Computer Vision Controlled Humanoid Robotic Arm

M S Firdouse,
School of Engineering,
SLIIT City Uni
mohamedshuhaib38@gmail.com

L Benorith
School of Engineering,
SLIIT City Uni
benorith.l@sliit.lk

Abstract **- This paper presents the design and implementation of a low-cost, vision-based gesture-controlled humanoid robotic arm that mimics human hand and wrist movements in real time. The system uses a USB webcam and MediaPipe for hand landmark detection, OpenCV for image processing, and a Raspberry Pi 4 to compute landmark vectors and control servo motors via a PCA9685 driver. Calibration modes were introduced for each joint to ensure accurate servo mapping. The solution supports full gesture-based manipulation of a five-fingered robotic hand, including wrist orientation, with minimal latency and no physical contact. The system provides a more intuitive and natural method for robotic arm control compared to traditional input devices and has potential applications in prosthetics, automation, and human-robot interaction.**

Keywords **- Computer Vision, Robotic Arm, OpenCV, MediaPipe**

I. INTRODUCTION

Robotic arms are programmable mechanical manipulators designed to replicate the motions and functionalities of a human arm. These devices are typically structured as kinematic chains, consisting of rigid links connected by joints that allow controlled movement. The terminal part of the chain is referred to as the "end-effector," commonly designed as a gripper to interact with objects. Robotic arms have become essential tools in various industries, including automotive manufacturing, electronics assembly, medical surgery, hazardous material handling, and space exploration. One of the primary advantages of robotic arms is their ability to operate in environments that are dangerous, repetitive, or inaccessible to humans. A notable example is NASA's robotic exploration missions using the MER rovers, where robotic arms equipped with scientific instruments were remotely controlled to study Martian terrain. Traditional robotic arm systems are typically controlled using manual input methods such as keypads, joysticks, or pre-programmed instructions. While effective, these methods often require extensive calibration, technical expertise, and lack natural interaction. To overcome these limitations, alternative control paradigms such as voice commands, wearable sensor gloves, and vision-based gesture recognition have emerged. Among these, vision-based control systems stand out for their non-intrusive operation, enabling intuitive human–robot interaction without the need for physical contact or specialized wearables. Leveraging advancements in real-time image processing and machine learning, gesture recognition via standard cameras has become a viable solution for robotic control.

This paper presents a computer vision–driven humanoid robotic arm system that uses a USB camera, OpenCV, and MediaPipe to detect and interpret hand gestures in real time. The system maps human finger positions and wrist orientation to corresponding servo motor angles using a calibration-based approach. The entire control process is executed on a Raspberry Pi 4, ensuring portability and low cost, making the solution suitable for applications in prosthetics, education, and assistive robotics.

II. PROBLEM STATEMENT AND SCOPE

Despite the increasing use of robotic arms in industrial and assistive domains, existing control methods often depend on specialized hardware such as gloves, EMG sensors, or voice recognition modules. These systems suffer from limitations including hardware complexity, calibration overhead, user discomfort, and susceptibility to environmental noise. The problem this paper addresses is the lack of a low-cost, contactless, and easily deployable control method for humanoid robotic arms that can replicate complex human gestures in real time.

This project focuses on a vision-based approach that leverages MediaPipe and OpenCV to track hand gestures without requiring wearable devices. The scope is limited to finger and wrist movement replication using six degrees of actuation, without incorporating full-arm dynamics or grasp force feedback. This design makes the system suitable for educational, prototyping, and assistive use cases, but not yet optimized for industrial-grade manipulation tasks or high-speed dynamic control.

III. RELATED WORK

Robotic arm control has evolved from hardware-intensive, wearable systems to more flexible, vision-driven architectures. Early approaches used infrared emitters and glove-mounted sensors for gesture recognition, achieving fast response but requiring custom hardware and limited gesture vocabulary [1]. To improve accessibility, vision-only methods emerged. For example, RGB color segmentation was used to track hand motion via webcam input, enabling real-time control without wearables, though performance degraded under poor lighting or

https://doi.org/10.54389/FLMW1698

varied skin tones [2]. Others employed compact vision modules like OpenMV to detect objects and apply inverse kinematics, but these systems faced processing limitations [3]. Sensor-based alternatives, such as gloves with flex sensors and accelerometers, offered high fidelity but reduced comfort and required calibration [4]. Voice-controlled arms also surfaced, using wireless microcontrollers and speech recognition, though often limited by noise and accent variability [5].

