Hand Gesture Calculator Project Report

1. Project Overview

This project implements a Hand Gesture Calculator using a webcam and MediaPipe hand tracking. The user can perform basic arithmetic operations by showing finger gestures corresponding to digits and operators. The expression is displayed on the screen, and the calculation result is shown after making the equals gesture.

2. Tools and Libraries Used

- Python 3
- OpenCV for webcam and image processing
- MediaPipe for real-time hand detection and finger tracking
- FPDF for PDF report generation (used here)

Install libraries using:

`pip install opencv-python mediapipe fpdf`

3. Step-by-Step Code Explanation

The program captures video from the webcam and processes each frame to detect a single hand using MediaPipe. It then identifies which fingers are up by comparing landmark positions. Based on the finger pattern, a gesture is recognized which maps to digits, operators, equals sign, or clear command.

The recognized gestures are appended to build an arithmetic expression displayed on screen. When the equals gesture (fist) is detected, the expression is evaluated using Python's eval function, and the result is displayed. The clear gesture resets the expression and result.

A delay mechanism prevents multiple detections of the same gesture in quick succession.

4. Complete Code

```
import cv2
import mediapipe as mp

mp_hands = mp.solutions.hands
hands = mp_hands.Hands(max_num_hands=1)
mp_draw = mp.solutions.drawing_utils

cap = cv2.VideoCapture(0)

gesture_map = {
    (0,1,0,0,0): "1",
    (0,1,1,0,0): "2",
    (0,1,1,1,0): "3",
    (0,1,1,1,1): "4",
    (1,1,1,1,1): "5",
    (1,1,0,0,0): "+",
    (1,0,0,0,0): "-",
    (1,1,1,0,0): "*",
    (0,0,0,0,0): "=",
    (1,0,0,0,1): "C"
}

equation = ""
result = ""
last_gesture = None
delay_counter = 0
delay_threshold = 20

def fingers_up(hand_landmarks):
    tips = [4, 8, 12, 16, 20]
    fingers = []
    if hand_landmarks.landmark[tips[0]].x < hand_landmarks.landmark[tips[0]-1].x:
        fingers.append(1)
    else:
        fingers.append(0)
    for tip in tips[1:]:
        if hand_landmarks.landmark[tip].y < hand_landmarks.landmark[tip - 2].y:
            fingers.append(1)
        else:
            fingers.append(0)
    return tuple(fingers)

while True:
    success, img = cap.read()
    if not success:
        break

    img = cv2.flip(img, 1)
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    result_hands = hands.process(img_rgb)

    if result_hands.multi_hand_landmarks:
        hand_landmarks = result_hands.multi_hand_landmarks[0]
        mp_draw.draw_landmarks(img, hand_landmarks, mp_hands.HAND_CONNECTIONS)

        finger_tuple = fingers_up(hand_landmarks)
        cv2.putText(img, f"Fingers: {finger_tuple}", (10, 50),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255,255,255), 2)

        if delay_counter == 0:
            if finger_tuple in gesture_map:
                gesture = gesture_map[finger_tuple]
                if gesture != last_gesture:
                    if gesture == "C":
                        equation = ""
                        result = ""
                    elif gesture == "=":
                        try:
                            result = str(eval(equation))
                        except:
                            result = "Error"
                        equation = ""
                    else:
                        equation += gesture
                    last_gesture = gesture
                    delay_counter = delay_threshold
        else:
            delay_counter -= 1
    else:
        last_gesture = None

    cv2.putText(img, f"Expr: {equation}", (10, 100),
                cv2.FONT_HERSHEY_SIMPLEX, 1.5, (255, 255, 255), 3)

    if result != "":
        cv2.putText(img, f"Result: {result}", (10, 170),
                    cv2.FONT_HERSHEY_SIMPLEX, 1.5, (0, 0, 255), 3)

    cv2.imshow("Hand Gesture Calculator", img)

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

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

5. How to Run

1. Make sure you have Python 3 installed.

2. Install required libraries:

   pip install opencv-python mediapipe

3. Save the code as hand_gesture_calculator.py

4. Run the script:

   python hand_gesture_calculator.py

5. Use finger gestures as mapped to perform calculations.

6. Press 'q' to exit.

6. Conclusion

This project demonstrates the integration of computer vision and real-time hand gesture recognition to create a natural user interface for a calculator. It can be extended with more gestures and features for enhanced usability.