A Group Project Report on

SIGN LANGUAGE DETECTION AND VOICE TRANSLATION

Submitted to partial fulfillment of the academic requirements of

Jawaharlal Nehru Technological University Hyderabad
For the award of the degree of

Bachelor of Technology

in

(2018 – 2022)

Submitted By

K. DASHARATH	K.UDAY	T.MADHUKAR
18311A0522	18311A0525	18311A0550

Under the esteemed Guidance of

Mr. GUGULOTHU RAVI

Internal Guide
Assistant Professor, Department of CSE

and

Ms. B. VASUNDHARA DEVI

Project Coordinator
Assistant Professor, Department of CSE

SNIST
Sreenidhi Institute of Science and Technology
Because, Life is all about taking the right decisions
AUTONOMOUS

DEPARTMENT OF COMPUTER SCIENCE AN ENGINEERING

SREENIDHI INSTITUTE OF SCIENCE AND TECHNOLOGY
(Autonomous)
Yamnampet, Ghatkesar Mandal, Medchal District, Hyderabad – 501301

Batch No: A-11

Submission Date : January 2022

CERTIFICATE

This is to certify that this Group Project report on "**SIGN LANGUAGE DETECTION AND VOICE TRANSLATION**", submitted by K,DASHARATH (18311A0522), K. UDAY (18311A0525) and T. MADHUKAR (18311A0550) in the year 2022 in partial fulfillment of the academic requirements of Jawaharlal Nehru Technological University for the award of the degree of Bachelor of Technology in Computer Science and Engineering, is a bonafide work that has been carried out by them as part of their **Project- I** during **Fourth Year First Semester,** under our guidance. This report has not been submitted to any other institute or university for the award of any degree.

Mr. Gugulothu Ravi
Assistant Professor
Department of CSE
Internal Guide

Ms. B Vasundhara Devi
Assistant Professor
Department of CSE
Group Project Coordinator

Dr. Aruna Varanasi
Professor
HOD-CSE

External Examiner
Date:

DECLARATION

We, K.DASHARATH(18311A0522), K.UDAY (18311A0525) and T. MADHUKAR (18311A0550) students of **SREENIDHI INSTITUTE OF SCIENCE AND TECHNOLOGY, YAMNAMPET, GHATKESAR,** studying IVth year Ist semester, **COMPUTER SCIENCE AND ENGINEERING** solemnly declare that the Group Project work, titled **"SIGN LANGUAGE DETECTION AND VOICE TRANS LATION"** is submitted to **SREENIDHI INSTITUTE OFSCIENCE AND TECHNOLOGY** for partial fulfillment for the award of degree of Bachelor of technology in **COMPUTER SCIENCE AND ENGINEERING**.

It is declared to the best of our knowledge that the work reported does not form part of any dissertation submitted to any other University or Institute for award of any degree.

ACKNOWLEDGEMENT

We would like to express our gratitude to all the people behind the screen who helped me to transform an idea into a real application.

We would like to express our heart-felt gratitude to our parents without whom we would not have been privileged to achieve and fulfill our dreams. We are grateful to our principal, **Dr. T. Ch. Siva Reddy,** who most ably runs the institution and has had the major hand in enabling us to do this project.

We profoundly thank **Dr**. **Aruna Varanasi**, Head of the Department of Computer Science & Engineering who has been an excellent guide and also a great source of inspiration to our work.

We would like to thank our internal guide **Mr. Gugulothu Ravi** for his technical guidance, constant encouragement and support in carrying out our project at college.

We also extend our gratitude to our project coordinator **Ms. B Vasundhara Devi**, who lent us valuable guidance and led us towards the staged completion of the project.

The satisfaction and euphoria that accompany the successful completion of the task would be great but incomplete without the mention of the people who made it possible with their constant guidance and encouragement crowns all the efforts with success. In this context, I would like thank all the other staff members, both teaching and non-teaching, who have extended their timely help and eased my task.

KARKA DASHARATH 18311A0522
KOMMINENI UDAY 18311A0525
THUMMALA MADHUKAR 18311A0550

SIGN LANGUAGE DETECTION AND VOICE TRANSLATION

Abstract

Sign language is one of the oldest and most natural form of language for communication, but since most people do not know sign language and interpreters are very difficult to come by, we have come up with a real time method using neural networks for fingerspelling based American sign language. In our method, the hand is first passed through a filter and after the filter is applied the hand is passed through a classifier which predicts the class of the hand gestures, after identifying the alphabet with the help of the model we use trying to convert the identified alphabet to the speech, to communicate with the blind, our idea is to make a communication between the deaf and blind people, our method provides 89 % accuracy for the 26 letters of the alphabet.

List of Figures

INDEX

CHAPTER – I

INTRODUCTION

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.

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.[5] 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. Existing System

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

In the existing system the tutor should be necessary to understand the American Sign Language, one wanted to learn the sign language if we want to communicate with the deaf people, there is no any method that they wanted to communicate with the deaf and blind people.

1.3 Proposed system

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.

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 text 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.

2.SYSTEM OVERVIEW

System design is used for understanding the construction of system. We have explained the flow of our system and the software used in the system in this section.

2.1 Flow of the system

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.

