ChatGPT 4o

Share

Save as PDF

RU

hi i am a university student taking a module on "edge computing and analytics". i need your help guiding me through a lab. the lab is on "video analytics with raspberry pi using web camera"

Memory updated

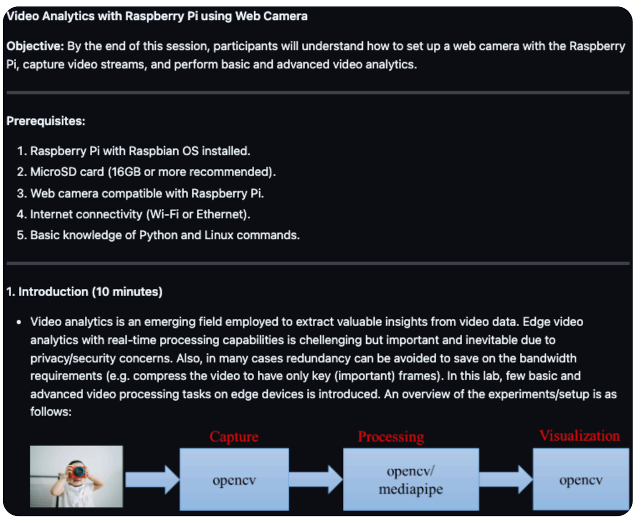
Hey YR! That sounds like a fun and hands-on lab. I'd be happy to guide you through it. Could you share a bit more about what the lab requires? For example:

- Do you already have a Raspberry Pi set up?
- Are you using a specific programming language (like Python)?
- What kind of video analytics are you working on (e.g. motion detection, object recognition, face detection)?
- Do you have the lab instructions or objectives?

Once I know that, I can walk you through it step by step.

okay i will be providing you the lab sheet in sections

Perfect! Send over the first section when you're ready, and we'll tackle it together step by step.


Video Analytics with Raspberry Pi using Web Camera

Objective: By the end of this session, participants will understand how to set up a web camera with the Raspberry Pi, capture video streams, and perform basic and advanced video analytics.

Prerequisites:

1. Raspberry Pi with Raspbian OS installed.
2. MicroSD card (16GB or more recommended).
3. Web camera compatible with Raspberry Pi.
4. Internet connectivity (Wi-Fi or Ethernet).
5. Basic knowledge of Python and Linux commands.

---

1. Introduction (10 minutes)

- Video analytics is an emerging field employed to extract valuable insights from video data. Edge video analytics with real-time processing capabilities is chellenging but important and inevitable due to privacy/security concerns. Also, in many cases redundancy can be avoided to save on the bandwidth requirements (e.g. compress the video to have only key (important) frames). In this lab, few basic and advanced video processing tasks on edge devices is introduced. An overview of the experiments/setup is as follows:

Great, thanks for sharing the first section! Here's a breakdown and how we'll approach this lab:

🧠 Objective Summary

By the end of the lab, you'll:

- Connect a webcam to a Raspberry Pi
- Capture and stream video
- Perform **basic and advanced** video analytics (e.g., object detection or motion tracking)

🛠️ Prerequisites Checklist

Make sure you have the following ready:

- ✅ Raspberry Pi with **Raspbian OS**
- ✅ **16GB+ MicroSD card**
- ✅ USB webcam that works with Raspberry Pi
- ✅ Internet access (Wi-Fi or Ethernet)
- ✅ Familiarity with **Python** and basic **Linux commands**

Let me know if you're missing any of these so I can help you set it up.

🧪 Lab Structure Overview

The lab is divided into three stages using OpenCV and Mediapipe:

1. **Capture** – Use `opencv` to grab video from your webcam.
2. **Processing** – Analyze the video using `opencv` and/or `mediapipe` .
3. **Visualization** – Display results using `opencv` (e.g., bounding boxes, FPS counters).

We can proceed as follows:

1. Set up the environment (installing OpenCV & Mediapipe).
2. Test basic video capture from webcam using OpenCV.

3. Try simple processing tasks (e.g., grayscale conversion, face detection).
4. Move on to advanced processing (e.g., hand/pose tracking using Mediapipe).
5. Show the processed video stream in a window.

Send over the next section of the lab sheet when you're ready!

2. Setting up the Raspberry Pi (10 minutes)

- Booting up the Raspberry Pi.
- Setting up Wi-Fi/Ethernet.
- System updates:

```
sudo apt update
sudo apt upgrade
```

3. Connecting and Testing the Web Camera (5 minutes)

- Please ensure the web camera is working and proceed to subsequent steps.

4. Introduction to real-time video processing on raspberry pi (20 minutes)

- **[Important!] Set up and activate a virtual environment named "video" for this experiment (to avoid conflicts in libraries) as below. You can also reuse the virtual environment "image" as we are employing opencv and mediapipe libraries for video analytics**

```
sudo apt install python3-venv
python3 -m venv video
source video/bin/activate
```

- Installing OpenCV:

```
pip install opencv-python
```

Awesome, thanks for sending the next part! Here's a step-by-step guide based on what this section covers:

🔧 2. Setting up the Raspberry Pi (10 minutes)

If you haven't already:

1. **Boot the Pi** – Power it on and log in.
2. **Connect to Internet** – Use either Wi-Fi or Ethernet.
3. **Update the system:**

bash Copy Edit

```bash
sudo apt update
sudo apt upgrade
```

🎥 3. Connecting and Testing the Web Camera (5 minutes)

Make sure the webcam is properly plugged in.

To test if it's detected:

```bash
ls /dev/video*
```

You should see something like `/dev/video0`. You can also try:

```bash
v4l2-ctl --list-devices
```

Or quickly test using Python:

```python
import cv2

cap = cv2.VideoCapture(0)
if cap.isOpened():
    print("Webcam is working!")
else:
    print("Webcam not detected.")
cap.release()
```

⚙️ 4. Real-time Video Processing Setup (20 minutes)

✅ Virtual Environment Setup (important)

This keeps your libraries clean and separate from system Python:

```bash
sudo apt install python3-venv
python3 -m venv video
source video/bin/activate
```

You should now see `(video)` in your terminal prompt.

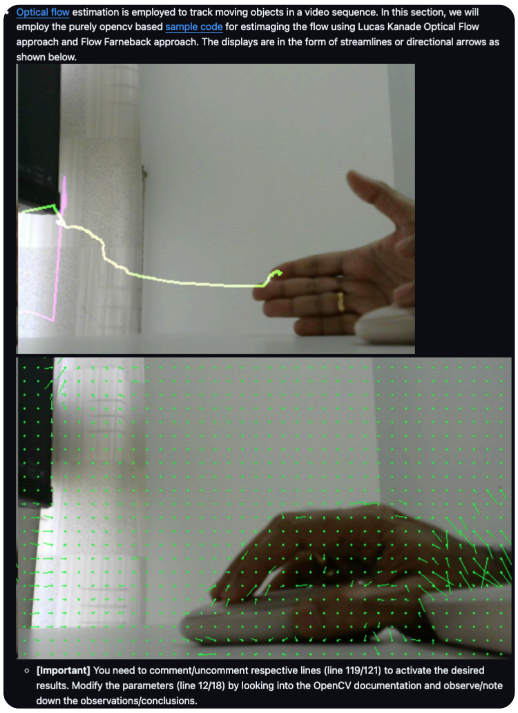
✅ Install OpenCV:

```bash
```

```
pip install opencv-python
```

> Let me know if you face any errors during installation — sometimes Raspberry Pi can be fussy with packages depending on the version.

🟢 Once you've completed all that, let me know and I'll help with the next step (which will probably include writing a script to display the video stream). Ready when you are!



sample code:
#%% OpenCV based real-time optical flow estimation and tracking
# Ref: https://github.com/daisukelab/cv_opt_flow/tree/master
import numpy as np
import cv2

#%% Generic Parameters
color = np.random.randint(0,255,(100,3)) # Create some random colors

