Welcome

Thank you for choosing Freenove products!

About Battery

First, read the document **About_Battery.pdf** in the extracted folder.
If you have not downloaded the zip file, please do so via the link below and extract it on your computer.
https://github.com/Freenove/Freenove_Omni_Wheel_Car_Kit_for_Raspberry_Pi_Pico

Get Support and Offer Input

Freenove provides free and responsive product and technical support, including but not limited to:

- Product quality issues
- Product use and build issues
- Questions regarding the technology employed in our products for learning and education
- Your input and opinions are always welcome
- We also encourage your ideas and suggestions for new products and product improvements

For any of the above, you may send us an email to:

support@freenove.com

Safety and Precautions

Please follow the following safety precautions when using or storing this product:

- Keep this product out of the reach of children under 6 years old.
- This product should be used only when there is adult supervision present as young children lack necessary judgment regarding safety and the consequences of product misuse.
- This product contains small parts and parts, which are sharp. This product contains electrically conductive parts. Use caution with electrically conductive parts near or around power supplies, batteries and powered (live) circuits.
- When the product is turned ON, activated or tested, some parts will move or rotate. To avoid injuries to hands and fingers, keep them away from any moving parts!
- It is possible that an improperly connected or shorted circuit may cause overheating. Should this happen, immediately disconnect the power supply or remove the batteries and do not touch anything until it cools down! When everything is safe and cool, review the product tutorial to identify the cause.
- Only operate the product in accordance with the instructions and guidelines of this tutorial, otherwise parts may be damaged or you could be injured.
- Store the product in a cool dry place and avoid exposing the product to direct sunlight.
- After use, always turn the power OFF and remove or unplug the batteries before storing.

About Freenove

Freenove provides open source electronic products and services worldwide.

Freenove is committed to assist customers in their education of robotics, programming and electronic circuits so that they may transform their creative ideas into prototypes and new and innovative products. To this end, our services include but are not limited to:

- Educational and Entertaining Project Kits for Robots, Smart Cars and Drones
- Educational Kits to Learn Robotic Software Systems for Arduino, Raspberry Pi, micro: bit and Raspberry Pi Pico W.
- Electronic Component Assortments, Electronic Modules and Specialized Tools
- **Product Development and Customization Services**

You can find more about Freenove and get our latest news and updates through our website:

http://www.freenove.com

Copyright

FREENOVE TM

Contents

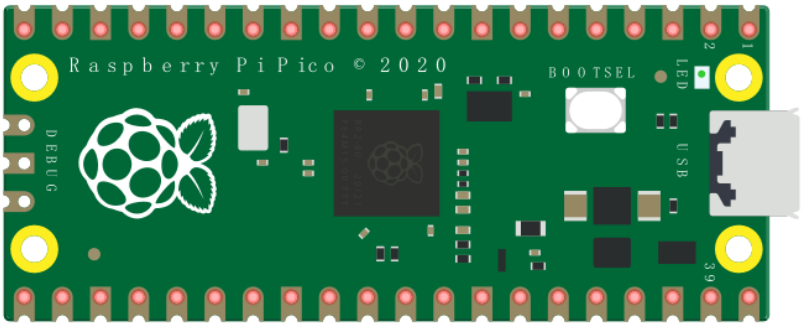
Welcome 2
Contents 4
Raspberry Pi Pico 1
Raspberry Pi Pico W 4
Raspberry Pi Pico 2 7
Car Expansion Board 10
Pin Definition of the Car Board 11
List 12
Car Expansion Board 12
Machinery Parts 13
Transmission Parts 14
Electronic Parts 15
Wires 16
Tools 16
Required but NOT Contained Parts 17
Freenove app 19
Install Freenove app 19
Introduction to the APP 23
Hardware_Communication_Protocol 25
Chapter 0 Installation of Arduino IDE 29
Arduino Software 29
Environment Configuration 31
Additional Remarks 36
Chapter 1 Car Assembly 37
Assembling the Car 37
Library Installation 57
Chapter 2 Uploading First Sketch (LED Blink) 59
Uploading Adruino-compatible Firmware for Raspberry Pi Pico (W) 59
Chapter 3 Serial 63
Related Knowledge 63
Circuit 64
Sketch 64
Chapter 4 Motor Test 67
4.1 Motor 67
4.2 Motor Speed Control through PWM 74
4.3 TT Motor with Encoder 78
Chapter 5 Buzzer Test 88
Related Knowledge 88
Schematic 89
Sketch 89
Chapter 6 ADC Test 91

ADC....91
Schematic....92
Sketch....92
Chapter 7 LED Test....94
Related Knowledge....94
Schematic....95
Sketch....95
Chapter 8 Gyroscope Test....99
Component Knowledge....99
Circuit....101
Schematic....101
Chapter 9 Compass Test....107
Project 9.1 Compass Calibration....107
Project 9.2 Yaw Angle Reading....112
Chapter 10 Ultrasonic Sensor Test....114
Component Knowledge....114
Circuit....115
Schematic....116
Chapter 11 Infrared Test....119
Component Knowledge....119
Schematic....121
Sketch....122
Chapter 12 Bluetooth Test....126
Chapter 13 Kinematic Analysis....132
Related Knowledge....132
Circuit....137
Sketch....137
Chapter 14 Manual Control....142
Circuit....142
Sketch....143
Chapter 15 Free Head Mode....151
Related Knowledge....151
Circuit....151
Sketch....152
Chapter 16 Lock Head Mode....162
Circuit....162
Sketch....162
Chapter 17 Around Mode....171
Related Knowledge....171
Sketch....174
Circuit....176
About Around Mode....176
Chapter 18 Sonar Mode....178
Sonar Mode Summary....178

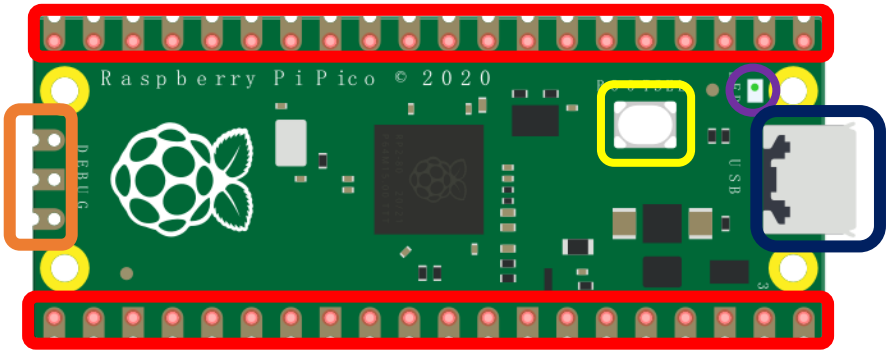
Circuit 178
Sketch 179
About Sonar Mode 184
Chapter 19 Infrared Control 185
Circuit 185
Sketch 186
Chapter 20 Bluetooth Car 190
Sketch 190
What's next? 192

Raspberry Pi Pico

Before learning Pico, we need to know about it. Here we use an imitated diagram of Pico, which resembles the actual Pico.

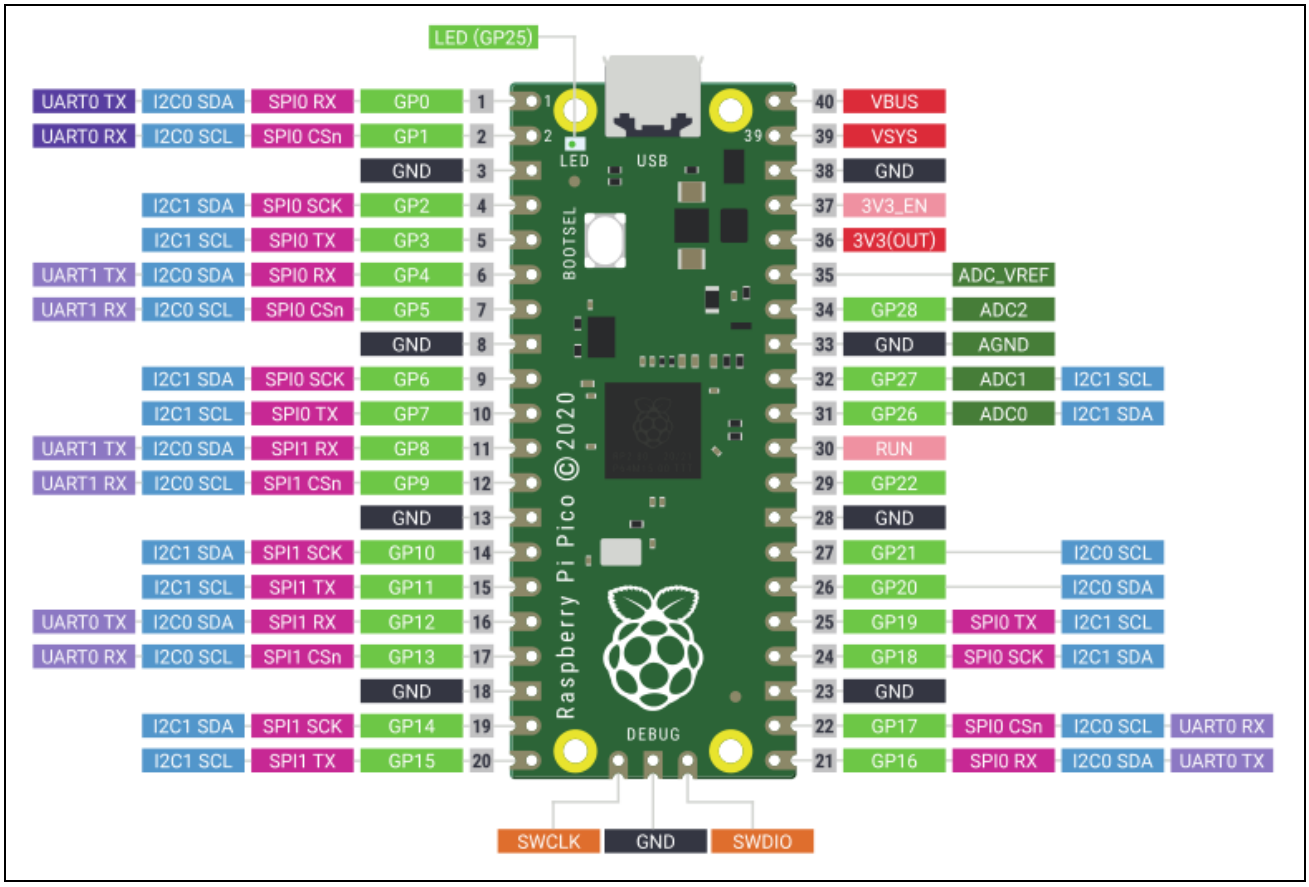


The hardware interfaces are distributed as follows:



Frame color	Description
	Pins
	BOOTSEL button
	USB port
	LED
	Debugging

Function definition of pins:



Color	Pins	Color	Pins
	GND		Power
	GPIO		ADC
	UART(default)		UART
	SPI		I2C
	System Control		Debugging

For details: https://datasheets.raspberrypi.org/pico/pico-datasheet.pdf

UART, I2C, SPI Default Pin

In Arduino IDE, the default pins of serial port are Pin0 and Pin1.
Note: Serial port is virtualized by RP2040. Therefore, when using the serial port, please enable the verification function of DTR. It can work under any baud rate.

UART

Function	Default
UART_BAUDRATE	X
UART_BITS	8
UART_STOP	1
UART_TX	Pin 0
UART_RX	Pin 1

I2C

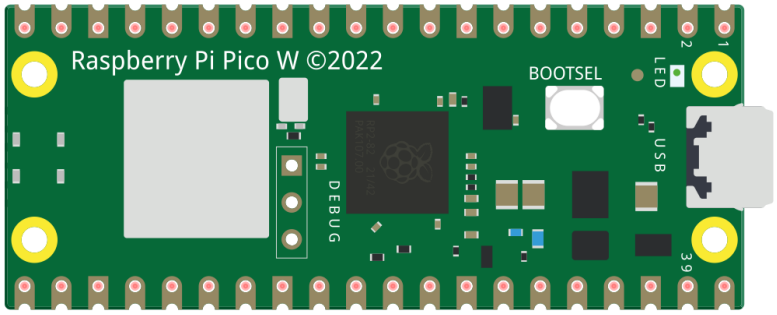
Function	Default
I2C Frequency	400000
I2C_SDA	Pin 4
I2C_SCL	Pin 5

SPI

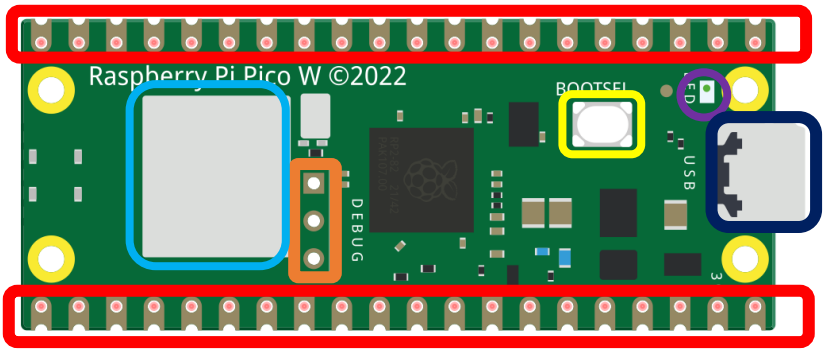
Function	Default
SPI_BAUDRATE	1000000
SPI_POLARITY	0
SPI_PHASE	0
SPI_BITS	8
SPI_FIRSTBIT	MSB
SPI_SCK	Pin 18
SPI_MOSI	Pin 19
SPI_MISO	Pin 16
SPI_SS	Pin 17

Raspberry Pi Pico W

Raspberry Pi Pico W adds CYW43439 as the WiFi function on the basis of Raspberry Pi Pico. It is connected to RP2040 chip through SPI interface.

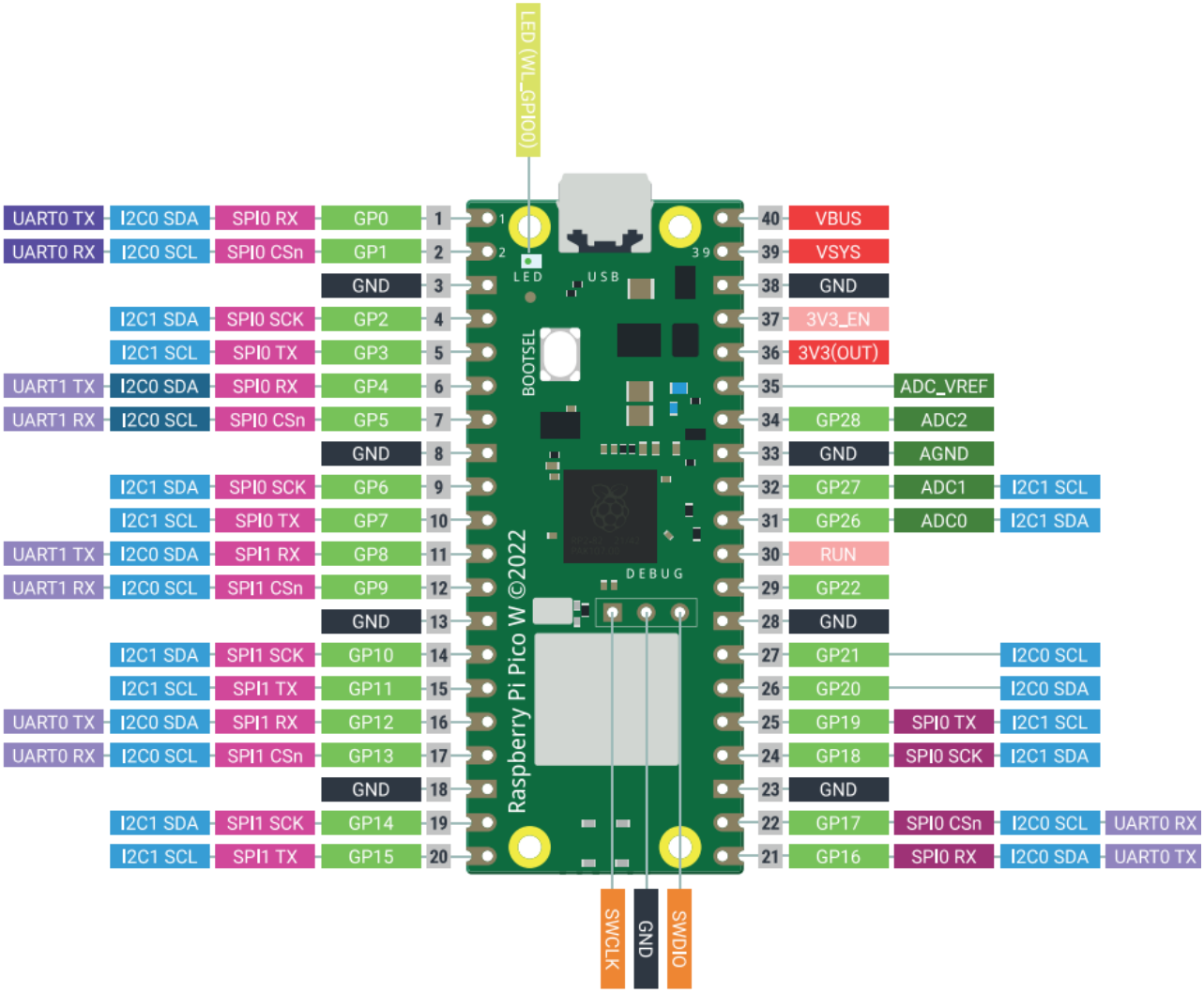


The hardware interfaces are distributed as follows:



Frame color	Description
	Pins
	BOOTSEL button
	USB port
	LED
	Debugging
	Wireless

Function definition of pins:



Color	Pins	Color	Pins
	GND		Power
	GPIO		ADC
	UART(default)		UART
	SPI		I2C
	System Control		Debugging

For details: https://datasheets.raspberrypi.com/picow/pico-w-datasheet.pdf

UART, I2C, SPI, Wireless Default Pin

In Arduino IDE, the default pins of serial port are Pin0 and Pin1.
Note: Serial port is virtualized by RP2040. Therefore, when using the serial port, please enable the verification function of DTR. It can work under any baud rate.

UART

Function	Default
UART_BAUDRATE	X
UART_BITS	8
UART_STOP	1
UART_TX	Pin 0
UART_RX	Pin 1

I2C

Function	Default
I2C Frequency	400000
I2C_SDA	Pin 4
I2C_SCL	Pin 5

SPI

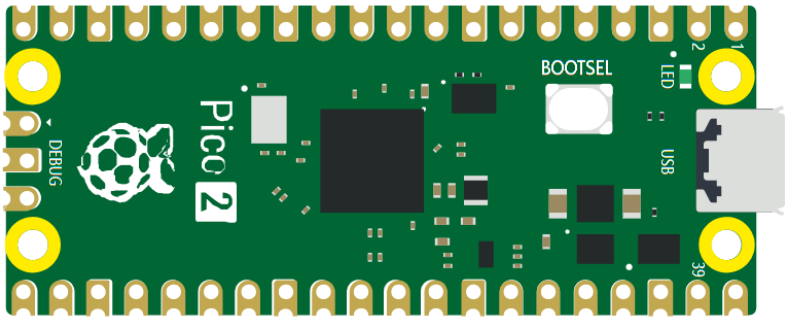
Function	Default
SPI_BAUDRATE	1000000
SPI_POLARITY	0
SPI_PHASE	0
SPI_BITS	8
SPI_FIRSTBIT	MSB
SPI_SCK	Pin 18
SPI_MOSI	Pin 19
SPI_MISO	Pin 16
SPI_SS	Pin 17

Wireless

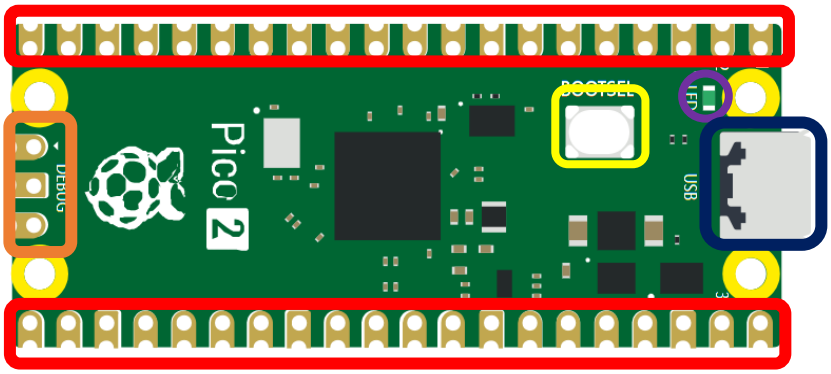
Function	Default
WL_ON	GPIO23
WL_D	GPIO24
WL_CLK	GPIO29_ADC
WL_CS	GPIO25

Raspberry Pi Pico 2

Raspberry Pi Pico 2 uses RP2350 chip as the main controller, which equipped with dual Cortex-M33 or Hazard3 processors, capable of running up to 150 MHz, providing a significant boost in processing power, compared with the original pico. It also doubles the memory with 520KB of SRAM and 4MB of onboard flash memory, with the ADC sampling frequency increasing to up to 500ksps. In addition, it adds 8 more PWM channels, and features additional interfaces like 2× Timer with 4 alarms, 1× AON Timer and 4 x PIO.

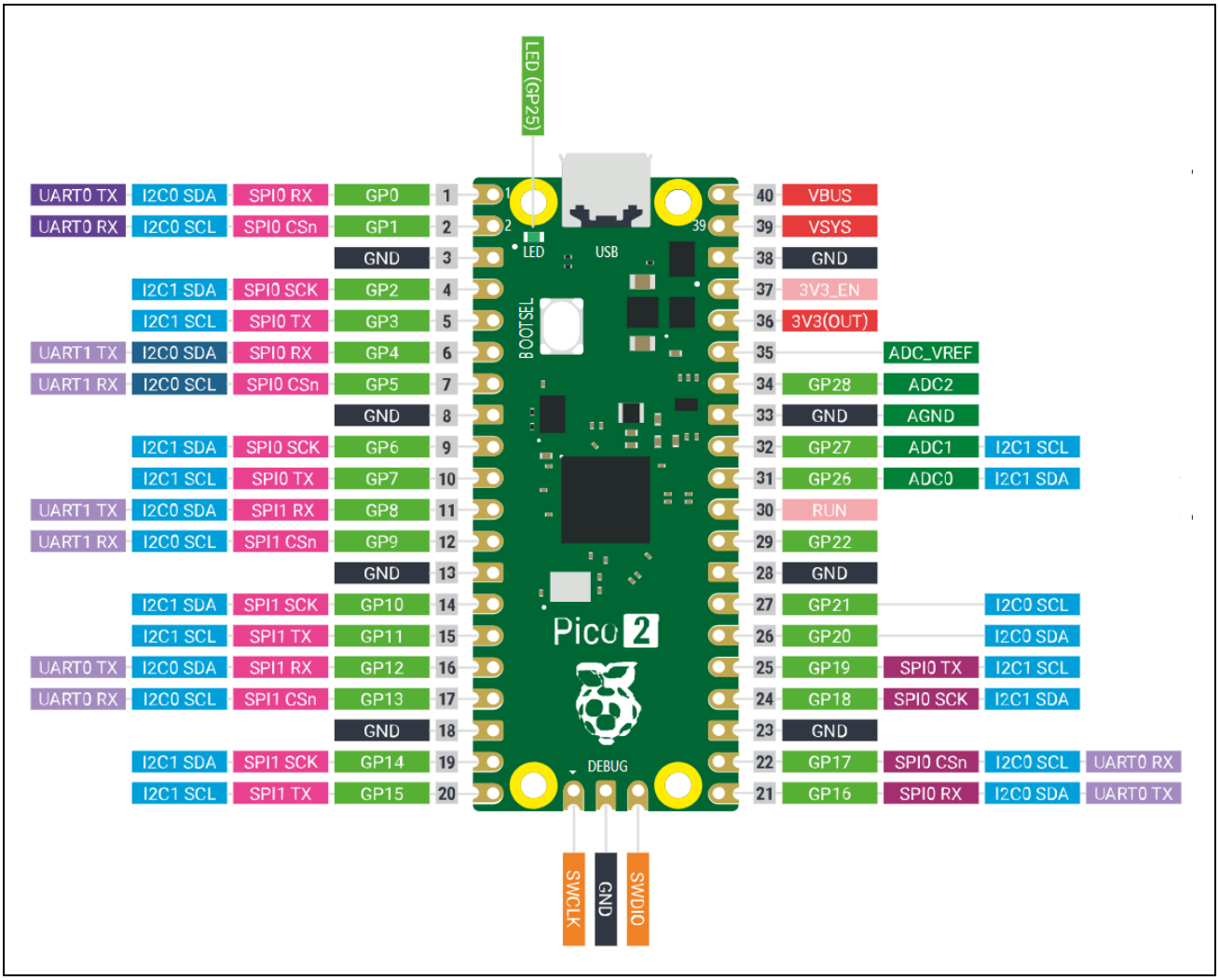


The hardware interfaces are distributed as follows:



Frame color	Description
	Pins
	BOOTSE button
	USB port
	LED
	Debugging

Function definition of pins:



Color	Pins	Color	Pins
	GND		Power
	GPIO		ADC
	UART(default)		UART
	SPI		I2C
	System Control		Debugging

For details: https://datasheets.raspberrypi.com/pico/pico-2-datasheet.pdf

UART, I2C, SPI Default Pin

In Arduino IDE, the default pins of serial port are Pin0 and Pin1.
Note: Serial port is virtualized by RP2350. Therefore, when using the serial port, please enable the verification function of DTR. It can work under any baud rate.

UART

Function	Default
UART_BAUDRATE	X
UART_BITS	8
UART_STOP	1
UART_TX	Pin 0
UART_RX	Pin 1

I2C

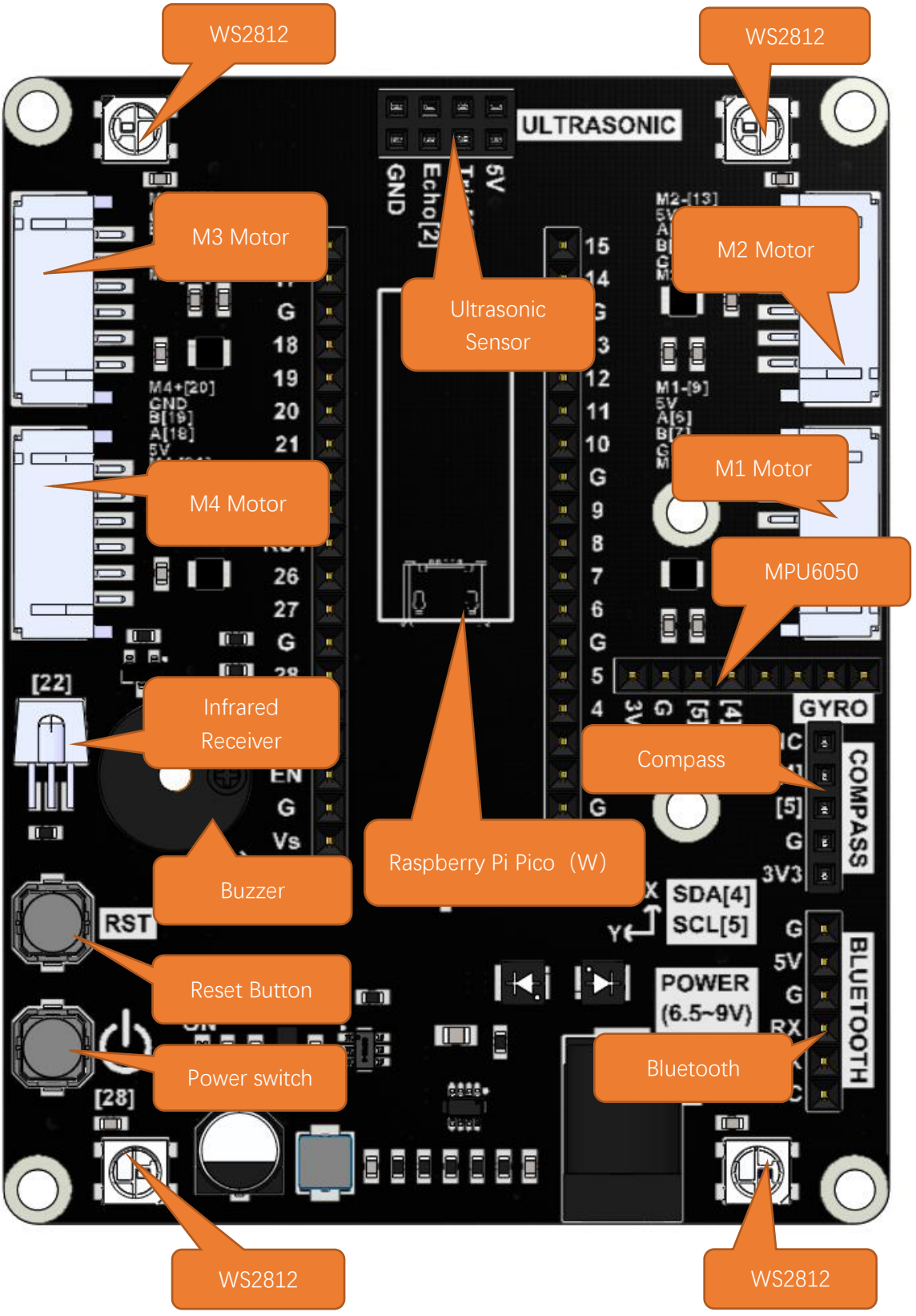
Function	Default
I2C Frequency	400000
I2C_SDA	Pin 4
I2C_SCL	Pin 5

SPI

Function	Default
SPI_BAUDRATE	1000000
SPI_POLARITY	0
SPI_PHASE	0
SPI_BITS	8
SPI_FIRSTBIT	MSB
SPI_SCK	Pin 18
SPI_MOSI	Pin 19
SPI_MISO	Pin 16
SPI_SS	Pin 17

Car Expansion Board

The function diagram of the Raspberry Pi Pico W car is as follows:



Pin Definition of the Car Board

To learn what each GPIO corresponds to, please refer to the following table.

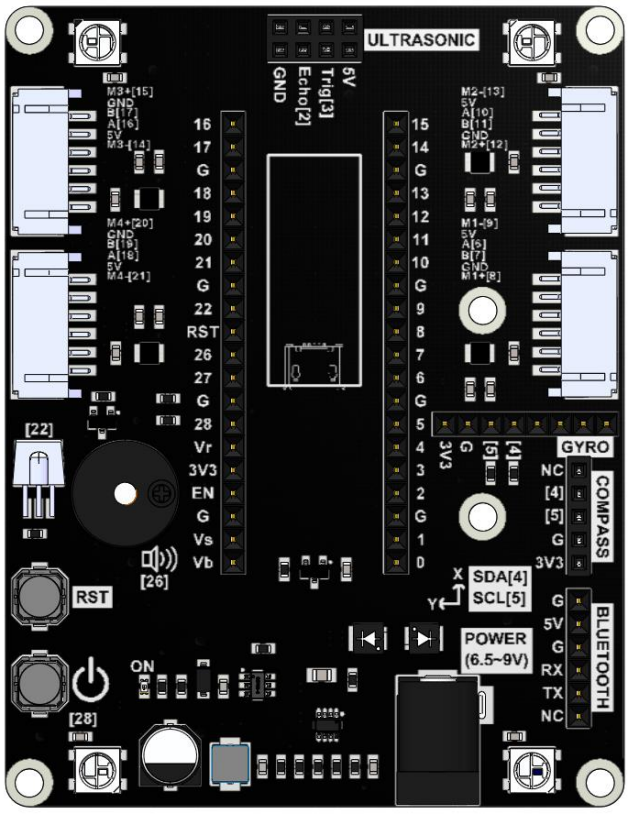
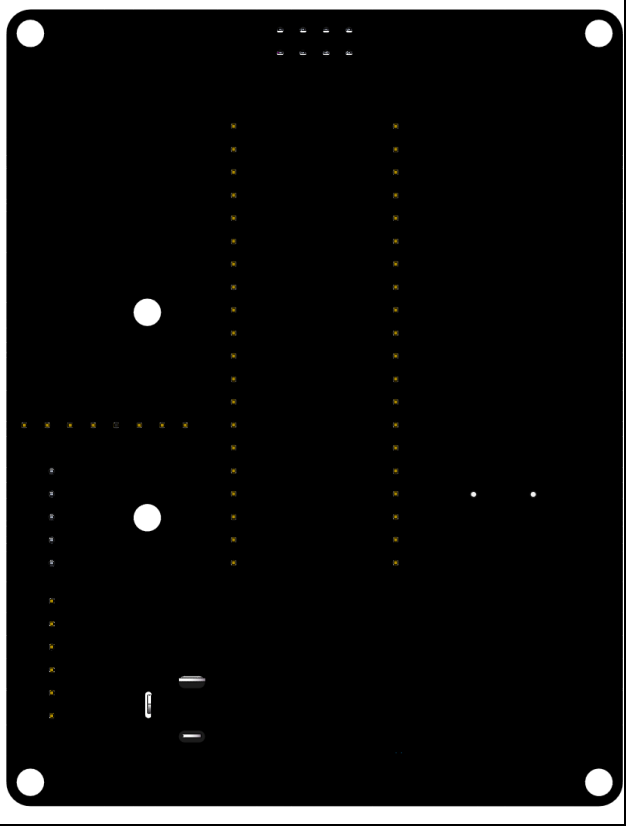
The functions of the pins are allocated as follows:

Pins of Raspberry Pi Pico W	Functions	Description
GPIO0	BlueTooth_TX	Bluetooth
GPIO1	BlueTooth_RX	
GPIO2	M1_A	TT Motor with Encoder 1
GPIO3	M1_B	
GPIO4	M1_IN1	
GPIO5	M1_IN2	
GPIO11	M2_A	TT Motor with Encoder 2
GPIO10	M2_B	
GPIO12	M2_IN1	
GPIO13	M2_IN2	
GPIO14	M3_A	TT Motor with Encoder 3
GPIO15	M3_B	
GPIO16	M3_IN1	
GPIO17	M3_IN2	
GPIO20	M4_A	TT Motor with Encoder 4
GPIO21	M4_B	
GPIO18	M4_IN1	
GPIO19	M4_IN2	
GPIO6/SDA	IIC-SDA	Compass && MPU6050
GPIO7/SCL	IIC-SCL	
GPIO8	Echo	Ultrasonic Sensor
GPIO9	Trig	
GPIO28	WS2812	WS2812RGB LEDs
GPIO27/A1	Battery Power	Battery Power Measurement
GPIO22	Buzzer	Buzzer
GPIO26	IR	Integrated IR Receiver

List

If you have any concerns, please feel free to contact us via support@freenove.com

Car Expansion Board

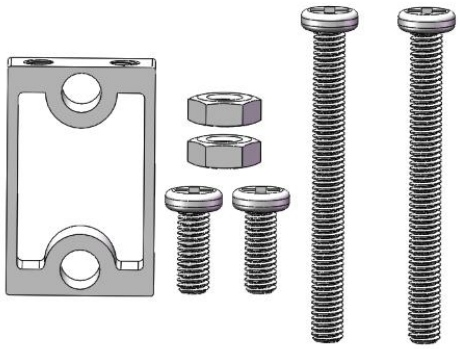
Top	Bottom
	

Machinery Parts

M3*30 Copper Standoff  x5 Freenove	M3*11 Copper Standoff  x3 Freenove	M3*10 Countersunk Head Screw  x5 Freenove
M3*8 Screw  x12 Freenove	M3 Nut  x5 Freenove	

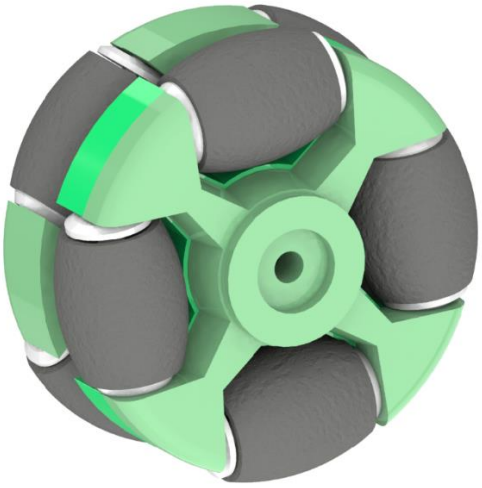
Transmission Parts

Motor bracket package x4
(Motor Bracket x1, M3x8 Screws x2, M3x30 Screws x2) x4

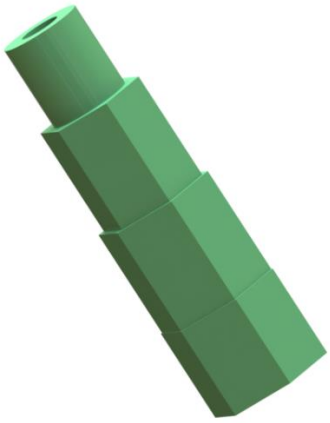


TT Motor with Encoder x 4

Omni wheels x 4

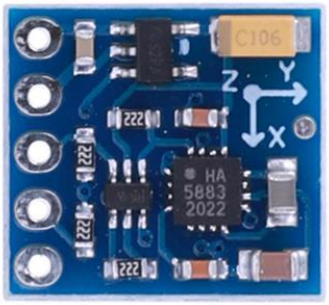
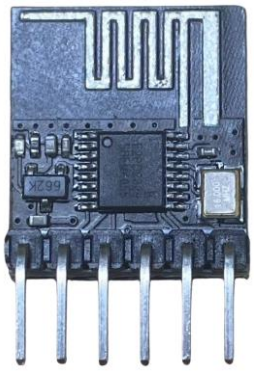
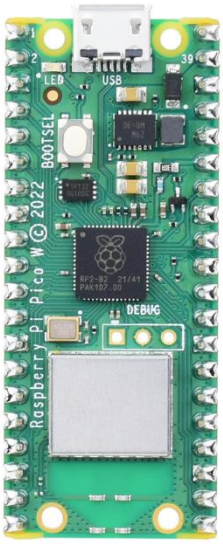


Coupling for Omni Wheels x4



M2.5x25 Screw x4

Electronic Parts

Digital Compass x 1 	MPU6050 Gyroscope x 1 	Bluetooth x 1 
Infrared Emitter x1 	Raspberry Pi Pico W x1 or Raspberry Pi Pico x1 	
Battery Holder x 1 	Ultrasonic Module x1 	

Wires

USB Data Cable（Micro USB）1m

PH2.0mm-6P Wires-Same Direction 200mm

Tools

Cross Srewdriver-3mm x 1

Required but NOT Contained Parts

2 x 3.7V 18650 lithium **rechargeable** batteries with continuous discharge current >3A.
It is easier to find proper battery on eBay than Amazon. Search "18650 high drain" on eBay.
We provide some battery links in the "About Battery.pdf".

Preface

Welcome to use Omni Wheeled Car Kit for Raspberry Pi Pico (W). Following this tutorial, you can make a very cool car with many functions.

Based on the Raspberry Pi Pico (W) development board, a popular IoT control board, this kit uses the very popular Arduino IDE for programming, so you can share and exchange your experience and design ideas with enthusiasts all over the world. The parts in the kit include all electronic components, modules, and mechanical components required for making the car. They are all individually packaged. There are detailed assembly and debugging instructions in this book. If you encounter any problems, please feel free to contact us for fast and free technical support.

support@freenove.com

This car does not require a high threshold for users. Even if you know little professional knowledge, you can make your own smart car easily with the guidance of the tutorial. If you're really interested in Raspberry Pi Pico (W) and hope to learn how to program and build circuits, please visit our website: www.freenove.com

or contact us to buy our kit designed for beginners:

Freenove Ultimate Kit for Raspberry Pi Pico.

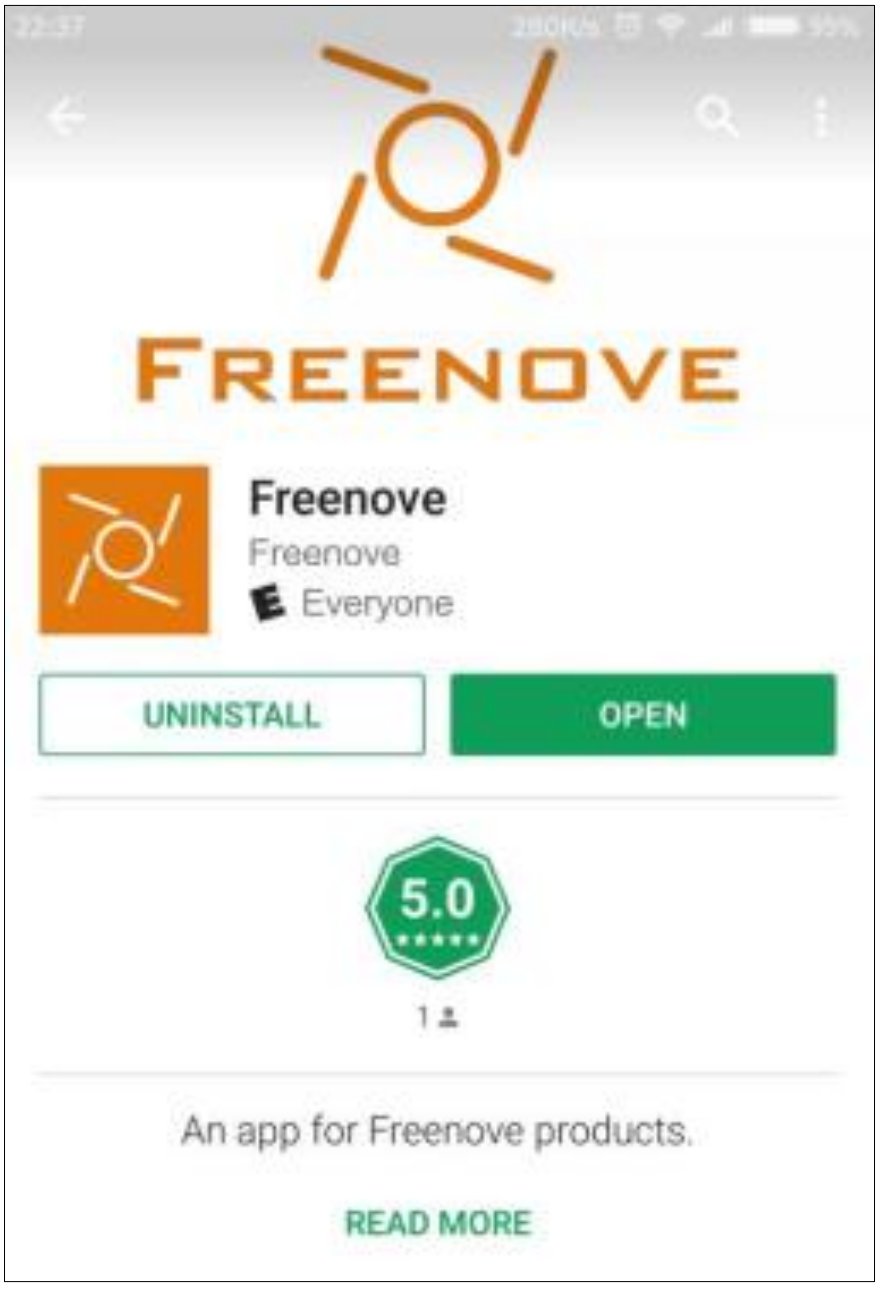
Freenove app

Install Freenove app

There are three ways to install our app.

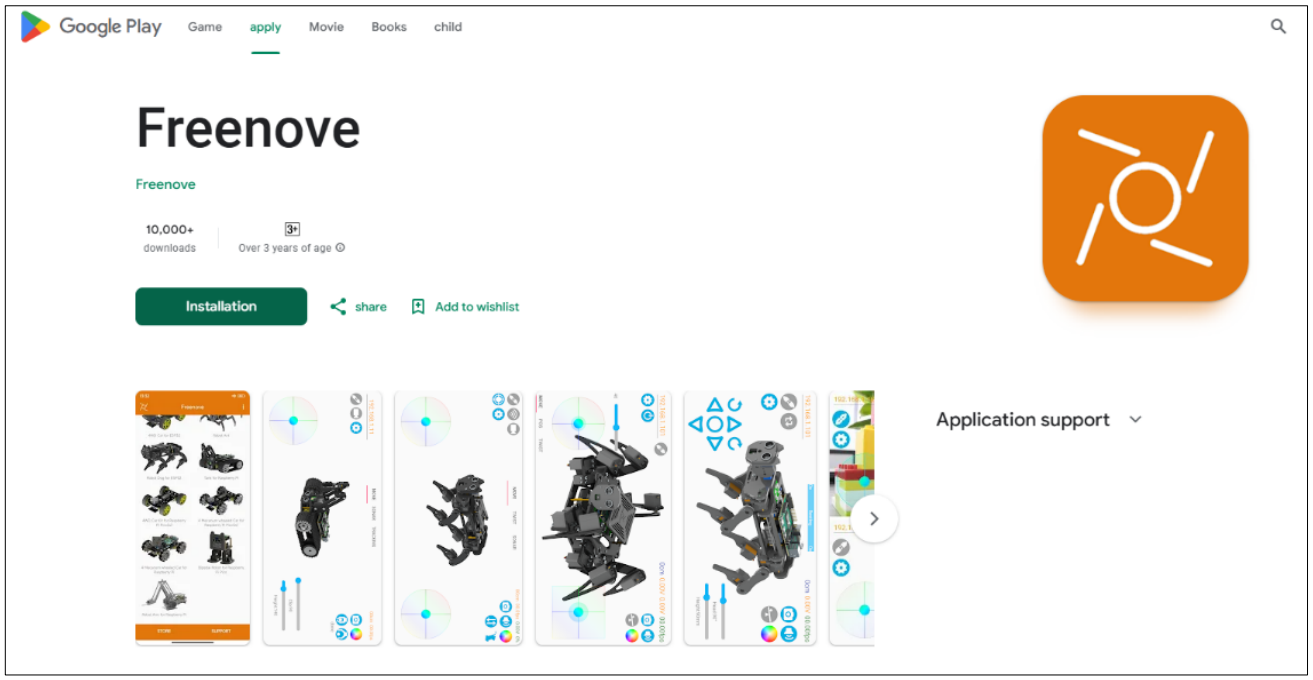
1. Google Play Software

Search “Freenove” on Google Play to download and install.



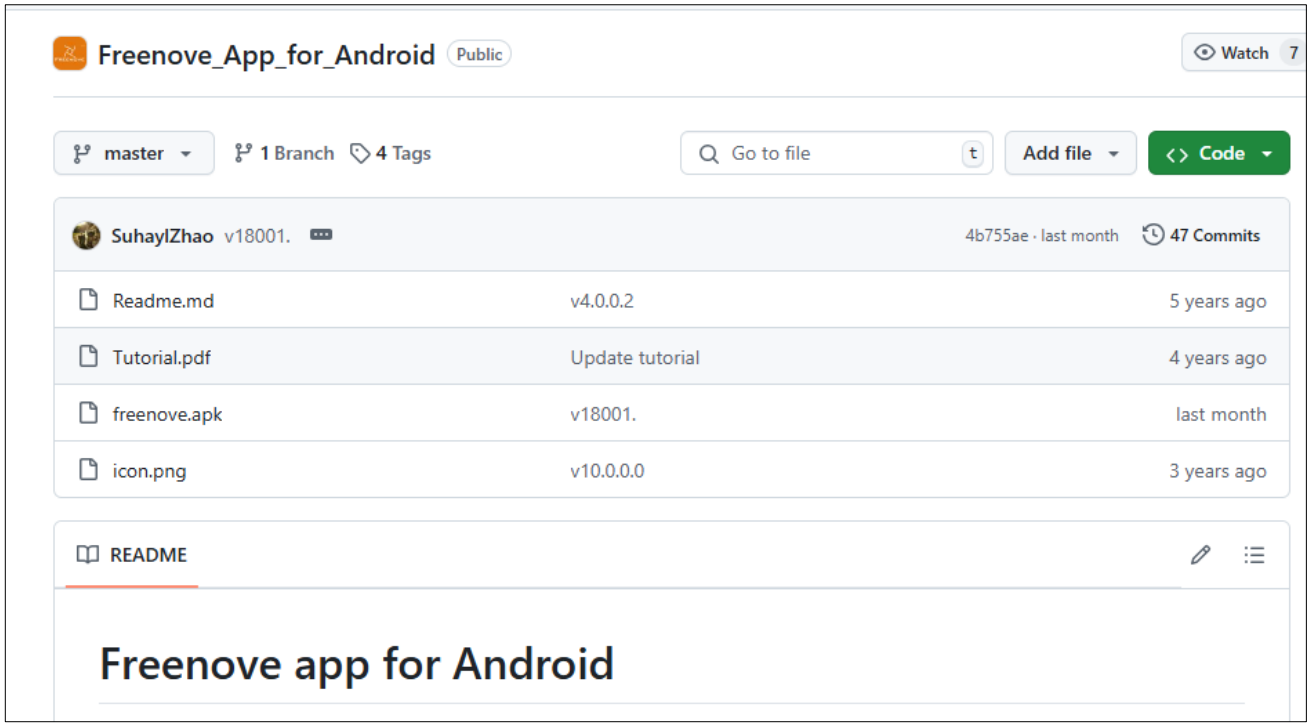
2. Google Play Website

Visit https://play.google.com/store/apps/details?id=com.freenove.suhayl.Freenove, and click install

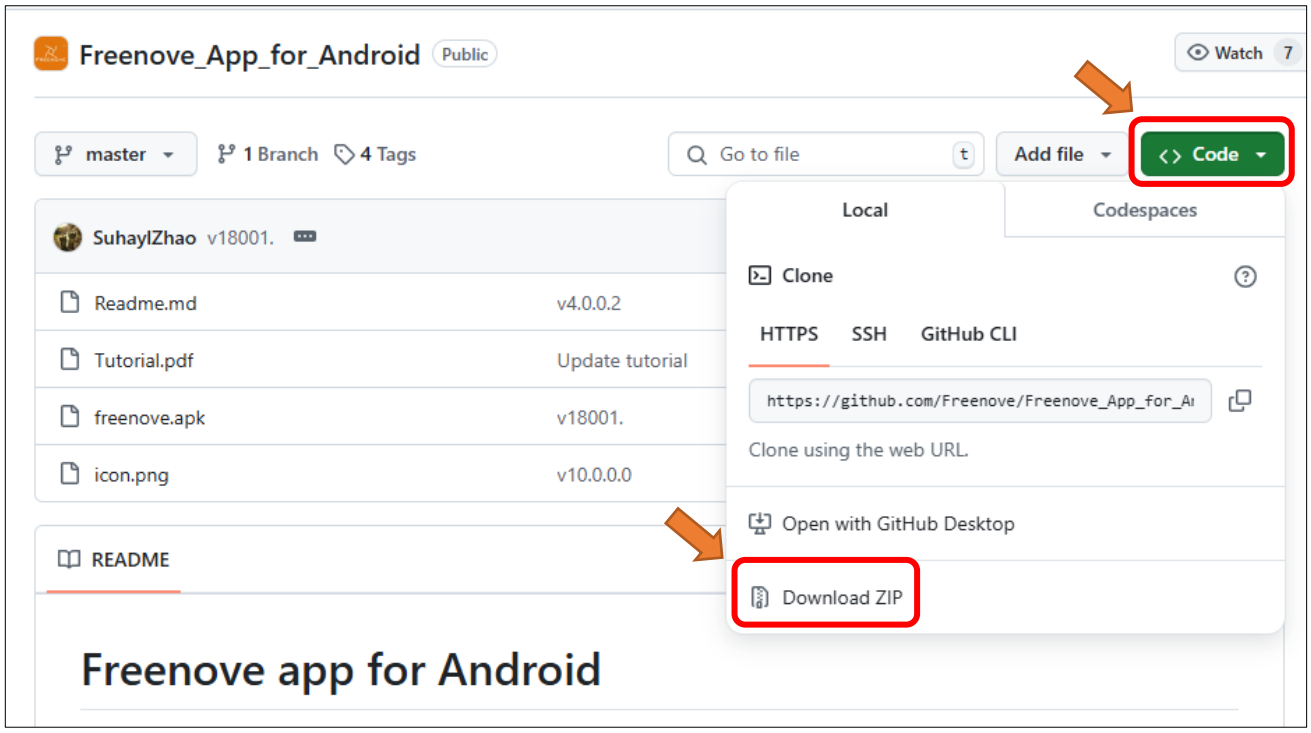


3. GitHub

Visit https://github.com/Freenove/Freenove_app_for_Android, download the files in this library, and install freenove.apk to your Android phone manually.



Expand the options of "**Code**", and click "**Download ZIP**".

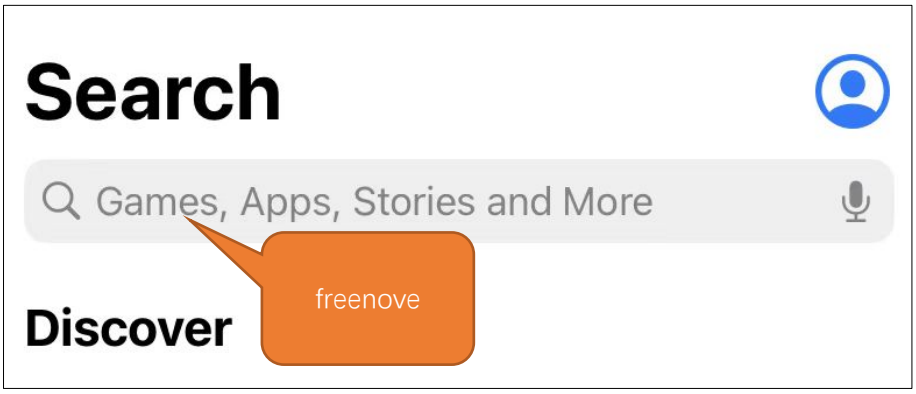


4. iOS

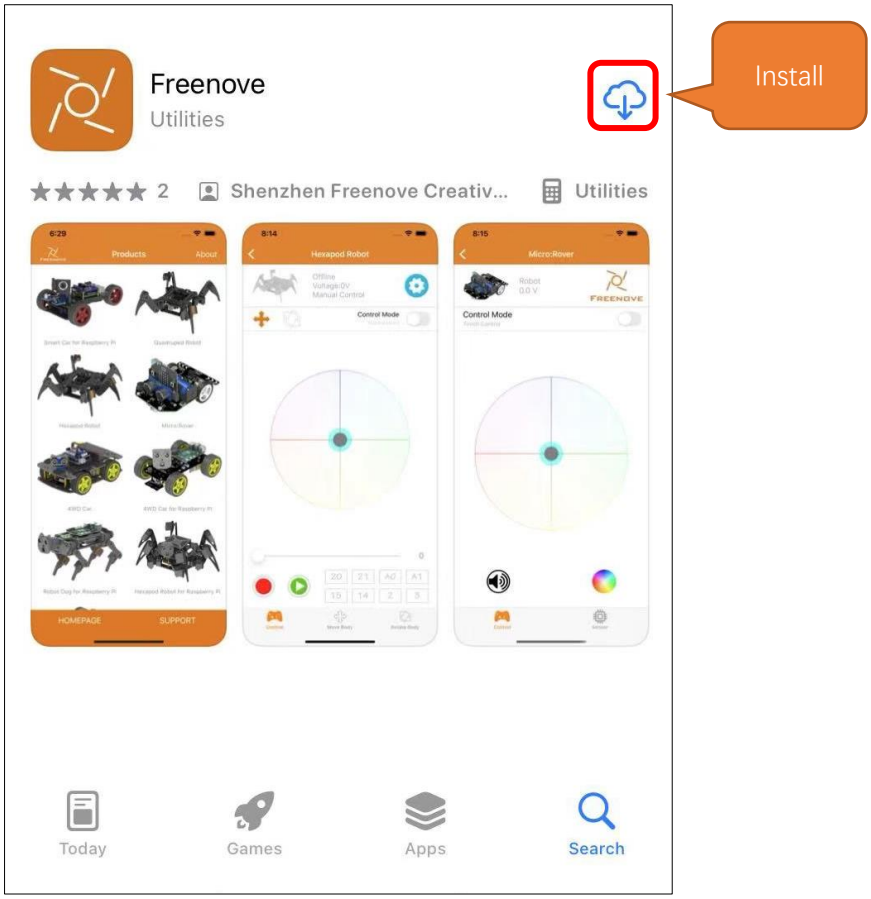
Open App Store on your iPhone/iPad.

Tap **Search** on the menu bar at the bottom

Input **freenove** to search



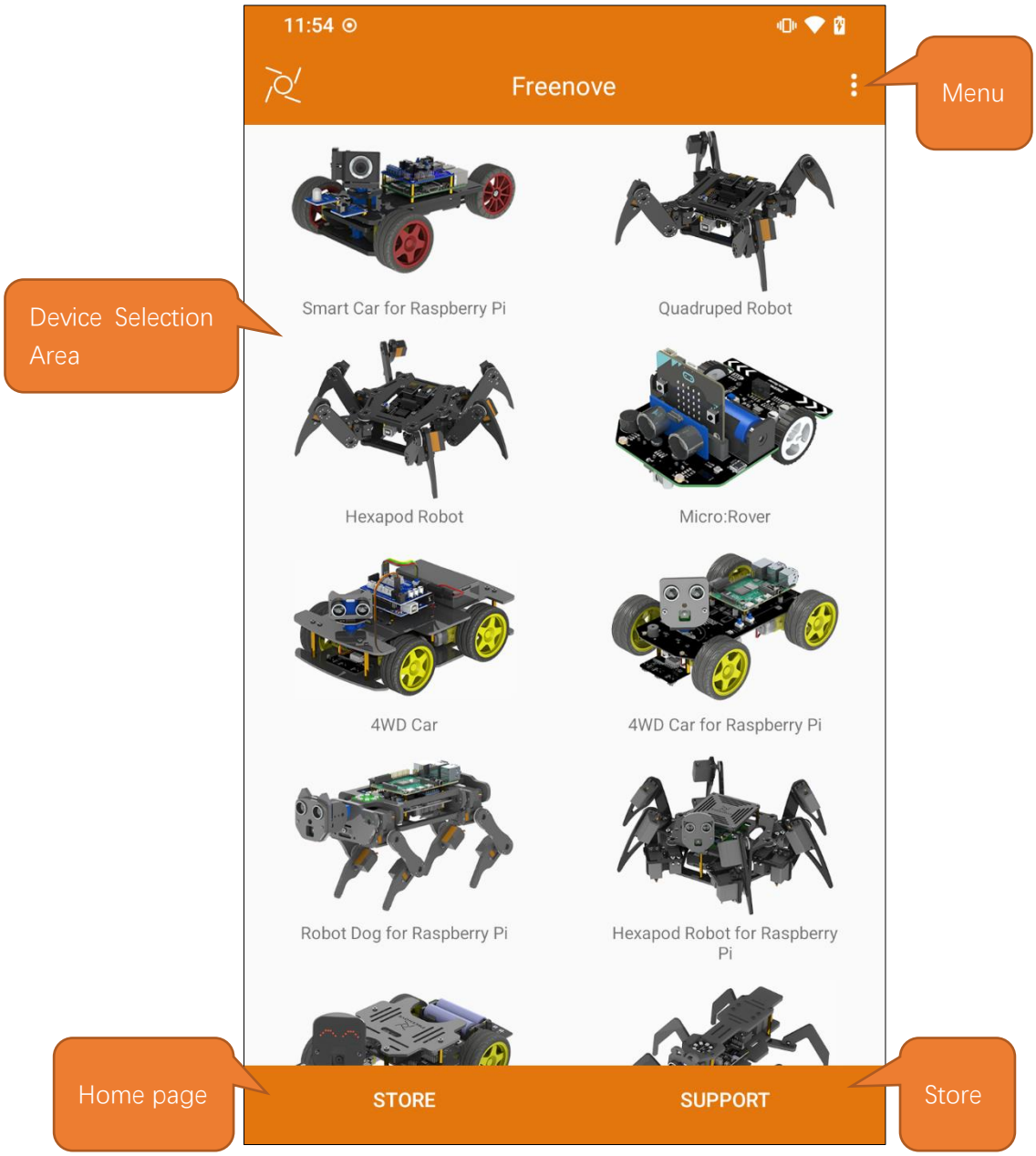
Click the icon to install.



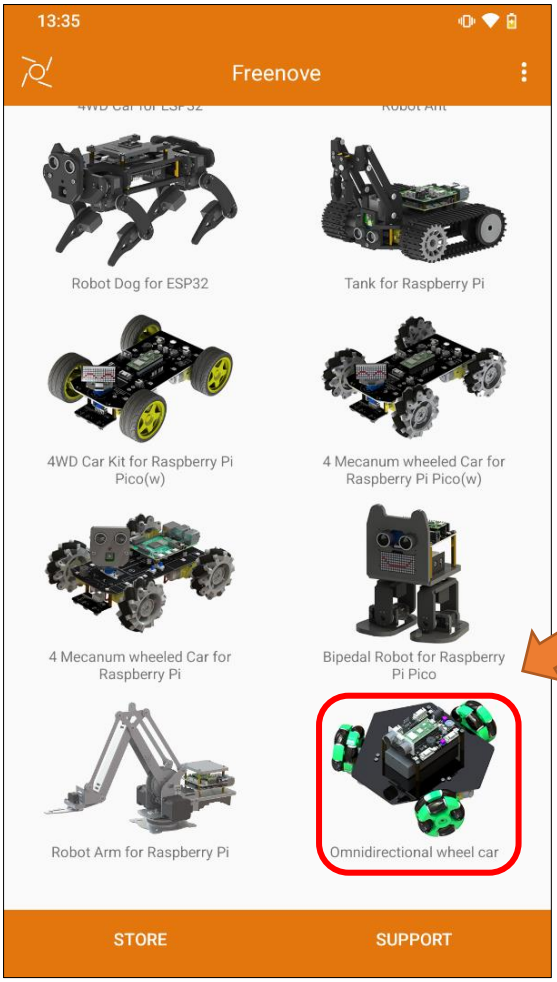
Introduction to the APP

Menu

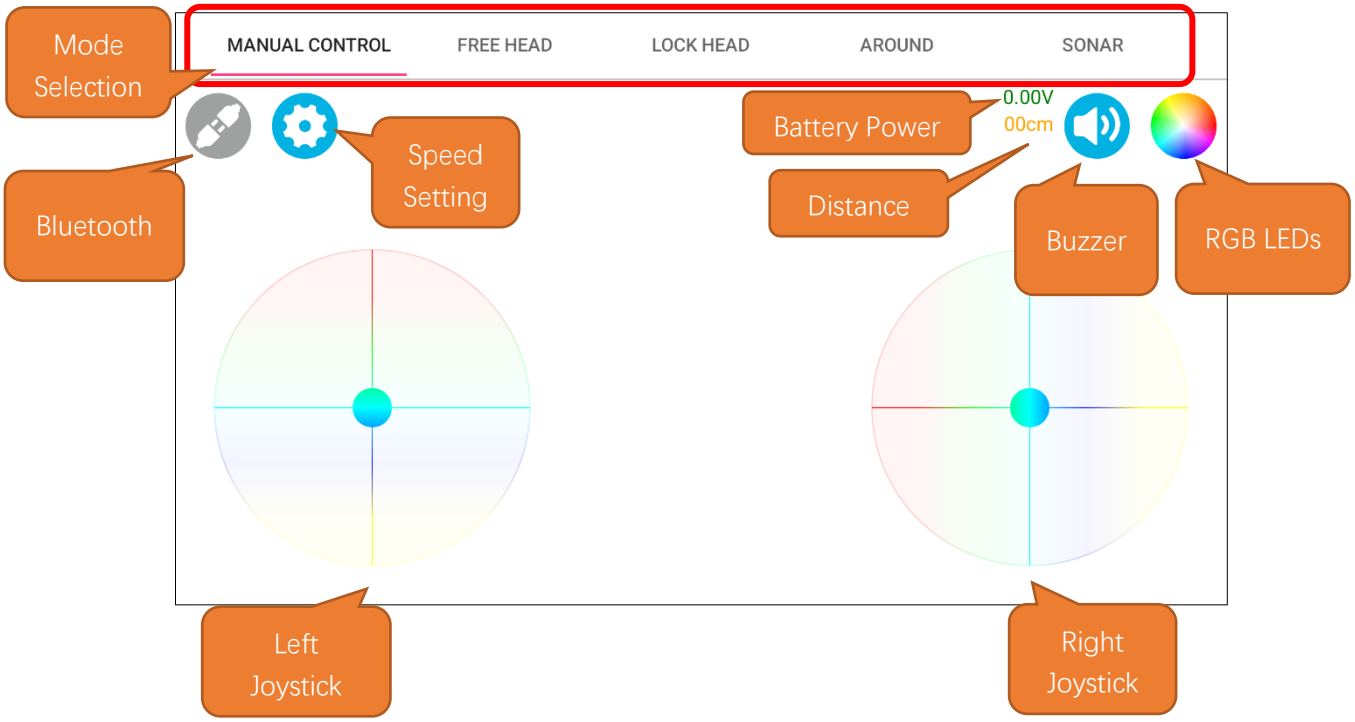
Open application “Freenove”, as shown below:



Scroll down to find Omnidirectional wheel car and tap it.



The interface is as shown below:



Hardware_Communication_Protocol

Supported Communication

Communication	Bluetooth Device Name
Bluetooth	BT05

Communication Command Format

The communication command format is as follows:
"A#10#20#30#40#50#\n".
Where the first character of each command is the command character, used to distinguish the general category of the command, such as "A".
The character "#" is the delimiter between the command character and the parameters, used to split the string.
Each command ends with a newline character "#\n", which is used to separate each command.

When parsing commands, one should first split the commands with “\n”, then split the command character and parameters of each command with “#”. If there is any remaining content after splitting with “\n”, it should be concatenated into the parsing of the subsequent commands.

Each command consists of a command character and parameters, with one command character and a variable number of parameters, ranging from 0 to n, depending on the specific command.

Command Characters of FNK0097

```
#define ENTER                '\n'
#define INTERVAL_CHAR        '#'
#define CMD_MOTOR_Bluetooth  'A'
#define CMD_LED_Bluetooth    'C'
#define CMD_BUZZER_Bluetooth 'D'
#define CMD_ULTRASONIC_Bluetooth 'E'
#define CMD_STATE_Bluetooth  'M'
#define CMD_CIRCLE_Bluetooth 'B'
#define CMD_BATTERY _Bluetooth 'F'
```

Communication Protocol of FNK0097

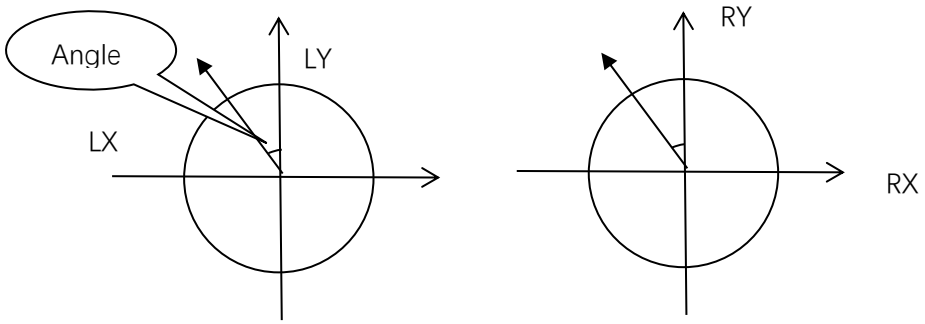
CMD_MOTOR_Bluetooth = "A"

This command controls the car's basic movements.

Format: A#Parameter1#Parameter2#Parameter3#Parameter4\n, where:

- Parameter1 represents the angle between the joystick direction and the Y-axis, with a range of 0-180 for the negative semi-axis and 0-(-180) for the positive semi-axis.
- Parameter2 represents the length of the joystick, with a range of 0-100.
- Parameter3 represents the angle of the second joystick, with a similar range as Parameter1.
- Parameter4 represents the length of the second joystick, with a range similar to Parameter2.

App Commands	Actions
CMD_M_MOTOR#0#100#0#0\n	Car moves forward (at a speed of 100)
CMD_M_MOTOR#180#100#0#0\n	Car moves backward (at a speed of 100)
CMD_M_MOTOR#0#0#90#100\n	Car turns left (at a speed of 100)
CMD_M_MOTOR#0#0#-90#100\n	Car turns right (at a speed of 100)
CMD_M_MOTOR#0#0#0#0\n	Car stops
CMD_M_MOTOR#90#100#0#0\n	Car moves left (at a speed of 100)
CMD_M_MOTOR#-90#100#0#0\n	Car moves right (at a speed of 100)
CMD_M_MOTOR#45#100#0#0\n	Car moves diagonally forward to the left (at a speed of 100)
CMD_M_MOTOR#-45#100#0#0\n	Car moves diagonally forward to the right (at a speed of 100)
CMD_M_MOTOR#135#100#0#0\n	Car moves diagonally backward to the left (at a speed of 100)
CMD_M_MOTOR#-135#100#0#0\n	Car moves diagonally backward to the right (at a speed of 100)



CMD_LED_Bluetooth = "C"

This command is to change the LED modes.

App Commands	Modes
C#0\n	OFF

C#1\n	RGB control (manually, with RGB input)
C#2\n	Pursuit Mode (with RGB input)
C#3\n	Blink Mode (with RGB input)
C#4\n	Breathing Mode (with RGB input)
C#5\n	Rainbow Breathing Mode (RGB parameters are invalid)

CMD_BUZZER_Bluetooth = "D"

This command controls the buzzer.

The frequency of the buzzer is fixed at 2000.

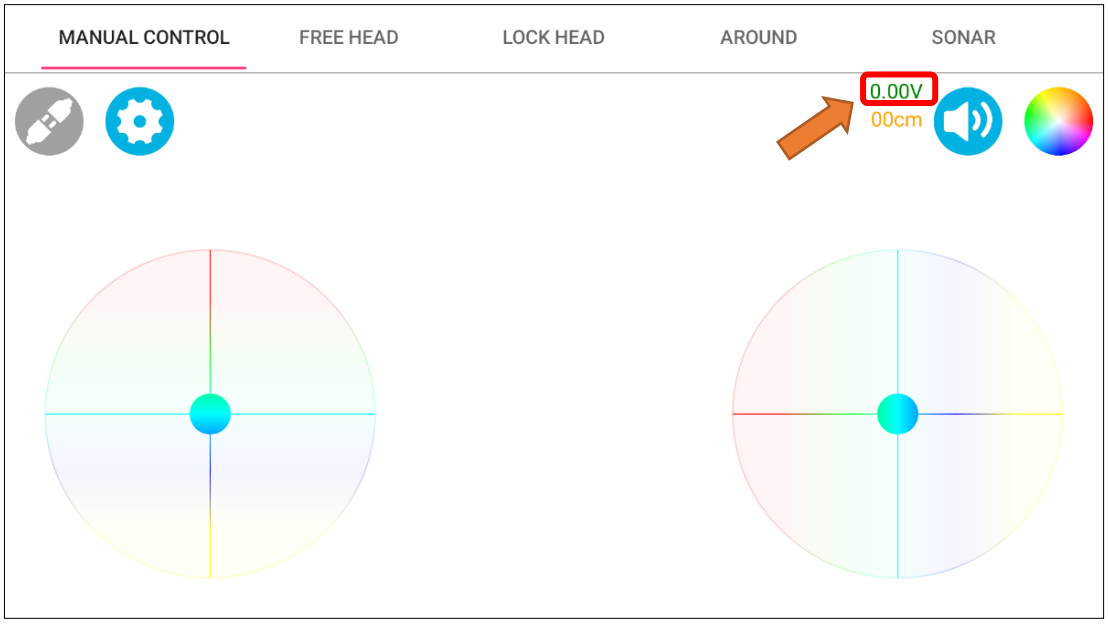
App Command	Action
D#2000\n	Activate buzzer
D#0\n	Deactivate buzzer

CMD_POWER_Bluetooth = "F"

This command checks the battery power.

