School of Electrical and Electronic Engineering

University of Manchester

Final year project

Android Application for Sign Language Recognition

Vishisht M. Tiwari

9109605

supervised by

Dr. Hujun Yin

May, 2017

Acknowledgement

I would like to take this opportunity to express my appreciation for my supervisor Dr Hujun Yin for his constructive suggestion, advises and his support in helping me complete this project.

I would also like to thank my friends Vartul Agrawal, Divij Gupta, Bawika Punshi, Tashi Wangyal, Abhishek Chakrobarty, Hartaj Singh Trehan and Ananya Gupta for helping me develop the training sets used in this project.

Contents

List of Figures v

List of Tables vii

Abstract viii

1 Introduction 1

1.1 Aims and Objectives 2
1.2 Changes made from progress report [1] 3
1.2.1 Changing from British Sign Language to American Sign Language . 3
1.2.2 Locating fingertips instead of skeletonising fingers 3
1.2.3 Not using machine learning techniques to determine gestures 4

2 Literature Review 4

2.1 American Sign Language 4
2.2 OpenCV 5
2.3 Image pre-processing 6
2.3.1 Colour based Image segmentation 6
2.3.2 Background subtraction 7
2.4 Morphological operations 7
2.4.1 Dilate 8
2.4.2 Erode 8
2.5 Contours 8
2.6 Moments and Centroid 8
2.7 Convex Hull and Convexity Defects 9
2.8 Existing sign language recognition research 10
2.8.1 Real-Time Static Hand Gesture Recognition for American Sign Language (ASL) in Complex Background [1, 3] 10
2.8.2 A fingertips detection method based on the combination of the centroid and Harris corner algorithm [10] 12

3 Methodology and Implementation 18

3.1 Skin Colour segmentation using MATLAB 18
3.2 Skin Colour segmentation on static images 18
3.3 Skin Colour segmentation in real-time 19
3.4 OpenCV 20
3.5 Hand segmentation using Java 20
3.5.1 Skin Colour segmentation 20
3.5.2 Background Subtraction 21
3.5.3 Final Hand Segmentation Technique Used 22
3.6 Morphological Operations 22
3.6.1 Gaussian Blurring 23
3.6.2 Dilation 23
3.6.3 Erosion 24
3.7 Obtaining the largest contour 24
3.7.1 Biggest Contour 24
3.8 Extracting Contour Information 25
3.8.1 Moments and Centroid 25
3.8.2 Convex Hull and Convexity Defects 26
3.8.3 Removing false positives in convex hull 27
3.9 Lines connecting the centroid with the vertices 29
3.10 Recognising Gestures 30
3.10.1 Creating dataset 30
3.10.2 Comparing the data obtained from current frame with dataset . . . 31
3.10.3 Final program on computer 33
3.10.4 Transformation to an android application 35

4 Test Results and Analysis 36

4.1 Results of the demo software created on computer 36
4.2 Results obtained from the android application 37
4.3 Analysis 39

4.4 Performance of other projects detecting American Sign Language 39
4.4.1 1st project: Real-Time Static Hand Gesture Recognition for American Sign Language (ASL) in Complex Background [3] 40
4.4.2 2nd project: Appearance Based Recognition of American Sign Language Using Gesture Segmentation [12] 40
4.4.3 Comparing results of the discussed projects with the application developed in the project 41

5 Conclusion **42**

6 Further Work **43**

7 References **45**

Appendix A- Program Code **47**

Appendix B- Progress Report **77**

List of Figures

2.1 American Sign Language alphabetical symbols. Image obtained from Journal of Signal and Information Processing [3] 5
2.2 Left figure is a convex hull while the right figure is not a convex hull. Image obtained from Computational Geometry Lecture Notes HS 2013 [9] 10
2.3 After image pre-processing. Image obtained from Journal of Signal and Information Processing [3] 11
2.4 Gaussian Probability Distribution. Image obtained from 17th IEEE/ACIS International Conference [10] 13
2.5 Binary image based on two-dimensional Gaussian probability distribution. Image obtained from 17th IEEE/ACIS International Conference [10] 14
2.6 Harris corner detection space. Image obtained from 17th IEEE/ACIS International Conference [10] 15
2.7 Left- Feature points evaluated using Harris corner, Right- Convex hull formed using the feature points. Image obtained from 17th IEEE/ACIS International Conference [10] 16
2.8 Removing false feature points. Image obtained from 17th IEEE/ACIS International Conference [10] 17
3.1 Colour Segmentation on static image using MATLAB 19
3.2 Colour Segmentation on a video stream using MATLAB 19
3.3 Binary image using HSV colour space model segmentation 21
3.4 Left- Original Image, Centre- Image before dilation, Right- Image after dilation 23
3.5 Left- Original Image, Centre- Image before erosion, Right- Image after erosion 24
3.6 Left- Before obtaining largest contour Right- After obtaining largest contour 25
3.7 Blue circle drawn by the software program showing the centroid of the hand 26
3.8 Green lines show convex hull and red spots show the farthest convexity defect 27
3.9 Red circles specifying vertices of the convex hull. Left- With distance thresholding Right- Without distance thresholding 28

3.10 Left- Vertices detected between the black lines, Right- No region for vertex region specified 29
3.11 Lines connecting the centroid and the fingertips 30
3.12 Snapshot of the dataset containing the lengths between fingers for different gestures 31
3.13 UML flowchart explaining the comparisons with the dataset 32
3.14 Flowchart explaining all the steps of image processing 33
3.15 Snapshot of the final program detecting the gesture 'B' of American sign language 34
3.16 'B' gesture in American sign language detected using the android application. 36

List of Tables

2.1 Experimental Results. Data obtained from 17th IEEE/ACIS International Conference [10] 18
4.1 Experimental Results from the software developed on computer. Images obtained from http://lifeprint.com/asl101/topics/wallpaper1.htm [11] . . . 37
4.2 Experimental Results from android application. Images obtained from http://lifeprint.com/asl101/topics/wallpaper1.htm [11] 38
4.3 Experimental Results from the first project for comparison. Data obtained from Journal of Signal and Information Processing [3] 40
4.4 Experimental Results from the second project for comparison. Data obtained from International Journal on Computer Science and Engineering [12] 41
4.5 Comparison of overall recognition rate of different projects 42

Abstract

With a rising demand in disability solutions, sign language recognition has become an important research problem in the field of computer vision. Current sign language detection systems, however, lack the basic characteristics like accessibility and affordability which is necessary for people with speech disability to interact with everyday environment. This project focuses on providing a portable and affordable solution for understanding sign languages by developing an android application. The report provides a summarisation of the basic theory and steps involved in developing this android application that uses gesture recognition to understand alphabets of American sign language. The project uses different image processing tools to separate hand from the rest of the background and then implements pattern recognition techniques for gesture recognition. A comprehensive summarisation of the results obtained from the various tests performed has also been provided to illustrate the efficacy of the application.

1 Introduction

Technology can very well be regarded as the solution to almost all human problems. From the invention of wheel that helped humans in the basic needs of travelling to the latest technological inventions in artificial intelligence that helps in auto manoeuvring of cars and planes, technology has always served as the medium that reduces human effort. In recent years, technology has also been influential in making the lives of disabled people easier and bringing them closer to the society.[1]

Many technologies around the world such as self-driving cars, automatic wheel chairs and 3-D printed limbs, have helped disabled people live an easier life. One such field of engineering which is helping disabled people is computer vision. Computer vision refers to extraction, analysis and understanding of information from an image or from each frame of a video stream.[1]

Vision is an important sense that humans use to interpret the environment. Analysis and understanding of surrounding environment in a short span of time makes it on one of the most important senses in human beings. A very important feature of human vision is the perception of depth which is achieved using a combination of information received from both the eyes.

For people with speech and hearing disability, vision perception becomes an even more important ability to interpret the environment. Sign languages become an important part of people with speech and hearing disability and sometimes is only way people can communicate with the society. This makes the problem of understanding sign language even more important and hence the field of computer vision even more purposeful.

This report provides a comprehensive discussion of the 3rd year project of developing an android application that uses gesture recognition to understand static gestures of American Sign Language. This is a bespoke project, chosen to meet the needs of people with speaking disabilities. Although many sign gestures recognition systems have been developed in the past, they lack the basic characteristics like accessibility and affordability. Previous sign recognition systems are implemented on computers and use 3D cameras and

gloves to detect sign language gestures. These systems, however, are not portable and hence cannot be used by speech impaired people to interact with society in everyday environment. With the latest advancement in mobile camera technologies, this project aims to provide a system which is more mobile than the present day technologies used for sign language gesture recognition.

The project tests and implements different image processing techniques to extract the hand from the rest of the background and eliminate background noises that can affect the results. The project then uses gesture recognition procedures to detect the specific gesture and translate it into text. The main challenge involved in a computer vision project for understanding sign language is depth perception. Because this project is using a single camera android phone to acquire images, depth perception, as is observed in case of human eye, is impossible to achieve in this project. Other challenges in image analysis include lighting variations, background noise and background motions. The report also provides a discussion regarding how these challenges were overcome in this project.

1.1 Aims and Objectives

The purpose of this project is to develop an android mobile application that uses gesture recognition to understand American sign language. The android application should use image processing techniques to distinguish the hand from the background and then identify the fingertips to determine the gestures.

The recognition is limited to only static gestures and hence is only used for detecting static American sign language alphabets. The android application should understand the gestures and show it in text form. The project has the following main tasks:

- Research about different Image Processing techniques
- Extract skin coloured pixels and convert every frame to binary image
- Extract hand pixels from rest of the hand be extracting the largest contour

- Identify fingertips and centre of the hand from the largest contour
- Using the distances between the fingers and centroid, recognise the gesture
- Display the gesture in textual form to show which alphabet is shown as a gesture

1.2 Changes made from progress report [1]

As the project progressed in 2nd semester, few changes have been introduced in the project. These changes have been discussed below in detail.

1.2.1 Changing from British Sign Language to American Sign Language

The project was initially designed to recognise British sign language that uses two hands for representing alphabets. This caused a problem while testing with a mobile phone as one hand was always preoccupied with holding the phone. Since American sign language only uses one hand for representing alphabets, hence the focus of the project was changed to detecting American sign language.

1.2.2 Locating fingertips instead of skeletonising fingers

After researching through different image processing techniques and implementing them in different conditions, it was observed that skeletonising fingers and hands was susceptible to noise due to changing light conditions. Hence a different technique is proposed where skin coloured pixels are extracted from the image and fingertips are located. This method is much less prone to background noise and hence is decided as the basis of gesture recognition in the project.

1.2.3 Not using machine learning techniques to determine gestures

The project is now more focused on detecting the fingertips and using these fingertips to identify gestures. Because of this change, machine learning techniques were not needed and hence were not used for detecting gestures. The gestures, now, are recognised using pattern recognition techniques that use the distances between fingertips and centre of the hand to identify gestures.

2 Literature Review

2.1 American Sign Language

American sign language(ASL) is a sign language used by the deaf community in the United States and Canada. Besides North America, dialects of American Sign Language is spoken in many countries around the world, including west Africa and southeast Asia. ASL originated from American School for the Deaf in Hartford, Connecticut in the early 19th century, and since then, has propagated widely via schools for the deaf and deaf community and is now spoken by almost 500,000 people around the globe. [2]

ASL signs have a large number of phonemic components and involve movements of face, hands and torso. Each sign is composed of a number of distinctive components and uses either one or both hands. Signs are differentiated depending on the orientation of the hand and its location on the body.

ASL possesses a set of 26 alphabet signs which can be used to spell out words from the English Language. 24 of these signs are static while 2 signs, 'J' and 'Z', involve movement of the hand. These signs operate on the basis of 19 hand-shapes of ASL.

Figure 2.1: American Sign Language alphabetical symbols. Image obtained from Journal of Signal and Information Processing [3]

2.2 OpenCV

OpenCV is an open source library that focuses on providing pre-developed functions for computer vision projects. Originally written in C and C++, the functions have also been converted for other languages such as Python, Java and MATLAB. The library provides helpful functions in not only the basic image processing techniques but also different gesture detection algorithms. Few applications of OpenCV library have been given below: [4]

- Facial Recognition
- Segmentation
- Augmented Reality
- Motion Tracking
- Gesture Recognition

OpenCV is a cross-platform library able to run on not only the desktop platforms such as Windows, Linux and macOS but also mobile platforms such as Android and iOS. [4]

This project uses different OpenCV libraries to first develop a hand gesture recognition program on a computer and is then redeveloped to be used in a hand gesture recognition android application.

2.3 Image pre-processing

2.3.1 Colour based Image segmentation

The process of dividing a digital image into different segments is referred as image segmentation. Image segmentation helps in easier analysis of images and hence is an important tool for image processing and pattern recognition. Usually, segmentation is used to determine edges and boundaries of certain objects, part of objects and surfaces. Segmentation uses a certain visual characteristic such as colour, texture, intensity etc. to distinguish pixels. [5]

Colour based image segmentation uses the colour feature of every pixel to differentiate meaningful pixels in an image. Colour based segmentation is important on the assumption that the pixel clusters with homogenous colours correspond to a relevant object. Because colour based segmentation uses different colour features for extractions, different colour spaces such as RGB(Red, Green, Blue), HSV(Hue, Saturation, Value), CMY(Cyan, Magenta, Yellow), XYZ and YUV can be used. [5]

The project uses HSV colour space model for image segmentation due to its insensitivity to lighting variations. HSV, standing for hue, saturation and brightness value, is the most common cylindrical-coordinate representations of points in RGB colour space model. Unlike RGB, HSV separates colour information (chroma) from intensity or lighting(luma). This colour model represents colours(hue) in terms of their shade(saturation) and brightness value(value). Each of the components in the HSV colour space model have been described below:

- Hue is expressed from 0 to 360 degrees described as hues of red(starting at 0), yellow(starting at 60), green(starting at 120), cyan(starting at 180), blue(starting

at 240) and magenta(starting at 300).

- Saturation represents the amount of grey in the colour of the pixel.
- Value describes the brightness or intensity of the pixel. [6]

2.3.2 Background subtraction

Background subtraction is another image pre-processing technique that extracts pixels of moving objects from static backgrounds. This tool is widely used in computer vision and pattern detection involving moving objects. Background subtraction generates a foreground mask that contains pixels of all the moving objects in an image. This is done by subtracting current frame from a background model containing only static objects. Background subtraction has various advantages over other image segmentation technique such as it is not affected by little variations in lighting, is more robust and does not give false positives due to similar coloured objects in the background. The major drawback, however, for background subtraction is the background should be constant and the object to be detected should be moving continuously.

2.4 Morphological operations

Morphological operators are image processing tools that perform operations based on the shape and morphology of an image. Morphological operators can be used for various applications such as uniting disjointed elements, removing noise and segregating individual objects. In morphological operations, the value of each pixel is based on comparison with its neighbours. The two most primitive morphological operators used in this project have been discussed below.

2.4.1 Dilate

Dilation is one of the most basic morphological operation that uses a local maximum operation. It uses a set of coordinate points known as structuring element value which increases the overall area of the foreground pixels and helps in smoothing concavities. Dilation is usually operated on binary images but sometimes can also be used on grayscale images. [7]

2.4.2 Erode

Erosion is the other most basic morphological operation along with dilation that uses a local minimum operation. Erosion also uses a set of coordinate points knows as structuring element which decreases the overall area of the foreground pixels and helps in smoothing protrusions. Erosion is also usually operated on binary images but can also be used on grayscale images. [8]

2.5 Contours

Contours are defined as a continuous curve joining all the points with the same characteristics in an image. Contours, in this project, refers to the continuous white features in the binary images.

2.6 Moments and Centroid

In physics, moment is an expression that determines how a physical object is located or arranged and can be used to determine the centroid of any object. In terms of image processing, moment is used to determine the centroid of a contour by assigning a certain weight, equal in intensity, to every pixel in the contour. To determine the centroid of the contour, both the zero order and first order moments around x and y axis are calculated. The zero-order moment is calculated as follows:

$$M_{00} = M = \sum_{x=0}^{w} \sum_{y=0}^{h} f(x, y) \tag{1}$$

The first-order moments are calculated as follows:

$$M_{10} = Sum_x = \sum_{x=0}^{w} \sum_{y=0}^{h} x f(x, y) \tag{2}$$

$$M_{01} = Sum_y = \sum_{x=0}^{w} \sum_{y=0}^{h} y f(x, y) \tag{3}$$

After obtaining values of moments from the above equations, centroid can be derived:

$$Centroid = (x_c, y_c) = \left(\frac{Sum_x}{M}, \frac{Sum_y}{M} \right) \tag{4}$$

2.7 Convex Hull and Convexity Defects

Convex hull of a set of points 'X' refers to the smallest intersection of all convex sets containing the points 'X'. An important property of a convex hull is that a line connecting any two points in 'X' should always lie completely inside the convex hull. The convex hull can be imagined as a stretched rubber band containing all the points in 'X', where the rubber band is stretched due to the outer most points of 'X'. In reference to image processing, convex hull refers to the smallest intersection of all convex sets containing a specific set of pixels. [9]

Convexity defects refers to the cavity in a convex hull which is not part of the original set of points. These points are included in the convex hull even though they are not part of the original set of points.

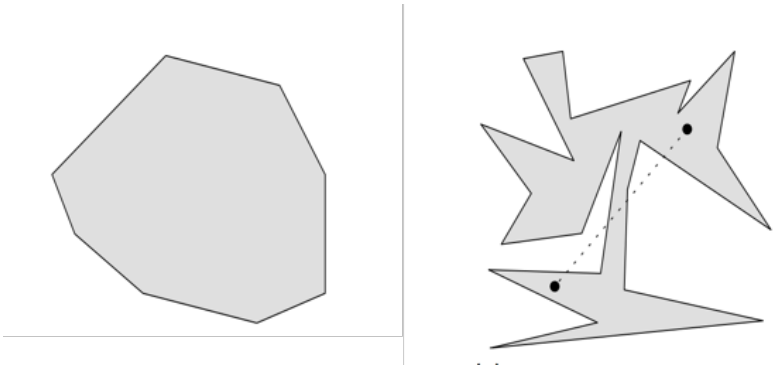


Figure 2.2: Left figure is a convex hull while the right figure is not a convex hull. Image obtained from Computational Geometry Lecture Notes HS 2013 [9]

2.8 Existing sign language recognition research

To understand the basics of sign language recognition, different image processing and pattern recognition methods were explored. This helped in understanding the approach that can be taken in completing the project. Few of the relevant papers studied for this purpose have been discussed below.

2.8.1 Real-Time Static Hand Gesture Recognition for American Sign Language (ASL) in Complex Background [1, 3]

This paper describes using a real-world RGB camera for real-time hand gesture recognition. The paper uses four stages for recognition consisting of image pre-processing, region extraction, feature extraction and feature matching. Gesture recognition is only used for recognising alphabetical symbols and is not used for any other complicated gestures. Each of these steps have been described below in detail.

Image Capturing An RGB image is captured first using a 10-megapixel web camera. Each image captured is of 160 x 120 dimension and has to be pre-processed to reduce computation time.

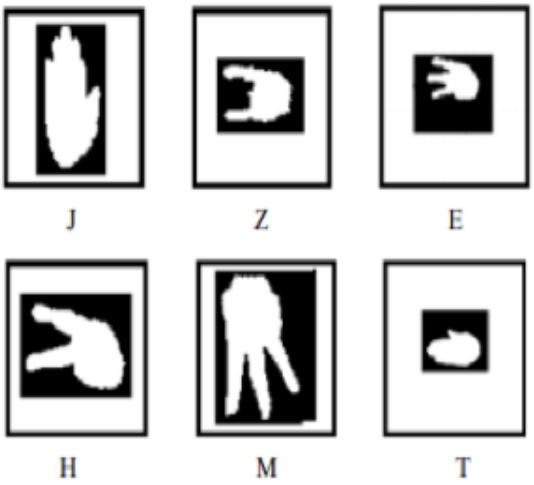


Figure 2.3: After image pre-processing. Image obtained from Journal of Signal and Information Processing [3]

Image Pre-processing For skin detection in the image, minimum and maximum skin probabilities of input RGB image are calculated. Data from normalised skin probability is used for producing grey threshold at zero which produces a binary image. Noise reduction is also achieved in this step by applying median filtering to the red channel to reduce "salt and pepper" noise. The binary image is then passed through Gaussian filer to smooth the image.

Region Extraction Connectivity of binary image is calculated using "bwlebel" method and "regionprops" is applied to find bounding box for the hand and to extract the region of interest.

Feature Extraction Feature vector is formed using centroid given by (Cx, Cy).

$$C_x, C_y = \frac{1}{n}\left(\sum_{t=1}^{n} X_t, \sum_{t=1}^{n} Y_t\right) \tag{5}$$

In equation 5, X_t and Y_t refers to the moment of each pixel in x-axis and y-axis

respectively and n refers to the zero-order moment. Area of the hand is calculated using the "bwarea" command in MATLAB.

Feature Matching Feature matching is done by calculating the Euclidean distance between feature vector of input real-time image and that of the training images.

$$E_D = \sqrt{(X_t - X_1)^2 + (Y_t - Y_1)^2} \tag{6}$$

The smallest Euclidean distance is used for gesture recognition.

Results This paper used 100 gestures for each alphabet and the success rate was measured depending on how many gestures were correctly recognised for each alphabet. The method received a high success rate with highest accuracy of 100% for letter E and V and lowest accuracy of 85% for letter I and U. [3]

2.8.2 A fingertips detection method based on the combination of the centroid and Harris corner algorithm [10]

Authors Jiajun Chen, Mengmeng Han, Shu Yang and Yuchun Chang provide a robust and innovative method of hand fingertip detection and identification in a complex background without the use of gloves. The program developed in the paper recognises the number of fingers displayed by detecting the number of fingertips. The paper describes 4 stages for finger detection and identification consisting of hand region detection, feature point detection, removing the neighbourhood points, fingertip detection and finger identification. Each of the aforementioned steps have been described below in detail.

Hand Region Segmentation The paper uses YCbCr colour space model for segmenting hand from the background. Cb and Cr are the blue and red-difference chroma components while Y describes the luminance component of the image. The paper relies

on the fact that skin colour will be different from the rest of the background and hence has a limitation that no skin colour object should be present in the background while the program is detecting and recognising hand gestures.

The paper regards the distribution of skin colour clustering in YCbCr space as two-dimensional Gaussian probability distribution.

$$P(C_b, C_r) = (K - M)C(K - M)^T \tag{7}$$

$$K = [C_b, C_r] \tag{8}$$

$$M = E(K) \tag{9}$$

$$P(C_b, C_r) = (K - M)C(K - M)^T \tag{10}$$

$$C = \begin{bmatrix} cov(x,x) & cov(x,y) \\ cov(y,x) & cov(y,y) \end{bmatrix} \tag{11}$$

In the above equations, M refers to the mean value of the sample pixels, C refers to the covariance of skin colour similarity model and K is the value of the sample pixels in the YC_bC_r colour space. 'x' and 'y' represents the positions of the pixel and 'z'($P(C_b, C_r)$) represents the probability.

