PROJECT - H

ika modaledudama.

Mundu urgent ga google open chei , and akkada “ vs code download” ani otthu mama.

ippudu ah vs code ni download chesko.okkasari adi download ipoyaka next steps chepta!

okasari download chesaka open cheyyandi.

oka welcome page ostadi adi close chei! ne desktop meda normal ga oka folder create chesi dani peru project ani pettu, and ah empty folder ni drag chesi vs code lo padesey!

ah tarwata ade chrome lo python download chesko!

like this ➖

STEP1:- Installing Python:

1. Open Chrome or any web browser.
2. In the search bar, type "Python download" and press Enter.
3. Click on the link that says "Download Python" (this should lead to the official Python website).
4. Choose the latest version of Python and download the installer.
5. Once the download is complete, open the downloaded file and follow the installation instructions:
    - On Windows: Run the installer, check "Add Python to PATH," and follow the setup wizard.
    - On Mac: Open the downloaded `.pkg` file and follow the installation instructions.
    - On Linux: Follow the provided instructions for your specific distribution.

Step 2: Creating a New Project

2.1 Creating a New Folder:

1. Open the File Explorer (Windows), Finder (Mac), or your file manager (Linux).
2. Navigate to a location where you want to create your project folder (e.g., Documents).
3. Right-click (or Control-click on Mac) and select "New Folder."
4. Name the folder `project`.

2.2 Opening the Folder in VS Code:

1. Open Visual Studio Code.
2. Click on "File" in the top menu and select "Open Folder."
3. Navigate to the `project` folder you created and open it.

Step 3: Installing Necessary Libraries

3.1 Opening the Terminal in VS Code:

1. In Visual Studio Code, click on "Terminal" in the top menu.
2. Select "New Terminal" from the dropdown.

3.2 Installing Libraries:

1. In the terminal, type the following commands and press Enter after each one:

Step 4: Create a Virtual Environment

1. **Open the Terminal or Command Prompt:**

   Open your terminal (Mac/Linux) or Command Prompt (Windows).

2. **Navigate to Your Project Directory:**

   Navigate to the directory where you want to create your project. You can use the `cd` command to change directories. For example:

   ```
   cd path/to/your/project
   ```

3. **Create the Virtual Environment:**

   To create a virtual environment, use the following command:

   ```
   python -m venv venv
   ```

- `python -m venv venv` tells Python to create a virtual environment named `venv` in your project directory. You can name your virtual environment anything you like, but `venv` is a common convention.

4. **Activate the Virtual Environment:**

   After creating the virtual environment, you need to activate it. The command to activate the virtual environment depends on your operating system.

   - **On Windows:**

     ```
     .\venv\Scripts\activate
     ```

   - **On Mac and Linux:**

     ```
     source venv/bin/activate
     ```

   When the virtual environment is activated, you should see the environment name (e.g., `(venv)`) at the beginning of your command prompt.

Step 5: Install Required Packages

Once the virtual environment is activated, you can install the required packages using `pip`. For this project, we will need OpenCV, MediaPipe, TensorFlow, and NumPy.

```
pip install opencv-python mediapipe tensorflow numpy
```

Step 6: Verify the Installation

To verify that the packages are installed correctly, you can use the following command:

```
pip list
```

This command will list all the packages installed in the virtual environment, and you should see `opencv-python`, `mediapipe`, `tensorflow`, and `numpy` among them.

Summary

1. **Create a Virtual Environment:**

   ```
   python -m venv venv
   ```

2. **Activate the Virtual Environment:**

   - **On Windows:**

     ```
     .\venv\Scripts\activate
     ```

   - **On Mac and Linux:**

     ```
     source venv/bin/activate
     ```

3. **Install Required Packages:**

   ```
   pip install opencv-python mediapipe tensorflow numpy
   ```

```
```

4. **Verify the Installation:**

   ```
   pip list
   ```

After completing these steps, you will have a virtual environment set up with all the necessary packages installed. Let me know once you have completed this setup, and we can proceed with the next steps of the project!

