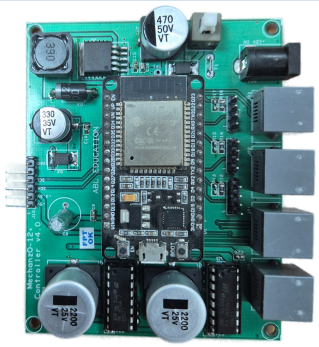
ROBOQUEST

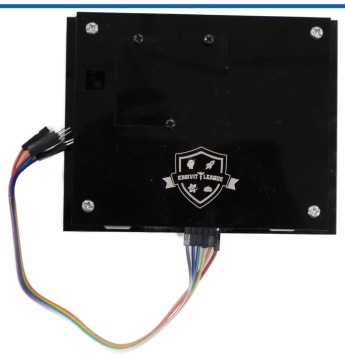
CONTROLLER & REMOTE V4.0

Instruction & User Manual

WIRELESS ROBOT CONTROL • 4-MOTOR DRIVE • 4DoF ROBOTIC ARM SUPPORT



RoboQuest Controller V4.0 — external view



Controller enclosure and connector arrangement

RJ11 motor connectors and DC power input

1 Product Overview

The RoboQuest V4.0 system consists of a wireless RoboQuest Remote V4.0 and RoboQuest Controller V4.0. Together they provide wireless control of a four-motor robot and can be expanded with a PCA9685 servo driver for a 4DoF robotic arm. The controller also provides three 4-pin ultrasonic sensor interfaces for autonomous applications such as maze solving and obstacle avoidance.

2 System at a Glance

COMPONENT	FUNCTION
RoboQuest Remote V4.0	Wireless user interface for robot movement and robotic-arm servo control
RoboQuest Controller V4.0	Receives wireless commands and drives four DC motors
PCA9685 Servo Driver	Servo interface for a 4DoF robotic arm
Ultrasonic Sensor Ports	Three 4-pin interfaces for autonomous sensing
ESP-NOW	Wireless communication between remote and controller

3 RoboQuest Controller V4.0

The controller is an ESP32-based motor controller with two L293D motor-driver ICs and four RJ11 motor outputs. A 12V Li-ion battery connects through the DC jack. A 6-pin connector is provided for a PCA9685 servo driver, and three 4-pin connectors near the ESP32 are available for ultrasonic sensors.

3.1 Features

- ESP32-based wireless control
- 2 × L293D motor drivers
- 4 × RJ11 motor connectors
- 12V Li-ion battery input via DC jack
- M1 and M3 = left-side motors (can be changed as per the requirement)
- M2 and M4 = right-side motors
- 6-pin PCA9685 interface
- 3 × 4-pin ultrasonic sensor connectors

3.2 Motor Port Mapping

PORT	ROBOT POSITION	PURPOSE
Motor 1	Left side	Left-side drive motor
Motor 2	Right side	Right-side drive motor
Motor 3	Left side	Left-side drive motor
Motor 4	Right side	Right-side drive motor

3.3 RJ11 Motor Connection

Each RJ11 connector connects one DC motor. In the current design, RJ11 Pin 2 and Pin 3 carry the motor power connections (12V and GND). Motor direction depends on polarity. If required, reverse the motor polarity according to the robot wiring. Do not connect unrelated signals or external power to motor ports.

3.4 Power Input

Connect the recommended 12V Li-ion battery to the DC jack located on the same face as the four RJ11 motor connectors. Confirm battery voltage and polarity before powering the system.

3.5 PCA9685 Servo Driver

A 6-pin connector is located on the side opposite the RJ11/DC-jack face. Jumper wires can be used to connect a PCA9685 module. The PCA9685 provides servo control for a 4DoF robotic arm.

3.6 Ultrasonic Sensor Ports

Three 4-pin connectors near the ESP32 are provided for ultrasonic sensors. They allow the controller to be used for maze-solving, obstacle avoidance and autonomous navigation projects.

4 RoboQuest Remote V4.0

The RoboQuest Remote V4.0 is the handheld wireless controller. It uses an ESP32 development module and ESP-NOW communication. Two joysticks control four servo motors for a robotic hand/4DoF arm, while eight buttons provide out of which right side 4 buttons for robot movement and programmable inputs.

