Hardware Accelerated Vehicle Detection Using Computer Vision for a Dynamic Traffic System

Undergraduate Project Proposal

by

Marc Patrick Cruz Celon
2013-14742
B.S. Computer Engineering

Timothy Joshua Dy Chua
2013-13318
B.S. Computer Engineering

Paul Grant Reyes Ilaga
2013-58131
B.S. Computer Engineering

Advisers:

Professor Nestor Michael Tiglao
Professor Wilbert Jethro Limjoco

University of the Philippines, Diliman
April 2017

Abstract

Hardware Accelerated Vehicle Detection Using Computer Vision for a Dynamic Traffic System

Since the invention of automobiles and vehicles, heavy traffic has been a widespread problem. In order to solve or mitigate heavy traffic, we propose a traffic signal system that combines techniques such as Dynamic Scheduling, Hardware Acceleration and Computer Vision. Unlike current traffic signal systems where intervals and durations are pre-defined, our proposed system would be able to measure and adjust to traffic density in real time. Our proposed system is also not limited to measuring traffic density; but also possibly for future work, a system that is also able to detect car collisions. The system would be composed of Traffic Management Modules that makes use of hardware such as Single-board Computers like the Raspberry Pi and the hardware accelerators such as Zybo FPGAs. As far as network protocols are concerned, these systems would be interconnected through the use of the Transmission Control Protocol(TCP) for the safe delivery of packets. Finally, as for the overall architecture, a star topology would provide a centralized system that has its end nodes independent to one another as preferred. In conclusion, all these systems would work hand in hand to bring forth a system that could accurately adjust traffic signal duration in relation to traffic density in real time.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

Since the advent of automobile vehicles, heavy traffic has been a recurring problem in everyday life. According to Merriam Webster, traffic is defined as "the movement (as of vehicles or pedestrians) through an area or along a route" [4]. Consequently, traffic becomes "heavy" when the movement of cars along a road slows down to the point of the vehicles being in a stand still. This incident occurs all around the world wherever there is a bad traffic management system accompanying a certain amount of cars. Heavy traffic has been known to cause some health and psychological problems. Some of the bad effects of heavy traffic include the waste of time, increased stress levels, increased in weight, and the rise of carbon monoxide levels. [5]

For this undergraduate student project however, the scope of this project covers only Metro Manila in the Philippines. In the Philippines, some of the causes of heavy traffic could be attributed to the following: road accidents [6], a huge number of cars [7], bad weather [8], processions or events, and lastly, the traffic signal control system.

According to the Metro Manila Accident Recording and Analysis System which is headed by the Metro Manila Development Authority (MMDA), there have been around one hundred thousand road accidents in Metro Manila for the year 2016. 68,499 of which have occurred during daytime, while 40,823 happened during nighttime. Keep in mind that this covers only in Metro Manila meaning there have been more than at least a hundred thousand car accidents nationwide. [6]

"There are 2,317,204 registered vehicles in Metro Manila; vehicle density is 3,643 per square kilometer, much higher than Singapore or Tokyo; and the volume of vehicles plying EDSA is estimated at 520,000 per day in both directions - over and above the capacity which should only

be 288,000 per day in both directions [7]. “ (Senator Zubiri, 2016) This emphasizes the fact that Metro Manila has a detrimental transportation problem because of the overwhelming number of cars. Consequently, this has been a factor in producing heavy traffic.

The Philippines is known for being a country that is very religious, and is placed the tenth most religious out of forty countries because of its historical upbringing [9]. Especially in Metro Manila, the number of processions held year round is quite notable. Because of these road processions, there would be road blockage; and if there is road blockage, cars would have to go through other roads to arrive at their destination. This is problematic because the roads that are not blocked would have an increased amount of vehicles, thereby increasing the chance for heavy traffic to occur.

The last cause of heavy traffic is the traffic signal control system. The usual set-up of traffic signal in the majority of the Philippines is that the system relies heavily on pre-determined time intervals and is basically open-ended. The main problem is that the system does not react adaptively to the number of cars that are actually on the road. For example, given a four-way intersection with three of its roads free of cars, the last road with cars would still have to wait for its turn because of the round robin system; and this paper aims to solve this particular problem with the use of computer vision. The purpose of this paper would be to provide a traffic signal system that adapts to the actual number of cars on the road; and with it, calculate the appropriate duration of a green-light period and red-light period.

As mentioned, computer vision would be used to solve this problem. Computer vision basically allows the computer to see as a human does, however this sense of sight limits to the basic observational skills such as classifying whether a given image is a vehicle or not. The caveat of this technique is its processing power cost. This means that if you want to perform computer vision algorithms in lightning fast speeds, it would be very computationally expensive. [10] A solution to this would be to use techniques such as hardware acceleration. This technique would offload the processes that are expensive in terms of processing to devices like the graphical processing unit. [11] Another device that is dedicated for hardware acceleration would be the Field Programmable Gate Array, and this paper aims to employ hardware acceleration on this device specifically [12].

1.2 Proposal Flow and Organization

Chapter 2 presents the review of the related literature/ related work. Discussed here is the many types of techniques into regulating and monitoring traffic or measuring traffic density. It also mentions the advantages and disadvantages of each type of system. What follows after

the different techniques of measuring traffic density is the traffic management algorithms. These algorithms determine the amount of time the green light of a traffic signal will show, and also it would take into account the numbers of cars on the intersection. Also for the discussion of traffic management algorithms, cases for a single intersection and for two intersections are also covered. Moreover, as the focus of this paper is to be centered on computer vision, an analysis of different techniques of image processing as to why and how they are used would also be given.

Chapter 3 presents the statement of the problem in the most explicit form. Along with the problem statement, the objectives of this project will be introduced as to formally declare what is to be expected by the end of the undergraduate student project schedule.

Chapter 4 displays the procedure intended for the project. This discusses the methodology while also stating the techniques to be used in the project as well as the components involved in the project.

Chapter 5 presents the results of the project including the data that was gathered for analysis.

Chapter 6 presents a schedule to be followed in realizing the project. The half-way and full-way milestones are also presented and are expected to be accomplished upon the specified deadline.

Chapter 2

Review of Related Work/ Related Literature

As a project that integrates a number of techniques, it is required to have covered all the main areas of technology that are to be used in this project. These areas mainly include traffic regulation techniques, intelligent traffic management algorithms, and finally computer vision along with hardware acceleration. Each of these areas would have several techniques mentioned along with their advantages and disadvantages as to how they would be utilized in our undergraduate student project.

2.1 Traffic Regulation/ Monitoring Techniques

There have been many proposed solutions to solve the crisis of heavy traffic. Several of which would be discussed along with its advantages and its disadvantages.

The current system for traffic monitoring and regulation implemented in Metro Manila is the Hermes, a project undertaken by Indra [13]; Indra is a tech company that involves itself in defense and security, transport and traffic, and etc. [14]. The Hermes makes use of electromagnetics in the form of induction loops to estimate the traffic density on each side of the intersection [15]. Once a car enters an induction loop, there would be a change in inductance which would be measured thereby sensing the car. An advantage of this technique is that the cost of these loops are inexpensive thus providing a system that could be used widely in terms of economy. However, the cost lies in the need to excavate roads to be able to install more induction loops for scalability. [16] These could also incorrectly account for cars which happen to park on the side of the road on top of a induction loop.

The second method of measuring traffic density is through the use of radio frequency identification (RFID). The method uses three components: the roadside unit, the junction unit, and the mobile unit. The mobile units are simply cars which contain a RFID tag, while the roadside units and the junction units have a RFID reader. The RFID reader would be able to sense or detect all the RFID tags in each of the surrounding cars thereby approximating the density of traffic. [17] The complexity of this system would be to have the government impose a law that requires all car manufacturers to include an RFID tag along with their products. Another problem would be interference when it comes to operation with metallic objects and water; these factors would decrease the accuracy of the traffic density approximation. There would also be an issue of privacy as anyone with an RFID reader is able to read the tags. [18]

The third method involves computer vision through the use of cameras. Computer vision is defined as "a field of computer science that works on enabling computers to see, identify and process images in the same way that human vision does, and then provide appropriate output [19]." In this traffic intersection scenario, computer vision allows for the actual count of the number of cars; as it is determined by object identification in a given frame of a video feed. A unique advantage this technique has over the other methods is the capability to expand or evolve the functionality of this method. Computer vision, on top of calculating the traffic density, would also be able to detect car collisions or road blockages due to a sudden inflow of cars. However, the main drawback is that it requires a huge amount of processing power that would be devoted to image processing, as it involves the preprocessing of images, segmentation, and etc. [20]

2.2 Intelligent Traffic Management Algorithm

Regarding most provinces in the Philippines, the Traffic Management Systems (TMS) on intersections use a static approach [13]. This means that for every phase of the signalling system, there is a predetermined time interval at which the phase will execute. A phase is a combination of traffic signals that allows a safe flow of traffic. There is also a predetermined order at which these phases will run [21]. This approach is open-ended and therefore, has a large room for improvement. Systems like these are prone to inefficiency due to dynamic factors such as vehicular traffic throughout the day and other specific cases such as accidents happening on a specific lane or emergency vehicles. We must try to implement a TMS that is intelligent enough to consider these cases. Luckily for us, as stated in the previous section, we have a method of gathering information about the current traffic state. We can use these collected information and treat it as an input to our system, while the traffic signals will be our system output.

2.2.1 Single Intersections

Chakraborty et.al. [1] has designed a traffic management control system that fits well with our desired system. It implements a priority based scheduling for choosing which phase will be activated next while the greenlight duration, also known as phase duration, will depend on the frequency of high priority vehicles (HPV) arriving at the intersection.

Phase scheduling works mainly with the priority assignment system. It identifies that some vehicles require the use of the intersection more than other vehicles. Even on emergency vehicles, there are still different priorities that are assigned. For example, an ambulance will have a much higher priority over a police patrol, but a fire truck will have a higher priority over an ambulance. In essence, the green light signal will be given to the phase that has the most number of HPVs. When all phases have the same priority, the system will decide the next phase by considering the traffic density. [1]

Phase duration is a function of the frequency of the arrival of high priority vehicles. It is initially given the maximum duration, and this duration will be inversely proportional with the frequency of HPVs. For this approach however, there may be a smarter alternative. Remember that on normal operations wherein no HPVs are present, the green light duration will be quasi-static since the green light duration will remain on its default value. [1] We will try to fuse this method with Osman et.al.'s work [20].

Osman et.al. [20] have designed an algorithm to dynamically allocate the waiting and running time of each intersection by considering the ratios of the amount of vehicles that is incoming and outgoing from the junction, and therefore, dynamically allocating the phase duration. This will be implemented using a junction "snapshot". The snapshot will be taken every phase duration, established by the default value, multiplied by the number of phases.

Using a processor-intersection analogy, an intersection can be treated as a dual core processor [22]. This is because the traffic signal system is able to provide the green light to two non-conflicting lane at the same time to improve throughput. An example of this would be in a four-way intersection, two opposite left turn signals can be given at the same time; since the cars that would pass through would not collide with each other. However, a left turning lane and an opposite straight lane can not be given the green light at the same time. Shown in figurex is a compatibility diagram that represents all possible cases of these signal pairs in a four-way intersection.

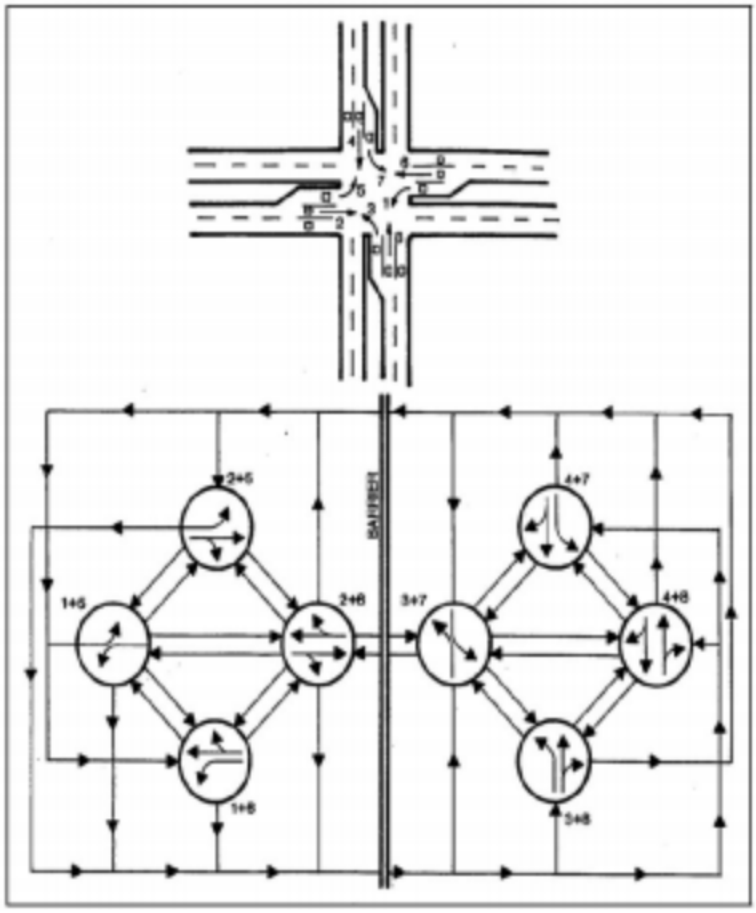

Figure 2.1: Primary Phasing Options[1]

For this project, a dynamic scheduling algorithm will be implemented. There are two algorithms that are going to be considered. These two algorithms Oldest Arrival First (OAF), Younes's and Boukerche's Intelligent Traffic Light Control (ITLC). [1]

For OAF, the order of the first vehicle to arrive per lane is recorded. This information will then be used to schedule the ordering of the lanes, which as the name suggests, with the oldest arrival first. On the other hand, ITLC is an algorithm that prioritizes lanes that has a higher density. ITLC defines a ready area where the traffic density will be calculated. Once the traffic density is calculated, the time it takes to clear the lane will then be computed. The scheduling will be based in the density. The lane with the maximum density will be identified and then the non-competing lanes' traffic density will be compared. Once the pair has been identified, it will be then added to the queue. The duration that is to be added tied with the phase will be whichever duration is greater. After this, the density for the selected lanes will be zero and then the cycle

will run again until all of the densities are zero. The figure below is the aforementioned algorithm and its test results. [1]

Algorithm 1: Intelligent Traffic Light Scheduling Algorithm

Data: TL : Traffic Light; RA: ready area; d_i : The traffic density of the traffic flow i inside RA; t_i: the required time for all vehicles inside RA, at the traffic flow i to cross the traffic intersection.

```
compute d_i and t_i of all traffic flows inside RA;
while d_i of any of the traffic flows at TL > 0 do
    let j the traffic flow with the maximum traffic density (d_j);
    let i1 and i2 the traffic flows that can cross the traffic intersection simultaneously with the traffic flow (j) ;
    if d_i1 > d_i2 then
        P_ji1 = schedule (j, i1);
        d_j = 0.0; d_i1 = 0.0;
        t_j = 0.0; t_i1 = 0.0;
    else
        P_ji2 = schedule (j, i2);
        d_j = 0.0; d_i2 = 0.0;
        t_j = 0.0; t_i2 = 0.0;
    end
    Adjust the t_k of all other traffic flows inside the ready area;
end
```

Algorithm 2: Schedule Function

```
INPUT: traffic flows i and j
if t_i > t_j then
    return t_i
else
    return t_j
end
```

Figure 2.2: ITLC Algorithm [1]

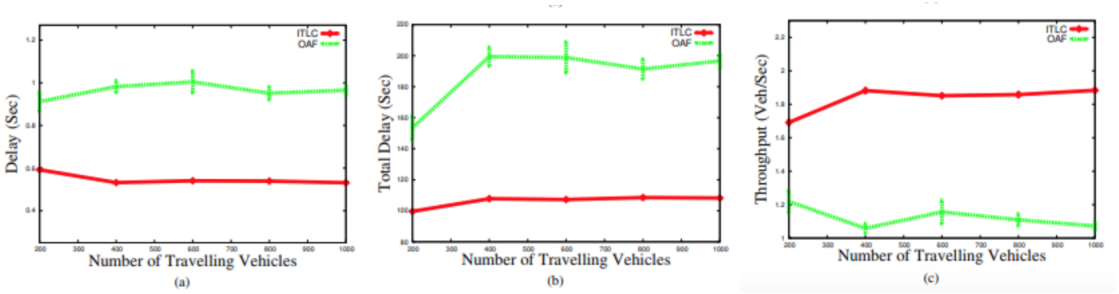

Figure 2.3: ITLC Test Results [1]

From Figure 2.3 (a) and (b), in terms of delay, ITLC is better, this is due to the principle of the maximum density first schedule of ITLC, which assigns the longest green phase to the largest detected density and less number of vehicles wait on the traffic flows with the lowest traffic density.

