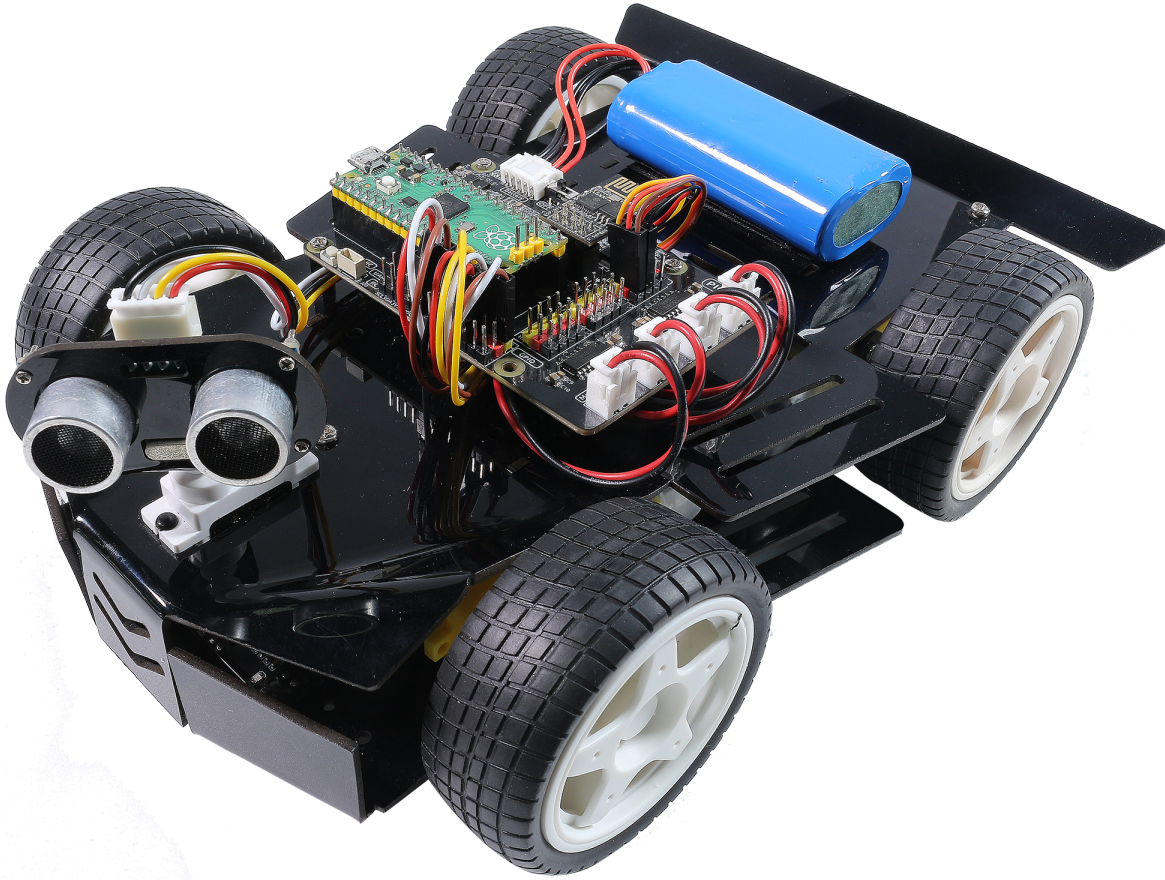

SunFounder pico_4wd_car

www.sunfounder.com

Jun 28, 2023

CONTENTS

1	1. Assemble the Car	5
2	2. Play Mode	7
2.1	Quick User Guide	7
2.2	RGB Boards Related	14
2.3	STTI	15
2.4	Grayscale Module Related	16
2.5	Ultrasonic Module Related	21
3	3. Programming Mode	23
3.1	1. Basic Usage on Thonny	23
3.2	2. Learn Modules	42
3.3	3. Funny Projects	100
3.4	4. Play Pico 4WD with APP	118
4	4. Appendix	157
4.1	Introduction to Raspberry Pi Pico	157
4.2	Pico RDP	159
4.3	Schematic and Structure Drawing	163
4.4	Thonny IDE Introduction	164
4.5	18650 Battery	165
5	5. FAQ	167
5.1	Q1: NO MicroPython(Raspberry Pi Pico) Interpreter Option on Thonny IDE?	167
5.2	Q2: Cannot open Pico code or save code to Pico via Thonny IDE?	168
5.3	Q3: Unable to connect to COM51: Cannot configure port?	168
5.4	Q4How can I use the STT mode on my Android device?	169
6	6. Thank You	175
7	Copyright Notice	177



Do you want to learn to program or have your own programming robot? The market is filled with a wide variety of robots, some of which use the micro:bit platform, others use Arduino or Raspberry Pi platforms. If you're not sure what to select and where to begin, then our Pico 4WD car can help you begin your journey of discovery.

This car uses the Raspberry Pi Pico, a powerful small microcontroller introduced by Raspberry Pi in recent years, which can be used to learn MicroPython, Arduino or graphical programming language.

A simple structure and full functionality make the Pico 4WD car a perfect choice for your needs. There are a variety of features on it, including remote control, line tracking, cliff detection, obstacle avoidance, object following, and more.

Additionally, it features 24-bit WS2812 RGB LEDs that can be used as directional indicators or as cool lighting effects.

For quick play, we have uploaded libraries and scripts at the factory, which you just need to assemble, then you can controlling it with your smartphone.

One point that must be mentioned is that the car is equipped with a rechargeable battery, which can be charged directly through the expansion board.

Features

- Main Board: Raspberry Pi Pico, Pico RDP(Robotics Development Platform)
- Programming Software: Thonny
- Programming Language: MicroPython
- Input: Ultrasonic module, Grayscale module, Speed module
- Output: Motor, WS2812 RGB Boards, Servo,
- Voltage: 6.6V~8.4V

- Battery life: 90min
- Battery charge time: 130min
- Functions: Remote Control, Obstacle Avoidance, Object Following, Line Track, Cliff Detection, Speech Control
- WiFi: ESP01s
- Remote Control: APP - SunFounder Controller
- Size: 9.45" x 6.69" x 3.94" (240mm x 170mm x 100mm(LxWxH))

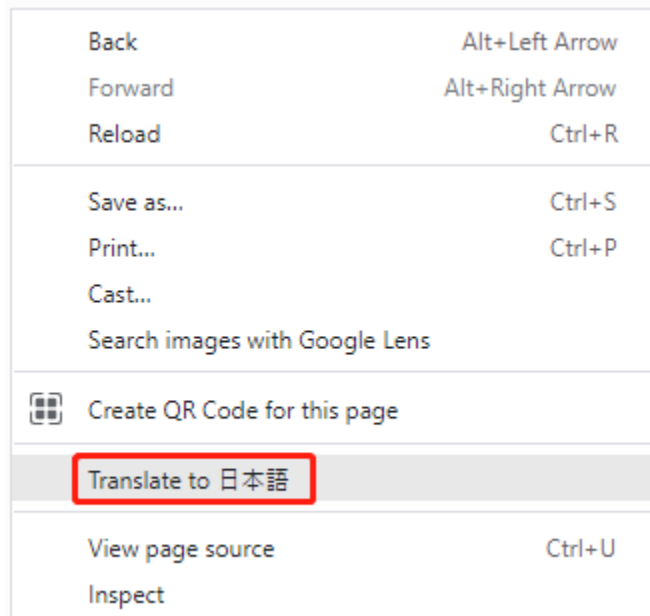
Here is the Email: cs@sunfounder.com.

About the display language

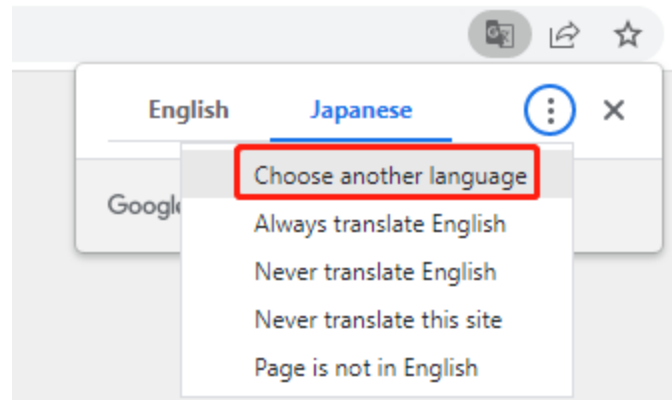
In addition to English, we are working on other languages for this course. Please contact service@sunfounder.com if you are interested in helping, and we will give you a free product in return. In the meantime, we recommend using Google Translate to convert English to the language you want to see.

The steps are as follows.

- In this course page, right-click and select **Translate to xx**. If the current language is not what you want, you can change it later.



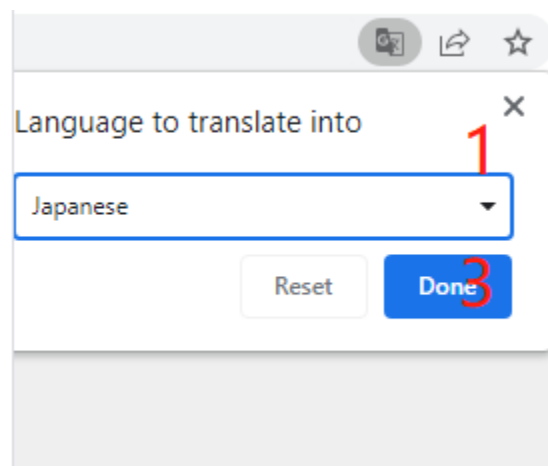
- There will be a language popup in the upper right corner. Click on the menu button to **choose another language**.



- Select the language from the inverted triangle box, and then click **Done**.

Arabic
Armenian
Azerbaijani
Bangla
Basque
Belarusian
Bosnian
Bulgarian
Burmese
Catalan

2



Source Code

SunFounder Pico-4wd Car Code

Or check out the code at

1. ASSEMBLE THE CAR

Follow the PDF tutorial below to assemble the car first, and then refer to [2. Play Mode](#) or [3. Programming Mode](#) to make it move.

- (PDF) Pico-4wd V2.0 Assemble Instruction

Assemble Tutorial Video

About Remove Rivets

If you feel that the rivets are installed incorrectly, you can remove them by the method shown below.

About the Rivet Size

2. PLAY MODE

This Pico-4wd is shipped with the code pre-uploaded so that you can control it directly with the APP after powering up.

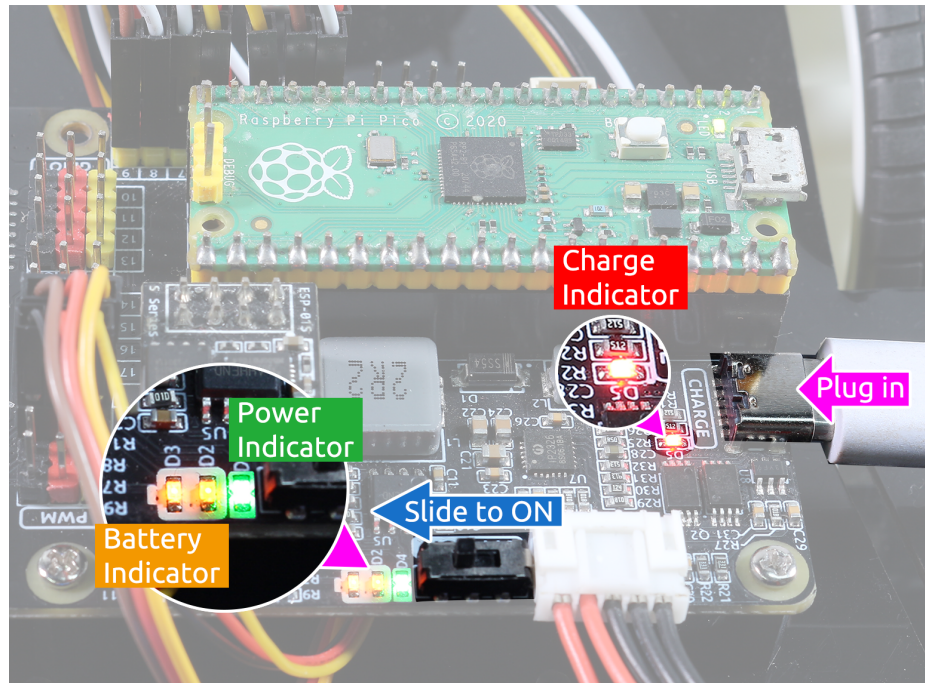
Here, the SunFounder self-developed App - SunFounder Controller is used. With this app, users can customize a controller to control their robot. It is compatible with Raspberry Pi, Raspberry Pi Pico/Pico W and Arduino boards.

Let's see how to control Pico 4WD car with this APP.

Note: If the `main.py` file in the Raspberry Pi Pico has been modified, but you still want to use Play Mode, then you just need to save the `app_control.py` file under the `pico_4wd_car/examples/app_control` path as `main.py` again.

2.1 Quick User Guide

1. Install [SunFounder Controller](#) from **APP Store(iOS)** or **Google Play(Android)**.
2. Let's start the Pico 4WD Car.
 - When first used or when the battery cable is unplugged, Pico RDP will activate its over-discharge protection circuitry(Unable to get power from battery).
 - Therefore, you'll need to plug in a Type-C cable for about 5 seconds to release the protection status.
 - At this time look at the battery indicators, if both battery indicators are off, please continue to plug in the Type-C cable to charge the battery.

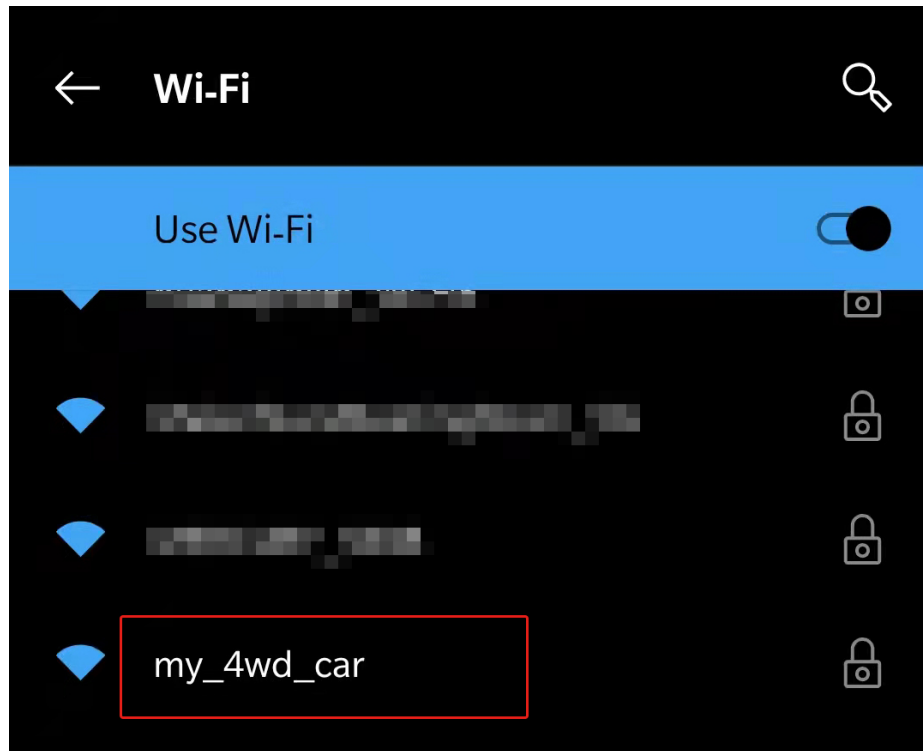


Note: Additionally, the LED on the ESP01S module will blink indicating that your mobile device is not connected.

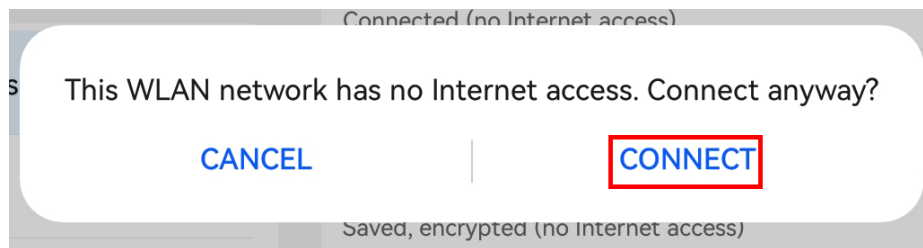
3. Connect to my_4wd_car LAN.

Your mobile device should now be connected to Pico 4WD Car's LAN so that they are on the same network.

- Find my_4wd_car on the WLAN of the mobile phone (tablet), enter the password 12345678 and connect to it.



- The default connection mode is AP mode. So after you connect, there will be a prompt telling you that there is no Internet access on this WLAN network, please choose to continue connecting.

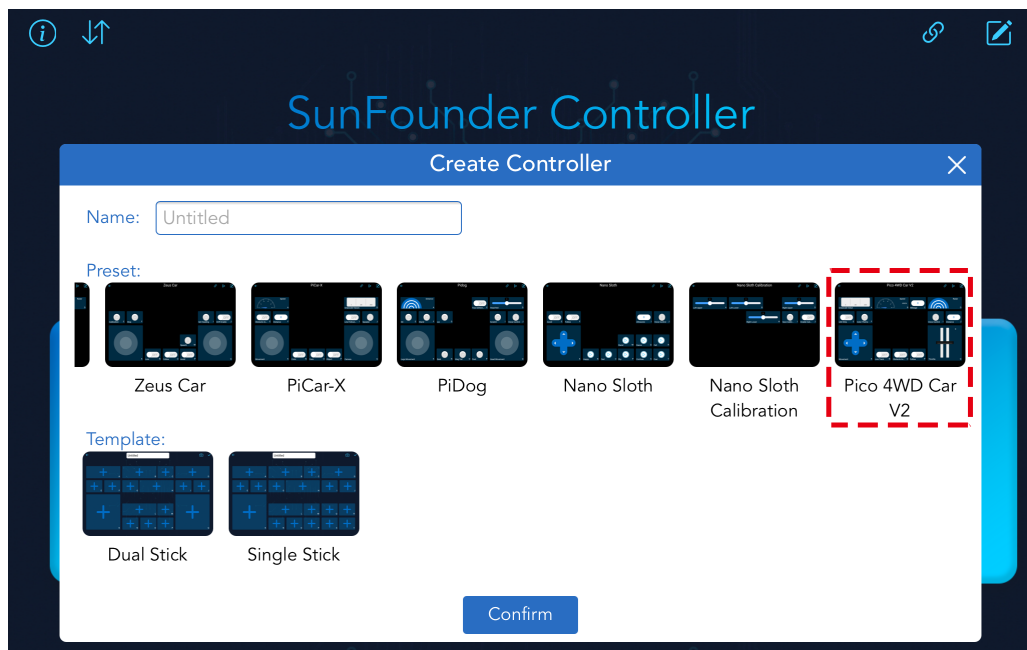


4. Create a controller.

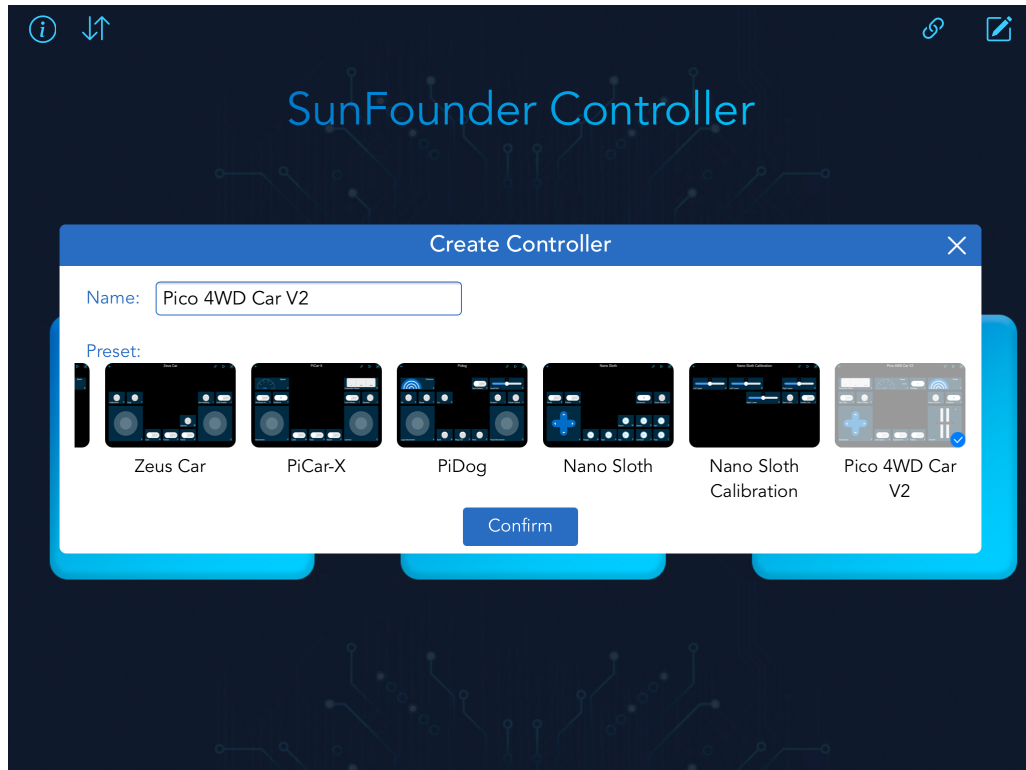
- Open SunFounder Controller and click on the + to create a new controller.




- We have preset controller for Pico-4wd, you can choose it directly.




- Define a name for this Controller and click **Confirm**.

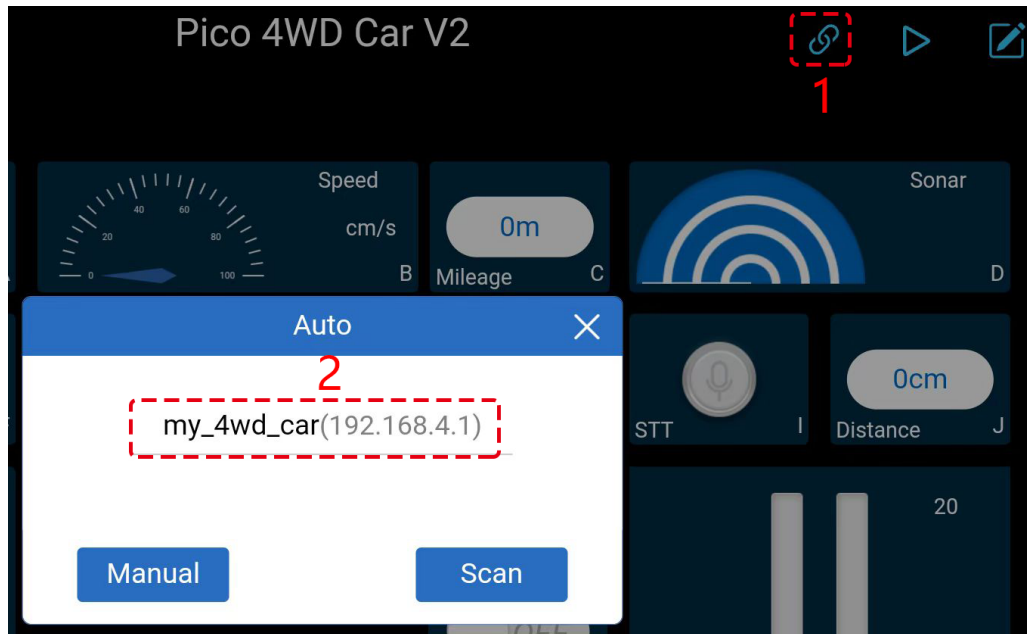


- You are now inside the controller, which already has several widgets set up. Click the  button in the upper right corner.



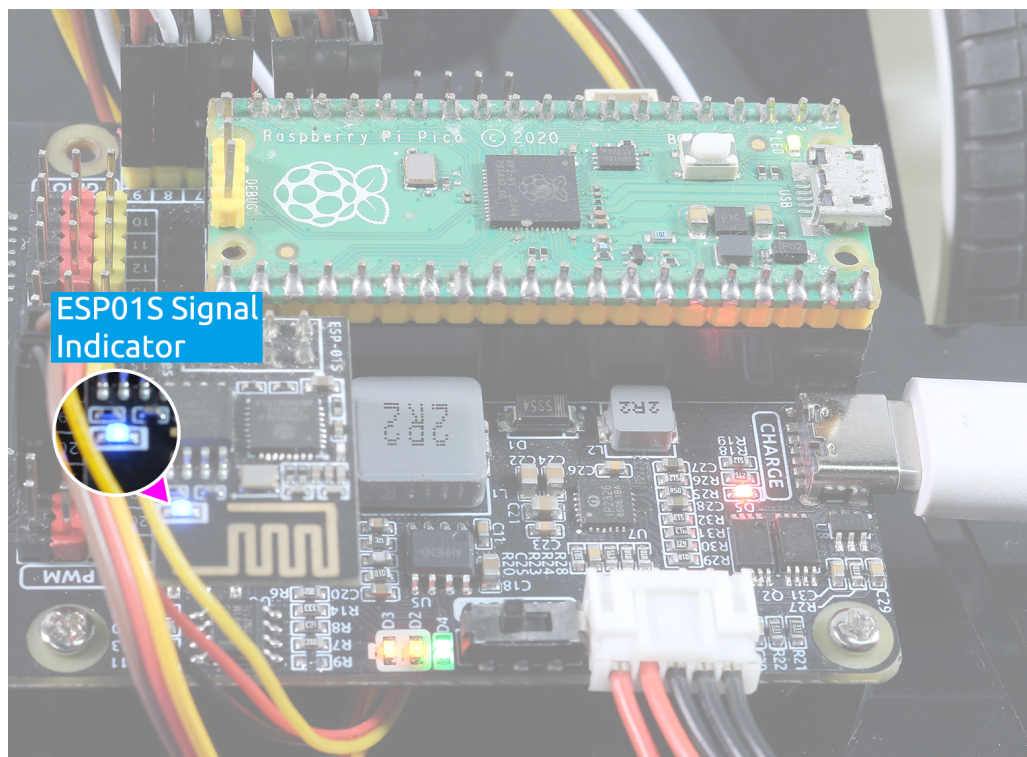
5. Connect and run the Controller.


- Now connect the SunFounder Controller to the Pico 4WD Car via the  button to start communication. Wait a few seconds and my_4wd_car (IP) will appear, click on it to connect.



Note: You need to make sure that your mobile device is connected to the my_4wd_car LAN, if you are not seeing the above message for a long time.

- After the “Connected Successfully” message appears and the product name will appear in the upper right corner.
- At the same time, the LED on the ESP01S module will stop flashing.



- Now, click the  button to control your Pico 4WD Car with these widgets.



6. Here are the functions of the widgets.

- **Moving Related**

- **Movement(K)**: Control the car to move back and forth, left and right.
- **Throttle(Q)**: Set the power of the car to move.
- **Cliff(M)**: Switch to cliff detection mode. As long as it's enabled, the car will stop automatically when it reaches the edge of a table or staircase when using the **Movement(K)** widget to move it. After you get it back to a safe area, you can control it to keep moving.

- **RGB Boards Related**

- **Underglow(E)**: Turn on/off the bottom RGB Boards.
- **Color(F)**: Switch to different colors.

- **STT**: Switching to STT(Speech to Text) mode.

- **Grayscale Module Related**

- **Grayscale Value(A)**: Shows the grayscale values detected by the Grayscale module and status indication in three different environments.
- **Line Track(N)**: Switching to line track mode.

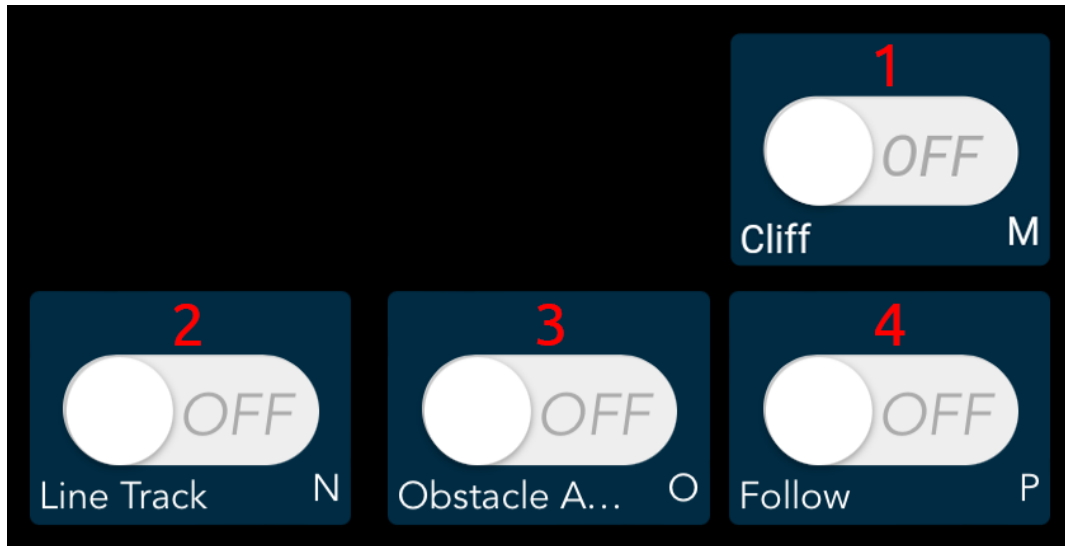
- **Ultrasonic Module Related**

- **Sonar(D)**: Shows obstacles detected by Ultrasonic module.
- **Distance(J)**: Shows the distance of obstacles.
- **Obstacle Avoidance(O)**: Switching to obstacle avoidance mode.
- **Follow(P)**: Switch to follow mode.

- **Speed(B)**: Shows the speed of the car.

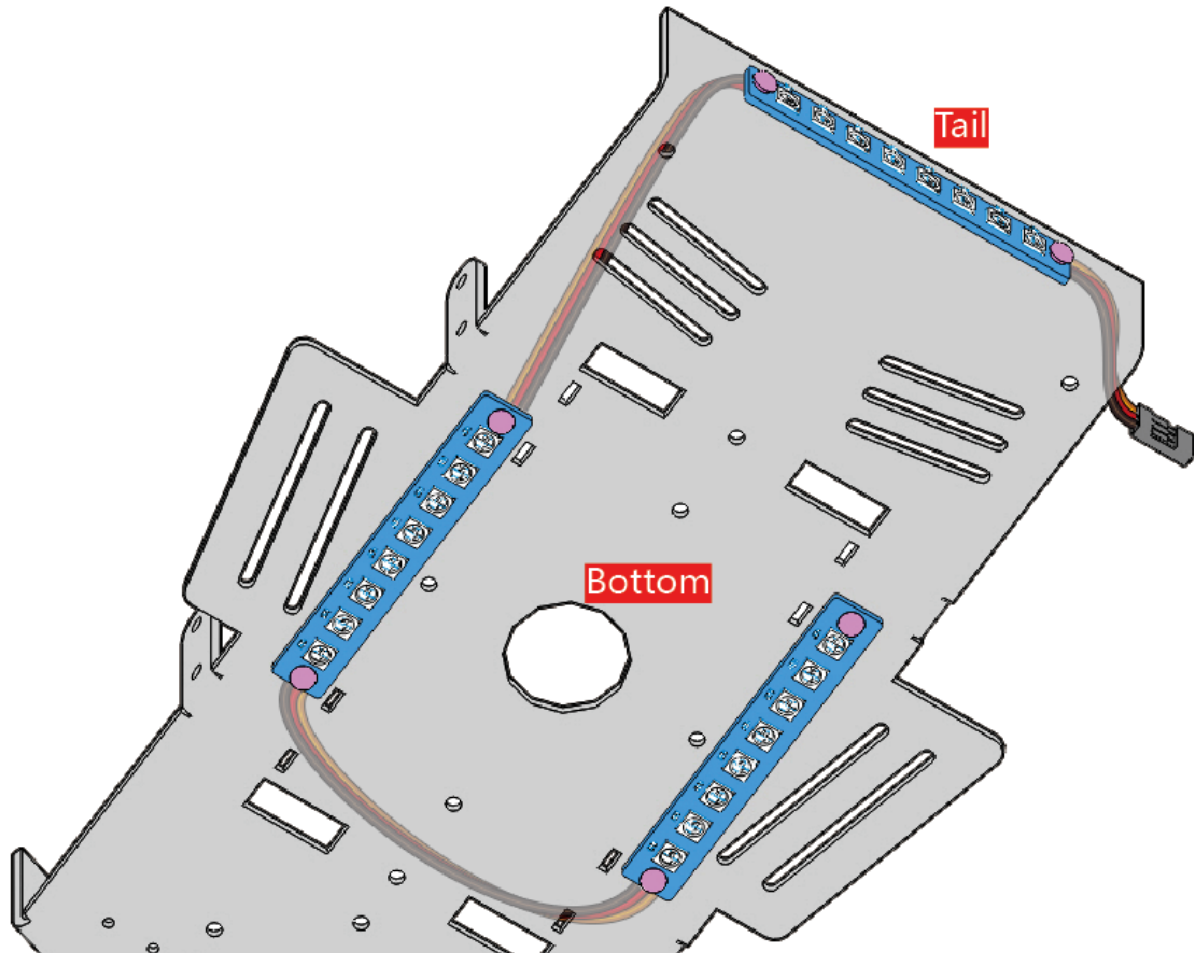
- **Mileage(C)**: Shows the mileage of the car in motion.

Note: As shown in the figure, the four modes run at different priority levels and cannot be enabled together.



2.2 RGB Boards Related

There are three 8-bit RGB Boards on the Pico 4WD Car, two at the bottom and one at the tail.



- **Underglow(E)** widget's function is to turn on or off bottom RGB boards.
- With the **Color(F)** widget, you can switch the color between 6 different colors: red, orange, yellow, green, blue, and purple.

By default, the RGB board at the tail lights up red when braking; while turning left or right, the two RGB LEDs on the left or right side light up orange.

2.3 STTI

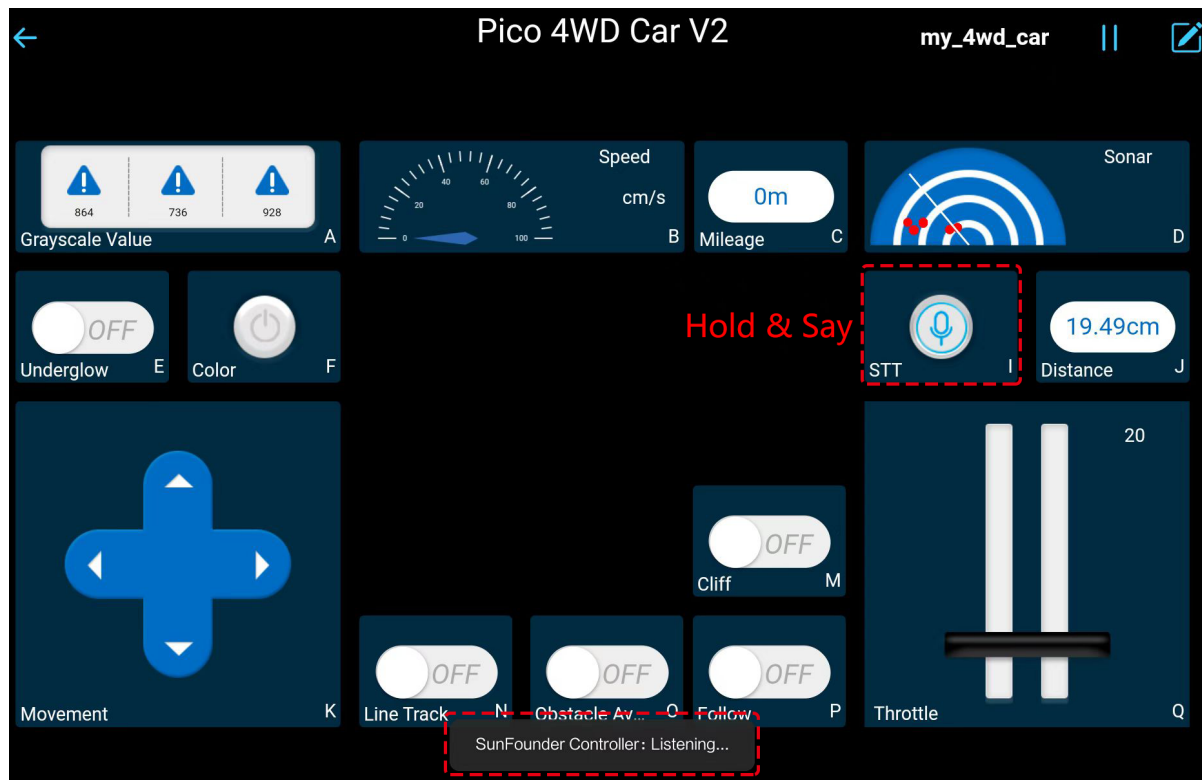
Warning: Android devices cannot use the STT(Speech to Text) mode this time(AP Mode). Because the STT mode requires the Android mobile device to be connected to the Internet and to install the Google service component.

While iOS devices use offline voice recognition engine, no network connection is required, AP and STA mode connection are both available.

If you want to use the STT mode on your Android device, please refer to [Q4How can I use the STT mode on my Android device?](#).

The Pico 4WD Car can also be controlled using STT in SunFounder Controller. Pico 4WD Car will perform the set actions based on the commands you say to your mobile device.

Now tap and hold the **STT(I)** widget and say any of the following commands to see what happens.



- stop: All movements of the car can be stopped.
- forward: Let the car move forward.
- backward: Let the car move backward.
- left: Let the car turn left.
- right: Let the car turn right.

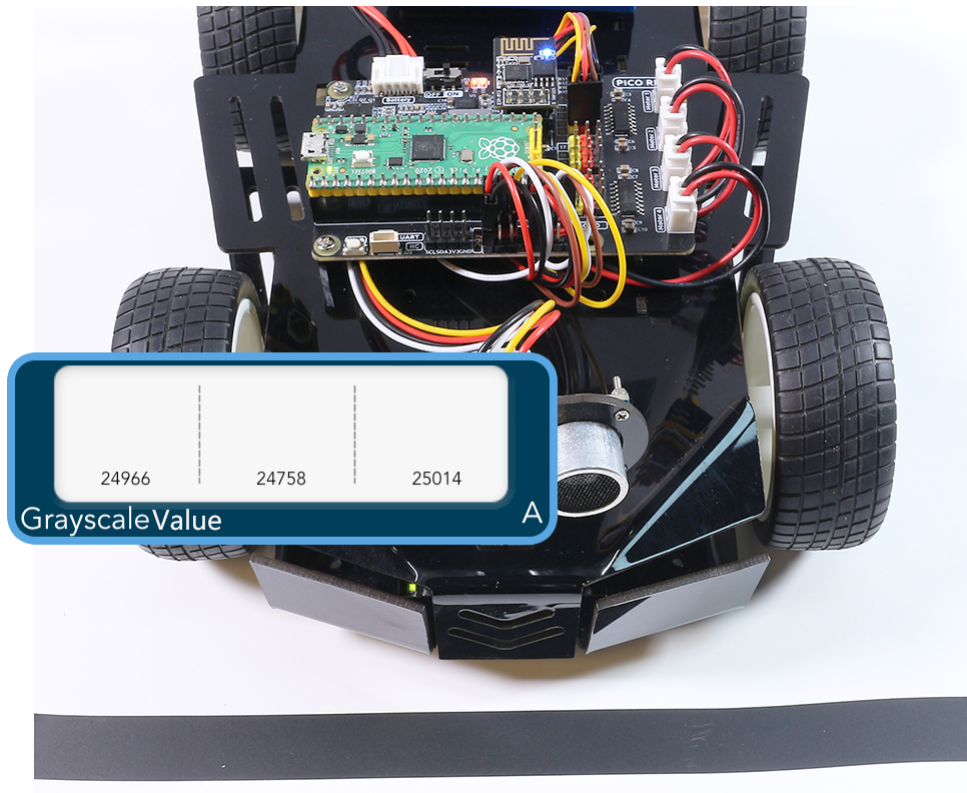
2.4 Grayscale Module Related

While this controller is running, **Grayscale Value(A)** will show the values of the three grayscale sensors in real time.

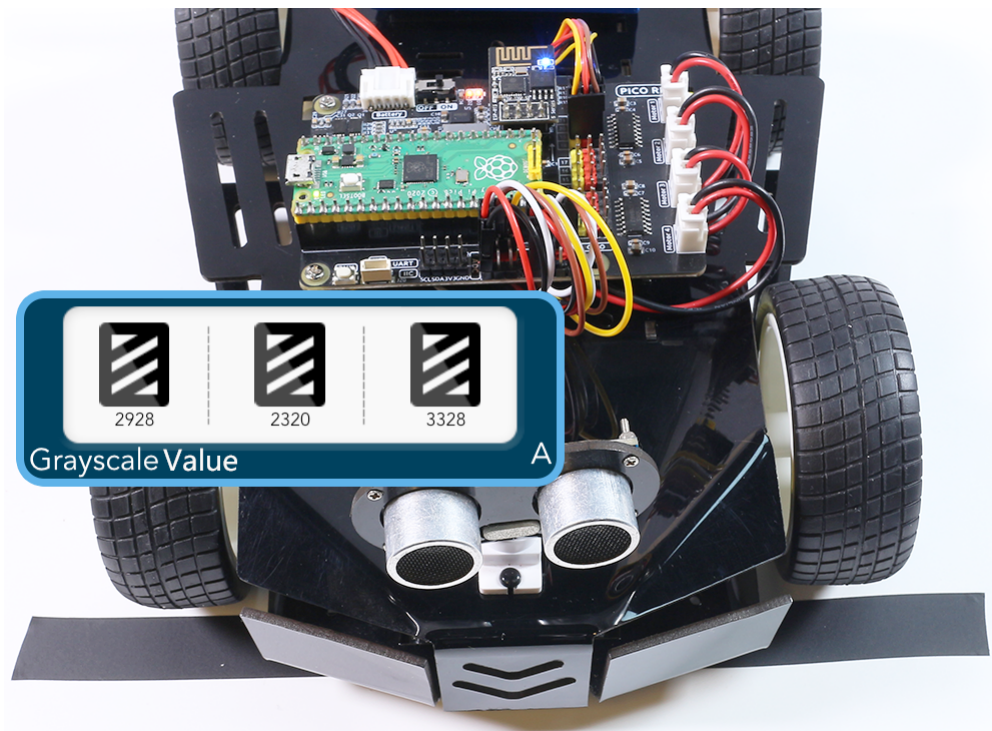
If you want to switch to Line Track mode (open the **Line Track(N)** widget), then you need to set the thresholds according to the current environment first, as follows.

1. Place the grayscale module in three environments: white, black and hanging in the air (10cm or more) to see how the data in the changes.

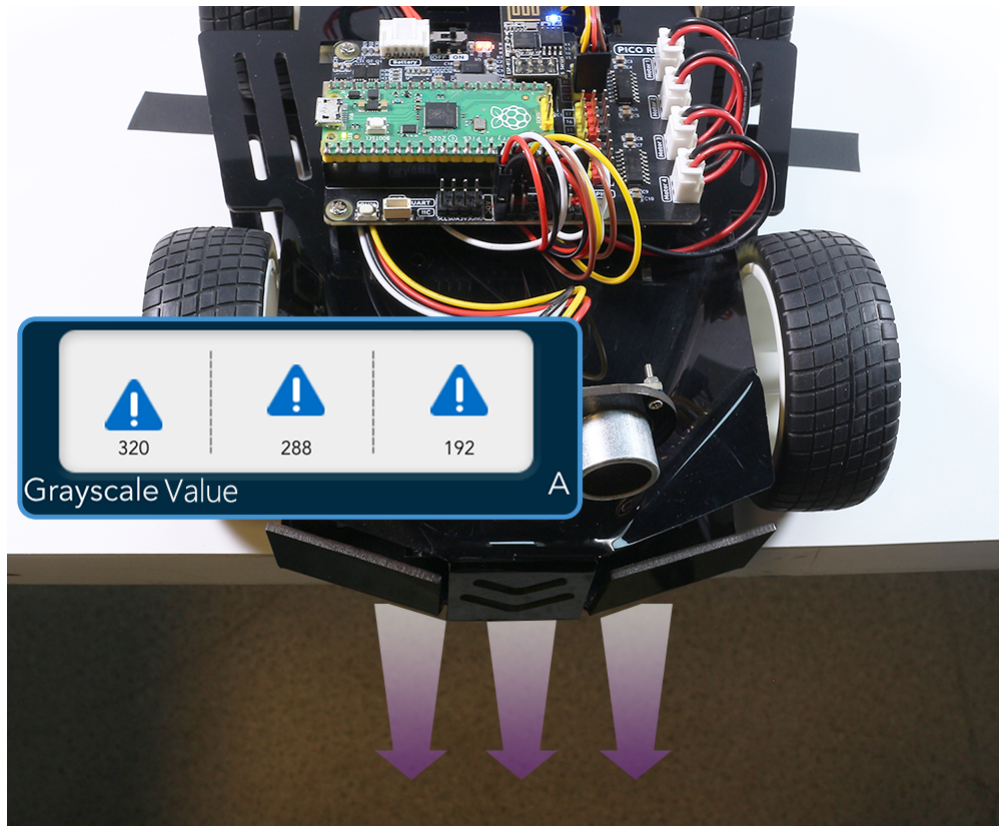
White surface You will find that the value of the white surface is generally large, for example mine is around 240,000.




Black line The value on the black line will be smaller, and now I'm at about 2000.

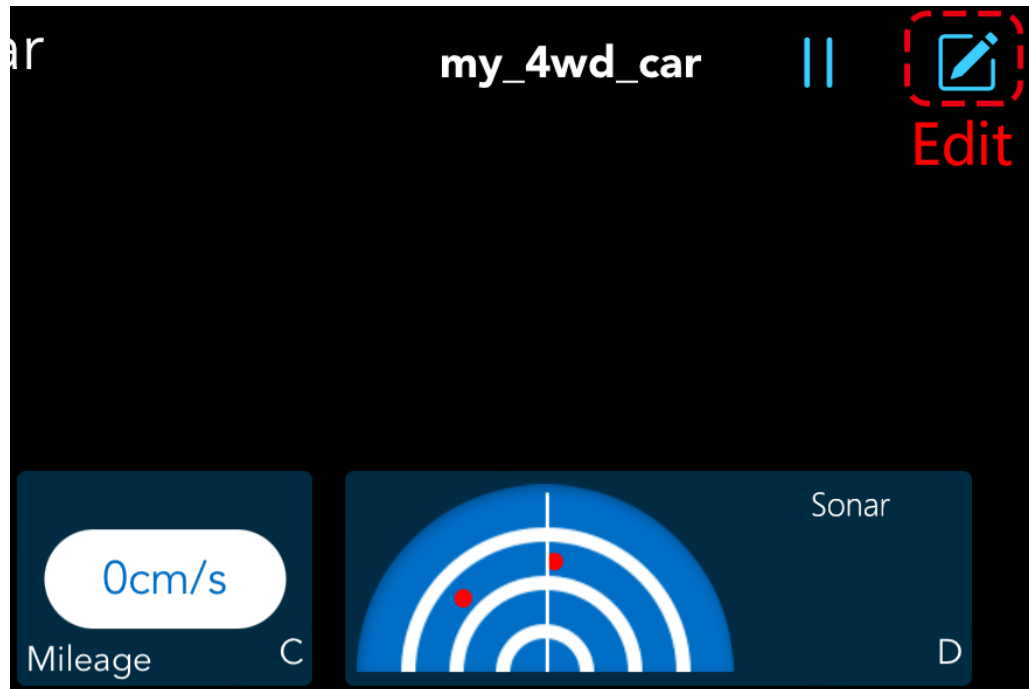


Overhang (10cm or more) And the value of the overhang will be even smaller, already less than 1000 in my environment.



2. Set the threshold value.

- My car reads around 24000 in the white area and around 2000 in the black line, so I set `line_ref` to about the middle value of 10000.
- In the cliff area it reads less than 1000, so I set `cliff_ref` to 1000.
- Now click the  button to enter edit mode.



- Click on the **Settings** button in the upper right corner of the **Grayscale Value(A)** widget.



- Fill in the cliff and line thresholds.

Widget Configuration

Name

Grayscale Value

Cliff_Ref

1000

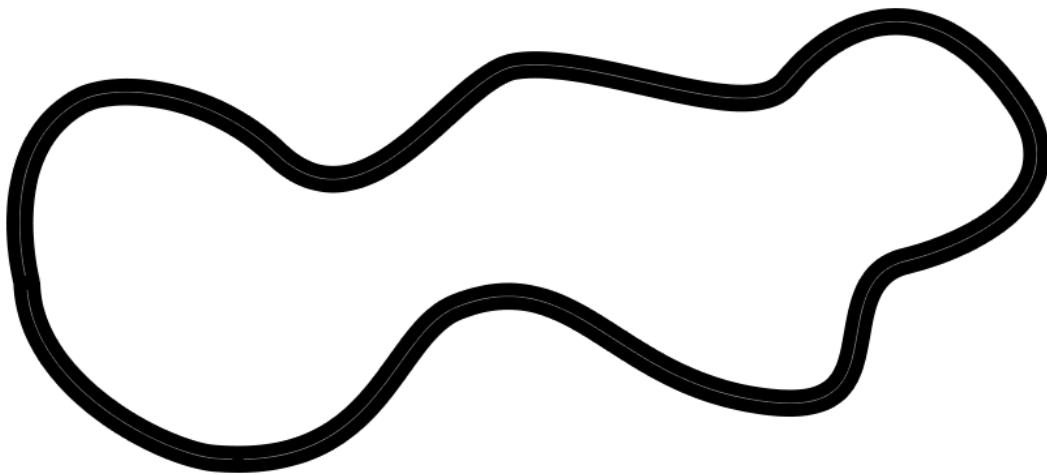
Line_Ref

10000

Confirm

3. Now that the car and the app are set up, we need to use the electrical tape to stick a line to track.

Note: The line you stick must be at least 1cm wide and the bend angle should not be less than 90°.



4. Place the Pico 4WD Car on your stickied line, open the **Line Track(N)** widget, and it will track the line.

2.5 Ultrasonic Module Related

Obstacle Avoidance

Turn on the **Obstacle Avoidance(O)** widget to switch to obstacle avoidance mode.

- The Pico 4WD car will keep moving forward and its ultrasonic sonar keeps turning.
- If an obstacle is detected in a certain direction, it will stop and detect it again from left to right.
- If it detects an obstacle on the left, it will turn to the right.
- If an obstacle is detected on the right, it will move to the left.
- It detects quickly, so you will find that it will detect as it goes until it is away from the obstacle, and then move forward.

Object Following

Open **Follow(P)** widget to switch to follow mode.

- When you put your hand or other objects in front of the car at a distance of about 20cm, the car will follow your hand or object to move forward, turn left and turn right.
- Be careful not to move your hand or object too fast, and keep the distance within 20cm.

3. PROGRAMMING MODE

The programming mode chapter is divided into 3 main parts in total, and you are advised to read them in order.

- *1. Basic Usage on Thonny*: This contains installation of the Thonny IDE, installation of MicroPython firmware and libraries for Pico, online and offline running of scripts.
- *2. Learn Modules*: This is a new section added in v2.0, allowing you to learn how each module works, then learn how to program them step by step, and finally encapsulate them individually into libraries.
- *3. Funny Projects*: This is the part that combines different modules to make Pico 4WD achieve different interesting functions.

Note:

Download the relevant code from the link below.

- [SunFounder Pico-4wd Car V2.0 Code](#)
-

3.1 1. Basic Usage on Thonny

In this section, you will learn how to download and install Thonny, install MicroPython firmware, upload libraries, run scripts and make scripts run offline.

If you are familiar with using Thonny, you can skip this chapter.

For *2. Install MicroPython on Your Pico(Optional)* and *3. Upload the Libraries to Pico (Optional)* with optional flags after them, it is because the Pico 4WD Car is shipped with MicroPython firmware and the corresponding libraries already installed. So these two steps can be left out in general, but of course you can do it if you want.

3.1.1 1. Install Thonny IDE(Important)

Before you can start to program Pico with MicroPython, you need an integrated development environment (IDE), here we recommend Thonny. Thonny comes with Python 3.7 built in, just one simple installer is needed and you're ready to learn programming.

Note: Since the Raspberry Pi Pico interpreter only works with Thonny version 3.3.3 or later, you can skip this chapter if you have it; otherwise, please update or install it.

