
SunFounder esp-4wd

www.sunfounder.com

Jul 12, 2023

CONTENTS

1	About This Kit	1
1.1	Introduction to ESP32 RDP	2
1.2	Components List	5
1.3	For Arduino User	6
1.4	For MicroPython User	51
1.5	FAQ	96
1.6	Thank You	97
2	Copyright Notice	99

ABOUT THIS KIT

The ESP-4WD is an ESP32 RDP based, cool, robot car kit that everyone can have.

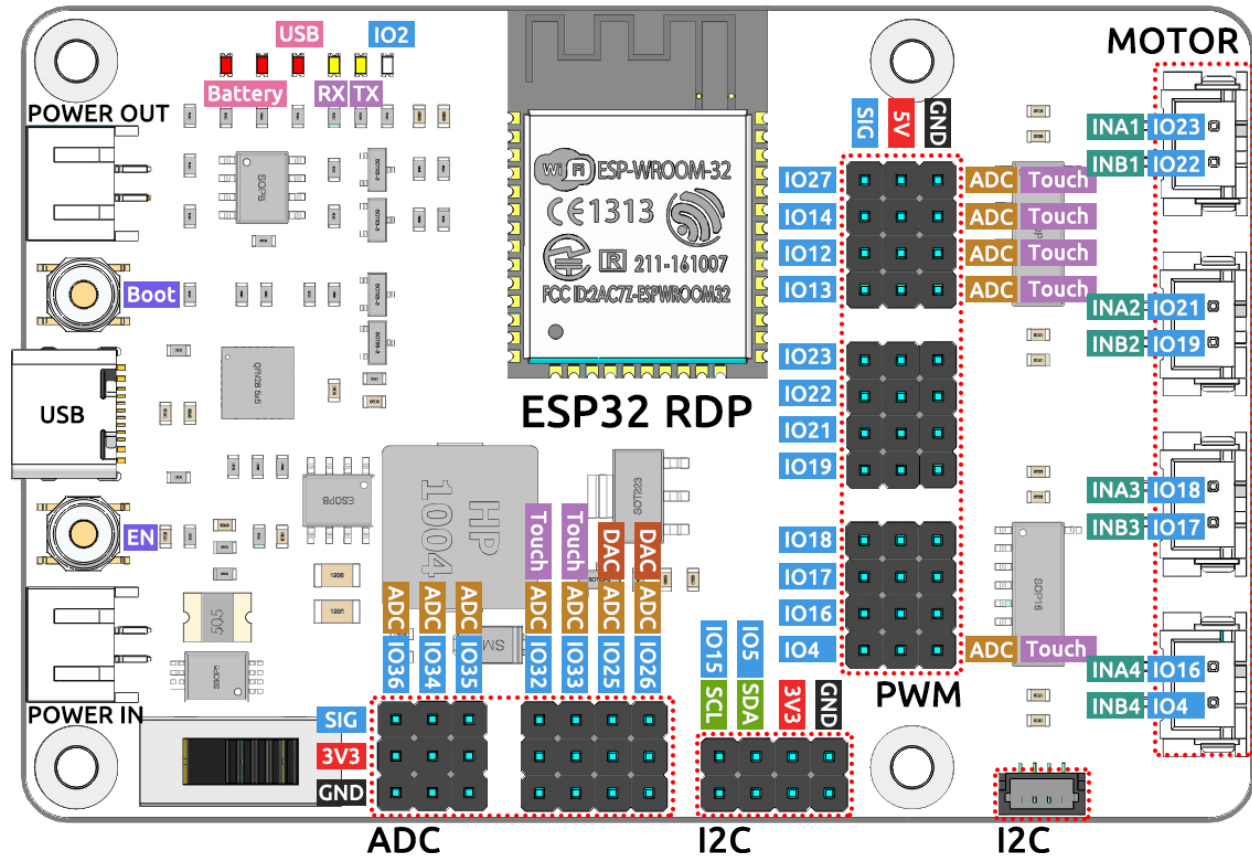
Equipped with gray sensor module and ultrasonic module, it can perform line tracking, cliff detection, follow and obstacle avoidance functions. The RGB boards assembled at the bottom and rear of the car make it the coolest spirit in the dark.

In order to make it easier for you to get started with ESP-4WD car, we provide both Arduino code and Python code. You can choose different language according to your preference.

In addition you can use an APP, the SunFounder Controller, to visually control or access the individual data of the car.

1.1 Introduction to ESP32 RDP

1.1.1 ESP32 RDP's pins



ESP32 Robot Development Platform (RDP) is a general-purpose WIFI+BT+BLE MCU module using Espressif's ESP32-WROOM-32D module.

It has industry leading integration of WIFI+Bluetooth solution, rich peripheral interfaces to support Arduino compiler and other compilers for development.

It also has an IO expansion interface circuit, LED power indicator circuit, LED control circuit, dual power supply circuit, voltage measurement circuit and 4-channel DC motor driver circuit on board.

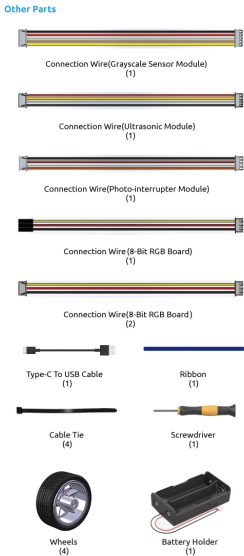
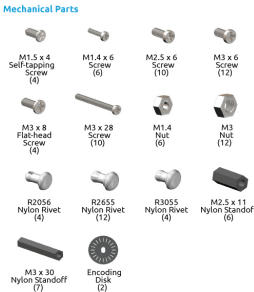
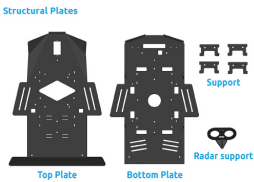
When you are developing and debugging with ESP32 RDP, you can connect peripherals according to your needs, and the rich external interfaces can make your project interesting.

1.1.2 Features

- Microcontroller: ESP32-WROOM-32D (4M) module
- Bluetooth protocol: Bluetooth v4.2 BR/EDR and BLE standards
- WIFI protocol: 802.11 b/g/n (802.11n, speeds up to 150 Mbps), 2.4 GHz ~ 2.5 GHz frequency range
- TYPE-C: Can be used as a power supply for the board or as a communication interface to connect a PC and the ESP32-WROOM-32 module.
- EN button: reset button
- Boot button: Press and hold the Boot button while press the EN button to enter the “firmware download” mode and download the firmware through the serial port.
- Input voltage: 7.0-30.0V (PH2.0-2P), 5.0V (TYPE-C)
- Output voltage: 7.0-30.0V (PH2.0-2P), 5.0V, 3.3V
- Output current: 5V/5A, 3.3V/1A, sleep state 20mA
- One channel SH1.0-4P port: I2C port.
- Four channel XH2.54-4P port: DC motor port
- 12 x PWM channel, 4 x ADC channel, 4 x GPIO pins.

You can find more details about the ESP32 module [HERE](#).

1.2 Components List



Note: After opening the package, please check whether the quantity of components is compliance with product description and whether all components are in good condition.

1.3 For Arduino User

Welcome to the guide for the ESP-4WD car Kit (Arduino version).

In this chapter, you will learn to configure your Arduino IDE, test all the components before assembling and then complete the assembly of the car. Once assembled, use the Arduino code to make the car do some interesting applications.

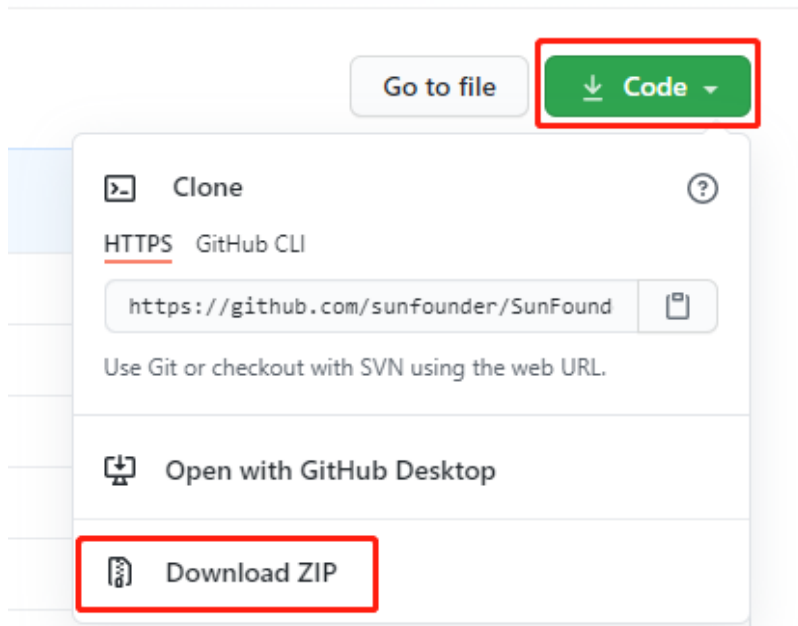
In addition you can use an app, the SunFounder Controller, to visually control or access the individual data of the car.

1.3.1 Preparations

Before you start using the kit, you will need to complete the following steps.

Download the ESP-4WD Package

Click [here](#) to download the ESP-4WD codes. After unzipping the zip file you have downloaded, you will see all the relevant files for the ESP-4WD.



Install Arduino IDE

Go to [Arduino Software Page](#) to download the Arduino IDE accordingly to your operating system then follow the instructions to install it.

Downloads



Arduino IDE 1.8.13

The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. This software can be used with any Arduino board.

Refer to the [Getting Started](#) page for Installation instructions.

SOURCE CODE

Active development of the Arduino software is [hosted by GitHub](#). See the instructions for [building the code](#). Latest release source code archives are available [here](#). The archives are PGP-signed so they can be verified using [this](#) gpg key.

DOWNLOAD OPTIONS

Windows Win 7 and newer
Windows ZIP file

Windows app Win 8.1 or 10 [Get](#)

Linux 32 bits
Linux 64 bits
Linux ARM 32 bits
Linux ARM 64 bits

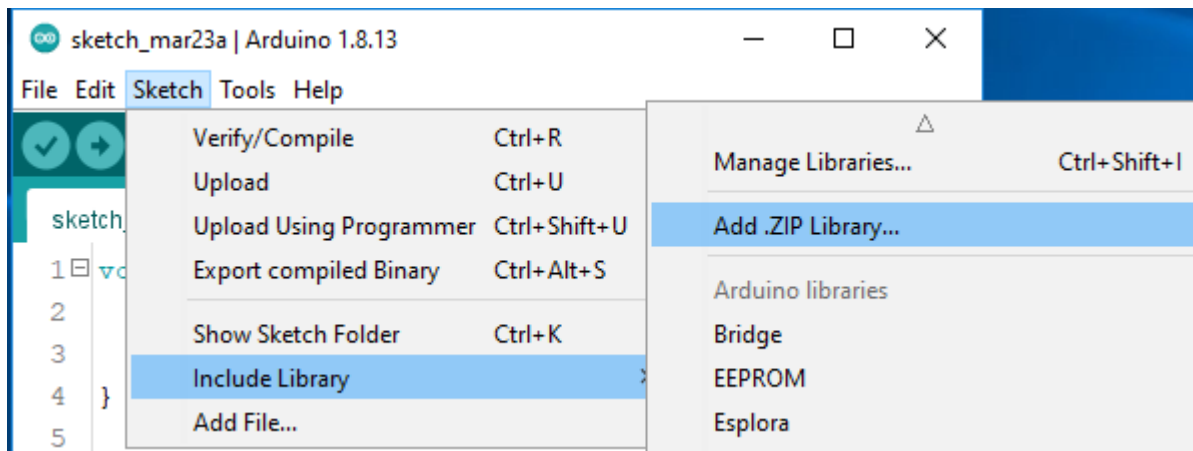
Mac OS X 10.10 or newer

[Release Notes](#)
[Checksums \(sha512\)](#)

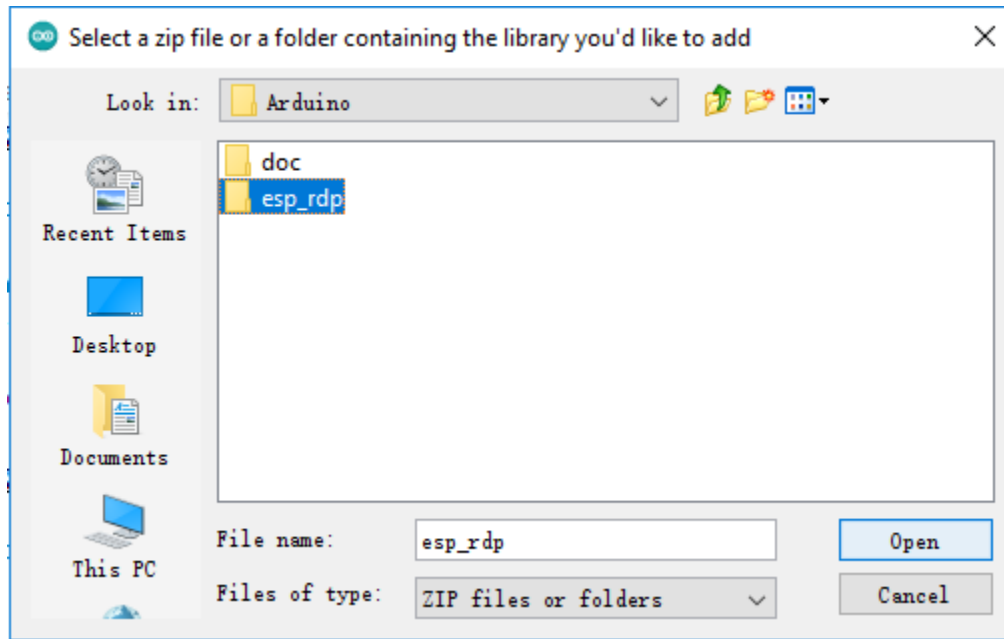
Add Libraries

Add esp_rdp

Click Sketch -> Include Library -> Add .ZIP Library.

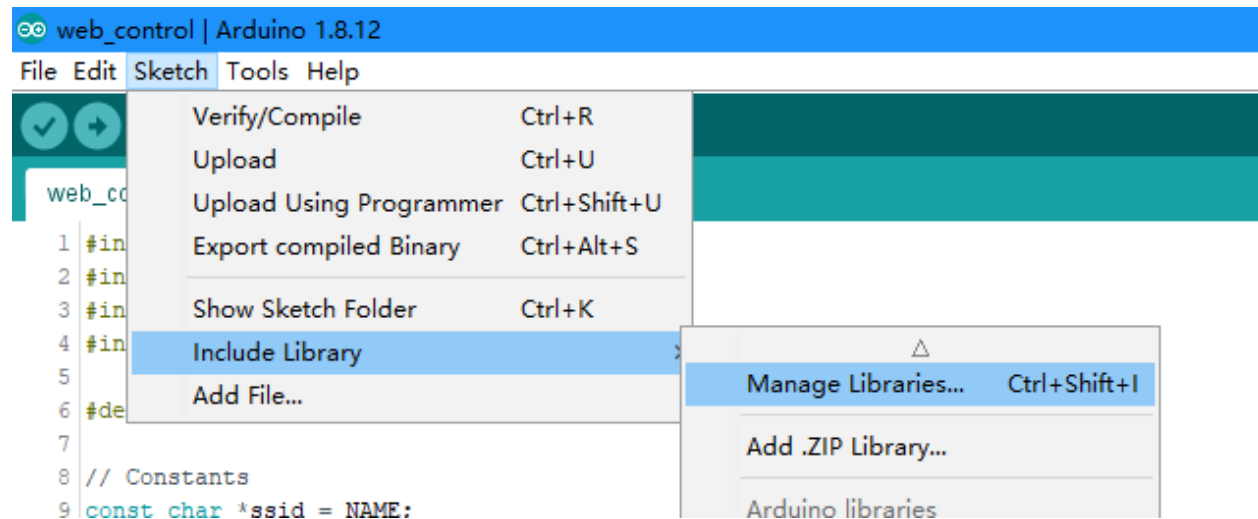


Select the **esp_rdp** library in the path of esp-4wd/Arduino, which you have downloaded before(<https://github.com/sunfounder/esp-4wd>), then click **OK**. After that you can call the functions in this library to control the esp-4wd car.

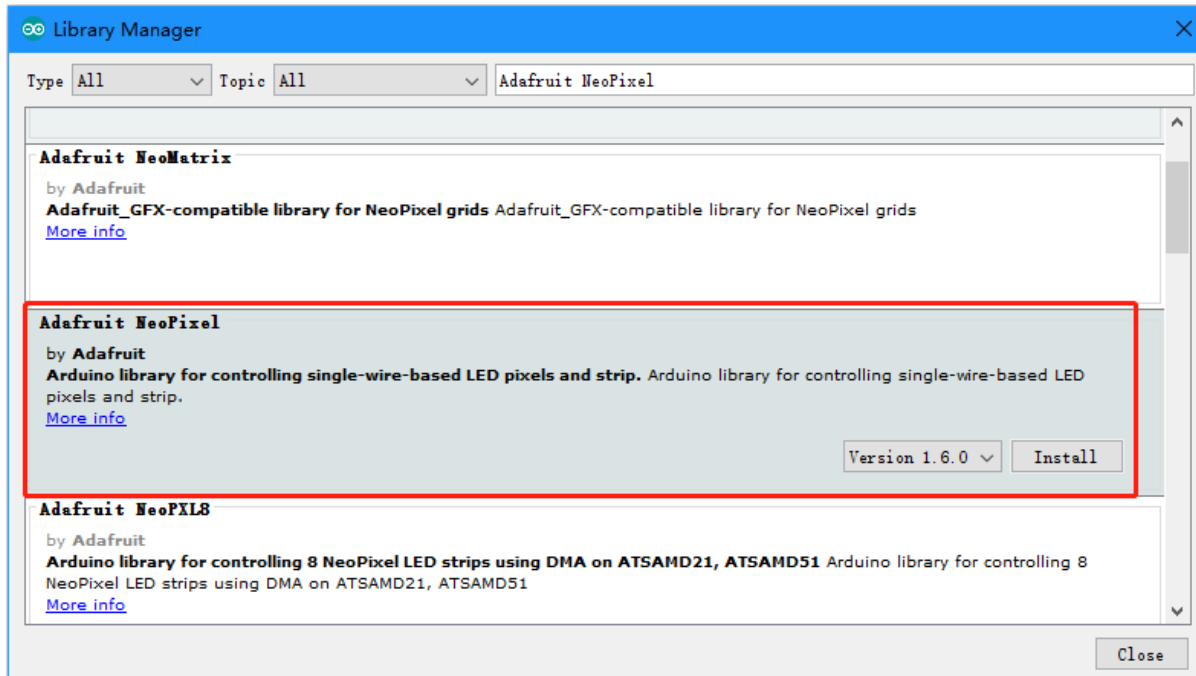


Add Adafruit NeoPixel

Click **Sketch** -> **Include Library** -> **Manage Libraries**.



Type in **Adafruit NeoPixel**, find it on the scroll down page and click **Install**. When using the 8-bit RGB board, you will need to call the functions in this library.



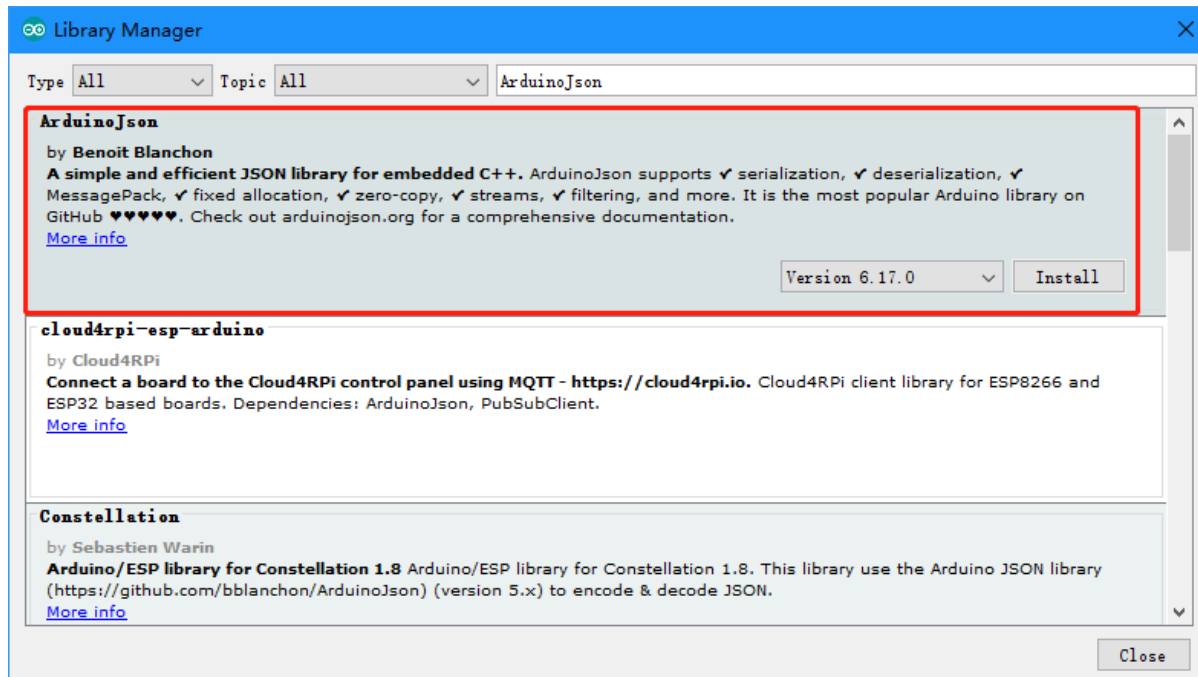
Add WebSockets

Continue to add the **WebSockets** library from the **Library Manage** page. This library is used to open the APP control service.



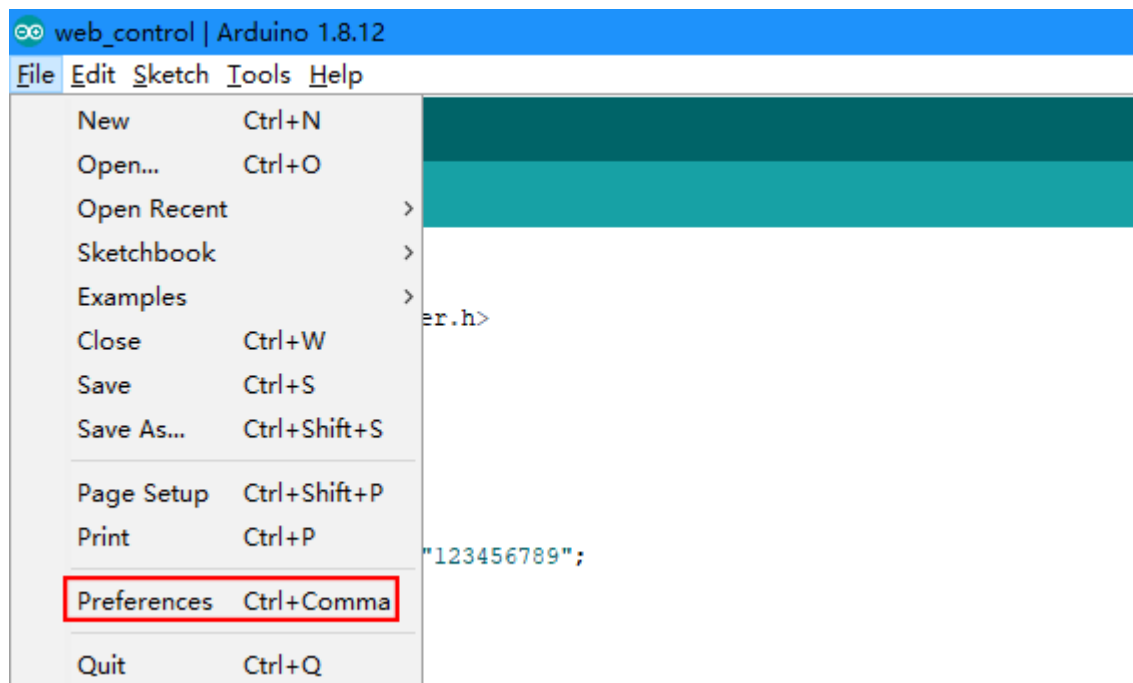
Add ArduinoJson

Continue to add the **ArduinoJson** library from the **Library Manage** page. This library is used for communication between the APP and the ESP-4WD car.

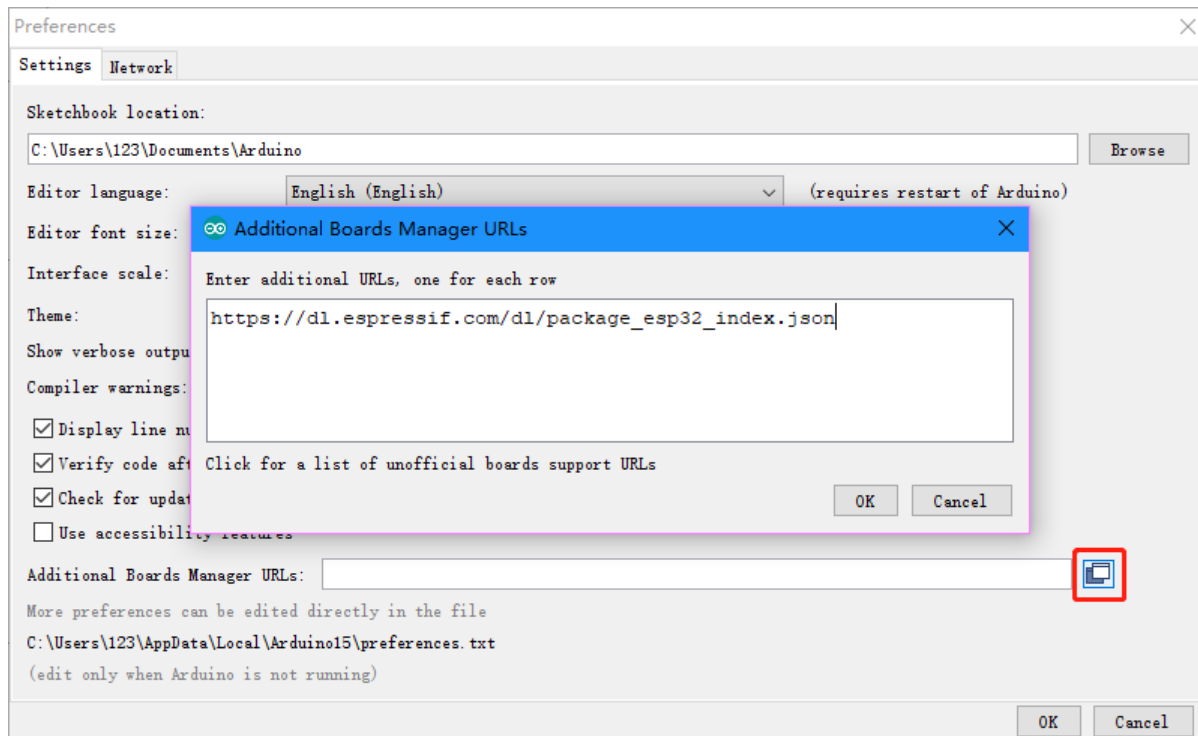


Installing ESP32 Add-on in Arduino IDE

1. In your Arduino IDE, go to **File -> Preferences**.



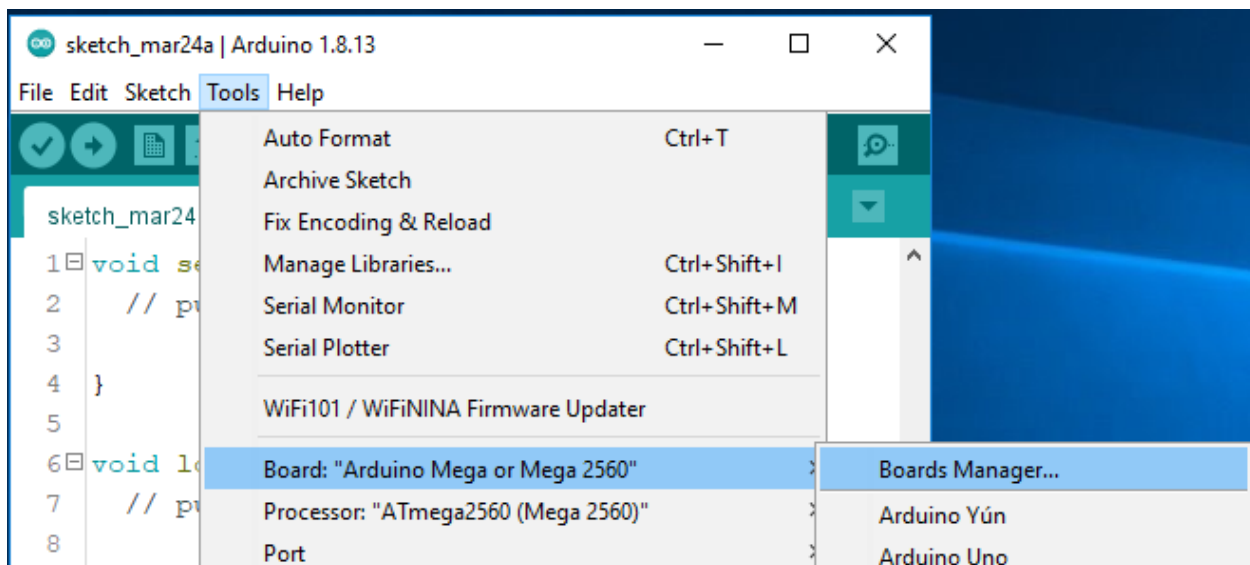
2. In the Preferences interface, click the **Upload** icon, and type in this URL, https://dl.espressif.com/dl/package_esp32_index.json, click OK.



