

AIML - PROJECT REPORT

TITLE: LIVE SIGN LANGUAGE DETECTOR FOR ENGLISH ALPHABETS

DOMAIN: MACHINE LEARNING AND COMPUTER VISION

AIM:

To develop a real-time sign language recognition system that captures hand gestures, converts them into corresponding letters, and provides audio feedback. The system utilizes computer vision and machine learning for accurate gesture detection. It aims to assist individuals with hearing impairments and support sign language learning.

ALGORITHM:

1. Capture Video Feed:

- Use OpenCV to capture the live video feed from a webcam.

2. Hand Detection:

- Use Mediapipe (a framework by Google) to detect and track hand landmarks. Mediapipe provides a predefined model for hand tracking that returns 21 key landmarks on the hand.

3. Feature Extraction:

- Extract and analyze the position of key landmarks on the hand (such as fingertips, thumb tip, and other joint positions).

4. Gesture Recognition:

- Based on the relative position of the hand landmarks, classify the gesture by comparing the relative positions of the fingertips to their base joints. Each hand gesture is associated with a letter of the alphabet (A-Z).

5. Display and Audio Feedback:

- Once a gesture is identified, display the corresponding letter on the screen. Use the pyttsx3 library to provide audio feedback, speaking out the letter detected.

6. Continuous Process:

- This process repeats in a loop to continuously detect and recognize hand gestures as the user interacts with the system.

TOOLS AND TECHNOLOGIES USED:

1. Python

- The core programming language used for the development of the project.

2. OpenCV

- A computer vision library used to capture the video feed from the webcam and process the image frames.

3. Mediapipe

- A framework by Google that provides pre-built models for hand detection and tracking. It detects 21 hand landmarks and provides real-time hand tracking.

4. Tkinter

- A Python library used for creating the graphical user interface (GUI). Tkinter is used here to display the live video feed and the detected letter.

5. Pillow (PIL)

- A Python Imaging Library used to convert images from OpenCV (which are in NumPy array format) into images that can be displayed in the Tkinter interface.

6. pyttsx3

- A Python text-to-speech library used to convert the detected letter into speech output.

INPUT:

The input to the system is the **live video feed** from a webcam. The webcam captures the user's hand gestures, which are then processed in real-time.

PROGRAM:

```
from tkinter import *

from PIL import Image, ImageTk

import cv2

import mediapipe as mp

import pytsx3

win = Tk()

width = win.winfo_screenwidth()

height = win.winfo_screenheight()

win.geometry("%dx%d" % (width, height))

win.configure(bg="#FFFFFF7")

win.title('SIGN LANGUAGE CONVERTER')

global img, finalImage, finger_tips, thumb_tip, cap, image, rgb, hand, results, _, w, h, status,
mpDraw, mpHands, hands, label1, upCount, cshow

cap = None

Label(win,

      text='SIGN LANGUAGE CONVERTER',

      font=('Helvetica', 18, 'bold'),

      bd=5,

      bg='black',

      fg='white',

      relief=SOLID,

      width=200
```

```
).pack(pady=15, padx=300)
```

```
def wine():
```

```
    global finger_tips, thumb_tip, mpDraw, mpHands, cap, w, h, hands, label1, img
```

```
    finger_tips = [8, 12, 16, 20]
```

```
    thumb_tip = 4
```

```
    w = 500
```

```
    h = 400
```

```
    if cap:
```

```
        cap.release() # Release the previous video capture
```

```
    label1 = Label(win, width=w, height=h, bg="#FFFFFF7")
```

```
    label1.place(x=40, y=200)
```

```
    mpHands = mp.solutions.hands
```

```
    hands = mpHands.Hands()
```

```
    mpDraw = mp.solutions.drawing_utils
```

```
    cap = cv2.VideoCapture(0) # Start video capture
```

```
def live():
```

```
    global v, upCount, cshow, img
```

```
    cshow = ""
```

```
    upCount = StringVar()
```

```
    _, img = cap.read()
```

```
    img = cv2.resize(img, (w, h))
```

```
    rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