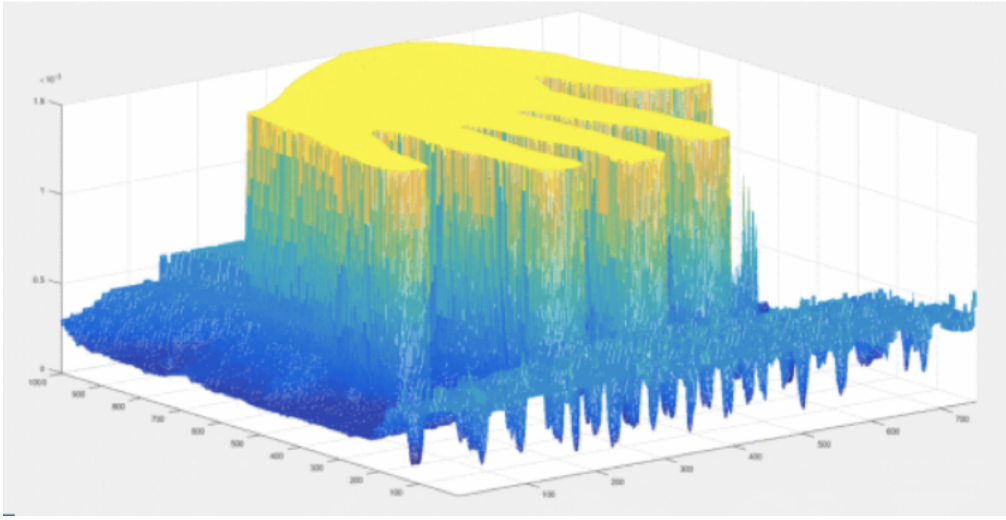


Figure 2.4: Gaussian Probability Distribution. Image obtained from 17th IEEE/ACIS International Conference [10]

Figure 2.4 shows the two-dimensional Gaussian probability distribution of a test image used in the paper. The program will consider a pixel as a hand region if the probability of a specific region is higher than a specific threshold.

The paper uses 50 test subjects under different lighting conditions to calculate the value of 'M' and 'C' for different hand gestures. Figure 2.5 shows a binary image of a hand based on two-dimensional Gaussian probability distribution.

Figure 2.5: Binary image based on two-dimensional Gaussian probability distribution. Image obtained from 17th IEEE/ACIS International Conference [10]

Feature Point Extraction Feature point is a position in an image that can be extracted robustly and Harris feature point is a feature point where the curvature is locally maximum. Harris corner detection algorithm uses the significant changes in image intensity in multiple direction to detect a corner. It is able to differentiate the corner from an edge where the image intensity changes in a certain direction only. This phenomenon is formulated by examining the change in intensity resulting from shifts in a local window. At a corner point, the image intensity changes greatly when the window is shifted in an arbitrary direction. However, at an edge point, the image intensity will change greatly when the window is shifted in perpendicular direction. Hence, Harris detector uses a second order moment matrix to detect corners.

For a particular image I, the auto-correlation matrix M at a point (x, y) is given as follows:

$$M(x,y) = \sum_{x,y} \omega(x,y) \begin{bmatrix} I_x^2(x,y) & I_x I_y(x,y) \\ I_x I_y(x,y) & I_y^2(x,y) \end{bmatrix} \tag{12}$$

Here I_x and I_y are the respective derivatives of pixel intensity in the x and y direction at point (x, y) and weighting function is the circular Gaussian weighting function given below.

$$\omega(x,y) = g(x,y,\sigma^2) exp\left(-\frac{x^2+y^2}{2\sigma}\right) \tag{13}$$

Matrix M(x, y) describes the shape of the autocorrelation measure as a result of shifts in window location. If λ_1 and λ_2 are the eigenvalues of M(x, y), the (λ_1, λ_2) space can be divided into three regions shown in figure 2.6

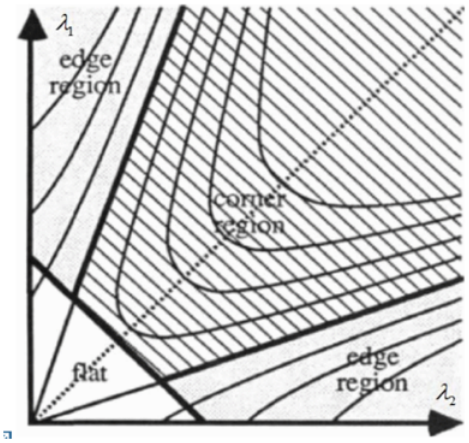


Figure 2.6: Harris corner detection space. Image obtained from 17th IEEE/ACIS International Conference [10]

The expression of the angular point T can be calculated as follows:

$$T(x,y) = \frac{\lambda_1 \lambda_2}{\lambda_1 + \lambda_2} \tag{14}$$

$$T_{relative}(x,y) = \frac{T(x,y)}{T_{max}(x,y)} \tag{15}$$

According to Harris algorithm, if the $T_{relative}$ value of target pixel is greater than the

threshold value then the pixel is one of the feature points.

After evaluating all the feature points, the program calculates the distance between each point and discards any feature point which is within a certain threshold distance of another feature point.

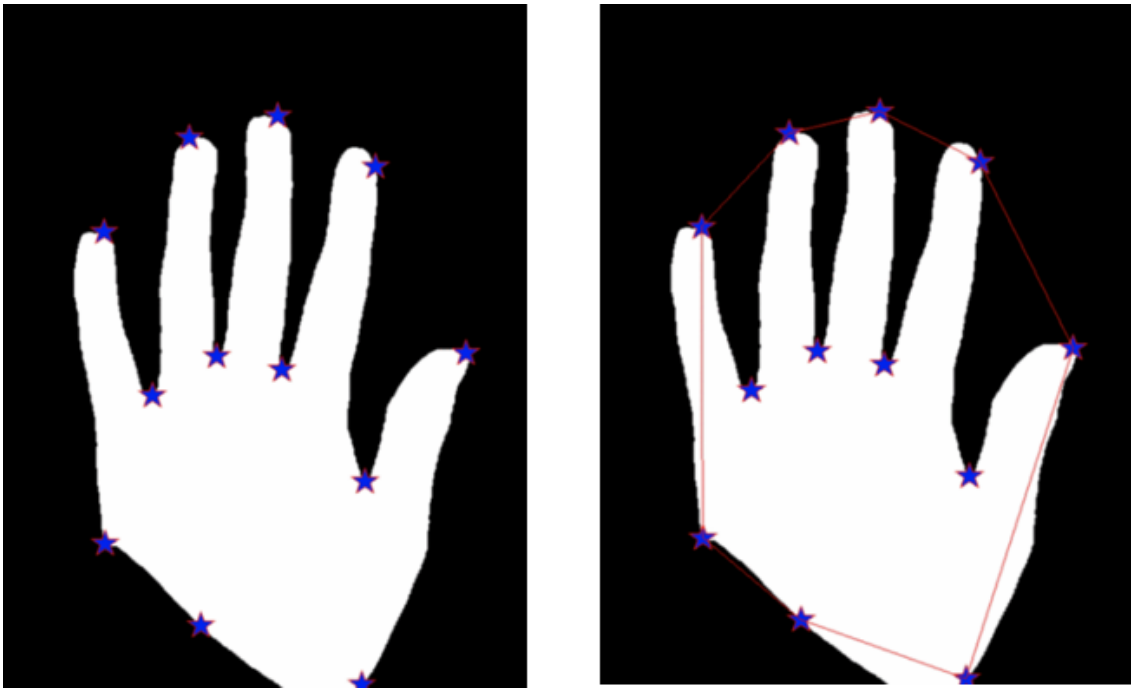


Figure 2.7: Left- Feature points evaluated using Harris corner, Right- Convex hull formed using the feature points. Image obtained from 17th IEEE/ACIS International Conference [10]

Fingertips Detection The paper uses a method of convex hull and centroid algorithm to detect fingertips. The program uses the feature points evaluated from the previous step to construct the convex hull. The convex hull constructed from all the feature points found before is shown in figure 2.7.

The program also calculates the centroid for all the pixels representing the hand in the image. The centroid is calculated as follows:

$$M_{total} = \sum_{x}\sum_{y} f\left(x, y\right) \tag{16}$$

$$M_x = \sum_x \sum_y [x * f(x, y)] \tag{17}$$

$$M_y = \sum_x \sum_y [y * f(x, y)] \tag{18}$$

$$X_{centroid} = \frac{M_x}{M_{total}} \tag{19}$$

$$Y_{centroid} = \frac{M_y}{M_{total}} \tag{20}$$

Here, f(x, y) refers to the pixel value which can be either 1 or 0 depending on the value of the pixel in the binary images.

The program then discards all the feature points, that are not part of the convex hull or are below the centroid as false features and hence successfully estimates the fingertips from all the feature points. This has been shown in figure 2.8.

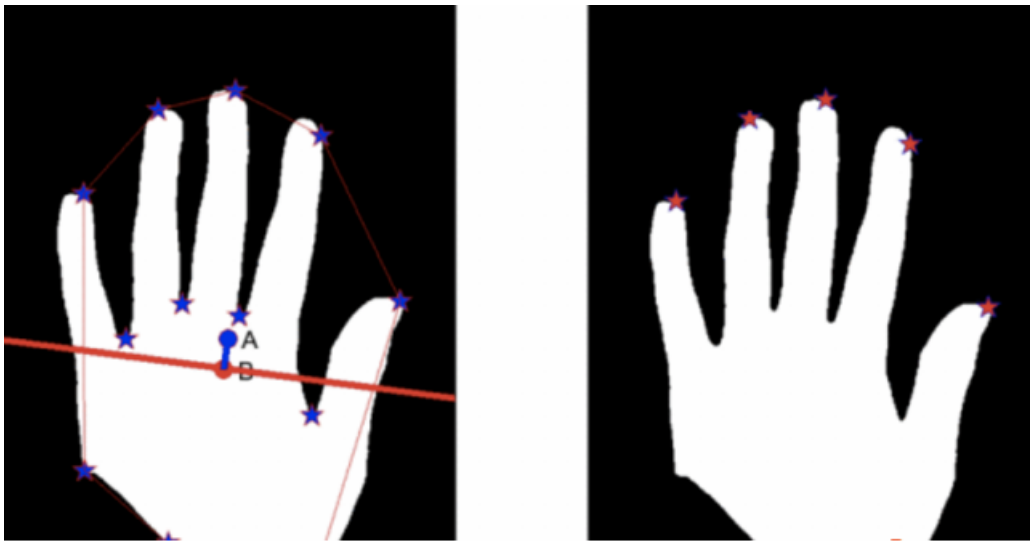


Figure 2.8: Removing false feature points. Image obtained from 17th IEEE/ACIS International Conference [10]

Fingertips Identification and Result The authors of the paper verify the performance of their software by detecting the number of fingers displayed by the test subjects. A total of 250 test images were collected from 50 different subjects under different conditions. The results are tabularised in table 2.1.

Table 2.1: Experimental Results. Data obtained from 17th IEEE/ACIS International Conference [10]

Number of fingers	Test images	Correct	Recognition rate
1	50	49	98%
2	50	48	96%
3	50	47	94%
4	50	46	92%
5	50	47	94%
ALL	250	237	94.8%

3 Methodology and Implementation

3.1 Skin Colour segmentation using MATLAB

To gain an initial understanding of the methods and processes that can be used to separate hand from rest of the background, the first version of the project was initiated in MATLAB. MATLAB was chosen for the initial version as it provides user friendly prebuilt functions for implementing image processing techniques. This initial version involved developing a program for colour based image segmentation which was used to distinguish the hand from the rest of the background.

3.2 Skin Colour segmentation on static images

The first step of the project was to develop a MATLAB program that separated the hands from rest of the background in a static image. HSV colour space was used to separate skin colour from the rest of the background. HSV colour space model was selected due to its insensitivity to the lighting conditions under which the image was produced. Minimum and a maximum threshold values for hue, saturation and brightness were selected to distinguish all range of skin colours from the background. The image was then converted into a binary image where all pixels with hue, saturation and brightness within the threshold range were changed to a value of 1(white) while all other pixels were changed to a value of 0(black). The result of this MATLAB program is shown in

figure 3.1.

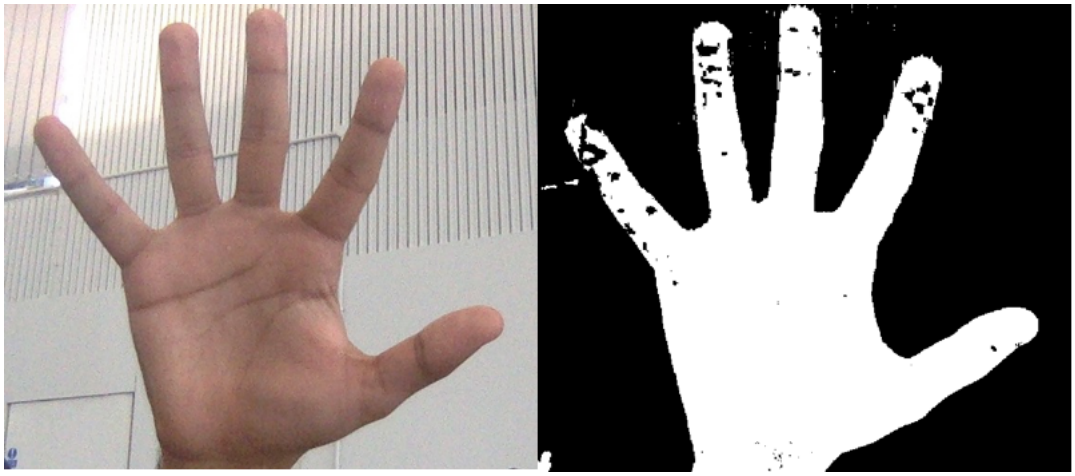


Figure 3.1: Colour Segmentation on static image using MATLAB

3.3 Skin Colour segmentation in real-time

The next step of the project was to modify the above program to work on a real-time video stream. The program implemented skin colour segmentation on every frame of the real-time video. The program used the built-in or attached webcam of the computer to distinguish all the skin coloured pixels from the background and convert the frames into binary frames. The result of this MATLAB program on one frame has been shown in figure 3.2.

Figure 3.2: Colour Segmentation on a video stream using MATLAB

3.4 OpenCV

OpenCV is an open source library that contains pre-developed functions for computer vision projects. OpenCV functions are also available in Java which is the programming language used in this project. This project uses OpenCV libraries for basic computer vision and image processing operations. A few of those operations have been mentioned below:

- RGB to HSV conversion
- Dilation and Erosion
- Blurring
- Finding contours
- Forming Convex hulls

3.5 Hand segmentation using Java

After creating a primitive version of the hand colour segmentation program in MATLAB, the project was then progressed onto Java. Java was chosen as the programming language to continue this project forward because Java is the foundation language for android application programming. Different hand segmentation techniques were attempted using prebuilt OpenCV functions. The techniques have been described below in detail.

3.5.1 Skin Colour segmentation

Skin colour segmentation uses a specific colour space model to distinguish pixels using a certain colour feature. The project uses an HSV colour space model to differentiate skin coloured pixels from the background. The skin colour segmentation implemented in Java is very similar to the program implemented in MATLAB but uses an OpenCV function to extract an HSV image from an RGB image. The right threshold values of hue, saturation

and brightness is chosen to detect skin colour. The HSV image was then converted into a binary image where the desired pixels were given a value 1(white) and the background pixels were given a value 0(black).

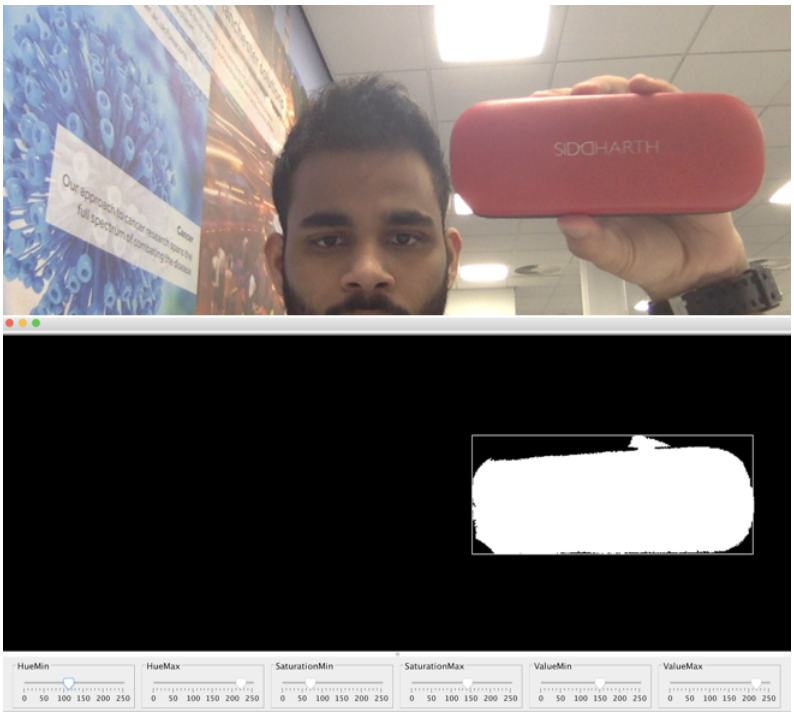


Figure 3.3: Binary image using HSV colour space model segmentation

3.5.2 Background Subtraction

Background subtraction is another image processing technique that was tested to distinguish moving fingertips from the background. Background subtraction generates a foreground mask that is calculated by subtracting the current frame from the background model containing only static images. Background subtraction was tried in this project as an alternative to skin colour segmentation because of the problems faced in skin colour segmentation due to similarly coloured objects in the background. Background subtraction was implemented using pre-built OpenCV functions. Three method of background subtraction that were tried and tested in this project are described below:

- Background Subtractor MOG: This is a Gaussian mixture background segmentation algorithm based on the paper "An improved adaptive background mixture model for

real-time tracking with shadow detection" by P.KadewTraKuPong and R. Bowden in 2001. The algorithm uses a method to model each background pixel by a mixture of K Gaussian distributions.

- Background Subtractor MOG2: This is another Gaussian mixture background segmentation algorithm based on two papers by Z.Zivkovic, "Improved adaptive Gaussian mixture model for background subtraction" in 2004 and "Efficient Adaptive density estimation per image pixel for the task of background subtraction" in 2006. This algorithm selects the appropriate number of Gaussian distribution for each pixel. An advantage of this method is that it provides adaptability to varying scenes due to illumination changes.

- Background Subtractor KNN: This method finds a group of k objects in the training set closest to the test object. It bases the assignment of a label on the predominance of a particular class in the neighbourhood.

All these methods convert normal RGB frame to a binary frame where all the moving object pixels are converted to a value 1(white) while all the static objects are converted to a value 0(black).

3.5.3 Final Hand Segmentation Technique Used

Despite the demerits of skin colour segmentation (such as susceptibility to varying light conditions and background noises) this method was chosen over background subtraction because background subtraction needs a static background for successful implementation. This project, however, will be implemented as a mobile application and hence would not have a static background.

3.6 Morphological Operations

Binary images can contain numerous imperfections due to background noises or varying conditions. Morphological operations are simple image processing techniques which can

be used to remove or reduce these imperfections. The project incorporates the three most primitive morphological operations. These operations have been discussed below.

3.6.1 Gaussian Blurring

In image processing, Gaussian blurring refers to blurring an image to remove detailing and noise. The project uses blurring to get rid of the low-level noise caused by disturbances in the background. The project uses blurring, right after converting RGB image to binary image using the HSV colour space. Gaussian blurring is implemented in this project using a prebuilt OpenCV function.

3.6.2 Dilation

Dilation is applied in binary images or in some cases to grey scale images to smooth concavities caused by noises in the background. It increases the overall area by using a local maximum operation. Dilation is implemented using a prebuilt OpenCV function and is applied in the project after gaussian blurring.

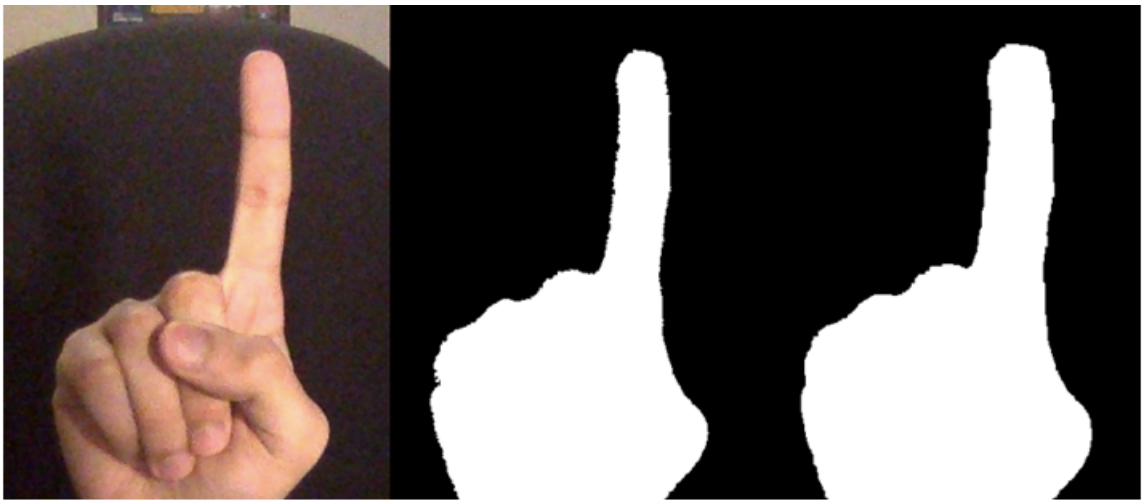


Figure 3.4: Left- Original Image, Centre- Image before dilation, Right- Image after dilation

3.6.3 Erosion

Conversely, erosion is applied to binary images or in some cases to grey scale images to remove the protrusions caused by noises and false results in the background. It erodes away the boundaries of the foreground pixels using a local minimum operation. Erosion is implemented using a prebuilt function in OpenCV and is applied on the binary image after dilation to get rid of further imperfections.

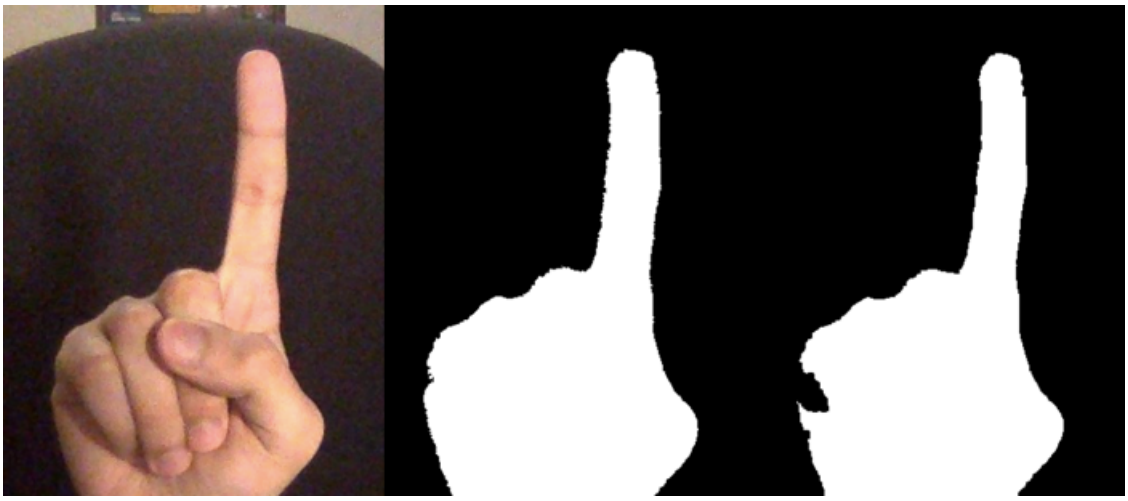


Figure 3.5: Left- Original Image, Centre- Image before erosion, Right- Image after erosion

3.7 Obtaining the largest contour

Contour refers to a curve formed by a continuous set of points. In terms of computer vision, a contour refers to a continuous set of pixels with similar visual attributes. This project refers to contours as a continuous set of white pixels.

