PROJECT PROFILE

Project Name	INVISI-SCRIPT
Platform	Windows
Tool	Python(Version – 3.12.5) Visual Studio Code opencv numpy mediapipe easyocr streamlit
Project Duration	90 days
Internal Project Guide	Prof. Helly Patel
Submitted By	Muskan Marghani (22082701024)
Group Number	4
Course	BSc.(IT) Data Science Sem-5
University Name	Ganpat University
Submitted To	Department of Computer science
Academic Year	2024-2025
Objective	Introducing interactive way to interact with digital technology

INTRODUCTION

2.1 INTODUCTION

- Invisi-Script is an innovative project that transforms the way we interact with digital text by leveraging an air canvas.

- This system allows users to write in the air using gestures, which are then converted into text in real-time.

- It combines gesture recognition technology with text conversion algorithms to create a novel and intuitive method for text input and interaction.

- This innovative approach eliminates the need for physical input devices, such as keyboards or touchscreens, and provides a more natural and intuitive method for text creation.

- **Air Canvas Interface:**
  Users interact with a virtual canvas where they can write or draw in the air using hand gestures.

- **Gesture Recognition Technology:**
  Utilizes sensors and cameras to capture and interpret user gestures, converting them into digital commands.

- **Real-Time Text Conversion:**
  Transforms gestures into text instantly, allowing for immediate feedback and interaction.

2.2 PROBLEM STATEMENT

- **Inefficiency in Multitasking for Creative Professionals:**
  Designers, artists, and creators often switch between creative tools and text editors, disrupting their workflow when writing notes or annotations.

- **Lack of Efficient Contactless Input Methods in Hygiene-Sensitive Environments:**
  In hospitals, laboratories, or sterile work environments, physical contact with keyboards or touchscreens can risk contamination.

- **Limited Accessibility for Individuals with Physical Disabilities:**
  People with physical impairments may struggle with traditional text input methods like keyboards or touchscreens.

- **Ineffective Tools for Real-Time Collaboration and Annotation in Remote Work:**
  During online meetings or brainstorming sessions, professionals need to quickly add notes or annotations while multitasking, but switching between text tools and design platforms is cumbersome.

- **High Learning Curve and Low Engagement in Educational Tools:**
  Educational platforms often rely on traditional input methods like keyboards or writing tablets, which may not engage students effectively.

2.3 OBJECT

- Create a technology that allows users to input text without physical contact with any device.
- Design an air canvas interface where text can be written using hand gestures.
- Develop a gesture recognition engine that accurately detects and interprets user movements.
- Ensure text conversion occurs instantaneously.
- Create a user-friendly interface that is easy to interact with and visually clear.
- Adapt the system for different scenarios, including personal use, educational environments, and professional applications.
- Gather feedback from users to refine gesture recognition accuracy and overall system performance.

2.4 TECHNOLOGY STACK

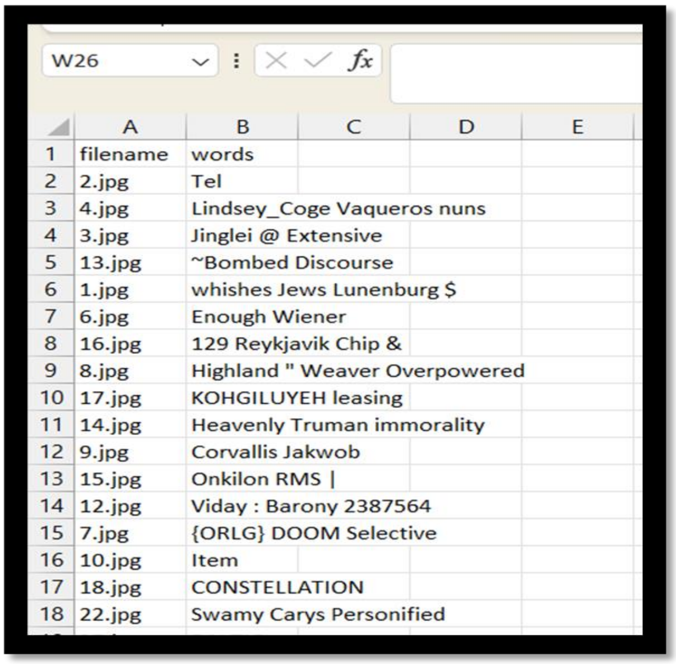
Platform	Windows 11 Version – 22H2
Language	Python – 3.12. 5
Integrated development environment	Visual Studio Code
Libraries	OpenCV Mediapipe Easyocr
Web Development Framework	Streamlit
Hardware	Camera

DATA COLLECTION

3.1 DATA SOURCE

- This Project is using easyocr library.
- EasyOCR, as the name suggests, is a Python package that allows computer vision developers to effortlessly perform Optical Character Recognition.
- This library supports over more than 85 languages.
- A varied dataset of text images is fundamental for getting started with EasyOCR.
- It helps the OCR system to handle a wide range of text styles, fonts, and orientations, enhancing the system's overall effectiveness.

