

1. 初次使用SERVO DRIVER WITH ESP32

- 1.1 安装Arduino IDE
- 1.2 安装Arduino core for the ESP32
- 1.3 下载产品程序并安装依赖库
- 1.4 上传程序到机器人

2. 使用驱动板

3. 进阶功能

- 3.1 更改最大舵机ID
- 3.2 设置MAC地址
 - ESP-NOW的相关链接
- 3.3 舵机类型选择
- 3.4 WIFI: AP与STA模式
- 3.5 扩展更多的RGB-LED
- 3.6 ESP32引脚功能定义

4. 二次开发

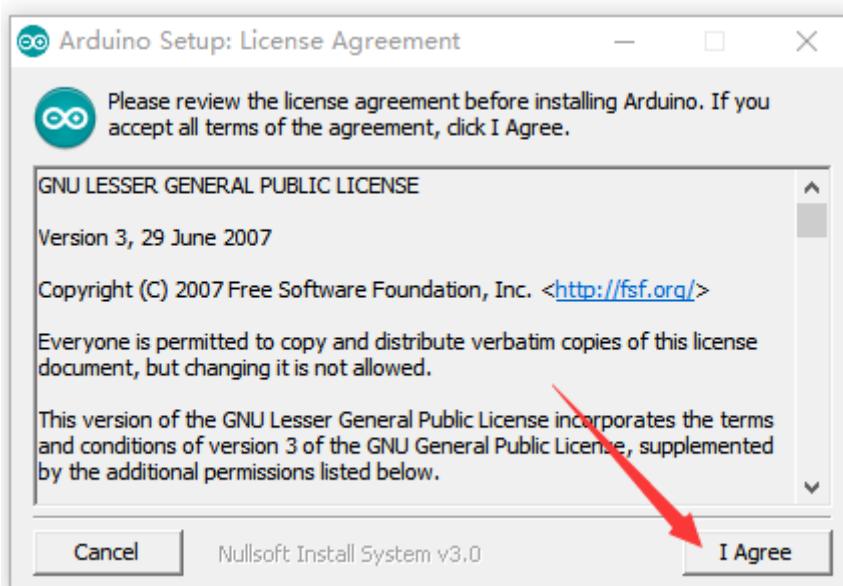
- 4.1 使用Arduino IDE例程
- 4.2 Raspberry Pi/Jetson/PC等上位机例程（Python）

5. 功能测试

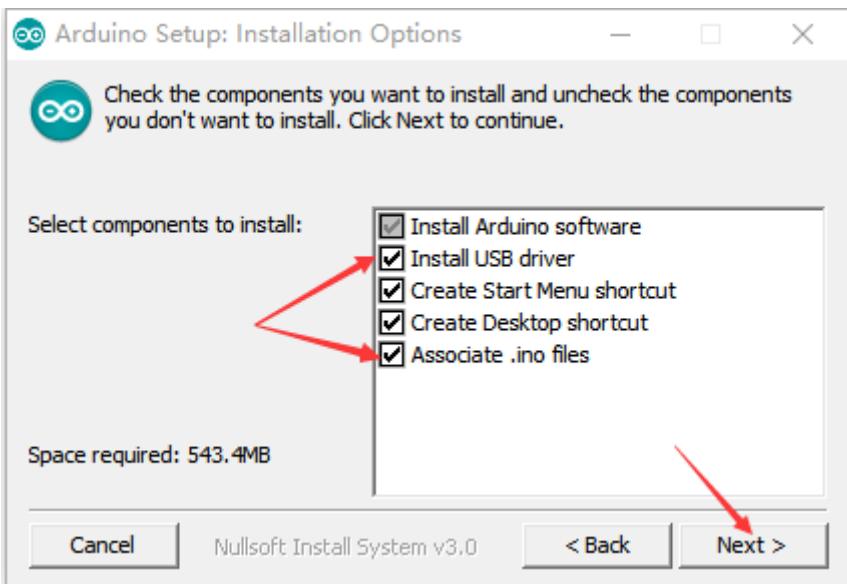
1. 初次使用SERVO DRIVER WITH ESP32

1.1 安装Arduino IDE

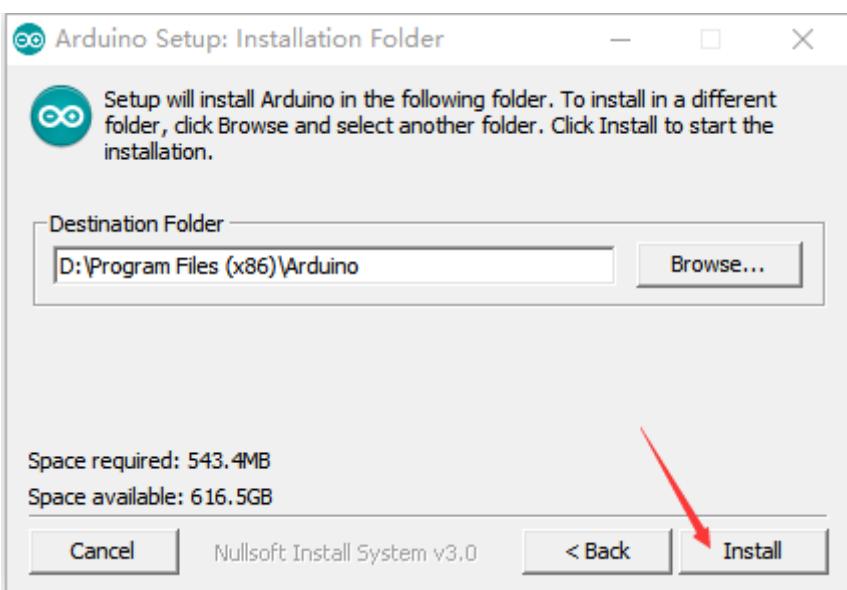
- 你可以直接点击这个[连接](#)从[Arduino.cc](#)下载Arduino IDE。
- 运行安装包。
- 点击 `I Agree`。



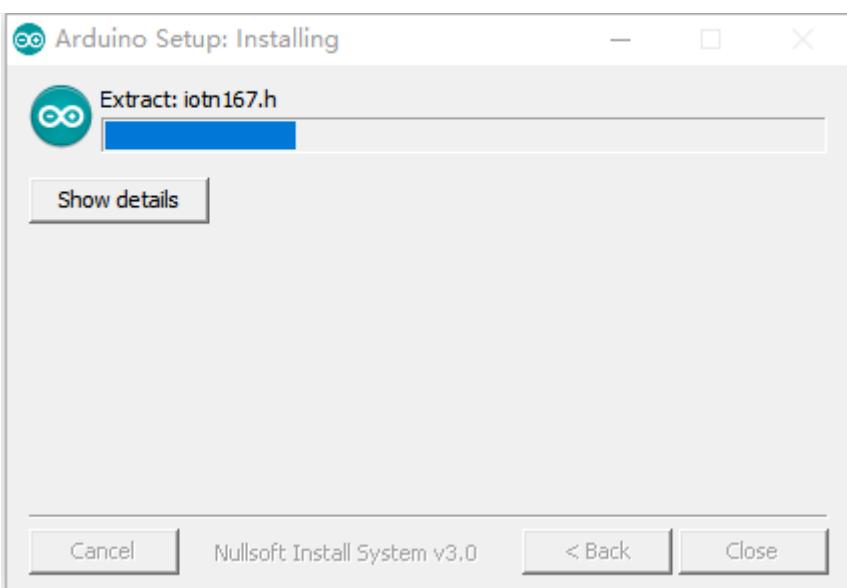
- 勾选 `Install USB driver` 和 `Associate.ino files`，然后点击 `Next`。