#%% Parameters for Lucas Kanade optical flow approach [Ref: https://cseweb.ucsd.edu//classes/sp02/cse252/lucaskanade81.pdf]
# params for ShiTomasi corner detection
feature_params = dict(maxCorners = 100,

```
                    qualityLevel = 0.3,
                    minDistance = 7,
                    blockSize = 7 )

# Parameters for lucas kanade optical flow
lk_params = dict( winSize  = (15,15),
                  maxLevel = 2,
                  criteria = (cv2.TERM_CRITERIA_EPS |
cv2.TERM_CRITERIA_COUNT, 10, 0.03))


#%% Flow estimation is always with respect to previous frame
and the below code is required to be done for the first time as
called from main
def set1stFrame(frame):

   # Converting to gray scale
   frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

   p0 = cv2.goodFeaturesToTrack(frame_gray, mask=None,
**feature_params) # Corner detection using
https://docs.opencv.org/3.4/d4/d8c/tutorial_py_shi_tomasi.html

   # Create a mask image for drawing purposes
   mask = np.zeros_like(frame)

   return frame_gray,mask,p0

#%% Lucas Kanade optical flow approach [Ref:
https://cseweb.ucsd.edu//classes/sp02/cse252/lucaskanade81.
pdf]
def LucasKanadeOpticalFlow (frame,old_gray,mask,p0):

   # Converting to gray scale
   frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)



   # calculate optical flow
   if (p0 is None or len(p0) ==0):
      p0 = np.array([[50, 50], [100, 100]],
dtype=np.float32).reshape(-1, 1, 2)
   p1, st, err = cv2.calcOpticalFlowPyrLK(old_gray, frame_gray,
                              p0, None, **lk_params)

   if p1 is not None:
```

```
        # Select good points (skip no points to avoid errors)
        good_new = p1[st==1]
        good_old = p0[st==1]

        # draw the tracks
        for i, (new,old) in enumerate(zip(good_new,good_old)):
            a,b = new.ravel()
            c,d = old.ravel()
            mask = cv2.line(mask, (int(a),int(b)), (int(c),int(d)),
color[i].tolist(), 2)
            frame_gray = cv2.circle(frame_gray, (int(a),int(b)), 5,
color[i].tolist(), -1)
        img = cv2.add(frame, mask)

        # Now update the previous frame and previous points
        old_gray = frame_gray.copy()
        p0 = good_new.reshape(-1,1,2)

    return img,old_gray,p0


#%% Computes a dense optical flow using the Gunnar
Farneback's algorithm.
step = 16

def DenseOpticalFlowByLines(frame, old_gray):

    # Converting to gray scale
    frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    h, w = frame_gray.shape[:2]
    y, x = np.mgrid[step//2:h:step, step//2:w:step].reshape(2,-1)

    flow = cv2.calcOpticalFlowFarneback(old_gray, frame_gray,
None,
                                        0.5, 3, 15, 3, 5, 1.2, 0)  #
https://docs.opencv.org/4.x/dc/d6b/group__video__track.html#g
a5d10ebbd59fe09c5f650289ec0ece5af

    fx, fy = flow[y,x].T

    # Plot the streamlines
    lines = np.vstack([x, y, x+fx, y+fy]).T.reshape(-1, 2, 2)
    lines = np.int32(lines + 0.5)
    cv2.polylines(frame, lines, 0, (0, 255, 0))
    for (x1, y1), (x2, y2) in lines:
```

```
        cv2.circle(frame, (x1, y1), 1, (0, 255, 0), -1)
    return frame


#%% Open CV Video Capture and frame analysis
cap = cv2.VideoCapture(0)

# Check if the webcam is opened correctly
if not cap.isOpened():
    raise IOError("Cannot open webcam")

firstframeflag = 1

# The loop will break on pressing the 'q' key
while True:
    try:

        if (firstframeflag):
            # Capture one frame
            ret, frame = cap.read()

            old_gray,mask,p0 = set1stFrame(frame)

            firstframeflag = 0

        # Capture one frame
        ret, frame = cap.read()

        img = DenseOpticalFlowByLines(frame, old_gray)

        #img,old_gray,p0 =
LucasKanadeOpticalFlow(frame,old_gray,mask,p0)

        cv2.imshow("Optical Flow", img)

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

    except KeyboardInterrupt:
        break

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

Nice! This section is all about **real-time optical flow estimation** – a cool method to track motion in video. Here's how it breaks down and how you can experiment with it:

🎯 Goal of This Section

Use **Lucas-Kanade** and **Farneback** methods from OpenCV to:

- Track moving features
- Visualize motion with arrows or streamlines

🧪 Two Optical Flow Approaches in the Script

1. Lucas-Kanade Optical Flow (Sparse, based on tracking key points)

- Controlled by this line (currently commented):

  python   Copy   Edit

  ```python
  #img,old_gray,p0 = LucasKanadeOpticalFlow(frame,old_gray,mask,p0)
  ```

- Tracks good features across frames and draws colored trails

2. Farneback Dense Optical Flow (uses the whole image grid)

- Currently active in the script:

  python   Copy   Edit

  ```python
  img = DenseOpticalFlowByLines(frame, old_gray)
  ```

🛠️ How to Switch Between the Two

Edit these lines in the loop:

python   Copy   Edit

```python
# To use Farneback (default)
img = DenseOpticalFlowByLines(frame, old_gray)
```

```python
# To use Lucas-Kanade instead
# img,old_gray,p0 = LucasKanadeOpticalFlow(frame,old_gray,mask,p0)
```

Just **comment/uncomment** to choose the method.

🔧 Try Modifying These Parameters

To get different motion tracking effects:

Line 12:

```python
color = np.random.randint(0,255,(100,3))
```

→ Change to fewer colors like `(50,3)` or use fixed colors.

Lines 18-22: Lucas-Kanade params

```python
feature_params = dict( maxCorners = 100, qualityLevel = 0.3, ...)
```

- `maxCorners` → fewer points = less visual clutter
- `qualityLevel` → lower value = more features detected
- `winSize` in `lk_params` → affects optical flow smoothness

✅ To Run the Script

Make sure:

- You're in your `video` virtual environment
- You've installed OpenCV with `pip install opencv-python`
- Then run:

  ```bash
  python3 your_script_name.py
  ```

Press `q` to quit the webcam stream.

Let me know once you've tried running this or if you'd like help tweaking the parameters and observing results!

could you explain the code to me in detail?

Absolutely! Let’s walk through the **optical flow estimation script** line by line so you fully understand what’s going on.

🧠 Overview

This Python script uses OpenCV to:

1. Capture video from your webcam
2. Estimate motion between video frames using either:
   - **Lucas-Kanade Optical Flow** (sparse points)
   - **Farneback Optical Flow** (dense grid)
3. Visualize the motion using arrows or lines.

🔍 Section-by-Section Breakdown

📦 Imports

python Copy Edit

```python
import numpy as np
import cv2
```

- `numpy` is used for numerical operations and managing point arrays.
- `cv2` is OpenCV — the main computer vision library used here.

🎨 Color Initialization

```python
color = np.random.randint(0,255,(100,3))
```

- Creates 100 random RGB colors.
- Used to draw motion lines in Lucas-Kanade output.

🎯 Lucas-Kanade Parameters

Shi-Tomasi Corner Detection

```python
feature_params = dict(
    maxCorners = 100,
    qualityLevel = 0.3,
    minDistance = 7,
    blockSize = 7 )
```

- Finds up to 100 strong corners in the image to track.
- These are features the algorithm will try to follow across frames.

LK Optical Flow Params

```python
lk_params = dict(
    winSize  = (15,15),
    maxLevel = 2,
    criteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03))
```

- `winSize` is the size of the search window.

- `maxLevel` = number of image pyramid levels to search across.
- `criteria` = when to stop the iterative search (either after 10 steps or if error < 0.03).

🧱 Function: `set1stFrame()`

```python
def set1stFrame(frame):
    frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    p0 = cv2.goodFeaturesToTrack(frame_gray, mask=None, **feature_params)
    mask = np.zeros_like(frame)
    return frame_gray, mask, p0
```

- Converts the first frame to grayscale.
- Detects corners (`p0`) to track.
- Creates a blank mask for drawing motion lines.

🏃 Function: `LucasKanadeOpticalFlow()`

```python
def LucasKanadeOpticalFlow(frame, old_gray, mask, p0):
    frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    p1, st, err = cv2.calcOpticalFlowPyrLK(old_gray, frame_gray, p0, None,
**lk_params)
```

- Converts the new frame to grayscale.
- Calculates the new positions (`p1`) of the points `p0` using optical flow.
- `st` tells which points were found successfully.

Then it draws motion lines:

```python
    for i, (new, old) in enumerate(zip(good_new, good_old)):
        ...
```

```
        mask = cv2.line(...)
        frame_gray = cv2.circle(...)
```

And finally updates the tracked points:

python Copy Edit

```python
    p0 = good_new.reshape(-1,1,2)
```

🌊 Function: `DenseOpticalFlowByLines()`

python Copy Edit

```python
def DenseOpticalFlowByLines(frame, old_gray):
    frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

- Converts current frame to grayscale.
- Computes a dense optical flow between old and new frames using:

  python Copy Edit

  ```python
  cv2.calcOpticalFlowFarneback()
  ```

Then it creates a grid and draws arrows:

python Copy Edit

```python
    lines = np.vstack([x, y, x+fx, y+fy]).T.reshape(-1, 2, 2)
    cv2.polylines(frame, lines, 0, (0, 255, 0))
```

📷 Webcam Loop

python Copy Edit

```python
cap = cv2.VideoCapture(0)
```

- Starts capturing from your default webcam.

python

```python
if not cap.isOpened():
    raise IOError("Cannot open webcam")
```

- Check for webcam issues.

🔁 Main Loop

python

```python
while True:
    ...
    if firstframeflag:
        ret, frame = cap.read()
        old_gray, mask, p0 = set1stFrame(frame)
        firstframeflag = 0
```

- Reads the first frame and initializes tracking points.

Then for each frame:

python

```python
    ret, frame = cap.read()
```

- Captures a new frame

Use **either**:

python

```python
    img = DenseOpticalFlowByLines(...)
```

or:

python

```python
    img, old_gray, p0 = LucasKanadeOpticalFlow(...)
```

And display it:

python

```
    cv2.imshow("Optical Flow", img)
```

Quit when `q` is pressed:

python Copy Edit

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

🧹 Cleanup

python Copy Edit

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

- Releases webcam and closes OpenCV windows.

✅ Summary

Part	What it Does
`set1stFrame()`	Initializes grayscale + features
`LucasKanadeOpticalFlow()`	Tracks sparse points using LK
`DenseOpticalFlowByLines()`	Tracks all pixels using Farneback
Main Loop	Reads webcam and displays flow

Let me know if you'd like:

- Help running this on your Pi
- To visualize both methods side-by-side
- To add extra features like FPS counter or recording

We can enhance this further too!