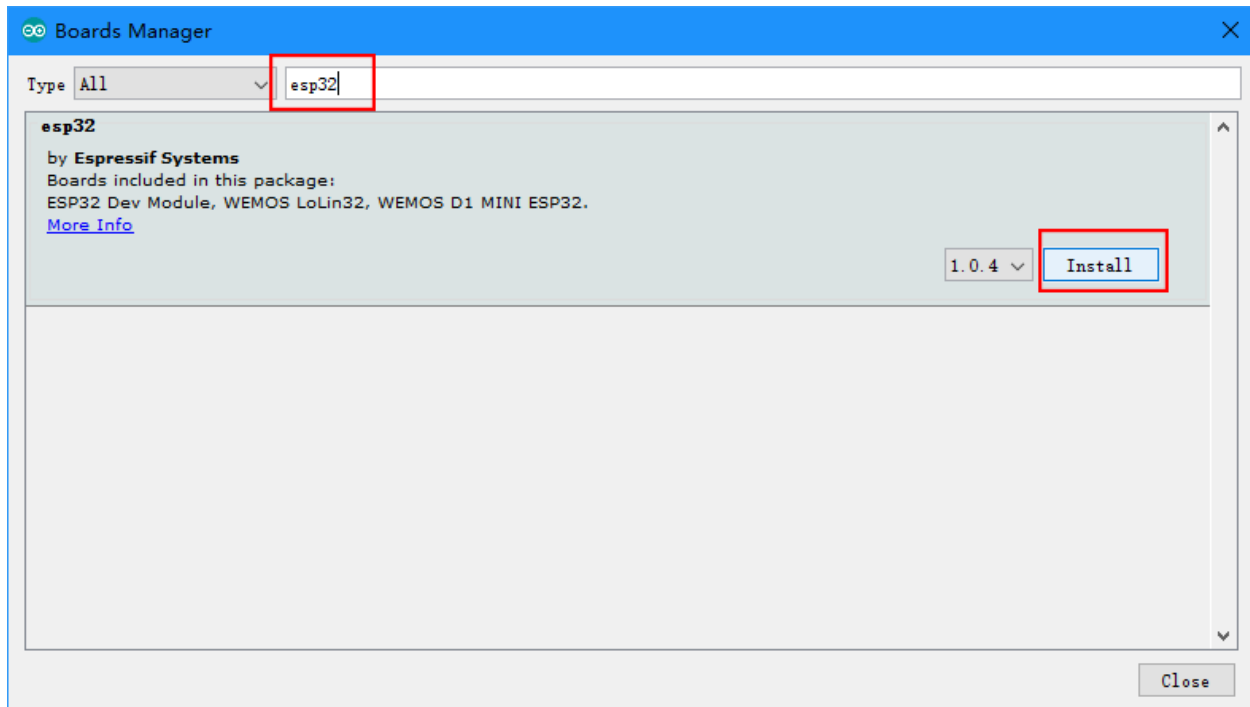
Note: If you already have the ESP8266 boards URL, you can separate the URLs with a comma as follows:

https://dl.espressif.com/dl/package_esp32_index.json, http://arduino.esp8266.com/stable/package_esp8266com_index.json

3. Open the Boards Manager by clicking **Tools -> Board -> Boards Manager**.



4. Search for **ESP32** and press install button for the **ESP32 by Espressif Systems**:

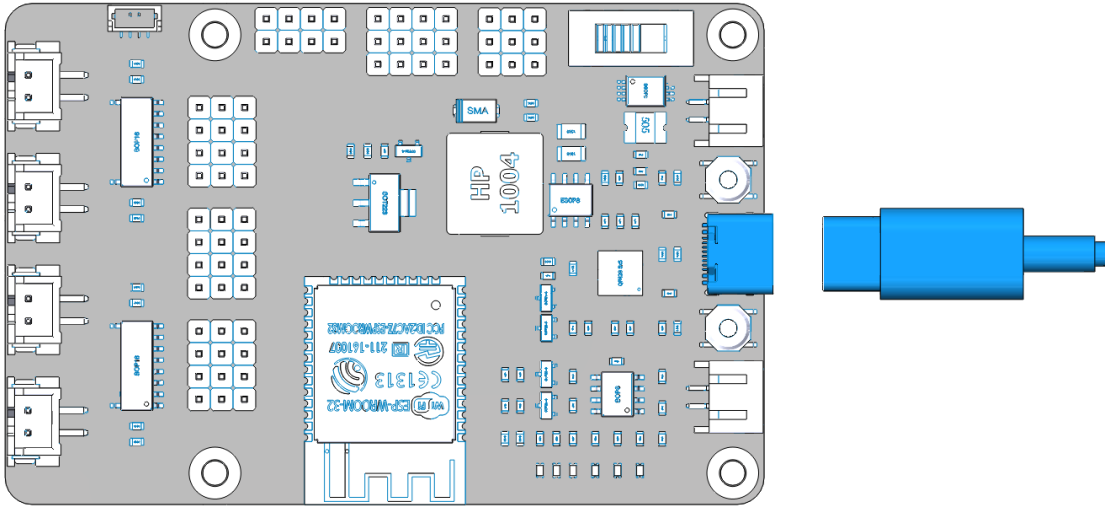


5. That's it. If your network is poor, errors may be reported during the installation process, please click **Install** button again until the installation is successful.

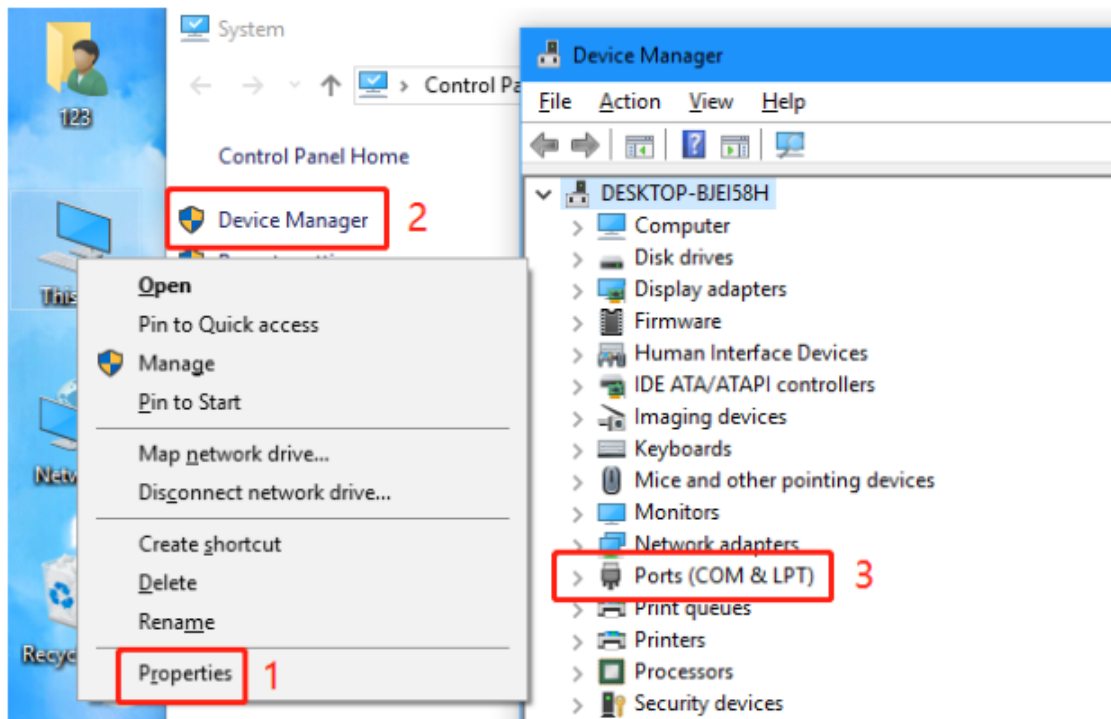


Install Driver

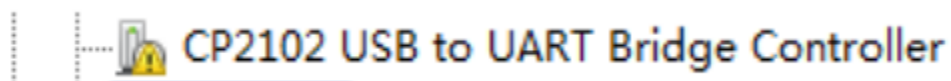
When you connect the ESP32 RDP board to the computer with a Type-C USB cable, the computer may not be able to recognize it. In this case, you need to install the driver manually.



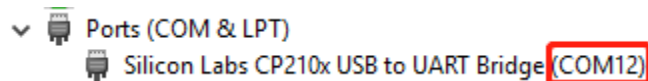
Right-click the **This PC** icon, and then click Properties -> Device Manager -> Ports to check the COM port information.



If the COM port(COMxx) does not appear, you need to download CP210x USB to UART Bridge VCP Drivers and install it.



Check the COM port information again. If the COM port(COMxx) can be displayed, the driver installation is successful.



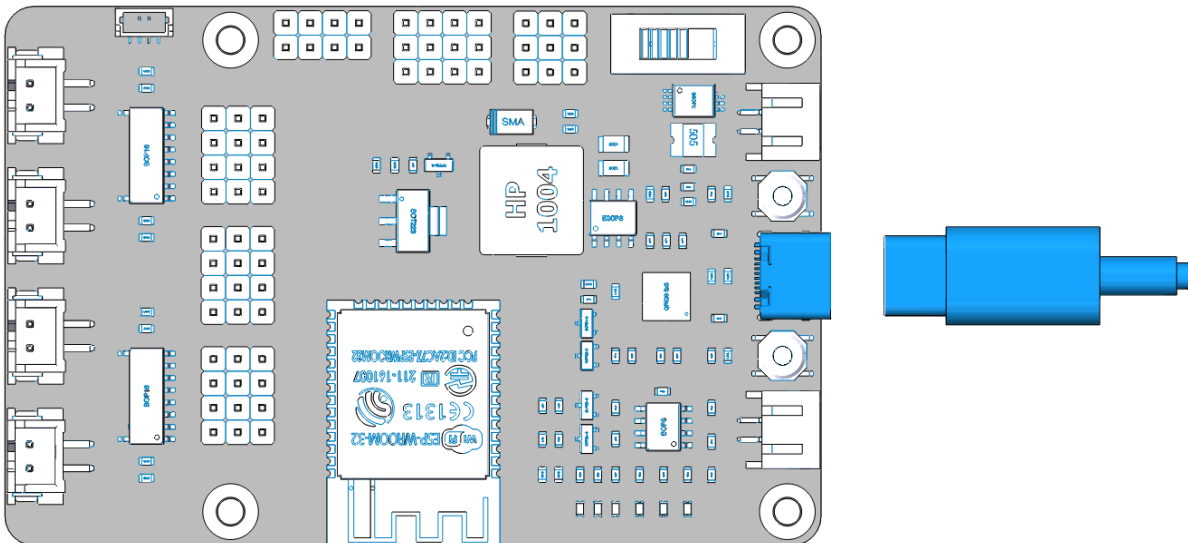
1.3.2 Test the Components with Arduino

Before assembling the ESP-4WD car, you need to test each component to make sure it is working properly. If there are any problems with these components, please contact us.

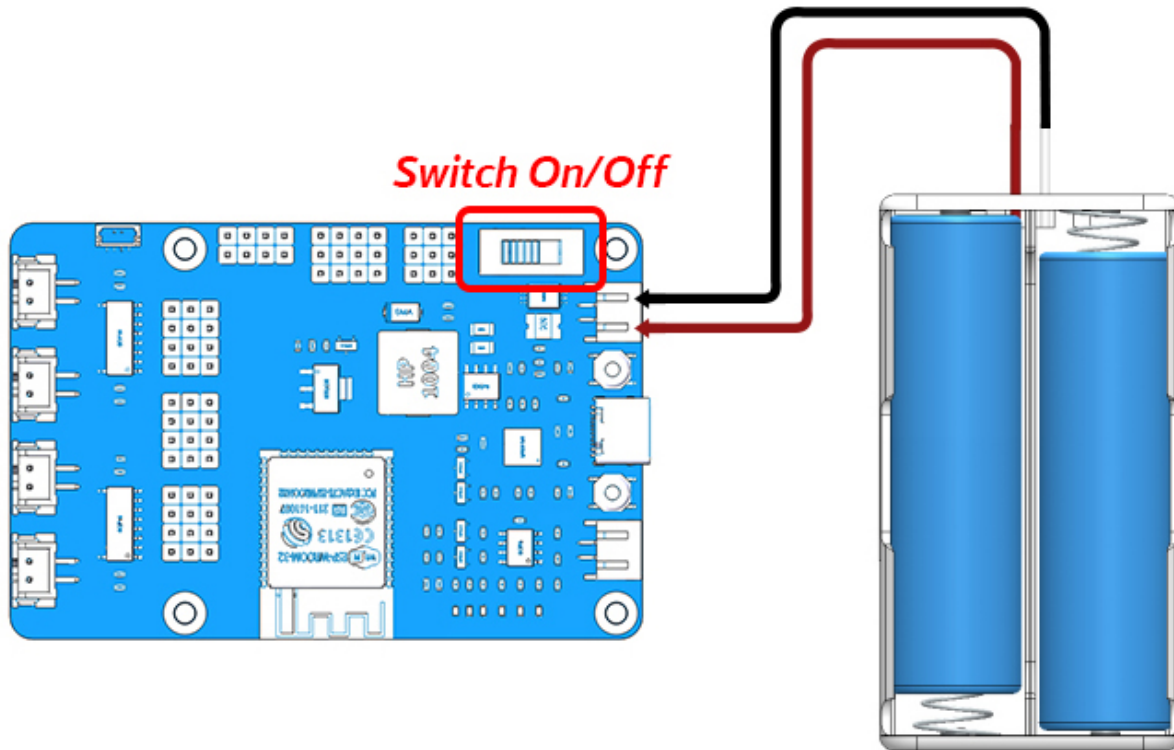
- *Power to ESP32 RDP*
- *Test the Motors*
- *Test the Ultrasonic Module*
- *Test the Grayscale Sensor Module*
- *Test the RGB Board*
- *Test the Servo*

Power to ESP32 RDP

In order to upload the code to the ESP32 RDP, you need to connect it to the computer with a Type-C USB cable.

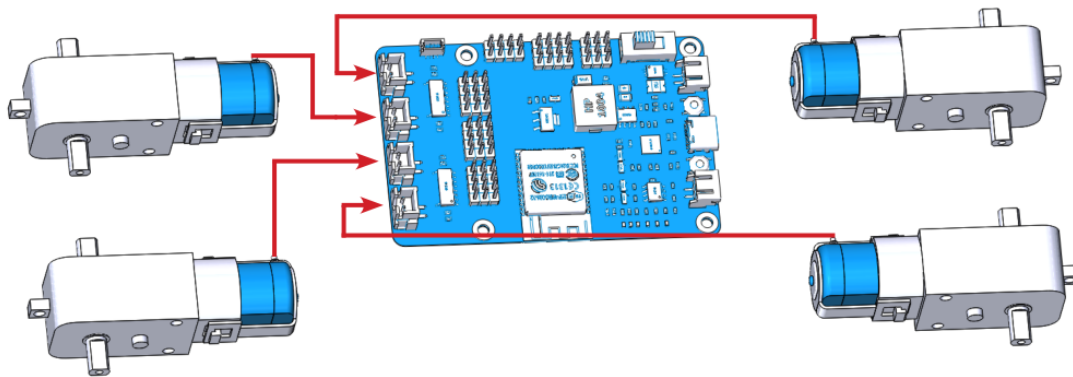


Then plug in the battery holder with two batteries to power other components and do not forget to slide the switch to ON.



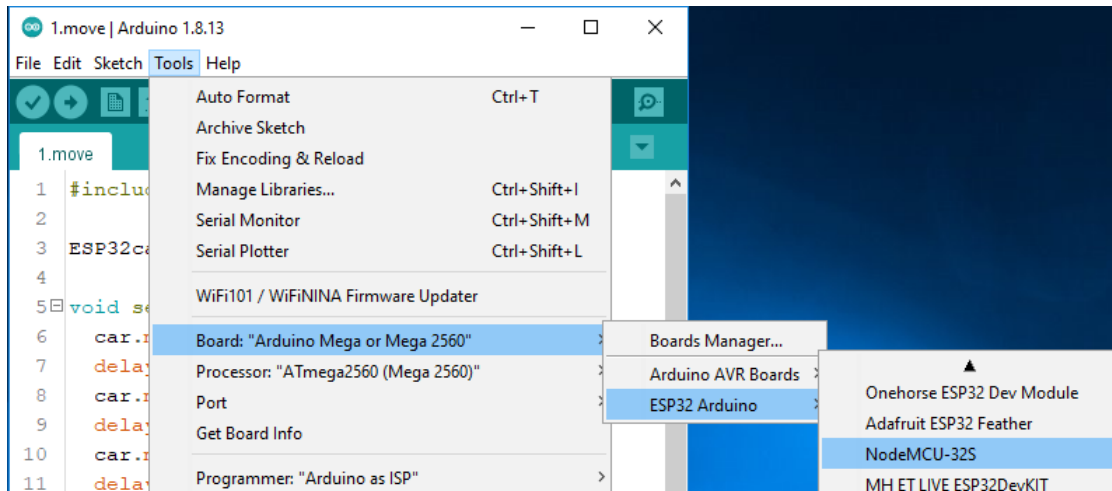
Test the Motors

Complete the wiring according to the diagram.

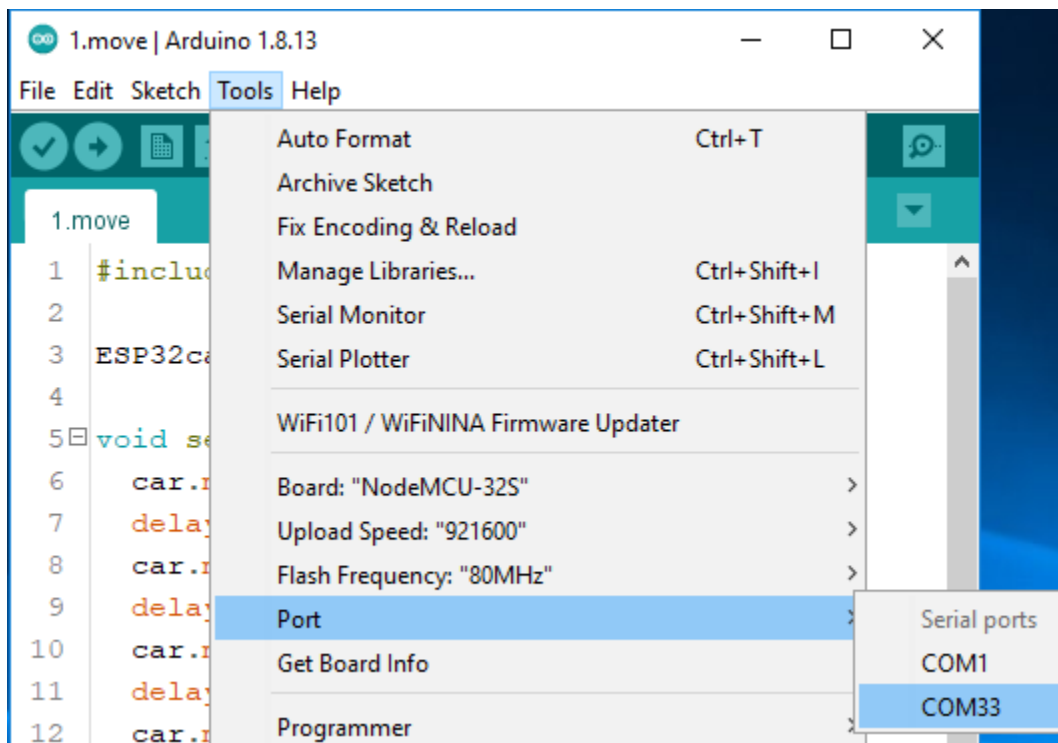


Open the `1.move.ino` file under the path `esp-4wd\Arduino\esp_rdp\examples\1.move`, you should have downloaded **esp-4wd** folder at [github](#) before.)

Select your Board in **Tools > Board** menu (ESP32 Arduino -> NodeMCU-32S), you need to scroll down several times to find it.

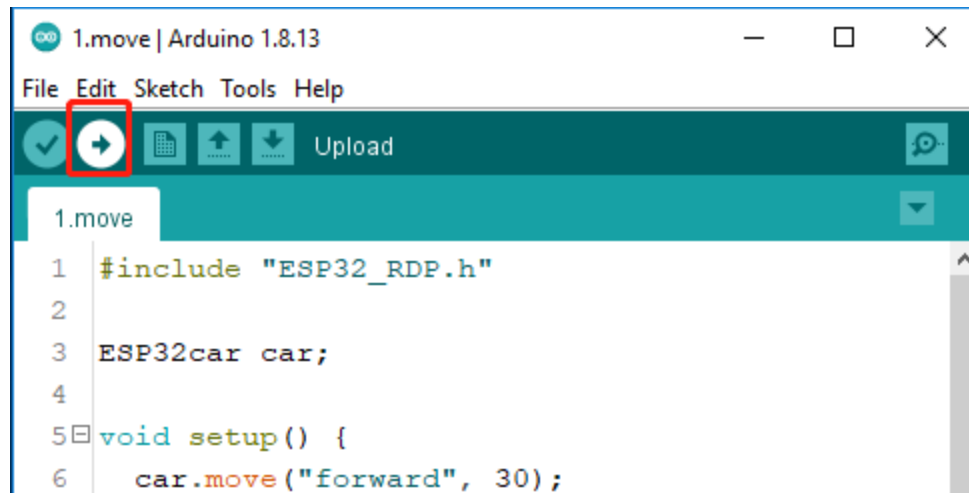


Select the Port.



Note: If you can not see the COM port in the Arduino IDE, you need to check whether the [CP210x USB to UART Bridge VCP Drivers](#) is installed successfully, or plug in/out the USB cable multiple times.

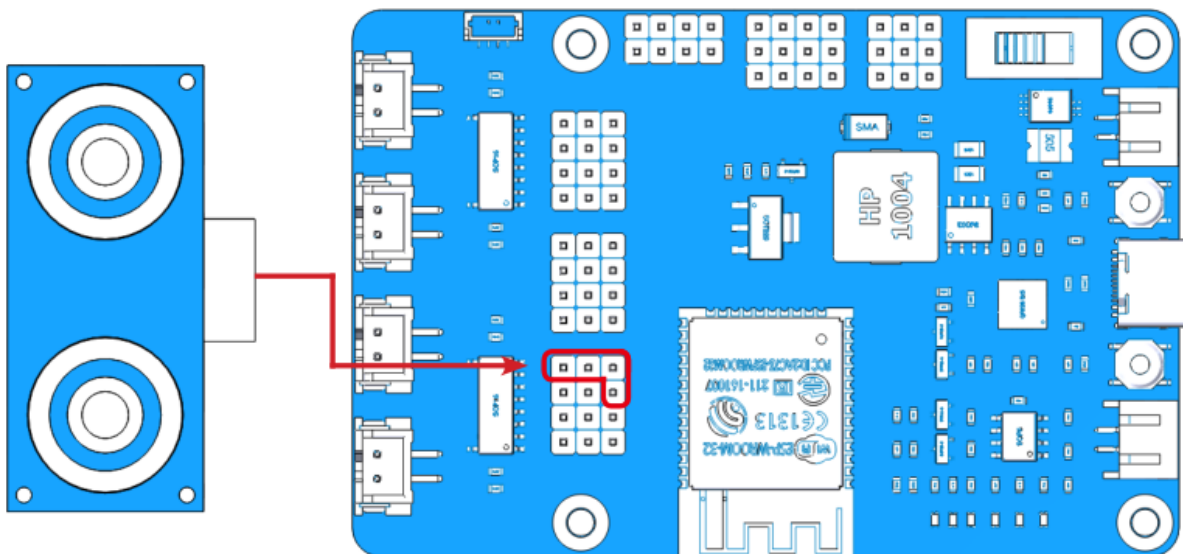
Upload the code to the ESP32 RDP.



After the code upload is complete, you will see that the four motors are rotating and changing the direction of rotation, and finally stop.

Test the Ultrasonic Module

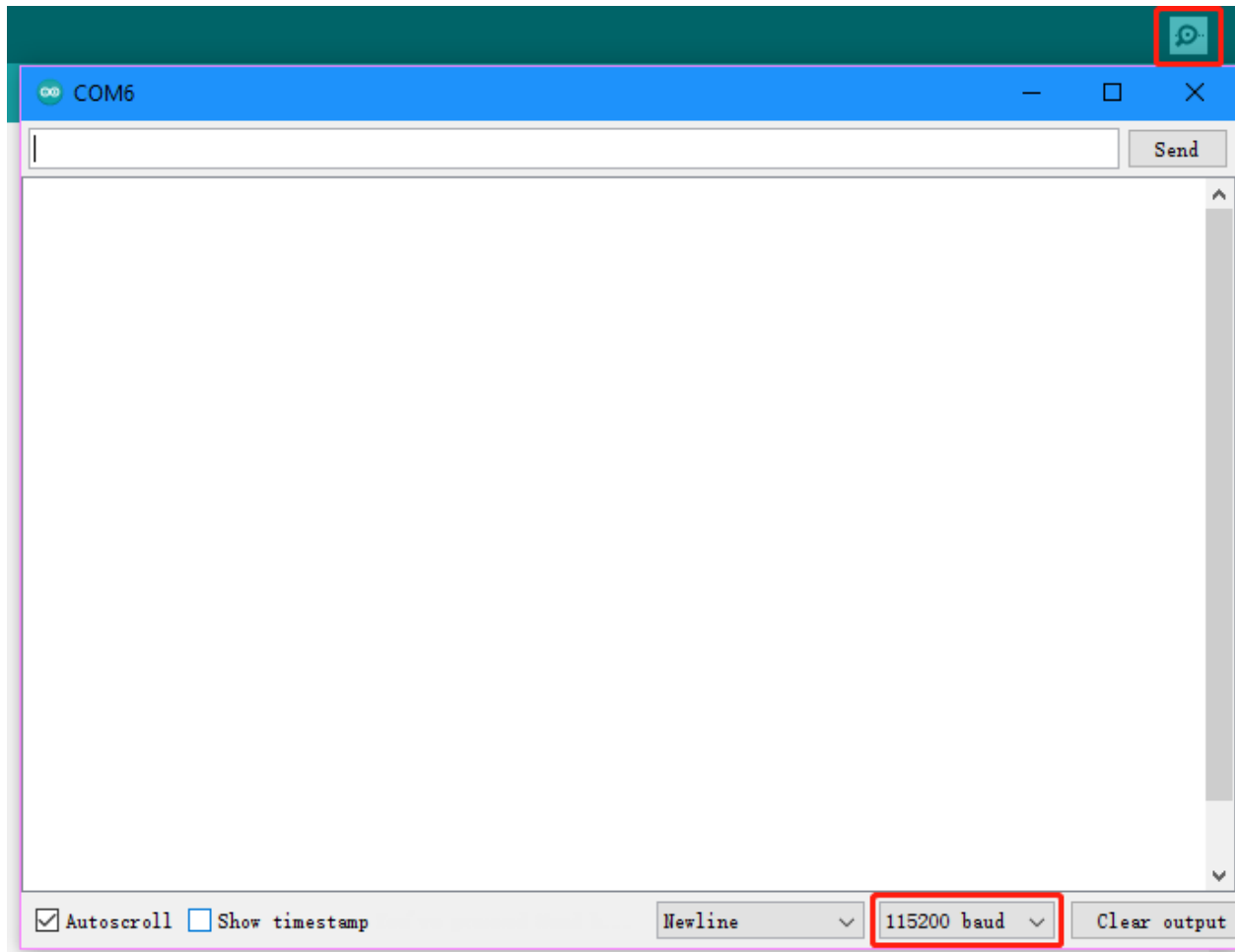
Complete the wiring according to the diagram.



ULTRASONIC		ESP32 RDP
Trig	->	IO13
Echo	->	IO12
VCC	->	5V
GND	->	GND

Upload the file 2.ultrasonic.ino under the path esp-4wd\Arduino\esp_rdp\examples\2.

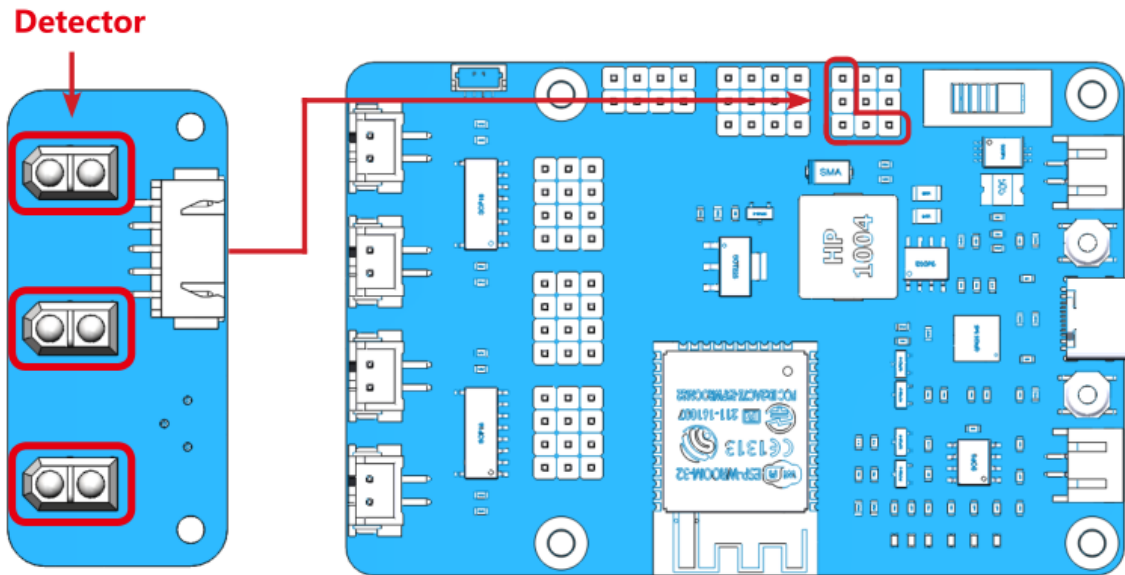
ultrasonic. Click the magnifier icon in the upper right corner to open the Serial Monitor and set the baudrate to 115200.



The Serial Monitor will print the distance value read by the ultrasonic module.

Test the Grayscale Sensor Module

Complete the wiring according to the diagram.



GRAYSCALE		ESP32 RDP
S0	->	IO35
S1	->	IO34
S2	->	IO36
3V3	->	3V3
GND	->	GND

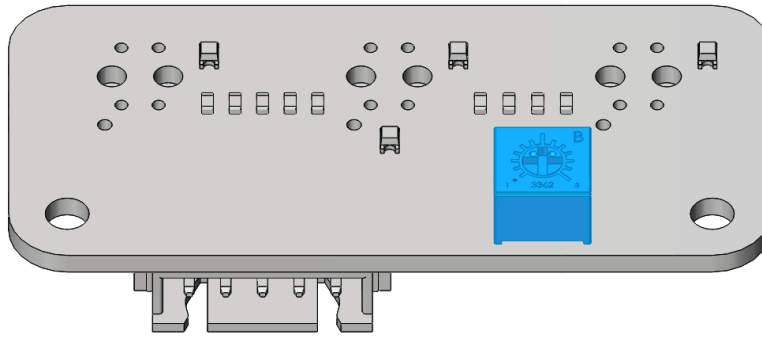
After uploading the `3.grayValue.ino` file under the path `esp-4wd\Arduino\esp_rdp\examples\3.grayValue`, the Serial Monitor will print the reading value of the grayscale sensor.

- Normally, it will detect a value above 1500 on white ground.
- On black ground, it will detect values below 900.
- On a cliff, it will detect a value below 110.
- If the reading is 0, it means that the height of the detector is higher than its detection range.

If the grayscale sensor module does not detect normal values, you will need to calibrate it. In order to make the calibrated value suitable for the assembled effect, the detector should be about **7mm** from the ground.

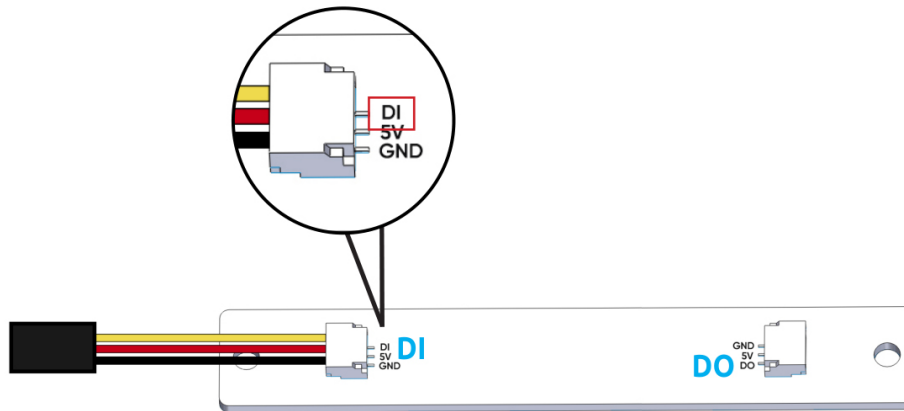
Now, Place it above the white ground and turn the potentiometer clockwise so that the reading is greater than 1100 (usually around 2000-4095).

