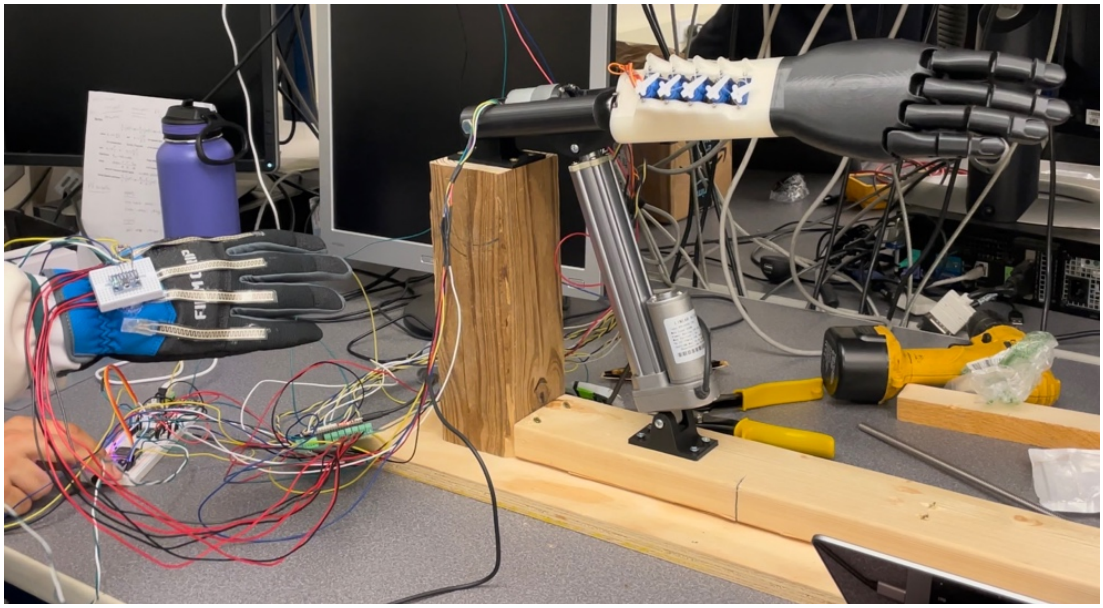
ME 507 Term Project
Robot Arm



Source Code: https://github.com/cagena/RobotArm.git
Documentation: https://cagena.github.io/RobotArm/

By
Corey Agena, Daniel Ceja, and Parker Tenney
December 5, 2022

Table of Contents

List of Figures

Page

List of Tables

Page

Introduction

This is the ME507 Project report for the Robotic Arm. This term project was assigned by Professor Ridgley with the goal of developing a custom device alongside a custom electrical design in order to improve the team members skills in the area of Mechatronics. For our specific project we were very motivated to pursue the Robot Arm as we though there was a wide number of applications for our future design and that it would challenge us with the shear complexity of the project scope. At the beginning it was a bit daunting looking into the amount of hardware, electrical, and software design necessary but after diving it and completing it all it was very rewarding. Upon initial inspection some of the potential customers could be companies that work with items dangerous for humans to touch such as chemicals. This would allow for the human to step back from the dangerous item, yet still be able to work for it. Similarly, the glove that we developed could be used to control our Robotic arm but could also be configured for any other computer-controlled system. Giving the user control of something with their hand is very intuitive and gives them a sort of human connection to the mechanical system. The arm itself could be used as a prosthetic arm where the user could control it with something other than their hand, maybe with movement of various muscles of what is left of their arm. As a team we thoroughly enjoyed this project, and this has encouraged us to pursue further iterations of our design for possibility of commercial use in the future. All stages of the design process as well as the results of our design are outlined in the full report below.

Specifications

The specifications for the project can be seen below in Table 1. These specifications drove the design process for the project and facilitated the many levels or iterations required

Specifications	Criteria	Result
Wrist Motion Minimum	180 deg	Pass
Arm Up and Down Motion Minimum	90 deg	Pass
Finger Motion Minimum	45 deg	Pass
Arm Weight Maximum	20 lbs	Pass
Number of fingers Controlled	5	Pass
Maximum Price	$250	Pass

Table 1. Specifications list

As you can see above all of these were specifications, we set prior to starting the project, and the final result passed every single one.

Design Development

The design process was broken down into three sections: mechanical, electrical, and software design. Each design will be described in this section of the report.