Modify the parameters (line 12/18) by looking into the OpenCV documentation and observe/note down the observations/conclusions.

line 12:
feature_params = dict(maxCorners = 100,
qualityLevel = 0.3,
minDistance = 7,
blockSize = 7)

line 18:
lk_params = dict(winSize = (15,15),
maxLevel = 2,
criteria = (cv2.TERM_CRITERIA_EPS |
cv2.TERM_CRITERIA_COUNT, 10, 0.03))

Great! Let's go through the parameters in `feature_params` (line 12) and `lk_params` (line 18), and suggest meaningful modifications to see how they affect the optical flow results visually.

🧪 1. `feature_params` (Line 12)

This controls how the **Shi-Tomasi corner detection** selects points to track.

🔧 Original:

python

```python
feature_params = dict(
    maxCorners = 100,
    qualityLevel = 0.3,
    minDistance = 7,
    blockSize = 7 )
```

📘 Parameter Descriptions:

Parameter	Purpose	Impact
`maxCorners`	Max number of features to detect	More points = denser motion trails

Parameter	Purpose	Impact
`qualityLevel`	Accept only corners with quality > x * best corner	Lower = more (possibly bad) points
`minDistance`	Minimum distance between corners	Higher = more spread out
`blockSize`	Size of neighborhood used for corner detection	Bigger = smoother, less sensitive

---

✅ Modified Version 1: Detect more features, even if lower quality

```python
feature_params = dict(
    maxCorners = 200,
    qualityLevel = 0.01,
    minDistance = 5,
    blockSize = 3 )
```

Expected Observations:

- **More tracking points**, even on low-texture regions.
- May result in **messier** output (some poor-quality points).
- Useful for detecting subtle movement in hand/fingers.

✅ Modified Version 2: Detect fewer, more reliable features

```python
feature_params = dict(
    maxCorners = 50,
    qualityLevel = 0.5,
    minDistance = 10,
    blockSize = 7 )
```

Expected Observations:

- **Fewer points**, but more **stable** and accurate.
- Great for **clear visual trails** with less clutter.

🧪 2. `lk_params` (Line 18)

This controls the **Lucas-Kanade optical flow** tracking algorithm.

🔧 Original:

python Copy Edit

```python
lk_params = dict(
    winSize  = (15,15),
    maxLevel = 2,
    criteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03))
```

📘 Parameter Descriptions:

Parameter	Purpose	Impact
`winSize`	Size of the window to search in	Larger = more tolerant to motion
`maxLevel`	Pyramid levels	Higher = better with fast/large motion
`criteria`	When to stop refining	Controls speed vs accuracy

✅ Modified Version 1: Handle larger and faster motion

python Copy Edit

```python
lk_params = dict(
    winSize  = (31,31),
    maxLevel = 4,
    criteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 20, 0.01))
```

Expected Observations:

- Can **better track quick hand movements**.
- Slightly **slower performance** due to more calculations.

✅ Modified Version 2: Make it lightweight (fast and responsive)

python Copy Edit

```
lk_params = dict(
    winSize  = (10,10),
    maxLevel = 1,
    criteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 5, 0.1))
```

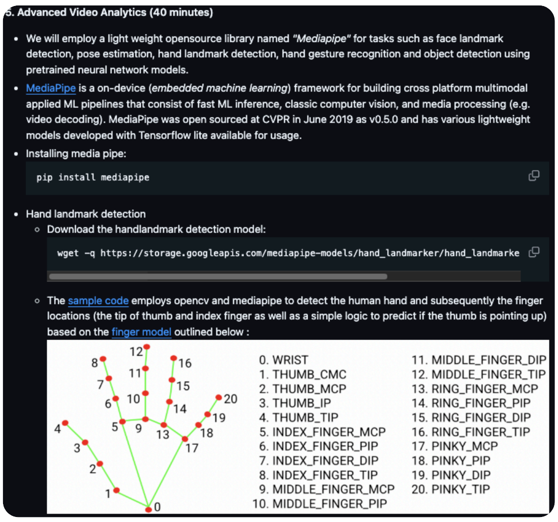
Expected Observations:

- **Faster**, but **may lose accuracy** on fast motion.
- Might skip frames or jump around when object moves quickly.

🧾 Suggested Experiment Plan

Config	Description	Expected Result
V1	More features, sensitive tracker	Dense trails, messy but responsive
V2	Fewer, more reliable features	Sparse trails, more stable
V3	Large window & pyramid	Good for fast motion, slower
V4	Small window & pyramid	Fast, but less accurate tracking

Would you like me to generate a version of the code that cycles through these configs so you can compare them quickly on your Pi?



5. Advanced Video Analytics (40 minutes)

- We will employ a light weight opensource library named *"Mediapipe"* for tasks such as face landmark detection, pose estimation, hand landmark detection, hand gesture recognition and object detection using pretrained neural network models.
- MediaPipe is a on-device (*embedded machine learning*) framework for building cross platform multimodal applied ML pipelines that consist of fast ML inference, classic computer vision, and media processing (e.g. video decoding). MediaPipe was open sourced at CVPR in June 2019 as v0.5.0 and has various lightweight models developed with Tensorflow lite available for usage.
- Installing media pipe:

```
pip install mediapipe
```

- Hand landmark detection
  - Download the handlandmark detection model:

```
wget -q https://storage.googleapis.com/mediapipe-models/hand_landmarker/hand_landmarke
```

  - The sample code employs opencv and mediapipe to detect the human hand and subsequently the finger locations (the tip of thumb and index finger as well as a simple logic to predict if the thumb is pointing up) based on the finger model outlined below :

sample code:
#%% Reference:
https://github.com/googlesamples/mediapipe/tree/main/examples/hand_landmarker/raspberry_pi

```
# Download hand land mark detector model wget -q
https://storage.googleapis.com/mediapipe-
models/hand_landmarker/hand_landmarker/float16/1/hand_lan
dmarker.task
import cv2
import mediapipe as mp
from mediapipe.tasks import python
from mediapipe.tasks.python import vision

#%% Parameters
numHands = 2 # Number of hands to be detected
model = 'hand_landmarker.task' # Model for finding the hand
landmarks Download using wget -q
https://storage.googleapis.com/mediapipe-
models/hand_landmarker/hand_landmarker/float16/1/hand_lan
dmarker.task
minHandDetectionConfidence = 0.5 # Thresholds for detecting
the hand
minHandPresenceConfidence = 0.5
minTrackingConfidence = 0.5
frameWidth = 640
frameHeight = 480

# Visualization parameters
MARGIN = 10  # pixels
FONT_SIZE = 1
FONT_THICKNESS = 1
HANDEDNESS_TEXT_COLOR = (88, 205, 54) # vibrant green


#%% Create an HandLandmarker object.
base_options =
python.BaseOptions(model_asset_path=model)
options = vision.HandLandmarkerOptions(
        base_options=base_options,
        num_hands=numHands,

min_hand_detection_confidence=minHandDetectionConfidenc
e,

min_hand_presence_confidence=minHandPresenceConfidenc
e,
        min_tracking_confidence=minTrackingConfidence)
detector =
vision.HandLandmarker.create_from_options(options)
```

```
#%% Open CV Video Capture and frame analysis
cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, frameWidth)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, frameHeight)

# Check if the webcam is opened correctly
if not cap.isOpened():
    raise IOError("Cannot open webcam")

# The loop will break on pressing the 'q' key
while True:
    try:
        # Capture one frame
        ret, frame = cap.read()

        frame = cv2.flip(frame, 1) # To flip the image to match with
camera flip

        # Convert the image from BGR to RGB as required by the
TFLite model.
        rgb_image = cv2.cvtColor(frame,
cv2.COLOR_BGR2RGB)


        # Run hand landmarker using the model.
        mp_image =
mp.Image(image_format=mp.ImageFormat.SRGB,
data=rgb_image)
        detection_result = detector.detect(mp_image)

        hand_landmarks_list = detection_result.hand_landmarks

        #handedness_list = detection_result.handedness # Could
be used to check for which hand

        # Loop through the detected hands to visualize.
        for idx in range(len(hand_landmarks_list)):
            hand_landmarks = hand_landmarks_list[idx]

            # Detect Thumb and draw a circle on the thumb tip
            x = int(hand_landmarks[4].x * frame.shape[1]) # Index 4
corresponds to the thump tip as from
https://developers.google.com/mediapipe/solutions/vision/hand
_landmarker
            y = int(hand_landmarks[4].y * frame.shape[0])
```

```
            cv2.circle(frame, (x, y), 5, (0, 255, 0), -1)


            # Detect Thumb and draw a circle on the index finger
tip
            x = int(hand_landmarks[8].x * frame.shape[1]) # Index 8
corresponds to the index finger tip as from
https://developers.google.com/mediapipe/solutions/vision/hand
_landmarker
            y = int(hand_landmarks[8].y * frame.shape[0])
            cv2.circle(frame, (x, y), 5, (0, 255, 0), -1)

            # Define a threshold for thumb is up/down and display
when thums up
            threshold = 0.1
            thumb_tip_y = hand_landmarks[4].y
            thumb_base_y = hand_landmarks[1].y # Index 1
corresponds to the thump base as from
https://developers.google.com/mediapipe/solutions/vision/hand
_landmarker
            thums_up = thumb_tip_y < thumb_base_y - threshold

            if thums_up:
                cv2.putText(frame, 'Thumb Up', (10,30),
                            cv2.FONT_HERSHEY_DUPLEX,
                            FONT_SIZE,
HANDEDNESS_TEXT_COLOR, FONT_THICKNESS,
cv2.LINE_AA)

            cv2.imshow('Annotated Image', frame)

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

    except KeyboardInterrupt:
        break

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

Awesome — now we're diving into **advanced video analytics** using **MediaPipe** and **OpenCV** to detect **hand landmarks** and interpret gestures like a “thumbs up” in real-time. Let's break down this entire section so it's crystal clear for you.

🧠 Goal of This Part

- Use a **pretrained hand landmark detection model** to identify 21 key hand points.
- Use logic to **track the thumb and index finger tips**.
- Detect if the **thumb is pointing up**, and display a message accordingly.

🧩 Code Breakdown

🧰 Imports

python Copy Edit

