|  |
| --- |
| ZipDt |
| Embedded Computer Vision Application |
| With ZedBoard |
|  |
| **Michael Davis, Michael Driscoll, Yifan Ge** |
| **4/29/2013** |

|  |
| --- |
| This document outlines a senior design project in which an image processing application is developed for the ZedBoard. This project aims to incorporate OpenCV image processing functions in software along with hardware implementations of certain image processing operations as accelerators on the Zynq. |

Table of Contents

[Project Overview 2](#_Toc355020036)

[Background 3](#_Toc355020037)

[Proposed Solution 3](#_Toc355020038)

[Design Overview 3](#_Toc355020039)

[Block Game Behavior 3](#_Toc355020040)

[Engineering Process 4](#_Toc355020041)

[Filtering Methods 5](#_Toc355020042)

[Skin Color Filtering 5](#_Toc355020043)

[Background Subtraction 5](#_Toc355020044)

[Object Recognition 6](#_Toc355020045)

[HAAR Classification 6](#_Toc355020046)

[Hough Transform 6](#_Toc355020047)

[Contour Analysis 6](#_Toc355020048)

[Hand Gesture Recognition Methods 7](#_Toc355020049)

[Accelerators 7](#_Toc355020050)

[Memory Sharing 8](#_Toc355020051)

[Specification, Testing, and Results 10](#_Toc355020052)

[Adjustments 21](#_Toc355020053)

[In Retrospect 21](#_Toc355020054)

[Design Evolution 23](#_Toc355020055)

[System Architecture 23](#_Toc355020056)

[System - version 1 23](#_Toc355020057)

[System - version 2 26](#_Toc355020058)

[Software Implementation 26](#_Toc355020059)

[Final Implementation Details 27](#_Toc355020060)

[Bill of Materials 32](#_Toc355020061)

[Discussion 32](#_Toc355020062)

[In Retrospect 33](#_Toc355020063)

[References 36](#_Toc355020064)

# Project Overview

This goal of this project was to design an application for the ZedBoard that would stimulate interest in using the board for project development. The ZedBoard was released in Fall 2012 and is a Zynq development board distributed by Avnet. For this project, the application was designed to demonstrate two of the original target applications of the board: video processing and hardware acceleration. To demonstrate the capabilities of the Zynq platform, the application is also specified to process high definition (HD) images with a base resolution of 720p. Image processing operations are performed on input images from a USB webcam and processed to accomplish responsive game control. In order to reinforce the benefits of hardware acceleration and show the practicality of integrating accelerators using Zynq, the project was designed in two phases. The first phase involved application implementation using a purely software solution, with OpenCV algorithms running entirely on the ARM processors of the Zynq chip. The second phase included utilization of the programmable logic (PL) to implement a hardware accelerator capable of decreasing the execution time of chosen OpenCV functions.

In order to accomplish the assigned goals, the project was designed to implement a motion-controlled block game similar to Tetris. The choice of this application allowed us to demonstrate the capabilities of the ZedBoard in relation to both time variant and time invariant analysis. This game was designed to be controlled by a player’s hand in the field of view of the camera. For controlling the horizontal location of a piece, analysis was performed on the center of mass of the user’s hand between frame. Rotation controls were implemented by analyzing the number of fingers held up by the user. Because hand gestures were chosen to provide input commands, the application required no external peripherals, minimizing the resulting costs of our project.

After developing the application in software, specific OpenCV operations were identified as candidates for acceleration based on their execution time and algorithm. For this particular implementation, morphological open and close operations for noise filtering were identified as good candidates based on their high execution time in relation to other OpenCV functions utilized in the program. In OpenCV, these operations are accomplished by calling the cvErode() and cvDilate() functions. In order to acceleration these functions, synthesis of appropriate pcores was performed with Vivado HLS. For cvErode(), an accelerator was produced that was capable of performing an erosion operation three to four times faster than its software counterpart. A cvDilate() accelerator was also generated, but this did not achieve a positive acceleration factor due to a limited optimization time period for accelerator development.

# Background

## Proposed Solution

In order to begin development of the specified motion-controlled block game, a use scenario was developed to provide a more detailed description about how the game should behave. This explanation is provided below.

### Design Overview

The application is designed as a hand motion and hand gesture controlled block game. We have defined hand gestures as the number of fingers the user has extended on the hand being monitored by the system. Hand motion is defined as the movement of the hand in the columns of pixels between subsequent captured frames.

Prior to the start of gameplay, the system will undergo a short calibration period. During this time the user should remain still and stand in the location that they will play the game from. The user’s hand that will be used to control the game should not be held up during this time. This calibration stage will gather information about the environment so that the system may better extract the user’s hand motion and gestures.

To begin the game, the user will raise their hand that will be used to control the game with their palm parallel to the camera’s image plane. At this point the block game will be initiated. The user will be able to then control the horizontal motion of game pieces by moving their hand from side to side in front of the camera. The orientation of each piece will be controlled by changing the number of fingers the user has extended.

### Block Game Behavior

The motion-controlled game consists of seven different blocks, which are geometric shapes composed of four square blocks shown in Figure 1. The playing space, or board, is a 10 column by 22 row space (only 20 of which are visible).

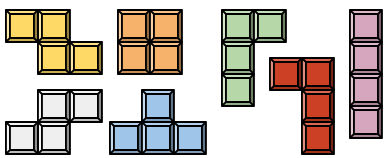


Figure 1 The seven shapes of the blocks used in the block game. The colors may change upon implementation.

The user begins the game with zero points, an empty playing area, and on the first level of difficulty. At this point, a tetromino is randomly generated for the user at the top of the screen. This piece begins to descend downward in the playing space at a relatively slow speed (due to the lowest difficulty level). At this point, the user is able to rotate the piece by 90 degree clockwise increments, move the piece from side to side within the playing area, or force the piece to drop as far as it can in the playing space. Game pieces may not be rotated or moved if the manipulation results in a collision between the user-controlled piece and a previously placed tetromino or the boundaries of the playing area. When a piece reaches the bottom of the playing area or attempts to move downward onto another piece it stops its descent and the user is no longer able to control the piece. When the user loses control of a tetromino, another is randomly generated at the top of the screen and the process is repeated. The game ends when tetrominoes are stacked to the top of the playing area, preventing another tetromino from being generated.

The user is able to score points in the game by filling an entire row in the playing area. When an entire row is filled with blocks of tetrominoes, it is deleted causing all blocks above the cleared row to move downward one space to accommodate for the removed row. Due to the shapes of the tetrominoes, it is possible to eliminate more than one row at a time. These row completions are more difficult, and as such, reward the user with more points. The scoring is as follows: 1 row 40 points, 2 rows 100 points, 3 rows 300 points, 4 rows 1200 points. The level of difficulty is increased with every 10 rows cleared. As the difficulty is increased, the rate of descent of the user-controlled tetrominoes is increased.

## Engineering Process

As an initial step in our design process, we performed significant research pertaining to standard image processing methods and hardware acceleration. This research guided the initial design evolution of the project and the system architecture. As explained in later sections, additional research was performed based on the results of various implementations of the methods listed below. The sections below describe in detail the initial design considerations for the project, as well as critical information that guided project development.

### Filtering Methods

Most image processing applications involve some form of image filtering in order to isolate specific features of an image. As a result, an investigation of several likely implementations of filtering for our game development is warranted to ensure that each method is feasible. Two likely methods of image filtering that would assist in isolating the hand for gesture and motion recognition are skin color filtering and background subtraction.