[1]

2.2.2 Multiple Intersections

To implement an algorithm that considers multiple intersections, we need to look at Chen et.al's work. They basically said that considering the signals and data from intersections around you such as phase duration and phase scheduling, how can we make it such that there will be a continuous green light signalling? This is also called as the green wave concept. We want a green wave to occur because it increases the overall throughput of the system. They implemented this through a predictive algorithm that aims to determine the maximum aggregated passing rate of the whole system and then selects it for the next phase. [23]

2.3 Computer Vision for Traffic Monitoring

Computer Vision has been successfully used to robustly identify in real-time various images of interest such as faces [24], walking pedestrians [25], and alongside other criteria. Different approaches have also been successfully used for vehicle detection such as background subtraction [26], edge detection with model-based tracking [27], and Machine Learning [28, 29, 30, 31].

Background subtraction is a computationally simple processing technique that allows a computer to notice the difference in an image with respect to a reference. As such, it can be reliably used to identify and track moving objects [26]. It works by plainly subtracting intensity values of a reference image (called the background) to the current sample, and filtering the output to emphasize the most obvious changes. Some algorithms introduce adaptiveness for better performance in changes to lighting and weather conditions. However, background subtraction algorithms still perform poorly when subjected to noise, occlusion, vehicles becoming part of the background, and etc.

Edge detection algorithms isolate edges (boundaries of two objects) in an image with a relatively high degree of accuracy. A model of a car could then be constructed from the resulting edge values. Popular detectors such as the Canny and Sobel deliver very high throughput, scalability and flexibility for various methods [27, 32]. It is also more resistant to noise than the background subtraction algorithm. However, it still does not handle occlusion very well, as two cars could be considered as one object using this algorithm. Furthermore, model-based tracking requires a large amount of tweaking and detail to properly optimize the recognition rate. [33] Thus, both background subtraction and edge detection are best used in combination with other classifiers to improve performance.

Machine learning approaches to traffic monitoring have been gaining popularity over the past decade or so due to the inherent versatility in applications and the increase in computing power available [24, 25, 28, 29, 30, 31]. Machine learning algorithms, in this context, usually take extracted features from an image and put it through a trained classifier that outputs whether the chosen image is a car or not. Feature extraction can be done using traditional methods of identifying objects of interest (background subtraction, edge detection). [32]

Viola and Jones suggested using integral images instead to compute features in constant time [34]. These features (called Haar-like features) can then be subjected into thresholds to determine whether the image in the detection window is the desired object or not. To decrease classification time and improve results, a learning algorithm based on Adaptive Boosting (AdaBoost for short) is used to select a smaller number of critical features. These critical features can then be cascaded to form the overall classifier. [24] This method was originally designed and commonly used for face detection but has been successfully applied to vehicle, pedestrian, and other types of objects monitored for smart traffic solutions. [28, 30, 35, 36]

Dalal and Triggs suggested a simpler architecture based on histograms of oriented gradients (HOGs) [37]. HOG features are models of the distribution of local intensity gradients or edge directions of a detection window; moreover, these histograms can be calculated without obtaining the precise gradient/edge positions. [25] This feature can then be inputted into a trained linear support vector machine (linear SVM) which will classify the image as car or not-car. A linear SVM is a machine learning algorithm that tries to find the best "division" between data points and to maximize effectiveness in classification. Due to the use of HOG, it is more robust in finding the general shape of the object (ex. It can detect pedestrians in a variety of poses, as long as they are in an upright position). Thus, it has been successfully applied to vehicle detection and monitoring [38, 29, 39].

As processing power increased, more recent studies have been taken on the viability of Neural Networks in vehicle detection to great degree of success [30, 31]. In particular, Deep Convolutional Networks (DCNs) are more robust to spatial invariance, lighting/weather changes, occlusion between vehicles, and changes in camera angles with respect to other methods, and lends itself to better and thorough classification [40]. However, many machine learning algorithms, especially Convolutional Neural Networks, require a huge amount of resources to train and deploy. Many of them are computationally expensive and require a high-end GPU to achieve an acceptable frame rate [29, 40]. However, some machine learning algorithms have been implemented in field-programmable gate arrays (FPGAs) to take advantage of their inherent parallelizability as a form of hardware acceleration [41, 42, 43]. Ilas (2017) also suggested a modification to the HOG algorithm

to speed it up for FPGA computation [44].

Chapter 3

Problem Statement and Objectives

As of now, the traffic system in Metro Manila is inadequate to accurately manage traffic. There have been attempts such as Indra's Hermes that uses induction loops, however using induction loops would require road reconstruction and maintenance. In light of this, our proposed system to alleviate this problem would utilize Computer Vision. However, with utilizing Computer Vision would come with its advantages and disadvantages. One disadvantage as mentioned previously is its immense processing power requirement. That is why alongside Computer Vision, we aim to utilize Hardware Acceleration by offloading the machine learning algorithms onto the Zybo Field Programmable Gate Array for faster computation. This system would also only be limited to traffic density measurement. Therefore, our proposed system to alleviate this problem would integrate techniques such as Internet of Things, Computer Vision, and Video Sensor Networks. We believe that these techniques would be sufficient in regulating traffic as it adds to the adaptability of the system by a large margin.

The objectives of this project are the following:

- To develop and implement a closed-loop traffic management system covering two intersections
- Computer Vision for Vehicle Detection
- Hardware Acceleration with the FPGA
- To use SUMO for simulating a traffic environment wherein our system will be operating

We propose to create an Intelligent Traffic Management System that integrates with a variety of techniques as previously mentioned. This system would then be implemented in a closed loop system with SUMO.

Chapter 4

Methodology

4.1 RASPI-Node

This device is a Raspberry Pi 3 Model B V1.2 that is running on Raspbian-Stretch v2018-04-18.

4.1.1 Transfer of Images

There were two kinds of image transferring techniques used. The first one used is the command “scp” which stands for secure copy. Security of data is guaranteed as files and passwords when transferred are all encrypted beforehand. Scp also makes use of “ssh” or secure shell which allows the host to read or write files in the computer it is connected to through ssh. [45]Scp was used in transferring the images from the Unity Engine to the RASPI-Node. To allow these devices to use “scp” as part of a python script that is running on the devices during runtime requires paramiko. Paramiko is an application programming interface that provides functions to utilize the scp and ssh terminal commands in the python language, therefore making those two commands available for scripting. Think of it as an automated copy and paste service that runs on python. Copying is done in one device and pasting on the host device or vice versa. [46]

The second one used is a technique otherwise known as struct packing. This technique is used to send the snapshots from the RASPI-Node into the Zybo FPGA. What this technique offers is to send data in a compact form. [47]This reduces memory usage and is quite useful in sending between small devices such as embedded systems like the Zybo FPGA.

4.1.2 Traffic Duration Calculations

The algorithm to be implemented for traffic scheduling is ITLC.

First is the initiation phase. The following information must be stored on the code.

1. Non-conflicting lane pairs.
2. Non-conflicting lane pair signal.
3. Non-conflicting lane pair yellow signal.
4. Alpha, F, and Stf, d

For the first three bullets, the non-conflicting lane pair diagram must be consulted. The roads are labeled from 1 to 8 clockwise from the right-turning northern lane. The right-turning and the middle lane is considered as a single lane only because they are always given the green light together. The diagram is implemented using a list of tuples. The nth list element contains a tuple that represents the two possible pairs of the nth - 1 lane.

For the lane pair signal, Unity supports traffic light control by letting the user provide a string. The string will correspond to the traffic lights of all the lane. The traffic lights are labeled from zero to twenty clockwise from the right-turning northernmost lane. There are two separate traffic lights for the left-turning lanes and the right-turning lanes. One is for the left turning vehicles and the other is for the straight going vehicles. This then provides five traffic lights to a three-lane road. For a four-way intersection, that will total to 20 traffic lights. Each traffic light will be assigned a character on a 20-character string. Each character can either be:

- “r” - red
- “G” - green
- “y” - yellow

There will be two traffic signal string variables, one is for the red and green traffic light and the other is for the corresponding red and yellow traffic light combination. The n-th element i-th tuple on the traffic signal list will correspond to the nth element i-th tuple traffic light combination. The yellow traffic light string is generated from the red and green traffic light string by replacing all the greens to yellow.

For item number four; alpha, F, and Stf is used for the clearing time estimation. “F” represents the distance of a single car from the intersection. x will yield the distance of the x-th car from the intersection. “Stf” will represent the average velocity of the cars in the intersection, while “alpha” represents the starting delay of the first car in the road. The formula $(F/Stf) * d + alpha$ will then yield the estimate time for a road of “d” number of cars to clear.

For the main scheduling phase, the algorithm that is provided by the ITLC paper is followed. The first step is to calculate the vehicle density per lane. Vehicle density by definition is the number of cars normalized per unit distance. To simplify, instead of calculating the actual density, the average number of cars per lane is obtained instead. This is because the input for our duration estimation is the number of cars rather than the density. Since the left-turning and the straight going lanes are treated as one, we must average the number of cars on these two lanes. Since clearing time information and average vehicle density, represented by vehicle number, is already obtained, the algorithm can now be implemented and the schedule can then be determined. Yellow intermediate stages are also inserted in between schedule entries and are provided the duration of three seconds.

For the last step, the execution phase. The phase sequence and the phase duration is already provided by the previous phase so the execution phase is fairly straightforward. We just wait for the duration to pass before providing the next traffic signal string to TraCI until the schedule is finished execution.

4.1.3 OpenCV for Benchmarking

OpenCV is an open-source library mainly used for software programs that is in the field of computer vision. The library contains numerous functions for complexity abstraction which allows a wide range of users to create computer vision themed projects with ease. With this library equipped, the RASPI-node would be capable of vehicle detection. [48]

This then will be used to compare against the performance of the Zybo FPGA in terms of processing speed or frames per second. The Zybo FPGA would represent the hardware implementation, and the RASPI-Node, the software implementation.

4.2 RASPI-Sink

The RASPI sink will be used primarily for data collection and debugging. The two RASPI nodes will periodically send data to the raspi sink for logging. The raspi sink will generate a sink.csv file at the end of the experiment with all of the data that it has received.

The logged data are the following:

- Simulation time in seconds
- Total processing time of the latest generated schedule
- Total time for image transfers over the network

- Total time for the extraction of the vehicle data from the latest image
- Difference of the number of cars extracted using computer vision compared to actual number of cars on the ready area
- Number of cars that entered the north road
- Number of cars that entered the east road
- Number of cars that entered the south road
- Number of cars that entered the west road
- Number of cars that approached the intersection
- Instantaneous throughput
- Average throughput
- Number of cars that left the intersection

4.3 Simulation

Since our traffic management system needs a simulation tool that can provide digital video streams and programmable traffic signalling systems, Simulation of Urban MObility (SUMO) fits very well within our requirement. SUMO can be interfaced with game engines such as Unity and Unreal Engine using widely-available modules on the Internet. This provides us with a tool that enables video streams that can be used for video processing and object identification.

We can also build custom road networks using SUMO. From single-junction systems up to a complicated network of intersections, SUMO allows multiple setups for our system. This may also help with the implementation of IoT paradigms such as sensor networks which requires multiple intersections to communicate with each other for an optimized flow of data.

The traffic signalling systems within SUMO can also be interfaced with Python scripts through TraCI (Traffic Control Interface) which uses TCP/IP and since TCP/IP is a widely implemented protocol, we can even interface SUMO directly to our hardware.

For the simulation process, SUMO(Simulation of Urban MObility) and the Unity Engine will be used to model the intersection that the Traffic Management Modules will interface with. SUMO will be responsible for the generation of the traffic scenario and configuration, while Unity will be responsible for rendering the traffic scenario such that the scenario would seem “life-like”.

Both SUMO and Unity will be running at unpredictable and varying frequencies, therefore a "Server" program is needed to synchronize the two.

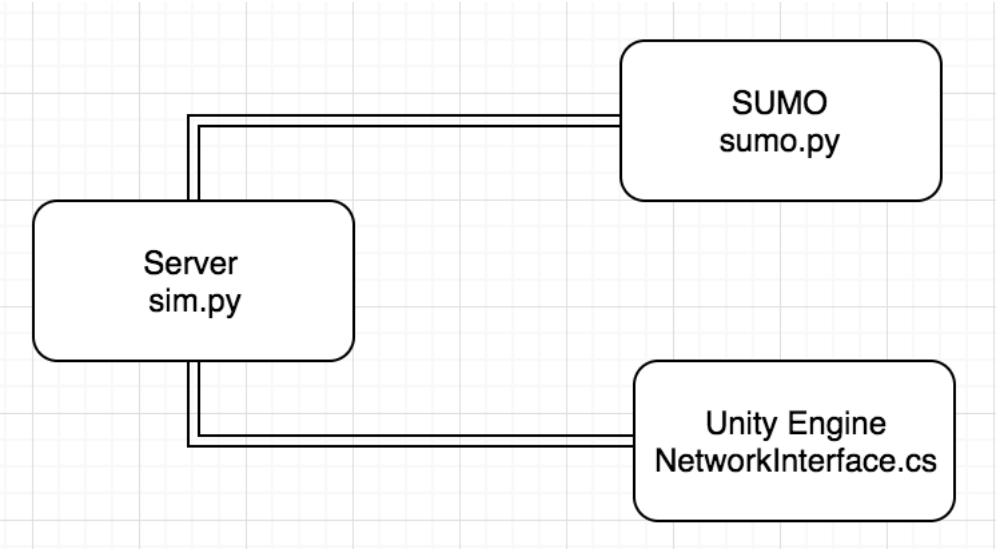

Figure 4.1: Simulation Diagram

4.3.1 Simulation of Urban MObility(SUMO)

Since SUMO is a program that requires multiple files to run, it is imperative to design a framework. The framework will consist of multiple folders that collaborate to produce the simulation. Below are the mentioned folders. [49]

- config - This is where all configuration files are stored.
  - sumo - This is where all .sumocfg files are stored. *.sumocfg files are responsible for the simulation settings such as step duration, start time, end time, output path, etc.
  - vehicles - This is where all vehicle definitions are stored in .txt files. Vehicle definitions include acceleration, maximum velocity, deceleration etc.
- data - This is where the simulation .xml report files will be generated after each simulation. Report files include runtime logs and configuration.
- maps - This is where .net.xml files created from netedit are stored. *.net.xml files contain the information about the road network. Information such as traffic signal programs, road length, road junctions, etc. can be found in this file.

- routes - This is where .rou.xml files are stored. *.rou.xml files contain information about when vehicle instances will be generated, where it comes from, what roads it will pass through, and its destination.

Since a framework is implemented, it is practical to make work easier by using a filename convention. All files related to a SUMO project should have the same filename regardless of the extension. This can be seen in the project repository included in the appendix. Any deviations from this convention is supported by using a command line argument as shown in the program's help page. [50]

Once the program is initiated, it will search for all the required files in the framework and then generate the necessary route file using Dijkstra's algorithm. [51]

Once the initialization is completed, the program will then run SUMO through TraCI (Traffic Control Interface); TraCI is a python-based library used as an Application Programming Interface (API) for SUMO. TraCI allows us to automate, extract data, and run commands in the simulation environment using a Python script. The program can then extract data such as the number of vehicles and their positions to be sent to the server which will then be used later in Unity. [52]

4.3.2 Unity Engine

A TCP/IP interface program should be implemented for Unity as it is the medium of communication between itself and the server. This program should also handle the processing of the data from the network buffer to the game scene. C# is an object oriented programming language and it also has the UnityEngine library which will be very useful; therefore, it will be used for this project. The server will send traffic data extracted from the simulation to this program.

Game engines render frames through periodic intervals and processing is done between each frame. At the start of the game, Unity will read vehicle information from a table. The information comprises of both vehicle id and the vehicle's respective positions; this information is then used to render the vehicle in its position.Initially, at the start of the simulation, the table will be empty meaning no vehicles will be rendered into the simulation. In between frames, Unity waits for data from the network buffer. The network buffer data will come from the sumo script which will pass through the server. Once it receives data, it will populate the table accordingly such that in the next frame, the table will have information and vehicles will begin to show up in the game scene.

Unity gives access to the use of "in-game cameras" which allow us to view the rendered

scene from a camera's perspective. For this simulation, a total of eight cameras would be used; and these cameras are positioned across the two intersections where each of these intersections has four roads each. Now, given that there are a total of eight roads for two intersections, each camera would be stationed in the middle of these intersections and would be facing their corresponding road. For the road makeup, each road is defined to have six lanes. Three of which are lanes for incoming cars, and the remaining three are lanes for outgoing cars.

What Unity can offer is that users are able to attach scripts to these game objects or cameras. These cameras are attached with a similar script each, and these scripts tell the cameras to take snapshots of what they see and save it as "PNG" files to the directory path stated. These are how in game snapshots are made, and these are the snapshots which are obtained by the RASPI-Nodes.

