Sheffield Hallam University

SHARPENS YOUR THINKING

SHEFFIELD HALLAM UNIVERSITY

ENGINEERING PROGRAM

FPGA BASED REAL TIME EYE TRACKING SYSTEM

FINAL REPORT

Submitted by

B.A. SAMEERA SANDARUWAN

Supervised by

Mr. Kalyanapala Marakkalage

M. ENG. (HONS) IN ELECTRONIC ENGINEERING

MODULE: 16-6080

November, 2015

Preface

This report describes project work carried out within the engineering program at Sri Lanka Institute of Information Technology University between March and November 2015.

The submission of the report is in accordance with the requirement of the degree of Electronic Engineering (MEng) under the guidance of the University.

Acknowledgement

At the end of a successful journey seeking to bring to light a FPGA based Eye tracking system based on concepts which were completely novel to us, it is with the utmost pleasure that we acknowledge the contribution of all individuals who has being supportive in making this project a success.

I would like to take this opportunity to express my deep sense of gratitude and profound feeling of admiration to my project supervisor and our project coordinator of final year projects, Mr. Kalyanapala Marakkalage. Many thanks go to him who helped me in this work and under whose supervision that we gained a clear concept of what I should do. My special thanks to Sri Lanka Institute of Information Technology for giving an opportunity to carry out this individual project. I would like to gratefully acknowledge our Head of Electronic & Computer Engineering department, Dr. Lasantha Seneviratne, for his guidance throughout the course. I would also like to thank all the lecturers and staffs in SLIIT who gave their support and my heartiest thanks to my parents for their great concern and care. Finally I like to thank my college mates and friends for their ideas and help.

Abstract

This this project a FPGA based real-time eye tracking system will be implemented based on algorithms which will be tested on Matlab. The mentioned system will be implemented with a FPGA development board and a camera module. The system will track user's eyes and output a corresponding indication to the output display. The real-time video stream also will be on the same output display. The main aspects of FPGA based eye tracking systems are the accuracy, speed, robustness, power consumption and cost. In this project my main objective is to implement a system with high accuracy.

Index terms – Eye tracking, FPGA, HDL, image processing, real-time

Table of Contents

Table of Figures

Table of Tables

Abbreviations

FPGA	- Field Programmable Gate Array
HCI	- Human Computer Interface
HDL	- Hardware Description Language
MCT	- Modified Census Transform
SVM	- Support Vector Machines
KNN	- k nearest neighbor

1. Introduction

The present world is moving to a place where, the physical control of the things around us is no longer there. With the emergence of new technologies, humans are trying to control the things around us by not associating physically. That means we are trying to interact with the digital equipment the way we interact with other humans such as voice, hand and body gestures, head and eye movements.

In the last few years, these non-physical interactions have made their way to the consumer market. Those technologies are Voice recognition & control and hand gesture recognition & control. These technologies have come to a very good place in the consumer market. But the one most important thing that isn't their yet is the eye movement detection & control technology.

Eye tracking is not a new technology. It's been there at the research level for few years. But now it's starting to make its way to the consumer market. Most of those systems are either fully software based systems or a combination of a hardware unit and a software.

The system that will be implemented as my project is going to be a fully hardware based system (FPGA). There are few advantages and disadvantages of making a fully hardware based system. The advantages are, it's going to be much faster (real-time) than the current consumer products, it's going to be accurate and the power consumption will be much lower. The main disadvantage is the system won't be much flexible as a software based system.

The system that will be implemented as my project will be a FPGA based system that will track the users eyes in real-time by using a single camera module. The output will be a two indicators that will show the position of the eyes on a display.

1.1 Literature survey

The problem of automatic face detection has been an active area of study for some time and numerous methodologies exist. These may he broadly categorized as appearance-based, feature-based or color-based. Appearance-based approaches [1,2] treat face detection as a recognition problem and, in general, do not utilize human knowledge of facial appearances, but rely on machine learning and statistical methods to ascertain such characteristics. Feature-based methods detect faces by detecting features such as the eyes and the nose, or parts or combinations thereof [3,4], and usually rely on low-level operators and a priori heuristic rules of the human face structure. Finally, color-based methods may be viewed as a special case of feature-based methods, where the feature in question is the color of the human skin [5,6]. The advantage of color-based face detection methods is that, while computationally efficient, they are generally robust to geometric transformations, such as scale, orientation and viewpoint changes, since such transformations do not affect the color of skin, as well as to complex backgrounds and illumination variations. Our method also adopts the skin color-based detection approach and can be summarized as follows.

The first step is the subsampling of each video frame, by dividing it into non-overlapping blocks of a fixed size and averaging the values within each block to produce a single pixel in the subsampled frame. The aim of this process is twofold. First, it reduces the amount of data for the subsequent processing. Second, it results in the generation of larger skin patches which decrease the susceptibility of the system to features such as a moustache or spectacles. Clearly, the subsampling factor must be chosen carefully so as to strike a balance between computational efficiency and detection performance.

The second main stage of the face detection and tracking method is color-based skin pixel detection in a video frame. Here, this detection relies on a statistical histogram-based skin color model. More specifically, a statistical skin color model has been created offline, during a training phase, by projecting manually segmented skin pixels onto a histogram. The training database encompasses the skin tones of different people and of different ethnicities, e.g. Caucasian, Asian, and African, etc.; captured under various illumination conditions. During skin detection, an unknown pixel is projected onto the histogram to retrieve a skirl color probability value from the appropriate bin, which is then used in conjunction with a threshold to decide the classification of the pixel as skin or non-skin.

With such filtering approach it common practice to convert the original RGB color space to an alternative color space for the creation of the skin color model and the subsequent filtering. Desirable features of such a color space include dimensionality reduction for a more efficient implementation (e.g. 2D as opposed to the 3D RGB space), robustness to illumination variations, and the clustering of the skin color of different people and with different ethnicities in one or more compact regions Thus, a number of appropriate color spaces have been proposed in the literature, including the normalized rg space, the UV components of CIE LUV, and the HS components of HSV, among others. In this work, our skin color filter utilizes a 2D adaptation of the 3D log-opponent color space IRgBy which has been successfully used for skin detection [6], termed Log(RG) for simplicity and defined as

$$\begin{gathered} \mathrm{L}_1 = L(\mathrm{G}) + L(\mathrm{RGB}_{\max}) \\ \mathrm{L}_2 = L(\mathrm{R}) - L(\mathrm{G}) + L(\mathrm{RGB}_{\max}) \\ \text{with } L(x) = \log_{10}(x+1) \cdot 10^6 \cdot 2^{-15} \\ \text{and } \mathrm{RGB}_{\max} = 255 \text{ for 24-bit colour} \end{gathered} \tag{1}$$

Assuming a 24-bit color depth for the RGB frame, the relations of (1) also include scaling to give each of the two Log(RG) values an 8-bit integer representation. Note that, with this color space, the 13 component of the video signal is not used. Extensive experimentation has shown us that the elimination of the B plane, which has the least contribution to skin color under the most common illuminants, has an insignificant impact with regards to skin filtering compared with using the complete 3D log-opponent color space. This elimination greatly simplifies the associated implementation, and with the additional advantage that only the R and G planes of the original video signal require sub sampling. Despite of the robustness of the chosen (or indeed any) color system to illumination changes and the inclusion of various skin tones into the statistical model, color-based skin detection can produce sub-optimal results for different illuminations and people when a static model is employed. Thus, what is needed is an adaptive model, which can be calibrated to different people and to different illuminants. Bete, this process is user initiated and relics on LogRG-converted pixels of the original frame, which are contained in a small predefined region of fixed size. The current skin color model and the new calibration data are then fused to produce an adapted model. The exact mechanism of this will be explored later. Obviously, the user initiating the adaptation process will be aware of the size and location of the predefined region where his/her skin should appear for a successful calibration. Any non-skin pixels accidentally contained in the calibration area can be eliminated using the simple rule R>=G and R>=B, which we have found to be true for skin under most illuminants.

1.1.1 Previous work

Baluja et al. [7] advocate artificial neural network for eye tracking. In this system one major key point is the number of eye images that are required to train the neural network. Likewise every edge is dealt with freely, which brings about handling of repetitive information. Smith and Pitas [8] use shading predicates to find face and the eyes utilizing minima examination. Because of high affectability to lighting conditions, this system, on the other hand, is not vigorous. Kim, Jung et al. [9] suggested FPGA based parallel architecture for real-time eye detection. For the implementation a combination of Modified Census Transform (MCT) and Ada-Boost algorithm was used which was proposed by I. Choi and D. Kim.

Ando, Koji et al. [10] implemented a low power FPGA based eye tracking system. In the process, a new method with seven-segment rectangular template to track between-the-eyes point (BTE) was used. This was a much simpler method which didn’t require computationally expensive tools such as Ada-Boost, SVM, Neural Network, etc. Through their simpler approach the low power system was implemented successfully.

Frank, Peter et al. [11] approached the problem in a different way. Their main target was to remove the burden of calibration. But to overcome that problem, the solution they used was use two cameras as input sources. For the two input sources, two processing FPGA units were used which made the system much faster. The only problem is that it would be hard to implement this into a portable device in the future.

1.1.2 Main platforms

Eye tracking systems can be implemented in several different platforms.

1. Fully software based systems
2. Combination of hardware and software
3. Fully hardware based systems
4. Programmable hardware based systems

Completely programming based frameworks require some sort of broadly useful PC to do their calculations. Be that as it may, each universally useful PC is not quite the same as each other. So despite the fact that the first implantation had amazing results, we can't expect the same results from each frameworks that the product is utilized. All the general purpose computers we used presently are designed to handle a lot of multitasking. But the eye tracking software's need a lot of computational power constantly to execute their complex image processing algorithms. If these software's didn't get the power it needs, the software might get stuck which means they aren't much reliable.

The commercially available eye tracking systems [12] are implemented with both a hardware unit and a software part. These hardware units are implemented using IR sensors which consume a lot of power. Then need some calibration before using for every different user and IR sensors doesn't work well with different light conditions in the surrounding.

Immaculate equipment frameworks are commonly actualized as Application Specific Integrated Circuits (ASIC). Contrasted with different innovations, ASIC have a high working recurrence bringing about better execution, low power utilization, high level of parallelism and settled outline devices. Nonetheless, a lot of improvement time is required to advance and execute the outlines. Other than the ASIC arrangements are not adaptable and can't be changed, bringing about high improvement expenses and hazard. The product execution of eye following in DSP offers awesome arrangement of adaptability combined with parallel information handling. The disadvantages of chip are both higher force utilization, and second rate execution contrasted with custom ASIC. Configurable stages, for example, FPGA, join the points of interest from both immaculate equipment and unadulterated programming arrangements. All the more particularly, the high parallelism and computational rate of equipment and the adaptability and short outline time of programming. Along these lines greater part of existing face and eye following frameworks are executed on programmable gadgets (FPGA, microcontrollers) incorporated with implanted programming in view of multiprocessor stage.

1.1.3 Current FPGA based systems

The non-restrictive non-contact eye taking after systems proposed so far may be gathered into two arrangements. The first makes utilization of infrared (IR) gadgets and adventures the intelligent properties of understudies. The eye following here is straightforward, precise however high vitality utilization because of different IR sources. Frameworks of the second gathering track eyes with the guide of "common" camera module. Constrains and Limitations

When trying to detect and track human eyes using a camera, there are few aspects we need to look out for.

