Welcome

Thank you for choosing Freenove products!

About Battery

First, read the document **About_Battery.pdf** in the unzipped folder.
If you have not downloaded the zip file, please download and unzip it via the link below.
https://github.com/Freenove/Freenove_Robot_Dog_Kit_for_ESP32/archive/master.zip
https://github.com/Freenove/Freenove_ESP32_Robot_Dog_SDK_for_Arduino_IDE/archive/master.zip

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 and micro: bit
- 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

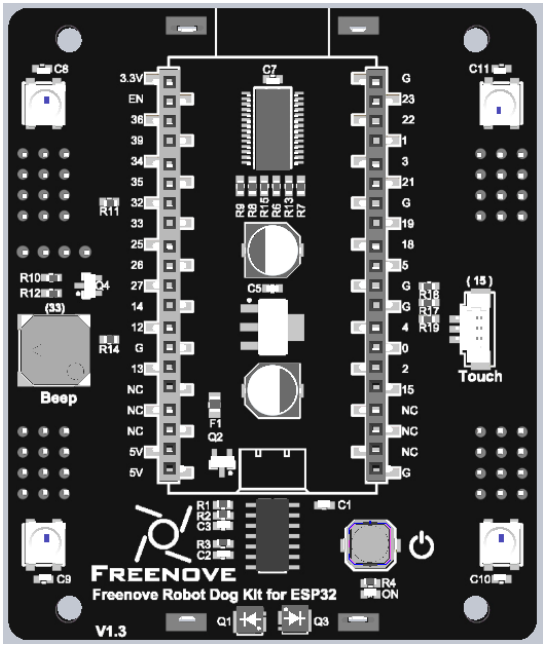
Contents

List

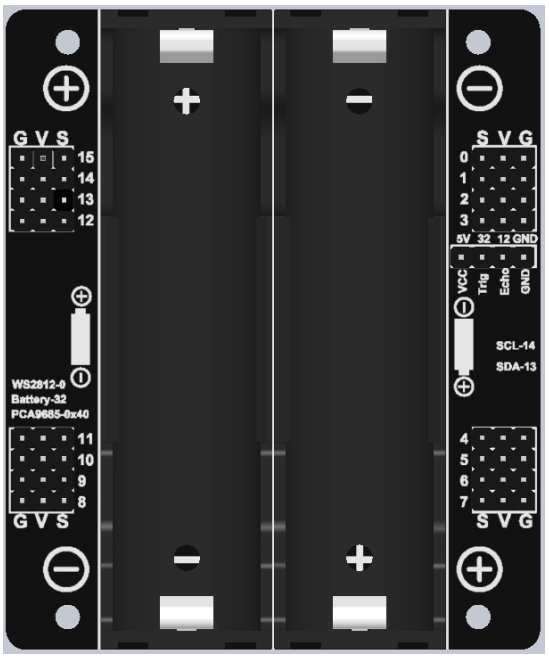
If you have any concerns, please feel free to contact us at support@freenove.com

Robot Expansion Board for ESP32

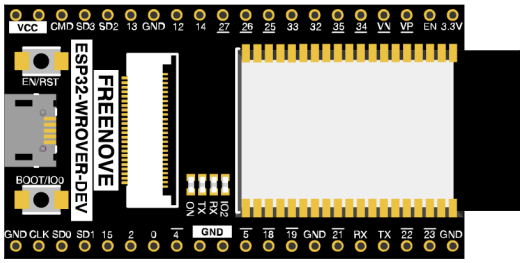
Top



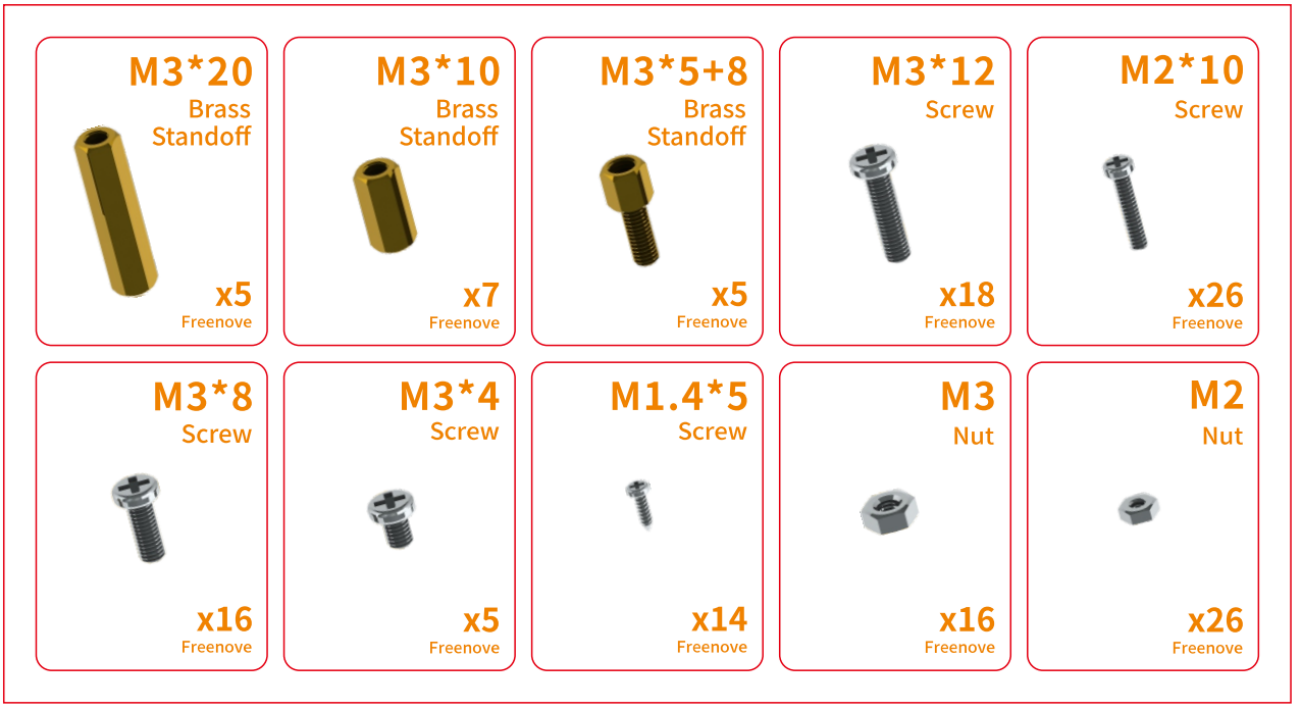
Bottom



ESP32

ESP32	OV2640
	

Machinery

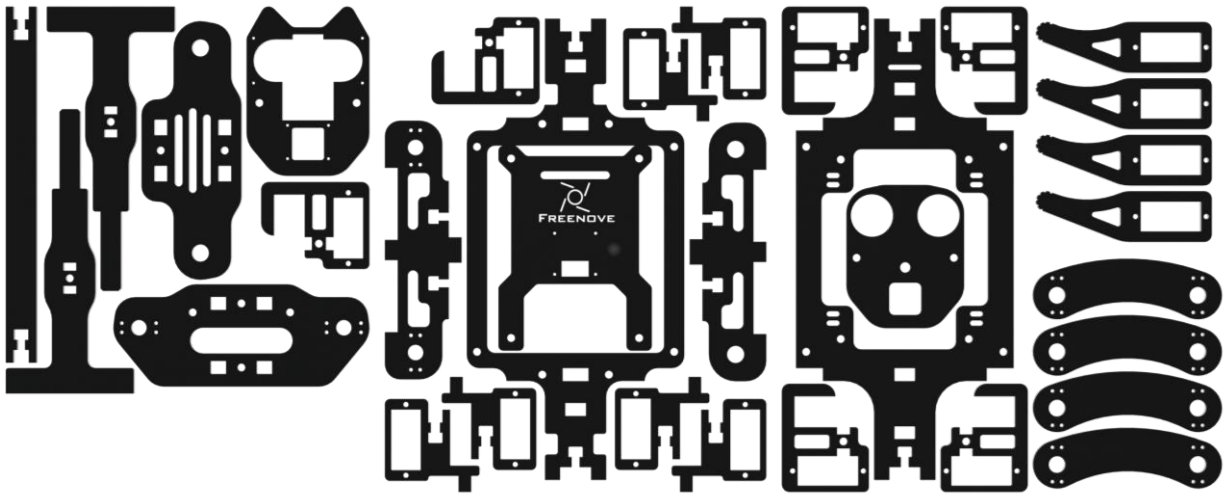


Transmission

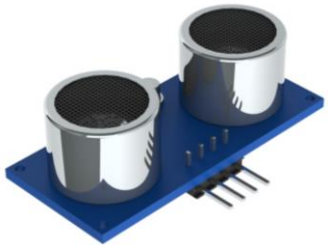
Servo package x12

Acrylic

Acrylic x1



Electronic

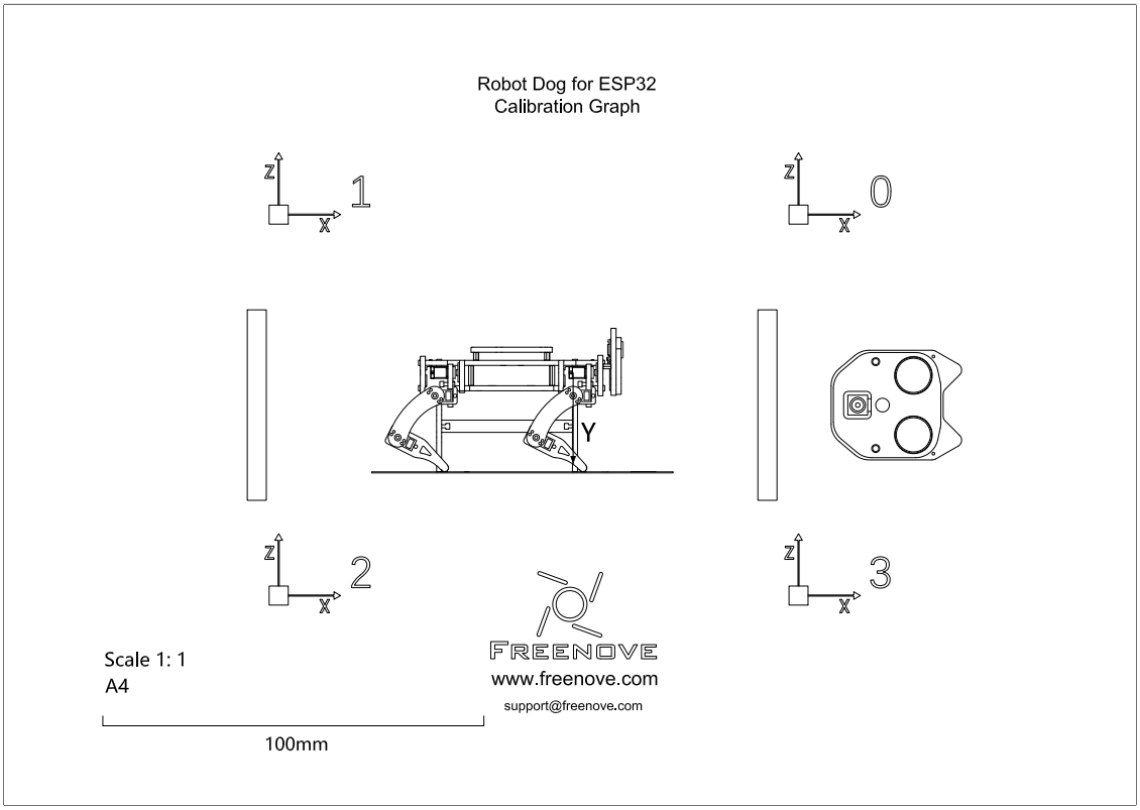
Extension board for camera 	HC-SR04 ultrasonic module 	Touch module 

Tools

Cross screwdriver (3mm) x1 Cross screwdriver (2mm) x1 	Cable tidy x80cm 
Pry bar 	USB cable 
FPC camera cable 	
3P LED cable 	4P jumper wire 

Calibration

Calibration graph



Required but NOT Included Parts

2 X 18650 flat top batteries with **continuous discharge current ≥ 10A**

It is not easy to find proper batteries on Amazon. **Search 18650 3.7V high drain on eBay** or other websites.

Preface

Welcome to use Freenove Robot Dog Kit for ESP32. Following this tutorial, you can make a very cool robot dog with many functions.

This kit is based on ESP32, a popular control panel, so you can exchange your experience and design ideas with many enthusiasts all over the world. The parts in this kit include all electronic components, modules, and mechanical components required for making the robot dog. And all of them are packaged individually. There are detailed assembly and commissioning instructions in this book.

If you encounter any problems, please feel free to contact us for quick and free technical support.

support@freenove.com

This book aims to help enthusiasts assemble the robot dog and download related codes. You can read and download the codes via the link below:
https://github.com/Freenove/Freenove_ESP32_Dog_Firmware

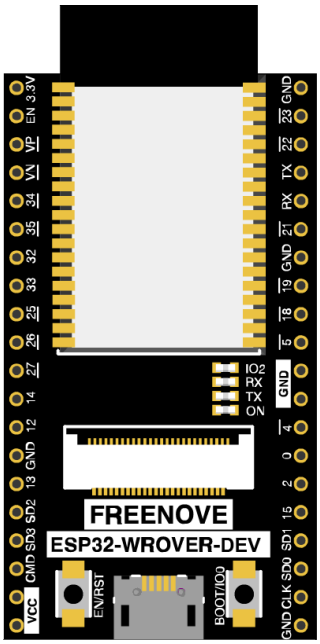
You can refer to another esp32 kit designed for starters: Freenove_Ultimate_Starter_Kit_for_ESP32.
https://www.freenove.com/store.html

ESP32

ESP32-Wrover comes with two different antenna packages, PCB (on-board) antenna and IPEX™ antenna. The PCB on-board antenna is an integrated antenna in the chip module itself, so it is convenient to carry and design. The IPEX™ antenna is a metal antenna connector, derived from the integrated antenna of the chip module itself, which is used to enhance the signal of the ESP32 module.

PCB on-board antenna	IPEX antenna
	

In this tutorial, the ESP32-WROVER is designed based on PCB on-board antenna packaged ESP32-WROVER module.

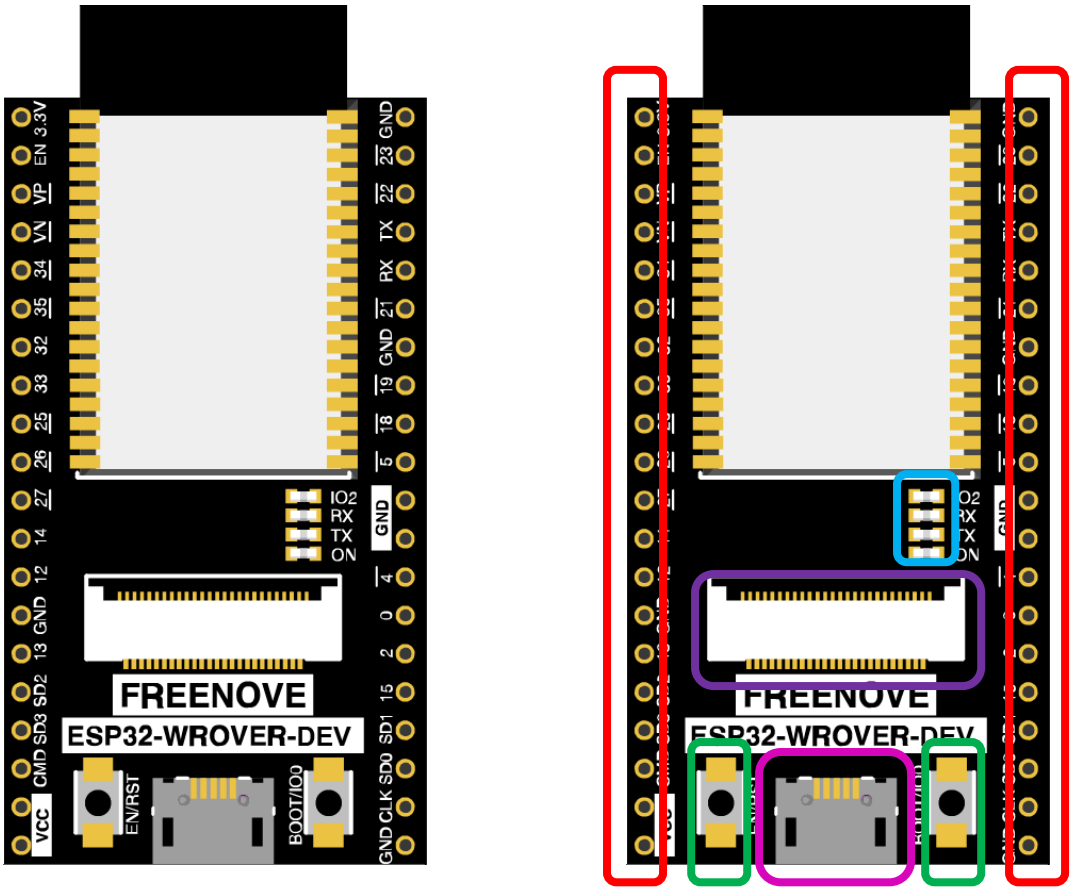


Pins Used for the Robot

To learn what each GPIO corresponds to, please refer to the following table.
The functions of the pins are allocated as follows:

Pins of ESP32	Functions	Description
GPIO36	Camera interface	CSI_Y6
GPIO39		CSI_Y7
GPIO34		CSI_Y8
GPIO35		CSI_Y9
GPIO25		CSI_VYSNC
GPIO26		SIOD
GPIO27		SIOC
GPIO4		CSI_Y2
GPIO5		CSI_Y3
GPIO18		CSI_Y4
GPIO19		CSI_Y5
GPIO21		XCLK
GPIO22		PCLK
GPIO23		HREF
GPIO13	I2C port	SDA
GPIO14		SCL
GPIO32	Battery detection / Ultrasonic-Trig port	A6 / Trig
GPIO12	Ultrasonic-Echo port	Echo
GPIO33	Buzzer port	Buzzer
GPIO15	Touch Sensor port	Touch
GPIO0	WS2812 port	WS2812
GPIO1	Serial port	TX
GPIO3		RX

The hardware interfaces of ESP32 are distributed as follows:

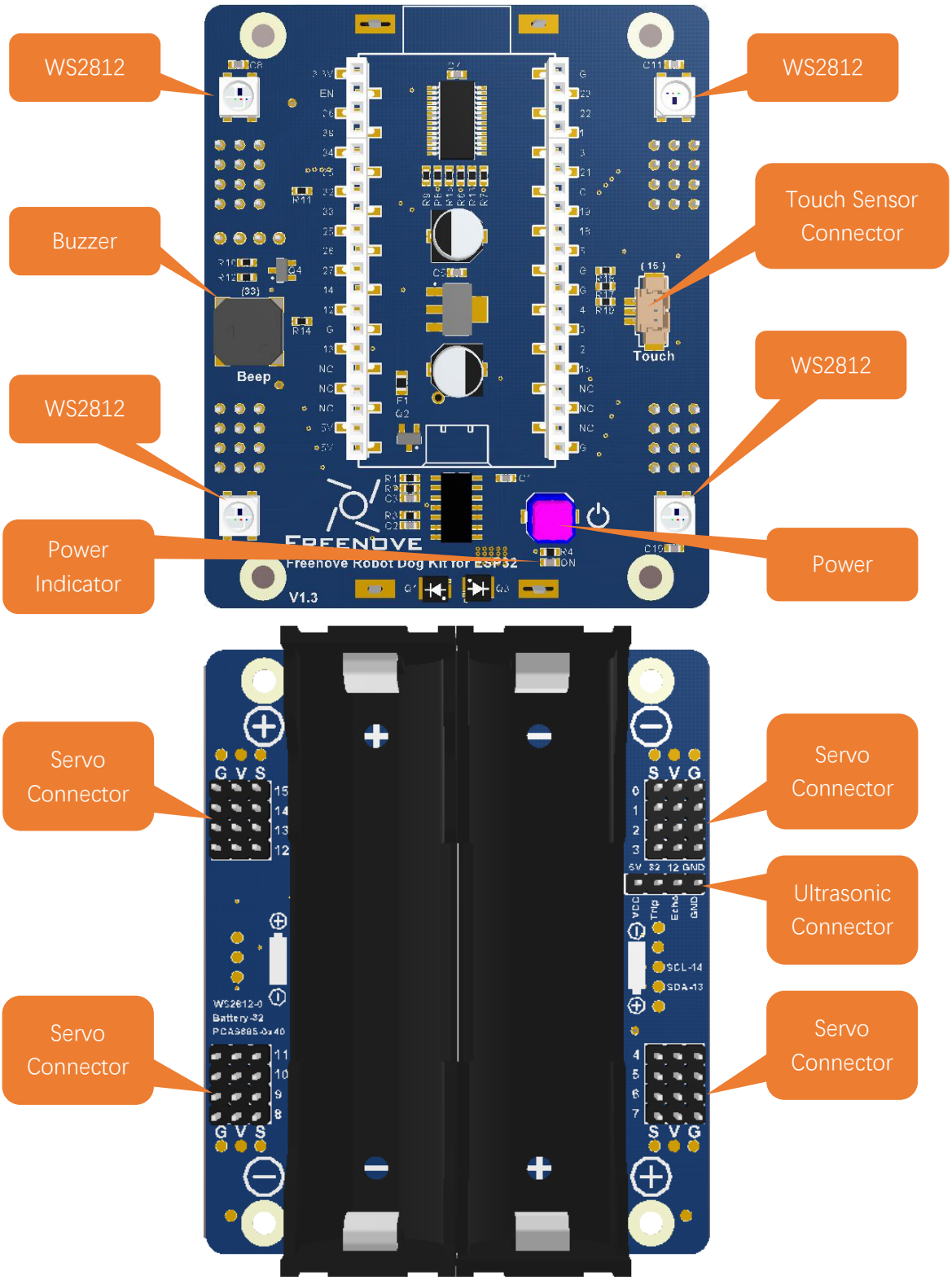


Compare the left and right images. We have boxed off the resources on the ESP32 in different colors to facilitate its understanding.

Box color	Corresponding resources introduction
	GPIO pin
	LED indicator
	Camera interface
	Reset button, Boot mode selection button
	USB port

Robot Expansion Board for ESP32

The functions of the board are as follows:



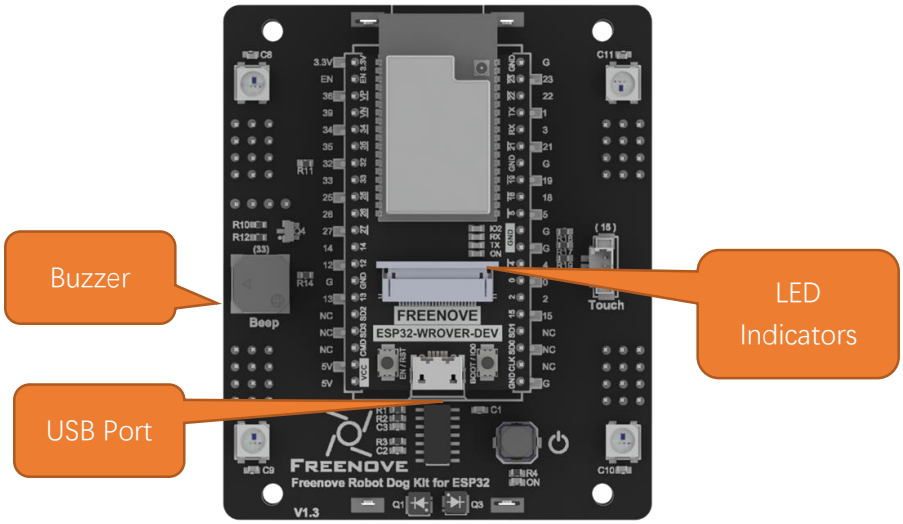
Note: Please refer to the marks on the battery holder to install batteries; otherwise, the circuit will not work.

Chapter 0 Install CH340 and Burn Firmware

Check Firmware

Please note that the firmware has been burnt by default, so generally, you do not need to burn it again.
Please follow the steps below to check whether firmware has been burnt:

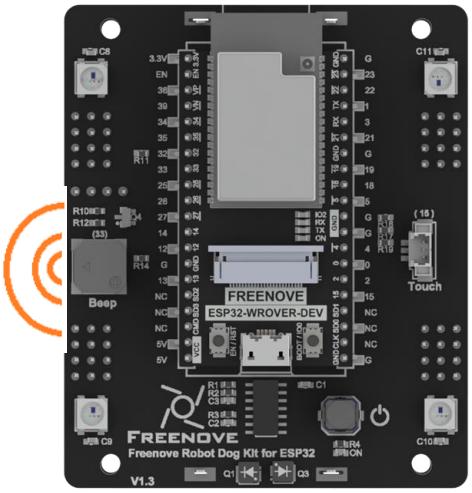
1. Plug the ESP32 to the expansion board.



Pay attention to the orientation of the ESP32 and make sure it is plugged in correctly; otherwise, it may damage the robot.

2. Connect ESP32 to your computer with a USB cable.

After connecting, you can see the yellow LED on ESP32 stay ON and the blue LED blink twice every second. Meanwhile, the buzzer makes 4 warning sounds to tell you the camera has not yet been installed and then a pleasant sound to indicate the finish of initialization.



If the above phenomenas happen, it means the firmware has been burnt.

If the firmware has been burnt on your robot, please skip to Chapter 1.
Otherwise, please continue with the following steps.

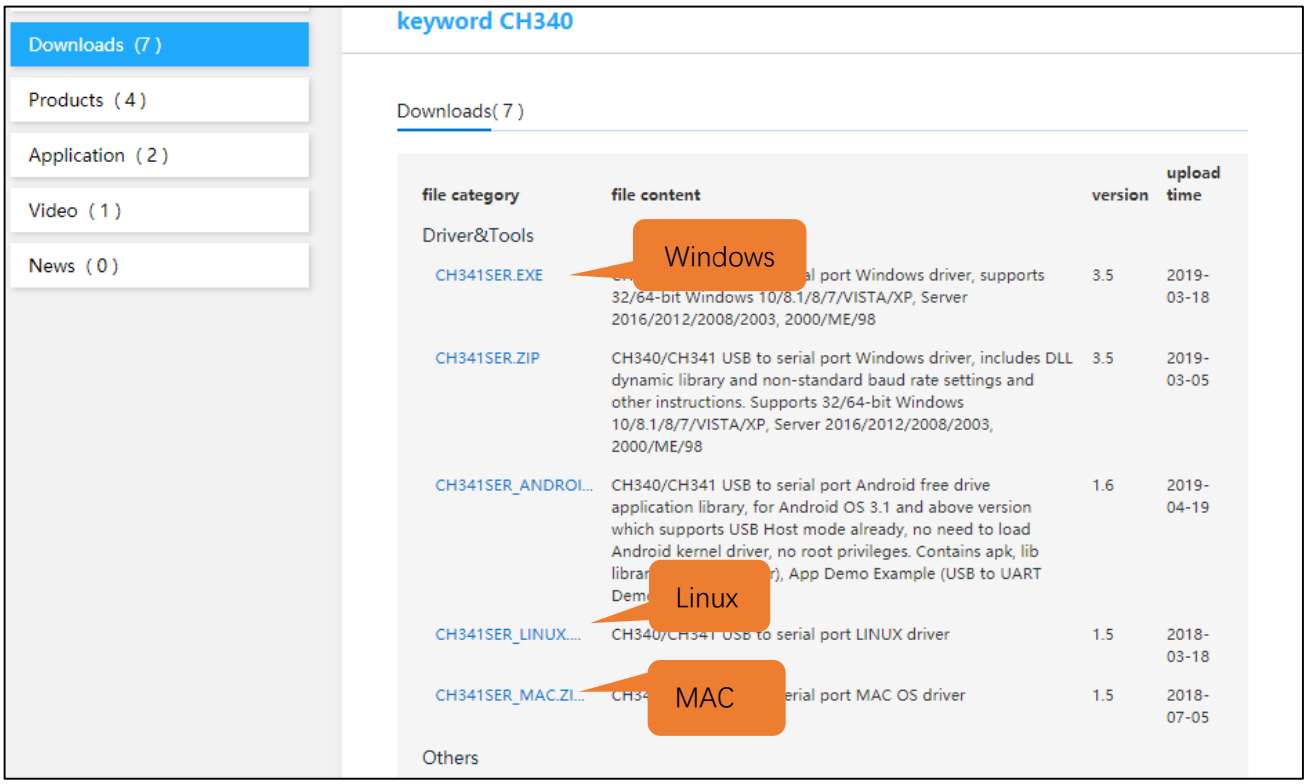
Please send emails to us (support@freenove.com) if you have any questions regarding the robot.

Need support? ✉ support@freenove.com

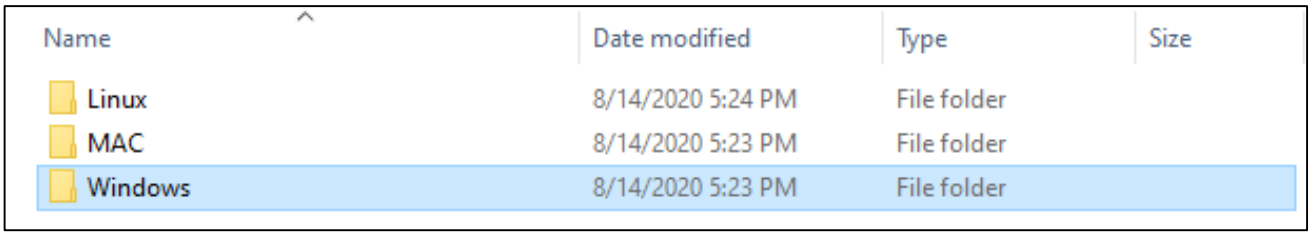
Install CH340 Driver

The computer uploads codes to ESP32 via CH340, so we need to install CH340 driver on our computer before using.

1. First, download the CH340 driver. Click http://www.wch-ic.com/search?q=CH340&t=downloads to download the appropriate one based on your operating system.



If you would not like to download the installation package, you can open **"Freenove_Robot_Dog_Kit_for_ESP32/CH340"**. We have prepared the installation package.

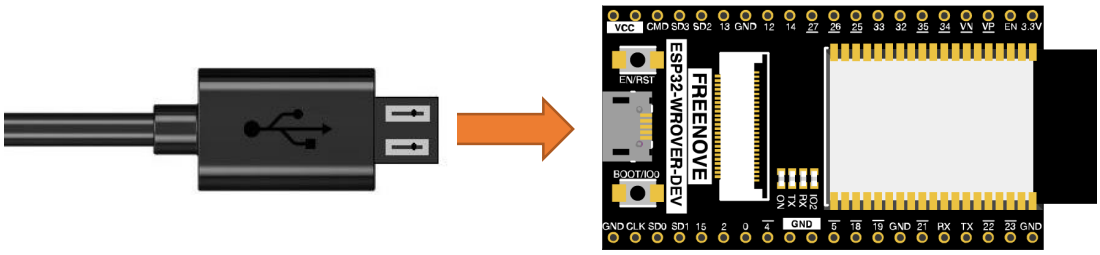


Next, we will explain how to install CH340 on different operating systems including Windows, Mac OS and Linux.

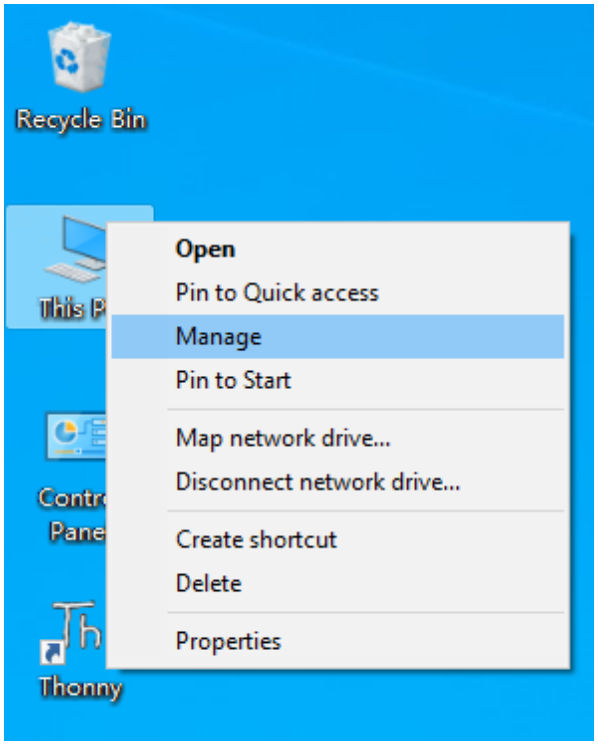
Windows

Check the Installation of CH340

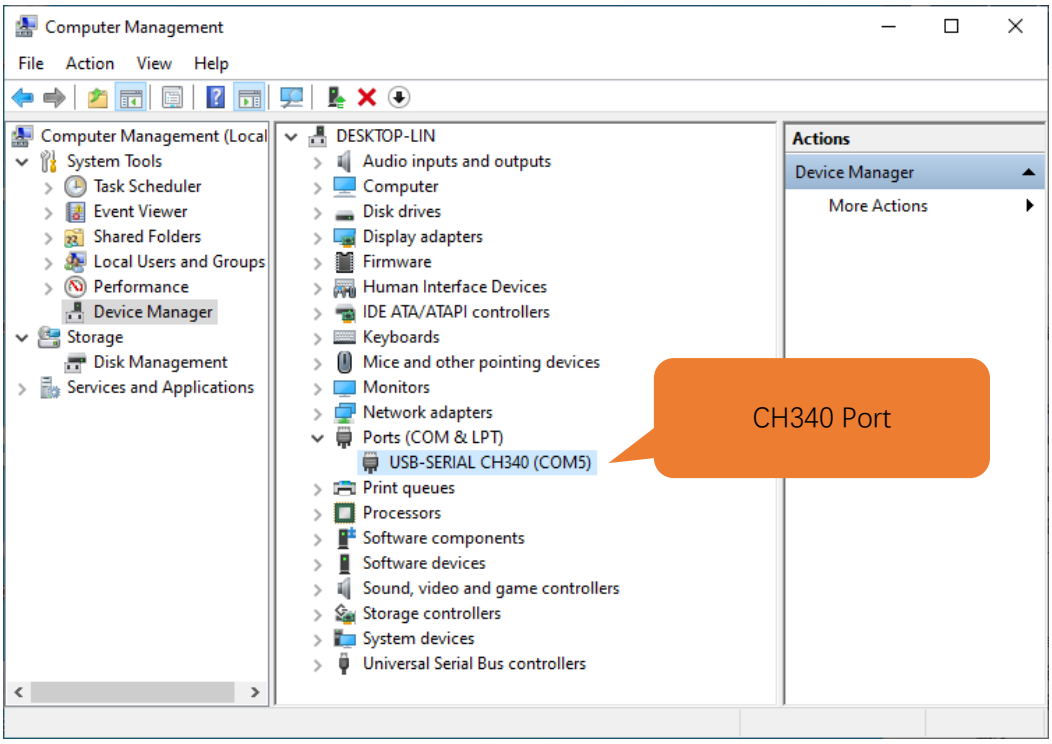
1. Connect esp32 to your computer with a USB cable.



2. Right-click on “This PC” on your computer desktop and select “Manage”.



3. Click on "Device Manager" on the left of the pop-up window, and then click on "Ports" on the right. If your computer has installed CH340 driver, you can see the port: USB-SERIAL CH340 (COMx).

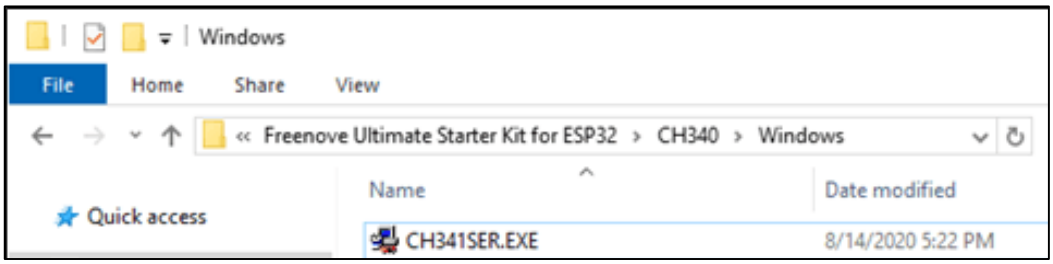


If so, you can click here to move to the next step.

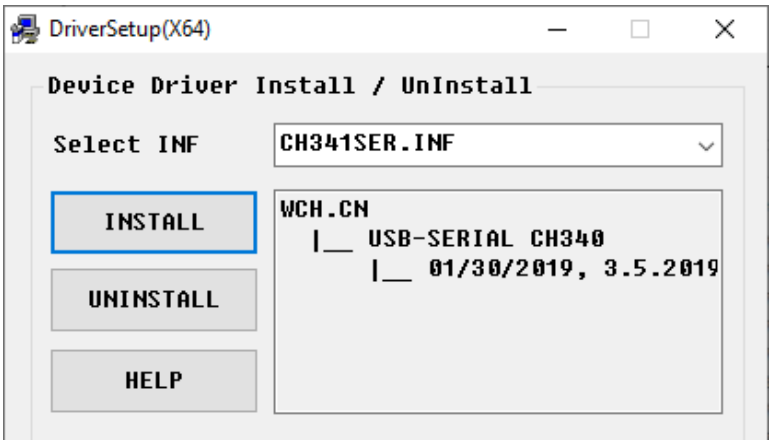
If CH340 (COMx) does not show on your computer, you need to install CH340 driver.

Install CH340 Driver

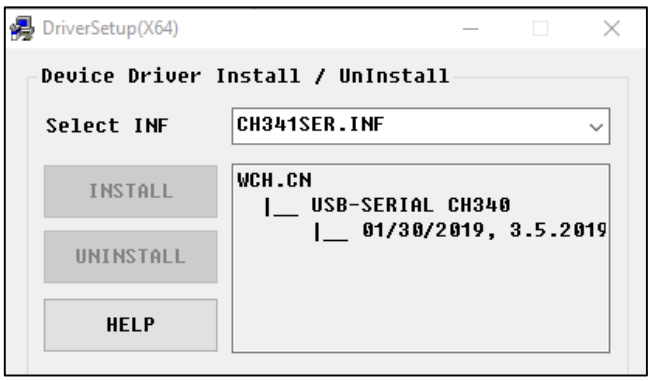
4. Open "**Freenove_Robot_Dog_Kit_for_ESP32/CH340/Windows/**".



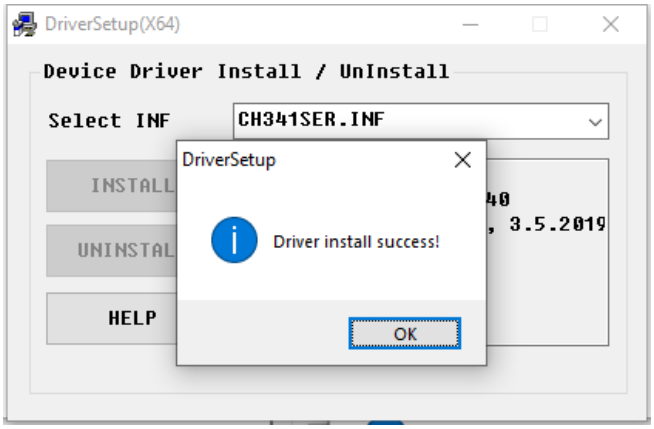
5. Double click to run the file "**CH341SER.EXE**", whose interface is as below:



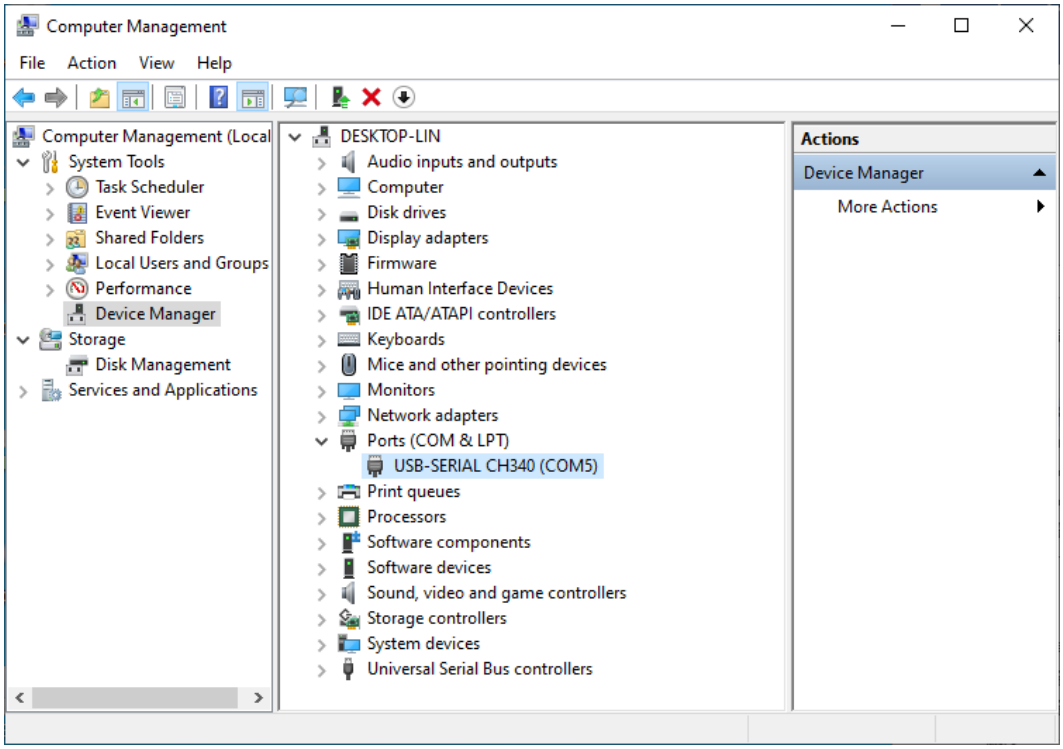
6. Make sure ESP32 has connected to your computer and then click “INSTALL”. Wait for the installation to finish.



The following window indicates that the installation finishes.



7. After installation, open device manager again and you can see the port USB-SERIAL CH340 (COMx).

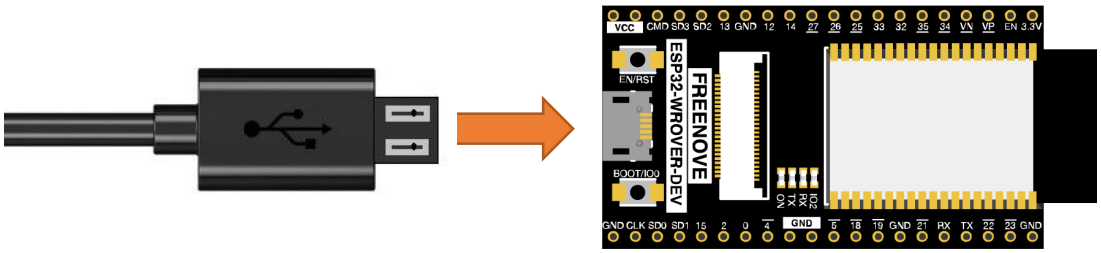


So far, CH340 has been installed. Close all dialog boxes.

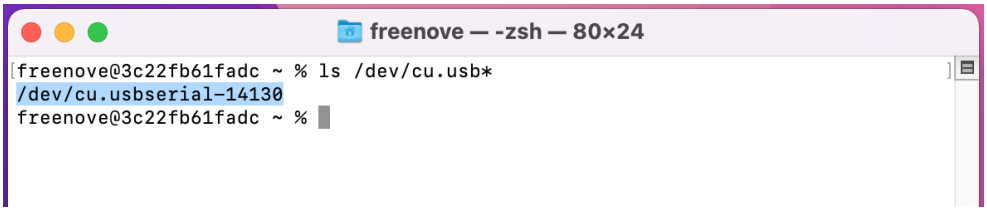
Mac OS

Check the Installation of CH340

Connect esp32 to your computer with a USB cable.



Open Terminal of Mac OS, and type in the command **ls /dev/cu.usb***



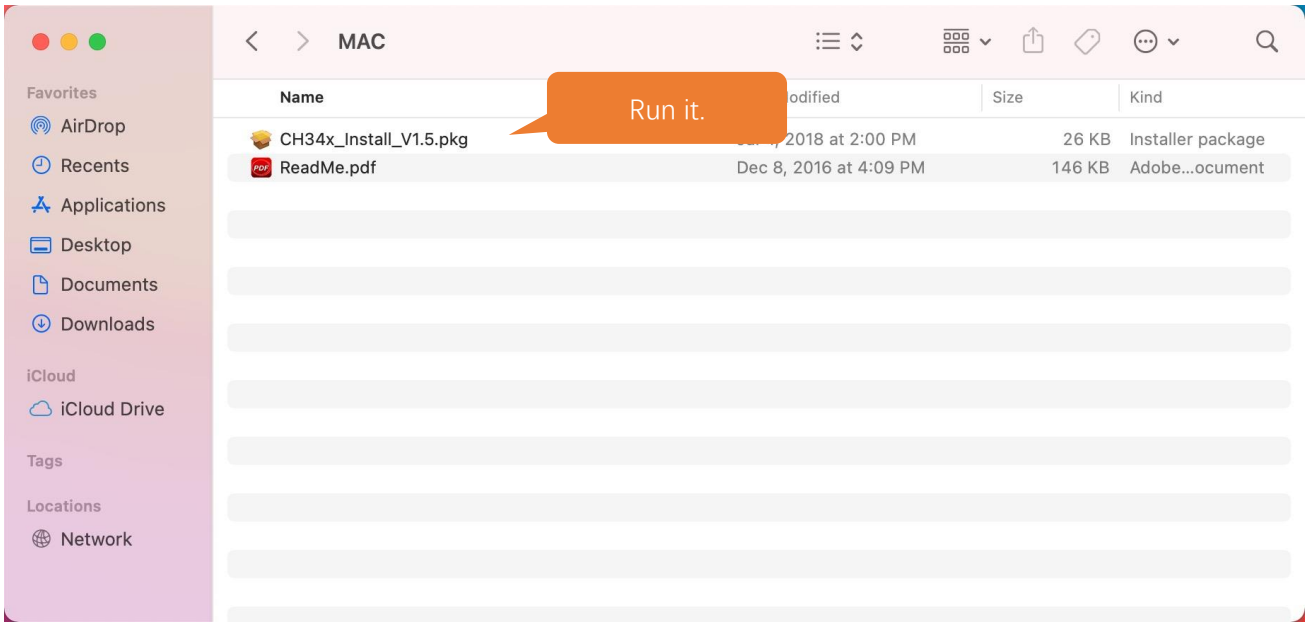
If your Terminal prints the message similar to the above, then your computer has installed the CH340. You can click here to move to the next step.

Otherwise, please continue with the following the steps.

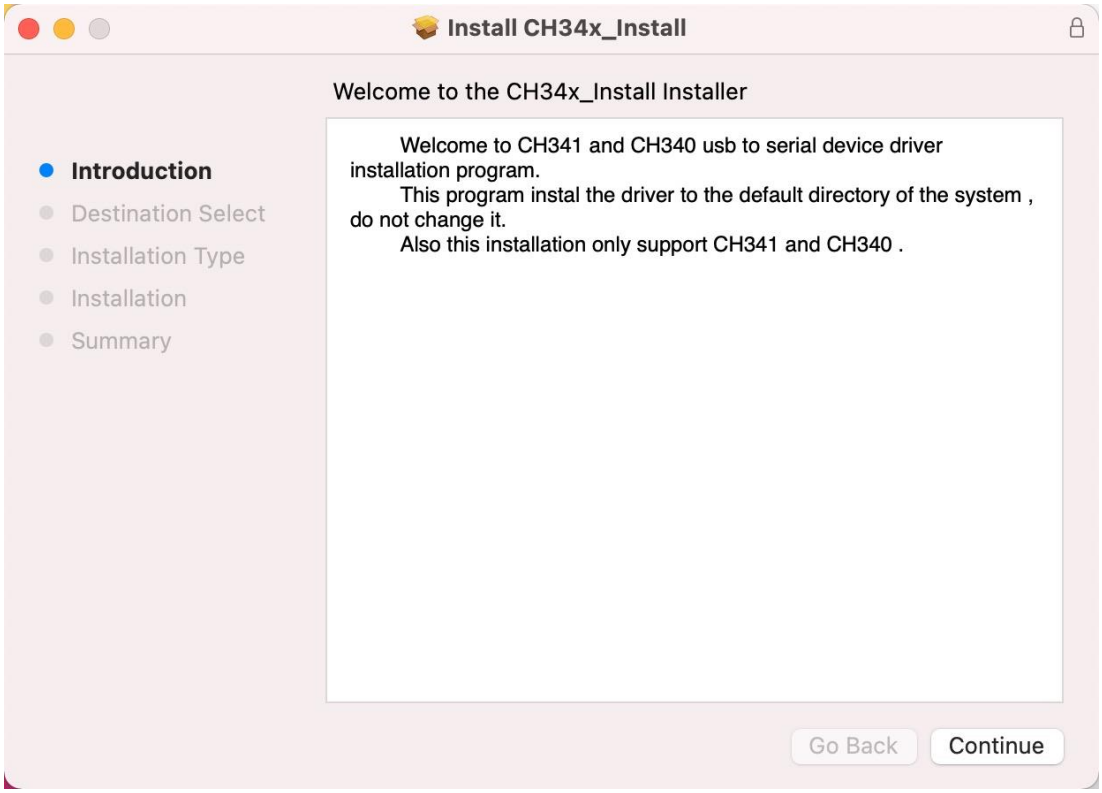
Install CH340

If you would not like to download the installation package, you can open **"Freenove_Robot_Dog_Kit_for_ESP32/CH340"**. We have prepared the installation package.

