

# Hiwonder

## Arduino Development

### V1.0

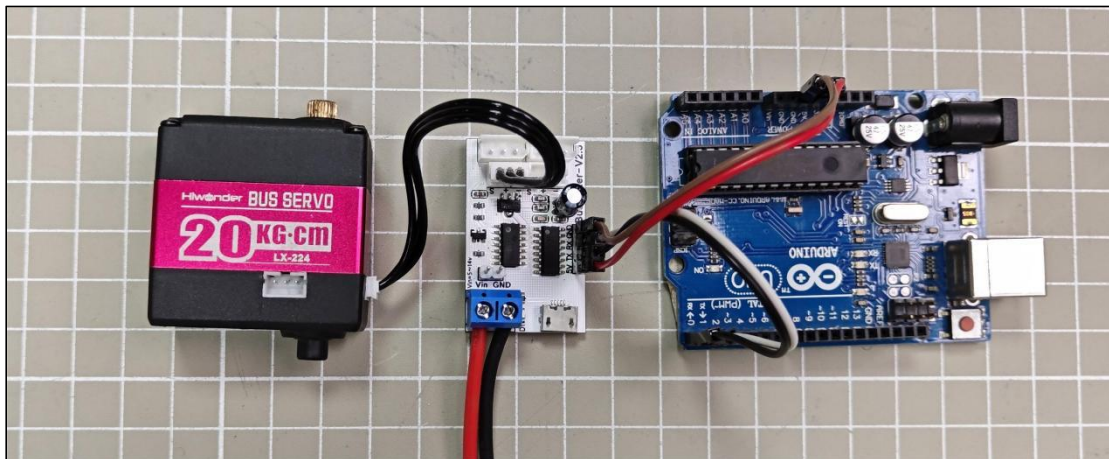
---



## 1. Getting Started

### 1.1 Wiring Instruction

This section employs Arduino UNO and a bus servo debug board for development, powered by a 11.1V 6000mAh lithium battery. Connect the bus servo to any bus interface on the bus servo debugging board, then connect the serial port of the bus servo debugging board to the serial port of Arduino (Connection method: 5V to 5V, RX to TX, TX to RX, GND to GND).



Note:

① When using our lithium battery, connect the lithium battery interface wire with red to positive (+) and black to negative (-) to the DC port.

② If the interface wire is not connected to the lithium battery, do not directly connect it to the battery interface wire to avoid a short circuit between positive and negative poles, which may cause a short circuit.

---

## 1.2 Environment Configuration

Install Arduino IDE software on PC. The software package is stored in “2. Software -> Arduino Environment Configuration”. For the detailed operations of Arduino IDE, please refer to the relevant tutorials.

## 2. Development Case

### 2.1 Case 1 Read Bus Servo Information

In this case, the servo ID, position, temperature and other information will be displayed in the terminal monitor.