Eyes can glitch faster. So the systems should be fast enough to capture every move of the eye. The distance from the camera to users' eyes is very important. Depending on different kinds of activities, this distance may vary. But our system should have the ability to detect & tracking eyes from an acceptable distance much like up to 2 meters.

The lighting condition in the surrounding area will effect to the accuracy of the eye detection upon different image processing algorithms. The other constrain is the fact that some users use spectacles. This should be considered when the system is implemented. Finally the system should not have any calibration methods for different users, since it might not be much user-friendly to the users.

These constrains and limitations should be thoroughly inspected when the system is implemented.

1.2 Background

Eye tracking has an immeasurable measure of utilizations, for example, Human Computer Interfaces, Virtual reality, augmented reality, business use, action acknowledgment, in-vehicle research, psychological and restorative examination, correspondence frameworks for incapacitated etc.

But the eye tracking systems currently available have few problems that prevent them from being used on applications mentioned above. The currently available systems are either fully software based or a software based system with a hardware unit. The biggest problem with the software based system is they are slow. Real time image processing need a lot of processing power and the general purpose computers are slower at outputting a lot of processing power constantly.

The problem with the software & hardware combined system is they consume a lot of power because the hardware eye tracking unit is implemented using IR sensors. The other problem is these IR sensor systems are not much accurate and need calibration for every different user.

For the eye tracking technology to be used in a lot of applications there should be few abilities it should acquire.

- Real-time
- High accuracy
- Low power consumption
- Low cost

The main outcome that I am trying to archive thought my project is to implement a FPGA based real time eye tracking system with higher accuracy.

1.3 Statement of the problem

There are few problems with the currently available FPGA based eye tracking systems.

- Accuracy
- Calibration
- Use of several input devices

The accuracy of the currently systems are pretty good. But my belief is they can be improved.

The other problem with the currently available systems is the calibration. Some of the devices need a calibration before using for every different user. This is not a quality of a user friendly system. They system should have the ability to interact with any user without any calibration or changes to the settings.

The other problem with few systems available right now is the use of several input devices. That means some systems use two cameras to track the eyes. This compromise the implementation of a portable system.

These problems should be addressed carefully to implement a good eye tracking system for the future.

1.4 Aims and objective

1.4.1 Aims

- The main aim of this project is to develop a FPGA based real-time eye tracking system that can detect human eyes with higher accuracy.
- Develop the system with only one input device (camera).
- Develop the system to work with any user without any calibration.

1.4.2 Objectives

- Learn deeply about different kinds of image processing algorithms and use some of them to get the higher accuracy that is aims of this project.
- Learn more about FPGAs and create an optimized system that can be implemented into a portable device in the future.

2. Relevant Theory and Analysis

Several image processing algorithms should be used when implementing the eye tracking system. There are several algorithms that are already available to use. There are two main categories in these algorithms such as feature extraction algorithms and clustering algorithms. Feature extraction algorithms output different type of features of the specific image such as edges, curvatures, lines, junctions and shapes. Clustering algorithms are used to sort the input data into different groups using the different feature data extracted from the feature extraction algorithms. Both types of algorithms are necessary while implementing the system.

2.1 Viola Jones Algorithm

The Viola–Jones object detection system [13] is the first object detection structure to give aggressive item location rates progressively proposed in 2001 by Paul Viola and Michael Jones. Although it can be prepared to distinguish an assortment of article classes, it was inspired essentially by the issue of face identification.

The issue to be tackled is identification of countenances in a picture. A human can do this effortlessly, yet a PC needs exact directions and limitations. To make the assignment more reasonable, Viola–Jones requires full view frontal upright countenances. Along these lines keeping in mind the end goal to be distinguished, the whole face must point towards the camera and ought not to be tilted to either side. While it appears these imperatives could decrease the calculation's utility to some degree, on the grounds that the discovery step is frequently trailed by an acknowledgment venture, by and by these cutoff points on posture are very satisfactory.

The characteristics of Viola–Jones algorithm which make it a good detection algorithm are:

- Robust – very high detection rate & very low false-positive rate always.
- Real time – For practical applications at least 2 frames per second must be processed.
- Face detection only - The goal is to distinguish faces from non-faces

The algorithm has four stages:

a) Haar Feature Selection
b) Creating an Integral Image
c) Ada-boost Training
d) Cascading Classifiers

2.2 Ada-boost

Ada-Boost [14] is a machine learning meta-calculation detailed by Yoav Freund and Robert Schapire. It can be utilized as a part of conjunction with numerous different sorts of learning calculations to enhance their execution. The yield of the other learning calculations is consolidated into a weighted whole that speaks to the last yield of the supported classifier. AdaBoost is versatile as in consequent feeble learners are changed for those examples misclassified by past classifiers. AdaBoost is touchy to loud information and anomalies. In a few issues, on the other hand, it can be less defenseless to the overfitting issue than other learning calculations. The individual learners can be powerless, yet the length of the execution of every one is somewhat superior to anything irregular speculating (i.e., their mistake rate is littler than 0.5 for paired arrangement), the last model can be demonstrated to unite to a solid learner

While each learning calculation will tend to suit some issue sorts superior to anything others, and will commonly have a wide range of parameters and designs to be balanced before accomplishing ideal execution on a dataset, AdaBoost is regularly alluded to as the best out-of-the-case classifier. At the point when utilized with choice tree learning, data accumulated at every phase of the AdaBoost calculation about the relative "hardness" of every preparation test is sustained into the tree developing calculation such that later trees tend to concentrate on harder to group cases.

2.3 Template Matching

Template matching [15] is a procedure in digital image processing for discovering little parts of an image which coordinate a template image. It can be utilized as a part of assembling as a piece of value control, an approach to explore a mobile robot, or as an approach to identify edges in images.

For templates without strong features, or for when the greater part of the layout image constitutes the matching image, a layout based methodology may be successful. As previously stated, since layout based format matching may conceivably require testing of an extensive number of focuses, it is conceivable to lessen the quantity of reducing so as to inspect focuses the determination of the pursuit and layout images by the same component and performing the operation on the resultant scaled back images, giving a hunt window of information focuses inside of the inquiry image so that the layout does not need to seek each practical information point, or a mix of both.

2.4 Support Vector Matching (SVM)

In machine learning, support vector machines [16] are regulated learning models with related learning algorithms that break down information and perceive examples, utilized for grouping and relapse examination. Given an arrangement of training illustrations, each checked for fitting in with one of two classifications, a SVM training algorithm assembles a model that allots new samples into one classification or the other, making it a non-probabilistic double direct classifier. A SVM model is a representation of the samples as focuses in space, mapped so that the cases of the different classifications are separated by an unmistakable crevice that is as wide as could be allowed. New cases are then mapped into that same space and anticipated to fit in with a classification in light of which side of the hole they fall on.

Notwithstanding performing straight arrangement, SVMs can effectively perform a non-direct characterization utilizing what is known as the portion trap, certainly mapping their inputs into high-dimensional component spaces.

At the point when information is not named, a directed learning is unrealistic, and an unsupervised learning is required, that would discover regular bunching of the information to gatherings, and guide new information to these shaped gatherings. The grouping algorithm which gives a change to the support vector machines is called support vector bunching is profoundly utilized as a part of mechanical applications either when information is not marked or when just some information is named as a preprocessing for a characterization pass; the grouping strategy was distributed.

2.5 Neural Network

In machine learning and cognitive science, artificial neural networks (ANNs) [17] are a group of models propelled by biological neural networks (the focal sensory systems of creatures, specifically the mind) and are utilized to gauge or inexact capacities that can rely on upon an expansive number of inputs and are for the most part obscure. Artificial neural networks are by and large exhibited as frameworks of interconnected "neurons" which trade messages between one another. The associations have numeric weights that can be tuned in view of experience, making neural nets versatile to inputs and equipped for learning.

For instance, a neural system for penmanship acknowledgment is characterized by an arrangement of data neurons which may be actuated by the pixels of an info picture. Subsequent to being weighted and changed by a capacity (controlled by the system's architect), the initiations of these neurons are then gone on to different neurons. This procedure is rehashed until at long last, a yield neuron is initiated. This figures out which character was perused.

Like other machine learning strategies - frameworks that gain from information - neural networks have been utilized to tackle a wide assortment of assignments that are difficult to unravel utilizing customary standard based programming, including PC vision and discourse acknowledgment.

2.6 K-Nearest Neighbor

In example acknowledgment, the k-Nearest Neighbors [18] algorithm is a non-parametric system utilized for arrangement and regression. As a part of both cases, the data comprises of the k nearest preparing cases in the component space. The yield relies on upon whether k-NN is utilized for grouping or relapse:

In k-NN arrangement, the yield is a class membership. An item is ordered by a lion's share vote of its neighbors, with the article being relegated to the class most basic among its k closest neighbors In the event that k = 1, then the item is just allocated to the class of that solitary closest neighbor.

In k-NN relapse, the yield is the property estimation for the article. This worth is the normal of the estimations of its k closest neighbors.

k-NN is a kind of occurrence based learning, or sluggish realizing, where the capacity is just approximated locally and all calculation is conceded until order. The k-NN algorithm is among the most straightforward of all machine learning algorithms.

Both for characterization and relapse, it can be helpful to dole out weight to the commitments of the neighbors, so that the closer neighbors contribute more to the normal than the more far off ones. For instance, a typical weighting plan comprises in giving every neighbor a weight of 1/d, where d is the separation to the neighbor.

The neighbors are taken from an arrangement of articles for which the class or the item property estimation is known. This can be considered as the preparation set for the algorithm, however no express preparing step is required.

A weakness of the k-NN algorithm is that it is delicate to the neighborhood structure of the information. The algorithm has nothing to do with and is not to be mistaken for k-implies, another well-known machine learning system.

3. Approach / Methodology

3.1 Approach

This project was divided to two main parts. First part is using Matlab to test different types of algorithms and their characteristics. By using Matlab, the algorithms can be customized to our specific purpose and test them, it they are simple enough to implement on the FPGA. After testing the algorithms in Matlab, they are implemented and tested in FPGA.

First skin color detections is done. Then using that, face image is extracted. Then a separate eye detection algorithm is applied to the extracted face image.

3.2 Methodology

3.2.1 Streaming video from Camera to LCD display through the FPGA

Accessing the LCD display through the FPGA will allow me to generate and display different kinds of graphics on the display. This will be very useful when creating a GUI. The camera module has a parallel output. These parallel data should be captured frame by frame to create the image which the camera is seeing, inside the FPGA. Normally the output device's resolution should match the resolution of the input device to display the correct images. There should be a lot of programming to be done to output a constant video stream from camera to the display through the FPGA.

3.2.2 Skin color filtering

Skin color detection is a very simple method to detect a face in an image compared to all the other face detection algorithms available. But the accuracy is low because it will detect any area with skin. But in this case we don't need that much accuracy.

First the RGB image will be converted to YUV color space because in that color space U means the chrominance value of the image. Chrominance value is directly connected to the skin color. The filtering algorithm will be tested in Matlab and then implemented in FPGA.

3.2.3 Color object detection and tracking – FPGA

To detect different colors separately, a color filtering algorithm should be implemented. After that different color objects can be isolated. Then those data can be inserted to a tracking algorithm which will then start tracking the color objects.

By implementing a color object tracking algorithm, a lot of things can be learned about implementing a tracking algorithm successfully in FPGA.