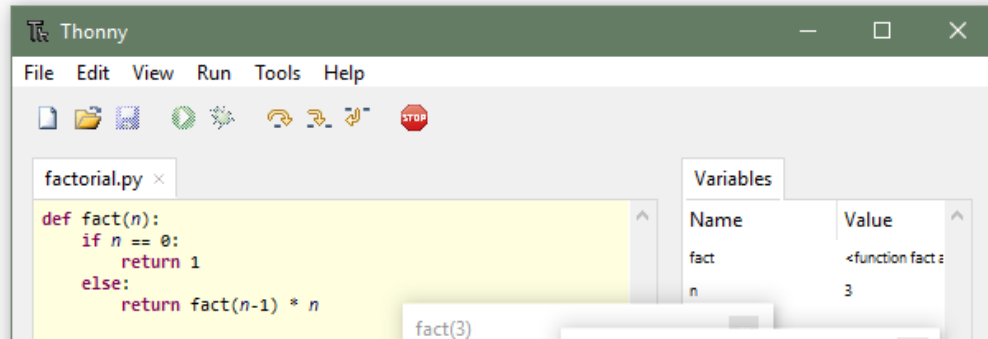
1. You can download it by visiting the website. Once open the page, you will see a light gray box in the upper right corner, click on the link that applies to your operating system.

Thonny

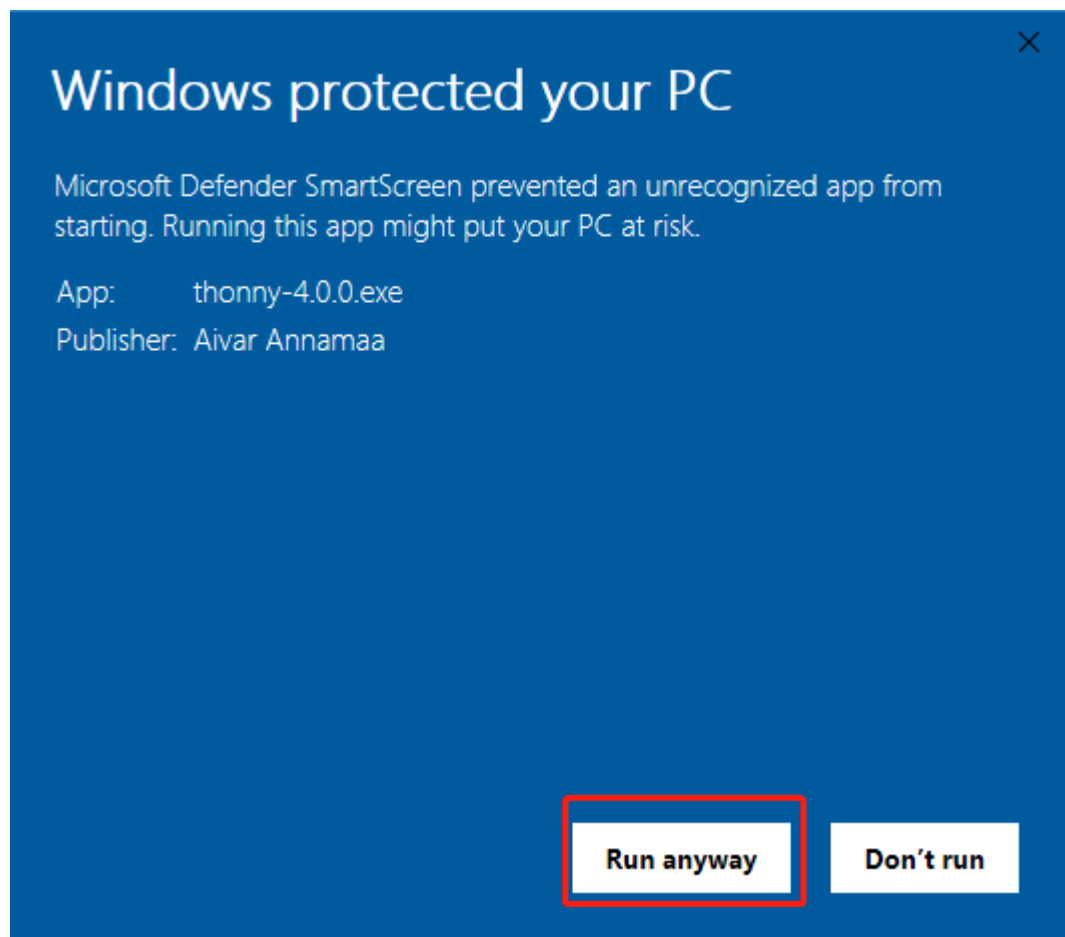
Python IDE for beginners



Download version **4.0.0** for
Windows • Mac • Linux



2. The installers have been signed with a new certificate which hasn't built up its reputation yet. You may need to click through your browser warning (e.g. choose "Keep" instead of "Discard" in Chrome) and Windows Defender warning (**More info Run anyway**).



3. Next, click **Next** and **Install** to finish installing Thonny.



3.1.2 2. Install MicroPython on Your Pico(Optional)

Warning: Since we have already installed the MicroPython firmware and related libraries for the Raspberry Pi Pico at the factory. So this step and the previous step can be skipped. But it is okay to follow the process if you want to.

MicroPython is a software implementation of a programming language largely compatible with Python 3, written in C, that is optimized to run on a microcontroller.

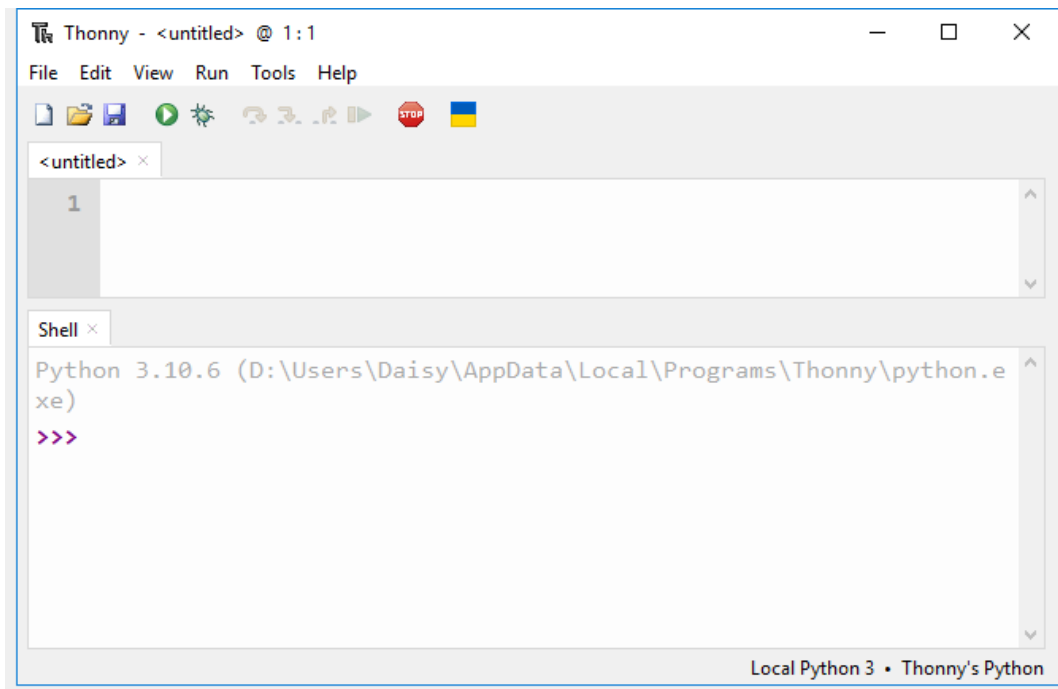
MicroPython consists of a Python compiler to bytecode and a runtime interpreter of that bytecode. The user is presented with an interactive prompt (the REPL) to execute supported commands immediately. Included are a selection of core Python libraries; MicroPython includes modules which give the programmer access to low-level hardware.

- Reference:

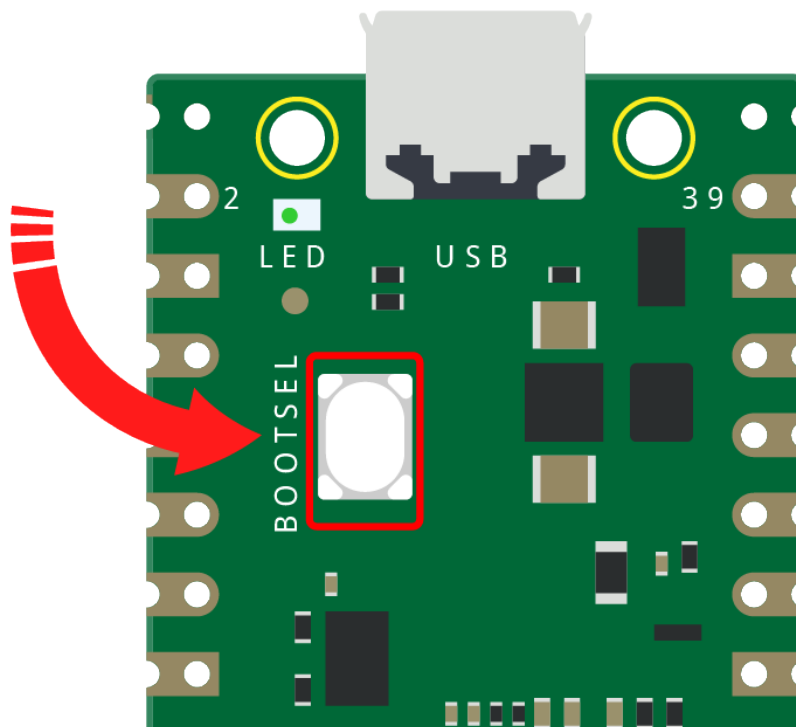
Now come to install MicroPython into Raspberry Pi Pico, Thonny IDE provides a very convenient way for you to install it with one click.

Note: If you do not wish to upgrade Thonny, you can use the Raspberry Pi official by dragging and dropping an `rp2_pico_xxxx.uf2` file into Raspberry Pi Pico.

1. Open Thonny IDE.

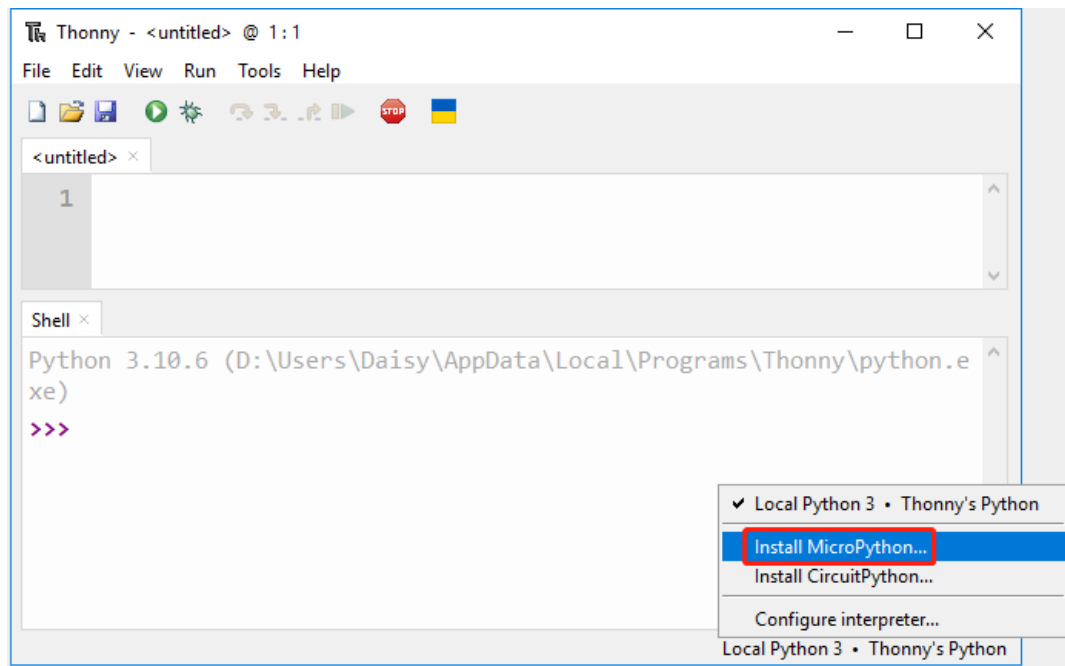


2. Press and hold the **BOOTSEL** button and then connect the Pico to computer via a Micro USB cable. Release the **BOOTSEL** button after your Pico is mount as a Mass Storage Device called **RPI-RP2**.

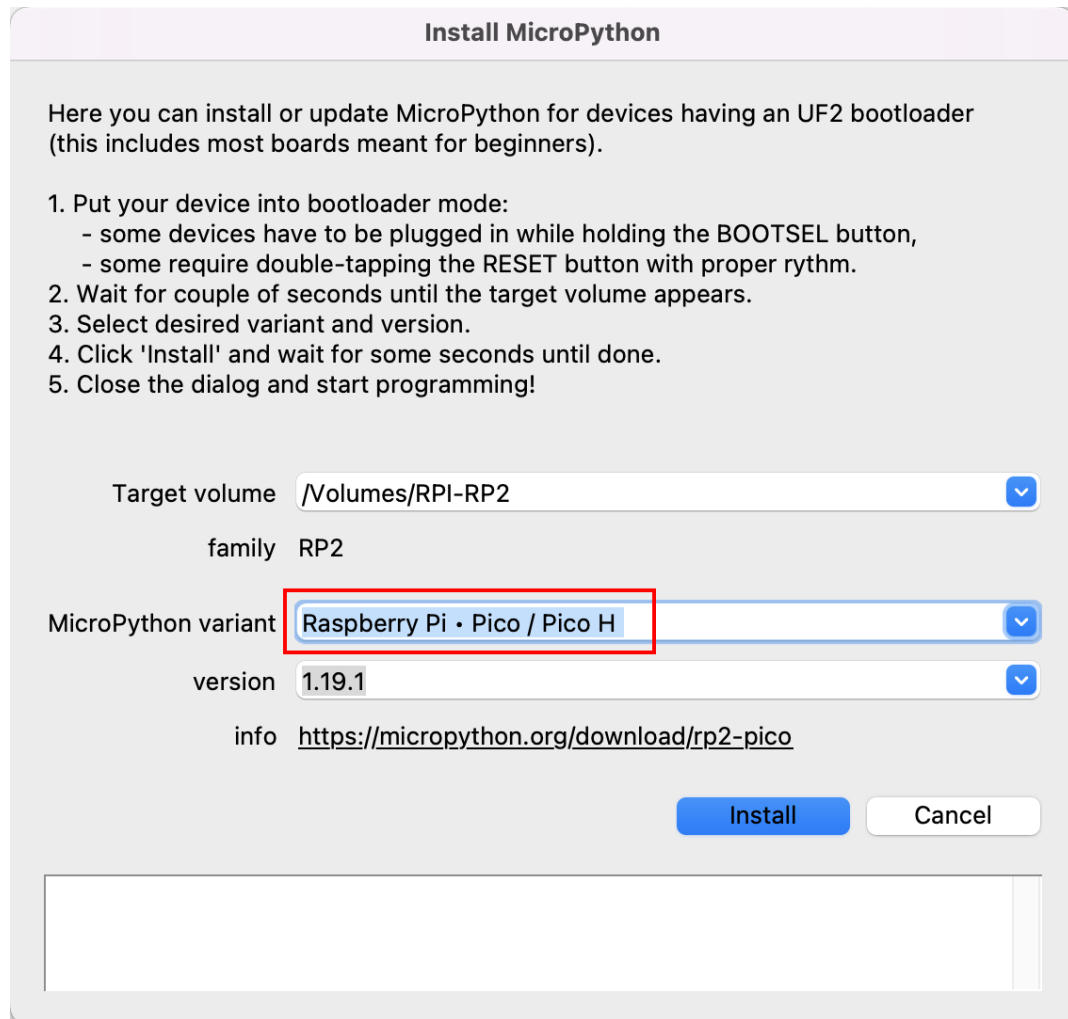


3. In the bottom right corner, click the interpreter selection button and select **Install Micropython**.

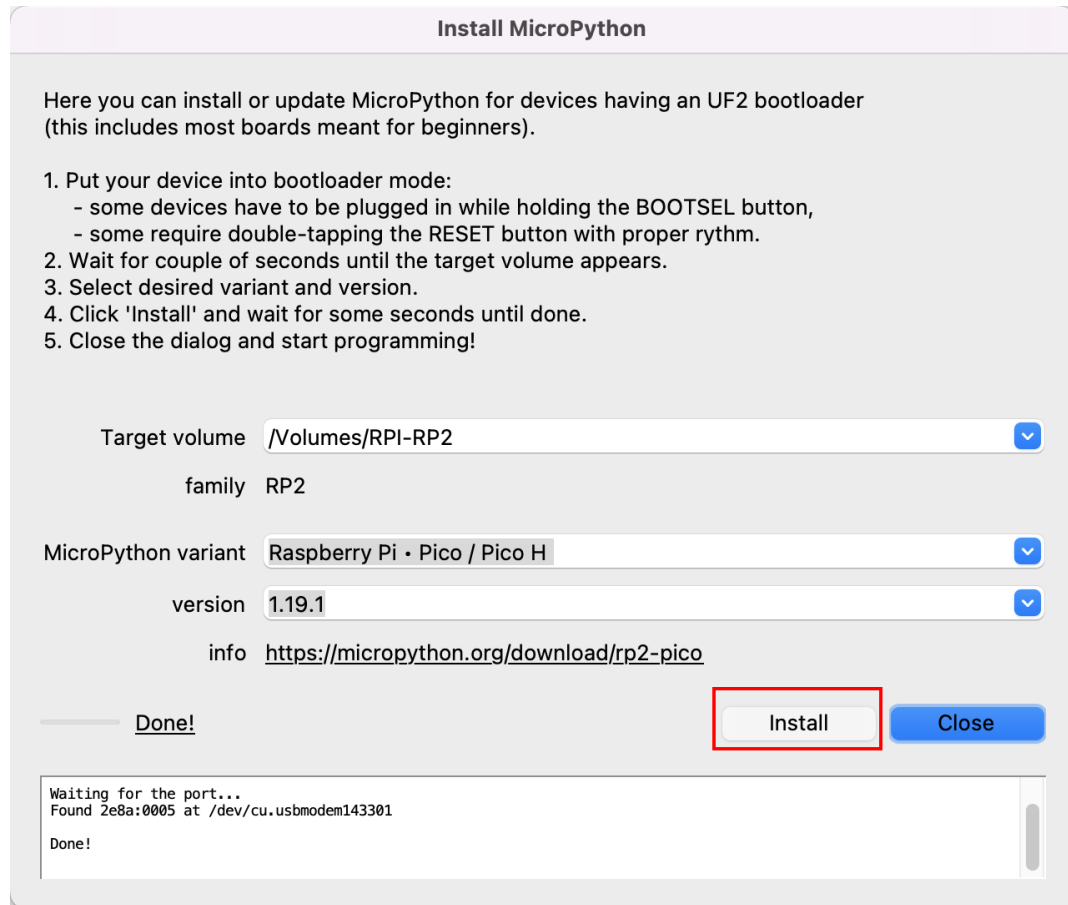
Note: If your Thonny does not have this option, please update to the latest version.



4. In the **Target volume**, the volume of the Pico you just plugged in will automatically appear, and in the **Micropython variant**, select **Raspberry Pi.Pico/Pico H**.



5. Click the **Install** button, wait for the installation to complete and then close this page.



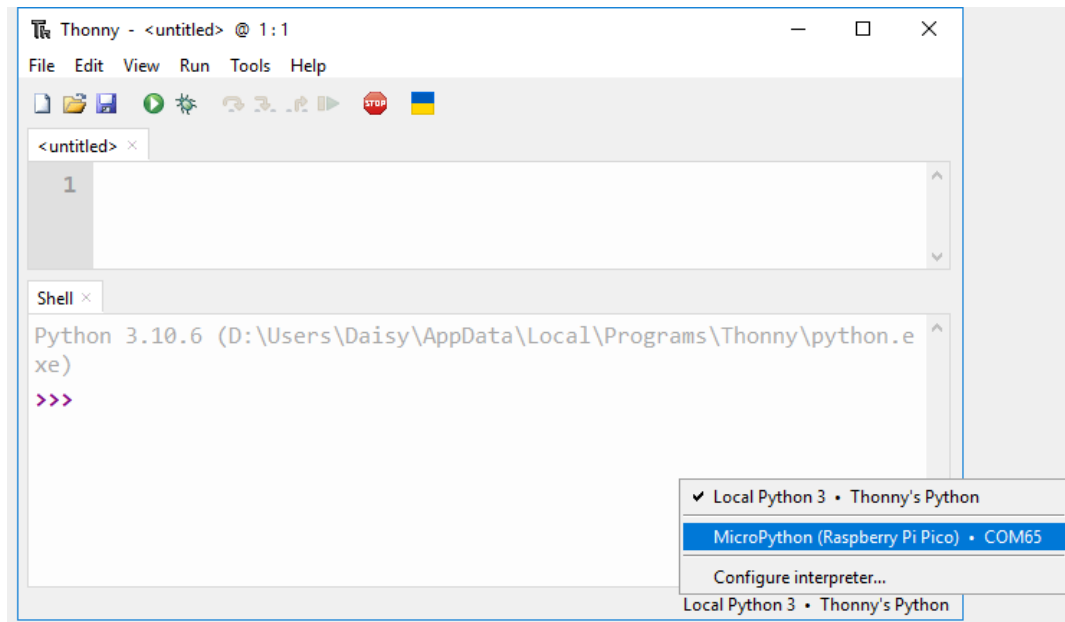
Congratulations, now your Raspberry Pi Pico is ready to go.

3.1.3 3. Upload the Libraries to Pico (Optional)

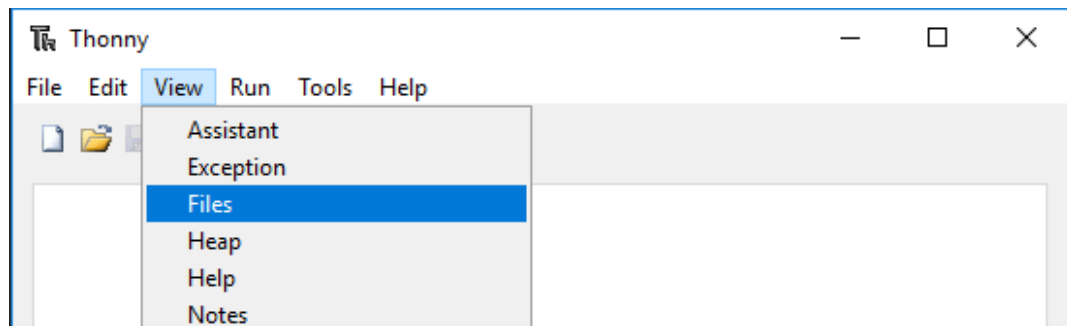
Warning: Since we have already installed the MicroPython firmware and related libraries for the Raspberry Pi Pico at the factory. So this step and the next step can be skipped. But it is okay to follow the process if you want to.

Before using Pico-4wd Car, you need to upload its related libraries in Raspberry Pi Pico.

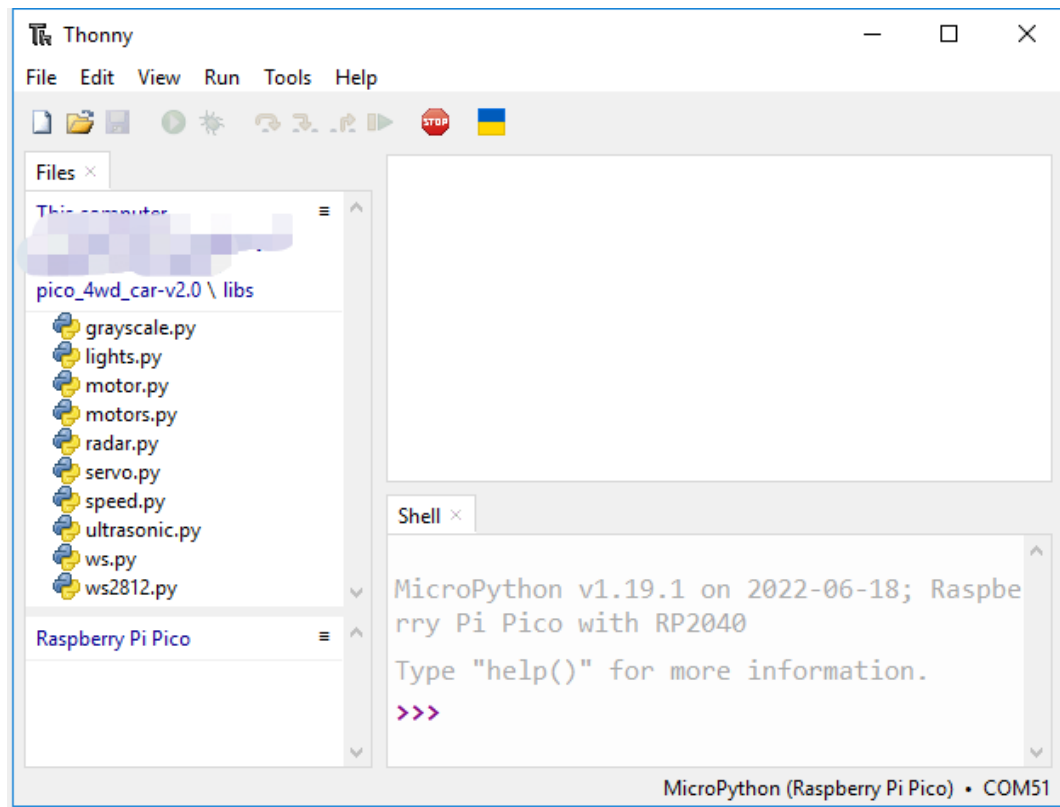
1. Download the relevant code from the link below.
 - SunFounder Pico-4wd Car V2.0 Code
2. Open Thonny IDE and plug the Pico into your computer with a micro USB cable and click on the “MicroPython (Raspberry Pi Pico).COMXX” interpreter in the bottom right corner.



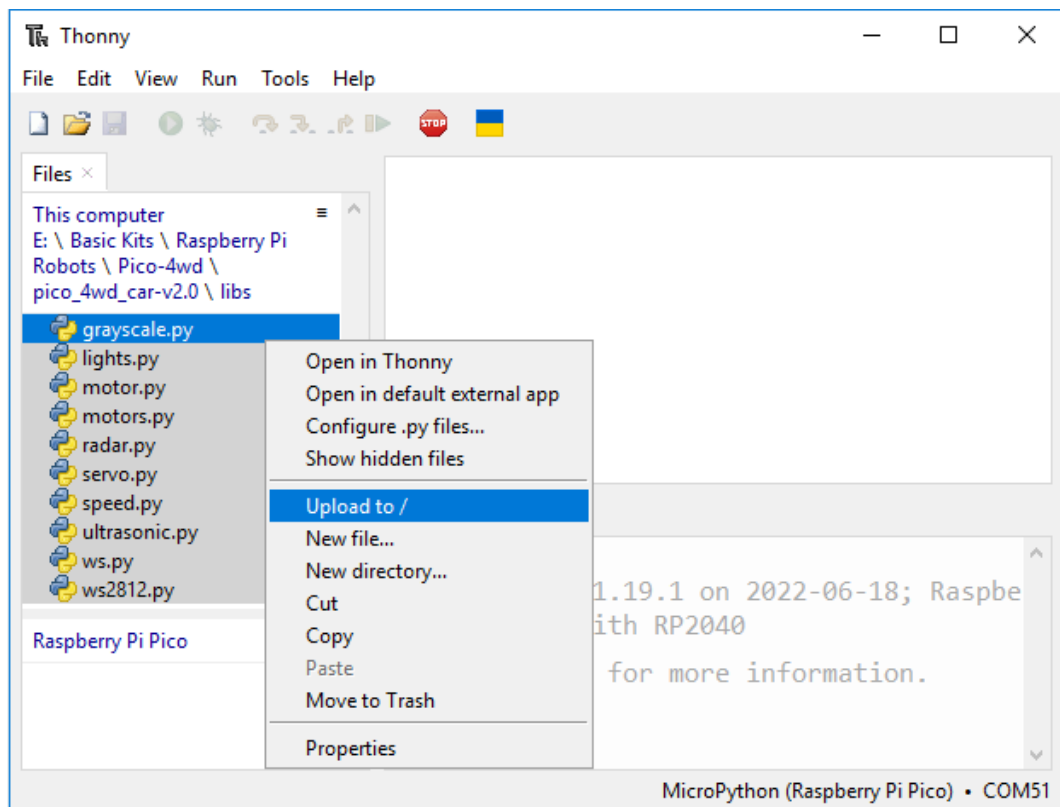
3. In the top navigation bar, click **View -> Files**.



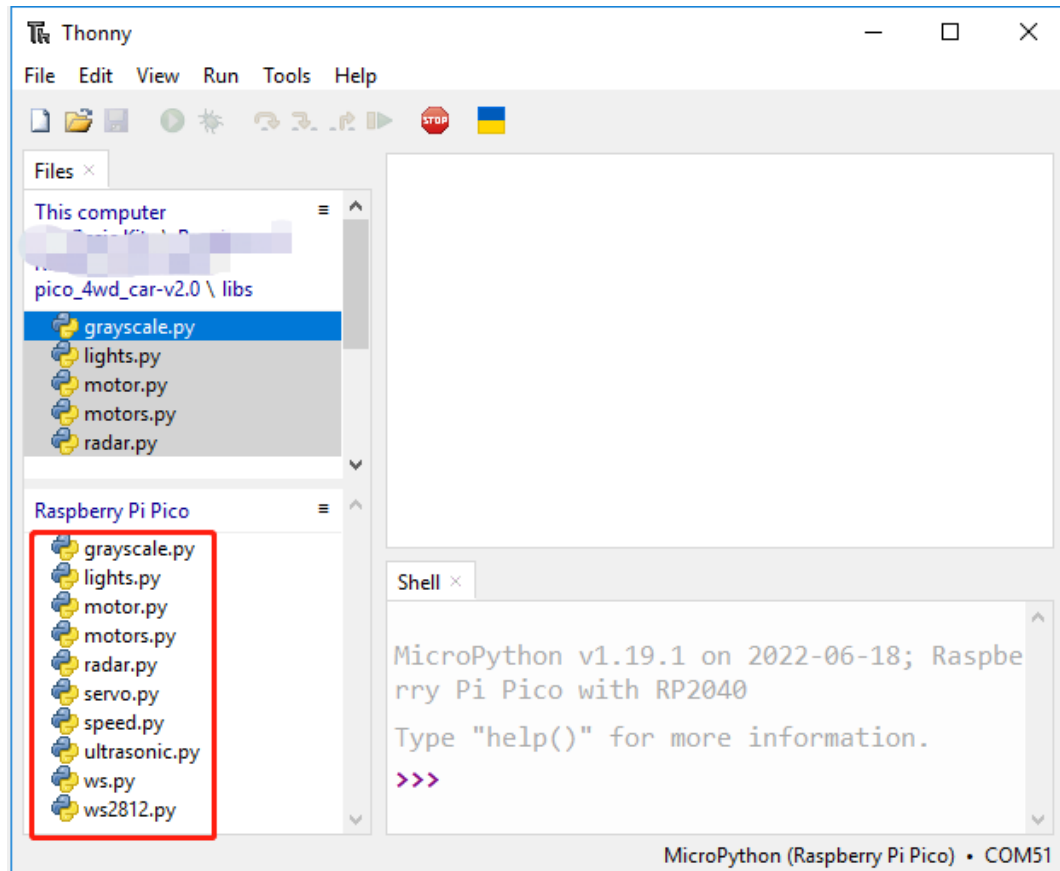
4. Switch the path to the folder where you downloaded the [code package](#) before, and then go to the `pico_4wd_car-v2.0/libs` folder.



5. Select these 3 files, right-click and click **Upload to**, it will take a while to upload.



6. Now you will see the files you just uploaded inside your drive Raspberry Pi Pico.



3.1.4 4. Run Script Online (Important)

In each project you will be prompted to copy the code into Thonny or open the `xxx.py` script in the `pico_4wd_car-v2.0\examples` path. Then run it.

If you are not familiar with operating on Thonny, you can refer to the following tutorial to learn how to open, create, run or save scripts.

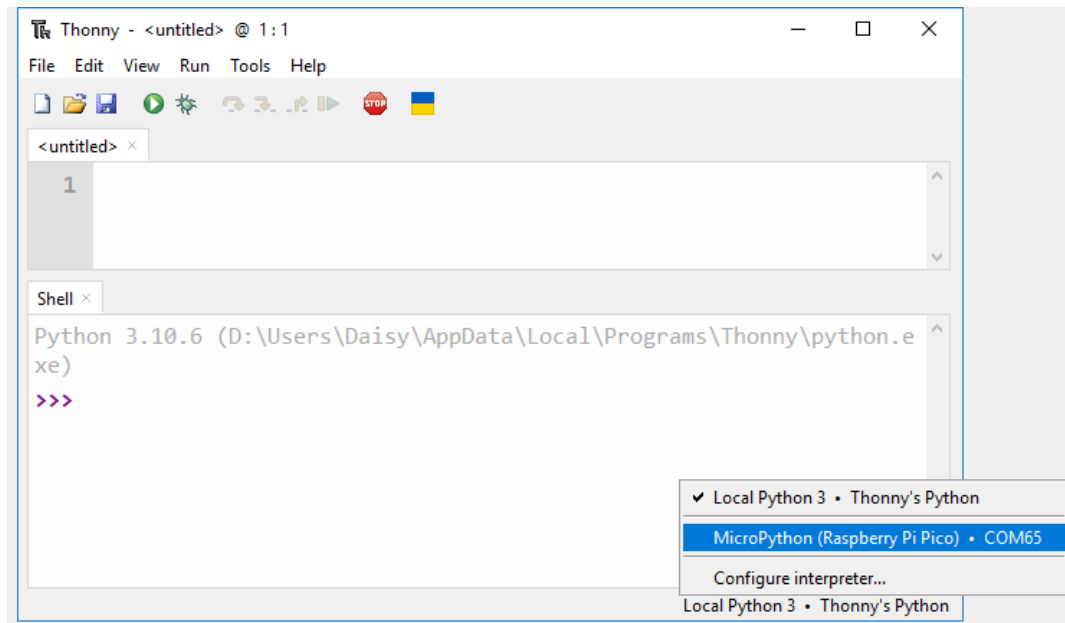
However, you must first download the package and upload the library, as described in [3. Upload the Libraries to Pico \(Optional\)](#).

Note: Here online refers to having the Raspberry Pi Pico plugged into your computer and running the script by clicking the  button or pressing F5.

The next project [5. Run Script Offline\(Important\)](#) will show you how to make the script run when the Raspberry Pi Pico is booted.

Open and Run Script Directly

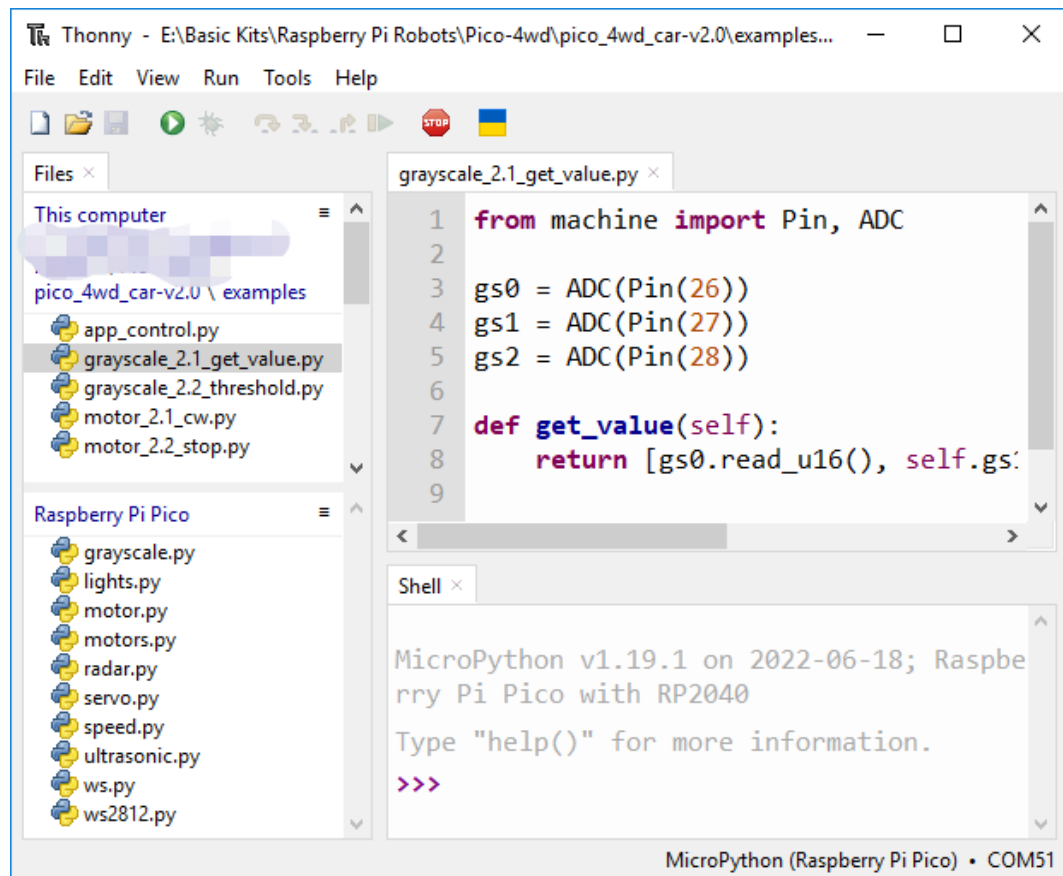
1. Open Thonny IDE and plug the Pico into your computer with a micro USB cable and click on the “MicroPython (Raspberry Pi Pico).COMxx” interpreter in the bottom right corner.



2. Open Script

For example, `grayscale_2_get_value.py`.

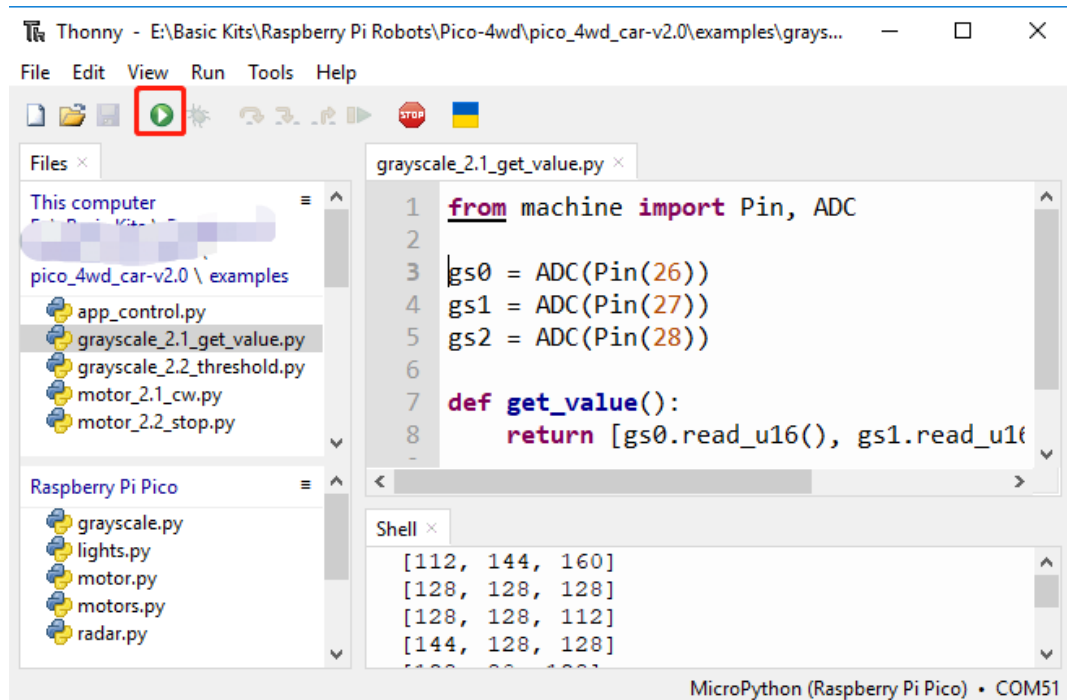
If you double click on it, a new window will open on the right. You can open more than one script at the same time.




3. Run the Script

To run the script, click the  button or press F5.

If the code contains any information that needs to be printed, it will appear in the Shell; otherwise, only the %Run -c \$EDITOR_CONTENT message will appear.



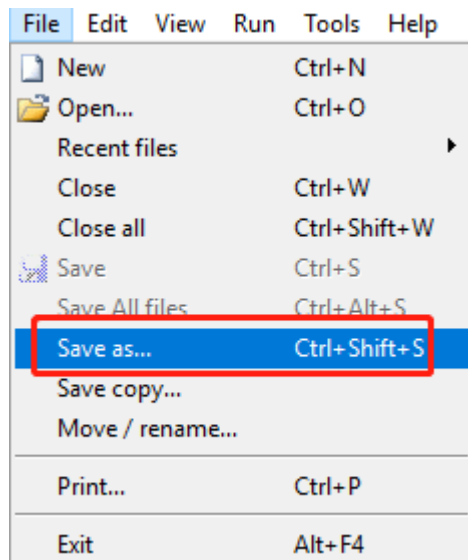
4. Stop Running

To stop the running code, click the  button. The `%RUN -c $EDITOR_CONTENT` command will disappear after stopping.

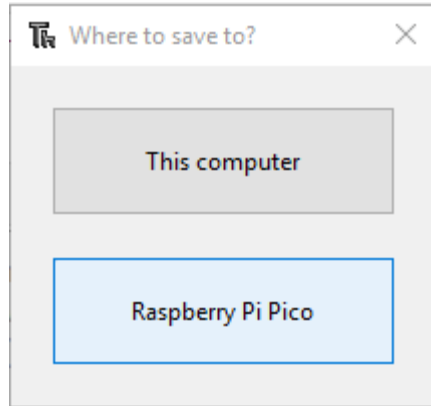
5. Save or Save as

You can save changes made to the open script by pressing **Ctrl+S** or clicking the **Save** button on Thonny.

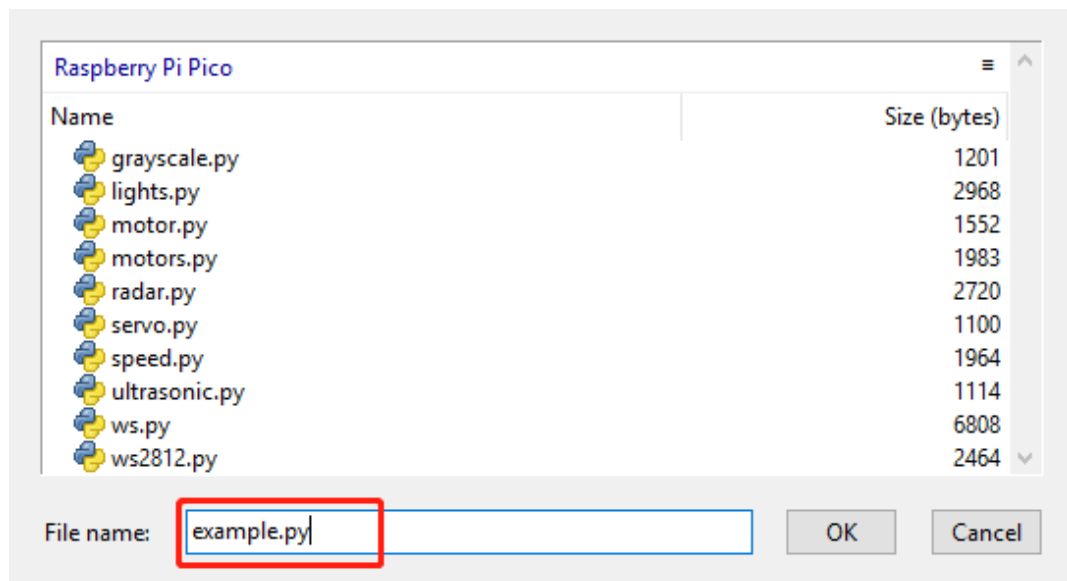
The code can be saved as a separate file within the **Raspberry Pi Pico** by clicking on **File -> Save As**.



Select **Raspberry Pi Pico**.



Then click **OK** after entering the file name and extension **.py**. On the **Raspberry Pi Pico** drive, you will see your saved file.

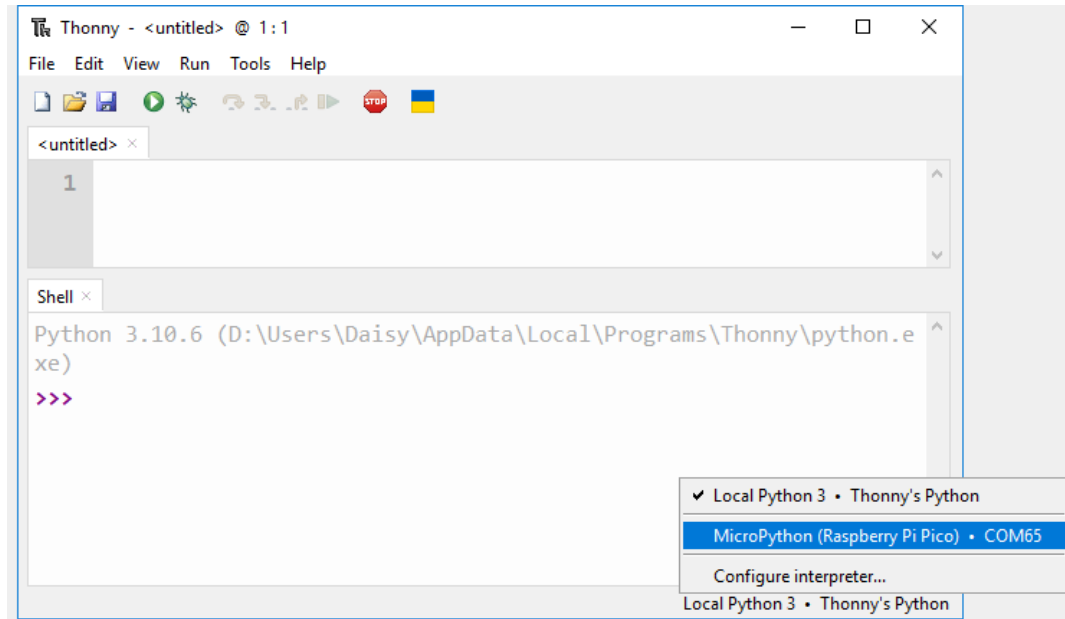


Note: Regardless of what name you give your code, it's best to describe what type of code it is, and not give it a meaningless name like `abc.py`. When you save the code as `main.py`, it will run automatically when the power is turned on.

Create File and Run it

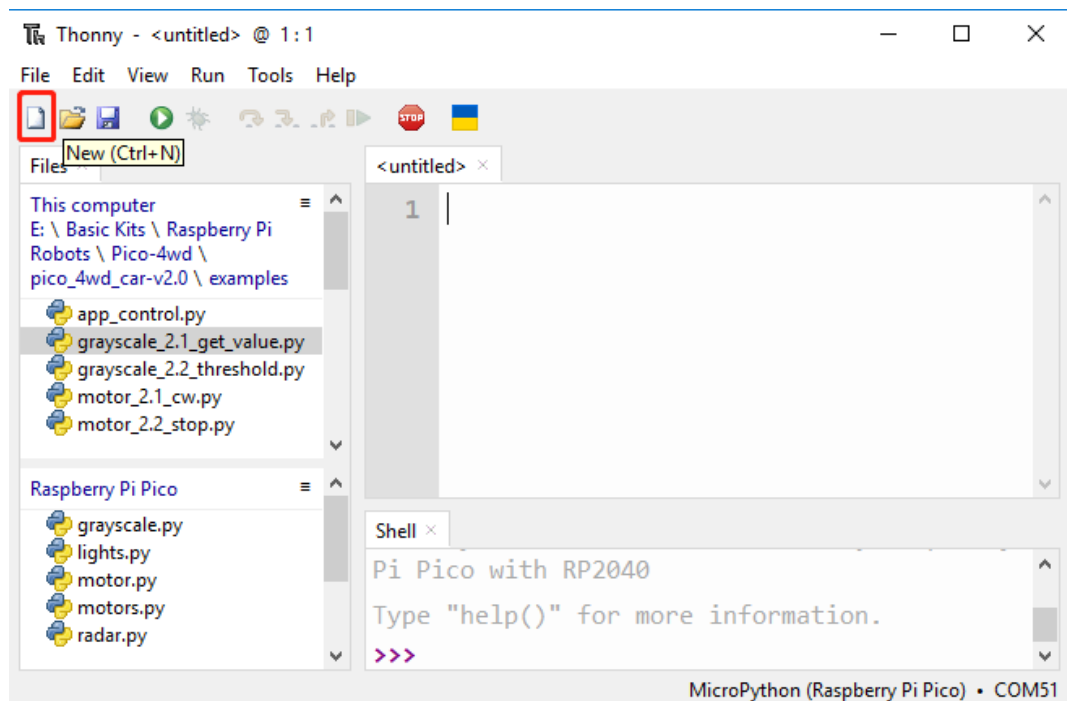
You can also create a new script to copy the code from the project page and run it.

1. Open Thonny IDE and plug the Pico into your computer with a micro USB cable and click on the “MicroPython (Raspberry Pi Pico).COMxx” interpreter in the bottom right corner.



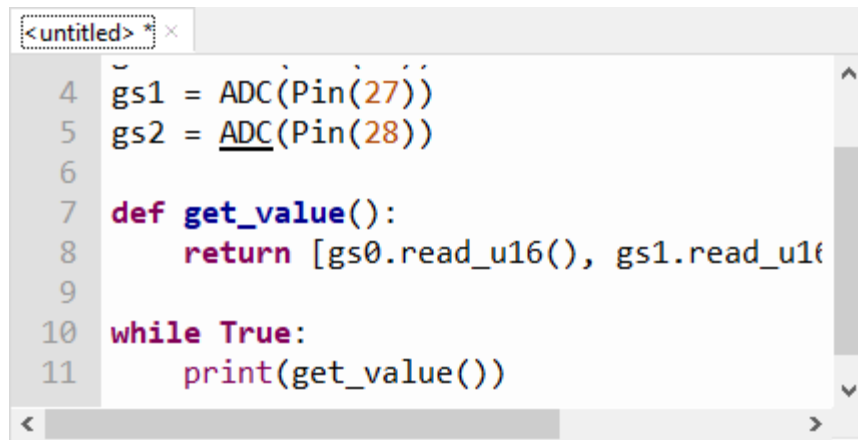
2. Create a new file

Open Thonny IDE, click **New** button to create a new blank file.



3. Copy Code

Copy the code from the project to the Thonny IDE.



```

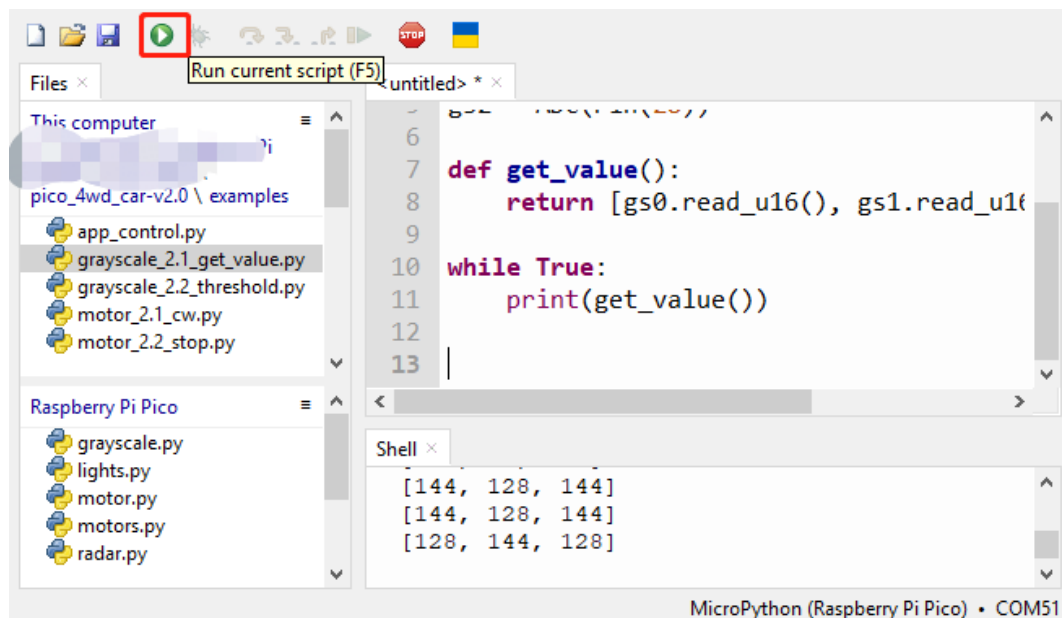
<untitled> *
4  gs1 = ADC(Pin(27))
5  gs2 = ADC(Pin(28))
6
7  def get_value():
8      return [gs0.read_u16(), gs1.read_u16()]
9
10 while True:
11     print(get_value())

```


4. Run the script

To run the script, click the  button or press F5.