```python
import cv2
import mediapipe as mp
from mediapipe.tasks import python
from mediapipe.tasks.python import vision
```

- `cv2` = OpenCV, used for webcam and drawing.
- `mediapipe` = Machine learning vision library from Google.
- `mediapipe.tasks` = Used for loading the pre-trained model and running detection.

⚙️ Parameters

python Copy Edit

```python
numHands = 2
model = 'hand_landmarker.task'
```

- Track up to **2 hands**.
- Using a **.task file** (TFLite-based pretrained model) for hand landmarks.

python Copy Edit

```
minHandDetectionConfidence = 0.5
minHandPresenceConfidence = 0.5
minTrackingConfidence = 0.5
```

- These thresholds filter low-confidence predictions.

🧠 Setup HandLandmarker

python

Copy Edit

```python
base_options = python.BaseOptions(model_asset_path=model)
options = vision.HandLandmarkerOptions(...)
detector = vision.HandLandmarker.create_from_options(options)
```

- Initializes the MediaPipe hand detector with all the parameters.

🎥 Webcam Capture + Preprocessing

python

Copy Edit

```python
cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, frameWidth)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, frameHeight)
```

- Starts the webcam and sets resolution to 640×480.

🔁 Main Loop

python

Copy Edit

```python
frame = cv2.flip(frame, 1)
rgb_image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
```

- Flips horizontally (to match mirror view) and converts to RGB for MediaPipe input.

python Copy Edit

```python
mp_image = mp.Image(...)
detection_result = detector.detect(mp_image)
```

- MediaPipe performs hand detection on the RGB image.

🖐 Process Hand Landmarks

python Copy Edit

```python
hand_landmarks_list = detection_result.hand_landmarks
```

- Gets a list of `21 landmarks` per hand (if any are detected).

📌 Landmark Visualization:

python Copy Edit

```python
hand_landmarks[4]  → Thumb tip
hand_landmarks[8]  → Index finger tip
hand_landmarks[1]  → Thumb base (CMC)
```

Each of these has:

python Copy Edit

```python
hand_landmarks[i].x  # Horizontal position (normalized: 0.0 to 1.0)
hand_landmarks[i].y  # Vertical position (same)
```

Converted to pixel coordinates:

python Copy Edit

```python
x = int(hand_landmarks[i].x * frame.shape[1])
y = int(hand_landmarks[i].y * frame.shape[0])
```

👍 Thumb-Up Detection

Copy Edit

python

```python
thums_up = thumb_tip_y < thumb_base_y - threshold
```

- Logic: if **thumb tip is higher (less y)** than the base, it's considered **"thumbs up."**
- If `thums_up == True`, display "Thumb Up" on screen.

🪄 Final Output

python

```python
cv2.imshow('Annotated Image', frame)
```

- Shows the video feed with landmarks and text drawn on top.

🔎 Hand Landmark Index Reference

From the image in your lab:

- `0` = Wrist
- `4` = Thumb tip
- `8` = Index finger tip
- `20` = Pinky tip
  Etc.

You can use this map to track any finger or part of the hand!

✅ Summary of What's Happening

Step	What it Does
🧠 Load Model	Downloads and loads TFLite hand landmark model
🎥 Webcam	Captures and flips each frame
✋ Detect Hands	Uses MediaPipe to find up to 21 landmarks per hand

Step	What it Does
📍 Draw Landmarks	Plots thumb and index tip positions
👍 Detect Gesture	Detects "Thumb Up" if thumb tip is higher than base

---

Would you like:

- To add more gesture logic? (e.g., two-finger pinch, open palm?)
- To save the annotated video output?
- To show FPS or overlay more info?

Just let me know and I can help you build that in too!

Modify the code to show all the 21 finger points and observe the same while moving the hand.

Great! The instruction here is to **modify the code to visualize all 21 finger landmarks**, not just the thumb and index finger.

Here's how you can update your code to display **all 21 finger points**:

🧩 Replace this block:

(Currently in your loop)

python

```python
# Detect Thumb and draw a circle on the thumb tip
x = int(hand_landmarks[4].x * frame.shape[1])
y = int(hand_landmarks[4].y * frame.shape[0])
cv2.circle(frame, (x, y), 5, (0, 255, 0), -1)

# Detect Index Finger tip
x = int(hand_landmarks[8].x * frame.shape[1])
y = int(hand_landmarks[8].y * frame.shape[0])
cv2.circle(frame, (x, y), 5, (0, 255, 0), -1)
```

✅ With this:

```python
# Loop through all 21 landmarks and draw them
for i, landmark in enumerate(hand_landmarks):
    x = int(landmark.x * frame.shape[1])
    y = int(landmark.y * frame.shape[0])
    cv2.circle(frame, (x, y), 5, (0, 255, 0), -1)
    cv2.putText(frame, str(i), (x + 5, y - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.4,
(0, 255, 255), 1)
```

💡 What this does:

- Draws a green dot (`cv2.circle`) on all 21 landmark positions
- Labels each dot with its **index (0–20)** using `cv2.putText`

🔍 What to Observe:

- As you move your hand, the points should **stick to fingers** and move fluidly
- Indexes match the MediaPipe hand model (e.g. 4 = thumb tip, 8 = index tip, etc.)
- You can experiment with labeling just tips or drawing finger connections (let me know if you want that too)

Ready to move on to more gesture detection or want to visualize finger bones next?

- Modify the code to predict the number of fingers and display the same overlaid on the image as text (e.g. if four fingers are raised, display '4' on the screen and if three fingers on one hand and two on the other, the display should be '5').

Perfect! The new instruction is to modify the code so it can **detect how many fingers are raised** and **display the count** on the screen — summing across both hands if needed.

Let's walk through the logic and the code you'll need to insert:

🧠 Finger Detection Logic

Each finger has:

- A **tip landmark** (4, 8, 12, 16, 20)
- A **lower joint** (landmark before the tip)

General Rule:

If the **tip y-coordinate** is **above** (i.e. less than) the joint y-coordinate, the finger is considered **raised**.

Finger	Tip	Joint Below
Thumb	4	2 or 3 *(slightly trickier because it's horizontal)*
Index	8	6
Middle	12	10
Ring	16	14
Pinky	20	18

For simplicity, we can assume the **thumb is up** if:

python Copy Edit

```python
thumb_tip_x > thumb_ip_x  # For right hand
```

Or:

python Copy Edit

```python
thumb_tip_x < thumb_ip_x  # For left hand
```

We'll keep it simple for now: assume **right hand** logic.

✅ Updated Code Snippet to Count Raised Fingers

Add this inside your loop (right after getting `hand_landmarks`):

Copy Edit

python

```python
raised_fingers = 0

# Thumb (we'll use x-coordinates for thumb detection)
if hand_landmarks[4].x > hand_landmarks[3].x:
    raised_fingers += 1

# Other 4 fingers (y-coordinates: tip < pip means finger is up)
if hand_landmarks[8].y < hand_landmarks[6].y:
    raised_fingers += 1
if hand_landmarks[12].y < hand_landmarks[10].y:
    raised_fingers += 1
if hand_landmarks[16].y < hand_landmarks[14].y:
    raised_fingers += 1
if hand_landmarks[20].y < hand_landmarks[18].y:
    raised_fingers += 1
```

Then outside the `for` loop (after checking all hands), sum the total:

python Copy Edit

```python
# Sum across hands
total_fingers += raised_fingers
```

And after the loop over all hands:

python Copy Edit

```python
# Display total on screen
cv2.putText(frame, f'Fingers Raised: {total_fingers}', (10, 50),
            cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 255), 2)
```

🧩 Final Integration Steps

1. Initialize `total_fingers = 0` **before** the hand landmarks loop.
2. Add the above detection code inside your `for idx in range(len(hand_landmarks_list))` loop.
3. Add the `putText` display after the hand loop but **inside the main `while` loop**.

Let me know if you'd like:

- Left vs. right hand logic
- A visual bar/meter of fingers
- Sound alert when a certain number of fingers is reached

Happy to extend it!

can you show me the final code

Absolutely! Here's the **final modified code** that does the following:

✅ Detects hands using MediaPipe
✅ Draws all 21 landmarks
✅ Counts **raised fingers across both hands**
✅ Displays the total number of fingers raised as overlay text

---

✅ Final Code

python

Copy Edit

```python
import cv2
import mediapipe as mp
from mediapipe.tasks import python
from mediapipe.tasks.python import vision

# Parameters
numHands = 2
model = 'hand_landmarker.task'
minHandDetectionConfidence = 0.5
minHandPresenceConfidence = 0.5
minTrackingConfidence = 0.5
frameWidth = 640
frameHeight = 480