3.2.4 Eye detection and tracking algorithm

First a feature extraction algorithm should be used to extract features from a set of sample eye pictures and non-eye pictures. The main idea is to use haar feature in here. Then a clustering algorithm should be used to separate the eye pictures from non-eye pictures. Clustering algorithms such as k-Nearest neighbor, SVM or neural network can be used here.

4. Design and Implementation

4.1 Process of the project

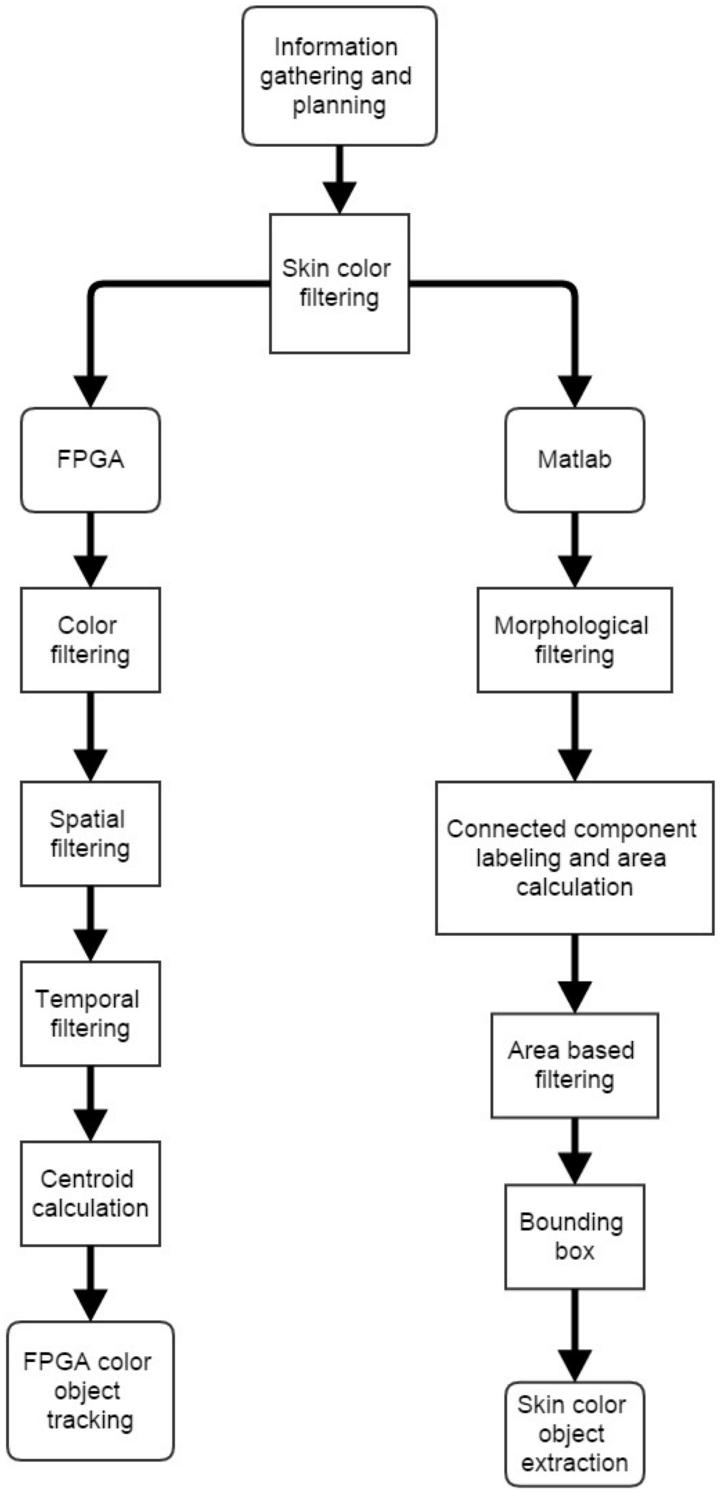


Figure 1. Project process block diagram

4.2 Skin color extraction algorithm

The skin color detection algorithm here was derived from the method described in [1]. Color segmentation has been proved to be and effective method to detect face region due to its low computational requirements and ease of implementation. Compared to the featured-based method, the color-based algorithm required very little training, which means it can be easily implemented inside a FPGA.

First, the original image was converted to a different color space, namely modified YUV. Then the skin pixels were segmented based on the appropriate U range. Morphological filtering was applied to reduce false positives. Then each connected region of detected pixels in the image was labeled. The area of each labeled region was computed and an area-based filtering was applied. Only regions with large area were considered face regions. The centroid of each face region was also computed to show its location.

4.2.1 Modified YUV color space

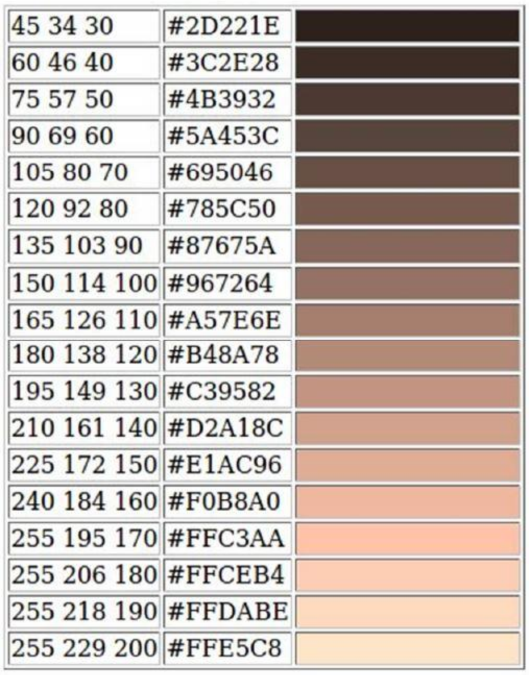
Converting the skin pixel information to the modified YUV color space would be more advantageous since human skin tones tend to fall within a certain range of chrominance values (I.e. U-V component), regardless of the skin type. The conversion equations are show as follows [1].

$$Y = \frac{R + 2G + B}{4}$$

$$U = R - G$$

$$V = B - G$$

These equations allowed thresholding to work independently of skin color intensity.



45 34 30	#2D221E	🟫
60 46 40	#3C2E28	🟫
75 57 50	#4B3932	🟫
90 69 60	#5A453C	🟫
105 80 70	#695046	🟫
120 92 80	#785C50	🟫
135 103 90	#87675A	🟫
150 114 100	#967264	🟫
165 126 110	#A57E6E	🟫
180 138 120	#B48A78	🟫
195 149 130	#C39582	🟫
210 161 140	#D2A18C	🟫
225 172 150	#E1AC96	🟫
240 184 160	#F0B8A0	🟫
255 195 170	#FFC3AA	🟫
255 206 180	#FFCEB4	🟫
255 218 190	#FFDABE	🟫
255 229 200	#FFE5C8	🟫

Figure 2. Different skin color samples

4.2.2 Thresholding/skin color detection

After skin pixels were converted to the modified YUV space, the skin pixels can be segmented based on the following experimented threshold.

$$18 < U < 45$$

$$-40 < V < 11$$

As seen in Figure 2, the blue channel had the least contribution to human skin color. Additionally, according to [2], leaving out the blue channel would have little impact on thresholding and skin filtering. This also implies the insignificance of the V component in the YUV format. Therefore, the skin detection algorithm using here was based on the U component only. Applying the suggested threshold for the U component would produce a binary image with raw segmentation result, as depicted in Figure 3.

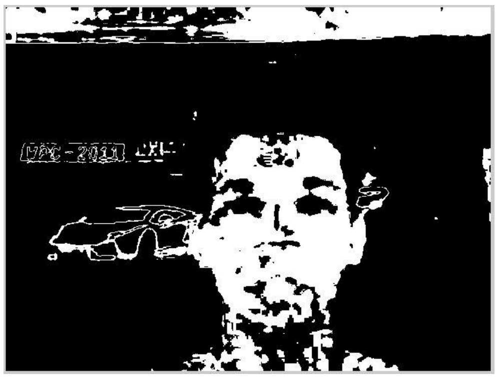


Figure 3. Skin color filtering

4.3 Matlab Implementation

4.3.1 Morphological filtering

Realistically, there are so many other objects that have color similar to the skin color. As seen In Figure 3, there are lots of false positives present in the raw segmentation result. Applying morphological filtering, including erosion and hole filling would, firstly, reduce the background noise and, secondly, fill In missing pixels of the detected face regions, as illustrated in Figure 4. MATLAB provided built-in functions *imerode* and *imfill* for these two operations.

```
outp = imerode(inp, strel('square',5));
```

The command *imerode* erodes the input image *inp* using a square of size 3 as a structuring element and returns the eroded image *outp*, This operation removed any group of pixels that had size smaller than the structuring element's,

```
outp = imfill(inp, ‘holes’);
```

The command *imfill* fills holes in the binary Input Image *inp* and produces the output image *outp*. Applying this operation allowed the missing pixels of the detected face regions to be filled in. Thus, it made each face region appear as one connected region.

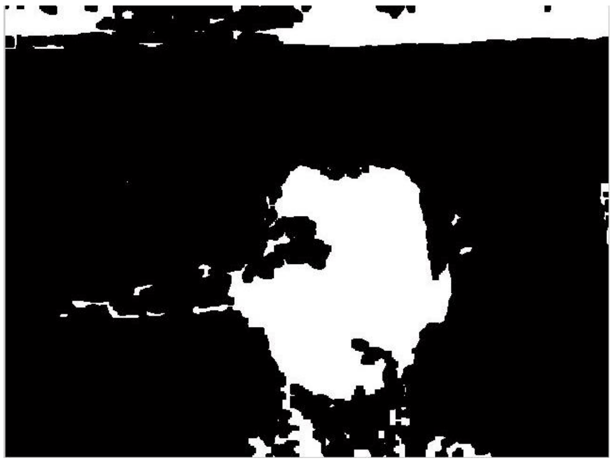


Figure 4. Results after morphological filtering

4.3.2 Connected component labeling and area calculation

After each group of detected pixels became one connected region, connected component labeling algorithm was applied. This process labeled each connected region with a number, allowing us to distinguish between different detected regions. The built-in function *bwlabe1* for this operation was available in MATLAB. In general, there are two main methods to label connected regions in a binary image known as recursive and sequential algorithms.

```
[L, n] = bwlabel(inp);
```

The command *bwlabel* labels connected components in the input image *inp* and returns a matrix L, of the same size as *inp*. *L* contains labels for all connected regions in *inp*. *n* contains the number of connected objects found in *inp*. The command *regionprops* can be used to extract different properties, including area and centroid, of each labeled region in the label matrix obtained from *bwlabe1*.

```
face_region = regionprops(L,'Area');

face_area = [face_region.Area];
```

The two commands above performed two tasks (1) extract the area information of each labeled region (2) store the areas of all the labeled regions in the array face_area in the order of their labels. For instance face_area(1) = 102 would mean the area of the connected component with label "1" is 102 pixels.

4.3.3 Area based filtering

Note that morphological filtering only removed some background noise, but not all. Filtering detected regions based on their areas would successfully remove all background noise and any skin region that was not likely to be a face. This was done based on the assumption that human faces are of similar size and have largest area compared to other skin regions, especially the hands. Therefore, to be considered a face region, a connected group of skin pixels need to have an area of at least 100% of the largest area. This number was obtained from experiments on training images. Therefore, many regions of false positives could be removed in this stage.

```
face_id× = find(face_area==(1)*max(face area))

face_shown = ismember (L, face_idx)
```

These two commands performed the following tasks (1) look for the connected regions whose areas were of 100% of the largest area and store their corresponding indices in *face_idx(*2) output the image *face_shown* that contained the connected regions found in (1).