### Skin Color Filtering

As as initial method of filtering, skin color filtering can be used to eliminate a significant number of objects from a frame. This is fairly simple to accomplish in software, as the use of a single OpenCV method, cvCvtColor(), can be used in order to effectively isolate pixels in a frame that correspond to a predetermined pixel range. It is important to note that conversion to another color space is commonly performed in order to provide a more accurate color range for skin tone. With OpenCV, we were able to develop a filter that isolated skin using HSV values. Other examples have implemented a similar filter using the YCbCr color space. The YCbCr color space is generally more commonly used than RGB, as RGB is not considered efficient for transmission purposes. More complex methods for skin filtering include the use of a probabilistic model, which involves an algorithmic determination of color range values based on a number of sample images.

### Background Subtraction

With the desire to use hand gestures, one aspect that needed to be addressed was the influence of the user’s face on gesture recognition. One possible way to eliminate this potential problem is through background subtraction. OpenCV contains direct functionality to perform background subtraction on a frame. In practical applications, background subtraction should be based on a large number of images in order to eliminate noise from the resulting filtered image. We were able to develop a background subtraction algorithm based on 300 frames, which requires 10 seconds of calibration based on a 30 fps camera. Because we desire less than 45 seconds for calibration time in our final design, we did not desire to implement an algorithm based on more frames, as the frame rate of the program may decrease when utilizing the ZedBoard. Further filtering may be performed on a image after background subtraction in order to eliminate any stray noise or small movement that could potentially interfere with image processing techniques. OpenCV contains functionality to effectively eliminate small regions identified to be in the foreground, as well as join large regions of pixels in the foreground to form blobs.

### Object Recognition

As our desired motion-controlled game requires some form of object recognition in order to determine several unique hand gestures, we analyzed our prior research in order to determine which algorithms could be implemented with OpenCV to perform the desired task. Three object recognition methods that were researched throughout the course of the semester included Haar classification, Hough transforms, and Contour analysis.

### HAAR Classification

Haar classification involves analysis of an image based on the use of a Haar classifier .xml file. This .xml file is created through analysis of description files that contain both positive and negative samples of the desired object. OpenCV contains a Haar training utility that may be used to create an .xml file from the aforementioned description files. In a sample implementation, we were able to develop a program that effectively identified faces in a frame. This was accomplished by using one of the facial recognition Haar classifier files that was packaged with the OpenCV library. A possible method for developing our own .xml file would involve storing all available frames from a camera capture and packaging those images into corresponding description files. In terms of hardware implementation, we researched several cases in which Haar classification was implemented with an FPGA. On standard resolution (480p) images, cases involving facial recognition usually took approximately 0.25 seconds. However, this was accomplished at a fairly low frame rate (<10 fps). [2] [3]

### Hough Transform

For hough transforms, we found that this method could be used to identify the circularity of fingertips and webbing in an individual frame. We could then associate the number of detected circles in an image, as well as their relative position, to a specific gesture. Furthermore, there are many existing examples that provide an outline on how to perform a Hough transform with an FPGA. However, upon researching the algorithm to perform this task in hardware, we found it to be computationally expensive. This is rather costly considering response time is a high priority task for the project. One example of a hough transform, as implemented on an FPGA, resulted in a throughput of 25 fps. This did not appear to be promising, largely due to the fact that the throughput would likely decrease when implementing the algorithm in software. [4] [5]

### Contour Analysis

We also analyzed the potential of determining the contour of the objects in a frame and analyzing said contour to associate the object with a particular gesture. We were able to implement this object recognition method in OpenCV by utilizing particular functions that streamline the process to determine all contours in an image. Prior to contour determination, extensive filtering was required in order to isolate the hand in the frame. Concerning hardware implementation, we again found several examples where contour tracing was performed with an FPGA. From these examples, we found that the feasibility of implementing contour tracing for our project in hardware is fairly high. This is because the throughput of the contour tracing algorithm in hardware, with parallelism, can approach 200 fps. As a result, a contour accelerator would likely improve the performance of our overall system, which is a desirable feature of our project in order to highlights the capabilities of the ZedBoard. [6]

### Hand Gesture Recognition Methods

As our game implementation will implement hand gesture recognition, we found it useful to research exact cases where a similar operation was performed in hardware. Two academic papers, which provided a promising example for our final design, involved the use of an FPGA to identify the number of finger an individual was holding up. This was accomplished by isolating the hand initially using a skin-color model. After filtering, the center of mass of the hand was determined using an average of all pixels within the identified hand area. The number of fingers could then be determined by enumerating the number of color transitions at a discrete radius around the center of mass of the hand. This may be applicable to our development of a motion-controlled game because we could directly relate the number of fingers being held up to the rotational position of a block. If we were to use this method, we would likely implement background subtraction and small area filtering in order to eliminate the user’s face from the camera image, as the skin filter alone would not accomplish this task. [7] [8]

### Accelerators

The hardware accelerated version of the project will feature custom hardware accelerators in the programmable logic of the Zynq. The goal of these accelerators will be to increase the performance of an image processing operation. Implementing the hardware accelerators will likely take a significant amount of time and effort. As a result, the portions of the algorithm that are the most computationally intensive or would most benefit from parallelization will be considered the best candidates for acceleration in order to provide the most benefit for our efforts.

The accelerators will be visible in the application space of the PS. To communicate with the accelerators, Linux device drivers will be written to access the peripheral’s control registers from the application space. Use of the open, mmap, ioctl, read, and write system function calls will be good command candidates for interfacing with the hardware accelerator peripheral. In order to control access to the shared frame buffers while the accelerator is operating on the data, semaphores will be used to provide mutual exclusion. Interrupts will be connected to the PS interrupt controller from the PL to signal accelerator operation completion and to post the frame buffer semaphore.

### Memory Sharing

Since the processing system will be reading images from the camera and storing them in memory, the accelerator peripherals in the programmable logic must share a memory range with the processing system. Multiple methods for sharing memory have been investigated over the course of the past semester.

The Direct Memory Access Channel (DMAC) peripheral request interface is a feature of the processing system’s DMA controller which allows CPU-visible dedicated DMA channels to be directed to peripherals. These channels allow specific memory-to-peripheral and peripheral-to-memory transfers to be made possible using DMA from peripheral memory that might otherwise not be addressable. This feature has the benefit of allowing multiple DMA transfers to occur simultaneously, however they must all be initiated by the CPU. Having the CPU initiate all partial transfers of images to the PL would not be ideal for a hardware accelerator because it would introduce additional latency to the system, creating initial limits on the total possible acceleration that could be achieved. In addition, the extra communication between the accelerator peripheral and the CPU would increase design complexity. [9]

The Accelerator Coherency Port (ACP) is a cache coherent interconnect between the PS and PL in the Zynq. For our project, this interconnect will not be ideal since cache values will be modified on memory access. This could result in cache thrashing and limit the performance of the OS and our image processing application, since the frequent large memory accesses made by the accelerator would be evicting data being used by other applications from the share cache. Since the data being accessed by the accelerator is not likely to be re-used by the CPU this will have a solely negative effect on system performance. [10]

The AXI general purpose (GP) interconnects are useful for sending short messages between the PS and PL. This type of interconnect will allow PL access to the memory controller when configured through the central interconnect controller, but using this variety of interconnect is not ideal for large transfers since AXI GP interconnects lack FIFOs. In addition, these interconnects do not support a burst mode, can only handle up to 8 issued reads/writes, and are limited to a 32-bit data bus width. All of these factors make the AXI GP interconnects not a favorable option for image data transfer, however, they are effective, as previously mentioned, for short message passing. These interconnects will the appropriate choice for the peripheral control interface to the PS. [10]