4.3.3 Server

This will be a simple TCP/IP server that is implemented in Python. All other components in the simulation block will communicate through the server, and there will be no inter-module communication without the help of the server. The Server would be responsible for the synchronization required between SUMO and the Unity engine.

4.4 Computer Vision

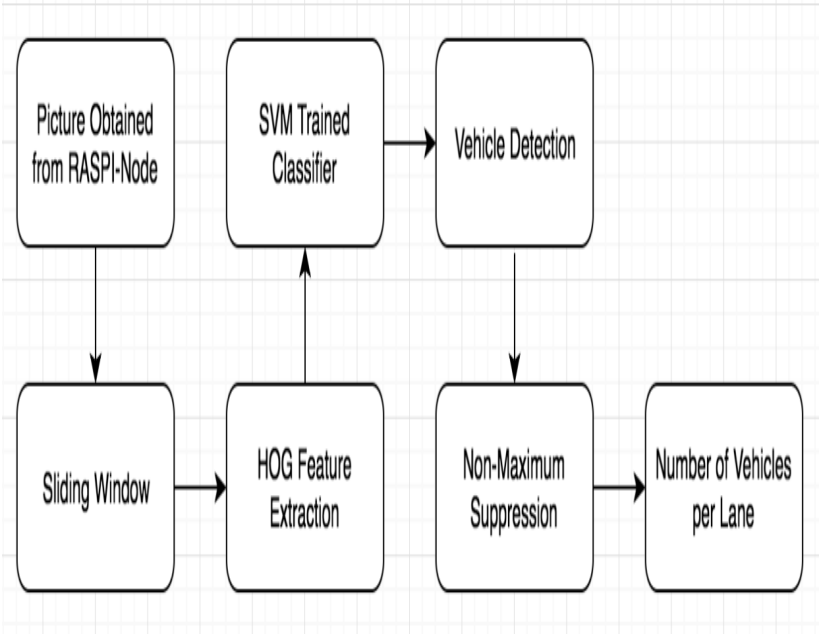

Figure 4.2: Computer Vision Modules

4.4.1 Sliding Window Optimization

Given a picture that is to be processed for vehicle detection, the first step is to perform sliding windows. Sliding windows, as the name suggests, are windows which traverse the picture in the form of rectangles. The image contained in these sliding windows would be cropped and resized for vehicle detection; the image which was cropped is now called the region of interest(ROI). Also, the sizes of these rectangles and how the rectangles traverse the image can vary according to the user's preference. The image below has a blue rectangle, and that is simply the sliding window. [53]

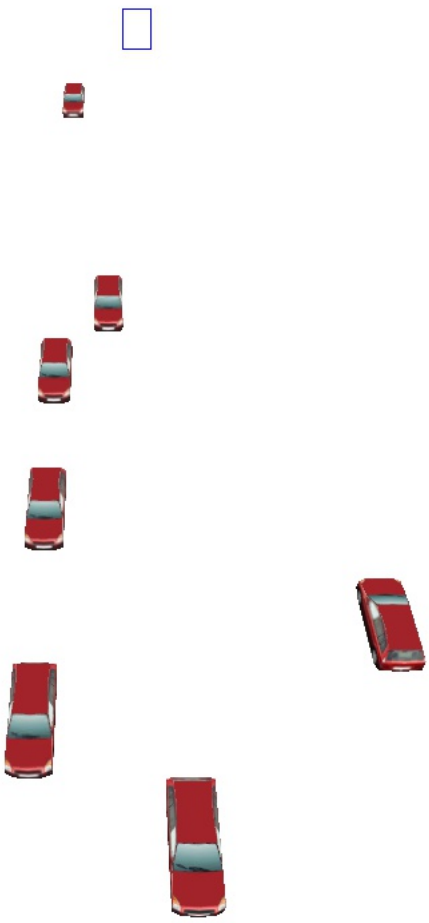

Figure 4.3: An image obtained from Unity with a sliding window in blue

For example the sliding window might start at the top leftmost part of the image which would be the first row and traversing in a left to right fashion; and after it has reached the end image, it would start from the leftmost but this time at the second row. As you can see this kind of image traversal is quite inefficient for the purpose of this project.

Below is an attached image taken from the Unity Engine.

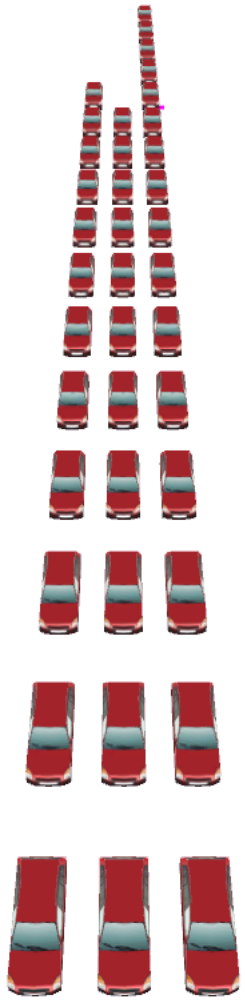

Figure 4.4: An image obtained from Unity with a sliding window in blue

As previously stated, the picture taken by the RASPI-Node from the Unity Engine is from an ingame camera. It would be unnecessary for the sliding window to traverse through the whole image to search for vehicles. To optimize this algorithm, the sliding windows will only be traversing from top to bottom of the three incoming lanes of vehicles as seen in the picture. Furthermore, the area of the sliding window rectangles will increase as the sliding window falls down the rows of the image to account for the change in image depth; since the farther the car is from the user the smaller it would seem.

4.4.2 Histogram of Oriented Gradients Feature Extraction

For any algorithm in Computer Vision or Machine Learning, you would a set of inputs that would be tested for the output; and these inputs are called features. In the example of Computer Vision where this field centralizes on images, the features might be the pixel values that represent the image. These features are then inputted into the "hypothesis equation" where an output can be found. HOG Feature Extraction is a level above just using pixel values as features. The HOG abbreviation stands for histogram of oriented gradients where it takes the gradients of these pixel values and then bin them in a histogram. This method is used over the other feature extraction algorithms due to its high performance in classifying objects. [37]

The algorithm works by calculating the x and y gradients of the image, and converting it into oriented gradient vectors. The image is then divided into multiple cells, which then are grouped into overlapping blocks as shown in Fig. 4.4.

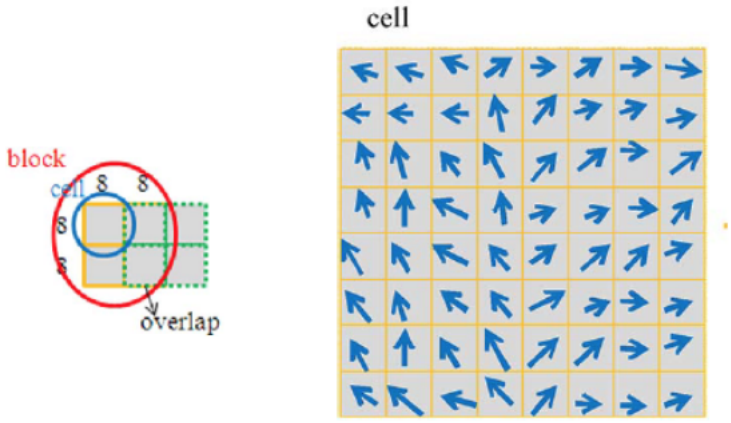

Figure 4.5: A representation of the oriented gradient vectors in an 8x8 cell, and the representation of the overlapping blocks and cells in an image. [2]

For each cell, a histogram will be created by "binning" the vectors' magnitudes via their angles. In most implementations, a "bin" represents a 20° interval from 0° - 180°, for a total of 9 bins per cell, or 36 bins per block. To complete the algorithm, the histogram is normalized at the block level to account for contrast at local regions, which results in a vector containing the bin values per block.

A fixed detection window is used over multiple scales of the sampled image (to also detect different car sizes) to extract the HOG at that particular local area. For example, a 64x64 pixel window region of interest in an image with 8x8 cells and 16x16 blocks will generate a descriptor

with 1764 values. This descriptor can then be fed into a linear support vector machine (SVM), to classify whether the object enclosed is a vehicle.

4.4.3 Support Vector Machine(SVM) Trained Classifier

The Support Vector Machine (SVM) model maps the HOG descriptors as n-dimensional points, and the classifier as the hyperplane separating the vehicle and non-vehicle points from each other. The classifier is represented as an n-dimensional vector with an additional offset b:

$$\vec{w} \cdot \vec{x} - b = 0$$

Where$\vec{w}$ representing the normal vector to the hyperplane. To classify an n-dimensional feature $\vec{x}$, the above equation is evaluated and compared to 0, (or -1, 1, or other values depending on the loss function).

All SVMs used in the implementation will be pre-trained and tested on desktop computers fed with online open-source datasets, along with rendered SUMO and personally acquired images.

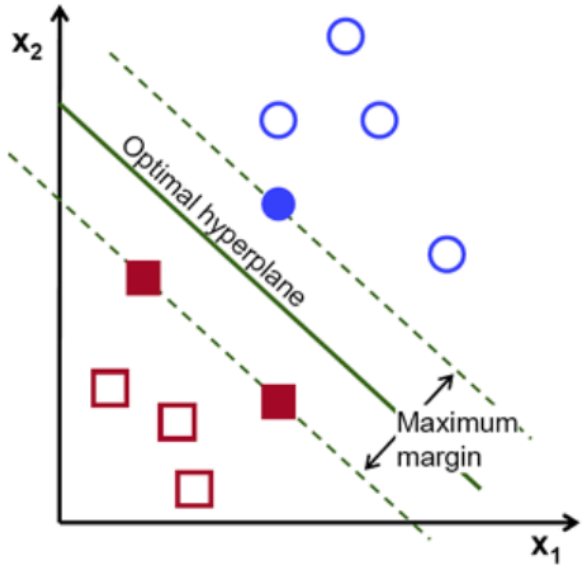

Figure 4.6: SVM Representation of Dividing Two Sets of Data.

4.4.4 Non-Maximum Suppression

At the end of the sliding window traversal, the Support Vector Machine would have detected vehicles over multiple images or regions of interest, and these are called bounding boxes. Some of these detected vehicles might represent the same vehicle, because these bounding boxes might be displaying the same vehicle. This means that these bounding boxes are intersecting each other as shown in the image below, and this is to be expected from using sliding windows. The Non-maximum suppression algorithm would remedy this. Basically, what the non-maximum suppression algorithm does is it sorts the list of rectangles where vehicles have been detected. There is also a threshold when using non-maximum suppression; this threshold is user determined and says how much intersected area is allowed before it declares both vehicles detected the same. The elements of this list each contain the x and y coordinates of the top-left most element of these bounding boxes and their length and width. With this information, you would be able to calculate the area of the intersected region and then if it passes the threshold, then you will keep the bounding box. Alternatively if it fails the threshold, remove it. Attached below are two figures; the one on the left is before the non-maximum suppression algorithm is applied, and the one on the right is after. [54]

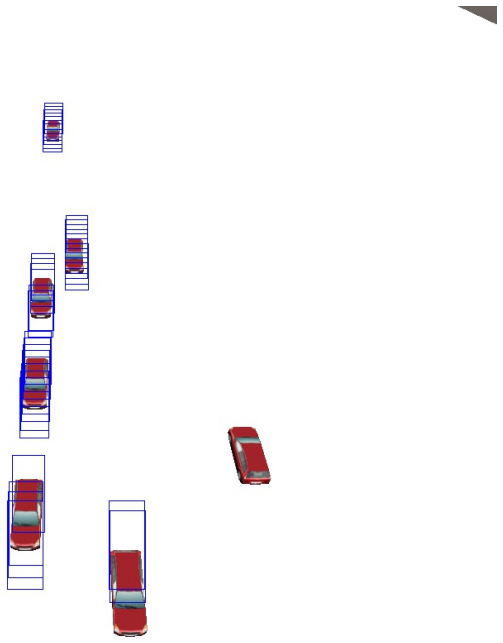

Figure 4.7: An image showing the bounding boxes

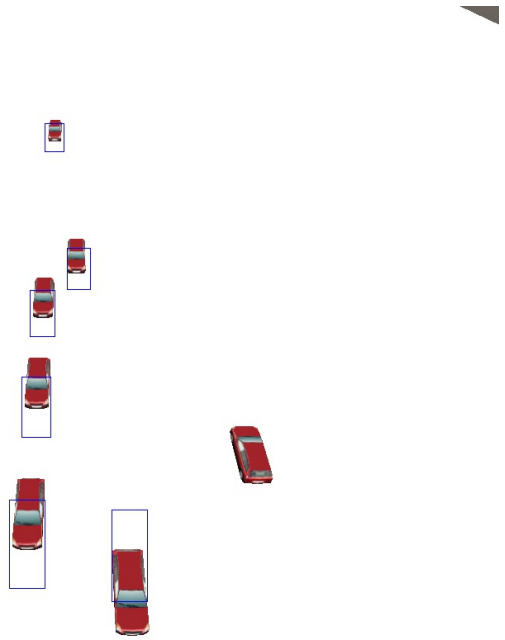

Figure 4.8: An image of simplified bounding boxes after applying Non-Maximum Suppression

4.5 Zybo Zynq-7000 Field Programmable Gate Array(FPGA)

This device is a Digilent Zybo Zynq-7000 ARM/FPGA SoC Trainer Board, and is running on Xilinux 2.0.

4.5.1 System on a Chip(SOC) and Field Programmable Gate Array(FPGA) Interaction

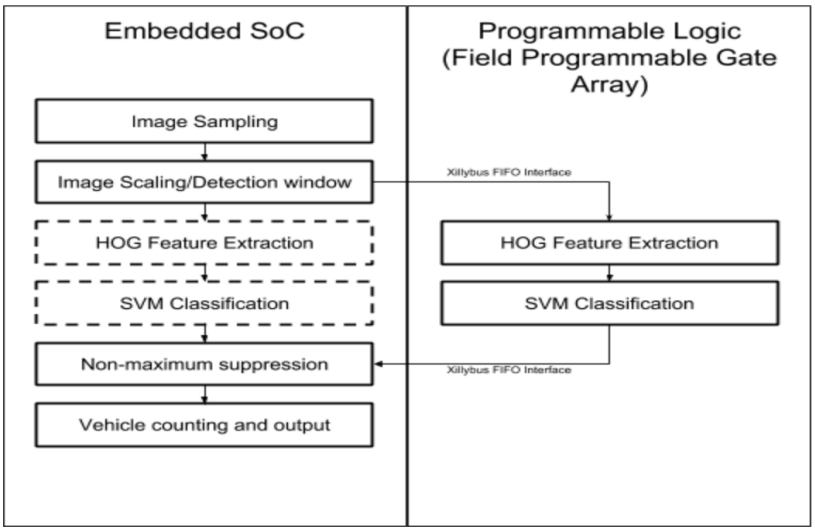

Figure 4.9: Hardware Accelerated Vehicle Counting Representation

The Zybo Zynq-7000 FPGA board comprises of two parts that are interconnected. It has both an embedded System on a Chip (SoC) and a Field Programmable Gate Array(FPGA); the System on a Chip can be likened to a computer with an operating system. The System on a Chip obtains snapshots of the traffic from the RASPI-Node from Fig. 4.1, generates sliding windows based on the optimization from Section 4.4.1. The windows are then transferred to the FPGA to detect the vehicles in the image sample. The System on a Chip then collates and counts the number of found "vehicles" from the images processed by the Field Programmable Gate Array. The System on a Chip then sends the number of cars found in the pictures to the RASPI-node.

To facilitate communication between the SoC and the FPGA, data handling is performed by the Xillybus IP Core and kernel driver. It virtualizes the Programmable Logic(PL) of the FPGA as a series of device files, that any user-space application can read and write from. On the PL, the data is presented as a FIFO stream, making it easier to model data flow operation to and from the FPGA.