```
    results = hands.process(rgb)
```

```
if results.multi_hand_landmarks:

    for hand in results.multi_hand_landmarks:

        lm_list = [lm for lm in hand.landmark]

        finger_tips = [8, 12, 16, 20]

        finger_fold_status = [lm_list[tip].y > lm_list[tip - 2].y for tip in finger_tips] # Check Y-
axis for folding

        thumb_tip = lm_list[4]

        thumb_base = lm_list[2]

        index_tip = lm_list[8]

        middle_tip = lm_list[12]

        ring_tip = lm_list[16]

        pinky_tip = lm_list[20]

        if all(finger_fold_status) and thumb_tip.y < thumb_base.y:

            cshow = "A"

        elif not any(finger_fold_status) and thumb_tip.y < thumb_base.y:

            cshow = "B"

        elif (lm_list[8].y > lm_list[6].y and lm_list[8].y < lm_list[5].y) and \

            (lm_list[12].y > lm_list[10].y and lm_list[12].y < lm_list[9].y):

            cshow = "C"

        elif finger_fold_status == [False, True, True, True] and thumb_tip.y < thumb_base.y:

            cshow = "D"

        elif all(finger_fold_status) and thumb_tip.y > thumb_base.y:

            cshow = "E"
```

```

elif ((thumb_tip.x - index_tip.x) ** 2 + (thumb_tip.y - index_tip.y) ** 2) ** 0.5 < 0.05 \
    and not any(finger_fold_status[1:]):

    cshow = "F"

elif finger_fold_status == [False, True, True, True] and thumb_tip.y > thumb_base.y:

    cshow = "G"

elif finger_fold_status == [False, False, True, True] and \
    abs(index_tip.x - middle_tip.x) < 0.05:

    cshow = "H"

elif finger_fold_status == [True, True, True, False] and thumb_tip.y < thumb_base.y:

    cshow = "I"

elif finger_fold_status == [True, True, True, False] and thumb_tip.x < thumb_base.x:

    cshow = "J"

elif finger_fold_status == [False, True, True, True] and thumb_tip.y < thumb_base.y and
/thumb_tip.x > index_tip.x:

    cshow = "K"

elif finger_fold_status == [False, True, True, True] and thumb_tip.x < thumb_base.x:

    cshow = "L"

elif all(finger_fold_status) and thumb_tip.y > lm_list[9].y:

    cshow = "M"

elif all(finger_fold_status) and thumb_tip.y > lm_list[10].y:

    cshow = "N"

elif ((thumb_tip.x - index_tip.x) ** 2 + (thumb_tip.y - index_tip.y) ** 2) ** 0.5 < 0.05
and \ ((thumb_tip.x - middle_tip.x) ** 2 + (thumb_tip.y - middle_tip.y) ** 2) ** 0.5 < 0.05:

```

```
cshow = "O"

elif finger_fold_status == [True, False, True, True] and thumb_tip.y < thumb_base.y:

    cshow = "P"

elif finger_fold_status == [True, True, True, False] and thumb_tip.y < thumb_base.y:

    cshow = "Q"

elif finger_fold_status == [False, False, True, True] and middle_tip.x < index_tip.x:

    cshow = "R"

elif all(finger_fold_status) and thumb_tip.y > thumb_base.y:

    cshow = "S"

elif all(finger_fold_status) and thumb_tip.y < index_tip.y and thumb_tip.y <
middle_tip.y:

    cshow = "T"

elif finger_fold_status == [False, False, True, True] and \

    abs(index_tip.x - middle_tip.x) > 0.1:

    cshow = "U"

elif finger_fold_status == [False, False, True, True]:

    cshow = "V"

elif finger_fold_status == [False, False, False, True]:

    cshow = "W"

elif finger_fold_status == [True, True, True, True] and index_tip.y > lm_list[6].y:

    cshow = "X"

elif finger_fold_status == [True, True, True, False] and thumb_tip.y < thumb_base.y:

    cshow = "Y"
```

```
elif finger_fold_status == [False, True, True, True] and thumb_tip.x < index_tip.x:

    cshow = "Z"

else:

    cshow = ""

upCount.set(cshow)

mpDraw.draw_landmarks(rgb, hand, mpHands.HAND_CONNECTIONS)

cv2.putText(rgb,

    f"Detected Letter: {cshow}",

    (10, 50),

    cv2.FONT_HERSHEY_SIMPLEX,

    1,

    (0, 0, 0),

    2,

    cv2.LINE_AA)

image = Image.fromarray(rgb)

finalImage = ImageTk.PhotoImage(image)

label1.configure(image=finalImage)

label1.image = finalImage

win.after(1, live)

def voice():

    engine = pyttsx3.init()

    engine.say(upCount.get())

    engine.runAndWait()
```



```
wine()
```

```
button_width = 15
```

```
button_height = 2
```

```
button_font = ('Helvetica', 12, 'bold')
```

```
button_y_start = height // 2 - 100
```

```
Button(win,
```

```
    text='Live',
```

```
    bg='black',
```

```
    fg='white',
```

```
    relief=RAISED,
```

```
    width=button_width,
```

```
    height=button_height,
```

```
    font=button_font,
```

```
    command=live
```

```
).place(x=width - 250, y=button_y_start)
```

```
Button(win,
```

```
    text='Sound',
```

```
    bg='black',
```

```
    fg='white',
```

```
    relief=RAISED,
```

```
    width=button_width,
```

```
    height=button_height,
```

```
    font=button_font,
```

```
command=voice

).place(x=width - 250, y=button_y_start + 100)

Button(win,

    text='Exit',

    bg='black',

    fg='white',

    relief=RAISED,

    width=button_width,

    height=button_height,

    font=button_font,

    command=win.destroy

).place(x=width - 250, y=button_y_start + 200)

