SIGN LANGUAGE RECOGNITION

Thesis submitted to Visvesvaraya National Institute of Technology, Nagpur in partial fulfilment of requirement for the award of degree of

BACHELOR OF TECHNOLOGY

(COMPUTER SCIENCE AND ENGINEERING)

Submitted By

Disha Nathani (BT11CSE033)

Jyoti Ingle (BT11CSE045)

Rani Meshram (BT11CSE057)

Rasika Mohod (BT11CSE058)

Under the guidance of

Ms. Mansi A. Radke

Department of Computer Science and Engineering

Visvesvaraya National Institute of Technology

Nagpur 440 010 (India)

April 2015

CERTIFICATE

This is to certify that the project work entitled — "**Sign Language Recognition**", is a bonafide work done by **Disha Nathani, Jyoti Ingle, Rasika Mohod, Rani Meshram** In the Department of Computer Science and Engineering, Visvesvaraya National Institute of Technology, Nagpur, in partial fulfilment of the requirements for the award of **Bachelor Of Technology** in **"Computer Science and Engineering"**

Project Guide:

Ms. Mansi A. Radke
Associate Professor,
Department of
Computer Science and Engineering,
VNIT, Nagpur

Head of Department:

Dr.P.S.Deshpande
Professor and Head,
Department of
Computer Science and Engineering,
VNIT, Nagpur

Department of Computer Science and Engineering

Visvesvaraya National Institute of Technology

2014-2015

DECLARATION

We, hereby declare that the thesis titled — **Sign Language Recognition** submitted herein has been carried out by us in the Department of Computer Science and Engineering of Visvesvaraya National Institute of Technology, Nagpur. The work is original and has not been submitted earlier as a whole or in part for the award of any degree / diploma at this or any other Institution / University.

Disha Nathani	**Jyoti Ingle**	**Rani Meshram**	**Rasika Mohod**
(BT11CSE033)	**(BT11CSE045)**	**(BT11CSE057)**	**(BT11CSE058)**

Date: 15.04.2015

ACKNOWLEDGEMENT

We take this opportunity to express gratitude to our project guide Ms. Mansi A. Radke, Assistant Professor, Department of Computer Science and Engineering, VNIT Nagpur for her valuable guidance and suggestion in our work without which the fulfilment of project would have been impossible. We are grateful to her for her constant encouragement and guidance. It was through her relentless efforts that we have been able to fulfil our project requirement. She has been a source of inspiration and we thank her for her timely guidance in the conduct of our project work. We sincerely thank her for providing each and every facility required for the successful completion of the project.

We would also like to thank all the teaching and non-teaching staff of the Computer Science and Engineering Department for supporting us and providing necessary facilities at all the times.

Disha Nathani,

Jyoti Ingle,

Rani Meshram,

Rasika Mohod

ABSTRACT

Sign language recognition software aims at real time detection of various gestures performed by user, processing them and producing output in form of speech. These gestures can be basic 2 dimensional or 3 dimensional. For processing these differences, project is divided into two phases.

First phase aims at recognising 2 dimensional gestures. Continuous video input processing is to be done and 2 dimensional gestures are recognised and returned in form of speech.
Second phase aims for recognising 3 dimensional along with previous 2 dimensional gestures. It is to be developed along similar lines but with consideration of depth information for 3 dimensional processing.

TABLE OF CONTENTS

1. INTRODUCTION

INTRODUCTION

There is always a need to communicate using sign languages, such as communicating with speech and hearing challenged people. Additionally, there are situations when silent communication is preferred; for example, during an operation, a surgeon may gesture to the nurse for assistance. It is hard for most people who are not familiar with a sign language to communicate without an interpreter. Thus, software that transcribes symbols in sign languages into plain text and speech can help with real-time communication, and it may also provide interactive training for people to learn a sign language. Gesture recognition has become an important research field with the current focus on hand gesture recognition.

The project is performed in two phases. In phase 1, we made a Linux-based application for live motion gesture recognition using webcam input in C++. This project is a combination of live motion detection and gesture identification. This application uses the webcam to detect gesture made by the user and gives output accordingly in form of speech. The user has to perform a particular gesture. The webcam captures this and identifies the gesture, recognises it (against a set of known gestures) and gives result corresponding to it. This application can be made to run in the background while the user runs other programs and applications.

The project essentially consists of four parts:

1. Taking input from the webcam and converting it into a form that can be processed easily.
2. Intercepting the gesture from the input of the webcam.
3. Recognising the gesture from a database of gestures.
4. According to the intercepted gesture, give result in speech form.

Difference between subsequent frames helps detect motion. We included further modifications to eliminate noise so that only the moving object (hand or finger) may be interpreted. The interpreted gesture is scanned against a set of known gesture to find which gesture matches

the best. The output in speech form is then produced. Few basic gestures are incorporated, while the user is given the choice of adding other gestures.

In phase 2, we wish to build an application which can recognise 3D gestures along with 2D gestures using Microsoft Kinect XBOX 360. It is developed on similar lines as in phase 1 but for accuracy and faster results some changes have been made. It is developed in C# of Visual Studio 2013.

This phase consists three parts:

1. Taking input from the Kinect.
2. Intercepting the gesture from the input using predefined modules for Kinect in Visual Studio.
3. According to the intercepted gesture, give result in speech form.

The accuracy and the robustness make this system a versatile component that can be integrated in a variety of applications in daily life.

SCOPE

- Live motion detection and sign identification
- Detection of any kind of sign
- Used for educational, organisational or individual purposes
- Important tool in social development

Further, we wish to incorporate machine learning. The software should be able to recognise new gestures on its own and add them to the system. It can also be provided as a Mobile application to users on their smartphones. Thus, no special setup would have to be done and it can be used anytime, at any place, by anyone.

2. BACKGROUND

MOTIVATION

As human beings, the basic feature which distinguishes us from the normal animals, is speech. We can say what we think or feel. This helps us in expressing clearly. The exchange of ideas in this manner makes discussing and working on them possible. This process is an integral part of development in every walk of life. It also helps to make world a better place to live.

