude value is being



loaded for use farther down the pipeline; and a completed value is being erased from memory so that the memory can be reused. Our team could think of no way to do this using a single block of RAM, so we decided upon this switched model.

Cell column number and bin number are concatenated to form an address. While one block sums the incoming data, the other makes its stored data available via another address line. The values stored in the accessible block are erased as they are read. Once the row changes, the values in the summing bank are ready, and the blocks switch roles.

The pipelined nature of this structure creates a potential for data hazards, and so we have introduced data hazard protection to both memory banks. This could have been solved with a write-before-read storage order, but the hardware FPGA structures are not optimized for this operation, and this would leave us with no way to erase values as we read them, or at least no easy way we could think of.

3.2.2 Sliding Window

The sliding window block was the last high level block we added to our design. We added it after we had successfully simulated our replacement to the shared memory block. We had successfully run a buggy version of our project on the board, and discovered that we were only using 10% of the area available on the board. We had been worried about space all semester, and still hadn't done any object detecting.

Worries about space were in part why we had decided to drop work on the SVM classifier – we had calculated that it would require at least 105 multipliers and adders each 16 bits wide, 15 RAM blocks each with a depth of 256 by 16, and several thousand bytes readily accessible storage for the support vectors. We knew we didn't have that much space, so we decided to create a smaller and smaller sliding window detector using specially calculated values until we found something that would fill up the rest of our available space. This happened to be a 5x5 sliding window.

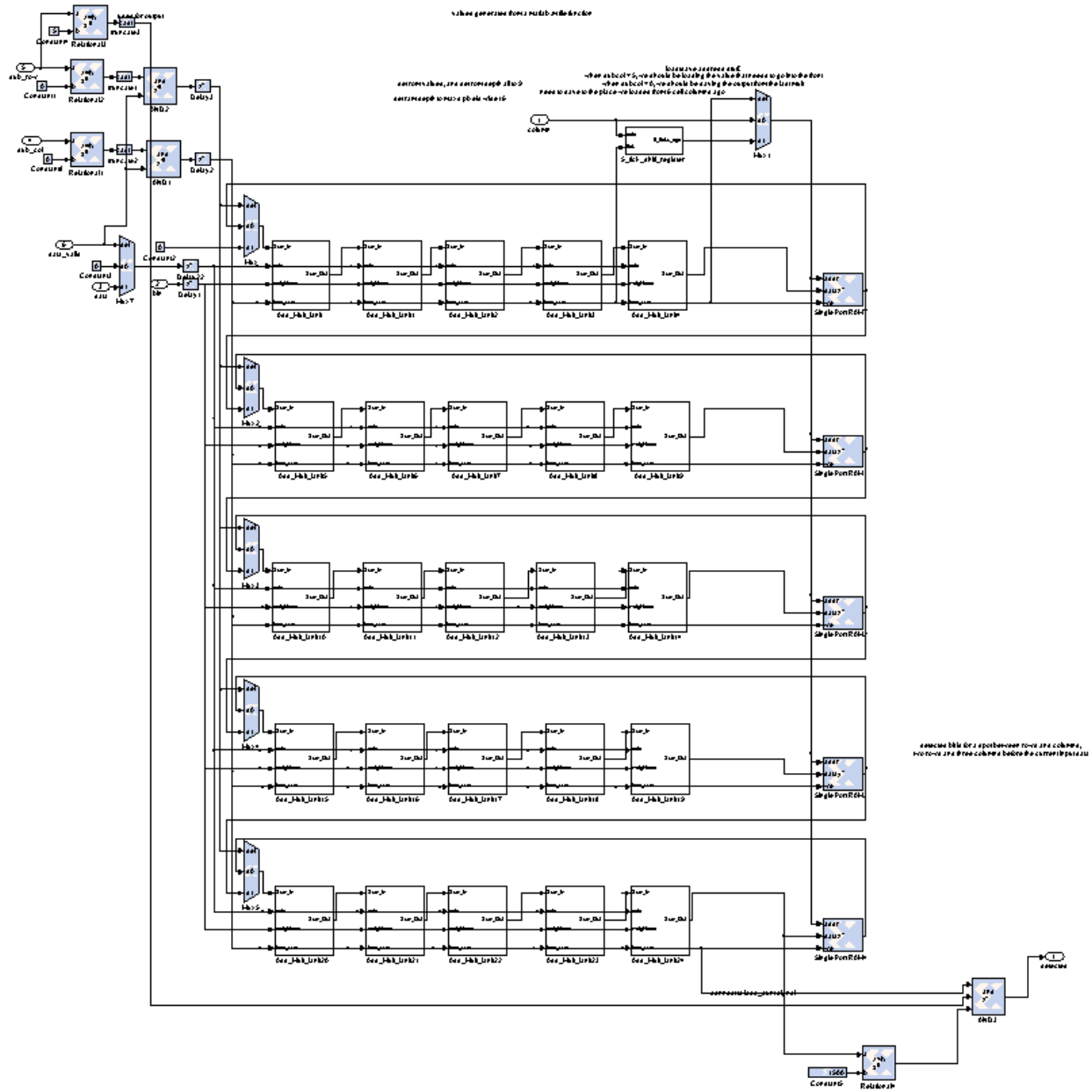


Figure 17: 5x5 Sliding Window Detector

Each time a magnitude and Bin are calculated, these values are passed to the sliding window detector, and sent to each one of the multiplier-adder elements. Each element has its own table of signed weights to scale incoming magnitudes of each bin. Each adder-multiplier element represents a cell in a detection window. When the cells are given a signal, they each pass their values on to the next cell in the detection window. In an image full of 5x5 detection windows, each pixel will contribute to 25 different detection windows. These calculations must be done as the pixel or magnitude arrives. This is why there are 25 processing elements. A value comes off the end of the pipeline; it is compared to a

threshold value. If it is greater than this threshold value, an object is detected. This amounts to a simplistic SVM classifier.

Although the sliding window is currently in parallel with the histogram calculation, it would be trivial to move the sliding window to another point in the pipeline after binning (and/ or normalization) with a simple change of control signals.

The Sliding Window Detector was implemented as a last resort effort to do object detection. Due to time constraints we had to manually calibrate our threshold values for detecting objects. If we had more time and more research we would write an algorithm to compute these values for us and achieve better performance.