If the code contains any information that needs to be printed, it will appear in the Shell; otherwise, only the %Run -c \$EDITOR_CONTENT message will appear.

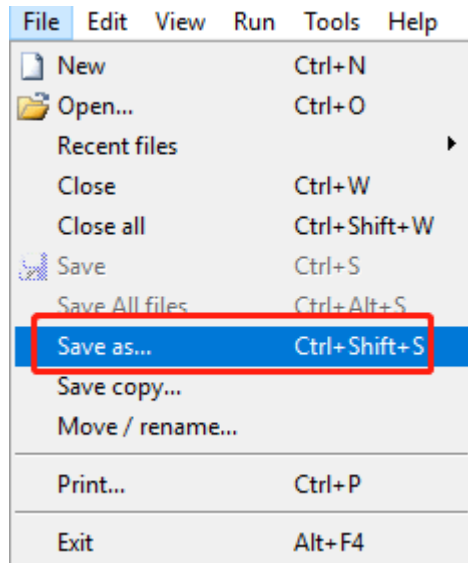


5. Stop Running

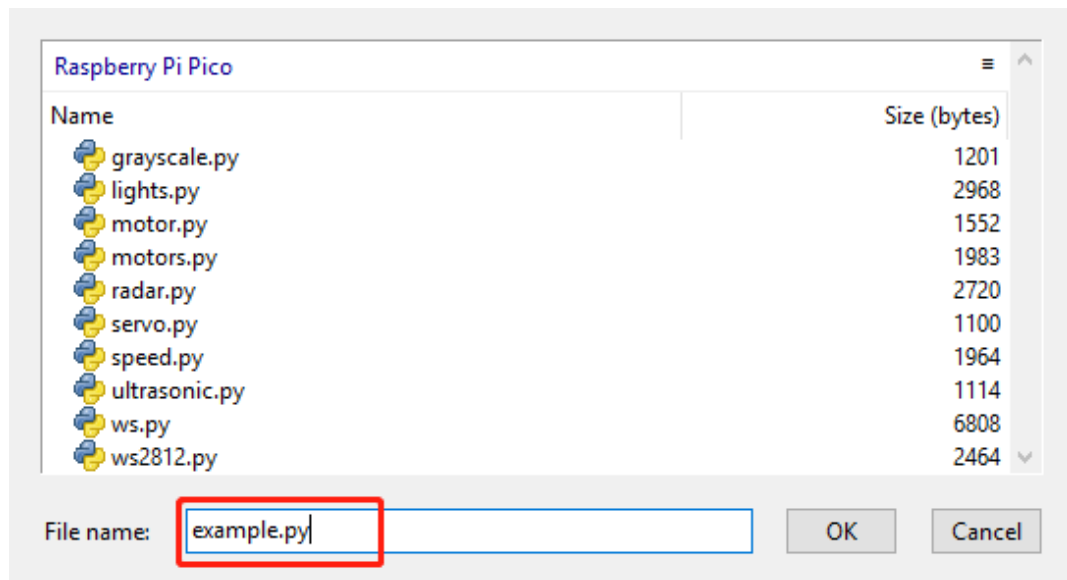
To stop the running code, click the  button. The %RUN -c \$EDITOR_CONTENT command will disappear after stopping.

6. Save

The script can be saved as a separate file within the **Raspberry Pi Pico** or **This Computer** by pressing **Ctrl+S** or clicking the **Save** button on Thonny.




Then click **OK** after entering the file name and extension `.py`.



Note: Regardless of what name you give your code, it's best to describe what type of code it is, and not give it a meaningless name like `abc.py`. When you save the code as `main.py`, it will run automatically when the power is turned on.

3.1.5 5. Run Script Offline(Important)

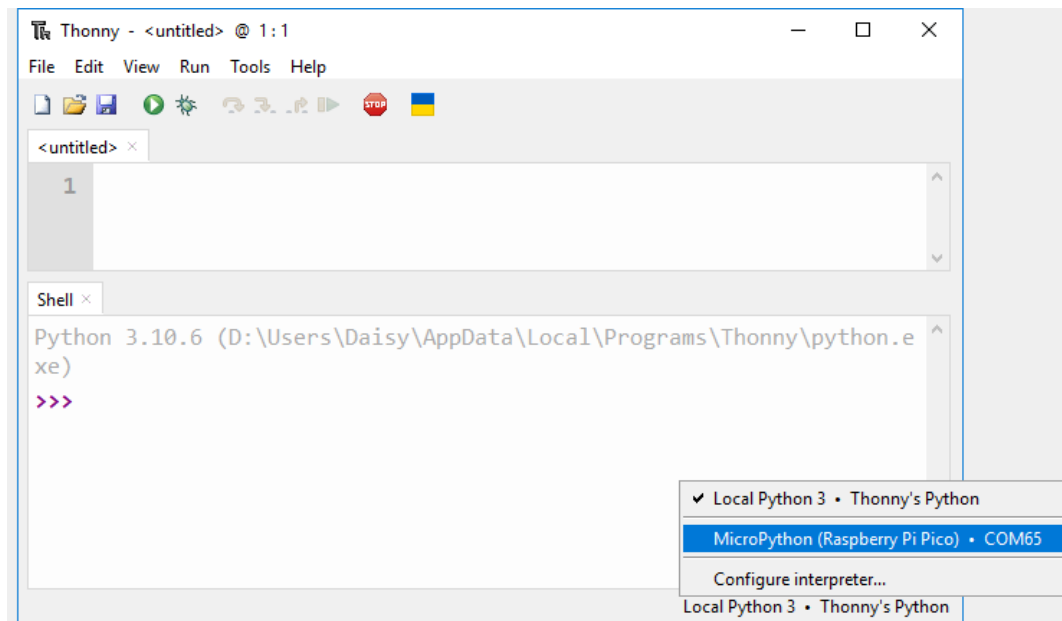
We can click the  button in Thonny to get the script running in the Raspberry Pi Pico, but this method requires you to connect the Raspberry Pi Pico to your computer with a Micro USB cable.

This method is not very convenient if your Pico 4WD car needs to go on the ground and move around.

So how do you get scripts to work without the USB cable?

An easy way to do this is to save this script as `main.py` to the Raspberry Pi Pico.

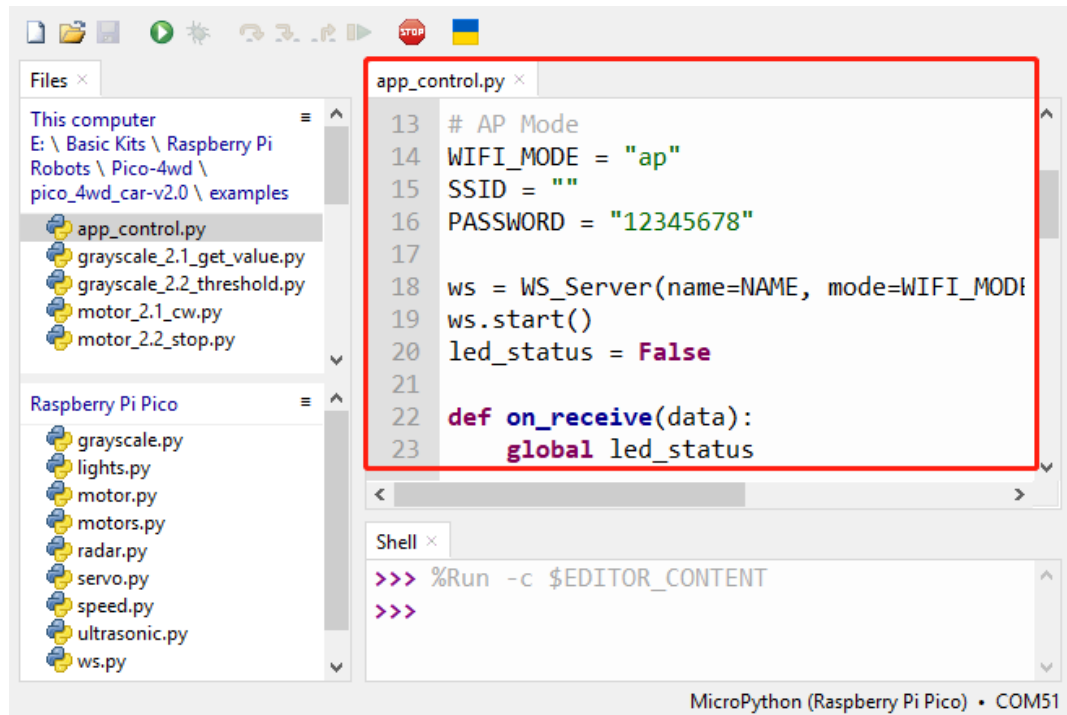
1. Open Thonny IDE and plug the Pico into your computer with a micro USB cable and click on the “MicroPython (Raspberry Pi Pico).COMxx” interpreter in the bottom right corner.



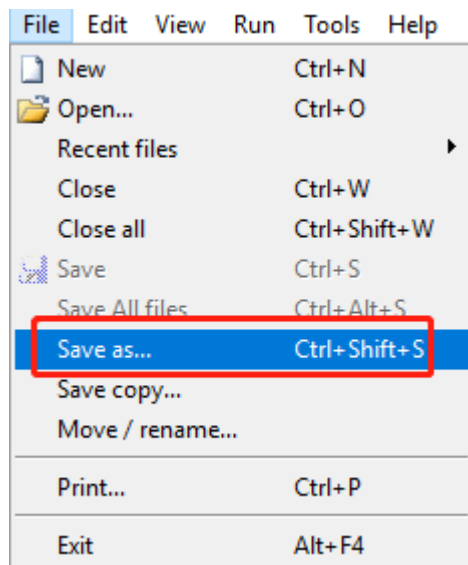
2. Open the script first.

For example, `app_control.py`.

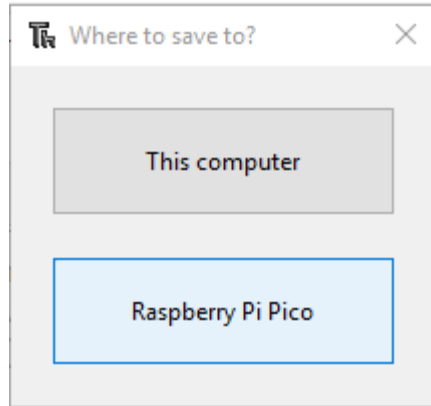
If you double click on it, a new window will open on the right.



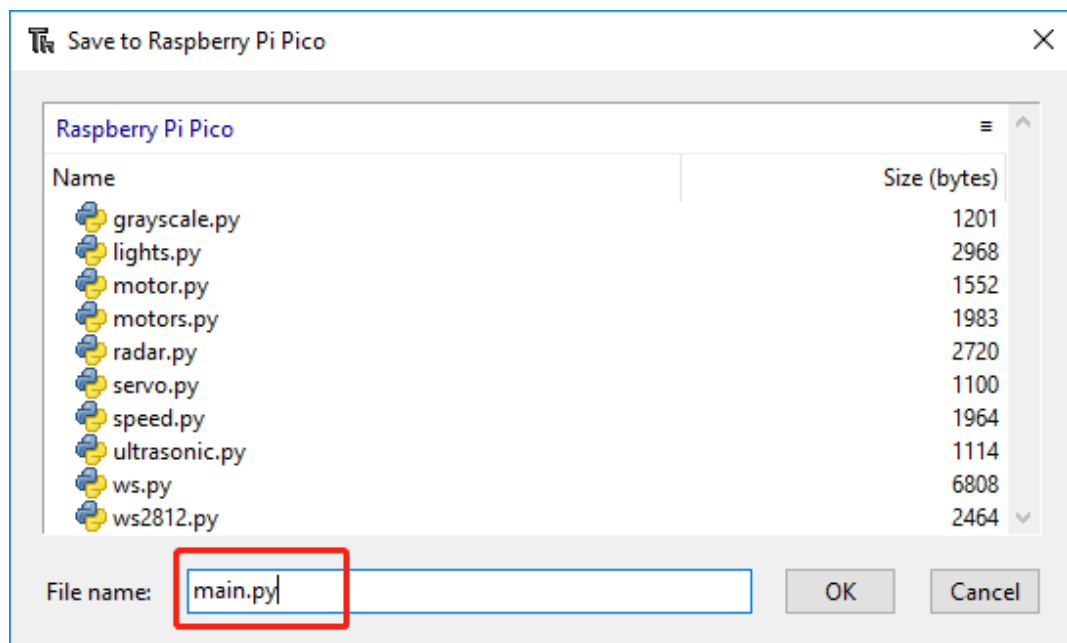
3. Then click **File -> Save As**.



4. Select **Raspberry Pi Pico** in the pop-up window that appears.



5. Set the file name to `main.py`. If you already have the same file in your Pico, it will prompt to overwrite it.



6. Now you can unplug the USB cable, turn on the power switch of Pico-4wd, and Pico-4wd will automatically run this `main.py` script.

3.2 2. Learn Modules

Here, you will learn how each module works, and how to use them in scripts as well as write all the relevant code to the library to be used later in your projects.

3.2.1 1. Motor

In this chapter, you will learn how motors work and how the 4 motors work together to make the car move.

From driving the 4 motors directly with high and low levels, to using PWM to set the speed, to encapsulating the motor related code into modules (libraries).

With this step-by-step method, you can learn how to move the car thoroughly.

1. Introduce the Motor



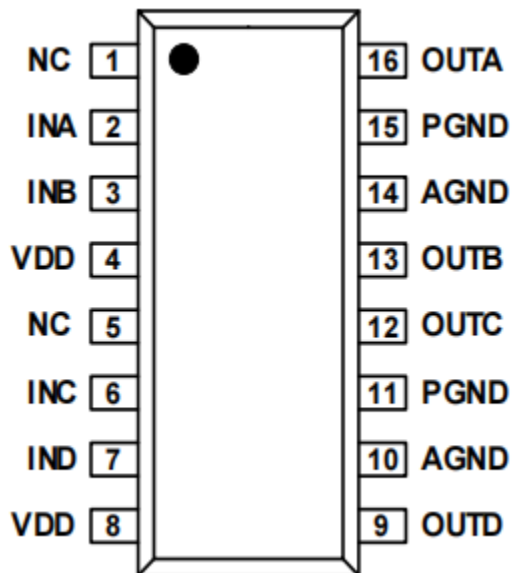
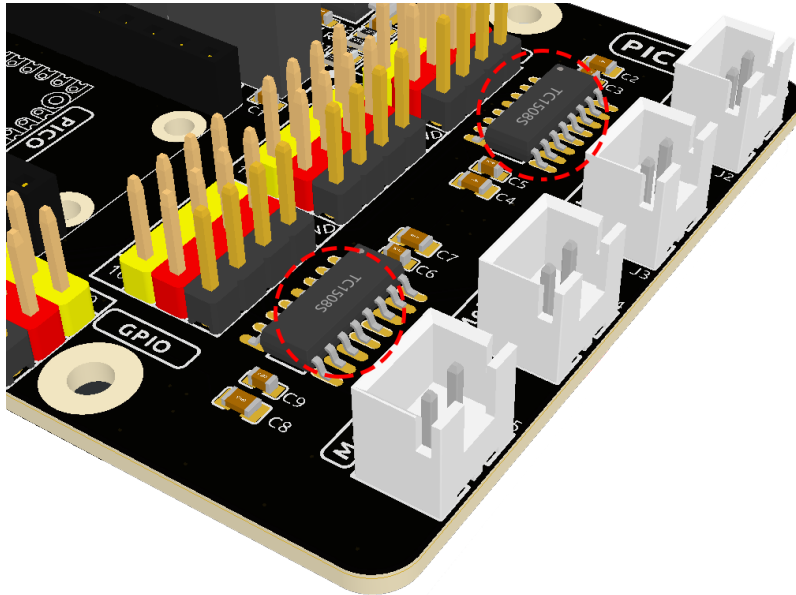
- Rated Voltage: 3~6V
- Continuous No-Load Current: 150mA +/- 10%
- Min. Operating Speed (3V): 90+/- 10% RPM
- Min. Operating Speed (6V): 200+/- 10% RPM
- Stall Torque (3V): 0.4kg.cm
- Stall Torque (6V): 0.8kg.cm
- Gear Ratio: 1:48
- Body Dimensions: 70 x 22.3 x 36.9mm
- Wires: Gray and Black, 24AWG, 250mm
- Connector: XH2.54-2P
- Weight: 30.6g

This kit comes with 4 TT Motors, which are ordinary DC motors with matching gearboxes to provide low speed but high torque. This motor has a gear ratio of 1:48 and comes with 2 x 250mm wires with XH2.54-2P connector.

When the motor is powered, it rotates in one direction. Invert the polarity of the power supply, and the motor will rotate the other way.

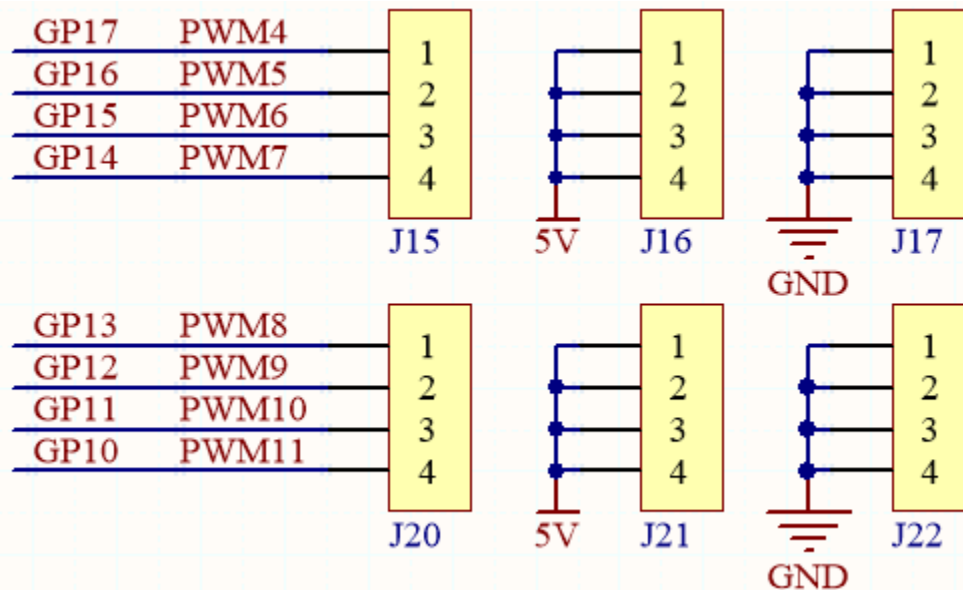
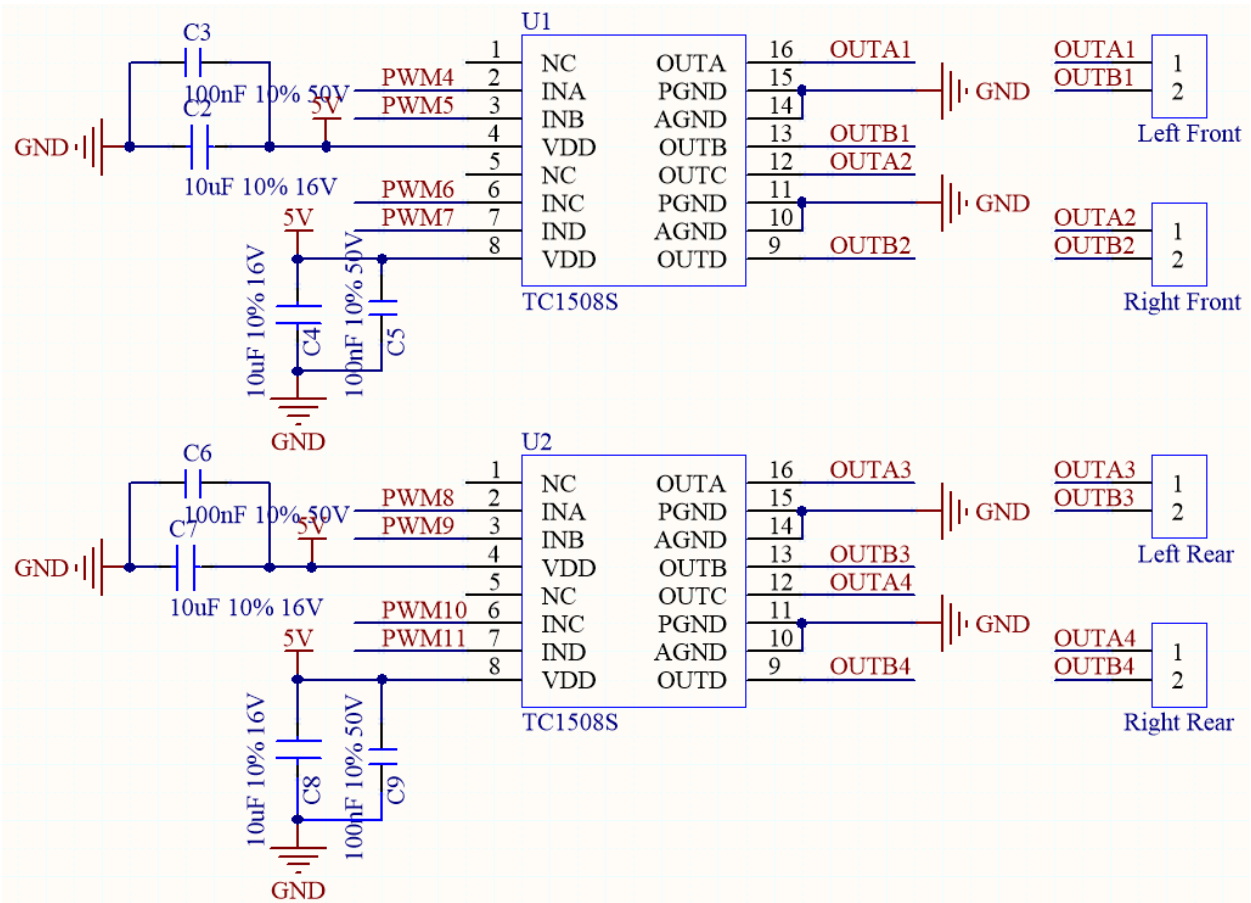
As this kind of motor requires a lot of current, using the main board's IO port directly may not work and may damage the board. Hence, a motor driver chip is required.

TC1508S Chip

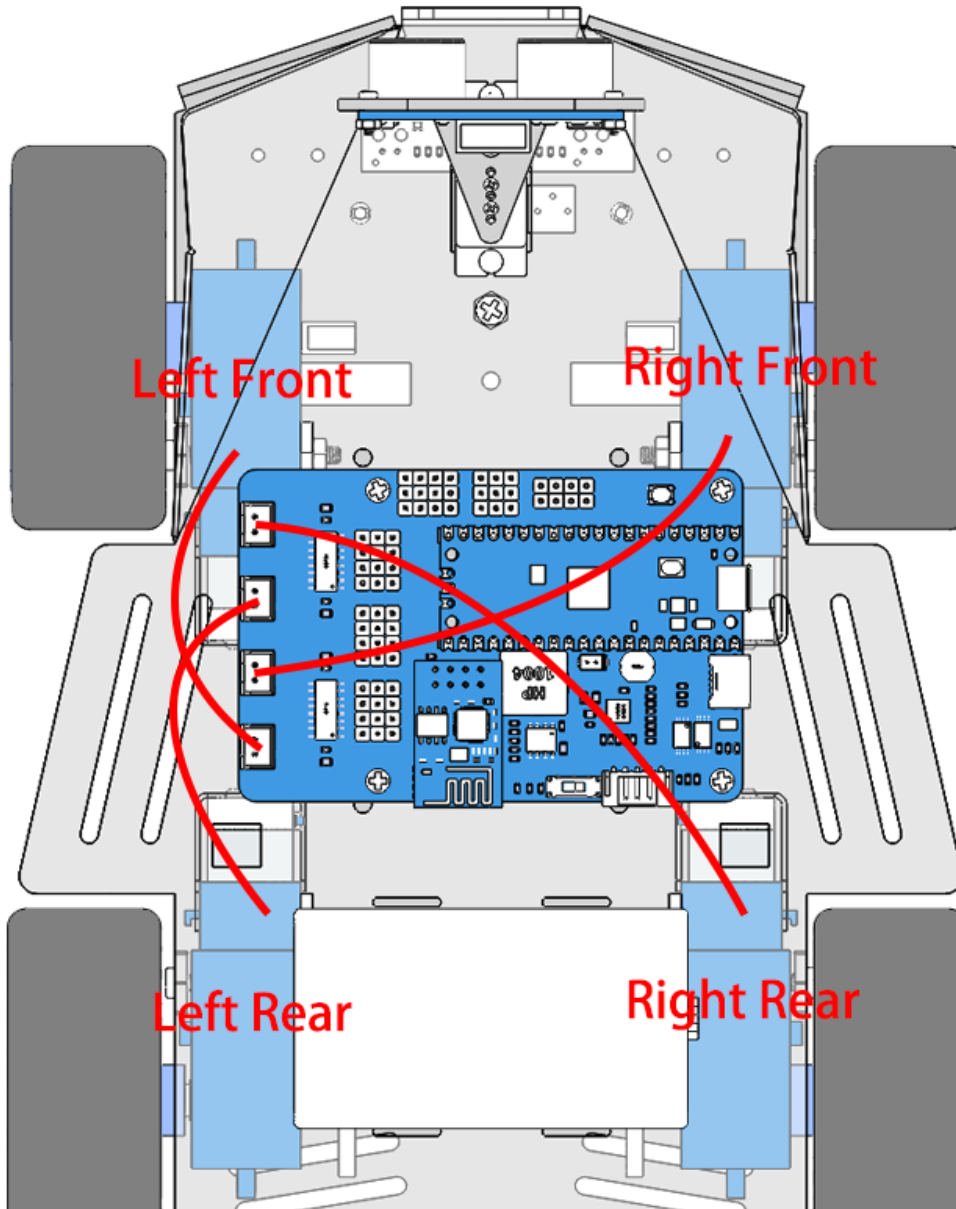


The TC1508S is used in the Pico RDP (Robotics Development Platform), which can drive two motors at the same time with a wide operating voltage range of 2.2~5.5V and a maximum continuous output current of 1.8A.

Two TC1508S chips are used here to drive the four motors, and the corresponding schematic is shown below.



And the corresponding positions of the motors on the car are as follows.



So the corresponding control pins and rotation directions of the 4 motors are as follows.

Table 1: **Motor-Pin List**

Motor	PinA	PinB
Left Front	GP17	GP16
Right Front	GP15	GP14
Left Rear	GP13	GP12
Right Rear	GP11	GP10

Table 2: Motor Work

PinA	PinB	Work
H	L	Rotate Clockwise(CW)
L	H	Rotate Counter-clockwise(CCW)
H	H	Stop

2. Get the Motor Rotating

Principle of Motor Rotation

As shown in *1. Introduce the Motor*, the motors of the Pico 4WD Car are driven by the TC1508S chips. The control pins corresponding to the 4 motors and the direction of rotation are shown below.

Table 3: Motor-Pin List

Motor	PinA	PinB
Left Front	GP17	GP16
Right Front	GP15	GP14
Left Rear	GP13	GP12
Right Rear	GP11	GP10

Table 4: Motor Work

PinA	PinB	Work
H	L	Rotate Clockwise(CW)
L	H	Rotate Counter-clockwise(CCW)
H	H	Stop

Now let's start writing the script to see how the motors turn.

Motor Turns Clockwise

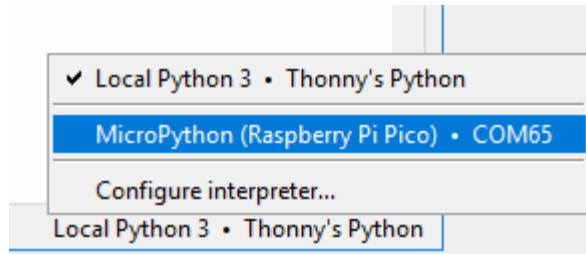
1. Take Right Rear Motor for example, it is controlled by GP11 and GP10. Write low for GP11 and high for GP10.


```
import machine

pinA = machine.Pin(11, machine.Pin.OUT)
pinB = machine.Pin(10, machine.Pin.OUT)

pinA.low()
pinB.high()
```

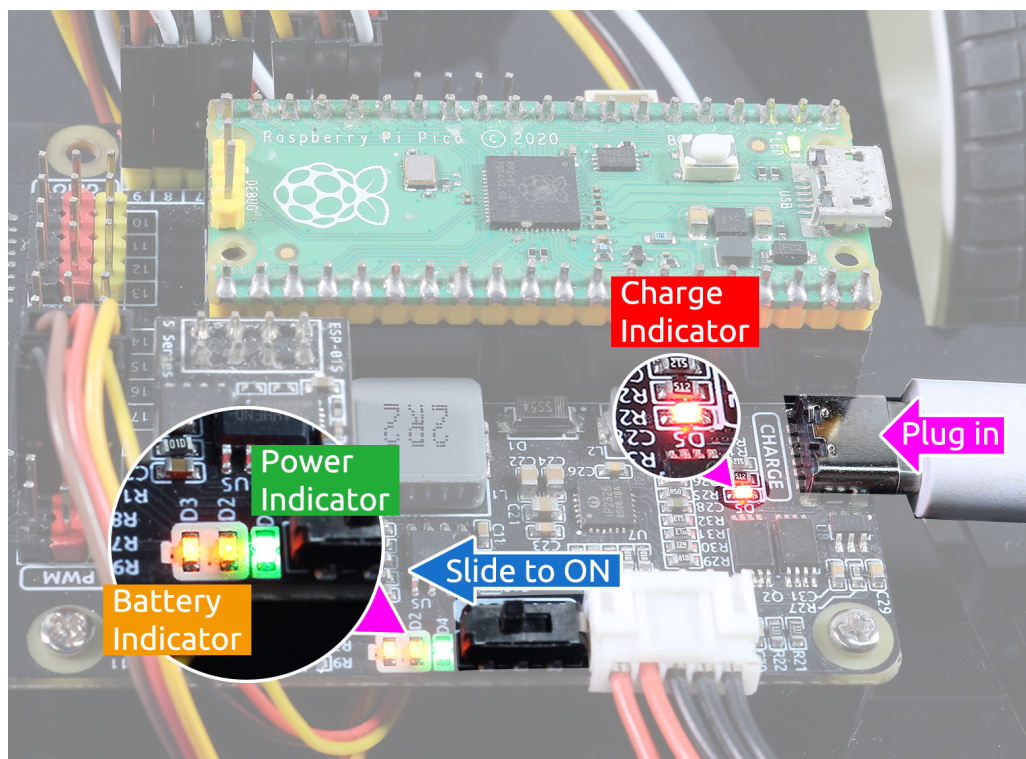
2. Copy the above code into Thonny or open the `motor_2_cw.py` under the path of `pico_4wd_car-v2.0\examples\learn_modules`.
3. Use a micro USB cable to connect the Pico to your computer and select the "MicroPython (Raspberry Pi Pico) COMxx" interpreter.



4. Click  button or simply press F5 to run it.

5. Start the Pico 4WD car.

- When first used or when the battery cable is unplugged, Pico RDP will activate its over-discharge protection circuitry (Unable to get power from battery).
- Therefore, you'll need to plug in a Type-C cable for about 5 seconds to release the protection status.
- At this time look at the battery indicators, if both battery indicators are off, please continue to plug in the Type-C cable to charge the battery.



6. As you hold the Pico 4WD Car up high, you will be able to see the right rear motor turning clockwise.

Next you can run the following scripts in sequence to see what happens.

Stop the Motor

Write two high levels, motor stops.

```
import machine

pinA = machine.Pin(11, machine.Pin.OUT)
pinB = machine.Pin(10, machine.Pin.OUT)
```

(continues on next page)

(continued from previous page)

```
pinA.high()
pinB.high()
```

Motor Turns Counter-clockwise

Reversing the high and low levels, the motor will rotate counterclockwise.

```
import machine

pinA = machine.Pin(11, machine.Pin.OUT)
pinB = machine.Pin(10, machine.Pin.OUT)

pinA.high()
pinB.low()
```

Stop 4 Motors

Stop all motors.

```
import machine

for i in range(10,18):
    pin = machine.Pin(i, machine.Pin.OUT)
    pin.high()
```

3. Controlling the Speed of Motors

In the previous project, we can give high and low levels to the motor to make it spin or stop in a constant speed.

But how to set different speed?

About PWM

In this case, you can use , it allows you to give analogue behaviours to digital devices, such as motors. This means that rather than motor being simply spin or stop, you can control its speed.

Happily, handling Pico PWM in MicroPython is very simple, requiring only three simple commands.

- Create a pin as a PWM object.

```
pin = machine.PWM(machine.Pin(17))
```

- Tell Raspberry Pi Pico how often to switch the power between ON and OFF.

```
pin.freq(20000)
```

- Tell the pin for how long it should be ON each time. For Raspberry Pi Pico in MicroPython, this can range from 0 to 65535. 65535 would be 100% of the time, a value of around 32727 indicates 50%.

```
pin.duty_u16(0xFFFF)
```

Rotating the motor at high speed (motor_3_set_speed.py)

- Set pinB to high, i.e. PWM value is 0xFFFF (65535).
- Then the motor will rotate clockwise by setting pinA to any value between 0 and 0xFFFF. In general, the smaller the value, the faster the speed, and the maximum speed is obtained at 0.

```
import machine

pinA = machine.Pin(17, machine.Pin.OUT)
pinB = machine.Pin(16, machine.Pin.OUT)

pwmA = machine.PWM(pinA) # Create PWM object
pwmB = machine.PWM(pinB) # Create PWM object
pwmA.freq(20000)
pwmB.freq(20000)

# fast
pwmA.duty_u16(0x0000)
pwmB.duty_u16(0xFFFF)
```

Rotating motor at slow speed

Now turn the left front motor clockwise in the same way, but you will find that it will spin much slower.

```
import machine

pinA = machine.Pin(13, machine.Pin.OUT)
pinB = machine.Pin(12, machine.Pin.OUT)

pwmA = machine.PWM(pinA)
pwmB = machine.PWM(pinB)
pwmA.freq(20000)
pwmB.freq(20000)

# slow
pwmA.duty_u16(0xFFFF)
pwmB.duty_u16(0xAAAA)
```

Stop 4 Motors

To stop all motors, write 0xFFFF to all PWM pins.

```
import machine

for i in range(10,18):
    pin = machine.PWM(machine.Pin(i, machine.Pin.OUT))
    pin.freq(20000)
    pin.duty_u16(0xFFFF)
```

4. Move the Car

Previously, we have learned how to make a single motor turn at different speeds.

In this project, we will learn how 4 motors work together to make the Pico 4WD car go forward, backward, left and right.

You can push the Pico 4WD car forward to see how the 4 wheels (motors) are turning. You will find that the two wheels (motors) on the left side are turning counterclockwise, while the two wheels (motors) on the right side are turning clockwise.

So you can use a parameter to determine whether the motor is on the left or the right side, and thus set its direction of rotation.

Move Forward (motor_4_forward.py)

- Here, create a function `motor_run()` to control the movement of the car, and use the parameter `position` to determine the position of the motors.
- When `position > 0`, it means the corresponding motor is on the **right side**, so let the motor turn clockwise.
- When `position < 0`, it means the corresponding motor is on the **left side**, so let the motor turn counterclockwise.

```
import machine
import time

pin = []
motor_pin = [17,16,15,14,13,12,11,10]

for i in range(8):
    pin.append(None)
    pin[i] = machine.PWM(machine.Pin(motor_pin[i],machine.Pin.OUT))
    pin[i].freq(20000)

def motor_run(power,pinA,pinB,position):
    power= int(power/100.0*0xFFFF)
    if position>0:
        # clockwise
        pinA.duty_u16(0xFFFF-power)
        pinB.duty_u16(0xFFFF)
    elif position<0:
        # anticlockwise
        pinA.duty_u16(0xFFFF)
        pinB.duty_u16(0xFFFF-power)

try:

    power = 50

    # forward
    motor_run(power,pin[0],pin[1],-1) #left front
    motor_run(power,pin[2],pin[3],1) #right front
    motor_run(power,pin[4],pin[5],-1) #left rear
    motor_run(power,pin[6],pin[7],1) #right rear
    time.sleep(2)

finally:
    # stop
    power = 0
    motor_run(power,pin[0],pin[1],-1) #left front
    motor_run(power,pin[2],pin[3],1) #right front
    motor_run(power,pin[4],pin[5],-1) #left rear
    motor_run(power,pin[6],pin[7],1) #right rear
```

- Copy the above code into Thonny or open the `motor_4_forward.py` under the path of `pico_4wd_car-v2.0\examples\learn_modules`.
- After running and powering up the Pico 4WD car, you will see the car move forward.

Note: In order to allow the car to move on the ground without the USB cable connected, you need to save this script as `main.py` to Raspberry Pi Pico, see [5. Run Script Offline\(Important\)](#) for a tutorial.

Move Backward

- As well as forward, the car also has stop, backward, and steering situations. To reverse the direction of the motor, we would normally write a negative number to power, but the above code would return errors.
- Therefore, the above code requires one more layer of conditional judgment, as shown below.

```
import machine
import time

pin = []
motor_pin = [17,16,15,14,13,12,11,10]

for i in range(8):
    pin.append(None)
    pin[i] = machine.PWM(machine.Pin(motor_pin[i],machine.Pin.OUT))
    pin[i].freq(20000)

def motor_run(power,pinA,pinB,position):
    value= int(abs(power)/100.0*0xFFFF)
    if position*power>0:
        # clockwise
        pinA.duty_u16(0xFFFF-value)
        pinB.duty_u16(0xFFFF)
    elif position*power<0:
        # anticlockwise
        pinA.duty_u16(0xFFFF)
        pinB.duty_u16(0xFFFF-value)
    elif position*power==0:
        # stop
        pinA.duty_u16(0xFFFF)
        pinB.duty_u16(0xFFFF)

try:

    power = 50

    # backward
    motor_run(-power,pin[0],pin[1],-1) #left front
    motor_run(-power,pin[2],pin[3],1) #right front
    motor_run(-power,pin[4],pin[5],-1) #left rear
    motor_run(-power,pin[6],pin[7],1) #right rear
    time.sleep(2)

finally:
    # stop
    power = 0
    motor_run(power,pin[0],pin[1],-1) #left front
    motor_run(power,pin[2],pin[3],1) #right front
    motor_run(power,pin[4],pin[5],-1) #left rear
    motor_run(power,pin[6],pin[7],1) #right rear
```

The car can now go backward, turn left or right by changing the positive and negative values of power.

About the Steering

The movement of the Pico 4WD car is controlled by 4 motors. So you have two ways to make it steer.

Take turning right as an example

1. The left motors turn clockwise and the right motors turn counterclockwise.

```
power = 50

# turn right
motor_run(power, pin[0], pin[1], -1) #left front
motor_run(-power, pin[2], pin[3], 1) #right front
motor_run(power, pin[4], pin[5], -1) #left rear
motor_run(-power, pin[6], pin[7], 1) #right rear
```

2. The speed of the left motors is greater than the speed of the right motors.

```
power = 80

# also turn right
motor_run(power, pin[0], pin[1], -1) #left front
motor_run(power/2, pin[2], pin[3], 1) #right front
motor_run(power, pin[4], pin[5], -1) #left rear
motor_run(power/2, pin[6], pin[7], 1) #right rear
```

5. motor.py Module (Control Motor)

When Pico 4WD's various components work together, the code can be very long and difficult to understand.

So here we will learn how to encapsulate the motor code into a module (library), so that later we can import the library and call the functions inside.

The steps are as follows.

1. Now encapsulate the motor code from the previous project as `Motor()`.

```
from machine import Pin, PWM

def mapping(x, in_min, in_max, out_min, out_max):
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min

class Motor():
    def __init__(self, pin_a, pin_b, dir=1):
        self.pwm1 = PWM(Pin(pin_a, Pin.OUT))
        self.pwm2 = PWM(Pin(pin_b, Pin.OUT))
        self.pwm1.freq(20000)
        self.pwm2.freq(20000)
        self.dir = dir

    def run(self, power:int):
        if power == 0:
            self.pwm1.duty_u16(0xffff)
            self.pwm2.duty_u16(0xffff)
        else:
            value = mapping(abs(power), 0, 100, 20, 100) # power less_
            # than 20 is useless
            value = int(value / 100.0 * 0xffff)

            if power*self.dir > 0:
                self.pwm1.duty_u16(0xffff - value)
```

(continues on next page)

(continued from previous page)

```

        self.pwm2.duty_u16(0xffff)
    else:
        self.pwm1.duty_u16(0xffff)
        self.pwm2.duty_u16(0xffff - value)

```

2. To use this class, first declare four Motor objects.

```

left_front = Motor(17, 16, dir=-1)
right_front = Motor(15, 14, dir= 1)
left_rear = Motor(13, 12, dir=-1)
right_rear = Motor(11, 10, dir= 1)

```

3. Then use the run() function to get the individual motors to turn. Here the speed is set to a positive power (80), so the car will move forward.

```

power = 80
left_front.run(power)
right_front.run(power)
left_rear.run(power)
right_rear.run(power)

```

4. Then, the complete code is shown below.

```

from machine import Pin, PWM

def mapping(x, in_min, in_max, out_min, out_max):
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min

class Motor():
    def __init__(self, pin_a, pin_b, dir=1):
        self.pwm1 = PWM(Pin(pin_a, Pin.OUT))
        self.pwm2 = PWM(Pin(pin_b, Pin.OUT))
        self.pwm1.freq(20000)
        self.pwm2.freq(20000)
        self.dir = dir

    def run(self, power:int):
        if power == 0:
            self.pwm1.duty_u16(0xffff)
            self.pwm2.duty_u16(0xffff)
        else:
            value = mapping(abs(power), 0, 100, 20, 100)
            value = int(value / 100.0 * 0xffff)

            if power*self.dir > 0:
                self.pwm1.duty_u16(0xffff - value)
                self.pwm2.duty_u16(0xffff)
            else:
                self.pwm1.duty_u16(0xffff)
                self.pwm2.duty_u16(0xffff - value)

if __name__ == '__main__':
    # init

```

(continues on next page)

(continued from previous page)

```

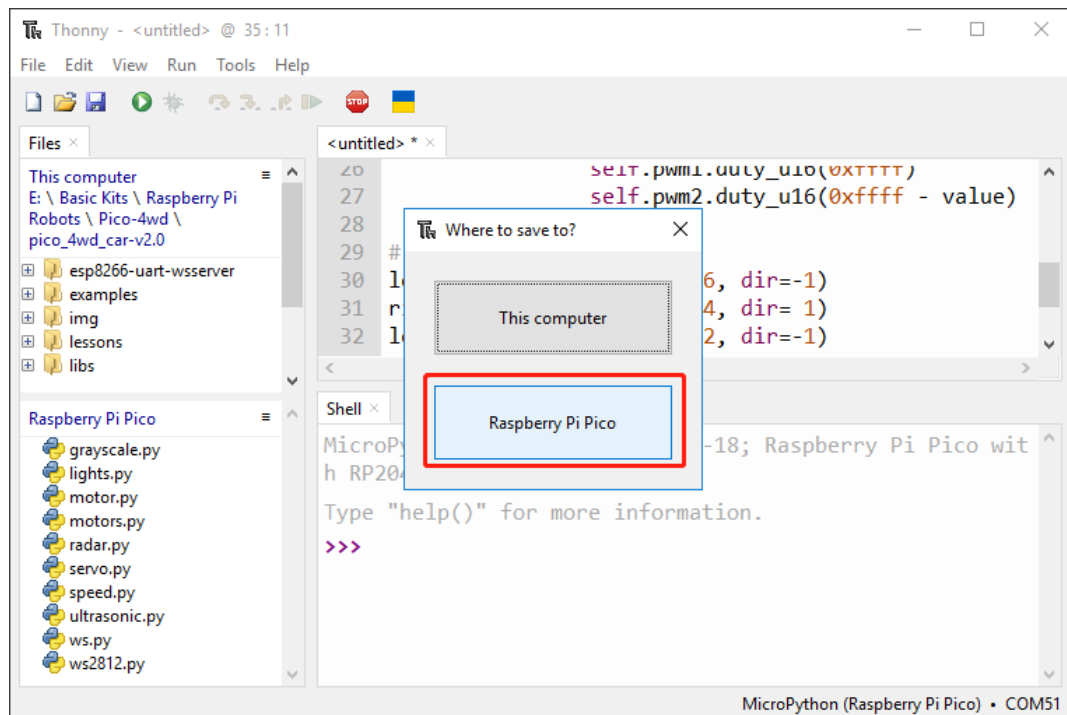
left_front  = Motor(17, 16, dir=-1)
right_front = Motor(15, 14, dir= 1)
left_rear   = Motor(13, 12, dir=-1)
right_rear  = Motor(11, 10, dir= 1)

try:
    # forward
    power = 80
    left_front.run(power)
    right_front.run(power)
    left_rear.run(power)
    right_rear.run(power)
    time.sleep(5)

finally:
    # stop
    power = 0
    left_front.run(power)
    right_front.run(power)
    left_rear.run(power)
    right_rear.run(power)
    time.sleep(0.2)

```

5. Now, create a new script on Thonny. Copy all the above code into this script. After pressing **Ctrl+S**, select **Raspberry Pi Pico** as the save path.

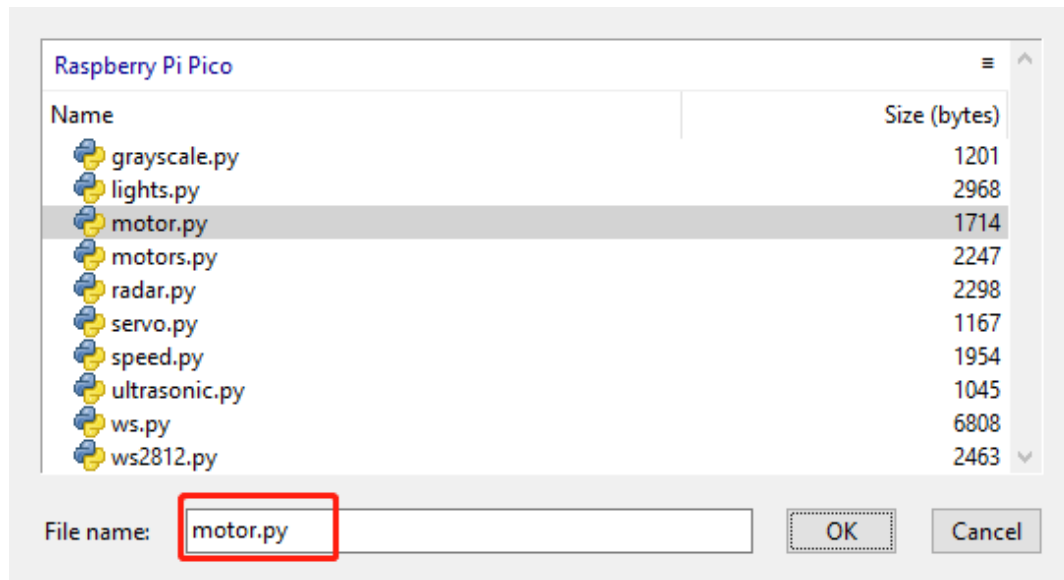



6. Fill in `motor.py` as the filename.

Note:

- You will notice that the Raspberry Pi Pico already has a file called `motor.py` in it.

- The Pico 4WD car already has the modules(libraries) pre-installed, so it can be played right out of the box.
- So here you can choose to overwrite to the original file.



7. To run the script, click the  button or press F5. When you power up the Pico 4WD car, you will see it move forward.

Warning: At the moment, this `motor.py` is not the final version. It needs a smooth speed effect, which is included in the [7. Smooth Speed Effect](#) project.

6. `motors.py` (Control Car)

Although we already have the `motor.py` library to simplify the code a bit, we need to write the speed for each motor separately when the car is moving.

```
left_front.run(power)
right_front.run(power)
left_rear.run(power)
right_rear.run(power)
```

This would make the code very long, so in this project we will create a `motors.py` module(library) to further optimize and simplify the code for the 4 motors.

For example, after optimization, you can make the car move forward with just the following command.

```
move(forward, 50)
```

The steps are as follows.

1. Now, create a new script on Thonny. Copy all the below code into this script. You will see here that `motor.py` is imported as a library.

```

from motor import Motor
import time

# init
left_front  = Motor(17, 16, dir=-1)
right_front = Motor(15, 14, dir= 1)
left_rear   = Motor(13, 12, dir=-1)
right_rear  = Motor(11, 10, dir= 1)
motors = [left_front, right_front, left_rear, right_rear]

# run all 4 motors
def set_motors_power(powers:list):
    ''' set motors power
        powers list, 1*4 list powers of each motor, the order is [left_
    ↪front, right_front, left_rear, right_rear]
    '''
    if len(powers) != 4:
        raise ValueError("powers should be a 1*4 list.")

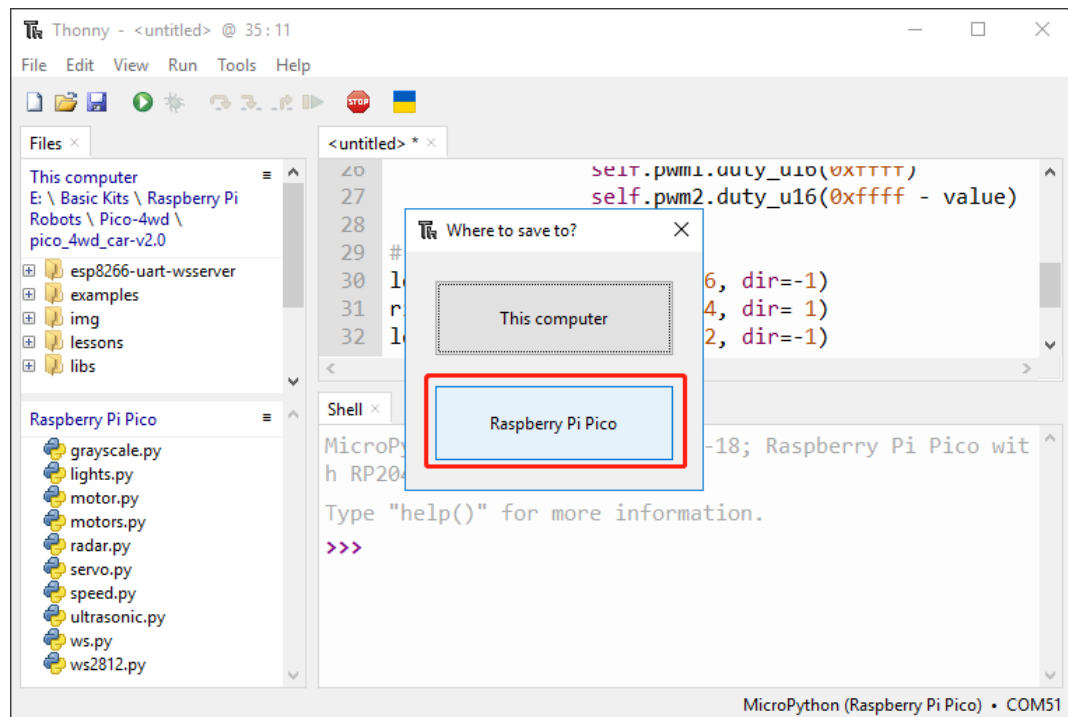
    for i, motor in enumerate(motors):
        motor.run(powers[i])

def move(action, power=0):
    if action == "forward":
        set_motors_power([power, power, power, power])
    elif action == "backward":
        set_motors_power([-power, -power, -power, -power])
    elif action == "left":
        set_motors_power([-power, power, -power, power])
    elif action == "right":
        set_motors_power([power, -power, power, -power])
    else:
        set_motors_power([0, 0, 0, 0])

# call the car move funtion
if __name__ == "__main__":
    speed = 50
    act_list = [
        "forward",
        "backward",
        "left",
        "right",
        "stop",
    ]
    for act in act_list:
        print(act)
        move(act, speed)
        time.sleep(1)

```

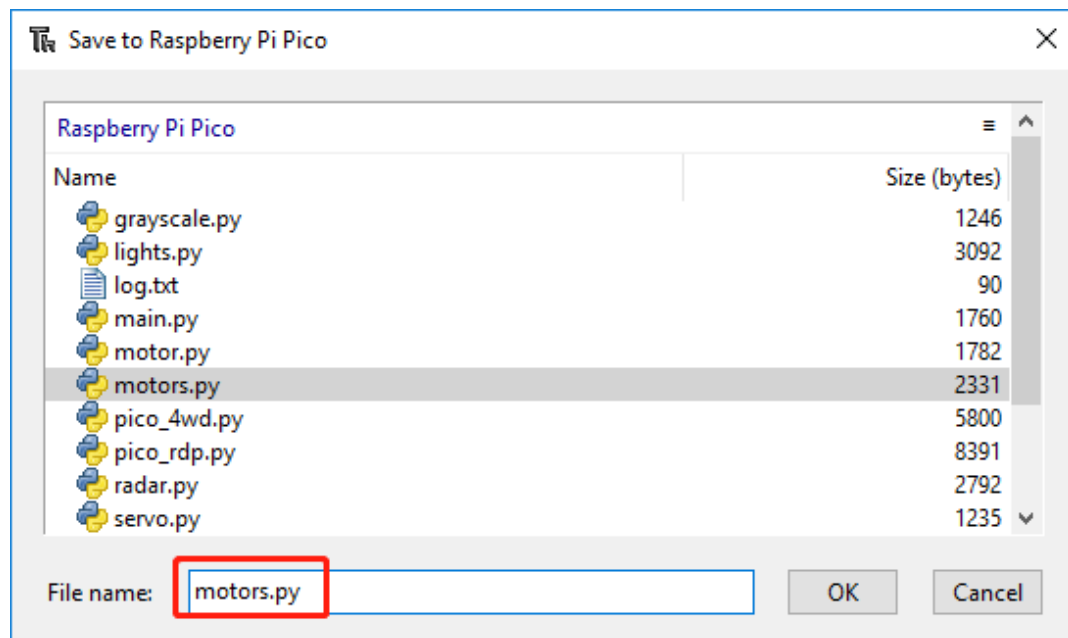
2. After pressing Ctrl+S, select **Raspberry Pi Pico** as the save path.