- 选择 Arduino IDE 的安装路径，然后点击 **Install**。



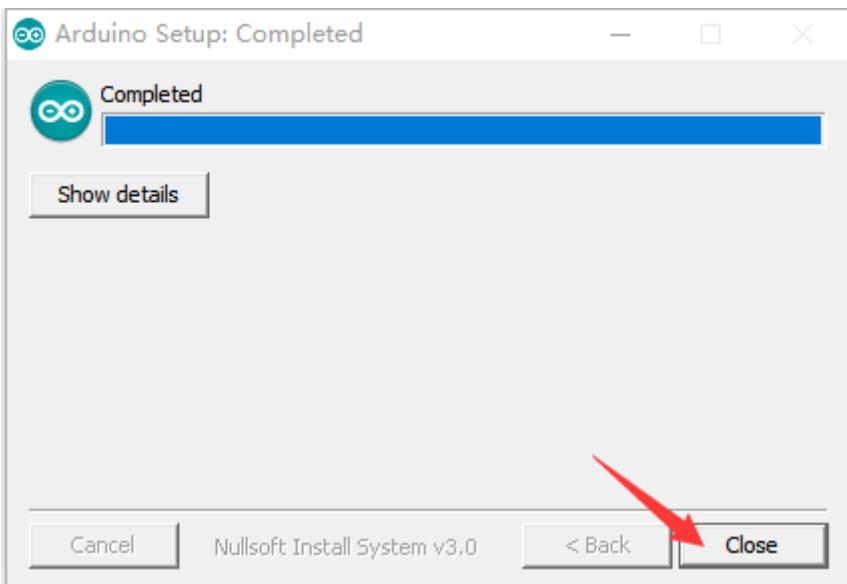
- 等待安装完成。



- 安装需要的驱动。



- Arduino IDE 安装完成。

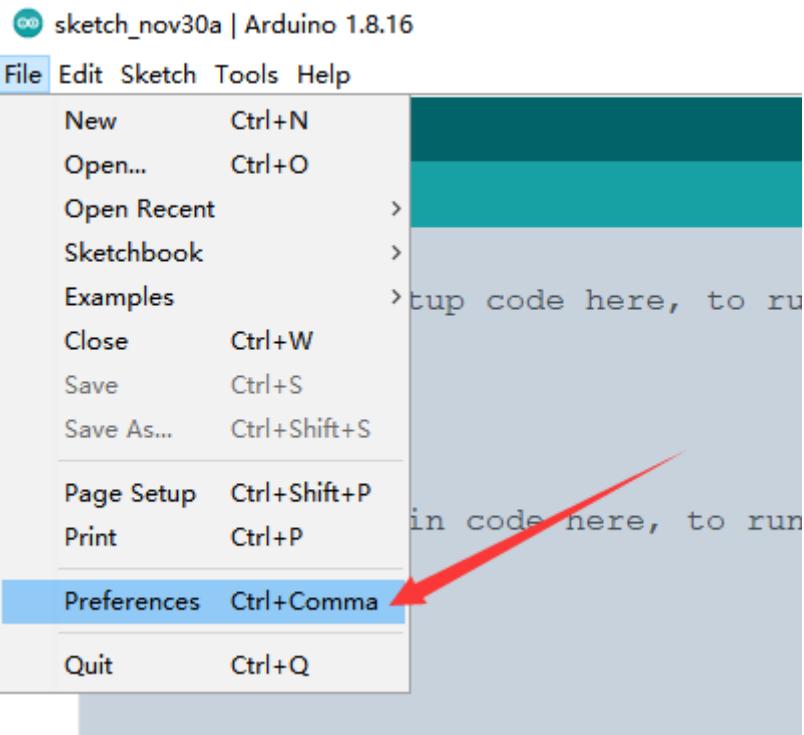


1.2 安装Arduino core for the ESP32

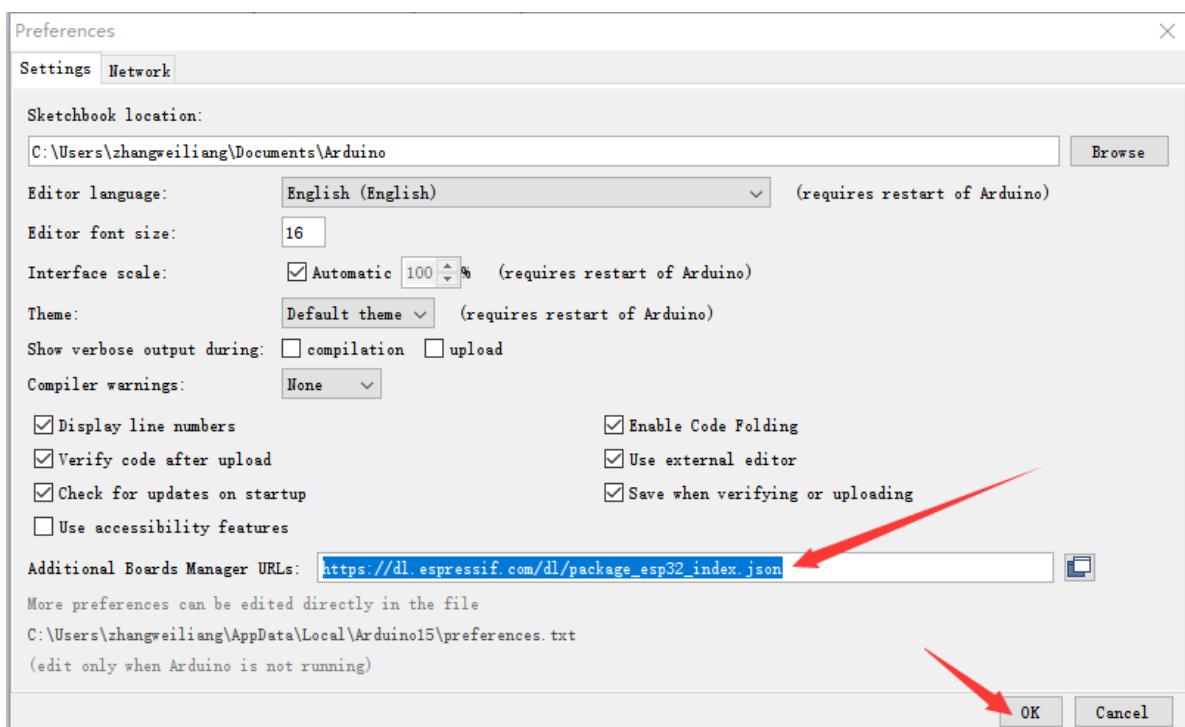
- 运行 Arduino IDE，点击 File。



- 点击 Preferences。

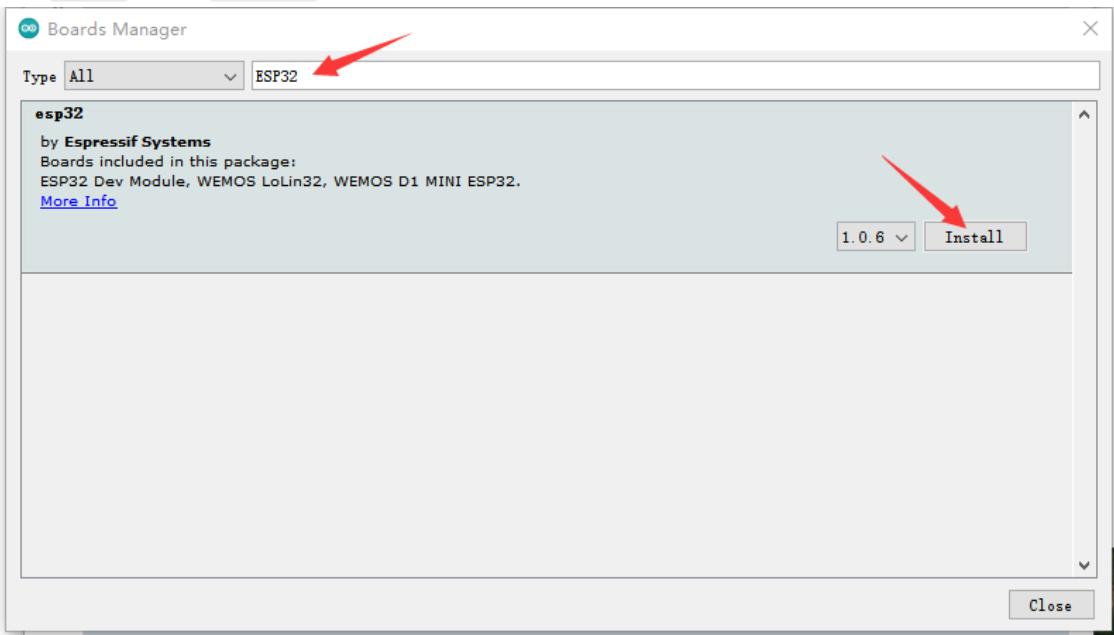


- 在 Additional Boards Manager URLs 内填写 https://dl.espressif.com/dl/package_esp32_index.json。然后点击 OK。