3.7.1 Biggest Contour

Biggest contour is referred as the contour(curve) enclosing the largest area in the image. In a binary image, the biggest contour refers to the largest white feature in the image. Extracting the largest contour helps in removing noise and segregating relevant objects from other irrelevant objects in the background with similar visual characteristics. In the

project, the area of each of the contours is calculated using a prebuilt OpenCV function. The areas are then compared with each other and the largest contour is used as it is assumed the largest contour will contain the fingertips.

Figure 3.6: Left- Before obtaining largest contour Right- After obtaining largest contour

3.8 Extracting Contour Information

3.8.1 Moments and Centroid

Moments, in physics, refers to the arrangement of different particles in a physical body that can be used to understand its responses under different external factors. In image processing, moments are used to estimate the properties of a set of pixels by assuming each pixel as a particle. Each pixel is treated as the same and hence can be used to derive different properties of a set of pixels. Moments are then used to calculate the centroid of the foremost pixels which refers to the centre of mass of the set of pixels by assigning every pixel the same mass. The project uses prebuilt OpenCV functions to calculate the moments and centroid. The centroid here is used to find the centre of the biggest contour, in this case the hand, which is then used to derive different properties of the contour such as the distance of each fingertips from the centroid etc.

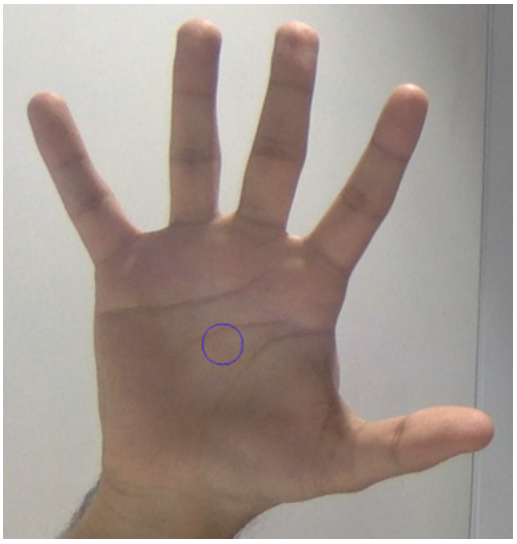


Figure 3.7: Blue circle drawn by the software program showing the centroid of the hand

3.8.2 Convex Hull and Convexity Defects

After detecting the hand and separating it from the background, the next step is to recognise the hand gestures. The project uses a concept of convex hull for recognition purposes. In mathematics, convex hull for a set of specific points 'X', refers to a convex set that contains all the points of 'X'. It can be imagined as a stretched rubber band containing all the points inside it. In terms of image processing, convex hull is a polygon with the smallest area that contains a specific set of pixels inside them. A polygon containing all the white feature pixels inside a polygon can be referred as a convex hull in a binary image. Convex hull is produced using prebuilt OpenCV functions to determine the fingertips and the basic hand shape which can then be used to understand the gesture. Each vertex of the convex hull can be assumed as the fingertip and the properties such as area, edge length, and length from centre can be used to determine the hand signs.

Convexity defects refers to all the points that are inside the convex hull but are not the part of the initial points that were used to construct the convex hull. The furthest convexity defect from the convex hull can also aid the overall program in detecting the hand gesture. [9]

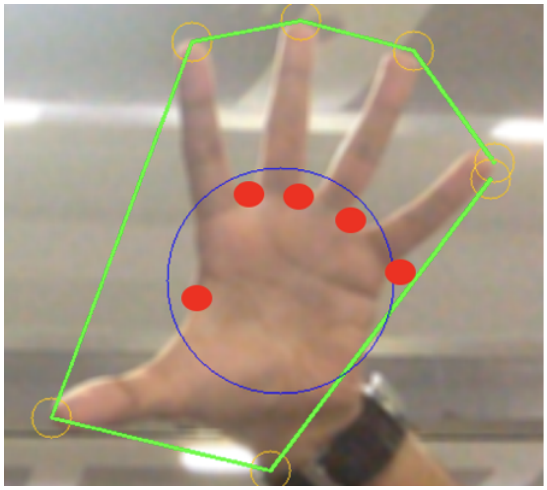


Figure 3.8: Green lines show convex hull and red spots show the farthest convexity defect

3.8.3 Removing false positives in convex hull

While constructing the convex hull around the hand, it was observed that, along with detecting the correct fingertips, the convex hull also had other false vertices which were not supposed to be used for the hand gesture recognition. The different types of false convex hull vertices have been described below:

- Vertices too close to each other- While detecting the hand, sometimes, the vertex were too close to each other and hence the program detected more than one finger tip on one finger.
- Convex hull including pixels of the forearm- In many cases, if the forearm of the test subject is visible, the program detected it as part of the hand which could compromise the hand gesture capabilities.

Different methods were implemented to remove the above mentioned false positives. The three methods used in this project has been described below.

Distance Thresholding To achieve one vertex of convex hull on one finger tip, a distance threshold value was set. This distance was the minimum distance possible between

any two vertices of the convex hull. This distance was specified in terms of pixels and was set at a value of 20 pixels. Hence any vertex which is within 20 pixels of another vertex was ignored and no false positives were observed on the same fingertip.

Figure 3.9: Red circles specifying vertices of the convex hull. Left- With distance thresholding Right- Without distance thresholding

Specifying certain area of the screen for vertex detection To make sure the forearms were not detected as part of the hand, the area of the screen was limited where a convex hull vertex could have been constructed. Because it was observed that in most cases, gestures using the fingertips would be shown only in the centre of the frame, a horizontal band was created within each frame. Vertices outside these bands were discarded as they were either part of the forearm or some other skin coloured object in the background.

However, it is important to note that, the program will be accompanied with an instruction explaining the user how to use the program. This will include the directive that the fingertips should always be inside the detection zone which will be indicated using two black lines as shown in figure 3.10.

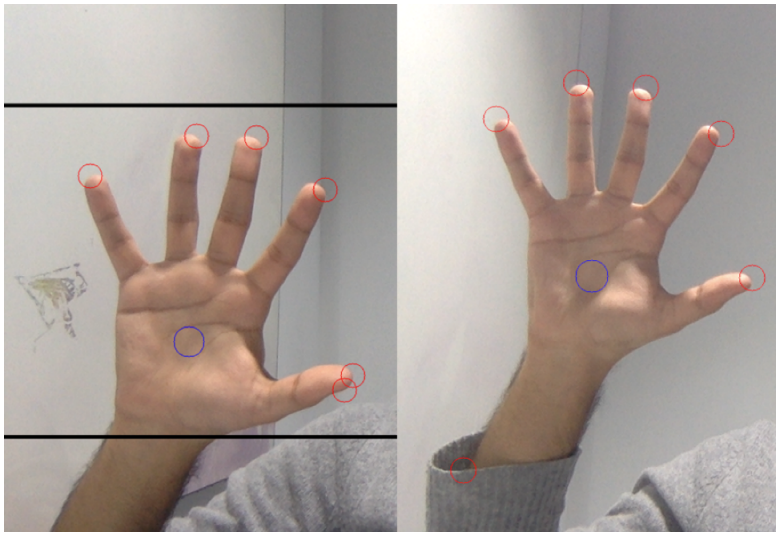


Figure 3.10: Left- Vertices detected between the black lines, Right- No region for vertex region specified

Limit on max vertices of the convex hull A limit on the number of vertices on the convex hull were also enforced to remove false positives. After trial and error, it was estimated that 20 vertices should be enough to determine the hand gesture and hence the program has a limit of 20 points on the vertices of the convex hull.

3.9 Lines connecting the centroid with the vertices

To help display the gestures in terms of processed images, the program shows connected lines between the centroid and the vertices of the convex hull as shown in figure 3.11. The program will use the lengths of these lines and the distances between each vertices of the convex hull to recognise the gesture. The different combinations of lengths of these lines will help to distinguish one gesture from another.

The distances between any two points are calculated using simple Pythagoras theorem shown below.

$$Length = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2} \tag{21}$$

Here (x_1, y_1) and (x_2, y_2) are the coordinates of the first and second point respectively.

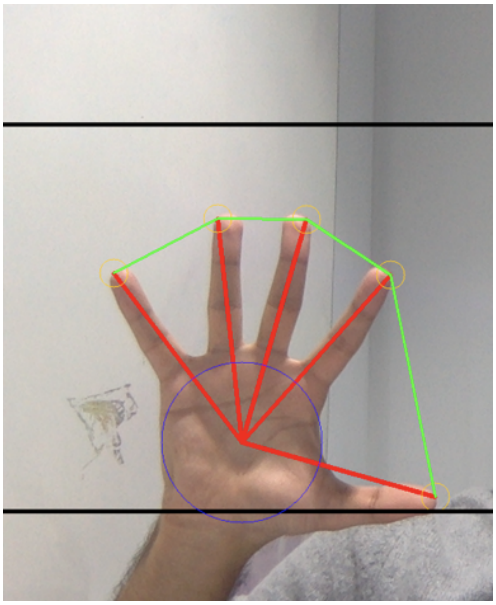


Figure 3.11: Lines connecting the centroid and the fingertips

3.10 Recognising Gestures

The program uses the lengths of the lines connecting the centroid and vertices of the convex hull to recognise gestures and distinguish it from other gestures. In case of some alphabets, the distance between two vertices of the convex hull is also used but this is done only in the following two cases:

- When the program is confusing the gesture of the alphabet with the gesture of some other alphabet
- When the gesture of the alphabet produces a high number of false positives

3.10.1 Creating dataset

A dataset was created containing the distances between the centroid and fingertips for all the gestures that need to be recognised. Twenty(20) different test subjects were used to collect the gestures for all the gestures and a special program was created that calculated the distance between the centroid and vertices for all the gestures obtained from the test

subjects. This data was then stored in the program as a double dimensional array. A snapshot of the double dimensional dataset has been shown in figure 3.12.

```
private double[][] distanceFromCentroid = {{100.00000, 97.12687, 86.61439, 121.50944, 102.95812, 100.00000, 89.17655, 125.10383, 115.45426, 112.137
        {100.00000, 92.29221, 78.04556, 105.51183, 108.35150, 100.00000, 84.56354, 114.32366, 128.13029, 118.25428, 100.00000, 135.19261, 94.77610,
        {100.00000, 99.23577, 115.11892, 100.77011, 100.00000, 116.00547, 86.86669, 86.20283, 100.00000},
        {100.00000, 97.97592, 106.72310, 108.07551, 102.06590, 100.00000, 108.92789, 110.30823, 93.70043, 91.80385, 100.00000, 101.26721, 92.52790,
        {100.00000, 98.19263, 83.54441, 115.75966, 106.85232, 101.84064, 100.00000, 85.08216, 117.89038, 108.81908, 119.69682, 117.53345, 100.00000
        {100.00000, 54.44577, 106.66054, 183.66901, 100.00000, 195.90235, 93.75539, 51.04584, 100.00000},
        {100.00000, 81.64063, 122.48803, 100.00000},
        {100.00000, 53.95330, 107.79975, 185.34546, 100.00000, 199.80194, 92.76460, 50.04956, 100.00000},
        {100.00000, 62.06874, 46.45068, 161.11169, 100.00000, 74.83747, 215.28211, 133.62290, 100.00000},
        {100.00000, 61.96105, 44.43716, 161.39172, 100.00000, 71.71790, 225.03686, 139.43520, 100.00000},
        {100.00000, 66.14439, 54.31321, 151.18440, 100.00000, 82.11309, 184.11728, 121.78326, 100.00000},
        {100.00000, 72.71501, 55.04103, 137.52319, 100.00000, 75.69419, 181.68263, 132.11054, 100.00000},
        {100.00000, 102.15016, 72.69978, 56.61859, 97.89510, 100.00000, 71.16952, 55.42682, 137.55200, 140.50959, 100.00000, 77.88000, 176.62045, 1
        {100.00000, 97.13032, 69.19025, 102.95446, 100.00000, 71.23445, 144.52903, 140.38152, 100.00000},
        {100.00000, 81.49701, 122.70388, 100.00000},
        {100.00000, 97.42741, 97.65941, 102.64052, 100.00000, 100.23813, 102.39668, 99.76243, 100.00000},
        {100.00000, 71.64574, 58.44599, 59.42751, 139.57565, 100.00000, 81.57636, 82.94634, 171.09815, 122.58453, 100.00000, 101.67938, 168.27223,
        {100.00000, 87.55052, 87.74699, 114.21976, 100.00000, 100.22440, 113.96402, 99.77610, 100.00000},
        {100.00000, 91.69165, 94.07383, 109.06118, 100.00000, 102.59803, 106.29949, 97.46776, 100.00000},
        {100.00000, 73.83600, 135.43528, 100.00000},
        {100.00000, 91.58468, 96.44940, 109.18856, 100.00000, 105.31171, 103.68131, 94.95620, 100.00000},
        {100.00000, 76.12349, 65.60153, 131.36550, 100.00000, 86.17777, 152.43548, 116.03920, 100.00000},
        {100.00000, 87.36679, 91.32631, 93.52067, 114.45997, 100.00000, 104.53206, 107.04373, 109.49747, 95.66443, 100.00000, 102.40277, 106.92823,
        {100.00000, 113.82281, 87.57989, 70.35113, 87.85585, 100.00000, 76.94406, 61.80759, 114.18147, 129.96455, 100.00000, 80.32795, 142.14413, 1
```

Figure 3.12: Snapshot of the dataset containing the lengths between fingers for different gestures

As discussed above, in some cases, an additional dataset was created that stored the distance between the two(2) vertices of the convex hull. This was only created for a few gestures and was generated from the gestures obtained from the twenty(20) subjects used before.

3.10.2 Comparing the data obtained from current frame with dataset

After developing the datasets, they were stored in the gesture recognition program in the format of arrays. The arrays were then compared with the combinations of value of distances obtained from the current frame. If the distances between the centroid and the vertices of the convex hull are within the set threshold values of any of the arrays in the dataset, the gesture is recognised as one of the gestures corresponding to the array. The recognised gesture is then shown on the screen. However, if the gesture recognised is a conflicting gesture, a further check is done. The distances between the vertices of the convex hull are then checked against the alternate dataset containing the values of distances between vertices. If the values are within 20 pixels of this dataset as well, the gesture is recognised. A use case flow diagram describing methodology of gesture recognition in each frame using comparisons of the gestures has been shown below.

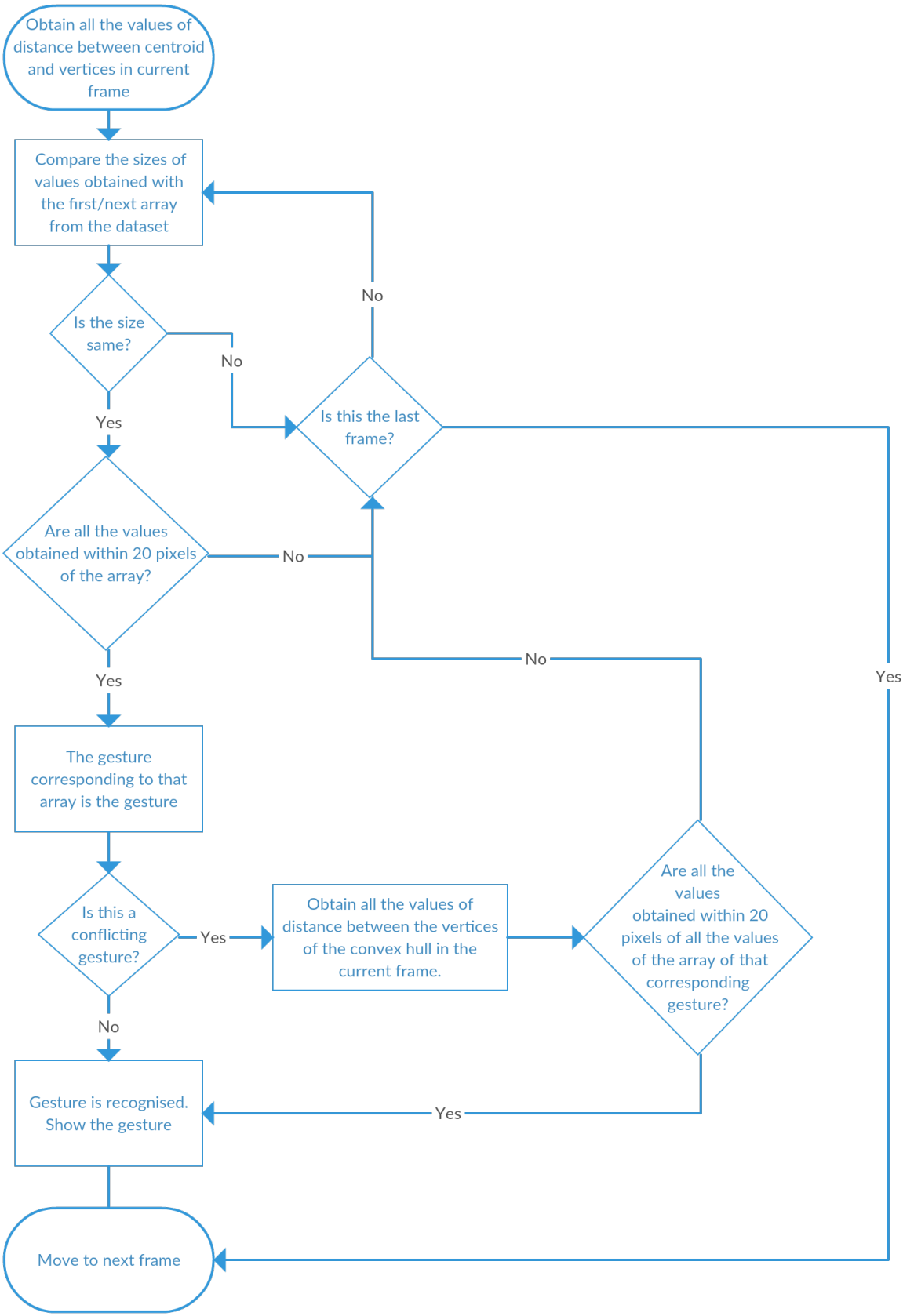


Figure 3.13: UML flowchart explaining the comparisons with the dataset

3.10.3 Final program on computer

A special UI (user interface) was created showing all the necessary information and settings needed for gesture recognition. The flowchart below provides a design idea of all the image processing techniques used in this project.

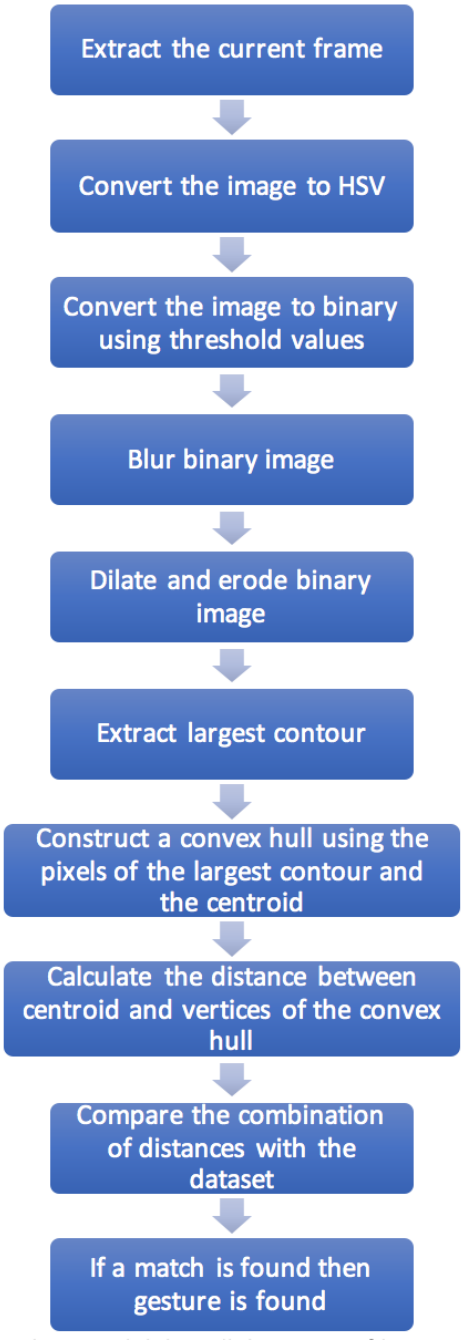


Figure 3.14: Flowchart explaining all the steps of image processing

The code for the user interface was integrated into the main gesture recognition program. A snapshot of the UI of the program has been shown in figure 3.15.

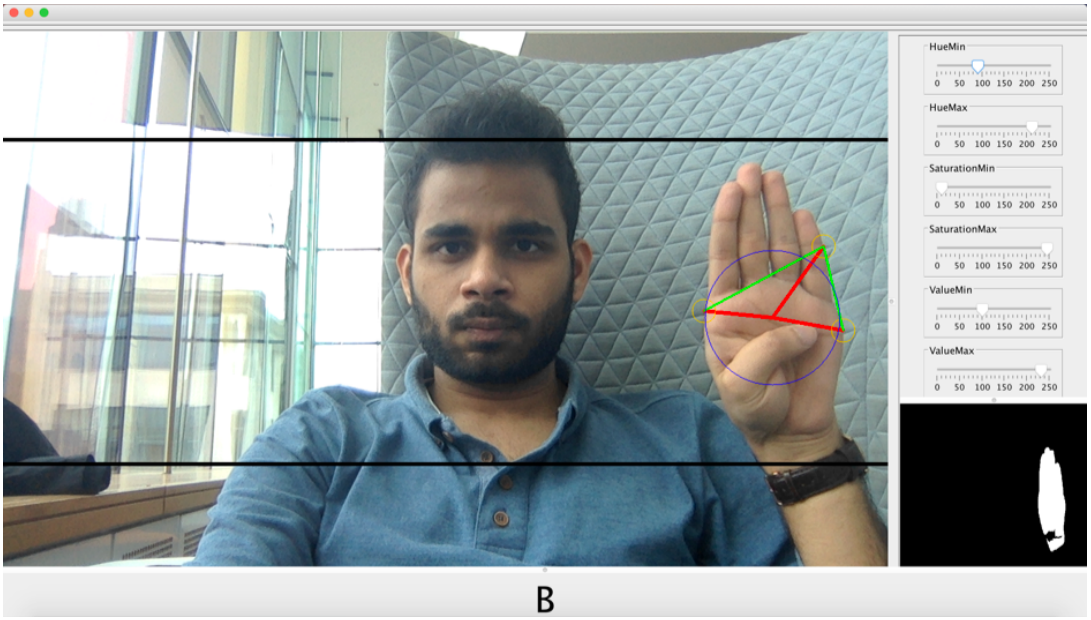


Figure 3.15: Snapshot of the final program detecting the gesture 'B' of American sign language

The different components of the UI have been described below:

- Main Screen: Main component of the UI shows the original video feed from the webcam along with the lines connecting the centroid and the vertices of the convex hull. Figure 3.15 shows the gesture for the alphabet 'B' in American sign language.
- Binary Image Screen: This component of the UI shows the binary images after the current frame has been processed. This binary image shows the largest contour after the skin colour components of the image have been extracted. This is shown in the lower right part of figure 3.15.
- Setting the threshold: This component of the UI is used for changing the maximum and minimum threshold value of hue, saturation and brightness. This was included in the UI because, in different light conditions, the threshold values need to be changed to detect the skin colour. This is shown on the top right part of the screen as can be seen from figure 3.15.