But some unfortunate beings among us, don't have this ability. This makes them unable to express freely. It might happen that their views, ideas, expressions or feelings go unattended. Whereas, they can be of equal significance in certain cases. They still have the ability to think. So their ideas can also be equally interesting and worth considering. Who knows, even they could make a difference!

Such kind of people are usually not considered to be as much privileged; normal. They are looked down at. This creates a division between humans and they have to live differently.

As each human has the right to express, speech and respect; there should not be any different case for these deaf and dumb people. They should have equal right for the same.
To empower them with the rights which are unknowingly being taken away from them, we aim to develop this application.

The basic motivation comes from our wish to make a difference and we hope this application would help us achieve our aim.

APPLICATION and TARGET AUDIENCE

This software can have applications in any type of field where human interaction is an integral part. This would enable the deaf and dumb people to communicate with anybody, even the ones who are not well versed with the sign language. It can be used in educational institutions for teaching purposes. Separate organizations would not have to be made for the deaf and dumb. It can also be used in industries, enterprises or similar kind so that they can also be employed with the others. It can mainly help in development of all humans equally.

This aims to target mainly the underprivileged people, to help them communicate with everyone. Also, a handy tool for the people to understand sign language who don’t know it.

SIGN LANGUAGE

One of the most precious gifts of nature to the human breed is the ability to express oneself by responding to the events occurring in one's surroundings. Every physically normal human being sees, listens and then reacts to the situations by speaking himself out. But there are some less fortunate ones who are deprived of this valuable gift. Such people, mainly the deaf and the dumb, rely on some sort of sign language for communicating their feelings to others. Gestures are considered as the most natural expressive way for communications between these people and others.

Hand gesture is a method of non-verbal communication for human beings for its freer expressions much more other than body parts.

Sign language consists of different types of gestures. These may be hand movements, alphabetical representations, gestures including facial as well as bodily expressions, etc.

These can also be classified as 2D and 3D gestures according to various aspects of the gesture. For example, a simple horizontal hand movement can require only 2D processing, whereas an alphabetical representation is 3D since it involves depth processing also finger state recognition.

'Figure 1' shows a 2D Hello gesture and 'Figure 2' shows some alphabetical representations in 3D.

Figure 1

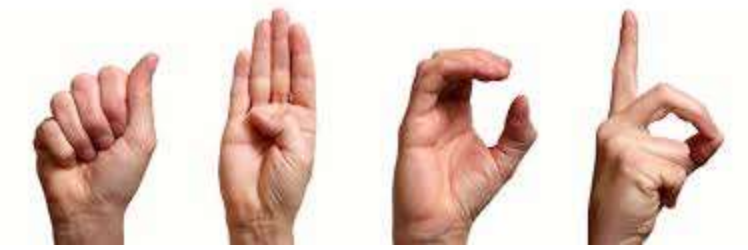

Figure 2

To process these different types on a system, different ways and algorithms can be used. We use the classification on basis of dimension and devise algorithms accordingly.

KINECT

Kinect is a line of motion sensing input devices by Microsoft for Xbox 360 and Xbox One video game consoles and Windows PCs. Based around a webcam-style add-on peripheral, it enables users to control and interact with their console/computer, through a natural user interface using gestures and spoken commands.

Figure 3

'Figure 3' shows the Kinect Xbox 360 camera used in our peoject.

The hardware components, including the Kinect sensor and the USB hub through which the Kinect sensor is connected to the computer.

The device features an "RGB camera, depth sensor and multi-array microphone running proprietary software", which provide full-body 3D motion capture, facial recognition and voice recognition capabilities.

There's a trio of hardware innovations working together within the Kinect sensor:

- Colour VGA video camera - This video camera aids in facial recognition and other detection features by detecting three colour components: red, green and blue. Microsoft calls this an "RGB camera" referring to the colour components it detects.
- Depth sensor - An infrared projector and a monochrome CMOS (complimentary metal-oxide semiconductor) sensor work together to "see" the room in 3-D regardless of the lighting conditions.
- Multi-array microphone - This is an array of four microphones that can isolate the voices of the players from the noise in the room. This allows the player to be a few feet away from the microphone and still use voice controls.

Kinect allows developers to write Kinecting apps in C++/CLI, C#, or Visual Basic .NET.

The software technology enables advanced gesture recognition, facial recognition and voice recognition.

Kinect has the following features:

- Raw sensor streams: Access to low-level streams from the depth sensor, colour camera sensor, and four-element microphone array.
- Skeletal tracking: The capability to track the skeleton image of one or two people moving within Kinect's field of view for gesture-driven applications.
- Advanced audio capabilities: Audio processing capabilities include sophisticated acoustic noise suppression and echo cancellation, beam formation to identify the current sound source, and integration with Windows speech recognition API.

It is used in various fields. It is used in surgery, gaming, designing and architecture, sign language etc.

3. APPROACH

APPROACH

Phase 1

In this phase we have implemented the 2D gesture recognition. The program works as follows:

- The code continuously streams video from the camera and processes the frames of the video to recognize the gesture.

- It then subtracts the RGB value of the pixels of the previous frame from the RGB values of the pixels of the current frame.

- Then this image is converted to octachrome (8 colors only - red, blue, green, cyan, magenta, yellow, white, black). This makes most of the pixels neutral or grey.

- This is followed by the greying of those pixels not surrounded by 20 non-grey pixels, in the function crosshair (IplImage*img1, IplImage* img2).

- The non-grey pixels that remain represent proper motion and noise is eliminated.

- A database is provided with the code which contains a set of points for each gesture.

- As the user performs the gesture, a set of points are generated (using the average of x & y coordinates of non-grey pixels in each frame) which are matched with the gestures in the databases to find the best match.

- To match the gestures the points are appropriately scaled as per their standard deviation and then corresponding points of the user's gesture and that from the database are compared.