4.3.4 Centroid Computation

The next stage was to determine face location. The centroid of each connected labeled face region can be calculated by averaging the sum of X coordinates and Y coordinates separately. The centroid of each Face region in Figure 5 is denoted by the red circle. Here the centroid of the largest connected region was extracted using *regionprops*.

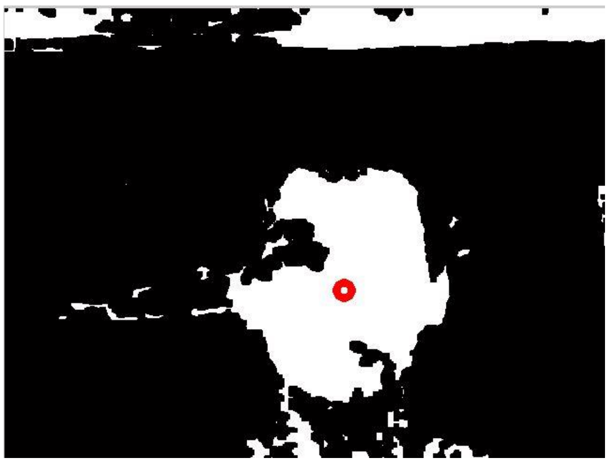


Figure 5. Results after centroid calculation

4.3.5 Bounding Box

Using the same method that was used to calculate the centroid, coordinated of bounding boxes can be found for each labeled area. Using these bounding box coordinates, each detected face can be extracted.

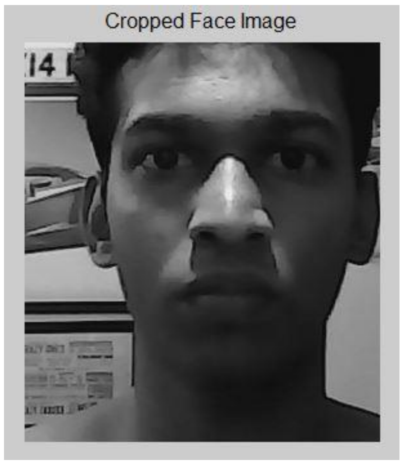


Figure 6. Automatically cropped skin color area

Before the algorithm was developed in Verilog, it was implemented and tested on still images in MATLAB to verify its functionality, as illustrated from Figure 3 to Figure 6. The Image Processing Toolbox provided in MATLAB allowed the process of, developing and testing the algorithm to be more efficient. Furthermore, verifying the accuracy of the detection algorithm on still pictures provided fair results. However, the transition from MATLAB to Verilog coding was not as smooth as expected due to the limitation· ns of the Verilog language. Specifically, morphological filtering was modified and extended to spatial and temporal filtering. In addition, connected component labeling was not implemented in Verilog as this language does not allow calling a function recursively. A modified detection algorithm will be discussed later.

4.4 FPGA Implementation

4.4.1 FPGA VGA controller

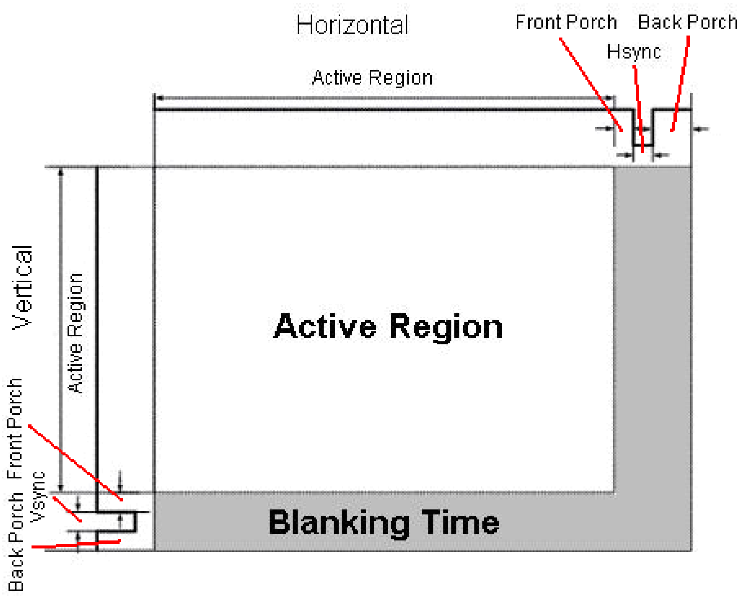


Figure 7. Sync cycles and display area of a LCD display

Table 1. VGA display parameters

	Horizontal	Vertical
Front porch	16	11
sync pulse	96	2
back porch	48	31
active area	640	480
total area	800	524

Pixel clock:	27 MHz
Time per pixel:	37 ns
Length of 1 row:	800 pixels
Time per row:	29.6 us

4.4.2 Process from analog camera to VGA display

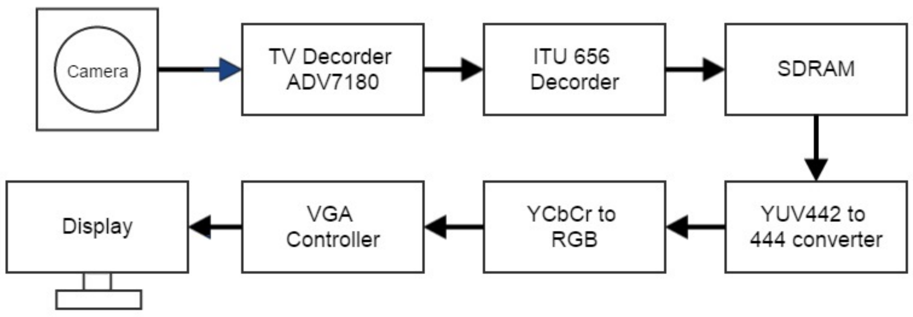


Figure 8. Analog camera to Display process block diagram

ITU-R BT.656

ITU656 describe a simple digital video protocol for streaming uncompressed PAL or NTSC signals. The protocol build upon the 4:2:2 digital video encoding parameters defined by ITU-R recommendation BT.601 which provide interfaced video data streaming each field separately & uses the YCbCr color space and a 13.5 MHz sampling frequency for pixels. Can be either 8 bit or 10 bit.

Y'CbCr 4:2:2

The two chroma components are sampled at half the sample rate of luma. The horizontal resolution is halved. This reduce the bandwidth of an uncompressed video signal by 1/3 with little to no visual difference.

Y'CbCr 4:4:4

Each of three Y'CbCr components have the same sample rate. This scheme is sometimes used in high-end film scanners and cinematic post production.

4.4.3 Eye detection & tracking process

To understand what algorithms are used, what are their features, pros & cons of using them, a decision was made to firstly implement the eye detection & tracking algorithms in a computer software such as Matlab.

As the final implementation of the algorithms will be on FPGA, there had to be some important properties within the eye detection algorithm;

- It should be simple enough so it can be implemented inside the FPGA.
- It should be accurate
- It should be fast & real-time.

To detect eyes accurately, first some unique properties of eye should be found that can be seen by a camera. Because in here, normal object detection or motion detection methods can't be used since the eyes can move relative to the face.

Object detection can be done by either using color detection to isolate the color of the object or using a special property of the specific object such as, a ball can be categorized as a circular object or a box can be categorized as a rectangular object.

Motion detection can be done by getting the difference of two adjacent frames. This way the difference image will only highlight the part where the moving object moved. By using filters, the difference image can be improved to isolate the moved object.

The problem with using these methods for eye tracking is, the eyes can move relative to the head. So even though the head is tracked using motion tracking, the eyes won't have the same movements. Motion detection and tracking can't be used only for eyes because eyes are relatively small and their movements relative to the head can't be isolated because of the light condition of the surrounding.

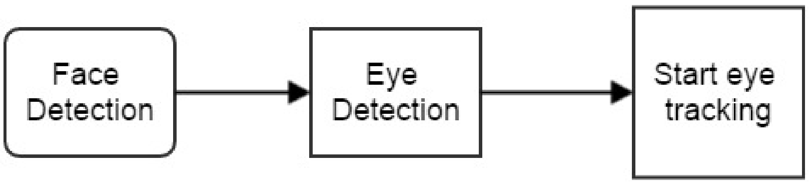


Figure 9. Eye tracking process

4.4.4 RGB to Gray

A predefined matrix is used to convert RGB color scale into gray scale color space.

$$Gray = \begin{bmatrix} 0.299 & 0.587 & 0.114 \end{bmatrix} \begin{bmatrix} Rin \\ Gin \\ Bin \end{bmatrix}$$

By converting all R, G, B values to the same calculated gray value, a Gray scale image is produced.

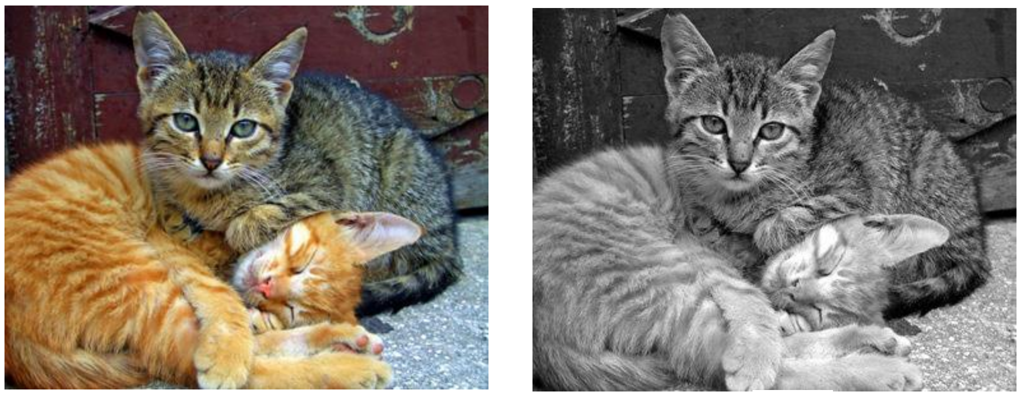


Figure 10. (a) RGB image, (b) Grayscale image

The advantage of converting RGB to grayscale is, it saves a lot of memory when the picture is saved on a memory for processing. Normally, for every pixel, three bytes were needed. So for a 640 x 480 picture, it takes 921.6 kBytes. But when we save grayscale image, it only takes 307.2 kBytes. Even though it takes less memory for a gray scale image, it still have all the information, a RGB image contain.

4.4.5 Thresholding

Since 10 bit color was used in Verilog, adjusting the aforementioned U range yields

$$40 < U < 296$$

In this step, each input video frame was converted to a “binary image” showing the segmented raw result.

4.4.6 Spatial filtering

This step was similar to the erosion operation used in the software algorithm. However, the structuring element used here did not have any particular shape. Instead, for every pixel p, its neighboring pixels in a 10x10 neighborhood were checked. If more than 75% of its neighbors were skin pixels, p was also a skin pixel. Otherwise p was a non-skin pixel. This allowed most background noise to be removed because usually noise scattered randomly through space, as shown in Figure 11, In Figure 12, because p only had 4 neighboring pixels categorized as skin, p was concluded to be a non-skin pixel and, thus, converted to a background pixel.

