April 2017

AWeD: Automatic Weapons Detection

Carlos Monterrosa Diaz
Worcester Polytechnic Institute

Georgios Karapanagos
Worcester Polytechnic Institute

Glen Mould
Worcester Polytechnic Institute

Paul Solomon Raynes
Worcester Polytechnic Institute

Skyler Alsever
Worcester Polytechnic Institute

AWeD

(Automatic Weapons Detection)

Advisors: Susan Jarvis

Nicholas Bertozzi

Written by :

Skyler Alsever

Georgios Karapanagos

Carlos Monterrosa Diaz

Glen Mould

Paul Raynes

A Major Qualifying Project WORCESTER POLYTECHNIC INSTITUTE

Submitted to the Faculty of

Worcester Polytechnic Institute

In partial fulfillment of the requirements for the

Degree in Bachelor of Science

in

Electrical and Computer Engineering

October 2016- April 2017

Abstract

The goal of this project is to design an integrated system that allows for fast and reliable processing of high quality video data and in doing so detect and react to the presence of a firearm or other weaponry when used in a threatening or dangerous manner. This is accomplished through the combined use of computer vision processing techniques implemented on an FPGA as well as a convolutional neural network trained to determine the presence of a threat.

Table of Contents

Abstract 1
Table of Authors 5
Introduction 9
Background Information 10
Crime Statistics and Areas of Needed Improvement 10
Current Standards for General Weapons Surveillance 11
Computer Vision: An Alternative Approach to Surveillance 12
Current Approaches to CV Surveillance 13
Improving the Standard: Hardware Acceleration and the FPGA 15
Project Methodology 16
System Overview, Design Specification, and the Computer Vision Pipeline 16
Camera / Image Sensor Requirements 17
Image Pre-Processing and Filtering 18
Softcore Processing: FPGAs and HDLs 21
Machine Learning - Neural Network for Object Detection 23
Summary of ML Algorithms 26
OpenCV 27
Decision Making and Alert Systems 28
Power Considerations 29
Mechanical Components: 2 DoF Gimbal 30
Mechanical Components: Mechanical Zoom 31
Verification, Testing, and Methods of Analysis 31
Data Gathering 31
Verification and Testing 32
Project Timeline, Goals and Budget Information 34
Operational Milestones and Stretch Goals 34
Implementation Timeline and Gantt Chart 36
Hardware Resources 37

Proposed camera solution: 37
Proposed FPGA Development Board Solution 38
Software Resources and Licensing 38
Total Budget Review and Expenditures 39
Background Conclusion 39
Design Specifications 41
Top Level System Block Diagram 41
Camera 42
Mechanical Components 43
Base 44
Horizontal Gear Piece 45
Vertical Gear Piece 46
Going from Digital to Physical 46
Sensors 47
Image Preprocessing 48
Camera Interface 49
HPS and FPGA Communication 49
Processing Filters 49
Contrast Correction 52
Edge Detection 54
2D Sobel Filter 54
3D Sobel Filter 57
Foreground Detection 58
Implementation of Algorithms on DE1-SoC 61
The Linux Kernel 65
Interrupt Service Routines 67
Machine Learning - Building a Binary Classifier 69
Implementing the CNN 69
Creating the Training Datasets 71
Training the Model 73
Threat Detection Routine 74
Raspberry Pi Prototype 76
Gimbal Control 77

System Testing and Evaluation 79

Testing the CNN 79

Implementing the HPS-FPGA data transfer 82

Filters 89

Suggested Improvements / Changes 97

UVC Driver 97

Board Output Voltage 99

Conclusion 100

Glossary 101

References 103

Appendix - A: Random Forest for Object Detection 107

Appendix - B: Multi-classifier Boosting 110

Appendix - C: Computer Vision: Training Script 113

Appendix - D: Computer Vision: Testing Script 116

Appendix - E: Computer Vision: Helper Functions 120

Appendix - F: Matlab ImageToText.m code 130

Appendix - G: Matlab TextToImage.m code 131

Appendix - H: Sobel Filter Verilog Code 132

Table of Authors

1. Title Page……………………………………………………………………………………………………Glen
2. Introduction…………………………………………………………………………………………………..Carlos
3. Background Information
    1. Crime Statistics and Areas of Needed Improvement……………………………………..Carlos
    2. Current Standards for General Weapon's Surveillance…………………………………...Glen
    3. Computer Vision: An Alternative Approach to Surveillance…………………………George
    4. Current Approaches to Surveillance…………………………………………………………....Glen
    5. Improving the Standard: Hardware Acceleration and the FPGA………………………Paul
4. Project Methodology
    1. System Overview, Design Specification, and the Computer Vision Pipeline
        1. Camera / Image Sensor Requirements ……………………………………….....Carlos
        2. Image Pre-Processing and Filtering…………………………………….....Carlos, Glen
        3. Softcore Processing: FPGA's and HDL's…………………………………………..Skyler
        4. Machine Learning: Generating a Comparative Profile………………………George
        5. OpenCV ……………………………………………………………………………………..Paul
        6. Decision Making and Alert System……………………………………………..…....Glen
        7. Power Considerations…………………………………………………………..………..Glen
        8. Mechanical Components: 2 DoF Gimbal……………………………………………..Paul
        9. Mechanical Components: Mechanical Zoom……………………...………………..Paul
    2. Verification, Testing, and Methods of Analysis …………………………………………George
5. Project Timeline, Goals, and Budget Information…………………………………………………..All
    1. Operational Milestones and Stretch Goals……………………………………...…………..All
    2. Implementation Timeline and Gantt Chart……………………………………………………..Glen

3. Hardware Resources……All
    1. Proposed Camera Solutions……Carlos
    2. Proposed FPGA Development Board Solution……Skyler
4. Software Resources and Licensing……All
5. Total Budget Review and Expenditures……All

6. Background Conclusion……All

7. Design Specifications……All
    1. Top Level System Block Diagram……Glen
    2. Camera……Carlos
    3. Mechanical Components……Paul
        1. Base……Paul
        2. Horizontal Gear Piece……Paul
        3. Vertical Gear Piece……Paul
        4. Going from Digital to Physical……Paul
        5. Sensors……Paul
    4. Image Preprocessing……Carlos, Glen
        1. Camera Interface……Glen
        2. HPS and FPGA Communication……Glen
        3. Processing Filters……Glen
        4. Contrast Correction……Glen
        5. Edge Detection……Glen
        6. 2D Sobel Filter……Glen
        7. 3D Sobel Filter……Glen
        8. Foreground Detection……Carlos
    5. Implementation of Algorithms on DE1-SoC……Skyler

6. The Linux Kernel……………………………………………………………Skyler
7. Interrupt Service Routines……………………………………………………Skyler
8. Machine Learning – Building a Binary Classifier……………………………George
    1. Implementing the CNN……………………………………………………George
    2. Creating the Training Datasets………………………………………………George
    3. Training the Model…………………………………………………………George
    4. Threat Detection Routine……………………………………………………George
    5. Raspberry Pi Prototype………………………………………………………George
    6. Gimbal Control……………………………………………………………George
8. System Testing and Evaluation………………………………………………………All
    1. Testing the CNN…………………………………………………………George
    2. Implementing the HPS-FPGA data transfer……………………………………Glen
    3. Filters………………………………………………………………Carlos, Glen
9. Suggested Improvements / Changes…………………………………………………All
    1. UVC Driver………………………………………………………Carlos, Skyler
    2. Board Output Voltage……………………………………………Paul, Skyler
10. Conclusion……………………………………………………………………Paul
11. Glossary………………………………………………………………………All
12. References……………………………………………………………………All
13. Appendix - A: Random Forest for Object Detection…………………………George
14. Appendix - B: Multi-classifier Boosting……………………………………George
15. Appendix - C: Computer Vision: Training Script………………………………George
16. Appendix - D: Computer Vision: Testing Script………………………………George
17. Appendix - E: Computer Vision: Helper Functions……………………………George

18. Appendix - F: Matlab ImageToText.m code……………………………………………..…………Carlos, Glen

19. Appendix - G: Matlab TextToImage.m code…………………………………….…………….Carlos, Glen

20. Appendix - H: Sobel Filter Verilog Code……………………………………………….………Carlos, Glen

Introduction

Public safety is a major concern in today's modern society. Modern weaponry and firearms pose serious threats to the safety and security of the everyday people, and recent events and media coverage have only further publicized the inherent dangers that one may face in even the most public of places. It is hoped that through the vigilance of the common citizen and the swift response of the authorities, violent perpetrators who threaten others with dangerous weaponry can be quickly and reliably apprehended and fatalities prevented. But often times, when threatened with the very real danger of a live firearm, people panic, and their justified self-preservation may prevent the proper authorities from being notified, causing small but noticeable delays in police response time at best and resulting in the loss of lives from failure to respond at worst.

These dangerous situations can often be prevented by proper monitoring technologies such as closed circuit television. However, there are many public and even private areas of trespass for which such technologies could not easily be implemented. A public park, for instance, cannot efficiently be monitored by CCTV systems with human operators due simply in part to their size. Trying to correct this issue by adding additional manpower costs both time and money and increases the chance that simple human error might occur due to negligence, absence, or simple misinterpretation of data. And when a single slip up could lead to death or injury, reliability and consistency becomes a very important factor.

As such, a system that removes humans from the equation may be an ideal solution. Using the CCTV model of approach, but then attempting to automate the process of observation and alert can help standardize and streamline the process, but at this current time the technology needed to do so is expensive, complicated and somewhat inaccurate due mostly to the limitations of current non-specialized software and hardware versus the principal challenge of processing large amounts of high quality video data continually over a small amount of time.

The focus of our Major Qualifying Project is to design an integrated system that allows for fast and reliable processing of high quality video data and in doing so can detect and react to the presence of a firearm or other weaponry when used in a threatening or dangerous manner. The system would be designed to allow for easy integration in areas where public and private safety is a major concern such as parks, banks, schools and areas of transit. The system would also be designed to be as autonomous as physically possible, reducing the need for a centralized processing hub and allowing for quick and seamless integration in areas or buildings of varying size and complexity. It is hoped that through the design and implementation of such a system or device, we can come ever closer to ensuring the safety and security of everyday citizens at both the public and private level.

Background Information

Crime Statistics and Areas of Needed Improvement

Gun violence is an increasing problem in the United States in recent years. There were 372 mass shootings in the US in 2015, killing 475 people and wounding 1,870, according to the Mass Shooting Tracker[16] . As gun crime increases, the percentage of gun robberies also increases. According to the FBI in 2015 there were 4,091 bank robberies which 1,725 of them were perpetuated using a firearm[17]. Current systems that are implemented, such as silent alarms or panic buttons, rely in the fact that the silent alarm or panic button must be pressed. However this becomes harder when someone is pointing a gun at you. The Portland Police bureau[3] recommends remaining calm and following the robbers instruction as their guide to robbery response. Users might not have the chance to press the button which means that the authorities might not ever be called in for help. Or even if the button is pressed the time it took them to able to press it might be enough for the robbery or shooter to flee the scene or a tragic accident might happen. In places such as parks, or any public facilities where it is very hard to install these type of signaling devices a problem arises. In other areas that these type of alert of system are not implemented and someone using other devices tries signal for help, there lies a problem that can be solved. There are definitely areas of improvements that could save lives or prevent the robbery as a whole.

Current Standards for General Weapons Surveillance

In attempts to combat gun crimes, a significant amount of research has gone into many gun detection and prevention based technologies. Most modern research into this field focuses on the detection of firearms in general, using technologies such as X-ray and Electro-magnetic scanning and profiling to identify and report when a weapon is found on a person's body. These technologies are excellent and profiling and identifying the presence of a firearm using various signals, sensors and profile recognition, but they do have some major drawbacks in terms of real world applications. The primary faults in these systems are that in many states and counties, open carry of weapons is allowed in banks, parks and other highly public places. A system that primarily focuses on automatically reporting a gun's presence and calling the authorities cannot be realistically implemented in these places because doing so for someone who simply "has a gun" can be viewed as an infringement on basic rights permitted by the 2nd amendment. It can also lead to numerous false flags in which police are called and their time occupied in chasing after someone who isn't causing an actual disturbance. In order for an efficient and fair auto-call type gun-detection system to be made, the system making a call has to not only identify that a gun is present but be able to make intelligent decisions based on the context of how that gun is being used. In this way, isolating detection solely to a set list of situations a gun is used in, false alarms and rights infringements can be reduced but to do so with many of the aforementioned technologies is almost impossible. Other technologies do, however exist outside the current normal that should prove far more capable of accomplishing this complex task: namely systems dealing with processed vision and optical imaging.

Computer Vision: An Alternative Approach to Surveillance

Computer vision (CV) technologies are nowadays commonly used as an effective approach to automated vision processing. CV algorithms can analyze input video streams using models of human behavior to provide high-level information of the recorded events. Effectively, CV offers contextual understanding as part of detecting potential threats. As a result, recent years have seen intelligent video surveillance (IVS) systems being developed to replace the current CCTV systems. By implementing CV technologies, these IVS systems can efficiently monitor an area without requiring human agents for their operation.

Most automated surveillance systems are organized in a similar manner "with low-level image processing techniques feeding into tracking algorithms which in turn feed into higher level scene analysis and/or behaviour analysis modules" [1]. The process, in more detail, can be visualized in figure 2-1 below:

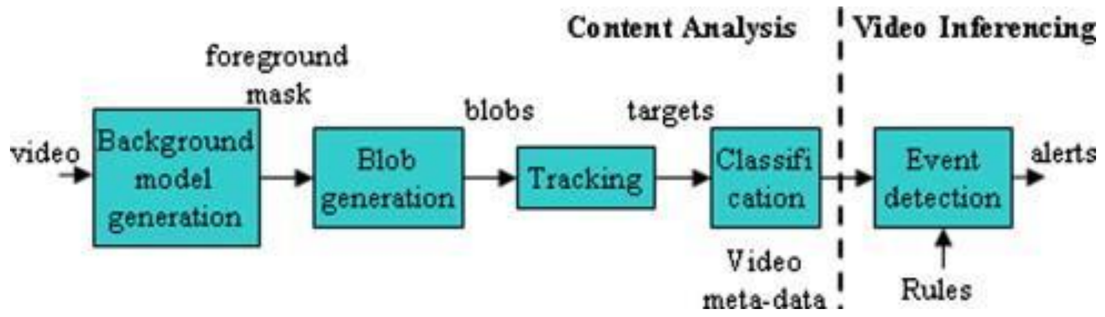


Figure 2-1: Flow diagram of the computer vision pipeline for a security application [2]

The video input from a recording device is processed by the background model generation module, which looks at whether the present pixels differ from the typical ground-truth values. The different pixels are called foreground and are grouped to form into blobs. Blobs are tracked throughout the stream and classified as targets, whose behavior is documented in the output video metadata. With this process, the vast data that is generated by the current CCTV solutions can be handily filtered and anything important is succinctly summarized in the video metadata, the output of the low-level content analysis component. As long as the CV algorithm is fast enough to process the data in real-time, an IVS system can monitor all operating video cameras simultaneously.

The metadata generated gets processed by a high-level algorithm in order to detect if any event has occurred. The rules for an event are set up by the developers of an IVS during its configuration. Based on these rules, the algorithm can then trigger any appropriate response to the events detected. An IVS can, therefore, sound an alarm when it detects a breach of security. Given that the rules implemented can accurately predict the respective events, an IVS can successfully replace human agents who are necessary for current CCTV surveillance systems.

Current Approaches to CV Surveillance

After consideration, we believe that with computer vision we are able to bring our projects ideas to fruition. By adopting the computer vision techniques, we would be able to correct and build upon the ideas and concepts that would bring us closer to a safer and smarter world. With the aid of computer vision, a project was able to identify basic skeletal parts of human beings, the head and the 4 limbs. They were then able to identify the different poses of multiple people in the camera's view to determine a selection of poses that may be deemed as threatening and dangerous or peaceful and harmless to society. Being able to distinguish between a person pointing a gun at someone and the victim having their hands raised, as opposed to non-threatening poses, paves a great way for the authorities to be alerted, when malicious actions are taking place. There were 5 instances used in this project, 80% of the algorithm detections for the 5 body parts were correctly positive, the 20% that could not be detected, had problems because the body parts were not distinguishable due to shadows and arms being very close to the body [13]. Also, In a study in AGH University of Science and Technology in Krakow, Poland, titled Automated Detection of Firearms and Knives in a CCTV Image, a team was able to use computer vision through sliding window mechanisms, background detection and canny edge detection from already existing surveillance cameras, CCTV cameras. This project was based on detecting both knives and firearms. The way the computer vision worked was that from the frames it is fed, it would search for human beings, once found, it would search for the arms of that person and then try to determine if they are holding any objects, if so, they would cross reference the object with their database of positive and negative results of either knives or firearms. This project yields a specificity and sensitivity of knife detection algorithm of 94.93% and 81.18% respectively whilst having a specificity and sensitivity of firearm detection algorithm of 96.69% and 35.98% respectively for the video containing dangerous objects and a firearm detection algorithm 100% specificity for video with harmless objects. Meaning that when detecting the firearms, all the cases that were detected where 100% firearms, but not all the cases where firearms were present, were they reported [12]. From these statistics, it can be seen that the guns were a more detectable object than the knives were. These are due to a number of facts, including the database of knives and guns, not being of a broader spectrum of situations, for example indoors vs outdoors, time of day, weather patterns etc. The main issue is that they are using camera images from CCTV camera recording, which suffer from low resolution and blurriness due to poor quality and inexpensiveness of the cameras. With all of these problems the project faced, they decided to send an alert to a human operator, who would make the decision to call the police or not. To solve all of these problems we aim on getting higher quality cameras that would provide our algorithms with more detail to clearly and more accurately identify the difference between dangerous and harmless objects to furthermore safeguard societies citizens. With an extremely accurate system, the time difference that the middleman, operator, introduces when making the final judgement call to call the police or not could be eradicated, thus decreasing the time it takes for the police to get there from when a threat is detected.

Other projects like Terahertz detection of illegal objects disperses high range frequencies from 100GHz to 500GHz for scanned imaging to be able to detect metal and dielectric objects under clothing. These frequencies use the reflections from the high frequencies to create an image that is then used as input to detect the different dangerous object that are being carried under people's clothing [11]. Our project specifically does not want to use any invasive techniques/methods to detect dangerous weapons. We would not want our project to alert if a concealed weapon is being carried. This not only touches upon privacy issues but also create false alarms in our system because carrying a concealed weapon does not mean the weapon is going to be used for malicious intent, thus the police do not need to be alerted. It is only when this weapon is being wielded in the open that it then becomes a threat to the surrounding people, thus, needs to trigger an alert.

Improving the Standard: Hardware Acceleration and the FPGA

Of previous projects, the one most similar to the project we intend to undertake is the Automated Detection of Firearms and Knives in a CCTV Image project out of AGH. Fundamentally, both their and our projects seek to solve the same problem, information overload of a human operator of a CCTV or similar system. Both projects aim to automate the weapon detection in these systems. Ultimately, though, the system developed at AGH was found to be unusable for firearm detection. The system resulted in too many false positives and was deemed unusable as a result. We seek to outperform their project by removing one major limitation for the system, the lack of use of specialized hardware. We intend to create a better system by utilizing hardware preprocessing on the images before sending them off for software processing resulting in a faster and more consistent reading of weapons.

Project Methodology

System Overview, Design Specification, and the Computer Vision Pipeline

Based on our research findings we formulated a block diagram of our IVS system. The basic functional requirements can be met with one recording unit, capturing video stream, and a processor, running the computer vision algorithms. A response system is also needed to appropriately address detected events. Specialized hardware is used to handle the preprocessing of the images, in an attempt to improve the system's performance both in terms of processing speed and detection accuracy. The resulting block diagram is shown in figure 3-1 below:

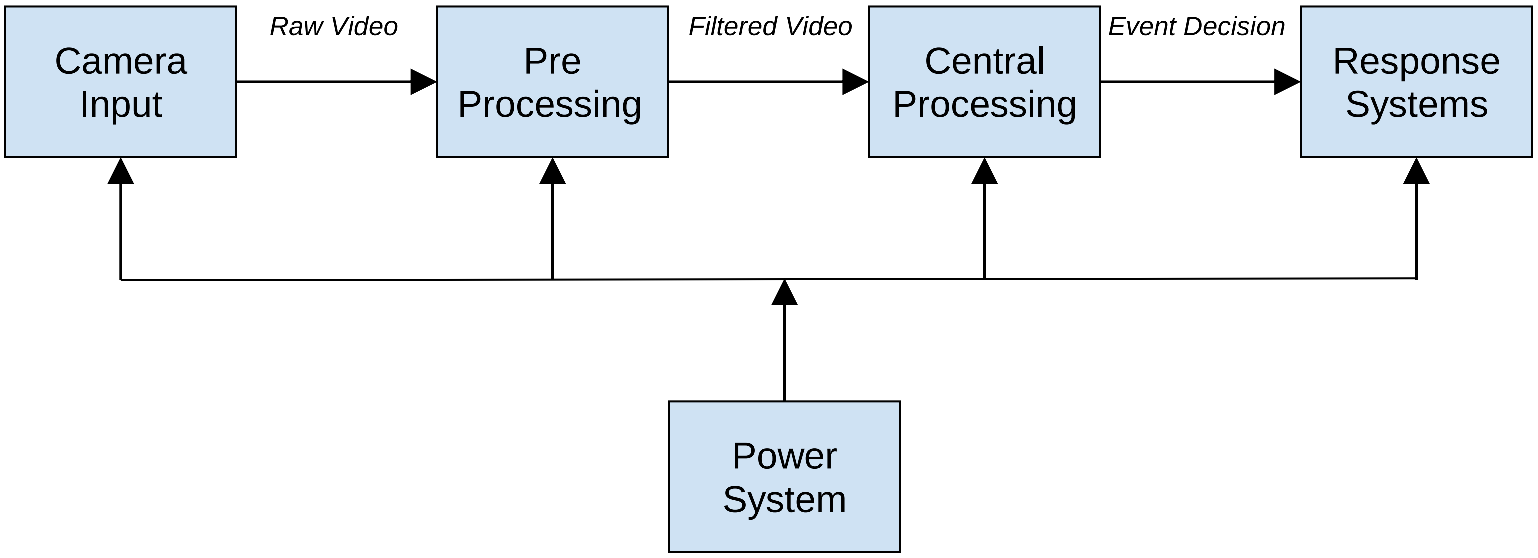


Figure 3-1: Basic block diagram of our IVS system.

The input data acquired by the camera gets quickly processed by the specialized hardware, which efficiently filters it. The filtered video gets more slowly processed by the high-level CV algorithms. Based on the detected events, the central processing system can call the response system to trigger an alarm or send an emergency message. All system modules will be separately connected to the power system.

Camera / Image Sensor Requirements

The camera is the main and only input device used in the project. Therefore the criteria needed for the camera was investigated to find the specifications needed for a camera that fits our needs. The criteria for a camera are its resolution, sensor size, mono or color and CCD(charged couple device) or CMOS(complementary metal-oxide semiconductor) sensor[18]. Each of the specifications mentioned above were needed to find the optimal camera however the price was the deciding factor.

In terms of the resolution of the camera, its a measure of the detail of the image being captured. However, you could also explain resolution of camera as how large the image taken can be reproduced. This means that the higher the resolution of the camera the higher the output printed image size is. For this project there was no need to have a very high resolution camera since the pre-processing stage of the system filtered and trimmed down any unnecessary pixel that input image contains. Also the memory had to be taken into consideration since the higher resolution camera produce a bigger file. The camera resolution should have been between 1 and 10 megapixels. [19]

Also the sensor size needed to be taken into consideration when selecting the camera. The sensor size was important since a bigger sensor area will capture an image with higher quality, however it required a larger diameter. This made the lens of the camera bulkier as the size grew. For example, the Samsung Galaxy S6 has a sensor area of 22.5 mm[20]. Therefore the suggested sensor size should be in the range of 15mm and 80mm. For this range every high end phone lies within it. Cameras that are used in the top of the line phones tend to be cheaper and produce mid to high resolution images.

In addition, the camera can either be mono colored or colored. In relation to the project a colored was ideal. With a colored camera there was access to more pre-processing algorithms and object recognition algorithms. This was due to the body of the person holding the gun being able to be filtered as a pink blob.

Finally, the type of image sensor CCD and CMOS is another feature that needed to be considered when choosing the best camera suited for the project. CCD sensor create a higher quality image with low noise which help reduce the preprocessing stage since there is less noise to be filtered. This was identified as the ideal type of sensor to be used since CMOS sensors need more light to create a low image. For the CMOS sensor, the extra noise caused by low light needs to be dealt with in the preprocessing stage. After further research, a lot of surveillance cameras use CCD sensors since they need to work 24 hours a day including night time where there are small amounts of light present. However, in the current camera market CMOS cameras comprise of almost every digital camera out in the market. [21]

The camera interface that the camera and the FPGA needed to communicate affected which board was bought. If the camera chosen had a micro-USB to USB interface, the board would have needed to have a USB port. The interface would not have been a problem if the chosen camera was a current FPGA module such as the OV5640[21], which is a 5-megapixel FPGA camera module. However, if we chose this module the FPGA that is going to be chosen should have enough power.

Image Pre-Processing and Filtering

The signal received from the camera will be subjected to two stages of preprocessing, one for general image correction and enhancement and one for data extraction and alteration to prepare the image for efficient analysis. Though the camera we hope to use would be of a high resolution, distortions can still be introduced to the signal that would decrease the probability of detecting the firearm or the poses of a potential perpetrator. Techniques must therefore be applied to correct the contrast, color, and sharpness and determine different depths in the frames of the video signal. Applying such techniques allows the incoming image signal to remain consistent across multiple light conditions / settings / times of day while still being accurate to the reference images used to train the systems reference profile. Once this is determined, the image can be processed with a higher detection rate to determine edges of objects within the frames and can further detect the different shapes and objects in the frames.

The first thing that would need to be done to augment the image signal would be to correct its color. Color hue is typically one of the more difficult attributes to correct but redistributing the color saturation or correcting for illumination artifacts in the intensity channel would help our later algorithms and processes. Color intensity is usually the only color information that should be enhanced since it carries a lot of the information required for our processes and is the most readily adjustable. When working with the RGB channels it is normally advised to convert them into a certain color space and making alterations strictly to that color space before converting it back to RGB. Doing otherwise could cause serious, unwanted alterations to be made to the signal. [CV Metrics pg 56-57] Properly adjusted color information within the image can help the system differentiate between certain objects more so than would grayscale alone. Skin tones for instance can be used to identify the presence of a human body in the image and the positioning of the body therewithin. Guns themselves generally tend to fall within the black - silver spectrum of colors and can be more readily located by matching profile color palettes to that of the image object being analysed. Color can also be used subtractively, flagging and removing irrelevant data and shapes in the image such as plants, flooring, furniture and other objects whose color profiles do not match with those found in a person holding a gun.

Noise, as a consequence of the nature of light and the many sources of it and of false reflections / scattering there within, would inevitably be present in the signal and would require removal so as to increase the accuracy of the image signal under scrutiny. The goal of basic image noise removal is to remove the noise present without distorting the underlying image heavily. One way this could be achieved is through contrast correction. Contrast correction can be implemented using thresholding, which segments the image at certain intensity levels to reveal features of foreground background and certain objects. There are several forms of thresholding: floor, the lowest pixel intensity allowed, ceiling, the highest pixel intensity allowed, ramp, shape of the pixel ramp between the floor and the ceiling like linear or log, point, may be a binary threshold point without a floor ceiling or ramp. Global Thresholding technique Lookup Tables(LUTs) are good for contrast remapping. First iterative experimentation is used to find the best floor, ceiling and ramp, then the LUTs can be generated into data tables which would then be used to set thresholds in fast code. Below (Figure 3-1) is an example of linear ramp function used.

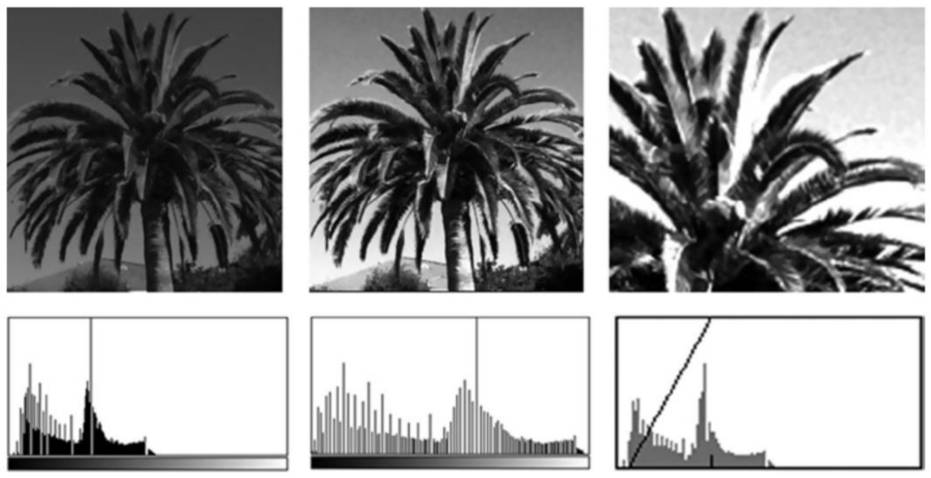


Figure 3-1: example of a linear ramp function used for contrast correction[34]. *(Left) Original image shows palm frond detail compressed into a narrow intensity range obscuring details. (Center) Global histogram equalization restores some detail. (Right) LUT remap function spreads the intensity values to a narrower range to reveal details of the palm fronds. The section of the histogram under the diagonal line is stretched to cover the full intensity range in the right image; other intensity regions are clipped. The contrast corrected image will yield more gradient information when processed with a gradient operator such as Sobel*

Most of these Computer Vision algorithms processes mentioned are fairly large processes that must be repeatedly implemented across a two dimensional array of constantly updating data. Implementing the necessary functions required to extract information from an image frame, such as whether or not a pre-defined item is present within that frame, would therefore take an incredibly long time to implement in purely serial hardware as each individual pixel in an 1080 x 1920 video pixel video feed would then need subjected to the same process one at a time. And while doing so is possible across the fastest cpus / serial processors on market, it often comes at the expense of higher power draw and increased chip size. In most embedded platforms and applications, these trade offs cannot be made with the limited resources available, leading to a desperate need for more power / size efficient hardware to better divide the processing time via another method.

Softcore Processing: FPGAs and HDLs

FPGAs are reprogrammable hardware logic circuits made up of arrays of hardware logic gates and lookup tables and help improve performance of a computerized system by adding a adjustable hardware resources to the design space. They are blank slates that can be designed to either augment existing serial hardware or to create complete implementations of logical operations in stand alone. It differs to a microcontroller such that a microcontroller is an already pre-designed circuit that you are unable to alter the main capabilities of and can only instruct and re-allocate resources based on the code that you write. For an FPGA however, you would have to design the hardware of the circuit to do exactly what you want it to do, including all signal paths, voltage considerations and system lag. There are two most popular types of Hardware Description Languages (HDLs) used nowadays to configure FPGAs, Verilog and VHDL. The HDL is then synthesized into a bit file that is then used to configure the FPGA. This synthesis process is creating gate level representations from a higher-level description of design like the HDL.

An advantage of an FPGA is that it runs faster than a microprocessor as it is a hardware-based implementation, but a disadvantage is that, the configuration is created using (HDL) that is stored on the RAM, so once the power goes out, it would have to be reconfigured. But there are some FPGAs that have a flash chip that can automatically reconfigure it on power up.

For our project implementation we will be using a SoC (System on Chip) type FPGA, which contains primarily the gate arrays and lookup tables one would find within a standard FPGA as well as specially designed HSPs (hard system processor) that essentially functions akin to a low capacity serial processor. This selection allows us to make significant reductions in process time for most computer vision algorithms we'll be running both by enabling the parallelization of as much of these algorithms as possible (logical hardware implementation via the FPGA) while still being able to implement and process the dedicatedly serial operations that cannot be parallelized such as the control operations and decision making. In addition to this, using a SoC style FPGA allows us to reduce costs by no longer requiring multiple serial micro-controllers/processors in addition to our FPGA chip, reducing overall cost of the system and making potential manufacturing more affordable. It also allows us to condense and isolate the image based systems of the device from the decision making logic engine of the device, which will be implemented on a more straightforward micro-controller type implementation instead. This will keep code simpler and less co-dependent allowing for faster and more optimal testing at the component level.

Due to the complex nature of the problem at hand, namely the complexities involved with taking a high resolution image stream and translating it into usable data from which to make decisions on, there are actually several ways in which we can implement this SoC type module within the overall computer vision pipeline. The main two ways in which this can be done for our respective project exist at the low and intermediate stages of this pipeline and involve using the FPGA's mostly parallel nature to either accelerate the image preprocessing algorithms needed to clean and adjust the incoming image data to be more compliant with later algorithms such as edge detection and feature extraction, or to simply accelerate the aforementioned algorithms themselves. For our system we will be attempting to do both, creating hardware implementations through the FPGA that specialize in each sub-process and transformation and connecting them all through the HSP control circuit, attempting to implement a system that is both spatially and temporally pipelined in the process. This should allow for the fastest implementation times of the Computer Vision algorithms possible without sacrificing image quality or algorithm reliability.

Machine Learning - Neural Network for Object Detection

IVS systems need to detect threats and take appropriate action(s) with relatively small delays in order to be effective. The time constraints of visual recognition systems have motivated research on efficient classification algorithms. A machine learning (ML) approach is proposed by Lee et al using convolutional neural networks (CNNs) in a hierarchical feature model (HFM). A CNN is a feed-forward neural network that consists of a collection of receptive fields, imitating the way brain neurons are organized in the visual cortex. A neural network consists of a number of nodes each of which holds a different weight. Collectively, the differences in weight between the nodes form a pattern. Items that belong to the same class form similar patterns and therefore models of numerical thresholds for each node can form a classifier.

In its classical form, in a neural network N nodes are used to classify an object with N features. A convolutional neural network is a more efficient variation that reduces the number of nodes and therefore the computational time. A series of convolution and max-pooling layers are used to reduce the number of necessary nodes. Convolution is an operation that attempts to extract features that accurately represent the responses of nodes. As a result, similar items that are given as input to the network will generate the same classification outcome. Max-pooling is a down-sampling method that reduces the number of nodes by keeping the highest values of neighboring nodes. The highest weights, therefore these contributing more to the distinction between classes, are used to represent their respective areas and the nodes with smaller weights are dropped from the generated model.

The HFM generated by the CNN is used to create a hierarchical classifier ensemble (HCE), which ultimately performs the object detection. The object detection framework can be visualized in the figure 3-2 below:

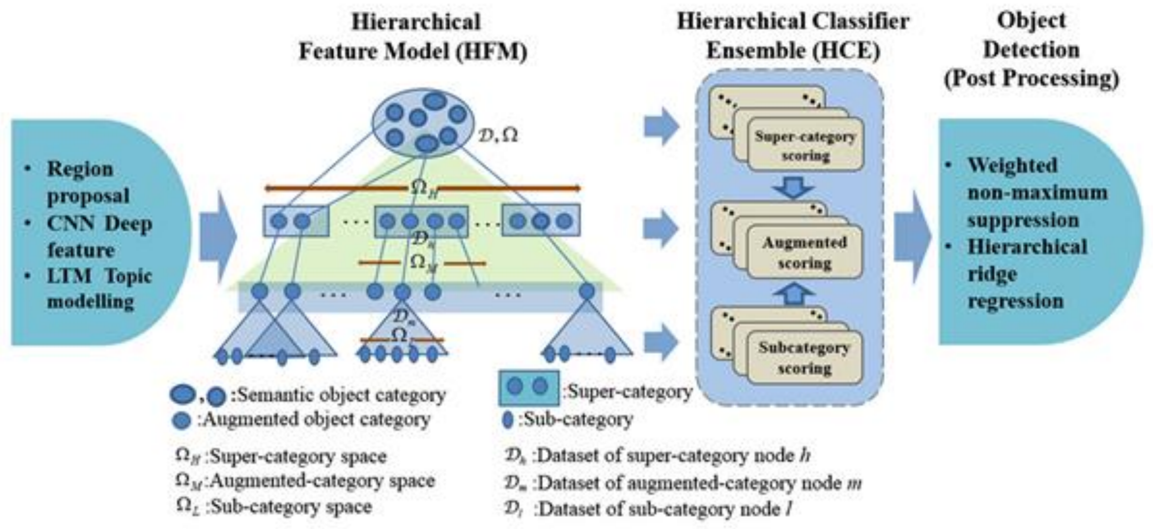

Hierarchical Feature Model (HFM)
Hierarchical Classifier Ensemble (HCE)
Object Detection (Post Processing)
Region proposal
CNN Deep feature
LTM Topic modelling
$\mathcal{D}, \Omega$
Ω_H
$\mathcal{D}_h$
Ω_M
$\mathcal{D}_m$
Ω_l
Super-category scoring
Augmented scoring
Subcategory scoring
Weighted non-maximum suppression
Hierarchical ridge regression
:Semantic object category
:Augmented object category
:Super-category
:Sub-category
Ω_H :Super-category space
Ω_M :Augmented-category space
Ω_L :Sub-category space
$\mathcal{D}_h$:Dataset of super-category node h
$\mathcal{D}_m$:Dataset of augmented-category node m
$\mathcal{D}_l$:Dataset of sub-category node l

Figure 3-2: "Proposed object detection framework based on the hierarchical feature model (HFM), and hierarchical classifier ensemble (HCE)" [4]

The authors identify three advantages of their proposed framework [4]:

- Including an augmented object category resolves issues with inter-class ambiguity and intra-class variation.
- HFM is shown to be more effective coupled with the HCE, because HFM's clustering facilitates building the HCE.
- Confusing data samples are clustered properly to sub-categories and overall detection accuracy can be improved.

As the first step of the algorithm, the regions of interest (ROIs) are detected by using the region proposal EdgeBoxes algorithm. The EdgeBoxes algorithm is implemented as proposed by Zitnick and Dollr in their "Edge boxes: locating object proposals from edges" paper. [5] The features that are present in a given ROI are generated by using a 16-layer CNN. [4] The CNN is implemented as proposed by Girshick in his paper "Fast r-cnn." [6] The normalized ROI features that are extracted can be represented in a deep-feature HFM as shown in the figure 3-2.

The resulting HFM has three different levels: the inter-class (H-level), the augmented class (M-level), and the intra-class (L-level). The root node has all the H-level nodes as children, the H-level nodes have one or more M-level children nodes, which respectively have one or many L-level leaves as children. The hierarchical classifier ensemble (HCE) "is built by training the multi-category classifier at each node of HFM, which is an assembly of one-versus-all SVMs". [4] Support vector machines (SVMs) is a partial case of kernel-based methods and is a technique that was originally intended to build optimal binary classifiers. [5] An SVM implementation is included in the OpenCV library. [5]

The system is able to learn to detect specific data-driven hierarchical categories (in our case firearms and human figure can be two possible choices) by using a latent topic model (LTM). The LTM is built by fitting a mixture model on the feature representation of a ROI that is extracted by the 16-layer CNN. The resulting LTM can summarize each ROI as a combination of K topics, where each topic corresponds to one or more super-categories (inter-class nodes). [5] The LTM analysis results in a quantitative representation of the HFM shown in figure 3-2. The category space Ω and its respective dataset D is reflected by subsets Ω_h (super-category space), Ω_m (augmented category space), and Ω_l (sub-category space) and their corresponding datasets D_h, $D_{m,\text{ and}}$ D_l.

The spaces $\Omega_{h,}$ Ω_m, and Ω_l spaces are used to train binary SVM classifiers. To improve prediction accuracy, separate classifiers are trained for each space and a combination of their scoring ultimately results in the object classification. The mathematics for producing $|\Omega_x|$ classifiers $\varphi_1 \dots \varphi_x$ for space Ω_x and their projected pseudo-probabilities are shown below:

Category	Classifiers	Multi-class margin	Normalized margin	Pseudo-probability
Ω_h	$\phi_1 \ldots \phi\lvert\Omega_h\rvert$	$\xi h(r) = P(h(k)\ \vert r) - \max P(h\vert r)$	$\varphi h(r) = A + B / (1 + \exp(-C \times \xi h(r)))$	$P(y = h\vert r) = 1 / (1 + \exp(\alpha \times \phi h(r) + \beta))$
Ω_m	$\phi'_1 \ldots \phi'\lvert\Omega_m\rvert$	$\xi' m(r) = P(m(k')\ \vert r) - \max P(m\vert r)$	$\varphi' m(r) = A + B / (1 + \exp(-C \times \xi' m(r)))$	$P(y = m\vert r) = 1 / (1 + \exp(\alpha \times \phi' m(r) + \beta))$
Ω_l	$\phi''_1 \ldots \phi''\lvert\Omega_l\rvert$	$\xi'' l(r) = P(l(k'')\ \vert r) - \max Plh\vert r)$	$\varphi'' l(r) = A + B / (1 + \exp(-C \times \xi'' l(r)))$	$P(y = l\vert r) = 1 / (1 + \exp(\alpha \times \phi'' l(r) + \beta))$

Table 3-1: Formulas to generate the classifiers for the HCE component.

