Motion gesture based robotic claw

Final Report of Engineering Clinics - System Design, ECS2002.

Under the guidance of Prof.Anoop Kumar Mishra

Submitted by:
JAINIL DESAI- 23BCE8912
SAROJ CHAUDHURI - 23BCE8908
URMIT KAUNDAL - 23BCE7406
SIDDHARTH NALLURI - 23BCE7959
PATIL SRUSHTEE ABHIJEET - 23BCE7553
VINEET CHANDRA DHARMAPURAM - 23BCE9176

ABSTRACT:-

This project presents the development of a gesture-controlled robotic claw system that combines real-time computer vision, servo actuation, and embedded systems using a Raspberry Pi. The core objective is to enable users to intuitively control a robotic claw using hand gestures, eliminating the need for physical controllers or complex user interfaces.
The system utilizes a standard webcam to capture live video, which is processed using MediaPipe Hands, an advanced machine learning solution that detects hand landmarks with high accuracy. These landmarks are analyzed to calculate essential control metrics, such as hand openness and positional coordinates along the X and Y axes. The data is then transmitted in real time to the Raspberry Pi through a WebSocket communication protocol, ensuring low-latency interaction between the vision system and the robotic hardware.
On the hardware side, the Raspberry Pi is connected to a PCA9685 servo driver, which controls three servo motors. These motors are responsible for managing the claw's grip (open/close), left-right movement (X-axis), and up-down motion (Y-axis). The claw reacts to the user's hand gestures in real time, translating the computed angles into smooth mechanical movements. To enhance performance and reduce jitter, exponential smoothing is applied to all input values before controlling the servos.
A key innovation in the system is the inclusion of an L-gesture-based hand switching mechanism, allowing the user to seamlessly switch control between the left and right hand. This improves the flexibility and usability of the system, especially in dynamic scenarios. The software architecture is modular and scalable, supporting future upgrades such as voice commands, object detection, and cloud-based control.
The complete system was tested under various lighting conditions and usage scenarios, demonstrating robust performance and stable control. It effectively bridges the gap between gesture recognition and physical robotics, showcasing potential applications in assistive technology, human-computer interaction, remote robotic operations, and education. The project provides a solid foundation for future advancements in gesture-controlled systems by combining computer vision, embedded control, and real-time communication.

INTRODUCTION:-

This project combines computer vision and robotics to enable intuitive, gesture-based control of a robotic claw with a Raspberry Pi. Through the use of a webcam and MediaPipe Hands, the system recognizes hand movements in real time, translating natural gestures into accurate commands for servo motors. The unique design also incorporates a gesture-based hand switching and smoothing algorithms for consistent operation, rendering it a good solution for interactive robotics.

Background of the Project

In recent years, the integration of **computer vision** and **robotics** has led to the development of intelligent systems that can interact with humans more naturally and intuitively.
Traditional methods of controlling robotic arms or claws often rely on physical input devices such as joysticks, buttons, or pre-programmed paths. While effective, these methods can be unintuitive, especially for users unfamiliar with robotics or in situations where hands-free interaction is preferable.
Advancements in **hand tracking and gesture recognition** technologies have opened new possibilities in the field of **human-computer interaction (HCI)**. With the introduction of frameworks like **MediaPipe Hands** by Google, real-time hand detection has become more accessible and reliable, even with standard webcams and minimal computational resources. These technologies enable machines to interpret human gestures as input commands, making interaction more seamless.
On the hardware side, the **Raspberry Pi** has become a popular platform for prototyping embedded systems due to its affordability, compact size, and compatibility with a wide range of peripherals. Combined with modules like the **PCA9685 servo driver**, it becomes a powerful controller for robotic applications that require precise and synchronized movement.
This project leverages these technological advancements to create a **gesture-controlled robotic claw system**. By tracking the user's hand gestures and translating them into real-time control signals, the project aims to provide an intuitive and low-cost solution for remote or contactless manipulation. This approach not only enhances user interaction but also paves the way for potential applications in fields such as **assistive technology**, **education**, **automation**, and **rehabilitation**. The background of this work lies in addressing the need for more **natural, interactive, and accessible robotic control systems**.

