A MINI PROJECT REPORT

ON

Sign Language Recognition Using Hand Gestures

Submitted to Mumbai University

In the partial fulfillment of the requirement for the award of the degree of

Bachelor of Engineering

In

COMPUTER ENGINEERING

By

Mr. Shadab Shaikh. **(17DCO74)**

Mr. Obaid Kazi. **(17DCO69)**

Mr. Mohd Adnan Ansari. **(17DCO63)**

Under the guidance of

Mr. Muhammed Salman Shamsi

Assistant Professor

Department of Computer Engineering

Anjuman-I-Islam's Kalsekar Technical Campus

Affiliated to Mumbai University

KHANDA GOAN, NEW PANVEL, NAVI MUMBAI, MAHARASHTRA

2018-2019

Department of Computer Engineering

Anjuman-I-Islam's Kalsekar Technical Campus

Affiliated to Mumbai University

KHANDA GAON, NEW PANVEL, NAVI MUMBAI, MAHARASHTRA

2018-2019

DECLARATION BY THE CANDIDATE

Shadab Shaikh, Obaid Kazi, Mohd Adnan Ansari bearing **Roll number: 17DCO74**, **17DCO69, 17DCO63** respectively, hereby declare that the mini project report entitled **"Sign Language Recognition Using Hand Gestures"**, is a record of bonafide work carried out by us and the results embodied in this project have not been reproduced or copied from any source. The results of this project report have not been submitted to any other University or Institute for the award of any other Degree or Diploma.

Shadab Shaikh	**(17DCO74)**
Obaid Kazi	**(17DCO69)**
Mohd Adnan Ansari	**(17DCO63)**

Department of Computer Engineering

Anjuman-I-Islam's Kalsekar Technical Campus

Affiliated to Mumbai University

KHANDA GAON, NEW PANVEL, NAVI MUMBAI, MAHARASHTRA

2018-2019

CERTIFICATE

This is to certify that the project report entitled **"Sign Language Recognition Using Hand Gestures"**, submitted by **Mr. Shadab Shaikh**, **Mr. Obaid Kazi, Mr. Mohd Adnan Ansari** bearing **Roll. No.: 17DCO74, 17DCO69, 17DCO63** respectively, in the partial fulfillment of the requirements for the award of the degree of **Bachelor of Computer Engineering** is a record of bonafide work carried out by them for the course **Mini Project CSM605**.

Mini Project Guide

(Prof. Muhammed Salman Shamsi)

Mini Project Coordinator

(Prof. Muhammed Salman Shamsi)

Program Owner

(Prof. Tabrez Khan)

INDEX

CONTENTS

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

Some of the major problems faced by a person who are unable to speak is they cannot express their emotion as freely in this world. Utilize that voice recognition and voice search systems in smartphone(s).[1] Audio results cannot be retrieved. They are not able to utilize (Artificial Intelligence/personal Butler) like google assistance, or Apple's SIRI etc.[2] because all those apps are based on voice controlling.

There is a need for such platforms for such kind of people. American Sign Language (ASL) is a complete, complex language that employs signs made by moving the hands combined with facial expressions and postures of the body. It is the go-to language of many North Americans who are not able to talk and is one of various communication alternatives used by people who are deaf or hard-of-hearing.[3]

While sign language is very essential for deaf-mute people, to communicate both with normal people and with themselves, is still getting less attention from the normal people.[4] The importance of sign language has been tending to ignored, unless there are areas of concern with individuals who are deaf-mute. One of the solutions to talk with the deaf-mute people is by using the mechanisms of sign language.

Hand gesture is one of the methods used in sign language for non-verbal communication. It is most commonly used by deaf & dumb people who have hearing or talking disorders to communicate among themselves or with normal people.[3] Various sign language systems have been developed by many manufacturers around the world but they are neither flexible nor cost-effective for the end users.[6]

1.2 SCOPE

One of the solutions to communicate with the deaf-mute people is by using the services of sign language interpreter. But the usage of sign language interpreters could be

expensive.[3] Cost-effective solution is required so that the deaf-mute and normal people can communicate normally and easily. [3]

Our strategy involves implementing such an application which detects pre-defined American sign language (ASL) through hand gestures. For the detection of movement of gesture, we would use basic level of hardware component like camera and interfacing is required. Our application would be a comprehensive User-friendly Based system built on PyQt5 module.

Instead of using technology like gloves or kinect, we are trying to solve this problem using state of the art computer vision and machine learning algorithms. [6]

This application will comprise of two core module one is that simply detects the gesture and displays appropriate alphabet. The second is after a certain amount of interval period the scanned frame would be stored into buffer so that a string of character could be generated forming a meaningful word.

Additionally, an-addon facility for the user would be available where a user can build their own custom-based gesture for a special character like period (.) or any delimiter so that a user could form a whole bunch of sentences enhancing this into paragraph and likewise. Whatever the predicted outcome was, it would be stored into a .txt file.

1.3 PROBLEM STATEMENT

Given a hand gesture, implementing such an application which detects pre-defined American sign language (ASL) in a real time through hand gestures and providing facility for the user to be able to store the result of the character detected in a txt file, also allowing such users to build their customized gesture so that the problems faced by persons who aren’t able to talk vocally can be accommodated with technological assistance and the barrier of expressing can be overshadowed.

CHAPTER 2

SYSTEM SPECIFICATION

2.1 SYSTEM REQUIREMENT

2.1.1 HARDWARE REQUIREMENTS

- Intel Core i3 3rd gen processor or later.
- 512 MB disk space.
- 512 MB RAM.
- Any external or inbuild camera with minimum pixel resolution 200 x 200 (300ppi or 150lpi) 4-megapixel cameras and up.

2.1.2 SOFTWARE REQUIREMENTS

- Microsoft Windows XP or later / Ubuntu 12.0 LTS or later /MAC OS 10.1 or later.
- Python Interpreter (3.6).
- TensorFlow framework, Keras API.
- PyQT5, Tkinter module.
- Python OpenCV2, scipy, qimage2ndarray, winGuiAuto, pypiwin32, sys, keyboard, pyttsx3, pillow libraries.

2.2 SYSTEM FEATURES

- User-friendly based GUI built using industrial standard PyQT5.
- Real time American standard character detection based on gesture made by user.
- Customized gesture generation.
- Forming a stream of sentences based on the gesture made after a certain interval of time.

CHAPTER 3

SYSTEM DESIGN

3.1 SYSTEM ARCHITECTURE

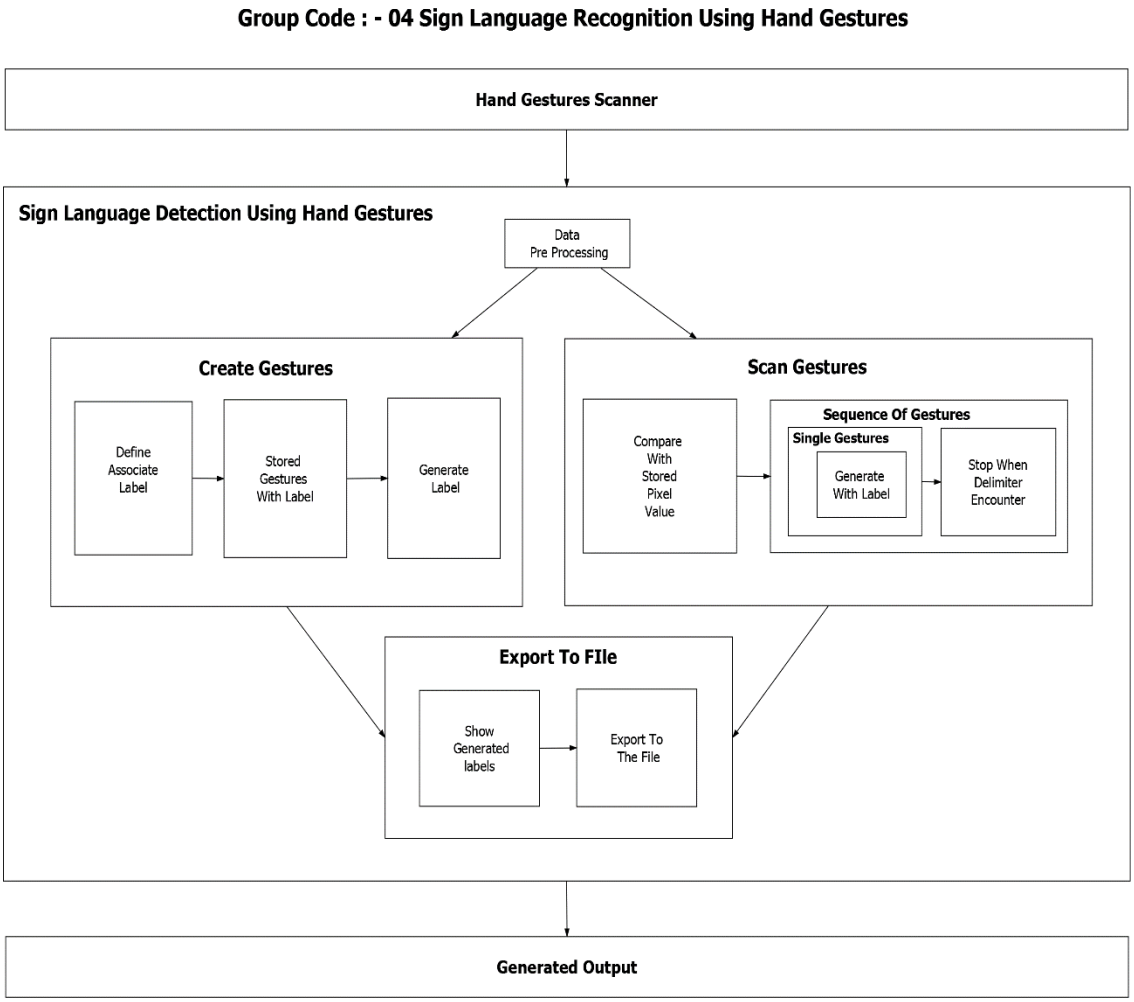

Fig 1: System Architecture for Sign Language Recognition Using Hand Gestures.

3.2 MODULES IN THE SYSTEM

- **Data Pre-Processing –** In this module, based on the object detected in front of the camera its binary images is being populated. Meaning the object will be filled with solid white and background will be filled with solid black. Based on the pixel's regions, their numerical value in range of either 0 or 1 is being given to next process for modules.

- **Scan Single Gesture –** A gesture scanner will be available in front of the end user where the user will have to do a hand gesture. Based on Pre-Processed module output, a user shall be able to see associated label assigned for each hand gestures, based on the predefined American Sign Language (ASL) standard inside the output window screen.
- **Create gesture** –A user will give a desired hand gesture as an input to the system with the text box available at the bottom of the screen where the user needs to type whatever he/she desires to associate that gesture with. This customize gesture will then be stored for future purposes and will be detected in the upcoming time.
- **Formation of a sentence –** A user will be able to select a delimiter and until that delimiter is encountered every scanned gesture character will be appended with the previous results forming a stream of meaning-full words and sentences.
- **Exporting –** A user would be able to export the results of the scanned character into an ASCII standard textual file format.

3.3 USE CASE DIAGRAM

Group Code: -4 Sign Language Recognition Using Hand Gestures

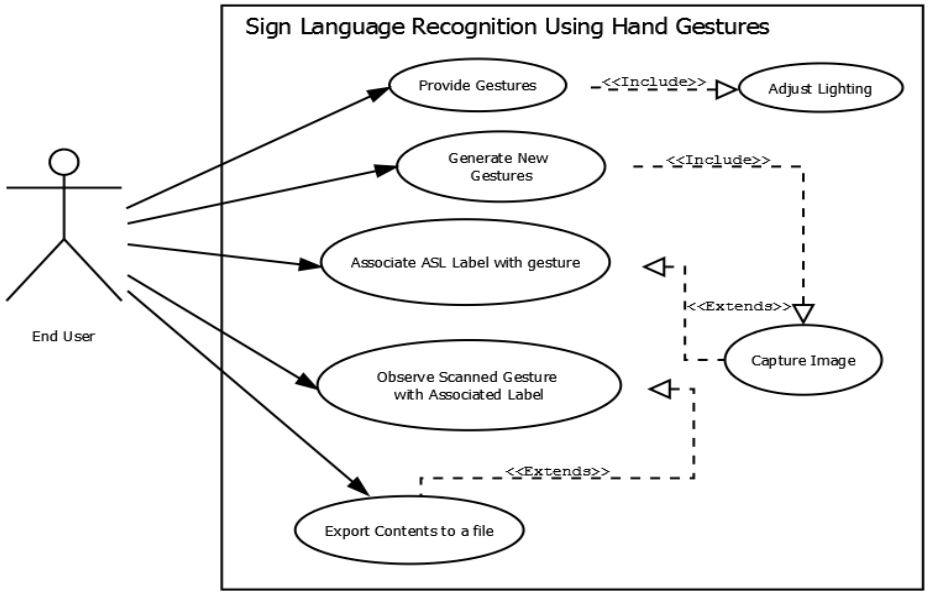

Fig 2: Use Case Diagram for Sign Language Recognition Using Hand Gestures.