Parameters A, B, and C for the normalized margin formula are determined through empirical fitting and the parameters α and β for the pseudo-probability formula are determined by logistic regression. [4]

The effectiveness of the resulting pseudo-probabilities in classifying several daily objects (for instance bus, table, and bottle) is demonstrated by the authors of the paper. The algorithm was trained and tested on the PASCAL VOC 2007 and 2012 test sets and most of their detection accuracies range between 60% and 90% based on the different targets or variations of the algorithm. [4] In addition, the CNN approach is expected to remain relatively effective when there is a problem of inter-class ambiguity, which can certainly prove an issue when attempting to detect different types of firearms. For this reason, the described object detection framework is a promising choice for the ML component of our IVS.

Summary of ML Algorithms

Because of its unsupervised nature and its ability to process large data efficiently, random forest is going to be the first machine learning algorithm that we will be implementing. At first, we will be implementing the general framework and testing it against the publicly available PASCAL datasets. An additional verification step will then be added to check for a human figure and its pose to determine the context. Finally, the implementation of the framework will be customized by testing its performance against the dataset containing firearms and knives created by the AGH university researchers [8]. At any stage of the development process, if the random forest's performance seems to decline due to inter-class ambiguity and overfitting issues, MCBoost can be tested as a more straightforward approach. On the other hand, if the random forest cannot accurately predict the classification of an object, the CNN approach can be tested as well.

OpenCV

In order to facilitate the development of the machine learning algorithm, we are using a CV library. More specifically, we are going to be using OpenCV as our library of choice. The main advantages provided by OpenCV are its ease of use for our application and its mutability. OpenCV was designed with the specific intention of high efficiency and use in real time applications. It features hundreds of functions and unique objects for use in a variety of CV applications. In addition, it features a large general purpose ML library. This library was specifically designed to be used in CV based ML problems. All of OpenCV is open source and easy to modify to fit our specific application.

The OpenCV ML Library, or MLL for short, features about a dozen ML algorithms. Though all of these algorithms work differently, they share a common set of methods that allow the user to interact with them. CvStatModel() and ~CvStatModel() are the constructor and deconstructor for the models. The constructor can also be used to train the model on construction. train() is used to train the model while predict() is used to predict the label or value of any new data. save() and load() are used to save and load a model from an XML or YAML file respectively. write() and read() do a similar task, but are the generic forms and are generally not used.

Decision Making and Alert Systems

Once a threat is detected, the gimbal, mechanical zoom and alert system come into play. If the system is unable to determine if a firearm is present or not, the mechanical zoom would be used to zoom into the object to further determine if the object is a firearm and an alert should be sent. If a threat is detected, the gimbal, would allow the camera to follow the culprit in the x- or y- plane, keeping the culprit at the center of the frame at all times during detection.

The alert system is an important component of the project. This system, after getting input from the processor, will decide whether an alert should be sent out, presence of a dangerous situation, or whether no alert should be sent, no dangerous alerts present.

Most current systems today are either provided through a security company or a Do It Yourself (DIY)/Monitor It Yourself (MIY) which have a few implementation techniques. The latter is a method in which, any individual could go to a store like Walmart and purchase an alert system kit for a one-time fee that they would have to assemble themselves at home using instruction manuals. This system opens up a communication line between the police and the customer. The first option is an alert system offered from a security company that would install the system for you with a monthly or annual subscription fee. This system would send an alert to the security company that is providing the service, who have an operator reviewing the surveillance to determine if the alert is false or not [14]. Sometimes the operator would even give a call to further investigate before the police are called. This method adds a delay to the time the police are alerted, but is used because according to Security Sales & Integration, an increase in the number of false alarms to the police leads to a decrease in the priority of response from the police to the location. In order to make sure that the location always has high priority response from the police, they use the operator.

There are many trigger techniques in use. Three popular ones are: the panic button trigger, the security code trigger, the cash tray bill trap trigger. [15] The panic button is a button that it placed at a strategic place in the location such that the police can be quickly responded to. (e.g. In a bank it is placed right under the bank teller's desk, so that if a robber enters the premises, all they have to do is extend their hand under their desk and push the button). The security code trigger, is a code, separate from your alarm's Personal Identification Number (PIN) code that is to be entered into the alarm system's keypad, used to send an alert to the police in case someone threatens you to disable your alarm system. The cash tray bill trap trigger is also convenient as it alerts the police, when a bill is removed from the trap that easily inserts into a bank teller's cash tray.

Our alert system to have a high priority response from the police with as few false alarms as possible. Our system can be implemented in a number of ways, but the way we envisioned it was for it to automatically and directly alert the police without an operator. This part of the system would run in parallel with the camera detection, as the location needs always to be monitored and kept secure. The alert system would simply have two actions, if a signal is sent to it, saying there is an alert, the system should be able to wake and send an alert to the police, in either an SMS form or prerecorded form. Otherwise if nothing is detected and sent, the alert system should be asleep. Unlike the other systems with their manual triggers, our system will to be automatically triggered by the detection of a presence of a firearm, like a pistol, and irregular human pose, like the perpetrator, holding the gun and people raising their hands. The alert system would not conduct any process but would take action solely based on direction from central processing.

Power Considerations

The whole system is going to need a power source and there are two orientations that would be able to provide the system the power we need, a self-contained system or a system receiving power directly from the building's electricity. For the self-contained system, this would need batteries that would power the system, an advantage to this is that the system would be able to run if the main power in the building is off and the power source would be a direct DC source, the disadvantage to this is that the system's battery, however low power we would like it to run, can only supply a finite amount of power and would need changing, this introduces a time frame in which the system would not be running, thus there would be no surveillance for a period if the batteries run out. Given the DC voltage being supplied there would only be a need to amplify the voltage. Now using the other system orientation, the system would be connected directly to the mains of the building and would run with the power supplied to the system from the building, an advantage to this is that the system would not need attendance to change the power supply as it would be constantly receiving the power it requires and would never run out as is the case in using a battery, thus the system would always be able to run its surveillance features and keep the location safe. A disadvantage to this system would be that given the power is coming straight from the building's main power supply, the voltage would be in AC voltage and the components require DC voltage, thus another component would be needed for the system, an AC to DC converter. This component would do exactly what it says, this added component would be able to convert the AC power supply from the mains of the building to the DC current that the systems components require to operate in. This would make sure that the components are receiving the correct input voltage, instead of damaging the system.

Mechanical Components: 2 DoF Gimbal

Our entire system will be mounted on a 2 degree of freedom gimbal to allow the camera to sweep a larger area and focus on a threat when one is detected. A change of a few degrees in the camera's angle can mean a drastic change in the scene that is being viewed. In order to facilitate this high degree of accuracy that is needed and to also prevent the gimbal from moving when not intended, we will use a worm drive on both axes. This gear arrangement consists of a cylindrical worm screw driving a larger worm gear. It creates a very large mechanical advantage and has the unique quality of being unable to be back driven, which means that only the motor can drive the system, not the weight of the system itself. This will allow the gimbal to hold its position without needing to use power.

Mechanical Components: Mechanical Zoom

In addition to the gimbal, we will potentially be implementing zoom functionality to the camera. In order to implement this, we intend to use a third party varifocal lens designed for use on CCTV systems. A varifocal lens is a camera lens that is able to vary its focal length, but does not maintain focus as the focal length changes. These are commonly used with modern day cameras as a majority of cameras have some sort of auto-focus feature that can maintain focus as the focal length changes. For many varifocal lenses, the zoom and focus features are adjusted manually by rotating parts of the lens. These can be easily automated by attaching either a simple wheel or belt between the lens and a motor.

Verification, Testing, and Methods of Analysis

Data Gathering

The effectiveness of an IVS system that uses a machine learning algorithm largely depends on the quality of the data sets available. The data sets are necessary for both the development stage, where data sets are needed to train the algorithm, as well as the testing stage, where parameters of the system can be re-configured based on test results. When collecting representative data of our events we also need to create the corresponding groundtruth. "This groundtruth describes the expectations for the system." [9] For instance, the appropriate groundtruth for the machine learning algorithm would indicate the correct events that should be detected and their appropriate responses. The corresponding groundtruth of a given data set can vary depending on whether it is used for training or testing purposes.

The almost infinite range of possible inputs to the recording device "makes completely exhaustive testing of vision algorithms virtually impossible" [9]. Since all computer vision algorithms process frame by frame, our data set can include both static images as well as video streams and both should prove useful. In the early stages of development, we can use data sets that are publicly available. The researchers from the AGH university generated a dataset that "consists of 12,899 images" and their corresponding groundtruth "divided into 9340 NE and 3559 PE images" (NE = negative events, i.e. no firearms should be detected, PE = positive event, i.e. firearms is present) [10]. The AGH research team also generated a "training and testing set were the same size, with 8.5 min of recording resulting in approximately 12,000 frames each" [10]. These publicly available data sets will be mainly needed during the early development stages and should be replaced by (or merged with) larger data sets as well as data sets that we choose to develop.

Verification and Testing

The data sets can also be used to test and verify the effectiveness of an IVS system. The algorithms can be simulated with test images or video as input and its output compared with the respective groundtruth. However, because "the system has to perform predictably across an intractable number of scenarios and environmental conditions", we will need to work with development tiers. [9] The types of scenarios and environmental parameters are fixed in the first developments tiers, but become more relaxed in next tiers. In addition, we need to "ensure correct and predictable software performance when porting technology between algorithm development environments (such as Matlab and high level programming languages like C# and C++) and product deployment code environments such as low-level C-code with processor-specific optimizations" [9]. We can ensure that by testing the IVS system as a complete unit. The corresponding unit test will look at whether the output from the input data matches its corresponding groundtruth. Unit tests will not provide any useful metric for evaluation purposes, "instead, the expectation is that each test should pass." [9]

The overall performance of our system can be evaluated by measuring either the false alarm rates per hour (FA/Hr) or the probability of detection (P_D) of an event. Calculating FA/Hr can be more challenging since "FA/Hr should only be measured on long videos with no or only a few real events, representative of the real deployment." While the AGH team's data can be used to calculate P_D, different data sets will be needed for to acquire more verification metrics. In later (more technically demanding) development tiers, the groundtruth can also be replaced by a fuzzy groundtruth, where more than one representations of the data are accepted as correct. For instance, in frames when a person is changing between poses, both the previous and next pose can be accepted as correct readings of the data. In addition to ensuring overall performance by unit testing, we also need to test each block of the IVS system separately. When changing parameters and adding functionalities these tests can help reach better metrics by identifying which module has a lower overall performance. [9]

For the background generation module, the groundtruth needs to appropriately merge marked areas of the foreground that were previously of interest to the background when they are no longer relevant. [9] For example, when a person leaves the premises of a bank, but is still visible, the foreground should not include the person as they cannot pose a threat anymore. To evaluate the performance of the background generation module we can use "the percentage of correctly detected foreground pixels, and the number of pixels falsely labeled as foreground." [9] Based on the development tier, the groundtruth should appropriately reflect the expectations from the algorithms. In addition, the groundtruth can also be replaced by a fuzzy groundtruth, where more than one representations of the background are accepted as correct. For instance, at the exact frame when a person exits the area of interest, the groundtruth should accept the person being included in either the foreground or the background.

The blob generation module can be tested by calculating P_D, the probability that a groundtruth blob is matched to any blobs detected from the algorithm. Researchers Venetianer and Dent "recommend a non-linear weighting to compute Pd: if the groundtruth and detected targets overlap by more than a certain percentage, it is considered a perfect detection" [9]. Fuzzy groundtruth can also prove useful for the blob generation module, as an error of few pixels in small distant blobs can result in no overlap but still penalize the system for not detecting unimportant blobs.

Testing the tracking of targets can be done by matching each track from the groundtruth to the closest one detected. If the track detected is closer than a threshold, meaning that the target was accurately tracked, this target is marked as successfully tracked. The performance metric is therefore a percentage of the groundtruth tracks being matched. Finally, the classification component can be evaluated based on how many of the detected blobs were correctly assigned with their respective event.

All testing will be conducted in two stages: performance testing and regression testing. Performance testing evaluates the effectiveness of our IVS system. Performance testing ensures the basic tier requirements are met. The unit tests that will be used for performance testing are expected to always be successful. Regression testing ensures the performance of our system remains consistent as we add more functionalities to our system and as we move to next development tiers and relax assumptions. During regression testing, the metrics from the component testing will be used as a basis to appropriately revise parameters as needed. The ultimate goal of both stages of testing is to increase the overall performance of an IVS system and ensure it remains consistent.

Project Timeline, Goals and Budget Information

Operational Milestones and Stretch Goals

Our team has decided to structure the organization and development of our project using a two-pronged approach. As such, the main modules and functions required to implement our system have been broken into 2 separate categories: Softcore Implementations, including all filters, accelerators, extractors and any other pre-processing type tasks that can be implemented on the FPGA logic, and Hardcore Implementations to be implemented across the Hard System Processor of the SoC, including the C, OpenCV functions, machine learning algorithms and the basic state based control signals. From this break down we have assigned two primary teams, one devoted to hardware language (Verilog) programming and one devoted to system level C coding (OpenCV).

Due to the complexity and scale of each module in the system, we've opted to have both teams work in tandem, developing different but related parts of the system and essentially working towards each other and a full system integration. As such, each team has various milestone goals to achieve within their respective focus.

For the OpenCV team, these Milestone are:

- Implementing a Basic Non-specific machine learning algorithm on desktop
- Refining it to focus on a specific image profile
- Generating a training profile based on a set of gathered gun video data
- Scaling it back to work and operate on embedded logic
- Identifying Areas of possible bottleneck for hardware acceleration
- Integrating it into the full system pipeline.

For the Verilog team, these Milestones are:

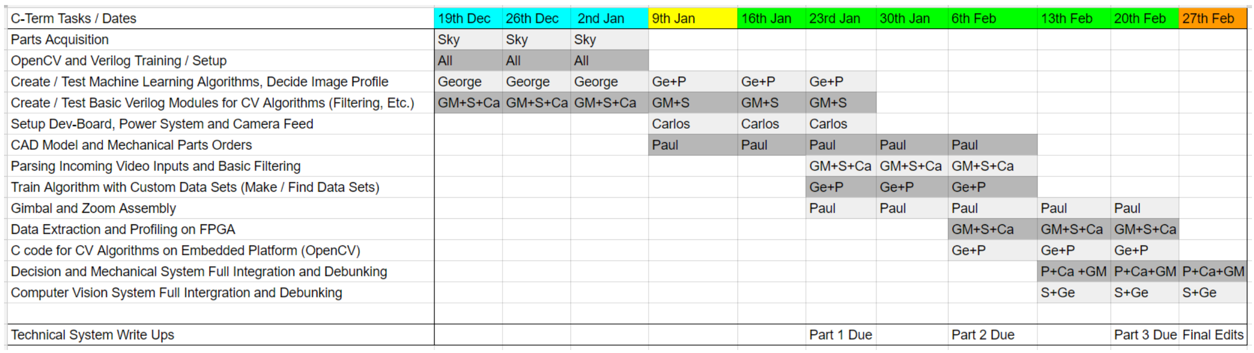
- Translating General Purpose image filters and operations to Verilog Modules
- Combining modules together to pre-process the video stream
- Test the Verilog modules for accuracy and image quality
- Implement hardware acceleration resources for Serial Process Acceleration

- Integrate into the full system pipeline.

These submodule milestones then lead into our total project goals. At the bare minimum, we hope to have a working system that can identify a generic, easy to detect object (such as a pink square), track that object and send out a warning signal in real time. From there, we'll move on using real guns to train the machine learning algorithm, with the intent to detect a specific and easy to isolate gun type such as handguns or revolvers.

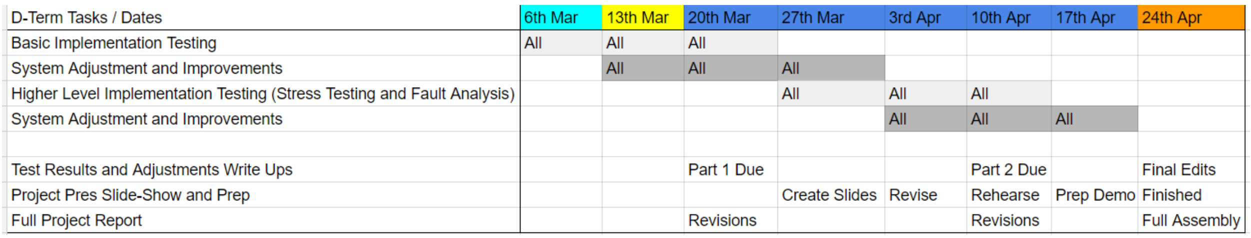
Implementation Timeline and Gantt Chart

We sat down and drew out the plan for the upcoming 2 terms, C term and D term. As can be seen in figures 4-1 and 4-2 below, our Gantt chart's time is based on each week. The main tasks of our project are broken down into a hardware component or software component to be fulfilled within the number of weeks allotted to the tasks. The dates are color coded to indicate the different times within the next 5 months of the project's lifespan. The cyan blue dates are weeks in which school is not in session. The yellow dates are weeks which school resumes. The green (C term) and dark blue (D term) dates are those in which school is in session. The orange dates are the weeks in which school goes on a break (C Term)/ends for the academic year (D Term). At the bottom of each Gantt chart there is a section for our deliverables and when we would want to be working on them or finishing them.



C-Term Tasks / Dates	19th Dec	26th Dec	2nd Jan	9th Jan	16th Jan	23rd Jan	30th Jan	6th Feb	13th Feb	20th Feb	27th Feb
Parts Acquisition	Sky	Sky	Sky								
OpenCV and Verilog Training / Setup	All	All	All								
Create / Test Machine Learning Algorithms, Decide Image Profile	George	George	George	Ge+P	Ge+P	Ge+P					
Create / Test Basic Verilog Modules for CV Algorithms (Filtering, Etc.)	GM+S+Ca	GM+S+Ca	GM+S+Ca	GM+S	GM+S	GM+S					
Setup Dev-Board, Power System and Camera Feed				Carlos	Carlos	Carlos					
CAD Model and Mechanical Parts Orders				Paul	Paul	Paul	Paul	Paul			
Parsing Incoming Video Inputs and Basic Filtering						GM+S+Ca	GM+S+Ca	GM+S+Ca			
Train Algorithm with Custom Data Sets (Make / Find Data Sets)						Ge+P	Ge+P	Ge+P			
Gimbal and Zoom Assembly						Paul	Paul	Paul	Paul	Paul	
Data Extraction and Profiling on FPGA								GM+S+Ca	GM+S+Ca	GM+S+Ca	
C code for CV Algorithms on Embedded Platform (OpenCV)								Ge+P	Ge+P	Ge+P	
Decision and Mechanical System Full Integration and Debunking									P+Ca +GM	P+Ca+GM	P+Ca+GM
Computer Vision System Full Intergration and Debunking									S+Ge	S+Ge	S+Ge
Technical System Write Ups						Part 1 Due		Part 2 Due		Part 3 Due	Final Edits

Figure 4-1: Gantt Chart for C Term



D-Term Tasks / Dates	6th Mar	13th Mar	20th Mar	27th Mar	3rd Apr	10th Apr	17th Apr	24th Apr
Basic Implementation Testing	All	All	All					
System Adjustment and Improvements		All	All	All				
Higher Level Implementation Testing (Stress Testing and Fault Analysis)				All	All	All		
System Adjustment and Improvements					All	All	All	
Test Results and Adjustments Write Ups			Part 1 Due			Part 2 Due		Final Edits
Project Pres Slide-Show and Prep				Create Slides	Revise	Rehearse	Prep Demo	Finished
Full Project Report			Revisions			Revisions		Full Assembly

Figure 4-2: Gantt Chart for D Term

Hardware Resources

Proposed camera solution:

After taking into considerations the specifications that were needed and after selecting which FPGA board is going to be used. A final decision was made in terms of the camera component. The camera that was picked is the ELP 2.8-12mm varifocal lens with 2MP. This camera has knobs that be adjusted which would fulfill the mechanical zoom component of the system. The lens is between the ranges of 2.8 to 12mm as stated above. The camera is powered by USB, the development board chosen has an USB input, this means that no extra module will be needed to communications interface between the camera and the board. The camera outputs file in MJPEG format which is ideal since it is widely used. However, its compression is not as good as others but the resolution that the camera would be used would not be highest it can provide. The camera uses a CMOS sensor since most commercial cameras have this type of sensor due its manufacturing price. The camera is also multi-colored which is one of the requirements that the system needs. If we encounter a problem with this camera during our implementation, two other options have been selected as our backup plan. The OV 5460 which is an fpga camera module with a 5MP pixel resolution however is does not have adjustable lens. To overcome this an extra lens that can be adjusted mechanically would be bought and attached. This module outputs an 8/10 bit raw RGB and at 1080p it can record at 30fps. This camera is better in terms of resolution however the varifocal lens camera provided a lot more in terms of the mechanical parts of the system.

Proposed FPGA Development Board Solution

The proposed FPGA development board chosen is the DE1-SOC DE Board from Terasic. The board has the latest Altera SOC chip the cyclone V. Which is optimized for video processing implementations. It has 85,000 flip flops and 4,450 Kbits of embedded memory. In terms of memory the development board has 64MB of of SDRAM which is going to be used to store the image streams that the camera is capturing. The camera chosen output size means that a 4 minute long video will have a size of 20MB. Which is very good considering the retail price of the board. In terms of the communications that this board has it contains: two USB ports , one USB to UART , 10/100/1000 Ethernet, PS/2 mouse/keyboard and an IR Emitter/Receiver. For the connectors in the board we have Two 40-pin Expansion Headers (voltage levels: 3.3V), One 10-pin ADC Input Header and a One LTC connector (One Serial Peripheral Interface (SPI) Master ,one I2C and one GPIO interface). It requires a 12V DC input. Overall this FPGA development board should satisfy our needs.

Software Resources and Licensing

The current software that will the utilized will be Visual Studio 2015 as our c++ IDE. For image processing and machine learning the OpenCV database will be used. This database is the best open source computer vision database that has been used and updated over the period of 10 years. The library will be linked to the IDE to be able to test the algorithms before the parts arrive at the start of C term. OpenCv version 3.1 for c++ will be used although there are c, java and python libraries as well. In addition the PASCAL VOC and AGH datasets will be used to start training the machine learning algorithms. The first dataset contains common objects found in everyday life such as cars, houses, bicycles, etc… The second dataset contains weapons.

Total Budget Review and Expenditures

According to the MQP guidelines every team member is allowed $250 to spend on parts for the project on hand. Table 4.1 shows the breakdown of our final expenses.

Table 4-1: Budget Breakdown

Weapon Detection Budget BreakDown	
Total Budget	$1,250.0
DE1-SoC DE Board from Terasic(Academic Discount)	$175.0
Camera	$60.0
Motors/Servors	$10.0
Total Spent	$245.0
Extra Expenses Left	$1,005.0

As we see can see from the table above most of the components for the system are already bought. The only part that is not considered here are the gimbal 3D-printed parts. Most of this parts will be done using the 3D printer in the robotics department. The budget allocation is good we have a good cushion if some problems with parts arises or any other inconvenience.

Background Conclusion

All the individual components for the system have been decided. The decisions were made after doing the necessary background research. Firstly, we decided that an FPGA would be ideal to acts as a hardware accelerator for the image processing algorithms being implemented. The system will be implemented as SoC. After the FPGA was designed, the camera was selected taking into consideration the specifications datasheet of the board. The chosen camera required power through USB input. In addition, other options for cameras were presented in the event that an unexpected problem arises with the current camera setup.

For the machine learning algorithms, 3 were investigates thoroughly. It was concluded that the preferred option would be a random forest however depending on how the implementation goes the algorithm might change to a Multiple-Classifier Boosting or a Neural Network.

After researching and selecting the main components of the weapon recognition system, the mechanical aspects of the system were chosen. A Gimbal with two degrees of freedom and varifocal lens will be implemented. The Gimbal will be almost on its entirety 3D printed however some parts would need to be built in a machine shop. Milestones and stretch goals were also set for the team to have a clear understanding of the status of completing the system. These will serve as guidelines however they are not set in stone. The stretch goals will only be implemented if the base case of the system is done and there is extra time to implement them.

In addition, a Gantt chart based on the milestones was created clearly showing the different goals that need to be completing by particular dates. This again will help visualize where we are in the project and if we are behind or ahead of the time frame. Finally, the budget was written clearly showing a breakdown of the parts that were bought and their respective prices.

Design Specifications

Top Level System Block Diagram

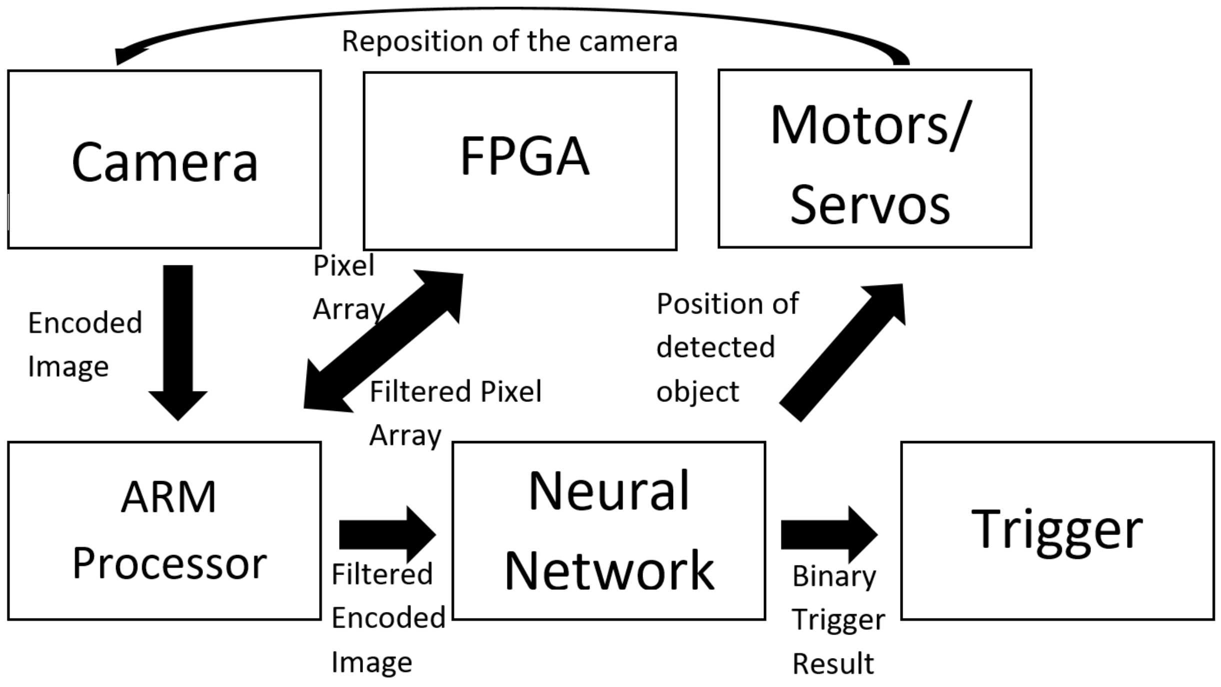


Figure 6-1: Top Level System Block Diagram

Our system takes frames from a camera which are decoded and sent as pixel arrays for filtering to be done on them and sent back. The filtered pixel arrays are then passed to a neural network which then detects if a weapon is present, activates a trigger and sends a signal to the motors or servos to change the scope for the frames.

Camera

Figure 6-2: ELP 2.8-12mm Varifocal Lens 2.0megapixel Usb Camera[31]

For the finalized design of the project, the ELP 2.8-12mm Usb Camera module from ELP was selected to its small profile, high quality varifocal lens, and native support for communication with Linux based operating systems. The camera itself operates under UVC communication protocols allowing for easy driver-less integration with most systems that already support them within their USB driver/handler, including most modern open source Linux OS kernels.

In order to integrate the camera with the chosen development board, the DE1-SoC from Terasic, certain additional Linux resources are required. The main tools of integration needed are:

- a frame grabbing application designed to communicate with the camera and save both multi-frame video streams as well as single frame images. Due to the nature of our implementation, this frame grabber must be fully deployable from the commandline or included as pre-installed function libraries.
- an integrated ISR module designed to run the frame grabbing application and store visual information in the correct updating cyclical structure as well as pass that structure's address and information to the main line system.
- a calibration module that runs at startup with the frame grabber to ensure that the camera

itself is operating in the correct data compression and resolution settings, in this case 1080 x 720 pixels under H.264 encoding.

Each of these three additional resources are implemented to ensure proper storing and accessing of the video data and are installed in the custom uBoot kernel prior to startup.

Mechanical Components

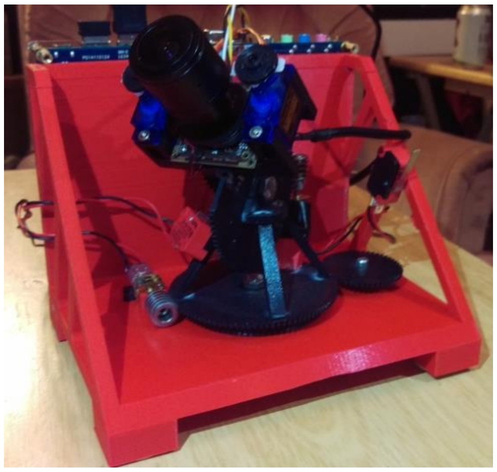


Figure 6-3: Full Assembled Gimbal

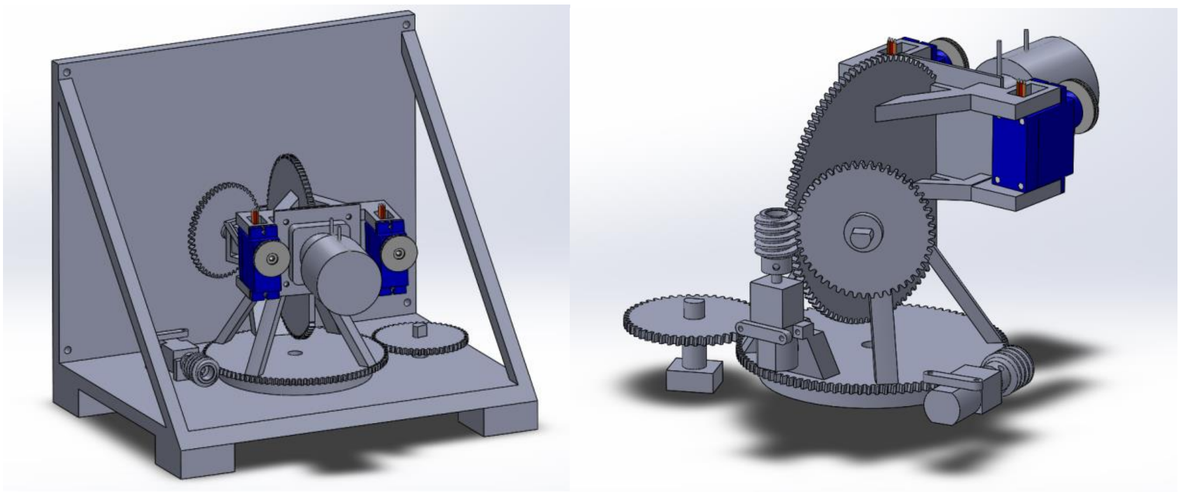


Figure 6-4: Full CAD model of the mechanical components of the system.

The finished model is a 5in by 5.5in by 6.63in box with the gimbal inside. The gimbal has an approximate range of 60 degrees vertically and 50 degrees horizontally. The two main axis are controlled by worm drives geared 1:98. This number was derived after determining the worm being used in the drive and the size of the gears for the drive. The gears were determined to be 3in diameter based on the size of the camera. The worm being used had .06in long teeth and a pitch of 32. From this information, we can calculate the pitch diameter of the gears, 3.06in, by adding the diameter of the gears without teeth with the length of the teeth. The number of teeth on a gear is the pitch diameter multiplied by the pitch, in our case 98. Because the worm used has a single helix, the worm acts as a single toothed gear in this system. This results in the mechanical advantage mentioned earlier of 1:98. The motors being used are rated to spin at 100 RPM at 6V. When supplied the 3.3V from the board, the motor will spin at 55 RPM. This will make the axis of the gimbal spin at 0.56 RPM, or about 3.4 degrees per second. This is slow enough to allow the gimbal precise control over the position of the camera but fast enough to effectively track an object moving across the screen.

The gimbal is divided into three main parts, the base, the horizontal gear piece, and the vertical gear piece. Each of these parts had some combination of motors, servos, encoders, and the camera mounted onto them. In addition, each of these parts are attached to the others by a single screw, allowing them to rotate in a single dimension relative to the part they are attached to.

Base

The base acts as the main mounting point and housing for the rest of the parts. It features mounting points for the horizontal gear piece in the center, a motor to the side of the horizontal gear piece, an encoder to the other side of the horizontal gear piece, and the main processing board on the back. The base has four feet holding it .5in off of the ground to allow room for fasteners to hold the horizontal gear piece, motor, and encoder in place. There is also a simple resting block to assist in holding the motor still.

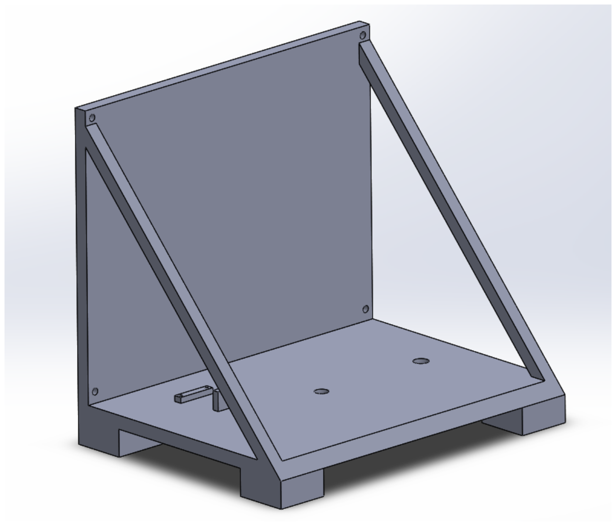


Figure 6-5: CAD model of the system's base

Horizontal Gear Piece

The horizontal gear piece controls the left/right movement of the camera and holds the vertical gear piece, a motor, and an encoder. All of the structure of the gear piece is mounted on top of a 3in worm gear driven by a motor on the base. The smaller of the two structures holds the motor controlling the vertical gear piece. The larger structure holds the vertical gear piece and an encoder to track the vertical gear piece's movement.

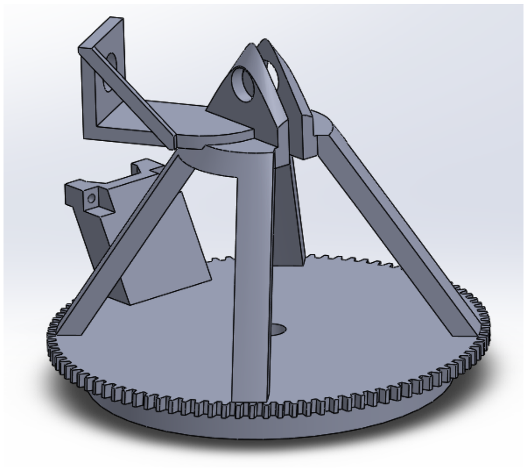


Figure 6-6: CAD model for the horizontal gear piece

Vertical Gear Piece

The vertical gear piece controls the up/down movement of the camera and holds the servos controlling the camera's zoom functionality. Space is left behind the camera to allow room for the wires that attach to it. The two servos are offset from each other so that one can control the outer ring of the camera, the zoom, while the other controls the inner ring, the focus.

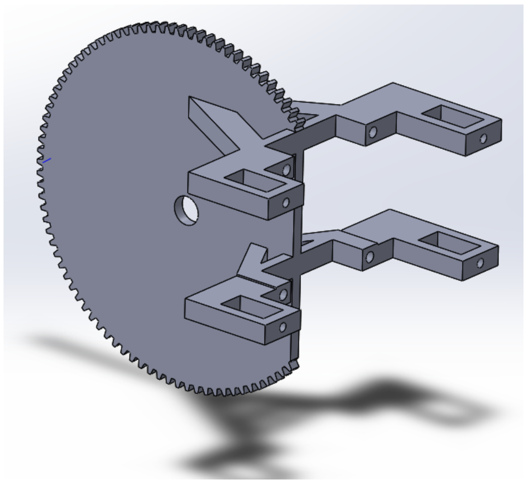


Figure 6-7: CAD model of the vertical gear piece

Going from Digital to Physical

Due to the complexity of the parts, all of the parts were 3D printed. Many of the sections, especially the horizontal gear piece, had to be broken down into smaller parts to facilitate printing. The parts were then glued back together to get the finished assembly. In addition, nuts were glued to the vertical gear piece to provide a solid mounting point for the camera. The resulting assembly was strong, but also very lightweight. As a result, the assembly unfortunately was blown off of a table a couple days before the pictures below were taken, resulting in some parts not being in place.

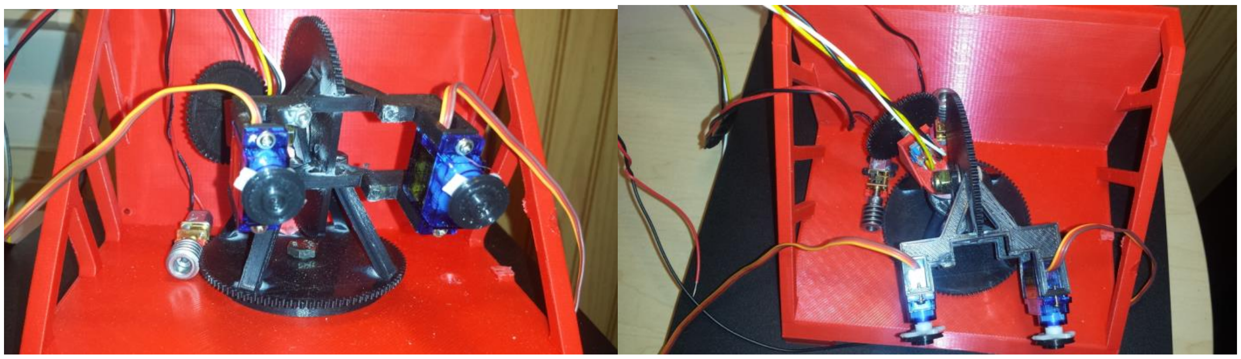


Figure 6-8: Images of the final assembly of the mechanical component

As can be seen in figure 6-8 above, some parts were not printed exactly as they were modeled. The most obvious of these is the base. The base itself is shorter than the digital model and it has additional supports on the side. These changes were done due to the limitations of the printer being used. The base was slightly too tall for the printer and the supports were needed so that the part did not collapse while printing. Other than these and a couple other extremely minor changes, the whole printing process was fast and easy for our application. The parts ended up coming together quickly once printed, resulting in our main delays being the time needed for the printing itself and for ordered parts to come in.

Sensors

Connected to both of the axis are encoders. The encoders are each geared with gears half the size of the axial gears, resulting in a mechanical advantage of 2:1 to the axis themselves. This is the same as 1:49 to the motors in the case of the encoder for the vertical axis. The primary use of the encoders is to prevent the gimbal from driving either of the axis too far and breaking itself.

In addition to the encoders, each axis has a limit switch at one of the two outer limits of the axis. For the vertical axis, the switch is at the lower limit, while for the horizontal axis the switch is at the left limit. When initializing, the system first drives the vertical axis down until it hits the lower limit to determine its location, using the value of the encoder here and the known range of the gimbal to later prevent oversteer. Next, the system drives the horizontal axis left until it hits the left limit to do the same with the horizontal axis.

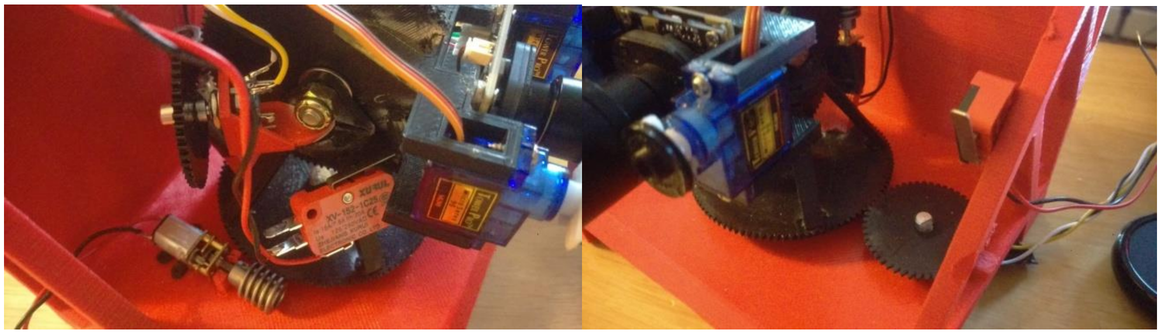


Figure 6-9: Images of the encoders and limit switches as they are attached in the system

Image Preprocessing

Even though the specifications of our camera and the quality output of the camera are great, when using this video feed for identification of the weapon, smaller frames of the image are used. These smaller frames used are a zoomed in section of the entire frames and as such are a more pixelated version of the image and it is hard to differentiate where edges are created. Also, at a distance similar color start to blend in with each other and are harder to differentiate. With the three filters we are using, the pre processed feed would make it easier to detect weapons by the artificial intelligence (AI).