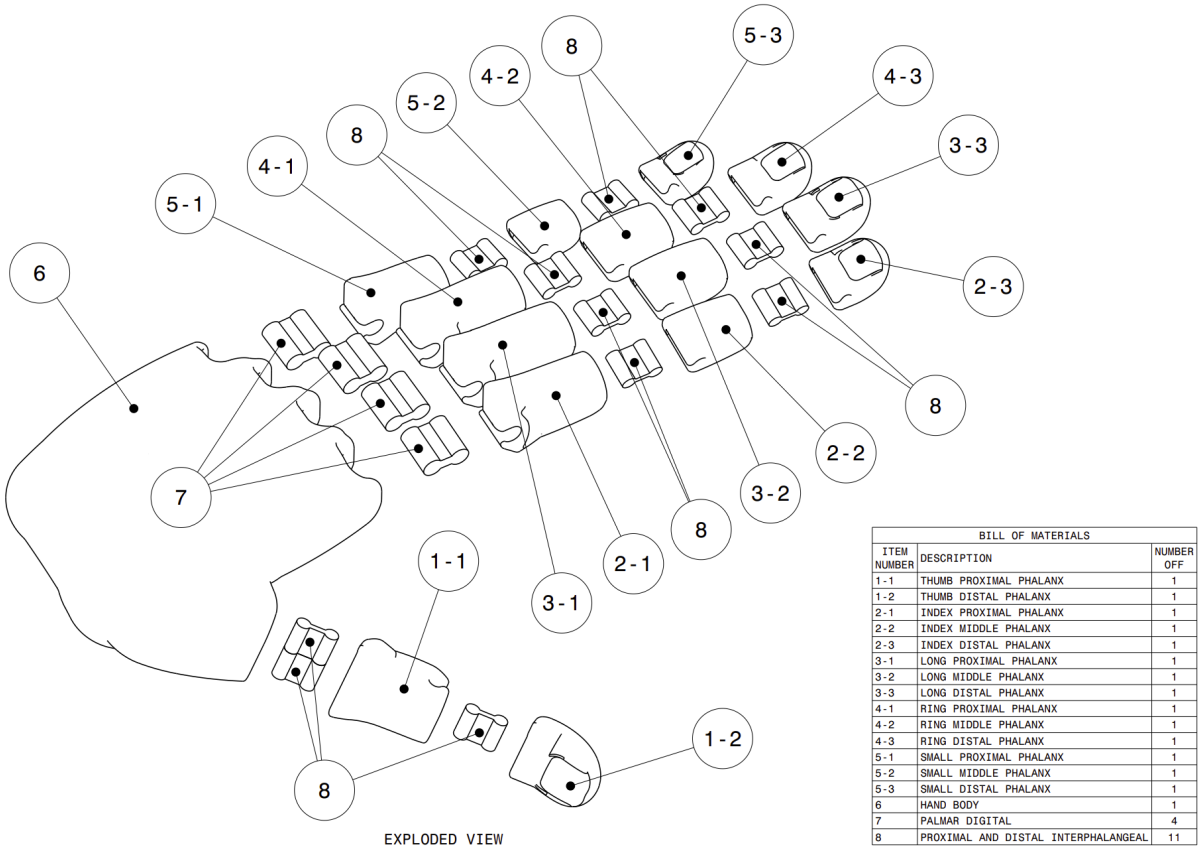
Hardware Design

The hardware design for this project was relatively large in scope. The shear number of the components was manageable, but they were very intricate designs on their own. The details of each piece of hardware will be discussed throughout this section.

The most crucial component to the system is the hand itself. It is very important to not only have a hand that looks real but also ensure that it functions properly and interfaces with the rest of our system well. Research was completed into trying to find a hand that was compatible with these goals and after looking at many different options we settled with the one seen in Figure 1 below.

Figure 1. Hand CAD File

This hand was then taken into SolidWorks and modified to have 7 mounting holes on the bottom for screws to attach it to the upper forearm assembly. The hand works by running fishing string through the holes of the finger, through the hand, and out the bottom. The string is then pulled and the finger closes, and when the string is let go the hand then retracts to its original position. The whole hand assembly in detail can be seen in Figure 2 below.



BILL OF MATERIALS

ITEM NUMBER	DESCRIPTION	NUMBER OFF
1-1	THUMB PROXIMAL PHALANX	1
1-2	THUMB DISTAL PHALANX	1
2-1	INDEX PROXIMAL PHALANX	1
2-2	INDEX MIDDLE PHALANX	1
2-3	INDEX DISTAL PHALANX	1
3-1	LONG PROXIMAL PHALANX	1
3-2	LONG MIDDLE PHALANX	1
3-3	LONG DISTAL PHALANX	1
4-1	RING PROXIMAL PHALANX	1
4-2	RING MIDDLE PHALANX	1
4-3	RING DISTAL PHALANX	1
5-1	SMALL PROXIMAL PHALANX	1
5-2	SMALL MIDDLE PHALANX	1
5-3	SMALL DISTAL PHALANX	1
6	HAND BODY	1
7	PALMAR DIGITAL	4
8	PROXIMAL AND DISTAL INTERPHALANGEAL	11

Figure 2. 3D Printed Hand Assembly

Parts 1 through 6 were all made on an Ender 3 Pro 3D printer using PLA filament as these structures are meant to be rigid. Parts 7 and 8 are made with a flexible TPU filament as these are what allow the fingers to bend and bounce back to their original position. This hand assembly alone had a total of 30 individually 3D printed parts.

This hand was then connected to the upper arm assembly which holds all the servos for controlling the fingers. The Cad Model for this upper arm assembly can be seen in Figure 3 below.

