CHAPTER 1
INTRODUCTION

Human gestures constitute a space of motion expressed by the body, face, and/or hands. Among a variety of gestures, hand gesture is the most expressive and the most frequently used. Gestures have been used as an alternative form to communicate with computers in an easy way. This kind of human-machine interfaces would allow a user to control a wide variety of devices through hand gestures. Most work in this research field tries to elude the problem by using markers, marked gloves or requiring a simple background. Glove-based gesture interfaces require the user to wear a cumbersome device, and generally carry a load of cables that connect the device to a computer. A real-time gesture recognition system which can recognize 46 ASL letter spelling alphabet and digits was proposed. The gestures that are recognized by are static gestures without any motion. This paper introduces a hand gesture recognition system to recognize 'dynamic gestures' of which a single gesture is performed in complex background. Unlike previous gesture recognition systems, our system neither uses instrumented glove nor any markers. The new barehanded proposed technique uses only 2D video input. This technique involves detecting the hand location, tracking the trajectory of the moving hand, and analysing the hand-position variations. Then the obtained motion information is been used in the recognition phase of the gesture. Our Hand Gesture Recognition system generally involves two stages:

Hand Detection:

The technique is built around the idea that Splits the input video screen into two parts and processes each part separately, so that the processing in the two parts is similar and simultaneous. The operators used for image processing must be kept low time consuming in order to obtain the fast processing rate needed to achieve real time speed. The detection steps are:

(a) Skin tone detection
(b) Motion detection
(c) Combination of motion & skin color

(d) Region Identification

Hand Recognition:

Since the considered gestures are processed dynamically, we need a mechanism to recognize gesture using their axial movement. The recognition scheme must be roubust and should accommodate variation in the attributes gesture.

At this stage, we divide the analysis phase into two modules. (a) One hand moveme analysis. The angle and distanc of the hand centroid, and the counters of the movements in each direction were used to recognize the gestures.

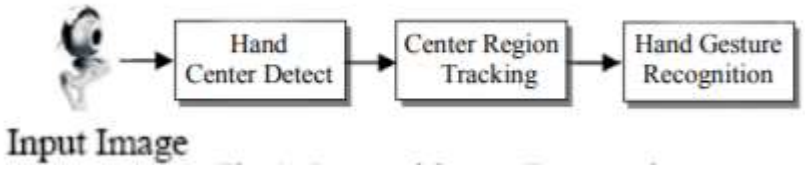

Fig 1.1. Hand Detection System as a Part of Hand Gesture Recognition System

1.1 Computer Vision:

Computer vision is a field that includes methods for acquiring, processing, analyzing, and understanding images and, in general, high-dimensional data from the real world in order to produce numerical or symbolic information, e.g., in the forms of decisions. A theme in the development of this field has been to duplicate the abilities of human vision by electronically perceiving and understanding an image. Computer vision is the transformation of data from a still or video camera into either a decision or a new representation.

The classical problem in computer vision, image processing, and machine vision is that of determining whether or not the image data contains some specific object, feature, or activity. The decision might be “there is a persons hand in this scene” or “what the color of the car is?” This task can normally be solved robustly and without effort by a human, but is still not satisfactorily solved in computer vision for the general case: arbitrary objects in arbitrary situations. The existing methods for dealing with this problem can at best solve it only for specific objects, such as simple geometric objects (e.g., polyhedral), human faces, printed or hand-written characters, or vehicles, and in specific situations, typically described

in terms of well-defined illumination, background, and pose of the object relative to the camera.

1.2 OpenCV:

OpenCV is a Intel® Open Source Computer Vision Library. It is a collection of C functions and a few C++ classes that implement some popular Image Processing and Computer Vision algorithms.

OpenCV (Open Source Computer Vision Library) is a library of programming functions mainly aimed at real time computer vision, developed by Intel and now supported by Willow Garage. The library is OS/hardware/window manager independent.

The library is written in C and C++ and runs under Linux, Windows and Mac OS X. There is active development on interfaces for Python, Ruby, Matlab, and other languages. OpenCV was designed for computational efficiency and with a strong focus on real time applications. Open CV is written in optimized C and can take advantage of multi core processors. If you desire further automatic optimization on Intel architectures you can buy Intel's Integrated Performance Primitives (IPP) libraries, which consist of low-level optimized routines in many different algorithmic areas. OpenCV automatically uses the appropriate IPP library at runtime if that library is installed.

Features of OpenCV

- Image data manipulation (allocation, release, copying, setting, conversion).
- Image and video I/O (file and camera based input, image/video file output).
- Matrix and vector manipulation and linear algebra routines (products, solvers, eigenvalues, SVD).
- Various dynamic data structures (lists, queues, sets, trees, graphs).
- Basic image processing (filtering, edge detection, corner detection, sampling and interpolation, color conversion, morphological operations, histograms, image pyramids).
- Motion analysis (optical flow, motion segmentation, tracking).
- Object recognition (Eigen-methods, HMM).
- Basic GUI (display image/video, keyboard and mouse handling, scroll-bars).
- Image labeling (line, conic, polygon, text drawing)

One of OpenCV's goals is to provide a simple-to-use computer vision infrastructure that helps people build fairly sophisticated vision applications quickly. The OpenCV library contains over 500 functions that span many areas in vision, including factory product inspection, medical imaging, security, user interface, camera calibration, stereovision, and robotics, because computer vision and machine learning often go hand-in hand.

OpenCV API's:

Various API's in OpenCV library that are useful in face detection and image processing are:

- core- provides the core functionalities of OpenCV and contains basic structures and operations.
- imgproc- provides basic image processing operations like contrast, brightness, blur etc.
- video- provides video analysis functionalities.
- calib3d- provides camera calibration and 3D construction. Although we got most of our images in a 2D format they do come from a 3D world. It allows us to find out from 2D image information about 3D.
- features2d-provides feature detection and operations in 2D.
- objdetect- provides object detection functions like face detection, edge detection.
- ml- machine learning library is a set of classes and functions for statistical classification, regression, and clustering of data.

1.3 Viola Jones Method:

The Viola–Jones object detection framework is the first object detection framework to provide competitive object detection rates in real-time proposed in 2001 by Paul Viola and Michael Jones. Although it can be trained to detect a variety of object classes, it was motivated primarily by the problem of face detection. This algorithm is implemented in OpenCV as cvHaarDetectObjects(). This approach to detecting objects in images combines four key concepts:

- The features that Viola and Jones used are based on Haar wavelets. Haar wavelets are single wavelength square waves (one high interval and one low interval). In two dimensions, a square wave is a pair of adjacent rectangles - one light and one dark.

The actual rectangle combinations used for visual object detection are not true Haar wavlets. Instead, they contain rectangle combinations better suited to visual recognition tasks. Because of that difference, these features are called Haar features, or Haarlike features, rather than Haar wavelets. The presence of a Haar feature is determined by subtracting the average dark-region pixel value from the average light-region pixel value. If the difference is above a threshold (set during learning), that feature is said to be present.