win.mainloop()
```

EXECUTION STEPS:

1. Initialize the GUI:

- The GUI is created using Tkinter, with a black background and a large window to display the live video feed.
- A label is placed within the window to show the live video stream from the webcam.

2. Hand Detection Setup:

- The hand detection process is set up using the Mediapipe framework.
- Mediapipe detects 21 key hand landmarks (such as fingertips, base of the fingers, and wrist).

3. Start Webcam Feed:

- OpenCV is used to access the webcam and start capturing frames.

4. Process Each Frame for Hand Gesture Recognition:

- For each frame captured from the webcam, the landmarks of the hand are identified using Mediapipe.
- The positions of the landmarks are analyzed to determine whether they correspond to a specific hand gesture.

5. Gesture Classification:

- Based on predefined rules (such as the relative position of fingertips and their base joints), the gesture is classified into a letter.
- A set of conditions is used to check the positions of each finger, for example:
 - Fist with the thumb out corresponds to letter "A".
 - All fingers up (palm flat) corresponds to "B", and so on.

6. Display Detected Letter:

- The detected letter is displayed on the screen using a Label widget from Tkinter.

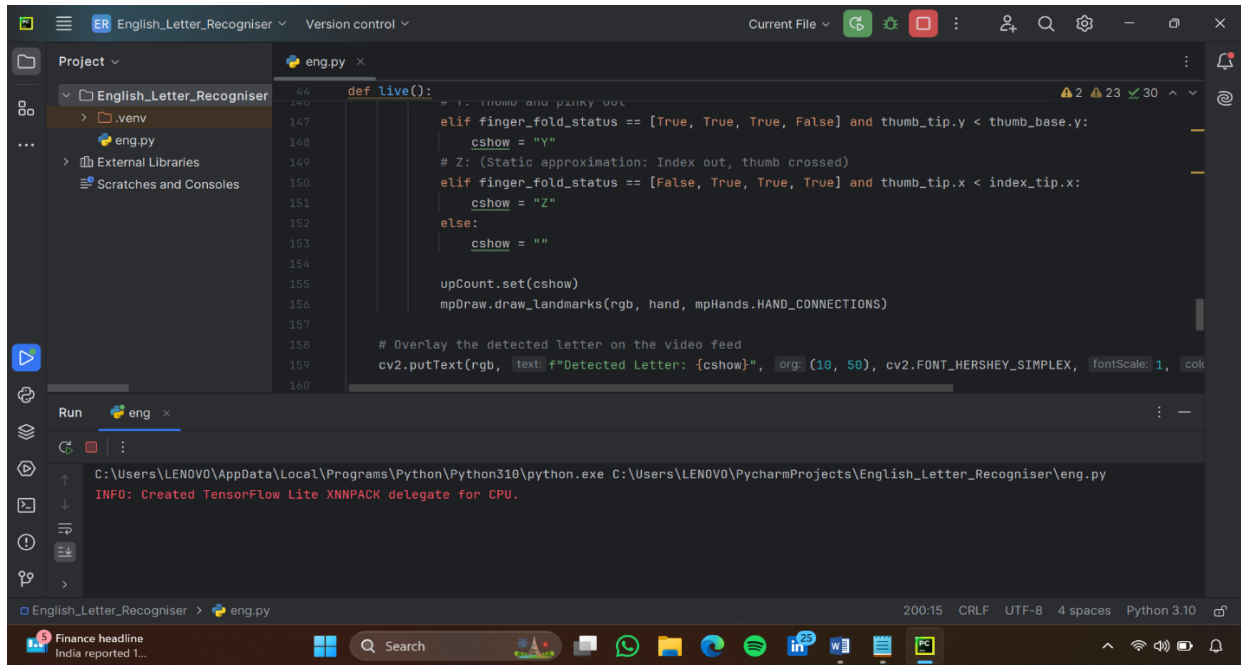
7. Provide Audio Feedback:

- After detecting the gesture, the system uses pyttsx3 to speak out the letter.

8. Repeat the Process:

- The process of hand gesture recognition continues indefinitely, with the live() function being called repeatedly using the win.after() method

OUTPUT:

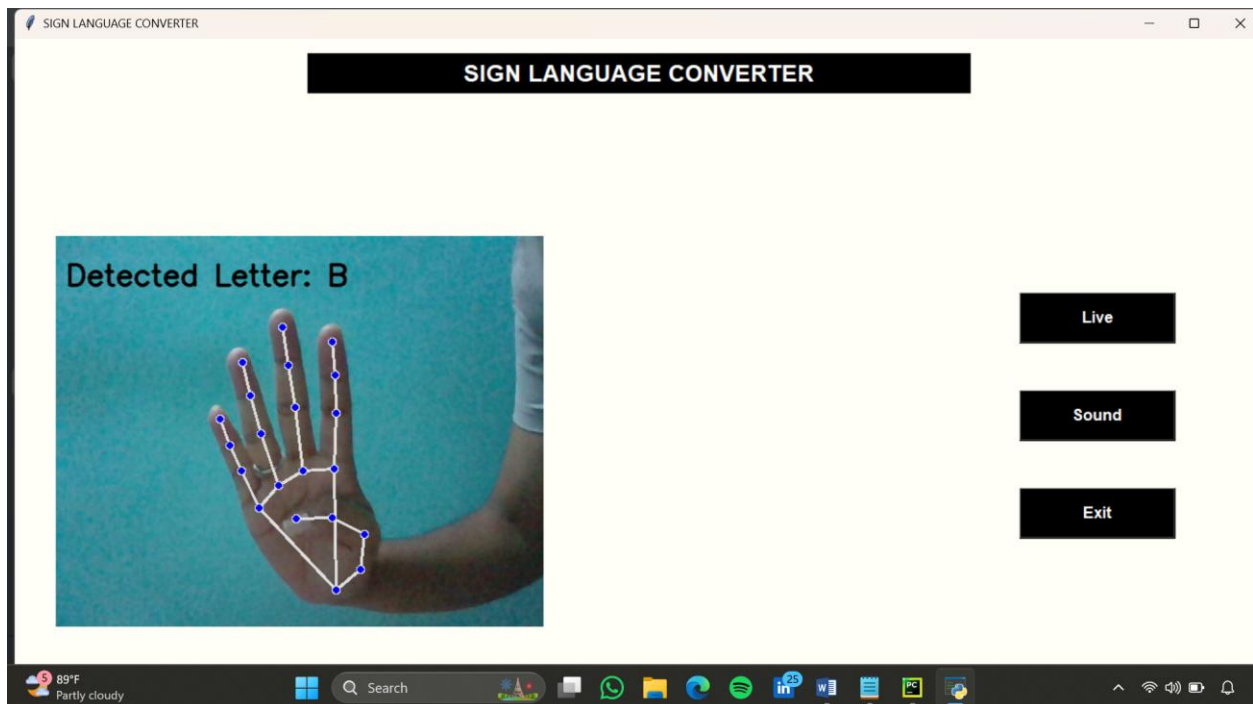


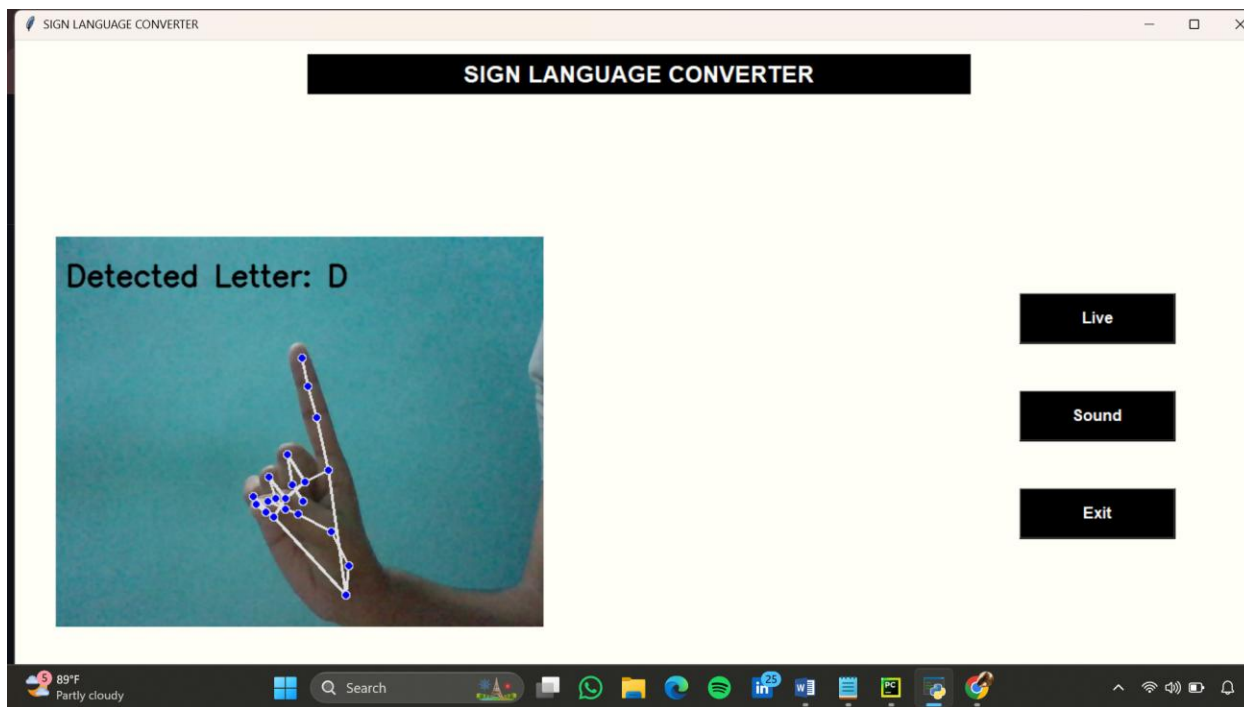
The screenshot shows the PyCharm IDE interface. The top toolbar includes icons for file operations, search, and settings. The left sidebar displays the project structure for 