3.4 ACTIVITY DIAGRAM

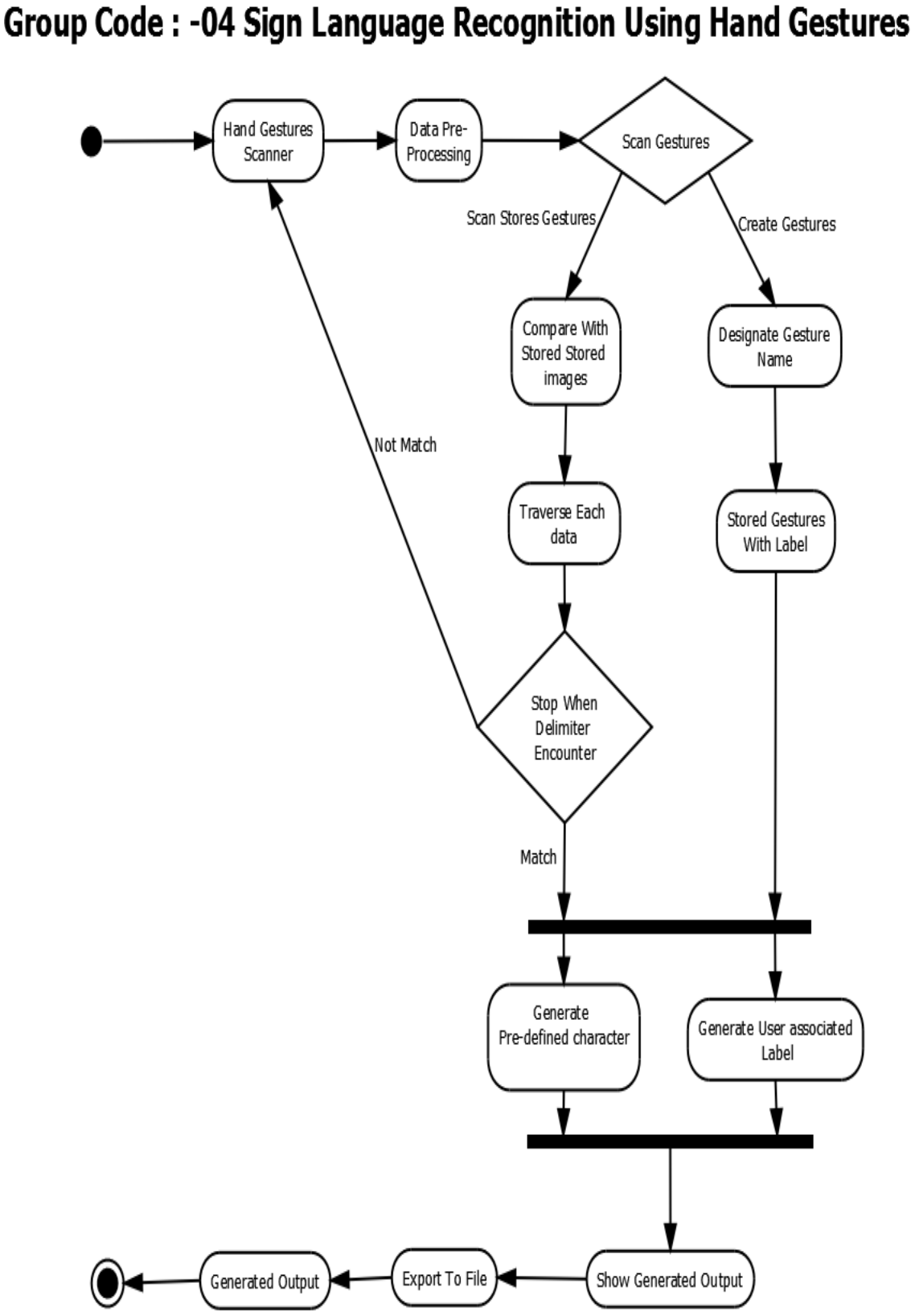

Fig 3: Activity Diagram for Sign Language Recognition Using Hand Gestures.

CHAPTER 4

IMPLEMENTATION

4.1 CODE SNIPPETS

Dashboard.py

```
__author__ = 'Shadab Shaikh, Obaid Kazi, Ansari Mohd Adnan'

from PyQt5 import QtWidgets, uic
from PyQt5.QtWidgets import QMessageBox
from PyQt5.QtCore import QUrl
from PyQt5.QtGui import QImage
from PyQt5.QtGui import QPixmap
from PyQt5 import QtCore                                    #importing pyqt5 libraries
from scipy.ndimage import imread                            #will help in reading the images
from PyQt5.QtCore import QTimer,Qt
from PyQt5 import QtGui
from tkinter import filedialog                              #for file export module
from tkinter import *
import tkinter as tk
from matplotlib import pyplot as plt                        #for gesture viewer
from matplotlib.widgets import Button
import sys                                                  #for pyqt
import os                                                   #for removal of files
import cv2                                                  #for the camera operations
import numpy as np                                          #proceesing on images
import qimage2ndarray                                       #convers images into matrix
import win32api
import winGuiAuto
import win32gui
import win32co                          #for removing title cv2 window and always on top
import keyboard                         #for pressing keys
import pyttsx3                          #for tts assistance
import shutil                           #for removal of directories
index = 0                               #index used for gesture viewer
engine = pyttsx3.init()                 #engine initialization for audio tts assistance

def nothing(x):
        pass

image_x, image_y = 64,64                                    #image resolution

from keras.models import load_model
classifier = load_model('ASLModel.h5')                      #loading the model

def fileSearch():
        """Searches each file ending with .png in SampleGestures dirrectory so that
        custom gesture could be passed to predictor() function"""
        fileEntry=[]
        for file in os.listdir("SampleGestures"):
            if file.endswith(".png"):
                fileEntry.append(file)
        return fileEntry

def load_images_from_folder(folder):
    """Searches each images in a specified directory"""
    images = []
    for filename in os.listdir(folder):
        img = cv2.imread(os.path.join(folder,filename))
        if img is not None:
            images.append(img)
    return images

def toggle_imagesfwd(event):
        """displays next images act as a gesutre viewer"""
        img=load_images_from_folder('TempGest/')
        global index

        index += 1

        try:
                if index < len(img):
                        plt.axes()
                        plt.imshow(img[index])
                        plt.draw()
```

```
        except:
                pass

def toggle_imagesrev(event):
        """displays previous images act as a gesutre viewer"""
        img=load_images_from_folder('TempGest/')
        global index

        index -= 1

        try:
                if index < len(img) and index>=0:
                        plt.axes()
                        plt.imshow(img[index])
                        plt.draw()
        except:
                pass

def openimg():
        """displays predefined gesture images at right most window"""
        cv2.namedWindow("Image", cv2.WINDOW_NORMAL )
        image = cv2.imread('template.png')
        cv2.imshow("Image",image)
        cv2.setWindowProperty("Image",cv2.WND_PROP_FULLSCREEN,cv2.WINDOW_FULLSCREEN)
        cv2.resizeWindow("Image",298,430)
        cv2.moveWindow("Image", 1052,214)


def removeFile():
        """Removes the temp.txt and tempgest directory if any stop button is pressed oor
        application is closed"""
        try:
                os.remove("temp.txt")
        except:
                pass
        try:
                shutil.rmtree("TempGest")
        except:
                pass

def clearfunc(cam):
        """shut downs the opened camera and calls removeFile() Func"""
        cam.release()
        cv2.destroyAllWindows()
        removeFile()

def clearfunc2(cam):
        """shut downs the opened camera"""
        cam.release()
        cv2.destroyAllWindows()

def saveBuff(self,cam,finalBuffer):
        """Save the file as temp.txt if save button is pressed in sentence formation
        through gui"""
        cam.release()
        cv2.destroyAllWindows()
        if(len(finalBuffer)>=1):
                f=open("temp.txt","w")
                for i in finalBuffer:
                        f.write(i)
                f.close()


def capture_images(self,cam,saveimg,mask):
        """Saves the images for custom gestures if button is pressed in custom gesture
        generationn through gui"""
        cam.release()
        cv2.destroyAllWindows()
        if not os.path.exists('./SampleGestures'):
                os.mkdir('./SampleGestures')

        gesname=saveimg[-1]
        if(len(gesname)>=1):
                img_name = "./SampleGestures/"+"{}.png".format(str(gesname))
                save_img = cv2.resize(mask, (image_x, image_y))
                cv2.imwrite(img_name, save_img)
```

```
def controlTimer(self):
        # if timer is stopped
        self.timer.isActive()
        # create video capture
        self.cam = cv2.VideoCapture(0)
        # start timer
        self.timer.start(20)


def predictor():
        """ Depending on model loaded and customgesture saved prediction is made by
        checking array or through SiFt algo"""
        import numpy as np
        from keras.preprocessing import image
        test_image = image.load_img('1.png', target_size=(64, 64))
        test_image = image.img_to_array(test_image)
        test_image = np.expand_dims(test_image, axis = 0)
        result = classifier.predict(test_image)
        gesname=''
        fileEntry=fileSearch()
        for i in range(len(fileEntry)):
                image_to_compare = cv2.imread("./SampleGestures/"+fileEntry[i])
                original = cv2.imread("1.png")
                sift = cv2.xfeatures2d.SIFT_create()
                kp_1, desc_1 = sift.detectAndCompute(original, None)
                kp_2, desc_2 = sift.detectAndCompute(image_to_compare, None)

                index_params = dict(algorithm=0, trees=5)
                search_params = dict()
                flann = cv2.FlannBasedMatcher(index_params, search_params)

                matches = flann.knnMatch(desc_1, desc_2, k=2)

                good_points = []
                ratio = 0.6
                for m, n in matches:
                      if m.distance < ratio*n.distance:
                            good_points.append(m)
                if(abs(len(good_points)+len(matches))>20):
                        #goodpoints and matccches sum from 1.png and customgestureimages
                        #is grater than 20
                        gesname=fileEntry[i]
                        gesname=gesname.replace('.png','')
                        if(gesname=='sp'):
                                #sp is replaced with <space>
                                gesname=' '
                        return gesname

        if result[0][0] == 1:
                  return 'A'
        elif result[0][1] == 1:
                  return 'B'
        elif result[0][2] == 1:
                  return 'C'
        elif result[0][3] == 1:
                  return 'D'
        elif result[0][4] == 1:
                  return 'E'
        elif result[0][5] == 1:
                  return 'F'
        elif result[0][6] == 1:
                  return 'G'
        elif result[0][7] == 1:
                  return 'H'
        elif result[0][8] == 1:
                  return 'I'
        elif result[0][9] == 1:
                  return 'J'
        elif result[0][10] == 1:
                  return 'K'
        elif result[0][11] == 1:
                  return 'L'
        elif result[0][12] == 1:
                  return 'M'
        elif result[0][13] == 1:
                  return 'N'
```

```
        elif result[0][14] == 1:
                return 'O'
        elif result[0][15] == 1:
                return 'P'
        elif result[0][16] == 1:
                return 'Q'
        elif result[0][17] == 1:
                return 'R'
        elif result[0][18] == 1:
                return 'S'
        elif result[0][19] == 1:
                return 'T'
        elif result[0][20] == 1:
                return 'U'
        elif result[0][21] == 1:
                return 'V'
        elif result[0][22] == 1:
                return 'W'
        elif result[0][23] == 1:
                return 'X'
        elif result[0][24] == 1:
                return 'Y'
        elif result[0][25] == 1:
                return 'Z'


def checkFile():
        """retrieve the content of temp.txt for export module """
        checkfile=os.path.isfile('temp.txt')
        if(checkfile==True):
                fr=open("temp.txt","r")
                content=fr.read()
                fr.close()
        else:
                content="No Content Available"
        return content

class Dashboard(QtWidgets.QMainWindow):


        def __init__(self):
                super(Dashboard, self).__init__()
                self.setWindowFlags(QtCore.Qt.WindowCloseButtonHint |
                QtCore.Qt.WindowMinimizeButtonHint | QtCore.Qt.FramelessWindowHint)
                cap = cv2.VideoCapture('gestfinal2.min.mp4')

                # Read until video is completed
                while(cap.isOpened()):
                        ret, frame = cap.read()
                        if ret == True:
                # Capture frame-by-frame
                                ret, frame = cap.read()
                                cv2.namedWindow("mask", cv2.WINDOW_NORMAL)
                                cv2.imshow("mask", frame)

                                cv2.setWindowProperty("mask",cv2.WND_PROP_FULLSCREEN,cv2.W
                                INDOW_FULLSCREEN)
                                cv2.resizeWindow("mask",720,400)
                                cv2.moveWindow("mask", 320,220)

                                if cv2.waitKey(25) & 0xFF == ord('q'):
                                        break

                        else:
                                break

                # When everything done, release
                cap.release()

                # Closes all the frames
                cv2.destroyAllWindows()
                self.setWindowIcon(QtGui.QIcon('icons/windowLogo.png'))
                self.title = 'Sign language Recognition'
                uic.loadUi('UI_Files/dash.ui', self)
                self.setWindowTitle(self.title)
                self.timer = QTimer()
                self.create.clicked.connect(self.createGest)
```

```
        self.exp2.clicked.connect(self.exportFile)
        self.scan_sen.clicked.connect(self.scanSent)
        if(self.scan_sinlge.clicked.connect(self.scanSingle)==True):
                self.timer.timeout.connect(self.scanSingle)
        self.create.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.scan_sen.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.scan_sinlge.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.exp2.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.exit_button.clicked.connect(self.quitApplication)
        self._layout = self.layout()
        self.label_3 = QtWidgets.QLabel()
        movie = QtGui.QMovie("icons/dashAnimation.gif")
        self.label_3.setMovie(movie)
        self.label_3.setGeometry(0,160,780,441)
        movie.start()
        self._layout.addWidget(self.label_3)
        self.setObjectName('Message_Window')

def quitApplication(self):


        """shutsdown the GUI window along with removal of files"""
        userReply = QMessageBox.question(self, 'Quit Application', "Are you sure
        you want to quit this app?", QMessageBox.Yes | QMessageBox.No,
        QMessageBox.No)
        if userReply == QMessageBox.Yes:
                removeFile()
                keyboard.press_and_release('alt+F4')

def createGest(self):


        """ Custom gesture generation module"""
        try:
                clearfunc(self.cam)
        except:
                pass
        gesname=""
        uic.loadUi('UI_Files/create_gest.ui', self)
        self.setWindowTitle(self.title)
        self.create.clicked.connect(self.createGest)
        self.exp2.clicked.connect(self.exportFile)
        if(self.scan_sen.clicked.connect(self.scanSent)):
                controlTimer(self)
        self.scan_sinlge.clicked.connect(self.scanSingle)
        self.linkButton.clicked.connect(openimg)
        self.create.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.scan_sen.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.scan_sinlge.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.exp2.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.pushButton_2.clicked.connect(lambda:clearfunc(self.cam))
        try:
                self.exit_button.clicked.connect(lambda:clearfunc(self.cam))
        except:
                pass
        self.exit_button.clicked.connect(self.quitApplication)
        self.plainTextEdit.setPlaceholderText("Enter Gesture Name Here")
        img_text = ''
        saveimg=[]
        while True:
                ret, frame = self.cam.read()
                frame = cv2.flip(frame,1)
                try:
                        frame=cv2.resize(frame,(321,270))
                        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
                        img2 = cv2.rectangle(frame, (150,50),(300,200), (0,255,0),
                        thickness=2, lineType=8, shift=0)
                except:
                        keyboard.press_and_release('esc')

                height2, width2, channel2 = img2.shape
                step2 = channel2 * width2
                # create QImage from image
                qImg2 = QImage(img2.data, width2, height2, step2,
                QImage.Format_RGB888)
                # show image in img_label
                try:
```

```
                        self.label_3.setPixmap(QPixmap.fromImage(qImg2))
                        slider2=self.trackbar.value()
                except:
                        pass

                lower_blue = np.array([0, 0, 0])
                upper_blue = np.array([179, 255, slider2])
                imcrop = img2[52:198, 152:298]
                hsv = cv2.cvtColor(imcrop, cv2.COLOR_BGR2HSV)
                mask = cv2.inRange(hsv, lower_blue, upper_blue)

                cv2.namedWindow("mask", cv2.WINDOW_NORMAL )
                cv2.imshow("mask", mask)

                cv2.setWindowProperty("mask",cv2.WND_PROP_FULLSCREEN,cv2.WINDOW_F
                ULLSCREEN)
                cv2.resizeWindow("mask",170,160)
                cv2.moveWindow("mask", 766,271)

                hwnd = winGuiAuto.findTopWindow("mask")
                win32gui.SetWindowPos(hwnd, win32con.HWND_TOP,
                0,0,0,0,win32con.SWP_NOMOVE | win32con.SWP_NOSIZE |
                win32con.SWP_NOACTIVATE)

                try:
                        ges_name = self.plainTextEdit.toPlainText()
                except:
                        pass
                if(len(ges_name)>=1):
                        saveimg.append(ges_name)
                else:
                        saveimg.append(ges_name)
                        ges_name=''

                try:
                        self.pushButton.clicked.connect(lambda:capture_images(self
                        ,self.cam,saveimg,mask))
                except:
                        pass

                gesname=saveimg[-1]

                if keyboard.is_pressed('shift+s'):
                        if not os.path.exists('./SampleGestures'):
                                os.mkdir('./SampleGestures')
                        if(len(gesname)>=1):
                                img_name =
                                "./SampleGestures/"+"{}.png".format(str(gesname))
                                save_img = cv2.resize(mask, (image_x, image_y))
                                cv2.imwrite(img_name, save_img)
                        break

                if cv2.waitKey(1) == 27:
                        break

        self.cam.release()
        cv2.destroyAllWindows()

        if os.path.exists("./SampleGestures/"+str(gesname)+".png"):
                QtWidgets.QMessageBox.about(self, "Success", "Gesture Saved
                Successfully!")

def exportFile(self):


        """export file module with tts assistance and gesturre viewer"""
        try:
                clearfunc2(self.cam)
        except:
                pass
        uic.loadUi('UI_Files/export.ui', self)
        self.setWindowTitle(self.title)
        self.create.clicked.connect(self.createGest)
        self.exp2.clicked.connect(self.exportFile)
        self.scan_sen.clicked.connect(self.scanSent)
        self.scan_sinlge.clicked.connect(self.scanSingle)
        self.create.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
```

```
        self.scan_sen.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.scan_sinlge.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.exp2.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.exit_button.clicked.connect(self.quitApplication)
        content=checkFile()
        self.textBrowser_98.setText("          "+content)
        engine.say(str(content).lower())
        try:
                engine.runAndWait()
        except:
                pass
        if(content=="File Not Found"):
                self.pushButton_2.setEnabled(False)
                self.pushButton_3.setEnabled(False)
        else:
                self.pushButton_2.clicked.connect(self.on_click)
                try:
                        self.pushButton_3.clicked.connect(self.gestureViewer)
                except:
                        pass

def on_click(self):


        """Opens tkinter window to save file at desired location """
        content=checkFile()
        root=Tk()
        root.withdraw()
        root.filename =  filedialog.asksaveasfilename(initialdir = "/",title =
        "Select file",filetypes = (("Text files","*.txt"),("all files","*.*")))
        name=root.filename
        #fr.close()
        fw=open(name+".txt","w")
        if(content=='No Content Available'):
                content=" "
        fw.write(content)
        try:
                os.remove("temp.txt")
                shutil.rmtree("TempGest")
        except:
                QtWidgets.QMessageBox.about(self, "Information", "Nothing to
                export")
        fw.close()
        root.destroy()

        if not os.path.exists('temp.txt'):
                if os.path.exists('.txt'):
                        os.remove('.txt')
                else:
                        QtWidgets.QMessageBox.about(self, "Information", "File
                        saved successfully!")
                        self.textBrowser_98.setText("          ")

def gestureViewer(self):


        """gesture viewer through matplotlib """
        try:
                img=load_images_from_folder('TempGest/')
                plt.imshow(img[index])
        except:
                plt.text(0.5, 0.5, 'No new Gesture Available',
        horizontalalignment='center',verticalalignment='center')
        axcut = plt.axes([0.9, 0.0, 0.1, 0.075])
        axcut1 = plt.axes([0.0, 0.0, 0.1, 0.075])
        bcut = Button(axcut, 'Next', color='dodgerblue', hovercolor='lightgreen')
        bcut1 = Button(axcut1, 'Previous', color='dodgerblue',
        hovercolor='lightgreen')

        #plt.connect('button_press_event', toggle_imagesfwd)
        bcut.on_clicked(toggle_imagesfwd)
        bcut1.on_clicked(toggle_imagesrev)
        plt.show()
        axcut._button = bcut          #creating a reference for that element
        axcut1._button1 = bcut1
```

```
def scanSent(self):

        """sentence formation module """
        try:
                clearfunc(self.cam)
        except:
                pass
        uic.loadUi('UI_Files/scan_sent.ui', self)
        self.setWindowTitle(self.title)
        self.create.clicked.connect(self.createGest)
        self.exp2.clicked.connect(self.exportFile)
        if(self.scan_sen.clicked.connect(self.scanSent)):
                controlTimer(self)
        self.scan_sinlge.clicked.connect(self.scanSingle)
        try:
                self.pushButton_2.clicked.connect(lambda:clearfunc(self.cam))
        except:
                pass
        self.linkButton.clicked.connect(openimg)
        self.create.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.scan_sen.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.scan_sinlge.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.exp2.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        try:
                self.exit_button.clicked.connect(lambda:clearfunc(self.cam))
        except:
                pass
        self.exit_button.clicked.connect(self.quitApplication)
        img_text = ''
        append_text=''
        new_text=''
        finalBuffer=[]
        counts=0
        while True:
                ret, frame =self.cam.read()
                frame = cv2.flip(frame,1)
                try:
                        frame=cv2.resize(frame,(331,310))


                        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
                        img = cv2.rectangle(frame, (150,50),(300,200), (0,255,0),
                        thickness=2, lineType=8, shift=0)
                except:
                        keyboard.press_and_release('esc')
                        keyboard.press_and_release('esc')

                height, width, channel = img.shape
                step = channel * width
                # create QImage from image
                qImg = QImage(img.data, width, height, step,
                QImage.Format_RGB888)
                # show image in img_label
                try:
                        self.label_3.setPixmap(QPixmap.fromImage(qImg))
                        slider=self.trackbar.value()
                except:
                        pass

                lower_blue = np.array([0, 0, 0])
                upper_blue = np.array([179, 255, slider])
                imcrop = img[52:198, 152:298]
                hsv = cv2.cvtColor(imcrop, cv2.COLOR_BGR2HSV)
                mask1 = cv2.inRange(hsv, lower_blue, upper_blue)


                cv2.namedWindow("mask", cv2.WINDOW_NORMAL )
                cv2.imshow("mask", mask1)

                cv2.setWindowProperty("mask",cv2.WND_PROP_FULLSCREEN,cv2.WINDOW_F
                ULLSCREEN)
                cv2.resizeWindow("mask",118,108)
                cv2.moveWindow("mask", 905,271)

                hwnd = winGuiAuto.findTopWindow("mask")
```

```
                win32gui.SetWindowPos(hwnd, win32con.HWND_TOP,
                0,0,0,0,win32con.SWP_NOMOVE | win32con.SWP_NOSIZE |
                win32con.SWP_NOACTIVATE)

                try:
                        self.textBrowser.setText("\n         "+str(img_text))
                except:
                        pass
                img_name = "1.png"
                save_img = cv2.resize(mask1, (image_x, image_y))
                cv2.imwrite(img_name, save_img)
                img_text=predictor()
                if cv2.waitKey(1) == ord('c'):
                                try:
                                        counts+=1
                                        append_text+=img_text
                                        new_text+=img_text
                                        if not os.path.exists('./TempGest'):
                                                os.mkdir('./TempGest')
                                        img_names =
                                        "./TempGest/"+"{}{}.png".format(str(counts)
                                        ,str(img_text))
                                        save_imgs = cv2.resize(mask1, (image_x,
                                        image_y))
                                        cv2.imwrite(img_names, save_imgs)
                                        self.textBrowser_4.setText(new_text)
                                except:
                                        append_text+=''

                                if(len(append_text)>1):
                                        finalBuffer.append(append_text)
                                        append_text=''
                                else:
                                        finalBuffer.append(append_text)
                                        append_text=''

                try:
                        self.pushButton.clicked.connect(lambda:saveBuff(self,self.
                        cam,finalBuffer))
                except:
                        pass
                if cv2.waitKey(1) == 27:
                        break

                if keyboard.is_pressed('shift+s'):
                        if(len(finalBuffer)>=1):
                                f=open("temp.txt","w")
                                for i in finalBuffer:
                                        f.write(i)
                                f.close()
                        break


        self.cam.release()
        cv2.destroyAllWindows()

        if os.path.exists('temp.txt'):
                QtWidgets.QMessageBox.about(self, "Information", "File is
                temporarily saved ... you can now proceed to export")
        try:
                self.textBrowser.setText("            ")
        except:
                pass

def scanSingle(self):


        """Single gesture scanner """
        try:
                clearfunc(self.cam)
        except:
                pass
        uic.loadUi('UI_Files/scan_single.ui', self)
        self.setWindowTitle(self.title)
        self.create.clicked.connect(self.createGest)
        self.exp2.clicked.connect(self.exportFile)
        self.scan_sen.clicked.connect(self.scanSent)
```

```
                if(self.scan_sinlge.clicked.connect(self.scanSingle)):
                        controlTimer(self)
                self.pushButton_2.clicked.connect(lambda:clearfunc(self.cam))
                self.linkButton.clicked.connect(openimg)
                self.create.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
                self.scan_sen.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
                self.scan_sinlge.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
                self.exp2.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
                try:
                        self.exit_button.clicked.connect(lambda:clearfunc(self.cam))
                except:
                        pass
                self.exit_button.clicked.connect(self.quitApplication)
                img_text = ''
                while True:
                        ret, frame = self.cam.read()
                        frame = cv2.flip(frame,1)
                        try:
                                frame=cv2.resize(frame,(321,270))
                                frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
                                img1 = cv2.rectangle(frame, (150,50),(300,200), (0,255,0),
                                thickness=2, lineType=8, shift=0)
                        except:
                                keyboard.press_and_release('esc')

                        height1, width1, channel1 = img1.shape
                        step1 = channel1 * width1
                        # create QImage from image
                        qImg1 = QImage(img1.data, width1, height1, step1,
                        QImage.Format_RGB888)
                        # show image in img_label
                        try:
                                self.label_3.setPixmap(QPixmap.fromImage(qImg1))
                                slider1=self.trackbar.value()
                        except:
                                pass

                        lower_blue = np.array([0, 0, 0])
                        upper_blue = np.array([179, 255, slider1])

                        imcrop = img1[52:198, 152:298]
                        hsv = cv2.cvtColor(imcrop, cv2.COLOR_BGR2HSV)
                        mask = cv2.inRange(hsv, lower_blue, upper_blue)

                        cv2.namedWindow("mask", cv2.WINDOW_NORMAL )
                        cv2.imshow("mask", mask)

                        cv2.setWindowProperty("mask",cv2.WND_PROP_FULLSCREEN,cv2.WINDOW_F
                        ULLSCREEN)
                        cv2.resizeWindow("mask",118,108)
                        cv2.moveWindow("mask", 894,271)

                        hwnd = winGuiAuto.findTopWindow("mask")
                        win32gui.SetWindowPos(hwnd, win32con.HWND_TOP,
                        0,0,0,0,win32con.SWP_NOMOVE | win32con.SWP_NOSIZE |
                        win32con.SWP_NOACTIVATE)

                        try:
                                self.textBrowser.setText("\n\n\t"+str(img_text))
                        except:
                                pass

                        img_name = "1.png"
                        save_img = cv2.resize(mask, (image_x, image_y))
                        cv2.imwrite(img_name, save_img)
                        img_text = predictor()

                        if cv2.waitKey(1) == 27:
                            break

                self.cam.release()
                cv2.destroyAllWindows()

app = QtWidgets.QApplication([])
win = Dashboard()
win.show()
sys.exit(app.exec())
```

