First Final Version – **Awesome Robot Hand**

System Introduction

In many high-risk production environments, precise and repetitive manual tasks are common, especially in places like nuclear power plants where maintenance and repairs are required. Under these conditions, not only meticulous hand movements are needed, but also rapid responses to unexpected situations. To minimize the risk to workers, we have developed a robotic hand. This robot hand allows operators to carry out complex and hazardous tasks from a safe environment, effectively ensuring their safety and health through remote control.

Our idea for the Robot Hand was conceived through several brainstorming sessions. The project was developed for Cornell's ECE 5725 Design with Embedded Operating Systems class, which focuses on the design of microcontroller-based systems using embedded Linux. To accomplish this, we incorporated programming capabilities that allow the Robot Hand to complete pre-programmed tasks. This feature provides efficiency and safety in scenarios where repeating the same task could be potentially harmful or tedious for a human. Additionally, we integrated Flask for data communication, enabling remote operation of the Robot Hand via an external camera's real-time data stream.

The robot hand has three modes of control: video mode, precision mode, and pre-programmed mode. In video mode, users can place their hand in front of the camera and the robot hand will mimic the hand's movements in real-time. In precision mode, users can manipulate the simulated hand's joints displayed on the touch screen to control the robot hand's precise movement. In pre-programmed mode, users can pre-edit action sequences, and the robot hand will execute the sequence accordingly.

Design overview

The robot hand (1) is controlled by Raspberry Pi4. The outer casing of the device is made from 3D printing, and it employs six servos to control finger flexion and joint rotation. A Pi camera is

utilized to capture real hand movements video stream, deploying the *MediaPipe* model onto a Raspberry Pi for processing the images captured by the camera, and retrieving the real-time coordinates of the hand's joints extracted by *MediaPipe* model from the video stream to control the motors in real-time. It is worth noting that, as of the completion of this project, only the Raspberry Pi 4 running on Buster is compatible with *MediaPipe*. Consequently, to ensure compatibility with *MediaPipe*, we downgrade the version of Raspberry Pi 4 from Bullseye to Buster.

Control of the Robot Hand is facilitated through a Python script using pygame, hosted on a Raspberry Pi 4. This script is responsible for mode switching and processing user input. Interaction with the Robot Hand is achieved through the Pi's touchscreen and buttons. In the real-time control modes, the script captures the coordinates of the hand's joints in the real world and employs an algorithm to calculate the bending angle of each finger and the rotational angle of the thumb based on these coordinates. In precision mode, users can manipulate the bending angle of the fingers by dragging the joints displayed on the screen, with the script dynamically calculating and reflecting the angles on the Robot Hand. Additionally, by pressing physical buttons, users can conveniently add desired actions to the queue for pre-programmed movements, enhancing the versatility and ease of use of the Robot Hand.

Mechanical Design

In the construction of our Robot Hand, we employed a Prusa 3D printer to create the fingers and arm from open-source *.*stl* files, which provided a customizable and cost-effective basis for our design. To assemble the fingers, we used elastic cords to link the three joints of each finger together and attach them to the palm. The elasticity of the cords ensures that the fingers maintain a straight posture in the absence of external forces.

For the actuation mechanism, we opted for a high-strength fishing line with a maximum load capacity of 10 pounds to control the bending of the fingers. This fishing line runs along the inside of each finger, one end attached to the fingertip while the other connects to a spool mounted on a servo motor. As the motor rotates, the line winds around the spool, pulling the

fingertip and causing the finger to bend. The degree of motor rotation determines the bending angle of the fingers, forming the core of our control algorithm. This approach effectively transforms the challenge of finger manipulation into a problem of motor control, allowing for precise and responsive movements of the Robot Hand.

As shown in Figure 1, our Robot Hand incorporates a total of six servo motors, where five of these motors (Servo Motor 1-5) are used to control the bending of the fingers. The remaining motor (Servo Motor 0) is specifically designed to control the rotation of the thumb's joint, an essential feature for executing complex tasks such as grasping objects. All six motors are housed within the forearm's casing. To ensure efficient operation without interference, the forearm's internal structure includes three square columns. These columns are strategically placed to secure the motors in distinct positions, preventing any overlap or entanglement of the fishing line tendons that drive the finger movements.