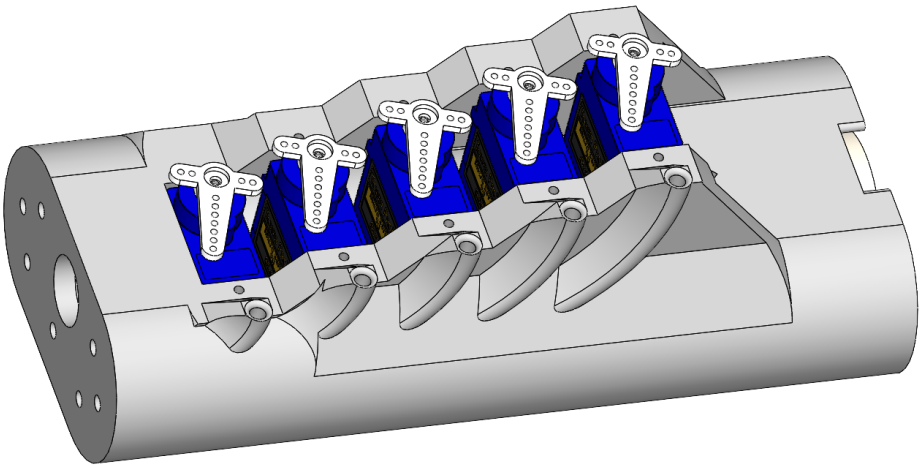


Figure 3. 3D Printed Upper Arm Assembly with Servos

In this figure you can see the large 3D printed part in the off-white color with the servos in blue. There are a total of 5 servos, one for each of the fingers. The hand would be attacked to this assembly on the left side with the screws mounting it through those small holes. The large hole on the left side is to feed the fishing string through for the fingers. This fishing string then runs by the servos and through those five curved pipes running vertically on the pipe. The reason these are curved is to ensure that the fishing string doesn't face too much resistance when moving around a sharp corner, thus the curves ensure a smooth transition to the servos. The servos then take this fishing string and attach to the end of those bright white T connectors on the servo. These then spin in the clockwise orientation to tension the string, thus closing the finger. Each servo is on a different vertical plane to ensure that when they rotate, they do not hit each other. This design went through over 5 different redesigns to ensure that it was a compact and efficient as possible. This assembly also had to spin on order to simulate the rotating of one's arm. This was done by rigidly attaching the upper arm assembly to a DC motor shaft located on the lower arm assembly. This DC motor that was selected also had an encoder attached to it to allow for control of the wrist motion. This can be seen in Figure 4 below.

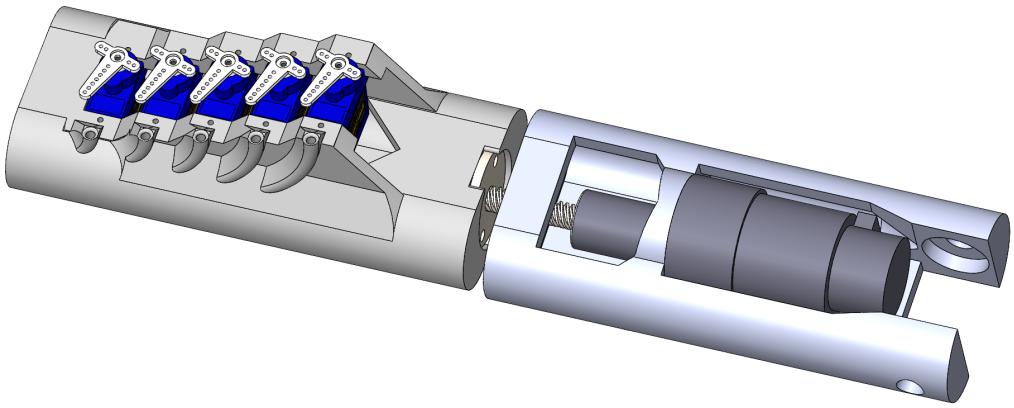


Figure 4. 3D Printed Upper Arm and Lower Arm Assembly

The motor above is pictured in black and is rigidly attached to the lower arm 3D print seen in blue gray on the right. The spinning shaft of this motor is then rigidly attached to the lead screw with a collar that then rigidly attached to the upper arm assembly with another collar that is bolted to the upper arm assembly. On the right side of the lower arm 3D printed part you can see a hole. This hole allows for the arm to be mounted to the wood frame as seen in Figure 5 below.

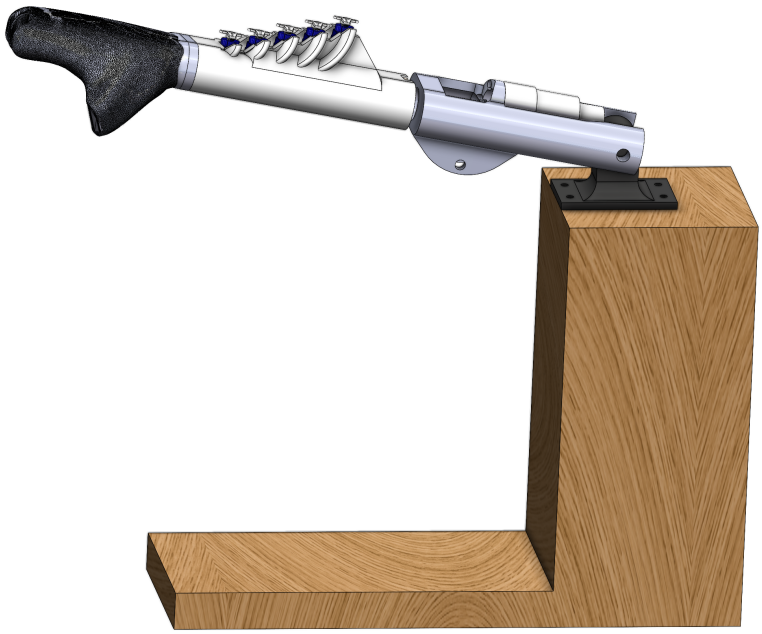


Figure 5. Full Arm Assembly Mounted to Frame

This arm is then mounted to the bracket on top of the wood frame on the right. This acts as a pin to enable the arm to move from horizontal to vertical. As you can see on the lower arm 3D print there is a curved extrusion with a hole on the bottom. This allows for the linear actuator to attach to the arm as seen in Figure 6.