MARGIN = 10
FONT_SIZE = 1
FONT_THICKNESS = 2
TEXT_COLOR = (0, 255, 255)
```

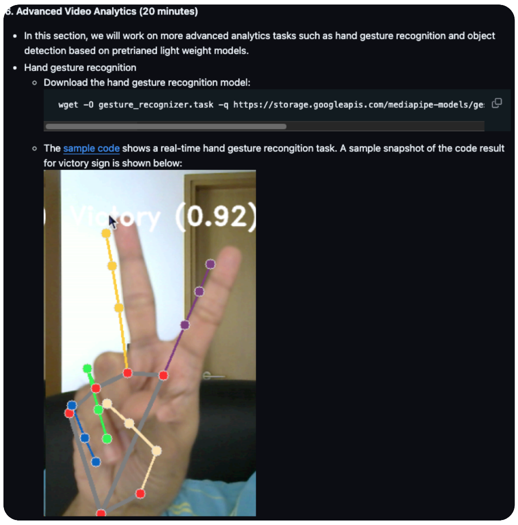
```
# Setup hand landmark detector
base_options = python.BaseOptions(model_asset_path=model)
options = vision.HandLandmarkerOptions(
    base_options=base_options,
    num_hands=numHands,
    min_hand_detection_confidence=minHandDetectionConfidence,
    min_hand_presence_confidence=minHandPresenceConfidence,
    min_tracking_confidence=minTrackingConfidence)
detector = vision.HandLandmarker.create_from_options(options)

# Start webcam
cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, frameWidth)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, frameHeight)

if not cap.isOpened():
    raise IOError("Cannot open webcam")

while True:
    try:
        ret, frame = cap.read()
        frame = cv2.flip(frame, 1)
        rgb_image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

        mp_image = mp.Image(image_format=mp.ImageFormat.SRGB, data=rgb_image)
        detection_result = detector.detect(mp_image)
        hand_landmarks_list = detection_result.hand_landmarks

        total_fingers = 0

        for hand_landmarks in hand_landmarks_list:
            # Show all 21 landmarks
            for i, landmark in enumerate(hand_landmarks):
                x = int(landmark.x * frame.shape[1])
                y = int(landmark.y * frame.shape[0])
                cv2.circle(frame, (x, y), 5, (0, 255, 0), -1)
                cv2.putText(frame, str(i), (x + 5, y - 5),
cv2.FONT_HERSHEY_SIMPLEX, 0.4, (0, 255, 255), 1)

            # Count raised fingers
            raised_fingers = 0

            # Thumb: check horizontal direction (assuming right hand)
            if hand_landmarks[4].x > hand_landmarks[3].x:
                raised_fingers += 1
```

```
            # Other fingers: tip is higher than PIP joint (y is smaller)
            if hand_landmarks[8].y < hand_landmarks[6].y:
                raised_fingers += 1
            if hand_landmarks[12].y < hand_landmarks[10].y:
                raised_fingers += 1
            if hand_landmarks[16].y < hand_landmarks[14].y:
                raised_fingers += 1
            if hand_landmarks[20].y < hand_landmarks[18].y:
                raised_fingers += 1

            total_fingers += raised_fingers

        # Display total raised fingers
        cv2.putText(frame, f'Fingers Raised: {total_fingers}', (10, 40),
                    cv2.FONT_HERSHEY_SIMPLEX, FONT_SIZE, TEXT_COLOR,
FONT_THICKNESS, cv2.LINE_AA)

        cv2.imshow('Finger Counter', frame)

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

    except KeyboardInterrupt:
        break

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

Let me know if you want to:

- Add left/right hand support
- Log finger data to a file
- Trigger actions based on certain gestures

I’m happy to help level it up!



sample code:
#%% Reference:
https://github.com/googlesamples/mediapipe/blob/main/examples/gesture_recognizer/raspberry_pi/
# Download hand gesture detector model wget -O gesture_recognizer.task -q https://storage.googleapis.com/mediapipe-models/gesture_recognizer/gesture_recognizer/float16/1/gesture_recognizer.task
import cv2
import mediapipe as mp
import time

from mediapipe.tasks import python # import the python wrapper
from mediapipe.tasks.python import vision # import the API for calling the recognizer and setting parameters
from mediapipe.framework.formats import landmark_pb2 #The base land mark atlas
mp_hands = mp.solutions.hands
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles

#%% Parameters
numHands = 2 # Number of hands to be detected
model = 'gesture_recognizer.task' # Model for hand gesture detection Download using wget -O gesture_recognizer.task -q https://storage.googleapis.com/mediapipe-models/gesture_recognizer/gesture_recognizer/float16/1/gesture_recognizer.task
minHandDetectionConfidence = 0.5 # Thresholds for detecting the hand

```
minHandPresenceConfidence = 0.5
minTrackingConfidence = 0.5
frameWidth = 640
frameHeight = 480

# Visualization parameters
row_size = 50  # pixels
left_margin = 24  # pixels
text_color = (0, 0, 0)  # black
font_size = 1
font_thickness = 1

# Label box parameters
label_text_color = (255, 255, 255)  # white
label_font_size = 1
label_thickness = 2

#%% Initializing results and save result call back for appending
results.
recognition_frame = None
recognition_result_list = []

def save_result(result: vision.GestureRecognizerResult,
                unused_output_image: mp.Image,timestamp_ms:
int):

    recognition_result_list.append(result)

#%% Create an Hand Gesture Control object.
# Initialize the gesture recognizer model
base_options =
python.BaseOptions(model_asset_path=model)
options =
vision.GestureRecognizerOptions(base_options=base_options,

running_mode=vision.RunningMode.LIVE_STREAM,
                                        num_hands=numHands,

min_hand_detection_confidence=minHandDetectionConfidenc
e,

min_hand_presence_confidence=minHandPresenceConfidenc
e,

min_tracking_confidence=minTrackingConfidence,
                                        result_callback=save_result)
```

```
recognizer =
vision.GestureRecognizer.create_from_options(options)

#%% Open CV Video Capture and frame analysis (setting the
size of the capture resolution as per the model requirements)
cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, frameWidth)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, frameHeight)

# Check if the webcam is opened correctly
if not cap.isOpened():
    raise IOError("Cannot open webcam")

# The loop will break on pressing the 'q' key
while True:
    try:
        # Capture one frame
        ret, frame = cap.read()

        frame = cv2.flip(frame, 1) # To flip the image to match with
camera flip

        # Convert the image from BGR to RGB as required by the
TFLite model.
        rgb_image = cv2.cvtColor(frame,
cv2.COLOR_BGR2RGB)
        mp_image =
mp.Image(image_format=mp.ImageFormat.SRGB,
data=rgb_image)

        current_frame = frame

        # Run hand landmarker using the model.
        recognizer.recognize_async(mp_image, time.time_ns() //
1_000_000)

        if recognition_result_list:

          # Draw landmarks and write the text for each hand.
          for hand_index, hand_landmarks in enumerate(
              recognition_result_list[0].hand_landmarks):

            # Calculate the bounding box of the hand
            x_min = min([landmark.x for landmark in
hand_landmarks])
            y_min = min([landmark.y for landmark in
```

```
hand_landmarks])
        y_max = max([landmark.y for landmark in
hand_landmarks])

        # Convert normalized coordinates to pixel values
        frame_height, frame_width = current_frame.shape[:2]
        x_min_px = int(x_min * frame_width)
        y_min_px = int(y_min * frame_height)
        y_max_px = int(y_max * frame_height)

        # Get gesture classification results
        if recognition_result_list[0].gestures:
          gesture =
recognition_result_list[0].gestures[hand_index]
          category_name = gesture[0].category_name
          score = round(gesture[0].score, 2)
          result_text = f'{category_name} ({score})'

          # Compute text size
          text_size = \
          cv2.getTextSize(result_text,
cv2.FONT_HERSHEY_DUPLEX, label_font_size,
                          label_thickness)[0]
          text_width, text_height = text_size

          # Calculate text position (above the hand)
          text_x = x_min_px
          text_y = y_min_px - 10  # Adjust this value as needed

          # Make sure the text is within the frame boundaries
          if text_y < 0:
            text_y = y_max_px + text_height

          # Draw the text
          cv2.putText(current_frame, result_text, (text_x, text_y),
                      cv2.FONT_HERSHEY_DUPLEX,
label_font_size,
                      label_text_color, label_thickness,
cv2.LINE_AA)

        # Draw hand landmarks on the frame using the atlas
        hand_landmarks_proto =
landmark_pb2.NormalizedLandmarkList()
        hand_landmarks_proto.landmark.extend([
          landmark_pb2.NormalizedLandmark(x=landmark.x,
y=landmark.y,
```

```
                                    z=landmark.z) for landmark in
                        hand_landmarks
                      ])
                      mp_drawing.draw_landmarks(
                        current_frame,
                        hand_landmarks_proto,
                        mp_hands.HAND_CONNECTIONS,

                mp_drawing_styles.get_default_hand_landmarks_style(),

                mp_drawing_styles.get_default_hand_connections_style())

                    recognition_frame = current_frame
                    recognition_result_list.clear()

                  if recognition_frame is not None:
                      cv2.imshow('gesture_recognition', recognition_frame)


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

              except KeyboardInterrupt:
                  break

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

You're now working with **MediaPipe's gesture recognition model**, which is a more advanced version of hand tracking. This lets you identify **specific hand gestures** like *Victory*, *Thumbs Up*, *Open Palm*, etc., with a confidence score.

Let's walk through the final script and what it's doing — and how you can expand on it if needed.

✅ What This Code Does

Functionality	**How it Works**
🖐 Detect hands	Uses `gesture_recognizer.task` model
🔤 Classify gesture	Recognizes gesture (e.g., *Victory*)
💯 Displays confidence	Shows gesture label with confidence score

Functionality	How it Works
🔗 Draws landmarks	Draws 21-point hand skeleton using MediaPipe drawing utils

---

🔍 How Recognition Works

Key Part:

python