4.1 Specifications

PARAMETER	SPECIFICATION
Product	RoboQuest Remote V4.0
Controller	ESP32 Development Module
Communication	ESP-NOW
Pairing	Receiver MAC-ID mapped
Approx. range	50 m
Joysticks	2
Buttons	8
Battery	2 × 3.7V 1000 mAh Li-ion
Charging	Remove batteries and use compatible external Li-ion charger
Connection indicator	Blue onboard ESP32 LED

4.2 Joysticks

The two joysticks are used to control four servo motors for the robotic hand/4DoF robotic arm. The exact servo-axis mapping is determined by the installed firmware.

4.3 Buttons

BUTTON GROUP	CURRENT FUNCTION
Right-side 4 buttons	Forward, Backward, Left and Right robot movement
Left-side 4 buttons	Programmable; currently not in use

The four left-side buttons are reserved for future firmware functions such as gripper control, special robot actions or competition-specific commands.

5 Wireless Connection Procedure

1. Power the RoboQuest Controller first.
2. Wait for the controller to initialize.
3. Turn ON the RoboQuest Remote V4.0.
4. The remote automatically communicates with the receiver configured by MAC ID.
5. When connection is established, the blue onboard ESP32 LED indicates the connected/ communication status.
6. Operate the robot using the right-side movement buttons and the robotic arm using the joysticks.

6 Battery & Charging

The remote uses two 3.7V 1000 mAh Li-ion batteries. The batteries must be removed before charging.

- Turn OFF the remote.
- Remove both batteries.
- Charge them using a compatible Li-ion battery charger.
- Check battery and charger condition.
- Reinstall batteries with correct polarity.
- Do not use damaged, swollen or leaking batteries.

7 Normal Operating Sequence

STEP	ACTION
1	Install charged remote batteries
2	Connect 12V Li-ion battery to controller
3	Turn ON controller
4	Turn ON remote
5	Confirm blue ESP32 LED connection indication
6	Test robot movement in a safe area
7	Operate robotic arm using joysticks
8	Turn OFF system after use

8 Safety Precautions

- Disconnect power before changing wiring.
- Use the recommended 12V battery for the controller.
- Remove remote batteries before charging.
- Use a compatible Li-ion charger.
- Keep hands and objects away from moving wheels and robotic mechanisms.
- Do not short battery terminals.
- Verify motor direction before operation.

9 Troubleshooting

PROBLEM	POSSIBLE CAUSE	ACTION
Remote does not connect	Controller not ON	Turn controller ON before remote
Remote does not connect	Low battery	Charge batteries
No blue connection indication	Range/communication issue	Move closer and restart both units
Robot moves opposite	Motor polarity/wiring	Check and reverse motor polarity if required
Motor not moving	RJ11/motor connection	Power OFF and inspect connection
Servo not moving	PCA9685/wiring/power	Check PCA9685 and servo connections
Ultrasonic not working	Sensor connection/firmware	Check 4-pin connection and firmware

10 Applications

- RoboQuest competition robots
- 4DoF robotic arm/hand
- Educational robotics
- Maze-solving robots
- Obstacle-avoidance robots
- STEM and robotics laboratory activities

11 Quick Reference

ITEM	INFORMATION
Controller power	12V Li-ion battery via DC jack
Motor outputs	4 × RJ11
Motor mapping	M1/M3 Left; M2/M4 Right
Servo expansion	6-pin PCA9685 interface
Ultrasonic expansion	3 × 4-pin ports
Remote battery	2 × 3.7V 1000 mAh Li-ion
Remote charging	Remove batteries; use external Li-ion charger
Wireless	ESP-NOW / MAC-ID mapped
Range	Approx. 50 m
Start sequence	Controller ON → Remote ON
Connection indicator	Blue onboard ESP32 LED

12 Document Information

Product: RoboQuest Controller V4.0 and RoboQuest Remote V4.0
Document: Instruction & User Manual
Revision: 1.0

13 MAC ID identification

Purpose

This program is used to identify the **Wi-Fi Station (STA) MAC Address** of the ESP32 used in the RoboQuest Controller or RoboQuest Remote.

The MAC address is required for configuring **ESP-NOW communication**, as the RoboQuest Remote communicates with the specific RoboQuest Controller using its receiver MAC ID.