We'll name the file `gestures.py`.

Step 4: Create the Main Python Script

1. **Open Your Text Editor or IDE:**

   Open your preferred text editor or Integrated Development Environment (IDE) (e.g., VS Code, PyCharm).

2. **Create a New Python File:**

   Create a new file and save it as `gestures.py` in your project directory.

3. **Add Initial Code to Capture Video and Detect Hands:**

   We'll start by writing the initial code to capture video from the webcam and detect hands using MediaPipe.

Step 7: Create the Python File

Step-by-Step Implementation

1. Import Libraries

First, we need to import the necessary libraries. This includes OpenCV for video capture and MediaPipe for hand tracking.

```
# Step 1: Import Libraries
import cv2
import mediapipe as mp

# Explanation:
# cv2: OpenCV library for video capture and processing
# mediapipe: Library for hand tracking and landmark detection
```

2. Initialize MediaPipe Hands

Initialize the MediaPipe Hands solution to detect and track hand landmarks.

```
# Step 2: Initialize MediaPipe Hands
mp_hands = mp.solutions.hands
hands = mp_hands.Hands()
mp_draw = mp.solutions.drawing_utils

# Explanation:
# mp_hands: Accesses the hands solution in MediaPipe
# hands: Initializes the hands module for hand detection
# mp_draw: Utility to draw hand landmarks on the frames
```

3. Initialize Video Capture

Set up video capture using the default webcam.

```
# Step 3: Initialize Video Capture
cap = cv2.VideoCapture(0)
```

```
# Explanation:
# cap: Captures video from the default camera (usually the we
bcam)
```

4. Capture and Process Each Frame

Capture video frames in a loop, process each frame to detect hands, and draw hand landmarks.

```
# Step 4: Capture and Process Each Frame
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    # Flip the frame horizontally for a later selfie-view dis
play
    frame = cv2.flip(frame, 1)
    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    results = hands.process(frame_rgb)

    if results.multi_hand_landmarks:
        for hand_landmarks in results.multi_hand_landmarks:
            mp_draw.draw_landmarks(frame, hand_landmarks, mp_
hands.HAND_CONNECTIONS)

    # Explanation:
    # ret, frame: Reads a frame from the video capture
    # cv2.flip(frame, 1): Flips the frame horizontally for a
mirror view
    # cv2.cvtColor(frame, cv2.COLOR_BGR2RGB): Converts the fr
ame from BGR to RGB
    # hands.process(frame_rgb): Processes the frame to detect
hand landmarks
```

```
    # mp_draw.draw_landmarks(frame, hand_landmarks, mp_hands.
HAND_CONNECTIONS): Draws the detected hand landmarks on the f
rame
```

5. Display the Frame

Display the processed video frames with hand landmarks.

```
# Step 5: Display the Frame
cv2.imshow('Hand Gesture Recognition', frame)

if cv2.waitKey(1) & 0xFF == ord('q'):
    break

# Explanation:
# cv2.imshow('Hand Gesture Recognition', frame): Displays the
frame with hand landmarks
# cv2.waitKey(1): Waits for 1 millisecond for a key press. If
'q' is pressed, the loop breaks
```

6. Release Resources

Release the webcam and close all OpenCV windows when the loop is exited.

```
# Step 6: Release Resources
cap.release()
cv2.destroyAllWindows()

# Explanation:
# cap.release(): Releases the webcam
# cv2.destroyAllWindows(): Closes all OpenCV windows
```