- To determine the presence or absence of hundreds of Haar features at every image location and at several scales efficiently, Viola and Jones used a technique called an Integral Image. In general, "integrating" means adding small units together. In this case, the small units are pixel values. The integral value for each pixel is the sum of all the pixels above it and to its left. Starting at the top left and traversing to the right and down, the entire image can be integrated with a few integer operations per pixel.

- To select the specific Haar features to use, and to set threshold levels, Viola and Jones use a machine-learning method called AdaBoost. AdaBoost combines many "weak" classifiers to create one "strong" classifier. "Weak" here means the classifier only gets the right answer a little more often than random guessing would. That's not very good. But if you had a whole lot of these weak classifiers, and each one "pushed" the final answer a little bit in the right direction, you'd have a strong, combined force for arriving at the correct solution. AdaBoost selects a set of weak classifiers to combine and assigns a weight to each. This weighted combination is the strong classifier.

- Viola and Jones combined a series of AdaBoost classifiers as a filter chain that's especially efficient for classifying image regions. Each filter is a separate AdaBoost classifier with a fairly small number of weak classifiers. The acceptance threshold at each level is set low enough to pass all, or nearly all, face examples in the training set. The filters at each level are trained to classify training images that passed all previous stages. (The training set is a large database of faces, maybe a thousand or so.) During use, if any one of these filters fails to pass an image region, that region is immediately

classified as "Not Hand" When a filter passes an image region, it goes to the next filter in the chain. Image regions that pass through all filters in the chain are classified as "Hand" Viola and Jones dubbed this filtering chain a cascade.

1.4 CNN (Convolutional Neural Network)

The reason why convolutional neural network (CNN) is applied to handle the face recognition problem is as Lawrence once stated, the problem is ill-posed. So that even if some models may work very well in some cases, they cannot generalize well to those 'unseen' images. Convolutional Neural Network (CNN) are able to incorporate some constraints and get some degree of shift and deformation invariance.

A CNN usually consists of many layers and each of them will contain one or planes. There are input layer, convolutional layer, subsampling/max-pooling layer, fully-connected/output layer, and usually followed with an MLP or other type of classifier to train the features extracted.

Convolutional layers are the most important part of CNN. As its name says, convolution is used in this layer so as to detect features of the image. Usually, a convolutional layer will contain multiple planes, which use different convolution kernels to detect multiple features.

In CNN, once a feature is detected in the convolutional layer, its exact position in the image will become not so important. So we will have a subsampling layer, which use max-pooling or averaging or other method to do local averaging and subsampling operation.
The unique connection strategy of CNN allows us to reduce the number of weights that need to be computed.

1.4.1 Technical Work of CNN

We implemented three different classifiers from scratch:

1. a baseline classifier with one convolutional layer,
2. a CNN with a fixed size of five convolutional layers, and
3. a deeper CNN with a paramaterizable number of convolutional layers, filter dimensions, and number of filters.

For each of these models, we tuned parameters including learning rate, regularization, and dropout. We also experimented with using batch normalization and fractional max-pooling .Finally, we implemented multiple classifiers using finetuning with variations on the number of layers retained, the number of layers backpropagated through, and the initial network used. We experimented with fine-tuning using two existing models:

1. VGG16, the model from Caffe's Model Zoo which was trained on ImageNet,
2. VGGFace, a network trained on a facial recognition data set.

Baseline classifier

We implemented a baseline softmax classifier using features from a single convolutional layer. The architecture was one convolutional layer, followed by one fully connected layer, and a final softmax layer. The initial baseline did not use any regularization, dropout, or batch normalization.

Three-layer CNN

We implemented a first-pass CNN with a fixed depth of three convolutional layers. The model was trained using the architecture outlined in Table 1 and was trained using the following characteristics.

- Parametrized dropout rate, learning rate, and l2 regularization
- Batch normalization (optional) after each layer
- Adam update rule
- Weight initialization for using ReLU nonlinearities as presented by He et al.
- 3x3 convolutional filters with stride 1 and zeropaddingto preserve spatial size
- 2x2 max pools with a stride of 2

Deeper CNN

The architecture of our deeper CNN is outlined. We use the same network characteristics as the three-layer CNN with additional network structure parameters. With parameterizable layer depths and filter sizes, this model has a greater possible capacity than the five-layer.

Advantages and Disadvantages of Neural Network

Neural networks offer a number of *advantages*, including requiring less formal statistical training, ability to implicitly detect complex nonlinear relationships between dependent and independent variables, ability to detect all possible interactions between predictor variables, and the availability of multiple training algorithms . Generally, neural network approaches achieve higher accuracy in detection as compare to other approaches because they are trained with large number of samples.

Disadvantages include its "black box" nature, greater computational burden, proneness to over fitting, and the empirical nature of model development.

1.5 Keras:

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.

Use Keras if you need a deep learning library that:

- Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).
- Supports both convolutional networks and recurrent networks, as well as combinations of the two.
- Runs seamlessly on CPU and GPU.

Guiding principles:

- User friendliness. Keras is an API designed for human beings, not machines. It puts user experience front and center. Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear and actionable feedback upon user error.
- Modularity. A model is understood as a sequence or a graph of standalone, fully-configurable modules that can be plugged together with as little restrictions as possible. In particular, neural layers, cost functions, optimizers, initialization

schemes, activation functions, regularization schemes are all standalone modules that you can combine to create new models.

- Easy extensibility. New modules are simple to add (as new classes and functions), and existing modules provide ample examples. To be able to easily create new modules allows for total expressiveness, making Keras suitable for advanced research.
- Work with Python. No separate models configuration files in a declarative format. Models are described in Python code, which is compact, easier to debug, and allows for ease of extensibility.

1.6 Numpy:

Numpy is a library for the Python programming language, adding support for large multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

Limitations

- Inserting or appending entries to an array is not as trivially possible as it is with Python's lists.
- The *np.pad(...)* routine to extend arrays actually creates new arrays of the desired shape and padding values, copies the given array into the new one and returns it.
- NumPy's *np.concatenate([a1,a2])* operation does not actually link the two arrays but returns a new one, filled with the entries from both given arrays in sequence.
- Reshaping the dimensionality of an array with *np.reshape(...)* is only possible as long as the number of elements in the array does not change.