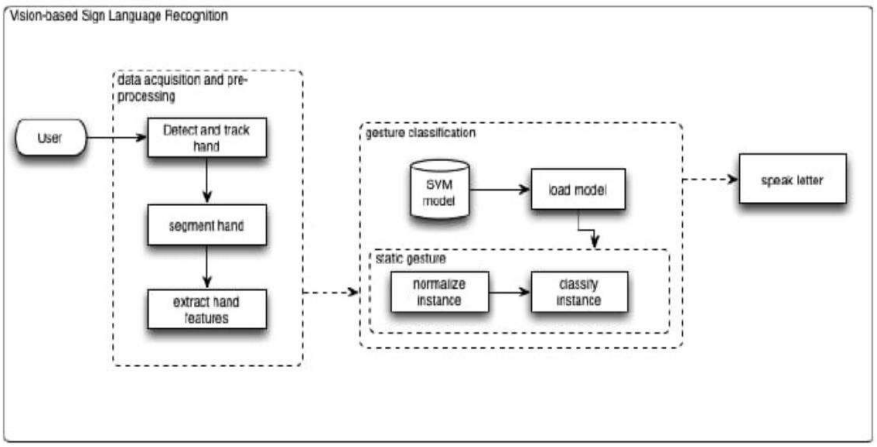

Fig 2.1 Rough idea for implementation

2.2 Flow diagram of the system

The main aim is to in this application is, 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.

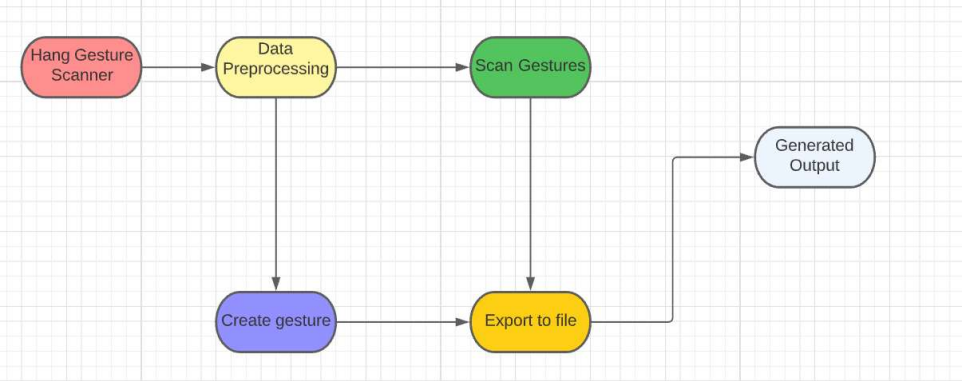

Fig 2.2 implementation roadmap

2.3 Functional Requirement Specification

The System after careful analysis has been identified to be present with the following 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.

2.4 Performance Requirements

Performance is measured in terms of the output provided by the application. Requirement specification plays an important part in the analysis of a system. Only when the requirement specifications are properly given, it is possible to design a system, which will fit into required environment. It rests largely with the users of the existing system to give the requirement specifications because they are the people who finally use the system. This is because the requirements have to be known during the initial

2.5 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, Kera's API.
- PyQT5, Tkinter module.
- Python OpenCV2, SciPy, qimage2ndarray, winGuiAuto, pypiwin32, sys, keyboard, pyttsx3, pillow libraries.

2.6 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.

3.SYSTEM DESIGN

Systems design is the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. One could see it as the application of systems theory to product development. Object-oriented analysis and design methods are becoming the most widely used methods for computer systems design.

3.1 Architectural Design

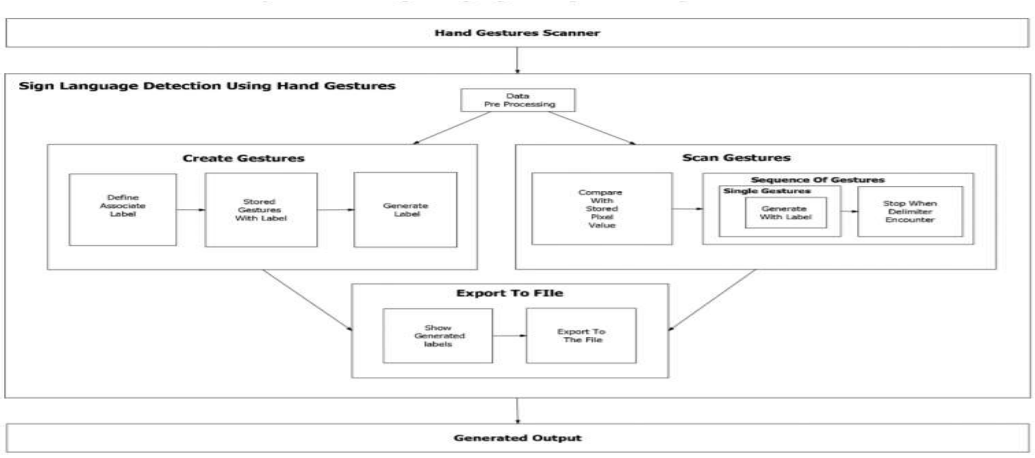

Fig 3.1: Architectural Design

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.