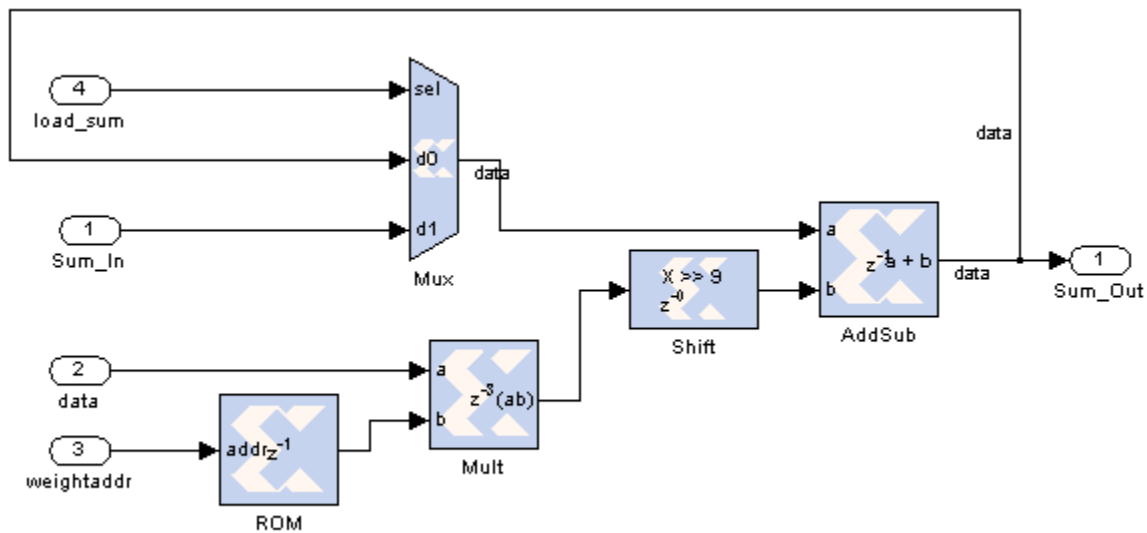


Figure 18: Processing Element in the Sliding Window Detector

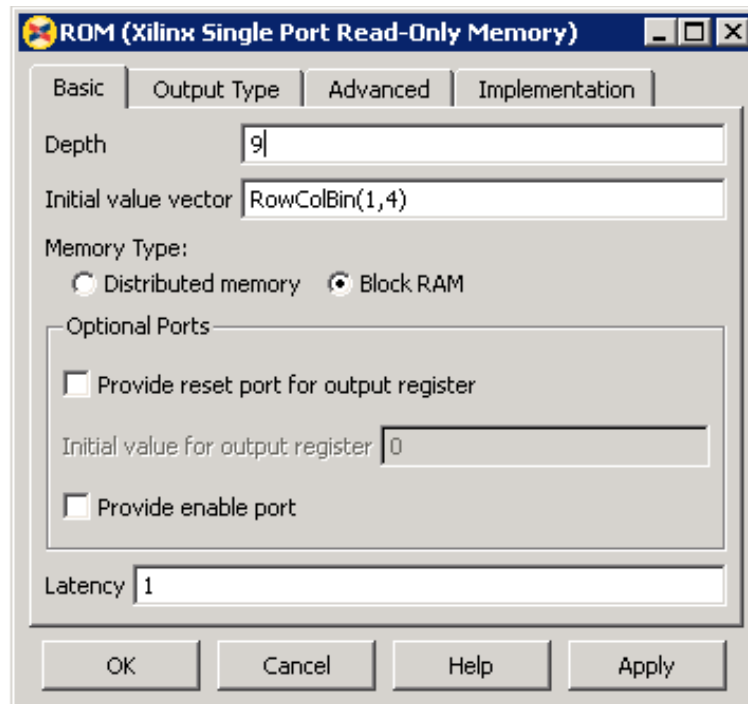


Figure 19: Settings for Rom in Each Processing Element

```
function [ value ] = RowColBin( row, col)
%ROWCOLBIN generates the ROM values for the Sliding Window Detector
%   Generates an array of 9 values between -127 and 127 to fit in a signed
%   8 bit value.

%   Values are based on the orthogornality of the cell's bin
%   direction in relation to a vector from the center to the center of the
%   cell.  Calibrated to 5x5 bins with the center at the center of the
%   window.  Valid arguments are integers between 0 and 5 inclusive.

%generates average value of (-0.27324 * 127), which we then correct for
value = [0:8] %use this line for matrix
for bin = 0:8
    topoint = [row-2, col-2];
    norm = sqrt(dot(topoint, topoint));
    topoint = topoint/(norm+.00001);
    out = 127.0 * (.27324 + 1-2*abs(dot(topoint,[cos((bin*pi())/9.0 +
(pi())/18.0)), sin((bin * pi())/9.0 + pi()/18.0)])));
    %matrix access
    value(1,bin+1)=out; %use this line for matrix storage
end
end
```

Figure 20: Matlab Code to Generate Rom Values

3.2.3 Normalization

Normalization would have provided an excellent addition to our project. We took it out when we encountered our shared memory problems, and did not have time to put it back in before the end of the semester, although we believe our design to be a very nearly working. In fact, the working sliding window block was based on this block.

The theoretical throughput of this Normalization block is enough to normalize the entire 2x2 cell block, instead of just one cell, allowing redundant normalization of data which has been shown to improve SVM classification.