In order to eliminate a substantial amount of time, given the 3 term limit for our project, we are using HDL Compatible Matlab Simulink blocks that are that would then be able to be used with Matlab's HDL Coder to generate Verilog code that can then be turned into .SOF files using Quartus Prime Software that can then be run on the Altera FGPA Board. Simulink provides a visual approach to system design, letting us visually see the flow of data in the system. Since Simulink has an HDL coder pre-installed with various pre-made function algorithm blocks, it allows for a quick turnaround from vision algorithms and filter concepts to HDL implementations on our Altera board.

Camera Interface

Once the camera feed is processed and accessible by the board, the FPGA would be able to run Verilog code that runs the filters on the feed to produce a more refined image that would make it easier for the artificial intelligence to identify a weapon.

HPS and FPGA Communication

We came to realize that it was not as direct as we had thought it was to use the camera feed input on the operating system, on the FPGA. In doing so we would have to create a way for the FPGA to obtain the image files for the image filters to be run on and sent back to the operating system.

Processing Filters

When first tackling the image preprocessing, we set of to model it in Matlab first, using Simulink blocks. This was a test to see if the filters we were choosing were going to give us the desired effect we wanted. We set up blocks that were not necessarily compatible with the HDL Coder, but would be able to give us some results for the filters we were choosing to use. We demonstrated color space conversion, image sharpening, contrast correction and foreground detection which can be seen in the following figures 6.9 to 6.12.

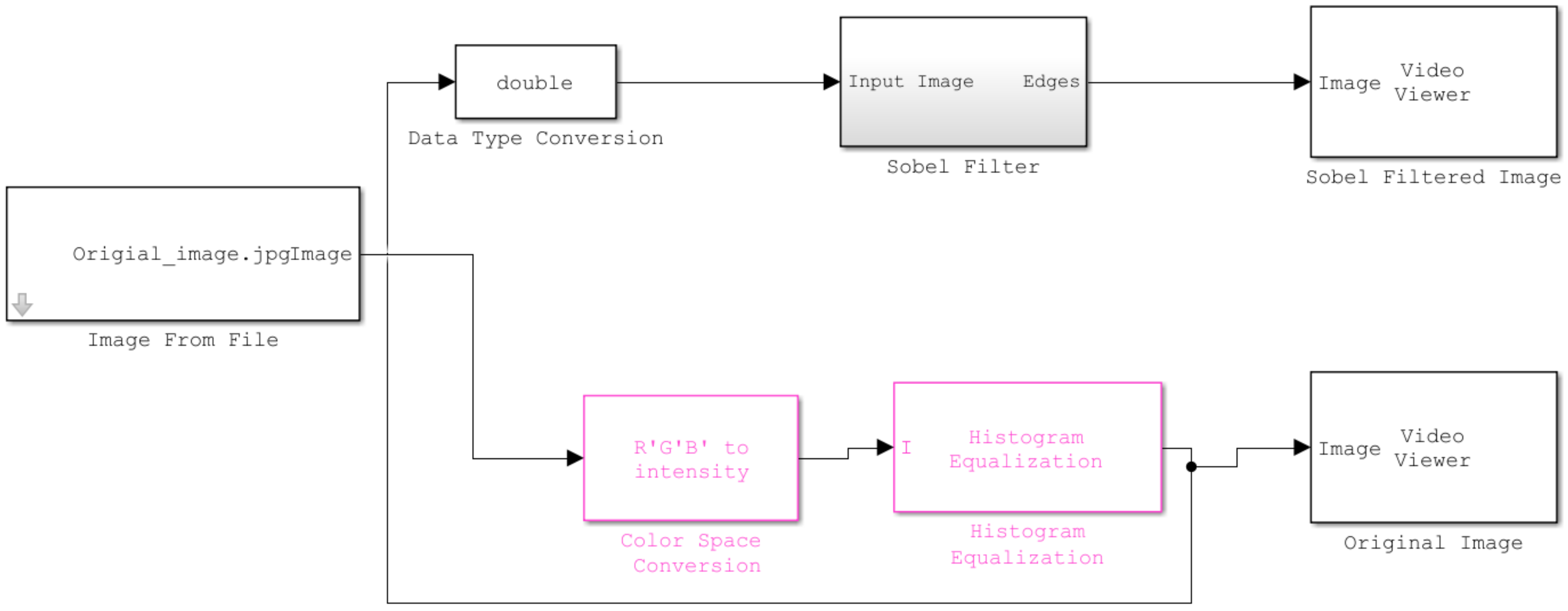


Figure 6-9: Image of the Simulink blocks for color space conversion, contrast correction and Sobel filter [30]

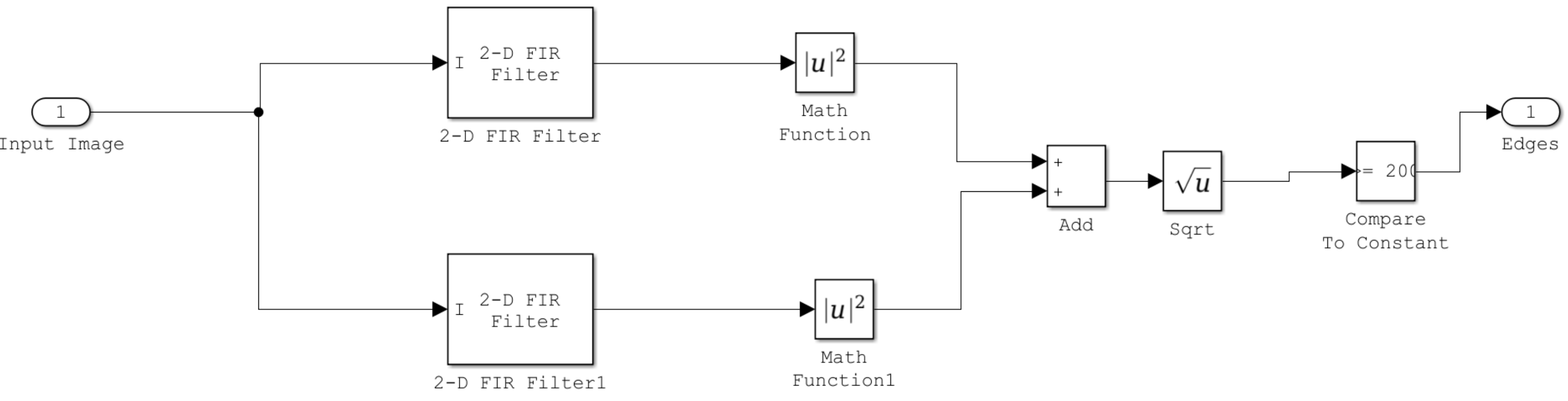


Figure 6-10: Break down of the sobel filter subsystem block [30]

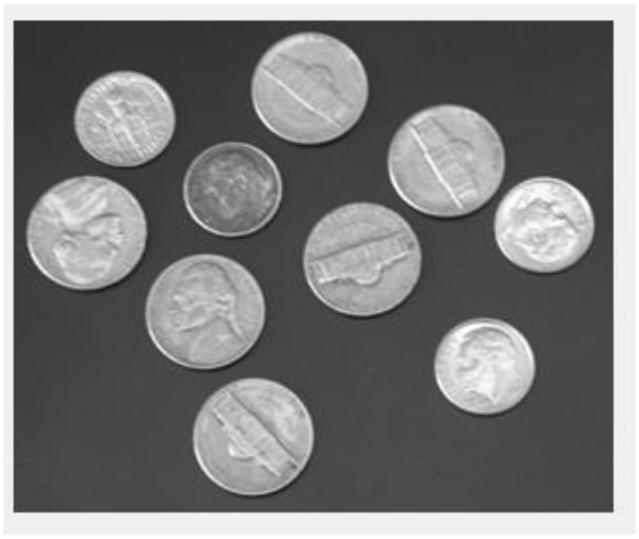


Figure 6-11: Original image for the edge detection filters

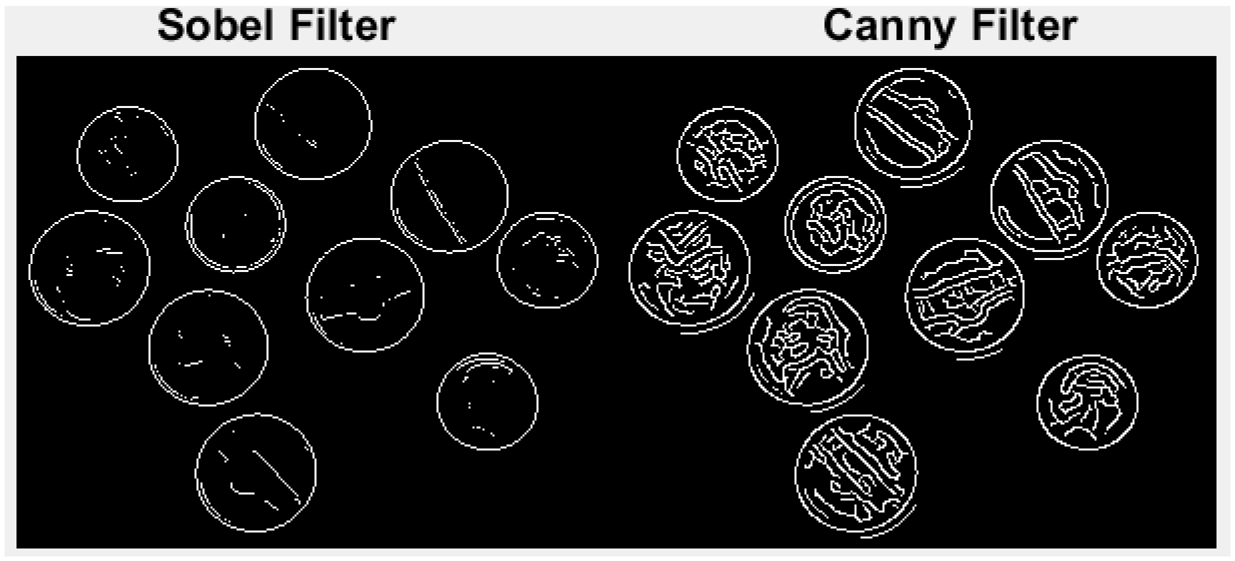


Figure 6-12: Pictures shown for the applied Matlab example of sobel (left) and Canny (right) filters

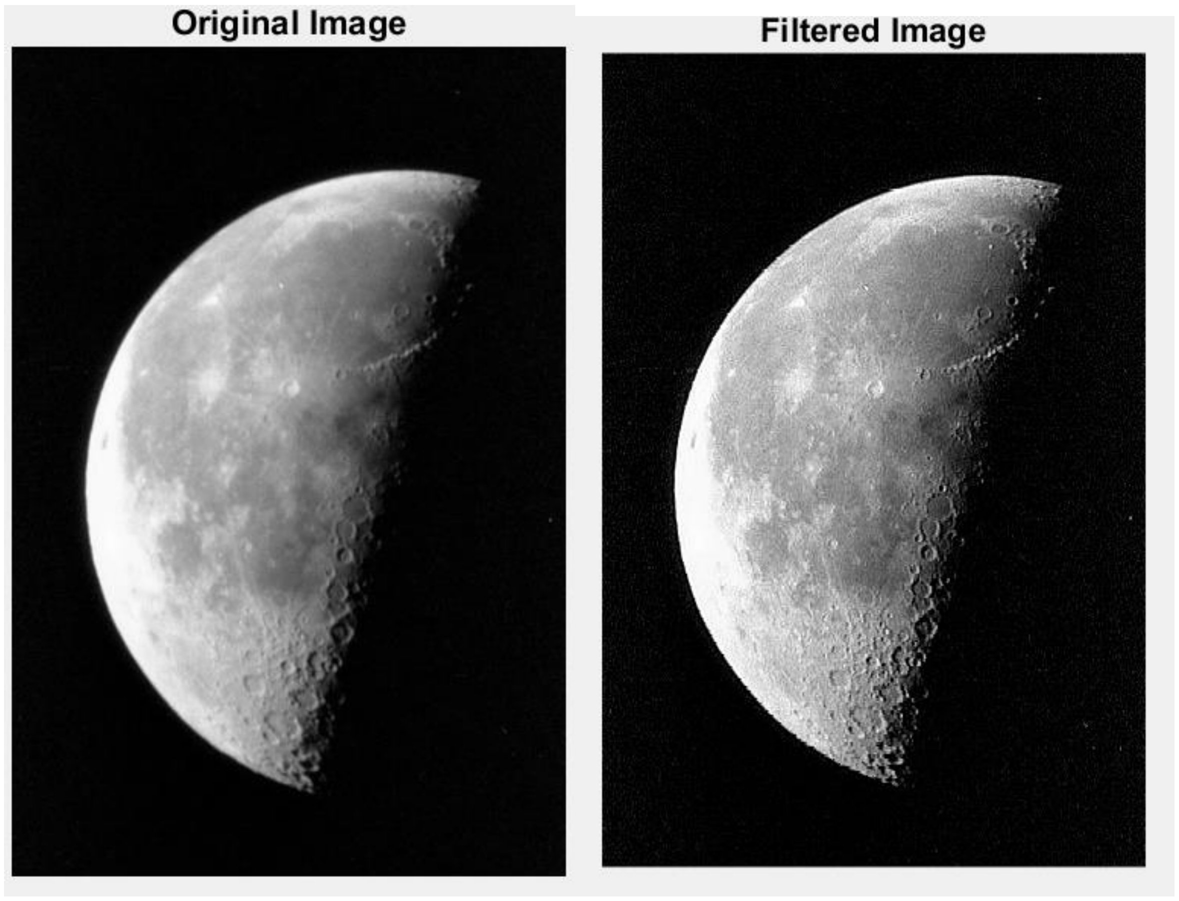


Figure 6-13: Original (left) and filtered (right) picture for sharpening

We have narrowed down to three filter processes needed to enhance the image and make weapons more defined and easily detectable. The processes are contrast correction, edge detection and foreground and background detection.

Contrast Correction

A contrast correction filter would be necessary as it would be able to further identify the different objects that lie within a limited number of pixels when detecting in a smaller frame.

The histogram equalization is used to correct contrast in the image, this would help in cases of low light. This accepts an image and spreads out the most frequent intensity values to help better distinguish objects in an image. The Matlab Simulink block model design can be seen figure 6-14 and an example of its input and output can be seen in figures 6.15 and 6.16.

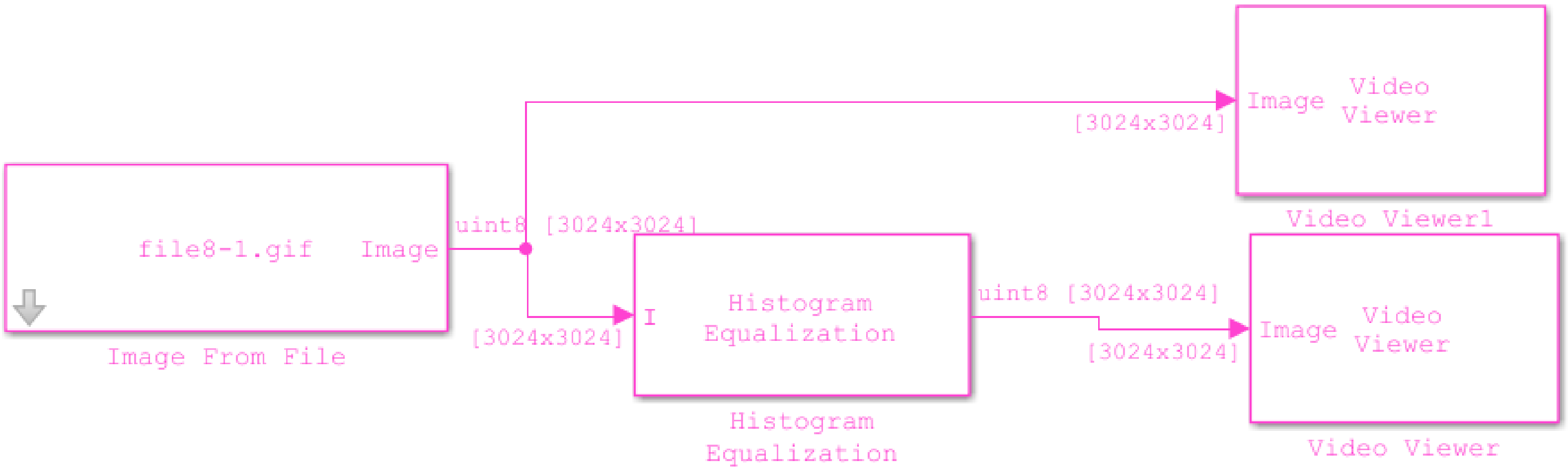


Figure 6-14: Matlab Simulink blocks for the contrast correction

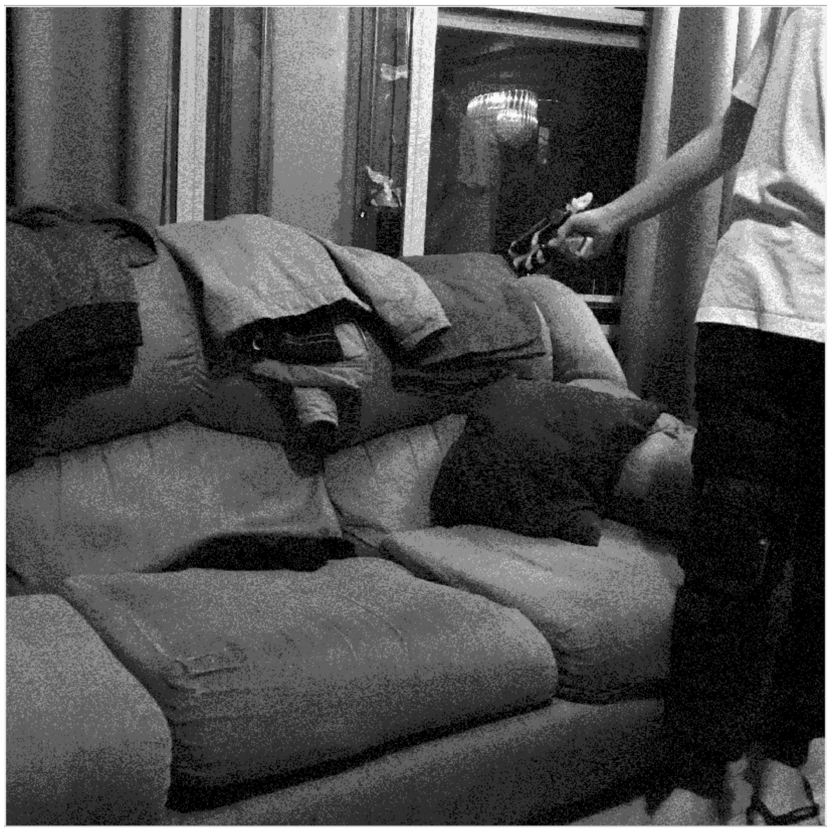


Figure 6-15: Original image for contrast correction

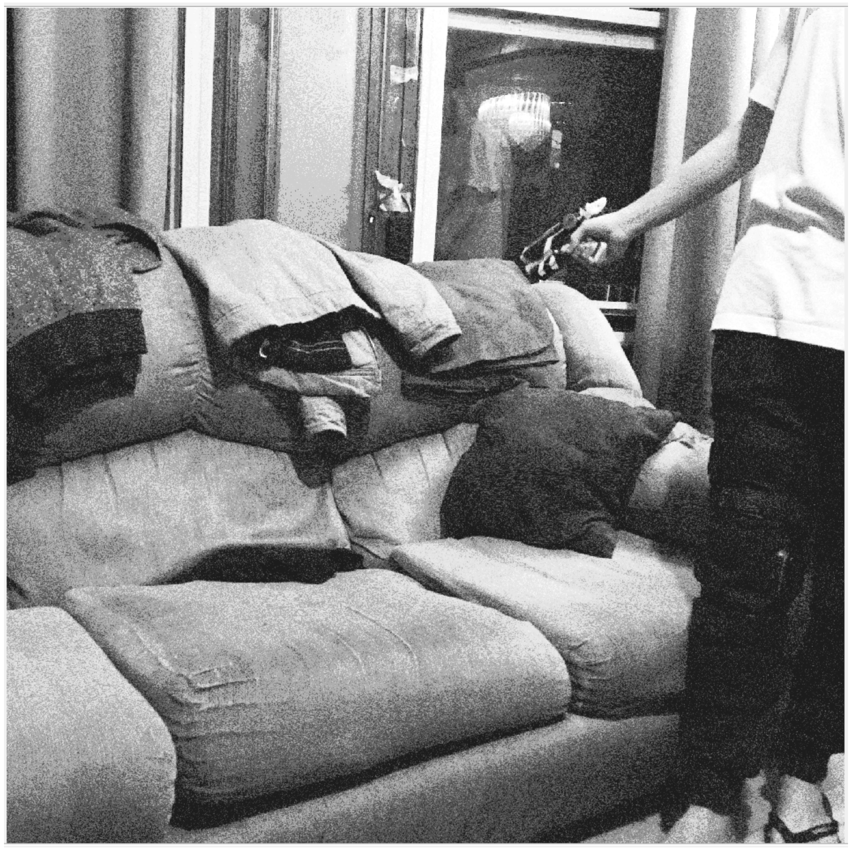


Figure 6-16: Contrast corrected image

Edge Detection

There were two filters that we were considering: Sobel and Canny algorithms. We decided on a sobel filter because this filter was able to generate the edges for us as well as being computationally simple enough to keep the latency of the system low. The Canny algorithm is more complex than the Sobel algorithm and identifies edges within the object, not just around the object. These extra edges within the object is not required.

2D Sobel Filter

For the edge detection, we decided to use the pre existing algorithm for Sobel filter as it was pretty straightforward to implement and delivered results that meet our expectations and requirements. The Sobel operator performs a 2-D spatial gradient measurement on an image and so emphasizes regions of high spatial frequency that correspond to edges. The Sobel operations are a number of mathematical calculations and as such the 2D matrix of the incoming image is converted into a double. Once the input is a double, it is split into two perfectly identical matrices and dealt with as the x (horizontal) and y (vertical) edge detection before it is put back together to form the full function. Each is convolved with a set matrix kernel which vary from each other by a 90-degree rotation to give the x and y parts. These can be seen in figure 6-17 below represented as Gx and Gy:

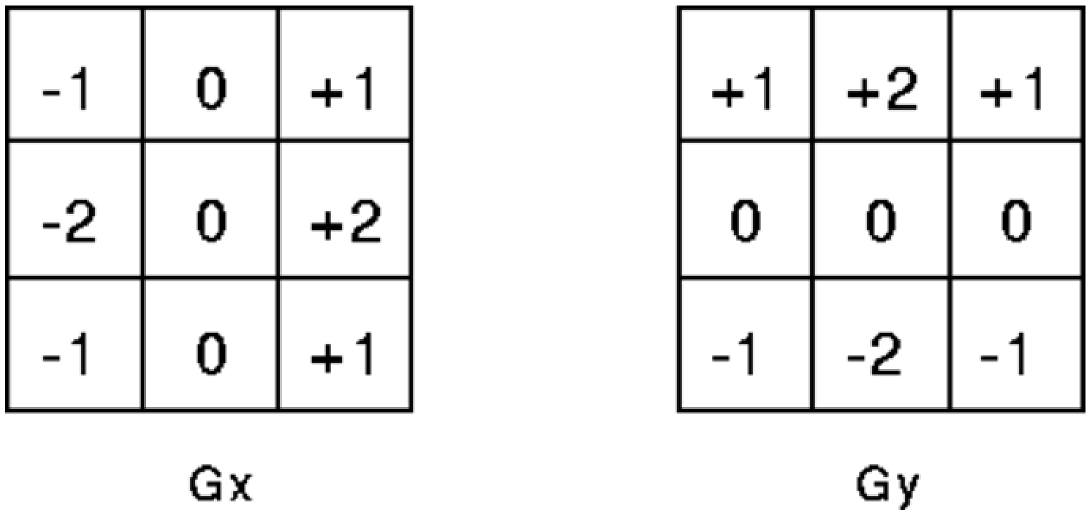


Figure 6-17: Picture of the Convolution matrix kernels used for edge detection [22]

Once convolution takes place, these signals are then each squared, and the square root of the sum is taken. Once the gradients are computed for the input 2D image, a comparison is made against a threshold value. For this experiment, we use a threshold value of 200. The maximum threshold value, which depends on the convolution kernel used, is (1 + 2 + 1)*255 - (1 + 2 +1)*0 = 1020. The threshold comparison is used to convert any of the gradients that are more than or equal to the threshold value of 200 to the white value on the color scale of 0 to 255, which is 255. All other gradient values that are less than the threshold value are converted to the black value 0. In doing this, the image is converted back to the color scale (i.e. image format) and now represents the edges of the input image. Figures 6-17, 6-18 and 6-19 respectively show the Matlab Simulink block for the implementation, the image used for the edge detection and the edge detected image.

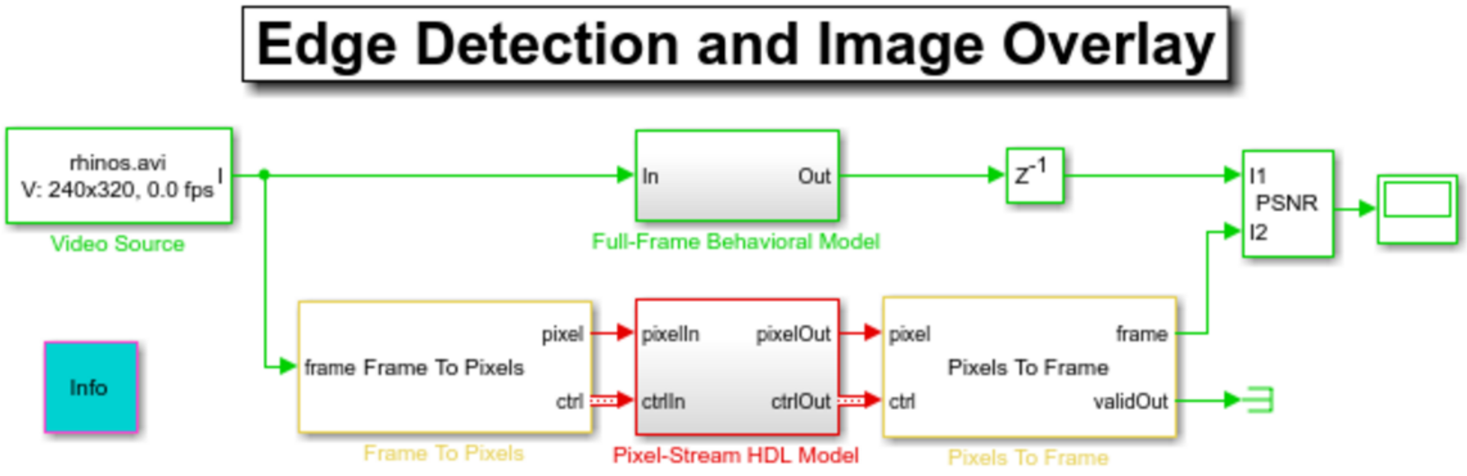


Figure 6-18: Edge Detection Block Diagram [31]

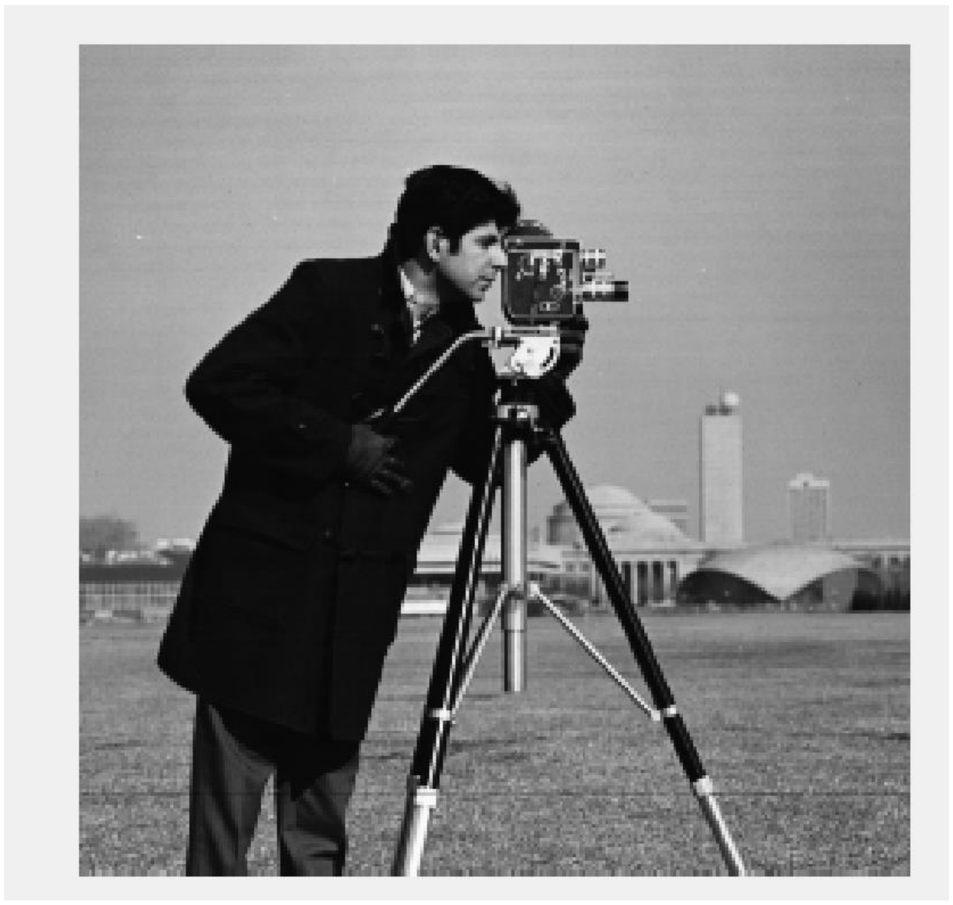


Figure 6-19 : Original picture used for the sobel filter[32]

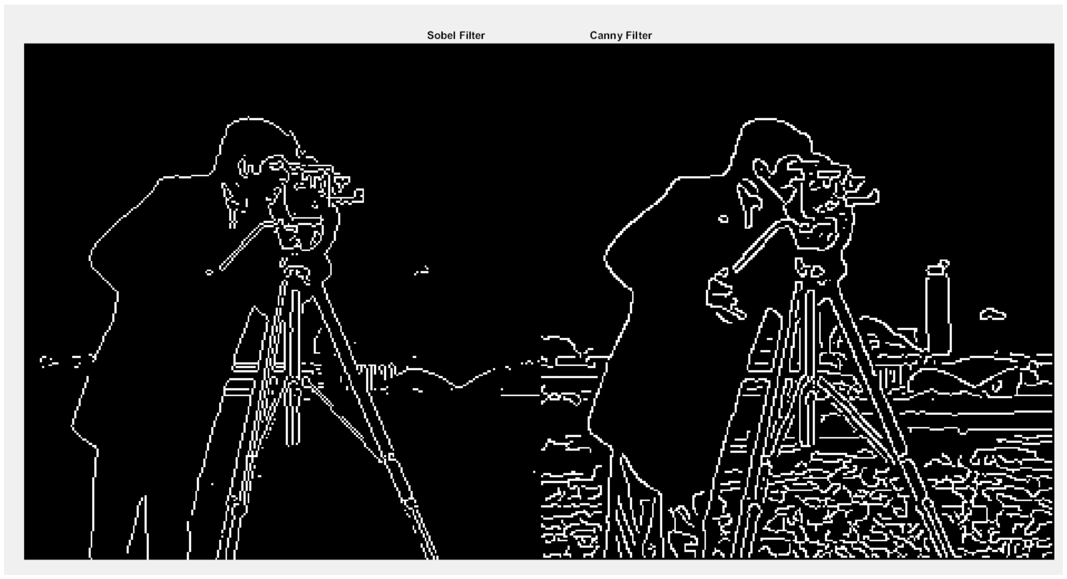


Figure 6-20: Output image of sobel filter at two different thresholds[32]

3D Sobel Filter

We are implementing a 3D filter that would be able to take the RBG input from the camera and detect the edges of the objects in the feed. With the 3D filter, we would not need the color space conversion to make the feed a grayscale 2D input. The 3D filter would work just as the 2D filter worked, just that the video stream would be split into 3 identical streams and the convolution in following figures 21, 22 and 23 would each be assigned and applied to one of the three streams. The resulting streams would then each be squared and a square root applied to the sum the squares for the gradient calculation of the image frame in the video feed. These gradients are then compared to threshold values. The threshold comparison is used to convert any of the gradients that are more than or equal to the threshold value of 200 to the white value on the color scale of 0 to 255, which is 255. All other gradient values that are less than the threshold value are converted to the black value 0. These assignments of black and white, eliminated all other unnecessary information on the picture and presents the edges of the input video feed.

$$\underset{x-1}{\begin{bmatrix} -1 & -3 & -1 \\ -3 & -6 & -3 \\ -1 & -3 & -1 \end{bmatrix}} \quad \underset{x}{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}} \quad \underset{x+1}{\begin{bmatrix} 1 & 3 & 1 \\ 3 & 6 & 3 \\ 1 & 3 & 1 \end{bmatrix}}$$

Figure 6-21: 3D Sobel filter convolution matrix kernel for x plane edge detection [23]

$$\underset{y-1}{\begin{bmatrix} 1 & 3 & 1 \\ 0 & 0 & 0 \\ -1 & -3 & -1 \end{bmatrix}} \quad \underset{y}{\begin{bmatrix} 3 & 6 & 3 \\ 0 & 0 & 0 \\ -3 & -6 & -3 \end{bmatrix}} \quad \underset{y+1}{\begin{bmatrix} 1 & 3 & 1 \\ 0 & 0 & 0 \\ -1 & -3 & -1 \end{bmatrix}}$$

Figure 6-22: 3D Sobel filter convolution matrix kernel for y plane edge detection [23]

$$\underset{z-1}{\begin{bmatrix} -1 & 0 & 1 \\ -3 & 0 & 3 \\ -1 & 0 & 1 \end{bmatrix}} \quad \underset{z}{\begin{bmatrix} -3 & 0 & 3 \\ -6 & 0 & 6 \\ -3 & 0 & 3 \end{bmatrix}} \quad \underset{z+1}{\begin{bmatrix} -1 & 0 & 1 \\ -3 & 0 & 3 \\ -1 & 0 & 1 \end{bmatrix}}$$

Figure 6-23: 3D Sobel filter convolution matrix kernel for z plane edge detection [23]

Foreground Detection

Foreground detection is one of the main block that comprise background subtraction. Background subtraction is widely used for classifying image pixels into either foreground or background in presence of stationary cameras. A Gaussian Mixture Model (GMM) model is one such popular method used for background subtraction due to a good compromise between robustness to various practical environments and real-time constraints. By using the Gaussian Mixture Model background model, frame pixels are removed from the required video to obtain the desired results. The foreground image processing has the following block stream.

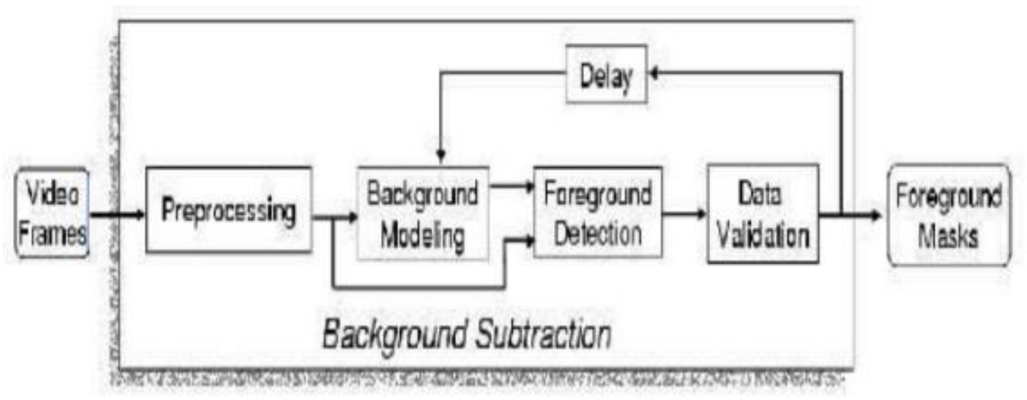
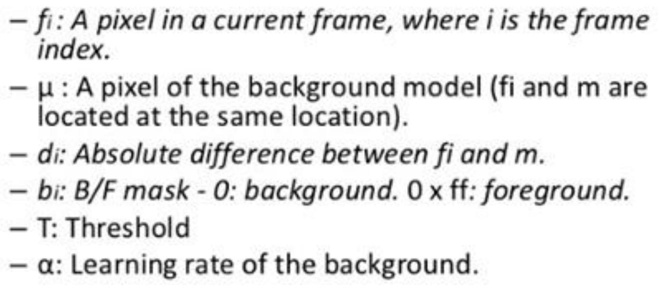


Figure 6-24: Background Subtraction Block Diagram [24]

The pre-processing block deals with cleaning up the image that contains device noise or unnecessary environmental elements such as rain or snow. The background modeling block create compute the model that will be used to compare it to future frames of the video and output If that frame is a background or a foreground. The foreground detection is the step where the background model is compared with the video frame and identifies candidate foreground pixels. Finally the data validations block removes pixels that are not relevant to the image.

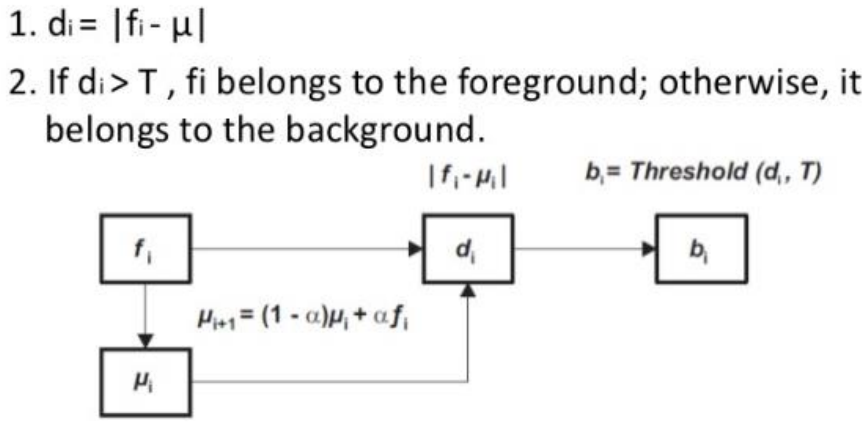
A more in depth look at the foreground detection, shows the computations that are done to decide a given frame is considered background or foreground. The algorithm compares each input pixels to the mean 'µ' of the associated components. If the value of a pixel is close enough to a chosen component's mean, then that component is counted as the matched component. To be a matched component, the difference between the pixels and mean must be less than compared to the component's standard deviation. 2. Secondly, update the Gaussian weight, mean and standard deviation (variance) to reflect the new obtained pixel value. The image below shows the different parameters that the algorithm uses.



- *f_i : A pixel in a current frame, where i is the frame index.*
- μ : A pixel of the background model (fi and m are located at the same location).
- *d_i: Absolute difference between fi and m.*
- *b_i: B/F mask - 0: background.* 0 x ff*: foreground.*
- T: Threshold
- α: Learning rate of the background.

Figure 6-25: Foreground Algorithm Parameters [25]

The image shown below illustrates the block diagram of the foreground algorithm.



1. $d_i = |f_i - \mu|$
2. If $d_i > T$, fi belongs to the foreground; otherwise, it belongs to the background.