4.2 SCREENSHOTS

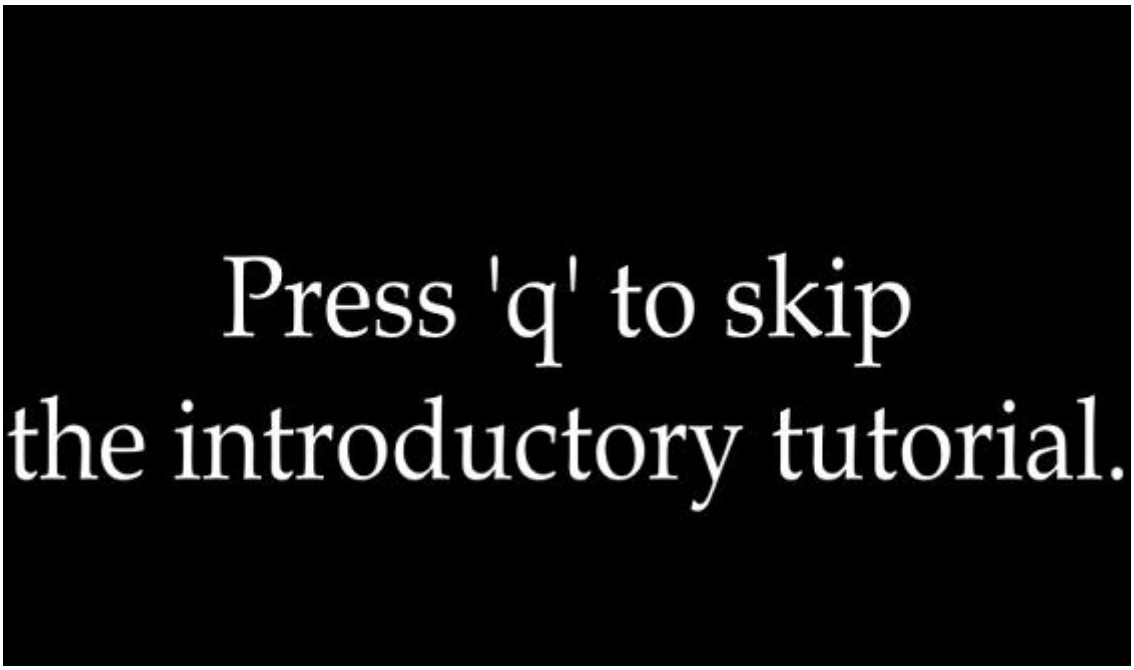

Fig 4.2.1: Skip Video.

Fig 4.2.2: Dashboard with Sample Gesture Animation.