- The gesture which has the least sum of squares of the differences between the corresponding points is returned as the match for the gesture.

- When the gesture recognized it is converted to speech using espeak library.

In this experiment we have used DROIDCAM which uses camera of phone rather than using a simple webcam. We have used it because mobile camera gives more pixels than the webcam which gives more accuracy for capturing the images.

The continuous processing of images was not possible in this approach. Because of comparing the captured image with stored gestures the processing becomes slow. So we needed a new approach which lead us to phase 2.

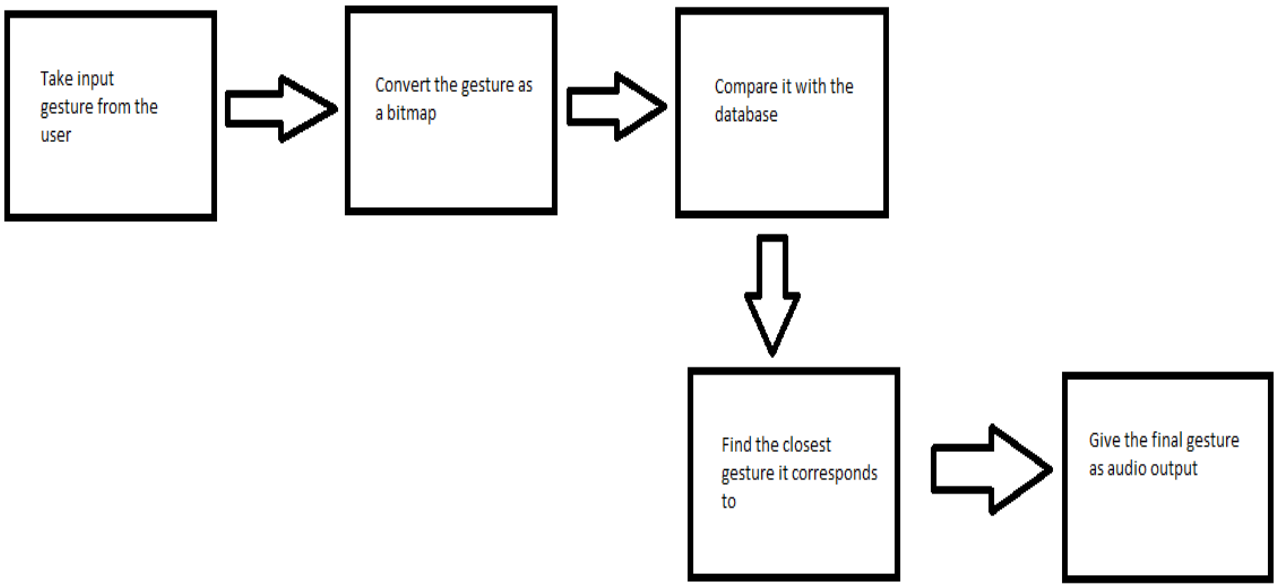

Figure 4

'Figure 4' shows the flowchart for algorithm used in phase 1.

Phase 2

In this phase of the project we have used KINECT XBOX 360 v1.8 for continuous processing. This phase gives more accuracy and is faster than the previous phase.

- The code continuously streams video from the camera and processes the frames of the video to recognize the gesture.

- The Kinect sensor uses a 20 joint tracking system in skeletal mode. They are namely head, shoulder center, shoulder right, shoulder left, elbow right, elbow left, wrist right, wrist left, hand right, hand left, spine, hip center, hip right, hip left, knee right, knee left, ankle right, ankle left, foot right, foot left.

- Kinect uses the structured light technique to generate discrete measurements of the three dimensional physical environment.

- Once this task has been done, it looks for a set of points whose coordinates vary at a constant depth. For those joints that remain obscured, the Kinect software goes a step further than just detecting and reacting to what it can "see."

- It distinguishes users and their movements even if they're partially hidden. Kinect extrapolates what the rest of your body is doing as long as it can detect some parts of it.

- The kinect in its skeletal mode is accessed using C# code.

- Each gesture is divided into segments. Each segment has if condition specifying the positions of joints.

- When a gesture is performed it is processed continuously. It checks all if conditions and identifies the gesture.

- The identified gesture is thus converted to speech using System.speech in visual studio.

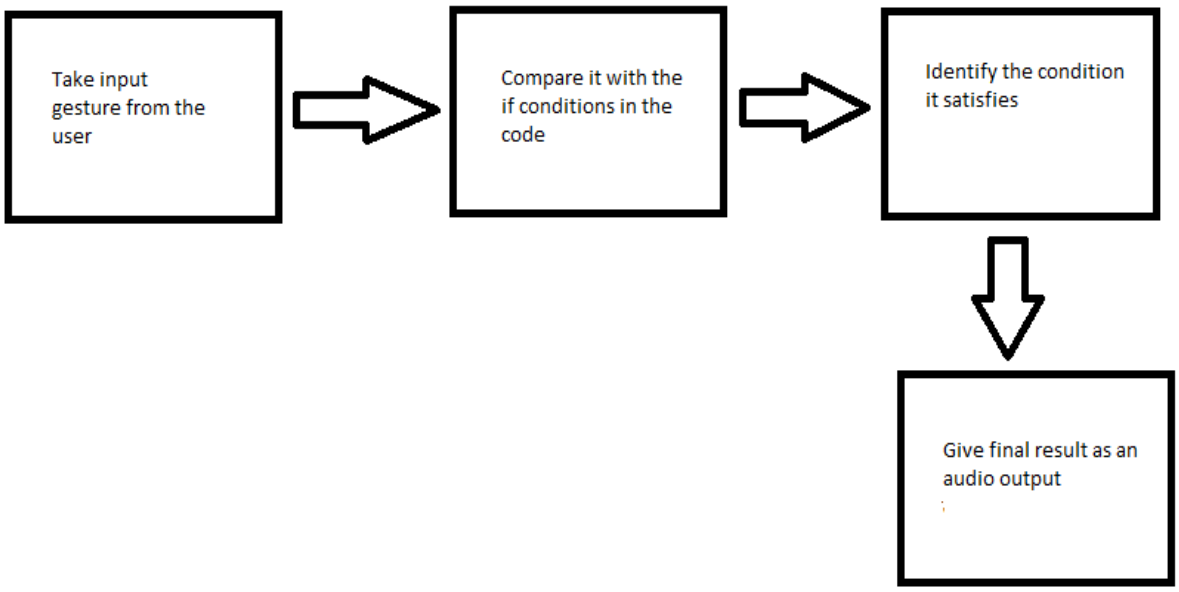

Figure 5

'Figure 5' shows the flow chart for the algorithm used in phase 2.

