Sign Detection System for Real Time Applications

William Christensen and Brock Harris

Advisor: Prof. Cherrice Traver

November 2019

Contents

Introduction **1**

History of Driver Assisted Cars . . . 1

Why are these cars important? . . . 1

How do they work? . . . 1

Background **2**

Computer Vision . . . 2

Feature Extraction . . . 3

Object Classification . . . 3

Hardware Acceleration . . . 3

Design Requirements **4**

Goal of the Project . . . 4

Design Specifications **4**

Overall Operation . . . 4

Performance . . . 5

Design Alternatives **5**

Camera type . . . 5

USB Camera . . . 5

FPGA Camera . . . 6

Our Choice . . . 6

Color space . . . 6

Grayscale . . . 6

RGB . . . 6

LAB . . . 7

Our Choice . . . 7

Feature Extraction Algorithm . . . 7

Histogram of Oriented Gradients . . . 7

Scale Invariant Feature Transform . . . 7

Our choice . . . 8

Classification algorithm . . . 8

Support Vector Machines . . . 8
K Nearest Neighbor . . . 9
Our Choice . . . 9
SVM implementation . . . 9
Bare Metal . . . 9
Linux . . . 9
Our Choice . . . 10

Design **10**
System Overview . . . 10
Materials & Economic . . . 11
Histogram of Oriented Gradients . . . 11
Support Vector Machine . . . 14
Hardware Software Co-design . . . 14

Testing Plan **15**
Preprocessing tests . . . 15
Sign Recognition . . . 15
Real-Time video ability . . . 15

Division of Labor **16**

Project Schedule for Winter Term **16**

References **18**

Introduction

History of Driver Assisted Cars

In mid October of 2015 the automotive engineering company, Tesla, made headlines with their new car, the Tesla Model S. They were creating a system that would allow the car to drive and park semi autonomously. This means that using advanced sensor technology such as radar, lidar, sonar, and cameras, these cars can preform certain task without input from the driver. Because of steps like this in the automotive industry, functionality such as parking without input from the driver is now becoming common place.

Why are these cars important?

Cars have always been a very dangerous tool, over 90 people die from automotive accidents every day in America and technologies like driver assisted cars can lead to a much safer driving experience for everybody. In order obtain a driver's license in America a person needs to get many hours of training and pass a basic proficiency test. Despite this every driver still looses focus and makes mistakes all the time. How do we remedy this? One way that car companies are trying to solve this problem is by making sure that the driver doesn't miss important details such as lights, road signs, or things in the road. By using sensors that assist the driver through providing split second information, the car can pick up the slack that the human let's go. This, however, is not a easy task. By making sensing technology that is not only effective, but also cheap, and energy efficient, we can make these driver assisted cars more prevalent on the road.

How do they work?

To accomplish this semi autonomous driving capability, many companies have put multiple sensors in their cars to detect obstacles in all directions. The most important sensor in most cars today is a camera. Unlike other sensors cameras are able to capture a level of detail beyond the shapes or distance of objects. using cameras, we can see what is written on a signs, walls, or billboards. These messages, while generally meant for drivers can still provide a great deal of useful information to the car if it is interpreted correctly.

Cameras are sensors that capture a snapshot of a environment, into a two dimensional array of values. Each of these values is called a pixel, and each pixel is made up of three color channels, red, green, and blue. Not every pixel has important information for a driver, so the challenge for developers is how to extract only the most important information in the least amount of time possible. Processing images can be very time consuming because it often requires a processor to move through every pixel and decide if it fits into a pattern that is useful or not. With image full HD images (1920x180) there are 2,073,600 pixels to process

and, more often than not, each of these pixels needs to be processed more than once to gather the needed information.

Because of the problems surrounding dealing with these cameras finding algorithms that can deal with image inputs in a time sensitive manner. We want to create a prototype of a system that can detect signs in real time and report its finding s back to a driver of the car or to a central system. We have chosen to detect stop signs, for this prototype. For this we plan to use Computer Vision algorithms that have been shown to work with similar problems. However we also know these are not fast enough to simply implement them on a computer. So we have planned to use special hardware to run some of the algorithms on the input images.