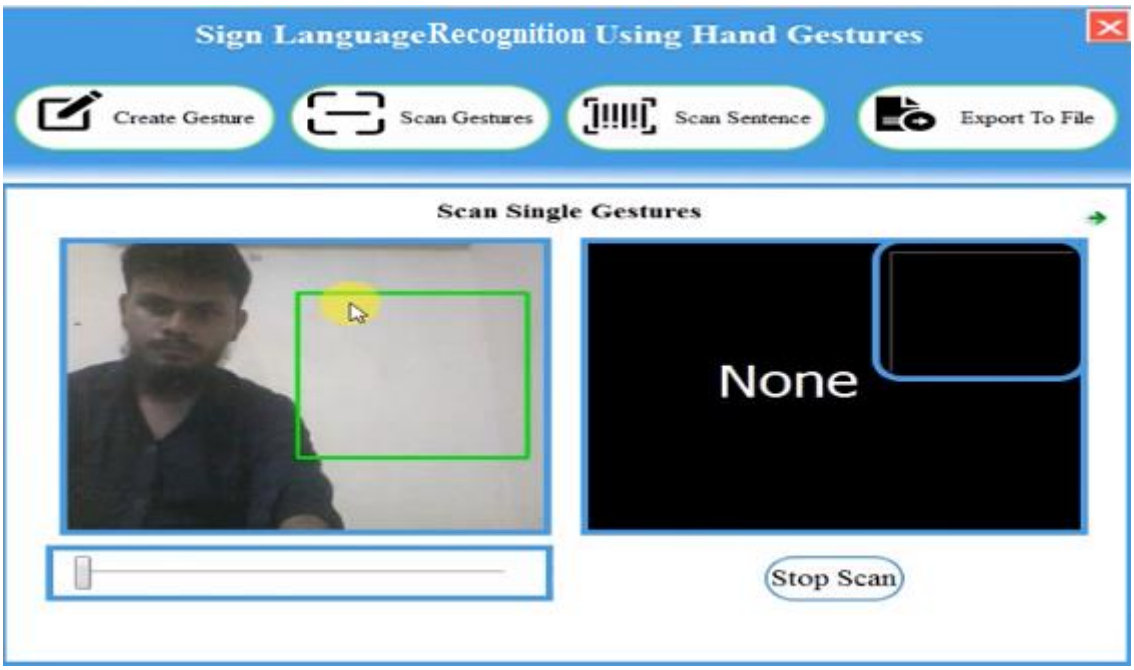

Fig 4.2.3: Single Gesture.

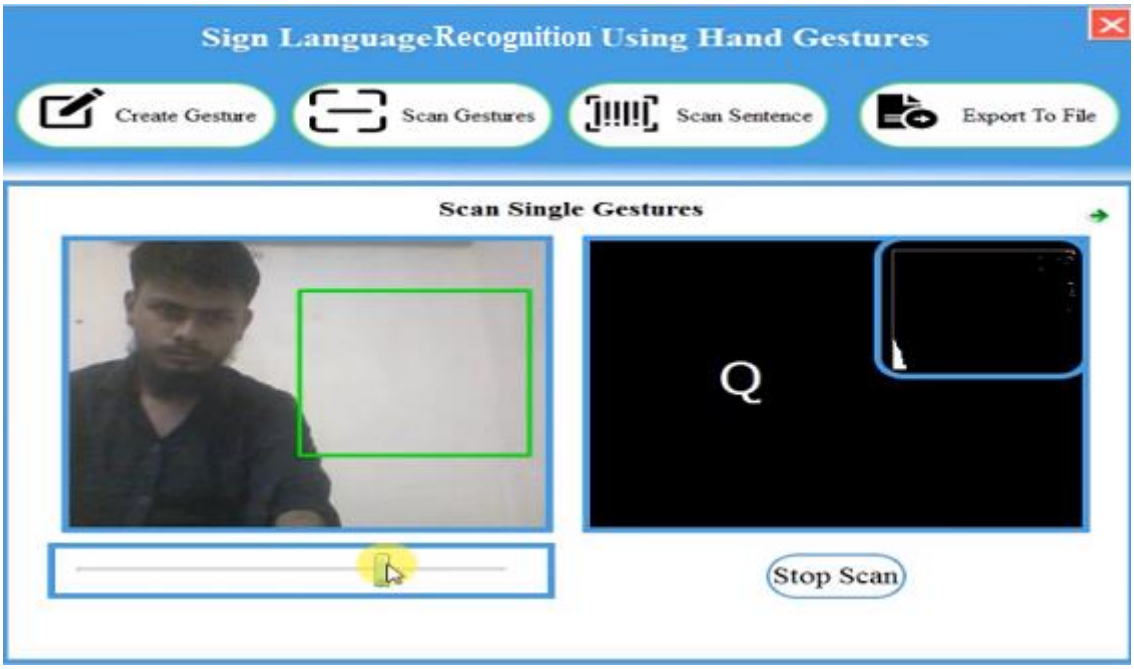

Fig 4.2.4: Adjusting Camera Light as Needed.

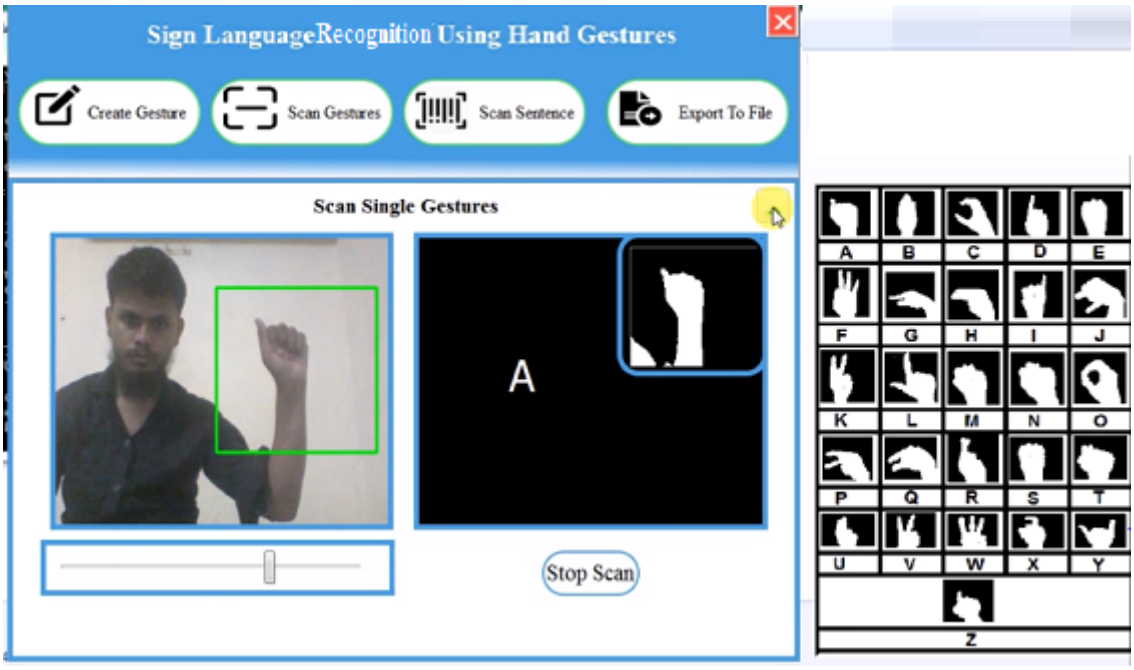

Fig 4.2.5: Placing Hand Inside Rectangle Box.

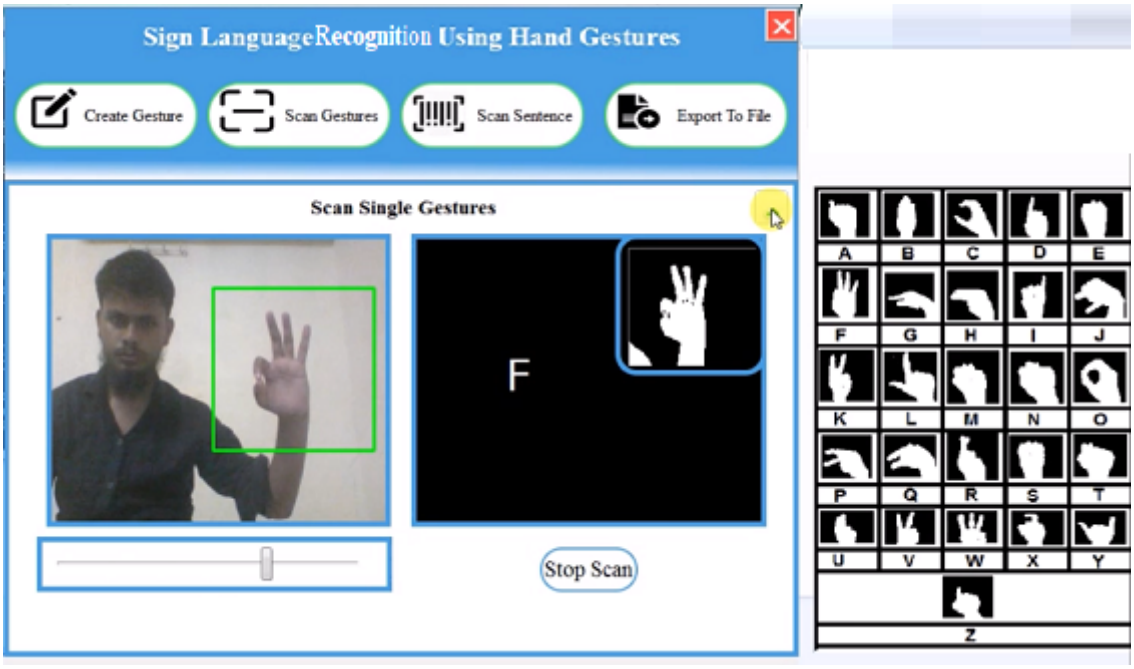

Fig 4.2.6: Single Gesture Output.

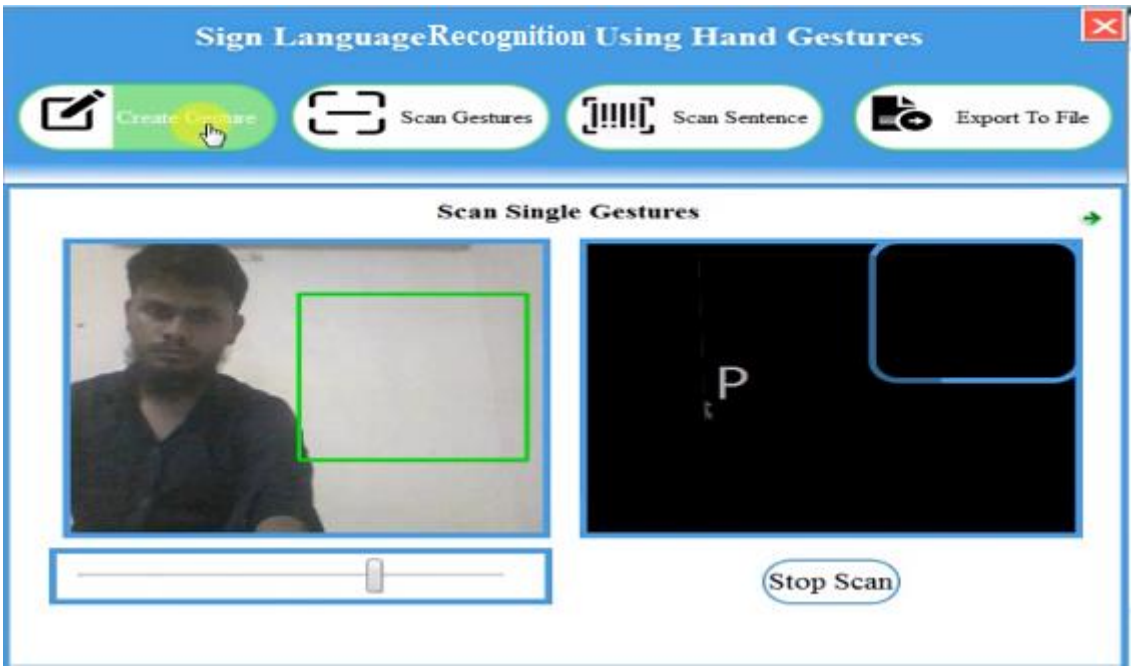

Fig 4.2.7: Custom Gesture Generation.

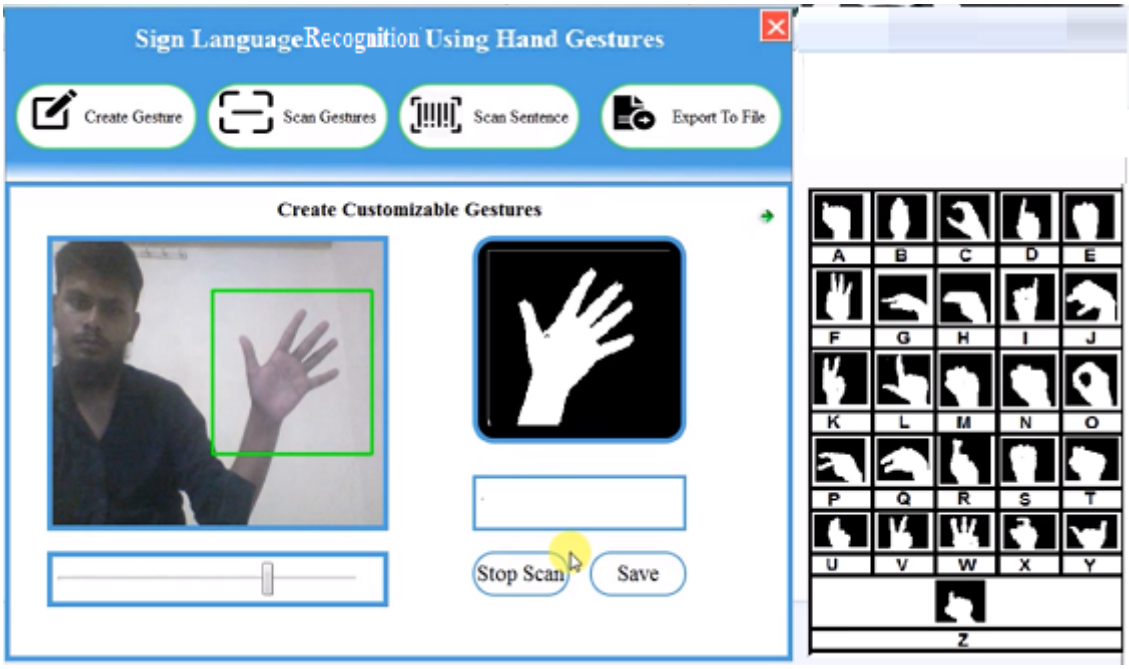

Fig 4.2.8: Assigning Label to Gesture.

Fig 4.2.9: Gesture Saved Successfully.

