1. Overview/Project Description

This project implements a simple five‑finger robot hand using an ESP32 Dev Module, MG90S micro‑servos, a 4×4 membrane keypad, and a 1602 parallel LCD, where all the heavy lifting and complicated calculations are performed by the ROS 2 environment.

The way the project flows is as follows:

1. Keypad Input: User presses a key on the mesh pad, so far the limit is 10 but can be expanded.
2. ROS 2 Request: Arduino (ESP32) publishes the requested position to a ROS 2 topic.
3. ROS 2 Response: A ROS 2 node sends back five servo angles via a custom message, due to there being 5 fingers.
4. Servo Motors Move: ESP32 reads the angles and drives each finger servo.
5. LCD Feedback: The LCD displays “Position #” immediately after the key on the keypad is pressed.
6. Hold & Reset: The hand holds the pose for 5 s then returns to flat “rest” (open‑palm) position.

Due to some issues, there are two versions of Arduino sketches. In total there are two Arduino sketches (direct‑drive and PCA9685‑driven) and two ROS 2 packages (`rob499_final_cm` for messages/services, `rob499_final` for Python logic).

2. Motivation & Goals

- One of the main drives was to challenge myself with what I was doing. I have always enjoyed using the Arduino IDE, and I thought it would be interesting to see how it could connect to the ROS 2 environment.
- Another reason behind why I chose a robotic hand, is because in my family my grandpa is missing a limb, so I have had some exposure and experience with the need for prosthetic limbs.
- I also wanted the ability to see if demonstration learning (muscle memory) could be achieved, and would be able to be expandable.
- I wanted to also test my ability to create python modules, since it is something I struggle with.

3. System Architecture

Packages:
`rob499_final_cm`(CMake)

- `Position.msg` (`int32[5] angles`)
- `SelectPosition.srv` (request: `int32 index`; response: `Position`)

`rob499_final` (ament_python)

- Nodes:
    - `position_input_node.py` (manual input → JSON storage)
    - `averaging_node.py` (average folder of JSON configs)
    - `processed_position_node.py` (rename raw → final)
    - `keypad_trigger_node.py` (CLI keypad simulation → `/final_position`)
    - `arduino_comm_node.py` (simulate sending `/final_position` → Arduino)

Launch file: `launch_all.py` to start the full pipeline

These nodes proved the concept of recording, storing, averaging, and publishing final gesture data. They can be integrated with micro-ROS on hardware, or run entirely in simulation, however I did not test this.

Code on the Arduino IDE:
Sketches:

1. `rob499_arduino_code.ino` (PCA9685)
    a. Subscribes to `/servo_positions` (Float32MultiArray)
    b. Displays "Position #" on a 1602 LCD
    c. Drives up to 16 servos via PCA9685 over I^{2}C
2. `rob499_arduino_no_driver.ino` (direct-drive)
    a. Publishes keypad presses to `/position_request`
    b. Subscribes to `/servo_positions`
    c. Drives five servos directly on ESP32 GPIOs (using ESP32Servo)
    d. Provides more reliable operation under heavy load
    e. Breadboard power rails must be bridged on both sides.
    f. LCD contrast pot must be wired correctly (left→GND, right→5 V, middle→V0).
    g. The PCA9685 board presented power-bus issues under load; direct-drive sketch was used to work around this.

4. Results

- I was able to establish a clear connection between the Arduino IDE and the ROS 2 environment.
- I was able to successfully create all python packages needed to establish the connection, and to perfectly execute the calculations needed for the outcome of this project.

- I was able to successfully construct a robot hand. There did have to be two versions, with one final version that did not use a servo motor driver.
- I was able to get successful LCD displays and successful movements of each driver when prompted.

Here are some direct tests we can do on the robot hand to test whether or not it functions as it is supposed to.
Using the direct-drive sketch:

1. 1. Keypad: reliably accepts 1–9; non-numeric keys ignored.
2. 2. LCD: consistently displays “Position #” with no gibberish after remapping to conflict-free pins.
3. 3. Five servos: sweep to nine distinct gestures:
    a. Pinky closes
    b. Pinky + ring close
    c. Pinky + ring + middle close
    d. Pinky + ring + middle + index close
    e. All five fingers close
    f. Pinky + ring + thumb close
    g. All fingers half-close
4. Each position is held for 5 seconds, and then goes back to rest.