The slave device proactively sends data to the master device in the format: P#Battery Voltage\n (Example: P#8.12\n). In the APP, this voltage value will be displayed, with the unit being millivolts (mv).

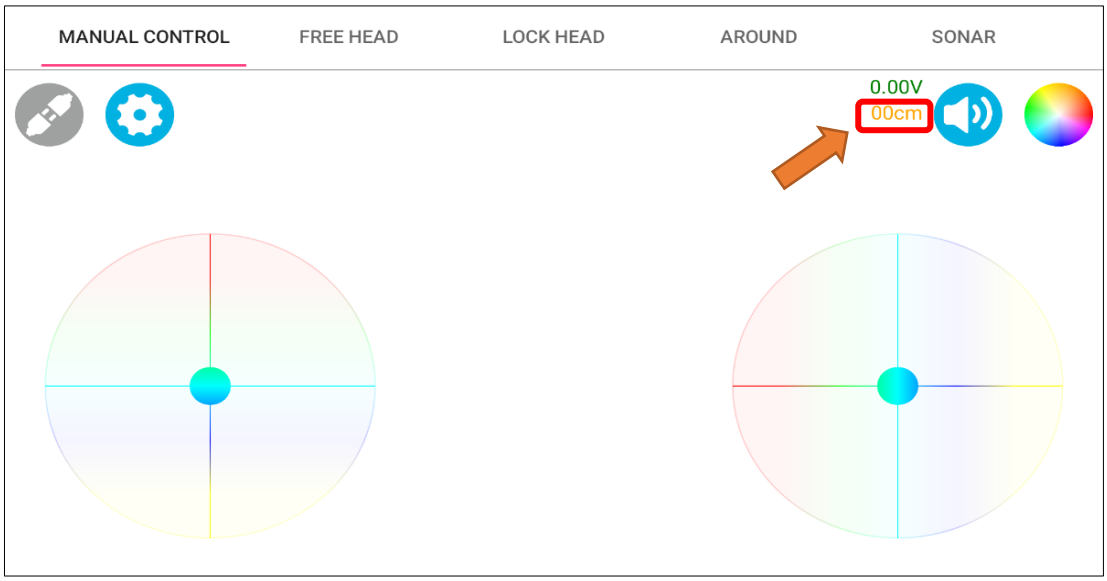
APP Command	Display on APP
P#8123\n	8.123V



CMD_ULTRASONIC_Bluetooth = "E"

The slave device proactively sends data to the master device in the format: E#Distance Value\n (Example: E#22\n). In the APP, this distance value will be displayed, with the unit being centimeters (cm).

APP Command	Display on APP
E#22\n	22cm



CMD_STATE_Bluetooth = "M"

This command changes the car's mode.

Format: M#Mode No.#\n (Example: M#0#\n)

APP Command Table

APP Commands	Modes
M#0#\n	Manual Control
M#1#\n	Free Head Mode
M#2#\n	Lock Head Mode
M#3#\n	Around Mode
M#4#\n	Sonar Mode

CMD_CIRCLE_Bluetooth = "B"

Thi command changes the parameters of Around Mode.

Format: B#circle direction#circle radus#\n (Example: B#0#100#\n)

APP Command Table:

APP Commands	Modes
B#0#100#\n	Clockwise, with a circle radius of 100 cm.
B#1#100#\n	Counterclockwise, with a circle radius of 100 cm.

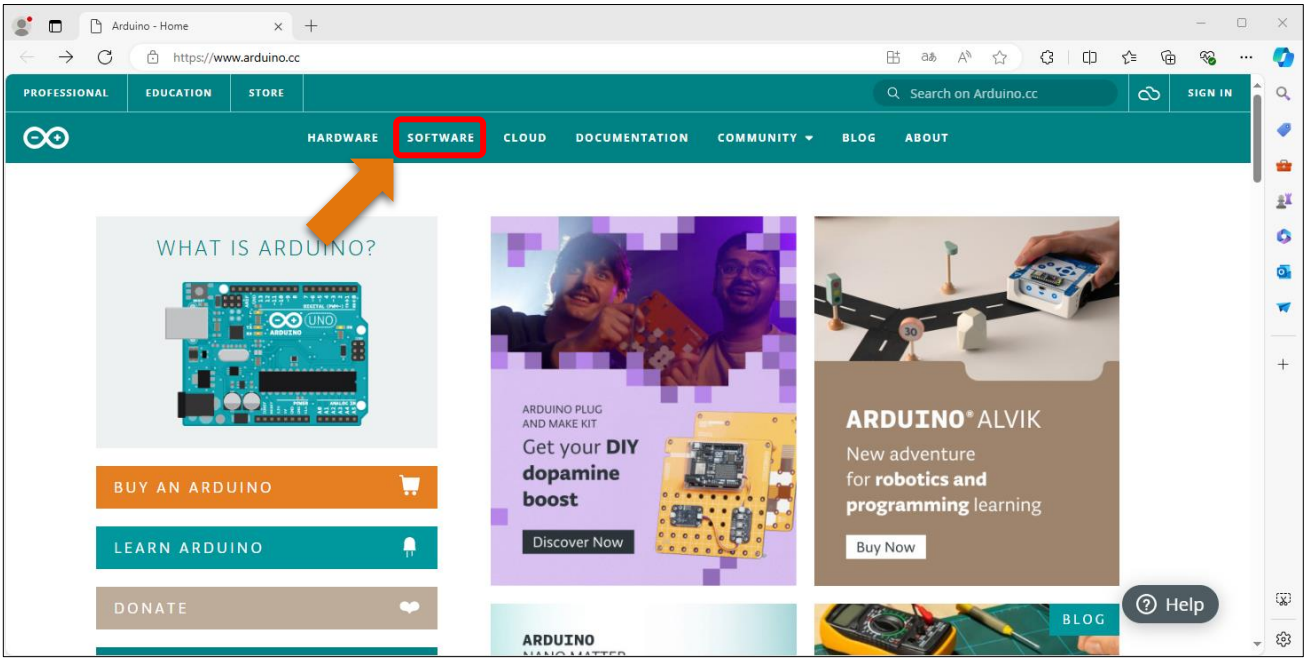
Please note: The circle radius can be changed according to requirements, with 100 cm used as an example in the table.

Chapter 0 Installation of Arduino IDE

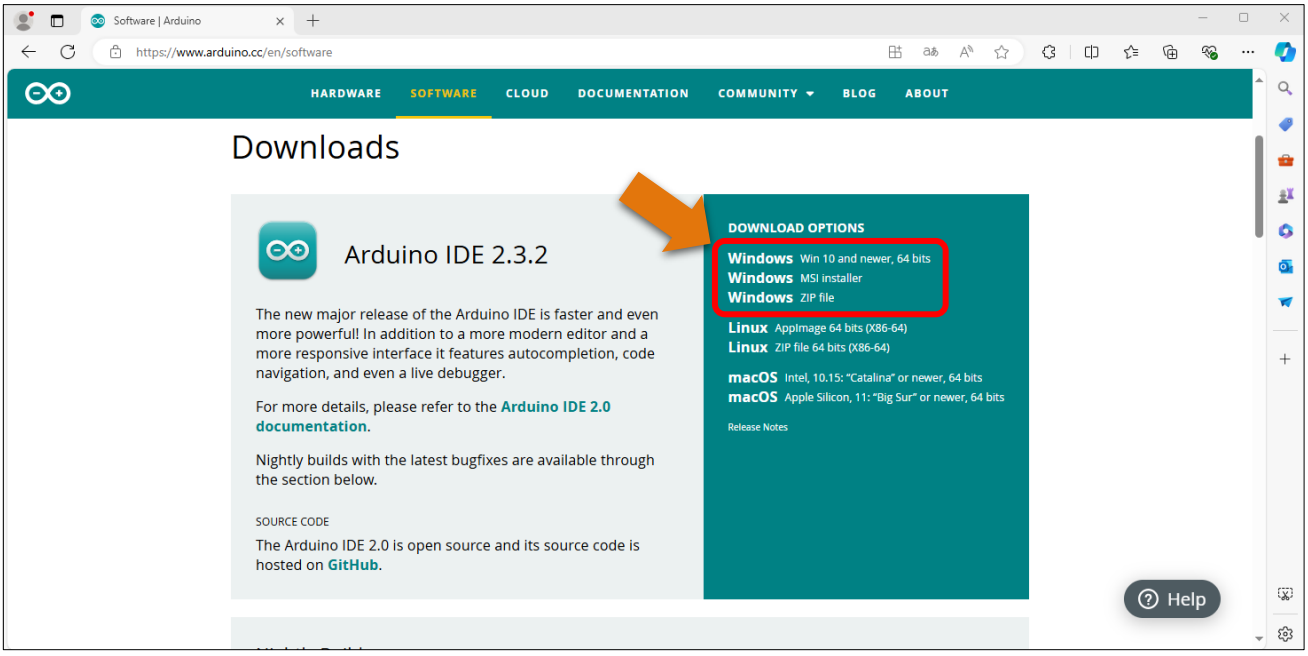
Arduino Software

Arduino Software (IDE) is used to write and upload the code for Arduino Board.

First, install Arduino Software (IDE): visit https://www.arduino.cc, click "**SOFTWARE**" to enter the download page.

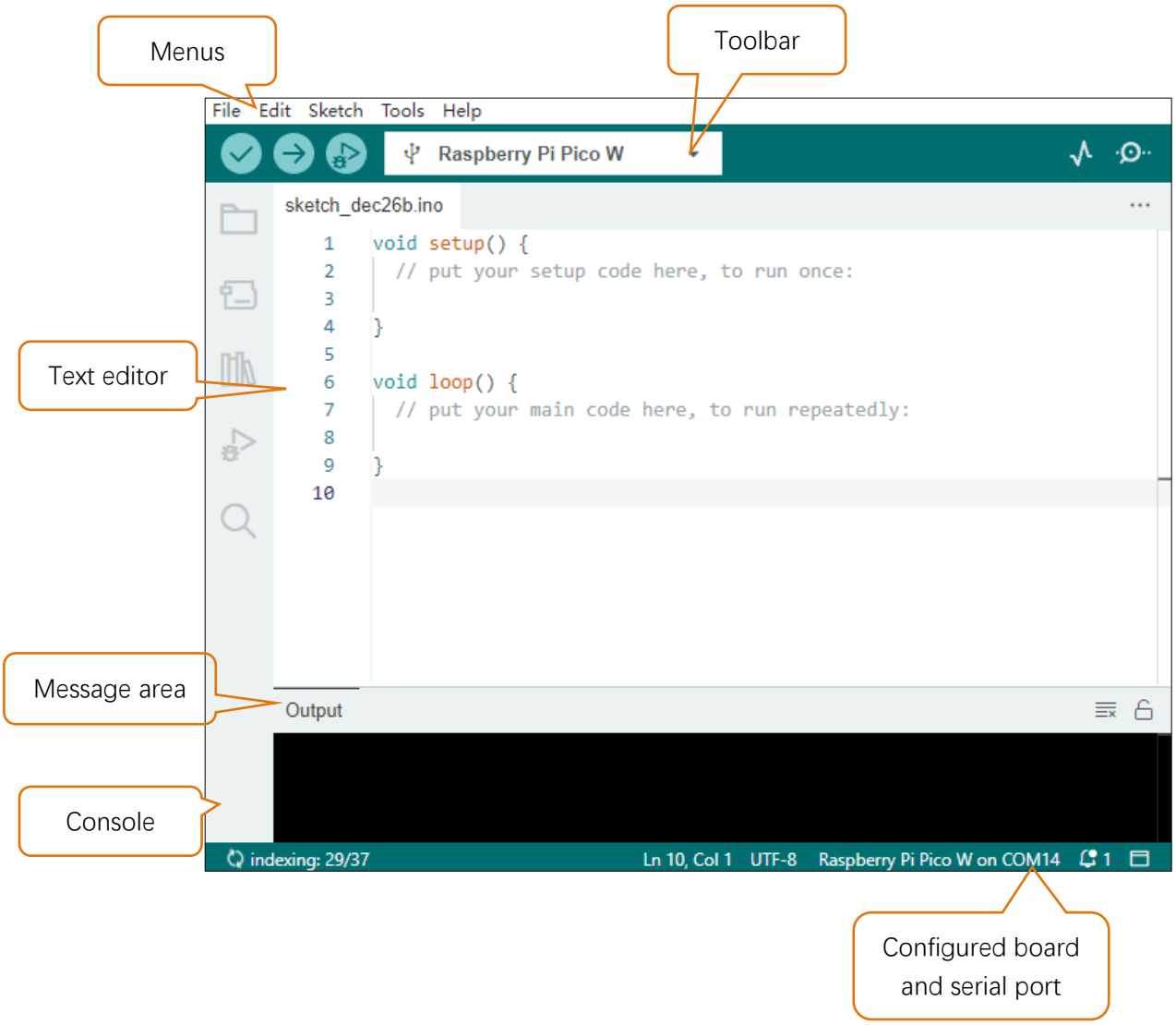


Select and download corresponding installer based on your operating system. If you are a Windows user, please select the "Windows" to download and install the driver correctly.



After the download completes, run the installer. For Windows users, there may pop up an installation dialog during the installation. When it popes up, please allow the installation.
After installation is completed, an Arduino Software shortcut will be generated on the desktop. Run the Arduino Software.

The interface of Arduino Software is as follows:



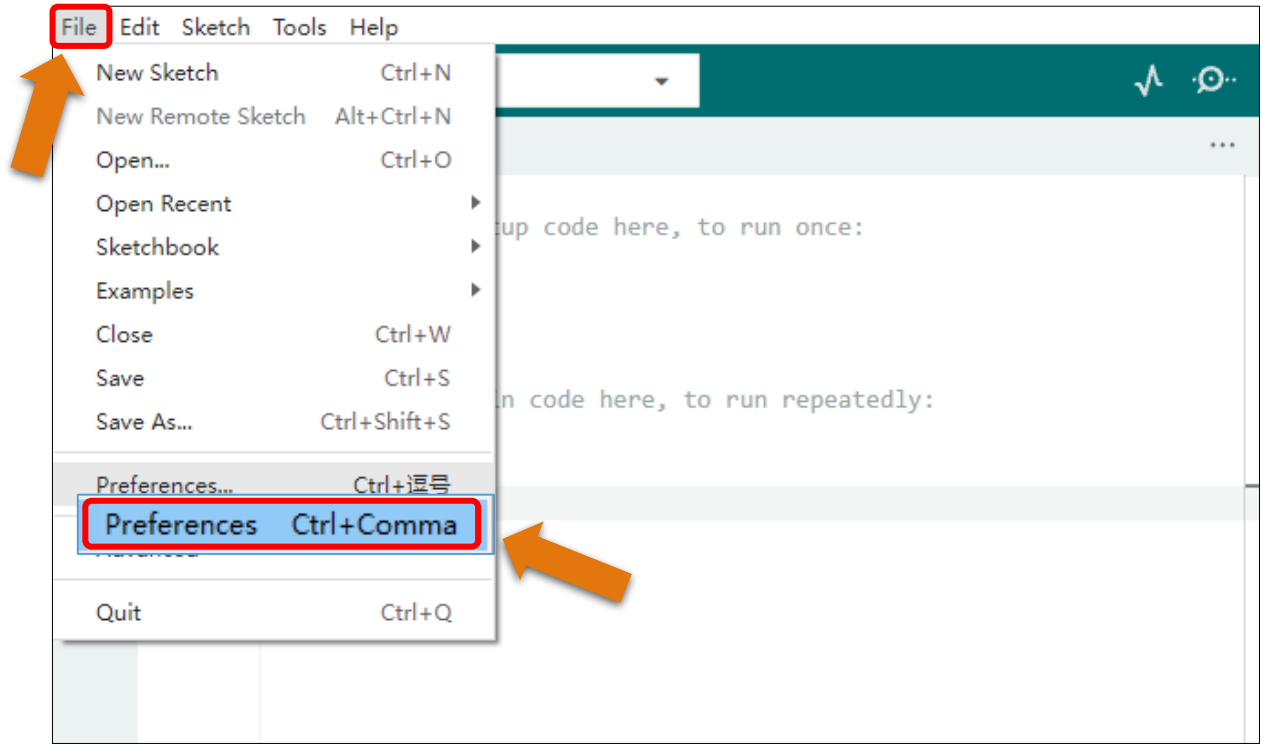
Programs written with Arduino Software (IDE) are called **sketches**. These sketches are written in the text editor and saved with the file extension**.ino**. The editor has features for cutting/pasting and searching/replacing text. The message area gives feedback while saving and exporting and also displays errors. The console displays text output by the Arduino Software (IDE), including complete error messages and other information. The bottom right-hand corner of the window displays the configured board and serial port. The toolbar buttons allow you to verify and upload programs, create, open, and save sketches, and open the serial monitor.

	Verify	Check your code for compile errors.
	Upload	Compile your code and upload them to the configured board.
	Debug	Help you to detect and solve problems in the code
	Plotter	Convert the information output to the serial port into a waveform chart for visual analysis
	Serial Monitor	Open the serial monitor.

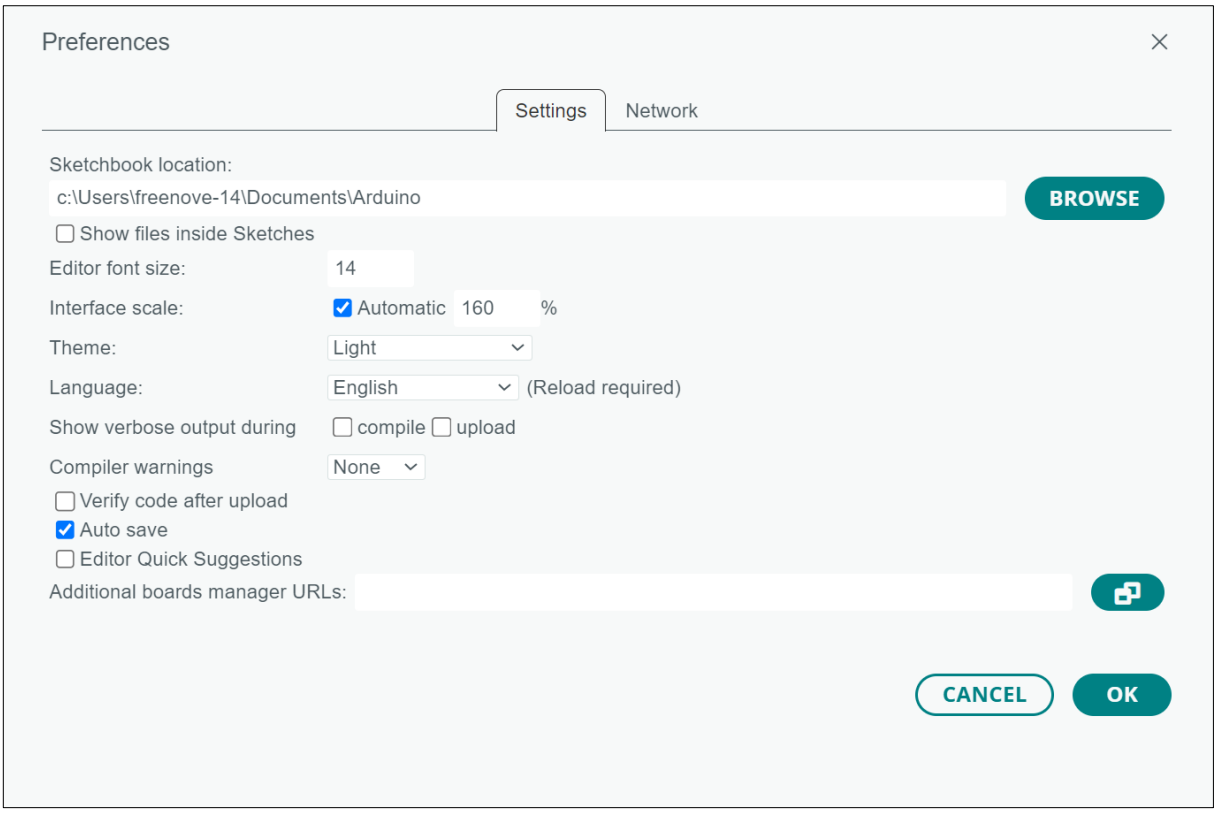
Additional commands are found within the five menus: File, Edit, Sketch, Tools, Help. The menus are context sensitive, which means only those items relevant to the work currently being carried out are available.

Environment Configuration

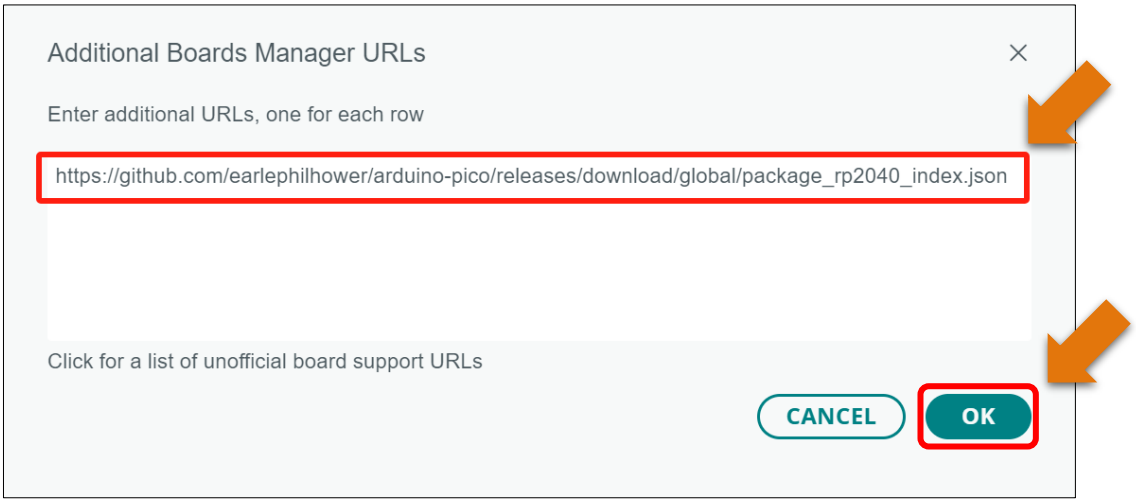
First, open the software platform arduino, and then click **File** in Menus and select **Preferences**.

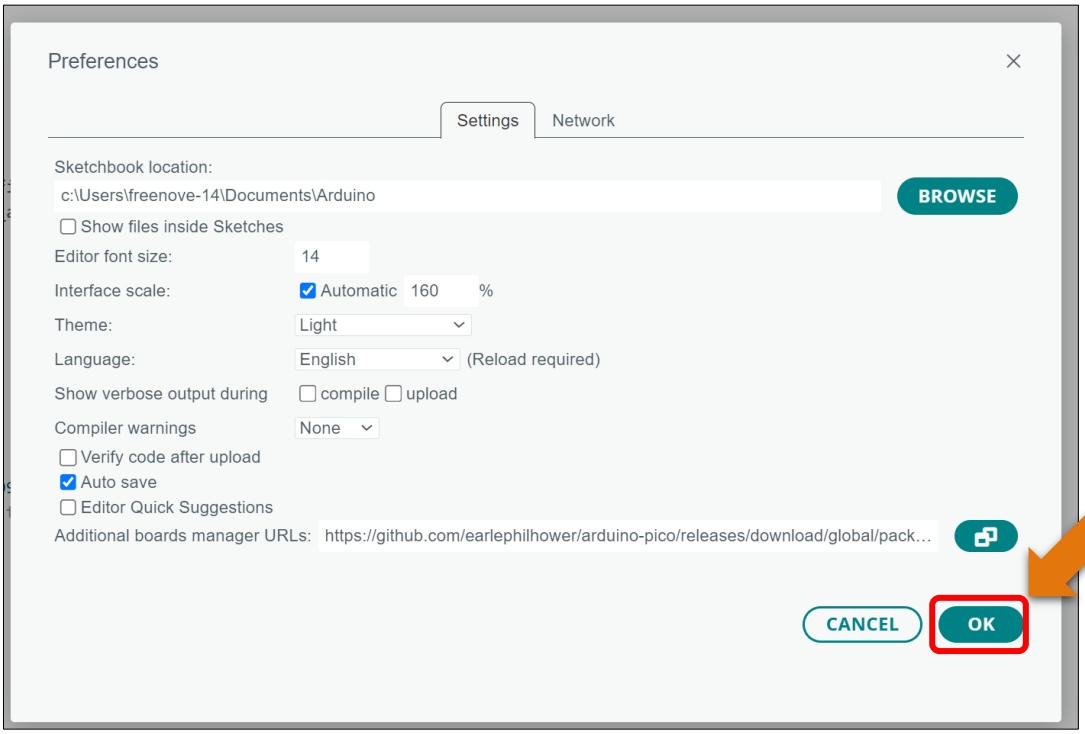


Second, click on the symbol behind "Additional Boards Manager URLs"

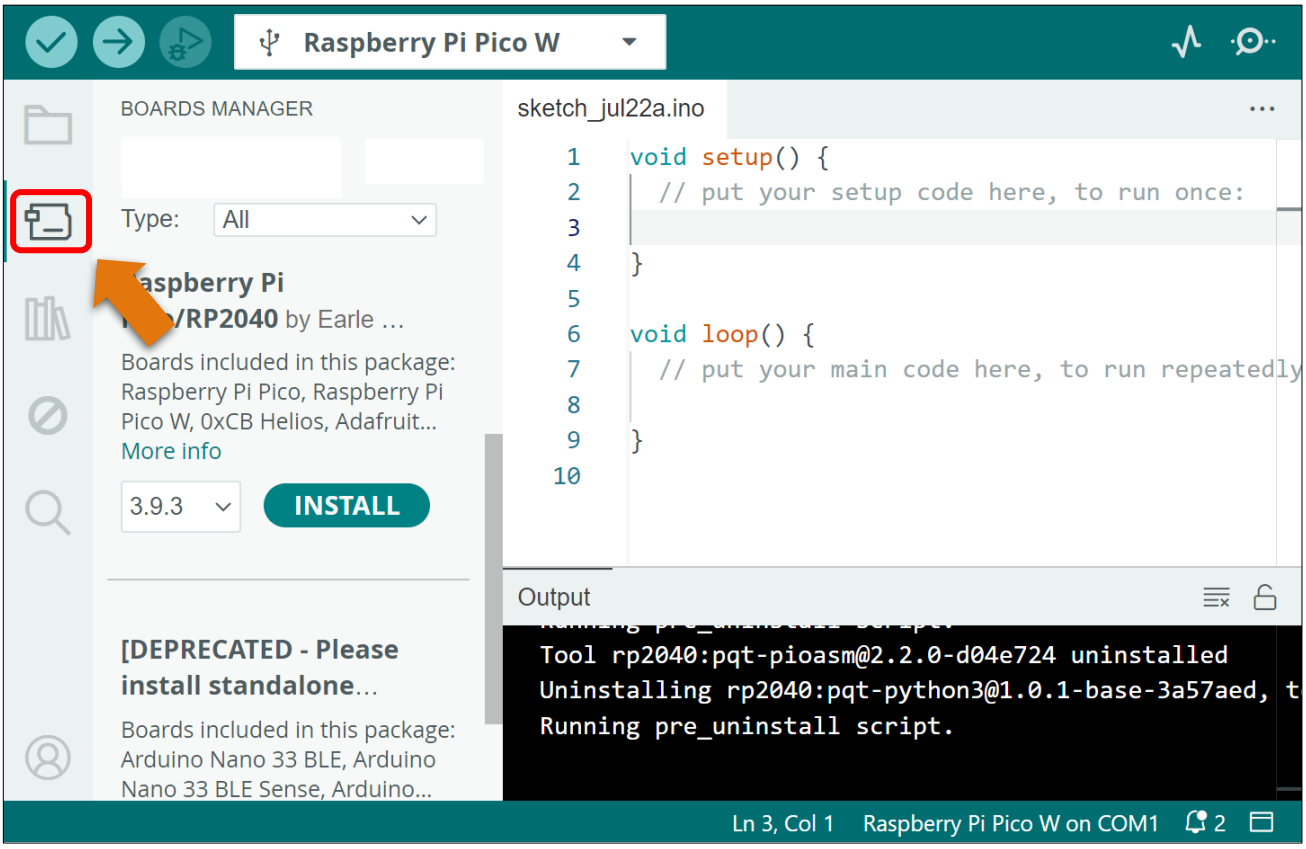


Third, fill in https://github.com/earlephilhower/arduino-pico/releases/download/global/package_rp2040_index.json in the new window, click **OK**, and click **OK** on the Preferences window again.

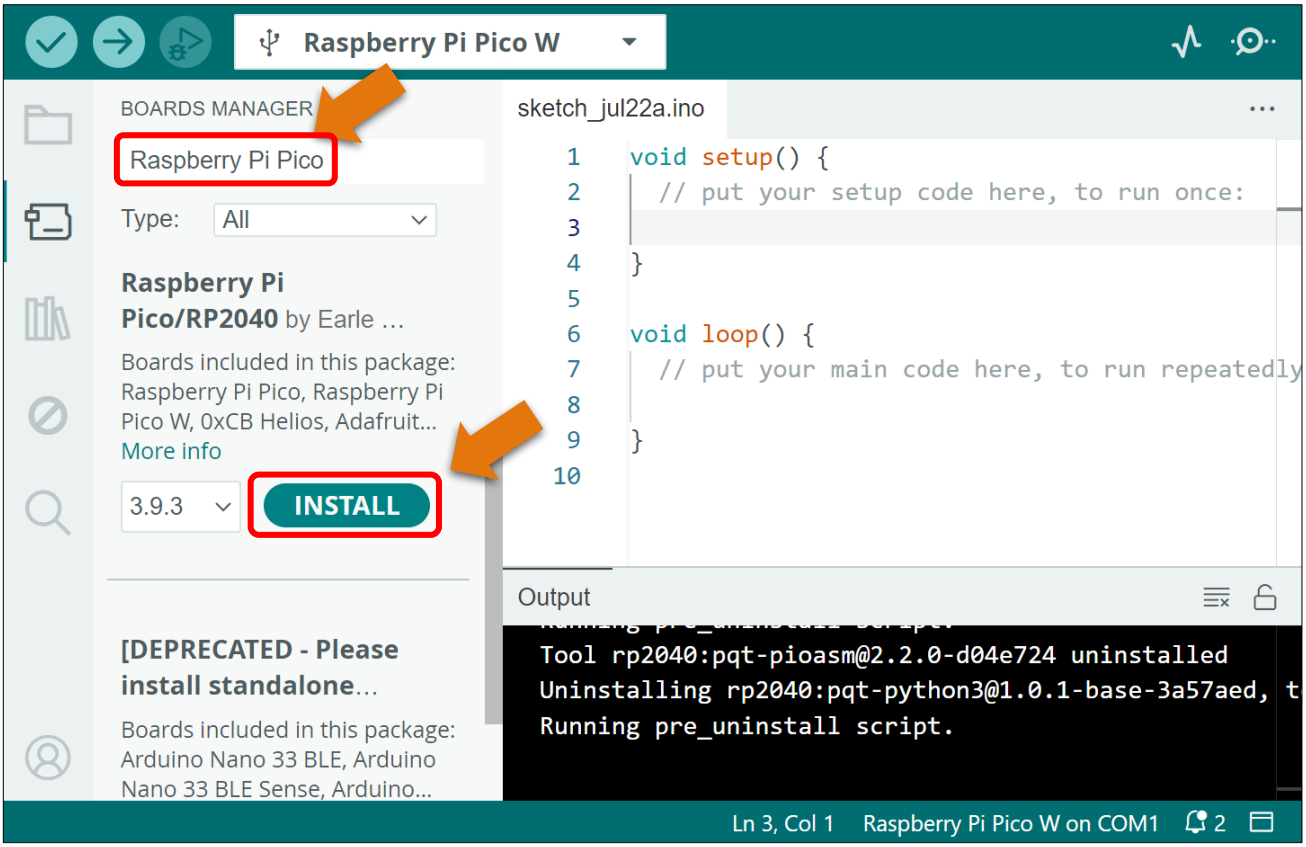



Preferences
Settings
Network
Sketchbook location:
c:\Users\freenove-14\Documents\Arduino
BROWSE
Show files inside Sketches
Editor font size:
14
Interface scale:
Automatic
160
%
Theme:
Light
Language:
English
(Reload required)
Show verbose output during
compile
upload
Compiler warnings
None
Verify code after upload
Auto save
Editor Quick Suggestions
Additional boards manager URLs:
https://github.com/earlephilhower/arduino-pico/releases/download/global/pack…
CANCEL
OK

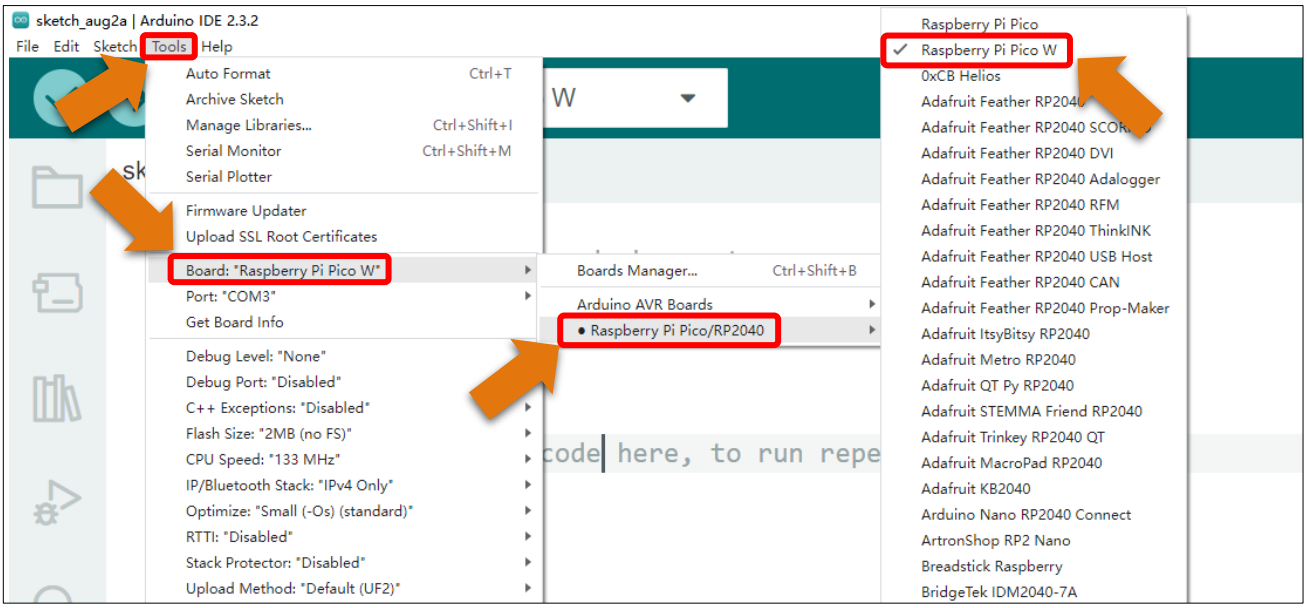
Fourth, click "**Boards Manager**".



Fifth, input "Raspberry Pi Pico" in the window below and press Enter, click "**Install**" to install.



When finishing installation, click **Tools** in the Menus again and select **Board: "Raspberry Pi Pico/RP2040"**, and then you can see information of Raspberry Pi Pico (W). Click "**Raspberry Pi Pico W**" so that the Raspberry Pi Pico W programming development environment is configured.

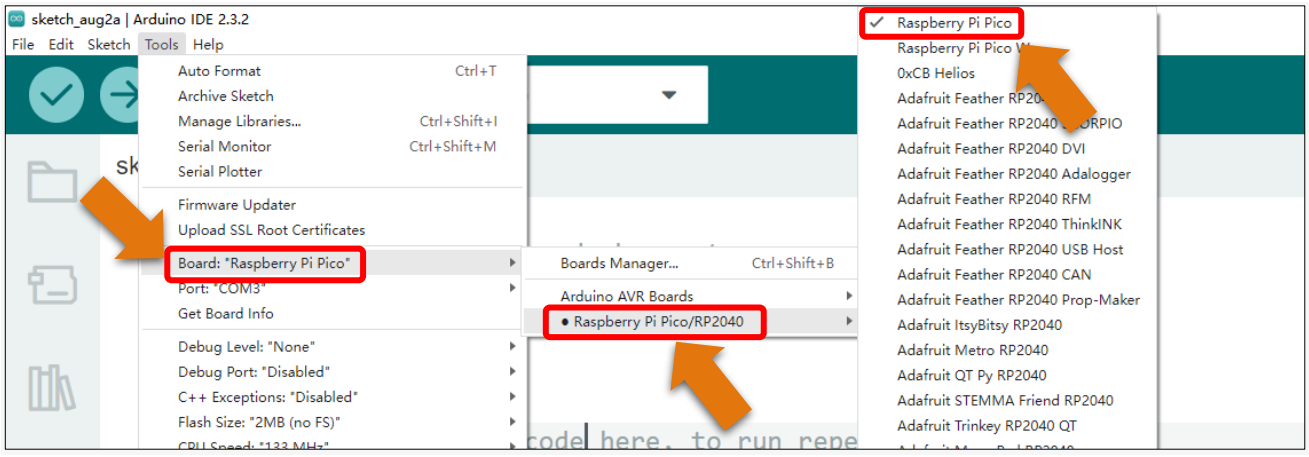


Additional Remarks

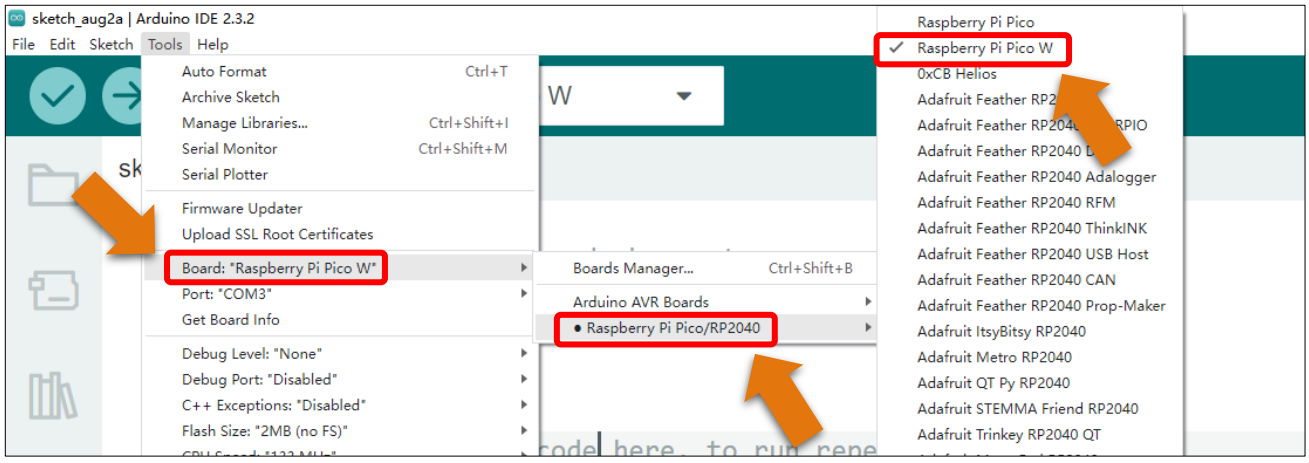
To finish this tutorial, you can use a Raspberry Pi Pico W or a Raspberry Pi Pico. The two boards have the same form factor, and their only difference is that Pico W features with an onboard single-band 4.802GHz wireless interface using Infineon CYW2. That is to say, the hardware Raspberry Pi Pico W is almost the same as the normal pico except for the wireless interface, which enables Pico W to work with WiFi
No matter which board you use, the procedure for each project is the same. You only need to select the correct board and upload the corresponding code based on the board you use.

In this book, we use Raspberry Pi Pico W to illustrate the operation.

The followings show the configuration on Arduino for the use of each board.
For Raspberry Pi Pico:
Click **Tools** on Menu bar, click **Board**, select "**Raspberry Pi Pico/RP2040**", and select **Raspberry Pi Pico**.



For Raspberry Pi Pico W:
Click **Tools** on Menu bar, click **Board**, select "**Raspberry Pi Pico/RP2040**", and select **Raspberry Pi Pico W**.



Chapter 1 Car Assembly

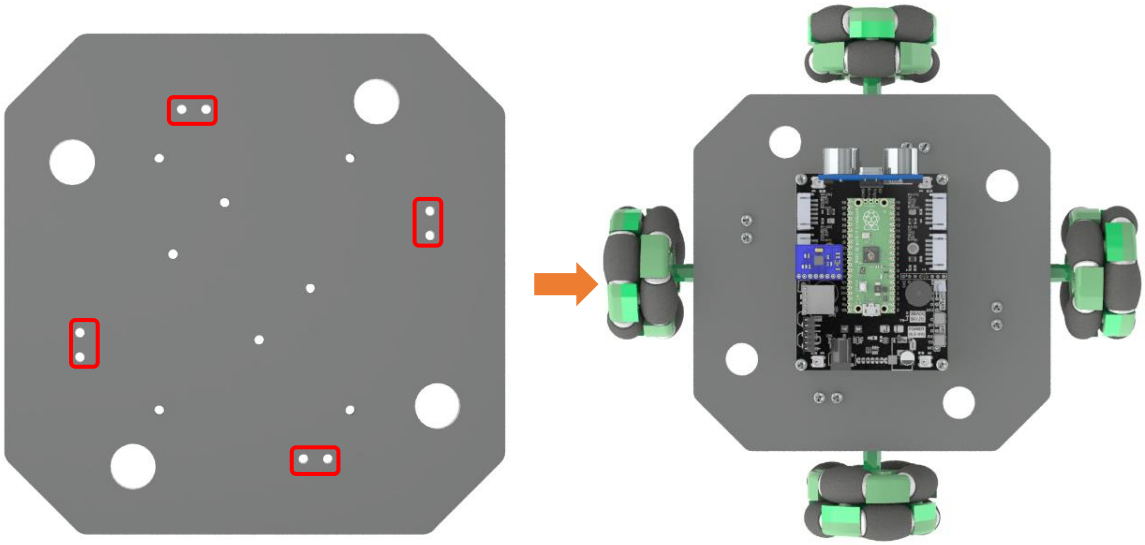
Before assembling the car, please check the part list to ensure they are complete.
If you have any concerns, please feel free to contact us via support@freenove.com

Assembling the Car

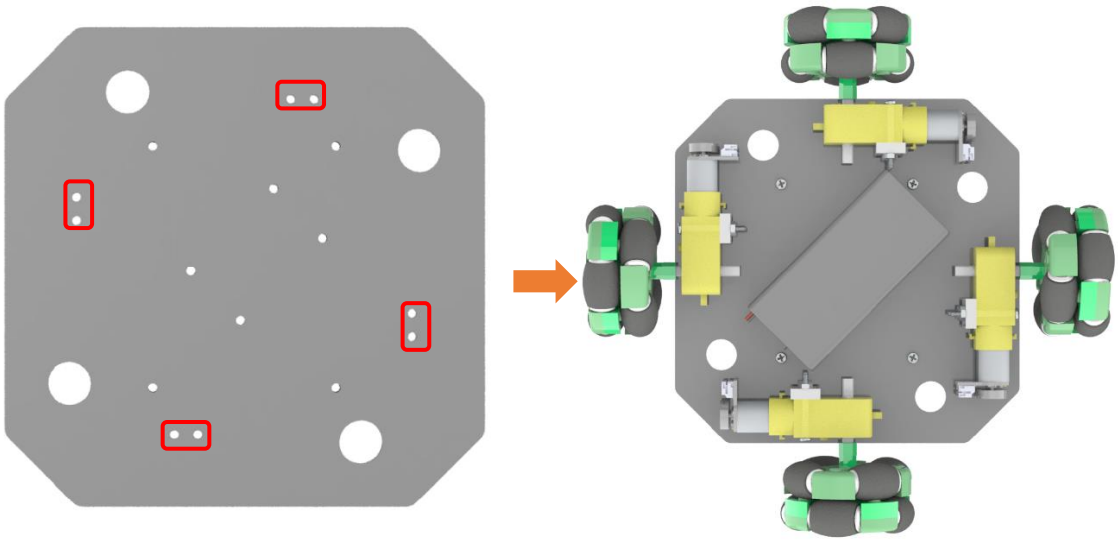
Distinguish the top and bottom sides of the car chassis.

The bottom plate of the car has a distinction between the top and bottom, which can be distinguished through the holes on the car. Here is a schematic diagram:

Top:

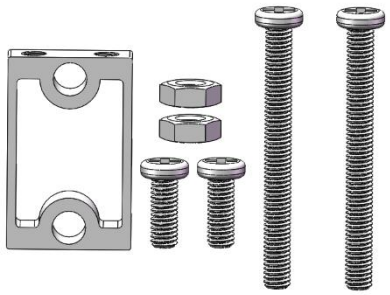


Bottom:



Installing Motor and Wheels

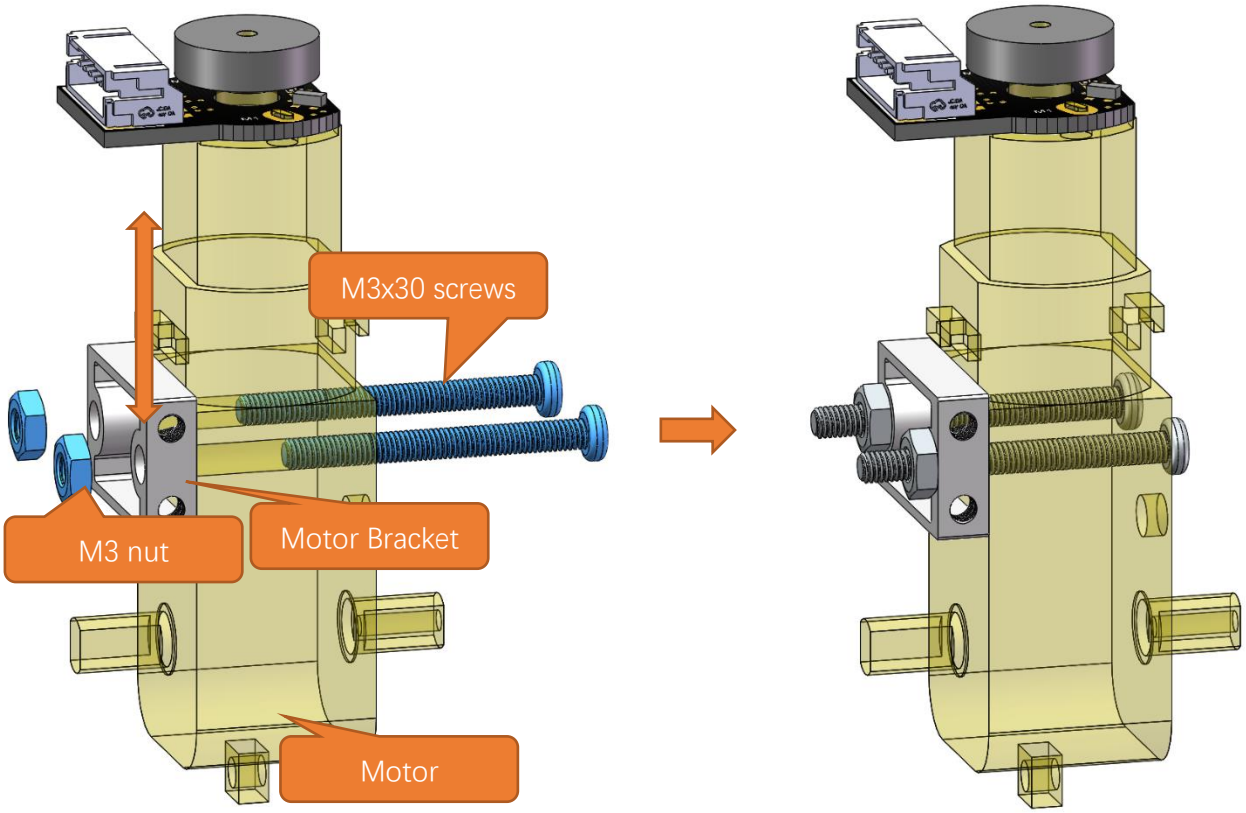
There are 4 bracket packages to fix motors, each contain an aluminum bracket, two M3*30 screws, two M3*8 screws and two M3 nuts. The parts of each set are as shown below:



Installation steps:

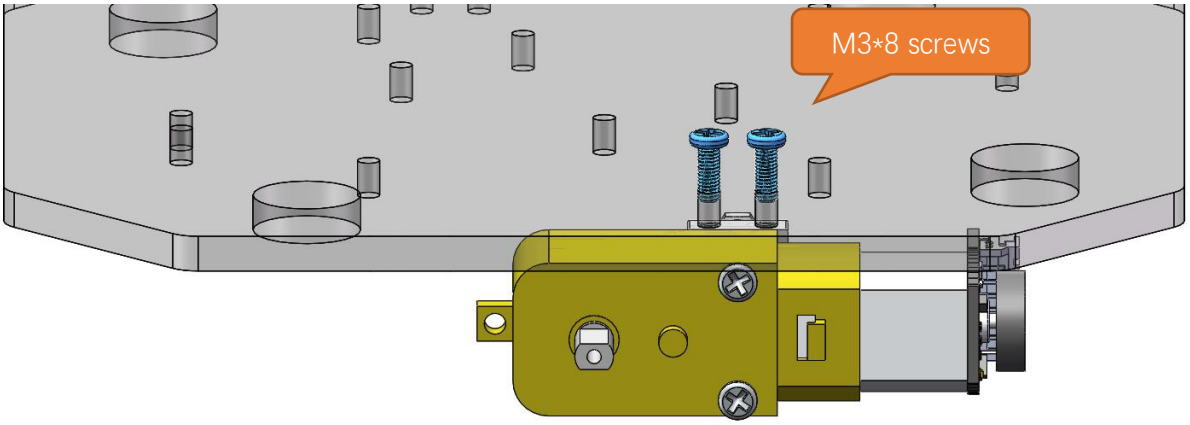
Step 1 Motor Assembly

Mount the motor bracket to the motor with two M3x30 screws and two M3 nuts. Please note that the connector of the encoder should be at the same side of the motor bracket, as shown below:

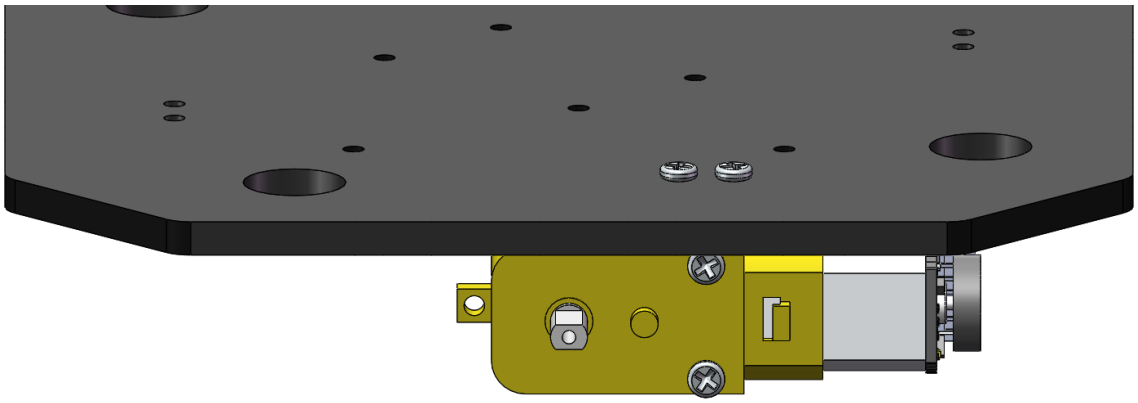


Step 2: Mount the motors to the bottom plate with two M3x8 screws.

Please note that the motor bracket should be inward and the motor axis is facing out at the center of the board edge, as shown below:

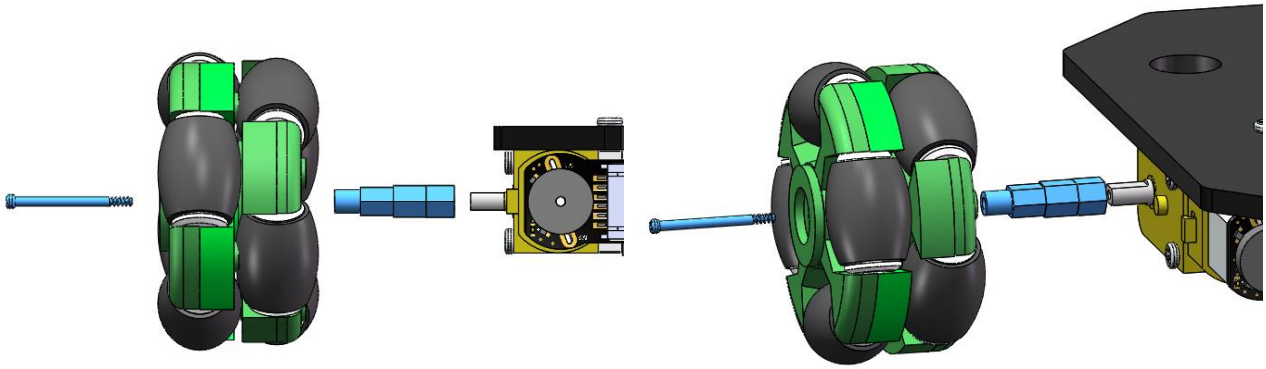


After assembly:

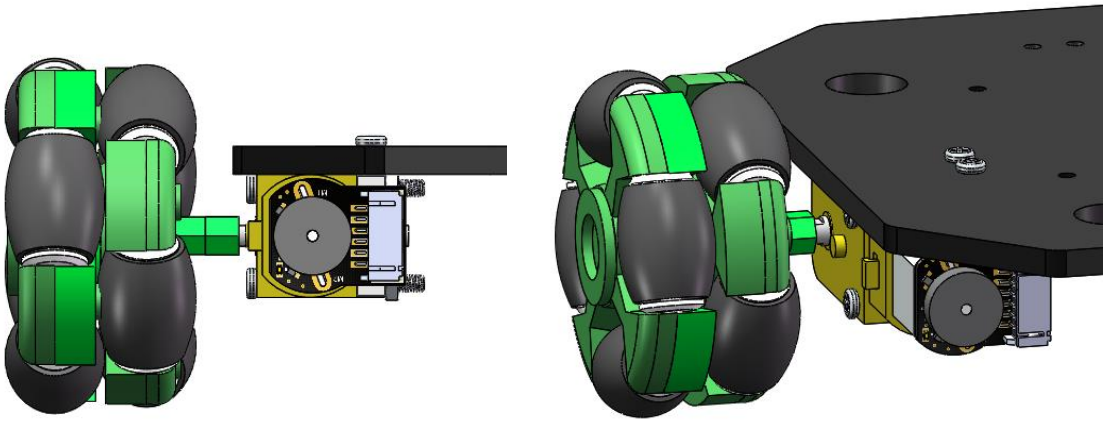


Step 3: Wheel assembly

Fix the omni wheels to the car with the accompanying coupling and screw.

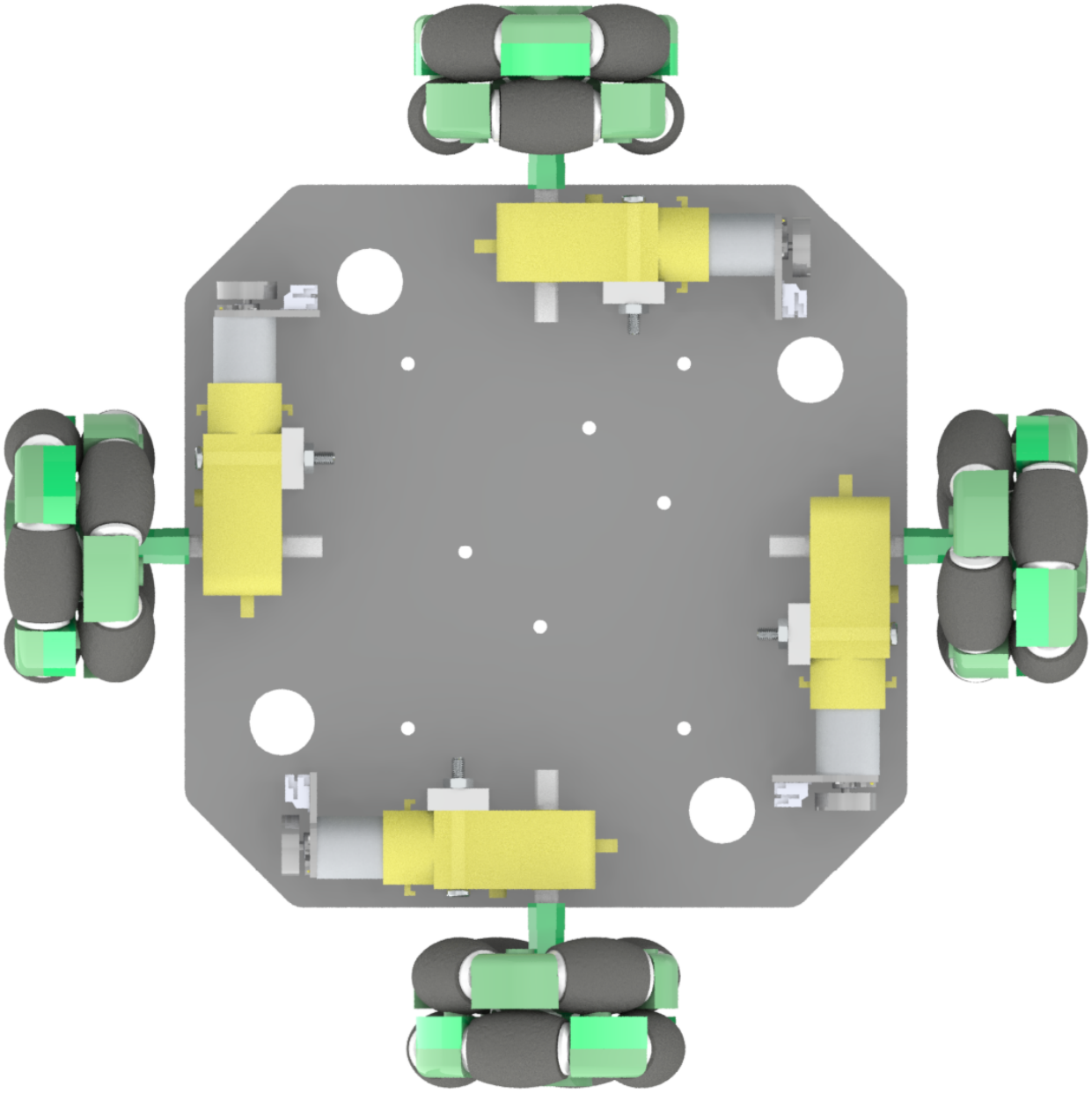


After assembly:



Step 4: Repeat the above steps to assemble the rest motors and wheels.

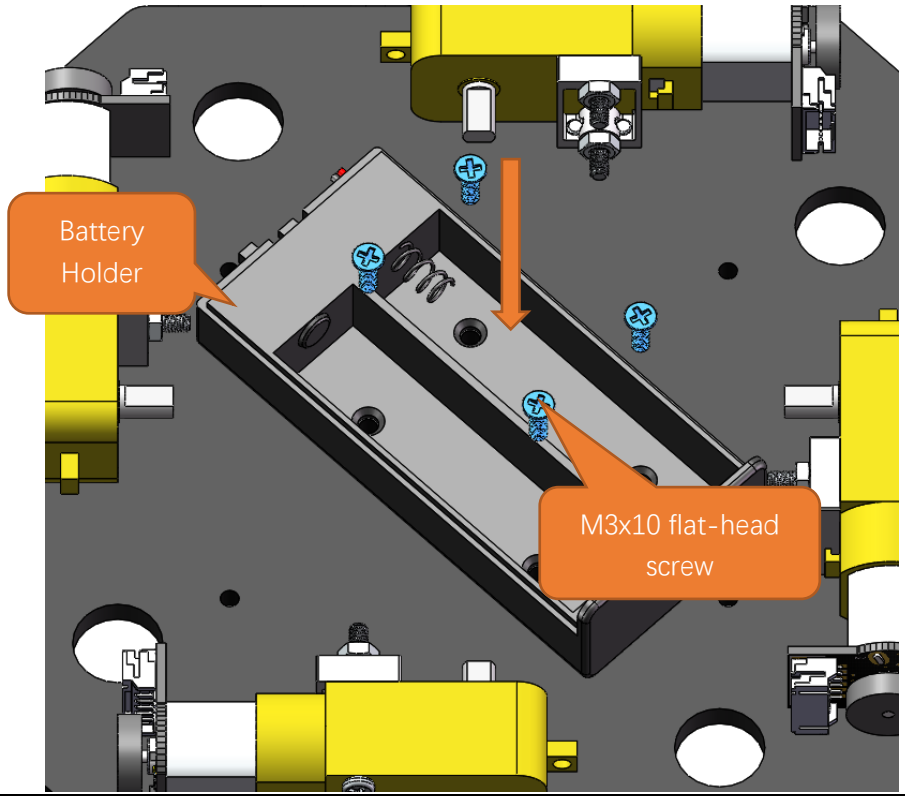
After assembly:



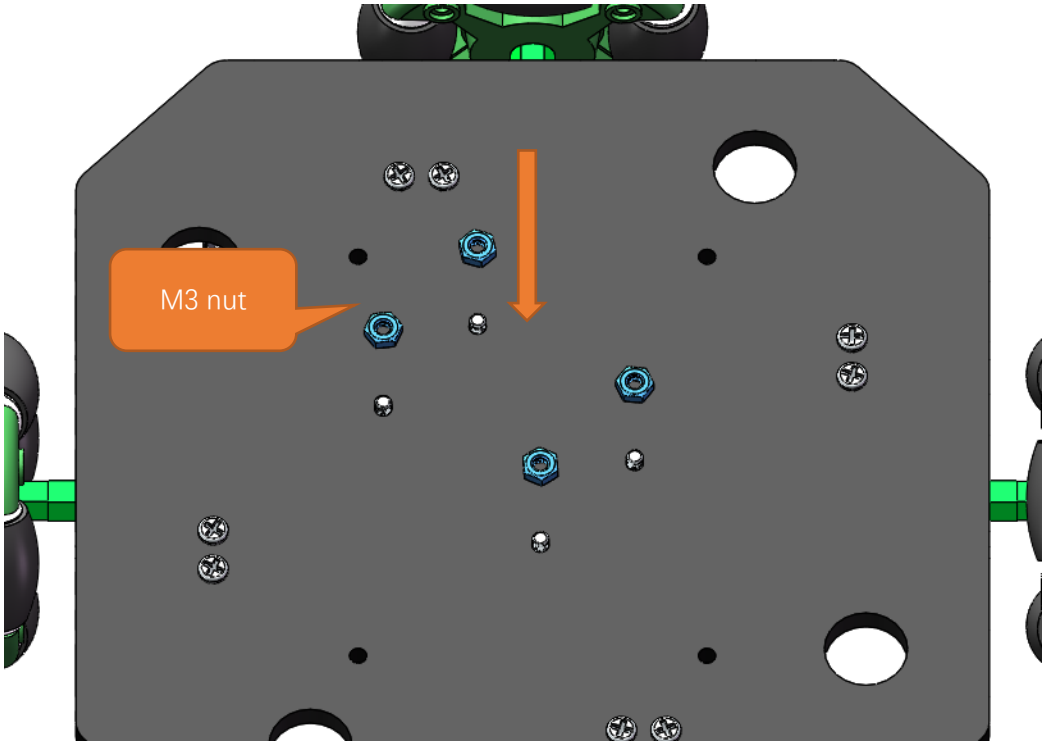
Battery Holder Assembly

Step 1: Battery holder assembly

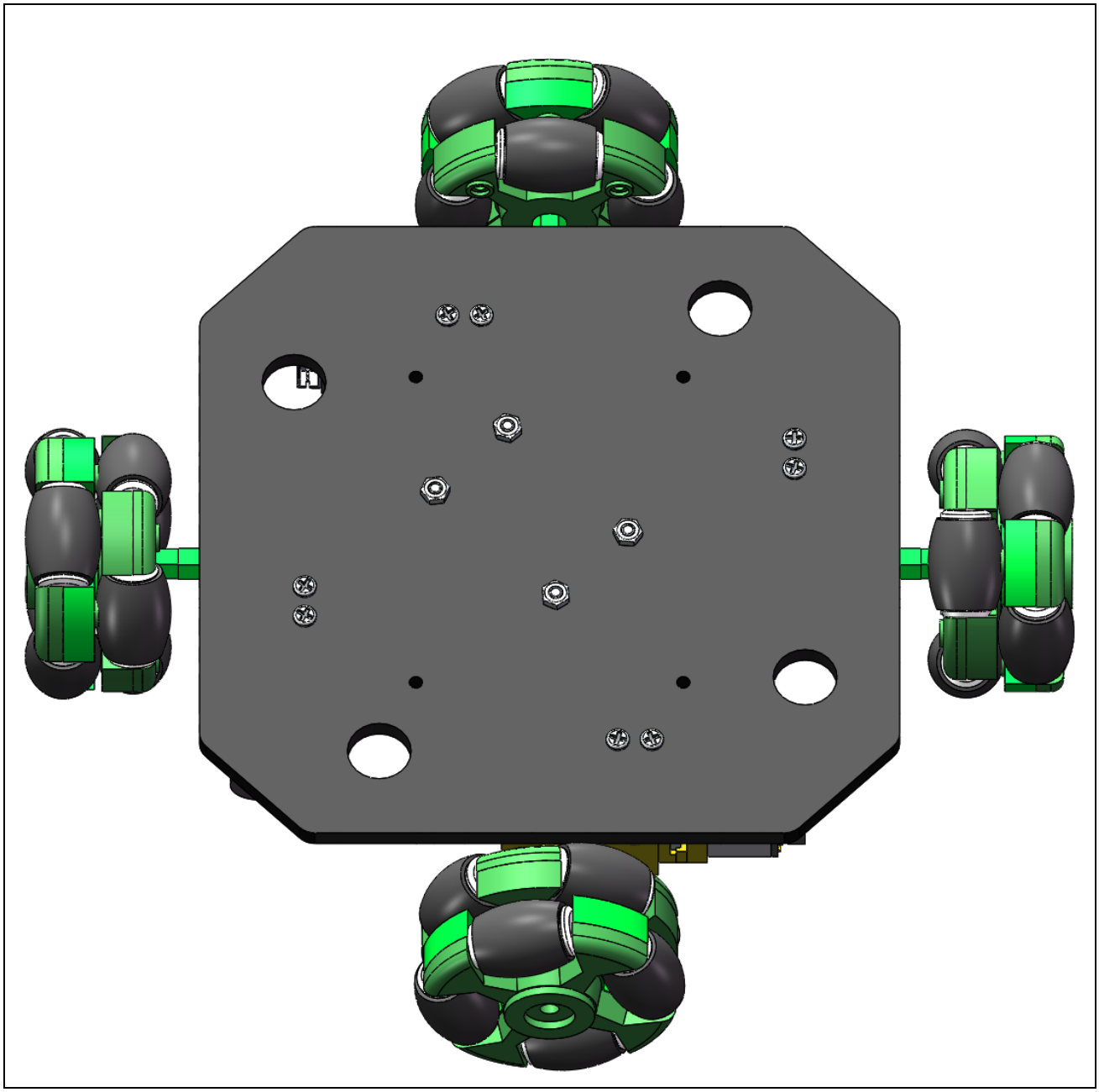
1. Put the battery holder on the back of the car chassis.
2. Mount the battery holder to the plate with four M3x10 flat-head screws.



Step2: Fix it with four M3 nuts.