4.5.2 Implementing the Image Processing Algorithm on FPGA

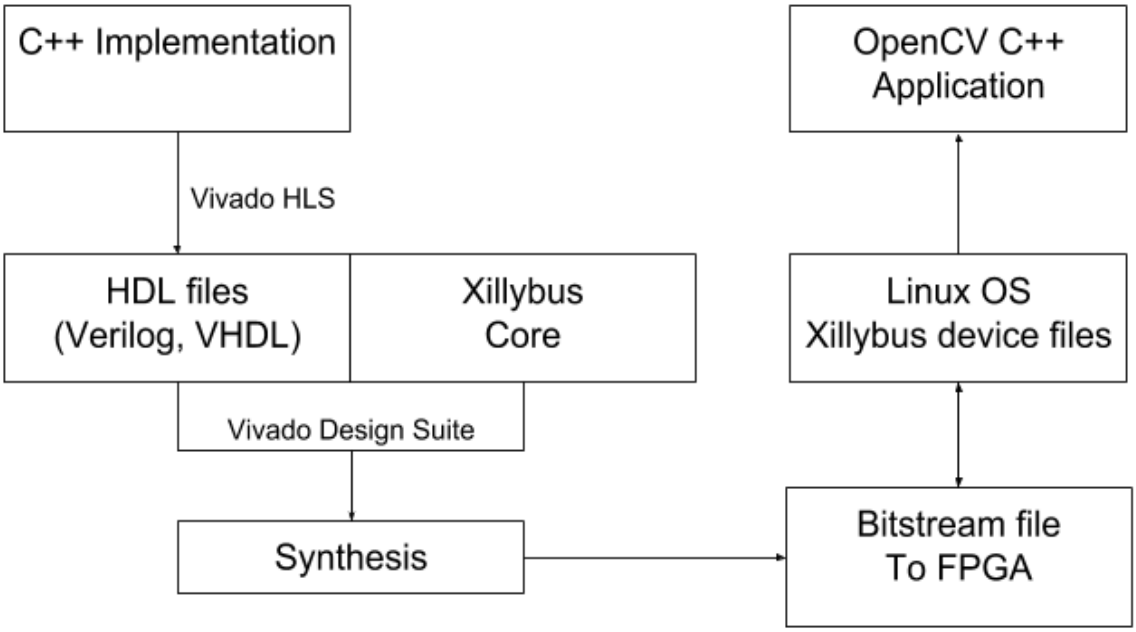

Figure 4.10: Development Flow of FPGA

Fig. 4.10 above shows the development flow of the Image processing algorithm in the Zybo SoC. The OpenCV C++ application reads the image received from the RASPI-node via TCP/IP, and passes raw image data to the FPGA via the Xillybus device files. The C++ implementation of the HOG-SVM algorithm is synthesized along with the XIllybus Core into a bitstream file, which is used to program the FPGA at boot time.

While there are many open-source implementations and libraries for the HOG algorithm, its computational complexity inhibits their use on the Zynq Zybo platform, especially when the SoC-FPGA interconnect is included in the synthesis. For example, the Xillybus Core already consumes around 30% of the Zybo's Lookup Tables (LUT). Most implementations also cache the entire image inside the FPGA during processing to increase latency, however the Zybo's 2.1 Mb of block RAM will run into problems when processing standard quality images (480p, 720p, 1080p).

	Xilinx HOG Implementation (Grayscale, 1080p)	Zynq Zybo resources (XC7Z010)
Block RAM (18k)	177	120
DSP48E Slices	36	80
Flip Flops	15205	35200
LUT	13343	17600

Table 4.1: Resource utilization summary of Xilinx's HOG implementation for FPGAs. This does not yet include image loading and SoC interconnect. [3]

To account for this, the HOG algorithm was scaled down to minimize area consumption and latency. To speed up development time, the Vivado HLS environment was used to synthesize the C++ code into integratable HDL code (Verilog, VHDL, etc.) [55]. Furthermore, the hardware accelerations used by Chen et. al. [2] were applied to further increase the minimization of resources used at the cost of some accuracy.

The final dataflow of the HOG implementation is presented below:

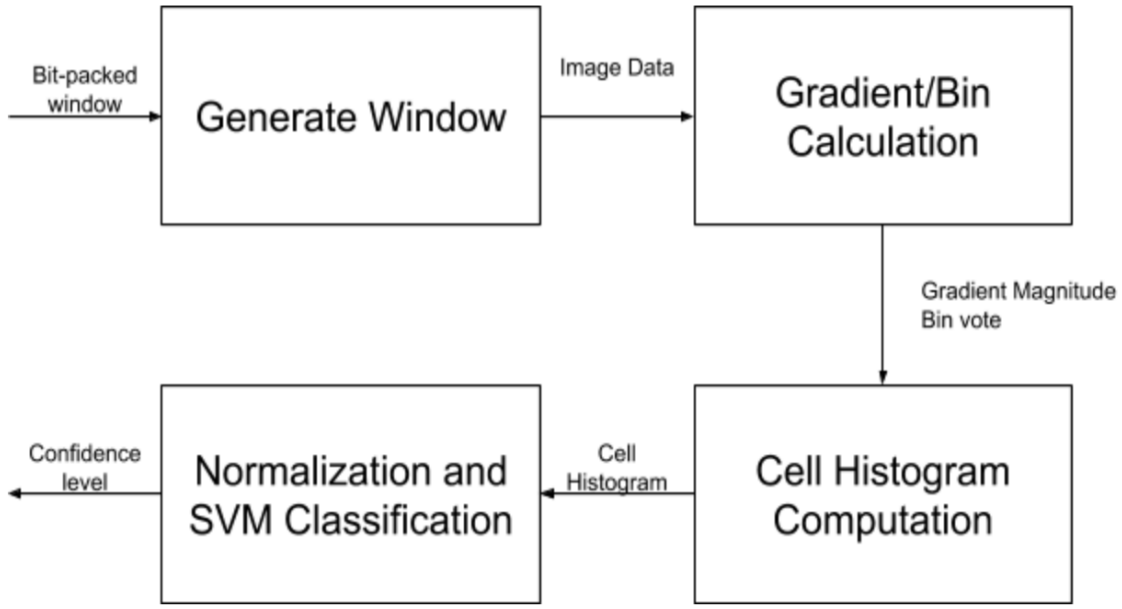

Figure 4.11: Proposed stream-based implementation of the HOG-SVM algorithm in the FPGA

4.5.3 Image Streaming and Gradient Calculation

The Xillybus device driver supports a 32-bit FIFO, which makes it ideal to pack image data with (composed of 8-bit intensity values from a grayscale image). Image data is unpacked

from the FIFO and saved to a 64x64 pixel window.

The gradient is calculated as the result of the functions:

$$dX_{x,y} = image[x+1, y] - image[x-1, y]$$

$$dY_{x,y} = image[x, y+1] - image[x, y-1]$$

Once the gradient has been calculated, magnitude and bin votes for that specific pixel can be calculated and saved. The magnitude and orientation of a gradient [dXx,y,dYx,y] can be calculated using the following:

$$mag_{x,y} = \sqrt{dX_{x,y}^2 + dY_{x,y}^2}$$

$$angle_{x,y} = \arctan(\frac{dY_{x,y}}{dX_{x,y}})$$

However, both arctangent and square root functions are resource consuming. Thus we use the method Chen, et. al. proposed the following approximations for the square root:

$$mag(x, y) = max((0.875a + 0.5b),\ a)$$

$$a = max(x,\ y)$$

$$b = min(x,\ y)$$

For the arctangent, skipping straight into the bin classification was faster; the property

$$x\tan(\theta_i) \leq y \leq x\tan(\theta_{i+1})$$

holds true for $\theta_i = 0, 10, ..80$. With the tangent function being an odd function (symmetric along the x-axis), the bin values for $\theta_i = 90, 100, ..180$ can be derived too, transforming the arc tangent into a series of multiplications with constants. To further reduce computation overhead, the tangent constants can be approximated as a series of bit shifted values, making multiplication a series of bit shifts and additions. The corresponding bin (and next nearest bin) can then be calculated by comparing dX with the succeeding approximations.

Tangent	Approximation
tan(0)	0
tan(10)	$1 >> 3 + 1 >> 4$
tan(20)	$1 >> 2 + 1 >> 3$
tan(30)	$1 >> 1 + 1 >> 4 + 1 >> 6$
tan(40)	$1 >> 1 + 1 >> 2 + 1 >> 4$
tan(50)	$1 + 1 >> 3 + 1 >> 4$
tan(60)	$1 + 1 >> 2 + 1 >> 2$
tan(70)	$2 + 1 >> 1 + 1 >> 2$
tan(80)	$5 + 1 >> 1 + 1 >> 3 + 1 >> 5$

Table 4.2: $\tan(\theta_i)$ approximate values [2]

4.5.4 Cell Histogram Computation

The 64 x 64 pixel window is then split into 8 x 8 cells, for a total of 64 cells. Each cell is its own 8 x 8 pixel region on the window, and its gradients and bin orientations are collated into a histogram, which is an array composed of the cumulative magnitudes for each bin orientation. The magnitude is weighted equally across its nearest and next nearest bin. Cell histograms are then stored into an intermediate buffer to be classified.

4.5.5 Normalization and SVM classification

A block can be represented as a 2x2 grouped cell array, with their magnitudes normalized. This requires the use of the inverse square root operator, to calculate the normalization value to be multiplied to the cell histograms.

Chen. et. al. [2] uses Carmack's fast inverse square root algorithm, with an IEEE 754 compliant number x as input:

$$fast\,inv\,sqrt(x) = y_d(\frac{2 - xy_d^2}{2})$$

$$y_d = \{(x_{binary} >> 1) - 0x5F3759DF\}_{float}$$

This generates a very close first order approximation of the inverse square root, while limiting the function into just floating point multiplication and addition. Once the inverse square root has been computed, it can be multiplied to all cell histogram bins to create the block histograms.

The n-dimensional HOG descriptor for the window is constructed by concatenating the computed block histograms together. To classify the HOG into a useful decision value, classifier

values are pre-saved onto the FPGA for immediate computation as block histograms are generated. The resulting product is accumulated and added to the classifier offset to generate the decision value output. As a general rule of thumb, a value of > 0 represents a positive window sample (vehicle), and the opposite represents a negative sample (non-vehicle).

4.6 Integration

4.6.1 Traffic Management Module

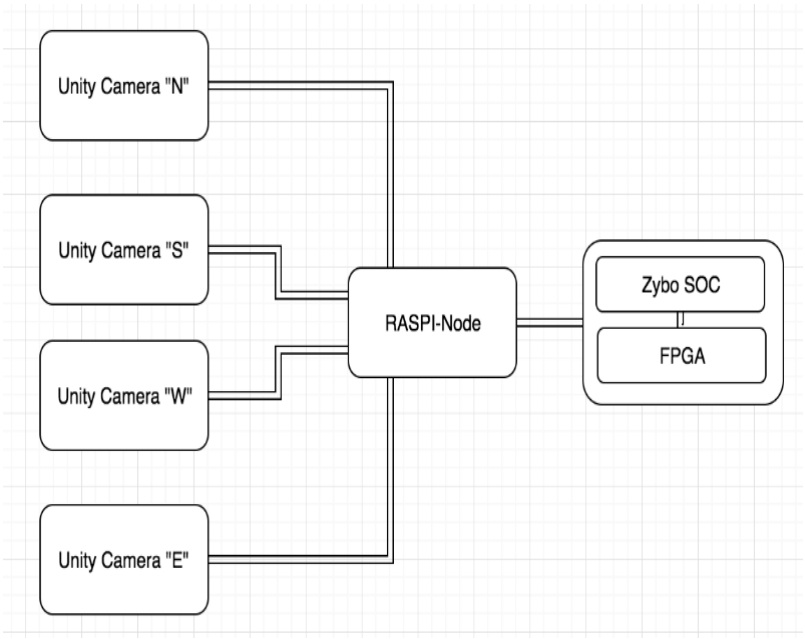

Figure 4.12: Traffic Management Module Representation

The Intelligent Traffic Management System (ITMS) would be comprised of Traffic Management Modules(TMM). There would be only one Traffic Management Module over an intersection, and a Traffic Management Module would comprise of a single-board computer or the RASPI-Node and a Zybo Zynq-7000 FPGA as shown in the diagram.The RASPI-Node will obtain snapshots of the traffic per road from the intersection simulated by the Unity engine Cameras. The SBC (RASPI-Node) would then send the four snapshots into the Zybo Zynq-7000 FPGA for vehicle counting. The FPGA would then process the sampled images and be returning the number of vehicles inside the images to the RASPI-Node it is connected to. The RASPI-Node, upon receiving the count of vehicles would calculate the appropriate traffic signal light durations. The RASPI-Nodes would also be sending data to the RASPI-Sink for analytic purposes.

4.6.2 Complete Closed Loop ITMS

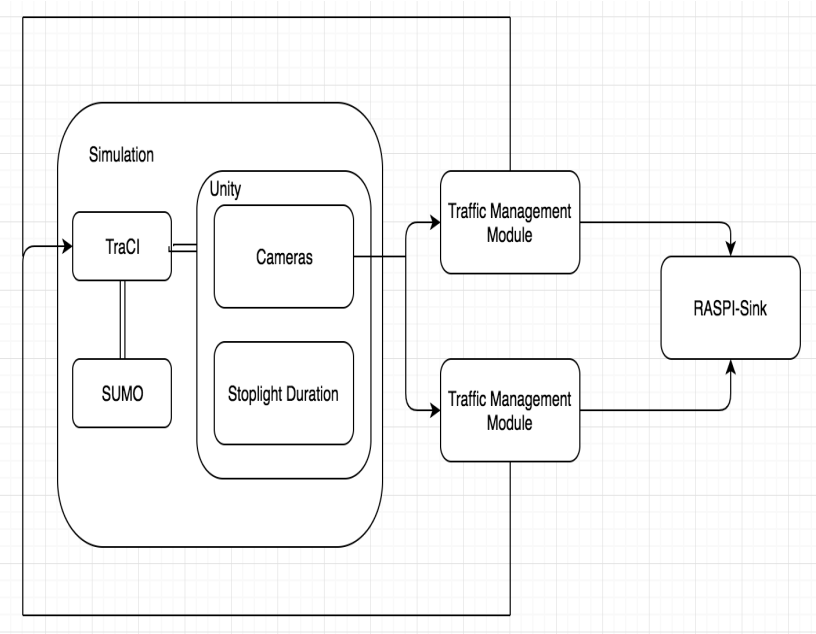

Figure 4.13: Traffic Management Module Representation

The two traffic management modules are both connected to the RASPI-Sink. Both of these would send data to the RASPI-Sink which would be analyzed by the user. The Unity cameras act as inputs to the two traffic management modules, and alternatively, the Unity stoplight duration for all the stoplights in Unity would be sourced from the output of the two traffic management modules.

Firstly, our Traffic Management Modules(TMM) will receive snapshots taken from Unity Engine cameras that are dispersed over two intersections. After some processing, the TMMs will pass out the traffic data they have collected to the RASPI-Sink. The sink(s) will now process the data and then present it to the user through the use of the terminal window provided by the operating system.

TraCI which is short for "Traffic Control Interface" allows SUMO to interface with python scripts through Transmission Control Protocol/Internet Protocol; and since we can run python inside our RASPI-Nodes, we allow the RASPI-Nodes to directly communicate with SUMO since they are on the same network. Once SUMO has received these traffic signals, it will adjust its traffic signal durations to the data received. The internal traffic simulation will now change depending on the control signals given, and then we can use the game engines to generate the snapshots to

be sent again to the TMMs which, finally, completes the feedback loop.

4.6.2.1 Ethernet and Network

The mode of connection amongst the devices uses the Ethernet protocol. For simulation purposes, this method would suffice as it provides a reliable connection amongst devices. To enable the scp and ssh methods through Paramiko, the devices must have a shared connection between each other. All of these devices would be connected to the same local area network which gives all of these devices an ip address. It is sufficient to think of ip addresses as names given to devices, so that these devices would be able to address one another. The Ethernet protocol utilizes wired connections, and that is why all of the devices would be connected to the router through physical Ethernet cables. The devices would then finally be able to communicate with each other. [56]

4.6.2.2 TCP Interaction

TCP is implemented in the system inside the python scripts. Python has the socket library which allows for easy to use socket programming functionalities which can be used to utilize TCP. There are two sides in the TCP connection, and these are the server and the client. The task of the server in this case would be to have the capability to host one or multiple clients. In this project, the Zybo Zynq-7000 works as a server awaiting the connection from a RASPI-Node acting as a client. The RASPI-Sink is also a server where it awaits the connection from the RASPI-Node.

For the Zybo Zynq-7000, the only time that the RASPI-Node will connect is when it will send the images to the Zybo. The Zybo will then return the number of vehicles found to the RASPI-Node through the connection, and then the RASPI-Node will disconnect.

The connection between the RASPI-Sink and the RASPI-Node is always maintained. Since the RASPI-Node will contFor the Zybo Zynq-7000, the only time that the RASPI-Node will connect is when it will send the images to the Zybo. The Zybo will then return the number of vehicles found to the RASPI-Node, and then the RASPI-Node will disconnect.

The connection between the RASPI-Sink and the RASPI-Node is always maintained, since the RASPI-Node will continually send information to the RASPI-Sink.

4.7 Performance Evaluation and Metrics

Data collected at SBC sinks will consist of vehicle counts from each lane and the stoplight state. This will be compared to a control system modelled after current traffic systems in Metro

Manila (i.e. static phases and fixed intervals between signals). Differences in throughput, average speed, and waiting time (all provided in SUMO) will be measured to confirm improvement.

Furthermore, the accuracy of the modules in quickly and accurately identifying vehicles will be measured. Overall frames-per-second (FPS) performance of the module will be tracked to confirm the real-time response of the system, and to determine potential bottlenecks. Bounding boxes returned by the hardware-accelerated vehicle classifier will also be cross-checked with the video input to determine precision (ratio of correct detections to number of detections made) and recall (ratio of correct detections to number of correct items). These parameters will then be compared to current software implementations of the vehicle detection algorithm to measure speedup and changes in accuracy.