CHAPTER 2

LITERATURE REVIEW

Year	Author	Title	Method	Challenges	Perfomance
2015	Sourav Bhowmick, Sushant Kumar & Anurag Kumar	Hand Gesture Recognition of English Alphabets Using ANN	Classification of the gesture with ANN, Gesture Recognition System – Glove based system and vision based system	A skin color based hand segmentation technique has been used exploiting a hybrid HSV + YCbCr color model	Isolated gesture – 92.5% Continuous gesture – 87.14%
2013	Zafar Ahmad Ansari & Gaurav Harit	Nearest Neighbour Classification of Indian Sign Language gesture using Kinect form	RGB camera, gloves & depth camera. SVM, Homogeneous Texture Descriptor (HTD)	Requirement of roubust scale, orientation and rotation invariant features. Requirement of fast segmentation fast removal of high image noise.	90.68%
2015	Sourav Bhowmick, Anjan Kumar	Continuous Hand Gesture Recognition for	Multi layer perception Artificial	Able to handle the problem of recognition of	95.76%

	Talukdar & Kandarpa Kumar Sarma	English Alphabets.	Neural Network (MLP-ANN)	similar gesture.	
2013	Adithya V, Vinod P.R, Usha Gopalkrishnan	Artificial neural network based method for Indian Sign Language Recognition	Image acquisition and preprocessing, Hand segmentation, Feature extraction & Classification	Gesture Recognition isolated as well as continuous sign gesture and recognize similar gesture	91.11%
2013	Joyeeta Singha, Karan Das	Recognition of Indian Sign Language in Live Video	Neural network based features and HMM was used for recognizing various hand gestures.	The input frame was converted to HSV color space. The binary linked object was being considered.	96.25%

Table 2.1 Comparison of various different methods for hand gesture recognition system

CHAPTER 3

SOFTWARE REQUIREMENT ANALYSIS

3.1 PyCharm:

PyCharm is an Integrated Development Environment (IDE) used in computer programming, specifically for the Python language. It is developed by the Czech company JetBrains. It provides code analysis, a graphical debugger, an integrated unit tester, integration with version control systems (VCSes), and supports web development with Django.

PyCharm is cross-platform, with Windows, macOS and Linux versions. The Community Edition is released under the Apache License, and there is also Professional Edition released under a proprietary license.

Features:

- Coding assistance and analysis, with code completion, syntax and error highlighting, linter integration, and quick fixes.
- Project and code navigation: specialized project views, file structure views and quick jumping between files, classes, methods and usages.
- Python refactoring: including rename, extract method, introduce variable, introduce constant, pull up, push down and others.
- Support for web frameworks: Django, web2py and Flask.
- Integrated Python debugger.
- Integrated unit testing, with line-by-line code coverage.
- Google App Engine Python development.

3.2 TensorFlow:

TensorFlow is an open-source software library for dataflow programming across a range of tasks. It is a symbolic math library, and is also used for machine learning

applications such as neural networks. It is used for both research and production at Google, often replacing its closed-source predecessor.

TensorFlow is Google Brain's second generation system. While the reference implementation runs on single devices, TensorFlow can run on multiple CPUs and GPUs (with optional CUDA and SYCL extensions for general-purpose computing on graphics processing units). TensorFlow is available on 64-bit Linux, macOS, Windows, and mobile computing platform including Android and iOS.

TensorFlow computations are expressed as stateful dataflow graphs. The name TensorFlow derives from the operations that such neural networks perform on multidimensional data arrays. These arrays are referred to as "tensors".

Tensor processing unit (TPU):

In May 2016 Google announced its Tensor processing unit (TPU), an ASIC built specifically for machine learining and tailored for TensorFlow. TPU is a programmable AI accelerator designed to provide high throughput of low-precision arithmetic (e.g., 8-bit), and oriented towards using or running models rather than training them. Google announced they had been running TPUs inside their data centers for more than a year, and had found them to deliver an order of magnitude better-optimized performance per watt for machine learning.

In May 2017 Google announced the second-generation, as well as the availability of the TPUs in Google Compute Engine. The second-generation TPUs deliver up to 180 teraflops of performance, and when organized into clusters of 64 TPUs, provide up to 11.5 petaflows.

3.3 System Requirement:

Operating System	Windows 10
CPU	Intel i7 core processor
GPU	Nvidia 1050Ti Graphics
Memory	8 GB DDR3 RAM
Hard Drive	500 GB

Table 3.1 System Specification

CHAPTER 4

SYSTEM DESIGN

4.1 System Architecture:

The system proposed is to recognize hand gesture. The technique that is used to recognize hand gesture is computer vision. The overall system architecture is shows in figure 4.1. The whole system of hand gesture recognition divided into four phases: Image Acquisition, Image Pre-processing, Feature Extraction and Hand Gesture Recognition.

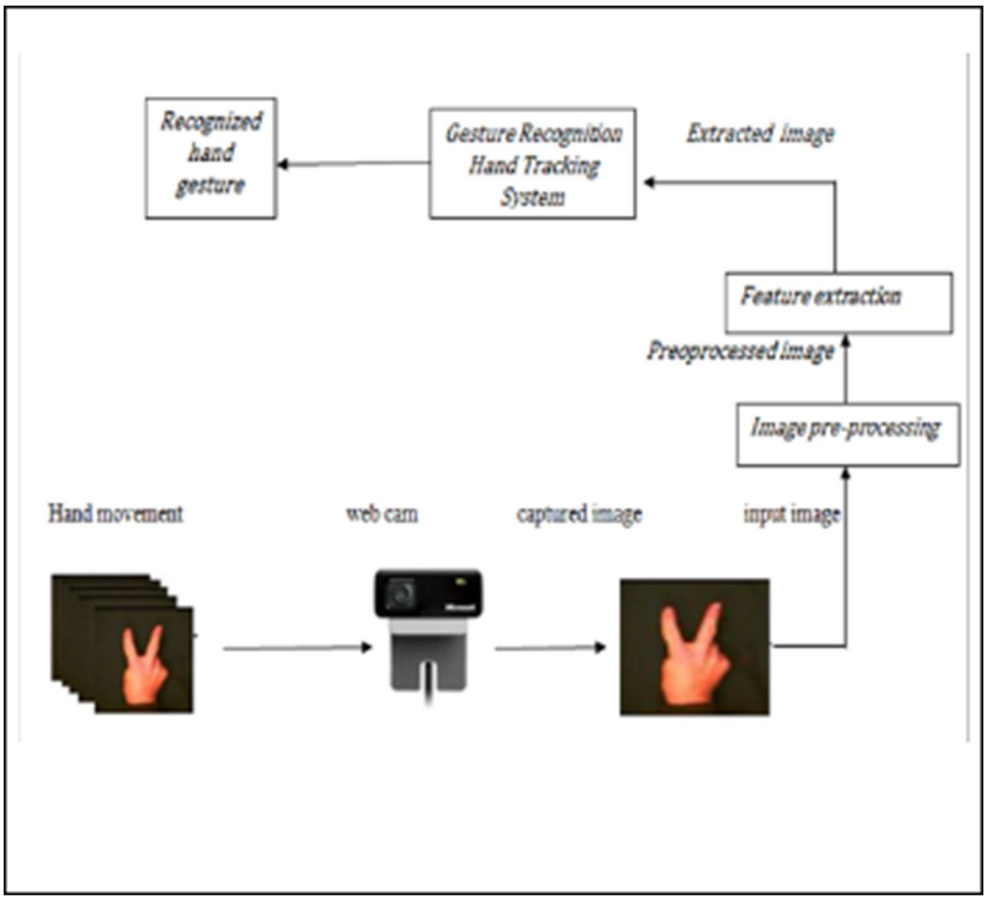

Figure 4.1 System Architecture

Reading the image, Frame extraction and Pre-processing comes under image acquisition module. The initiation of the acquisition is first done manually. A camera sensor is needed in order to capture the features/web cam. Local changes such as noise and digitised errors should not change the image scenes and information's. In order to satisfy the memory requirements and the environmental scene conditions, pre- processing of the image is highly important. The first most step of pre-processing block is filtering. It is used to remove the unwanted noise from the image.
Primarily, glove-based analysis where either mechanical or optical sensors are attached to a glove that transforms finger flexions into electrical signals to determine the hand posture. Currently, the vision-based analysis is used mostly, which deals with the way human beings perceive information about their surroundings. The data- base for these vision-based systems is created by selecting the gestures with predefined meaning, and multiple samples of each gesture are considered to increase the accuracy of the system. In this paper, we have used the vision-based approach for our gesture recognition application. Several approaches have been proposed previously to recognize the gestures using soft computing approaches such as artificial neural networks (ANNs), and genetic algorithms. ANNs are the adaptive self-organizing technologies that solved a broad range of problems such as identification and control, game playing and decision making, pattern recognition medi- cal diagnosis, financial applications, and data mining in an easy and convenient manner.
Hand Gesture is habitually used in everyday life style. It is so natural way to communicate. Hand gesture recognition method is used in the application area such as for Controlling mouse and/or keyboard functionality, 3D World, Hu- man/Robot Manipulation and Instruction Communicate at a distance.