Figure 6. Full Robot Arm Assembly in Horizontal Position

The linear actuator pictured in white is mounted to the wood frame with the black 3D printed mount seen on the bottom and allows the arm to move from its horizontal state above in Figure 6 to the vertical state seen below seen in Figure 7.

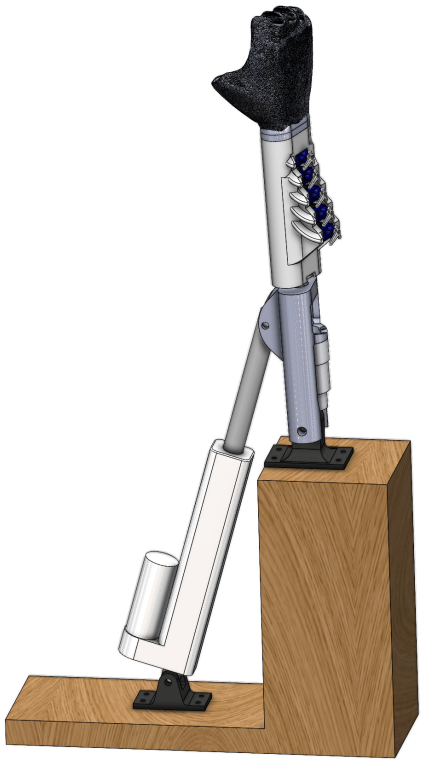


Figure 7. Full Robot Arm Assembly in Vertical Position

The full assembly as seen in both figures is incredibly complex and required a lot of hardware such as nuts and bolts. The servo mounting in particular was very tedious as they needed a total of 10 M2 nuts and bolts which were very small and difficult to work with. The upper arm assembly which includes all the servos, fishing string, hand mounting screws, and motor collar took approximated 4 hours to assemble. The final product of the mechanical design can be seen in Figure 8 below.

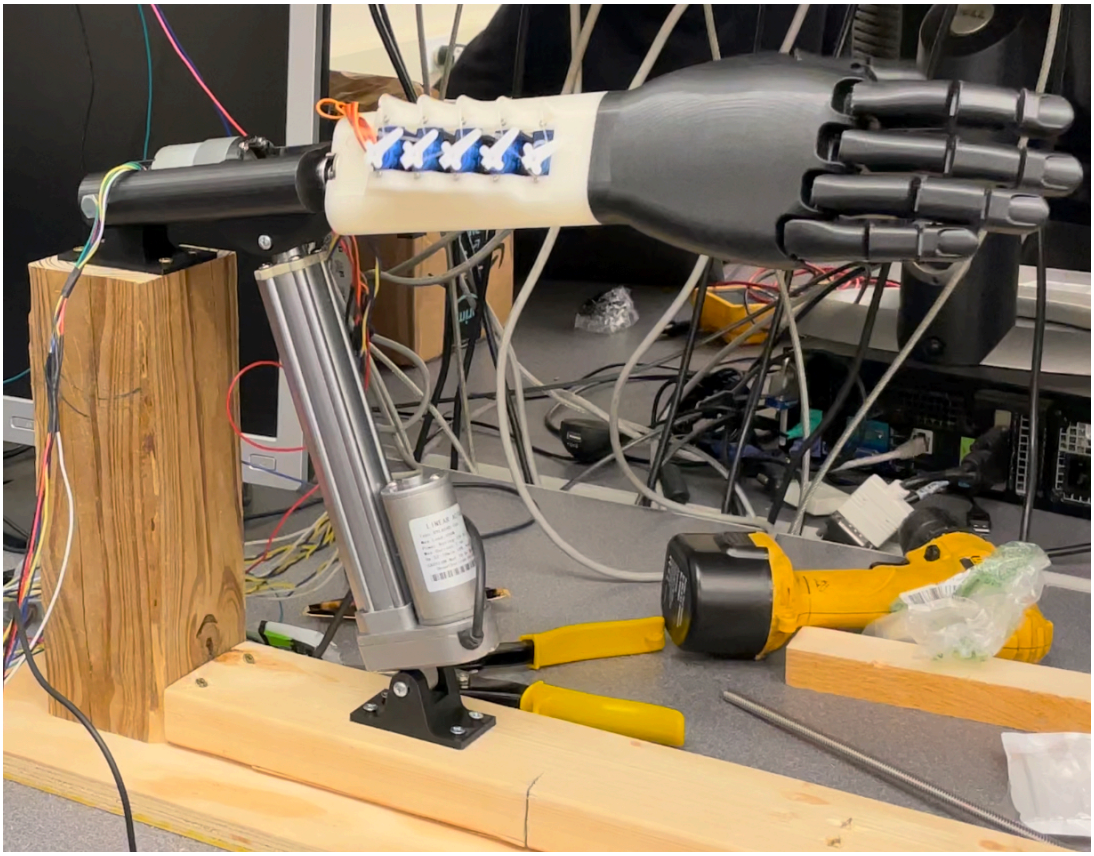


Figure 8. Final Robot Arm Assembly

This is the full mechanical system for the project and the team was very happy with how it turned out. When implementing the electrical and software design we ran into no issues with the hardware holding us back. Details on the rest of those systems are discusses in the sections below.