MOTIVATION AND PROBLEM STATEMENT:-

Context:
Millions of people worldwide live with disabilities affecting upper limb mobility, such as spinal cord injuries, cerebral palsy, muscular dystrophy, or amputations. These individuals often struggle with daily tasks like eating, drinking, operating appliances, or manipulating objects, relying heavily on caregivers or restrictive assistive devices. Traditional solutions, such as mouth-operated joysticks, sip-and-puff systems, or eye-tracking interfaces, are often expensive, require extensive training, or lack the precision and versatility needed for complex tasks. A gap exists in accessible, intuitive, and affordable technologies that empower users to regain independence in daily activities.

Challenges:

1. Limited Accessibility: Many assistive devices require fine motor control or physical strength, which users with paralysis or tremors may lack.
2. High Cost: Advanced robotic arms or prosthetics are often prohibitively expensive for individuals or healthcare systems.
3. Lack of Customization: Rigid designs do not adapt to users' varying levels of mobility or evolving needs.

PLAN OF ACTION:-

1. Identify and procure all required components.
2. Document the problem statement, objectives, and budget.
3. Assemble the robotic claw and connect it to the Raspberry Pi.
4. Set up the camera module and servo motor controller. Install Raspberry Pi OS and necessary software.
5. Document the hardware setup and initial testing.
6. Develop the gesture recognition model using AI/ML tools.
7. Integrate the gesture recognition system with the robotic claw.
8. Test and optimize the system for accuracy and responsiveness.
9. Document the software development and integration process.
10. Conduct final testing and optimize the system for performance.
11. Prepare the final demonstration and compile all documentation. Submit the final report and present the project.

Objectives of the Proposed Work

1. To design and implement a gesture-controlled robotic claw system using a Raspberry Pi, enabling hands-free operation through real-time hand tracking.
2. To utilize computer vision techniques—specifically MediaPipe Hands—for detecting hand landmarks and interpreting gestures for robotic control.
3. To develop a WebSocket-based communication system that transmits gesture data to the Raspberry Pi for real-time servo actuation.
4. To integrate and control multiple servo motors via the PCA9685 module, allowing precise control of claw movements along the X, Y axes and grip (open/close).
5. To introduce a gesture-based hand switching mechanism to improve user control and interaction flexibility.
6. To enhance servo stability and responsiveness using exponential smoothing techniques for gesture input values.
7. To test and evaluate the system under real-world conditions, ensuring accurate gesture detection, stable communication, and reliable servo response.
8. To demonstrate a cost-effective and intuitive robotic control solution that bridges computer vision and embedded hardware systems for real-time applications.

PROCEDURE/METHODOLOGY:-

- **Hardware Installation:**
    - Selected a Raspberry Pi as the main controller for its compact size and flexibility.
    - Installed a PCA9685 servo driver board and connected it via the I2C bus.
    - Wired servos to designated channels on the PCA9685 to control the claw's open/close mechanism and its X (left/right) and Y (up/down) movements.
    - Mounted a webcam in a fixed position to capture clear, real-time video for accurate hand tracking.
- **Software Setup:**
    - Configured the development environment using Python.
    - Utilized libraries such as asyncio and websockets for asynchronous communication between devices.
    - Integrated specialized hardware libraries (board, busio, adafruit_pca9685) for servo control.
    - Implemented MediaPipe Hands for real-time hand detection and gesture recognition.
    - Developed modular Python scripts to separate hand tracking from servo control logic.
- **System Integration:**
    - Mapped hand metrics (openness and position) obtained from the MediaPipe processing to corresponding servo angles.
    - Established a WebSocket connection between the hand tracking module and the Raspberry Pi server to transmit control commands.
    - Incorporated a gesture-based hand switching mechanism (using an L-shaped gesture) to alternate control between hands.
    - Applied exponential moving average smoothing to the hand metrics to reduce jitter and ensure smooth servo operations.
- **Testing and Evaluation:**
    - Conducted initial tests to verify individual component functionality:
        - Checked the accuracy of hand detection and gesture recognition using the webcam.
        - Verified correct servo responses to the specified angle commands.
    - Evaluated real-time performance by measuring WebSocket latency and responsiveness during rapid hand movements.