- Gesture Screen: This component of the UI displays the recognised gesture on the screen. This component is at the bottom of the screen as can be seen from figure 3.15.

It is important to note that all these components are only included because this was supposed to be a test version. The final version will only have a main screen showing the live video feed from the camera and a tab showing the gestures.

3.10.4 Transformation to an android application

The main challenge observed in conversion to an android application was to extract each frame from the camera of the phone and process it in real-time to detect the gesture. The software was programmed in android studio which is a special software developed by Google for generating android applications. A special function called *android.hardware.camera2* was used to develop a camera application for an Android phone which processes the image in the background, compares the data received from the current frame with the stored dataset and displays the gesture if the gesture is detected.

The gestures are displayed using a *toast* function in android. *Toast* is an android UI tool which is used to display simple feedbacks in android applications. In the project, it appears as a small pop up when a gesture is detected on the screen. *Toast* runs on an additional thread which is constructed every time a gesture is detected in the application.

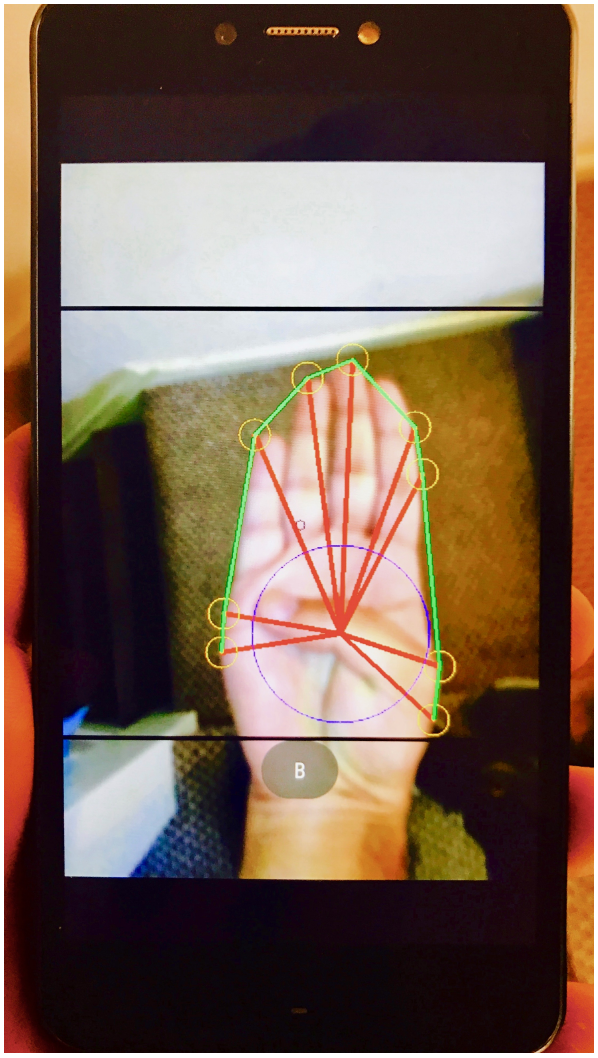


Figure 3.16: 'B' gesture in American sign language detected using the android application.

4 Test Results and Analysis

4.1 Results of the demo software created on computer

A test version version of the software was first developed on the computer using OpenCV. Because the computer version was a test version, the program was only tested for identification of basic signs like counting numbers. The software achieved high level of accuracy with simple hand number signs. Table 4.1 shows the recognition rate achieved with all the 5 gestures used for recognition purposes in this test.

Table 4.1: Experimental Results from the software developed on computer. Images obtained from http://lifeprint.com/asl101/topics/wallpaper1.htm [11]

Gesture	Test Samples	Correct	Success rate
	20	20	100%
	20	20	100%
	20	20	100%
	20	19	95%
	20	20	100%
ALL	100	99	99%

4.2 Results obtained from the android application

After achieving a high success rate with the computer version of the program, the focus of the project shifted towards the android application. Because this was going to be the final version of the software, the program was tested with all the 24 static gestures of the alphabets of American sign language. This test was conducted with a homogenous black background to make sure the results were not adulterated due to background noises. 20 test subjects were used to test all the different 24 gestures of ASL and the results were recorded. The success rate for each of the gestures has been shown in table 4.2. Table 3 also shows the gestures of the alphabets in American sign language.

From table 4.2, it can be observed that gesture for 'J' and 'Z' were not tested as these gestures are not static. Other than that, the success rate was very high with an overall recognition rate of 92%.

Table 4.2: Experimental Results from android application. Images obtained from http://lifeprint.com/asl101/topics/wallpaper1.htm [11]

Gesture	Test Samples	Success rate	Gesture	Test Samples	Success rate
	20	90%		20	85%
	20	100%		20	95%
	20	90%		20	85%
	20	100%		20	85%
	20	90%		20	100%
	20	100%		20	85%
	20	85%		20	80%
	20	85%		20	85%
	20	100%		20	100%
	N/A	N/A		20	100%
	20	100%		20	100%
	20	90%		20	90%
	20	90%		N/A	N/A

4.3 Analysis

On further analysis the results displayed in table 4.2, the following conclusions can be drawn.

- High Success Rates: Highest recognition rate was observed for gestures 'B', 'D', 'I', 'K', 'L', 'R', 'V', 'W' and 'Y'. A high success rate was achieved for these gestures as the hand shape and orientation for these gestures is different compared to the hand gestures of the alphabets of American Sign Language.
- Low Success Rates: Lowest recognition rate was observed for gestures 'G', 'H', 'N', 'P', 'Q', 'S', 'T' and 'U'. In case of most of these gestures, the reason for this poor performance was down to the similarities between hand shape and orientation of different gestures. In most of the cases of false positives, similar looking gestures such as 'G' and 'H', 'M' and 'N', and 'S' and 'T' were being confused by the application.
- False positives caused by simple gestures: Few gestures created false positives due to background noises. This was mostly caused by gestures that were not very complicated, noticed mostly in the case of gestures of 'L' and 'Y'.
- Non-static gestures: The program was not created for non-static gestures and hence results for gestures 'J' and 'Z' were not inspected during the testing of the android application.

4.4 Performance of other projects detecting American Sign Language

After deriving the results of different gestures from the android application, the results were compared with other pre-existing projects aimed at gesture recognition of alphabets in American sign language. Because, this kind of project has not been implemented as a mobile application before, the projects used here for comparison are other computer implemented gesture recognition softwares.

4.4.1 1st project: Real-Time Static Hand Gesture Recognition for American Sign Language (ASL) in Complex Background [3]

This project was conducted by Jayashree R. Pansare, Shravan H. Gawande and Maya Ingle and was published in Journal of Signal and Information Processing. The project uses image processing techniques to determine the American sign language gesture. It uses a webcam to extract the images which are then transferred to a computer for processing. The project used 100 samples for each sign and achieved an overall success rate of 90.19%. The recognition results of 26 hand gestures has been given in table 4.3.

Table 4.3: Experimental Results from the first project for comparison. Data obtained from Journal of Signal and Information Processing [3]

Sign	Samples	Success Rate	Sign	Samples	Success Rate
A	100	90%	N	100	95%
B	100	86%	O	100	86%
C	100	90%	P	100	90%
D	100	85%	Q	100	92%
E	100	100%	R	100	90%
F	100	85%	S	100	93%
G	100	90%	T	100	90%
H	100	92%	U	100	85%
I	100	85%	V	100	100%
J	100	86%	W	100	96%
K	100	90%	X	100	85%
L	100	92%	Y	100	92%
M	100	90%	Z	100	90%

4.4.2 2nd project: Appearance Based Recognition of American Sign Language Using Gesture Segmentation [12]

This project was conducted by Vaishali S Kulkarni and Dr. S.D.Lokhande and was published in the International Journal on Computer Science and Engineering. The project uses feature extraction methods and neural networks to recognise static gestures of alphabets in American Sign Language. The image, once processed, is converted into a feature vector which is then compared with feature vectors of the images in training set. The

accuracy of recognition of all the 26 alphabetic gestures has been shown in table 4.4.

Table 4.4: Experimental Results from the second project for comparison. Data obtained from International Journal on Computer Science and Engineering [12]

Sign	Samples	Success Rate	Sign	Samples	Success Rate
A	8	87.5%	N	8	87.5%
B	8	87.5%	O	8	87.5%
C	8	87.5%	P	8	100%
D	8	100%	Q	8	100%
E	8	100%	R	8	87.5%
F	8	100%	S	8	87.5%
G	8	87.5%	T	8	100%
H	8	87.5%	U	8	100%
I	8	100%	V	8	100%
J	8	100%	W	8	100%
K	8	87.5%	X	8	75%
L	8	87.5%	Y	8	87.5%
M	8	100%	Z	8	75%

The authors used 8 samples for each sign and were able to obtain an overall accuracy of 92.78% in the project. The signs of non-static gestures such as 'J' and 'Z' were changed to random static gestures for the sake of evaluating results.

4.4.3 Comparing results of the discussed projects with the application developed in the project

It can be concluded that the recognition rate achieved from the android application is comparable with any pre-developed American sign language recognition system. This project was developed as a mobile application while the other pre-developed projects discussed here used computer webcams for gesture recognition. It is important to note that mobile phones are more susceptible to background noise and lighting variations because of their portability over computers. Table 4.5 shows a comparison of the different recognition rates of the android application and the other projects discussed here.

Table 4.5: Comparison of overall recognition rate of different projects

Recognition rate of android application	92%
Recognition rate of 1st discussed project [3]	90.19%
Recognition rate of 2nd discussed project [12]	92.78%

5 Conclusion

The project provided a great outlook in the field of computer vision and image processing particularly in the areas of hand detection and gesture recognition. Different techniques in image processing were tried and tested which provided a greater understanding in different areas of image processing. Along with the powerful techniques, many limitations of image processing and computer vision such as background noises and light susceptibility were also observed. Learning about these limitations was important as different approaches were formularised such as obtaining largest contour, dilation and erosion to remove noises from the background.

During the start of the project, even thought the focus of the project was always clear, the initial objectives were set too high. The project was initially concerned with discerning all the gestures in British sign language. However, as the project started, it was realised that discerning all gestures and specially non-static gestures will be very hard to incorporate in the project in the given timeframe. Hence, due to the limited timeframe only the gestures of the alphabets were used for recognition purposes. The project also changed its focus from British sign language to American sign language. This was because the gestures the alphabets in British sign language use 2 hands while the alphabets in American sign language use only one hand. Hence it was easier to test with the gestures of American sign language as one hand, while testing, was always occupied in holding the phone.

OpenCV was also an integral part of the project as many functions of OpenCV were used for basic image processing operations. OpenCV was used for extracting the skin coloured pixels, obtaining the largest contour, dilating and eroding the binary image etc.

Although, the final version of the android application was tested with a black background, the application performs well even in complex backgrounds. The only limitation of the android application is to avoid skin coloured objects in the background. However, despite these limitations, the project is considered a success as it was successfully able to distinguish between different gestures and was able to recognise more than 20 gestures.

The project was a great starting point to understand the basics of image processing and computer vision and helped to gain a deeper understanding in how computer vision can be used to solve the problems of the present and the future.

6 Further Work

Whilst the project is complete on its own, there a lot of aspects such as robustness and accuracy where the application can be improved. The tests conducted in this project were performed in a controlled environment. To gain the best accuracy, the tests were conducted in front of a homogenous background that is not skin coloured. The following steps can be taken to make the application more robust:

- Use machine learning to predict next letters: The Android application can use machine learning techniques to predict the next letter and hence can make the system more robust. This can be helpful in the case when the gestures for two letters are very similar and the gesture recognition system can get confused between the two alphabets.
- Using other image processing techniques: The system, along with using the distances between the fingers to predict gestures, can also use other image processing techniques to certify if the gesture detected is correct or not. Image processing techniques such as skeletonising the fingers and eigenfaces can be used to validate the results obtained from the initial gesture recognition method.
- Using face recognition for eliminating face from the background: In many cases, the face of the subject hindered with the results of the gesture recognition system. This

is because the face of the subject is also skin coloured and in cases where the face becomes larger than the hand, the system will start sensing the face as the main contour and start ignoring the hand. This problem can be eliminated by having a face recognition system in the system that eliminates the face from the system.

Apart from increasing the robustness of the system, there are many additional features that could not be added in the system due to the time limitations of 3rd year project. Few additional features that can be added to the application have been discussed below:

- Including dynamic gestures: The time constraints in 3rd year only allowed the projects to implement static gesture recognition in American sign language. However, the method used in this project for detecting gesture recognition can be transformed to use dynamic gesture recognition as well. Every frame, once it detects the initial part of a dynamic gesture, will check if the next gesture is same as what is stored in the database. If all the series of distances between fingers is same as the series of frames of a gesture, then the gesture is identified.

- Incorporating other sign languages: There are more than 20 types of sign languages used around the world. This can cause problems in communicating people with disabilities if they know a different kind of sign language other than American sign language. The same method used here can be used to detect other types of sign languages as well. The application can provide an option selecting the type of sign language to be detected and the application will start using the database of that specific sign language for recognition.

- Uploading gestures on the app: The application can be connected with a database where people can upload their own gestures if it is not already included in the application. This newly added gesture will then be downloaded to other phones using this application and the gesture can be used by other people around the world using this application.

7 References

[1] V. M. Tiwari, *Android Application to understand British Sign Language using Gesture recognition: Third Year Individual Project-Progress Report*, School of Electrical and Electronic Engineering, University of Manchester, Manchester, UK, 2016.

[2] *American Sign Language fact sheet*, Colorado commission for dead and hard of hearing, Denver, Colorado.

[3] J. R. Pansare, S. H. Gawande, M. Ingle *Real-Time Static Hand Gesture Recognition for American Sign Language (ASL) in Complex Background*, Journal of Signal and Information Processing, 2012, 3, pp. 364-367.

[4] OpenCV *OpenCV*, 01 June 2000. [Online]. Available: http://www.opencv.org. [Accessed 01 November 2016].

[5] D. Khattab, H. M. Ebied, A. S. Hussein, M. F. Tolba *Color Image Segmentation Based on Different Color Space Models Using Automatic GrabCut*, Scientific World Journal, Hindawi Publishing Corporation, 2014, pp. 1-10.

[6] J. H. Bear *HSV*, 03 March 2017. [Online]. Available: https://www.thoughtco.com/what-is-hsv-in-design-1078068. [Accessed 02 April 2017].

[7] R. Fisher, S. Perkins, A. Walker and E. Wolfart, *Dilation*, 01 October 2000. [Online]. Available: http://homepages.inf.ed.ac.uk/rbf/HIPR2/dilate.htm. [Accessed 04 November 2016].

[8] R. Fisher, S. Perkins, A. Walker and E. Wolfart, *Dilation*, 01 October 2000. [Online]. Available: http://homepages.inf.ed.ac.uk/rbf/HIPR2/erode.htm. [Accessed 04 November 2016].

[9] B. Gartner, M. Hoffmann, *Computational Geometry Lecture Notes HS 2013*, ETH Zurich, 2014, pp. 25-37.

[10] J. Chen at al., *A fingertip detection method based on the combination of the centroid and Harris corner algorithm*, Software Engineering, Artificial Intelligence, Networking

and Parallel/Distributed Computing (SNPD), 2016 17th IEEE/ACIS International Conference, Shanghai, China, IEEE, 2016.

[11] David Rakowski, *American Sign Language Gallaudet Font*, Lifeprint 1991. [Online]. Available: http://lifeprint.com/asl101/topics/wallpaper1.htm. [Accessed 08 April 2017].

[12] V. S. Kulkarni, S. D. Lokhande, *Appearance Based Recognition of American Sign Language Using Gesture Segmentation*, International Journal on Computer Science and Engineering, 2010, 02, 03, pp.560-565.

Appendix A- Program Code

Matlab Code for initial hand segmentation

The following code was used to separate skin coloured pixels from the rest of the background.

```
clear all;
close all;

 filelocation  = '/Users/Vishisht/Desktop/yoyoyo.jpg';
filenames = dir( filelocation );

imageset_Original = imread(filelocation);
imageset_Original1 = imageset_Original;
imageset_HSV = rgb2hsv(imageset_Original1);
h=imageset_HSV(:,:,1);
s=imageset_HSV(:,:,2);

[r1, c1, v1]=find(h>0.1 | s<=0.15 | s>0.5); %non skin
numid1=size(r1,1);

for z=1:numid1
    imageset_Original(r1(z),c1(z),:)=0;
end

[r2, c2, v2]=find(~(h>0.9 | s<=0.15 | s>0.5)); %skin
numid2=size(r2,1);

for z=1:numid2
    imageset_Original(r2(z),c2(z),:)=255;
end

imshow(imageset_Original);
```

Java code of the trial version of hand gesture recognition on computer

The following java code was used to create a primitive version of the hand gesture recognition system. The code for the graphical user interface is incorporated in the gesture recognition code.

```
import java.awt.BorderLayout;
```

```
import java.awt.Dimension;
import java.awt.EventQueue;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.image.BufferedImage;

import javax.swing.BorderFactory;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JSlider;
import javax.swing.JSplitPane;

import org.opencv.core.*;
import org.opencv.core.CvType;
import org.opencv.core.Mat;
import org.opencv.core.MatOfPoint;
import org.opencv.core.Scalar;
import org.opencv.imgproc.*;
import org.opencv.core.Size;
import org.opencv.core.Point;

import java.util.ArrayList;
import java.util.List;

public class Recognizing_Fingers_Simple extends JFrame {
    Webcam webcam = new Webcam();

    ImageConverter cvt1 = new ImageConverter();
    ImageConverter cvt2 = new ImageConverter();
    ImageConverter cvt3 = new ImageConverter();
    ImageConverter cvt4 = new ImageConverter();
    ImageConverter cvt5 = new ImageConverter();

    Mat originalMat;

    BufferedImage originalImage;
    BufferedImage hsvImage;
    BufferedImage hsvImageApplied;
    BufferedImage linesImage;
    BufferedImage contourImage;

    private JPanel topPane;
    private JPanel secondTopPane;
    private JPanel textPane;
```

```
private JPanel contentPane;
private JPanel sidePane;
private JPanel sliderPane;
private JPanel smallPane;

private JSlider hueSliderMin;
private JSlider saturationSliderMin;
private JSlider valueSliderMin;
private JSlider hueSliderMax;
private JSlider saturationSliderMax;
private JSlider valueSliderMax;

JLabel gesture = new JLabel("Unkown");

int border = 1;

static final int FPS_MIN = 0;
static final int FPS_MAX = 255;
static final int FPS_INIT = 10;

private double[][] values = {{158.03, 147.58, 132.74},
        {91.68, 199.9, 103.31},
        {275.26, 303.49, 120.15},
        {265.6, 279.41, 279.29},
        {278.0, 283.04, 270.73, 258.44},
        {265.2, 274.12, 288.29, 286.23, 274.24},
        {250.4, 135.68, 132.83, 165.89},
        {241.85, 321.65, 157.85},
        {237.3, 285.82, 317.33, 144.84},
        {243.09, 274.14, 285.04, 292.86, 147.68},
        };

private double[][] ratioFromCentroid = {{100.00000, 97.12687, 86.61439,
    121.50944, 102.95812, 100.00000, 89.17655, 125.10383, 115.45426, 112.13710,
    100.00000, 140.28781, 82.29814, 79.93360, 71.28203, 100.00000},
        {100.00000, 92.29221, 78.04556, 105.51183, 108.35150, 100.00000, 84.56354,
            114.32366, 128.13029, 118.25428, 100.00000, 135.19261, 94.77610,
            87.47096, 73.96854, 100.00000},
        {100.00000, 99.23577, 115.11892, 100.77011, 100.00000, 116.00547,
            86.86669, 86.20283, 100.00000},
        {100.00000, 97.97592, 106.72310, 108.07551, 102.06590, 100.00000,
            108.92789, 110.30823, 93.70043, 91.80385, 100.00000, 101.26721,
            92.52790, 90.65506, 98.74865, 100.00000},
        {100.00000, 98.19263, 83.54441, 115.75966, 106.85232, 101.84064,
            100.00000, 85.08216, 117.89038, 108.81908, 119.69682, 117.53345,
            100.00000, 138.56063, 127.89882, 86.38588, 84.82457, 72.17057,
```

100.00000, 92.30531, 93.58712, 91.89565, 78.18680, 108.33613,
100.00000},
{100.00000, 54.44577, 106.66054, 183.66901, 100.00000, 195.90235,
93.75539, 51.04584, 100.00000},
{100.00000, 81.64063, 122.48803, 100.00000},
{100.00000, 53.95330, 107.79975, 185.34546, 100.00000, 199.80194,
92.76460, 50.04956, 100.00000},
{100.00000, 62.06874, 46.45068, 161.11169, 100.00000, 74.83747,
215.28211, 133.62290, 100.00000},
{100.00000, 61.96105, 44.43716, 161.39172, 100.00000, 71.71790,
225.03686, 139.43520, 100.00000},
{100.00000, 66.14439, 54.31321, 151.18440, 100.00000, 82.11309,
184.11728, 121.78326, 100.00000},
{100.00000, 72.71501, 55.04103, 137.52319, 100.00000, 75.69419,
181.68263, 132.11054, 100.00000},
{100.00000, 102.15016, 72.69978, 56.61859, 97.89510, 100.00000, 71.16952,
55.42682, 137.55200, 140.50959, 100.00000, 77.88000, 176.62045,
180.41807, 128.40267, 100.00000},
{100.00000, 97.13032, 69.19025, 102.95446, 100.00000, 71.23445,
144.52903, 140.38152, 100.00000},
{100.00000, 81.49701, 122.70388, 100.00000},
{100.00000, 97.42741, 97.65941, 102.64052, 100.00000, 100.23813,
102.39668, 99.76243, 100.00000},
{100.00000, 71.64574, 58.44599, 59.42751, 139.57565, 100.00000, 81.57636,
82.94634, 171.09815, 122.58453, 100.00000, 101.67938, 168.27223,
120.55988, 98.34836, 100.00000},
{100.00000, 87.55052, 87.74699, 114.21976, 100.00000, 100.22440,
113.96402, 99.77610, 100.00000},
{100.00000, 91.69165, 94.07383, 109.06118, 100.00000, 102.59803,
106.29949, 97.46776, 100.00000},
{100.00000, 73.83600, 135.43528, 100.00000},
{100.00000, 91.58468, 96.44940, 109.18856, 100.00000, 105.31171,
103.68131, 94.95620, 100.00000},
{100.00000, 76.12349, 65.60153, 131.36550, 100.00000, 86.17777,
152.43548, 116.03920, 100.00000},
{100.00000, 87.36679, 91.32631, 93.52067, 114.45997, 100.00000,
104.53206, 107.04373, 109.49747, 95.66443, 100.00000, 102.40277,
106.92823, 93.41977, 97.65361, 100.00000},
{100.00000, 113.82281, 87.57989, 70.35113, 87.85585, 100.00000, 76.94406,
61.80759, 114.18147, 129.96455, 100.00000, 80.32795, 142.14413,
161.79243, 124.48966, 100.00000},
{100.00000, 122.28842, 81.77389, 100.00000},
{100.00000, 81.98226, 73.95071, 75.39166, 77.02731, 121.97761, 100.00000,
90.20331, 91.96095, 93.95608, 135.22521, 110.86068, 100.00000,
101.94853, 104.16035, 132.64067, 108.74181, 98.08871, 100.00000,
102.16954, 129.82408, 106.43271, 96.00582, 97.87653, 100.00000},