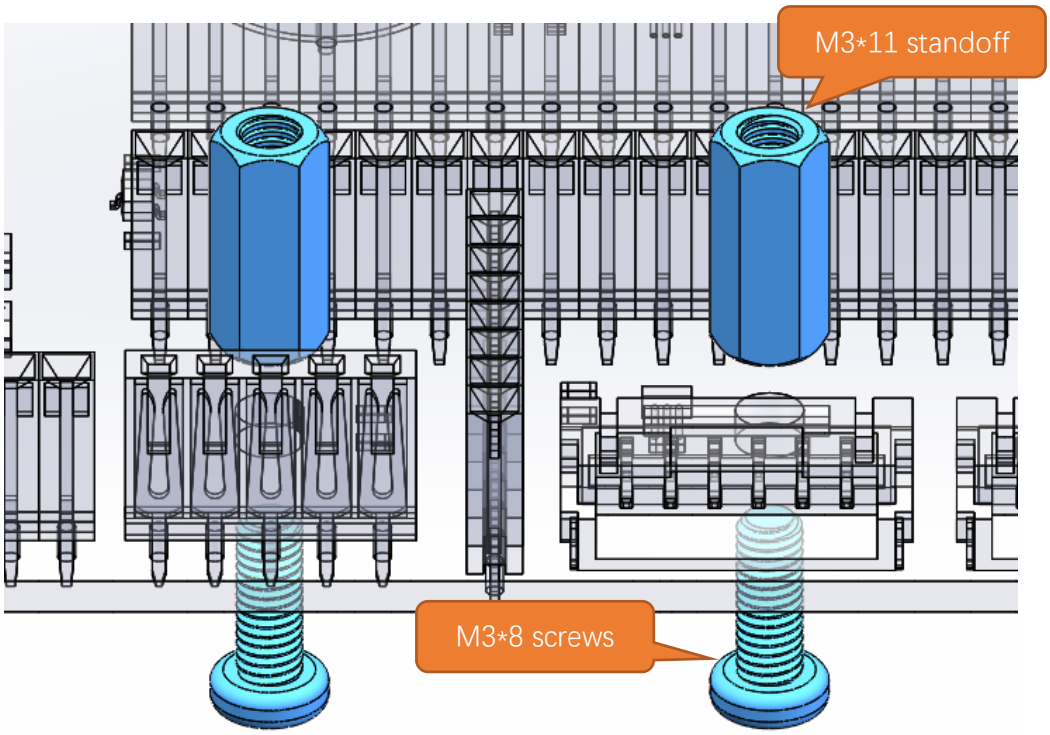
After assembly:

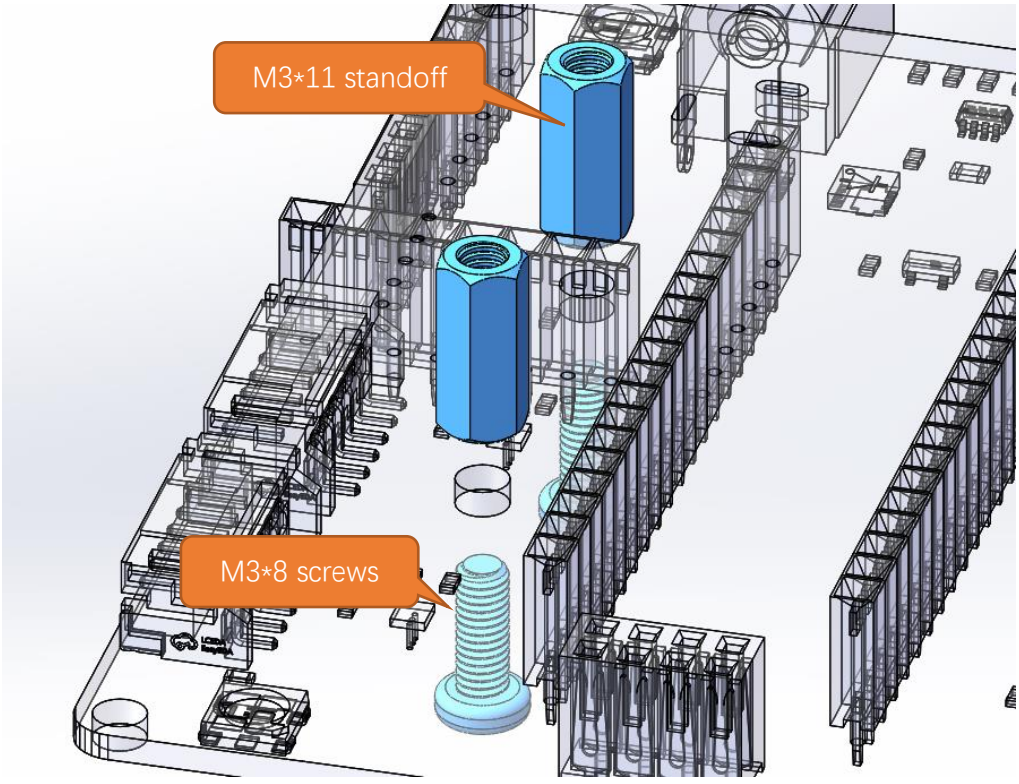


Expansion Board Assembly

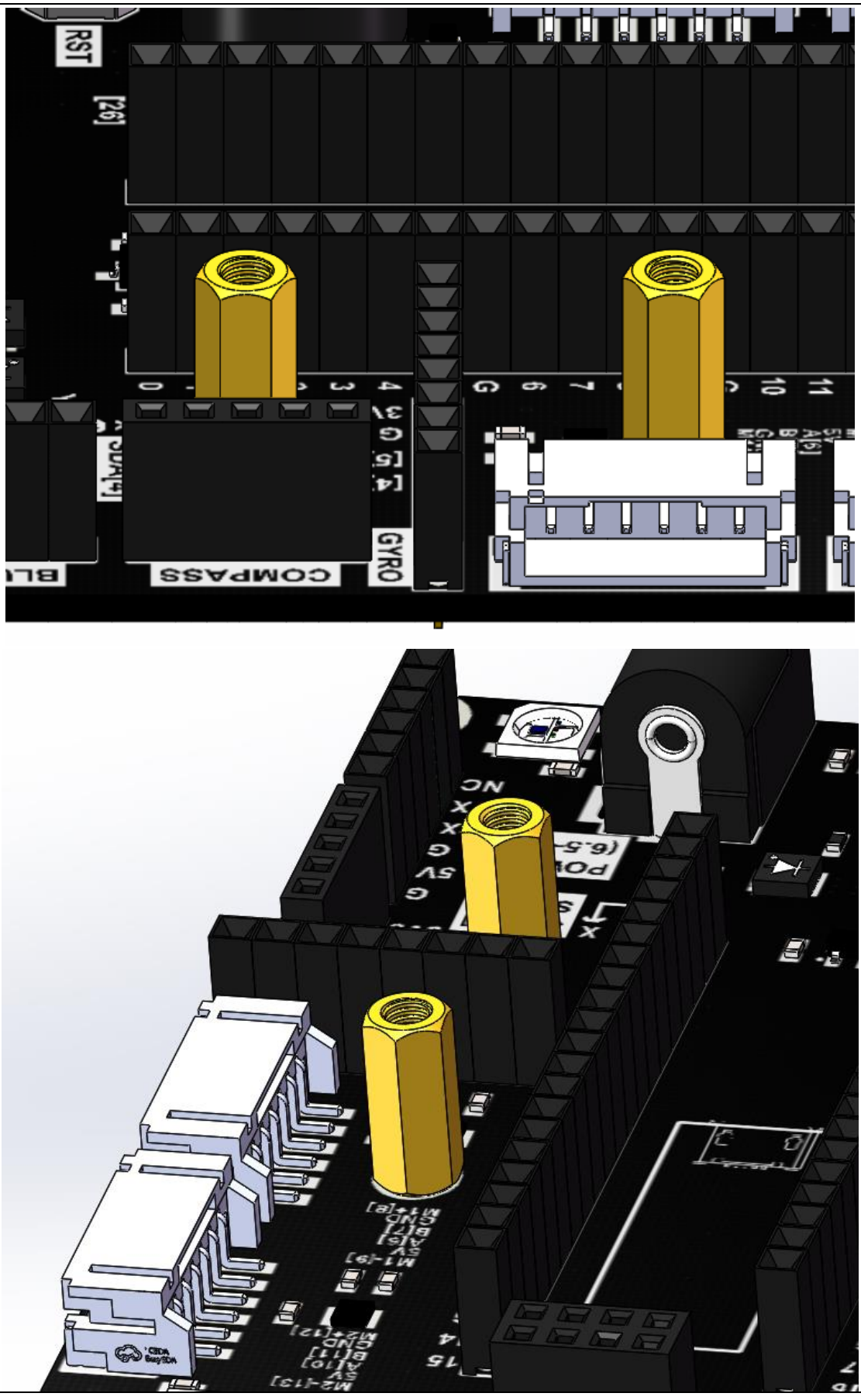
Step 1: Standoffs Installation

1. Put two M3x11 standoffs on the expansion board.
2. Fix the standoffs with two M3x8 screws.

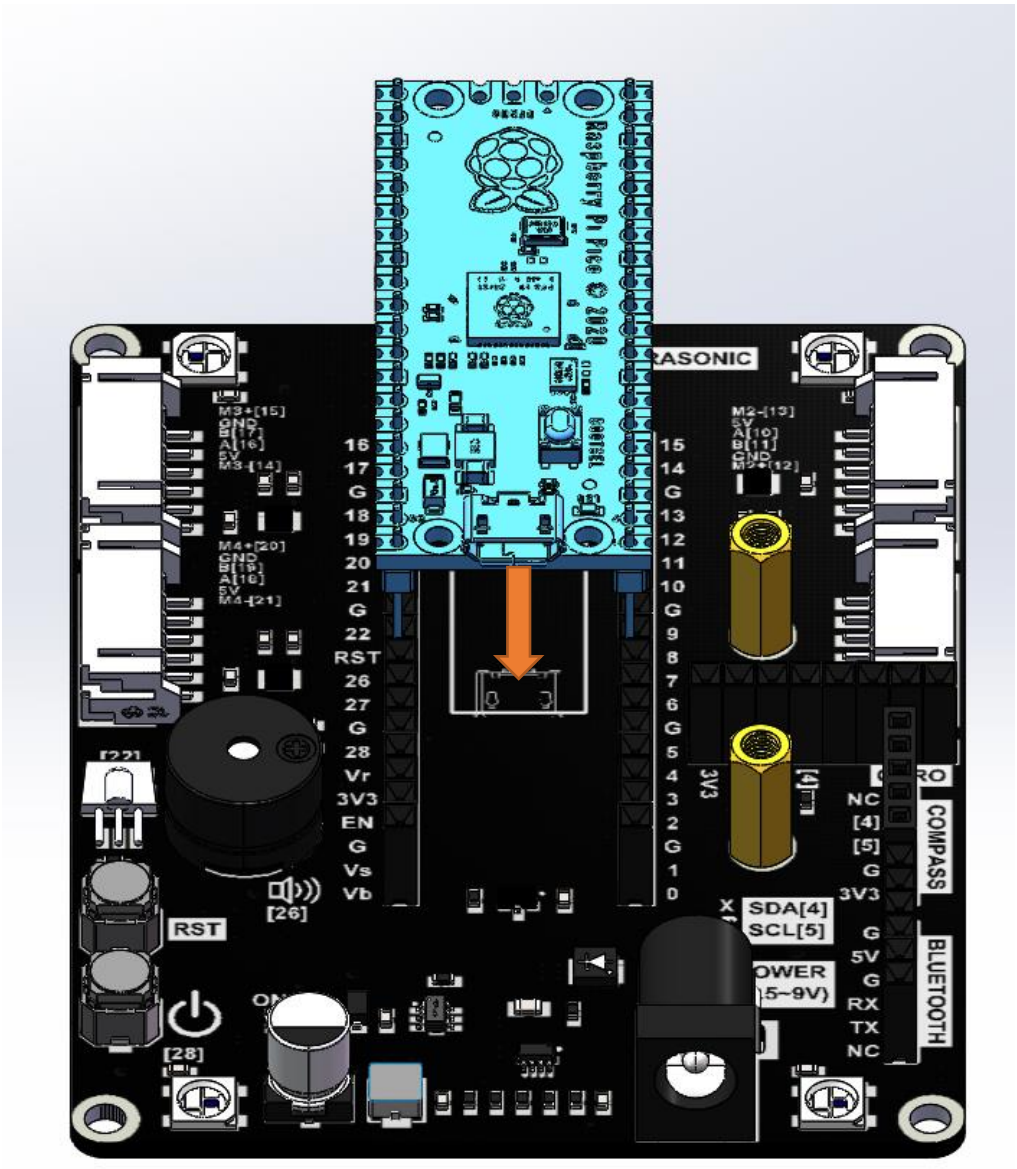


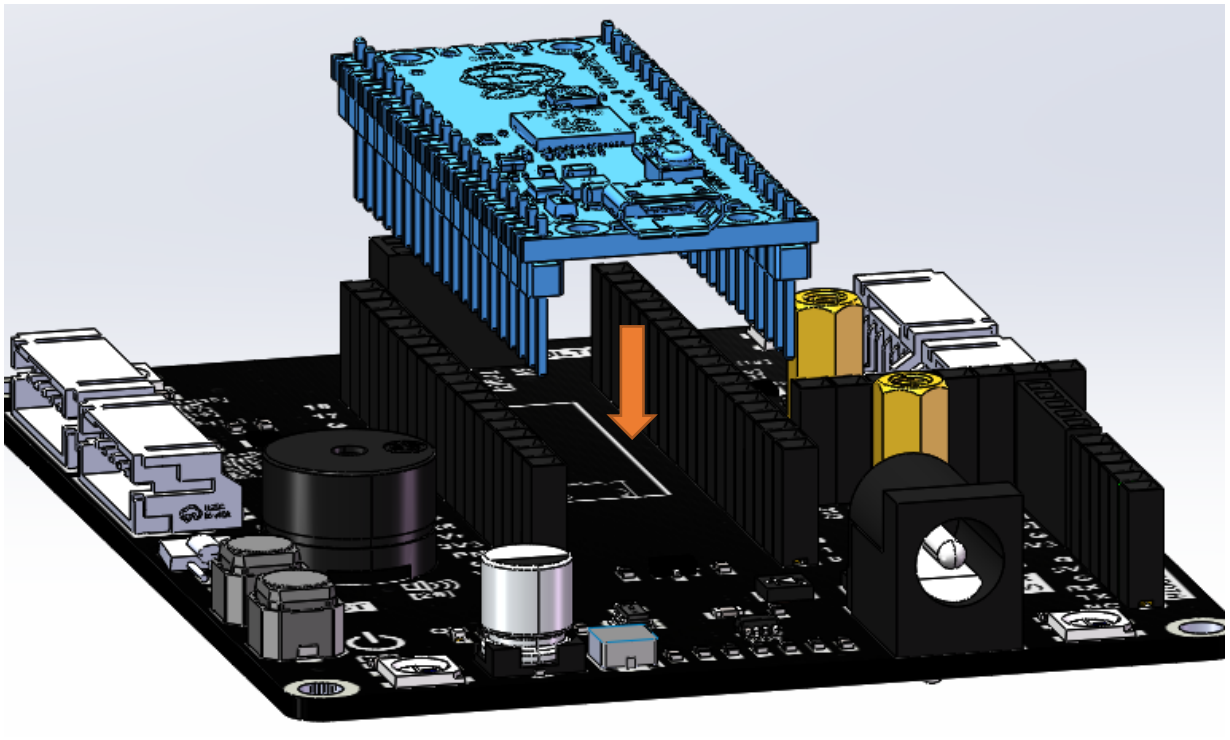


After assembly:

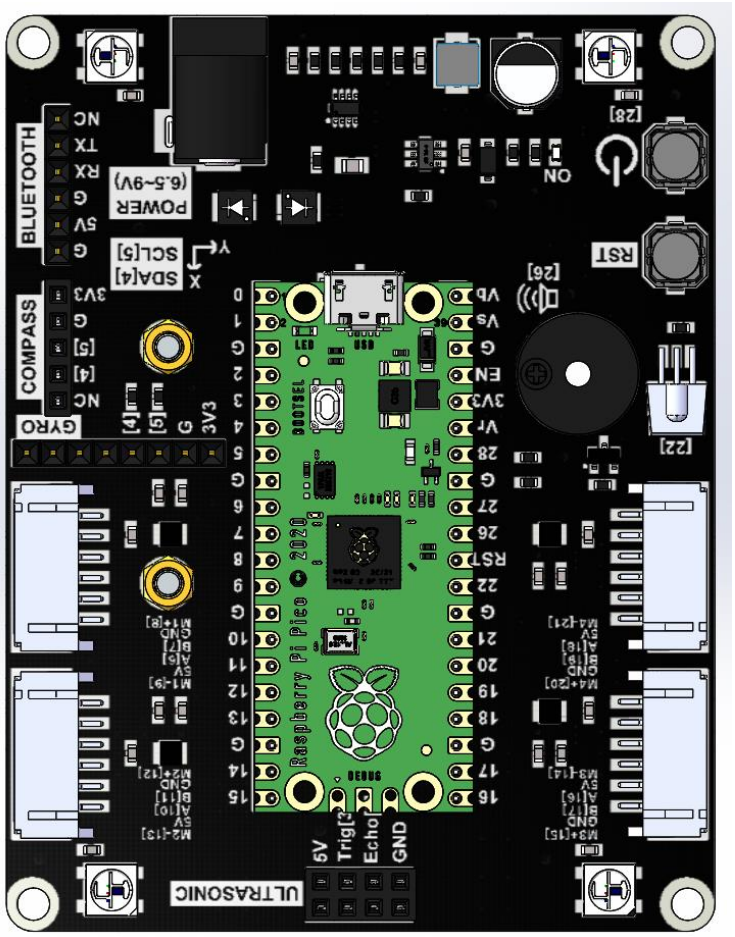


Step 2: Plug in the Raspberry Pi Pico (or Raspberry Pi Pico W/ Raspberry Pi Pico 2)


Raspberry Pi Pico © 2020
RASONIC
M3+[15]
GND
B[17]
A[16]
5V
M3-[14]
M4+[20]
GND
B[19]
A[18]
5V
M4-[21]
M2-[13]
5V
A[10]
B[11]
GND
M2+[12]
16
17
G
18
19
20
21
G
22
RST
26
27
G
28
Vr
3V3
EN
G
Vs
Vb
15
14
G
13
12
11
10
G
9
8
7
6
G
5
4
3
2
G
1
0
3V3
[4]
NC
[4]
[5]
G
3V3
G
5V
G
RX
TX
NC
COMPASS
BLUETOOTH
SDA[4]
SCL[5]
OWER
5~9V)
[22]
[26]
RST
[28]



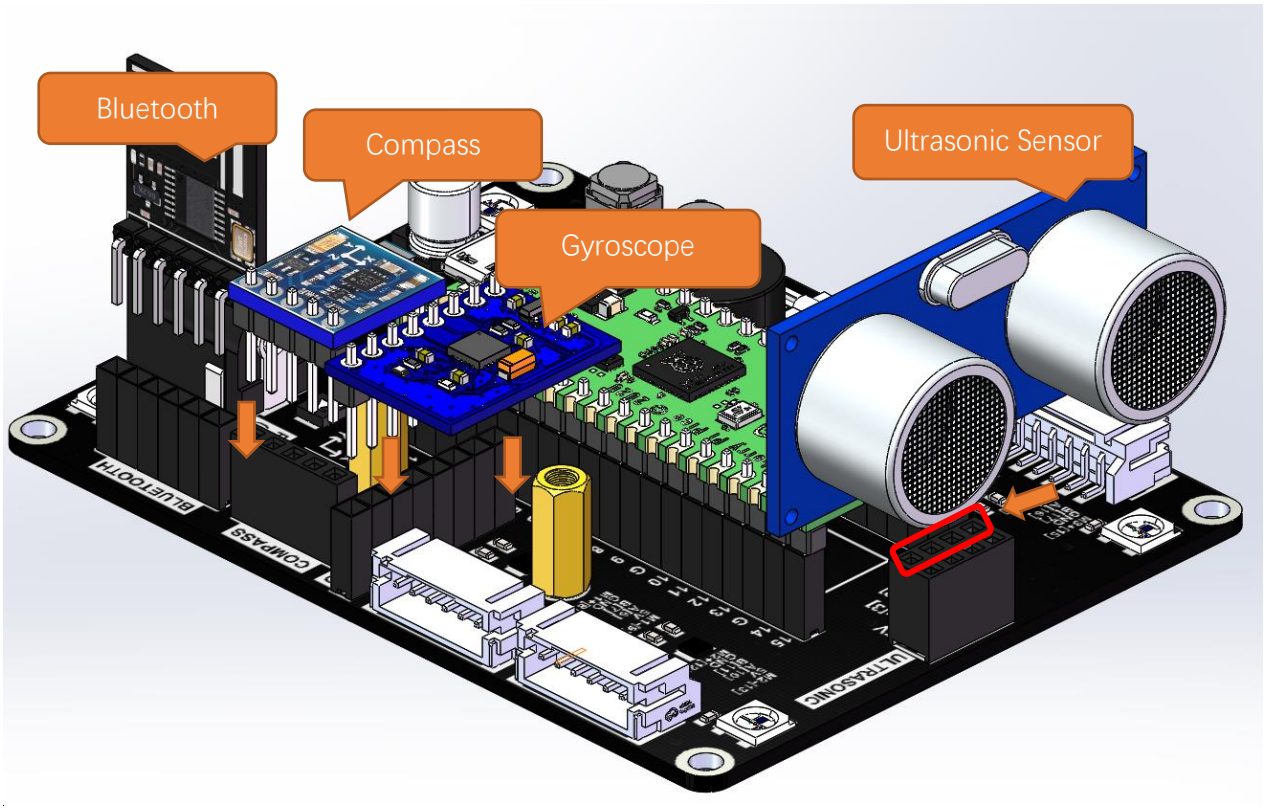
After assembly:



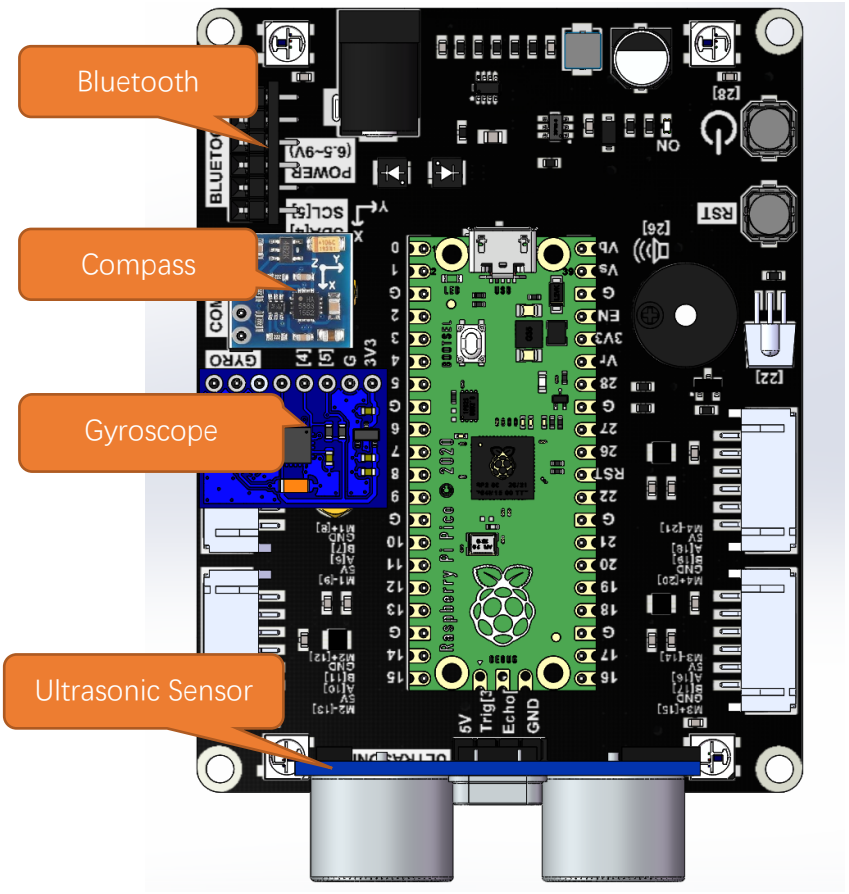
Caution: The Raspberry Pi Pico (W) has a specific orientation for installation. Ensure correct placement based on the markings on the expansion board. Improper installation may result in damage to the

Raspberry Pi Pico (W).

Step 3: Plug in four peripheral modules.



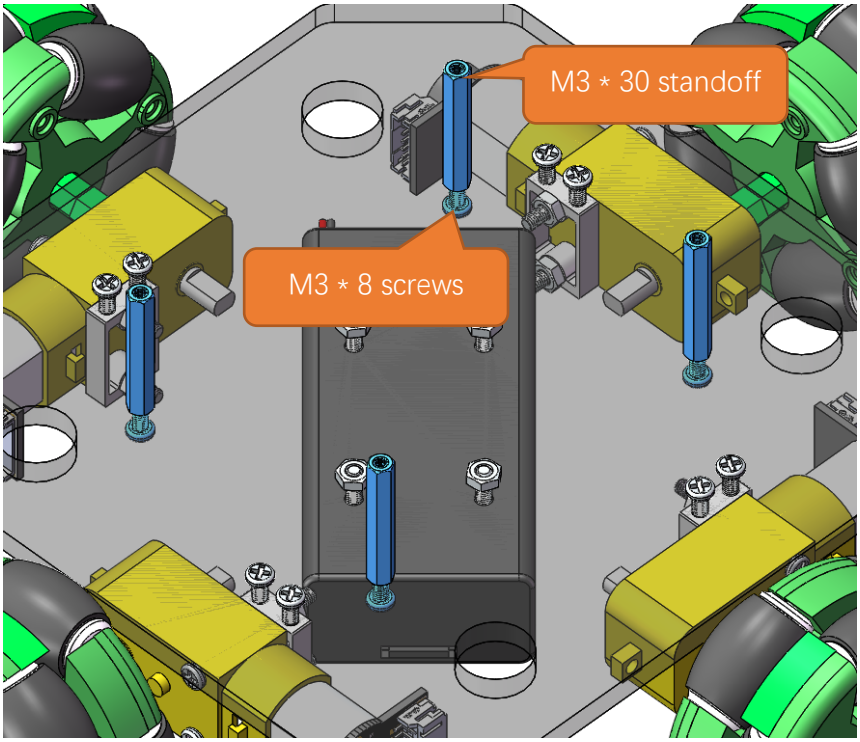
After assembly:

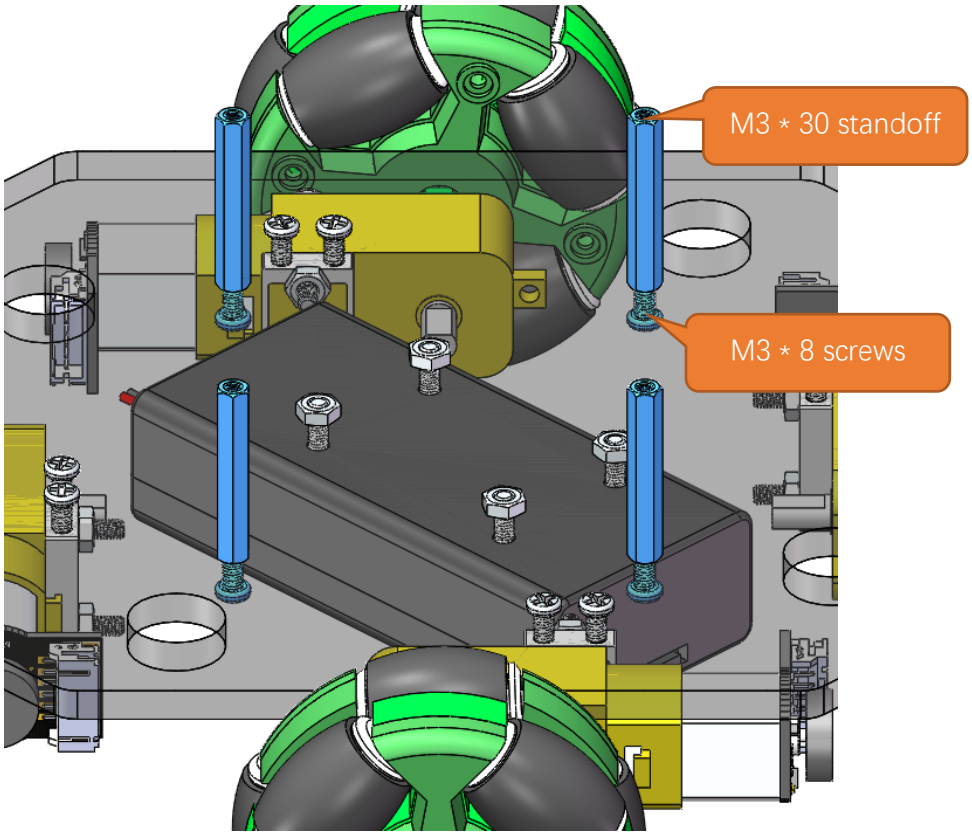


Mounting the Expansion Board to Car Chassis

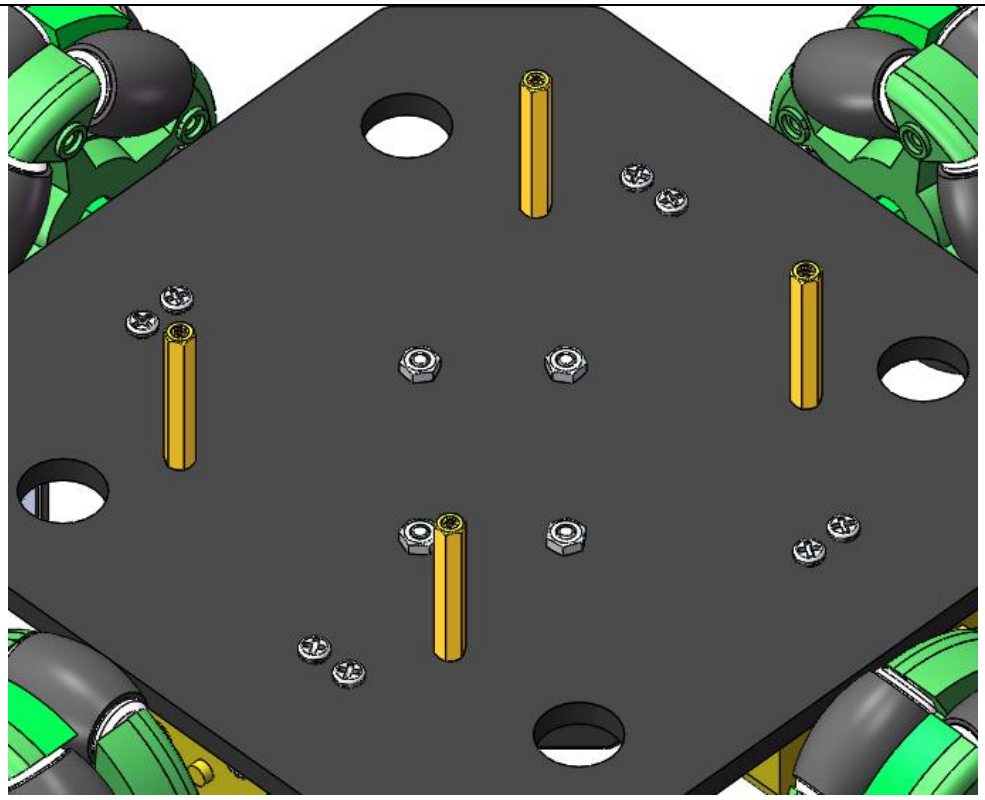
Step 1: Installing four M3x30 standoffs.

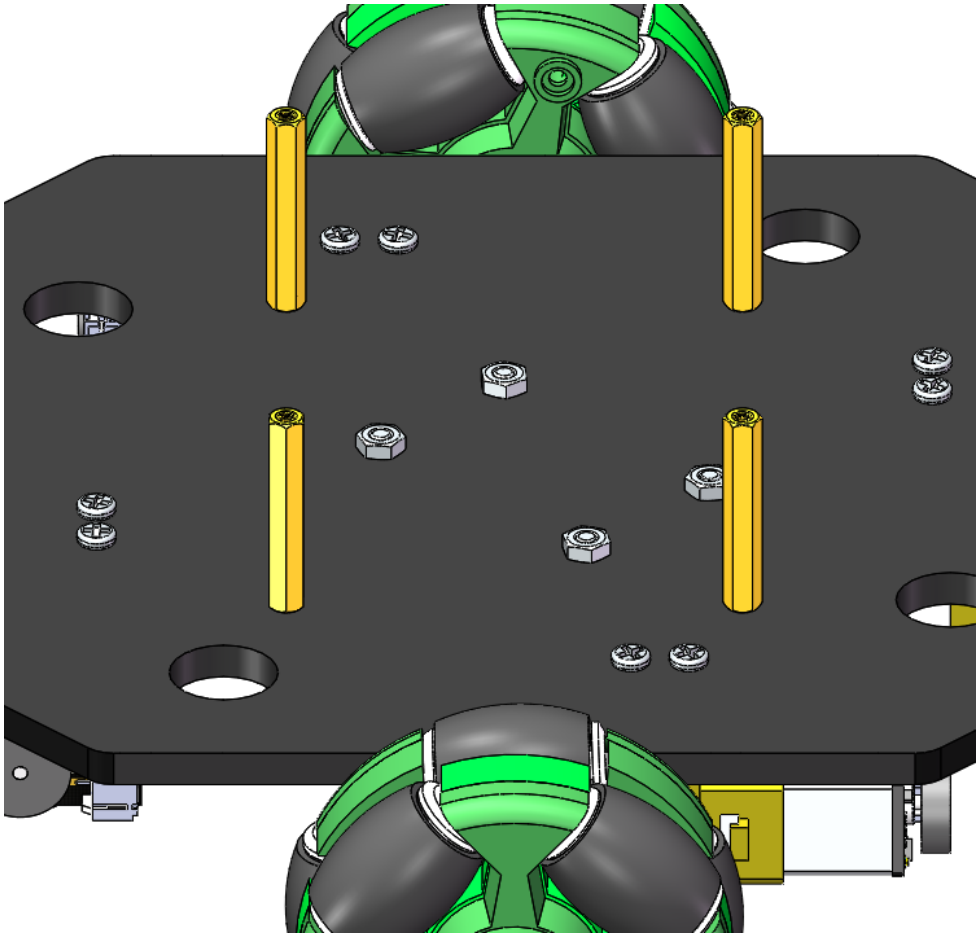
1. Put four M3x30 standoffs on the top of the chassis.
2. Tighten with four M3x8 screws.



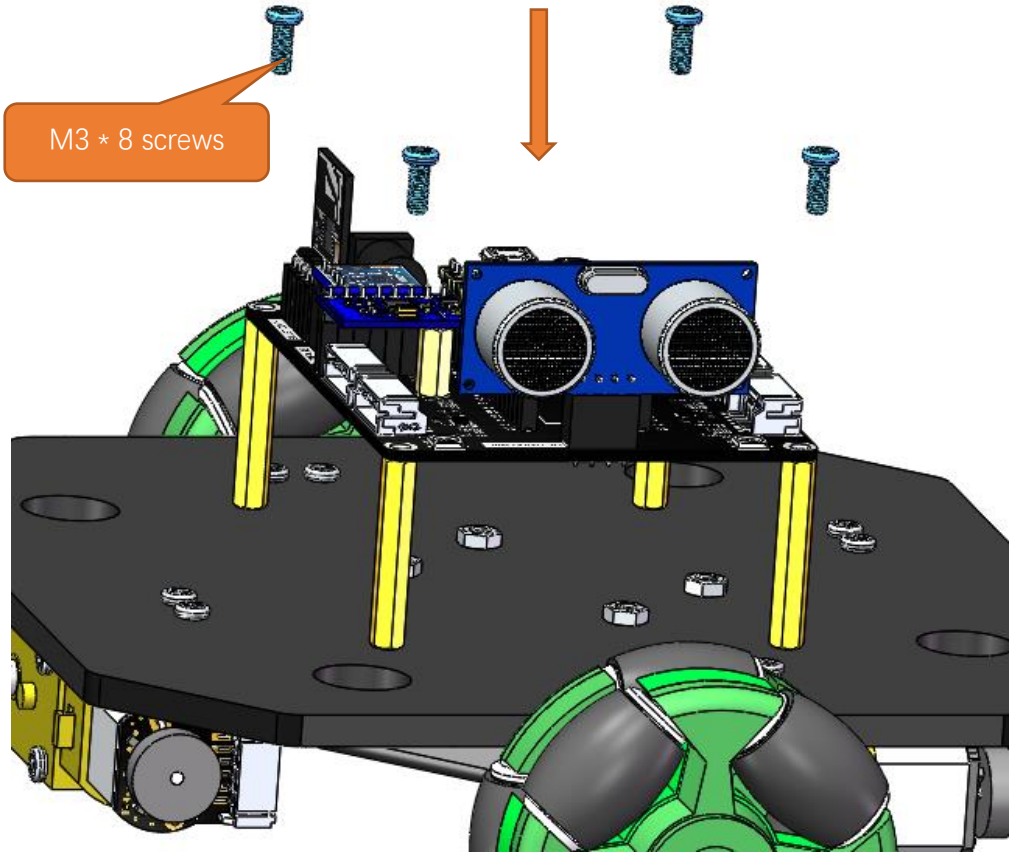


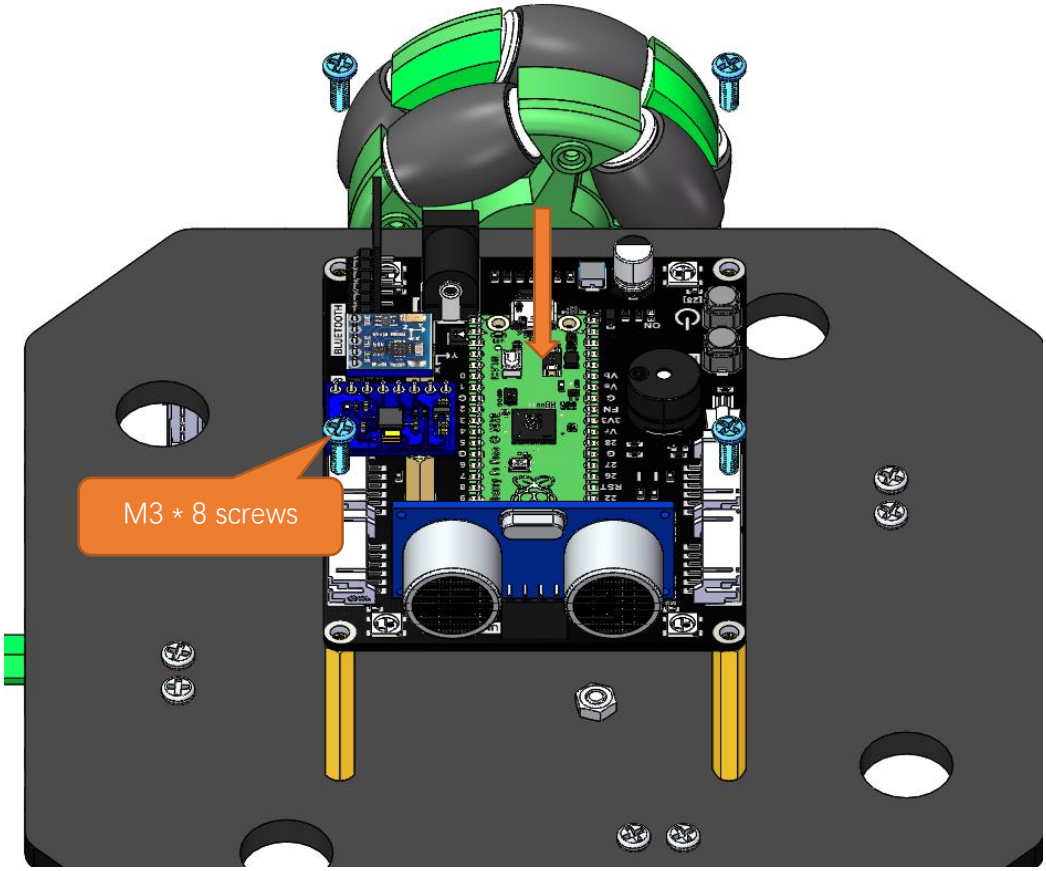
After assembly:



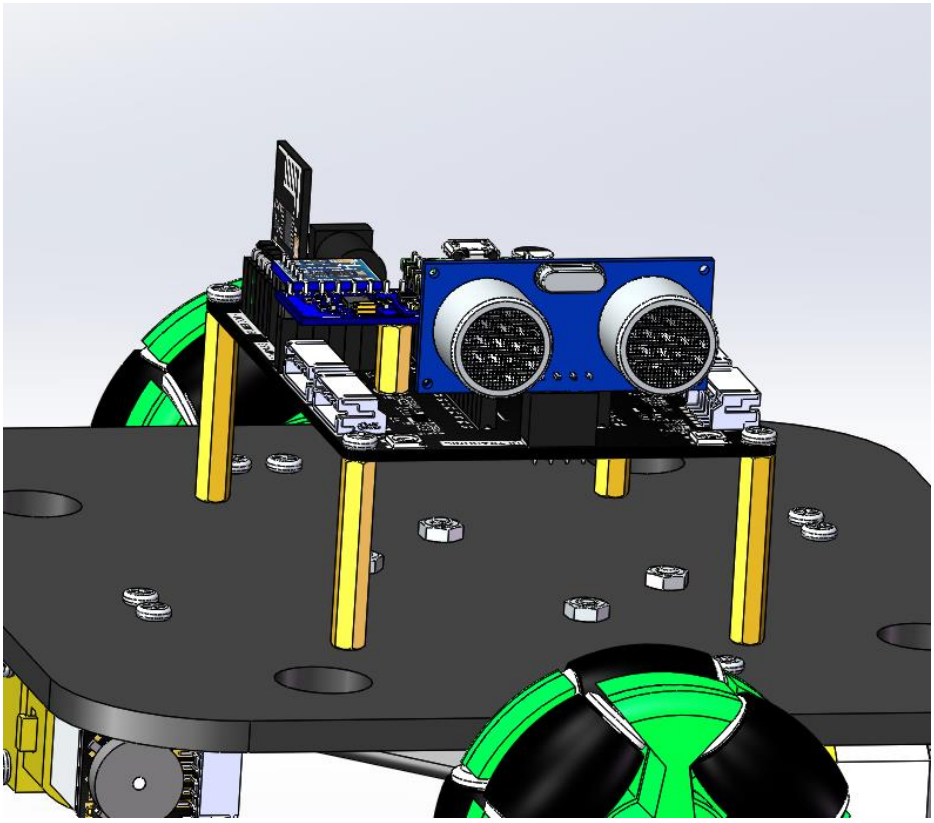


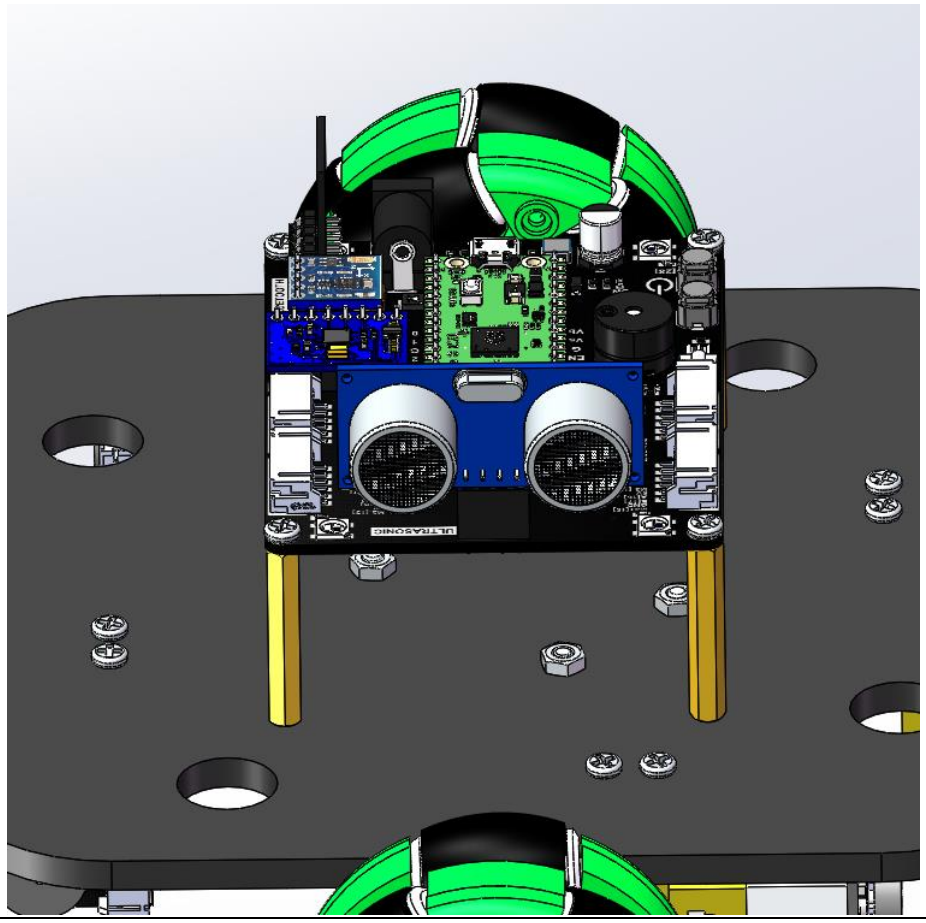
Step 2: Fix the expansion board to the standoffs with four M3x8 screws.



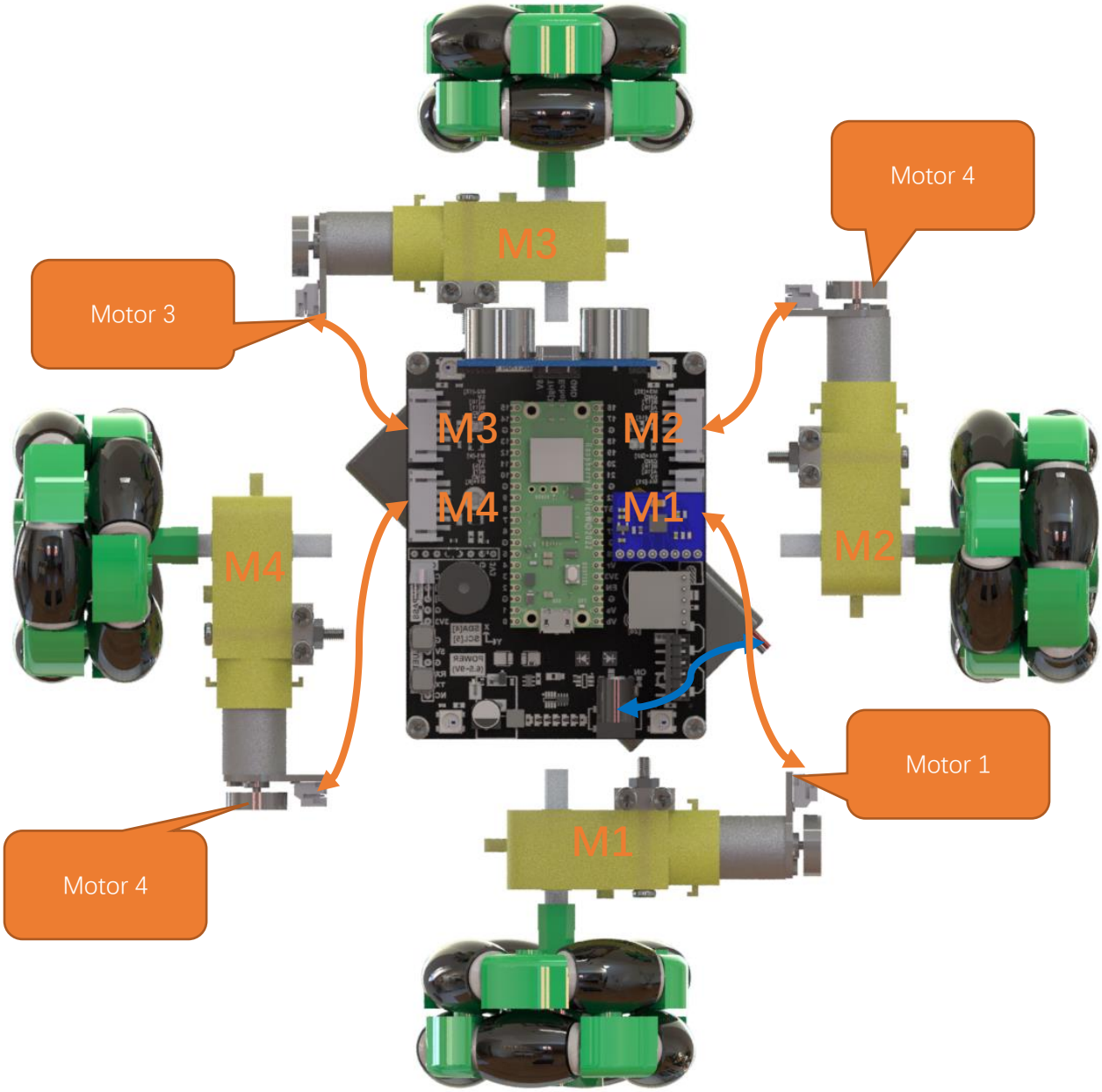


After assembly:




ULTRASONIC

Wires Connection



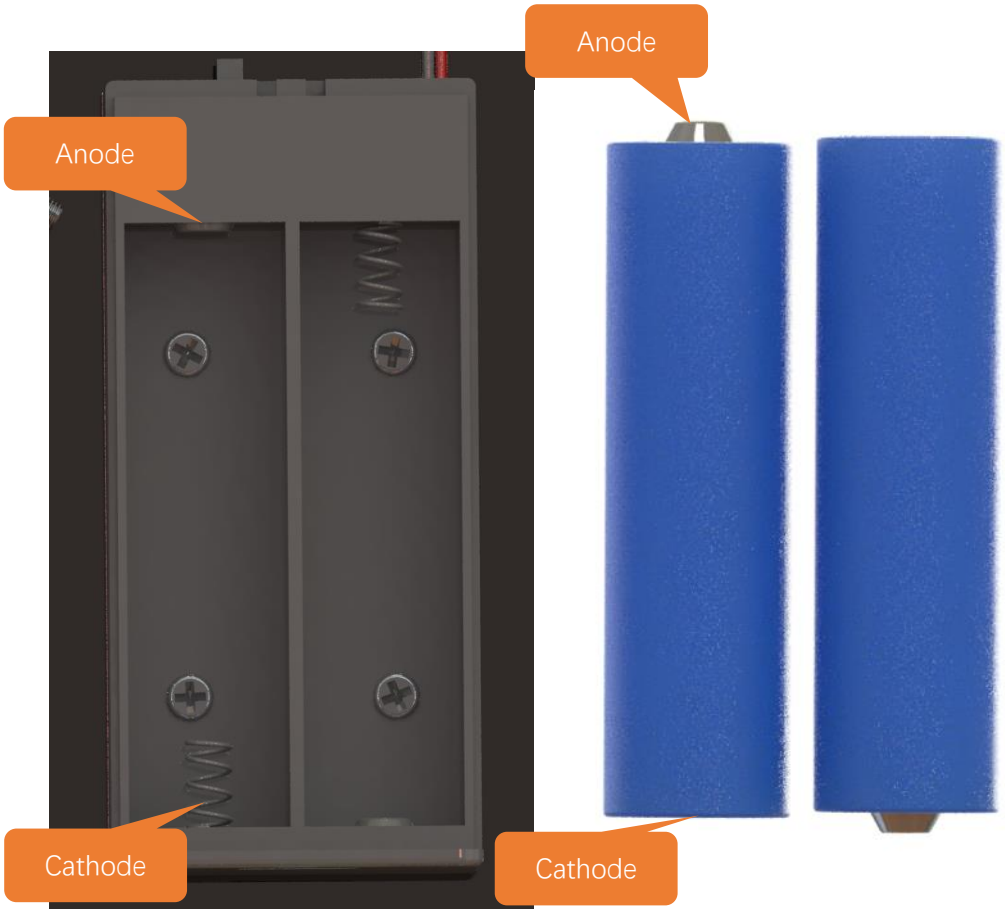
Caution: In the diagram above, the orange wires are motor wires and the blue ones are power cables.

Installing two 18650 batteries

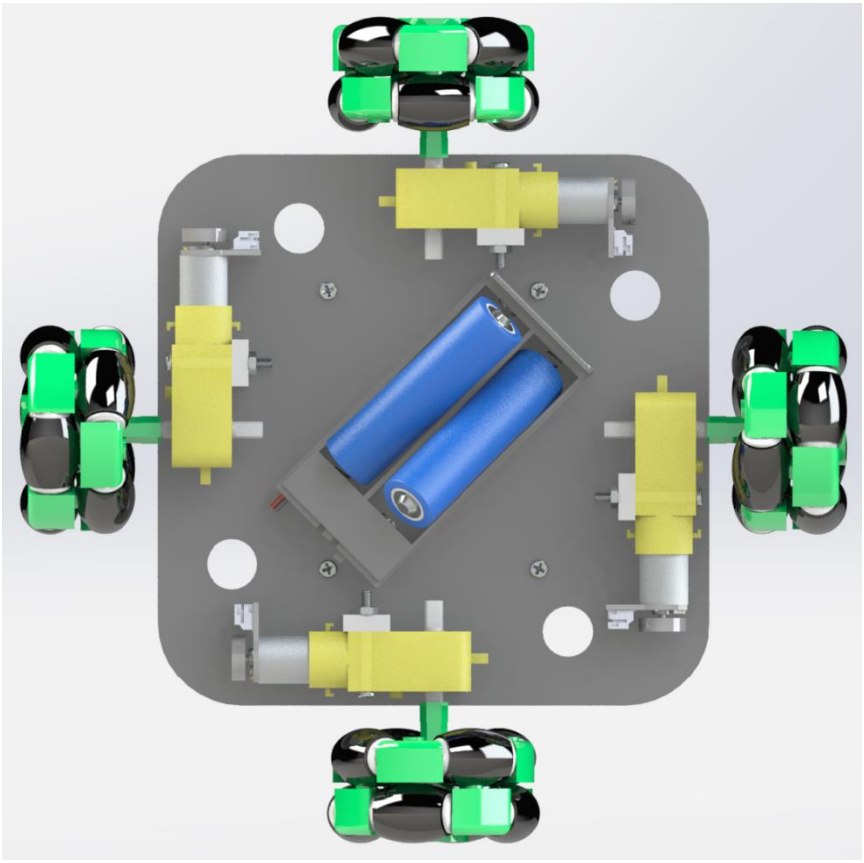
Please follow the steps to install the batteries. Reverse installation of batteries may damage the board.

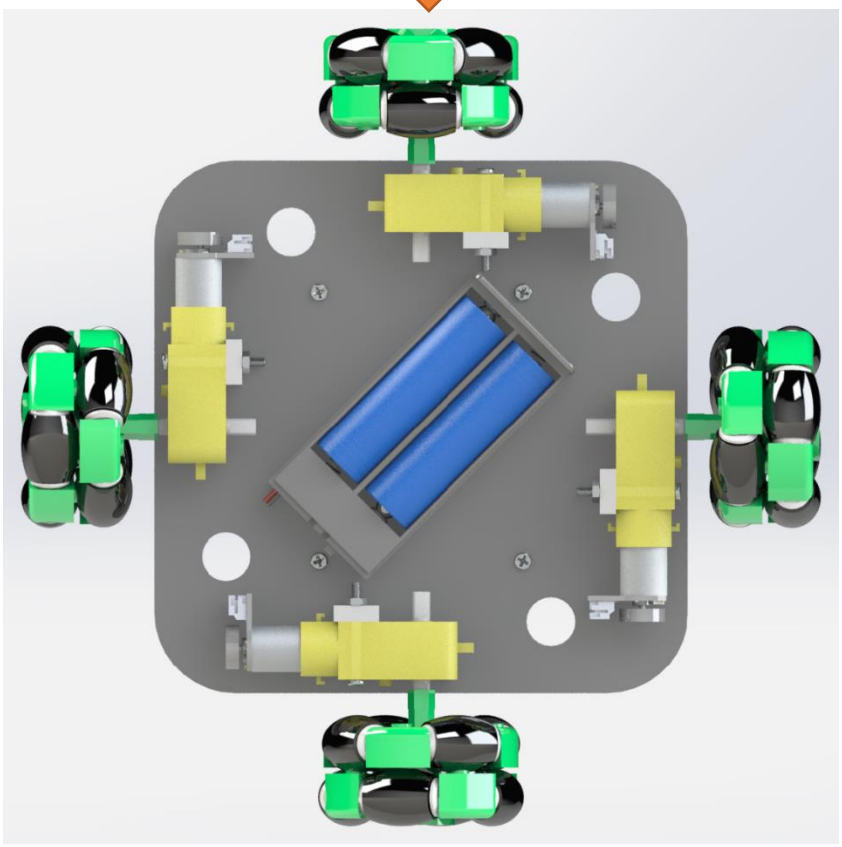
If you have any concerns, please feel free to contact us via support@freenove.com

Step 1: Distinguishing positive and negative of the batteries.

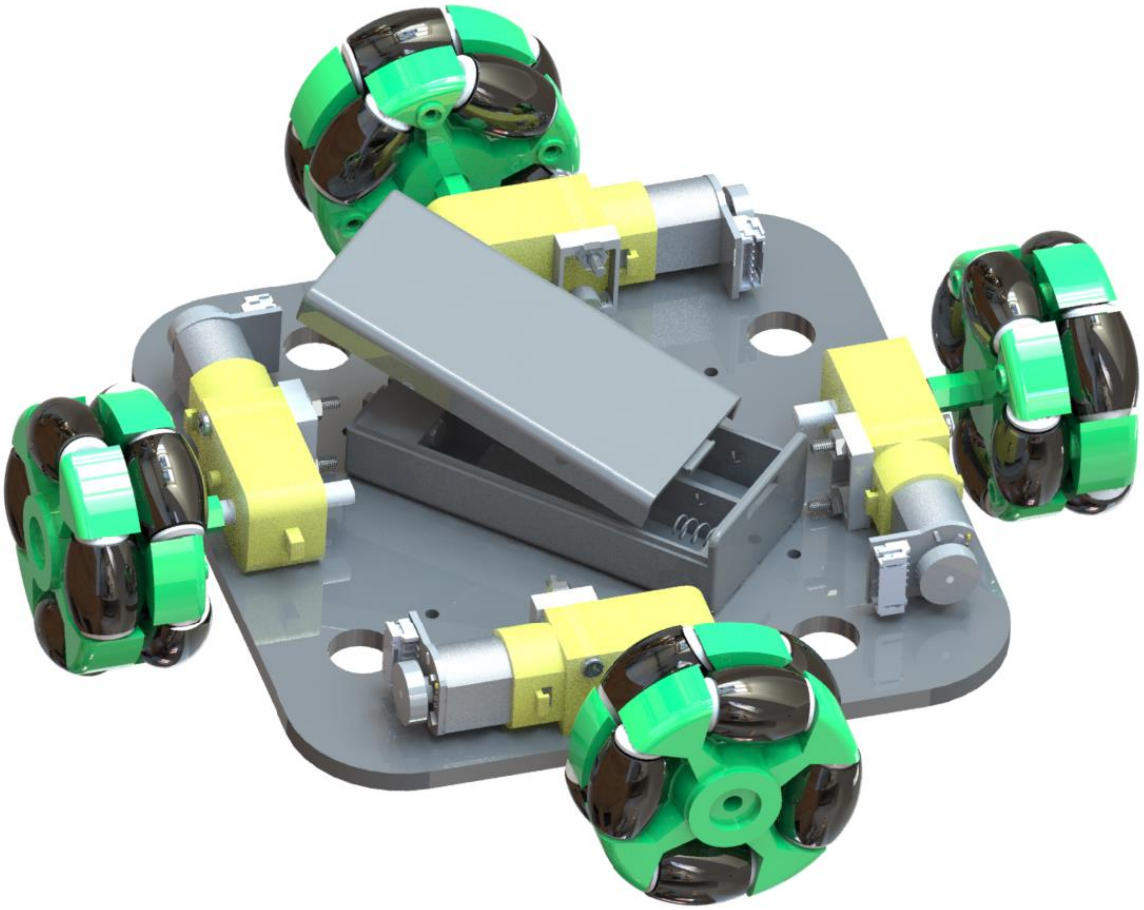


Step 2: Correctly install the batteries.





Step3: Install the cover for battery holder.



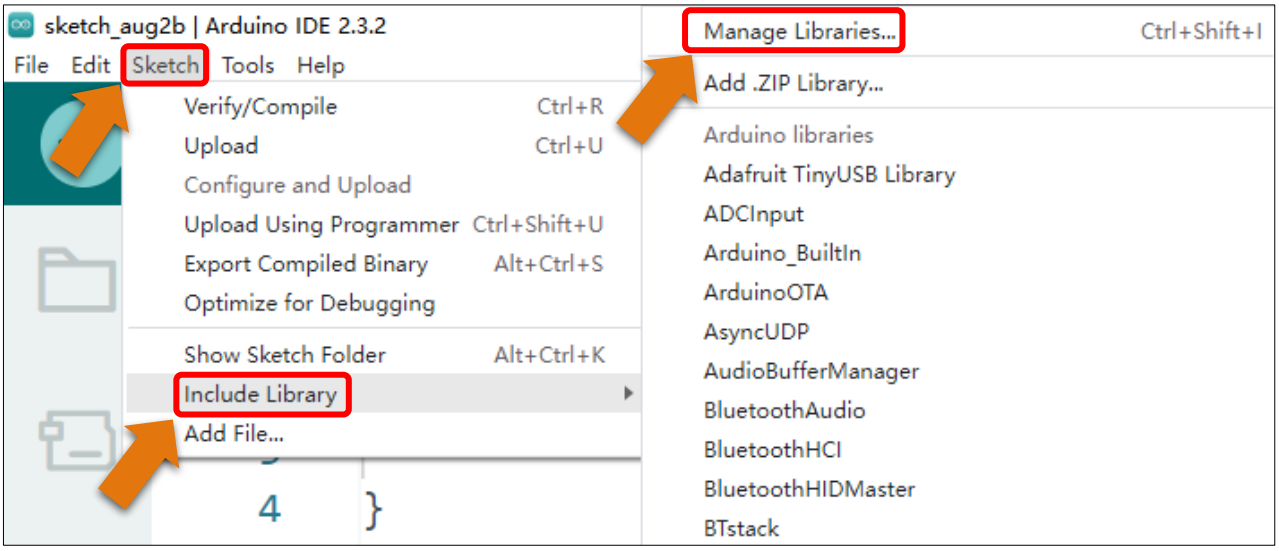
Library Installation

Before uploading the code, we need to include the necessary libraries first to use the functions and tools they provide. This section will introduce how to include libraries on Arduino IDE.

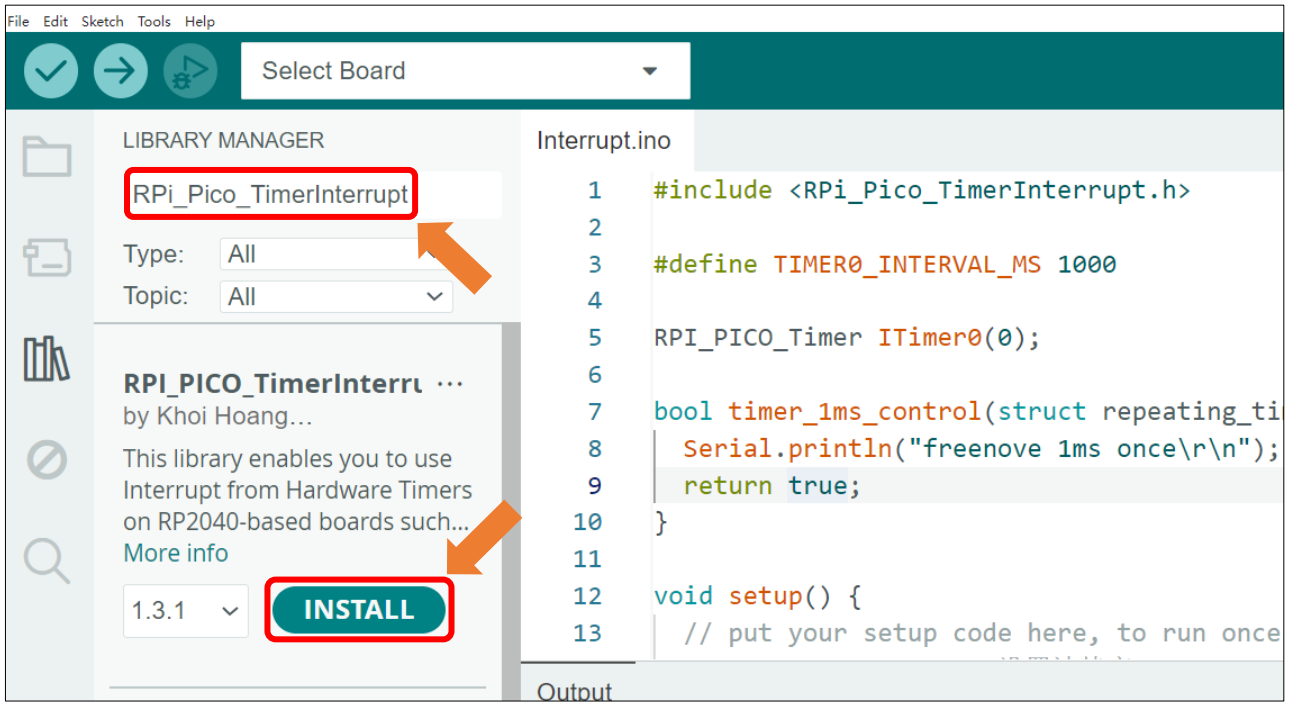
Here are two ways to do it. **Method 2 is preferred.**

Method 1

Open Arduino IDE, click **Sketch** on Menu bar ->**Include Library** -> **Manage Libraries**.



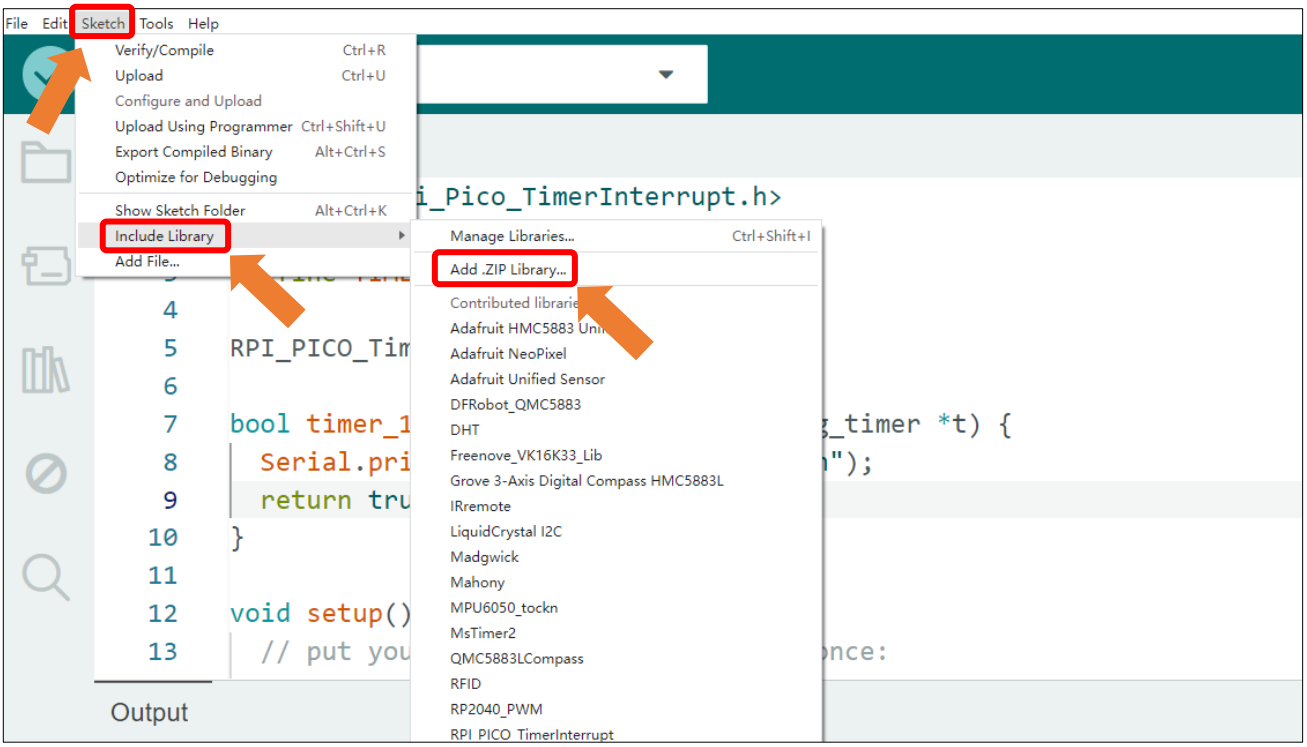
There is an input field on the right top of the pop-up window. Enter **Rpi_Pico_TimerInterrupt** there and click to **install** the library boxed in the following figure.



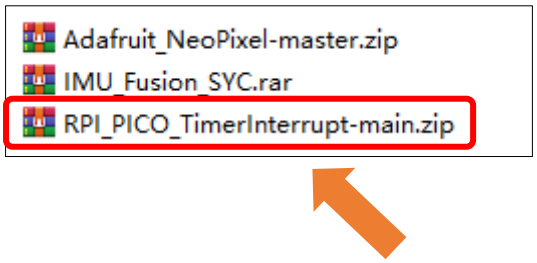
Wait for the installation to finish.

Method 2

Open Arduino IDE, click **Sketch** on **Menu bar->Include Library ->Add .ZIP library**.



On the pop-up window, select **RPI_PICO_TimerInterrupt-main.zip** in Libraries folder under **"car_4_wheel\Libraries"**, and then click Open.

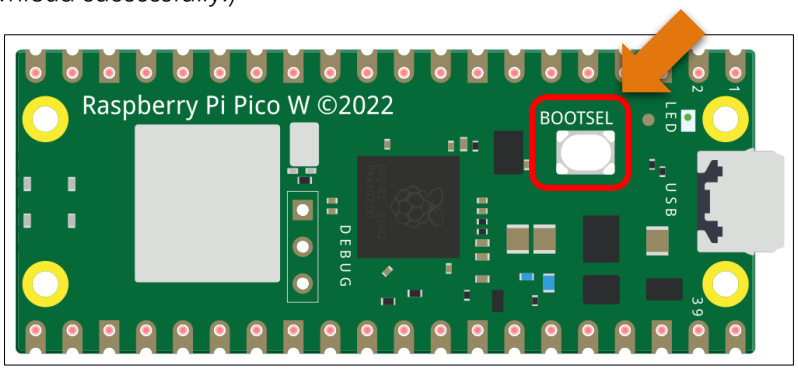


Chapter 2 Uploading First Sketch (LED Blink)

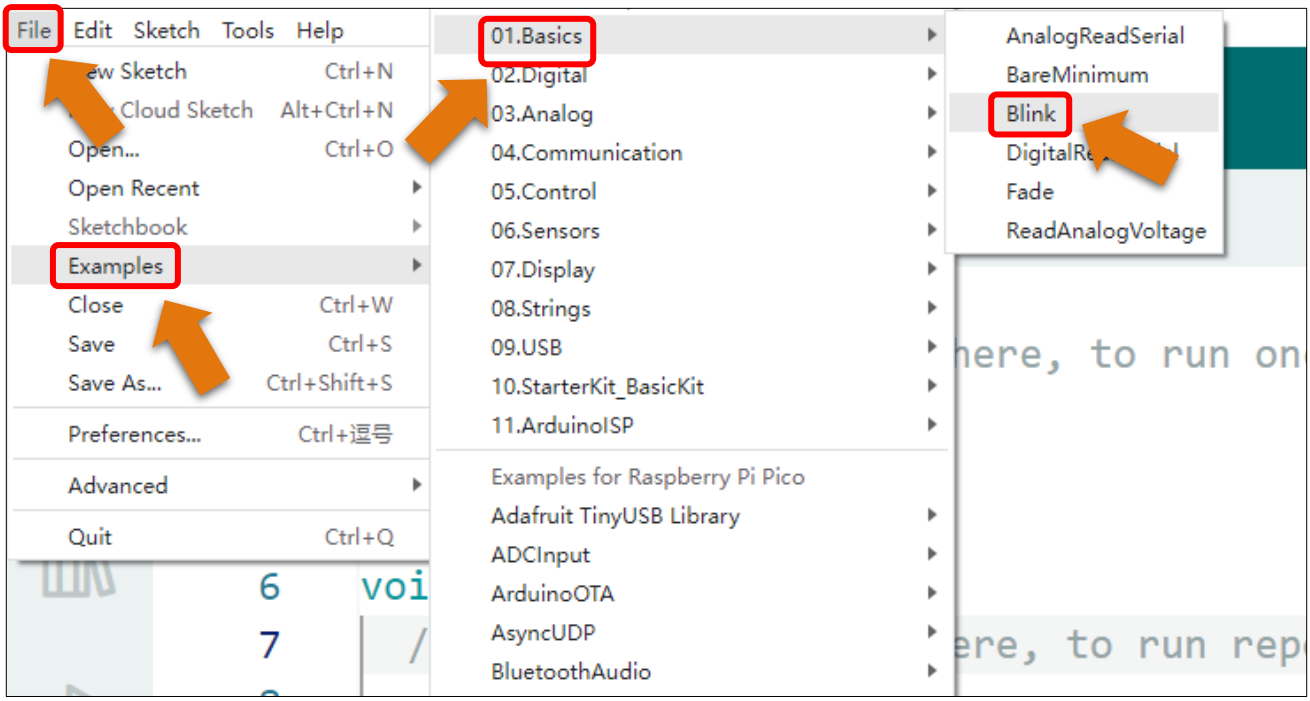
Uploading Adruino-compatible Firmware for Raspberry Pi Pico (W)

To program a new Raspberry Pi Pico (W) with Arduino, you need to upload an Adruino-compatible Firmware for it. Please refer to the following steps to configure.

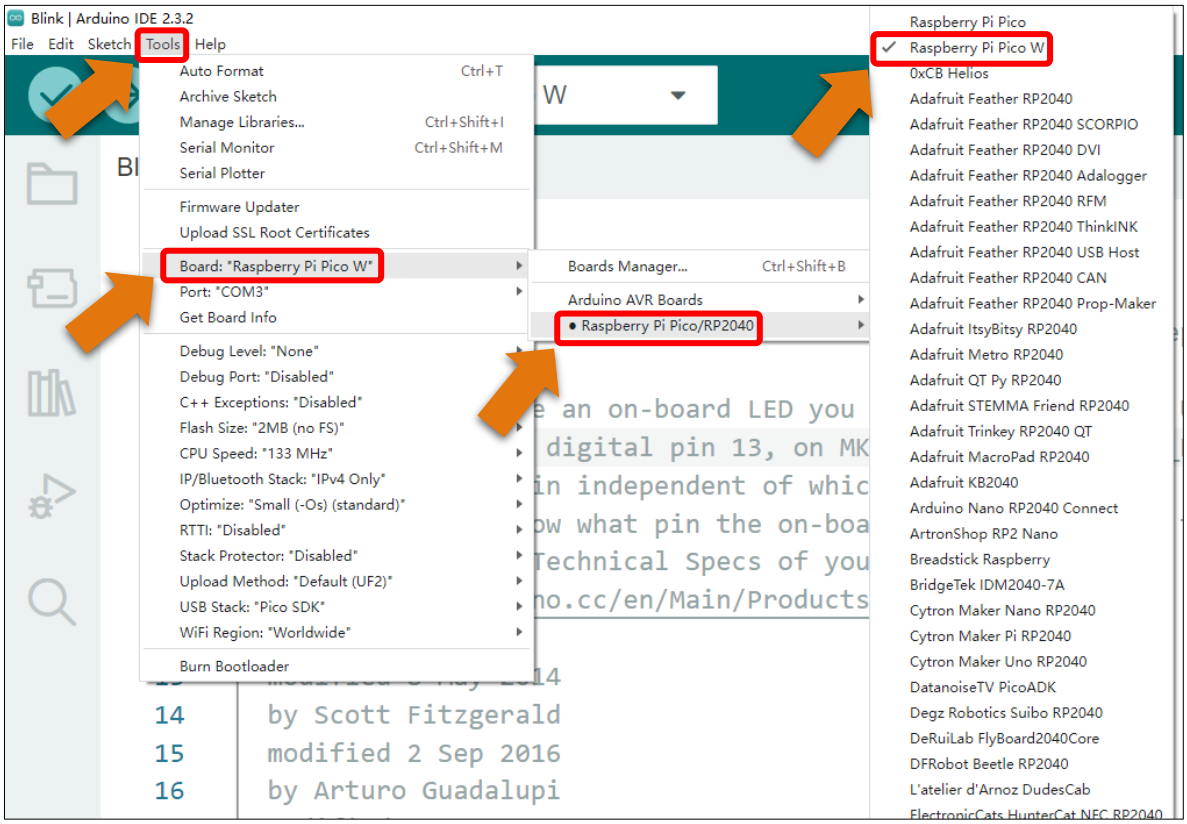
1. Disconnect Pico (W) from your computer. Press and hold the white button (BOOTSEL) on Pico (W) while connecting it to your computer. (Note: Be sure to hold the button before powering the Pico, otherwise the firmware will not download successfully.)