$|f_i - \mu_i|$
b_i= Threshold (d_i, T)
f_i
d_i
b_i
$\mu_{i+1} = (1 - \alpha)\mu_i + \alpha f_i$
μ_i

Figure 6-26: Algorithm Block diagram [26]

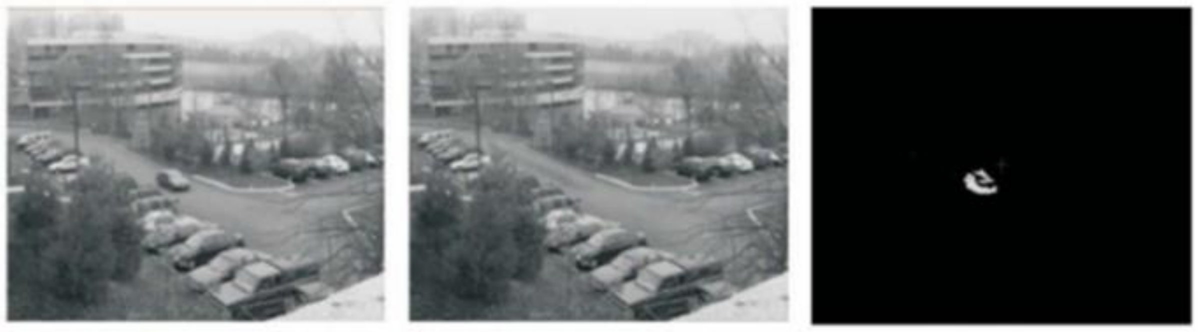


Figure 6-27: Original Image, Background Model, output Image. [27]

Implementation of Algorithms on DE1-SoC

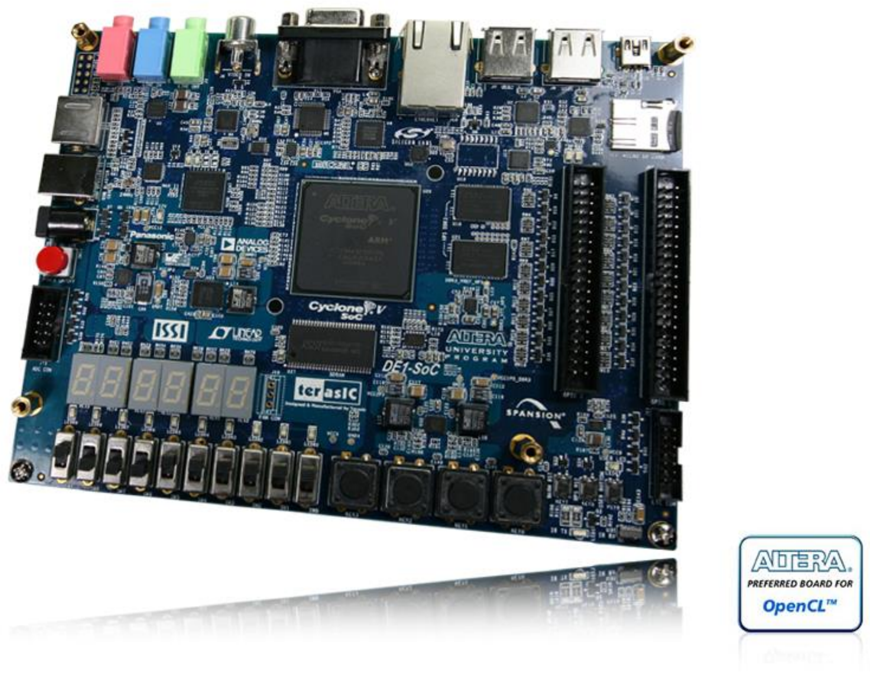


Figure 6-31: DE1-SoC Development Kit by Terasic

Both the Image Pre-Processing and the Machine Learning subsystems operate off the DE1-SoC development board. This board features an integrated Cyclone V SoC device with a Dual-Core ARM cortex-9 Hard Processing System, 85K programmable logic elements in standard FPGA arrangement, 4450 Kbits of embedded memory native to the SoC, as well as 64 MB of SDRAM, 1 GB of DDR3 SDRAM for the HPS and 2 hard memory controllers. The board also contains multiple I/O peripherals including push-button user keys, switches, LEDs, 2 40-pin expansion GPIO headers, 2 USB 2.0 ports with ULPI interface, and a set of ADC and DAC arrays for signal processing applications.

The schematic below illustrates the overall configuration for all peripheral hardware devices and how they communicate between both the ARM HPS and the FPGA architecture.

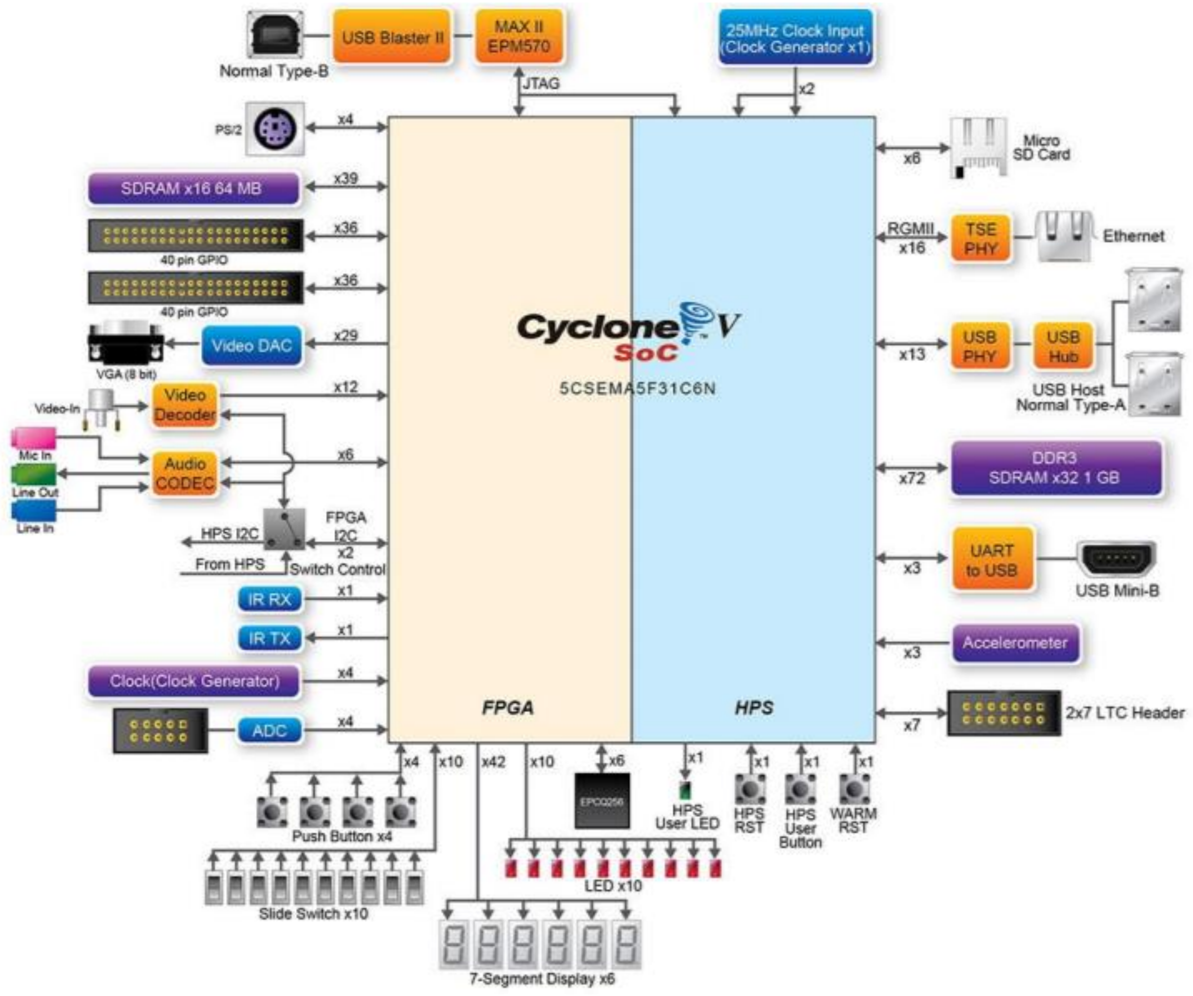


Figure 6-32 Board Hardware Layout

As is implied through the above schematic, only certain peripheral devices are directly connected to the HPS section of the Cyclone V. As a result of this, all peripheral devices which connect directly to the FPGA architecture cannot innately communicate with the HPS. To combat this issue while still maintaining the advantages gained from direct connection to the FPGA, the Cyclone V implements a lightweight bridge between the HPS and FPGA architectures on which data can be transferred, and allowing the HPS to act as master to peripherals on the FPGA side.

The main peripherals accessed through this project are the Clock Generator (for implementing timing structures necessary for control logic), the ADC pin port (for reading data off of the potentiometer sensors), the 40 pin GPIO expansion headers (for reading from the limit switches and driving the DC and Servo Motors), and the SDRAM (for local memory storage during image conversion and filtering). In order to save design time and considering that no special considerations needed to be made regarding these peripherals with the sole exception of the PWM generation needed for the servo motor pins, it was decided to implement most of the FPGA interfaces for these pins using the reference Verilog modules provided in the DE1-SoC computer design reference from Altera.

The DE1-SoC computer design reference is a preset library of simple interface FPGA designs that make using peripherals through the FPGA faster and more efficient. Each of the implemented controllers and data registers are memory mapped to addresses across the lightweight FPGA-to-HPS bridge starting from a base address of 0xC0000000 with a max word size of 32 bits. Several of the peripherals are implemented as parallel ports, supporting input, output, and bidirectional data transfer through proper direction register configuration. Most of the peripherals also have data mask settings pre-implemented to allow for interrupt generation from the FPGA. These interrupts are handled by the generic interrupt controller on the HPS and are generated whenever a register corresponding to a peripheral device changes value. This primarily comes into play in the implementation of the timer controller and clock generator but also plays a small role in the motor control Interrupt Service routine in regards to the limit switches.

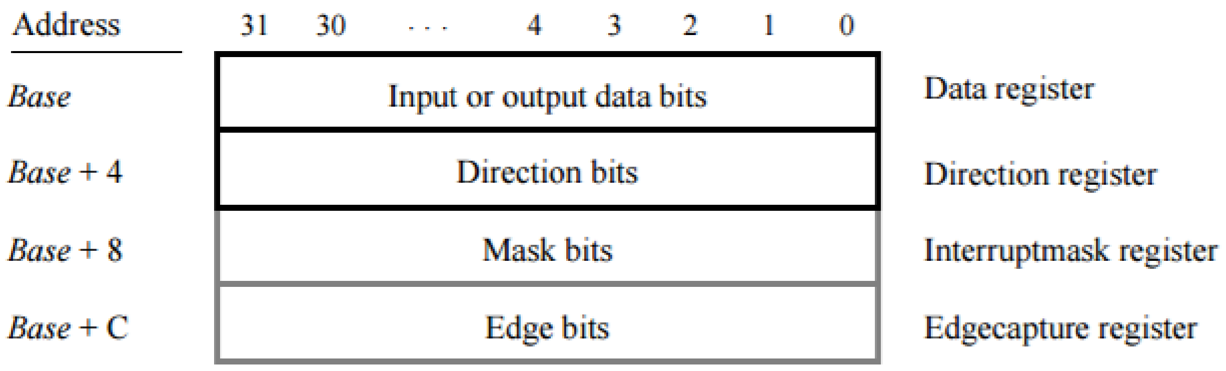


Figure 6-33 SDRAM Memory Structure

Proper configuration of the parallel GPIO expansion header ports is responsible for most of our peripheral interfacing and therefore of particular interest. Each pin can hold a state of either 1 or 0, with 1 corresponding to a 3.3 volt lead line at approximately 50 ma and 0 corresponding to ground. General procedure for outputting values includes setting all direction registry values necessary to 0 and assigning the corresponding pin a 1 or 0 value, while input lines simply require directional settings of 1. This is generally handled through the Init_ procedures of each appropriate ISR that interacts with the GPIO lines (Data Registers D0 through D31).

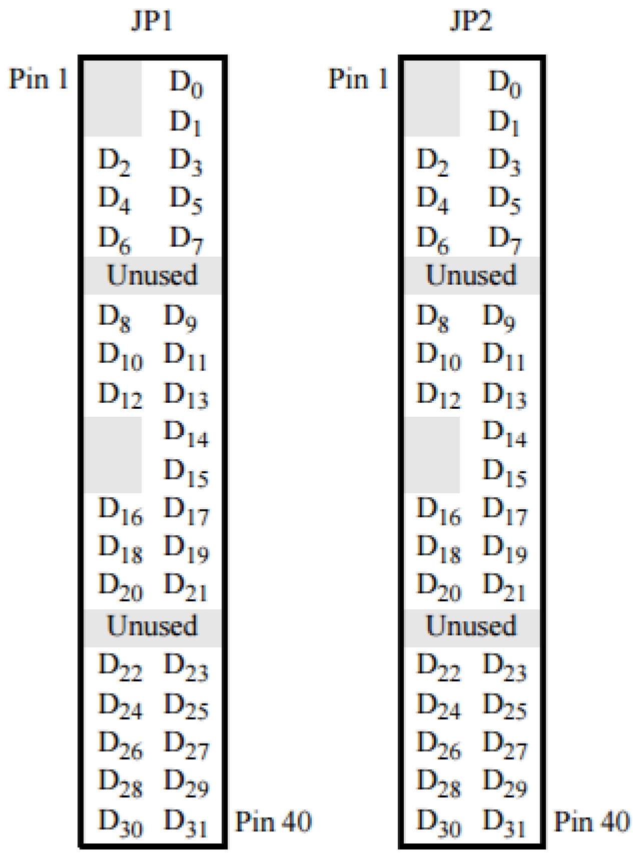

JP1
JP2
Pin 1
Pin 40
D0 D1 D2 D3 D4 D5 D6 D7
Unused
D8 D9 D10 D11 D12 D13 D14 D15 D16 D17 D18 D19 D20 D21
Unused
D22 D23 D24 D25 D26 D27 D28 D29 D30 D31

Figure 6-34 GPIO Pin Structure

The Linux Kernel

Figure 6-35 ARM-Linux Logo [33]

In an effort to simplify design requirements, it was decided that a Linux operating system be implemented on our ARM HPS. Doing so allowed the design of the image processing subsystem to be contained to a single binary executable, easily understandable and operable through ARM cross compilation; removed the need for custom written peripheral drivers for our UVC based webcam, and keep our control flow to a base level interrupt control flow which comes native in Linux based kernel systems.

Our Linux distribution is modified version of the general purpose Linux kernel source (v3.18.0) compiled for ARM based architectures. Kernel compilation and correct building on the DE1-SoC board requires 4 things:

- A Pre-loader script which prepares the Cyclone V to handle the instructions given by the main bootloader
- A Boot-loader which takes and creates a bootable image to be loaded onto an SDcard and used to extract and instantiate Linux on the DE1-SoC
- A Linux Kernel Source
- A Linux file system core for access to and command level interaction and control with the Linux kernel

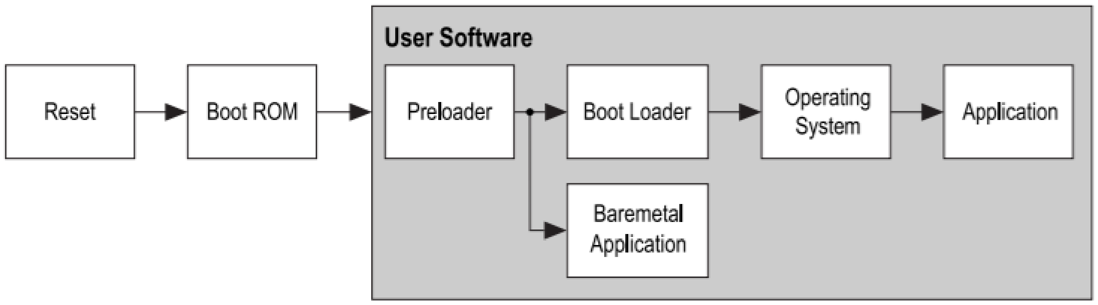
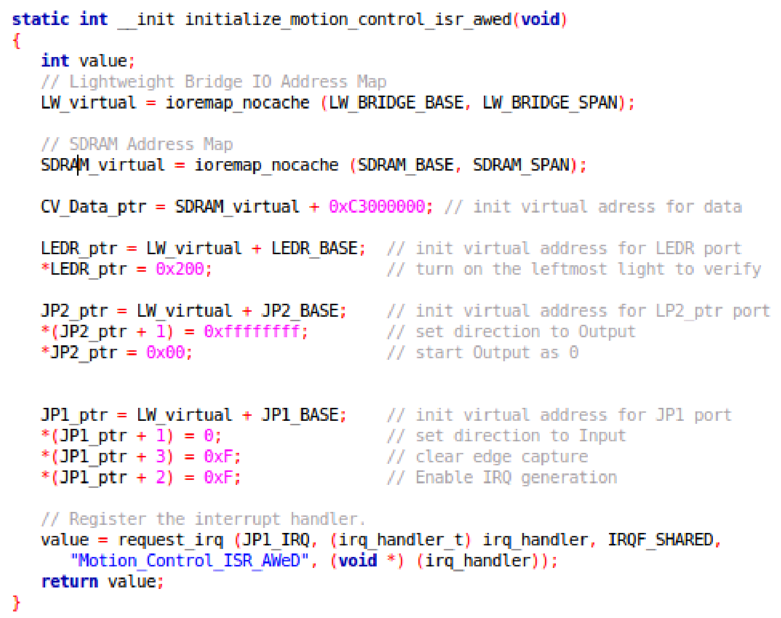


Figure 6-36 HPS Layout

In addition to all this, there must also be a device tree blob or .dtb file present on boot which tells the Linux operating system how to interface with all of the peripherals and hardware resources native to both the HPS and the board itself. Unfortunately, no predefined device trees are available for the DE1-SoC, meaning this .dtb file must either be ripped from an already existent Linux implementation (such as those provided on the Altera university site), converted back to a device tree source file (.dts) which can yield some instabilities or substituted with a more commonly available device tree from another development board such as the Cyclone V professional development board at the expense of some native peripheral interfacing on the HPS.

Interrupt Service Routines



```c
static int __init initialize_motion_control_isr_awed(void)
{
   int value;
   // Lightweight Bridge IO Address Map
   LW_virtual = ioremap_nocache (LW_BRIDGE_BASE, LW_BRIDGE_SPAN);

   // SDRAM Address Map
   SDRAM_virtual = ioremap_nocache (SDRAM_BASE, SDRAM_SPAN);

   CV_Data_ptr = SDRAM_virtual + 0xC3000000; // init virtual adress for data

   LEDR_ptr = LW_virtual + LEDR_BASE;  // init virtual address for LEDR port
   *LEDR_ptr = 0x200;                  // turn on the leftmost light to verify

   JP2_ptr = LW_virtual + JP2_BASE;    // init virtual address for LP2_ptr port
   *(JP2_ptr + 1) = 0xffffffff;        // set direction to Output
   *JP2_ptr = 0x00;                    // start Output as 0


   JP1_ptr = LW_virtual + JP1_BASE;    // init virtual address for JP1 port
   *(JP1_ptr + 1) = 0;                 // set direction to Input
   *(JP1_ptr + 3) = 0xF;               // clear edge capture
   *(JP1_ptr + 2) = 0xF;               // Enable IRQ generation

   // Register the interrupt handler.
   value = request_irq (JP1_IRQ, (irq_handler_t) irq_handler, IRQF_SHARED,
      "Motion_Control_ISR_AWeD", (void *) (irq_handler));
   return value;
}
```

Figure 6-37 ISR Motion Control Initialization Example

The overall control flow of our system is handled by 3 interrupt service routines which are called by the generic interrupt handler of the HPS. The 3 main ISRs are as follows:

1. Camera Capture ISR - responsible for the interfacing between the HPS and the Vari-focal lens Webcamera peripheral as well as saving captured image feed to the image filter buffer
2. Image Detection ISR - responsible for calling the image detection binary executable generated by our neural net and passing the filtered image stream data into it
3. Response System ISR - responsible for initializing the positional orientation of the gimbal as well as its scanning and tracking features and sending an alarm signal if the Detection ISR has output a threat.

Each of the ISRs are added as kernel modules to the main interrupt service handler native to the Linux OS during the board's initial boot up process. This is done through modifications to the uboot script to allow for additional modules to be added before the main Linux operation is launched.

Each ISR is driven using an interrupt timer peripheral on the HPS architecture, each of which is set during each initialization of the ISR at run time. For the most part, these timers function independently, and are solely responsible for the timing of each routine.

The Camera Capture ISR is initialized by associating the ISR with the first interrupt timer signal (HPS Timer 0). The timer is constructed in such a way that the ISR will trigger every .1 seconds to take a new snap shot of the camera's input. Frames are captured using functions provided through the Fswebcamera software as well as the video for Linux utility libraries. Once captured, a frame is saved to an allocated location on the FPGA's SDRAM. The SDRAM is initialized to be able to hold 5 separate images at once, to allow for foreground detection and verification of varying degrees and scales. Every time a new image is captured, it pushes all of the other images down one and drops the 6th oldest image. This buffer is then connected to the pre-processing filter systems input lines, allowing for RT level processing and filtering.

The Image Detection ISR runs at the same .1 second resolution as the Camera Capture ISR. Every .1 seconds, the detection ISR checks the output bus registers of the pre-processing filter system (also stored in SDRAM), makes a local copy of that image, and passes the copy into the image detection binary callback. This callback function executes the image detection binary executable in a separate process stream and then ends the ISR. To accommodate for the length of time it takes to run the detection algorithm, multiple instances are allowed to run at once, taking advantage of the dual core nature of the ARM. If a threat is detected during the running of the binary, the process stream is set to append to the count value of the third ISR timer (essentially forcing an interrupt) while setting the appropriate Data Values in SDRAM to begin the orientation and zoom procedures.

The last ISR, the Motion and Response ISR, is initialized at a .05 second resolution which is 50% faster than the other two so as to prevent the limit switches and potentiometer value changes from being ignored. This ISR is initialized to read in values from the SDRAM and the 2 GPIO header expansion ports. For ease of implementation sake, the right most GPIO (JP1) has been denoted as an input line while the left most GPIO (JP1) is set to output. The Limit Switches and potentiometers are connected to JP2 while the LEDs, motors and servos are connected to the pins of JP1. Decision making in the ISR is handled through case based logic. The ISR first checks the SDRAM to see if initialization values for the potentiometers have been set. If no such values are detected (as would be denoted by a value of "1"), the gimbal then begins its initialization routine, going to its vertical and horizontal extremes until it hits a limit switch, at which point the routine saves and stores the potentiometers initial values using the HPS's ADC. Once both values are set, the ISR checks to see if a threat has been registered on the SDRAM's data line. If one has not, it will simply begin panning left and right in a loop. If one has, it checks the SDRAM data to see which window the threat was reported in, and adjusts based on that threat's location (two flags in the bottom two windows moves the camera down, one in the top left moves the camera left and up, center only causes the camera to zoom in while the full size window only causes the camera to zoom out).

Machine Learning - Building a Binary Classifier

Implementing the CNN

The computer vision module uses a CNN to scan the incoming camera frames for firearms. The implementation of the CNN is based on Google's open source software library for machine learning, called Tensorflow. Tensorflow allows the creation of graphs, where each graph represents a mathematical model, to facilitate numerical computations necessary for machine learning algorithms. The graphs consist of nodes, each one representing a mathematical operation. Multi-dimensional arrays, called tensors, are used as graph edges to communicate the data between the different nodes. The CNN is defined, therefore, as a collection of nodes, where a tensor is given as input and another tensor returned as output from the last node of the graph. The input tensor is the incoming camera frame and the output tensor will be the classification label.

The TFLearn library is used to facilitate development, as it provides a higher-level API to Tensorflow. The network architecture loosely follows the architecture proposed by the TFLearn's image classification example, which uses the CIFAR-10 dataset. As a result, the input tensor was a 32x32 (width x height) RGB image, or 3 channels, giving a total of 32x32x3 = 3072 nodes. Initially, the dimensions were kept as 32x32 and were later adjusted to 80x45.
In order to extract the features that we are trying to detect from the input tensor, we will use a combination of convolution and max-pooling steps:

(1) - 32 filter convolution => (2) – 2 stride max-pooling => (3) - 64 filter convolution =>

(4) – 64 filter convolution => (5) – 2 stride max-pooling

Every convolution layer applies a number of filters (either 32 or 64), each of which is a square window with a 3x3 pixels size. For instance, the first convolution layer takes as input the incoming tensor and summarizes it with 32 filters of 3x3 smaller neural networks. Every max-pooling step is a reduction operation, where each rectangular window of size 2x2 pixels is represented by its maximum value. As a result, each max-pooling layer reduces the dimensions to a quarter. The complete network architecture is shown in the figure below:

By applying this routine, the incoming nodes can be summarized using fewer resources. Each convolution splits the images into tiles and attempts to summarize them with a small neural networks. Each max-pooling step performs an efficient down-sampling by keeping the most interesting bits (with respect to graph weights).

After the network architecture has been defined, with each training step a regression layer is applied and the weights of the model refined. With every training step, the weights are either increased or decreased based on the network's performance. The amount of change is defined as the learning rate of the network and set at 0.001. With a larger learning rate, the network is trained more quickly but may result in overfitting. In to avoid over-fitting, a percentage of the data is dropped during each training step. Each node of the network has an equal probability of being dropped, which is defined as the dropout rate of the network. The dropout rate is initially defined as 0.50 and later adjusted to 0.15.

The output tensor from the last max-pooling layer is summarized by a fully-connected layer of 512 nodes. These 512 nodes hold the summarized features from the incoming camera frame. The softmax, or normalized exponential, activation function is used to generate a binary classification output from the 512 nodes. The complete network architecture is shown in the figure below:

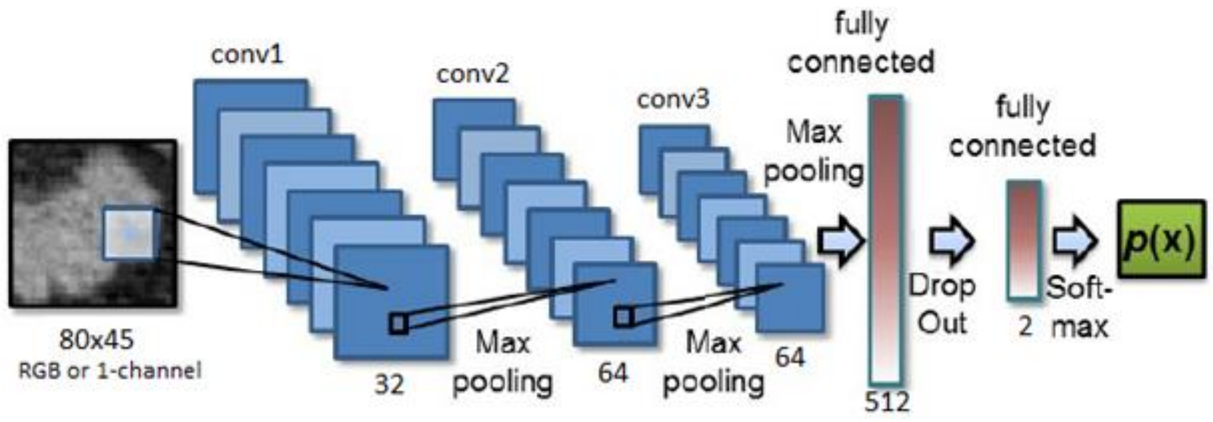


Figure 7-1: Neural Network Architecture [35]

Creating the Training Datasets

The objective of our ML algorithm is to function as a binary classifier that will take an image as an input and return whether there is a threat (C=1), i.e. a visible firearm, or whether it is safe (C=0). To train a CNN to respond accordingly we will need a dataset of positive (C=1) and negative (C=0) examples. During the development stages of the algorithm, a Nerf Gun was used as the positive class because its distinct colors and features make it an easier target to test with.

During early stages of development, the dataset was created by taking individual images of the Nerf gun with varying backgrounds. To populate our dataset we initially included 2500 negative 100x100 images from the AGH public knives dataset [8]. OpenCV, an open source computer vision library, was later used to extract individual frames from a video. By using videos, a larger number of training instances can be handily generated but at the cost of quality and focus. A Python script, using OpenCV, was used to populate the negative training dataset by splitting each image into its four quarters. Because a negative instance should contain no visible firearms, for every negative instance, four additional ones were added by the populating script.

Initially, all images for training are separated into two class folder, 'classes/0/' and 'classes/1/', where the 0-class is the negative, i.e. no nerf-gun class, and the 1-class is the positive, i.e. nerf-gun detected, class. Two Python scripts are used to preprocess the images: renameFile() and genResized(). First, renameFile() renames all images in the databases to "fileX.jpg", where X is the image number. The produced files now have a consistent naming and type. The database is then formatted with the genResized() script, which resizes all images to a specific size, in our case 80x45 pixels.

The formatted images are converted to RGB matrices, in a 80x45x3 = width*height*colors format. The matrices are split to a training dataset (80% of total) and a validation dataset (20% of total). The RGB values of each pixel were normalized from their original 0-255 range to 0-1. Finally, for each of the training and validation datasets, an equal in length array of Bytes is appended that holds the class label (00H – safe, 01H –threat) of each image as part of the supervised learning.

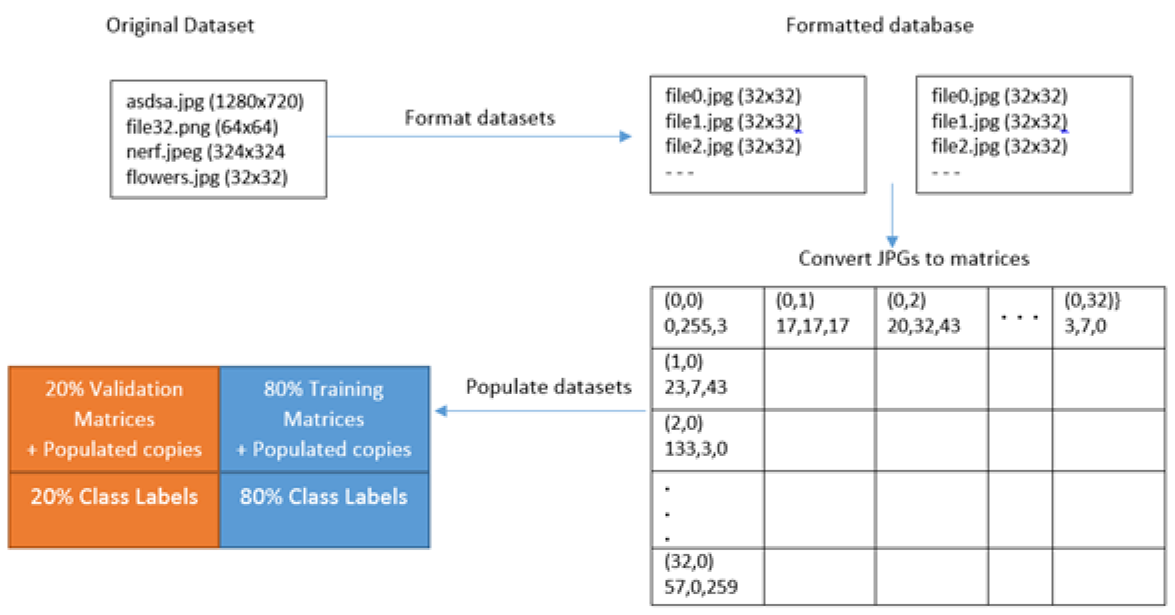
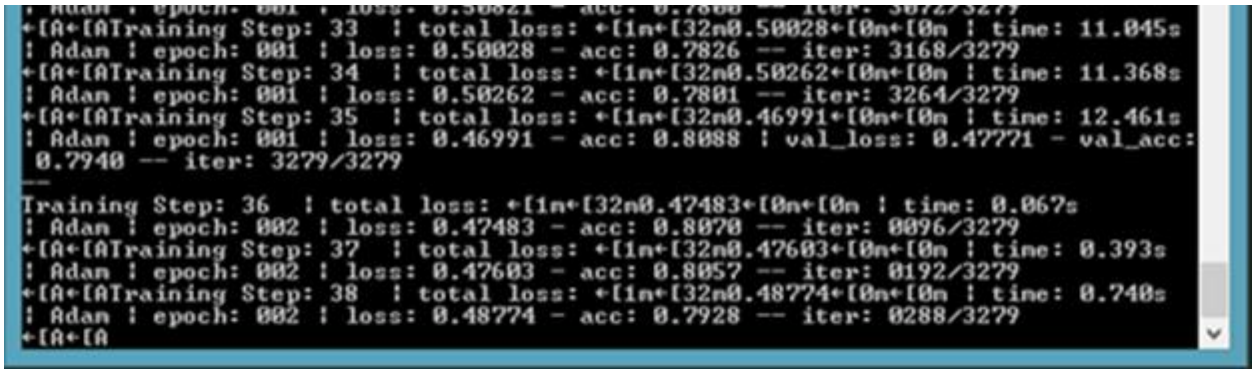


Figure 7-2: Dataset Pre-Processing Pipeline

Training the Model

The generated training and validation datasets are further populated prior to training with the help of Tensorflow functions. Copies of the original images are added to the dataset, after being either blurred, flipped horizontally or rotated. All three processes are applied at random and with random parameters, i.e. the blur coefficient or angle of rotation. The training routine consists of a combination of convolution and max-pooling steps, as shown before. The CNN is trained to recognize the patterns generated by the negative and positive samples of our training sets, representing the safe and threat scenarios for a given number of training steps, called epochs. Most models were trained for 100-300 epochs, depending on the size of the training dataset and the difficulty of the task. Low environmental-noise datasets generally required less epochs and training for more resulted in overfitting.

During training, the console output shows some statistics, such as the training step, the accuracy and the process time. These statistics help while refining the parameters of the neural network. While the model is being trained, a snapshot of the model's metadata is saved every 4 training steps. Each training step with the current configuration takes approximately 2 minutes to process. At the end of the training, the final model is saved, together with a checkpoint file that is necessary to load the model.



```
| Adam | epoch: 001 | loss: 0.50821 - acc: 0.7808 -- iter: 3072/3279
←[A←[ATraining Step: 33  | total loss: ←[1m←[32m0.50028←[0m←[0m | time: 11.045s
| Adam | epoch: 001 | loss: 0.50028 - acc: 0.7826 -- iter: 3168/3279
←[A←[ATraining Step: 34  | total loss: ←[1m←[32m0.50262←[0m←[0m | time: 11.368s
| Adam | epoch: 001 | loss: 0.50262 - acc: 0.7801 -- iter: 3264/3279
←[A←[ATraining Step: 35  | total loss: ←[1m←[32m0.46991←[0m←[0m | time: 12.461s
| Adam | epoch: 001 | loss: 0.46991 - acc: 0.8088 | val_loss: 0.47771 - val_acc:
 0.7940 -- iter: 3279/3279
--
Training Step: 36  | total loss: ←[1m←[32m0.47483←[0m←[0m | time: 0.067s
| Adam | epoch: 002 | loss: 0.47483 - acc: 0.8070 -- iter: 0096/3279
←[A←[ATraining Step: 37  | total loss: ←[1m←[32m0.47603←[0m←[0m | time: 0.393s
| Adam | epoch: 002 | loss: 0.47603 - acc: 0.8057 -- iter: 0192/3279
←[A←[ATraining Step: 38  | total loss: ←[1m←[32m0.48774←[0m←[0m | time: 0.740s
| Adam | epoch: 002 | loss: 0.48774 - acc: 0.7928 -- iter: 0288/3279
←[A←[A
```

Figure 7-3: Console output during training

The CNN is implemented and trained in Python using the TFLearn wrapper library for Tensorflow. The trained model is exported using Tensorflow's freezeGraph() function that returns a saved snapshot of the weights of the model. In order for the model to be ported to our FPGA, we needed to build it targetting an ARM-Linux processor. For this purpose, a C++ version of our project was implemented with the Tensorflow C++ API. A virtual machine was set up using VirtualBox and running Ubuntu 16.10 in order to build the C++ project, with the help of the Bazel build tool.

Threat Detection Routine

The same network architecture used for training is also defined for the testing routine. The 512-node CNN takes a 80x45x3 RGB matrix as input and returns a [x][y] tuple, where x+y = 1, x = prob(no-threat), and y = prob(threat). Essentially, the network returns two probabilities for two mutually exclusive events as floats, with a value between 0 and 1. An example result would be in the form [0.4, 0.6], meaning that the network predicts that there is a 40% (C=0) chance that the image is safe and 60% (C=1) that a firearm is present.

The test images have a 1280x720 size. A sliding routine is implemented in Python that returns 5 sub-images from the original images to be scanned using OpenCV's rectangle() function. A sliding window of size 640x360 produces 5 images by scanning a 2x2 matrix of the image plus an additional 640x360 sub-image at the center of the frame. The 5 images of size 640x360 and the original one are all resized to 80x45 pixel images and then converted to a 80x45x3 matrix (like Figure 1). The complete detection routine with the sliding window is visually demonstrated the figure below:

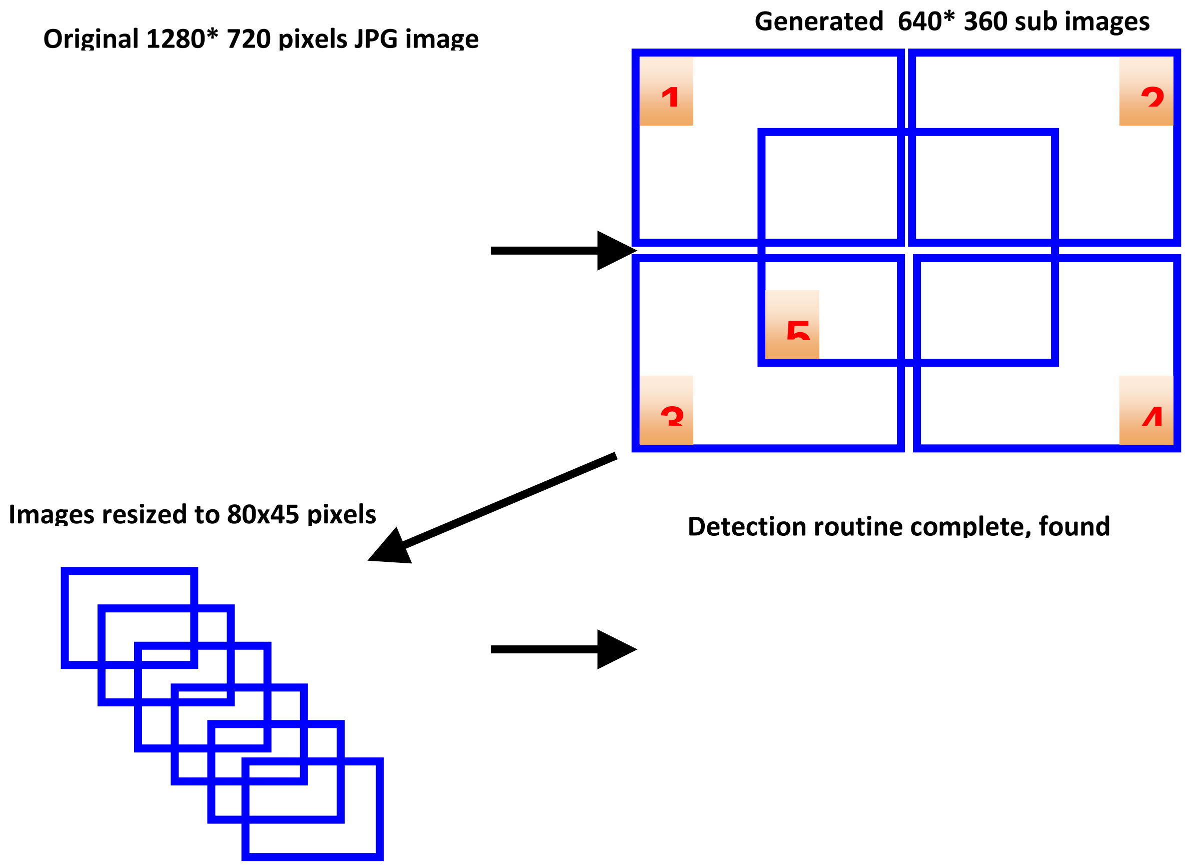

Original 1280* 720 pixels JPG image
Generated 640* 360 sub images
1
2
5
3
4
Images resized to 80x45 pixels
Detection routine complete, found

Figure 7-4: Detection Routine Example

By using the sliding window routine, each incoming frame generates 6 probabilities from the CNN. As a result, the sliding window is a trade-off between processing time and accuracy. If the resources permitted, the sliding window could be performed with an even smaller size to generate more sub-images to test. The threshold for triggering the alarm is currently set at 2 out of 6 having over 0.5 probability of being a threat or 1 out of 6 having a probability bigger than or equal to 0.75. By setting this type of threshold, the number of false positives is reduced. When used without the sliding window routine, the threshold for an alarm remains at a 0.5 output.