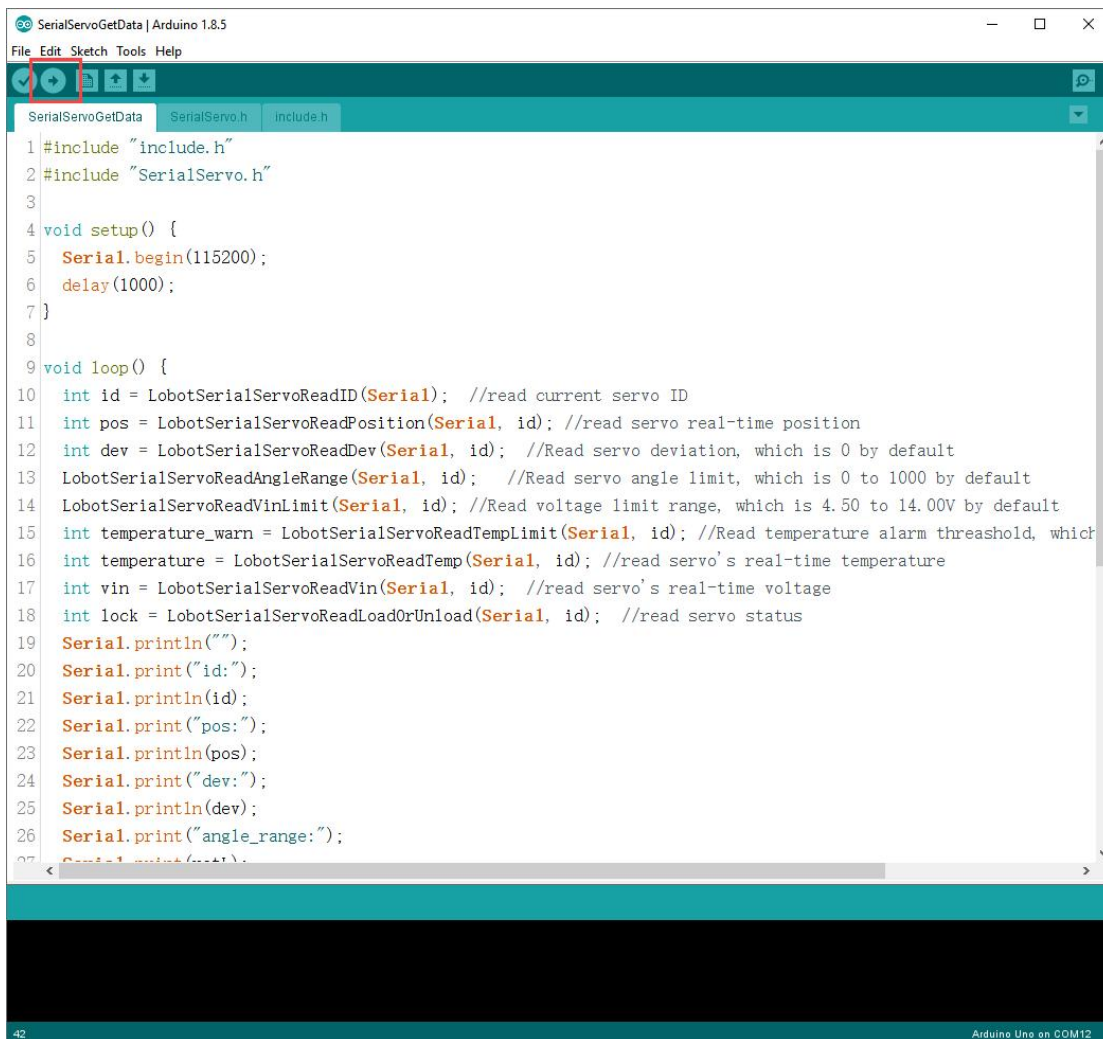
```
id:2
pos:999
dev:-24
angle_range:0-1000
vin_range:4.50-14.00 V
temperature_warn:85
temperature:29
vin:8.11
lock:0
```

### 2.1.1 Run Program

Open the program “SerialServoGetData.ino” in “1. Arduino Development/ Case 1 Read Bus Servo Information”. Connect Arduino UNO to your

computer, and then click  to download the program.

Note: If an upload failure message appears, try disconnecting the servo debugging board from the Arduino, then upload again. After the upload is complete, reconnect the servo debugging board to the Arduino.



```

SerialServoGetData | Arduino 1.8.5
File Edit Sketch Tools Help
SerialServoGetData SerialServo.h include.h
1 #include "include.h"
2 #include "SerialServo.h"
3
4 void setup() {
5   Serial.begin(115200);
6   delay(1000);
7 }
8
9 void loop() {
10  int id = LobotSerialServoReadID(Serial); //read current servo ID
11  int pos = LobotSerialServoReadPosition(Serial, id); //read servo real-time position
12  int dev = LobotSerialServoReadDev(Serial, id); //Read servo deviation, which is 0 by default
13  LobotSerialServoReadAngleRange(Serial, id); //Read servo angle limit, which is 0 to 1000 by default
14  LobotSerialServoReadVinLimit(Serial, id); //Read voltage limit range, which is 4.50 to 14.00V by default
15  int temperature_warn = LobotSerialServoReadTempLimit(Serial, id); //Read temperature alarm threshold, which
16  int temperature = LobotSerialServoReadTemp(Serial, id); //read servo's real-time temperature
17  int vin = LobotSerialServoReadVin(Serial, id); //read servo's real-time voltage
18  int lock = LobotSerialServoReadLoadOrUnload(Serial, id); //read servo status
19  Serial.println("");
20  Serial.print("id:");
21  Serial.println(id);
22  Serial.print("pos:");
23  Serial.println(pos);
24  Serial.print("dev:");
25  Serial.println(dev);
26  Serial.print("angle_range:");
27  Serial.print("angle_range:");
28
42 Arduino Uno on COM12
  
```

### 2.1.2 Performance

After the program runs, the terminal monitor will display servo status information.

```
id:2
pos:999
dev:-24
angle_range:0-1000
vin_range:4.50-14.00 V
temperature_warn:85
temperature:29
vin:8.11
lock:0
```

- ① **Id:** servo ID. Take “2” as example.
- ② **pos:** The current position of the servo. In this example, it is 999.
- ③ **dev:** servo deviation. In this example, it is -24.
- ④ **angle range:** servo limit range. In this example, it is 0-1000.
- ⑤ **voltage range:** Servo voltage range. In this example, it is 4.5~14V.
- ⑥ **temperature warn:** Servo over-temperature alarm threshold. In this example, it is 85°C.
- ⑦ **temperature:** Current temperature of the servo. In this example, it is 29°C.
- ⑧ **vin:** Current voltage value of the servo. In this example, it is 8.11V.
- ⑨ **lock:** Whether the servo is powered on. In this example, it is 0, which means it is powered off. When it is 1, it means it is powered on.