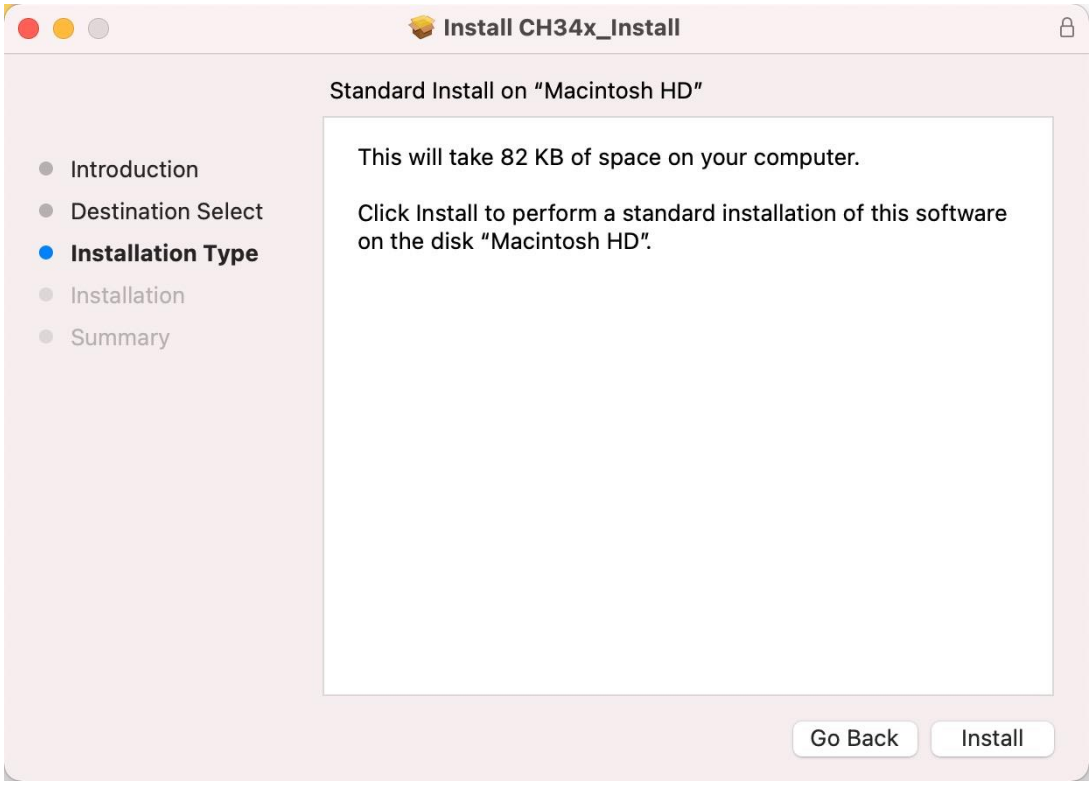
Open the folder "**Freenove_Robot_Dog_Kit_for_ESP32/CH340/MAC/**"

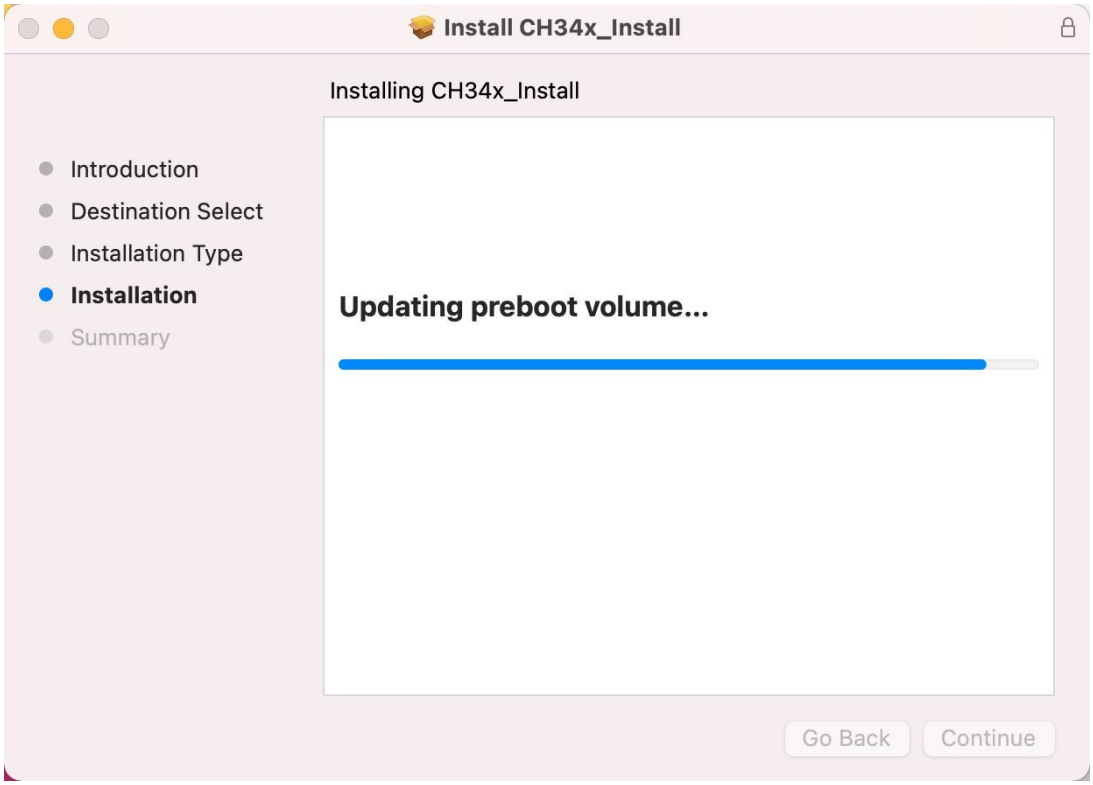


Click Continue.

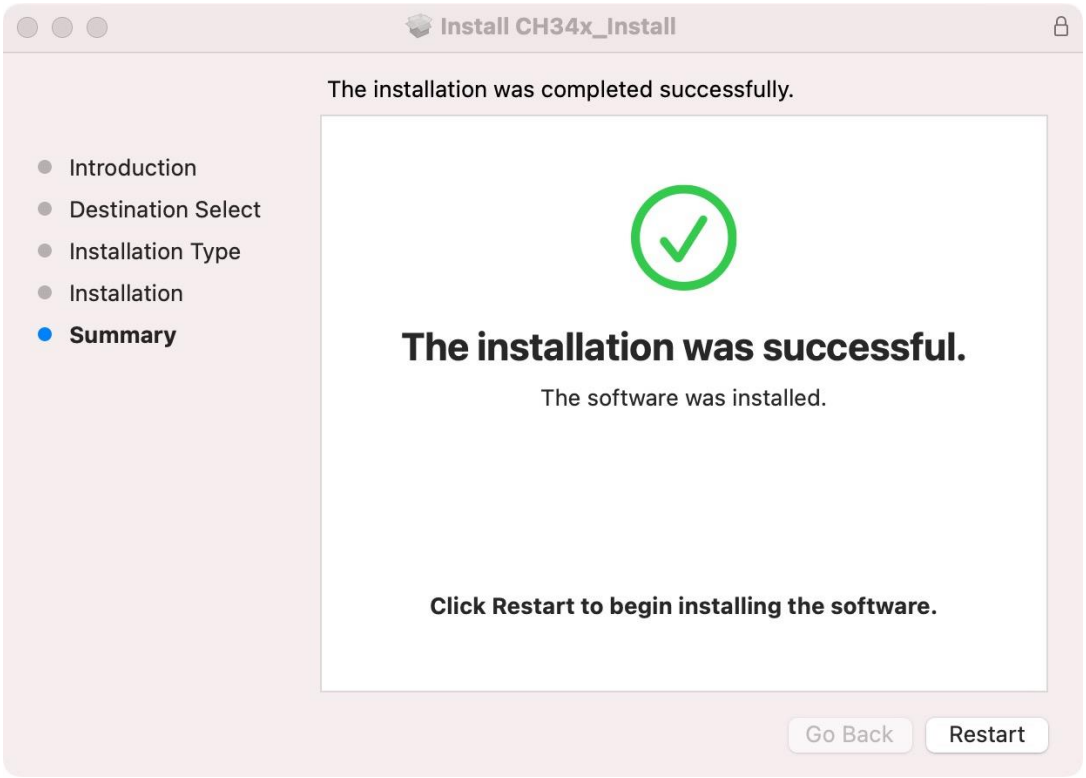


Click Install and wait for it to finish.

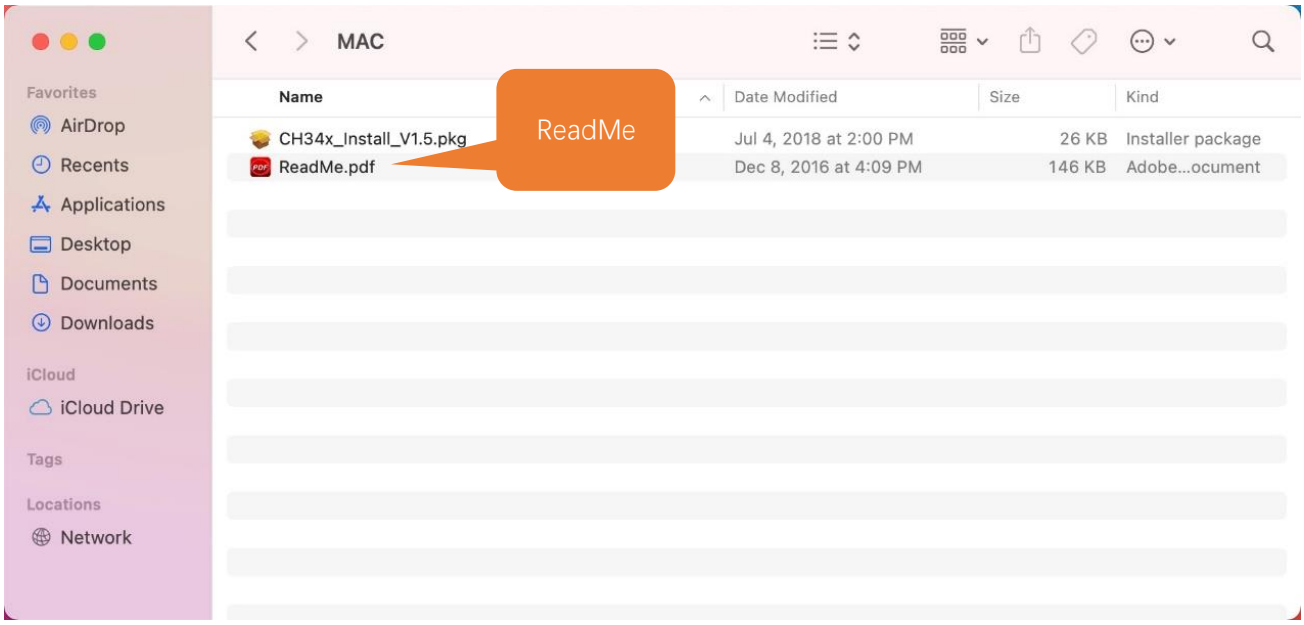




Restart your PC.



If CH340 is still not installed after the above steps, please refer to the ReadMe.pdf to install.



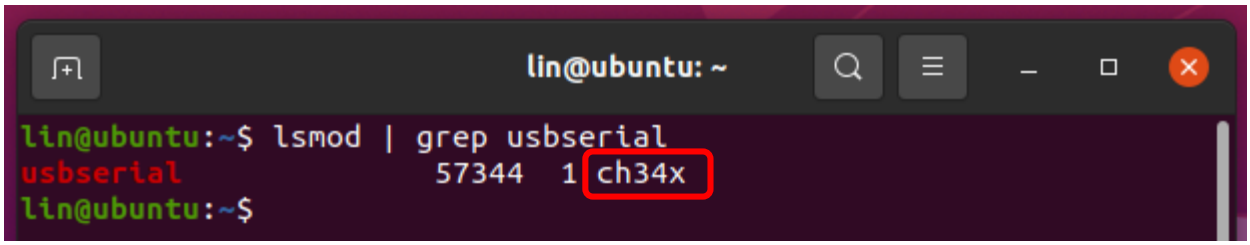
Burn the Firmware

For this product, ESP32 has burned the required firmware by default. If your ESP32 does not have the firmware or the firmware does not work, please re-burn the firmware with the following steps.
We will explain respectively for Windows, Mac OS and Linux systems.

Linux

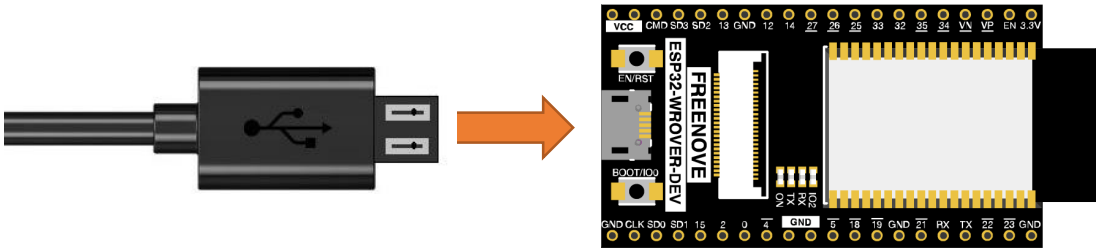
Check the Installation of CH340

Open the system terminal and type in the command: **lsmod | grep usbserial**. If your computer has installed the driver, you should see the following information:

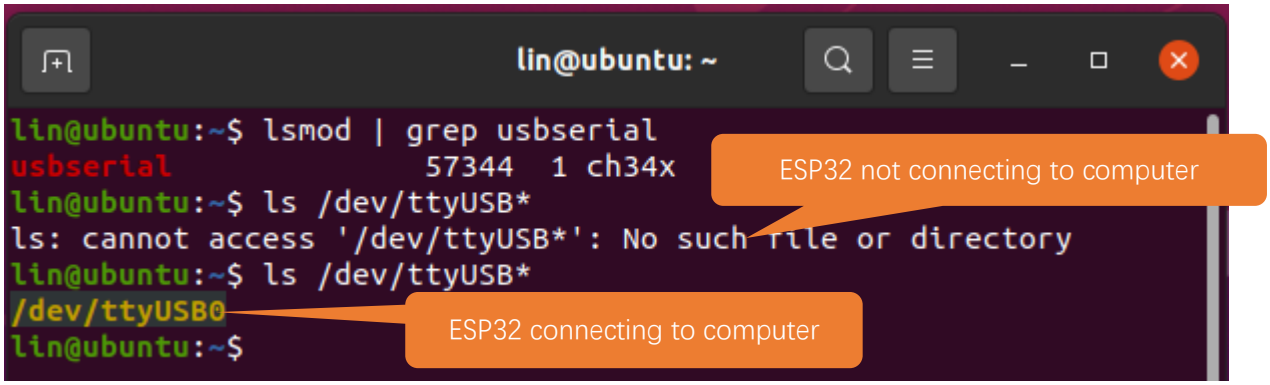


If the driver has been installed, you can determine the port used by ESP32 to communicate with your computer in this way:

1. When ESP32 is not connected to your computer, open system terminal and type in the command **ls /dev/tty***
2. Connect ESP32 to your computer with a USB cable and type in the command **ls /dev/tty*** again.



Compare the results. As shown below, **/dev/ttyUSB0** is the port that ESP32 communicates with your computer.



If your computer has installed CH340, you can click here to skip to the next step.

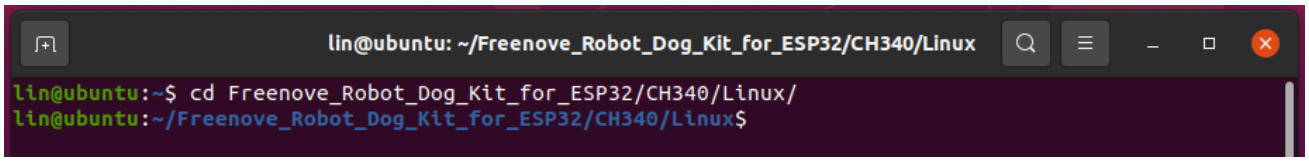
Install CH340

If you connect the ESP32 to your computer but it does not detect **/dev/ttyUSB0**, then it has not installed CH340 yet.

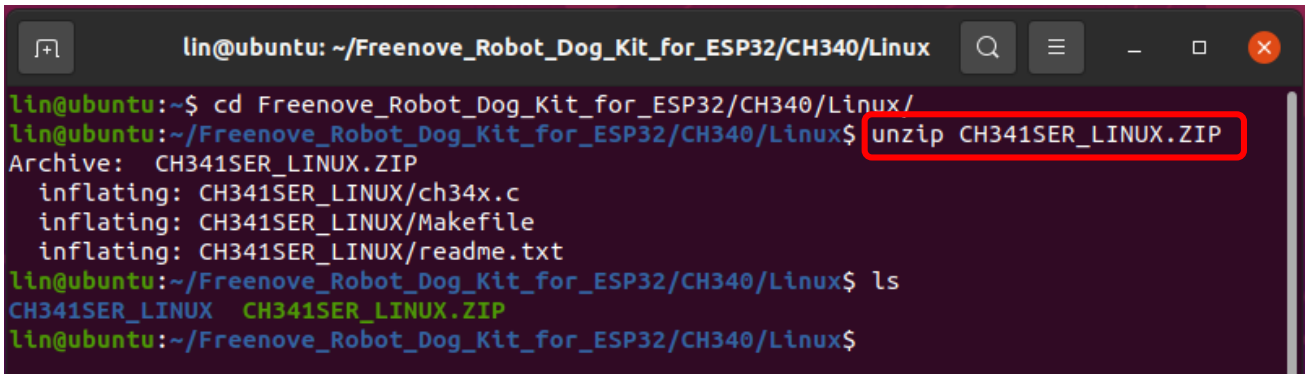
Please follow the steps below to install CH340 driver.

We have prepared the installation package for you: "**Freenove_Robot_Dog_Kit_for_ESP32/CH340/LINUX/**".

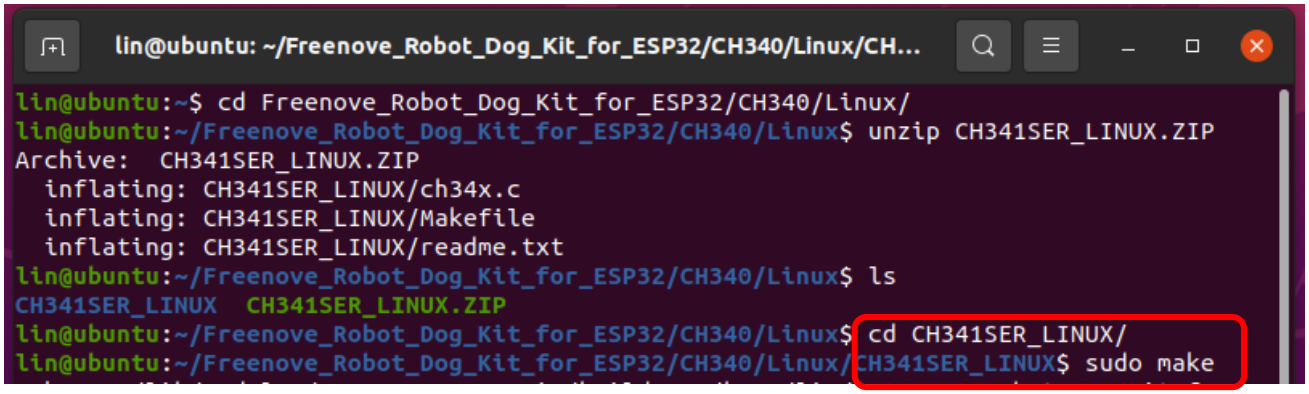
1. Enter the folder on terminal: **cd Freenove_Robot_Dog_Kit_for_ESP32/CH340/Linux/**



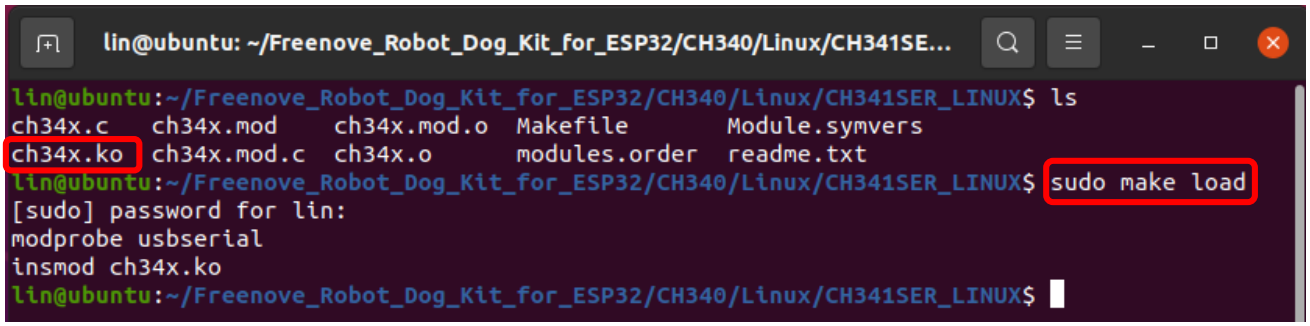
2. Unzip the installation package: **unzip CH341SER_LINUX.ZIP**



3. Enter the unzipped folder and type in the command **sudo make** to compile and generate the file ch34x.ko.



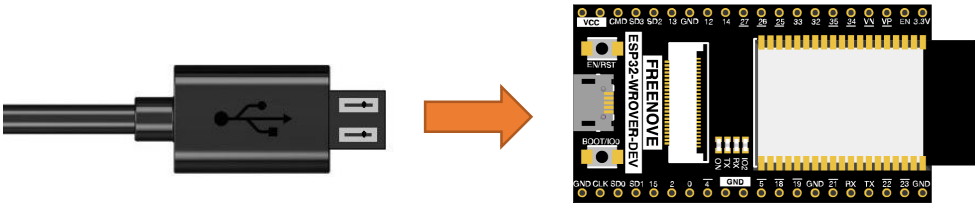
4. Use the ls command to check the file. As you can see below, the ch34.ko has been generated under the current directory.
5. Type in the command to upload the file to the system: **sudo make load**



So far, the ch340 driver has been installed.

Windows

First, connect ESP32 to your computer with a USB cable.

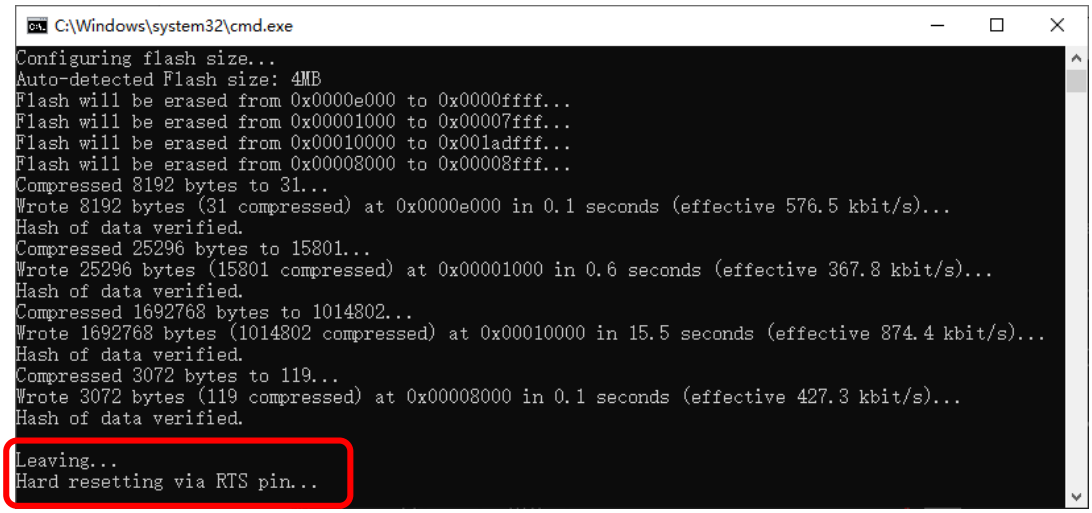


Second, open "windows.bat" under the directory of Freenove_Robot_Dog_Kit_for_ESP32/Firmware/Windows with txt editor, and modify the COMx in the file according to the port USB-SERIAL CH340 (COMx) on your computer.

```
@echo off

:start

echo.

esptool.exe --port COM4 erase_flash

echo.

esptool.exe --chip esp32 --port COM4 --baud 2000000 --before default_reset --after hard_reset write_flash -z --flash_mode

echo.

pause

echo.
```

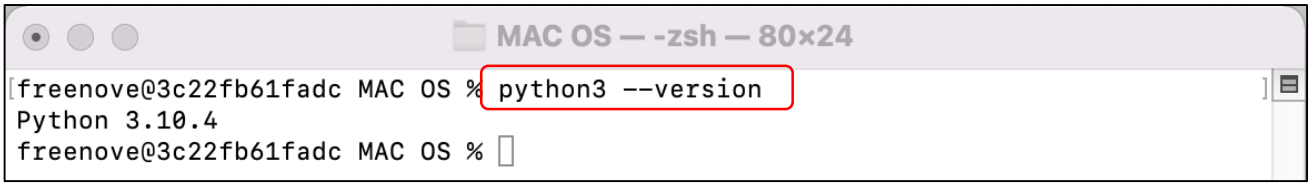
Note: Do NOT modify other contents.

Third, save and close the file. Double-click it to run and wait for it to finish downloading.



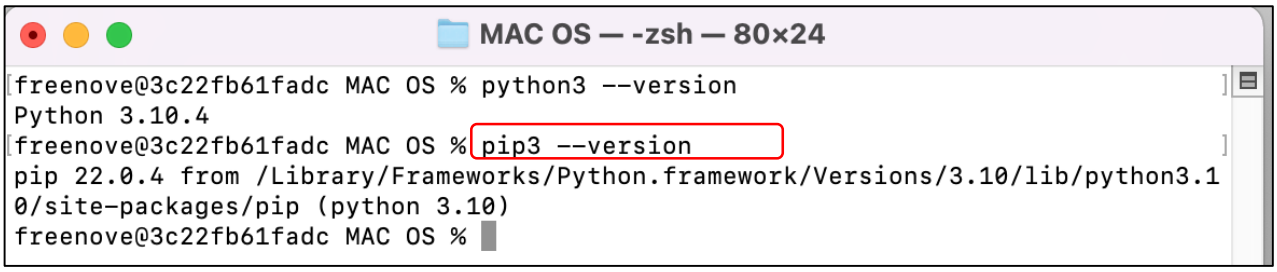
Mac OS

1. Open Terminal on your computer and type in the command **python3 --version** to check whether python3 has been installed on your computer.

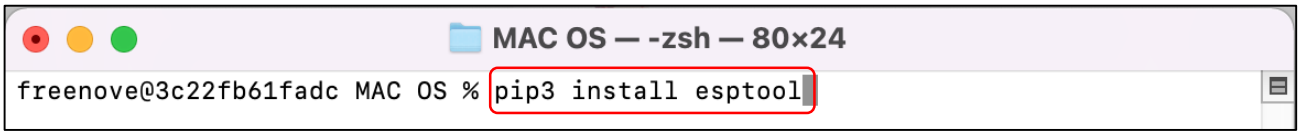


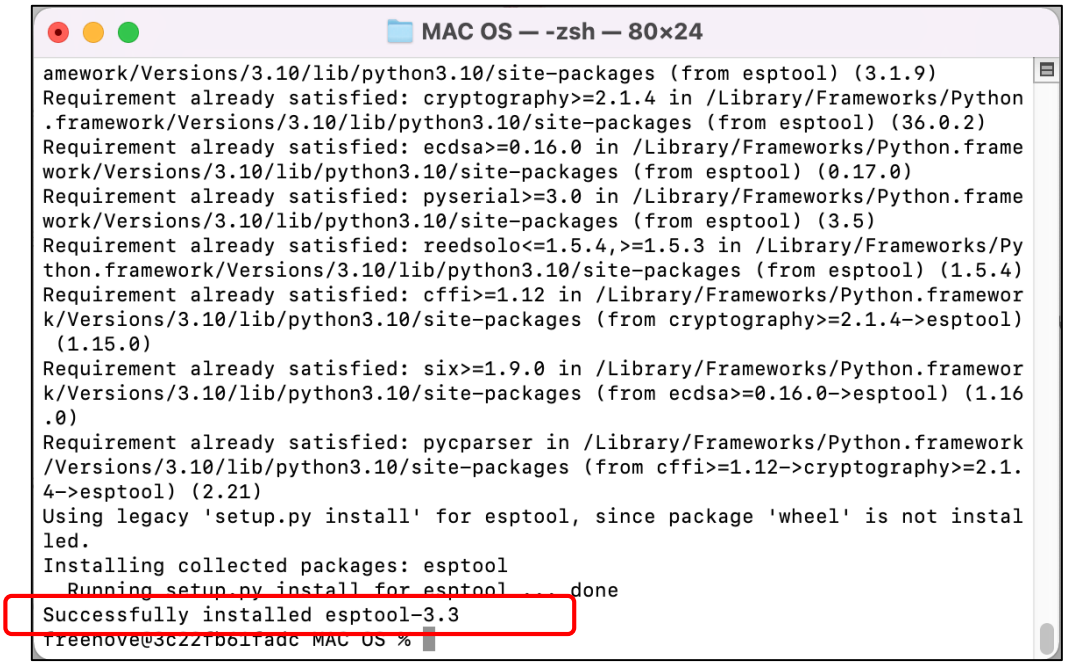
If your computer has not yet installed python3, please type in the command to install: **brew install python3**

2. Type in the command **pip3 --version** to check whether python3 has integrated with pip3. If it has not, please type in the command **curl https://bootstrap.pypa.io/get-pip.py | python3** to install.



3. Enter the command to install firmware-downloading tool: **pip3 install esptool**

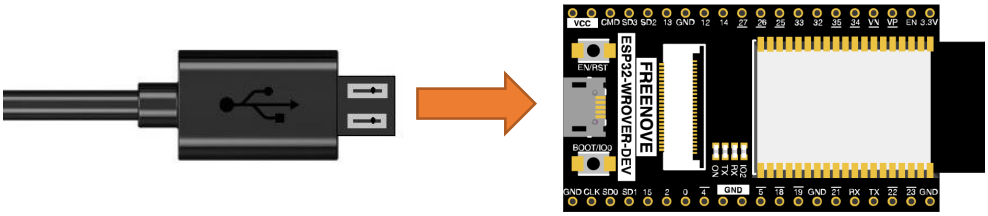




4. Check whether esptool has been installed: **pip3 list**

```
MAC OS — -zsh — 80×24
.0)
Requirement already satisfied: pycparser in /Library/Frameworks/Python.framework
/Versions/3.10/lib/python3.10/site-packages (from cffi>=1.12->cryptography>=2.1.
4->esptool) (2.21)
Using legacy 'setup.py install' for esptool, since package 'wheel' is not instal
led.
Installing collected packages: esptool
  Running setup.py install for esptool ... done
Successfully installed esptool-3.3
freenove@3c22fb61fadc MAC OS % pip3 list
Package      Version
------------ -------
bitstring    3.1.9
cffi         1.15.0
cryptography 36.0.2
ecdsa        0.17.0
esptool      3.3
pip          22.0.4
pycparser    2.21
pyserial     3.5
reedsolo     1.5.4
setuptools   58.1.0
six          1.16.0
freenove@3c22fb61fadc MAC OS %
```

5. Connect ESP32 to your computer with the USB cable.



6. Open Mac OS Terminal and type in the command to check whether ESP32 can be detected: **ls /dev/cu.usb***

```
freenove — -zsh — 80×24
freenove@3c22fb61fadc ~ % ls /dev/cu.usb*
/dev/cu.usbserial-14130
freenove@3c22fb61fadc ~ %
```

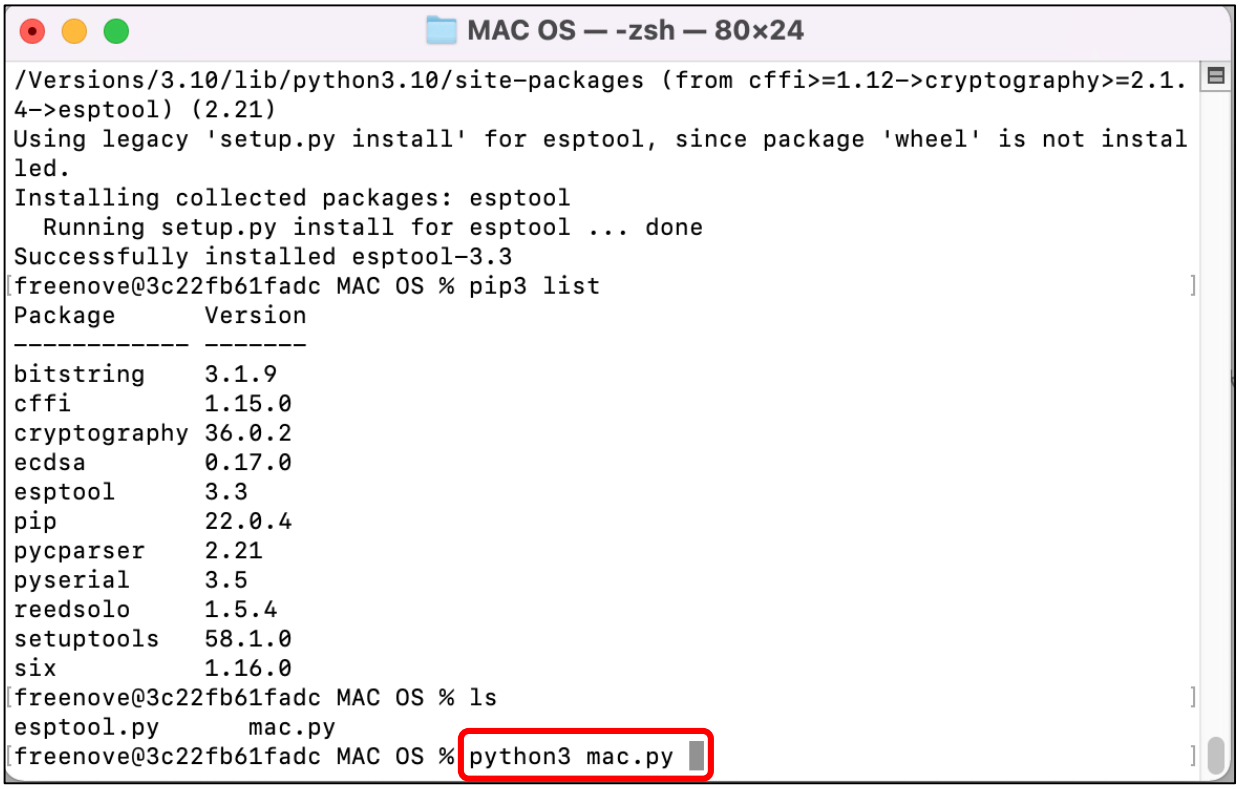
The port number may vary among different computers. Here we take "/dev/cu.usbserial-14130" as an example. Copy the serial number.

7. Open mac.py under the directory of Freenove_Robot_Dog_Kit_for_ESP32/Firmware/MAC OS/ with txt editor, and modify the port number to that of your computer. Save it and exit.

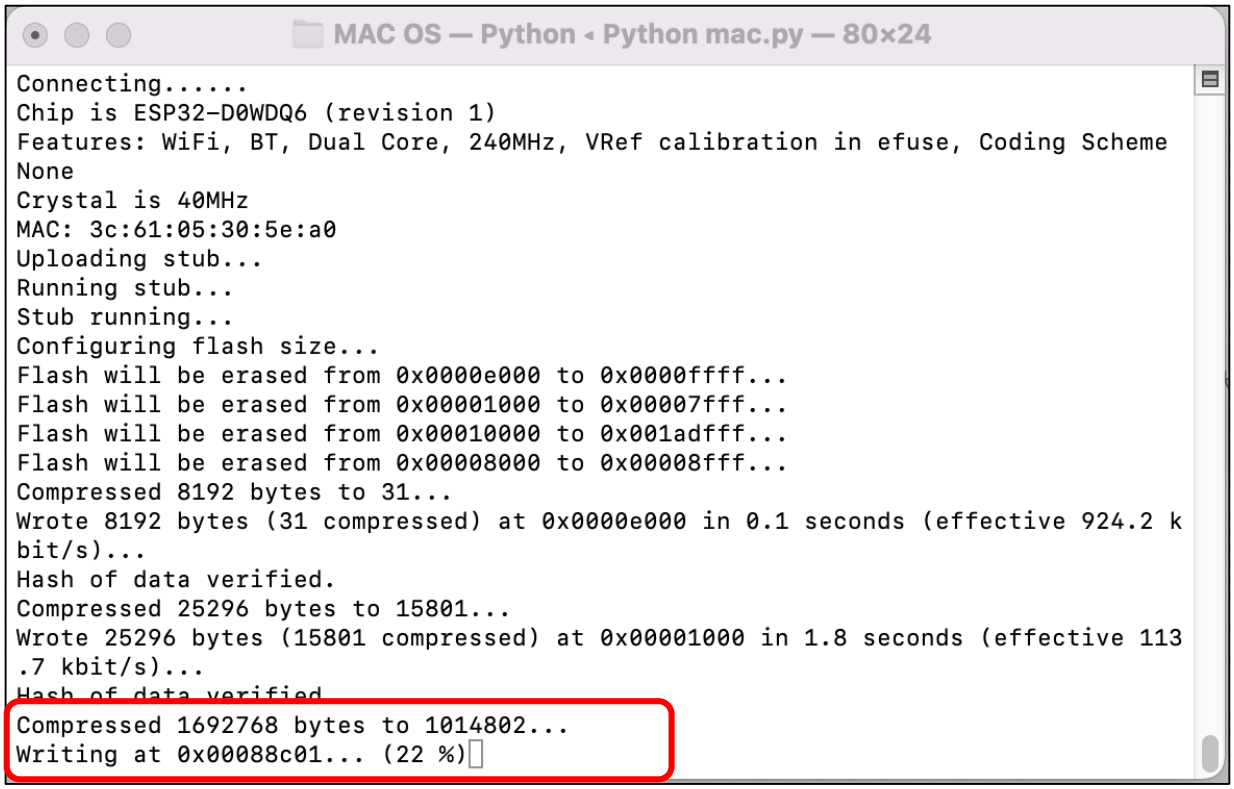
```
import os
import sys
os.system("esptool.py -p /dev/cu.usbserial-14130 erase_flash")
os.system("esptool.py --chip esp32 -p /dev/cu.usbserial-14130 --baud 115200 --before default_reset --after hard_reset write_f
```

Note: Please make sure only the port number is changed and other information, including space is not changed; otherwise, the firmware may fail to burn.

8. Enter the command: **cd Freenove_Robot_Dog_Kit_for_ESP32/Firmware/MAC OS/** and **python3 mac.py** one by one to install the firmware to esp32.



9. Wait for it to finish.



Linux

1. Check whether your computer has installed python3. If it has, please skip to the next step.

```
lin@ubuntu: ~
lin@ubuntu:~$ python3 --version
Python 3.8.10
lin@ubuntu:~$
```

If it has not, please run the command to install.

```
lin@ubuntu: ~
lin@ubuntu:~$ sudo apt install python3.8.10
```

2. Check whether pip3 has been installed on your computer. If it has, please move on to the next step. Otherwise, please run the following commands to install:

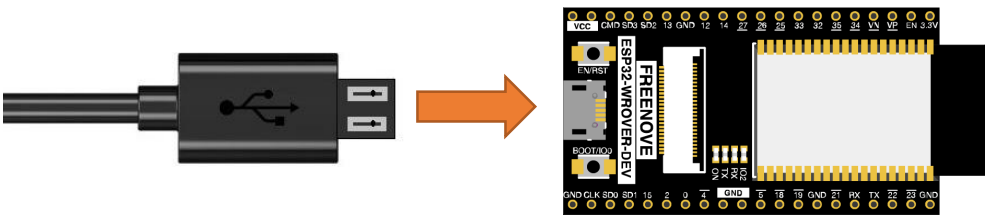
```
sudo apt update&
sudo apt install python3-pip
```

```
lin@ubuntu: ~
lin@ubuntu:~$ python3 --version
Python 3.8.10
lin@ubuntu:~$ pip3 -version

Command 'pip3' not found, but can be installed with:

sudo apt install python3-pip

lin@ubuntu:~$ sudo apt install python3-pip
```

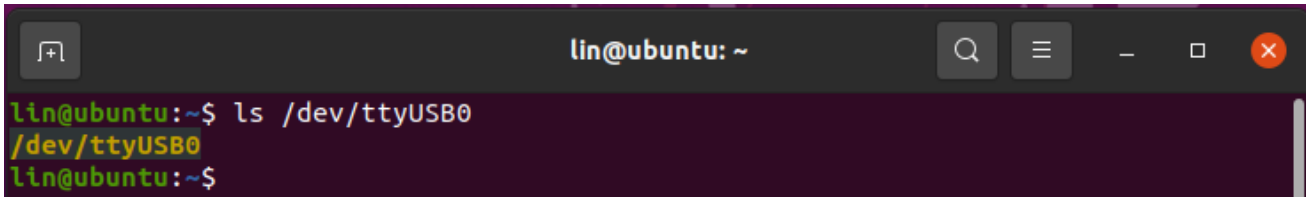
3. Type in the command: pip3 install esptool

```
lin@ubuntu: ~
lin@ubuntu:~$ pip3 install esptool
```

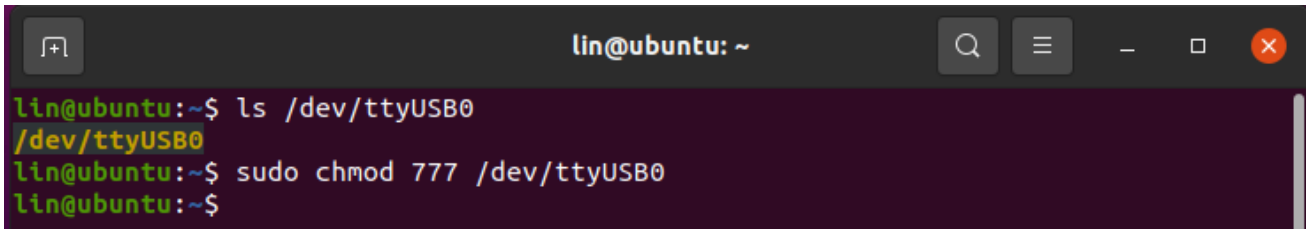
4. Connect ESP32 to your computer with the USB cable.



5. Enter the command **ls /dev/ttyUSB0** to check the port number.

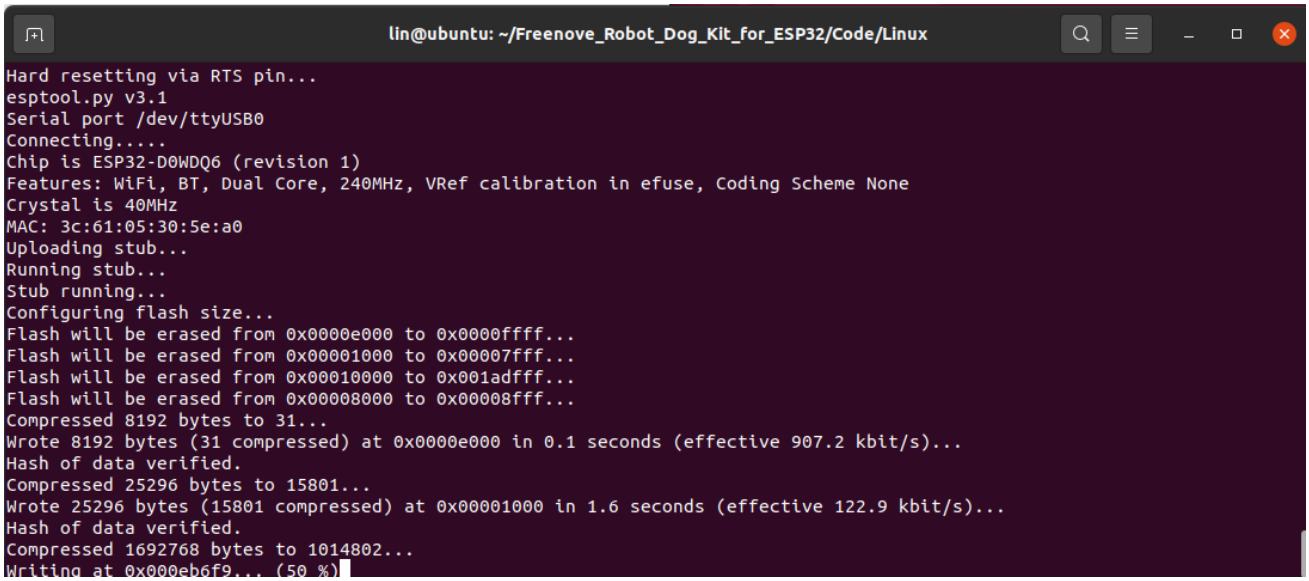


6. Enter the command **sudo chmod 777 /dev/ttyUSB0**



Note: The above command is to give permission to/dev/ttyUSB0. Without this, the code may fail to download.

7. Enter the directory of Freenove_Robot_Dog_Kit_for_ESP32/Firmware/Linux and enter the command **python3 linux.py**



8. Wait for it to finish download.

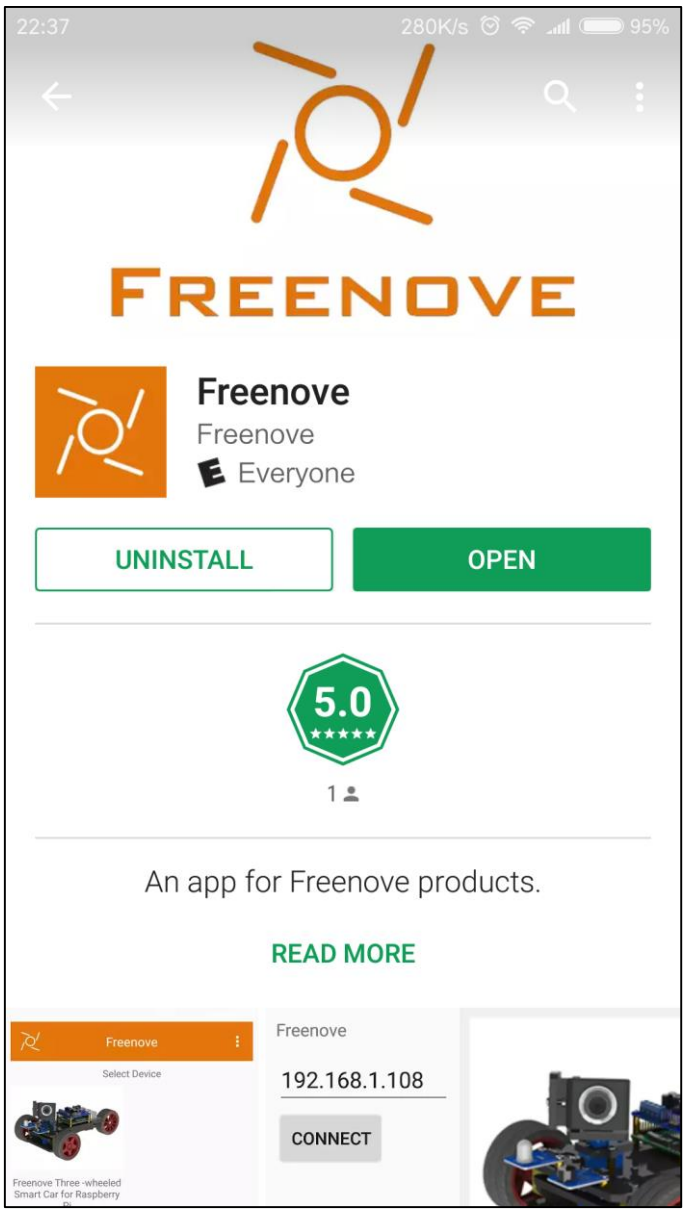
Chapter 1 Install Freenove App

Here are three installation methods. You can choose any one of them.

Install Freenove App

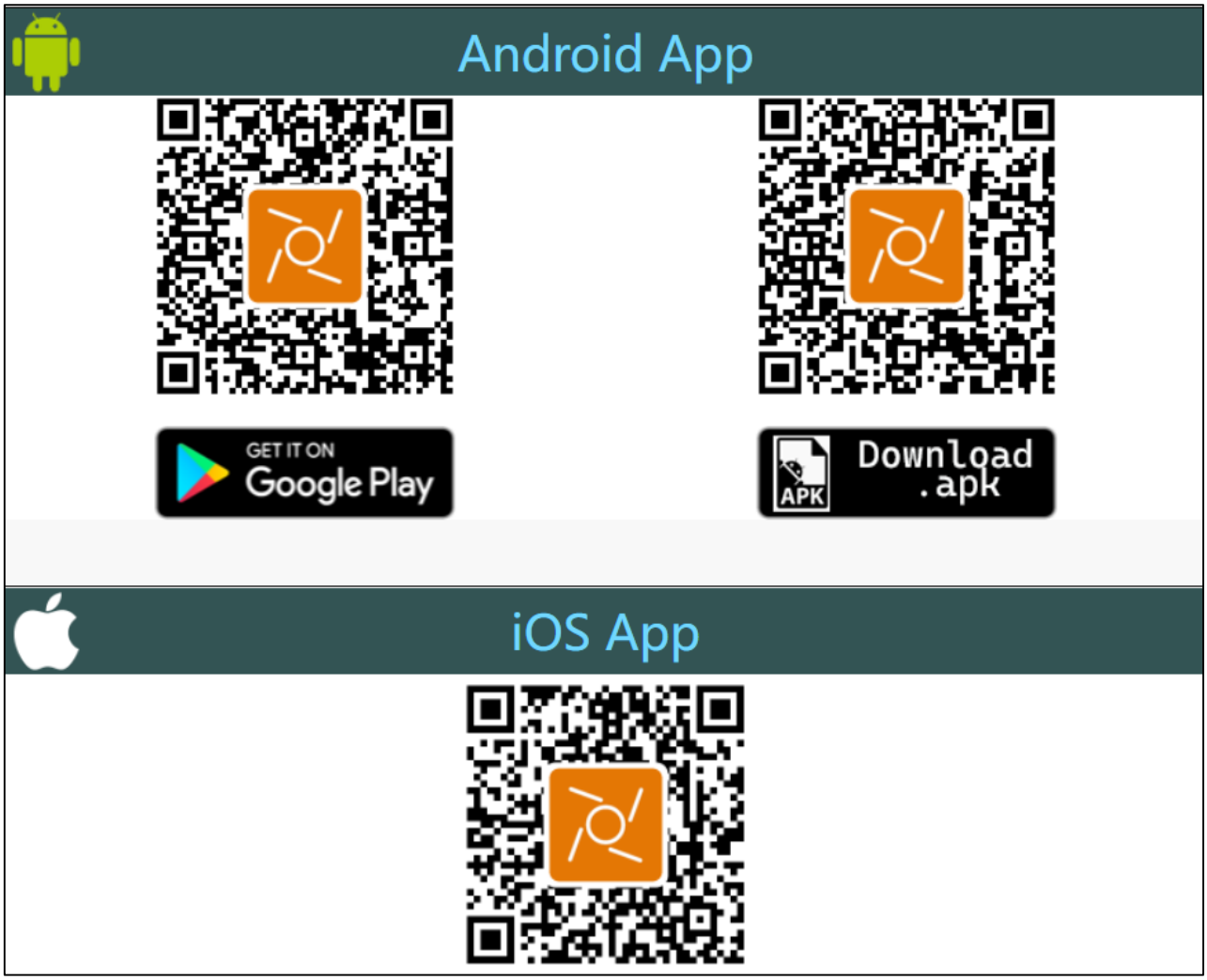
Method 1

Open Google Play on your phone and search “Freenove” to download.