The main components of the system include a 3D-printed structure, custom printed circuit board (PCB), servo motors for finger control, DC motor for wrist control, and a linear actuator for arm control.

Electrical Design

The electrical design focuses on receiving readings from sensors on a glove then applying control to the robot arm. It was determined that the glove would use a flex sensor on each finger to determine finger movement and an inertial measurement unit (IMU) to determine wrist and arm movement. Then it would control servos to move the fingers, a DC motor for the wrist, and a linear actuator for the arm. The important electrical components are defined in Table 2 below.

Table 2. Bill of materials for major electrical components

Part No.	Part	Qty.	Source
1	Yosoo Flex Sensor 0-500g	5	Amazon
2	Servo Motors	5	On hand
3	Gyroscope - Hiletgo MPU 6050	1	Amazon
4	DC Motor	1	ME507 Lab
5	Linear Actuator	1	ME507 Lab
6	External Power Supplies (5V,12V)	2	ME507 Lab

A custom PCB was designed to improve the quality of electrical connections and to make the final product cleaner and more functional. It removes bread boards from the design which can have faulty connections and allows wires to unplug. The initial plan was to connect all components in the system to the microcontroller through the custom PCB. The PCB components fro the initial design are detailed in Table 3 below.

Table 3. Bill of materials for PCB

Part No.	Part	Qty.	Source
1	ESP32 Microcontroller	1	ME507 Lab
2	Custom PCB	1	JLCPCB
3	Level Shifter (TXS0108E)	1	SparkFun
4	TB6612FNG Dual Motor Driver	1	Pololu
5	Screw Terminal w/ 2 Pins 2.54 mm	11	Amazon
6	Screw Terminal w/ 3 Pins 2.54 mm	5	Amazon
7	Screw Terminal w/ 5 Pins 2.54 mm	1	Amazon
8	Surface capacitor 0.1 uF 805	1	Mouser
9	Surface capacitor 10 uF 805	1	Mouser
10	Ferrite Bead 805 package	1	Mouser
11	Surface resistor 2000 ohm 805	1	Mouser

The board includes the ESP32 microcontroller and two break out boards: a level shifter and motor driver. The PCB design implemented screw terminals for wired connections and utilized female headers to mount the breakout boards. The PCB was designed using eagle and is shown

below in Figure 9. Some highlights of the design are the include filtering, level shifting, and the voltage divider circuit. A ferrite bead was placed at the 3.3 V output of the ESP32 as the first part of a filter. The intention being to keep the voltage supply to each sensor steady. The second part of the filter is having a 0.1 uF and 10 uF capacitor in parallel to filter low and high frequency noise. The level shifter increases 3.3 V to 5 V so that the servo motors receive the expected voltage for the pwm signals. It is also important because it prevents over voltages from breaking the ESP32 microcontroller. Figure 9 shows the low- and high-level side of the level shifter with female headers. The low side is connected to the Vcc of 3.3 V and the high-level side is powered by an external power supply that also powers the servo motors. The last highlight is the voltage divider circuit, which is implemented for every flex sensor, which provided a voltage to ESP32 that corresponded with its resistance.

