Student Guide: Hand Tracking with LED and DC Motor Control

Computer Vision and Hardware Integration with Raspberry Pi

Table of Contents

Introduction & Setup

What You'll Learn

By completing these three projects, you'll master: - Computer vision with OpenCV and MediaPipe - GPIO control on Raspberry Pi - Hardware interfacing with LEDs and motors - Real-time gesture recognition systems

Required Software Installation

```
# Install all required Python packages
pip install cvzone
pip install opencv-python
pip install mediapipe
pip install gpiozero
```

Safety First!

- **LED Modules**: Use 3-pin LED modules with built-in resistors - they're much safer!
- **Never exceed voltage ratings**: 3.3V or 5V for modules
- **Disconnect power** when changing connections
- **Keep workspace organized** and clean

Project 1: Single LED Control

Learn the Basics of Hand Tracking and GPIO

Goal: Turn an LED ON/OFF with a 3-finger gesture

Understanding 3-Pin LED Modules

3-pin LED modules are your best friend for learning! Here's why: - **Built-in resistor** - no math needed, no LED burnout - **Just 3 connections** - VCC (power), GND (ground), Signal (control) - **Beginner-friendly** - much harder to wire incorrectly

```
   LED PCB
               ← LED

 S  VCC  GND    ← Pins



           Ground (Black/Brown wire)
          Power +3.3V or +5V (Red wire)
         Signal/Control (Yellow/Orange wire)
```

Hardware Setup

Step 1: Connect Your LED Module

```
LED Module Pin    →    Raspberry Pi Pin

VCC (Red wire)    →    3.3V (Pin 1) or 5V (Pin 2)
GND (Black wire)  →    GND (Pin 6)
Signal (Yellow)   →    GPIO 17 (Pin 11)
```

Step 2: Test Your Connection Before writing the hand tracking code, test your LED with this simple script:

```python
from gpiozero import LED
from time import sleep

# LED module connected to GPIO 17
led = LED(17)

print("Testing LED module...")
for _ in range(3):
    led.on()
    print("LED ON")
    sleep(1)
    led.off()
    print("LED OFF")
    sleep(1)
print("Test complete!")
```

Complete Hand Tracking LED Code

```
from cvzone.HandTrackingModule import HandDetector
import cv2
from gpiozero import LED
import cvzone

# Initialize LED on GPIO pin 17
led = LED(17)

# Initialize the webcam
cap = cv2.VideoCapture(0)

# Initialize the HandDetector
detector = HandDetector(staticMode=False, maxHands=2, modelComplexity=1,
                       detectionCon=0.5, minTrackCon=0.5)

# LED status flag for display
led_status = False

try:
    while True:
        # Capture frame from webcam
        success, img = cap.read()

        if not success:
            print("Failed to read from camera")
            break

        # Find hands in the current frame
        hands, img = detector.findHands(img, draw=True, flipType=True)

        # Check if any hands are detected
        if hands:
            # Get first hand information
            hand1 = hands[0]

            # Count fingers
            fingers1 = detector.fingersUp(hand1)
            finger_count = fingers1.count(1)

            # Control LED based on finger count
            if finger_count == 3:
                led.on()
                led_status = True
                led_color = (0, 255, 0)  # Green when LED is ON
            else:
                led.off()
                led_status = False
                led_color = (0, 0, 255)  # Red when LED is OFF

            # Display finger count on screen
            cvzone.putTextRect(img, f"Fingers: {finger_count}", (50, 50),
                             scale=2, thickness=2,
                             colorT=(255, 255, 255), colorR=(255, 0, 255))

            # Display LED status
            status_text = "LED: ON" if led_status else "LED: OFF"
            cvzone.putTextRect(img, status_text, (50, 100),
                             scale=2, thickness=2,
                             colorT=(255, 255, 255), colorR=led_color)

            # Visual indicator circle
            cv2.circle(img, (300, 75), 20, led_color, cv2.FILLED)

            # Print to console
            print(f"Fingers: {finger_count} | LED: {'ON' if led_status else 'OFF'}")

        else:
            # No hands detected - turn off LED
            led.off()
            led_status = False

            # Display status when no hands detected
            cvzone.putTextRect(img, "No hands detected", (50, 50),
                             scale=2, thickness=2,
                             colorT=(255, 255, 255), colorR=(128, 128, 128))
            cvzone.putTextRect(img, "LED: OFF", (50, 100),
                             scale=2, thickness=2,
                             colorT=(255, 255, 255), colorR=(0, 0, 255))

        # Display the image
        cv2.imshow("Hand Tracking - LED Control", img)

        # Exit on 'q' key press
        if cv2.waitKey(1) & 0xFF == ord('q'):
            print("Exiting...")
            break
```

```
except KeyboardInterrupt:
    print("\nProgram interrupted by user")

finally:
    # Clean up
    led.off()
    led.close()
    cap.release()
    cv2.destroyAllWindows()
    print("Cleanup complete - LED turned off and resources released")
```