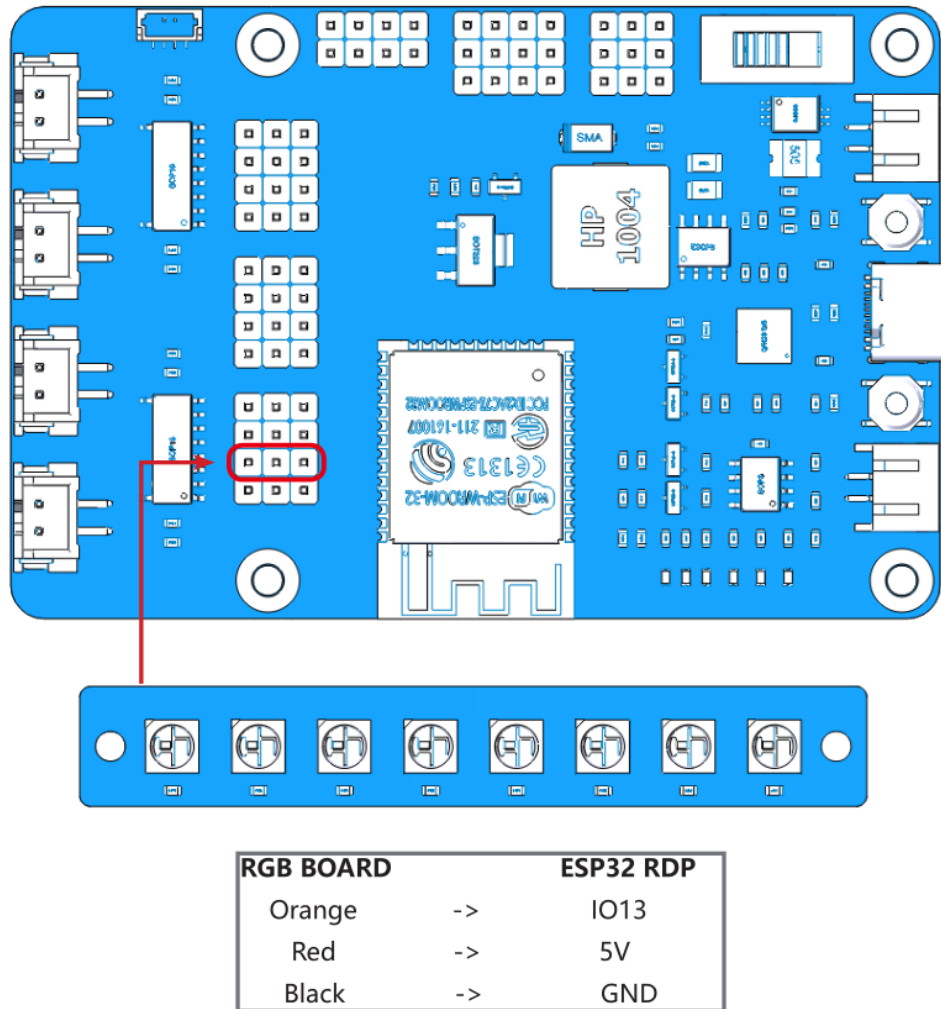
Then place it above the dark ground and turn the potentiometer counterclockwise to make it less than 900 (usually between 200 and 600). Repeat several times to get the maximum difference in both cases.



Test the RGB Board

Complete the wiring according to the diagram.

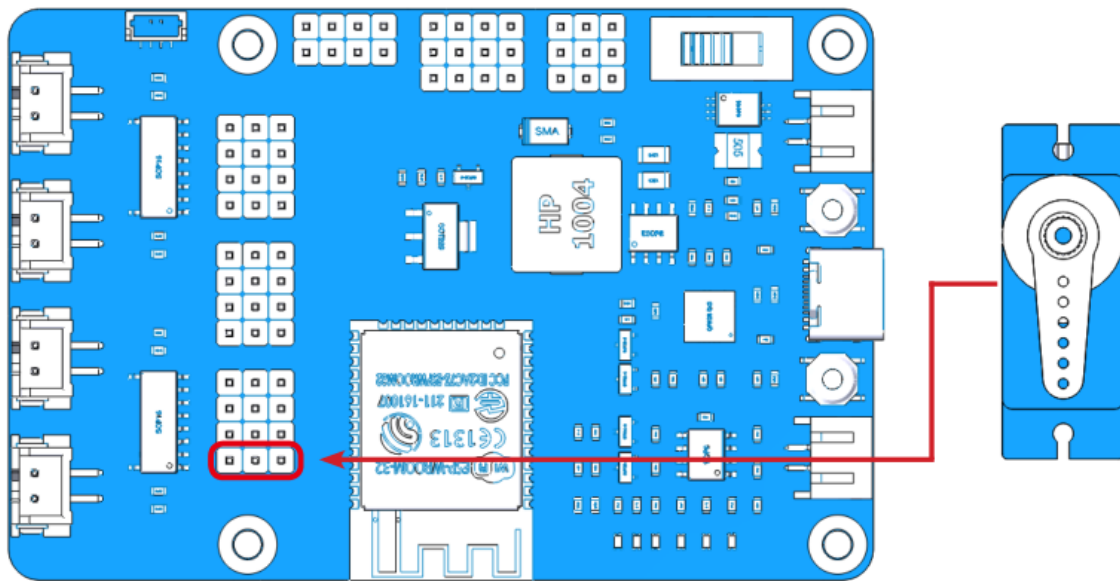




After uploading the `4.flashingLight.ino` file under the path `esp-4wd\Arduino\esp_rdp\examples\4.flashingLight`, the RGB board flashes every 0.5 seconds and changes color every time it flashes.

Test the Servo

Complete the wiring according to the diagram.



SERVO		ESP32 RDP
Orange	->	IO27
Red	->	5V
Brown	->	GND

Insert a rocker arm into the servo shaft, then upload the `servo.ino` file under the path `esp-4wd\Arduino\esp_rdp\examples\servo`, the servo will first turn left 30 degrees, then turn right 30 degrees, and finally back to 0 degrees.

Note: In the process of assembling the car in the following chapter, the servo needs to be kept at 0 degrees, so please do not upload new codes on the ESP32 RDP until the part with the Servo is assembled.

1.3.3 Hardware Assembling

In this chapter, you will learn to assemble an ESP-4WD car. Before starting the assembly, please make sure that all components are operating normally and the servo is calibrated to the 0 degree.



1.3.4 Play the Car with Arduino

There are two ways for arduino users to learn and play the ESP-4WD car: Code Control and APP Control. You can directly use arduino code to control the car, or DIY a remote control.

It should be noted that before using APP Control, you need to understand Code Control first.

Code Control

In this chapter you will learn to control the ESP-4WD car with Arduino codes.

You can copy the code below into the Arduino IDE or directly open the code file under the path `esp-4wd\Micropython\esp_rdp\examples`.

1-4 and servo code files have been used in the chapter *Test the Components with Arduino*, just for simple testing of motor, ultrasonic module, Grayscale Sensor Module, 8-bit RGB board and Servo.

Here, we focus on the three codes of 5-7.

In addition, the 8 and 9 code files are used in the *APP Control* chapter, and will not be explained here.

waterfallLight

After running `5.waterfallLight.ino`, the LEDs on the 8-bit RGB board under the car will turn on in random colors from the first to the 24th, and then turn off one by one from both ends to the middle.

After that, the RGB lights will turn on in random colors from the 24th to the first, and then turn off one by one from both ends to the middle.

```
#include "ESP32_RDP.h"

ESP32car car;

void setup() {
}

void loop() {
    int red = random(255);
    int green = random(255);
    int blue = random(255);
    for (int i = 0; i < 24; i++)
    {
        car.set_num_light(i, red, green, blue);
        delay(20);
    }
    for (int i = 23; i > 11; i--)
    {
        int j = 23 - i;
        car.set_num_light(i, 0, 0, 0);
        car.set_num_light(j, 0, 0, 0);
        delay(40);
    }
    for (int i = 23; i >= 0; i--)
    {
        car.set_num_light(i, red, green, blue);
        delay(20);
    }
}
```

(continues on next page)

(continued from previous page)

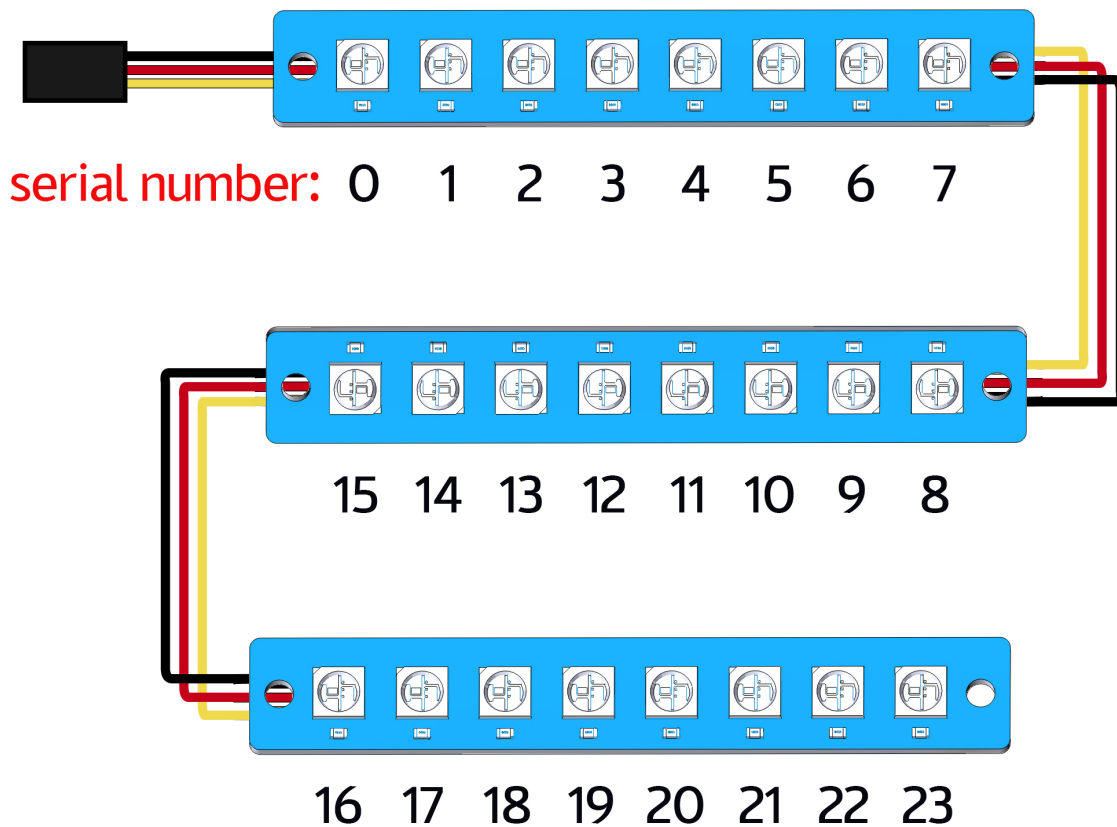
```

for (int i = 23; i > 11; i--)
{
    int j = 23 - i;
    car.set_num_light(i, 0, 0, 0);
    car.set_num_light(j, 0, 0, 0);
    delay(40);
}

```

The statement to light up the LED is `car.set_num_light(i, red, green, blue);` the first parameter is the number of the light, and the last three parameters are the RGB value.

For example, `car.set_num_light(4, 255, 0, 0)` means to make the No. 4 LED light up in red.



measureSpeed

Run `6.measureSpeed.ino`, the car will move at a random speed, and the 2-ch Photo- interrupter module will detect the speed of the car.

The light emitted from the transmitting end of the 2-ch Photo-interrupter module to the receiving end will pass through the Encoding Disk (with 20 holes). When the receiving end does not receive the light, it will send a 0 to the microcontroller, otherwise it will send a 1.

This means that when a total of 20 1 are detected, the wheel of the car has turned one round (a distance of the wheel circumference has been traveled forward).

In the same way, we can detect the frequency of the 1 received by the microcontroller and calculate the speed of the car in cm/s.

```
#include "ESP32_RDP.h"

ESP32car car;

#define SPEED_PIN_LEFT    26
#define SPEED_PIN_RIGHT   25

hw_timer_t * timer = NULL;
volatile uint8_t left_Counter = 0;
volatile uint8_t right_Counter = 0;
int counter = 0;
float carSpeed = 0;

void on_left_Pin() {
    left_Counter++;
}

void on_right_Pin() {
    right_Counter++;
}

void IRAM_ATTR onTimer() {
    counter = (left_Counter + right_Counter);
    left_Counter = 0;
    right_Counter = 0;
}

float get_speed() {
    float value = 0;
    value = float(counter) / 2.0 / 20.0 * 2.0 * 3.14 * 3.3;
    return value;
}

void setup() {
    Serial.begin(115200);
    pinMode(SPEED_PIN_LEFT, INPUT);
    attachInterrupt(SPEED_PIN_LEFT, on_left_Pin, RISING);
    pinMode(SPEED_PIN_RIGHT, INPUT);
    attachInterrupt(SPEED_PIN_RIGHT, on_right_Pin, RISING);
    // Set 80 divider for prescaler (see ESP32 Technical Reference Manual for more
    // info).
    timer = timerBegin(2, 80, true);
    // Attach onTimer function to our timer.
```

(continues on next page)

(continued from previous page)

```

    timerAttachInterrupt(timer, &onTimer, true);
    // Set alarm to call onTimer function every second (value in microseconds).
    // Repeat the alarm (third parameter)
    timerAlarmWrite(timer, 1000000, true);
    // Start an alarm
    timerAlarmEnable(timer);
}

void loop() {
    int speed = random(100);
    car.move("forward", speed);
    carSpeed = get_speed();
    Serial.println(carSpeed);
    delay(1000);
}

```

morePlay

Run 7.morePlay.ino, this example provides 4 ways to play ESP-4WD car. You can switch between different modes by modifying the value of the **mode** variable.

```

#include "ESP32_RDP.h"

ESP32car car;

int mode = 1;

void setup() {
}

void loop() {
    switch (mode)
    {
        case 1:
            car.avoid(40, 30);
            break;
        case 2:
            car.follow(40, 30);
            break;
        case 3:
            car.is_on_edge(110);
            break;
        case 4:
            car.track_line(400, 50);
            break;
    }
}

```

Function Introduction of morePlay

Note: For how the following 4 functions implement the corresponding functions, please refer to `ESP32_RDP.cpp` and `ESP32_RDP.h` under the path `esp-4wd\Arduino\esp_rdp\src`.

avoid()

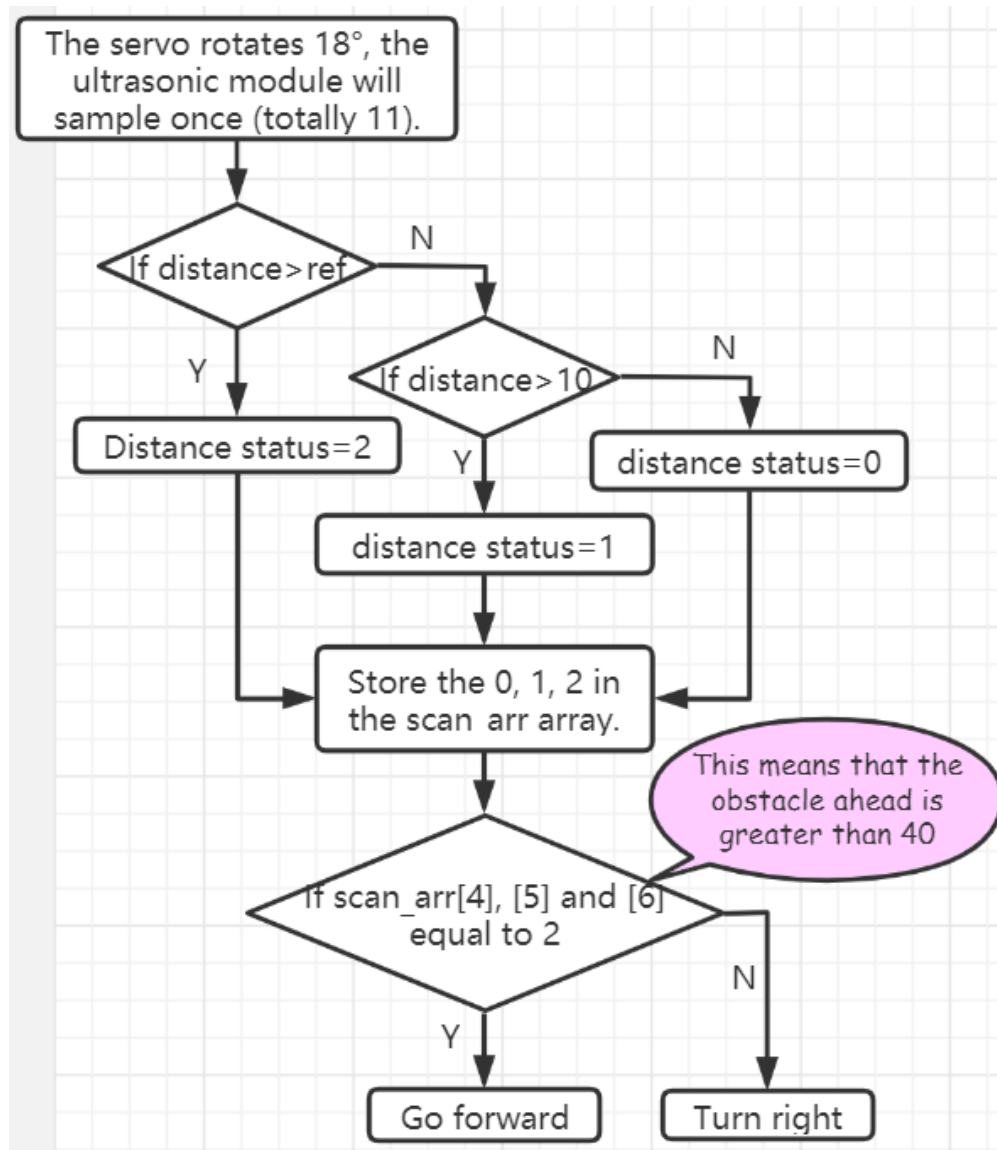
The default mode(mode=1) is obstacle avoidance.

`avoid(int ref, int speed)`

- `ref` refers to the reference distance value.
- `speed` refers to the forward speed.

ESP-4WD car will move forward at 30% speed and return the distance state according to the obstacle in front.

- If distance > 40, return the distance state 2, if distance > 10, then return 1, otherwise it will return 0.
- If the obstacle distance is greater than 40, the car will move forward, otherwise it will turn right.

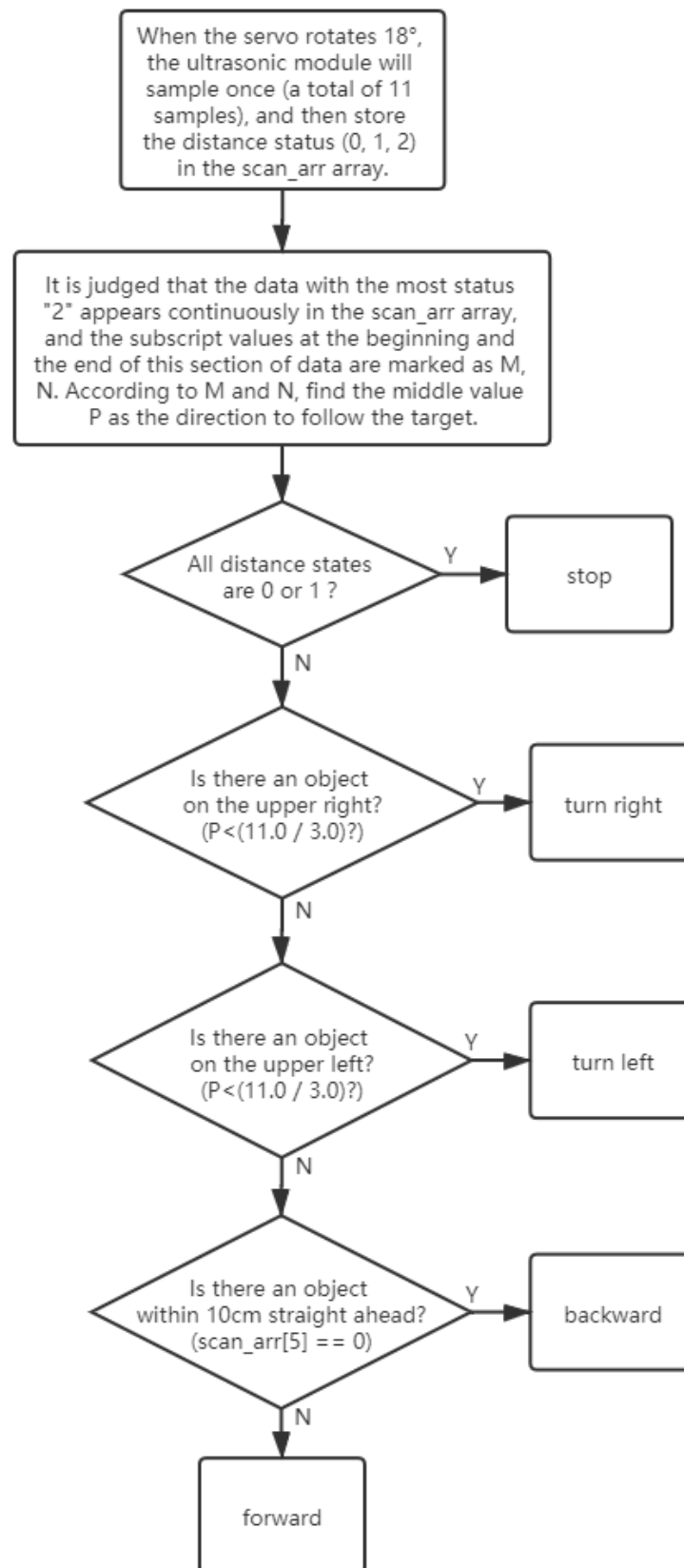
**follow()**

Modify the value of the mode variable to 2 so that the mode is set to follow.

```
follow(int ref, int speed)
```

- ref refers to the reference distance value.
- speed refers to the forward speed.

ESP-4WD car will move forward at 30% speed and automatically follow objects within 40cm in front.



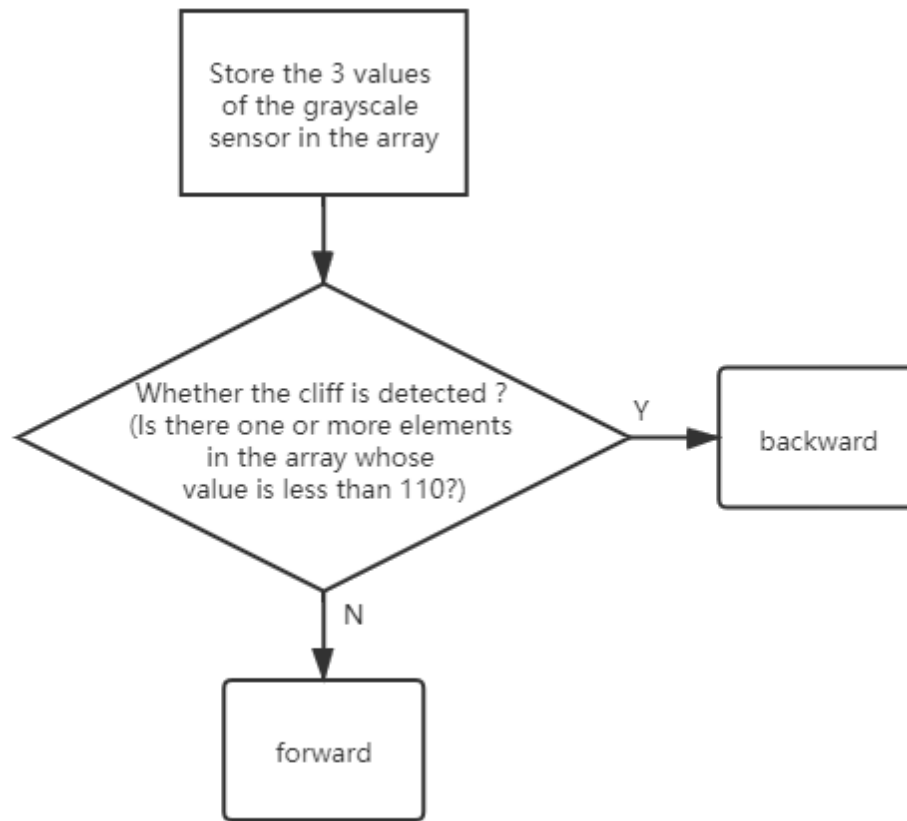
is_on_edge()

Modify the value of the mode variable to 3 so that the mode is set to cliff detection.

```
is_on_edge(int ref)
```

- `ref` refers to the reference gray value.

When ESP-4WD car detects a cliff (a place where the grayscale sensor's detection value is below 110), it will retreat a certain distance.

**track_line()**

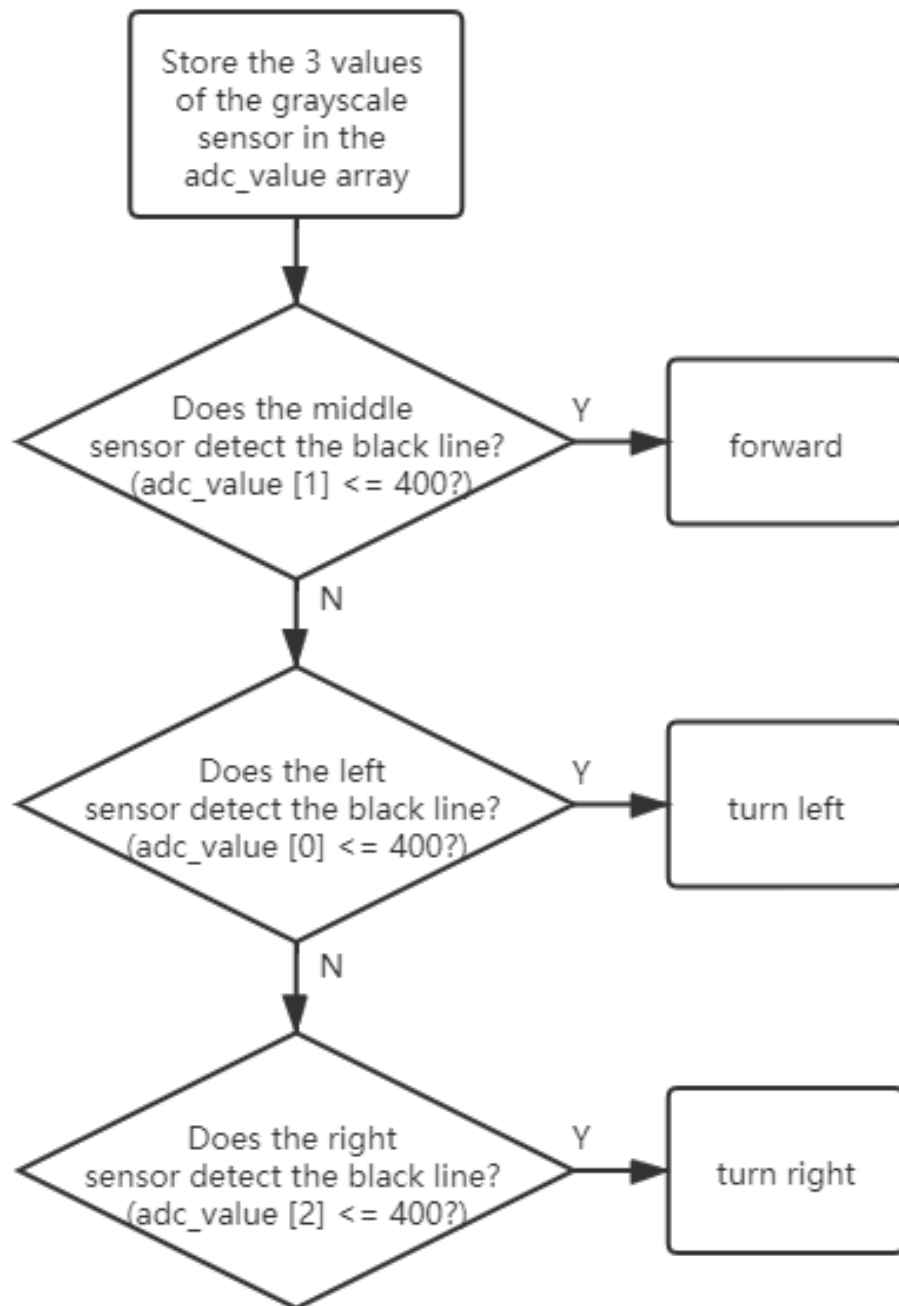
Modify the value of the mode variable to 4 so that the mode is set to track line.

```
track_line(int ref, int speed)
```

- `ref` refers to the reference gray value.
- `speed` refers to the forward speed.

Note: You can replace ``ref``(400) with another number, which is the threshold between the black line and the white ground read by the grey scale sensor.

The ESP-4WD car moves along the black line on the white ground.



APP Control

In this chapter, you will learn to use a APP - Sunfounder Controller to control the car.

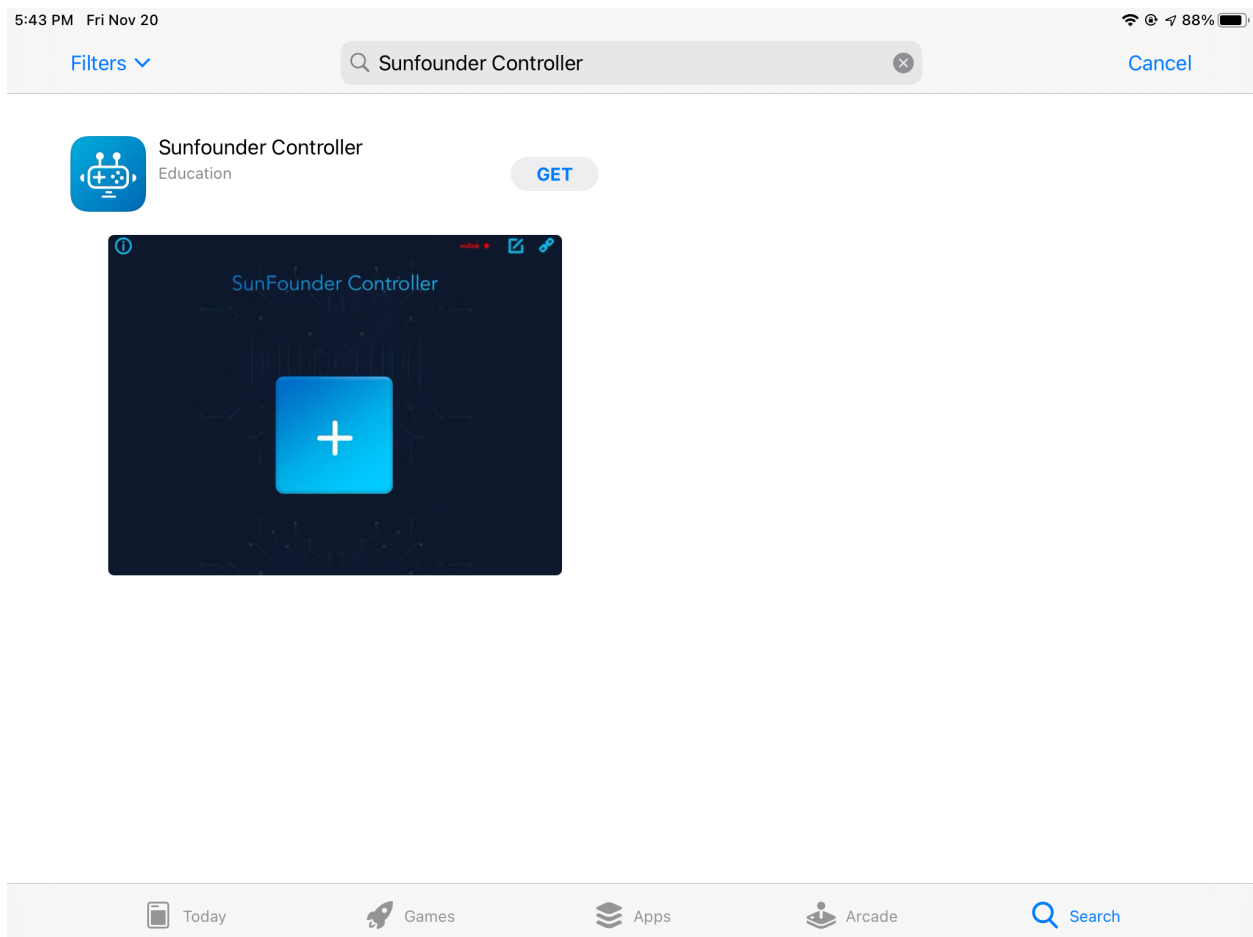
The complete operation process is as follows **Install Sunfounder Controller -> Establish Communication -> Control the Car with APP.**

You can check the **About Sunfounder Controller** and **DIY Controller** sections according to your choice.