- Tested system robustness under varying lighting conditions and backgrounds to ensure consistent MediaPipe performance.
- Iteratively refined both software and hardware configurations based on test results to improve overall system reliability and control accuracy.

Results and Discussion

The system demonstrated effective real-time control of a robotic claw using hand gestures, with both the computer vision and servo control components performing reliably. During testing, the MediaPipe Hands module accurately detected hand landmarks and computed relevant metrics such as hand openness and position, which translated smoothly into control commands for the servos. The responsiveness of the WebSocket communication allowed near-instantaneous adjustments of the claw's position and grip, validating the approach for interactive robotics applications.

Key observations include:

- Accurate Gesture Recognition:
  The system reliably distinguished between different hand gestures, including the specialized L-shaped gesture for hand switching. This feature contributed to a more versatile control interface, although it required precise calibration to minimize false positives during rapid hand movements.
- Real-Time Performance:
  The integration of MediaPipe Hands with the Raspberry Pi's WebSocket server enabled real-time feedback and control. Despite the processing demands of video capture and hand landmark detection, the system maintained a satisfactory frame rate and minimal latency, demonstrating the feasibility of deploying such technology in interactive settings.
- System Limitations and Improvements:
  While the overall performance was robust, certain limitations were observed. The sensitivity of gesture detection under variable lighting conditions and the occasional misinterpretation of rapid hand movements suggest potential areas for further refinement. Future iterations could incorporate adaptive thresholding and enhanced gesture recognition algorithms to improve accuracy under diverse operational environments.

Overall, the project successfully combines computer vision, real-time networking, and hardware control, showcasing a promising direction for intuitive, gesture-based interfaces in robotics.

Conclusion

The proposed project effectively demonstrates a gesture-controlled robotic claw system powered by real-time hand tracking and servo motor control. By leveraging MediaPipe Hands for gesture recognition and a WebSocket server on the Raspberry Pi for communication, the system enables intuitive, natural hand motion to control the robotic claw. The integration of exponential smoothing ensures fluid and stable servo movements, while the inclusion of a hand-switching gesture enhances flexibility and enriches user interaction. Overall, the project seamlessly blends computer vision, real-time networking, and embedded systems to deliver a responsive, interactive, and user-friendly robotic interface.

Future Scope

1. Wireless Hand Tracking Module
   Integrate the hand tracking system into a standalone device (such as a mobile phone or camera module) to increase portability and enable fully wireless operation.
2. Autonomous Object Detection and Grasping
   Incorporate AI-based object detection to allow the robotic claw to recognize, approach, and pick up objects autonomously based on environmental context or predefined instructions.
3. Expanded 3D Movement and Extra Degrees of Freedom (DOF)
   Enhance the system's flexibility by adding additional servo motors to support full 3D movement—such as forward-backward motion, wrist rotation, and more nuanced positioning.
4. Hybrid Voice and Gesture Control
   Combine voice commands with hand gestures for a richer and more dynamic control system, especially useful in smart environments or assistive technology applications.
5. Mobile App or Web-Based Interface
   Develop a user-friendly interface for smartphones or browsers to allow remote monitoring and control of the robotic claw.
6. Gesture Customization via Machine Learning
   Enable users to train and personalize their own gesture sets using machine learning, enhancing accessibility and tailoring the system to individual preferences.
7. Real-World Applications in Industry and Assistance
   Extend the system's utility to industrial automation, prosthetics, or assistive devices for individuals with disabilities, underlining its potential for meaningful real-world impact.