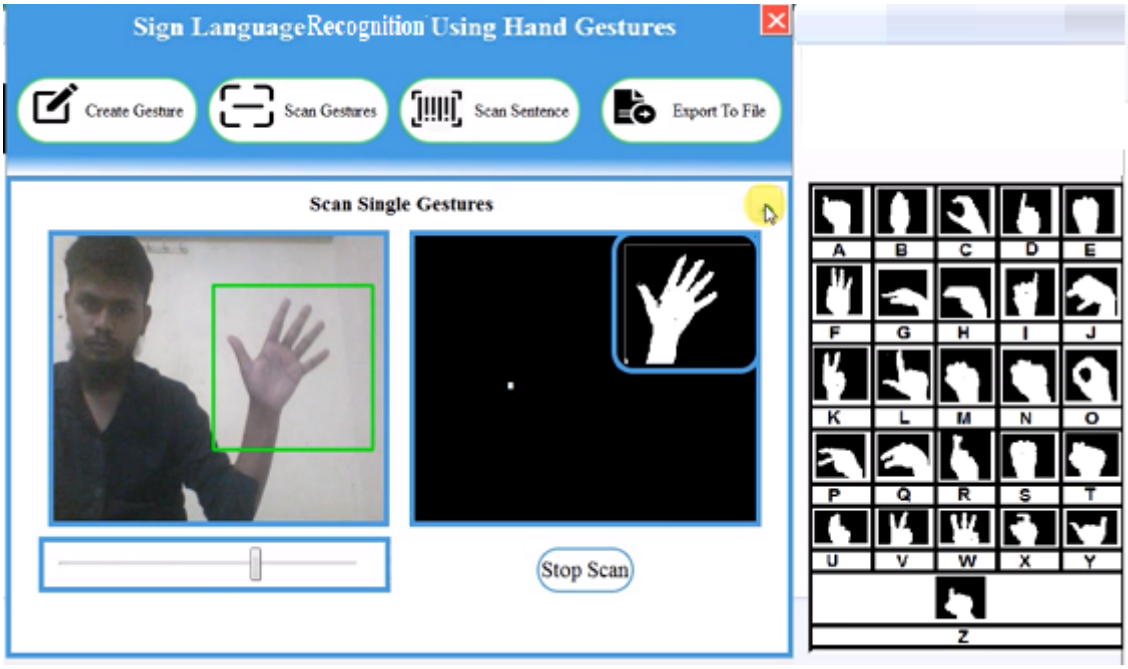

Fig 4.2.10: Scanning & Testing Newly Generated Gesture.

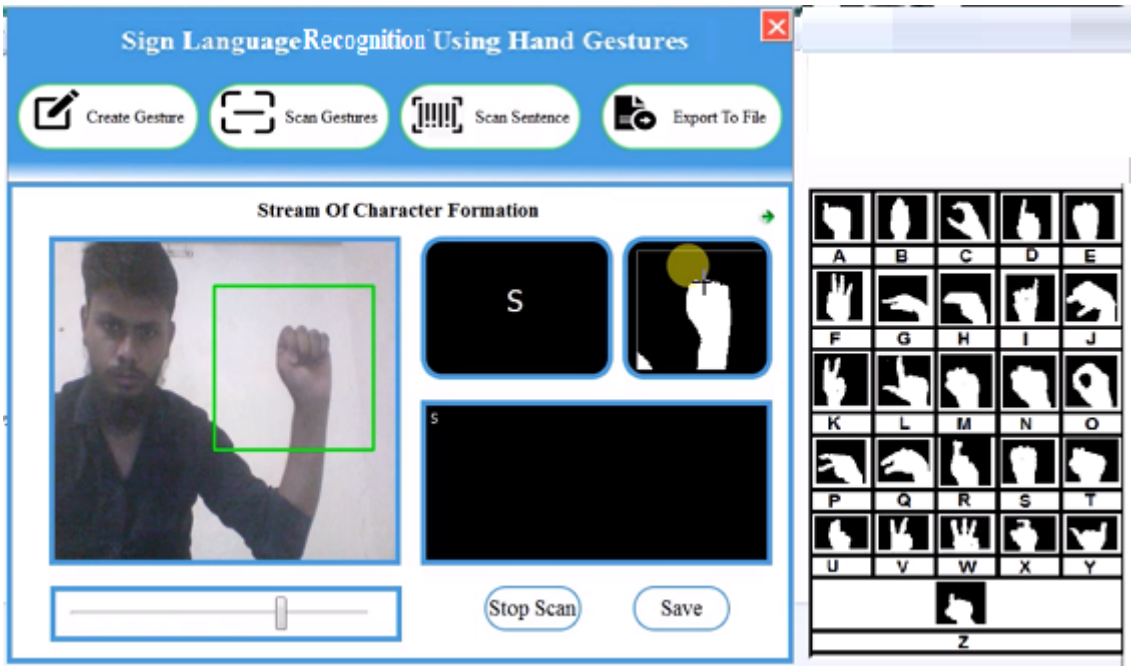

Fig 4.2.11: Sentence Formation, Focusing on The Top Right Window Pressing C to Form Sentence.

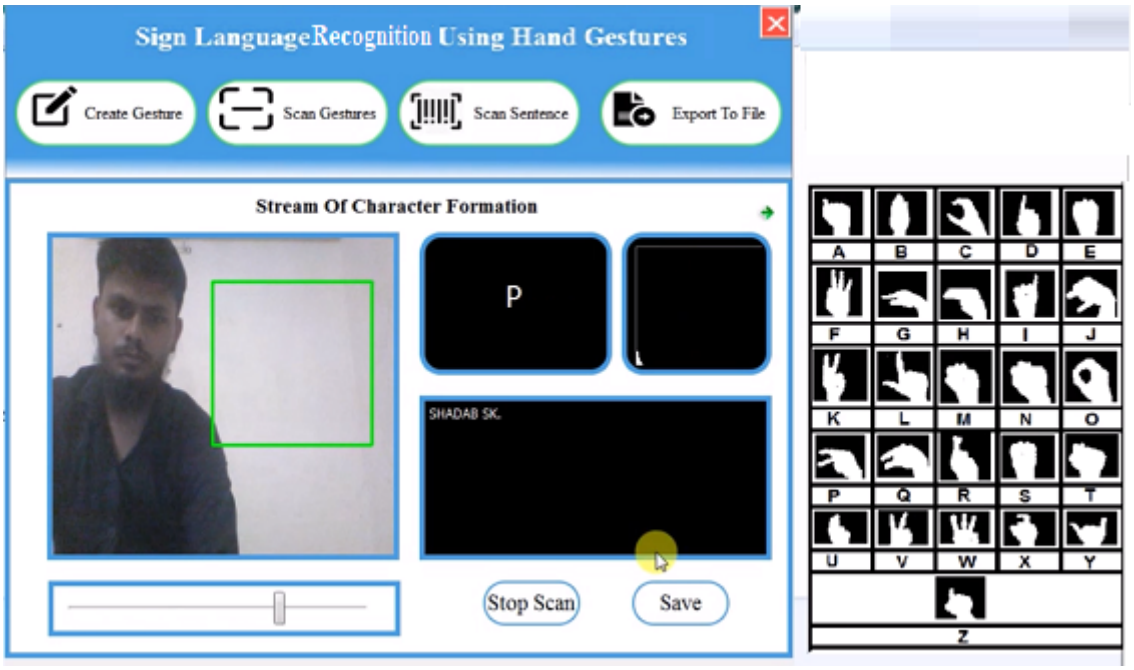

Fig 4.2.12: Sentence Formed.

Fig 4.2.13: Sentence Saved Temporarily.

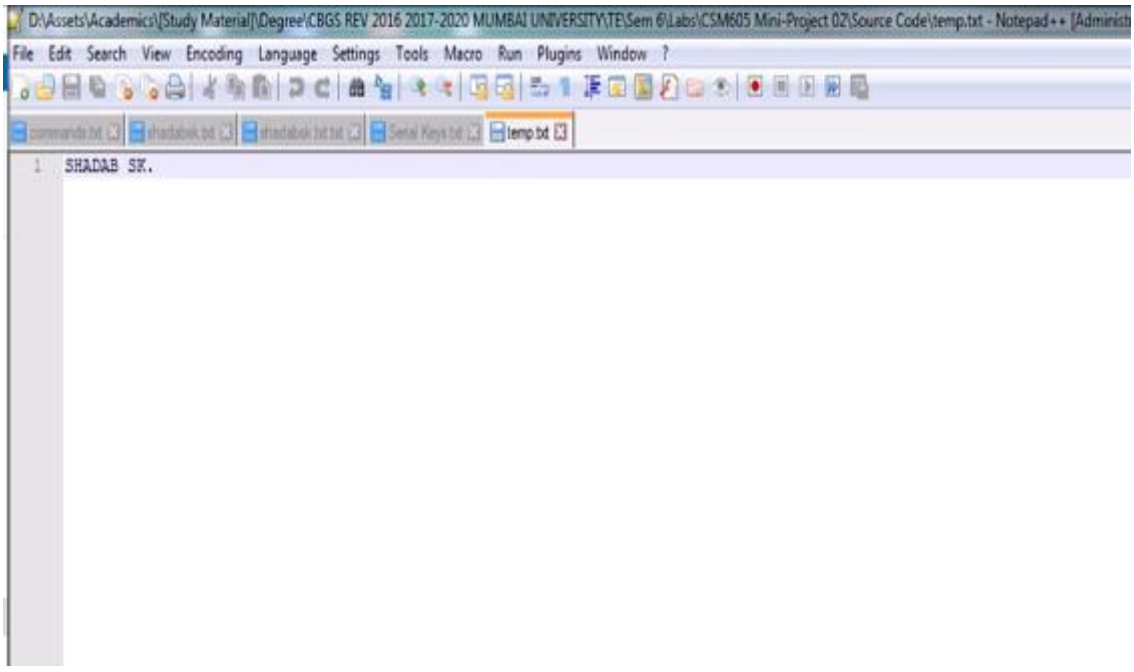

Fig 4.2.14: Content of Newly Generated temp.txt.

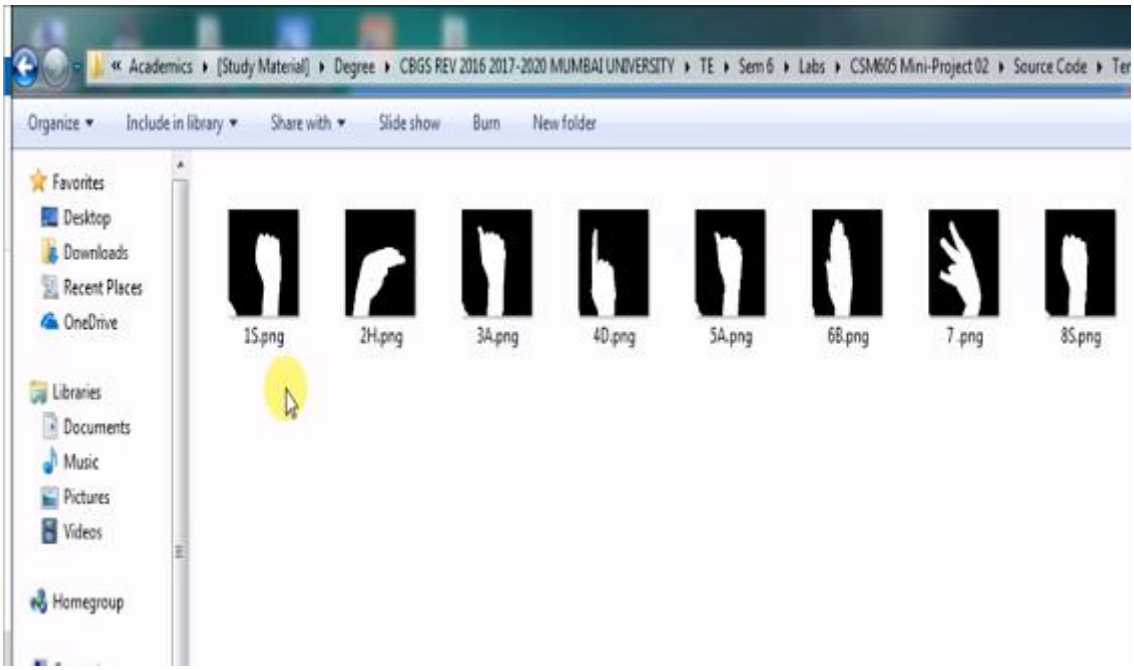

Fig 4.2.15: Content of Newly Generated TempGest Directory.

Fig 4.2.16: Export with TTS Assistance and Gesture Viewer.

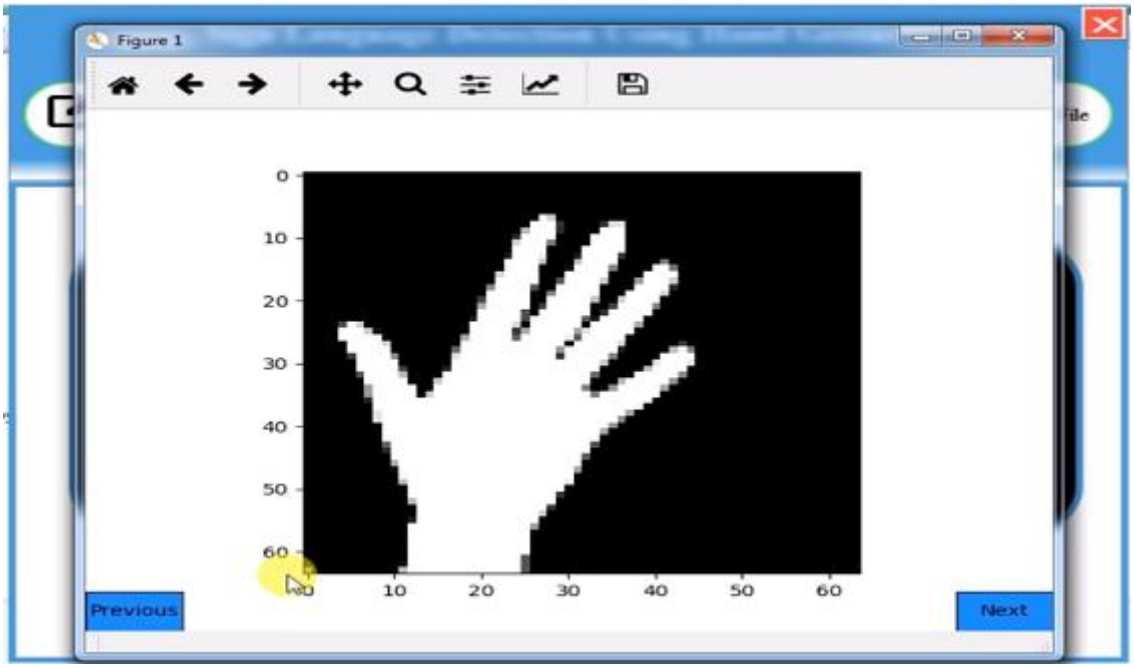

Fig 4.2.17: Gesture Viewer Sample.

