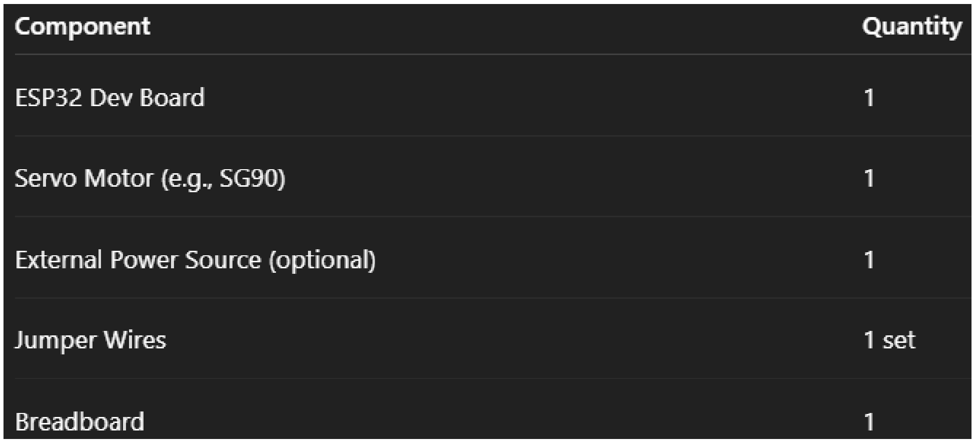
ESP32 Servo Motor Control using ESP32Servo Library

1. Objective

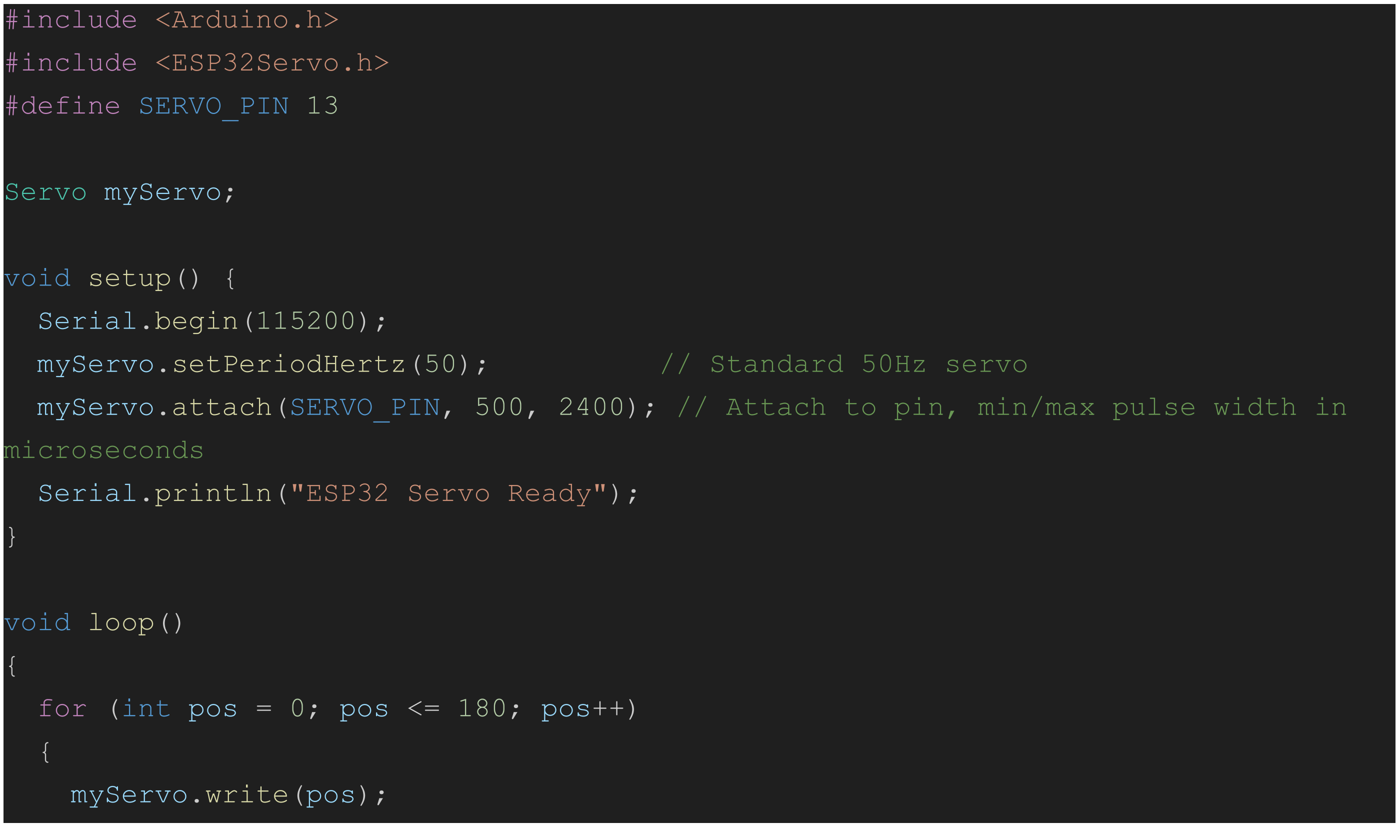
The goal of this project is to **control a servo motor** using an **ESP32** and make it **sweep smoothly from 0° to 180° and back**, repeating indefinitely.