IMPLEMENTED GESTURES:

The gestures implemented in the phase 2 are:

1. Namaste
2. Hello
3. What is
4. Thank you

*The conditions that are used to identify 'NAMASTE' gesture are:

SEGMENT 1:

Y (handright) > Y (elbowright)

Y (handleft) > Y (elbowleft)

X (shoulderright) > X (handright)

X (shoulderleft) < X (handleft)

Y (shoulderright) > Y (handright)

Y (shoulderleft) > Y (handleft)

X (handright) – X (handleft) < 0.005

*The conditions that are used to identify 'HELLO' gesture are:

SEGMENT 1:

Y (head) < Y (handright)

Y (handright) > Y (elbowright)

X (handright) > X (elbowright)

SEGMENT 2:

Y (head) < Y (handright)

Y (handright) > Y (elbowright)

X (handright) < X (elbowright)

4. IMPLEMENTATION

EXPERIMENTAL SETUP

This section describes the implementation details of both the phases of project. It lists the libraries and platforms used to progress through the project. The methods for enhancement through knowledge acquisition are also illustrated.

SYSTEM ARCHITECTURE

Phase 1: This software is developed using 'C++' environment on Linux Operating system. The camera used is mobile camera with droid-cam application.

Phase 2: In this phase software is developed using 'visual C#' environment on windows Operating system. Kinect Xbox 360 (version 1.8) is the camera used for gesture input.

'C#' & 'C++' are chosen because of its ability to model the Object Oriented programming paradigm, since this work naturally divided into a series of classes and subclasses. Moreover C# provides easy to handle tools to build a better GUI to provide the software an elegant view.

In Phase 1 DROIDCAM is used which uses camera of mobile phone rather than using a simple webcam. Initially we used it because mobile camera generally provides more pixels than the webcam which gives more accuracy for capturing the images. However, the continuous processing of images was not possible in this approach as each new gesture input by user was compared with the stored gesture images thus resulting in slow processing. So we needed a new approach which lead us to phase 2. Kinect helped us to improve processing speed also allowed recognition of 3D gestures which was not possible with normal camera. In phase 1 we have implemented our project on ubuntu platform which I not user friendly for normal non-technical persons. Thus later to make our project easily accessible we implemented it on windows operating system which is the most used platform by non-technical persons.

PHASE 1

Here we are using mobile camera to take gestures as input using droid-cam application to connect it with computer. The definition of images are stored in the form of bitmap which are then compared with any new gesture user performs as input. With this only 2D view of images is available.

We have used the OpenCV library for handling and manipulating input from the webcam. Difference between subsequent frames helps detect motion. We included further modifications to eliminate noise so that only the moving object (hand or finger) may be interpreted. The interpreted gesture is scanned against a set of known gesture to find which gesture matches the best. The output in speech form is then produced. In the project we used the OpenCV library for handling and manipulating videos and images. Few basic gestures are incorporated, while the user is given the choice of adding other gestures.

Implementing the code is very simple. The packages required for compiling the code are gcc, opencv-doc, libcv2.1, linhighgui2.1, libcvaux2.1, libcv-dev, libcvaux-dev, linhighgui-dev, libx11-dev, and libxtst-dev. Also espeak package is required for tts (text to speech) functionality in Ubuntu. These packages are collectively installed from the Synaptic Package Manager or using individual system commands:

```
$ sudo apt-get install [package-name]
```

After installing all the packages the file - install.cpp is compiled using the command:

```
$ g++ install.cpp -o install
```

Running the file install in turn compiles all the other required files, provided the required libraries are installed correctly and up-to-date.

```
$ ./install
```

If the file 'install' runs correctly we don’t need to do this. Alternatively, we can compile all the files individually using the command:

$ g++ `pkg-config opencv --cflags` [filename].cpp -o [filename] `pkg-config opencv --libs` -lX11 -lXtst

The files to be compiled are: initialize.cpp, main.cpp, gesture.cpp, addgesture.cpp, checkgesture.cpp and delgesture.cpp.

Before beginning we run the file initialize:

$ ./initialize

This adjusts the frames per second (fps) of the webcam and the computer's memory and initializes the standard gestures.

HOW TO MODIFY OR DELETE GESTURES

Already existing gestures and their functions are checked by running the command:

```
$ ./checkgesture m

Load Successful: scmmd.bin

 Gesture: m

 Command: espeak -v en 'gesture M'
```

New gestures are added by the command:

```
 $ ./add-gesture [gesture-character] [custom command]

 For example:

 $ ./addgesture t thankyou
```

The [gesture-character] has to be a single character like 'w' or 'n', i.e. something like 'star' will not do for the [gesture-character]. Also the system command must be a valid system command. Then following the terminal instructions we perform the gesture three times. Each time we perform the gesture in a similar way and at the same speed. If a gesture already exists for the character, it will be overwritten.

HOW TO RUN

To run the program, we run the file 'gesture'. This file runs in the background without any window; the only way to know it is running is to check whether the webcam is on.

```
$ ./gesture
```

The file keeps running in the background taking input from the camera continuously. While making gestures we have to make sure that gestures are quite accurate and that the camera doesn't move. An arbitrary movement may also be interpreted as one of the gestures and the corresponding action is performed. All the gestures have to be done correctly and slowly. Each gesture have to last at least for half a second to be counted as a valid gesture.