Background

When discussing the specifics of the design and choices behind it, a discussion of the methodologies behind the process is necessary to provide background on key terms and phrases used throughout the rest of the report. Both the ideas of Computer Vision and image processing will be brought up, but the difference between them is very important. Image processing takes in some image and outputs a modified image in some way. Computer Vision is a more complicated idea which may use image processing in order to accomplish its goal but ultimately is not necessarily image processing. Computer Vision focuses more on what a machine can see in an image based on patterns, and takes the input image and outputs some task-important data.

Computer Vision

Research behind Computer Vision began in the 60's by Artificial Intelligence (AI) researchers, like those at MIT in 1966[7]. From this, there has been significant advancement towards incredibly accurate and advanced Computer Vision systems that can identify any number of things. These systems have used a number of different algorithms to identify what they wish to find, whether it be motion, an object, or similar images. However this project requires a specific type of computer vision for object detection in order to detect stop signs.

This type of Computer Vision usually has a few steps involving image processing, feature extraction, and finally the most important section classification. The prepossessing is fairly simple and involves a few basic functions to change the image so that features are more easily extracted. There will be further discussions of feature extraction and object classification.

Feature Extraction

Feature extraction refers to a broad idea of taking image data and converting it to a more compressible set of data that is more meaningful to a computer where features like large changes in pixels are more prevalent and small changes or noise in the image can be ignored. An example of this kind of algorithm is Histogram of Oriented Gradients(HOG), defined by a Dalal and Triggs in 2005[4]. HOG was created to allow for efficient people detection. It operates by extracting groupings of gradients, which are changes in pixel values, into histograms in areas across the entire image. This is just one algorithm for feature extraction and is algorithm selected for this project. The reasons for this will be discussed later.

Object Classification

Object Classification refers to the process of taking the processed data that came from an image, and using some algorithm or function and determining whether the image contains a specific object. This requires a pattern recognition algorithm that needs to be trained on a labeled data set. One commonly used classifier is called a non-linear Support Vector Machine (SVM) created in 1992 by Boser, Guyon, and Vapnik[3]. This is an algorithm that allows for classification of data using a kernel method to handle non-linear classifications for more accurate classifications. The specifics of the SVM which is used in this project will be discussed in the design section.

Hardware Acceleration

Hardware Acceleration is a process in which algorithms can be accelerated by parallelized processes. This allows these to run in algorithmic specific hardware, usually on a Feild Programmable Gate Array (FPGA). FPGAs are very good at increasing the speed of highly parallelizable algorithms. One of the most successful algorithms that has been accelerated like this was the HOG, discussed earlier. Because, this algorithm, while very effective at detecting people without using a neural network, was still very slow. Advani et al.[2] showed empirically that the algorithm could be accelerated significantly by using hardware acceleration. They also showed that and FPGA implementation was the best way to implement this algorithm compared to standard computing. Hahnle et al. [5] created a system that uses the accelerated hardware capabilities of the FPGA to create a real time pedestrian detection system on high resolution images. Their system worked on full HD images and classified and image in under 150 µsec. Ngo et al. [6] also boast a very fast extraction rate of 526 frames per second per image from their results which shows an incredible speed for an an algorithm running on this kind of data size.

Design Requirements

For this system we need to be able to determine whether a standard American stop sign in a 720x1280 (HD) image. We would have two output states in total. One output state that signifies there is a stop sign in the image, and another output state that signifies there is not a stop sign in the image. Because stop signs often are not uniform we need the system to work on all sign regardless of the level of vandalism that might be on the sign. If the sign is clear to the user, it should be clear to our system.

Because we want this system to be useful in a driver assisted setting, we want this system to be able to do the classification in real time. This means that each image needs to be classified before the next image comes in. We want our system to work on cameras that operate at 30fps. This means that the image must be classified in under 3.3ms. We also need it to have a acceptable level of accuracy. We need to the system to have an accuracy of 80%. It is also more important that we have a low false positive score or no stop signs are missed. It is better to predict more stop signs that are not there.

Goal of the Project

The goal of this project is to be able to detect the presence of a road sign in an image in real time. To accomplish this we need the system to be able to process and then detect road signs in an image in under 0.03 seconds with a accuracy of at least 80%. These two criteria are important because the system needs to be function on video input. While we are not trying to make a system that could be sold to consumers, we want to make the system as robust as possible, so that it could be used in future work without issues.

