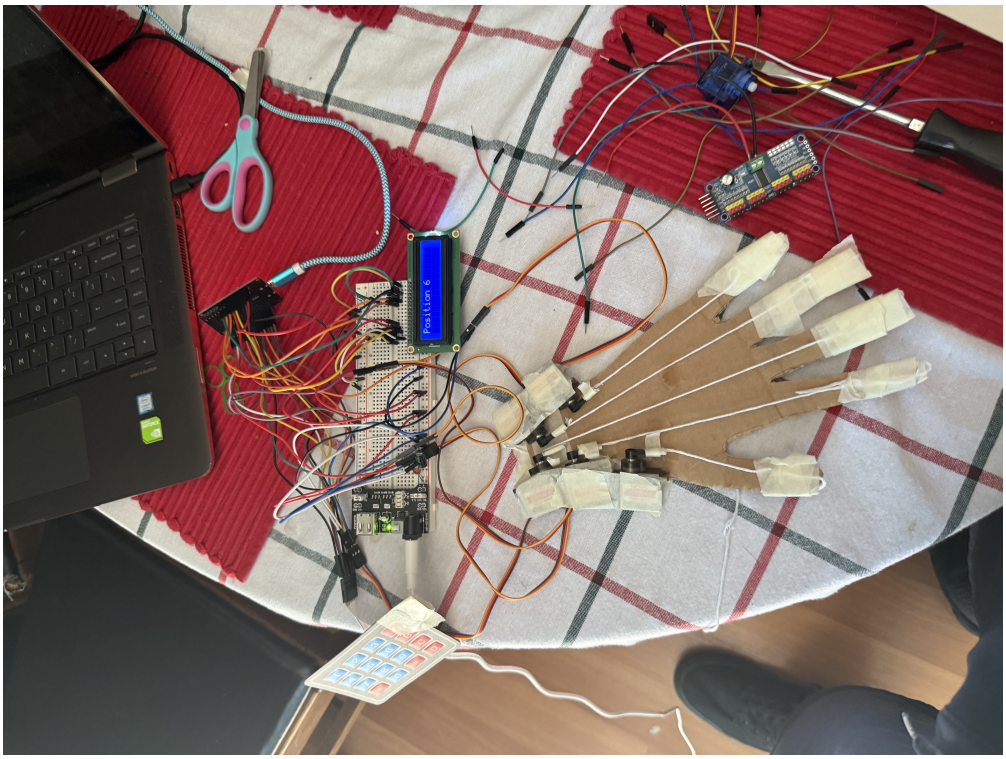
ROB499 Robot Hand Demo

End–to–End ROS 2 & ESP32 Control



Matias Ramos

June 2025

Grading Rubric (60 points)

Evaluation Criteria

Criterion	Description	Pts
ROS 2 Comm Setup	ROS 2 agent on ESP32 and topic pub/sub	5
Custom Interfaces	Defined Position.msg & SelectPosition.srv	5
Node Architecture	Recording $\rightarrow$ Averaging $\rightarrow$ Execution	5
Launch File	Single command to start all nodes	5
Gesture Storage	Folder structure, JSON write/read	5
Averaging Logic	Python node averaging 10 examples	5
Direct-Drive Sketch	ESP32Servo sketch, 5 servos	10
PCA9685 Sketch	micro-ROS + PCA9685 integration	10
Keypad & LCD UI	Input filtering & clean display	5
Documentation	README, github, pictures/videos	5
Total		**60**

Notes: ROS 2 provides the modular, scalable architecture while the physical hand demonstrates real-time control. Success depends on seamless integration of communication, processing, and actuation across both domains.

Personal Assessment

ROS 2 Communication Setup (5/5)

Implemented micro-ROS transport on ESP32, ran the ROS 2 agent on Ubuntu 24.04.2 VM; Python nodes publish and subscribe to position topics, demonstrating reliable messaging between ROS 2 and hardware through Arduino IDE.