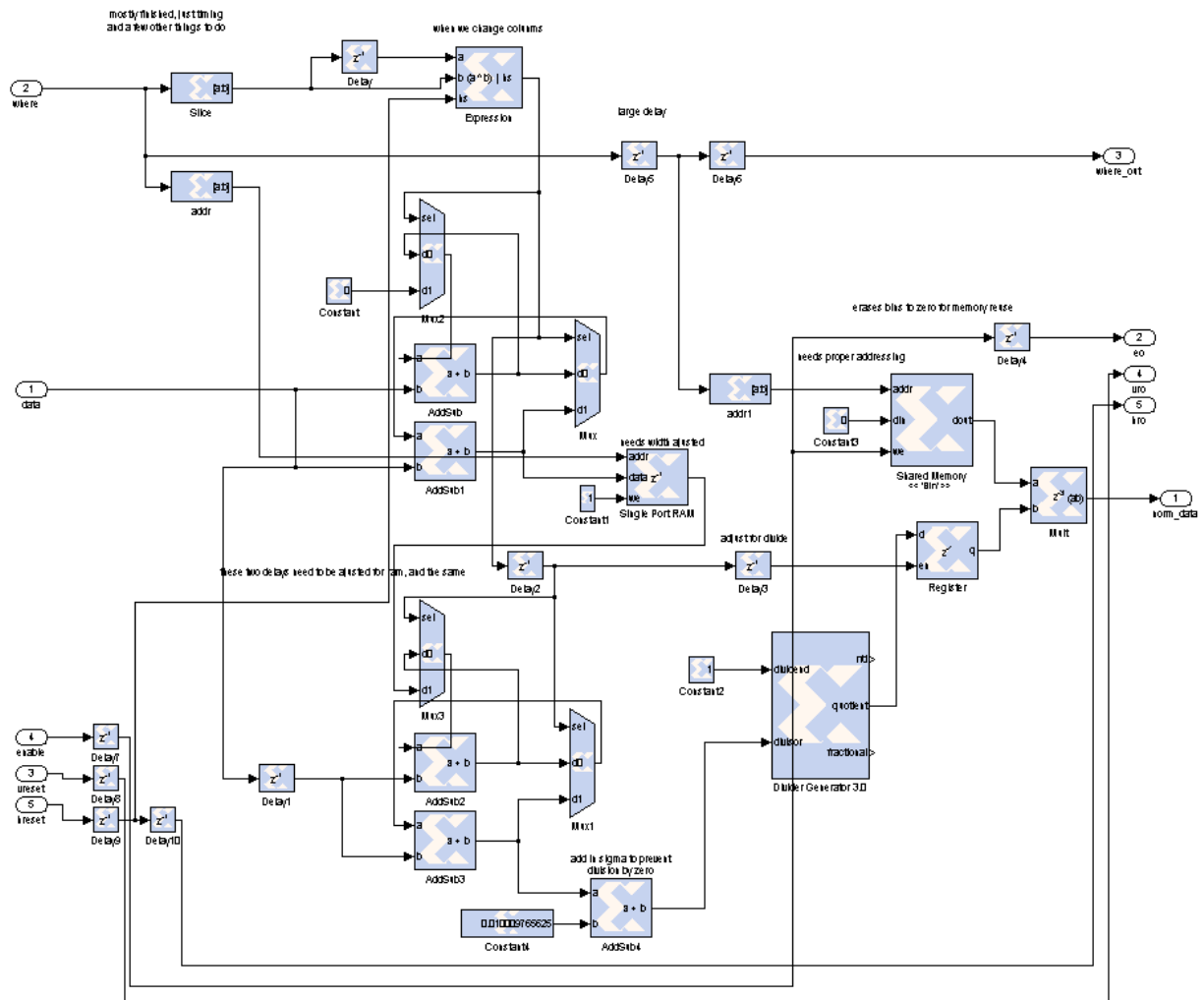


Figure 21: Block for 2x2 Cell Normalization of the Bottom Right Cell

Chapter 4: Test Date with Proof of Functional Design

4.1 Testing Histogram of Oriented Gradient

Testing was performed using the Simulink simulation functionality, and by observing video output from the hardware running. Unit tests were performed in Simulink for the bin selection blocks, and the control logic blocks – these are successful and can be seen below. Basic functionality of these blocks was working in the second week of April, but minor changes had to be made throughout the rest of the project.