'English_Letter_Recogniser', showing a 'venv' folder and the 'eng.py' file. The main editor window displays the 'eng.py' file with Python code. The code defines a 'live()' function that processes video frames to detect letters. It uses 'cv2' for image processing and 'mpHands' for hand landmark detection. The function sets a color 'cshow' based on the detected letter and updates a 'upCount' variable. The bottom panel shows the 'Run' output, indicating that the program executed successfully and created a TensorFlow Lite XNNPACK delegate for the CPU.

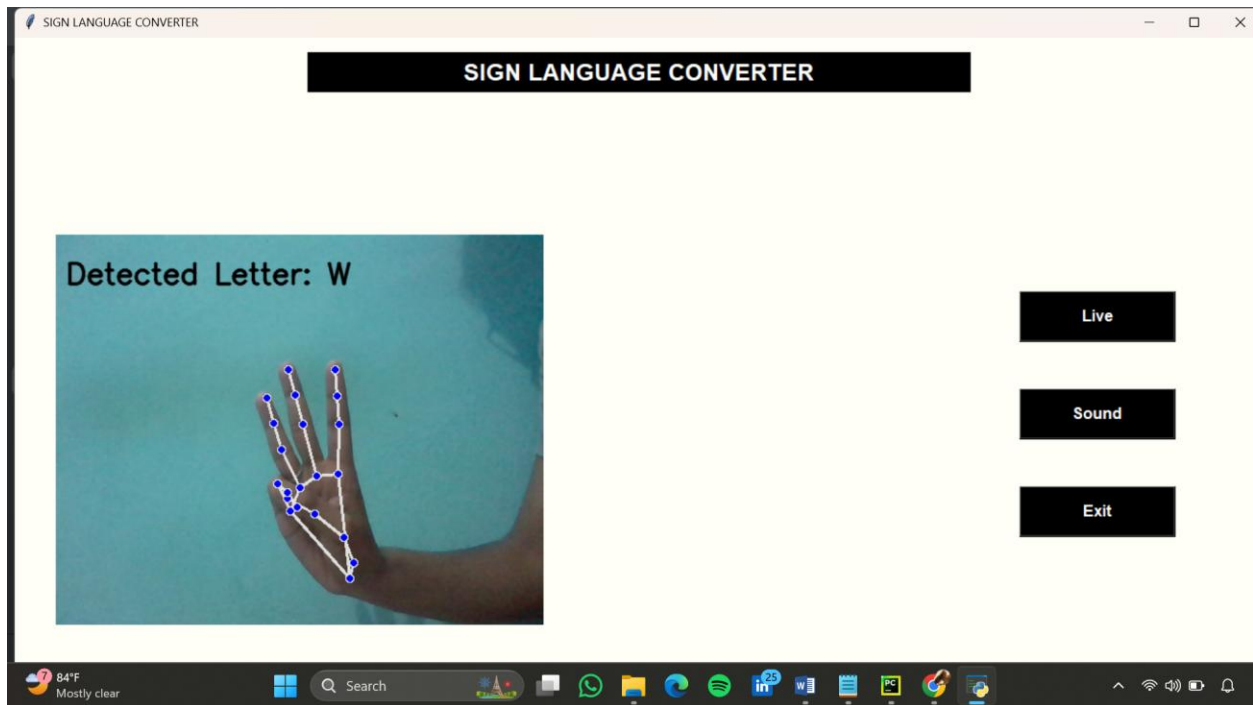
```
def live():  
    # ...  
    elif finger_fold_status == [True, True, True, False] and thumb_tip.y < thumb_base.y:  
        cshow = "Y"  
    # Z: (Static approximation: Index out, thumb crossed)  
    elif finger_fold_status == [False, True, True, True] and thumb_tip.x < index_tip.x:  
        cshow = "Z"  
    else:  
        cshow = ""  
  