```python
gesture = recognition_result_list[0].gestures[hand_index]
category_name = gesture[0].category_name
score = round(gesture[0].score, 2)
result_text = f'{category_name} ({score})'
```

- `category_name` could be: `"Victory"` , `"Thumb_Up"` , `"Open_Palm"` , etc.
- `score` is the confidence (e.g., 0.92 = 92% confident).

Visual Feedback:

python

```python
cv2.putText(current_frame, result_text, (text_x, text_y), ...)
```

- Displays the recognized gesture on the video above the hand.

📦 Pre-trained Gestures Supported

MediaPipe typically supports:

- **Open Palm**
- **Closed Fist**
- **Victory** ✌️
- **Thumb Up**
- **Thumb Down**
- (Others depend on the model version)

You can print detected gestures with:

```python
print(recognition_result_list[0].gestures)
```

🛠️ Ideas to Extend This

Would you like to:

- Trigger custom actions for specific gestures? (e.g., play a sound on *Victory*)
- Display **gesture history** or **gesture counts**?
- Save screenshots when a gesture is detected?
- Switch between **multiple gesture models** or add your own custom ones?

Let me know what direction you'd like to explore — this is the fun part!

can you explain the code to me in detail?

Absolutely! Let's go through the **hand gesture recognition code** you provided, **step by step**, so you fully understand how it works. I'll break it into logical blocks with explanations.

🧠 OVERVIEW

This script:

1. Uses a **MediaPipe Gesture Recognizer** model
2. Captures video from webcam
3. Detects hands and hand **landmarks**
4. Recognizes **predefined gestures** (like "Victory", "Thumb Up", etc.)
5. Displays the recognized gesture label and confidence score on the screen

🧩 SECTION-BY-SECTION BREAKDOWN

📦 Imports and Setup

python

```python
import cv2
import mediapipe as mp
import time
from mediapipe.tasks import python
from mediapipe.tasks.python import vision
from mediapipe.framework.formats import landmark_pb2
```

- `cv2` : OpenCV, used for webcam, drawing, and GUI
- `mediapipe` : Main framework
- `vision` : Where `GestureRecognizer` is implemented
- `landmark_pb2` : Used to format and draw 3D hand landmarks

⚙️ Parameters

python

```python
numHands = 2
model = 'gesture_recognizer.task'
minHandDetectionConfidence = 0.5
minHandPresenceConfidence = 0.5
minTrackingConfidence = 0.5
frameWidth = 640
frameHeight = 480
```

- Loads **gesture recognition model**
- Accepts **up to 2 hands**
- Sets detection/tracking confidence thresholds
- Webcam resolution is 640x480

🎨 Text Drawing Settings

python

```python
text_color = (0, 0, 0)
label_text_color = (255, 255, 255)
label_font_size = 1
label_thickness = 2
```

Defines font size, color, and thickness for overlaying gesture results.

🗃️ Result Handling Function

python

```python
recognition_result_list = []

def save_result(result: vision.GestureRecognizerResult,
                unused_output_image: mp.Image, timestamp_ms: int):
    recognition_result_list.append(result)
```

- This function is **automatically called** when a gesture is recognized.
- It stores the result in a list (`recognition_result_list`) to be drawn later.

🤖 Gesture Recognizer Setup

python

```python
base_options = python.BaseOptions(model_asset_path=model)
options = vision.GestureRecognizerOptions(
    base_options=base_options,
    running_mode=vision.RunningMode.LIVE_STREAM,
    num_hands=numHands,
    ...
    result_callback=save_result
```

```
)
recognizer = vision.GestureRecognizer.create_from_options(options)
```

- Loads the pre-trained gesture recognition model
- Uses **LIVE_STREAM** mode (i.e., real-time frame-by-frame detection)
- Attaches the callback to store results (`save_result`)

📷 Webcam Initialization

python Copy Edit

```python
cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, frameWidth)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, frameHeight)
```

- Opens your default webcam and sets the resolution

🔁 Main Loop

python Copy Edit

```python
while True:
```

The main processing loop that:

1. Captures a frame
2. Converts it to RGB
3. Runs gesture recognition
4. Displays results

🧠 Process Each Frame

python Copy Edit

```python
frame = cv2.flip(frame, 1)
rgb_image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
mp_image = mp.Image(image_format=mp.ImageFormat.SRGB, data=rgb_image)
recognizer.recognize_async(mp_image, time.time_ns() // 1_000_000)
```

- Flips image (mirror view)
- Converts BGR → RGB (MediaPipe expects RGB)
- Converts to MediaPipe format
- Sends the frame to the model asynchronously using a timestamp

---

🖐 Drawing Results

python Copy Edit

```python
if recognition_result_list:
    for hand_index, hand_landmarks in
enumerate(recognition_result_list[0].hand_landmarks):
```

If any gesture results are available:

- Loop through each detected hand
- For each hand:
  - Get landmark coordinates
  - Draw the hand connections and points
  - Display the gesture label and confidence

🏷 Draw Gesture Label

python Copy Edit

```python
gesture = recognition_result_list[0].gestures[hand_index]
category_name = gesture[0].category_name
score = round(gesture[0].score, 2)
result_text = f'{category_name} ({score})'
cv2.putText(current_frame, result_text, (text_x, text_y), ...)
```

- Displays something like:
  `Victory (0.92)` or `Thumb_Up (0.85)`
- Drawn **above the detected hand**

🎨 Draw Landmarks

python

```python
hand_landmarks_proto = landmark_pb2.NormalizedLandmarkList()
hand_landmarks_proto.landmark.extend([...])
mp_drawing.draw_landmarks(...)
```

- Converts detected hand landmarks into MediaPipe's internal format
- Uses `draw_landmarks` to render bones and points with styling

🖼️ Show Final Frame

python

```python
cv2.imshow('gesture_recognition', recognition_frame)
```

- Displays the annotated video frame with everything drawn

🔚 Clean Exit

python

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

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

- Exits cleanly when you press `q`

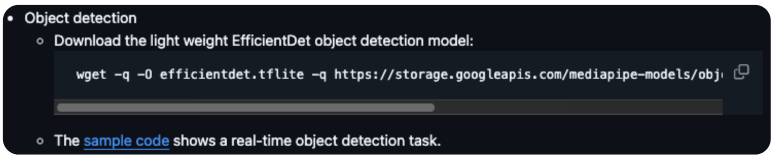
✅ Summary of the Flow

Step	Description
1. Webcam Frame	Captured and flipped
2. Converted to RGB	Prepared for the model
3. Gesture Recognition	Hand landmarks and gesture predicted
4. Draw Results	Landmarks + Gesture label + Confidence
5. Display	Real-time annotated video feed

Would you like help modifying this to:

- Trigger events for specific gestures (e.g., play sound on “Victory”)?
- Count how many times a gesture occurs?
- Use gestures to control GPIO on Raspberry Pi?

Let me know how you’d like to extend it!


- Object detection
  - Download the light weight EfficientDet object detection model:

```
wget -q -O efficientdet.tflite -q https://storage.googleapis.com/mediapipe-models/obj
```

  - The sample code shows a real-time object detection task.

sample code:
#%% Reference:
https://github.com/googlesamples/mediapipe/blob/main/examples/object_detection/raspberry_pi
# Download lightweight ftlite EfficientDet model using wget -q -O efficientdet.tflite -q https://storage.googleapis.com/mediapipe-models/object_detector/efficientdet_lite0/int8/1/efficientdet_lite0.tflite
import cv2
import mediapipe as mp
import time

from mediapipe.tasks import python # import the python wrapper
from mediapipe.tasks.python import vision # import the API for

calling the recognizer and setting parameters

```
#%% Parameters
maxResults = 5
scoreThreshold = 0.25
frameWidth = 640
frameHeight = 480
model = 'efficientdet.tflite'

# Visualization parameters
MARGIN = 10  # pixels
ROW_SIZE = 30  # pixels
FONT_SIZE = 1
FONT_THICKNESS = 1
TEXT_COLOR = (0, 0, 0)  # black


#%% Initializing results and save result call back for appending
results.
detection_frame = None
detection_result_list = []

def save_result(result: vision.ObjectDetectorResult,
unused_output_image: mp.Image, timestamp_ms: int):

    detection_result_list.append(result)

#%% Create an object detection model object.
# Initialize the object detection model
base_options =
python.BaseOptions(model_asset_path=model)
options =
vision.ObjectDetectorOptions(base_options=base_options,

running_mode=vision.RunningMode.LIVE_STREAM,
                                    max_results=maxResults,
score_threshold=scoreThreshold,
                                    result_callback=save_result)
detector = vision.ObjectDetector.create_from_options(options)

#%% Open CV Video Capture and frame analysis (setting the
size of the capture resolution as per the model requirements)
cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, frameWidth)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, frameHeight)
```

```
# Check if the webcam is opened correctly
if not cap.isOpened():
   raise IOError("Cannot open webcam")