Raspberry Pi Prototype

A prototype of the threat detection routine was implemented using a Raspberry Pi 3.0. The Raspberry Pi was chosen as it is based on an ARMv7A processor and has a 1GB of available RAM like the Altera programming board. There is also an active community of Pi developers providing tutorials, troubleshooting support and open source libraries online. In addition, the GUI provided by Raspberry Pi's Raspbian OS facilitates development and testing.

In order to port the project on the Raspberry Pi, the C++ detection routine using Tensorflow needed to be compiled targeting the ARMv7A architecture. The Bazel build tool was compiled using GCC and linked within the Raspbian OS. The Bazel tool was then used to compile the Tensorflow project and to generate the executable files. The model graph was also edited for compatibility issues. The nodes in the graph that corresponded to the dropout operations were removed, as the mathematical operations were neither compatible with the 32-bit architecture, nor necessary for the detection routine. A webcam was used to capture the input camera frames, using the UCV driver for Raspbian.

The Pi detection routine used OpenCV to read in the camera frames from the webcam. The camera capture's dimensions were set to 1280x720 and the sliding window was also implemented for the C++ routine using OpenCV, with the same parameters as discussed before. Whenever a threat was detected, a test alarm was played via HDMI or Pi's audio output.

Gimbal Control

During operation, the gimbal pans left and right to get a view of the entire area. When a threat is detected, the gimbal will begin to track it and zoom in order to maintain a good view of the threat. In order to determine the location of the threat and the level of zoom, the results of the sliding window are analyzed. Depending on which windows a threat is detected in, the system can know the direction of the threat relative to the current view and adjust accordingly. In order to determine level of zoom, the number of windows that return a threat are analyzed. The more windows that detect a threat, the larger the threat is on screen. Psuedocode for these behaviors can be seen below.

```
// Global vars
Int rightBound
Int leftBound

// Initialization
// Vertical Axis
While (limit switch is not pressed) {
        Drive axis down
}
Stop axis
for(15 degrees) {
        Drive axis up
}
Stop axis

// Horizontal Axis
While (limit switch is not pressed) {
        Drive axis left
}
Stop axis
leftBound = horizontalPot.value
rightBound = leftBound - value (dont know value yet, need to test for value)

// Panning
While (!threat detected) {
        If (drivingLeft) {
                Drive axis left
} else if (drivingRight) {
        Drive axis right
}
        If (horizontalPot.value >= leftBound) {
                drivingLeft = false
                drivingRight = true
        }
        If (horizontalPot.value <= rightBound ) {
                drivingLeft = true
                drivingRight = flase
        }
}

// Threat tracking
While (threat detected) {
        If (threatLocation = leftHalf) {
                If (horizontalPot.value <= leftBound) {
                        Drive axis left
                } else {
                        Stop axis
                }
        } else if (threatLocation = rightHalf) {
                If (horizontalPot.value >= rightBound) {
                        Drive axis left
                } else {
                        Stop axis
                }
        }
If (threatLocation = upperHalf) {
                Drive axis up
        } else if (threatLocation = lowerHalf) {
                If (!limit switch pressed) {
                        Drive axis down
                } else {
                        Stop axis
                }
        }
        If (numberOfWindows > 2) {
                Zoom out
        }
        else {
                Zoom in
        }
}

// Threat no longer detected
While (!vertical limit switch pressed) {
        Drive axis down
}
Stop axis
For (15 degrees) {
        Drive axis up
}
Stop axis
Resume panning
```

Figure 7-5: Pseudocode for gimbal control

System Testing and Evaluation

Testing the CNN

During the development of the CNN, statistics were gathered in order to evaluate its performance and refine its parameters. The time and space complexity of the algorithm was evaluated based on the training time, the detection time and the size of the model graph of weights. By varying the format of our input node and training the model for 10 epochs, the below table is produced:

TABLE 7-1: CNN Development Statistics

Model Type (WxH)	Training Time - 10 epochs / seconds	Graph Size
32x32 RGB	48.07	8,424 KB
80x45 RGB	85.41	30,952 KB
64x64 RGB	89.48	33,000 KB
100x100 RGB	165.64	80,232 KB
160x90 RGB	218.72	117,992 KB

The training time does not increase exponentially and therefore wasn't a constraint. Most models needed around 200 epochs training that should be completed in around 12 hours training time for the worst case (160x90). The graph size was around 118MB in the worst case, which is relatively small compared to the available 8GB in the SD card.

Based on the validation set (20% of the training dataset), cross-validation statistics are gathered during training, the validation loss and the validation accuracy. The validation loss is a measurement of the average error per classification and the validation accuracy is the percentage of the validation instance correctly recalled. These statistics help determine whether the CNN model is capable of extracting features from the training dataset. In Table XX, the cross-validation statistics are depicted with varying dropout and learning rates for the model:

Table 7-2 : Cross-validation statistics with Varying Dropout / Learning Rate

Dropout Rate	Learning Rate	Validation Loss	Validation Accuracy %
1	0.001	5.55	75.88
	0.005	5.55	75.88
	0.1	5.55	75.88
5	0.001	0.2047	93.97
	0.005	5.55	75.88
	0.1	5.55	75.88

10	0.001	0.1873	91.46
	0.005	5.55	75.88
	0.1	5.55	75.88
15	0.001	0.1712	92.46
	0.005	5.55	75.88
	0.1	5.55	75.88
25	0.001	0.2027	90.95
	0.005	5.55	75.88
	0.1	5.55	75.88

Because 75.88% was the percentage of the negative training instances in the dataset, all table rows shaded in blue show cases where the model wasn't able to learn anything meaningful, in contrast to rows shaded in red. The parameters offering the best results were observed to be 15% dropout rate and learning rate equal to 0.001.

The processing time of the CNN and the detection routine was evaluated using the Pi Prototype. The input size was varied between 32x32 and 160x90 pixels and the run time was measured both with and without the sliding window routine. The resulting processing times for reading 10 camera frames as input, processing them and classifying them are summarized in the table below:

Table 7-3: Average run times to process 10 frames for varying CNN node size.

Model Type (WxH)	Processing time / sec	Processing time sliding window / sec
32x32 RGB	1.85	2.04
80x45 RGB	4.75	16.34
64x64 RGB	5.08	18.16
100x100 RGB	11.26	48.59
160x90 RGB	16.16	72.00

The 80x45 size was chosen as it offers either 0.6 FPS or 2.1 FPS, resulting in a couple of seconds delay in the worst case. With the addition of different modules responsible for capturing and pre-processing the frames, the FPS of the detection routine can be even higher than the one shown by the Pi prototype.

Implementing the HPS-FPGA data transfer

Due to the camera we bought not being able to be connected directly to the Field Programmable Gate Array part of the DE1-SOC board, we had to connect it through USB to the operating system on our HPS. In order to be able to implement the filters on the camera input we had to send the data from the Linux OS we booted on the Hard Processing System to a buffer line on the FPGA's 64MB SDRAM. Luckily to our advantage of our board having both the ARM processor and the FPGA components, there is a way for both components of the board to exchange information from one to another (i.e. HPS to FPGA and FPGA to HPS) using the Advanced eXtensible Interface (AXI) bridge. The AXI bridge is a pre-existing communication line for the interfacing of components on the FPGA with the HPS. There are 2 types of AXI bridges for communication between the HPS and the FPGA (Figure a):

- HPS to FPGA Bridge
- Lightweight HPS to FPGA Bridge

We are using the HPS to FPGA Bridge instead of the Lightweight HPS to FPGA Bridge because it works with a larger number of bits; While the HPS to FPGA can use 32, 64 or 128 bits, the lightweight can only handle 32 bits.

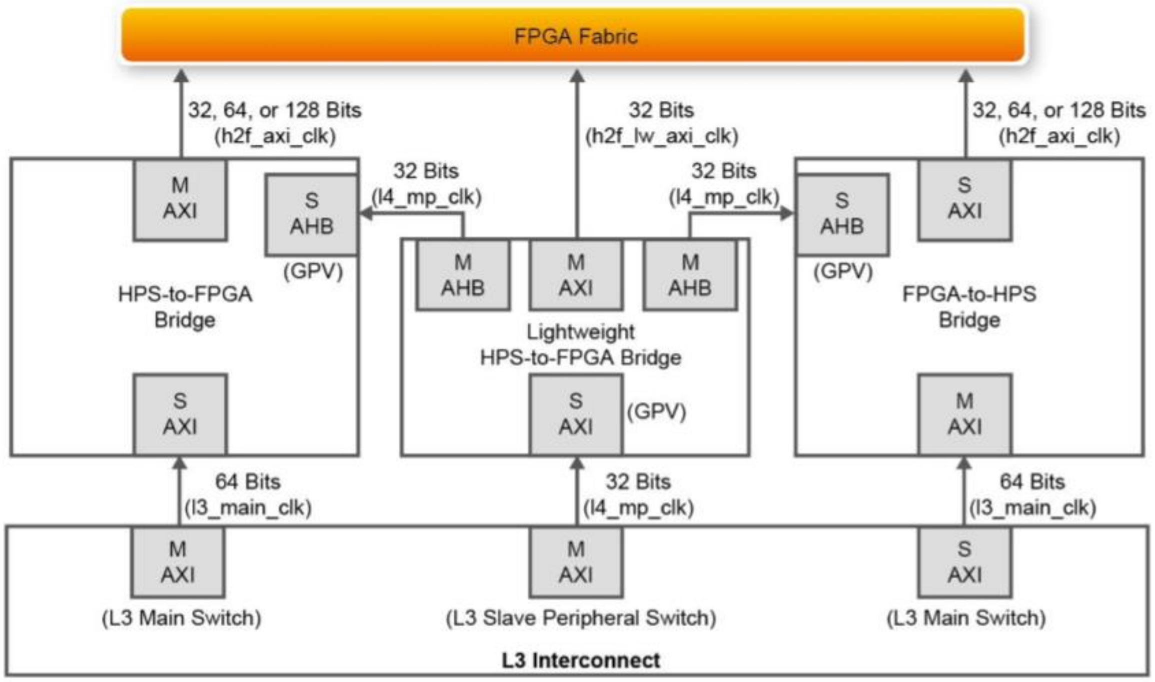
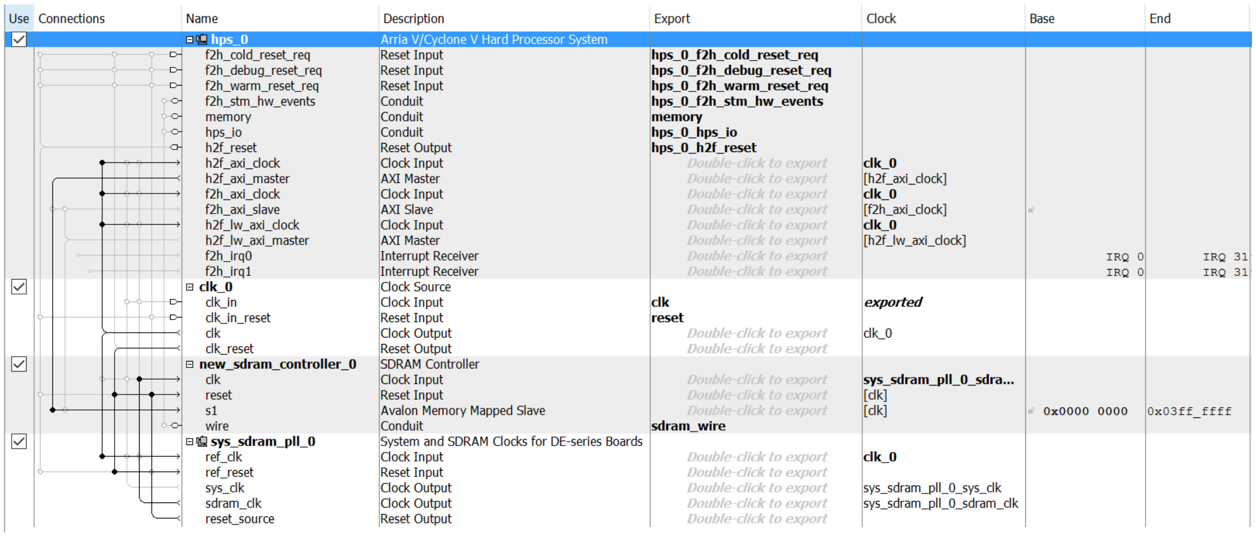


Figure 8-1: Picture of the AXI Bridge

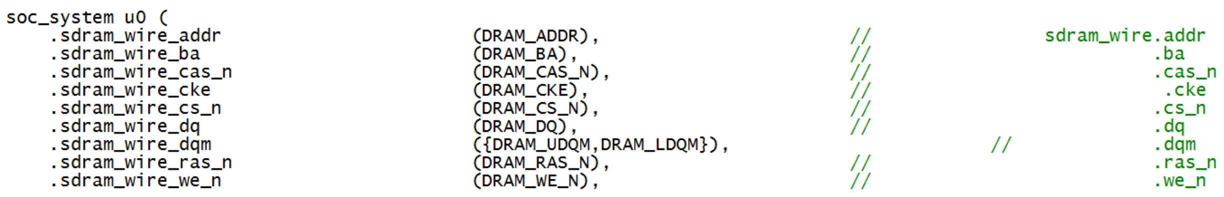
Using Quartus II and Altera system integration tool QSYS we were able to establish this communication link from the FPGA to the HPS and vice versa. Using Quartus, for all the components to be used on the FPGA, the pin locations to all these components must be mapped on the project. For FPGA components, the devices are allocated to specific pin locations that are for those purposes (Figure 8-4) but for the HPS components there are no pin locations available to declare, so they are just listed with pin directions and IO standards (Figure 8-5).

Using QSYS we are able to utilize the Avalon memory mapped master and slave to establish the communication link between HPS and FPGA. IP Cores are used to establish control over the HPS and SDRAM Controller (Figure 8-2). HPS serves as the master and the SDRAM is the slave so the AXI master is connected to Avalon memory mapped slave of the SDRAM. In order to correctly access the data from the SDRAM, the clock input would have to be changed. The board has a standard clock speed of 50MHz but the SDRAM desires a clock speed of 100MHz thus a the SDRAM controller takes an input of 100MHz from a SDRAM clock generator which uses Phased Lock Loop (PLL) to generate a 100MHz clock speed from 50MHz (Figure 8-2). Once these are all connected, the source code for the SDRAM Controller is generated (Figure 8-6) and added to the Verilog system module (Figure 8-3). Once these components are made a batch file generate_hps_qsys_header.sh is provided by Altera to generate a header file, hps_0.h, for the HPS C code (Figure 8-7).



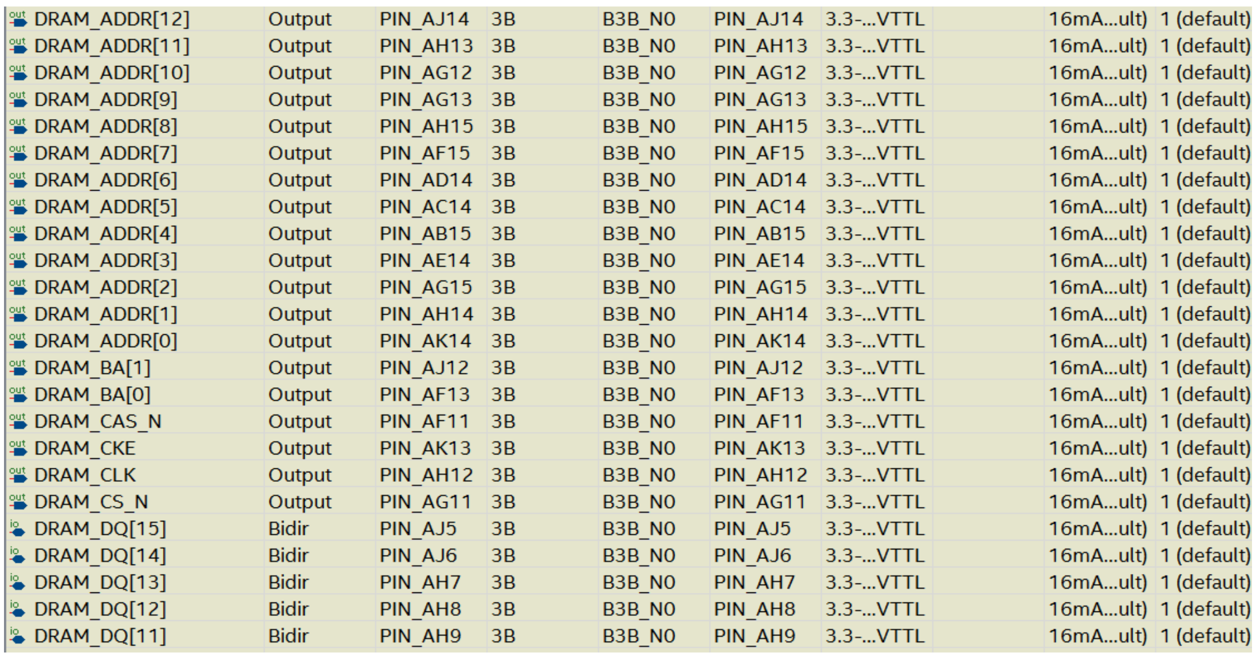
Use	Connections	Name	Description	Export	Clock	Base	End
✓		**hps_0**	Arria V/Cyclone V Hard Processor System				
		f2h_cold_reset_req	Reset Input	**hps_0_f2h_cold_reset_req**			
		f2h_debug_reset_req	Reset Input	**hps_0_f2h_debug_reset_req**			
		f2h_warm_reset_req	Reset Input	**hps_0_f2h_warm_reset_req**			
		f2h_stm_hw_events	Conduit	**hps_0_f2h_stm_hw_events**			
		memory	Conduit	**memory**			
		hps_io	Conduit	**hps_0_hps_io**			
		h2f_reset	Reset Output	**hps_0_h2f_reset**			
		h2f_axi_clock	Clock Input	*Double-click to export*	**clk_0**		
		h2f_axi_master	AXI Master	*Double-click to export*	[h2f_axi_clock]		
		f2h_axi_clock	Clock Input	*Double-click to export*	**clk_0**		
		f2h_axi_slave	AXI Slave	*Double-click to export*	[f2h_axi_clock]		
		h2f_lw_axi_clock	Clock Input	*Double-click to export*	**clk_0**		
		h2f_lw_axi_master	AXI Master	*Double-click to export*	[h2f_lw_axi_clock]		
		f2h_irq0	Interrupt Receiver	*Double-click to export*		IRQ 0	IRQ 31
		f2h_irq1	Interrupt Receiver	*Double-click to export*		IRQ 0	IRQ 31
✓		**clk_0**	Clock Source				
		clk_in	Clock Input	**clk**	***exported***		
		clk_in_reset	Reset Input	**reset**			
		clk	Clock Output	*Double-click to export*	clk_0		
		clk_reset	Reset Output	*Double-click to export*			
✓		**new_sdram_controller_0**	SDRAM Controller				
		clk	Clock Input	*Double-click to export*	**sys_sdram_pll_0_sdra...**		
		reset	Reset Input	*Double-click to export*	[clk]		
		s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x0000 0000	0x03ff_ffff
		wire	Conduit	**sdram_wire**			
✓		**sys_sdram_pll_0**	System and SDRAM Clocks for DE-series Boards				
		ref_clk	Clock Input	*Double-click to export*	**clk_0**		
		ref_reset	Reset Input	*Double-click to export*			
		sys_clk	Clock Output	*Double-click to export*	sys_sdram_pll_0_sys_clk		
		sdram_clk	Clock Output	*Double-click to export*	sys_sdram_pll_0_sdram_clk		
		reset_source	Reset Output	*Double-click to export*			

Figure 8-2: Screenshot of QSYS Avalon memory map



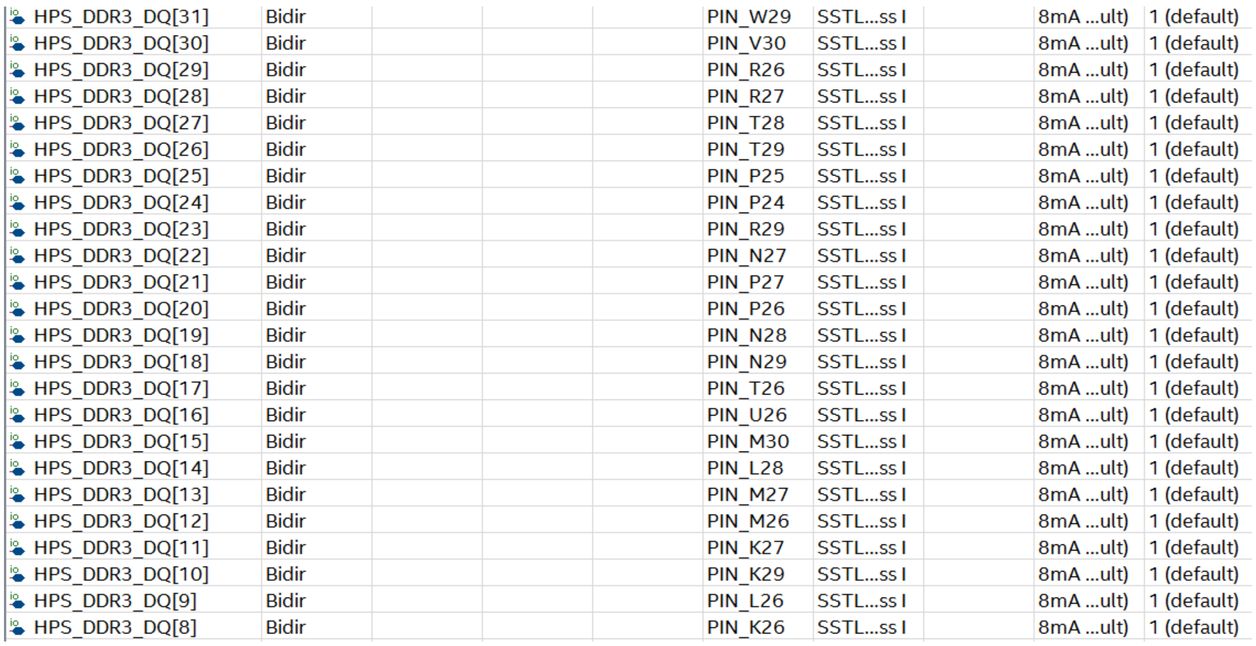
```
soc_system u0 (
    .sdram_wire_addr                        (DRAM_ADDR),                    //               sdram_wire.addr
    .sdram_wire_ba                          (DRAM_BA),                      //                         .ba
    .sdram_wire_cas_n                       (DRAM_CAS_N),                   //                         .cas_n
    .sdram_wire_cke                         (DRAM_CKE),                     //                          .cke
    .sdram_wire_cs_n                        (DRAM_CS_N),                    //                         .cs_n
    .sdram_wire_dq                          (DRAM_DQ),                      //                         .dq
    .sdram_wire_dqm                         ({DRAM_UDQM,DRAM_LDQM}),               //                  .dqm
    .sdram_wire_ras_n                       (DRAM_RAS_N),                   //                         .ras_n
    .sdram_wire_we_n                        (DRAM_WE_N),                    //                         .we_n
```

Figure 8-3: Screenshot of Quartus SDRAM source code



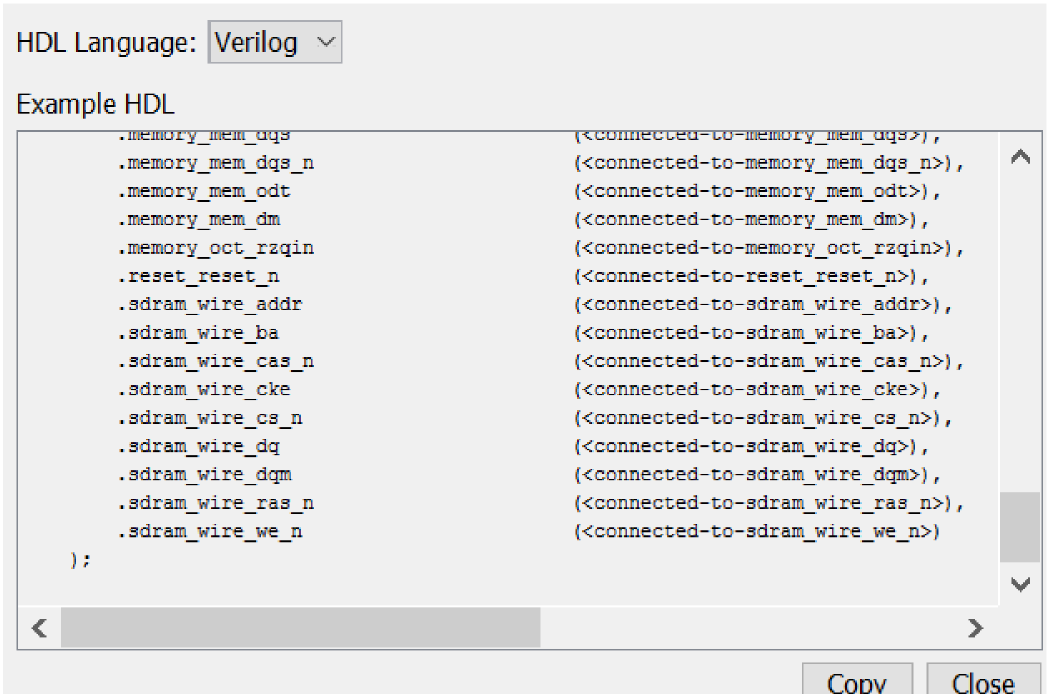
DRAM_ADDR[12]	Output	PIN_AJ14	3B	B3B_N0	PIN_AJ14	3.3-...VTTL		16mA...ult)	1 (default)
DRAM_ADDR[11]	Output	PIN_AH13	3B	B3B_N0	PIN_AH13	3.3-...VTTL		16mA...ult)	1 (default)
DRAM_ADDR[10]	Output	PIN_AG12	3B	B3B_N0	PIN_AG12	3.3-...VTTL		16mA...ult)	1 (default)
DRAM_ADDR[9]	Output	PIN_AG13	3B	B3B_N0	PIN_AG13	3.3-...VTTL		16mA...ult)	1 (default)
DRAM_ADDR[8]	Output	PIN_AH15	3B	B3B_N0	PIN_AH15	3.3-...VTTL		16mA...ult)	1 (default)
DRAM_ADDR[7]	Output	PIN_AF15	3B	B3B_N0	PIN_AF15	3.3-...VTTL		16mA...ult)	1 (default)
DRAM_ADDR[6]	Output	PIN_AD14	3B	B3B_N0	PIN_AD14	3.3-...VTTL		16mA...ult)	1 (default)
DRAM_ADDR[5]	Output	PIN_AC14	3B	B3B_N0	PIN_AC14	3.3-...VTTL		16mA...ult)	1 (default)
DRAM_ADDR[4]	Output	PIN_AB15	3B	B3B_N0	PIN_AB15	3.3-...VTTL		16mA...ult)	1 (default)
DRAM_ADDR[3]	Output	PIN_AE14	3B	B3B_N0	PIN_AE14	3.3-...VTTL		16mA...ult)	1 (default)
DRAM_ADDR[2]	Output	PIN_AG15	3B	B3B_N0	PIN_AG15	3.3-...VTTL		16mA...ult)	1 (default)
DRAM_ADDR[1]	Output	PIN_AH14	3B	B3B_N0	PIN_AH14	3.3-...VTTL		16mA...ult)	1 (default)
DRAM_ADDR[0]	Output	PIN_AK14	3B	B3B_N0	PIN_AK14	3.3-...VTTL		16mA...ult)	1 (default)
DRAM_BA[1]	Output	PIN_AJ12	3B	B3B_N0	PIN_AJ12	3.3-...VTTL		16mA...ult)	1 (default)
DRAM_BA[0]	Output	PIN_AF13	3B	B3B_N0	PIN_AF13	3.3-...VTTL		16mA...ult)	1 (default)
DRAM_CAS_N	Output	PIN_AF11	3B	B3B_N0	PIN_AF11	3.3-...VTTL		16mA...ult)	1 (default)
DRAM_CKE	Output	PIN_AK13	3B	B3B_N0	PIN_AK13	3.3-...VTTL		16mA...ult)	1 (default)
DRAM_CLK	Output	PIN_AH12	3B	B3B_N0	PIN_AH12	3.3-...VTTL		16mA...ult)	1 (default)
DRAM_CS_N	Output	PIN_AG11	3B	B3B_N0	PIN_AG11	3.3-...VTTL		16mA...ult)	1 (default)
DRAM_DQ[15]	Bidir	PIN_AJ5	3B	B3B_N0	PIN_AJ5	3.3-...VTTL		16mA...ult)	1 (default)
DRAM_DQ[14]	Bidir	PIN_AJ6	3B	B3B_N0	PIN_AJ6	3.3-...VTTL		16mA...ult)	1 (default)
DRAM_DQ[13]	Bidir	PIN_AH7	3B	B3B_N0	PIN_AH7	3.3-...VTTL		16mA...ult)	1 (default)
DRAM_DQ[12]	Bidir	PIN_AH8	3B	B3B_N0	PIN_AH8	3.3-...VTTL		16mA...ult)	1 (default)
DRAM_DQ[11]	Bidir	PIN_AH9	3B	B3B_N0	PIN_AH9	3.3-...VTTL		16mA...ult)	1 (default)

Figure 8-4: Screenshot of Pin programmer for SDRAM



HPS_DDR3_DQ[31]	Bidir				PIN_W29	SSTL...ss I		8mA ...ult)	1 (default)
HPS_DDR3_DQ[30]	Bidir				PIN_V30	SSTL...ss I		8mA ...ult)	1 (default)
HPS_DDR3_DQ[29]	Bidir				PIN_R26	SSTL...ss I		8mA ...ult)	1 (default)
HPS_DDR3_DQ[28]	Bidir				PIN_R27	SSTL...ss I		8mA ...ult)	1 (default)
HPS_DDR3_DQ[27]	Bidir				PIN_T28	SSTL...ss I		8mA ...ult)	1 (default)
HPS_DDR3_DQ[26]	Bidir				PIN_T29	SSTL...ss I		8mA ...ult)	1 (default)
HPS_DDR3_DQ[25]	Bidir				PIN_P25	SSTL...ss I		8mA ...ult)	1 (default)
HPS_DDR3_DQ[24]	Bidir				PIN_P24	SSTL...ss I		8mA ...ult)	1 (default)
HPS_DDR3_DQ[23]	Bidir				PIN_R29	SSTL...ss I		8mA ...ult)	1 (default)
HPS_DDR3_DQ[22]	Bidir				PIN_N27	SSTL...ss I		8mA ...ult)	1 (default)
HPS_DDR3_DQ[21]	Bidir				PIN_P27	SSTL...ss I		8mA ...ult)	1 (default)
HPS_DDR3_DQ[20]	Bidir				PIN_P26	SSTL...ss I		8mA ...ult)	1 (default)
HPS_DDR3_DQ[19]	Bidir				PIN_N28	SSTL...ss I		8mA ...ult)	1 (default)
HPS_DDR3_DQ[18]	Bidir				PIN_N29	SSTL...ss I		8mA ...ult)	1 (default)
HPS_DDR3_DQ[17]	Bidir				PIN_T26	SSTL...ss I		8mA ...ult)	1 (default)
HPS_DDR3_DQ[16]	Bidir				PIN_U26	SSTL...ss I		8mA ...ult)	1 (default)
HPS_DDR3_DQ[15]	Bidir				PIN_M30	SSTL...ss I		8mA ...ult)	1 (default)
HPS_DDR3_DQ[14]	Bidir				PIN_L28	SSTL...ss I		8mA ...ult)	1 (default)
HPS_DDR3_DQ[13]	Bidir				PIN_M27	SSTL...ss I		8mA ...ult)	1 (default)
HPS_DDR3_DQ[12]	Bidir				PIN_M26	SSTL...ss I		8mA ...ult)	1 (default)
HPS_DDR3_DQ[11]	Bidir				PIN_K27	SSTL...ss I		8mA ...ult)	1 (default)
HPS_DDR3_DQ[10]	Bidir				PIN_K29	SSTL...ss I		8mA ...ult)	1 (default)
HPS_DDR3_DQ[9]	Bidir				PIN_L26	SSTL...ss I		8mA ...ult)	1 (default)
HPS_DDR3_DQ[8]	Bidir				PIN_K26	SSTL...ss I		8mA ...ult)	1 (default)

Figure 8-5: Screenshot of Pin programmer for HPS DDR3 RAM

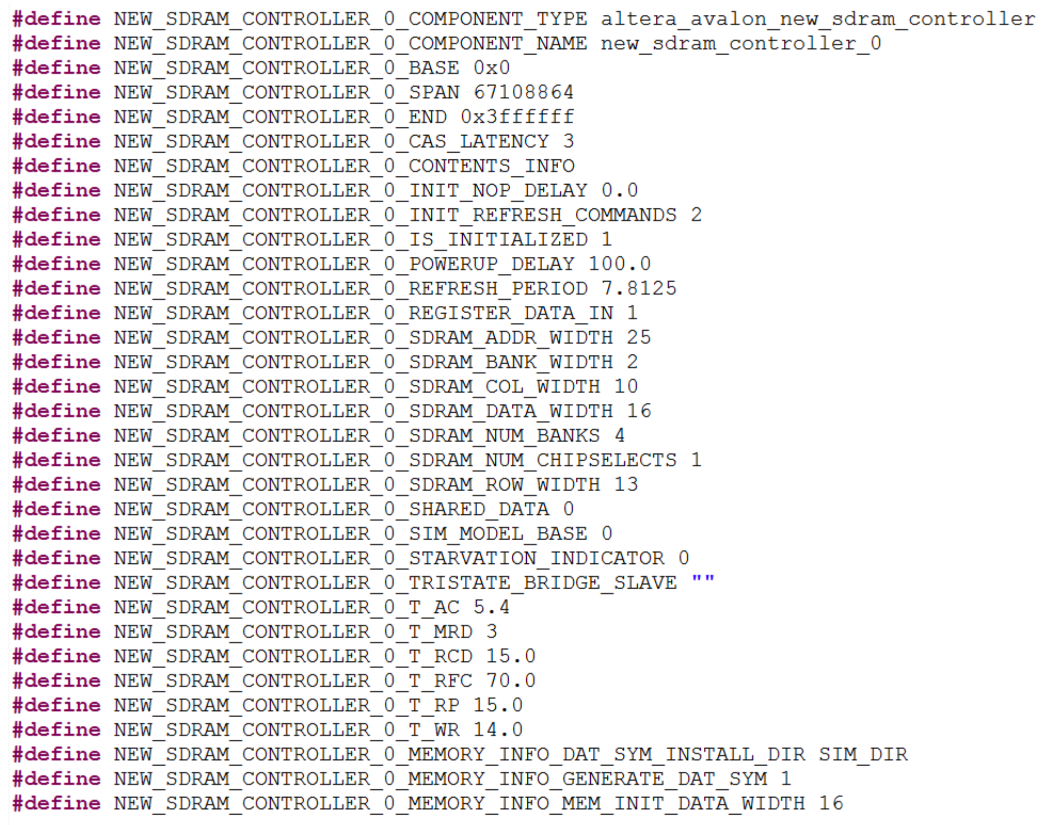


HDL Language: Verilog

Example HDL

```
        .memory_mem_dqs                         (<connected-to-memory_mem_dqs>),
        .memory_mem_dqs_n                       (<connected-to-memory_mem_dqs_n>),
        .memory_mem_odt                         (<connected-to-memory_mem_odt>),
        .memory_mem_dm                          (<connected-to-memory_mem_dm>),
        .memory_oct_rzqin                       (<connected-to-memory_oct_rzqin>),
        .reset_reset_n                          (<connected-to-reset_reset_n>),
        .sdram_wire_addr                        (<connected-to-sdram_wire_addr>),
        .sdram_wire_ba                          (<connected-to-sdram_wire_ba>),
        .sdram_wire_cas_n                       (<connected-to-sdram_wire_cas_n>),
        .sdram_wire_cke                         (<connected-to-sdram_wire_cke>),
        .sdram_wire_cs_n                        (<connected-to-sdram_wire_cs_n>),
        .sdram_wire_dq                          (<connected-to-sdram_wire_dq>),
        .sdram_wire_dqm                         (<connected-to-sdram_wire_dqm>),
        .sdram_wire_ras_n                       (<connected-to-sdram_wire_ras_n>),
        .sdram_wire_we_n                        (<connected-to-sdram_wire_we_n>)
    );
```

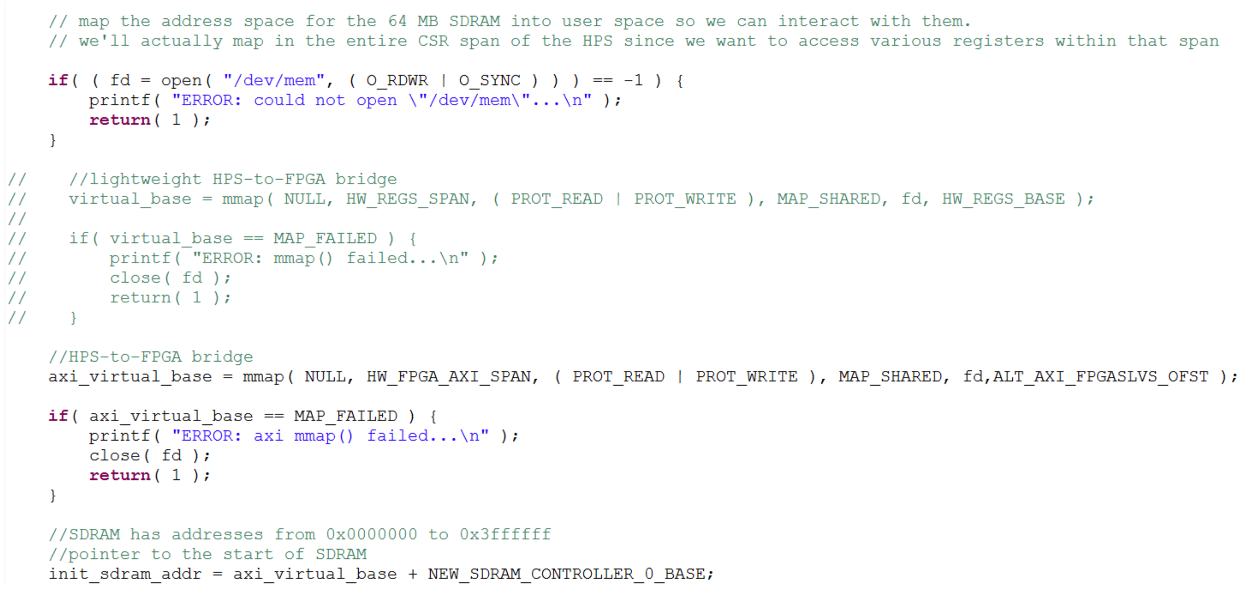
Figure 8-6: Screenshot of generated source code for system in QSYS



```
#define NEW_SDRAM_CONTROLLER_0_COMPONENT_TYPE altera_avalon_new_sdram_controller
#define NEW_SDRAM_CONTROLLER_0_COMPONENT_NAME new_sdram_controller_0
#define NEW_SDRAM_CONTROLLER_0_BASE 0x0
#define NEW_SDRAM_CONTROLLER_0_SPAN 67108864
#define NEW_SDRAM_CONTROLLER_0_END 0x3ffffff
#define NEW_SDRAM_CONTROLLER_0_CAS_LATENCY 3
#define NEW_SDRAM_CONTROLLER_0_CONTENTS_INFO
#define NEW_SDRAM_CONTROLLER_0_INIT_NOP_DELAY 0.0
#define NEW_SDRAM_CONTROLLER_0_INIT_REFRESH_COMMANDS 2
#define NEW_SDRAM_CONTROLLER_0_IS_INITIALIZED 1
#define NEW_SDRAM_CONTROLLER_0_POWERUP_DELAY 100.0
#define NEW_SDRAM_CONTROLLER_0_REFRESH_PERIOD 7.8125
#define NEW_SDRAM_CONTROLLER_0_REGISTER_DATA_IN 1
#define NEW_SDRAM_CONTROLLER_0_SDRAM_ADDR_WIDTH 25
#define NEW_SDRAM_CONTROLLER_0_SDRAM_BANK_WIDTH 2
#define NEW_SDRAM_CONTROLLER_0_SDRAM_COL_WIDTH 10
#define NEW_SDRAM_CONTROLLER_0_SDRAM_DATA_WIDTH 16
#define NEW_SDRAM_CONTROLLER_0_SDRAM_NUM_BANKS 4
#define NEW_SDRAM_CONTROLLER_0_SDRAM_NUM_CHIPSELECTS 1
#define NEW_SDRAM_CONTROLLER_0_SDRAM_ROW_WIDTH 13
#define NEW_SDRAM_CONTROLLER_0_SHARED_DATA 0
#define NEW_SDRAM_CONTROLLER_0_SIM_MODEL_BASE 0
#define NEW_SDRAM_CONTROLLER_0_STARVATION_INDICATOR 0
#define NEW_SDRAM_CONTROLLER_0_TRISTATE_BRIDGE_SLAVE ""
#define NEW_SDRAM_CONTROLLER_0_T_AC 5.4
#define NEW_SDRAM_CONTROLLER_0_T_MRD 3
#define NEW_SDRAM_CONTROLLER_0_T_RCD 15.0
#define NEW_SDRAM_CONTROLLER_0_T_RFC 70.0
#define NEW_SDRAM_CONTROLLER_0_T_RP 15.0
#define NEW_SDRAM_CONTROLLER_0_T_WR 14.0
#define NEW_SDRAM_CONTROLLER_0_MEMORY_INFO_DAT_SYM_INSTALL_DIR SIM_DIR
#define NEW_SDRAM_CONTROLLER_0_MEMORY_INFO_GENERATE_DAT_SYM 1
#define NEW_SDRAM_CONTROLLER_0_MEMORY_INFO_MEM_INIT_DATA_WIDTH 16
```