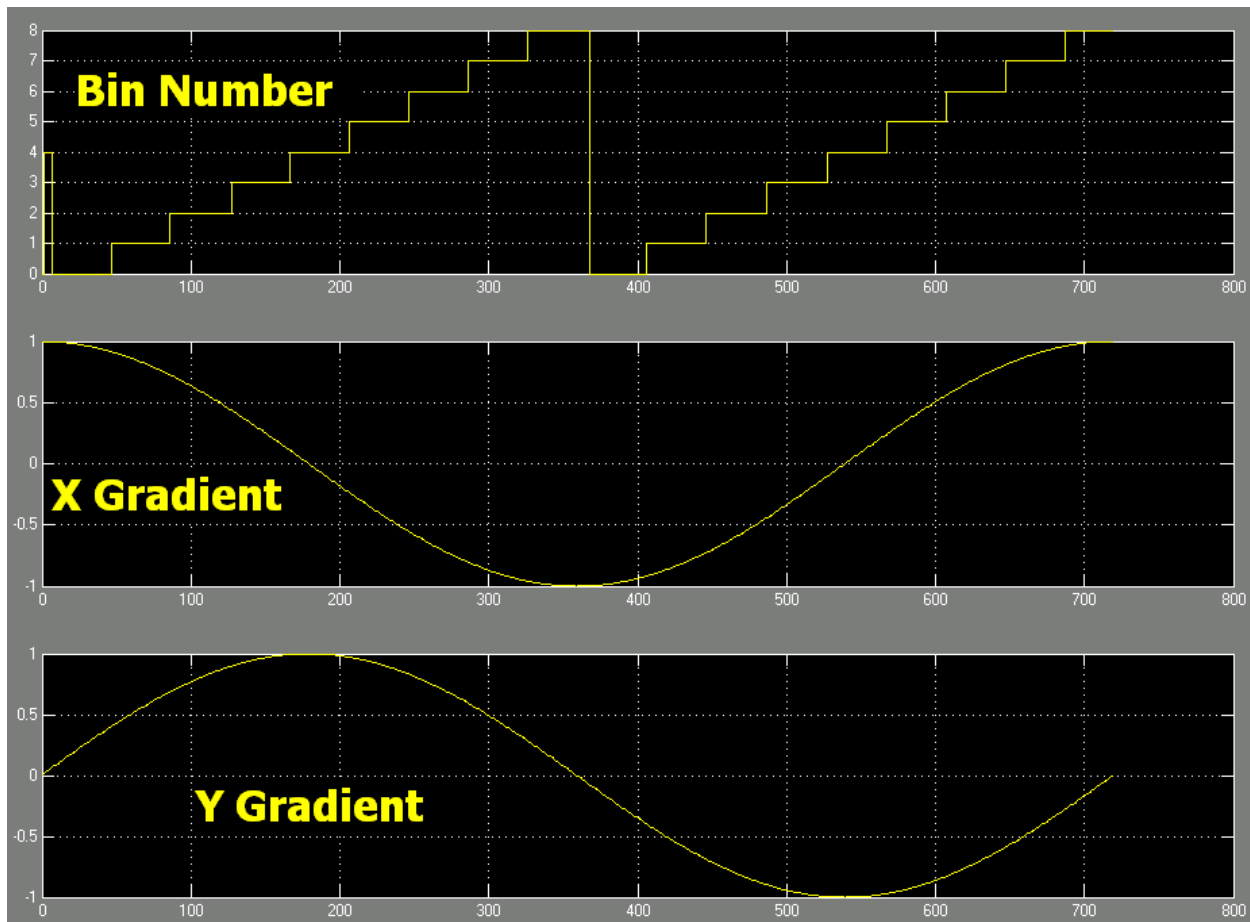


Figure 22: Simulation of Bin Selection Block

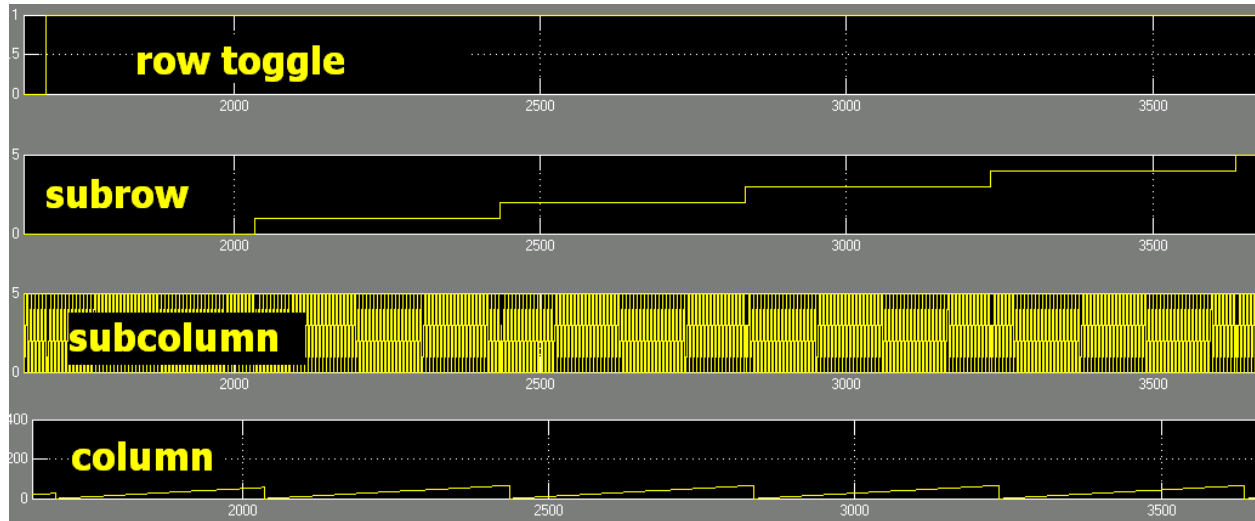


Figure 23: Simulation of Cell Control Block

Other Simulink simulations were used to make sure that the memory slots were correctly resetting, and loading, and to verify appropriate arithmetic precision for all of the operations involving gradient magnitude. One simulation revealed that our sliding window block was drifting due to a statistical anomaly in the bin weights, which was easily corrected in MATLAB. Using MATLAB we were able to verify the new values.

```
>> RowColBin(1,3)

ans =

    16.0141    95.9619   139.5641    54.3572   -17.9024   -68.4991   -91.3302   -83.6419   -46.3617

>> RowColBin(2,3)

ans =

   117.5953    34.7027   -32.8719   -76.9781   -92.2960   -76.9781   -32.8719    34.7027   117.5953
```

Figure 24: Checking Orientation Bin Weights for Sliding Window in MATLAB

It is very difficult to debug image data by looking at a scope, and so the remainder of testing was done by analyzing video data. Timing errors in vertical and horizontal synchronization signals both become immediately apparent by skewing of the image. Bin saturation and noise are readily visible. We initially

had issues with bit significance, and overflows, but were able to correct these over time. Many of these errors do not make sense as static images.



Figure 25: First working Video - Gradient Magnitude Only

Some special control structures were used to help display the computed data to the video display. By accessing the data stored in the memory bank in the correct order, we were able to spread representations of the bin values over the corresponding locations relative to the center of each cell. Green was used to indicate a strong gradient, and red was used to indicate a weak gradient.

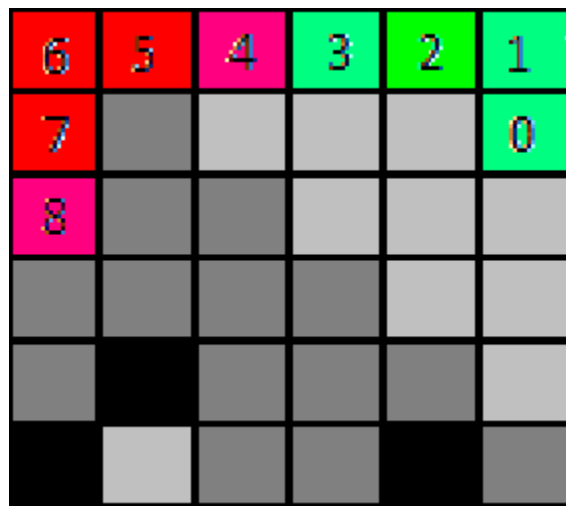


Figure 26: Sample Orientation Bin Output for a Cell in Video

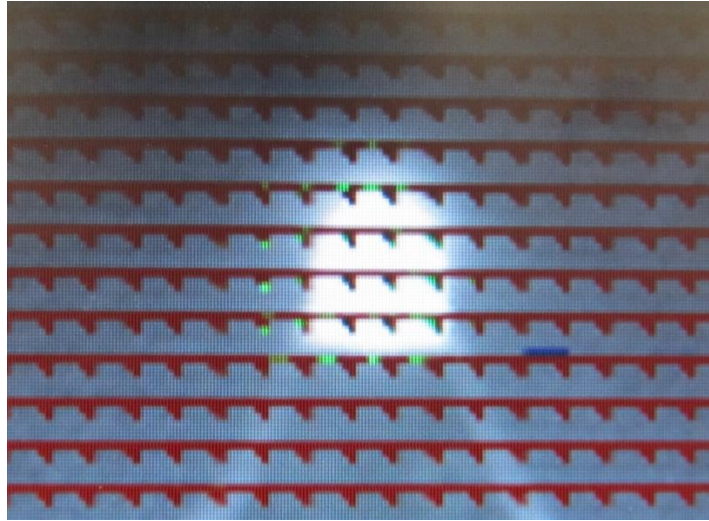


Figure 27: Orientation Binning Demonstrated

Object detection was output in blue at the bottom of a cell. However, we did not have the resources to line the object detection up with the center of the detection windows, so the detection indicator is present several rows below and to the right of the detected object.

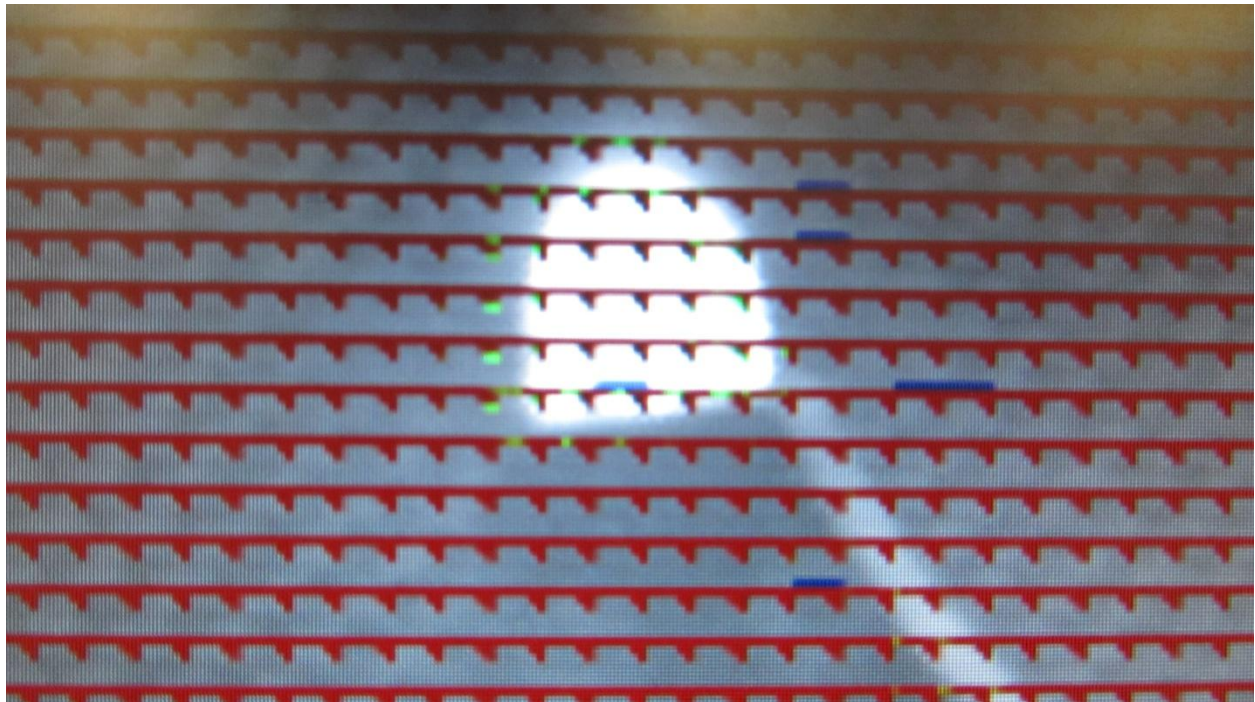


Figure 28: Detection of an LED Demonstrated, Showing Offset

Without normalization, detection is somewhat limited to high contrast objects, but does work impressively well on them. Object detection still works on less contrasted scenes, but not as impressively, as seen in the figures on the following page.