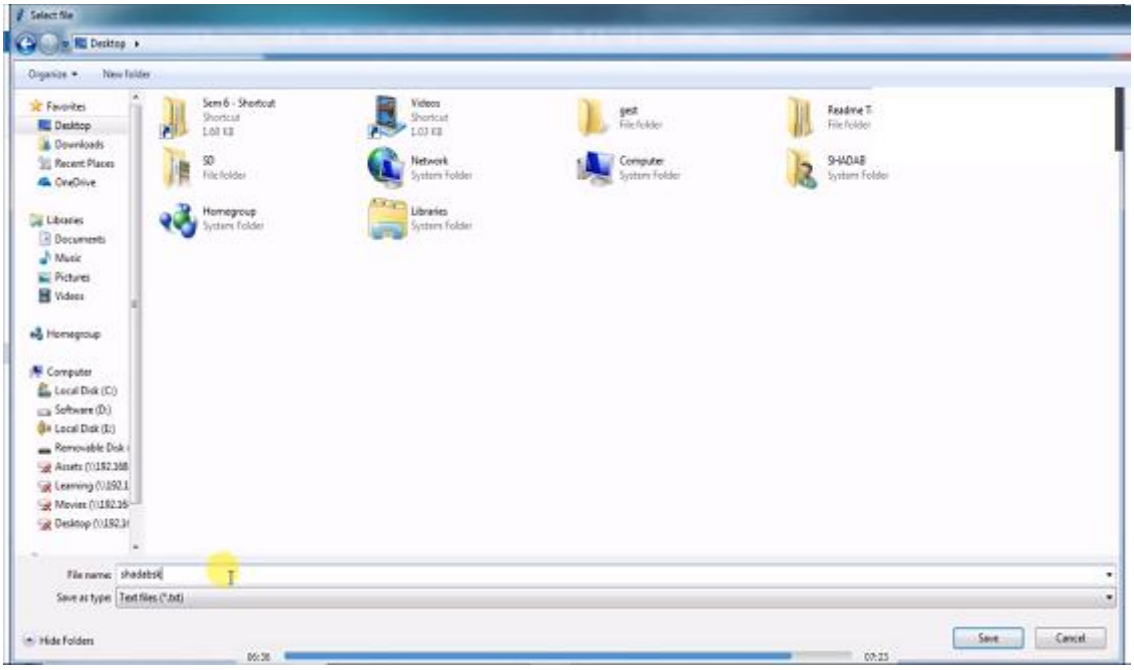

Fig 4.2.18: Exporting the File at Desired Location.

Fig 4.2.19: File Saved Successfully.

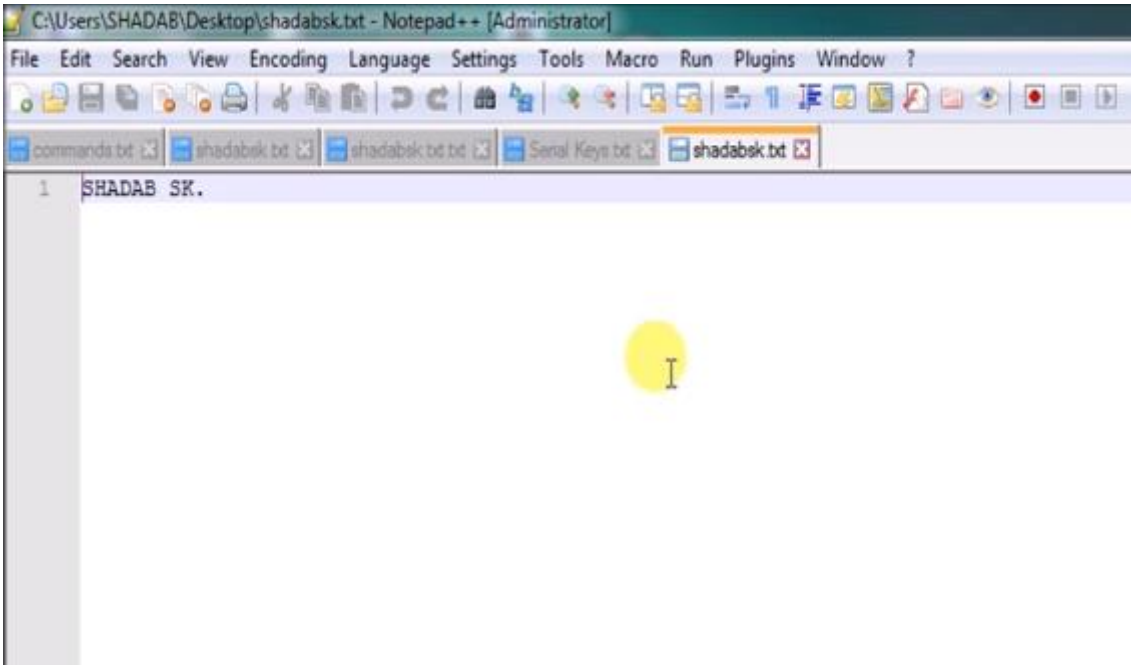

Fig 4.2.20: Content of Newly Saved File.

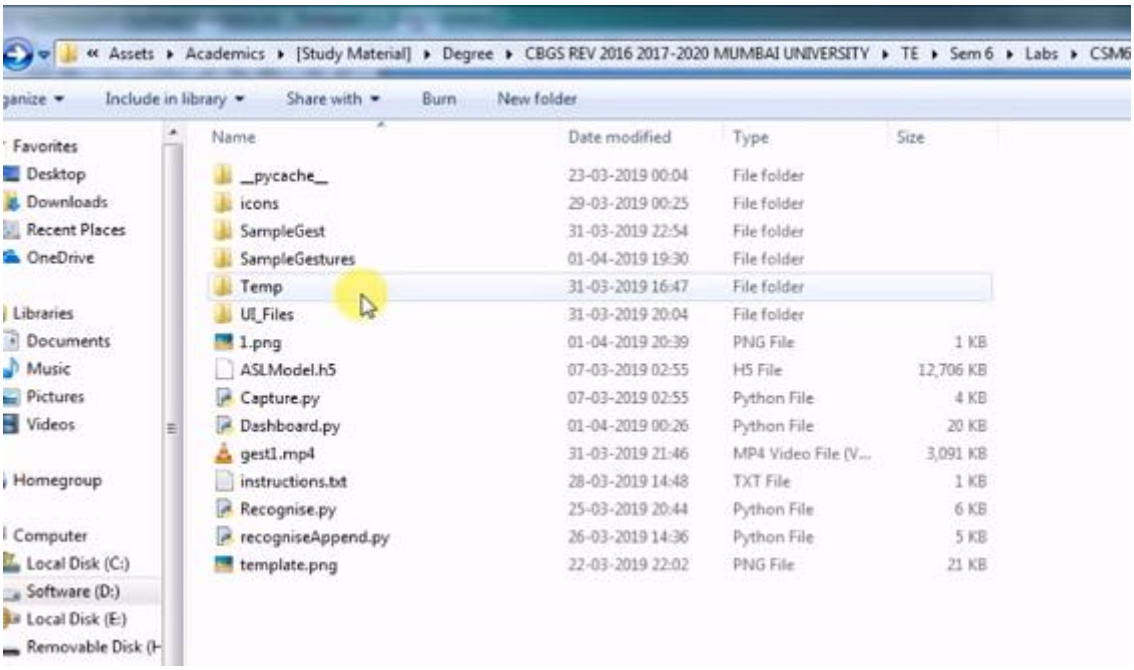

Fig 4.2.21: Temporary Files Deleted.

Fig 4.2.22: No Content Available.

Fig 4.2.23: Quitting the Application.

CHAPTER 5

CONCLUSION

5.1 CONCLUSION

From this project/application we have tried to overshadow some of the major problems faced by the disabled persons in terms of talking. We found out the root cause of why they can't express more freely. The result that we got was the other side of the audience are not able to interpret what these persons are trying to say or what is the message that they want to convey.

Thereby this application serves the person who wants to learn and talk in sign languages. With this application a person will quickly adapt various gestures and their meaning as per ASL standards. They can quickly learn what alphabet is assigned to which gesture. Add-on to this custom gesture facility is also provided along with sentence formation. A user need not be a literate person if they know the action of the gesture, they can quickly form the gesture and appropriate assigned character will be shown onto the screen.

Concerning to the implementation, we have used TensorFlow framework, with keras API. And for the user feasibility complete front-end is designed using PyQT5. Appropriate user-friendly messages are prompted as per the user actions along with what gesture means which character window. Additionally, an export to file module is also provided with TTS(Text-To-Speech) assistance meaning whatever the sentence was formed a user will be able to listen to it and then quickly export along with observing what gesture he/she made during the sentence formation.

5.2 FUTURE SCOPE

- It can be integrated with various search engines and texting application such as google, WhatsApp. So that even the illiterate people could be able to chat with other persons, or query something from web just with the help of gesture.
- This project is working on image currently, further development can lead to detecting the motion of video sequence and assigning it to a meaningful sentence with TTS assistance.

REFERENCES

REFERENCES

[1] Shobhit Agarwal, "What are some problems faced by deaf and dumb people while using todays common tech like phones and PCs", 2017 [Online]. Available: https://www.quora.com/What-are-some-problems-faced-by-deaf-and-dumb-people-while-using-todays-common-tech-like-phones-and-PCs, [Accessed April 06, 2019].

[2] NIDCD, "american sign language", 2017 [Online]. Available: https://www.nidcd.nih.gov/health/american-sign-language, [Accessed April 06, 2019].

[3] Suharjito MT, "Sign Language Recognition Application Systems for Deaf-Mute People A Review Based on Input-Process-Output", 2017 [Online]. Available: https://www.academia.edu/35314119/Sign_Language_Recognition_Application_Systems_for_Deaf-Mute_People_A_Review_Based_on_Input-Process-Output [Accessed April 06, 2019].

[4] M. Ibrahim, "Sign Language Translation via Image Processing", [Online]. Available: https://www.kics.edu.pk/project/startup/203 [Accessed April 06, 2019].

[5] NAD, "American sign language-community and culture frequently asked questions", 2017 [Online]. Available: https://www.nad.org/resources/american-sign-language/community-and-culture-frequently-asked-questions/ [Accessed April 06, 2019].

[6] Sanil Jain and K.V.Sameer Raja, "Indian Sign Language Character Recognition" , [Online]. Available: https://cse.iitk.ac.in/users/cs365/2015/_submissions/vinsam/report.pdf [Accessed April 06, 2019].

APPENDICES

APPENDIX A: MANAGING VERSIONS OF THE APP USING GITHUB SCM TOOL.

Repository URL - https://github.com/shadabsk/Sign-Language-Recognition-Using-Hand-Gestures-Keras-PyQT5-OpenCV

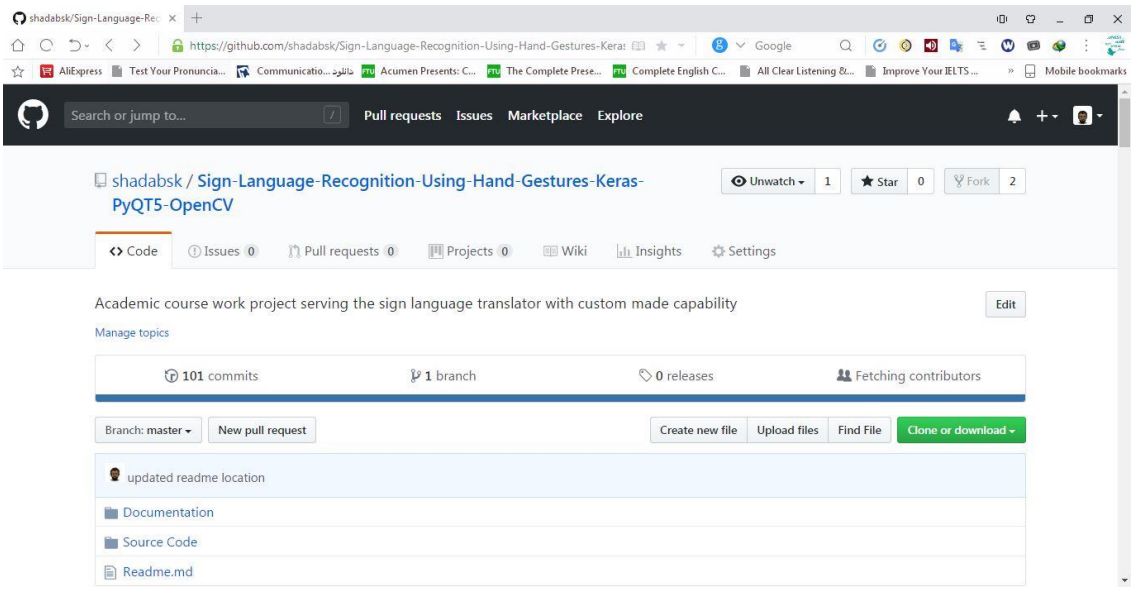

Fig 5.1: Repository overview.

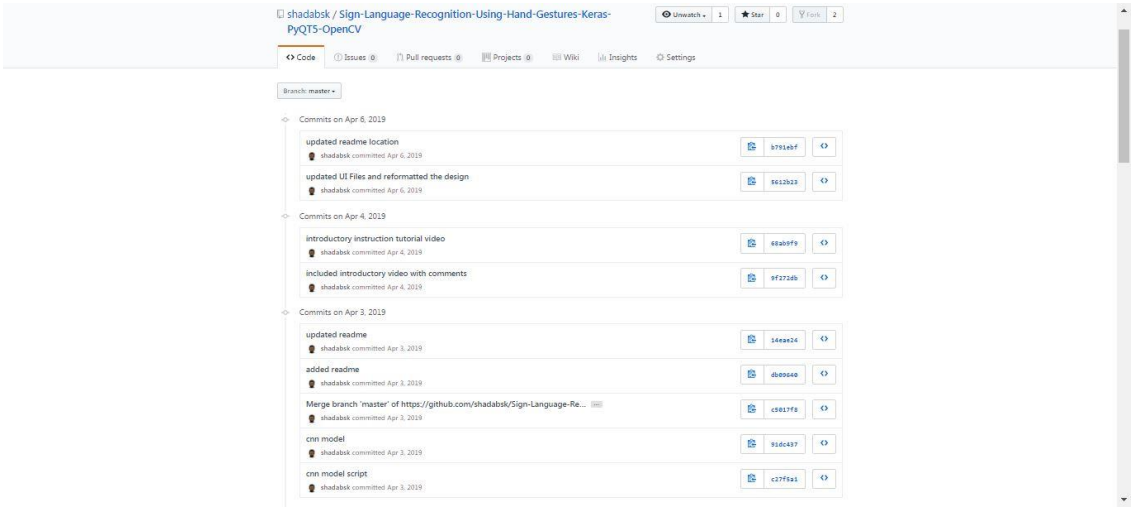

Fig 5.2: Master branch commit history.

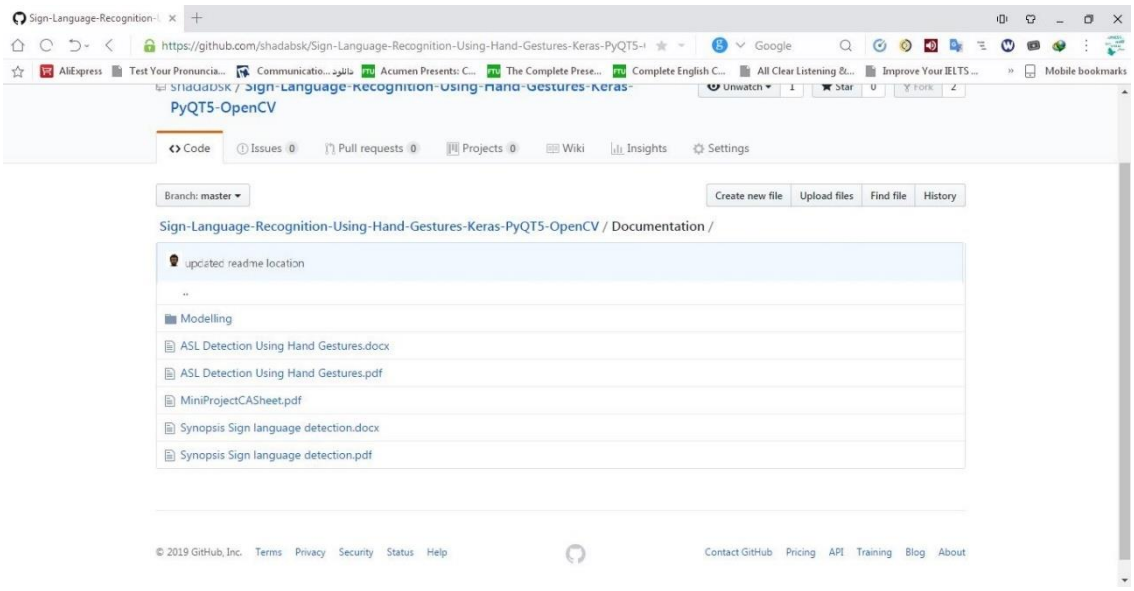

Fig 5.3: Documentation section.