Lastly, the metrics to be used for evaluating the effectiveness of the traffic control algorithms are throughput and average waiting time. Throughput will be defined as the overall number of vehicles that pass through the road system, while average waiting time will be the average time difference when a vehicle arrives on an intersection until it passes through.

Chapter 5

Results and Analysis

5.1 Preliminary Results

5.1.1 Computer Vision on RASPI and FPGA

The table below shows the datasets used for the preliminary results with their corresponding number of actual number of positives and negatives.

Video Name	Source	Description	Positives	Negatives
MVI_39031	DETRAC	Direct front view, daytime	1152	1416
MMDA_4079	MMDA	Roxas Boulevard - Quirino Avenue intersection	287	403

Table 5.1: Dataset Descriptions

The table below shows the comparison test of the FPGA versus the RASPI.

- FN = False Negatives
- FP = False Positives

FPGA							
Video Name	FN	FP	Proc Time	Prec	Recall	F1-Score	Windows/sec
MVI_39031	63	6	0.266152587	0.9945205479	0.9453125	0.9692923899	9648.60056
MMDA_4079	72	1	0.086805526	0.9953703704	0.7491289199	0.8548707753	7948.802706
RASPI							
Video Name	FN	FP	Proc Time	Prec	Recall	F1-Score	Windows/Sec
MVI_39031	37	13	3.6345863	0.9884751773	0.9678819444	0.9780701754	706.5453364
MMDA_4079	73	1	0.980910317	0.9953488372	0.7456445993	0.8525896414	703.4282218

Table 5.2: FPGA vs RASPI Comparison

The figure above shows preliminary results of the FPGA co-processing design when compared to the Raspberry Pi OpenCV implementation. Even though the hardware approximation algorithms create inconsistencies and slight inaccuracies in the design, the detection rates remain roughly the same for both the Raspberry Pi and the FPGA co-processing design.

There are some inaccuracies in the MMDA dataset sample, only detecting around 75% of vehicles on both FPGA and Raspberry Pi designs. This is caused by the more occluded nature of Philippine roads, and the quality of the CCTV camera used (dirt on the screen, moving/blurring, resolution changes).

However, the biggest gain in the results is the Windows per sec (W/s) throughput. By piping image data to the FPGA, it was able to increase throughput from around 700 W/s to 9648 W/s for the DETRAC dataset sample and 7948 W/s for the MMDA dataset sample. The difference in throughput is most likely due to the system call overhead (the OS communicates with the FPGA via read() and write() calls to the Xillybus device FIFOs).

The rest of the test dataset is still being sampled and benchmarked for performance and accuracy ratings.

5.1.2 Traffic System Simulation

The following data is obtained by running the simulation using ideal vehicle counting data which is enabled by SUMO; this means that all the vehicles in the simulation are accounted for. The vehicle seeding is done by rolling every second if a car will enter the system for 1800 seconds except for the heavy traffic scenario which was seeded for 4000s. The hit probability for the rolling will establish the traffic scenario (light, moderate, or heavy). The hit probability used is the following:

- Light: 25%
- Moderate: 50%
- Heavy: 75%

The percentages mean that for example, given a “light” traffic scenario, SUMO will only spawn vehicles into the system at a probability of 25%.

The average throughput per intersection is then plotted over time. The statically configured traffic algorithm is also displayed on the graph for reference. All graphs are generated using python’s matplotlib library. The legend for each graph is as follows:

- Red : represents a dynamic system over intersection 1

- Green : represents a dynamic system over intersection 2
- Blue : represents a static system over intersection 1
- Yellow : represents a static system over intersection 2

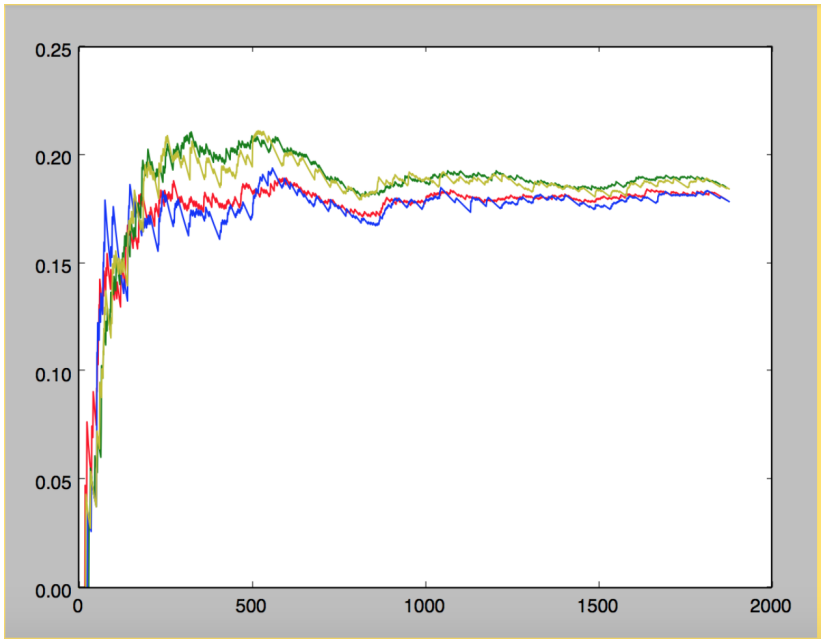

Figure 5.1: Average throughput over time with light traffic

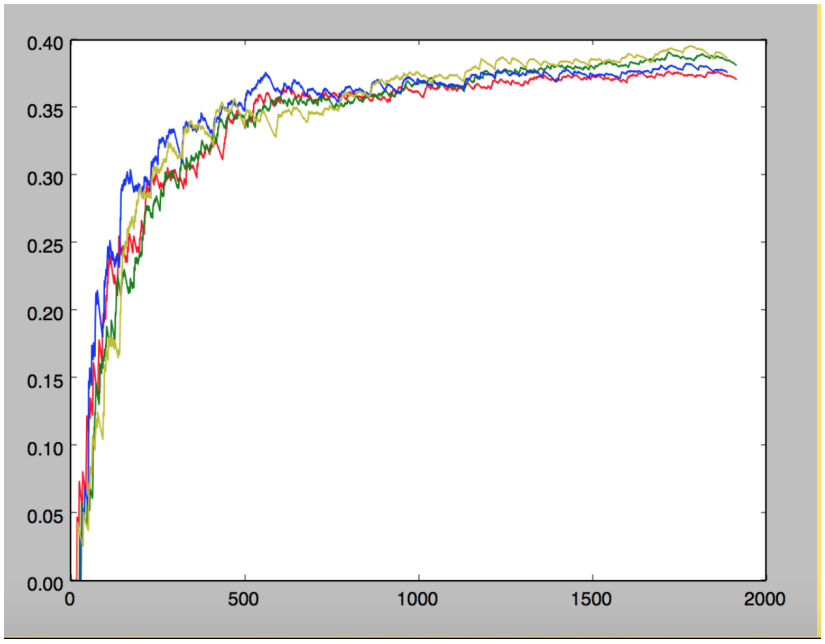

Figure 5.2: Moderate throughput over time with moderate traffic

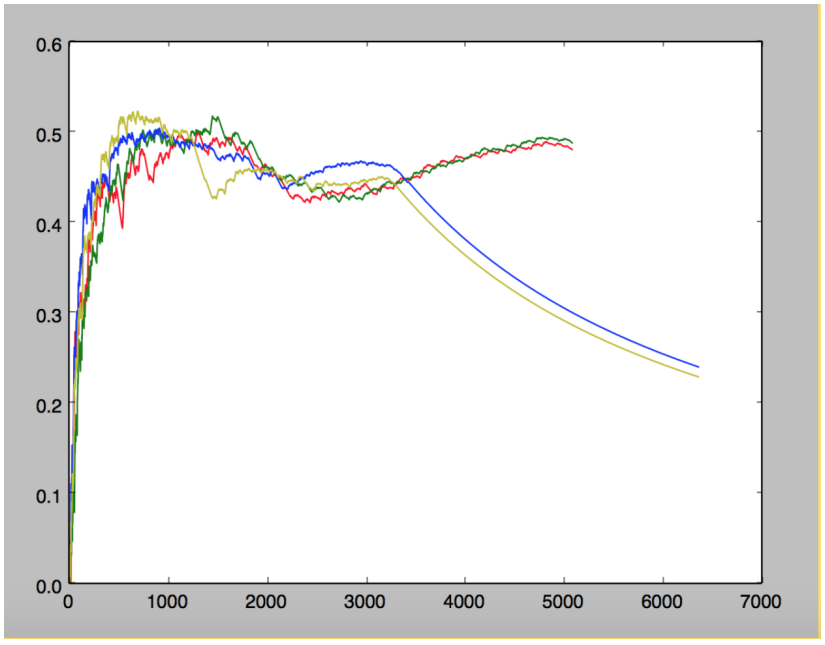

Figure 5.3: Average throughput over time with heavy traffic

From Fig 5.3, it is apparent that there is a huge decline of the average throughput for the static algorithms. This is due to the occurence of a gridlock. A gridlock is defined as the blockage

of an intersection. This results from an overwhelming number of cars that are entering one lane simultaneously which consequently will block the other lanes.

5.2 Final Results

To follow.

Chapter 6

Project Schedule and Deliverables

6.1 Gantt Chart

Deliverables	Weeks	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	Person	Color
RASPI-Node able to receive images from SUMO/Unity	2																	Timothy	
RASPI-Node sending images to FPGA	2																	Paul	
RASPI-Node receiving car count from FPGA	2																	Patrick	
RASPI-Node communication with RASPI-Sink	2																	Everyone	
SUMO Framework	2																		
Traffic Signals Interfacing	3																		
Unity Interfacing	2																		
Training Sample Dataset and Testing	2																		
Image Scaling, Sliding Window, Feature Extraction(HOG) on FPGA	2																		
Implementation of trained SVM classifier on FPGA	3																		
Midway Integration	1																		
Able to send traffic data of one intersection to RASPI-Sink	2																		
Complete operation of one intersection with ITMS Algorithm	2																		
Simultaneous operation of two intersections with ITMS Algorithm	3																		
ITMS Validation	2																		
ITMS Migration to SBC	2																		
Data Collection of RASPI-Sink	3																		
Benchmarking, optimization of TMM	2																		
Implementation of other sample CV approaches through OpenCV	3																		
Comparison of results, speedup, and results	2																		
Fullway Integration	1																		
Documentation	16																		

Table 6.1: Gantt Chart

6.2 Halfway-point Deliverables

At the 8th week of the project, the objectives accomplished should be

- The RASPI-node should be able to receive data from the Unity Engine.
- The Zybo FPGA is able to perform image processing and vehicle counting from the snapshots sent by the RASPI-node.
- The RASPI-node from TMM Module is able to send data to the RASPI-sink.

6.3 Final Deliverables

At the 16th week of the project, the objectives accomplished should be

- The proposed system would be running on the SUMO, and it would be covering two intersections.
- The system would be operating in a closed loop environment with SUMO.

Appendix A

Programs

A.1 Traffic Management Module

A.1.1 RASP-node.py

```
#!/usr/bin/env python

import sys
import paramiko
import socket
import thread
import select
import string
import os
from datetime import datetime

def startproc() :
    # Getting picture from simulator...
    try:
        start = datetime.now()
        t = paramiko.Transport((sim_hostname, scp_port))
        t.connect(username=sim_username, password=sim_password)
        sftp = paramiko.SFTPClient.from_transport(t)
        for x in range(0, 4):
            sftp.get(sim_source[x], sim_to_rasp_dest[x])

    finally:
```

```
    t.close()
    done = datetime.now()

delta = done - start
print "Time duration of PROCESS Simulator to Raspi Photo SCP: " , delta.
    microseconds, "us"


# Sending picture to Zybo
try:
    start = datetime.now()
    t = paramiko.Transport((zybo_hostname, scp_port))
    t.connect(username=zybo_username, password=zybo_password)
    sftp = paramiko.SFTPClient.from_transport(t)
    for x in range(0, 4):
        sftp.put(rasp_source[x] , rasp_to_zybo_dest[x])

finally:
    t.close()
    done = datetime.now()
delta = done - start
print "Time duration of PROCESS Raspi to Zybo Photo SCP: " , delta.
    microseconds, "us"


# Send "startcv" command to zybo
try:
    try:
        inputs = [timsocket,sys.stdin]
        start = datetime.now()
        message = "Raspi: startcv"
        timsocket.send(message)
        while(received == 0):
            read,write,exception = select.select(inputs,[],[])
            for s in read:
                if s == timsocket:
                    data=timsocket.recv(4096)
                    if not data:
```

```
                            print "Connection closed!"
                            timsocket.close()
                            break
                        else:
                            print data
                            stringreceived = data.strip().split()
                            numberofcarsfound = stringreceived[1]
                            sys.stdout.flush()
                            done = datetime.now()
                            delta = done - start
                            print "Time duration of PROCESS Recv from Zybo to
                                Raspi: " , delta.microseconds/1000, "ms"
                            return numberofcarsfound
            except KeyboardInterrupt:
                timsocket.close()
                sys.exit()
        except KeyboardInterrupt:
            timsocket.close()
            sys.exit()

sink_hostname = "192.168.0.27"
sink_username = "timothychua"
sink_password = "caputdraconis"

scp_port = 22
# sim_hostname = "192.168.0.16"
# sim_username = "paulgrantilaga"
# sim_password = "password"
# sim_source = "/Users/paulgrantilaga/Documents/itms/ITMS-Unity/timpic-.png"
# sim_to_rasp_dest = "screenshot.jpg"
sim_hostname = "192.168.0.32"
sim_username = "timothychua"
sim_password = "caputdraconis"
sim_source = [ "/Users/timothychua/Desktop/screenshot1.jpg", "/Users/
    timothychua/Desktop/screenshot2.jpg",
"/Users/timothychua/Desktop/screenshot3.jpg", "/Users/timothychua/Desktop/
    screenshot4.jpg"]
sim_to_rasp_dest = [ "screenshot1.jpg", "screenshot2.jpg", "screenshot3.jpg",
```

```
    "screenshot4.jpg" ]



zybo_hostname = "192.168.0.32"
zybo_username = "root"
zybo_password = "alohamora"
rasp_source = [ "screenshot1.jpg", "screenshot2.jpg", "screenshot3.jpg", "
    screenshot4.jpg" ]
rasp_to_zybo_dest = [ "/root/Desktop/screenshot1.jpg", "/root/Desktop/
    screenshot2.jpg", "/root/Desktop/screenshot3.jpg"
, "/root/Desktop/screenshot4.jpg" ]



print 'Zybo SECTION'
timsocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
print 'Socket Created!'
port = int(raw_input('Port Number: '))
timsocket.connect((zybo_hostname,port))
print 'Connected!'
received = 0


messagelist = []
print 'Raspi Sink SECTION'
chuasocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
print 'We will be using your localhost for listening but what port should we
    use?'
port = int(raw_input('Port: '))
chuasocket.bind(('',port))
print 'Listening for a connection....'
inputs=[sys.stdin,chuasocket]
outputs=[]
chuasocket.listen(1)
peerconn,ipaddr = chuasocket.accept()
print 'Raspi Sink connected with IPAddr,Port:', ipaddr
inputs.append(peerconn)
```

```
outputs.append(peerconn)
while(1):
    readable,writable,exception = select.select(inputs,outputs,[])
    for s in readable:
        if s == peerconn:
            data=s.recv(4096)
            if not data:
                inputs.remove(s)
                outputs.remove(s)
                s.close()
                chuasocket.listen(1)
                peerconn,ipaddr = chuasocket.accept()
                print 'Raspi Sink connected with IPAddr,Port:', ipaddr
                inputs.append(peerconn)
                outputs.append(peerconn)
                break
            else:
                print data.strip()
                sys.stdout.flush()
                stringreceived = data.strip().split()
                if stringreceived[0] == "startproc":
                    peerconn.send(str(startproc()))
        elif s == sys.stdin:
            message = sys.stdin.readline()
            message = message.strip()
            messagelist.append(message)
```

A.1.2 RASP-sink.py

```
import sys
import paramiko
import socket
import thread
import select
import string
import os
from datetime import datetime

def beginTMM():
```

```
    try:
        while(1):
            timsocket.send("startproc")
            try:
                inputs = [timsocket,sys.stdin]
                read,write,exception = select.select(inputs,[],[])
                for s in read:
                    if s == timsocket:
                        data=timsocket.recv(4096)
                        if not data:
                            print "Connection closed!"
                            timsocket.close()
                            breakloop=1
                            break
                        else:
                            return data
            except KeyboardInterrupt:
                timsocket.close()
                sys.exit()
    except KeyboardInterrupt:
        timsocket.close()
        sys.exit()


timsocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
print 'Socket Created!'
# print 'Please input the IPaddr and Port Number:        '
ipaddr = "192.168.0.32"
port = 5000
timsocket.connect((ipaddr,port))
print 'Connected!'
while(1):
    answer = raw_input("Start process? ")
    if answer == "startproc" :
        start = datetime.now()
        print "The number of cars at the moment is: " + str(beginTMM())
        done = datetime.now()
        delta = done - start
```

```
        print "Time duration of TMM Process: " , delta.microseconds/1000, "ms
            "
```