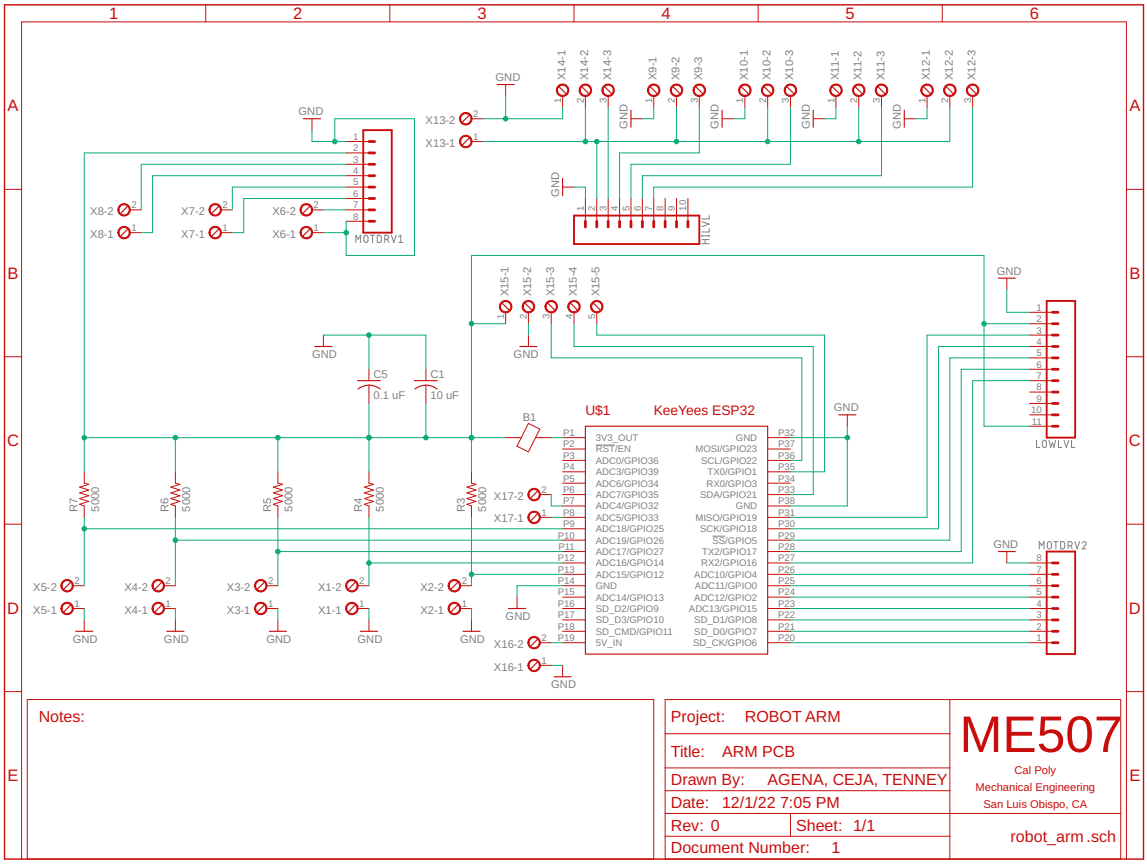


Figure 9. Eagle Schematic of the PCB

The schematic above was used to generate a board file where vias, and other aspects of the board were edited. The final PCB is shown below in Figure 10. Here you can see the considerations that went into designing the board:

- Screw terminals were kept on the outer edge of the board to ensure easy access
- Power traces were kept large by using trace widths of 0.050 in. and the Eagles polygon tool for the 12 V power to drive the motor and linear actuator
- Large heat sinks with vias were added to mitigate heat.
- Minimal traces were placed on the bottom of the board (occurs near the resistor pads)

- 805 package surface mount components were used to reduce size, but ensure workability
- Silk screen was utilized to show what goes where
- The board is split into sections starting from the bottom left and going clockwise (flex sensors, IMU, servo motors, and the DC motor and linear actuator – labeled motors)

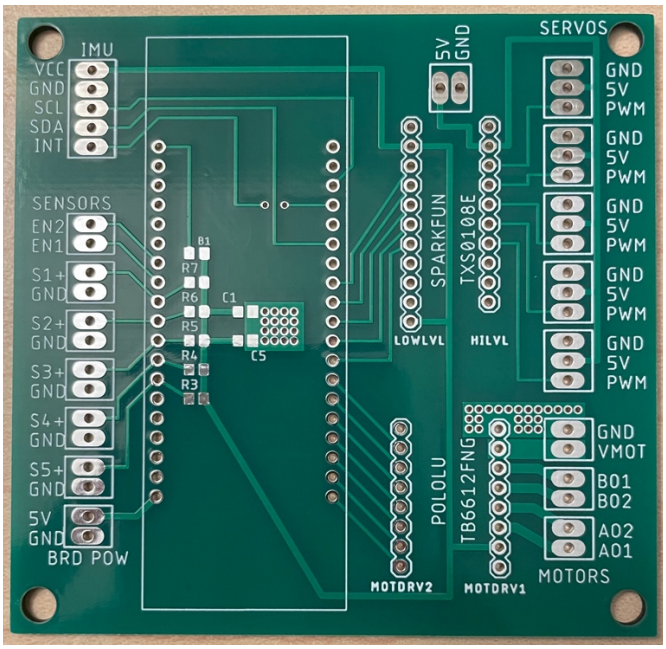


Figure 10. PCB ordered from JLCPCB

The final board, with all surface mounted components, is shown below in Figure 11. In the end the flex sensor section and servo motor sections were the only ones to operate on the board. The IMU, DC motor, and linear actuator was moved to a bread board and controlled by a second ESP32. The separation of the electrical system between flex sensors and IMU was necessary because the motor driver that is connected on the bottom right is attached to pins dedicated to the flash memory of the ESP32. Those pins would not work for control. The other issue being that there were not many other pins left on the ESP32 board to wire to instead. The empty pins on the left of the board with no traces in Figure 10 are either other flash memory pins or input pins. This means that the board design was flawed and if there was more time a new design would be made working around the flash memory pins and potentially placing sensors on pins that were input only. Another consideration is to only use the required pins on the level shifter, this would free up pins on the microcontroller. Finally, if another prototyping PCB was required, traces off the ESP32 to vias would be created to allow changes to the board with wires.