{100.00000, 92.33390, 87.08120, 87.43798, 87.08837, 108.30258, 100.00000,
94.31120, 94.69759, 94.31895, 114.83534, 106.03195, 100.00000,
100.40970, 100.00823, 114.36678, 105.59931, 99.59197, 100.00000,
99.60016, 114.82590, 106.02323, 99.99177, 100.40144, 100.00000},
{100.00000, 100.21092, 70.20328, 65.13945, 68.53854, 79.39017, 100.46312,
99.78953, 100.00000, 70.05553, 65.00235, 68.39428, 79.22308,
100.25167, 142.44348, 142.74391, 100.00000, 92.78689, 97.62867,
113.08612, 143.10316, 153.51681, 153.84060, 107.77384, 100.00000,
105.21817, 121.87726, 154.22778, 145.90332, 146.21105, 102.42892,
95.04062, 100.00000, 115.83290, 146.57903, 125.96018, 126.22584,
88.42818, 82.04976, 86.33126, 100.00000, 126.54352, 99.53902,
99.74896, 69.87966, 64.83916, 68.22258, 79.02420, 100.00000},
{100.00000, 77.12223, 70.48651, 70.92132, 70.61688, 129.66430, 100.00000,
91.39583, 91.95964, 91.56489, 141.87113, 109.41418, 100.00000,
100.61688, 100.18497, 141.00131, 108.74336, 99.38690, 100.00000,
99.57074, 141.60919, 109.21217, 99.81537, 100.43111, 100.00000},
{100.00000, 101.48148, 94.76494, 94.37422, 98.54015, 100.00000, 93.38151,
92.99650, 105.52426, 107.08758, 100.00000, 99.58770, 105.96114,
107.53093, 100.41401, 100.00000},
{100.00000, 194.83634, 166.84993, 51.32513, 100.00000, 85.63594, 59.93410,
116.77340, 100.00000},
{100.00000, 194.73798, 51.35105, 100.00000},
{100.00000, 202.13185, 200.25519, 49.47266, 100.00000, 99.07157, 49.93628,
100.93714, 100.00000},
{100.00000, 196.38233, 199.45493, 138.81402, 50.92108, 100.00000,
101.56460, 70.68559, 50.13664, 98.45950, 100.00000, 69.59668,
72.03884, 141.47154, 143.68501, 100.00000},
{100.00000, 82.68628, 143.37554, 120.93905, 100.00000, 173.39701,
69.74691, 57.67112, 100.00000},
{100.00000, 72.60551, 131.84318, 137.73060, 100.00000, 181.58840,
75.84769, 55.06960, 100.00000},
{100.00000, 68.95921, 119.14967, 145.01325, 100.00000, 172.78281,
83.92806, 57.87613, 100.00000},
{100.00000, 92.57205, 56.05192, 108.02396, 100.00000, 60.54951,
178.40601, 165.15411, 100.00000},
{100.00000, 65.67570, 114.79384, 152.26331, 100.00000, 174.78890,
87.11269, 57.21187, 100.00000},
{100.00000, 73.95012, 123.58132, 135.22629, 100.00000, 167.11443,
80.91838, 59.83924, 100.00000},
{100.00000, 75.67898, 68.74207, 113.24152, 132.13709, 100.00000, 90.83377,
149.63405, 145.47133, 110.09122, 100.00000, 164.73395, 88.30683,
66.82971, 60.70394, 100.00000},
{100.00000, 106.65926, 96.71418, 61.87120, 54.97764, 93.75651, 100.00000,
90.67584, 58.00827, 51.54511, 103.39745, 110.28296, 100.00000,
63.97324, 56.84547, 161.62610, 172.38921, 156.31537, 100.00000,
88.85821, 181.89214, 194.00482, 175.91550, 112.53884, 100.00000},

```
            {100.00000, 71.20141, 64.68752, 111.37901, 140.44666, 100.00000, 90.85146,
                156.42810, 154.58932, 110.06978, 100.00000, 172.18005, 89.78352,
                63.92714, 58.07874, 100.00000},
            {100.00000, 66.92880, 60.35098, 149.41250, 100.00000, 90.17192,
                165.69738, 110.89927, 100.00000},
            {100.00000, 64.55422, 57.06684, 154.90854, 100.00000, 88.40141,
                175.23311, 113.12037, 100.00000},
            {100.00000, 81.49286, 77.73744, 75.60614, 122.71014, 100.00000, 95.39172,
                92.77641, 128.63815, 104.83090, 100.00000, 97.25834, 132.26438,
                107.78602, 102.81894, 100.00000},
            {100.00000, 100.40098, 67.95491, 62.55069, 62.13572, 99.60062, 100.00000,
                67.68351, 62.30087, 61.88756, 147.15641, 147.74648, 100.00000,
                92.04735, 91.43669, 159.87034, 160.51139, 108.63974, 100.00000,
                99.33659, 160.93803, 161.58336, 109.36529, 100.66784, 100.00000},
            {100.00000, 110.25780, 79.45951, 71.89604, 90.69653, 100.00000, 72.06702,
                65.20721, 125.85026, 138.75973, 100.00000, 90.48135, 139.08972,
                153.35727, 110.52001, 100.00000},
            {100.00000, 92.23409, 87.51596, 84.64415, 108.41979, 100.00000, 94.88462,
                91.77100, 114.26487, 105.39116, 100.00000, 96.71853, 118.14166,
                108.96688, 103.39281, 100.00000},
            {100.00000, 96.92710, 67.59297, 63.04651, 62.83039, 103.17032, 100.00000,
                69.73588, 65.04528, 64.82232, 147.94438, 143.39821, 100.00000,
                93.27376, 92.95404, 158.61307, 153.73906, 107.21129, 100.00000,
                99.65722, 159.15864, 154.26786, 107.58005, 100.34396, 100.00000},
            };

    /*
     * for finger detection
     */
    private static final int MAX_POINTS = 20;
    private int contourAxisAngle;
    private Point cogPt = new Point(); // center of gravity (COG) of contour
    private ArrayList<Point> fingerTips = new ArrayList<Point>();
    private Point[] tipPts, endPts, foldPts;
    private float[] depths;
    private double[] valueFingers;
    private double[] ratioFingers;
    private int division = 5;
    private static final int MAX_FINGER_DISTANCE = 70;
    /*
     *
     */

    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
```

```
                try {
                    Recognizing_Fingers_Simple frame = new
                        Recognizing_Fingers_Simple();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the frame.
     */
    public Recognizing_Fingers_Simple() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        setBounds(100, 100, 1450, 820);
        topPane = new JPanel();
        secondTopPane = new JPanel();
        textPane = new JPanel();
        contentPane = new JPanel();
        sidePane = new JPanel();
        sliderPane = new JPanel();
        smallPane = new JPanel();

        JSplitPane topSplitPaneH = new JSplitPane(JSplitPane.VERTICAL_SPLIT);
        JSplitPane splitPaneV = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT);
        JSplitPane splitPaneH = new JSplitPane(JSplitPane.VERTICAL_SPLIT);

        topPane.add(topSplitPaneH, BorderLayout.CENTER);
        topSplitPaneH.setLeftComponent(secondTopPane);
        topSplitPaneH.setRightComponent(textPane);

        secondTopPane.add(splitPaneV, BorderLayout.CENTER);
        splitPaneV.setLeftComponent(contentPane);
        splitPaneV.setRightComponent(sidePane);

        sidePane.add(splitPaneH, BorderLayout.CENTER);
        splitPaneH.setLeftComponent(sliderPane);
        splitPaneH.setRightComponent(smallPane);

        secondTopPane.setPreferredSize(new Dimension(1450, 720));
        textPane.setPreferredSize(new Dimension(1450, 100));
        contentPane.setPreferredSize(new Dimension(1200, 720));
        sidePane.setPreferredSize(new Dimension(250, 720));
```

```
sliderPane.setPreferredSize(new Dimension(250, 480));
smallPane.setPreferredSize(new Dimension(250, 240));

hueSliderMin = new JSlider(JSlider.HORIZONTAL, FPS_MIN, FPS_MAX,
    FPS_INIT);
saturationSliderMin = new JSlider(JSlider.HORIZONTAL, FPS_MIN,
    FPS_MAX, FPS_INIT);
valueSliderMin = new JSlider(JSlider.HORIZONTAL, FPS_MIN, FPS_MAX,
    FPS_INIT);
hueSliderMax = new JSlider(JSlider.HORIZONTAL, FPS_MIN, FPS_MAX,
    FPS_INIT);
saturationSliderMax = new JSlider(JSlider.HORIZONTAL, FPS_MIN,
    FPS_MAX, FPS_INIT);
valueSliderMax = new JSlider(JSlider.HORIZONTAL, FPS_MIN, FPS_MAX,
    FPS_INIT);

gesture.setFont(gesture.getFont().deriveFont(48.0f));

hueSliderMin.setMajorTickSpacing(50);
hueSliderMin.setMinorTickSpacing(10);
hueSliderMin.setPaintTicks(true);
hueSliderMin.setPaintLabels(true);
hueSliderMax.setMajorTickSpacing(50);
hueSliderMax.setMinorTickSpacing(10);
hueSliderMax.setPaintTicks(true);
hueSliderMax.setPaintLabels(true);
saturationSliderMin.setMajorTickSpacing(50);
saturationSliderMin.setMinorTickSpacing(10);
saturationSliderMin.setPaintTicks(true);
saturationSliderMin.setPaintLabels(true);
saturationSliderMax.setMajorTickSpacing(50);
saturationSliderMax.setMinorTickSpacing(10);
saturationSliderMax.setPaintTicks(true);
saturationSliderMax.setPaintLabels(true);
valueSliderMin.setMajorTickSpacing(50);
valueSliderMin.setMinorTickSpacing(10);
valueSliderMin.setPaintTicks(true);
valueSliderMin.setPaintLabels(true);
valueSliderMax.setMajorTickSpacing(50);
valueSliderMax.setMinorTickSpacing(10);
valueSliderMax.setPaintTicks(true);
valueSliderMax.setPaintLabels(true);

hueSliderMin.setValue(112);
hueSliderMax.setValue(158);
saturationSliderMin.setValue(10);
```

```
        saturationSliderMax.setValue(142);
        valueSliderMin.setValue(100);
        valueSliderMax.setValue(232);

        hueSliderMin.setBorder(BorderFactory.createTitledBorder("HueMin"));
        hueSliderMax.setBorder(BorderFactory.createTitledBorder("HueMax"));
        saturationSliderMin.setBorder(BorderFactory.createTitledBorder("
            SaturationMin"));
        saturationSliderMax.setBorder(BorderFactory.createTitledBorder("
            SaturationMax"));
        valueSliderMin.setBorder(BorderFactory.createTitledBorder("ValueMin"));
        valueSliderMax.setBorder(BorderFactory.createTitledBorder("ValueMax"));

        textPane.add(gesture);

        sliderPane.add(hueSliderMin);
        sliderPane.add(hueSliderMax);
        sliderPane.add(saturationSliderMin);
        sliderPane.add(saturationSliderMax);
        sliderPane.add(valueSliderMin);
        sliderPane.add(valueSliderMax);

        setContentPane(topPane);

        new RunnerThread().start();
    }

    @Override
    public void paint(Graphics g) {
        super.paint(g);
        g = contentPane.getGraphics();

        if (originalImage != null) {
            setSize(originalImage.getWidth() + border, originalImage.getHeight() +
                border);
        }
    }

    public class RunnerThread extends Thread {
        public void run() {
            while (contentPane == null || contentPane.getGraphics() == null) {
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException ex) {
                }
            }
```

```
        Graphics g = contentPane.getGraphics();
        Graphics2D g2 = (Graphics2D) g;
        Graphics gSmall = smallPane.getGraphics();
        Graphics2D g2Small = (Graphics2D) gSmall;
        while (true) {
            updateFrame(g2, g2Small);
        }
    }

    private int angle(Point tip)
    // calculate the angle between the tip and its neighbouring folds
    // (in integer degrees)
    {
        return Math.abs((int) Math.round(Math.toDegrees(Math.atan2(tip.x, tip.
            y - ((2*originalMat.height())/3)))));
    }

    public void updateFrame(Graphics2D g, Graphics2D gSmall) {
        originalMat = webcam.getNewImage(true);
        Mat hsvMat = new Mat();
        Mat hsvMatApplied = new Mat();

        //Imgproc.blur(originalMat, originalMat, new Size(7, 7));
        Imgproc.cvtColor(originalMat, hsvMat, Imgproc.COLOR_BGR2HSV);

        Scalar hsvMin = new Scalar(hueSliderMin.getValue(), saturationSliderMin
            .getValue(),
                valueSliderMin.getValue());
        Scalar hsvMax = new Scalar(hueSliderMax.getValue(),
            saturationSliderMax.getValue(),
                valueSliderMax.getValue());

        double minHue, minSaturation, minValue, maxHue, maxSaturation,
            maxValue;

        if (hsvMat.get(originalMat.height()/2, originalMat.width()/2)[0] > 15) {
            minHue = hsvMat.get(originalMat.height()/2, originalMat.width()/2)
                [0] - 15;
        }
        else {
            minHue = hsvMat.get(originalMat.height()/2, originalMat.width()/2)
                [0];
        }
        if (hsvMat.get(originalMat.height()/2, originalMat.width()/2)[1] > 25) {
            minSaturation = hsvMat.get(originalMat.height()/2, originalMat.width
                ()/2)[1] - 25;
```

```
}
else {
    minSaturation = hsvMat.get(originalMat.height()/2, originalMat.width
        ()/2)[1];
}
if (hsvMat.get(originalMat.height()/2, originalMat.width()/2)[2] > 60) {
    minValue = hsvMat.get(originalMat.height()/2, originalMat.width()/2)
        [2] - 60;
}
else {
    minValue = hsvMat.get(originalMat.height()/2, originalMat.width()/2)
        [2];
}
if (hsvMat.get(originalMat.height()/2, originalMat.width()/2)[0] < 240) {
    maxHue = hsvMat.get(originalMat.height()/2, originalMat.width()/2)
        [0] + 15;
}
else {
    maxHue = hsvMat.get(originalMat.height()/2, originalMat.width()/2)
        [0];
}
if (hsvMat.get(originalMat.height()/2, originalMat.width()/2)[1] < 230) {
    maxSaturation = hsvMat.get(originalMat.height()/2, originalMat.width
        ()/2)[1] + 25;
}
else {
    maxSaturation = hsvMat.get(originalMat.height()/2, originalMat.width
        ()/2)[1];
}
if (hsvMat.get(originalMat.height()/2, originalMat.width()/2)[2] < 195) {
    maxValue = hsvMat.get(originalMat.height()/2, originalMat.width()/2)
        [2] + 60;
}
else {
    maxValue = hsvMat.get(originalMat.height()/2, originalMat.width()/2)
        [2];
}

//Scalar hsvMin = new Scalar(minHue, minSaturation, minValue);
//Scalar hsvMax = new Scalar(maxHue, maxSaturation, maxValue);

Core.inRange(hsvMat, hsvMin, hsvMax, hsvMatApplied);
//Imgproc.blur(hsvMatApplied, hsvMatApplied, new Size(1, 1));
Mat dilateElement = Imgproc.getStructuringElement(Imgproc.
    MORPH_RECT, new Size(1, 1));
Mat erodeElement = Imgproc.getStructuringElement(Imgproc.
```

```
    MORPH_RECT, new Size(1, 1));
//Imgproc.erode(hsvMatApplied, hsvMatApplied, erodeElement);
//Imgproc.dilate(hsvMatApplied, hsvMatApplied, dilateElement);
//Imgproc.blur(hsvMatApplied, hsvMatApplied, new Size(1, 1));

Mat contourMat = new Mat(hsvMatApplied.rows(), hsvMatApplied.cols(),
    CvType.CV_8UC1);

// MatOfPoint contourMat = new MatOfPoint();
List<MatOfPoint> contours = new ArrayList<MatOfPoint>();
Mat hierarchy = new Mat();
Imgproc.findContours(hsvMatApplied, contours, hierarchy, Imgproc.
    RETR_LIST, Imgproc.CHAIN_APPROX_SIMPLE);

double maxVal = 0;
int maxValIdx = 0;

for (int contourIdx = 0; contourIdx < contours.size(); contourIdx++) {
    double contourArea = Imgproc.contourArea(contours.get(contourIdx))
        ;
    if (maxVal < contourArea) {
        maxVal = contourArea;
        maxValIdx = contourIdx;
    }
}

Imgproc.drawContours(contourMat, contours, maxValIdx, new Scalar(255,
    255, 255), -1);
// System.out.println(contourMat);

if (maxValIdx > 0) {

    extractContourInfo(contours.get(maxValIdx), 1);
    // find the COG and angle to horizontal of the contour

    findFingerTips(contours.get(maxValIdx), 1);
    // detect the fingertips positions in the contour

    valueFingers = new double[fingerTips.size()];

    Mat lines = originalMat;
    if (contourMat.height() > 0 && contourMat.width() > 0 &&
        fingerTips.size() != 0) {
        for (int i = 0; i < fingerTips.size(); i++) {

            Point center = new Point(originalMat.width()/2, originalMat.
```

```
            height()/2);
        Point topLeft = new Point(0, originalMat.height()/division);
        Point topRight = new Point(originalMat.width(), originalMat.
            height()/division);
        Point bottomLeft = new Point(0, (division - 1)*originalMat.
            height()/division);
        Point bottomRight = new Point(originalMat.width(), (division
             - 1)*originalMat.height()/division);
        //Imgproc.circle(lines, center, 5, new Scalar(0, 0, 0));
        Imgproc.line(lines, topLeft, topRight, new Scalar(0, 0, 0), 4)
            ;
        Imgproc.line(lines, bottomLeft, bottomRight, new Scalar(0,
            0, 0), 4);


        Point pt1 = fingerTips.get(i);
        Imgproc.line(lines, cogPt, pt1, new Scalar(255, 0, 0), 4);
        Imgproc.circle(lines, pt1, 16, new Scalar(255, 204, 0));
        if (i > 0) {
            Point pt2 = fingerTips.get(i - 1);
            Imgproc.line(lines, pt2, pt1, new Scalar(0, 255, 0), 2);
        }
        valueFingers[i] = (Math.sqrt(Math.pow((pt1.x - cogPt.x), 2)
            + Math.pow((pt1.y - cogPt.y), 2)));
        //valueFingers[i] *= 100;
        //valueFingers[i] = (double) Math.round(ratioFingers[i]);
        //valueFingers[i] /= 100;
    }

    ratioFingers = new double[valueFingers.length * valueFingers.
        length];
    for (int i = 0; i < valueFingers.length; i++) {
        for (int j = 0; j < valueFingers.length; j++) {
            ratioFingers[(i * valueFingers.length) + j] = 100 * (
                valueFingers[i] / valueFingers[j]);
        }
    }
    for (int i = 0; i < ratioFingers.length; i++) {
        //System.out.print(ratioFingers[i] + " ");
    }
    //System.out.println();
    Imgproc.circle(lines, cogPt, 90, new Scalar(0, 0, 255));

    originalImage = cvt1.mat2image(originalMat);
    hsvImage = cvt2.mat2image(hsvMat);
    hsvImageApplied = cvt3.mat2image(hsvMatApplied);
```

```
                    contourImage = cvt4.mat2image(contourMat);

                    linesImage = cvt5.mat2image(lines);

                    g.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                        RenderingHints.VALUE_ANTIALIAS_ON); // linesmoothing
                    g.drawImage(linesImage, 0, 0, null);
                    gSmall.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                        RenderingHints.VALUE_ANTIALIAS_ON); // linesmoothing
                    gSmall.drawImage(contourImage, 0, 0, 250, 240, null);

                    boolean same = true;

                    for (int i = 0; i < 50; i++) {
                        if (ratioFromCentroid[i].length == ratioFingers.length) {
                            System.out.println(ratioFromCentroid[i].length);
                            for (int j = 0; j < ratioFingers.length; j++) {
                                if (ratioFingers[j] <= (ratioFromCentroid[i][j] + 20)
                                    && ratioFingers[j] >= (ratioFromCentroid[i][j] -
                                    20)) {
                                    same &= true;
                                } else {
                                    same &= false;
                                }
                                //System.out.println(ratioFingers[j]);
                            }
                            if (same == true) {
                                gesture.setText("B");
                                break;
                                //System.out.println(i);
                            }
                            else {
                                gesture.setText("B");
                            }
                        }
                    }
                }
            }
            try {
                Thread.sleep(30);
            } catch (InterruptedException ex) {
            }
        }
    }
}
```