A.1.3 Zybo Zynq-7000 FPGA

```
import socket
import thread
import select
import string
import sys
import time



messagelist = []
timsocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
print 'We will be using your localhost for listening but what port should we use?'
port = int(raw_input('Port: '))
timsocket.bind(('',port))
print 'Listening for a connection....'
inputs=[timsocket,sys.stdin]
outputs=[]
timsocket.listen(1)
peerconn,ipaddr = timsocket.accept()
inputs.append(peerconn)
outputs.append(peerconn)
while(1):
        readable,writable,exception = select.select(inputs,outputs,[])
        for s in readable:
                if s == peerconn:
                        data=s.recv(4096)
                        if not data:
                                inputs.remove(s)
                                outputs.remove(s)
                                s.close()
                                timsocket.listen(1)
```

```
                        peerconn,ipaddr = chuasocket.accept()
                        print 'Raspi Node connected with IPAddr,Port:', ipa
                        inputs.append(peerconn)
                        outputs.append(peerconn)
                        break
                else:
                        print data.strip()
                        sys.stdout.flush()
                        stringreceived = data.strip().split()
                        if stringreceived[1] == "startcv":
                                # Do Computer Vision things.....
                                # Continue Doing Computer Vision things...
                                # Detect number of cars!
                                # For example number of cars = 2
                                # time.sleep(3)
                                number_of_cars = 2
                                message = "Zybo: " + str(number_of_cars) +
                                messagelist.append(message)
        for s in writable:
                if s == peerconn:
                        if(messagelist):
                                s.send(messagelist[0])
                                messagelist.pop(0)
```

A.2 Simulation

A.2.1 sumo.py

```
#!/usr/bin/python

from __future__ import absolute_import
from __future__ import print_function
from collections import defaultdict

import os
import sys
```

```
import optparse
import subprocess
import string
import random
import string
import itertools
import socket
import time
import select

# import TraCI
sys.path.append(os.path.join("/opt/local/Library/Frameworks/Python.framework/
    Versions/2.7/lib/python2.7/site-packages/"))
import traci

# options handler
def get_options():
    optParser = optparse.OptionParser()

    optParser.add_option("-x", "--execute", action="store",
                         dest="execute", default=False, help="if all filenames
                             are the same")
    optParser.add_option("-v", "--verbose", action="store_true",
                         dest="verbose", default=False, help="verbose")
    optParser.add_option("-m", "--map", action="store",
                         dest="map_file", help="net file at sumo/maps")
    optParser.add_option("-c", "--v_file", action="store",
                         dest="vehicle_config", help="vehicle config at sumo/
                             config/vehicles")
    optParser.add_option("-d", "--dont_generate", action="store_false",
                         dest="generate", default=True, help="do not generate
                             route file")
    optParser.add_option("-r", "--route_file", action="store",
                         dest="route_file", help="route file to be read or
                             written on sumo/routes")
    optParser.add_option("-s", "--sumo_config", action="store",
                         dest="sumo_config", help="sumo config file on sumo/
                             config/sumo")
```

```
    options, args = optParser.parse_args()

    return options

class Graph:
    def __init__(self):
        self.nodes = set()
        self.edges = defaultdict(list)
        self.distances = {}

    def add_node(self, value):
        self.nodes.add(value)

    def add_edge(self, from_node, to_node, distance):
        self.edges[from_node].append(to_node)
        self.edges[to_node].append(from_node)
        self.distances[(from_node, to_node)] = distance


def dijsktra(graph, initial):

    visited = {initial: 0}
    path = {}

    nodes = set(graph.nodes)

    while nodes:
        min_node = None
        for node in nodes:
            if node in visited:
                if min_node is None:
                    min_node = node
                elif visited[node] < visited[min_node]:
                    min_node = node

        if min_node is None:
            break
```

```
            nodes.remove(min_node)
            current_weight = visited[min_node]

            for edge in graph.edges[min_node]:
                weight = current_weight + graph.distances[(min_node, edge)]
                if edge not in visited or weight < visited[edge]:
                    visited[edge] = weight
                    path[edge] = min_node

    return path

def router(path_set, initial, end):
    path = ""
    current = end
    while True:
        path = current + path
        current = path_set[current]
        path = " " + current + path
        if (current == initial):
            break
    path = path.strip()
    path = "<route id=\"" + initial + end + "\" edges=\"" + path + "\" />"
    path_name = initial + end
    return path, path_name

def generate_routefile(route_file, vehicle_config, map_file):
    vehicles = []
    with open(vehicle_config, "r") as vehicle:
        buf = vehicle.readline()
        while (buf):
            if buf.strip() == "{":
                instance = [""] * 3
                buf = vehicle.readline().strip()
                while buf != "}":
                    buf2 = buf.split("=")
                    if (buf2[0] == "name"):
                        instance[0] = buf2[1]
```

```
                elif (buf2[0] == "chance"):
                    instance[1] = buf2[1]
                else:
                    instance[2] = instance[2] + buf + ' '
                buf = vehicle.readline().strip()
            instance[2] = instance[2].strip()
            vehicles.append(instance)
        buf = vehicle.readline()

graph = Graph()
with open(map_file, "r") as net:
    buf = net.readline().strip()
    while (buf):
        buf = buf.strip()
        if (buf.find("internal") < 0 and buf.find("edge") >= 0):
            if (not buf == "</edge>"):
                buf = buf.split(" ")
                edge = buf[1][3:].split("\"")[1]
                nodeFrom = buf[2][6:].split("\"")[0]
                nodeTo = buf[3][4:].split("\"")[0]
                graph.add_node(nodeFrom)
                graph.add_node(nodeTo)
                buf = net.readline().strip().split(" ")
                graph.add_edge(nodeFrom, nodeTo, float(buf[4][8:].split
                    ("\"")[0]))
        buf = net.readline()

nameRoutes = []
with open(route_file, "w") as routes:
    print("<routes>", file=routes)
    for i in vehicles:
        print("\t<vType id=" + i[0] + " " + i[2] + "/>", file=routes)
    for start in graph.nodes:
        try:
            int(start)
            continue
        except:
            path = dijsktra(graph, start)
```

```
                for end in graph.nodes:
                    try:
                        int(end)
                        continue
                    except:
                        if (start != end):
                            r = router (path, start, end)
                            print("\t" + r[0], file=routes)
                            nameRoutes.append(r[1])

        random.seed(42)
        N = 3600
        p = 0.1
        vehNr = 0
        for i in range(N):
            proc = random.uniform(0, 1)
            if proc < p:
                print('\t<vehicle id="%i" type="car" route="%s" depart="%i"
                    />' % (
                    vehNr, nameRoutes[int(round((len(nameRoutes) - 1) *
                        random.uniform(0, 1)))], i), file=routes)
                vehNr += 1
        print("</routes>", file=routes)

# run the simulation
def run():
    ids = []
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect(("localhost", 10000))
    print ("simulating...")
    while traci.simulation.getMinExpectedNumber() > 0:
        begin = time.time()
        readable, writeable, exceptional = select.select([s], [], [], 0.001)
        if s in readable:
            print("recv: " + s.recv(1024))
        else:
            traci.simulationStep()
            s.send("begin\n")
```

```
            ids = traci.vehicle.getIDList()
            for vehid in ids:
                s.send(str(vehid) + " " + str(traci.vehicle.getPosition(vehid
                    )) +
                        " " + traci.vehicle.getTypeID(vehid) + " " + str(
                            traci.vehicle.getAngle(vehid)) + "\n") # id(x,y)
            s.send("end\n")
            end = time.time()
            print ("step: " + str(end - begin) + "s")
    s.send("simulation ended\n")
    traci.close()
    sys.stdout.flush()

if __name__ == "__main__":

    # options processing

    options = get_options()
    if (options.execute):
        sumo_config = "sumo/config/sumo/" + options.execute + ".sumocfg"
        data_out = "sumo/data/" + options.execute + ".xml"
        vehicle_config = "sumo/config/vehicles/" + options.execute + ".txt"
        route_file = "sumo/routes/" + options.execute + ".rou.xml"
        map_file = "sumo/maps/" + options.execute + ".net.xml"
        if (options.map_file):
            map_file = "sumo/maps/" + options.map_file + ".net.xml"
        if (options.vehicle_config):
            vehicle_config = "sumo/config/vehicles/" + options.vehicle_config
                + ".txt"
        if (options.route_file):
            route_file = "sumo/routes/" + options.route_file + ".rou.xml"
        if (options.sumo_config):
            sumo_config = "sumo/config/sumo/" + options.sumo_config + ".
                sumocfg"

    if (options.verbose):
        print("sumo_config: %s" % (sumo_config))
        print("data_out: %s" % (data_out))
```

```
        print("vehicle_config: %s" % (vehicle_config))
        print("route_file: %s" % (route_file))
        print("map_file: %s" % (map_file))
    if (options.generate):
        try:
            generate_routefile(route_file, vehicle_config, map_file)
            if (options.verbose):
                print("route file generated")
        except:
            print ("route file generation failed")
    else:
        if (options.verbose):
                print("skip route file generation")
    if (options.verbose):
        print("running sumo through traci")
    traci.start(["/opt/local/bin/sumo-gui", "-c", sumo_config,
                            "--tripinfo-output", data_out])
    run()
```

A.2.2 sim.py

```
#!/usr/bin/python

import socket
import sys
import Queue
import select

port = input("Port: ")
listen = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
listen.bind(('', port))
listen.listen(1)

inputs = [listen, sys.stdin]
outputs = []

while 1:
```

```
readable, writeable, exceptional = \
    select.select(inputs, outputs, inputs)

for s in readable:
    if s is listen:
        connection, client_address = s.accept()
        print client_address[0] + ' (' + \
            str(client_address[1]) + ") connected"
        connection.setblocking(0)
        inputs.append(connection)
        outputs.append(connection)
    elif s is sys.stdin:
        if len(inputs) == 3:
            buf = sys.stdin.readline()
            inputs[2].send(buf)
        else:
            buf = sys.stdin.readline()
            print "Nobody can hear you."
    else:
        data = s.recv(1024)
        if data != "":
            while True:
                try:
                    inputs[2].send(data)
                except:
                    continue
                else:
                    break
        elif s in outputs:
            outputs.remove(s)
            inputs.remove(s)
            s.close()
            print "He left."
```

A.2.3 NetworkInterface.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;
using System.IO;
using System.Net.Sockets;

public class NetworkInterface : MonoBehaviour
{
    public String host = "localhost";
    public Int32 port = 10000;
    public GameObject car;

    public class Vehicle {
        public int id;
        public float x;
        public float y;
        public string type;
        public float angle;

        public Vehicle (int a, float b, float c, string d, float e) {
            id = a;
            x = b;
            y = c;
            type = d;
            angle = e;
        }
    }

    internal Boolean socket_ready = false;
    internal String input_buffer = "";
    Dictionary <int, Vehicle> vehicles = new Dictionary<int, Vehicle>();
    Dictionary <int, GameObject> inGame = new Dictionary<int, GameObject>();
    TcpClient tcp_socket;
    NetworkStream net_stream;
    StreamWriter socket_writer;
```

```
StreamReader socket_reader;

void Start()
{
    StartCoroutine(UpdateVehicles());
    // net_stream.ReadTimeout = 1;
}

void Update()
{
    GameObject inst;
    foreach (KeyValuePair<int, GameObject> temp in inGame)
    {
        if (!vehicles.ContainsKey(temp.Key))
        {
            Destroy(temp.Value);
            inGame.Remove(temp.Key);
        }
    }
    foreach(KeyValuePair<int, Vehicle> temp in vehicles)
    {
        if (!inGame.ContainsKey(temp.Key))
        {
            inst = Instantiate(car, new Vector3(temp.Value.x, 0.5f, temp.
                Value.y), Quaternion.Euler(-89.98f, temp.Value.angle, 0));
            inGame.Add(temp.Key, inst);
        }
        else
        {
            inGame[temp.Key].transform.position = new Vector3(temp.Value.
                x, 0.5f, temp.Value.y);
            inGame[temp.Key].transform.rotation = Quaternion.Euler(-89.98
                f, temp.Value.angle, 0);
        }
    }
}
```

```
void Awake()
{
    // Application.targetFrameRate = 25;
    setupSocket();
}

void OnApplicationQuit()
{
    closeSocket();
}

public void setupSocket()
{
    try
    {
                    Debug.Log("connecting");
        tcp_socket = new TcpClient(host, port);

        net_stream = tcp_socket.GetStream();
        // tcp_socket.ReceiveTimeout = 1;
        socket_writer = new StreamWriter(net_stream);
        socket_reader = new StreamReader(net_stream);

        socket_ready = true;
                    Debug.Log("done");
    }
    catch (Exception e)
    {
            // Something went wrong
        Debug.Log("Socket error: " + e);
    }
}

public String readSocket()
{
    if (!socket_ready)
    {
        return "";
```

```
    }

    if (net_stream.DataAvailable)
    {
        try
        {
            return socket_reader.ReadLine();
        }
        catch (Exception e)
        {
            // Something went wrong
            Debug.Log("Socket error: " + e);
        }
    }
    return "";
}

public void closeSocket()
{
    if (!socket_ready)
        return;

    socket_writer.Close();
    socket_reader.Close();
    tcp_socket.Close();
    socket_ready = false;
}

private IEnumerator UpdateVehicles()
{
    string received_data;
    string type;
    int vID;
    float x, y, angle;
    while (true)
    {
        received_data = readSocket();
        if (received_data != "" && received_data == "begin")
```

```
            {
                // print(received_data);
                vehicles.Clear();
                while (received_data != "end")
                {
                    if (received_data != "" && received_data != "begin")
                    {
                        // print(received_data);
                        string[] tokens = received_data.Split(' ');
                        vID = int.Parse(tokens[0]);
                        x = float.Parse(tokens[1].Substring(1, tokens[1].
                            Length - 2));
                        y = float.Parse(tokens[2].Substring(0, tokens[2].
                            Length - 2));
                        type = tokens[3];
                        angle = float.Parse(tokens[4]);
                        Vehicle temp = new Vehicle(vID, x, y, type, angle);
                        vehicles.Add(temp.id, temp);
                        received_data = readSocket();
                    }
                    else
                    {
                        received_data = readSocket();
                    }
                }

            }
            yield return null;
        }
    }
}
```

A.3 Computer Vision

A.3.1 Benchmarking

A.3.1.1 Accuracy

```
#include <opencv2/opencv.hpp>
```

```
#include <iostream>
#include <fstream>

using namespace cv;
using namespace std;

int main(int argc, char** argv)
{
        const char* keys =
        {
                "{help h|       | show help message}"
                "{dir   |       | path of directory containing images to test
                   }"
                "{hog   |       | hog_descriptor used}"
                "{svm   |       | hog_descriptor used}"
                "{out   |out.txt| textfile to write to}"
                "{g     |false  | use gpu}"
        };
        CommandLineParser parser(argc, argv, keys);
        if (argc < 2 || parser.has("help") || parser.has("h") || !parser.has
           ("dir") || !parser.has("hog") || !parser.has("out"))
        {
                //print help
                parser.printMessage();
                exit(0);
        }
        String directory = parser.get<String>("dir");
        String hog_file = parser.get<String>("hog");
        String output_file = parser.get<String>("out");
        bool use_gpu = parser.get<bool>("g");
        cv::HOGDescriptor cpu_hog;
        cv::Ptr<cv::cuda::HOG> gpu_hog = Algorithm::load<cv::cuda::HOG>(
           hog_file);
        cpu_hog.load(hog_file);
        ofstream output;
        output.open(output_file);
        vector<String> file_names;
        glob(directory, file_names);
```

```
output << "Starting testing for directory " << directory << endl;
cuda::GpuMat gpu_image;
int count_images = 0, detected_vehicles = 0;
for (int i = 0; i < file_names.size(); i++)
{
        Mat image = imread(file_names[i]);
        Mat gray_image;
        cvtColor(image, gray_image, COLOR_BGR2GRAY);
        if (image.empty())
        {
                //continue
                continue;
        }
        vector<Point> detections;
        vector<Rect> multi_detections;
        vector<double> found_weights;
        if (use_gpu)
        {
                gpu_image.upload(gray_image);
                gpu_hog->detect(gpu_image, detections, &found_weights
                    );
                //gpu_hog->detectMultiScale(gray_image,
                    multi_detections, &found_weights);
        }
        else
                //cpu_hog.detectMultiScale(gray_image,
                    multi_detections, found_weights);
                cpu_hog.detect(gray_image, detections, found_weights)
                    ;
        output << "Processing " << file_names[i] << " : ";
        output << detections.size() << " vehicles found." << endl;
        detected_vehicles += detections.size() == 0 ? 0 : 1;
        count_images++;
}

cout << "Detected " << detected_vehicles << " out of " <<
   count_images << " processed." << endl;
cout << "HIT RATE: " << (double)detected_vehicles / count_images <<
```