Biologically inspired methods like EMG-based control added intuitive prosthetic capability but required precise sensor placement and regular tuning [6]. More recently, computer vision systems using OpenCV and MediaPipe on embedded platforms have enabled robust, camera-based gesture recognition with minimal hardware. These systems have been used for real-time control and IOT-enabled remote actuation, though they remain sensitive to occlusion and network reliability. Other strategies include physical mimicry through linked miniature arms, and kinematic simulations of multi-DOF humanoid limbs. Image-based grasping techniques using PiCamera further show the role of embedded vision in object manipulation tasks.

Overall, these studies reflect a growing shift toward multimodal, software-centric solutions that aim to balance precision, cost, and usability in robotic arm control system

IV. System Architecture

Hardware Design

The proposed system consists of six MG995 servo motors (five for fingers, one for wrist), controlled by a PCA9685 servo driver via I2C. A 5V/3A external power supply provides consistent power for motors. A USB webcam captures video input, while the Raspberry Pi 4 Model B (8GB RAM) acts as the control core.

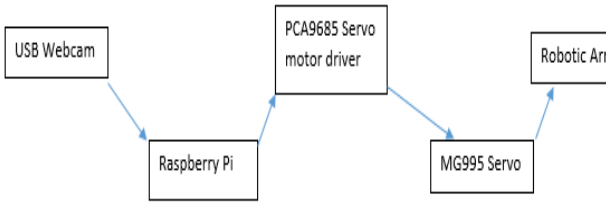

Figure 20: Hardware Design

Figure 1 shows the overall hardware flow of the project. The frames captured by the Webcam is sent to the raspberry pi for processing, that sends the relevant angles into the PCA9685 motor driver which in turn rotates the servo motors to required angles.

Figure 21: Final Robotic Arm

Software Design

Python-based software is used, leveraging OpenCV for video capture and pre-processing, and MediaPipe for hand landmark detection. The `cvzone` wrapper facilitates simpler landmark access. The Adafruit `ServoKit` library controls servo outputs, and calibration profiles are stored in JSON files.

Functional Overview

- **Hand Tracking:** MediaPipe detects 21 hand landmarks with high accuracy and robustness to hand orientation.

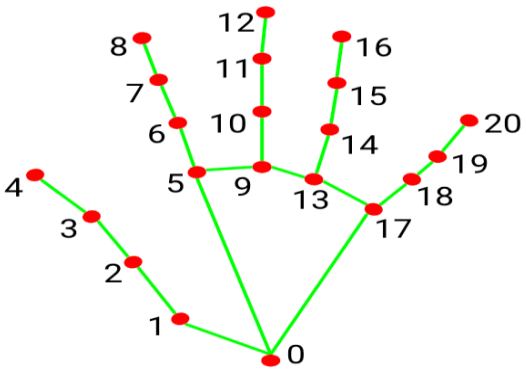

Figure 22: Hand Landmarks

- **Vector Analysis:** Joint angles are calculated using vector geometry and dot product techniques.

- **Servo Mapping:** Each finger angle is scaled to servo values using calibration bounds. Custom mappings allow adjustments for mechanical variance.
- **Calibration Mode:** Allows per-servo tuning for extended, bent, and scaling parameters, which are stored in JSON.
- **Failsafe and Shutdown**: atexit routines and interrupt handling ensure the servos are returned to neutral when tracking is lost or the program exits.
- **Live Feedback:** Visual indicators for hand detection and servo angle overlays are displayed in the video window.

V. GESTURE MAPPING AND CONTROL

Angle Calculation

For each finger, angles are calculated between landmark pairs: the fingertip and the proximal interphalangeal (PIP) joint. The angle is determined using the arctangent function from the vector formed by the landmark coordinates. The computed raw angle is then scaled using a user-defined calibration multiplier to account for hand variation. The system limits finger movement to a maximum of 90° for safety and realism.

