# hand gesture recognition

In [8]:

```
pip install --upgrade pip
```

```
Requirement already satisfied: pip in ./myenv/lib/python3.7/site-packa
ges (23.2)
Collecting pip
  Obtaining dependency information for pip from https://files.pythonho
sted.org/packages/50/c2/e06851e8cc28dcad7c155f4753da8833ac06a5c704c109
313b8d5a62968a/pip-23.2.1-py3-none-any.whl.metadata (https://files.pyt
honhosted.org/packages/50/c2/e06851e8cc28dcad7c155f4753da8833ac06a5c70
4c109313b8d5a62968a/pip-23.2.1-py3-none-any.whl.metadata)
  Using cached pip-23.2.1-py3-none-any.whl.metadata (4.2 kB)
Using cached pip-23.2.1-py3-none-any.whl (2.1 MB)
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 23.2
    Uninstalling pip-23.2:
      Successfully uninstalled pip-23.2
Successfully installed pip-23.2.1
Note: you may need to restart the kernel to use updated packages.
```

In [2]:

```
import mediapipe as mp
import cv2
import numpy as np
import uuid
import os
```

In [3]:

```
# installed the required dependencies for your environment.
#drawing_utils module provides utility functions to draw landmarks and connectio
#hands module contains the Hand Tracking solution provided by mediapipe, which
mp_drawing = mp.solutions.drawing_utils
mp_hands = mp.solutions.hands
```

In [15]:

```
cap = cv2.VideoCapture(0)

with mp_hands.Hands(min_detection_confidence=0.8, min_tracking_confidence=0.5) a
    while cap.isOpened():
        ret, frame = cap.read()

        # BGR 2 RGB
        image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

        # Flip on horizontal
        image = cv2.flip(image, 1)

        # Set flag
        image.flags.writeable = False

        # Detections
        results = hands.process(image)

        # Set flag to true
        image.flags.writeable = True

        # RGB 2 BGR
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

        # Detections
        print(results)

        # Rendering results
        if results.multi_hand_landmarks:
            for num, hand in enumerate(results.multi_hand_landmarks):
                mp_drawing.draw_landmarks(image, hand, mp_hands.HAND_CONNECTIONS
                                        mp_drawing.DrawingSpec(color=(121, 22, 7
                                        mp_drawing.DrawingSpec(color=(250, 44, 2
                                         )


        cv2.imshow('Hand Tracking', image)

        if cv2.waitKey(10) & 0xFF == ord('q'):
            break

cap.release()
cv2.destroyAllWindows()
```

```
<class 'mediapipe.python.solution_base.SolutionOutputs'>
<class 'mediapipe.python.solution_base.SolutionOutputs'>
<class 'mediapipe.python.solution_base.SolutionOutputs'>
<class 'mediapipe.python.solution_base.SolutionOutputs'>
```

In [4]:

```python
import os

directory_name = 'Output Images'

if not os.path.exists(directory_name):
    os.mkdir(directory_name)
    print(f"Directory '{directory_name}' created.")
else:
    print(f"Directory '{directory_name}' already exists.")

```

```
Directory 'Output Images' already exists.
```

In [5]:

```
cap = cv2.VideoCapture(0)

with mp_hands.Hands(min_detection_confidence=0.8, min_tracking_confidence=0.5) a
    while cap.isOpened():
        ret, frame = cap.read()

        # BGR 2 RGB
        image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

        # Flip on horizontal
        image = cv2.flip(image, 1)

        # Set flag
        image.flags.writeable = False

        # Detections
        results = hands.process(image)

        # Set flag to true
        image.flags.writeable = True

        # RGB 2 BGR
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

        # Detections
        print(results)

        # Rendering results
        if results.multi_hand_landmarks:
            for num, hand in enumerate(results.multi_hand_landmarks):
                mp_drawing.draw_landmarks(image, hand, mp_hands.HAND_CONNECTIONS
                                        mp_drawing.DrawingSpec(color=(121, 22, 7
                                        mp_drawing.DrawingSpec(color=(250, 44, 2
                                         )

        # Save our image
        cv2.imwrite(os.path.join('Output Images', '{}.jpg'.format(uuid.uuid1()))
        cv2.imshow('Hand Tracking', image)

        if cv2.waitKey(10) & 0xFF == ord('q'):
            break

cap.release()
cv2.destroyAllWindows()
```

```
<class 'mediapipe.python.solution_base.SolutionOutputs'>
<class 'mediapipe.python.solution_base.SolutionOutputs'>
<class 'mediapipe.python.solution_base.SolutionOutputs'>
<class 'mediapipe.python.solution_base.SolutionOutputs'>
<class 'mediapipe.python.solution_base.SolutionOutputs'>
```