4.2 Data Flow within system:

Figure 4.2 shows the gesture recognition hand tracking system. This system consists of three stages: background removal using GMM, skin feature extraction using HSV colour space, hand tracking using Euclidistance and finally gesture recognition. In the first stage input image of hand gestures are acquiesced by digital camera in approxi- mate frame rate. In second stage a rotation, translation, scaling and orientation invariant feature extraction

method has been introduce to extract the feature of the input image based on moment feature extraction method. Finally, a neural network is used to recognize the hand gestures.

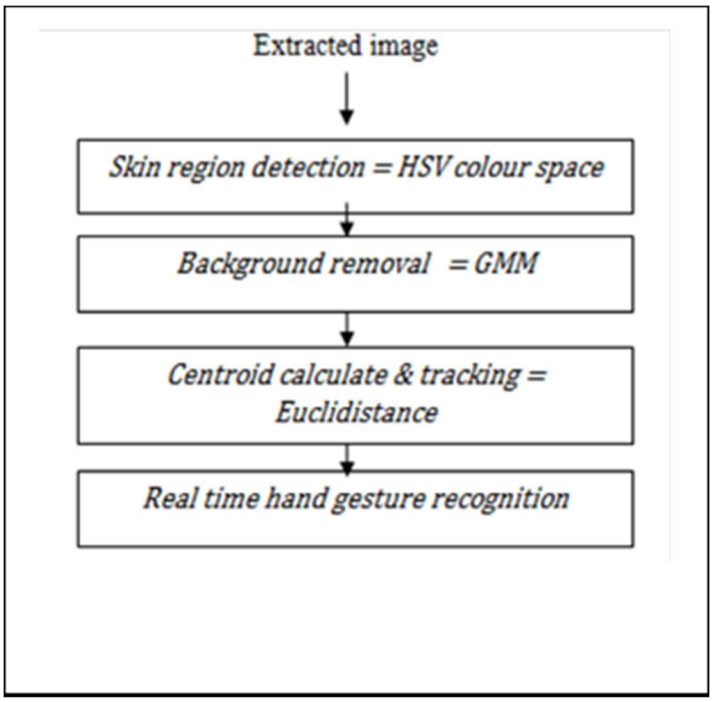

Figure 4.2 Gesture Recognition Hand Tracking System

CHAPTER 5

IMPLEMENTATION

The complete Hand Gesture Recognition system can be divided into the following process:

- Hand Detection
- Image acquisition
- Image preprocessing
- Feature extraction
- Hand Gesture Recognition

5.1 Hand Detection:

Hand detection is a computer technology that determines the locations and sizes of human hand in arbitrary (digital) images. It detects hand features and ignores anything else, such as buildings, trees and bodies.

Early hand-detection algorithms focused on the detection of frontal human hand, whereas newer algorithms attempt to solve the more general and difficult problem of multi-view hand detection. That is, the detection of hand that are either rotated along the axis from the hand to the observer (in-plane rotation), or rotated along the vertical or left-right axis (out-of-plane rotation), or both. The newer algorithms take into account variations in the image or video by factors such as face appearance, lighting, and pose.

```
import cv2
import numpy as np

hand_cascade = cv2.CascadeClassifier('hand.xml')
# eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')
cam = cv2.VideoCapture(0)

while True:
    ret,img = cam.read()
    gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    hands = hand_cascade.detectMultiScale(gray_img, 1.3, 5)
    for(x,y,w,h) in hands:
        cv2.rectangle(img, (x,y), (x+w, y+h),(255,0,0),2)
        roig= gray_img[y:y+h, x:x+w]
        roic= img[y:y+h, x:x+w]
        # eyes=eye_cascade.detectMultiScale(roig)
        # for(a,b,c,d) in eyes:
        #     cv2.rectangle(roic, (a,b), (a+c,b+d),(0,255,0),2)

    cv2.imshow('fed',img)
    k=cv2.waitKey(30) & 0xff
    if k == 27:
        break

cam.release()
cv2.destroyAllWindows()

```

Figure 5.1 Hand Detection

It is very easy to use a webcam stream as input to the hand gesture recognition system instead of a file list. Basically it just needs to input frames from a camera instead of image from a file. And run forever until the user wants to quit, instead of just running until the file list is run out. OpenCv provides the cvCreateCameraCapture() also known as cvCaptureFromCAM() for this.

5.2 Image Acquisition:

Once a face has been detected in an image we need to capture the face region so as to do further processing of image for recognition purposes. This is setting of region of interest. A Region of Interest (ROI) is a portion of an image that we want to filter or perform some other operation on. We take the image consisting of the face as input image and set a rectangle ROI on the face area.Function used for this in OpenCv is:

Parameters:

- **image** –A pointer to the image header
- **rect** –The ROI rectangle

This ROI rectangle is an image which needs to be saved to another image which has the same properties as the input image. This is done using cvSave().

```
import cv2
import numpy as np
# import matplotlib.pyplot as plt
img =cv2.imread('mask.jpg',0)
cv2.imwrite('mask2.jpg',img)
# cam = cv2.VideoCapture(0)
#
#
# _, img = cam.read()
img= cv2.imread('116.jpg')
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

lower = np.array([0,48,80],dtype='uint8')
upper = np. array([20,255,255], dtype='uint8')

mask = cv2.inRange(hsv, lower, upper)

kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(11,11))
mask= cv2.erode( mask, kernel,iterations=2)
mask=cv2.dilate(mask, kernel,iterations=2)

mask = cv2.GaussianBlur(mask, (3,3), 0)

op= cv2.bitwise_and(img, img, mask=mask)

_, contour ,_ = cv2.findContours(mask,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
cv2.drawContours(op, contour, -1, (0,255,0), thickness=3)