3.2 UML Diagrams

UML Diagrams for our application are as follows:

3.2.1 Use Case Diagrams

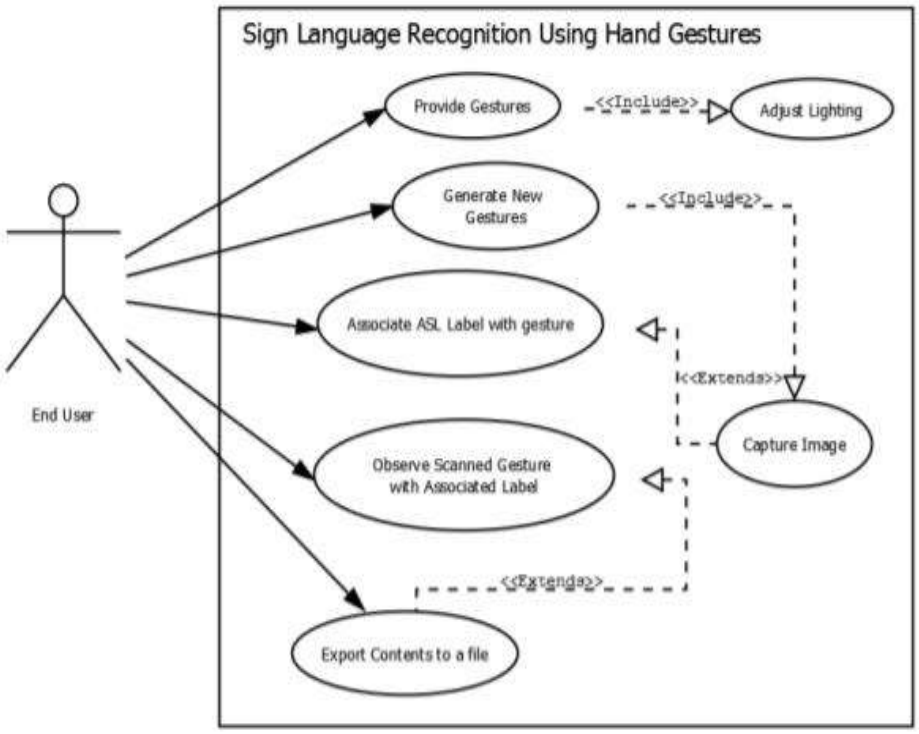

Fig 3.2.1: Use case Diagram for entire application

End User Use Case:

The User is responsible for providing the gestures to the application, which includes adjust lighting, generate the new gestures, associate American Sign Language Label with the gesture, Observe scanned Gesture with associated Label, export contents to a file.

3.2.2 Class Diagrams

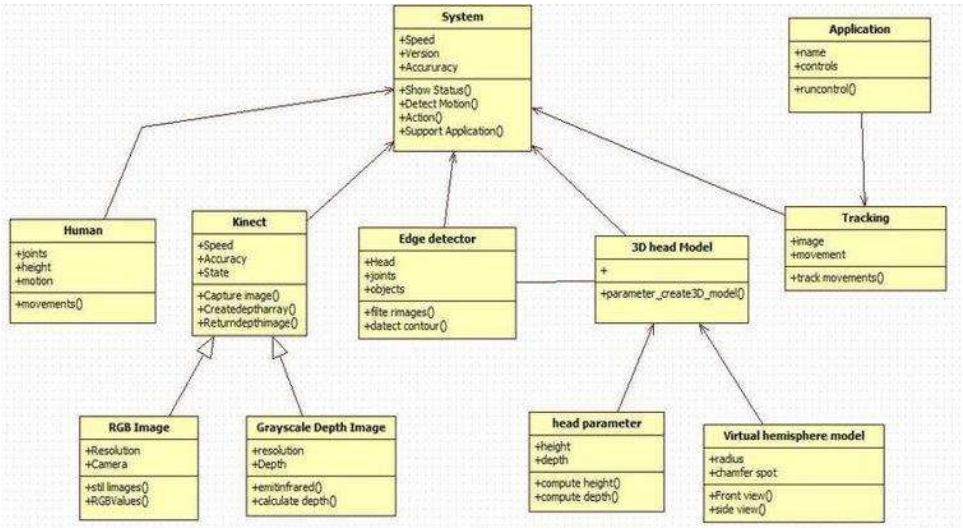

Fig 3.3.2 class Diagram for entire application

In our application we mainly identified seven classes namely, system, application, human, Kinect, Edge Detector, 3D heads Model, tracking with the help of all this classes we will implement our model, this classes also include few various classes like RGB image, Gray Scale Depth Image, Head Parameter, virtual Hemisphere Model