In [6]:

```
current_directory = os.getcwd()
print("Current directory:", current_directory)
```

```
Current directory: /Users/myyntiimac
```

```
## cap = cv2.VideoCapture(0): This line initializes a video capture object cap using the OpenCV library. It's used to capture frames from the default camera (index 0).

with mp_hands.Hands(min_detection_confidence=0.8, min_tracking_confidence=0.5) as hands:: This line uses a context manager to create a Hands instance from the mp_hands module, which is the hand tracking solution from the mediapipe library. The min_detection_confidence and min_tracking_confidence parameters set the confidence thresholds for hand detection and tracking, respectively. The context manager ensures that the resources are properly managed and released when done.

while cap.isOpened():: This starts a loop that will continue as long as the capture is open (video frames are being received).

ret, frame = cap.read(): This line reads a video frame from the capture object. The ret variable will be True if a frame is successfully read, and False if there are no more frames to read. The frame variable holds the image data of the current frame.

image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB): Converts the frame from the BGR color space (used by OpenCV) to the RGB color space.

image = cv2.flip(image, 1): Flips the image horizontally (creates a mirror effect).

image.flags.writeable = False: Temporarily sets the writeable flag of the image array to False, preventing accidental modification.

results = hands.process(image): Processes the image using the Hands instance to detect and track hand landmarks. The results are stored in the results variable.

image.flags.writeable = True: Resets the writeable flag of the image array to True to allow modifications.

image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR): Converts the image back from RGB to BGR color space.

print(results): Prints the results of hand detection and tracking, including the landmarks' positions and their confidence scores.

if results.multi_hand_landmarks:: Checks if any hands were detected and tracked in the current frame.

for num, hand in enumerate(results.multi_hand_landmarks):: Iterates over each detected hand and its corresponding landmarks.

```

```
mp_drawing.draw_landmarks(image, hand, mp_hands.HAND_CONNECTIONS, ...): Draws landmarks and connections on the image using the draw_landmarks function from mp_drawing. The DrawingSpec parameters determine the appearance of the drawn landmarks and connections.

cv2.imshow('Hand Tracking', image): Displays the modified image with landmarks and connections.

if cv2.waitKey(10) & 0xFF == ord('q'):: Waits for a key press for a specified delay (10 milliseconds) and checks if the key pressed is 'q'. If 'q' is pressed, the loop breaks.

cap.release(): Releases the video capture object, freeing up resources.

cv2.destroyAllWindows(): Closes all OpenCV windows.

This code captures video frames from the default camera, processes them using the mediapipe hand tracking solution, and displays the frames with hand landmarks and connections drawn on them. The loop continues until the user presses the 'q' key.




User
os.mkdir('Output Images')
ChatGPT
The line os.mkdir('Output Images') is used to create a new directory named "Output Images" in the current working directory. This directory can be used to store or save images generated or processed during the program's execution.

Here's what the line does:

os: This refers to the Python built-in os module, which provides a way to interact with the operating system (e.g., creating directories, working with files).

.mkdir('Output Images'): This calls the mkdir function of the os module to create a new directory. The argument 'Output Images' specifies the name of the directory you want to create.
```