Design Specifications

Overall Operation

The basic input to the system will be a camera that will feed raw video to be processed into the FPGA that will then determine if there is a stop sign then it will be reported to the user via an On/Off System like an LED. The FPGA will also be communication with a Hard Processor on the same chip. These two things communicate via 16 bit high speed busses on the DE10 board that we plan to use. The basic flow of the system is shown below.

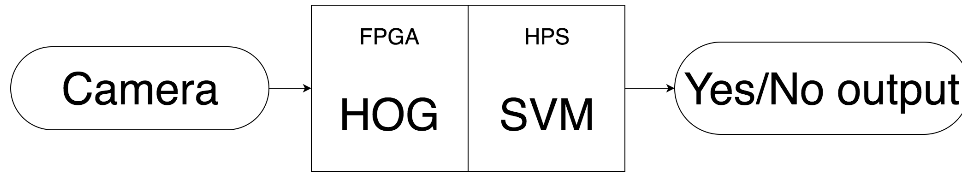

Figure 1: General Flowchart for Operation

These basic inputs and outputs are all the necessary outputs for the entire system most of the work will be done internally in the system rather than in anything physical.

Performance

In order to solve our problem the system will need to take in image data from a camera, process and detect signs all before the next image is captured from the camera. The standard for what we will consider real time is 30 frames per second (FPS) with a resolution of 1280x720 that most cheap off the shelf cameras can maintain. The system should be able to detect the presence of a stop sign and report its presence to the user. Most systems that currently run Computer Vision applications run with an accuracy from 65% to 95% depending on the difficulty of detection due to colors difference and edge blur. Because signs tend to have similar images, we would like to see 80% accuracy from the final system.

Design Alternatives

During the design stage of our project we looked into many different solutions and different iterations for each part of the project. We will be discussing each of these below.

Camera type

The camera was one of the first things we looked at in the entire project. As it is the only input we have for our system it was important to choose a camera that had good enough specs while also being simple enough to integrate into the final system. We looked primarily at two options a USB Camera, and FPGA specific camera.

USB Camera

A USB camera or webcam could be used and would operate as a UVC or USB Video Class Device and would operate by sending video in a multitude of different video formats and color formats [1]. This would be difficult to do because many of these devices are designed for a consumer who doesn't care about specifications or

about what communication protocol is being used. This means there is no data on which format or how data is being sent from the the device and what we could use to decode and use the data for feature extraction and classification.

FPGA Camera

The FPGA compliant camera is a camera created by the producers of the selected System on a Chip (SoC) and is designed to provide direct uncompressed digital picture data and the formatting of this data is well documented and allows the camera to be easily integrated into the project. However because of this special camera is more expensive and camera specs may be lower than what we could find for a market ready camera of the same price.

Our Choice

We chose to used the FPGA camera. This camera provides more support for the hardware we are looking at. This support allows us to focus more effort on the algorithm. The price of the camera is a small price to pay as USB webcam communication adds a significant amount of complexity to our project.

Color space

The color space is how the color information is represented as we pass the image through the algorithm. The camera we have chosen in the sections above uses the bayer color scheme. With is a modification of the RGB color space. it can be useful in some circumstances to convert the pixels into a different color space before processing them. This converstion, while adding some computational time, can lead to an increase accuracy of the system.

Grayscale

This removes all the colors from the image uniformly so that the image is made up of shade of gray. Reducing the image to grayscale reduces the performance of the algorithm by 1.5%. Because of this we decided to look into other solutions like RGB or LAB which allow for full color imaging.

RGB

The RGB color space consists of all possible colors that can be made by the combination of red, green, and blue light. It's a popular model in photography, television, and computer graphics.

LAB

Cie-L*ab is defined by lightness and the color-opponent dimensions a and b, which are based on the compressed Xyz color space coordinates. Lab is particularly notable for it's use in delta-e calculations.

Our Choice

We chose to use the RGB color scheme, because [4] showed that the color scheme does not have a large impact on the accuracy. This also means that we have to do minimal color conversion before feeding the pixels in to the HOG algorithm as we could simply select a common camera with RGB output.

Feature Extraction Algorithm

The purpose of this algorithm is to take an image and create feature that could be useful to a machine learning algorithm to detect something in an image.