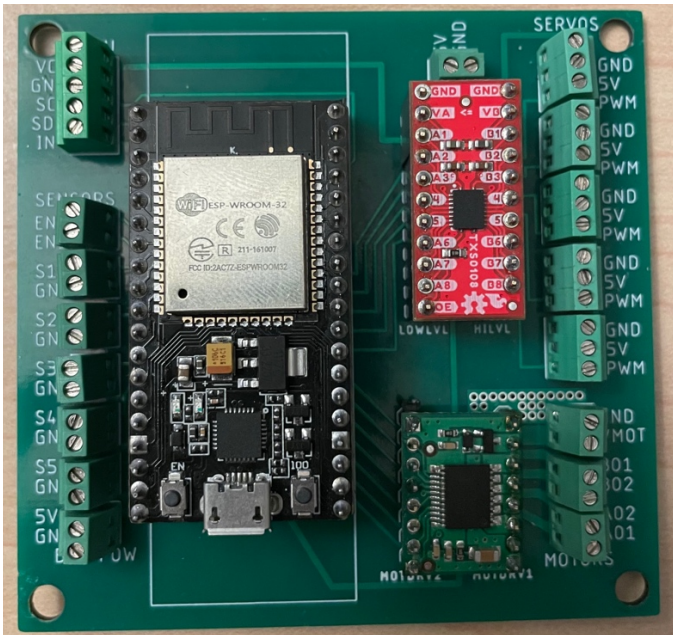


Figure 11. Completed PCB

The components were integrated with PCB smoothly, until the IMU became troublesome. The IMU printed multiple errors to the serial monitor, and the group believed the errors were due to software issues. The problem kept occurring and occasionally the IMU would work. But, by the next day or the next test it threw the same errors. Eventually, after consulting Dr. Ridgely, the team determined that the resistors on the IMU break out board could be the issue. Once a new 5.6 kΩ resistor was placed across the SDA and Vcc, and another across the SCL and Vcc the issue was resolved and the IMU did not have more issues. The final glove used in the project is shown if Figure 12. It shows the IMU and the implementation of the two resistors. It also shows the flex sensors, which are adhered with an adhesive spray.

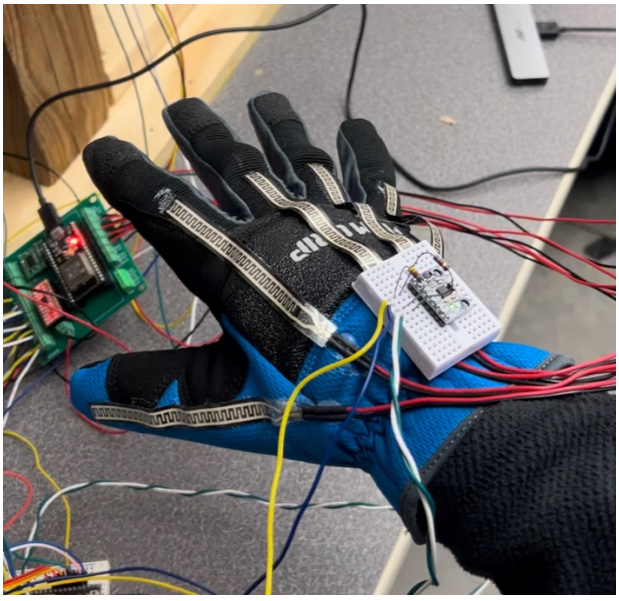


Figure 12. Final Glove Design

Toward the end of the project an issue was discovered with using WIFI on the ESP32. When using WIFI, ADC 2 cannot be used. This meant that GPIO pins 12, 13, 14, 25, 26, and 27 cannot be used while using WIFI. Since the flex sensors were implemented on these pins the webpage could not be implemented on the ESP32 dedicated to the flex sensors. Instead, it was implemented on the ESP32 dedicated to the IMU which did not use ADC.

Software Design

For this project the team decided developed two separate software interfaces, one that dealt with the flex sensors and servos controlling each finger and the other dealt with the IMU wrist and arm up and down control. These two interfaces are shown below, each with their own task diagrams. These two interfaces are each controlled by separate ESP32's. The reason behind this is that one ESP32 did not have enough usable pins to deal with both interfaces. As seen below in Figure 13 the tasks use a Shares and Queue structure to pass in variables between tasks.