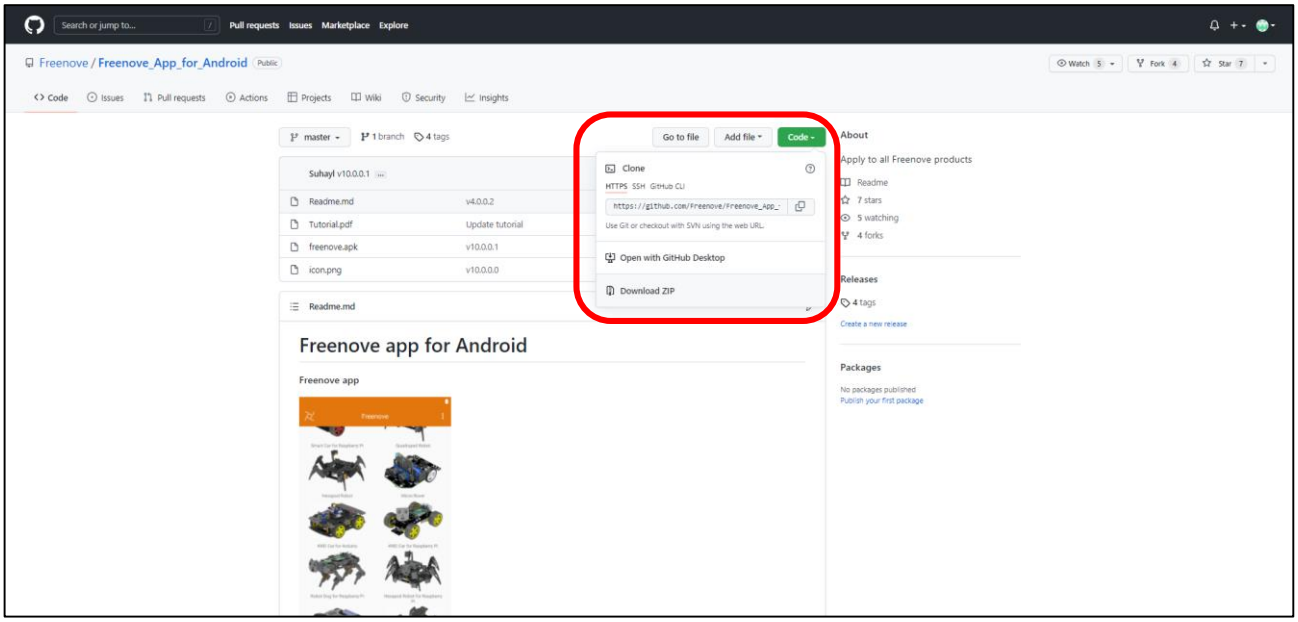
Method 2

Visit the website https://www.freenove.com/app.html with your computer and choose the one corresponding to your phone system to download, and then transfer it to your phone to install.



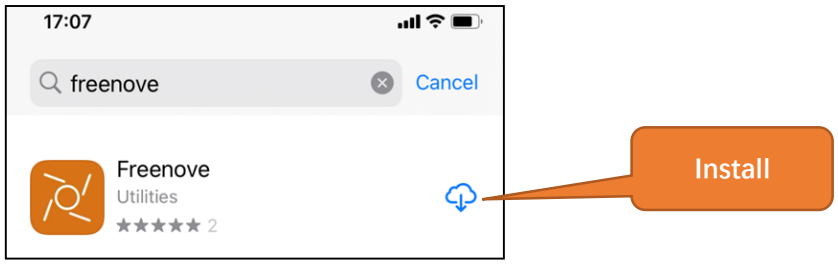
Method 3

Use your phone or computer to visit the website: https://github.com/Freenove/Freenove_app_for_Android, click Download ZIP under Code.



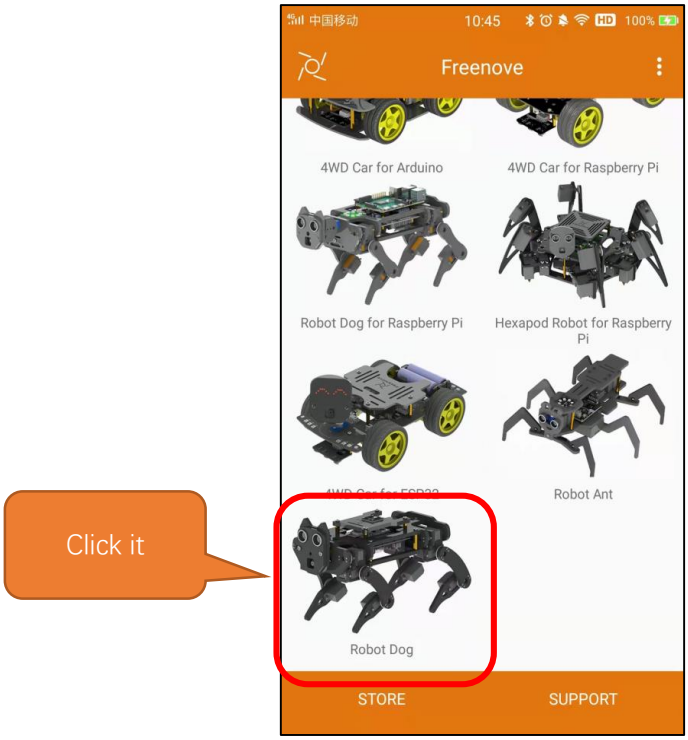
IOS

Open the iPhone's APP Store, search Freenove and install it.

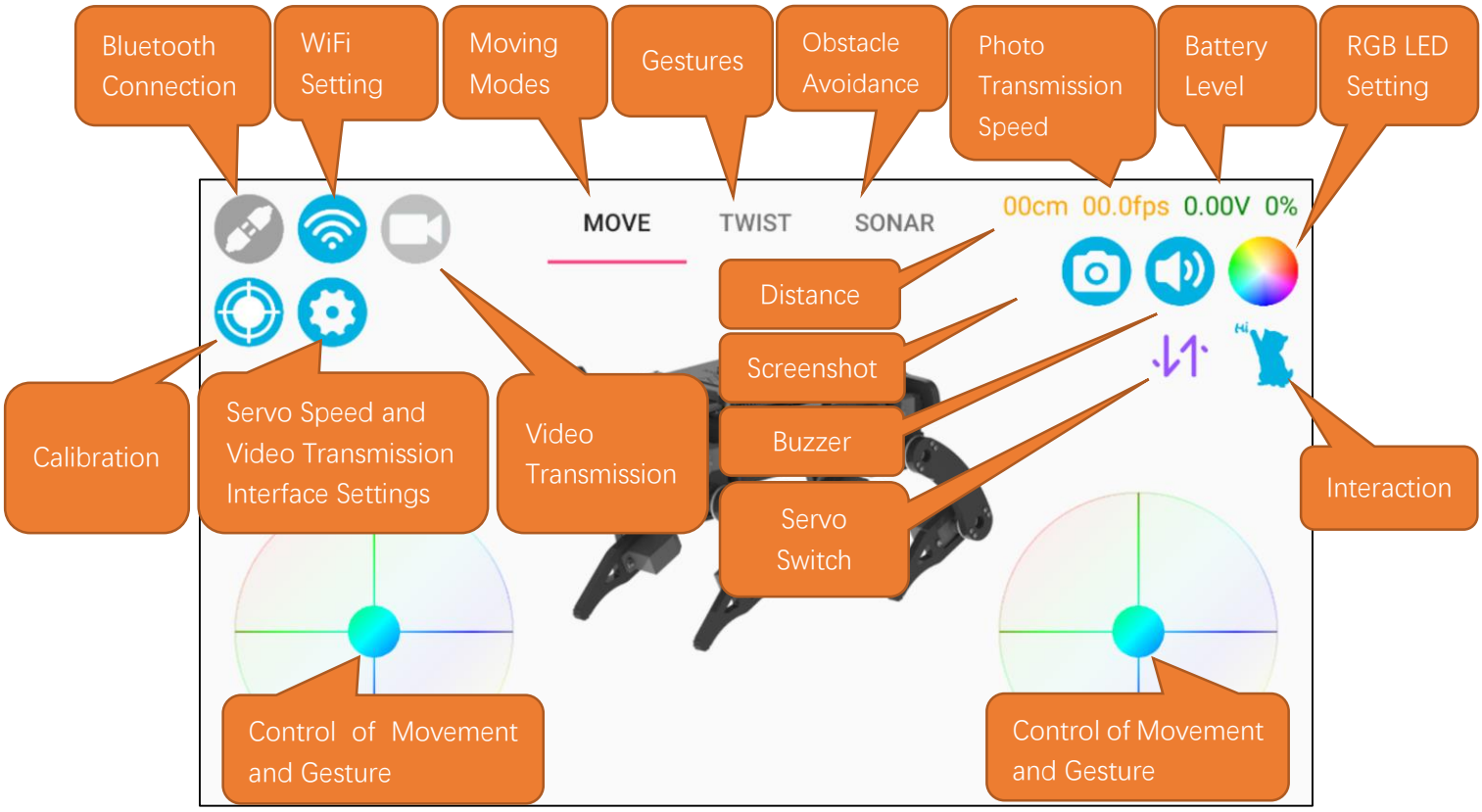


Introduction to Freenove App

Open the app and select the ESP32 robot Dog.



Before using the phone App to control the robot dog, we need to understand the interface first.



Chapter 2 Robot Assembly

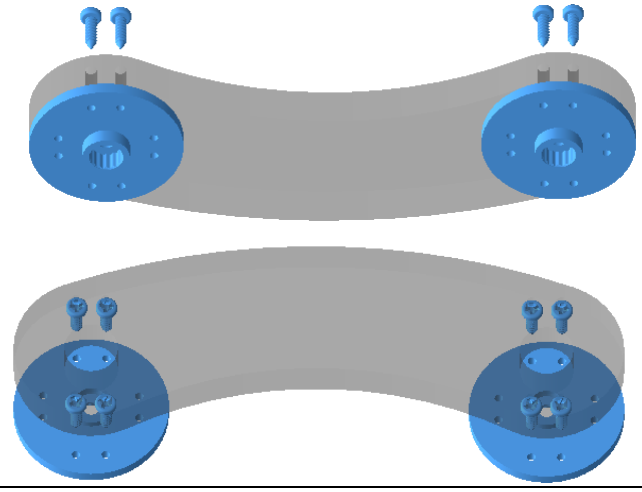
Please follow the tutorial to assemble the robot; otherwise, it may not function well.

Step 1 Assembly of Disc Servo Arms

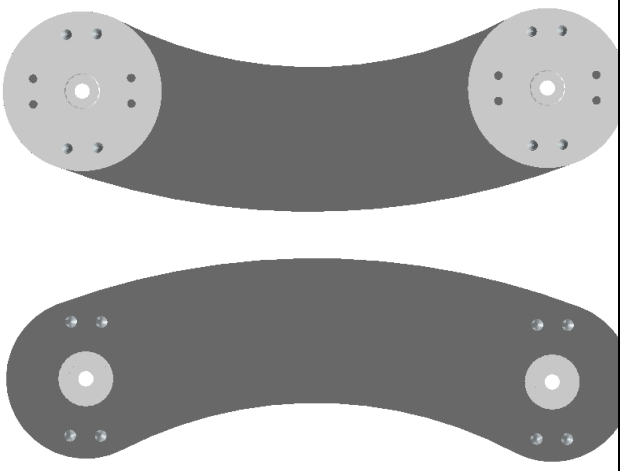
1. Take out 8 disc servo arms and 32 M1.4*8 screws from servo packages.

2. Mount two disc servo arms to the acrylic part with eight M1.4*8 screws.

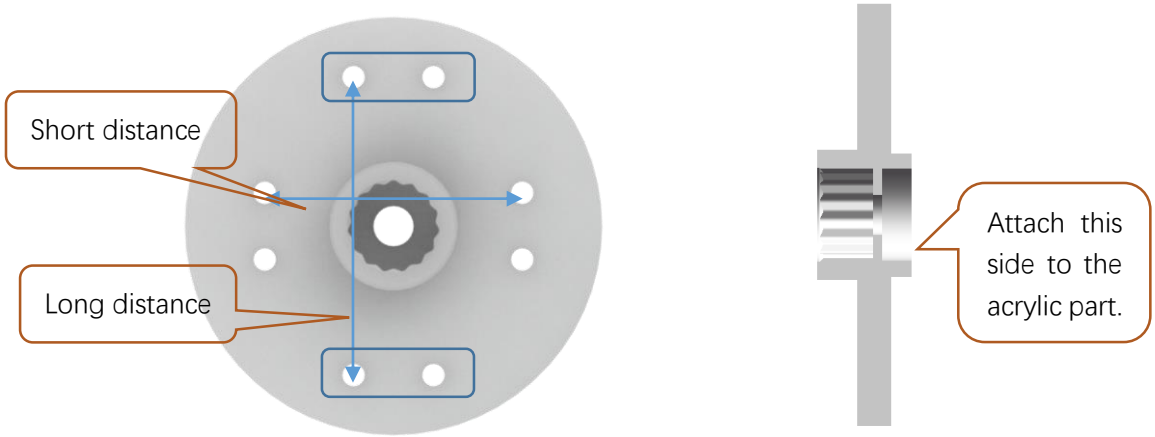
Below is a perspective view of the front and back.



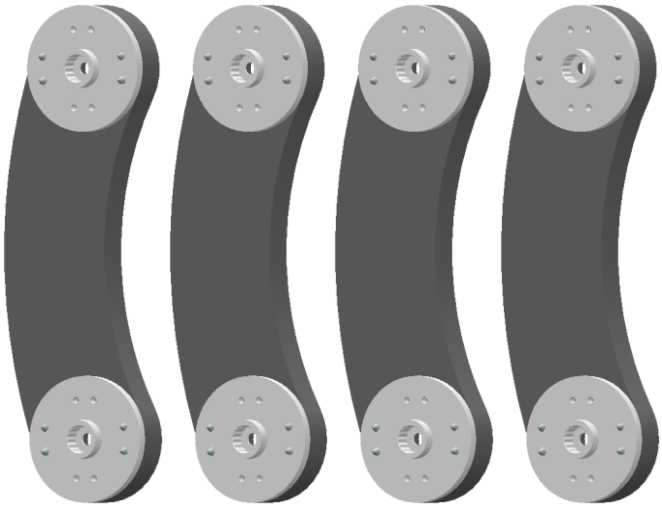
The front and back after assembly are as shown below.



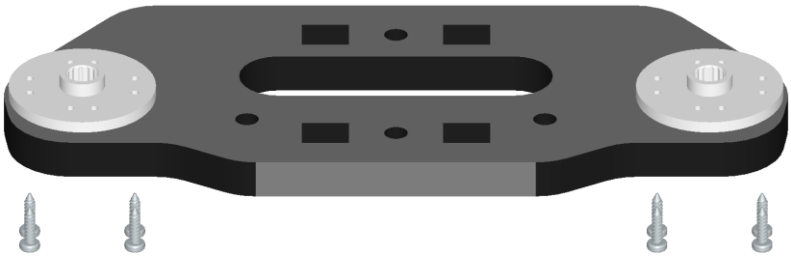
Note: The distance between the two sets of holes is different. Please use the ones with longer distance.



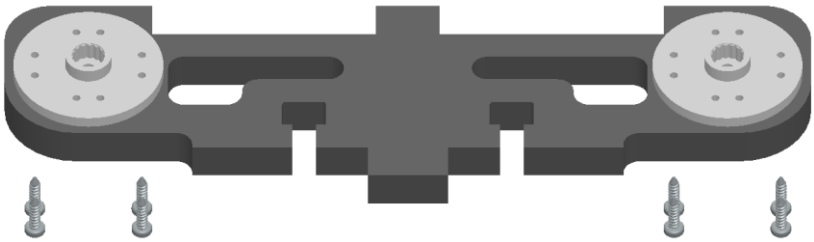
3. After assembly, you will get the following four parts.



4. Mount two disc servo arms with eight M1.4*8 screws to the head board.

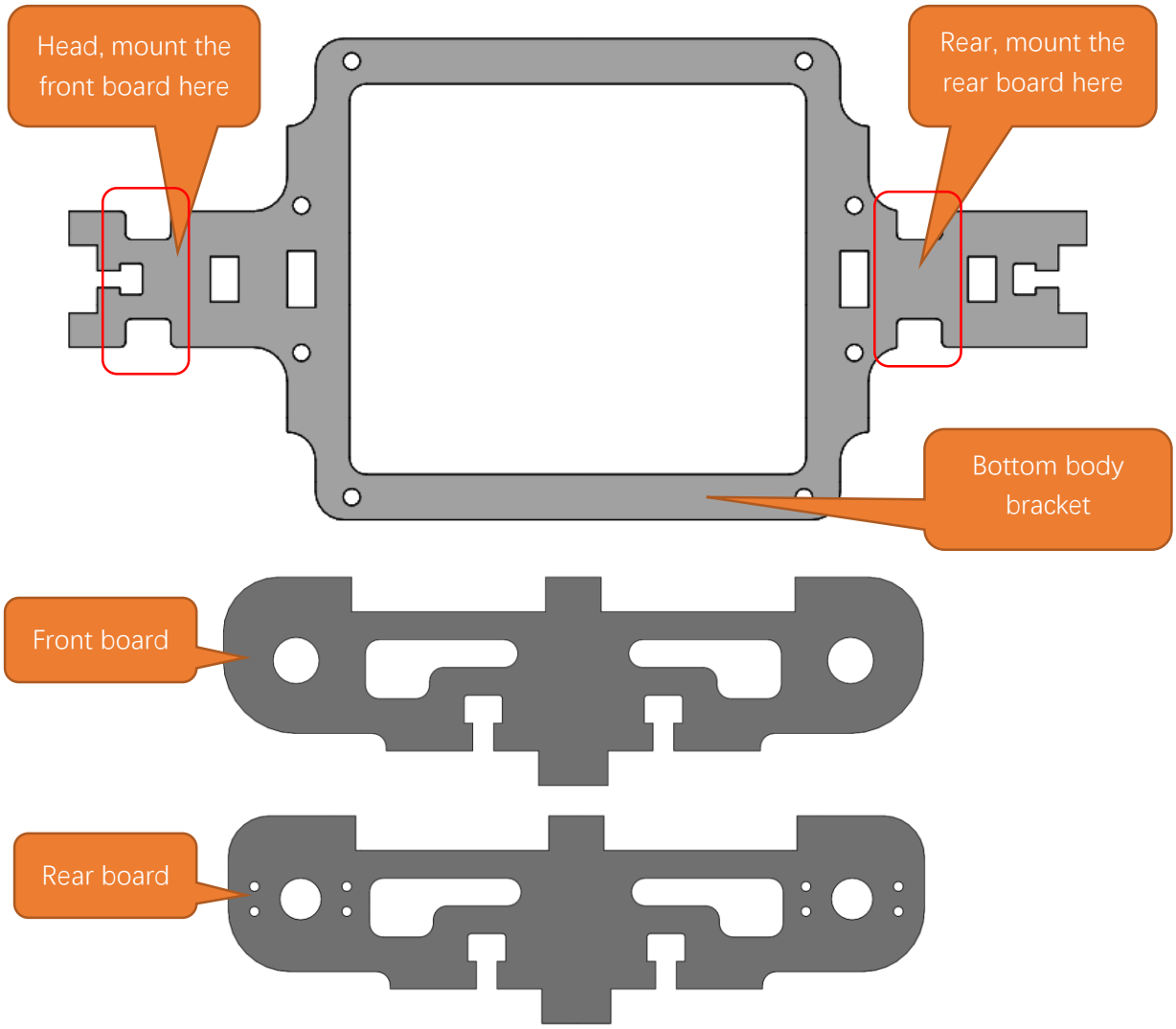


5. Mount two disc servo arms with eight M1.4*8 screws to the rear board.

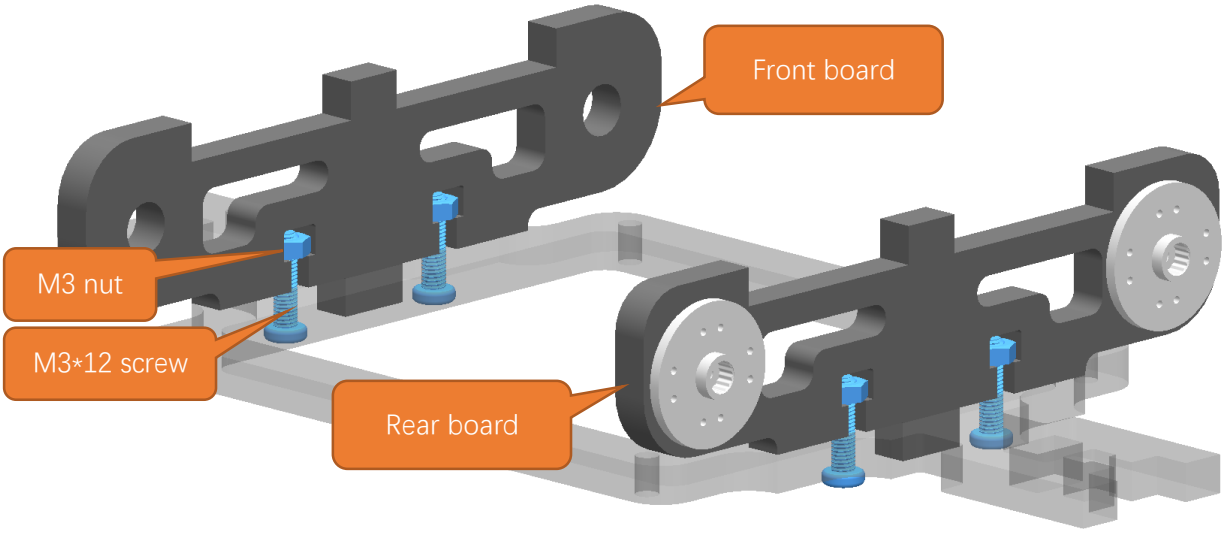


Step 2 Assembly of Body Bracket

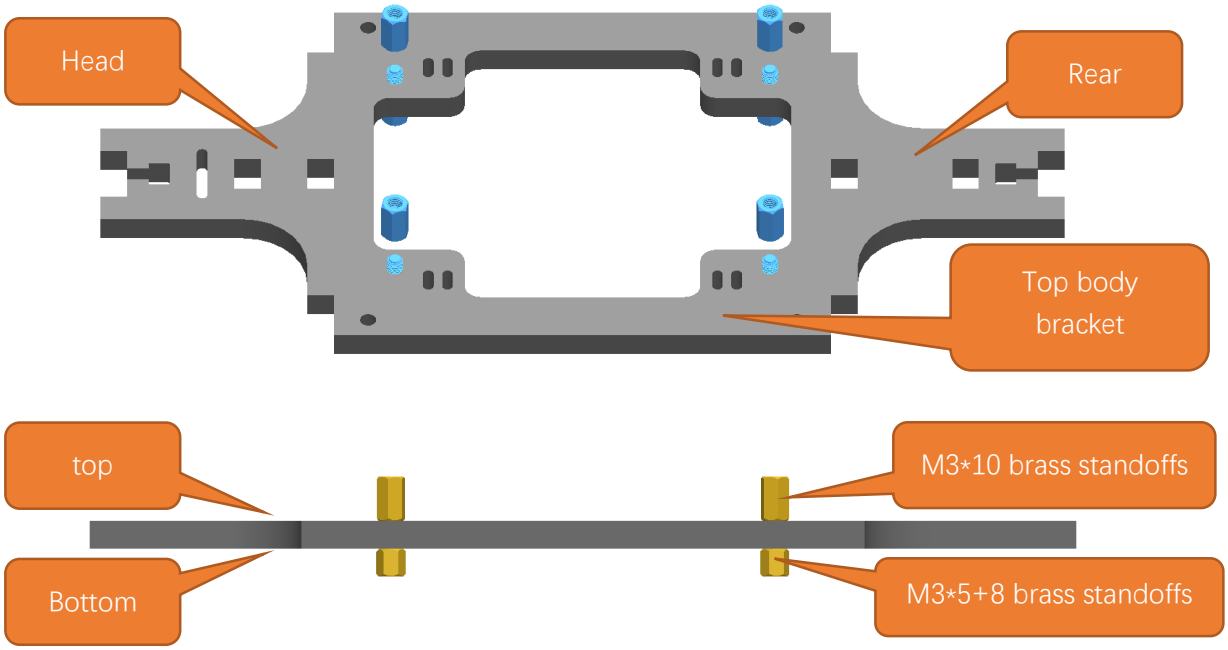
1. Mount the below two acrylic parts to the bottom body bracket with M3*12 screws and M3 nuts.



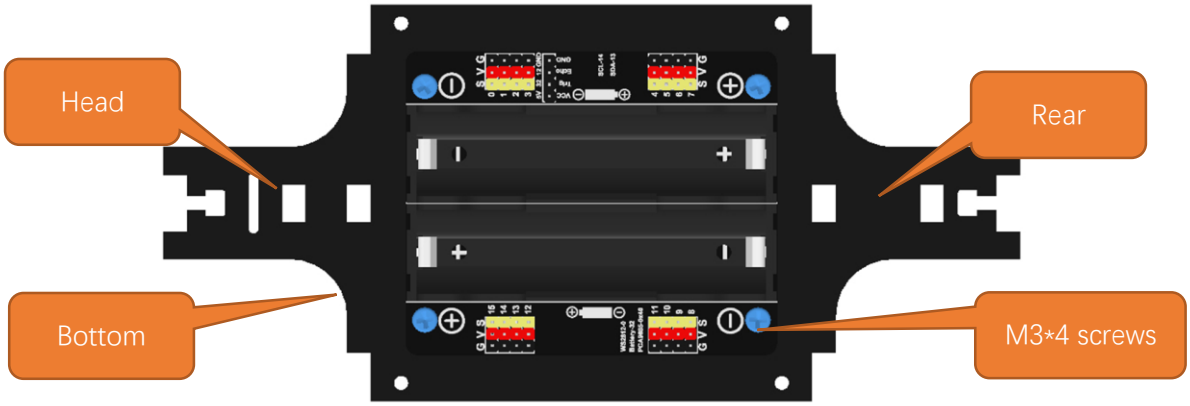
It should look as below after assembly. Pay attention to the direction of the rear board.



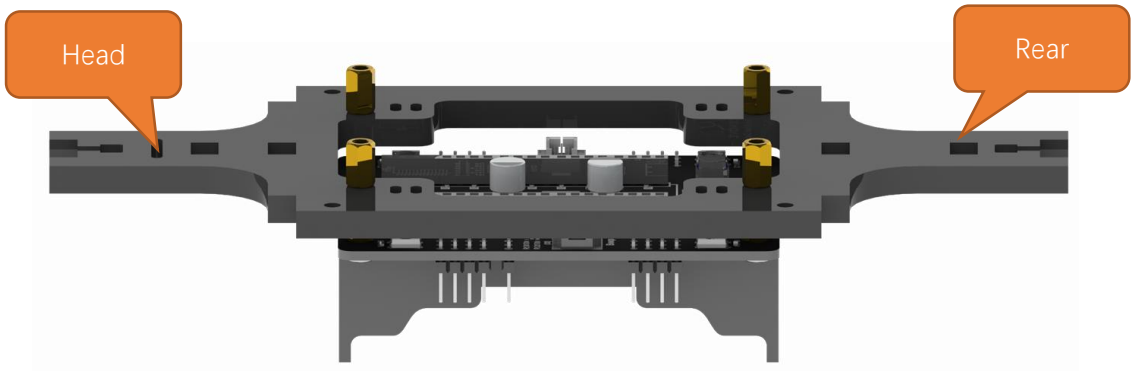
2. Mount four M3*5+8 and four M3*10 brass standoffs to the top body bracket.

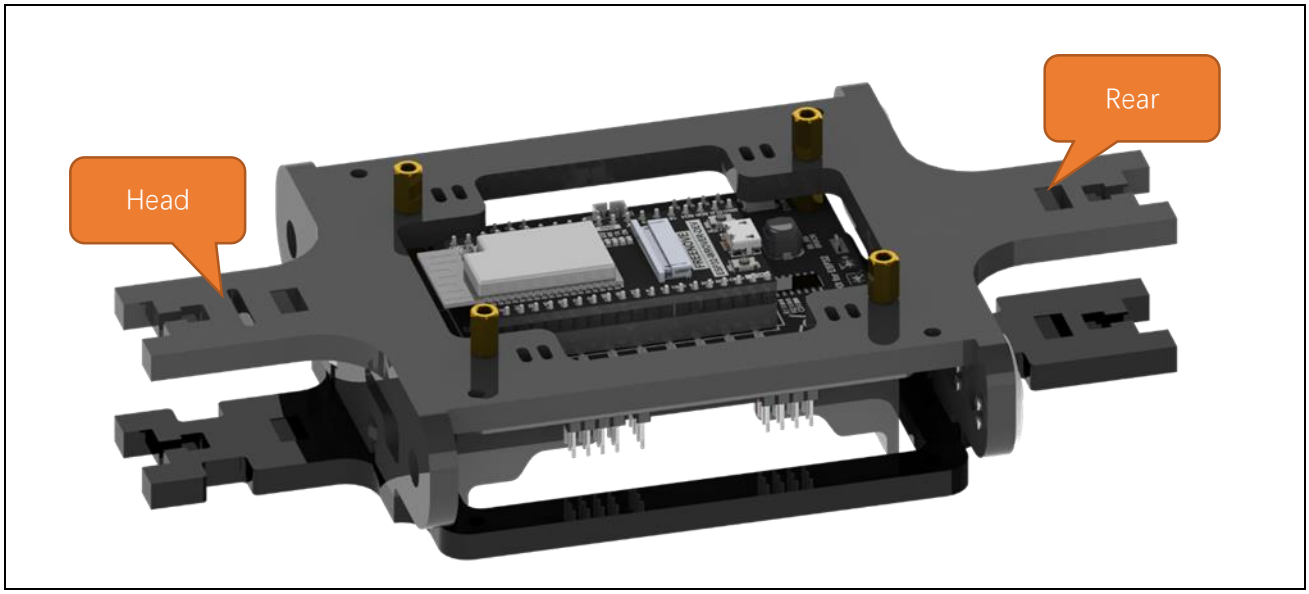


3. Mount the board to the top body bracket with four M3*4 screws.



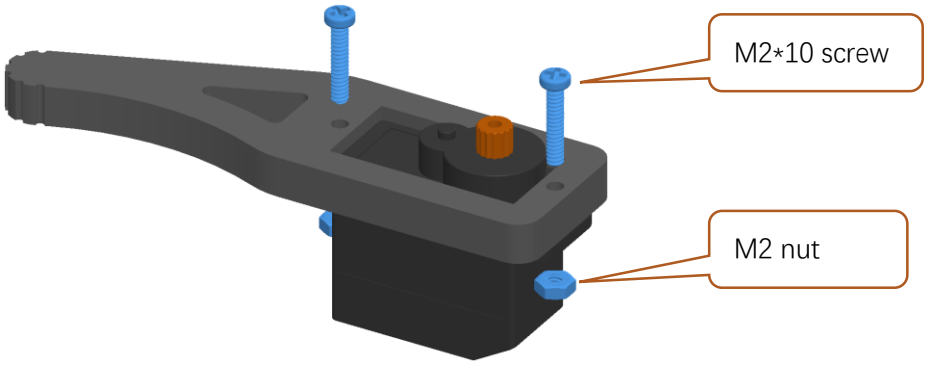
After assembly:



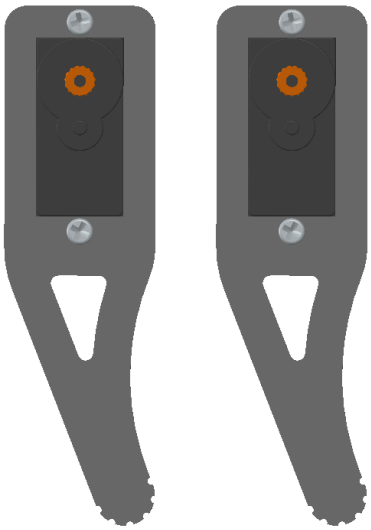

Head
Rear

Step 3 Assembly of Legs

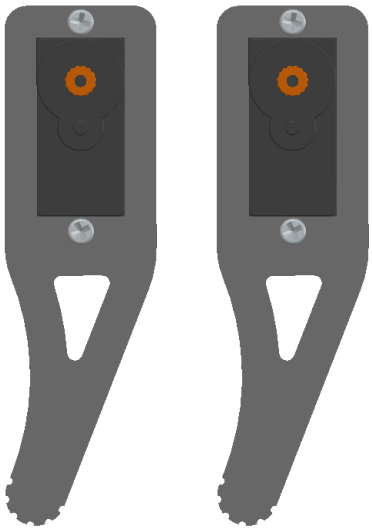
1. Mount four servos to four shanks with M2*10 screws and M2 nuts.



Left shank

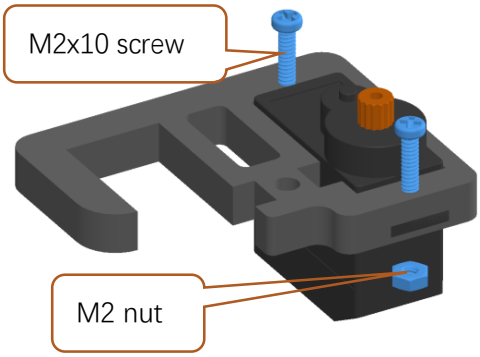


Right shank

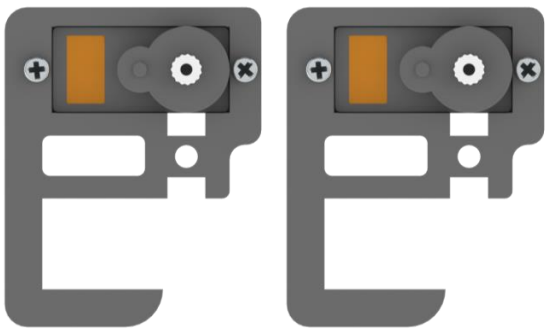


2. Mount two servos to the below acrylic parts with M2*10 screws and M2 nuts.

Assembly diagram

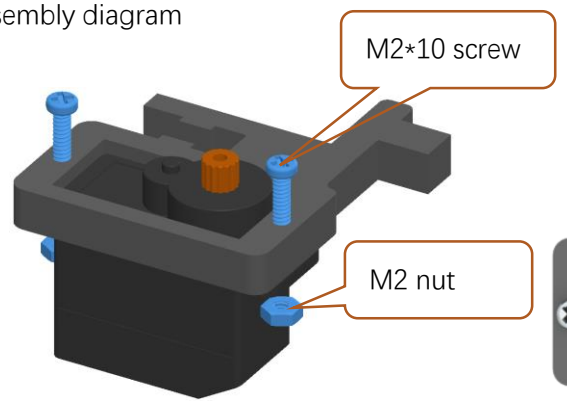


After assembly

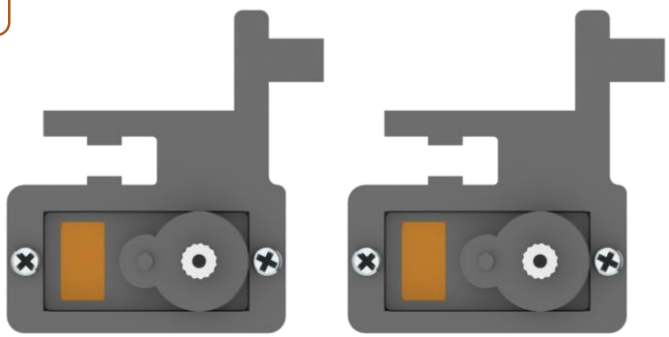


3. Mount two servos to the below acrylic parts with M2*10 screws and M2 nuts.

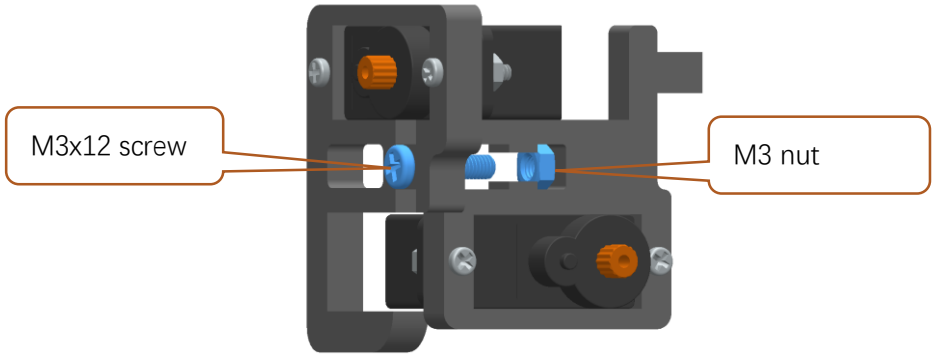
Assembly diagram



After assembly



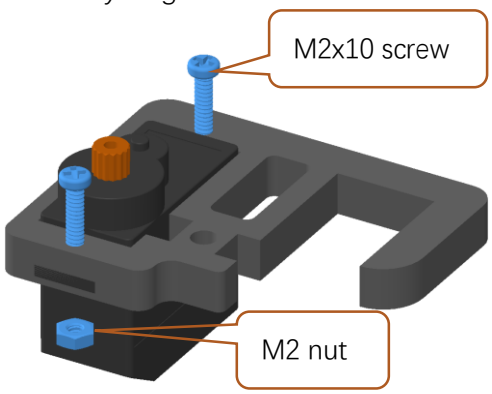
4. Mount the two set of acrylic parts above with M3*12 screws and M3 nuts.



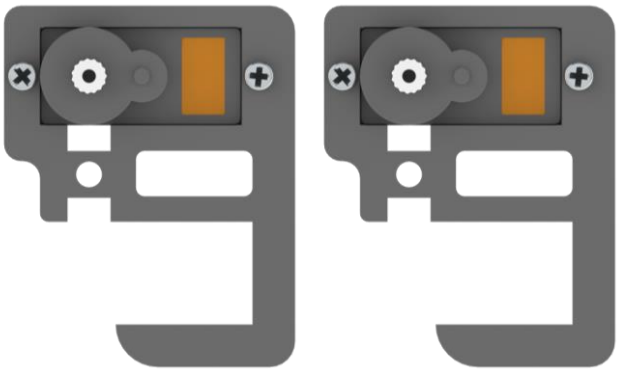
5. Mount two servos to the below acrylic parts with M2*10 screws and M2 nuts.

Note: The direction is different from that in No.2!

Assembly diagram



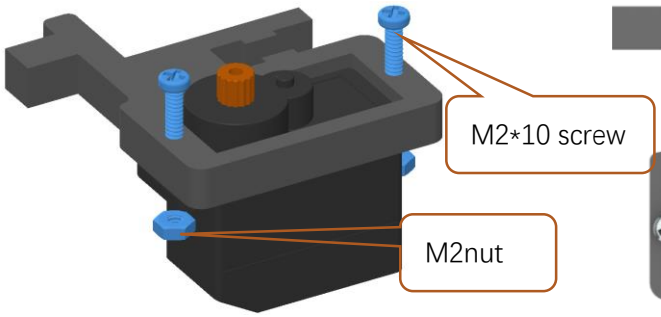
After assembly



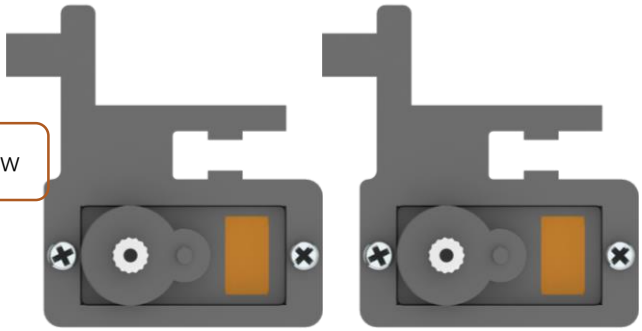
6. Mount two servos to the below acrylic parts with M2*10 screws and M2 nuts.

Note: The direction is different from that in No.3!

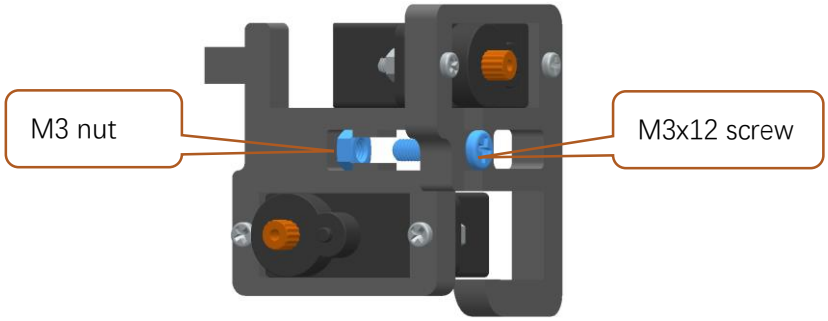
Assembly diagram



After assembly

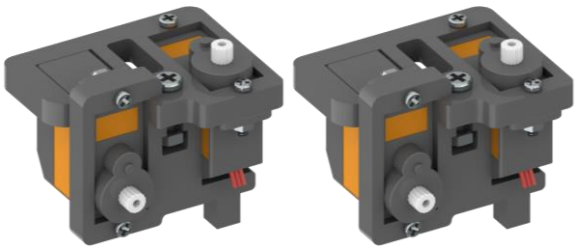


7. Mount the two set of acrylic parts abovw with M3*12 screws and M3 nuts.。

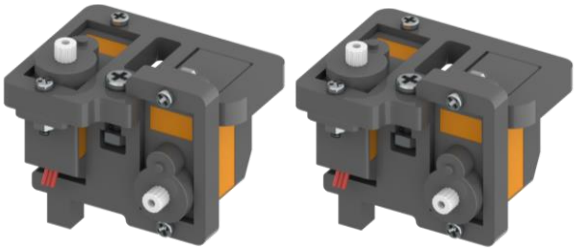


Note: The direction is different from that in No.4!

Left

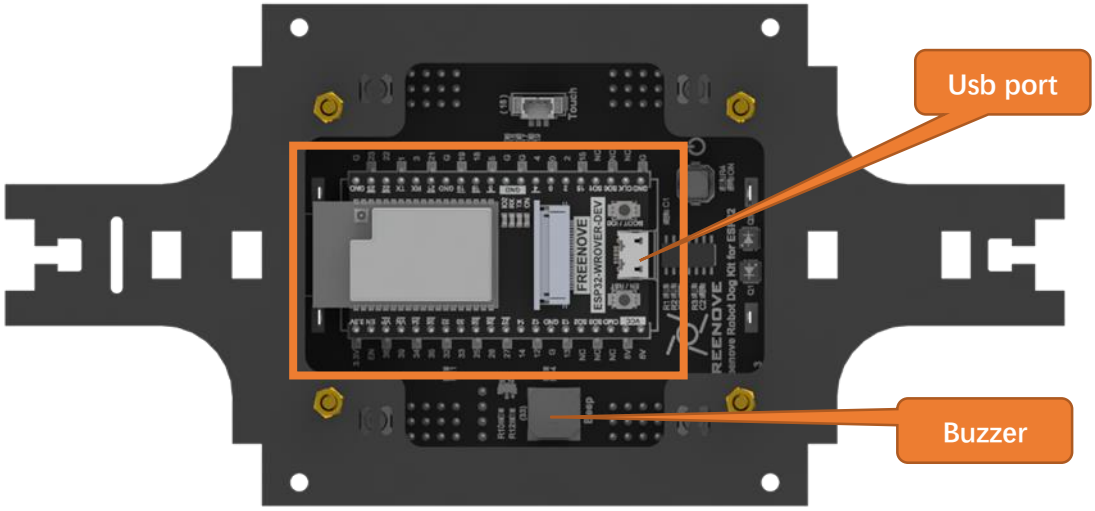


Right



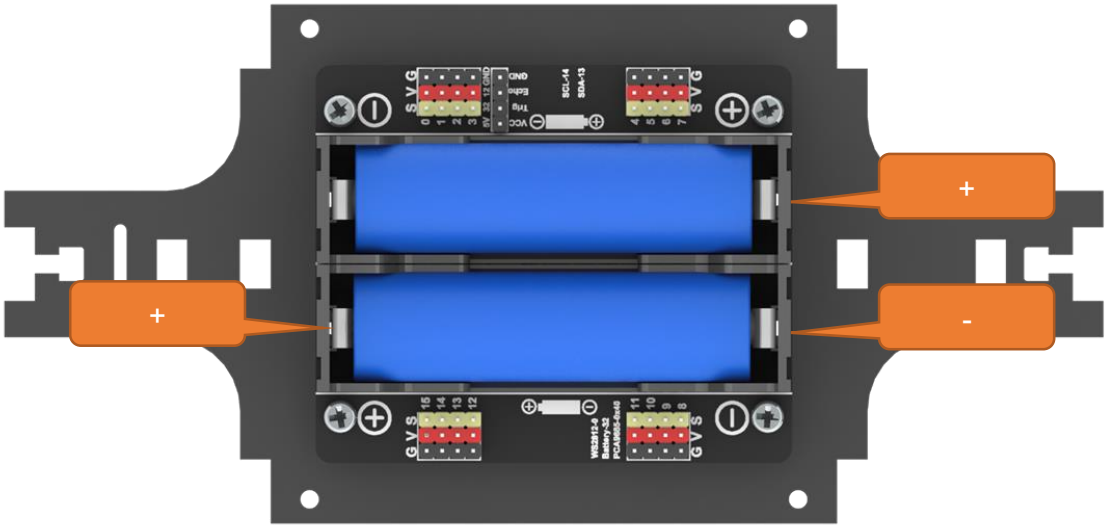
Step 4 Adjustment of Servo Angles

1. Plug ESP32 into the driver board.



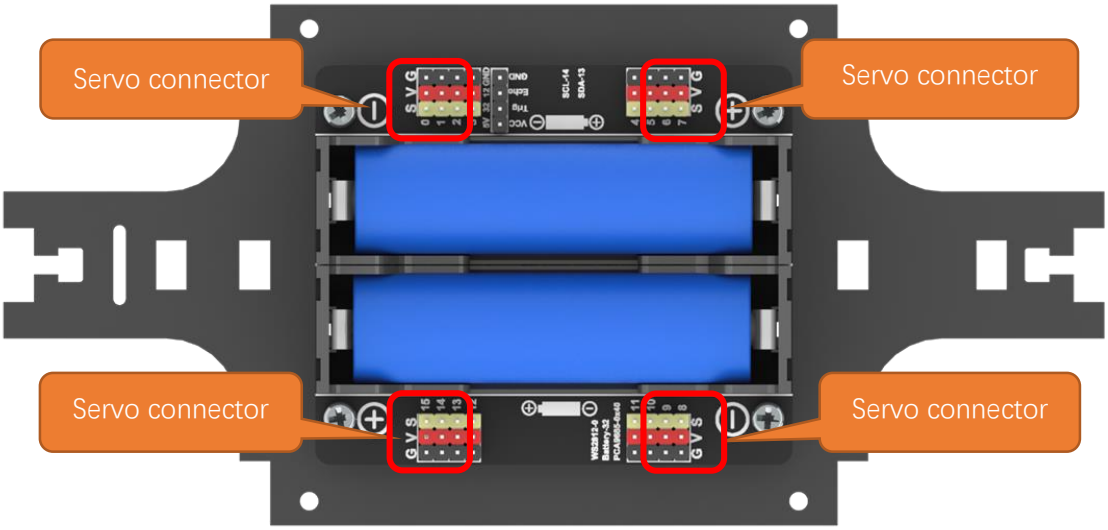
Pay attention to the orientation of the ESP32. Wrong installation may lead to damage.

2. Install the batteries.

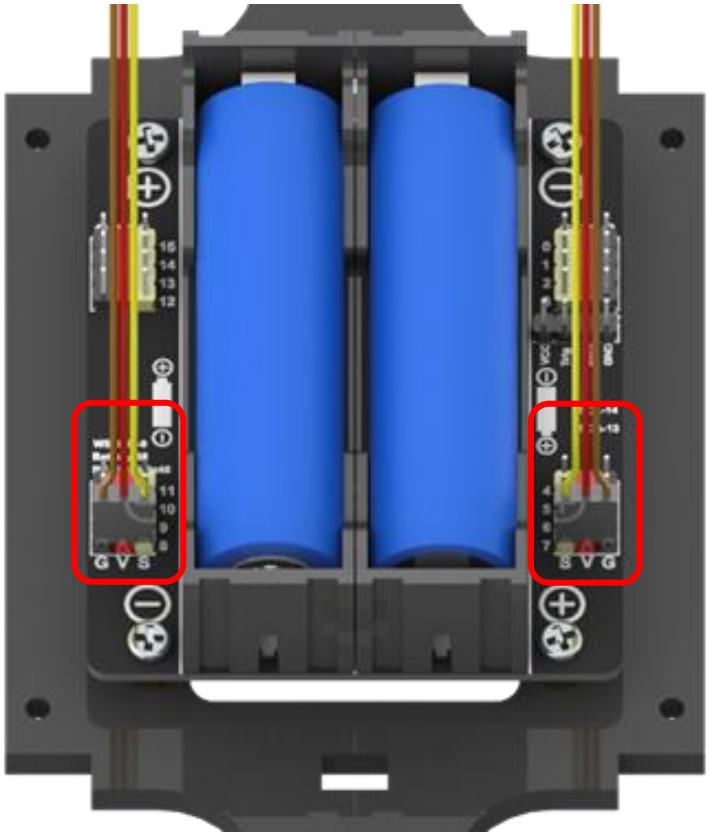


Put in the batteries according to the silkscreen. Wrong installation may result in malfunction.