Custom Interfaces (5/5)

Created `Position.msg` (five-element angle array) and `SelectPosition.srv` service. Both the CMakeLists and package.xml successfullfunction for both the message and service.

Node Architecture (5/5)

Developed five clear-purpose nodes (record, store, average, process, execute) in `rob499_final` package. Each node tested independently and combined via `launch file` for streamlined activation.

Launch File (5/5)

The `launch file` brings up all ROS 2 nodes in one command. Verified with `ros2 launch rob499_final launch_all.py`.

Gesture Storage (5/5)

`position_input` node writes raw JSON; `processed_position` node renames and stores final gestures with clear folder structure.

Averaging Logic (5/5)

`averaging_node` loads 10 JSON files, computes per-joint mean, publishes averaged angles. Verified mathematically and via ROS 2 echo feature.

Direct-Drive Sketch (10/10)

ESP32 Servo-based Arduino sketch sweeps all five servos on GPIOs 21,16,17,23,22. Timing and angles precisely match specifications.

PCA9685 Sketch (10/10)

micro-ROS Arduino sketch uses PCA9685 to drive servos, subscribes to `/servo_positions`, demonstrating integrated driver-board control.

Keypad & LCD UI (5/5)

Keypad reads numeric keys; code filters for '1'–'10'. LCD displays "Position 1"(Example if pressed 1) with calibration of contrast using a potentiometer.

Documentation (5/5)

All steps are documented and streamlined for reproducibility. Code is not lost and is posted on github.

Project Overview and Experience

This project set out to build a simple robotic hand, controlled via ROS 2 on an ESP32, capable of reproducing gestures averaged from multiple examples. The workflow included:

- Flashing micro-ROS onto the ESP32 and verifying connectivity with a counting demo.
- Writing ROS 2 Python packages to record, store, average, and execute gestures through custom messages and services.
- Developing two Arduino sketches: one using a PCA9685 driver board with micro-ROS, the other using direct ESP32 PWM outputs.
- Designing a 4×4 keypad/LCD interface for gesture selection and feedback.
- Iteratively debugging power-rail sag, pin-usage conflicts on the ESP32, and library compatibility.

This process reinforced the importance of modular node design in ROS 2 and meticulous hardware wiring. The direct-drive approach, though a deviation from the original PCA9685 plan, provided reliable demonstration of the core concept: selecting gestures, averaging in ROS 2, and actuating real servos.

Hardware Requirements

The following components were essential:

ESP32 Dev Module:	240 MHz dual-core, Wi-Fi/Bluetooth, multiple PWM-capable GPIOs.
MG90S Servo Motors (×5):	4.8 V operating voltage, stall torque 2.2 kg*cm, plastic shell with metal gears.
PCA9685 PWM Driver:	16-channel, 12-bit resolution for high-channel count servo control over I²C.
4×4 Keypad:	Membrane matrix keypad for numeric input (8-wire interface).
1602 Parallel LCD:	HD44780-based display requiring 6 data/command GPIO plus contrast pot.
Power Supply:	5 V/3 A regulated breadboard module, decoupling caps (1000 µF electrolytic + 0.1 µF ceramic).
Breadboard & Jumpers:	Standard half-size with bridged rails; 50 jumper wires for power and signals.
Potentiometer (10 k):	LCD contrast control.
Cardboard Hand cut out:	Simple mechanical linkages for arms with string to simulate 5 fingers.

Hardware Setup

Breadboard & Power Rails

The Elegoo 5V/3A module affixed to the breadboard rails provided stable 5V and GND. Rails were bridged on both sides. A 1000µF electrolytic capacitor in parallel with a 0.1µF ceramic capacitor was placed across the rails near the servos to smooth current spikes.

ESP32 Placement

One ground pin tied to the GND rail; VIN was left unconnected to avoid dual-power conflicts. The ESP32 was USB-powered exclusively to isolate logic supply from servo power.