---

#### Reason for Garbled Information in Arduino IDE Serial Monitor:

Since the servo debugging board and USB share the same serial port, when the Arduino sends data to the servo debugging board, our PC can also receive the data. Therefore, the garbled information printed out is the data packet that Arduino sends to the servo debugging board, and we can ignore this information.

---

### 2.1.3 Case Program Analysis

- **Import the Required Function Package**

```
1 #include "include.h"
2 #include "SerialServo.h"
```

The two function packages are located in the same directory as the program file and mainly encapsulate various functional modules for bus servo

communication. We can use the variables and functions defined within these packages to control the servo.

- **Initialize Serial Port**

```
4 void setup() {  
5     Serial.begin(115200);  
6     delay(1000);  
7 }
```

The baud rate for bus servo communication is fixed at 115200. Before communicating with the servo, the Arduino serial port baud rate needs to be set to 115200.

- **Obtain and Print Servo Status Information**

```
int id = LobotSerialServoReadID(Serial); //read current servo ID  
int pos = LobotSerialServoReadPosition(Serial, id); //read servo real-time position  
int dev = LobotSerialServoReadDev(Serial, id); //Read servo deviation, which is 0 by default  
LobotSerialServoReadAngleRange(Serial, id); //Read servo angle limit, which is 0 to 1000 by default  
LobotSerialServoReadVinLimit(Serial, id); //Read voltage limit range, which is 4.50 to 14.00V by default  
int temperature_warn = LobotSerialServoReadTempLimit(Serial, id); //Read temperature alarm threshold, which is 85 degrees by default  
int temperature = LobotSerialServoReadTemp(Serial, id); //read servo's real-time temperature  
int vin = LobotSerialServoReadVin(Serial, id); //read servo's real-time voltage  
int lock = LobotSerialServoReadLoadOrUnload(Serial, id); //read servo status
```

By calling the above functions, you can obtain various status information of the servo. This status information includes the servo ID, position, deviation, angle range, voltage range, over-temperature alarm threshold, current temperature, voltage, and the power-on status of the servo.

```
Serial.print("id:");  
Serial.println(id);  
Serial.print("pos:");  
Serial.println(pos);  
Serial.print("dev:");  
Serial.println(dev);  
Serial.print("angle_range:");  
Serial.print(retL);  
Serial.print("-");  
Serial.println(retH);  
Serial.print("vin_range:");  
Serial.print((float)vinL/1000);  
Serial.print("-");  
Serial.print((float)vinH/1000);  
Serial.println(" V");  
Serial.print("temperature_warn:");  
Serial.println(temperature_warn);  
Serial.print("temperature:");  
Serial.println(temperature);  
Serial.print("vin:");  
Serial.println((float)vin/1000);  
Serial.print("lock:");  
Serial.println(lock);
```

After obtaining the status information, you can print out these parameters. Each piece of information will use println to print the final value and add a newline, aligning the output results.

## 2.2 Case 2 Set Bus Servo ID


This case will modify servo ID and display the new ID through the terminal monitor.