3. Fill in `motors.py` as the filename.

Note:

- You will notice that the Raspberry Pi Pico already has a file called `motors.py` in it.
- The Pico 4WD car already has the modules(libraries) pre-installed, so it can be played right out of the box.
- So here you can choose to overwrite to the original file.



4. To run the script, click the  button or press F5. When you power up the Pico 4WD car, you will see it move forward, backward, turn left, turn right and stop.

Warning: At the moment, this `motors.py` is not the final version. It needs a smooth speed effect, which is included in the [7. Smooth Speed Effect](#) project.

7. Smooth Speed Effect

The motor is a high power device and will generate high current during fast reversal, which may cause the Raspberry Pi Pico to not work.

Therefore, we need to add code for smooth speed in both `motor.py` and `motors.py`.

Note: You need to open `motor.py` and `motors.py` in Raspberry Pi Pico separately, add the following highlighted sections to them and save them.

- `motor.py`

```
from machine import Pin, PWM
import time

def mapping(x, in_min, in_max, out_min, out_max):
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min

class Motor():
    def __init__(self, pin_a, pin_b, dir=1):
        self.pwm1 = PWM(Pin(pin_a, Pin.OUT))
        self.pwm2 = PWM(Pin(pin_b, Pin.OUT))
        self.pwm1.freq(20000)
        self.pwm2.freq(20000)
        self.dir = dir
        self.current_power = 0

    def run(self, power:int):
        self.current_power = power
        if power == 0:
            self.pwm1.duty_u16(0xffff)
            self.pwm2.duty_u16(0xffff)
        else:
            value = mapping(abs(power), 0, 100, 20, 100)
            value = int(value / 100.0 * 0xffff)

            if power*self.dir > 0:
                self.pwm1.duty_u16(0xffff - value)
                self.pwm2.duty_u16(0xffff)
            else:
                self.pwm1.duty_u16(0xffff)
                self.pwm2.duty_u16(0xffff - value)

if __name__ == '__main__':
```

(continues on next page)

(continued from previous page)

```

# init
left_front  = Motor(17, 16, dir=-1)
right_front = Motor(15, 14, dir= 1)
left_rear   = Motor(13, 12, dir=-1)
right_rear  = Motor(11, 10, dir= 1)

try:
    # forward
    power = 80
    left_front.run(power)
    right_front.run(power)
    left_rear.run(power)
    right_rear.run(power)
    time.sleep(5)

finally:
    # stop
    power = 0
    left_front.run(power)
    right_front.run(power)
    left_rear.run(power)
    right_rear.run(power)
    time.sleep(0.2)

```

- motors.py

```

from motor import Motor
import time

left_front  = Motor(17, 16, dir=-1)
right_front = Motor(15, 14, dir= 1)
left_rear   = Motor(13, 12, dir=-1)
right_rear  = Motor(11, 10, dir= 1)
motors = [left_front, right_front, left_rear, right_rear]

def set_motors_power(powers:list):
    ''' set motors power
        powers list, 1*4 list powers of each motor, the order is [left_front, right_
→front, left_rear, right_rear]
    '''
    if len(powers) != 4:
        raise ValueError("powers should be a 1*4 list.")

    for i, motor in enumerate(motors):
        motor.run(powers[i])

def set_motors_power_gradually(powers:list):
    '''
        slowly increase power of the motor, to avoid high reverse voltage from motors
    '''
    if len(powers) != 4:
        raise ValueError("powers should be a 1*4 list.")

    flags = [True, True, True, True]
    while flags[0] or flags[1] or flags[2] or flags[3]:

```

(continues on next page)

(continued from previous page)

```

    for i, motor in enumerate(motors):
        if motor.current_power > powers[i]:
            motor.run(motor.current_power - 1)
        elif motor.current_power < powers[i]:
            motor.run(motor.current_power + 1)
        else:
            flags[i] = False
    time.sleep_ms(1)

def stop():
    set_motors_power([0, 0, 0, 0])

def move(action, power=0):
    if action == "forward":
        set_motors_power_gradually([power, power, power, power])
    elif action == "backward":
        set_motors_power_gradually([-power, -power, -power, -power])
    elif action == "left":
        set_motors_power_gradually([-power, power, -power, power])
    elif action == "right":
        set_motors_power_gradually([power, -power, power, -power])
    else:
        set_motors_power_gradually([0, 0, 0, 0])

if __name__ == "__main__":
    speed = 50
    act_list = [
        "forward",
        "backward",
        "left",
        "right",
        "stop",
    ]
    for act in act_list:
        print(act)
        move(act, speed)
        time.sleep(1)

```

After optimizing the code, you still only need to write `move(forward, 50)` to make the car move, but avoid the damage caused by the motor's extremely fast reverse rotation.

3.2.2 2. Ultrasonic Module and Servo

In this section, you will learn how the servo works, how to make it turn to different angles. Then learn how ultrasonic works and how to get the distance value. Then learn the use of sonar with the combination of servo and ultrasonic.

1. Introduce Servo

What is a Servo



- Brown Line: GND
- Orange Line: Signal pin, connect to the PWM pin of main board.
- Red wire: VCC

The servo is a type of motor that can rotate very precisely. In most cases, servo motors feature a control circuit that provides feedback about the current position of the motor shaft, allowing the motor to rotate very precisely. If you want to rotate an object at some specific angle or distance, then you use a servo.

Features

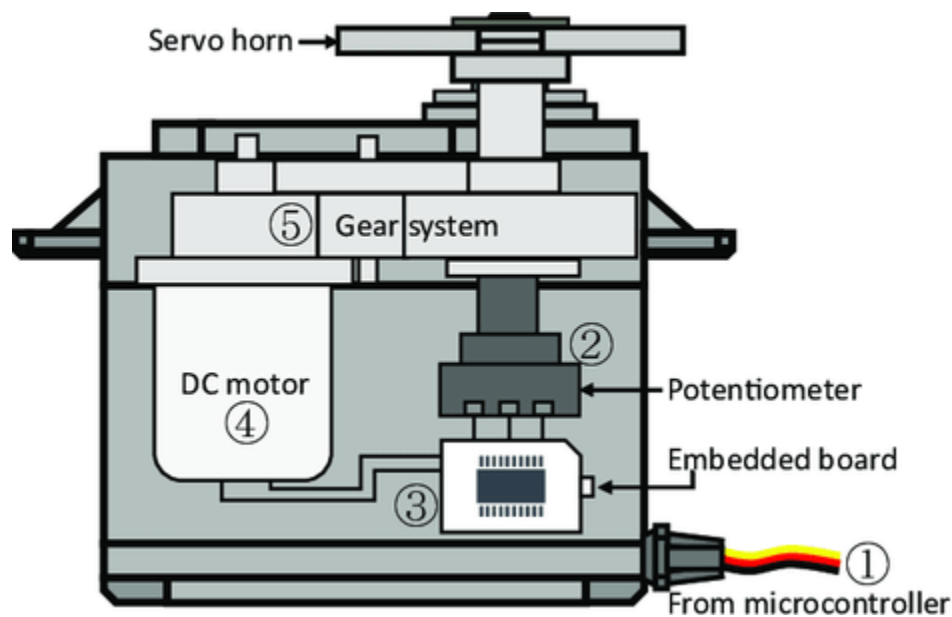
- Name: SG90
- Size: 22.4 x 12.5 x 23.8mm
- Weight: 10g \pm 5%
- Wire length: 25cm \pm 1cm
- Working voltage: 4.8v-6v
- Blocking torque: 1.3kg.cm-1.6kg.cm
- No-load speed: 0.09sev/60°
- No-load current: 90mA
- Output shaft: 20T

How it work?

A servo is generally composed of the following parts: case, shaft, gear system, potentiometer, DC motor, and embedded board.

It works like this:

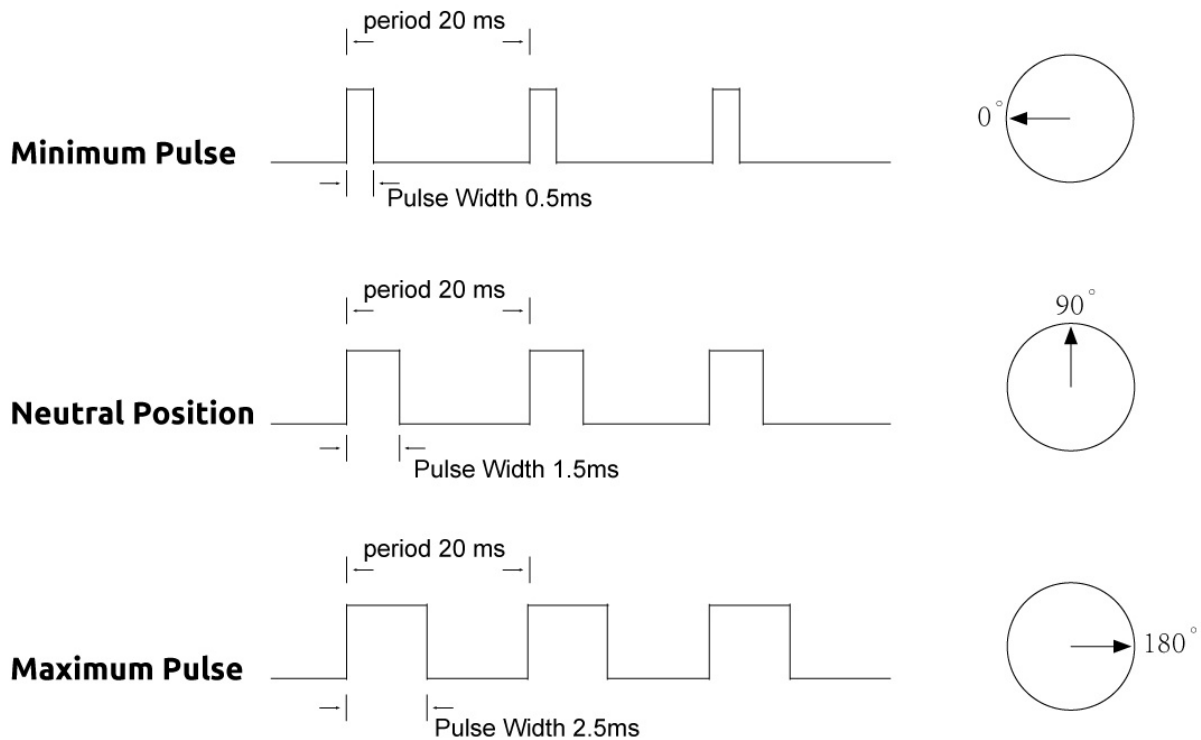
- The microcontroller sends out PWM signals to the servo, and then the embedded board in the servo receives the signals through the signal pin and controls the motor inside to turn.
- As a result, the motor drives the gear system and then motivates the shaft after deceleration.
- The shaft and potentiometer of the servo are connected together.
- When the shaft rotates, it drives the potentiometer, so the potentiometer outputs a voltage signal to the embedded board.
- Then the board determines the direction and speed of rotation based on the current position, so it can stop exactly at the right position as defined and hold there.



Work Pulse

The angle is determined by the duration of a pulse that is applied to the control wire. This is called Pulse width Modulation.

- The servo expects to see a pulse every 20 ms. The length of the pulse will determine how far the servo turns.
- For example, a 1.5ms pulse will make the servo turn to the 90 degree position (neutral position).
- When a pulse is sent to a servo that is less than 1.5 ms, the servo rotates to a position and holds its output shaft some number of degrees counterclockwise from the neutral point.
- When the pulse is wider than 1.5 ms the opposite occurs.
- The minimal width and the maximum width of pulse that will command the servo to turn to a valid position are functions of each servo.
- Generally the pulse will be about 0.5 ms ~ 2.5 ms wide.



2. Let Servo Rotate


In this chapter, we will show you how to make the servo work.

In a nutshell, to make the servo work you need to write 0.5ms to 2.5ms pulses to it every 20ms, for the principle see [1. Introduce Servo](#).

So we get the following code.

Code

```
import machine
import time
servo = machine.PWM(machine.Pin(18))
servo.freq(50)
def mapping(x, in_min, in_max, out_min, out_max):
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min
def set_angle(pin, angle):
    pulse_width=mapping(angle, -90, 90, 2.5,0.5)
    duty=int(mapping(pulse_width, 0, 20, 0,65535))
    pin.duty_u16(duty)
for angle in range(-90,90,5):
    set_angle(servo,angle)
    time.sleep(0.1)
```

You can copy the above code into Thonny or open the `sonar_2_servo_rotate.py` under the path of `pico_4wd_car-v2.0\examples\learn_modules`. Then click the  button or press F5 to run it.

When you power up the Pico 4WD car, you will see the servo rotate from 0° to 180°.

How it works?

Let's analyze this code.

- The servo needs to accept the signal in a 20ms cycle, which means setting a PWM with a frequency of 50Hz(1/20ms).

```
servo = machine.PWM(machine.Pin(18))
servo.freq(50)
```

- Use the mapping() function to map the servo angle range (-90 ~ 90) to pulse width range (0.5 ~ 2.5ms).

```
pulse_width=mapping(angle, -90, 90, 0.5, 2.5)
```

- Converts the pulse width from period to duty cycle. Since duty_u16() cannot be used with decimals (the value cannot be of floating point type), we use int() to force the duty to int type.

```
duty=int(mapping(pulse_width, 0, 20, 0, 65535))
pin.duty_u16(duty)
```

3. servo.py Module

Similarly, in order to make the code better readable when the servo is used with other components, the servo-related code is encapsulated into a library as follows.

Note: The encapsulated library servo.py has been saved in pico_4wd_car-v2.0\libs, which may differ from the ones shown in the course, so please refer to the file under libs path when using it.

```
from machine import Pin, PWM
import time

def mapping(x, in_min, in_max, out_min, out_max):
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min

class Servo():
    MAX_PW = 2500
    MIN_PW = 500
    PERIOD = 20000

    def __init__(self, pin):
        self.servo = PWM(Pin(pin, Pin.OUT))
        self.servo.freq(50)

    def set_angle(self, angle):
        try:
            angle = int(angle)
        except:
            raise ValueError("Angle value should be int value, not %s"%angle)
        if angle < -90: # most left
            angle = -90
        if angle > 90: #most right
            angle = 90

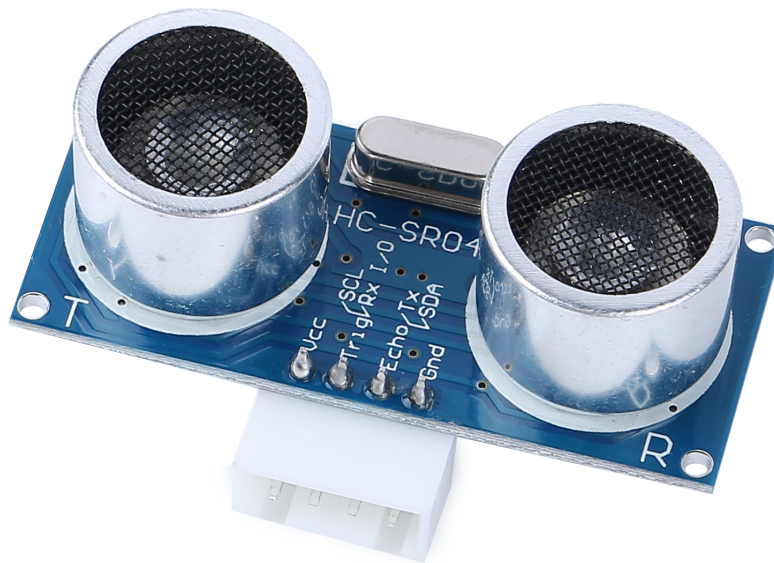
        pulse_width=mapping(angle, -90, 90, self.MAX_PW, self.MIN_PW)
        duty=int(mapping(pulse_width, 0, self.PERIOD, 0 ,0xffff))
        self.servo.duty_u16(duty)
```

(continues on next page)

(continued from previous page)

```
if __name__ == '__main__':
    servo = Servo(18)
    for i in range(0, -90, -1):
        servo.set_angle(i)
        time.sleep(0.01)
    for i in range(-90, 90, 1):
        servo.set_angle(i)
        time.sleep(0.01)
    for i in range(90, 0, -1):
        servo.set_angle(i)
        time.sleep(0.01)
```

4. Introduce Ultrasonic Module



- **TRIG**: Trigger Pulse Input
- **ECHO**: Echo Pulse Output
- **GND**: Ground
- **VCC**: 5V Supply

This is the HC-SR04 ultrasonic distance sensor, providing non-contact measurement from 2 cm to 400 cm with a range accuracy of up to 3 mm. Included on the module is an ultrasonic transmitter, a receiver and a control circuit.

You only need to connect 4 pins: VCC (power), Trig (trigger), Echo (receive) and GND (ground) to make it easy to use for your measurement projects.

Features

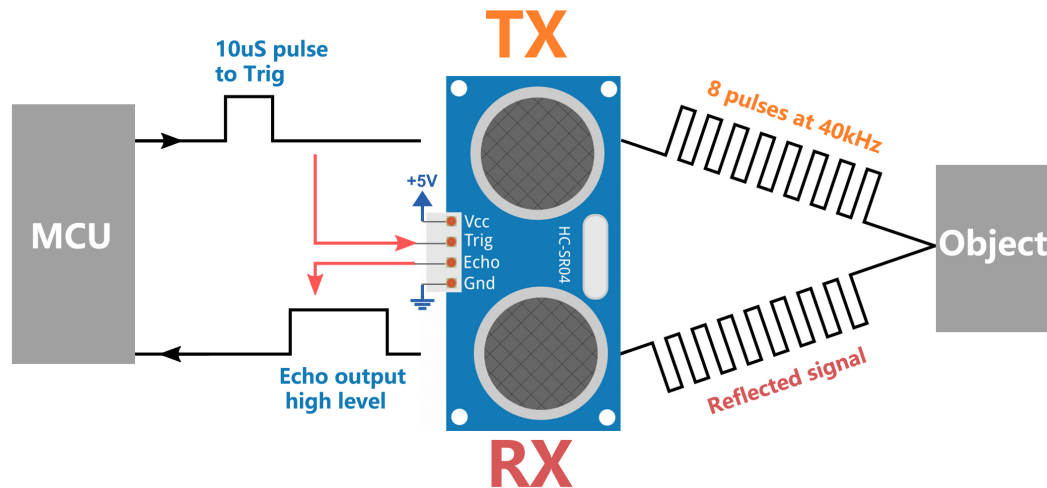
- Working Voltage: DC5V
- Working Current: 16mA
- Working Frequency: 40Hz
- Max Range: 500cm
- Min Range: 2cm

- Trigger Input Signal: 10uS TTL pulse
- Echo Output Signal: Input TTL level signal and the range in proportion
- Connector: XH2.54-4P
- Dimension: 46x20.5x15 mm

Principle

The basic principles are as follows:

- Using IO trigger for at least 10us high level signal.
- The module sends an 8 cycle burst of ultrasound at 40 kHz and detects whether a pulse signal is received.
- Echo will output a high level if a signal is returned; the duration of the high level is the time from emission to return.
- Distance = (high level time x velocity of sound (340M/S)) / 2



Formula:

- $\text{us} / 58 = \text{centimeters distance}$
- $\text{us} / 148 = \text{inch distance}$
- $\text{distance} = \text{high level time} \times \text{velocity (340M/S)} / 2$

Note:

- This module should not be connected under power up, if necessary, let the module's GND be connected first. Otherwise, it will affect the work of the module.
- The area of the object to be measured should be at least 0.5 square meters and as flat as possible. Otherwise, it will affect results.

5. Detecting Distance

In this project, we will show you how to make the ultrasonic module detect the distance of the obstacle ahead.

The transmitting probe of the ultrasonic module emits high-frequency sound waves (ultrasonic waves), which are reflected back and detected by the receiving probe when they touch an object.

By calculating the time from the emission to the reception of ultrasonic waves, multiplied by the speed of sound, the distance of the obstacle can be obtained.

Code

```
import machine
import time

trig = machine.Pin(6,machine.Pin.OUT)
echo = machine.Pin(7,machine.Pin.IN)


def distance():
    # pulse
    trig.high()
    time.sleep_us(10)
    trig.low()

    # get time
    pulse_width_us = machine.time_pulse_us(echo, machine.Pin.on)
    pulse_width_s= pulse_width_us/ 1000000.0

    # calculate the distance
    distance_m = pulse_width_s * 340 / 2
    distance_cm = (distance_m *100)

    return distance_cm

while True:
    dis = distance()
    print ('Distance: %.2f' % dis)
    time.sleep_ms(300)
```

You can copy the above code into Thonny or open the `sonar_5_get_distance.py` under the path of `pico_4wd_car-v2.0\examples\learn_modules`. Then click the  button or press F5 to run it.

After powering up the Pico 4WD, move the obstacle in front of it back and forth to verify the distance value (unit: cm).

6. ultrasonic.py Module

Similarly, in order to make the code better readable when the Ultrasonic module is used with other components, the related code is encapsulated into a library as follows.

Note: The encapsulated library `ultrasonic.py` has been saved in `pico_4wd_car-v2.0\libs`, which may differ from the ones shown in the course, so please refer to the file under `libs` path when using it.

```
from machine import Pin, time_pulse_us
import time
```

(continues on next page)

(continued from previous page)

```

class Ultrasonic():
    SOUND_SPEED = 340.0 # m/s
    MAX_DISTANCE = 300 # cm
    TIMEOUT = int((2*MAX_DISTANCE)/SOUND_SPEED/100* 1000000) # 17647 us

    def __init__(self, trig_Pin, echo_Pin):
        """Initialize Input(echo) and Output(trig) Pins."""
        self._trig = Pin(trig_Pin, Pin.OUT)
        self._echo = Pin(echo_Pin, Pin.IN)

    def _pulse(self):
        """Trigger ultrasonic module with 10us pulse."""
        self._trig.high()
        time.sleep_us(10)
        self._trig.low()

    def get_distance(self):
        """Measure pulse length and return calculated distance [cm]."""
        self._pulse()
        pulse_width = time_pulse_us(self._echo, Pin.on, self.TIMEOUT) / 1000000.0
        #pulse_width = time_pulse_us(self._echo, Pin.on) / 1000000.0
        distance = pulse_width * self.SOUND_SPEED / 2 * 100
        return distance

if __name__ == '__main__':
    # init
    sonar = Ultrasonic(6, 7)

    # get distance
    while True:
        distance = sonar.get_distance()
        print(distance)
        time.sleep(0.2)

```

7. Sonar Scanning

In the previous projects, we have learned the servo and ultrasonic modules separately and encapsulated their related scripts into libraries.

In this project, we'll write scripts to make the servo and ultrasonic work together to output the distances of obstacles ahead at once.

Code

Simply put, the servo rotates back and forth and the ultrasonic module detects distances at specific angles.

To get more comprehensive data, we need to have the detected distances in each direction output together.

```

from servo import Servo
from ultrasonic import Ultrasonic
import time

servo = Servo(18)
ultrasonic = Ultrasonic(6, 7)

```

(continues on next page)

(continued from previous page)

```

sonar_angle = 0 # current angle
sonar_step = 30 # Scan angle for each step

SONAR_MAX_ANGLE = 90
SONAR_MIN_ANGLE = -90

sonar_data = []
for i in range((SONAR_MAX_ANGLE-SONAR_MIN_ANGLE)/sonar_step+1):
    sonar_data.append(None)

def get_distance_at(angle):
    global sonar_angle
    sonar_angle = angle
    servo.set_angle(sonar_angle)
    #time.sleep(0.04)
    distance = ultrasonic.get_distance()
    if distance < 0:
        return -1
    else:
        return distance


def sonar_move():
    global sonar_angle, sonar_step
    if sonar_angle >= SONAR_MAX_ANGLE:
        sonar_angle = SONAR_MAX_ANGLE
        sonar_step = -abs(sonar_step)
    elif sonar_angle <= SONAR_MIN_ANGLE:
        sonar_angle = SONAR_MIN_ANGLE
        sonar_step = abs(sonar_step)
    sonar_angle += sonar_step

def mapping(x, in_min, in_max, out_min, out_max):
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min

def sonar_scan():
    global sonar_data
    sonar_move()
    distance = get_distance_at(sonar_angle)
    index= int(mapping(sonar_angle, SONAR_MIN_ANGLE, SONAR_MAX_ANGLE, 0, len(sonar_
    ↪data)-1))
    sonar_data[index]=distance
    return sonar_data

while True:
    print(sonar_scan())
    time.sleep(0.3)

```

You can copy the above code into Thonny or open the `sonar_7_sonar_scan.py` under the path of `pico_4wd_car-v2.0\examples\learn_modules`. Then click the  button or press F5 to run it.

When you power up the Pico 4WD car, you will see arrays like this `[1, 0, 0, 1, 1, 1, 1]` in the Shell.

How it works?

1. Import `servo.py` and `ultrasonic.py`, and instantiate them. In addition, set the scan angle to $-90 \sim 90$, and the angle interval to 30° .


```

from servo import Servo
from ultrasonic import Ultrasonic
import time

servo = Servo(18)
ultrasonic = Ultrasonic(6, 7)

sonar_angle = 0 # current angle
sonar_step = 30 # Scan angle for each step

SONAR_MAX_ANGLE = 90
SONAR_MIN_ANGLE = -90

```

2. Ultrasonic module detects every 30° as the servo rotates and returns distance.

```

def get_distance_at(angle):
    global sonar_angle
    sonar_angle = angle
    servo.set_angle(sonar_angle)
    #time.sleep(0.04)
    distance = ultrasonic.get_distance()
    if distance < 0:
        return -1
    else:
        return distance

```

3. The sonar_move() here actually makes the servo rotate in the set angle range (-90 ~ 90) at 30° intervals.

```

def sonar_move():
    global sonar_angle, sonar_step
    if sonar_angle >= SONAR_MAX_ANGLE:
        sonar_angle = SONAR_MAX_ANGLE
        sonar_step = -abs(sonar_step)
    elif sonar_angle <= SONAR_MIN_ANGLE:
        sonar_angle = SONAR_MIN_ANGLE
        sonar_step = abs(sonar_step)
    sonar_angle += sonar_step

```

4. Now to combine the above two functions, let the ultrasonic detect the distance while the servo is moving and output the 7 distance values as an array at once.

```

def sonar_scan():
    global sonar_data
    sonar_move()
    distance = get_distance_at(sonar_angle)
    index= int(mapping(sonar_angle, SONAR_MIN_ANGLE, SONAR_MAX_ANGLE, 0,
↪len(sonar_data)-1))
    sonar_data[index]=distance
    return sonar_data

```

8. sonar.py Module

In the previous project, we learned how the sonar scanner detects obstacles. That is the servo is turned from left to right and the ultrasonic module detects every specific angle, then several sets of distance values are output at once.

However, this kind of data is not very convenient for calculation, so in practice, we can set a threshold value, and then compare the detected distance against this threshold value, and then output a 0 or 1.

In this way, we encapsulate the sonar scanning code into a library and then add more calculations to it, so that it can be easily imported for use in the project.

Note: The final encapsulated library `sonar.py` has been saved in `pico_4wd_car-v2.0\libs`, which may differ from the ones shown in the course, so please refer to the file under `libs` path when using it.

You can learn how `sonar.py` is encapsulated in the following steps.

1. Determine if there are obstacles

- Most of the time, the car only needs to know whether there are obstacles in all directions.
- So here set a distance threshold with the variable `SONAR_REFERENCE`.
- And then create the function `get_sonar_status()` to determine if the distance is greater than the threshold, then output 1, otherwise output 0.
- Then output the processed data at once, and you get array like this `[1, 0, 0, 1, 1, 1, 1]`.

```
from servo import Servo
from ultrasonic import Ultrasonic
import time

servo = Servo(18)
ultrasonic = Ultrasonic(6, 7)

sonar_angle = 0
sonar_step = 30

SONAR_MAX_ANGLE = 90
SONAR_MIN_ANGLE = -90
SONAR_REFERENCE = 20

sonar_data = []
for i in range((SONAR_MAX_ANGLE-SONAR_MIN_ANGLE)/sonar_step+1):
    sonar_data.append(None)

def get_distance_at(angle):
    global sonar_angle
    sonar_angle = angle
    servo.set_angle(sonar_angle)
    #time.sleep(0.04)
    distance = ultrasonic.get_distance()
    if distance < 0:
        return -1
    else:
        return distance

def sonar_move():
    global sonar_angle, sonar_step
```

(continues on next page)

(continued from previous page)

```

    if sonar_angle >= SONAR_MAX_ANGLE:
        sonar_angle = SONAR_MAX_ANGLE
        sonar_step = -abs(sonar_step)
    elif sonar_angle <= SONAR_MIN_ANGLE:
        sonar_angle = SONAR_MIN_ANGLE
        sonar_step = abs(sonar_step)
    sonar_angle += sonar_step

def get_sonar_status(distance):
    if distance > SONAR_REFERENCE or distance < 0:
        return 1
    else:
        return 0

def mapping(x, in_min, in_max, out_min, out_max):
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min

def sonar_scan():
    global sonar_data
    sonar_move()
    distance = get_distance_at(sonar_angle)
    index= int(mapping(sonar_angle, SONAR_MIN_ANGLE, SONAR_MAX_ANGLE, 0,
↪len(sonar_data)-1))
    status=get_sonar_status(distance)
    sonar_data[index]=status
    return sonar_data

while True:
    print(sonar_scan())
    time.sleep(0.1)

```

2. Get complete data before judging

Additionally, if we use `sonar_data` directly for obstacle determination, the data on the left becomes an interference item when the left side obstacle disappears and the sonar scans the right side.

It makes more sense to determine an obstacle after a sonar cycle has been scanned and complete data has been collected.

```

...
...

def get_distance_at(angle):
    ...

def sonar_move():
    ...

def get_sonar_status(distance):
    ...

def mapping(x, in_min, in_max, out_min, out_max):
    ...

def sonar_scan():
    global sonar_data
    sonar_move()
    distance = get_distance_at(sonar_angle)
    index=int(mapping(sonar_angle, SONAR_MIN_ANGLE, SONAR_MAX_ANGLE, 0,
↪len(sonar_data)-1))

```

(continues on next page)

(continued from previous page)

```

status=get_sonar_status(distance)
sonar_data[index]=status
if (index == 0 or index == len(sonar_data)-1) and None not in sonar_data:
    return sonar_angle,distance,sonar_data
else:
    return sonar_angle,distance,status

while True:
    _,_,result = sonar_scan()
    if type(result) is not int:
        print(result)
    time.sleep(0.1)

```

3. Further optimization

In order to be compatible with more complex programs, we created two more functions to modify the rotation rules and distance determination of the sonar.

```

...
...

def get_distance_at(angle):
    ...

def sonar_move():
    ...

def get_sonar_status(distance):
    ...

def mapping(x, in_min, in_max, out_min, out_max):
    ...

def sonar_scan():
    ...

def set_sonar_scan_config(scan_range=None,step=None):
    global SONAR_MAX_ANGLE, SONAR_MIN_ANGLE, sonar_angle, sonar_step, sonar_
↪data

    # update changed
    item = 0
    if scan_range is None or scan_range is SONAR_MAX_ANGLE-SONAR_MIN_ANGLE:
        item+=1
    else:
        SONAR_MAX_ANGLE = int(scan_range / 2)
        SONAR_MIN_ANGLE = SONAR_MAX_ANGLE-scan_range
    if step is None or abs(sonar_step) is abs(step):
        item+=1
    else:
        sonar_step=int(step)
    if item is 2: # if nothing change, return
        return

    # re-create the data list
    sonar_data = []
    for i in range(scan_range/abs(sonar_step) +1):
        sonar_data.append(None)

    sonar_angle=0

```

(continues on next page)

(continued from previous page)

```

servo.set_angle(sonar_angle)

def set_SONAR_REFERENCE(ref):
    global SONAR_REFERENCE
    SONAR_REFERENCE = int(ref)

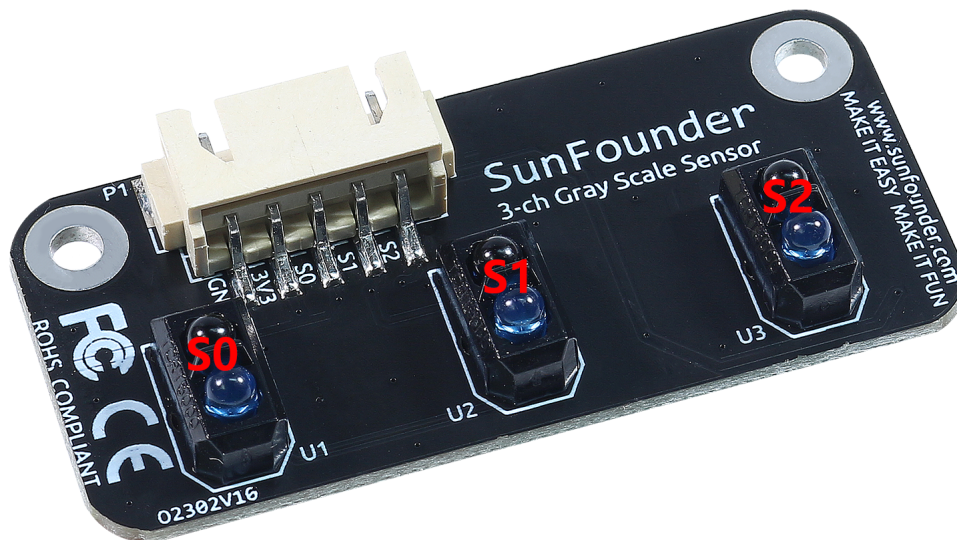
if __name__ == '__main__':
    try:
        set_sonar_scan_config(180,30)
        set_SONAR_REFERENCE(20)
        while True:
            _,_,status = sonar_scan()
            if type(status) is not int:
                print(status)
            time.sleep(0.1)
    finally:
        servo.set_angle(0)

```

3.2.3 3. Grayscale Module

In this section, you will learn how grayscale modules work, judging the environment by grayscale and making a module (library).

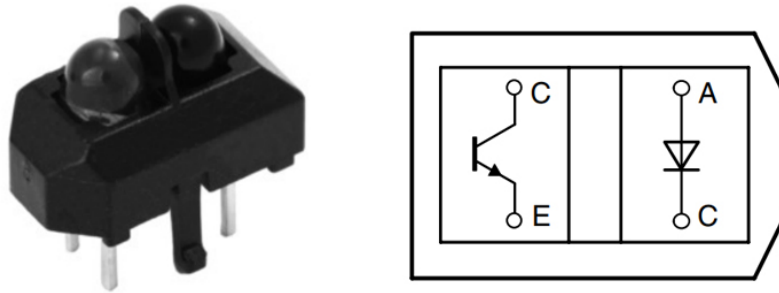
1. Introduce Grayscale Module



- **GND:** Ground Input
- **3V3:** 3.3 to 5V DC Supply Input
- **S0/S1/S2:** The output values of the three TCRT5000 transmitting sensors, the outputs are analog values.

This is a grayscale sensor module consisting of 3 transmitting sensors, which can be used for line following and edge detection.

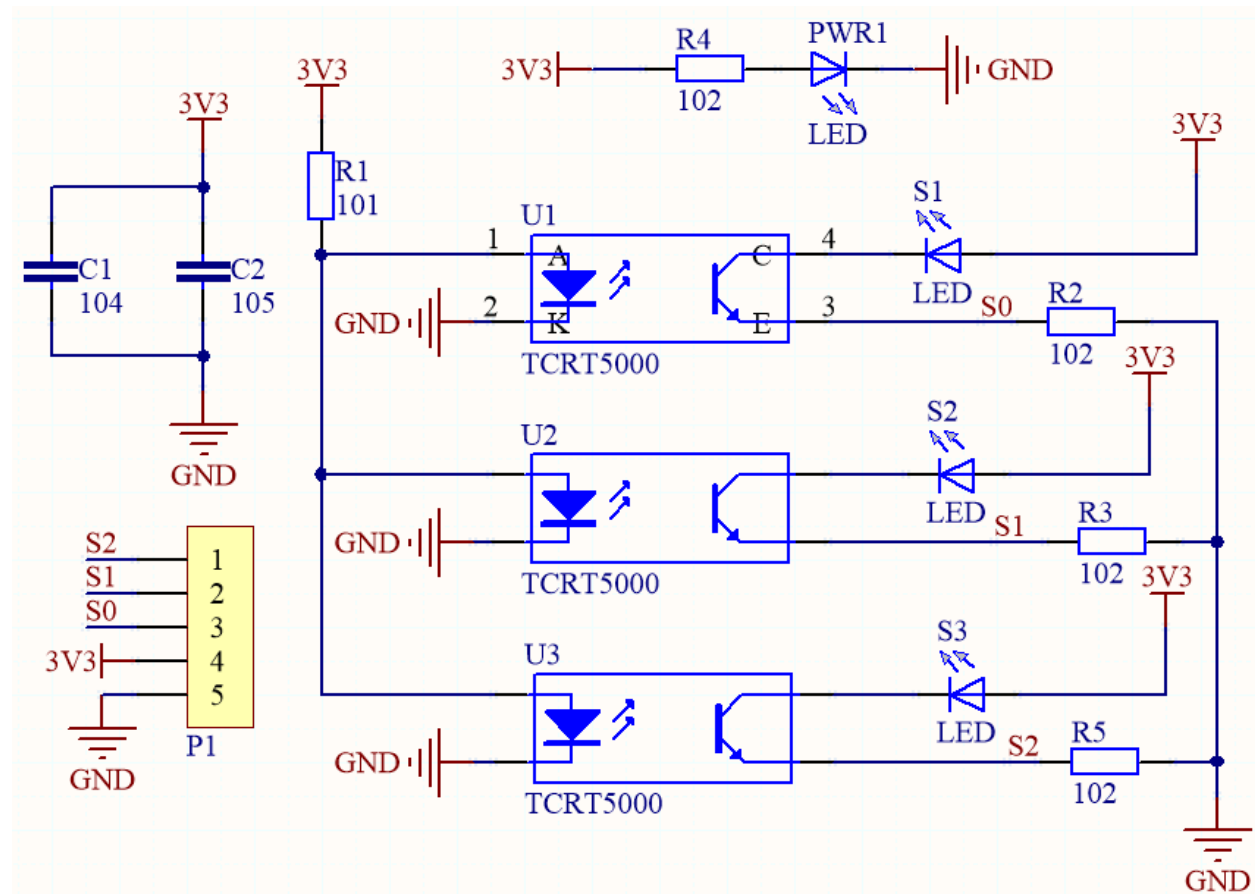
The transmitter sensor consists of an infrared light-emitting diode and a phototransistor covered with a lead material to block visible light.



When working, the IR LED of TCRT5000 continuously emits infrared light (invisible light) with a wavelength of 950nm. When the emitted infrared light is not reflected back by the obstacle or the reflection intensity is insufficient, the phototransistor does not work. When the infrared light is reflected with sufficient intensity and received by the phototransistor at the same time, the phototransistor is in working condition and provides output.

Schematic Diagram

Here is the schematic of the module



- When working, the IR LED of TCRT5000 continuously emits infrared light (invisible light) with a wavelength of 950nm.
- When the infrared light is reflected, the phototransistor is in working condition and output analog value.
- In general, white surface return value > black surface's > cliff's.

- Also the brighter the surface, the more reflections, resulting in an increase in current, the brighter the LED.

Features

- 3 Channels
- Operating Voltage: 3.3 ~ 5V
- Output: analog value
- Sensor Type TCRT5000
- Connector Model XH2.54-WS-5P
- Operating Temperature: -10 °C to +50 °C
- Dimensions: 50mm x 25mm

2. Grayscale Detection

This project will teach us how to read the grayscale module values and determine whether the Pico 4WD detects a line or cliff based on those values.

With the previous section *1. Introduce Grayscale Module*, you can know that the value on the **white surface** > **black line** > **cliff**.

Code

You will see that if you hang the Pico 4WD car in the air, the Shell will print “Danger!”. If the grayscale module detects a black line, True will be printed. On white surfaces, False will be printed.

```
from machine import Pin, ADC
import time

gs0 = ADC(Pin(26))
gs1 = ADC(Pin(27))
gs2 = ADC(Pin(28))
edge_ref = 1000    # edge_reference
line_ref = 10000   # line_reference

def get_value():
    return [gs0.read_u16(), gs1.read_u16(), gs2.read_u16()]

def is_on_edge():
    gs_list = get_value()
    for value in gs_list:
        if value <= edge_ref:
            return True
    return False

def get_line_status():
    gs_list = get_value()
    line_status = []
    for value in gs_list:
        line_status.append(value < line_ref)
    return line_status

while True:
    print(get_value())
    time.sleep(0.2)
    if is_on_edge():
```

(continues on next page)


(continued from previous page)

```
print("Danger!")
else:
    print(get_line_status())
```

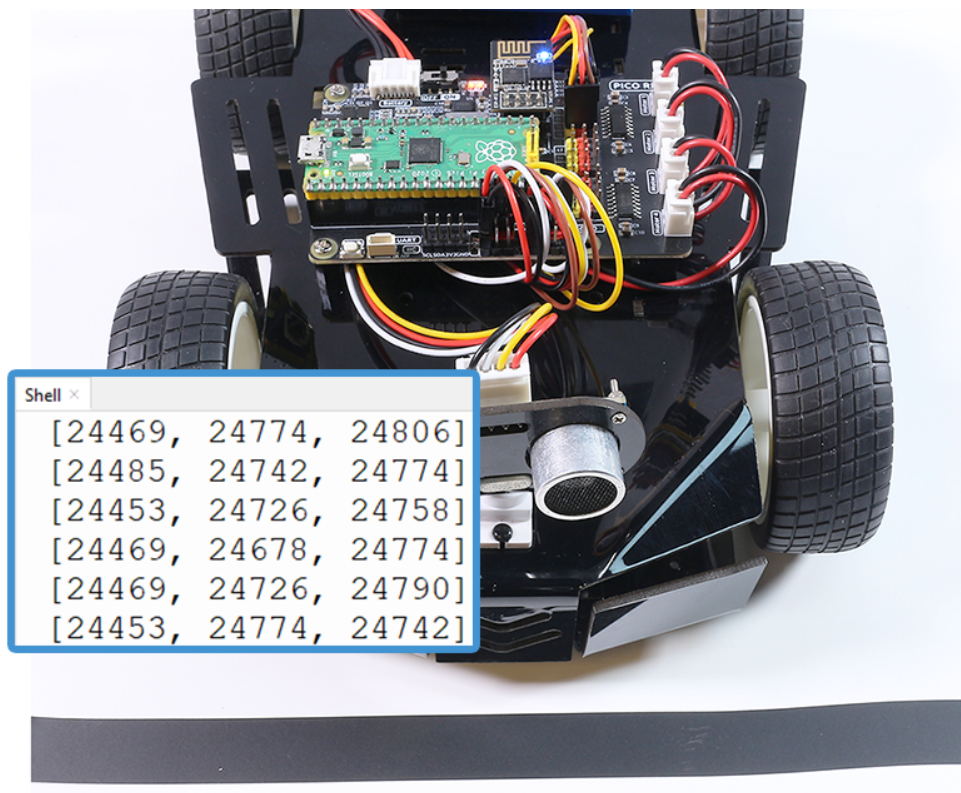
What's More

Note that `edge_ref` and `line_ref` are the cliff and line threshold set according to my current environment, what you actually use may be different.

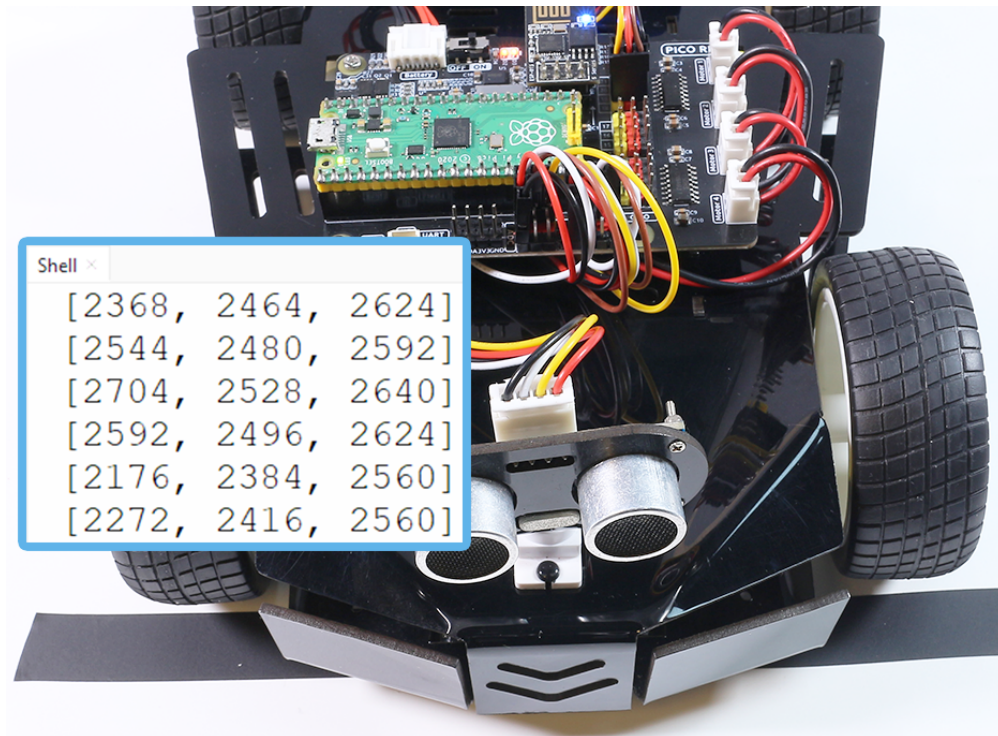
You need to get them by following these steps.

1. Copy the above code into Thonny or open the `grayscale_2_get_value.py` under the path of `pico_4wd_car-v2.0\examples\learn_modules`.
2. To run the script, click the  button or press F5.
3. After powering up the Pico 4WD car, place the grayscale module in three environments: white, black and hanging in the air (10cm or more) to see how the data in the Shell changes (keeping the USB cable connected to the computer).

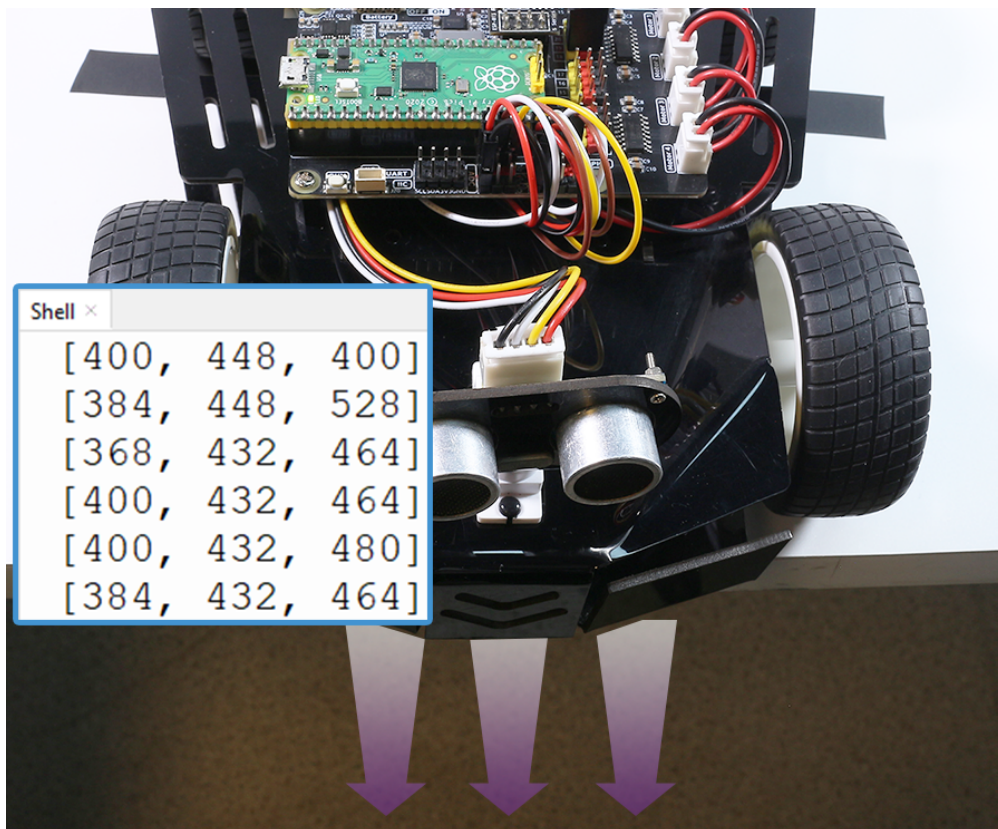
White surface You will find that the value of the white surface is generally large, for example mine is around 240,000.



Black line The value on the black line will be smaller, and now I'm at about 2000.



Overhang (10cm or more) And the value of the overhang will be even smaller, already less than 1000 in my environment.



4. Taking my detection environment as an example.

- My car reads around 24000 in the white area and around 2000 in the black line, so I set `line_ref` to about the middle value of 10000.
- In the cliff area it reads less than 1000, so I set `edge_ref` to 1000.

3. grayscale.py Module

Similarly, in order to keep the code simple and organized when using the grayscale module with other modules, we have encapsulated all its related code into a library.

When you use it, you just need to import this library, and you can call the functions inside directly.

For example, the following two functions can be used to know if a cliff or black line is detected.

```
is_on_edge()  
get_line_status()
```

Note: The final encapsulated library `grayscale.py` has been saved in `pico_4wd_car-v2.0\libs`, which may differ from the ones shown in the course, so please refer to the file under `libs` path when using it.

1. Create Grayscale class

Encapsulate the code related to the grayscale module into a library, as follows.

```
from machine import Pin, ADC  
  
class Grayscale():  
  
    def __init__(self, pin0=26, pin1=27, pin2=28):  
        self.gs0 = ADC(Pin(pin0))  
        self.gs1 = ADC(Pin(pin1))  
        self.gs2 = ADC(Pin(pin2))  
        self.edge_ref = 1000    # edge_reference  
        self.line_ref = 10000  # line_reference  
  
    def get_value(self):  
        return [self.gs0.read_u16(), self.gs1.read_u16(), self.gs2.read_  
↪u16()]  
  
    def is_on_edge(self):  
        gs_list = self.get_value()  
        return gs_list[2] <= self.edge_ref or gs_list[1] <= self.edge_ref or_  
↪gs_list[0] <= self.edge_ref  
  
    def get_line_status(self):  
        gs_list = self.get_value()  
        return [int(value < self.line_ref) for value in gs_list]  
  
# init  
gs = Grayscale(26, 27, 28)  
  
# detect  
while True:  
    if gs.is_on_edge():  
        print("Danger!")  
    else:  
        print(gs.get_line_status())
```

2. Add threshold changeable function

We have written cliff and black line thresholds in this module(library), but the thresholds are different for different environments when using it. So 2 other functions were created to make it easy for you to modify the thresholds in the main program.

```
from machine import Pin, ADC

class Grayscale():

    def __init__(self, pin0=26, pin1=27, pin2=28):
        self.gs0 = ADC(Pin(pin0))
        self.gs1 = ADC(Pin(pin1))
        self.gs2 = ADC(Pin(pin2))
        self.edge_ref = 1000 # edge_reference
        self.line_ref = 10000 # line_reference

    def get_value(self):
        return [self.gs0.read_u16(), self.gs1.read_u16(), self.gs2.read_u16()]

    def is_on_edge(self):
        gs_list = self.get_value()
        return gs_list[2] <= self.edge_ref or gs_list[1] <= self.edge_ref or gs_list[0] <= self.edge_ref

    def get_line_status(self):
        gs_list = self.get_value()
        return [int(value < self.line_ref) for value in gs_list]

    def set_edge_reference(self, value):
        self.edge_ref = value

    def set_line_reference(self, value):
        self.line_ref = value

if __name__ == '__main__':
    import time

    # init
    gs = Grayscale(26, 27, 28)

    # config
    gs.set_edge_reference(800)
    gs.set_line_reference(12000)

    # detect
    while True:
        if gs.is_on_edge():
            print("Danger!")
        else:
            print(gs.get_line_status())
            time.sleep(0.2)
```

3.2.4 4. Speed Module

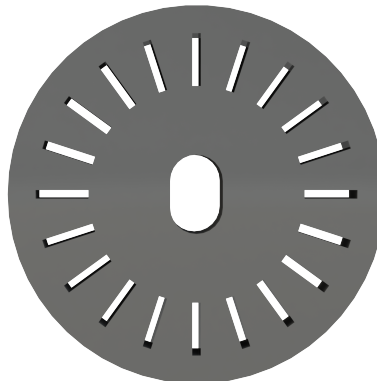
1. Introduce Speed Module



- **GND:** Ground Input
- **3V3:** 3.3 to 5V DC Supply Input
- **1Y and 2Y:** The output values of the 2 H2010 photoelectric sensors. Output 0 when blocked, otherwise output 1.