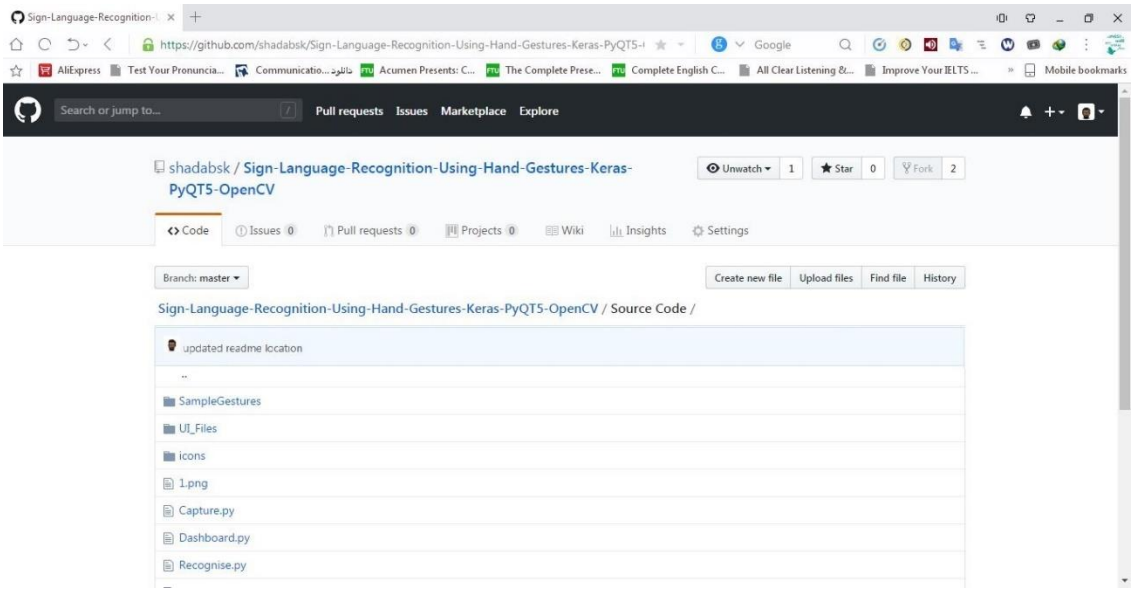

Fig 5.4: Source code section.

APPENDIX B: SCHEDULING THE PROJECT USING AGILE SCRUM METHODOLOGY WITH ONLINE TRELLO PLATFORM

Board URL - https://trello.com/b/Ch19Fm8W/sign-language-detection-using-hand-gestures

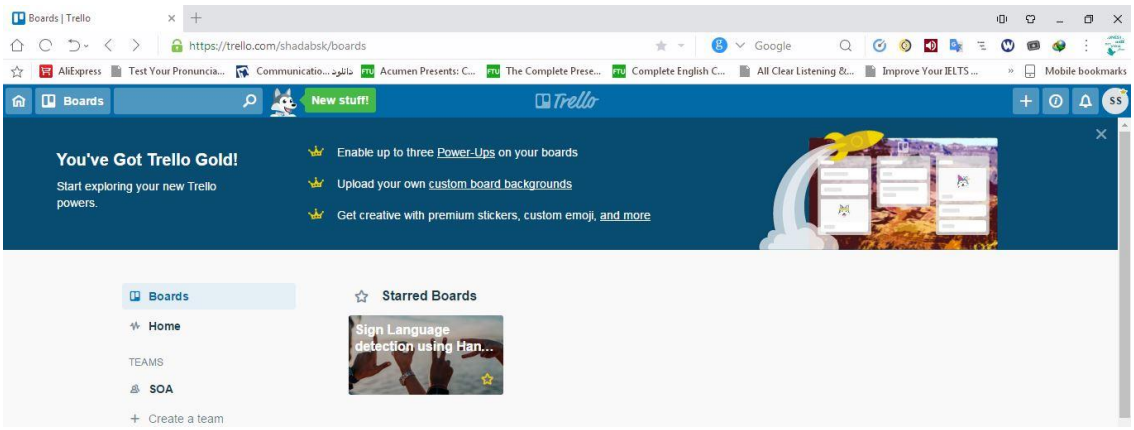

Fig 6.1: Billboard view.

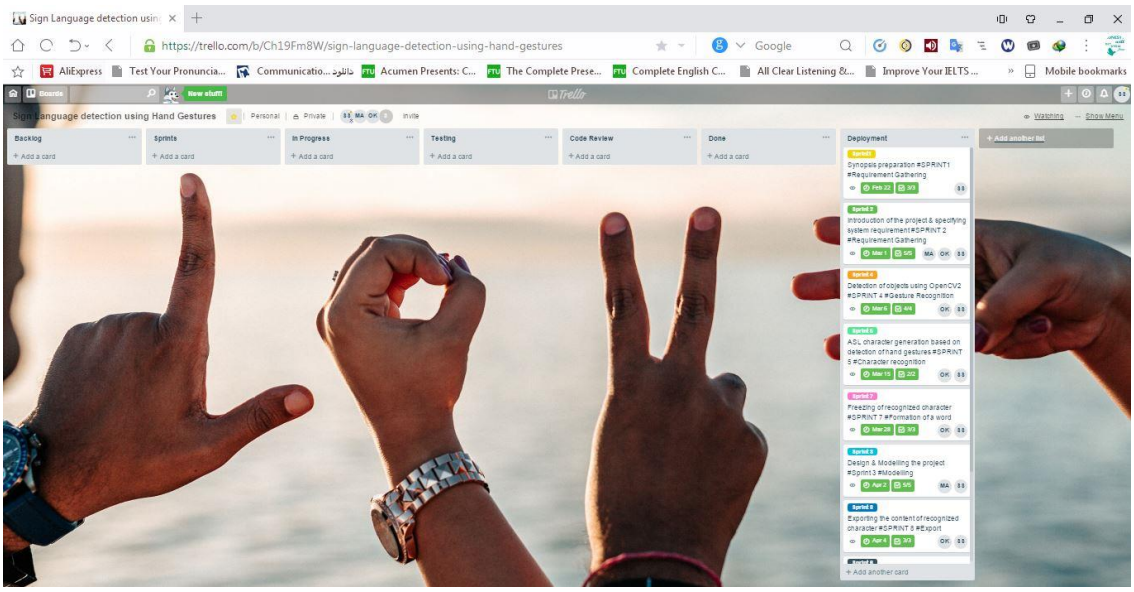

Fig 6.2: Inside view along with each phases of app.

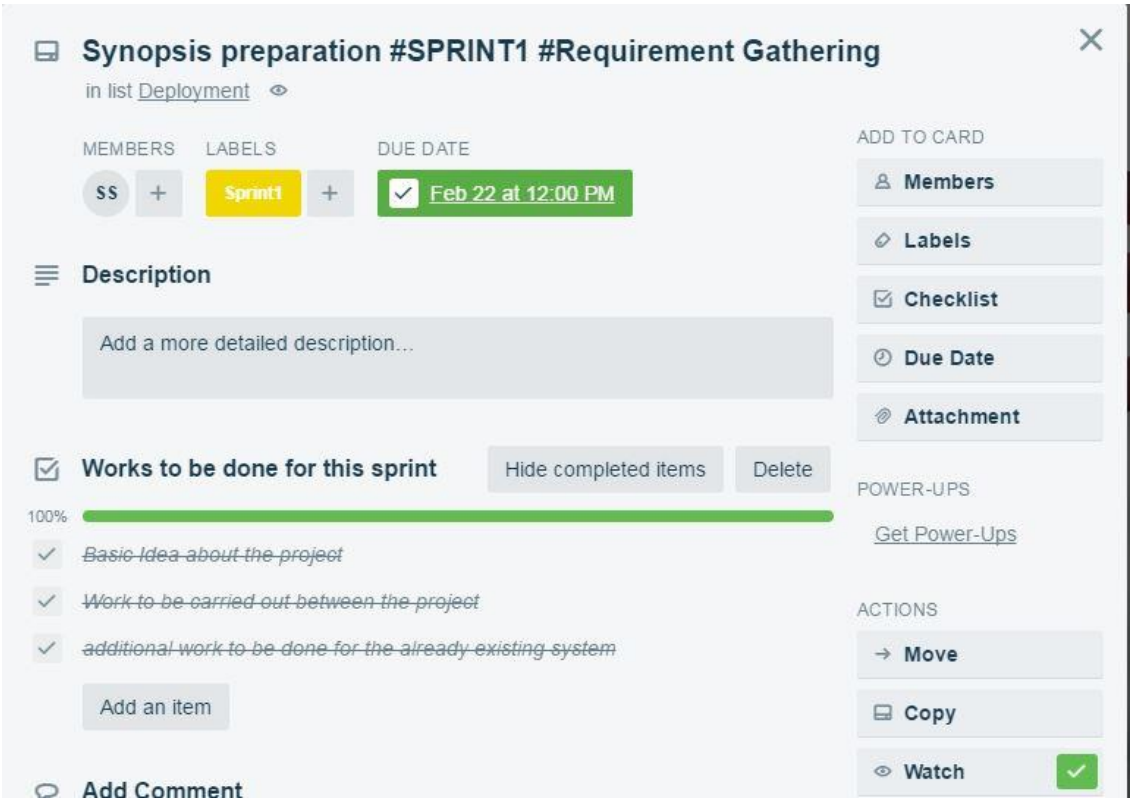

Fig 6.3: schedule and task of sprint 1 "synopsis preparation" with assigned members

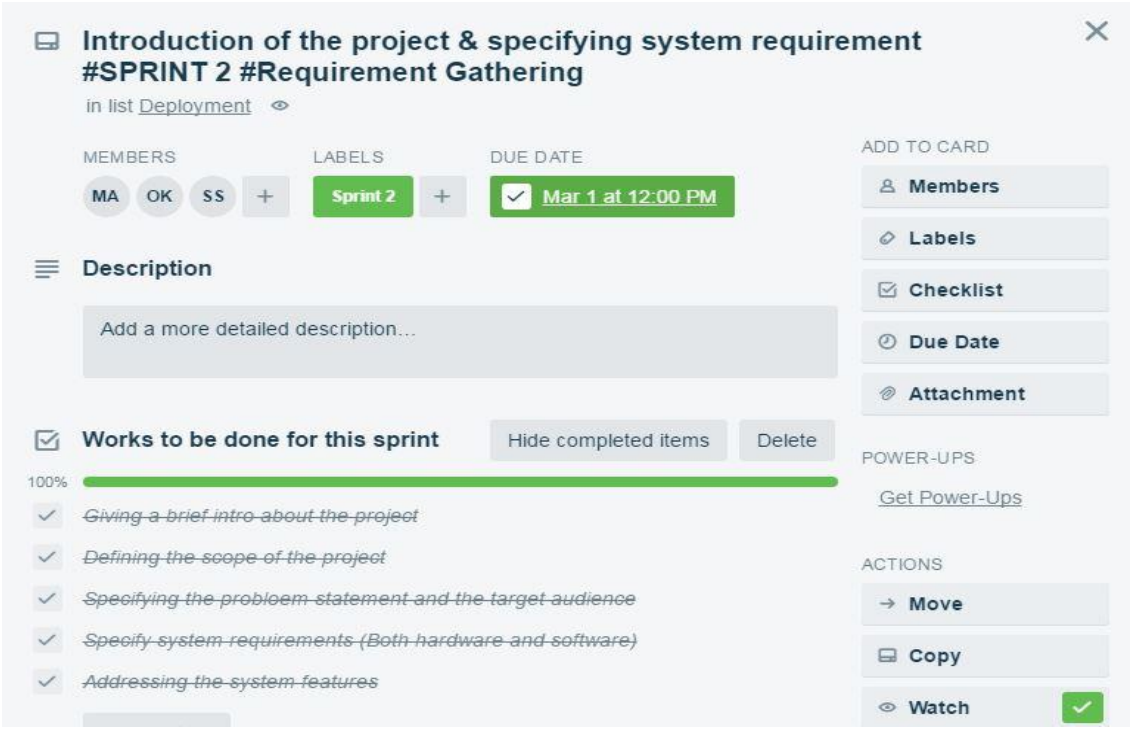

Fig 6.4: schedule and task of sprint 2 "requirement specification" with assigned members

Fig 6.5: schedule and task of sprint 4 "ASL char generation" along with assigned members

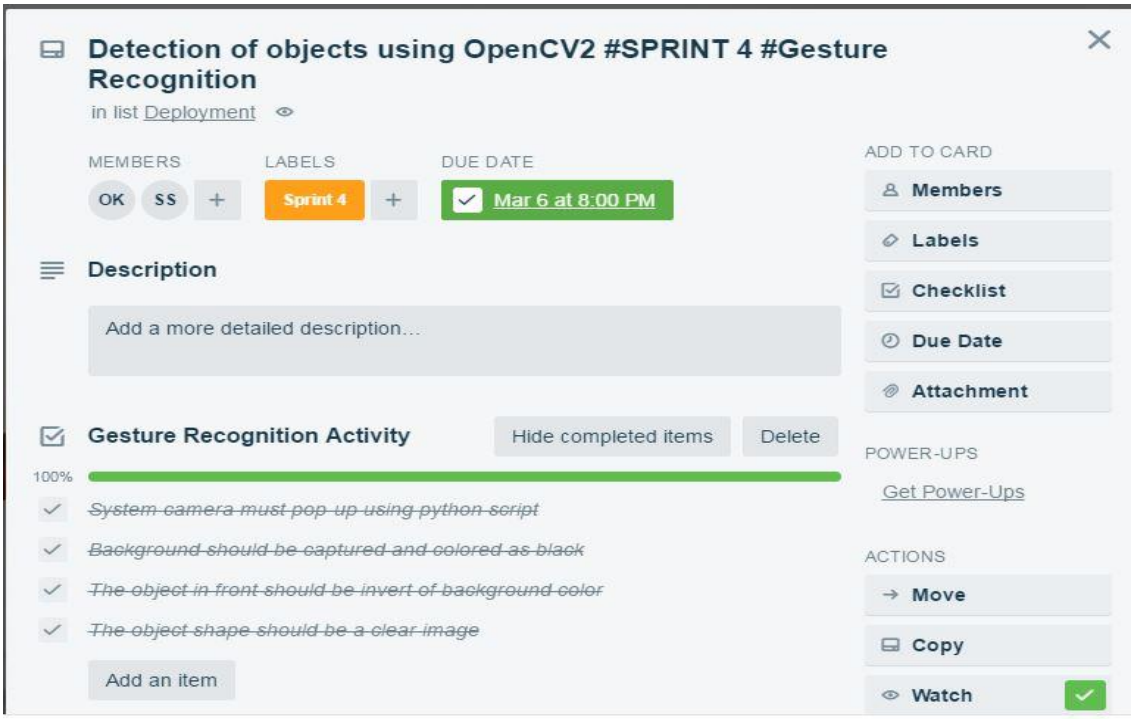

Fig 6.6: schedule and task of sprint 5 "Gesture recognition" along with assigned members