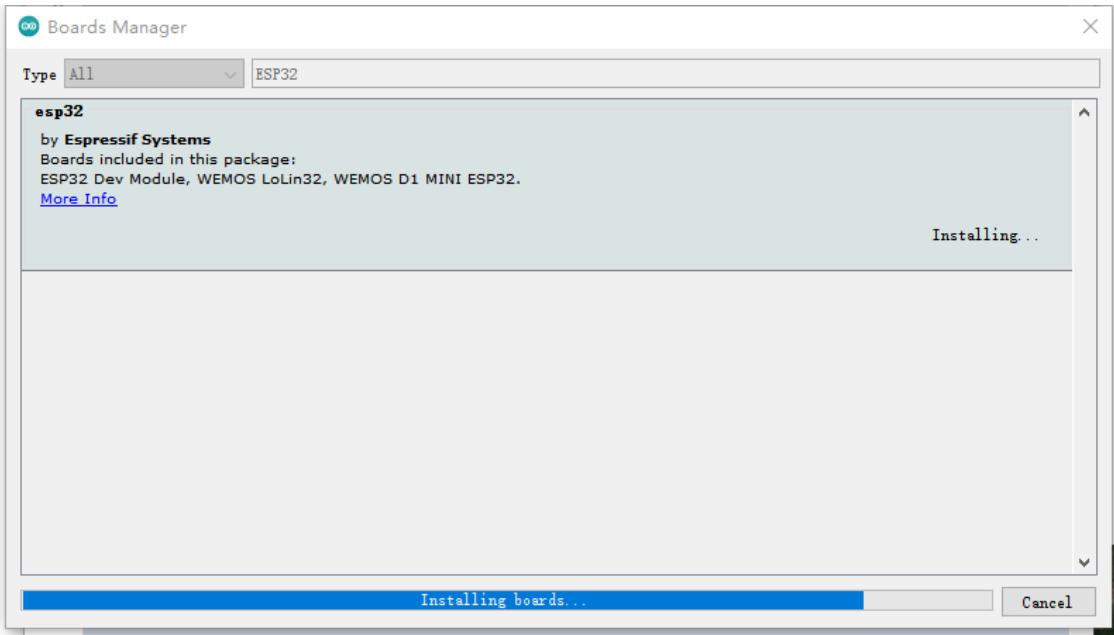


- 重启IDE, 点击 Tools > Board > Boards Manager, 打开 Boards Manager。

- 填写 `ESP32`，点击 `Install`。



- 等待安装完成。



1.3 下载产品程序并安装依赖库

通过库管理器安装的库有：

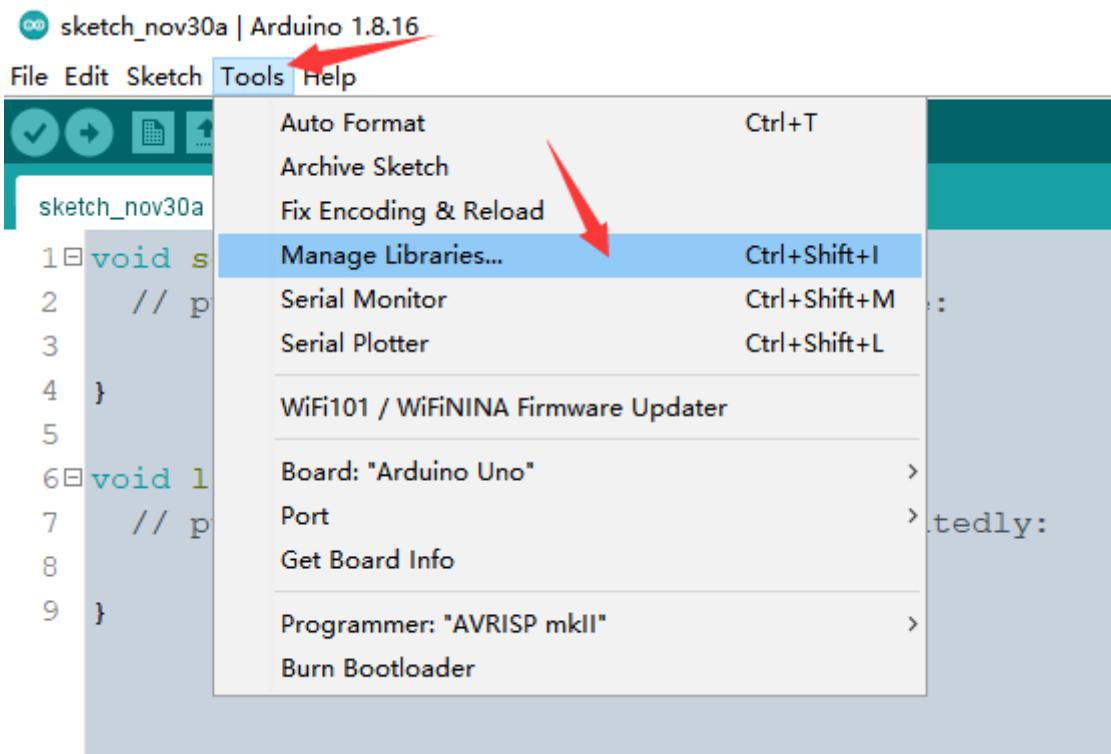
Adafruit SSD1306

Adafruit_NeoPixel

通过复制文件夹安装的库有：

SCServo

- 点击 `Tools` > `Manage Libraries`，打开库管理器。



- 依次在搜索框中输入需要通过库管理器安装的库的名称，并安装这些库。

Adafruit SSD1306

Adafruit_NeoPixel

鼠标放上去有 update 的点 update，没有 update 的选择最新的版本后点击 Install。

- 点击此[Github链接](#)，下载产品程序到本地，或者可以在微雪官网下载产品程序。
- 复制 scservo 到 \Documents\Arduino\libraries。

Adafruit_BusIO	2021/8/31 15:16
Adafruit_GFX_Library	2021/8/31 15:16
Adafruit_NeoPixel	2021/8/17 9:04
Adafruit_PWM_Servo_Driver_Library	2021/8/16 18:14
Adafruit_SSD1306	2021/11/30 13:50
ArduinoJson	2021/11/30 13:49
AsyncTCP	2021/8/18 17:13
ESP8266_and_ESP32_OLED_driver_for...	2021/8/31 15:16
ESPAsyncWebServer	2021/8/18 17:14
FabGL	2021/10/14 14:01
ICM20948_WE	2021/11/30 13:52
INA219_WE	2021/11/30 13:55
PCA9685	2021/8/16 9:06
SCServo	2021/9/13 9:46