Project 1 Challenges

Easy Modifications: 1. **Change trigger**: Make LED turn on with 2 fingers instead of 3 2. **Reverse logic**: LED on by default, off when showing 3 fingers 3. **Add blink**: Make LED blink when showing 5 fingers

Intermediate Challenges: 1. **Timer**: LED stays on for 5 seconds after gesture 2. **Multiple triggers**: Different finger counts do different things 3. **Brightness control**: Use PWM to dim LED based on finger count

---

Project 2: Multiple LED Control

Expand Your GPIO Skills with Complex Logic

Goal: Control 3 different LEDs with 1, 2, and 3 finger gestures

Hardware Setup

Connect Three LED Modules:

```
LED Module 1 (Red):
VCC → 3.3V (Pin 1)    GND → GND (Pin 6)     Signal → GPIO 17 (Pin 11)

LED Module 2 (Yellow):
VCC → 3.3V (Pin 17)   GND → GND (Pin 9)     Signal → GPIO 27 (Pin 13)

LED Module 3 (Green):
VCC → 3.3V (Pin 1)    GND → GND (Pin 14)    Signal → GPIO 22 (Pin 15)
```

Pro Tip: You can share VCC and GND connections! Connect multiple red wires to the same 3.3V pin, and multiple black wires to the same GND pin.

Test Each LED Independently

Always test before the main code:

```
from gpiozero import LED
from time import sleep

leds = [LED(17), LED(27), LED(22)]

for i, led in enumerate(leds):
    print(f"Testing LED {i+1}")
    led.on()
    sleep(1)
    led.off()
```