3. Plug in the servo cables to the servo connector pins randomly.



Pay attention to the color of the cable on each side.



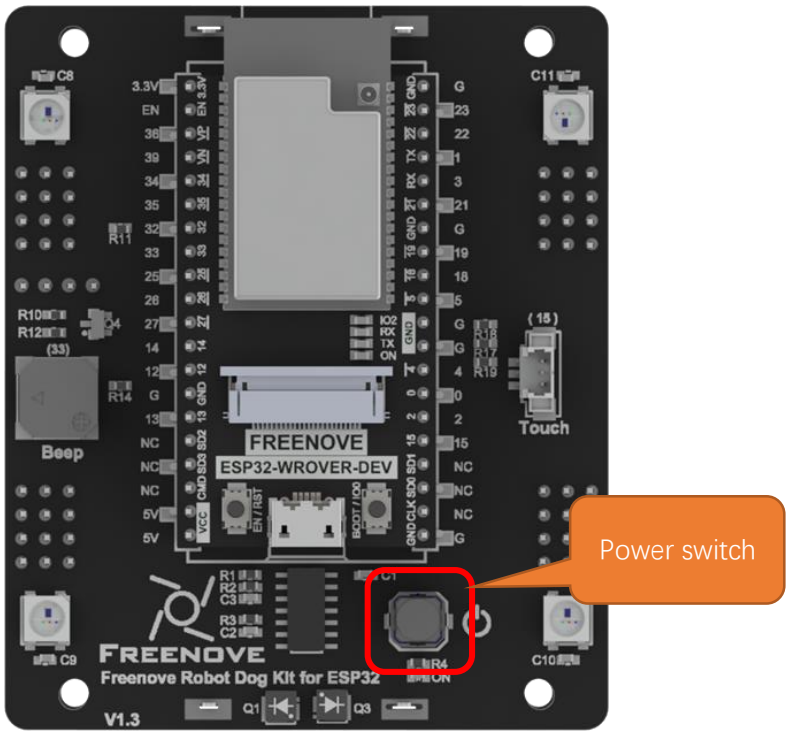
G-GND (brown cable)

V-VCC (red cable)

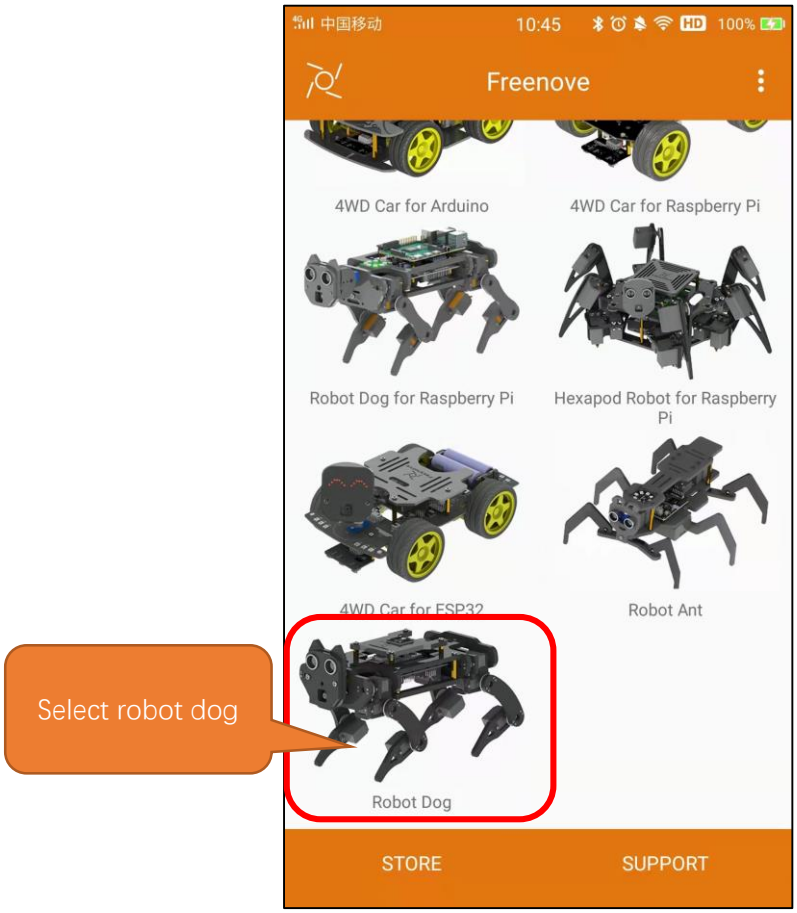
S-Signal (orange cable)

Note: Do NOT connect the cables in reverse. Otherwise, the servos will not work and be damaged.

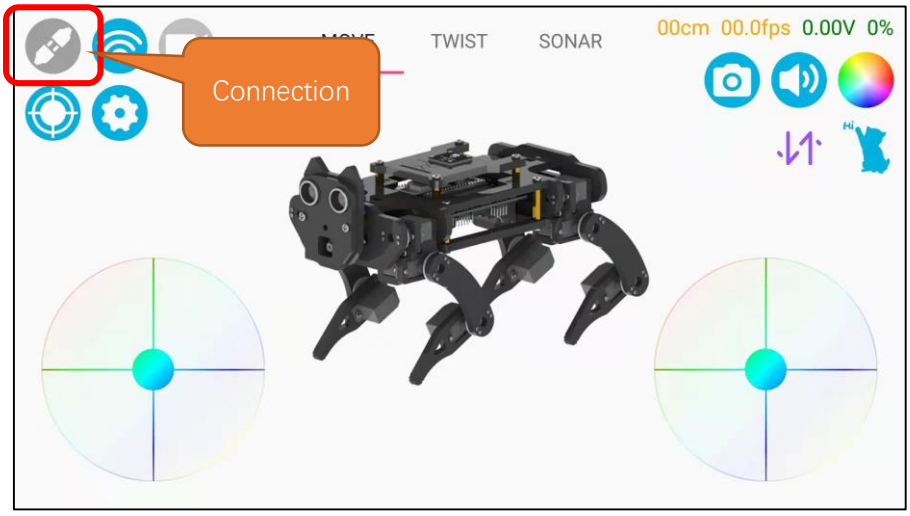
4. Turn ON power switch.



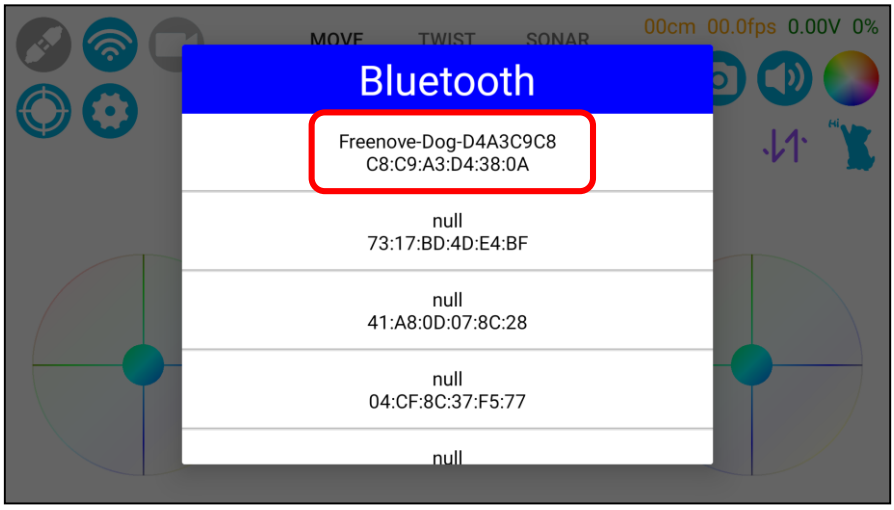
5. Open Freenove App and select robot dog.



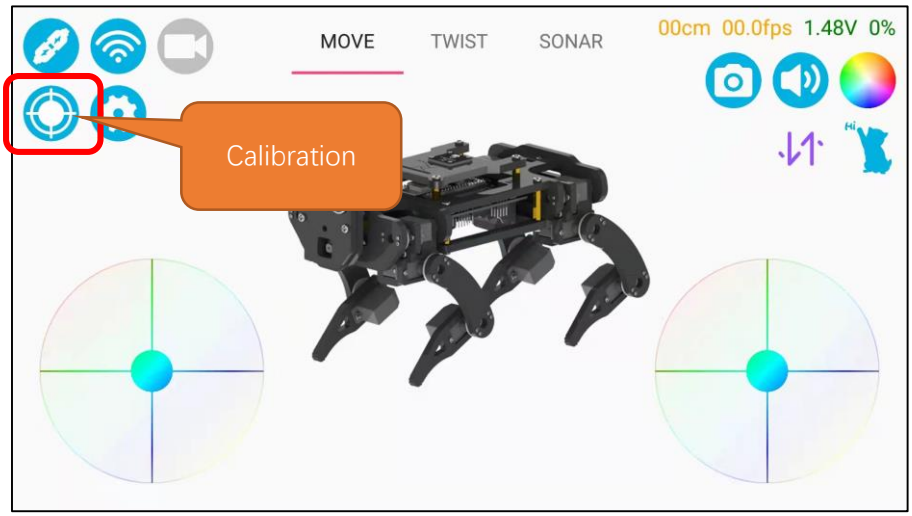
6. Turn ON Bluetooth of your phone and tap the Connect button on the app.



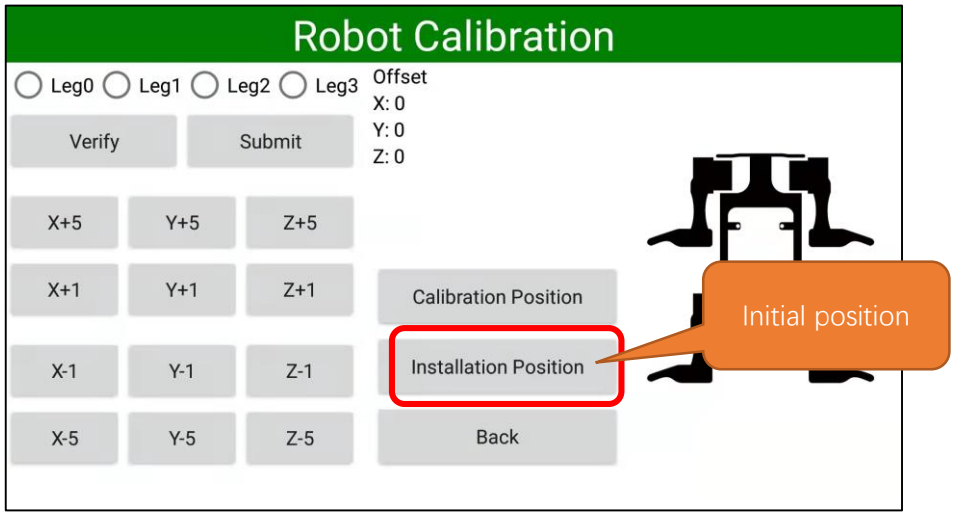
7. Select “Freenove-Dog-XXXXXXXX”.



8. Tap the calibration button.



9. Tap Installation Position and all the servos will rotate to 90 degrees.



Note: The purpose of adjusting the angle of the servos is to ensure that there will not be too much deviation after assembly to avoid malfunction. Therefore, this step is very important.

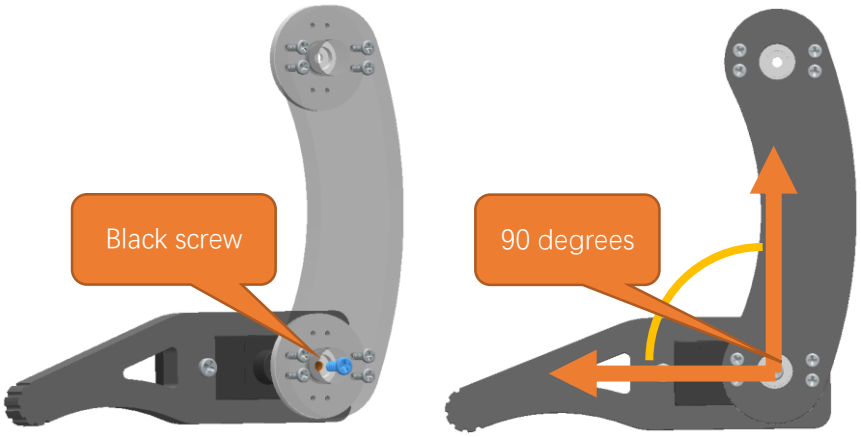
Step 5 Assembly of Legs to Body

Please keep power ON and all the servo cables connecting to the board during assembly.

We need to ensure servos remain at 90 degrees when assembling. (Servo cables are not shown in the following instructions.)

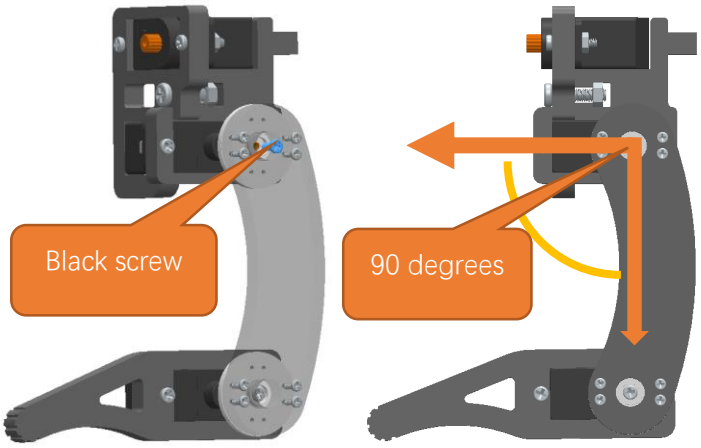
1. Assemble the servos to the acrylic parts with **black screws in servo packages.**

Assemble them as close to 90 degrees as possible. Angles at 70 – 110 degrees are acceptable.

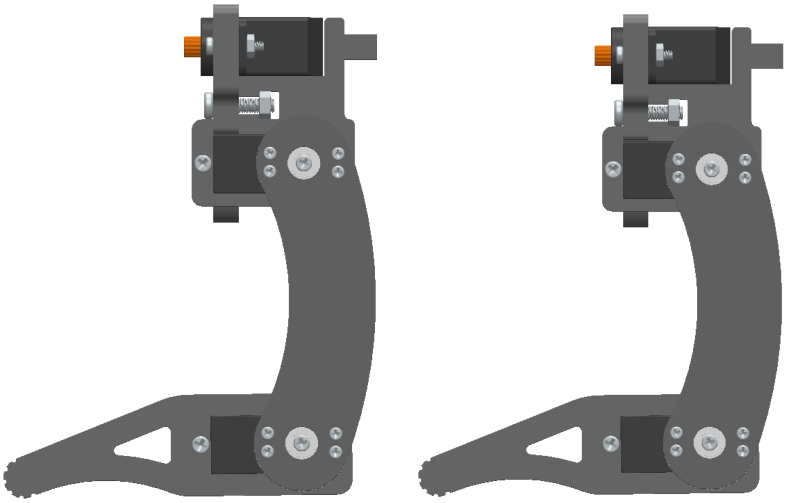


2. Assemble the servos to the acrylic parts with black screws in servo packages.

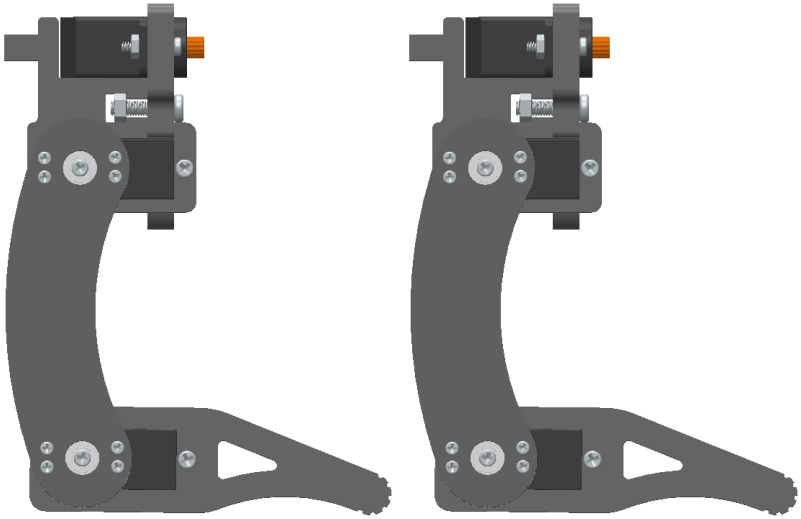
Assemble them as close to 90 degrees as possible. Angles at 70 – 110 degrees are acceptable.



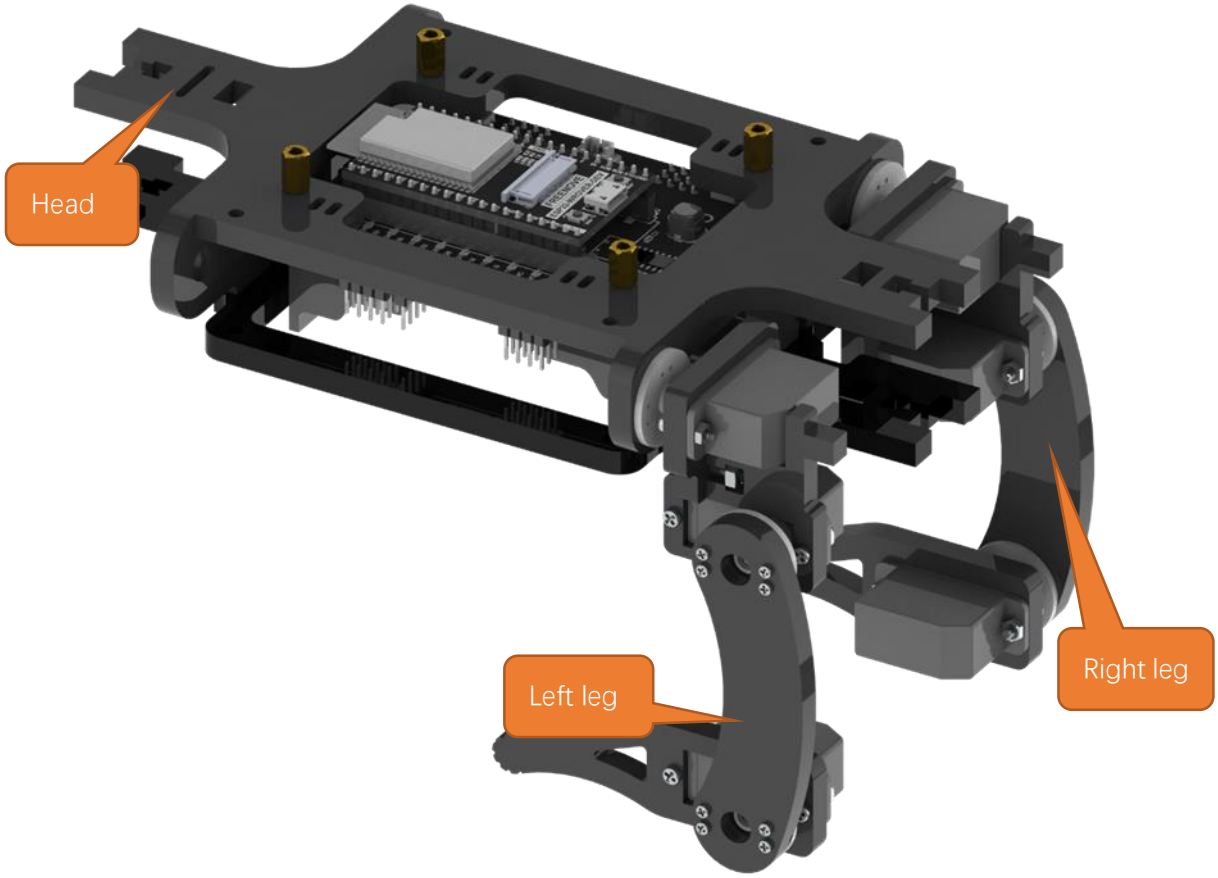
Repeat the above step to make two same left legs.



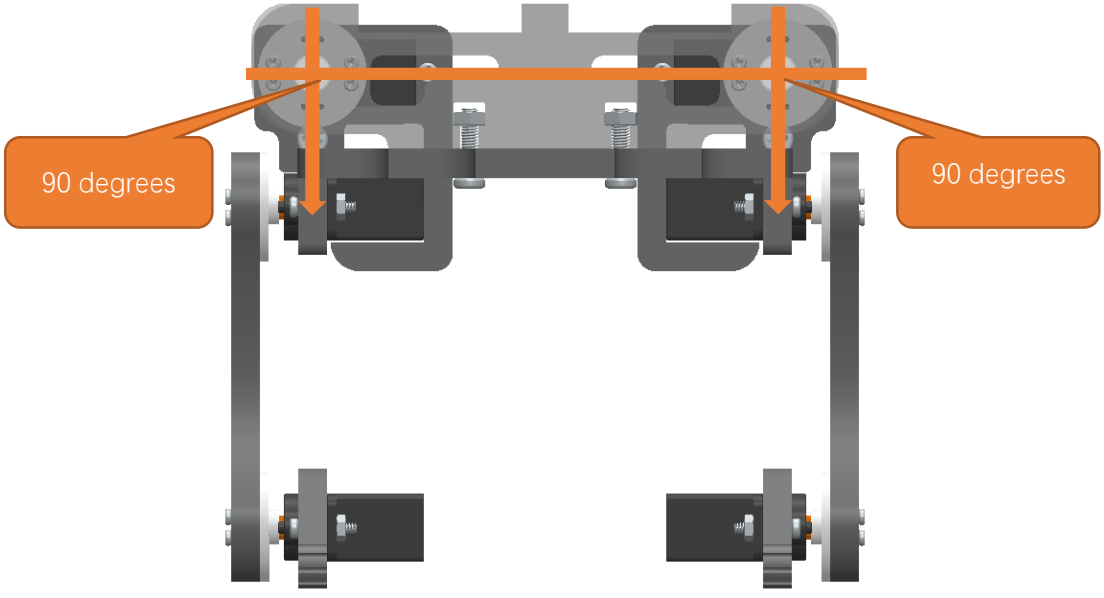
3. Similar to the above steps, assemble two right legs.



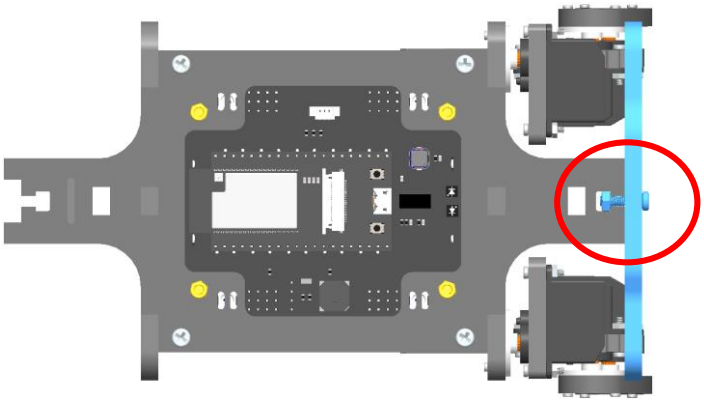
4. Mount one left and one right legs to the rear board with black screws in the servo package.

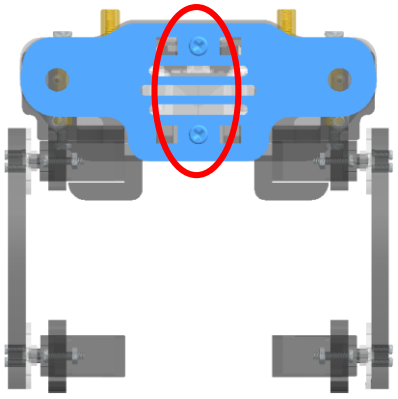


Assemble them as close to 90 degrees as possible. Angles at 70 – 110 degrees are acceptable.

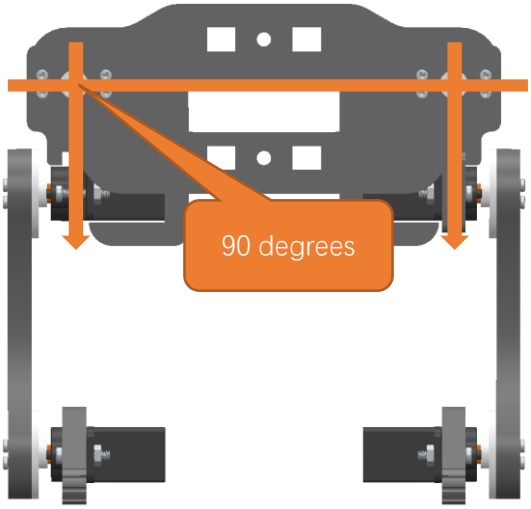


5. Mount the rear board to the rear of the brackets with two M3*12 screws and two M3 nuts.



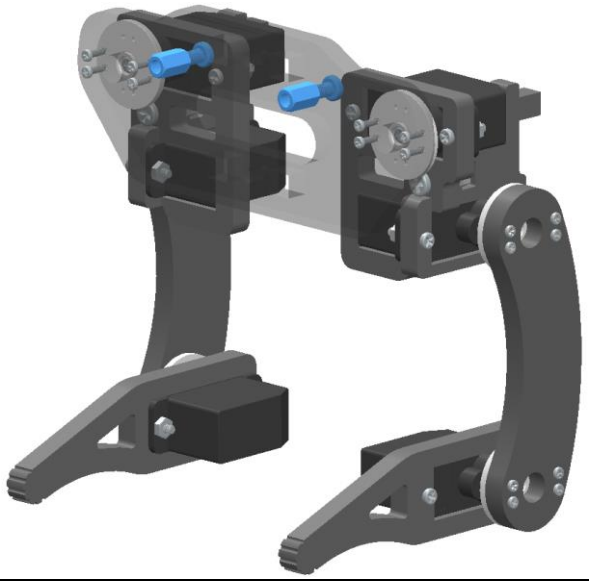


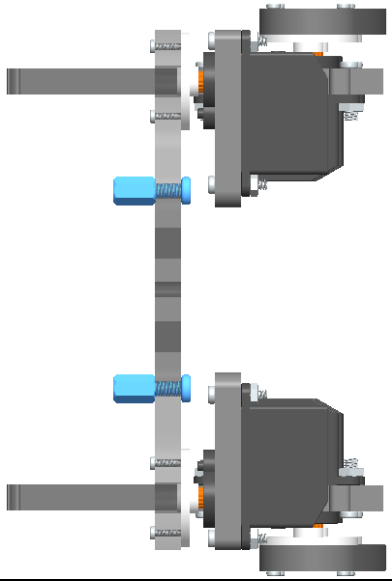
6. Mount one left and one right leg to the front board with black servo screws in the servo package. Assemble them as close to 90 degrees as possible. Angles at 70 – 110 degrees are acceptable.



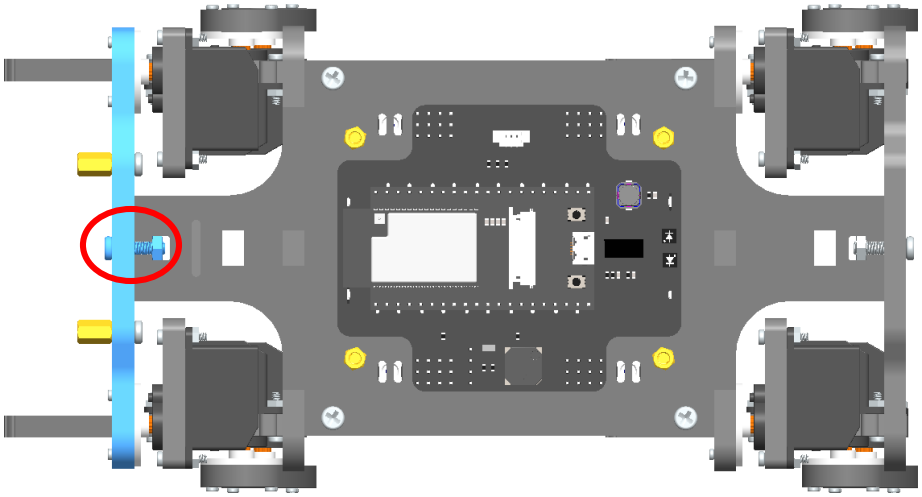
Note: After installing all Servo at 90 degrees, power can be turned off.

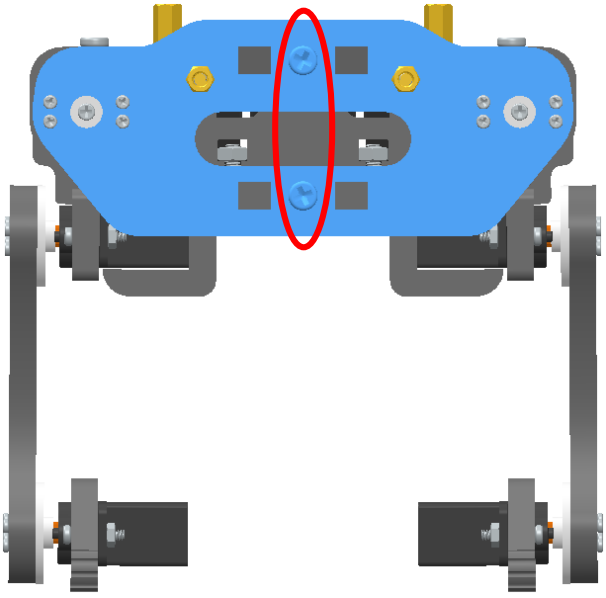
7. Mount two M3*10 standoffs to the front board with two M3*8 screws.



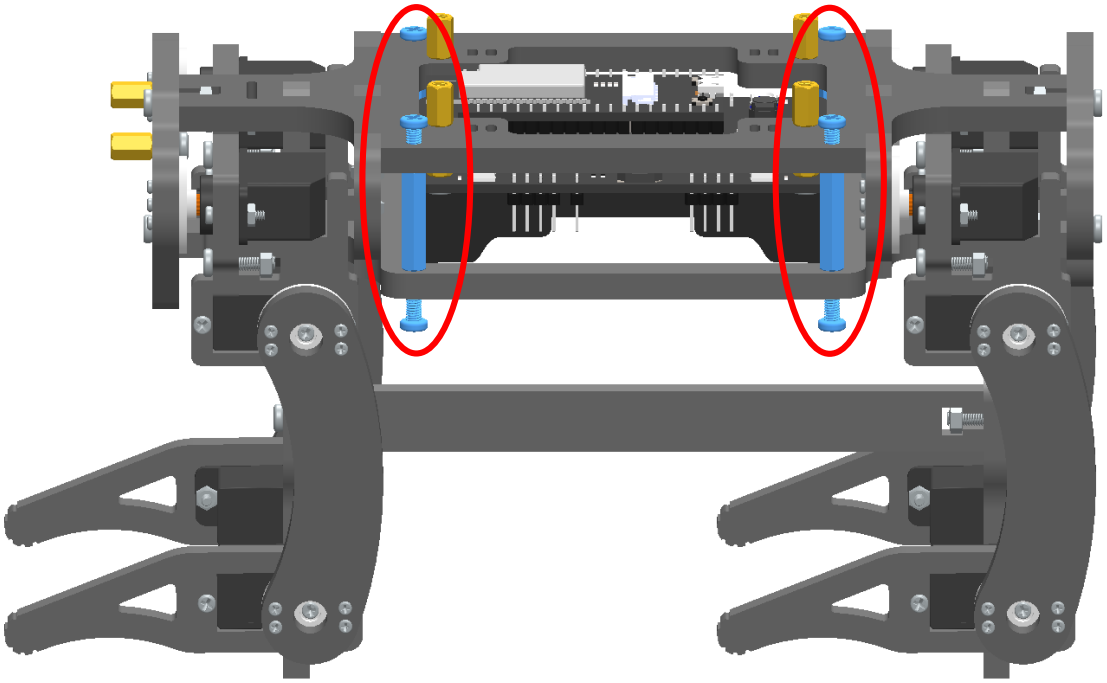


8. Mount the front board to the head of body bracket with a M3*12 screw and a M3 nut.



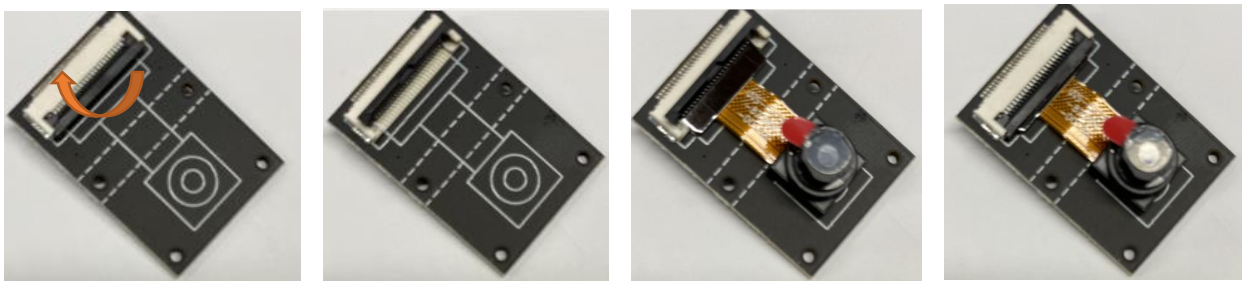


9. Fix the body brackets with eight M3*8 screws and four M3*20 brass standoffs.



Step 6 Assembly of Head and Wire

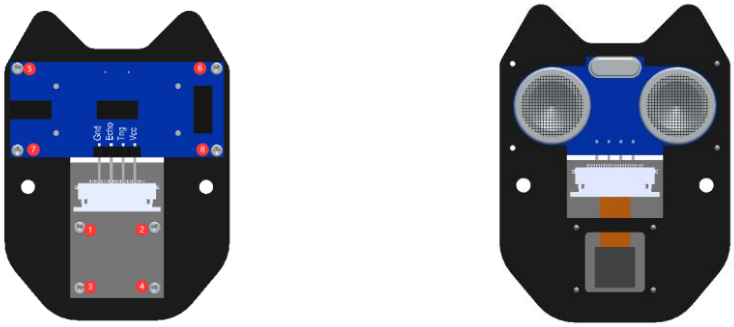
1. Connect the camera to its extension board.



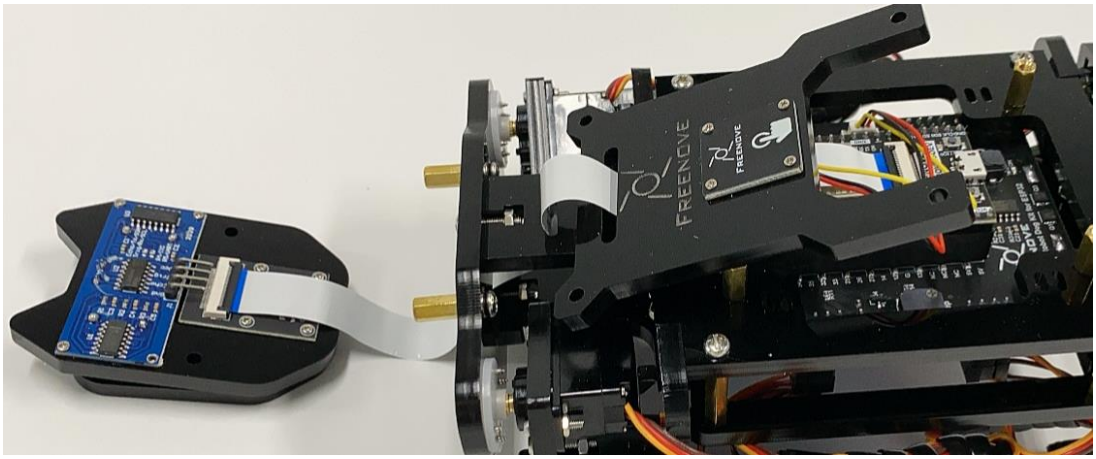
Gently pull up the lock with your fingernail or a plastic stick.

Remember to **keep power OFF** when assembling the head to avoid damaging the camera.

2. Mount the camera and ultrasonic modules to the head acrylic part.



3.



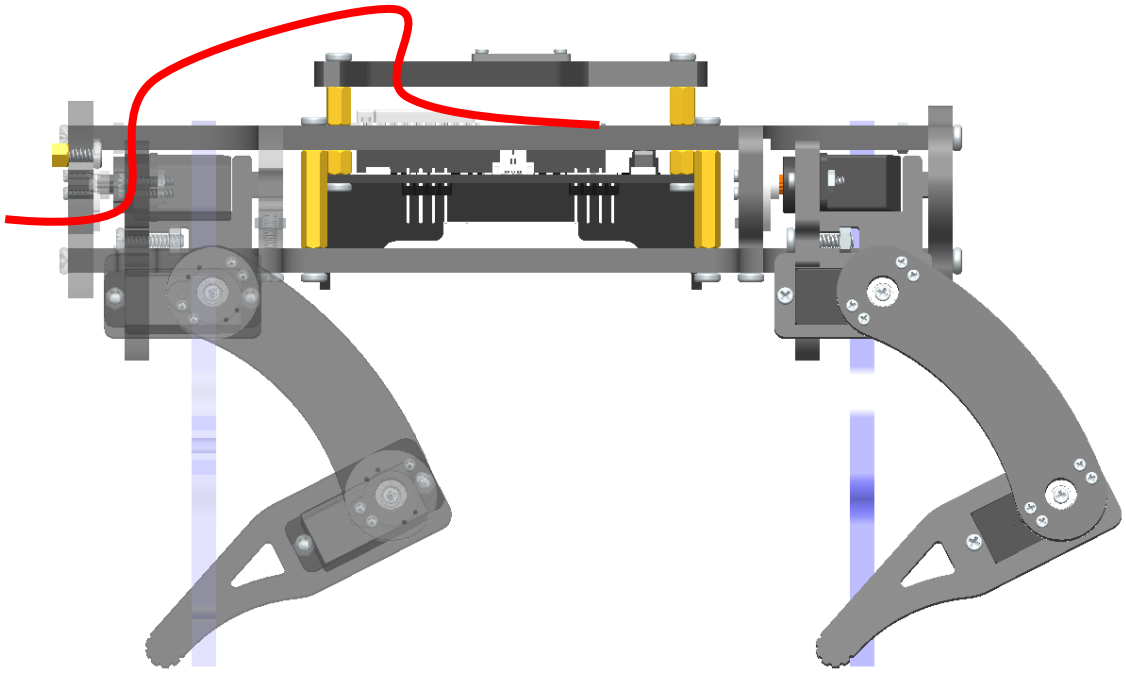
Plug one end of the camera cable into the camera extension board. **Pay attention to blue side.**

√ ×

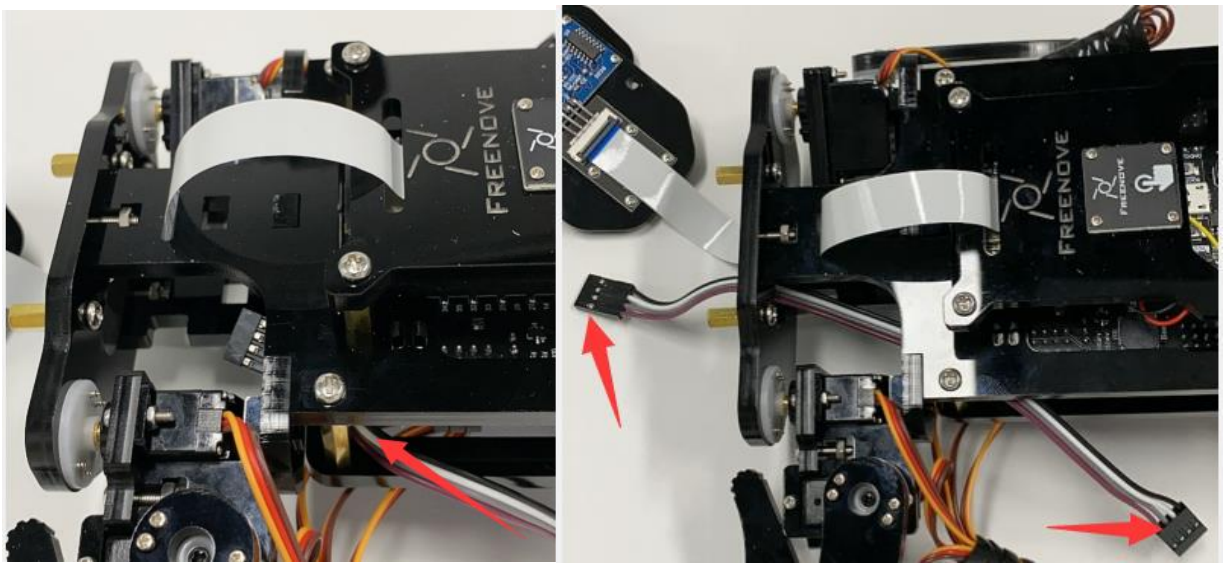
Plug the other end into the ESP32.

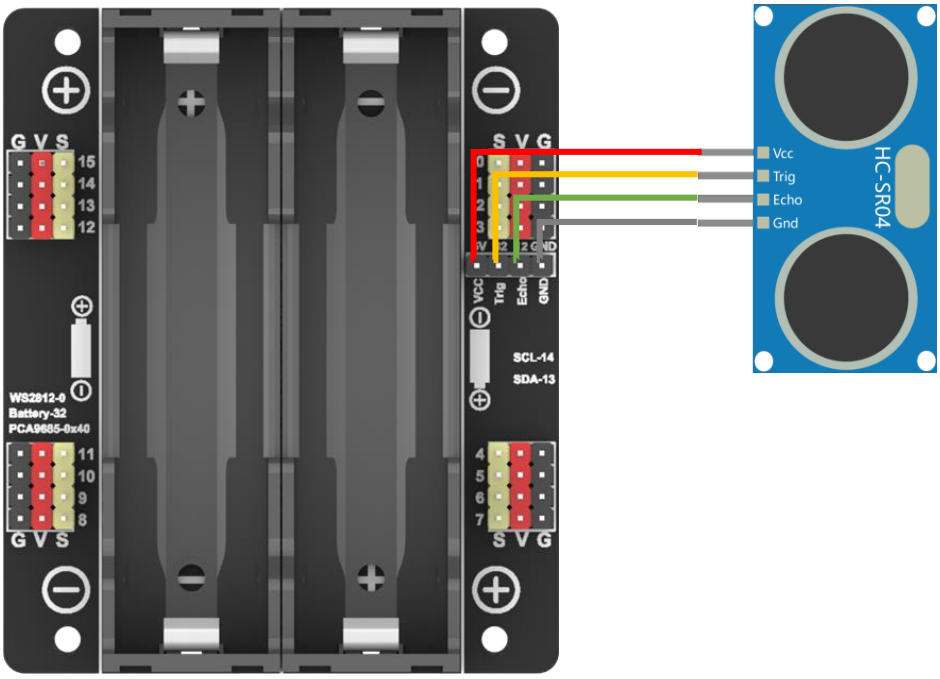
√

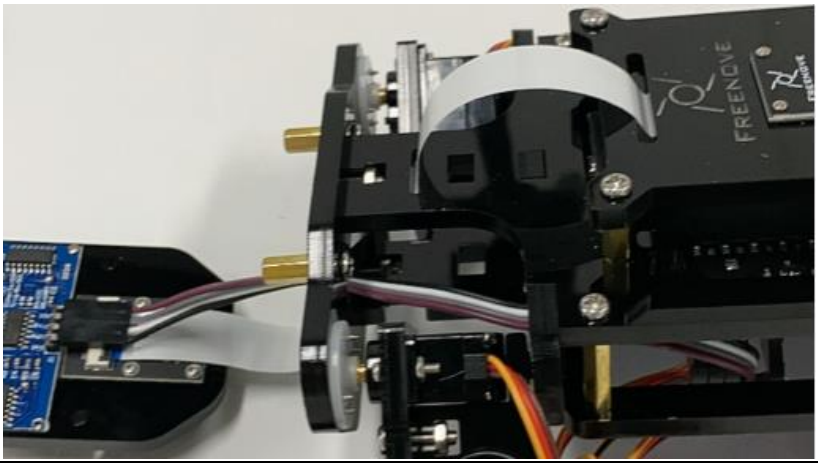
×



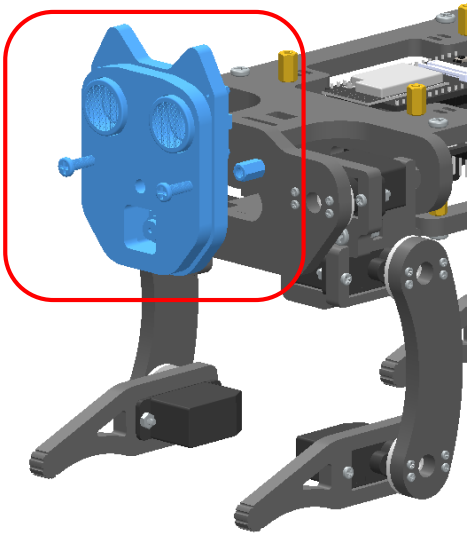
4. Connect the ultrasonic module to driver board with the 4P DuPont cable.





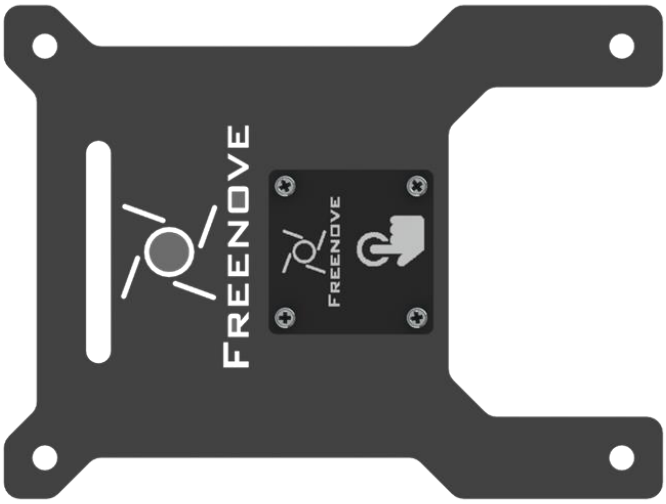


5. Mount the two acrylic parts for head to the body with two **M3*12** screws.

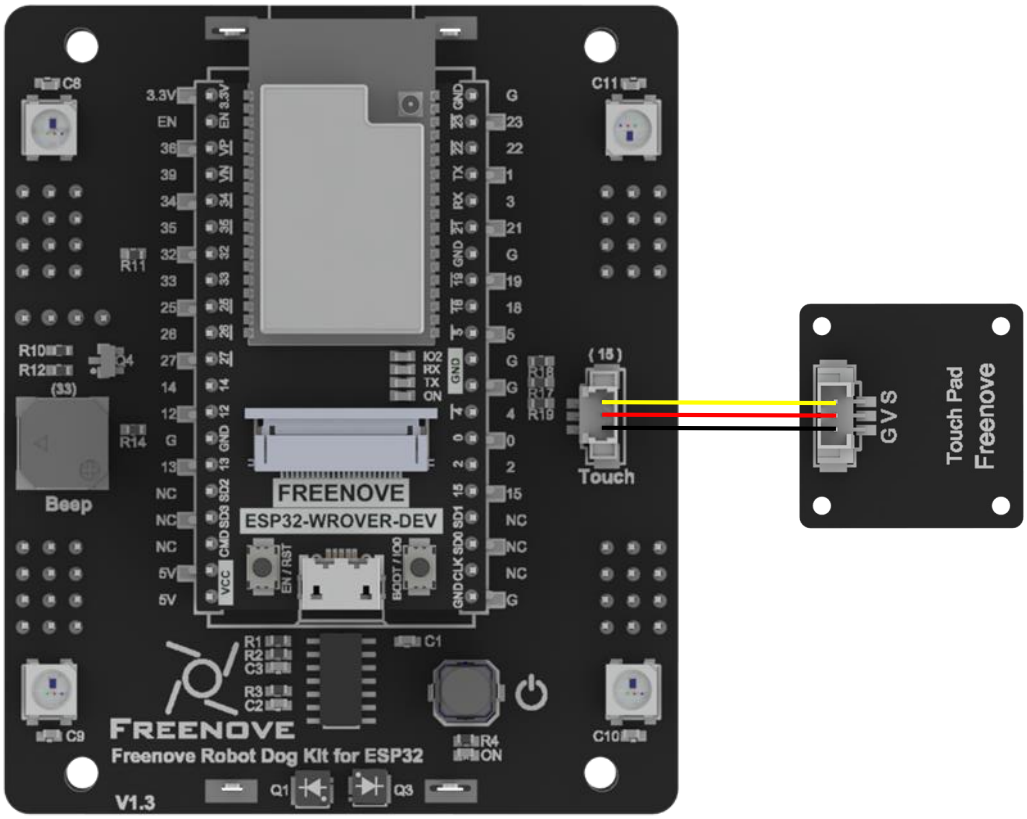


Step 7 Assembly of the Cover

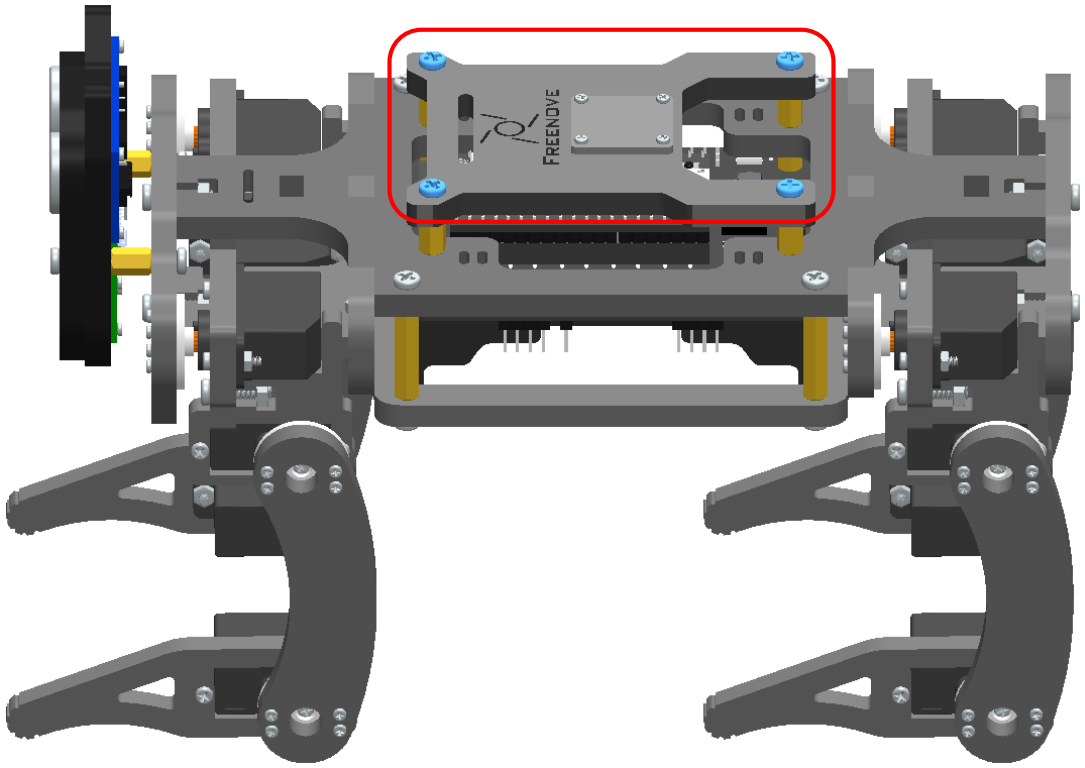
1. Mount the touch sensor to the cover with four M1.4*5 screws.