The AXI high performance (HP) interconnect provides a direct pathway from the programmable logic to the processing system memory controller, allowing the attached peripheral to initiate its own memory transfers. This interconnect features a configurable data bus width between 32 and 64 bits as well as a configurable burst length. These features allow the memory communication to be optimized for a given application. One of the most promising features of the AXI HP interconnects is the presence of FIFOs to buffer read/write values. These allow a large number of read and write requests to be placed and handled when there is an opportunity, lowering overall transfer latencies. Creating an accelerator with a master-mode AXI HP interconnect will allow the accelerator to initiate memory transfers whenever the peripheral requires additional image data, or must write new data to shared memory. This autonomous behavior will allow the issuing CPU to continue other operations or schedule new processes. A potential issue with using the AXI HP interconnect is the possibility of introducing too much memory traffic and slowing down CPU memory accesses. Peripherals will need to be carefully designed to prevent too much memory traffic. [10]

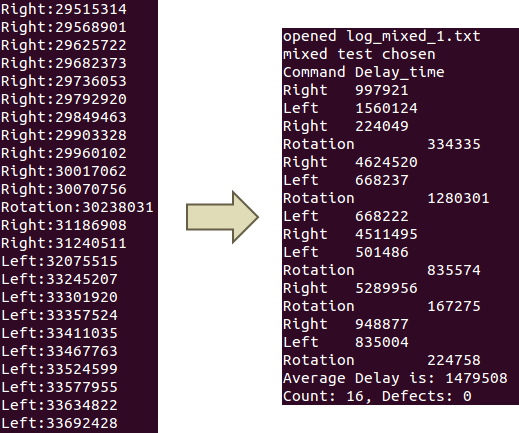
# Specification, Testing, and Results

In this section, the specification and testing results are presented in Table 1. The detailed test procedure are also discussed in this section.

Table 1 Specification and testing results

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Requirement | Specified Performance Level | Evaluation Test | Experimental Result | Met / Not Met | Comments |
| System Gesture Recognition Time | <1 second | While the game program, input gestures will be recorded to an external log file, containing both the specific command and the time at which the command was input. This file will be parsed to measure the time required to transition from two sequential actions. | Rotation: VGA - 0.408s  720p - 0.786s  Direction: VGA - 0.326s  720p - 0.733s | Met |  |
| Minimum Number of Distinct Hand Gestures | 3 | N/A | 2 distinct gestures | Not Met | Based on empirical performance of the contour defect detection algorithm, the number of hand gestures was limited to 2, separated by a specific number of defects. This prevented any inadvertent gestures that could be caused by small changes in the number of defects detected. |
| Hand Motion Detection | at least 2 directions | N/A | 2 directions | Met | The user can currently control the horizontal direction of a piece by moving their hand left or right. |
| Processed Image Resolution | 720p or higher | N/A | See comments | Met | The user can currently choose among 480p, 720p, and 1080p resolution options when running the game executable |
| User Input Recognition Range | 3-5 ft | For this test, a specific sequence of inputs will be tested at 1 foot increments in the recognition range. These inputs will then be compared with the output log to ensure accuracy from those locations. | 3-5 ft range confirmed | Met |  |
| Percent Correct Rotation Recognition | >70% | Similar to the test for System Gesture Recognition Time, a record of inputs will be passed to an external log file with respective times. This log file will then be compared with a script that contains the users inputs for the specific log. In this case, only hand gestures will be used in the script | VGA - 97%  720p - 97.6% | Met |  |
| Percent Correct Hand Motion Recognition | >70% | The same test will be perform in this case, with the only input being left and right hand movements. | VGA - 90%  720p - 93.1% | Met |  |
| Maximum Calibration Time | 45 seconds | Time will be measured from the start of program execution to the first recorded action by transferring command information to an external log file. In this case, the user will input a command immediately when the program starts. | N/A | Met | Because there were lingering issues with noise that arose from the background subtraction method, a skin filtering algorithm was implemented to isolate contours. As a result, time is no longer required to collect frames for the background model, drastically reducing the waiting time required before the user can play the game. |
| Maximum Gesture Length Time Requirement | 1 second | As input detection is dependent upon a number of frames for specific gestures, a frame rate comparison will be made to ensure that gestures are not required to take more than 1 second to complete. | Rotation:  VGA - 0.352s  720p - 0.62s  Direction:  VGA - 0.27s  720p - 0.57s | Met |  |
| Output display resolution | VGA or higher | N/A | up to 1080p | Met | We are using the HDMI port on the board for video output. |

For testing the listed performance requirements above, additional code was added to the program in order to produce an output log file. This file contains information about which commands were performed by the system and when they were performed in relation to program initialization time. Since the log will print all of the commands that the system recognizes from the user, there will be redundant information as shown in Figure 2(a). Each line is in the format of *Command: Timestamp from initialization time*. Thus, we created an interpreter which extracted the useful information from the log file to a more readable version as shown in Figure 2(b). Each line is in the format of *Command: Time lasted since the last command*. There are different versions of interpreters depending on the test schemes, which will be described in the following paragraphs.



**a**  **b**

Figure 2 a). One portion of the original log files from the system in one test trial; b) Processed log file by the mix-version interpreter.

All corresponding testing data and detailed analysis is appended in associated CD containing all available project files.

In order to test system recognition time, a series of repetitive actions was performed by the user for the game system. Motion recognition time was determined by measuring the overall time difference between left and right controls when the user repetitively moved their hand left and right in the camera’s field of view. Rotation recognition time was determined by measuring half of the time difference between sequential rotate commands while the user repetitively opened and closed their hand, as the user must open and then close their hand before sequential rotation commands are detected. We conducted 4-minute recognition time test procedure for both motion and rotation. The first set of results were produced in VGA resolution. Table 2 displays the analyzed result from the raw data. Figure 3 shows the corresponding distribution diagrams for the recognition times. As test results indicates, the data has a T-shape like distribution around the average. Hence, the average values were chosen to represents the recognition times for motion and rotation.

Table 2 Recognition Time test results summary for Motion and Rotation in microseconds with VGA resolution

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Motion | | | **Rotation** | | |
| Total Count | | 167 | **Total Count** | | 108 |
|  | **MIN** | 53525 |  | **MIN** | 28510.5 |
|  | **MAX** | 1169539 |  | **MAX** | 611632 |
|  | **Average** | 326090.51 |  | **Average** | 407745.82 |
|  | **Std** | 175632.45 |  | **Std** | 140665.66 |
|  |  |  |  |  |  |
|  | **Interval** | **Frequency** |  | **Interval** | **Frequency** |
|  | 75000 | 1 |  | 37500 | 2 |
|  | 150000 | 2 |  | 75000 | 1 |
|  | 225000 | 82 |  | 112500 | 4 |
|  | 300000 | 16 |  | 150000 | 3 |
|  | 375000 | 18 |  | 187500 | 3 |
|  | 450000 | 26 |  | 225000 | 4 |
|  | 525000 | 6 |  | 262500 | 3 |
|  | 600000 | 6 |  | 300000 | 3 |
|  | 675000 | 2 |  | 337500 | 4 |
|  | 750000 | 1 |  | 375000 | 1 |
|  | 825000 | 1 |  | 412500 | 3 |
|  | 900000 | 2 |  | 450000 | 26 |
|  | 975000 | 1 |  | 487500 | 18 |
|  | 1050000 | 1 |  | 525000 | 13 |
|  | 1125000 | 1 |  | 562500 | 15 |
|  | 1200000 | 1 |  | 600000 | 4 |
|  |  |  |  | 637500 | 1 |

Figure 3 Distribution diagram for motion and rotation recognition time test result in VGA resolution

A similar test was also conducted in 720p resolution. The results are presented in Table 3 and Figure 4. The recognition times for both motion and rotation were longer than the results in VGA resolution but expressed a very similar pattern, which was likely caused by the increased number of pixels. 1080p resolution results were not tested because the game does not function properly with this setting, largely as a result of a low processing frame rate.