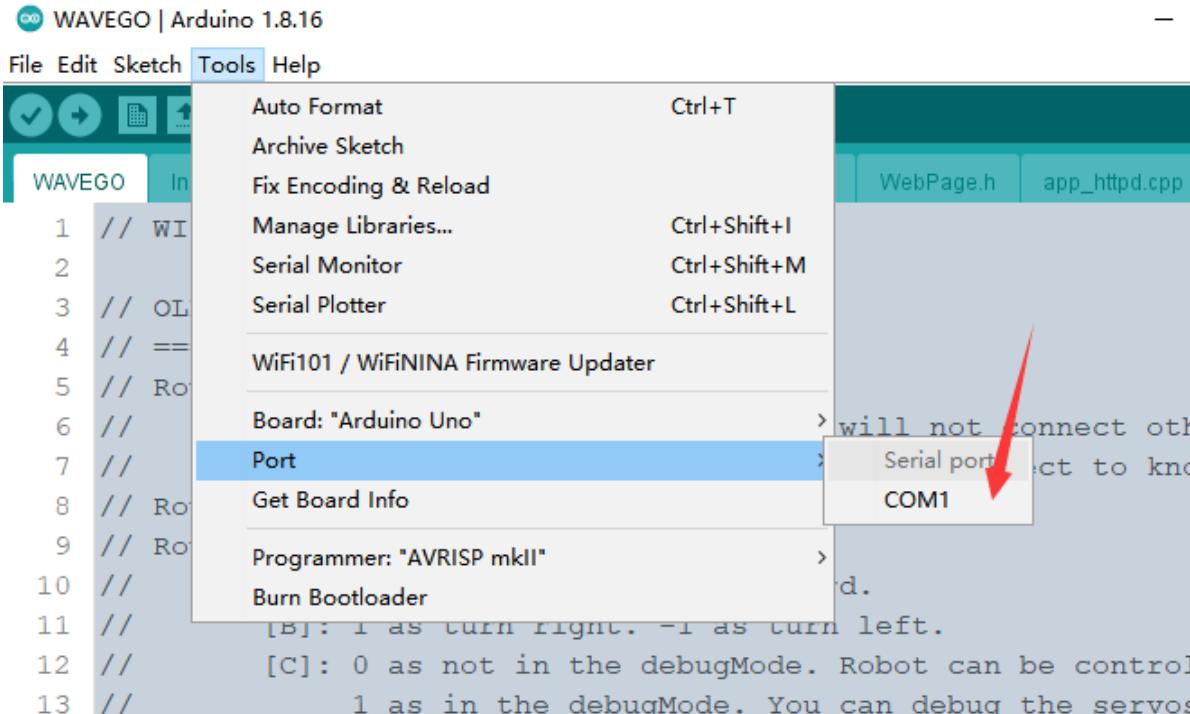
1.4 上传程序到机器人

- 双击运行 ServoDriver.ino。

BOARD_DEV.h	2021/12/3 9:43
CONNECT.h	2021/12/3 10:39
PreferencesConfig.h	2021/12/3 9:43
ServoDriver.ino	2021/12/3 14:09
STSCTRL.h	2021/12/3 10:38
WEBPAGE.h	2021/12/2 18:33

- 点击 Tools > Port 记住已有的COM Port，不用点击它。

不同电脑这里显示的COM是不一样的，记住现在已有COM。

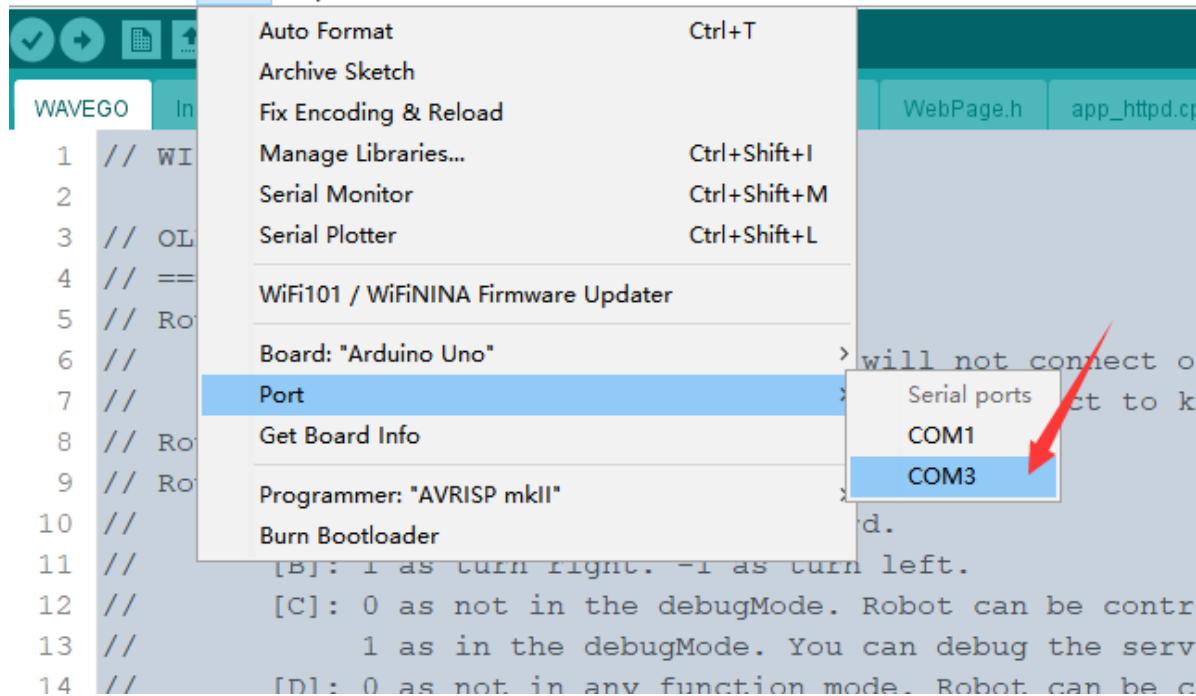


- 连接驱动板和电脑。
- 点击 Tools > Port，点击这个新的COM。

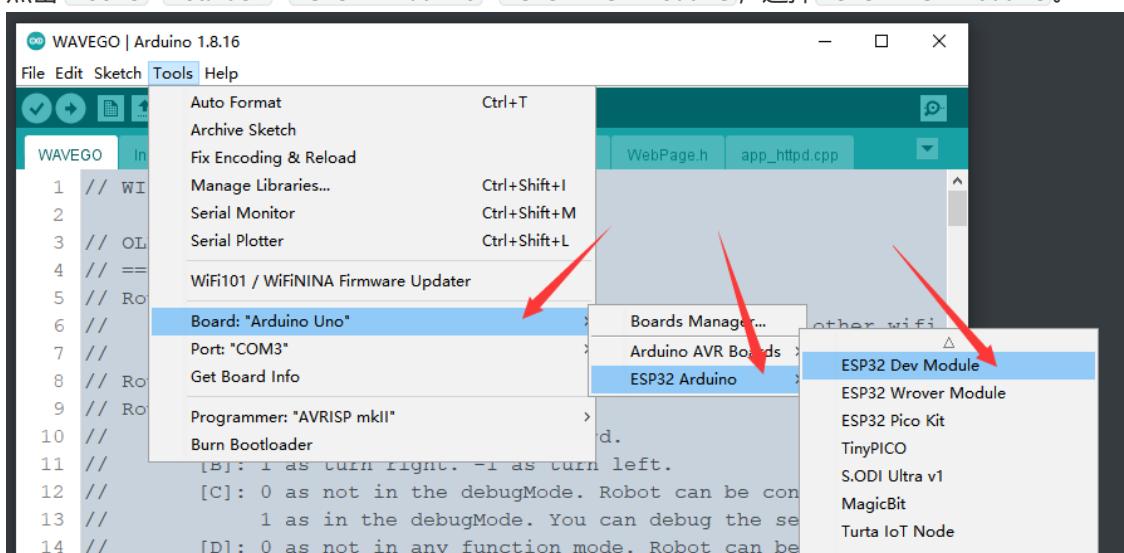
不同电脑这里新出现的COM是不一样的，点击新出现的COM。

如果没有新的COM出现，参考Q&A的USB驱动章节。

File Edit Sketch Tools Help



- 点击 Tools > Boards : > ESP32 Arduino > ESP32 Dev Module , 选择 ESP32 Dev Module。



- 其它设置如下:

Upload Speed: "921600"

CPU Frequency: "240MHz(WiFi/BT)"

Flash Frequency: "80MHz"

Flash Mode: "QIO"