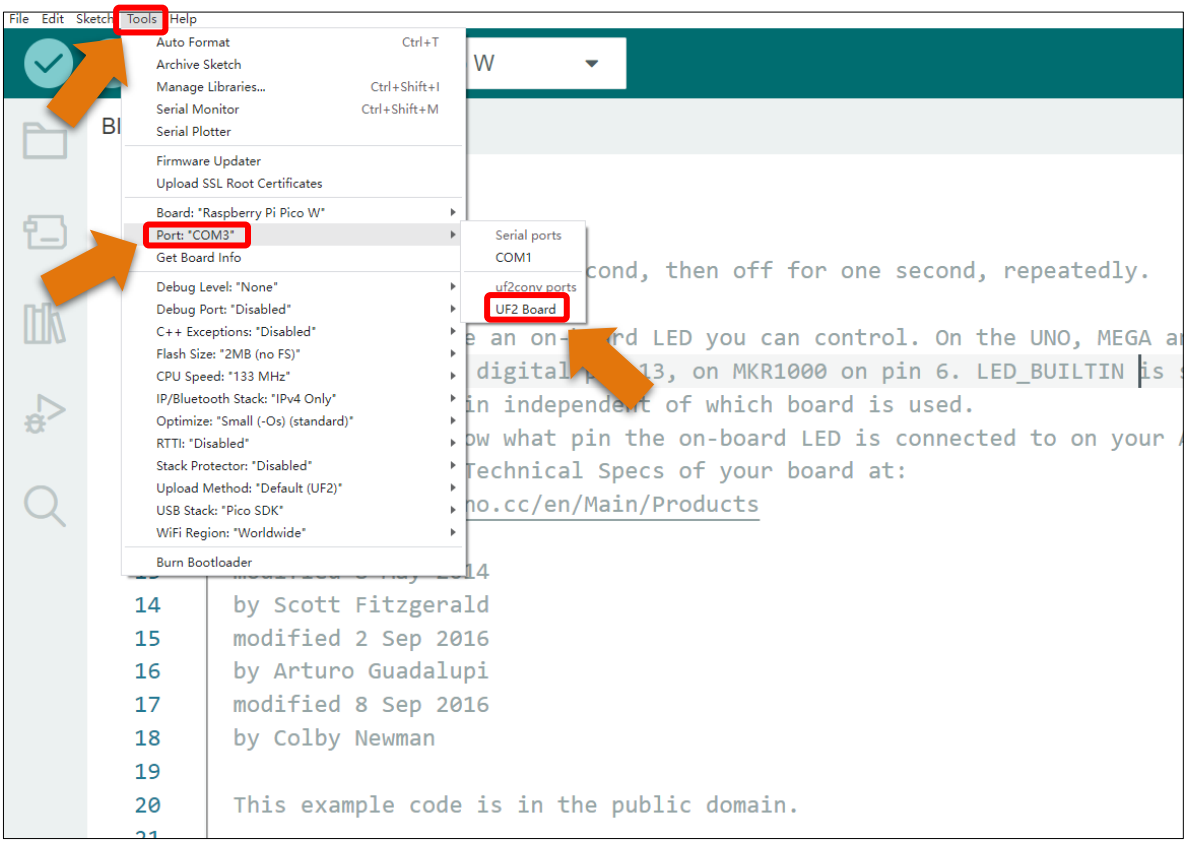
2. Open Arduino IDE. Click **File**>**Examples**>**01.Basics**>**Blink**.



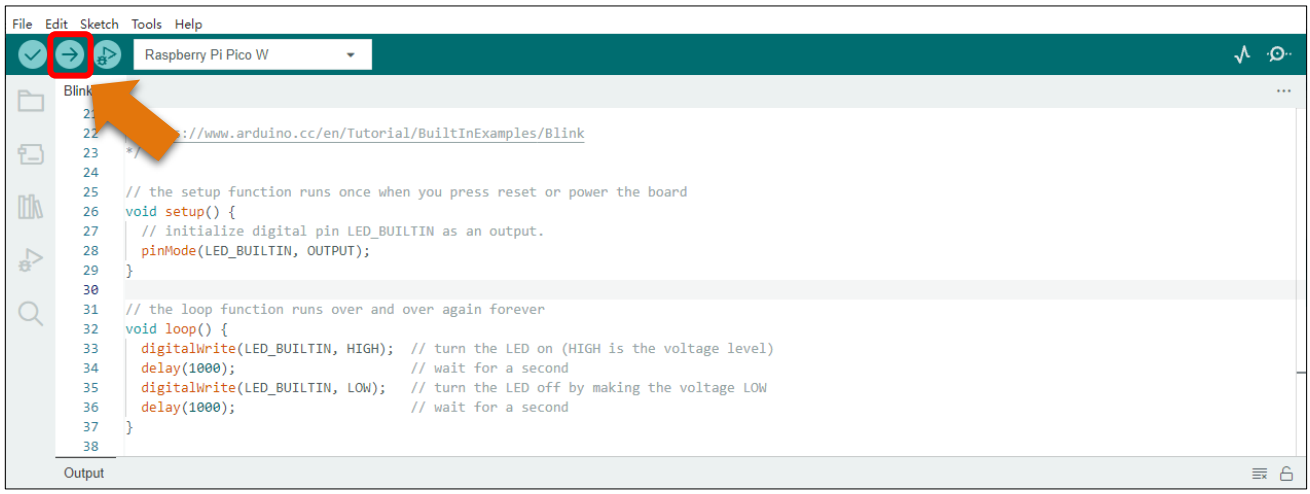
3. Click **Tools>Board>Raspberry Pi RP2040 Boards>Raspberry Pi Pico W**.



4. Click **Tools>Port>UF2 Board**.

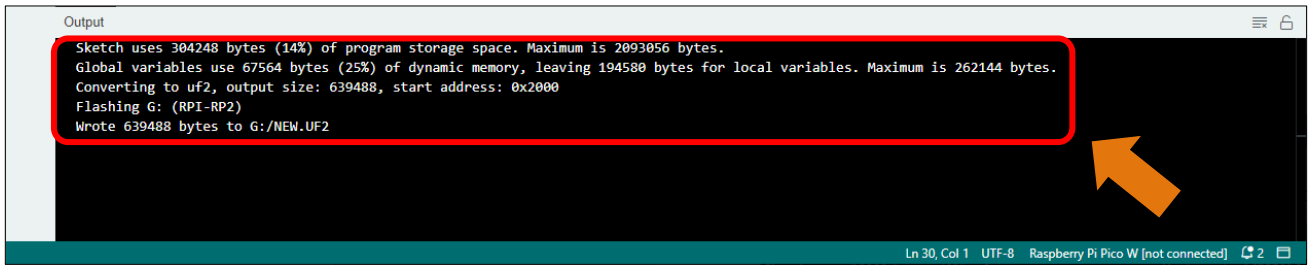


5. **Upload** sketch to Raspberry Pi Pico W.

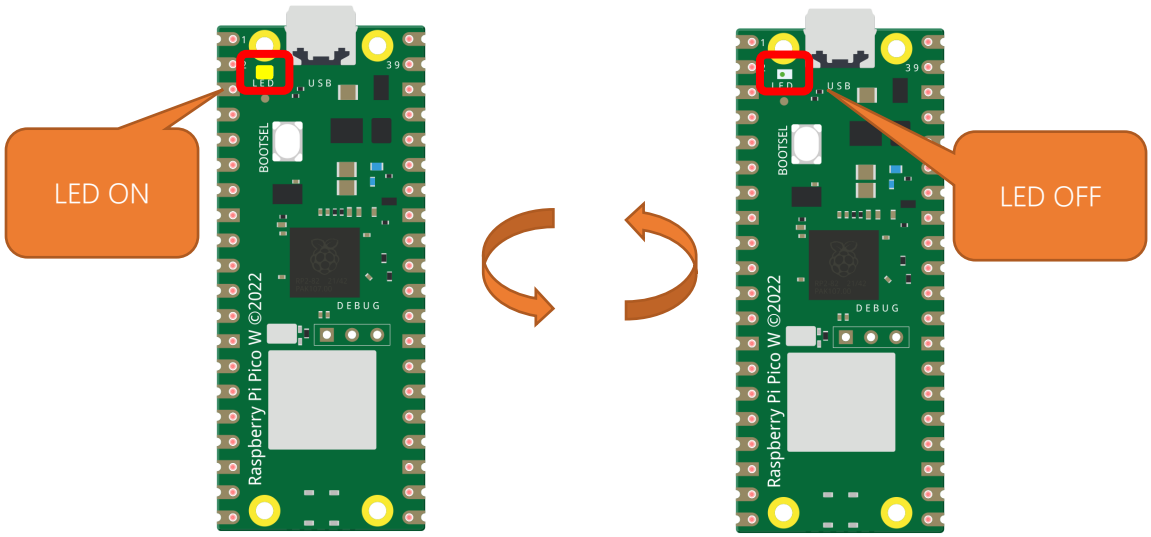



```
  ://www.arduino.cc/en/Tutorial/BuiltInExamples/Blink
*/

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);  // turn the LED on (HIGH is the voltage level)
  delay(1000);                      // wait for a second
  digitalWrite(LED_BUILTIN, LOW);   // turn the LED off by making the voltage LOW
  delay(1000);                      // wait for a second
}
```

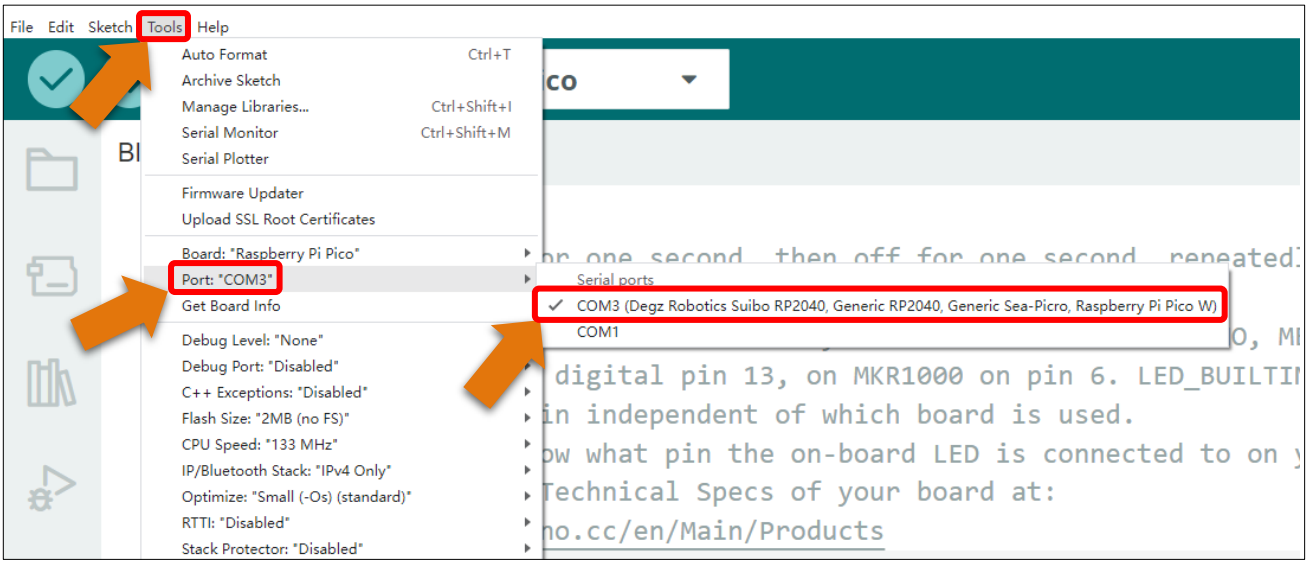
When the sketch finishes uploading, you can see messages as below.



The indicator on Pico W starts to flash.



5, Click **Tools>Port>COMx (Raspberry Pi Pico W)**. X of COMx varies from different computers. Please select the correct one on your computer. In our case, it is COM3.



Note:

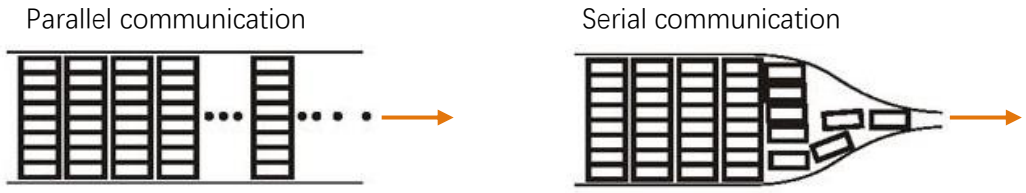
1. At the first use of Arduino to upload sketch for Pico (W), you need to select a port on the UF2 board. After that, each time before uploading sketch, please check whether the port has been selected; otherwise, the downloading may fail.
2. Sometimes, Pico (W) may lose firmware due to the code and fail to work. At this point, you can upload firmware for Pico (W) as mentioned above.
3. Please note that after uploading the first code to Pico, the port UF2_Board will disappear and Pico will create a virtual serial port. Taking Windows system as an example, as shown in the figure above, a port named COMx shows up. Here, x represents a number. The value of x may vary among different computers.

Chapter 3 Serial

Related Knowledge

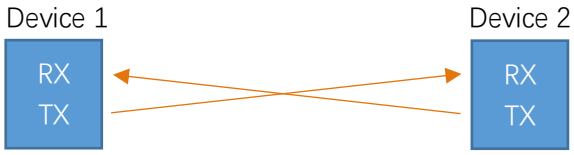
Serial and parallel communication

Serial communication uses one data cable to transfer data one bit by another in turn, while parallel communication means that the data is transmitted simultaneously on multiple cables. Serial communication takes only a few cables to exchange information between systems, which is especially suitable for computers to computer, long distance communication between computers and peripherals. Parallel communication is faster, but it requires more cables and higher cost, so it is not appropriate for long distance communication.



Serial communication

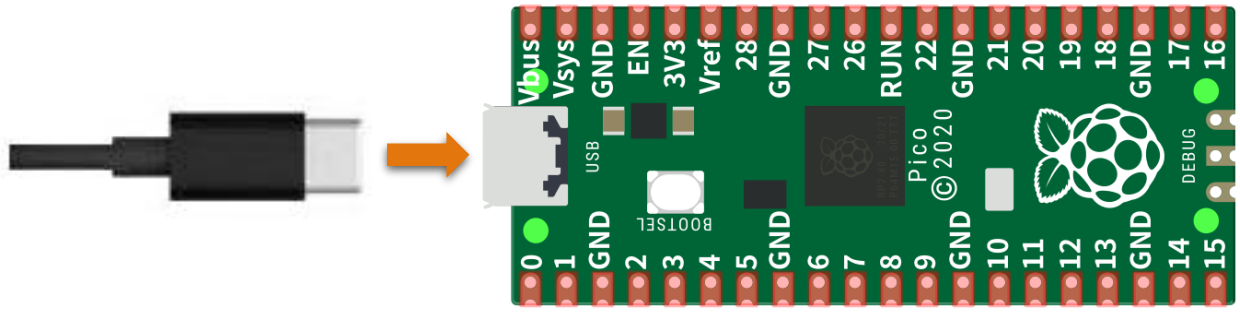
Serial communication generally refers to the Universal Asynchronous Receiver/Transmitter (UART), which is commonly used in electronic circuit communication. It has two communication lines, one is responsible for sending data (TX line) and the other for receiving data (RX line). The serial communication connections of two devices use is as follows:



Before serial communication starts, the baud rate in both sides must be the same. Only use the same baud rate can the communication between devices be normal. The baud rates commonly used are 9600 and 115200.

Circuit

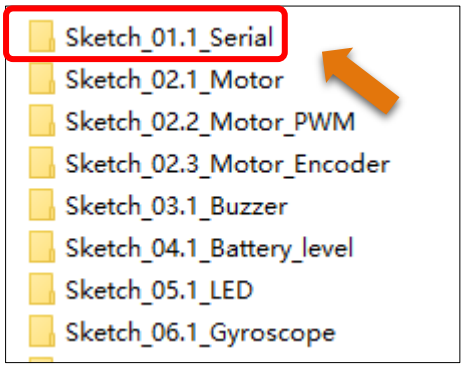
Connect Pico to the computer with USB cable.



If you need any support, please feel free to contact us via: support@freenove.com

Sketch

Next, we download the code to Raspberry Pi Pico (W) to test the motor. Open "**Sketch_01.1_Serial**" folder under **"Freenove_Omni_Wheel_Car_Kit_for_Raspberry_Pi_Pico\Four-Wheel\Sketch"** and double-click "**Sketch_01.1_ Serial.ino**".

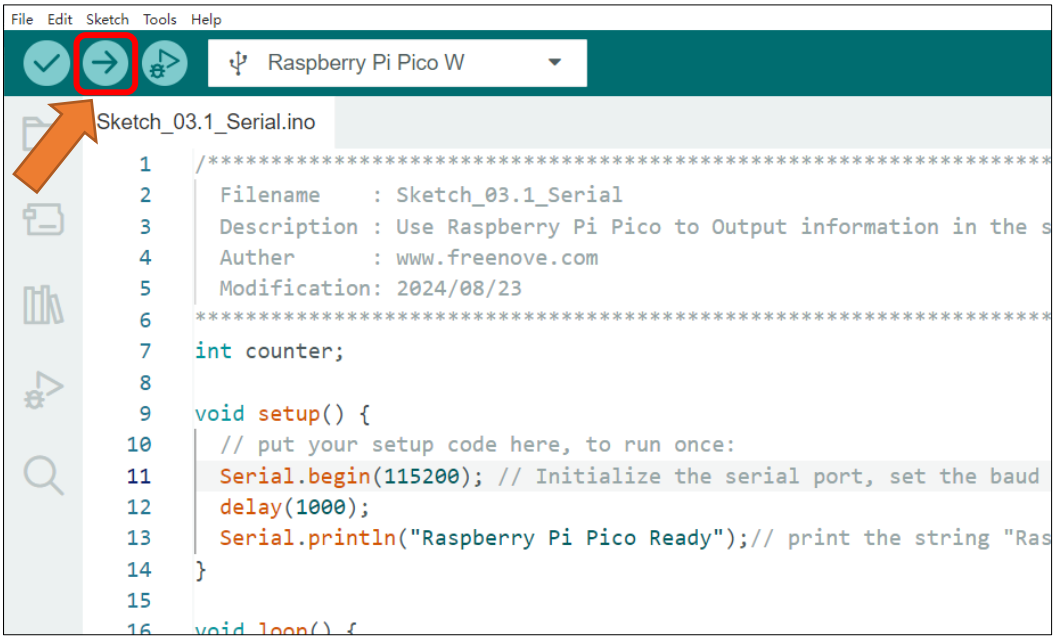


Code

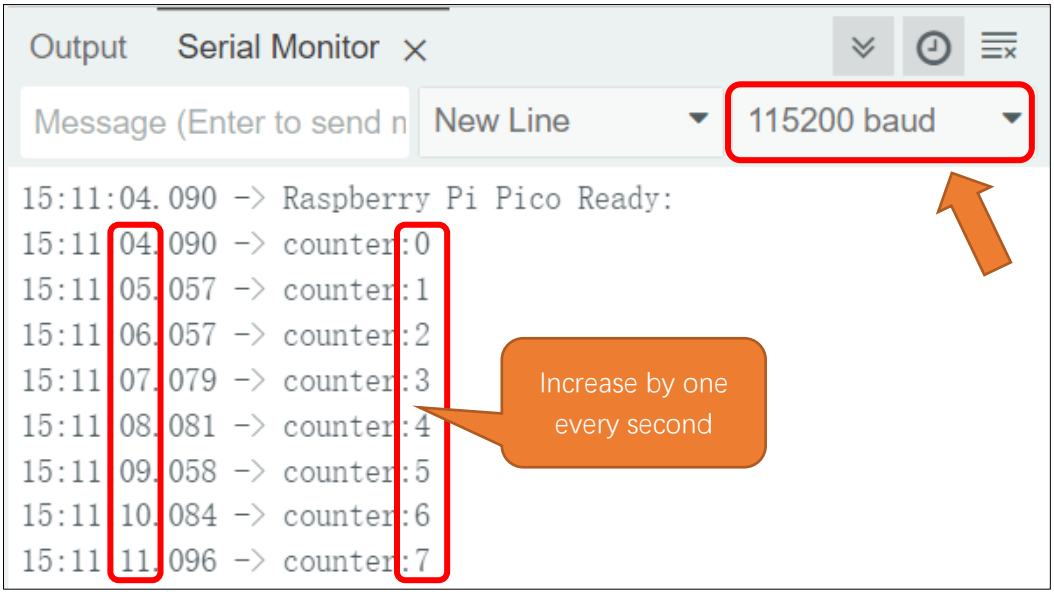
```
int counter; // Define a variable as a data sending to serial port

void setup() {
   // put your setup code here, to run once:
   Serial.begin(115200); // Initialize the serial port, set the baud rate to 115200
   delay(3000);
   Serial.println("Raspberry Pi Pico Ready"); // print the string "Raspberry Pi Pico Ready"
}

void loop() {
   // put you main code here, to run repeatedly:
```

```
12      Serial.print("counter:");
13      Serial.println(counter++);// Increment variable by 1 and print
14      delay(1000); // Delay 1 second
    }
```

This code achieves the function of printing data on serial monitor. Click "**Upload**" to upload the code to Pico.



After downloading the code, open the serial port monitor, and set the baud rate to **115200**



Reference

`void setup ()` The 'setup' function serves as an initializer. It is the first block of code to be executed upon code upload, and it runs only a single time during the program's operation. It is imperative that this function remains intact and unaltered; attempting to delete or comment it out will result in compilation errors.
`void loop ()` The 'loop' function begins execution after the 'setup' function has completed and continues to run repeatedly. This is where you should place code that needs to be continuously executed. Be aware that this function must not be deleted or commented out; doing so will lead to compilation errors.
`void delay(unsigned long ms);` The 'delay' function in Arduino is used to halt the program's execution for a specified amount of time. It takes one argument, which represents the duration of the pause in milliseconds (ms). For instance, 'delay(1000)' will pause the program for 1 second, as 1 second is equivalent to 1000 milliseconds.
`void Serial.begin(long baud);` The 'Serial.begin' function in Arduino is used to initialize the serial communication. It accepts one argument, which specifies the baud rate to be set.
`size_t Serial.print(int val);` The 'Serial.print' function in Arduino is used for sending data to the serial monitor. The value passed as an argument to this function will be printed to the serial monitor.
`size_t Serial.println(int val);` The 'Serial.println' function in Arduino is similar to 'Serial.prin'`, but it differs in that it appends a newline character at the end of the data being sent. This is useful for formatting the output in the serial monitor, making it easier to read by separating lines of text.

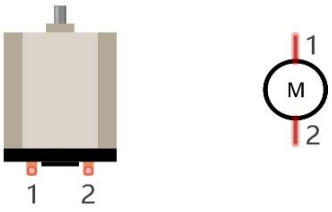
Chapter 4 Motor Test

4.1 Motor

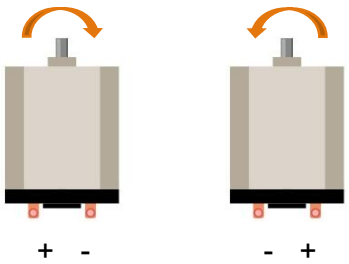
Related Knowledge

Motor

A motor is a device that converts electrical energy into mechanical energy. Motor consists of two parts: stator and rotor. When motor works, the stationary part is stator, and the rotating part is rotor. Stator is usually the outer case of motor, and it has terminals to connect to the power. Rotor is usually the shaft of motor, and can drive other mechanical devices to run. Diagram below is a small DC motor with two pins.

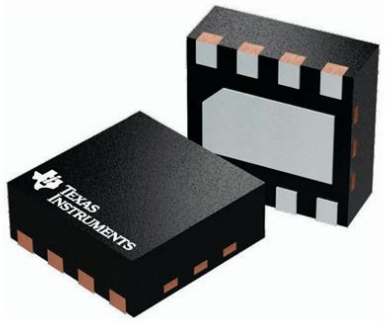


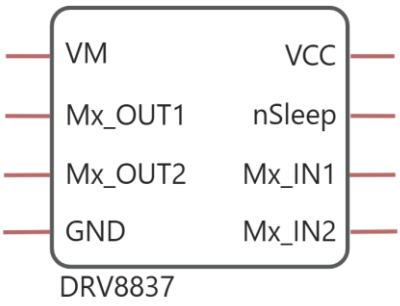
When a motor is connected to the power supply, it will rotate in one direction. Reverse the polarity of power supply, then the motor rotates in opposite direction.



DRV8837

The DRV8837 is a DC motor driver chip that can control the motor's forward rotation, reverse rotation, and braking.



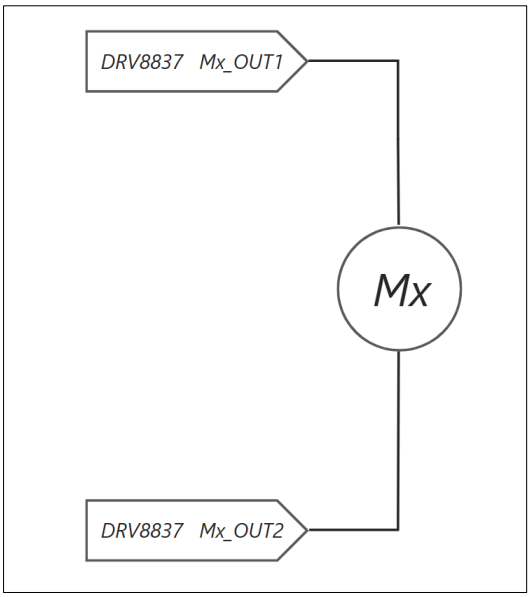


Port description of DRV8837 is as follows:

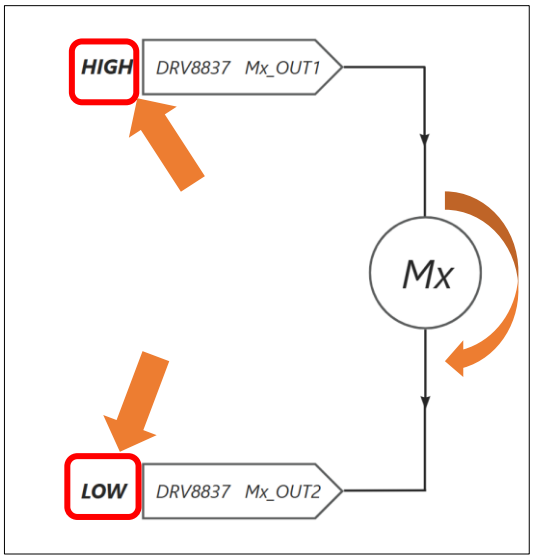
PIN	DESCRIPTION
VM	Motor power supply
OUT1 / OUT2	Motor output Connect these pins to the motor winding.
GND	Device ground
VCC	Logic power supply
nSleep	Sleep mode input When this pin is in logic low, the device enters low-power sleep mode. The device operates normally when this pin is logic high. Internal pulldown
In1 / In2	IN1 input / IN2 input

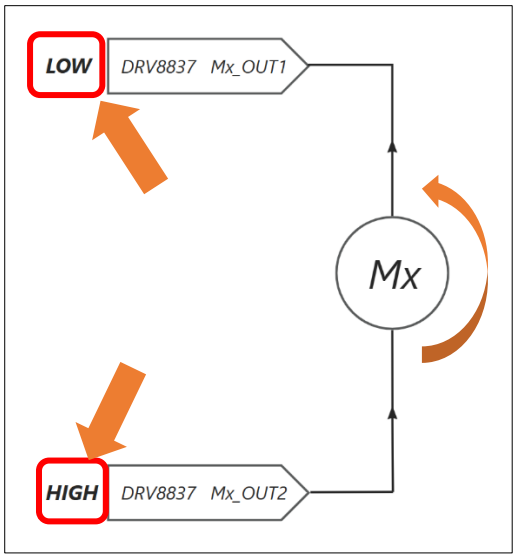
For more details, please see datasheet.

The two terminals of the motor connect to two output channels of DRV8837.



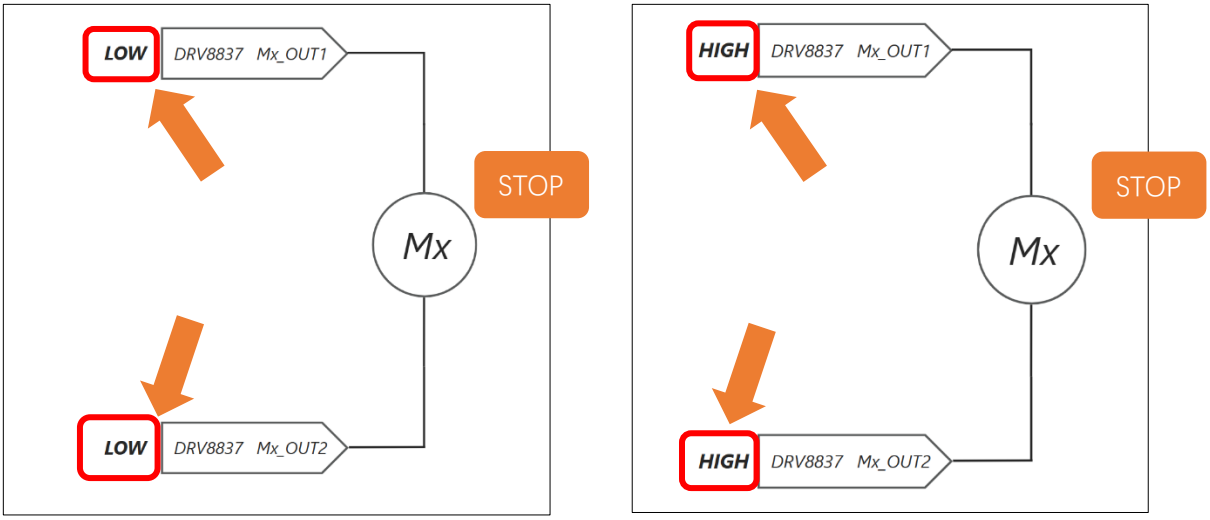
When one output channel outputs high and the other outputs low, the motor will rotate in one direction. If the signals of the two channels are exchanged, that is, the high level and the low level are interchanged, the direction of rotation of the motor will change.





.

When the two channels simultaneously output high or low, the motor enters braking mode.

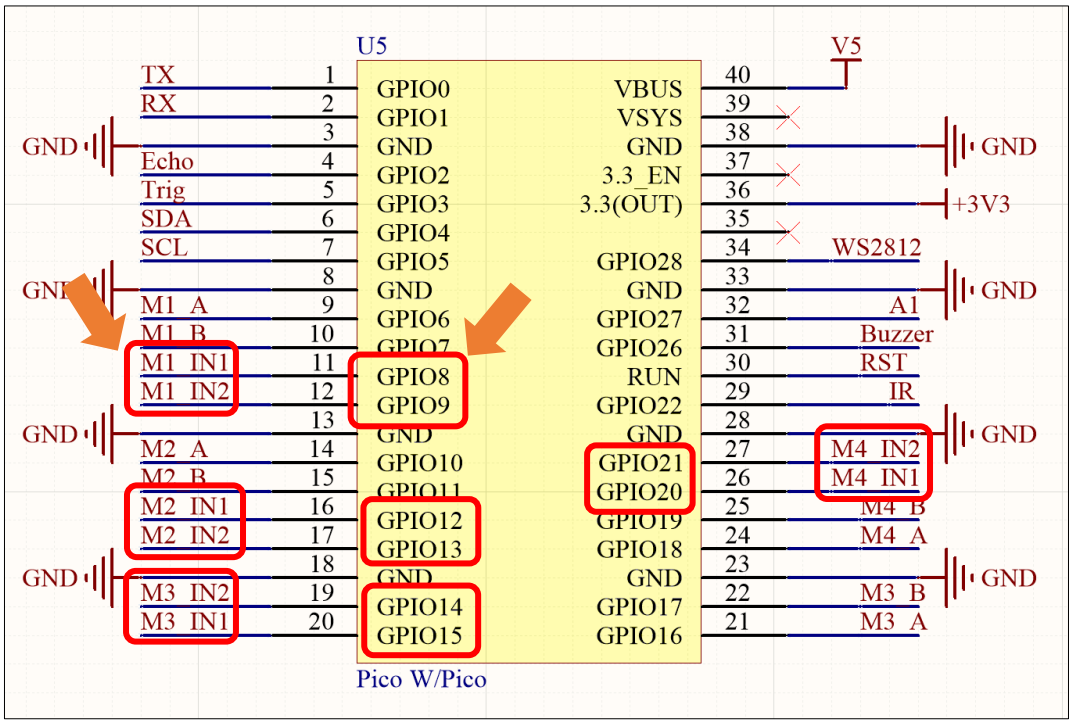


Here is the truth table for the motor states:

Mx_IN1	Mx_IN2	Rotating direction of the wheels
1	0	Forward
0	1	backward
1	1	Stop
0	0	Stop

Schematic

Next, we will start to learn the circuit connection of the car. As shown below, M1_IN1 connects to GPIO8 of the Pico, which means we can control the signal of M1_IN1 via GPIO8.

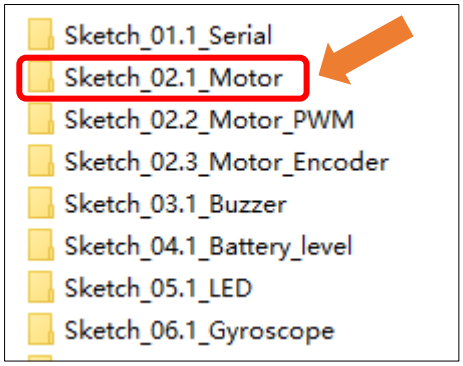


As can be seen from the schematic above, GPIO8 and GPIO9 controls motor M1, GPIO12 and GPIO13 controls M2, GPIO14 and GPIO15 controls M3, and GPIO20 and GPIO21 controls M4.

Note: The 'x' in Mx_IN1 should be between 1 and 4, corresponding to the identification numbers of the four motors. For instance, M1_IN1 refers to the first control pin for Motor 1, while M1_IN2 refers to the second control pin for Motor 1. Both pins are used in conjunction to control the motor's operations such as forward motion, reverse motion, and halting, where a 0 signifies a low voltage state and a 1 indicates a high voltage state.

Sketch

Next, we download the code to Raspberry Pi Pico (W) to test the motor. Open "**Sketch_02.1_Motor**" folder under **"Freenove_Omni_Wheel_Car_Kit_for_Raspberry_Pi_Pico\Four-Wheel\Sketch"** and double-click "**Sketch_02.1_Motor.ino**".

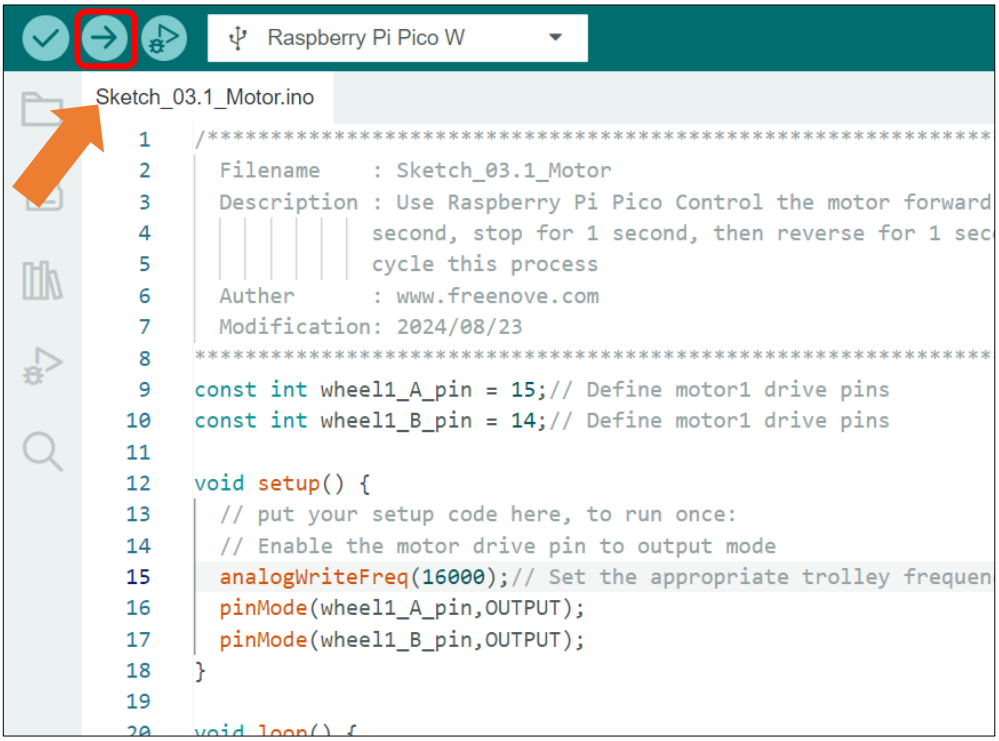


Code

```
const int wheel1_A_pin = 8; // Define motor drive pins
const int wheel1_B_pin = 9; // Define motor drive pins
void Motor_init()
{
  pinMode(wheel1_A_pin,OUTPUT);
  pinMode(wheel1_B_pin,OUTPUT);
}
void setup() {
  // put your setup code here, to run once:
  Motor_init();// Motor initialization
}
void loop() {
  // put you main code here, to run repeatedly:
  digitalWrite(wheel1_A_pin,1);  // Forward rotation
  digitalWrite(wheel1_B_pin,0);
  delay(1000);

  digitalWrite(wheel1_A_pin,0);  //stop
  digitalWrite(wheel1_B_pin,0);
  delay(1000);
```

```
21
22    digitalWrite(wheel1_A_pin,0);  // Contrarotation
23    digitalWrite(wheel1_B_pin,1);
24    delay(1000);
25
26    digitalWrite(wheel1_A_pin,0);  // Contrarotation
27    digitalWrite(wheel1_B_pin,0);
28    delay(1000);
29  }
```

This code is to control the motors to rotate. Click "**Upload**" to upload the code to Pico.

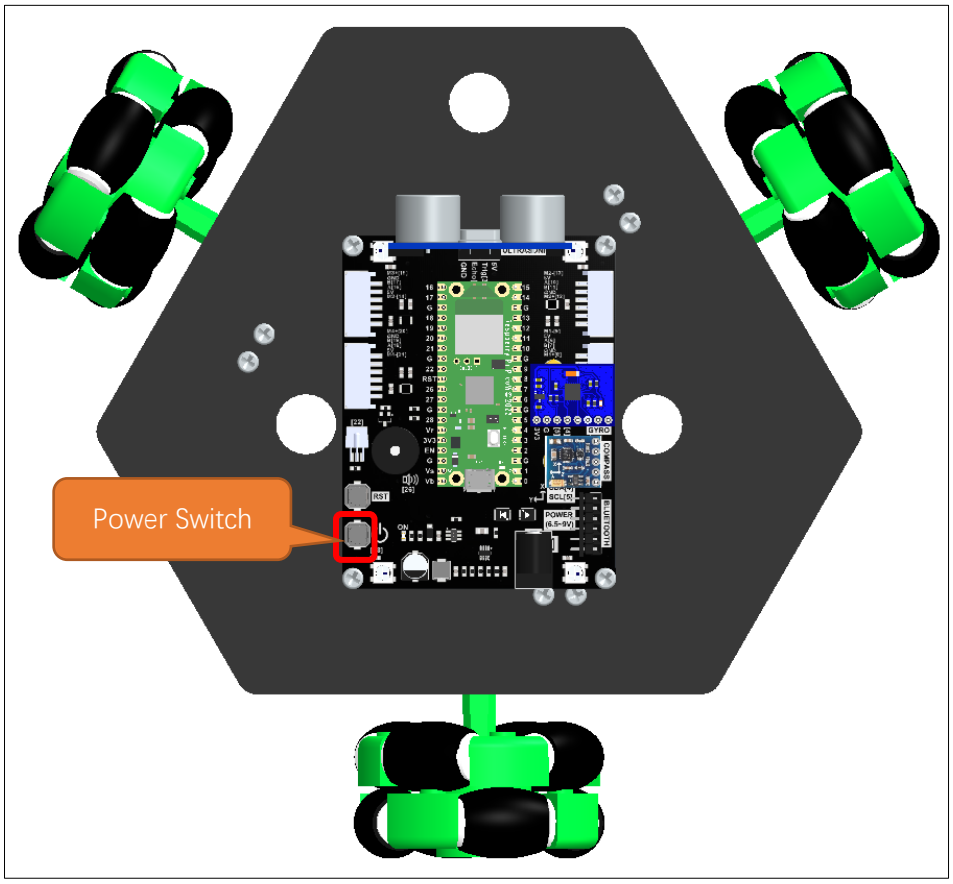


After the code finishes uploading, you can see Motor M1 rotate forward for one second, and then halt for one second, followed by rotating reversely for one second.

Notes:

1. The expansion board is designed with an anti-reverse flow circuit. If the motor does not respond after the code uploads, please check whether the power switch on the car expansion board is turned on.

2. There is a switch on the battery holder; please check if it is turned on.


Power Switch


Power Switch

Code Explanation

If you are not familiar with Arduino IDE, you can visit https://www.arduino.cc/reference/en/

Define the drive pins for the motor. In this code, wheel1_A_pin corresponds to M1_IN1 pin, and wheel1_B_pin to M1-IN2.

```
1  const int wheel1_A_pin = 8; // Define motor drive pins
2  const int wheel1_B_pin = 9; // Define motor drive pins
```

In the Initialization function Setup(), configure the motor driving pins to output mode.

```
6  pinMode(wheel1_A_pin,OUTPUT);
7  pinMode(wheel1_B_pin,OUTPUT);
```

Continously call the motor control function in the loop function loop() to have the motors runs in different directions.

```
15 void loop() {
16   // put you main code here, to run repeatedly:
17   digitalWrite(wheel1_A_pin,1);  // Forward rotation
18   digitalWrite(wheel1_B_pin,0);
19   delay(1000);
20
21   digitalWrite(wheel1_A_pin,0);  //stop
22   digitalWrite(wheel1_B_pin,0);
23   delay(1000);
24
25   digitalWrite(wheel1_A_pin,0);  // Contrarotation
26   digitalWrite(wheel1_B_pin,1);
27   delay(1000);
28
29   digitalWrite(wheel1_A_pin,0);  //stop
30   digitalWrite(wheel1_B_pin,0);
31   delay(1000);
32 }
```

Reference

const

In Arduino programming, 'const' is a keyword used to define constant variables. Variables defined with 'const' cannot be modified during the program execution; otherwise, it will cause a compilation error.

void pinMode(int pin, int mode) ;

Configures the specified pin to behave either as an input or an output.

Parameters

pin: the pin number to set the mode of Motor.

mode: INPUT, OUTPUT, INPUT_PULLDOWM, or INPUT_PULLUP.

void digitalWrite (int pin, int value) ;

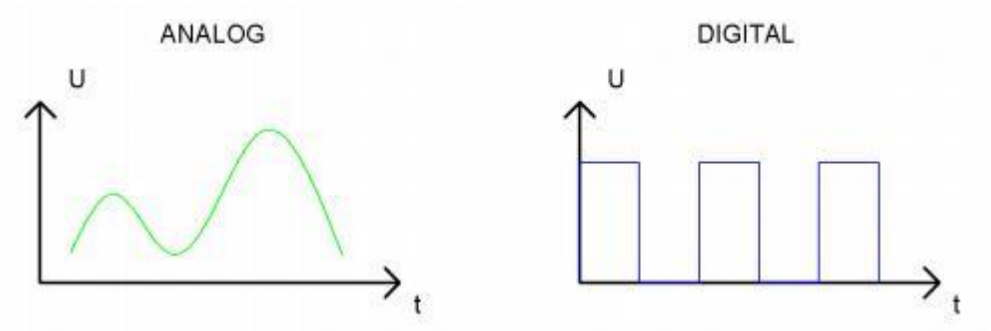
Writes the value HIGH or LOW (1 or 0) to the given pin which must have been previously set as an output.

4.2 Motor Speed Control through PWM

Related Knowledge

Analog & Digital

An Analog Signal is a continuous signal in both time and value. On the contrary, a Digital Signal or discrete - time signal is a time series consisting of a sequence of quantities. Most signals in life are analog signals. A familiar example of an Analog Signal would be how the temperature throughout the day is continuously changing and could not suddenly change instantaneously from 0℃ to 10℃. However, Digital Signals can instantaneously change in value. This change is expressed in numbers as 1 and 0 (the basis of binary code). Their differences can more easily be seen when compared when graphed as below.

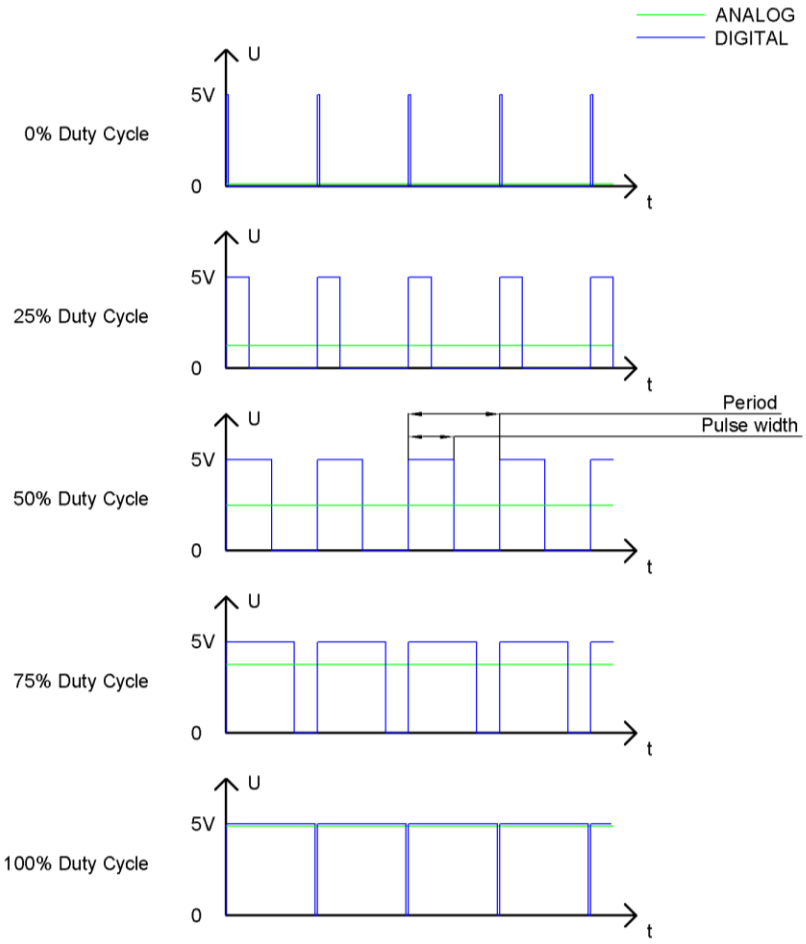


In practical application, we often use binary as the digital signal, which is a series of 0's and 1's. Since a binary signal only has two values (0 or 1), it has great stability and reliability. Lastly, both analog and digital signals can be converted into the other.

PWM

PWM, Pulse Width Modulation, uses digital pins to send certain frequencies of square waves, that is, the output of high levels and low levels, which alternately last for a while. The total time for each set of high levels and low levels is generally fixed, which is called the period (the reciprocal of the period is frequency). The time of high level outputs are generally called "pulse width", and the duty cycle is the percentage of the ratio of pulse duration, or pulse width (PW) to the total period (T) of the waveform.

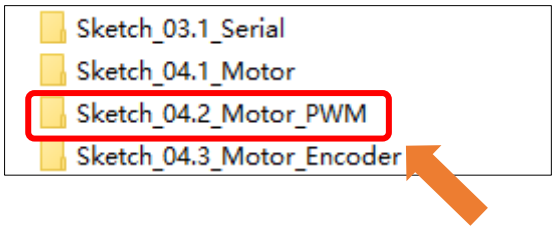
The longer the output of high levels last, the larger the duty cycle and the higher the corresponding voltage in analog signal will be. The following figures show how the analogs signal voltage vary between 0V-5V (high level is 5V) corresponding to the pulse width 0%-100%:



The longer the PWM duty cycle is, the higher the output power will be. Now that we understand this relationship, we can use PWM to control the speed of DC motor.

Sketch

Next, we download the code to Raspberry Pi Pico (W) to change the speed of the motors through PWM. Open "**Sketch_02.2_Motor_PWM**" folder under **"Freenove_Omni_Wheel_Car_Kit_for_Raspberry_Pi_Pico\Four-Wheel\Sketch"** and double-click "**Sketch_02.2_Motor_PWM.ino**".



Code

```
const int wheel1_A_pin = 8;// Define motor drive pins
const int wheel1_B_pin = 9;// Define motor drive pins

void Motor_init()
{
  analogWriteFreq(16000); // Set the motor PWM frequency to 16kHz
  pinMode(wheel1_A_pin,OUTPUT);
  pinMode(wheel1_B_pin,OUTPUT);
}

void setup() {
  // put your setup code here, to run once:
  Motor_init();// Motor initialization
}

void loop() {
   // put you main code here, to run repeatedly:
   // Stop
   analogWrite(wheel1_A_pin,0);
   analogWrite(wheel1_B_pin,0);
   delay(1500);// Delay 1.5 second
   // The speed is 200
   analogWrite(wheel1_A_pin,200);
   analogWrite(wheel1_B_pin,0);
   delay(1500);// Delay 1.5 second
   // The speed is 255
   analogWrite(wheel1_A_pin,255);
   analogWrite(wheel1_B_pin,0);
   delay(1500);// Delay 1.5 second
}
```

After downloading the code, in a clockwise direction, the motor M1 starts from a stationary state, accelerates to its maximum speed, and then slow down to return to a stationary state, repeating this process.

Code Explanation

If you are not familiar with Arduino IDE, you can visit https://www.arduino.cc/reference/en/ to learn more.
Set the motor's PWM frequency in the initialization function Setup().

```
6  analogWriteFreq(16000);// Set the motor PWM frequency to 16kHz
```

Set the motor driving pins to output mode.

```
7  pinMode(wheel1_A_pin,OUTPUT);
8  pinMode(wheel1_B_pin,OUTPUT);
```

Continuously change the motors running speed in the loop function.

```
19 analogWrite(wheel1_A_pin,0);
20 analogWrite(wheel1_B_pin,0);
21 delay(1500);// Delay 1.5 second
22 // The speed is 200
23 analogWrite(wheel1_A_pin,200);
24 analogWrite(wheel1_B_pin,0);
25 delay(1500);// Delay 1.5 second
26 // The speed is 255
27 analogWrite(wheel1_A_pin,255);
28 analogWrite(wheel1_B_pin,0);
29 delay(1500);// Delay 1.5 second
```

Reference

void analogWriteFreq(uint32_t frequency);

analogWriteFreq function is used to set the PWM signal frequency, where frequency is the PWM frequency to be set, in Hertz (Hz).

void analogWrite(uint8_t pin, int value);

Arduino IDE provides the function, analogWrite(pin, value), which can make ports directly output PWM waves. Every pin on Pico board can be configured to output PWM. In the function called analogWrite(pin, value),the parameter "pin" specifies the port used to output PWM wave. The range of value is 0-255, which represents the duty cycle of 0%-100%.

In order to use this function, we need to set the port to output mode.

4.3 TT Motor with Encoder

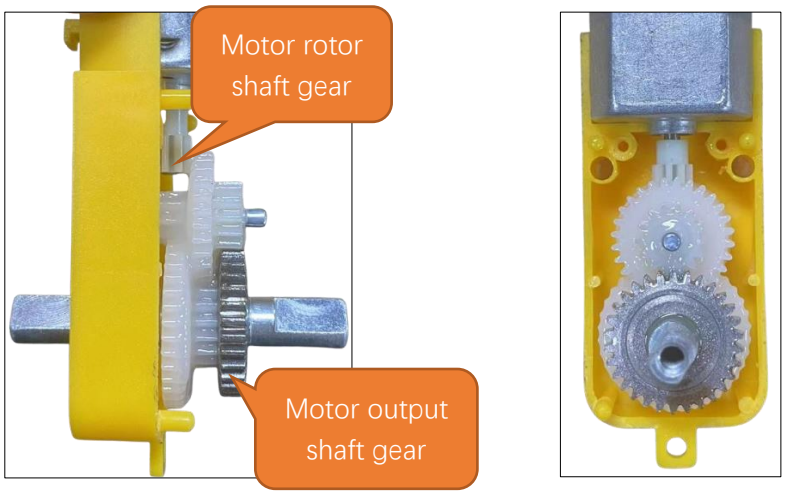
Within the realms of automation and robotics, motor control is a key technology. Simple pulse width modulation (PWM) speed control is not sufficient for achieving the high level of accuracy. To precisely determine the motor's speed and thereby achieve high-precision motor control, we have used encoder-equipped motors on this product. Below is an introduction to these motors with encoder.

Related Knowledge

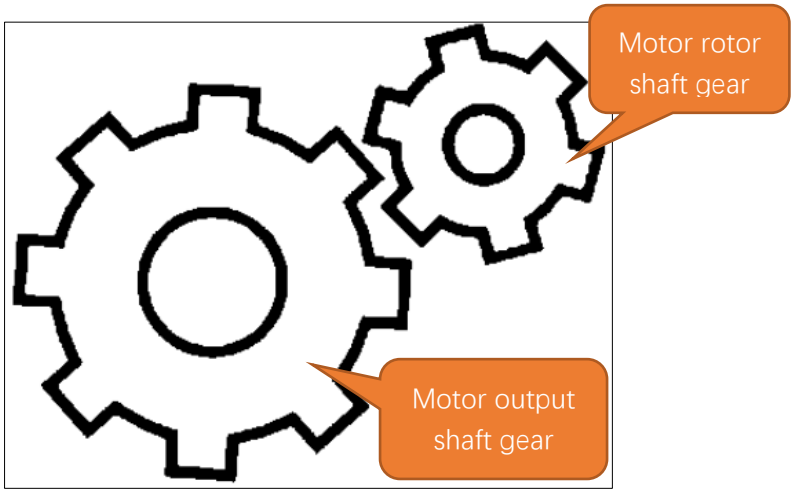
The main body of the motors consists of a hall encoder and a gear reduction motor.

Gear Reduction Motor

Compared to regular motors, a gear reduction motor has an additional gearbox, which is a mechanical transmission device. It applies the combination of large and small gears to change the gear ratio, designed to reduce the motor's speed and increase its torque. The internal structure of the motor gearbox is shown in the figure below. The gearbox contains multiple gear stages, and the total gear ratio is the product of the gear ratios of each stage. The gear ratio is equal to the number of teeth on the output gear divided by the number of teeth on the input gear.



The gear set inside the gearbox can be simplified as the following model:



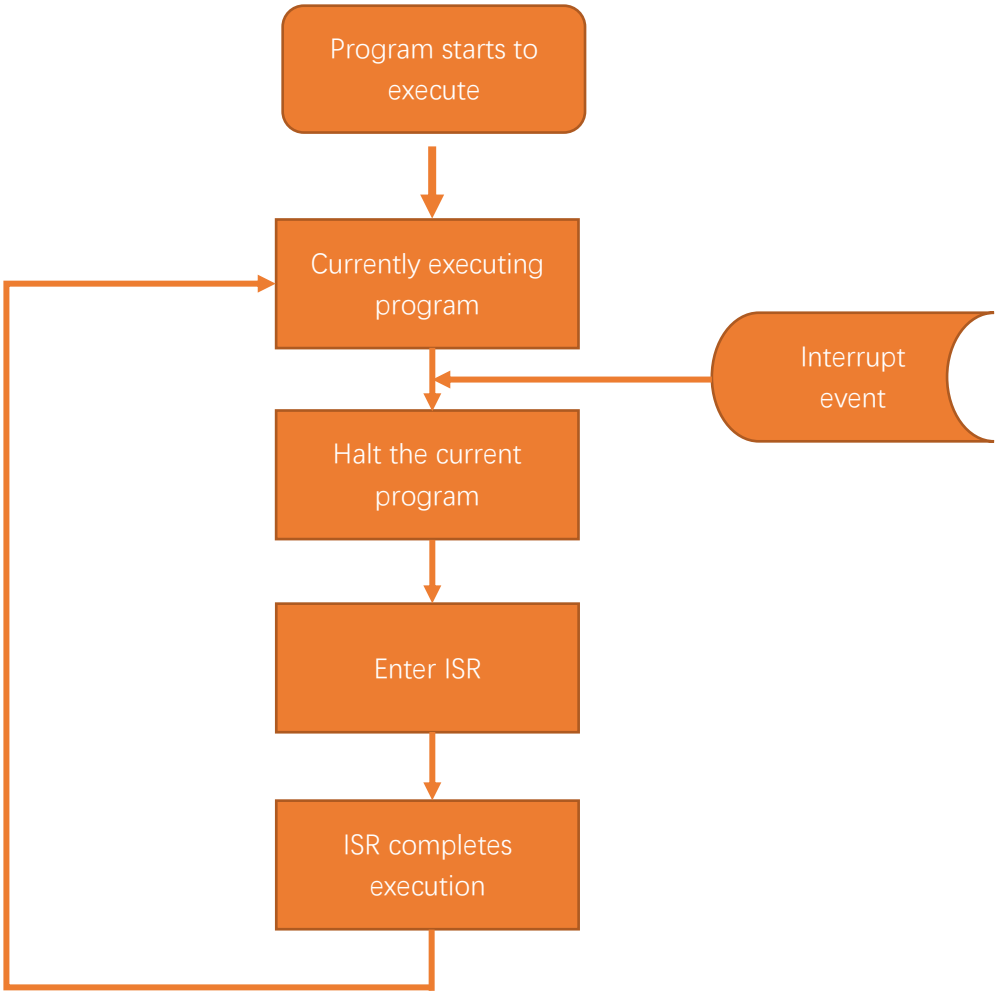
In this model, when the rotor connection shaft rotates N times, the main shaft rotates once, achieving the purpose of reducing the motor speed while increasing the motor torque. At this point, the reduction ratio of the motor is 1:N.

Note that the value of N is determined by the specific parameters of the motor. In the omnidirectional wheel robot, we provide a motor with a reduction ratio of 1:48, meaning the rotor connection shaft rotates 48 times for the main shaft to rotate once.

If you need any support, please feel free to contact us via: support@freenove.com

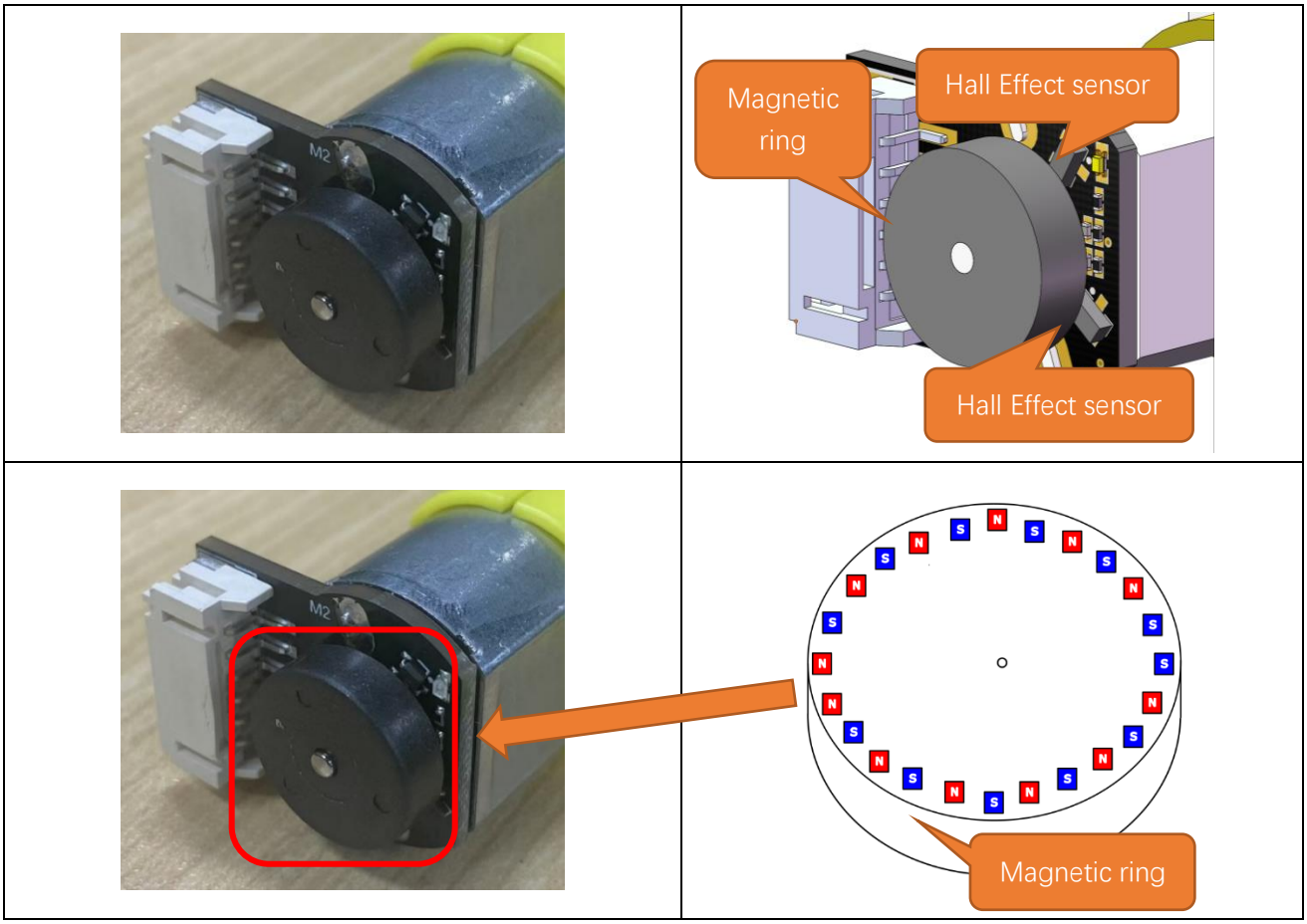
How Interrupts Work

When an interrupt event is triggered, the Raspberry Pi Pico (W) receives an interrupt signal. At this moment, the Raspberry Pi Pico (W) pauses the currently executing program and instead executes the Interrupt Service Routine (ISR). The ISR contains the code that needs to be processed after the interrupt event occurs. Once the ISR execution is complete, the Raspberry Pi Pico will return to the state it was in before the interrupt occurred and continue executing the paused program, as shown in the diagram below.

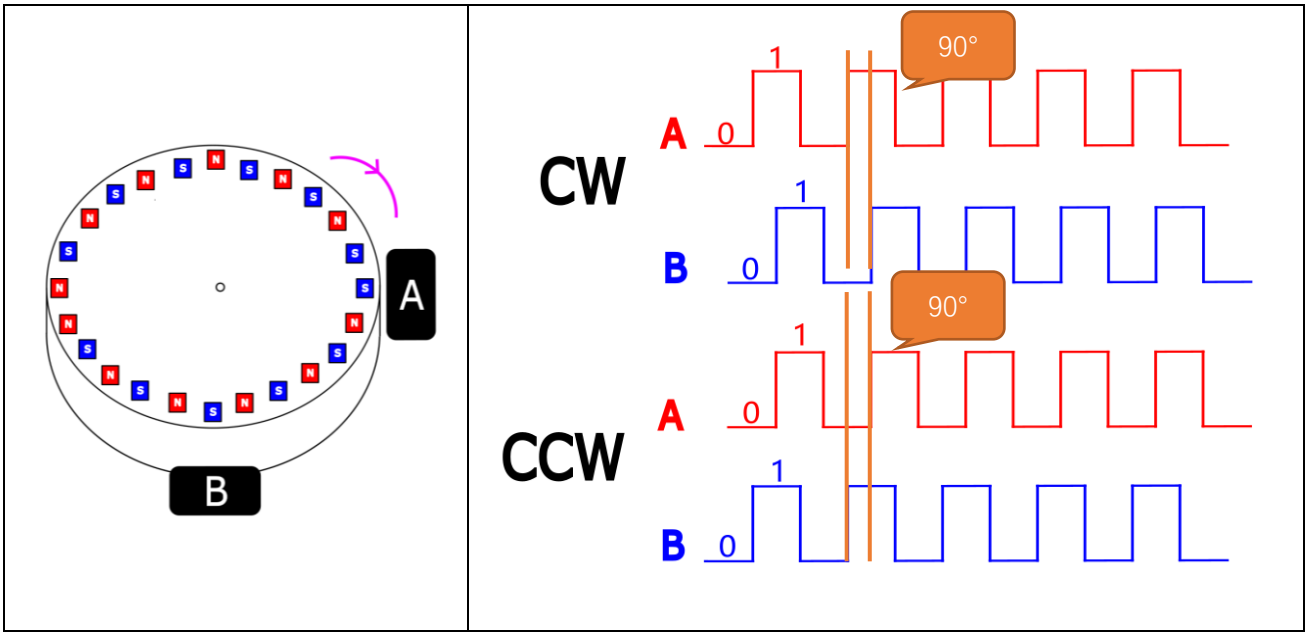


Hall Encoder

The Hall encoder consists of a magnetic ring and two Hall effect sensors. The magnetic ring has 12 pairs of poles, which means there are 12 North (N) poles and 12 South (S) poles, as shown in the diagram below:



As the magnetic ring rotates, the magnetic poles on the ring continuously move past the Hall effect sensors. These sensors produce different voltage level changes depending on whether they encounter a North (N) pole or a South (S) pole on the magnetic ring, as illustrated in the diagram below:

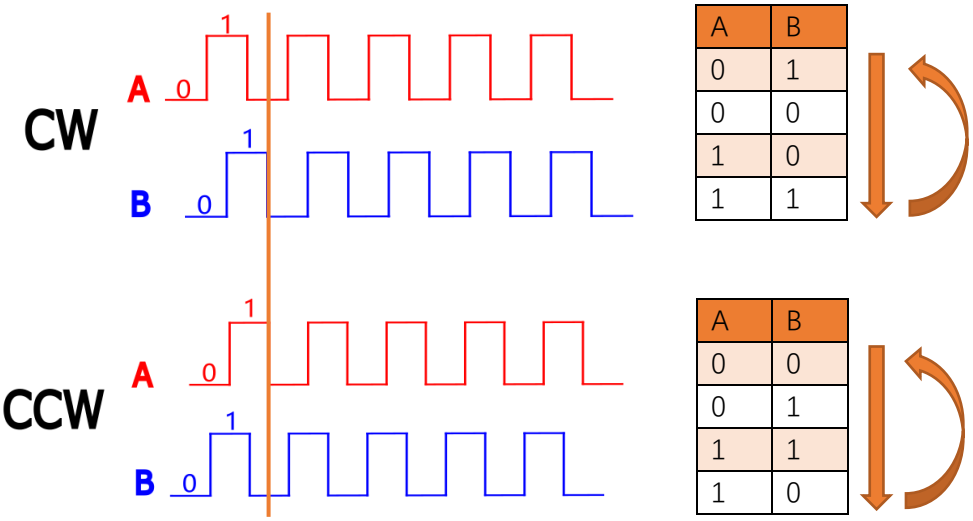


Code Knowledge

Encoder Value Obtaining

The Hall effect sensors have two output signals, A and B. By analyzing their outputs, the direction of rotation can be determined.

Since A and B are 90° out of phase, their voltage level changes occur in a sequential order. By examining the changes in these levels, the direction of rotation can be identified, as shown in the diagram below:



A	B
0	1
0	0
1	0
1	1

A	B
0	0
0	1
1	1
1	0

From the diagram above, it's clear that the combination order of A and B varies with the direction of rotation. By comparing the previous A and B combination to the current one, we can determine the direction of rotation.

For instance:
If the previous A and B combination is 11 and the current combination is 01, we can refer to the table in the diagram to conclude that the rotation direction is clockwise.

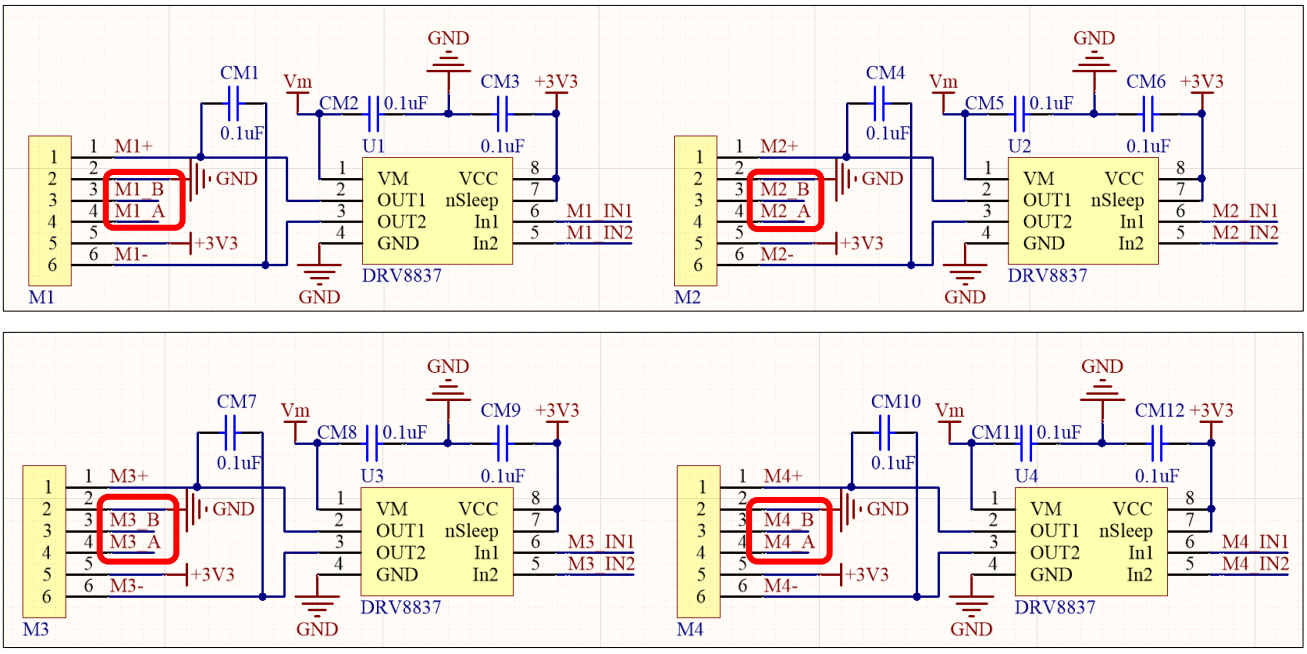
If you need any support, please feel free to contact us via: support@freenove.com

Schematic

The four-wheel omnidirectional robot collects analog signals using a Raspberry Pi Pico (W). It utilizes analog inputs and hardware interrupts to capture the electrical signals generated by the encoders, allowing it to determine the current speed and position of the motors.