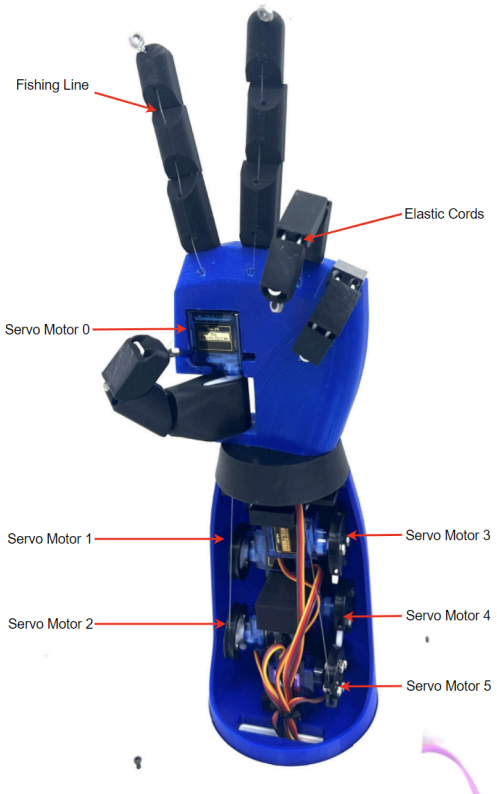


Figure 1. Assembly diagram of the robotic hand showcasing its constituent parts and their interconnections.

Circuit Design

As shown in the figure 2, a Raspberry Pi is used as the microcontroller, connected to a piTFT screen to display the user interface, six SG90 servo motors, and a Pi Camera. Each SG90 servo motor has a 5V power supply pin, a ground pin, and a GPIO pin used for the PWM input to control the motor's angle. And the Pi Camera is connected to the Raspberry Pi via a flex cable.

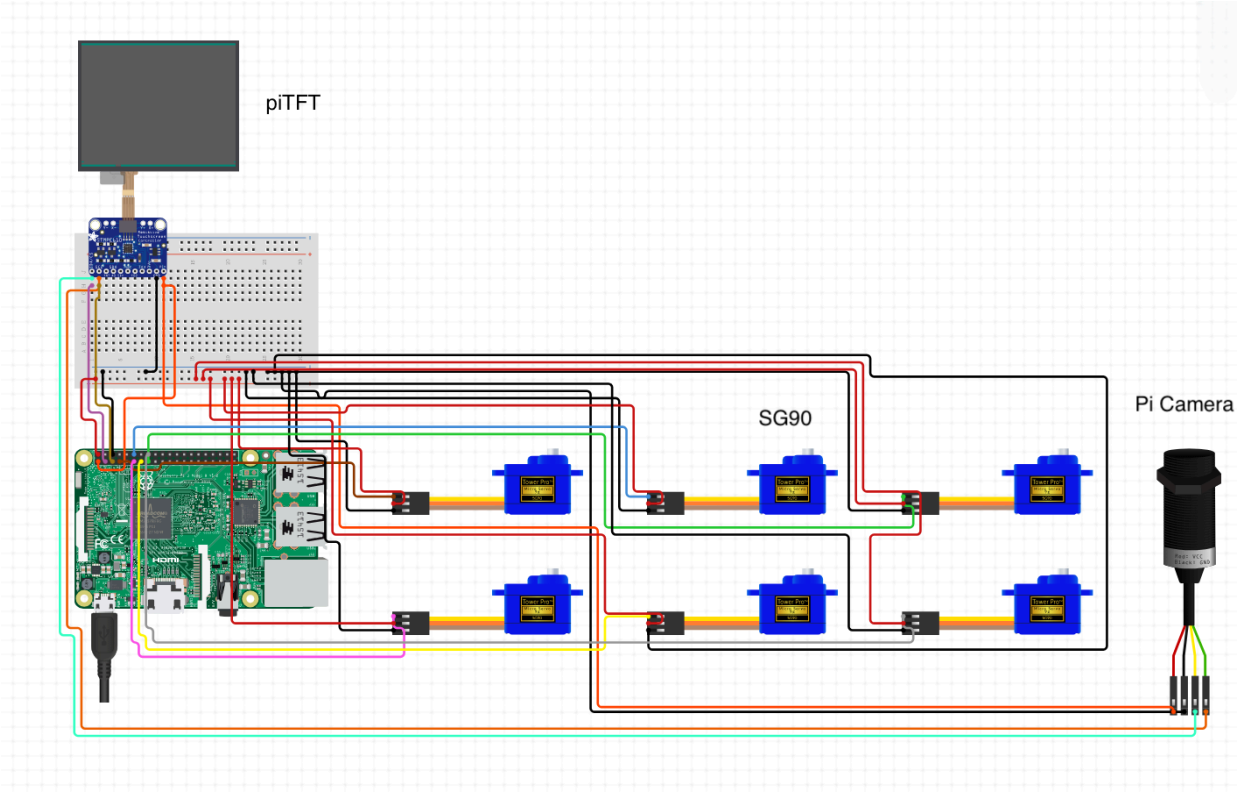
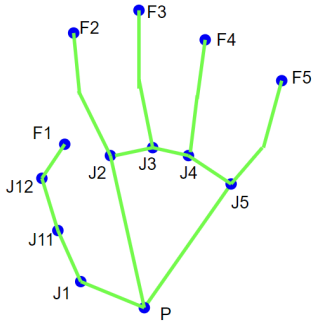
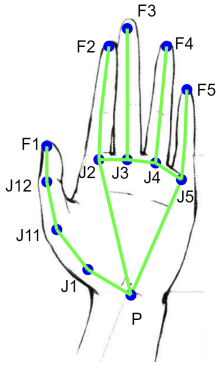