Java Code for Android Application

Following code was used to develop the android application for gesture recognition of the alphabets of American Sign Language.

```
package com.example.vishisht.opencvcamera;

import android.content.Context;
import android.content.pm.ActivityInfo;
import android.hardware.camera2.*;
import android.os.Build;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.SurfaceView;
import android.view.View;
import android.widget.Toast;

import org.opencv.android.BaseLoaderCallback;
import org.opencv.android.CameraBridgeViewBase;
import org.opencv.android.JavaCameraView;
import org.opencv.android.LoaderCallbackInterface;
import org.opencv.android.OpenCVLoader;
import org.opencv.core.Core;
import org.opencv.core.CvType;
import org.opencv.core.Mat;
import org.opencv.core.MatOfInt;
import org.opencv.core.MatOfInt4;
import org.opencv.core.MatOfPoint;
import org.opencv.core.Point;
import org.opencv.core.Scalar;
import org.opencv.core.Size;
import org.opencv.imgproc.Imgproc;
import org.opencv.imgproc.Moments;


import java.util.ArrayList;
import java.util.List;

import static java.lang.Math.abs;

public class MainActivity extends AppCompatActivity implements
    CameraBridgeViewBase.CvCameraViewListener2 {

    private static String TAG = "MainActivity";
```

```
JavaCameraView javaCameraView;

private CameraCaptureSession mSession;
private CaptureRequest.Builder mBuilder;
private CameraDevice mCameraDevice;
private CameraManager mCameraManager;

static private double[][] values =
        {{165, 286, 198, 184, 146},//A
                {165, 287, 200, 186, 155},
                {165, 286, 200, 186, 156},
                {163, 288, 200, 187, 163},
                {173, 288, 201, 188},
                {174, 290, 200, 186},
                {174, 289, 200, 188, 155},
                {172, 290, 196, 186},
                {163, 285, 195, 186, 170, 153},
                {160, 278, 190, 180, 147},
                {159, 271, 186, 188, 176, 138},
                {157, 272, 189, 175, 135},
                {157, 272, 188, 189, 175, 150},
                {155, 271, 187, 189, 174, 147},
                {156, 274, 190, 192, 177, 152},
                {158, 274, 190, 191, 178, 155},
                {155, 271, 187, 175, 151},
                {149, 255, 174, 166, 144},
                {151, 255, 176, 151, 137},
                {155, 260, 180, 180, 136},
                {154, 259, 182, 179},
                {155, 256, 189, 184, 178, 147},
                {158, 268, 196, 190, 145},
                {167, 278, 205, 199, 199, 153},
                {167, 283, 209, 202, 156},
                {171, 288, 209, 203, 160},
                {174, 293, 208, 203, 156},
                {171, 292, 199, 199, 160},
                {170, 294, 199, 199, 194, 156},
                {176, 299, 203, 205, 199, 163},
                {176, 299, 204, 205, 200, 161},
                {178, 307, 205, 207, 202, 170},
                {180, 311, 209, 211, 167},
                {176, 303, 205, 206, 200, 160},
                {170, 290, 200, 198, 192, 153},
                {167, 284, 196, 195, 188, 148},
                {159, 272, 190, 148},
                {155, 262, 184, 145},
```

```
{147, 245, 171, 136},
{138, 238, 161, 121},
{136, 234, 158, 119},
{137, 233, 155, 157, 151, 121},
{140, 238, 165, 157, 124},
{145, 248, 172, 165, 130},
{151, 260, 180, 173, 143},
{162, 274, 190, 183, 148},
{165, 280, 193, 151},
{154, 165, 264, 184, 181, 152},
{157, 267, 183, 178, 146},
{162, 276, 188, 183, 150},//A

{85, 205, 254, 310, 296, 247, 180, 138, 134, 126},//B
{84, 225, 269, 330, 320, 268, 197, 146, 120},
{86, 225, 264, 322, 308, 262, 238, 134},
{89, 218, 260, 318, 302, 257, 132},
{82, 193, 236, 292, 277, 234, 211, 118},
{99, 172, 214, 264, 247, 201, 104},
{98, 175, 212, 262, 245, 199, 172, 104},
{99, 175, 213, 264, 248, 200, 104},
{99, 173, 211, 262, 250, 200, 103, 104, 111},
{99, 173, 214, 263, 245, 199, 174, 104},
{98, 170, 212, 262, 250, 200, 177, 103, 104},
{97, 169, 211, 262, 244, 198, 102, 104},
{106, 170, 204, 257, 240, 193, 102},
{102, 168, 208, 258, 242, 195, 173, 104},
{106, 167, 209, 257, 240, 195, 170, 104, 112},
{102, 164, 205, 257, 241, 194, 99},
{104, 171, 207, 256, 240, 193, 99, 103},
{109, 170, 207, 232, 257, 244, 194, 99, 103, 112},
{102, 169, 208, 257, 240, 195, 102},
{98, 166, 204, 253, 236, 192, 167, 97, 98, 106},
{97, 160, 205, 250, 238, 188, 96, 96},
{93, 168, 204, 251, 236, 191, 96},
{88, 157, 195, 242, 227, 184, 91},
{83, 154, 190, 235, 220, 178, 159, 86},
{76, 154, 182, 224, 211, 173, 86, 82},
{75, 153, 224, 215, 175, 156, 86, 81, 78},
{74, 158, 191, 232, 219, 180, 88},
{79, 160, 192, 238, 224, 184, 91, 87},
{84, 161, 197, 242, 227, 186, 163, 93},
{86, 167, 198, 248, 234, 189, 169, 96, 95},
{87, 158, 199, 250, 234, 192, 96},
{89, 167, 203, 253, 237, 195, 98, 97},
{90, 165, 203, 253, 238, 195, 99, 98},
```

```
                {92, 166, 203, 258, 240, 194, 174, 99, 99},
                {93, 169, 208, 261, 245, 198, 101, 101},
                {93, 175, 211, 265, 249, 203, 102},
                {93, 172, 209, 265, 254, 202, 183, 103, 103},
                {95, 168, 210, 237, 268, 256, 205, 104},
                {113, 175, 216, 277, 263, 205, 183, 115},
                {114, 179, 220, 246, 278, 260, 206, 187, 112, 114},
                {119, 173, 216, 276, 262, 204, 178, 119},
                {123, 178, 219, 247, 277, 260, 204, 179, 113, 120},
                {125, 179, 221, 281, 261, 207, 184, 115, 119},
                {134, 182, 226, 254, 288, 269, 212, 186, 120, 131},
                {136, 191, 228, 263, 294, 279, 215, 124, 134},
                {147, 189, 227, 260, 291, 271, 214, 190, 124, 133},
                {164, 193, 225, 291, 269, 211, 124, 136},
                {159, 186, 226, 257, 288, 273, 209, 181, 123, 136},
                {156, 139, 185, 222, 254, 285, 270, 206, 179, 120, 1, 132},
                {152, 182, 217, 247, 280, 262, 205, 181, 118, 127},//B

static private double[][] AValues = {{151, 244, 53, 46},
        {188, 204, 75, 55, 68},
        {163, 198, 44, 50, 39},
        {167, 252, 49, 65},
        {163, 172, 53, 113},
};

static private double[][] BValues = {{139, 78, 72, 45, 65, 164},
        {24, 119, 30, 44, 68, 50, 60, 183},
        {148, 21, 49, 70, 50, 59, 157, 21},
        {144, 70, 72, 50, 56, 187},
        {148, 73, 70, 51, 57, 156, 25},
};

static private double[][] CValues = {{404, 58, 36, 26, 85, 27, 131},
        {84, 265, 31, 38, 96, 134},
        {285, 41, 36, 110, 25, 135, 206},
        {107, 278, 35, 34, 105, 26, 126, 219},
        {82, 265, 61, 93, 133},
};

static private double[][] DValues = {{238, 261, 53, 41},
        {74, 144, 39, 219, 45, 36},
        {249, 269, 51},
        {254, 282, 51, 57},
        {195, 226, 43, 29},
};
```

```
static private double[][] EValues = {{96, 35, 105, 75, 107, 34},
        {92, 122, 56, 98, 24},
        {86, 120, 58, 90},
        {74, 25, 28, 92, 53, 100, 23},
        {168, 40, 57, 121, 118, 115, 50},
};

static private double[][] YValues = {{332},
        {338},
        {319},
        {308, 36},
};

static private String[] alphabetValues = new String[values.length];
private double[] valueFingers, valueBetweenFingers;
private String gesture = "Hello", previousGesture = "Hello";

/*
 * for finger detection
 */
Mat mRgba, hsvMat, hsvMatApplied, contourMat, lines;
private static final int MAX_POINTS = 40;
private int contourAxisAngle;
private Point cogPt = new Point(); // center of gravity (COG) of contour
private static final int MAX_FINGER_DISTANCE = 20;
/*
 *
 */


BaseLoaderCallback mLoaderCallBack = new BaseLoaderCallback(this) {
    @Override
    public void onManagerConnected(int status) {
        switch (status) {
            case BaseLoaderCallback.SUCCESS: {
                javaCameraView.enableView();
                break;
            }
            default: {
                super.onManagerConnected(status);
                break;
            }
        }
    }
};
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);

    try {
        mBuilder.set(CaptureRequest.FLASH_MODE, CameraMetadata.
            FLASH_MODE_TORCH);
        mSession.setRepeatingRequest(mBuilder.build(), null, null);
    } catch (Exception e) {
        e.printStackTrace();
    }

    javaCameraView = (JavaCameraView)findViewById(R.id.java_camera_view);
    javaCameraView.setVisibility(SurfaceView.VISIBLE);
    javaCameraView.setCvCameraViewListener(this);
}

private int angle(Point tip)
// calculate the angle between the tip and its neighbouring folds
// (in integer degrees)
{
    return abs((int) Math.round(Math.toDegrees(Math.atan2(tip.x, tip.y - ((2*
        mRgba.height())/3)))));
}

@Override
public void onCameraViewStarted(int width, int height) {
    mRgba = new Mat(height, width, CvType.CV_8UC4);
    hsvMat = new Mat(height, width, CvType.CV_8UC4);
    hsvMatApplied = new Mat(height, width, CvType.CV_8UC4);
    contourMat = new Mat(hsvMatApplied.rows(), hsvMatApplied.cols(), CvType
        .CV_8UC1);
}

@Override
public void onCameraViewStopped() {
    mRgba.release();
    contourMat.release();
}

@Override
public Mat onCameraFrame(CameraBridgeViewBase.CvCameraViewFrame
    inputFrame) {
```

```
mRgba = inputFrame.rgba();

Mat mRgbaT = mRgba.t();
Core.flip(mRgba.t(), mRgbaT, 1);
Imgproc.resize(mRgbaT, mRgbaT, mRgba.size());

Imgproc.blur(mRgba, mRgba, new Size(7, 7));
Imgproc.cvtColor(mRgba, hsvMat, Imgproc.COLOR_BGR2HSV);

Scalar hsvMin = new Scalar(95, 10, 100);
Scalar hsvMax = new Scalar(133, 142, 232);

System.out.println(hsvMat.get(mRgba.height()/2, mRgba.width()/2));

double minHue, minSaturation, minValue, maxHue, maxSaturation, maxValue
    , tempHue = 30, tempSaturation = 80, tempValue = 100;

if (hsvMat.get(mRgba.height()/2, mRgba.width()/2)[0] > tempHue) {
    minHue = hsvMat.get(mRgba.height()/2, mRgba.width()/2)[0] - tempHue
        ;
}
else {
    minHue = hsvMat.get(mRgba.height()/2, mRgba.width()/2)[0];
}
if (hsvMat.get(mRgba.height()/2, mRgba.width()/2)[1] > tempSaturation) {
    minSaturation = hsvMat.get(mRgba.height()/2, mRgba.width()/2)[1] -
        tempSaturation;
}
else {
    minSaturation = hsvMat.get(mRgba.height()/2, mRgba.width()/2)[1];
}
if (hsvMat.get(mRgba.height()/2, mRgba.width()/2)[2] > tempValue) {
    minValue = hsvMat.get(mRgba.height()/2, mRgba.width()/2)[2] -
        tempValue;
}
else {
    minValue = hsvMat.get(mRgba.height()/2, mRgba.width()/2)[2];
}
if (hsvMat.get(mRgba.height()/2, mRgba.width()/2)[0] < (255 - tempHue)) {
    maxHue = hsvMat.get(mRgba.height()/2, mRgba.width()/2)[0] +
        tempHue;
}
else {
    maxHue = hsvMat.get(mRgba.height()/2, mRgba.width()/2)[0];
}
if (hsvMat.get(mRgba.height()/2, mRgba.width()/2)[1] < (255 -
```

```
        tempSaturation)) {
        maxSaturation = hsvMat.get(mRgba.height()/2, mRgba.width()/2)[1] +
            tempSaturation;
    }
    else {
        maxSaturation = hsvMat.get(mRgba.height()/2, mRgba.width()/2)[1];
    }
    if (hsvMat.get(mRgba.height()/2, mRgba.width()/2)[2] < (255 - tempValue))
        {
        maxValue = hsvMat.get(mRgba.height()/2, mRgba.width()/2)[2] +
            tempValue;
    }
    else {
        maxValue = hsvMat.get(mRgba.height()/2, mRgba.width()/2)[2];
    }

    //Scalar hsvMin = new Scalar(minHue, minSaturation, minValue);
    //Scalar hsvMax = new Scalar(maxHue, maxSaturation, maxValue);

    Core.inRange(hsvMat, hsvMin, hsvMax, hsvMatApplied);
    Imgproc.blur(hsvMatApplied, hsvMatApplied, new Size(1, 1));
    Mat dilateElement = Imgproc.getStructuringElement(Imgproc.MORPH_RECT,
        new Size(1, 1));
    Mat erodeElement = Imgproc.getStructuringElement(Imgproc.MORPH_RECT,
        new Size(1, 1));
    Imgproc.erode(hsvMatApplied, hsvMatApplied, erodeElement);
    Imgproc.dilate(hsvMatApplied, hsvMatApplied, dilateElement);
    Imgproc.blur(hsvMatApplied, hsvMatApplied, new Size(1, 1));

    contourMat = new Mat(hsvMatApplied.rows(), hsvMatApplied.cols(), CvType
        .CV_8UC1);

    // MatOfPoint contourMat = new MatOfPoint();
    List<MatOfPoint> contours = new ArrayList<MatOfPoint>();
    Mat hierarchy = new Mat();
    Imgproc.findContours(hsvMatApplied, contours, hierarchy, Imgproc.
        RETR_LIST, Imgproc.CHAIN_APPROX_SIMPLE);

    double maxVal = 0;
    int maxValIdx = 0;

    for (int contourIdx = 0; contourIdx < contours.size(); contourIdx++) {
        double contourArea = Imgproc.contourArea(contours.get(contourIdx));
        if (maxVal < contourArea) {
            maxVal = contourArea;
            maxValIdx = contourIdx;
```

```
        }
    }

    Imgproc.drawContours(contourMat, contours, maxValIdx, new Scalar(255, 255,
        255), -1);
    // System.out.println(contourMat);

    if (maxValIdx > 0) {

        MatOfPoint bigContour = contours.get(maxValIdx);

        extractContourInfo(bigContour, 1);
        // find the COG and angle to horizontal of the contour

        findFingerTips(bigContour, 1);
        // detect the fingertips positions in the contour

        for (int i = 0; i < alphabetValues.length; i++) {
            if (i < 50) {
                alphabetValues[i] = "A";
            }
            else if (i >= 50 && i < 100){
                alphabetValues[i] = "B";
            }
            else if (i >= 100 && i < 150){
                alphabetValues[i] = "C";
            }
            else if (i >= 150 && i < 200){
                alphabetValues[i] = "D";
            }
            else if (i >= 200 && i < 250){
                alphabetValues[i] = "E";
            }
            else if (i >= 250 && i < 300){
                alphabetValues[i] = "F";
            }
            else if (i >= 300 && i < 350){
                alphabetValues[i] = "G";
            }
            else if (i >= 350 && i < 400){
                alphabetValues[i] = "H";
            }
            else if (i >= 400 && i < 450){
                alphabetValues[i] = "I";
            }
            else if (i >= 450 && i < 500){
```

```
            alphabetValues[i] = "L";
        }
        else if (i >= 500 && i < 550){
            alphabetValues[i] = "M";
        }
        else if (i >= 550 && i < 600){
            alphabetValues[i] = "P";
        }
        else if (i >= 600 && i < 650){
            alphabetValues[i] = "Q";
        }
        else if (i >= 650 && i < 700){
            alphabetValues[i] = "R";
        }
        else if (i >= 700 && i < 750){
            alphabetValues[i] = "U";
        }
        else if (i >= 750 && i < 800){
            alphabetValues[i] = "V";
        }
        else if (i >= 800 && i < 850){
            alphabetValues[i] = "W";
        }
        else if (i >= 850 && i < 900){
            alphabetValues[i] = "X";
        }
        else if (i >= 900 && i < 950){
            alphabetValues[i] = "Y";
        }
    }

    valueFingers = new double[fingerTips.size()];
    valueBetweenFingers = new double[fingerTips.size()];

    lines = mRgba;
    if (contourMat.height() > 0 && contourMat.width() > 0 && fingerTips.
        size() != 0) {
        for (int i = 0; i < fingerTips.size(); i++) {

            Point center = new Point(mRgba.width()/2, mRgba.height()/2);
            Point topLeft = new Point(mRgba.width()/division, 0);
            Point topRight = new Point(mRgba.width()/division, mRgba.
                height());
            Point bottomLeft = new Point((division - 1)*mRgba.width()/
                division, 0);
            Point bottomRight = new Point((division - 1)*mRgba.width()/
```

```
                division, mRgba.height());
            Imgproc.circle(lines, center, 5, new Scalar(0, 0, 0));
            Imgproc.line(lines, topLeft, topRight, new Scalar(0, 0, 0), 4);
            Imgproc.line(lines, bottomLeft, bottomRight, new Scalar(0, 0, 0),
                4);

            Point pt1 = fingerTips.get(i);
            Imgproc.line(lines, cogPt, pt1, new Scalar(255, 0, 0), 4);
            Imgproc.circle(lines, pt1, 16, new Scalar(255, 204, 0));
            if (i > 0) {
                Point pt2 = fingerTips.get(i - 1);
                Imgproc.line(lines, pt2, pt1, new Scalar(0, 255, 0), 2);
                valueBetweenFingers[i - 1] = (Math.sqrt(Math.pow((pt1.x -
                    pt2.x), 2) + Math.pow((pt1.y - pt2.y), 2)));
                Log.i("Myapp", Integer.toString(i - 1) + "   " + Integer.
                    toString((int)valueBetweenFingers[i - 1]) + "+");
            }
            valueFingers[i] = (Math.sqrt(Math.pow((pt1.x - cogPt.x), 2) +
                Math.pow((pt1.y - cogPt.y), 2)));
            Log.i("Myapp", Integer.toString(i) + "   " + Integer.toString((int
                )valueFingers[i]) + "=");
            //System.out.println("Myapp" + " " + Integer.toString(i) + "   "
                + Double.toString(abs(valueFingers[i])));
        }
        Imgproc.circle(lines, cogPt, 90, new Scalar(0, 0, 255));
    }

    runOnUiThread(new Runnable() {
        public void run() {
            for (int i = 0; i < values.length; i++) {
                boolean same = true;
                if (values[i].length == valueFingers.length) {
                    for (int j = 0; j < valueFingers.length; j++) {
                        if (valueFingers[j] <= (values[i][j] + 20) &&
                            valueFingers[j] >= (values[i][j] - 20)) {
                            same &= true;
                        } else {
                            same &= false;
                        }
                    }

                    if (same == true && (new String(alphabetValues[i]).
                        equals("A"))){
                        for (int a = 0; a < AValues.length; a++){
                            if (AValues[a].length == valueBetweenFingers.
                                length){
```

```
                    for (int b = 0; b < valueBetweenFingers.
                        length; b++){
                        if (valueBetweenFingers[b] <= (AValues[a
                            ][b] +50) && valueBetweenFingers[b]
                            >= (AValues[a][b] -50)) {
                            same &= true;
                        }
                        else {
                            same &= false;
                        }
                    }
                }
            }
        }
        else if (same == true && (new String(alphabetValues[i
            ]).equals("D"))){
            for (int a = 0; a < DValues.length; a++){
                if (DValues[a].length == valueBetweenFingers.
                    length){
                    for (int b = 0; b < valueBetweenFingers.
                        length; b++){
                        if (valueBetweenFingers[b] <= (DValues[
                            a][b] +50) && valueBetweenFingers[b]
                            >= (DValues[a][b] -50)) {
                            same &= true;
                        }
                        else {
                            same &= false;
                        }
                    }
                }
            }
        }
        else if (same == true && (new String(alphabetValues[i
            ]).equals("E"))){
            for (int a = 0; a < EValues.length; a++){
                if (EValues[a].length == valueBetweenFingers.
                    length){
                    for (int b = 0; b < valueBetweenFingers.
                        length; b++){
                        if (valueBetweenFingers[b] <= (EValues[a
                            ][b] +50) && valueBetweenFingers[b]
                            >= (EValues[a][b] -50)) {
                            same &= true;
                        }
                        else {
```

```
                        same &= false;
                    }
                }
            }
        }
    }
    else if (same == true && (new String(alphabetValues[i
        ]).equals("I"))){
        for (int a = 0; a < IValues.length; a++){
            if (IValues[a].length == valueBetweenFingers.
                length){
                for (int b = 0; b < valueBetweenFingers.
                    length; b++){
                    if (valueBetweenFingers[b] <= (IValues[a
                        ][b] +50) && valueBetweenFingers[b]
                        >= (IValues[a][b] -50)) {
                        same &= true;
                    }
                    else {
                        same &= false;
                    }
                }
            }
        }
    }
    else if (same == true && (new String(alphabetValues[i
        ]).equals("M"))){
        for (int a = 0; a < MValues.length; a++){
            if (MValues[a].length == valueBetweenFingers.
                length){
                for (int b = 0; b < valueBetweenFingers.
                    length; b++){
                    if (valueBetweenFingers[b] <= (MValues[
                        a][b] +50) && valueBetweenFingers[b]
                        >= (MValues[a][b] -50)) {
                        same &= true;
                    }
                    else {
                        same &= false;
                    }
                }
            }
        }
    }
    else if (same == true && (new String(alphabetValues[i
        ]).equals("R"))){
```

```
        for (int a = 0; a < RValues.length; a++){
            if (RValues[a].length == valueBetweenFingers.
                length){
                for (int b = 0; b < valueBetweenFingers.
                    length; b++){
                    if (valueBetweenFingers[b] <= (RValues[a
                        ][b] +50) && valueBetweenFingers[b]
                        >= (RValues[a][b] -50)) {
                        same &= true;
                    }
                    else {
                        same &= false;
                    }
                }
            }
        }
    }
    else if (same == true && (new String(alphabetValues[i
        ]).equals("Y"))){
        for (int a = 0; a < YValues.length; a++){
            if (YValues[a].length == valueBetweenFingers.
                length){
                for (int b = 0; b < valueBetweenFingers.
                    length; b++){
                    if (valueBetweenFingers[b] <= (YValues[
                        a][b] +50) && valueBetweenFingers[b]
                        >= (YValues[a][b] -50)) {
                        same &= true;
                    }
                    else {
                        same &= false;
                    }
                }
            }
        }
    }

    gesture = alphabetValues[i];
    if (same == true) {
        Log.i("Myapp", "----------" +
            alphabetValues[i] + "----------");
    }
    if (same == true && !(new String(gesture).equals(
        previousGesture))) {
        Toast.makeText(MainActivity.this, alphabetValues[i],
            Toast.LENGTH_SHORT).show();
```

```
                                previousGesture = gesture;
                                break;
                            }

                        }
                    }
                }
            });

        }
        return mRgba;
    }
}
```

Appendix B- Progress Report

Android Application to understand British Sign Language using Gesture Recognition

Third Year Individual Project–Progress Report

Nov 2016

Vishisht M. Tiwari

9109605

Supervisor: Dr Hujun Yin

Table of Contents

1. Introduction 1
1.1. Motivation 1
1.2. Aims and objectives 1
2. Literature Review 2
2.1. Principal Component Analysis and background mathematics 2
2.1.1. Covariance 2
2.1.2. Eigenvectors and Eigenvalues 2
2.1.3. Principal Component Analysis 3
2.2. Eigenfaces 3
2.2.1. Generating Eigenfaces using PCA 3
2.3. Real-Time Static Hand Gesture Recognition for American Sign Language (ASL) in Complex Background 4
2.3.1. Image Capturing 4
2.3.2. Image Pre-processing 4
2.3.3. Region Extraction 4
2.3.4. Feature Extraction 4
2.3.5. Feature Matching 5
2.3.6. Results 5
2.4. Indian sign language recognition using Eigen Value weighted Euclidean distance based classification technique 5
2.4.1. Skin Filtering 5
2.4.2. Hand Cropping 6
2.4.3. Feature Extraction 6
2.4.4. Classifier 6
2.4.5. Results 6
3. Progress to Date 6
3.1. Eigenfaces 7
3.2. Facial recognition using eigenfaces 7
3.3. Identifying a face using eigenfaces 7
3.4. Reconstructing face using eigenfaces 8
4. Planning 8
4.1. Proposed system for gesture recognition 9
4.1.1. Image Capturing (Planned hardware) 9
4.1.2. Image Preprocessing 9

4.1.3. Skeletonisation 9
4.1.4. Feature Extraction 9
4.1.5. Feature Matching 10
4.2. Android Application 10
4.3. Alternate Solutions 10
5. Conclusion 10
6. References 11
7. Appendix 12
7.1. Steps for deriving eigenfaces for face recognition 12
7.1.1. Database of face images 12
7.1.2. Training images 12
7.1.3. Converting images into vectors 12
7.1.4. Converting images into vectors 12
7.1.5. Covariance 12
7.1.6. Eigenvectors and Eigenvalues 12
7.1.7. Coefficients 13
7.1.8. Coefficient of test image 13
7.1.9. Euclidean Distance 13
7.2. MATBAB code 14
7.2.1. For facial recognition using 5 training image for each subject 14
7.2.2. Reconstruction of face 15
7.2.3. For identifying face using eigenfaces 17
7.3. Gantt Chart 20
7.4. Risk Assessment 21

1. Introduction

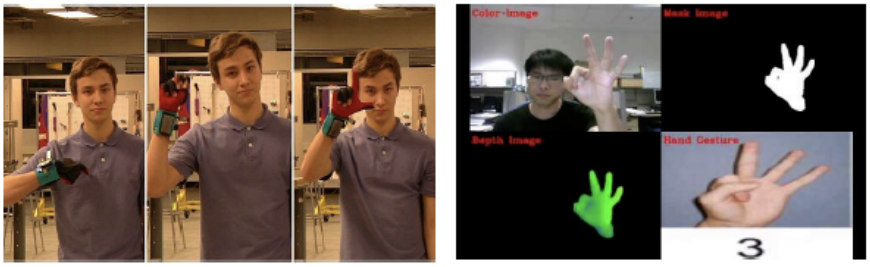


Figure 1: Gesture recognition using gloves and 3-D camera (Online, 2016)

Technology has long been regarded as the solution for all human problems. From the invention of wheel that helped humans in the basic needs of travelling to the latest technological inventions in artificial intelligence that helps in auto maneuvering of cars and planes, technology has always served as the medium that reduces human effort. In recent years, technology has also been influential in making the lives of disabled people easier and bringing them closer to the society.