Working results were only achieved in the last few days. As can be seen in these images, both rudimentary objet detection and HOG calculation work, as of April 26.



Figure 29: Demonstration of Working Histogram of Oriented Gradients and Object Detection



Figure 30: Object Detection Working In a Lower Contrast Scene

4.2 Testing Support Vector Machine

The Support Vector Machine is a model for a pedestrian, and not pedestrian. The easiest way for our design team to test the accuracy of our generated SVM is to find another dataset to run through our SVM. The INRIA dataset for pedestrians actually comes with a dataset for testing.

Testing starts the same way that training does, by creating a test database containing HoG descriptors for all the images we will be testing. We will keep track of

- The Number of images we expect to be pedestrians
- The number of images we expect to be not pedestrians
- The number of images we expected to be pedestrians but were detected as not pedestrians
- The number of images that were detected as pedestrians but are not.

Saving these values will allow us to figure out what percentage of pedestrians we detect, as well as the percentage of false positive results we will get.

Below is a sample analysis of our pedestrian dataset. Note that the returned database containing results is suppressed in this image.

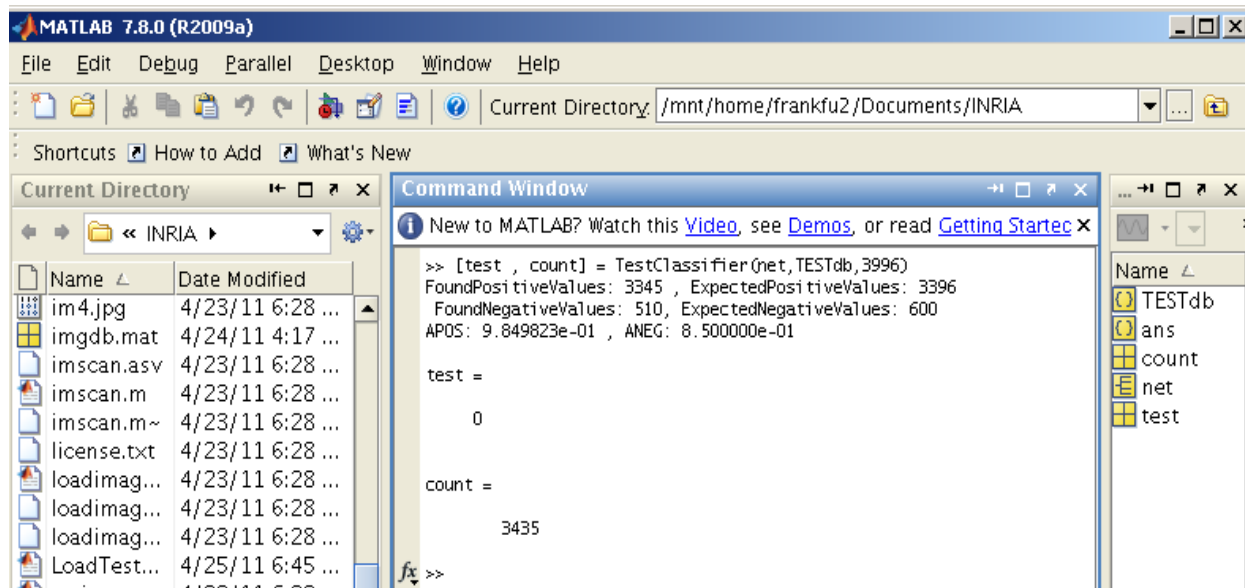


Figure 31: MATLAB Pedestrian Dataset Analysis

The results of this test indicate that we successfully detected 98.4% of the pedestrian results but we only detected 85% of negative values. This means that we should adjust our training dataset to include many more negative images to reduce the number of false positives generated.

Attached in the appendix 3.2 is the TestClassifier function used above.

Chapter 5: Final Cost, Schedule, Summary and Conclusion

5.1 Final Cost

Due to the fact that this is continues project from previous semester. The Spartan 3A FPGA board provided by Xilinx has been reused as the main design platform, along with all the parts and accessories. For the software portion, Matlab and Simulink are supplied by Michigan State University engineering department. We had some problems to get the license for Xilinx ISE design suite but with some trail licenses, the team was able to complete the project without buying the software.

Over the entire semester, team 4 did not spend anything from the design budget, thanks to the generous donation from our sponsor and the help of the engineering department.

5.2 Schedule

- The ECE 480 design team has made a gantt chart from the beginning of the semester. Including major role and timeline for each group member and we have been follow the schedule closely and we were able to accomplish most of our task on time.

-Important dates and events

Date	Class Overview
Jan. 10	Team Oral Proposal Presentations
Mar. 2	Individual Application Notes due to facilitators
Apr. 1	Team Technical Lectures
Apr. 6	Team's Design Issues paper due
Apr. 15	Professional self-assessment papers due
Apr. 22	Evaluation Day: Course evaluations (SIRS)
Apr. 25	Final Reports due
Apr. 26	Design Day
Apr. 29	<i>Notebooks due</i>
Apr. 29	Final CD with ALL documentation due
Apr. 29	evaluation of the contributions of your team members
May. 2	evaluation of the contributions of your team members

5.3 Summary

Design Team 4 has laid impressive groundwork for object detection implementation and further image processing algorithm. The team structured their algorithms and designed the system in a manner that could be expanded upon. In the semester, the team successfully created and trained Support Vector Machine with a giant database of pedestrians in Matlab, and created Histogram of Oriented Gradients algorithms and a sliding window detector, which can be placed over the original video feed and displayed on the monitor.

Support Vector Machine is a binary classifier that can be “trained” to identify arbitrary data sets. With a set of input data, it is able to distinguish between two different classes and compare these classes against input data. In developing the Support Vector Machine model, the team used the High Performance Computing Center (HPCC) to manage massive amounts of memory and take advantage of multi-threaded applications to accelerate the development and creating the Pedestrian model.