Figure 8-7: Screenshot of hps_0.h

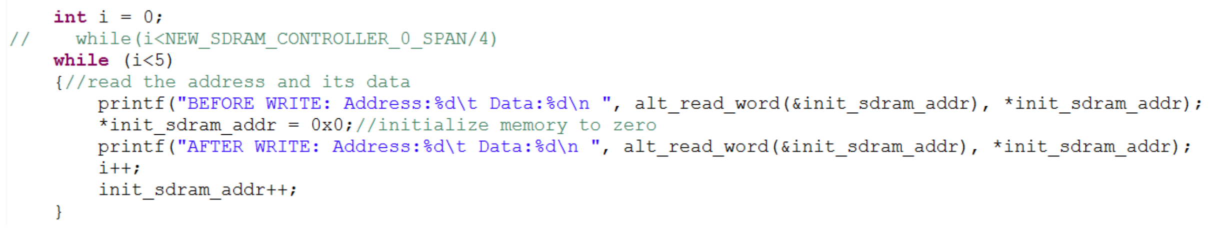
Once all the links are made and all the groundwork for the SDRAM and HPS connections are set, we would be able to use the HPS to receive and write data to and read data from the SDRAM. Using Altera provided Eclipse for the DE1 SOC board, in C code we were able to use memory mapping to map physical memory addresses on the board to virtual memory addresses. Here we are able to access the memory device driver "/dev/mem" with the open system call then using the mmap system call to map the HPS physical address to a virtual address. From here we are able to use the SDRAM offset that is provided from the generated header file to be able to correctly use the designated memory address mapped for the SDRAM (Figure 8-8). From here we use C code to navigate through the memory address of the SDRAM for read from and write to functions.



```
    // map the address space for the 64 MB SDRAM into user space so we can interact with them.
    // we'll actually map in the entire CSR span of the HPS since we want to access various registers within that span

    if( ( fd = open( "/dev/mem", ( O_RDWR | O_SYNC ) ) ) == -1 ) {
        printf( "ERROR: could not open \"/dev/mem\"...\n" );
        return( 1 );
    }

//    //lightweight HPS-to-FPGA bridge
//    virtual_base = mmap( NULL, HW_REGS_SPAN, ( PROT_READ | PROT_WRITE ), MAP_SHARED, fd, HW_REGS_BASE );
//
//    if( virtual_base == MAP_FAILED ) {
//        printf( "ERROR: mmap() failed...\n" );
//        close( fd );
//        return( 1 );
//    }

    //HPS-to-FPGA bridge
    axi_virtual_base = mmap( NULL, HW_FPGA_AXI_SPAN, ( PROT_READ | PROT_WRITE ), MAP_SHARED, fd,ALT_AXI_FPGASLVS_OFST );

    if( axi_virtual_base == MAP_FAILED ) {
        printf( "ERROR: axi mmap() failed...\n" );
        close( fd );
        return( 1 );
    }

    //SDRAM has addresses from 0x0000000 to 0x3ffffff
    //pointer to the start of SDRAM
    init_sdram_addr = axi_virtual_base + NEW_SDRAM_CONTROLLER_0_BASE;
```

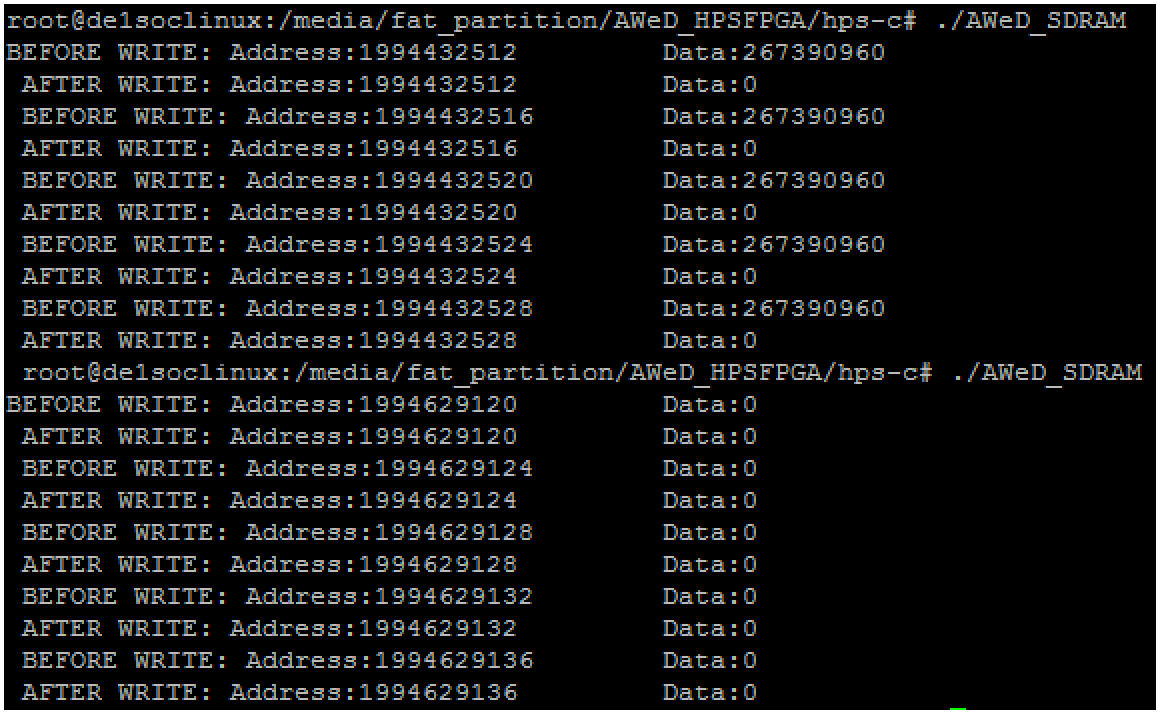
Figure 8-8: Screenshot of SDRAM virtual memory mapping

For testing the SDRAM, we wrote C code that maps the SDRAM memory and accesses five consecutive 32 bit memory locations (Figure 8-9). A while loop is used to increment a counter variable that allows us to move through these five consecutive 32 bit memory slots whilst read and write operations are done on the memory. For each memory location the address and data are printed out using the printf command, then using pointer logic the data in the memory location is initialized to 0x0 then the same memory address and data are printed out, but with the memory data value changed (Figure 8-10). A makefile provided by Altera used to compile C code for the ARM architecture was used to compile the executable to test the SDRAM (Figure 8-11). In Figure j, you would realize that the executable AWeD_SDRAM is run twice (./AWeD_SDRAM), this is to show that the previous memory written to with 0x0 was indeed stored.



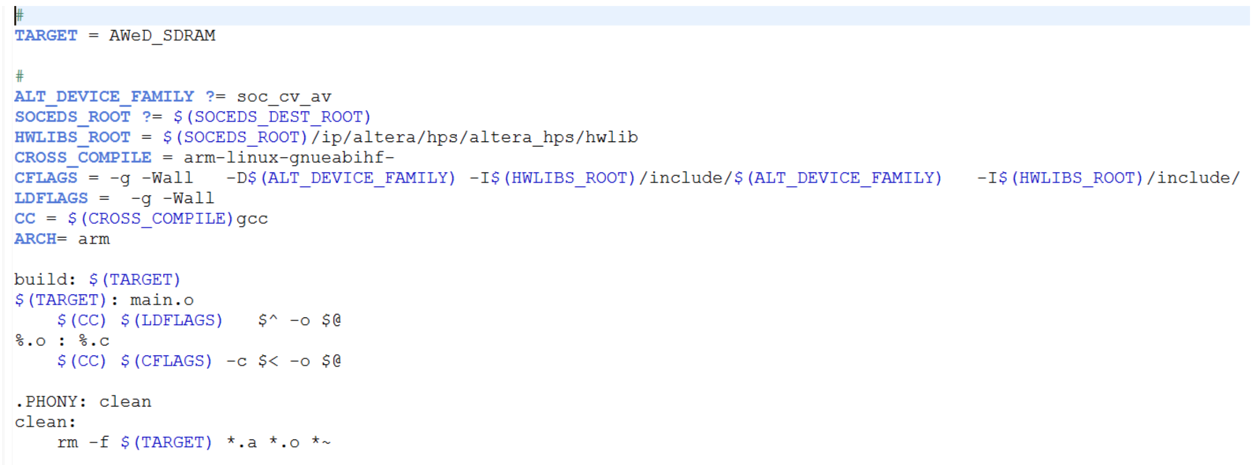
```
    int i = 0;
//    while(i<NEW_SDRAM_CONTROLLER_0_SPAN/4)
    while (i<5)
    {//read the address and its data
        printf("BEFORE WRITE: Address:%d\t Data:%d\n ", alt_read_word(&init_sdram_addr), *init_sdram_addr);
        *init_sdram_addr = 0x0;//initialize memory to zero
        printf("AFTER WRITE: Address:%d\t Data:%d\n ", alt_read_word(&init_sdram_addr), *init_sdram_addr);
        i++;
        init_sdram_addr++;
    }
```

Figure 8-9: Screenshot of the test C Code



```
root@de1soclinux:/media/fat_partition/AWeD_HPSFPGA/hps-c# ./AWeD_SDRAM
BEFORE WRITE: Address:1994432512         Data:267390960
 AFTER WRITE: Address:1994432512         Data:0
 BEFORE WRITE: Address:1994432516        Data:267390960
 AFTER WRITE: Address:1994432516         Data:0
 BEFORE WRITE: Address:1994432520        Data:267390960
 AFTER WRITE: Address:1994432520         Data:0
 BEFORE WRITE: Address:1994432524        Data:267390960
 AFTER WRITE: Address:1994432524         Data:0
 BEFORE WRITE: Address:1994432528        Data:267390960
 AFTER WRITE: Address:1994432528         Data:0
 root@de1soclinux:/media/fat_partition/AWeD_HPSFPGA/hps-c# ./AWeD_SDRAM
BEFORE WRITE: Address:1994629120         Data:0
 AFTER WRITE: Address:1994629120         Data:0
 BEFORE WRITE: Address:1994629124        Data:0
 AFTER WRITE: Address:1994629124         Data:0
 BEFORE WRITE: Address:1994629128        Data:0
 AFTER WRITE: Address:1994629128         Data:0
 BEFORE WRITE: Address:1994629132        Data:0
 AFTER WRITE: Address:1994629132         Data:0
 BEFORE WRITE: Address:1994629136        Data:0
 AFTER WRITE: Address:1994629136         Data:0
```

Figure 8-10: Screenshot of the test output

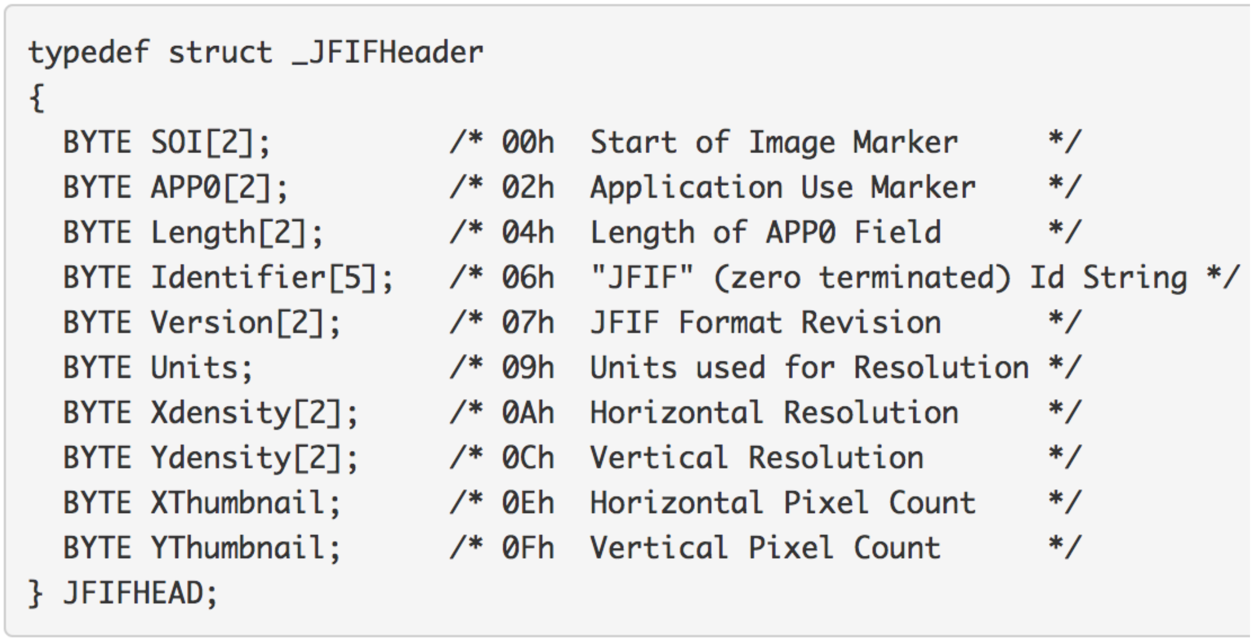


```
#
TARGET = AWeD_SDRAM

#
ALT_DEVICE_FAMILY ?= soc_cv_av
SOCEDS_ROOT ?= $(SOCEDS_DEST_ROOT)
HWLIBS_ROOT = $(SOCEDS_ROOT)/ip/altera/hps/altera_hps/hwlib
CROSS_COMPILE = arm-linux-gnueabihf-
CFLAGS = -g -Wall   -D$(ALT_DEVICE_FAMILY) -I$(HWLIBS_ROOT)/include/$(ALT_DEVICE_FAMILY)   -I$(HWLIBS_ROOT)/include/
LDFLAGS =  -g -Wall
CC = $(CROSS_COMPILE)gcc
ARCH= arm

build: $(TARGET)
$(TARGET): main.o
    $(CC) $(LDFLAGS)   $^ -o $@
%.o : %.c
    $(CC) $(CFLAGS) -c $< -o $@

.PHONY: clean
clean:
    rm -f $(TARGET) *.a *.o *~
```

Figure 8-11: Screenshot of the Makefile

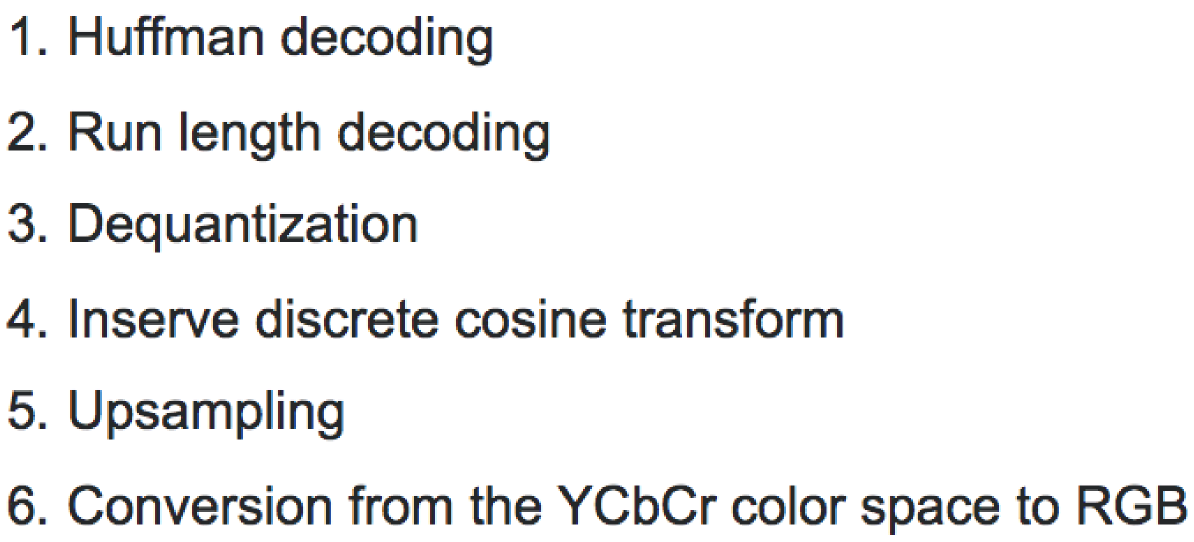
Filters

Our first approach to be able to test the algorithms was to test them using modelsim. To be able to do this we needed to decompress the JPEG into raw data. The raw data would be stored in a text file that would be inputted to the testbench. When we first thought about the problem for extracting the pixel array from the encoded JPEG image, we researched into how to extract raw image data on a byte level. We found out that each JPEG image has a header that consists of 20 bytes of information that defines the file as a JPEG (Figure 8-12). The first two bytes of a JPEG image are FF D8 and the last 2 bytes are FF D9, this made it easy to identify an image in memory. After being able to identify where the data needed from the byte stream where, we needed to figure out how we would be able to identify the red, green and blue (RGB) values of each pixel. Here we came into an obstacle as we realized that in order to get the RGB values of the pixels in a JPEG we would have to code a few decoding processes to run on the JPEG file (Figure 8-13). To solve this problem we sought to use functions from an open source JPEG library called, jpeglib, to extract the 2-D pixel array of the JPEG images being stored. However we found out that these functions would add more instructions for the ARM processor which affects the performance of the machine learning algorithm. After this consideration, Matlab was our third option. We came across two Matlab scripts, ImageToText.m and TextToImage.m, that were able to extract the pixel bytes from the image and store it in a text file and vice versa. The original image and its corresponding raw byte file can be found in figures 8-14 and 8-15 respectively.



```
typedef struct _JFIFHeader
{
  BYTE SOI[2];          /* 00h  Start of Image Marker     */
  BYTE APP0[2];         /* 02h  Application Use Marker    */
  BYTE Length[2];       /* 04h  Length of APP0 Field      */
  BYTE Identifier[5];   /* 06h  "JFIF" (zero terminated) Id String */
  BYTE Version[2];      /* 07h  JFIF Format Revision      */
  BYTE Units;           /* 09h  Units used for Resolution */
  BYTE Xdensity[2];     /* 0Ah  Horizontal Resolution     */
  BYTE Ydensity[2];     /* 0Ch  Vertical Resolution       */
  BYTE XThumbnail;      /* 0Eh  Horizontal Pixel Count    */
  BYTE YThumbnail;      /* 0Fh  Vertical Pixel Count      */
} JFIFHEAD;
```

Figure 8-12: Structure of a JPEG header file [31]



1. Huffman decoding
2. Run length decoding
3. Dequantization
4. Inserve discrete cosine transform
5. Upsampling
6. Conversion from the YCbCr color space to RGB

Figure 8-13: Processes to decode the RGB from the JPEG encoded format [32]

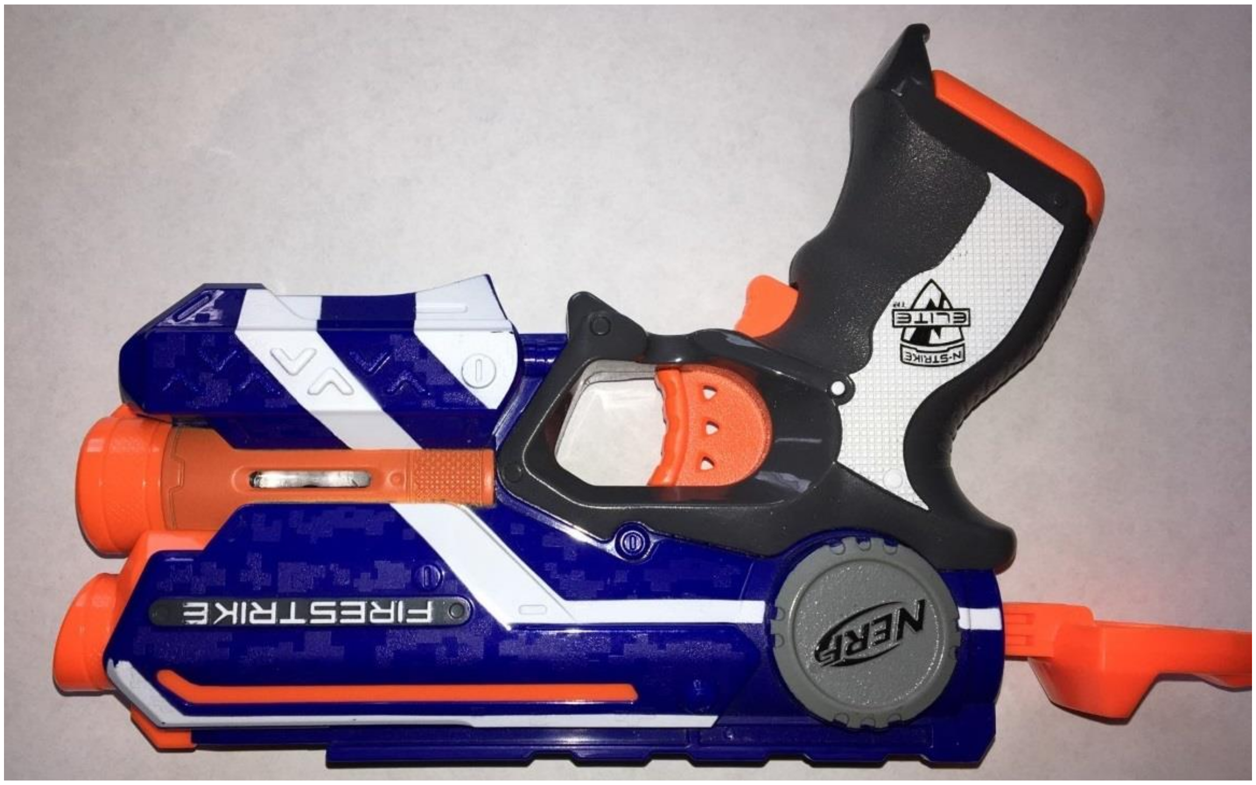
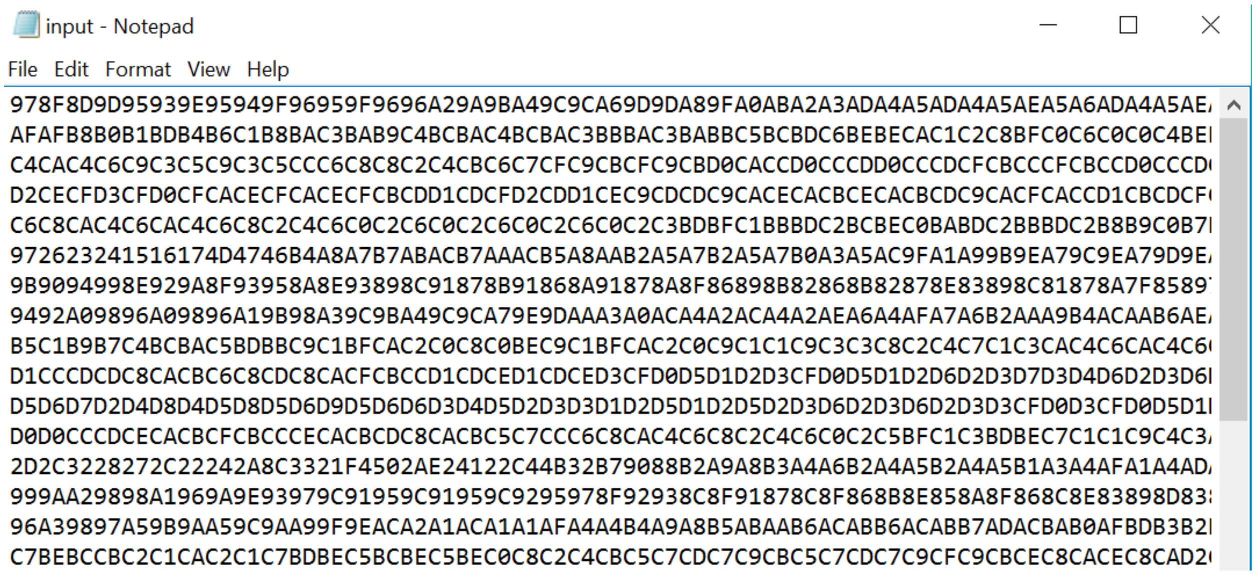


Figure 8-14: Original Nerf gun image



```
978F8D9D95939E95949F96959F9696A29A9BA49C9CA69D9DA89FA0ABA2A3ADA4A5ADA4A5AEA5A6ADA4A5AE
AFAFB8B0B1BDB4B6C1B8BAC3BAB9C4BCBAC4BCBAC3BBBAC3BABBC5BCBDC6BEBECAC1C2C8BFC0C6C0C0C4BE
C4CAC4C6C9C3C5C9C3C5CCC6C8C8C2C4CBC6C7CFC9CBCFC9CBD0CACCD0CCCDD0CCCDCFCBCCCFCBCCD0CCCD
D2CECFD3CFD0CFCACECFCACECFCBCDD1CDCFD2CDD1CEC9CDCDC9CACECACBCECACBCDC9CACFCACCD1CBCDCF
C6C8CAC4C6CAC4C6C8C2C4C6C0C2C6C0C2C6C0C2C6C0C2C3BDBFC1BBBDC2BCBEC0BABDC2BBBDC2B8B9C0B7
972623241516174D4746B4A8A7B7ABACB7AAACB5A8AAB2A5A7B2A5A7B0A3A5AC9FA1A99B9EA79C9EA79D9E
9B9094998E929A8F93958A8E93898C91878B91868A91878A8F86898B82868B82878E83898C81878A7F8589
9492A09896A09896A19B98A39C9BA49C9CA79E9DAAA3A0ACA4A2ACA4A2AEA6A4AFA7A6B2AAA9B4ACAAB6AE
B5C1B9B7C4BCBAC5BDBBC9C1BFCAC2C0C8C0BEC9C1BFCAC2C0C9C1C1C9C3C3C8C2C4C7C1C3CAC4C6CAC4C6
D1CCCDCDC8CACBC6C8CDC8CACFCBCCD1CDCED1CDCED3CFD0D5D1D2D3CFD0D5D1D2D6D2D3D7D3D4D6D2D3D6
D5D6D7D2D4D8D4D5D8D5D6D9D5D6D6D3D4D5D2D3D3D1D2D5D1D2D5D2D3D6D2D3D6D2D3D3CFD0D3CFD0D5D1
D0D0CCCDCECACBCFCBCCCECACBCDC8CACBC5C7CCC6C8CAC4C6C8C2C4C6C0C2C5BFC1C3BDBEC7C1C1C9C4C3
2D2C3228272C22242A8C3321F4502AE24122C44B32B79088B2A9A8B3A4A6B2A4A5B2A4A5B1A3A4AFA1A4AD
999AA29898A1969A9E93979C91959C91959C9295978F92938C8F91878C8F868B8E858A8F868C8E83898D83
96A39897A59B9AA59C9AA99F9EACA2A1ACA1A1AFA4A4B4A9A8B5ABAAB6ACABB6ACABB7ADACBAB0AFBDB3B2
C7BEBCCBC2C1CAC2C1C7BDBEC5BCBEC5BEC0C8C2C4CBC5C7CDC7C9CBC5C7CDC7C9CFC9CBCEC8CACEC8CAD2
```

Figure 8-15: Screenshot of the raw byte text file of original Nerf gun image

As you can see above the image is now converted from a JPEG to a raw byte text file. If the text file above was to be run in through the text file to image convertor we would see the original nerf gun image.

Greyscale color conversion is the first filter that is implemented. This filter reduces the data needed to be processed by 3 from a 3 channel Red, Green, Blue (RGB) to a single channel grayscale. This filter takes each R, G and B values as input and outputs a single set of values for the grayscaled image. The implementation of the grayscale conversion hinged on this line of Verilog code:

```
assign Y = (R>>2)+(R>>5)+(G>>1)+(G>>4)+(B>>4)+(B>>5);
```

The Verilog code is written with bitwise operations rather than solely arithmetic operations not only because of the ease of conversion but also because it would be able to run faster and with less memory used than arithmetic functions.

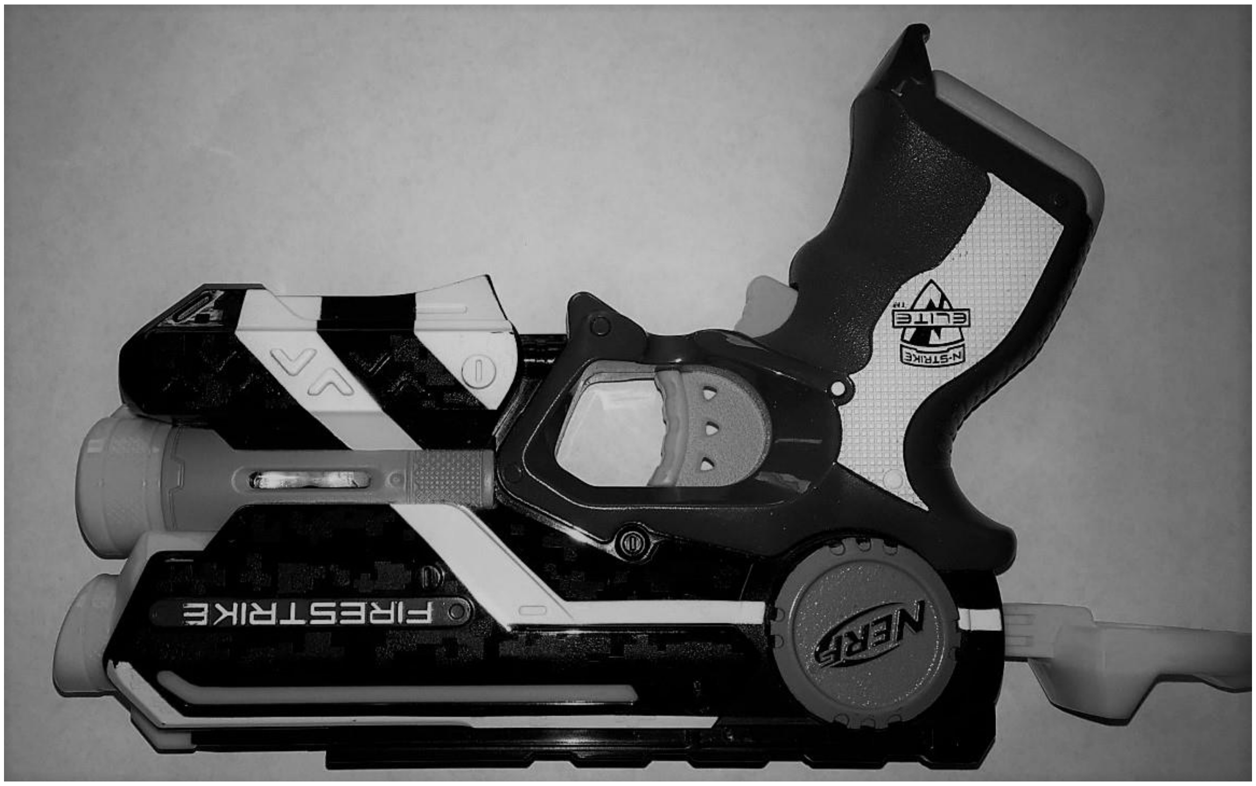
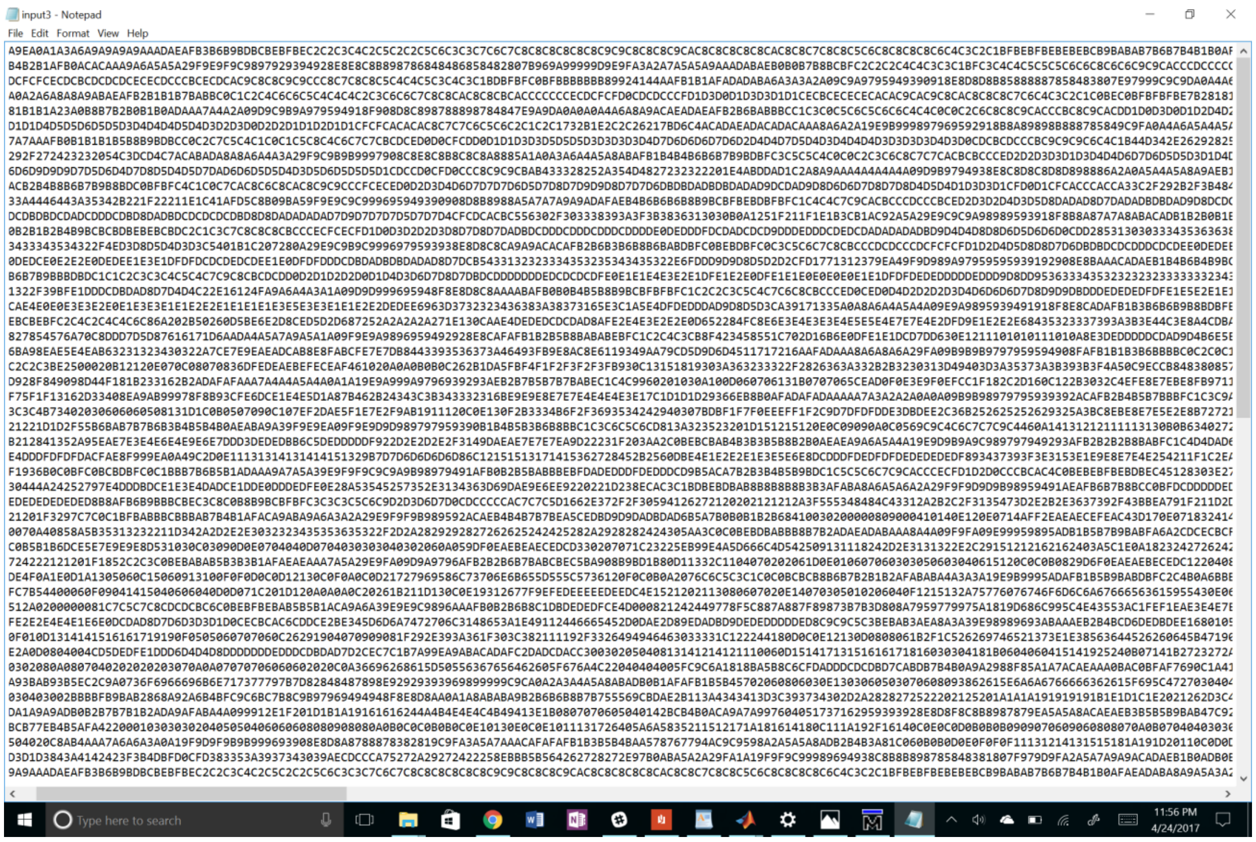