Arduino Code

```
#include <WiFi.h>
#include <esp_wifi.h>
void setup() {
  Serial.begin(115200);
  delay(1000);
  WiFi.mode(WIFI_MODE_STA);
  delay(500);
  uint8_t mac[6];
  esp_wifi_get_mac(WIFI_IF_STA, mac);
  Serial.print("MAC Address: ");
  for (int i = 0; i < 6; i++) {
    if (mac[i] < 16) {
      Serial.print("0");
    }
    Serial.print(mac[i], HEX);
    if (i < 5) {
      Serial.print(":");
    }
  }
  Serial.println();
}
void loop() {
}
```

How to Use the Program

1. Connect the ESP32 board to the computer using a USB cable.
2. Open the Arduino IDE.
3. Select the correct **ESP32 board** under **Tools → Board**.
4. Select the correct **COM Port** under **Tools → Port**.
5. Upload the above program to the ESP32.
6. Open the **Serial Monitor**.
7. Set the baud rate to **115200 baud**.
8. The ESP32 MAC address will be displayed.

Example Serial Monitor Output

```
MAC Address: D4:8A:FC:9F:26:DC
```

Record this MAC address carefully. The MAC address of the **RoboQuest Controller** is required when configuring the RoboQuest Remote for ESP-NOW communication.

> **Important** — The MAC address is unique to the ESP32 device. Do not use the example MAC address shown above for another ESP32. Always run the MAC ID checking program on the actual ESP32 being configured.

14 RoboQuest Controller Firmware

Purpose

This program is the main firmware for the RoboQuest Controller V4.0. It receives control data wirelessly from the RoboQuest Remote using ESP-NOW and controls:

- Four DC motors through two L293D motor drivers
- Four servo motors through the PCA9685 servo driver
- A 4DoF robotic arm consisting of Base, Shoulder, Elbow and Gripper servos
- Forward, Backward, Left and Right robot movement

Hardware Configuration

COMPONENT	GPIO / CHANNEL	FUNCTION
L293D #1 ENA	GPIO 19	Rear motor enable
L293D #1 ENB	GPIO 23	Rear motor enable
Rear Right IN1/IN2	GPIO 33 / 32	Motor 2
Rear Left IN3/IN4	GPIO 13 / 4	Motor 1
L293D #2 ENA	GPIO 5	Front motor enable
L293D #2 ENB	GPIO 18	Front motor enable
Front Right IN1/IN2	GPIO 14 / 27	Motor 4
Front Left IN3/IN4	GPIO 26 / 25	Motor 3
PCA9685 Base	Channel 0	Base servo
PCA9685 Shoulder	Channel 1	Shoulder servo
PCA9685 Elbow	Channel 2	Elbow servo
PCA9685 Gripper	Channel 3	Gripper servo