Table 3 Recognition Time test results summary for Motion and Rotation in microseconds with 720p resolution

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Motion | | | Rotation | | |
| Total Count | |  | Total Count | |  |
|  | MIN | 166735 |  | MIN | 83460.5 |
|  | MAX | 1879090 |  | MAX | 3091875.5 |
|  | Average | 733263.378 |  | Average | 785978.31 |
|  | Std | 323512.78 |  | Std | 317040.35 |
|  |  |  |  |  |  |
|  | Interval | Frequency |  | Interval | Frequency |
|  | 200000 | 16 |  | 100000 | 1 |
|  | 300000 | 0 |  | 200000 | 1 |
|  | 400000 | 15 |  | 300000 | 7 |
|  | 500000 | 1 |  | 400000 | 3 |
|  | 600000 | 19 |  | 500000 | 6 |
|  | 700000 | 165 |  | 600000 | 4 |
|  | 800000 | 0 |  | 700000 | 6 |
|  | 900000 | 4 |  | 800000 | 25 |
|  | 1000000 | 0 |  | 900000 | 29 |
|  | 1100000 | 23 |  | 1000000 | 22 |
|  | 1200000 | 11 |  | 1100000 | 12 |
|  | 1300000 | 0 |  | 1200000 | 1 |
|  | 1400000 | 9 |  | 1300000 | 0 |
|  | 1500000 | 1 |  | 1400000 | 0 |
|  | 1600000 | 6 |  | 1500000 | 2 |
|  | 1700000 | 3 |  | 1600000 | 0 |
|  | 1800000 | 1 |  | 1700000 | 0 |
|  | 1900000 | 4 |  | 1800000 | 0 |

Figure 4 Distribution diagram for motion and rotation recognition time test result in 720p resolution

With the recognition time, we were able to obtain the maximum gesture length time by subtracting system execution time from the recognition time. This is because time required for one gesture to hold could be considered as a subset of the overall time for system gesture recognition time, as the recognition time is measured from the end of the last interpreted input to the interpretation of a new input. Table 4 shows the sampled execution time for both VGA and 720p resolutions. Since all the frames are processed with the same sets of functions, we assume the execution time for motion and rotation images are the same. The maximum gesture length time is calculated and shown in Table 5.

Table 4 Execution time test results in microsecond

|  |  |  |
| --- | --- | --- |
|  | **VGA** | **720p** |
| Trial 1 | 53085 | 167442 |
| Trial 2 | 56851 | 167229 |
| Trial 3 | 53787 | 167106 |
| Trial 4 | 56991 | 167505 |
| Trial 5 | 53217 | 167182 |
| Trial 6 | 56728 | 168152 |
| Trial 7 | 56898 | 167444 |
| Trial 8 | 53202 | 167567 |
| Trial 9 | 56806 | 166752 |
| Trial 10 | 57072 | 166870 |
| Trial 11 | 57113 | 166989 |
| Trial 12 | 56790 | 166875 |
| Average | 55711.67 | 167259.42 |

Table 5 Maximum gesture length time in microsecond

|  |  |  |
| --- | --- | --- |
| Maximum Gesture Length Time | | |
|  | Motion | Rotation |
| Recognition Time | 326091 | 407746 |
| Execution Time | 55712 | 55712 |
| VGA | 270379 | 352034 |
|  |  |  |
| Recognition Time | 733263 | 785978 |
| Execution Time | 167259 | 167259 |
| 720p | 566004 | 618719 |

Recognition rates were tested in a similar procedure, except with user input based on pre-produced scripts. For testing gesture recognition rate, the tester input a specific number of rotation commands. This value was then compared to the empirical number of rotations performed in the game. For motion recognition, the tester input a series of left and right commands in a particular order. Again, this script was compared with empirical output in the log file to determine an appropriate recognition rate. The log file is processed by different versions of the interpreter. In the motion test, the interpreter will only accept direction command and regard rotation as false detection, vice versa in the rotation test. Table 6 and Table 7 present the results for both VGA and 720p resolution, respectively. The substantial increase in the number of false detection in rotation test is because whenever there is a close and open motion, the center-of-mass of the hand will move slightly, which is recognized as a motion command by the system. However, because this motion is so small that the user will not experience any changes in the game. In order to ensure limited influence of motion commands with gestures commands, tests were also performed on a combined script with both motion and rotation commands. The test result is shown in Table 8. All of these tests achieved a recognition rate greater than 90%, significantly exceeding the minimum requirement of 70%.

Table 6 Maximum gesture length time in microsecond

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Motion Recognition Rate Test | | | | |
| VGA | Scripts | Motion | Error | False Detection |
| Trial 1 | 34 | 37 | 3 | 7 |
| Trial 2 | 34 | 41 | 7 | 13 |
| Trial 3 | 34 | 38 | 4 | 7 |
| Trial 4 | 34 | 35 | 1 | 2 |
| Trial 5 | 34 | 36 | 2 | 8 |
| Average | 34 | 37.4 | 3.4 | 7.4 |
| Recognition Rate | | 90.0% | |  |
|  |  |  |  |  |
| Rotation Recognition Rate Test | | | | |
| VGA | Script | Rotation | Error | False Detection |
| Trial 1 | 14 | 15 | 1 | 46 |
| Trial 2 | 14 | 14 | 0 | 192 |
| Trial 3 | 14 | 15 | 1 | 21 |
| Trial 4 | 14 | 14 | 0 | 13 |
| Trial 5 | 14 | 14 | 0 | 493 |
| Average | 14 | 14.4 | 0.4 | 153 |
| Recognition Rate | | 97.1% | |  |

Table 7 Recognition rate test results in 720p resolution

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Motion Recognition Rate Test | | | | |
| 720p | Script | Motion | Error | False Detection |
| Trial 1 | 34 | 32 | 2 | 16 |
| Trial 2 | 34 | 30 | 4 | 10 |
| Trial 3 | 34 | 35 | 1 | 6 |
| Average | 34 | 32.33 | 2.33 | 10.67 |
| Recognition Rate | | 93.1% | |  |
|  |  |  |  |  |
| Rotation Recognition Rate Test | | | | |
| 720p | Script | Rotation | Error | False Detection |
| Trial 1 | 14 | 14 | 0 | 138 |
| Trial 2 | 14 | 14 | 0 | 126 |
| Trial 3 | 14 | 13 | 1 | 118 |
| Average | 14 | 13.67 | 0.33 | 127.33 |
| Recognition Rate | | 97.6% | |  |

Table 8 Combined recognition rate test in VGA resolution

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Script** | **Trial 1** | **Trial 2** | **Trial 3** | **Trial 4** |
| **L** | L | L | R | L |
| **R** | R | R | O | R |
| **O** | O | O | L | O |
| **R** | R | R | R | R |
| **L** | L | L | O | L |
| **O** | O | O | MISS R | O |
| **L** | L | MISS L | L | L |
| **R** | R | R | O | R |
| **L** | L | L | L | L |
| **O** | O | O | R | R |
| **L** | MISS L | MISS L | L | O |
| **R** | R | R | O | L |
| **O** | O | O | L | R |
| **L** | R | R | R | O |
| **O** | L | L | O | L |
| **R** | O | O | L | O |
| **L** |  | R | R | R |
| **R** |  |  | O | L |
| **L** |  |  | R | R |
| **R** |  |  | L | L |
| **O** |  |  |  | R |
| **R** |  |  |  | O |
| **O** |  |  |  | R |
| **L** |  |  |  | O |
|  |  |  |  | L |

For user input range testing, the scripts used for recognition rates were utilized to perform testing at 1 foot increments in the 3 feet to 5 feet range. With the same procedure, it was ensured that at least a 90% recognition rate was achieved for each testing distance. The result for each test in VGA resolution is shown Table 9.