2. Required Components



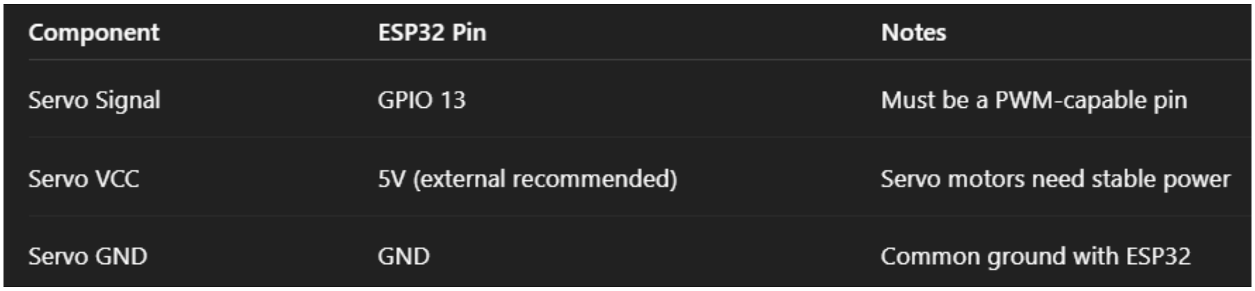
Component	Quantity
ESP32 Dev Board	1
Servo Motor (e.g., SG90)	1
External Power Source (optional)	1
Jumper Wires	1 set
Breadboard	1

3. Code: Full Listing



```
#include <Arduino.h>
#include <ESP32Servo.h>
#define SERVO_PIN 13

Servo myServo;

void setup() {
  Serial.begin(115200);
  myServo.setPeriodHertz(50);           // Standard 50Hz servo
  myServo.attach(SERVO_PIN, 500, 2400); // Attach to pin, min/max pulse width in
microseconds
  Serial.println("ESP32 Servo Ready");
}

void loop()
{
  for (int pos = 0; pos <= 180; pos++)
  {
    myServo.write(pos);
    delay(15);
  }
  delay(500);

  for (int pos = 180; pos >= 0; pos--) {
    myServo.write(pos);
    delay(15);
  }
  delay(500);
}
```

4. Hardware Wiring & Pin Configuration



Component	ESP32 Pin	Notes
Servo Signal	GPIO 13	Must be a PWM-capable pin
Servo VCC	5V (external recommended)	Servo motors need stable power
Servo GND	GND	Common ground with ESP32

5. Code Walkthrough (Line-by-Line Explanation)

🔹 **Libraries & Setup**

```
#include <Arduino.h>
#include <ESP32Servo.h>
```

- `Arduino.h`: Base functions like `delay()`, `Serial.begin()`
- `ESP32Servo.h`: Provides servo control functions for ESP32, since the standard `Servo.h` doesn’t work well on ESP32

🔹 Define Servo Pin

```
#define SERVO_PIN 13
```

- Assigns GPIO 13 for the servo signal wire

🔹 Create Servo Object

```
Servo myServo;
```

- Creates a servo object called `myServo` that you’ll control in the code

🔹 Setup Function

```
void setup() {
  Serial.begin(115200);
```

- Initializes serial communication for debugging/logging at 115200 baud rate

```
  myServo.setPeriodHertz(50);           // Standard 50Hz servo
```