```

Figure 5.2 Image Acquisition

5.3 Image Preprocessing:

In this module, by means of early vision techniques, hand images are normalized and if desired, they are enhanced to improve the recognition performance of the system. Image preprocessing is a significant step before applying any other technique on images. Image preprocessing can be of different kinds, for noise removal, smoothing thresholding, background removal etc., it can be rightly said that the kind of preprocessing required is greatly dependent on the application under consideration. The data that has been acquired may not be suitable for applying further processing. It consists of many noisy and redundant data. So this data needs to be preprocessed. Preprocessing is done to improve the efficiency, accuracy and robustness of the proposed system. It also reduces the complexity of the system by removing redundant data.Some or all of the following pre-processing steps may be implemented in a face recognition system:

```
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(11,11))
mask= cv2.erode( mask, kernel,iterations=2)
mask=cv2.dilate(mask, kernel,iterations=2)

mask = cv2.GaussianBlur(mask, (3,3), 0)

op= cv2.bitwise_and(img, img, mask=mask)

_, contour ,_ = cv2.findContours(mask,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
cv2.drawContours(op, contour, -1, (0,255,0), thickness=3)


# cv2.imwrite('mask.png',mask)
# cv2.imwrite('op.png',op)

cv2.imwrite('mask.jpg',mask)


while True:
    cv2.imshow('Original Image',img)
    cv2.imshow("HSV", hsv)
    cv2.imshow("mask", mask)
    cv2.imshow("Hand", op)
```

Figure 5.3 Image Preprocessing

5.3.1 Grayscale conversion:

In many applications involving hand gesture recognition, the inputs are first binarized to form a two level image based on the threshold value. It is usual to preprocess the scanned images to do the image enhancement in the presence of noise and other types of distortions that occur during the scanning process. In this preprocessing stage, the image is enhanced, resized and binarized to make the image clear and accurate. It is necessary to employ the non linear technique for processing the face images prior to binarization.

There are several methods available for thresholding image to produce binary image. An experimental performance evaluation of several such techniques may found in. These methods include fixed global threshold, Otsu threshold and other techniques. The input grayscale pixels are denoted by *X*i € {0, 1}. The corresponding output binarization pixels are denoted by *bi* € {0, 1} where 0 refers to „black' and 1 refers to 'white'.

Image enhancement is necessary to remove noise by filtering the image, adjust the contrast of the image and enhance the edge of the face image before binarization process and training the facial recognition. Edge enhancement can be achieved by noise presents in the input images but tend to smooth the edge details.

In this project, the global image threshold using Otsu's method will be chosen as the technique to binarize the face images. This method finds the global threshold *t* that minimizes the infraclass variance of the resulting black and white pixels of the image.

The threshold of this method can be shown as:

$$bi = \begin{cases} 1 \text{ if } x_i \geq t \\ 0 \text{ if } x_i < t \end{cases}$$

Where,

x_i = input grayscale pixel

b_i = output binarization pixel

OpenCv function: cvtColor(img, g_img, CV_BGR2GRAY);

Parameters:

- **src** –Source image: 8-bit unsigned, 16-bit unsigned (CV_16UC...), or single-precisionfloating-point.
- **dst** –Destination image of the same size and depth as src .

- **code** –Color space conversion code. BGR2GRAY for grayscale coversion.
- **dstCn** –Number of channels in the destination image. If the parameter is 0, the numberof the channels is derived automatically from src and code .

5.3.2 Resizing:

It is usually done to change the acquired image size to a default image size such as 128 x 128, on which the face recognition system operates. This is mostly encountered in systems where face images are treated as a whole like the one proposed in this thesis.
The function used for resizing the image is,
resize(g_img,tmpsize,size,CV_INTER_LIAR)

Parameters:

- **src** –Source image.
- **dst** –Destination image. It has the size dsize (when it is non-zero) or the size computedfrom src.size() , fx , and fy . The type of dst is the same as of src .
- **interpolation** –
  Interpolation method:
- **INTER_NEAREST** - a nearest-neighbor interpolation
- **INTER_LINEAR** - a bilinear interpolation (used by default)
- **INTER_AREA** - resampling using pixel area relation. It may be a preferred method forimage decimation, as it gives moire'-free results. But when the image is zoomed, it is similar to the INTER_NEAREST method.
- **INTER_CUBIC** - a bicubic interpolation over 4x4 pixel neighborhood
- **INTER_LANCZOS4** - a Lanczos interpolation over 8x8 pixel neighborhood

5.3.3 Histogram equalization:

It is usually done on too dark or too bright images in order to enhance image quality and to improve face recognition performance. It modifies the dynamic range (contrast range) of the image and as a result, some important facial features become more apparent.

5.3.4 Median filtering:

For noisy images especially obtained from a camera or from a frame grabber, median filtering can clean the image without losing information.

5.3.5 High-pass filtering:

Feature extractors that are based on facial outlines, may benefit the results that are obtained from an edge detection scheme. High-pass filtering emphasizes the details of an image such as contours which can dramatically improve edge detection performance.

5.3.6 Background removal:

In order to deal primarily with facial information itself, face background can be removed. This is especially important for face recognition systems where entire information contained in the image is used. It is obvious that, for background removal, the preprocessing module should be capable of determining the face outline.

5.3.7 Illumination Normalization:

Face images taken under different illuminations can degrade recognition performance especially for face recognition systems based on the principal component analysis in which entire face information is used for recognition. A picture can be equivalently viewed as an array of reflectivity's r(x). Thus, under a uniform illumination I, the corresponding picture is given by:

The normalization comes in imposing a fixed level of illumination I_0 at a reference point x_0 on a picture. In actual practice, the average of two reference points, such as one under each eye, each consisting of 2 x 2 arrays of pixels can be used.

5.4Feature Extraction:

Feature Extraction stage is necessary because certain features has to be extracted so that they are unique for each gesture or sign. After the decision is made that a sign is present, then the last frame is taken into consideration and features like Eigen values and Eigen vectors are extracted from that frame. The procedureto calculate the features i.e. Eigen value and Eigen vector are given in as follows:

```
import tensorflow as tf
from keras.backend.tensorflow_backend import set_session
config = tf.ConfigProto()
config.gpu_options.allow_growth = True
config.gpu_options.per_process_gpu_memory_fraction = 0.3
set_session(tf.Session(config=config))

from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout

classifier = Sequential()

classifier.add(Conv2D(32, (3, 3), input_shape=(48, 48, 3), activation='relu'))
classifier.add(MaxPooling2D(pool_size=(2, 2)))
classifier.add(Dropout(.20))

classifier.add(Conv2D(16, (3, 3), activation='relu'))
classifier.add(MaxPooling2D(pool_size=(2, 2)))
classifier.add(Dropout(.20))

classifier.add(Conv2D(8, (3, 3), activation='relu'))
classifier.add(MaxPooling2D(pool_size=(2, 2)))
classifier.add(Dropout(.20))

classifier.add(Flatten())
classifier.add(Dense(256, activation='relu'))
classifier.add(Dropout(.5))
classifier.add(Dense(1, activation='sigmoid'))

classifier.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Figure 5.4 Feature Extraction

Step 1:Frame resizing- Let us assume the last frame is ‘X’. ‘X’ is resized to 70 by 70.

Step 2: Mean and Covariance calculation- Mean ‘M’ and Covariance ‘C’ is calculated.

Step 3:The Eigen values and Eigen vectors are calculated from the above covariance ‘C’ and the Eigen vectors are arranged in such a manner that the Eigen values are in descending order.

Step 4: Data Compression- Out of 70 Eigen vectors only first 5 principle vectors were considered, thus reducing the dimension of the matrix.

5.5 Usage of Neural Networks for Recognition:

Neural networks are composed of simple elements operating in parallel. The neuron model is the one that widely used in artificial neural networks with some minor modifications on it.

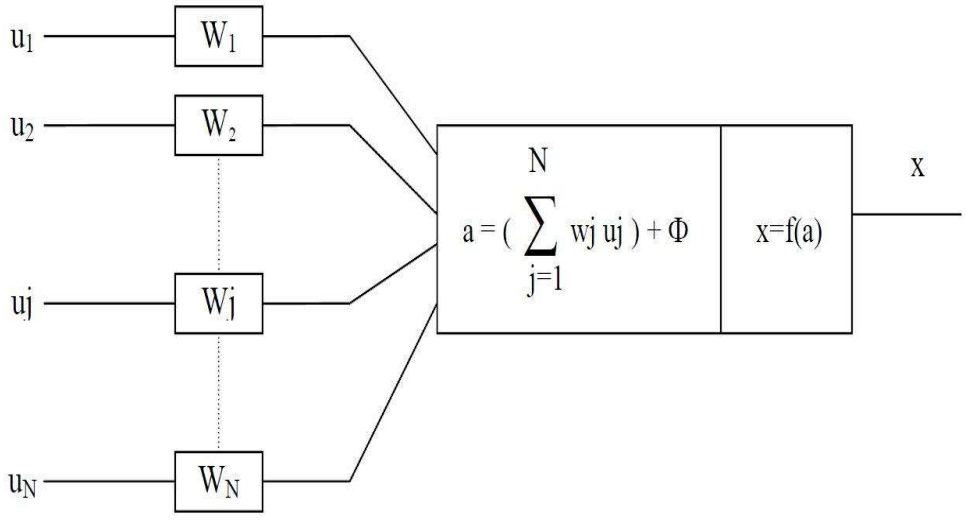

Figure 5.5 Artificial neuron

The artificial neuron given in this figure has N input, denoted as u_1, u_2,…u_N. Each line connecting these inputs to the neuron is assigned a weight, which is denoted as w_1,w_2,….,w_N respectively. The threshold in artificial neuron is usually represented by Φ and the activation is given by the formula

The inputs and weight are real values. A negative value for a weight indicates an inhibitory connection while a positive value indicating excitatory one. If Φ is positive, it is usually referred as bias. For its mathematical convenience (+) sign is used in the activation formula. Sometimes, the threshold is combined for simplicity into the summation part by assuming an imaginary input $u_0 = +1$ and a connection weight $w0 = \Phi$. Hence the activation formula becomes

$$a = (\sum_{j=1}^{n} w_j u_j)$$

The vector notation

$$a = (\sum_{j=1}^{n} w_j u_j) + \Phi$$

$$A = w^T u + \Phi$$

is useful for expressing the activation of neuron.

The neural output function f(a) can be Linear

Threshold:

$$f(a) = \begin{cases} 0 & a \le 0 \\ 1 & 0 < a \end{cases}$$

Ramp:

$$f(a) = \begin{cases} 0 & a \le 0 \\ a / K & 0 < a \\ 1 & K < a \end{cases}$$

Sigmoid:

$$f(a) = 1/(1 + \exp(-Ka))$$

Function implementations can be done by adjusting the weights and the threshold of the neuron. Furthermore, by connecting the outputs of some neurons as inputs to the others, neural network will be established, and any function can be implemented by these networks. The last layer of neurons is called the output layer and the layers between the input and output layer are called the hidden layers. The input layer is made up of special input neurons, transmitting only the applied external input to their outputs. In a network, if there is only the layer of input nodes and a single layer of neurons constituting the output layer then they are called single layer network. If there are one or more hidden layers, such networks are called multilayer networks. There are two types of network architecture: recurrent and feed forward neural network.

5.6 Convolutional Network

The problem of hand gesture from 2D images is typically very ill-posed, i.e. there are many models which fit the training points well but do not generalize well to unseen images. In other words, there are not enough training points in the space created by the input images in order to allow accurate estimation of class probabilities throughout the input space. Additionally, for MLP networks with the 2D images as input, there is no invariance to translation or local deformation of the images .

Convolutional networks (CN) incorporate constraints and achieve some degree of shift and deformation invariance using three ideas: local receptive fields, shared weights, and spatial subsampling. The use of shared weights also reduces the number of parameters in the system aiding generalization. Convolutional networks have been successfully applied to character recognition [24, 22, 23, 5, 3].

A typical convolutional network . The network consists of a set of layers each of which contains one or more planes. Approximately centered and normalized images enter at the input layer. Each unit in a plane receives input from a small neighborhood in the planes of the previous layer. The idea of connecting units to local receptive fields dates back to the 1960s with the perceptron and Hubel andWiesel's discovery of locally sensitive, orientation-selective neurons in the cat's visual system.

The weights forming the receptive field for a plane are forced to be equal at all points in the plane. Each plane can be considered as a feature map which has a fixed feature detector that is convolved with a local window which is scanned over the planes in the previous layer. Multiple planes are usually used in each layer so that multiple features can be detected. These layers are called convolutional layers. Once a feature has been detected, its exact location is less important.

Hence, the convolutional layers are typically followedby another layer which does a local averaging and subsampling operation (e.g. for a subsampling factor of2: yij = (x2i;2j + x2i+1;2j + x2i;2j+1 + x2i+1;2j+1) =4 where yij is the output of a subsampling plane at position i; j and xij is the output of the same plane in the previous layer). The network is trained with the usual backpropagation gradient-descent procedure . A connection strategy can be used to reduce the number of weights in the network. For example, with reference to figure 5, Le Cun et al. connect thefeature maps in the second convolutional layer only to 1 or 2 of the maps in the first subsampling layer (the connection strategy was chosen manually).

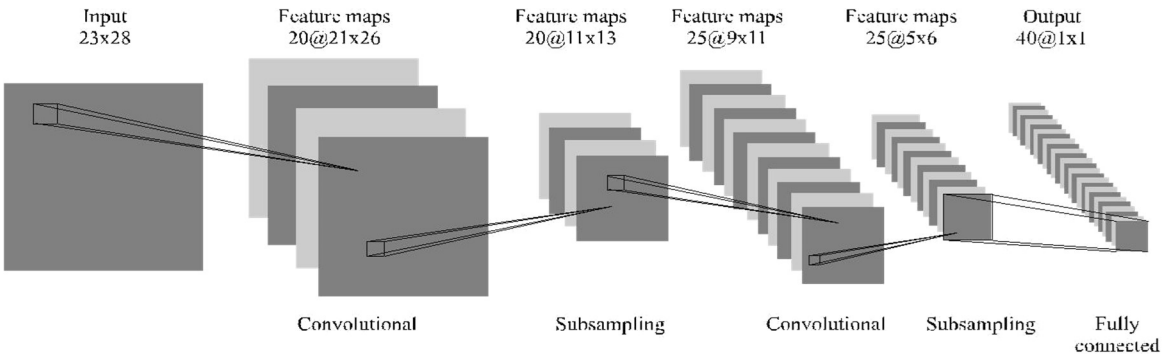

Figure 5.6 A typical convolutional network.