3.2 DATA DESCRIPTION



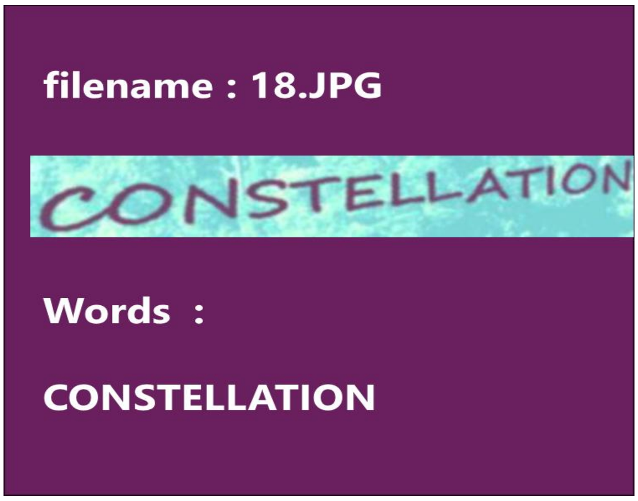
W26

	A	B	C	D	E
1	filename	words			
2	2.jpg	Tel			
3	4.jpg	Lindsey_Coge Vaqueros nuns			
4	3.jpg	Jinglei @ Extensive			
5	13.jpg	~Bombed Discourse			
6	1.jpg	whishes Jews Lunenburg $			
7	6.jpg	Enough Wiener			
8	16.jpg	129 Reykjavik Chip &			
9	8.jpg	Highland " Weaver Overpowered			
10	17.jpg	KOHGILUYEH leasing			
11	14.jpg	Heavenly Truman immorality			
12	9.jpg	Corvallis Jakwob			
13	15.jpg	Onkilon RMS |			
14	12.jpg	Viday : Barony 2387564			
15	7.jpg	{ORLG} DOOM Selective			
16	10.jpg	Item			
17	18.jpg	CONSTELLATION			
18	22.jpg	Swamy Carys Personified			

ROW: 9854

COLUM: 2

Example→



filename : 18.JPG

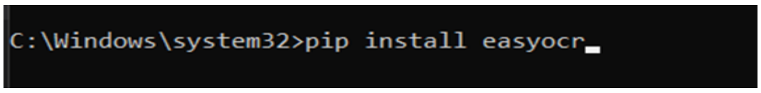
Words :

CONSTELLATION

3.3 DATA COLLECTION METHOD

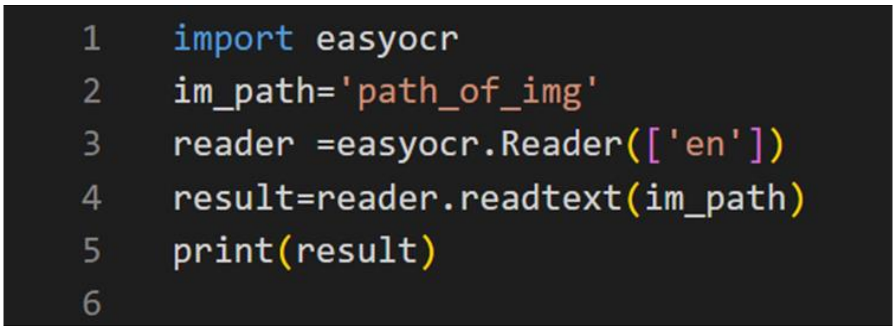
- Reference Website: https://github.com/JaidedAI/EasyOCR

- Install easyocr on machine by writing following command in Command prompt.

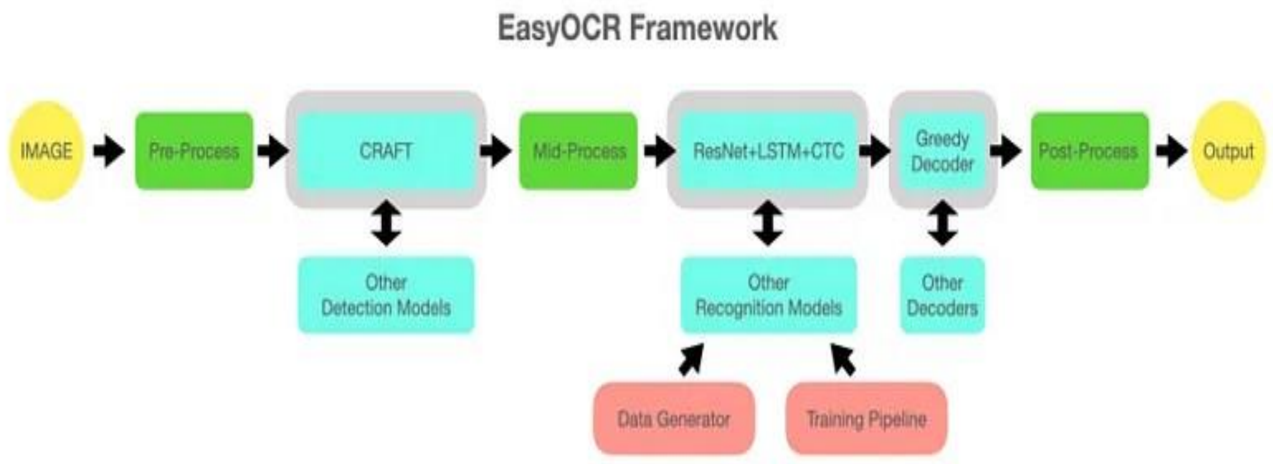


```
C:\Windows\system32>pip install easyocr
```

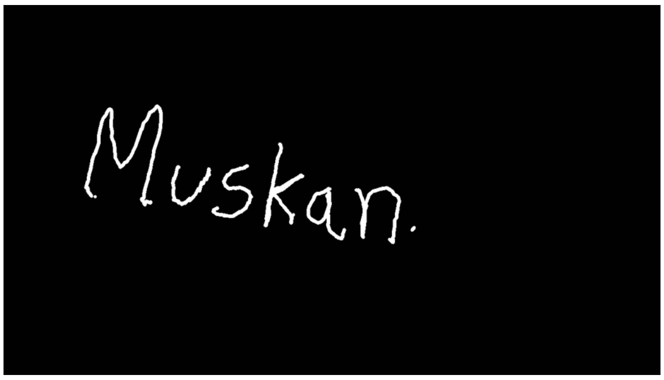
- Then Open the python file where you would like to use this library and write the following code.