3.2.3 Sequence Diagrams

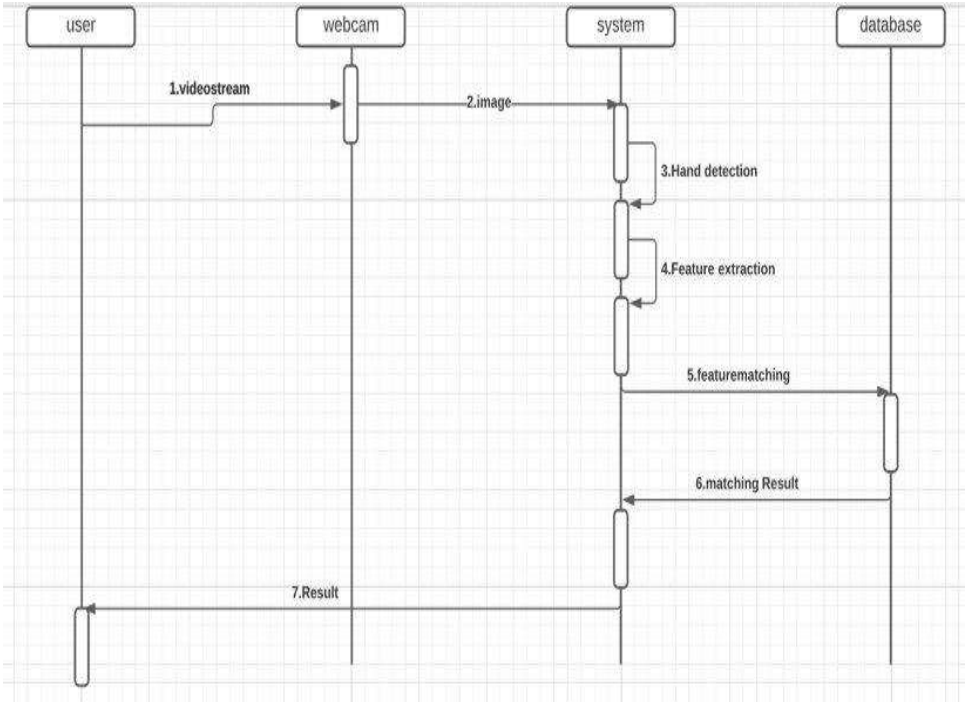

Fig 3.3.3: Sequence Diagram for entire application.

In Sequence diagram sequence of actions are triggered, at first user enters the application, the User Interface is video stream, where the image of the hand is captured with the help of hand detection, then next step is feature extraction, feature engineering and matching the result.

3.2.4 Activity Diagrams

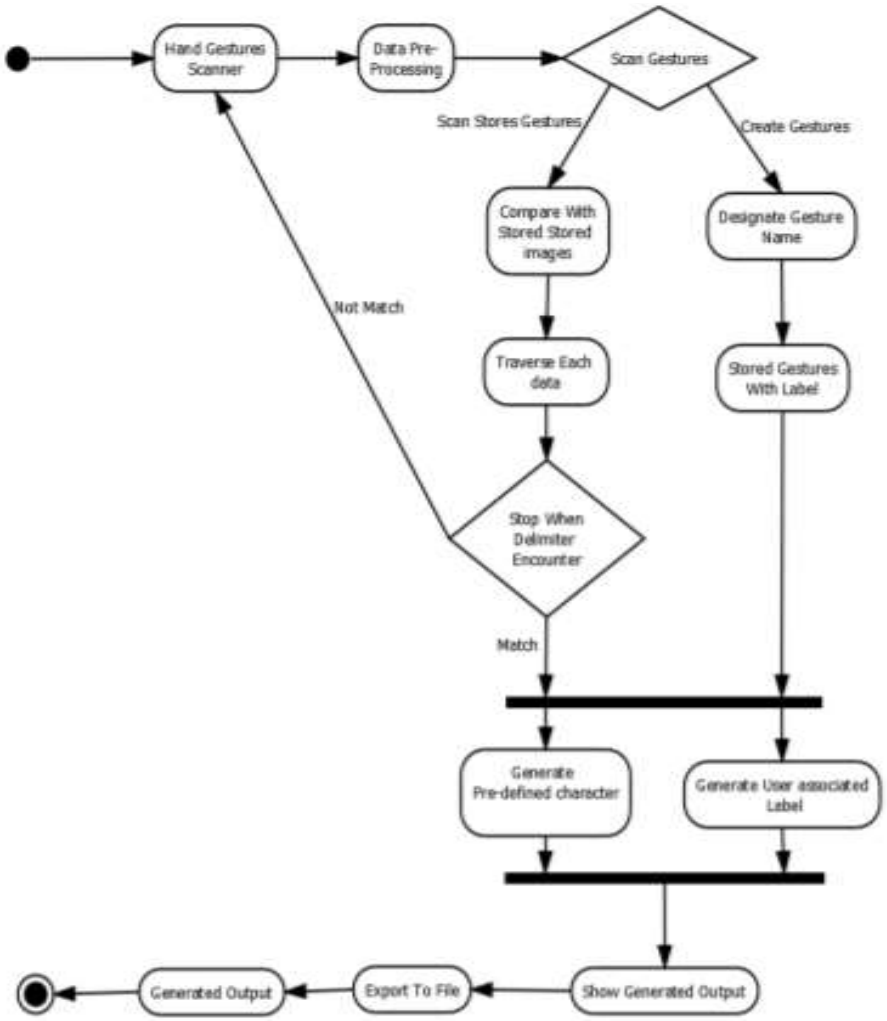

Fig 3.3.4: Activity diagram of entire model.

In activity diagram sequence of actions are triggered, the application is starting by the user, at first hand gesture is scanned then Data preprocessing, after data preprocessing there we can scan the gestures, then we have two options, scan stored gestures and the create gesture, then we can show the generated output, then we export to the file and then we generate the output.

4.SYSTEM IMPLEMENTATION

The implementation stage of any project is a true display of the defining moments that make a project a success or a failure. The implementation stage is defined as the system or system modifications being installed and made operational in a production environment. The phase is initiated after the system has been tested and accepted by the user. This phase continues until the system is operating in production in accordance with the defined user requirements.