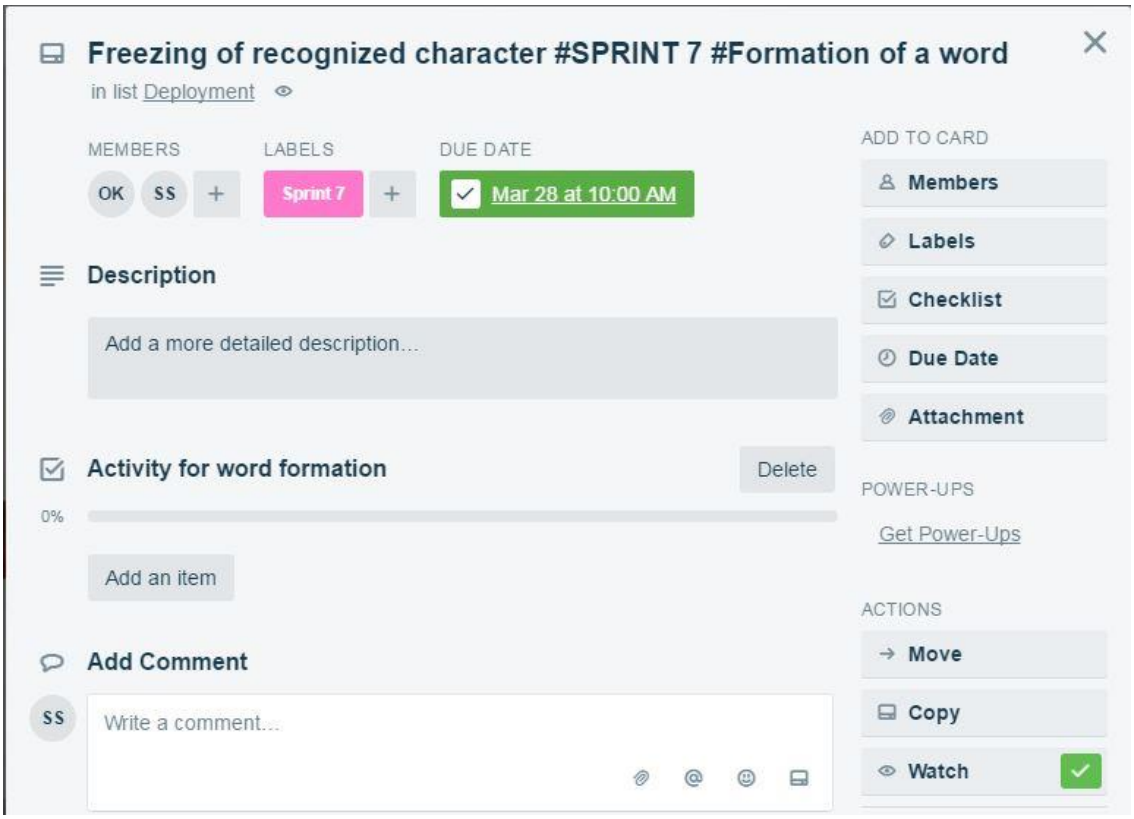

Fig 6.7: schedule of sprint 7 "Freezing of recognized character" along with assigned members

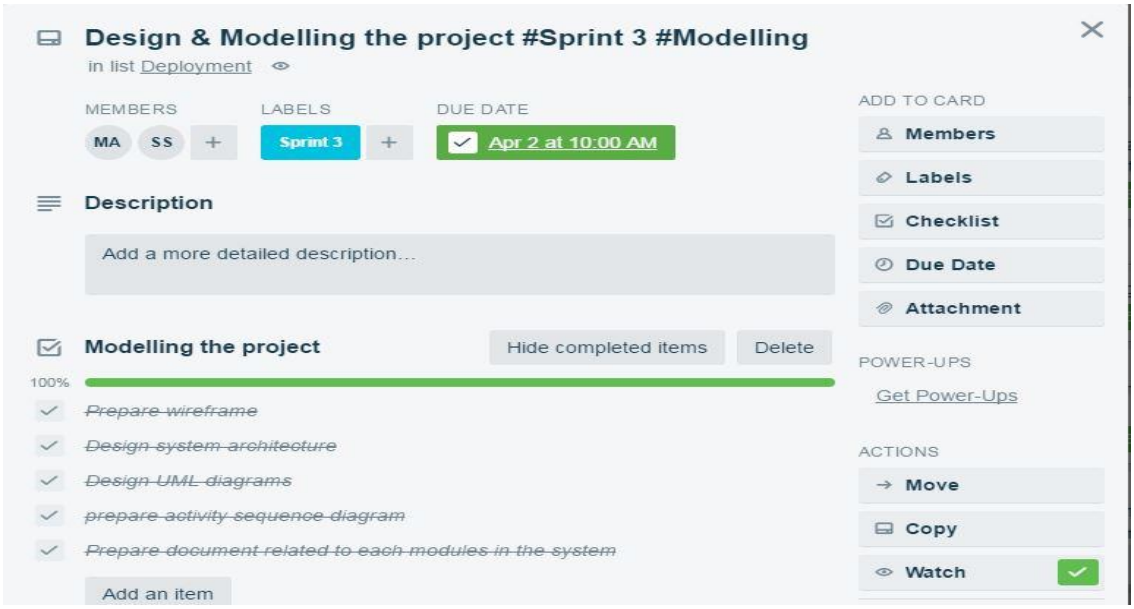

Fig 6.8: schedule and task of sprint 3 "Design & Modelling the project" along with assigned members

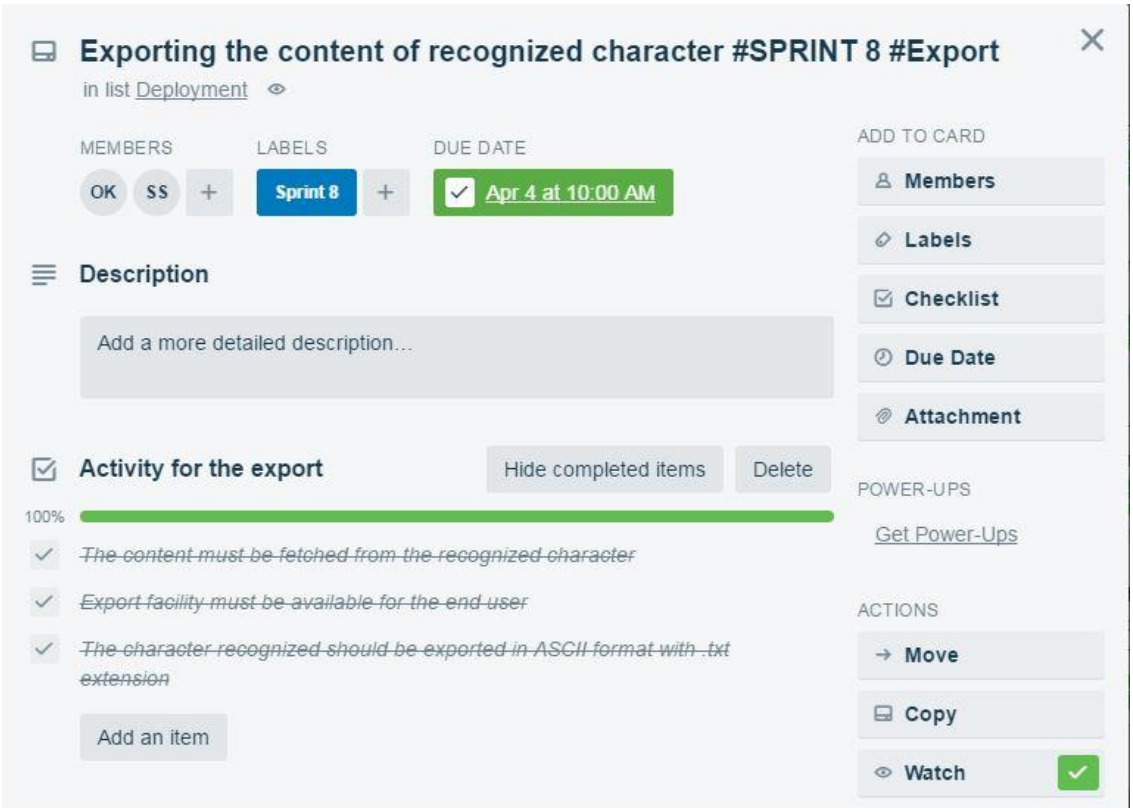

Fig 6.9: schedule and task of sprint 8 "Exporting the content" along with assigned members

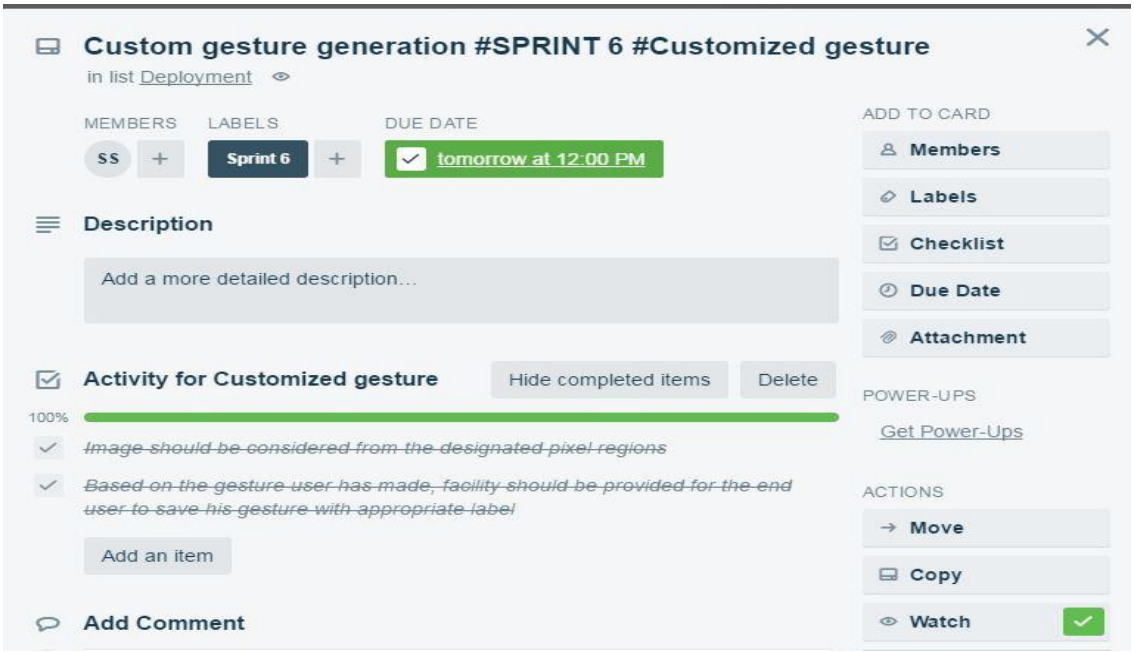

Fig 6.10: schedule and task of sprint 6 "Custom gesture generation" along with assigned members

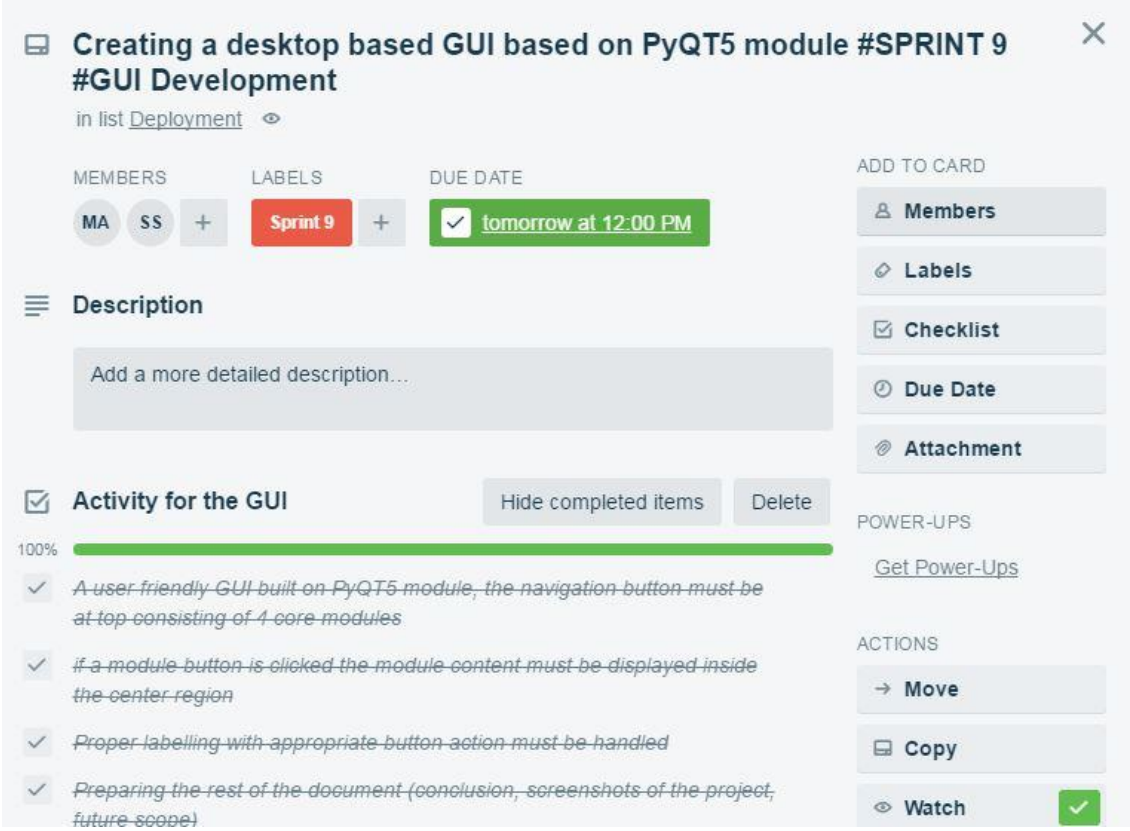

Fig 6.11: schedule and task of sprint 9 "Creating a desktop-based GUI" along with assigned members

Fig 6.12: Members view