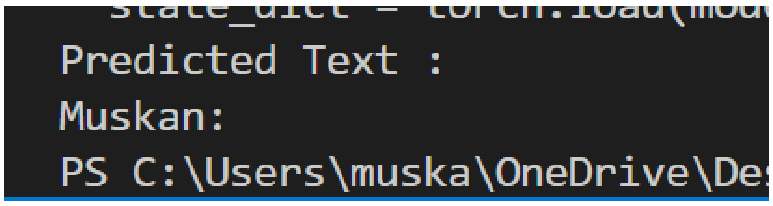


```
import easyocr
im_path='path_of_img'
reader =easyocr.Reader(['en'])
result=reader.readtext(im_path)
print(result)
```



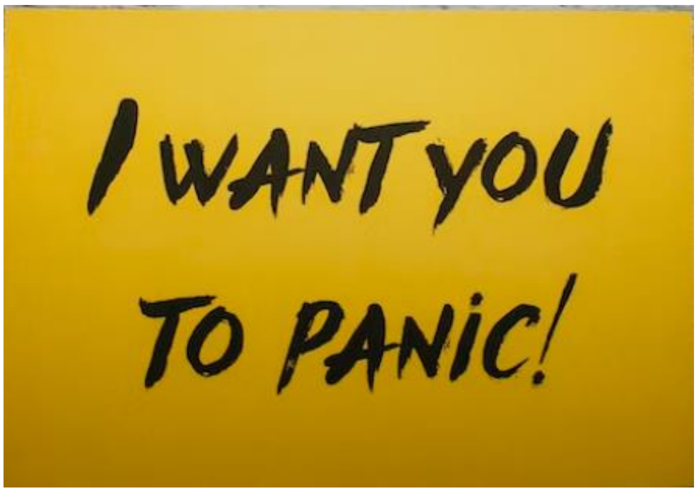
Example 1:

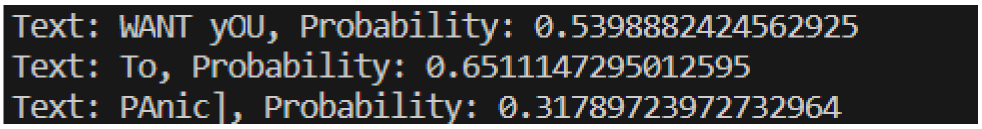




```
   state_dict = torch.load(mode
 Predicted Text :
 Muskan:
 PS C:\Users\muska\OneDrive\De
```

Example 2:





```
Text: WANT yOU, Probability: 0.5398882424562925
Text: To, Probability: 0.6511147295012595
Text: PAnic], Probability: 0.31789723972732964
```

METHODOLOGY

Invisi-Script leverages advanced computer vision and OCR techniques to allow users to write in the air and convert their handwriting into digital text. Below is an expanded methodology, providing comprehensive details for each step involved in the development process

- System Design and Framework Selection
- Gesture Recognition using Mediapipe
- Air Canvas Implementation
- Snapshot Capture and OCR Integration
- Text Editor Integration
- Testing and Validation
- Deployment and Optimization

This detailed methodology ensures the seamless development of Invisi-Script, covering everything from gesture recognition and air canvas implementation to OCR integration and user interaction. By focusing on accuracy, usability, and scalability, the project is positioned to deliver a robust and innovative solution for air-writing and text recognition.

4.1 SYSTEM DESIGN AND FRAMEWORK SELECTION

- **Objective**:

  Develop a real-time application for air-writing, gesture recognition, and text extraction.

- **Technology Stack**:

  Programming Language: Python for its extensive library support and ease of use.

  Key Libraries:

  - → **OpenCV**: For real-time video capture and image processing.
  - → **MediaPipe**: To track and classify hand gestures using pre-trained models.
  - → **EasyOCR**: For recognizing handwritten characters from saved images.
  - → **NumPy**: For efficient manipulation of the canvas as a multidimensional array.

- **Hardware**:

  A device with an integrated or external camera (e.g., laptop webcam or mobile phone camera).

- **User Interface**:

  Design an intuitive interface with a split-screen layout:

  - → **Left Section**: Air canvas for drawing gestures.
  - → **Right Section**: Text editor to display and edit recognized text.

4.2 GESTURE RECOGNITION USING MEDIAPIPE

- **Real-Time Video Capture**:

  Use **OpenCV** to continuously capture video frames from the camera.

- **Hand Detection**:

  Employ **MediaPipe Hands** to identify 21 key landmarks on the user's hand, such as fingertips, joints, and the wrist.

