CMPUT 312 Final Project

Robotic Hand - The GitHub Repository

Appendix

I. Abstract

The Robotic Hand that was built can successfully and repeatedly perform tasks such as making specific hand positions like a fist, the rock on symbol, peace symbol and the middle finger, which was a comedic hit, it can also count from one to five. The more advanced task is the hand imitation task which reads the angles of the fingers and their joints from a human hand through a visual recognition program, the server then sends these angles to the Raspberry Pi which sends the angles to the servos and thus imitates a human hand. This task made everyone who tried it out smile and laugh in joy that they were able to get the robot to follow the movements of their hand, some of these individuals also thought it was very creepy. The beginning of the project was very ambitious and had many oversights and despite the time frame to complete the project, there was a very successful outcome.

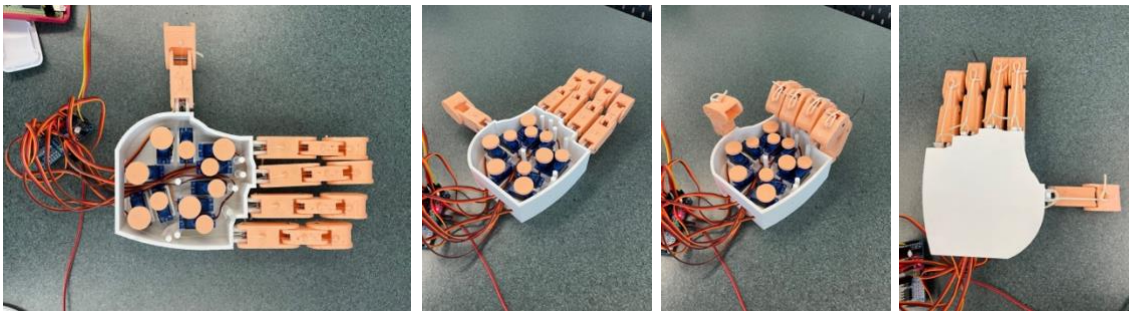
II. Motivation

This project was inspired by the incredible complexity of a real hand, and the thousands of years of evolution to design a hand that allows humans to perform very complex tasks. There was also motivation from current research into robotic hands, specifically from a company called shadow robotics. The hand that they produce was used by Google, Open AI and other research firms to produce multiple case studies. I also drew motivation from my current hands, which, because of many concussions, shake quite badly and sometimes to the extent where I cannot eat, I thought it would be amazing to explore the possibilities of creating a hand and exploring the mechanisms and technologies behind it.

III. Design

The design of the hardware was quite tedious, many iterations were made and scrapped as the design was evaluated. The design is completely authentic, and nothing was taken from the internet besides the 3D model for the servo I used in the hand. The images below showcase the assembled hand including the extended pose and the fist pose. The entire hand was roughly modelled after the size and shape of my real hand. The only exception to this was making sure the palm was big enough to fit all 10 of the SG90 Micro Servos inside it. The design of the

project is split up into two major sections; the hardware and the software which are discussed for in depth below.

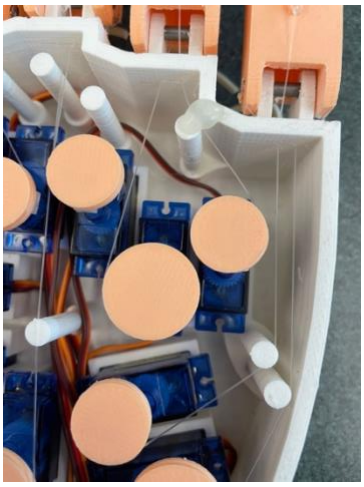


IV. Hardware

To preface this section, the complete list of hardware parts includes 1 palm section, 4 finger bases, 4 finger mid joints, 4 fingertips, 1 thumb base, 1 thumb tip, 5 small line spools, 5 large line spools, 10 SG90 Micro Servos, fishing line, 0.125” steel rod, 5 rubbed bands, 1 Sun founder PCA 9685 16 channel servo controller, 1 Raspberry Pi Zero, 1 camera and some DuPont wires.

The finger base and tip holes had 0 tolerance for the steel rod connections, the joints depend on the friction between the rod and finger base/tips to stay secure, the palm connections and the finger mid joints both were drilled out to allow a frictionless connection between themselves and the rod. This allowed for fluid finger movement while maintaining a simple method of joint connection.