    upCount.set(cshow)  
    mpDraw.draw_landmarks(rgb, hand, mpHands.HAND_CONNECTIONS)  
  
    # Overlay the detected letter on the video feed  
    cv2.putText(rgb, text=f"Detected Letter: {cshow}", org=(10, 50), cv2.FONT_HERSHEY_SIMPLEX, fontScale=1, color=cshow)
```

Run: eng
C:\Users\LENOVO\AppData\Local\Programs\Python\Python310\python.exe C:\Users\LENOVO\PycharmProjects\English_Letter_Recogniser\eng.py
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.

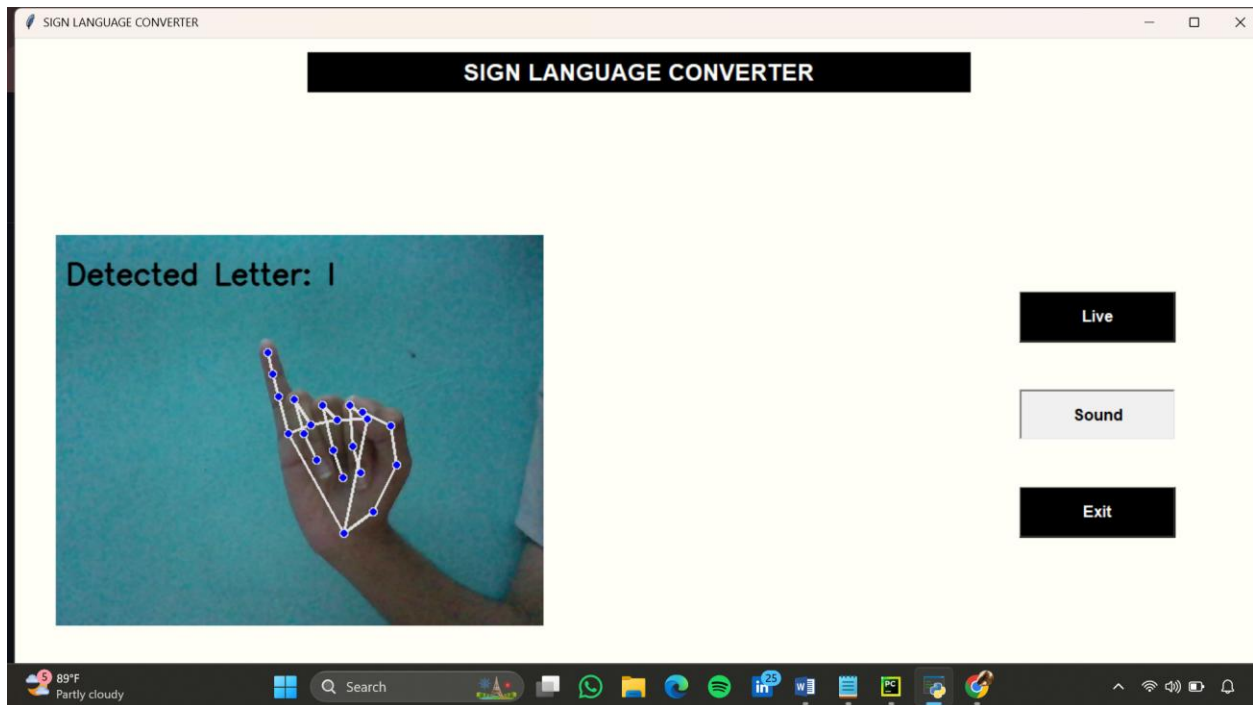
TRAIL – DETECTING IMAGE







TRAIL – GETTING OUTPUT IN VOICE FORM



USE CASES:

Assistive Technology

This system can be used as a tool to help people with hearing impairments communicate by translating sign language gestures into text and speech. It can act as a bridge for non-verbal communication in various settings like schools, hospitals, and public places.

Educational Tool for Learning Sign Language

This system can be used as an interactive way to learn sign language. It provides real-time feedback, which helps users practice and improve their hand gestures to match the correct sign language alphabet.

RESULT:

The sign language recognition system successfully detects hand gestures from a live webcam feed and converts them into corresponding letters of the alphabet. The system displays the detected letter on the screen and also provides audio feedback through a text-to-speech engine. It works in real-time with minimal delay and accurately recognizes a variety of hand gestures. This tool can be used for assistive communication and educational purposes in learning sign language.