Histogram of Oriented Gradients

HOG starts by taking an image and converting it to gray-scale or by taking the maximum color value for that pixel and making that the gray-scale value. Once in gray scale it moves over the image from left to right then top to bottom looking for gradients in the image. A gradient is a place were the first pixel moves from a low pixel value to a high pixel or a negative gradient when a pixel moves from high value to low value. For each pixel a direction vector is created that represents the direction of the gradient for that pixel. The once the left to right and top to bottom gradient vectors for the pixel are created they are combined to get the overall gradient vector for that pixel.

Once every pixel has an overall gradient vector there are put into groups of 3x3 (something we need to decide) and gradient vectors for that group are put together. Also a bin value is determined so that if two pixel vectors are within say 10 degrees of each other they are added together to make a larger vector in that direction. This results in a block of pixels having a star of vectors in the middle where there might be one or two vectors that are clearly the longest. This start of vectors is the feature that is fed into the support vector machine.

Scale Invariant Feature Transform

This is an algorithm that is commonly used in shape tracking and shape detection. This algorithm starts by finding a series of interest points in an image. These interest points can be found through a edge detector line detection, other methods. It then creates a feature for each one of the interest points by looking at

the gradients around the interest points. It then takes these gradient collects them into a histogram and compares them to all the found features of the object.

The algorithm is very effective at tracking object objects in images. It is used in robotic application for mapping rooms by finding fixed points in the image and watching how they move in subsequent images. This is very similar to our ultimate goal. Despite the fact that it could work well for our goal This algorithm does not seem to be easily implementable on an FPGA and would not be as fast to run as the HOG transform. It may also not be as effective in low light level images due to a potential lack of interest points.

Our choice

We chose to use the HOG algorithms because it has been shown to be effective at detecting different kinds of objects and is much better at detecting objects that are not sharply defined. While stop signs are often sharply defined, they also do sometimes have obstructions that could lead to worse performance on SIFT. HOG is also much easier to implement in hardware and would allow for more a speed up.

Classification algorithm

The purpose of this algorithm is to take a list of features from an image and decide whether that image contains the object we are looking for. There are many types of classification algorithms we could have used but we looked at two algorithms that have been show to work on the HOG features.

Support Vector Machines

An SVM is a machine learning algorithm that takes in a set of features from an image and using data from a large data set can determine where an object is in that image. It does this by turning each image that it receives into a point on a plot. It then uses the plot point it already knows the answer to to create a linear function to separate the images with the object and without the object. Once this line is created all the algorithm needs to do is check if the new data point is above or below the line to determine if it contains the object. This algorithm is commonly used in conjunction with HOG and relatively common algorithm, with lots of resources. Tends to provide better results than K Nearest Neighbors (k-NN), another classification algorithm we will discuss after this. However, This algorithm is a little bit more complex to implement than k-NN and may require more training data in order to work well

K Nearest Neighbor

K Nearest neighbor takes a plot of values just like the SVM but in order to determine whether the image contains the object it takes a some amount k of the nearest neighbors to the near plot point. If the majority of those neighbors are classified to contain the object then the algorithm returns a positive identification. This algorithm is relatively simple implementation, and simple to understand. However, May not be as effective as other algorithms due to its simplicity.

Our Choice

We will be implementing the SVM algorithm for the project. We chose this algorithm because we believe that it will provide better results than the k-NN and the resources available will be enough allow us to overcome any difficulties related to implementation.

SVM implementation

When programming the hardware processor to run with the FPGA there are two options for running software on the processor. The software can be run on either the bare metal or using a Linux distribution installed on the processor. Most computers that run software have an operating system that keeps track of all the parts of the program. It keeps track of memory allocation it also makes sure that cores are being used efficiently and a lot of other things under the hood.

Bare Metal

When software is run on bare metal this means that it is run without any operating system to manage it. Running the software on the bare metal process removes all the computation time used to maintain a Linux distribution running on the background, but it does add complexity to the programming that can take extra time that we do not have. Because our program is not very complicated, we do not expect the added complexity to be unmanageable. This method also requires a special ARM compiler. We would need to license this compiler to be able to write code for bare metal. If we cannot get a university discount we will be unable to proceed with this method.