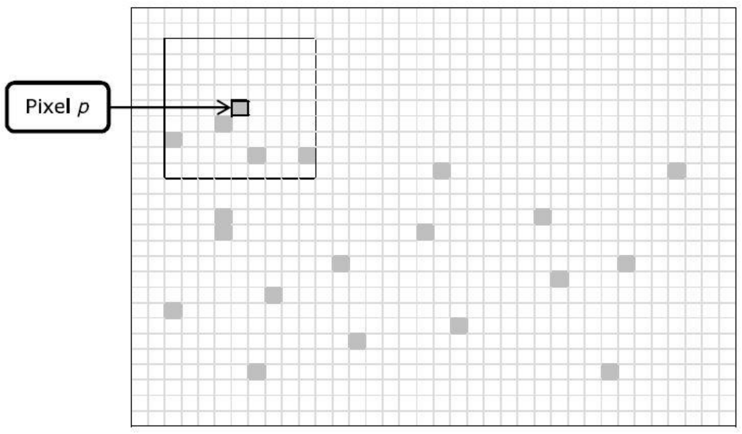


Figure 11. Spatial filtering for a pixel P before filtering

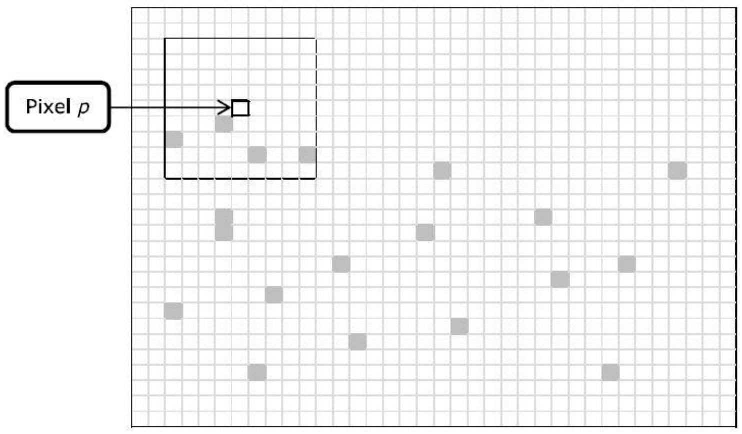


Figure 12. Spatial filtering for a pixel P after filtering

To examine the neighbors around a pixel, their values needed to be stored. Therefore, ten shift registers were created to buffer the values of ten consecutive rows in each frame. As seen in Figure 13, each register was 640-bit long to hold the binary values of 640 pixels in a row. Each bit in data_regl was updated according to the X coordinate. For instance, when the X coordinate was 2, data_regl[2] was updated according to the result of thresholding from the previous stage. Thus, data_regl was updated every clock cycle. After all the bits of data_regl were updated, its entire value was shifted to data_reg2. Thus, other registers (from data_reg2 to data_reg10) were only updated when the X coordinate was 0. Values of data_reg2 to data_reg10 were used to examine a pixel's neighborhood.

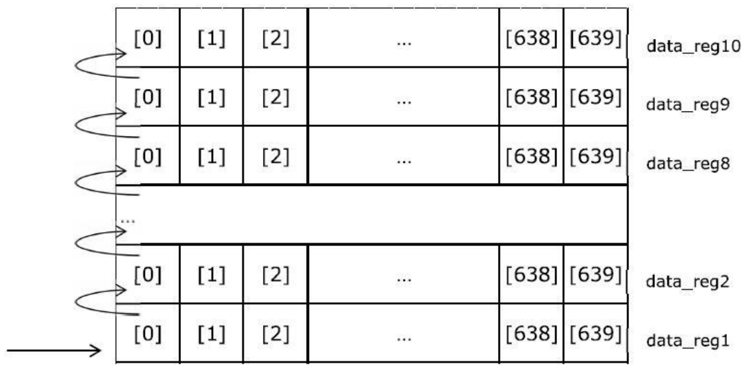


Figure 13. Ten shift registers for ten consecutive rows

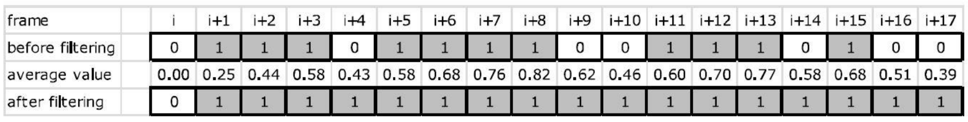
There was a trade-off between the number of shift registers being used (i.e. the size of the neighborhood) and the performance of the spatial filter. A larger neighborhood required more registers to be used but, at the same time, allowed more noise to be removed.

4.4.7 Temporal Filtering

Even small changes in lighting could cause flickering and made the result displayed on the VGA screen less stable. Applying temporal filtering allowed flickering to be reduced significantly. The idea of designing such a filter was borrowed from the project "Real-Time Cartoonifier" (see References for more information of this project). The temporal filter was based on the following equation. avg_out = (3/4) avg_in + (1/4) data data: filtered result obtained from the previous stage of a pixel, namely p, in current frame avg_in: average value of p from previous frame avg_out: average value of p in current frame This is approximately equal to averaging four consecutive frames over time. To ease the computational effort, the equation above can be re-written as

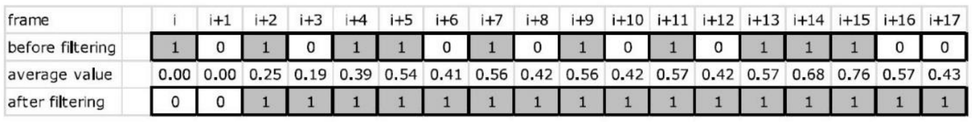
```
avg_out = avg_in — (1/4) avg_in + (1/4) data;

avg_out = avg_in — avg_in>>2 + data>>2;
```

The filtered result of a pixel in this stage was determined based on its average value (i.e. avg_out). If its average value was greater than 0.06 (number obtained from experiments), the pixel was considered skin. Otherwise, the pixel was non-skin. Figure 14 and Figure 15 illustrate the process of temporal filtering for two pixels pl and p2. In both examples, pixel pl and p2 are truly skin pixels. However, the results before filtering were unstable due to light variations. The temporal filter smoothed the output and, thus, reduced flicker significantly.



frame	i	i+1	i+2	i+3	i+4	i+5	i+6	i+7	i+8	i+9	i+10	i+11	i+12	i+13	i+14	i+15	i+16	i+17
before filtering	0	1	1	1	0	1	1	1	1	0	0	1	1	1	0	1	0	0
average value	0.00	0.25	0.44	0.58	0.43	0.58	0.68	0.76	0.82	0.62	0.46	0.60	0.70	0.77	0.58	0.68	0.51	0.39
after filtering	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 14. Example of temporal filtering for a pixel p1



frame	i	i+1	i+2	i+3	i+4	i+5	i+6	i+7	i+8	i+9	i+10	i+11	i+12	i+13	i+14	i+15	i+16	i+17
before filtering	1	0	1	0	1	1	0	1	0	1	0	1	0	1	1	1	0	0
average value	0.00	0.00	0.25	0.19	0.39	0.54	0.41	0.56	0.42	0.56	0.42	0.57	0.42	0.57	0.68	0.76	0.57	0.43
after filtering	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 15. Example of temporal filtering for a pixel p2

4.4.8 Centroid Computation

Finally, centroid was computed to locate the object region. Because connected component labeling was not implemented as initially planned, it was infeasible to calculate the centroid for each object region separately. This limited the number of objects to be detected to two as maximum. First assume that only one object was present. Therefore, its centroid would just be the centroid of all detected pixels, as shown in Figure 16. Note that this calculation would only be correct if one object was present.

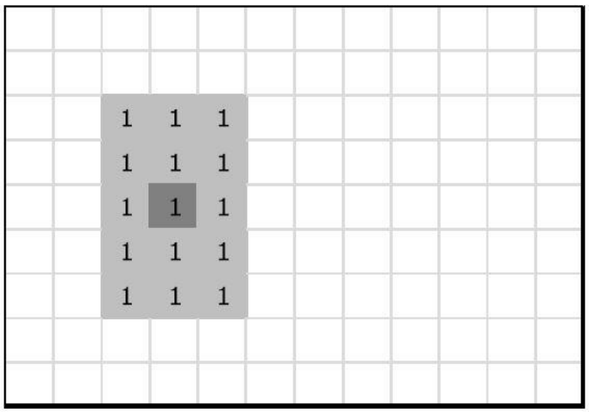


Figure 16. Centroid for all detected pixels

Although the pixels of one object region might not be connected (and labeled) as originally planned, simply calculating the centroid of all detected pixels still gave a good estimate for the object location, as shown in Figure 15. Since area-based filtering was also not applied (due to the lack of connected component labeling), other skin regions—mostly the hands were not entirely removed. However, even if the hands were present, calculating the centroid of all detected pixels still allowed us to locate the object region. This was a reasonable estimate because, compared to the object area, the area of the hand/hands was much smaller.