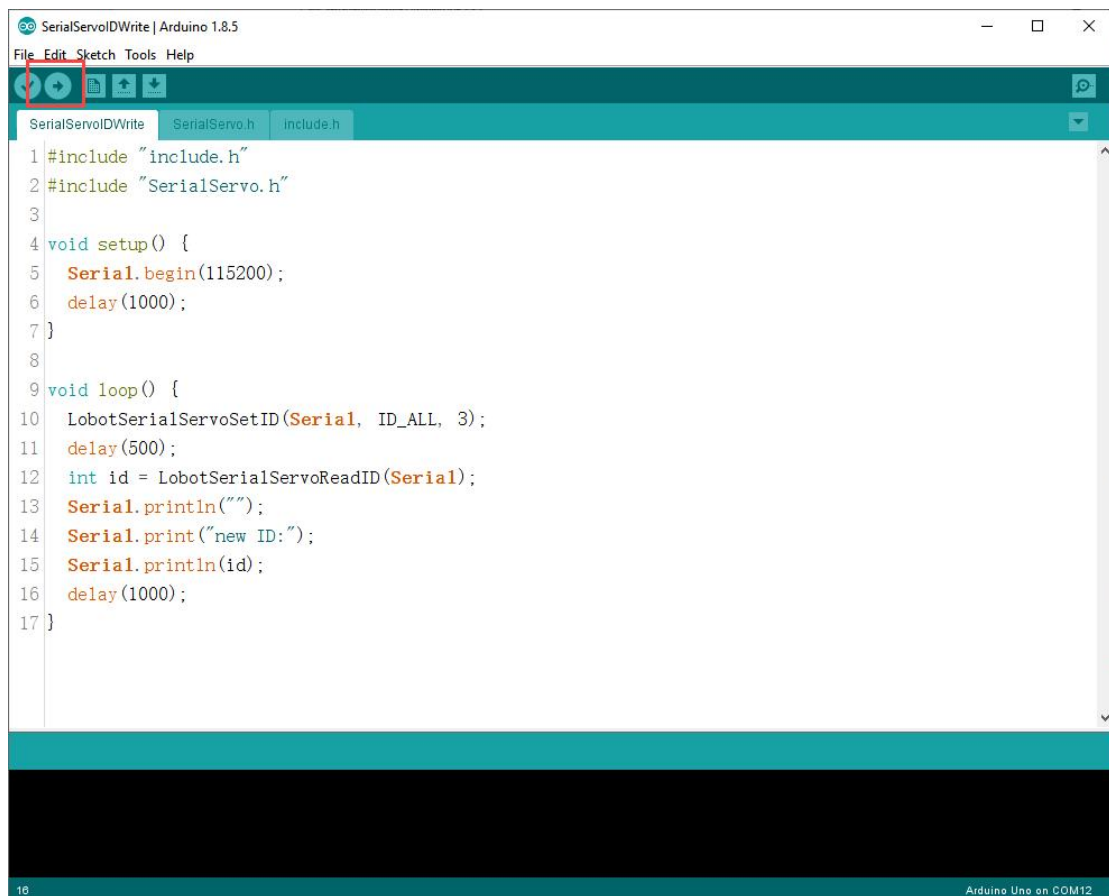





## 2.2.1 Run Program

Open the program “**SerialServoIDWrite.ino**” in “**1. Arduino Development/ Case 2 Set Bus Servo ID**”.

Connect Arduino UNO to your computer, and then click  to download the program. (Note: If an upload failure message appears, try disconnecting the servo debugging board from the Arduino, then upload again. After the upload is complete, reconnect the servo debugging board to the Arduino.)



After uploading, click  on the upper right corner. Set the baud rate to 115200 to view the information.

## 2.2.2 Performance

After the program runs, the new servo ID displays on the terminal monitor.



The specific meaning for each information is as follow:

New ID: the newest ID after modification. It is 3 in here.

### Reason for Garbled Information in Arduino IDE Serial Monitor:

Since the servo debugging board and USB share the same serial port, when the Arduino sends data to the servo debugging board, our PC can also receive the data. Therefore, the garbled information printed out is the data packet that Arduino sends to the servo debugging board, and we can ignore this information.

### 2.2.3 Case Program Analysis

- **Import the Required Function Package**

```
1 #include "include.h"
2 #include "SerialServo.h"
```

```
2 #include "SerialServo.h"
```

The two function packages are located in the same directory as the program file and mainly encapsulate various functional modules for bus servo communication. We can use the variables and functions defined within these packages to control the servo.

- **Initialize Serial Port**

```
4 void setup() {
5     Serial.begin(115200);
6     delay(1000);
7 }
```

The baud rate for bus servo communication is fixed at 115200. Before communicating with the servo, the Arduino serial port baud rate needs to be set to 115200.

- **Modify Servo ID Information**



```
LobotSerialServoSetID(Serial, ID_ALL, 3);  
delay(500);
```

By calling the **LobotSerialServoSetID()** function, you can modify the ID value of the servo connected to the bus servo debugging board to 3. In this case, the value of the **ID\_ALL** variable is 254, which represents the broadcast ID in the bus servo communication protocol and can be used to read the ID value of unknown servos.

After the setting is complete, print out the new ID.

```
int id = LobotSerialServoReadID(Serial);  
Serial.println("");  
Serial.print("new ID:");  
Serial.println(id);  
delay(1000);
```


By reading the servo ID, print out the newly modified servo ID.