Steps to be followed are:

1. User performs gesture to be added to database of the project

Figure 6

2. Gesture is then added to the database in form of bitmap image.

   Eg.: ‘Figure 7’ shows bitmap image formed when Gesture “M” is added to database

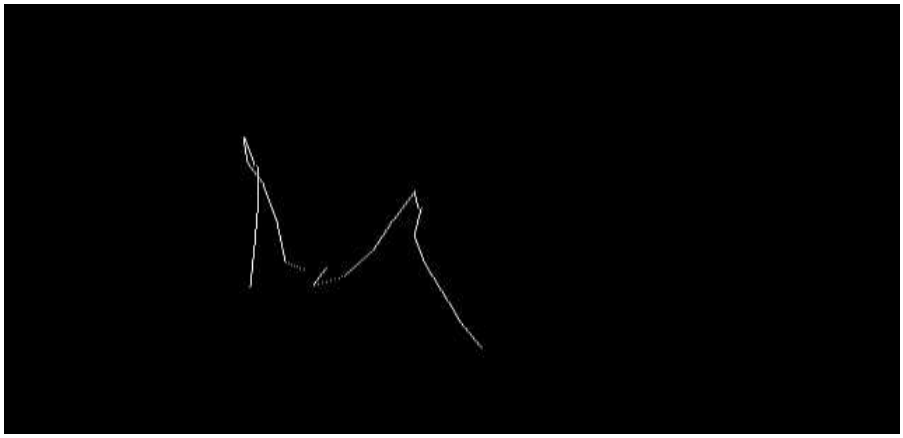

Figure 7

3. User now performs gesture to be recognized by software. This gesture is now converted to bitmap and compared against all images stored in database.

   Eg.: ‘Figure 8’ shows bitmap image formed with respect to Gesture “M” performed by new user.

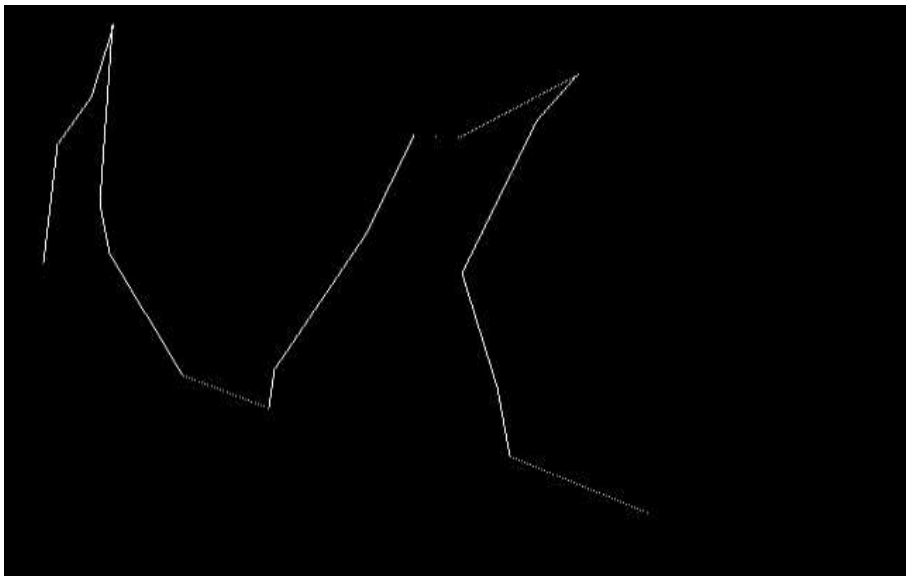

Figure 8

4. When recognized by software it gives output the word associated with that gesture in the form of speech using espeak command.

   Eg.: Gesture “M” recognized by program runs command as “espeak -v en 'gesture M'” .

PHASE 2

In this phase we have improved our processing speed [i.e. increased fps (frames per second)] using 3D camera Kinect Xbox 360. Also gesture definition is not stored in the form of bitmap but implicitly mentioned in the form of code snippet for each different gesture. The mode we have used here of Kinect is skeletal mode which provides 3D view of image in the form of X-Y-Z axes.

Initial Set-Up made is:

- Download and launch of KinectSDK-v1.8_1409-Setup executable
- To ensure that the Kinect sensor is connected to the power hub and the power hub is plugged into an outlet. USB cable is plugged from the power hub into a USB 3.0 port on our computer.
- To ensure that speaker is turned on.

Implementing the code is simple. Visual Studio 13 platform is used for development of the project in this phase. Text to speech conversion is done in windows by using ‘System.Speech’ library in Visual Studio.

HOW TO MODIFY OR DELETE GESTURES

We add new gestures by explicitly writing code snippet for its definition and including it in our program. Every gesture is divided into no. of segments. Each of the segment is then given a specific definition in code with different conditions provided based on the position of joints of skeleton body along X-Y-Z axes. Later these segments are ordered in code according to their occurrence in corresponding gesture.

Deletion of any gesture is performed easily by directly removing code snippet of that specific gesture from project.

HOW TO RUN

To run the project, we open its solution file in Visual Studio and then build and run the main file 'Program.cs' in Visual Studio. We can also run the project directly by running its executable file located in bin folder of project. This file runs in the background with console window.

The file keeps running in the background taking input from the camera continuously. While making gestures we have to make sure that user is in range of Kinect camera, gestures are quite accurate and that the camera doesn't move. An arbitrary movement may also be interpreted as one of the gestures and the corresponding action is performed. All the gestures have to be done correctly and with proper pace.

Steps to be followed are:

1. User performs gesture definition to be added to database of the project
   Eg.: 'Figure 9' shows the way "Hello" gesture will be defined to satisfy skeleton position conditions as shown above in above image.