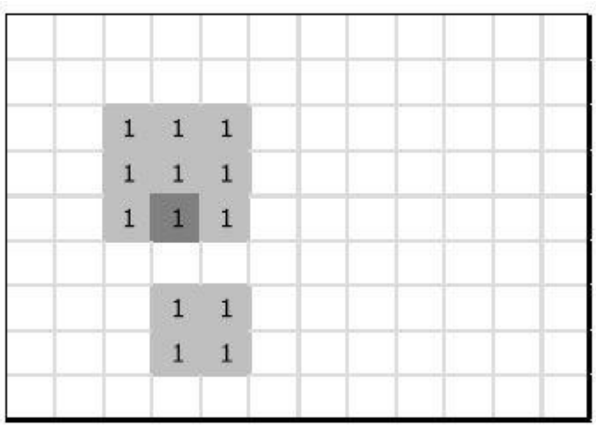


Figure 17. Centroid for all detected pixels - two objects

5. Results

5.1 Matlab testing

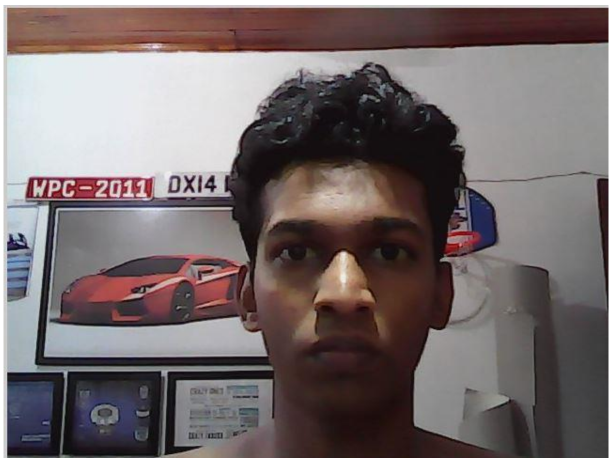


Figure 18. Original Image

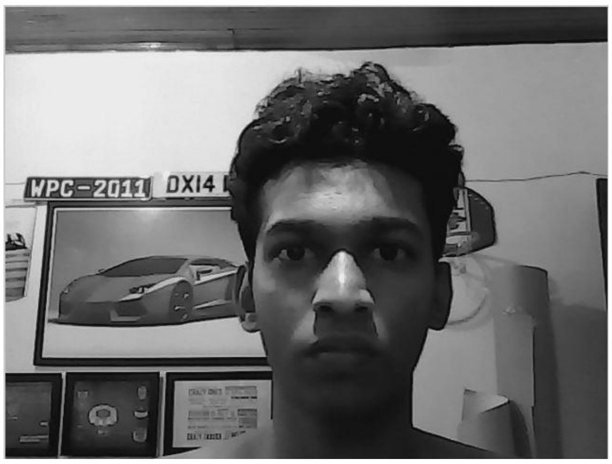


Figure 19. Gray scale image

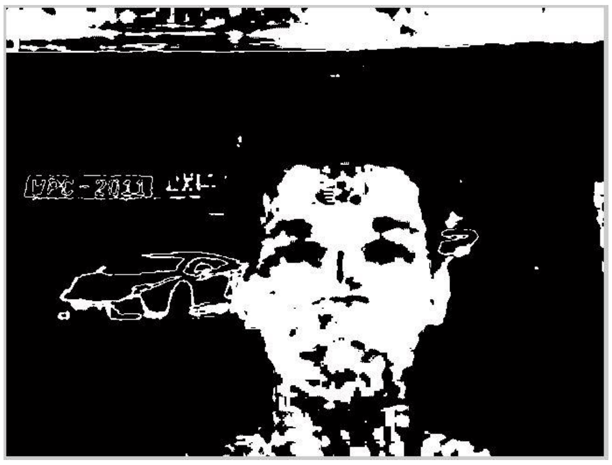


Figure 20. Skin color filtered image

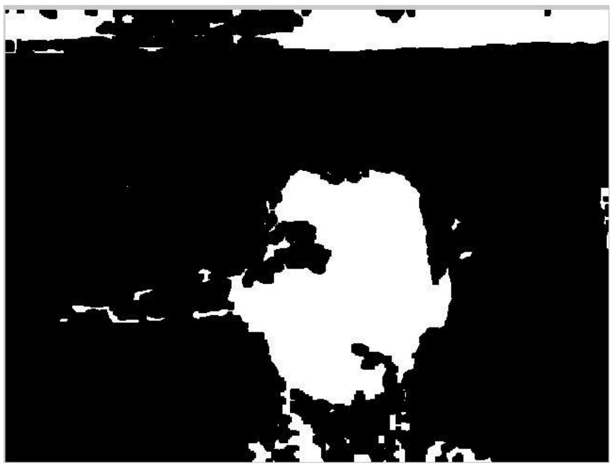


Figure 21. Output of morphological filtered image

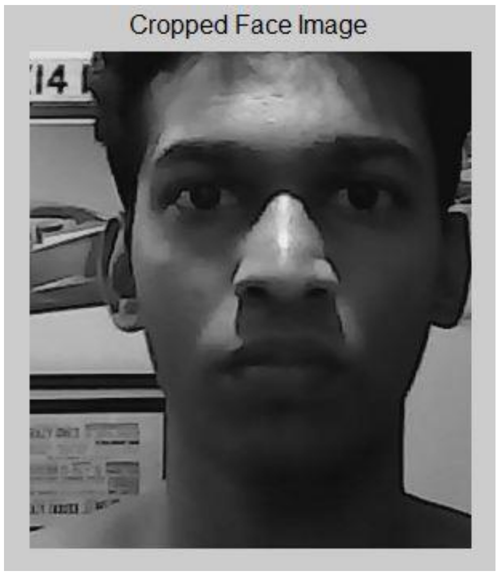


Figure 22. Cropped skin color image

5.2 FPGA testing

Skin color, red, green color tracking

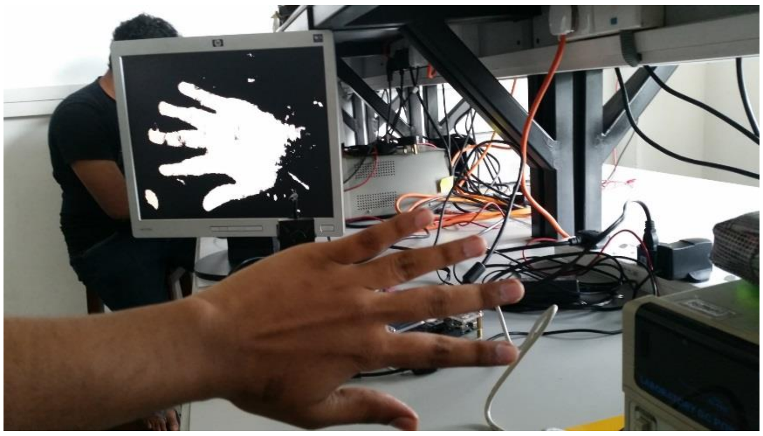


Figure 23. Color thresholding

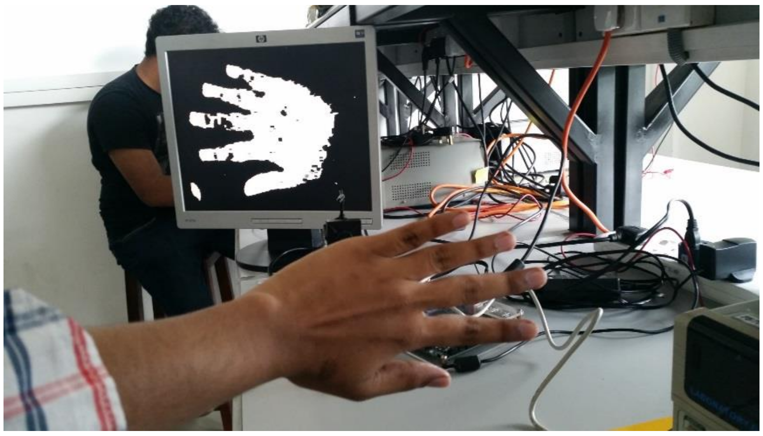


Figure 24. Spatial filtering

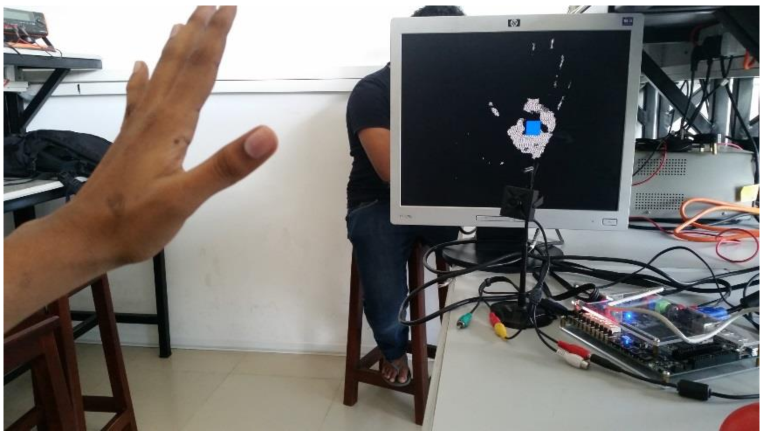


Figure 25. Temporal filtering and Centroid calculation

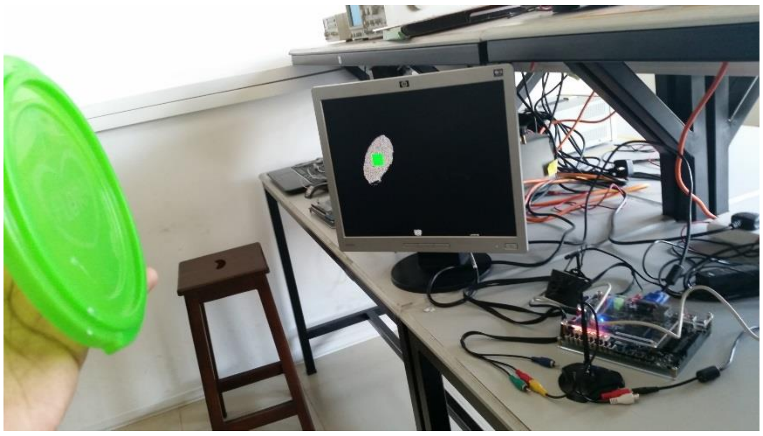


Figure 26. Green Color tracking

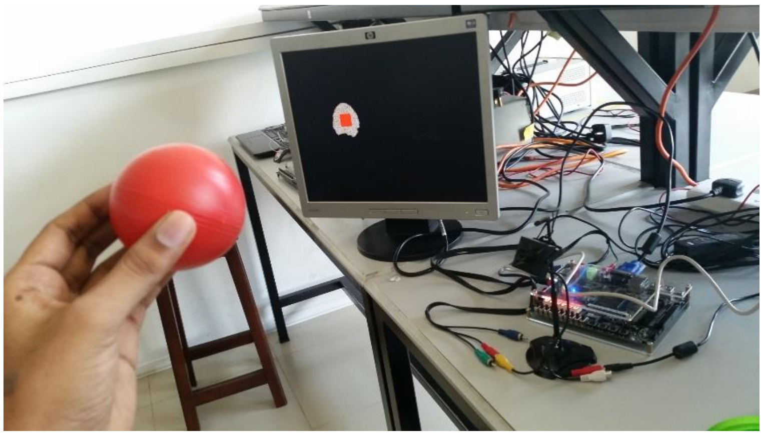


Figure 27. Red Color Tracking

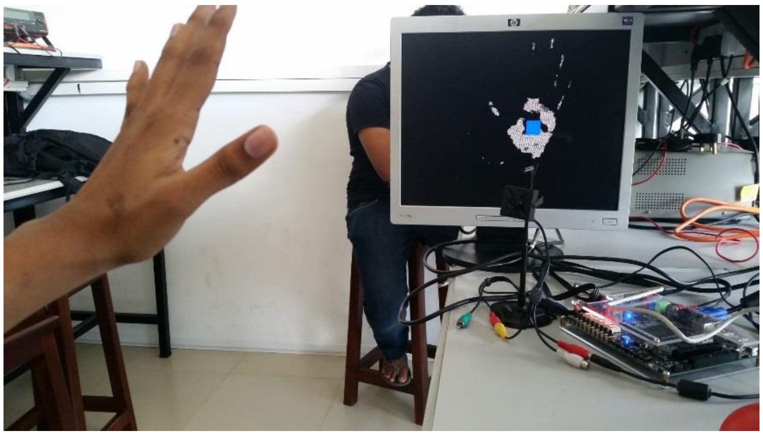


Figure 28. Skin color tracking

FPGA compilation results

Table 2. FPGA program resource usage

Device	Altera 5CSEMA5F31C6
Total pins	241 / 457
Total block memory bits	45,024 / 4,065,280
Total DSP Blocks	8 / 87
Total PLLs	1 / 6
Logic utilization (in ALMs)	8,272 / 32,070
Clock	50 MHz

6. Discussion of results

As you can see from figure 18 and figure 20, when skin color thresholding is done, there's a tendency to filter out red color also. As seen in figure 20 and figure 21, morphological filter removes a lot of noise in the background in the Matlab program. As compared in figure 23 and figure 24, the spatial filter remove a lot of noise in the FPGA program. Figure 25 shows how the temporal filtering and centroid calculation works in FPGA program. Figure 26 to 28 shows different color tracking and skin color tracking in FPGA.

6.1 Speed

A clock of 27 MHz was used for the object detection and tracking algorithm. Since the timing was synchronized with the VGA clock, the VGA display was able to update within the time gap between drawing two consecutive frames, Therefore, the camera was able to detect and track objects in real time.