IMAGES OF THE PROTOTYPE:-

References:-

G. Bradski, "The OpenCV Library," Dr. Dobb's Journal of Software Tools, 2000. [Online]. Available: https://docs.opencv.org/

Google Developers, "MediaPipe Hands," 2023. [Online]. Available: https://developers.google.com/mediapipe/solutions/vision/hand_landmarker

TensorFlow, "TensorFlow.js: Machine Learning for the Web," 2023. [Online]. Available: https://www.tensorflow.org/js

Raspberry Pi Foundation, "Raspberry Pi Documentation," 2023. [Online]. Available: https://www.raspberrypi.com/documentation/computers/raspberry-pi.html

H. Ahmed, J. Alzubi, and Y. Ryu, "Hands-Free Control Using AI-Based Gesture Recognition," arXiv preprint arXiv:2109.09078, 2021. [Online]. Available: https://arxiv.org/abs/2109.09078

NVIDIA, "CUDA Toolkit Documentation," 2023. [Online]. Available: https://developer.nvidia.com/cuda-toolkit

CODE:-

RASPBERRYPI SERVER FILE:-

```
import asyncio
import websockets
import board
import busio
from adafruit_pca9685 import PCA9685
from adafruit_motor import servo
import time  # Added for cooldown timing

# Initialize PCA9685 Servo Driver
i2c = busio.I2C(board.SCL, board.SDA)
pca = PCA9685(i2c)
```

```
pca.frequency = 50
# Assign servo channels
claw_servo = servo.Servo(pca.channels[0])  # Claw openness
y_servo = servo.Servo(pca.channels[1])  # Up-down
x_servo = servo.Servo(pca.channels[4])  # Left-right
# Open and close angles
OPEN_ANGLE = 0
CLOSE_ANGLE = 105
# Added for hand switching cooldown
LAST_SWITCH_TIME = 0
SWITCH_COOLDOWN = 0.5  # seconds
def move_servo(servo, target_angle):
    """Moves the servo directly to the target angle."""
    servo.angle = target_angle
async def handle_client(websocket):  # Added path parameter
    """Receives openness, X, and Y angles and moves the servos."""
    global LAST_SWITCH_TIME  # Added for cooldown tracking
    async for message in websocket:
        current_time = time.time()
        # Skip commands during cooldown period after switch
        if current_time - LAST_SWITCH_TIME < SWITCH_COOLDOWN:
            continue
        try:
            if message == "SWITCH":  # Added hand switch message handling
                print("Hand switch detected - ignoring commands for 1.5s")
                LAST_SWITCH_TIME = current_time
                continue
            openness, x_angle, y_angle = map(float, message.split(","))
            claw_angle = CLOSE_ANGLE - ((openness / 100) * (CLOSE_ANGLE -
OPEN_ANGLE))
            print(f"Openness: {openness:.2f}%, Claw: {claw_angle:.2f}, X: {x_angle:.2f}, Y:
{y_angle:.2f}")
            move_servo(claw_servo, claw_angle)
            move_servo(x_servo, x_angle)
            move_servo(y_servo, y_angle)
        except ValueError:
            print(f"Invalid message received: {message}")
async def main():
    """Starts the WebSocket server."""
    async with websockets.serve(
        handle_client,
        "0.0.0.0",
        8765,
        ping_interval=None  # Added to prevent servo stuttering
    ):
        print("WebSocket server started on ws://0.0.0.0:8765")
        await asyncio.Future()

if _name_ == "_main_":
    asyncio.run(main())
```

```python
import cv2
import mediapipe as mp
import numpy as np
import asyncio
import websockets
import time

# Initialize MediaPipe Hands with GPU acceleration
mp_hands = mp.solutions.hands
mp_drawing = mp.solutions.drawing_utils
hands = mp_hands.Hands(
    static_image_mode=False,
    max_num_hands=2,
    min_detection_confidence=0.7,
    min_tracking_confidence=0.7,
    model_complexity=1  # Balanced performance/accuracy
)

# WebSocket setup
RASPBERRY_PI_IP = "192.168.176.81"
PORT = 8765
WS_URL = f"ws://{RASPBERRY_PI_IP}:{PORT}"