Each finger base along with the thumb base and tip are controlled by one servo which has a small line spool superglued to the top. The maximum rotation for this servo is 120 degrees, then the base will have rotated about its joint by the maximum of 90 degrees. The finger mid joints and tips are both controlled by a single servo jointly as for most of the population, when you bend the tip of your finger the mid joint also goes with it. It should be mentioned that these servos have a maximum rotation of 180 degrees, and despite the larger spool size for these servos, there still wasn’t enough room to allow full contraction of the finger mid joint and tips. It should also be mentioned that the fingertip will completely rotate about its connection before the mid joint will which is a minor shortcoming of this design. Each finger and the thumb have an elastic band on the back side which provides the resistance force, this way when the servo is driven backwards effectively unwinding the spool, the finger will retract into the straight position. The different spool sizes are showcased in the image to the right, these spools control the ring finger movement.

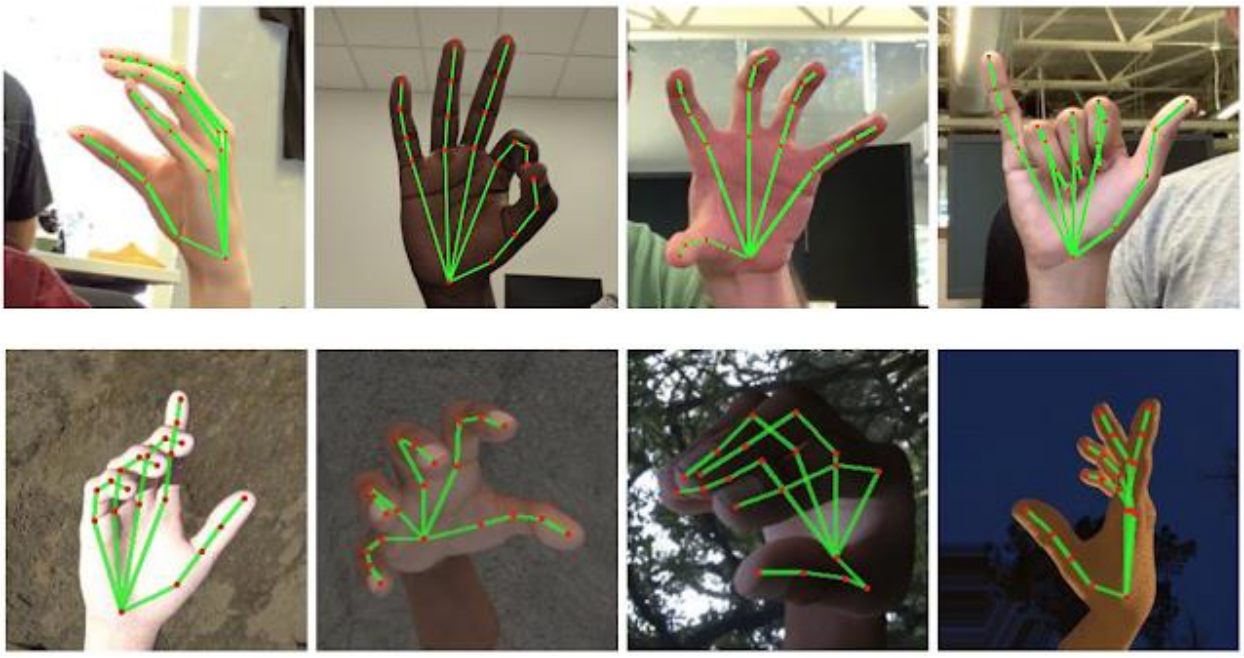


The raspberry pi functionality was designed in two different settings. Firstly, the university Wi-Fi blocks the MAC address of all RBPi’s so that eliminated the ability to communicate with it wirelessly. The raspberry pi was booted in gadget mode which allows the

hardwired connection between a laptop, in this case a MacBook Pro, this method was also tested with Windows and Linux operating systems and unfortunately did not work, however, this may have been due to the machines being used for testing not being compatible. The second setting was over any other wireless internet connection which allowed the RBPi to connect. All that must be provided in both cases is the iPv4 of the host machine and on the network.

V. Software

The heart of the software begins at the RBPi, the main program for the control system of the hand is executed on this device. The entire project was developed using Python, and various external python libraries mentioned in the GitHub repository. Some rather important mentions are the use of OpenCV and Google's mediapipe library which was used for the hand detection machine learning algorithm. The images below shows the mediapipe library in use.