6.2 Accuracy

Error seemed to occur when two or more objects are present. At that moment the system will track the center of all detected objects, which is not accurate. When the system is tracking skin color objects, there's a tendency to track objects which has similar color to red or orange color. When the object moves away from the camera, the accuracy of object tracking decreases.

6.3 Limitations

The FPGA system only tracks color objects, such as Green, Red and skin color. It cannot track any other colors. The analog camera that is used here is very sensitive to lighting of the surrounding. Because of that, system has difficulty filtering red, green and skin colors at different lighting conditions.

7. Critiques

Using FPGA I could only achieve up to tracking different color objects such as green, red and skin color. Different switches in the FPGA correspond to different color selections. Another set of switches are used to jump between different debugging modes such as, thresholding, spatial filtering and temporal filtering with centroid calculation.

Using matlab I could only achieve up to extracting the face area using skin color detection.

The aim was to after extracting the face area, to apply an eye detection algorithm to that face image. By testing this theory in matlab, the next step was to implement it in FPGA.

The biggest problem with using Matlab as an algorithm testing medium was that, it had a lot of built in functions which were specific to image processing. These in-built function made it much easier to implement different algorithms. But when it was time to implement them in FPGA, it was really difficult because there were no image processing libraries for FPGA. So every in-built function in matlab had to be implemented manually in FPGA to make the algorithm work.

Because of this problem, most of the algorithms which were implemented on Matlab were all implemented by me. The idea was to use only minimum amount of Matlab built in functions.

Almost all the tasks that were originally planned took a lot more time than I estimated. It's mainly because, most of the tasks required much more knowledge than what I had. It took more time to acquire the knowledge required and finish the tasks. Some of the tasks had to be changed when the project was processing, because there were much more new things that I learned about the topic when I was doing the process.

8. Costing

The cost for the final system is as follows.

Table 3. Expenditure

Item	Price
Altera FPGA development board	35000
Camera module	1500
Total	**36500**

9. Conclusion and further work

9.1 Conclusion

The purpose of this project was to implement a FPGA based eye tracking system. The project was embedded to a vast range which I didn't anticipate at the beginning of the project.

FPGA is much faster when compared to a Matlab program which does the same job. But because of the lack of image processing libraries in FPGA, makes it much harder to implement different algorithms.

I have achieved up to color object tracking algorithms in FPGA. In Matlab my program can successfully extract the human face from an image. The color object tracking algorithm can be modified to track any other object in the future.

9.2 Further work

First a feature extraction algorithm should be used to extract different features of eyes. Haar features would be a good method to be used in here. Then a simple eye detection algorithm should be implemented in Matlab using different clustering algorithms such as K – Nearest neighbor, Support Vector Machines (SVM) or Neural Networks. K nearest neighbor is a simple, lite weight algorithm which can be implemented on FPGA. But the problem with KNN is the accuracy will be much less. After implementing an accurate, lightweight algorithm in matlab, it should then be implemented in FPGA.

9.3 Future

Microprocessors in the portable smart devices these days are becoming more and more powerful every day. They handle multitasking more like a desktop computer.

Samsung smartphones use their front camera to detect the user's eye and then execute several actions according to them. Some of these features are dimming the display while the user isn't looking at the display, keep the display ON when the user is reading an article, automatic pausing & resuming videos and so on. The problem is, these feature doesn't work properly & fast enough.

To overcome these problems, there can be SoCs with a separate FPGA unit & an ARM processor unit in the future. With these new SoCs, the FPGA unit can separately handle the eye tracking part. That way the applications that device can use eye tracking would be wider and faster.

References

[1]. Turk, M., Pentland, A., “Eigen faces for Recognition”, *Journal of Cognitive Neuroscience*, 1991, vol. 3, no. 1, pp. 71-86.

[2]. Yang, M. H., Ahuja, N., Kriegman, D., “Face Detection Using a Mixture of Linear Subspaces”, *In Proceedings of 4th IEEE International Conference on Automatic Face and Gesture Recognition FG00*, 2000, pp. 70-76.

[3]. Viola, P., Jones, M. J., “Robust Real-Time Object Detection”, *Technical Report CRL 2001/01*, Cambridge Research Laboratory, Compaq, 2001.

[4]. Leung, T. K., Burl, M. C., Perona, P., “Finding Faces in Cluttered Scenes Using Random Labelled Graph Matching”, *In Proceedings of 5th IEEE International Conference on Computer Vision ICCV95*, 1995, pp. 637- 644.

[5]. Yang, M. H., Ahuja, N., “Detecting Human Faces in Color Images”, *In Proceedings of 5th IEEE International Conference on Image Processing ICIP98*, 1998, vol. 1, pp. 127-130.

[6]. Fleck, M., Forsyth, D., Bregler, C., “Finding naked people”, *In Proceedings of 4th European Conference on Computer Vision ECCV96*, 1996, vol. 2, pp. 593-602.

[7]. S.Baluja and D.Pomerleau, *“Non-intrusive gaze tracking using artificial neural networks”*, Technical report CMU-CS-94-102.

[8]. P.Smith et al, “Monitoring head/eye motion for driver alertness using one camera”. Proc. Int.Conf. Pattern Recognition, 2000.

[9]. D. Kim et al, “An FPGA-based Parallel Hardware Architecture for Real-time Eye Detection”. Proc. Semi. Tech. & Sci., June, 2012.

[10]. T. Ando et al, “A low power FPGA implementation of eye tracking”. Pp.1573-1576, 2010.

[11]. H. Peter et al, *“Real-time calibration free autonomous eye tracker”*. Proc. ICASSP 2010, pp.762-765, 2010.

[12]. Eye Tracking by Tobii, available from www.tobii.com

[13]. Wikipedia, "*Viola–Jones object detection framework*", 2001. [Online]. Available: https://en.wikipedia.org/wiki/Viola%E2%80%93Jones_object_detection_framework. [Accessed: 05- May- 2015].

[14]. Y. Freund and R. Schapire, "*A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting*", Journal of Computer and System Sciences, vol. 55, no. 1, pp. 119-139, 1997.

[15]. Wikipedia, "*Template matching*", 2009. [Online]. Available: https://en.wikipedia.org/wiki/Template_matching. [Accessed: 05- May- 2015].

[16]. Wikipedia, "*Support vector machine*", 2001. [Online]. Available: https://en.wikipedia.org/wiki/Support_vector_machine. [Accessed: 05- May- 2015].

[17]. Wikipedia, "*Artificial neural network*", 2001. [Online]. Available: https://en.wikipedia.org/wiki/Artificial_neural_network. [Accessed: 05- May- 2015].

[18]. Wikipedia, "*K-nearest neighbors algorithm*", 2006. [Online]. Available: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm. [Accessed: 05- May- 2015].

[19]. V.G.Moshnyaga, *"The use of eye-tracking for PC Energy management"*, Proc. ETRA 2010, pp.113-116, 2010.

[20]. S. Paschalakis & M. Bober, "*A Low Cost FPGA System for High Speed Face Detection and Tracking*".

Appendix A

Matlab code

```
A = imread('t1.jpg');
figure(1), imshow(rgb2gray(A));
[H,W,D] = size(A);
B = A;
for y=1:1:H
    for x=1:1:W
        % extract image R,G,B values
        R = A(y,x,1);
        G = A(y,x,2);
        B = A(y,x,3);
        % calculate Y,U,V values
        Y = (R+(2*G)+B)/4;
        U = R-G;
        V = B-G;
        % store Y,U,V values
        C(y,x,1) = Y;
        C(y,x,2) = U;
        C(y,x,3) = V;
        D(y,x) = U;
        if (U>18 && U<45)
            E(y,x) = 255;
        else
            E(y,x) = 0;
        end
    end
end
F = im2bw(E);
F = imfill(F, 'holes');
G = imerode(F, strel('square', 5));

figure(5), imshow(A);
figure(6), imshow(G);

[L, n] = bwlabel(G);

face_region = regionprops(L, 'Area');
face_area = [face_region.Area];

face_idx = find(face_area == (1)*max(face_area));
face_shown = ismember(L, face_idx);

FaceData = regionprops(face_shown, 'BoundingBox');
boxes = cat(1, FaceData.BoundingBox);

rect = boxes(1,:);
Face_gray = rgb2gray(imcrop(A, rect));
figure(2), imshow(Face_gray), title('Cropped Face Image');
```