Linux

Linux manages a lot of things related to the programming for the programmer. There is little overhead when it comes to system management. Because of this, using the operating system is a compelling idea. We want to spend most of our time working on the feature extraction section of the project. Linux does however

reduce the speed of any program we write on it because it is managing so many things in the background. By the nature of the project, speed is very important to our final goal.

Our Choice

Both sides have pros and cons and either way would work for our project. We are going to program the SVM on bare metal, because of the speed increase and that we do not think that bare will increase the complexity of our software a significant amount. If, however, we cannot get the license for the compiler we will have to used the Linux distribution.

Design

this term we worked on developing a design for the system so that we could get a better understanding for what we will be implementing next term. We cover each aspect of the design below. We start with a overview to give the reader a better intuitive idea of the design and allow for allow for an understanding of the flow of the system. After that we go into materials that will be required. Through this section the reader will get a better understanding of the physical systems that we have chosen for the design. We then go into detail about each of the parts of the system.

System Overview

For our system we chose to combine aspects of work done by Hanhle et al., Advani et al., and Ngo et al. [5, 2, 6] into a system that will allow for both fast computation and ease of development. The system overview can be found in figure 2.

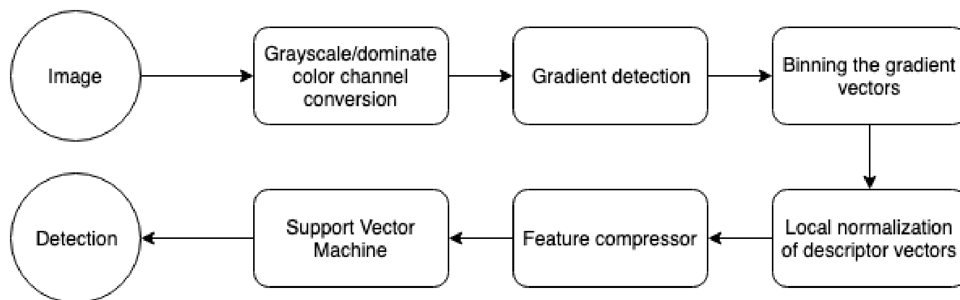

Figure 2: Flow diagram for the overall system

We can see from this diagram above that there are 6 major sections of this system. We start with a camera input. The camera we have chosen outputs each of its pixels in a bayer color pattern. This means

our first step is to convert these pixels into a gray scale form. We then find the gradient vector for each pixel. Next we create a histogram graph for a group of pixels and normalize that histogram. Once we have normalized it we can compress the data and move it into the SVM to classify the image. The first 5 blocks of this block diagram will be done on an FPGA. The last step, SVM. While the operation of the SVM can be sped up by implementing the algorithm on an FPGA, we have not chosen to do this because the added complexity of implementing it on the FPGA is not worth the performance gain for this project.

Materials & Economic

Our development system will be designed using a Field Programmable Gate Array(FPGA) in order to be able to quickly redesign any hardware that doesn't work or that can be improved. The current design uses a Terasic De10-standard to implement hardware and software with the FPGA and ARM processor on the board the De10 SoC retails for about $350 and would most likely be the most expensive part of the project. The De10 could however be taken from the Electrical Engineering Department's current collection to reduce costs for the project. The project will also require a camera to take input to the system, for this the TRDB D5M be used. This a camera sensor that is compatible with the de10, with a cost of $89.

Histogram of Oriented Gradients

There are 5 major steps involved in extracting HOG features from and image. All of these features will be preformed on a FPGA to allow for parallelizable computation. The first step of the process can be seen below in figure 3. The first aspect is moving the image pixel produced by the camera into gray scale.