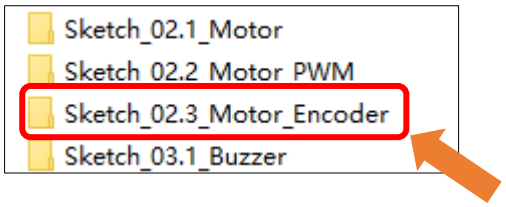
The following diagram illustrates the encoder motor circuitry, where:
M1_A corresponds to GPIO6
M1_B corresponds to GPIO7
M2_A corresponds to GPIO10
M2_B corresponds to GPIO11
M3_A corresponds to GPIO16
M3_B corresponds to GPIO17
M4_A corresponds to GPIO18
M4_B corresponds to GPIO19.

Please note that this car kit uses four 48:1 gear reduction TT motors with encoder. For each complete revolution of the wheels, the magnetic ring on the motor rotates 48 times. Each encoder motor outputs 13 pulses per revolution, resulting in a total of 2496 pulse signals for each complete turn of the motor.



Sketch

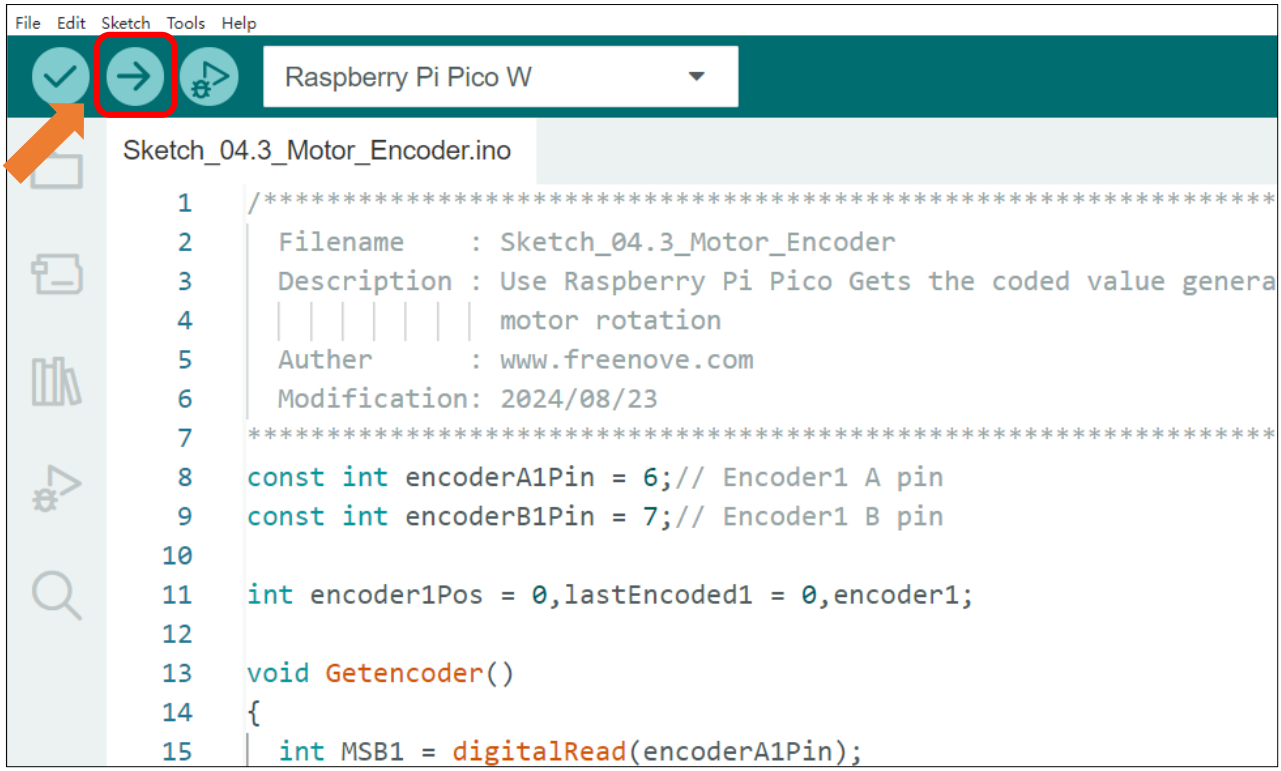
Next, we download the code to Raspberry Pi Pico (W) to have the TT motor with encoder read encoder value. Open "**Sketch_02.3_Motor_Encoder**" folder under "**Freenove_Omni_Wheel_Car_Kit_for_Raspberry_Pi_Pico\Four-Wheel\Sketch**" and double-click "**Sketch_02.3_Motor_Encoder**".



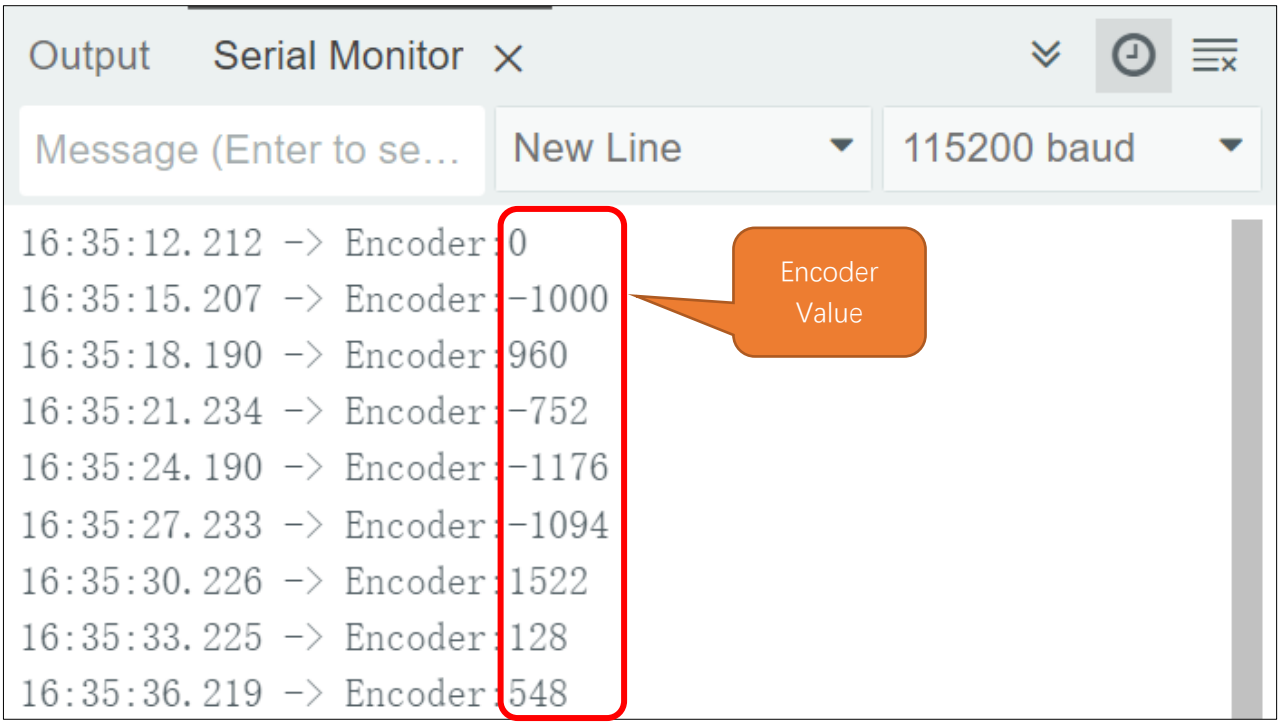
Code

```
const int encoderA1Pin = 6;// Encoder1 A pin
const int encoderB1Pin = 7;// Encoder1 B pin

int encoder1Pos = 0,lastEncoded1 = 0,encoder1;
void Getencoder()
{
   int MSB1 = digitalRead(encoderA1Pin);
   int LSB1 = digitalRead(encoderB1Pin);

   int encoded1 = (MSB1 << 1) | LSB1;
   int sum1 = (lastEncoded1 << 2) | encoded1;

   // Determine the motor rotation direction
   if (sum1 == 0b1101 || sum1 == 0b0100 || sum1 == 0b0010 || sum1 == 0b1011) encoder1Pos++;
   if (sum1 == 0b1110 || sum1 == 0b0111 || sum1 == 0b0001 || sum1 == 0b1000) encoder1Pos--;

   lastEncoded1 = encoded1;
}
void setup() {
   // put your setup code here, to run once:
   // Set the serial port baud rate to 115200
   Serial.begin(115200);
   // Enable encoder pin to input mode
   pinMode(encoderA1Pin,INPUT);
   pinMode(encoderB1Pin,INPUT);
   // Enter interrupt when level reverses
   attachInterrupt(digitalPinToInterrupt(encoderA1Pin), Getencoder, CHANGE);
   attachInterrupt(digitalPinToInterrupt(encoderB1Pin), Getencoder, CHANGE);
}
void loop() {
   // put your main code here, to run repeatedly:
   // The encoder's value is printed every 3 seconds
   Serial.print("Encoder:");
   Serial.println(encoder1Pos);
   encoder1Pos = 0;
   delay(3000);
}
```

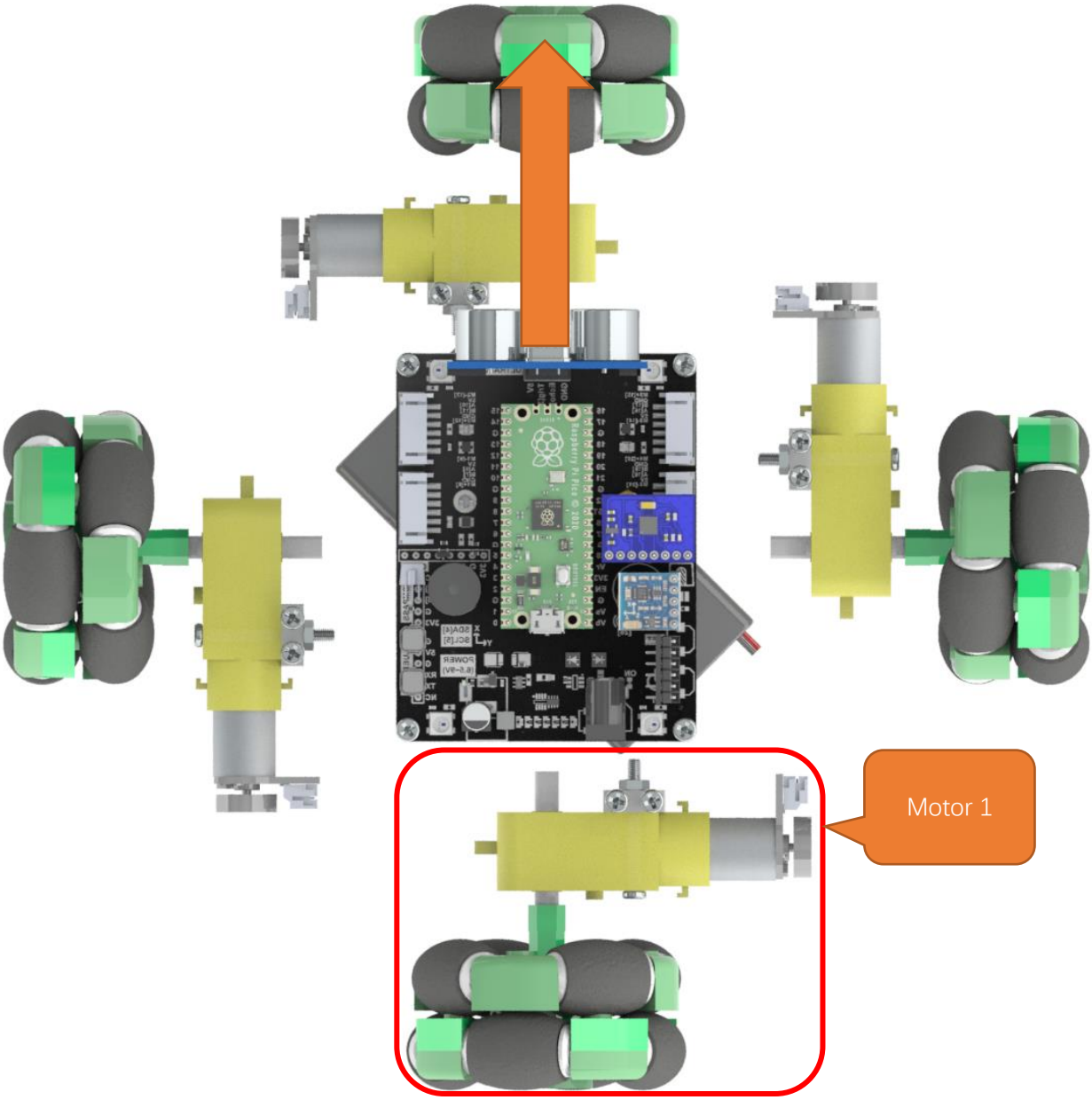
This code allows the pico to change the motors' speed. Click "**Upload**" to download the code.



After downloading the code, manually rotate Motor1, and the serial monitor will print the encoder value every three seconds. The larger the rotation amplitude, the greater the encoder value.



The position of Motor 1 is as shown below:



Code Explanation

If you are not familiar with Arduino IDE, you can visit https://www.arduino.cc/reference/en/ to learn more.

Define the object of interrupt timer.

```
1  RPI_PICO_Timer ITimer0(0);
```

Enable the encode pin to input mode.

```
6  pinMode(encoderA1Pin, INPUT);
7  pinMode(encoderB1Pin, INPUT);
```

Enable the external interrupt to enter the Getencoder function when the encoder pin level changes.

```
29 attachInterrupt(digitalPinToInterrupt(encoderA1Pin), Getencoder, CHANGE);
31 attachInterrupt(digitalPinToInterrupt(encoderB1Pin), Getencoder, CHANGE);
```

In the interrupt service function, retrieve the encoder value.

```
10 int MSB1 = digitalRead(encoderA1Pin);
11 int LSB1 = digitalRead(encoderB1Pin);
12 int encoded1 = (MSB1 << 1) | LSB1;
13 int sum1 = (lastEncoded1 << 2) | encoded1;
14 if (sum1 == 0b1101 || sum1 == 0b0100 || sum1 == 0b0010 || sum1 == 0b1011)
15   encoder1Pos++;
16 if (sum1 == 0b1110 || sum1 == 0b0111 || sum1 == 0b0001 || sum1 == 0b1000)
17 encoder1Pos--;
```

Output the encoder value every two seconds.

```
34 void loop() {
35   // put your main code here, to run repeatedly:
36   encoder1Pos = 0;
37   delay(2000);
38   Serial.println(encoder1Pos);
39 }
```

Reference

<<

The left shift operator is a bitwise operator that performs a left shift operation. The left side of the operator specifies the data to be shifted, while the right side indicates how many positions to shift. This operation moves the binary bits of a number to the left by the specified number of positions, filling in zeros on the right.

|

The bitwise OR operator is a bitwise operator that performs an OR operation on the bits of the numbers on its left and right. The rule of OR operation is that if one of the two bits is 1, the result is 1; otherwise, it is 0.

||

The logical OR operator is a logical operator commonly used for conditional testing. The rule for the logical OR operation is that if at least one side of the operator is true, the result is true; otherwise, it is false.

int digitalRead (int pin);

This function returns the value read at the given pin. It will be "HIGH" or "LOW"(1 or 0) depending on the logic level at the pin.

void attachInterrupt(pin_size_t pin, voidFuncPtr callback, PinStatus mode)

The attachInterrupt function is used to set up an external interrupt that calls the interrupt service routine (ISR) when the controller pin level changes.

Parameters:

pin: The interrupt number of the pin to be monitored.
callback: The name of the interrupt service routine (ISR) function.
mode:
LOW: Triggered when the pin is low.
HIGH: Triggered when the pin is high.
FALLING: Triggered when the pin transitions from high to low.
RISING: Triggered when the pin transitions from low to high.
CHANGE: Triggered when the pin level changes.

`digitalPinToInterrupt (uint8_t pin)`

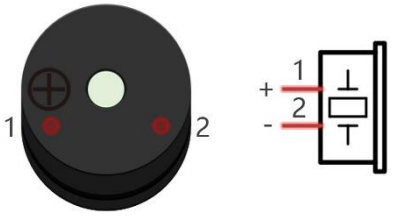
The digitalPinToInterrupt function maps a digital pin to its corresponding interrupt number. The interrupt numbers can vary between different controllers, so this function helps ensure that the correct interrupt is used for the specified digital pin.

Chapter 5 Buzzer Test

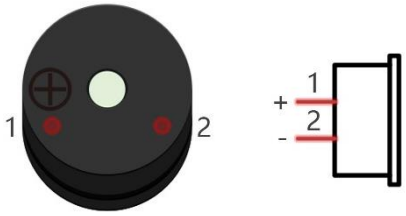
Related Knowledge

Buzzer is a sounding component, which is widely used in electronic devices such as calculator, electronic warning clock and alarm. Buzzer has two types: active and passive. Active buzzer has oscillator inside, which will sound as long as it is supplied with power. Passive buzzer requires external oscillator signal (generally use PWM with different frequency) to make a sound.

Active buzzer



Passive buzzer



Active buzzer is easy to use. Generally, it can only make a specific frequency of sound. Passive buzzer requires an external circuit to make a sound, but it can be controlled to make a sound with different frequency. The resonant frequency of the passive buzzer is 2 kHz, which means the passive buzzer is loudest when its resonant frequency is 2 kHz.

How to identify active and passive buzzer

1. Usually, there is a label on the surface of active buzzer covering the vocal hole, but this is not an absolute judgment method.
2. Active buzzers are more complex than passive buzzers in their manufacture. There are many circuits and crystal oscillator elements inside active buzzers; all of this is usually protected with a waterproof coating (and a housing) exposing only its pins from the underside. On the other hand, passive buzzers do not have protective coatings on their underside. From the pin holes viewing of a passive buzzer, you can see the circuit board, coils, and a permanent magnet (all or any combination of these components depending on the model.

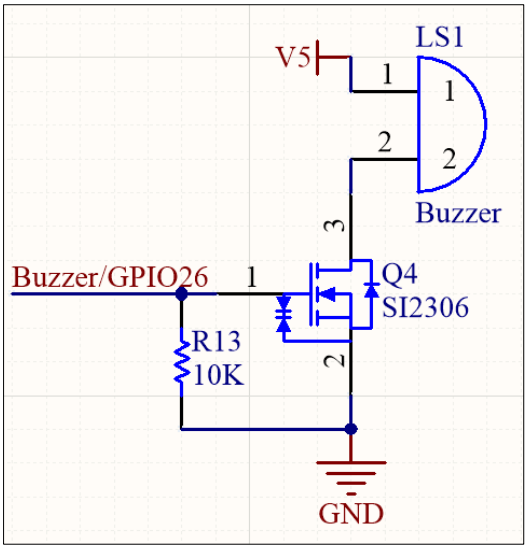
Active buzzer

Passive buzzer

The buzzer used in this car is a passive buzzer that can make sounds with different frequency.

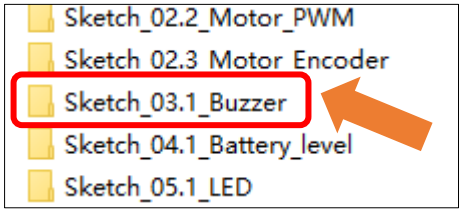
Schematic

As we can see, the buzzer is controlled by GPIO26 of Raspberry Pi Pico W. When the buzzer receives PWM signal, N-MOS will be activated to make the buzzer sound. When the buzzer receives no signal, it will be controlled at low level by R2 and N-MOS will not be activated, so the buzzer will not make any sounds.



Sketch

Next, we download the code to Raspberry Pi Pico (W) to test the buzzer. Open "03.1_Buzzer" folder under **"Freenove_Omni_Wheel_Car_Kit_for_Raspberry_Pi_Pico\Four-Wheel\Sketch"** and double-click **"03.1_Buzzer.ino"**.

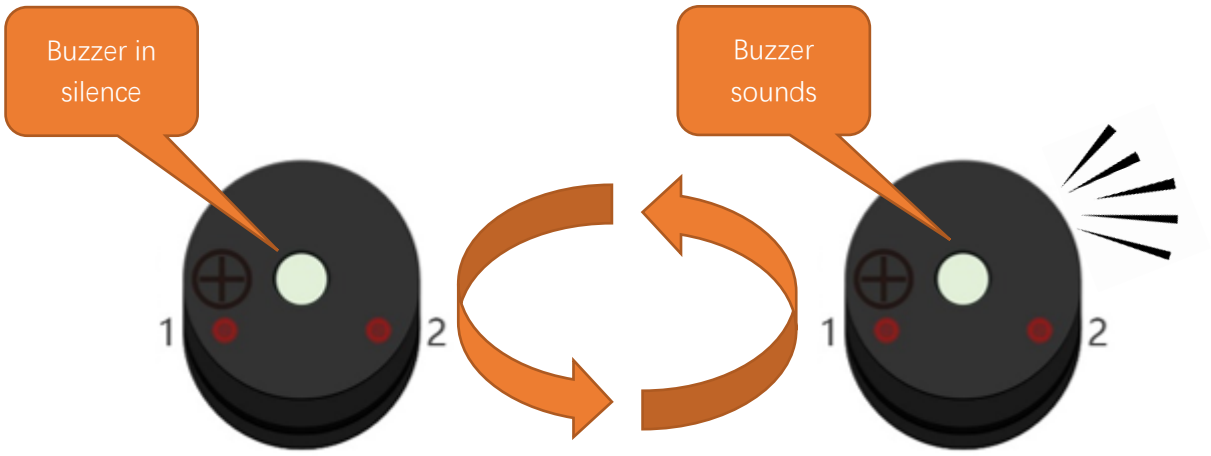


Code

```
1  const int PIN_BUZZER = 26;
2
3  void setup() {
4    // put your setup code here, to run once:
5    pinMode(PIN_BUZZER,OUTPUT);
6  }
7
8  void loop() {
```

```
9     // put your main code here, to run repeatedly:
10    tone(PIN_BUZZER,100);
11    delay(500);
12    tone(PIN_BUZZER,0);
13    delay(500);
14  }
```

After downloading the code, the buzzer makes sounds at 100Hz every 500 milliseconds.



Code Explanation

If you are not familiar with Arduino IDE, you can visit https://www.arduino.cc/reference/en/ to learn more.

Define the pin of buzzer.

```
1   const int PIN_BUZZER = 26;
```

Enable the buzzer pin to output mode.

```
5   pinMode(PIN_BUZZER,OUTPUT);
```

Call the function to have the buzzer make sounds at 100Hz every 500ms, and repeat this process.

```
8   void loop() {
9     // put your main code here, to run repeatedly:
10    tone(PIN_BUZZER,100);
11    delay(500);
12    tone(PIN_BUZZER,0);
13    delay(500);
14  }
```

Reference

void tone(uint8_t pin, unsigned int frequency, unsigned long duration = 0);

The tone function is commonly used to control a buzzer.

Parameters:

pin: The pin number to control.

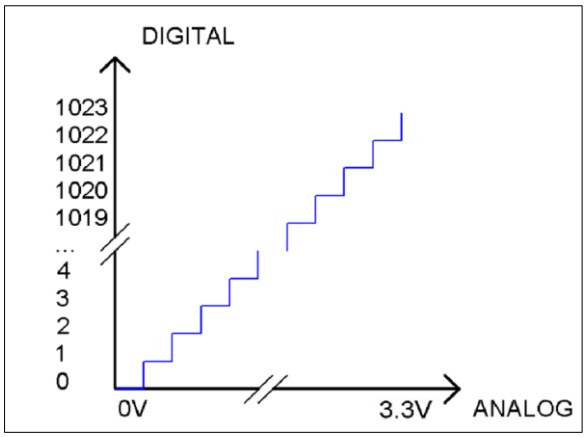
frequency: The frequency of the tone to be generated, in Hz, with a range of 0 to 65,535.

duration: (Optional) The duration for which the tone should play.

Chapter 6 ADC Test

ADC

ADC is an electronic integrated circuit used to convert analog signals such as voltages to digital or binary form consisting of 1s and 0s. The range of our ADC on Raspberry Pi Pico (W) is 10 bits, which means the resolution is 2^10=1024, and it represents a range (at 3.3V) will be divided equally to 1024 parts. The rage of analog values corresponds to ADC values. So the more bits the ADC has, the denser the partition of analog will be and the greater the precision of the resulting conversion.

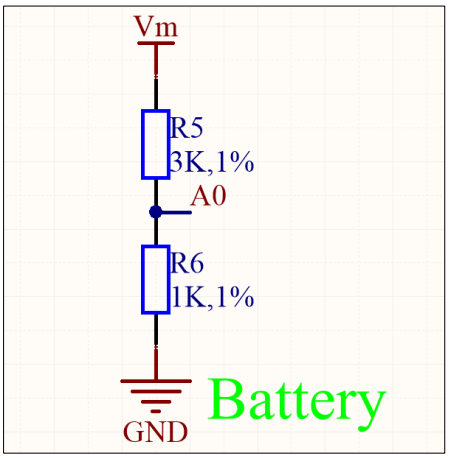


Subsection 1: the analog in rang of 0V---3.3/1023 V corresponds to digital 0;
Subsection 2: the analog in rang of 3.3/1023 V---2*3.3 /1023V corresponds to digital 1;
The following analog will be divided accordingly.
The conversion formula is as follows:

$$ADC\ Value = \frac{\text{Analog Voltage}}{3.3} * 1023$$

Schematic

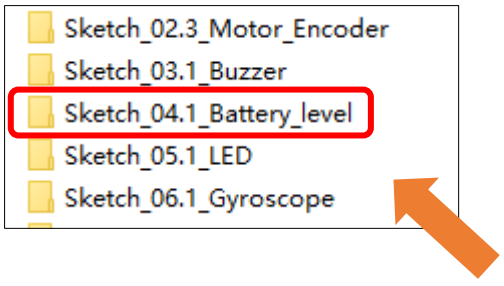
As we can see, the car reads the voltage of the batteries through GPIO27 of Raspberry Pi Pico (W).



The voltage acquisition range of GPIO27 on Raspberry Pi Pico (W) is 0-3.3V, while the car is powered by two 18650-lithium batteries, whose voltage is 8.4V when they are fully charged, which exceeds the acquisition range of Raspberry Pi Pico (W). Therefore, after passing through the voltage divider circuit composed of R3 and R4, the voltage at A0 will be about 1/4 of the battery voltage, 8.4/4=2.1V, which is within the voltage collection range of GPIO27.

Sketch

In this section, we will use GPIO27 of Raspberry Pi Pico (W) to read the voltage value of the batteries and print it on serial monitor. Open "Sketch_04.1_Battery_level" folder in **"Freenove_Omni_Wheel_Car_Kit_for_Raspberry_Pi_Pico\Four-Wheel\ \Sketches"** and then double-click "**Sketch_04.1_Battery_level.ino**".

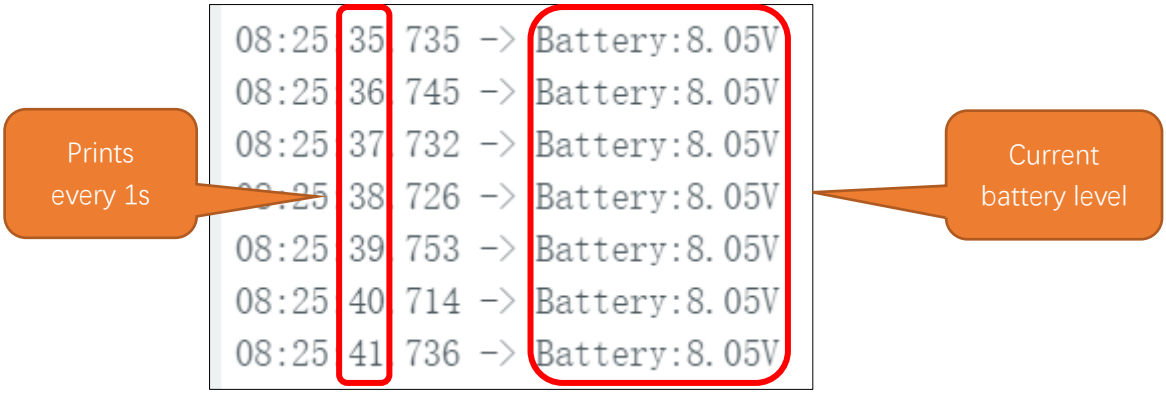


Code

```
1  const int Battery_ADC = 27;
2  float Battery;
3
4  void setup() {
5    // put your setup code here, to run once:
6    pinMode(Battery_ADC,INPUT);
```

```
7     Serial.begin(9600);
8   }
9
10  void loop() {
11    // put your main code here, to run repeatedly:
12    Battery = analogRead(Battery_ADC)*3.3/1023*4;
13    Serial.println(Battery);
14    delay(1000);
15  }
```

After downloading the code, we can see current battery level be printed on serial monitor every 1 second.



Code Explanation

Define the ADC sampling pin.

```
1   const int Battery_ADC = 27;
```

Set the pin to input mode.

```
6   pinMode(Battery_ADC,INPUT);
```

Initialize the pico serial port and set the baud rate to 9600.

```
7   Serial.begin(9600);
```

Obtain the battery voltage every 1 second and print on Serial Monitor.

```
10  void loop() {
11    // put your main code here, to run repeatedly:
12    Battery = analogRead(Battery_ADC)*3.3/1023*4;
13    Serial.println(Battery);
14    delay(1000);
15  }
```

Reference

int analogRead(uint8_t pin);

The analogRead function is used to read an analog input. Upon a successful read, it returns an integer value in the range of 0 to 1023.

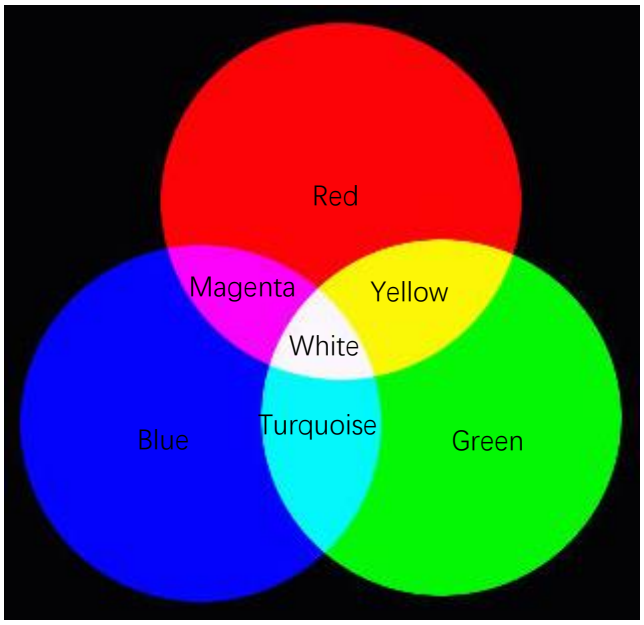
Parameters:

pin: The analog input pin number to read from.

Chapter 7 LED Test

Related Knowledge

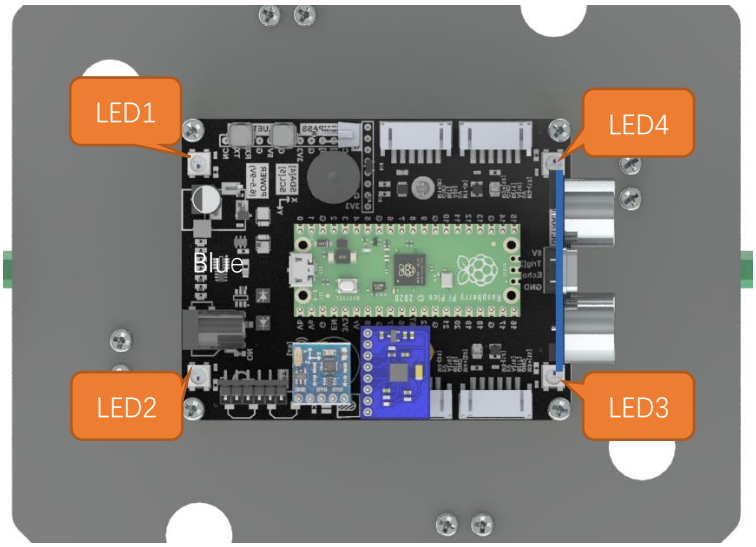
Red, green, and blue are called the three primary colors. When you combine these three primary colors of different brightness, it can produce almost all kinds of visible light.



RGB

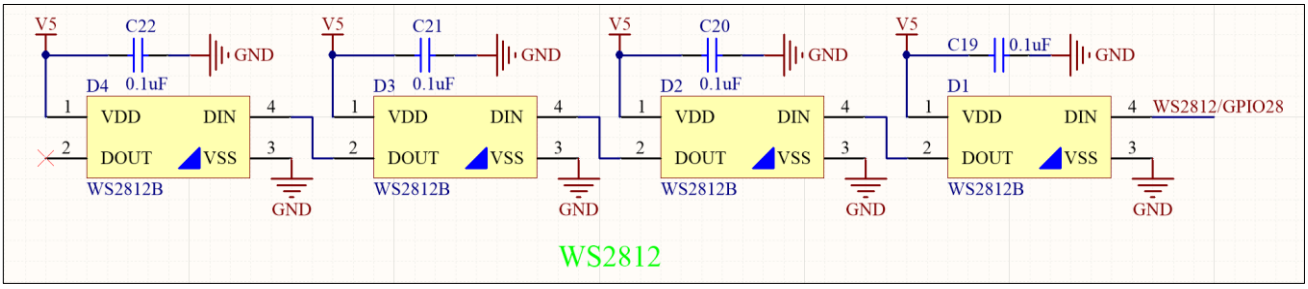
The LED of the car is composed of eight LED, each of which is controlled by one pin and supports cascading. Each LED can emit three basic colors of red, green and blue, and supports 256-level brightness adjustment, which means that each LED can emit 2^24=16,777,216 different colors.

We know from previous section that, control board controls LEDs to emit a total of 256(0-255) different brightness with PWM. So, through the combination of three different colors of LEDs, RGB LED can emit 256^3=16777216 Colors, 16Million colors.



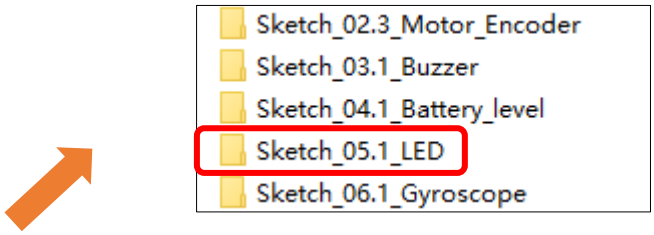
Schematic

As shown below, the DOUT of each LED is connected with DIN of the next LED, and the 4 LED can be controlled to emit colorful colors by inputting control signals through LED.



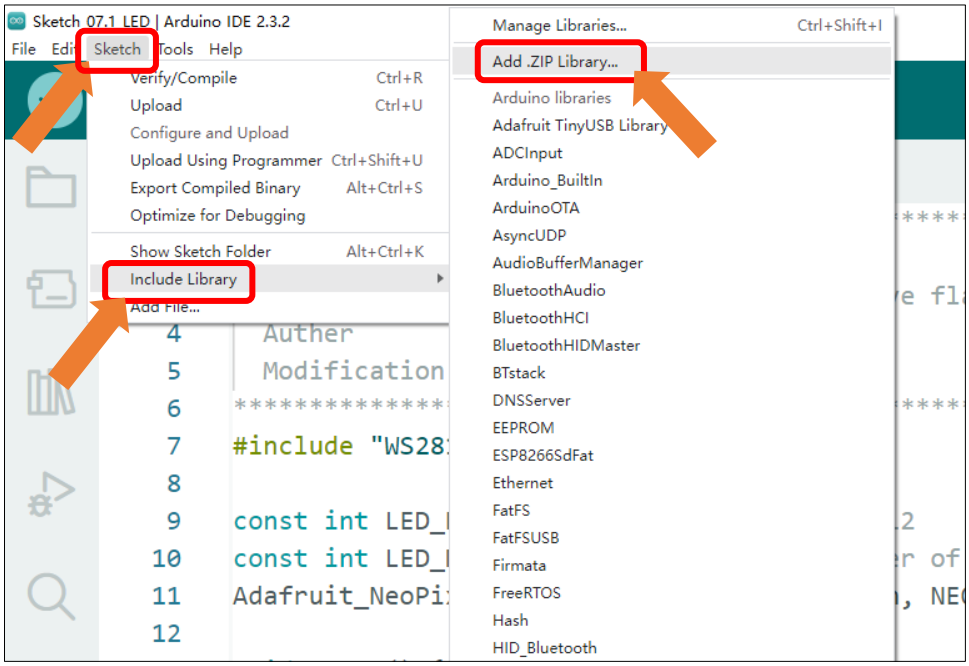
Sketch

Next, we will download the code to Raspberry Pi Pico (W) to test the LED. Open the folder **"Sketch_05.1_LED"** in the **"Freenove_Omni_Wheel_Car_Kit_for_Raspberry_Pi_Pico\Four-Wheel\ \Sketches"** and then double click **"Sketch_05.1_LED.ino"**.

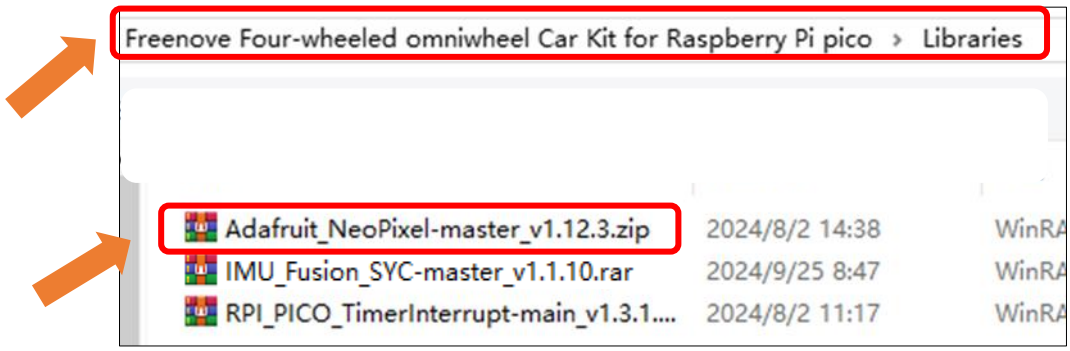


Before uploading the sketch, please make sure the LED driver library has been installed. If not, please install it as follows.

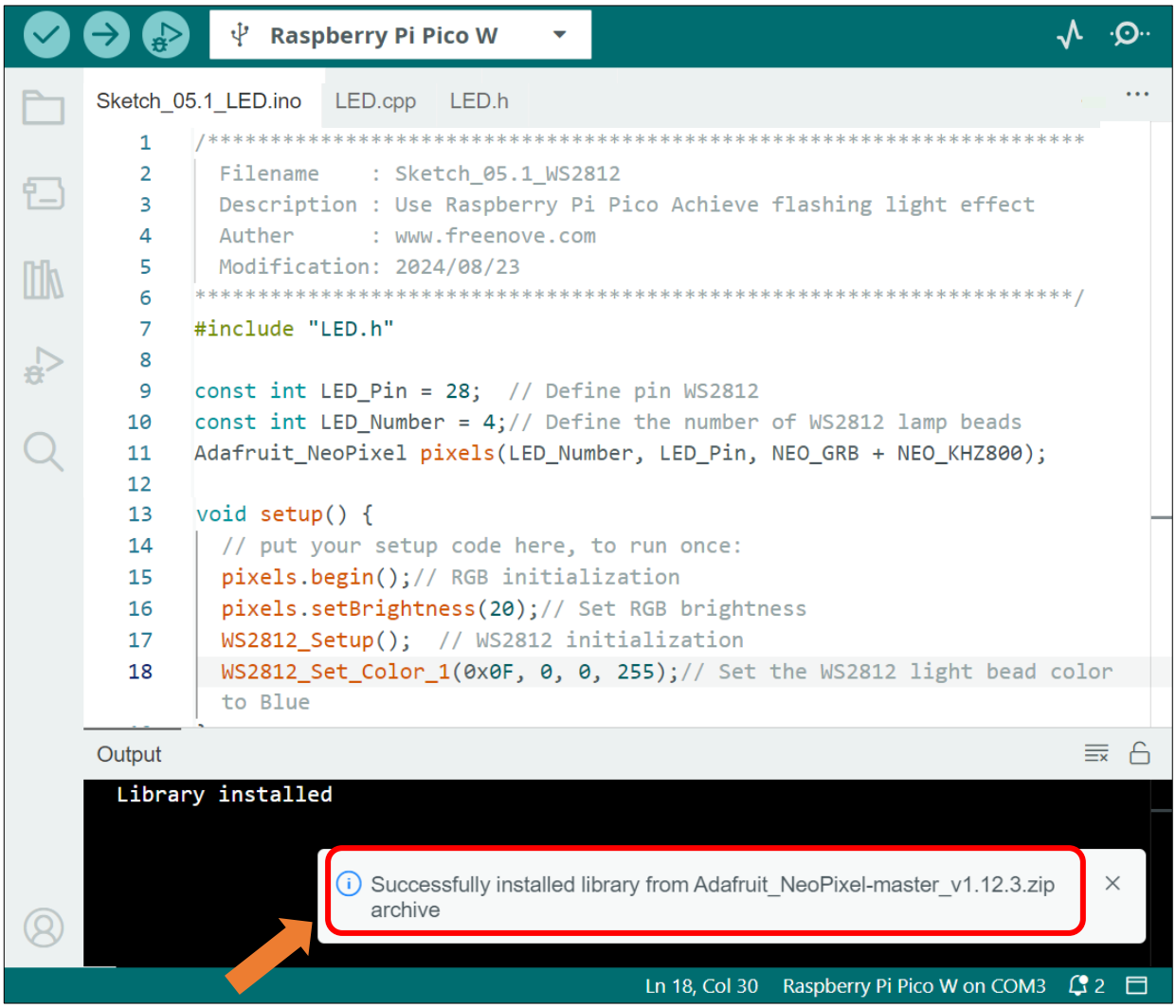
Open Arduino IDE, click **"Sketch"** >- **"Include Library"** >- **"Add.ZIP Library..."**.



Select **"Adafruit_NeoPixel-master_v1.12.3.zip" under the directory of "Freenove Four-wheeled omniwheel Car Kit for Raspberry Pi pico \Libraries".**



Wait for the installation to finish.

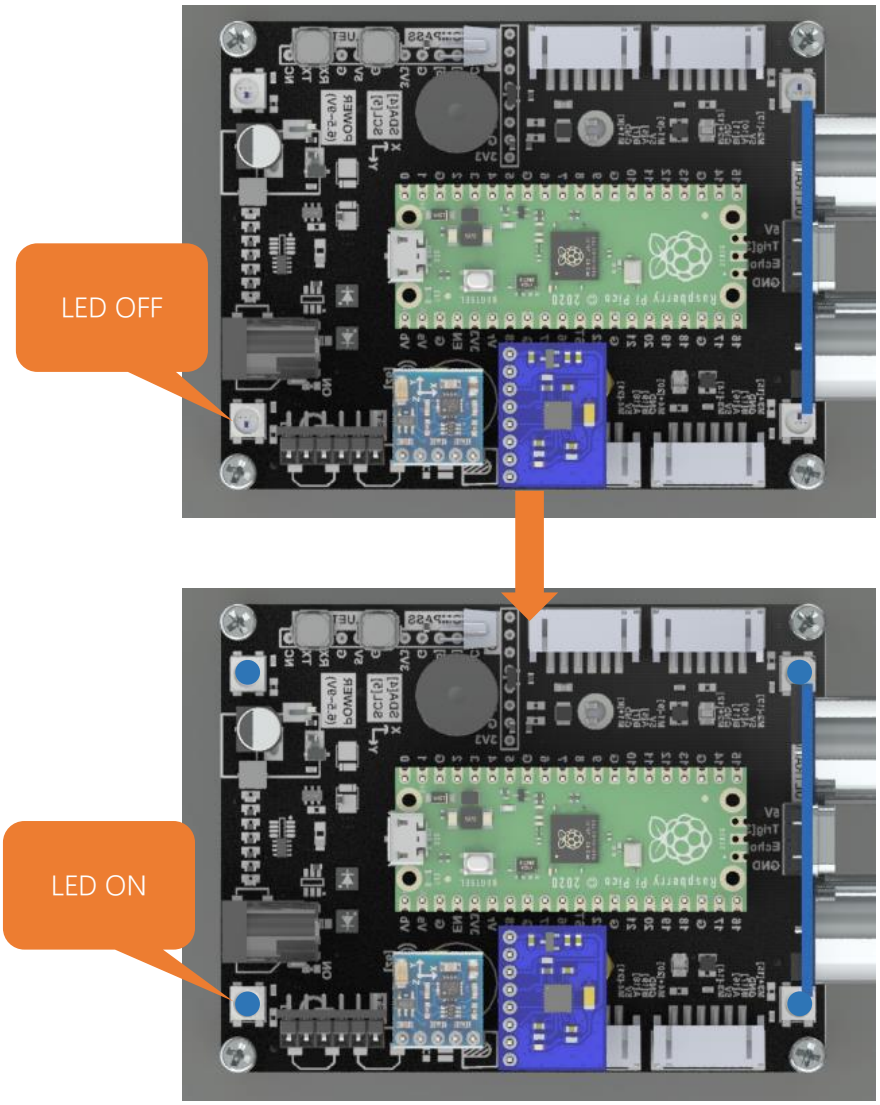


Code

```
#include "LED.h"

const int LED_Pin = 28;  // Define pin LED
const int LED_Number = 4;// Define the number of LED lamp beads
Adafruit_NeoPixel pixels(LED_Number, LED_Pin, NEO_GRB + NEO_KHZ800);
```

```
6
7   void setup() {
8       // put your setup code here, to run once:
9       pixels.begin();// RGB initialization
10      pixels.setBrightness(20);// Set RGB brightness
11      WS2812_Setup(); // WS2812 initialization
12      WS2812_Set_Color_1(0x0F, 0, 0, 255);// Set the WS2812 light bead color to Blue
13  }
14
15  void loop() {
16      // put your main code here, to run repeatedly:
17      WS2812_Show(1);// Achieve flashing light effect
18  }
```

Verify and upload the code, the four WS2812 LEDs will turn blue at the same time.



Code Explanation:

Add the header file of LED. Each time before controlling LED, please add its header file.

```
1   #include <WS2812.h>
```

Define LED pin.

```
4  const int LED_Pin = 28;  // Define pin WS2812
```

Define LED number.

```
1  const int LED_Number = 4;// Define the number of WS2812 lamp beads
```

Initialize RGB and set the brightness to 20.

```
11 pixels.begin();// RGB initialization
12 pixels.setBrightness(20);// Set RGB brightness
```

Initialize LED and set the color.

```
48 WS2812_Setup(); // WS2812 initialization
49 WS2812_Set_Color_1(0x0F, 0, 0, 255);// Set the WS2812 light bead color to Blue
```

Display the color of LED. After setting the color, you need to call show function to display it.

```
51 ws2812_strip.show(1);
```

Reference

void WS2812_Show(int mode)

The WS2812_Show function can change the LED mode.

Parameters:

0: LED off
1: Steady on mode
2: Chasing mode
3: Blinking mode
4: Breathing mode
5: Rainbow mode

Chapter 8 Gyroscope Test

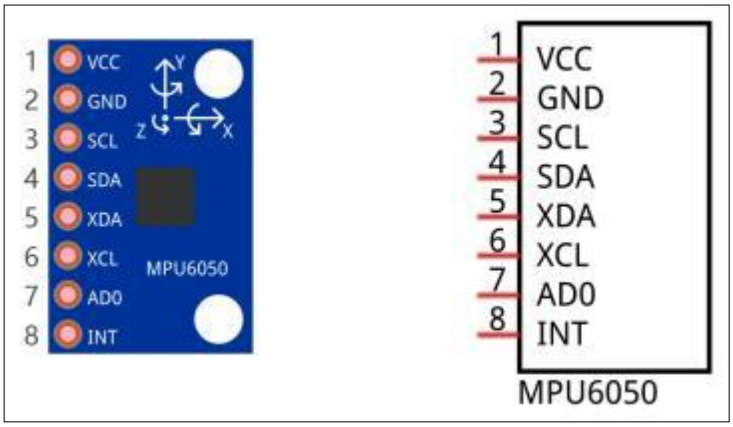
Component Knowledge

I2C communication

I2C (Inter-Integrated Circuit) is a two-wire serial communication mode, which can be used to the connection of micro controller and its peripheral equipment. Devices using I2C communication must be connected to the serial data (SDA) line, and serial clock (SCL) line (called I2C bus). Each device has a unique address and can be used as a transmitter or receiver to communicate with devices connected to the bus.

MPU6050 Gyroscope

The MPU6050 follows the I2C communication protocol and the default address is 0x68. It is widely used to assist with balancing vehicles, robots and aircraft, mobile phones and other products which require stability to control stability and attitude or which need to sense same.



The port description of the MPU6050 module is as follows:

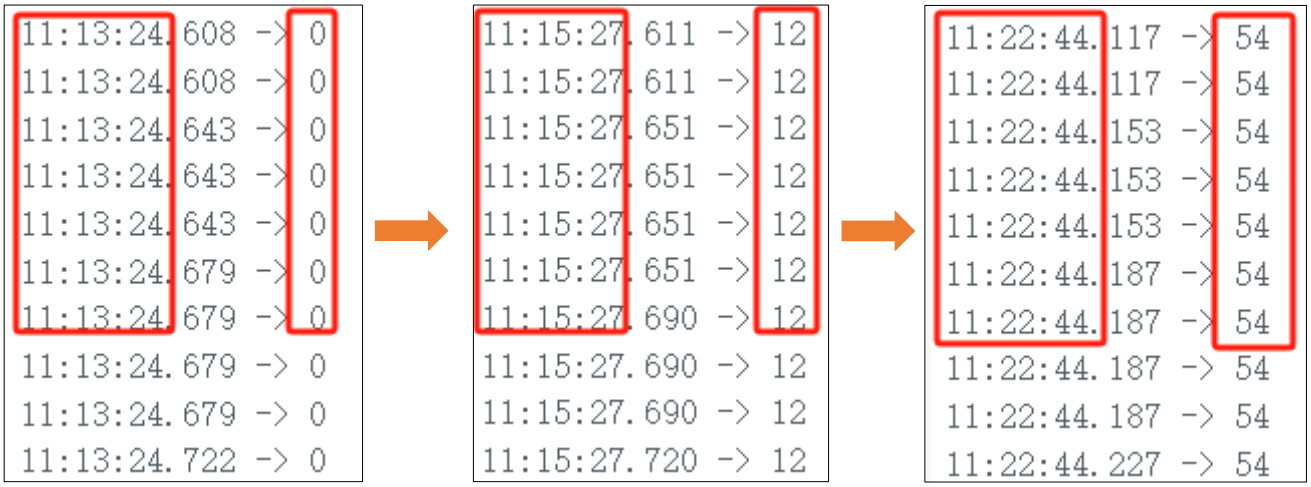
Pin name	Pin number	Description
VCC	1	Positive pole of power supply with voltage 5V
GND	2	Negative pole of power supply
SCL	3	I2C communication clock pin
SDA	4	I2C communication clock pin
XDA	5	I2C host data pin which can be connected to other devices.
XCL	6	I2C host clock pin which can be connected to other devices.
AD0	7	I2C address bit control pin. Low level: the device address is 0x68 High level: the device address is 0x69

INT	8	Output interrupt pin

For more details, please refer to datasheet.

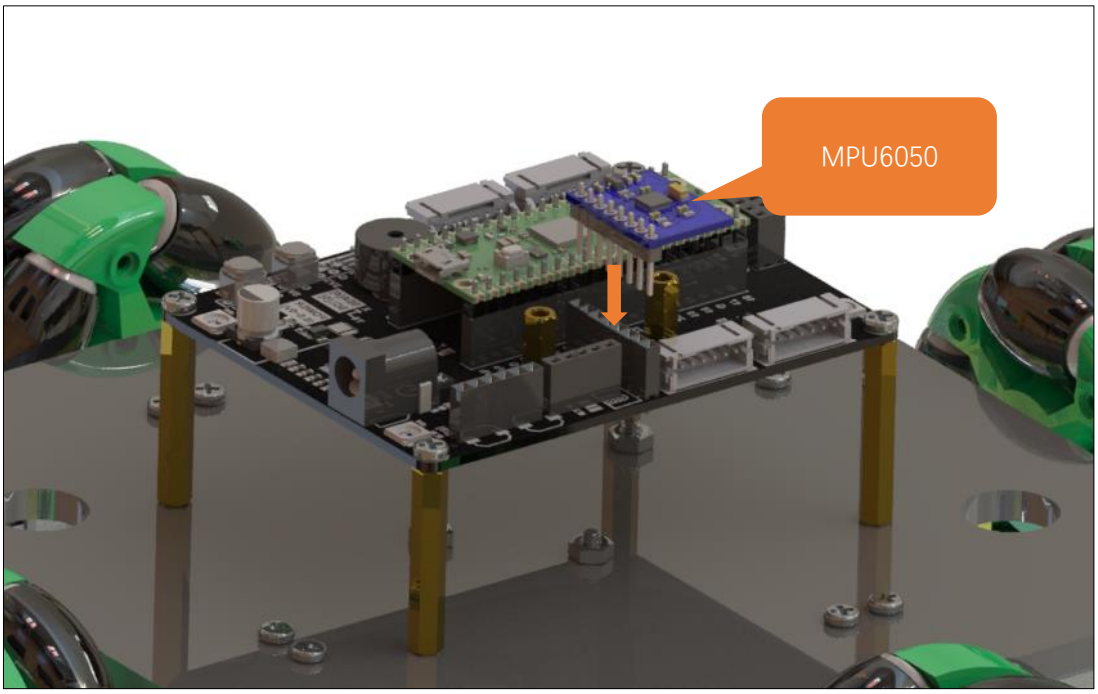
Zero Drift

During the calibration of a gyroscope, the initial angle measurement may be accurate; however, over time, the zero position tends to drift, resulting in an accumulating error. This phenomenon is referred to as zero drift. The underlying cause is that obtaining angle information from the gyroscope's raw data requires integration. In this process, the angular velocity signal often contains small biases and drifts. These biases accumulate through integration, leading to progressively increasing errors over time, which can eventually cause circuit saturation, preventing the output of accurate angle measurements.



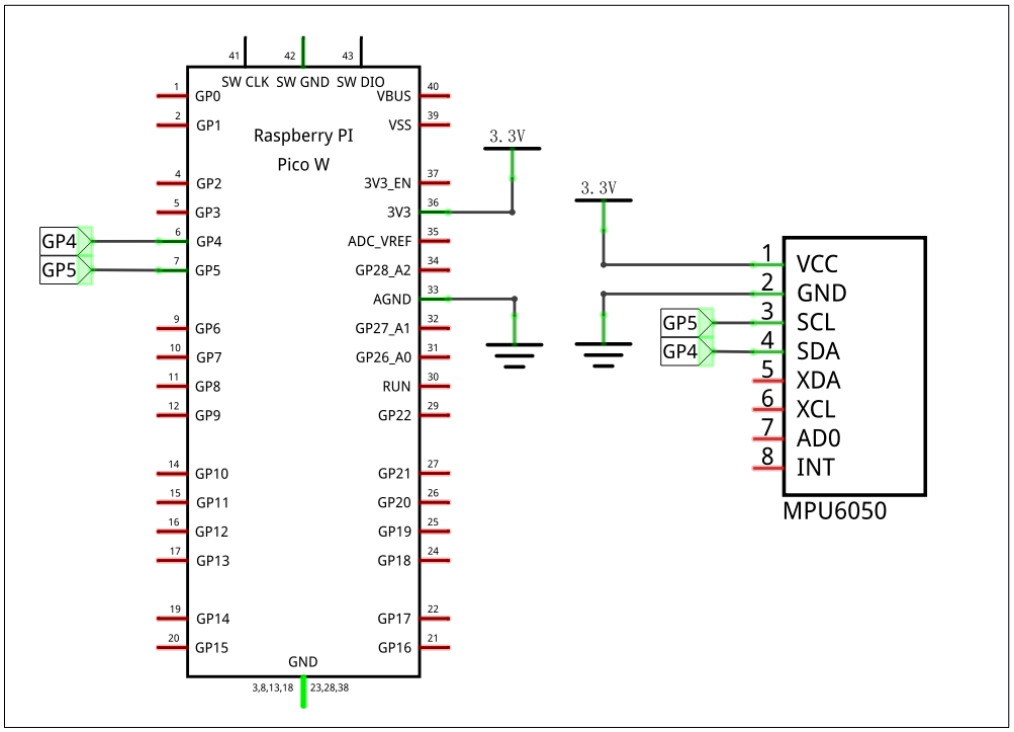
Circuit

Plug the MPU6050 chip to the expansion board.



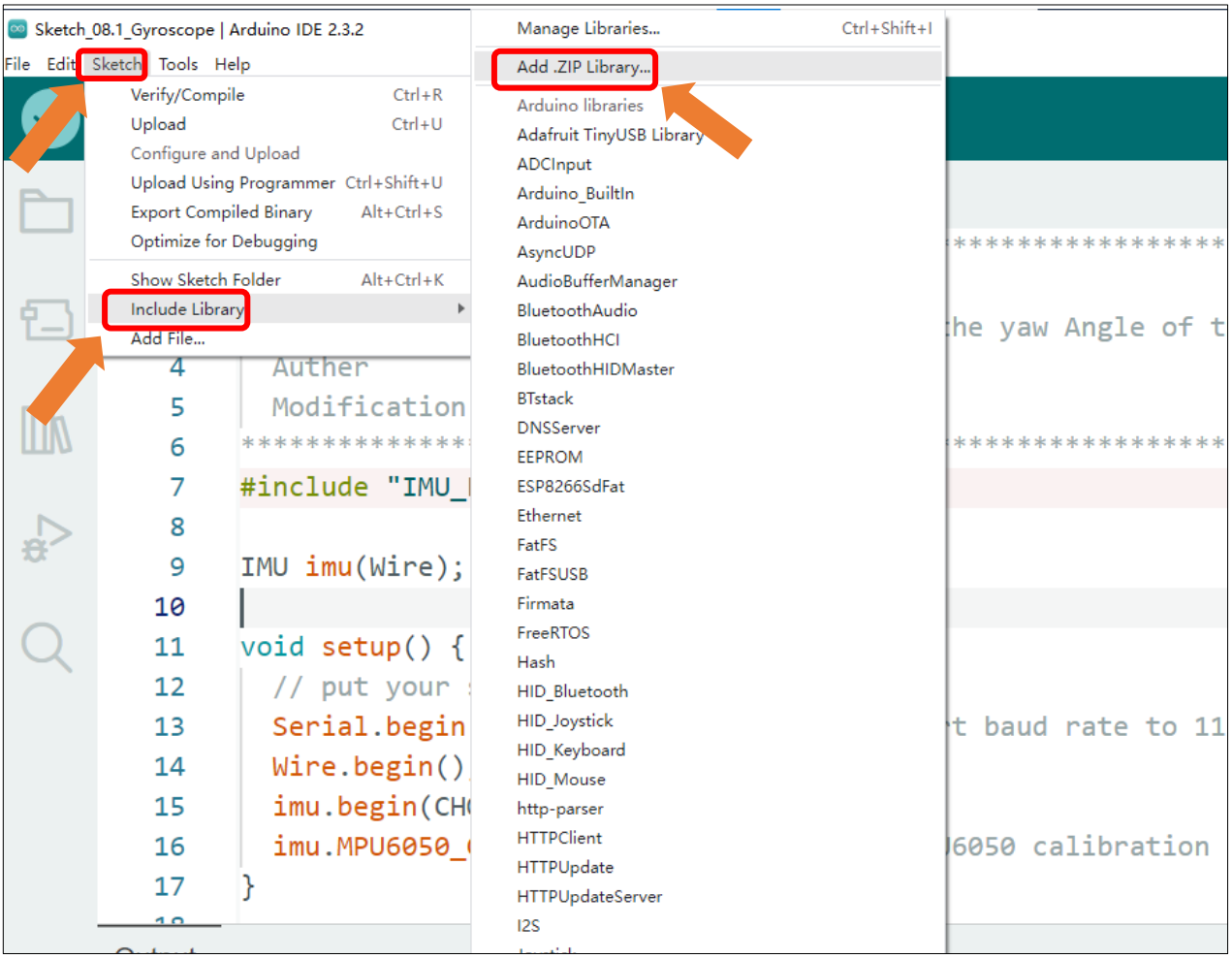
Schematic

As can be seen from the schematic below, the SDA and SCL of the MPU6050 are connected to GP4 and GP5 of the Pico (W).

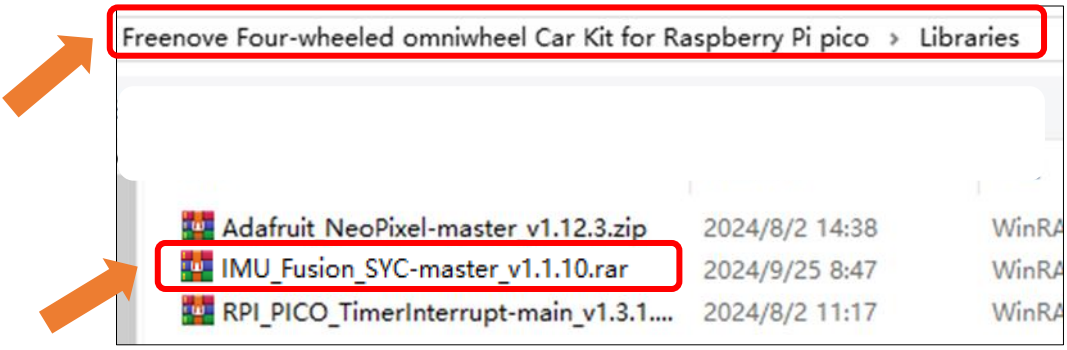


Sketch

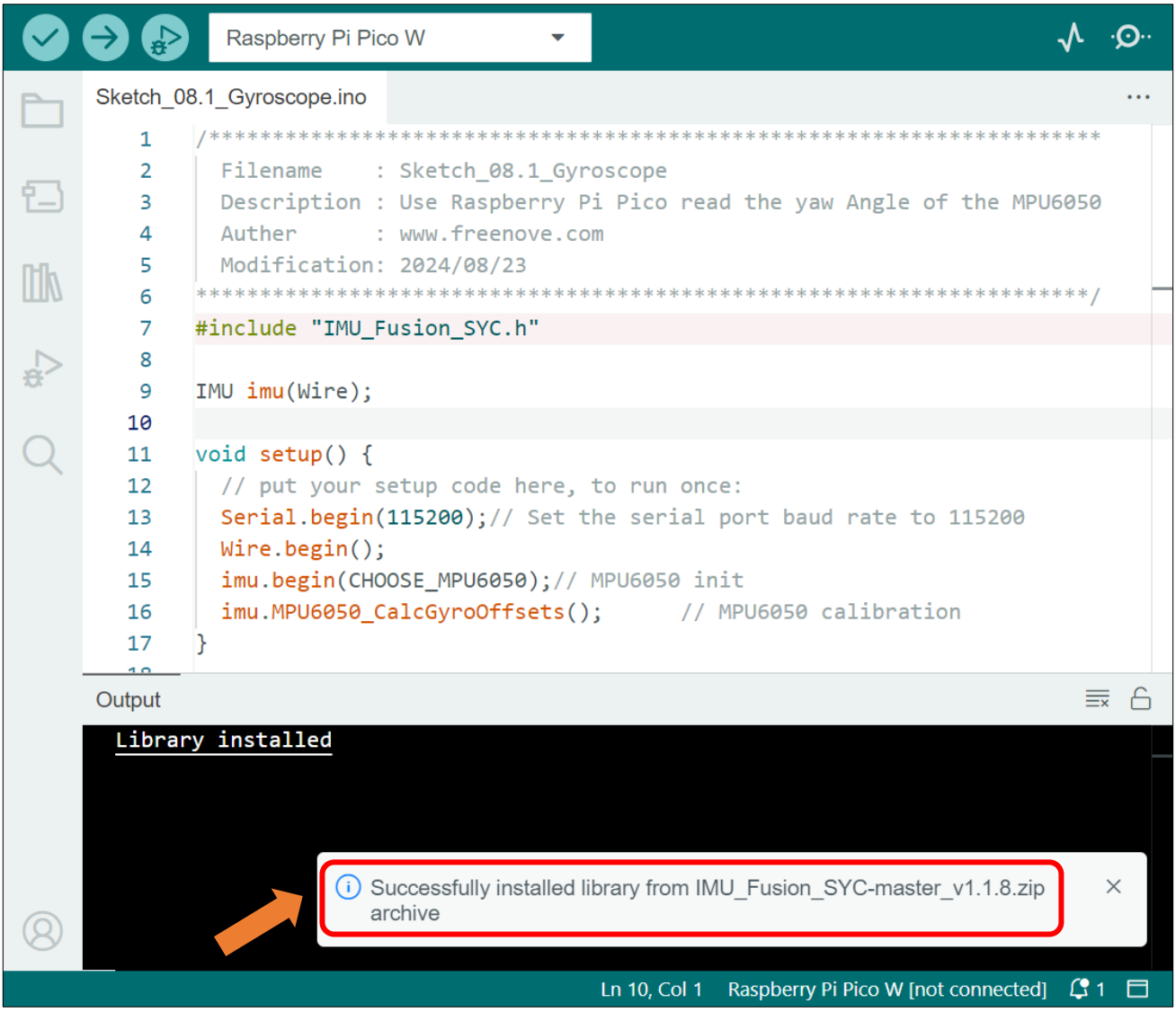
As we use the I2C interface to read the gyrpscope's data, we need to install the related library before use. Open Arduino IDE, click **"Sketch"** -> **"Include Library" ->"Add.ZIP Library..."**.



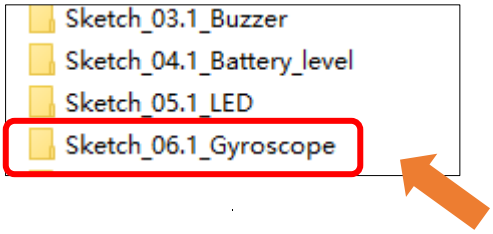
Select **"IMU_Fusion_SYC-master_v1.1.10.zip"** under the directory of **"Freenove Four-wheeled omniwheel Car Kit for Raspberry Pi pico \Libraries"**.



Wait for the installation to finish.



Open "Sketch_08.1_Gyroscope" folder in **"Freenove_Omni_Wheel_Car_Kit_for_Raspberry_Pi_Pico\Four-Wheel\ \Sketches"** and then double-click "**Sketch_08.1_ Gyroscope.ino".**



Code

```
#include "IMU_Fusion_SYC.h"

IMU imu(Wire);

void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);// Set the serial port baud rate to 115200
```

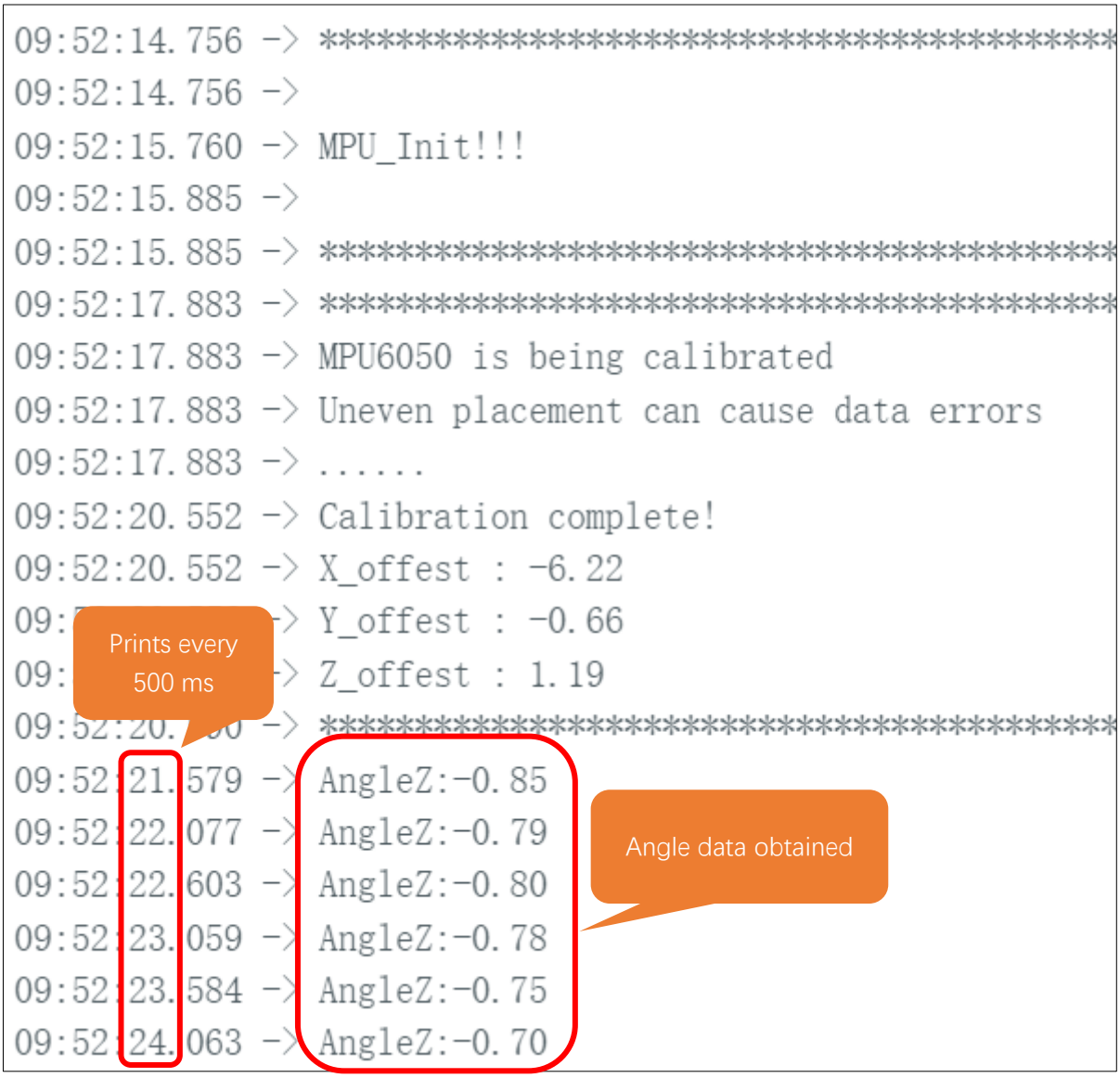
```
8     Wire.begin();
9     imu.begin(CHOOSE_MPU6050);// MPU6050 init
10    imu.MPU6050_CalcGyroOffsets();      // MPU6050 calibration
11    Serial.println("");
12  }
13
14  void loop() {
15    // put your main code here, to run repeatedly:
16    // Obtain MPU6050 data every 500ms and calculate the angles
17    imu.Calculate();// MPU6050 update raw data
18    Serial.print("AngleZ:");
19    Serial.println(imu.getAngleZ());// Print yaw angle
20    delay(500);
21  }
```

Compile and upload the sketch. Do NOT touch the circuit board once the Upload button is clicked, until the message **"Calibration complete"** is printed.

After the gyroscope initialization is complete, it takes 1-3 seconds for self-calibration. During this process, please do not touch the circuit board to ensure more accurate calibration.

```
11:30:17.819 -> ********************************************************************************
11:30:17.819 ->
11:30:18.780 -> MPU_Init!!!
11:30:18.913 ->
11:30:18.913 -> ********************************************************************************
11:30:20.923 -> ********************************************************************************
11:30:20.923 -> MPU6050 is being calibrated
11:30:20.923 -> Uneven placement can cause data errors
11:30:20.923 -> ......
11:30:23.588 -> Calibration complete!
11:30:23.588 -> X_offset : -6.20
11:30:23.623 -> Y_offset : -0.72
11:30:23.623 -> Z_offset : 1.17
11:30:23.623 -> ********************************************************************************
```

Once the initialization is complete, you will see the angle data obtained from the gyroscope printed in the serial monitor, with a period of 500 milliseconds.