- *Install Sunfounder Controller*
- *About Sunfounder Controller*
- *Establish Communication*
- *Control the Car with APP*
- *DIY Controller*

Install Sunfounder Controller

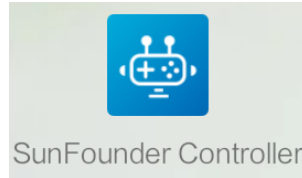
Open App Store (iOS/Mac OS X system) or Play Store (Android/Windows/Linux system), then search and download Sunfounder Controller.



About Sunfounder Controller

Page Introduction

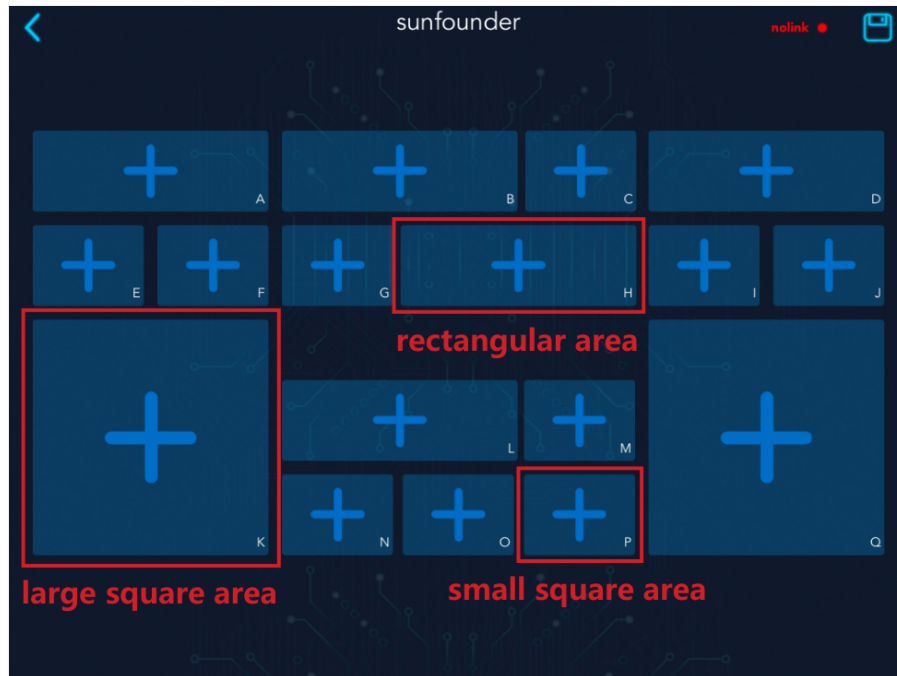
Start the Sunfounder Controller.



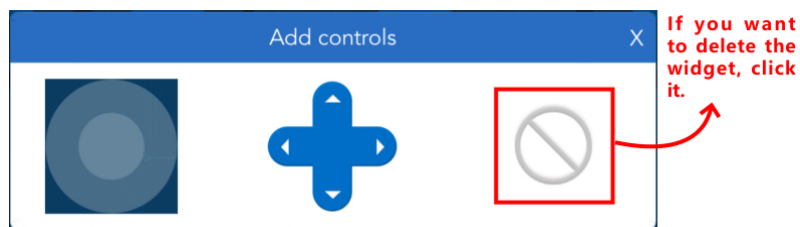
Click the middle button to add a new controller.



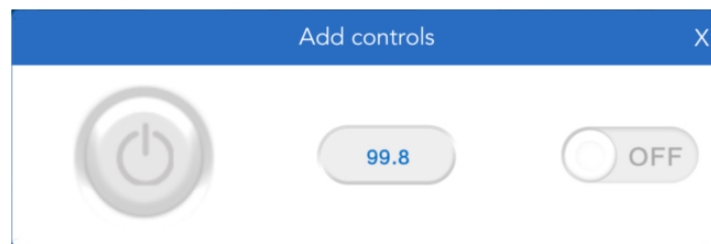
Sunfounder Controller is a platform that can add custom controllers. It reserves many widget interfaces. There are a total of 17 areas from A to Q. Each area has selectable widgets.



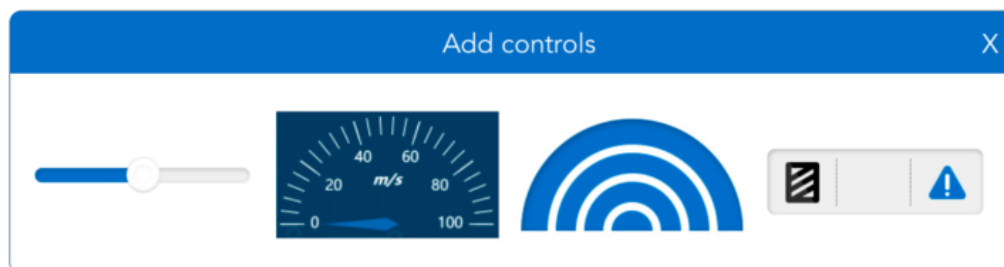
The available widgets in the **large square area** include joystick and D-Pad.



The available widgets in the **small square area** include button, digital display and switch.



The available widgets for the **rectangular area** include slider, dial, ultrasonic radar and grayscale detection tool.



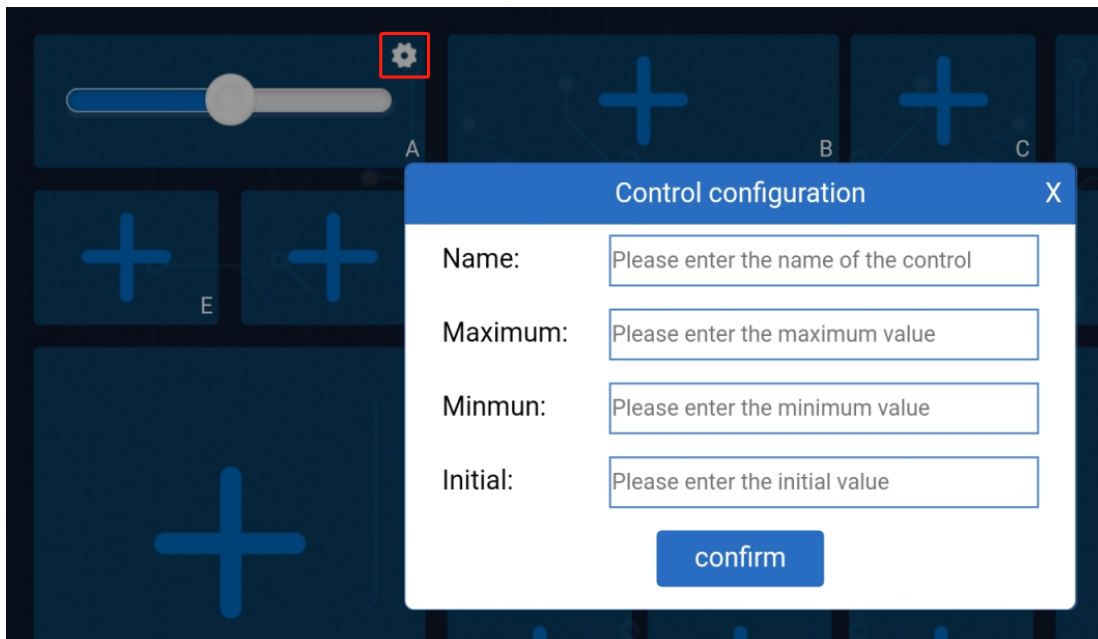
Widgets List

Here, you will learn the parameter types and ranges of control widgets and data widgets.


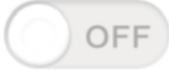








Control Widgets

The control widgets of Sunfounder Controller include buttons, switches, joystick, D-Pad, and slider.

You can modify the name, parameter range and initial value of some widgets by clicking the settings button in the upper right corner



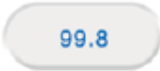


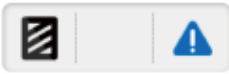



When we use these control widgets, ESP-4WD car will receive the control data. Through these control data, we can write code to control the car.

Widget	Trigger Event	Data (Type)
	Hold	True(boolean)
	Release	False(boolean)
	Open (ON)	True(boolean)
	Close (OFF)	False(boolean)
	Move the middle cursor	axis array[x,y] x_axis range: -100~100 (int) y_axis range: -100~100 (int)
	Hold 	"forward" (string)
	Hold 	"backward" (string)
	Hold 	"left" (string)
	Hold 	"right" (string)
	Sliding slider	Range: 0~100 (int) Range can be modified through 

Data Widgets

The data widget of Sunfounder Controller includes digital displays, dial, ultrasonic radar, and grayscale detection tool.

When we send sensor data to these data widgets, we can show the data on the corresponding widgets. At the same time, you can also modify the name, unit and parameter range of the data widget by clicking the setting icon in the upper right corner.

Widget	Data (Type)	Display
	-1000.00~1000.00(float)	Can not be modified
	0~100(int)	Range can be modified through 
	Array with 3 elements (float)	Element Value<110:  Element Value<400:  Element Value>= 400:no pattern (Can be modified)
	Array with 2 elements (float)	First element: the angle value Second element: distance value

Establish Communication

There are two ways to establish communication between Sunfounder Controller and ESP-4WD car: One is AP mode, the other is STA mode.

- **AP Mode:** You need to connect Sunfounder Controller to the hotspot released by ESP-4WD car.
- **STA Mode:** You need to connect Sunfounder Controller and ESP-4WD car to the same LAN.

We can switch the communication mode by modifying the code `8.app_control.ino` under the path `esp-4wd\Arduino\esp_rdp\examples\8.app_control` and defining the `SWITCH_MODE` variable as `ap` or `sta`.

```
#define SWITCH_MODE "ap"
```

AP Mode

If you want to use AP mode, you need to connect Sunfounder Controller to the hotspot released by ESP-4WD car.

1. Open the code `8.app_control.ino`, modify the `NAME` and `AP_PASSWORD` to yours.

```
#define NAME "ESP-4WD Car"
#define AP_PASSWORD "123456789"
```

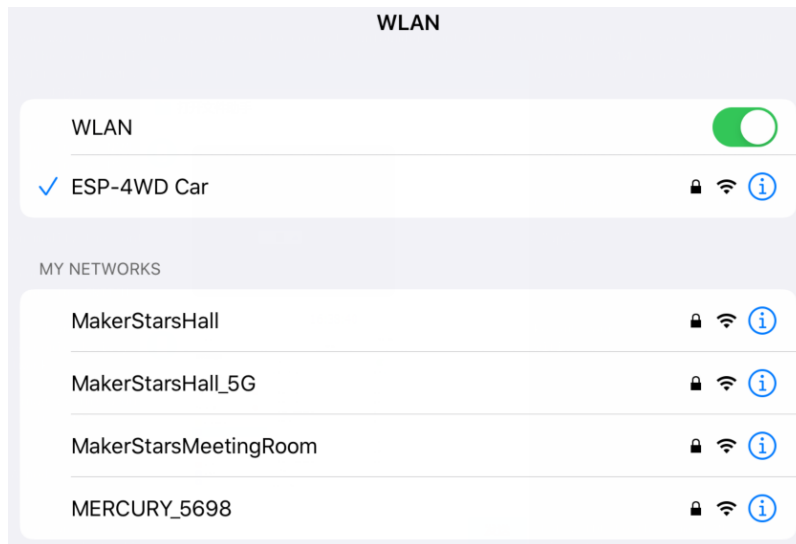
Note: The NAME in the code is both the SSID and the name of the car, if you have more than one EPS-4WD Car, you need to set different NAMES for them to avoid a wrong connection.

In addition, you need to set a password of more than 8 digits.

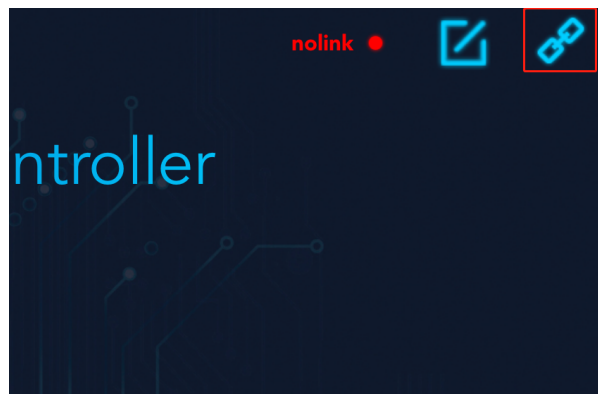
2. Then define the SWITCH_MODE variable as ap.

```
#define SWITCH_MODE "ap"
```

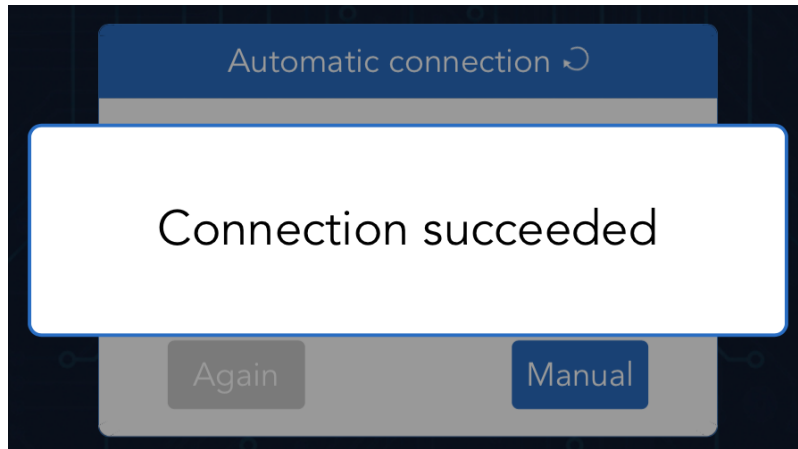
3. After downloading the code, ESP-4WD car will send a hotspot signal, then take out your mobile device, open the WLAN management interface and connect to the wifi network.



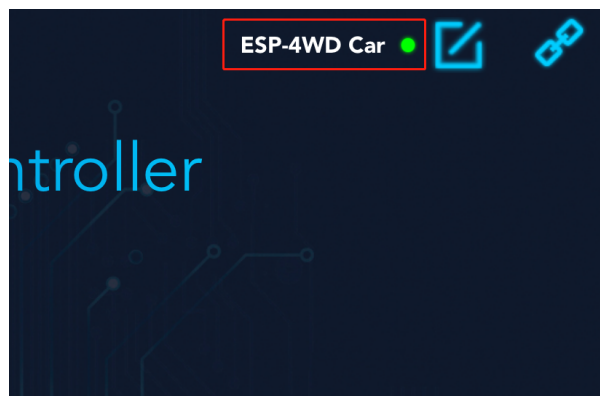
4. Open **Sunfounder Controller** and click the **Connect** icon on the top right corner.



5. A prompt box will appear if the connection is successful.



6. And the name of the car will be shown on APP.



STA Mode

If you want to use STA mode, you need to connect Sunfounder Controller and ESP-4WD car to the same LAN.

1. Open the code `8.app_control.ino`, modify the `STA_NAME` and `STA_PASSWORD` to yours.

```
#define STA_NAME "MakerStarsHall"  
#define STA_PASSWORD "sunfounder"
```

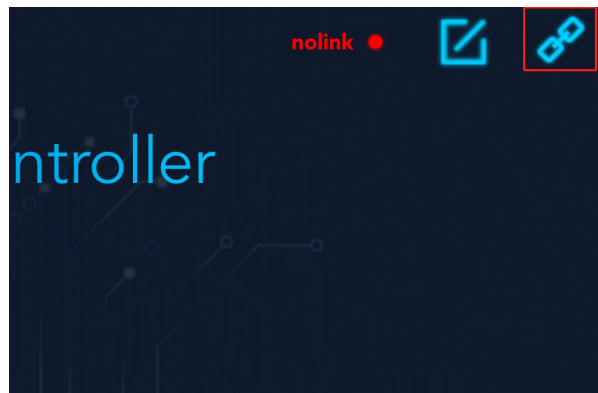
2. Then define the `SWITCH_MODE` variable as `sta`.

```
#define SWITCH_MODE "sta"
```

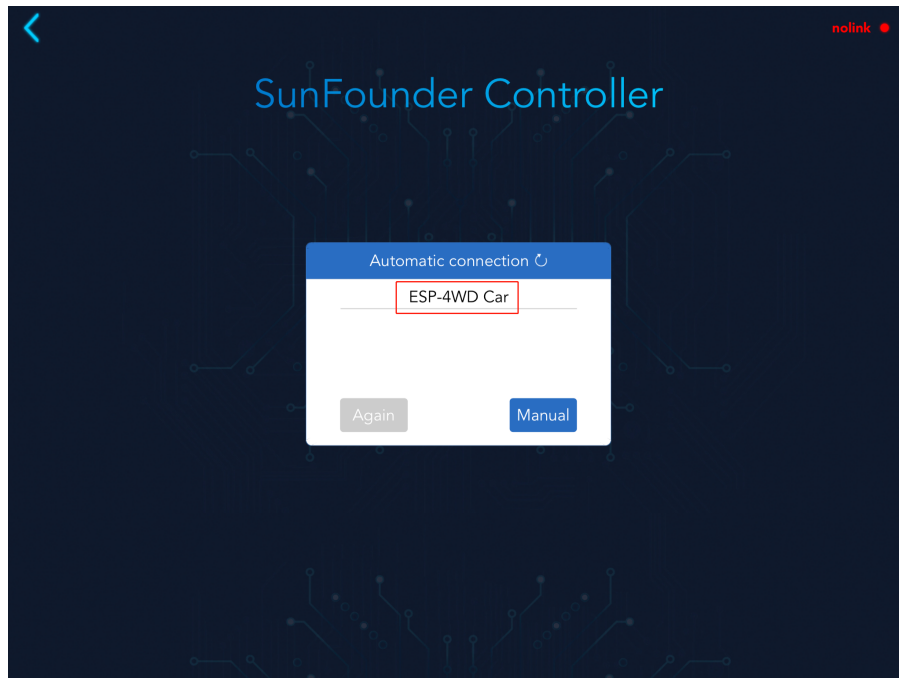
3. After downloading the code, ESP-4WD car will automatically connect to the wifi network, and at the same time take out your mobile device, open the WLAN management interface and connect to this wifi network.



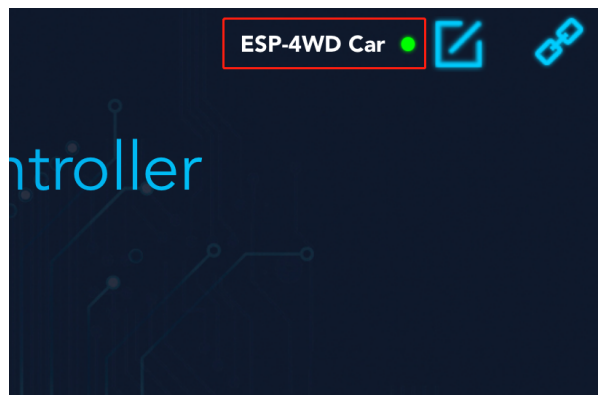
4. Open Sunfounder Controller and click the **Connect** icon on the top right corner.



5. Find the car name in the pop-up window and click on it.



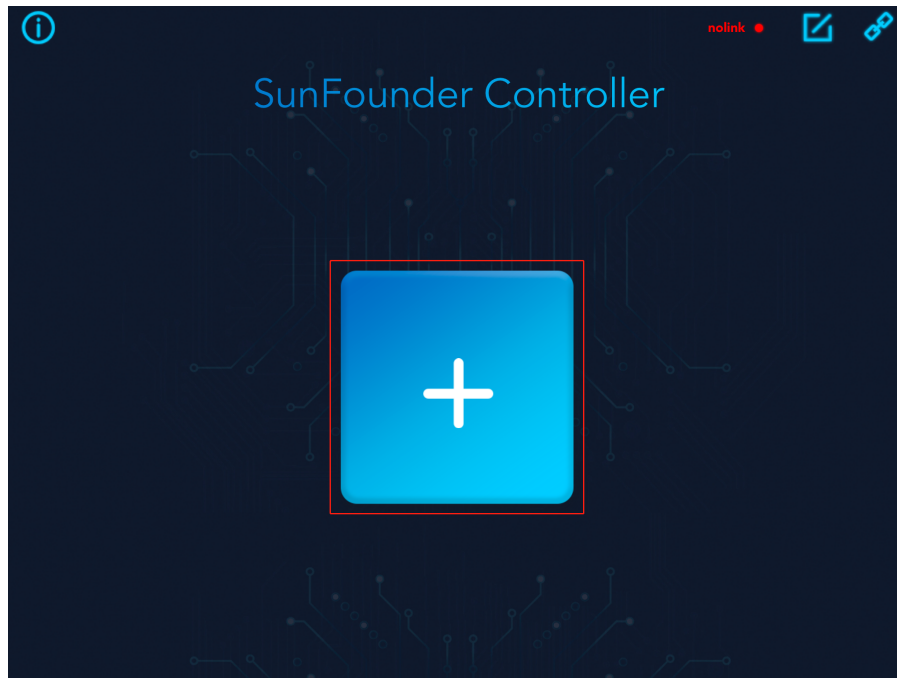
6. After connecting, the name of the car will be showed on APP.



Control the Car with APP

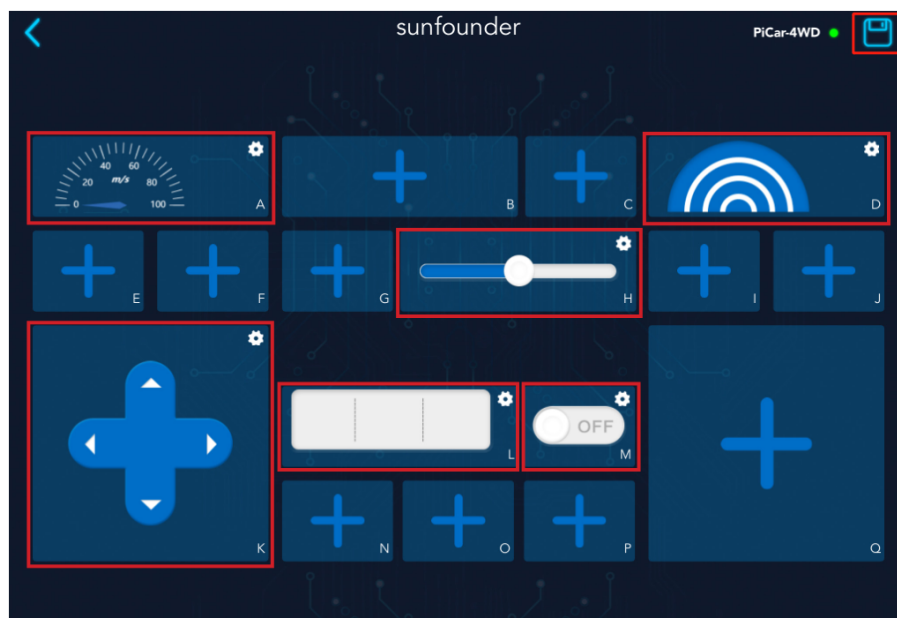
Either way, you can get the SunFounder Controller and ESP-4WD car to establish communication, next you will learn how to control the car with APP.

1. Open Sunfounder Controller, click the + to create an empty controller.

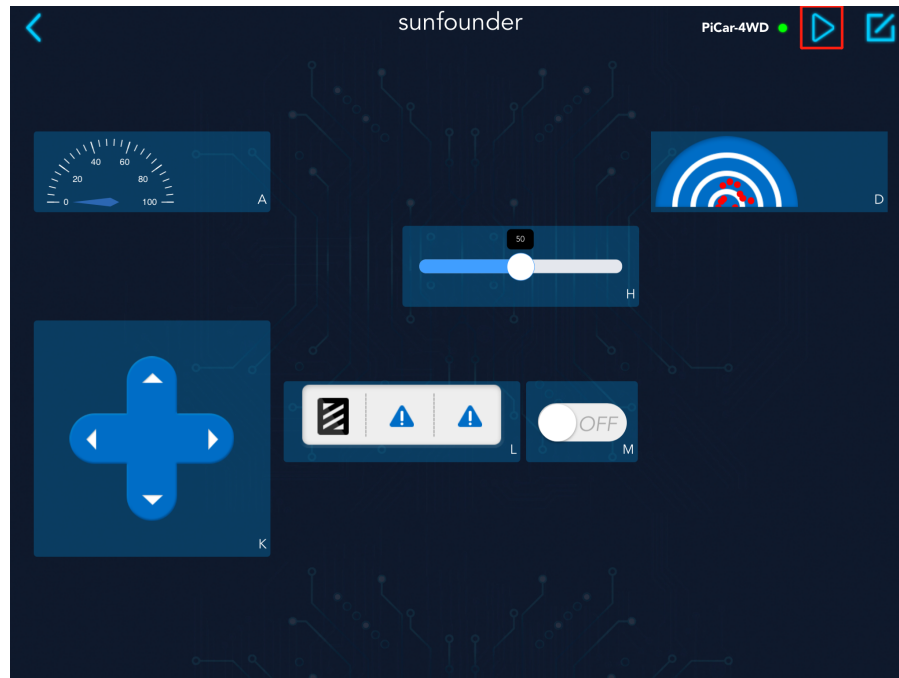


2. As shown in the figure, select the corresponding widget and click the icon in the upper right corner to save.

- **Widget A:** Show the driving speed of the car.
- **Widget D:** Simulate radar scanning.
- **Widget H:** Control the driving speed of the car.
- **Widget K:** Control the driving direction of the car.
- **Widget L:** Show the detection result of the grayscale sensor.
- **Widget M:** Control the on and off of the RGB board.



3. Click the start button in the upper right corner, and then try to use these widgets to control ESP-4WD car.



DIY Controller

If you want to DIY a new controller, you need to understand the communication process between the ESP-4WD car and the Sunfounder Controller. Open the `9.test_control.ino` file under the path `esp-4wd\Arduino\esp_rdp\examples\9.test_control`. You will go through this code to see how they communicate with each other.

Program framework

First, let us understand the general operating framework of the program.

Turn the code to line 103. In `setup()`, the `temp_data` variable defines the device information and proofreading information of ESP-4WD car, and sends it to Sunfounder Controller through the `deserializeJson()` function.

```
void setup() {
  String stringone = "{\"Name\":\"";
  String stringtwo = "\", \"Type\":\"ESP-4WD Car\", \"Check\":\"SunFounder_";
  ↪Controller\"}";
  temp_data = stringone + String(AP_NAME) + stringtwo;
  deserializeJson(doc_send, temp_data);
}
```

This `if` statement is used to determine the communication mode between ESP-4WD car and Sunfounder Controller. You can change the communication mode by modifying `SWITCH_MODE`.

```
if(SWITCH_MODE == "ap")
{
  WiFi.softAP(AP_NAME, AP_PASSWORD);
  ...
}
else if(SWITCH_MODE == "sta")
{

```

(continues on next page)

(continued from previous page)

```

    WiFi.begin(STA_NAME, STA_PASSWORD);
    ...
}

```

These two lines of statements indicate that the APP service starts running and enters the event processing process.

```

websocket.begin();
websocket.onEvent(onWebSocketEvent);

```

Then, we turn the code to line 26, `onWebSocketEvent()` is the event handling function, which uses the switch statement to determine the event type. The event types that the current program can trigger are `WStype_DISCONNECTED`, `WStype_CONNECTED`, and `WStype_TEXT`.

- `WStype_DISCONNECTED` is a disconnected event. The processing method is to print the disconnected information.
- `WStype_CONNECTED` is a connected event. The processing method is to print the connection information, and then send the device information to Sunfounder Controller.
- `WStype_TEXT` is a sending and receiving event, and we will process the received and sent string information in this event.

```

void onWebSocketEvent(uint8_t client_num,
                     WStype_t type,
                     uint8_t * payload,
                     size_t length) {
char output[300];
// Figure out the type of WebSocket event
switch(type) {

    // Client has disconnected
    case WStype_DISCONNECTED:
        Serial.printf("[%u] Disconnected!\n", client_num);
        break;

    // New client has connected
    case WStype_CONNECTED:
    {
        IPAddress ip = websocket.remoteIP(client_num);
        Serial.printf("[%u] Connection from ", client_num);
        Serial.println(ip.toString());
        websocket.sendTXT(client_num, temp_data);
    }
    break;

    case WStype_TEXT:
        ...

```

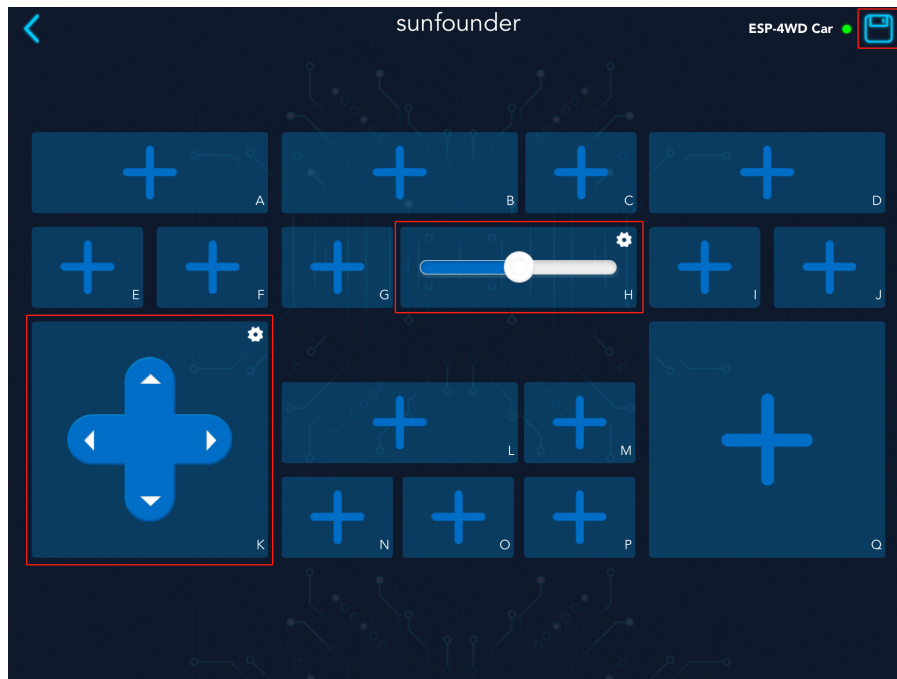
By modifying the content of the `WStype_TEXT` event, we can understand the data receiving and sending process between ESP-4WD car and Sunfounder Controller.

Receiving

The ESP-4WD car receives data from the Sunfounder Controller and sends its own sensor data to the Sunfounder Controller. Let's find out the data received by ESP-4WD car from Sunfounder Controller.

Step 1: Create new controller

Run the code, `9.test_control.ino`, re-establish communication, and then open Sunfounder Controller to create a new controller. We add a slider in the H area and a D-Pad in the K area. After adding, click the icon in the upper right corner to save.