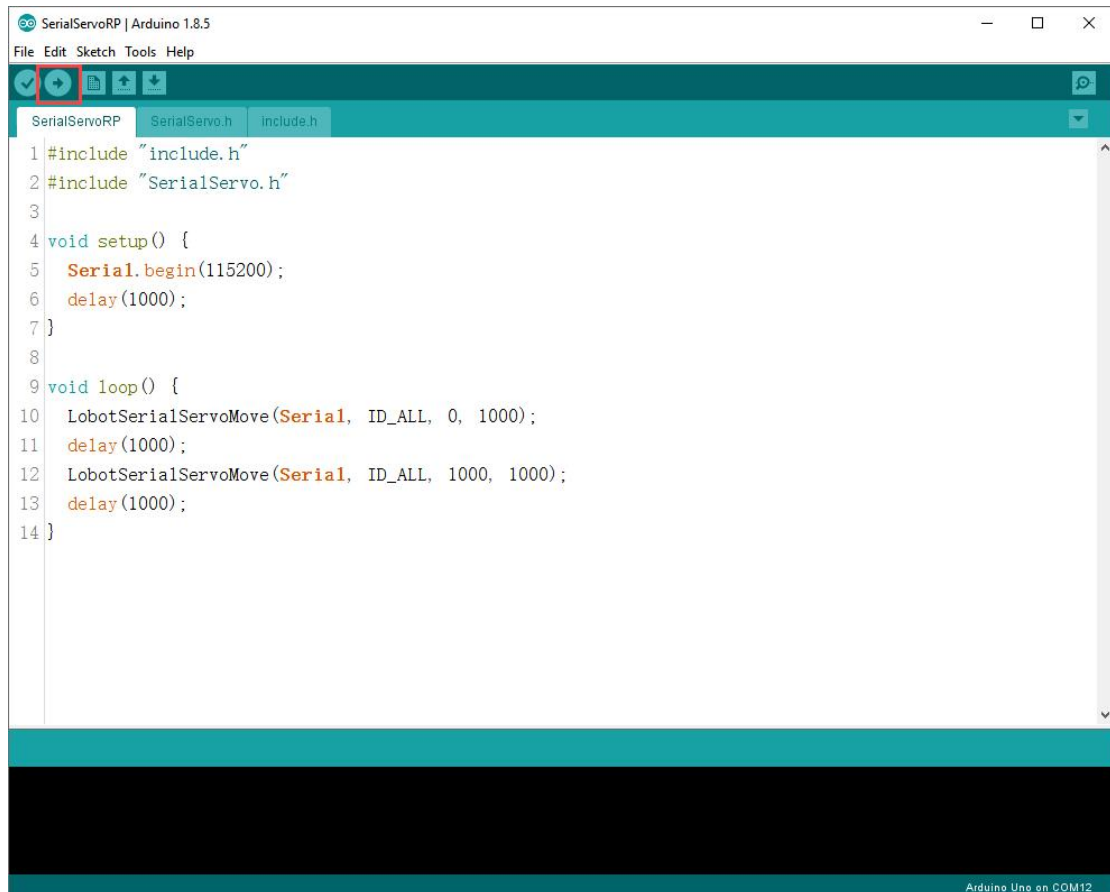
## 2.3 Case 3 Control Bus Servo to Rotate

In Case 3, Arduino UNO is used to control servo to rotate back and forth between positions 0 and 1000 at 1-second intervals

### 2.3.1 Run Program

Open the program “**SerialServoRP.ino**” in “**1. Arduino Development/ Case 3 Control Bus Servo to Rotate**”.

Connect Arduino UNO to your computer, and then click  to download the program. (Note: If an upload failure message appears, try disconnecting the servo debugging board from the Arduino, then upload again. After the upload is complete, reconnect the servo debugging board to the Arduino.)



After uploading, connect the servo control board to Arduino. Then, servo starts rotating.

### 2.3.2 Performance

After the program runs, the servo will rotate back and forth between positions 0 and 1000 with a 1-second interval.

### 2.2.3 Case Program Analysis

- **Import the Required Function Package**

```
1 #include "include.h"
2 #include "SerialServo.h"
```

The two function packages are located in the same directory as the program file and mainly encapsulate various functional modules for bus servo communication. We can use the variables and functions defined within these packages to control the servo.

- **Initialize Serial Port**

```
4 void setup() {  
5   Serial.begin(115200);  
6   delay(1000);  
7 }
```

The baud rate for bus servo communication is fixed at 115200. Before communicating with the servo, the Arduino serial port baud rate needs to be set to 115200.

### ● Control Servo to Rotate

```
9 void loop() {  
10  LobotSerialServoMove(Serial, ID_ALL, 0, 1000);  
11  delay(1000);  
12  LobotSerialServoMove(Serial, ID_ALL, 1000, 1000);  
13  delay(1000);  
14 }
```

By calling the **LobotSerialServoMove()** function, you control the rotation of the servo. The above process mainly achieves the following: the servo rotates to position 0 over 1 second, waits for 1 second, and then rotates to position 1000 over 1 second.