Code:

```
__author__ = 'Madhukar,Dasharath,Uday'

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 imageio 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 win32con                    #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("C:/Users/thummala
madhukar/OneDrive/Desktop/IV_1_project/Sign-Language-Recognition-Using-Hand-
Gestures-Keras-PyQT5-OpenCV-master/Source Code/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
```

Defining the Clear Function

```
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('.C:/Users/thummala
madhukar/OneDrive/Desktop/IV_1_project/Sign-Language-Recognition-Using-Hand-
Gestures-Keras-PyQT5-OpenCV-master/Source Code/SampleGestures'):
os.mkdir('C:/Users/thummala madhukar/OneDrive/Desktop/IV_1_project/Sign-
Language-Recognition-Using-Hand-Gestures-Keras-PyQT5-OpenCV-master/Source
Code/SampleGestures')

gesname=saveimg[-1]
if(len(gesname)>=1):
```

```
img_name = "C:/Users/thummala madhukar/OneDrive/Desktop/IV_1_project/Sign-Language-Recognition-Using-Hand-Gestures-Keras-PyQT5-OpenCV-master/Source Code/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)
```

Defining the predictor function:

```
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("C:/Users/thummala madhukar/OneDrive/Desktop/IV_1_project/Sign-Language-Recognition-Using-Hand-Gestures-Keras-PyQT5-OpenCV-master/Source Code/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 matcches
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'
```

Defining the check file function:

```
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
```

Defining the GUI Class:

```
class Dashboard(QtWidgets.QMainWindow):
def __init__(self):
```

```
super(Dashboard, self).__init__()
self.setWindowFlags(QtCore.Qt.WindowMinimizeButtonHint)
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.WINDOW_F
ULLSCREEN)
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('C:/Users/thummala
madhukar/OneDrive/Desktop/IV_1_project/Sign-Language-Recognition-Using-Hand-
Gestures-Keras-PyQT5-OpenCV-master/Source Code/icons/windowLogo.png'))
self.title = 'Sign language Recognition'
uic.loadUi('C:/Users/thummala madhukar/OneDrive/Desktop/IV_1_project/Sign-
Language-Recognition-Using-Hand-Gestures-Keras-PyQT5-OpenCV-master/Source
Code/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("C:/Users/thummala
madhukar/OneDrive/Desktop/IV_1_project/Sign-Language-Recognition-Using-Hand-
Gestures-Keras-PyQT5-OpenCV-master/Source Code/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')
```

Defining the Create Gesture function:

```
def createGest(self):
""" Custom gesture generation module"""
try:
clearfunc(self.cam)
except:
pass
gesname=""
uic.loadUi('C:/Users/thummala madhukar/OneDrive/Desktop/IV_1_project/Sign-
Language-Recognition-Using-Hand-Gestures-Keras-PyQT5-OpenCV-master/Source
Code/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('C:/Users/thummala
madhukar/OneDrive/Desktop/IV_1_project/Sign-Language-Recognition-Using-Hand-
Gestures-Keras-PyQT5-OpenCV-master/Source Code/SampleGestures'):
os.mkdir('C:/Users/thummala madhukar/OneDrive/Desktop/IV_1_project/Sign-
Language-Recognition-Using-Hand-Gestures-Keras-PyQT5-OpenCV-master/Source
Code/SampleGestures')
if(len(gesname)>=1):
img_name = "C:/Users/thummala madhukar/OneDrive/Desktop/IV_1_project/Sign-
Language-Recognition-Using-Hand-Gestures-Keras-PyQT5-OpenCV-master/Source
Code/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("C:/Users/thummala
madhukar/OneDrive/Desktop/IV_1_project/Sign-Language-Recognition-Using-Hand-
Gestures-Keras-PyQT5-OpenCV-master/Source
Code/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('C:/Users/thummala  madhukar/OneDrive/Desktop/IV_1_project/Sign-
Language-Recognition-Using-Hand-Gestures-Keras-PyQT5-OpenCV-master/Source
Code/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
#buttonaxe._button = bcut
```

Defining the Scan Sentence function:

```
def scanSent(self):
"""sentence  formation module """
try:
clearfunc(self.cam)
except:
pass
uic.loadUi('C:/Users/thummala  madhukar/OneDrive/Desktop/IV_1_project/Sign-
Language-Recognition-Using-Hand-Gestures-Keras-PyQT5-OpenCV-master/Source
Code/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
```

Defining the Single Alphabet function:

```
def scanSingle(self):
"""Single gesture scanner """
try:
clearfunc(self.cam)
except:
pass
uic.loadUi('C:/Users/thummala madhukar/OneDrive/Desktop/IV_1_project/Sign-Language-Recognition-Using-Hand-Gestures-Keras-PyQT5-OpenCV-master/Source Code/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())
```

5. OUTPUT SCREENS

Output Screens of various functionalities in our application are shown over here along with the description.