2. Use a 3P wire to connect the touch module to the robot dog drive board. The figure below does not show the acrylic part.

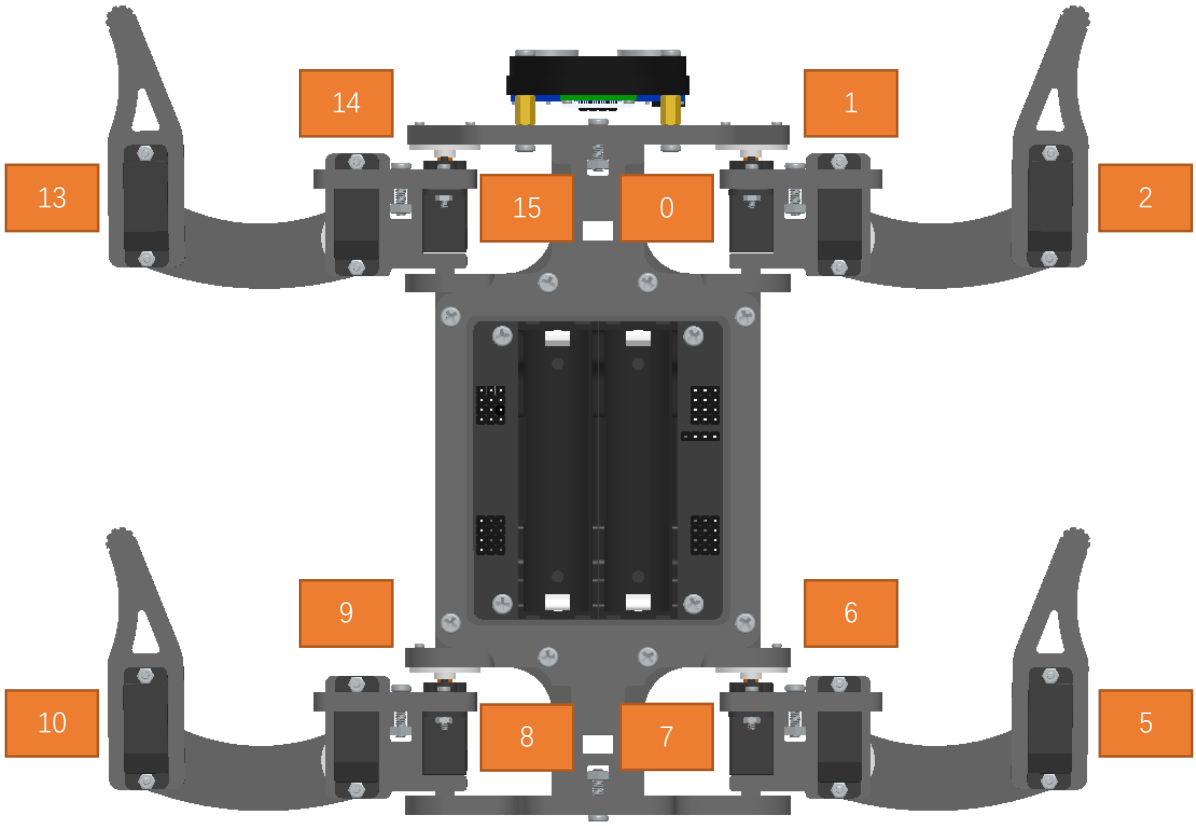


3. Mount the cover to the robot with four M3*8 screws.



Step 8 Servo Wiring

Reconnect the servo cables in accordance with the sequence below.

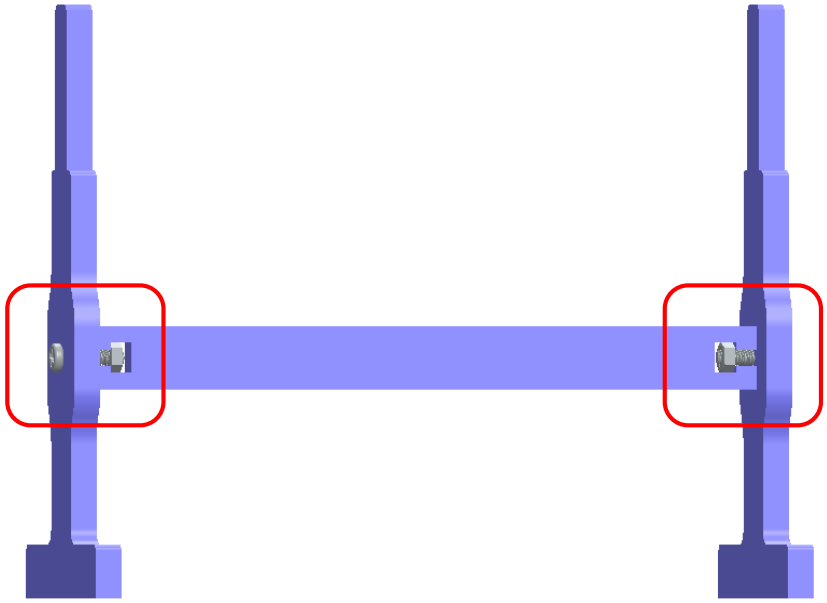


G-GND (brown cable)
V-VCC (red cable)
S-Signal (orange cable)
Note: Servo ports 3, 4, 11 and 12 are not connected by default.

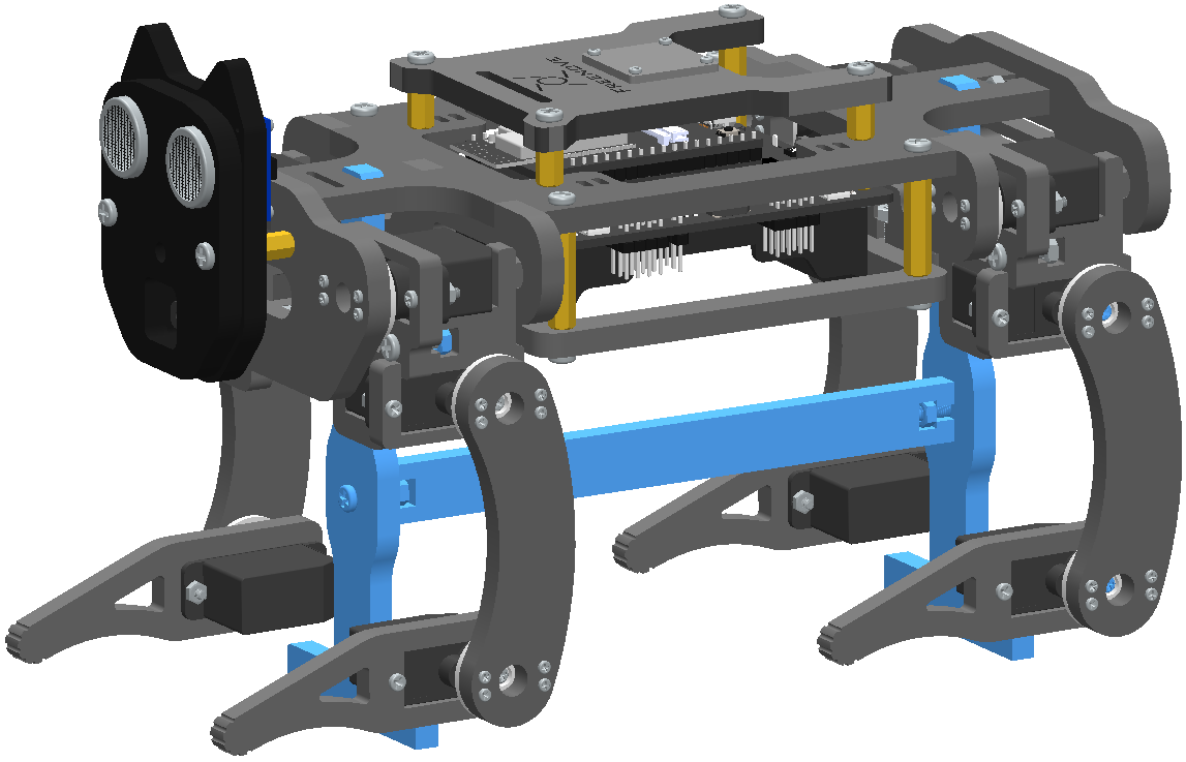
At this point, the robot dog has been assembled and can walk, but because it has not been calibrated, it cannot walk properly. Calibration is a very important task that determines whether your robot dog can walk perfectly. Please be patient with the next steps.

Step 9 Assembly of Calibration Bracket

1. Assemble the calibration bracket with two M3*12 screws and two M3 nuts.

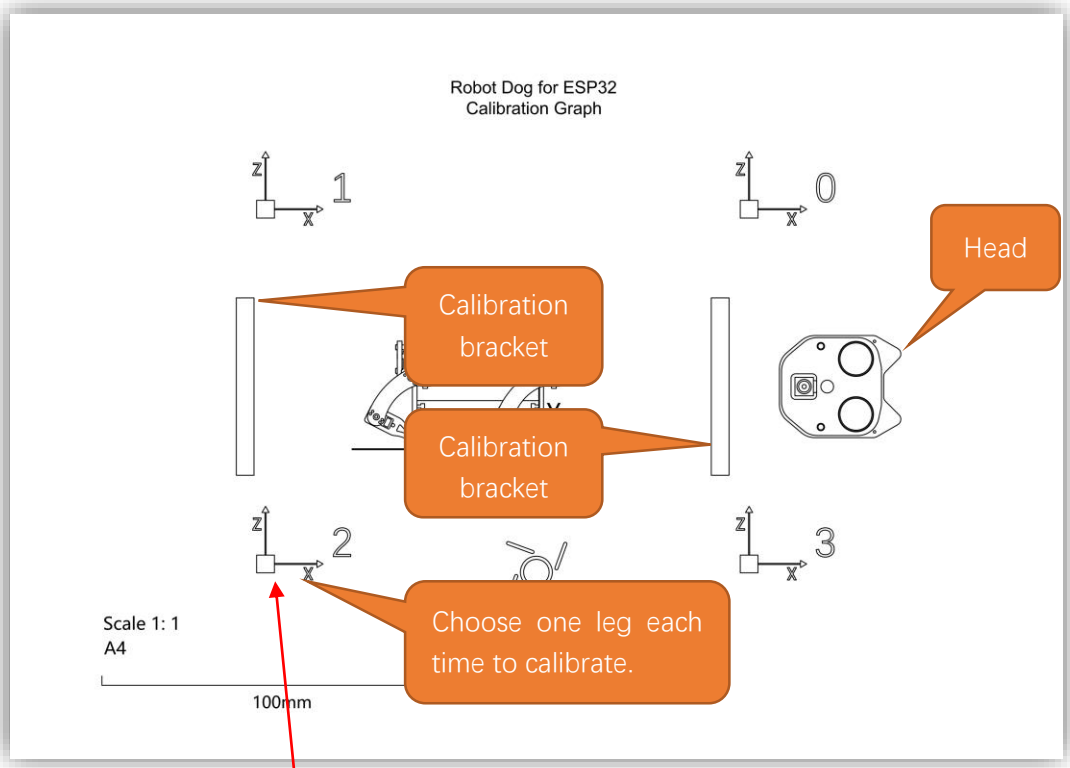


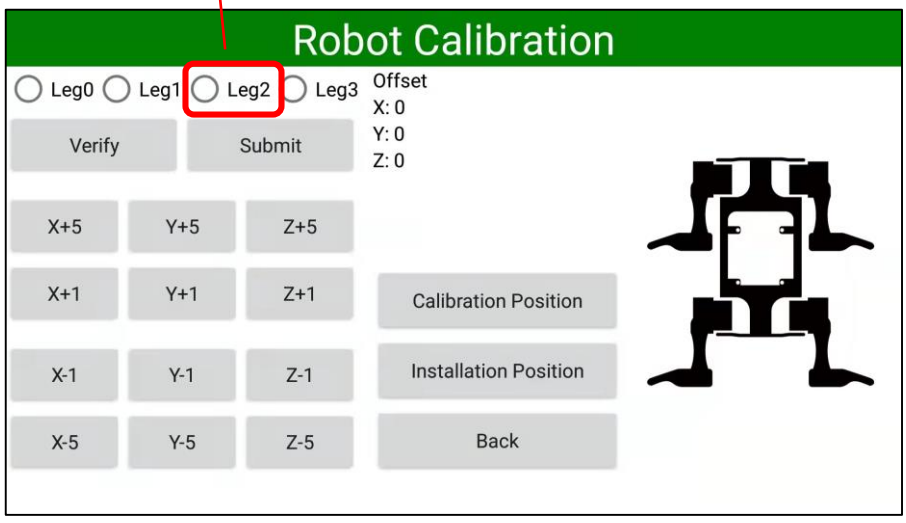
2. Insert the calibration bracket into the robot dog to suspend the four legs.



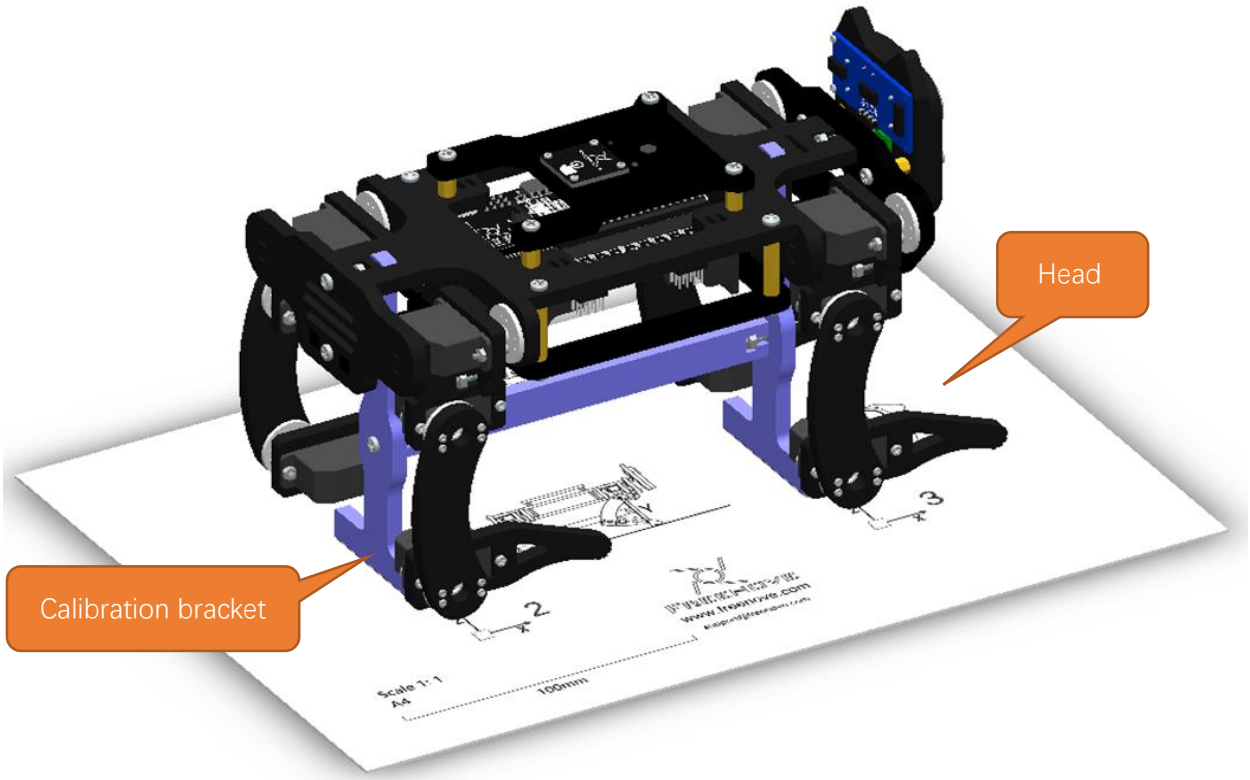
Step 10 Calibration

1. Take out the calibration graph.

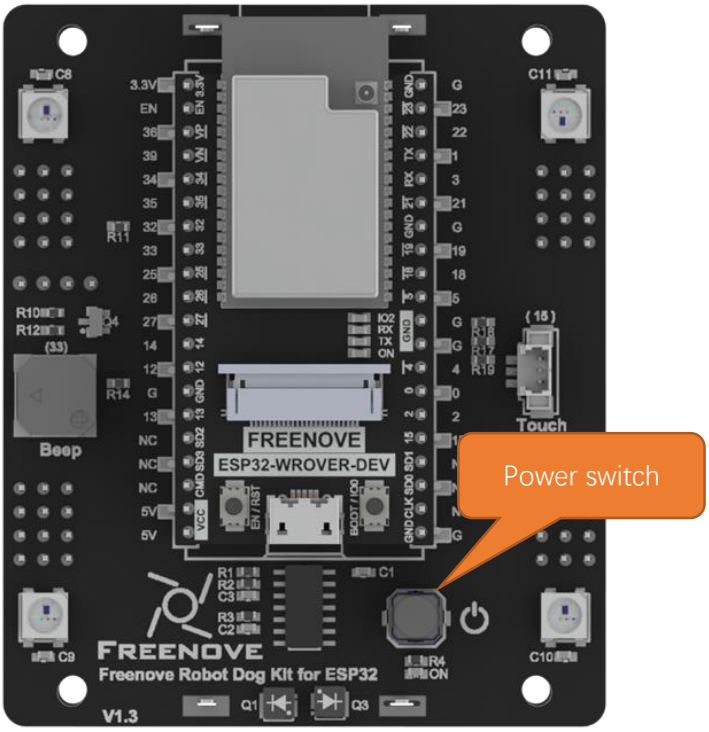




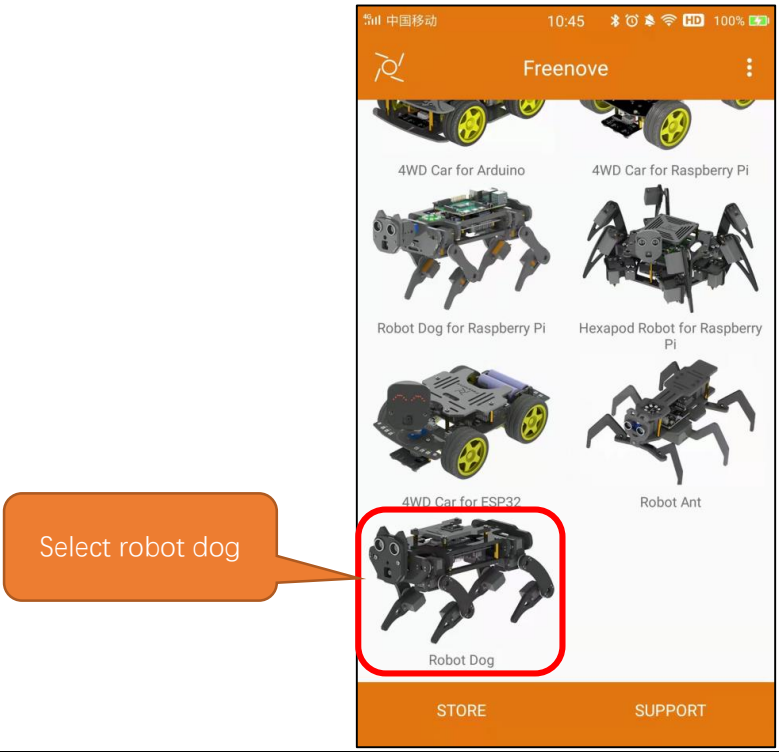
2. Put the robot on the calibration graph.



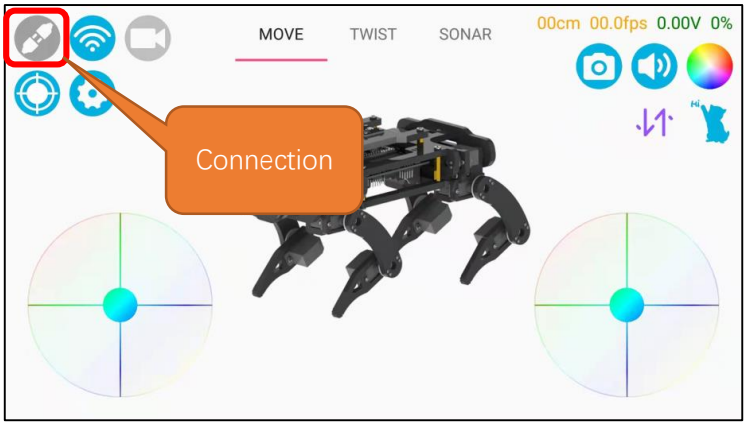
3. Turn ON the power switch on the driver board. (Batteries are not included. Please buy them yourself.)



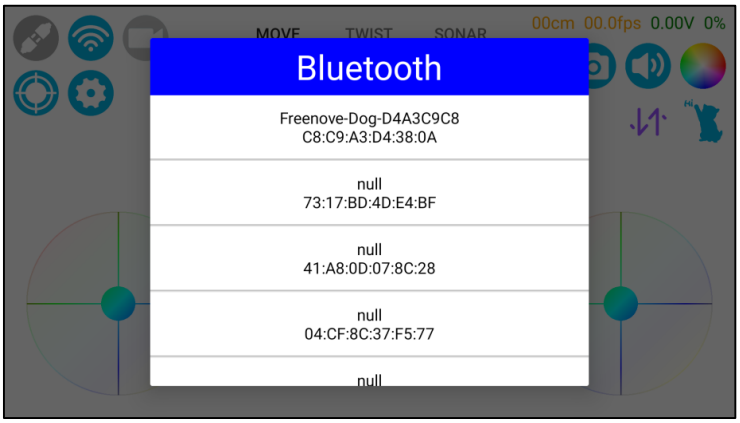
4. Open Freenove App and tap Robot Dog.



5. Turn ON Bluetooth of your phone and tap the Connect button on the app.

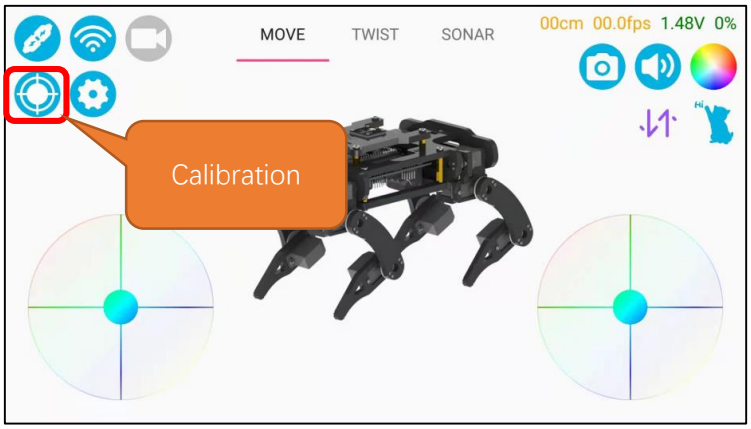


6. Select "Freenove-Dog-XXXXXXXX".

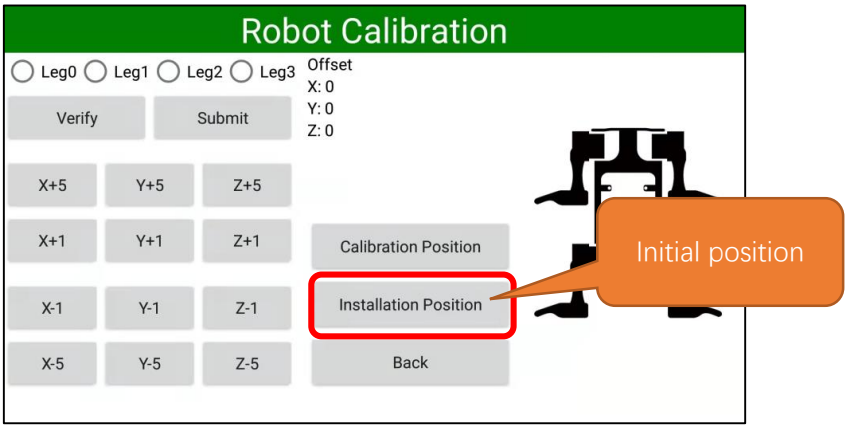


If you do not turn ON the power or upload code to ESP32, you cannot find the Bluetooth of robot dog.

7. Tap the calibration button.

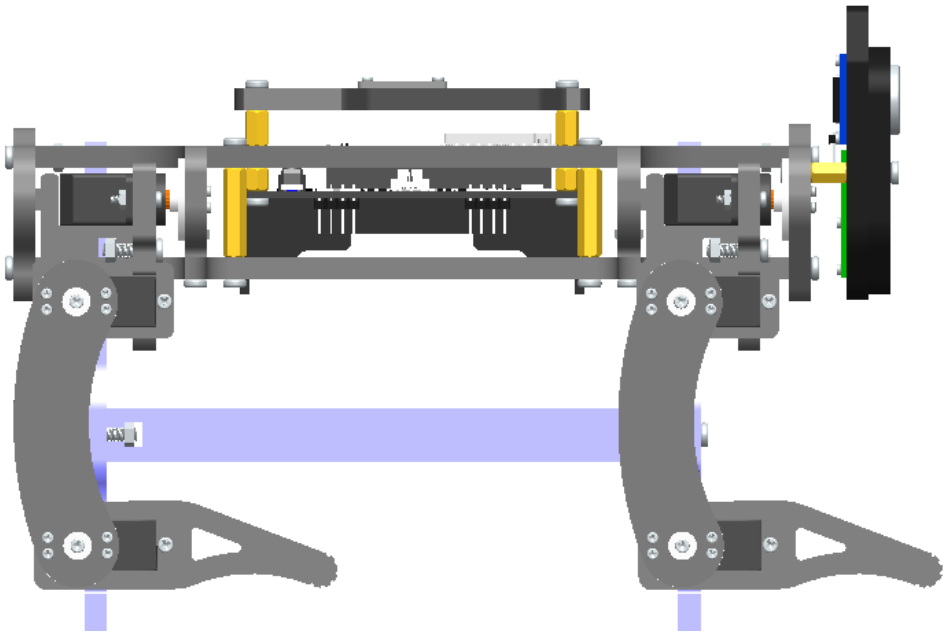


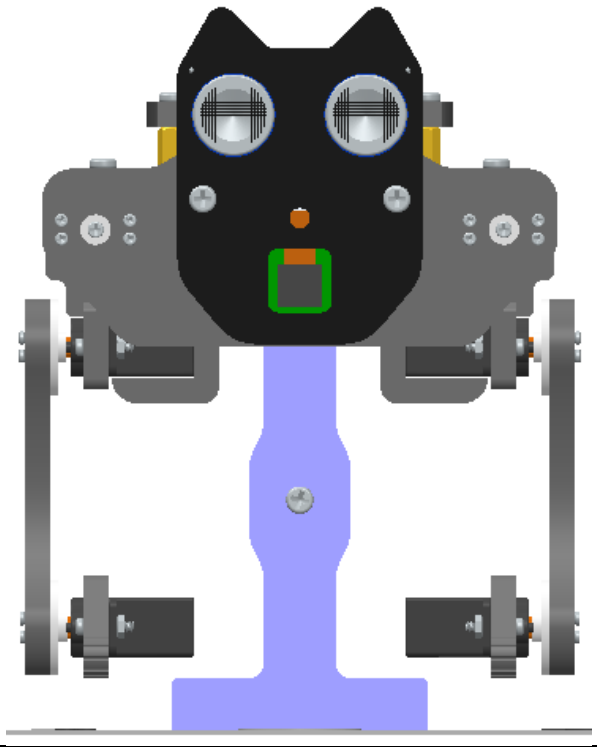
8. Tap Installation Position and all the servos will rotate to 90 degrees.



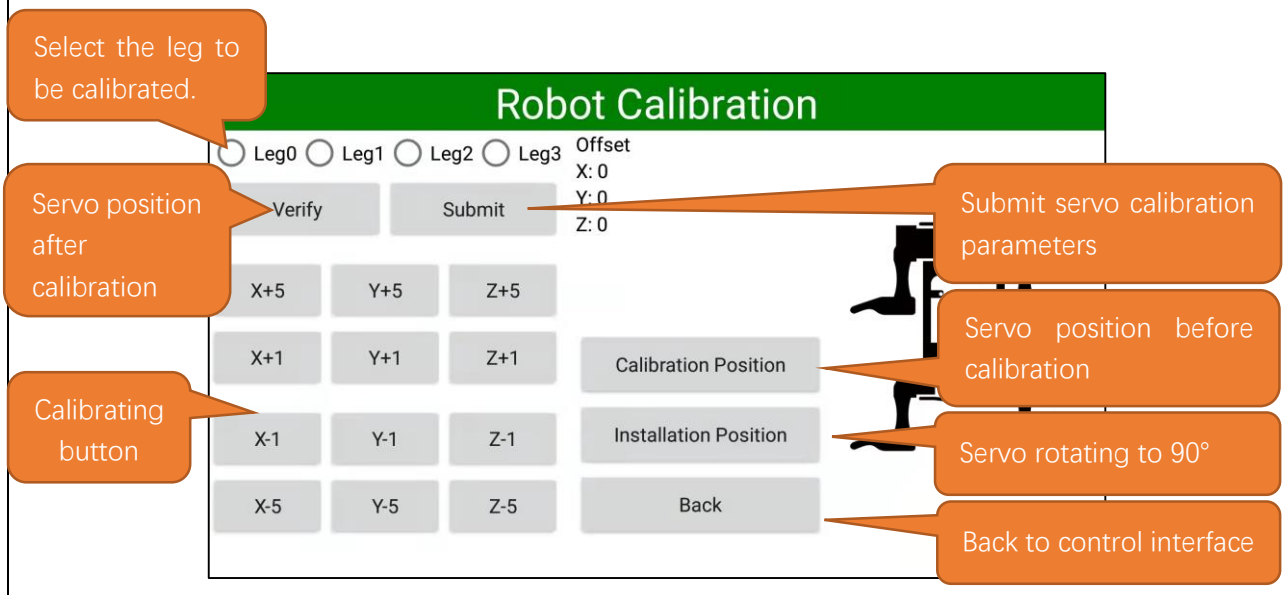
9. The robot dog turns all Servo into 90 degree position. As shown in the figure below.

Note: after pressing the button, **if the Servo position in a different position from the following image, disassemble the Servo and install it in the correct position.** The smaller the deviation Angle, the better.

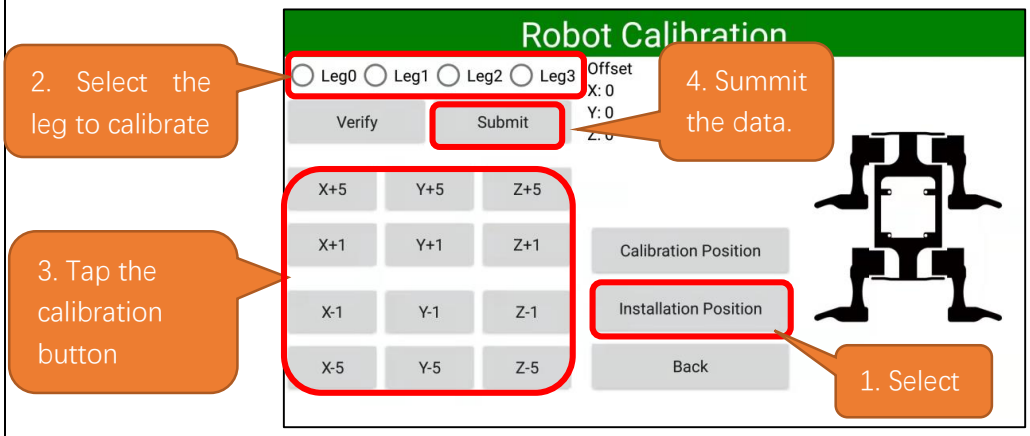




10. Introduction to the interface.



11. Select the leg to calibrate. Calibrate the legs by adjusting the X, Y and Z axes to make the tiptoe match the point on the calibration graph.

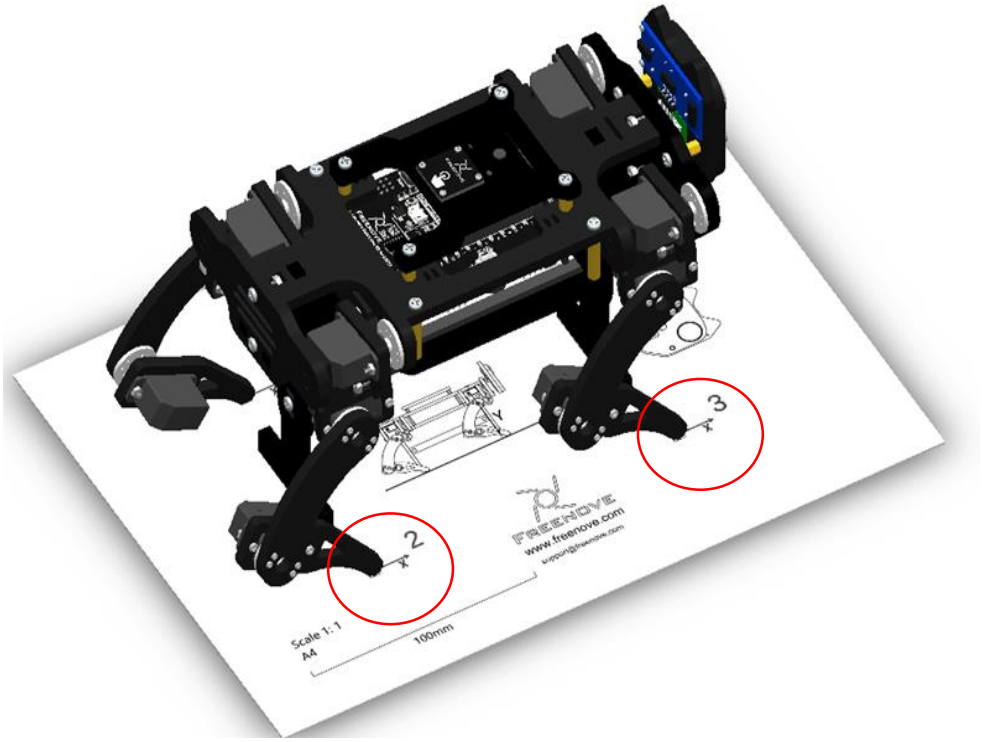


Operation process of robot dog leg calibration:

1. Select the leg to calibrate. Tap the calibration button to adjust the position until the tiptop match the point on calibration graph.
2. Tap “Submit” to upload the data after calibration to robot dog. Without this process, the result will not be saved.
3. Repeat the above steps until all the four legs complete calibration.

The robot after calibration is as shown below:

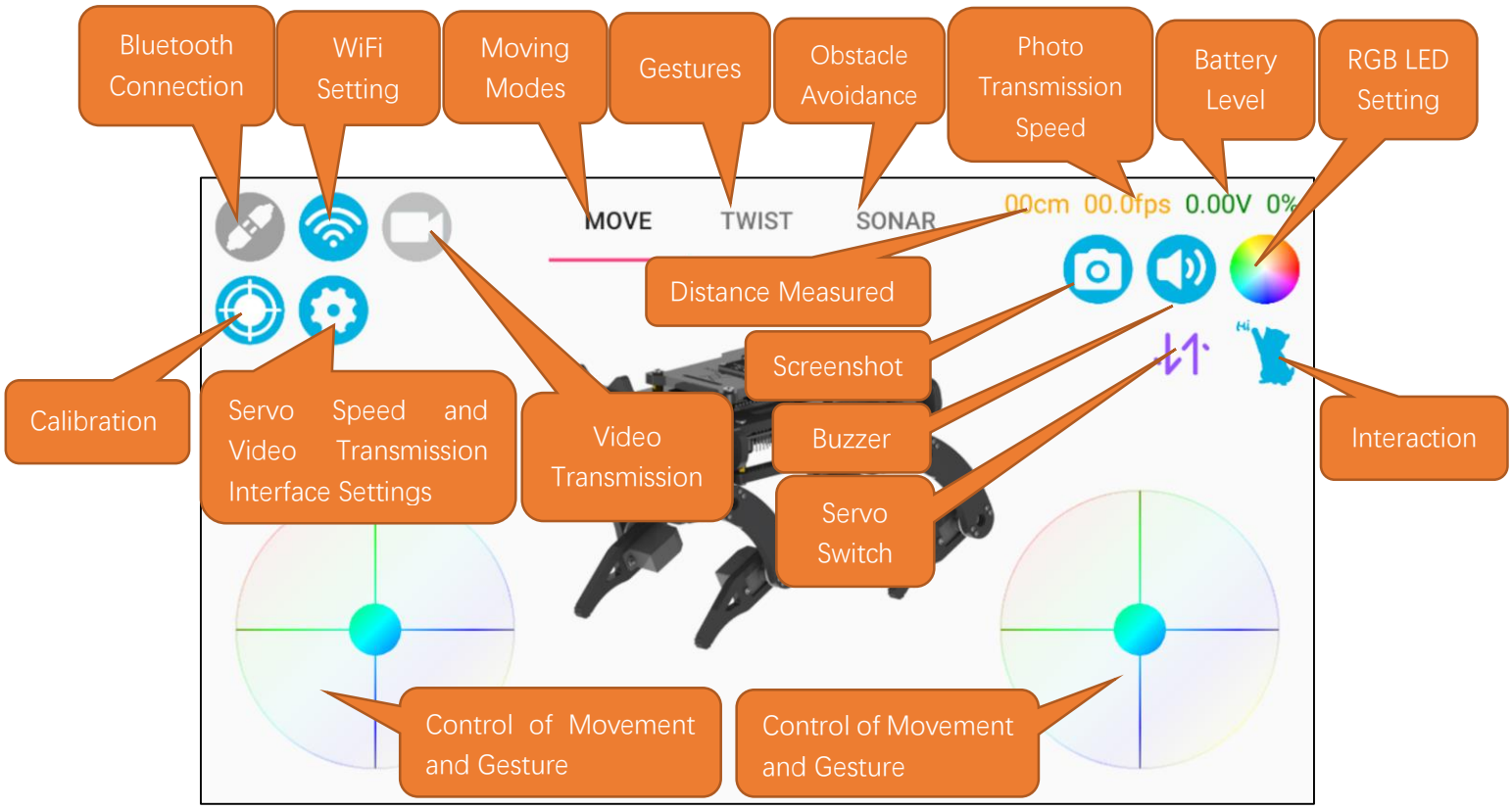
During the adjustment process, you can click the Verify button to Verify that the calibrated leg is accurate. If it is not accurate, the leg needs to be recalibrated. The corrected leg is allowed to have a certain error. The error of +-5mm is allowed in the XZ direction. The Y direction needs to hit the ground right, not jacking up the robot dog, and not hanging off the ground.



By now, all the preparation work for robot dog has been done. You can now play the robot for fun.

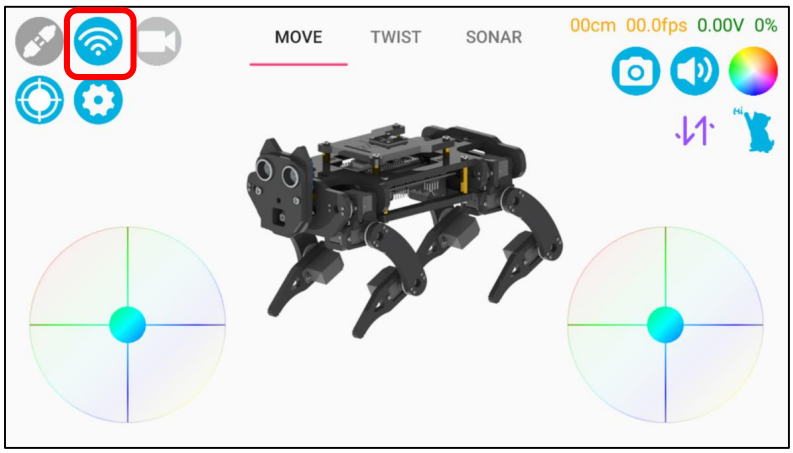
Chapter 3 Functions of Freenove App

Introduction to Main Interface



Wi-Fi Configuration

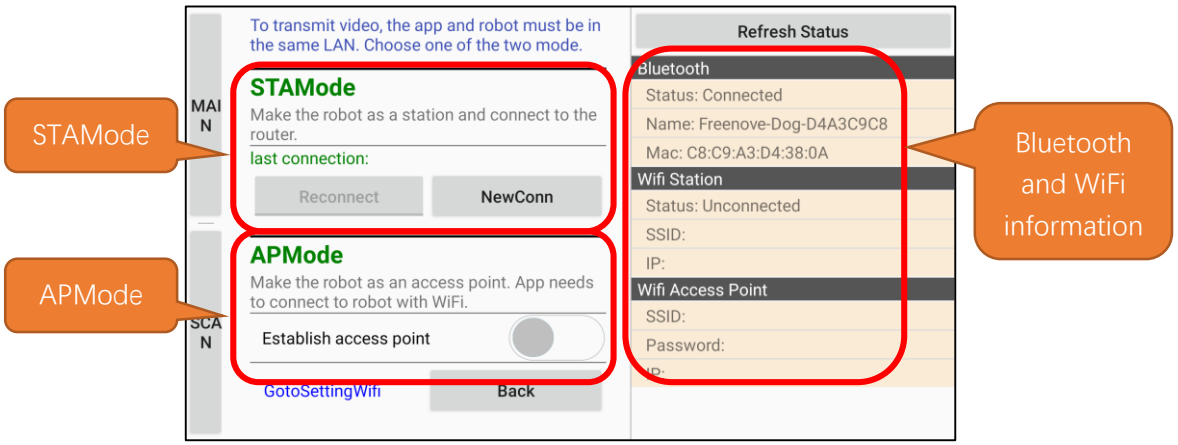
1. Tap Wi-Fi Setting button.



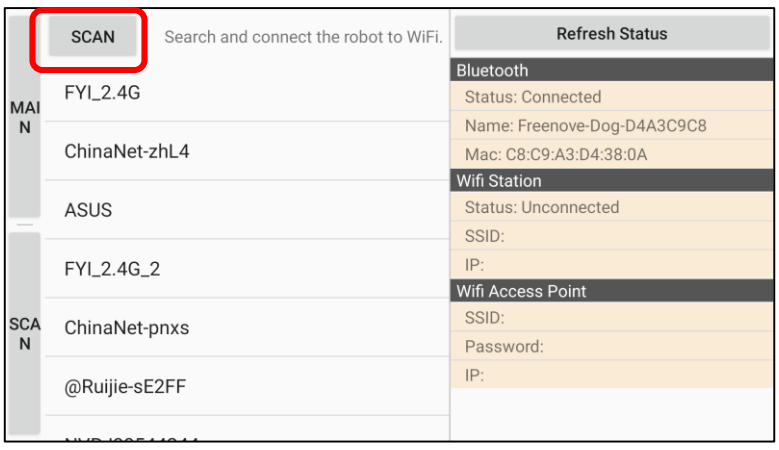
2. Introduction to Wi-Fi configuration interface.

A. To connect the robot dog to a Wi-Fi network, please select STAMode, which can connect the robot to a designated Wi-Fi.

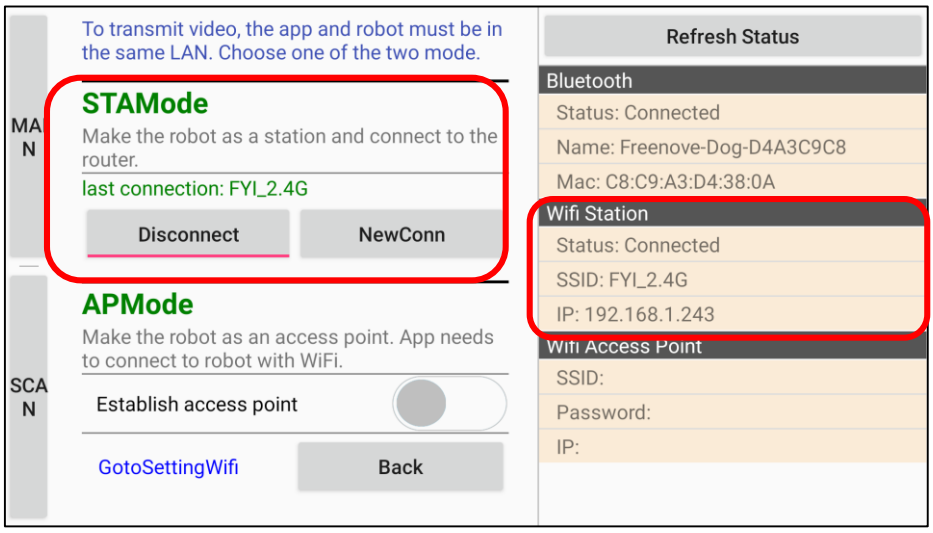
B. When you are outdoors or without available Wi-Fi network, you can select APMode. It can create Wi-Fi network on the robot dog itself to connect to your phone.



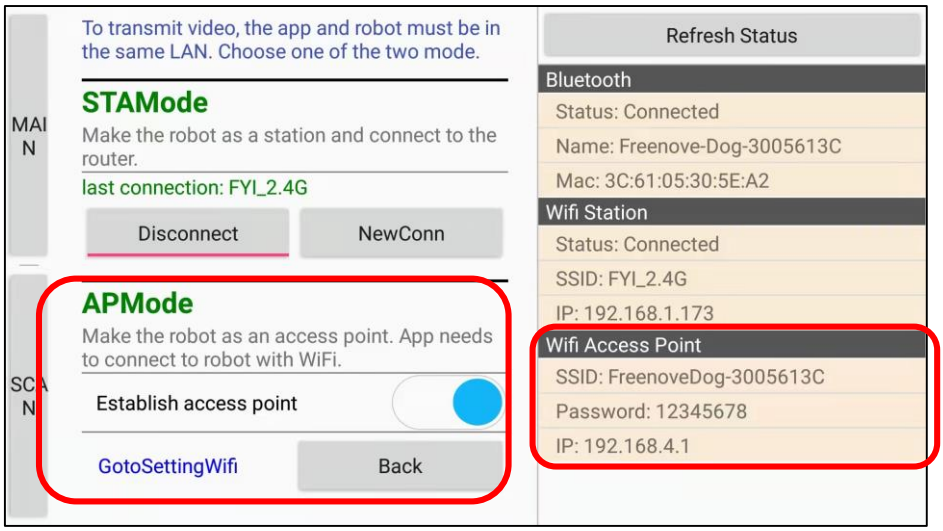
3. Tap NewConn button to enter the Scan interface. Tap Scan button and select the Wi-Fi to be connected.



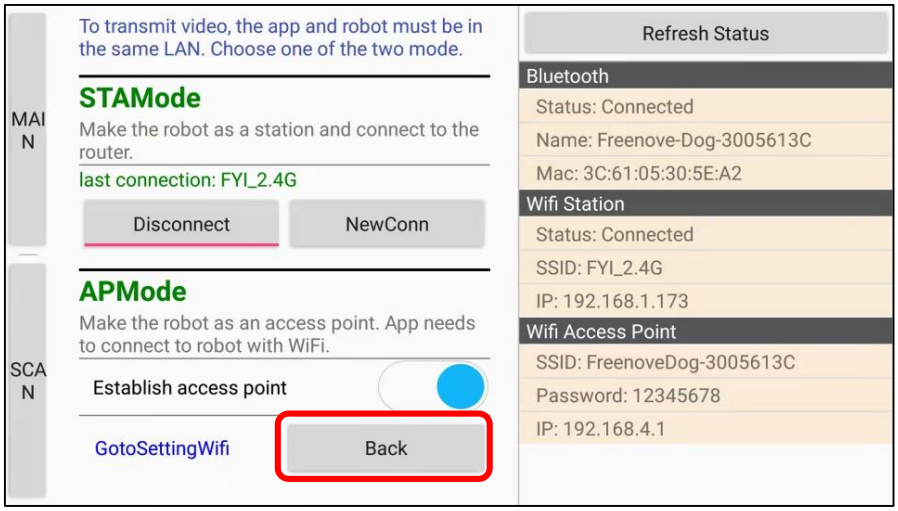
4. The interface of successful connection is as below.