Table 9 Range test result in VGA resolution

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Motion Range Test | | | | |
| VGA | Script | Motion | Error | False Detection |
| 3 ft | 23 | 23 | 0 | 0 |
| 4 ft | 23 | 23 | 0 | 1 |
| 5 ft | 23 | 23 | 0 | 1 |
| Recognition Rate | | 100.00% | |  |
|  |  |  |  |  |
| Rotation Range Test | | | | |
| VGA | Script | Rotation | Error | False Detection |
| 3 ft | 15 | 14 | 1 | 32 |
| 4 ft | 15 | 14 | 1 | 605 |
| 5 ft | 15 | 14 | 1 | 6 |
| Recognition Rate | | 92.86% | |  |

## Adjustments

As seen in the requirements table, a majority of the constraints for this project were met. The one requirements not accomplished by the application was the minimum number of distinct hand gestures. Our specification for this listed a minimum value of three gestures. However, the current implementation of the motion-controlled game only provides a system response for two gestures: If the user is holding up less than four fingers or if the user is holding up more than four fingers. This result is largely based on the empirical performance of the contour defect detection algorithm used to determine the number of fingers on an identified hand. The decision to limit user gesture recognition to two distinct gestures prevents any inadvertent commands due to small changes in the number of contour defects detected.

## In Retrospect

Given the current status of the requirements table, it would have been beneficial to add a requirement pertaining to hardware acceleration speed-up factor. This would have assisted us in the decision regarding which OpenCV functions we would like to accelerate, as we would have been able to compare the execution time in software with the time required to transfer image data from software to hardware. However, this decision could not have been made at the end of last semester, due to lack of experience with the ZedBoard and the Zynq chip. If a new project was developed based on the same platform, a new constraint pertaining to acceleration would be included in the requirements table.

Furthermore, although the maximum calibration time requirement was relevant for previous iterations of the project, it no longer pertains to the final implementation. This is because the calibration time constraint is only applicable to cases where a minimum number of input frames are required to produce a corresponding background model for the environment. Since a background subtraction method is no longer used, there is no longer a minimum calibration time before the user can start playing the game.

# Design Evolution

## System Architecture

As previously described in the problem overview, our system contains two main sections: a software implementation, and a hardware accelerated component on the FPGA fabric. In the following discussion, both sections are presented in each system version.

In the course of project development, the system underwent one major revision after panel 5 which revealed major flaws critical to reasonable performance of the system. This revision was able to improve the performance from 1-2 fps to 16 fps in the software implementation. It also made it possible for hardware accelerator show 3.5X acceleration factor on one of the OpenCV functions. The detailed system evolution decisions are discussed in this section.

### System - version 1

In order to use appropriate code to identify the user’s hand movements and gestures, we implemented several algorithms that were discovered in the research performed in the previous semester. This implementation allowed us to acquire a responsive motion-controlled game fully capable of running on a PC, but with limited performance capabilities on the ZedBoard.

#### Software Implementation

In the initial software version of this project, two main algorithmic methods were implemented in order to isolate the hand and track its motion: background subtraction and optical flow. As a high-level description, an optical flow algorithm will track the locations of a discrete number of points between frames. In the context of this application, the OpenCV function cvCalcOpticalFlowPyrLK() was used in order to attain the magnitude and direction of the change in distance for various points on the user’s hand in a frame. An image overlayed with vectors to portray the output of this particular algorithm may be seen in the Figure 5 below.

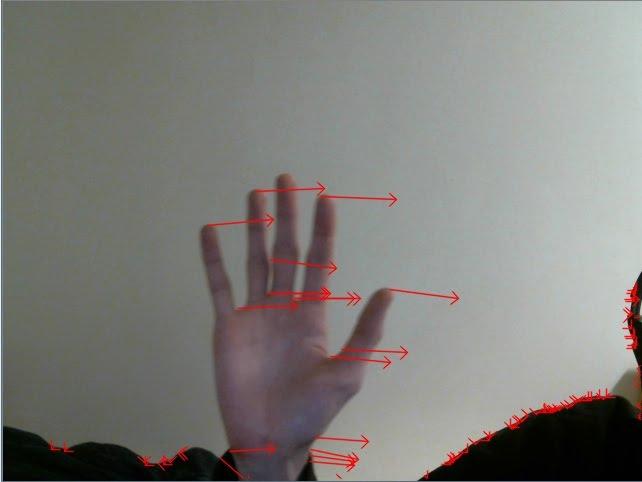


Figure 5 Depiction of optical flow vectors overlaid on source image

In order to recognize particular motions, the tail ends of a minimum number of these vectors must lie in a particular area of the camera’s field of view. In other words, if the user wants to move a piece in the right direction, the user’s hand must pass a particular threshold on the right side of the camera’s field of view. The disadvantages of this particular algorithm were related to both functionality and performance. For proper game behavior, the user’s hand was required to be in the center of the camera’s field of view. If the user’s hand was not centered, then inadvertent commands at the beginning of program execution were likely.

For this initial iteration of software, a background subtraction method was implemented to isolate the hand from non-moving background elements. Background subtraction was performed by first developing a background model for the environment before the user was allowed to control the game. In the prototype demonstrated in panels 4 and 5, 100 frames were collected and used to create a background model at the beginning of program execution. After developing this model, all incoming frames from the webcam were masked by this model to create a binary image where all high intensity pixels represented foreground elements. A small size subtraction method was then used to eliminate small instances of stray noise. A depiction containing a masked image and the corresponding noise elimination filter may be seen in the Figure 6 below.



Figure 6 Binary Mask generated by background subtraction method (Left) and Mask after small size subtraction (Right)

In terms of functionality, the decision to use a background subtraction model led to inadvertent performance. If the user moved their body while playing the game, extra noise was interpreted in the image processing flow. This led to both undesired rotational commands and a lack of responsiveness to horizontal movement by the hand. These issues were induced because the initial background subtraction method was non-adaptive, meaning that the model was only based off of the frames collected during the calibration phase of the program. Because of this, any changes made to the surrounding environment resulted in new objects introduced into the foreground of the image.

Overall, in terms of software performance, the version of the project demonstrate during panel 5 exhibited significant issues pertaining to the frame rate of the motion-controlled game output. These delays largely occurred due to the time required to perform optical flow calculation and analysis on each generated vector output from the algorithm. As shown during the panel, the resulting prototype operated with a throughput of 1-2 frames per second at VGA input resolution. This made the game highly difficult to play. From empirical testing the latter stages of the project, we determined that a minimum frame rate of 5 fps was required in order to achieve a playable state.

Another problem encountered in the progression of the project was the possibility of accelerating chosen OpenCV functions with the programmable logic. This problem pertains to the OpenCV functions utilized for background subtraction, as well as the main function used for optical flow pattern recognition, cvCalcOpticalFlowPyrLK(). Although research was performed on FPGA implementation for each of these algorithms, the difficulty of implementation was not researched to a sufficient extent. This resulted in the inclusion of expensive algorithms in the image processing flow that could not be converted to an appropriate accelerator, given the time constraints of project development.

#### Hardware Implementation

In the partial hardware implementation, the cvContourArea function was implemented as a peripheral in the programmable logic. This cvContourArea peripheral was controlled using a linux device driver. However, through testing it was discovered that the software performance of the cvContourArea() function was already sufficiently fast. As a result, the overhead required to copy contour data to the programmable logic and back to the processing system was too high, resulting in a deceleration factor. Ultimately this meant that the system performance remained too low to create a playable experience.

## System - version 2

Based on the undesirable performance and problems experienced with the previous version of the implementation, improved methods to detect the hand and corresponding controls were researched and implemented. These new functions also had an overall goal to reduce the image processing time required for each frame.