Figure 2. Circuit diagram of the robotic hand

Algorithm Design



a) Schematic representation of the robotic hand with labeled key joints



b) Corresponding real hand with labeled joints mirroring the schematic representation

Figure 3. Comparison of schematic and real hand images

The library MediaPipe we utilize can provide the three-dimensional coordinate points of our hand, as depicted in Figure 3. To translate these coordinates into finger angles, it is essential to align the points as illustrated in the left-hand Figure 3.a. This alignment process ensures accurate mapping between the detected hand landmarks and the corresponding joints of the Robot Hand, enabling precise control and manipulation of its movements.

To begin with, we categorize the detected points into three distinct groups, as illustrated in Figure 3.a: finger-tips (points F1-F5), finger-joints (points J1-J5 and J11, J12), and the palm-point (point P). Correspondingly, these points align with the designated points in Figure 3.b: finger-tips correspond to point F, finger-joints correspond to point J, and the palm-point corresponds to point P.

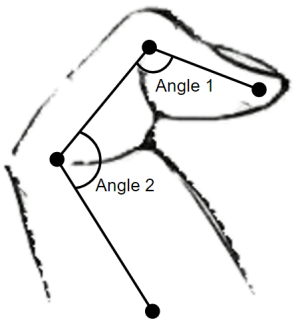


Figure 4. Thumb movement illustration with labeled angle.

For the thumb (finger 1), the control methodology is relatively straightforward. The two angles, as depicted in Figure 4, controlling the thumb's motion can be computed using the following formula:

$$\text{Angle1} = \arccos\left(\frac{\overrightarrow{J12F1} \cdot \overrightarrow{J12J11}}{\left|\overrightarrow{J12F1}\right| \cdot \left|\overrightarrow{J12J11}\right|}\right)$$

$$\text{Angle2} = \arccos\left(\frac{\overrightarrow{J11J1} \cdot \overrightarrow{J11J12}}{\left|\overrightarrow{J11J1}\right| \cdot \left|\overrightarrow{J11J12}\right|}\right)$$

Subsequently, these calculated angles can be mapped directly to the Pulse Width Modulation (PWM) signals of the servo motors 0 and 1, respectively. This mapping process is linear and therefore straightforward to implement, ensuring seamless and precise control of the thumb's movements.

For the remaining four fingers (2-4), we compute the lengths of FJ, JP, and FP for each finger by calculating the Euclidean distance between the aforementioned points' coordinates. Specifically, for finger 2, for instance, we determine FJ as the distance between F2 and J2, JP as the distance between J2 and P, and FP as the distance between F2 and P.

In reality, when bending a finger without rotating the hand, the length JP remains constant while the lengths FJ and FP change. To capture this behavior, we introduce the concept of a bend ratio, calculated as:

$$BendRatio = FJ / JP$$

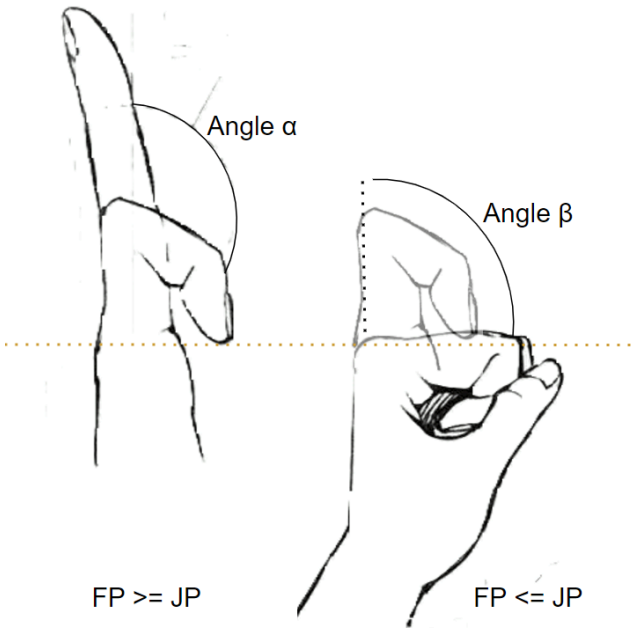


Figure 5. Fingers bending illustration with labeled angle.