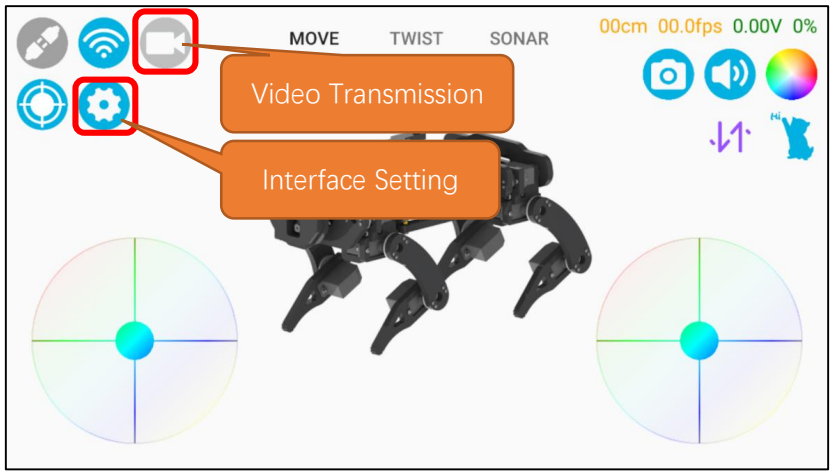
5. To use APMode, please tap the switch below and then connect your phone to the Wi-Fi shown on the right.



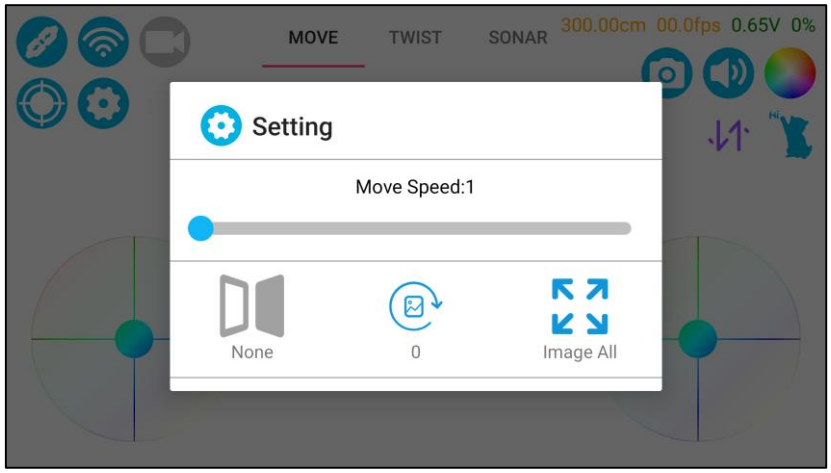
6. Tap Back to exit the configuration interface.



7. After the robot dog connects to Wi-Fi network, tap the video transmission and your phone will display the frames captured by the robot.



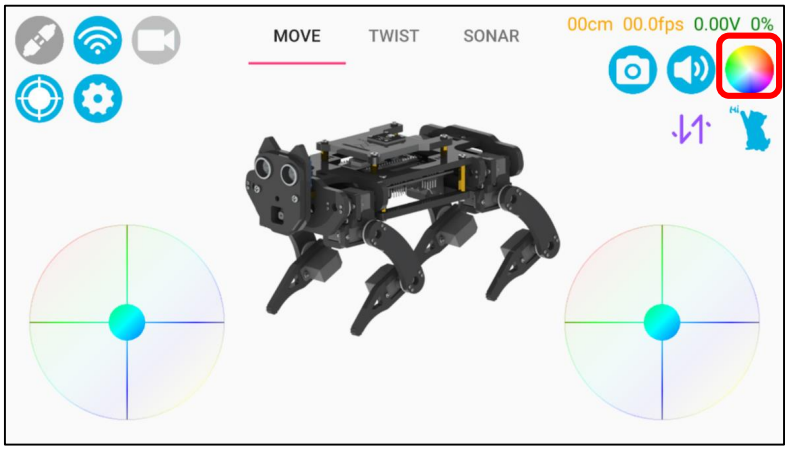
Tip: Tap the Interface Settings button and you can adjust the frames displayed on the phone.



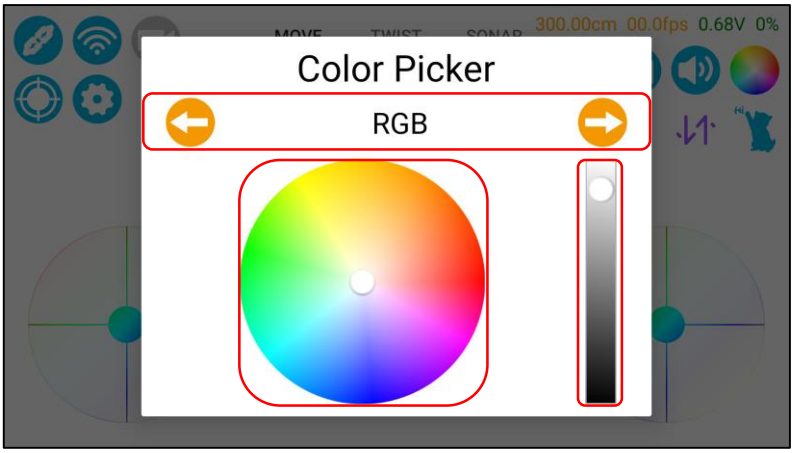
Note: Do not use speed above class 6 for long periods of time as there is a risk of damage to Servo.

RGB LED Control

1. Tap RGB LED control function.

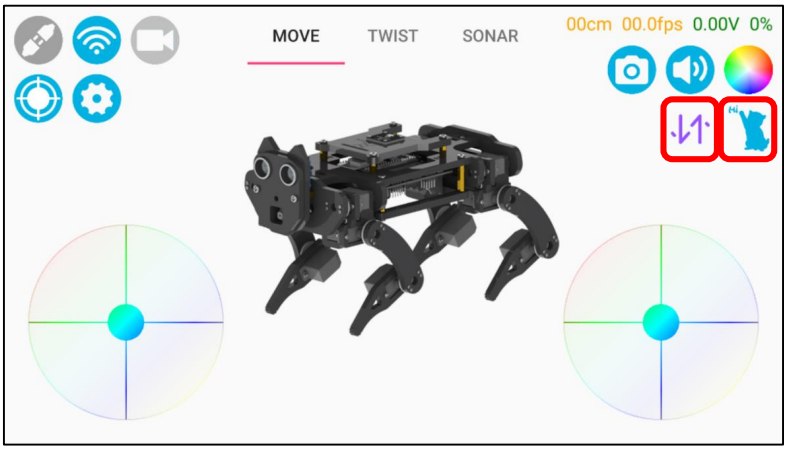


2. The left and right arrows are used to switch the LED modes, the slider to control the brightness and the color palette to set the color.



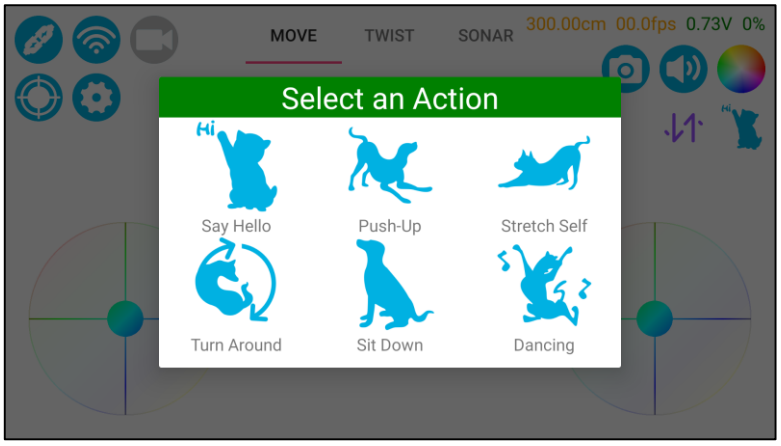
Interaction Function

1. Tap the Interaction button.



Tip: Tapping the up and down arrow icon can make the robot dog lie down and unload the servos.

2. The robot dog can do different actions with different actions being tapped.



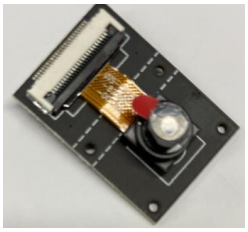
Chapter 4 Q&A

1. The buzzer makes four warning sound after powered ON.

Reason: The robot makes the sounds indicating that data cannot be obtained from the camera.

Troubleshooting:

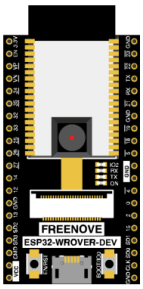
a. The camera does not contact well with the extension board.
Please remove and reconnect the camera to try again.



b. The FPC cable does not contact well with the extension board.
Please remove and reconnect the cable.

c. The FPC cable does not contact well with the extension board.
Please remove and reconnect the cable.

d. The camera may be damaged. Please try to connect the camera to esp32 directly to test.
If the issue remains, then the camera module is damaged.
Please send us an email (support@freenove.com) to solve the issue.



2. When using the video transmission function, the Bluetooth is disconnected.

Reason: As the Bluetooth and WIFI of ESP32 share the same antenna and a large amount of data is transmitted through WIFI during video transmission, turning off Bluetooth can better reduce the interference with WIFI communication. Therefore, once the video transmission function is used, the robot's Bluetooth is turned off by default, and both commands and video transmission are communicated through WIFI.

3. When using the video transmission function, sometimes the response of the robot dog is not timely.

Reason: When the robot dog uses video transmission, only WIFI is enabled for communication. Whether it has a lag depends on whether the signal of the router connected to the robot dog is stable. If the WIFI signal is poor at some point, it will cause the robot dog to be stuck.

4. It is better to control the robot dog using Bluetooth rather than WIFI when not using video transmission.

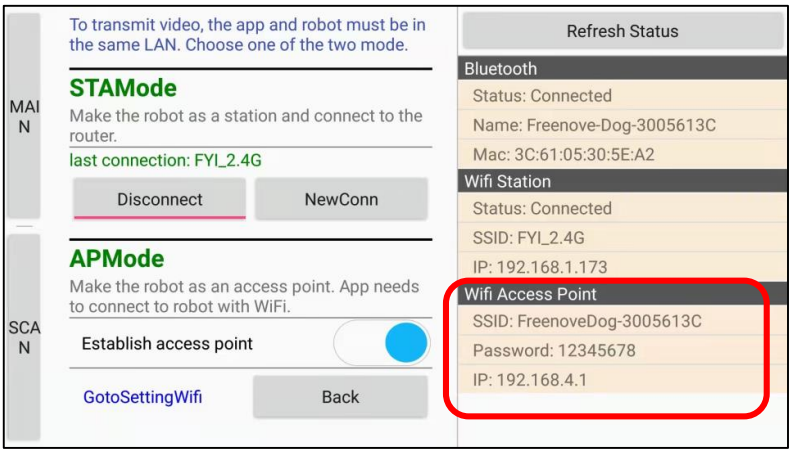
Reason: When not transmitting videos, little data is transferred between the mobile APP and the robot dog. In this case, both WIFI and Bluetooth can control the robot dog well, but the WIFI function may be affected by the router signal, while the Bluetooth function does not have such problem, as it is directly sent by the robot dog to the mobile phone. Therefore, it is strongly recommended to use the Bluetooth function to control the robot dog, and turn off WIFI to increase the stability of the Bluetooth function when video transmission is not used.

5. Q: Which mode is better? Can both modes be enabled simultaneously?

A: When not using video transmission, you can turn on Bluetooth and WIFI at the same time. However, they share the same control commands, so we recommend turning off WIFI and only enabling Bluetooth, which almost causes no lag.
When using video transmission, the robot's Bluetooth is turned off by default to avoid interference to the WIFI function.

6. Q: The WIFI has STA and AP modes, which one should I choose when using video transmission?

A: If there is a router with great signal available, we recommend using STA mode; otherwise, AP mode is recommended.
Note: In AP mode, you need to connect your phone to the robot's hotspot, whose SSID and password are as follows:

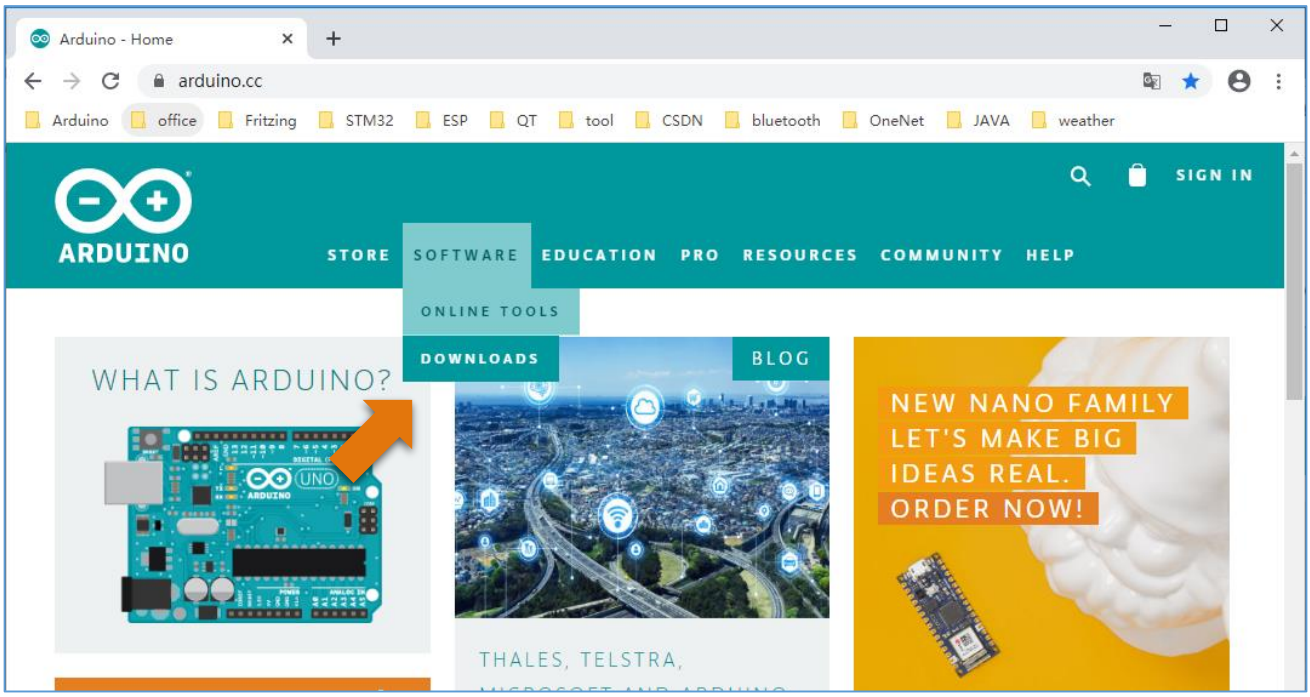


Chapter 5 Arduino Software

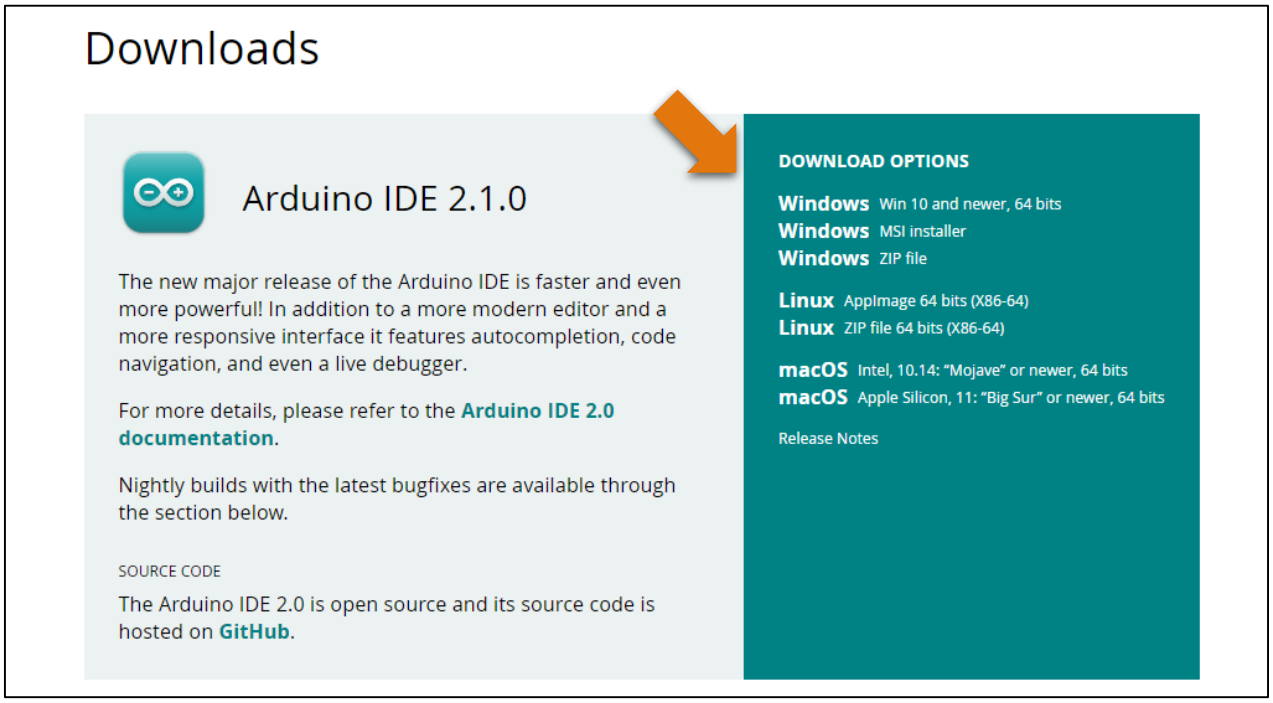
Chapters starting with this chapter are introductions to the robot code. If you are interested in it, you can continue to read.
The content of this chapter is the installation and configuration of the Arduino IDE, the platform for robot dog code.

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 "Download" to enter the download page.



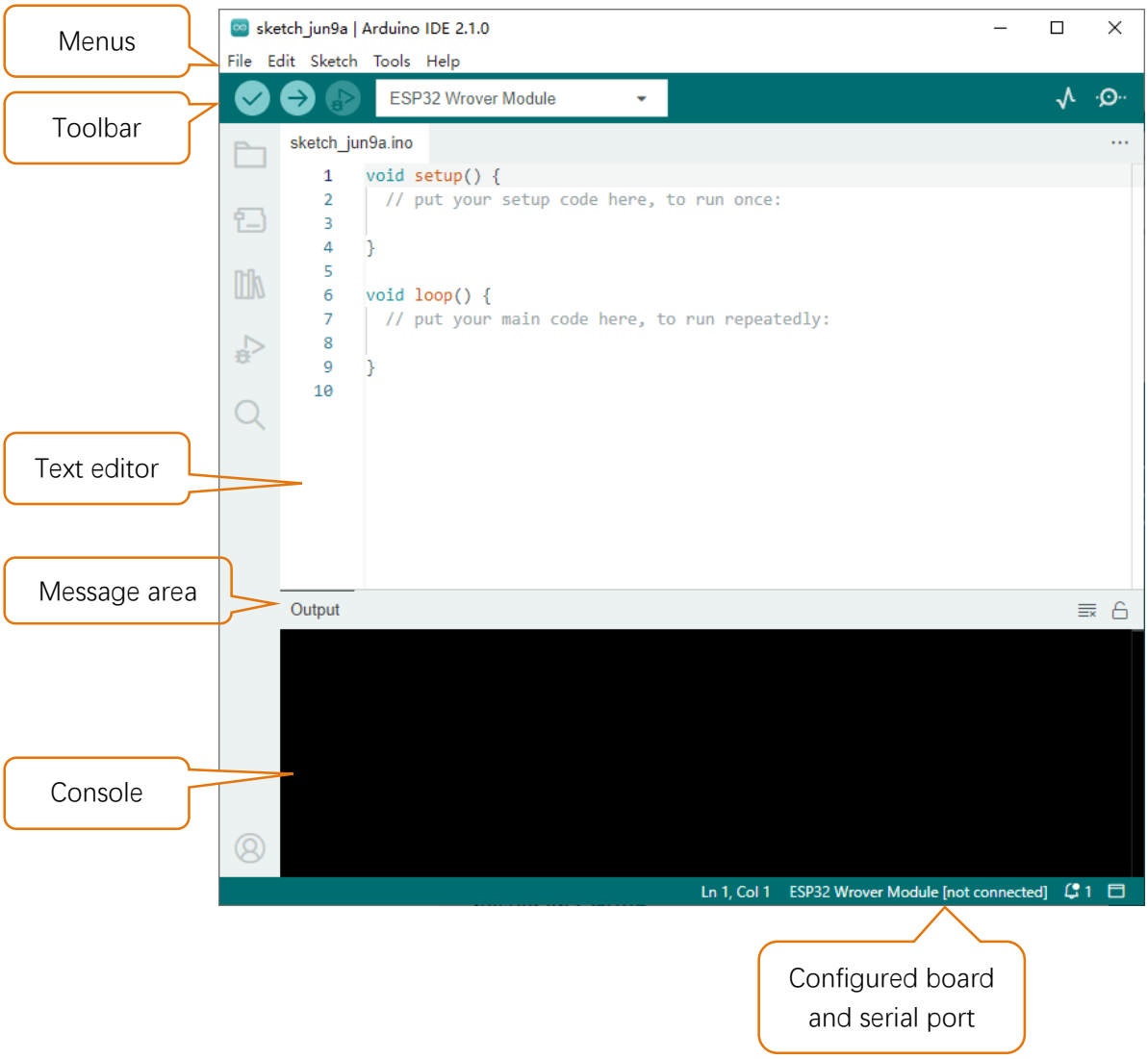
Select and download corresponding installer according to your operating system. If you are a Windows user, please select the "Windows Installer" to download and install the driver correctly.



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 in the desktop. Run the Arduino Software.

The interface of Arduino Software is as follows:


Menus
Toolbar
Text editor
Message area
Console
sketch_jun9a | Arduino IDE 2.1.0
File Edit Sketch Tools Help
ESP32 Wrover Module
sketch_jun9a.ino
1 void setup() {
2 // put your setup code here, to run once:
3
4 }
5
6 void loop() {
7 // put your main code here, to run repeatedly:
8
9 }
10
Output
Ln 1, Col 1 ESP32 Wrover Module [not connected] 1
Configured board and serial port

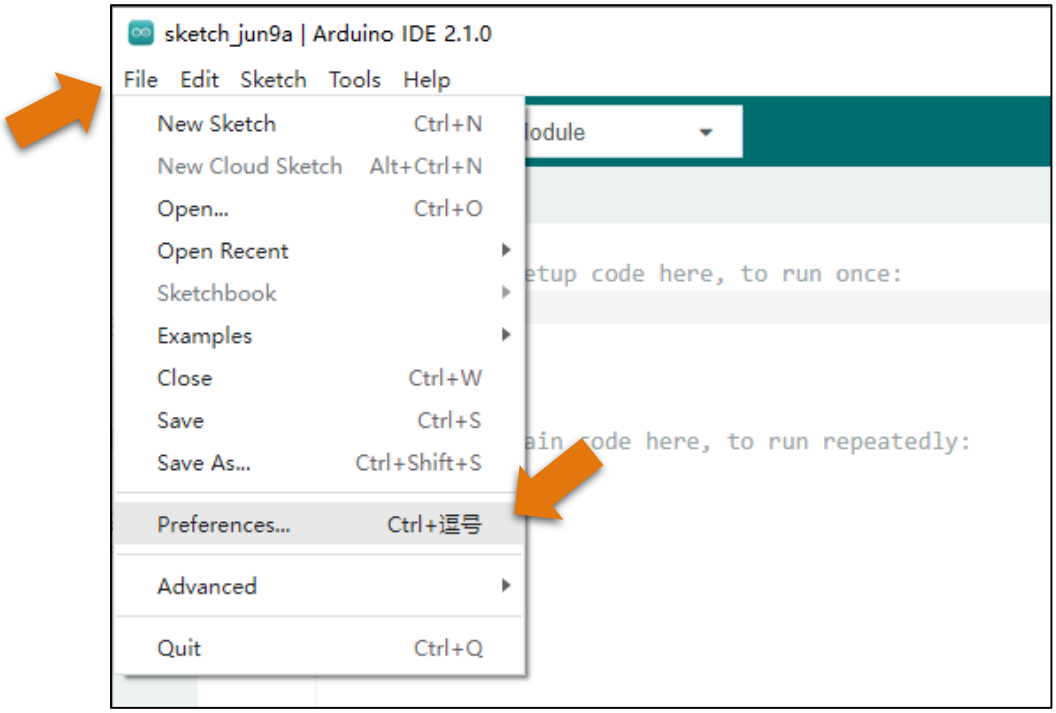
Programs written with Arduino Software (IDE) are called **sketches**. These sketches are written in the text editor and saved with the file extension**.ino**. The editor has features for cutting/pasting and searching/replacing text. The message area gives feedback while saving and exporting and also 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	Cooperate with Debug tool to debug code.
	Serial Plotter	The data received by the serial port is drawn as a line graph.
	Serial Monitor	Open the serial monitor.

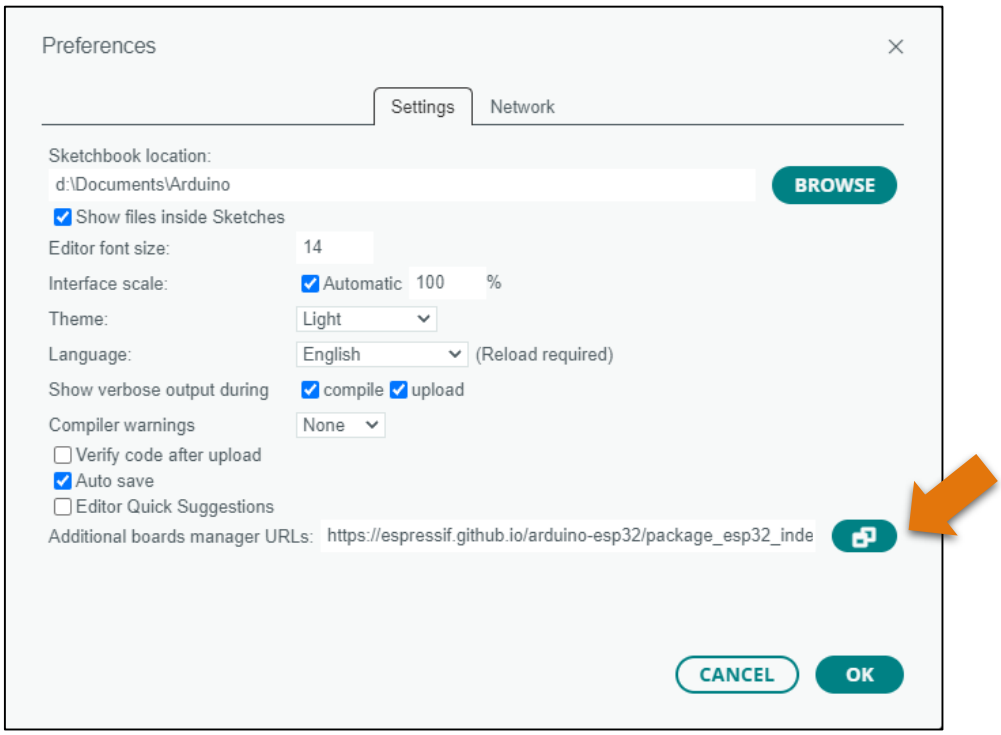
Additional commands are found within the five menus: File, Edit, Sketch, Tools, Help. The menus are context sensitive, which means only those items relevant to the work currently being carried out are available.

Environment Configuration

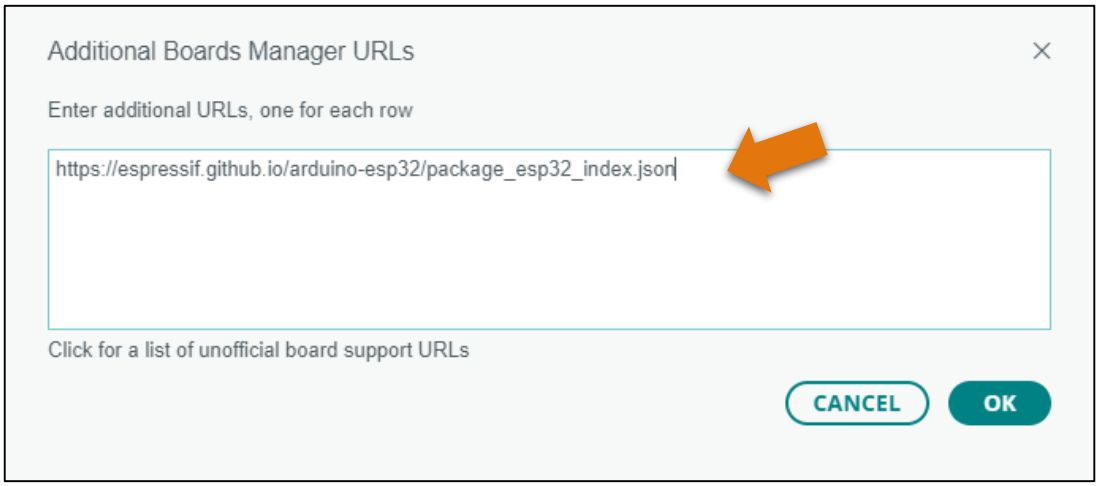
First, open the software platform arduino, and then click File in Menus and select Preferences.



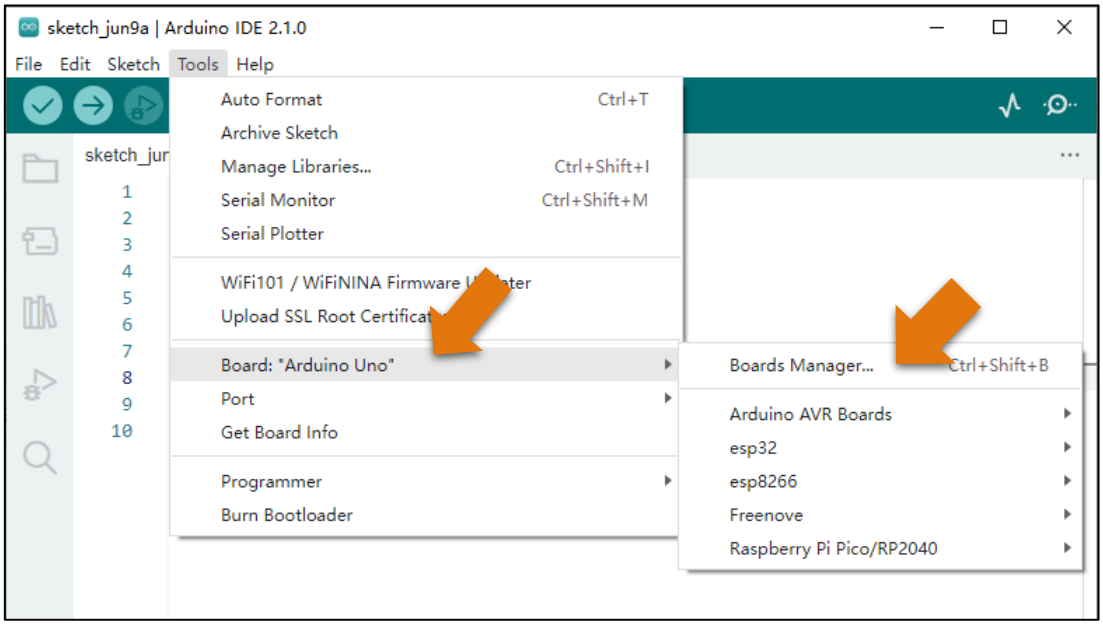
Second, click on the symbol behind "Additional Boards Manager URLs"



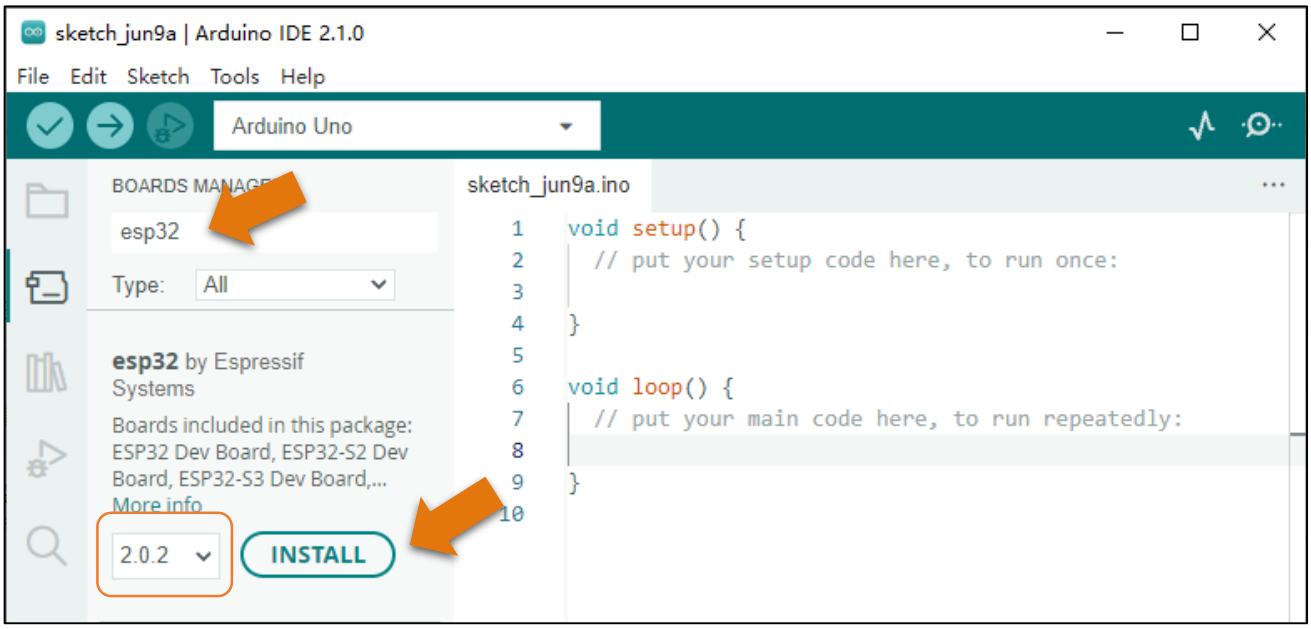
Third, fill in https://espressif.github.io/arduino-esp32/package_esp32_index.json in the new window, click OK, and click OK on the Preferences window again.



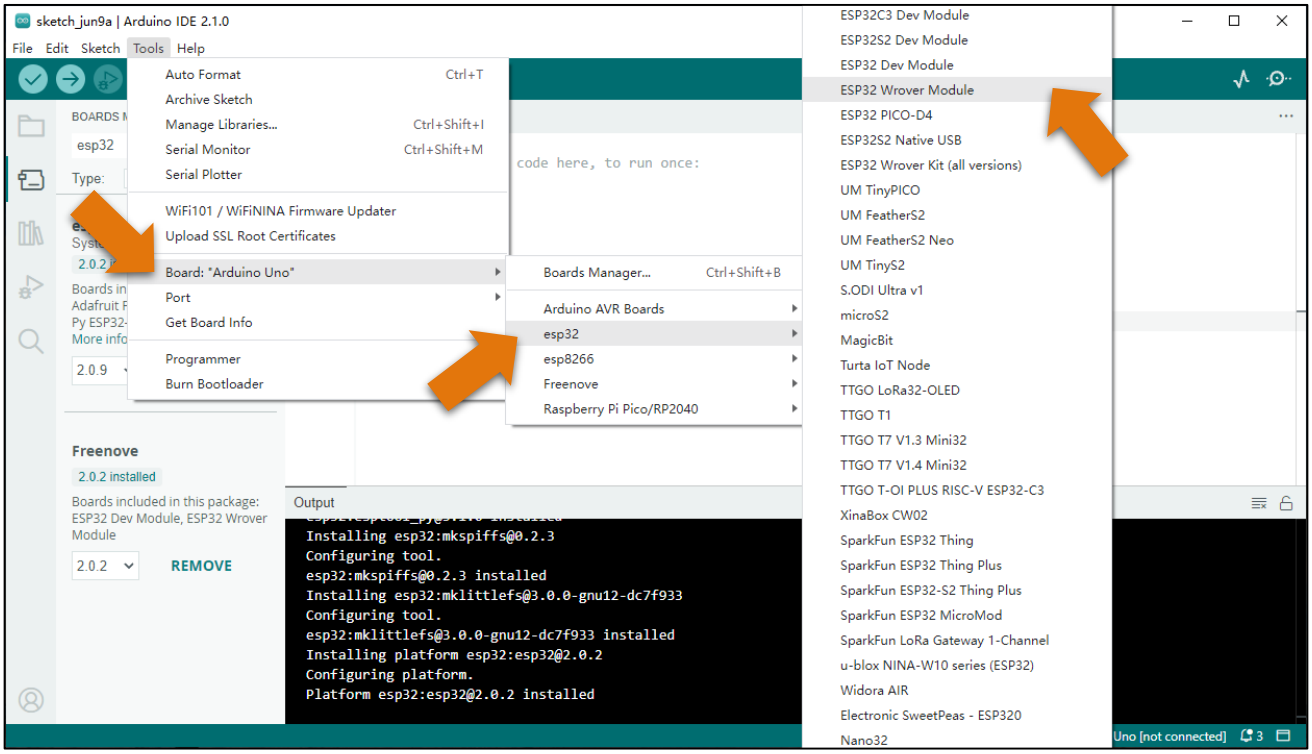
Fourth, click Tools in Menus, select Board: "Arduino Uno", and then select "Boards Manager".



Fifth, input "esp32" in the window below, and press Enter. Select version 2.0.2. Click "Install" to install.



When finishing installation, click Tools in the Menus again and select Board: "Arduino Uno", and then you can see information of ESP32-WROVER. click "ESP32-WROVER" so that the ESP32 programming development environment is configured.



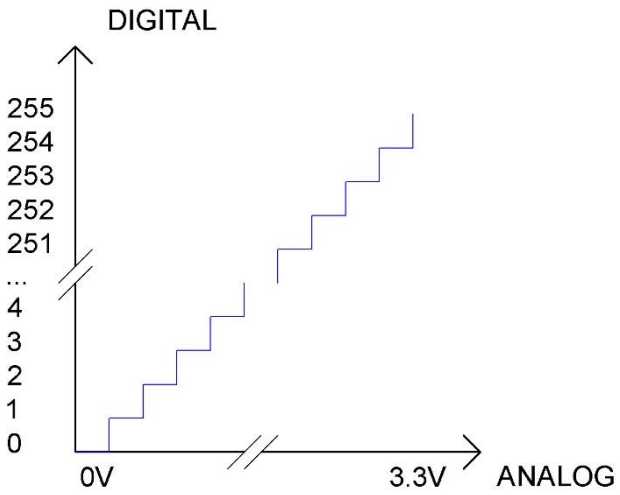
Chapter 6 Battery

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

Related knowledge

ADC

An 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 ESP32 is 12 bits, that means the resolution is 2^12=4096, and it represents a range (at 3.3V) will be divided equally to 4096 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.

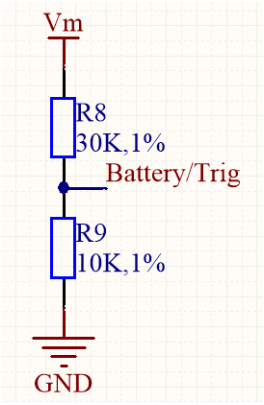


Subsection 1: the analog in rang of 0V---3.3/4095 V corresponds to digital 0;
Subsection 2: the analog in rang of 3.3/4095 V---2*3.3 /4095V corresponds to digital 1;
...
The following analog will be divided accordingly.
The conversion formula is as follows:

$$ADC\ Value = \frac{\text{Analog Voltage}}{3.3} * 4095$$

Schematic

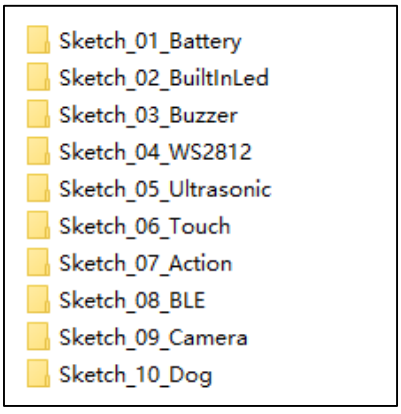
As we can see, the robot reads the voltage of the batteries through GPIO32 of ESP32. Because the battery voltage is not read frequently, this GPIO is also used to control the Ultrasonic.



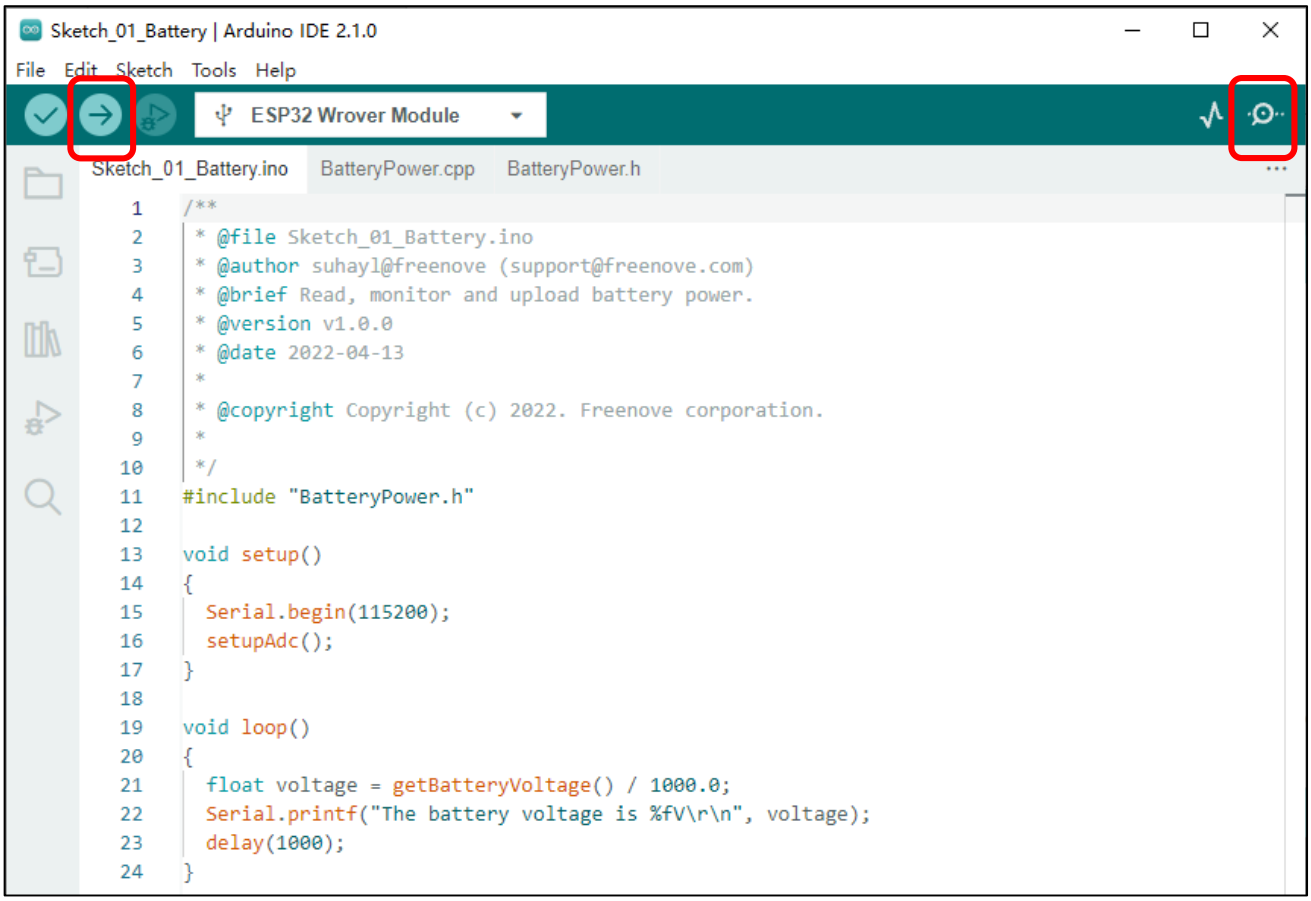
As shown in the figure above, the positive pole Vm of the power supply is controlled by the switch.
1. When the switch is turned on, the voltage value of VM is the voltage value of the battery.
2. Battery/Trig is connected to GPIO32 pin of ESP32.
3. The voltage acquisition range of GPIO32 on ESP32 is 0-3.3V, while the robot dog is powered by two 18650 lithium batteries, and the voltage is 8.4V when fully charged, which exceeds the voltage acquisition range of ESP32. Therefore, after passing through the voltage divider circuit composed of R3 and R4, the voltage at Battery/Trig is about 1/4 of the battery voltage, 8.4 / 4 = 2.1V, which is within the voltage acquisition range of GPIO32.

Sketch

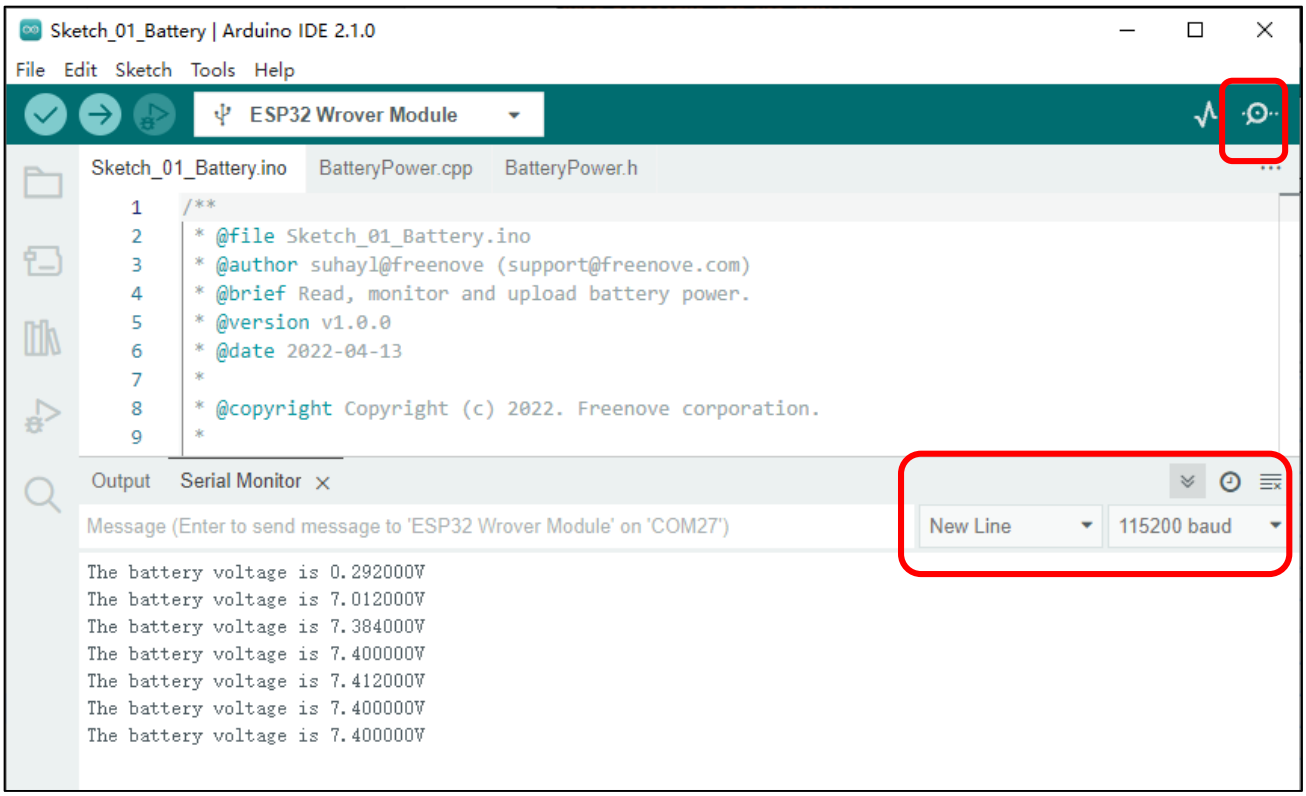
In this section, we will use GPIO32 of ESP32 to read the voltage value of the batteries and print it on serial monitor. Open "Sketch_01_Battery" folder in **"Freenove_Robot_Dog_Kit_for_ESP32\Sketches"** and then double-click "Sketch_01_Battery.ino".



Sketch_01_Battery



Click the upload button in the upper left corner to upload the code to esp32. Open the serial monitor in the upper right corner and set the baud rate to 115200.



The following is the code:

```
#include "BatteryPower.h"
void setup(){
  Serial.begin(115200);
  setupAdc();
}
void loop(){
  float voltage = getBatteryVoltage() / 1000.0;
  Serial.printf("The battery voltage is %fV\r\n", voltage);
  delay(1000);
}
```

In the Arduino IDE, the setup() function is usually used as an initialization function, and the code is executed only once. Serial.begin() is the initialization function of the serial port. The parameter in the bracket indicates the speed of the serial port communication, and the unit is usually Bit. The baud rate is set to 115200 here, which means that 115200 bits of data are transmitted within one second. The setupAdc() function is used to initialize the battery voltage pin of the esp32.

```
3  void setup(){
4    Serial.begin(115200);
5    setupAdc();
6  }
```

In the Arduino IDE, the loop() function is usually used as a repeated execution function, and the code in the function will be executed in an infinite loop. The getBatteryVoltage() function is used to obtain the voltage value of the battery in millivolts.