Arduino Code

```
#include <WiFi.h>
#include <esp_now.h>
#include <Wire.h>
#include <Adafruit_PWMServoDriver.h>
// ================================================
// PCA9685
// ================================================
```

```
Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver(0x40);
// ====================================================
// SERVO CHANNELS
// ====================================================
#define BASE_SERVO      0
#define SHOULDER_SERVO  1
#define ELBOW_SERVO     2
#define GRIPPER_SERVO   3
// ====================================================
// SERVO LIMITS
// ====================================================
#define SERVO_MIN 120
#define SERVO_MAX 500
// ====================================================
// SERVO POSITIONS
// ====================================================
int basePos     = 90;
int shoulderPos = 90;
int elbowPos    = 90;
int gripperPos  = 90;
// ====================================================
// L293D #1 (Rear Motors)
// ====================================================
#define ENA1 19
#define ENB1 23
#define IN1_1 33      // Right Rear
#define IN2_1 32
#define IN3_1 13      // Left Rear
#define IN4_1 4
// ====================================================
// L293D #2 (Front Motors)
// ====================================================
#define ENA2 5
#define ENB2 18
#define IN1_2 14      // Right Front
#define IN2_2 27
#define IN3_2 26      // Left Front
#define IN4_2 25
// ====================================================
// ESP NOW DATA
// ====================================================
typedef struct
{
  char moveCmd;
  int servo1;
  int servo2;
  int servo3;
  int servo4;
} ControlData;
ControlData rxData;
// ====================================================
// ANGLE TO PCA9685 PULSE
// ====================================================
int angleToPulse(int angle)
{
  return map(angle, 0, 180, SERVO_MIN, SERVO_MAX);
}
// ====================================================
// APPLY SERVOS
// ====================================================
void applyServos()
{
  pwm.setPWM(BASE_SERVO, 0, angleToPulse(basePos));
  pwm.setPWM(SHOULDER_SERVO, 0, angleToPulse(shoulderPos));
  pwm.setPWM(ELBOW_SERVO, 0, angleToPulse(elbowPos));
  pwm.setPWM(GRIPPER_SERVO, 0, angleToPulse(gripperPos));
}
// ====================================================
// SERVO UPDATE
// ====================================================
```

```
void updateServos()
{
  if(rxData.servo1 > 110) basePos += 2;
  else if(rxData.servo1 < 70) basePos -= 2;
  if(rxData.servo2 > 110) shoulderPos += 2;
  else if(rxData.servo2 < 70) shoulderPos -= 2;
  if(rxData.servo3 > 110) elbowPos += 2;
  else if(rxData.servo3 < 70) elbowPos -= 2;
  if(rxData.servo4 > 110) gripperPos += 2;
  else if(rxData.servo4 < 70) gripperPos -= 2;
  basePos = constrain(basePos,0,180);
  shoulderPos = constrain(shoulderPos,20,160);
  elbowPos = constrain(elbowPos,20,160);
  gripperPos = constrain(gripperPos,30,120);
  applyServos();
}
// ====================================================
// MOTOR CONTROL
// ====================================================
void setMotors(
  int rr1, int rr2,
  int lr1, int lr2,
  int rf1, int rf2,
  int lf1, int lf2)
{
  // Rear Right
  digitalWrite(IN1_1, rr1);
  digitalWrite(IN2_1, rr2);
  // Rear Left
  digitalWrite(IN3_1, lr1);
  digitalWrite(IN4_1, lr2);
  // Front Right
  digitalWrite(IN1_2, rf1);
  digitalWrite(IN2_2, rf2);
  // Front Left
  digitalWrite(IN3_2, lf1);
  digitalWrite(IN4_2, lf2);
}
// ====================================================
// MOVEMENTS
// ====================================================
void forward()
{
  setMotors(
    LOW, HIGH,
    HIGH, LOW,
    HIGH, LOW,
    HIGH, LOW
  );
}
void back()
{
  setMotors(
    HIGH, LOW,
    LOW, HIGH,
    LOW, HIGH,
    LOW, HIGH
  );
}
void left()
{
  setMotors(
    LOW, HIGH,
    LOW, HIGH,
    HIGH, LOW,
    LOW, HIGH
  );
}
void right()
{
```

```
  setMotors(
    HIGH, LOW,
    HIGH, LOW,
    LOW, HIGH,
    HIGH, LOW
  );
}
void Stop()
{
  setMotors(
    LOW, LOW,
    LOW, LOW,
    LOW, LOW,
    LOW, LOW
  );
}
// =======================================================
// ESP NOW RECEIVE
// =======================================================
void onReceive(
  const esp_now_recv_info *info,
  const uint8_t *incomingData,
  int len)
{
  if(len != sizeof(ControlData))
    return;
  memcpy(&rxData, incomingData, sizeof(rxData));
  switch(rxData.moveCmd)
  {
    case 'F':
      forward();
      break;
    case 'B':
      back();
      break;
    case 'L':
      left();
      break;

    case 'R':
      right();
      break;
    default:
      Stop();
      break;
  }
  updateServos();
  Serial.print("CMD: ");
  Serial.print(rxData.moveCmd);
  Serial.print(" | Base:");
  Serial.print(basePos);
  Serial.print(" | Shoulder:");
  Serial.print(shoulderPos);
  Serial.print(" | Elbow:");
  Serial.print(elbowPos);
  Serial.print(" | Gripper:");
  Serial.println(gripperPos);
}
// =======================================================
// SETUP
// =======================================================
void setup()
{
  Serial.begin(115200);
  pinMode(ENA1, OUTPUT);
  pinMode(ENB1, OUTPUT);

  pinMode(ENA2, OUTPUT);
  pinMode(ENB2, OUTPUT);
  pinMode(IN1_1, OUTPUT);
```

```
  pinMode(IN2_1, OUTPUT);
  pinMode(IN3_1, OUTPUT);
  pinMode(IN4_1, OUTPUT);
  pinMode(IN1_2, OUTPUT);
  pinMode(IN2_2, OUTPUT);
  pinMode(IN3_2, OUTPUT);
  pinMode(IN4_2, OUTPUT);
  digitalWrite(ENA1, HIGH);
  digitalWrite(ENB1, HIGH);
  digitalWrite(ENA2, HIGH);
  digitalWrite(ENB2, HIGH);
  Stop();
  Wire.begin();
  pwm.begin();
  pwm.setPWMFreq(50);
  applyServos();
  WiFi.mode(WIFI_STA);
  WiFi.disconnect();
  if(esp_now_init() != ESP_OK)
  {
    Serial.println("ESP-NOW INIT FAILED");
    while(true);
  }

  esp_now_register_recv_cb(onReceive);
  Serial.println("ROBOT READY");
}
// =======================================================
// LOOP
// =======================================================
void loop()
{
}
```