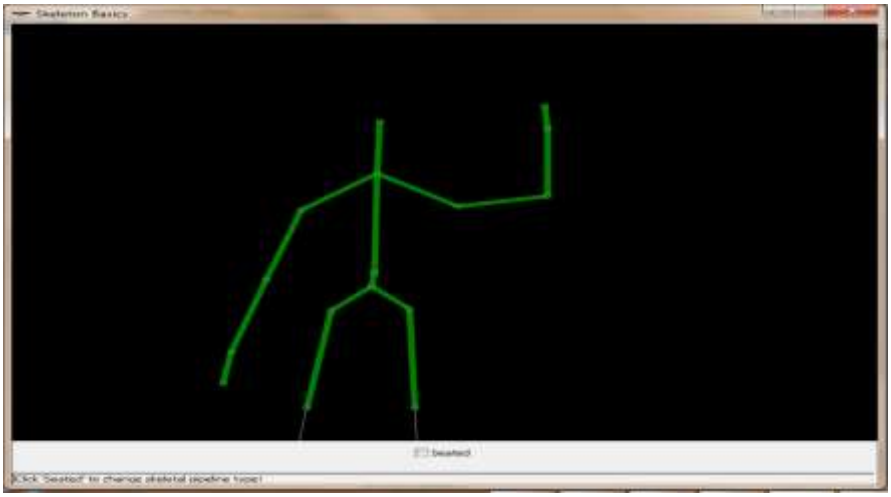

Figure 9

2. User now performs gesture to be recognized by software. This gesture is now compared against all gesture definitions mentioned in code.

   Eg.: ‘Figure 10’ shows user performing ”Hello” gesture. The image shows the skeleton view, depth view and 3D view of user performing “Hello” gesture from left to right respectively in image.

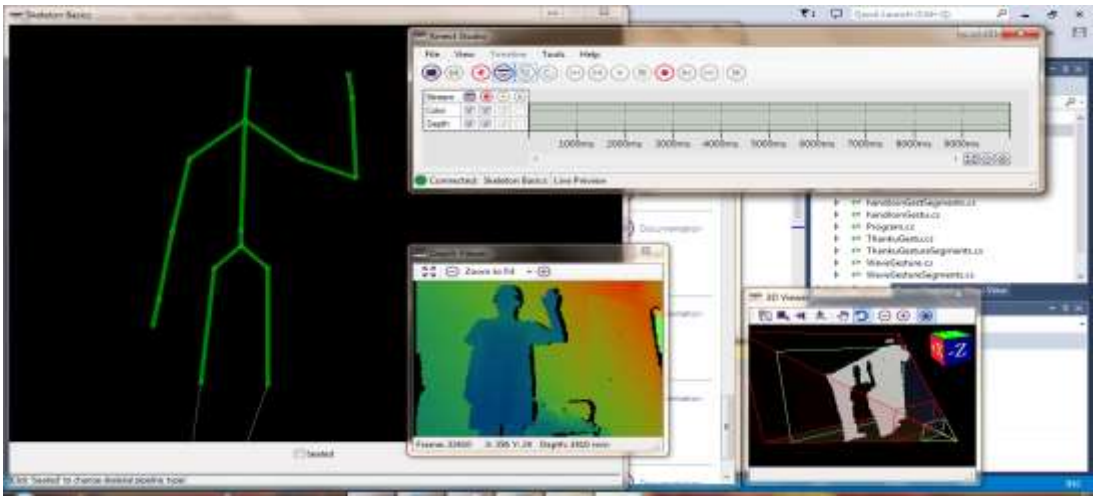

Figure 10

3. When recognized by software it gives output the word associated with that gesture on commend console and also in the form of speech using speak command.

   Eg.: ‘Figure 11 ’ shows “ Hello!! “ printed on output console window when user performs “Hello” gesture.

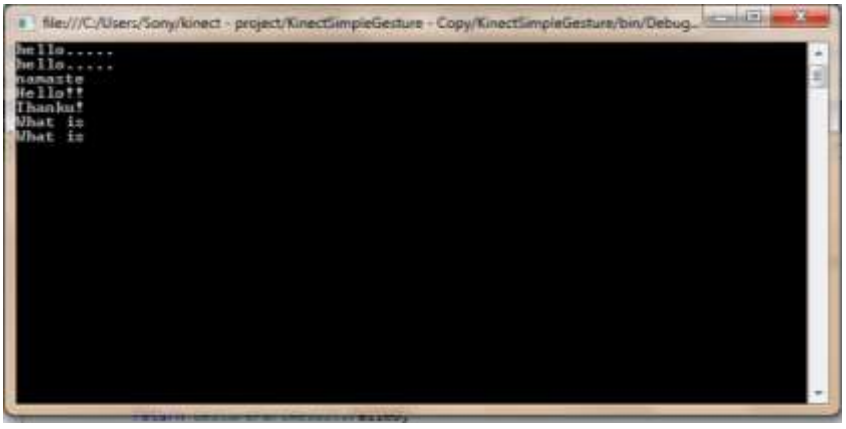

Figure 11

5. CHALLENGES

CHALLENGES

Phase 1:

1. The fps (frames per second) of webcam was around 10 which gave less accuracy in capturing the image. So we used mobile camera using DROIDCAM application to access the camera. This gave us fps around 18-20 which gave us better results.

2. The environment during the recording should be still with less disturbance.

3. The gestures need to be accurate and complete. Incomplete gestures don't give appropriate results.

4. Maintaining a database caused the application to run slowly and processing took time as it had to search through entire database to find the appropriate one. This has been worked upon in phase 2.

5. 3D gestures were not detected which is why we needed phase 2 which uses z axis also for detecting depth of images.

Phase 2:

1. The user performing the gesture should be in the range of kinect.

2. The code runs slow in the visual studio 2013 so we need to run the .exe file directly from the bin folder present in the project.

3. Gestures in the sign language mostly consists of finger movements but the kinect version 1.8 is unable to detect finger joints. For that version 2.0 should be used.