The servos are controlled by a python library from Adafruit, the control is relatively simple for this project, a simple assignment with the desired angle achieves the goal. For the imitation task, all that was needed was the calculation of the angle from the finger coordinates given by mediapipe. The calculations are displayed in the figure to the right. Mediapipe returns the coordinates for different joint locations on the hand, so two 3 dimensional vectors need to be calculated and then an angle between those vectors can be calculated using the dot product and norms. Once the angles were obtained, they were scaled appropriately depending on the type of joint they were manipulating. In this case the calculation was for one of the finger bases. Then the angles are sent over the socket connection to the RBPi which then sent the angles to the servos

```
vec1 = [
  coords[fingerBase[1]].x - coords[fingerBase[0]].x,
  coords[fingerBase[1]].y - coords[fingerBase[0]].y,
  coords[fingerBase[1]].z - coords[fingerBase[0]].z,
]
vec2 = [
  coords[fingerBase[2]].x - coords[fingerBase[1]].x,
  coords[fingerBase[2]].y - coords[fingerBase[1]].y,
  coords[fingerBase[2]].z - coords[fingerBase[1]].z,
]
numer = np.dot(vec1, vec2)
denom = np.linalg.norm(vec1) * np.linalg.norm(vec2)
angle = math.acos(numer / denom) * (180 / math.pi)
```

The Socket communication was established by setting up the server on the host machine and the client on the RBPi, once all the necessary angles were calculated from the mediapipe image and then scaled appropriately, they were sent back-to-back in the order of thumb, pointer, middle, ring and pinky fingers. Then the client would sleep for 0.1 seconds in order to move the hand and then allow itself to receive a new set of angles again. This can be seen in the figures below with the client on the left and the server on the right.

```
thumb = client.pollData()
client.sendDone()
pointer = client.pollData()
client.sendDone()
middle = client.pollData()
client.sendDone()
ring = client.pollData()
client.sendDone()
pinky = client.pollData()
client.sendDone()
thumb = ast.literal_eval(thumb)
pointer = ast.literal_eval(pointer)
middle = ast.literal_eval(middle)
ring = ast.literal_eval(ring)
pinky = ast.literal_eval(pinky)

self.setHandAngles(thumb, pointer, middle, ring,
pinky)sleep(0.1)
```

```
self.cs.send(thumb.encode("UTF-8"))
# Waiting for the client to send a response to the server
reply = self.cs.recv(128).decode("UTF-8")
queue.put(reply)
self.cs.send(pointer.encode("UTF-8"))
# Waiting for the client to send a response to the server
reply = self.cs.recv(128).decode("UTF-8")
queue.put(reply)
self.cs.send(middle.encode("UTF-8"))
# Waiting for the client to send a response to the server
reply = self.cs.recv(128).decode("UTF-8")
queue.put(reply)
self.cs.send(ring.encode("UTF-8"))
# Waiting for the client to send a response to the server
reply = self.cs.recv(128).decode("UTF-8")
queue.put(reply)
self.cs.send(pinky.encode("UTF-8"))
# Waiting for the client to send a response to the server
reply = self.cs.recv(128).decode("UTF-8")
queue.put(reply)
```

Now that the servo control, angle calculations and the socket communication have been outlined, the imitation task is a neat combination of the three. One issue that was encountered and not solved was that, depending on the orientation of the human hand in the camera, the calculation of the angle of the fingers was different. One attempted solution was to add a constant to the calculated angle in order to adjust it, but because this error varies based on orientation, the constant amplified the error, and the robotic hand modelled the real hand even worse in the imitation task.

VI. Future Work

One idea for future work stems from the extra unused slots on the servo controller, there is another 6 slots for servos. These extra servos could allow for a greater degree of freedom to allow more complex movement or even to give each finger 3 degrees of freedom instead of only 2.

Continuing from the previous idea for the future, the extra servos would allow for the potential connection to an arm and having wrist movement to both rotate the hand and tilt the hand. Supporting the hand with an arm could potentially remove the need to have servos in the hand, this would allow for a cleaner, thinner, and more realistic design for the hand while maintaining the same functionality.

Another option instead of allowing for wrist movements would be creating a tilt for each finger with the extra servo slots, this way the hand could effectively cross its fingers, doing this

would allow the hand to create and display the American Sign Language alphabet, allowing it to spell out words and names. It would be interesting to see a robot perform the functions of ASL, furthermore, this could be applied to machine learning as well, taking in a live stream of words, and signing them in translation.

References

Dexterous Robotic Hands & Teleoperated Robots. Shadow Robot. (2022, October 23). Retrieved November 6, 2022, from https://www.shadowrobot.com/

Home. mediapipe. (n.d.). Retrieved November 6, 2022, from https://google.github.io/mediapipe/

Home. Robot Nano Hand - open source robot hand project. (2022, February 4). Retrieved November 6, 2022, from https://robotnanohand.com/

Parloma robotic hand builds and upgrades. (2017, May 30). Retrieved November 6, 2022, from https://parloma.github.io/news/robotic%20hand/2017/05/30/parloma-robotic-hand-builds-and-upgrades.html