Complete Multiple LED Code

```
from cvzone.HandTrackingModule import HandDetector
import cv2
from gpiozero import LED
import cvzone

# Initialize LEDs on different GPIO pins
led1 = LED(17)  # LED for 1 finger - GPIO 17
led2 = LED(27)  # LED for 2 fingers - GPIO 27
led3 = LED(22)  # LED for 3 fingers - GPIO 22

# Initialize the webcam
cap = cv2.VideoCapture(0)

# Initialize the HandDetector
detector = HandDetector(staticMode=False, maxHands=2, modelComplexity=1,
                       detectionCon=0.5, minTrackCon=0.5)

# LED status dictionary
led_status = {1: False, 2: False, 3: False}

def control_leds(finger_count):
    """Control LEDs based on finger count"""
    # Turn off all LEDs first
    led1.off()
    led2.off()
    led3.off()
    led_status[1] = False
    led_status[2] = False
    led_status[3] = False

    # Turn on appropriate LED based on finger count
    if finger_count == 1:
        led1.on()
```

```
            led_status[1] = True
        elif finger_count == 2:
            led2.on()
            led_status[2] = True
        elif finger_count == 3:
            led3.on()
            led_status[3] = True

        return led_status

    def draw_led_indicators(img, led_status, finger_count):
        """Draw LED status indicators on the image"""
        # Base positions for LED indicators
        x_start = 50
        y_start = 50
        spacing = 120

        # LED 1 Indicator
        color1 = (0, 255, 0) if led_status[1] else (100, 100, 100)
        cvzone.putTextRect(img, "LED1", (x_start, y_start),
                           scale=1.5, thickness=2,
                           colorT=(255, 255, 255), colorR=color1)
        cv2.circle(img, (x_start + 70, y_start - 10), 15, color1, cv2.FILLED)

        # LED 2 Indicator
        color2 = (0, 255, 0) if led_status[2] else (100, 100, 100)
        cvzone.putTextRect(img, "LED2", (x_start + spacing, y_start),
                           scale=1.5, thickness=2,
                           colorT=(255, 255, 255), colorR=color2)
        cv2.circle(img, (x_start + spacing + 70, y_start - 10), 15, color2, cv2.FILLED)

        # LED 3 Indicator
        color3 = (0, 255, 0) if led_status[3] else (100, 100, 100)
        cvzone.putTextRect(img, "LED3", (x_start + spacing*2, y_start),
                           scale=1.5, thickness=2,
                           colorT=(255, 255, 255), colorR=color3)
        cv2.circle(img, (x_start + spacing*2 + 70, y_start - 10), 15, color3, cv2.FILLED)

        # Display finger count
        cvzone.putTextRect(img, f"Fingers: {finger_count}", (50, 120),
                           scale=2, thickness=3,
                           colorT=(255, 255, 255), colorR=(255, 0, 255))

        # Display instructions
        cvzone.putTextRect(img, "1 finger = LED1 | 2 fingers = LED2 | 3 fingers = LED3",
                           (50, 450), scale=1, thickness=1,
                           colorT=(255, 255, 255), colorR=(0, 0, 0))

    try:
        while True:
            # Capture frame from webcam
            success, img = cap.read()

            if not success:
                print("Failed to read from camera")
                break

            # Find hands in the current frame
            hands, img = detector.findHands(img, draw=True, flipType=True)

            # Check if any hands are detected
            if hands:
                # Get first hand information
                hand1 = hands[0]

                # Count fingers
                fingers1 = detector.fingersUp(hand1)
                finger_count = fingers1.count(1)

                # Control LEDs based on finger count
                led_status = control_leds(finger_count)

                # Draw LED indicators
                draw_led_indicators(img, led_status, finger_count)

                # Print to console
                status_str = f"Fingers: {finger_count} | "
                status_str += f"LED1: {'ON' if led_status[1] else 'OFF'} | "
                status_str += f"LED2: {'ON' if led_status[2] else 'OFF'} | "
                status_str += f"LED3: {'ON' if led_status[3] else 'OFF'}"
                print(status_str)

                # Check if a second hand is detected
                if len(hands) == 2:
                    hand2 = hands[1]
                    fingers2 = detector.fingersUp(hand2)
                    hand2_count = fingers2.count(1)

                    # Display second hand info
                    cvzone.putTextRect(img, f"Hand 2: {hand2_count} fingers", (50, 160),
                                     scale=1.5, thickness=2,
                                     colorT=(255, 255, 255), colorR=(128, 128, 255))
            else:
                # No hands detected - turn off all LEDs
                led1.off()
                led2.off()
                led3.off()
                led_status = {1: False, 2: False, 3: False}

                # Display status when no hands detected
                draw_led_indicators(img, led_status, 0)
                cvzone.putTextRect(img, "No hands detected - All LEDs OFF", (50, 200),
                                 scale=1.5, thickness=2,
                                 colorT=(255, 255, 255), colorR=(128, 128, 128))

            # Display the image
            cv2.imshow("Hand Tracking - Multiple LED Control", img)

            # Exit on 'q' key press
            if cv2.waitKey(1) & 0xFF == ord('q'):
                print("\nExiting...")
                break

    except KeyboardInterrupt:
        print("\nProgram interrupted by user")

    finally:
        # Clean up - turn off all LEDs
        led1.off()
        led2.off()
        led3.off()
        led1.close()
        led2.close()
        led3.close()
        cap.release()
        cv2.destroyAllWindows()
        print("Cleanup complete - All LEDs turned off and resources released")
```

Project 2 Challenges

Pattern Mode Challenges: 1. **Cumulative**: 1 finger = LED1, 2 fingers = LED1+LED2, 3 fingers = All LEDs 2. **Traffic Light**: 1=Red, 2=Yellow, 3=Green with proper transitions 3. **Sequential**: LEDs light up in sequence, then turn off in sequence

Advanced Challenges: 1. **Fade Effects**: Use PWM to fade LEDs in/out 2. **Timer System**: LEDs auto-turn off after 5 seconds 3. **Binary Display**: Show finger count in binary using LEDs

Project 3: DC Motor Control

Advanced Hardware with PWM Speed Control

Goal: Control motor speed with hand gestures using an L298N motor driver

Understanding the L298N Motor Driver

The L298N is like a "translator" between your Raspberry Pi and motor: - **PWM Speed Control**: Variable speed using duty cycle - **Direction Control**: Forward/backward/stop/brake - **Current Protection**: Handles high motor currents safely - **External Power**: Uses separate power supply for motor

Hardware Setup

Safety First with Motors: - Secure motor before testing - Start with low speeds - Keep fingers away from moving parts - Have emergency stop ready

L298N Connections:

```
Raspberry Pi          L298N Driver Board

GPIO 18 (Pin 12) --> ENA (Enable A - PWM speed control)
GPIO 23 (Pin 16) --> IN1 (Motor A Input 1)
GPIO 24 (Pin 18) --> IN2 (Motor A Input 2)
GND (Pin 6)      --> GND (Common ground)

L298N Power Connections:

+12V --> External power supply positive (6-12V)
GND  --> External power supply negative
5V   --> Can power Raspberry Pi if needed (optional)

Motor Connections:

OUT1 --> Motor positive terminal
OUT2 --> Motor negative terminal
```

CRITICAL: Remove the ENA jumper on the L298N board to enable PWM speed control!