FPGA code

```
// ********** Display Options *********************************** //
assign VGA_Red =  (sw_9 && sw_8) ? mRed:
                  (~sw_9 && sw_8) ? raw_R :
                  (sw_9 && ~sw_8) ? fltr3_R: fltr2_R;
// fltr3 displays final result, fltr2 displays result after spatial
filtering

assign VGA_Green = (sw_9 && sw_8)? mGreen :
                  (~sw_9 && sw_8)? raw_G:
                  (sw_9 && ~sw_8)? fltr3_G: fltr2_G;

assign VGA_Blue = (sw_9 && sw_8)? mBlue:
                  (~sw_9 && sw_8)? raw_B:
                  (sw_9 && ~sw_8)? fltr3_B: fltr2_B;
// ************************************************************* //

// ********** input Options ************************************ //
assign temp = ((mRed*299)+(mGreen*587)+(mBlue*114))/1000;


always@(posedge TD_CLK27)
begin

// ********** Registering X, Y coordinates ********************** //
      VGA_X1 <= VGA_X;
      VGA_Y1 <= VGA_Y;
// ********************************************************* //

      if (~KEY[1])
      begin
            data_reg1 <= 'd0;

            sumX <= 30'b0;
            sumY <= 30'b0;
            cntr <= 19'b0;
            sumX_L <= 30'b0;
            sumY_L <= 30'b0;
            cntr_L <= 19'b0;
            sumX_R <= 30'b0;
            sumY_R <= 30'b0;
            cntr_R <= 19'b0;

            avgX_lpf <= avgX;
            avgY_lpf <= avgY;
            avgX_L2 <= avgX_L;
            avgY_L2 <= avgY_L;
            avgX_R2 <= avgX_R;
            avgY_R2 <= avgY_R;
            fltr_reg <= 16'd0;
      end
      else
      begin

// ********** Thresholding ************************************** //
```

```
//REd
if(csw==2'b00) begin
      if (((mRed-temp) > 10'd50) && ((mRed-temp) < 10'd500)) begin
            //raw result
            raw_R <= 10'h3FF;
            raw_G <= 10'h3FF;
            raw_B <= 10'h3FF;
            data_reg1[VGA_X1] <= 1'b1; // * Update the bit according
to the X coordinate * //
      end
      else begin
            //raw result
            raw_R <= 10'h0;
            raw_G <= 10'h0;
            raw_B <= 10'h0;
            data_reg1[VGA_X1] <= 1'b0; // * Update the bit according
to the X coordinate * //
      end
end
//Green
if(csw==2'b01) begin
      if (((mGreen-temp) > 10'd50) && ((mGreen-temp) < 10'd500))
begin
            //raw result
            raw_R <= 10'h3FF;
            raw_G <= 10'h3FF;
            raw_B <= 10'h3FF;

            data_reg1[VGA_X1] <= 1'b1; // * Update the bit according
to the X coordinate * //
      end
      else begin
            //raw result
            raw_R <= 10'h0;
            raw_G <= 10'h0;
            raw_B <= 10'h0;

            data_reg1[VGA_X1] <= 1'b0; // * Update the bit according
to the X coordinate * //
      end
end
//blue
if(csw==2'b10) begin
      if (((mBlue-temp) > 10'd50) && ((mBlue-temp) < 10'd500)) begin
            //raw result
            raw_R <= 10'h3FF;
            raw_G <= 10'h3FF;
            raw_B <= 10'h3FF;

            data_reg1[VGA_X1] <= 1'b1; // * Update the bit according
to the X coordinate * //
      end
      else begin
            //raw result
            raw_R <= 10'h0;
            raw_G <= 10'h0;
            raw_B <= 10'h0;

            data_reg1[VGA_X1] <= 1'b0; // * Update the bit according
to the X coordinate * //
      end
end

//skin color
if(csw==2'b11) begin
      if (((mRed-mGreen) > 10'd40) && ((mRed-mGreen) < 10'd296))
begin
            //raw result
            raw_R <= 10'h3FF;
            raw_G <= 10'h3FF;
            raw_B <= 10'h3FF;

            data_reg1[VGA_X1] <= 1'b1; // * Update the bit according
to the X coordinate * //
      end
      else begin
            //raw result
            raw_R <= 10'h0;
            raw_G <= 10'h0;
            raw_B <= 10'h0;

            data_reg1[VGA_X1] <= 1'b0; // * Update the bit according
to the X coordinate * //
      end
end

// ********* Spatial Filtering ******************************** //
            if (VGA_X1 == 10'b0) begin
                  data_reg2 [639:0] <= data_reg1 [639:0];
                  data_reg3 [639:0] <= data_reg2 [639:0];
                  data_reg4 [639:0] <= data_reg3 [639:0];
                  data_reg5 [639:0] <= data_reg4 [639:0];
                  data_reg6 [639:0] <= data_reg5 [639:0];
                  data_reg7 [639:0] <= data_reg6 [639:0];
                  data_reg8 [639:0] <= data_reg7 [639:0];
                  data_reg9 [639:0] <= data_reg8 [639:0];
                  data_reg10 [639:0] <= data_reg9 [639:0];

            end

            if ((data_reg2[VGA_X1-'d4]+data_reg2[VGA_X1-
'd3]+data_reg2[VGA_X1-'d2]+data_reg2[VGA_X1-'d1]+data_reg2[VGA_X1]
                  +
data_reg2[VGA_X1+'d1]+data_reg2[VGA_X1+'d2]+data_reg2[VGA_X1+'d3]+da
ta_reg2[VGA_X1+'d4]
                  + data_reg3[VGA_X1-'d4]+data_reg3[VGA_X1-
'd3]+data_reg3[VGA_X1-'d2]+data_reg3[VGA_X1-'d1]+data_reg3[VGA_X1]
                  +
data_reg3[VGA_X1+'d1]+data_reg3[VGA_X1+'d2]+data_reg3[VGA_X1+'d3]+da
ta_reg3[VGA_X1+'d4]
                  + data_reg4[VGA_X1-'d4]+data_reg4[VGA_X1-
'd3]+data_reg4[VGA_X1-'d2]+data_reg4[VGA_X1-'d1]+data_reg4[VGA_X1]
```

```
                  +
data_reg4[VGA_X1+'d1]+data_reg4[VGA_X1+'d2]+data_reg4[VGA_X1+'d3]+da
ta_reg4[VGA_X1+'d4]
                  + data_reg5[VGA_X1-'d4]+data_reg5[VGA_X1-
'd3]+data_reg5[VGA_X1-'d2]+data_reg5[VGA_X1-'d1]+data_reg5[VGA_X1]
                  +
data_reg5[VGA_X1+'d1]+data_reg5[VGA_X1+'d2]+data_reg5[VGA_X1+'d3]+da
ta_reg5[VGA_X1+'d4]
                  + data_reg6[VGA_X1-'d4]+data_reg6[VGA_X1-
'd3]+data_reg6[VGA_X1-'d2]+data_reg6[VGA_X1-'d1]+data_reg6[VGA_X1]
                  +
data_reg6[VGA_X1+'d1]+data_reg6[VGA_X1+'d2]+data_reg6[VGA_X1+'d3]+da
ta_reg6[VGA_X1+'d4]
                  + data_reg7[VGA_X1-'d4]+data_reg7[VGA_X1-
'd3]+data_reg7[VGA_X1-'d2]+data_reg7[VGA_X1-'d1]+data_reg7[VGA_X1]
                  +
data_reg7[VGA_X1+'d1]+data_reg7[VGA_X1+'d2]+data_reg7[VGA_X1+'d3]+da
ta_reg7[VGA_X1+'d4]
                  + data_reg8[VGA_X1-'d4]+data_reg8[VGA_X1-
'd3]+data_reg8[VGA_X1-'d2]+data_reg8[VGA_X1-'d1]+data_reg8[VGA_X1]
                  +
data_reg8[VGA_X1+'d1]+data_reg8[VGA_X1+'d2]+data_reg8[VGA_X1+'d3]+da
ta_reg8[VGA_X1+'d4]
                  + data_reg9[VGA_X1-'d4]+data_reg9[VGA_X1-
'd3]+data_reg9[VGA_X1-'d2]+data_reg9[VGA_X1-'d1]+data_reg9[VGA_X1]
                  +
data_reg9[VGA_X1+'d1]+data_reg9[VGA_X1+'d2]+data_reg9[VGA_X1+'d3]+da
ta_reg9[VGA_X1+'d4]
                  + data_reg10[VGA_X1-'d4]+data_reg10[VGA_X1-
'd3]+data_reg10[VGA_X1-'d2]+data_reg10[VGA_X1-
'd1]+data_reg10[VGA_X1]
                  +
data_reg10[VGA_X1+'d1]+data_reg10[VGA_X1+'d2]+data_reg10[VGA_X1+'d3]
+data_reg10[VGA_X1+'d4]) > 7'd75) begin
                  fltr2 <= 10'h3FF;
                  fltr2_R <= 10'h3FF;
                  fltr2_G <= 10'h3FF;
                  fltr2_B <= 10'h3FF;
            end

            else begin
                  fltr2 <= 10'h0;
                  fltr2_R <= 10'h0;
                  fltr2_G <= 10'h0;
                  fltr2_B <= 10'h0;
            end
// ************************************************************ //

// **** Computing centroid for all detected pixels *********** //
            if ((VGA_X1 > 10'd20) && (VGA_X1 < 10'd620) && (VGA_Y1 >
10'd20) && (VGA_Y1 < 10'd460)) begin
                  if (fltr3 == 10'h3FF) begin
                        sumX <= sumX + VGA_X1;
                        sumY <= sumY + VGA_Y1;
                        cntr <= cntr + 19'b1;
                  end
            end

            if ((VGA_X1 == 10'd2) && (VGA_Y1 == 10'd478)) begin
                  avgX <= sumX / cntr;
                  avgY <= sumY / cntr;
                  avgX_lpf <= avgX_lpf - (avgX_lpf >> 'd2) + (avgX
>> 'd2);
                  avgY_lpf <= avgY_lpf - (avgY_lpf >> 'd2) + (avgY
>> 'd2);
                  sumX <= 30'b0;
                  sumY <= 30'b0;
                  cntr <= 19'b0;
            end
// ************************************************************ //

// ********** Temporal Filtering ******************************* //
            if ((VGA_X1 < avgX_lpf + 10'd5) && (VGA_X1 > avgX_lpf -
10'd5) && (VGA_Y1 < avgY_lpf - 10'd5) && (VGA_Y1 > 10'd5)) begin
                  fltr_reg <= fltr_reg + fltr3[0];
            end
            else if ((VGA_X1 == 10'd600) && (VGA_Y1 == 10'd400))
begin
                  fltr_reg <= 16'd0;
            end

            if (fltr_reg > 16'd50) begin
                  if (avg2 > 10'b1110111111) begin // can also try
b1110110111
                        fltr3 <= 10'h3FF;
                        fltr3_R <= 10'h3FF;
                        fltr3_G <= 10'h3FF;
                        fltr3_B <= 10'h3FF;
                        // Draw centroid
                        if (cntr > 19'd500) begin // Set the
threshold so when #pixels is too small, nothing will be detected
                              if ((VGA_X1 < avgX_lpf + 10'd20) &&
(VGA_X1 > avgX_lpf - 10'd20) &&
                                  (VGA_Y1 < avgY_lpf + 10'd20) &&
(VGA_Y1 > avgY_lpf - 10'd20)) begin
                                    fltr3_R <= 10'h3FF;
                                    fltr3_G <= 10'h0;
                                    fltr3_B <= 10'h0;
                              end
                        end
                  end
                  else begin
                        fltr3 <= 10'h0;
                        fltr3_R <= 10'h0;
                        fltr3_G <= 10'h0;
                        fltr3_B <= 10'h0;
                        if (cntr > 19'd500) begin
                              if ((VGA_X1 < avgX_lpf + 10'd20) &&
(VGA_X1 > avgX_lpf - 10'd20) &&
                                     (VGA_Y1 < avgY_lpf + 10'd20) &&
(VGA_Y1 > avgY_lpf - 10'd20)) begin
                                    fltr3_R <= 10'h3FF;
                                    fltr3_G <= 10'h0;
                                    fltr3_B <= 10'h0;
                              end
                        end
                  end
            end
            else begin
                  if (avg2 > 10'b1110111111) begin //before:
b1110111111, (2) b1110110111
                        fltr3 <= 10'h3FF;
                        fltr3_R <= 10'h3FF;
                        fltr3_G <= 10'h3FF;
                        fltr3_B <= 10'h3FF;
                        // Draw centroid
                        if (cntr > 19'd500) begin // Set the
threshold so when #pixels is too small, nothing will be detected
                              if ( ((VGA_X1 < avgX_R2 + 10'd10) &&
(VGA_X1 > avgX_R2 - 10'd10) &&
                                       (VGA_Y1 < avgY_R2 + 10'd10)
&& (VGA_Y1 > avgY_R2 - 10'd10)) ||
                                      ((VGA_X1 < avgX_L2 + 10'd10)
&& (VGA_X1 > avgX_L2 - 10'd10) &&
                                       (VGA_Y1 < avgY_L2 + 10'd10)
&& (VGA_Y1 > avgY_L2 - 10'd10)) ) begin
                                    fltr3_R <= 10'h0;
                                    fltr3_G <= 10'h0;
                                    fltr3_B <= 10'h3FF;
                              end
                        end
                  end
                  else begin
                        fltr3 <= 10'h0;
                        fltr3_R <= 10'h0;
                        fltr3_G <= 10'h0;
                        fltr3_B <= 10'h0;
                        if (cntr > 19'd500) begin
                              if ( ((VGA_X1 < avgX_R2 + 10'd10) &&
(VGA_X1 > avgX_R2 - 10'd10) &&
                                       (VGA_Y1 < avgY_R2 + 10'd10)
&& (VGA_Y1 > avgY_R2 - 10'd10)) ||
                                      ((VGA_X1 < avgX_L2 + 10'd10)
&& (VGA_X1 > avgX_L2 - 10'd10) &&
                                       (VGA_Y1 < avgY_L2 + 10'd10)
&& (VGA_Y1 > avgY_L2 - 10'd10)) ) begin
                                    fltr3_R <= 10'h0;
                                    fltr3_G <= 10'h0;
                                    fltr3_B <= 10'h3FF;
                              end
                        end
                  end
            end

      end //else
end //always
```