```
            endl;

        output.close();
    return 0;
}
```

A.3.1.2 Frames per Second

```
#include <opencv2/opencv.hpp>
#include <iostream>
#include <fstream>


using namespace cv;
using namespace std;


int main(int argc, char** argv)
{
        const char* keys =
        {
                "{help h|       | show help message}"
                "{dir   |       | path of directory containing images to test
                   }"
                "{hog   |       | hog_descriptor used}"
                "{svm   |       | hog_descriptor used}"
                "{out   |out.txt| textfile to write to}"
                "{g     |false  | use gpu}"
        };


        CommandLineParser parser(argc, argv, keys);


        if (argc < 2 || parser.has("help") || parser.has("h") || !parser.has
           ("dir") || !parser.has("hog") || !parser.has("out"))
        {
                //print help
                parser.printMessage();
```

```
        exit(0);
}


String directory = parser.get<String>("dir");
String hog_file = parser.get<String>("hog");
String output_file = parser.get<String>("out");
bool use_gpu = parser.get<bool>("g");


cv::HOGDescriptor cpu_hog;
cv::Ptr<cv::cuda::HOG> gpu_hog = cv::cuda::HOG::create(Size(64,64),
    Size(16,16), Size(8,8), Size(8,8), 9);
cpu_hog.load(hog_file);


std::vector<float> svm_model;
svm_model = cpu_hog.svmDetector;
gpu_hog->setSVMDetector(svm_model);




ofstream output;
output.open(output_file);


vector<String> file_names;
glob(directory, file_names);


output << "Starting testing for directory " << directory << endl;
cuda::GpuMat gpu_image;
int count_images = 0, detected_vehicles = 0;
double seconds = 0;
for (int i = 0; i < file_names.size(); i++)
{
        Mat image = imread(file_names[i]);
```

```
Mat gray_image;
cvtColor(image, gray_image, COLOR_BGR2GRAY);


if (image.empty())
{
        //continue
        continue;
}


vector<Point> detections;
vector<Rect> multi_detections;
vector<double> found_weights;


auto start = cv::getTickCount();
if (use_gpu)
{
        gpu_image.upload(gray_image);
        gpu_hog->setNumLevels(13);
        gpu_hog->setHitThreshold(0);
        gpu_hog->setWinStride(Size(8,8));
        gpu_hog->setScaleFactor(1.05);
        gpu_hog->setGroupThreshold(8);
        gpu_hog->detectMultiScale(gpu_image, multi_detections
            );
}
else
{
        cpu_hog.nlevels = 13;
        cpu_hog.detectMultiScale(gray_image, multi_detections
            , 0, Size(8,8), Size(0,0), 1.05);
}
auto end = cv::getTickCount();


seconds += end - start;
```

```
            //output << "Processing " << file_names[i] << " : ";
            //output << detections.size() << " vehicles found." << endl;
            detected_vehicles += multi_detections.size();
            count_images++;
        }


        seconds /= cv::getTickFrequency();


        cout << "Detected " << detected_vehicles << " out of " <<
            count_images << " images processed." << endl;
        cout << "AVG FPS: " << (double) count_images/seconds << endl;


        output.close();
        return 0;
}
```

A.3.2 Training

A.3.2.1 Negative Sampler

```
#include "opencv2/imgproc.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/ml.hpp"
#include "opencv2/objdetect.hpp"
#include "opencv2/core.hpp"
#include "opencv2/opencv.hpp"

#include <iostream>
#include <string>

using namespace cv;
using namespace cv::ml;
using namespace std;
```

```
void sample_neg(const String & dirname, vector< Mat > & neg_lst, const Size &
    size)
{
        vector< String > files;
        glob(dirname, files);
        cout << files.size() << " images found" << endl;

        Rect box;
        box.width = size.width;
        box.height = size.height;

        const int size_x = box.width;
        const int size_y = box.height;

        srand((unsigned int)time(NULL));

        Mat image;
        for (size_t i = 0; i < files.size(); i++)
        {
                image = imread(files[i]);
                if (image.cols >= box.width && image.rows >= box.height)
                {
                        Mat temp = image;
                        // resize(image, temp, Size(1920, 1080));
                        box.x = rand() % (temp.cols - size_x);
                        box.y = rand() % (temp.rows - size_y);
                        Mat roi = temp(box);

                        neg_lst.push_back(roi.clone());

                        imshow("Sampling...", roi);
                        waitKey(1);
                }
        }
}

void load_images(const String & dirname, vector< Mat > & img_lst, bool
    showImages = false)
```

```
{
        vector< String > files;
        glob(dirname, files);

        for (size_t i = 0; i < files.size(); ++i)
        {
                Mat img = imread(files[i]); // load the image
                if (img.empty())            // invalid image, skip it.
                {
                        cout << files[i] << " is invalid!" << endl;
                        continue;
                }
                ;
                Mat resized_img;
                resize(img, resized_img, Size(96, 96), 0, 0,
                   INTER_LINEAR_EXACT);

                if (showImages)
                {
                        imshow("image", img);
                        waitKey(1);
                }
                img_lst.push_back(img);
        }
}

int main(int argc, char **argv)
{
        const char* keys =
        {
                "{help h|       | show help message}"
                "{dir    |       | path of directory containing images to
                   sample}"
                "{dw     |       | width of the sample window}"
                "{dh     |       | height of the sample window}"
                "{res    |       | folder of output samples}"
        };
```

```
CommandLineParser parser(argc, argv, keys);

if (argc < 2 || parser.has("help") || !parser.has("dw") || !parser.
    has("dh") || !parser.has("res") || !parser.has("dir"))
{
        parser.printMessage();
        cout << argv[0] << " - Samples a random window within for
            every frame in the folder/video specified, and stores it
            in the destination folder." << endl;
        cout << "Example Usage: " << argv[0] << " -dir=C:/path/to/
            input/folder -dw=64 -dh=64 -res=C:/path/to/output/folder"
            << endl;
        exit(0);
}

String image_dir = parser.get< String >("dir");
String out_dir = parser.get< String >("res");
int win_height = parser.get< int >("dh");
int win_width = parser.get< int >("dw");

vector< Mat > pos_lst, full_neg_lst, neg_lst, gradient_lst;
vector< int > labels;

cout << "Starting sampling at " << image_dir << "..." << endl;
sample_neg(image_dir, neg_lst, Size(win_width, win_height));

for (int i = 0; i < neg_lst.size(); i++)
{
        string dir = out_dir + "/file_";
        dir += to_string(i + 1) + ".png";

        imwrite(dir, neg_lst[i]);
        //cout << dir << endl;
}

cout << "Sampling finished." << endl << "Please check the output
    folder: " << out_dir << endl;
return 0;
```

```
}
```

A.3.2.2 OpenCV

```
#include "opencv2/imgproc.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/ml.hpp"
#include "opencv2/objdetect.hpp"
#include <iostream>
#include <time.h>

using namespace cv;
using namespace cv::ml;
using namespace std;


vector< float > get_svm_detector(const Ptr< SVM >& svm);
void convert_to_ml(const std::vector< Mat > & train_samples, Mat& trainData);
void load_images(const String & dirname, vector< Mat > & img_lst, bool
    showImages);
void sample_neg(const vector< Mat > & full_neg_lst, vector< Mat > & neg_lst,
    const Size & size);
void computeHOGs(const Size wsize, const vector< Mat > & img_lst, vector< Mat
    > & gradient_lst, bool use_flip);
void test_trained_detector(String obj_det_filename, String test_dir, String
    videofilename);

vector< float > get_svm_detector(const Ptr< SVM >& svm)
{
        // get the support vectors
        Mat sv = svm->getSupportVectors();
        const int sv_total = sv.rows;
        // get the decision function
        Mat alpha, svidx;
        double rho = svm->getDecisionFunction(0, alpha, svidx);
        CV_Assert(alpha.total() == 1 && svidx.total() == 1 && sv_total == 1);
        CV_Assert((alpha.type() == CV_64F && alpha.at<double>(0) == 1.) ||
                (alpha.type() == CV_32F && alpha.at<float>(0) == 1.f));
        CV_Assert(sv.type() == CV_32F);
```

```
        vector< float > hog_detector(sv.cols + 1);
        memcpy(&hog_detector[0], sv.ptr(), sv.cols * sizeof(hog_detector[0]))
            ;
        hog_detector[sv.cols] = (float)-rho;
        return hog_detector;
}
/*
* Convert training/testing set to be used by OpenCV Machine Learning
    algorithms.
* TrainData is a matrix of size (#samples x max(#cols,#rows) per samples), in
     32FC1.
* Transposition of samples are made if needed.
*/
void convert_to_ml(const vector< Mat > & train_samples, Mat& trainData)
{
        //--Convert data
        const int rows = (int)train_samples.size();
        const int cols = (int)std::max(train_samples[0].cols, train_samples
            [0].rows);
        Mat tmp(1, cols, CV_32FC1); //< used for transposition if needed
        trainData = Mat(rows, cols, CV_32FC1);
        for (size_t i = 0; i < train_samples.size(); ++i)
        {
                CV_Assert(train_samples[i].cols == 1 || train_samples[i].rows
                    == 1);
                if (train_samples[i].cols == 1)
                {
                        transpose(train_samples[i], tmp);
                        tmp.copyTo(trainData.row((int)i));
                }
                else if (train_samples[i].rows == 1)
                {
                        train_samples[i].copyTo(trainData.row((int)i));
                }

        }
}
```

```
void load_images(const String & dirname, vector< Mat > & img_lst, bool
    showImages = false)
{
        vector< String > files;
        glob(dirname, files);
        for (size_t i = 0; i < files.size(); ++i)
        {
                Mat img = imread(files[i]); // load the image
                if (img.empty())            // invalid image, skip it.
                {
                        cout << files[i] << " is invalid!" << endl;
                        continue;
                }
                ;
                Mat resized_img;
                resize(img, resized_img, Size(96, 96), 0, 0,
                    INTER_LINEAR_EXACT);


                if (showImages)
                {
                        imshow("image", img);
                        waitKey(1);
                }
                img_lst.push_back(img);
                // img_lst.push_back(resized_img);
        }
}

void sample_neg(const vector< Mat > & full_neg_lst, vector< Mat > & neg_lst,
    const Size & size)
{
        Rect box;
        box.width = size.width;
        box.height = size.height;

        const int size_x = box.width;
```

```
        const int size_y = box.height;
        srand((unsigned int)time(NULL));
        clog << full_neg_lst.size() << endl;

        for (size_t i = 0; i < full_neg_lst.size(); i++)
                if (full_neg_lst[i].cols >= box.width && full_neg_lst[i].rows
                    >= box.height)
                {
                        box.x = rand() % (full_neg_lst[i].cols - size_x);
                        box.y = rand() % (full_neg_lst[i].rows - size_y);
                        Mat roi = full_neg_lst[i](box);
                        neg_lst.push_back(roi.clone());
                }
}

void computeHOGs(const Size wsize, const vector< Mat > & img_lst, vector< Mat
    > & gradient_lst, bool use_flip)
{
        HOGDescriptor hog;
        hog.winSize = wsize;
        Mat gray;
        vector< float > descriptors;

        for (size_t i = 0; i < img_lst.size(); i++)
        {
                if (img_lst[i].cols >= wsize.width && img_lst[i].rows >=
                    wsize.height)
                {

                        Rect r = Rect((img_lst[i].cols - wsize.width) / 2,
                                (img_lst[i].rows - wsize.height) / 2,
                                wsize.width,
                                wsize.height);
                        cvtColor(img_lst[i](r), gray, COLOR_BGR2GRAY);
                        hog.compute(gray, descriptors, Size(8, 8), Size(0, 0)
                            );
                        gradient_lst.push_back(Mat(descriptors).clone());
                        if (use_flip)
```

```
                        {
                                flip(gray, gray, 1);
                                hog.compute(gray, descriptors, Size(8, 8),
                                    Size(0, 0));
                                gradient_lst.push_back(Mat(descriptors).clone
                                    ());
                        }
                }
        }
}

void test_trained_detector(String obj_det_filename, String test_dir, String
    videofilename)

{
        cout << "Testing trained detector..." << endl;
        HOGDescriptor hog;
        hog.load(obj_det_filename);

        vector< String > files;
        glob(test_dir, files);

        int delay = 1;
        VideoCapture cap;

        if (videofilename != "")
        {
                if (videofilename.size() == 1 && isdigit(videofilename[0]))
                        cap.open(videofilename[0] - '0');
                else
                        cap.open(videofilename);
        }


        obj_det_filename = "TEST_" + obj_det_filename;
        namedWindow(obj_det_filename, WINDOW_NORMAL);

        VideoWriter w_video;
```

```
for (size_t i = 0;; i++)
{
        Mat img;

        if (cap.isOpened())
        {
                cap >> img;
                delay = 1;
        }

        else if (i < files.size())
        {
                img = imread(files[i]);
        }

        if (img.empty())
        {
                return;
        }

        vector< Rect > detections;
        vector< double > foundWeights;

        hog.detectMultiScale(img, detections, foundWeights);
        for (size_t j = 0; j < detections.size(); j++)
        {
                Scalar color = Scalar(0, foundWeights[j] *
                    foundWeights[j] * 200, 0);
                rectangle(img, detections[j], color, img.cols / 400 +
                     1);
        }

        imshow(obj_det_filename, img);

        if (!w_video.isOpened())
        {
                w_video.open("sample.mp4", VideoWriter::fourcc('x', '
```

```
                        v', 'i', 'd'), 30, img.size(), false);

                if (!w_video.isOpened())
                {
                        throw std::runtime_error("can't create video
                            writer");
                }
            }
            w_video << img;

            if (waitKey(delay) == 27)
            {
                return;
            }
        }
}

int main(int argc, char** argv)

{
        const char* keys =
        {
            "{help h|      | show help message}"
            "{pd     |      | path of directory contains possitive images}"
            "{nd     |      | path of directory contains negative images}"
            "{td     |      | path of directory contains test images}"
            "{tv     |      | test video file name}"
            "{dw     |      | width of the detector}"
            "{dh     |      | height of the detector}"
            "{f      |false| indicates if the program will generate and
                use mirrored samples or not}"
            "{d      |false| train twice}"
            "{t      |false| test a trained detector}"
            "{v      |false| visualize training steps}"
            "{fn     |my_detector.yml| file name of trained SVM}"
        };

        CommandLineParser parser(argc, argv, keys);
```

```
if (parser.has("help"))
{
        parser.printMessage();
        exit(0);
}

String pos_dir = parser.get< String >("pd");
String neg_dir = parser.get< String >("nd");
String test_dir = parser.get< String >("td");
String obj_det_filename = parser.get< String >("fn");
String videofilename = parser.get< String >("tv");
int detector_width = parser.get< int >("dw");
int detector_height = parser.get< int >("dh");
bool test_detector = parser.get< bool >("t");
bool train_twice = parser.get< bool >("d");
bool visualization = parser.get< bool >("v");
bool flip_samples = parser.get< bool >("f");

if (test_detector)
{
        test_trained_detector(obj_det_filename, test_dir,
            videofilename);
        exit(0);
}

if (pos_dir.empty() || neg_dir.empty())
{
        parser.printMessage();
        cout << "Wrong number of parameters.\n\n"
        cout << "Example command line:\n" << argv[0] << " -dw=64 -dh
            =128 -pd=C:/path/to/positives/folder -nd=C:/path/to/
            negatives/folder -td=C:/path/to/testing/folder -fn=
            HOG_SVM_DESCRIPTOR.xml -d\n"
        cout << "\nExample command line for testing trained detector
            :\n" << argv[0] << " -t -fn=HOG_SVM_DESCRIPTOR.xml -td=C:/
            path/to/testing/folder";
        exit(1);
```

```
}

vector< Mat > pos_lst, full_neg_lst, neg_lst, gradient_lst;
vector< int > labels;

clog << "Positive images are being loaded...";
load_images(pos_dir, pos_lst, visualization);

if (pos_lst.size() > 0)
{
        clog << "...[done]" << endl;
}
else
{
        clog << "no image in " << pos_dir << endl;
        return 1;
}

Size pos_image_size = pos_lst[0].size();
clog << pos_lst[0].cols << ' ' << pos_lst[0].rows << endl;

if (detector_width && detector_height)
{
        pos_image_size = Size(detector_width, detector_height);
}

else
{
        for (size_t i = 0; i < pos_lst.size(); ++i)
        {
                if (pos_lst[i].size() != pos_image_size)
                {
                        cout << "All positive images should be same
                            size!" << endl;
                        exit(1);
                }
        }
        pos_image_size = pos_image_size / 8 * 8;
```