5. Challenges & Lessons Learned

Here are some of the challenges I have already mentioned:

- I was able to solve this by having to read through a bunch of documents and learning how to pick older file versions of certain programs that would enable the code to run.
- One of the biggest challenges I had was just establishing a simple connection between the arduino IDE and ROS 2 environment.
- Another challenge I had was when I burnt out the ESP32 dev module, and had to replace it. The problem appeared when using the new board, since the PCA6985 or servo motor driver did not work anymore. It was the same code and same pins, however no matter what it would not work.
- I solved this problem by connecting the servos directly to the ESP32 board rather than a servo driver.

Here are some other challenges I faced that I did not mention earlier:

1. Power Delivery: Servos draw high inrush current; PCA9685’s power bus had to be bypassed with direct-drive and proper decoupling caps.
2. Pin Conflicts: ESP32 has many multipurpose pins; careful mapping is essential to avoid “input-only” or boot-strap conflicts.

3. Breadboard had loose railings that allowed jumper wires to slip out easily which caused some electrical issues.

6. What I have Learnt

The things I already established I learnt:

- I learnt the hard way that wiring and code must match, and that even small changes in wire setup have to be reflected in your code.
- I also learnt that the ROS 2 communication is a lot more complicated than what I thought, since at least for me sometimes the communication is not 100%.
- I believe that if I had messed around more, I could have done it a lot easier. I definitely should have messed around with the Wi-Fi function of the chip, which would facilitate communication across the two programs.
- I learnt that clear node architecture is ESSENTIAL, especially when working on something with so many layers, where each layer is essential to the next. Since there were so many places where stuff could go wrong, having clear architecture was essential.

Here are some other thing I learnt:

- Hardware "Small Errors" Multiply
- A single mis-wired breadboard rail or an unused OE pin on the PCA9685 could kill all five servos. You learn to check power rails, decoupling caps, and solder jumpers systematically, never assume "it's powered."
- Pin Selection Matters, not all pins are created equal.
- On the ESP32 many GPIO pins are input-only or reserved for boot/flash. Swapping the LCD and servo pins to non-conflicting, PWM-capable pins saved hours of "gibberish" and "no movement" debugging.
- Multiple MG90S servos inrush hundreds of milliamps. Adding a 1000 µF electrolytic +0.1 µF ceramic capacitor next to the PCA9685 (or power bus) stabilized the 5 V rail and prevented system resets.
- Testing each subsystem (LCD alone, keypad alone, single-servo direct-drive, PCA9685 LED blink) in isolation was critical. You can't troubleshoot a five-finger demo until you know each piece works on its own.

7. How to run

1. Clone & Build ROS 2

~~~
```bash
cd ~
~~~

```
git clone git@github.com:ramos-mat/rob499-Robot-Hand.git
cd ros2_ws/src
ln -s ~/my_robot_project/rob499_final_cm .
ln -s ~/my_robot_project/rob499_final .
cd ~/ros2_ws
colcon build
source install/setup.bash
ros2 launch rob499_final launch_all.py
```

2. Upload Arduino Sketch
    a. Open rob499_arduino_no_driver.ino in Arduino IDE
    b. Install: ESP32Servo, LiquidCrystal, Keypad libraries
    c. Select Board → ESP32 Dev Module, Port → /dev/ttyUSB0
    d. Upload the boards.
3. How to operate the demo
    a. Press 1–9 on keypad
    b. LCD shows "Position #"
    c. Five servos move to the specified gesture
    d. Wait 5 s, hand returns to flat rest

8. Conclusion and Final Thoughts

This project has demonstrated the end-to-end integration of ROS 2 and embedded arduino IDE to create a simple, interactive robot hand. By combining ROS 2 Python nodes for gesture recording, averaging, and execution with an ESP32 with Arduino‑style firmware driving five servos, a keypad, and a LCD screen.

Overall, this work builds a solid foundation for demonstration learning in human-robot interaction and also development of ROS 2 skills, such as how to record, average, store, and replay gestures.