Code Explanation

If you are not familiar with Arduino IDE, you can visit https://www.arduino.cc/reference/en/ to learn more.

Declare that the MPU communicates with Wire.

```
2  IMU imu(Wire);
```

Set the serial port baud rate to 115200

```
28  Serial.begin(115200);
```

I2C initialization

```
6  Wire.begin();
```

MPU6050 initialization

```
22  imu.begin(CHOOSE_MPU6050);
```

MPU6050 self-calibration.

```
28  imu.MPU6050_CalcGyroOffsets();
```

Obtain raw data of MPU6050 and calculate.

```
29  imu.Calculate();
```

Print the obtained yaw angles on serial monitor.

```
29  Serial.println(imu.getAngleZ());// Print yaw angle
```

Reference

Wire.begin();

Initialize I2C.

IMU::MPU6050_CalcGyroOffsets();

This function is part of the IMU_Fusion_SYC library. After calling this function, the gyroscope will automatically calibrate upon power-up and print the calibration information to the serial monitor.

IMU::begin();

This function is part of the IMU_Fusion_SYC library and is used to initialize the gyroscope.

Parameters:

CHOOSE_MPU6050: Initializes the gyroscope.

CHOOSE_QMC5883L: Initializes the electronic compass.

-CHOOSE_ALL: Initializes both the gyroscope and the electronic compass simultaneously.

IMU::getAngleZ();

This function is part of the IMU_Fusion_SYC library and can be used to obtain the yaw angle.

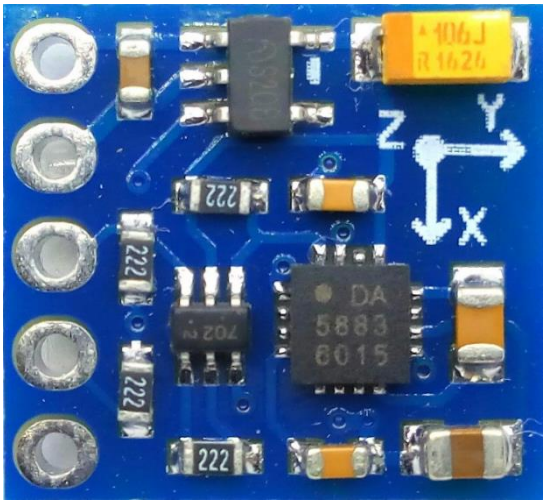
Chapter 9 Compass Test

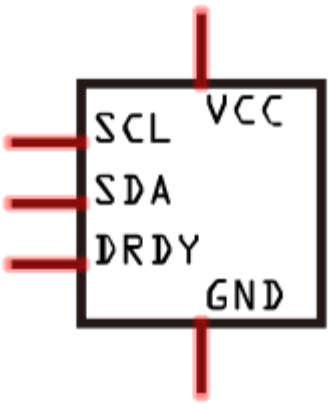
Project 9.1 Compass Calibration

Component Knowledge

A digital compass can detect the intensity and orientation of the Earth's magnetic field, offering directional insights. It measures the magnetic field strength across the X, Y, and Z axes and communicates this data to the MCU (Microcontroller Unit) via the I2C protocol.

The compass sensor is highly sensitive to magnetic fields and noise, and operating it in areas with intense magnetic fields or high noise levels can disrupt its functionality, causing inaccuracies in angle readings. To ensure precise angle measurements, it's crucial to calibrate the sensor before use.



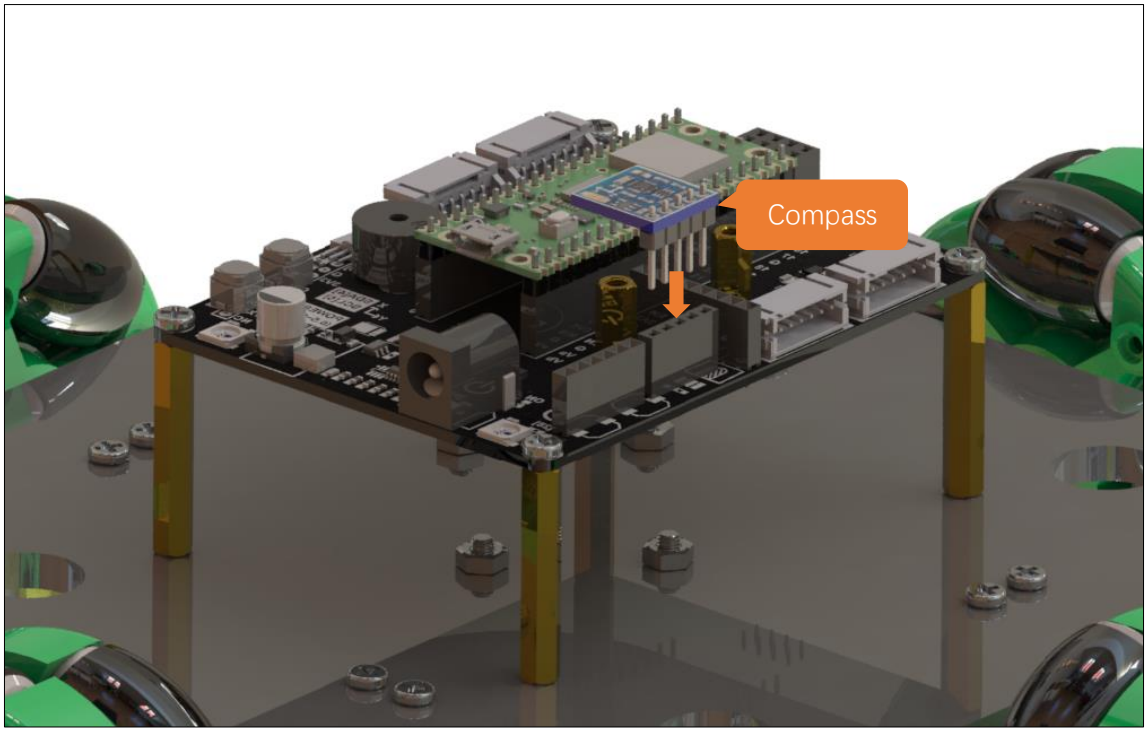


The port description of the compass module is as follows:

Pin name	Pin number	Description
VCC	1	Positive pole of power supply with voltage 5V
GND	2	Negative pole of power supply
SCL	3	I2C communication clock pin
SDA	4	I2C communication clock pin
DRDY	5	Data Ready signal, used to inform the sensor that new data are ready to be read.

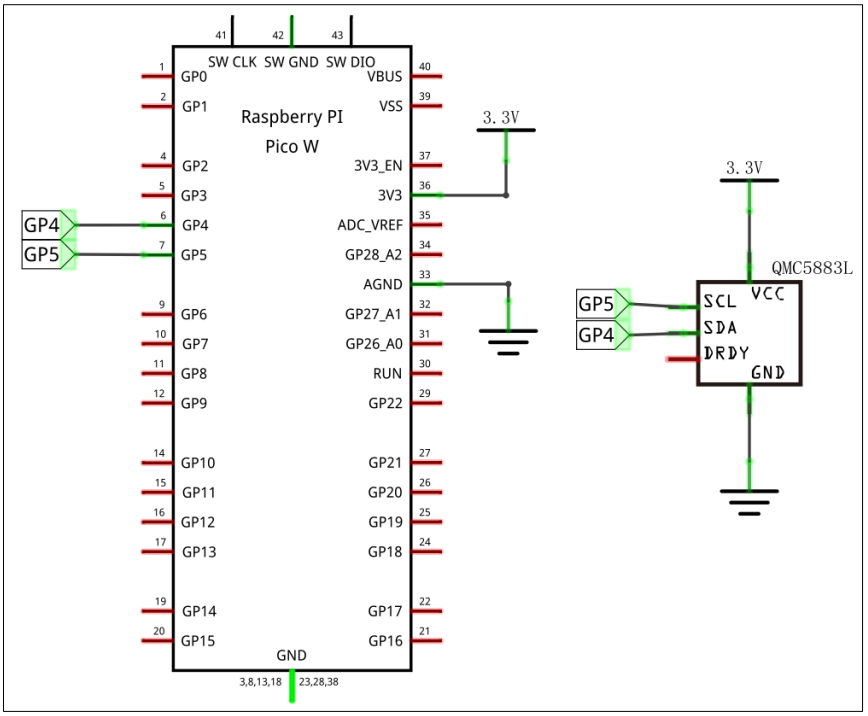
Circuit

Plug the compass sensor to the expansion board to have it connected to the Raspberry Pi Pico (W).



Schematic

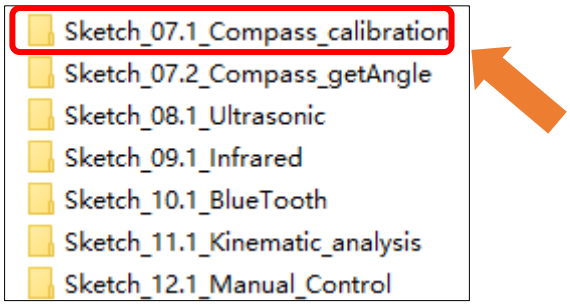
As shown in the schematic below, the SDA and SCL of the sensor are connected to GP4 and GP5 pf the Raspberry Pi Pico (W) respectively.



Sketch

In this section, we continued to use the IMI_Fusion_SYC library previously installed. Please make sure you have installed it before upload the sketch.

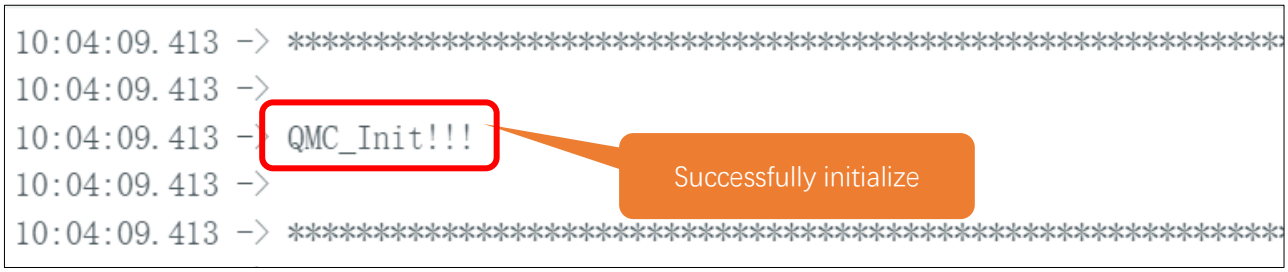
Now, let's calibrate the compass. Open "**Sketch_07.1_Compass_calibration**" folder in **"Freenove_Omni_Wheel_Car_Kit_for_Raspberry_Pi_Pico\Four-Wheel\ \Sketches"** and then double-click "**Sketch_07.1_Compass_calibration.ino**".



Code

```
#include "IMU_Fusion_SYC.h"

IMU imu(Wire);

void setup() {
    Serial.begin(9600);
    Wire.begin();
    imu.begin(CHOOSE_QMC5883L); // Select QMC5883L
}
void loop() {
    imu.QMC5883L_Calibration();
}
```

After downloading the code, open serial monitor and you can see the message "**QMC_init!!!**".



If the sensor fails to initialize, you will see the following message. In this case, please make sure the sensor is correctly connected.

```
10:06:32.385 -> ********************************************************************************
10:06:32.385 ->
10:06:32.385 -> QMC_Init false!!!        Please check whether the connection is correct
10:06:32.385 ->
10:06:32.385 -> ********************************************************************************
```

Initialization failure

Upon finishing initialization, it will start calibrating the compass. When progress bar starts, rotate the sensor at a constant speed until the progress bar reaches 100%.

```
10:06:33.397 -> ********************************************************************************
10:06:33.397 -> QMC5883L calibration will begin in 5s
10:06:33.397 -> After 5 seconds, Please rotate QMC5883L horizontally
10:06:33.397 -> Please note that the rotation speed should be uniform
10:06:33.397 -> Not too fast or too slow, otherwise it will affect the calibration data.....
10:06:38.388 -> [=                                                  ] 2%
10:06:38.994 -> [==                                                 ] 4%
10:06:39.560 -> [===                                                ] 6%
10:06:40.178 -> [====                                               ] 8%
10:06:40.780 -> [=====                                              ] 10%
10:06:41.361 -> [======                                             ] 12%
10:06:42.001 -> [=======                                            ] 14%
10:06:42.561 -> [========                                           ] 16%
10:06:43.200 -> [=========                                          ] 18%
```

Progress bar

When the progress bar reaches 100%, the calibration data will be printed on the serial monitor. At this point, please record the data of x_offset, y_offset, x_scale, and y_scale in a timely manner.

Please note that the calibration data vary among different compass.

```
11:22:59.201 -> ********************************************************************************
11:22:59.201 -> QMC calibration complete
11:22:59.201 -> Below is the raw data for calibration
11:22:59.201 -> XMax:0.00       XMin:-1870.00
11:22:59.201 -> YMax:402.00     YMin:0.00
11:22:59.201 -> Add the following calculated data to the calibration function
11:22:59.201 -> x_offset:-935.00        y_offset:201.00
11:22:59.201 -> x_scale:1.00    y_scale:0.21
11:22:59.201 -> ********************************************************************************
```

Record the data

Code Explanation

If you are not familiar with Arduino IDE, you can visit https://www.arduino.cc/reference/en/ to learn more.

Declare an object named imu.

```
3  IMU imu(Wire);
```

Activate the serial port and set the baud rate to 9600.

```
6  Serial.begin(9600);
```

Wire initialization.

```
24  Wire.begin();
```

Compass initialization.

```
22  imu.begin(CHOOSE_QMC5883L);
```

Compass calibration.

```
29  imu.QMC5883L_Calibration();
```

If you need any support, please feel free to contact us via: **support@freenove.com**

Project 9.2 Yaw Angle Reading

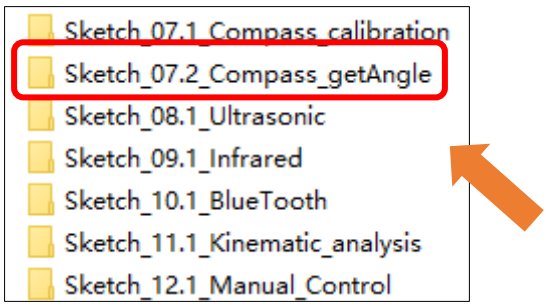
In the previous section, we introduce how to calibrate the compass. In this section, we will further tell us how to use the compass to read yaw angles.

Sketch

In this section, we continued to use the IMI_Fusion_SYC library previously installed. Please make sure you have installed it before upload the sketch.

Before uploading the sketch to read the yaw angle, please make sure you have calibrated the compass and recored the calibration data.

Open "**Sketch_07.2_Compass_getAngle**" folder in **"Freenove_Omni_Wheel_Car_Kit_for_Raspberry_Pi_Pico\Four-Wheel\ \Sketches"** and then double-click "**Sketch_07.2_Compass _getAngle.ino**".



Code

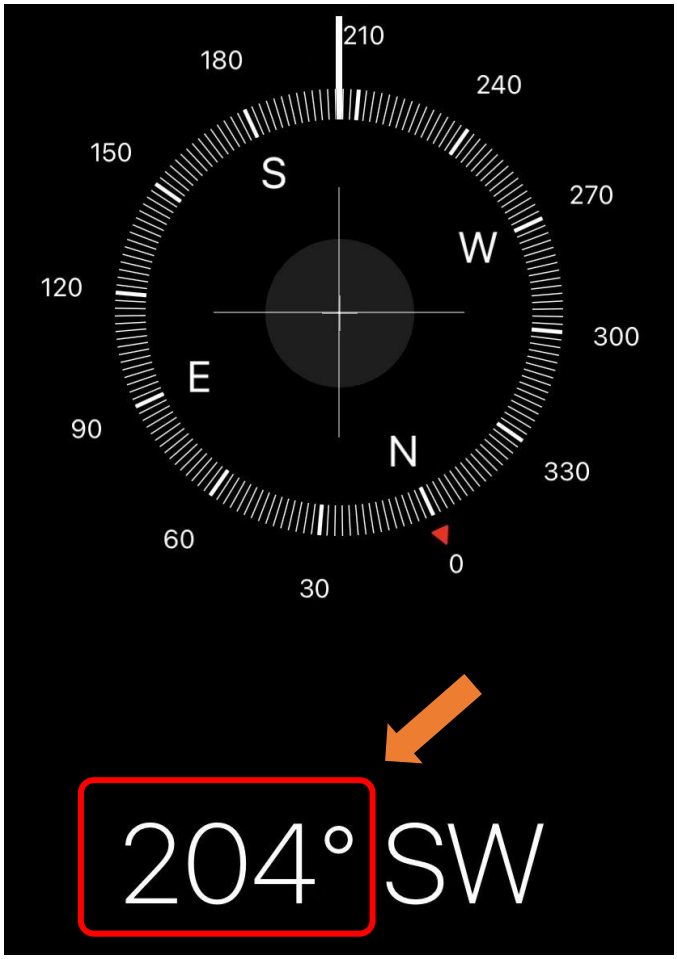
Input the **calibration data** to the code marked below, otherwise the reading angle will be incorrect.

```
#include "IMU_Fusion_SYC.h"

IMU imu(Wire);

void setup() {
  // put your setup code here, to run once:
   Serial.begin(9600);
   Wire.begin();
   imu.QMC5883L_SetOffset(x_offset,y_offset,0);
   imu.QMC5883L_SetScale(x_scale,x_scale,0);
   imu.Heading_Offset(360);       // Set Heading offset
   imu.begin(CHOOSE_QMC5883L);    // Only QMC5883L is initialized
}

void loop(){
  imu.Calculate();
  Serial.print("heading:");
```

```
18      Serial.println(imu.getHeading());
    }
```

After downloading the code, the current yaw angle will be printed on the serial monitor. You can verify whether it is correct by comparing with the value of the Compass software on your phone.

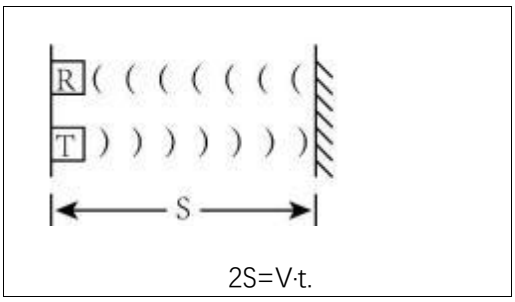
```
10:26:31.481 -> ************************************
10:26:31.481 ->
10:26:31.481 -> QMC_Init!!!
10:26:31.481 ->
10:26:31.481 -> ************************************
10:26:32.456 -> heading:203
10:26:32.487 -> heading:203
10:26:32.487 -> heading:203
10:26:32.487 -> heading:203
10:26:32.487 -> heading:203
10:26:32.487 -> heading:203
10:26:32.487 -> heading:203
```



Chapter 10 Ultrasonic Sensor Test

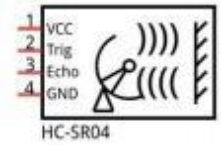
Component Knowledge

The Ultrasonic Ranging Module uses the principle that ultrasonic waves will reflect when they encounter any obstacles. This is possible by counting the time interval between when the ultrasonic wave is transmitted to when the ultrasonic wave reflects back after encountering an obstacle. Time interval counting will end after an ultrasonic wave is received, and the time difference (delta) is the total time of the ultrasonic wave's journey from being transmitted to being received. Because the speed of sound in air is a constant, and is about v=340m/s, we can calculate the distance between the Ultrasonic Ranging Module and the obstacle: s=vt/2.



2S=V·t.

The HC-SR04 Ultrasonic Ranging Module integrates both an ultrasonic transmitter and a receiver. The transmitter is used to convert electrical signals (electrical energy) into high frequency (beyond human hearing) sound waves (mechanical energy) and the function of the receiver is opposite of this. The picture and the diagram of the HC SR04 Ultrasonic Ranging Module are shown below:



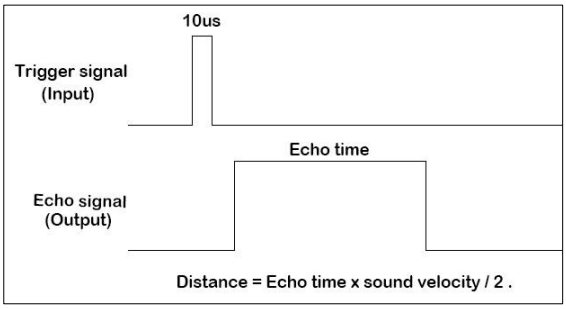
Pin description:

Pin	Description
VCC	power supply pin
Trig	trigger pin
Echo	Echo pin
GND	GND

Working voltage: 5V Working current: 12mA

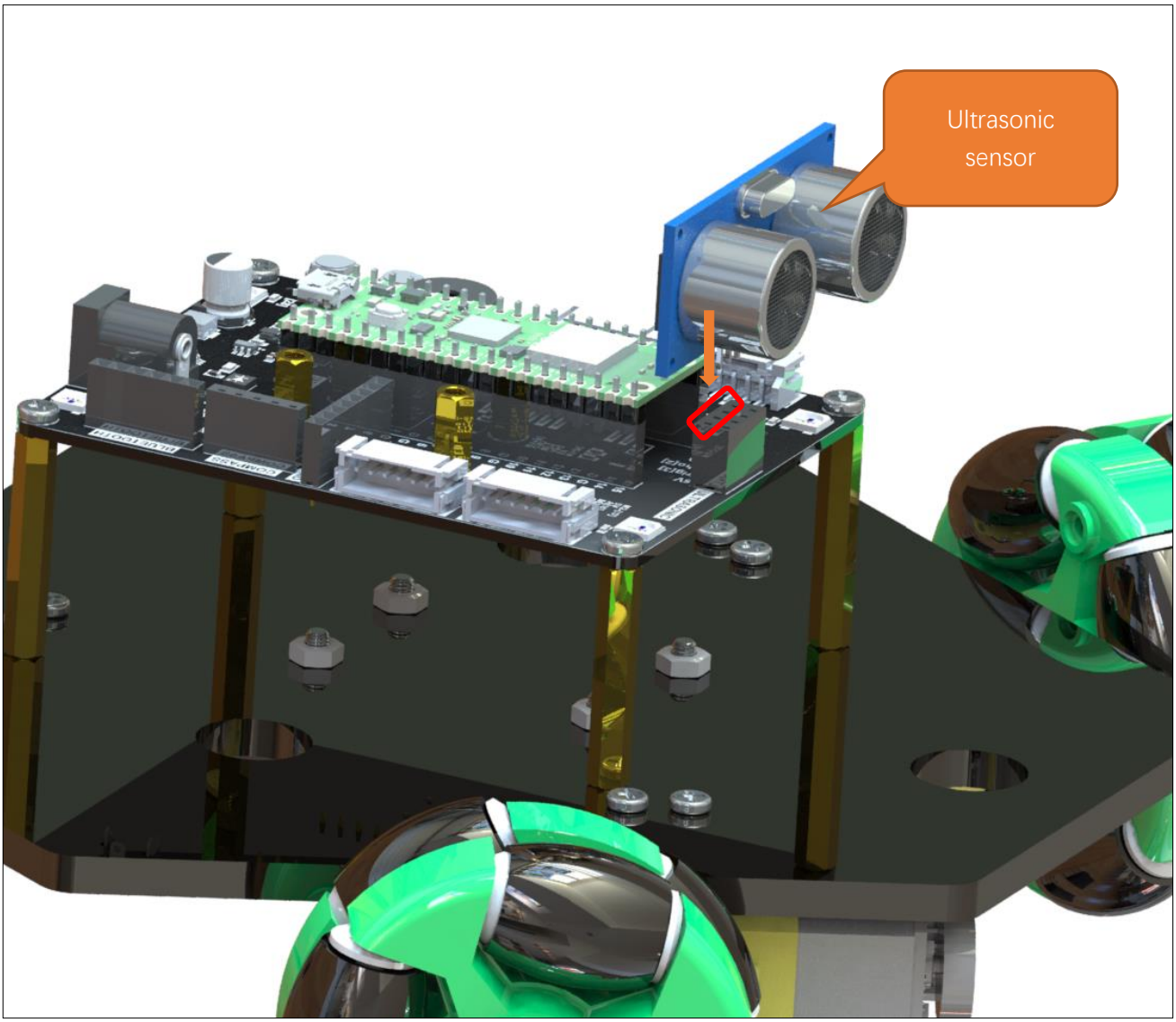
Minimum measured distance: 2cm Maximum measured distance: 200cm

Instructions for Use: output a high-level pulse in Trig pin lasting for least 10us, the module begins to transmit ultrasonic waves. At the same time, the Echo pin is pulled up. When the module receives the returned ultrasonic waves from encountering an obstacle, the Echo pin will be pulled down. The duration of high level in the Echo pin is the total time of the ultrasonic wave from transmitting to receiving, s=vt/2.



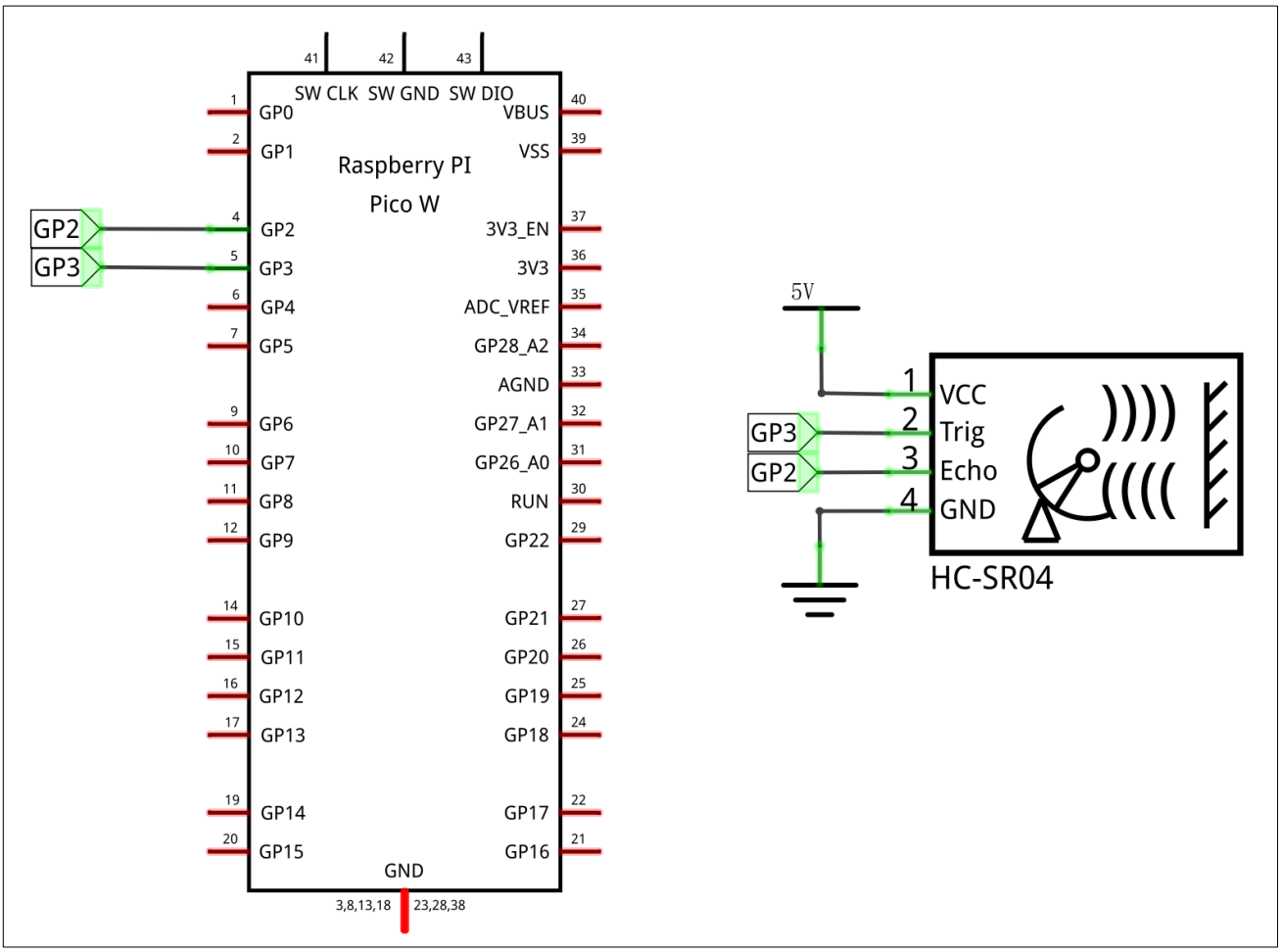
Circuit

Correctly plug the ultrasonic sensor to the front of the car.



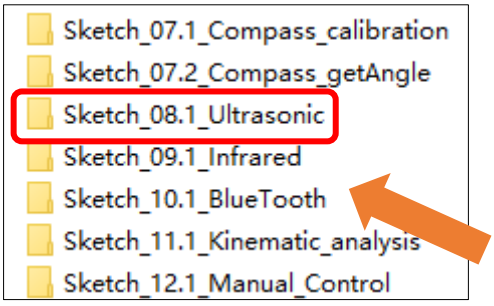
Schematic

As shown in the schematic below, the Echo and Trig of the ultrasonic sensor respectively connect to GP2 and GP3 of the pico (W).



Sketch

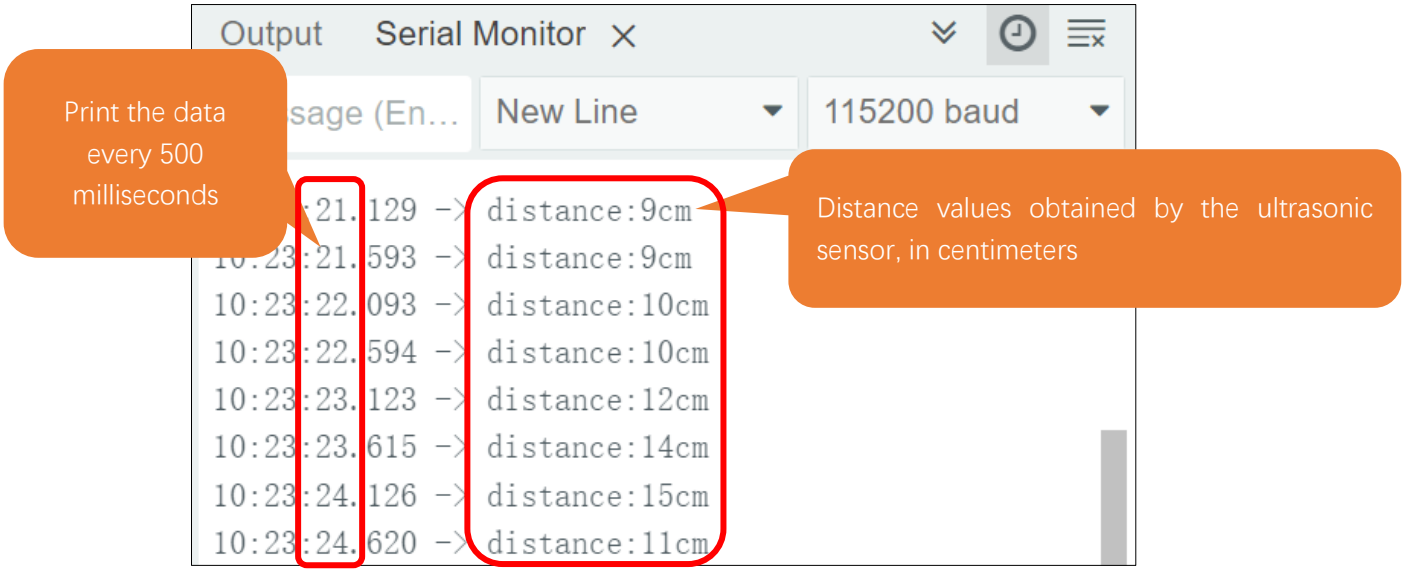
Open "Sketch_08.1_Ultrasonic" folder in **"Freenove_Omni_Wheel_Car_Kit_for_Raspberry_Pi_Pico\Four-Wheel\ \Sketches"** and then double-click **"Sketch_08.1_Ultrasonic.ino".**



Code

```
const int Trig = 3;// Define the Trig pin number of the Ultrasonic module
const int Echo = 2;// Define the Echo pin number of the Ultrasonic module

int Ultrasonic_distance;

void Ultrasonic_init()
{
   // Enable the Ultrasonic module Pin
   pinMode(Trig,OUTPUT);
   pinMode(Echo,INPUT);
}

void Read_Distance()
{
   digitalWrite(Trig,LOW);
   delayMicroseconds(2);
   digitalWrite(Trig,HIGH);
   delayMicroseconds(10);
   digitalWrite(Trig,LOW);

   Ultrasonic_distance = pulseIn(Echo,HIGH) / 58.8;// Calculated distance
}

void setup() {
   // put your setup code here, to run once:
   Serial.begin(115200);// Set the serial port baud rate to 115200
   Ultrasonic_init();
}

void loop() {
   // put your main code here, to run repeatedly:
   Read_Distance();
   Serial.print("distance:");
   Serial.print(Ultrasonic_distance);//The value of the print distance, in centimeters (cm)
   Serial.println("cm");
   delay(500);// Delay 0.5 second
}
```

After downloading the code, the pico prints the distance value every 500 milliseconds on the serial monitor.



Code Explanation

Define the pins of ultrasonic module.

```
1  const int Trig = 3;
2  const int Echo = 2;
```

Set the Trig pin to output and Echo to input.

```
8  pinMode(Trig,OUTPUT);
9  pinMode(Echo,INPUT);
```

Activate the serial port and set the baud rate to 115200.

```
7  Serial.begin(115200);
```

Make the Trig pin output high and last for 10 microseconds, so as to trigger the sensor.

```
10  digitalWrite(Trig,HIGH);
11  delayMicroseconds(10);
12  digitalWrite(Trig,LOW);
```

Obtain the duration of High level of the Echo pin and calculate the distance.

```
10  Ultrasonic_distance = pulseIn(Echo,HIGH) / 58.8;
```

Reference

```
long pulseIn(uint8_t pin, uint8_t value, unsigned long timeout = 1000000);
```

pulseIn returns the duration of the signal on a digital pin, measured in microseconds.

Parameters

pin: The pin to read.

Value: The type of pulse to measure.

HIGH: Measure the duration of the high signal.

LOW: Measure the duration of the low signal.

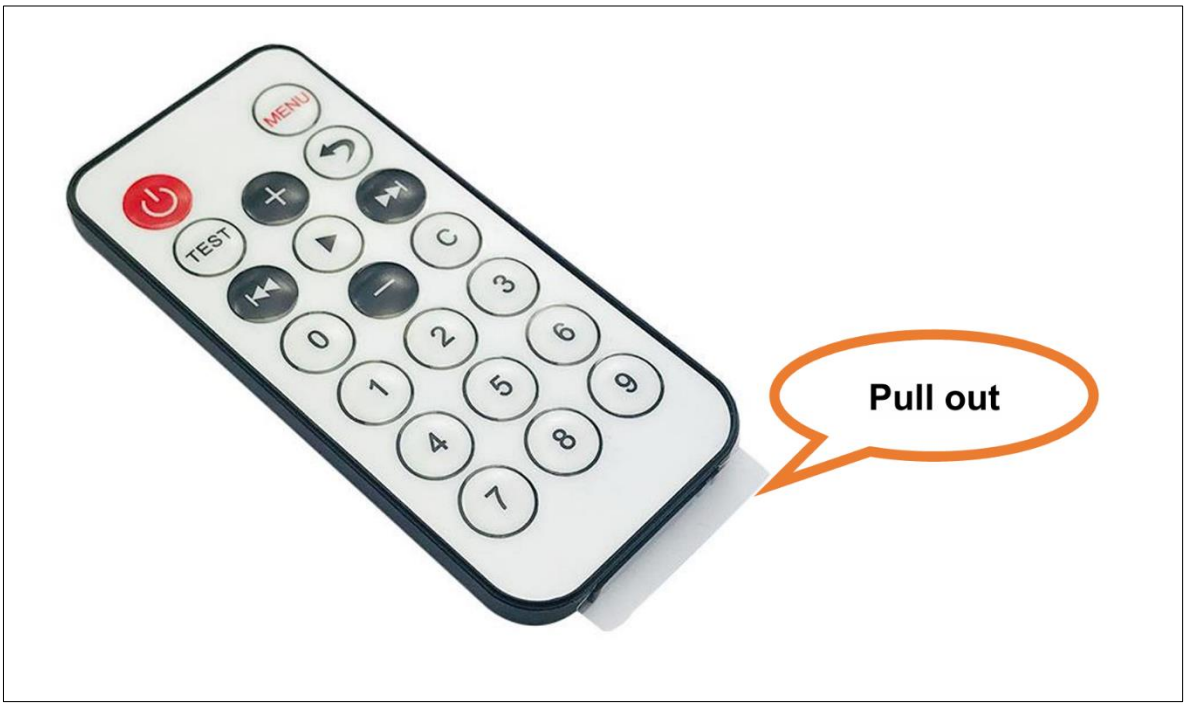
timeout: (optional) The maximum time to wait for the signal to change, default is 1 second

Chapter 11 Infrared Test

Component Knowledge

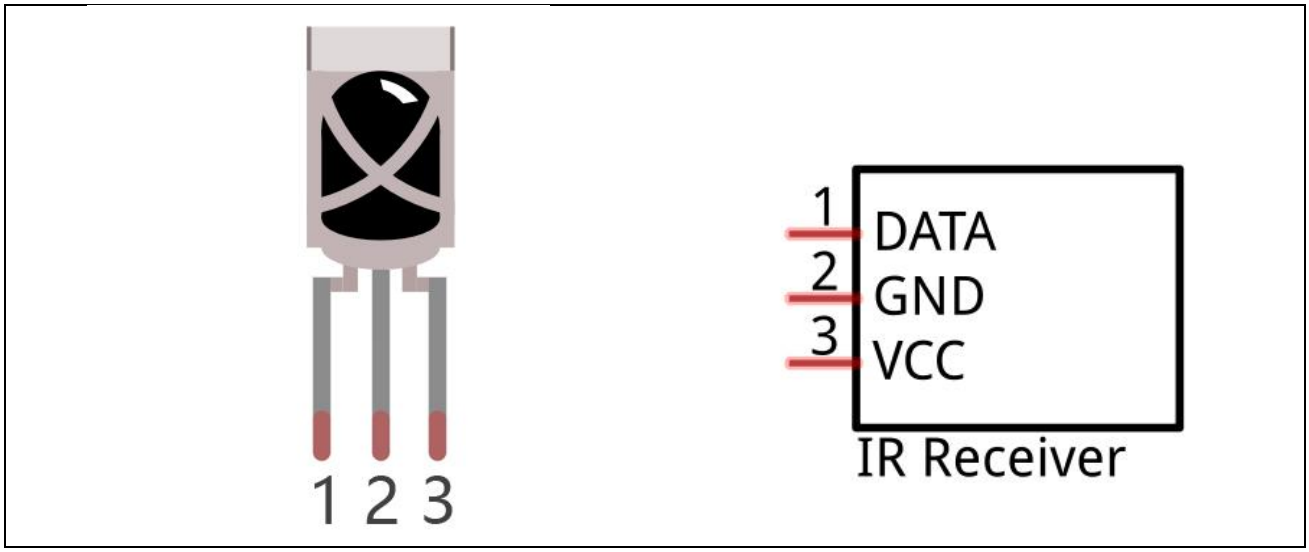
Infrared remote

An Infrared (IR) remote control is a device with a certain number of buttons. Pressing down different buttons will make the infrared emission tube, which is located in the front of the remote control, send infrared with different encoding. Infrared remote control technology is widely used in household appliances, such as TV, air conditioning, etc. Thus, it makes it possible for you to switch TV programs and adjust the temperature of the air conditioning away from them. The remote control we use is shown below:



Infrared receiver

Infrared (IR) receiver is a component which can receive the infrared light, so we can use it to detect the signal emitted by the infrared remote control. DATA pin here outputs the received infrared signal.

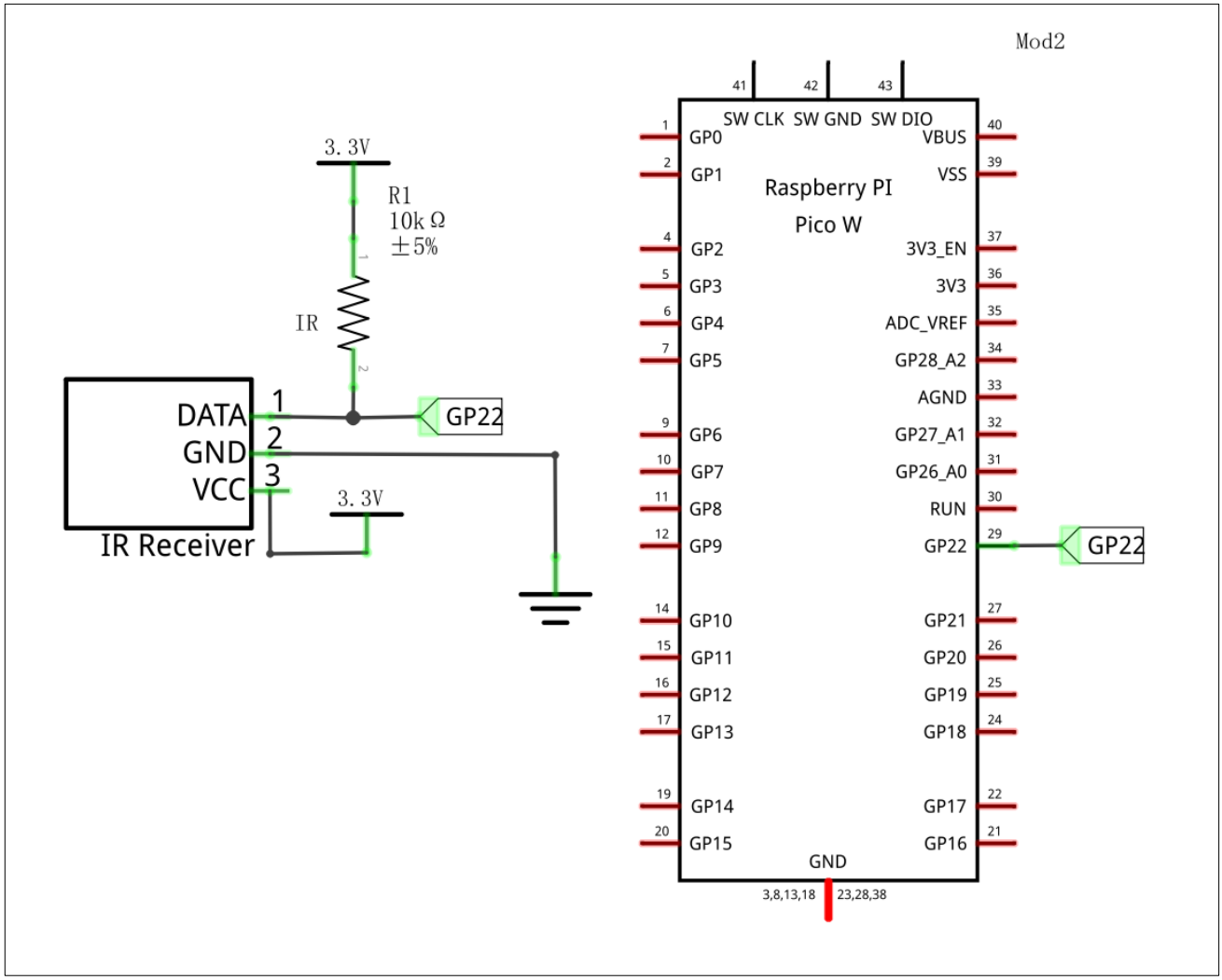


When you use the infrared remote control, it sends a key value to the receiving circuit according to the pressed key. The following is the key value that the receiving circuit will receive when each key of the infrared remote control is pressed.

ICON	KEY Value	ICON	KEY Value
	FFA25D	C	FFB04F
MENU	FFE21D	1	FF30CF
TEST	FF22DD	2	FF18E7
	FF02FD	3	FF7A85
	FFC23D	4	FF10EF
	FFE01F	5	FF38C7
	FFA857	6	FF5AA5
	FF906F	7	FF42BD
0	FF6897	8	FF4AB5
	FF9867	9	FF52AD

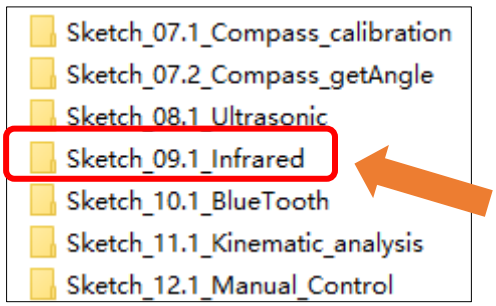
Schematic

As shown in the schematic, GP22 of the pico connects to the infrared receiver, which means we can access the infrared signal via GP22.



Sketch

Next, we download the code to Raspberry Pi Pico (W) to test the infrared receiver. Open **"Sketch_09.1_Infrared"** folder under **"Freenove_Omni_Wheel_Car_Kit_for_Raspberry_Pi_Pico\Four-Wheel\ \Sketches"** and double-click **"Sketch_09.1_Infrared.ino"**.



Code

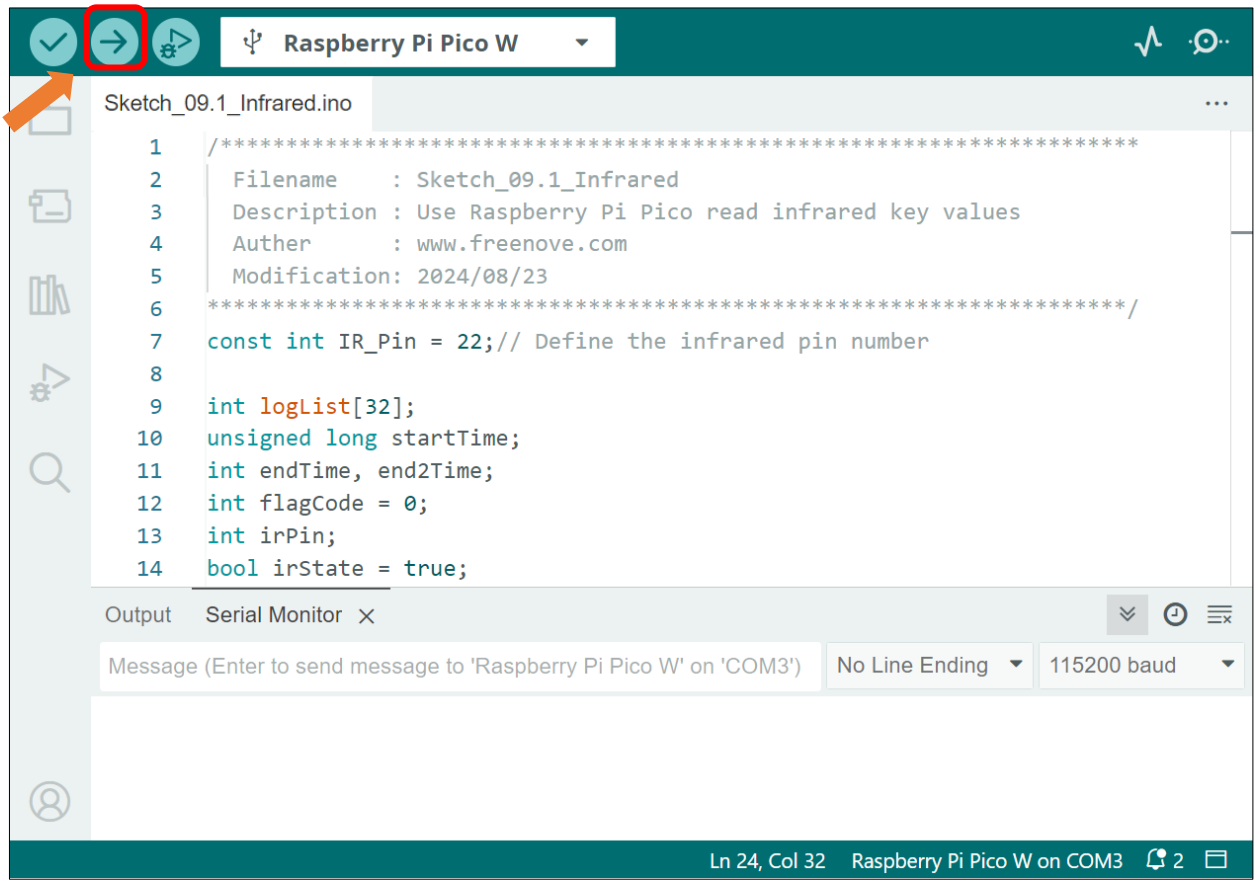
Sketch_09.1_Infrared.ino

```
const int IR_Pin = 22;// Define the infrared pin number

int logList[32];
unsigned long startTime;
int endTime, end2Time;
int flagCode = 0;
int irPin;
bool irState = true;

void setup() {
  Serial.begin(115200);// Set the serial port baud rate to 115200
  IR_Init(IR_Pin);// Infrared pin initialization
}

void loop() {
  if(flagCode){
    int irValue = IR_Decode(flagCode);// Read infrared key values
    Serial.println(irValue, HEX);// Print infrared key values
    IR_Release();
  }
}

void IR_Init(int pin) {
  irPin = pin;
  pinMode(irPin, INPUT_PULLUP);// Enable infrared pin as input pull-up
  attachInterrupt(digitalPinToInterrupt(irPin), IR_Read, CHANGE);// Start infrared interrupt
```

```
}
void IR_Read() {
  if (irState == true) {
    unsigned long lowTime, highTime, intervalTime;
    int num = 0;
    while (digitalRead(irPin) == LOW) {
      startTime = micros();
      while (digitalRead(irPin) == LOW) {
        lowTime = micros();
      }
      intervalTime = lowTime - startTime;
      while (digitalRead(irPin) == HIGH) {
        highTime = micros();
        intervalTime = highTime - lowTime;
        if (intervalTime > 10000) {
          end2Time = millis();
        if (num == 32) {
           flagCode = 1;
         endTime = millis();
          }
         else if (num == 0 && end2Time - endTime > 300 && end2Time - endTime < 400) {
             flagCode = 2;
             endTime = millis();
        }
          return;
        }
      }
      if (intervalTime < 2000) {
        if (intervalTime < 700) {
         logList[num ++] = 0;
        }
        else {
          logList[num ++] = 1;
        }
      }
    }
  }
}
unsigned long IR_Decode(int &code) {
  unsigned long irData = 0;
  irState=false;
  if (code == 1) {
    code = 0;
    for (int i = 0; i < 32; i ++) {
```

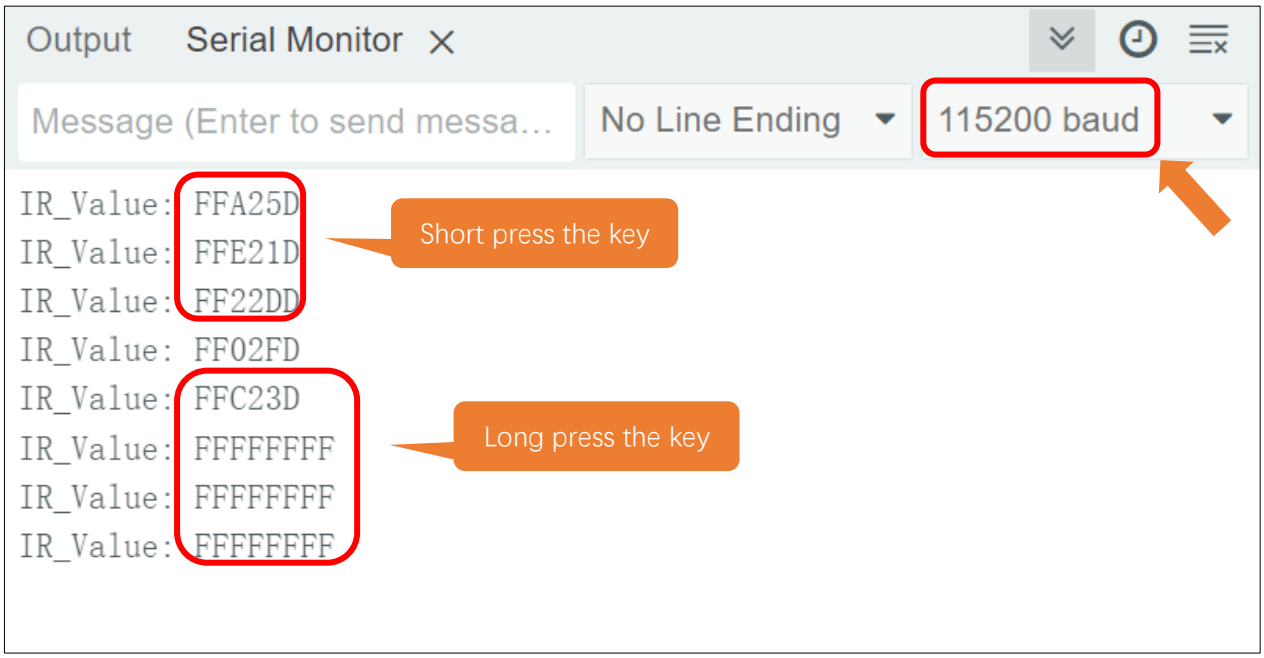
```
      if (logList[i] == 0) {
        irData <<= 1;
      }
      else {
        irData <<= 1;
        irData ++;
      }
      logList[i] = 0;
    }
  }
  if (code == 2) {
    code = 0;
    irData = 0xffffffff;
  }
  return irData;
}
void IR_Release(){
  irState=true;
}
```

This sketch uses the infrared receiving tube to receive the value sent form the infrared remote control, and print it out via the serial port. Click "**Upload**" to download the code.



After downloading the code, open the serial port monitor, set the baud rate to **115200**, press the IR remote

control, the pressed keys value will be printed out through the serial port.



Code Explanation

When the IR_Init() function is called, Pico initializes the infrared received pin (GP22) and sets the external interrupt, associating it with the IR_Read() function. Every time the infrared receives data, external interrupt calls IR_Read() function to receive data.

```
1  void IR_Init(int pin) {
2    irPin = pin;
3    pinMode(irPin, INPUT_PULLUP);// Enable infrared pin as input pull-up
4    attachInterrupt(digitalPinToInterrupt(irPin), IR_Read, CHANGE);// Start infrared interrupt
5  }
```

Assign the read infrared key value to the variable irValue.

```
1  int irValue = IR_Decode(flagCode);// Read infrared key values
```

Print the variable irValue in HEX format.

```
1  Serial.println(irValue, HEX);// Print infrared key values
```

Allow to receive new infrared key values again.

```
1  IR_Release();
```

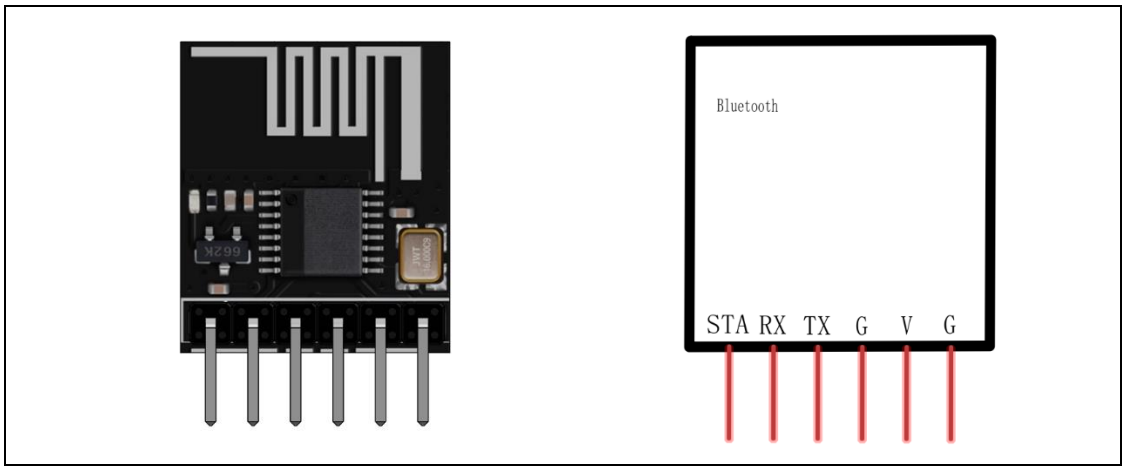
Chapter 12 Bluetooth Test

In earlier lessons, we explored wired serial communication. Yet, this approach is tethered to physical data cables, significantly constraining its potential use cases. Now, let's delve into Bluetooth modules, which facilitate wireless communication. We'll utilize a Bluetooth module to fetch commands from the Freenove APP and display them on the serial monitor.

Component Knowledge

Bluetooth

Bluetooth modules are devices designed for wireless communication, capable of transmitting data over short distances with the aim of facilitating low-power data transfer between devices. They rely on radio frequency signals, predominantly functioning in the 2.4 GHz band. This frequency spectrum is internationally recognized, ensuring compatibility and connectivity across various geographical locations.

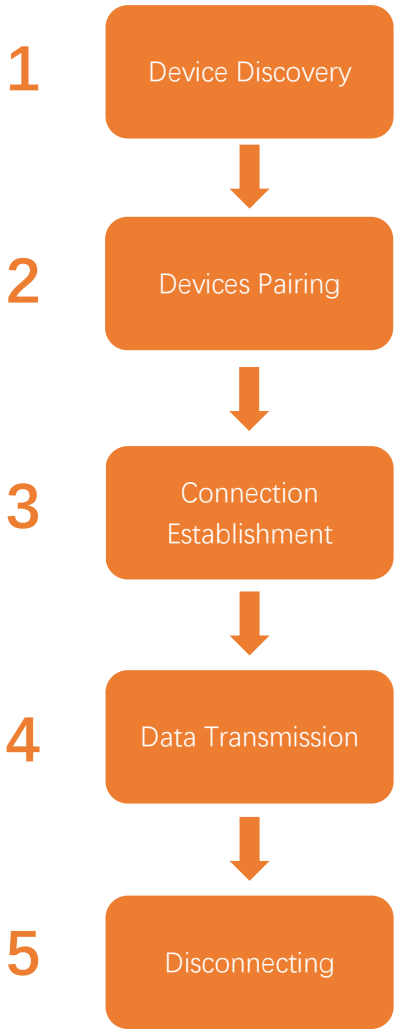


The port description of the Bluetooth module is as follows:

Pin name	Description
STA	The pin indicates the connection status.
RX	The pin receives Bluetooth data.
TX	The pin sends Bluetooth data.
VCC(V)	Positive pole of power supply with voltage 3.3V
GND(G)	Negative pole of power supply

Steps of Bluetooth Communication

The process that the Bluetooth realize wireless communication can be simplified to the following steps:



1. Device Discovery

When the Bluetooth feature of a device is activated, it initiates a process known as "device discovery." During this phase, the device broadcasts wireless signals to inquire if there are any nearby Bluetooth-enabled devices available for connection.

2. Devices Pairing

After identifying connectable devices, users must proceed with pairing. Pairing is akin to establishing a password for the devices, ensuring that only authorized devices can establish a connection.

3. Connection Establishment

Once the pairing is successfully completed, a connection is established between the devices. The Bluetooth protocol ensures that data can be transmitted smoothly. Bluetooth utilizes a technique known as "frequency hopping spread spectrum" (FHSS), rapidly switching between multiple frequencies, which reduces interference and enhances the stability of the connection.

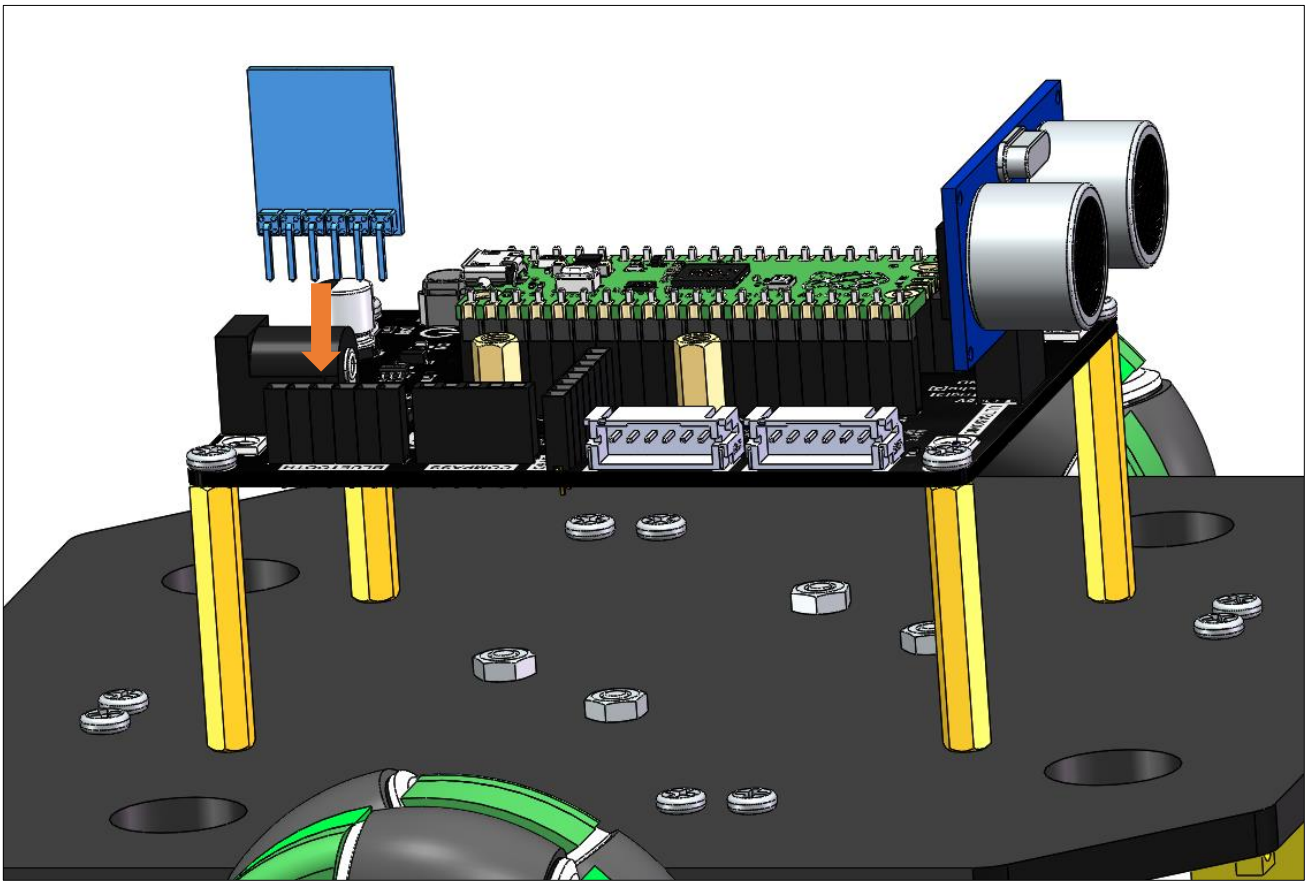
4. Data Transmission

After the connection is established, devices can commence data exchange. Data is divided into packets and sent via wireless signals. The Bluetooth protocol manages these packets, ensuring they arrive at the receiving device intact and error-free.

5. Disconnecting
   Once the data transmission is complete, devices can opt to disconnect. They will revert to the pairing mode, awaiting new connection requests.

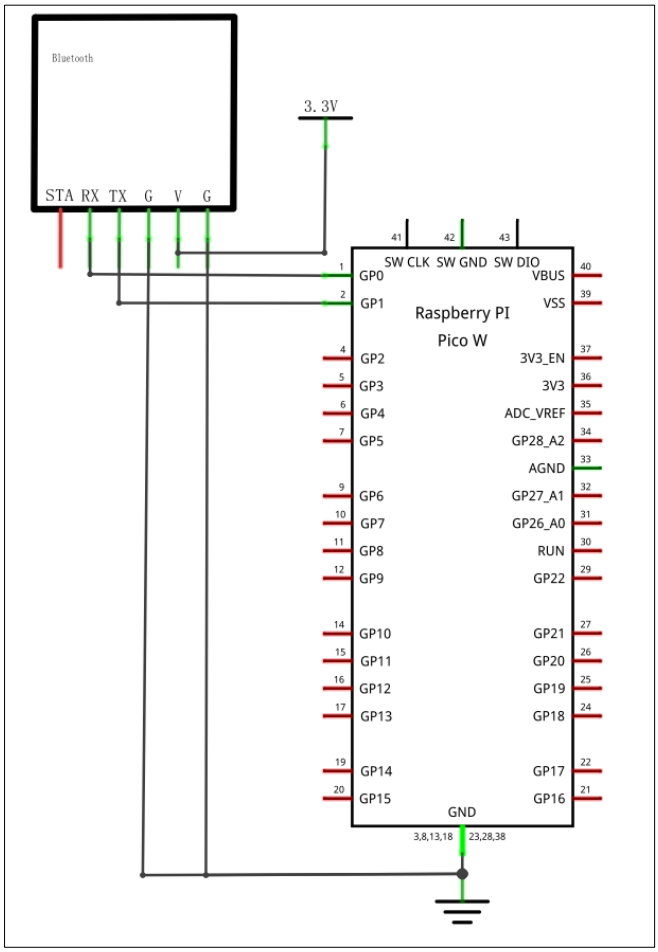
Circuit

Correctly plug the Bluetooth module to the car.



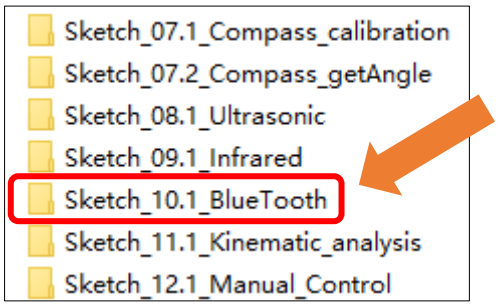
Schematic

As shown in the schematic below, the RXD and TXD pins of the Bluetooth module connects Pico (W)'s GP0 and GP1.



Sketch

Open "Sketch_10.1_Bluetooth" folder in **"Freenove_Omni_Wheel_Car_Kit_for_Raspberry_Pi_Pico\Four-Wheel\ \Sketches"** and then double-click **"Sketch_10.1_Bluetooth.ino".**



Code

```
1  #include <Wire.h>
```

```
#include "SoftwareSerial.h"

SoftwareSerial bluetooth(1, 0);

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  bluetooth.begin(115200);
}

void loop() {
  // put your main code here, to run repeatedly:
  if (bluetooth.available() > 0) {
    Serial.print("Data:");
    Serial.println(bluetooth.readString());
  }
}
```

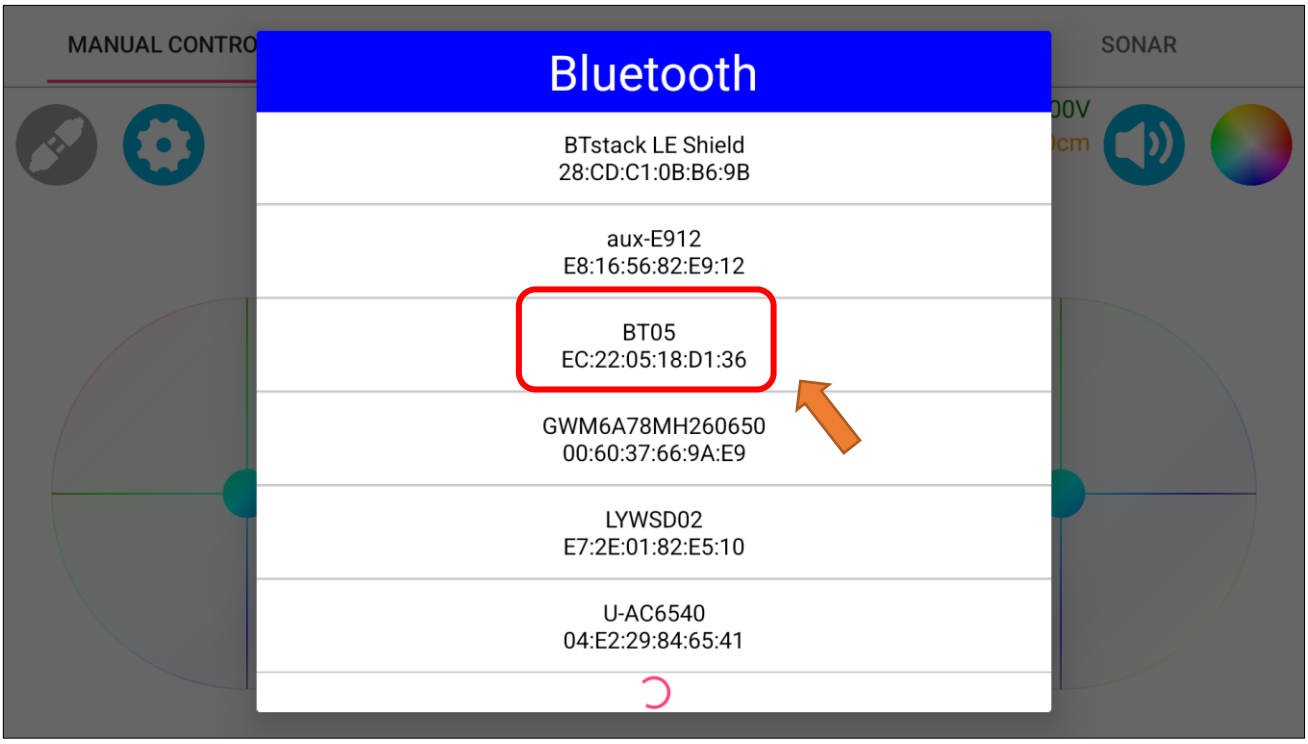
Connection

Open Freenove APP and tap the omniwheel Car Kit.

If you have not installed the Freenove APP to your phone, please refer to Freenove app.

Step 1: Click the connection icon on the left.

Step 2: Select the device named BT05. At this point, the blue indicator on the Bluetooth module blinks every one second, indicating it is waiting for pairing.



Step 3: When the connection icon turn blue and the indicator on the Bluetooth module blink rapidly, it indicates that the devices pair successfully.

Chapter 13 Kinematic Analysis

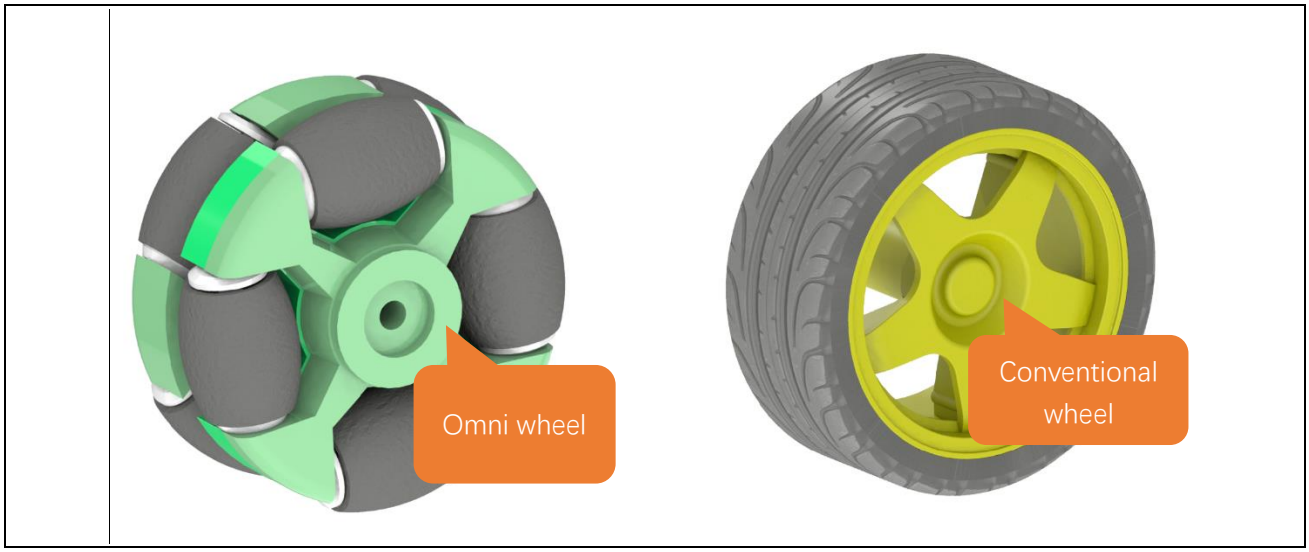
In this chapter, we will get to know the speciality of omni wheels and how an omni-wheeled car can move in any direction.