- **Gesture Classification**:

  Define gestures based on the relative positions of landmarks:

  - **Index Finger Up:**
    - → Detected when the index finger tip (landmark 8) is above the middle finger tip (landmark 12).
    - → Used for drawing on the canvas.
  - **Thumbs Up**:
    - → Identified when the thumb tip (landmark 4) is above the thumb base (landmark 2), with all other fingers closed.
    - → Clears the canvas.
  - **Pinky Finger Up**:
    - → Recognized when the Pinky finger tip (landmark 20) is above other fingers.
    - → Triggers canvas capture and saves the image.

- **Filtering and Accuracy**:

  Apply confidence thresholds provided by MediaPipe to avoid false positives.

  Smooth gesture tracking by averaging coordinates over consecutive frames.

4.3 AIR CANVAS IMPLEMENTATION

- **Canvas Initialization:**
  - Create a blank canvas using NumPy, where each pixel is an RGB value initialized to white.

- **Drawing Logic:**
  - Track the coordinates of the index finger tip (landmark 8).
  - Draw continuous lines by connecting successive points using OpenCV’s line () function.
  - Interpolate between points to ensure smooth strokes.

- **Clearing Mechanism:**
  - Detect the "thumbs up" gesture and reset the canvas by reinitializing the NumPy array to a blank state.

- **Canvas Movement:**
  - Allow users to move their hand across the canvas without drawing by detecting a specific gesture (e.g., index and middle fingers up).

4.4 SNAPSHOT CAPTURE AND OCR INTEGRATION

- **Capturing the Canvas:**
  - When the "Pinky finger up" gesture is detected, save the current canvas state as an image (captured_canvas.png) using OpenCV’s imwrite () function.

- **Optical Character Recognition (OCR):**
  - Load the saved image into EasyOCR’ s pre-trained model for character recognition.
  - Convert the canvas’s air-drawn text into a digital string.

- **Post-Processing OCR Output:**
  - Perform cleaning operations on the recognized text to correct minor errors:
    - → Remove unnecessary whitespace.
    - → Validate against a dictionary for common spelling errors.

4.5 TEXT EDITOR INTEGRATION

- **Text Insertion**:
    - Automatically insert the recognized text into a text editor within the interface.
- **Editor Features**:
    - Basic text formatting (bold, italic, underline).
    - Font size and style adjustment.
    - Options to save or export the text as a file.
- **User Interaction**:
    - Allow users to make manual corrections or enhancements to the recognized text.

4.6 TESTING AND VALIDATION

- **Gesture Recognition Testing**:
    - Test in various environments (lighting conditions, backgrounds) to ensure consistent gesture detection.

- **Canvas Performance**:
    - Evaluate responsiveness by measuring the delay between gesture input and its reflection on the canvas.

- **OCR Validation**:
    - Test with different handwriting styles and stroke sizes to assess accuracy.
    - Use metrics like precision, recall, and F1-score to measure recognition performance.

- **Usability Testing**:
    - Conduct user testing to evaluate ease of use and identify potential improvements in the interface.

4.7 DEPLOYMENT AND OPTIMIZATION

- **Deployment**:
  - Package the application for cross-platform use (desktop and mobile).

- **Optimization**:
  - Use efficient algorithms to reduce latency.
  - Minimize resource usage for smooth performance on low-spec devices.

MODEL DEVELOPMENT

Project Name: Invisi-Script

Objective: To develop a real-time system that allows users to write in the air using hand gestures and convert the handwritten input into digital text using OCR.

Problem statement: Handwriting and note-taking are critical tasks in many fields, but traditional tools like paper and pen may not always be convenient. Invisi-Script provides an innovative solution that enables users to write in the air using hand gestures and convert these gestures into editable digital text, eliminating the need for physical mediums.

- Scope of project
- Requirement
- Gesture recognition
- Air canvas
- Text editor
- Image to text
- User interface

5.1 SCOPE OF PROJECT

- **Gesture Recognition:** Detect and classify hand gestures in real-time to enable writing or triggering actions.
- **Air Canvas:** Visualize and record the user's handwriting in the air.
- **OCR Module:** Convert the captured air-writing into digital text.
- **User Interface:** Provide an intuitive platform for writing, editing, and saving the text.

5.2 REQUIREMENT

- **Software Requirements**

  Operating System: Windows 11

  Programming Language: Python 3.12.5

  IDE: Visual Studio Code

  Frameworks and Libraries:

  - OpenCV (real-time video capture and image processing)
  - MediaPipe (hand gesture recognition)
  - NumPy (canvas manipulation)
  - EasyOCR (text recognition)
  - Flask (optional for web-based deployment)

- **Hardware Requirements**

  Camera: Integrated or external webcam.

  Processor: Minimum i5 or equivalent.

  RAM: Minimum 8 GB.

5.3 GESTURE RECOGNIZATION

Input:

Real-time video frames from the camera.