In Histogram of Oriented Gradients algorithms, the team was able to convert the input source into gray scale and compute the gradients of the objects with 6 by 6 cells and each cell contains 9 bins which are used for storing and computational purposes. On the other hand, the idea of the algorithm is that the team first calculated horizontal and vertical gradients for each pixel in a frame of video source. These bins are then grouped into “cells” of a particular shape, and have each cell contribute to the histogram of oriented gradients for that cell by “voting” for the orientation which is centered on that cell. Typically, the cell votes are weighted with a function of the position of the individual cell, and the magnitude of the orientation centered on it. With the utilization of sliding window algorithm, the team was able to compute the weight and the magnitude of the cells and detect the object within the cells.

The critical failure of the team is that the team did not realize the limitation of the board, so that the team was not able to fit the trained Support Vector Machine on the board. After creating Histogram of Oriented Gradients algorithms, the design has only occupied ten percent of total memory slices of the board. Nevertheless, when the team decided to further develop the design into next stage with sliding window algorithm, they discovered that the size of the program had already exceeded the total memory slices of the board. Therefore, the team decided to reduce the components of slide window algorithm to a 5 by 5 window instead of 6 by 6. Fortunately, the team was able to implement the Histogram of Oriented Gradients algorithm and sliding window algorithm into the board with a limited memory size. However, the team was out of memory slices when trying to implement the trained Support Vector Machine into the board.

The design team has several recommendations for future teams that inherit this project regarding object detection. First, the team must familiarize themselves with Xilinx’s tools and especially with the capabilities of System Generator. For instance, in the beginning, the team used five shared memory objects referring to the same of shared memory in the design. However, the team discovered that the design can only support two shared memory objects per shared memory block, so it turned out that the team spent twice as much as time to try to redesign and fix the issue with two shared memory objects per shared memory block instead of five. Therefore, the team suggested that the future teams should



be familiar with Xilinx's tools and especially with the capacities of System Generator. By understanding the language level available within System Generator, an advanced algorithm can be easily created with a shortened time period. Second, the future teams must realize the limitation of the hardware before working on any design. If it is possible, the team recommended the future teams to get the latest FPGA board which has faster speed and larger memory size. Since the team had put the best effort to reduce the components of the design to make the program take less memory slices, the team was still unable to fit the trained Support Vector Machine into the board.

5.4 Conclusion

Design Team 04 has had many successes but we did have a small amount of technical difficulties and impossibilities. Our Design Team successfully was able to take an input signal from a video camera, Compute a histogram of orientated gradients. Store and address all the binned values in memory for accessing at a later stage. We also were able to complete the support vector machine to do the entire classification of this algorithm. We however had trouble getting it to fit on the board and were getting out of memory errors. Our team also did a good job at presenting but sometimes struggled with meeting deadlines. With only four group members in comparison the six the team from last semester finding time to do documentation, homework's, and then find time to do our project was not easy.

Overall we feel that our team accomplished all the goals of ECE480, collaborated as a group, created a product, and presented technical information as well as documented it.

Appendix 1: Technical Roles, Responsibilities, and Work Accomplished

A1.1 Meng-Chiao, Lee – Web Development



Meng-Chiao Lee's technical portion of the design project for the semester has involved to Xilinx resources research, hardware implementation, part of pipeline design, and debugging. In the beginning of the semester, Lee focused on doing the research on Xilinx resources and studying the resources specially hardware implementation from the previous team. During the research, Lee found out that the utilization of System Generator in MATLAB/Simulink and Xilinx Studio Platform was a key process in the hardware implementation. The reason is that System Generator enables the use of MATLAB/Simulink to design a model for

programming FPGA board. By using a Xilinx blockset, all of the downstream FPGA implementation and necessary files would be generated to program the FPGA automatically. Therefore, Lee spent first couple weeks of the semester to understand and run the existing tutorials to get familiar with System Generator. Very quickly, Lee was responsible for the team's first demo and second demo of implementing the edge detection model from last semester due to the experience on hardware implementation.

After doing the research and understanding the concept of System Generator and basic flow of hardware implementation, Lee started to focus on pipeline design. The design involved with cell control logic and location generator. For cell control logic, Lee designed a function that would divide the frame into certain number of cells base on the size of the frame. Each cell is composited of 36 (6X6) pixels. With the pixels and vertical and horizontal synchronizer as input sources and a combination of different types of logic gates, the function was able to perform a simple logic to count and reset the number of cell every frame. For location generator, Lee designed a function that would create locations (bins) to store the output value of Histogram of Oriented Gradient in the corresponding pixels. In this function, each cell should contain 9 bins, and able to reset the count in proper time.

After finishing the design, Lee started to help team to debug the program. First, the team found out that the design has many issues needed to be fixed, so they spent couple days to try to address them one by one. Once the team fixed each issue, Lee implemented the design into the board and checked the output, and moved to the other issue if the result is what the team expected.

A1.2 John Buether – Manager



I was the system architect for the project. I worked at the Simulink level. I broke up the project into different components, designed the functionality of each component, and laid out the components. I delegated as much of the responsibility for each of the components as I was able to, to Kan and Lee. I did this after the functionality of a component was well defined. Unfortunately, I could not explain the functionality of most of the design components to them, and so ended up laying out those components as well as designing them. I was also responsible for testing the project, making sure all the signals and timing were correct, and getting it to finally compile onto the hardware, although

I did not actually build the project – I cooperated with Lee while doing this, and he would build the project and put it onto the board.

Up until the second week of April, my job was to read papers, think, make drawings in my notebook, and communicate with everyone to make sure their jobs were going well. On April 6, we finally had computers set up that we could work on – there had been a lot of issues with this, so I'm told by Josh, and the others. I was given login information to a virtual machine running on Josh's home computer with Matlab, Simulink, the Xilinx ISE Design Suite, and the Xilinx libraries for Simulink installed. I was then able to start working performing in my technical role. Because my role can be much more easily understood while looking at the project, screenshots of the project in Simulink will be provided as an appendix.

I started with the previous group's Simulink file, and worked from there. My first major breakthrough was when I discovered that the majority of their design wasn't being used, and the signals were being dumped to ground. I then made a copy of their project, gutted everything I didn't need, and started laying out blocks for calculating the gradient, calculating the bin from the gradient, doing normalization, implementing the SVM, deciding how the output was going to be displayed. I committed the first design iteration with most of the blocks not completely filled and only placeholder delays on April 6. I then delegated parts of the bin calculation, all of cell control, and all of a no-longer present block for serial bin iteration (post binning) which I referred to as the location generator, to Kan and Lee. I continued to work on normalization, which I decided would be easiest as 2x2 cell L1 normalization, the SVM, calculating the gradient, and the part of the bin calculation that that bypasses the need for evaluation of trigonometry functions. This first iteration used 5 shared memory objects referring to the same section of shared memory, and was to evaluate an SVM classification on only one window.

After the logic had been filled in for the most part, I worked with Lee to try to put the first project on the board, knowing full well that if it worked it would only generate noise. During this process I read a lot of

technical manuals, and discovered that you can have only two shared memory objects per shared memory block in a design, so I had to rework a major part of the project. I “forked” the project to a new file so we could refer to the old design, and worked on a new memory system that used two memory banks each consisting of one dual port ram block. I dropped the normalization from the project at this point because it relied heavily on shared memory, and did nothing visually impressive – it was made with regards to our design day presentation, as I was beginning to worry about time. I also dropped SVM evaluation in favor of displaying the bin values, because I wasn’t sure we could get it working in time, and it would not be very functional or impressive without sliding window detection.