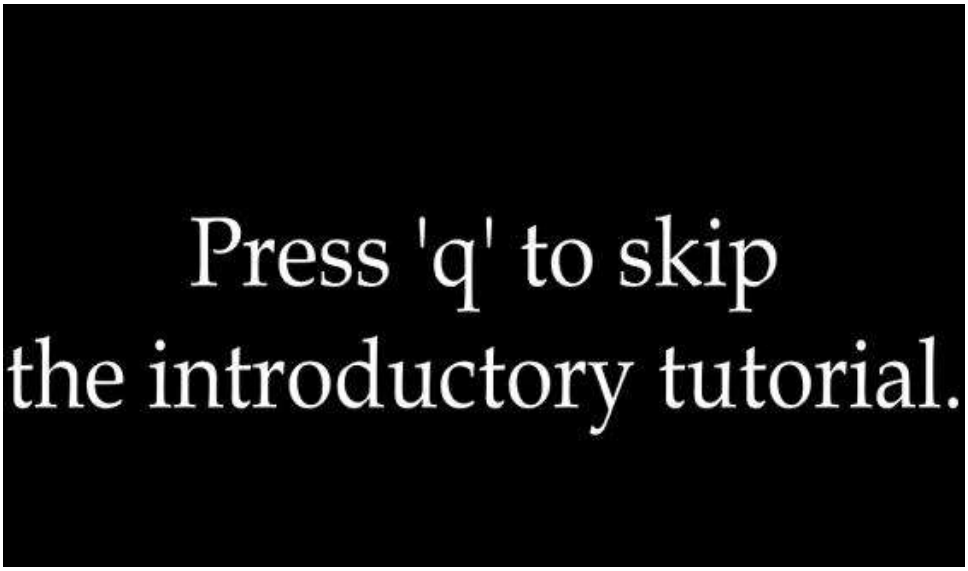

Fig 5.1 Introductory video

Fig 5.2 Dashboard with simple Animation of the application

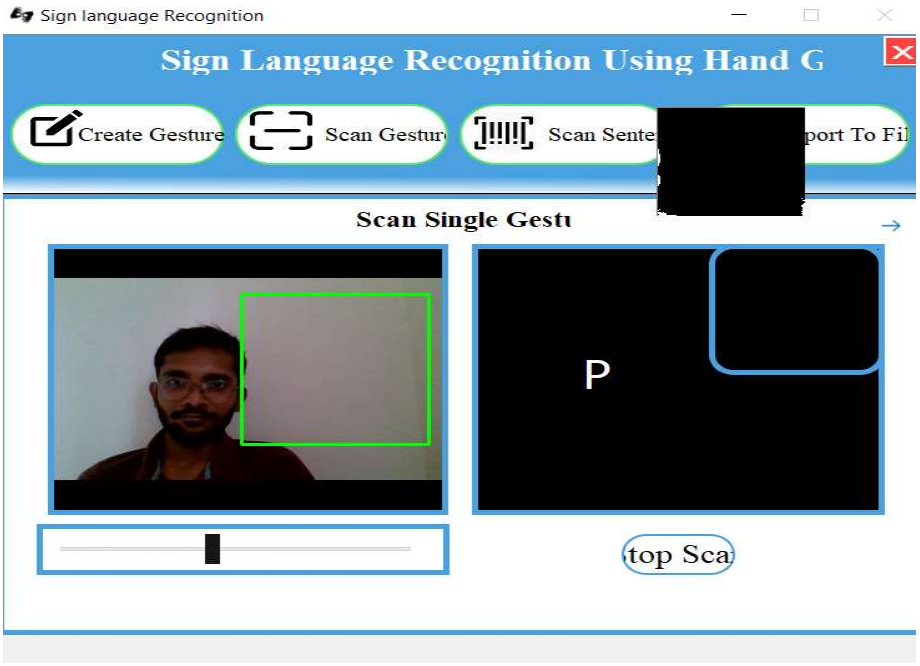

Fig 5.3 Scanning the Single Gesture

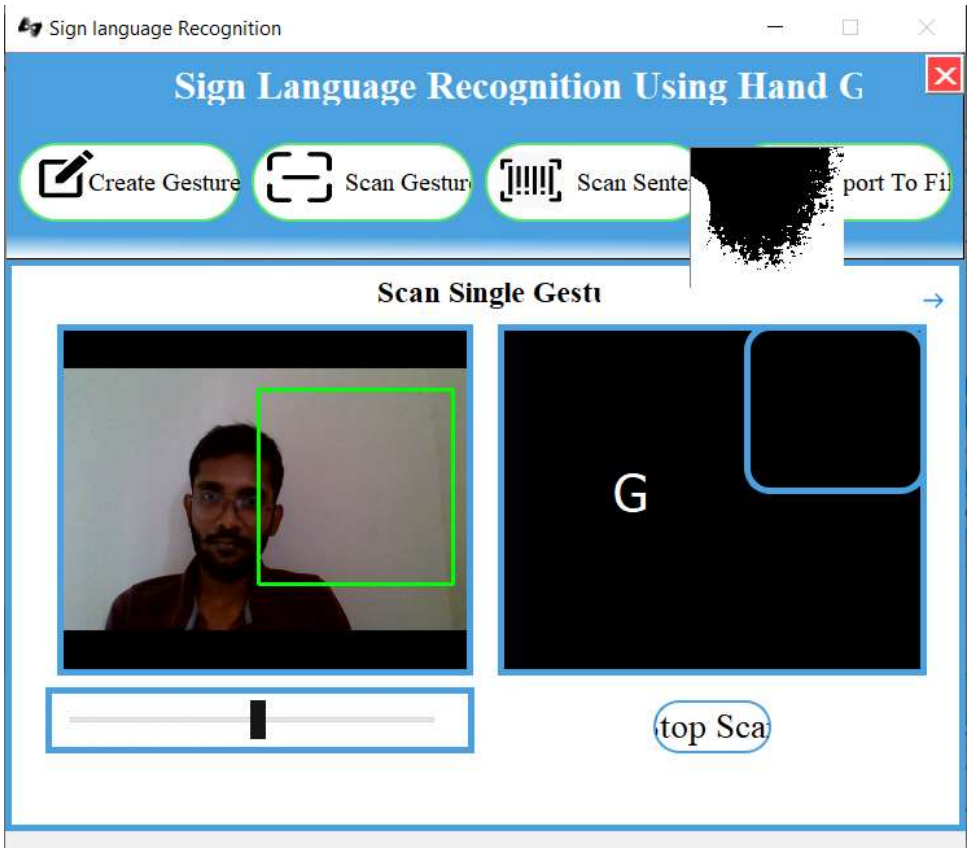