However, the relationship between the bend ratio and the rotation degree of the servo motor is not linearly dependent. For example, when FP = JP, the bend ratio is 0 while the rotation degree of the servo motor should be around 90 degrees. Therefore, we divide our coordinate-to-angle function into two linear parts based on the region of the finger tips. As illustrated in Figure 5 above, in the upper region where FP > JP, the degree is calculated as:

$$degree = degree^{*} * (1 - (BendRatio / BendRatio^{up}_{max}))$$

where the $degree^{*}$ is the rotation degree of the servo motor when FP = JP, $BendRatio^{up}_{max}$ is the $BendRatio$ when the finger is straight. These two parameters must be predefined through experimentation and may vary for different fingers.

Similarly, when the finger tip is in the lower region, the rotation degree of the servo motor can be calculated using:

$$degree = degree^{*} + (degree*_{max} - degree^{*}) * (BendRatio / BendRatio^{down}_{max})$$

where the $degree*_{max}$ is the maximum rotation degree of the servo motor, $BendRatio^{down}_{max}$ is the $BendRatio$ when the finger curls the tightest.

After studying human finger movement patterns, we've observed that when we bend our fingers, we typically initiate the bend at the second joint, and subsequently, when the angle at the second joint reaches approximately 90 degrees, we begin to bend the third joint. Consequently, an additional advantage of our control algorithm emerges: in the initial stage, when FP >= JP, the bend ratio linearly dictates the second joint's angle α as depicted in Figure 5. Similarly, in the subsequent stage, when FP <= JP, the bend ratio influences the third joint's angle β. This allows us to separately control the angles α and β of two different joints by simply using the bend ratio in corresponding conditions, streamlining the control process and enhancing the intuitiveness of the Robot Hand's movements.

Software Design

Video Mode

Figure 6. Robotic hand operating in video mode

In the Robot Hand's Video Mode, we have deployed the Google *Mediapipe*'s Hand Landmarks Detection model to a Raspberry Pi. After capturing the real-time video data stream using the PiCamera, we processed this data stream using the tool to obtain the coordinates of the 21 hand landmarks as shown in Figure 3. Utilizing the algorithm introduced above, we transformed these joint coordinates into finger bending angles, which we then transmitted to the servo motors, thus controlling the hand's movements.

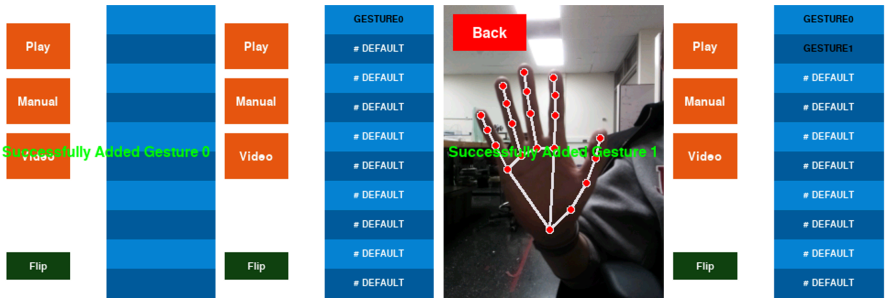
It is noteworthy that the transmission of joint data utilizes Flask, enabling network transmission. This means that if the camera and *Mediapipe* model are deployed on other devices such as smartphones or laptops, the robot hand can still be remotely controlled.

Precision Mode

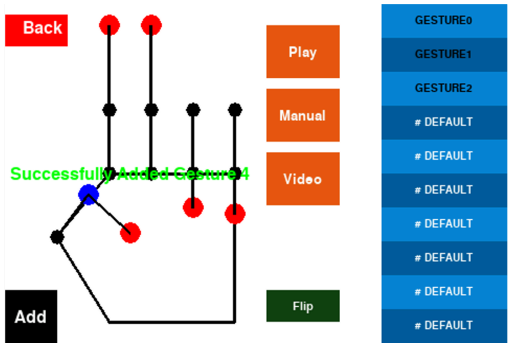
Figure 7. Robotic hand operating in precision mode

In the Robot Hand's Precision Mode, we've implemented a hand skeletal diagram using Python's Pygame on the PiTFT display as shown in Figure 7. The fingertips of the five fingers and the root joint of the thumb have been designed to be draggable, allowing users to precisely and customly configure the actions they want the robot hand to perform. Using the bending and tilting algorithm we designed, we've determined the angles for finger bending and joint rotation, and these data points are transmitted to the servo motors, enabling precise control of the robot hand's actions.