Calibration Modes

Three interactive calibration modes are supported within the Python script:

- Bent Mode (b): Assigns the angle value for a fully curled finger position.
- Extended Mode (e): Sets the angle corresponding to a fully open finger.
- Scale Mode (s): Adjusts sensitivity by linearly scaling the mapped angle from detected vector motion.

Servo selection (1–6), mode switching (c to toggle), and real-time angle adjustments using arrow keys (left/right) are implemented in the OpenCV window interface. Calibrations are saved in a JSON file and persist across sessions.

Servo Actuation

Each calculated finger or wrist angle is clamped between 0° and 270°, the safe range of MG995 servos. The ServoKit interface from Adafruit is used to send PWM commands over I2C to the PCA9685 driver. On startup or loss of hand detection, the system resets all servo positions to the neutral 'extended' state using a pre-defined method. Servo smoothing and safety reset are ensured using exception handling and the atexit module.

System Operation Workflow

The system begins by initializing the USB webcam, loading calibration parameters from a locally stored JSON file, and setting up the MediaPipe hand tracking module with OpenCV. Upon capturing each frame, the image is processed to detect the presence of one or more hands. If a hand is identified, MediaPipe extracts 21 landmark points from the detected hand.

These landmarks are analyzed to calculate vector angles for each finger by comparing the fingertip and proximal joint coordinates. The raw angles are scaled and mapped to servo-compatible angles using pre-calibrated values. Each servo's final target angle is clamped to safe operational limits (0°–270°) and sent via I2C to the PCA9685 driver through the Adafruit ServoKit library.

In the absence of hand detection or if the frame is invalid, the servos are reset to a neutral, extended position to prevent erratic movement. Additionally, when the user exits the program or interrupts execution, the atexit handler ensures all servo motors return to safe default positions. The system is designed to run continuously and robustly under varying lighting and motion conditions.

VI. RESULTS AND ANALYSIS

A. Tracking and Responsiveness

The hand tracking pipeline was evaluated on the Raspberry Pi 4 with 8GB RAM. The average detection and servo update loop maintained 10–15 FPS, resulting in a real-time response latency of approximately 1s between gesture execution and servo motion. This responsiveness was consistent under stable indoor lighting and single-hand usage.

B. Hand Classification and Role Assignment

The system utilizes MediaPipe's built-in hand classification to differentiate between the user's right and left hand. During each frame capture, if two hands are detected, the library provides the type of attribute ("Right" or "Left") for each hand instance. This is essential for ensuring consistent control logic and servo mapping. In the current implementation:

- Only the right hand is used to control the robotic arm, as determined by ["type"] == "Right".
- If both hands are in the frame, the system prioritizes the right hand and ignores the left, avoiding conflicting gesture inputs.
- If no "Right" hand is detected, the system falls back to a neutral servo position for safety.

Table 17: Right- & Left-Hand Conditions

Condition	Outcome
Right hand only in frame	Full control enabled
Left hand only in frame	Ignored by control logic

Both hands in frame	Right hand selected for processing
Right hand leaves frame	System resets servos to neutral position

The hand identification was consistently reliable across test sequences, including fast switching between hands. Misclassification did not occur in controlled lighting. However, occasional flipping (e.g., left hand labeled as right) was observed when hands were rotated sideways or partially occluded. This effect was minimized by instructing users to present hands palm-forward to the camera.

C. Calibration Mode Performance

To ensure accurate gesture replication across different servo alignments and user hand styles, the system implements a fully interactive calibration mode. This feature enables real-time adjustment of each servo's range of motion and sensitivity, directly impacting servo precision and usability.

1) Calibration Effectiveness

Before calibration the tensioned strings couldn't bend the finger properly due to being too tight or too slack. After using the calibration mode in the bent and extended angles it was possible to keep the servo motor angle in the desired position and properly tension the strings to make the movement as effective as possible.

Furthermore, the scale calibration was also useful in making the error low, so that the motor would spin rapidly or slowly corresponding to the finger angle shown in the frame. It helped in making sure the fingers didn't get stuck halfway into bending or extending.