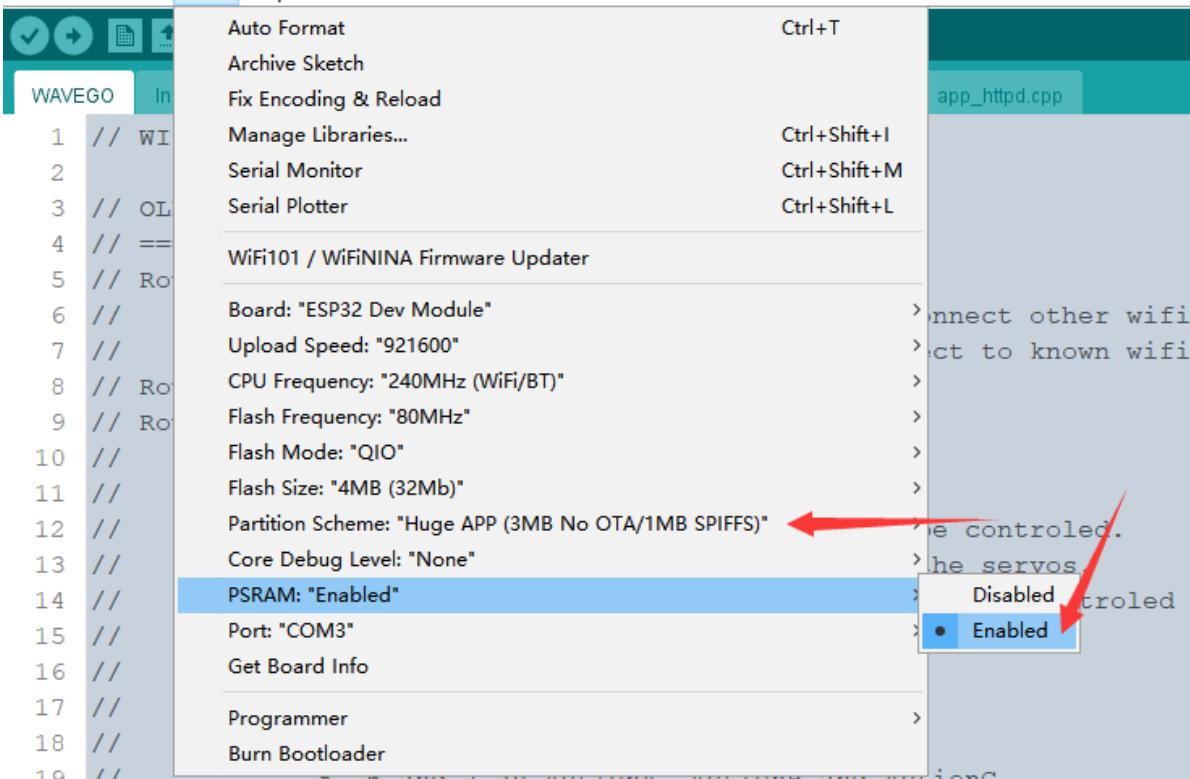
Flash Size: "4MB(32Mb)"

Partition Scheme: "Huge APP(3MB No OTA/1MB SPIFFS)"

PSRAM: "Enabled"

WAVEGO | Arduino 1.8.16

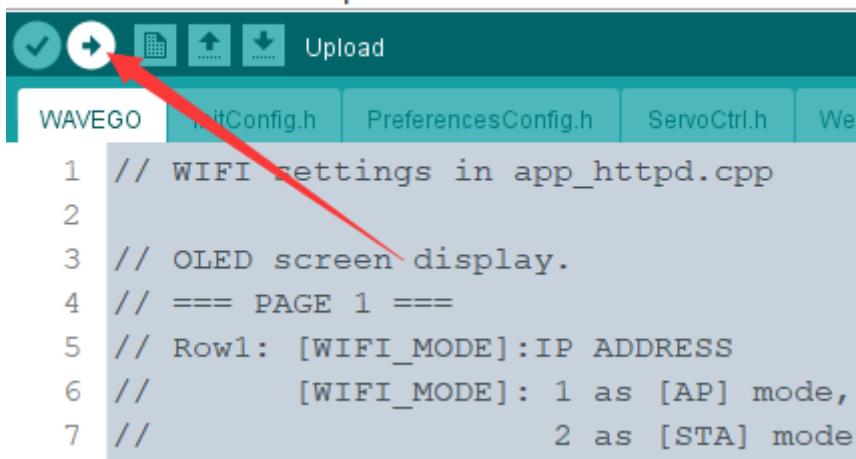
File Edit Sketch Tools Help



- 点击左上角的 upload 上传程序。

WAVEGO | Arduino 1.8.16

File Edit Sketch Tools Help



- 等待程序上传完成。

- 显示 Leaving... Hard resetting via RTS pin... 后表示已经上传成功。

```

WAVEGO | Arduino 1.8.16
File Edit Sketch Tools Help
WAVEGO InitConfig.h PreferencesConfig.h ServoCtrl.h WebCtrl.h WebPage.h app_httppd.cpp
1 // WIFI settings in app_httppd.cpp
2
3 // OLED screen display.
4 // === PAGE 1 ===
5 // Row1: [WIFI_MODE]:IP ADDRESS
6 // [WIFI_MODE]: 1 as [AP] mode, it will not connect other wifi.
7 // 2 as [STA] mode, it will connect to known wifi.
8 // Row2: [RSSI]
9 // Row3: [STATUS] [A] [B] [C] [D]
10 // [A]: 1 as forward. -1 as backward.
11 // [B]: 1 as turn right. -1 as turn left.
12 // [C]: 0 as not in the debugMode. Robot can be controled.
13 // 1 as in the debugMode. You can debug the servos.
14 // [D]: 0 as not in any function mode. Robot can be controled to move around.
15 // 1 as in steady mode. keep balancing.
16 // 2 as stayLow action.
17 // 3 as handshake action.
18 // 4 as jump action.
19 // 5, 6 and 7 as ActionA, ActionB and ActionC.
20 // 8 as servos moving to initPos, initPos is the middle angle for servos.
21 //

Done uploading.
Compressed 3072 Bytes to 119...
Writing at 0x00008000... (100 %)
Wrote 3072 bytes (119 compressed) at 0x00008000 in 0.0 seconds (effective 3071.9 kbit/s)...
Hash of data verified.

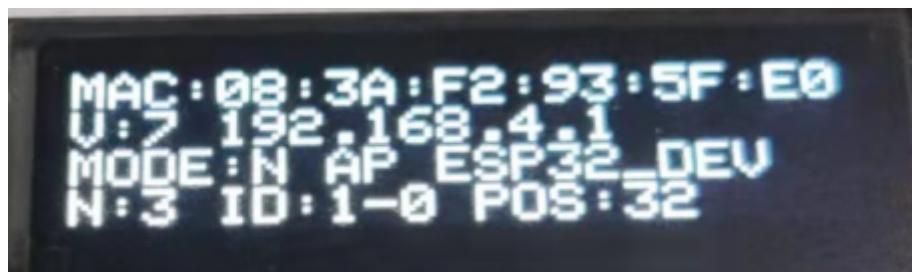
Leaving...
Hard resetting via RTS pin...

```

25 ESP32 Dev Module, Enabled, Huge APP (3MB No OTA/1MB SPIFFS), 240MHz (WiFi/BT), OIO, 80MHz, 4MB (32Mb), 921600, None on COM3

2. 使用驱动板

- 连接 sc15 Servo 舵机到驱动板，驱动板上有两个3pin的舵机接口，这两个接口是连通的，所以连接任意一个即可。
- 使用6-8.4V的DC电源供电，插上电源后驱动板会自动开机。



开机时，RGB蓝光说明驱动板开始初始化WIFI并建立WebServer，完成后驱动板会自动扫描已经连接的舵机，扫描时RGB灯为浅绿色，扫描完毕后RGB会熄灭。

开机后，OLED的第一行内容为本设备的MAC地址，该地址是独一无二的，用于ESP-NOW通信。

第二行 **V:** 是测得的电压。电压后面的是设备IP地址。

第三行 **MODE:** 是设备的角色，**N**是Normal，ESP-NOW相关功能关闭；**L**是Leader，ESP-NOW会发送当前 Active Servo 的ID和位置信息给 Follower，控制与 Follower 相连接的相同ID的舵机；**F**是 Follower，设备会通过ESP-NOW来接收从 Leader 发来的指令。