Gradient detection

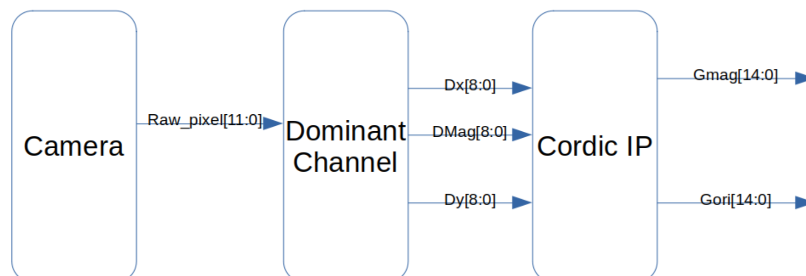

Figure 3: Block diagram for gradient detection system

The camera produces a 12 bit wide signal. This is used to transfer 4 bits for each of the 3 colors in each pixel. Instead of using the average value between all of the color channels to convert the images into gray scale we are using a technique called dominate channel conversion. The goal of this is to encode the change in color along with the change in gradient. We accomplish this by computing the gradient for each color channel and comparing them to determine which channel has the largest gradient. Then the output is 9 bit signed inter value representing the gradient of the pixel and its magnitude. The block diagram for this computation is shown in figure 4. The next step of the gradient detection is to compute the magnitude and direction of each of the gradients. The equations for computing the magnitude and direction of the gradient can be seen below in equations 1, 2.

$$\|G(x,y)\| = \sqrt{G_x(x,y)^2 + G_y(x,y)^2} \tag{1}$$

$$tan(\phi(x,y)) = \frac{G_y(x,y)}{G_x(x,y)} \tag{2}$$

Because of the complexity that come with computing square root operations on an FPGA we will be using the Cordic IP block to preform these computations.

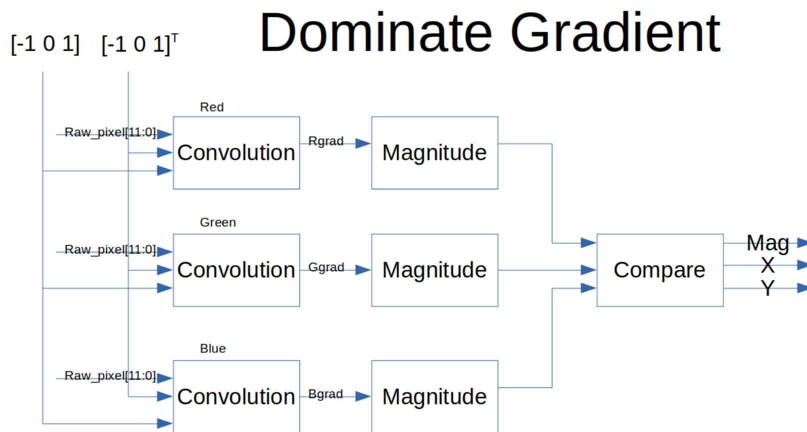

Figure 4: Design diagram for dominate gradient computation per pixel

Through this method we hope to get much more distinct gradient features because of the sharp change from red to white on stop signs. As we can see from figure 4, we start by convolving each color channel with the matrix $\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$ and $\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}^T$ Using these matrices we can find the gradient of the color channel. We then compute magnitude using equation 1. Finally we compare the magnitudes and pass the

largest one to the Cordic module. The cordic module outputs 15 bit wide signals to allow for 2 factional bit below the decimal point as specified by the Cordic IP.

We then move on to the histograming section of the algorithm. The goal of this section is to group all the gradients in an 8 by 8 area of the image into a histogram. This grouping removes noise from the image by making gradients that are not very strong less important slash visible that gradients that are strong. We foresee needing to parts to section, a binning block, and an aggregating block.

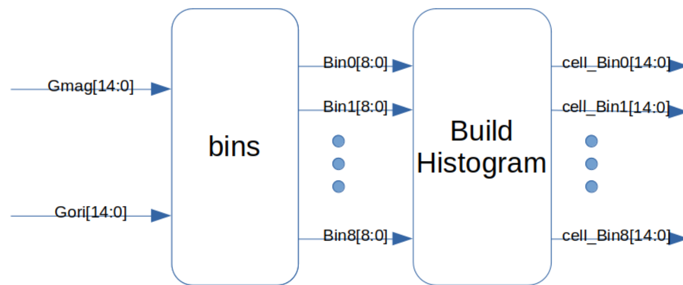

Figure 5: Block diagram for histogram creation

The purpose of the binning block is to take a gradient vector and classify it into a specific bin. To do this [2] has create a system that holds a lookup table for bins and stores already computed values for each orientation. This provides a speed increase that is necessary to this section. Unfortunately, the information for this implementation is limited. We hope to gain a better understanding of how to preform this stage in the design through reading other sources. The output of the binning block is a 9 bit integer value for each of the 9 bins that the orientation can be put into.