```
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(
    rescale=1. / 255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1. / 255)

train_set = train_datagen.flow_from_directory(
    'dataset/training',
    target_size=(48, 48),
    batch_size=32,
    class_mode='binary')

test_set = test_datagen.flow_from_directory(
    'dataset/testing',
    target_size=(48, 48),
    batch_size=32,
    class_mode='binary')

classifier.fit_generator(
    train_set,
    steps_per_epoch=8575,
    epochs=15,
    validation_data=test_set,
    validation_steps=2000)
```

Figure 5.7 A neural network

CHAPTER 6

TESTING

6.1 Simulation Result:

The template image for the class of alphabet V is as shown. First of all the depth data for the template is loaded, followed by hand segmentation. Then, hand contour is found and maximum inscribed circle is plot for the segmented hand. Lastly, the time-series curve is plot for the template image.

Gesture 1: V

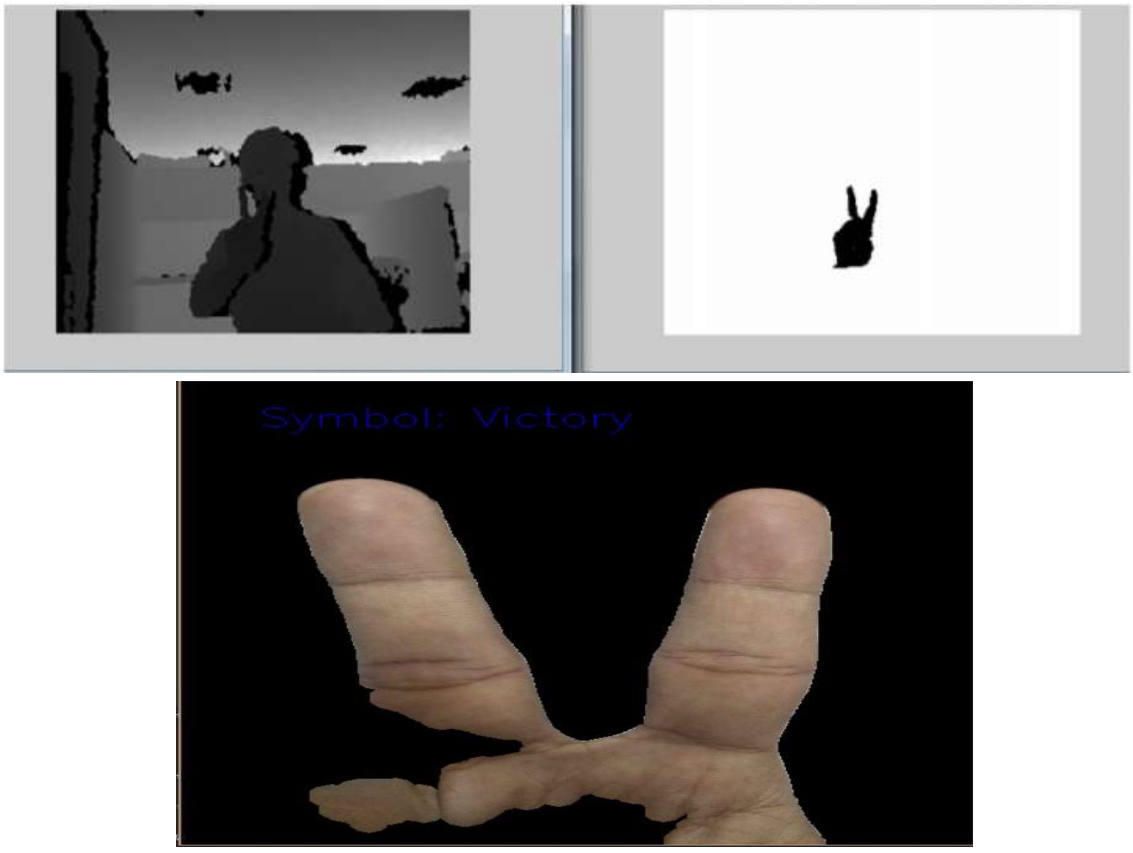

Figure 6.1 Clockwise from top-left: Depth image, segmented binary image, Hand contour

The template image for the class of alphabet 1 is as shown. First of all the depth data for the template is loaded, followed by hand segmentation. Then, hand contour is found and maximum inscribed circle is plot for the segmented hand. Lastly, the time-series curve is plot for the template image.

Gesture 2: 1

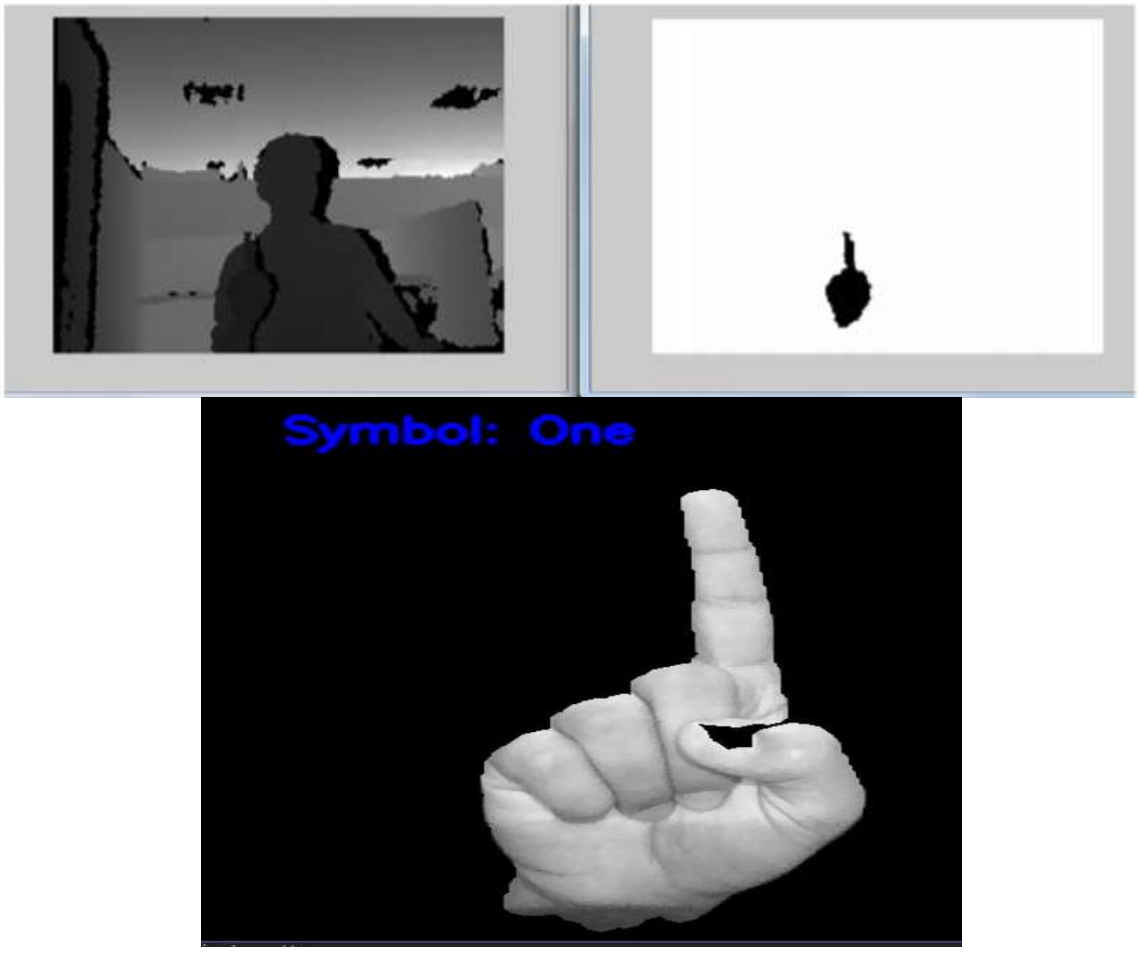

Figure 6.2 Clockwise from top-left: Depth image, segmented binary image, Hand contour

The image for the input hand gesture is as shown. First of all the depth data for the input gesture is loaded, followed by hand segmentation. Then, hand contour is found and maximum inscribed circle is plot for the segmented hand. Lastly, the time-series curve is plot for the input hand gesture image.

Input Hand

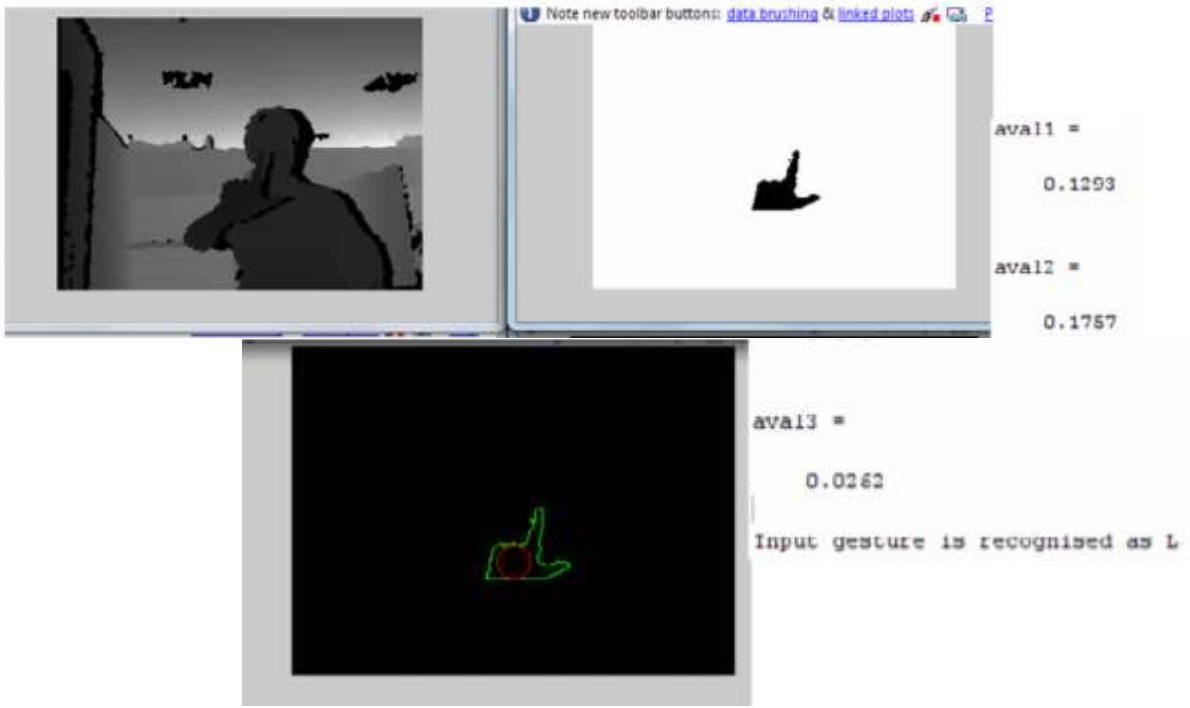

Figure 6.3 Clockwise from top-left: Depth image, segmented binary image, Hand contour

6.2 Discussion:

Once the depth data for the templates and the input hand gesture have been loaded and the time-series curve is plot for each signature, the next step is to recognise the class of the input hand gesture. The time-series curve for the input hand signature as well as template signatures is formed and is used for finger detection is also performed in each of the representations (this is nothing but feature extraction). Then, the Finger-Earth Mover's distance is found between the input hand gesture and each of the template signatures. The class having minimum FEMD is chosen as the recognised class for the given input hand gesture. For classification purpose, first the number