4. The segments of the gestures should be distinguishable from each other so that they are identified accurately.

WHAT WE HAVE LEARNT

Phase 1

We learnt how to use opencv libraries, espeak command on linux. We also got to know about the concept of frames per second (fps) and how to manage with memory requirements for achieving accuracy.

Phase 2

We learnt about the kinect and different modules it supports, how and where to use them. Also developing in c# was a new learning.

Apart from technical learnings, for better understanding of the project and the approach to follow we had to learn basic sign language gestures. Also we learnt finer nuances of team work and how to manage the time. We gained deeper understanding of the project by sharing our knowledge. Individual ideas also helped to develop the project.

6. CONCLUSION

CONCLUSION

Phase 1

In this we have implemented the recognition of some basic 2D gestures and converted it to the speech. In this, system recognises the gesture through camera and compares with the database which contains some gestures and converts them to speech with the help of espeak library. Gesture is recognised using grayscale modules in opencv library for image processing.

Phase 2

This system recognises the gestures of sign language and converts it to the speech. In this, Hand Gesture Recognition system based on Microsoft Kinect for Xbox is introduced. The system is motivated by the importance of real-time communication under specific situations. Here Kinect is used to increase recognition speed and accuracy. Using depth we can get wider range of gesture and it becomes easy to distinguish different gestures.

Using this system we can identify and recognise gesture and differentiate each and every gesture properly. In this system, a camera reads the movements of the human body and recognises the gesture according to conditions provided and convert it to speech with the help of espeak library.

This system helps deaf and dumb people to communicate with ease.

7. FUTURE SCOPE

FUTURE SCOPE

Some directions for future work are to further improve the accuracy of recognition, to add more scenarios such as fingerspelling of the alphabet.

The system may allow the recognition of continuous gestures that form words or sentences, and the narrowly defined scenarios may be expanded to discourse contexts.

8. REFERENCES

REFRENCES

1. Khan, Rafiqul Zaman, and Noor Adnan Ibraheem. "HAND GESTURE RECOGNITION: ALiterature." (2012).

2. Pradhan, Ashis, M. K. Ghose, Mohan Pradhan, Sameer Qazi, Tim Moors, Islam M. Ezz EL-Arab, H. Shehab El–Din et al. "A hand gesture recognition using feature extraction." *Int J Curr Eng Technol* 2, no. 4 (2012): 323-327.

3. Song, Lin, Rui Min Hu, Yu Lian Xiao, and Li Yu Gong. "Real-time 3d hand tracking from depth images." *Advanced Materials Research* 765 (2013): 2822-2825.

4. Nguyen, Phuoc Loc, Vivienne Falk, and Sarah Ebling. "Building an Application for Learning the Finger Alphabet of Swiss German Sign Language through Use of the Kinect." In *Computers Helping People with Special Needs*, pp. 404-407. Springer International Publishing, 2014.

5. Li, Yi. "Hand gesture recognition using Kinect." In *Software Engineering and Service Science (ICSESS), 2012 IEEE 3rd International Conference on*, pp. 196-199. IEEE, 2012.

6. Muttena, Sanjivi, S. Sriram, and R. Shiva. "Mapping gestures to speech using the kinect." In *Science Engineering and Management Research (ICSEMR), 2014 International Conference on*, pp. 1-5. IEEE, 2014.

7. Lahamy, Hervé, and D. Lichti. "Real-time hand gesture recognition using range cameras." In *Proceedings of the Canadian Geomatics Conference, Calgary, Canada*. 2010.

8. Raheja, Jagdish L., Ankit Chaudhary, and Kunal Singal. "Tracking of fingertips and centers of palm using kinect." In *Computational intelligence, modelling and simulation (CIMSiM), 2011 third international conference on*, pp. 248-252. IEEE, 2011.

APPENDIX

Code Snippet of Phase 2:

The code implemented in this phase is as follows:

Program.cs

```
using Microsoft.Kinect;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Speech.Synthesis;

namespace KinectSimpleGesture
{
   class Program

   {
      // Initiate all gestures
      static WaveGesture _gesture = new WaveGesture();
      static ThankuGestu thanku_gesture = new ThankuGestu();
      static handJoinGestu handJoin_gesture = new handJoinGestu();
      static WhatGesture what_gesture = new WhatGesture();

      static void Main(string[] args)
      {
         var sensor = KinectSensor.KinectSensors.Where(s => s.Status ==
KinectStatus.Connected).FirstOrDefault();
         Console.WriteLine("hello.....");
         if (sensor != null)
         {
            Console.WriteLine("hello.....");
            sensor.SkeletonStream.Enable();
            sensor.SkeletonStream.TrackingMode = SkeletonTrackingMode.Seated; // Use seated
tracking
            sensor.SkeletonFrameReady += Sensor_SkeletonFrameReady;
             _gesture.GestureRecognized += Gesture_GestureRecognized;
            thanku_gesture.GestureRecognized += Gesture_GestureRecognized;
            handJoin_gesture.GestureRecognized += Gesture_GestureRecognized;
            what_gesture.GestureRecognized += Gesture_GestureRecognized;
            sensor.Start();
         }

         Console.ReadKey();
      }
```

```
      static void Sensor_SkeletonFrameReady(object sender, SkeletonFrameReadyEventArgs e)
      {
         using (var frame = e.OpenSkeletonFrame())
         {
            if (frame != null)
            {
               Skeleton[] skeletons = new Skeleton[frame.SkeletonArrayLength];

               frame.CopySkeletonDataTo(skeletons);

               if (skeletons.Length > 0)
               {
                  var user = skeletons.Where(u => u.TrackingState ==
SkeletonTrackingState.Tracked).FirstOrDefault();
                  if (user != null)
                  {
                     _gesture.Update(user);
                     thanku_gesture.thankuUpdate(user);
                     handJoin_gesture.handJoinUpdate(user);
                     what_gesture.whatUpdate(user);
                  }
               }
            }
         }
      }

      // to give speak commands if gesture is recognized
      static void Gesture_GestureRecognized(object sender, EventArgs e)
      {
         WaveGesture wg=new WaveGesture();
         ThankuGestu tg = new ThankuGestu();
         handJoinGestu ng = new handJoinGestu();
         WhatGesture whatg = new WhatGesture();

         // if gesture recognized is hello
         if (sender.GetType().Name.ToString().Equals(wg.GetType().Name.ToString()))
         {
            Console.WriteLine("Hello!!");

           // to give speack command for hello gesture
            using (SpeechSynthesizer synth =
            new SpeechSynthesizer())
            {
               synth.Speak("Hello");
            }
         }
```

```
            // if gesture recognized is thanku
            else if (sender.GetType().Name.ToString().Equals(tg.GetType().Name.ToString()))
            {
               Console.WriteLine("Thanku!");

              // to give speack command for thanku gesture
               using (SpeechSynthesizer synth =
               new SpeechSynthesizer())
               {
                  synth.Speak("Thank You");
               }
            }
            // if gesture recognized is namaste
            else if (sender.GetType().Name.ToString().Equals(ng.GetType().Name.ToString()))
            {
               Console.WriteLine("namaste");

              // to give speack command for namaste gesture
               using (SpeechSynthesizer synth =
               new SpeechSynthesizer())
               {
                  synth.Speak("namaste");
               }
            }
            // if gesture recognized is what is
            else if (sender.GetType().Name.ToString().Equals(whatg.GetType().Name.ToString()))
            {
               Console.WriteLine("What is");

              // to give speack command for what is gesture
               using (SpeechSynthesizer synth =
               new SpeechSynthesizer())
               {
                  synth.Speak("What is");
               }
            }
         }
      }
   }
```