The servo rotation range is from 0 to 1000, corresponding to angles from 0° to 240°.


The value of the ID\_ALL variable is 254, which represents the broadcast ID in the bus servo communication protocol and can be used to control the position of servos with unknown IDs.

## 2.4 Case 4 Adjust Bus Servo Speed

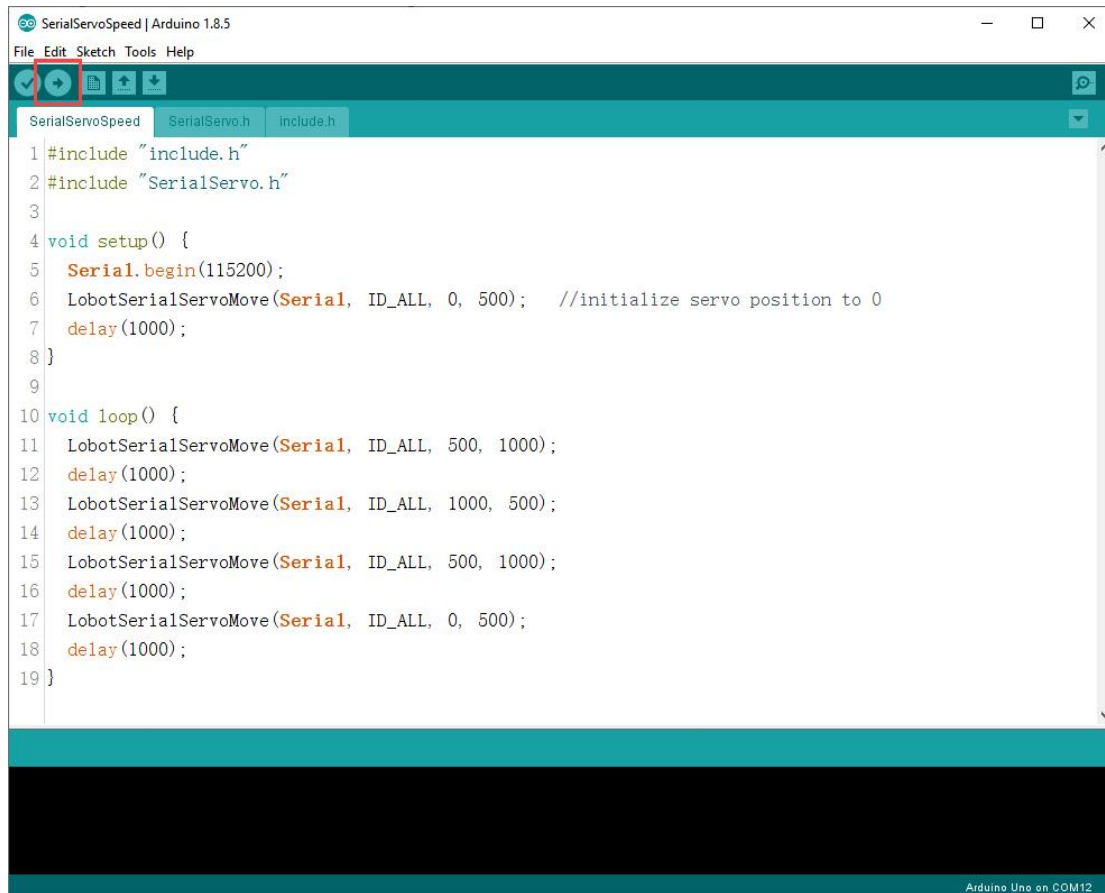
In this case, the servo is controlled by Arduino to rotate at different speeds.

### 2.4.1 Run Program

Open the program “**SerialServoSpeed.ino**” in “**1. Arduino Development/ Case 4 Adjust Bus Servo Speed**”.

Connect Arduino UNO to your computer, and then click  to download the program. (Note: If an upload failure message appears, try disconnecting the

servo debugging board from the Arduino, then upload again. After the upload is complete, reconnect the servo debugging board to the Arduino.)



After successful upload, connect the servo debugging board to Arduino, and the servo will start rotating at different speeds.

## 2.4.2 Performance

After the program runs, the servo will cycle through the following steps:

1. The servo starts from position 0.
2. It rotates to position 500 over 0.5 seconds.
3. It rotates to position 1000 over 1 second.
4. It rotates back to position 500 over 0.5 seconds.
5. Finally, it rotates back to position 0 over 1 second.

## 2.4.3 Case Program Analysis

- Import the Required Function Package

```
1 #include "include.h"
2 #include "SerialServo.h"
```

The two function packages are located in the same directory as the program file and mainly encapsulate various functional modules for bus servo communication. We can use the variables and functions defined within these packages to control the servo.