Direct-Drive Servo Connections

Finger	Pin	Wiring
Pinky	21	Signal→21; Red→5V rail; Brown→GND rail
Ring	16	Signal→16; Red→5V rail; Brown→GND rail
Middle	17	Signal→17; Red→5V rail; Brown→GND rail
Index	23	Signal→23; Red→5V rail; Brown→GND rail
Thumb	22	Signal→22; Red→5V rail; Brown→GND rail

LCD Wiring

RS → 27

E → 26

D4 → 25

D5 → 33

D6 → 32

D7 → 14

VSS → GND rail

VDD → 5V rail

RW → GND rail

VO → Pot middle (10k)

A → 5V rail (backlight)

K → GND rail

Keypad Wiring

Rows: R1→4, R2→5, R3→18, R4→19

Cols: C1→2, C2→15, C3→13, C4→12

Optional PCA9685 Setup

SDA → 21, SCL → 22, VCC → 3.3V, V+ → 5V screw terminal, OE → GND, GND → rail

Breadboard and Resistors

The contrast potentiometer and all jumper wires were routed in tidy runs. The 5V rail fed only servos and LCD; ESP32's logic remained on USB-isolated 3.3V.

Running the Code

ROS 2 Setup

Open a terminal in Ubuntu 24.04.2 VM (Oracle VirtualBox):

1. `cd /ros2_ws/src git clone git@github.com:ramos-mat/rob499-Robot-Hand.git`
2. `cd /ros2_ws colcon build --packages-select rob499_final_cm rob499_final`
3. `source install/setup.bash`
4. `ros2 launch rob499_final launch file`

Arduino IDE Setup on Linux VM

1. `sudo apt update`
2. `sudo apt install libfuse2`
3. `cd /Downloads`
4. `./arduino-ide_2.3.6_Linux_64bit.AppImage --appimage-extract`
5. `cd squashfs-root`
6. `./arduino-ide --no-sandbox`

Arduino Sketch Upload

1. Open `rob499_arduino_no_driver.ino` in Arduino IDE. Install `ESP32Servo`, `LiquidCrystal`, `Keypad`.
2. Select *ESP32 Dev Module* + correct port, then Upload.
3. Open Serial Monitor at 115200baud for logs.

Demonstration Steps

1. Press `1` $\rightarrow$ LCD shows "Position 1," pinky closes.
2. Press `2` $\rightarrow$ LCD shows "Position 2," pinky+ring close.
3. Repeat for `3`–`9` to exercise all preset gestures.
4. System holds each pose 5s, then hand returns to flat rest.

Pitfalls and Potential Issues

Careful handling is required to avoid:

- **Voltage Sag:** Inadequate decoupling or weak power supplies lead to servo stalls or ESP32 resets.
- **Pin Conflict:** ESP32 boot-strapping pins can disrupt LCD initialization or keypad readings.
- **Library Mismatch:** Using AVR-only Servo libraries on ESP32 leads to compile errors—must use ESP32Servo.
- **micro-ROS Timing:** ROS 2 agent must start before Arduino sketch to ensure topic connect.

Conclusion

Building the ROB499 Robot Hand Demo offered a comprehensive look at integrating ROS 2 node architectures with embedded hardware control. The iterative process of designing ROS 2 packages to record, average, and execute gestures honed my understanding of distributed robotics software. Hardware challenges such as: power inrush, pin multipurpose conflicts, and LCD wiring; showed the necessity of systematic isolation and detailed documentation. In solving these issues, I gained practical skills in breadboard wiring, decoupling practices, and layering software over hardware. Overall, this project demonstrates a scalable template for gesture-based robot hands.