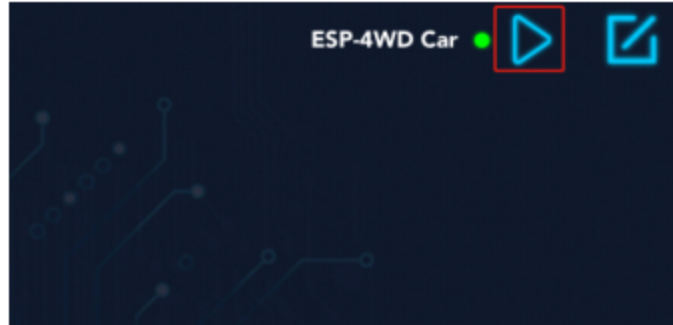


Step 2: Receive data from SunFounder Controller

Turn the code to line 53, in this `if` statement, we print out the string data (payload variable) received from Sunfounder Controller.

```
if(strcmp((char *)payload, temp_recv) != 0)
{
    memset(temp_recv, 0, 300);
    Serial.printf(" Received text: %s\n", payload);
    memcpy(temp_recv, (char *)payload, strlen((char *)payload));
}
```

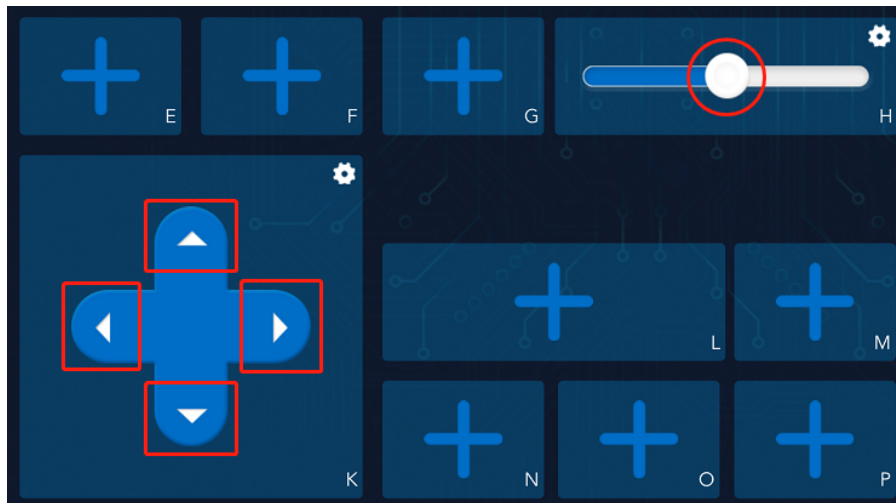
Click the start icon in the upper right corner to run the SunFounder Controller.



Open the Serial Monitor of arduino, we can find that the initial data of K control is the string `stop`, and the initial data of H widget is the int value 50.

```
Received text: {"A_region":null,"B_region":null,"C_region":null,"D_region":null,"E_region":null,"F_region":null,"G_region":null,"H_region":50,"I_region":null,"J_region":null,"K_region":"stop","L_region":[110,400],"M_region":false,"N_region":null,"O_region":null,"P_region":null,"Q_region":null}
```

Press the arrow keys of the **D-Pad** in the K area and slide the **slider** in the H area.



You can see that the D-Pad widget sends a string of data (“forward”, “backward”, “left”, “right”) to the ESP- 4WD car, while the slider widget will send an int data (range: 0-100).

```
Received text: {"A_region":null,"B_region":null,"C_region":null,"D_region":null,"E_region":null,"F_region":null,"G_region":null,"H_region":75,"I_region":null,"J_region":null,"K_region":"left","L_region":[110,400],"M_region":false,"N_region":null,"O_region":null,"P_region":null,"Q_region":null}
```

Step 3: Responding

When ESP-4WD car receives data from Sunfounder Controller, it needs to respond accordingly.

By modifying the content of the `WStype_TEXT` event, we can use the Sunfounder Controller widget to control the movement of the car. The K widget(D-Pad) controls the direction of the car, and the H widget(slider) controls the speed of the car.

Add the following code to line 62(a blank line).

```
car.move(doc_recv["K_region"], doc_recv["H_region"]);
```

After adding it, the content of the WStype_TEXT event is as follows. There are some commented contents not shown, please don't remove them.

```
case WStype_TEXT:
    if(strcmp((char *)payload, temp_recv) != 0)
    {
        memset(temp_recv, 0, 300);
        Serial.printf(" Received text: %s\n", payload);
        memcpy(temp_recv, (char *)payload, strlen((char *)payload));
    }
    deserializeJson(doc_recv, payload);
    car.move(doc_recv["K_region"], doc_recv["H_region"]);
    serializeJson(doc_send, output);
    if(strcmp(output, temp_send) != 0)
    {
        memset(temp_send, 0, 300);
        Serial.printf(" Send text: %s\n", output);
        memcpy(temp_send, output, strlen(output));
    }
    websocket.sendTXT(client_num, output);
    break;
```

- Before we understand the content of the WStype_TEXT event, please go to line 19, where there is a global variable doc_recv that can store the defined json object (similar to the structure of the C language) for receiving string data from Sunfounder Controller.

```
DynamicJsonDocument doc_recv(1024);
```

- Go back to WStype_TEXT event, through the function deserializeJson(), the variable payload (the string data received from the Sunfounder Controller) is converted into an operable variable doc_recv.

```
deserializeJson(doc_recv, payload);
```

- The value of doc_recv["K_region"] is the string data ("forward", "backward", "left", "right") sent by the K widget (D-Pad), the same as the value of doc_recv["H_region"] is the int data sent by H widget (slide) (range: 0-100).
- Pass doc_recv["K_region"] as the first parameter to the car.move() function to control the direction of the ESP-4WD car. Pass doc_recv["H_region"] as the second parameter to the car.move() function to control the speed.

```
car.move(doc_recv["K_region"], doc_recv["H_region"]);
```

After downloading the modified code and re-establishing communication, open the controller and click the start icon in the upper right corner to run the controller.

The D-Pad in the K area can control the direction of the ESP-4WD car, and the slider in the H area can control the speed.



Sending

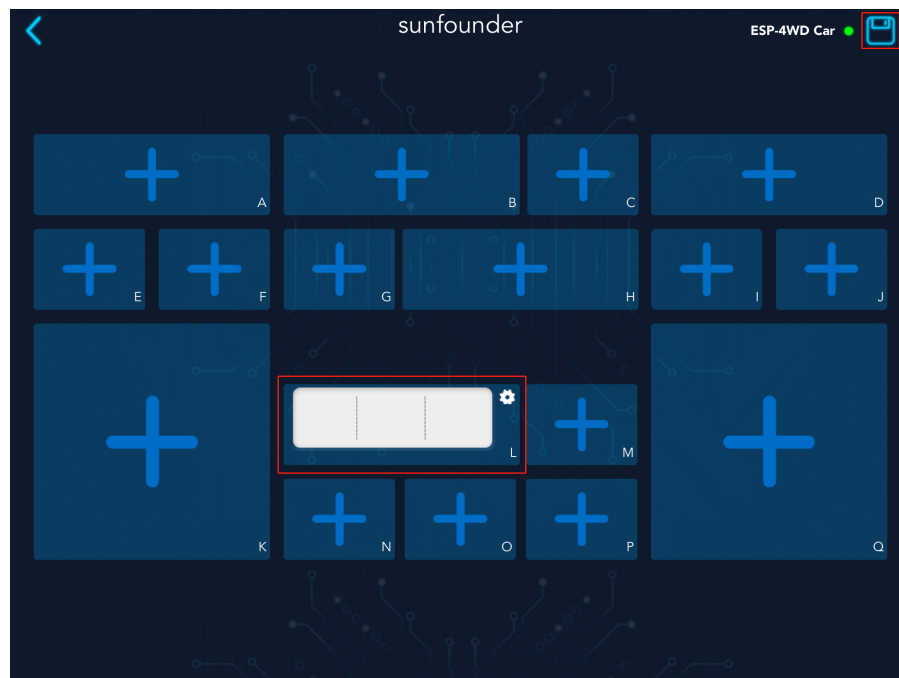
Let's take a closer look at how the ESP-4WD car sends its own sensor data to the Sunfounder Controller.

Step 1: Create new controller

Go back to WStype_TEXT event, cancel the comment in lines 67-71, to enable this code.

```
car.get_grayscale();
for(int i = 0; i < 3; i++)
{
    doc_send["L_region"][i] = car.adc_value[i];
}
```

Download this code, re-establish communication, and then open Sunfounder Controller to create a new controller. We add a grayscale detection tool in the L area. After adding, click the icon in the upper right corner to save.



Step 2: Send sensor data to SunFounder Controller

Turn the code to line 77. In this if statement, we print the variable `output` (the string information that stores the sensor data).

```
if(strcmp(output, temp_send) != 0)
{
    memset(temp_send, 0, 300);
    Serial.printf(" Send text: %s\n", output);
    memcpy(temp_send, output, strlen(output));
}
```

Click the start icon in the upper right corner to run the controller.



Open the Serial Monitor of Arduino, you will see that ESP-4WD car has sent device information, calibration information and grayscale sensor values to the Sunfounder Controller.

```
Send text: {"Name":"ESP-4WD Car","Type":"ESP-4WD Car","Check":"SunFounder
Controller","L_region":[304,336,243]}
```

Step 3: Responding

Let's write a piece of code that show the sensor data of ESP-4WD car on the widget of Sunfounder Controller. Widget L (grayscale detection tool) will show you the grayscale of the ground.

Let's re-explain the content of the WStype_TEXT event.

```
case WStype_TEXT:
    if(strcmp((char *)payload, temp_rcv) != 0)
    {
        memset(temp_rcv, 0, 300);
        Serial.printf(" Received text: %s\n", payload);
        memcpy(temp_rcv, (char *)payload, strlen((char *)payload));
    }
    deserializeJson(doc_rcv, payload);
    car.move(doc_rcv["K_region"], doc_rcv["H_region"]);
    car.get_grayscale();
    for(int i = 0; i < 3; i++)
    {
        doc_send["L_region"][i] = car.adc_value[i];
    }
    serializeJson(doc_send, output);
    if(strcmp(output, temp_send) != 0)
    {
        memset(temp_send, 0, 300);
        Serial.printf(" Send text: %s\n", output);
        memcpy(temp_send, output, strlen(output));
    }
    websocket.sendTXT(client_num, output);
    break;
```

- Before we understand the content of the `WStype_TEXT` event, please go to line 18, where the global variable `doc_send` defines an object that can store json objects(similar to the structure of the c language) for sending sensor data to the Sunfounder Controller.

```
DynamicJsonDocument doc_send(1024);
```

- Go back to the `WStype_TEXT` event, the `car.get_grayscale()` function is used to get the detection values of the three probes of the grayscale module. From left to right, they are `car.adc_value[0]` , `car.adc_value[1]` and `car.adc_value[2]` , which are stored in `doc_send["L_region"]`.

```
car.get_grayscale();
for(int i = 0; i < 3; i++)
{
    doc_send["L_region"][i] = car.adc_value[i];
}
```

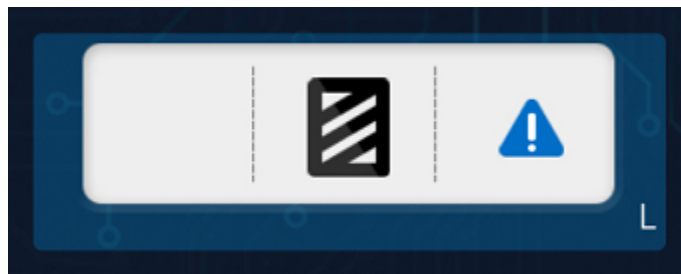
- Use the function `serializeJson()` to convert the variable `doc_send` (the stored sensor data) to the variable `output` (which is a string type recognized by the Sunfounder Controller).

```
serializeJson(doc_send, output);
```

- Then send the variable `client_num` (storing the device information and proofreading information of the ESP-4WD car) and variable `output` to Sunfounder Controller.

```
websocket.sendTXT(client_num, output);
```

Now, open the SunFounder Controller again, Widget D (grayscale detection tool) is showing the current ground conditions.



1.4 For MicroPython User

Welcome to the guide for the ESP-4WD car Kit (MicroPython version).

In this chapter, you will learn to burn firmware to the ESP32 RDP, install Thonny IDE, test all the components before assembling and then complete the assembly of the car. Once assembled, use the Arduino code to make the car do some interesting applications.

In addition you can use an APP, the SunFounder Controller, to visually control or access the individual data of the car.

1.4.1 Preparations

Before you start using the kit, you need to complete the following steps.

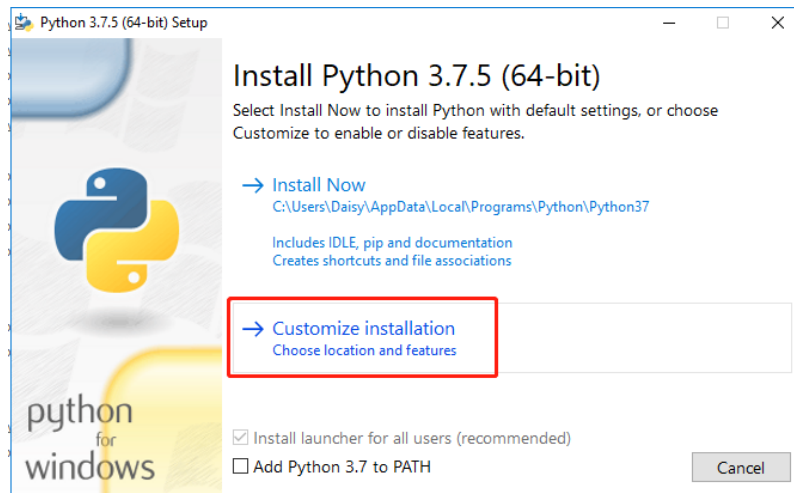
Burning Firmware

Install Python3

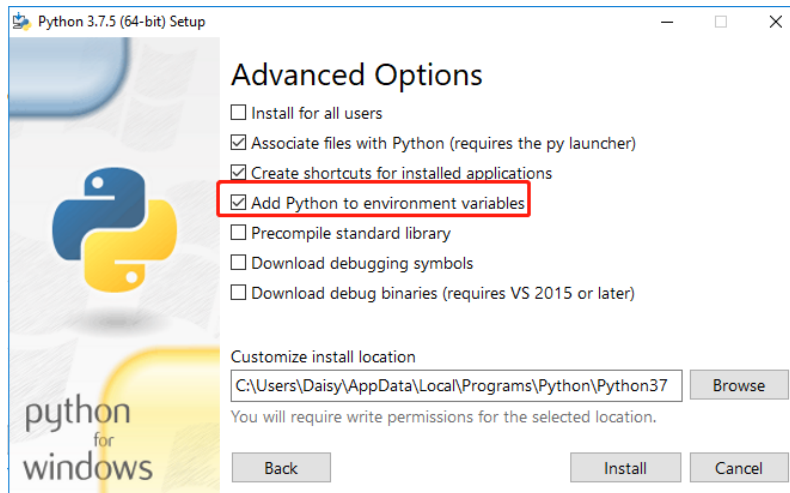
Go to the [Python official website](#) and choose the version of python3 for your PC. Many Linux and Mac OS X computers should have Python3 installed automatically.



Choose **Customize installation**.

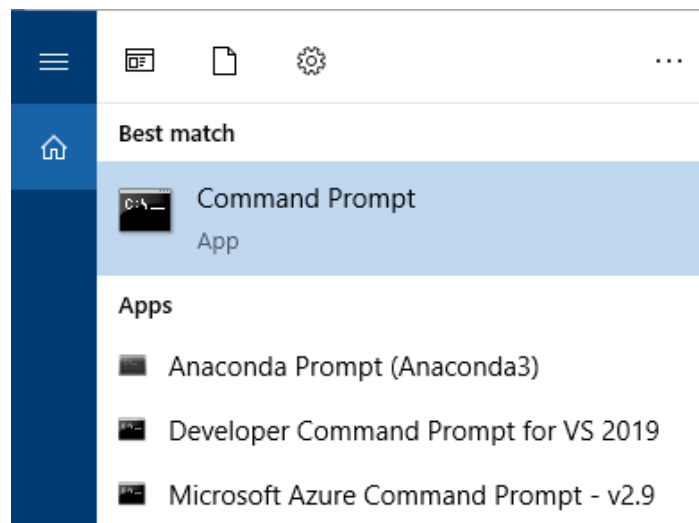


Then click Next, check “Add Python to environment variables” in Advanced Options.



Download Firmware Burning Tool

Open a Command Prompt (For windows users) or terminal (For Linux users).



Execute the following command to download esptool, a tool for burning firmware.

```
pip install esptool
```

Download the Firmware

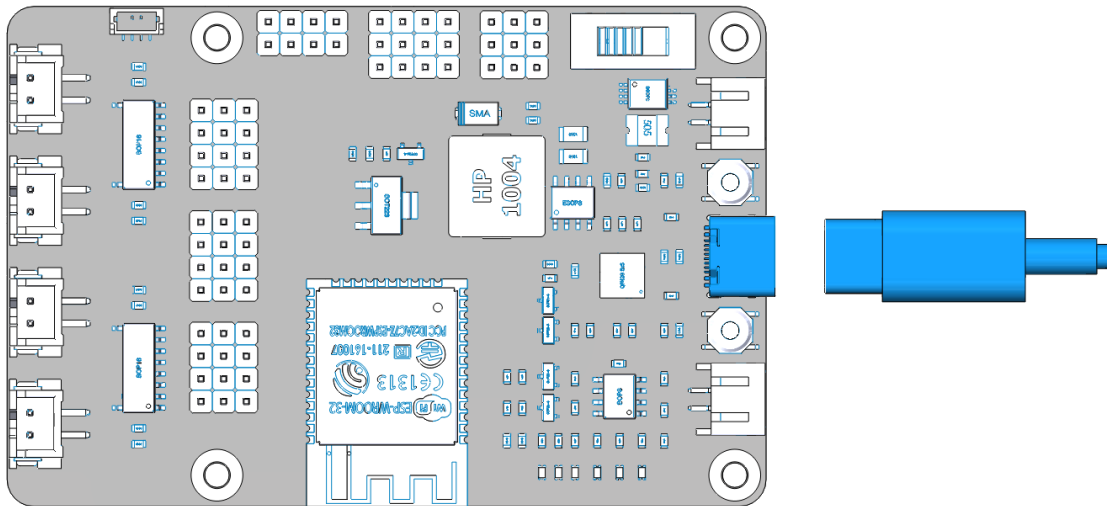
Go to the [MicroPython website](#). Download the general firmware to a local folder (The default path is C:/Users/username/Downloads), here is recommended to download **esp32-idf3-20200902-v1.13.bin**.

Note: If you are just starting with MicroPython, then the best choice is to choose **Stable** Firmware version. If you are an experienced MicroPython ESP32 advanced user, you can try the **unstable** version.

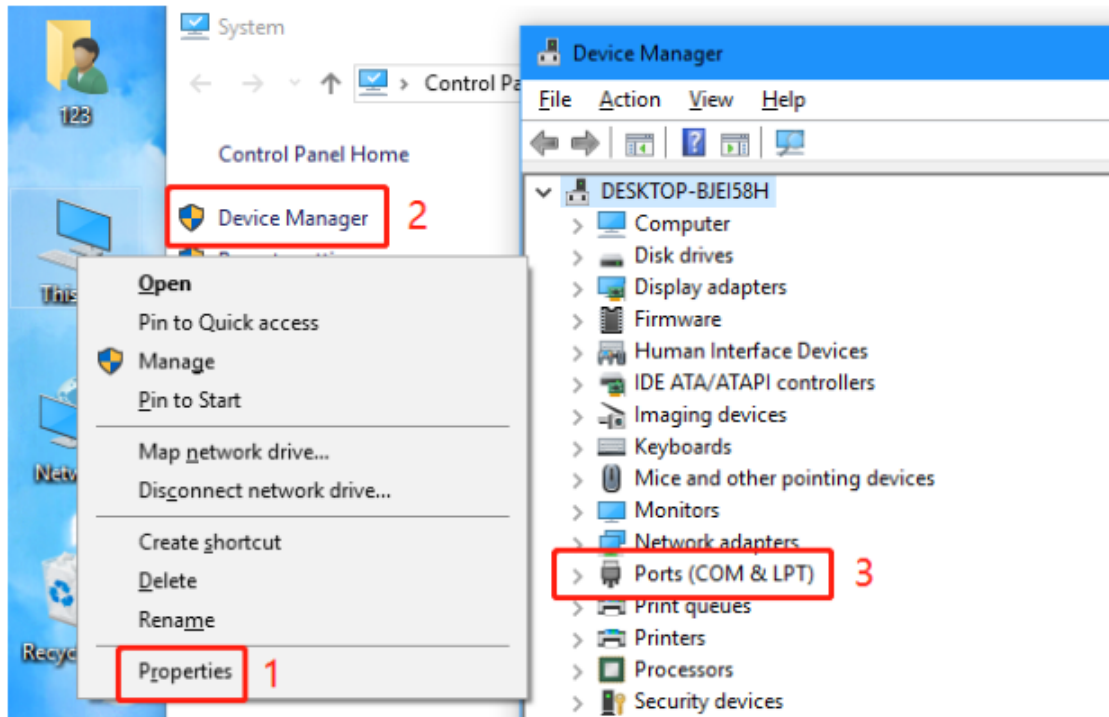
- GENERIC : esp32-idf3-20201207-unstable-v1.13-221-gc8b055717.bin
- GENERIC : esp32-idf3-20201202-unstable-v1.13-197-ga14ca31e8.bin
- GENERIC : esp32-idf3-20201201-unstable-v1.13-194-gf7225d1c9.bin
- GENERIC : esp32-idf3-20201130-unstable-v1.13-191-gee3706f4b.bin
- **GENERIC : esp32-idf3-20200902-v1.13.bin**
- GENERIC : esp32-idf3-20191220-v1.12.bin
- GENERIC : esp32-idf3-20190529-v1.11.bin
- GENERIC : esp32-idf3-20190125-v1.10.bin
- GENERIC : esp32-idf3-20180511-v1.9.4.bin

Install Driver

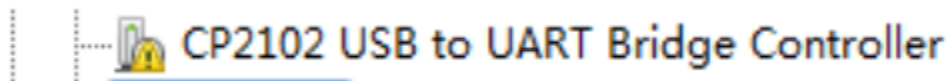
When you connect the ESP32 RDP board to the computer with a Type-C USB cable, the computer may not be able to recognize it. In this case, you need to install the driver manually.



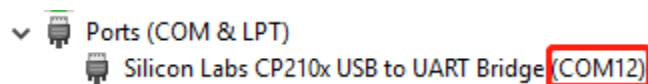
Right-click the **This PC** icon, and then click Properties -> Device Manager -> Ports to check the COM port information.



If the COM port(COMxx) does not appear, you need to download [CP210x USB to UART Bridge VCP Drivers](#) and install it.



Check the COM port information again. If the COM port(COMxx) can be displayed, the driver installation is successful.



Erase and Burn

Open the folder where you downloaded the firmware, the default path is C:/Users/username/Downloads, and execute the following command to erase the ESP32 RDP's flash.

```
esptool.py --port COM12 erase_flash
```

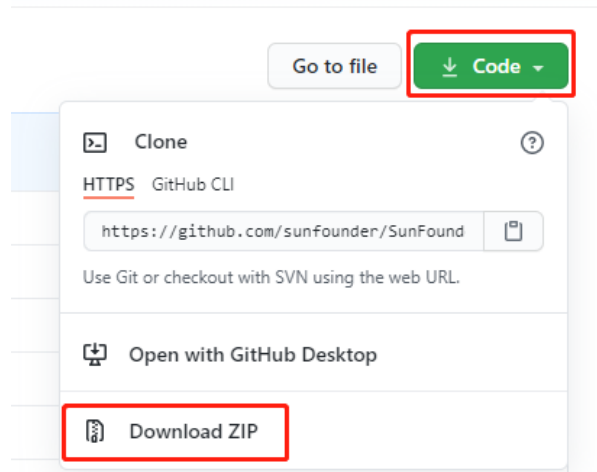
Then execute the following command to burn the firmware to the ESP32 RDP.

Note: Change the COM port and firmware name to yours.

```
esptool.py --chip esp32 --port COM12 write_flash -z 0x1000 esp32-idf3-20200902-v1.13.
↪bin
```

Download the ESP-4WD Package

Click [here](#) to download the ESP-4WD car kit codes. After unzipping the zip file you have downloaded, you will see all the relevant files for the ESP-4WD car kit.




Install Thonny

Thonny is an integrated development environment for running the Micropython code that controls the ESP-4WD car.

You can download it by visiting the [Thonny website](#). Once open the page, you will see a light gray box in the upper right corner, click on the link that applies to your operating system.

Thonny

Python IDE for beginners

 Download version **3.3.0** for
[Windows](#) • [Mac](#) • [Linux](#)

For the curious: [3.3.1](#)

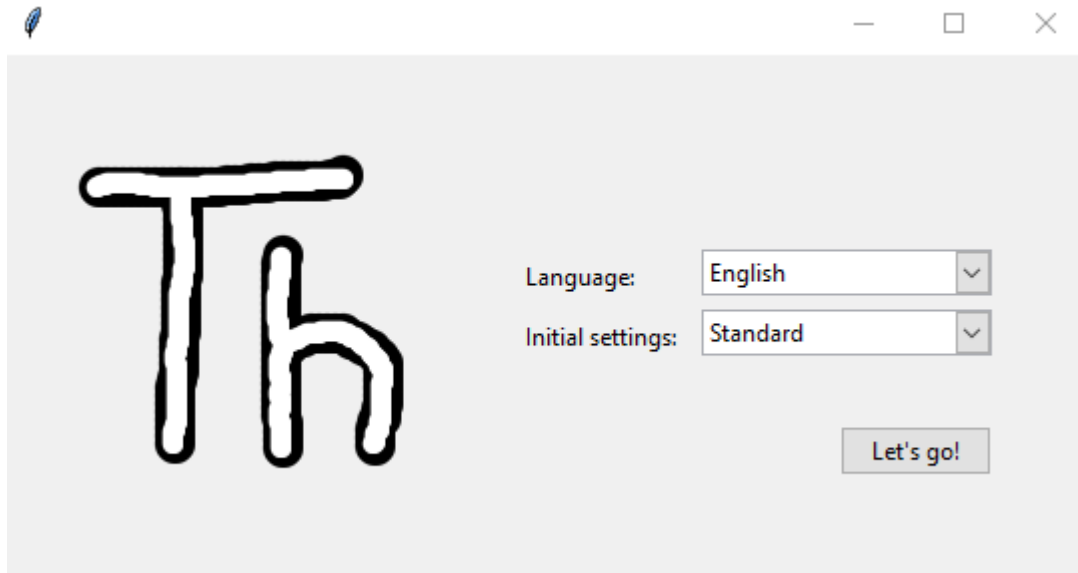
NB! Windows installer is signed with new identity and you may receive a warning dialog from Defender until it gains more reputation.

Just click "More info" and "Run anyway".

Open Thonny.

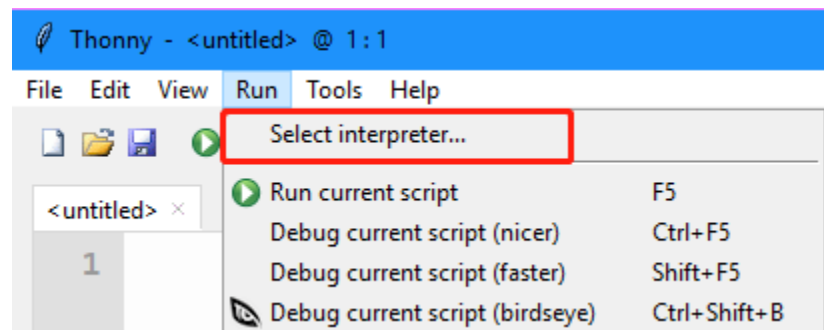


Select Language and Initial settings.

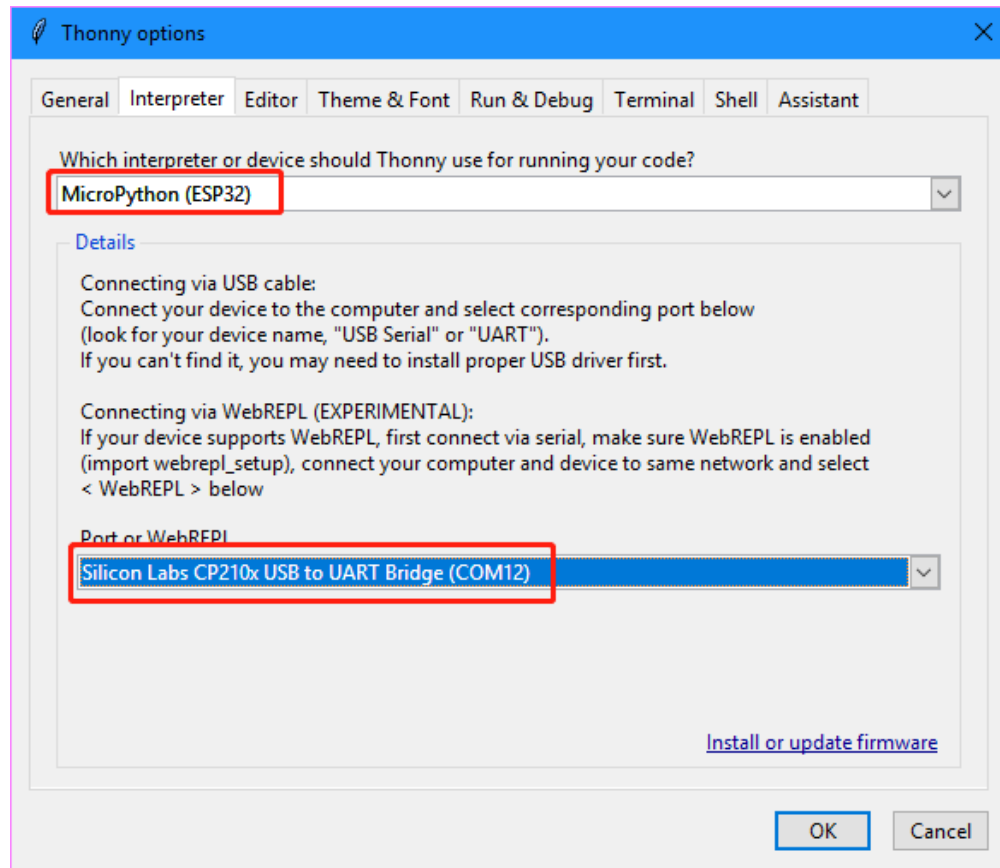


Upload Files to ESP32 RDP.

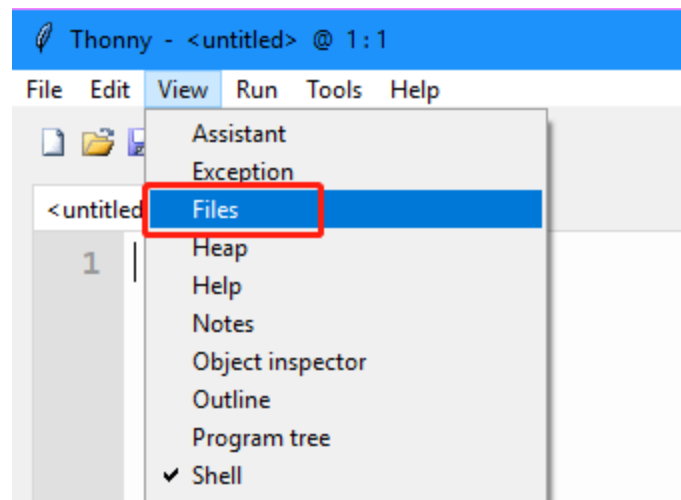
Click **Run** -> Select **interpreter**.