Full Code

Here is the full code combining all the steps:

```
# Step 1: Import Libraries
import cv2
import mediapipe as mp

# Step 2: Initialize MediaPipe Hands
mp_hands = mp.solutions.hands
hands = mp_hands.Hands()
mp_draw = mp.solutions.drawing_utils

# Step 3: Initialize Video Capture
cap = cv2.VideoCapture(0)

# Step 4: Capture and Process Each Frame
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    # Flip the frame horizontally for a later selfie-view dis
play
    frame = cv2.flip(frame, 1)
    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    results = hands.process(frame_rgb)

    if results.multi_hand_landmarks:
        for hand_landmarks in results.multi_hand_landmarks:
            mp_draw.draw_landmarks(frame, hand_landmarks, mp_
hands.HAND_CONNECTIONS)
```

```
    # Step 5: Display the Frame
    cv2.imshow('Hand Gesture Recognition', frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Step 6: Release Resources
cap.release()
cv2.destroyAllWindows()
```

Running the `gestures.py` Script

1. **Open Your Terminal or Command Prompt:**

   Make sure your virtual environment is activated. If it's not, activate it using the following commands:

   - **On Windows:**

     ```
     .\venv\Scripts\activate
     ```

   - **On Mac and Linux:**

     ```
     source venv/bin/activate
     ```

2. **Navigate to Your Project Directory:**

   Use the `cd` command to navigate to the directory where your `gestures.py` file is located. For example:

```
cd path/to/your/project
```

3. **Run the Script:**

   Run the script by using the following command:

   ```
   python gestures.py
   ```

What to Expect:

- The script will start capturing video from your webcam.
- If hand landmarks are detected, they will be displayed on the video feed.
- The video feed will be shown in a window titled 'Hand Gesture Recognition'.
- To exit the video feed, press the 'q' key on your keyboard.

Troubleshooting:

- **If you encounter errors, check the following:**
  - Ensure your virtual environment is activated.
  - Ensure all required packages (`opencv-python`, `mediapipe`, `tensorflow`, `numpy`) are installed.
  - Ensure your webcam is properly connected and accessible.

ippudu best part:-

Step-by-Step Implementation

We'll now add the following features:

1. **Extract Hand Landmarks:**
    - Extract the coordinates of hand landmarks detected by MediaPipe.
2. **Implement Gesture Recognition:**
    - Recognize specific hand gestures using a simple logic or a pre-trained model.
3. **Overlay Text Based on Gestures:**
    - Display corresponding text over the detected hand gesture.

Step 1: Extract Hand Landmarks

We'll extract the hand landmark coordinates and use them to recognize gestures.

Code Block to Extract Hand Landmarks

Add this code inside the `if results.multi_hand_landmarks:` loop to extract the landmarks:

```
# Initialize list to store landmark coordinates
landmark_list = []

for id, lm in enumerate(hand_landmarks.landmark):
    # Get the coordinates
    h, w, c = frame.shape
    cx, cy = int(lm.x * w), int(lm.y * h)
    landmark_list.append([cx, cy])
```

Step 2: Implement Gesture Recognition

For simplicity, let's recognize two basic gestures:

1. **Open Hand (Palm)**
2. **Pointing Up**

We'll use a simple logic based on the position of specific landmarks.

Code Block for Gesture Recognition

Add this code to recognize gestures based on the extracted landmarks:

```
if len(landmark_list) != 0:
    # Example logic for gesture recognition
    # Open Hand (Palm) Gesture
    if landmark_list[4][1] < landmark_list[3][1] and landmark_list[8][1] < landmark_list[6][1]:
        gesture = "Syntax Sarcasm"
    # Pointing Up Gesture
    elif landmark_list[4][1] > landmark_list[3][1] and landmark_list[8][1] < landmark_list[6][1]:
        gesture = "Join the Workshop"
    else:
        gesture = None
```

Step 3: Overlay Text Based on Gestures

Overlay the recognized gesture text on the video frame.