Fig 5.4 Adjusting the lighting for camera

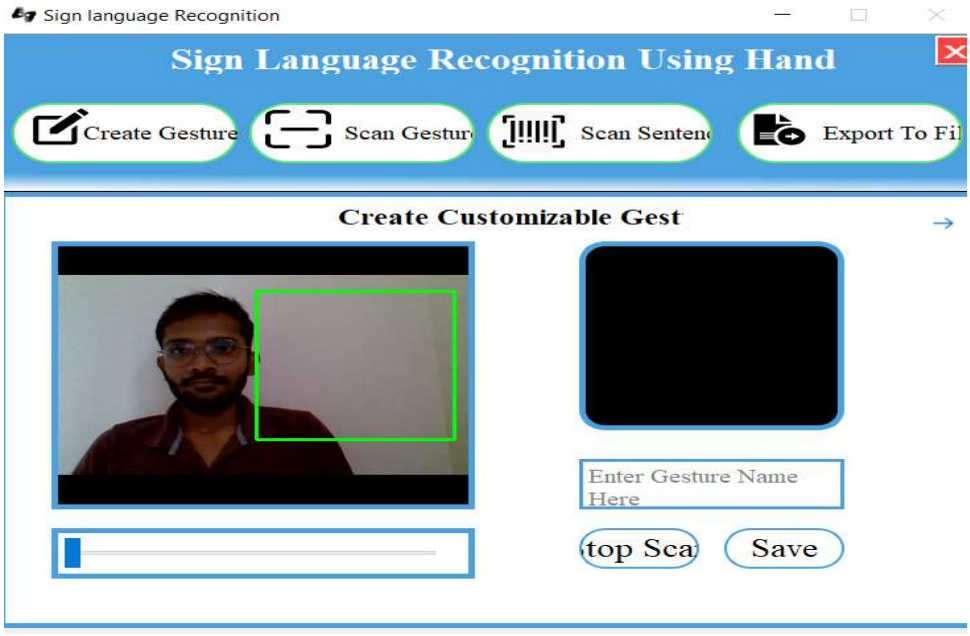

Fig 5.5 creating the customizable Gesture

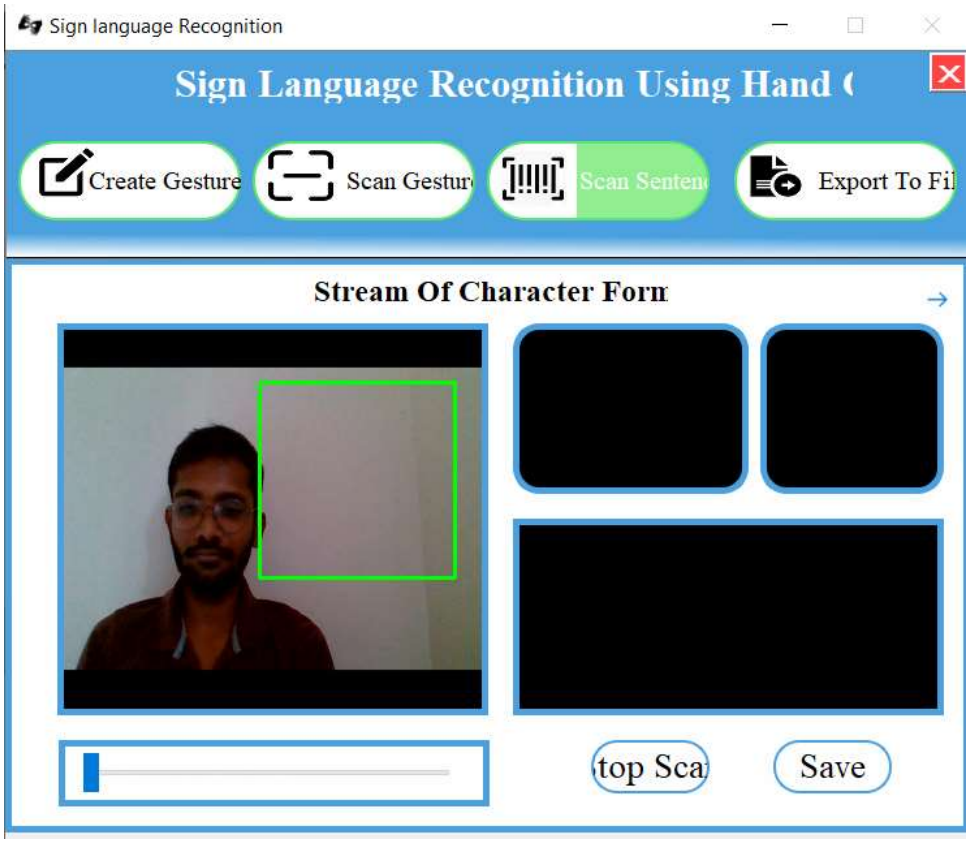

Fig 5.6 Scanning the stream of Characters

Fig 5.7 Exporting the file

Fig 5.8 File Saving successfully.