```
}

clog << "Negative images are being loaded...";
load_images(neg_dir, neg_lst, false);
// sample_neg(full_neg_lst, neg_lst, pos_image_size);
clog << "...[done]" << endl;

clog << "Histogram of Gradients are being calculated for positive
    images...";
computeHOGs(pos_image_size, pos_lst, gradient_lst, flip_samples);
size_t positive_count = gradient_lst.size();
labels.assign(positive_count, +1);
clog << "...[done] (positive count : " << positive_count << ")" <<
    endl;

clog << "Histogram of Gradients are being calculated for negative
    images...";
computeHOGs(pos_image_size, neg_lst, gradient_lst, flip_samples);
size_t negative_count = gradient_lst.size() - positive_count;
labels.insert(labels.end(), negative_count, -1);
CV_Assert(positive_count < labels.size());
clog << "...[done] (negative count : " << negative_count << ")" <<
    endl;

Mat train_data;
convert_to_ml(gradient_lst, train_data);

clog << "Training SVM...";
Ptr< SVM > svm = SVM::create();

/* Default values to train SVM */
svm->setCoef0(0.0);
svm->setDegree(3);
svm->setTermCriteria(TermCriteria(CV_TERMCRIT_ITER + CV_TERMCRIT_EPS,
     1000, 1e-3));
svm->setGamma(0);
svm->setKernel(SVM::LINEAR);
svm->setNu(0.5);
```

```
svm->setP(0.1); // for EPSILON_SVR, epsilon in loss function?
svm->setC(0.01); // From paper, soft classifier
svm->setType(SVM::EPS_SVR); // C_SVC; // EPSILON_SVR; // may be also
    NU_SVR; // do regression task
svm->train(train_data, ROW_SAMPLE, labels);
clog << "...[done]" << endl;

if (train_twice)
{
        clog << "Testing trained detector on negative images. This
            may take a few minutes...";
        HOGDescriptor my_hog;
        my_hog.winSize = pos_image_size;

        // Set the trained svm to my_hog
        my_hog.setSVMDetector(get_svm_detector(svm));
        vector< Rect > detections;
        vector< double > foundWeights;
        for (size_t i = 0; i < full_neg_lst.size(); i++)
        {
                if (full_neg_lst[i].cols >= pos_image_size.width &&
                    full_neg_lst[i].rows >= pos_image_size.height)
                        my_hog.detectMultiScale(full_neg_lst[i],
                            detections, foundWeights);
                else
                        detections.clear();

                for (size_t j = 0; j < detections.size(); j++)
                {
                        Mat detection = full_neg_lst[i](detections[j
                            ]).clone();
                        resize(detection, detection, pos_image_size,
                            0, 0, INTER_LINEAR_EXACT);
                        neg_lst.push_back(detection);
                }

                if (visualization)
                {
```

```
                    for (size_t j = 0; j < detections.size(); j
                        ++)
                    {
                        rectangle(full_neg_lst[i], detections
                            [j], Scalar(0, 255, 0), 2);
                    }
                    imshow("testing trained detector on negative
                        images", full_neg_lst[i]);
                    waitKey(1);
                }
            }
            clog << "...[done]" << endl;
            gradient_lst.clear();
            clog << "Histogram of Gradients are being calculated for
                positive images...";
            computeHOGs(pos_image_size, pos_lst, gradient_lst,
                flip_samples);
            positive_count = gradient_lst.size();
            clog << "...[done] (positive count : " << positive_count <<
                ")" << endl;

            clog << "Histogram of Gradients are being calculated for
                negative images...";
            computeHOGs(pos_image_size, neg_lst, gradient_lst,
                flip_samples);
            negative_count = gradient_lst.size() - positive_count;
            clog << "...[done] (negative count : " << negative_count <<
                ")" << endl;

            labels.clear();
            labels.assign(positive_count, +1);
            labels.insert(labels.end(), negative_count, -1);

            clog << "Training SVM again...";
            convert_to_ml(gradient_lst, train_data);
            svm->train(train_data, ROW_SAMPLE, labels);
            clog << "...[done]" << endl;
}
```

```
        HOGDescriptor hog;
        hog.winSize = pos_image_size;
        hog.setSVMDetector(get_svm_detector(svm));
        svm->save("SVM" + obj_det_filename);
        hog.save(obj_det_filename);

        if(test_detector)
                test_trained_detector(obj_det_filename, test_dir,
                    videofilename);
        return 0;

}
```

Bibliography

[1] P. S. Chakraborty, P. R. Sinha, and A. Tiwari, "Real time optimized traffic management algorithm for intelligent transportation systems," in *2015 IEEE International Conference on Computational Intelligence Communication Technology*, pp. 744–749, Feb 2015.

[2] P. Y. Chen, C. C. Huang, C. Y. Lien, and Y. H. Tsai, "An efficient hardware implementation of hog feature extraction for human detection," *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, pp. 656–662, April 2014.

[3] "Xilinx opencv user guide." https://www.xilinx.com/support/documentation/sw$_m$*anuals*/*xilinx*2017$_4$/*xilinx*
opencv − *user* − *guide.pdf*.

[4] M. Webster, "Traffic." https://www.merriam-webster.com/dictionary/traffic, Nov 2017.

[5] G. Network, "Stress, pollution, fatigue: How traffic jams affect your health." http://www.gmanetwork.com/news/lifestyle/healthandwellness/536203/stress-pollution-fatigue-how-traffic-jams-affect-your-health/story/, September 2017.

[6] E. Sauler, "Mmda report: Road accidents claimed 446 lives in 2016." http://newsinfo.inquirer.net/871187/mmda-report-road-accidents-claimed-446-lives-in-2016, Feb 2017.

[7] B. Romualdez, "Traffic worse than ever." http://newsinfo.inquirer.net/871187/mmda-report-road-accidents-claimed-446-lives-in-2016, Sept 2016.

[8] Carmudi, "The philippine traffic problem: How the new government plans to solve it." https://www.carmudi.com.ph/journal/philippine-traffic-problem-new-government-plans-solve/, Aug 2016.

[9] A. Bagaoisan, "Mmda report: Road accidents claimed 446 lives in 2016." http://news.abs-cbn.com/focus/01/07/16/why-filipinos-are-among-worlds-most-religious, Jan 2016.

[10] “Embedded computer vision: Which device should you choose?.” https://www.learnopencv.com/embedded-computer-vision-which-device-should-you-choose/.

[11] “How to turn hardware acceleration on and off in chrome.” https://www.lifewire.com/hardware-acceleration-in-chrome-4125122/.

[12] “What is an fpga?.” https://embeddedmicro.com/blogs/tutorials/what-is-an-fpga/.

[13] K. Tan, “Mmda launches new traffic control system âhermesâ.” https://www.carmudi.com.ph/journal/philippine-traffic-problem-new-government-plans-solve/, Jan 2014.

[14] INDRA, “About indra.” https://www.indracompany.com/en/indra.

[15] A. Manalo, “About project hermes in the philippines.” Personal Communication, Oct 2017.

[16] N. Detection, “Technical.” https://nortechdetection.com.au/support/technical/.

[17] A. Saikar, M. Parulekar, A. Badve, S. Thakkar, and A. Deshmukh, “Trafficintel: Smart traffic management for smart cities,” in *2017 International Conference on Emerging Trends Innovation in ICT (ICEI)*, pp. 46–50, Feb 2017.

[18] C. Finch, “Advantages disadvantages of rfid.” https://www.techwalla.com/articles/advantages-disadvantages-of-rfid.

[19] Techopedia, “Definition - what does computer vision mean?.” https://www.techopedia.com/definition/32309/computer-vision.

[20] T. Osman, S. S. Psyche, J. M. S. Ferdous, and H. U. Zaman, “Intelligent traffic management system for cross section of roads using computer vision,” in *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 1–7, Jan 2017.

[21] S. Associates, “How do traffic signals work?.” http://www.traffic-signal-design.com/how$_d o_t r a f f i c_s i g n a l s_w o r k . h t m$.

[22] Abraham Silberschatz, “Operating system concepts.” http://iips.icci.edu.iq/images/exam/Abraham-Silberschatz-Operating-System-Concepts—9th2012.12.pdf.

[23] L. W. Chen and C. C. Chang, “Cooperative traffic control with green wave coordination for multiple intersections based on the internet of vehicles,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, pp. 1321–1335, July 2017.

[24] Y.-Q. Wang, "An Analysis of the Viola-Jones Face Detection Algorithm," *Image Processing On Line*, vol. 4, pp. 128–148, 2014.

[25] M. Jacobsen, Z. Cai, and N. Vasconcelos, "Fpga implementation of hog based pedestrian detector," in *2015 International SoC Design Conference (ISOCC)*, pp. 191–192, Nov 2015.

[26] A. Varghese and G. Sreelekha, "Background subtraction for vehicle detection," in *2015 Global Conference on Communication Technologies (GCCT)*, pp. 380–382, April 2015.

[27] J.-C. Tai, S.-T. Tseng, C.-P. Lin, and K.-T. Song, "Real-time image tracking for automatic traffic monitoring and enforcement applications," *Image and Vision Computing*, vol. 22, no. 6, pp. 485 – 501, 2004.

[28] S. Shujuan, X. Zhize, W. Xingang, H. Guan, W. Wenqi, and X. De, "Real-time vehicle detection using haar-surf mixed features and gentle adaboost classifier," in *The 27th Chinese Control and Decision Conference (2015 CCDC)*, pp. 1888–1894, May 2015.

[29] Z. Chen, N. Pears, M. Freeman, and J. Austin, "Road vehicle classification using support vector machines," in *2009 IEEE International Conference on Intelligent Computing and Intelligent Systems*, vol. 4, pp. 214–218, Nov 2009.

[30] B. Luitel, Y. V. S. Murthy, and S. G. Koolagudi, "Sound event detection in urban soundscape using two-level classification," in *2016 IEEE Distributed Computing, VLSI, Electrical Circuits and Robotics (DISCOVER)*, pp. 259–263, Aug 2016.

[31] C. M. Bautista, C. A. Dy, M. I. Manalac, R. A. Orbe, and M. Cordel, "Convolutional neural network for vehicle detection," in *2016 IEEE Region 10 Symposium (TENSYMP)*, pp. 277–281, May 2016.

[32] K. R. A. Mallareddy, R.Chitti Babu, "Vehicle tracking using canny edge detector," Master's thesis, Sri Indu Institute of Engineering Technology, Vanasthalipuram, Hyderabad, 2014.

[33] Y.-L. Chen, B.-F. Wu, and C.-J. Fan, "Real-time vision-based multiple vehicle detection and tracking for nighttime traffic surveillance," in *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*, pp. 3352–3358, IEEE, 2009.

[34] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, pp. I–511–I–518 vol.1, 2001.

[35] P. Viola, M. J. Jones, and D. Snow, "Detecting pedestrians using patterns of motion and appearance," in *Proceedings Ninth IEEE International Conference on Computer Vision*, pp. 734–741 vol.2, Oct 2003.

[36] C. I. Patel and R. Patel, "Counting cars in traffic using cascade haar with klp," *International Journal of Computer and Electrical Engineering*, vol. 5, no. 4, p. 435, 2013.

[37] 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)*, vol. 1, pp. 886–893 vol. 1, June 2005.

[38] C. Dizon, L. Magpayo, and A. Uy, "An implementation of an open-space visual smart parking system," Master's thesis, University of the Philippines, Diliman, Diliman, 2017.

[39] X. Li and X. Guo, "A hog feature and svm based method for forward vehicle detection with single camera," in *2013 5th International Conference on Intelligent Human-Machine Systems and Cybernetics*, vol. 1, pp. 263–266, Aug 2013.

[40] L. Wang, Y. Lu, H. Wang, Y. Zheng, H. Ye, and X. Xue, "Evolving boxes for fast vehicle detection," in *2017 IEEE International Conference on Multimedia and Expo (ICME)*, pp. 1135–1140, July 2017.

[41] M. Drożdż and T. Kryjak, "Fpga implementation of multi-scale face detection using hog features and svm classifier," *Image Processing & Communications*, vol. 21, no. 3, pp. 27–44, 2016.

[42] M. Jacobsen, Z. Cai, and N. Vasconcelos, "Fpga implementation of hog based pedestrian detector," in *2015 International SoC Design Conference (ISOCC)*, pp. 191–192, Nov 2015.

[43] J. Cho, S. Mirzaei, J. Oberg, and R. Kastner, "Fpga-based face detection system using haar classifiers," in *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*, pp. 103–112, ACM, 2009.

[44] M. E. Ilas, "New histogram computation adapted for fpga implementation of hog algorithm: For car detection applications," in *2017 9th Computer Science and Electronic Engineering (CEEC)*, pp. 77–82, Sept 2017.

[45] "Use scp to securely transfer files between two unix computers." https://kb.iu.edu/d/agye.

[46] "Paramiko's documentation." http://docs.paramiko.org/en/2.4/.

[47] “Struct â interpret strings as packed binary data.” https://docs.python.org/2/library/struct.htmlstruct.pack.

[48] “About.” https://opencv.org/about.html/.

[49] “Simulation of urban mobility.” http://sumo.dlr.de/index.html.

[50] “Sumo user documentation.” http://sumo.dlr.de/userdoc/.

[51] “Python implementation of dijkstra’s algorithm.” https://gist.github.com/econchick/4666413.

[52] “Traci/interfacing traci from python.” http://sumo.dlr.de/userdoc/TraCI/Interfacing$_{T}raCI_{f}rom_{P}ython.htm$

[53] “Opencv c++ tutorial sliding window.” http://funvision.blogspot.com/2015/12/opencv-tutorial-sliding-window.html, December 2015.

[54] “Non-maximum suppression for object detection in python.” https://www.pyimagesearch.com/2014/11/17/non-maximum-suppression-object-detection-python/, November 2014.

[55] “Vivado design suite user guide: High-level synthesis.” https://kb.iu.edu/d/agye.

[56] “Ethernet.” http://searchnetworking.techtarget.com/definition/Ethernet.

[57] “traci.” http://www.sumo.dlr.de/daily/pydoc/traci.html.

[58] “Socket â low-level networking interface.” https://docs.python.org/2/library/socket.html.

[59] “Roll a ball tutorial.” https://unity3d.com/learn/tutorials/s/roll-ball-tutorial.

[60] “How to c socket programming.” http://csharp.net-informations.com/communications/csharp-socket-programming.htm.

[61] “Unity project and 3rd party apps.” https://answers.unity.com/questions/15422/unity-project-and-3rd-party-apps.html.

[62] “Ua-detrac benchmark suite.” http://detrac-db.rit.albany.edu/.

[63] “Open source computer vision library.” https://github.com/opencv/opencv.

[64] A. B. K., V. Venkatraman, A. R. Kumar, and S. D. S., “Accelerating real-time computer vision applications using hw/sw co-design,” in *2017 International Conference on Computer, Communications and Electronics (Comptelix)*, pp. 458–463, July 2017.

[65] “Xillybus..” Available at http://xillybus.com/downloads/xillybusproductbrief.pdf.

[66] T. Semertzidis, K. Dimitropoulos, A. Koutsia, and N. Grammalidis, "Video sensor network for real-time traffic monitoring and surveillance," *IET Intelligent Transport Systems*, vol. 4, pp. 103–112, June 2010.

[67] G. Litos, X. Zabulis, and G. Triantafyllidis, "Synchronous image acquisition based on network synchronization," in *2006 Conference on Computer Vision and Pattern Recognition Workshop (CVPRW'06)*, pp. 167–167, June 2006.

[68] MuleSoft, "What is an api? (application programming interface)." https://www.mulesoft.com/resources/api/what-is-an-api.

[69] http://www.ntp.org/ntpfaq/NTP-s-def.htm.

[70] Tristan-TB, "Fpga-based traffic classifier using the svm algorithm." https://github.com/twisterss/hardware-traffic-classifier, October 2014.

[71] S. Tonight, "Types of network topology." http://www.studytonight.com/computer-networks/network-topology-types.

[72] A. Redondi, L. Baroffio, L. Bianchi, M. Cesana, and M. Tagliasacchi, "Compress-then-analyze versus analyze-then-compress: What is best in visual sensor networks?," *IEEE Transactions on Mobile Computing*, vol. 15, pp. 3000–3013, Dec 2016.

[73] B. Cheng and G. P. Hancke, "A service-oriented architecture for wireless video sensor networks: Opportunities and challenges," in *IECON 2015 - 41st Annual Conference of the IEEE Industrial Electronics Society*, pp. 002667–002672, Nov 2015.

[74] R. Zheng, "Commentary paper on "an object- and task-oriented architecture for automated video surveillance in distributed sensor networks"," in *2008 IEEE Fifth International Conference on Advanced Video and Signal Based Surveillance*, pp. 347–347, Sept 2008.