of fingers in the input hand gesture is found. If the number of fingers is zero, then without any FEMD calculation, the input gesture is recognised as belonging to the class of alphabet S. Otherwise, if the number of fingers is one, then the input gesture is recognised as belonging to the class of alphabet I, without calculation of FEMD from the templates. Otherwise, if the number of fingers is two, the FEMD calculation is done for the input feature vectors, from each of the templates. Thereafter, the class with minimum FEMD is chosen as the recognised gesture. The result for one such input gesture is shown. The number of fingers was found to be two and so, the FEMD from templates of classes V is found as 0.1293.

CHAPTER 7

CONCLUSION AND FUTURE ENHANCEMENT

7.1 Conclusion:

- Hand gesture recognition system was implemented for two gesture classes (Y, I).
- The system was tested for three different inputs per class.
- Out of 6 testing images, 6 were recognized correctly.
- Thus, an accuracy of 100% was acheived.

7.2 Future Work:

The project involves distinguishing among two different alphabets of English language. Future work may include recognition of all the English alphabets. Further, we may move on to recognising words, from as large a dictionary as possible, from Indian Sign Language. Another method to improve the performance is by using a more accurate method for finger detection from the time-series curve like near-convex decomposition.

REFERENCE

[1] V Adithya, PR Vinod, and Usha Gopalakrishnan. Artificial neural network based method for indian sign language recognition. In *Information & Communication Technologies (ICT), 2013 IEEE Conference on*, pages 1080–1085. IEEE, 2013.

[2] Zafar Ahmed Ansari and Gaurav Harit. Nearest neighbour classification of indian sign language gestures using kinect camera. *Sadhana*, 41(2):161–182, 2016.

[3] Sourav Bhowmick, Sushant Kumar, and Anurag Kumar. Hand gesture recognition of english alphabets using artificial neural network. In *Recent Trends in Information Systems (ReTIS), 2015 IEEE 2nd International Conference on*, pages 405–410. IEEE, 2015.

[4] Sourav Bhowmick, Anjan Kumar Talukdar, and Kandarpa Kumar Sarma. Continuous hand gesture recognition for english alphabets. In *Signal Processing and Integrated Networks (SPIN), 2015 2nd International Conference on*, pages 443–446. IEEE, 2015.

[5] Varun Tiwari, Vijay Anand, Avinash G Keskar, and Vishal R Satpute. Sign language recognition through kinect based depth images and neural network. In *Advances in Computing, Communications and Informatics (ICACCI), 2015 International Conference on*, pages 194–198. IEEE, 2015.