Main Page and Pre-programming Mode



(a) Add gesture from main page (b) Add gesture from video mode



(c) Add gesture by dragging joints

Figure 8. Robotic hand operating in pre-programming mode on main page

Once the control program for the Robot Hand is initiated, it will enter the main page interface. From here, users have the option to access different modes by interacting with the touchscreen interface. By tapping the "manual" button displayed on the screen, users can seamlessly transition into precision mode. Alternatively, users can opt to enter video mode by tapping the corresponding "video" button on the touchscreen interface.

In the Robot Hand's Pre-programming Mode, we've integrated physical buttons on the PiTFT display. Users can easily add desired actions to the action storage sequence as shown in Figure 8.a, 8.b and 8.c by pressing these buttons. We've developed an action sequence storage stack on the PiTFT display using Pygame, enabling users to add, view, and delete actions from the list with the physical buttons as shown in Figure 7. The gesture stack that we designed has the capacity to store up to 20 different actions, with each page capable of displaying 10 gestures at a time. Users can easily navigate through the stored actions by interacting with the touchscreen interface. By tapping the "flip" button displayed on the screen, users can switch between viewing the first 10 stored gestures and the subsequent 10 gestures.

Once the user has edited and stored the desired action sequence, they can select a stored gesture and click the “play” button on touch screen as shown in Figure 8 to execute independently, or they can press the phycial "play all" button as shown in Figure 7 to make the robot hand perform the actions stored in the sequence in order.

Result

We are extremely satisfied with the outcome of our Robot Hand project. We successfully developed a robotic arm capable of replicating human hand movements with precision and flexibility. The Robot Hand is able to operate through cameras and touch screens, and it offers several modes of operation and user interaction. The project has been a great success in providing a solution for tasks requiring precise, repetitive, or dangerous movements, reducing the risk of injuries for human operators. The mechanical design of the Robot Hand is robust and visually appealing, achieving a perfect balance between aesthetics and functionality. It was designed to provide both reliability and versatility, allowing for various applications in different industries.

Despite the successes achieved with the Robot Hand, there are several areas we would like to address in future iterations. The primary concern involves the delay in hand movements caused by the limited computational power of the Raspberry Pi. The Raspberry Pi's processing capabilities are currently limiting the Robot Hand's ability to execute hand movements with the desired precision and real-time responsiveness. This can result in a delay of approximately 1 second in the execution of hand movements, which impacts the overall user experience.

To address this issue, we will work on using RPi performance tools (like perf) to improve code performance, and investigate the use of multiple RPi cores to speed slow processing paths. Additionally, we plan to explore alternative hardware solutions that can provide greater

processing power, such as using a more powerful microcontroller or integrating additional processing units. Improving embedded performance can also be particularly useful for applications deployed in remote areas with limited or unreliable network connectivity, such as in agriculture. By enabling autonomous operation without relying on network or cloud access, enhanced embedded performance can significantly improve the reliability, efficiency, and overall user experience of these systems.

In addition to the aforementioned improvements, we also plan to enhance the user interface of the Robot Hand by developing a dedicated mobile or desktop application. This application will allow users to connect to the Raspberry Pi and control the Robot Hand remotely in a more intuitive and user-friendly manner, without the need for SSH access. By providing an interface with a website whose server is hosted by Raspberry Pi, we aim to make the device more accessible to a broader audience and enhance the overall user experience.

Moreover, we also plan to explore the possibility of offloading the model deployment and processing tasks to more powerful computing devices, such as a PC or cloud-based server. This will help reduce the computational load on the Raspberry Pi, allowing for faster and more responsive hand movements. By leveraging external computing resources, we aim to further enhance the performance and capabilities of the Robot Hand.

References

1. 3D printed robot hand: https: //www.instructables.com/3D-Printed-Robotic-Hand/
2. Mediapipe: https://developers.google.com/mediapipe/solutions/vision/hand_landmarker

Parts

Raspberry Pi: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/

Pi camera: https://www.raspberrypi.com/products/camera-module-v2/

Servo

SG90 digital servo: https://www.towerpro.com.tw/product/sg90-7/