After completing these blocks, and correcting errors displayed by system generation, we were able to generate the system for the first time. We discovered our software licenses had expired, and we were unable to put the system onto the board. While waiting for Josh to fix the license problem, I learned how to use Simulink to do unit tests of the bin unit, the cell control, and the magnitude calculator. I estimated the delays that needed to go into the lines to correct timing differences between different parts of the circuit. I had to redo much of the cell control block, because it wasn’t working quite right, and then integrated the changes into the memory unit. At this point we had a license working, and were able to generate a project to display noise. A few corrections to timing provided edge detection. I corrected an error that occurred when I confused row and column lines, and we had a coherent edge detection image.

A few more iterations and I decided that we could probably do sliding window detection on something that wasn’t an SVM – Josh had the SVM calculated, and working on the computer, but we weren’t sure how to translate the algorithm to the board. The SVM did not work as we thought it did – we had expected only 945 values, but there were instead 135 or so vectors each of 945 values. I referred to one of the ideas I had earlier in the semester before we had decided on SVMs for demonstration purposes, and implemented a sliding window system through a series of iterations. I had Josh calculate values for this sliding window classifier. As I write this personal evaluation, I’m still finalizing the design.

In summary, I did the vast majority of the work on the implementation of the project itself. This is not because my group members are lazy – on the contrary, they work very hard. They were simply ill-equipped to work on most parts of this project. Neither Kan nor Lee had the knowledge required to design intricate processing systems, because they have not taken CSE 420 (which teaches, for example, the concept of a data hazard). They also did not fully understand the algorithms being used, and have no background in image processing – I got some of this knowledge from CSE 471, but also have a personal interest in it, and try to keep up on technical developments in the area. Neither of them have taken ECE 410, which might have made up some for not taking CSE 420, because of the final project in that course. Josh could have assisted me, but he was working on the other half of the project – the SVM. I feel that we did not have adequate resources for this project, in terms of people with appropriate experience or in terms of facilities.

A1.3 Kan Xie – Presentation Design



In this semester, my role in the senior design team has divided into two major parts. On the documentation side, as the presentation prepare person. I am responsible to make all the presentation material and set up the presentations. On the technique side, as the project is very strongly software based. My role along with the other electrical engineer in the team is mainly to support the technique needs to the computer engineer's work. From the beginning of the semester, I have been working on research the object detection algorithm and propose to the team different solutions to achieve the design goal. Once decided the main algorithm, I have found and learned from multiple research articles. When the design process started, I and Lee are

focused on the hardware portion of the project. My task is to write matlab logic blocks on the Simulink layout, I have complete numbers of counter and other logic blocks using matlab code and simulink toolbox on the project mainly the address output from the Histogram of Oriented Gradient, each vector is divided into 9 values depends on their angle and it outputs to a specific address so that later on it will be compared by the support vector machine classifier. Outside of Simulink design, I have also worked on design testing and debugging. Overall, through the senior design project, I have learned to the layout of FPGA and use matlab and simulink to program FPGA. And I had a good experience with object detection algorithm program and layout.

A1.4 Josh Frankfurth – Documentation



During the course of this semester I have learned the importance of working closely with others. If you do not work closely often the probability of being left behind in a project is high and playing catch-up is not an option with only a 3 month project.

Our Proposed Design Solution states that Design Team 04 will use Matlab and Simulink utilizing as many built in Xilinx and Matlab functions as possible to minimize the work that our team must do to accomplish low level object detection.

This semester I was responsible for creating a SVM (Support Vector Machine) in Matlab. We utilized Matlab in hopes that the transition to our FPGA would be easy. I was in charge of learning how to optimally use the HPCC (High Performance Computing Center), manage massive amounts of memory (10 to 20 Gigabytes of ram at a time) and take advantage of multi-threaded applications to accelerate the development and creating our Machine Learning System to enable the ability for pedestrian detection. Developing this system was not an easy task because it has to be in parallel with the development of our FPGA pipeline. Essentially everything that has to be done on the board must also be done in Matlab. The design team did not have a full understanding of the hardware until we began using and developing on the board. This is mainly due to the team never working with a FPGA before and having to take small implementation steps at first before we could take large ones like implementing a full machine classification system on a very basic system. Sometimes changes not for seen by the group a month ago would cause a small change in the SVM system and we would have to retrain our entire project. Part of my job required this so I wrote several small programs to be able to change the code, update it, queue compilation and retrain the system.

Our project was monitored by our facilitator who we meet with weekly to make sure we were on task. Our design criteria was mainly set half way through the semester but always required more tweaking especially in our FPGA pipeline. One important aspect of the pipeline is delays, which are necessary to allow for values that are being passed in parallel to reach the same destination at the same time. Timetables were a very difficult task for our group. With only four group members (two less than the previous semester) we had trouble finding time to write all our reports, do the homework's, then the labs for 6 weeks, the technical research for our project, our 3 other classes each, and then do our Senior Design Project. While we accomplished our goals it was only because of hard effort, and many weekly meetings, and late nights with co-operation between all of our group members. Our work was mainly portioned on the fly based on who was available to work on what aspects of the project when. The high level distribution involved myself working with all things matlab, John working with all things Simulink, and Kan and Lee devoting resources to wherever they were critical for development such as helping John with the pipeline and doing the posters, and documentation.

Appendix 2: Literature and Website Reference

Gilroy, Amy. "Back Up Camera Sales to Climb Over 40%". CEoutlook. September 8, 2010.

<http://ceoutlook.com/back-up-camera-sales-see-steady-gains/>

"NHTSA Proposes Rule to Reduce Back-Over Crashes". Ride connection. December 17, 2010.

<http://rideconnection.blogspot.com/2010/12/nhtsa-proposes-rule-to-reduce-back-over.html>