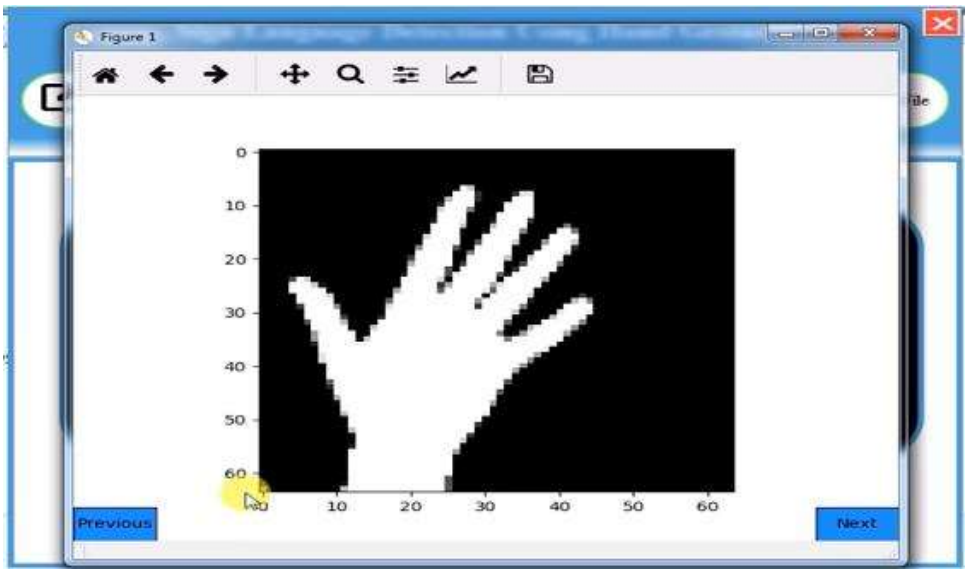

Fig 5.9 Gesture Viewer

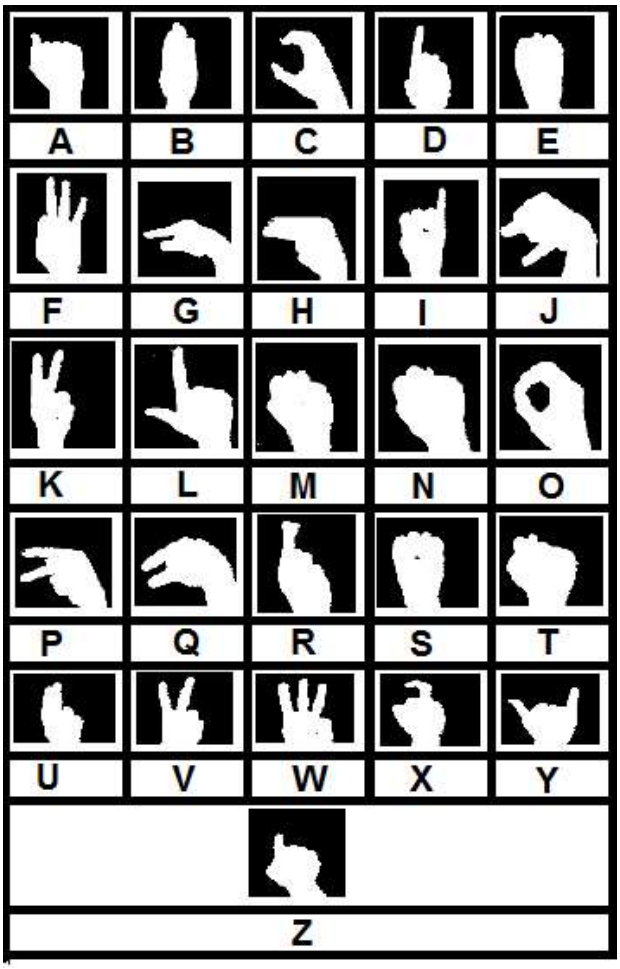

Fig 5.10 Gestures of 26 Alphabet

6.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.

7.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.

8.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-whileusing-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_Syste ms_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-culturefrequently-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]

APPENDIX-A: About Kera's, Neural network, PyQT5

Kera's:

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

Use Keras if you need a deep learning library that:

- Allows for easy and fast prototyping (through total modularity, minimalism, and extensibility).
- Supports both convolutional networks and recurrent networks, as well as combinations of the two.
- Supports arbitrary connectivity schemes (including multi-input and multi-output training).
- Runs seamlessly on CPU and GPU.

To know more about keras read the documentation at Keras.io.

Artificial Neural Networks: Artificial Neural Network is a connection of neurons, replicating the structure of human brain. Each connection of neuron transfers information to another neuron. Inputs are fed into first layer of neurons which processes it and transfers to another layer of neurons called as hidden layers. After processing of information through multiple layers of hidden layers, information is passed to final output layer. 10 There are capable of learning and they have to be trained. There are different learning strategies:

1. Unsupervised Learning
2. Supervised Learning
3. Reinforcement Learning

Convolution Neural Network: Unlike regular Neural Networks, in the layers of CNN, the neurons are arranged in 3 dimensions: width, height, depth. The neurons in a layer will only be connected to a small region of the layer (window size) before it, instead of all of the neurons in a fully-connected manner. Moreover, the final output layer would have dimensions (number of classes), because by the end of the CNN architecture we will reduce the full image into a single vector of class scores. 11 1. Convolution Layer: In convolution layer we take a small window size [typically of length 5*5] that extends to the depth of the input matrix. The layer consists of learnable filters of window size. During every iteration we slid the window by stride size [typically 1], and compute the dot product of filter entries and input values at a given position. As we continue this process well create a 2-Dimensional activation matrix that gives the response of that matrix at every spatial position. That is, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some color 2. Pooling Layer: We use pooling layer to decrease the size of activation matrix and ultimately reduce the learnable parameters.

PYQt5:

Qt for Python offers the official Python bindings for Qt, and has two main components:

- PySide6, so that you can use Qt6 APIs in your Python applications, and
- Shiboken6, a binding generator tool, which can be used to expose C++ projects to Python, and a Python module with some utility functions.

Porting from PySide2 to PySide6 provides information on porting existing PySide2 applications.

This project is available under the LGPLv3/GPLv3 and the Qt commercial license