The Serial.printf() function can print the content processed by ESP32 to the serial monitor through the serial port.

delay(x) is a delay function in milliseconds. Every time it is called, it causes the code execution to pause for x milliseconds,

```
8  void loop(){
9    float voltage = getBatteryVoltage() / 1000.0;
10   Serial.printf("The battery voltage is %fV\r\n", voltage);
11   delay(1000);
12 }
```

BatteryPower.h

```
#ifndef _BATTERYVOLTAGE_h
#define _BATTERYVOLTAGE_h

#include "Arduino.h"
#define VOLATAGE_RATIO   4      //Resistance partial voltage ratio
#define PIN_ANALOG_IN    32    //ESP32 GPIO number
#define NUM_OF_SAMPLES  64    //ADC sample times
void setupAdc();
uint32_t getBatteryVoltage();

#endif
```

BatteryPower.cpp

```
1  #include "BatteryPower.h"
2
3  void setupAdc(){
4      analogReadResolution(12);
5      analogSetAttenuation(ADC_11db);
6      adcAttachPin(PIN_ANALOG_IN);
7  }
8
9  uint32_t getBatteryVoltage(){
10     uint32_t adc_reading = 0;
11     for (int i = 0; i < NUM_OF_SAMPLES; i++){
12         adc_reading += analogRead(PIN_ANALOG_IN);
13     }
14     adc_reading /= NUM_OF_SAMPLES;
15     uint32_t voltage = adc_reading*3300/4096;
16     return voltage * VOLATAGE_RATIO;
17 }
```

ADC data sampling accuracy setting. Here it is set to 12 bits. 2^12=4096, so the sampling data range of ADC is 0-4095.

```
4      analogReadResolution(12);
```

ADC sampling voltage range setting of ESP32, which is 0-3.3V. ADC_11db represents the maximum range. Combined with the above analogReadResolution() function, when the voltage of the ADC collection point is 0V, the value collected by the ADC is 0; when the voltage of the ADC collection point is 3.3V, the value collected by the ADC is 4095.

```
5      analogSetAttenuation(ADC_11db);
```

Associate the ADC configuration to the pin.

```
6      adcAttachPin(PIN_ANALOG_IN);
```

Use analogRead(PIN_ANALOG_IN) to capture the ADC value at the ESP32 pin.

Continuously collect NUM_OF_SAMPLES ADC values at the ESP32 pins and calculate the average.

```
10     uint32_t adc_reading = 0;
11     for (int i = 0; i < NUM_OF_SAMPLES; i++){
12         adc_reading += analogRead(PIN_ANALOG_IN);
13     }
14     adc_reading /= NUM_OF_SAMPLES;
```

Convert the ADC value to the actual measured voltage value. The unit is millivolts.

```
15     uint32_t voltage = adc_reading*3300/4096;
```

We know from the previous introduction that the voltage at the ESP32 pin is 1/4 of the battery voltage. Therefore, the voltage value of the battery can be obtained by multiplying the voltage value measured by the ADC by VOLATAGE_RATIO(4).

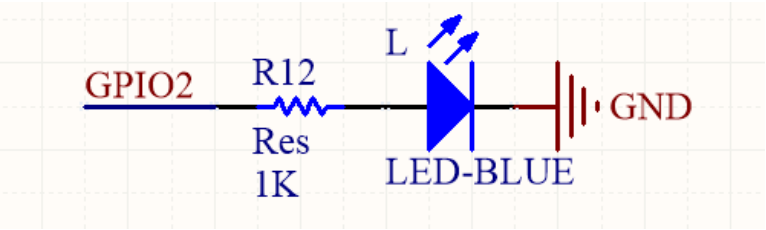
```
16     return voltage * VOLATAGE_RATIO;
```

Chapter 7 Built-in Led

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

Schematic

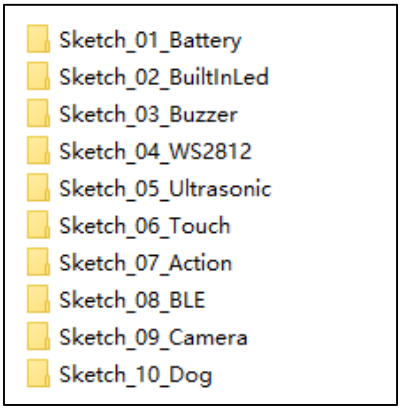
There are four LED lights on the ESP32 development board. ON is the power indicator, TX and RX are indicators for communication or code upload status, and IO2 is the onboard indicator light., which is connected on the GPIO2 pin of the ESP32.



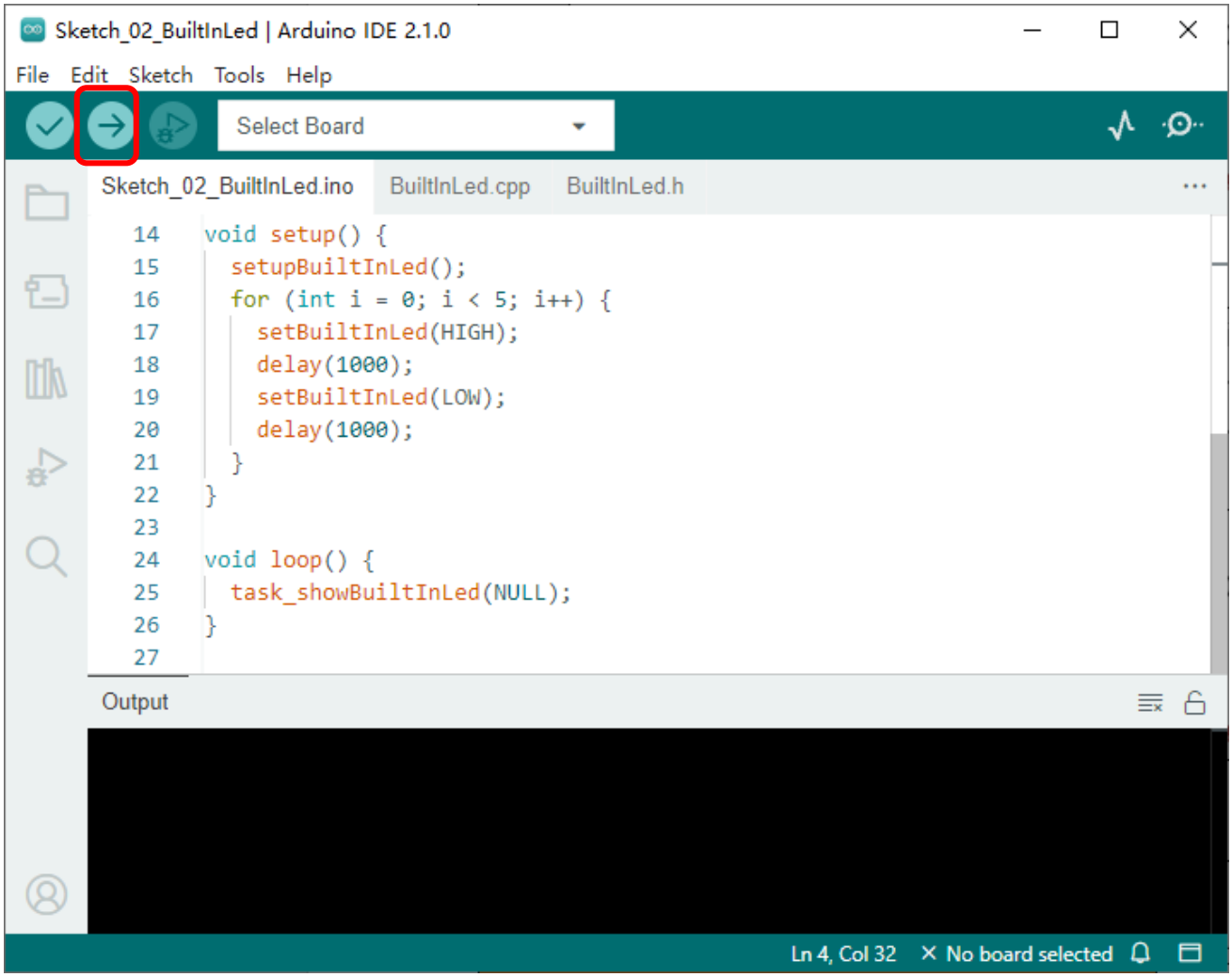
As shown in the figure above, when the ESP32 outputs a high level, the onboard indicator will light up. When the ESP32 outputs a low level, the onboard indicator light will be off. We can use it to indicate the running state of our code.

Sketch

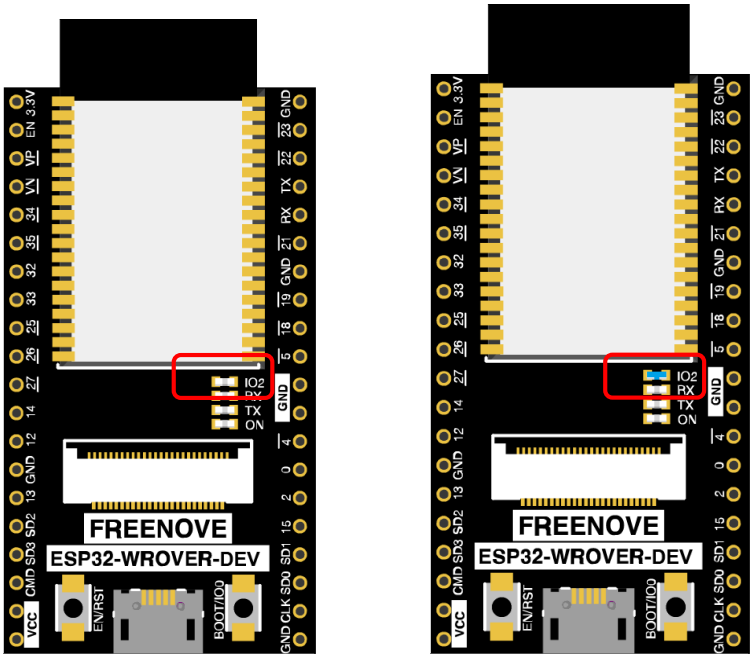
Open "Sketch_02_BuiltInLed" folder in **"Freenove_Robot_Dog_Kit_for_ESP32\Sketches"** and then double-click "Sketch_02_BuiltInLed.ino".



Sketch_02_BuiltInLed



Click the upload button in the upper left corner to upload the code to esp32. The LED on the ESP32 board will turn on and off every 1 second for 5 times, and enter the flash mode.



The following is the code:

```
#include "BuiltInLed.h"

void setup() {
  setupBuiltInLed();
  for (int i = 0; i < 5; i++) {
    setBuiltInLed(HIGH);
    delay(1000);
    setBuiltInLed(LOW);
    delay(1000);
  }
}

void loop() {
  task_showBuiltInLed(NULL);
}
```

Initializes the pins that control the onboard LEDs.

```
  setupBuiltInLed();
```

Use the setBuiltInLed(bool) function to control the on-board LED to turn on or off.

The LED light on the control board turns on and off every 1 second, which repeats 5 times.

```
  for (int i = 0; i < 5; i++) {
    setBuiltInLed(HIGH);
    delay(1000);
    setBuiltInLed(LOW);
    delay(1000);
  }
```

BuiltInLed.h

```
#ifndef _BUILTINLED_h
#define _BUILTINLED_h

#include "Arduino.h"
#define LED_BUILT_IN 2
void setupBuiltInLed();
void setBuiltInLed(bool state);

void task_showBuiltInLed(void *pvParameters);

#endif
```

BuiltInLed.cpp

```
#include "BuiltInLed.h"

uint32_t sqValue = 0b00000101;
uint32_t lastLedUpdateTime = 0;
uint16_t intervalTime = 100; // ms
bool ledStatus = false;
bool lastLedStatus = true;
uint8_t ledRunnigStatus = 0;

void setupBuiltInLed(){
    pinMode(LED_BUILT_IN, OUTPUT);
}

void setBuiltInLed(bool state){
  digitalWrite(LED_BUILT_IN, state);
}

void task_showBuiltInLed(void *pvParameters){
    static uint8_t bitIndex = 0;
    static uint8_t bitDiff = 0;
    if (millis() - lastLedUpdateTime > intervalTime){
        switch (ledRunnigStatus){
        case 0: // running
            ledStatus = ((sqValue >> bitIndex) & 0x01) ? true : false;
            if (ledStatus != lastLedStatus){
                digitalWrite(LED_BUILT_IN, (uint8_t)ledStatus);
                lastLedStatus = ledStatus;
                bitDiff++;
            }
            bitIndex++;
            if (bitIndex >= 10){
                intervalTime = 600;
                ledRunnigStatus = 1;
                bitDiff = 0;
                bitIndex = 0;
            }
            break;
        case 1: // off
            intervalTime = 100;
            ledRunnigStatus = 0;
            break;
        default:
            break;
```

```
        }
        lastLedUpdateTime = millis();
    }
}
```

Onboard LED initialization function.

```
void setupBuiltInLed(){
    pinMode(LED_BUILT_IN, OUTPUT);
}
```

The function that controls the on-board LED to turn on or off.

```
void setBuiltInLed(bool state){
  digitalWrite(LED_BUILT_IN, state);
}
```

Enter the if statement every occasionally. This usage has the same effect as the delay() function, but it will not cause the code to be pause.

```
    if (millis() - lastLedUpdateTime > intervalTime){
```

Control the LED's on and off states by shifting a variable, sqValue, and taking the least significant bit as the control signal.

If the current state of the LED is different from the previous state, control the LED to change its current state and update the previous state to the current state.

```
            ledStatus = ((sqValue >> bitIndex) & 0x01) ? true : false;
            if (ledStatus != lastLedStatus){
                digitalWrite(LED_BUILT_IN, (uint8_t)ledStatus);
                lastLedStatus = ledStatus;
                bitDiff++;
            }
```

Introducing a variable called bitIndex to control the shift amount, so as to change the number of shifts performed on sqValue each time,. If the number of shifts exceeds 9, the next time entering the if condition will be delayed by 600ms, and sqValue starts shifting from the 0th bit again.

```
            bitIndex++;
            if (bitIndex >= 10){
                intervalTime = 600;
                ledRunnigStatus = 1;
                bitDiff = 0;
                bitIndex = 0;
            }
```

If the previous shift cycle is completed, set the time for entering the if condition to 100ms each time and proceed to the next shift cycle.

```
        case 1: // off
            intervalTime = 100;
            ledRunnigStatus = 0;
            break;
```

Chapter 8 Buzzer

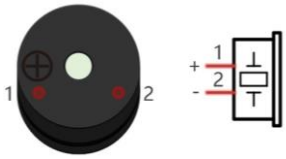
If you have any concerns, please feel free to contact us via support@freenove.com

Component knowledge

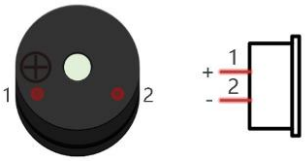
Buzzer

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

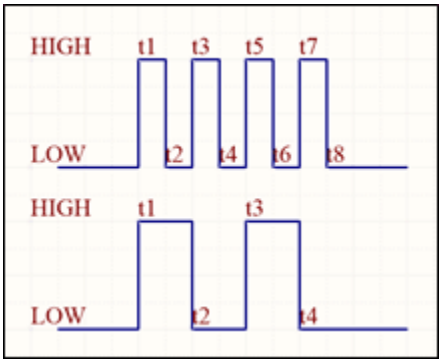


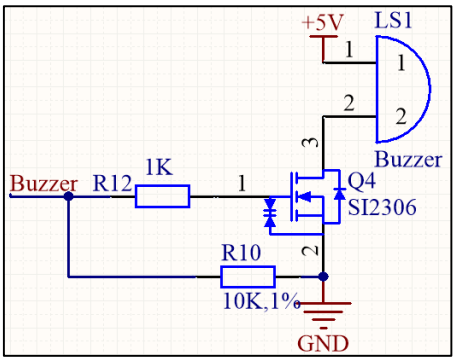
Active buzzer is easy to use. Generally, it can only make a specific frequency of sound. Passive buzzer requires an external circuit to make a sound, but it can be controlled to make a sound with different frequency. The resonant frequency of the passive buzzer is 2kHz, which means the passive buzzer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

Next, we will use an active buzzer to make a doorbell and a passive buzzer to make an alarm.

Schematic

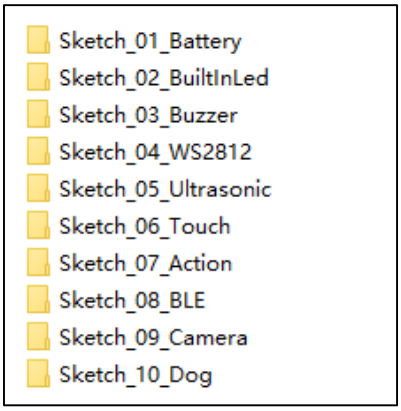
As shown in the figure below, we connect a buzzer to GPIO33 of ESP32. When GPIO33 outputs high level, the buzzer circuit is turned on; when GPIO33 outputs low level, the buzzer circuit is turned off. Since the buzzer is a passive buzzer, when we switch quickly between high and low levels, we can control the passive buzzer to make a sound. The frequency of the sound depends on the duration of the high and low levels (f=1/T, T=t1+t2).



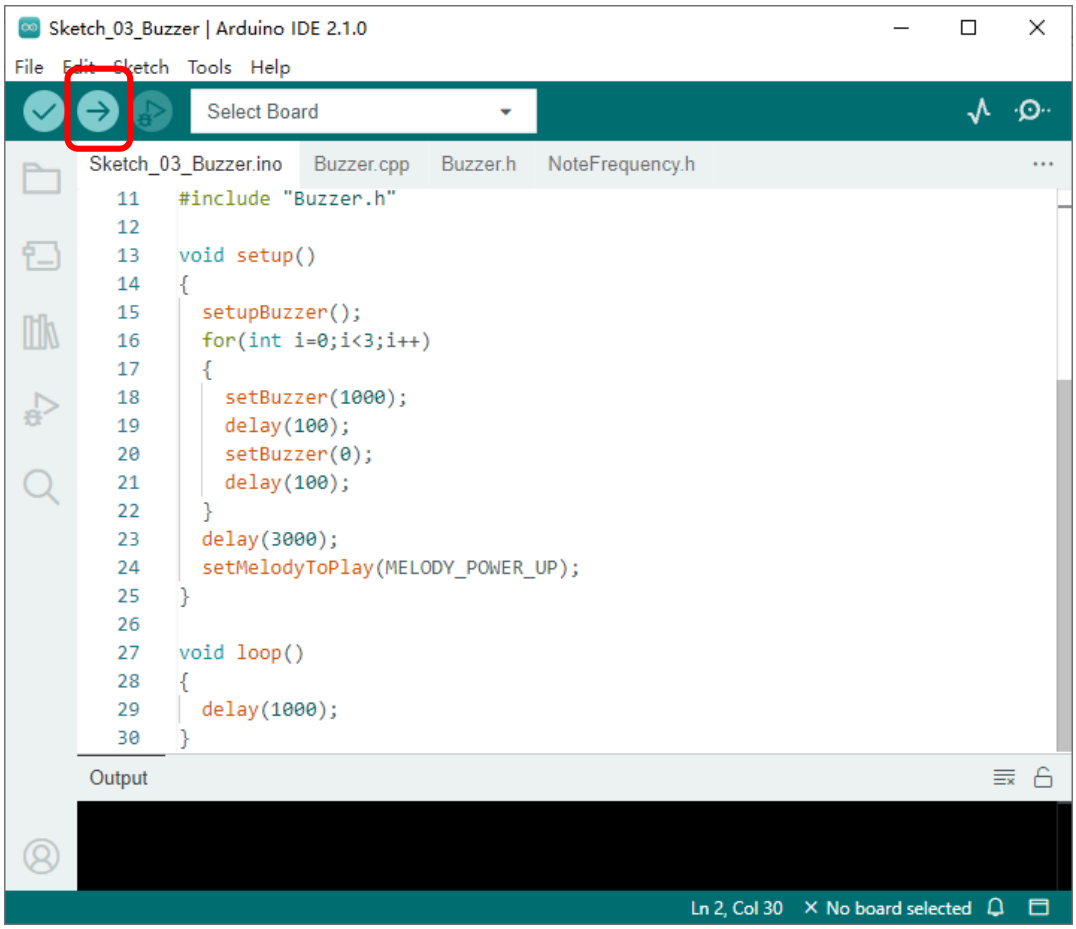


Sketch

Open "Sketch_03_Buzzer" folder in **"Freenove_Robot_Dog_Kit_for_ESP32\Sketches"** and then double-click "Sketch_03_Buzzer.ino".



Sketch_03_Buzzer



Upload the sketch and the buzzer will beep three times at a fixed frequency and then emit a pleasant sound.

The following is the code:

```
1  void setup() {
2    setupBuzzer();
3    for (int i = 0; i < 3; i++) {
4      setBuzzer(1000);
5      delay(100);
6      setBuzzer(0);
7      delay(100);
8    }
9    delay(3000);
10   setMelodyToPlay(MELODY_POWER_UP);
11 }
12
13 void loop() {
14   delay(1000);
15 }
```

Initialize the buzzer pin.

```
2    setupBuzzer();
```

Control the passive buzzer to emit a sound at specific frequency.

```
4      setBuzzer(1000);
```

Each time this function is called, a thread will be created and a melody will be played. The thread will be closed after playback is complete.

```
10   setMelodyToPlay(MELODY_POWER_UP);
```

Buzzer.h

```
1  #ifndef _BUZZER_h
2  #define _BUZZER_h
3
4  #include "NoteFrequency.h"
5  #include "Arduino.h"
6
7  #define MELODY_POWER_UP              0
8  #define MELODY_LOW_POWER             1
9  #define MELODY_NO_POWER              2
10 #define MELODY_WIFI_CONNECT_SUCCESS   3
11 #define MELODY_WIFI_CONNECT_FAILED    4
12 #define MELODY_WIFI_DISCONNECT        5
13 #define MELODY_BLE_CONNECT_SUCCESS    6
14 #define MELODY_BLE_DISCONNECT         7
15 #define MELODY_CAM_CONNECT_SUCCESS    8
16 #define MELODY_CAM_DISCONNECT         9
17 #define MELODY_CAM_FAILURE            10
18 #define MELODY_BEEP_1                 11
19 #define MELODY_BEEP_2                 12
```

```
#define MELODY_DHTG                 13
#define MELODY_BB_CLEAR_1           14
#define MELODY_BB_CLEAR_2           15
......
void setupBuzzer();
void setBuzzer(uint16_t freq);
void setMelodyToPlay(int m);
void task_Buzzered(void *pvParameters);

#endif
```

Buzzer.cpp

```
#include "Buzzer.h"

#define BUZZER_PIN (33)
#define BUZZER_CHN (0)
#define BUZZER_BIT (10)
#define BUZZER_FREQ (5000)

int melody = -1;

void setupBuzzer() {
  ledcSetup(BUZZER_CHN, BUZZER_FREQ, BUZZER_BIT);  // setup pwm channel
  ledcAttachPin(BUZZER_PIN, BUZZER_CHN);
  ledcWrite(BUZZER_CHN, 0);
  ledcWriteTone(BUZZER_CHN, 0);
}

void setBuzzer(uint16_t freq) {
  if (freq != 0) {
    ledcWrite(BUZZER_CHN, 512);
    ledcWriteTone(BUZZER_CHN, freq);
  } else {
    ledcWrite(BUZZER_CHN, 0);
    ledcWriteTone(BUZZER_CHN, 0);
  }
}

template<int N>  //Non-type template parameters
void playMelody(const int (&tune)[N], const float (&beat)[N], float speed) {
  for (int i = 0; i < N; i++) {
    setBuzzer(tune[i]);
    delay(beat[i] * speed);
    setBuzzer(0);
```

```
33    }
34    setBuzzer(0);
35  }
36
37  bool enableBuzzered = false;
38  TaskHandle_t taskHandle_Buzzered;
39
40  void setMelodyToPlay(int m) {
41    melody = m;
42    enableBuzzered = true;
43    xTaskCreateUniversal(task_Buzzered, "task_Buzzered", 2048, NULL, 1, &taskHandle_Buzzered, 0);
44  }
45
46  void task_Buzzered(void *pvParameters) {
47    if (enableBuzzered) {
48      switch (melody) {
49        case MELODY_POWER_UP:
50          playMelody(tunePowerUp, beatPowerUp, 300);
51          break;
52        case MELODY_LOW_POWER:
53          playMelody(tuneLowPower, beatLowPower, 300);
54          break;
55        case MELODY_NO_POWER:
56          playMelody(tuneNoPower, beatNoPower, 500);
57          break;
58        case MELODY_WIFI_CONNECT_SUCCESS:
59          playMelody(tuneWifiSucc, beatWifiSucc, 300);
60          break;
61        case MELODY_WIFI_CONNECT_FAILED:
62          playMelody(tuneWifiFailed, beatWifiFailed, 300);
63          break;
64        case MELODY_WIFI_DISCONNECT:
65          playMelody(tuneWifiDis, beatWifiDis, 300);
66          break;
67        case MELODY_BLE_CONNECT_SUCCESS:
68          playMelody(tuneBleSucc, beatBleSucc, 100);
69          break;
70        case MELODY_BLE_DISCONNECT:
71          playMelody(tuneBleDis, beatBleDis, 100);
72          break;
73        case MELODY_CAM_CONNECT_SUCCESS:
74          playMelody(tuneCamSucc, beatCamSucc, 100);
75          break;
76        case MELODY_CAM_DISCONNECT:
```

```
77          playMelody(tuneCamDis, beatCamDis, 100);
78          break;
79        case MELODY_CAM_FAILURE:
80          playMelody(tuneCamFailure, beatCamFailure, 500);
81          break;
82        case MELODY_BEEP_1:
83          playMelody(tuneBeep1, beatBeep1, 300);
84          break;
85        case MELODY_BEEP_2:
86          playMelody(tuneBeep2, beatBeep2, 100);
87          break;
88        case MELODY_DHTG:
89          playMelody(tuneDhtg, beatDhtg, 500);
90        case MELODY_BB_CLEAR_1:
91          playMelody(tuneBb1, beatBb1, 100);
92          break;
93        case MELODY_BB_CLEAR_2:
94          playMelody(tuneBb2, beatBb2, 100);
95          break;
96        default:
97          break;
98      }
99    }
100   vTaskDelete(xTaskGetCurrentTaskHandle());
101 }
```

Define the pin for the buzzer. ESP32's PWM channel, number of bits, and frequency.

```
3  #define BUZZER_PIN (33)
4  #define BUZZER_CHN (0)
5  #define BUZZER_BIT (10)
6  #define BUZZER_FREQ (5000)
```

Configure the pin to control the buzzer:

1. Configure the PWM channel, frequency, and number of digits.
2. Associate the PWM signal output channel with the pin that controls the buzzer.
3. Control the duty cycle of the PWM channel to 0.
4. Control the buzzer to make no sound.

```
10 void setupBuzzer() {
11   ledcSetup(BUZZER_CHN, BUZZER_FREQ, BUZZER_BIT);  // setup pwm channel
12   ledcAttachPin(BUZZER_PIN, BUZZER_CHN);
13   ledcWrite(BUZZER_CHN, 0);
14   ledcWriteTone(BUZZER_CHN, 0);
15 }
```

Call setBuzzer() to control the buzzer to emit sounds with different frequencies.

```
17 void setBuzzer(uint16_t freq) {
18   if (freq != 0) {
```

```
19      ledcWrite(BUZZER_CHN, 512);
20      ledcWriteTone(BUZZER_CHN, freq);
21    } else {
22      ledcWrite(BUZZER_CHN, 0);
23      ledcWriteTone(BUZZER_CHN, 0);
24    }
25  }
```

The melody playback function. Every time this function is called, a thread will be created to control the buzzer to play the tune with the specified frequency.

```
40  void setMelodyToPlay(int m) {
41    melody = m;
42    enableBuzzered = true;
43    xTaskCreateUniversal(task_Buzzered, "task_Buzzered", 2048, NULL, 1, &taskHandle_Buzzered, 0);
44  }
```

Buzzer thread callback function. Each time the thread is executed, the vTaskDelete(xTaskGetCurrentTaskHandle()) function is automatically called to close its own thread.

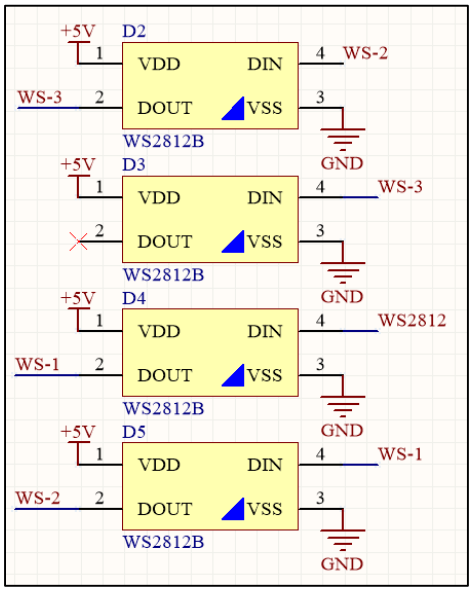
```
    void task_Buzzered(void *pvParameters) {
      if (enableBuzzered) {
        switch (melody) {
        ......
        }
      }
      vTaskDelete(xTaskGetCurrentTaskHandle());
    }
```

Chapter 9 WS2812

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

Schematic

As shown in the figure below, we connect the WS2812 to GPIO0 of ESP32. In this way, we can havel the WS2812 to display various colors by controlling GPIO0.

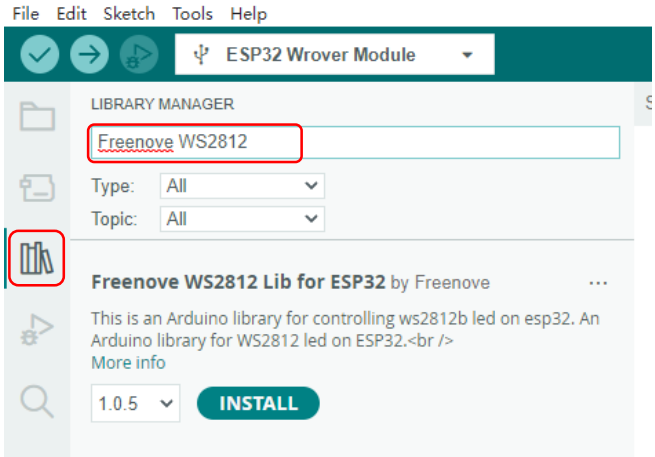


Sketch

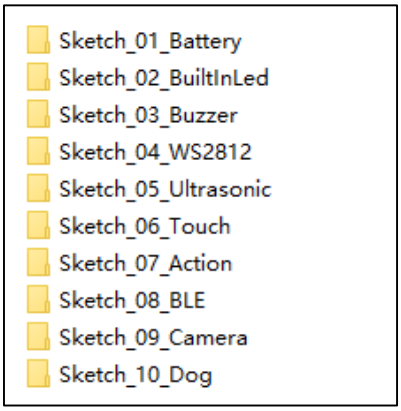
This code uses a library named "Freenove_WS2812_Lib_for_ESP32". If you have not installed it, please do so first.

How to install the library

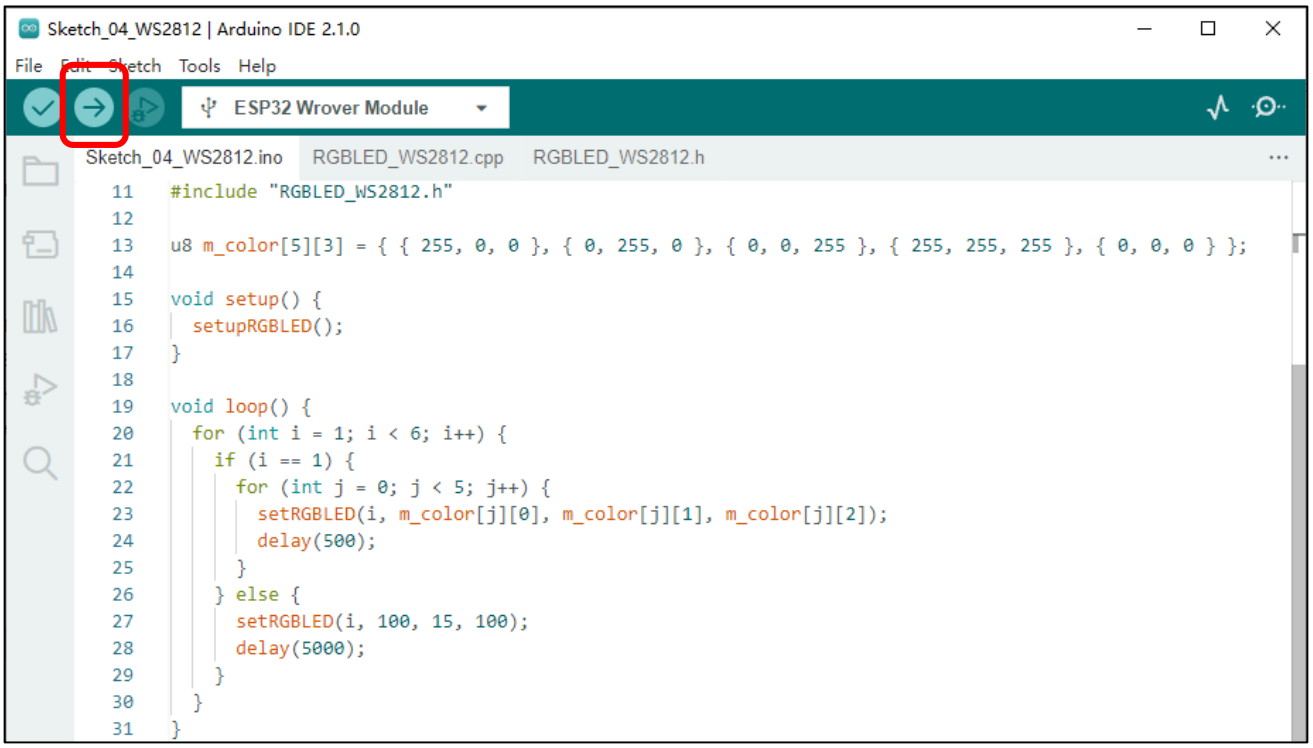
Click Library Manager on the left, enter “Freenove” in the search bar.



Open "Sketch_04_WS2812" folder in **"Freenove_Robot_Dog_Kit_for_ESP32\Sketches"** and then double-click "Sketch_04_WS2812.ino".



Sketch_04_WS2812



Upload the sketch to the esp32, the four WS2812 LEDs on the robot board will switch a colored light display mode every 5 seconds.

The following is the code:

```
#include "RGBLED_WS2812.h"

u8 m_color[5][3] = { { 255, 0, 0 }, { 0, 255, 0 }, { 0, 0, 255 }, { 255, 255, 255 }, { 0, 0, 0 } };

void setup() {
  setupRGBLED();
}

void loop() {
  for (int i = 1; i < 6; i++) {
    if (i == 1) {
      for (int j = 0; j < 5; j++) {
        setRGBLED(i, m_color[j][0], m_color[j][1], m_color[j][2]);
        delay(500);
      }
    } else {
      setRGBLED(i, 100, 15, 100);
      delay(5000);
    }
  }
}
```

Initialize the WS2812 LED pins.

```
  setupRGBLED();
```

When i=1, control the RGB LEDS to display the specified color. When i is not equal to 1, other patterns of colored lights are displayed.

```
    setRGBLED(i, m_color[j][0], m_color[j][1], m_color[j][2]);
```

RGBLED_WS2812.h

```
#ifndef _RGBLED_WS2812_h
#define _RGBLED_WS2812_h

#if defined(ARDUINO) && ARDUINO >= 100
#include "Arduino.h"
#else
#include "WProgram.h"
#endif

#include "Freenove_WS2812_Lib_for_ESP32.h"

#define LEDS_COUNT                4
#define LEDS_PIN                  0
#define CHANNEL                   0

```

```
#define LED_MODE_OFF            0
#define LED_MODE_RGB            1
#define LED_MODE_FOLLOWING      2
#define LED_MODE_BLINK          3
#define LED_MODE_BREATHING      4
#define LED_MODE_RAINBOW        5

void setupRGBLED();
void setRGBLED(uint8_t mode, uint8_t r, uint8_t g, uint8_t b);

void task_showRGBLeds(void *pvParameters);
void task_RGBLeds(void *pvParameters);

#endif
```

RGBLED_WS2812.cpp

```
#include "RGBLED_WS2812.h"

Freenove_ESP32_WS2812 strip = Freenove_ESP32_WS2812(LEDS_COUNT, LEDS_PIN, CHANNEL, TYPE_GRB);

struct LedConfig {
  uint8_t mode;
  uint8_t r;
  uint8_t g;
  uint8_t b;
} ledConfig_t;

int followingColorStep = 0;
int breathingStep = 2;
int rainbowStep = 2;
bool blinkStateUp = true;
bool breathingStateUp = true;
u32 lastStripUpdateTime = 0;

void setupRGBLED() {
  strip.begin();
  strip.setBrightness(255);
  xTaskCreateUniversal(task_RGBLeds, "task_RGBLeds", 4096, NULL, 1, NULL, 1);
}

void setRGBLED(uint8_t mode, uint8_t r, uint8_t g, uint8_t b) {
  ledConfig_t.mode = mode;
  ledConfig_t.r = r;
  ledConfig_t.g = g;
```

```
29    ledConfig_t.b = b;
30  }
31
32  void task_showRGBLeds(void *pvParameters) {
33    switch (ledConfig_t.mode) {
34      case LED_MODE_OFF:
35        if (millis() - lastStripUpdateTime > 500) {
36          for (int i = 0; i < LEDS_COUNT; i++) {
37            strip.setLedColorData(i, 0, 0, 0);
38          }
39          strip.show();
40          lastStripUpdateTime = millis();
41        }
42        break;
43      case LED_MODE_RGB:
44        if (millis() - lastStripUpdateTime > 10) {
45          for (int i = 0; i < LEDS_COUNT; i++) {
46            strip.setLedColorData(i, ledConfig_t.r, ledConfig_t.g, ledConfig_t.b);
47          }
48          strip.show();
49          lastStripUpdateTime = millis();
50        }
51        break;
52      case LED_MODE_FOLLOWING:
53        if (millis() - lastStripUpdateTime > 100) {
54          followingColorStep += 5;
55          int j = followingColorStep;
56          strip.setLedColor(j % 4, strip.Wheel((j / 4 * 86) & 255));
57          lastStripUpdateTime = millis();
58        }
59        break;
60      case LED_MODE_BLINK:
61        if (millis() - lastStripUpdateTime > 500) {
62          if (blinkStateUp) {
63            blinkStateUp = false;
64            for (int i = 0; i < LEDS_COUNT; i++) {
65              strip.setLedColorData(i, ledConfig_t.r, ledConfig_t.g, ledConfig_t.b);
66            }
67            strip.show();
68          } else {
69            blinkStateUp = true;
70            for (int i = 0; i < LEDS_COUNT; i++) {
71              strip.setLedColorData(i, 0, 0, 0);
72            }
```

```
73            strip.show();
74          }
75          lastStripUpdateTime = millis();
76        }
77        break;
78      case LED_MODE_BREATHING:
79        if (millis() - lastStripUpdateTime > 10) {
80          if (breathingStateUp) {
81            breathingStep += 10;
82            if (breathingStep >= 255) {
83              breathingStep = 255;
84              breathingStateUp = false;
85            }
86          } else {
87            breathingStep -= 10;
88            if (breathingStep <= 0) {
89              breathingStep = 0;
90              breathingStateUp = true;
91            }
92          }
93          int j = breathingStep;
94          strip.setBrightness(j);
95          for (int i = 0; i < LEDS_COUNT; i++) {
96            strip.setLedColorData(i, ledConfig_t.r, ledConfig_t.g, ledConfig_t.b);
97          }
98          strip.show();
99          lastStripUpdateTime = millis();
100       }
101       break;
102     case LED_MODE_RAINBOW:
103       if (millis() - lastStripUpdateTime > 10) {
104         rainbowStep += 2;
105         if (rainbowStep >= 255) {
106           rainbowStep = 0;
107         }
108         int j = rainbowStep;
109         for (int i = 0; i < LEDS_COUNT; i++) {
110           strip.setLedColorData(i, strip.Wheel((i * 256 / LEDS_COUNT + j) & 255));
111         }
112         strip.show();
113         lastStripUpdateTime = millis();
114       }
115       break;
116     default:
```

```
117         break;
118     }
119 }
120
121 void task_RGBLeds(void *pvParameters) {
122   while (1) {
123     task_showRGBLeds(NULL);
124   }
125 }
```

Apply for a colored light control object and configure this object.

```
3   Freenove_ESP32_WS2812 strip = Freenove_ESP32_WS2812(LEDS_COUNT, LEDS_PIN, CHANNEL, TYPE_GRB);
```

Initialize the RGB LEDs, and set their brightness to the maximum value. Start a thread and set task_RGBLeds() as the callback function of the thread.

```
19  void setupRGBLED() {
20    strip.begin();
21    strip.setBrightness(255);
22    xTaskCreateUniversal(task_RGBLeds, "task_RGBLeds", 4096, NULL, 1, NULL, 1);
23  }
```

RGB LEDs configuration function. Call this function to set the colored light mode and the red, green, and blue color values of the LEDs.

```
25  void setRGBLED(uint8_t mode, uint8_t r, uint8_t g, uint8_t b) {
26    ledConfig_t.mode = mode;
27    ledConfig_t.r = r;
28    ledConfig_t.g = g;
29    ledConfig_t.b = b;
30  }
```

LED display function. The thread will control the RGB LEDs to emit different colors and phenomena according to the configuration information of ledConfig_t.

```
32  void task_showRGBLeds(void *pvParameters) {
33    switch (ledConfig_t.mode) {
34      case LED_MODE_OFF:
...       ......
42        break;
43      case LED_MODE_RGB:
...       ......
51        break;
52      case LED_MODE_FOLLOWING:
...       ......
59        break;
60      case LED_MODE_BLINK:
...       ......
77        break;
78      case LED_MODE_BREATHING:
...       ......
```

```
101         break;
102       case LED_MODE_RAINBOW:
...         ......
115         break;
116       default:
117         break;
118     }
119   }
```

Thread callback function, when the thread starts, it will execute the color light display function cyclically.

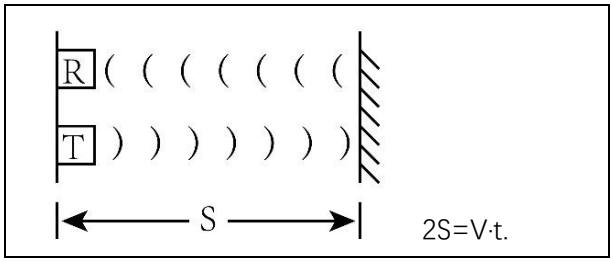
```
121   void task_RGBLeds(void *pvParameters) {
122     while (1) {
123       task_showRGBLeds(NULL);
124     }
125   }
```

Chapter 10 Ultrasonic Ranging

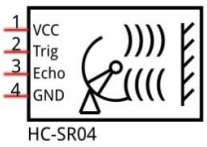
If you have any concerns, please feel free to contact us via support@freenove.com

Component Knowledge

The ultrasonic ranging module uses the principle that ultrasonic waves will be sent back when encounter obstacles. We can measure the distance by counting the time interval between sending and receiving of the ultrasonic waves, and the time difference 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, about v=340m/s, we can calculate the distance between the ultrasonic ranging module and the obstacle: s=vt/2.



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:

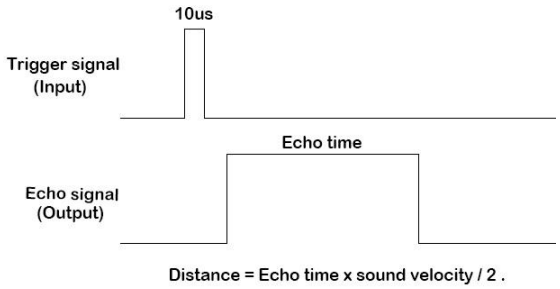
Pin	Description
VCC	power supply pin
Trig	trigger pin
Echo	Echo pin
GND	GND

Technical specs:

Working voltage: 5V Working current: 12mA

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

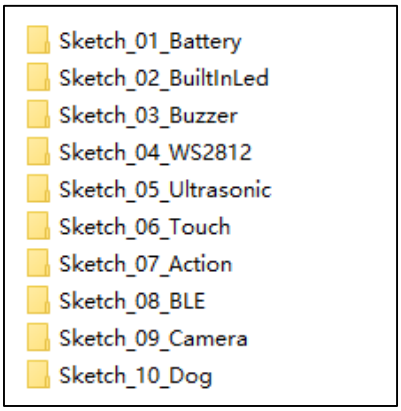
Instructions for use: output a high-level pulse in Trig pin lasting for least 10us, the module begins to transmit ultrasonic waves. At the same time, the Echo pin is pulled up. When the module receives the returned ultrasonic waves from encountering an obstacle, the Echo pin will be pulled down. The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



Sketch

In this chapter, we use GPIO32 of ESP32 as the Trig pin of the ultrasonic module, and GPIO12 as the Echo pin of the ultrasonic module.

Open "Sketch_05_Ultrasonic" folder in **"Freenove_Robot_Dog_Kit_for_ESP32\Sketches"** and then double-click "Sketch_05_Ultrasonic.ino".



Sketch_05_Ultrasonic

Upload the sketch to ESP32 and the robot will acquire ultrasonic data every 500 milliseconds and prints them out through the serial port.

The following is the code:

```
#include "UltrasonicRanging.h"

void setup() {
  Serial.begin(115200);
  setupSonar();
}

void loop() {
  float dist = getSonar();
  Serial.printf("dist: %.2fcm\r\n", dist);
  delay(500);
}
```

Initialize the pins of the ultrasonic module.

```
  setupSonar();
```

Get the data value of the ultrasonic module, the unit is cm.

```
  float dist = getSonar();
```

UltrasonicRanging.h

```
#ifndef _ULTRASONICRANGING_h
#define _ULTRASONICRANGING_h

#if defined(ARDUINO) && ARDUINO >= 100
#include "Arduino.h"
#else
#include "WProgram.h"
#endif

#define PIN_TRIG 32      //Ultrasonic trig pin
#define PIN_ECHO 12    //Ultrasonic echo pin

void setupSonar();
float getSonar();

#endif
```

UltrasonicRanging.cpp

```
#include "UltrasonicRanging.h"

constexpr int MAX_DISTANCE = 300;                // Maximum sensor distance is rated at 400-500cm.
constexpr float timeOut = MAX_DISTANCE * 60;  // timeOut= 2*MAX_DISTANCE /100 /340 *1000000 = MAX_DISTANCE*58.8
constexpr int soundVelocity = 340;                  // define sound speed=340m/s

void setupSonar() {
  pinMode(PIN_TRIG, OUTPUT);  // set trigPin to output mode
  pinMode(PIN_ECHO, INPUT);   // set echoPin to input mode
}
float getSonar() {
  unsigned long pingTime, t0, t1;
  float distance;
  // make PIN_TRIG output high level lasting for 10μs to triger HC_SR04
  digitalWrite(PIN_TRIG, HIGH);
  delayMicroseconds(10);
  digitalWrite(PIN_TRIG, LOW);
  // Wait HC-SR04 returning to the high level and measure out this waitting time
  t0 = micros();
  pingTime = pulseIn(PIN_ECHO, HIGH, timeOut);  // unit: us
  t1 = micros() - t0;
  // calculate the distance according to the time
  if (t1 < timeOut) {
    distance = (float)pingTime * soundVelocity / 2 / 10000;
  } else {
    distance = MAX_DISTANCE;
  }
  return distance;  // return the distance value
}
```

Ultrasonic module initialization function.

```
void setupSonar() {
  pinMode(PIN_TRIG, OUTPUT);  // set trigPin to output mode
  pinMode(PIN_ECHO, INPUT);   // set echoPin to input mode
}
```

The function that gets ultrasonic data.

```
11  float getSonar() {
12    unsigned long pingTime, t0, t1;
13    float distance;
14    // make PIN_TRIG output high level lasting for 10μs to triger HC_SR04
15    digitalWrite(PIN_TRIG, HIGH);
16    delayMicroseconds(10);
17    digitalWrite(PIN_TRIG, LOW);
18    // Wait HC-SR04 returning to the high level and measure out this waitting time
19    t0 = micros();
20    pingTime = pulseIn(PIN_ECHO, HIGH, timeOut);  // unit: us
21    t1 = micros() - t0;
22    // calculate the distance according to the time
23    if (t1 < timeOut) {
24      distance = (float)pingTime * soundVelocity / 2 / 10000;
25    } else {
26      distance = MAX_DISTANCE;
27    }
28    return distance;  // return the distance value
29  }
```

About function **pulseIn**():

int pulseIn(int pin, int level, int timeout);

pin: the number of the Arduino pin on which you want to read the pulse. Allowed data types: int.
value: type of pulse to read: either HIGH or LOW. Allowed data types: int.
timeout (optional): the number of microseconds to wait for the pulse to start; default is one second.

Chapter 11 Touch

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

Component Knowledge

Touch sensor

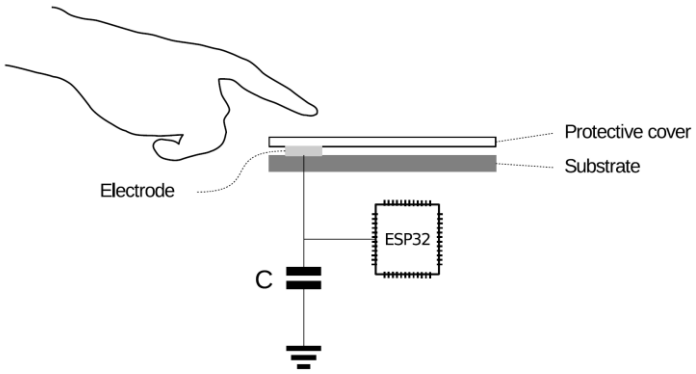
ESP32's touch sensor supports up to 10 GPIO channels as capacitive touch pins. Each pin can be used separately as an independent touch switch or be combined to produce multiple touch points. The following table is a list of available touch pins on ESP32.

Name of touch sensing signal	Functions of pins	GPIO number
T0	GPIO4	GPIO4
T1	GPIO0	GPIO0
T2	GPIO2	GPIO2
T3	MTDO	GPIO15
T4	MTCK	GPIO13
T5	MTDI	GPIO12
T6	MTMS	GPIO14
T7	GPIO27	GPIO27
T8	32K_XN	GPIO33
T9	32K_XP	GPIO32

The touch pin number is already defined in ESP32's code base. For example, in the code, you can use T0 to represent GPIO4.