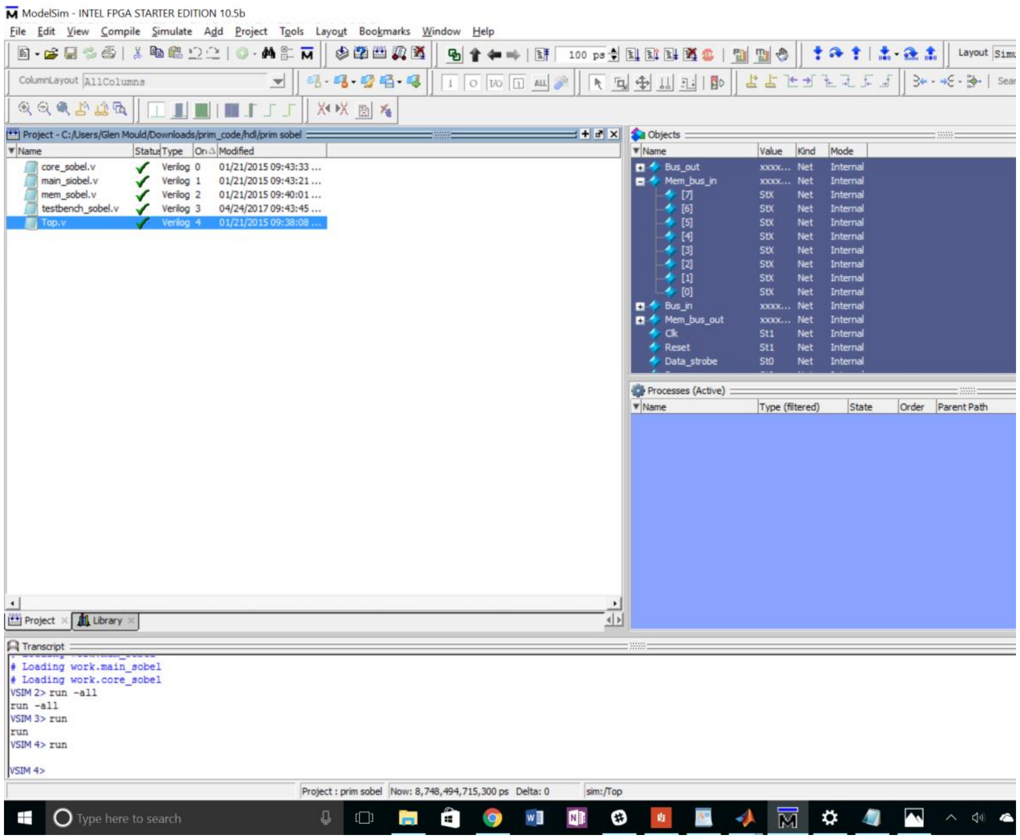
Figure 8-16: grayscale image



```
A9EA0A1A3A6A9A9A9A9AAADAEAFB3B6B9BDBCBEBFBEC2C2C3C4C2C5C2C2C5C6C3C3C7C6C7C8C8C8C8C8C9C9C8C8C8C9CAC8C8C8C8C8CAC8C8C7C8C8C5C6C8C8C8C8C6C4C3C2C1BFBEBFBEBEBEBCB9BABAB7B6B7B4B1B0AF
B4B2B1AFB0ACACAAA9A6A5A5A29F9E9F9C9897929394928E8E8C8B898786848486858482807B969A99999D9E9FA3A2A7A5A5A9AAADABAEB0B0B7B8BCBFC2C2C2C4C4C3C3C1BFC3C4C4C5C5C5C6C6C8C6C6C9C9CACCCDCCCC
DCFCFCECDCBCDCDCDCECECDCCCBCECDCAC9C8C8C9C9CCC8C7C8C8C5C4C4C5C3C4C3C1BDBFBFC0BFBBBBBB89924144AAFB1B1AFADADABA6A3A3A2A09C9A9795949390918E8D8D8B85888887858483807E97999C9C9DA0A4A
A0A2A6A8A8A9ABAEAFB2B1B1B7BABBC0C1C2C4C6C6C5C4C4C4C2C3C6C6C7C8C8CAC8C8CBCACCCCCCECDCFCFD0CDCDCCCFD1D3D0D1D3D3D1D1CECBCECECECACAC9CAC9C8CAC8C8C8C7C6C4C3C2C1C0BEC0BFBFBFBE7B28181
81B1B1A23A0B8B7B2B0B1B0ADAAA7A4A2A09D9C9B9A979594918F908D8C898788898784847E9A9DA0A0A0A4A6A8A9ACAEADAEAFB2B6BABBBCC1C3C0C5C6C5C6C6C4C4C0C0C2C6C8C8C9CACCCBC8C9CACDD1D0D3D0D1D2D4D2
D1D1D4D5D5D6D5D5D3D4D4D4D5D4D3D2D3D0D2D2D1D1D2D1D1CFCFCACACAC8C7C7C6C5C6C2C1C2C1732B1E2C2C26217BD6C4ACADAEADACADACAAA8A6A2A19E9B999897969592918B8A89898B888785849C9FA0A4A6A5A4A5
7A7AAAFB0B1B1B1B5B8B9BDBCC0C2C7C5C4C1C0C1C5C8C4C6C7C7CBCDCED0D0CFCDD0D1D1D3D3D5D5D5D3D3D3D3D4D7D6D6D6D7D6D2D4D4D7D5D4D3D4D4D4D3D3D3D3D4D3D0CDCBCDCCCBC9C9C9C6C4C1B44D342E2629282
292F272423232054C3DCD4C7ACABADA8A8A6A4A3A29F9C9B9B9997908C8E8C8B8C8C8A8885A1A0A3A6A4A5A8ABAFB1B4B4B6B6B7B9BDBFC3C5C5C4C0C0C2C3C6C8C7C7CACBCBCCCED2D2D3D3D1D3D4D4D6D7D6D5D5D3D1D4
6D6D9D9D9D7D5D6D4D7D8D5D4D5D7DAD6D6D5D5D4D3D5D6D5D5D5D1CDCCD0CFD0CCC8C9C9CBAB433328252A354D4827232322201E4ABDDAD1C2A8A9AAA4A4A4A4A09D9B9794938E8C8D8C8D8D898886A2A0A5A4A5A8A9AEB1
ACB2B4B8B6B7B9B8BDC0BFBFC4C1C0C7CAC8C6C8CAC8C9C9CCCFCECED0D2D3D4D6D7D7D7D6D5D7D8D7D9D9D8D7D7D6DBDBDADBDBDADAD9DCDAD9D8D6D6D7D8D7D8D4D5D4D1D3D3D1CFD0D1CFCACCCACCA33C2F292B2F3B484
33A4446443A35342B221F22211E1C41AFD5C8B09BA59F9E9C9C999695949390908D8B8988A5A7A7A9A9ADAFAEB4B6B6B6B8B9BCBFBEBDBFBFC1C4C4C7C9CACBCCCDCCCBCED2D3D2D4D3D5D8DADAD8D7DADADBDBDAD9D8DCDC
DCDBDBDCDADCDDDCDBD8DADBDCDCDCDCDBD8D8DADADADAD7D9D7D7D7D5D7D7D4CFCDCACBC556302F303338393A3F3B3836313030B0A1251F211F1E1B3CB1AC92A5A29E9C9C9A98989593918F8B8A87A7A8ABACADB1B2B0B1
0B2B1B2B4B9BCBCBDBEBEBCBDC2C1C3C7C8C8C8CBCCCECFCECFD1D0D3D2D2D3D8D7D8D7DADBDCDDDCDDDCDDDCDDDDE0DEDDDFDCDADCDCD9DDDEDDDCDEDCDADADADADBD9D4D4D8D8D6D5D6D6D0CDD2853130303343536363
3433343534322F4ED3D8D5D4D3D3C5401B1C207280A29E9C9B9C9996979593938E8D8C8CA9A9ACACAFB2B6B3B6B8B6BABDBFC0BEBDBFC0C3C5C6C7C8CBCCCDCDCCCDCFCFCFD1D2D4D5D8D8D7D6DBDBDCDCDDDCDCDEE0DEDEE
0DEDCE0E2E2E0DEDEE1E3E1DFDFDCDCDEDCDEE1E0DFDFDDDCDBDADBDBDADAD8D7DCB5433132323334353235343435322E6FDDD9D9D8D5D2D2CFD1771312379EA49F9D989A97959595939192908E8BAAACADAEB1B4B6B4B9B
B6B7B9BBBDBDC1C1C2C3C3C4C5C4C7C9C8CBCDCDD0D2D1D2D2D0D1D4D3D6D7D8D7DBDCDDDDDDEDCDCDCDFE0E1E1E4E3E2E1DFE1E2E0DFE1E1E0E0E0E1E1DFDFDEDEDDDDDEDDD9D8DD9536333435323232323333333234
1322F39BFE1DDDCDBDAD8D7D4D4C22E16124FA9A6A4A3A1A09D9D999695948F8E8D8C8AAAABAFB0B0B4B5B8B9BCBFBFBFC1C2C2C3C5C4C7C6C8CBCCCED0CED0D4D2D2D2D3D4D6D6D6D7D8D9D9DBDDDEDEDEDFDFE1E5E2E1E
CAE4E0E0E3E3E2E0E1E3E3E1E1E2E2E1E1E1E1E3E5E3E3E1E1E2E2DEDEE6963D3732323436383A38373165E3C1A5E4DFDEDDDAD9D8D5D3CA39171335A0A8A6A4A5A4A09E9A9895939491918F8E8CADAFB1B3B6B6B9B8BDBF
EBCBEBFC2C4C2C4C4C6C86A202B50260D5BE6E2D8CED5D2D687252A2A2A2A271E130CAAE4DEDEDCDCDAD8AFE2E4E3E2E2E0D652284FC8E6E3E4E3E3E4E5E5E4E7E7E4E2DFD9E1E2E2E68435323337393A3B3E44C3E8A4CDB
827854576A70C8DDD7D5D87616171D6AADA4A5A7A9A5A1A09F9E9A9896959492928E8CAFAFB1B2B5B8BABABEBFC1C2C4C3CB8F423458551C702D16B6E0DFE1E1DCD7DD630E1211101010111010A8E3DEDDDDDCDAD9D4B6E5E
6BA98EAE5E4EAB63231323430322A7CE7E9EAEADCAB8E8FABCFE7E7DB8443393536373A46493FB9E8AC8E6119349AA79CD5D9D6D4511717216AAFADAAA8A6A8A6A29FA09B9B9B9797959594908FAFB1B1B3B6BBBBC0C2C0C1
C2C2C3BE2500020B12120E070C08070836DFEDEAEBEFECEAF461020A0A0B0B0C262B1DA5FBF4F1F2F3F2F3FB930C13151819303A363233322F2826363A332B2B3230313D49403D3A35373A3B393B3F4A50C9ECCB848380857
D928F849098D44F181B233162B2ADAFAFAAA7A4A4A5A4A0A1A19E9A999A9796939293AEB2B7B5B7B7BABEC1C4C9960201030A100D060706131B0707065CEAD0F0E3E9F0EFCC1F182C2D160C122B3032C4EFE8E7EBE8FB9711
F75F1F13162D33408EA9AB99978F8B93CFE6DCE1E4E5D1A87B462B24343C3B343332316BE9E9E8E7E7E4E4E4E3E17C1D1D1D29366EB8B0AFADAFADAAAAA7A3A2A2A0A0A09B9B98979795939392ACAFB2B4B5B7BBBFC1C3C9
3C3C4B73402030606060508131D1C0B0507090C107EF2DAE5F1E7E2F9AB1911120C0E130F2B3334B6F2F3693534242940307BDBF1F7F0EEEFF1F2C9D7DFDFDDE3DBDEE2C36B252625252629325A3BC8EBE8E7E5E2E8B7272
21221D1D2F55B6BAB7B7B6B3B4B5B4B0AEABA9A39F9E9EA09F9E9D9D989797959390B1B4B5B3B6B8BBC1C3C6C5C6CD813A323523201D151215120E0C09090A0C0569C9C4C6C7C7C9C4460A141312121111113130B0B634027
B212841352A95EAE7E3E4E6E4E9E6E7DDD3DEDEDBB6C5DEDDDDDF922D2E2D2E2F3149DAEAE7E7E7EA9D22231F203AA2C0BEBCBAB4B3B3B5B8B2B0AEAEA9A6A5A4A19E9D9B9A9C989797949293AFB2B2B2B8BABFC1C4D4DAD
E4DDDFDFDFDACFAE8F999EA0A49C2D0E11131314131414151329B7D7D6D6D6D6D86C12151513171415362728452B2560DBE4E1E2E2E1E3E5E6E8DCDDDFDEDFDFDEDEDEDEDF893437393F3E3153E1E9E8E7E4E254211F1C2E
F1936B0C0BFC0BCBDBFC0C1BBB7B6B5B1ADAAA9A7A5A39E9F9F9C9C9A9B98979491AFB0B2B5BABBBEBFDADEDDDFDEDDDCD9B5ACA7B2B3B4B5B9BDC1C5C5C6C7C9CACCCECFD1D2D0CCCBCAC4C0BEBEBFBEBDBEC45128303E27
30444A24252797E4DDDBDCE1E3E4DADCE1DDE0DDDEDFE0E28A53545257352E3134363D69DAE9E6EE9220221D238ECAC3C1BDBEBDBAB8B8B8B8B3B3AFABA8A6A5A6A2A29F9F9D9D9B98959491AEAFB6B7B8BCC0BFDCDDDDDE
EDEDEDEDEDED8B8AFB6B9BBBCBEC3C8C0B8B9BCBFBFC3C3C3C5C6C9D2D3D6D7D0CDCCCCCAC7C7C5D1662E372F2F30594126272120202121212A3F555348484C43312A2B2C2F3135473D2E2B2E3637392F43BBEA791F211D2D
21201F3297C7C0C1BFBABBBCBBBAB7B4B1AFACA9ABA9A6A3A2A29E9F9F9B989592ACAEB4B4B7B7BEA5CEDBD9D9DADBDAD6B5A7B0B0B1B2B68410030200000809000410140E120E0714AFF2EAEAECEFEAC43D170E07183241
0070A40858A5B35313232211D342A2D2E2E3032323435353635322F2D2A2829292827262625242425282A2928282424305AA3C0C0BEBDBABBB8B7B2ADAEADABAAA8A4A09F9FA09E99959895ADB1B5B7B9BABFA6A2CDCECBC
C0B5B1B6DCE5E7E9E9E8D531030C03090D0E0704040D0704030303040302060A059DF0EAEBEAECEDCD330207071C23225EB99E4A5D666C4D542509131118242D2E3131322E2C29151212162162403A5C1E0A182324272624
724222121201F1852C2C3C0BEBABAB5B3B3B1AFAEAEAAA7A5A29E9FA09D9A9796AFB2B2B6B7BABCBEC5BA908B9BD1B80D11332C1104070202061D0E010607060303050603040615120C0C0B0829D6F0EAEAEBECEDC122040
DE4F0A1E0D1A1305060C15060913100F0F0D0C0D12130C0F0A0C0D21727969586C73706E6B655D555C5736120F0C0B0A2076C6C5C3C1C0C0BCBCB8B6B7B2B1B2AFABABA4A3A3A19E9B9995ADAFB1B5B9BABDBFC2C4B0A6BB
FC7B54400060F09041415040606040D0D071C201D120A0A0A0C20261B211D130C0E19312677F9EFEDEEEEEDEEDC4E1521202113080607020E14070305010206040F1215132A75776076746F6D6C6A676665636159554 30E0
512A0200000081C7C5C7C8CDCDCBC6C0BEBFBEBAB5B5B1ACA9A6A39E9E9C9896AAAFB0B2B6B8C1DBDEDEDFCE4D000821242449778F5C887A887F89873B7B3D808A7959779975A1819D686C995C4E43553AC1FEF1EAE3E4E7
FE2E2E4E4E1E6E0DCDAD8D7D6D3D3D1D0CECBCAC6CDDCE2BE345D6D6A7472706C3148653A1E49112446665452D0DAE2D89EDADBD9DEDEDDDDDED8C9C9C5C3BEBAB3AEA8A3A39E98989693ABAAAEB2B4BCD6DEDBDEE168010
0F010D1314141516161719190F0505060707060C26291904070909081F292E393A361F303C382111192F3326494946463033331C122244180D0C0E12130D0808061B2F1C526269746521373E1E38563644526260645B4719
E2A0D0804004CD5DEDFE1DDD6D4D4D8DDDDDDDEDDDCDBDAD7D2CEC7C1B7A99EA9ABACADAFC2DADCDACC30030205040813141214121110060D15141713151616171816030304181B06040604151419252 40B07141B2723272
0302080A0807040202020203070A0A07070706060602020C0A36696268615D50556367656462605F676A4C22040404005FC9C6A1818BA5B8C6CFDADDDCDCDBD7CABDB7B4B0A9A2988F85A1A7ACAEAAA0BAC0BFAF7690C1A4
A93BAB93B5EC2C9A0736F6966696B6E717377797B7D82848487898E92929393969899999C9CA0A2A3A4A5A8ABADB0B1AFAFB1B5B45702060806030E1303060503070608093862615E6A6A6766666362615F695C4727030404
030403002BBBBFB9BAB2868A92A6B4BFC9C6BC7B8C9B97969494948F8E8D8AA0A1A8ABABA9B2B6B6B8B7B755569CBDAE2B113A4343413D3C393734302D2A2828272522202125201A1A1A191919191B1E1D1C1E2021262D3C4
DA1A9A9ADB0B2B7B7B1B2ADA9AFABA4A099912E1F201D1B1A19161616244A4B4E4E4C4B49413E1B080707060504014 2BCB4B0ACA9A7A997604051737162959393928E8D8F8C8B8987879EA5A5A8ACAEAEB3B5B5B9BAB47C9
BCB77EB4B5AFA4220001030303020405050406060608080908080A0B0C0C0B0B0C0E10130E0C0E1011131726405A6A58352115121 71A181614180C111A192F16140C0E0C0D0B0B0B0909070609060808070A0B0704040303
504020C8AB4AAA7A6A6A3A0A19F9D9F9B9B999693908E8D8A8788878382819C9FA3A5A7AAACAFAFAFB1B3B5B4BAA578767794AC9C9598A2A5A5A8ADB2B4B3A81C060B0B0D0E0F0F0F11131214131515181A191D20110C0D0
D3D1D3843A4142423F3B4DBFD0CFD383353A3937343039AECDCCCA75272A29272422258EBBB5B5642627282 72E97B0ABA5A2A29FA1A19F9F9C99989694938C8B8B898785848381807F979D9FA2A5A7A9A9ACADAEB1B0ADB0
9A9AAADAEAFB3B6B9BDBCBEBFBEC2C2C3C4C2C5C2C2C5C6C3C3C7C6C7C8C8C8C8C8C9C9C8C8C8C9CAC8C8C8C8C8CAC8C8C7C8C8C5C6C8C8C8C8C6C4C3C2C1BFBEBFBEBEBEBCB9BABAB7B6B7B4B1B0AFAEADABA8A9A5A3A2
```

Figure 8-17: Grayscale Raw Bytes Text file

Sobel Filter was chosen to be able to define the edges of the objects in the frame more clearly. It accepts a greyscale image and outputs a binary image with an image consisting of only two colors: white and black. This filter was compiled and simulated in modelsim (Figure 8-18). The raw bytes were tested before and after the sobel filter and indeed the output was a binary file, that represented a 1 for white and 0 for black (Figure 8-19).


ModelSim - INTEL FPGA STARTER EDITION 10.5b
File Edit View Compile Simulate Add Project Tools Layout Bookmarks Window Help
ColumnLayout AllColumns
Layout Simu
Project Library
Type here to search

Project - C:/Users/Glen Mould/Downloads/prim_code/hdl/prim sobel

Name	Status	Type	On	Modified
core_sobel.v	✓	Verilog	0	01/21/2015 09:43:33 ...
main_siobel.v	✓	Verilog	1	01/21/2015 09:43:21 ...
mem_sobel.v	✓	Verilog	2	01/21/2015 09:40:01 ...
testbench_sobel.v	✓	Verilog	3	04/24/2017 09:43:45 ...
Top.v	✓	Verilog	4	01/21/2015 09:38:08 ...

Objects

Name	Value	Kind	Mode
Bus_out	xxxx...	Net	Internal
Mem_bus_in	xxxx...	Net	Internal
[7]	StX	Net	Internal
[6]	StX	Net	Internal
[5]	StX	Net	Internal
[4]	StX	Net	Internal
[3]	StX	Net	Internal
[2]	StX	Net	Internal
[1]	StX	Net	Internal
[0]	StX	Net	Internal
Bus_in	xxxx...	Net	Internal
Mem_bus_out	xxxx...	Net	Internal
Clk	St1	Net	Internal
Reset	St1	Net	Internal
Data_strobe	St0	Net	Internal

Processes (Active)

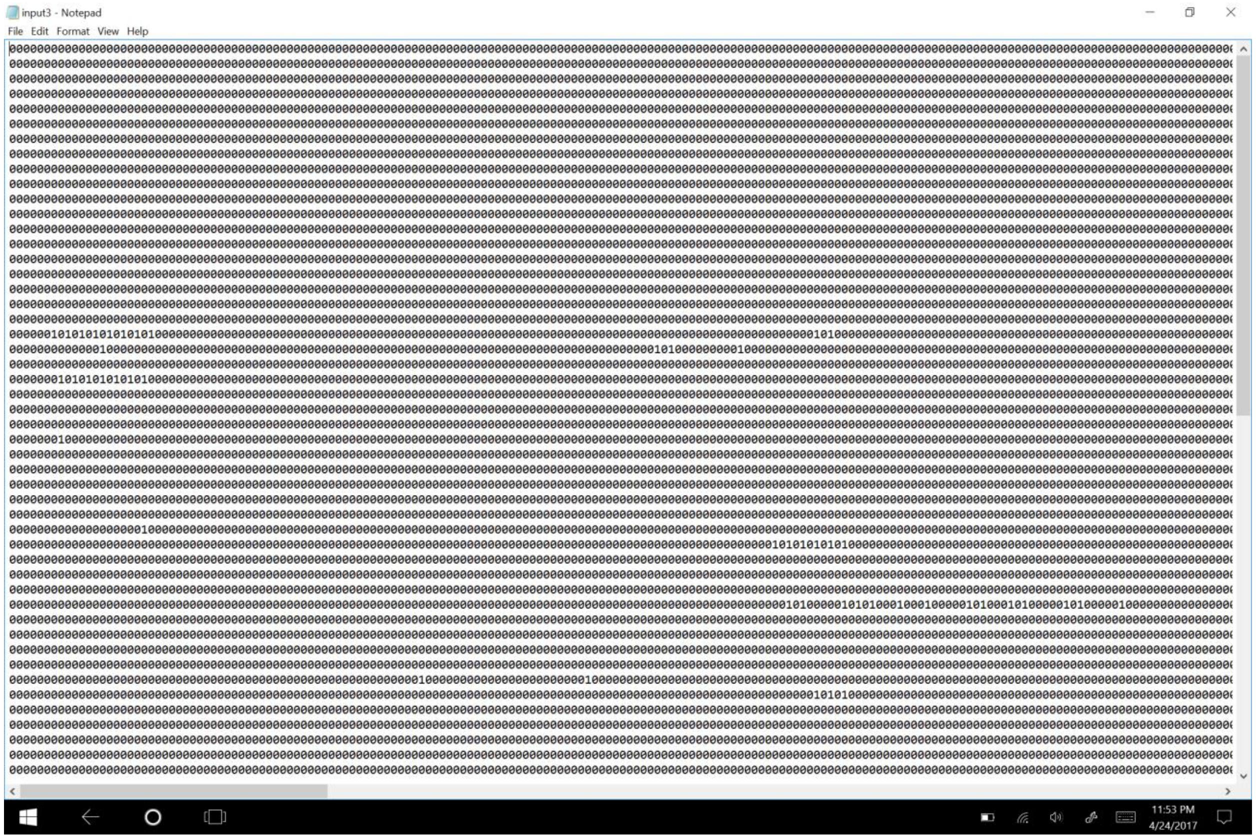
Name	Type (filtered)	State	Order	Parent Path

Transcript

```
# Loading work.main_sobel
# Loading work.core_sobel
VSIM 2> run -all
run -all
VSIM 3> run
run
VSIM 4> run

VSIM 4>
```

Project : prim sobel   Now: 8,748,494,715,300 ps Delta: 0   sim:/Top

Figure 8-18: ModelSim Compilation and Simulation of sobel filter



```
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000001010101010101010000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000010100000000000000000000000000000000000000000000000000000
0000000000000100000000000000000000000000000000000000000000000000000000000000000000000000000001010000000001000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000101010101010100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001010101010101000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000010100000101010001000100000101000101000001010000010000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000001000000000000000000000001000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000101010000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

Figure 8-19: Screenshot of raw bytes for sobel filtered image

Suggested Improvements / Changes

Our final system detects a specific type of Nerf gun. This is a significantly more limited detection than what we had set out to do. The main reason for this more limited scope was due to testing and experimenting on our machine learning algorithm. We did not have the time to get the thousands of extra pictures needed or spend the extra time every time we would retrain. Given the extra time and pictures, our system could theoretically be trained to detect a wider variety of weapons and also take into account the pose of the person with the gun, but due to our limited time, we were unable to train for these extra parameters.

Many small issues were encountered when assembling the gimbal. Most of these were caused by unfamiliarity with designing for 3D printing and could be easily avoided if the assembly were to be printed again. These issues do not include enough supports for some overhanging sections, larger parts being hard to divide to both fit the printer and be supported, and holes being printed too small for assembly. Several of the larger overhangs needed to have extra supports added before printing. This resulted in some supports being in the way of the moving parts and needing to be removed after printing. In addition, most of the larger parts were broken into smaller parts that were then glued back together after printing. This resulted in some loose or weak connections between parts which had to be glued back together several times. Most of the holes that were printed ended up being slightly too tight for the parts that they fit onto. Many of these holes were just drilled out, but some of the non-circular holes, such as the D shaped holes for the encoder gears, had to be melted larger in order to properly fit. This resulted in the encoders being unreliable at times due to the poor mounting of the gears.

UVC Driver

Due to time and hardware constraints in terms of the camera's interface being USB 2.0 the

HPS was used to be able to capture images. The camera can be interfaced through the FPGA. However, a USB driver was needed to be written for the board to recognize the device as a camera. The amount of research and development to be able to create such driver was not viable in our current time frame. The current implementation of the boards is running from a Linux SD card image that it mounted at boot up. This Linux version is made for embedded platforms. The UVC driver formally referred as USB Device Class definition for Video Devices(ref) is what defines video streaming functionality for the USB. With this driver, the Linux kernel is able recognize the USB device as a camera. This then can be used in different webcam software to be able to capture video or images needed for the detection algorithm. The software that was used to capture the frames of an incoming video was Fswebcam. This software was used because it does not require many extra supporting libraries and it can be controlled using the command line. The figure shown below shows the image capture from the command line.

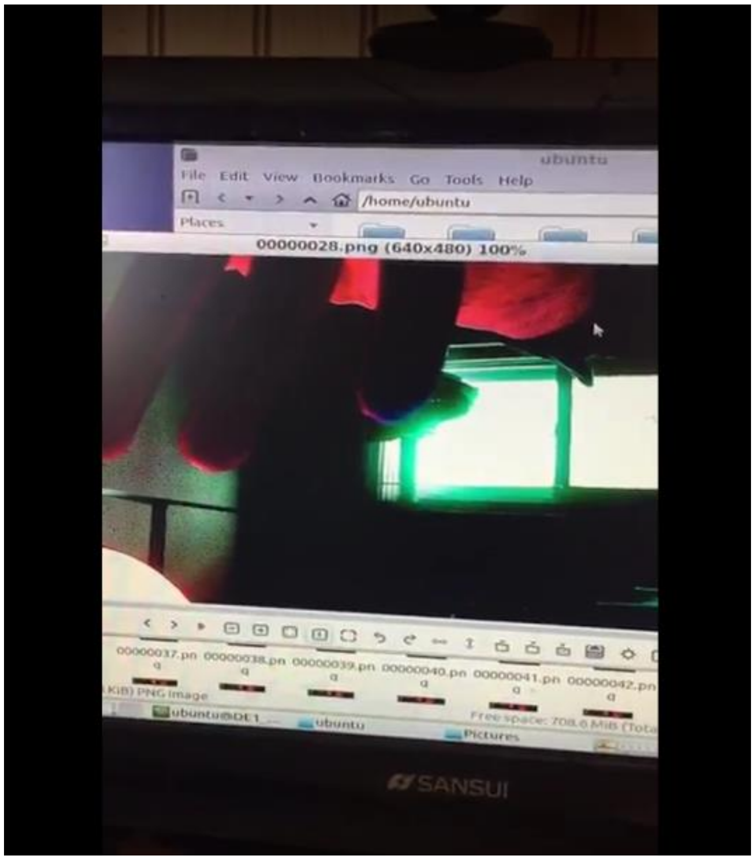

ubuntu
File Edit View Bookmarks Go Tools Help
/home/ubuntu
Places
00000028.png (640x480) 100%
00000037.png 00000038.png 00000039.png 00000040.png 00000041.png 00000042.png
PNG image
ubuntu@DE1
ubuntu
Pictures
SANSUI

Figure 9-1 Output Frame image

There were some initial setbacks while trying to obtain this driver since some sd card images provided by Altera don't have this driver installed, and required re-compilation to add the driver and get it to function properly. However, resources provided by Altera via email allowed for a partial remedy to this issue.

Board Output Voltage

The board we are using features two 40 pin I/O connector ports. 8 pins on each port are predesignated for use, leaving 32 configurable input and output pins on each port for us to use. The intent, was to use these ports as the input and output needed to interface with the motors, servos, and sensors of the gimbal. This worked fine for the sensors, but ran into issues for the motors. The plan was to have the DC motors connected to two pins. Have both pins set to 0 to keep the motor still and set one or the other to 1 in order to drive the motor in varying directions. The pins supposedly are connected to the 3.3V rail of the board, meaning that the motor would run on 3.3V when driving, enough to properly drive the gimbal. However, when we went to actually try this, the motor did not receive enough power to actually drive the gimbal. After testing many possibilities, we determined that it was due to the pins not actually outputting 0 and 3.3V when set to 0 and 1, but actually 0 and .56V instead while the motor was connected. After more research, we determined that the only feasible solution was to build voltage amplifiers that pulled directly from the 3.3V rail. Unfortunately, when we went to build the voltage amplifiers, we discovered that not only did we not have any working OpAmps, but we also did not have the proper capacitors to build a voltage amplifier from transistors. We also did not have enough time for the parts to come in in order to properly fix this problem.

Conclusion

The design and implementation of this project, though difficult and not without issues, shows significant amounts of potential in terms of the application of such designs going forward. That such high demand processes such as those required for computer vision analysis can be ported to run on ARM embedded based architectures alone speaks greatly towards the potential for embedded vision applications in everyday life. In addition to this, the Cyclone V's SoC based architecture, with its integration of programmable FPGA architecture, and with all of the flexibility and process re-distribution that such architecture allows, only further adds credence to the feasibility of such designs becoming more and more widespread.

The architecture and resources we were designing on ultimately were cheap, mostly meant for educational purposes and for far more simplistic demos than what is attempted through our project implementation. That we could create a prototype that came so close to meeting our goals implies that further development with more professional level tools may soon lead to systems of far greater practicality than our own entering the market and potentially addressing the very societally relevant issue of armed robbery and assault. Some of these systems may even be in development at this current time.

There was a lot to learn and a lot to discover in regards to seeing this project come to fruition. The team spent many long hours poring over technical documents from Altera, Google, Cornell, Terasic, Mathworks, ARM, Unix, and many more in an effort to fully understand and utilize what hardware we'd need and then how far we could push that hardware and use it to our advantage programmatically. Whether it was filter design optimization with Simulink, to kernel recompilation for ARM architectures with Linux, Neural Network creation and calibration with Tensorflow, or hardware driver reallocation and creation with Verilog and Quartus, these were all things that were not innately known to the team that have now become second nature and whose sum creates a system level design toolbox that can likely solve not only the problem we set out to address of violent gun based crimes but countless other issues and hardships addressing society.

Though there are many things in regards to this project that the team would like to revisit or see revisited, the work that was done and the knowledge that was gained throughout the project, all of which is dually reflected and stated in this report, has satisfied our team adequately. The accuracy and reliability of our system came very close to our intended benchmarks, and we now feel we have solid evidence that systems like ours may soon start contributing to lowering gun crime and swifter police response time. We look forward to carrying this knowledge on into our future endeavors and we hope that through our design process that we have indeed helped come ever closer to ensuring the safety, security, and comfort of everyday citizens at both the public and private level.

Glossary

AGH -- AGH University of Science and Technology in Krakow, Poland

FPGA -- Field Programmable Gate Array

CCTV -- Closed Circuit Television

CV -- Computer Vision

IVS -- Intelligent Video Surveillance

ML -- Machine Learning

MLL -- OpenCV Machine Learning Library

RPM -- Rotations per Minute

Sensitivity - How good our system is able to detect a firearm when a firearm is present (True Positive)

Specificity - How good our system would be at not detecting a firearm when a firearm is not present

(True negative)

References

[1] H. M. Dee and S. A. Velastin. How close are we to solving the problem of automated visual surveillance?: A review of real-world surveillance, scientific progress and evaluative mechanisms. Machine Vision and Applications 19(5), pp. 329-343. 2008.

[2] P. L. Venetianer and H. Deng. Performance evaluation of an intelligent video surveillance system – A case study. Computer Vision and Image Understanding 114(11), pp. 1292-1302. 2010.

[3] T. Kim and R. Cipolla, "Multiple Classifier Boosting and Tree-Structured Classifiers", in Machine Learning for Computer Vision (2013th ed.).

[4] B. Lee, E. Erdenee, S. Jin and P. K. Rhee, Efficient object detection using convolutional neural network-based hierarchical feature modeling. Signal, Image and Video Processing 10(8), pp. 1503-1510. 2016.

[5] D. Fleet, T. Pajdla, B. Schiele, T. Tuytelaars, "Edge Boxes: Locating Object Proposals from Edges " in Computer Vision - ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V.

[6] R. Girshick. Fast R-CNN. 2015.

[7] N. Razavi, "An introduction to random forests for multi-class object detection", in Outdoor and Large-Scale Real-World Scene Analysis: 15th International Workshop on Theoretical Foundations of Computer Vision, Dagstuhl Castle, Germany, June 26-July 1, 2011.

[8] M. Grega, A. Matiolanski, P. Guzik and M. Leszczuk. Automated detection of firearms and knives

in a CCTV image. Sensors 16(1), pp. 47. 2016

[9] P. L. Venetianer and H. Deng. Performance evaluation of an intelligent video surveillance system – A case study. Computer Vision and Image Understanding 114(11), pp. 1292-1302. 2010.

[10] M. Grega, A. Matiolanski, P. Guzik and M. Leszczuk. Automated detection of firearms and knives in a CCTV image. Sensors 16(1), pp. 47. 2016.

[11] R. Appleby, P. R. Coward and G. N. Sinclair. "Terahertz detection of illegal objects," in *SpringerLink*Anonymous Available: http://link.springer.com/chapter/10.1007/978-1-4020-6503-3_15.

[12]M. Grega, A. Matiolanski, P. Guzik and M. Leszczuk. Automated detection of firearms and knives in a CCTV image. Sensors 16(1), pp. 47. 2016.

[13] Automated Visual Recognition of Armed Robbery. Orlando. Available: http://crcv.ucf.edu/papers/robbery.pdf. [Online]. [Accessed: 05 -Nov -2016]

[14] J. Jacobson. (). *How Do DIY, MIY Security Systems Affect Police Response to Alarm Activation?*. Available: http://www.securitysales.com/article/how_do_diy_miy_security_systems_affect_police_response_to_alarm_activation.

[15] F. Sherman and F. Sherman. (). *How Do Silent Bank Alarms Work?*. Available: http://www.ehow.com/how-does_4564481_silent-bank-alarms-work.html.

[16] G. V. Archive, "Mass shootings - 2015," in GunViolencearchive.org, 2015. [Online]. Available: http://www.gunviolencearchive.org/reports/mass-shootings/2015. Accessed: Nov. 11, 2016.

[17] FBI, "Reports and publications," in FBI, Federal Bureau of Investigation, 2016. [Online]. Available: https://www.fbi.gov/stats-services/publications. Accessed: Nov. 20, 2016.

[18]T. Dempsey, "Compare digital camera sensor sizes: 1″-Type, 4/3, APS-C, full frame 35mm," 2013. [Online]. Available: http://photoseek.com/2013/compare-digital-camera-sensor-sizes-full-frame-35mm-aps-c-micro-four-thirds-1-inch-type/. Accessed: Nov. 16, 2016.

[19] john, "Working of digital camera-block diagram, parameters, color filtering," in Camera Technology, Electronic Circuits and Diagram-Electronics Projects and Design, 2010. [Online]. Available: http://www.circuitstoday.com/working-of-digital-cameras. Accessed: Nov. 3, 2016.

[20] "Infographic: The difference between CMOS and CCD sensors," in Educational, PetaPixel, 2016. [Online]. Available: http://petapixel.com/2016/05/10/infographic-difference-cmos-ccd-sensors/. Accessed: Nov. 4, 2016.

[21] "Camera Sensors," in sparkfun. [Online]. Available: https://cdn.sparkfun.com/datasheets/Sensors/LightImaging/OV5640_datasheet.pdf. Accessed: Dec. 20, 2016.

[22] http://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm

[23] http://www.aravind.ca/cs788h_Final_Project/gradient_estimators.htm

[24] Alavianmehr, Mohammad Ali, Ashkan Tashk, and Amir Sodagaran. "Video Foreground Detection Based on Adaptive Mixture Gaussian Model for Video Surveillance Systems." *Journal of Traffic and Logistics Engineering* 3.1 (2015): n. pag. Web

[25] Raviraj Singh Shekhawat, Software Developer at CEERI, Pilani Follow. "Background Subtraction." *LinkedIn SlideShare*. N.p., 09 Dec. 2011. Web. 05 Mar. 2017.

[26] Stauffer, C., and W.e.l. Grimson. "Adaptive Background Mixture Models for Real-time Tracking." *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)* (n.d.): n. pag. *Adaptive Background Mixture Models for Real-time Tracking*. IEE.

Web.

[27] Raviraj Singh Shekhawat, ;Software Developer at CEERI, Pilani Follow. "Background Subtraction." *LinkedIn SlideShare*. N.p., 09 Dec. 2011. Web. 05 Mar. 2017

[28] Tensorflow, *Image Recognition,* Available: www.tensorflow.org/tutorials/image_recognition

[29] Adam Gitney, *Deep Learning and Convolutional Neural Networks,* Available: https://medium.com/@ageitgey/machine-learning-is-fun-part-3-deep-learning-and-convolutional-neural-networks-f40359318721

[30]"Edge Detection - MATLAB & Simulink". *Mathworks.com*. N.p., 2017. Web. 27 Apr. 2017. Matlab Sobel Filter model: https://www.mathworks.com/discovery/edge-detection.html

[31] JPEG File Interchange Format: Summary From The Encyclopedia Of Graphics File Formats". *Fileformat.info*. N.p., 2017. Web. 27 Apr. 2017. Available: http://www.fileformat.info/format/jpeg/egff.htm

[32] http://stackoverflow.com/questions/23852907/c-get-rgb-from-byte-array

[33] "Arch Linux ARM". *Archlinuxarm.org*. N.p., 2017. Web. 27 Apr. 2017. Available: https://archlinuxarm.org/

[34] Krig, Scott. *Computer vision metrics*. California: Apress Media, 2014. pp79

[35] H. R. Roth et al. DeepOrgan: Multi-level deep convolutional networks for automated pancreas segmentation. 2015.

Appendix - A: Random Forest for Object Detection

A random forest is a collection of T binary decision trees, where each node of the tree can have up to two leaves as children. The random forest takes an image patch as input, evaluates it at each node (starting from the root node) and sends it to the left or right child of the node accordingly. The forest can be used for either a classification or a regression task. For a classification task, each leaf L of a forest returns a probability for the image patch to be classified as c given as p(c|L). For a regression task, the forest returns an estimate, as a distribution, for the location and scale of the object. [7]

During training, each tree is trained on a random subset of the training dataset. This helps avoid overfitting and efficiently processing large amounts of data. For each sampled image "For each sampled patch P_I that does not belong to the background, the offset to a reference point of the object d_I is stored" [7]. The training set therefore consists of patches P_I and their corresponding extracted image features F_I, their class label c_I and the offset d_I from the center of the patch. A recommended size for each image patch is 16x16 pixels" [7].

A split function is defined as $f_{\Phi}(P)$ that returns 0 or 1, given features Φ and image patch P, where 0 corresponds to the left child and 1 to the right child. The split function gets updated to reflect the training set "while the tree grows recursively:

1. Generate a random set of parameters Φ = {φk}.
2. Divide the set of patches A_{node} into two subset A_L and A_R for each φ ∈ Φ:
   - A_L (φ) = {P ∈ $A_{node} | f_{\Phi}(P) = 0$}
   - A_R (φ) = {P ∈ $A_{node} | f_{\Phi}(P) = 1$}
3. Select the split parameters φ∗ that maximize a gain function g:

$$\varphi^* = argmax\ g\ (\varphi, Anode)$$

$$where\ g\ (\varphi, A_{node}) = H\ (A_{node}) - \sum_{S \in \{L,R\}} \frac{|A_S\ (\varphi)|}{|A_{node}|}\ H\ (A_S\ (\varphi)).$$

$and\ \varphi \in \Phi$

Depending on the task, H(A) is chosen such that g measures the gain of the classification or regression performance of the children in comparison to the current node.

4. Continue growing with the training subsets A_L and A_R if some predefined stopping criteria are not satisfied; otherwise, create a leaf node and store the statistics of the training data A_{node}" [7].

The resulting class probability is given as:

$$p(L) = \frac{|A_c^L| * r_c}{\sum_c \quad (\,|A_c^L| * r_c)};\ r_c = \frac{|A|}{|A_c|}\ [7]$$

Where $|A_c^L|$ is the set of patches that reach leaf L with a class label c. Finally, by using the offset d of the samples one can calculate the spatial distribution by estimating $p(c, L)$. [7] The class probability is used for classification tasks and the spatial distribution to estimate the location for regression tasks.

Random forests are able to detect objects by creating a distribution of $p(L_t(y))$, where $h(c, x, s)$ is the probability of an object of class c, with size s to be located at reference point x, for the leaf L of tree t at image location y. The distribution is defined by having each tree vote on the classification of the incoming image patch. The votes of all trees for all image patches are averaged and the resulting value determines the classification of the processed image and therefore the appropriate threat detection.

As far as the implementation is concerned, low level features such as "color, gradients, or Gabor filters" can be used because they are computationally efficient. In order to avoid overfitting one can force a stopping criteria on the depth of the trees [7]. Finally, several options for the split function as well as the estimation of the spatial distribution will be investigated to find the most effective for firearm detection.

Appendix - B: Multi-classifier Boosting

Cambridge professors Tae-Kyun Kim and Roberto Cipolla propose a multi-classifier boosting algorithm called MCBoost, which attempts to efficiently and accurately detect multiple features of the processed image simultaneously. Boosting algorithms combine several weak-learners, i.e. relatively poor predictors of an event, to create strong learners, which are more closely correlated with the classification of an event. A training set consisting of positive (visible firearm) and negative (no visible threat) images is used to calculate the weights of multiple weak-learners whose combination results in strong learners. During training, the weak-learners that are observed to predict their strong learners (visible firearm, human figure, trigger detected) are given more weight and thus strong learners become more accurate as the training set grows. Finally, the strong learners can be used to classify events as threat in the processed images. [3]

The MCBoost algorithm considers "K strong classifiers, each of which is represented by a linear combination of weak-learners as

$$H_k(x) = \sum_t a_{kt} h_{kt}(x), \qquad k = 1, \ldots K$$

where akt and hkt are the weight and the score of *t-th* weak-learner of *k-th* strong classifier". Each weak-learner is a predictor of a single visual feature while strong classifiers are "devoted to a subset of input patterns" [3]. To aggregate multiple strong classifiers, a Noisy-OR is formulated as

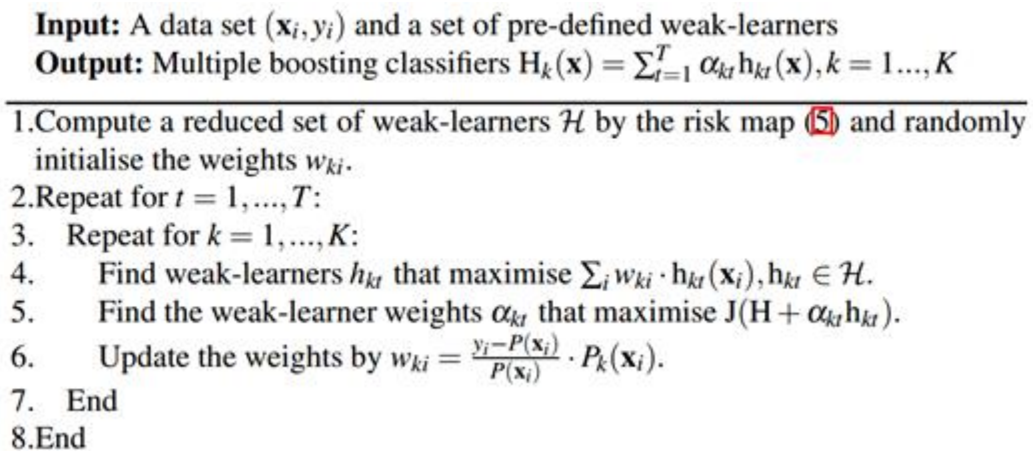
$$P(x) = 1 - \prod_k \left(1 - P_k(x)\right)$$

where P(x) = 1 / (1 + exp(-Hk(x)). The Noisy-OR framework calculates the "joint probability using all *k* classifiers for any x" [3]. If at least one classifier detects a threat, the image is flagged as positive. As a result, classifiers get trained to negatively flag all non-object images, which ensures low false-positive rates. Initially, sample weights wki are assigned for the *i-th* sample and the *k-th* classifier as either wki = 1 if it was detected in a positive sample (visible firearm) or wki = 1 / k if it was detected in a negative sample (non-object image) and wki = 0 otherwise. The goal is to maximize

$$\sum_t w_{kt} h_{kt}(x_i), \qquad h_{kt} \in \boldsymbol{H}$$

where xi is the *i-th* image sample and hkt ∈ {−1, +1} are the weak-learners from the set of all available weak-learners ***H.*** To optimize running speed, ***H*** can be a subset of the available weak-learners containing only the weak-learners located around the expected decision boundary (firearms of known max size can be assigned with decision boundaries). [3]

After the initialization of the weights, each new round updates the sample weights in an attempt to increase the overall accuracy of the classification algorithm and to minimize the risk of misclassifications. The weights are updated, following the AnyBoost method, by taking the derivative of a cost function defined for **J** with respect to the classifier score. The MCBoost algorithm is described in pseudocode in the figure below:



Input: A data set $(\mathbf{x}_i, y_i)$ and a set of pre-defined weak-learners
Output: Multiple boosting classifiers $\mathrm{H}_k(\mathbf{x}) = \sum_{t=1}^{T} \alpha_{kt} \mathrm{h}_{kt}(\mathbf{x}), k = 1\ldots, K$

```
1.Compute a reduced set of weak-learners $\mathcal{H}$ by the risk map (5) and randomly
  initialise the weights $w_{ki}$.
2.Repeat for $t = 1, \ldots, T$:
3.   Repeat for $k = 1, \ldots, K$:
4.      Find weak-learners $h_{kt}$ that maximise $\sum_i w_{ki} \cdot \mathrm{h}_{kt}(\mathbf{x}_i), \mathrm{h}_{kt} \in \mathcal{H}$.
5.      Find the weak-learner weights $\alpha_{kt}$ that maximise $\mathrm{J}(\mathrm{H} + \alpha_{kt}\mathrm{h}_{kt})$.
6.      Update the weights by $w_{ki} = \frac{y_i - P(\mathbf{x}_i)}{P(\mathbf{x}_i)} \cdot P_k(\mathbf{x}_i)$.
7.   End
8.End
```

Figure 4: Pseudocode for the MCBoost algorithm [3].

The outcome of the MCBoost algorithm is multiple boosting classifiers that can detect threats in processed images. These classifiers can also be used to create a decision tree that reduces even further the classification time. [3]

Appendix - C: Computer Vision: Training Script

```
#### Author: Georgios Karapanagos ####
# Training script for 2-node classification network
#
# For references see below:
"""
Based on the tflearn example located here:
https://github.com/tflearn/tflearn/blob/master/examples/images/convnet_cifar10.py

Loosely following Adam Geitgey's suggested network architucre, found at:
https://medium.com/@ageitgey/machine-learning-is-fun-part-3-deep-learning-and-
convolutional-neural-networks-f40359318721
"""

# # -*- coding: utf-8 -*-
from __future__ import division, print_function, absolute_import
import tensorflow as tf
# Import tflearn and some helpers
from tflearnmaster.tflearn.models import dnn
from tflearnmaster.tflearn.data_utils import shuffle, to_categorical
from tflearnmaster.tflearn.layers.core import input_data, dropout, fully_connected
from tflearnmaster.tflearn.layers.conv import conv_2d, max_pool_2d
from tflearnmaster.tflearn.layers.estimator import regression
from tflearnmaster.tflearn.data_preprocessing import ImagePreprocessing
from tflearnmaster.tflearn.data_augmentation import ImageAugmentation
import util_helpers
import time

start_time = time.time()

# Load the data set
trainSize = 1775
testSize = 443
size = 32
(X, Y), (X_test, Y_test) = util_helpers.prepMat(trainSize, testSize, 80 ,45)

# Convert to 2 classes boolean array
Y = to_categorical(Y, 2)
Y_test = to_categorical(Y_test, 2)

# Shuffle the data
```

```
X, Y = shuffle(X, Y)

# Make sure the data is normalized
img_prep = ImagePreprocessing()
img_prep.add_featurewise_zero_center()
img_prep.add_featurewise_stdnorm()

# Populate the dataset by flipping, rotating and blurring
img_aug = ImageAugmentation()
img_aug.add_random_flip_leftright()
img_aug.add_random_rotation(max_angle=25.)
img_aug.add_random_blur(sigma_max=3.)