- **Initialize Serial Port**

```
void setup() {  
  Serial.begin(115200);  
  LobotSerialServoMove(Serial, ID_ALL, 0, 500); //initialize servo position to 0  
  delay(1000);  
}
```

The baud rate for bus servo communication is fixed at 115200. Before communicating with the servo, the Arduino serial port baud rate needs to be set to 115200.

- **Control Servo to Rotate at Different Speed**

```
10 void loop() {  
11   LobotSerialServoMove(Serial, ID_ALL, 500, 1000);  
12   delay(1000);  
13   LobotSerialServoMove(Serial, ID_ALL, 1000, 500);  
14   delay(1000);  
15   LobotSerialServoMove(Serial, ID_ALL, 500, 1000);  
16   delay(1000);  
17   LobotSerialServoMove(Serial, ID_ALL, 0, 500);  
18   delay(1000);  
19 }
```

By calling the **LobotSerialServoMove()** function, the servo can be controlled to rotate. The process primarily achieves the following:

The servo motor rotates to position 500 in 1 second.

After a 1-second delay, it rotates to position 1000 in 0.5 seconds.

After another 1-second delay, it rotates back to position 500 in 1 second.

Following a 1-second delay, it rotates to position 0 in 0.5 seconds.

The rotation range of the servo is 0-1000, corresponding to an angle range of 0°-240°. The last parameter specifies the time required to move to the


designated position, with smaller values indicating faster speeds. The ID\_ALL variable has a value of 254, which represents a broadcast ID in the bus servo communication protocol and can be used to control the position of servos with unknown IDs.

## **2.5 Case 5 Teaching Record Operation**

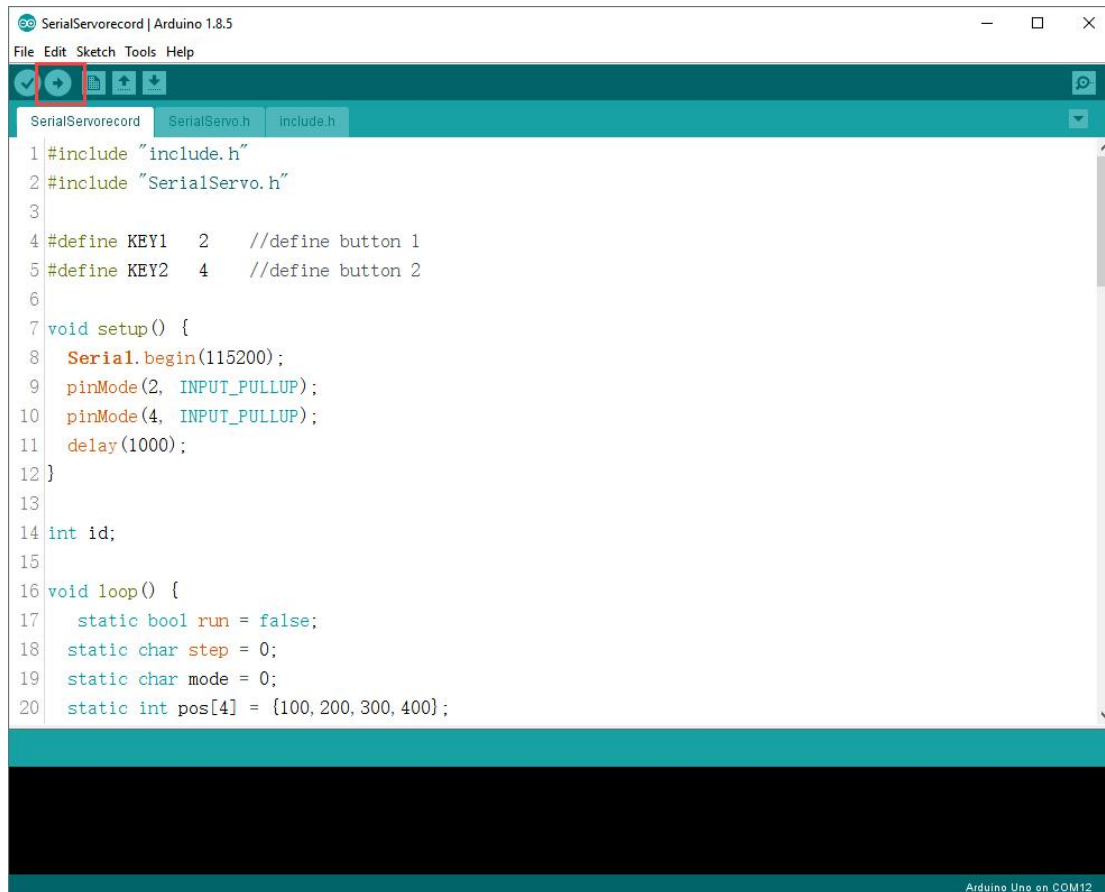
In this case, the Arduino UNO and buttons are used to control the servo. By storing positions, the servo motor can rotate to specified angles.