Test Motor Without Hand Tracking

Always test your motor setup first:

```python
from gpiozero import Motor, PWMOutputDevice
from time import sleep
```

```
pwm = PWMOutputDevice(18)
motor = Motor(23, 24)

# Test different speeds
for speed in [0.3, 0.5, 0.7, 1.0]:
    print(f"Testing {int(speed*100)}% speed")
    pwm.value = speed
    motor.forward()
    sleep(2)

motor.stop()
pwm.close()
motor.close()
```

Complete Motor Control Code

```
from cvzone.HandTrackingModule import HandDetector
import cv2
from gpiozero import Motor, PWMOutputDevice
import cvzone
from time import sleep

# L298N Motor Driver GPIO Pin Configuration
MOTOR_ENA = 18  # Enable A (PWM pin for speed control)
MOTOR_IN1 = 23  # Input 1 for Motor A
MOTOR_IN2 = 24  # Input 2 for Motor A

# Initialize motor with L298N driver
pwm = PWMOutputDevice(MOTOR_ENA)
motor = Motor(forward=MOTOR_IN1, backward=MOTOR_IN2)

# Speed mapping for finger counts
speed_map = {
    0: 0,     # No fingers - motor stopped
    1: 0.5,   # 1 finger - 50% speed
    2: 0.6,   # 2 fingers - 60% speed
    3: 0.7,   # 3 fingers - 70% speed
    4: 0.8,   # 4 fingers - 80% speed
    5: 1.0    # 5 fingers - 100% speed
}

# Initialize the webcam
cap = cv2.VideoCapture(0)

# Initialize the HandDetector
```

```
detector = HandDetector(staticMode=False, maxHands=2, modelComplexity=1,
                        detectionCon=0.5, minTrackCon=0.5)

# Motor state variables
current_speed = 0
motor_running = False

def set_motor_speed(finger_count):
    """Set motor speed based on finger count"""
    global current_speed, motor_running

    # Get speed from mapping
    target_speed = speed_map.get(finger_count, 0)

    if target_speed > 0:
        # Set PWM duty cycle for speed control
        pwm.value = target_speed
        # Start motor forward
        motor.forward()
        motor_running = True
    else:
        # Stop motor
        motor.stop()
        pwm.value = 0
        motor_running = False

    current_speed = target_speed
    return target_speed

def draw_motor_status(img, finger_count, speed):
    """Draw motor status and speed indicator on the image"""
    # Draw speed gauge background
    gauge_x, gauge_y = 450, 100
    gauge_width, gauge_height = 150, 30

    # Background rectangle
    cv2.rectangle(img, (gauge_x, gauge_y), (gauge_x + gauge_width, gauge_y + gauge_height),
                  (200, 200, 200), -1)

    # Speed fill (colored based on speed)
    if speed > 0:
        fill_width = int(gauge_width * speed)
        # Color gradient from green to red based on speed
        if speed <= 0.6:
            color = (0, 255, 0)  # Green for low speed
        elif speed <= 0.8:
```

```
        color = (0, 165, 255)  # Orange for medium speed
    else:
        color = (0, 0, 255)  # Red for high speed

    cv2.rectangle(img, (gauge_x, gauge_y),
                  (gauge_x + fill_width, gauge_y + gauge_height),
                  color, -1)

# Draw border
cv2.rectangle(img, (gauge_x, gauge_y), (gauge_x + gauge_width, gauge_y + gauge_height),
              (0, 0, 0), 2)

# Display finger count
cvzone.putTextRect(img, f"Fingers: {finger_count}", (50, 50),
                  scale=2, thickness=3,
                  colorT=(255, 255, 255), colorR=(255, 0, 255))

# Display motor speed percentage
speed_percent = int(speed * 100)
speed_color = (0, 255, 0) if motor_running else (0, 0, 255)
cvzone.putTextRect(img, f"Motor Speed: {speed_percent}%", (50, 100),
                  scale=2, thickness=3,
                  colorT=(255, 255, 255), colorR=speed_color)

# Display motor status
status_text = "RUNNING" if motor_running else "STOPPED"
status_color = (0, 255, 0) if motor_running else (0, 0, 255)
cvzone.putTextRect(img, f"Status: {status_text}", (50, 150),
                  scale=1.5, thickness=2,
                  colorT=(255, 255, 255), colorR=status_color)

# Draw speed indicators for each finger count
y_pos = 250
for fingers in range(1, 6):