# Performance tracking
prev_time = 0
fps_list = []

# Hand control variables
ACTIVE_HAND = "right"
LAST_SWITCH_TIME = 0
SWITCH_COOLDOWN = 2
MESSAGE_DISPLAY_TIME = 3
message_display_start = 0
show_switch_message = False

# Exponential Moving Average Smoothing
SMOOTHING_ALPHA = 0.3
smoothed_openness = None
smoothed_x_angle = None
smoothed_y_angle = None

def draw_hand_landmarks(frame, hand_landmarks):
    """Draw landmarks with gradient colors"""
    for i, landmark in enumerate(hand_landmarks.landmark):
        x, y = int(landmark.x * frame.shape[1]), int(landmark.y * frame.shape[0])
        color = (i * 12, 255 - i * 12, 255)
```

```
        cv2.circle(frame, (x, y), 6, color, -1)

def draw_modern_overlay(frame, fps):
    """Draw FPS and info overlay"""
    h, w = frame.shape[:2]
    overlay = frame.copy()
    cv2.rectangle(overlay, (10, h-100), (w-10, h-10), (0, 0, 0), -1)
    alpha = 0.4
    cv2.addWeighted(overlay, alpha, frame, 1 - alpha, 0, frame)

    fps_list.append(fps)
    if len(fps_list) > 30:
        fps_list.pop(0)
    avg_fps = sum(fps_list) / len(fps_list)
    cv2.putText(frame, f"FPS: {avg_fps:.1f}", (w - 120, 30),
                cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 255), 2, cv2.LINE_AA)
    return frame

def is_l_gesture(landmarks):
    """Detect L-shaped hand gesture"""
    wrist = landmarks[0]
    thumb_tip = landmarks[4]
    index_tip = landmarks[8]
    middle_tip = landmarks[12]
    ring_tip = landmarks[16]
    pinky_tip = landmarks[20]

    # Check if fingers are closed
    closed_fingers = all(
        np.linalg.norm(np.array([f.x, f.y]) - np.array([wrist.x, wrist.y])) < 0.1
        for f in [middle_tip, ring_tip, pinky_tip]
    )

    # Calculate angle between thumb and index
    thumb_vec = np.array([thumb_tip.x - wrist.x, thumb_tip.y - wrist.y])
    index_vec = np.array([index_tip.x - wrist.x, index_tip.y - wrist.y])
    angle = np.degrees(np.arccos(
        np.dot(thumb_vec, index_vec) / (np.linalg.norm(thumb_vec) *
np.linalg.norm(index_vec))
    ))
    return closed_fingers and (50 < angle < 100)

def compute_hand_openness(landmarks):
    """Calculate hand openness percentage"""
    finger_tips = [4, 8, 12, 16, 20]
    distances = []
    for tip in finger_tips:
        dist = np.linalg.norm(np.array([landmarks[tip].x, landmarks[tip].y]) -
```

```
                        np.array([landmarks[0].x, landmarks[0].y]))
        distances.append(dist)
    min_open, max_open = 0.05, 0.3
    avg_distance = np.mean(distances)
    return np.clip((avg_distance - min_open) / (max_open - min_open), 0, 1) * 100

def compute_angles(landmarks, frame_width, frame_height):
    """Calculate X and Y servo angles"""
    x_angle = np.clip((landmarks[0].x * frame_width / frame_width) * 180, 0, 180)
    y_angle = np.clip(90 + ((landmarks[0].y * frame_height / frame_height) * 90), 90, 180)
    return x_angle, y_angle

async def send_command(openness, x_angle, y_angle):
    """Send command to Raspberry Pi"""
    async with websockets.connect(WS_URL) as websocket:
        command_data = f"{openness:.2f},{x_angle:.2f},{y_angle:.2f}"
        await websocket.send(command_data)
        print(f"Sent: {command_data}")

# Main processing loop
cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1920)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 1080)

previous_openness = None
previous_x_angle = None
previous_y_angle = None

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

        # Mirror and convert to RGB
        frame = cv2.flip(frame, 1)
        rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

        # Process with MediaPipe
        results = hands.process(rgb_frame)