This is a module composed of 2 H2010 photoelectric sensors that can be used with an encoder disk to calculate the motor's rotation speed.

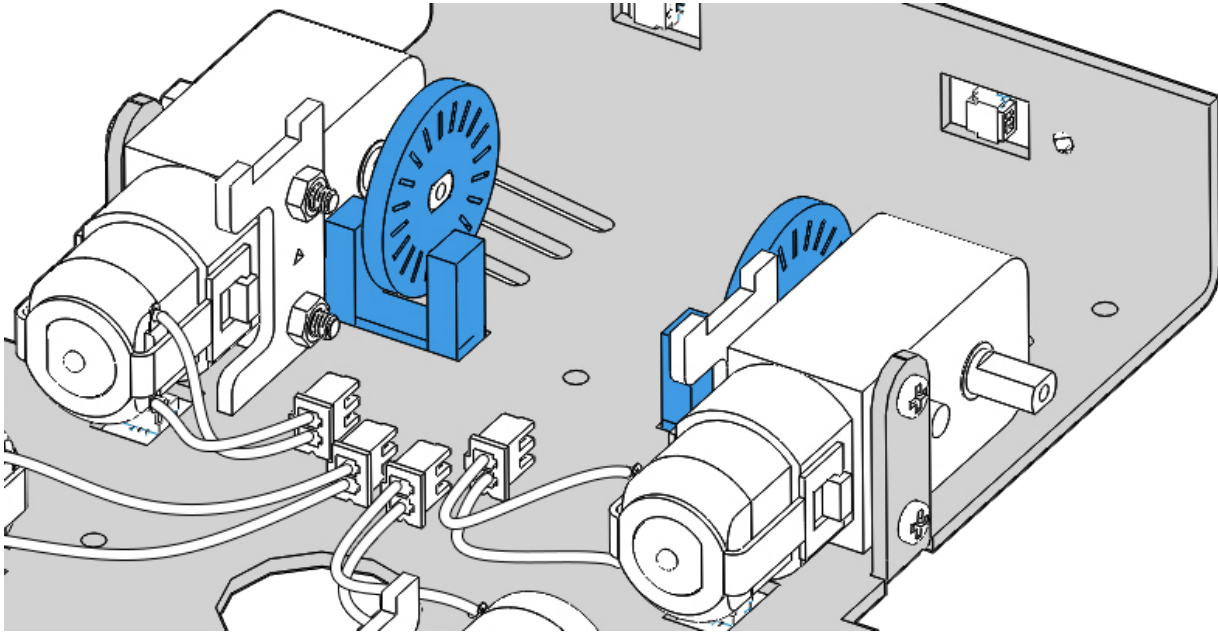
The encoder disk is a black plastic disk with 20 evenly spaced slots in the middle.



The use of all three of them is shown in the figure below. A disk encoder is inserted into a motor shaft, and the motor rotates with it as it passes across the middle of the speed sensor.

When the infrared ray emitted by the light-emitting diode of the speed module is blocked, a low level will be output; when the emitted infrared ray passes through the slot of the encoder disk, a high level will be output.

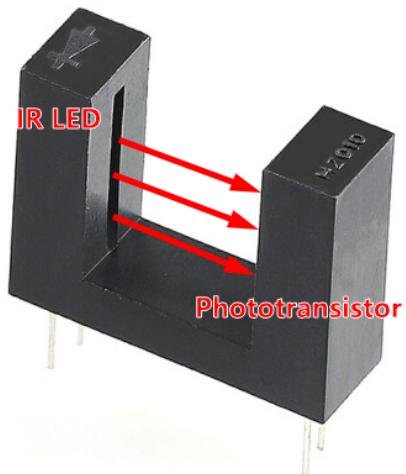
The motor generates 20 high levels in one revolution. By counting the number of high levels obtained per minute and dividing by 20, the speed of the motor is obtained.



Working Principle

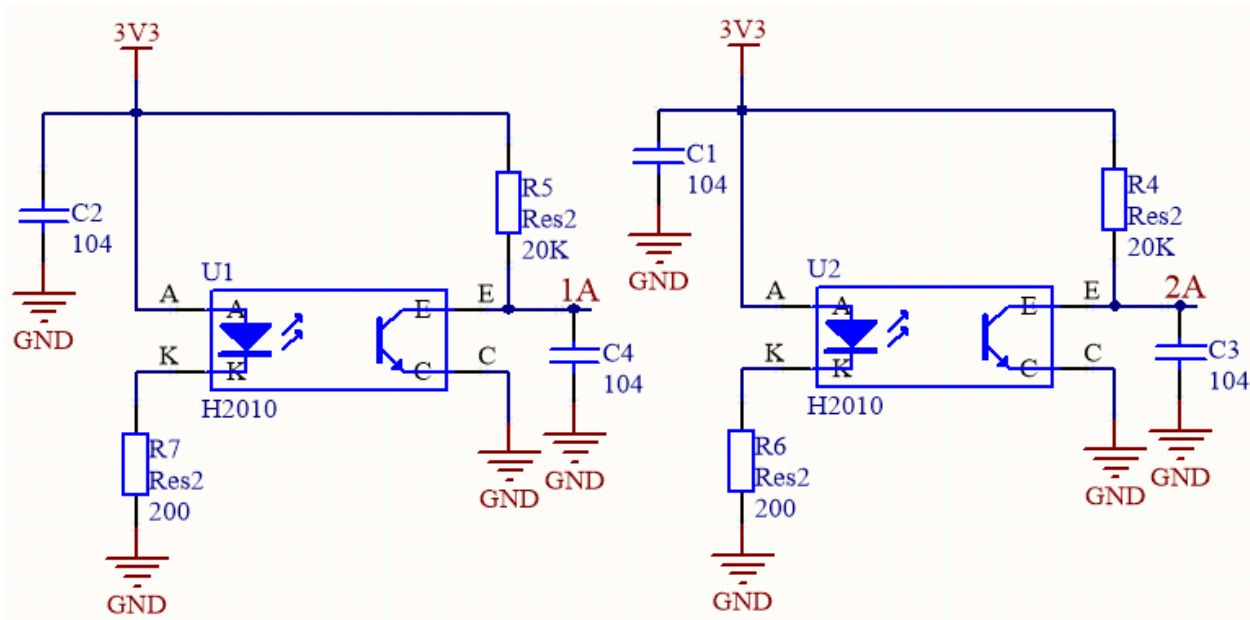
H2010

Speed module has two H2010 photoelectric sensors, each made up of a phototransistor and an infrared light emitter encased in a 10cm-wide black plastic case.



When operating, the infrared light-emitting diode continuously emits infrared light (invisible light), and the photosensitive triode will conduct if it receives it.

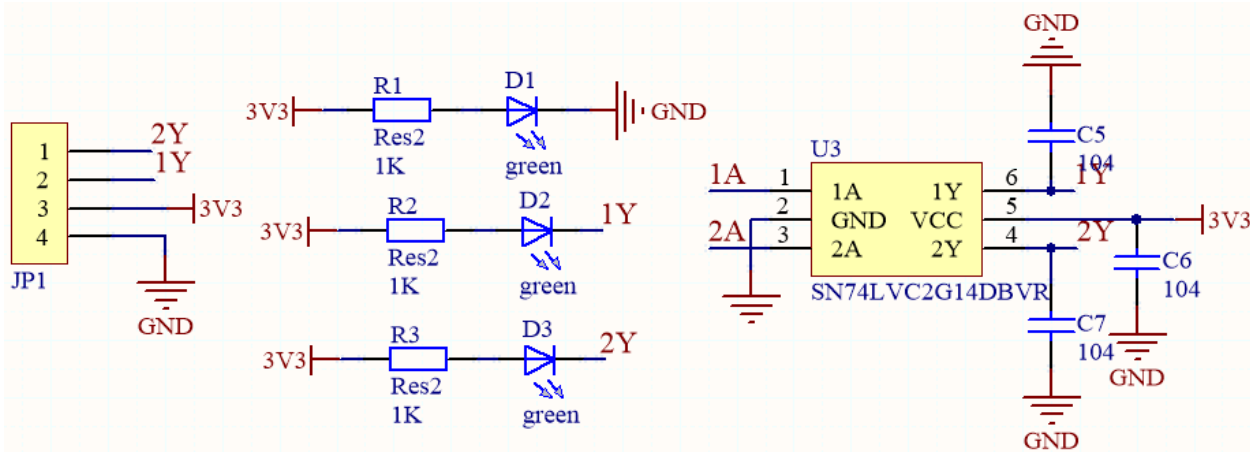
When it is connected to the circuit, the value of 1A and 2A is low when the phototransistor is conducting. If there is an object blocking the slot, 1A and 2A are high.



sn74lvc2g14dbvr

Then connect 1A and 2A to the inputs of the SN74LVC2G14 (). The SN74LVC2G14 is a dual schmitt trigger inverter, which is used here to boost and invert the signal and then output it.

The output values 1Y and 2Y will be output directly and determine the two indicator light states.



- **Summary:** When the H2010 photoelectric sensor on the speed module is blocked, the corresponding channel will output a low level and the indicator will light up. And vice versa.

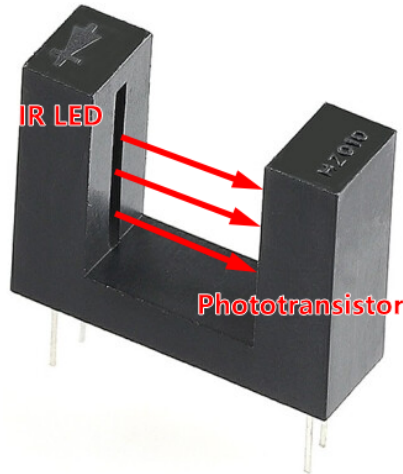
Features

- Operating Voltage: 3.3V ~ 5V DC
- Output format: Digital switching output (Output 0 when blocked, otherwise output 1)
- LED lights when it when blocked.
- PCB Size: 64x24x23mm
- Operating Current: 15mA
- 10mm wide slot with 12mm height detects objects via infrared light between its slot

- Fast output update since its optical

2. Speed Calculation

As described in *1. Introduce Speed Module*, the speed module's speed measurement principle is to count the number of times the infrared light pass through the encode disk per unit time.



Note:

- The complete script `speed_2_get_speed.py` is in the path `pico_4wd_car\examples\learn_modules`.
- Before running the following scripts, it is recommended to hang the car in the air so that the 4 wheels can turn freely.

Here are the steps to implement the speed calculation, which you can copy into Thonny to run.

1. Counting

Let's count how many times the infrared rays pass through the encode disk in a minute.

```
from machine import Timer, Pin

def on_left(ch):
    global left_count
    left_count += 1

def on_timer(ch):
    global left_count
    print(left_count)
    left_count = 0

# Interrupter, used to count
left_count = 0
left_pin = Pin(8, Pin.IN, Pin.PULL_UP)
left_pin.irq(trigger=Pin.IRQ_FALLING, handler=on_left)

# Timer, print count every 1000ms
tim = Timer()
```

(continues on next page)

(continued from previous page)


```

tim.init(period=1000, mode=Timer.PERIODIC, callback=on_timer)

# motor run
import motors
import time

try:
    while True:
        # for i in range(20,100,10):
            motors.move("forward",50)
        # time.sleep(1)
finally:
    motors.stop()
    time.sleep(0.2)

```

Copy the above code into Thonny and click the  button or press F5 to run it.

After powering up the Pico 4WD car, you will see the 4 motors turning (forward) while seeing the count in the Shell.

2. Calculate the RPM

Now let's calculate the RPM based on the counts. To improve the accuracy of the calculation, the calculation period is set to 200ms.

```

from machine import Timer, Pin
import math

duration = 200

def on_left(ch):
    global left_count
    left_count += 1

def on_timer(ch):
    global left_count
    # revolutions per second
    times = left_count * 1000 / duration
    rps=times/20.0
    print(rps)

    # clear count
    left_count = 0

# Interrupter, used to count
left_count = 0
left_pin = Pin(8, Pin.IN, Pin.PULL_UP)
left_pin.irq(trigger=Pin.IRQ_FALLING, handler=on_left)

# Timer, print speed
tim = Timer()
tim.init(period=duration, mode=Timer.PERIODIC, callback=on_timer)

# motor run
import motors
import time

```

(continues on next page)

(continued from previous page)

```

try:
    while True:
        for i in range(20,100,10):
            motors.move("forward",i)
            time.sleep(1)
finally:
    motors.stop()
    time.sleep(0.2)

```

After running the script, click **View -> Plotter** to view the RPM curve, and you can see that the higher the power, the faster the RPM.

3. Calculate moving speed

Next, the RPM is converted to moving speed (unit:cm/s). Here the moving speed is actually the RPM multiplied by the circumference of the wheel.

```

from machine import Timer, Pin
import math

WP = 2 * math.pi * 3.3 # wheel_perimeter(cm): 2 * pi * r
duration = 200

def on_left(ch):
    global left_count
    left_count += 1

def on_timer(ch):
    global left_count
    # revolutions per second
    rps = left_count * 1000 /duration /20.0
    # speed
    speed = rps * WP
    print(speed)
    # clear count
    left_count = 0

# Interrupter, used to count
left_count = 0
left_pin = Pin(8, Pin.IN, Pin.PULL_UP)
left_pin.irq(trigger=Pin.IRQ_FALLING, handler=on_left)

# Timer, print speed
tim = Timer()
tim.init(period=duration, mode=Timer.PERIODIC, callback=on_timer)

# motor run
import motors
import time

try:
    while True:
        for i in range(20,100,10):
            motors.move("forward",i)
            time.sleep(1)
finally:
    motors.stop()

```

(continues on next page)

(continued from previous page)

```
time.sleep(0.2)
```

4. Calculate the speed on both sides

In the case of a turn, there may be a situation where one side of the wheel is not rotating, but the car is actually moving. Then we can use both sides of the speed module to reduce error.

```
from machine import Timer, Pin
import math

WP = 2 * math.pi * 3.3 # wheel_perimeter(cm): 2 * pi * r
duration = 200

def on_left(ch):
    global left_count
    left_count += 1

def on_right(ch):
    global right_count
    right_count += 1

def on_timer(ch):
    global left_count, right_count
    # revolutions per second
    rps = (left_count + right_count) * 1000 / duration / 20.0 / 2
    # speed
    speed = rps * WP
    print(speed)
    # clear count
    left_count = 0
    right_count = 0

# Interrupter, used to count
left_count = 0
right_count = 0
left_pin = Pin(8, Pin.IN, Pin.PULL_UP)
left_pin.irq(trigger=Pin.IRQ_FALLING, handler=on_left)
right_pin = Pin(9, Pin.IN, Pin.PULL_UP)
right_pin.irq(trigger=Pin.IRQ_FALLING, handler=on_right)

# Timer, print speed
tim = Timer()
tim.init(period=duration, mode=Timer.PERIODIC, callback=on_timer)

# motor run
import motors
import time

try:
    while True:
        for i in range(20, 100, 10):
            motors.move("forward", i)
            time.sleep(1)
finally:
    motors.stop()
    time.sleep(0.2)
```

3. Mileage Calculation

With the ability to calculate speed, counting miles is really a piece of cake.

In this case, the mileage is calculated by multiplying the total number of revolutions by the circumference of the wheel.

```
from machine import Timer, Pin
import math

WP = 2 * math.pi * 3.3 # wheel_perimeter(cm): 2 * pi * r
duration = 200

def on_left(ch):
    global left_count
    left_count += 1

def on_right(ch):
    global right_count
    right_count += 1

def on_timer(ch):
    global left_count, right_count, total_count
    # mileage
    total_count += left_count + right_count
    mileage = total_count / 20.0 / 2 * WP
    # revolutions per second
    rps = (left_count + right_count) * 1000 / duration / 20.0 / 2
    # speed
    speed = rps * WP
    # clear count
    left_count = 0
    right_count = 0
    print("mileage(cm): ", mileage, " ; speed(cm/s): ", speed)

# Interrupter, used to count
left_count = 0
right_count = 0
total_count = 0
left_pin = Pin(8, Pin.IN, Pin.PULL_UP)
left_pin.irq(trigger=Pin.IRQ_FALLING, handler=on_left)
right_pin = Pin(9, Pin.IN, Pin.PULL_UP)
right_pin.irq(trigger=Pin.IRQ_FALLING, handler=on_right)

# Timer, print speed
tim = Timer()
tim.init(period=duration, mode=Timer.PERIODIC, callback=on_timer)


# motor run
import motors
import time


try:
    while True:
        for i in range(20, 100, 10):
            motors.move("forward", i)
            time.sleep(1)
finally:
    motors.stop()
```

(continues on next page)

(continued from previous page)

```
time.sleep(0.2)
```

Copy the above code into Thonny or open the `speed_3_get_mileage.py` under the path of `pico_4wd_car-v2.0\examples\learn_modules`. Then click the  button or press F5 to run it.

After powering up the Pico 4WD car, you can see that the speed keeps cycling between high and low, and Mileage keeps adding up unless you click  to stop the script from running.

4. speed.py Module

Similarly, in order to keep the code simple and organized when using the speed module with other modules, we have encapsulated all its related code into a library.

When you use it, you just need to import this library, and you can call the functions inside directly.

```
get_speed()
get_mileage()
```

Note: The encapsulated library `speed.py` has been saved in `pico_4wd_car-v2.0\libs`, which may differ from the ones shown in the course, so please refer to the file under `libs` path when using it.

```
from machine import Timer, Pin
import math

class Speed():

    WP = 2 * math.pi * 3.3 # wheel_perimeter(cm): 2 * pi * r
    duration = 200 # ms, Timer interval

    def __init__(self, pin1, pin2):
        # value for return
        self.total_count = 0
        self.speed = 0

        # Interrupter, used to count
        self.left_count = 0
        self.right_count = 0
        self.left_pin = Pin(pin1, Pin.IN, Pin.PULL_UP)
        self.left_pin.irq(trigger=Pin.IRQ_FALLING, handler=self.on_left)
        self.right_pin = Pin(pin2, Pin.IN, Pin.PULL_UP)
        self.right_pin.irq(trigger=Pin.IRQ_FALLING, handler=self.on_right)

        # Timer, print speed
        self.tim = Timer()
        self.tim.init(period=self.duration, mode=Timer.PERIODIC, callback=self.on_
↪timer)

    def on_left(self, ch):
        self.left_count += 1

    def on_right(self, ch):
        self.right_count += 1
```

(continues on next page)

(continued from previous page)

```

def on_timer(self, ch):
    # mileage
    self.total_count += self.left_count + self.right_count
    # revolutions per second
    rps = (self.left_count + self.right_count) * 1000 / self.duration / 20.0 / 2
    # speed
    self.speed = rps * self.WP
    # clear count
    self.left_count = 0
    self.right_count = 0

def get_speed(self): # Unit: cm/s
    return self.speed

def get_mileage(self): # Unit: m
    return self.total_count / 20.0 / 2 * self.WP / 100

def reset_mileage(self):
    self.total_count = 0

# init
sp = Speed(8, 9)

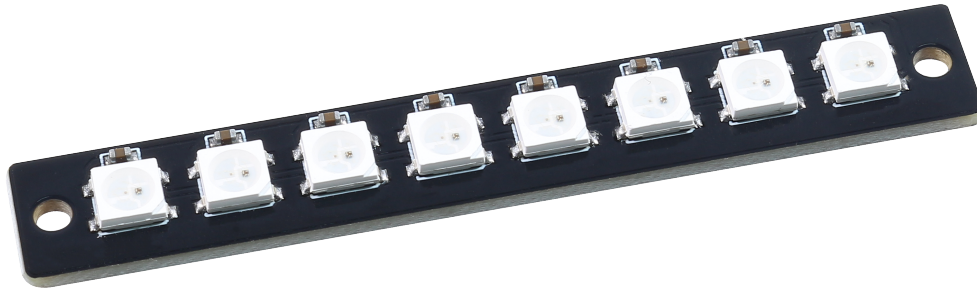
# detect
try:
    import motors
    import time
    while True:
        for i in range(20, 100):
            power = round(i / 10) * 10
            motors.move("forward", power)
            print(f'Power(%):{power}   Speed(cm/s):{sp.get_speed():.2f} Mileage(m):
→{sp.get_mileage():.2f}')
            time.sleep(0.2)
finally:
    motors.stop()
    time.sleep(0.2)

```

3.2.5 5. RGB Board

In this section, you will learn how the RGB Boards work and how to control the 3 RGB Boards to set them to different colors and different effects.

1. Introduce the RGB Boards



- **DO**: Control data signal output
- **DI**: Control data signal input
- **5V**: 5V DC Supply Input
- **GND**: Ground input

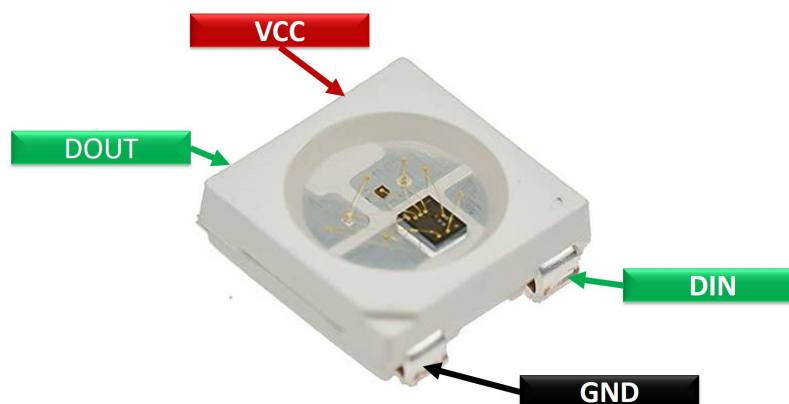
In this kit, there are 3 RGB Boards, each with 8 WS2812B, which can be used for cool light effect display or direction indication.

Features

- Work Voltage: +3.5 ~ +5.3
- RGB Type WS2812B
- Color: Full color
- Port: SH1.0-3P
- Demension: 71mm x 11mm
- Working Temperature: -25 ~ 80

WS2812B

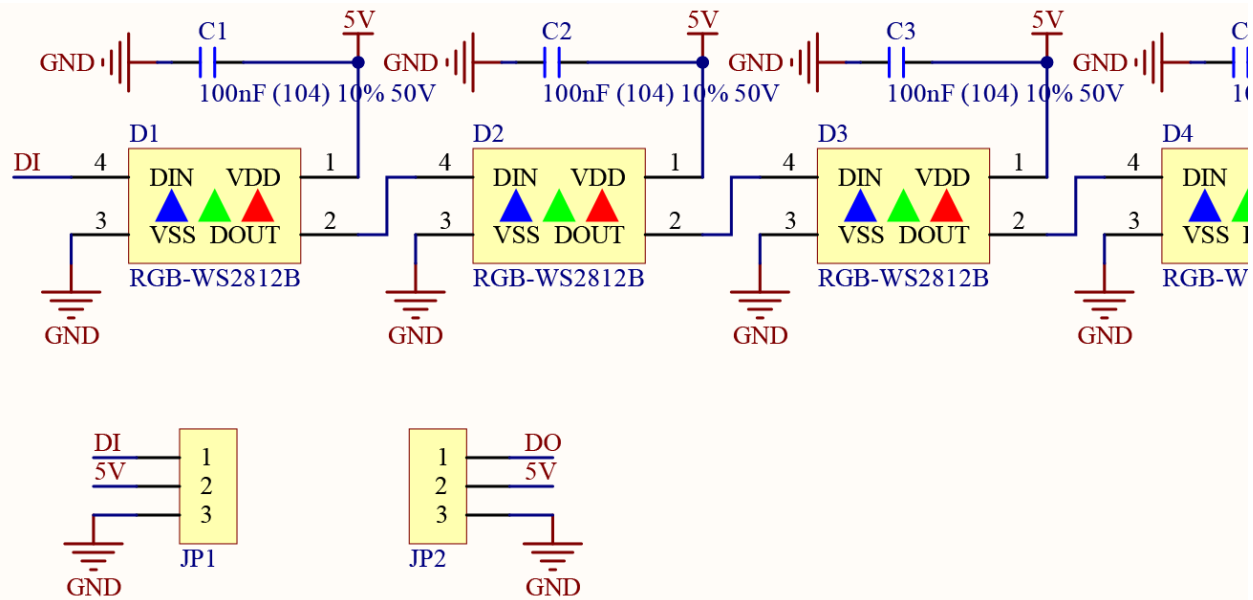
The WS2812B () is a intelligent control LED light source that the control circuit and RGB chip are integrated in a package of 5050 components.



The **DIN** pin receive data from controller, the first WS2812B collect initial 24bit data then sent to the internal data latch, the other data which reshaping by the internal signal reshaping amplification circuit sent to the next cascade WS2812B through the **DO** pin.

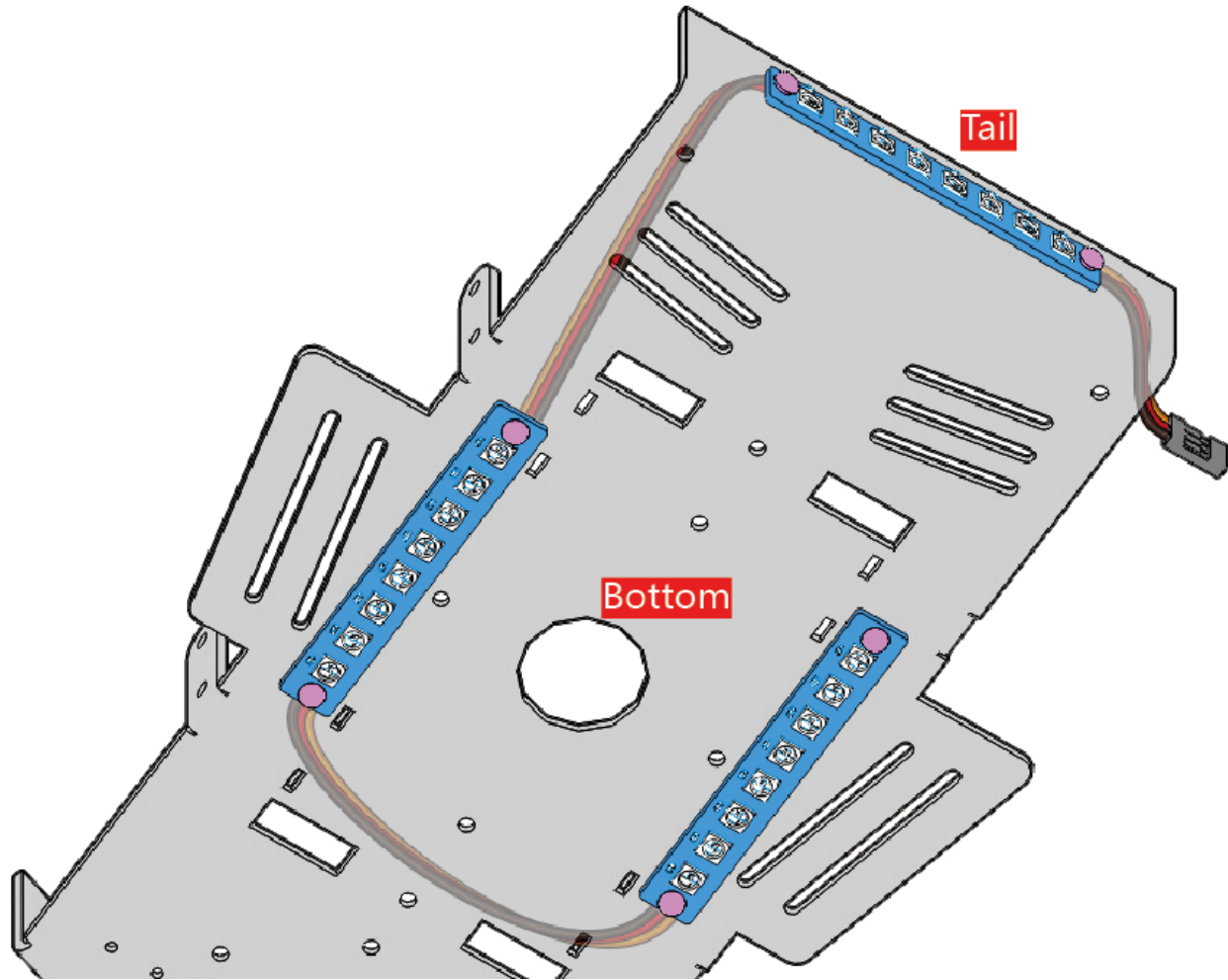
Schematic Diagram

Here is the schematic of the RGB Board (4 WS2812B's are shown here, 8 in total)



Data Transfer

As shown in the diagram below, the 24 bit data is passed from the tail RGB board to the bottom left RGB board, then to the bottom right RGB board when programming.



2. Light up

The RGB Board is a component that does not communicate in SPI or I2C, and it needs to handle things at a much lower level.

Pico has a solution for this: .

As well as the two main Cortex-M0+ processing cores, there are two PIO blocks that each have four state machines. These are really stripped-down processing cores that can be used to handle data coming in or out of the microcontroller, and offload some of the processing requirement for implementing communications protocols.

You can't program these cores with MicroPython - you have to use a special language for them - but you can program them from MicroPython.

1. The code to control RGB Board using PIO is shown below. You can copy the above code into Thonny or open the `rgb_2_light_up.py` under the path of `pico_4wd_car-v2.0\examples\learn_modules`.

Note: You can visit to learn more about the functions involved in the code below.

```
import array, time
from machine import Pin
```

(continues on next page)

(continued from previous page)

```

import rp2
from rp2 import PIO, StateMachine, asm_pio
# Configure the number of WS2812 LEDs.
LIGHT_NUM = 24
LIGHT_PIN = 19


@asm_pio(sideset_init=PIO.OUT_LOW, out_shiftdir=PIO.SHIFT_LEFT,
autopull=True, pull_thresh=24)
def ws2812():
    T1 = 2
    T2 = 5
    T3 = 3
    label("bitloop")
    out(x, 1) .side(0) [T3 - 1]
    jmp(not_x, "do_zero") .side(1) [T1 - 1]
    jmp("bitloop") .side(1) [T2 - 1]
    label("do_zero")
    nop() .side(0) [T2 - 1]

# Create the StateMachine with the ws2812 program, outputting on Pin(19).
sm = StateMachine(0, ws2812, freq=8000000, sideset_base=Pin(LIGHT_PIN))
# Start the StateMachine, it will wait for data on its FIFO.
sm.active(1)

# Display a pattern on the LEDs via a color value.
ar = array.array("I", [0 for _ in range(LIGHT_NUM)])

for i in range(LIGHT_NUM):
    ar[i] = 0xffaa00
sm.put(ar, 8)

```

2. To run the script, click the  button or press F5. When you power up the Pico 4WD car, you will see 3 RGB boards lit up in cyan color.

3. ws2812.py Module

In order to be able to make the RGB Board light up different colors and display different effects in a complex project, we need to encapsulate the previous project into a (module)library.

Note: The encapsulated library `ws2812.py` has been saved in `pico_4wd_car-v2.0\libs`, which may differ from the ones shown in the course, so please refer to the file under `libs` path when using it.

Hex Format

Next, let's look at the simple encapsulated code.

```

from machine import Pin
from rp2 import PIO, StateMachine, asm_pio
import array
import time

@asm_pio(sideset_init=PIO.OUT_LOW, out_shiftdir=PIO.SHIFT_LEFT,
autopull=True, pull_thresh=24)
def ws2812():

```

(continues on next page)

(continued from previous page)

```

T1 = 2
T2 = 5
T3 = 3
label("bitloop")
out(x, 1).side(0)[T3 - 1]
jmp(not_x, "do_zero").side(1)[T1 - 1]
jmp("bitloop").side(1)[T2 - 1]
label("do_zero")
nop().side(0)[T2 - 1]

class WS2812():

    def __init__(self, pin, num):
        # Configure the number of WS2812 LEDs.
        self.led_nums = num
        self.pin = pin
        self.sm = StateMachine(0, ws2812, freq=8000000, sideset_
→base=Pin(self.pin))
        # Start the StateMachine, it will wait for data on its FIFO.
        self.sm.active(1)

        self.buf = array.array("I", [0 for _ in range(self.led_nums)])

    def write(self):
        self.sm.put(self.buf, 8)


    def __getitem__(self, i):
        return self.buf[i]

    def __setitem__(self, i, value):
        self.buf[i] = value

# Display a pattern on the LEDs via a color value.
LIGHT_PIN = 19
LIGHT_NUM = 24
np = WS2812(LIGHT_PIN, LIGHT_NUM)

for i in range(LIGHT_NUM):
    np[i] = 0x00aaff
np.write()

```

Now you can copy this code into Thonny and then click the  button or press F5 to run it. When you power up the Pico 4WD car, you will see 3 RGB boards lit up in magenta color.

RGB Format

Additionally to hexadecimal, people often prefer RGB color representations. Therefore, we perform a little optimization, converting colors from HEX to RGB.

Focus on the highlighted functions to see how the hexadecimal format is converted to RGB format.

```

from machine import Pin
from rp2 import PIO, StateMachine, asm_pio
import array
import time

```

(continues on next page)

(continued from previous page)

```

@asm_pio(sideset_init=PIO.OUT_LOW, out_shiftdir=PIO.SHIFT_LEFT,
↳autopull=True, pull_thresh=24)
def ws2812():
    T1 = 2
    T2 = 5
    T3 = 3
    label("bitloop")
    out(x, 1).side(0)[T3 - 1]
    jmp(not_x, "do_zero").side(1)[T1 - 1]
    jmp("bitloop").side(1)[T2 - 1]
    label("do_zero")
    nop().side(0)[T2 - 1]

class WS2812():

    def __init__(self, pin, num):
        # Configure the number of WS2812 LEDs.
        self.led_nums = num
        self.pin = pin
        self.sm = StateMachine(0, ws2812, freq=8000000, sideset_
↳base=Pin(self.pin))
        # Start the StateMachine, it will wait for data on its FIFO.
        self.sm.active(1)

        self.buf = array.array("I", [0 for _ in range(self.led_nums)])

    def write(self):
        self.sm.put(self.buf, 8)

    def list_to_hex(self, color):
        if isinstance(color, list) and len(color) == 3:
            return (color[0] << 8) + (color[1] << 16) + (color[2])
        elif isinstance(color, int):
            value = (color & 0xFF0000)>>8 | (color & 0x00FF00)<<8 | (color &
↳0x0000FF)
            return value
        else:
            raise ValueError("Color must be 24-bit RGB hex or list of 3 8-
↳bit RGB, not %s"%color)

    def hex_to_list(self, color):
        if isinstance(color, list) and len(color) == 3:
            return color
        elif isinstance(color, int):
            r = color >> 8 & 0xFF
            g = color >> 16 & 0xFF
            b = color >> 0 & 0xFF
            return [r, g, b]
        else:
            raise ValueError("Color must be 24-bit RGB hex or list of 3 8-
↳bit RGB, not %s"%color)

    def __getitem__(self, i):
        return self.hex_to_list(self.buf[i])

    def __setitem__(self, i, value):
        self.buf[i] = self.list_to_hex(value)

```

(continues on next page)

(continued from previous page)

```
# Display a pattern on the LEDs via an array of LED RGB values.
LIGHT_PIN = 19
LIGHT_NUM = 24
np = WS2812(LIGHT_PIN, LIGHT_NUM)

for i in range(LIGHT_NUM):
    np[i] = [0,255,110]
np.write()
time.sleep(1)

for i in range(LIGHT_NUM):
    np[i] = 0xFF00AA
np.write()
```

As you can see at the bottom of the code, we use both RGB and hexadecimal colors, `[0, 255, 110]` and `0xFF00AA`. You can choose one according to your preference.

You can also copy it into Thonny and run it to see what effect it has.

4. light.py Module


If we want to add light effects to the car, we will need to change some LEDs separately. In order to add left turn indicator, we must control the LEDs with serial number 6 and 7; if we modify the right bottom RGB Board, we must control the LEDs with serial number 16~23.

When writing code, it is easy to get confused. To make things even easier, we created different functions that could control different RGB boards or LEDs at specific locations. These functions were then wrapped in a new library.

For example, if you want the 2 LEDs on the tail RGB board to be illuminated, use the following command.

```
set_rear_right_color(0xaa0000)
```

Note: The encapsulated library `lights.py` has been saved in `pico_4wd_car-v2.0\libs`, which may differ from the ones shown in the course, so please refer to the file under `libs` path when using it.

Now you can copy this code into Thonny and then click the  button or press F5 to run it.

When you power up the Pico 4WD car, you will see 3 RGB boards displaying many different color effects.

```
from ws2812 import WS2812

LIGHT_PIN = 19
LIGHT_NUM = 24

LIGHT_REAR_RIGHT = [0, 1]
LIGHT_REAR_MIDDLE = [2, 3, 4, 5]
LIGHT_REAR_LEFT = [6, 7]
LIGHT_REAR = [0, 1, 2, 3, 4, 5, 6, 7]

LIGHT_BOTTOM_LEFT = [8, 9, 10, 11, 12, 13, 14, 15]
LIGHT_BOTTOM_RIGHT = [16, 17, 18, 19, 20, 21, 22, 23]

np = WS2812(LIGHT_PIN, LIGHT_NUM)
```

(continues on next page)

(continued from previous page)

```

def set_all_color(color):
    ''' set color to all lights

        color list or hex, 1*3 list, the order is [red, green, blue]
    '''
    for i in range(24):
        np[i] = color
    np.write()

def set_color_at(num, color):
    if isinstance(num, list):
        for i in num:
            if i > LIGHT_NUM:
                raise ValueError("num element must be less than LIGHT_NUM.")
            np[i] = color
    elif type(num) is int:
        if num > LIGHT_NUM:
            raise ValueError("num must be less than LIGHT_NUM.")
        np[num] = color
    else:
        raise ValueError("num must be list or int.")
    np.write()

def set_bottom_left_color(color):
    for num in LIGHT_BOTTOM_LEFT:
        np[num] = color
    np.write()

def set_bottom_right_color(color):
    for num in LIGHT_BOTTOM_RIGHT:
        np[num] = color
    np.write()

def set_bottom_color(color):
    for num in LIGHT_BOTTOM_LEFT:
        np[num] = color
    for num in LIGHT_BOTTOM_RIGHT:
        np[num] = color
    np.write()

def set_rear_color(color):
    for num in LIGHT_REAR:
        np[num] = color
    np.write()

def set_rear_right_color(color):
    for num in LIGHT_REAR_RIGHT:
        np[num] = color
    np.write()

def set_rear_middle_color(color):
    for num in LIGHT_REAR_MIDDLE:
        np[num] = color
    np.write()

def set_rear_left_color(color):

```

(continues on next page)

(continued from previous page)

```

    for num in LIGHT_REAR_LEFT:
        np[num] = color
    np.write()

def set_off():
    set_all_color([0, 0, 0])

# call function

import time
set_all_color(0x33aa66)
time.sleep(0.5)
set_bottom_color([255, 255, 100])
time.sleep(0.5)
set_bottom_left_color(0x6633ff)
time.sleep(0.5)
set_bottom_right_color([255, 66, 100])
time.sleep(0.5)
set_rear_color(0x88aa00)
time.sleep(0.5)
set_rear_right_color(0xaa0000)
time.sleep(0.5)
set_rear_middle_color(0x00aa00)
time.sleep(0.5)
set_rear_left_color(0x0000aa)
time.sleep(0.5)

for i in range(8):
    set_off()
    time.sleep(0.01)
    set_color_at(i, 0xaa00cc)
    time.sleep(0.3)

set_off()
time.sleep(0.1)
set_color_at([1, 3, 5, 7], 0xccaa00)
time.sleep(0.5)

set_off()
time.sleep(0.1)
set_color_at([0, 2, 4, 6], 0x00ccaa)
time.sleep(0.5)

set_off()

```

3.3 3. Funny Projects

In this project, you will learn how to combine different modules to make the Pico 4WD car do interesting projects!

Note: You need to check if your Raspberry Pi Pico contains these libraries first. If not, you can refer to the tutorial [3. Upload the Libraries to Pico \(Optional\)](#) to upload them.

Note: In order to allow the car to move on the ground without the USB cable connected, you need to save this script as `main.py` to Raspberry Pi Pico, see [5. Run Script Offline\(Important\)](#) for a tutorial.

3.3.1 1. Don't Push Me

To prevent Pico 4WD car from running off a cliff, let's give it a little sense of self-preservation.

In this project, the car will be dormant, and if you push it over the edge of a cliff, it will be awakened, then back up and shake its head in displeasure.

Note:

- The complete script `project_1_cliff.py` is in the path `pico_4wd_car\examples\funny_projects`.
 - In order to allow the car to move on the ground without the USB cable connected, you need to save this script as `main.py` to Raspberry Pi Pico, see [5. Run Script Offline\(Important\)](#) for a tutorial.
-

Below are the steps to implement the cliff function, and you can copy them into Thonny to run them.

1. Import libraries

1. In this project, we use grayscale module, motor and servo, so we import the related libraries here.

```
import motors as car
from servo import Servo
from grayscale import Grayscale
import time

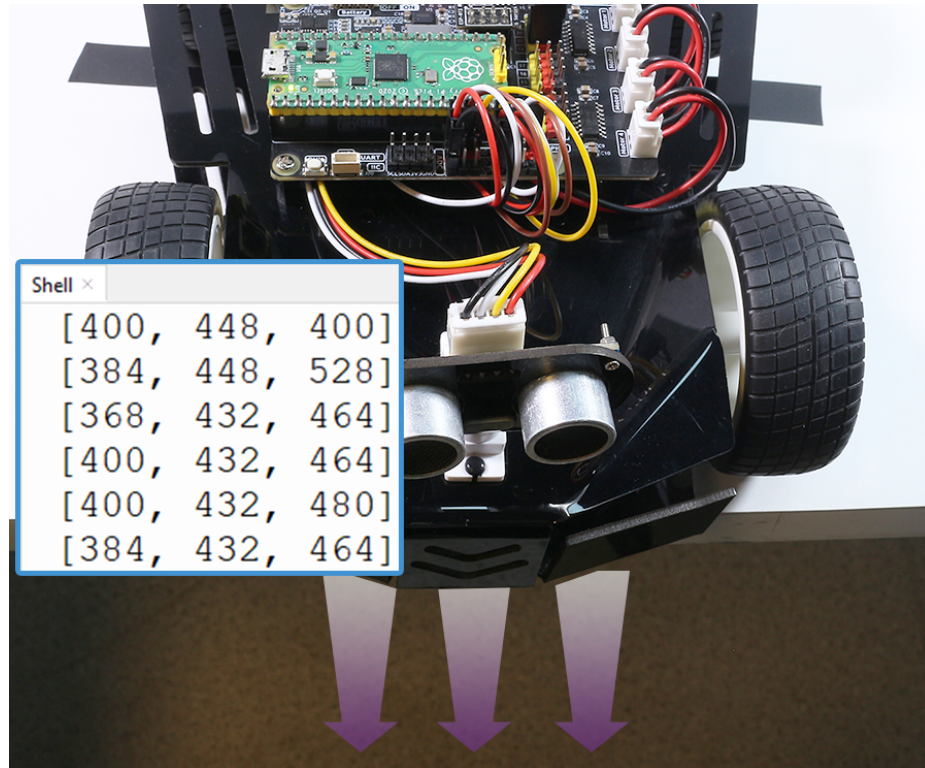
# init grayscale module
gs = Grayscale(26, 27, 28)
gs.set_edge_reference(1000)

# init servo
servo = Servo(18)
servo.set_angle(0)

MOTOR_POWER = 50

try:
    while True:
        print(gs.get_value())
        time.sleep(0.05)
finally:
    pass
```

- Note that this command `gs.set_edge_reference(1000)` is the cliff threshold set according to my current environment, what you actually use may be different.
- Run the script after copying it into Thonny and then hang the car in the air (10cm or more).



- Finally, fill in a value in `gs.set_edge_reference()` according to the result in the Shell.

2. Back up when detected

When the grayscale module detects a cliff, go backward; otherwise, stop in place.

```
import motors as car
from servo import Servo
from grayscale import Grayscale
import time

# init grayscale module
gs = Grayscale(26, 27, 28)
gs.set_edge_reference(1000)

# init servo
servo = Servo(18)
servo.set_angle(0)

MOTOR_POWER = 50

def main():
    while True:
        # print(gs.get_value())
        if gs.is_on_edge():
            car.move("backward", MOTOR_POWER)
        else:
            car.move("stop")

try:
    main()
```

(continues on next page)

(continued from previous page)

```

finally:
    car.move("stop")
    time.sleep(0.05)

```

3. Shaking head while backing up

To make the car more cute, let it shake its head while backing up.

```

import motors as car
from servo import Servo
from grayscale import Grayscale
import time

# init grayscale module
gs = Grayscale(26, 27, 28)
gs.set_edge_reference(1000)

# init servo
servo = Servo(18)
servo.set_angle(0)

MOTOR_POWER = 50

def shake_head():
    for angle in range(0, 90, 10):
        servo.set_angle(angle)
        time.sleep(0.01)
    for angle in range(90, -90, -10):
        servo.set_angle(angle)
        time.sleep(0.01)
    for angle in range(-90, 0, 10):
        servo.set_angle(angle)
        time.sleep(0.01)

def main():
    while True:
        # print(gs.get_value())
        if gs.is_on_edge():
            car.move("backward", MOTOR_POWER)
            shake_head()
            shake_head()
        else:
            car.move("stop")

try:
    main()
finally:
    car.move("stop")
    time.sleep(0.05)

```

3.3.2 2. Line Track

Let Pico-4wd walk its exclusive avenue! Use the black electrical tape to stick a line on a light-colored floor (or table). After running the script, you will see Pico-4wd tracking this line forward.

Note:

- The complete script `project_2_line.py` is in the path `pico_4wd_car\examples\funny_projects`.
 - In order to allow the car to move on the ground without the USB cable connected, you need to save this script as `main.py` to Raspberry Pi Pico, see [5. Run Script Offline\(Important\)](#) for a tutorial.
-

Below are the steps to implement the line track function, and you can copy them into Thonny to run them.

1. Get line threshold

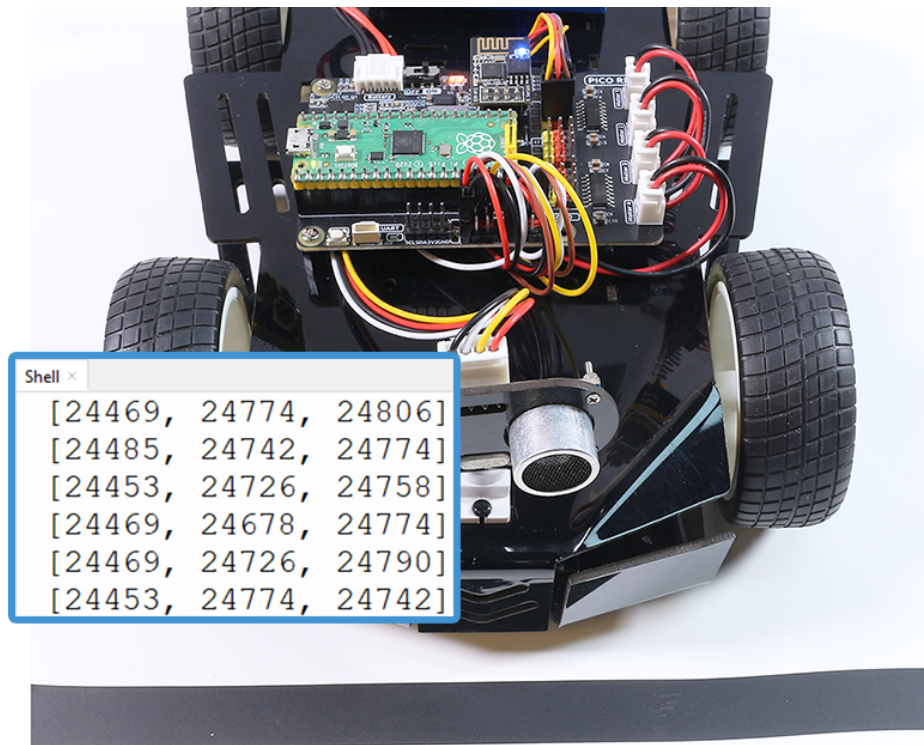
```
from grayscale import Grayscale
import motors as car
import lights
import time

# init grayscale module
gs = Grayscale(26, 27, 28)
gs.set_line_reference(10000)

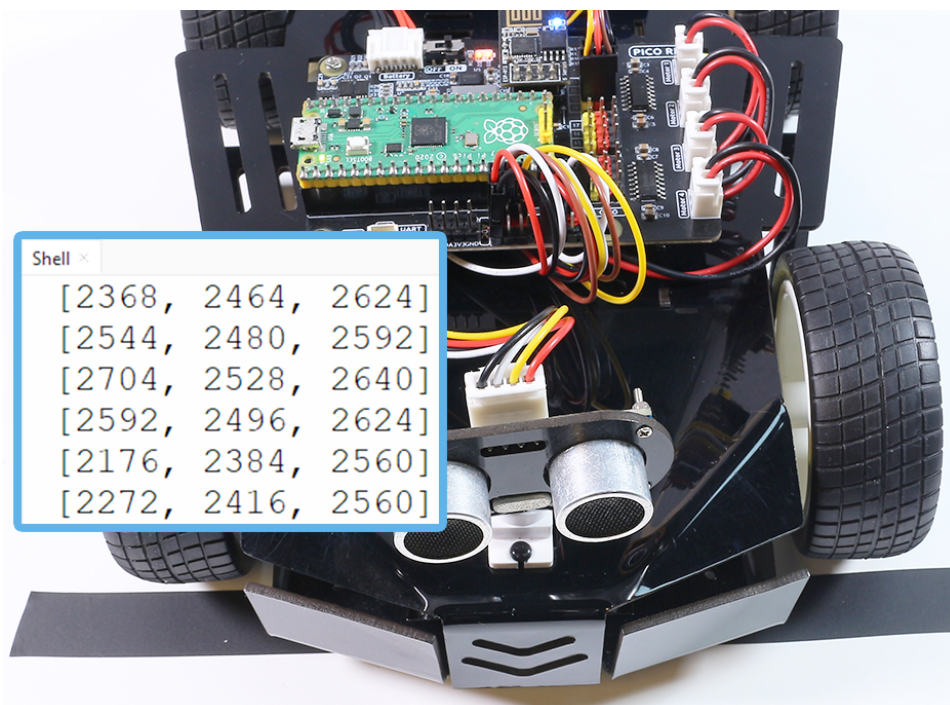
try:
    while True:
        print(gs.get_value())
        time.sleep(0.2)
finally:
    pass
```

- Run the script after copying it into Thonny.
- After powering up the Pico 4WD car, place the grayscale module in following environments to see how the data in the Shell changes (keeping the USB cable connected to the computer).

White surface You will find that the value of the white surface is generally large, for example mine is around 240,000.



Black line The value on the black line will be smaller, and now I'm at about 2000.



- My car reads around 24000 in the white area and around 2000 in the black line, so I set `set_line_reference()` to about the middle value of 10000.

2. Tracking the line

- When the detected grayscale value of the corresponding channel is less than

set_line_reference(10000), a 1 will be output, which means a black line is detected.