Now we definition for each gesture:

Eg.: Thanku gesture code snippet is given below

Thankugesture.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.Kinect;
using System;

namespace KinectSimpleGesture
{
   class ThankuGestu
   {
      readonly int WINDOW_SIZE = 30;

      ThankuGestureSegment[] _thankusegments;

      int _thankucurrentSegment = 0;
      int _thankuframeCount = 0;

      public event EventHandler GestureRecognized;

      public ThankuGestu()
      {
         ThankuSegment1 waveRightSegment1 = new ThankuSegment1();
         ThankuSegment2 waveRightSegment2 = new ThankuSegment2();
           _thankusegments = new ThankuGestureSegment[]
         {
            waveRightSegment2,
            waveRightSegment1
         };
      }
      // Updates the current gesture.
      public void thankuUpdate(Skeleton skeleton)
      {
         GesturePartResult result = _thankusegments[_thankucurrentSegment].thankuUpdate(skeleton);
            if (result == GesturePartResult.Succeeded)
            {
               if (_thankucurrentSegment + 1 < _thankusegments.Length)
               {

                  _thankucurrentSegment++;
                  _thankuframeCount = 0;
               }
```

```
else
                {
                    if (GestureRecognized != null)
                    {
                        GestureRecognized(this, new EventArgs());
                        Reset();
                    }
                }
            }
            else if (result == GesturePartResult.Failed || _thankuframeCount == WINDOW_SIZE)
            {
                Reset();
            }
            else
            {
                _thankuframeCount++;
            }
        }
        // Resets the current gesture.
        public void Reset()
        {
            _thankucurrentSegment = 0;
            _thankuframeCount = 0;
        }
    }

}
```

Now each gesture is divided into no. of segments and each segment is given its definition in code.

Eg.: Thank u gesture is divided into two segments as given follows:

Thanku gesture segments.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Kinect;
using System;
using System.Text;

namespace KinectSimpleGesture
{
   // Represents a single gesture segment which uses relative positioning of body parts to detect a gesture.
   public interface ThankuGestureSegment
   {
      //Updates the current gesture.
      GesturePartResult thankuUpdate(Skeleton skeleton);
   }
   public class ThankuSegment1 : ThankuGestureSegment
   {
      //Updates the current gesture.
      public GesturePartResult thankuUpdate(Skeleton skeleton)
      {
         // Hand above elbow

               if (skeleton.Joints[JointType.HandRight].Position.Z < skeleton.Joints[JointType.Head].Position.Z)
               {
                  return GesturePartResult.Succeeded;
               }
         // Hand dropped
         return GesturePartResult.Failed;
      }
   }
    public class ThankuSegment2 : ThankuGestureSegment
   {
      /// Updates the current gesture.

      public GesturePartResult thankuUpdate(Skeleton skeleton)
      {
         // Hand above elbow
         if ((skeleton.Joints[JointType.Head].Position.Y > skeleton.Joints[JointType.HandRight].Position.Y) && (skeleton.Joints[JointType.Spine].Position.Y < skeleton.Joints[JointType.HandRight].Position.Y) && (skeleton.Joints[JointType.ShoulderLeft].Position.X < skeleton.Joints[JointType.HandRight].Position.X) && (skeleton.Joints[JointType.HandRight].Position.X < skeleton.Joints[JointType.ShoulderRight].Position.X))
         {
```

```
            if (skeleton.Joints[JointType.HandRight].Position.Y >
skeleton.Joints[JointType.ElbowRight].Position.Y && skeleton.Joints[JointType.HandRight].Position.Y >
skeleton.Joints[JointType.ShoulderCenter].Position.Y)
                {

                    // Hand left of elbow
                    return GesturePartResult.Succeeded;
                }
            }
            // Hand dropped
            return GesturePartResult.Failed;
        }
    }
```

Status of Each segment whether successfully recognized or not is given by code below:

GesturePartResult.cs

```
using System;

namespace KinectSimpleGesture
{
    /// <summary>
    /// Represents the gesture part recognition result.
    /// </summary>
    public enum GesturePartResult
    {
        /// <summary>
        /// Gesture part failed.
        /// </summary>
        Failed,

        /// <summary>
        /// Gesture part succeeded.
        /// </summary>
        Succeeded
    }
}
```