The next step is to build the histogram. This is done by aggregating all the computed values for each of the pixel within an 8 by 8 window called a cell. By accumulating the orientation values together we are created groupings that will reduce the size of the data we are transferring and reduce noise in the pixel groups. The output of this block is a 9 bit integer value that represents the magnitude of all the gradients for each of the 9 orientation bins.

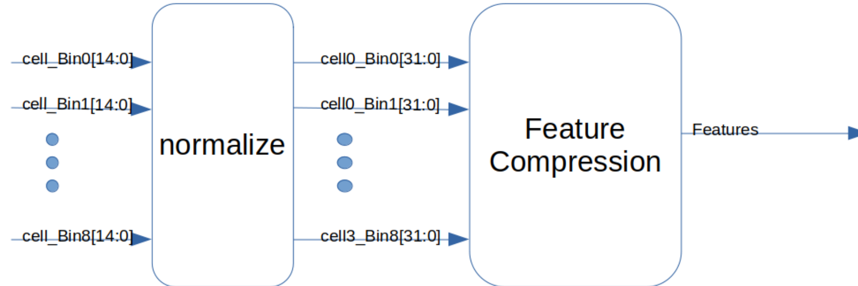

Figure 6: block Diagram for post processing steps

The final step of the process is the the post processing stage. This consists of the normalization of the histograms and the feature compression. Normalizing the histograms is important because [2] has shown a significant increase in detection rate when the features are normalize. to do this we preform an *L2-norm* normalization on 4 cells together to reduce the values in histogram to manageable values. the output of the normalizing block is the 9 bin histogram values for 4 cells. After they are normalized it is important that they are compressed. This allows for faster data transfer of the features to the SVM. The block diagram for this section can be seen in figure 6.

Support Vector Machine

For this project we are focusing most of our effort on the FPGA implementation of the HOG because of the scale of the task. Because of this we plan to use a simple implementation of the SVM for testing purposes. To create this SVM we will be using the SVMlight library. This is a fast prebuilt C library for SVM's. If we are able to get a license for the ARM compiler we will run this SVM on the Cyclone V SoC.

Hardware Software Co-design

Hardware, while often defined through a language called a Hardware Description Language, is defined very different from software. The development process for the two elements of the project are unique but they must be used together to get the system to work in unison. One problem that we have encountered is how to combine these two independent approaches to allow for optimal communication. This is solved through a methodology called Hardware Software co-design. This is a way to structure project to allow for ease

of development between hardware and software. For this problem we are using a tool called Quartus II. Quartus allows us to develop the hardware design in a language called VHDL, a language that allows for a detailed hardware description without having to think about the underlying logic that would be used. Quartus also allows us to add C code to our project and connect it to the VHDL design. To do this we will be using a tool called Qsys. Qsys, a part of the Quartus Design suite, allows us to define how information will flow from the FPGA to the (Hard Processing System)HPS and back. It also gives us a visualization of how each component will be connected and what kind of data can be transferred at what clock rate. It is important to keep track of details such as the clock domain of the data's origin and the clock domain of the destination. Qsys helps us to organize these details.

Testing Plan

Preprocessing tests

An important part of the system is the preprocessing of the image before it is fed into the recognition algorithm. To test this step we will provide the algorithm with still images, to ensure that the output is what we expect. Once we see images that could be effective in recognition we then need to ensure that the images are being processed fast enough to keep up with the standard 30fps input. For this we will time the speed at which the images are being processed and compare them to the desire speed of at most .015 seconds. We want to see the HOG produce the features in this small amount of time, but because of the nature of the feature it is hard tell if they are correct.

Sign Recognition

The system will need to effectively recognize stop signs in images with an accuracy of around 80% and will need to preform better than simply guessing a 50% yes or no response as to the existence of a sign. We can start by testing the recognition algorithm on the features extracted from the FPGA against a naive recognition strategy such as random assignment. Once The system shows a statistically significant increase in accuracy over the baseline algorithms we can adjust the algorithm to provide better results.

Real-Time video ability

The last test we will perform is test the system with the camera providing a continuous stream of data to the FPGA. This will be the real test of the system to show that it can keep up with the input and still maintain accurate detection accuracy.