- Then all three sets of data ([0, 1, 0]) will be output by get_line_status().
- The car is then moved according to the detection data from the three sensors.

```
from grayscale import Grayscale
import motors as car
import lights

gs = Grayscale(26, 27, 28)
gs.set_line_reference(10000)

MOTOR_POWER = 30

def line_track():
    while True:
        gs_data = gs.get_line_status()
        if gs_data == [0, 1, 0]:
            car.set_motors_power([MOTOR_POWER, MOTOR_POWER, MOTOR_POWER,
↪MOTOR_POWER]) # forward
        elif gs_data == [0, 1, 1]:
            car.set_motors_power([MOTOR_POWER, 0, MOTOR_POWER, 0]) # turn_
↪right at a small angle
        elif gs_data == [0, 0, 1]:
            car.set_motors_power([MOTOR_POWER, -MOTOR_POWER, MOTOR_POWER, -
↪MOTOR_POWER]) # turn right at a small angle
        elif gs_data == [1, 1, 0]:
            car.set_motors_power([0, MOTOR_POWER, 0, MOTOR_POWER]) # turn_
↪left at a small angle
        elif gs_data == [1, 0, 0]:
            car.set_motors_power([-MOTOR_POWER, MOTOR_POWER, -MOTOR_POWER,
↪MOTOR_POWER]) # turn left at a small angle

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

3. Add light effects

Finally, while driving, let the two RGB Boards at the bottom light up according to the direction of the car moving.

For example, when moving forward, the bottom two RGB Boards are lit in green. When turning right, let the right RGB Board light up, and when turning left, let the left RGB Board light up.

```
from grayscale import Grayscale
import motors as car
import lights

gs = Grayscale(26, 27, 28)
gs.set_line_reference(10000)

MOTOR_POWER = 30

def line_track():
    while True:
        gs_data = gs.get_line_status()
```

(continues on next page)

(continued from previous page)

```

        if gs_data == [0, 1, 0]:
            car.set_motors_power([MOTOR_POWER, MOTOR_POWER, MOTOR_POWER, ↵
↵MOTOR_POWER]) # forward
            lights.set_bottom_color([0, 100, 0])
        elif gs_data == [0, 1, 1]:
            car.set_motors_power([MOTOR_POWER, 0, MOTOR_POWER, 0]) # turn_
↵right at a small angle
            lights.set_off()
            lights.set_bottom_left_color([50, 50, 0])
        elif gs_data == [0, 0, 1]:
            car.set_motors_power([MOTOR_POWER, -MOTOR_POWER, MOTOR_POWER, -
↵MOTOR_POWER]) # turn right at a small angle
            lights.set_off()
            lights.set_bottom_left_color([100, 5, 0])
        elif gs_data == [1, 1, 0]:
            car.set_motors_power([0, MOTOR_POWER, 0, MOTOR_POWER]) # turn_
↵left at a small angle
            lights.set_off()
            lights.set_bottom_right_color([50, 50, 0])
        elif gs_data == [1, 0, 0]:
            car.set_motors_power([-MOTOR_POWER, MOTOR_POWER, -MOTOR_POWER, ↵
↵MOTOR_POWER]) # turn left at a small angle
            lights.set_off()
            lights.set_bottom_right_color([100, 0, 0])

    try:
        line_track()
    finally:
        car.move("stop")
        lights.set_off()

```

3.3.3 3. Objects Follow

In this project, Pico-4wd car will follow the object in front of it. For example, if you put your hand in front of Pico 4WD, it will rush towards your hand. And it will also play the advantage of sonar scanning, will judge your hand position and then adjust the forward direction.

Note:

- The complete script `project_3_follow.py` is in the path `pico_4wd_car\examples\funny_projects`.
- In order to allow the car to move on the ground without the USB cable connected, you need to save this script as `main.py` to Raspberry Pi Pico, see [5. Run Script Offline\(Important\)](#) for a tutorial.

Below are the steps to implement the object follow function, and you can copy them into Thonny to run them.

1. Sonar scanning

Let the car scan the area between -45° and 45° in front of it. Let the ultrasonic module read the values every 10° and then print out 10 sets of values at once.

```

import sonar as sonar
import motors as car
import time

try:
    sonar.set_sonar_scan_config(scan_range=90, step=10)
    while True:
        _, sonar_data = sonar.sonar_scan()
        # sonar_data: 0 is block, 1 is pass
        time.sleep(0.04)

        # If sonar data return a int, means scan not finished, and
        ↳ the int is current angle status
        if isinstance(sonar_data, int):
            continue # only list can go on

        print(sonar_data)

finally:
    pass

```

- Run the script after copying it into Thonny.
- After powering up the Pico 4WD car, you will see the array as shown below, 0 means that an obstacle has been detected.

```
[0, 0, 1, 1, 0, 0, 0, 0, 1, 1]
```

2. Divide the direction

Now to process the scan results. The scan range should be divided into three areas: left, middle and right. After that, determine where the center point of the largest obstacle is located, for example on the left, then let the car go to the left (here just show to the left, do not let the car move).

Let the car stop if the obstacle is too small or there is no obstacle.

```

import sonar as sonar
import motors as car
import time

def get_dir(data, split_str='0'):

    # get scan status of 0, 1
    data = [str(i) for i in data]
    data = "".join(data)
    print("Data: ", data)

    # Split 1, leaves the object path
    paths = data.split(split_str)
    print("Divide the Data: ", paths)

    # Find the max path
    max_paths = max(paths)
    print("Max Path: ", max_paths)

    # If no object
    if len(max_paths) < 3:
        return "stop"

```

(continues on next page)

(continued from previous page)

```

# Calculate the direction of the biggest one
position = data.index(max_paths) # find the biggest object position
position += (len(max_paths)-1)/2 # find the middle of the biggest object
print("Max Path's Direction: ",position)

# Divide the scanning area into three pieces and mark the right one
if position < len(data) / 3:
    return "left"
elif position > 2 * len(data) / 3:
    return "right"
else:
    return "forward"

try:
    sonar.set_sonar_scan_config(scan_range=90, step=10)
    while True:
        _,_, sonar_data = sonar.sonar_scan()
        # sonar_data: 0 is block, 1 is pass
        time.sleep(0.04)

        # If sonar data return a int, means scan not finished, and the int_
        ↪is current angle status
        if isinstance(sonar_data,int):
            continue # only list can go on

        direction = get_dir(sonar_data,split_str='1')
        print("The Car Should Go: ", direction)

finally:
    pass

```

- Run the script after copying it into Thonny.
- After powering up the Pico 4WD car, you will see the following data in the Shell. Now look at how this data is explained.

```

Data: 0011000011
Divide the Data: ['00', '', '0000', '', '']
Max Path: 0000
Max Path's Direction: 5.5
The Car Should Go: forward

```

- Separate the data 0011000011 with 1 as a separator to get the array ['00', '', '0000', '', ''].
- The largest obstacle is '0000'.
- They are located in the fourth to seventh bits of the data (the first bit is the 0th bit), and that center point is at the 5.5 bit position.
- Divide the length of the data (0011000011) into thirds. Since the length of this data set is 10, those less than 3.3 are on the left side and those greater than 6.6 are on the right side.
- 5.5 is in the middle, then the car should be forward.

3. Follow your hand

Make the Pico 4WD car move in the direction of the obstacle, for example, if there is an obstacle on the left, move to the left.

```
import sonar as sonar
import motors as car
import time

def get_dir(data, split_str='0'):

    # get scan status of 0, 1
    data = [str(i) for i in data]
    data = "".join(data)

    # Split 1, leaves the object path
    paths = data.split(split_str)
    max_paths=max(paths)

    # no object
    if len(max_paths)<3:
        return "stop"

    # Calculate the direction of the biggest one
    position = data.index(max_paths) # find the biggest object position
    position += (len(max_paths)-1)/2 # find the middle of the biggest object

    # Divide the scanning area into three pieces and mark the right one
    if position < len(data) / 3:
        return "left"
    elif position > 2 * len(data) / 3:
        return "right"
    else:
        return "forward"

def running(direction, power):
    if direction == "left":
        car.move("left", power)
    elif direction == "right":
        car.move("right", power)
    elif direction == "forward":
        car.move("forward", power)
    else:
        car.move("stop")

try:
    MOTOR_POWER = 20
    sonar.set_sonar_scan_config(scan_range=90, step=10)
    while True:
        __, sonar_data = sonar.sonar_scan()
        # sonar_data: 0 is block, 1 is pass
        time.sleep(0.04)

        # If sonar data return a int, means scan not finished, and the int_
        ↪ is current angle status
        if isinstance(sonar_data, int):
            continue # only list can go on

        direction = get_dir(sonar_data, split_str='1')
        running(direction, MOTOR_POWER)
```

(continues on next page)

(continued from previous page)

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

3.3.4 4. Obstacle Avoid

Let Pico-4wd do a challenging task: automatically avoid obstacles! When an obstacle is detected, instead of simply backing up, the sonar scans the surrounding area and finds the widest way to move forward.

This project is an advanced version of the previous project [3. Objects Follow](#). The previous project was to find the direction where the obstacle is, while here it is to find the direction where there is no obstacle. To get a better understanding of this project, it is recommended that you finish the previous project first.

Note:

- The complete script `project_4_avoid.py` is in the path `pico_4wd_car\examples\funny_projects`.
 - In order to allow the car to move on the ground without the USB cable connected, you need to save this script as `main.py` to Raspberry Pi Pico, see [5. Run Script Offline\(Important\)](#) for a tutorial.
-

Below are the steps to implement the obstacle avoidance function, and you can copy them into Thonny to run them.

1. Sonar scanning

- Let the car scan the area between -30° and 30° in front of it. Let the ultrasonic module read the values every 10° and then print out at once.
- Then process the scan results. In contrast to the previous project, here the scan data is separated by 0, resulting in an area with no obstacles.
- Data length is divided into left, middle, and right. Locate the center point of the region without obstacles and return "forward" if it is there.
- If both have obstacles, then return "left".

```
import sonar as sonar
import motors as car
import time

def get_dir(data, split_str='0'):

    # get scan status of 0, 1
    data = [str(i) for i in data]
    data = "".join(data)

    # Split 0, leaves the free path
    paths = data.split(split_str)

    # Find the max path
    max_paths=max(paths)

    # If no wide enough path
    if len(max_paths)<2:
```

(continues on next page)

(continued from previous page)

```

        return "left"

    # Calculate the direction of the widest one
    position = data.index(max_paths) # find the widest path position
    position += (len(max_paths)-1)/2 # find the middle of the widest path

    # Divide the scanning area into three pieces and mark the widest one
    if position < len(data) / 3:
        return "left"
    elif position > 2 * len(data) / 3:
        return "right"
    else:
        return "forward"

try:
    sonar.set_sonar_scan_config(scan_range=60, step=10)
    sonar.set_sonar_reference(30)
    while True:
        _, _, sonar_data = sonar.sonar_scan()
        # sonar_data: 0 is block, 1 is pass
        time.sleep(0.04)

        # If sonar data return a int, means scan not finished, and the int_
        is current angle status
        if isinstance(sonar_data, int):
            continue # only list can go on

        direction = get_dir(sonar_data, split_str='0')
        print(sonar_data)
        print("The Car Should Go: ", direction)

finally:
    pass

```

2. Avoiding obstacles

- Move according to the result returned after processing. For example, if "left" is returned, make the car turn left.
- While the car is turning, rotate the sonar scanner in the opposite direction until no obstacle is detected.
- For example, if it is turning left, the sonar scanner will turn to the right.

```

import sonar as sonar
import motors as car
import time

def get_dir(data, split_str='0'):

    # get scan status of 0, 1
    data = [str(i) for i in data]
    data = "".join(data)

    # Split 0, leaves the free path
    paths = data.split(split_str)

```

(continues on next page)

(continued from previous page)

```

# Find the max path
max_paths=max(paths)

# If no wide enough path
if len(max_paths)<4:
    return "left"

# Calculate the direction of the widest one
position = data.index(max_paths) # find the widest path position
position += (len(max_paths)-1)/2 # find the middle of the widest_
↪path

# Divide the scanning area into three pieces and mark the widest_
↪one
if position < len(data) / 3:
    return "left"
elif position > 2 * len(data) / 3:
    return "right"
else:
    return "forward"

def running(direction,power):
    if direction is "left":
        sonar.get_distance_at(20) # face right
        time.sleep(0.2)
        car.move("left", power*2)
        while True:
            distance = sonar.get_distance_at(20) # face right
            status = sonar.get_sonar_status(distance)
            if status is 1: # right position is pass
                break
        car.move("stop")
    elif direction is "right":
        sonar.get_distance_at(-20) # face left
        time.sleep(0.2)
        car.move("right", power*2)
        while True:
            distance = sonar.get_distance_at(-20) # face left
            status = sonar.get_sonar_status(distance)
            if status is 1: # left position is pass
                break
        car.move("stop")
    else:
        # pass
        car.move("forward",power)

try:
    MOTOR_POWER = 30
    sonar.set_sonar_scan_config(scan_range=60, step=10)
    sonar.set_sonar_reference(30)
    while True:
        _, _, sonar_data = sonar.sonar_scan()
        # sonar_data: 0 is block, 1 is pass
        time.sleep(0.04)

        # If sonar data return a int, means scan not finished, and_
        ↪the int is current angle status

```

(continues on next page)

(continued from previous page)

```

    if isinstance(sonar_data, int):
        continue # only list can go on
    direction = get_dir(sonar_data, split_str='0')
    running(direction, MOTOR_POWER)

finally:
    car.move("stop")

```

3. Reduce scanning angle

The car only needs to detect if there is an obstacle in front of it while moving forward. However, when it encounters an obstacle, it should stop and find a new route. Thus, the sonar scanner's search range should be changed from 60° to 180°.

```

import sonar as sonar
import motors as car
import time

def get_dir(data, split_str='0'):

    # get scan status of 0, 1
    data = [str(i) for i in data]
    data = "".join(data)

    # Split 0, leaves the free path
    paths = data.split(split_str)

    # Find the max path
    max_paths=max(paths)

    # If no wide enough path
    if len(max_paths)<4:
        return "left"

    # Calculate the direction of the widest one
    position = data.index(max_paths) # find the widest path position
    position += (len(max_paths)-1)/2 # find the middle of the widest path

    # Divide the scanning area into three pieces and mark the widest one
    if position < len(data) / 3:
        return "left"
    elif position > 2 * len(data) / 3:
        return "right"
    else:
        return "forward"

def running(direction, power):
    if direction is "left":
        sonar.get_distance_at(20) # face right
        time.sleep(0.2)
        car.move("left", power*2)
        while True:
            distance = sonar.get_distance_at(20) # face right
            status = sonar.get_sonar_status(distance)
            if status is 1: # right position is pass
                break
        car.move("stop")

```

(continues on next page)

(continued from previous page)

```

elif direction is "right":
    sonar.get_distance_at(-20) # face left
    time.sleep(0.2)
    car.move("right", power*2)
    while True:
        distance = sonar.get_distance_at(-20) # face left
        status = sonar.get_sonar_status(distance)
        if status is 1: # left position is pass
            break
    car.move("stop")
else:
    # pass
    car.move("forward", power)

try:
    MOTOR_POWER = 30
    SCAN_RANGE_PASS = 60
    SCAN_RANGE_BLOCK = 180
    SCAN_STEP = 10
    status = "pass"
    sonar.set_sonar_scan_config(scan_range=SCAN_RANGE_PASS, step=SCAN_STEP)
    sonar.set_sonar_reference(30)
    while True:
        _, _, sonar_data = sonar.sonar_scan()
        # sonar_data: 0 is block, 1 is pass
        time.sleep(0.04)

        # If sonar data return a int, means scan not finished, and the int_
        ↳ is current angle status
        if isinstance(sonar_data, int):
            if sonar_data is 0 and status is "pass": #If it finds an obstacle
                status = "block"
                car.move("stop")
                sonar.set_sonar_scan_config(SCAN_RANGE_BLOCK) # change scan_
                ↳ range to 180 and re-scan
                continue # only list can go on
            direction = get_dir(sonar_data, split_str='0')
            running(direction, MOTOR_POWER)
            status = "pass" # find a passable way
            sonar.set_sonar_scan_config(SCAN_RANGE_PASS) # change scan range to_
            ↳ 60 for go forward

finally:
    car.move("stop")

```

3.3.5 5. Light with Movement

We can make the car light up yellow when turning, red when going backwards and green when going forward.

Also such lighting effects can be added to other projects such as *1. Don't Push Me*, *3. Objects Follow* and *4. Obstacle Avoid*. It's easy to add, just import this light effect script to other projects as a library, no need to make complicated changes in the source code.

1. Realize interesting light effects (project_5_light.py)

The following is the light effect code for the movement of the car.

- When the car moves forward, the bottom RGB board and the middle 4 LEDs on the tail are lit in green, and backward is red.
- When turning left or right, the bottom RGB board and the left or right two LEDs on the tail are lit in yellow.

```
import motors as car
import lights
import time

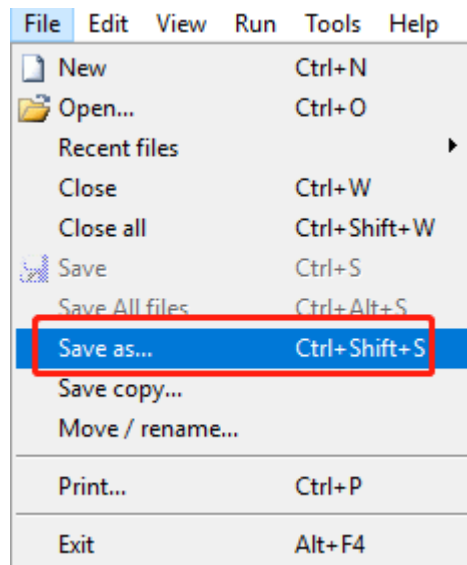
def move(action, power=0):
    car.move(action, power)
    if action is "forward":
        lights.set_off()
        lights.set_bottom_color(0x00aa00)
        lights.set_rear_middle_color(0x00aa00)
    elif action is "left":
        lights.set_off()
        lights.set_rear_left_color(0xaaaa00)
        lights.set_bottom_left_color(0xaaaa00)
    elif action is "right":
        lights.set_off()
        lights.set_rear_right_color(0xaaaa00)
        lights.set_bottom_right_color(0xaaaa00)
    elif action is "backward":
        lights.set_off()
        lights.set_rear_middle_color(0xaa0000)
        lights.set_bottom_color(0xaa0000)
    else:
        lights.set_off()

if __name__ == "__main__":
    try:
        while True:
            speed = 50
            act_list = [
                "forward",
                "backward",
                "left",
                "right",
                "stop",
            ]
            for act in act_list:
                print(act)
                move(act, speed)
                time.sleep(1)
    finally:
        move("stop")
        lights.set_off()
        time.sleep(0.05)
```

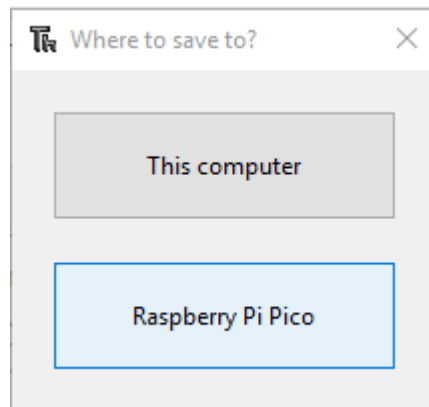
2. Upload to Raspberry Pico

In order to apply the above lighting effect to other projects, you need to save it to the Raspberry Pi Pico by following these steps.

1. Copy the above code into Thonny or open the `project_5_light.py` under the path of `pico_4wd_car-v2.0\examples\funny_projects`.
2. Then click **File -> Save As**.



3. Select **Raspberry Pi Pico** in the pop-up window that appears.



4. Set the file name to `project_5_light.py`. Of course you can also use another name (except `main.py` and `boot.py`), fill it in and then click **OK** to confirm.

3. Import in other Projects

If you want to use this light effect in any of the *1. Don't Push Me*, *3. Objects Follow* and *4. Obstacle Avoid* projects, all you need to do is import the library you just saved and comment out the original `motors` library.

```
import project_5_light as car
# import motors as car
from servo import Servo
from grayscale import Grayscale
import time

# init grayscale module
gs = Grayscale(26, 27, 28)
gs.set_edge_reference(1000)

...
```

3.4 4. Play Pico 4WD with APP

This section will guide you to build remote control projects by using SunFounder Controller, which means you can use your phone/tablet to control your Pico 4WD car.

Here are some examples that will teach you how to Implement communication between Raspberry Pi Pico and SunFounder controller. As well as some interesting projects.

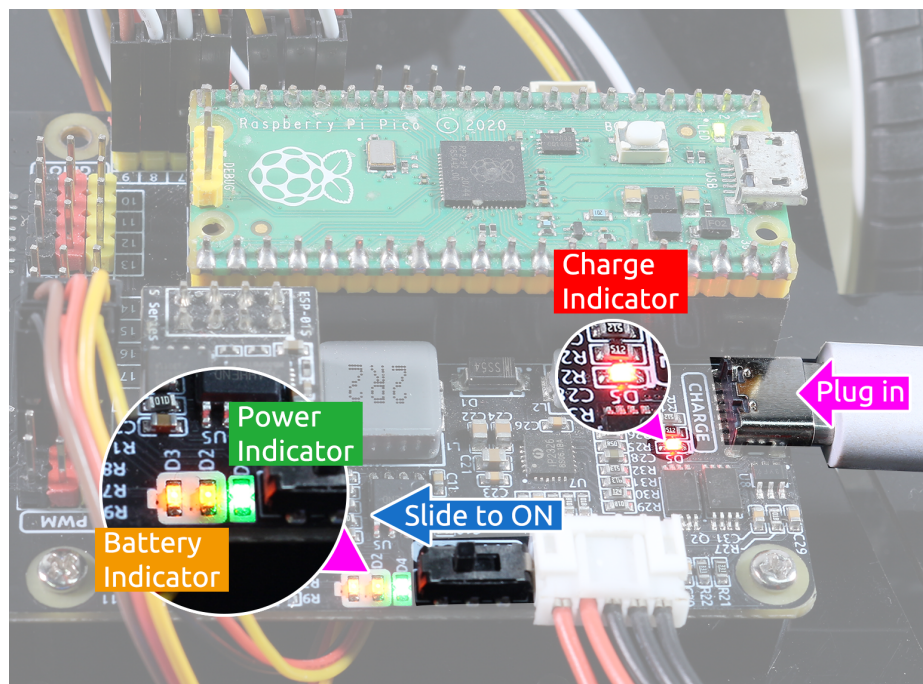
For more information on how each component works and the associated functions, check out [2. Learn Modules](#).

3.4.1 1. Basic Communication

If you have finished [2. Play Mode](#), you should know how to control Pico 4WD Car with SunFounder Controller. Here we will show you how they transfer data to each other.

Quick User Guide

1. Install [SunFounder Controller](#) from **APP Store(iOS)** or **Google Play(Android)**.
2. Run the `app_1_transfer.py` file under the `pico_4wd_car\examples\app_control` directory.
3. Let's start the Pico 4WD Car.
 - When first used or when the battery cable is unplugged, Pico RDP will activate its over-discharge protection circuitry(Unable to get power from battery).
 - Therefore, you'll need to plug in a Type-C cable for about 5 seconds to release the protection status.
 - At this time look at the battery indicators, if both battery indicators are off, please continue to plug in the Type-C cable to charge the battery.



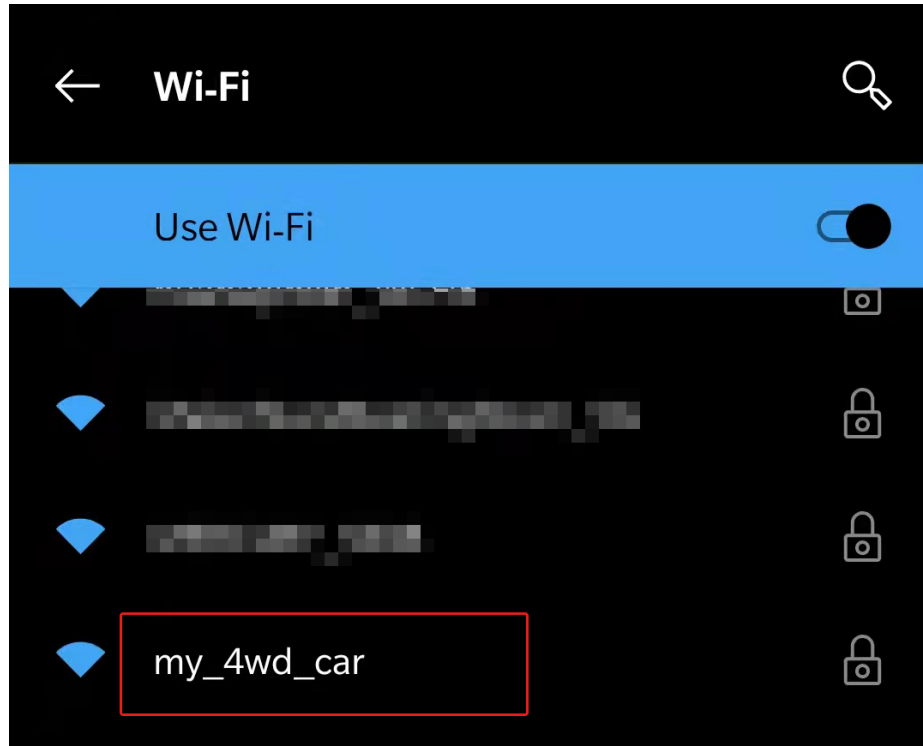
Note: Additionally, the LED on the ESP01S module will blink indicating that your mobile device is

not connected.

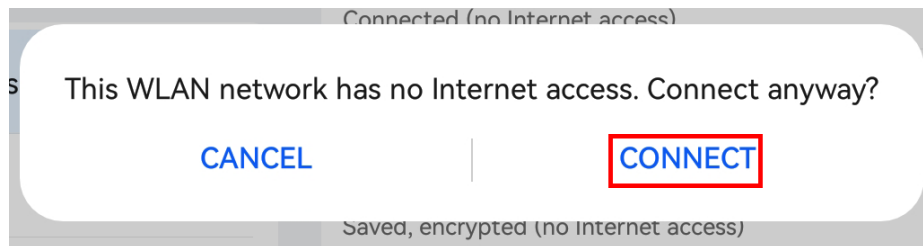
4. Connect to my_4wd_car LAN.

Your mobile device should now be connected to Pico 4WD Car's LAN so that they are on the same network.

- Find `my_4wd_car` on the WLAN of the mobile phone (tablet), enter the password 12345678 and connect to it.



- The default connection mode is AP mode. So after you connect, there will be a prompt telling you that there is no Internet access on this WLAN network, please choose to continue connecting.

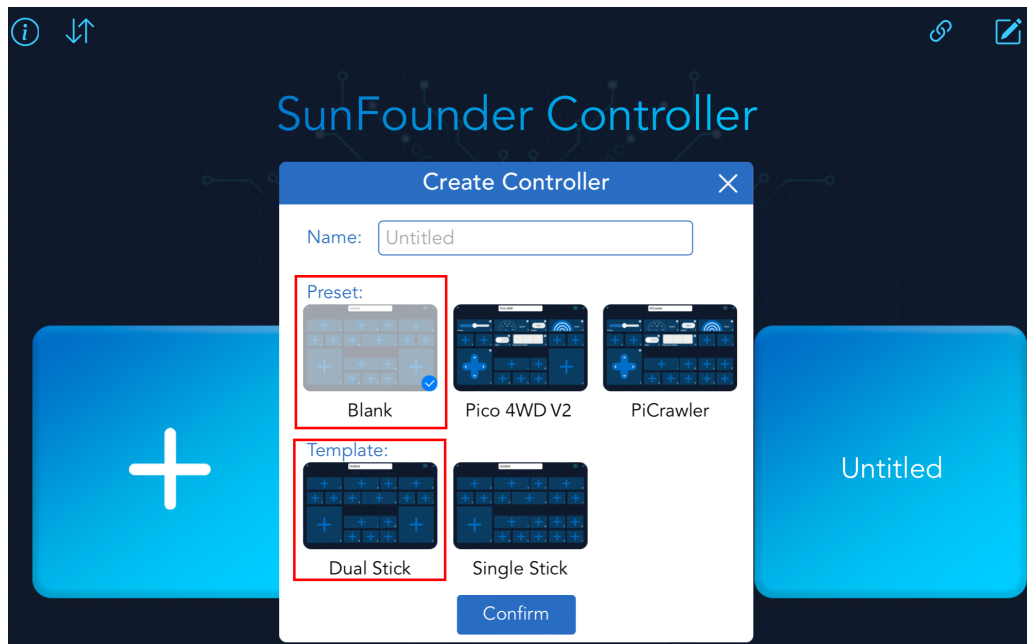


5. Create a controller.

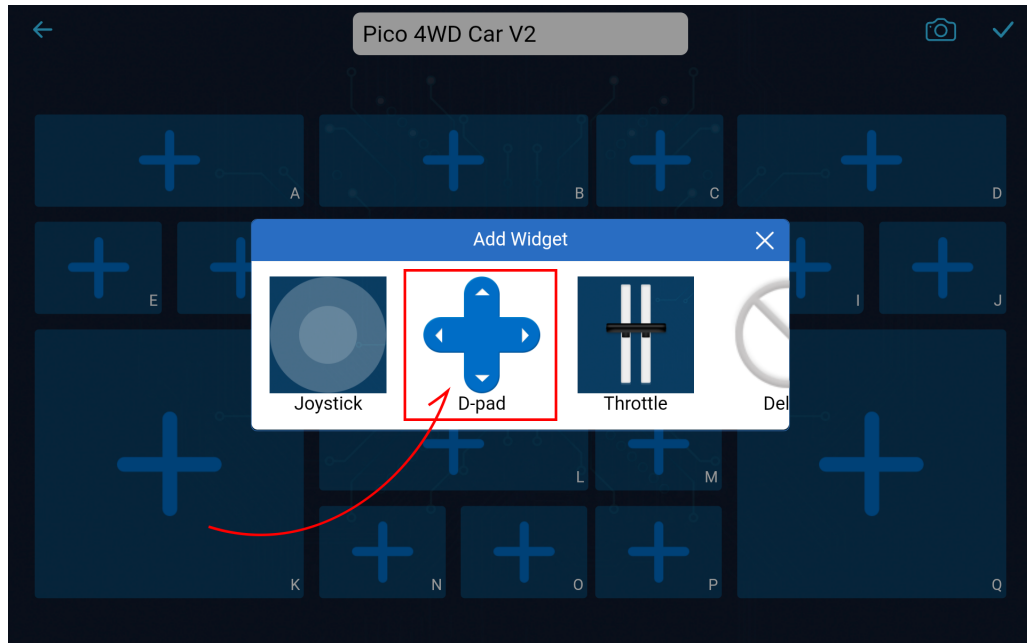
- Open SunFounder Controller and click on the + to create a new controller.



- Select the **Blank** and **Dual Stick** template, enter a name and click **Confirm**.



- You are now inside the controller. Click on the **K** area and select the **D-pad** widget.




- Then add a **Number** widget to the **J** area.




- Now you should see the interface like this.

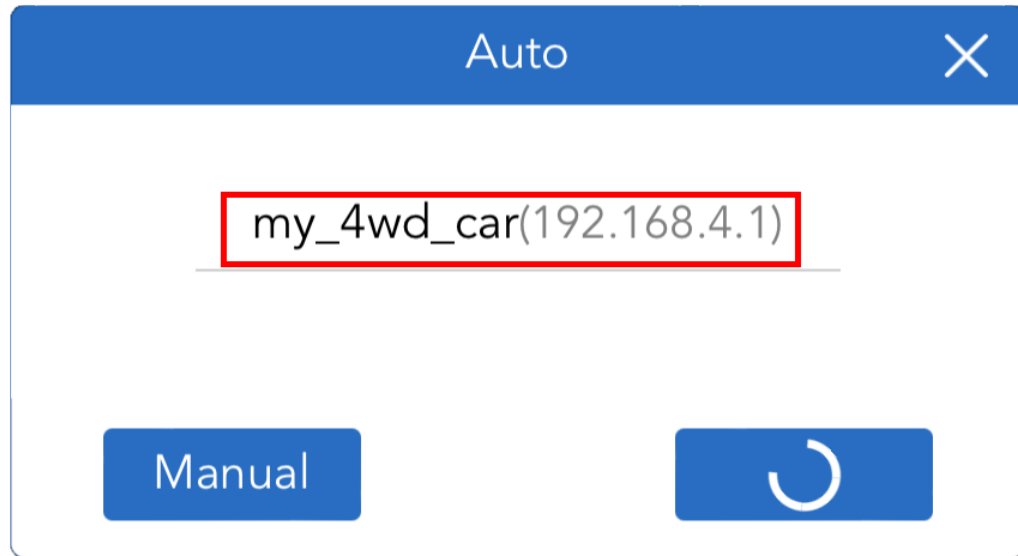


- Click the  button in the upper right corner.



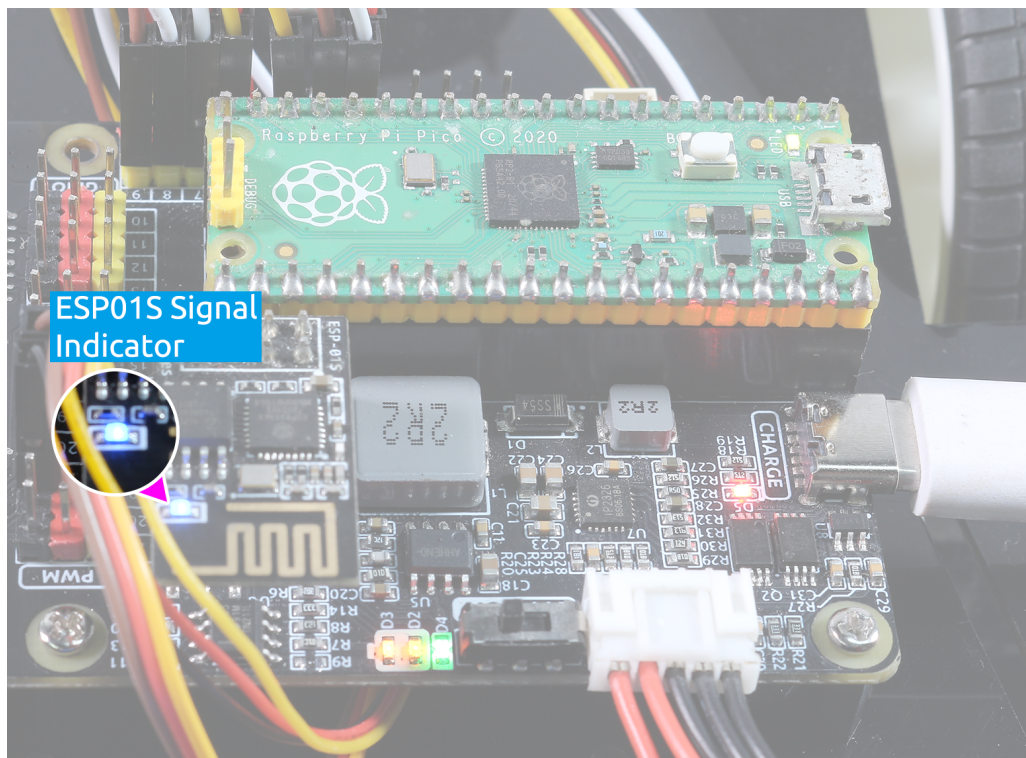
6. Connect and run the Controller.


- Now connect the SunFounder Controller to the Pico 4WD Car via the  button to start communication. Wait a few seconds and `my_4wd_car (IP)` will appear, click on it to connect.



Note: You need to make sure that your mobile device is connected to the my_4wd_car LAN, if you are not seeing the above message for a long time.

- After the “Connected Successfully” message appears and the product name will appear in the upper right corner.
- At the same time, the LED on the ESP01S module will stop flashing.



- After clicking the  button. The widget in the J area will show 34.67. The Shell in the Thonny IDE will return `stop`, and when you tap the D-Pad in the APP, it will return `forward`, `backward`, `left`,

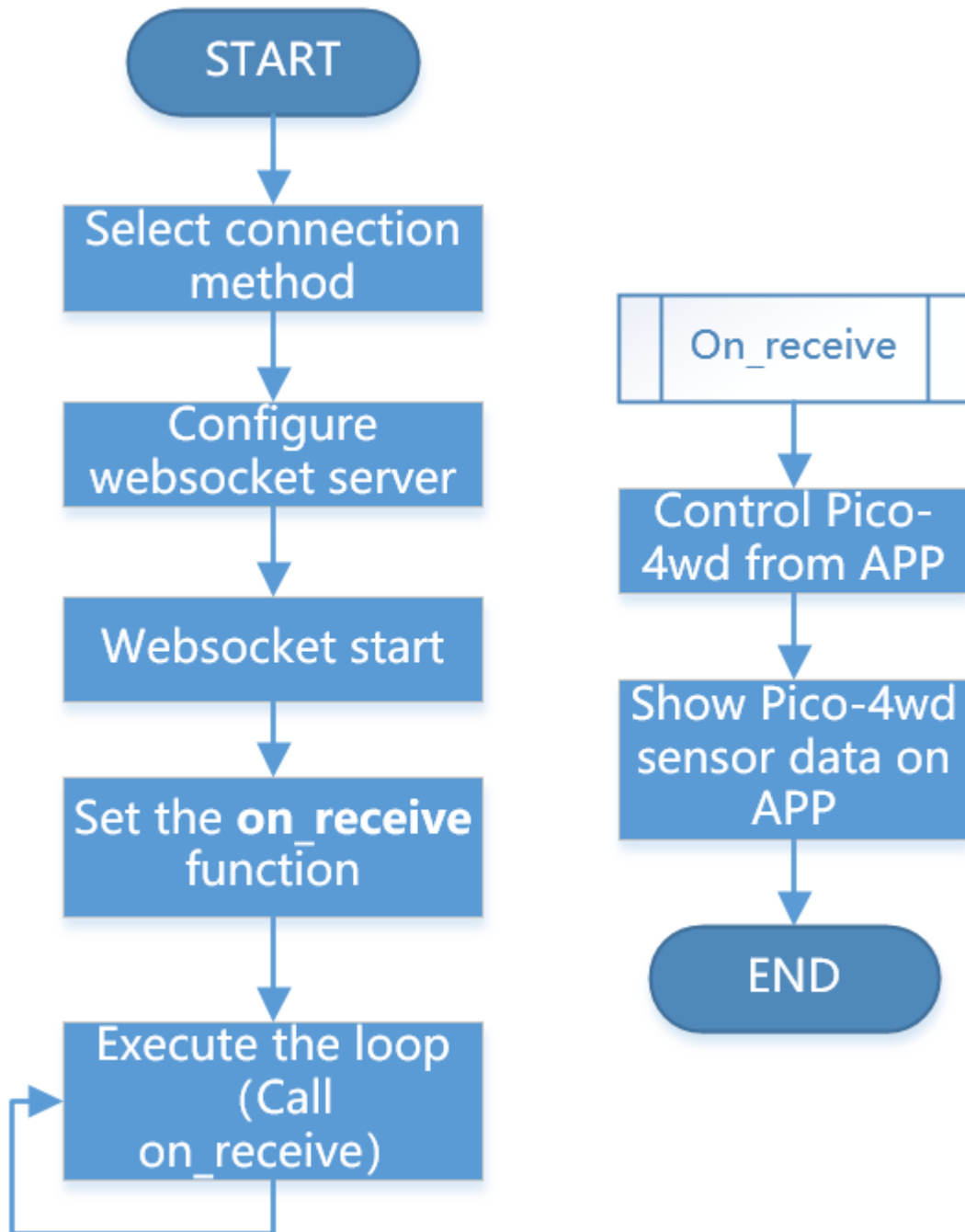
or right.

How it works?

The communication between Pico and Sunfounder Controller is based on the `websocket` protocol.

- [WebSocket - Wikipedia](#)

The specific workflow of APP Control gameplay is as follows:



```
import time
from ws import WS_Server
```

(continues on next page)

(continued from previous page)

```

from machine import Pin

'''Set name'''
NAME = 'my_4wd_car'

'''Configure wifi'''
# AP Mode
WIFI_MODE = "ap"
SSID = "" # your wifi name, if blank, use the set name "NAME"
PASSWORD = "12345678" # your password

'''----- Instantiate -----'''
ws = WS_Server(name=NAME, mode=WIFI_MODE, ssid=SSID, password=PASSWORD)
onboard_led = Pin(25, Pin.OUT)

'''----- on_receive (ws.loop()) -----'''
def on_receive(data):

    ''' the data from APP to PICO '''
    #print("recv_data: %s"%data)

    ''' if not connected, skip & stop '''
    if not ws.is_connected():
        return

    if 'K' in data.keys():
        print(data['K'])

    ''' the data send to APP '''
    ws.send_dict['J'] = 34.67

'''----- main -----'''
try:
ws.on_receive = on_receive
if ws.start():
    onboard_led.on()
    while True:
        ws.loop()
except Exception as e:
    print(e)
finally:
    onboard_led.off()

```

This code constitutes the basic framework of APP control. Here, you need to pay attention to the following two parts:

1. Setup websocket

There are two connection mode between Sunfounder Controller and Pico: One is **AP** mode, the other is **STA** mode.

- **AP Mode:** You need to connect Sunfounder Contorllr to the hotspot released by pico.
- **STA Mode:** You need to connect Sunfounder Controller and pico to the same WLAN.
- **AP Mode**

The default connection mode is **AP Mode**: The Pico releases the hotspot (the Wifi name is NAME in the code, here is my_4wd_car), the mobile phone (tablet) is connected to this LAN. This mode

allows you to remotely control pico in any situation, but will make your phone (tablet) temporarily unable to connect to the Internet.

```
from ws import WS_Server

'''Set name'''
NAME = 'my_4wd_car'

'''Configure wifi'''
# AP Mode
WIFI_MODE = "ap"
SSID = "" # your wifi name, if blank, use the set name "NAME"
PASSWORD = "12345678" # your password

# STA Mode
# WIFI_MODE = "sta"
# SSID = "<ssid>"
# PASSWORD = "<password>"

'''----- Instantiate -----'''
ws = WS_Server(name=NAME, mode=WIFI_MODE, ssid=SSID, password=PASSWORD)
```

• STA Mode

You can also use **STA** mode: Let the pico connects to your home WLAN, and your mobile phone (tablet) should also be connected to the same WLAN.

This mode is opposite to the **AP** mode and will not affect the normal use of the mobile phone (tablet), but will limit your pico from leaving the WLAN radiation range.

The way to start this mode is to comment out the three lines under **## AP Mode**, uncomment the three lines under **## STA Mode**, and change the SSID and PASSWORD to your home WIFI at the same time.

```
from ws import WS_Server

'''Set name'''
NAME = 'my_4wd_car'

'''Configure wifi'''
# AP Mode
# WIFI_MODE = "ap"
# SSID = "" # your wifi name, if blank, use the set name "NAME"
# PASSWORD = "12345678" # your password

# STA Mode
WIFI_MODE = "sta"
SSID = "<ssid>"
PASSWORD = "<password>"

'''----- Instantiate -----'''
ws = WS_Server(name=NAME, mode=WIFI_MODE, ssid=SSID, password=PASSWORD)
```

After completing the connection mode settings, Websocket will set up and start the server.

```
ws = WS_Server(name=NAME, mode=WIFI_MODE, ssid=SSID, password=PASSWORD)
```

2. Responding

The specific operation code of Pico and Sunfounder Controller is written on the `on_receive()` function. Usually, we need to write the codes for APP to control Pico on the front and the codes for APP to show Pico sensor data on the back.

```
def on_receive(data):

    ''' the data from APP to PICO '''
    #print("recv_data: %s"%data)

    ''' if not connected, skip & stop '''
    if not ws.is_connected():
        return

    if 'K' in data.keys():
        print(data['K'])

    ''' the data send to APP '''
    ws.send_dict['J'] = 34.67
```

Finally, `on_receive()` will be assigned to `ws.on_receive` and then called by `ws.loop`.

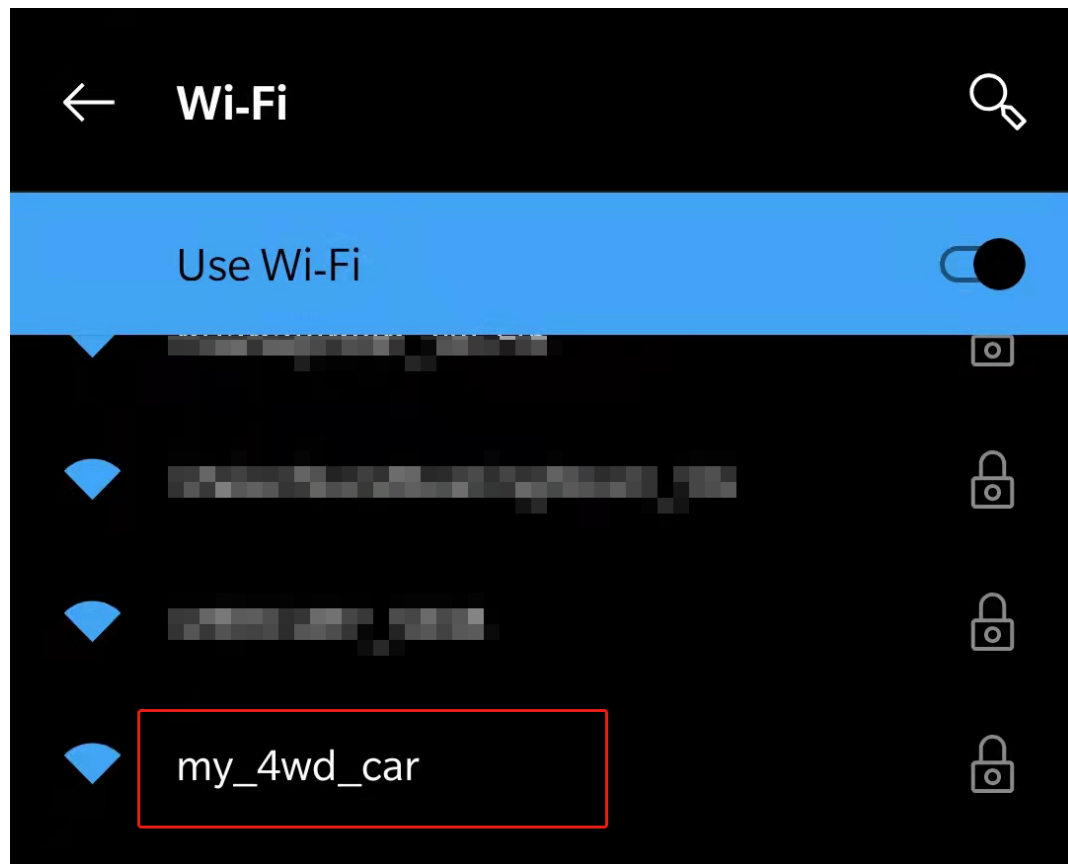
```
try:
ws.on_receive = on_receive
if ws.start():
    onboard_led.on()
    while True:
        ws.loop()
except Exception as e:
    print(e)
finally:
    onboard_led.off()
```

3.4.2 2. APP - Car Move

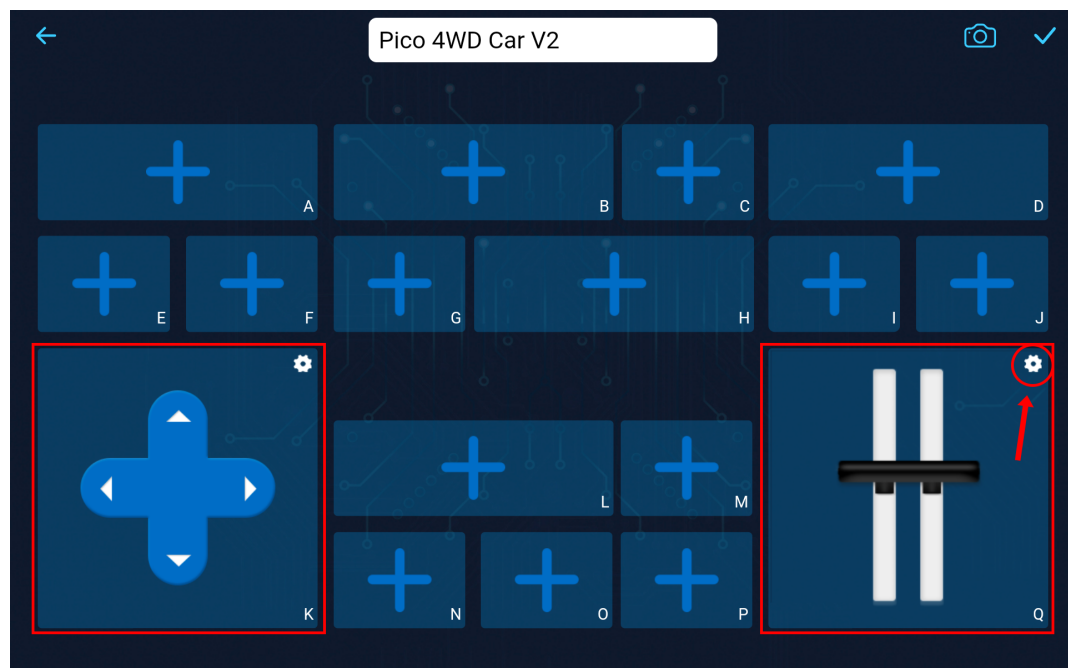
In this project, you will learn how to adjust the car's movement and speed using the APP.

Quick User Guide

1. Run the `app_2_move.py` file under the `pico_4wd_car\examples\app_control` path.
2. Then connect your device (phone/tablet) to `my_4wd_car`.



3. After opening the SunFounder controller, create a new controller and then add a **D-pad** widget to the **K** area and a **Throttle** widget to the **Q** area.



4. Click on the **set** button of the **Throttle** widget, change the maximum value to 100, the minimum value to 0, and the initial value to 20.



The image shows a 'Widget Configuration' dialog box with a blue header and a close button (X) in the top right corner. It contains five input fields arranged vertically, each with a label on the left and a value on the right. The first field is labeled 'Name' with the value 'Throttle'. The second field is labeled 'Minimum' with the value '0'. The third field is labeled 'Maximum' with the value '100'. The fourth field is labeled 'Initial' with the value '20'. The fifth field is labeled 'Auto reset' and has an unchecked checkbox. At the bottom center of the dialog is a blue button labeled 'Confirm'.

5. After saving (✓) and connecting (🔗) the controller, click (▶) to run it.
6. Now, you can use the D-Pad to control the movement of the car; the Throttle widget is used to adjust the power of the car (if it is 0, the car is not moving).

How it works?

This code can be seen in several steps.

1. Communication-related has been explained in the previous project, so we will skip it here.
2. Now let's see how to respond to the data transferred by the APP. In the `on_receive(data)` function, it responds to the data from the **Q** and **K** areas to change the values of `throttle_power` and `steer_power`, which together affect the movement and turning power of the car.

```
'''----- on_receive (ws.loop()) -----'''
def on_receive(data):
    global throttle_power, steer_power, dpad_touched

    ''' if not connected, skip & stop '''
    if not ws.is_connected():
        return

    # Move - power
```

(continues on next page)

(continued from previous page)

```

if 'Q' in data.keys() and isinstance(data['Q'], int):
    throttle_power = data['Q']
else:
    throttle_power = 0

# Move - direction
if 'K' in data.keys():
    #print(data['K'])
    if data['K'] == "left":
        dpad_touched = True
        if steer_power > 0:
            steer_power = 0
        steer_power -= int(throttle_power/2)
        if steer_power < -100:
            steer_power = -100
    elif data['K'] == "right":
        dpad_touched = True
        if steer_power < 0:
            steer_power = 0
        steer_power += int(throttle_power/2)
        if steer_power > 100:
            steer_power = 100
    elif data['K'] == "forward":
        dpad_touched = True
        steer_power = 0
    elif data['K'] == "backward":
        dpad_touched = True
        steer_power = 0
        throttle_power = -throttle_power
    else:
        dpad_touched = False
        steer_power = 0

```

- throttle_power: Used to adjust the moving power of the car.
- steer_power: For adjusting the turning power.
- dpad_touched: The default is False, True when receiving data from **K** area, thus making the car move.

3. Make the car move. The function my_car_move() is created to convert throttle_power and steer_power to left and right motors rotation power.

```

'''----- motors fuctions -----'''
def my_car_move(throttle_power, steer_power, gradually=False):
    power_l = 0
    power_r = 0

    if steer_power < 0:
        power_l = int((100 + 2*steer_sensitivity*steer_power)/
↪100*throttle_power)
        power_r = int(throttle_power)
    else:
        power_l = int(throttle_power)
        power_r = int((100 - 2*steer_sensitivity*steer_power)/
↪100*throttle_power)

```

(continues on next page)

(continued from previous page)

```

if gradually:
    car.set_motors_power_gradually([power_l, power_r, power_l, power_r])
else:
    car.set_motors_power([power_l, power_r, power_l, power_r])

```

4. Handler. The `remote_handler()` function is used to execute all the code related to the actual action of the car. The role in this project is to make the car move when the D-pad is tapped.