DTREG, "SVM – Support Vector Machines",

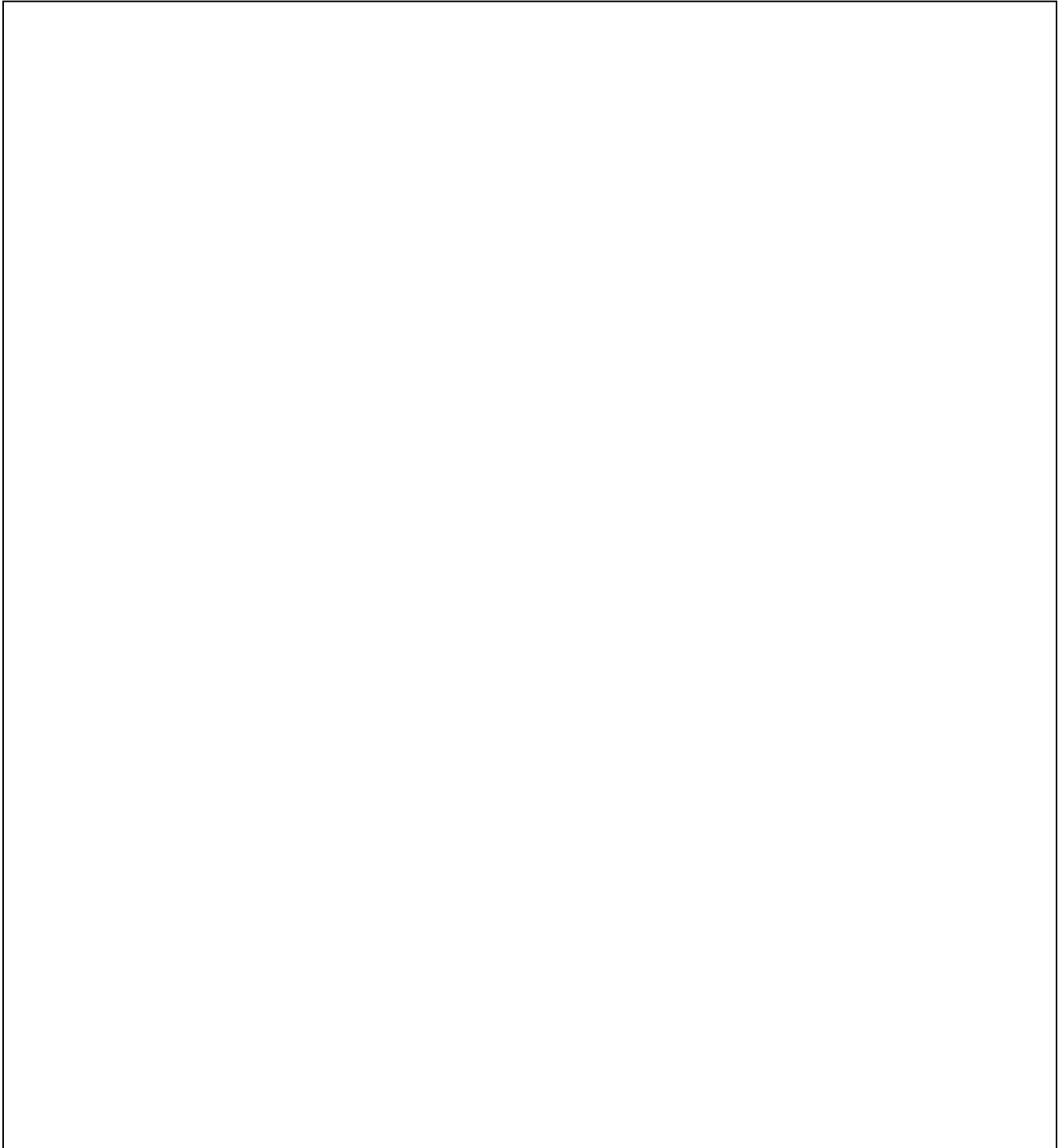
<http://www.dtreg.com/svm.htm>

Omid Sakhi , "Creating a support vector machine for face detection",

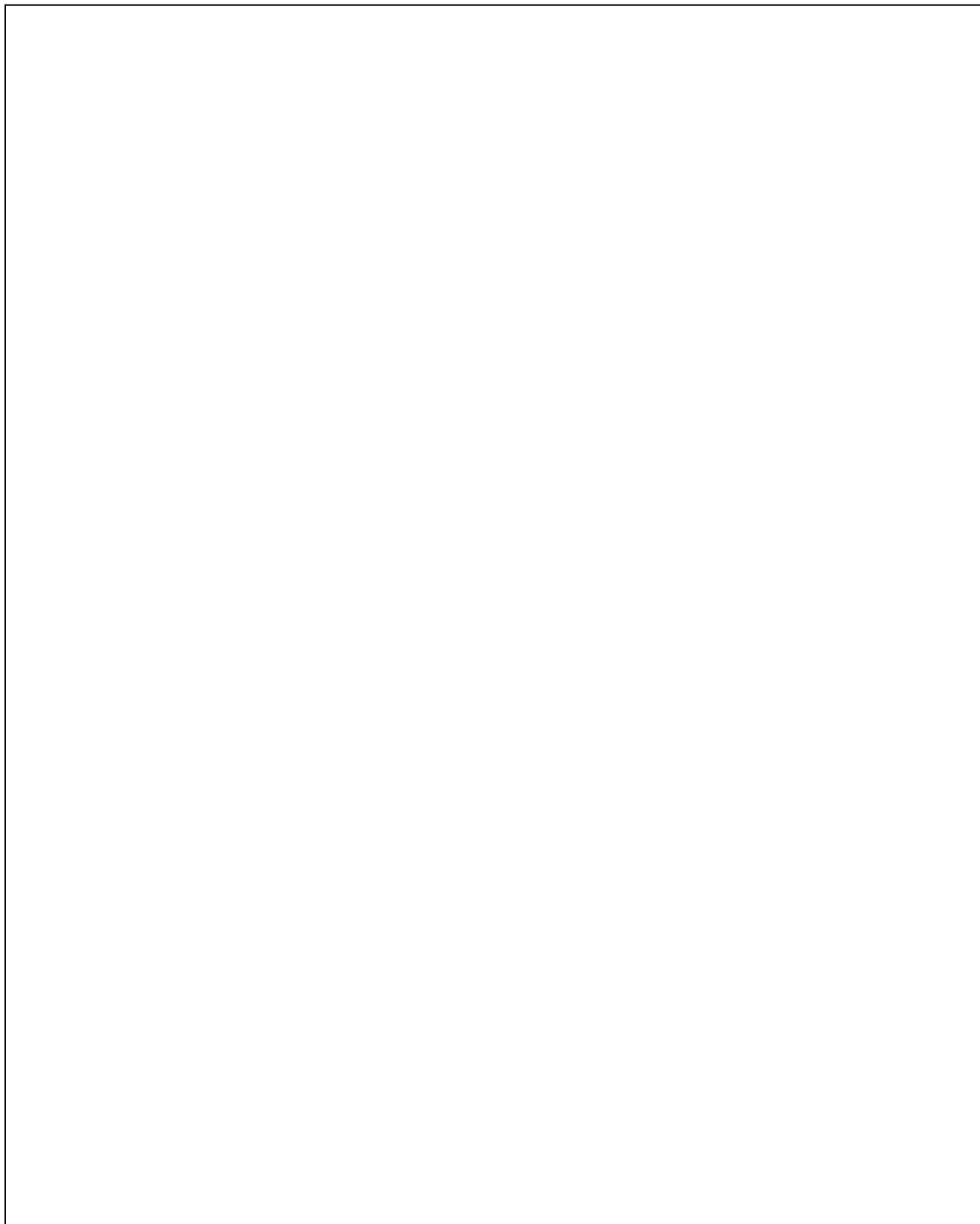
www.facedetectioncode.com

Appendix 3: Detailed Technical Attachments

A3.1 Histogram of Orientated Gradients in Matlab



A3.2 TestClassifier.m Matlab Function



Appendix 4: Gantt Chart