- Sets the PWM frequency to 50Hz, which is standard for most hobby servo motors

```
  myServo.attach(SERVO_PIN, 500, 2400); // Attach to pin, min/max pulse width in microseconds
```

- Connects the servo to `SERVO_PIN` with:
  - `500` = minimum pulse width (µs) → corresponds to 0°
  - `2400` = maximum pulse width (µs) → corresponds to 180°

```
  Serial.println("ESP32 Servo Ready");
```

- Prints a confirmation message to the Serial Monitor

🔹 Loop Function (Main Motion Logic)

🔸 Forward Sweep (0° → 180°)

```
  for (int pos = 0; pos <= 180; pos++)
  {
    myServo.write(pos);
    delay(15);
  }
  delay(500);
```

- Starts at `0°` and increments angle by `1°` until `180°`
- `myServo.write(pos)` sets the servo position
- `delay(15)` gives the servo time to physically move
- After reaching 180°, waits 0.5 seconds

🔸 Reverse Sweep (180° → 0°)

```
  for (int pos = 180; pos >= 0; pos--) {
    myServo.write(pos);
    delay(15);
  }
  delay(500);
```

- Starts at `180°` and decrements angle to `0°`
- Same logic as above, but in reverse

6. Expected Behavior

- The servo moves smoothly from 0° to 180°, pausing for 0.5s
- Then it moves back from 180° to 0°, again pausing 0.5s
- This repeats infinitely

7. Technical Background

🔹 Servo Control with PWM

- Servo position is controlled using pulse width modulation:
  - 0° = ~500µs pulse
  - 90° = ~1500µs
  - 180° = ~2400µs
- The signal is sent every 20ms (50Hz)

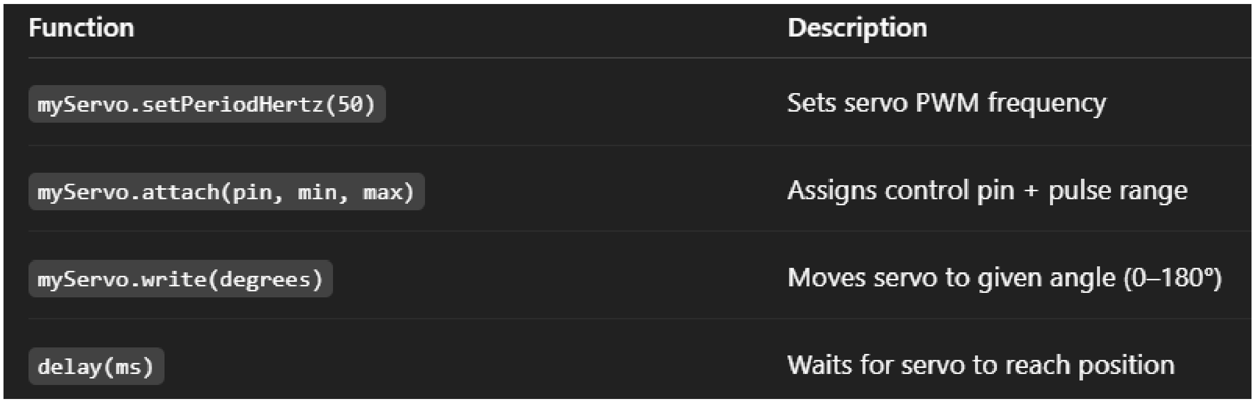
🔹 Why Use `ESP32Servo`?

- The standard Arduino `Servo` library is not optimized for ESP32
- `ESP32Servo` leverages ESP32's dedicated PWM hardware timers

8. Tips & Best Practices

- Don’t power servo directly from ESP32 — use external 5V source
- Ensure common ground between ESP32 and servo power supply
- You can change sweep speed by adjusting `delay(15)`
- Modify pulse width range (e.g., `600, 2400`) for different servo types

9. Summary Table



Function	Description
`myServo.setPeriodHertz(50)`	Sets servo PWM frequency
`myServo.attach(pin, min, max)`	Assigns control pin + pulse range
`myServo.write(degrees)`	Moves servo to given angle (0–180°)
`delay(ms)`	Waits for servo to reach position

10. Conclusion

This project shows how to:

- Control a servo motor with ESP32
- Use PWM with ESP32’s hardware timers
- Perform sweeping motion via programming
- Interface mechanical components with electronics

This is a foundational project in robotics, RC control, automation, and physical computing.