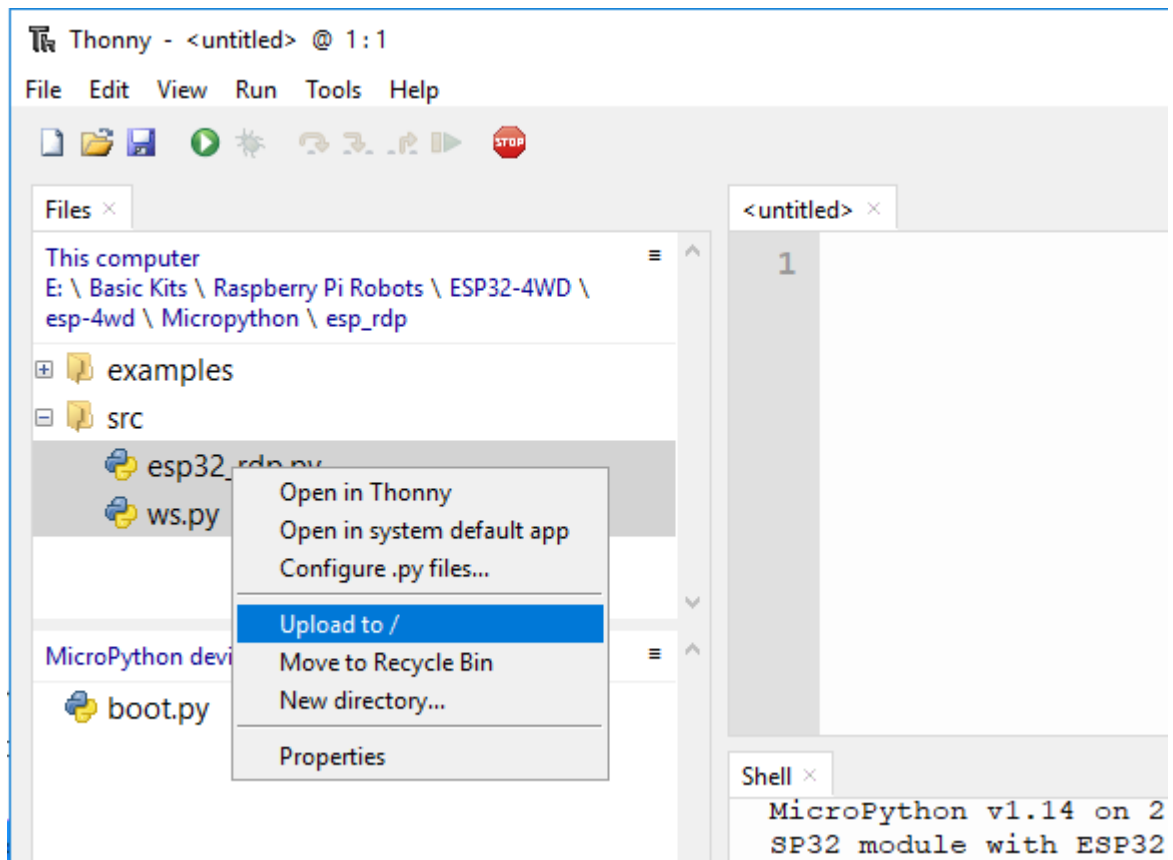
Select the interpreter **MicroPython (ESP32)** and then select COM port (the ESP32 RDP must be plugged into the computer first).



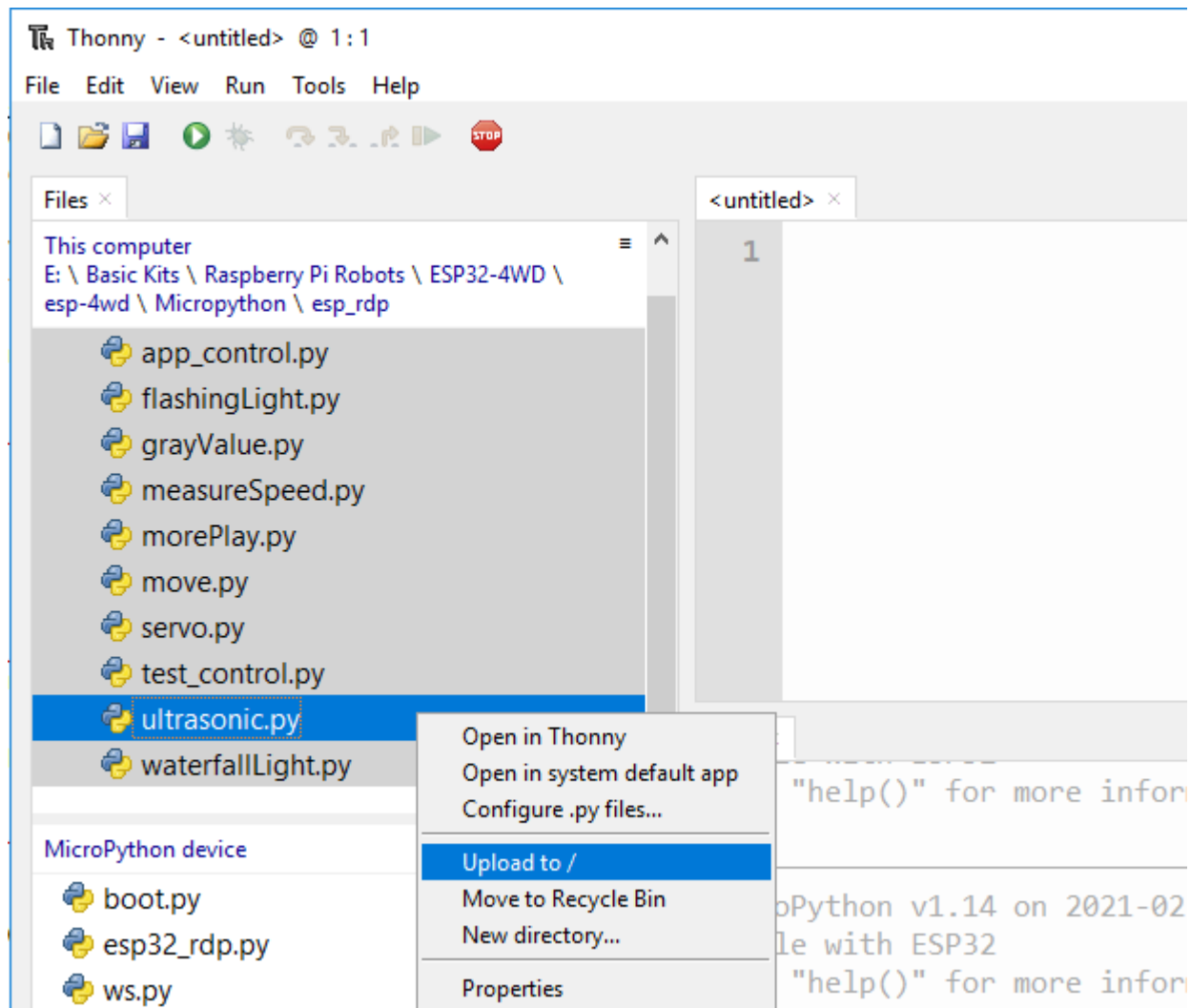
Click **View** -> **Files**.



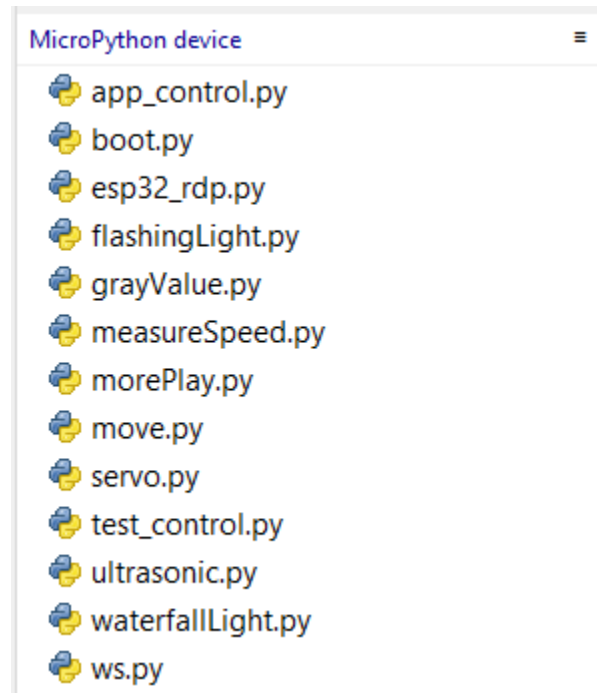
Go to the folder where you store the downloaded ESP-4WD package, find the files `esp32_rdp.py` and `ws.py` under the path `/esp-4wd/Micropython/esp_rdp/src`, then select both files and right click on them to upload them to the MicroPython device.



Use the same method to upload all Micropython files in the `examples` folder.



You can see the uploaded files in the MicroPython device window.



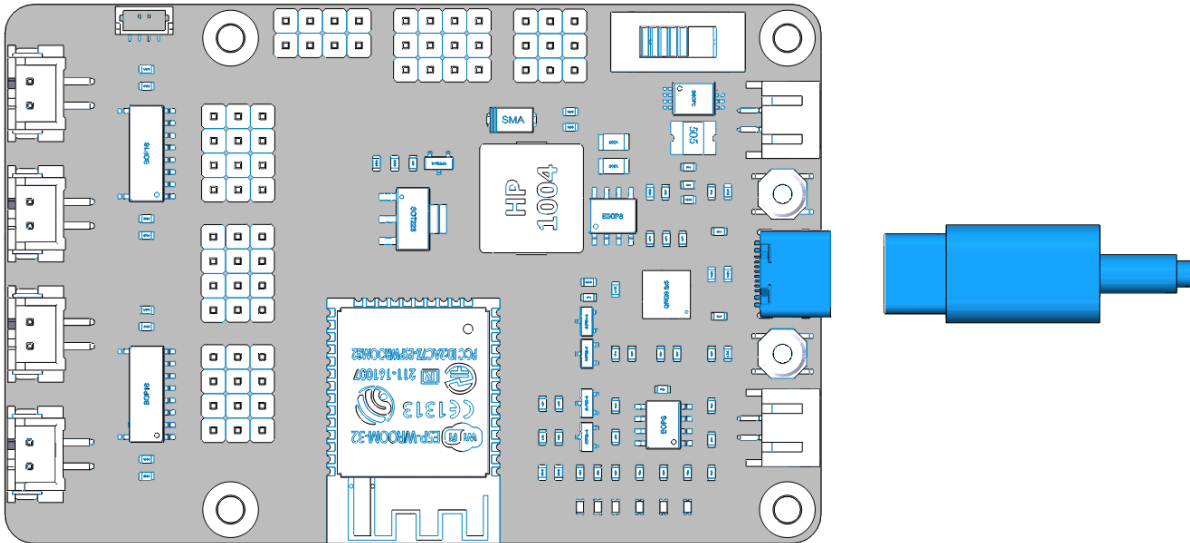
1.4.2 Test the Components with MicroPython

Before assembling the ESP-4WD car, you need to test each component to make sure it is working properly. If there are any problems with these components, please contact us.

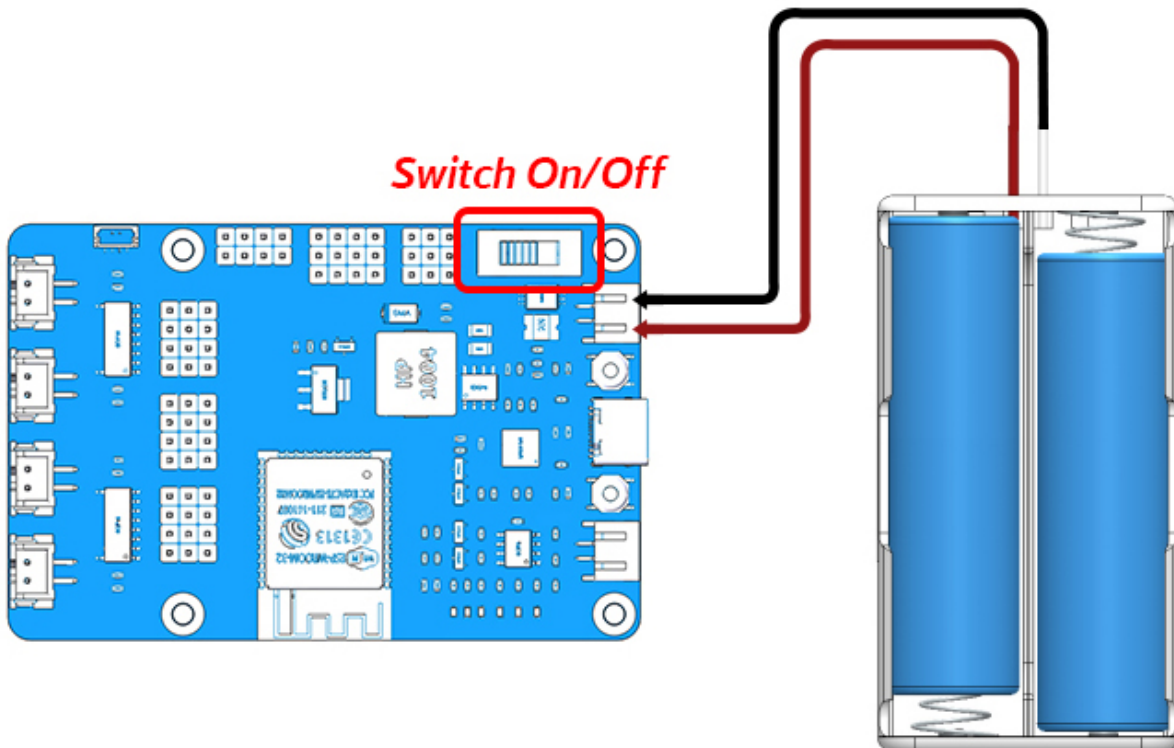
- *Power to ESP32 RDP*
- *Test the Motors*
- *Test the Ultrasonic Module*
- *Test the Grayscale Sensor Module*
- *Test the RGB Board*
- *Test the Servo*

Power to ESP32 RDP

In order to upload the code to the ESP32 RDP, you need to connect it to the computer with a Type-C USB cable.

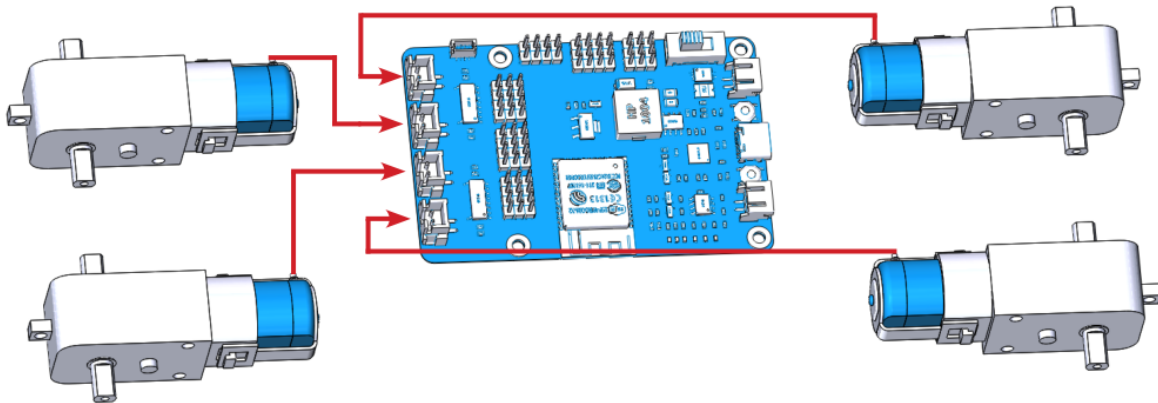


Then plug in the battery holder with two batteries to power other components and do not forget to slide the switch to ON.



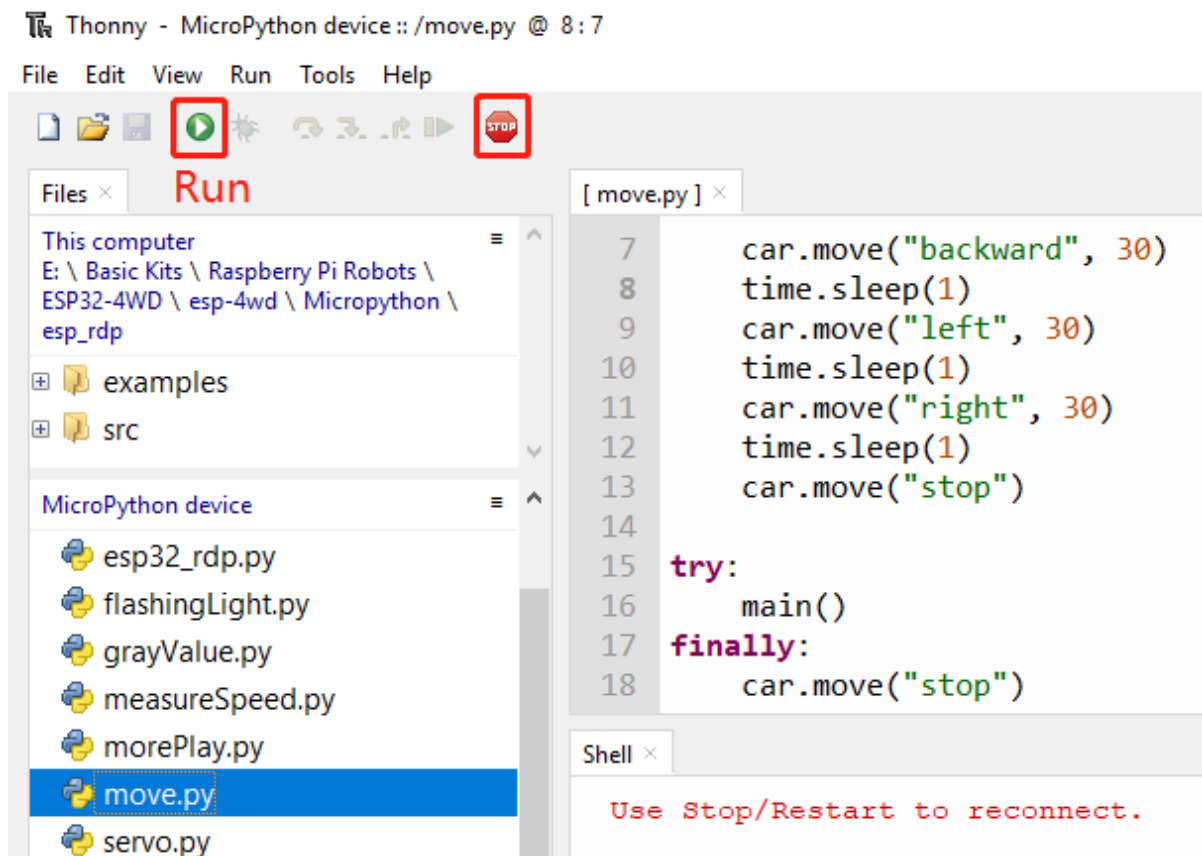
Test the Motors

Complete the wiring according to the diagram.



Double-click the `move.py` file in the MicroPython device window.

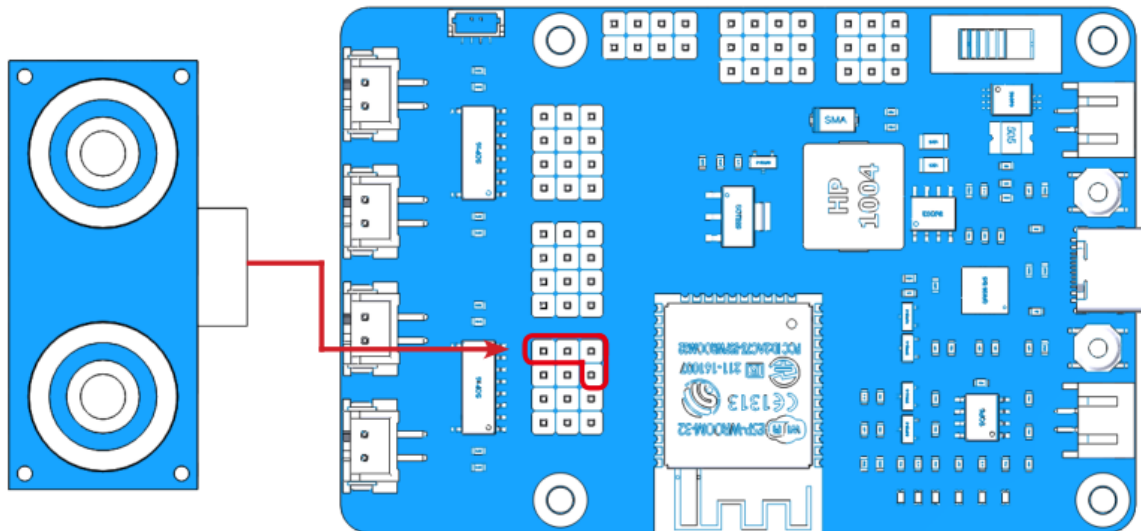
Click the **green play** icon at the top left to run the current script, and then click the **STOP** icon to Stop/Restart backend.



After running the code, you will see the four motors rotate and change the direction of rotation, and finally stop.

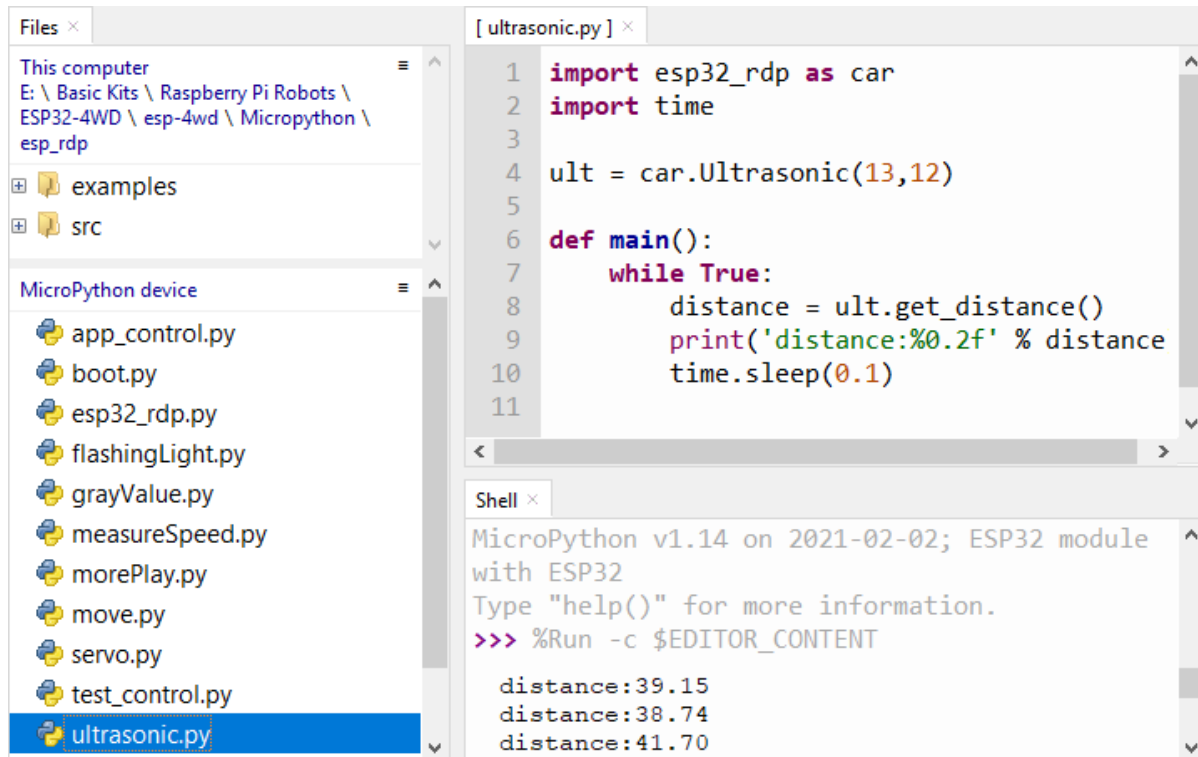
Test the Ultrasonic Module

Complete the wiring according to the diagram.



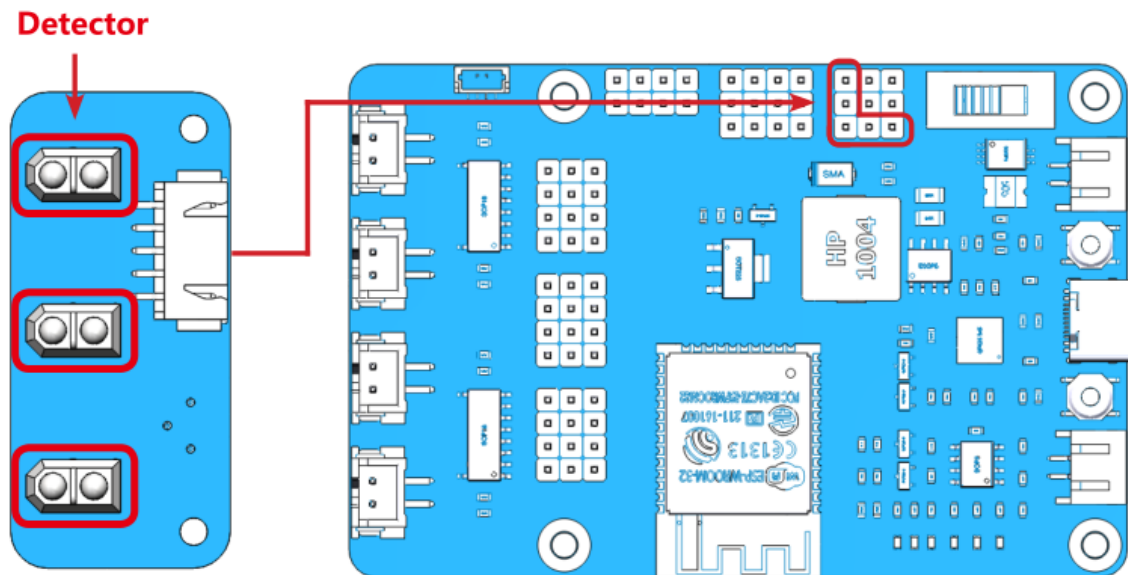
ULTRASONIC		ESP32 RDP
Trig	->	IO13
Echo	->	IO12
VCC	->	5V
GND	->	GND

Run `ultrasonic.py`, the Shell window under Thonny will always print the distance value read by the ultrasonic module.



Test the Grayscale Sensor Module

Complete the wiring according to the diagram.



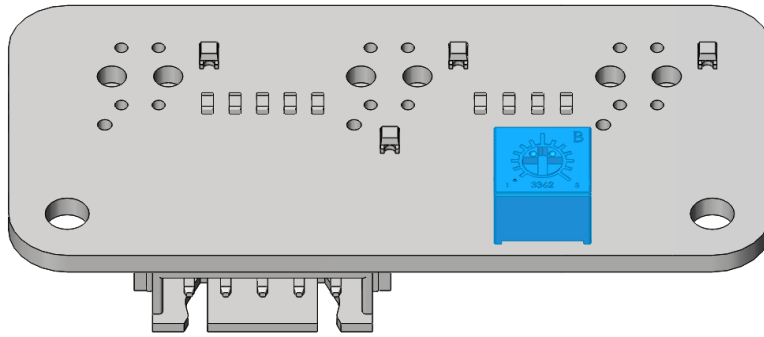
GRAYSCALE		ESP32 RDP
S0	->	IO35
S1	->	IO34
S2	->	IO36
3V3	->	3V3
GND	->	GND

Run `grayValue.py`, the Shell window under Thonny will always print the reading value of the grayscale sensor.

- Normally, it will detect a value above 1500 on white ground.
- On black ground, it will detect values below 900.
- On a cliff, it will detect a value below 110.
- If the reading is 0, it means that the height of the detector is higher than its detection range.

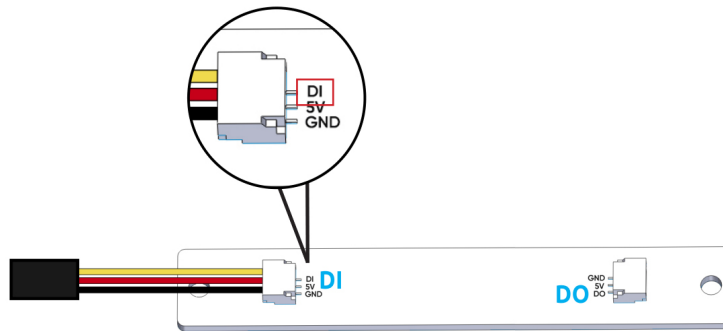
If the grayscale sensor module does not detect normal values, you will need to calibrate it. In order to make the calibrated value suitable for the assembled effect, the detector should be about **7mm** from the ground.

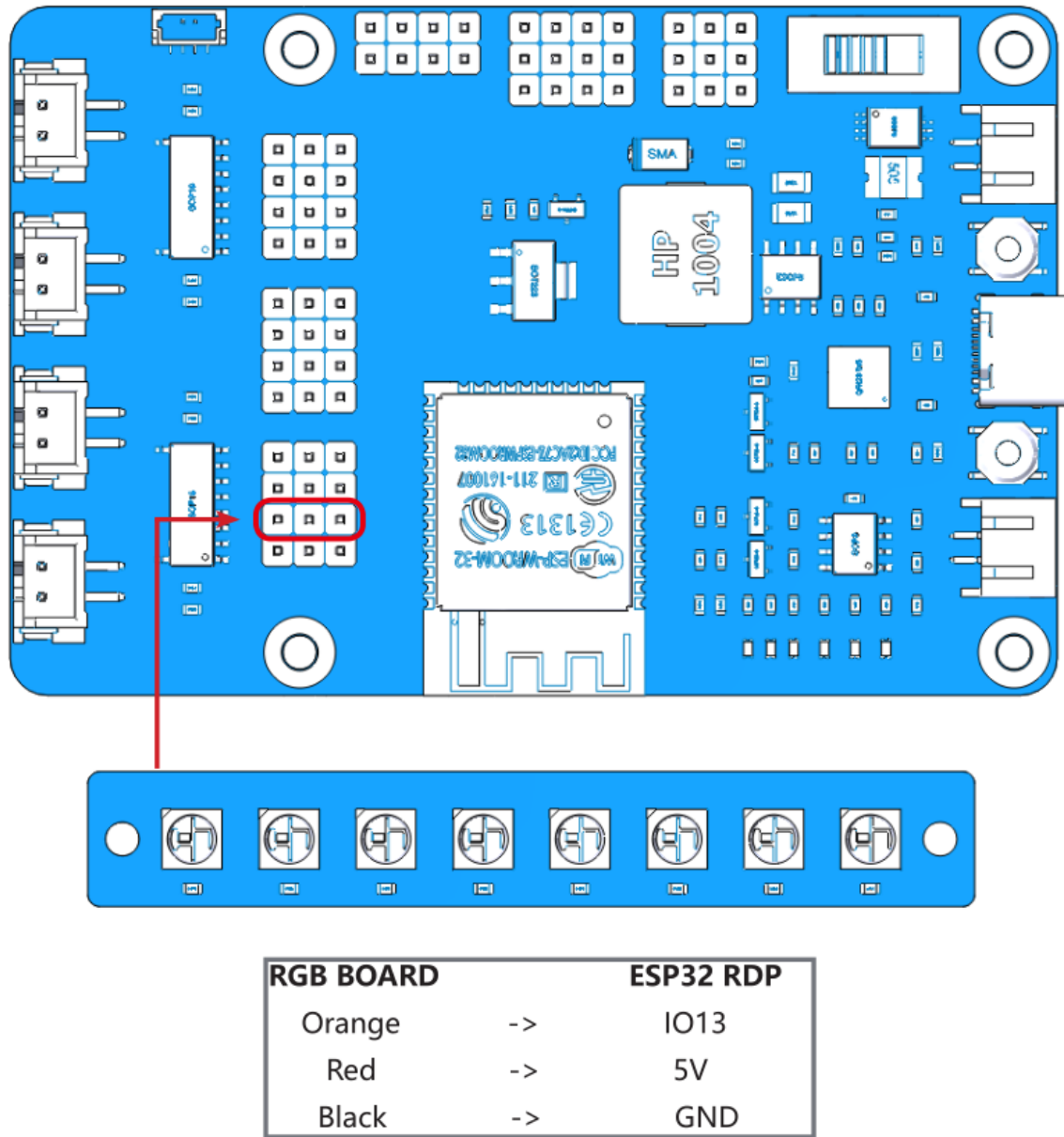
Now, Place it above the white ground and turn the potentiometer clockwise so that the reading is greater than 1500 (usually between 2000-4095). Then place it above the dark ground and turn the potentiometer counterclockwise to make it less than 900 (usually between 300 and 600). Repeat several times to get the maximum difference in both cases.