```

def remote_handler():
    global throttle_power, steer_power, dpad_touched

    if dpad_touched: # The car only moves when you press the K widget
        my_car_move(throttle_power, steer_power, gradually=True)

    ''' no operation '''
    if not dpad_touched:
        car.move('stop')

```

3.4.3 3. APP - SPEED

In this project, you can view the speed and mileage of the car's movement on the app.

Quick User Guide

1. Run the `app_3_speed.py` file under the `pico_4wd_car\examples\app_control` path.
2. Based on 2. APP - Car Move, add **Gauge** and **Number** widgets to the **B** and **C** areas as shown below.



3. After saving (✓) and connecting (🔗) the controller, click (▶) to run it.
4. Now, tap on the **D-Pad** to control the movement and adjust the power via the **Throttle** widget. You can also see the speed and mileage from the **B** and **C** areas.

How it works?

This project is generally the same as 2. *APP - Car Move*, except that two lines have been added to the `on_receive(data)` function to send the APP the speed and mileage data.

```
'''----- on_receive (ws.loop()) -----'''
def on_receive(data):
    global throttle_power, steer_power, move_status, dpad_touched

    ''' if not connected, skip & stop '''
    if not ws.is_connected():
        return

    # Speed measurement
    ws.send_dict['B'] = round(speed.get_speed(), 2) # uint: cm/s
    # Speed mileage
    ws.send_dict['C'] = speed.get_mileage() # unit: meter

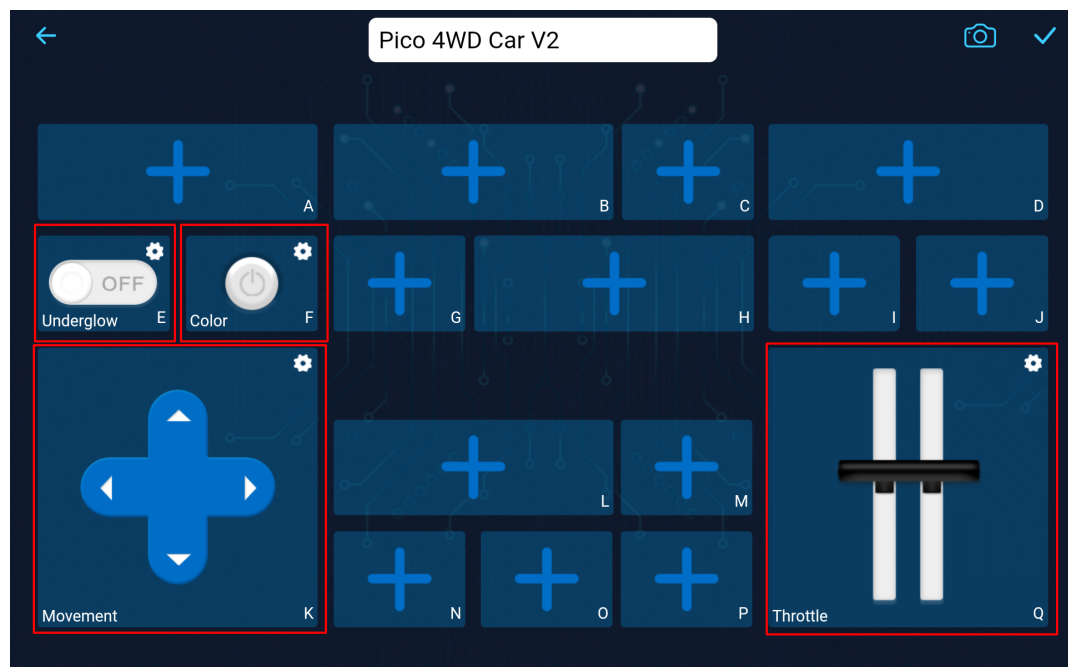
    # .....
```

3.4.4 4. APP - LIGHT

Now to control the RGB boards of the Pico 4WD car.

Quick User Guide

1. Run the `app_4_light.py` file under the `pico_4wd_car\examples\app_control` path.
2. Based on 2. *APP - Car Move*, add **Button** and **Switch** widgets to the **E** and **F** areas as shown below.



3. After saving (✓) and connecting (🔗) the controller, click (▶) to run it.
4. When you use the **D-pad** widget to make the car move, the tail light will show the direction. The on/off and color of underglow can be controlled through the **Switch** and **Button** widgets.

How it works?

This project is based on [2. APP - Car Move](#) with the addition of lighting effects.

1. Here, a total of three lighting effects are added and called in `remote_handler()`.

```
def remote_handler():
    global throttle_power, steer_power, move_status, dpad_touched

    if dpad_touched:
        my_car_move(throttle_power, steer_power, gradually=True)

    ''' no operation '''
    if not dpad_touched:
        move_status = "stop"
        car.move('stop')

    # ''' Bottom Lights '''
    bottom_lights_handler()
    # ''' Singal lights '''
    singal_lights_handler()
    # ''' Brake lights '''
    brake_lights_handler()
```

- `bottom_lights_handler()`: Controls the on/off and colors of the two RGB boards at the bottom.
- `singal_lights_handler()`: Control the left and right RGB LEDs of the rear RGB board, for example, when the car turns left, make the left two LEDs of the rear RGB board light up, and the same for the turn right.
- `brake_lights_handler()`: Controls the tail RGB board to light up red in breathing mode when the car stops or brakes.

2. About `singal_lights_handler()` function.

- When left is received, make the left two LEDs of the rear RGB board light up orange (`singal_on_color`).
- When right is received, make the right two LEDs of the rear RGB board light up orange (`singal_on_color`).
- Otherwise, let the left and right LEDs are off (`0x000000`).

```
def singal_lights_handler():
    if move_status == 'left':
        lights.set_rear_left_color(singal_on_color)
        lights.set_rear_right_color(0x000000)
    elif move_status == 'right':
        lights.set_rear_left_color(0x000000)
        lights.set_rear_right_color(singal_on_color)
    else:
        lights.set_rear_left_color(0x000000)
        lights.set_rear_right_color(0x000000)
```

3. About `brake_lights_handler()` function.

When stop is received, let the RGB board on the tail light up red(`brake_on_color`) in breathing mode(The brightness slowly goes from bright to dark and dark to bright.).

```
def brakeLights_handler():
    global is_move_last, brake_light_status, brake_light_time, led_
    status, brake_light_brightness
    global brake_light_brightness, brake_light_brightness_flag

    if move_status == 'stop':
        if brake_light_brightness_flag == 1:
            brake_light_brightness += 5
            if brake_light_brightness > 255:
                brake_light_brightness = 255
                brake_light_brightness_flag = -1
        elif brake_light_brightness_flag == -1:
            brake_light_brightness -= 5
            if brake_light_brightness < 0:
                brake_light_brightness = 0
                brake_light_brightness_flag = 1
        brake_on_color = [brake_light_brightness, 0, 0]
        lights.set_rear_color(brake_on_color)
    else:
        if is_move_last:
            lights.set_rear_middle_color(0x000000)
        else:
            lights.set_rear_color(0x000000)
        is_move_last = True
        brake_light_brightness = 255
```

4. About bottom_lights_handler() function.

- The variable led_status is True when the widget in the E area is ON, which causes the bottom two RGB boards to light up in a specific color.
- This specific color is selected from the array (led_theme[]) by tapping on the widget in the F area.

```
def bottom_lights_handler():
    global led_status
    if led_status:
        color = list(led_theme[str(led_theme_code)])
    else:
        color = [0, 0, 0]
    lights.set_bottom_color(color)
```

5. And the on_receive(data) function also has some changes based on 2. APP - Car Move.

- When you tap the D-pad buttons, the return left, right, forward or backward will cause the variable move_status to change simultaneously.
- This will allow the 3 RGB boards to display different effects as the car move.

```
def on_receive(data):
    global throttle_power, steer_power, move_status, dpad_touched
    global led_status, led_theme_code, led_theme_sum

    ''' if not connected, skip & stop '''
    if not ws.is_connected():
        return

    # Move - power
```

(continues on next page)

(continued from previous page)

```

if 'Q' in data.keys() and isinstance(data['Q'], int):
    throttle_power = data['Q']
else:
    throttle_power = 0

# Move - direction
if 'K' in data.keys():
    #print(data['K'])
    if data['K'] == "left":
        dpad_touched = True
        move_status = 'left'
        if steer_power > 0:
            steer_power = 0
        steer_power -= int(throttle_power/2)
        if steer_power < -100:
            steer_power = -100
    elif data['K'] == "right":
        dpad_touched = True
        move_status = 'right'
        if steer_power < 0:
            steer_power = 0
        steer_power += int(throttle_power/2)
        if steer_power > 100:
            steer_power = 100
    elif data['K'] == "forward":
        dpad_touched = True
        move_status = 'forward'
        steer_power = 0
    elif data['K'] == "backward":
        dpad_touched = True
        move_status = 'backward'
        steer_power = 0
        throttle_power = -throttle_power
    else:
        dpad_touched = False
        move_status = 'stop'
        steer_power = 0

if throttle_power == 0:
    move_status = 'stop'

```

6. In addition, the `on_receive(data)` function also responds to widgets in the **E** and **F** areas.

- The widget in the **E** area is used to turn on/off the bottom RGB boards, while the widget in the **F** area is used to change colors.
- Assign the return value of the **E** area widget to the variable `led_status`.
- If `led_status` is `True` (the widget in the **E** area is toggled ON), then determine if the widget in the **F** area is tapped.
- If so, switch to the next color in the array `led_theme[]`.

```

def on_receive(data):
    global throttle_power, steer_power, move_status, dpad_touched
    global led_status, led_theme_code, led_theme_sum

    ''' if not connected, skip & stop '''

```

(continues on next page)

(continued from previous page)

```

if not ws.is_connected():
    return

...

# LEDs switch
if 'E' in data.keys():
    led_status = data['E']

if led_status:
    # LEDs color theme change
    if 'F' in data.keys() and data['F'] == True:
        led_theme_code = (led_theme_code + 1) % led_theme_sum
        print(f"set led theme color: {led_theme_code}, {led_
↵theme[str(led_theme_code)][0]}")

```

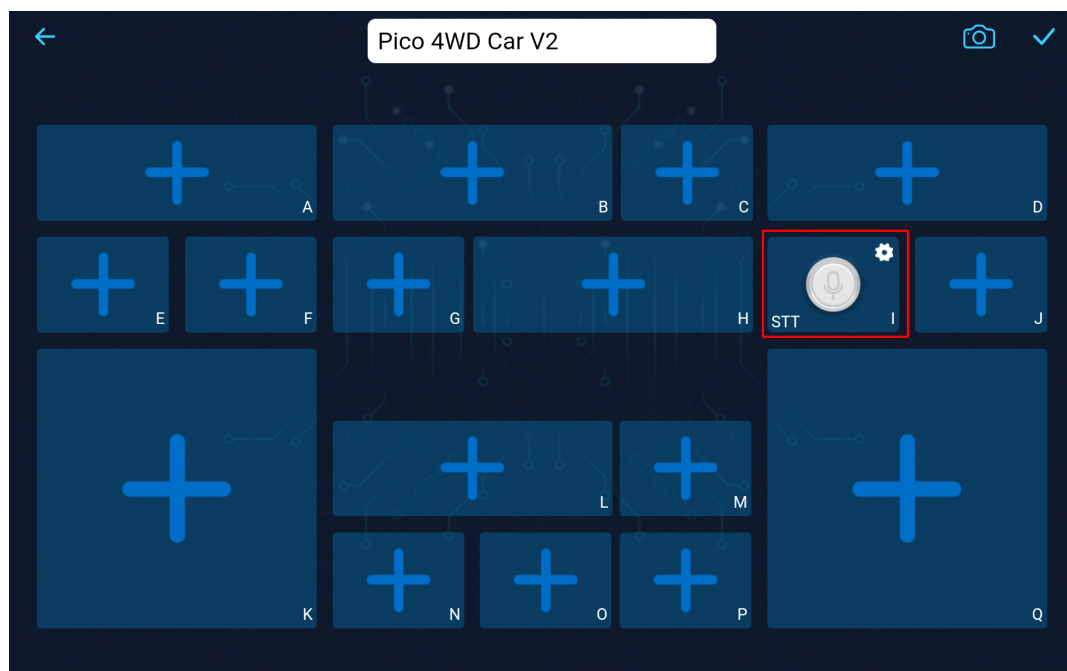
3.4.5 5. Speech

Warning: If you are using an Android device, then you need to change the AP mode in the script to STA mode and download the Google Voice Engine, for a detailed tutorial please refer to [Q4How can I use the STT mode on my Android device?](#).




The Pico 4WD Car can also be controlled using speech in SunFounder Controller. Pico 4WD Car will perform the set actions based on the commands you say to your mobile device.

Quick User Guide

1. Add **Button** widget to **I** area.



2. Run the `app_5_STT.py` file under the `pico_4wd_car\examples\app_control` path.

3. After saving () and connecting () the controller, click  to run it.
4. Now tap and hold the **Speech Control(I)** widget and say any of the following commands to see what happens.
 - stop: All movements of the car can be stopped.
 - forward: Let the car move forward.
 - backward: Let the car move backward.
 - left: Let the car turn left.
 - right: Let the car turn right.

How it works?

Speech to Text, is where the speech recognition engine on your mobile device recognizes your voice and converts it into text. When the data is transferred to the Pico 4WD car, it is already recognized text, such as forward and so on.

1. During speech recognition, close words may appear, e.g., I say forward and it recognizes as forwhat, so we define a dictionary to cope with this situation. You can add or modify this dictionary to make Pico 4WD respond to more of your commands.

```
'''----- Configure Voice Control Commands -----'''
voice_commands = {
    # action : [[command , similar commands], [run time(s)]]
    "forward": [["forward", "forwhat", "for what"], 3],
    "backward": [["backward"], 3],
    "left": [["left", "turn left"], 1],
    "right": [["right", "turn right", "while", "white"], 1],
    "stop": [["stop"], 1],
}
```

2. Recognize the voice command from APP in on_receive(data).

```
def on_receive(data):
    global current_voice_cmd, voice_start_time, voice_max_time

    ''' if not connected, skip & stop '''
    if not ws.is_connected():
        return

    # Voice control
    voice_text = None
    if 'I' in data.keys() and data['I'] != '':
        ws.send_dict['I'] = 1
        voice_text = data['I']
    else:
        ws.send_dict['I'] = 0

    if voice_text != None:
        print(f"voice_text: {voice_text}")
        for vcmd in voice_commands:
            if voice_text in voice_commands[vcmd][0]:
                print(f"voice control match: {vcmd}")
                current_voice_cmd = vcmd
                voice_max_time = voice_commands[vcmd][1]
                break
        else:
            print(f"voice control without match")
```

3. In `remote_handler()` function, the car moves according to the commands.

```
def remote_handler():
    global current_voice_cmd, voice_start_time, voice_max_time
    ''' Voice Control '''
    if current_voice_cmd != None :

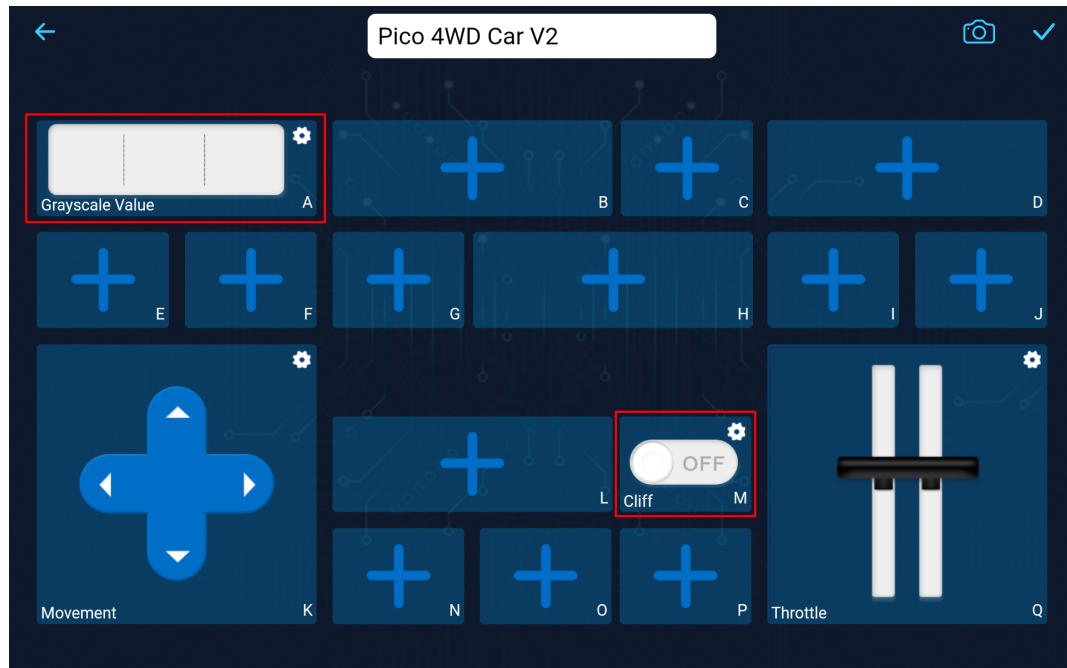
        if voice_max_time != 0:
            if voice_start_time == 0:
                voice_start_time = time.time()
            if ((time.time() - voice_start_time) < voice_max_time):
                if current_voice_cmd == "forward":
                    car.move("forward", VOICE_CONTROL_POWER)
                elif current_voice_cmd == "backward":
                    car.move("backward", VOICE_CONTROL_POWER)
                elif current_voice_cmd == "right":
                    car.move("right", VOICE_CONTROL_POWER)
                elif current_voice_cmd == "left":
                    car.move("left", VOICE_CONTROL_POWER)
                elif current_voice_cmd == "stop":
                    car.move("stop")
            else:
                current_voice_cmd = None
                voice_start_time = 0
                voice_max_time = 0
        else:
            car.move("stop")
```

3.4.6 6. Anti-Fall

Here, we added an anti-fall mode to the Pico 4WD car. If you move the Pico 4WD car to the edge of a table or stairs, it will stop moving. Unless you let it back up to a safe area, it will continue to move.

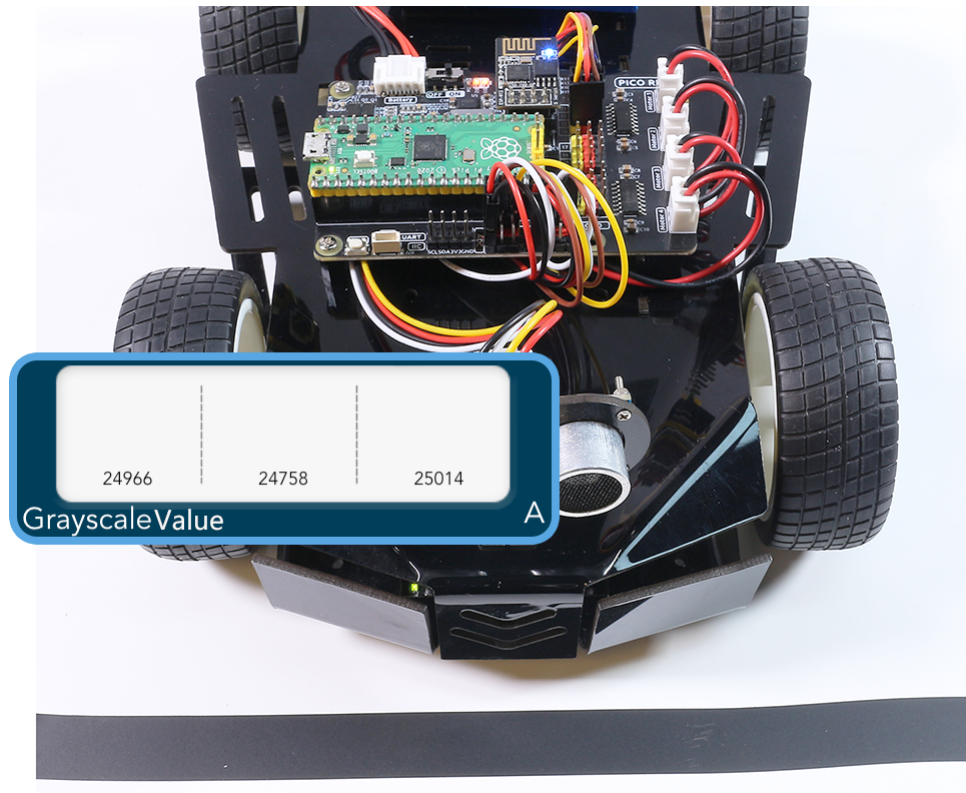
Quick User Guide

1. Run the `app_6_cliff.py` file under the `pico_4wd_car\examples\app_control` path and then power on the Pico 4WD car.
2. Based on [2. APP - Car Move](#), add **Grayscale Indicator** and **Switch** widgets to **A** and **M** areas as shown below.

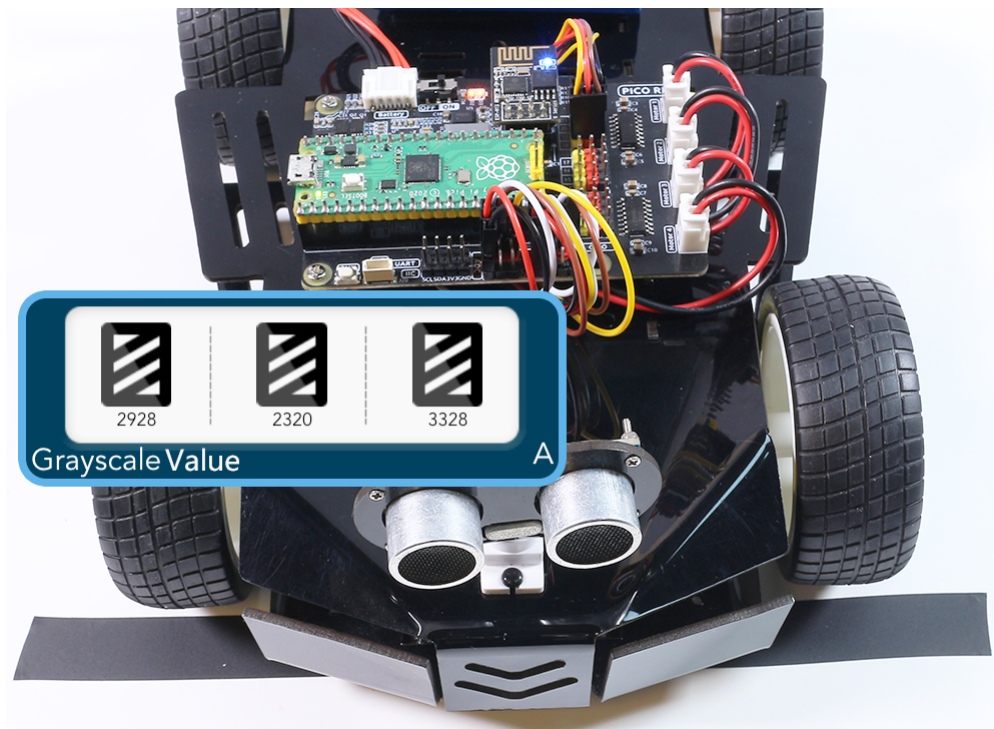


3. After saving (✓) and connecting (🔗) the controller, click (▶) to run it.
4. While this controller is running, **Grayscale Value(A)** will show the values of the three grayscale sensors in real time.
5. Place the grayscale module in three environments: white, black and hanging in the air (10cm or more) to see how the data in the changes.

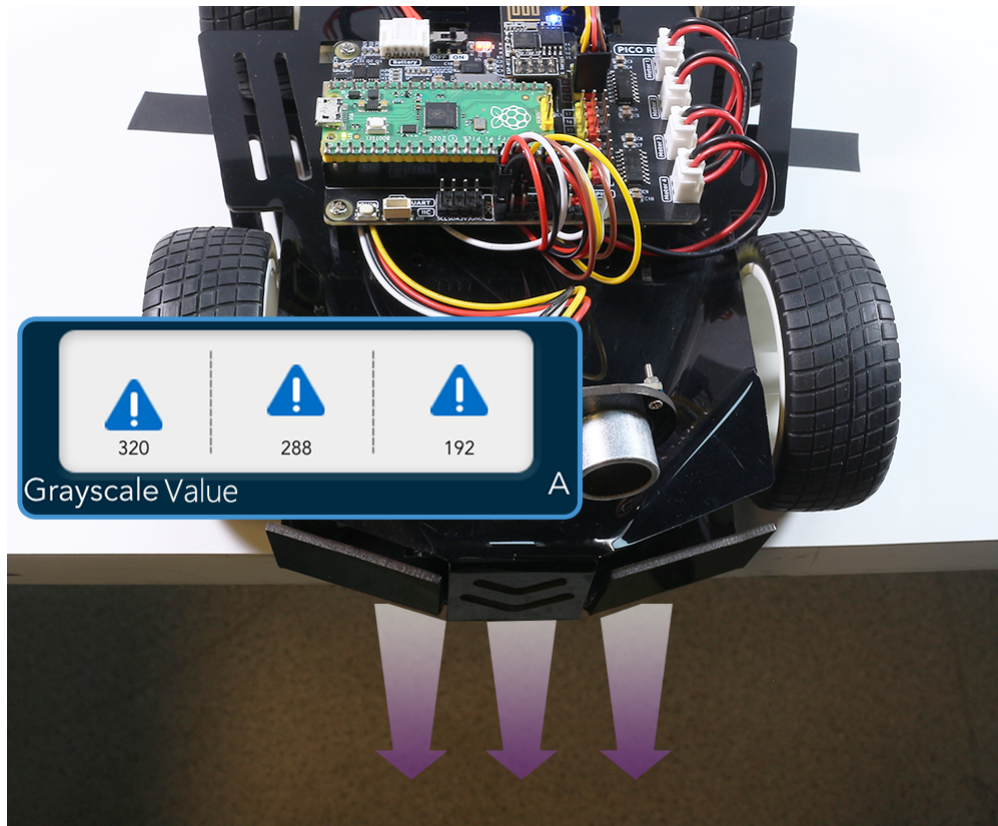
White surface You will find that the value of the white surface is generally large, for example mine is around 240,000.




Black line The value on the black line will be smaller, and now I'm at about 2000.

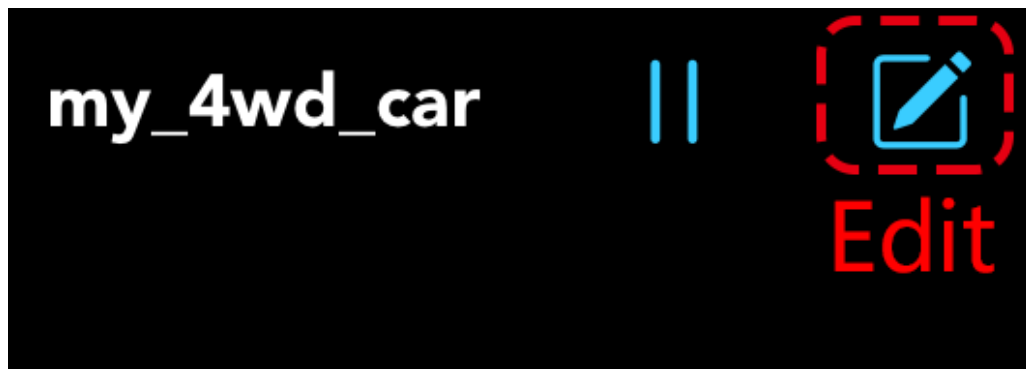


Overhang (10cm or more) And the value of the overhang will be even smaller, already less than 1000 in my environment.

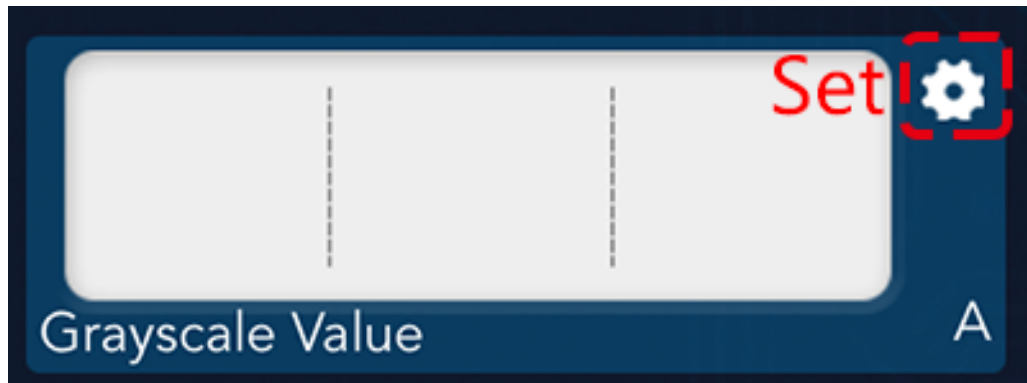


6. Set the threshold value.

- My car reads around 24000 in the white area and around 2000 in the black line, so I set `line_ref` to about the middle value of 10000.
- In the cliff area it reads less than 1000, so I set `cliff_ref` to 1000.
- Now click the  button to enter edit mode.



- Click on the **Settings** button in the upper right corner of the **Grayscale Value(A)** widget.



- Fill in the cliff and line thresholds.

7. Now re-save the SunFounder Controller and toggle the **Switch** widget to ON. If you move the Pico 4WD car to the edge of a table or stairs, it will stop moving. Unless you let it back up to a safe area, it will continue to move.

How it works?

1. This project is based on [2. APP - Car Move](#) and adds some responsiveness to the grayscale module, as reflected in the widgets in the **A** and **M** areas.
 - Send the grayscale value to area **A** for showing.
 - Then read the value of the widget in area **A**. If there are set thresholds, then use the set thresholds, otherwise use the default thresholds.
 - When the widget in area **M** is toggled to ON, the output value is `True` to let Pico 4WD car switch to the anti-fall mode.

```
def on_receive(data):
    global throttle_power, steer_power, dpad_touched
    global mode
```

(continues on next page)

(continued from previous page)

```

''' if not connected, skip & stop '''
if not ws.is_connected():
    return

''' remote control'''
# Move - power
...

# Move - direction
...

''' data to display'''
# grayscale
ws.send_dict['A'] = grayscale.get_value()

# grayscale reference
if 'A' in data.keys() and isinstance(data['A'], list):
    grayscale.set_edge_reference(data['A'][0])
    grayscale.set_line_reference(data['A'][1])
else:
    grayscale.set_edge_reference(GRAYSCALE_CLIFF_REFERENCE_DEFAULT)
    grayscale.set_line_reference(GRAYSCALE_LINE_REFERENCE_DEFAULT)

# mode select:
if 'M' in data.keys() and data['M'] == True:
    if mode != 'anti fall':
        mode = 'anti fall'
        print(f"change mode to: {mode}")
else:
    if mode != None:
        mode = None
        print(f"change mode to: {mode}")

```

2. Then, in the `remote_handler()` function, add some judgments.

- After switching to anti-fall mode, if the Pico 4WD car is at the edge of the table and stairs, it will stop and the D-pad can only control the car to back up.
- The car can only continue to move with D-pad control after backing up to a safe area.

```

def remote_handler():
    global throttle_power, steer_power, dpad_touched

    ''' move && anti-fall '''
    if mode == "anti fall":
        if grayscale.is_on_edge():
            if dpad_touched and throttle_power < 0: # only for backward
                my_car_move(throttle_power, steer_power, gradually=True)
            else:
                car.move("stop")
        else:
            if dpad_touched:
                my_car_move(throttle_power, steer_power, gradually=True)
            else:
                car.move("stop")
    elif dpad_touched:

```

(continues on next page)

(continued from previous page)

```

my_car_move(throttle_power, steer_power, gradually=True)

''' no operation '''
if not dpad_touched:
    car.move('stop')

```

3.4.7 7. Line Track

Let Pico 4wd walk on its exclusive avenue! Tape a line on a light-colored ground (or table) with black insulating tape. You will see Pico-4wd track the line to forward.

Warning: When pasting this line, there should be no sharp turns so that the car does not drive off the path.

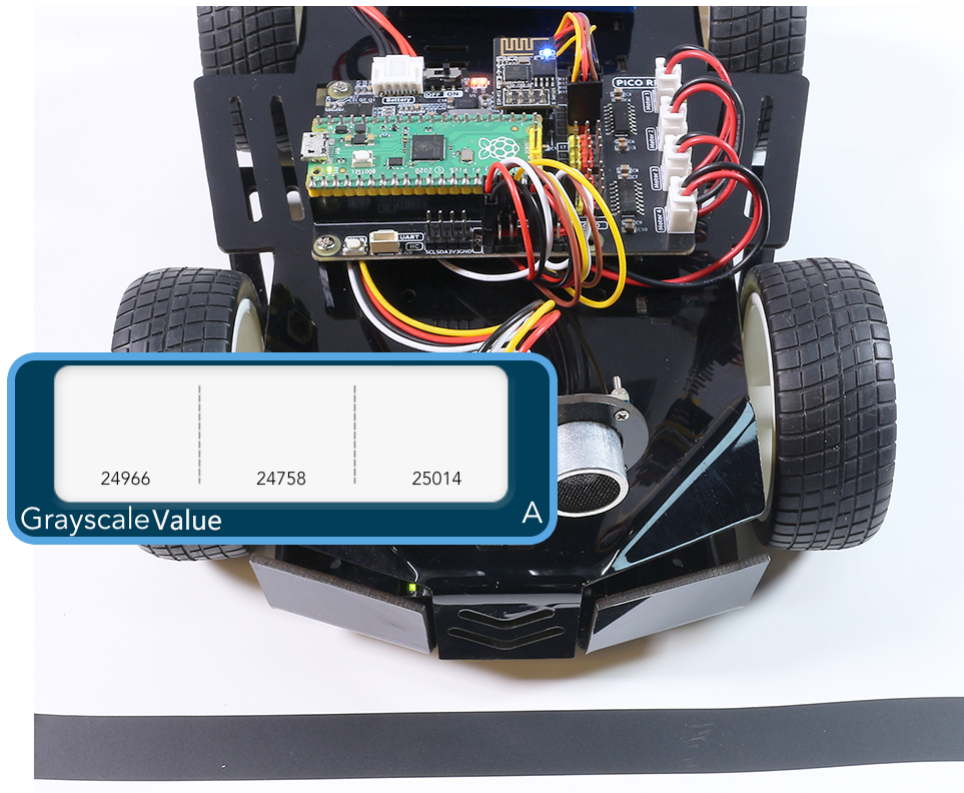
Quick User Guide

1. Run the `app_7_line_track.py` file under the `pico_4wd_car\examples\app_control` path and then power on the Pico 4WD car.
2. As shown below, create a controller that adds **Grayscale Indicator** and **Switch** widgets to **A** and **N** areas, respectively.

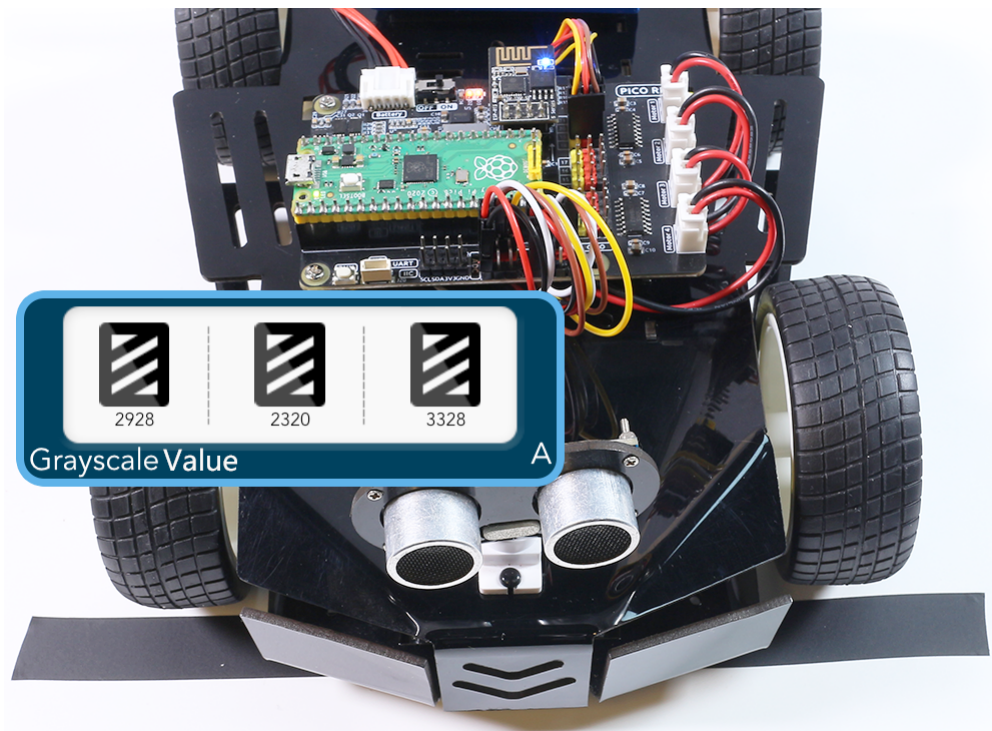


3. After saving (✓) and connecting (🔗) the controller, click (▶) to run it.
4. While this controller is running, **Grayscale Value(A)** will show the values of the three grayscale sensors in real time.
5. Place the grayscale module in three environments: white, black and hanging in the air (10cm or more) to see how the data in the changes.

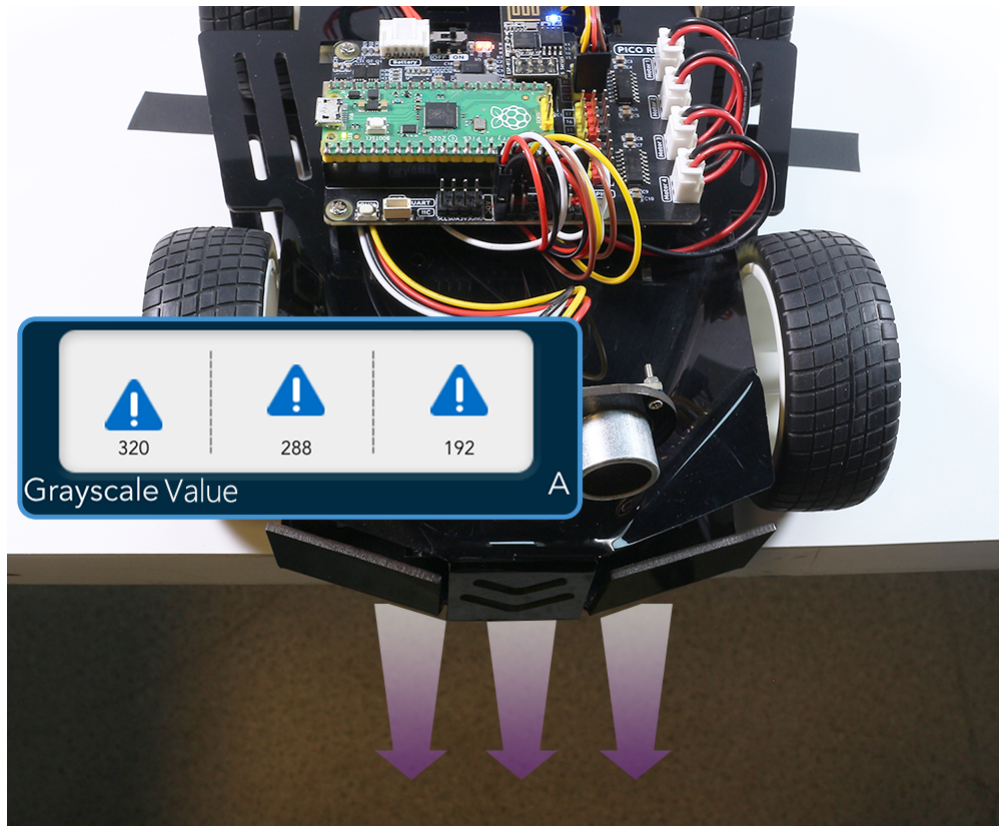
White surface You will find that the value of the white surface is generally large, for example mine is around 240,000.




Black line The value on the black line will be smaller, and now I'm at about 2000.

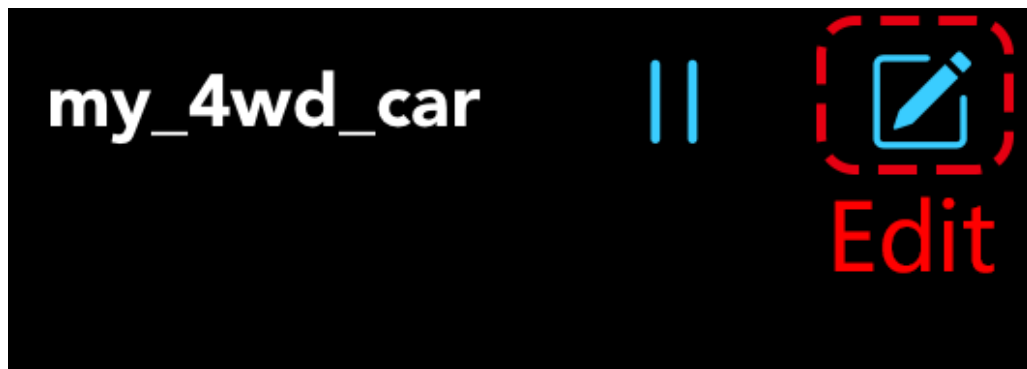


Overhang (10cm or more) And the value of the overhang will be even smaller, already less than 1000 in my environment.

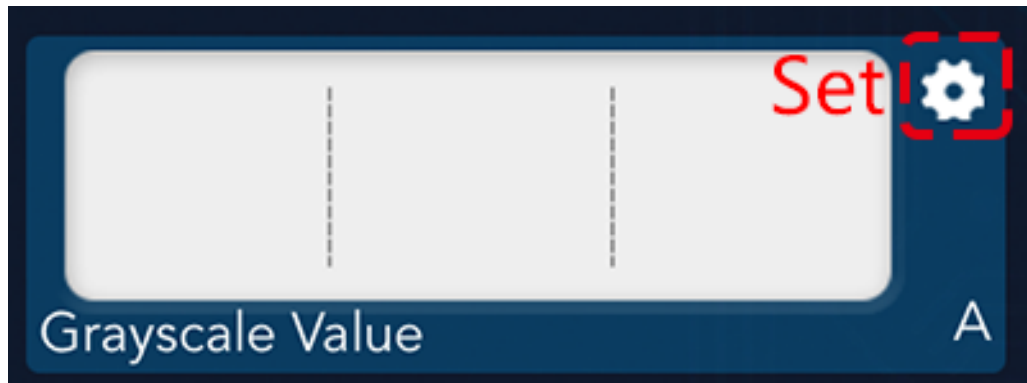


6. Set the threshold value.

- My car reads around 24000 in the white area and around 2000 in the black line, so I set `line_ref` to about the middle value of 10000.
- In the cliff area it reads less than 1000, so I set `cliff_ref` to 1000.
- Now click the  button to enter edit mode.



- Click on the **Settings** button in the upper right corner of the **Grayscale Value(A)** widget.



- Fill in the cliff and line thresholds.

7. Now re-save the SunFounder Controller and toggle the **Switch** widget to ON. Place the car on the black line and you will see the Pico-4wd tracking line advancing.

How it works?

1. In `on_receive(data)`, the grayscale values are sent to the **A** area for showing and responding to the data from the **N** area.
 - Send the grayscale value to area **A** for showing.
 - Then read the value of the widget in area **A**. If there are set thresholds, then use the set thresholds, otherwise use the default thresholds.
 - When the widget in area **N** is toggled to ON, the output value is `True` to let Pico 4WD car switch to the line track mode.

```
'''----- on_receive (ws.loop()) -----'''
def on_receive(data):
    global mode
```

(continues on next page)

(continued from previous page)

```

''' if not connected, skip & stop '''
if not ws.is_connected():
    return

''' data to display'''
# grayscale
ws.send_dict['A'] = grayscale.get_value()

# grayscale reference
if 'A' in data.keys() and isinstance(data['A'], list):
    grayscale.set_edge_reference(data['A'][0])
    grayscale.set_line_reference(data['A'][1])
else:
    grayscale.set_edge_reference(GRAYSCALE_CLIFF_REFERENCE_DEFAULT)
    grayscale.set_line_reference(GRAYSCALE_LINE_REFERENCE_DEFAULT)

# mode select:
if 'N' in data.keys() and data['N'] == True:
    if mode != 'line track':
        mode = 'line track'
        print(f"change mode to: {mode}")
    else:
        if mode != None:
            mode = None
            print(f"change mode to: {mode}")

```

2. Create a `line_track()` function to move the car in different directions based on the detection result of the grayscale module.

- When the detected grayscale value of the corresponding channel is less than set threshold, a 1 will be output, which means a black line is detected.
- Then all three sets of data ([0, 1, 0]) will be output by `get_line_status()`.
- Then make the car move in different directions according to the output data.

```

'''----- line_track -----'''
def line_track():
    global line_out_time
    _power = LINE_TRACK_POWER
    gs_data = grayscale.get_line_status()
    #print(f"gs_data: {gs_data}, {grayscale.line_ref}")

    if gs_data == [0, 0, 0] or gs_data == [1, 1, 1] or gs_data == [1, 0, 1]:
        if line_out_time == 0:
            line_out_time = time.time()
            if (time.time() - line_out_time > 2):
                car.move('stop')
                line_out_time = 0
            return
        else:
            line_out_time = 0

    if gs_data == [0, 1, 0]:
        car.set_motors_power([_power, _power, _power, _power]) # forward
    elif gs_data == [0, 1, 1]:

```

(continues on next page)

(continued from previous page)

```

        car.set_motors_power([_power, int(_power/5), _power, int(_power/
↵5)]) # right
        elif gs_data == [0, 0, 1]:
            car.set_motors_power([_power, int(-_power/2), _power, int(-_power/
↵2)]) # right plus
        elif gs_data == [1, 1, 0]:
            car.set_motors_power([int(_power/5), _power, int(_power/5), _
↵power]) # left
        elif gs_data == [1, 0, 0]:
            car.set_motors_power([int(-_power/2), _power, int(-_power/2), _
↵power]) # left plus

```

3. In `remote_handler()` function, the `line_track()` function will be called if the line tracking mode is turned on, otherwise the car is stopped.

```

def remote_handler():

    ''' move && anti-fall '''
    if mode == 'line track':
        line_track()

    ''' no operation '''
    if mode == None:
        car.move('stop')

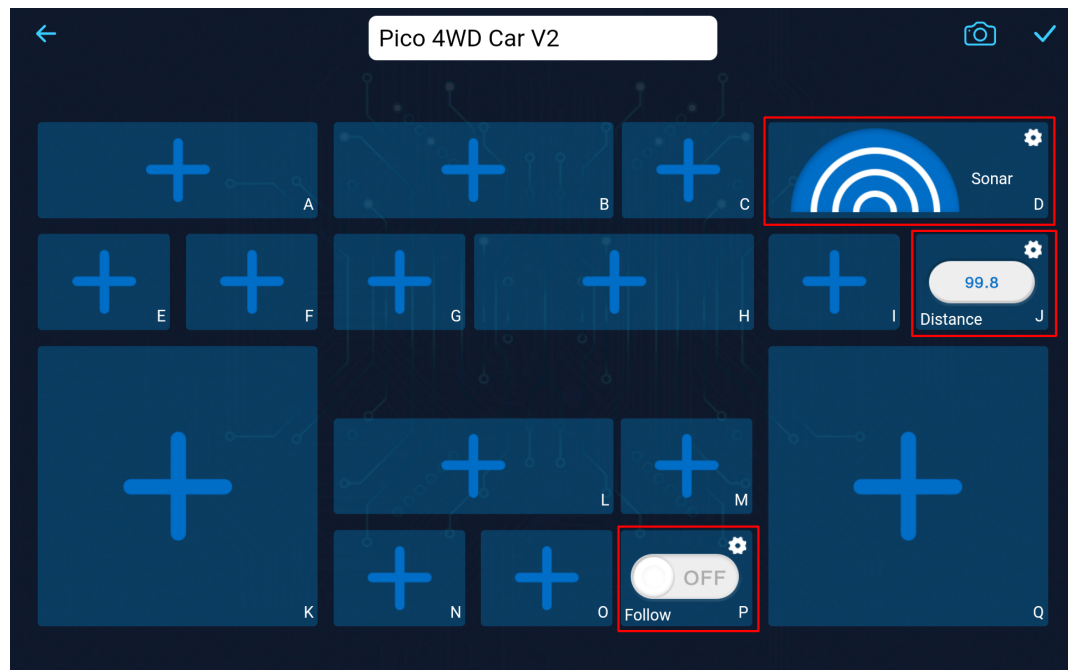
```

3.4.8 8. APP - Follow

In this project, Pico-4wd car will follow the object in front of it. For example, if you put your hand in front of Pico 4WD, it will rush towards your hand. And it will also play the advantage of sonar scanning, will judge your hand position and then adjust the forward direction.

Quick User Guide

1. Run the `app_9_avoid.py` file under the `pico_4wd_car\examples\app_control` path and then power on the Pico 4WD car.
2. As shown below, create a controller that adds **Radar**, **Number**, and **Button** widgets to the **D**, **J**, and **P** areas, respectively.



3. After saving (✓) and connecting (🔗) the controller, click (▶) to run it.
4. The car will follow your hand forward when you toggle the **Switch** widget in the **P** area to ON.

How it works?

1. In `on_receive(data)`, the distances and angles are sent to the **D** and **J** areas for showing, then responding to the data from the **P** area.

```
def on_receive(data):
    global mode

    ''' if not connected, skip & stop '''
    if not ws.is_connected():
        return

    ''' data to display'''
    # # sonar and distance
    ws.send_dict['D'] = [sonar_angle, sonar_distance]
    ws.send_dict['J'] = sonar_distance

    # mode select:
    if 'P' in data.keys() and data['P'] == True:
        if mode != 'follow':
            mode = 'follow'
            print(f"change mode to: {mode}")
    else:
        if mode != None:
            mode = None
            print(f"change mode to: {mode}")
```

2. Write the `follow()` and `get_dir()` functions to make the car go in a different direction based on the sonar detection.

Note: The explanation of the `get_dir()` function can be found in 3. *Objects Follow*.

```
'''----- get_dir (sonar sacn data to direction) -----'''
def get_dir(sonar_data, split_str="0"):
    # get scan status of 0, 1
    sonar_data = [str(i) for i in sonar_data]
    sonar_data = "".join(sonar_data)

    # Split 0, leaves the free path
    paths = sonar_data.split(split_str)

    # Calculate where is the widest
    max_paths = max(paths)
    if split_str == "0" and len(max_paths) < 4:
        return "left"
    elif split_str == "1" and len(max_paths) < 2:
        return "stop"

    # Calculate the direction of the widest
    pos = sonar_data.index(max_paths)
    pos += (len(max_paths) - 1) / 2
    delta = len(sonar_data) / 3
    if pos < delta:
        return "left"
    elif pos > 2 * delta:
        return "right"
    else:
        return "forward"

'''----- follow -----'''
def follow():
    global sonar_angle, sonar_distance

    sonar.set_sonar_scan_config(FOLLOW_SCAN_ANGLE, FOLLOW_SCAN_STEP)
    sonar.set_sonar_reference(FOLLOW_REFERENCE)

    #----- scan -----
    sonar_angle, sonar_distance, sonar_data = sonar.sonar_scan()
    # time.sleep(0.02)

    # If sonar data return a int, means scan not finished, and the int is
    ↪current angle status
    if isinstance(sonar_data, int):
        return

    #---- analysis direction ----
    direction = get_dir(sonar_data, split_str='1')

    #----- move -----
    if direction == "left":
        car.move("left", FOLLOW_TURNING_POWER)
    elif direction == "right":
        car.move("right", FOLLOW_TURNING_POWER)
    elif direction == "forward":
        car.move("forward", FOLLOW_FORWARD_POWER)
    else:
```

(continues on next page)

(continued from previous page)

```
car.move("stop")
```

3. In `remote_handler()` function, the `follow()` function will be called if the follow mode is turned on, otherwise the car is stopped.

```
def remote_handler():
    ''' follow hand '''
    if mode == 'follow':
        follow()

    ''' no operation '''
    if mode == None:
        car.move('stop')
```

3.4.9 9. APP - Avoid

Pico 4WD can avoid obstacles automatically!

When an obstacle is detected, instead of simply backing up, the sonar scans the surrounding area and finds the widest way to move forward.

Quick User Guide

1. Run the `app_8_follow.py` file under the `pico_4wd_car\examples\app_control` path and then power on the Pico 4WD car.
2. As shown below, create a controller that adds **Radar**, **Number**, and **Button** widgets to the **D**, **J**, and **O** areas, respectively.



3. After saving (✓) and connecting (🔗) the controller, click (▶) to run it.

4. As soon as you toggle the **Switch** widget in the **O** area to ON, the Pico 4WD car will move forward freely and change direction automatically when it encounters obstacles.

How it works?

1. In `on_receive(data)`, the distances and angles are sent to the **D** and **J** areas for showing, then responding to the data from the **P** area.