Advance wheel chairs and self driving cars are few of the first steps in helping disabled population to overcome everyday problems. In field of gesture recognition, many papers are published almost every year that focuses on different ways to understand sign language and convert them into either textual or verbal form. Figure1 show few such concepts that are used for understanding sign language using gesture recognition. However, making these technologies available to the world at a reasonable price is a huge challenge for technology and mankind has still not been able to overcome this challenge.

1.1. Motivation

The motivation behind this project is to make the technology of gesture recognition more approachable by people around the world to help understand the gestures of speech disabled population easily. This project focuses on using affordable technology of mobile phones to understand simple alphabets of British sign language (BSL) using gesture recognition. The final aim of the project will be to develop an android application that can be downloaded from any android phone around the world making this technology approachable to every speech disabled person in the world.

1.2. Aims and objectives

The purpose of this project is to develop a fully functioning android mobile application that recognises the alphabet gestures in British sign language and convert them in a textual form. The project aims to use image processing and machine learning to achieve the desired result. Image processing will be used to convert hand gestures into binary images which will then be recognised using machine learning tools. The project has following main tasks:

- Research about image processing and machine learning.
- Write basic image processing algorithms to produce binary images of gestures and skeletonise the gestures to facilitate their understanding.
- Produce MNIST kind database for gestures and use them as training images for gesture recognition.
- Implement machine learning to recognise the correct gestures.
- Understand the basics of android programming.
- Develop an android application that uses the above mechanisms for gesture recognition.

2. Literature Review

2.1. Principal Component Analysis and background mathematics

An important aspect of a project with gesture recognition is high level image processing. To understand basic image processing, a facial recognition exercise using principal component analysis was first studied.

PCA is a statistical technique that is widely used in finding patterns in high dimension data and is also used in image recognition and compression. The report first describes the mathematical concepts used in PCA such as eigenvalues, eigenvectors and covariance and then goes onto explain the use of PCA in facial recognition using eigenfaces.

2.1.1. Covariance

Statistics as a whole relies on the idea of relationship between each value in a data set. Mean of a data specifies the middle point of the data set, while the standard deviation/variance specifies how spread out the data is from the mean value in 1 dimension. Covariance on the other hand is statistical concept that measures the variance of a data set in 2 dimensions. The formula for covariance is as follows:

$$\mathrm{cov}(X,Y) = \frac{\sum_{i=1}^{n}(X_i - \overline{X})(Y_i - \overline{Y})}{n-1} \quad ---1$$

2.1.2. Eigenvectors and Eigenvalues

Eigenvector, mathematically is a vector which, after a linear transformation, does not change its direction. Eigenvector has been explained below using an example.

$$\begin{pmatrix}2 & 3\\ 2 & 1\end{pmatrix} \mathrm{x} \begin{pmatrix}1\\3\end{pmatrix} = \begin{pmatrix}11\\5\end{pmatrix} \quad ---2$$

$$\begin{pmatrix}2 & 3\\ 2 & 1\end{pmatrix} \mathrm{x} \begin{pmatrix}3\\2\end{pmatrix} = \begin{pmatrix}12\\8\end{pmatrix} = 4 \mathrm{x} \begin{pmatrix}3\\2\end{pmatrix} \quad ---3$$

In both the examples above, $\begin{pmatrix}2 & 3\\ 2 & 1\end{pmatrix}$ is the transformation matrix.

In equation 2, the resulting vector is not a multiple of the original vector and hence is not an eigenvector for the transformation matrix. In equation 3, however, the resulting vector is a multiple of the original vector and hence is an eigenvector for the transformation matrix.

Eigenvalue and eigenvector are closely related topics and are always calculated in pair. Eigenvalue shows the value, the original value is scaled after the transformation. In equation 3, 3 is the eigenvalue associated with the eigenvector $\binom{3}{2}$.

2.1.3. Principal Component Analysis

Principal Component Analysis is a statistical procedure that is used to identify patterns in data and then reveal the similarities and differences of the data. PCA uses a new coordinate system in which every data point gets a new value. These new axes don't have any physical importance and are a combination of heights and weights called principal components that are chosen in such a way that one axis gets a lot of variation. An important advantage of PCA is data compression. PCA compresses the data by reducing the dimensions without losing much information. (Smith, 2002)

2.2. Eigenfaces

Eigenfaces refers to an eigenvectors based approach to face-recognition. Eigenfaces works on the idea of capturing the variation in a collection of face images and use this information to compare of individual faces. Eigenfaces are basically the eigenvectors of the covariance matrix of the set of face images or the principal components of a distribution of face images. In this case, an image with X pixels is considered a vector in X dimensional space.

Figure 2: Eigenfaces

Eigenface approach for face recognition provides two advantages:

- Extract relevant facial information from the images.
- Represent face images more efficiently with reduced computation and space complexity.

2.2.1. Generating Eigenfaces using PCA

A set of Eigenfaces is generated by performing principal component analysis on a large set of facial images. A human face image can then be reconstructed using a combination of the Eigenfaces and average face.

The steps for creating a set of Eigenfaces are as follows:

- A set of training images is prepared

- The mean of all the images has to be taken and then subtracted from each of the images.
- Covariance matrix of all the image is then calculated.
- Eigenvectors and eigenvalues are then calculated from the covariance matrix. This eigenvector is the Eigenface and has the same dimensionality as the original image and hence can be seen as an image itself.
- Arrange the eigenvalues in descending order and sort the eigenvectors accordingly. Choose the principal components. (Turk & Pentland, 1991)

2.3. Real-Time Static Hand Gesture Recognition for American Sign Language (ASL) in Complex Background

This paper describes using a real-world RGB camera for real-time hand gesture recognition. The paper uses four stages for recognition consisting of image preprocessing, region extraction, feature extraction and feature matching. Gesture recognition is only used for recognising alphabetical symbols and is not used for any other complicated gestures. Each of these steps have been described below in detail.

Figure 3: ASL Symbols (Pansare, Gawande, & Maya,

2.3.1. Image Capturing

An RGB image is captured first using a 10 megapixel web camera. Each image captured is of 160 x 120 dimension and has to be pre-processed to reduce computation time.

2.3.2. Image Pre-processing

For skin detection in the image, minimum and maximum skin probabilities of input RGB image are calculated. Data from normalised skin probability is used for producing gray threshold at zero which produces a binary image. Noise reduction is also achieved in this step by applying median filtering to red channel to reduce "salt and pepper" noise. The binary image is then passed through Gaussian filer to smooth the image.

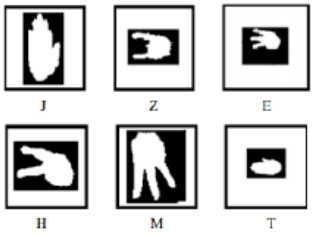


Figure 4: After Image Preprocessing (Pansare, Gawande, & Maya, 2012)

2.3.3. Region Extraction

Connectivity of binary image is calculated using "bwlebel" method and "regionprops" is applied to find bounding box for the hand and to extract the region of interest.

2.3.4. Feature Extraction

Feature vector is formed using centroid given by (C_x, C_y) .

$$C_X, C_y = \frac{1}{n}\left(\sum_{t=1}^{n} X_t, \sum_{t=1}^{n} Y_t\right) \quad ---4$$

Area of the hand is calculated using the "bwarea" command in MATLAB.

2.3.5. Feature Matching

Feature matching is done by calculating the Euclidean distance between feature vector of input real-time image and that of the training images.

$$E_D = \sqrt{(X_t - X_1)^2 + (Y_t - Y_1)^2} \quad ---5$$

The smallest Euclidean distance is used for gesture recognition.

2.3.6. Results

This paper used 100 gestures for each alphabet and the success rate was measured depending on how many gestures were correctly recognised for each alphabet. The method received a high success rate with highest accuracy of 100% for letter E and V and lowest accuracy of 85% for letter I and U. (Pansare, Gawande, & Maya, 2012)

2.4. Indian sign language recognition using Eigen Value weighted Euclidean distance based classification technique

The paper uses the concept of eigenvectors and eigenvalues to recognise two hand gestures with an accuracy of 97%. The proposed system in the paper considers 24 alphabets of Indian sign language. Each alphabet has 10 training images making the total database size of 240 images. The steps followed for gesture recognition has been described below:

Figure 5: 24 alphabets of Indian Sign Language (Singha & Karen, 2013)

2.4.1. Skin Filtering

The first phase of the paper describes the process of extracting skin coloured pixels from non skin coloured pixels.

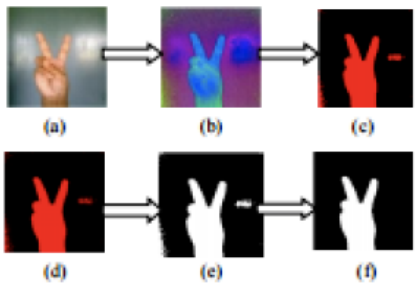


Figure 6: a) RGB image, b) HSV image, c) Filtered image, d) Smoothened image, e) Grayscale image, f) Biggest binary object (Singha & Karen, 2013)

The image is first converted from RGB image to HSV image by separating hue, saturation and value making the images less sensitive to illumination. The HSV image is then filtered, smoothened and finally a grayscale image is obtained which only comprises of skin colored pixels. To eliminate other skin colored object in the background such as a dress or wood, the biggest binary linked object is extracted which in most cases will be the hand.

2.4.2. Hand Cropping

The next step described in the paper involves cropping the hand to the wrist part. This eliminates the undesired region and makes the image smaller which will be faster for computations. The following steps are involved for hand cropping:

- The image after filtering is scanned from all directions to detect the wrist.
- Once the wrist is located, the maximum and minimum positions of the white pixels is identified. The X_{min}, X_{max}, Y_{min}, Y_{max} is obtained. One of these coordinates is the position of the wrist.
- The image is cropped along the specified coordinates.

2.4.3. Feature Extraction

This step involves calculating the eigenvalues and eigenvectors from the cropped image. The method for calculating eigenvectors has already been discussed previously in the report.

The paper only considers first 5 significant eigenvectors and the rest are neglected. This process, as discussed before, provides advantages in terms of data compression without loss of much information.

2.4.4. Classifier

The paper uses classifier to recognise different hand gestures. The paper has devised a new classification technique that is eigenvalue weighted Euclidean distance between eigenvectors. This involves two levels of classification:

- Classification based on Euclidean distance: In the paper, the Euclidean distance was found between the eigenvectors of the test image and the database images.
- Classification based on Eigen value weighted Euclidean distance: According to this new approach, the difference of eigenvalues between test image and database image was calculated and multiplied with the Euclidean distance calculated before. The sum of results for each image was added and the minimum value was considered the gesture.

2.4.5. Results

The paper used 10 images for each symbol and was able to achieve a success rate of 97%. The paper used both the classifiers for recognition and the second classification way improved the success rate from 87% to 97%. (Singha & Karen, 2013)

3. Progress to Date

As discussed in the literature review section, to gain a better understanding of image processing, the project was started by facial recognition exercises using Principal

component analysis. PCA is used for finding patterns in high dimension data and in this case, is used for capturing the variation in a collection of face images. The eigenvectors produced in this case is called eigenfaces.

It is important to note that PCA is used here to recognise different faces as an exercise and can be used to identify different gestures as well. Even though PCA is a very powerful image processing tool, this project will not be using PCA for image processing and is only used as an exercise technique. It is also important to note that all the programs described below used 200 eigenfaces and the rest of the eigenfaces were neglected.

3.1. Eigenfaces

The program for facial recognition using eigenfaces was developed on MATLAB. The program uses a database of training images to find patterns and recognise variations in facial features. In this project, the database contains 1 to 9 facial images of 40 subjects. Different number of training images for each subject is loaded to understand the effects of training images on success rate of the program. The detailed explanation for deriving the eigenfaces and the MATLAB code for different eigenfaces programs has been described in the appendix 6.1.

3.2. Facial recognition using eigenfaces

For the purpose of facial recognition, a certain number of images for each subject has to be loaded into the program as training images. This number varied from 1 image to 9 images for each subject. After calculating the eigenvector and eigenvalue for each training image, the eigenvector of test image was calculated. The lowest Euclidean distance with the training image would determine the recognised face. Facial recognition was achieved using 1, 5 and 9 images for each subject. The success rate in each case has been specified in table 1. MATLAB code for this program has been given in appendix 6.2.1.

No. of training image for each subject	Success rate
1	68.05%
5	90.5%
9	92.5%

Table 1: Success rate for facial recognition using eigenfaces with different no. of training images

3.3. Identifying a face using eigenfaces

The program was also used to identify faces rather than recognise them. For this purpose, the program was loaded with 200 training images from 40 different subjects. After executing principal component analysis

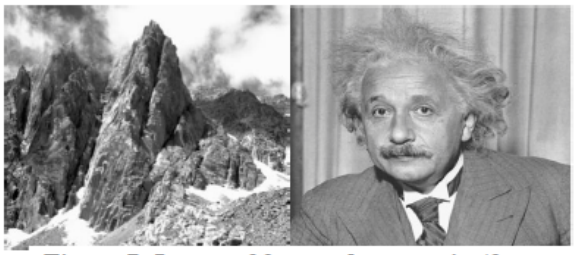


Figure 7: Image of face and mountain (featured from http://www.plexoft.com/cgi-bin/P.cgi)

on all of the images, the program recognised the similarities and variations of facial features. To test the identification feature, the program was loaded with a grayscale image of Dr Albert Einstein and grayscale image of a mountain as shown in figure 7.

Figure 8 shows the difference in coefficients that was achieved when program was loaded with the image of a mountain and the image of the face. As it can be seen, the discrepancy is much larger in the case of mountains than in the case of the face. This discrepancy is used to identify face image from a non face image. MATLAB code is given in appendix 6.2.3.

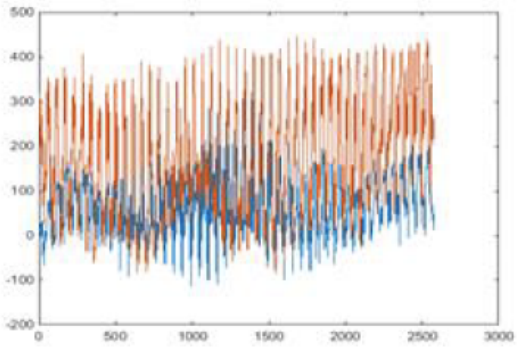


Figure 8: Difference between eigenvectors for face and image (Blue waveform is for face and Orange waveform is for mountain)

3.4. Reconstructing face using eigenfaces

Another task achieved using the program was to reconstruct each training image from the eigenfaces. This task helps to understand the amount of information lost by neglecting eigenvectors. The images shown below show the reconstructed image of one of the subject. Figure 9 shows the reconstructed image when different number of eigenfaces were used for reconstruction.

Figure 9: Reconstructing images using eigenfaces. A) Original Image, B) 500 Eigenfaces, C) 200 Eigenfaces, D) 100 Eigenfaces, E) 20 Eigenfaces

Figure 9 is a perfect example that shows the amount of information lost with each eigenface. Another important point to note from figure 9 is that 200 eigenfaces are enough to contain 99% information of each face. It only losses few details that are not very important for the purpose of facial recognition. MATLAB code is given in appendix 6.2.2.

4. Planning

After analysing different papers regarding gesture recognition and going through the image processing exercise, the project will start focusing on the gesture recognition for British sign language. Gantt chart for the project has been given in appendix 6.3.

As discussed before, the project will only be recognising very simple alphabet gestures in British sign

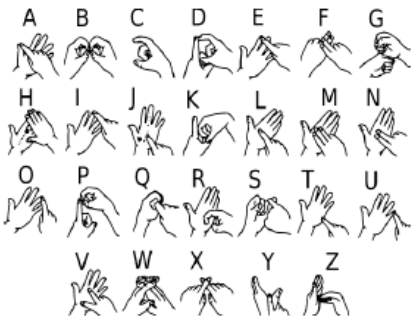


Figure 10: Alphabets in British Sign Language

language. The steps for gesture recognition system which will be used has been described below in detail. It is important to note that there are many ways to solve the problem of gesture recognition but the following method has been used because this method provides a high success rate in other papers and is expected to finish in given time.

4.1. Proposed system for gesture recognition

Figure 11 shows the block diagram for the proposed gesture recognition system that will be used in the project.

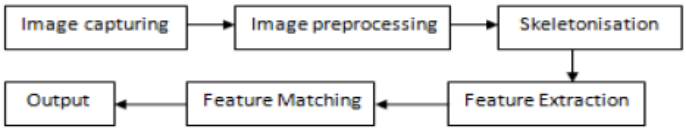


Figure 11: Block diagram of the proposed system

The following steps describe in detail the processes that will be used for gesture recognition programme.

4.1.1. Image Capturing (Planned hardware)

Because the project is expected to work as an application, the camera for image capture will be an RGB camera of a mobile phone.

4.1.2. Image Preprocessing

After capturing an RGB image, the image is then converted into an HSV image to reduce the effects of illumination. As discussed in literature review section, HSV color space separates three components: Hue, Saturation and Value. The image will then be smoothened and filtered and converted into a grayscale image where skin colored objects will become white pixels and non skin colored objects will become black colored. The biggest binary linked object will be considered to eliminate other skin colored object such as a dress or wood branch.

4.1.3. Skeletonisation

The next step will be to work on Skeletonisation of the hands by extracting the extreme points of the hand. This will be achieved using star Skeletonisation which is a process to extract the feature points from the foreground. The feature consists of vectors that join the extremities of the hand to the centroid of the hand. Extremities are calculated using boundary tracking in a clockwise or anti-clockwise direction.

4.1.4. Feature Extraction

Feature extraction involves extracting the data from the skeletonised image such as distance of extremities of the hand from the centroid and saved to be tested against a database of gestures.

4.1.5. Feature Matching

The feature extracted will be checked against a database of training images of different gestures in British sign language alphabets. The database will be an MNIST kind database. MNIST is a database of handwritten digits that is used as training images for image processing systems. The project will aim to use a database like this for the gestures and feature extracted from skeletonised image will be checked against each image in this database to recognise the gesture.

4.2. Android Application

What separates this project from every other gesture recognition project is the idea of implementing it on the phone. For this purpose, the programme will be implemented as android application in java programming language. The project will be built as an android application because of the availability of high number of libraries in java for gesture recognition. The allotted budget of £150 will be used to buy an android phone that will be used to develop the android application and will be used to show the outcome of the project in the final presentation. The project will require a high end phone because of the following reasons:

- The phone needs to have a good camera, at least 10 megapixels to be able to get the right resolution images for image processing.
- The processor of the phone should be powerful enough to be able to perform image processing as fast as possible.

4.3. Alternate Solutions

The following section provides few alternate solutions to gesture recognition system in case the above described system starts facing problems:

- Use OpenCV libraries for gesture recognition.
- Use PCA to understand gestures and use Euclidean distance to recognise gestures.
- Using fuzzy logic and other machine learning tools to implement gesture recognition.

5. Conclusion

This report aims at giving a summary of the work done till now and provides a plan of the future work that needs to be completed in the report. The project has till now focused on gaining the right technical knowledge to implement a complicated gesture recognition programming as an android application. The focus of the project will now switch to completing the practical implementations of the project and achieving a gesture recognition that can understand as many British sign language alphabets as possible. The long term aim of the project is to provide a disability solution for speech disabled people to be able to interact will anyone in their environment using sign language.

6. References

[1] Darwish, S. M., Madbouly, M. M., & Khorsheed, M. B. (2016). Hand Gesture Recognition for Sign Language: A New Higher Order Fuzzy HMM Approach. *International Journal of Engineering and Technology* , 157-164.

[2] Liang, H. (2013). *Robust Hand Gesture Recognition with the Depth Camera*. Retrieved from https://www.youtube.com/watch?v=YLonOoXy8-Y

[3] Online, M. (2016, 4 25). *Mail Online*. Retrieved 11 5, 2016, from Mail Online: http://www.dailymail.co.uk/sciencetech/article-3557362/SignAloud-gloves-translate-sign-language-movements-spoken-English.html

[4] Pansare, J. R., Gawande, S. H., & Maya, I. (2012). Real-Time Static Hand Gesture Recognition for American. *Journal of Signal and Information Processing* , 364-367.

[5] Roomi, S. M., Priya, R. J., & Jayalakshmi, H. (2010). Hand Gesture Recognition for Human-Computer Interaction . *Journal of Computer Science* , 1002-1007.