Controller Functions

Motor Control

The controller supports four basic robot movements:

- **F** – Forward
- **B** – Backward
- **L** – Left
- **R** – Right
- **S** – Stop

Robotic Arm Control

The PCA9685 controls four servo channels:

SERVO	PCA9685 CHANNEL	INITIAL POSITION
Base	0	90°
Shoulder	1	90°
Elbow	2	90°
Gripper	3	90°

The servo operating limits are programmed individually to prevent excessive movement:

- Base: **0°–180°**
- Shoulder: **20°–160°**
- Elbow: **20°–160°**
- Gripper: **30°–120°**

ESP-NOW Communication

The controller receives a structured data packet containing:

- Robot movement command
- Base servo value
- Shoulder servo value
- Elbow servo value
- Gripper servo value

The received data is processed by the ESP-NOW receive callback, after which the controller updates the motors and robotic arm.

15 RoboQuest Remote Firmware

Purpose

The **RoboQuest Remote V4.0 firmware** is responsible for reading the eight push buttons and two joysticks and transmitting the control data to the RoboQuest Controller using **ESP-NOW**.

The remote sends:

- Robot 1 movement commands
- Robot 2 movement commands
- Four servo control values
- ESP-NOW connection status through the onboard LED

Remote Control Mapping

CONTROL	GPIO	FUNCTION
S1	GPIO 22	Robot 1 – Forward
S2	GPIO 23	Robot 1 – Backward
S3	GPIO 25	Robot 1 – Right
S4	GPIO 26	Robot 1 – Left
S5	GPIO 13	Robot 2 – Forward
S6	GPIO 18	Robot 2 – Backward
S7	GPIO 19	Robot 2 – Right
S8	GPIO 21	Robot 2 – Left
Right Joystick X	GPIO 35	Servo 1
Right Joystick Y	GPIO 34	Servo 2
Left Joystick X	GPIO 33	Servo 3
Left Joystick Y	GPIO 32	Servo 4
ESP32 LED	GPIO 2	ESP-NOW transmission status

Servo Joystick Mapping

The joystick values are converted into servo angles from **0° to 180°**.

JOYSTICK	ESP32 PIN	SERVO
Right X-axis	GPIO 35	Servo 1 – Base
Right Y-axis	GPIO 34	Servo 2 – Shoulder
Left X-axis	GPIO 33	Servo 3 – Elbow
Left Y-axis	GPIO 32	Servo 4 – Gripper

The joystick center region is approximately **90°**, while movement toward either side produces a corresponding servo angle.