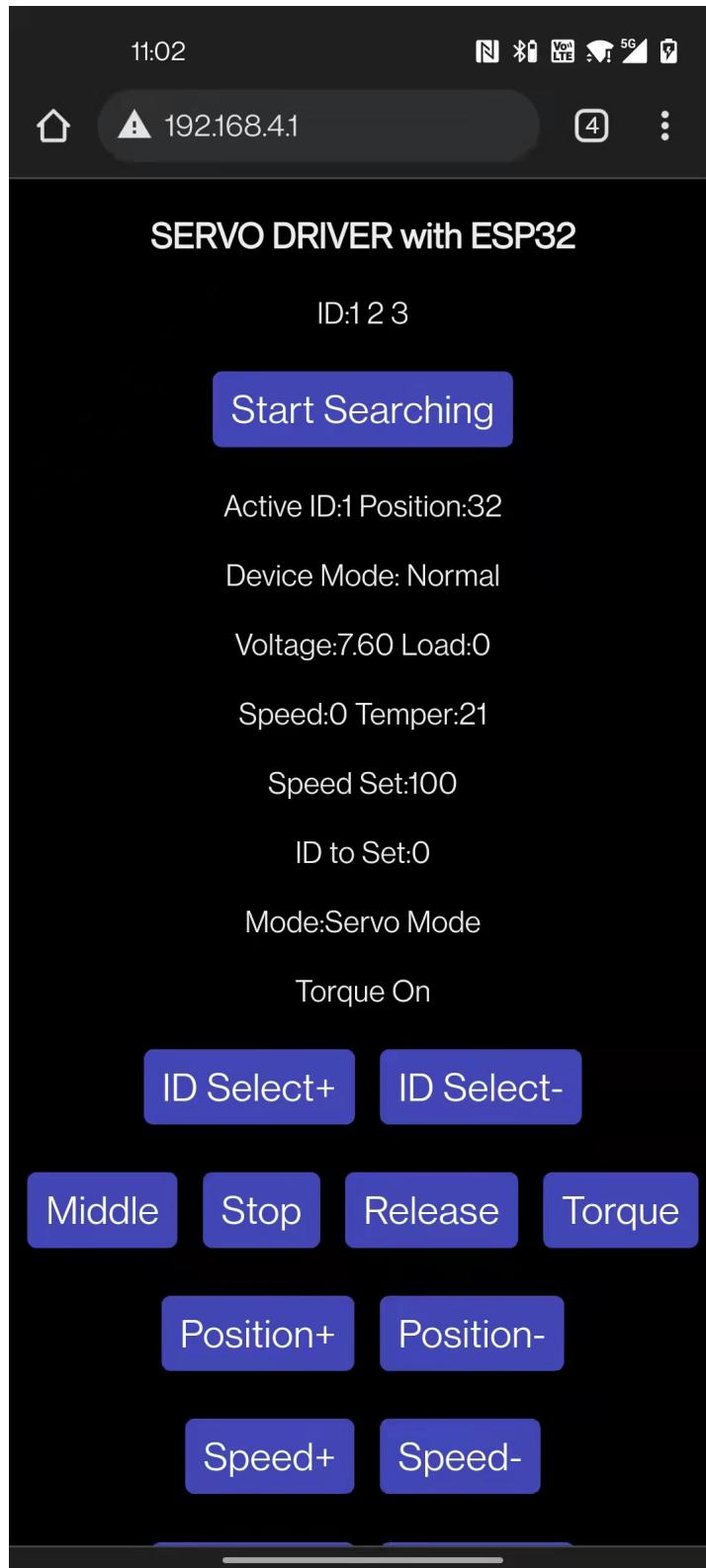
设备的角色后面为 **AP** 或 **STA**，**AP** 模式下，设备会建立一个WIFI热点；**STA** 模式下，设备会连接已知的WIFI。

AP 模式下，**AP** 后面是设备的 **SSID**；**STA** 模式下，**STA** 后面是信号强度 **RSSI**。

第一行 **N:** 是 Number，显示上次扫描时所连接的舵机数；**ID:** 是当前所选的舵机(**Active Servo**)，舵机后面 -0 说明该舵机处于舵机模式下； -3 说明该舵机处于电机模式下。**POS:** 是舵机当前所处的位置。

- 手机上下载并安装 chrome 浏览器，或者其它相同内核的浏览器。
- 默认程序中，驱动板开机后会自动建立一个WIFI热点，使用手机搜索WIFI热点。
- 默认程序中，WIFI名称为 `ESP32_DEV`，WIFI密码为 `12345678`。
- 连接成功后打开浏览器，地址栏输入 `192.168.4.1`，打开ESP32的WEB界面。

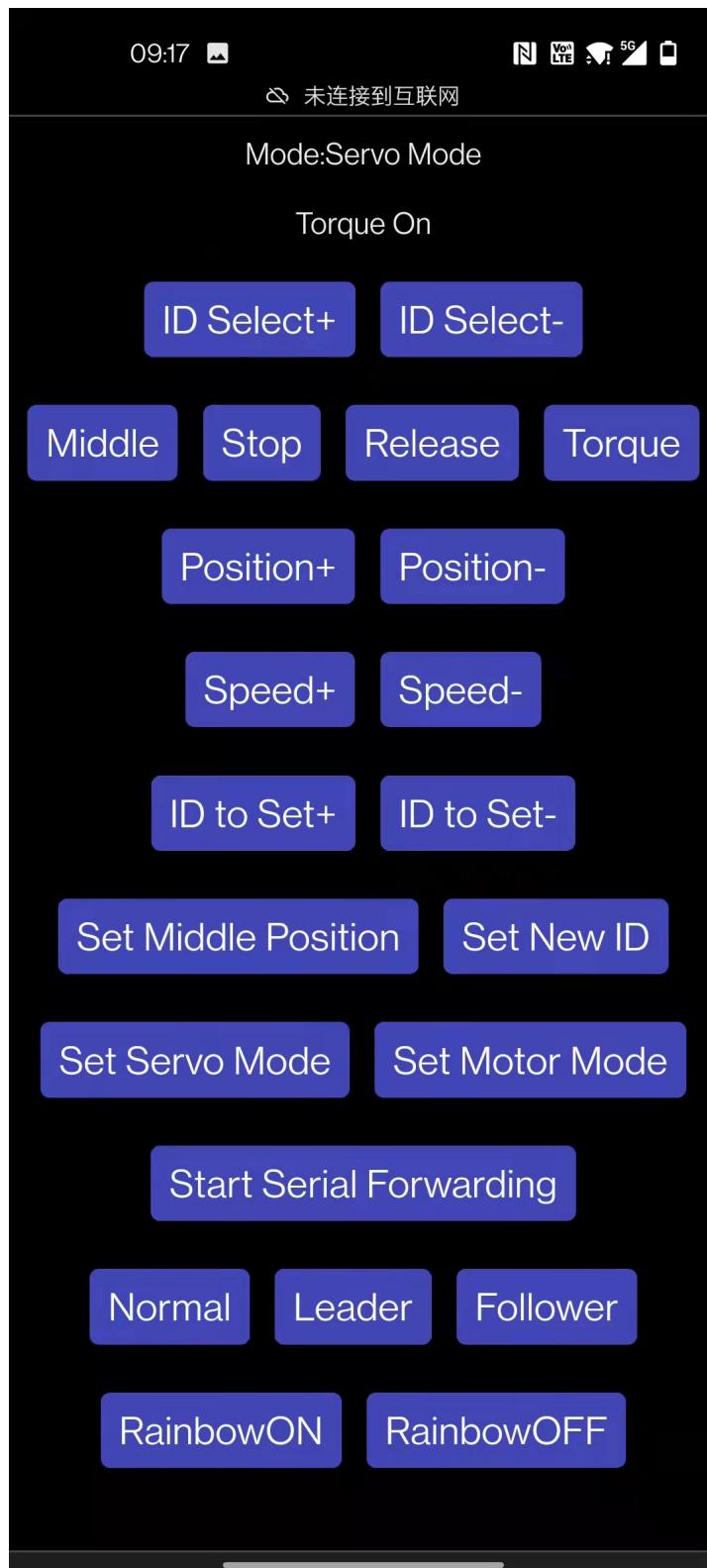
一般情况下，如果先连接舵机后上电，驱动板在开机时会扫描已经连接的舵机。如果先开机后连接舵机，则需要手动点一下手机WEB界面上的 `start searching`，然后等待一会舵机会扫描完成，包括重新设置舵机ID后也需要重新扫描舵机。



- 如果你只连接了一个舵机 `ID Select+` 和 `ID select-` 按键是没有作用的；如果你连接了多个不同 ID 的舵机，可以用这两个按键来选择舵机，当前所选择的舵机 ID 显示在 `ID:` 后面；如果你连接了多

个相同ID的舵机，很可能会ping不通导致驱动板没有舵机ID可以选择，此时需要断开多余的连接，只连接一个舵机，更改ID后在将它和其它ID的舵机串联起来。

- `Middle` 按键，按下后舵机会转动到舵机摆动范围的中间位置。
- `Stop` 按键，按下后舵机会停止转动。
- `Release` 按键，按下后舵机关闭扭矩锁，人可以用手转动舵机（由于内部存在多级减速齿轮，所以之间转动舵盘可能会很吃力，要有一个力臂才能转动）。
- `Torque` 按键，按下后打开扭矩锁，当收到外力时，舵机会尽力保持在一个位置不动。
- `Position+`，按下后舵机位置数值增加，舵机开始顺时针转动，松开后转动停止。
- `Position-`，按下后舵机位置数值减小，舵机开始逆时针转动，松开后转动停止。
- `Speed+ Speed-`，按下后会增加/减小舵机速度，更改后的速度显示在 `speed Set:` 后面，舵机的运动速度将采用这一速度。