    speed_val = int(speed_map[fingers] * 100)
    indicator_color = (0, 255, 0) if finger_count == fingers else (150, 150, 150)
    cvzone.putTextRect(img, f"{fingers} finger(s): {speed_val}%",
                      (50, y_pos + (fingers-1)*40),
                      scale=1, thickness=1,
                      colorT=(255, 255, 255), colorR=indicator_color)

# Safety warning
if speed >= 0.8:
    cvzone.putTextRect(img, "HIGH SPEED!", (450, 50),
                      scale=1.5, thickness=2,
                      colorT=(255, 255, 255), colorR=(0, 0, 255))
```

```python
try:
    print("Hand Tracking Motor Control Started")
    print("Show fingers to control motor speed:")
    print("1 finger = 50% | 2 = 60% | 3 = 70% | 4 = 80% | 5 = 100%")
    print("Press 'q' to quit")

    # Initial motor stop
    motor.stop()
    pwm.value = 0

    while True:
        # Capture frame from webcam
        success, img = cap.read()

        if not success:
            print("Failed to read from camera")
            break

        # Find hands in the current frame
        hands, img = detector.findHands(img, draw=True, flipType=True)

        # Check if any hands are detected
        if hands:
            # Get first hand information
            hand1 = hands[0]

            # Count fingers
            fingers1 = detector.fingersUp(hand1)
            finger_count = fingers1.count(1)

            # Set motor speed based on finger count
            speed = set_motor_speed(finger_count)

            # Draw motor status and indicators
            draw_motor_status(img, finger_count, speed)

            # Print to console
            print(f"Fingers: {finger_count} | Speed: {int(speed*100)}% | Status: {'Running'

        else:
            # No hands detected - stop motor
            set_motor_speed(0)

            # Display status when no hands detected
            draw_motor_status(img, 0, 0)
```

```
                cvzone.putTextRect(img, "No hands detected - Motor STOPPED", (50, 200),
                                 scale=1.5, thickness=2,
                                 colorT=(255, 255, 255), colorR=(128, 128, 128))

            # Display instructions
            cvzone.putTextRect(img, "L298N Motor Control | Press 'q' to quit",
                             (50, 470), scale=1, thickness=1,
                             colorT=(255, 255, 255), colorR=(0, 0, 0))

            # Display the image
            cv2.imshow("Hand Tracking - DC Motor Control", img)

            # Exit on 'q' key press
            if cv2.waitKey(1) & 0xFF == ord('q'):
                print("\nStopping motor and exiting...")
                break

    except KeyboardInterrupt:
        print("\nProgram interrupted by user")

    finally:
        # Clean up - stop motor and release resources
        print("Performing cleanup...")
        motor.stop()
        pwm.value = 0
        pwm.close()
        motor.close()
        cap.release()
        cv2.destroyAllWindows()
        print("Cleanup complete - Motor stopped and resources released")
```

Project 3 Challenges

Speed Control Challenges: 1. **Custom Speed Curve**: Non-linear speed mapping (exponential curve) 2. **Smooth Ramping**: Gradual speed changes instead of instant 3. **Speed Limiting**: Maximum speed cap for safety

Advanced Features: 1. **Direction Control**: Forward/reverse with hand orientation 2. **Dual Motor**: Control two motors with both hands 3. **Emergency Stop**: Closed fist stops all motors immediately

---

Troubleshooting Guide

Camera Issues

Problem: "Failed to read from camera" **Solutions**: - Try different camera indices: `cv2.VideoCapture(1)` or `cv2.VideoCapture(2)` - Check camera permissions: `ls /dev/video*` - Ensure good lighting for hand detection

GPIO Permission Errors

Problem: "Permission denied accessing GPIO" **Solutions**: - Run with sudo: `sudo python your_script.py` - Add user to gpio group: `sudo usermod -a -G gpio $USER` - Reboot after adding to group