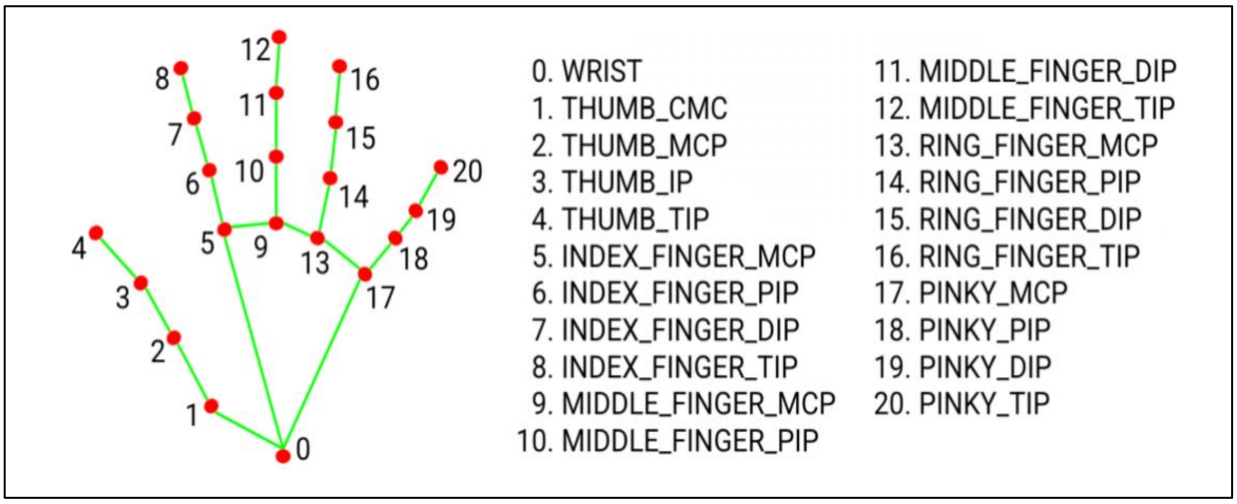
Processing:

Use MediaPipe Hands to detect and track 21 hand landmarks.

Apply rules based on landmark positions to classify gestures (e.g., "index finger up" for drawing).

Output:

Identified gesture and corresponding coordinates for drawing or action.



```
mp_hands = mp.solutions.hands
hands = mp_hands.Hands( max_num_hands=1, min_detection_confidence=0.8,
min_tracking_confidence=0.8)
mp_drawing = mp.solutions.drawing_utils
```

This code Above sets up hand detection using MediaPipe:

- Find one hand in a video.Only detect if it’s 80% sure.
- Track the hand’s movement with 80% confidence.
- Draw landmarks (fingers/joints) on the video.
- It detects and tracks one hand and shows its position.
- mp_hands = mp.solutions.hands:

This line gets the "hand detection" part of MediaPipe so that the program knows how to find hands in the video.

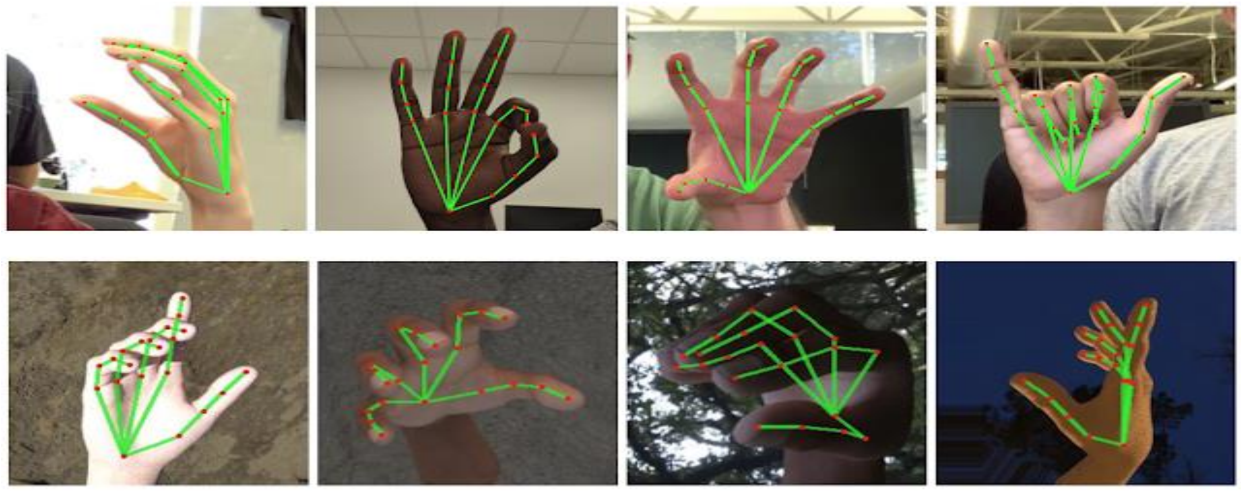
- hands = mp_hands.Hands(max_num_hands=1, min_detection_confidence=0.8, min_tracking_confidence=0.8):

  This creates the actual hand detector.

  - It will look for only one hand in the video.
  - It will be very sure(80% confidence or higher) that what it finds is really a hand.
  - It will also try to track the hand as it moves, with the same level of confidence (80%).

- mp_draw = mp.solutions.drawing_utils:

  This is used to draw the hand landmarks(like the positions of the fingers and joints) on the video, so you can see what the program detects.



```
if results.multi_hand_landmarks:
    for hand_landmarks in results.multi_hand_landmarks:
        mp_drawing.draw_landmarks(frame,
                                  hand_landmarks,mp_hands.HAND_CONNECTIONS)


         # Extract coordinates of key landmarks
         landmarks = hand_landmarks.landmark
          x_index_finger = int(landmarks[8].x * frame.shape[1])
          y_index_finger = int(landmarks[8].y * frame.shape[0])
          x_thumb = int(landmarks[4].x * frame.shape[1])
          y_thumb = int(landmarks[4].y * frame.shape[0])
          x_middle_finger = int(landmarks[12].x * frame.shape[1])
          y_middle_finger = int(landmarks[12].y * frame.shape[0])
          x_pinky = int(landmarks[20].x * frame.shape[1])
```

```
      y_pinky = int(landmarks[20].y * frame.shape[0])
           # Distance calculations to determine gesture actions
                distance_index_thumb  =  np.hypot(x_index_finger  -  x_thumb,
y_index_finger - y_thumb)
           distance_index_middle = np.hypot(x_middle_finger - x_index_finger,
y_middle_finger - y_index_finger)
           distance_index_pinky = np.hypot(x_pinky - x_index_finger, y_pinky -
y_index_finger)

if distance_index_thumb > 40  and distance_index_middle > 50
    and distance_index_pinky > 50:
     if x_prev == 0 and y_prev == 0:
            x_prev, y_prev = x_index_finger, y_index_finger
                       cv2.line(canvas,  (x_prev,  y_prev),  (x_index_finger,
y_index_finger), (0, 0, 0), 5)
             x_prev, y_prev = x_index_finger, y_index_finger
```

This code allows drawing on a canvas using only the index finger

1. **Gesture Detection:** It checks if only the index finger is raised while the others are down.
2. **First Position Check:** If it’s the first drawing action, it saves the index finger's current position.
3. **Drawing:** It draws a line from the previous finger position to the current position.
4. **Position Update:** Updates the previous position to the current index finger's location.
5. **Debounce Reset:** Resets a counter to manage gesture timing.

- **Gesture Detection:**
  It checks if the index finger is raised (distance between it and other fingers exceeds certain thresholds)

- **Setting Previous Position:**
  If this is the first drawing action (indicated by `x_prev` and `y_prev` being 0),
  it saves the current position of the index finger (`x_index_finger`, `y_index_finger`) as the previous position.

- **Writing:**

  It draws a line on the canvas from the previous position (`x_prev`, `y_prev`) to the current position of the index finger.

  The line is white (`(255, 255, 255)`) and has a thickness of 5 pixels.

- **Updating Previous Position:**

  It then updates `x_prev` and `y_prev` to the current index finger position for the next line drawing.

- **Resetting Gesture Debounce:**

  Finally, it resets a counter (`gesture_debounce`) that likely helps manage the timing of gestures to prevent accidental multiple detections.

```
 elif distance_index_thumb < 60 and distance_index_middle > 50 and
distance_index_pinky > 50:
         canvas = np.ones((frame.shape[0], frame.shape[1], 3), dtype=np.uint8)
* 255
          x_prev, y_prev = 0, 0  # Reset previous position to avoid connecting
lines
```

Above code is used to detect the gesture where thumb is up so that the canvas is cleared.

```
elif distance_index_pinky < 50:
                        cv2.imwrite(“captured_canvas.png”, canvas)
                        extracted_text = con(‘captured_canvas.png’)
                        print(extracted_text)
```

Above code is used to detect the gesture where pinky finger is up so that the text written on the canvas is submitted for further process.

```
else:  x_prev, y_prev = 0, 0  # Reset previous position
```

Above code is used to detect the gesture where 2 fingers are up and at that time nothing is written in the canvas.

5.4 AIR CANVAS

Input: Live Video.

Processing:

- Draw strokes on a blank canvas using OpenCV.
- Connect consecutive points to create smooth handwriting.

Output: Updated canvas displayed in real-time.

```
#Import required library
import streamlit as st
import cv2
import os
os.environ['TF_ENABLE_ONEDNN_OPTS’] ='0'
os.environ['KMP_DUPLICATE_LIB_OK’] ='True'
import mediapipe as mp
import numpy as np

canvas_placeholder = st.empty()
canvas = np.ones((480, 640, 3), dtype="uint8") * 255
x_prev, y_prev = 0, 0
```

- Canvas: A blank image where drawing happens.
- X_prev, y_prev: Track the previous position of the index finger to connect points for writing text.
- Gesture_debounce: Prevents unwanted writing actions after certain gestures.
- Canvas_placeholder: where our air canvas will be visible to user.

```
cap = cv2.VideoCapture(0)
    if not cap.isOpened():
        st.error("Could not open the camera.")
    else:
```

```
while cap.isOpened():
            ret, frame = cap.read()
            if not ret:
                st.error("Failed to capture image from camera.")
               break
```

- Initializing the video capture variable to work with the video as frame by frame.
- checking whether the camera is on or not and frame is being captured
- showing error message if couldn’t capture the video
- else taking frame by frame of video and working with it

```
 # Flip the frame for natural interaction
frame = cv2.flip(frame, 1)
frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
results = hands.process(frame_rgb)
```

This code part prepares a video frame for hand detection by mirroring it, reducing noise with a Gaussian blur, converting it to the expected color format, and then processing it to detect hand landmarks using mediapipe.