- `ID to Set+ ID to Set- Set New ID`, 用来选取需要改的舵机ID, 显示在 `ID to Set:` 后面。例如, 当你需要将ID为3的舵机改为ID为9的舵机, 你需要先点击 `ID Select` 将显示区域 `ID: 改为3` (前提是此时被扫描的舵机里面有ID为3的舵机), 然后使用 `ID to Set` 按键将 `ID to Set:` 后面的数字改为9, 然后点击 `Set New ID`, 在弹出的对话框点击OK或确定, 这样ID3的舵机就被设置为ID9的了, 此时你需要重新点击 `Start Searching` 来搜索舵机。
- `Set Middle Position`, ST系列专用, SC系列舵机没有该功能, 该按键用于将当前所选的舵机位置设置为舵机的中间位置。
- `Set Servo Mode`, 将当前的舵机设置为舵机模式, 舵机模式也是舵机的初始默认模式, 可以转动到某一个绝对位置。
- `Set Motor Mode`, 将当前的舵机设置为电机模式, 该模式下舵机可以连续转动, 可以控制每次转动的相对位置, 可用作减速电机 (SC系列舵机) 或步进电机 (ST系列舵机)。
- `Start Serial Forwarding`, 开启后舵机会开机串口转发, 转发Type-C的UART0与用来控制舵机的UART1的数据, 这样可以使用电脑上面的FD工具来对舵机进行调试, UART0的默认波特率为 `115200`。
- `Normal`, 将当前设备设置为 `Normal`, 该模式下不会启用ESP-NOW的相关功能。
- `Leader`, 将当前设备设置为 `Leader`, 该模式下设备会通过ESP-NOW发送当前 `Active Servo` 的 `ID` 和 `Position` 给 `Follower`。
- `Follower`, 将当前设备设置为 `Leader`, 该模式下设备会通过ESP-NOW接收来自 `Leader` 的指令。
- `RainbowON RainbowOFF`, 用于开启和关闭RGB彩虹灯, 你可以在RGB灯接口上扩展更多WS2812 RGB LED。

3. 进阶功能

3.1 更改最大舵机ID

每次扫描当前连接舵机的方法是Ping每一个ID, Ping通了某一个ID则说明有这个ID的舵机与驱动板连接, 所以如果要Ping的ID数量过多会占用很长的时间, 在 `ServoDriver.ino` 中定义了扫描舵机的最大ID, `MAX_ID`, 如果你控制的舵机较多或舵机ID数值比较大, 可以通过更改这个数值来调整每次扫描要Ping的ID数。最大为253, 默认为20。

```
// set the max ID.
int MAX_ID = 20;
```

3.2 设置MAC地址

驱动板之间使用ESP-NOW进行通信, 默认程序中支持一对一的控制, 你可以根据本节后面的教程来进行一对多或多对一的通信。

ESP-NOW通信的前提是需要知道接收端的MAC地址, 驱动板上电后会自动获取设备的MAC地址并显示在OLED屏幕的第一行。

驱动板的工作模式可以设置为 `Normal Leader Follower`, 当你使用一块驱动板控制另一块时, 控制端的驱动板为 `Leader`, 另一块被控制的驱动板为 `Follower`, 具体的操作步骤如下:

- 连接 `Follower` 到电脑, 记下 `Follower` 的MAC地址。
- 打开 `ServoDriver.ino`, 将 `DEFAULT_ROLE` 改为 `2`, 上传到 `Follower` 驱动板。

```
// set the default role here.  
// 0 as normal mode.  
// 1 as leader, ctrl other device via ESP-NOW.  
// 2 as follower, can be controlled via ESP-NOW.  
#define DEFAULT_ROLE 2
```

更改DEFAULT_ROLE这一步并不是必须的，更改这个后驱动板开机后会自动处于对应的工作模式，如果不更改这里，你需要在WEB页面手动设置。相同SSID的驱动板同时以AP模式运行时，你可能会分不清哪个名称对应哪个驱动板，关于WIFI的设置可以参考[WIFI: AP与STA模式](#)章节。

- 连接Leader驱动板到电脑，打开[servoDriver.ino](#)，将broadcastAddress[]改为Follower的MAC地址，记得在每个数值前加上0x。

```
// the MAC address of the device you want to ctrl.  
uint8_t broadcastAddress[] = {0x08, 0x3A, 0xF2, 0x93, 0x5F, 0xA8};
```

以上是示例，实际中每个设备的MAC地址都是不同的。

- 将DEFAULT_ROLE改为1，上传到Follower驱动板。

```
// set the default role here.  
// 0 as normal mode.  
// 1 as leader, ctrl other device via ESP-NOW.  
// 2 as follower, can be controlled via ESP-NOW.  
#define DEFAULT_ROLE 1
```

- 两个设备连接了相同ID的舵机后，Leader的Active Servo的ID和Position会被发送给Follower，控制与其相连接的舵机转动。

例程中仅适用于舵机工作在舵机模式才可以，如果工作在电机模式则不行，SC系列舵机在连续转动模式下不能控制角度。

手机连接驱动板时要注意，两个驱动板的SSID是否为相同的名称，如果为相同的名称可能会导致连接到错误的驱动板上。

你可以参考[WIFI: AP与STA模式](#)章节更改SSID。

ESP-NOW的相关链接

[ESP-NOW Two-Way Communication Between ESP32 Boards](#)

[ESP-NOW with ESP32: Send Data to Multiple Boards \(one-to-many\)](#)

[ESP-NOW with ESP32: Receive Data from Multiple Boards \(many-to-one\)](#)

3.3 舵机类型选择

该驱动板可以控制SC系列与ST系列舵机，两个系列的舵机参数主要区别如下：

	SC系列	ST系列
数字信号范围	0-1023	0-4095
舵机转动角度	200°	360°
电压范围	6-8.4V	6-12V

其它区别可参考舵机销售页面的Datasheet，电压由DC接口的供电电压决定，DC口直接向舵机供电。

在程序上，你需要根据自己的舵机类型来调整程序，原版程序适用于SC系列舵机，如果你使用的是SC系列舵机则不需要更改，相关代码在 `STSCTRL.h` 中，默认支持SC系列舵机的程序如下：

```
// === SC Servo ===
#define CTRL_SC_SERVO
SCSCL st;
float ServoDigitalRange = 1023.0;
float ServoAngleRange = 210.0;
float ServoDigitalMiddle= 511.0;
#define ServoInitACC 0
#define ServoMaxSpeed 1500
#define MaxSpeed_X 1500
#define ServoInitSpeed 1500
int SERVO_TYPE_SELECT = 2;
int MAX_MIN_OFFSET = 30;

// === ST Servo ===
// #define CTRL_ST_SERVO
// SMS_STS st;
// float ServoDigitalRange = 4095.0;
// float ServoAngleRange = 360.0;
// float ServoDigitalMiddle= 2047.0;
// #define ServoInitACC 100
// #define ServoMaxSpeed 4000
// #define MaxSpeed_X 4000
// #define ServoInitSpeed 2000
// int SERVO_TYPE_SELECT = 1
```

如果你需要控制ST系列舵机，需要注释掉SC舵机的部分，并将ST系列舵机的部分取消注释，更改后的代码如下：

```
// === SC Servo ===
// #define CTRL_SC_SERVO
// SCSCL st;
// float ServoDigitalRange = 1023.0;
// float ServoAngleRange = 210.0;
// float ServoDigitalMiddle= 511.0;
// #define ServoInitACC 0
// #define ServoMaxSpeed 1500
// #define MaxSpeed_X 1500
// #define ServoInitSpeed 1500
// int SERVO_TYPE_SELECT = 2;
// int MAX_MIN_OFFSET = 30;
```

```
// === ST Servo ===
#define CTRL_ST_SERVO
SMS_STS st;
float ServoDigitalRange = 4095.0;
float ServoAngleRange = 360.0;
float ServoDigitalMiddle= 2047.0;
#define ServoInitACC 100
#define ServoMaxSpeed 4000
#define MaxSpeed_X 4000
#define ServoInitSpeed 2000
int SERVO_TYPE_SELECT = 1
```