Related Knowledge

Omni Wheels

Omni wheels are a unique type of wheel that can move freely in any direction, not just linearly forward or backward. Unlike conventional wheels, omni wheels consist of two layers, and they feature a ring of small rollers around the perimeter that can spin freely.

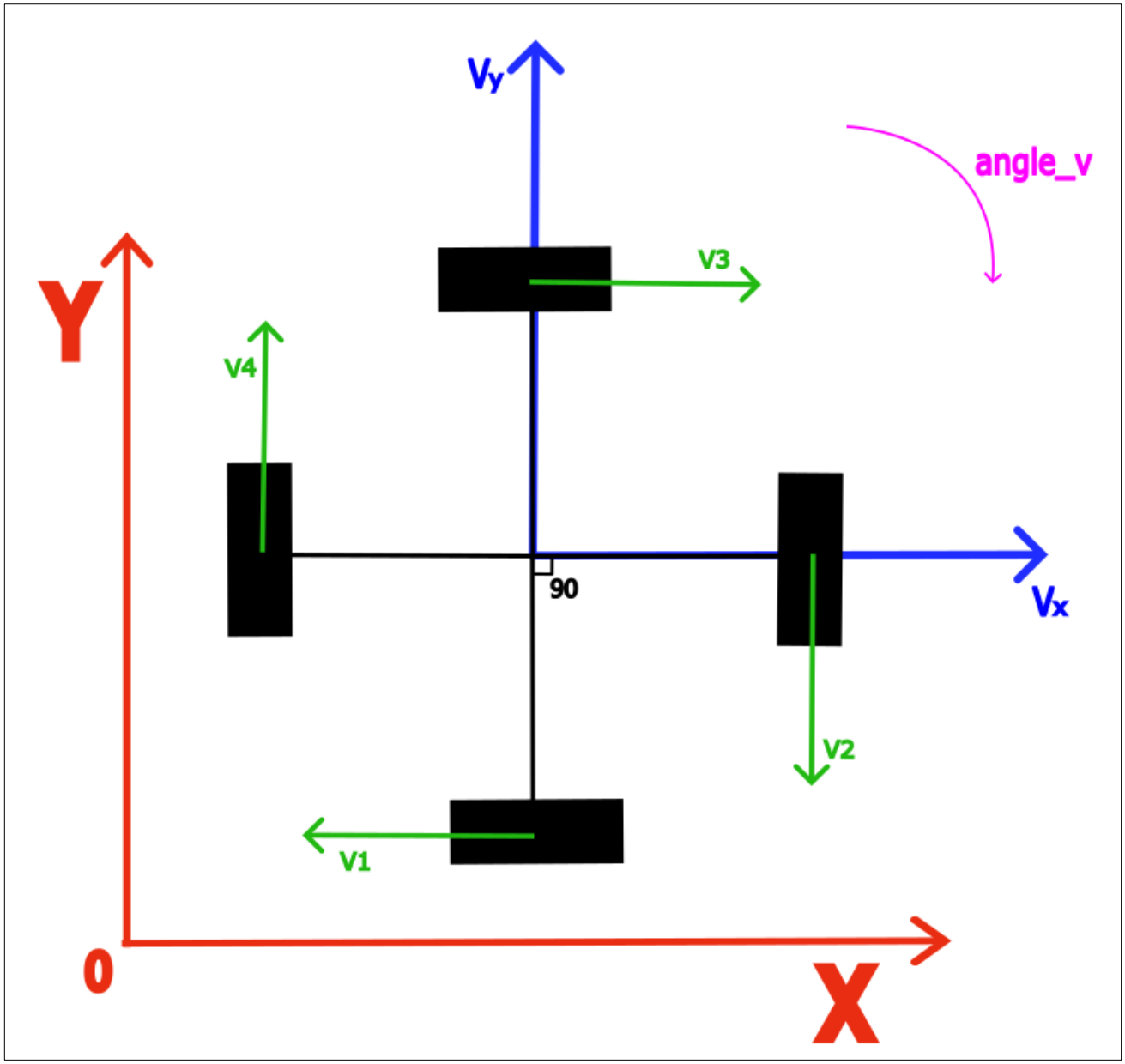
As shown in the figure below, the left one is an omni wheel and the right one is a conventional wheel.



Attitude Calculation of Omni-Wheeled Car

Omni wheels are specially designed to move freely in multiple directions, but a single or pair of these wheels alone cannot accomplish true omnidirectional movement. It is only through the coordinated action of three or more omni wheels that full omnidirectional capability is achieved. The design of an omni wheel enables it to exert force in various directions, yet a mere one or two wheels cannot provide the necessary support and control in all directions simultaneously. Each wheel of an omni wheeled robot car can independently control its rotational speed and direction, thereby enabling a variety of complex motion patterns, including straight-line movements, rotations, and diagonal motions.

The following figure shows the attitude analysis of the omni-wheeled car.



The symbols and their meaning are as described in the following table.

Symbols	Meaning
X	The X-axis of the Absolute Coordinate System
Y	The Y-axis of the Absolute Coordinate System
Vx	The Decomposition of the Car's Total Speed on the X-axis
Vy	The Decomposition of the Car's Total Speed on the Y-axis
V1	The Total Speed of Wheel 1
V2	The Total Speed of Wheel 2
V3	The Total Speed of Wheel 3
V4	The Total Speed of Wheel 4
angle_v	The Car's Angular Velocity

Let us decompose the motion of the car and compute the velocity of its four wheels individually.

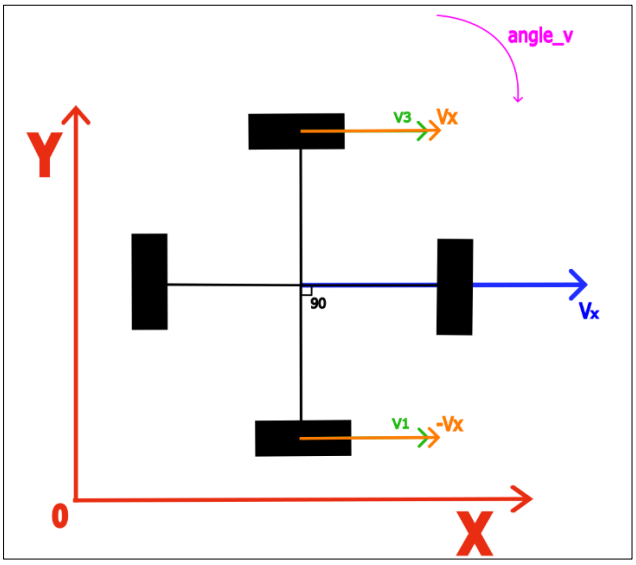
1. When the car moves purely along the X-axis, it has no velocity component along the Y-axis, as illustrated in the figure below:

   v1 = -Vx

   v2 = 0

   v3 = Vx

   v4 = 0



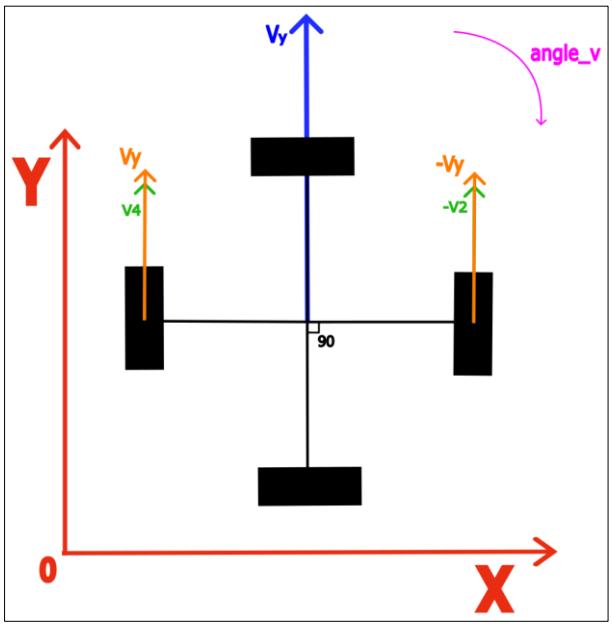
2. When the car moves purely along the Y-axis, it has no velocity component along the X-axis, as illustrated in the figure below:

   v1 = 0

   v2 = -Vy

   v3 = 0

   v4 = Vy



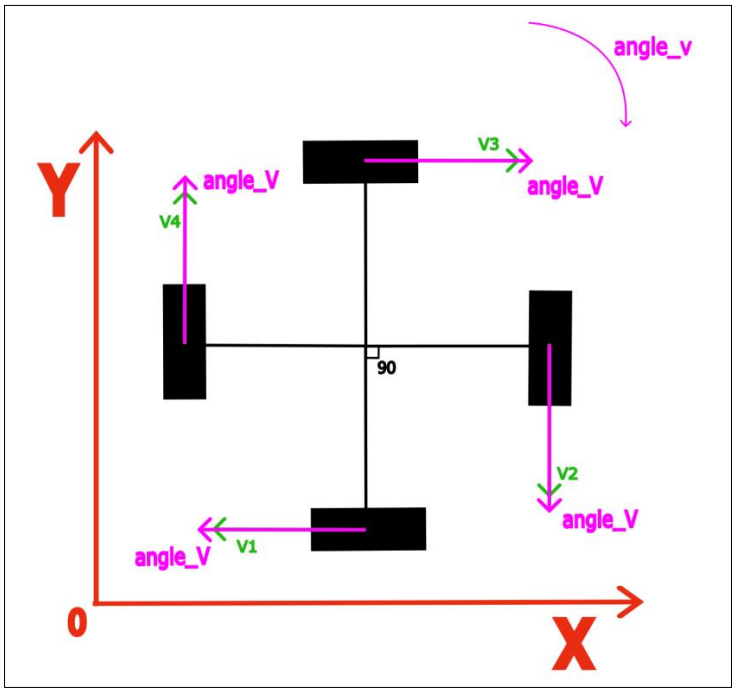
3. When the car is only performing rotational movement.

   v1 = angle_v

   v2 = angle_v

   v3 = angle_v

   v4 = angle_v



By superimposing the three situations, we can derive the motion equation for the car as follows:

   v1 = -Vx + angle_v

   v2 = -Vy + angle_v

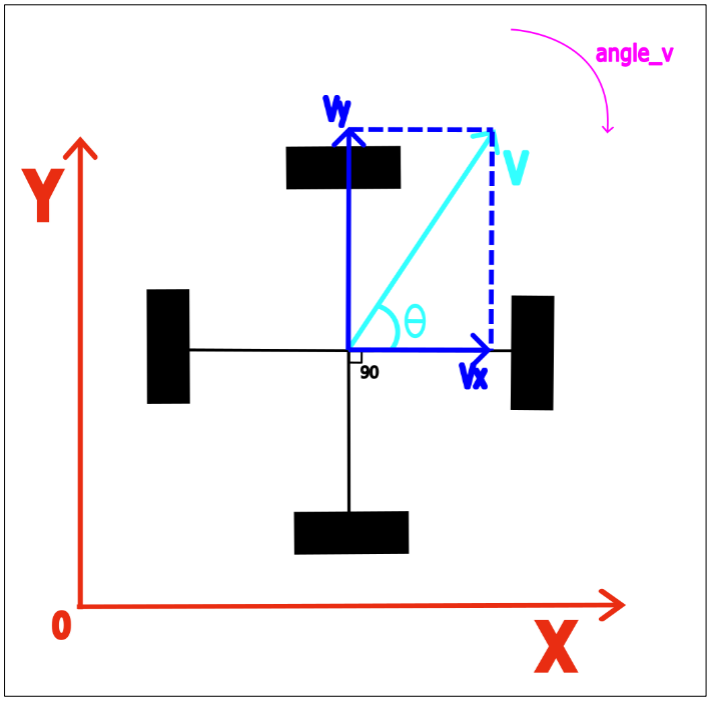
   v3 = Vx + angle_v

   v4 = Vy + angle_v

When the car moves in the direction of θ, the values of Vx and Vy can be determined by decomposing the velocity vector:

Vx = -V * sinθ

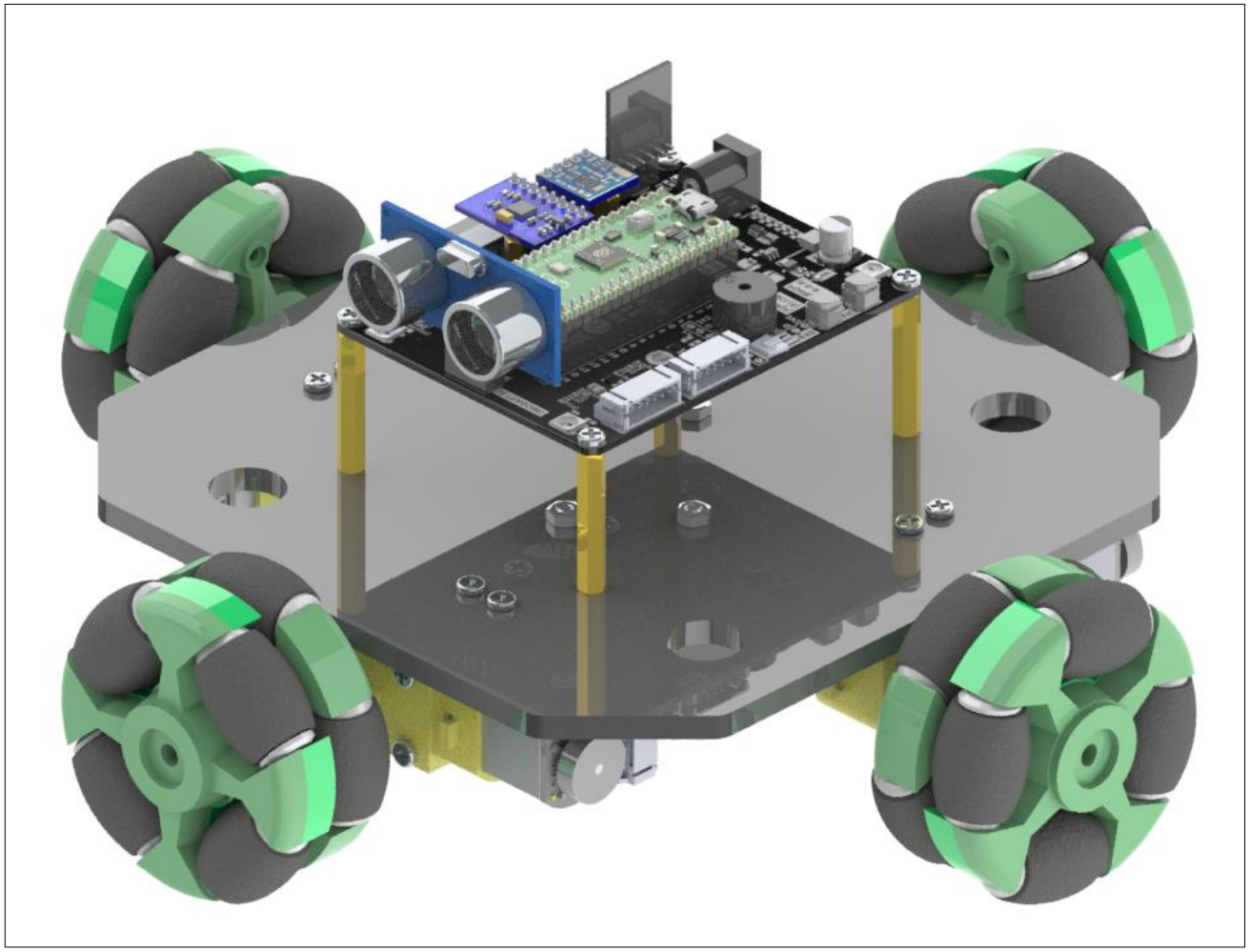
Vy = V * cosθ



Here, V represents the car's total speed, and θ is the angle of the car's translation in the XY plane, with the positive direction of the X-axis as the reference.

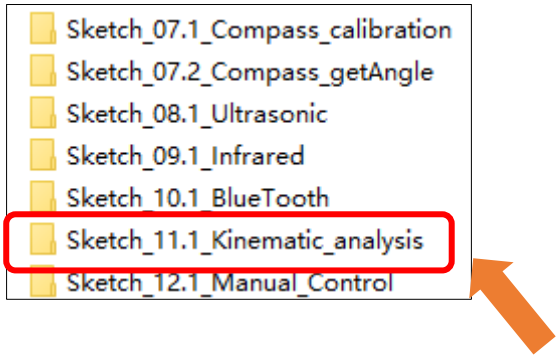
Circuit

For this chapter, we use the assembled car. Please refer to Chapter 1 for the detailed assembly process.



Sketch

Next, we download the code to Raspberry Pi Pico (W) to examine the attitude calculation. Open **"Sketch_11.1_Kinematic_analysis"** folder under **"Freenove_Omni_Wheel_Car_Kit_for_Raspberry_Pi_Pico\Four-Wheel\ \Sketches"** and double-click **"Sketch_11.1_Kinematic_analysis.ino"**.



Code

Sketch_11.1_Kinematic_analysis.ino

```
const int wheel1_A_pin = 8; // Motor1 drive pin
const int wheel1_B_pin = 9; // Motor1 drive pin

const int wheel2_A_pin = 12;// Motor2 drive pin
const int wheel2_B_pin = 13;// Motor2 drive pin

const int wheel3_A_pin = 15;// Motor3 drive pin
const int wheel3_B_pin = 14;// Motor3 drive pin

const int wheel4_A_pin = 20;// Motor4 drive pin
const int wheel4_B_pin = 21;// Motor4 drive pin

void setup() {
  // put your setup code here, to run once:
  Motor_init();
}

void loop() {
  // put your main code here, to run repeatedly:
  Motor_Control(200,0);    // Forward
  delay(1000);
  Motor_Control(-200,0);   // Back
  delay(1000);
  Motor_Control(200,90);   // Left translation
  delay(1000);
  Motor_Control(200,-90);  // Right translation
  delay(1000);
  Motor_Control(255,45);   // Shift 45 degrees to the left
  delay(1000);
  Motor_Control(255,-135); // Shift 135 degrees to the right
  delay(1000);
  Motor_Control(255,135);  // Shift 135 degrees to the left
  delay(1000);
  Motor_Control(255,-45);  // Shift 45 degrees to the right
  delay(5000);

}
void Motor_Control(int speed_v,int speed_a)
{
  int vx =  - speed_v * sin ( speed_a * PI / 180 );
```

```
    int vy =    speed_v * cos ( speed_a * PI / 180 );

    v1 =  vy;
    v2 = -vx;
    v3 = -vy;
    v4 =  vx;

    Motor_direction();//Motor limiting and motor output orientation
}

void Motor_direction()
{
    if(v1 > 0){ a1 = v1; b1 = LOW; }else{ a1 = LOW; b1 = -v1;}
    if(v2 > 0){ a2 = v2; b2 = LOW; }else{ a2 = LOW; b2 = -v2;}
    if(v3 > 0){ a3 = v3; b3 = LOW; }else{ a3 = LOW; b3 = -v3;}
    if(v4 > 0){ a4 = v4; b4 = LOW; }else{ a4 = LOW; b4 = -v4;}

    analogWrite(wheel1_A_pin,a1);
    analogWrite(wheel1_B_pin,b1);

    analogWrite(wheel2_A_pin,a2);
    analogWrite(wheel2_B_pin,b2);

    analogWrite(wheel3_A_pin,a3);
    analogWrite(wheel3_B_pin,b3);

    analogWrite(wheel4_A_pin,a4);
    analogWrite(wheel4_B_pin,b4);
}

void Motor_init()
{
    // Set the motor PWM frequency to 16 kHz
    analogWriteFreq(16000);
    // Enable the motor drive pin to output mode
    pinMode(wheel1_A_pin,OUTPUT);
    pinMode(wheel1_B_pin,OUTPUT);
    pinMode(wheel2_A_pin,OUTPUT);
    pinMode(wheel2_B_pin,OUTPUT);
    pinMode(wheel3_A_pin,OUTPUT);
    pinMode(wheel3_B_pin,OUTPUT);
    pinMode(wheel4_A_pin,OUTPUT);
    pinMode(wheel4_B_pin,OUTPUT);
}
```

After downloading the code, put the car on the floor, turn ON the power switch of the car, and you will see to car sequentially perform the actions of moving forward, moving backward, moving left, moving right, moving at a 45-degree angle to the left, moving at a 45-degree angle to the right, moving at a 135-degree angle to the left, and moving at a 135-degree angle to the right.

Code Explanation

When the Motor_init() function is called, pico enables the motors' pins to output mode.

```
void Motor_init()
{
    // Set the motor PWM frequency to 16 kHz
    analogWriteFreq(16000);
    // Enable the motor drive pin to output mode
    pinMode(wheel1_A_pin,OUTPUT);
    pinMode(wheel1_B_pin,OUTPUT);
    pinMode(wheel2_A_pin,OUTPUT);
    pinMode(wheel2_B_pin,OUTPUT);
    pinMode(wheel3_A_pin,OUTPUT);
    pinMode(wheel3_B_pin,OUTPUT);
    pinMode(wheel4_A_pin,OUTPUT);
    pinMode(wheel4_B_pin,OUTPUT);
}
```

The function Motor_Control() calculates the velocity of each wheel.

```
void Motor_Control(int speed_v,int speed_a)
{
    int vx =  - speed_v * sin ( speed_a * PI / 180 );
    int vy =    speed_v * cos ( speed_a * PI / 180 );

    v1 =  vy;
    v2 = -vx;
    v3 = -vy;
    v4 =  vx;

    Motor_direction();//Motor limiting and motor output orientation
}
```

The function Motor_Control() is called under the loop function to control the car's movement.

```
Motor_Control(255,135);   // Shift 45 degrees to the left
```

Reference

double sqrt(double x);

This function is used to calculate the square root of a number.

void Motor_Control (int speed_v,int speed_a);

The function is used to control the movement of the car.

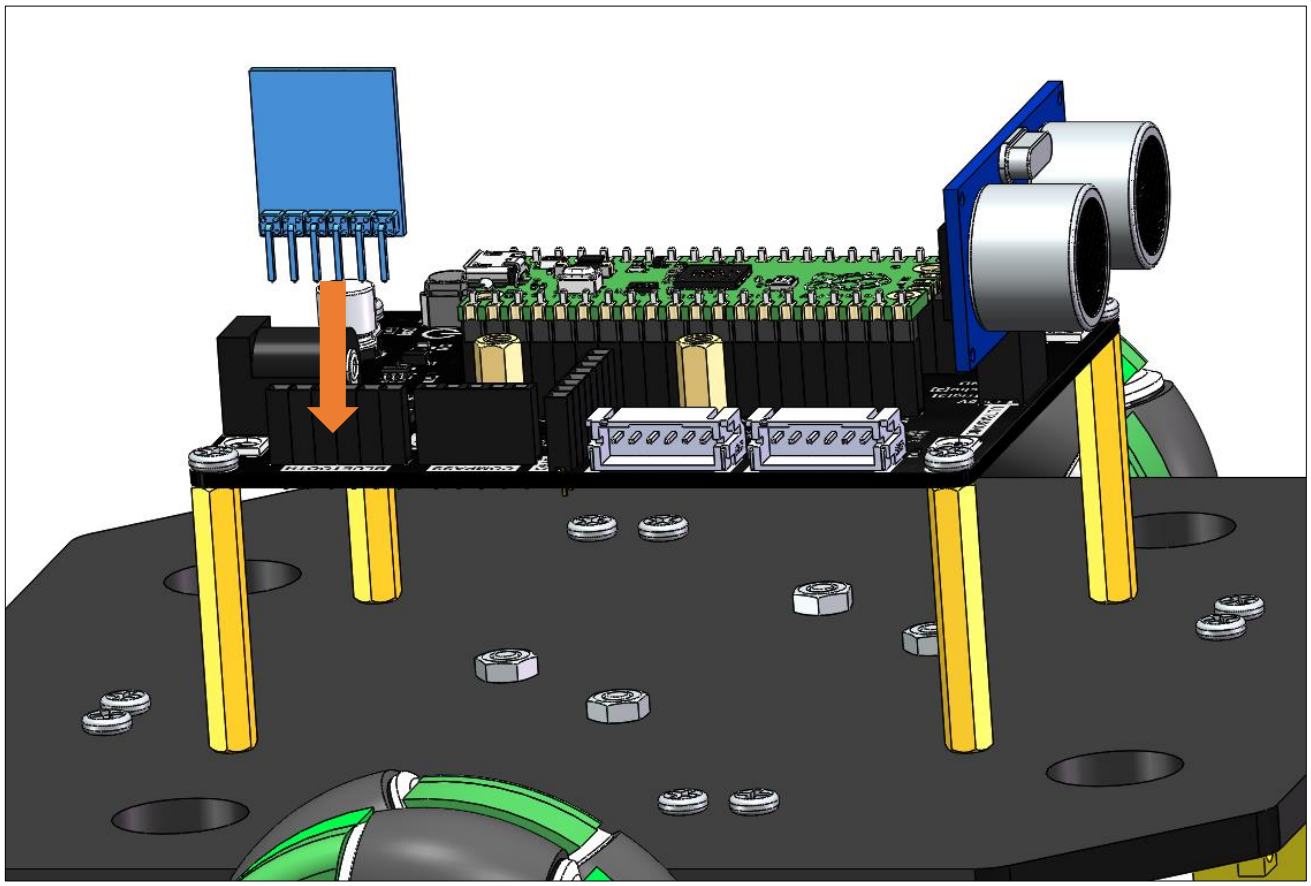
speed_v: the speed of the car's translation, ranging from -255 to 255.
speed_a: the direction of the car's movement, in degrees, ranging from -180 to 180.

Chapter 14 Manual Control

In the head mode, the car's forward direction is based on the orientation of the ultrasonic module, and it can translate in various directions. For controlling the movement, the left joystick modulates the car's translational direction and speed, whereas the right joystick governs the car's rotational direction and velocity. Operating both joysticks simultaneously enables the car to execute circular movements.

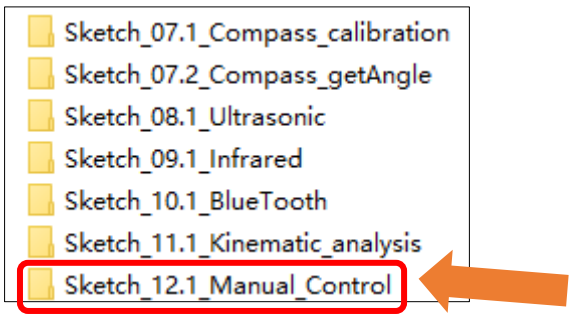
Circuit

In this chapter, we use the assembled car with the Bluetooth module connected. Please refer to Chapter 1 for the detailed assembly process.

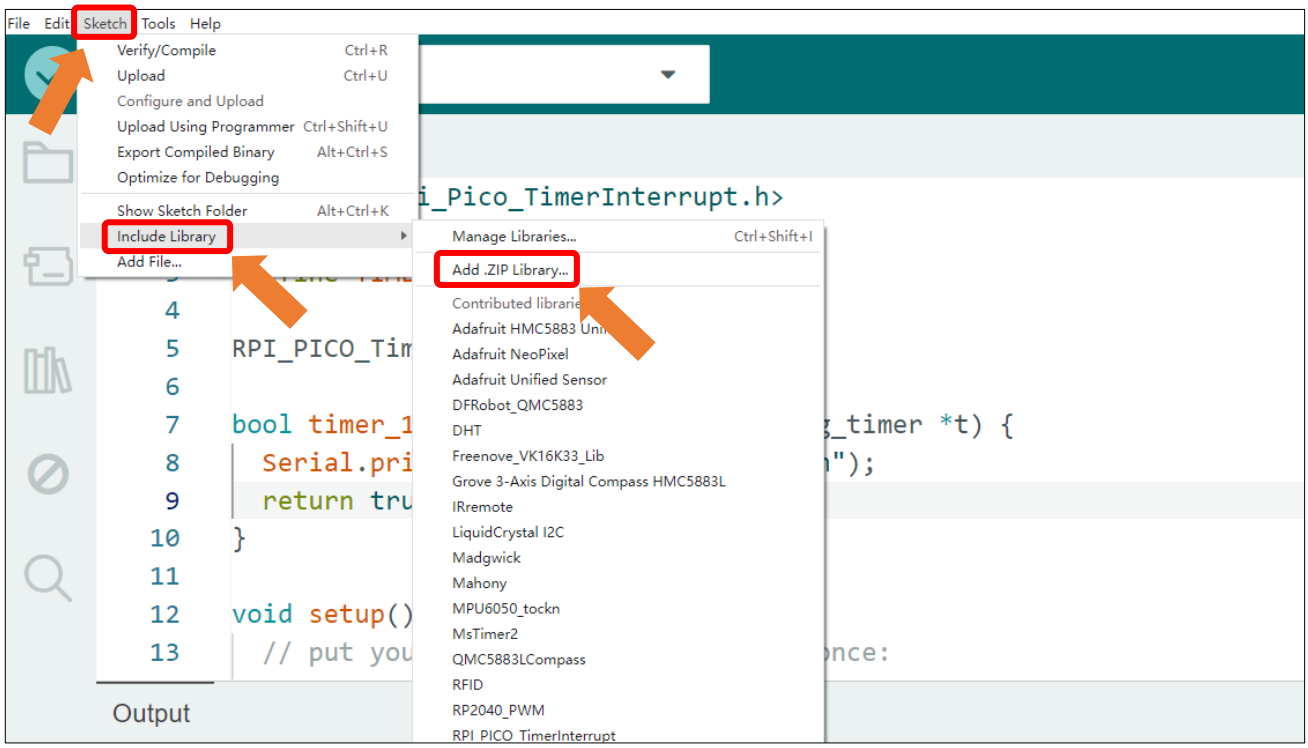


Sketch

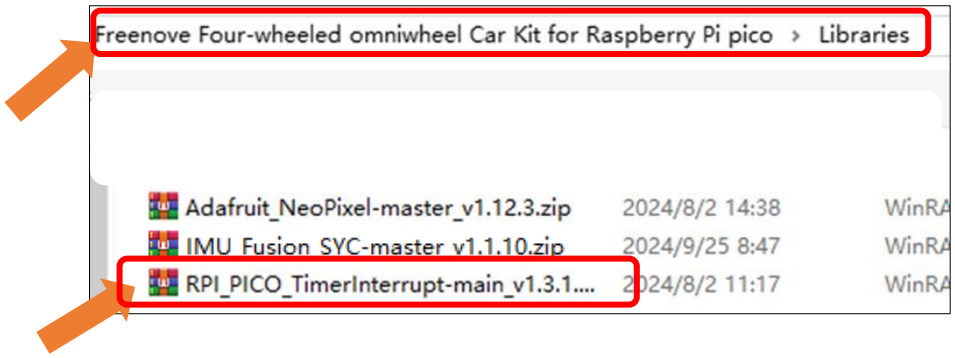
Open "Sketch_12.1_ Manual_Control" folder in **"Freenove_Omni_Wheel_Car_Kit_for_Raspberry_Pi_Pico\Four-Wheel\ \Sketches"** and then double-click **"Sketch_12.1_Manual Control.ino".**



Open Arduino IDE, click **Sketch** on **Menu bar->Include Library ->Add .ZIP library**.



Open **"RPI_PICO_TimerInterrupt-main_v1.3.1.zip"** under the directory **"Freenove Four-wheeled omniwheel Car Kit for Raspberry Pi pico \Libraries".**



Code

Sketch_12.1_Manual_Control

```
void Car_Control() {
  if (paramters[1] == 0 && paramters[2] == 0) {
    angle_out = 0;
    speed_out = 0;
  }else  // If the parameters are not equal to 0
  {
    angle_out = paramters[1];
    speed_out = paramters[2];
  }
  if (paramters[1] == 0 && paramters[2] == 0 && paramters[3] == 0 && paramters[4] == 0) {
    turn_speed = 0;
    turn_location = 0;
  } else {
    turn_speed = paramters[4];     // Rotation speed
    turn_location = paramters[3];  // Direction of rotation
  }
}

bool timer_1ms_control(struct repeating_timer *t) {
  // 5ms detects and accumulates the encoder data once
  if (millis() % 5 == 0) {
    Getencoder_Data();
    speed_add();
  }
  if (millis() % 15 == 0) {
    speed_calculate();  // Find the average speed and filter the burrs
    Turn_Control(speed_out, angle_out, turn_speed,turn_location,1);
  }
  return true;
}
void setup() {
  // put your setup code here, to run once:
  // Set the serial port baud rate to 9600
  Serial.begin(9600);

  // Encoder initialization
  Encoder_Init();

  // Timer interrupt setting
  ITimer0.attachInterruptInterval(TIMER0_INTERVAL_MS, timer_1ms_control);
```

```
41
42      Motor_init();  // Motor initialization
43  }
44
45  void loop() {
46      // Bluetooth processing
47      if (bluetooth.available() > 0) {              // If the serial port receives data
48        String message = bluetooth.readStringUntil('\n'); // Get the transferred instruction
49        if(message == "CONNECT OK\r")
50        {
51          bluetooth_state = true;
52          Serial.println("Bluetooth connected");
53        }
54      }
55      while(bluetooth_state)
56      {
57        if (bluetooth.available() > 0) {                    // If the serial port receives data
58          String message = bluetooth.readStringUntil('\n'); // Get the transmitted instruction
59          Serial.println(message);                          // print order
60
61          Get_Command(message);
62          if(message == "DISCONNECT\r")
63          {
64            bluetooth_state = false;
65            Serial.println("Bluetooth disconnected");
66          }
67          if(CmdArray[0] == CMD_MOTOR_Bluetooth)
68          {
69              Car_Control();
70          }
71          memset(CmdArray, 0, sizeof(CmdArray));
72          memset(paramters, 0, sizeof(paramters));
73        }
74      }
75  }
76  void Get_Command(String inputStringTemp) {
77      // Gets the instruction length
78      int string_length = inputStringTemp.length() + 1;
79      char str[string_length];
80      // Converts an array of String type to a Char type array
81      inputStringTemp.toCharArray(str, string_length);
82      char *token = strtok(str, INTERVAL_CHAr);  // Split the array
83      CmdArray[0] = String(token[0]);     // Put the command into an array
84      for (int i = 0; i < 5; i++) {
```

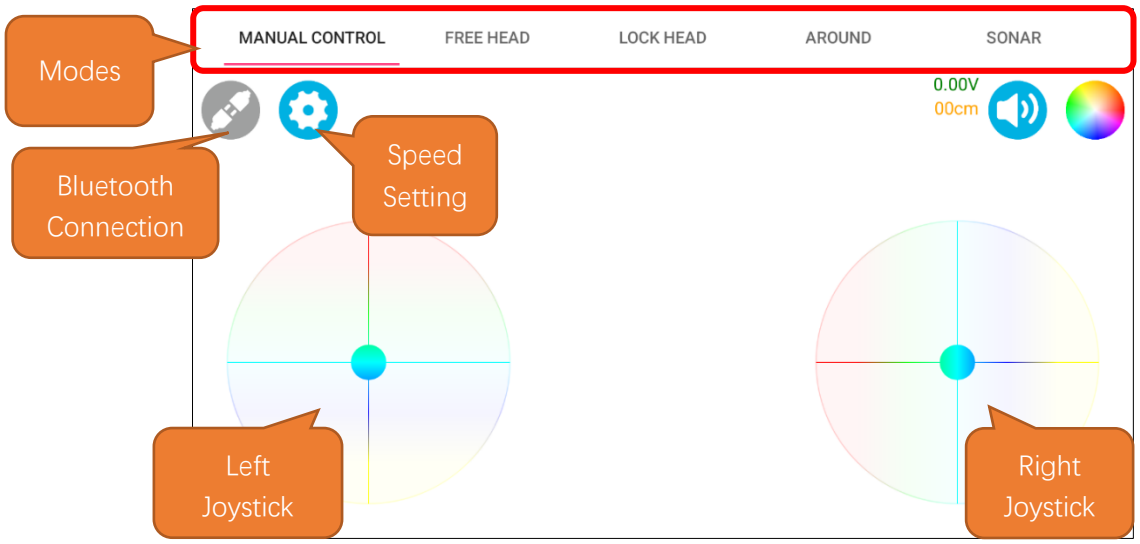
```
85        if (token != NULL) {
86          // Continue splitting the array until # is not detected
87          token = strtok(NULL, INTERVAL_CHAr);
88        }
89        // Convert the instruction to integer type
90        paramters[i + 1] = atoi(token);
91      }
92    }
93
```

Motor.cpp

```
1   void Turn_Control(int speed_v,int speed_a,int angle_v,int location)
2   {
3      vx =  - speed_v * sin ( speed_a * PI / 180 );
4      vy =    speed_v * cos ( speed_a * PI / 180 );
5      if( location >= 0 && location < 180)
6      {
7        angle_v = -angle_v;
8      }
9      v1 = -vx + angle_v;
10     v2 = vy + angle_v;
11     v3 = vx + angle_v;
12     v4 = -vy + angle_v;
13     // Speed PID control
14     pid_v1 = Speed1_PID(v1,speed1);
15     pid_v2 = Speed2_PID(v2,speed2);
16     pid_v3 = Speed3_PID(v3,speed3);
17     pid_v4 = Speed4_PID(v4,speed4);
18     Motor_direction();
19  }
```

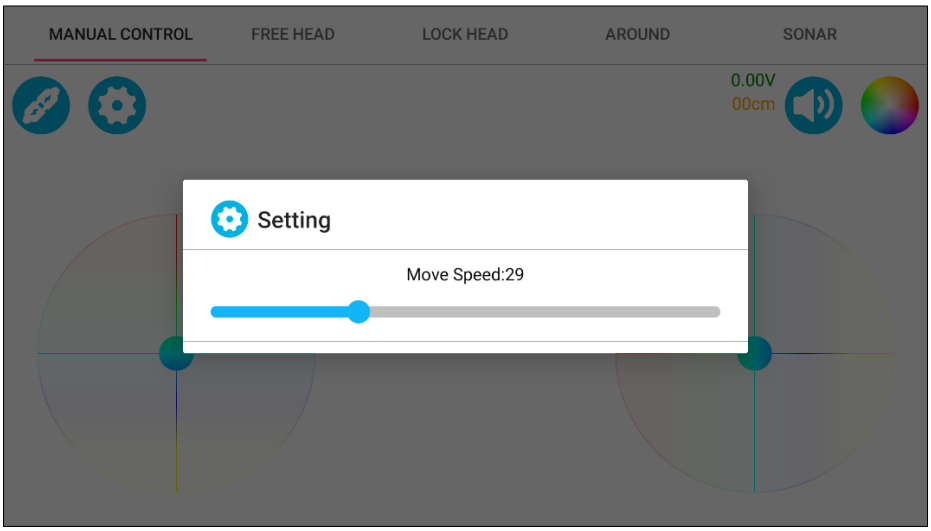
After downloading the code, open Freenove APP on your phone and connect it to the Bluetooth of the car. For installing the APP and connecting Bluetooth, please refer to Introduction to the APP

Interface Introduction

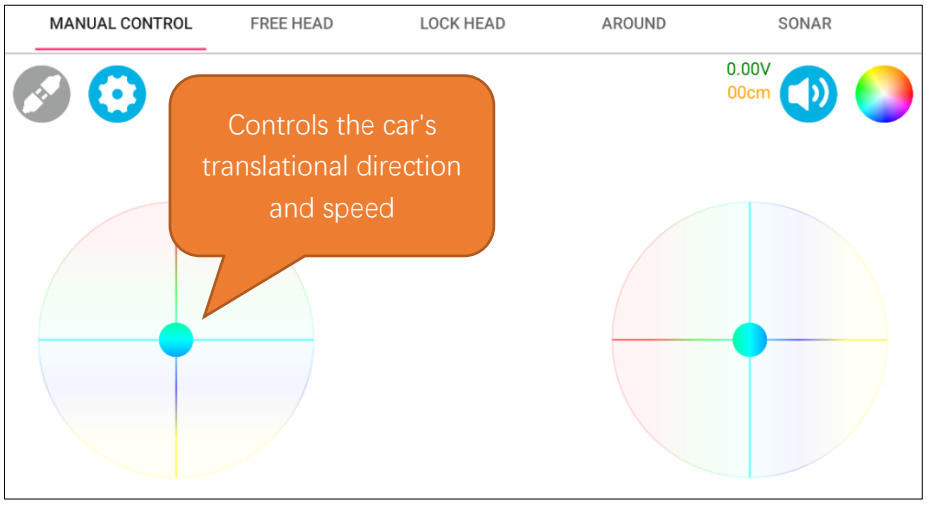


Operation Description

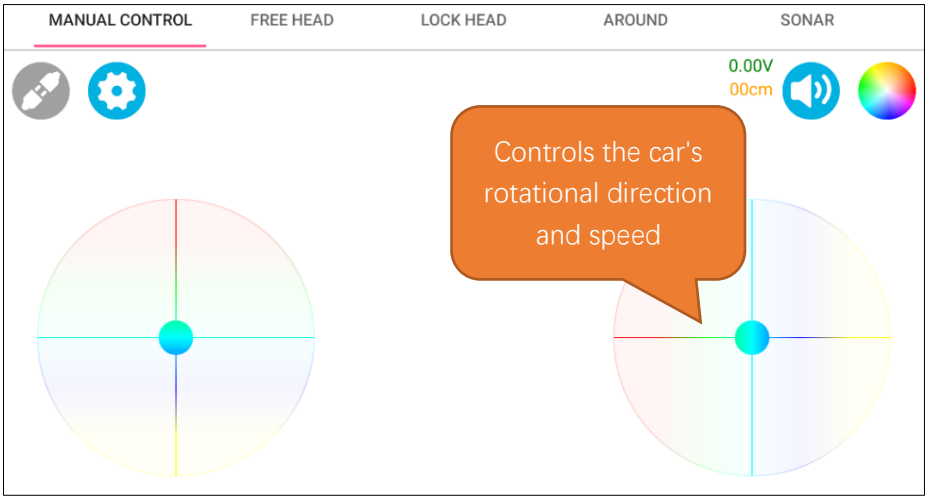
At the speed setting interface, you can set the maximum speed of the car. The range is from 0 to 100.



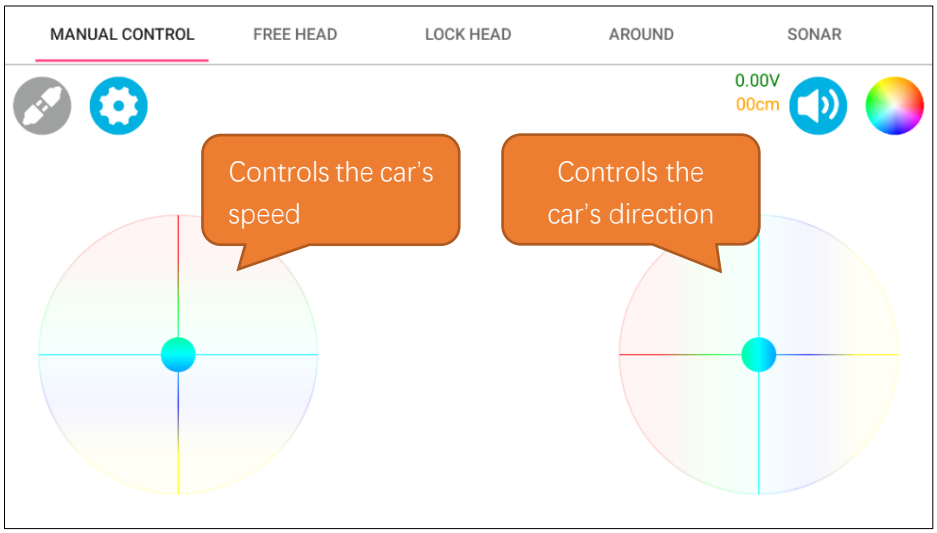
When only the left joystick is operated, it controls the car's translational direction and speed. When it is released, the car stops moving.



When only the right joystick is operated, it controls the car's rotational direction and speed. When it is released, the car stops moving.



Operating both joysticks simultaneously enables the car to execute circular movements, with the left controls the speed and the right controls the direction.



Code Explanation

Call the 'Get_Command()' function to parse the commands sent by the APP, storing integer data into the 'parameters' array and character data into the 'cmdArray' array.

```
void Get_Command(String inputStringTemp) {
  // Gets the instruction length
  int string_length = inputStringTemp.length() + 1;
  char str[string_length];
  // Converts an array of String type to a Char type array
  inputStringTemp.toCharArray(str, string_length);
  char *token = strtok(str, INTERVAL_CHAr);  // Split the array
  CmdArray[0] = String(token[0]);        // Put the command into an array
  for (int i = 0; i < 5; i++) {
    if (token != NULL) {
      // Continue splitting the array until # is not detected
      token = strtok(NULL, INTERVAL_CHAr);
    }
    // Convert the instruction to integer type
    paramters[i + 1] = atoi(token);
  }
}
```

Call the 'Car_Control()' function to control the car to perform corresponding actions according to the commands. For specific command protocols, please refer to the FNK0097 firmware communication protocol.

```
void Car_Control() {
  if (paramters[1] == 0 && paramters[2] == 0) {
    angle_out = 0;
    speed_out = 0;
  }else  // If the parameters are not equal to 0
  {
    angle_out = paramters[1];
    speed_out = paramters[2];
  }
  if (paramters[1] == 0 && paramters[2] == 0 && paramters[3] == 0 && paramters[4] == 0) {
    turn_speed = 0;
    turn_location = 0;
  } else {
    turn_speed = paramters[4];     // Rotation speed
    turn_location = paramters[3];  // Direction of rotation
  }
}
```

Calculate the current speed of the car every 15 milliseconds and call the function 'Turn_Control()' to control the movement of the car.

```
if (millis() % 15 == 0) {
```

```
    speed_calculate();  // Find the average speed and filter the burrs
    Turn_Control(speed_out, angle_out, turn_speed,turn_location, Manual_Control);
}
```

Perform kinematic decomposition on the car to calculate the speed of each wheel and incorporate PID control.

```
void Turn_Control(int speed_v,int speed_a,int angle_v,int location)
{
   vx =  - speed_v * sin ( speed_a * PI / 180 );
   vy =    speed_v * cos ( speed_a * PI / 180 );
   if( location >= 0 && location < 180)
   {
     angle_v = -angle_v;
   }
   v1 = -vx + angle_v;
   v2 = vy + angle_v;
   v3 = vx + angle_v;
   v4 = -vy + angle_v;
   // Speed PID control
   pid_v1 = Speed1_PID(v1,speed1);
   pid_v2 = Speed2_PID(v2,speed2);
   pid_v3 = Speed3_PID(v3,speed3);
   pid_v4 = Speed4_PID(v4,speed4);
   Motor_direction();
}
```

Reference

char *strtok(char *str, const char *delim);

This function is used to split a string into substrings based on a delimiter.
Str: The string to be split.
Delim: The delimiter character.

int atoi(const char *str);

This function is used to convert a string to an integer.
Str: The string representing the number to be converted.

void Turn_Control(int speed_v,int speed_a,int angle_v,int location);

This function is used to control the car's movement.

Parameters:
speed_v: The movement velocity of the car.
speed_a: The movement angle of the car.
angle_v: The angular velocity of the car.
location: The direction in which the car is turning.

Chapter 15 Free Head Mode

In Free head mode, the car's movement is not dictated by its own heading but is aligned with an absolute spatial direction. Upon switching to this mode, the system automatically adopts the current orientation of the ultrasonic module as the forward direction for the car. Tapping the Free head mode icon again allows for realignment of the forward direction. In controlling the car, the left joystick modulates the car's translational angle and velocity, whereas the right joystick governs the car's rotational direction and speed. Operating both joysticks simultaneously enables the car to execute a movement that combines rotation with omnidirectional motion.

Related Knowledge

Digital Compass and Gyroscope Data Fusion

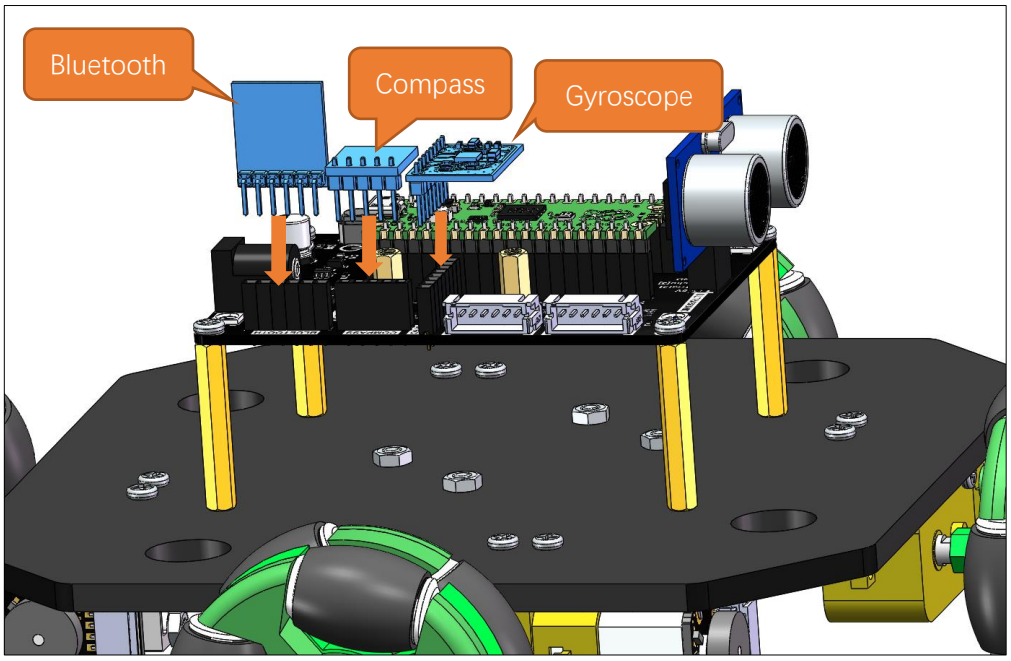
We have previously discussed the methods for obtaining angles from electronic compasses and gyroscopes. Each of these sensors has its own strengths, but they also face certain limitations when used separately. Gyroscopes are known for their good dynamic response and resistance to noise interference, but they are prone to drift at zero points. Conversely, digital compasses provide better performance in static conditions, yet they can be susceptible to instability in their outputs due to noise from motor startups, resulting in vehicle drift. Thus, by effectively integrating the data from both sensors, we can enhance the precision and reliability of posture estimation.

Sensors	Final Output Value	Advantages	Drawbacks
Gyroscope	Rotational Angle	good dynamic response and strong anti-interference capabilities	Zero Drift
Compass	Magnetic Field Angle	good static response and high accuracy	Bad anti-interference capabilities

To learn more about data fusion, you may refer to IMU_Fusion_SYC

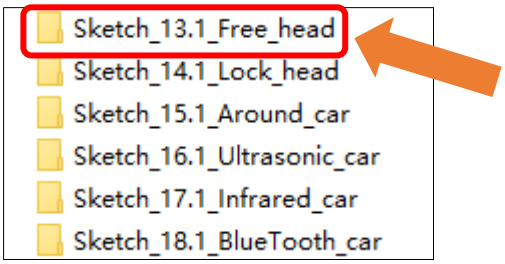
Circuit

Connect the Bluetooth module, gyroscope, and digital compass to the assembled car.
For detailed assembly process of the car, please refer to Chapter 1



Sketch

Open "Sketch_13.1_ Free_head" folder in **"Freenove_Omni_Wheel_Car_Kit_for_Raspberry_Pi_Pico\Four-Wheel\ \Sketches"** and then double-click **"Sketch_13.1_Free_head.ino".**



Code

Sketch_13.1_Free_head.ino

```
1  void Car_2_Control() {
2    if (paramters[1] == 0 && paramters[2] == 0) {
3      angle_out = 0;
4      speed_out = 0;
5    }else  //If the parameters are not equal to 0
6    {
7      angle_out = paramters[1];
8      speed_out = paramters[2];
9    }
10
```

```
    if(paramters[1] == 0 && paramters[2] == 0 && paramters[3] == 0 && paramters[4] == 0  &&
abs(AccMagnitude - 0.98) <= 0.2) {
      turn_speed = 0;
      turn_location = 0;
      angle_flag = angle;
      angle_state = 0;
    }else if (paramters[3] == 0 && paramters[4] == 0 && (paramters[1] != 0 || paramters[2] !=
0))
    {
      if(angle_flagstate == 1)
      {
        angle_flag = angle;
        angle_flagstate = 0;
      }
      angle_state = 1;
    }else if ((paramters[3] != 0 || paramters[4] != 0) && paramters[1] == 0 && paramters[2] ==
0)
    {
      angle_state = 0;
      angle_flag = angle;
      turn_speed = paramters[4];
      turn_location = paramters[3];
    }
    else{
      angle_state = 2;
      angle_flag = angle;
      turn_speed = paramters[4];
      turn_location = paramters[3];  // Rotation speed
      angle_flagstate = 1;           // Direction of rotation
    }
}

bool timer_1ms_control(struct repeating_timer *t) {
    // 5ms detects and accumulates the encoder data once
    if (millis() % 5 == 0) {
      Getencoder_Data();
      speed_add();
    }
    // 10ms Detects the current Angle once
    if (millis() % 10 == 0) {
      imu_state = 1;
    }
    if (millis() % 15 == 0) {
      speed_calculate();  // Find the average speed and filter the burrs
```

```
55        Turn_Control(speed_out, angle_out, turn_speed, angle_a_out, turn_location, 0);
56      }
57      return true;
58  }
59
60  void setup() {
61      // put your setup code here, to run once:
62      Serial.begin(9600);
63      bluetooth.begin(115200);
64
65      // Encoder initialization
66      Encoder_Init();
67
68      IMU_Init();
69
70      // Motor initialization
71      Motor_init();
72
73      int current_millis = millis();
74      while(millis() - current_millis < 10)
75      {
76        IMU_GetData();
77        angle_head = angle;
78      }
79
80      // Timer interrupt Settings
81      ITimer0.attachInterruptInterval(TIMER0_INTERVAL_MS, timer_1ms_control);
82  }
83
84  void loop() {
85      if (imu_state == 1) {
86          IMU_GetData();
87          imu_state = 0;
88      }
89      // Bluetooth processing
90      if (bluetooth.available() > 0) {              // If the serial port receives data
91        String message = bluetooth.readStringUntil('\n'); // Get the transferred instruction
92        if(message == "CONNECT OK\r")
93        {
94          bluetooth_state = true;
95          Serial.println("Bluetooth connected");
96        }
97      }
98      while(bluetooth_state)
```

```
99      {
100       if (imu_state == 1) {
101         IMU_GetData();
102         imu_state = 0;
103       }
104       if (bluetooth.available() > 0) {              // If the serial port receives data
105         String message = bluetooth.readStringUntil('\n'); // Get the transmitted instruction
106         Serial.println(message);                    // print order
107         Get_Command(message);
108         if(message == "DISCONNECT\r")
109         {
110           bluetooth_state = false;
111           Serial.println("Bluetooth disconnected");
112         }
113         if(CmdArray[0] == CMD_STATE_Bluetooth)
114         {
115           state = paramters[1];
116           car_stop();
117           if(paramters[1] == 0)
118           {
119             angle_head = angle;
120             angle_flag = angle_head;
121         }
122         }
123         if(CmdArray[0] == CMD_MOTOR_Bluetooth)
124         {
125           Car_2_Control();
126         }
127         if(CmdArray[0] == CMD_BUZZER_Bluetooth)
128         {
129           if(state == 0)
130           {
131             angle_head = angle;
132             angle_flag = angle_head;
133           }
134         }
135         memset(CmdArray, 0, sizeof(CmdArray));
136         memset(paramters, 0, sizeof(paramters));
137       }
138     }
139   }
140
141   void Get_Command(String inputStringTemp) {
142     int string_length = inputStringTemp.length() + 1;
```

```
    char str[string_length];
    inputStringTemp.toCharArray(str, string_length);
    char *token = strtok(str, INTERVAL_CHAr);
    CmdArray[0] = String(token[0]);
    for (int i = 0; i < 5; i++) {
      if (token != NULL) {
        token = strtok(NULL, INTERVAL_CHAr);
      }
      paramters[i + 1] = atoi(token);
    }
}
```

Angle.cpp:

```
void IMU_GetData()
{
    imu.Calculate();
    AccMagnitude = imu.getAccMagnitude();
    float newValue = imu.Data_Fusion(); // Get fusion data
    // Update the window
    sum -= data[index_];    // Subtract the old value
    data[index_] = newValue;// Add a new value
    sum += newValue;        // Accumulate new values
    index_ = (index_ + 1) % windowSize; // Update the index

    // Calculate the average value
    angle = sum / windowSize;
}
```

Motor.cpp

```
void Turn_Control(int speed_v,int speed_a,int angle_v, int location)
{
    vx =  - speed_v * sin ( (angle_flag - angle_head + speed_a) * PI / 180 );
    vy =    speed_v * cos ( (angle_flag - angle_head + speed_a) * PI / 180 );
    if(angle_state == 0)
    {
      if( location >= 0 && location < 180)
      {
        angle_v = -angle_v;
      }
    }else if(angle_state == 1){// Push the left joysticks only
      angle_v = Angle_PID_Realize(angle_flag,angle);
      if(angle_v > 50) angle_v = 50;else if(angle_v < -50) angle_v = -50;
    }else if(angle_state == 2){// Push the left and right joysticks at the same time
      angle_counter++;
      if(angle_counter < 3)
      {
```

```
        if( location > 0 && location < 180)
        {
          angle_v = -angle_v;
          location_state = 1;
        }
        if( location > -180 && location < 0)
        {
          location_state = 2;
        }
        v1 = angle_v;
        v2 = angle_v;
        v3 = angle_v;
        v4 = angle_v;
      }
      else if(angle_counter < 4)
      {
        angle_flag = angle;
        if(location_state == 1)
          angle_flag -= 10;
        else if(location_state == 2)
          angle_flag += 20;
        vx =  - speed_v * sin ( (angle_flag - angle_head + speed_a) * PI / 180 );
        vy =    speed_v * cos ( (angle_flag - angle_head + speed_a) * PI / 180 );
        location_state = 0;
        v1 = -vx;
        v2 = vy;
        v3 = vx;
        v4 = -vy;
      }
      else{
        angle_counter = 0;
      }
    }
    if(angle_state != 2)
    {
        v1 = -vx + angle_v;
        v2 = vy + angle_v;
        v3 = vx + angle_v;
        v4 = -vy + angle_v;
    }
    // Speed PID control
    pid_v1 = Speed1_PID(v1,speed1);
    pid_v2 = Speed2_PID(v2,speed2);
    pid_v3 = Speed3_PID(v3,speed3);
```

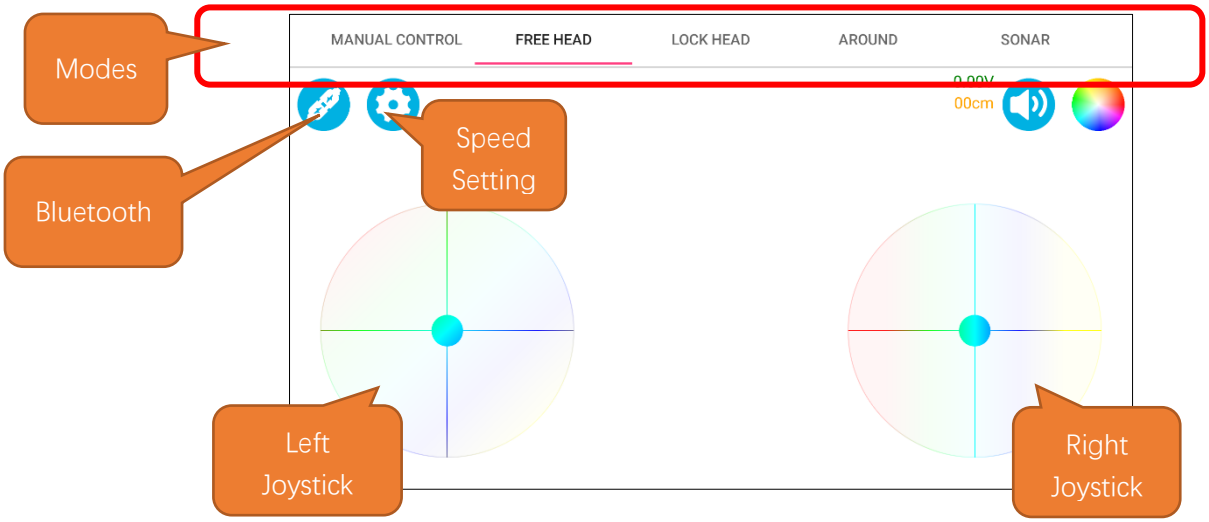
```
62      pid_v4 = Speed4_PID(v4,speed4);
63      Motor_direction();
64  }
```

Before uploading the sketch, please make sure the IMU_Fusion_SYC library has been installed, so that the functions and tools it provides can be used. If the library was not installed yet, please refer to IMU_Fusion_SYC.

After downloading the code, open Freenove APP and click the connection icon to connect the Bluetooth. For installing the APP and Bluetooth connection, please refer to Introduction to the APP

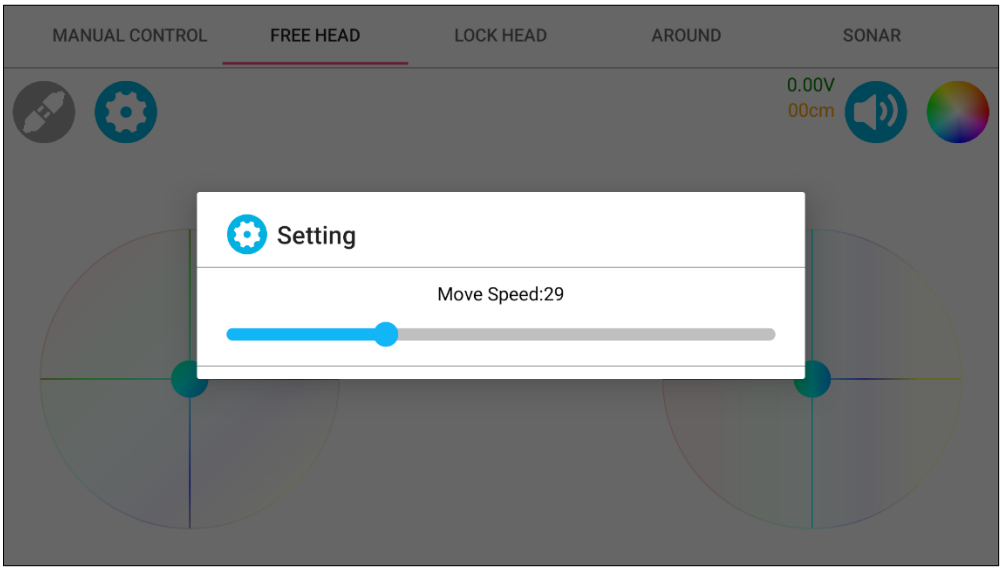
If you need any support, please feel free to contact us via: support@freenove.com

Interface Introduction

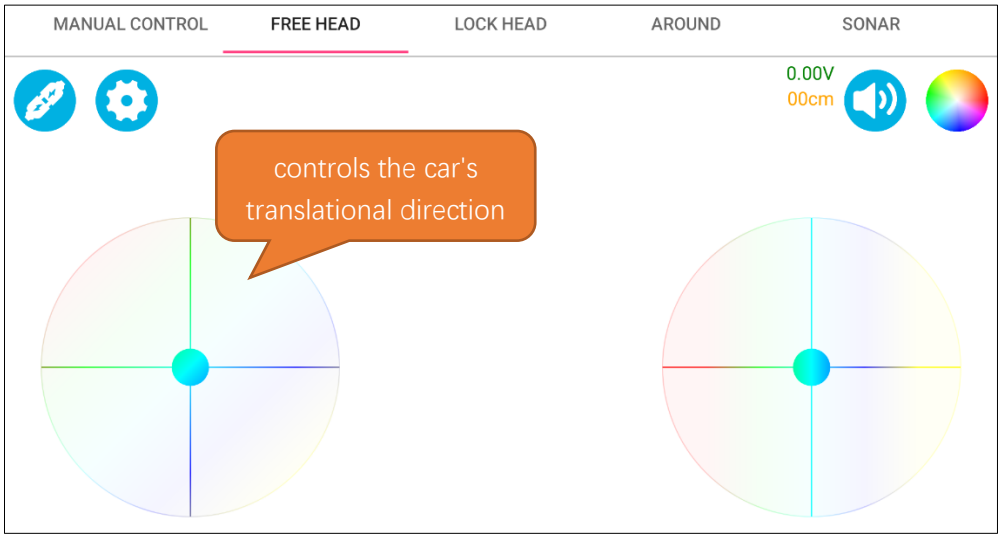


Operation Description

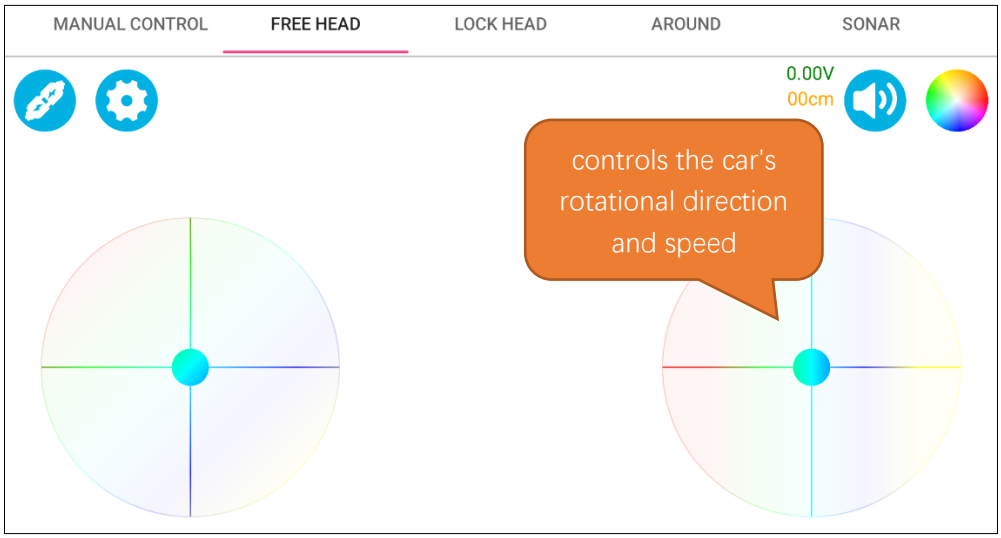
At the speed setting interface, you can set the maximum speed of the car. The range is from 0 to 100.



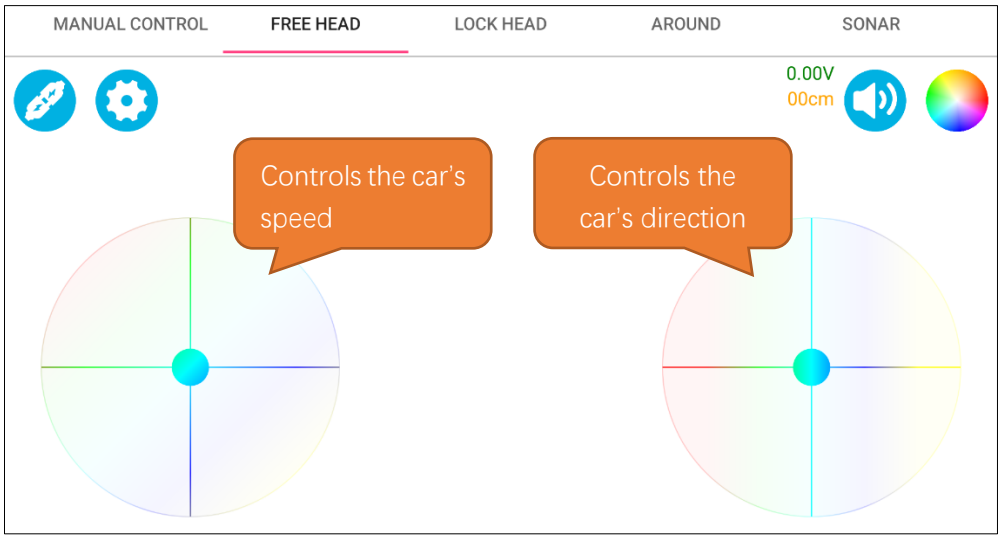
When only the left joystick is operated, it controls the car's translational direction and speed. When it is released, the car stops moving.



When only the right joystick is operated, it controls the car's rotational direction and speed. When it is released, the car stops moving.



Operating both joysticks simultaneously enables the car to execute circular movements, with the left controls the speed and the right controls the direction.



Tap the FREE HEAD button again will change the direction of the car's head to the current direction.

Code Explanation

Call the function IMU_GetData() to obtain the fused angle data. Accumulate the obtained angles and calculate the average multiple times to achieve a more stable fused angle.

```
void IMU_GetData()
{
   imu.Calculate();// Calculate the raw data
   angle = imu.Data_Fusion(); // Get fusion data
}
```

Call the Car_2_Control() function to control the car to perform specific actions. Enumerate the possible operations of the joystick and set the corresponding motion logic based on different states.

```
void Car_Control() {
   if (paramters[1] == 0 && paramters[2] == 0) {//
     angle_out = 0;
     speed_out = 0;
   }else  //If the parameters are not equal to 0
   {
     angle_out = paramters[1];
     speed_out = paramters[2];
   }
   if(paramters[1] == 0 && paramters[2] == 0 && paramters[3] == 0 && paramters[4] == 0  &&
abs(AccMagnitude - 0.98) <= 0.2) {
     turn_speed = 0;
     turn_location = 0;
     angle_flag = angle;
     angle_state = 0;
   }else if (paramters[3] == 0 && paramters[4] == 0 && (paramters[1] != 0 || paramters[2] !=
0))
```

```
    {
      if(angle_flagstate == 1)
      {
        angle_flag = angle;
        angle_flagstate = 0;
      }
      angle_state = 1;
    }else if ((paramters[3] != 0 || paramters[4] != 0) && paramters[1] == 0 && paramters[2] ==
0)
    {
      angle_state = 0;
      angle_flag = angle;
      turn_speed = paramters[4];
      turn_location = paramters[3];
    }
    else{
      angle_state = 2;
      angle_flag = angle;
      turn_speed = paramters[4];
      turn_location = paramters[3];  // Rotation speed
      angle_flagstate = 1;           // Direction of rotation
    }
}
```

Reference

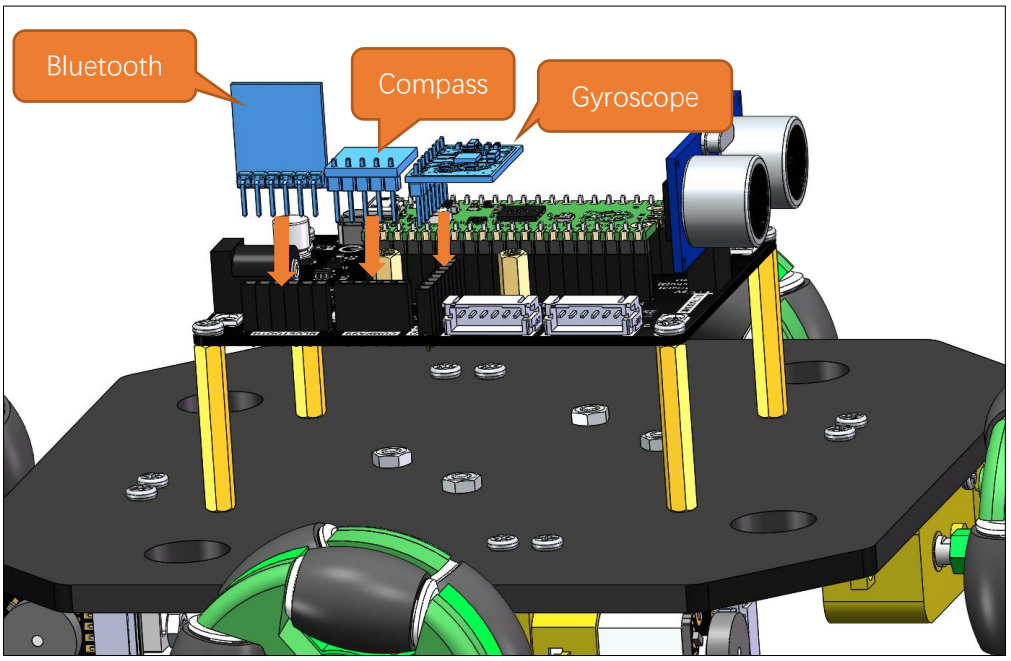
IMU::Calculate()
This function is located in the IMU_Fusion_SYC library, which is called to calculate various angle data.
IMU::getAccMagnitude()
This function is located in the IMU_Fusion_SYC library, which is called to calculate the current acceleration.
IMU::Data_Fusion()
This function is located in the IMU_Fusion_SYC library, which is called to compute the angles after data fusion from sensors such as the MPU6050 and QMC5883L

Chapter 16 Lock Head Mode

In Lock Head Mode, the car's direction of travel is consistently aligned with an absolute direction in space. Even if the car is manually lifted and its orientation is altered, it will automatically realign to its original angle. To control the car, the left joystick is used to adjust the translational speed and direction, while the right joystick is to control its rotational angle.