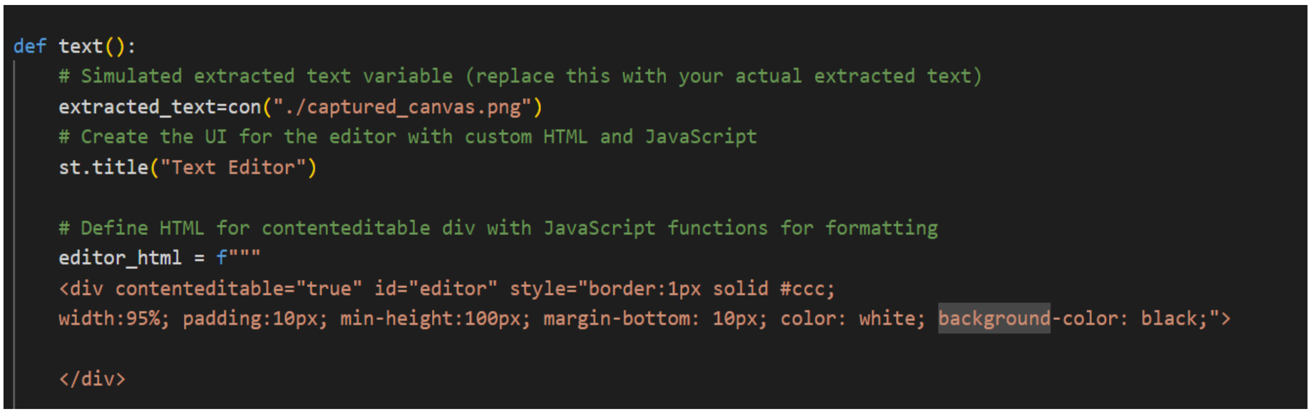
- Flipping the Frame for Natural Interaction
- Applying Gaussian Blur to Reduce Noise
- Converting the Blurred Frame from BGR to RGB
- Processing the Frame with Hand Detection

5.5 TEXT EDITOR

Libraries requirement:

Streamlit: Used for building web apps with a simple UI.

Custom Module (con): Extracts text from captured_canvas.png.



```
def text():
    # Simulated extracted text variable (replace this with your actual extracted text)
    extracted_text=con("./captured_canvas.png")
    # Create the UI for the editor with custom HTML and JavaScript
    st.title("Text Editor")

    # Define HTML for contenteditable div with JavaScript functions for formatting
    editor_html = f"""
    <div contenteditable="true" id="editor" style="border:1px solid #ccc;
    width:95%; padding:10px; min-height:100px; margin-bottom: 10px; color: white; background-color: black;">

    </div>
```

- extracted_text=con("./captured_canvas.png"):

  Extracts text from an image using the con function.

- Simulated Extracted Text:

  Placeholder for actual extracted text.

- st.title("Text Editor"):

  Displays the title on the Streamlit page.

- Custom HTML:

  Renders an editable text area using HTML and JavaScript.

- Contenteditable Div:

  Allows users to type directly in the web app.

```
  <br>
     <button style="color: white; background-color: #007bff; border: none;
padding: 5px 10px;
         cursor: pointer;" onclick="document.execCommand('bold', false,
'');">Bold</button>
     <button style="color: white; background-color: #007bff; border: none;
padding: 5px 10px;
        cursor: pointer;" onclick="document.execCommand('italic', false,
'');">Italic</button>
     <button style="color: white; background-color: #007bff; border: none;
padding: 5px 10px;
cursor: pointer;" onclick="document.execCommand('underline', false,
'');">Underline</button><br><br>
```

```
<button style="color: white; background-color: #007bff; border: none;
padding: 5px 10px;
  cursor: pointer;" onclick="appendText('{extracted_text}');">
  Add Text</button>
  <br>
  <br>
  <br>
  <button style="color: white; background-color: #007bff;
  border: none; padding: 5px 10px;
  cursor: pointer;" onclick="downloadContent()">Download
  as .html</button>
```

Bold, Italic, Underline Buttons: Uses JavaScript's document.execCommand to apply formatting.

'Add Text' Button: Inserts the extracted text into the editable area with the appendText function.

Download Button: Saves the content as an .html file using the downloadContent JavaScript function.

Blob: Creates a downloadable HTML file from the content.

HTML File: User can download their edited text.

```
        // Function to download the content of the editor as an HTML file
        function downloadContent() {{
            var content = document.getElementById('editor').innerHTML;
            var blob = new Blob([content], {{ type: 'text/html' }});
            var url = URL.createObjectURL(blob);
            var a = document.createElement('a');
            a.href = url;
            a.download = 'text_editor_output.html';
            document.body.appendChild(a);
            a.click();
            document.body.removeChild(a);
            URL.revokeObjectURL(url);
        }}
```

This function is responsible for downloading the current content of the editor as an HTML file.

Purpose: It retrieves the HTML content from an element with the ID editor. The innerHTML property allows you to access the content, including any HTML tags within that element.

Blob Creation: A Blob (Binary Large Object) is created from the content. It represents the data as a file-like object of immutable, raw data. The second argument specifies the MIME type, which is set to 'text/html' to indicate that the content is HTML.

Object URL Creation: A temporary URL representing the Blob is created using URL.createObjectURL(blob). This URL can be used as a link to download the Blob data.

Anchor Element Creation: An <a> (anchor) element is created programmatically. The href attribute is set to the previously created Blob URL, and the download attribute specifies the default name of the downloaded file (text_editor_output.html).

Download Trigger: The anchor element is added to the document body, and the click () method is invoked programmatically to trigger the download action.

Cleanup: After the download is initiated, the anchor element is removed from the document body to keep the DOM clean, and the Blob URL is revoked using URL.revokeObjectURL(url) to free up memory.

```
// Function to append text
       function appendText(newText) {{
           var editor = document.getElementById('editor');
           editor.innerText += newText;
           editor.focus(); // Set focus back to the editor
       }}
```

This function is designed to append new text to the existing content of the editor. Here's how it works

Getting the Editor Element: It retrieves the element with the ID editor and assigns it to the variable editor.