3.4 WIFI: AP与STA模式

AP模式下，设备会建立一个WIFI热点，可以通过手机连接这个热点来控制/调试舵机。

STA模式下，设备会连接一个已知的WIFI热点。

你可通过更改 `servodriver.ino` 内的代码来自定义AP模式下驱动板建立的WIFI热点的SSID和密码：

```
// WIFI_AP settings.
const char* AP_SSID = "ESP32_DEV";
const char* AP_PWD = "12345678";
```

你可通过更改 `servodriver.ino` 内的代码来自定义STA模式下驱动板要连接的WIFI的SSID和密码：

```
// WIFI_STA settings.
const char* STA_SSID = "OnePlus 8";
const char* STA_PWD = "40963840";
```

- 设置AP模式，设备代码中默认为AP模式，更改 `servodriver.ino` 内的 `DEFAULT_WIFI_ROLE` 为 1。

```
// set the default wifi mode here.
// 1 as [AP] mode, it will not connect other wifi.
// 2 as [STA] mode, it will connect to know wifi.
#define DEFAULT_WIFI_MODE 1
```

- 设置STA模式，更改 `servodriver.ino` 内的 `DEFAULT_WIFI_ROLE` 为 2。

```
// set the default wifi mode here.
// 1 as [AP] mode, it will not connect other wifi.
// 2 as [STA] mode, it will connect to know wifi.
#define DEFAULT_WIFI_MODE 2
```

3.5 扩展更多的RGB-LED

驱动板背面有一个 `PH1.25-3P母座`，三个针脚的旁边有丝印标注每个针脚的功能，分别为 `5V GND OUT`，`OUT` 与其它 `WS2812 5V` 的 `IN` 连接，用来扩展更多的RGB-LED灯珠。

驱动板上有两颗板载RGB LED，控制它们的代码在 `RGB_CTRL.h` 中，编号分别为 0、1，如果你扩展了更多的灯珠，则需要修改 `ServoDriver.h` 中的 `NUMPIXELS` 的数值。

```
#define NUMPIXELS 10
```

3.6 ESP32引脚功能定义

- `GPIO 23` 用于控制 RGB LED。

```
// the GPIO used to control RGB LEDs.  
// GPIO 23, as default.  
#define RGB_LED 23
```

- `GPIO 18` 作为 RX，`GPIO 19` 作为 TX，与舵机控制电路进行通信。

```
// the uart used to control servos.  
// GPIO 18 - S_RXD, GPIO 19 - S_TXD, as default.  
#define S_RXD 18  
#define S_TXD 19
```

- 用于舵机控制的 UART 的波特率为 1000000。

```
void servoInit(){  
    Serial1.begin(1000000, SERIAL_8N1, S_RXD, S_TXD);  
    st.pSerial = &Serial1;  
    while(!Serial1) {}  
    ...
```

- 用于下载电路和上位机通信的 UART 的波特率为 115200。

```
void setup() {  
    Serial.begin(115200);  
    while(!Serial) {}  
    ...
```

- `GPIO 21` 作为 SDA，`GPIO 22` 作为 SCL，用于控制OLED屏幕。

```
// the IIC used to control OLED screen.  
// GPIO 21 - S_SDA, GPIO 22 - S_SCL, as default.  
#define S_SCL 22  
#define S_SDA 21
```

4. 二次开发

- Example 文件夹内包含了Arduino IDE、Jetson（Python语言）、Raspberry Pi（Python语言）的使用例程。
- 你可以直接在该驱动板上面使用Arduino IDE的例程做二次开发。
- 如果你想使用Jetson或Raspberry Pi来控制该驱动板，你有两种方式可以选择：
 - 使用Jetson或Raspberry Pi直接控制驱动板，我们所提供的例程适用于该方法。

- 将驱动板当作下位机来使用，其中进行连杆逆解等执行类运算，上位机负责决策类运算，我们后续会在具体的产品中提供相关例程。

4.1 使用Arduino IDE例程

Arduino IDE的例程位于 `examples\arduinoIDE\`，全部在 `SERVO DRIVER with ESP32` 上面测试通过，不需要更改任何代码。

舵机的供电来自于DC接口，所以测试时DC接口要与电源连接。

SC系列舵机的例程在 `SCSCL` 文件夹内。

ST系列舵机的例程在 `STSCL` 文件夹内。

- `SCSCL\Broadcast`: 广播控制所有舵机，测试时舵机会以最大的速度大范围摆动，不要将舵机与其它结构连接。
- `SCSCL\Ping`: 用来测试舵机是否准备就绪，通过更改 `TEST_ID` 来设置需要查看的舵机ID，测试时打开串口监视器查看舵机状态。

```
int TEST_ID = 3;
```

- `SCSCL\ProgramEprom`: 用于更改舵机的ID，将ID为 `ID_ChangeFrom` 的舵机ID改为 `ID_Chango`。

```
ID_ChangeFrom = 1;  
ID_Chango = 2;
```

- `SCSCL\RegWritePos`: 用于异步写的例程，先分别设置每个舵机的目标位置、速度，然后调用以下函数统一开始运动。

```
RegwriteAction();
```

- `SCSCL\SyncWritePos`: 用于同步写的例程，控制多个舵机一起运动。
- `SCSCL\WritePos`: 用于控制单独某一个舵机运动。
- `SCSCL\FeedBack`: 用于获取舵机的位置、速度、负载、电压、温度、移动状态。
- `STSCL\CalibrationOfs`: 用于设置舵机中位的例程，调用以下函数后，舵机当前的位置会作为舵机的中间位置。

```
st.CalibrationOfs(ID);
```

- `STSCL\FeedBack`: 用于获取舵机的位置、速度、负载、电压、温度、移动状态。
- `STSCL\Ping`: 用来测试舵机是否准备就绪，通过更改 `TEST_ID` 来设置需要查看的舵机ID，测试时打开串口监视器查看舵机状态。

```
int TEST_ID = 3;
```

- `STSCL\ProgramEprom`: 用于更改舵机的ID，将ID为 `ID_ChangeFrom` 的舵机ID改为 `ID_Chango`。

```
ID_ChangeFrom = 1;  
ID_Chango = 2;
```

- STSCL\RegWritePos：用于异步写的例程，先分别设置每个舵机的目标位置、速度，然后调用以下函数统一开始运动。

```
RegWriteAction();
```

- STSCL\SyncWritePos：用于同步写的例程，控制多个舵机一起运动。
- STSCL\WritePos：用于控制单独某一个舵机运动。

4.2 Raspberry Pi/Jetson/PC等上位机例程（Python）

Python语言的舵机控制例程位于 examples\python\ 中。

当你使用这个例程来直接控制舵机时，需要开启驱动板的串口透传模式，可以通过WEB端操作，点击 Start Serial Forwarding 按键开启串口透传，为了方便，你也可以更改 servodriver.ino 中的 SERIAL_FORWARDING 为 true，这样驱动板开机后会自动进入到串口透传模式。

当驱动板处于串口透传模式时，使用波特率 115200 与驱动板通信。

- ping.py：ping命令例子
- read_write.py：普通读写例子
- sync_write.py：同步写例子
- sync_read_write.py：同步写与同步读例子

当使用例程时，需要根据自己的实际情况改写一些代码，比如USB设备名称。例如Windows：“COM1”，Linux：“/dev/ttyUSB0”。

```
DEVICENAME = '/dev/ttyUSB0'      # Check which port is being used on your  
controller
```

protocol_end 用于选择舵机类型，ST系列舵机为 0，SC系列舵机为 1。

```
protocol_end = 1  # SCservo bit end(STS/SMS=0, SCS=1)
```

5. 功能测试

- 双击运行 servodriver.ino。
- 连接驱动板，点击Arduino IDE左上角的箭头来上传程序到驱动板。
- 上传成功后断开驱动板，连接一个舵机，使用DC口供电（6-8.4V范围内的电源都可以），上电后屏幕亮起，扫描舵机的同时RGB灯发出浅绿色光。
- 使用手机搜索WIFI，名称 ESP32_DEV，密码 12345678。
- 打开手机上的Chrome浏览器（或其它Chrome内核的浏览器），访问 192.168.4.1。
- 按 Position+ 或 Position- 来转动舵机，如果舵机转动说明测试成功。

以上测试步骤所测试的内容包括：供电、通信、舵机控制、OLED、RGB LED、自动下载电路
手机附近需要保证只有一个驱动板处于开机状态，否则相同的WIFI名称不容易判断所连接的是哪个驱动板。