        current_time = time.time()

        if results.multi_hand_landmarks:
            for hand_landmarks, handedness in zip(results.multi_hand_landmarks,
results.multi_handedness):
                landmarks = hand_landmarks.landmark
                hand_label = handedness.classification[0].label.lower()
```

```
            # Check for L-gesture to switch hands
            if is_l_gesture(landmarks) and (current_time - LAST_SWITCH_TIME) >
SWITCH_COOLDOWN:
                ACTIVE_HAND = "left" if ACTIVE_HAND == "right" else "right"
                LAST_SWITCH_TIME = current_time
                message_display_start = current_time
                show_switch_message = True
                print(f"Switched dominant hand to: {ACTIVE_HAND}")

            if hand_label == ACTIVE_HAND:
                # Draw landmarks
                mp_drawing.draw_landmarks(frame, hand_landmarks,
mp_hands.HAND_CONNECTIONS)
                draw_hand_landmarks(frame, hand_landmarks)

                # Compute metrics
                openness = compute_hand_openness(landmarks)
                x_angle, y_angle = compute_angles(landmarks, frame.shape[1], frame.shape[0])

                # Apply smoothing
                if smoothed_openness is None:
                    smoothed_openness = openness
                    smoothed_x_angle = x_angle
                    smoothed_y_angle = y_angle
                else:
                    smoothed_openness = SMOOTHING_ALPHA * openness + (1 -
SMOOTHING_ALPHA) * smoothed_openness
                    smoothed_x_angle = SMOOTHING_ALPHA * x_angle + (1 -
SMOOTHING_ALPHA) * smoothed_x_angle
                    smoothed_y_angle = SMOOTHING_ALPHA * y_angle + (1 -
SMOOTHING_ALPHA) * smoothed_y_angle

                # Send command if significant change
                if (abs(smoothed_openness - (previous_openness or 0))) > 2 or \
                  (abs(smoothed_x_angle - (previous_x_angle or 0))) > 2 or \
                  (abs(smoothed_y_angle - (previous_y_angle or 0))) > 2:
                    asyncio.run(send_command(smoothed_openness, smoothed_x_angle,
smoothed_y_angle))
                    previous_openness = smoothed_openness
                    previous_x_angle = smoothed_x_angle
                    previous_y_angle = smoothed_y_angle

                # Display values
                cv2.putText(frame, f"Active: {ACTIVE_HAND.upper()}", (10, 30),
                            cv2.FONT_HERSHEY_DUPLEX, 0.7, (0, 255, 0), 2)
                cv2.putText(frame, f"Openness: {smoothed_openness:.2f}%", (10, 70),
                            cv2.FONT_HERSHEY_DUPLEX, 1, (0, 255, 0), 2)
                cv2.putText(frame, f"X Servo: {smoothed_x_angle:.2f}degree", (10, 110),
```

```
                        cv2.FONT_HERSHEY_DUPLEX, 1, (255, 255, 0), 2)
                cv2.putText(frame, f"Y Servo: {smoothed_y_angle:.2f}degree", (10, 150),
                        cv2.FONT_HERSHEY_DUPLEX, 1, (255, 0, 255), 2)

        # Calculate and display FPS
        curr_time = time.time()
        fps = 1/(curr_time - prev_time)
        prev_time = curr_time
        frame = draw_modern_overlay(frame, fps)

        # Show switch message if needed
        if show_switch_message and (current_time - message_display_start) <
MESSAGE_DISPLAY_TIME:
            cv2.putText(frame, f"Switched to {ACTIVE_HAND} hand!", (495, 650),
                    cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 255), 2)
        else:
            show_switch_message = False

        # Display
        cv2.imshow("Hand Tracking", frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

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