Appending Text: The new text passed as an argument (new Text) is appended to the existing text in the editor using the inner Text property, which ensures that the appended text does not contain any HTML formatting.

Setting Focus: The focus is set back to the editor, allowing the user to continue typing or interacting with it without needing to click back into the editor.

```
st.components.v1.html (editor_html, height=500)
```

Embedding Custom HTML: This feature is particularly useful when you want to include rich content or components that are not natively supported by Streamlit.

For example, you might want to include a custom text editor, a chart, or a widget created with HTML/CSS/JavaScript.

5.6 IMAGE TO TEXT

Input: Saved canvas image after a "snapshot" gesture.

Processing:

- Use EasyOCR to extract text from the image.
- Preprocess the text for noise reduction and formatting.

Output: Recognized text as a string.

Required library:

- Import easyocr
- Import useful library of python
- Using easyocr library to extract text from the image and converting it into digital text.

Function:

```
def con(im_path):
          words = []
          reader =Reader(['en'])
          result=reader.readtext(im_path)
          print("Predicted Text  is :")
          for i in result:
              words.append(i[1])
              sentence = ' '.join(words)
    return sentence
```

5.7 USER INTERFACE

- Making 2 columns
- First column covers 30% of screen and second column covers 70% of the screen.
- Make a title of website “Invisi script” which will be align in the centre with help of html integration
- In column one we will open text editor
- In column two we will open air canvas
- Setting width of column 2 as 70%.

```
col1,col2 =st.columns([1,1])

with col1:
    st.markdown(
    "<h1 style='text-align:left; font-size=56'>INVISI-SCRIPT</h1>",
    unsafe_allow_html=True
    )

col3,col4 = st.columns([3,7])

with col3:
    text()

with col4:
    air_canvas()
```

EVALUATION

6.1 PERFORMANCE MATRIX

- Out of every 10 words 9 are predicted correctly.
- Out of every 10 sentence 7 are predicted correctly.
- Out of every 15 special symbols all are predicted correctly.

- Precision: ~75–85%
- Recall: ~70–80%
- F1-Score: ~72–82%

RESULT AND ANALYSIS

7.1 KEY FINDINGS

1. The invisi-Script achieves high accuracy in recognizing predefined gestures using OpenCV and MediaPipe.

2. Minimal lag during real-time drawing, enhancing the user experience.

3. Effective implementation of gesture-based features like clearing the canvas or capturing snapshots.

4. High text recognition accuracy using EasyOCR, particularly for handwritten characters.

5. Small errors in cases of inconsistent handwriting or unusal characters.

6. Easy OCR achieves over 85% accuracy for legible English handwriting.

7. Designed for inclusivity, supporting users with limited mobility.

8. Simple and user-friendly interface.

9. Compatibility with various devices, including desktops and mobile platforms.

CONCLUSION

8.1 LIMITATION

- Difficulty detecting gestures under dim lighting conditions.
- Some words and sentence are predicted incorrectly

8.2 FUTURE WORK

- Enhance OCR capabilities for **multiple languages** and **complex handwriting styles**.
- Optimize gesture detection for **various lighting** and environmental conditions.
- Expand integration with **web and mobile applications** for broader usability.
- **Building MathAI:** Where user will write easy to complex mathematical equations in air and get its solution along with detailed explanation using generative AI.

8.3 SUMMERY OF ACHIEVEMENT

- **Gesture Recognition**

  The project successfully recognizes hand gestures like drawing with the index finger, clearing the canvas with a thumbs-up gesture, and saving snapshots by raising the pinky finger. These gestures work seamlessly, making the system intuitive and user-friendly.

- **Air Canvas Functionality**

  A responsive air canvas was developed, allowing users to draw in the air with minimal delay. Features like clearing the canvas and instantly capturing drawings were also implemented for added convenience.

- **Text Conversion**

  Easyocr was integrated to convert air-drawn text into digital format.
  The system performs well with clear handwriting, making it practical for everyday use.

- **User-Friendly Design**

  The platform is designed to be simple and accessible, catering even to users with limited technical expertise or mobility challenges.

- **Cross-Platform Support**

  The system is compatible with both desktop and mobile devices, making it versatile and widely applicable

- **Real-World Applications**

  The project demonstrated its potential for practical use in areas like education(e.g., remote learning), accessibility (e.g., helping individuals with mobility limitations), and creative arts (e.g., digital sketching).

REFRENCES

9.1 WEBSITE REFERENCE

1. https://medium.com/@nelsonizah/text-detection-in-images-with-easyocr-in-python-3e336c462c16
2. https://www.analyticsvidhya.com/blog/2021/06/text-detection-from-images-using-easyocr-hands-on-guide/
3. https://github.com/JaidedAI/EasyOCR
4. https://www.geeksforgeeks.org/opencv-python-tutorial/
5. https://opencv.org
6. https://www.datacamp.com/tutorial/opencv-tutorial
7. https://www.datacamp.com/tutorial/streamlit
8. https://streamlit.io
9. https://chat.openai.com
10. https://stackoverflow.com

9.2 YOUTUBE TUTORIALS

1. https://youtu.be/oXlwWbU8l2o?si=m7YnowWRymUxXmPp
2. https://youtu.be/C3-WnwzsaJA?si=0PigMFCHwrGPmdlo
3. https://www.youtube.com/watch?v=YzvMpvXyUfs