LED Module Issues

Problem: LED module doesn't light up **Troubleshooting Steps**: 1. Check all three connections (VCC, GND, Signal) 2. Verify wire connections are secure 3. Test with multimeter (should see ~3.3V between VCC and GND) 4. Try different GPIO pin 5. Some modules are Active LOW (try inverting logic in code) 6. Test with simple gpiozero script

Motor Problems

Problem: Motor doesn't run **Troubleshooting Steps**: 1. Is external power connected and turned on? 2. Is the ENA jumper removed from L298N? 3. Are motor connections secure to OUT1 and OUT2? 4. Test motor directly with power supply 5. Check all GPIO connections 6. Use multimeter to verify connections

Hand Detection Issues

Problem: Hand not detected consistently **Solutions**: 1. Improve lighting - bright, even lighting works best 2. Use plain background behind your hand 3. Adjust detection confidence: `detectionCon=0.3` (lower = more sensitive) 4. Clean camera lens 5. Ensure hand is within camera frame 6. Try different hand positions

---

Tips & Tricks

Hand Tracking Tips

- **Lighting is key**: Bright, even lighting gives best results
- **Plain backgrounds**: Avoid busy backgrounds behind your hand
- **Hand position**: Keep hand clearly visible, fingers spread apart
- **Distance**: Stay 1-3 feet from camera for optimal detection
- **Practice gestures**: Some finger combinations work better than others

GPIO Best Practices

- **Always clean up**: Use try/finally blocks to release GPIO pins
- **Test incrementally**: Test each component before combining
- **Use modules**: 3-pin LED modules are much safer than bare LEDs
- **Check connections**: Loose connections cause intermittent problems
- **Monitor current**: Don't exceed GPIO current limits

Code Organization Tips

- **Functions**: Break code into functions for clarity
- **Comments**: Comment your GPIO pin assignments
- **Constants**: Use constants for pin numbers at the top of your code
- **Error handling**: Add try/except blocks for robust operation
- **Status feedback**: Always provide visual/console feedback

Debugging Strategies

1. **Start simple**: Test each component individually first
2. **Add print statements**: Debug hand detection with print outputs
3. **Visual feedback**: Use on-screen indicators to see what's detected
4. **Incremental building**: Add one feature at a time
5. **Use test scripts**: Create simple test scripts for each component

Safety Reminders

- **LED modules**: Much safer than bare LEDs - use them!
- **Motor safety**: Secure motor before testing, start slow
- **Power management**: Always disconnect power when changing connections
- **Emergency stops**: Always have a way to quickly stop motors
- **Clean workspace**: Keep workspace organized and clean

Performance Optimization

- **Frame rate**: Lower resolution can improve detection speed
- **Detection confidence**: Adjust based on your lighting conditions
- **Reduce processing**: Skip frames if detection is too slow
- **Optimize imports**: Only import what you need

Next Steps & Extensions

Beginner Extensions

1. **Sound feedback**: Add buzzer sounds for each gesture
2. **Multiple patterns**: Create different LED patterns

3. **Timer functions**: Add delay and timeout features

Intermediate Extensions

1. **Web interface**: Create web page to monitor status
2. **Data logging**: Save gesture data to CSV files
3. **Custom gestures**: Train your own gesture recognition

Advanced Extensions

1. **IoT integration**: Connect to cloud services with MQTT
2. **Machine learning**: Train custom gesture models
3. **Robotics**: Build complete robotic arm with multiple motors
4. **Computer vision pipeline**: Add face detection, object tracking

Final Project Ideas

- **Simon Says Game**: LED pattern matching game
- **Robotic Car**: Complete gesture-controlled vehicle
- **Smart Home Controller**: Control multiple devices with gestures
- **Accessibility Device**: Assist people with mobility challenges

---

Resources for Further Learning

Documentation

- CVZone Hand Tracking
- GPIO Zero Documentation
- MediaPipe Hands
- Raspberry Pi GPIO Pinout

Video Tutorials

- Search for "MediaPipe Hand Tracking" tutorials
- "L298N Motor Driver" setup guides
- "Raspberry Pi GPIO" basics

Books

- "Physical Computing with Python" - Raspberry Pi Foundation
- "Computer Vision Projects with OpenCV and Python"
- "Getting Started with Raspberry Pi"

Remember: Start simple, test often, and don't be afraid to experiment! The 3-pin LED modules make these projects much more beginner-friendly, so take advantage of their built-in safety features as you learn.

Good luck with your hand tracking projects!