Arduino Code

```
// ====================================
// ROBOQUEST REMOTE 4.0
// ESP-NOW TRANSMITTER
// ====================================
#include <WiFi.h>
#include <esp_now.h>
// ====================================
// RECEIVER MAC
// ====================================
uint8_t receiverMAC[] =
{
  0x20, 0x9B, 0xA9, 0x7C, 0xE7, 0x7C
};
// ====================================
// JOYSTICKS
// ====================================
#define R_VRX 35
#define R_VRY 34
#define L_VRX 33
#define L_VRY 32
// ====================================
// ROBOT 1 BUTTONS
// ====================================
#define S1 22      // Forward
#define S2 23      // Backward
#define S3 25      // Right
#define S4 26      // Left
// ====================================
// ROBOT 2 BUTTONS
// ====================================
#define S5 13      // Forward
#define S6 18      // Backward
#define S7 19      // Right
#define S8 21      // Left
// ====================================
#define LED_PIN 2
// ====================================
// DATA PACKET
// ====================================
typedef struct
{
    char robot1Cmd;
    char robot2Cmd;
    int servo1;
    int servo2;
    int servo3;
    int servo4;
} ControlData;
ControlData data;
// ====================================
// SEND CALLBACK
// ====================================
void onSend(const wifi_tx_info_t *info,
            esp_now_send_status_t status)
{
    digitalWrite(LED_PIN,
                 status == ESP_NOW_SEND_SUCCESS);
}
// ====================================
// JOYSTICK TO SERVO
// ====================================
int joystickToServo(int value)
{
    if (value >= 1700 && value <= 2400)
        return 90;
```

```
    if (value < 1700)
        return map(value, 0, 1699, 0, 89);
    return map(value, 2401, 4095, 91, 180);
}
// ====================================
// SETUP
// ====================================
void setup()
{
    Serial.begin(115200);
    pinMode(LED_PIN, OUTPUT);
    pinMode(S1, INPUT_PULLUP);
    pinMode(S2, INPUT_PULLUP);
    pinMode(S3, INPUT_PULLUP);
    pinMode(S4, INPUT_PULLUP);
    pinMode(S5, INPUT_PULLUP);
    pinMode(S6, INPUT_PULLUP);
    pinMode(S7, INPUT_PULLUP);
    pinMode(S8, INPUT_PULLUP);
    WiFi.mode(WIFI_STA);
    WiFi.disconnect();
    if (esp_now_init() != ESP_OK)
    {
        Serial.println("ESP NOW INIT FAILED");
        return;
    }

    esp_now_register_send_cb(onSend);
    esp_now_peer_info_t peerInfo = {};
    memcpy(peerInfo.peer_addr,
           receiverMAC,
           6);
    peerInfo.channel = 0;
    peerInfo.encrypt = false;
    if (esp_now_add_peer(&peerInfo) != ESP_OK)
    {
        Serial.println("PEER ADD FAILED");
        return;
    }
    Serial.println("ROBOQUEST REMOTE 4.0 READY");
}
// ====================================
// LOOP
// ====================================
void loop()
{
    //---------------------------------
    // Default Stop
    //---------------------------------
    data.robot1Cmd = 'S';
    data.robot2Cmd = 'S';
    //---------------------------------
    // ROBOT 1
    //---------------------------------
    if (!digitalRead(S1))
        data.robot1Cmd = 'F';
    else if (!digitalRead(S2))
        data.robot1Cmd = 'B';
    else if (!digitalRead(S3))
        data.robot1Cmd = 'R';
    else if (!digitalRead(S4))
        data.robot1Cmd = 'L';
    //---------------------------------
    // ROBOT 2
    //---------------------------------
    if (!digitalRead(S5))
        data.robot2Cmd = 'F';
    else if (!digitalRead(S6))
        data.robot2Cmd = 'B';
```

```
    else if (!digitalRead(S7))
        data.robot2Cmd = 'R';
    else if (!digitalRead(S8))
        data.robot2Cmd = 'L';
    //---------------------------------
    // Read Joysticks
    //---------------------------------
    int rvx = analogRead(R_VRX);
    int rvy = analogRead(R_VRY);
    int lvx = analogRead(L_VRX);
    int lvy = analogRead(L_VRY);
    //---------------------------------
    // Servo Angles
    //---------------------------------
    data.servo1 = joystickToServo(rvx);
    data.servo2 = joystickToServo(rvy);
    data.servo3 = joystickToServo(lvx);
    data.servo4 = joystickToServo(lvy);
    //---------------------------------
    // Send Packet
    //---------------------------------
    esp_now_send(receiverMAC,
                 (uint8_t *)&data,
                 sizeof(data));
    //---------------------------------
    // Debug
    //---------------------------------
    Serial.print("R1:");
    Serial.print(data.robot1Cmd);
    Serial.print("  R2:");
    Serial.print(data.robot2Cmd);
    Serial.print(" | Servo1:");
    Serial.print(data.servo1);
    Serial.print(" Servo2:");
    Serial.print(data.servo2);
    Serial.print(" Servo3:");
    Serial.print(data.servo3);
    Serial.print(" Servo4:");
    Serial.println(data.servo4);
    Serial.print("RVX:");
    Serial.print(rvx);

    Serial.print(" RVY:");
    Serial.print(rvy);
    Serial.print(" LVX:");
    Serial.print(lvx);
    Serial.print(" LVY:");
    Serial.println(lvy);
    Serial.println("--------------------------------");
    delay(40);
}
```

Scan this QR code for the assembly of 4DoF Robotic arm

NB: The Servo motors to be connected to the controller before fixing it to the robotic arm for adjusting the angle of rotation.