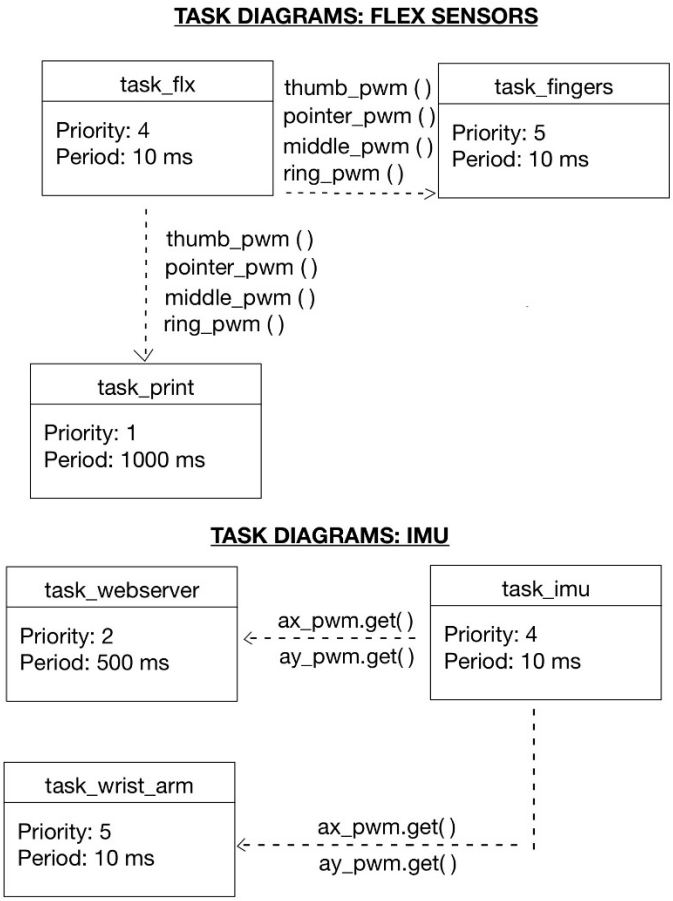


Figure 13. Task Diagrams

Flex Sensor Portion

For the flex sensor portion of the software interface, whose source code can be found under the RobotArm_Flex folder, several libraries apart from the standard Arduino were used that were not written by the team like the library ESP32Servo written by madhephataestus and the standard ME507 libraries which include the Shares and Queues files and PrintStream written by JR Ridgely (spluttflob). Using these libraries tasks were created so that the flex sensor shared a pwm signals to the servo motors and based on those individual pwm signals the hand would be closed or open. The flex sensors were put into a voltage divider which then allowed us to collect different readings as the resistance changed due to their flex position. The readings were gathered using the analogRead() function and then its range remapped to correspond to an acceptable signal for the servo motors. The remapped values were then placed in a shared variable. An example of this code is shown below in Figure 14.

```
while (true)
{
    // Read the thumb sensor and map it to a pwm value.
    thumb_flx = analogRead(THUMB_R);
    thumb_pwm.put(map(thumb_flx, 0, 4095, 180, 0));
```

Figure 14. Flex Sensor Reading

To control the servo motors the shared readings of the flex sensor were passed into an “on and off” state for the servos. An example of this code is shown below in Figure 15:

```
// A on and off state were created for each finger.
while (true)
{
    if (thumb_pwm.get() >= 50)
    {
        ThumbServo.write(0);
    }
    else
    {
        ThumbServo.write(180);
    }
```

Figure 15. Thumb Servo Code

Here we can see how thumb pwm signal is shared and if its value is greater than 50 the servo goes to position 0, else it goes to position 180.

IMU Portion

For the IMU portion of the software interface, whose source code can be found under the RobotArm_IMU, several libraries were used to control the robot arm and wrist using IMU readings. Other than the previously stated ME507 libraries, several libraries including: Adafruit_MPU6050 and Adafruit_Sensor libraries, sparkfun's SparkFun_TB6612 library, Wire.h, WiFi.h, and WebServer.h were used. Using these libraries tasks were created so that the IMU would share its position value which then determined the robot arms elevation and the position of the wrist. These IMU readings were handled in the main file using a task called task_imu which took IMU readings, changed the range using the map function and placed them in a share variable as shown below in Figure 16.

```
while (true)
{
    mpu.getEvent(&a, &g, &temp);
    ax = map(a.acceleration.x, -10, 10, 0, 180);
    ay = map(a.acceleration.y, -10, 10, 0, 180);
    ax_pwm.put(ax);
    ay_pwm.put(ay);
    vTaskDelay (10);
}
```

Figure 16. IMU Readings

To control the wrist, motion a simple P controller was implemented that the wrist would mirror the IMU's values. This P controller implementation is shown below in Figure 17.

```
while(true)
{
  float imu_current = -ax_pwm.get() +90;
  float encoder_current = encoder.getCount();
  float encoder_max = 450;
  float imu_max = 90;
  float gain_max = 255;
  float gain_motor = gain_max*((imu_current/imu_max)-(encoder_current/encoder_max));

  motor1.drive(-1*gain_motor);
```

Figure 17. P Controller for Wrist Motion

The ESP32 controlling this interface also creates a webserver using the ESP32 Webserver example provided by Dr. Ridgely. The example was altered to display the IMU readings that are used to control the arm and wrist motion. To access the webpage the user must connect their device to the WIFI of the ESP32. The WIFI name is robot_arm and the password is"password." Once connected to the WIFI, the user can access the webpage at the ESP32 address: 192.168.5.1. From there every time the IMU data button is clicked, the ESP32 will read 20 pwm values over 10 seconds and print the values in csv format once collected. After that the user can press the back button the webpage and collect IMU data again.

Results

Our system worked relatively well as we were able to accomplish the goal of controlling a robotic arm with a glove. The motion of the robotic arm was demonstrated to the class and showed each movement from the robot that we implemented. This included fingers opening and closing, wrist spinning, and arm up and down motions. With all these movements we were able to implement tasks structures and system control which we had learned from previous courses and new topics such as the creation of a webserver.

Although we did accomplish our goals, we did have some troubles in the project. The first was that our board design was not able to be used as we had initially intended due the motor driver using pins that were not useable in the ESP32. Due to this we had to split the software into two different ESP32; one using the board that we designed to control the servos and the other on a breadboard which controlled the wrist and arm motion. Another problem that we encountered was that our IMU did not have good resistors on the breakout board so the ESP32 had trouble taking readings from it. Finding this problem took many hours of debugging and help from our professor Ridgely. Once these two problems were ironed out the project proceeded very smoothly.

One thing that we would change to improve the functionality of the hand is replace the finger servos so that the fingers could actuate in a smoother manner. This would also give us a lot more grip strength and give us the possibility of holding objects. Another part that we would like to change is trying to implement everything into one microprocessor instead of having to split up the function between two ESP32. This would make the wiring and set up look a lot cleaner. In terms of software structure, we would like to change it so that it has different states. The way we have our software developed each task only has “one” constant state. Setting it up with different states would give us more control and options to add new features.

Overall, it was really satisfying and rewarding project seeing our idea go from a sketch on paper to a functioning robotic arm.

Video of working prototype: https://youtube.com/shorts/G3eh66s_egA?feature=share