Code Block for Text Overlay

Add this code to display the gesture text on the frame:

```
# Display the corresponding text
if gesture:
    cv2.putText(frame, gesture, (landmark_list[0][0] - 50, landmark_list[0][1] - 50),
                cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2,
cv2.LINE_AA)
```

Full Code with Gesture Recognition and Text Overlay

Here is the full updated code:

```
# Step 1: Import Libraries
import cv2
import mediapipe as mp

# Step 2: Initialize MediaPipe Hands
mp_hands = mp.solutions.hands
hands = mp_hands.Hands()
mp_draw = mp.solutions.drawing_utils

# Step 3: Initialize Video Capture
cap = cv2.VideoCapture(0)

# Step 4: Capture and Process Each Frame
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    # Flip the frame horizontally for a later selfie-view dis
play
    frame = cv2.flip(frame, 1)
    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    results = hands.process(frame_rgb)

    if results.multi_hand_landmarks:
        for hand_landmarks in results.multi_hand_landmarks:
            mp_draw.draw_landmarks(frame, hand_landmarks, mp_
hands.HAND_CONNECTIONS)
```

```
            # Initialize list to store landmark coordinates
            landmark_list = []
            for id, lm in enumerate(hand_landmarks.landmark):
                # Get the coordinates
                h, w, c = frame.shape
                cx, cy = int(lm.x * w), int(lm.y * h)
                landmark_list.append([cx, cy])

            # Gesture recognition logic
            if len(landmark_list) != 0:
                # Example logic for gesture recognition
                # Open Hand (Palm) Gesture
                if landmark_list[4][1] < landmark_list[3][1]
and landmark_list[8][1] < landmark_list[6][1]:
                    gesture = "Syntax Sarcasm"
                # Pointing Up Gesture
                elif landmark_list[4][1] > landmark_list[3]
[1] and landmark_list[8][1] < landmark_list[6][1]:
                    gesture = "Join the Workshop"
                else:
                    gesture = None

                # Display the corresponding text
                if gesture:
                    cv2.putText(frame, gesture, (landmark_lis
t[0][0] - 50, landmark_list[0][1] - 50),
                                cv2.FONT_HERSHEY_SIMPLEX, 2,
(0, 255, 0), 3, cv2.LINE_AA)  # Increased fontScale and thick
ness

    # Step 5: Display the Frame
    cv2.imshow('Hand Gesture Recognition', frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

```
# Step 6: Release Resources
cap.release()
cv2.destroyAllWindows()
```

Next Steps:

1. **Run the `gestures.py` script to test the gesture recognition and text overlay.**
2. **Verify that the recognized gestures display the corresponding text.**

What to Expect:

When you run the `gestures.py` script:

1. **Video Feed with Hand Detection:**
    - The script will open a window displaying the video feed from your webcam.
    - Hand landmarks will be drawn on detected hands.
2. **Gesture Recognition and Text Overlay:**
    - When specific hand gestures are recognized, corresponding text will be displayed on the screen near the hand.

How to Test:

Run the Script:

- Open your terminal or command prompt.
- Ensure your virtual environment is activated.
- Run the script using:

```
python gestures.py
```

1. **Perform the Gestures:**
    - **Open Hand (Palm) Gesture:**
        - Hold your hand open with the palm facing the camera.
        - Ensure your fingers are spread apart.
        - The text "Syntax Sarcasm" should appear near your hand.
    - **Pointing side Gesture:**
        - Point your index finger upwards with the other fingers folded.
        - Ensure your hand is pointing the first finger to the side with all fingers closed

2. **Exit the Script:**
    - To stop the script, press the 'q' key on your keyboard.

Gestures to Keep:

For the initial implementation, we have two simple gestures:

1. **Open Hand (Palm) Gesture:**
    - **Description:** Hand open with fingers spread apart.
    - **Displayed Text:** "Syntax Sarcasm"
    - **Use Case:** This gesture is common and easy to recognize, making it ideal for a demo.
2. **Pointing side Gesture:**
    - **Description:** Index finger pointing upwards with other fingers folded.
    - **Use Case:** This gesture is distinct and easy to perform, making it suitable for recognition.

Summary:

1. **Run the script:**

   ```
   python gestures.py
   ```

enjoy pandago!

output photo pettu marchipoku!

unta malla

byeeeeee!