### Software Implementation

In order to resolve the aforementioned performance issues with the initial system version, the decision was made to remove the optical flow and background subtraction algorithms. Both of these methods introduced high execution times to the image processing flow, with minimal functional gain. In place of these algorithms, a skin detection algorithm and center of mass tracking were implemented to support motion and gesture detection. The use of skin detection improved the reliability of the system by allowing the system to track the user’s hand even if the user moves within the frame. This is an improvement over the first version of the system, which would attempt to track the user, rather than their hand, if they moved within the frame. Furthermore, the previous version of the system incorrectly detected motion commands if the user’s hand was not initially placed in the center of the camera’s field of view. The skin detection method chosen is, however, unreliable in certain lighting conditions and in environments with red or brown tones in the background.

Motion detection is achieved by calculating the change in center of mass of the convexity defects, used to determine the hand gesture of the user, between frames. This replaces the optical flow algorithm used in the first version of the system and is significantly less computationally expensive, resulting in a much higher frame rate. Because the the game is designed to take place with a static background, the center of mass method used does not result in any loss of functionality. In fact, the method implemented achieves more stable results due to the occurrence of stray, high-magnitude vectors when utilizing optical flow.

#### Hardware Implementation

The functions chosen for hardware implementation in this version of the project were the cvErode and cvDilate functions. These were selected since they consumed a significant portion of the processing time in the software implementation. Also, morphological erosion and dilation have been implemented in hardware on FPGAs previously with speed-up factors.

Specifically, these accelerators were implemented using Vivado HLS. These accelerators accessed memory using a memory-mapped interface, as opposed to a streaming interface. This introduced some limits on how data could be read from and written to the frame buffers in HLS, however, we were unable to successfully implement accelerators that utilized a streaming bus interface.

These hardware implementations attempted to increase the performance of their respective software implementations by increasing the level of parallelization of the algorithms. This was achieved by reading portions of the frame being processed into temporary buffers in the programmable logic. Multiple portions of these temporary buffers were then accessed in parallel to achieve a faster processing rate than the ARM processor could achieve alone.