Test the RGB Board

Complete the wiring according to the diagram.

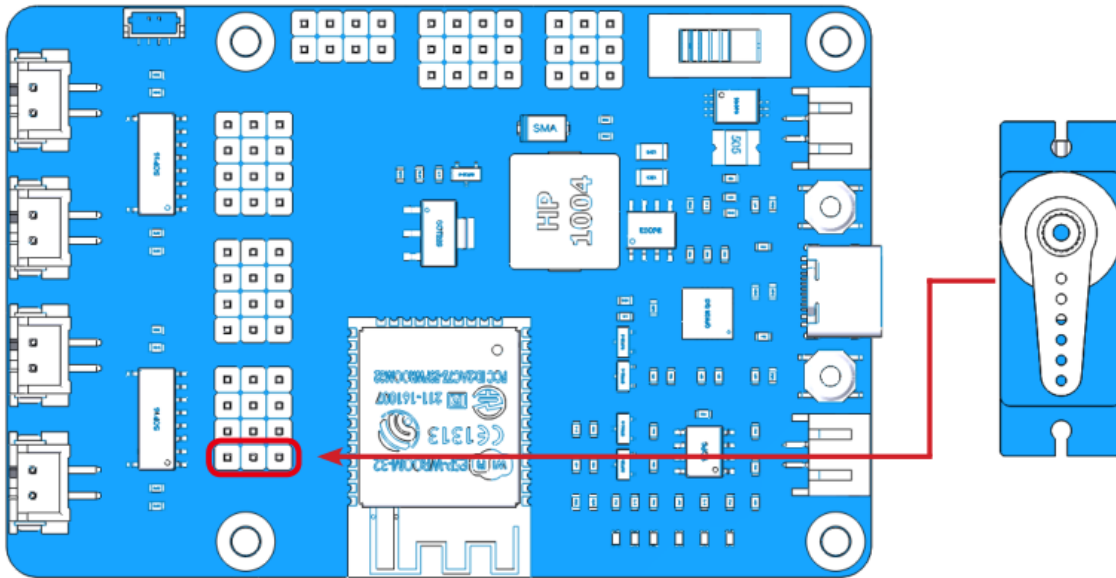




Run `flashingLight.py`, the RGB light under the car flashes every 0.5 seconds and changes color every time it flashes.

Test the Servo

Complete the wiring according to the diagram.

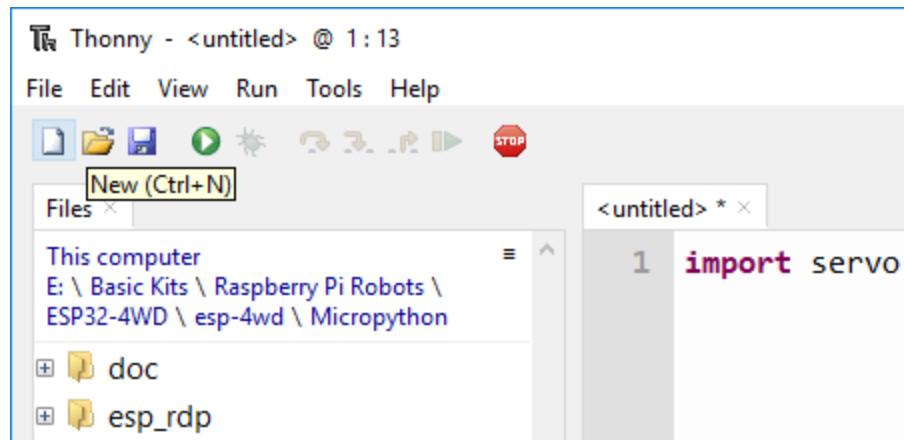


SERVO		ESP32 RDP
Orange	->	IO27
Red	->	5V
Brown	->	GND

Insert a rocker arm into the servo shaft, then run `servo.py`. the servo will first turn left 30 degrees, then turn right 30 degrees, and finally back to 0 degrees.

Note: In the car assembly process in the next chapter, the servo must be kept at 0 degrees. Here, we provide a way to make ESP32 RDP run `Servo.py` automatically when power on.

Click the new file icon in the upper left corner.

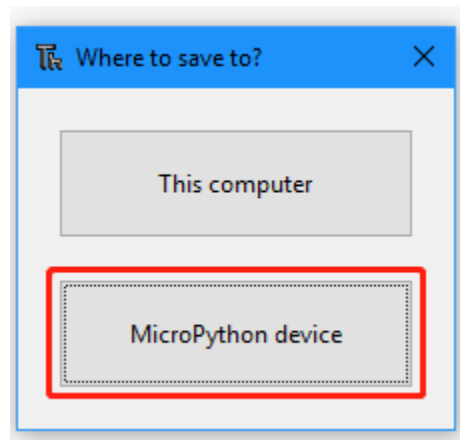


Enter the following code.

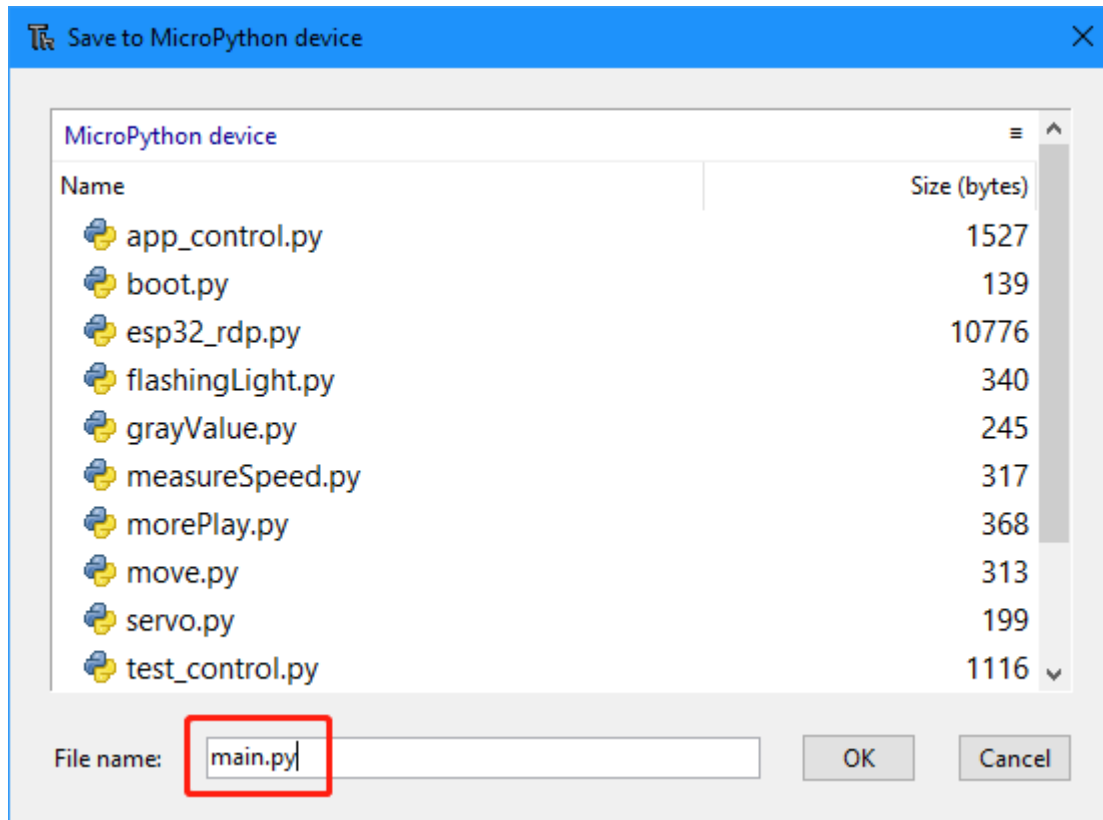
```
import servo
```

Note: You can replace servo with the name of other example code, depending on which code you want to run automatically when power is on.

Click the save button in the upper left corner, and select **MicroPython device** in the pop-up option box.



Name the file **main.py** .



Unplug the data cable and re-power the main control board. **main.py** will run automatically.

1.4.3 Hardware Assembling

In this chapter, you will start the hardware assembly of the ESP-4WD car. Before starting the assembly, make sure your device is in proper operation and calibrate the servo to the 0 degree position.



1.4.4 Play the Car with MicroPython

There are two ways to play for Micropython users to learn and use ESP-4WD car: Code Control and APP Control. You can directly use Micropython code to control the car, or DIY a remote control.

It should be noted that before using APP Control, you need to understand Code Control first.











Note: In the last two chapters, we stored a main.py file() for the ESP32 RDP, which allows the servo to automatically rotate to 0 degrees when powered up. Now that the ESP-4WD car is assembled, you can now delete main.py or replace the import name with other example's name.

Code Control

In this chapter you will learn to control the ESP-4WD car with MicroPython codes.

You can copy the code below into the Arduino IDE or directly open the code file under the path `esp-4wd\Micropython\esp_rdp\examples`.

The following code files have been used in the chapter *Test the Components with MicroPython*, just for simple testing of motor, ultrasonic module, Grayscale Sensor Module, 8-bit RGB board and Servo.

ID > esp-4wd > Micropython > esp_rdp > examples	
Name	Date modified
 app_control.py	4/19/2021 9:20 AM
 flashingLight.py	4/19/2021 9:20 AM
 grayValue.py	4/19/2021 9:20 AM
 measureSpeed.py	4/19/2021 9:20 AM
 morePlay.py	4/19/2021 9:20 AM
 move.py	4/19/2021 9:20 AM
 servo.py	4/19/2021 9:20 AM
 test_control.py	4/19/2021 9:20 AM
 ultrasonic.py	4/19/2021 9:20 AM
 waterfallLight.py	4/19/2021 9:20 AM

Here, we focus on the `measureSpeed.py`, `waterfallLight.py` and `morePlay.py`.

In addition, the `app_control.py` and `test_control.py` code files are used in the *APP Control* chapter, and will not be explained here.

waterfallLight

After running `waterfallLight.py`, the LEDs on the 8-bit RGB board under the car will turn on in random colors from the first to the 24th, and then turn off one by one from both ends to the middle.

After that, the RGB lights will turn on in random colors from the 24th to the first, and then turn off one by one from both ends to the middle.

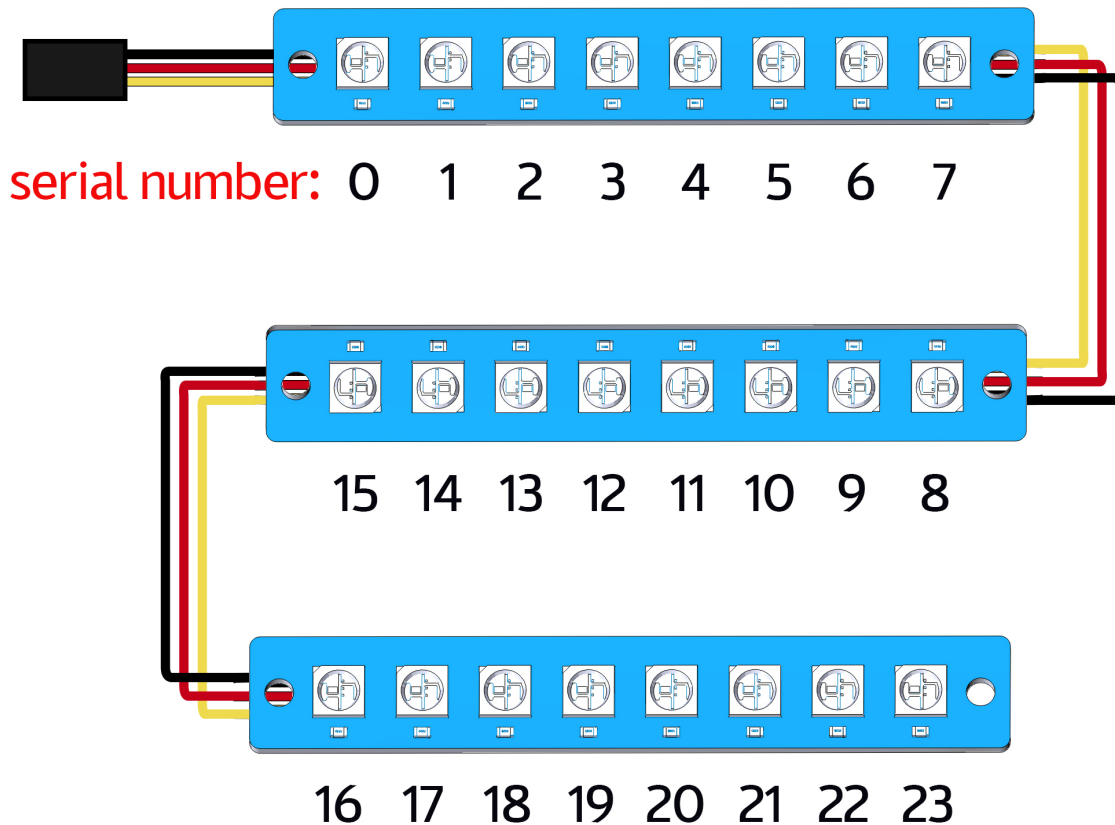
```
import esp32_rdp as car
import random
import time

def main():
    while True:
        color = [random.randint(0,255), random.randint(0,255), random.randint(0,255)]
        noColor = []
        for i in range(24):
            car.set_num_color(i, color)
            time.sleep_ms(20)
        for i in range(23,11,-1):
            j = 23-i
            car.set_num_color(i, noColor)
            car.set_num_color(j, noColor)
            time.sleep_ms(40)
        for i in range(23,0,-1):
            car.set_num_color(i, color)
            time.sleep_ms(20)
        for i in range(23,11,-1):
            j = 23-i
            car.set_num_color(i, noColor)
            car.set_num_color(j, noColor)
            time.sleep_ms(40)

    try:
        main()
    finally:
        car.set_light_off()
```

The sentence to light up the LED is `car.set_num_color(i, red, green, blue)`; the first parameter is the number of the light, and the last three parameters are the RGB value.

For example, `car.set_num_color(4, 255, 0, 0)` means to make the No. 4 LED light up in red.



measureSpeed

Run `measureSpeed.py`, the car will move at a random speed, and the 2-ch Photo- interrupter Module will detect the speed of the car.

The light emitted from the transmitting end of the 2-ch Photo-interrupter module to the receiving end will pass through the Encoding Disk (with 20 holes). When the receiving end does not receive the light, it will send a 0 to the microcontroller, otherwise it will send a 1.

This means that when a total of 20 1 are detected, the wheel of the car has turned one round (a distance of the wheel circumference has been traveled forward).

In the same way, we can detect the frequency of the 1 received by the microcontroller and calculate the speed of the car in cm/s.

```
import esp32_rdp as car
import random
import time

speed = car.Speed(26, 25)

def main():
    while True:
        car.move("forward", random.randint(0, 100))
```

(continues on next page)

(continued from previous page)

```
        time.sleep(1)
        carSpeed = speed.get_speed()
        print('distance:%d' % carSpeed)

try:
    main()
finally:
    car.move("stop")
```

morePlay

Run `morePlay.py`, this example provides 4 ways to play ESP-4WD car. You can switch between different modes by modifying the value of the variable `mode`.

```
import esp32_rdp as car

mode = 1

def main():
    while True:
        global mode
        if mode == 1:
            car.avoid(40,30)
        elif mode == 2:
            car.follow(40,30)
        elif mode == 3:
            car.is_on_edge(110)
        elif mode == 4:
            car.track_line(400,50)

try:
    main()
finally:
    car.move("stop")
```

Function Introduction of morePlay

Note: For how the following 4 functions implement the corresponding functions, please refer to `esp32_rdp.py` under the path `esp-4wd\Micropython\esp_rdp\src`.

avoid()

The default mode(mode=1) is obstacle avoidance.

`avoid(int ref, int speed)`

- `ref` refers to the reference distance value.
- `speed` refers to the forward speed.

ESP-4WD car will move forward at 30% speed and return the distance state according to the obstacle in front.

- If distance > 40, return the distance state 2, if distance > 10, then return 1, otherwise it will return 0.

- If the obstacle distance is greater than 40, the car will move forward, otherwise it will turn right.

follow()

Modify the value of the mode variable to 2 so that the mode is set to follow.

```
follow(int ref, int speed)
```

- `ref` refers to the reference distance value.
- `speed` refers to the forward speed.

ESP-4WD car will move forward at 30% speed and automatically follow objects within 40cm in front.

is_on_edge()

Modify the value of the mode variable to 3 so that the mode is set to cliff detection.

```
is_on_edge(int ref)
```

- `ref` refers to the reference gray value.

When ESP-4WD car detects a cliff (a place where the grayscale sensor's detection value is below 110), it will retreat a certain distance.

track_line()

Modify the value of the mode variable to 4 so that the mode is set to track line.

```
track_line(int ref, int speed)
```

- `ref` refers to the reference gray value.
- `speed` refers to the forward speed.

Note: You can replace ``ref``(400) with another number, which is the threshold between the black line and the white ground read by the grey scale sensor.

The ESP-4WD car moves along the black line on the white ground.

APP Control

In this chapter, you will learn to use a APP - Sunfounder Controller to control the car.

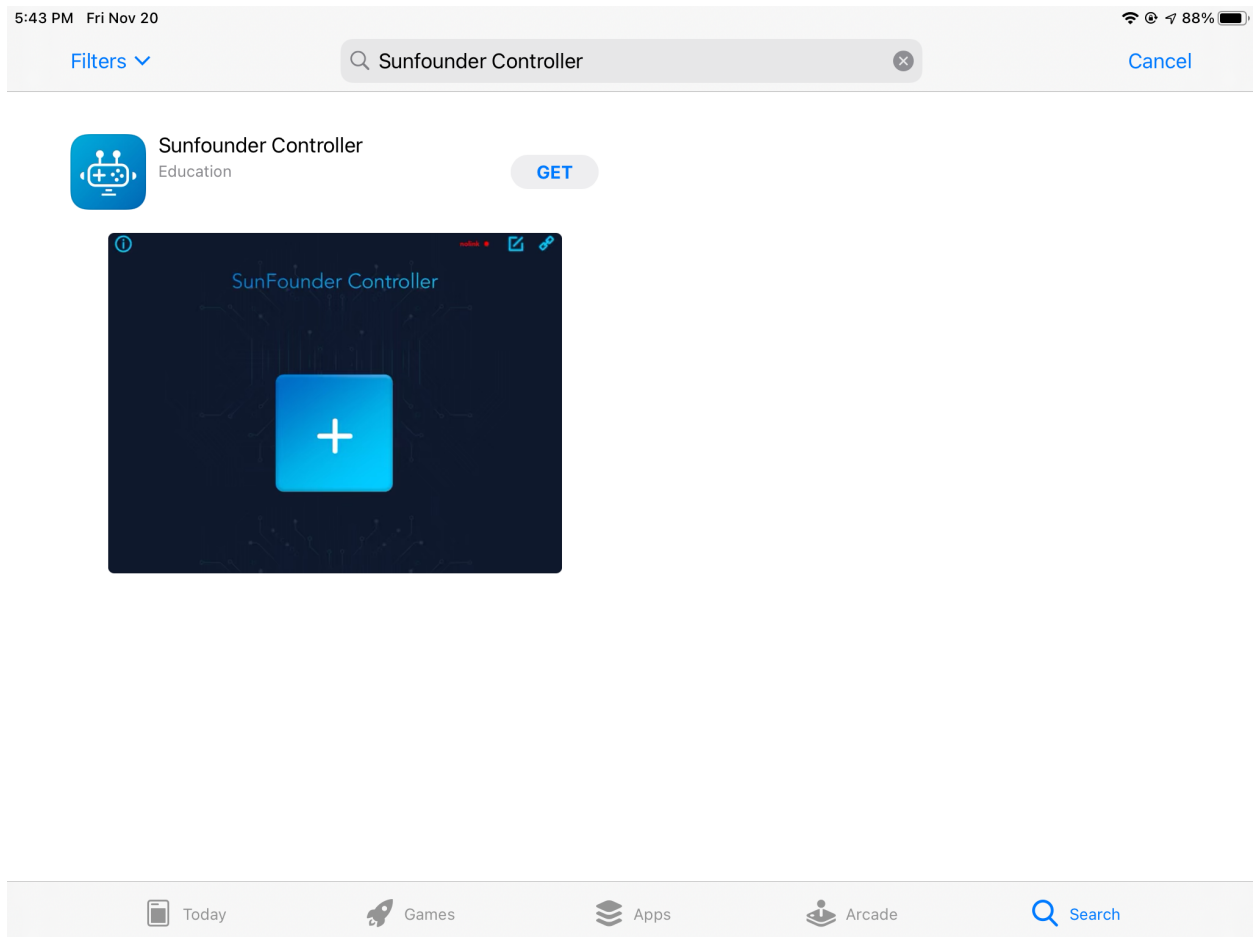
The complete operation process is as follows **Install Sunfounder Controller -> Establish Communication -> Control the Car with APP.**

You can check the **About Sunfounder Controller** and **DIY Controller** sections according to your choice.

- *Install Sunfounder Controller*
- *About Sunfounder Controller*
- *Establish Communication*
- *Control the Car with APP*
- *DIY Controller*

Install Sunfounder Controller

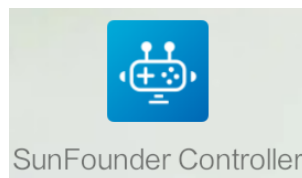
Open App Store (iOS/Mac OS X system) or Play Store (Android/Windows/Linux system), then search and download Sunfounder Controller.



About Sunfounder Controller

Page Introduction

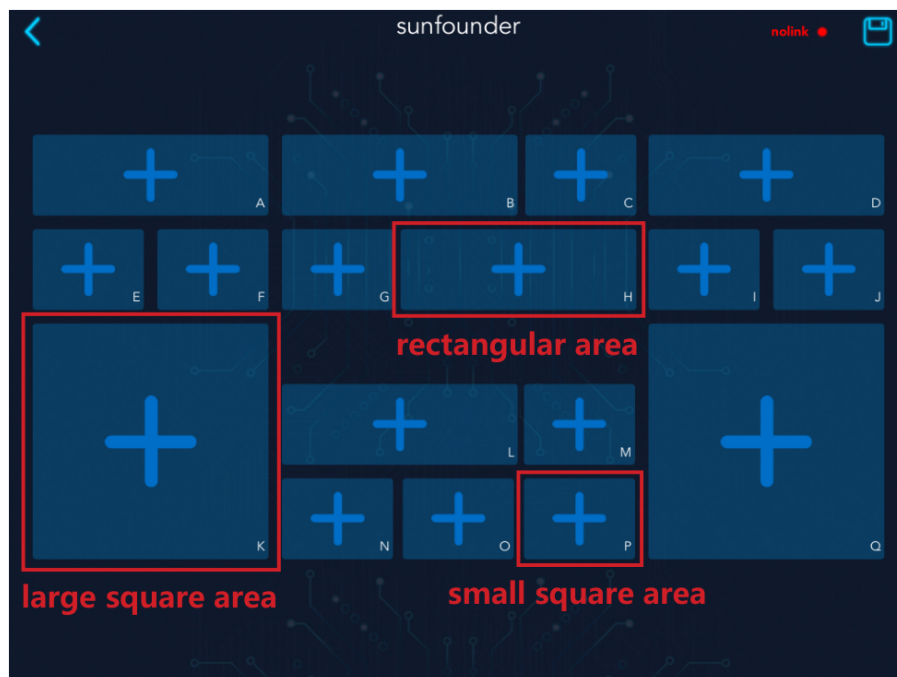
Start the Sunfounder Controller.



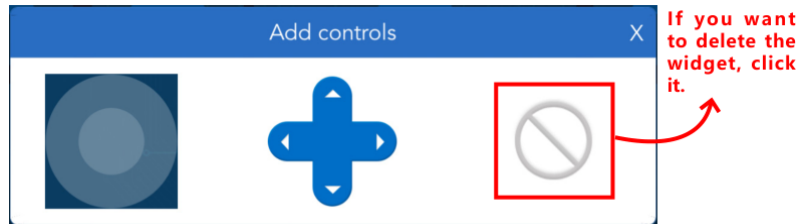
Click the middle button to add a new controller.



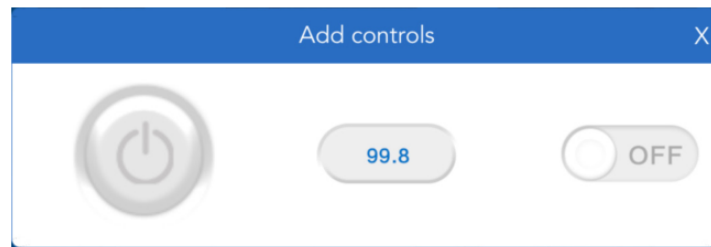
Sunfounder Controller is a platform that can add custom controllers. It reserves many widget interfaces. There are a total of 17 areas from A to Q. Each area has selectable widgets.



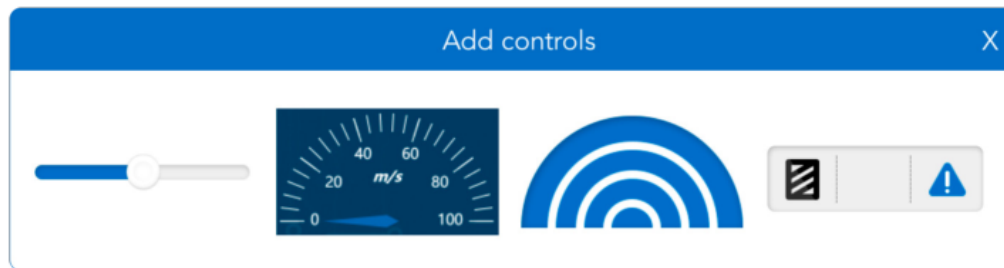
The available widgets in the **large square area** include joystick and D-Pad.



The available widgets in the **small square area** include button, digital display and switch.



The available widgets for the **rectangular area** include slider, dial, ultrasonic radar and grayscale detection tool.



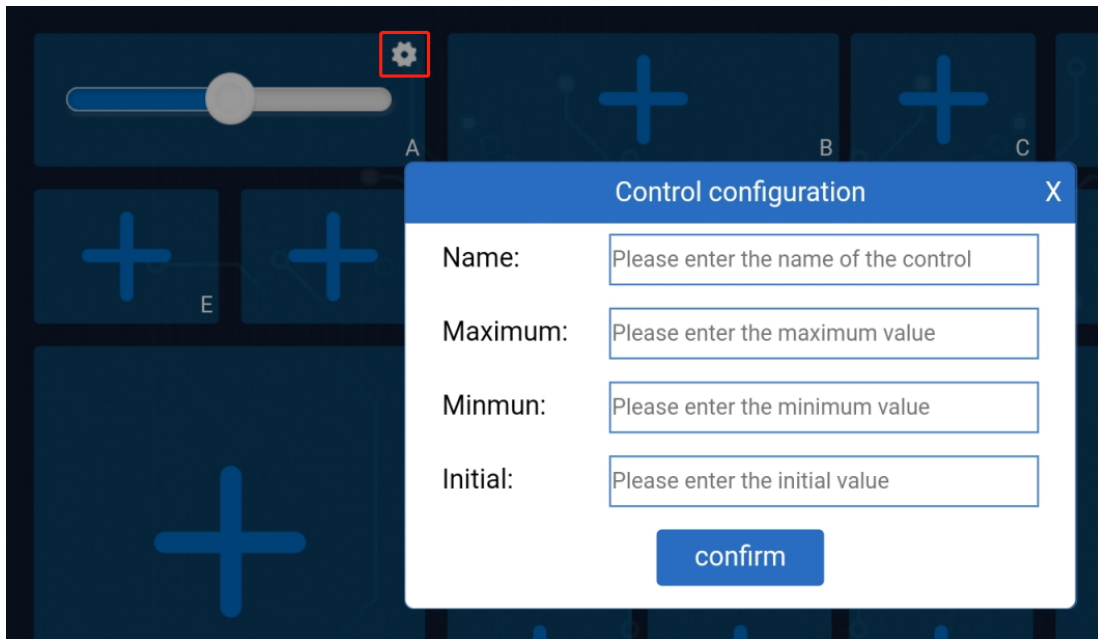
Widgets List

Here, you will learn the parameter types and ranges of control widgets and data widgets.











Control Widgets

The control widgets of Sunfounder Controller include buttons, switches, joystick, D-Pad, and slider.

You can modify the name, parameter range and initial value of some widgets by clicking the settings button in the upper right corner.



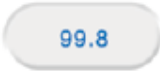


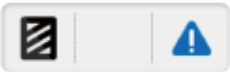



When we use these control widgets, proofreading information of the ESP-4WD car will receive the control data. Through these control data, we can write code to control the car.

Widget	Trigger Event	Data (Type)
	Hold	True(boolean)
	Release	False(boolean)
	Open (ON)	True(boolean)
	Close (OFF)	False(boolean)
	Move the middle cursor	axis array[x,y] x_axis range: -100~100 (int) y_axis range: -100~100 (int)
	Hold 	"forward" (string)
	Hold 	"backward" (string)
	Hold 	"left" (string)
	Hold 	"right" (string)
	Sliding slider	Range: 0~100 (int) Range can be modified through 

Data Widgets

The data widget of Sunfounder Controller includes digital displays, dial, ultrasonic radar, and grayscale detection tool.

When we send sensor data to these data widgets, we can show the data on the corresponding widgets. At the same time, you can also modify the name, unit and parameter range of the data widget by clicking the setting icon in the upper right corner.

Widget	Data (Type)	Display
	-1000.00~1000.00(float)	Can not be modified
	0~100(int)	Range can be modified through 
	Array with 3 elements (float)	Element Value<110:  Element Value<400:  Element Value>= 400:no pattern (Can be modified)
	Array with 2 elements (float)	First element: the angle value Second element: distance value

Establish Communication

There are two ways to establish communication between Sunfounder Controller and ESP-4WD car: One is AP mode, the other is STA mode.

- **AP Mode:** You need to connect Sunfounder Controller to the hotspot released by ESP-4WD car.
- **STA Mode:** You need to connect Sunfounder Controller and ESP-4WD car to the same LAN.

We can switch the communication mode by modifying the code `ws.py` and defining the `SWITCH_MODE` variable as `ap` or `sta`.

```
SWITCH_MODE = "ap"
```

AP Mode

If you want to use AP mode, you need to connect Sunfounder Controller to the hotspot released by ESP-4WD car.

1. Open the code `ws.py`, modify the `NAME` and `AP_PASSWORD` to yours.

```
NAME = 'ESP-4WD Car'
AP_PASSWORD = "123456789"
```

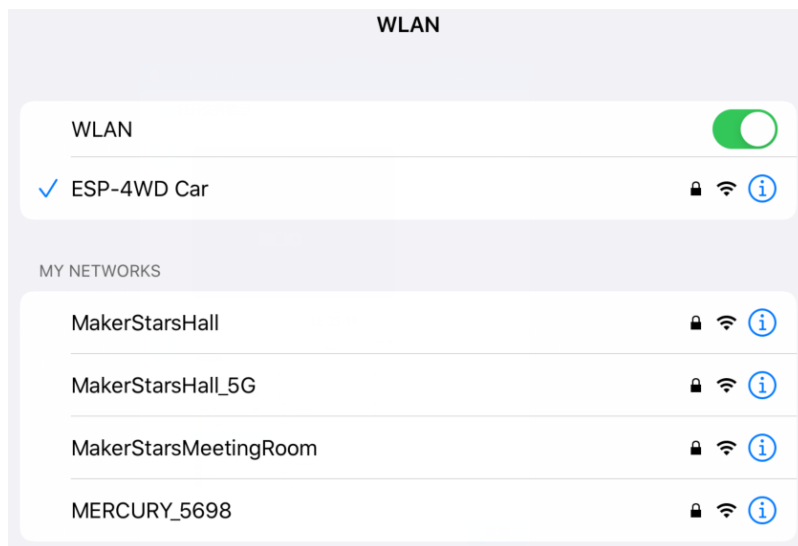
Note: The NAME in the code is both the SSID and the name of the car, if you have more than one EPS-4WD car, you need to set different NAMES for them to avoid a wrong connection.