### **2.5.1 Run Program**

Open the program “**SerialServorecord.ino**” in “**1. Arduino Development/ Case 5 Teaching Record Operation**”.

Connect Arduino UNO to your computer, and then click  to download the program. **(Note: If an upload failure message appears, try disconnecting the servo debugging board from the Arduino, then upload again. After the upload is complete, reconnect the servo debugging board to the Arduino.)**





## 2.5.2 Performance

After the program starts, pressing button 1 will cut the power to the servo, allowing you to manually move the servo horn to any angle. Press button 2 to record the current position (up to 4 positions can be recorded; if more than 4 positions are recorded, the first position will be overwritten). After recording the positions, press button 1 again to power the servo on. While the servo is powered on, pressing button 2 will make the servo rotate sequentially to the recorded angles, starting from the first position.

## 2.5.3 Program Analysis

- Import the Required Function Package

```

1 #include "include.h"
2 #include "SerialServo.h"

```

The two function packages are located in the same directory as the program file and mainly encapsulate various functional modules for bus servo

communication. We can use the variables and functions defined within these packages to control the servo.

- **Initialize Serial Port**

```
void setup() {  
    Serial.begin(115200);  
    pinMode(2, INPUT_PULLUP);  
    pinMode(4, INPUT_PULLUP);  
    delay(1000);  
}
```

The baud rate for bus servo communication is fixed at 115200. Before communicating with the servo, the Arduino serial port baud rate needs to be set to 115200.

- **Control Servo at Different Speed**

```
void loop() {  
    static bool run = false;  
    static char step = 0;  
    static char mode = 0;  
    static int pos[4] = {100, 200, 300, 400};  
    uint16_t temp;  
    id = LobotSerialServoReadID(Serial);
```

Initialize Variables for the Program.

**pos[4]** is used to store 4 positions, making it easy to set the servo positions when controlling the servo motor.

**id** represents the current servo ID.

**run** is a flag for controlling the servo motor's rotation. When run is set to 1, the servo motor starts rotating.

**step** indicates which position the servo motor is currently operating at.

**mode** indicates the current mode of the program. **mode** set to 0 indicates control mode, and **mode** set to 1 indicates read mode.

```
24  if (mode == 0)
25  {
26      if (run)
27      {
28          LobotSerialServoMove(Serial, id, pos[step++], 500);
29          if (step == 4)
30          {
31              step = 0;
32              run = false;
33          }
34          delay(1000);
35      }
36      if (!digitalRead(KEY2))
37      {
38          delay(10);
39          if (!digitalRead(KEY2))
40          {
41              run = true;
42              step = 0;
43              delay(500);
44          }
45      }
46      if (!digitalRead(KEY1))
47      {
48          delay(10);
49          if (!digitalRead(KEY1))
50          {
51              LobotSerialServoUnload(Serial, id);
52              mode = 1;
53              step = 0;
54              delay(500);
55          }
56      }
57  }
```

When the program is in control mode, it checks the run flag variable. If run is set to 1, the servo motor will sequentially rotate to the 4 recorded positions. After completing the rotations, the run flag is reset.

If button 2 is pressed, the run flag is set to true, allowing the servo to rotate.

If button 1 is pressed, mode is set to 1, changing the current mode to input mode and cutting power to the servo .

```
58     if (mode == 1)
59     {
60         if (!digitalRead(KEY2))
61         {
62             delay(10);
63             if (!digitalRead(KEY2))
64             {
65                 pos[step++] = LobotSerialServoReadPosition(Serial, id);
66                 if (step == 4)
67                     step = 0;
68                 delay(500);
69             }
70         }
71         if (!digitalRead(KEY1))
72         {
73             delay(10);
74             if (!digitalRead(KEY1))
75             {
76                 temp = LobotSerialServoReadPosition(Serial, id);
77                 LobotSerialServoMove(Serial, id, temp, 200);
78                 mode = 0;
79                 delay(500);
80             }
81         }
82     }
83 }
84 }
```

When the program is in input mode, you can modify the stored 4 position information. When button 2 is pressed, the current servo position is recorded and stored in the pos[4] array. If the recorded position information exceeds 4 positions, it will start overwriting from the first position.

If button 1 is pressed, the servo motor is powered on and held at its current position.