## Final Implementation Details

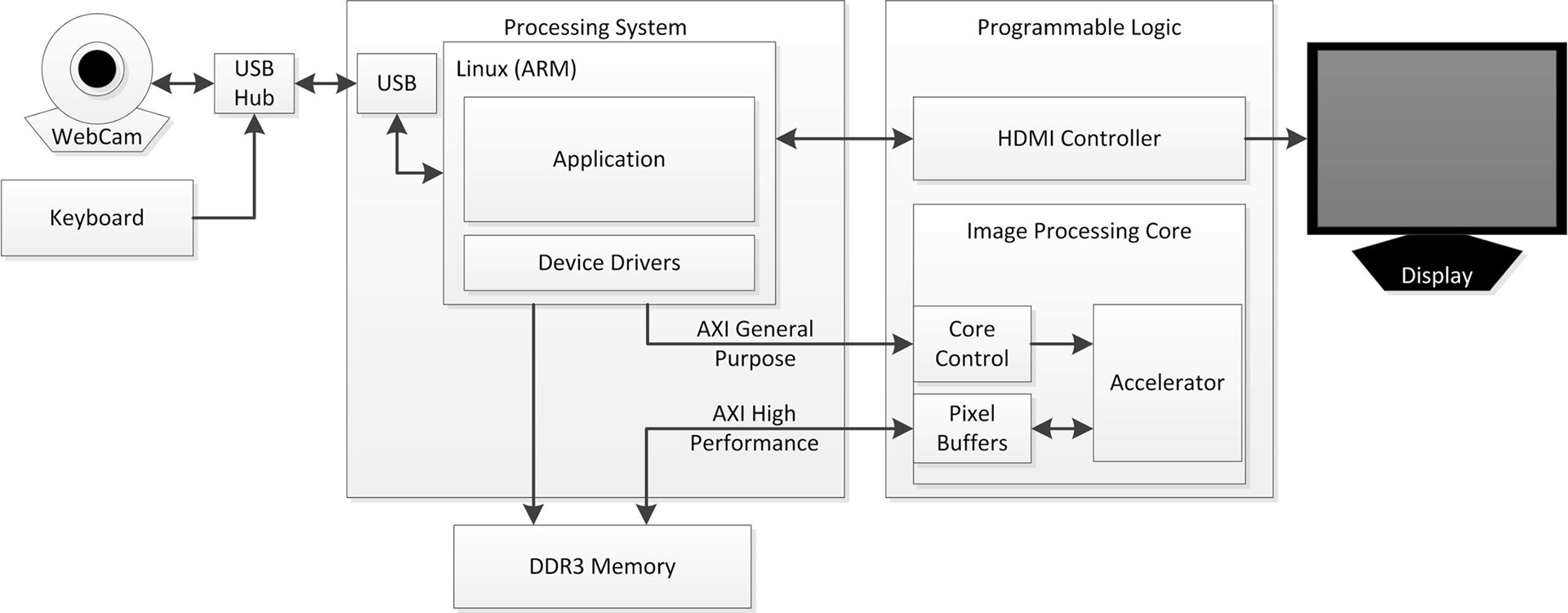


Figure 7 Block diagram for entire system

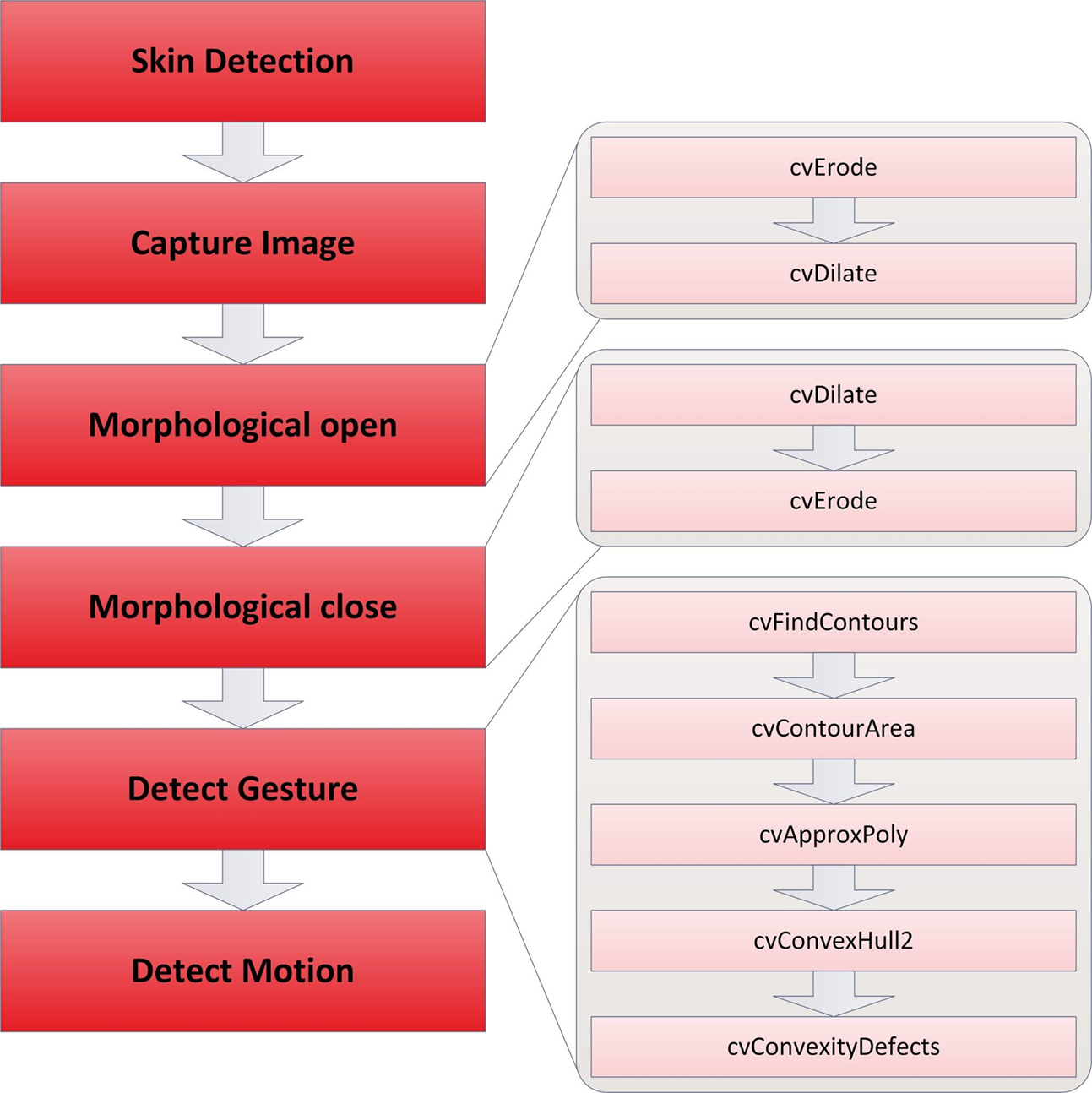


Figure 8 Image processing flow with corresponding OpenCV functions

In this final software implementation of the project, each frame read into the Zedboard from the USB webcam is initially sent through a skin detection filter. Based on the v4l function calls used for webcam input, each frame is stored in a YUV color space. The blue-difference (Cb) and red-difference (Cr) chroma component channels of the YUYV image are then used to compare each pixel in a given frame to predefined values that estimate the range of skin color. If the pixel values for a particular pixel fall within the range for skin color, then that pixel is identified as skin in the frame. This comparison is performed for each pixel in a given input frame. From the information collected, a binary image is created, where a white pixel indicates skin and a black pixel indicates a pixel that falls outside of the skin color range. All other operations in the image processing flow are performed on this generated image.



Figure 9 Example of skin filter used for isolating a hand contour

In order to prevent undesired noise from influencing the skin-filtered frame, morphological open and close operations are performed. In OpenCV, these operations are performed by calling the cvErode() and cvDilate() functions on a particular image. An implementation of this procedure can be seen block diagram above. In the context of this application, erosion and dilation play a role in joining disparate elements in an image, such as the potential separation of a finger with a ring. These functions also ensure separation of individual elements that may be located in close proximity to one another. For the motion-controlled game, this helps ensure that two fingers close to one another are not identified as a single element.

After the source image has been filtered for skin and been passed through morphological open and close modules, the frame can be tested for a potential gesture and information can be collected pertaining to the relative position of the user’s hand. As an initial step to this process, the contours of all objects in the frame are determined by calling the cvFindContours() functions. This function returns pointers to all available contours, in terms of their vertices, in a single-channel image. Next, the area of each contour is determined using the cvContourArea() function. For our application, the two largest contours in the frame are isolated for analysis. In terms of the use scenario, these two contours should represent the user’s hand and the user’s face.

Based on the use scenario for the project, the rotation of each piece in the game was designated to be controlled by the user’s hand gestures. Specifically, rotation is performed if the user holds up more than four fingers. In order to determine this from the filtered image, the number of defects associated with a contour is calculated through the use of the cvConvexityDefects() function. As the user’s hand is expected to be the largest skin-colored element in the camera’s field of view, the number of convexity defects for the largest contour in the frame is analogous to the number of fingers being held up by the user. Thus, if there are more than four convexity defects, a rotation command is input from the user. In order to prevent inadvertent rotation from occurring, the user’s hand must close before rotation can occur again. In this implementation of this logic, the number of convexity defects must pass below four defects for three consecutive frames before the game will accept another rotation command.

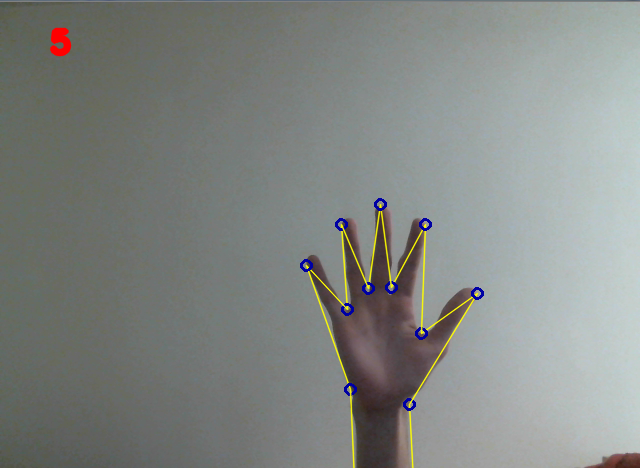


Figure 10 Sample output of contour defect detection algorithm overlaid on source image

For motion recognition, the user can control the horizontal location of a piece by moving their hand left or right. This was implemented in software by first calculating the center of mass of the hand, based on the largest detected contour in the frame. This center of mass calculation is performed on each frame that passes through the image processing pipeline. If the difference in the x-coordinate for the hand’s center of mass exceeds a particular threshold, the system detects a right or left command from the user, dependent upon the direction of hand motion. Similar to rotation, there is a designated time delay for motion commands so that excessive sensitivity does not become an issue with lower resolution input frames, which results in a higher throughput frame rate.

In addition, the final implementation of this project contained refined versions of hardware accelerators for the cvErode() and cvDilate() OpenCV functions. A linux device driver was written to control the accelerators from userspace. The device driver supported system ioctl function calls on the accelerators for configuring frame resolution parameters and starting the accelerated operation. The device driver also supported system read function calls. Reads on the device would block until the an interrupt was received from the accelerator by the interrupt handler, signifying operation completion. The performance of these accelerators is listed in Table 10.

Table 10 Performance results are OpenCV functions in software and hardware

|  |  |  |  |
| --- | --- | --- | --- |
| 720p | Software | Hardware | Speedup Factor |
| cvErode | 40400us  24fps | 11600us  86fps | 3.48 |
| cvDilate | 40100us  25fps | 62600us  16fps | 0.64 |

In order to display the game to the user, a connection was made to a monitor via the HDMI port available on the board. The ncurses library was used to provide the graphical capabilities required to produce colored blocks for game pieces. The use of ncurses also ensured that a limited number of third party libraries were required to run the game properly. An image of the game output can be seen in the Figure 11 below. A window is provided in the bottom section to the screen to indicate which object is being tracked in the input frame and provide an estimate of the camera’s field of view to the user.



Figure 11 Output from ZedBoard via HDMI for game display

# Bill of Materials

Table 11 Final Project Cost

|  |  |  |  |
| --- | --- | --- | --- |
| Item description | Cost | Count | Total |
| Logitech C920 Camera | $74.99 | 1 | $74.99 |
| 8GB SD Card | $23.74 | 1 | $23.74 |
| ZedBoard | $0 (donated by Avnet) | 1 | $0 |
|  |  | Total Product Cost | $98.73 |

Table 12 Total Project Cost

|  |  |  |  |
| --- | --- | --- | --- |
| Item Description | Cost | Count | Total |
| Final Product Cost  (see table above) | $98.73 | 1 | $98.73 |
| 8GB SD Card | $23.74 | 3 | $71.22 |
|  |  | Total Product Cost | $169.95 |

## Discussion

Overall, the total cost for this project was much lower than the total budget of $600. A significant reason for this was the donation of three ZedBoards to Bucknell University from Avnet for the purposes of this project. The retail price for a ZedBoard from Avnet is $395. Without the cost of the ZedBoard, the only required purchases included a webcam for camera input and replacement SD cards to store boot files and project files. These replacement cards were required because of damage done to the stock SD cards from the mechanical stress associated with removal and insertion of cards between PC workstations and the development board.