[6] Sherman, P. (n.d.). *British sign language alphabet*. Retrieved 11 5, 2016, from British sign language alphabet: http://www.wpclipart.com/sign_language/British_sign_language_alphabet.png.html

[7] Singha, J., & Karen, D. (2013). Indian Sign Language Recognition Using EigenValue Weigted Euclidean distance Based Classification Technique. *International Journal of Advanced Computer Science and Applications* , 188-195.

[8] Sirovich, L., & Kirby, M. (1987). Low-dimensional procedure for the characterization of human faces. *Optical Society of America* , 519-524.

[9] Smith, L. I. (2002). *A tutorial on Principal Components Analysis.*

[10] Turk, M., & Pentland, A. (1991). Eigenfaces for Recognition. *Jounal of Cognitive Neuroscience* , 71-86.

[11] Viola, P., & Jones, M. (2001). Robust Real-time Object Detection. *Second International Workshop on Statistical and Computational Theories of Vision- Modelling, Learning, Computing, and Sampling*, (pp. 1-25). Vancouver.

[12] Yin, H. EEEN60178: Digital Image Engineering. In *Lecture Notes.* Manchester, United Kingdom.

[13] Zhang, S., & Turk, M. (2008). *Eigenfaces*. Retrieved 11 1, 2016, from http://www.scholarpedia.org/: http://www.scholarpedia.org/article/Eigenfaces

7. Appendix

7.1. Steps for deriving eigenfaces for face recognition

As a basic exercise for image processing, eigenfaces were used for facial recognition from static face images. MATLAB was used for executing face recognition using eigenfaces and the steps for developing the program has been described below.

7.1.1. Database of face images

A database of training images is used to test the software for face recognition. The database used in this case is the database of faces from the AT&T Laboratories in Cambridge. The database contains 10 different images of distinct subjects. Each of these images was taken with different facial expressions at different times with varying light conditions. All the images were taken against a dark homogenous background.

7.1.2. Training images

For face recognition, few images from each subject was loaded onto the program to form a set of eigenfaces. These are called the training images as these are used to train the program to learn about the facial details. Each training image was converted in 56 x 46 dimension before starting. The accuracy of the program can increase or decrease depending on the number of training images that are loaded into the program. This can be seen from the facial recognition exercise explained in the progress to date section of this report.

7.1.3. Converting images into vectors

Principal component analysis uses orthogonal transformation for obtainig linearly correlated data. This step is done mostly on vectors. Hence to perform pricipal component analysis, each image is treated as a vector by transforming it into column of pixels of 2576 x 1.

7.1.4. Converting images into vectors

The average of all the image vectors are calculated and then subtracted from each of the images to study the variability of each image.

7.1.5. Covariance

The next step is to perform the covariance of the matrix of images. This is performed by combining all the training images into a matrix(M). The covariance is then calculated by matrix multiliplication of M and transpose of M.

7.1.6. Eigenvectors and Eigenvalues

Eigenvectors and eigenvalues are then calculated from the covariance matrix. Calculating eigenvectors from such a large matrix requires a complicated iterative method. For this purpose, the eig command of the matrix was used to calulate the eigenvector and eigenvalues.

The eigenvalues are then arranged in descending order and eigenvectors are sorted accordingly as discussed in literature review.

Each of the eigenvectors has the same dimensions as the the original image and itself can be formed as an image. The total number of eigenvectors obtained in this case is 2576 but most of the information can be obtained from the first few eigenvectors. The following table gives an idea about the information lost by choosing a certain number of eigenvectors.

No. of Eigenvectors	Information saved
60	86.65%
100	91.68%
200	97.06%
400	99.99%

Table 2: Information saved against number of eigenvectors used

As it can be seen from the above table, most of the information can be obtained from the first 200 eigenvectors and hence the code used in the project uses the first 200 eigenvectors for face recognition.

The eigenvector calculated here are called eigenfaces.

7.1.7. Coefficients

For face recognition, coefficient for each of the test image is calculated by using matrix multiplication of eigenvector with each subtracted image vector. This gives a coefficient associated with each training image.

7.1.8. Coefficient of test image

A test image is an image that is tested against all other training images to recognise the most similar face. For eigenfaces method, the coefficient of this particular image is also calculated.

7.1.9. Euclidean Distance

Once the coefficient is derived, Euclidean distance is calculated between the coefficient of the test image and the coefficient of every training image. The training image with the least distance will be recognised as the facial image.

$$ED_{(p,q)} = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \qquad ---6$$

7.2. MATBAB code

7.2.1. For facial recognition using 5 training image for each subject

```
clear all
close all

num_Images = 0; % initialise number of images to zero

% Extract training images from the folder address. The first for loop
% specifies the subject no. and the second specifies the image no.
for i = 1:40
    for j = 1:5
        filelocation =
strcat('C:\Users\Vishisht\Desktop\Eigenfaces\att_faces\s', num2str(i), '\',
num2str(j), '.pgm');
        filenames = dir(filelocation);
        imageset{j, i} = imresize(imread(filelocation),[56,46]);
    end
end

% Convert the images into column vector. This loop also saves the total no.
% of images
for i = 1:40
    for j = 1:5
        num_Images = num_Images+1;
        column_Imageset{num_Images} = imageset{j,i}(:);
    end
end

% Vectors are added to calculate the average of all training image
sum_Image = 0;
for i = 1:num_Images
    sum_Image = imadd((double(column_Imageset{i})),sum_Image);
end

average_Image = (sum_Image/200); % average of all 200 vectors are
calculated

% Average vector is subtracted from each vector images
for i = 1:num_Images
    subtracted_Imageset{i} =
imsubtract(double(column_Imageset{i}),average_Image);
end

% All vectors are combined together to form a matrix so that covariance
% matric can be calculated
for i = 1:num_Images
    combination_Column_Imageset(:,i) = subtracted_Imageset{i};
end

% C is the covariance matrix
C = combination_Column_Imageset*combination_Column_Imageset';

% 'eigs' command is used to calculate first 200 eigenvectors and
% eigenvalues
[V,D] = eigs(C,200);

% Arrange eigenvalue in descanding order and sort corresponding
```

```
% eigenvectors
eigval = diag(D);
eigval = eigval(end:-1:1);
V = -V;

% Coefficients for each image is calculated
for i = 1:num_Images
    coefficients{i} = V'*double(subtracted_Imageset{i});
end

% The following loop extracts the test image for each subject, calculates
% the coefficient and checks if the image identified is the right image
% recognised or not.
correct = 0;
error = 0;
for i = 1:40
    labels{i} = (5*i)+[-4,-3,-2,-1,0];
    for j = 6:10
        test_Filelocation =
strcat('C:\Users\Vishisht\Desktop\Eigenfaces\att_faces\s', num2str(i), '\',
num2str(j), '.pgm');
        test_filename = dir(test_Filelocation);
        test_Image = imresize(imread(test_Filelocation),[56,46]);
        test_Column_Image = test_Image(:);
        test_Subtracted_Image =
imsubtract(double(test_Column_Image),average_Image);
        test_C =
double(test_Subtracted_Image)*double(test_Subtracted_Image');
        test_coefficients = V'*double(test_Subtracted_Image);

        % Euclidean ditance is calculated
        euclidean_Distance = sqrt(sum((coefficients{1} -
test_coefficients).^2));
        image_Number = 1;
        for k = 2:num_Images
            new_Euclidean_Distance = sqrt(sum((coefficients{k} -
test_coefficients) .^2));
            if (new_Euclidean_Distance < euclidean_Distance)
                euclidean_Distance = new_Euclidean_Distance;
                image_Number = k;
            end
        end
        % The followng loop checks if the image recognised is correct or
        % not
        if (ismember(image_Number, labels{i}))
            correct = correct + 1;
        else
            error = error + 1;
        end
    end
end

% The following comments calculate and print the efficiency.
efficiency = (correct/(400 - num_Images))*100;
strcat('The efficiency of the program is: ', num2str(efficiency), '%')
```

7.2.2. Reconstruction of face

```
clear all
close all
```

```
num_Images = 0; % initialise number of images to zero

% Extract training images from the folder address. The first for loop
% specifies the subject no. and the second specifies the image no.
for i = 1:40
    for j = 1:5
        filelocation =
strcat('C:\Users\Vishisht\Desktop\Eigenfaces\att_faces\s', num2str(i), '\',
num2str(j), '.pgm');
        filenames = dir(filelocation);
        imageset{j, i} = imresize(imread(filelocation),[56,46]);
    end
end

% Convert the images into column vector. This loop also saves the total no.
% of images
for i = 1:40
    for j = 1:5
        num_Images = num_Images+1;
        column_Imageset{num_Images} = imageset{j,i}(:);
    end
end

% Vectors are added to calculate the average of all training image
sum_Image = 0;
for i = 1:num_Images
    sum_Image = imadd((double(column_Imageset{i})),sum_Image);
end

average_Image = (sum_Image/200); % average of all 200 vectors are
calculated

% Average vector is subtracted from each vector images
for i = 1:num_Images
    subtracted_Imageset{i} =
imsubtract(double(column_Imageset{i}),average_Image);
end

% All vectors are combined together to form a matrix so that covariance
% matric can be calculated
for i = 1:num_Images
    combination_Column_Imageset(:,i) = subtracted_Imageset{i};
end

% C is the covariance matrix
C = combination_Column_Imageset*combination_Column_Imageset';

% 'eigs' command is used to calculate first 200 eigenvectors and
% eigenvalues
[V,D] = eigs(C,200);

% Arrange eigenvalue in descanding order and sort corresponding
% eigenvectors
eigval = diag(D);
eigval = eigval(end:-1:1);
V = -V;

% Coefficient and weights of the image to be reconstructed is calculated
```

```
test_Filelocation =
strcat('C:\Users\Vishisht\Desktop\Eigenfaces\att_faces\s4\1.pgm');
test_filename = dir(test_Filelocation);
test_Image = imresize(imread(test_Filelocation),[56,46]);
test_Column_Image = test_Image(:);
test_Subtracted_Image =
imsubtract(double(test_Column_Image),average_Image);

coefficients = V'*double(test_Subtracted_Image);

weights = V*coefficients;

% Reconstructed image is formed using weights and average image
reconstructed_Images = weights + average_Image;

% Reconstructed image is shown
imshow(mat2gray(reconstructed_Images));
```

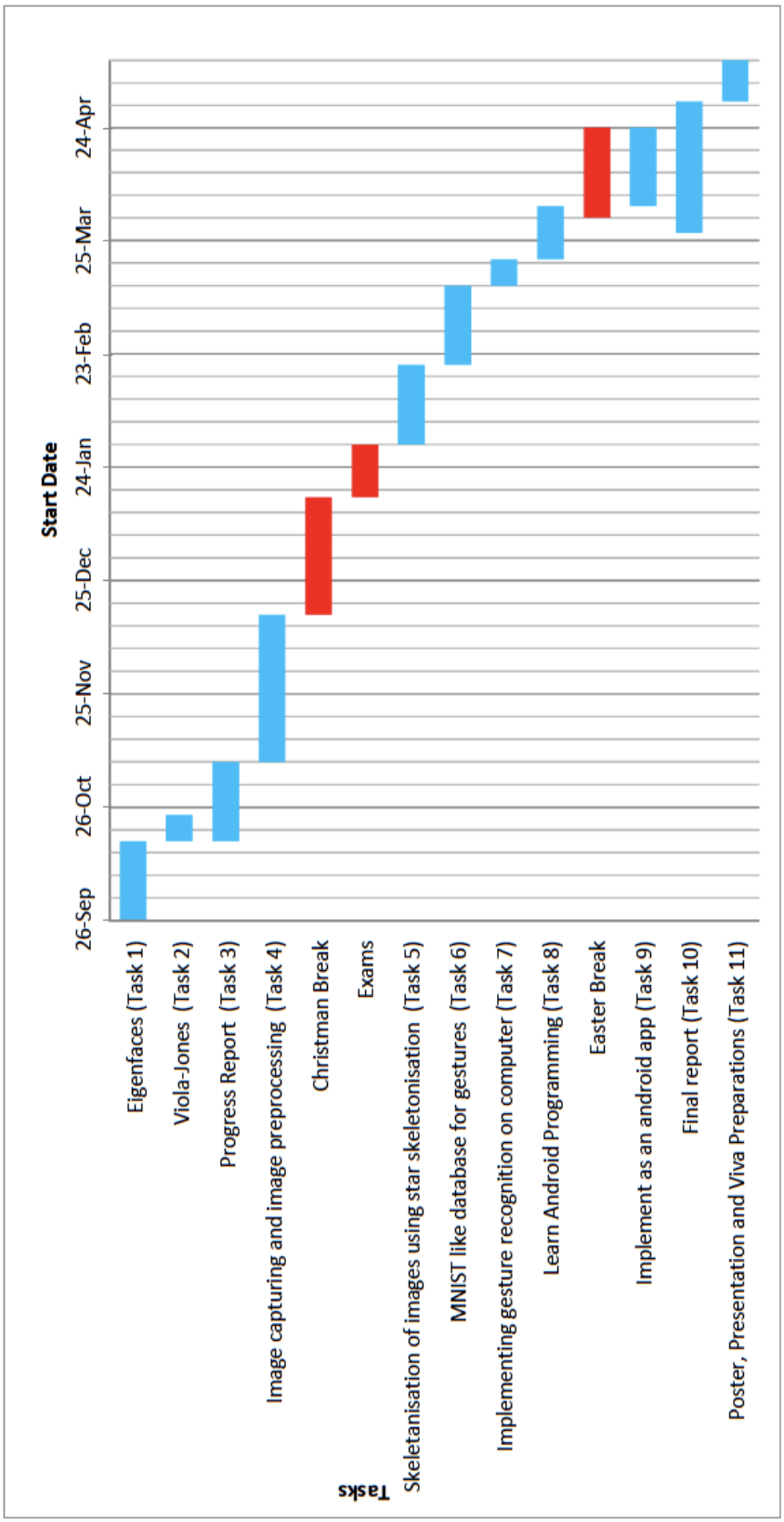
7.2.3. For identifying face using eigenfaces

```
clear all
close all

num_Images = 0; % initialise number of images to zero

% Extract training images from the folder address. The first for loop
% specifies the subject no. and the second specifies the image no.
for i = 1:40
    for j = 1:5
        filelocation =
strcat('C:\Users\Vishisht\Desktop\Eigenfaces\att_faces\s', num2str(i), '\',
num2str(j), '.pgm');
        filenames = dir(filelocation);
        imageset{j, i} = imresize(imread(filelocation),[56,46]);
    end
end

% Convert the images into column vector. This loop also saves the total no.
% of images
for i = 1:40
    for j = 1:5
        num_Images = num_Images+1;
        column_Imageset{num_Images} = imageset{j,i}(:);
    end
end

% Vectors are added to calculate the average of all training image
sum_Image = 0;
for i = 1:num_Images
    sum_Image = imadd((double(column_Imageset{i})),sum_Image);
end

average_Image = (sum_Image/200); % average of all 200 vectors are
calculated

% Average vector is subtracted from each vector images
for i = 1:num_Images
```

```
    subtracted_Imageset{i} =
imsubtract(double(column_Imageset{i}),average_Image);
end

% All vectors are combined together to form a matrix so that covariance
% matric can be calculated
for i = 1:num_Images
    combination_Column_Imageset(:,i) = subtracted_Imageset{i};
end

% C is the covariance matrix
C = combination_Column_Imageset*combination_Column_Imageset';

% 'eigs' command is used to calculate first 200 eigenvectors and
% eigenvalues
[V,D] = eigs(C,200);

% Arrange eigenvalue in descanding order and sort corresponding
% eigenvectors
eigval = diag(D);
eigval = eigval(end:-1:1);
V = -V;

% Coefficients for each image is calculated
for i = 1:num_Images
    coefficients{i} = V'*double(subtracted_Imageset{i});
end

% Weights for each image is calculated by multiplying vectors and
% coefficients
for i = 1:num_Images
    weights{i} = V*coefficients{i};
end

Sample = 0;
for i = 1:num_Images
    Sample = weights{i} + Sample;
end

Sample = Sample + average_Image;

% Eigenvector, coefficients and weight for non face image is calculated
test_M_Filelocation =
strcat('C:\Users\Vishisht\Desktop\Eigenfaces\NoFace1.pgm');
test_M_filename = dir(test_M_Filelocation);
test_M_Image = imresize(imread(test_M_Filelocation),[56,46]);
test_M_Column_Image = test_M_Image(:);
test_M_Subtracted_Image =
imsubtract(double(test_M_Column_Image),average_Image);
test_M_C =
double(test_M_Subtracted_Image)*double(test_M_Subtracted_Image');
test_M_coefficients = V'*double(test_M_Subtracted_Image);
test_M_weights = V*test_M_coefficients;
test_M_Sample = test_M_weights + double(test_M_Column_Image);

% Eigenvector, coefficients and weight for face image is calculated
test_F_Filelocation =
strcat('C:\Users\Vishisht\Desktop\Eigenfaces\Face1.pgm');
test_F_filename = dir(test_F_Filelocation);
```

```
test_F_Image = imresize(imread(test_F_Filelocation),[56,46]);
test_F_Column_Image = test_F_Image(:);
test_F_Subtracted_Image =
imsubtract(double(test_F_Column_Image),average_Image);
test_F_C =
double(test_F_Subtracted_Image)*double(test_F_Subtracted_Image');
test_F_coefficients = V'*double(test_F_Subtracted_Image);
test_F_weights = V*test_F_coefficients;
test_F_Sample = test_F_weights + double(test_F_Column_Image);

% difference between the weights for training images and face and non face
% image is calculated and plotted below
Difference_F = test_F_Sample - Sample;
Difference_M = test_M_Sample - Sample;
plot(test_F_Sample);
hold on;
plot(test_M_Sample);
hold off;
```

7.3. Gantt Chart



Tasks	Task 1	Task 2	Task 3	Task 4	Christmas	Exams	Task 5	Task 6	Task 7	Task 8	Easter	Task 9	Task 10	Task 11
Start Date	26-Sep	17-Oct	17-Oct	7-Nov	16-Dec	16-Jan	30-Jan	20-Feb	13-Mar	20-Mar	31-Mar	3-Apr	27-Mar	1-May
Duration (days)	21	7	21	39	31	14	21	21	7	14	24	21	35	11

7.4. Risk Assessment

LOCATION/ACTIVITY: D45a, Sackville Street Building

WORK ACTIVITY/ WORKPLACE (WHAT PART OF THE ACTIVITY POSES RISK OF INJURY OR ILLNESS)	HAZARD (S) (SOMETHING THAT COULD CAUSE HARM, ILLNESS OR INJURY)	LIKELY CONSEQUENCES (WHAT WOULD BE THE RESULT OF THE HAZARD)	WHO OR WHAT IS AT RISK (INCLUDE NUMBERS AND GROUPS)	EXISTING CONTROL MEASURES IN USE (WHAT PROTECTS PEOPLE FROM THESE HAZARDS)	WITH EXISTING CONTROLS				MEASURE REQUIRED TO PREVENT OR REDUCE RISK (WHAT NEEDS TO BE DONE TO MAKE THE ACTIVITY AS SAFE AS POSSIBLE)	PERSON RESPONSIBLE FOR ACTIONS AND AGREED TIMESCALES TO ACHIEVE THEM	WITH NEW CONTROLS			
					SEVERITY		RISK RATING	RISK ACCEPTABLE			SEVERITY		RISK RATING	RISK ACCEPTABLE
Working on Computer	Headache and eyestrain caused by screen glare	Slight Minor injury / illness – immediate 1st Aid only / slight loss	Mr Vishisht M. Tiwari	Suitable Room Lighting	2	2	4	YES	Position monitor correctly. Take breaks	Mr Vishisht M. Tiwari immediately	2	1	2	yes
Working on Computer	Fatigue and backaches due to bad posture	Slight Minor injury / illness – immediate 1st Aid only / slight loss	Mr Vishisht M. Tiwari		2	2	4	YES	Improve workstation ergonomics. Take breaks	Mr Vishisht M. Tiwari immediately	2	1	2	yes
Working on Computer	Prolonged use of keyboard and mouse can result in upper limb disorders	Slight Minor injury / illness – immediate 1st Aid only / slight loss	Mr Vishisht M. Tiwari		2	2	4	YES	Improve workstation ergonomics. Take breaks	Mr Vishisht M. Tiwari immediately	2	1	2	yes
General Work	Stress caused by unforeseen problems, long hours and tight deadlines	Slight Minor injury / illness – immediate 1st Aid only / slight loss	Mr Vishisht M. Tiwari		2	2	4	YES	Consult supervisor if struggling with workload. Take breaks	Mr Vishisht M. Tiwari immediately	2	1	2	yes

General Work Environment	Slips & Trips	Moderate Injury/ illness of 3 days or more absence (reportable category)/ Moderate	Mr Vishisht M. Tiwari		3	1	3	Y E S	Keep work place clean.	Mr Vishisht M. Tiwari immediately	3	1	3	Y e s

RISK ASSESSOR	**NAME: Vishisht M. Tiwari**	**SIGNED:VISHISHT M. TIWARI**	**Revised 06-11-16**	**THIS RISK ASSESSMENT WILL BE SUBJECT TO A REVIEW NO LATER THAN: (12 months)**

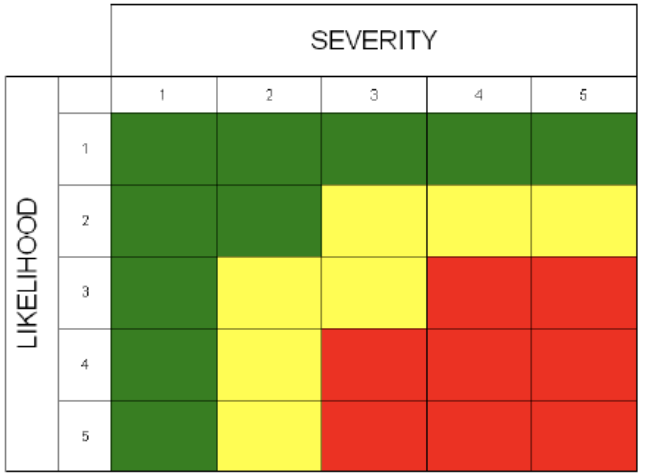
SEVERITY VALUE = Potential consequence of an incident/injury given current level of controls.

5 Very High Death / permanent incapacity / widespread loss
4 High Major Injury (Reportable Category) / Severe Incapacity / Serious Loss
3 Moderate Injury / illness of 3 days or more absence (reportable category) / Moderate loss
2 Slight Minor injury / illness - immediate 1st Aid only / slight loss
1 Negligible No injury or trivial injury / illness / loss

LIKELIHOOD = what is the potential of an incident or injury occurring given the current level of controls.

5 Almost certain to occur
4 Likely to occur
3 Quite possible to occur
2 Possible in current situation
1 Not likely to occur

The intersection of the chosen column with the chosen row is the Risk classification.



1- 5 LOW - Tolerable - Monitor and Manage

6 - 10 MEDIUM - Review and introduce additional controls to mitigate to ALARP

12 - 25 HIGH - Intolerable Stop Work and immediately introduce further control measures