2) User Interaction and Usability

Calibration was conducted through a keyboard-driven interface, with visual feedback overlayed on the OpenCV video stream. Users were able to:

- Select servo channels (1 to 6)
- Choose calibration modes (b for bent, e for extended, s for scale)
- Adjust parameters incrementally using arrow keys

The calibration process was intuitive and fast, requiring under 2 minutes on average to fully configure all six servos.

3) Persistence and Reliability

All calibration parameters were saved automatically in a persistent JSON file. System testing confirmed that:

- Calibration values were retained across reboots.
- The atexit safeguard successfully wrote updated values even during abrupt terminations.
- Manual re-tuning was unnecessary unless servo positions or mechanical alignment changed.

4) Servo Response Accuracy

The servo system exhibited reliable and consistent behavior in replicating human hand gestures. After calibration, the finger and wrist movements closely matched user input, with noticeably smoother transitions and more natural motion curves. Repetitive gestures produced nearly identical servo responses across cycles, indicating stable actuation and mapping accuracy.

The system responded quickly to changes in hand posture, maintaining real-time synchronization between detected gestures and mechanical motion. Even during extended operation, the servos held their positions steadily without drift or jitter.

VII. CONCLUSION

The vision-controlled humanoid robotic arm in this project is implemented using Python 3, running on a Raspberry Pi 4 platform. The real-time video input is processed using OpenCV, while MediaPipe handles the landmark detection for finger and wrist gestures. The communication between the hand tracking system and servo controller is managed through the I2C interface connected to a PCA9685 module. Raspberry Pi is responsible for gesture recognition, landmark angle processing, and mapping to servo motion to avoid the limitations of microcontroller processing speed.

The PWM driver controls the movement of MG995 servo motors, and the motion response is stable and consistent across multiple gesture cycles. The servo signals can smoothly and accurately drive finger articulation and wrist rotation, confirmed through visual testing. During runtime, the system continuously tracks the user's hand, calculates joint angles, and sends appropriate commands to each servo channel to mimic hand posture.

In actual performance, the robotic arm was able to follow basic gestures such as hand opening, closing, and directional pointing. With the integration of calibration and gesture mapping modules, the system exhibits flexibility and adaptability for different users and mechanical configurations. The final build is low-cost, portable, and easy to set up, making it suitable for education, research, and assistive applications. In future iterations, the addition of force sensors or smart feedback modules can further enhance motion fidelity and bring the robotic arm closer to natural human interaction goals.

REFERENCES

[1] S. Bularka, R. Szabo, M. Otesteanu, and M. Babaita, "Robotic Arm Control with Hand Movement Gestures," in Proc. 2018 41st Int. Conf. on Telecommunications and Signal Processing (TSP), Athens, Greece, 2018, pp. 1–4.

[2] P. S. Lengare and M. E. Rane, "Human Hand Tracking Using MATLAB to Control Arduino-Based Robotic Arm," in Proc. 2015 Int. Conf. on Pervasive Computing (ICPC), Pune, India, 2015, pp. 1–6.

[3] J. Wang and H. Peng, “Object Grabbing of Robotic Arm Based on OpenMV Module Positioning,” in Proc. 2023 2nd Int. Conf. on Artificial Intelligence and Computer Information Technology (AICIT), Yichang, China, 2023, pp. 45–50.

[4] D. Crăciun, N. Bălăceanu, A. Chirită, and C. Ciobanu, “Robotic Arm Control via Hand Movements,” in Proc. 2022 Int. Symp. on Electronics and Telecommunications (ISETC), Timisoara, Romania, 2022, pp. 1–4.

[5] R. S. Kanash, S. E. Alavi, and A. A. Abed, “Design and Implementation of Voice-Controlled Robotic Arm,” in Proc. 2021 Int. Conf. on Communication & Information Technology (ICICT), Basrah, Iraq, 2021, pp. 22–27.

[6] P. S. Keerthi, S. Al Mamun, R. A. Tonima, S. A. Shanta, A. Ahmed, and M. S. R. Zishan, “Design and Implementation of a Human Prosthetic Hand,” in Proc. 2021 2nd Int. Conf. on Robotics, Electrical and Signal Processing Techniques (ICREST), Dhaka, Bangladesh, 2021, pp. 104–109.