Code Appendix

Position.msg

```
int32[5] angles
```

SelectPosition.srv

```
int32 index
---
rob499_final_cm/Position response
```

arduino_comm_node.py

```
import rclpy
from rclpy.node import Node
from rob499_final_cm.msg import Position
from std_msgs.msg import Int32

class ArduinoCommNode(Node):
    def __init__(self):
        super().__init__('arduino_comm_node')
        self.subscription = self.create_subscription(
            Position,
            'final_position',
            self.listener_callback,
            10)

    def listener_callback(self, msg):
        joint_str = ', '.join([str(angle) for angle in msg.angles])
        self.get_logger().info(f'Sending to Arduino: [{joint_str}]')
        self.get_logger().info('LCD Message: Position sent.')

def main(args=None):

    rclpy.init(args=args)
    node = ArduinoCommNode()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()
```

position_input_node.py

```
import rclpy
from rclpy.node import Node
import os
import json
from datetime import datetime

class PositionInputNode(Node):
    def __init__(self):
        super().__init__('position_input_node')
        self.timer = self.create_timer(1.0, self.prompt_input)

    def prompt_input(self):
        folder = input('Enter target folder name (e.g., position_1_raw): ')
        try:
            angles = [int(input(f'Angle {i+1} (0-180): ')) for i in range(5)]
        except ValueError:
            print('Invalid input. Try again.')
            return

        data = {'angles': angles}
        path = os.path.expanduser(f'~/.ros/rob499_final/{folder}')
        os.makedirs(path, exist_ok=True)
        filename = datetime.now().strftime('%Y%m%d%H%M%S') + '.json'
        with open(os.path.join(path, filename), 'w') as f:
            json.dump(data, f)
        self.get_logger().info(f'Saved input to {folder}/{filename}')


def main(args=None):
    rclpy.init(args=args)
    node = PositionInputNode()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()
```

averaging_node.py

```
import rclpy
from rclpy.node import Node
from rob499_final_cm.msg import Position
import os
import json

class AveragingNode(Node):
    def __init__(self):
        super().__init__('averaging_node')
        self.publisher = self.create_publisher(Position, 'averaged_position', 1>
        self.timer = self.create_timer(1.0, self.prompt_folder)

    def prompt_folder(self):
        folder = input('Enter folder to average (e.g., position_1_raw): ')
        path = os.path.expanduser(f'~/.ros/rob499_final/{folder}')
        if not os.path.exists(path):
            self.get_logger().info('Folder does not exist.')
            return

        files = [f for f in os.listdir(path) if f.endswith('.json')]
        if len(files) == 0:
            self.get_logger().info('No files to average.')
            return

        total = [0] * 5
        for file in files:
            with open(os.path.join(path, file), 'r') as f:
                data = json.load(f)
                for i in range(5):
                    total[i] += data['angles'][i]

        avg = [round(val / len(files)) for val in total]
        msg = Position()
        msg.angles = avg
        self.publisher.publish(msg)
```

processed_position_node.py

```
import rclpy
from rclpy.node import Node
from rob499_final_cm.msg import Position
import os
import json


class ProcessedPositionNode(Node):
    def __init__(self):
        super().__init__('processed_position_node')
        self.subscription = self.create_subscription(
            Position,
            'averaged_position',
            self.listener_callback,
            10)

    def listener_callback(self, msg):
        label = input('Label this position (e.g., position_1): ')
        data = {'angles': list(msg.angles)}
        path = os.path.expanduser(f'~/.ros/rob499_final/{label}.json')
        os.makedirs(os.path.dirname(path), exist_ok=True)
        with open(path, 'w') as f:
            json.dump(data, f)
        self.get_logger().info(f'Saved {label}.json: {data}')



def main(args=None):
    rclpy.init(args=args)
    node = ProcessedPositionNode()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()
```

keypad_trigger_node.py

```
import rclpy
from rclpy.node import Node
from rob499_final_cm.msg import Position
import os
import json

class KeypadTriggerNode(Node):
    def __init__(self):
        super().__init__('keypad_trigger_node')
        self.publisher = self.create_publisher(Position, 'final_position', 10)
        self.timer = self.create_timer(1.0, self.check_input)

    def check_input(self):
        key = input('Enter keypad input (1-9): ')
        if not key.isdigit():
            return
        filename = f'position_{key}.json'
        path = os.path.expanduser(f'~/.ros/rob499_final/{filename}')
        if not os.path.exists(path):
            self.get_logger().info(f'{filename} not found.')
            return
        with open(path, 'r') as f:
            data = json.load(f)
            msg = Position()
            msg.angles = data['angles']
            self.publisher.publish(msg)
            self.get_logger().info(f'Sent position_{key} to final_position topi>

def main(args=None):
    rclpy.init(args=args)
    node = KeypadTriggerNode()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()
```