# Definition of our Network Architecture
# Input is a 32x32 image with 3 color channels (red, green and blue)
network = input_data(shape=[None, 80, 45, 3],
                     data_preprocessing=img_prep,
                     data_augmentation=img_aug,
                     name="input_node")
network = conv_2d(network, 32, 3, activation='relu')
network = max_pool_2d(network, 2)
network = conv_2d(network, 64, 3, activation='relu')
network = conv_2d(network, 64, 3, activation='relu')
network = max_pool_2d(network, 2)

# Fully-connected 512 node neural network
network = fully_connected(network, 512, activation='relu')

# Drop a percentage to reduce over-fitting
network = dropout(network, 0.15)

# Final 2-node network that predicts output of '0' or '1' for two classes
network = fully_connected(network, 2, activation='softmax', name="out")

# Set parameters for training method
network = regression(network, optimizer='adam',
                     loss='categorical_crossentropy',
                     learning_rate=0.001)

# Initialize DNN model
model = dnn.DNN(network, tensorboard_verbose=0, checkpoint_path='newDemo_checkpoint')

train_time = time.time()
```

```

# Train the model, by fitting data for n_epoch steps
model.fit(X, Y, n_epoch=50, shuffle=True, validation_set=(X_test, Y_test),
        show_metric=True, batch_size=96,
        snapshot_epoch=True,
         run_id='det_cnn')

# Export trained model to file
model.save("newDemo_checkpoint")

util_helpers.save_graph(model.session,"output","newDemo_checkpoint","checkpoint_state","input_graph.pb","output_graph.pb")


print("--- %s seconds training ---" % (time.time() - train_time))
print("--- %s seconds total ---" % (time.time() - start_time))
```

Appendix - D: Computer Vision: Testing Script

```
#### Author: Georgios Karapanagos ####
# Testing script for 2-node classification network
#
# For references see below:
"""
Based on the tflearn example located here:
https://github.com/tflearn/tflearn/blob/master/examples/images/convnet_cifar10.py

Loosely following Adam Geitgey's suggested network architucre, found at:
https://medium.com/@ageitgey/machine-learning-is-fun-part-3-deep-learning-and-
convolutional-neural-networks-f40359318721
"""

# -*- coding: utf-8 -*-
from __future__ import division, print_function, absolute_import
import tensorflow as tf
# Import tflearn and some helpers
from tflearnmaster.tflearn.models import dnn
from tflearnmaster.tflearn.layers.core import input_data, dropout, fully_connected
from tflearnmaster.tflearn.layers.conv import conv_2d, max_pool_2d
from tflearnmaster.tflearn.layers.estimator import regression
from tflearnmaster.tflearn.data_preprocessing import ImagePreprocessing
from tflearnmaster.tflearn.data_augmentation import ImageAugmentation
from scipy.misc import toimage, imresize, imsave
import numpy as np
from PIL import Image
import util_helpers
from heapq import heappush, heappop

# Node size
size = 32
sizeH = 45
sizeW = 80

# Make sure the data is normalized
img_prep = ImagePreprocessing()
img_prep.add_featurewise_zero_center()
img_prep.add_featurewise_stdnorm()

# Populate the dataset by flipping, rotating and blurring
img_aug = ImageAugmentation()
```

```
img_aug.add_random_flip_leftright()
img_aug.add_random_rotation(max_angle=25.)
img_aug.add_random_blur(sigma_max=3.)

# Definition of our Network Architecture
# Input is a 32x32 image with 3 color channels (red, green and blue)
network = input_data(shape=[None, 80, 45, 3],
                     data_preprocessing=img_prep,
                     data_augmentation=img_aug,
                     name="input_node")
network = conv_2d(network, 32, 3, activation='relu')
network = max_pool_2d(network, 2)
network = conv_2d(network, 64, 3, activation='relu')
network = conv_2d(network, 64, 3, activation='relu')
network = max_pool_2d(network, 2)

# Fully-connected 512 node neural network
network = fully_connected(network, 512, activation='relu')

# Drop a percentage to reduce over-fitting
network = dropout(network, 0.15)

# Final 2-node network that predicts output of '0' or '1' for two classes
network = fully_connected(network, 2, activation='softmax', name="out")

# Set parameters for training method
network = regression(network, optimizer='adam',
                     loss='categorical_crossentropy',
                     learning_rate=0.001)
#sess.run(init_op)
# Initialize DNN model
model = dnn.DNN(network, tensorboard_verbose=0, checkpoint_path='high_checkpoint')

# Load classifier
model.load("high_checkpoint-3600")

negSize = 1403
posSize = 1204
# negSize = 1403
# posSize = 1204
testSize = (posSize+negSize)+1
X_train = np.empty((testSize,sizeW,sizeH,3))

```

```
# Sliding window with test image to generate 10 images to test
imgs = []
for k in range(0,negSize):
    imgs += util_helpers.slidingWindowDemo("testHighBig/0/file" + str(k) + ".jpg",sizeH, sizeW)

for k in range(0,posSize):
    imgs += util_helpers.slidingWindowDemo("testHighBig/1/file" + str(k) + ".jpg",sizeH, sizeW)

trIndex = 0
foundInd = 0
totalProb = 0
threatFound = 0

# Test all images for threat
foundAt = []
maxNeg = []
for img in imgs:
    for x in range(0,sizeW):
        for y in range(0,sizeH):
        for color in range(0,3):
            X_train[trIndex][x][y][color] = float(img[x][y][color] / 255)

    # Predict
    prediction = model.predict([X_train[trIndex]])
    print("PREDICTION: " + str(prediction[0]))

    # Keep statistics of overall threat
    totalProb += prediction[0][1]
    #print("Found with " + str(prediction[0][1]) + " confidence.")
    # if threat is detected, prob(threat) > prob(nothreat)
    if(prediction[0][1] > prediction[0][0]):
        temp = toimage(img)
        if((trIndex / 5) < negSize ):
        heappush(maxNeg, (prediction[0][1]))

        # if prob(threat) >= 0.75
        if(prediction[0][1] >= 0.75):
        imsave("test-dataset/FOUND/EZ" + str(trIndex) + ".jpg", temp)
        threatFound += 2
        else:
        imsave("test-dataset/FOUND/" + str(trIndex) + ".jpg", temp)
        threatFound += 1
    trIndex += 1
    if(trIndex % 5 == 0):
        foundAt.append([trIndex, threatFound])
        threatFound = 0


z = 0
foundTrue = 0
foundFalse = 0
for k in foundAt:
    print(k)
    if(z<negSize):
        if(k[1] >= 2):
        foundFalse +=1
    else:
        if(k[1] >= 2):
        foundTrue += 1
    z += 1



print("-----------Printing results of Negative dataset---------------")
print("Found " + str(negSize-foundFalse) + " true negatives!")
print("Found " + str(foundFalse) + " false positives!")
print("Classified " + str(100*foundFalse/negSize) + "%% incorrectly!")

print("-----------Printing results of Positive dataset---------------")
print("Found " + str(foundTrue) + " true positives!")
print("Found " + str(posSize - foundTrue) + " false negatives!")
print("Classified " + str(100*(posSize - foundTrue)/posSize) + "%% incorrectly!")

print("Total Accuracy: " + str((negSize-foundFalse+posSize - foundTrue)/(negSize+posSize)) + " % ")
print("Found " + str(foundInd) + " matches")

# Print the top 20 false positives value
for k in range(0,20):
    V = heappop(maxNeg)
    print(V)
```

Appendix - E: Computer Vision: Helper Functions

```python
#### Author: Georgios Karapanagos ####
# Helper functions to prepare training and testing datasets
####

import glob, os
import numpy as np
import time
from array import *
from PIL import Image
from scipy.misc import toimage, imresize, imread, imsave
import tensorflow as tf
from tensorflow.python.tools import freeze_graph
import export
import cv2
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import scipy
from scipy import ndimage
import camera

# Renames all files in reg that match the type as a RegEx to file0.jpg - filen.jpg
# - example call: renameFile(r'C:/Users/George/Documents/MQP/Nerf-Dataset', r'*.jpeg')
def renameFile(dir, typeRegEx):
        i = 0
        for imagePath in glob.iglob(os.path.join(dir, typeRegEx)):
        title, ext = os.path.splitext(os.path.basename(imagePath))
        os.rename(imagePath, os.path.join(dir + "/file" + str(i)+ ".jpg"))
        i = i + 1
        # wait to avoid memory-write problem
        time.sleep(0.01)

# Generates a resized to height*width version of the given image database
# - example call: genResized("Nerf-Dataset",  ".jpeg", 34, "small", size)
def genResized(dir, type, imageNum, desc, height, width):
        for i in range(imageNum):
        # Load the image file
        img = imread(dir + desc + '/file' + str(i) + type, mode="RGB")

        # Scale it to 32x32
        img = imresize(img, (height, width), interp="bicubic").astype(np.float32, casting='unsafe')

        #Save the image to new folder
        imsave(dir + desc + '/file' + str(i) + '.jpg', img)


# Custom version of genResized with rotation and 1024x720 output
# - example call: genResizedTest("C:/Users/George/Documents/MQP/test-dataset",  ".jpeg", 34)
def genResizedTest(dir, type, imageNum):
        for i in range(imageNum):
        if(i >= 0):
                # Load the image file
                im = Image.open(dir + '/file' + str(i) + type)
                pix = im.load()

                # Scale it to 32x32
                #pix2 = (im.rotate(270)).resize([720,1024])
                pix2 = im.resize([720,1024])

                #Save the image to new folder
                pix2.save(dir + '/file' + str(i) + '.jpeg')

# Prepares training matrix of size*size input for the CNN
# - example call: prepMat(3279, 364, 32)
def prepMat(trainSize, testSize, sizeW, sizeH):
        i = 0
        X_train = np.empty((trainSize, sizeW, sizeH, 3))
        Y_train = array('B')
        X_test = np.empty((testSize, sizeW, sizeH, 3))
        Y_test = array('B')
        trainNum = 0
        testNum = 0
        # finds and loads all items in "classes" in the "train-dataset" folder that end with .jpg
        for dirname, dirnames, filenames in os.walk('./newDemo'):
        for filename in filenames:
                if filename.endswith('.jpg'):
                i = i + 1

                # Load the image file
                im = Image.open(os.path.join(dirname, filename))

                # Load image into matrix
                pix = im.load()

                # Get class label from folder name
                # e.g. /Classes/0/xx.jpg -> 0
                class_name = int(os.path.join(dirname).split('\\')[-1])
                w, h = im.size
                  # 9/10ths are the training set
                if(i%10 > 1):
                        # Append the class label as Byte
                        Y_train.append(class_name)
                        for x in range(0,sizeW):
                        for y in range(0,sizeH):
                                for color in range(0,3):
                                        X_train[trainNum][x][y][color] = float(pix[x,y][color] / 255)
                        trainNum+=1
                # 1/10th is the validation (or test) set
                else:
                        # Append the class label as Byte
                        Y_test.append(class_name)
                        for x in range(0,sizeW):
                        for y in range(0,sizeH):
                                for color in range(0,3):
                                        X_test[testNum][x][y][color] = float(pix[x,y][color] / 255)
                        testNum+=1

        print("Height", h)
        print("Width", w)
        print(i)
        print("test", testNum)
        print("train", trainNum)
        return (X_train, Y_train), (X_test, Y_test)

# GREYSCALE , 1-CHANNEL VERSION
def prepMatGrey(trainSize, testSize, sizeW, sizeH):
        i = 0
        X_train = np.empty((trainSize, sizeW, sizeH, 1))
        Y_train = array('B')
        X_test = np.empty((testSize, sizeW, sizeH, 1))
        Y_test = array('B')
        trainNum = 0
        testNum = 0
        # finds and loads all items in "classes" in the "train-dataset" folder that end with .jpg
        for dirname, dirnames, filenames in os.walk('./statTrain64'):
        for filename in filenames:
                if filename.endswith('.jpg'):
                i = i + 1

                        # Load the image file
                im = Image.open(os.path.join(dirname, filename))

                # Load image into matrix
                pix = im.load()

                # Get class label from folder name
                # e.g. /Classes/0/xx.jpg -> 0
                class_name = int(os.path.join(dirname).split('\\')[-1])
                w, h = im.size
                tempColor = 0
                  # 9/10ths are the training set
                if(i%10 > 1):
                        # Append the class label as Byte
                        Y_train.append(class_name)
                        for x in range(0,sizeW):
                        for y in range(0,sizeH):
                                #for color in range(0,3):
                                tempColor += 0.21 * float(pix[x,y][0] / 255)
                                tempColor += 0.72 * float(pix[x,y][1] / 255)
                                tempColor += 0.07 * float(pix[x,y][2] / 255)
                                X_train[trainNum][x][y][0] = tempColor
                                tempColor = 0
                        trainNum+=1
                # 1/10th is the validation (or test) set
                else:
                        # Append the class label as Byte
                        Y_test.append(class_name)
                        for x in range(0,sizeW):
                        for y in range(0,sizeH):
                                #for color in range(0,3):
                                tempColor += 0.21 * float(pix[x,y][0] / 255)
                                tempColor += 0.72 * float(pix[x,y][1] / 255)
                                tempColor += 0.07 * float(pix[x,y][2] / 255)
                                X_train[testNum][x][y][0] = tempColor
                                tempColor = 0
                        testNum+=1

        print("Height", h)
        print("Width", w)
        print(i)
        print("test", testNum)
        print("train", trainNum)
        return (X_train, Y_train), (X_test, Y_test)
```

```


# Scanning routine that produces 10 subsets of the image as output
# - example call: slidingWindow("test-dataset/file27.jpeg", 32)
def slidingWindow(img, sizeH, sizeW):
    # open image
    img = Image.open(img)

    # load pixels RGB arrays
    pix = img.load()

    # get image dimensions
    width = int(img.size[0])
    print("width: " + str(width))
    height = int(img.size[1])
    print("height: " + str(height))
    ##### 1280 (40*32) x 720 (30*32)
    ##### 720 (30*32) x 1280 (40*32)

    # output array
    out = []

    # first append full image resized to nodeSize*nodeSize
    out.append(imresize(img, (sizeH, sizeW), interp="bicubic").astype(np.float32, casting='unsafe'))

    #convert to numpy array
    imArr = np.empty((height,width,3))
    for x in range(0,w):
    for y in range(0,width):
            for color in range(0,3):
            imArr[x][y][color] = pix[x,y][color]

    ##### 1/4 rotation
    xStep = int(height/2)
    yStep = int(width/2)
    img_4 = []
    for i in range(0,9):
    img_4.append(np.empty((xStep,yStep,3)))

    xOffset = 0
    yOffset = 0
    index = 0
    # 5-level loop, best loop
    for w in range(0,3):
```

```
    for h in range(0,3):
        for x in range(0, xStep):
        for y in range(0, yStep):
            for color in range(0,3):
            tarX = int(x+(xOffset*xStep/2))
            tarY = int(y+(yOffset*yStep/2))
            img_4[index][x][y][color] = imArr[tarX][tarY][color]

        index += 1
        xOffset += 1
        print("yo")
    yOffset += 1
    xOffset = 0

    # Append the 4 generated images to output array    after resizing them
    for i in range(0,9):
    temp = toimage(img_4[i])
    out.append(imresize(temp, (sizeH, sizeW), interp="bicubic").astype(np.float32, casting='unsafe'))
    return out

# freeze_graph to save current weights into file to use in C++ testing routine
def save_graph(sess,output_path,checkpoint,checkpoint_state_name,input_graph_name,output_graph_name):
    # We save out the graph to disk, and then call the const conversion
    # routine.
    checkpoint_state_name = "checkpoint_state"
    input_graph_name = "input_graph.pb"
    output_graph_name = "output_graph.pb"

    input_graph_path = os.path.join("C:/Users/George/Documents/MQP/tmp/", input_graph_name)
    input_saver_def_path = ""
    input_binary = False
    input_checkpoint_path = os.path.join("C:/Users/George/Documents/MQP/", 'saved_checkpoint') + "-9000"

    # Note that we this normally should be only "output_node"!!!
    output_node_names = "out/Softmax"
    restore_op_name = "save/restore_all"
    filename_tensor_name = "save/Const:0"
    output_graph_path = os.path.join("C:/Users/George/Documents/MQP/tmp/", output_graph_name)
    clear_devices = True

    export.freeze_graph(input_graph_path, input_saver_def_path,
                        input_binary, input_checkpoint_path,
                output_node_names, restore_op_name,
                filename_tensor_name, output_graph_path,
                clear_devices, "")

# Scanning routine that produces 10 subsets of the image as output
# - example call: slidingWindow("test-dataset/file27.jpeg", 32)
def slidingWindowDemo(img, sizeH, sizeW):
    # open image
    img = Image.open(img)
    # img = cv2.imread(img,1)
    # load pixels RGB arrays
    pix = img.load()
    # pix = np.asarray(img)
    # get image dimensions
    width = int(img.size[0])
    print("width: " + str(width))
    height = int(img.size[1])
    print("height: " + str(height))
    ##### 1280 (40*32) x 720 (30*32)
    ##### 720 (30*32) x 1280 (40*32)

    out = []
    # first append full image resized to nodeSize*nodeSize
    out.append(imresize(img, (sizeH, sizeW), interp="bicubic").astype(np.float32, casting='unsafe'))

    # #convert to numpy array
    imArr = np.empty((height,width,3))
    for x in range(0,height):
    for y in range(0,width):
        for color in range(0,3):
        imArr[x][y][color] = pix[y,x][color]

    ##### 1/4 rotation
    xStep = int(height/2)
    yStep = int(width/2)
    img_4 = []
    for k in range(0,4):
    img_4.append(np.empty((xStep,yStep,3)))

    index = 0
    for w in range(0,2):
    for h in range(0,2):
        for x in range(0, xStep):
        for y in range(0, yStep):
            for color in range(0,3):
                tarX = int(x+(xStep * w))
                tarY = int(y+(yStep * h))
                img_4[index][x][y][color] = imArr[tarX][tarY][color]
        index +=1


    # Append the 4 generated images to output array    after resizing them
    for i in range(0,4):
    temp = toimage(img_4[i])
    width = int(temp.size[0])
    print("width: " + str(width))
    height = int(temp.size[1])
    print("height: " + str(height))
    out.append(imresize(temp, (sizeH, sizeW), interp="bicubic").astype(np.float32, casting='unsafe'))
    return out


# RGB to GRAY formula
def rgb2gray(rgb):
    return np.dot(rgb[...,:3], [0.299, 0.587, 0.114])

# Converts the training dataset from RGB to BW
# - example call: convertToBW("C:/Users/George/Documents/MQP/train-dataset",  ".jpeg", 34)
def convertToBW(dir, type, imageNum):
    for i in range(imageNum):
    if(i >= 0):
        # Load the image file
        # img = mpimg.imread('image.png')
        # gray = rgb2gray(img)
        # plt.imshow(gray, cmap = plt.get_cmap('gray'))
        # plt.show()

        readPath = dir + '/0/small-file' + str(i) + type
        writePath = "C:/Users/George/Documents/MQP/train-dataset-bw/0/small-file" + str(i) + '.jpg'
        img = Image.open(readPath).convert('L')
        img.save(writePath)
        # im_gray = cv2.imread(dir + '/1/small-file' + str(i) + type, cv2.IMREAD_GRAYSCALE)
        # #(thresh, im_bw) = cv2.threshold(im_gray, 128, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)
        # cv2.imwrite("C:/Users/George/Documents/MQP/train-dataset-bw/1/small-file" + str(i) + '.jpg', im_gray)
        # #Save the image to new folder

# Test sliding window, only appends original image
def slidingWindowDemoTest(img, nodeSize):
    # open image
    img = Image.open(img)

    # load pixels RGB arrays
    pix = img.load()

    # get image dimensions
    width = int(img.size[0])
    print("width: " + str(width))
    height = int(img.size[1])
    print("height: " + str(height))
    ##### 1280 (40*32) x 720 (30*32)
    ##### 720 (30*32) x 1280 (40*32)

    # output array
    out = []

    # first append full image resized to nodeSize*nodeSize
    out.append(imresize(img, (nodeSize, nodeSize), interp="bicubic").astype(np.float32, casting='unsafe'))

    return out


# Converts captures frames to their quarters. Essentially, creating a dataset 4x as big.
def toQuarters(imageNum):
    for i in range(imageNum):
    if(i%8 == 0):
        # Load the image file
        img = Image.open('toQuarter/file' + str(i) + '.jpg')

        pix = imresize(img, (720, 1280), interp="bicubic").astype(np.float32, casting='unsafe')
        # load pixels RGB arrays
        # pix = img.load()

        # get image dimensions
        width = int(img.size[0])
        print("width: " + str(width))
        height = int(img.size[1])
        print("height: " + str(height))
        ##### 1280 (40*32) x 720 (30*32)
        ##### 720 (30*32) x 1280 (40*32)

        #convert to numpy array
        imArr = np.empty((height,width,3))
        for x in range(0,height):
        for y in range(0,width):
            for color in range(0,3):
                imArr[x][y][color] = pix[x,y][color]

        ##### 1/4 rotation
        xStep = int(height/2)
        yStep = int(width/2)
        img_4 = []
        for k in range(0,4):
        img_4.append(np.empty((xStep,yStep,3)))

        index = 0
        for w in range(0,2):
        for h in range(0,2):
            for x in range(0, xStep):
            for y in range(0, yStep):
                for color in range(0,3):
                    tarX = int(x+(xStep * w))
                    tarY = int(y+(yStep * h))
                    img_4[index][x][y][color] = imArr[tarX][tarY][color]
            index +=1


        # Append the 9 generated images to output array    after resizing them
        for z in range(0,4):
        temp = toimage(img_4[z])
        #Save the image to new folder
        temp.save('Quarters/file' + str(i) + str(z) + '.jpg')
```

Appendix - F: Matlab ImageToText.m code

```
clc; close all

[fName,pName] = uigetfile('*.*');
row = 128;col = 128; noImage = 2;
I = imresize(imread([pName fName]), [row col]);
figure, imshow(uint8(I));

fid = fopen('input.txt', 'w');

for noI = 1:noImage
  for i = 1:row
    for j = 1:col
      %Break and Concat Color Component
      fprintf(fid, '%s\n',[dec2hex(I(i,j,1),2) dec2hex(I(i,j,2),2) dec2hex(I(i,j,3),2)]);
    end
  end
end
fclose(fid);
```

Appendix - G: Matlab TextToImage.m code

```
clc; close all;

fid = fopen("result_img.txt");
txtData = textscan(fid, '%s');
txtData = cell2mat(txtData{1,1});

textImage = zeros(row,col,3);

for noI = 1:noImage
    idx = (noI - 1) *row*col + 1:noI*row*col;
    tempData = txtData(idx,:);
    textImage(:,:,1) = reshape(hex2dec(tempData(:,1:2)), col, row)';
    textImage(:,:,2) = reshape(hex2dec(tempData(:,3:4)), col, row)';
    textImage(:,:,3) = reshape(hex2dec(tempData(:,5:6)), col, row)';

    figure, imshow(uint8(textImage));
end
```

Appendix - H: Sobel Filter Verilog Code

```
// -------------------------------------------------------------
//
// File Name: C:\Users\skids\Documents\MATLAB\hdl_prj\Sobel\hdlcoder_sobel\SobelCore.v
// Created: 2017-02-21 20:43:31
//
// Generated by MATLAB 9.1 and HDL Coder 3.9
//
// -------------------------------------------------------------


// -------------------------------------------------------------
//
// Module: SobelCore
// Source Path: hdlcoder_sobel/Pixel-Stream HDL Model/Edge Detection/Edge Detector/SobelCore
// Hierarchy Level: 3
//
// Sobel Core
//
// -------------------------------------------------------------

`timescale 1 ns / 1 ns

module SobelCore
          (
           clk,
           reset,
           enb,
           pixelInVec_0,
           pixelInVec_1,
           pixelInVec_2,
           ShiftEnb,
           Gv,
           Gh
          );



  input   clk;
  input   reset;
  input   enb;
  input   [7:0] pixelInVec_0;  // uint8
  input   [7:0] pixelInVec_1;  // uint8
  input   [7:0] pixelInVec_2;  // uint8
  input   ShiftEnb;
  output  signed [10:0] Gv;  // sfix11_En3
  output  signed [10:0] Gh;  // sfix11_En3

  reg [7:0] pixel1Shift;  // uint8
  reg [7:0] pixel1Shift2;  // uint8
  reg [7:0] pixel1Shift3;  // uint8
  reg [7:0] pixel3Shift;  // uint8
  reg [7:0] pixel3Shift2;  // uint8
  reg [7:0] pixel3Shift3;  // uint8
  wire [8:0] adder_1;  // ufix9
  wire [8:0] adder_2;  // ufix9
  wire [8:0] GvAdder1;  // ufix9
  reg [8:0] GvAdder1Delay;  // ufix9
  reg [7:0] pixel2Shift;  // uint8
  reg [7:0] pixel2Shift2;  // uint8
  reg [7:0] pixel2Shift3;  // uint8
  wire [7:0] p2S3x2;  // ufix8_E1
  reg [7:0] p2S3x2Delay;  // ufix8_E1
  wire [9:0] adder_add_cast;  // ufix10
  wire [9:0] adder_4;  // ufix10
  wire [9:0] GvAdder2;  // ufix10
  reg [9:0] GvAdder2Delay;  // ufix10
  wire [7:0] p2Sx2;  // ufix8_E1
  reg [7:0] p2Sx2Delay;  // ufix8_E1
  wire [8:0] adder_6;  // ufix9
  wire [8:0] adder_7;  // ufix9
  wire [8:0] GvAdder3;  // ufix9
  reg [8:0] GvAdder3Delay;  // ufix9
  wire [9:0] adder_add_cast_1;  // ufix10
  wire [9:0] adder_9;  // ufix10
  wire [9:0] GvAdder4;  // ufix10
  reg [9:0] GvAdder4Delay;  // ufix10
  wire [10:0] subtractor_sub_temp;  // ufix11
  wire [10:0] subtractor_1;  // ufix11
  wire [10:0] subtractor_2;  // ufix11
  wire signed [10:0] GvAdder5;  // sfix11
  wire signed [10:0] gvdtc1;  // sfix11_En3
  reg signed [10:0] gvdtc1Delay;  // sfix11_En3
  wire [7:0] p3S2x2;  // ufix8_E1
```

```
reg [7:0] p3S2x2Delay;  // ufix8_E1
wire [8:0] adder_11;  // ufix9
wire [8:0] adder_12;  // ufix9
wire [8:0] GhAdder3;  // ufix9
reg [8:0] GhAdder3Delay;  // ufix9
wire [9:0] adder_add_cast_2;  // ufix10
wire [9:0] adder_14;  // ufix10
wire [9:0] GhAdder4;  // ufix10
reg [9:0] GhAdder4Delay;  // ufix10
wire [7:0] p1S2x2;  // ufix8_E1
reg [7:0] p1S2x2Delay;  // ufix8_E1
wire [8:0] adder_16;  // ufix9
wire [8:0] adder_17;  // ufix9
wire [8:0] GhAdder1;  // ufix9
reg [8:0] GhAdder1Delay;  // ufix9
wire [9:0] adder_add_cast_3;  // ufix10
wire [9:0] adder_19;  // ufix10
wire [9:0] GhAdder2;  // ufix10
reg [9:0] GhAdder2Delay;  // ufix10
wire [10:0] subtractor_sub_temp_1;  // ufix11
wire [10:0] subtractor_4;  // ufix11
wire [10:0] subtractor_5;  // ufix11
wire signed [10:0] GhAdder5;  // sfix11
wire signed [10:0] ghdtc1;  // sfix11_En3
reg signed [10:0] ghdtc1Delay;  // sfix11_En3


always @(posedge clk or posedge reset)
  begin : p1Shift_process
    if (reset == 1'b1) begin
      pixel1Shift <= 8'b00000000;
    end
    else begin
      if (enb && ShiftEnb) begin
        pixel1Shift <= pixelInVec_0;
      end
    end
  end



always @(posedge clk or posedge reset)
  begin : p1Shift2_process
```

```
    if (reset == 1'b1) begin
      pixel1Shift2 <= 8'b00000000;
    end
    else begin
      if (enb && ShiftEnb) begin
        pixel1Shift2 <= pixel1Shift;
      end
    end
  end



always @(posedge clk or posedge reset)
  begin : p1Shift3_process
    if (reset == 1'b1) begin
      pixel1Shift3 <= 8'b00000000;
    end
    else begin
      if (enb && ShiftEnb) begin
        pixel1Shift3 <= pixel1Shift2;
      end
    end
  end



always @(posedge clk or posedge reset)
  begin : p3Shift_process
    if (reset == 1'b1) begin
      pixel3Shift <= 8'b00000000;
    end
    else begin
      if (enb && ShiftEnb) begin
        pixel3Shift <= pixelInVec_2;
      end
    end
  end



always @(posedge clk or posedge reset)
  begin : p3Shift2_process
    if (reset == 1'b1) begin
    pixel3Shift2 <= 8'b00000000;
   end
   else begin
    if (enb && ShiftEnb) begin
     pixel3Shift2 <= pixel3Shift;
    end
   end
  end



always @(posedge clk or posedge reset)
  begin : p3Shift3_process
   if (reset == 1'b1) begin
    pixel3Shift3 <= 8'b00000000;
   end
   else begin
    if (enb && ShiftEnb) begin
     pixel3Shift3 <= pixel3Shift2;
    end
   end
  end



assign adder_1 = {1'b0, pixel1Shift3};
assign adder_2 = {1'b0, pixel3Shift3};
assign GvAdder1 = adder_1 + adder_2;



always @(posedge clk or posedge reset)
  begin : reg_rsvd_process
   if (reset == 1'b1) begin
    GvAdder1Delay <= 9'b000000000;
   end
   else begin
    if (enb) begin
     GvAdder1Delay <= GvAdder1;
    end
   end
  end
```

```
always @(posedge clk or posedge reset)
  begin : p2Shift_process
    if (reset == 1'b1) begin
      pixel2Shift <= 8'b00000000;
    end
    else begin
      if (enb && ShiftEnb) begin
        pixel2Shift <= pixelInVec_1;
      end
    end
  end



always @(posedge clk or posedge reset)
  begin : p2Shift2_process
    if (reset == 1'b1) begin
      pixel2Shift2 <= 8'b00000000;
    end
    else begin
      if (enb && ShiftEnb) begin
        pixel2Shift2 <= pixel2Shift;
      end
    end
  end



always @(posedge clk or posedge reset)
  begin : p2Shift3_process
    if (reset == 1'b1) begin
      pixel2Shift3 <= 8'b00000000;
    end
    else begin
      if (enb && ShiftEnb) begin
        pixel2Shift3 <= pixel2Shift2;
      end
    end
  end
```

```
assign p2S3x2 = pixel2Shift3;



always @(posedge clk or posedge reset)
  begin : reg_rsvd_1_process
    if (reset == 1'b1) begin
      p2S3x2Delay <= 8'b00000000;
    end
    else begin
      if (enb) begin
        p2S3x2Delay <= p2S3x2;
      end
    end
  end



assign adder_add_cast = {1'b0, {p2S3x2Delay, 1'b0}};
assign adder_4 = {1'b0, GvAdder1Delay};
assign GvAdder2 = adder_4 + adder_add_cast;



always @(posedge clk or posedge reset)
  begin : reg_rsvd_2_process
    if (reset == 1'b1) begin
      GvAdder2Delay <= 10'b0000000000;
    end
    else begin
      if (enb) begin
        GvAdder2Delay <= GvAdder2;
      end
    end
  end



assign p2Sx2 = pixel2Shift;
```

```
always @(posedge clk or posedge reset)
  begin : reg_rsvd_3_process
    if (reset == 1'b1) begin
      p2Sx2Delay <= 8'b00000000;
    end
    else begin
      if (enb) begin
        p2Sx2Delay <= p2Sx2;
      end
    end
  end



assign adder_6 = {1'b0, pixel1Shift};
assign adder_7 = {1'b0, pixel3Shift};
assign GvAdder3 = adder_6 + adder_7;



always @(posedge clk or posedge reset)
  begin : reg_rsvd_4_process
    if (reset == 1'b1) begin
      GvAdder3Delay <= 9'b000000000;
    end
    else begin
      if (enb) begin
        GvAdder3Delay <= GvAdder3;
      end
    end
  end



assign adder_add_cast_1 = {1'b0, {p2Sx2Delay, 1'b0}};
assign adder_9 = {1'b0, GvAdder3Delay};
assign GvAdder4 = adder_add_cast_1 + adder_9;



always @(posedge clk or posedge reset)
  begin : reg_rsvd_5_process
    if (reset == 1'b1) begin
    GvAdder4Delay <= 10'b0000000000;
   end
   else begin
    if (enb) begin
     GvAdder4Delay <= GvAdder4;
    end
   end
  end



assign subtractor_1 = {1'b0, GvAdder2Delay};
assign subtractor_2 = {1'b0, GvAdder4Delay};
assign subtractor_sub_temp = subtractor_1 - subtractor_2;
assign GvAdder5 = subtractor_sub_temp;



// Gv: Right-shift 3 bit to perform divided by 8
assign gvdtc1 = GvAdder5;



always @(posedge clk or posedge reset)
  begin : reg_rsvd_6_process
   if (reset == 1'b1) begin
    gvdtc1Delay <= 11'sb00000000000;
   end
   else begin
    if (enb) begin
     gvdtc1Delay <= gvdtc1;
    end
   end
  end



assign p3S2x2 = pixel3Shift2;



always @(posedge clk or posedge reset)
  begin : reg_rsvd_7_process
```

```
    if (reset == 1'b1) begin
      p3S2x2Delay <= 8'b00000000;
    end
    else begin
      if (enb) begin
        p3S2x2Delay <= p3S2x2;
      end
    end
  end



assign adder_11 = {1'b0, pixel3Shift};
assign adder_12 = {1'b0, pixel3Shift3};
assign GhAdder3 = adder_11 + adder_12;



always @(posedge clk or posedge reset)
  begin : reg_rsvd_8_process
    if (reset == 1'b1) begin
      GhAdder3Delay <= 9'b000000000;
    end
    else begin
      if (enb) begin
        GhAdder3Delay <= GhAdder3;
      end
    end
  end



assign adder_add_cast_2 = {1'b0, {p3S2x2Delay, 1'b0}};
assign adder_14 = {1'b0, GhAdder3Delay};
assign GhAdder4 = adder_add_cast_2 + adder_14;



always @(posedge clk or posedge reset)
  begin : reg_rsvd_9_process
    if (reset == 1'b1) begin
      GhAdder4Delay <= 10'b0000000000;
    end
    else begin
      if (enb) begin
        GhAdder4Delay <= GhAdder4;
      end
    end
  end



assign p1S2x2 = pixel1Shift2;



always @(posedge clk or posedge reset)
  begin : reg_rsvd_10_process
    if (reset == 1'b1) begin
      p1S2x2Delay <= 8'b00000000;
    end
    else begin
      if (enb) begin
        p1S2x2Delay <= p1S2x2;
      end
    end
  end



assign adder_16 = {1'b0, pixel1Shift};
assign adder_17 = {1'b0, pixel1Shift3};
assign GhAdder1 = adder_16 + adder_17;



always @(posedge clk or posedge reset)
  begin : reg_rsvd_11_process
    if (reset == 1'b1) begin
      GhAdder1Delay <= 9'b000000000;
    end
    else begin
      if (enb) begin
        GhAdder1Delay <= GhAdder1;
      end
    end
  end



assign adder_add_cast_3 = {1'b0, {p1S2x2Delay, 1'b0}};
assign adder_19 = {1'b0, GhAdder1Delay};
assign GhAdder2 = adder_add_cast_3 + adder_19;



always @(posedge clk or posedge reset)
  begin : reg_rsvd_12_process
    if (reset == 1'b1) begin
      GhAdder2Delay <= 10'b0000000000;
    end
    else begin
      if (enb) begin
        GhAdder2Delay <= GhAdder2;
      end
    end
  end



assign subtractor_4 = {1'b0, GhAdder4Delay};
assign subtractor_5 = {1'b0, GhAdder2Delay};
assign subtractor_sub_temp_1 = subtractor_4 - subtractor_5;
assign GhAdder5 = subtractor_sub_temp_1;



// Gh: Right-shift 3 bit to perform divided by 8
assign ghdtc1 = GhAdder5;



always @(posedge clk or posedge reset)
  begin : reg_rsvd_13_process
    if (reset == 1'b1) begin
      ghdtc1Delay <= 11'sb00000000000;
    end
    else begin
      if (enb) begin
```

```
          ghdtc1Delay <= ghdtc1;
        end
      end
    end



  // Gv: Cast to the specified gradient data type. Full precision if outputing binary image only
  assign Gv = gvdtc1Delay;

  // Gh: Cast to the specified gradient data type. Full precision if outputing binary image only
  assign Gh = ghdtc1Delay;

endmodule  // SobelCore
```