# The loop will break on pressing the 'q' key
while True:
   try:
      # Capture one frame
      ret, frame = cap.read()

      frame = cv2.flip(frame, 1) # To flip the image to match with
camera flip

      # Convert the image from BGR to RGB as required by the
TFLite model.
      rgb_image = cv2.cvtColor(frame,
cv2.COLOR_BGR2RGB)
      mp_image =
mp.Image(image_format=mp.ImageFormat.SRGB,
data=rgb_image)

      current_frame = frame

      # Run object detection using the model.
      detector.detect_async(mp_image, time.time_ns() //
1_000_000)

      if detection_result_list:
         for detection in detection_result_list[0].detections:
            # Draw bounding_box
            bbox = detection.bounding_box
            start_point = bbox.origin_x, bbox.origin_y
            end_point = bbox.origin_x + bbox.width,
bbox.origin_y + bbox.height
            # Use the orange color for high visibility.
            cv2.rectangle(current_frame, start_point, end_point,
(0, 165, 255), 3)

            # Draw label and score
            category = detection.categories[0]
            category_name = category.category_name
            probability = round(category.score, 2)
            result_text = category_name + ' (' + str(probability) +
')'
            text_location = (MARGIN + bbox.origin_x,
```

```
                    MARGIN + ROW_SIZE + bbox.origin_y)
            cv2.putText(current_frame, result_text, text_location,
cv2.FONT_HERSHEY_DUPLEX,
                    FONT_SIZE, TEXT_COLOR,
FONT_THICKNESS, cv2.LINE_AA)

        detection_frame = current_frame
        detection_result_list.clear()

      if detection_frame is not None:
        cv2.imshow('object_detection', detection_frame)

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

    except KeyboardInterrupt:
      break

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

You're now in the **object detection** phase using **MediaPipe's EfficientDet model**. This is a lightweight deep learning model optimized for **real-time detection** on resource-constrained devices like the Raspberry Pi. Let's break down the code to understand what it's doing:

🧠 Objective

- Use **EfficientDet Lite0** model (TFLite format) to detect common objects.
- Draw **bounding boxes** and **labels** with **confidence scores**.
- Display it all live using your webcam.

🧩 SECTION-BY-SECTION EXPLANATION

📦 Imports

python Copy Edit

```python
import cv2
import mediapipe as mp
import time
from mediapipe.tasks import python
from mediapipe.tasks.python import vision
```

- `cv2` : OpenCV, used for webcam and drawing
- `mediapipe` : Google's ML vision framework
- `vision` : The part of MediaPipe that deals with object detection
- `time` : Used to timestamp frames for real-time detection

⚙️ Parameters

python Copy Edit

```python
maxResults = 5
scoreThreshold = 0.25
model = 'efficientdet.tflite'
```

- `maxResults` : Show up to 5 detected objects
- `scoreThreshold` : Only show predictions with >25% confidence
- `model` : The path to the pre-trained `.tflite` object detection model

---

🎨 Visualization Settings

python Copy Edit

```python
MARGIN = 10
ROW_SIZE = 30
FONT_SIZE = 1
TEXT_COLOR = (0, 0, 0)
```

- These control the appearance of the label text (e.g. "Person (0.89)")

📥 Store Detection Results

python

Copy Edit

```python
detection_result_list = []

def save_result(result, image, timestamp):
    detection_result_list.append(result)
```

- `save_result()` is a callback that stores detection results in a list.
- It's called automatically when `detector.detect_async()` completes.

🤖 Model Initialization

python

Copy Edit

```python
base_options = python.BaseOptions(model_asset_path=model)
options = vision.ObjectDetectorOptions(
    base_options=base_options,
    running_mode=vision.RunningMode.LIVE_STREAM,
    max_results=maxResults,
    score_threshold=scoreThreshold,
    result_callback=save_result)
detector = vision.ObjectDetector.create_from_options(options)
```

- Sets up the EfficientDet model with the parameters you defined.
- `LIVE_STREAM` mode = expects continuous video.
- Hooks in the `save_result()` function to handle detection outputs.

📷 Webcam Setup

python

Copy Edit

```python
cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
```

```
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
```

- Starts video capture from the Pi webcam or USB webcam.
- Sets the resolution to 640×480.

🔁 Main Loop

python

```python
while True:
    ...
```

Runs indefinitely until you press `q` or Ctrl+C.

🎥 Capture and Process Frame

python

```python
frame = cv2.flip(frame, 1)
rgb_image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
mp_image = mp.Image(image_format=mp.ImageFormat.SRGB, data=rgb_image)
```

- Flips the webcam feed horizontally.
- Converts the image to **RGB**, as required by the model.

🔍 Run Object Detection

python

```python
detector.detect_async(mp_image, time.time_ns() // 1_000_000)
```

- Asynchronously runs object detection and timestamps the frame.

📦 Draw Detection Results

python

```python
if detection_result_list:
    for detection in detection_result_list[0].detections:
        ...
```

For each detection:

- Extracts the bounding box:

  python

  ```python
  bbox = detection.bounding_box
  start_point = bbox.origin_x, bbox.origin_y
  end_point = bbox.origin_x + bbox.width, bbox.origin_y + bbox.height
  ```

- Draws the box using OpenCV:

  python

  ```python
  cv2.rectangle(...)
  ```

- Draws the label and confidence:

  python

  ```python
  result_text = 'person (0.89)'  # Example
  cv2.putText(...)
  ```

🖼️ Show Frame

python

```python
cv2.imshow('object_detection', detection_frame)
```

- Displays the annotated frame in a window titled `object_detection`

❌ Exit Safely

python Copy Edit

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

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

- Breaks the loop when `q` is pressed
- Releases the webcam and closes OpenCV windows

✅ Summary Table

Feature	Description
Model Used	`efficientdet_lite0.tflite`
Library	MediaPipe + OpenCV
Output	Bounding boxes + object labels + confidence
Display	Real-time on screen
Triggers	Detection happens continuously via `detect_async()`

Would you like to:

- Save detected object info to a log file?
- Count how often a specific object appears (e.g. "person")?
- Add color coding for different object types?

Let me know how you'd like to customize it!

Based on the above code, write a code to do object detection based video summarization (e.g. for a video with only frames having a cellphone)

Perfect — this step takes it to the next level! You're now being asked to do **object detection–based video summarization** using EfficientDet.

🧠 Objective

> **Extract only the frames from a video (or live webcam) where a specific object (e.g., a "cell phone") is detected, and save those frames as a summary video.**

✅ Key Tasks

1. Load EfficientDet object detector
2. Open webcam or video file
3. For each frame:
   - Detect objects
   - If "cell phone" is found → **save the frame**
4. Write saved frames to a new summary video

🧾 Final Code: Save Frames with "cell phone" Detected

python Copy Edit

```python
import cv2
import mediapipe as mp
import time
from mediapipe.tasks import python
from mediapipe.tasks.python import vision

# Parameters
model = 'efficientdet.tflite'
target_label = 'cell phone'  # You can change this to 'book', 'person', etc.
scoreThreshold = 0.4
frameWidth = 640
frameHeight = 480

# Store matched frames
matched_frames = []

# Setup model
```

```python
base_options = python.BaseOptions(model_asset_path=model)
options = vision.ObjectDetectorOptions(
    base_options=base_options,
    running_mode=vision.RunningMode.IMAGE,
    max_results=5,
    score_threshold=scoreThreshold
)
detector = vision.ObjectDetector.create_from_options(options)

# Open webcam or video
cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, frameWidth)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, frameHeight)

if not cap.isOpened():
    raise IOError("Cannot open webcam")

print("Looking for object:", target_label)

while True:
    ret, frame = cap.read()
    if not ret:
        break

    frame = cv2.flip(frame, 1)
    rgb_image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    mp_image = mp.Image(image_format=mp.ImageFormat.SRGB, data=rgb_image)

    # Run detection
    detection_result = detector.detect(mp_image)

    save_this_frame = False

    for detection in detection_result.detections:
        category = detection.categories[0]
        if category.category_name.lower() == target_label.lower():
            save_this_frame = True
            break

    if save_this_frame:
        matched_frames.append(frame.copy())

    cv2.imshow('Live Feed', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

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

# Save summary video
if matched_frames:
    print(f"Saving summary video with {len(matched_frames)} frames...")
    out = cv2.VideoWriter('summary_video.avi',
                          cv2.VideoWriter_fourcc(*'XVID'),
                          10,  # fps
                          (frameWidth, frameHeight))
    for f in matched_frames:
        out.write(f)
    out.release()
    print("summary_video.avi saved.")
else:
    print("No frames with the specified object were found.")
```

📦 What It Does

- **Live detection** from webcam
- **Checks if "cell phone"** appears in the detections
- **Saves only those frames**
- Exports a **summary video** named `summary_video.avi`

🔄 You Can Customize

To Do	Change
Detect from a video file	`cv2.VideoCapture("input.mp4")`
Detect multiple labels	Check for multiple `category_name` s
Save images instead of video	Use `cv2.imwrite()` for each frame

Would you like a version that works **on a saved video file** instead of live webcam? Or should we display the bounding boxes only on the filtered summary frames?

[Optional] Homework/Extended Activities:

1. Experiment with more advanced tracking algorithms available in OpenCV.

Great! This optional section encourages you to **experiment with more advanced tracking algorithms in OpenCV** — and that's a fantastic way to build on what you've learned in this lab.

🎯 What is Object Tracking?