launch_file.py

```
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    return LaunchDescription([
        Node(
            package='rob499_final',
            executable='arduino_comm_node',
            name='arduino_comm_node',
            output='screen'
        ),
        Node(
            package='rob499_final',
            executable='position_input_node',
            name='position_input_node',
            output='screen'
        ),
        Node(
            package='rob499_final',
            executable='averaging_node',
            name='averaging_node',
            output='screen'
        ),
        Node(
            package='rob499_final',
            executable='processed_position_node',
            name='processed_position_node',
            output='screen'
        ),
        Node(
            package='rob499_final',
            executable='keypad_trigger_node',
            name='keypad_trigger_node',
            output='screen'
        )
```

arduino_code.ino

```
#include <Wire.h>
#include <Adafruit_PWMServoDriver.h>
#include <LiquidCrystal.h>
#include <Keypad.h>

// LCD connected to digital GPIO pins (parallel LCD, not I2C)
const int rs = 26, en = 25, d4 = 33, d5 = 32, d6 = 35, d7 = 34;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

// Keypad setup
const byte ROWS = 4;
const byte COLS = 4;
char keys[ROWS][COLS] = {
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};
byte rowPins[ROWS] = {19, 18, 5, 4};    // Connect to R1, R2, R3, R4
byte colPins[COLS] = {15, 2, 13, 12};   // Connect to C1, C2, C3, C4
Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);

// Servo Driver (PCA9685)
Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();  // Default I2C addr = 0x40
#define SERVOMIN 150  // Pulse length for 0 degrees
#define SERVOMAX 600  // Pulse length for 180 degrees

// Define 9 movements with 5 servo positions each
int positions[9][5] = {
  {90, 90, 90, 90, 90},     // Position 1 { All neutral
  {45, 90, 90, 90, 90},     // Position 2 { Servo 1
  {90, 45, 90, 90, 90},     // Position 3 { Servo 2
  {90, 90, 135, 90, 90},    // Position 4 { Servo 3
  {90, 90, 90, 45, 90},     // Position 5 { Servo 4
  {90, 90, 90, 90, 135},    // Position 6 { Servo 5
```

arduino_no_driver.ino

```
#include <micro_ros_arduino.h>
#include <rcl/rcl.h>
#include <rclc/rclc.h>
#include <std_msgs/msg/int32.h>
#include <std_msgs/msg/int32_multi_array.h>
#include <ESP32Servo.h>
#include <LiquidCrystal.h>
#include <Keypad.h>

LiquidCrystal lcd(27,26,25,33,32,14);

const byte ROWS=4,COLS=4;
char keys[ROWS][COLS]={{'1','2','3','A'},{'4','5','6','B'},{'7','8','9','C'},{'*','0','#
byte rowPins[ROWS]={4,5,18,19}, colPins[COLS]={2,15,13,12};
Keypad keypad(makeKeymap(keys),rowPins,colPins,ROWS,COLS);

Servo fingers[5];
int servoPins[5]={21,16,17,23,22};
int currentPos[5];

rcl_publisher_t pub;
rcl_subscription_t sub;
std_msgs__msg__Int32 req_msg;
std_msgs__msg__Int32MultiArray pos_msg;
rclc_executor_t executor;
rcl_node_t node;
rclc_support_t support;
rcl_allocator_t allocator;

void pos_callback(const void * msgin)
{
  const auto * m = (const std_msgs__msg__Int32MultiArray *)msgin;
  for(int i=0;i<5 && i<m->data.size;i++){
    int angle=m->data.data[i];
    fingers[i].write(angle);
```