Division of Labor

Because this is a group project we have divided the project into two parts this term. Each part will have a designated leader, in charge of ensuring that milestones within that subsection are accomplished. The other partner is expected to help the leader with their section, but in a supporting role. One part is to work on fleshing out the details of the algorithm that we will be implementing. This part was spearheaded by Brock Harris. Brock was in charge of deciding what the structure of the algorithm would be and creating block diagrams that describe the algorithm in more detail so that it can be implemented in the future.

The second part was setting up and understanding the hardware software co-design tools and communications so that come next term, it is an easier transition from a theoretical design to an actual implementation. William Christensen was in charge of this aspect. William did example projects for the FPGA, the HPS, and Qsys to ensure that we know how each of these tools works at the level we need for the implementation.

Project Schedule for Winter Term

- Week 1: We hope to have a better understanding of how the FPGA and HPS communication will be connected. **Deliverable:**We will create a block diagram that clearly defines how the HOG system on the FPGA will communicate with the SVM on the HPS. We will also provide a concrete example of a feature that would be transferred to the HPS.

- Week 2: We need to have a simple classifier that we can use to test the HOG algorithm as we write it. To do this we will need to implement a SVM that we can run on the feature produced by the FPGA. To start with we can use a prebuilt SVM found in the openCV library and run it on the Linux system on the HPS, whether or not that will be our final HPS system **Deliverable:** A basic implementation of the SVM for testing purposes. It does not need to be implemented on bare metal yet.

- Week 3: We need to have a complete block diagram so that we can begin implementing the HOG algorithm. **Deliverable:** Each block subsection should be defined to the most basic component, we do not need to show actual logic but as detailed as possible. This should include what kind of data each block is transferring to the next block. try to provide examples of what the transferred data would look like for each section.

- Week 4: We will need to define what logic will go into each function in the block diagram. This will be the first step in implementing the project in HDL. **Deliverable:** Behavioral VHDL code of each block.

- Week 5: We will begin writing HDL code. start converting the logic schematics from week 4 into actual HDL. We will start with the Gradient Detection blocks **Deliverable:** A Tested HDL implementation of the blocks in the gradient detection section of the block Diagram.
- Week 6: We will continuing to implement the Histogram section of the block diagram. This step includes creating bins for each of the gradients and adding those gradients to the histogram. **Deliverable:** Written and tested HDL for the Histogram section of the block diagram.
- Week 7: We will try to finish up the HOG algorithm by working on the post processing step. This step includes the blocks for feature normalization and feature compression. Through these steps we hope to speed up the classification of the SVM **Deliverable:** Written and tested HDL for the post processing steps of the block diagram.
- Week 8: We want to pull it all together. By the end of this week each of the steps of the project will hopefully be working independently. We will connect them all together using Qsys and begin trying to fix bugs and error that come up. Unfortunately we do not really have enough time to work on speeding up the algorithm if it is slow so we will have to see the speed of the algorithm during this step. **Deliverable:** connected the HOG to the SVM using Qsys
- Week 9-10: For these weeks we will be testing the project. We will likely have a lot of problems with the subsections of this project so it is important that we have as much time as possible to test the components. **Deliverable:** Resolving as many bugs as possible.

References

[1] *Universal Serial Bus Device Class Definition for Video Devices: Video Camera Example.* August 2012.

[2] Siddharth Advani, Yasuki Tanabe, Kevin Irick, Jack Sampson, and Vijaykrishnan Narayanan. A scalable architecture for multi-class visual object detection. In *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–8. IEEE.

[3] Guyon Issabelle M. Boser, Berhardt E. and Vladamir N. Vapnik. A training algorithm for optimal margin classifiers. page 144, 1992.

[4] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1, June 2005.

[5] Michael Hahnle, Frerk Saxen, Matthias Hisung, Ulrich Brunsmann, and Konrad Doll. FPGA-based real-time pedestrian detection on high-resolution images. In *2013 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 629–635. IEEE.

[6] Vinh Ngo, Arnau Casadevall, Marc Codina, David Castells-Rufas, and Jordi Carrabina. A high-performance HOG extractor on FPGA. *CoRR*, abs/1802.02187, 2018.

[7] Seymour Papert. The Summer Vision Project. In *AI Memos 1959-2004*. MIT.