While **object detection** identifies where an object is in each frame (independently), **tracking** allows you to:

- **Track an object across frames** after it's been identified once
- Avoid expensive re-detection every frame
- Follow motion paths (e.g., for analysis or robotics)

🛠️ OpenCV Tracking Algorithms You Can Try

OpenCV provides multiple built-in tracking algorithms via `cv2.Tracker_create()` :

Tracker	Description
`KCF`	Kernelized Correlation Filters – good accuracy/speed
`CSRT`	More accurate but slower than KCF
`MIL`	Robust to occlusion, but less accurate
`TLD`	Learns during tracking – experimental
`MOSSE`	Very fast, less accurate
`MedianFlow`	Good for predictable motion, fails on occlusion
`BOOSTING`	Based on AdaBoost – older

🧪 Starter Code: Use OpenCV Tracker to Track an Object

```python
import cv2

# Choose tracker type: "KCF", "CSRT", etc.
tracker = cv2.TrackerCSRT_create()

cap = cv2.VideoCapture(0)
ret, frame = cap.read()

# Let user draw the bounding box
bbox = cv2.selectROI("Tracking", frame, False)
tracker.init(frame, bbox)

while True:
    ret, frame = cap.read()
    if not ret:
        break

    success, box = tracker.update(frame)

    if success:
        x, y, w, h = [int(v) for v in box]
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
        cv2.putText(frame, "Tracking", (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)
    else:
        cv2.putText(frame, "Lost", (20, 80), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0, 0, 255), 2)

    cv2.imshow("Tracking", frame)

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

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

🧪 Tips for Exploring

- Try switching the tracker type and compare accuracy and speed
- Test in low light, with occlusion, or fast motion

- Try integrating it with **object detection**: detect once, track after

---

🧠 Further Ideas

Want to take it further? Try:

- Combining EfficientDet + CSRT tracker: detect once, then track
- Counting how long a tracked object stays in view
- Plotting motion trails

Let me know if you'd like to build any of these and I'll help you scaffold it!

can you provide me with the code for another tracker type? CSRT is way too slow

Absolutely! If **CSRT is too slow**, try the **KCF (Kernelized Correlation Filter)** tracker — it's a great balance between **speed and accuracy**, and it works well in real-time on the Raspberry Pi.

✅ Here's the Code Using `KCF` Tracker

python Copy Edit

```python
import cv2

# Create a KCF tracker
tracker = cv2.TrackerKCF_create()

# Open webcam
cap = cv2.VideoCapture(0)
ret, frame = cap.read()

# Select ROI (draw box with mouse)
bbox = cv2.selectROI("KCF Tracker", frame, False)
tracker.init(frame, bbox)

while True:
```

```
    ret, frame = cap.read()
    if not ret:
        break

    # Update tracker
    success, box = tracker.update(frame)

    if success:
        x, y, w, h = [int(v) for v in box]
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
        cv2.putText(frame, "Tracking", (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)
    else:
        cv2.putText(frame, "Lost", (20, 80), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0, 0, 255), 2)

    cv2.imshow("KCF Tracker", frame)

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

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

📌 Notes

- You'll be prompted to **drag a bounding box** around the object you want to track (e.g., your hand, a face, or a bottle).
- Press **Enter** to confirm the selection.
- The tracker will then follow that object in real time.

🧪 Want Something Faster?

If even KCF is a bit slow and you don't need high precision:

- Try `MOSSE` tracker (very fast but less accurate):

python Copy Edit

```
tracker = cv2.legacy.TrackerMOSSE_create()
```

Let me know if you'd like to integrate this with **object detection → tracker handoff**, or want help comparing tracker FPS/performance!

2. Build a gesture based video player control (e.g. could use libraries like Pyautogui for the same)

this is my 2nd optional homework

Nice! Your second optional homework is super cool — you're asked to:

> 🎯 **Build a gesture-based video player control**
> (e.g., play/pause/skip using hand gestures, potentially using `pyautogui` to simulate keyboard shortcuts)

🧠 Idea: Control VLC (or any player) using gestures

Gesture	Action	Keyboard Shortcut
✋ Open palm	**Pause / Play**	Spacebar
👍 Thumbs up	**Volume Up**	`Up Arrow`
👎 Thumbs down	**Volume Down**	`Down Arrow`
👉 Victory sign	**Next / Forward**	`Right Arrow`
👈 Fist	**Previous / Back**	`Left Arrow`

We'll use:

- MediaPipe **gesture recognizer**
- `pyautogui` to simulate key presses

✅ Code: Gesture-based Player Control (VLC example)

> Make sure:

- “You install `pyautogui` :
  `pip install pyautogui` ”

python

Copy Edit

```python
import cv2
import pyautogui
import mediapipe as mp
import time
from mediapipe.tasks import python
from mediapipe.tasks.python import vision

# Setup
model = 'gesture_recognizer.task'
min_conf = 0.7  # Confidence threshold

# Load model
base_options = python.BaseOptions(model_asset_path=model)
options = vision.GestureRecognizerOptions(
    base_options=base_options,
    running_mode=vision.RunningMode.LIVE_STREAM,
    result_callback=lambda result, image, ts: results.append(result)
)
recognizer = vision.GestureRecognizer.create_from_options(options)

# Webcam
cap = cv2.VideoCapture(0)
results = []
last_gesture = ""
last_time = time.time()

while True:
    ret, frame = cap.read()
    if not ret:
        break

    frame = cv2.flip(frame, 1)
    rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    mp_img = mp.Image(image_format=mp.ImageFormat.SRGB, data=rgb)
    recognizer.recognize_async(mp_img, time.time_ns() // 1_000_000)

    # If gesture result is ready
    if results:
        gesture_list = results.pop().gestures
        if gesture_list:
            gesture = gesture_list[0][0]  # Most confident gesture
```

```
            name = gesture.category_name
            score = gesture.score

            # Debounce actions (1 sec cooldown)
            if name != last_gesture and score > min_conf and (time.time() -
last_time) > 1:
                print(f"Gesture Detected: {name} ({score:.2f})")
                last_time = time.time()
                last_gesture = name

                # Gesture-action mapping
                if name == "Open_Palm":
                    pyautogui.press("space")  # play/pause
                elif name == "Thumb_Up":
                    pyautogui.press("up")     # volume up
                elif name == "Thumb_Down":
                    pyautogui.press("down")   # volume down
                elif name == "Victory":
                    pyautogui.press("right")  # next/forward
                elif name == "Closed_Fist":
                    pyautogui.press("left")   # previous/back

    cv2.imshow("Gesture Control", frame)
    if cv2.waitKey(1) & 0xFF == ord("q"):
        break

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

🎮 How to Use This

- Open VLC or YouTube (or any app that responds to arrow keys and spacebar)
- Run the script
- Perform gestures in front of the camera — actions will be triggered!

🧪 Bonus Ideas

- Add GUI feedback on screen (“Paused”, “Volume Up”)

- Track gesture hold duration (e.g., hold thumbs up for volume ramp-up)
- Control custom video player (e.g., `cv2.VideoCapture` + OpenCV display)

Let me know if you want help making this work with your own app or want to build a full media controller!

3. Build a surveillance system based on video based motion detection.

this is my 3rd optional homework

Awesome! Your 3rd optional homework is to:

> 🎯 **Build a surveillance system based on video-based motion detection**

This means detecting when **something moves in front of the camera**, and triggering an action like:

- Logging the event
- Saving the frame or clip
- Raising an alert

Let's build a lightweight, real-time **motion detection system using OpenCV**.

✅ Basic Surveillance System Using Motion Detection

python

Copy Edit

```python
import cv2
import time

# Initialize camera
cap = cv2.VideoCapture(0)
ret, frame1 = cap.read()
ret, frame2 = cap.read()

motion_detected = False
save_count = 0

while cap.isOpened():
    # Compute the absolute difference between current and previous frame
```

```
    diff = cv2.absdiff(frame1, frame2)

    # Convert to grayscale + blur to reduce noise
    gray = cv2.cvtColor(diff, cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(gray, (5, 5), 0)

    # Apply threshold and find contours
    _, thresh = cv2.threshold(blur, 25, 255, cv2.THRESH_BINARY)
    dilated = cv2.dilate(thresh, None, iterations=2)
    contours, _ = cv2.findContours(dilated, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)

    motion_detected = False
    for contour in contours:
        if cv2.contourArea(contour) < 1000:
            continue
        motion_detected = True
        (x, y, w, h) = cv2.boundingRect(contour)
        cv2.rectangle(frame1, (x, y), (x + w, y + h), (0, 255, 0), 2)

    # Display motion status
    text = "Motion Detected" if motion_detected else "No Motion"
    cv2.putText(frame1, text, (10, 30), cv2.FONT_HERSHEY_SIMPLEX,
                1, (0, 0, 255) if motion_detected else (200, 200, 200), 2)

    cv2.imshow("Surveillance Feed", frame1)

    # Save frame when motion is detected
    if motion_detected:
        timestamp = time.strftime("%Y%m%d-%H%M%S")
        filename = f"motion_{timestamp}.jpg"
        cv2.imwrite(filename, frame1)
        print(f"[INFO] Motion captured → {filename}")
        time.sleep(1)  # delay to prevent spamming

    # Update frames
    frame1 = frame2
    ret, frame2 = cap.read()

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

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

🧠 How It Works

Step	Description
`absdiff()`	Finds difference between two consecutive frames
Threshold + blur	Isolates areas that have changed
Contours	Extracts moving objects
Frame capture	Saves frame if movement exceeds threshold