In addition, you need to set a password of more than 8 digits.

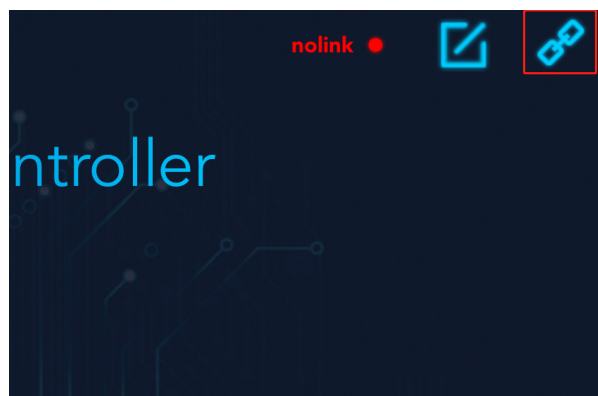
2. Then define the SWITCH_MODE variable as ap.

```
SWITCH_MODE = "ap"
```

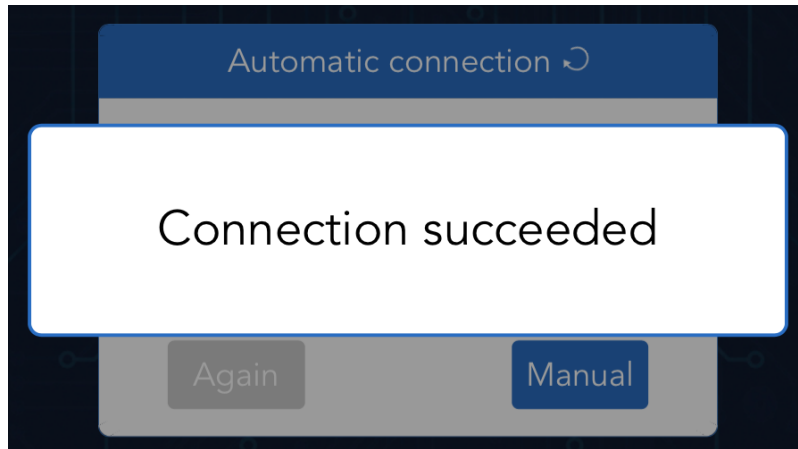
3. After downloading the code, ESP-4WD car will send a hotspot signal, then take out your mobile device, open the WLAN management interface and connect to the wifi network.



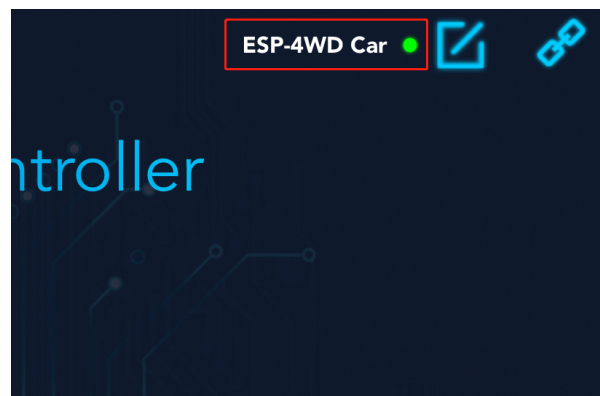
4. Open **Sunfounder Controller** and click the **Connect** icon on the top right corner.



5. A prompt box will appear if the connection is successful.



6. And the name of the car will be shown on APP.



STA Mode

If you want to use STA mode, you need to connect Sunfounder Controller and ESP-4WD car to the same LAN.

1. Open the code `ws.py`, modify the `STA_NAME` and `STA_PASSWORD` to yours.

```
STA_NAME = "MakerStarsHall"
STA_PASSWORD = "sunfounder"
```

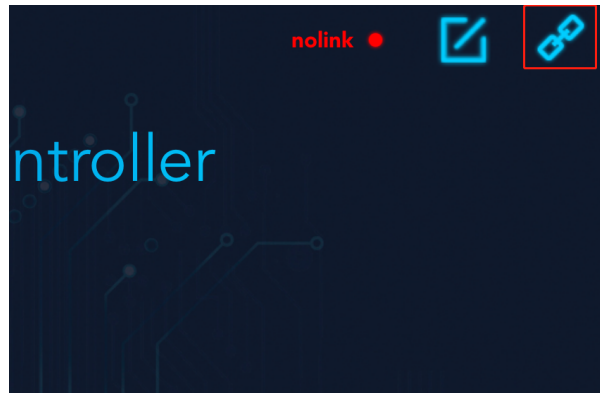
2. Then define the `SWITCH_MODE` variable as `sta`.

```
SWITCH_MODE "sta"
```

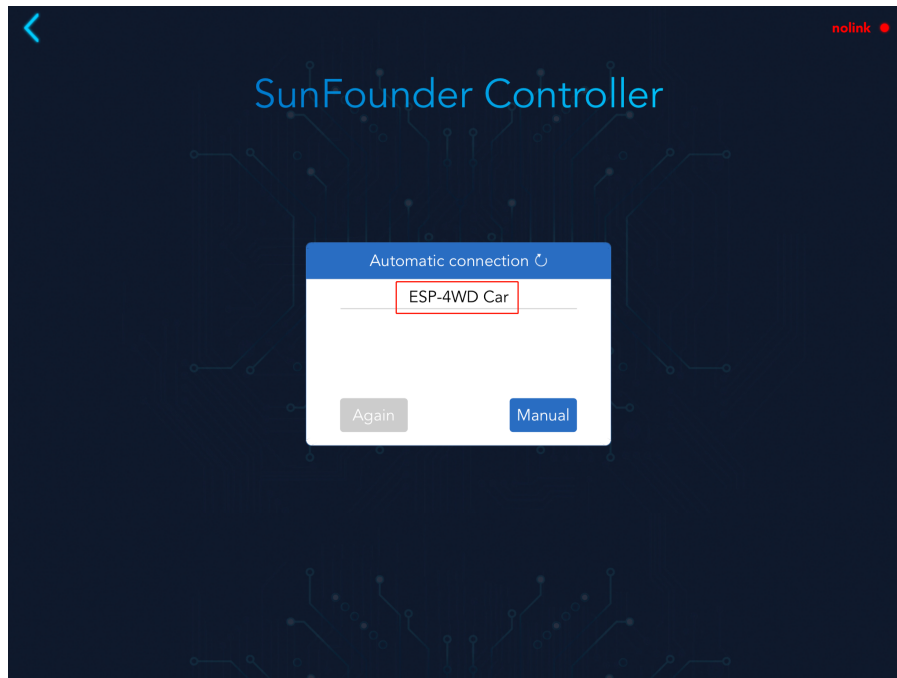
3. After downloading the code, ESP-4WD car will automatically connect to the wifi network, and at the same time take out your mobile device, open the WLAN management interface and connect to this wifi network.



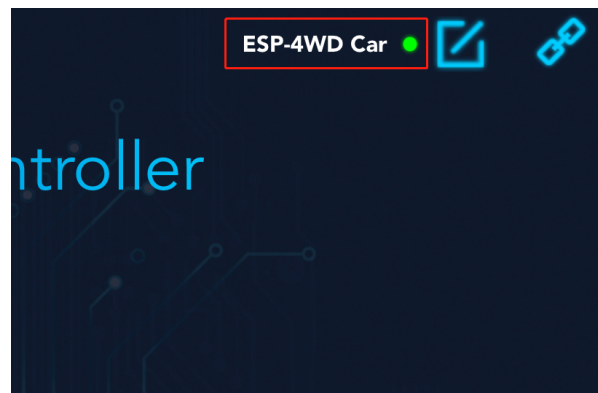
4. Open Sunfounder Controller and click the **Connect** icon on the top right corner.



5. Find the car name in the pop-up window and click on it.



6. After connecting, the name of the car will be shown on APP.



Control the Car with APP

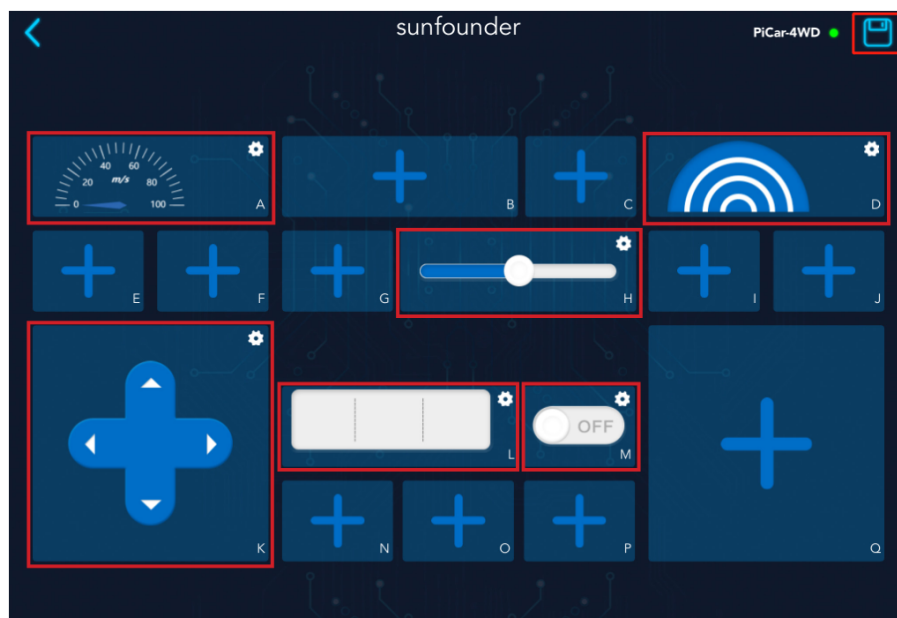
Either way, you can get the SunFounder Controller and ESP-4WD car to establish communication, next you will learn how to control the car with APP.

1. Open Sunfounder Controller, click the + to create an empty controller.

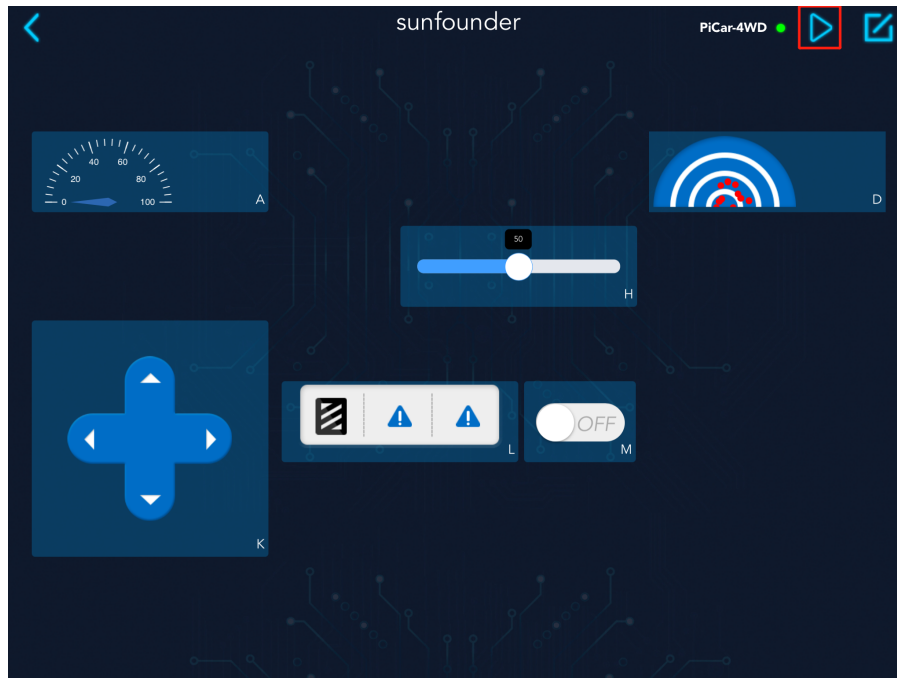


2. As shown in the figure, select the corresponding widget and click the icon in the upper right corner to save.

- **Widget A:** Show the driving speed of the car.
- **Widget D:** Simulate radar scanning.
- **Widget H:** Control the driving speed of the car.
- **Widget K:** Control the driving direction of the car.
- **Widget L:** Show the detection result of the grayscale sensor.
- **Widget M:** Control the on and off of the RGB board.



3. Click the start button in the upper right corner, and then try to use these widgets to control ESP-4WD car.



DIY Controller

If you want to DIY a new controller, you need to understand the communication process between the ESP-4WD car and the Sunfounder Controller. Open the `test_control.py` file. You will go through this code to see how they communicate with each other.

Program framework

First, let us understand the general operating framework of the program.

Turn the code to line 34. In the `main()` function, we have written the basic implementation code for build a controller.

- `ws.start()`: Establish communication between ESP-4WD car and Sunfounder Controller.
- `result = read()`: Read the received data and store it in the result variable.
- `write()` Send sensor data to Sunfounder Controller.

```
def main():
    ws.start()
    print("start")
    while True:
        result = read()
        if result != None:
            # coding the control function here.

            # coding the sensor function here.

            # ws.send_dict['L_region'] = car.get_grayscale_list() # example for test_
            ↪ sensor data sending.
        write()
        time.sleep_ms(15)
```

Open the `ws.py` file, turn the code to line 87, in the `start()` function, we switch the communication mode by judging the value of `SWITCH_MODE`.

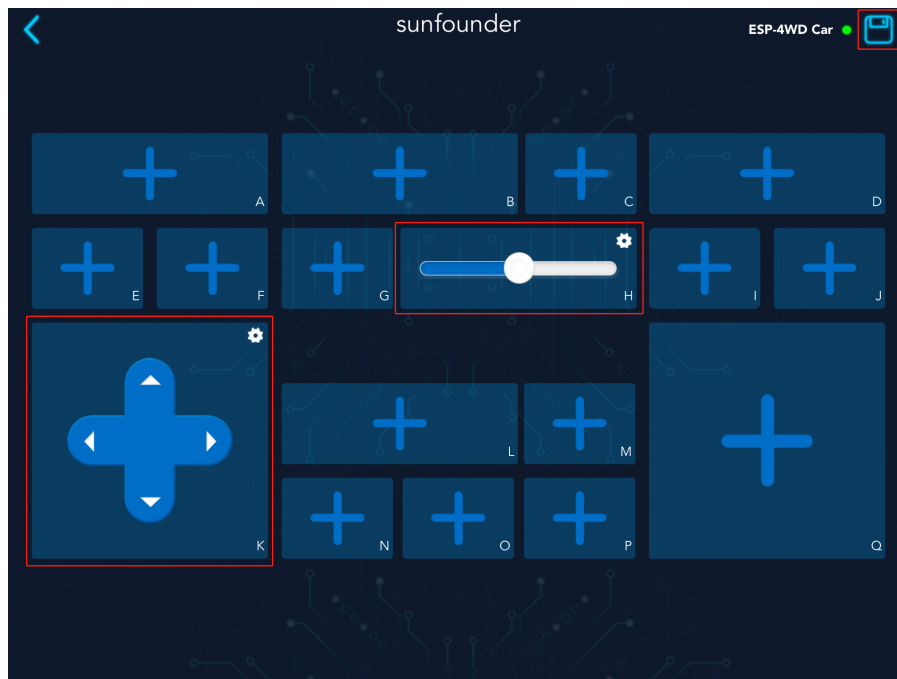
```
def start(self):
    if SWITCH_MODE == "ap":
        self.wlan = network.WLAN(network.AP_IF)
        self.wlan.config(essid=AP_NAME, authmode=4, password=AP_PASSWORD)
        self.wlan.active(True) # turning on the hotspot
    elif SWITCH_MODE == "sta":
        self.wlan = network.WLAN(network.STA_IF)
        self.wlan.active(True)
        self.wlan.connect(STA_NAME, STA_PASSWORD)
```

Receiving

The ESP-4WD car receives data from the Sunfounder Controller and sends its own sensor data to the Sunfounder Controller. Let's find out what data ESP-4WD car receives from Sunfounder Controller.

Step 1: Create new controller

Run the `test_control.py` file, re-establish communication, and then open Sunfounder Controller to create a new controller. We add a slider in the H area and a D-Pad in the K area. After adding, click the icon in the upper right corner to save.



Step 2: Receive data from SunFounder Controller

Turn the code to line 14, in the `read()` function, we have realized the receiving and printing of the data sent by the Sunfounder Controller. The function of the variable `temp` is to prevent repeated printing of data.

```
def read():
    global temp
    recv = ws.read()
    if recv == None:
```

(continues on next page)

(continued from previous page)

```

    return
recv_data = json.loads(recv)
if temp != recv_data:
    print("recv_data: %s\n"%recv_data)
    temp = recv_data
return recv_data

```

- Receive the Json object sent by Sunfounder Controller through the `ws.read()` function and store it in the `recv` variable.

```
recv = ws.read()
```

- The variable `recv` (Json object format) is parsed into `recv_data` dictionary through the `json.loads()` function.

```
recv_data = json.loads(recv)
```

- Print the value of variable `recv_data`.

```
print("recv_data: %s\n"%recv_data)
```

Click the start icon in the upper right corner to run the controller.



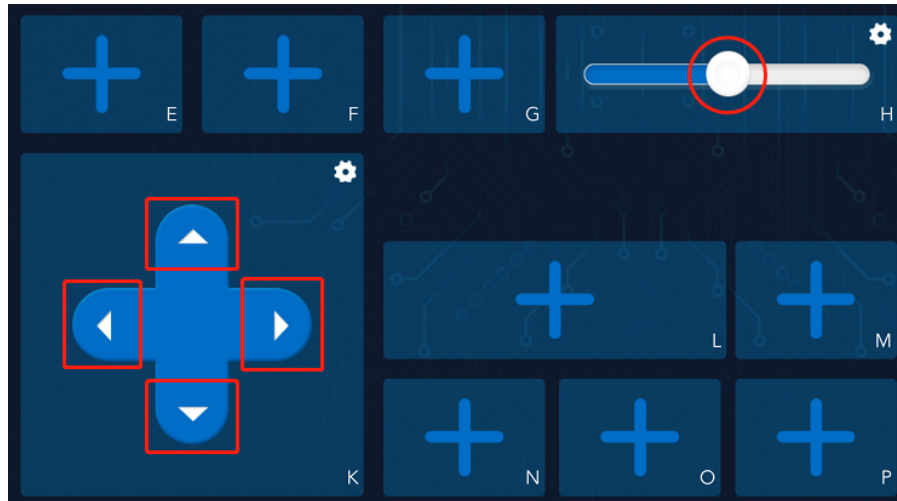
Open the Shell window under Thonny, we can find that the initial data of K control is the string `stop`, and the initial data of H widget is the int value 50.

```

Received text: {"A_region":null,"B_region":null,"C_region":null,"D_region":null,"E_
region":null,"F_region":null,"G_region":null,"H_region":50,"I_region":null,"J_
region":null,"K_region":"stop","L_region":[110,400],"M_region":false,"N_
region":null,"O_region":null,"P_region":null,"Q_region":null}

```

Press the arrow keys of the **D-Pad** in the K area and slide the **slider** in the H area.



You can see that the **D-Pad** widget sends a string of data (“forward”, “backward”, “left”, “right”) to the ESP-4WD car, while the slider widget will send an int data (range: 0-100).

```
Received text: {"A_region":null,"B_region":null,"C_region":null,"D_region":null,"E_region":null,"F_region":null,"G_region":null,"H_region":75,"I_region":null,"J_region":null,"K_region":"left","L_region":[110,400],"M_region":false,"N_region":null,"O_region":null,"P_region":null,"Q_region":null}
```

Step 3: Responding

When ESP-4WD car receives data from Sunfounder Controller, it needs to respond accordingly.

Let's write a piece of code that uses the widgets on the Sunfounder Controller to control the movement of the car. The K widget(D-Pad) controls the direction of the car, and the H widget(slider) controls the speed of the car.

Add the following highlighted code to line 41 (a blank line).

```
def main():
    ws.start()
    print("start")
    while True:
        result = read()
        if result != None:
            # coding the control function here.
            car.move(result['K_region'], result['H_region'])
            # coding the sensor function here.

            # ws.send_dict['L_region'] = car.get_grayscale_list() # example for test_
            ↪sensor date sending.
            write()
            time.sleep_ms(15)
```

- Through the `read()` function, you can receive the data sent by Sunfounder Controller and store it in the result dictionary.

```
result = read()
```

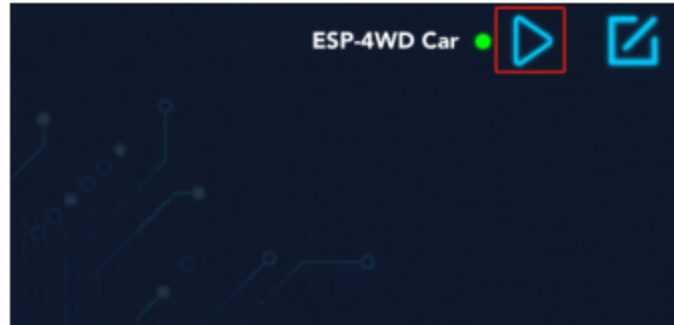
- Pass `result['K_region']` as the first parameter to the `car.move()` function to control the direction of the ESP-4WD car. Pass `result['H_region']` as the second parameter to the `car.move()` function to control the speed.

- The value of `result['K_region']` is the string data ("forward", "backward", "left", "right") sent by the K widget (D-Pad), the same as the value of `result['H_region']` is the int data sent by H widget (slide) (range: 0-100).

```
car.move(result['K_region'], result['H_region'])
```

After save and run the modified code and re-establishing communication, open the controller and click the start icon in the upper right corner to run it.

The D-Pad in the K area can control the direction of the ESP-4WD car, and the slider in the H area can control the speed.



Sending

Let's take a closer look at how the ESP-4WD car sends its own sensor data to the Sunfounder Controller.

Step 1: Create new controller

Open the `test_control.py` file and go to line 44, delete the comment symbol for this code.

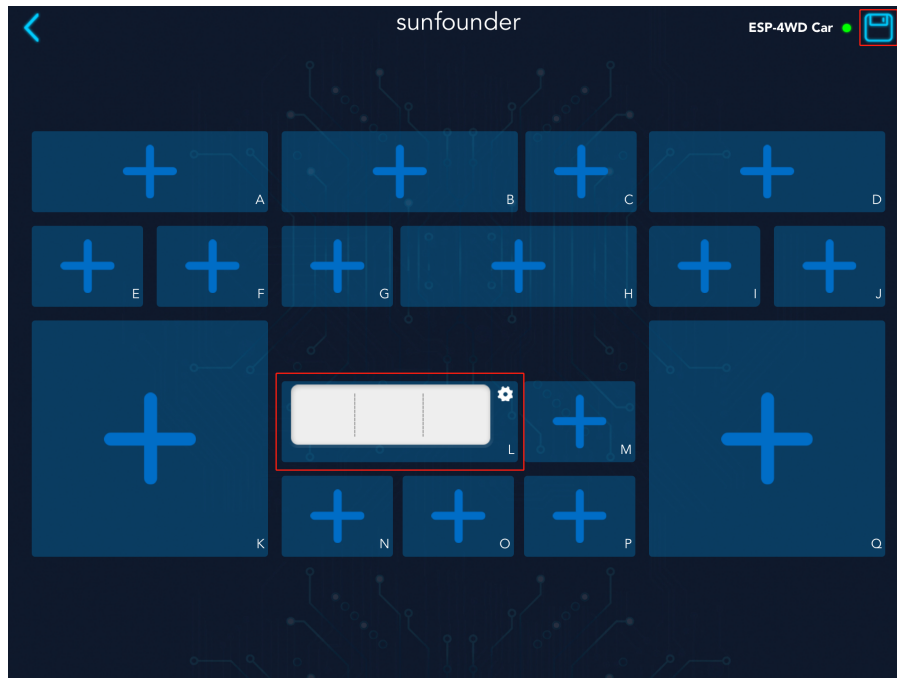
This code is used to get the grey scale sensor data from the `car.get_grayscale_list()` function and store it in the `ws.send_dict` dictionary and define the key as `L_region`.

```
def main():
    ws.start()
    print("start")
    while True:
        result = read()
        if result != None:
            # coding the control function here.

            # coding the sensor function here.

            ws.send_dict['L_region'] = car.get_grayscale_list() # example for test_
            ↪sensor data sending.
            write()
            time.sleep_ms(15)
```

Save and run this code, re-establish communication, and then open Sunfounder Controller to create a new controller. We add a grayscale detection tool in the L area. After adding, click the icon in the upper right corner to save.



Step 2: Receive data from SunFounder Controller

Open the `ws.py` file and go to line 25. Here, the equipment information and proofreading information of ESP-4WD Car are stored in the `send_dict` dictionary.

```
send_dict = {
    'Name': AP_NAME,
    'Type': 'ESP-4WD Car',
    'Check': 'SunFounder Controller',
}
```

Open the `test_control.py` and go to line 26, through the `write()` function, we send sensor data to the Sunfounder Controller, where the variable `temp_send` is used to prevent repeated printing of data.

```
def write():
    global temp_send
    ws.write(json.dumps(ws.send_dict))
    if temp_send != ws.send_dict:
        print("send_data:%s\n"%ws.send_dict)
        temp_send = ws.send_dict.copy
    return
```

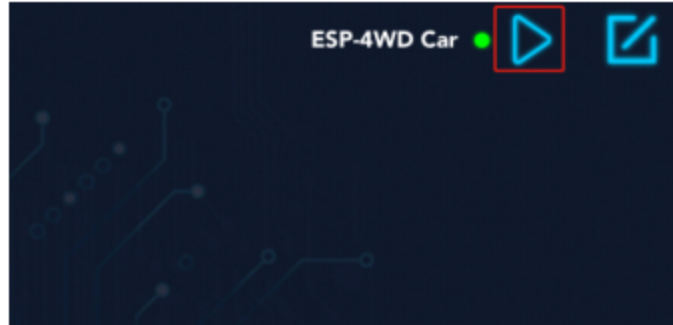
- First, use the `json.dumps()` function to convert the `ws.send_dict` dictionary into a Json object, and then use the `ws.write()` function to send the Json object storing the sensor data to the Sunfounder Controller.

```
ws.write(json.dumps(ws.send_dict))
```

- Then print the value of the `ws.send_dict` dictionary.

```
print("send_data:%s\n"%ws.send_dict)
```

Click the start icon in the upper right corner to run the controller.



Open the Shell window under Thonny, you will see that the ESP-4WD car has been sending device information, calibration information and the value of the grayscale sensor to the Sunfounder Controller.

```
Send text: {"Name":"ESP-4WD Car","Type":"ESP-4WD Car","Check":"SunFounder Controller","L_region":[304,336,243]}
```

Step 3: Responding

Let's write a piece of code that show the sensor data of Sunfounder Controller. Widget L (grayscale detection tool) will show you the grayscale of the ground.

Turn the code to line 34, Let's re-explain the content of the `main()` event.

```
def main():
    ws.start()
    print("start")
    while True:
        result = read()
        if result != None:
            ws.send_dict['L_region'] = car.get_grayscale_list()
            write()
            time.sleep_ms(15)
```

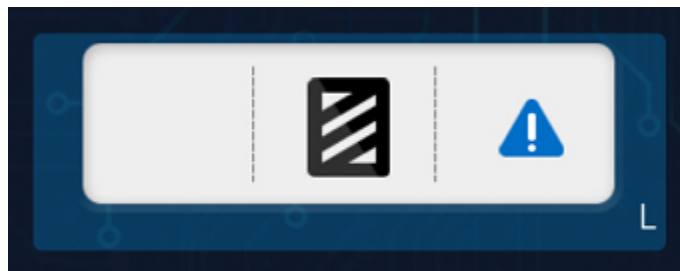
- By `car.get_grayscale_list()` function, we can get the list of grayscale sensor detection values and assign them to `ws.send_dict['L_region']`.

```
ws.send_dict['L_region'] = car.get_grayscale_list()
```

- Send sensor data, device information and proofreading information of the ESP-4WD car to Sunfounder Controller through `write()` function.

```
write()
```

Now, open the SunFounder Controller again, Widget D (grayscale detection tool) is showing the current ground conditions.



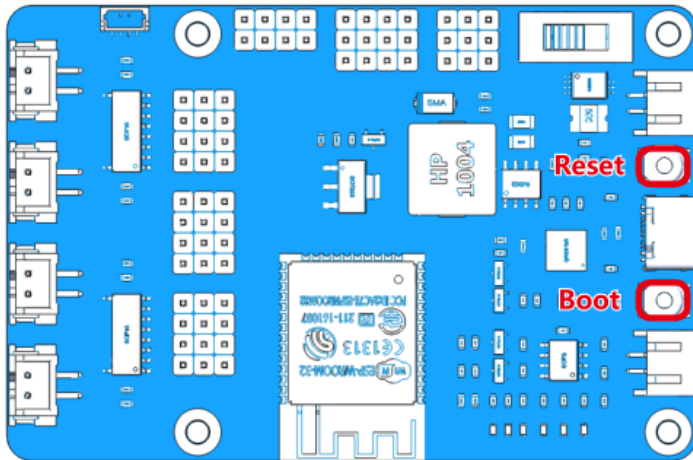
1.5 FAQ

In this chapter, we will answer the questions you may encounter when using ESP-4WD car.

Q1: When downloading arduino code, the serial port will not be connected for a long time.

```
esptool.py v2.6
Serial port COM12
Connecting....._....._
```

A: This is because the transistor on the ESP32 RDP fails to reset the development board automatically, you need to reset the development board manually.



Manual reset method: When the serial port is connected, press the Reset button and the Boot button at the same time, release the Reset button first, and then release the Boot button. If the following message appears, it means the download is successful.

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

Q2: When erasing the flash of ESP32 RDP, the serial port will not be connected for a long time.

```
C:\Users\123>esptool.py --port COM16 erase_flash
esptool.py v3.0
Serial port COM16
Connecting....._....._
```

A: This is because the transistor on the ESP32 RDP fails to reset the development board automatically, you need to reset the development board manually. Please refer to Q1 for the reset method. If the following message appears, it means the erasure is complete.

```
Stub running...
Erasing flash (this may take a while)...
Chip erase completed successfully in 8.0s
Hard resetting via RTS pin...
```

Q3: When downloading programs related to controlling the movement of the car, the car moves abnormally.

A: This is because the power consumption of the ESP-4WD car motor is relatively large, and you need to replace a battery with sufficient power.

1.6 Thank You

Thanks to the evaluators who evaluated our products, the veterans who provided suggestions for the tutorial, and the users who have been following and supporting us. Your valuable suggestions to us are our motivation to provide better products!

Particular Thanks

- Len Davisson
- Kalen Daniel
- Juan Delacosta

Now, could you spare a little time to fill out this questionnaire?

Note: After submitting the questionnaire, please go back to the top to view the results.

COPYRIGHT NOTICE

All contents including but not limited to texts, images, and code in this manual are owned by the SunFounder Company. You should only use it for personal study, investigation, enjoyment, or other non-commercial or nonprofit purposes, under the related regulations and copyrights laws, without infringing the legal rights of the author and relevant right holders. For any individual or organization that uses these for commercial profit without permission, the Company reserves the right to take legal action.