```
'''----- on_receive (ws.loop()) -----'''
def on_receive(data):
    global mode

    ''' if not connected, skip & stop '''
    if not ws.is_connected():
        return

    ''' data to display'''
    # # sonar and distance
    ws.send_dict['D'] = [sonar_angle, sonar_distance]
    ws.send_dict['J'] = sonar_distance

    # mode select:
    if 'O' in data.keys() and data['O'] == True:
        if mode != 'obstacle avoid':
            mode = 'obstacle avoid'
            sonar.set_sonar_reference(OBSTACLE_AVOID_REFERENCE)
            print(f"change mode to: {mode}")
        else:
            if mode != None:
                mode = None
                print(f"change mode to: {mode}")
```

2. Write the `obstacle_avoid()` and `get_dir()` functions to make the car go in a different direction based on the sonar detection.

Note: The explanation of the `get_dir()` function can be found in [3. Objects Follow](#).

```
'''----- get_dir (sonar sacn data to direction) -----'''
def get_dir(sonar_data, split_str="0"):
    # get scan status of 0, 1
    sonar_data = [str(i) for i in sonar_data]
    sonar_data = "".join(sonar_data)

    # Split 0, leaves the free path
    paths = sonar_data.split(split_str)

    # Calculate where is the widest
    max_paths = max(paths)
    if split_str == "0" and len(max_paths) < 4:
        return "left"
    elif split_str == "1" and len(max_paths) < 2:
        return "stop"

    # Calculate the direction of the widest
    pos = sonar_data.index(max_paths)
    pos += (len(max_paths) - 1) / 2
    delta = len(sonar_data) / 3
```

(continues on next page)

(continued from previous page)

```

if pos < delta:
    return "left"
elif pos > 2 * delta:
    return "right"
else:
    return "forward"

'''----- obstacle_avoid -----'''
def obstacle_avoid():
    global sonar_angle, sonar_distance, avoid_proc, avoid_has_obstacle

    # scan
    if avoid_proc == 'scan':
        if not avoid_has_obstacle:
            sonar.set_sonar_scan_config(OBSTACLE_AVOID_SCAN_ANGLE, OBSTACLE_AVOID_
↪SCAN_STEP)
            car.move('forward', OBSTACLE_AVOID_FORWARD_POWER)
        else:
            sonar.set_sonar_scan_config(180, OBSTACLE_AVOID_SCAN_STEP)
            car.move('stop')
            sonar_angle, sonar_distance, sonar_data = sonar.sonar_scan()
            if isinstance(sonar_data, int):
                # 0 means distance too close, 1 means distance safety
                if sonar_data == 0:
                    avoid_has_obstacle = True
                    return
                else:
                    return
            else:
                avoid_proc = 'getdir'

    # getdir
    if avoid_proc == 'getdir':
        avoid_proc = get_dir(sonar_data)
    # move: stop, forward
    if avoid_proc == 'stop':
        avoid_has_obstacle = True
        car.move('stop')
        avoid_proc = 'scan'
    elif avoid_proc == 'forward':
        avoid_has_obstacle = False
        car.move('forward', OBSTACLE_AVOID_FORWARD_POWER)
        avoid_proc = 'scan'
    elif avoid_proc == 'left' or avoid_proc == 'right':
        avoid_has_obstacle = True
        if avoid_proc == 'left':
            car.move('left', OBSTACLE_AVOID_TURNING_POWER)
            sonar_angle = 20 # servo turn right 20
        else:
            car.move('right', OBSTACLE_AVOID_TURNING_POWER)
            sonar_angle = -20 # servo turn left 20
        sonar.servo.set_angle(sonar_angle)
        time.sleep(0.2)
        avoid_proc = 'turn'

    # turn: left, right
    if avoid_proc == 'turn':

```

(continues on next page)

(continued from previous page)

```
sonar_distance = sonar.get_distance_at(sonar_angle)
status = sonar.get_sonar_status(sonar_distance)
if status == 1:
    avoid_has_obstacle = False
    avoid_proc = 'scan'
    car.move("forward", OBSTACLE_AVOID_FORWARD_POWER)
    sonar.servo.set_angle(0)
```

1. In `remote_handler()` function, the `obstacle_avoid()` function will be called if the obstacle avoid mode is turned on, otherwise the car is stopped.

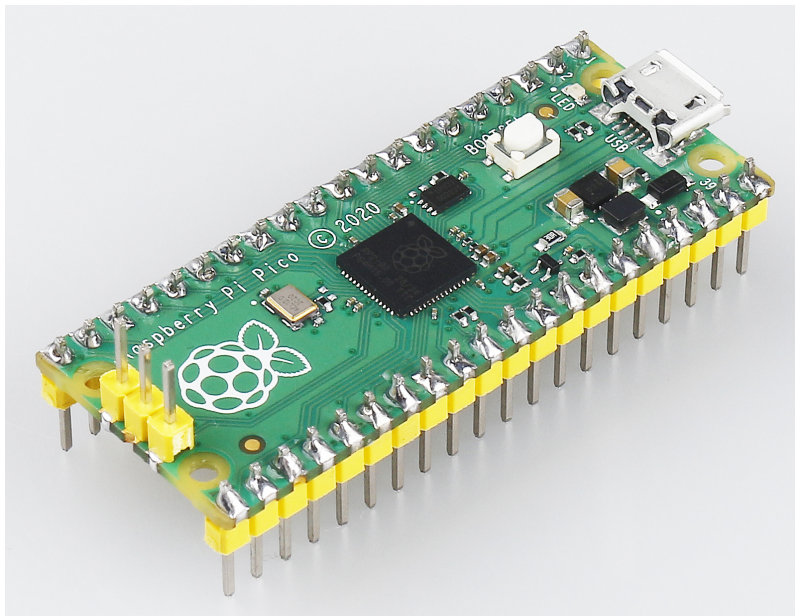
```
def remote_handler():

    ''' enable avoid function '''
    if mode == 'obstacle avoid':
        obstacle_avoid()

    ''' no operation '''
    if mode is None:
        car.move('stop')
```


4. APPENDIX

4.1 Introduction to Raspberry Pi Pico



The Raspberry Pi Pico is a microcontroller board based on the Raspberry Pi RP2040 microcontroller chip.

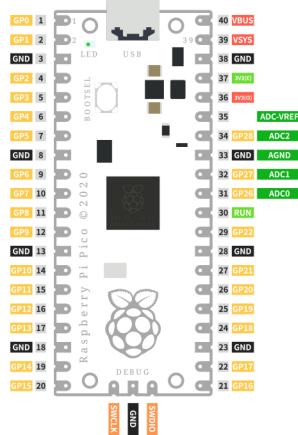
Whether you want to learn the MicroPython programming language, take the first step in physical computing, or want to build a hardware project, Raspberry Pi Pico – and its amazing community – will support you every step of the way. In the project, it can control anything, from LEDs and buttons to sensors, motors, and even other microcontrollers.

Features

- 21 mm × 51 mm form factor
- RP2040 microcontroller chip designed by Raspberry Pi in the UK
- Dual-core Arm Cortex-M0+ processor, flexible clock running up to 133 MHz
- 264KB on-chip SRAM
- 2MB on-board QSPI Flash
- 26 multifunction GPIO pins, including 3 analog inputs
- 2 × UART, 2 × SPI controllers, 2 × I2C controllers, 16 × PWM channels
- 1 × USB 1.1 controller and PHY, with host and device support

- 8 × Programmable I/O (PIO) state machines for custom peripheral support
- Supported input power 1.8–5.5V DC
- Operating temperature -20°C to +85°C
- Castellated module allows soldering direct to ier boards
- Drag-and-drop programming using mass storage over USB
- Low-power sleep and dormant modes
- Accurate on-chip clock
- Temperature sensor
- Accelerated integer and floating-point libraries on-chip

Pico's Pins



Name	Description	Function
GP0-GP28	General-purpose input/output pins	Act as either input or output and have no fixed purpose of their own
GND	0 volts ground	Several GND pins around Pico to make wiring easier.
RUN	Enables or disables your Pico	Start and stop your Pico from another microcontroller.
GPxx_ADCx	General-purpose input/output or analog input	Used as an analog input as well as a digital input or output – but not both at the same time.
ADC_VREF	Analog-to-digital converter (ADC) voltage reference	A special input pin which sets a reference voltage for any analog inputs.
AGND	Analog-to-digital converter (ADC) 0 volts ground	A special ground connection for use with the ADC_VREF pin.
3V3(O)	3.3 volts power	A source of 3.3V power, the same voltage your Pico runs at internally, generated from the VSYS input.
3v3(E)	Enables or disables the power	Switch on or off the 3V3(O) power, can also switches your Pico off.
VSYS	2-5 volts power	A pin directly connected to your Pico's internal power supply, which cannot be switched off without also switching Pico off.
VBUS	5 volts power	A source of 5V power taken from your Pico's micro USB port, and used to power hardware which needs more than 3.3V.

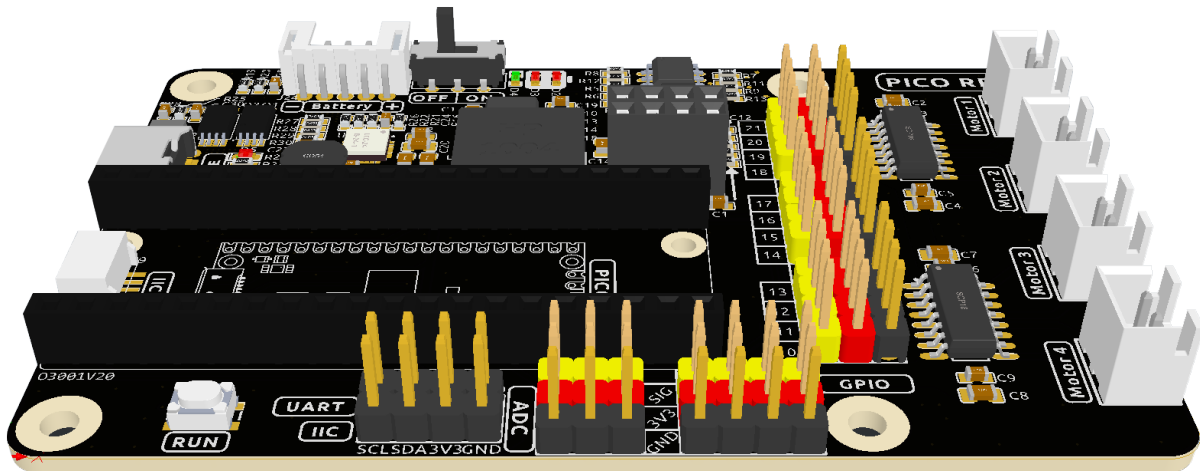
The best place to find everything you need to get started with your [Raspberry Pi Pico](#).

Or you can click on the links below:

- Raspberry Pi Pico product brief
- Raspberry Pi Pico datasheet
- Getting started with Raspberry Pi Pico: C/C++ development
- Raspberry Pi Pico C/C++ SDK
- API-level Doxygen documentation for the Raspberry Pi Pico C/C++ SDK
- Raspberry Pi Pico Python SDK
- Raspberry Pi RP2040 datasheet
- Hardware design with RP2040
- Raspberry Pi Pico design files
- Raspberry Pi Pico STEP file

4.2 Pico RDP

4.2.1 Introduction



The Pico Robotics Development Platform (RDP) is a Wi-Fi extension module for the Raspberry Pi Pico designed by SunFounder.

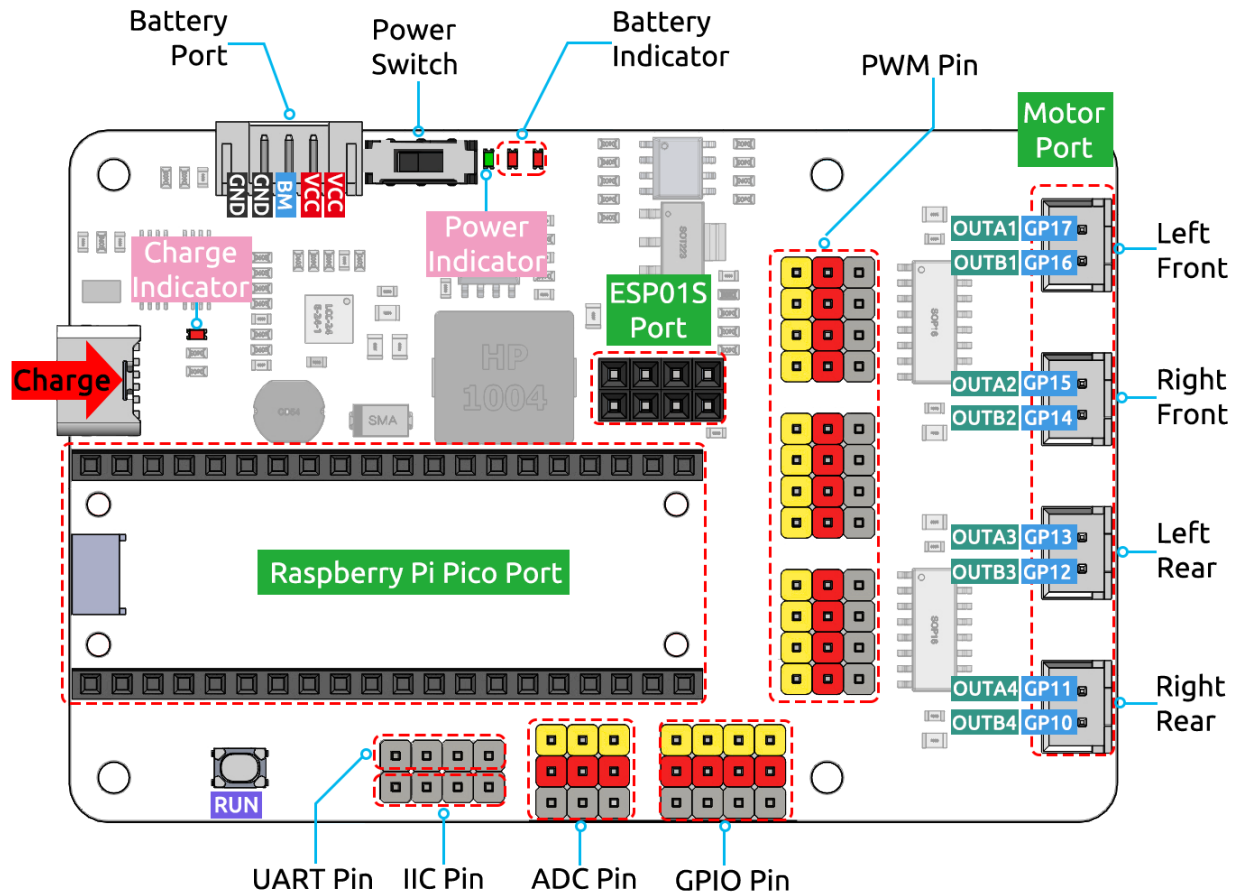
It uses the ESP01S module as a WiFi communication module for various remote control situations.

There are 12 channels of PWM pins, 3 channels of ADC pins, and 4 channels of GPIO pins reserved on this Pico RDP.

The Pico RDP also integrates two motor driver chips - TC1508S, which can drive 4 motors simultaneously.

Further, the Pico RDP has integrated battery charging circuitry, so you can plug in a 5V/2A USB-C cable directly to charge the included battery, which takes 130 minutes to fully charge.

4.2.2 Pico RDP Pinout

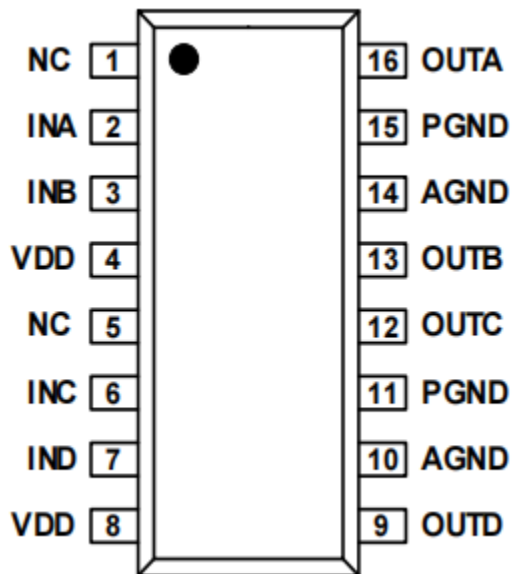
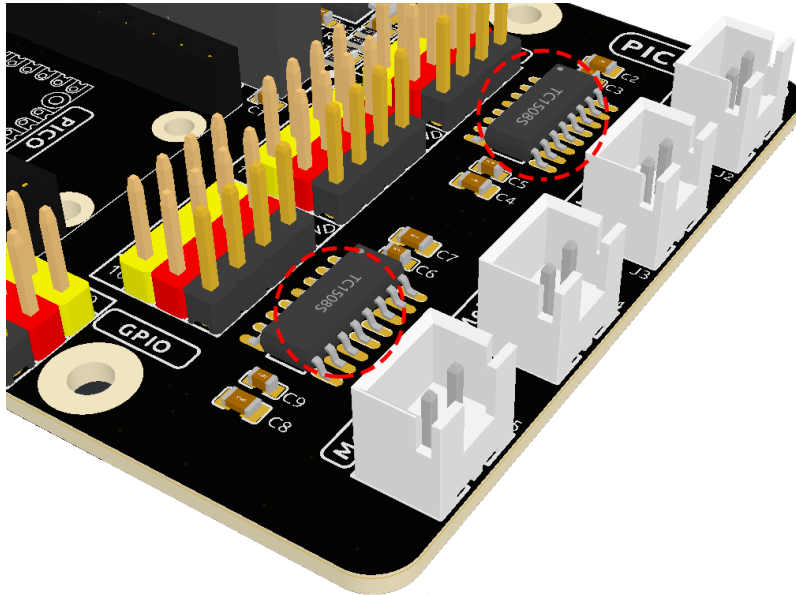


- **3 groups of indicators.**
 - **Charge Indicator:** This indicator lights up after plugging in the USC-C cable for charging.
 - **Power Indicator:** Turn the power switch to ON, the power indicator will light up.
 - **Battery Indicators:** Two indicators represent different power levels. When both battery indicators are off, you need to use the USB-C cable to charge. When charging, these two indicators will flash.
- **Battery Port**
 - 6.6V ~ 8.4V PH2.0-5P power input.
 - Powering the Pico RDP and Raspberry Pi Pico at the same time.
- **Power Switch**
 - Slide to ON to power on the Pico RDP.
- **Motor Port**
 - 4 groups of motor ports.
- **Charge Port**
 - After plugging into the 5V/2A USB-C cable, it can be used to charge the battery for 130min.
- **ESP01S Port**

- Used to plug in the ESP01S module.
- **Raspberry Pi Pico Port**
 - This connector is used to plug in the Raspberry Pi Pico, with the micro USB port facing outward.

4.2.3 Motor Port

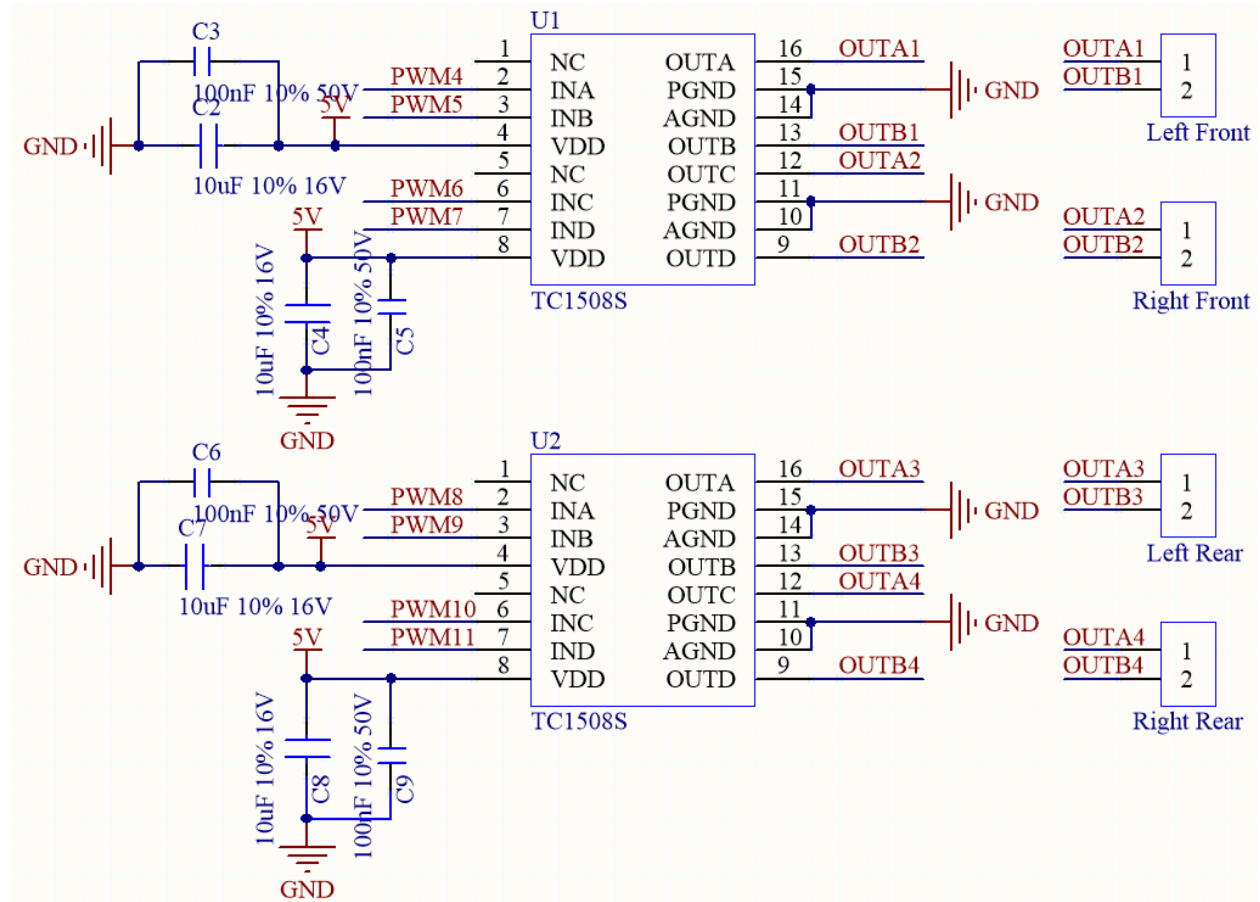
There are four motor connectors on the Pico RDP, controlled by two TC1508S motor driver chips, which operate over a wide voltage range of 2.2 to 5.5V, with a maximum continuous output current of 1.8A.

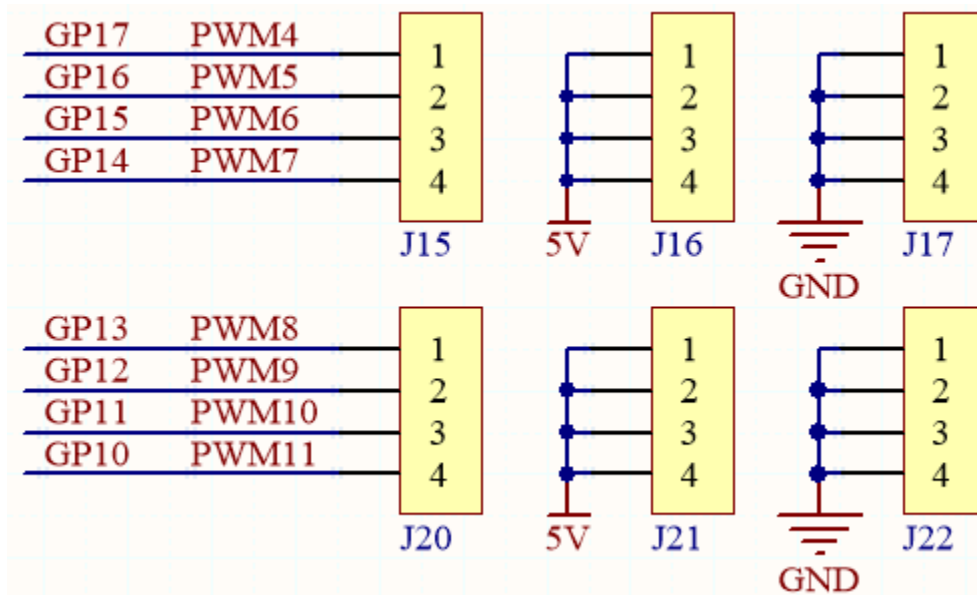


The following is a table of the input and output logic of the TC1508S chip.

Input				Output				Status
INA	INB	INC	IND	OUTA	OUTB	OUTC	OUTD	
L	L			Hi-Z	Hi-Z			Standby
H	L			H	L			Rotate
L	H			L	H			Reverse direction of rotation
H	H			L	L			Stop
		L	L			Hi-Z	Hi-Z	Standby
		H	L			H	L	Rotate
		L	H			L	H	Reverse direction of rotation
		H	H			L	L	Stop

Two TC1508S chips are used here to drive the four motors, and the corresponding schematic is shown below.





So the corresponding control pins of the 4 motor interfaces are shown below.

Raspberry Pi Pico	Pico RDP
GP17	OUTA1
GP16	OUTB1
GP15	OUTA2
GP14	OUTB2
GP13	OUTA3
GP12	OUTB3
GP11	OUTA4
GP10	OUTB4

4.3 Schematic and Structure Drawing

Schematic

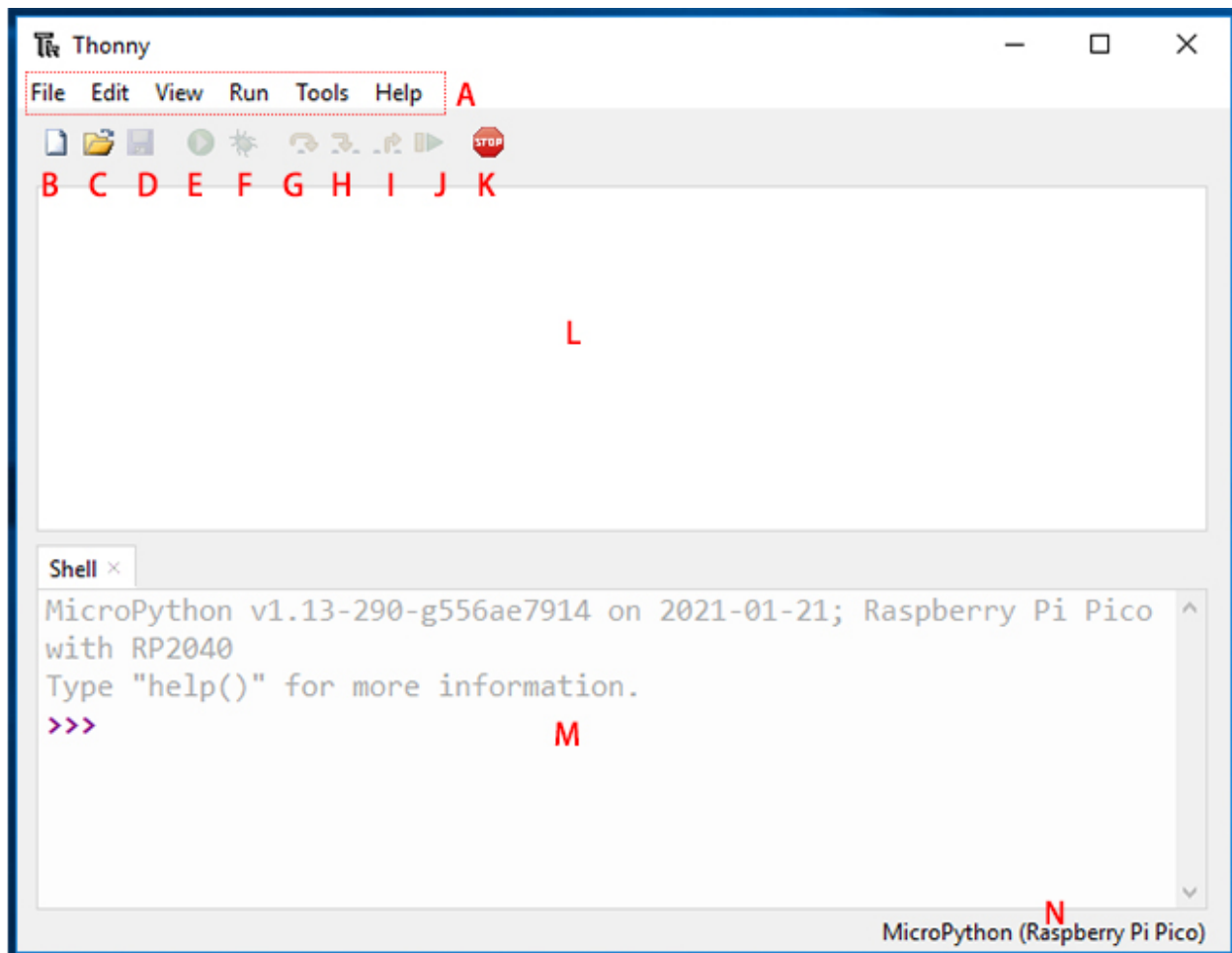
- Grayscale Module
- RGB Board
- Pico RDP
- Speed Module

Structure Drawing

- Lower Plate
- Upper Plate
- The Whole Car

4.4 Thonny IDE Introduction

- Ref:



- **A:** The menu bar that contains the file New, Save, Edit, View, Run, Debug, etc.
- **B:** This paper icon allows you to create a new file.
- **C:** The folder icon allows you to open files that already exist in your computer or Raspberry Pi Pico, if your Pico is already plugged into your computer.
- **D:** Click on the floppy disk icon to save the code. Similarly, you can choose whether to save the code to your computer or to the Raspberry Pi Pico.
- **E:** The play icon allows you to run the code. If you have not saved the code, save the code before it can run.
- **F:** The Debug icon allows you to debug your code. Inevitably, you will encounter errors when writing code. Errors can take many forms, sometimes using incorrect syntax, sometimes incorrect logic. Debugging is the tool for finding and investigating errors.

Note: The Debug tool cannot be used when MicroPython (Raspberry Pi Pico) is selected as the interpreter.

If you want to debug your code, you need to select the interpreter as the default interpreter and save as to your computer after debugging.

Finally, select the MicroPython (Raspberry Pi Pico) interpreter again, click the save as button, and re-save the debugged code to your Raspberry Pi Pico.

- The G, H and I arrow icons allow you to run the program step by step, but can only be started after clicking on the Debug icon. As you click on each arrow, you will notice that the yellow highlighted bar will indicate the line or section of Python that is currently evaluating.
 - **G**: Take a big step, which means jumping to the next line or block of code.
 - **H**: Take a small step means expressing each component in depth.
 - **I**: Exit out of the debugger.
- **J**: Click it to return from debug mode to play mode.
- **K**: Use the stop icon to stop running code.
- **L**: Script Area, where you can write your Python code.
- **M**: Python Shell, where you can type a single command, and when you press the Enter key, the single command will run and provide information about the running program. This is also known as REPL, which means “Read, Evaluate, Print, and Loop.”
- **N**: Interpreter, where the current version of Python used to run your program is displayed, can be changed manually to another version by clicking on it.

Note: NO MicroPython(Raspberry Pi Pico) Interpreter Option ?

- Check that your Pico is plugged into your computer via a USB cable.
 - The Raspberry Pi Pico interpreter is only available in version 3.3.3 or higher version of Thonny. If you are running an older version, please update.
-

4.5 18650 Battery



- **VCC**: Battery positive terminal, here there are two sets of VCC and GND is to increase the current and reduce the resistance.

- **Middle:** To balance the voltage between the two cells and thus protect the battery.
- **GND:** Negative battery terminal.

This is a custom battery pack made by SunFounder consisting of two 18650 batteries with a capacity of 2200mAh. The connector is PH2.0-5P, which can be charged directly after being inserted into the Pico RDP.

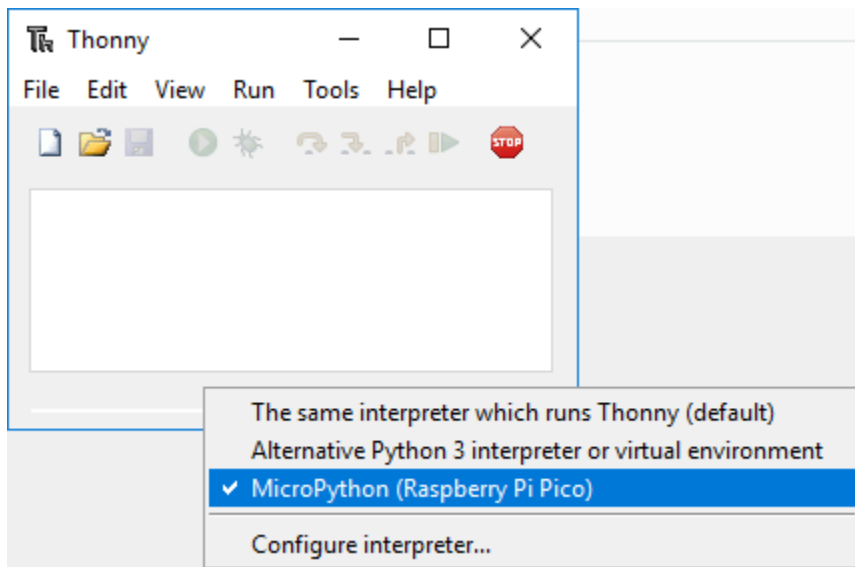
Pins Out

Features

- Battery charge: 5V/2A
- Battery output: 5V/5A
- Battery capacity: 3.7V 2000mAh x 2
- Battery life: 90min
- Battery charge time: 130min
- Connector: PH2.0, 5P

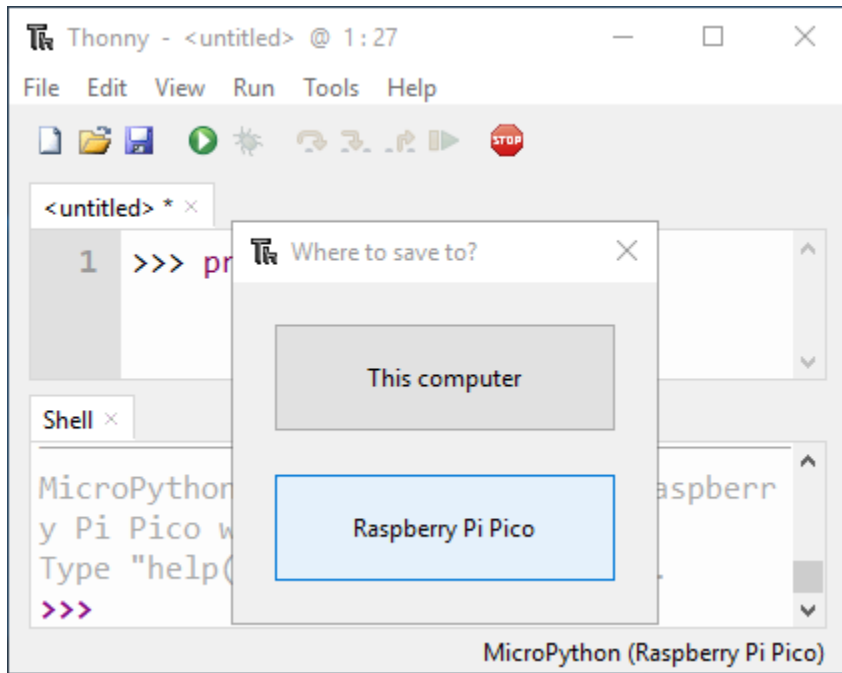
5. FAQ

5.1 Q1: NO MicroPython(Raspberry Pi Pico) Interpreter Option on Thonny IDE?



- Check that your Pico is plugged into your computer via a USB cable.
- Check that you have installed MicroPython for Pico (*2. Install MicroPython on Your Pico(Optional)*).
- The Raspberry Pi Pico interpreter is only available in version 3.3.3 or higher version of Thonny. If you are running an older version, please update (*1. Install Thonny IDE(Important)*).
- Plug in/out the micro USB cable several times.

5.2 Q2: Cannot open Pico code or save code to Pico via Thonny IDE?



- Check that your Pico is plugged into your computer via a USB cable.
- Check that you have selected the Interpreter as **MicroPython (Raspberry Pi Pico)**.

5.3 Q3: Unable to connect to COM51: Cannot configure port?

```
Unable to connect to COM51: Cannot configure port, something went wrong. Original_
↪message: PermissionError(13, 'A device attached to the system is not functioning.',
↪None, 31)

If you have serial connection to the device from another program, then disconnect it_
↪there first.

Process ended with exit code 1.
```

When you plug in your Raspberry Pi Pico to your computer, some times the above error message will appear, at this time you need to re-plug the USB cable and reopen the Thonny IDE.

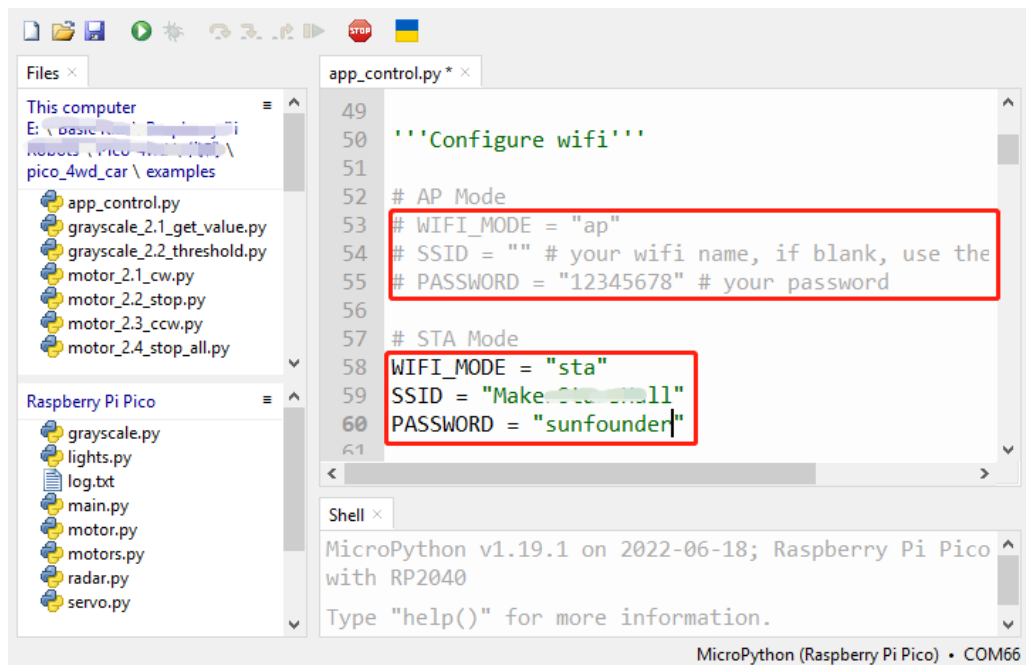
5.4 Q4How can I use the STT mode on my Android device?

Android devices cannot use the STT mode. Because the STT mode requires the Android mobile device to be connected to the Internet and to install the Google service component.

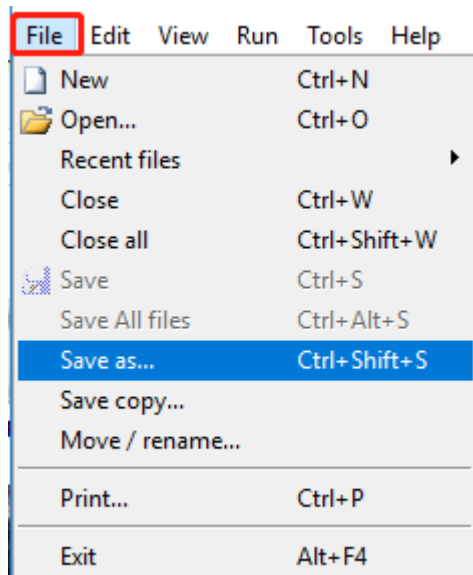
Now follow the steps below.

1. Modify the AP mode of `app_control.py` to STA mode.

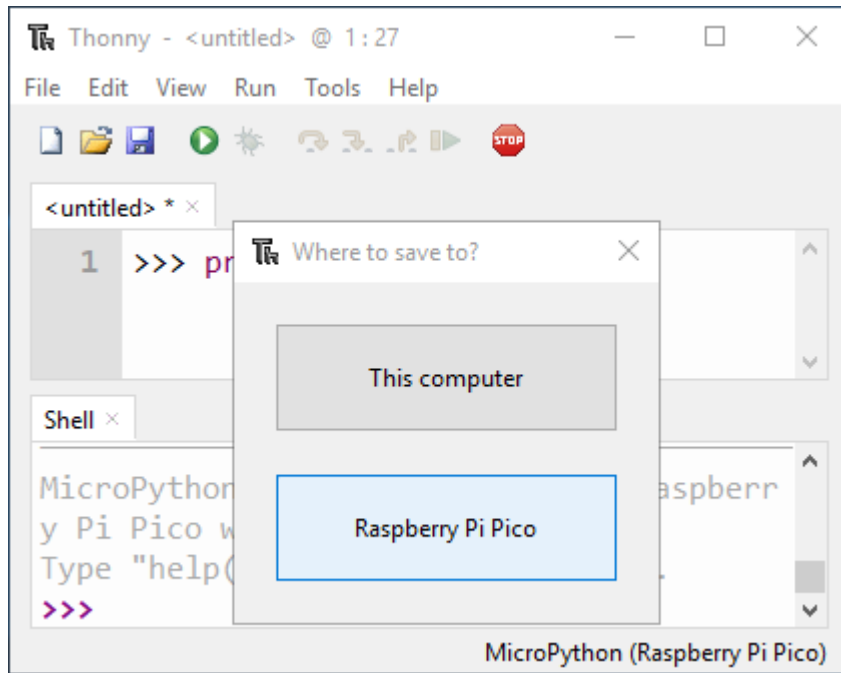
- Open the `app_control.py` under the path of `pico_4wd_car-v2.0\examples\learn_modules`.
- Then comment out the AP mode related code. Uncomment the STA mode related code and fill in the SSID and PASSWORD of your home Wi-Fi.



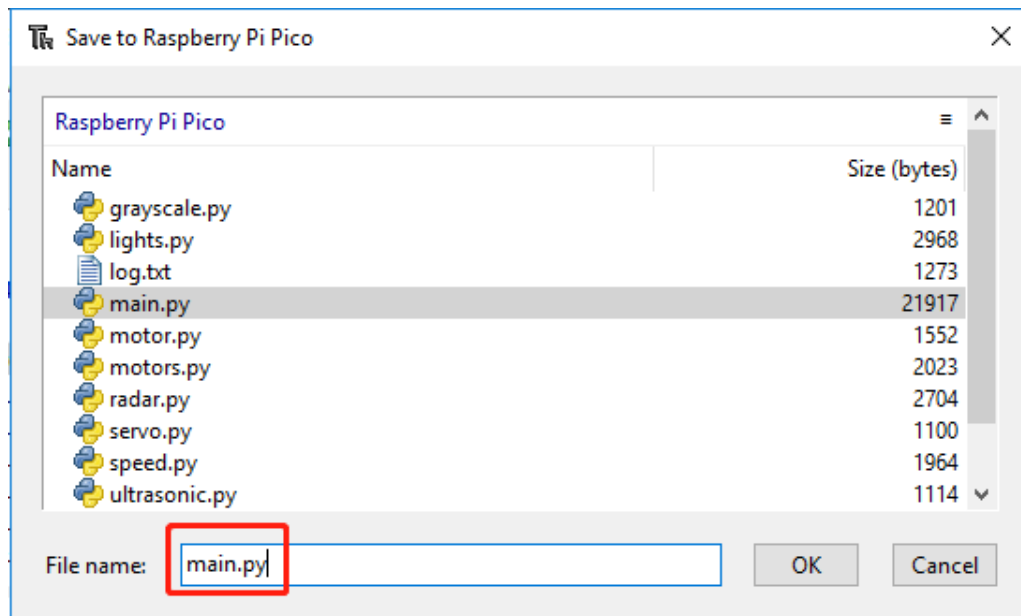
- Click **File** -> **Save as**.



- Select **Raspberry Pi Pico** in the pop-up window.




- After that, save `app_control.py` to Raspberry Pi Pico with a new name - `main.py`.



2. Search google in Google Play, find the app shown below and install it.

17:42

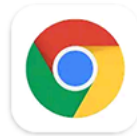
Bluetooth xiaoming_PC (25)

 google**Brave Private Web Browser**

Ad • Brave Software • Communication

*Block ads on every video*4.7 ★ 91 MB  100M+**Google**

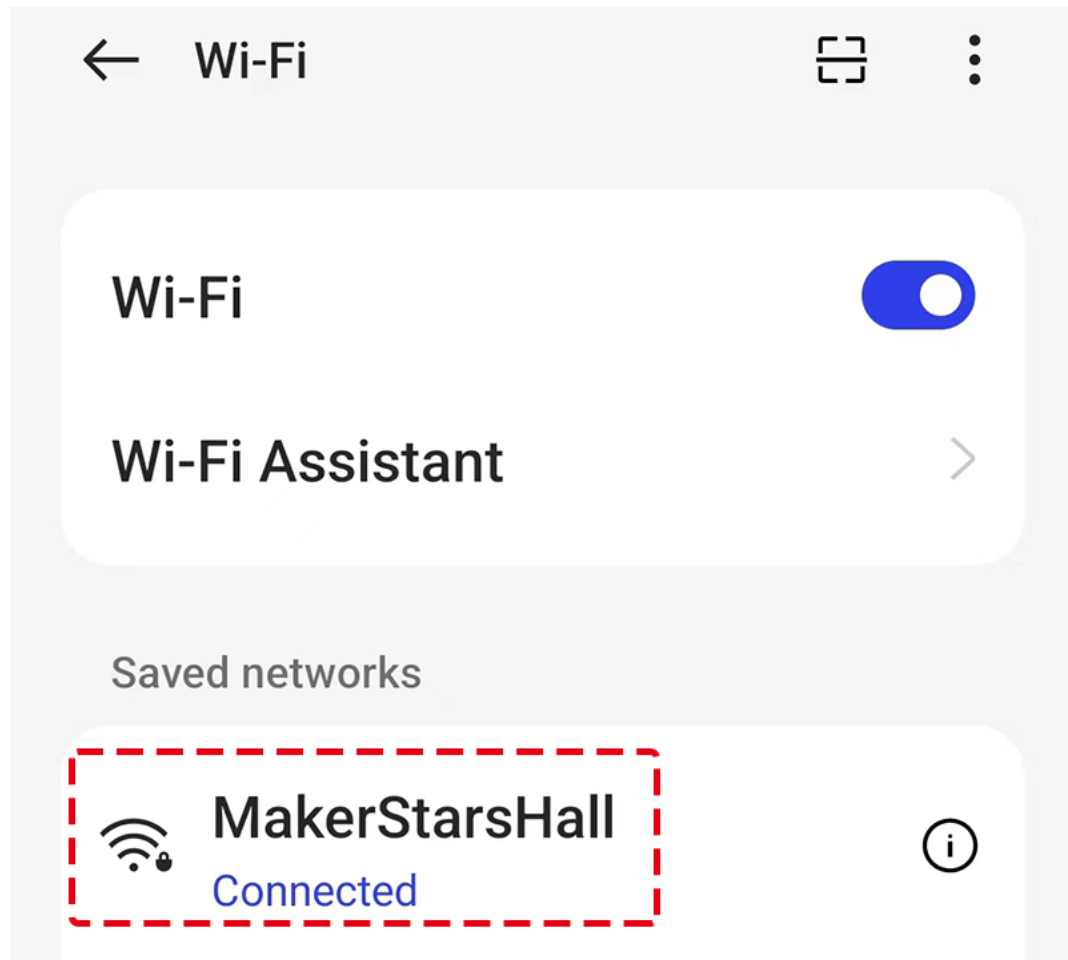
Google LLC • Tools • Browsers

4.3 ★ 73 MB  10B+**Google Chrome: Fast & Secure**

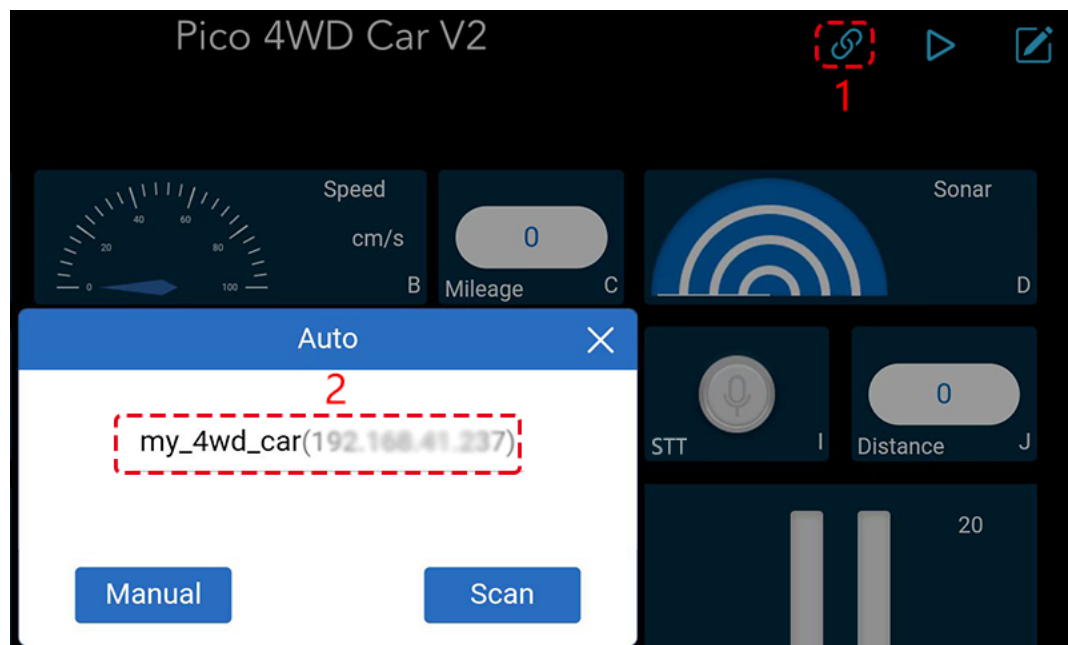
Google LLC • Communication • Tools • Browsers

 Installed

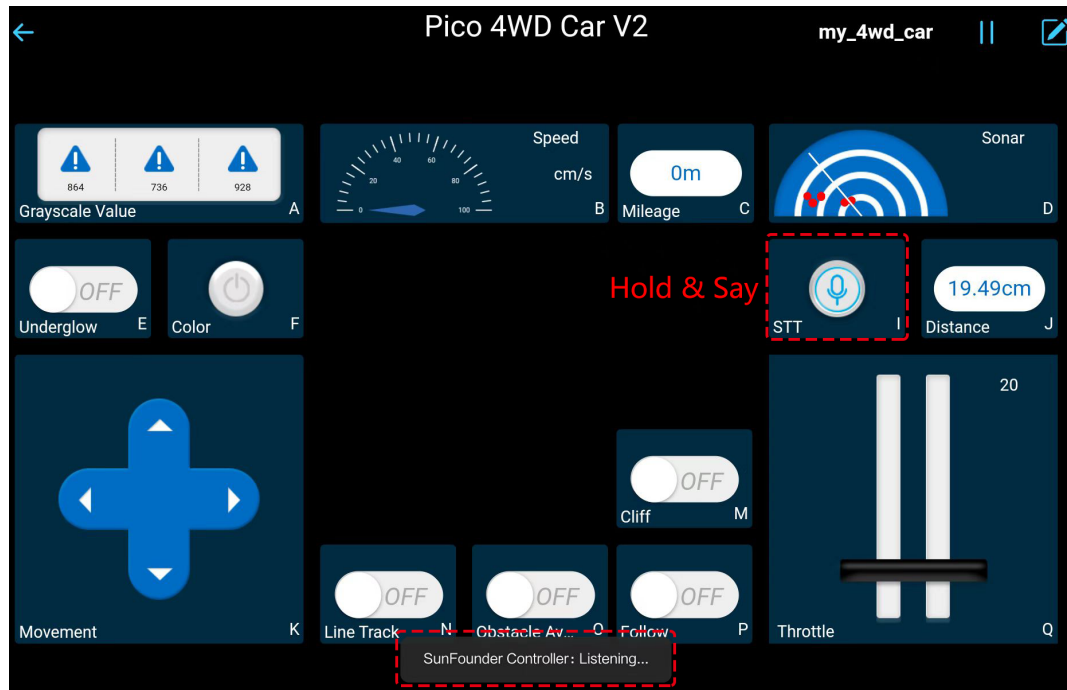
3. Connect your mobile device to the same Wi-Fi as you wrote in the script.



4. Open the controller previously created in SunFounder Controller and connect it to my_4wd_car.



5. Tap and hold the **STT(I)** widget after clicking the  button. A prompt will appear indicating that it is listening. Say the following command to move the car.



- stop: All movements of the car can be stopped.
- forward: Let the car move forward.
- backward: Let the car move backward.
- left: Let the car turn left.
- right: Let the car turn right.

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.