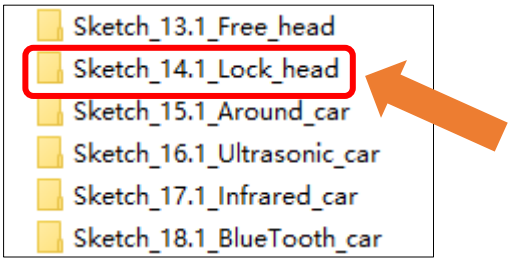
Circuit

Connect the Bluetooth module, gyroscope, and digital compass to the assembled car.
For detailed assembly process of the car, please refer to Chapter 1



Sketch

Open "**Sketch_14.1_Lock_head**" folder in **"Freenove_Omni_Wheel_Car_Kit_for_Raspberry_Pi_Pico\Four-Wheel\ \Sketches"** and then double-click **"Sketch_14.1_Lock_head.ino".**



Code

Sketch_14.1_Lock_head.ino

```
void Car_3_Control() {
   if (paramters[1] == 0 && paramters[2] == 0) {
     angle_out = 0;
     speed_out = 0;
   }
   else               //If the parameters are not equal to 0
   {
     angle_out = paramters[1];
     speed_out = paramters[2];
   }
   if(paramters[1] == 0 && paramters[2] == 0 && paramters[3] == 0 && paramters[4] == 0)
   {
     angle_a_out = angle_head;
   }
   else{

     angle_temp = paramters[3];
     if(angle_temp > -180 && angle_temp <= 0)
     {
       angle_a_out = map(angle_temp,0,-180,angle_head,angle_head + 180);
       if(angle_a_out > 360)
       {
         angle_a_out -= 360;
       }
     }
     if(angle_temp >= 0 && angle_temp < 180)
     {
       angle_a_out = map(angle_temp,0,180,angle_head,angle_head - 180);
     }
   }
}

bool timer_1ms_control(struct repeating_timer *t) {
   // 5ms detects and accumulates the encoder data once
   if (millis() % 5 == 0) {
     Getencoder_Data();
     speed_add();
   }
   // 10ms Detects the current Angle once
   if (millis() % 10 == 0) {
     imu_state = 1;
```

```
42      }
43      if (millis() % 15 == 0) {
44        speed_calculate();  // Find the average speed and filter the burrs
45        Turn_Control(speed_out, angle_out, turn_speed ,angle_a_out ,turn_location ,2);
46      }
47      return true;
48  }
49
50  void setup() {
51      // put your setup code here, to run once:
52      // Set the serial port baud rate to 9600
53      Serial.begin(9600);
54      bluetooth.begin(115200);
55
56      // Encoder initialization
57      Encoder_Init();
58
59      IMU_Init();
60
61      // Motor initialization
62      Motor_init();
63
64      int current_millis = millis();
65      while(millis() - current_millis < 10)
66      {
67        IMU_GetData();
68        angle_head = angle;
69      }
70
71      // Timer interrupt Settings
72      ITimer0.attachInterruptInterval(TIMER0_INTERVAL_MS, timer_1ms_control);
73  }
74
75  void loop() {
76      // Bluetooth processing
77      if (imu_state == 1) {
78          IMU_GetData();
79          imu_state = 0;
80      }
81      if (bluetooth.available() > 0) {              // If the serial port receives data
82        String message = bluetooth.readStringUntil('\n'); // Get the transferred instruction
83        if(message == "CONNECT OK\r")
84        {
85          bluetooth_state = 1;
```

```
        Serial.println("Bluetooth connected");
      }
    }
    while(bluetooth_state)
    {
      if (imu_state == 1) {
        IMU_GetData();
        imu_state = 0;
      }
      if (bluetooth.available() > 0) {                    // If the serial port receives data
        String message = bluetooth.readStringUntil('\n'); // Get the transmitted instruction
        Serial.println(message);                          // print order

        Get_Command(message);
        if(message == "DISCONNECT\r")
        {
          bluetooth_state = 0;
          Serial.println("Bluetooth disconnected");
        }
        if(CmdArray[0] == CMD_STATE_Bluetooth)
        {
          state = paramters[1];
          car_stop();
        }
        if(CmdArray[0] == CMD_MOTOR_Bluetooth)
        {
          if(state == 2)
            Car_3_Control();
        }
        memset(CmdArray, 0, sizeof(CmdArray));
        memset(paramters, 0, sizeof(paramters));
      }
    }
}

void Get_Command(String inputStringTemp) {
    int string_length = inputStringTemp.length() + 1;
    char str[string_length];
    inputStringTemp.toCharArray(str, string_length);
    char *token = strtok(str, INTERVAL_CHAr);
    CmdArray[0] = String(token[0]);
    for (int i = 0; i < 5; i++) {
      if (token != NULL) {
        token = strtok(NULL, INTERVAL_CHAr);
```

```
130         }
131         paramters[i + 1] = atoi(token);
132       }
133   }
134
```

Motor.cpp

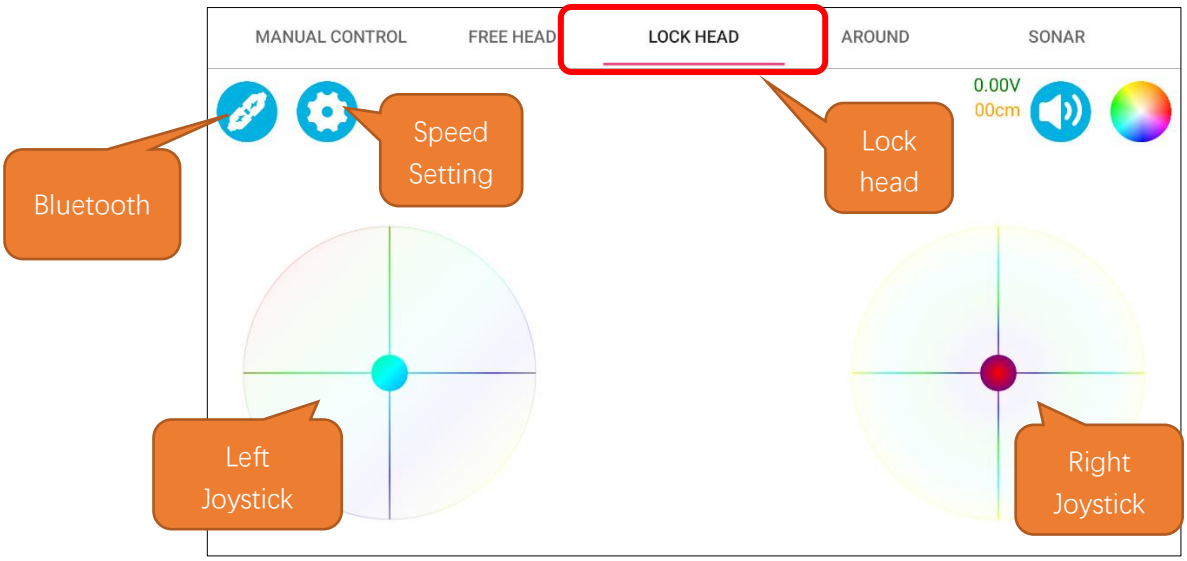
```
1   void Turn_Control(int speed_v,int speed_a,int angle_v,int angle_a)
2   {
3       angle_v = Angle_PID_Realize(angle_a,angle);
4
5       if(angle_v > 50) angle_v = 50;else if(angle_v < -50) angle_v = -50;
6
7       vx =  - speed_v * sin ( (angle - angle_head + speed_a) * PI / 180 );
8       vy =    speed_v * cos ( (angle - angle_head + speed_a) * PI / 180 );
9
10      v1 = -vx + angle_v;
11      v2 = vy + angle_v;
12      v3 = vx + angle_v;
13      v4 = -vy + angle_v;
14      // Speed PID control
15      pid_v1 = Speed1_PID(v1,speed1);
16      pid_v2 = Speed2_PID(v2,speed2);
17      pid_v3 = Speed3_PID(v3,speed3);
18      pid_v4 = Speed4_PID(v4,speed4);
19      Motor_direction();
20  }
```

Before uploading the sketch, please make sure the IMU_Fusion_SYC library has been installed, so that the functions and tools it provides can be used. If the library was not installed yet, please refer to IMU_Fusion_SYC. After downloading the code, open Freenove APP and click the connection icon to connect the Bluetooth. For installing the APP and Bluetooth connection, please refer to Introduction to the APP

Caution: Digital compasses are very sensitive to magnetic disturbances. After adding additional modules near the compass, it is essential to recalibrate it. Failure to do so may result in unstable operation of the car!

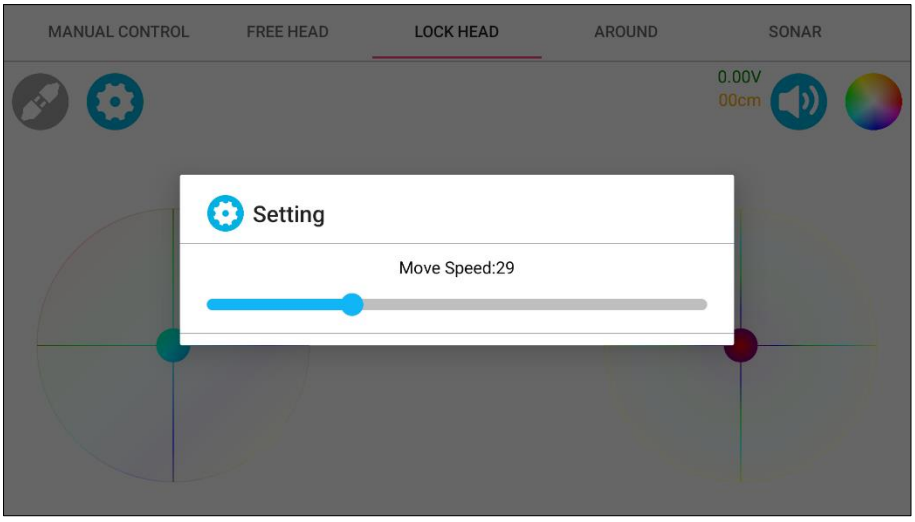
If you need any support, please feel free to contact us via: **support@freenove.com**

Interface Introduction

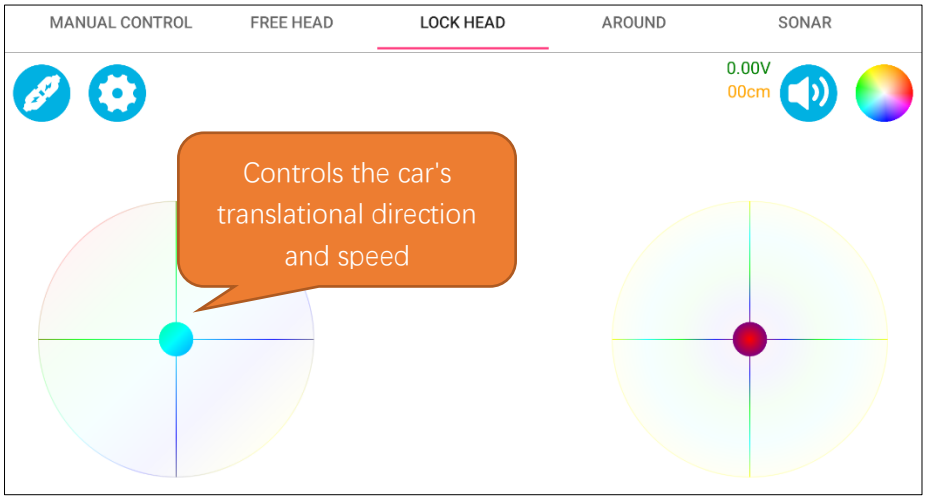


Operation Description

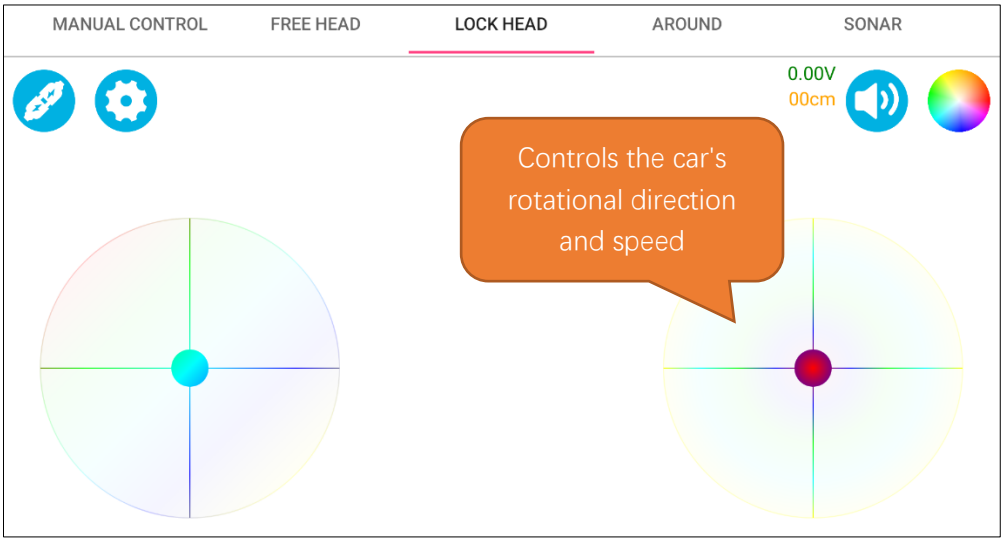
At the speed setting interface, you can set the maximum speed of the car. The range is from 0 to 100.



When only the left joystick is operated, it controls the car's translational direction and speed. When it is released, the car stops moving.



When only the right joystick is actuated, it controls the target orientation angle of the car, and the car will hold at the corresponding angle when the joystick is released.



Code Explanation

Call the Car_3_Control() function to control the car to perform corresponding actions according to the instructions.

```
void Car_3_Control() {
  if (paramters[1] == 0 && paramters[2] == 0) {
    angle_out = 0;
    speed_out = 0;
  }
  else               //If the parameters are not equal to 0
  {
    angle_out = paramters[1];
    speed_out = paramters[2];
  }
  if(paramters[1] == 0 && paramters[2] == 0 && paramters[3] == 0 && paramters[4] == 0)
  {
    angle_a_out = angle_head;
  }
  else{
    angle_temp = paramters[3];
    if(angle_temp > -180 && angle_temp <= 0)
    {
      angle_a_out = map(angle_temp,0,-180,angle_head,angle_head + 180);
      if(angle_a_out > 360)
      {
        angle_a_out -= 360;
      }
    }
    if(angle_temp >= 0 && angle_temp < 180)
    {
      angle_a_out = map(angle_temp,0,180,angle_head,angle_head - 180);
    }
  }
}
```

Use a timer interrupt to control the car to check the encoder value every 5ms, detect the angle every 10ms, and measure the speed and control the car's movement according to the encoder value every 15ms.

```
bool timer_1ms_control(struct repeating_timer *t) {
  // 5ms detects and accumulates the encoder data once
  if (millis() % 5 == 0) {
    Getencoder_Data();
    speed_add();
  }
  // 10ms Detects the current Angle once
  if (millis() % 10 == 0) {
```

```
9         imu_state = 1;
10      }
11      if (millis() % 15 == 0) {
12        speed_calculate();  // Find the average speed and filter the burrs
13        Turn_Control(speed_out, angle_out, turn_speed ,angle_a_out ,turn_location ,2);
14      }
15      return true;
16  }
```

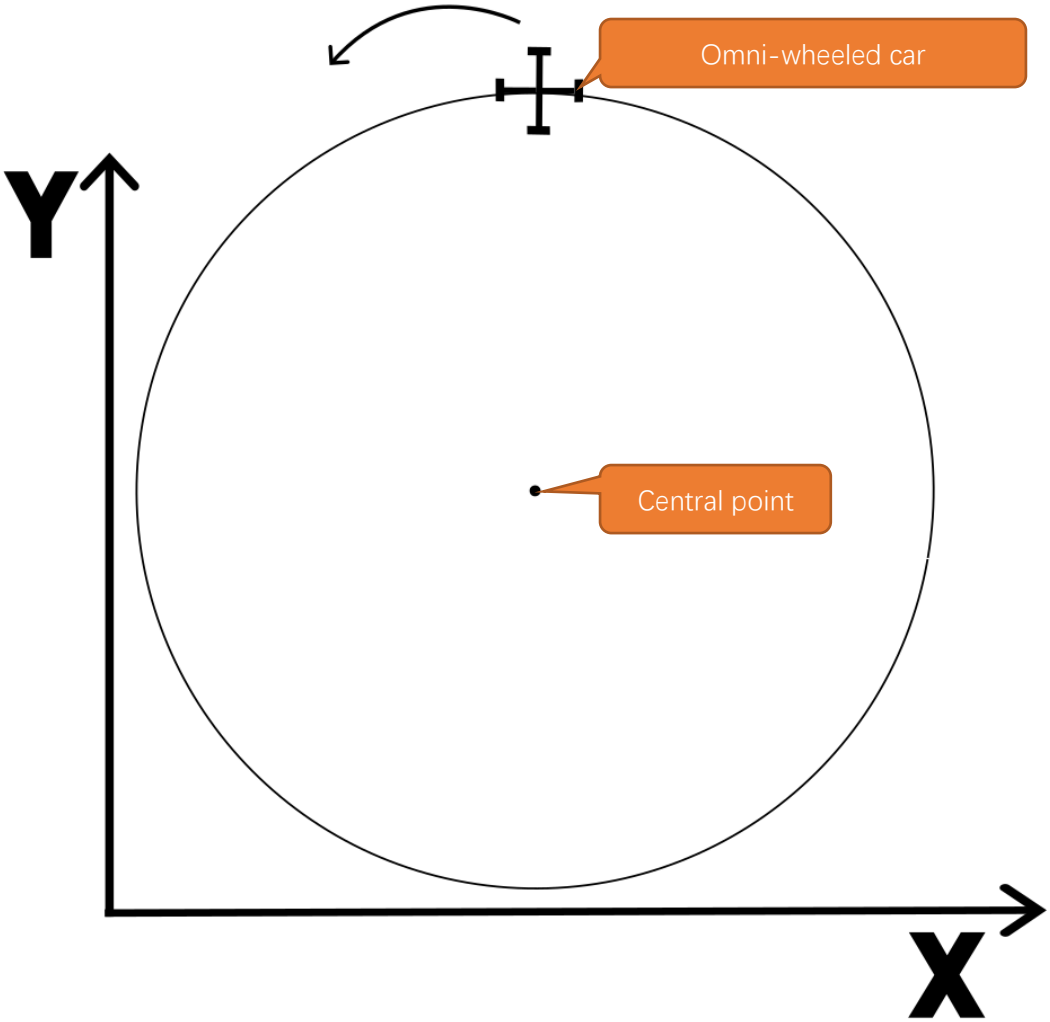
Chapter 17 Around Mode

In the Around Mode, the vehicle consistently orients itself around a fixed central point using the ultrasonic module as a reference, maintaining a constant speed for circular motion. Users can modify the car's orbital radius and direction via Freenove APP, with adjustable radius settings from 0 to 100 centimeters.

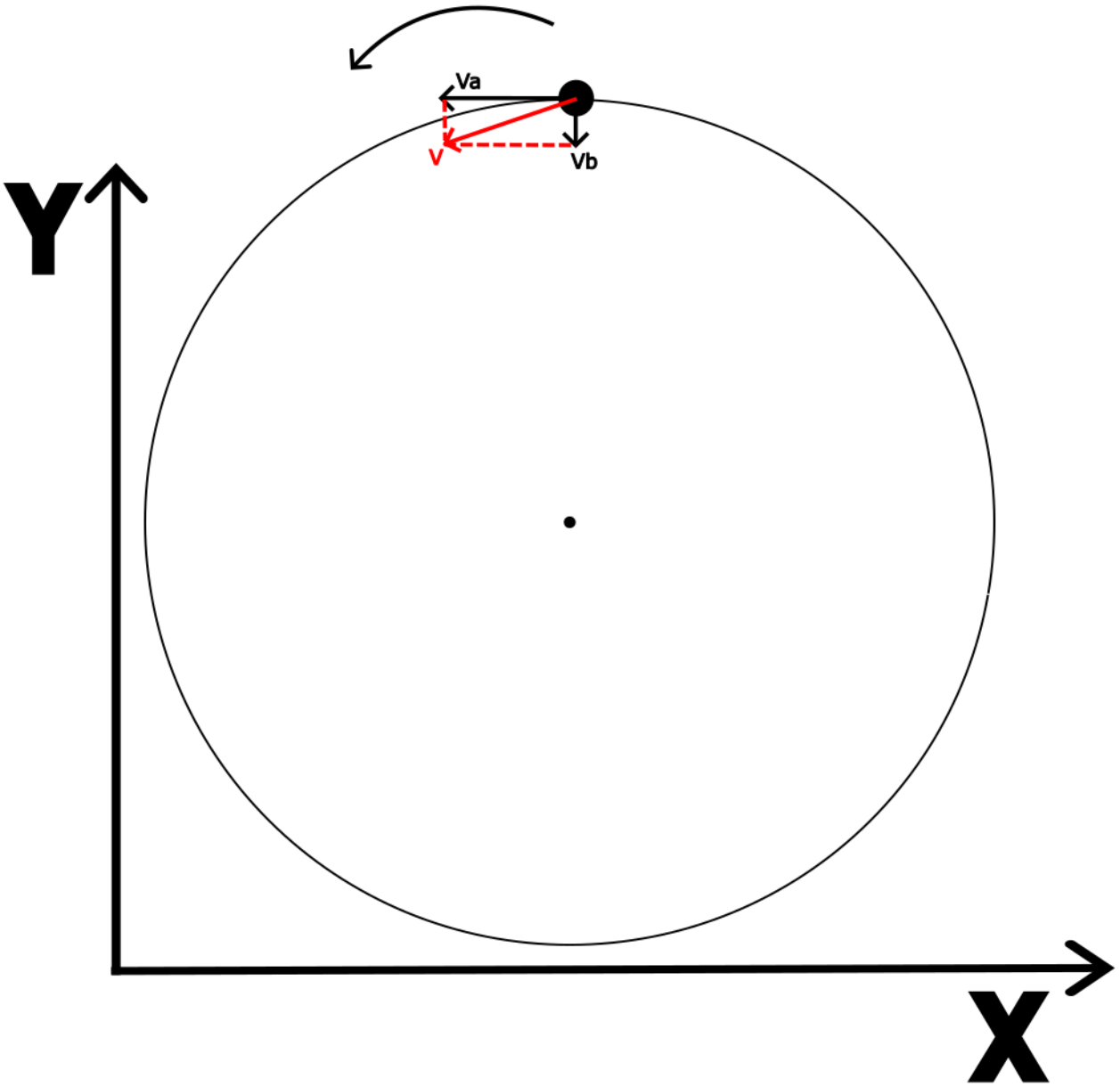
Related Knowledge

Around Mode Attitude Calculating

Around mode essentially enables the car to perform circular motion, which is a common type of composite movement where an object moves along a circular path around a central point.



To simplify the model for analysis, we can consider the omni-wheeled car as a particle for motion analysis. At this point, we can obtain the analytical results as shown in the following figure:



We place the car stationary on a point of the circular trajectory and decompose its velocity **V** into components **Va** and **Vb** along the x-axis and y-axis. **Va** is always aligned with the tangent to the circle at that point, while **Vb** points consistently towards the center of the circle. This allows us to consider the circular motion as a combination of two movements: **Va** is the speed of the car moving translationally in the horizontal direction, and **Vb** is the speed of the car's rotational motion, or angular velocity. Building on the formulas for pose calculation, we can derive the following equations:

$$\begin{aligned} Vx &= -Va * \sin 90 \\ Vy &= Va * \cos 90 \end{aligned}$$

$$\begin{aligned} v1 &= -Vx + Vb \\ v2 &= -Vy + Vb \\ v3 &= Vx + Vb \\ v4 &= Vy + Vb \end{aligned}$$

Using this formula, we can enable the omni-wheeled car to perform circular motion.

Around Mode Kinematic Analysis

Assuming the radius of the circular motion is R, the formula V = W * R tells us that when the value of the radius R remains constant, the ratio between V and W also remains constant. Substituting into the formula, we get:

$$R = Va \div Vb$$

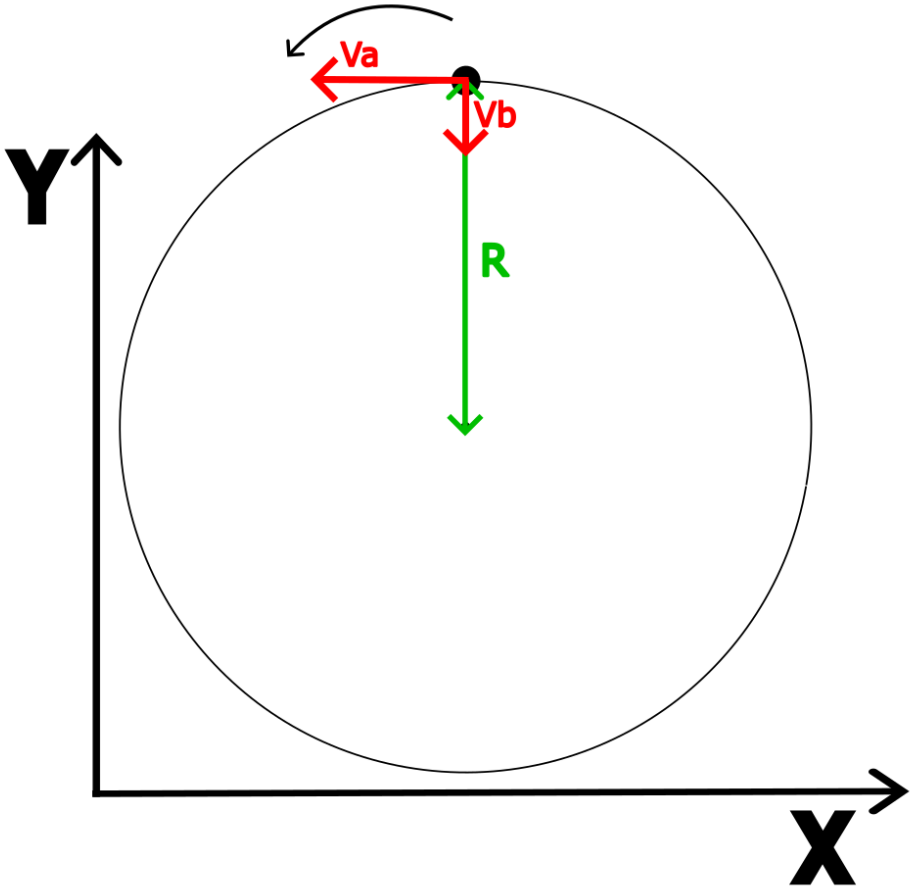
You only need to set one of the values for Va or Vb to determine the radius of the circular motion trajectory. The overall formula is:

$$Vb = R \div Va$$

$$Vx = -Va * \sin 90$$
$$Vy = Va * \cos 90$$

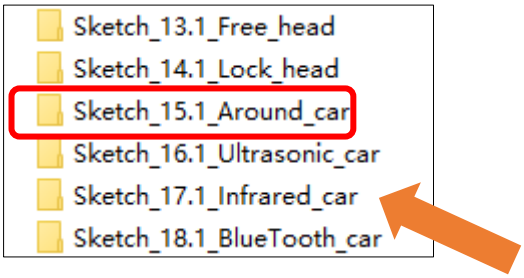
$$v1 = -Vx + Vb$$
$$v2 = -Vy + Vb$$
$$v3 = Vx + Vb$$
$$v4 = Vy + Vb$$

Please note: R and Va in the formula are the values that need to be set, and V1, V2, V3, V4 are the linear speeds of the four wheels.



Sketch

Open "**Sketch_15.1_ Around_car**" folder in **"Freenove_Omni_Wheel_Car_Kit_for_Raspberry_Pi_Pico\Four-Wheel\ \Sketches"** and then double-click **"Sketch_15.1_ Around_car.ino".**



Code

Sketch_15.1_ Around_car.ino

```
bool timer_1ms_control(struct repeating_timer *t) {
  if (millis() % 5 == 0) {
    Getencoder_Data();
    speed_add();
  }
  if (millis() % 15 == 0) {
    speed_calculate();       // Find the average speed and filter the burrs
    Circle_Control(2,30,clockwise);
  }
  return true;
}
```

Motor.cpp

```
void Circle_Control(float r, float V, float location)
{
  if(location == clockwise)
   {
     angle_circle = V / r;
     vx = - V * sin (90 * (PI / 180));
     vy = V * cos (90 * (PI / 180));
     v1 = -vx + angle_circle;
     v2 = -vy + angle_circle;
     v3 = vx + angle_circle;
     v4 = vy + angle_circle;
   }
   if(location == anticlockwise)
   {
     angle_circle = V / r;
```

```
16        vx = - V * sin (-90 * (PI / 180));
17        vy = V * cos (-90 * (PI / 180));
18        v1 = -vx + angle_circle;
19        v2 = -vy + angle_circle;
20        v3 = vx + angle_circle;
21        v4 = vy + angle_circle;
22      }
23      // Speed PID control
24      pid_v1 = Speed1_PID(v1,speed1);
25      pid_v2 = Speed2_PID(v2,speed2);
26      pid_v3 = Speed3_PID(v3,speed3);
27      pid_v4 = Speed4_PID(v4,speed4);
28
29      Motor_direction();// Motor limiting and motor output orientation
30  }
```

After compiling and uploading the code, place the omni-wheeled car on the ground and power it on. The omni-wheeled car will perform a uniform circular motion with a radius of 20 cm in a clockwise direction.

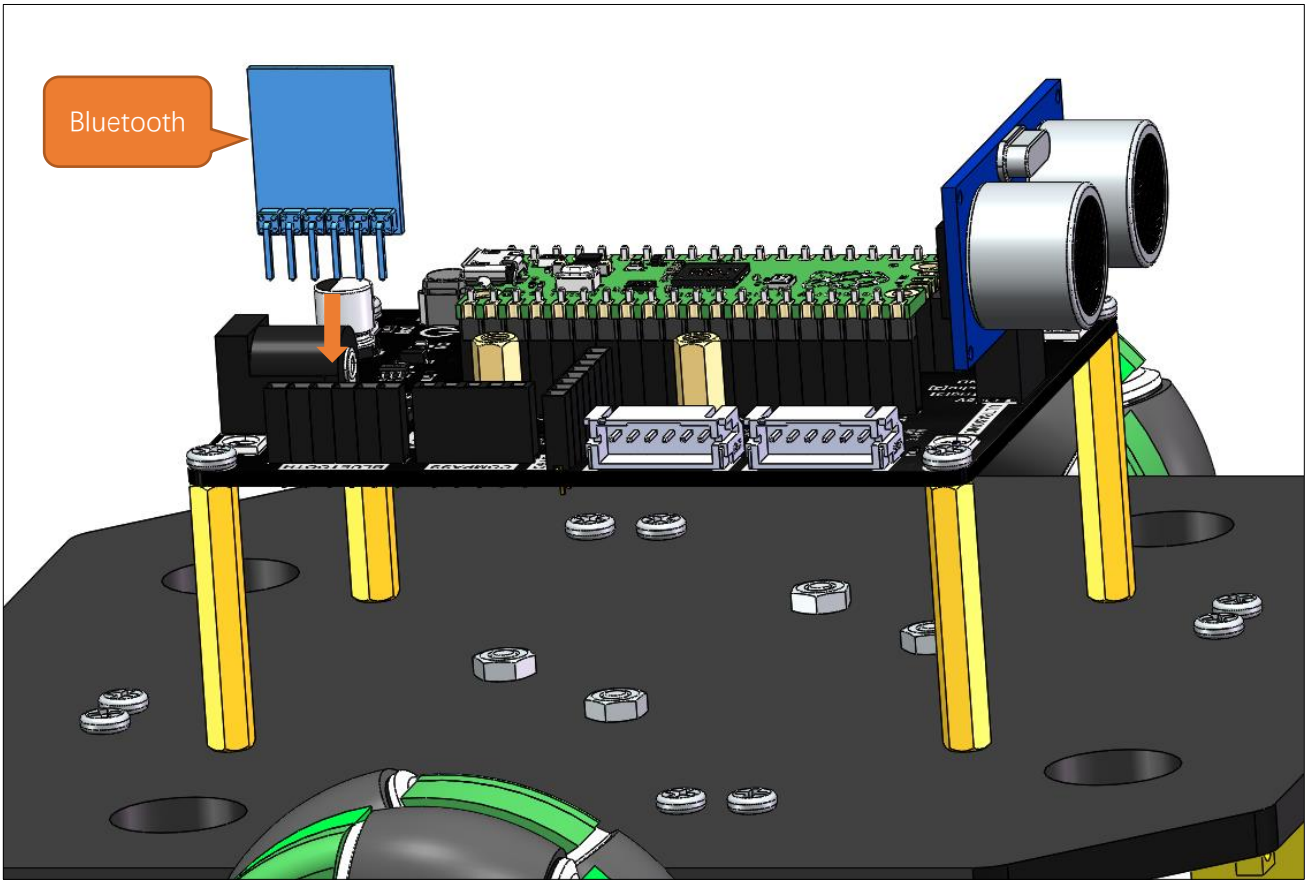
Code Explanation

Determine the input parameter; when the parameter is "clockwise", the car circles clockwise, and when the parameter is "anticlockwise", the car rotates counterclockwise.

```
1   if(location == clockwise)
2   {
3      angle_circle = V / r;
4      vx = - V * sin (90 * (PI / 180));
5      vy = V * cos (90 * (PI / 180));
6      v1 = -vx + angle_circle;
7      v2 = -vy + angle_circle;
8      v3 = vx + angle_circle;
9      v4 = vy + angle_circle;
10  }
11  if(location == anticlockwise)
12  {
13     angle_circle = V / r;
14     vx = - V * sin (-90 * (PI / 180));
15     vy = V * cos (-90 * (PI / 180));
16     v1 = -vx + angle_circle;
17     v2 = -vy + angle_circle;
18     v3 = vx + angle_circle;
19     v4 = vy + angle_circle;
20  }
```

Circuit

Plug the Bluetooth module to the assembled car. For detailed assembly process of the car, please refer to Chapter 1

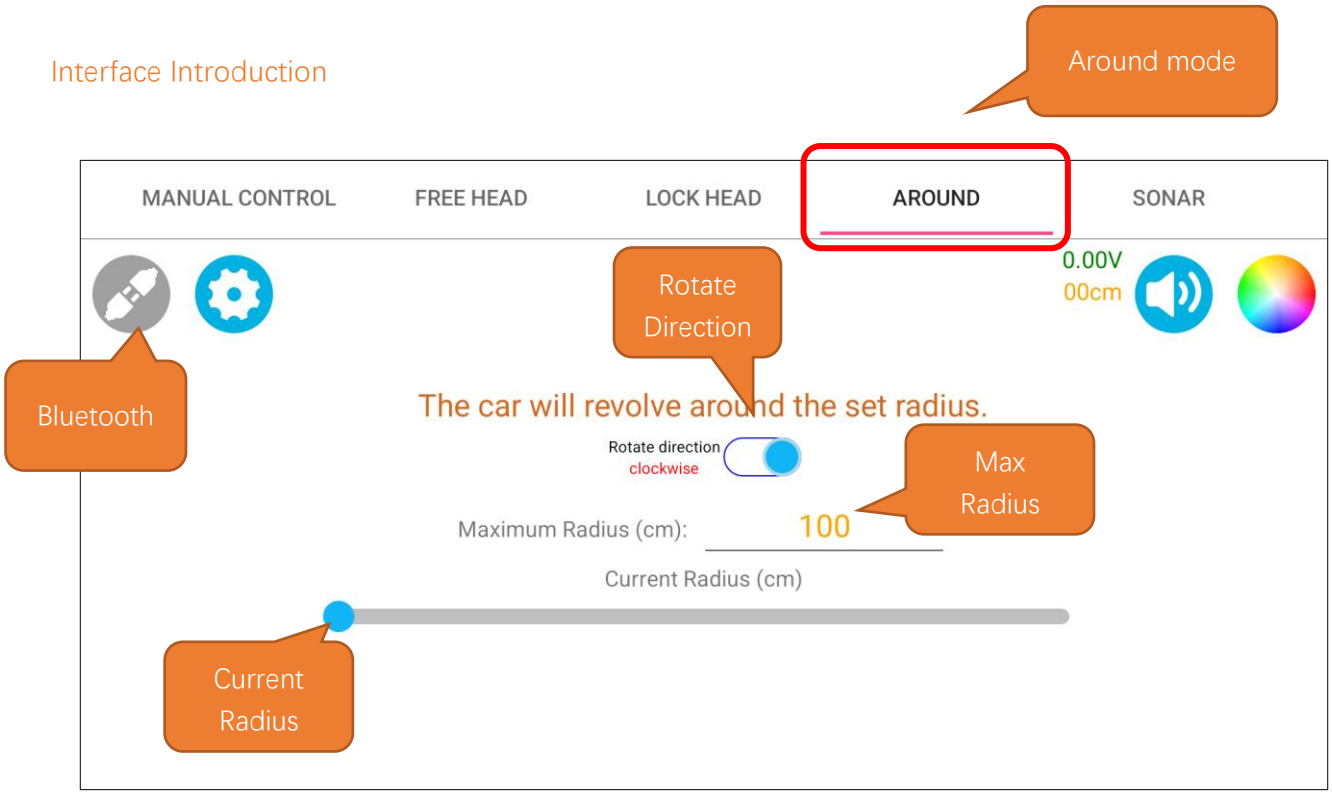


About Around Mode

For the connection of Bluetooth device, please refer to Freenove APP-Connection

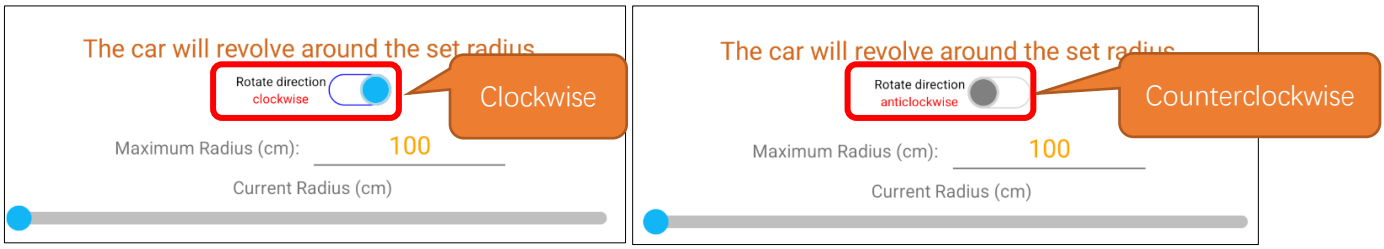
Please note that the example code for this part does not involve Bluetooth. To control the car via Bluetooth, please open"**Sketch_18.1_Bluetooth_car**" folder in **"Four-Wheel\Sketches"** and then double-click **"Sketch_18.1_Bluetooth_car.ino".**

Interface Introduction

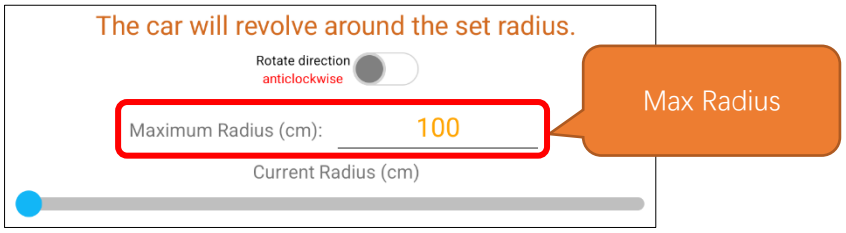


Operation Description

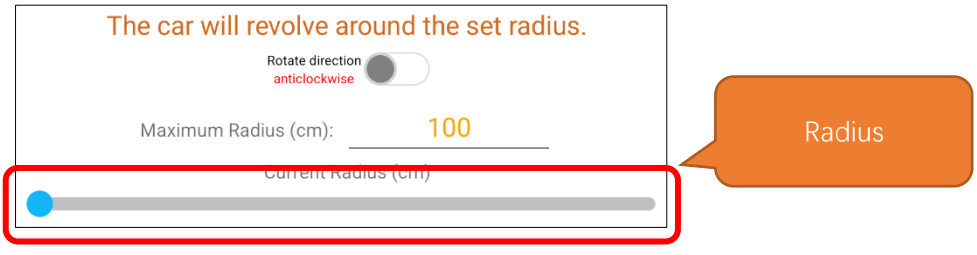
This button is to change the rotation direction. When it is enabled, the car rotates clockwise, and disabled, it rotates counterclockwise.



Input the maximum radius. If the number set is below 100, it will be 100.



Slide the bar to set the car's rotation radius. The range is from 0 to the max radius set.



Chapter 18 Sonar Mode

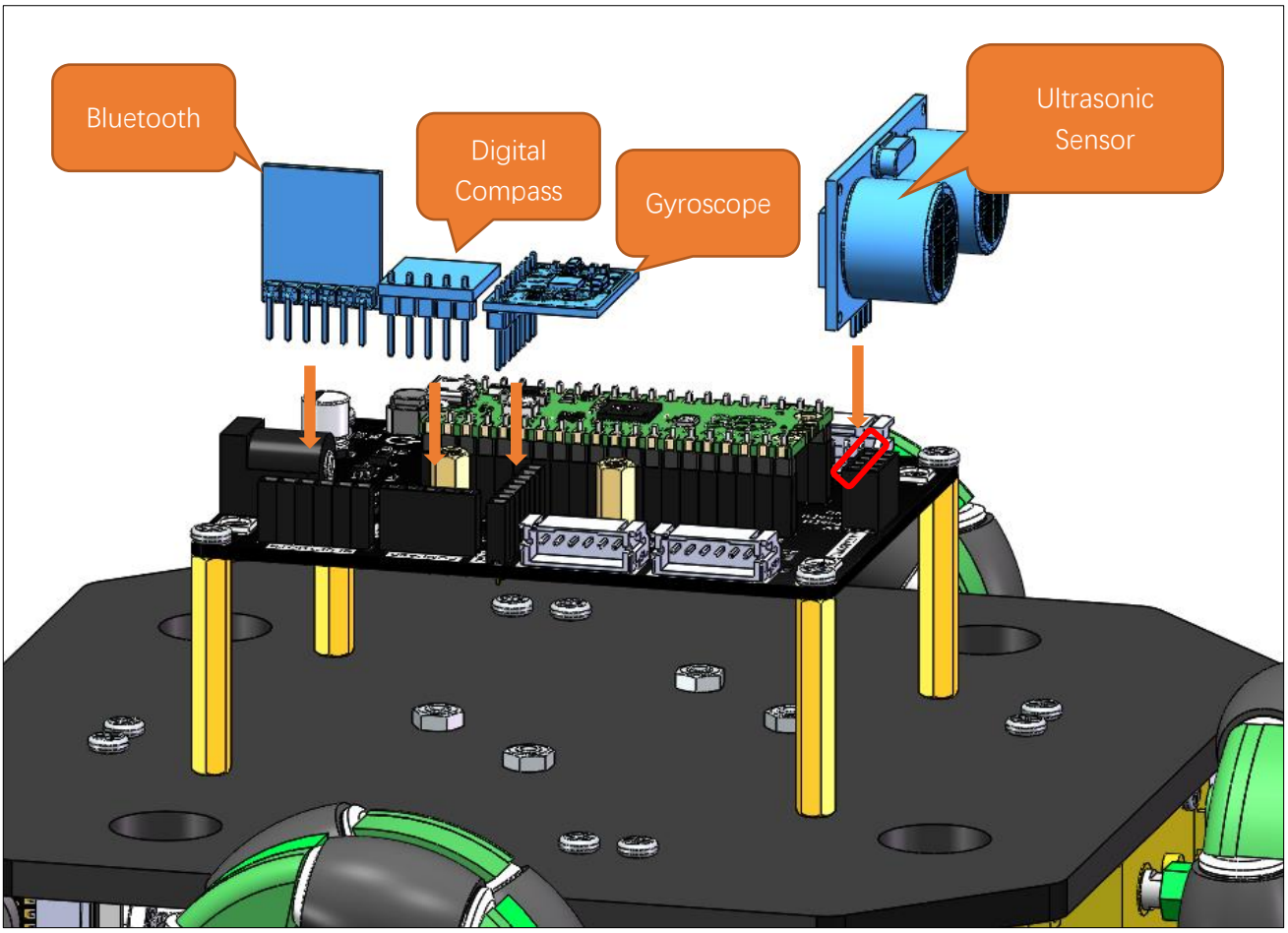
In the previous chapters, we have learned how to use an ultrasonic module to read distances. In this chapter, we will attempt to implement the obstacle-avoidance function for this car. When the mode is activated the car will move forward at a fixed speed. When it encounters an obstacle, it will read the distance from both sides and turn towards the more open direction to continue moving forward.

Sonar Mode Summary

In ultrasonic obstacle avoidance mode, the car will move forward at a fixed speed when there are no obstacles ahead. When it encounters an obstacle, it will rotate towards both sides and read the distance data, then turn towards the more open side to continue moving forward.

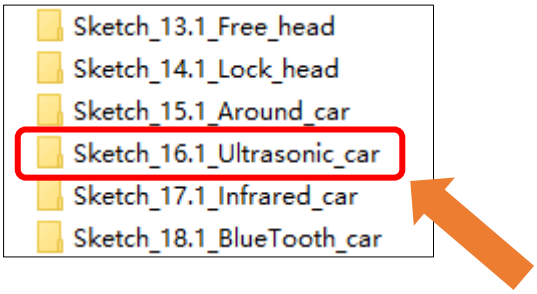
Circuit

In this section, we use the assembled car with ultrasonic module, gyroscope, compass module, and Bluetooth module connected. For detailed process of car assembly, please refer to Chapter 1



Sketch

Open "**Sketch_16.1_Ultrasonic_car**" folder in **"Freenove_Omni_Wheel_Car_Kit_for_Raspberry_Pi_Pico\Four-Wheel\ \Sketches"** and then double-click **"Sketch_16.1_Ultrasonic_car.ino".**



Code

Sketch_16.1_Ultrasonic_car.ino

```
void Ultrasonic_Run(int speed_v, int angle_a, int turn_speed)

{
   angle_v = Turn_PID_Realize(angle_a,angle);
   if(angle_v > turn_speed) angle_v = turn_speed;else if(angle_v < - turn_speed) angle_v = -
turn_speed;

   v1 = angle_v;
   v2 = speed_v + angle_v;
   v3 = angle_v;
   v4 = -speed_v + angle_v;

   pid_v1 = Speed1_PID(v1,speed1);
   pid_v2 = Speed2_PID(v2,speed2);
   pid_v3 = Speed3_PID(v3,speed3);
   pid_v4 = Speed4_PID(v4,speed4);
   Motor_direction();
}

void Ultrasonic_control(int speed, int turn_time, int distance)
{
   if((millis() - Ultrasonic_distance_time) > 20)
   {
     Ultrasonic_distance = Read_Distance();
     Ultrasonic_distance_time = millis();
   }
   if(Ultrasonic_mutex == 0)
```

```
    {
      if(Ultrasonic_distance > distance)// The distance is greater than 20cm
      {
        Ultrasonic_speed = speed;
        Ultrasonic_angle = angle;
      }
      else{
        Ultrasonic_mutex = 1;
        Ultrasonic_speed = 0;
      }
    }
    else{
      if(Ultrasonic_state == 0)// turn left
      {
        Ultrasonic_angle = Ultrasonic_angle - 90;
        Ultrasonic_state = 1;
        Ultrasonic_time = millis();// Obtain the system running time
      }else if((millis() - Ultrasonic_time) > turn_time && Ultrasonic_state == 1)// Turn 90
degrees to the left
      {
        Ultrasonic_compare[0] = Read_Distance();
        Ultrasonic_state = 2;
        Ultrasonic_time = millis();
      }else if((millis() - Ultrasonic_time) > turn_time && Ultrasonic_state == 2)// Turn right
back to the center
      {
        Ultrasonic_angle = Ultrasonic_angle + 90;
        Ultrasonic_state = 3;
        Ultrasonic_time = millis();
      }else if((millis() - Ultrasonic_time) > turn_time && Ultrasonic_state == 3)// stop
      {
        Ultrasonic_state = 4;
        Ultrasonic_time = millis();
      }else if((millis() - Ultrasonic_time) > turn_time && Ultrasonic_state == 4)// Turn 90
degrees to the right
      {
        Ultrasonic_angle = Ultrasonic_angle + 90;

        Ultrasonic_state = 5;
        Ultrasonic_time = millis();
      }
      else if((millis() - Ultrasonic_time) > turn_time && Ultrasonic_state == 5)// Turn 90
degrees to the right
      {
```

```
72          Ultrasonic_compare[1] = Read_Distance();
73          Ultrasonic_state = 6;
74          Ultrasonic_time = millis();
75        }else if((millis() - Ultrasonic_time) > turn_time && Ultrasonic_state == 6)
76        {
77          Ultrasonic_angle = Ultrasonic_angle - 90;// Go back to the middle
78          Ultrasonic_state = 7;
79          Ultrasonic_time = millis();
80        }else if((millis() - Ultrasonic_time) > turn_time && Ultrasonic_state == 7)
81        {
82          Ultrasonic_state = 8;
83          Ultrasonic_time = millis();
84        }else if((millis() - Ultrasonic_time) > turn_time && Ultrasonic_state == 8)
85        {
86          if(Ultrasonic_compare[0] > Ultrasonic_compare[1])
87          {
88            Ultrasonic_angle = Ultrasonic_angle - 90;
89          }else
90          {
91            Ultrasonic_angle = Ultrasonic_angle + 90;
92          }
93          Ultrasonic_state = 9;
94          Ultrasonic_time = millis();
95        }
96        else if((millis() - Ultrasonic_time) > turn_time && Ultrasonic_state == 9)
97        {
98          Ultrasonic_speed = speed;
99          Ultrasonic_state = 0;
100         Ultrasonic_mutex = 0;
101       }
102     }
103 }
```

Ultrasonic.cpp

```
1  int Read_Distance()
2  {
3    digitalWrite(Trig,LOW);
4    delayMicroseconds(2);
5    digitalWrite(Trig,HIGH);
6    delayMicroseconds(10);
7    digitalWrite(Trig,LOW);
8
9    Ultrasonic_distance = pulseIn(Echo,HIGH) / 58.8;
10   return Ultrasonic_distance;
11 }
```

Code Explanation

Use a timer interrupt to control the car to check the encoder value every 5ms, detect the angle every 10ms, and measure the speed and control its movement according to the encoder value every 15ms.

```
bool timer_1ms_control(struct repeating_timer *t) {
   if (millis() % 5 == 0) {
     Getencoder_Data();
     speed_add();
   }
   if(millis() % 10 == 0)
   {
     IMU_state = 1;
   }
   if (millis() % 15 == 0) {
     speed_calculate();    // Find the average speed and filter the burrs
     Ultrasonic_Run(Ultrasonic_speed, Ultrasonic_angle, 50);
   }
   return true;
}
```

Call the function Ultrasonic_Run () to control the car's movement.

```
void Ultrasonic_Run(int speed_v, int angle_a, int turn_speed)
{
   angle_v = Turn_PID_Realize(angle_a,angle);
   if(angle_v > turn_speed) angle_v = turn_speed;else if(angle_v < - turn_speed) angle_v = -
turn_speed;

   vx =  - speed_v * sin ( 0 * PI / 180 );
   vy =    speed_v * cos ( 0 * PI / 180 );

   v1 = -vx + angle_v;
   v2 = 0.5 * vx - sqrt(3) / 2 * vy + angle_v;
   v3 = 0.5 * vx + sqrt(3) / 2 * vy + angle_v;

   pid_v1 = Speed1_PID(v1,speed1);
   pid_v2 = Speed2_PID(v2,speed2);
   pid_v3 = Speed3_PID(v3,speed3);

   Motor_direction();
}
```

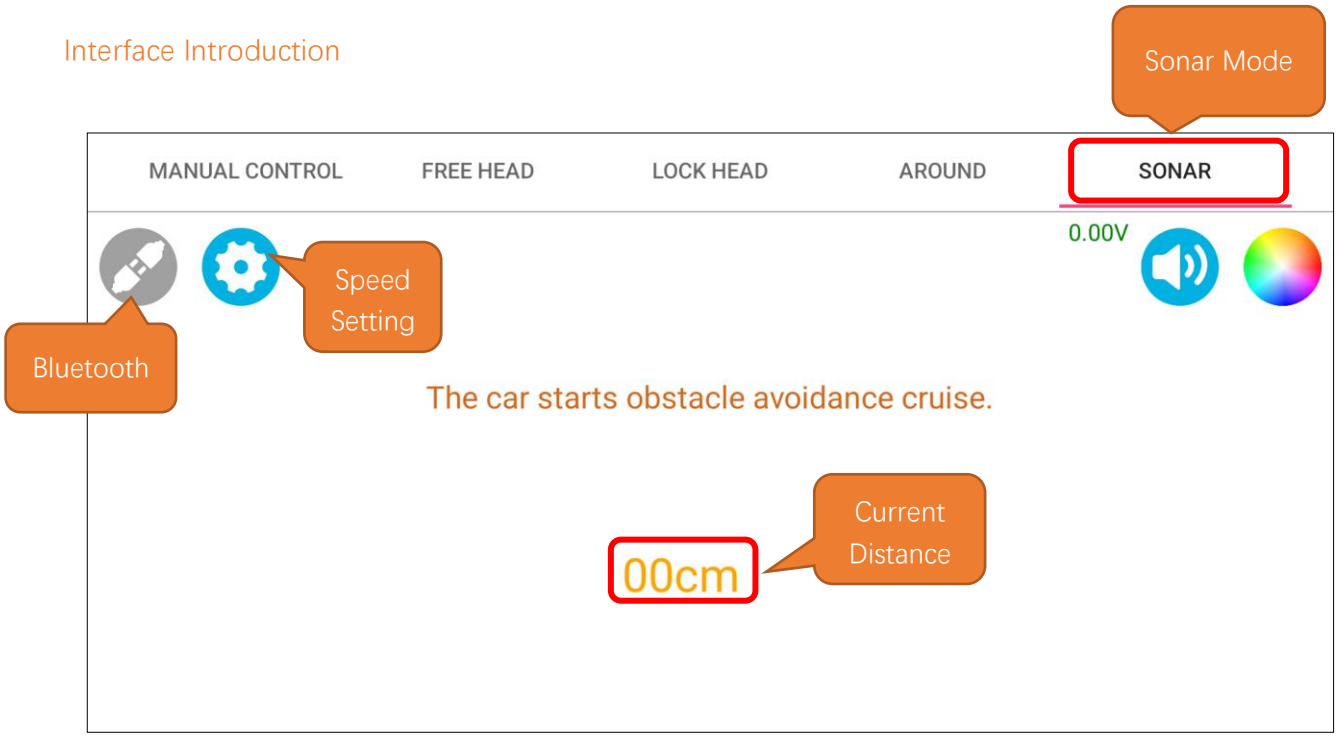
Reference

`void Ultrasonic_Run(int speed_v, int angle_a, int turn_speed)`
The function is designed to control the movement of the robot car. It calculates the speed of each motor by obtaining the target speed, rotation angle, and turning speed as parameters, and applies a PID control algorithm to achieve precise motion control. Parameters: speed_v: The target speed of the car angle_a: The target rotation angle of the car turn_speed: The target turning speed of the car
`void Ultrasonic_control(int speed, int turn_time, int distance)`
This function is designed to control the robot car's obstacle avoidance. It determines the action logic of the robot by obtaining speed, turning interval, and obstacle avoidance distance as parameters. Parameters: speed: The cruising speed of the robot, with a parameter range from 0 to 100. turn_time: The time interval between two turns of the car, in milliseconds. distance: The minimum safe distance between the car and an obstacle, in centimeters. When the distance is below this value, the car will initiate obstacle avoidance,.

About Sonar Mode

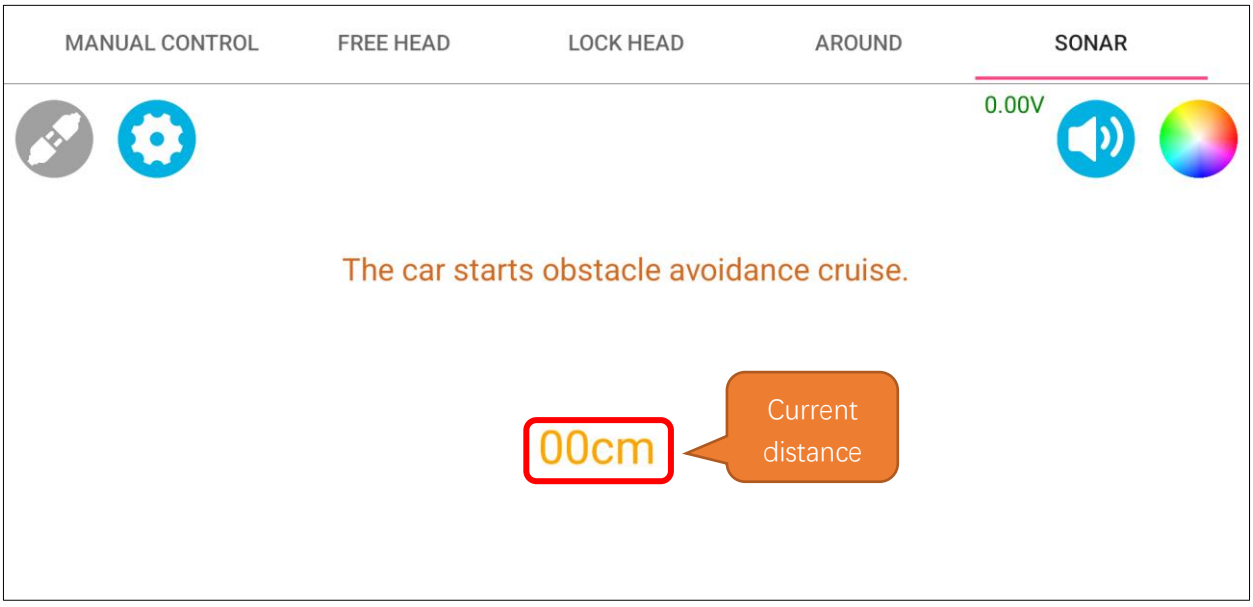
For the connection of Bluetooth device, please refer to Freenove APP-Connection
Please note that the example code in this section does not involve Bluetooth. To control the car via Bluetooth, please upload Sketch_18.1_Bluetooth_car.

Interface Introduction



Operation Description

Display the real-time distance between the car and the obstacles ahead in centimeters.

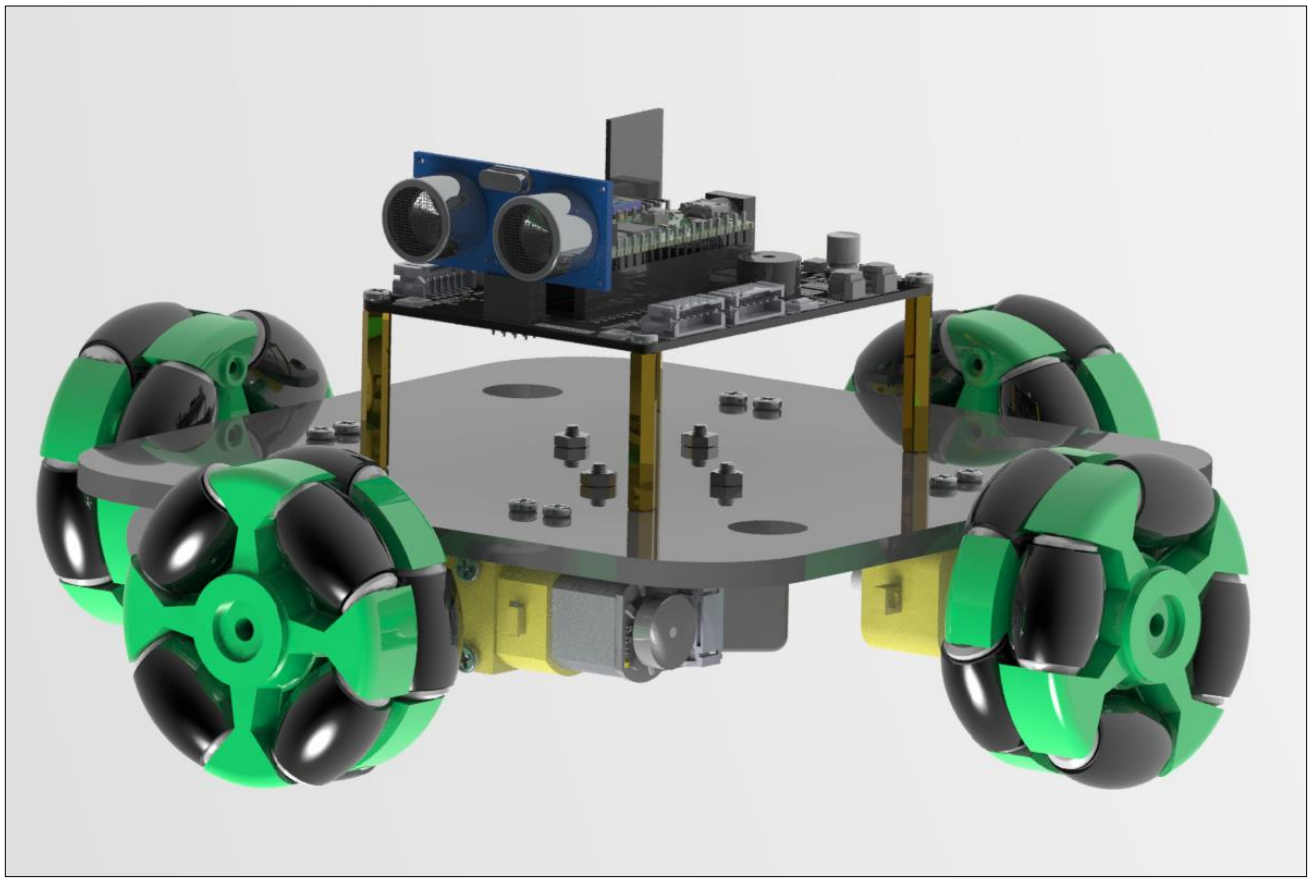


Chapter 19 Infrared Control

In the previous chapters, we learned how to use infrared modules. In this chapter, we will learn how to control the car's movement and change modes using an infrared remote controller.

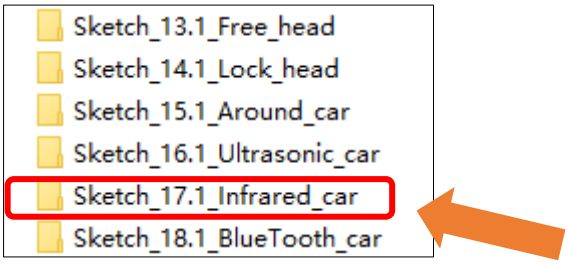
Circuit

For detailed assembly process of the car, please refer to Chapter 1



Sketch

Open "**Sketch_17.1_Infrared_car**" folder in **"Freenove_Omni_Wheel_Car_Kit_for_Raspberry_Pi_Pico\Four-Wheel\ \Sketches"** and then double-click **"Sketch_17.1_Infrared_car.ino".**



Code

Sketch_17.1_Infrared_car.ino

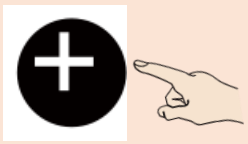
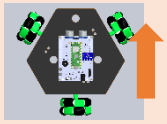
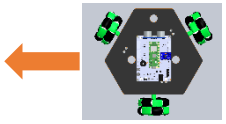
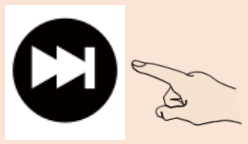
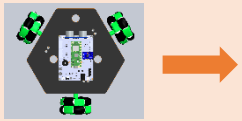
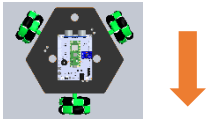
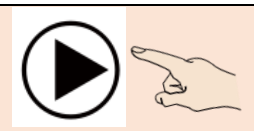
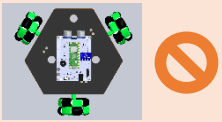
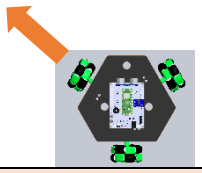
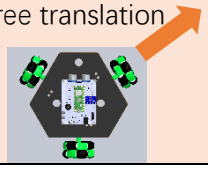
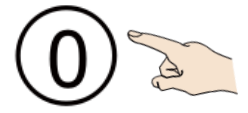
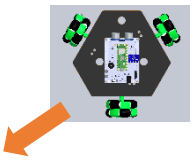
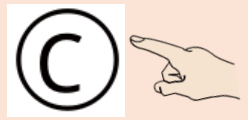
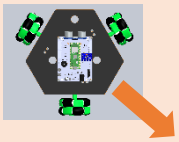
```
bool timer_1ms_control(struct repeating_timer *t) {
   if (millis() % 5 == 0) {
     Getencoder_Data();
     speed_add();
   }
   if (millis() % 15 == 0) {
     speed_calculate();  // Find the average speed and filter the burrs
     car_run(IR_v ,IR_a,IR_angle_v);
   }
   return true;
}

void setup() {
   // put your setup code here, to run once:
   // Set the serial port baud rate to 9600
   Serial.begin(9600);

    // Encoder initialization
   Encoder_Init();

   // Timer interrupt setting
   ITimer0.attachInterruptInterval(TIMER0_INTERVAL_MS, timer_1ms_control);

   Motor_init();  // Motor initialization

   IR_Init(22);   // Infrared pin initialization
}
```

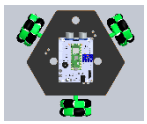
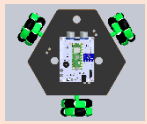
```
28
29  void loop() {
30      IR_Receive();
31      IR_Task();
32  }
33
34  void speed_add() {
35      speed1val += speed1;
36      speed2val += speed2;
37      speed3val += speed3;
38      speed4val += speed4;
39  }
40
41  void speed_calculate() {
42      speed1 = map(speed1val / 3,0,48,0,100);
43      speed2 = map(speed2val / 3,0,48,0,100);
44      speed3 = map(speed3val / 3,0,48,0,100);
45      speed4 = map(speed4val / 3,0,48,0,100);
46
47      speed1val = 0;
48      speed2val = 0;
49      speed3val = 0;
50      speed4val = 0;
51  }
```

Motor.cpp

```
1   void car_run(int speed_v,int speed_a,int angle_v)
2   {
3       vx = -speed_v * sin ( speed_a * PI / 180 );
4       vy =  speed_v * cos ( speed_a * PI / 180 );
5
6       v1 = -vx + angle_v;
7       v2 = -vy + angle_v;
8       v3 =  vy + angle_v;
9       v4 =  vx + angle_v;
10
11      pid_v1 = Speed1_PID(v1,speed1);
12      pid_v2 = Speed2_PID(v2,speed2);
13      pid_v3 = Speed3_PID(v3,speed3);
14      pid_v4 = Speed4_PID(v4,speed4);
15
16      Motor_direction();// Motor limiting and motor output redirection
17  }
```

After downloading the code, place the car on the ground and turn ON the power switch. The keys and their corresponding values and function are as shown in the table below:

Keys	Function	Key Value
	Forward 	FF02FD
	Left translation 	FFE01F
	Right translation 	FF906F
	Backward 	FF9867
	Stop 	FFA857
	Left 45-degree translation 	FF22DD
	Right 45-degree translation 	FFC23D
	Left 135-degree translation 	FF6897
	Right 135-degree translation 	FFB04F

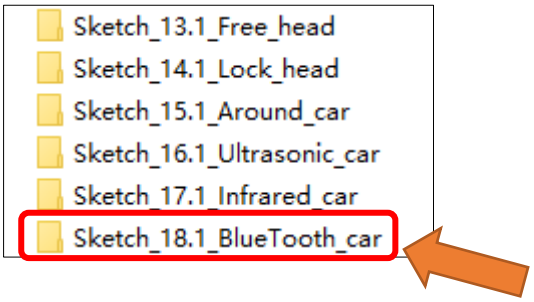
	Turn Left 	FFA25D
	Turn Right 	FFE214D

Chapter 20 Bluetooth Car

Next, let us integrate the functionalities and modules learned in previous chapters using a Bluetooth module, so as to control most features of the car with the phone APP.

Sketch

Open "Sketch_20.1_Bluetooth_car" folder in **"Freenove_Omni_Wheel_Car_Kit_for_Raspberry_Pi_Pico\Four-Wheel\ \Sketches"** and then double-click **"Sketch_20.1_Bluetooth_car.ino".**



Code

Sketch_18.1_Bluetooth_car.ino

```
#include "Bluetooth.h"
#include "User_TIM.h"

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);                 // Set the serial port baud rate to 9600
  BlueTooth_Init();                   // Encoder initialization
  Encoder_Init();
  IMU_Init();
  Buzzer_Setup();                     // Buzzer initialization
  WS2812_Setup();                     // WS2812 initialization
  Motor_init();                       // Motor initialization
  Ultrasonic_init();                  // Ultrasonic initialization
  IR_Init();                          // IR initialization
  uint32_t current_millis = millis();
  while (millis() - current_millis < 500) {
    IMU_GetData();
  }
  tone(PIN_BUZZER, 2000, 200);
  tim_init();
```

```
21  }
22
23  void loop() {
24      IR_Task();
25      WS2812_Show(ws2812_task_mode);
26      Bluetooth_Control();
27  }
```

After compiling and uploading the sketch, you can control the car's modes and functionalities with the phone app via Bluetooth. To install the APP, please refer to Freenove app

If you need any support, please feel free to contact us via: **support@freenove.com**

What's next?

Thank you again for choosing Freenove products.

THANK YOU for participating in this learning experience!

We have reached the end of this Tutorial. If you find errors, omissions, or you have suggestions and/or questions about the Tutorial or component contents of this Kit, please feel free to contact us:
support@freenove.com

We will make every effort to make changes and correct errors as soon as feasibly possible and publish a revised version.

If you want to learn more about Arduino, ESP32, Raspberry Pi Pico W, Raspberry Pi, Smart Cars, Robotics and other interesting products in science and technology, please continue to visit our website. We will continue to launch fun, cost effective, innovative and exciting products.

Thank you again for choosing Freenove products.