Therefore, the price for the final product involves only the cost of the webcam and a single SD card. The cost of the total project includes the cost of the final product, as well as three additional SD cards that were used in order to expedite development of the project by utilizing multiple ZedBoards simultaneously.

Summarily, this particular project would not have had significant increases in performance with an increased budget. The only gains that could have been acquired involve the use of an improved camera. However, this would have not met Avnet’s request that the project utilize an affordable camera, ideally under $100.

# In Retrospect

In the development of the project, several delays were caused by software performance and functionality. Overall, these problems could have likely been resolved by performing more extensive research prior to beginning to implement several prospective methods and algorithms. Additional research would have likely resulted in a determination that specific algorithms, including the optical flow algorithm, would not have been suitable candidates for hardware acceleration, and thus would not likely contribute to the designated goals of the project. Furthermore, additional research would have indicated a minimal functional increase with the inclusion of more complex algorithms. Because our use scenario dictates a fairly uniform background environment, a skin filter served a similar function as the background subtraction method, while maintaining a significantly lower execution time per frame. Similarly, the tracking of the center of mass of particular contours in the frame mirrored the performed of optical flow when using a simple surrounding environment. In general, the optical flow procedure would have only presented significant advantages if the surrounding environment was expected to have multiple objects moving simultaneously in the camera’s field of view.

Another lesson learned throughout the semester was the importance of communication with the client. During the semester, more knowledge was acquired concerning deliverable expectations. For example, about halfway through the semester, Avnet relayed more evidently that the desired project would demonstrate the availability and effectiveness of community tools produced by both Avnet and Xilinx for the board. They also desired production of tutorials that could be used in the future by other individuals who wished to complete projects using the ZedBoard. If this information was more perceptible in the initial stages of the project development cycle, a more extensive use of available resources would have occurred in order to reflect the usefulness of the ZedBoard over its competitors. Increased communication would have also limited the number of issues that arose when trying to transfer project documents to Avnet. In order to increase this communication, more regular meetings should have been scheduled. Throughout the course of the semesters, meetings were generally held in the post-panel weeks. A more efficient method may have been to hold bi-weekly meetings. This would have ensured that project goals followed Avnet’s vision for the project and answer any potential questions that either party had in relation to project development.

Furthermore, a more descriptive schedule should have been created and modified throughout the course of the semester in order to streamline of the development of the project. Although there was an initial schedule developed at the beginning of the semester to cover respective duties up to the first panel, vague assignments for the latter part of the semester interrupted progress. In retrospect, the schedule should have been reevaluated at frequent intervals in order to ensure that proper resources could be allocated to more strenuous portions of the application. Specifically, this would have assisted with software development, when performance issues prevented consistent game behavior. A more detailed schedule would have also assisted with hardware development, when development of accelerators for cvErode() an cvDilate() began. It became evident in the final stages of development that other hardware architecture decisions could have yielded better performance results.

If more time was allotted for the continued development of the motion-controlled game, significant improvements would be made to the image processing flow in order to ensure a version that is more capable of optimal performance in a wider range of environments. Specifically, more adjustments could be made to the existing skin filter to compensate for false recognition of dim lighting as skin tone. This includes the potential to utilize a probabilistic model to more accurately identify skin-colored objects in the input frame. Additional time may also be used to develop an adaptive background subtraction model to work in coordination with the currently implemented skin-filter. This would serve the purpose of eliminating potential skin tones in the surrounding environment and provide another solution to the problems encountered during panel 6.

Given additional time for development, the hardware accelerators would be rewritten to take advantage of the Vivado HLS implementations of the cvErode and cvDilate functions. These implementations have a proven functionality and acceleration capability. These HLS functions were not utilized in the final iteration of the project due to an inability to pass image data to their streaming interfaces. Furthermore, the tutorial documents for implementation of these functions were not released until approximately one week before panel 6. If more time were available, a reliable method for controlling a memory mapped to stream interface converter could be developed, allowing the project to have software control of accelerators with streaming interfaces. This would also provide an additional example to demonstrate the value of the community tools provided by Xilinx, a key overall goal of the project.

Clearly, there are many steps that could have been taken to improve the end result of the implementation of a motion-controlled block game. In future projects, these learnings will certainly be utilized in order to prevent potential barriers to progress and expedite project development. Nonetheless, the end result application was able to demonstrate the key goals of the project in the following ways:

1. Implementation of an image processing application using the Zynq chip
2. Acceleration of software functionality through the use of programmable logic
3. Extensive use of community tools to decrease required research time for project development

This progress would not have been possible without the contributions of Professor Watkins, Professor Thompson, and Professor Cheville. There were also considerable contributions and assistance from individuals as Avnet and Xilinx, including Jim Beneke, Mario Bergeron, Luc Langlois, and Stephen Neuendorffer.

# References

[1] “OpenCV.” Internet: http://opencv.willowgarage.com/wiki/Welcome/OS, Sept. 26, 2011 [Oct. 13, 2012].

[2] Junguk Cho, Shahnam Mirzaei, Jason Oberg, and Ryan Kastner. 2009. Fpga-based face detection system using Haar classifiers. In Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays (FPGA '09). ACM, New York, NY, USA, 103-112. doi: 10.1145/1508128.1508144 URL: http://cseweb.ucsd.edu/~kastner/papers/fpga09-face\_detection.pdf

[3] Changjian Gao; Shih-Lien Lu; , "Novel FPGA based Haar classifier face detection algorithm acceleration," Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on , vol., no., pp.373-378, 8-10 Sept. 2008 doi: 10.1109/FPL.2008.4629966 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4629966&isnumber=4629890

[4] Lee, P.; Evagelos, A.; , "An implementation of a multiplierless Hough Transform on an FPGA using hybrid-log arithmetic," Proc. SPIE 6811, Real-Time Image Processing 2008, 68110G (February 26, 2008); doi:10.1117/12.766459. URL: http://144.206.159.178/ft/CONF/16408621/16408634.pdf

[5] Tagzout, S.; Achour, K.; Djekoune, O.; , "Hough transform algorithm for FPGA implementation," Signal Processing Systems, 2000. SiPS 2000. 2000 IEEE Workshop on , vol., no., pp.384-393, 2000 doi: 10.1109/SIPS.2000.886737 URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=886737&isnumber=19159>

[6] Ratnayake, K.; Amer, A.; , "Sequential, Irregular and Complex Object Contour Tracing on FPGA," Image Processing, 2007. ICIP 2007. IEEE International Conference on , vol.5, no., pp.V-165-V-168, Sept. 16 2007-Oct. 19 2007 doi: 10.1109/ICIP.2007.4379791 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4379791&isnumber=4379738

[7] Raj, S.M.A.; Sreelatha, G.; Supriya, M.H.; , "Gesture recognition using field programmable gate arrays," Devices, Circuits and Systems (ICDCS), 2012 International Conference on , vol., no., pp.72-75, 15-16 March 2012 doi: 10.1109/ICDCSyst.2012.6188677 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6188677&isnumber=6188639

[8] Crampton, Stephen C. “Counting Fingers in Real Time Using Computer-Vision Techniques.” B.A., Middlebury College, 1986. URL: http://www.cs.bu.edu/gradprogram/thesis/SteveCrampton.MA.thesis.2004.pdf

[9] ARM. 2007. PrimeCell DMA Controller (PL330). r0p0. Technical Reference Manual URL:<http://infocenter.arm.com/help/topic/com.arm.doc.ddi0424a/DDI0424A_dmac_pl330_r0p0_trm.pdf>

[10] Xilinx. Nov. 16, 2012. Zynq-7000 All Programmable SoC (UG585). v1.4. Technical Reference Manual URL: http://www.xilinx.com/support/documentation/user\_guides/ug585-Zynq-7000-TRM.pdf