The electrical signals generated by touch are analog data, which are converted by an internal ADC converter. You may have noticed that all touch pins have ADC functionality.

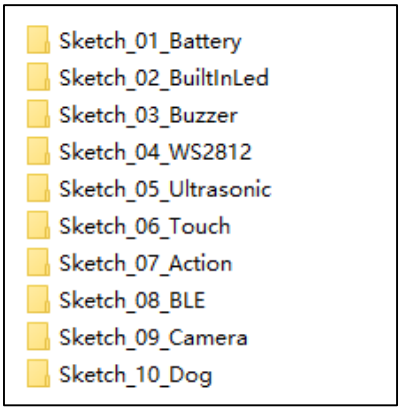
The hardware connection method is shown in the following figure.



In this chapter, we use the GPIO15 pin of ESP32 to simulate the touch function and read it.

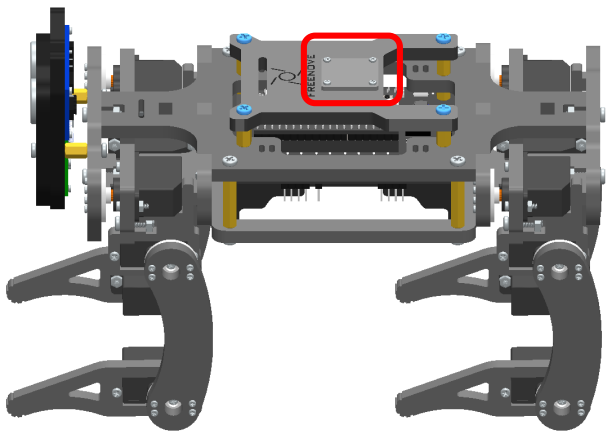
Sketch

Open "Sketch_06_Touch" folder in **"Freenove_Robot_Dog_Kit_for_ESP32\Sketches"** and then double-click "Sketch_06_Touch.ino".



Sketch_06_Touch

Upload the code to esp32, and the robot dog obtains the status of the touch sensor every 100 milliseconds and prints it out through the serial port. Touch the position circled in the figure below with your finger, and observe the changes in the content printed by the serial port.



The following is the code:

```
#include "TouchPad.h"

void setup() {
  Serial.begin(115200);
  setupTouchPad();
}

void loop() {
  int isTouch = getTouch();
  Serial.printf("Touch state: %d\r\n", isTouch);
  delay(100);
}
```

Initialize touch sensing function pins.

```
  setupTouchPad();
```

Get the status value of the touch sensor. When the sensor is not touched, the state value is 0. When it is touched, the status value is 1. The short press status value is 2, and the long press status value is 3.

```
  int isTouch = getTouch();
```

TouchPad.h

```
#ifndef _TOUCHPAD_h
#define _TOUCHPAD_h

#if defined(ARDUINO) && ARDUINO >= 100
#include "Arduino.h"
#else
#include "WProgram.h"
#endif
#include "driver/touch_pad.h"

#define PIN_TOUCH_PAD (TOUCH_PAD_NUM3)
#define TOUCH_THRESH_NO_USE (0)
#define TOUCH_THRESH_PERCENT (80)
#define TOUCHPAD_FILTER_TOUCH_PERIOD (10)

void setupTouchPad(void);

void __attribute__((weak)) isr_touchpad(void *arg);
extern void isr_touchpad(void *arg);
void isr_touchpad();
void task_TouchPad(void *pvParameters);
void task_Touch(void *pvParameters);
int getTouch(void);

#endif
```

TouchPad.cpp

```
#include "TouchPad.h"

static uint32_t s_pad_init_val;
static void tp_example_set_thresholds(void) {
  uint16_t touch_value;
  // read filtered value
  touch_pad_read_filtered(PIN_TOUCH_PAD, &touch_value);
  s_pad_init_val = touch_value;
  // set interrupt threshold.
  touch_pad_set_thresh(PIN_TOUCH_PAD, touch_value * 4 / 5);
}
bool touchPadVal = false;
void isr_touchpad(void *arg) {
  // uint16_t touchVal = 0;
  uint32_t touchStatus = 0;
  touchStatus = touch_pad_get_status();  // TOUCHPAD3 : 0000 1000 = 0x08
  if (touchStatus == 0x08)  // TOUCHPAD3
  {
    touchPadVal = true;
  }
  touch_pad_clear_status();
}

void setupTouchPad(void) {
  // Initialize touch pad peripheral, it will start a timer to run a filter
  ESP_ERROR_CHECK(touch_pad_init());
  // If use interrupt trigger mode, should set touch sensor FSM mode at 'TOUCH_FSM_MODE_TIMER'.
  touch_pad_set_fsm_mode(TOUCH_FSM_MODE_TIMER);
  // Set reference voltage for charging/discharging
  // For most usage scenarios, we recommend using the following combination:
  // the high reference valtage will be 2.7V - 1V = 1.7V, The low reference voltage will be 0.5V.
  touch_pad_set_voltage(TOUCH_HVOLT_2V7, TOUCH_LVOLT_0V5, TOUCH_HVOLT_ATTEN_1V);
  // Init touch pad IO
  touch_pad_config(PIN_TOUCH_PAD, TOUCH_THRESH_NO_USE);
  // Initialize and start a software filter to detect slight change of capacitance.
  touch_pad_filter_start(TOUCHPAD_FILTER_TOUCH_PERIOD);
  // Set thresh hold
  tp_example_set_thresholds();
  // set isr trigger mode
  touch_pad_set_trigger_mode(TOUCH_TRIGGER_BELOW);
  // Register touch interrupt ISR
  touch_pad_isr_register(isr_touchpad, NULL);
  // enable isr
```

```
44    touch_pad_intr_enable();
45
46    xTaskCreateUniversal(task_Touch, "task_Touch", 4096, NULL, 1, NULL, 1);
47  }
48
49  uint8_t touchMechineStatus = 0;
50  uint32_t t0 = millis(),  // start time point
51    t1 = 0,                // update time point
52    t2 = 0,                // release time
53    t3 = 0;                // pressed time
54  void task_TouchPad(void *pvParameters) {
55    switch (touchMechineStatus) {
56      case 0:  // Idle , Wait for the button to press.
57        if (touchPadVal) {
58          t0 = millis();
59          t1 = t0;
60          touchPadVal = false;
61          touchMechineStatus = 1;
62        }
63        break;
64      case 1:  // short press or long press ?
65        if (touchPadVal) {
66          t1 = millis();
67          touchPadVal = false;
68        }
69        t2 = millis() - t1;  // Time since the last press signal was detected
70        t3 = t1 - t0;        // calculate pressed time.
71        if (t3 > 700)        //  long pressed
72        {
73          Serial.printf("long press: %dms\n", t3);
74          touchMechineStatus = 3;  // wait release
75        }
76        if (t2 > 50)  // release time > 50, press time < 700, short press, enter short pressed state
77        {
78          touchMechineStatus = 2;
79        }
80        break;
81      case 2:  // short press
82        Serial.printf("short press: %dms\n", t3);
83        touchMechineStatus = 0;
84        break;
85      case 3:  // Wait for release after long press.
86        if (touchPadVal) {
87          t1 = millis();
```

```
        touchPadVal = false;
      }
      t2 = millis() - t1;
      if (t2 > 50)  // release time > 50
      {
        touchMechineStatus = 0;
      }
      break;
    default:
      break;
  }
}

void task_Touch(void *pvParameters) {
  Serial.printf("task_Touch is running...\r\n");
  while (1) {
    task_TouchPad(NULL);
  }
}

int getTouch(void) {
  return touchMechineStatus;
}
```

ESP32 touch function pin initialization.

```
void setupTouchPad(void) {
  // Initialize touch pad peripheral, it will start a timer to run a filter
  ESP_ERROR_CHECK(touch_pad_init());
  // If use interrupt trigger mode, should set touch sensor FSM mode at 'TOUCH_FSM_MODE_TIMER'.
  touch_pad_set_fsm_mode(TOUCH_FSM_MODE_TIMER);
  // Set reference voltage for charging/discharging
  // For most usage scenarios, we recommend using the following combination:
  // the high reference valtage will be 2.7V - 1V = 1.7V, The low reference voltage will be 0.5V.
  touch_pad_set_voltage(TOUCH_HVOLT_2V7, TOUCH_LVOLT_0V5, TOUCH_HVOLT_ATTEN_1V);
  // Init touch pad IO
  touch_pad_config(PIN_TOUCH_PAD, TOUCH_THRESH_NO_USE);
  // Initialize and start a software filter to detect slight change of capacitance.
  touch_pad_filter_start(TOUCHPAD_FILTER_TOUCH_PERIOD);
  // Set thresh hold
  tp_example_set_thresholds();
  // set isr trigger mode
  touch_pad_set_trigger_mode(TOUCH_TRIGGER_BELOW);
  // Register touch interrupt ISR
  touch_pad_isr_register(isr_touchpad, NULL);
  // enable isr
```

```
  touch_pad_intr_enable();

  xTaskCreateUniversal(task_Touch, "task_Touch", 4096, NULL, 1, NULL, 1);
}
```

Touch sensor thread callback function.

```
void task_Touch(void *pvParameters) {
  Serial.printf("task_Touch is running...\r\n");
  while (1) {
    task_TouchPad(NULL);
  }
}
```

Touch function threshold setting function. When the sensor is touched, the internal capacitor discharges, and the data change can be detected at the touch pin. Here, 4/5 of the value of the touch sensor without contact is set as the threshold trigger range.

```
static uint32_t s_pad_init_val;
static void tp_example_set_thresholds(void) {
  uint16_t touch_value;
  // read filtered value
  touch_pad_read_filtered(PIN_TOUCH_PAD, &touch_value);
  s_pad_init_val = touch_value;
  // set interrupt threshold.
  touch_pad_set_thresh(PIN_TOUCH_PAD, touch_value * 4 / 5);
}
```

Touch sensor interrupt trigger function. Execute this function every time the touch sensor is triggered, to set touchPadVal to true.

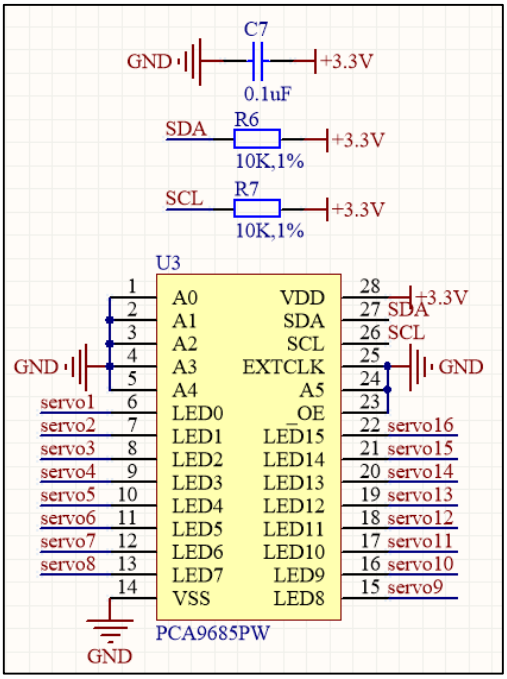
```
bool touchPadVal = false;
void isr_touchpad(void *arg) {
  // uint16_t touchVal = 0;
  uint32_t touchStatus = 0;
  touchStatus = touch_pad_get_status();  // TOUCHPAD3 : 0000 1000 = 0x08
  if (touchStatus == 0x08)  // TOUCHPAD3
  {
    touchPadVal = true;
  }
  touch_pad_clear_status();
}
```

Chapter 12 Basic Motion

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

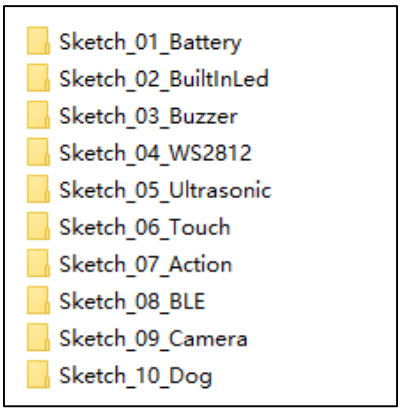
Schematic

As shown in the picture, we connect the SDA and SCL pins of PCA9685 chip to GPIO13 and GPIO14 of ESP32 respectively. In this way, we can communicate with PCA9685 through IIC, and control the servo through PCA9685 to rotate to any angle.



Sketch

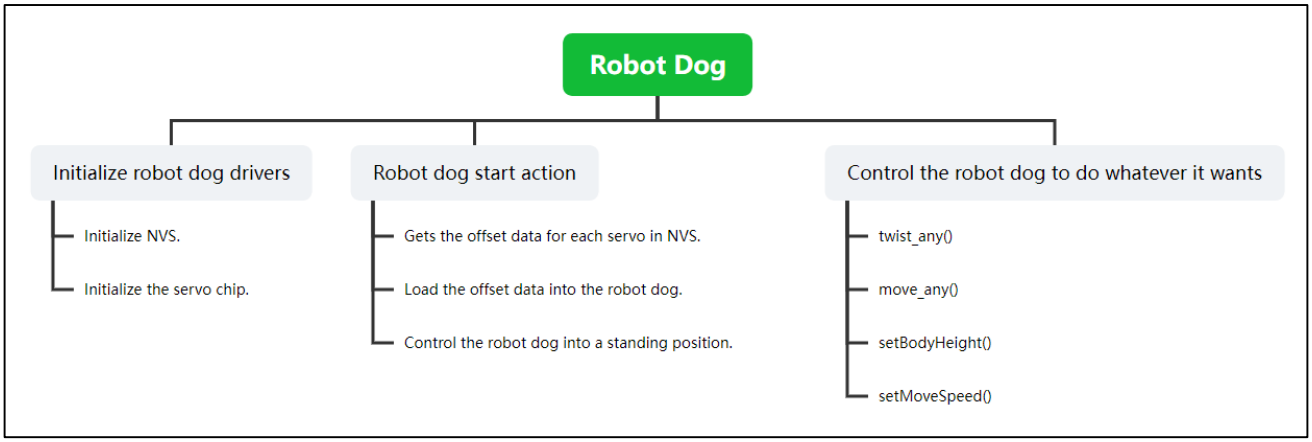
Open "Sketch_07_Action" folder in **"Freenove_Robot_Dog_Kit_for_ESP32\Sketches"** and then double-click "Sketch_07_Action.ino".



Sketch_07_Action

Upload the code to esp32 and the robot dog will make various actions.

Please note: If your robot dog has not been calibrated, please navigate back to Chapter 2 to calibrate the robot dog first.



The following is the code:

```
1  #include "Motion.h"
```

```
2
3   /*
4   Please note that the following functions are based on the premise that the robot dog has been calibrated.
5   If your robot dog has not been calibrated yet, calibrate before learning.
6   Otherwise, the steering gear may be damaged, or the machine action cannot achieve the desired purpose.
7   */
8
9   void setup() {
10    Serial.begin(115200);
11    Serial.println("\n\nProgram begin ... ");
12    prefs.begin(NMSPC_STORAGE);
13    pca.begin();
14    pca.releaseAllServo();
15    getServoOffsetFromStorage();
16
17    standUp();
18    delay(3000);
19
20    const uint8_t repeatCounts = 3;
21    const int8_t acts[][2][3] = {
22      { { 0, 0, 10 }, { 0, 0, -10 } },  //Rotate horizontally clockwise, counterclockwise horizontal rotation.
23      { { 0, 10, 0 }, { 0, -10, 0 } },  //High on the left and low on the right, low on the left and high on the right.
24      { { 10, 0, 0 }, { -10, 0, 0 } }   //High at the front and low at the back, low at the front and high at the back.
25    };
26    for (uint8_t i = 0; i < sizeof(acts) / sizeof(acts[0]); i++) {
27      for (int k = 0; k < repeatCounts; k++) {
28        for (int j = 0; j < 2; j++) {
29          twist_any(acts[i][j][0], acts[i][j][1], acts[i][j][2]);
30          delay(20);
31        }
32      }
33    }
34
35    //Rotate horizontally clockwise, and high at the front and low at the back, low at the front and high at the
36  back.
37    int8_t x = 0, y = 0, z = 0;
38    for (uint8_t j = 0; j < repeatCounts; j++) {
39      for (int i = 0; i < 360; i += 30) {
40        x = 10 * sin((double)i * PI / 180);
41        z = 10 * cos((double)i * PI / 180);
42        twist_any(x, y, z);
43      }
44    }
45    twist_any(0, 0, 0);  //Normal standing
```

```
46
47    //The whole body goes up and down.
48    for (int i = 0; i < repeatCounts; i++) {
49      setBodyHeight(BODY_HEIGHT_MIN);
50      delay(200);
51      setBodyHeight(BODY_HEIGHT_MAX);
52      delay(200);
53    }
54
55    //Move forward (10*10)mm in the positive direction of the x axis
56    for (int i = 0; i < 10; i++) {
57      move_any(0, 10, 0);
58    }
59    //Move back (10*10)mm in the positive direction of the x axis
60    for (int i = 0; i < 10; i++) {
61      move_any(0, -10, 0);
62    }
63
64    //Control the robot dog to sit down and disable all servoes to reduce power consumption
65    setBodyHeight(BODY_HEIGHT_MIN);
66    pca.releaseAllServo();
67  }
68
69  void loop() {
70    delay(10000);
71  }
```

Before controlling the robot dog to make various actions, you need to initialize the robot dog first, and get the calibration data of the servos from NVS If your robot dog has not been calibrated, please navigate back to Chapter 2 to calibrate the robot dog first.

```
12    prefs.begin(NMSPC_STORAGE);
13    pca.begin();
14    pca.releaseAllServo();
15    getServoOffsetFromStorage();
```

The robot dog action speed setting function, with the parameter ranging from 1 to 8. The larger the value, the faster the robot dog.

```
      void  setMoveSpeed(int spd);
```

The robot dog twisting function. By modifying different parameters, you can make the robot dog perform various body twisting movements.

```
      void  twist_any(int alpha, int beta, int gama);
```

Robot dog movement function, by modifying the parameters, control the robot dog to go forward at any direction, any step length, any rotation Angle, and any speed.

```
      * @brief Walk command, any direction, any step length, any spin angle, any speed
      *
```

```
 * @param alpha is the moving direction, with the x-axis direction as 0 degrees, counterclockwise as positive, clockwise as negative, and the unit is angle, [0-360]. The x direction is the direction of forward movement.
 * @param stepLength The length of each step (<=20).
 * @param gama Spin angle, in-situ rotation, positive counterclockwise, negative clockwise, in degrees. [0-360].
 * @param spd Movement speed, unit:  mm / 10ms.  [1,8]
void move_any(int alpha, float stepLength, int gama, int spd = 5);
```

Robot dog height setting function. The value ranges from 55 to 120.

```
void setBodyHeight(int h);
```

Servo unloading function. Each time this function is called, the servos will lose torque

```
pca.releaseAllServo();
```

Robot dog standing function. Every time this function is called, the robot dog will return to the standing position.

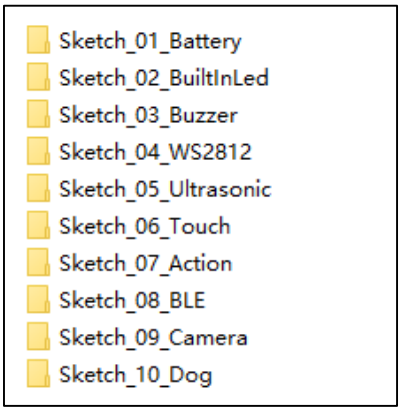
```
standUp();
```

Chapter 13 BLE

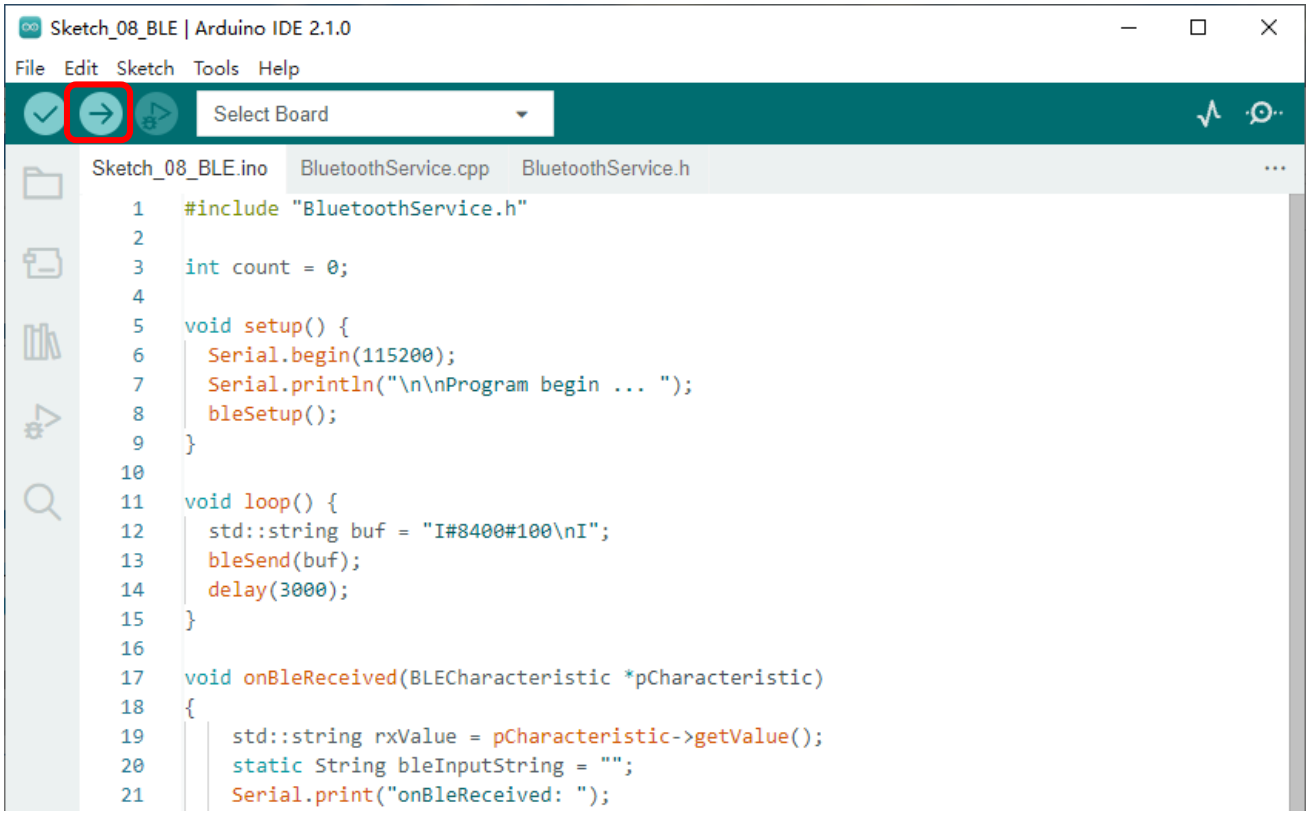
If you have any concerns, please feel free to contact us via support@freenove.com

Sketch

Open "Sketch_08_BLE" folder in **"Freenove_Robot_Dog_Kit_for_ESP32\Sketches"** and then double-click "Sketch_08_BLE.ino".

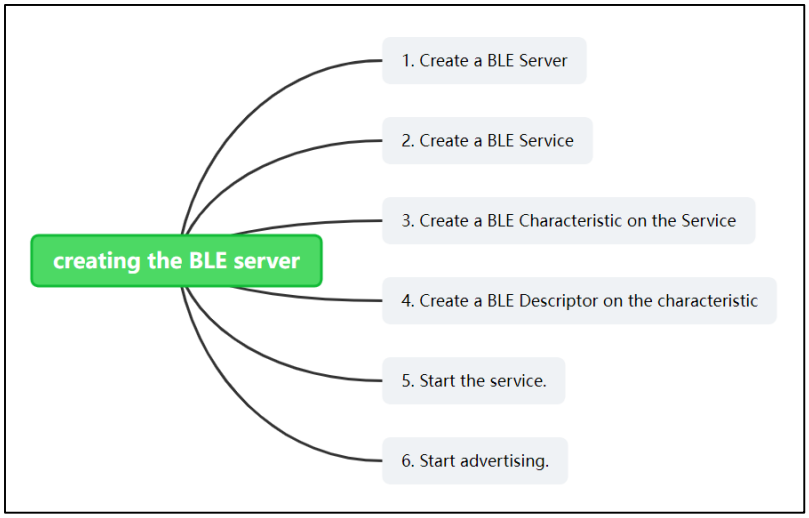


Sketch_08_BLE



Upload the code to esp32. Open the serial monitor and set the baud rate to 115200. Connect the mobile APP to the robot dog, and you can see all the commands sent by the mobile terminal on the serial monitor.

BLE Bluetooth data transparent transmission routine:



The following is the code:

```
#include "BluetoothService.h"

void setup() {
  Serial.begin(115200);
  Serial.println("\n\nProgram begin ... ");
  bleSetup();
}

void loop() {
  std::string buf = "I#8400#100\nI";
  bleSend(buf);
  delay(3000);
}

void onBleReceived(BLECharacteristic *pCharacteristic) {
  std::string rxValue = pCharacteristic->getValue();
  static String bleInputString = "";
  Serial.print("onBleReceived: ");
  if (rxValue.length() > 0) {
    for (int i = 0; i < rxValue.length(); i++) {
      bleInputString += rxValue[i];
      if (rxValue[i] == '\n') {
        Serial.println(bleInputString);
        /*You can do something with the information you receive*/
        bleInputString = "";  //Clear bleInputString.
      }
    }
  }
}
```

BLE Bluetooth initialization function.

```
6    bleSetup();
```

BLE Bluetooth send data function.

```
11   bleSend(buf);
```

Bluetooth low energy data receiving function.

```
15  void onBleReceived(BLECharacteristic *pCharacteristic) {
16    std::string rxValue = pCharacteristic->getValue();
17    static String bleInputString = "";
18    Serial.print("onBleReceived: ");
19    if (rxValue.length() > 0) {
20      for (int i = 0; i < rxValue.length(); i++) {
21        bleInputString += rxValue[i];
22        if (rxValue[i] == '\n') {
23          Serial.println(bleInputString);
24          /*You can do something with the information you receive*/
25          bleInputString = "";  //Clear bleInputString.
26        }
27      }
28    }
29  }
```

BluetoothService.h

```
1   #ifndef __BLUETOOTHSERVICE_H
2   #define __BLUETOOTHSERVICE_H
3
4   #include "Arduino.h"
5
6   #include <BLEDevice.h>
7   #include <BLEServer.h>
8   #include <BLEUtils.h>
9   #include <BLE2902.h>
10
11  class BLEServer;
12  class BLECharacteristic;
13  class BLEServerCallbacks;
14
15  void bleSetup();
16  void bleStop();
17  void bleSend(std::string msg);
18  extern void onBleReceived(BLECharacteristic *pCharacteristic);
19
20  #endif
```

BluetoothService.cpp

```
#include "BluetoothService.h"

// See the following for generating UUIDs:
// https://www.uuidgenerator.net/

#define SERVICE_UUID "0000ffe0-0000-1000-8000-00805f9b34fb"  // UART service UUID
#define CHARACTERISTIC_UUID_RX "0000ffe1-0000-1000-8000-00805f9b34fb"
#define CHARACTERISTIC_UUID_TX "0000ffe1-0000-1000-8000-00805f9b34fb"

BLEServer *pServer = NULL;
BLECharacteristic *pTxCharacteristic;
bool isBleConnected = false;
bool oldBleConnected = false;
uint8_t txValue = 0;

class MyServerCallbacks : public BLEServerCallbacks {
  void onConnect(BLEServer *pServer) {
    isBleConnected = true;
    // connecting
    if (isBleConnected && !oldBleConnected) {
      // do stuff here on connecting
      oldBleConnected = isBleConnected;
      Serial.println("Ble Connect");
    }
  };

  void onDisconnect(BLEServer *pServer) {
    isBleConnected = false;
    // disconnecting
    if (!isBleConnected && oldBleConnected) {
      oldBleConnected = isBleConnected;
      Serial.println("Ble Disconnect");
      delay(500);                  // give the bluetooth stack the chance to get things ready
      pServer->startAdvertising();  // restart advertising
    }
  }
};

class MyCallbacks : public BLECharacteristicCallbacks {
  void onWrite(BLECharacteristic *pCharacteristic) {
    onBleReceived(pCharacteristic);
  }
};
```

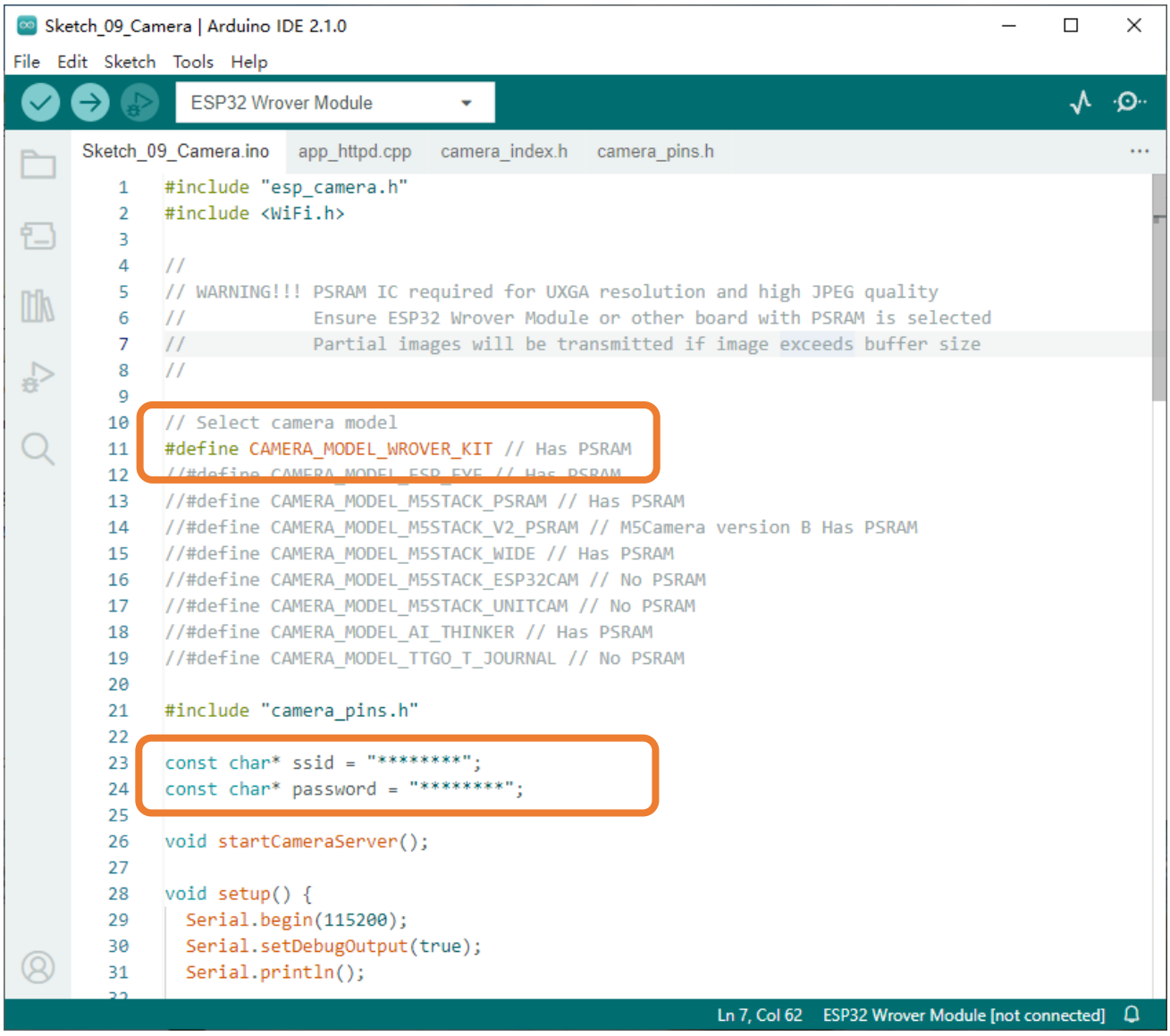
```
44
45  void bleSetup() {
46    // Create the BLE Device
47    String s = String("Freenove-Dog");
48    BLEDevice::init(std::string(s.c_str()));
49    BLEDevice::setMTU(512);
50    //  Create the BLE Server
51    pServer = BLEDevice::createServer();
52    pServer->setCallbacks(new MyServerCallbacks());
53    // Create the BLE Service
54    BLEService *pService = pServer->createService(SERVICE_UUID);
55    // Create a BLE Characteristic
56    pTxCharacteristic = pService->createCharacteristic(
57      CHARACTERISTIC_UUID_TX,
58      BLECharacteristic::PROPERTY_NOTIFY);
59    pTxCharacteristic->addDescriptor(new BLE2902());
60    BLECharacteristic *pRxCharacteristic = pService->createCharacteristic(
61      CHARACTERISTIC_UUID_RX,
62      BLECharacteristic::PROPERTY_WRITE);
63    pRxCharacteristic->setCallbacks(new MyCallbacks());
64    // Start the service
65    pService->start();
66    // Start advertising
67    pServer->getAdvertising()->start();
68    Serial.println("Bluetooth initialization complete, Waiting a client connection to notify...");
69  }
70
71  void bleStop() {
72    BLEDevice::deinit();
73  }
74
75  void bleSend(std::string msg) {
76    if (isBleConnected) {
77      pTxCharacteristic->setValue(msg);
78      pTxCharacteristic->notify();
79    }
80  }
```

Chapter 14 Camera Web Server

In this section, we'll use ESP32's video function as an example to study.
Connect ESP32 using USB and check its IP address through serial monitor. Use web page to access IP address to obtain video and image data.

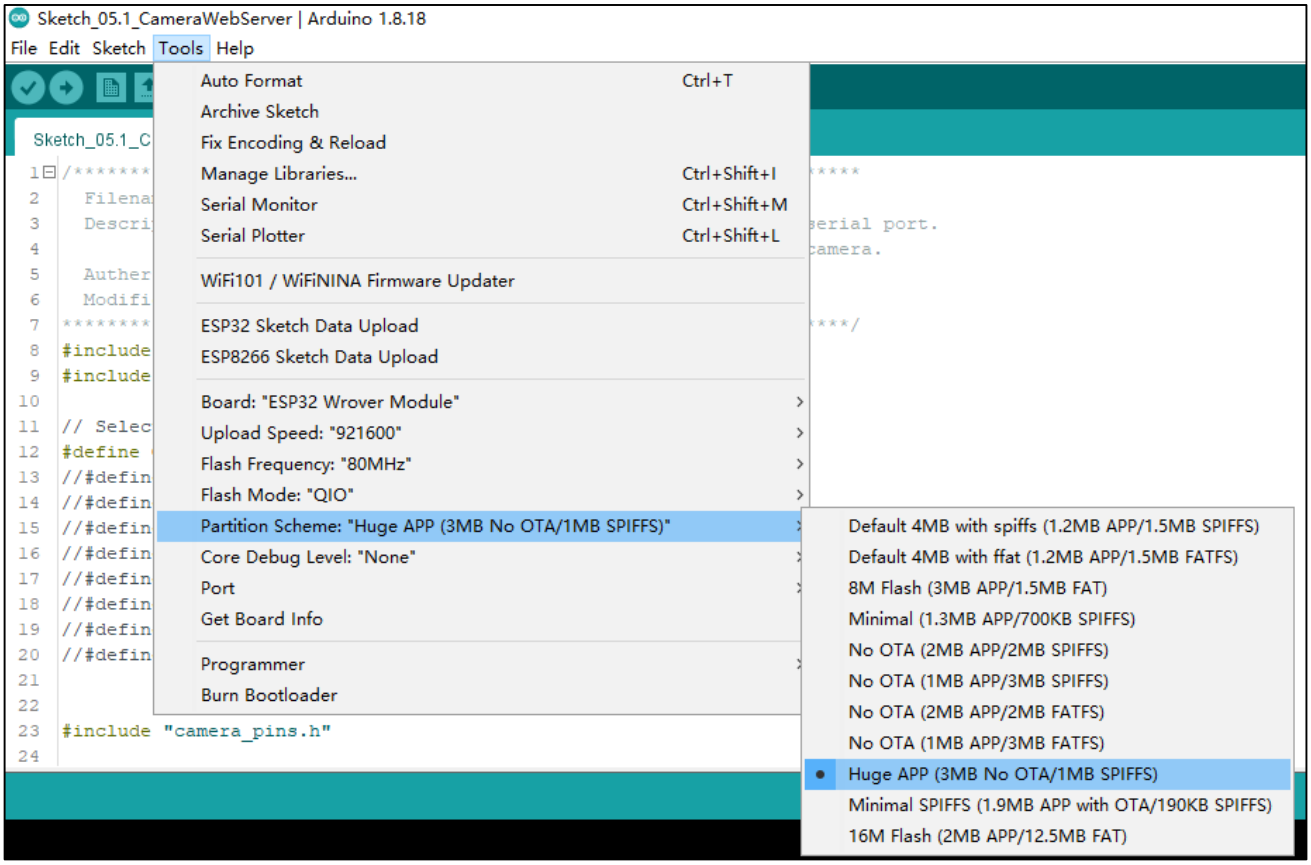
Sketch

Sketch_09_Camera

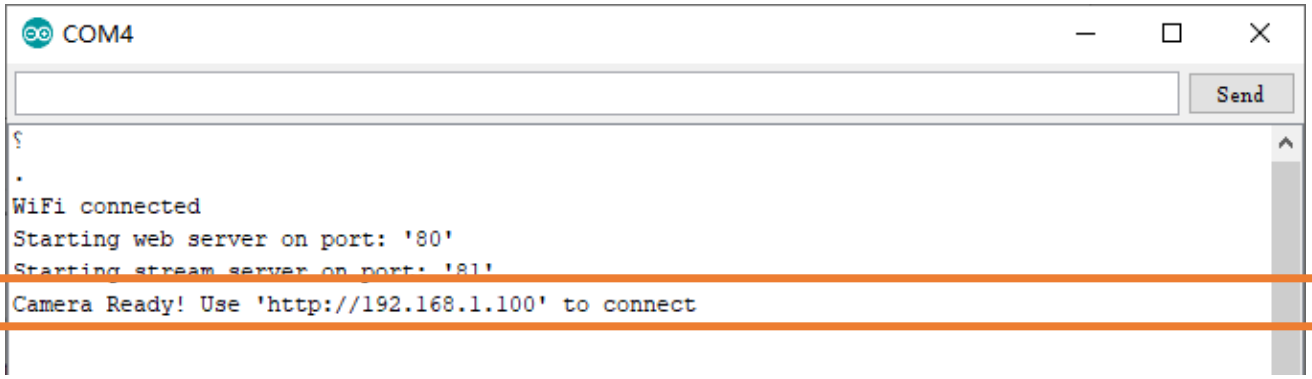


Before running the program, please modify your router's name and password in the box shown in the illustration above to make sure that your Sketch can compile and work successfully.

If your Arduino IDE prompts you that your sketch is out of your project's storage space, compile the code again as configured below.

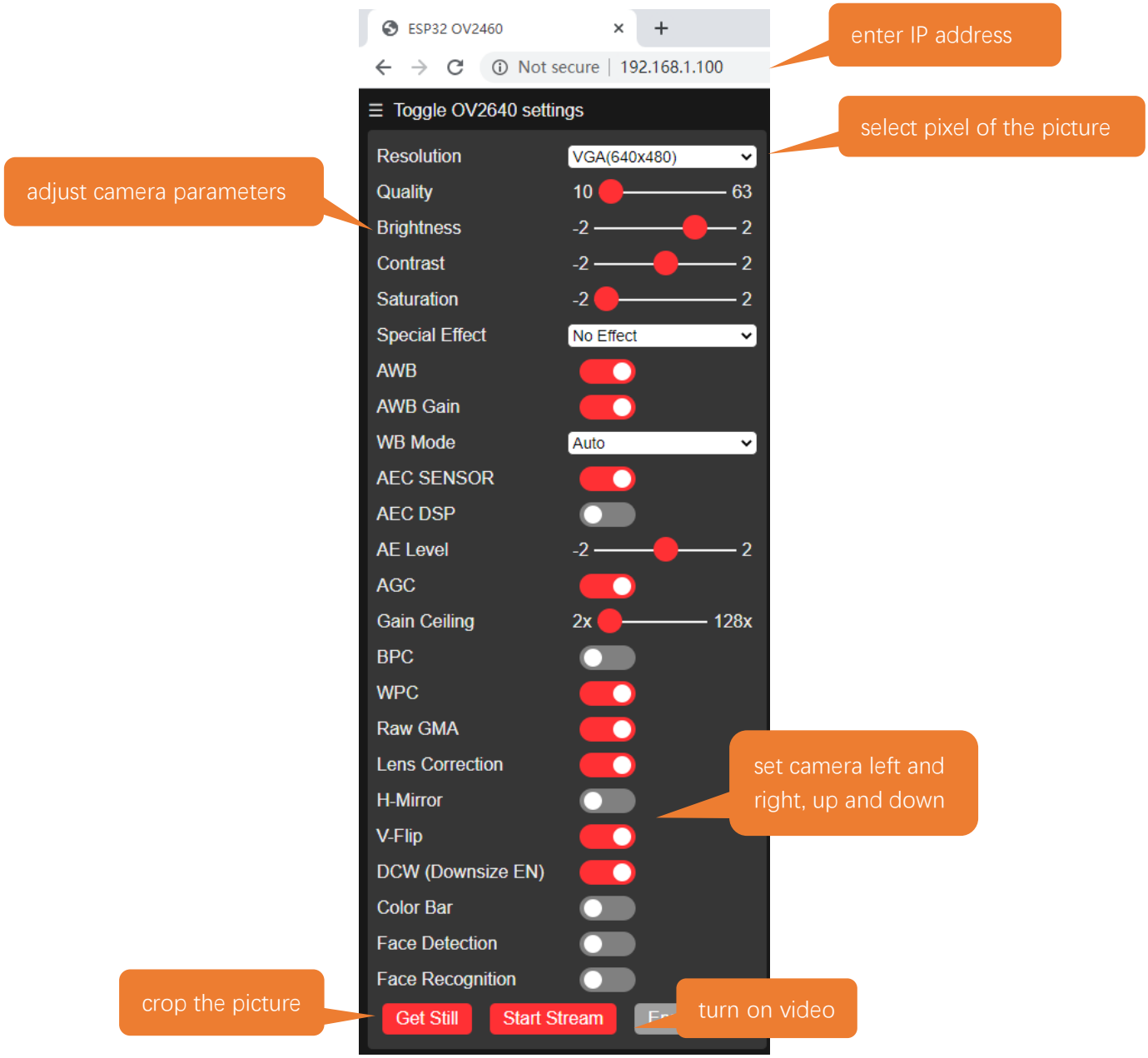


Compile and upload codes to ESP32, open the serial monitor and set the baud rate to 115200, and the serial monitor will print out a network link address.

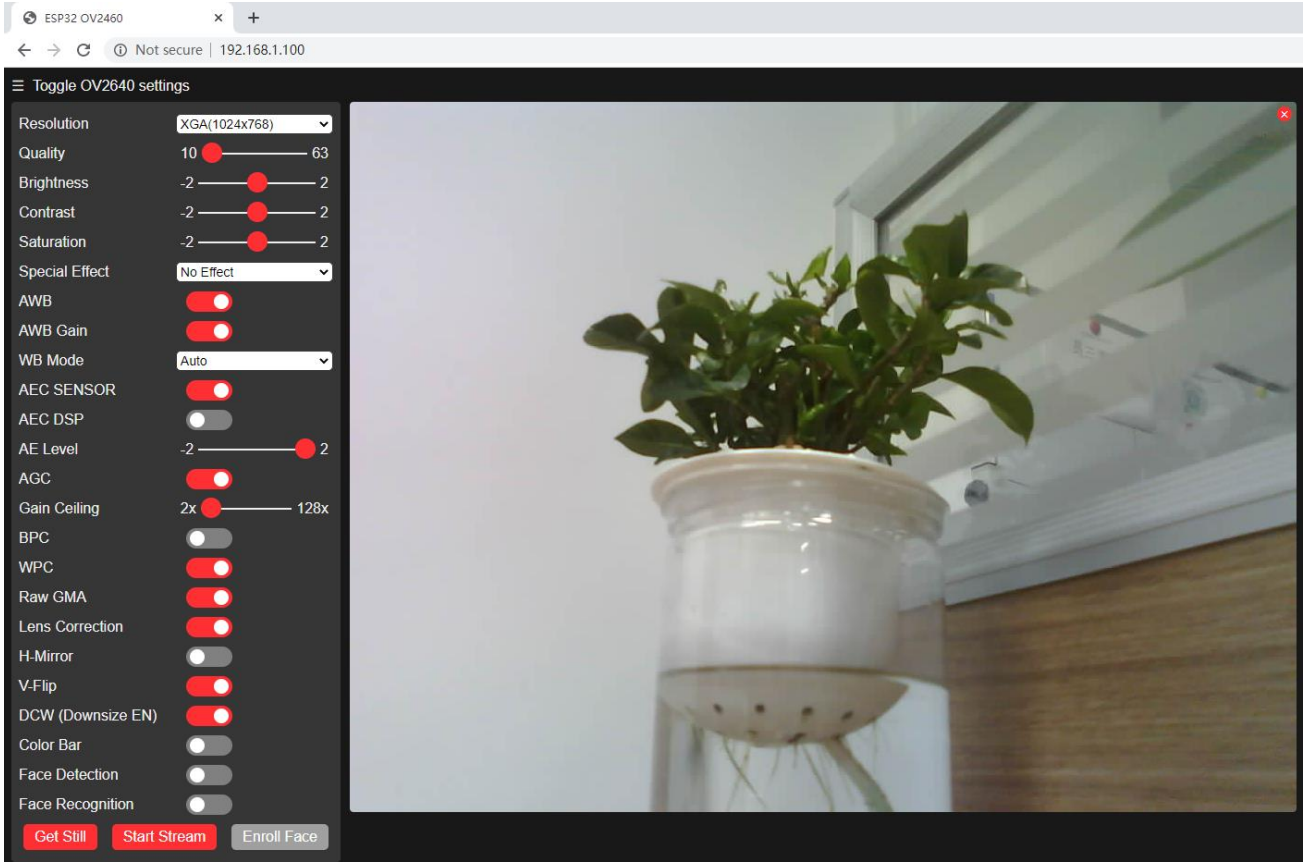


If your ESP32 has been in the process of connecting to router, but the information above has not been printed out, please re-check whether the router name and password have been entered correctly and press the reset key on ESP32-WROVER to wait for a successful connection prompt.

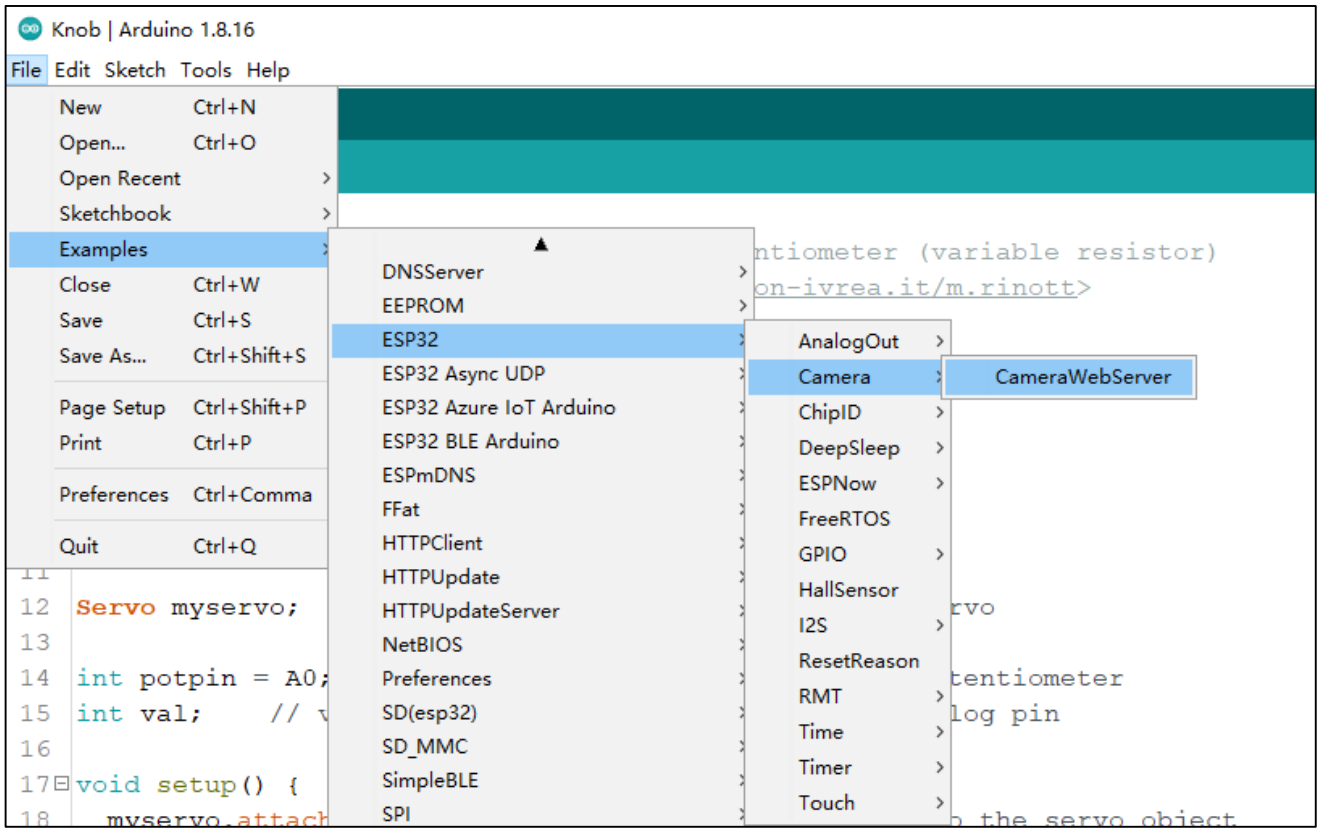
Open a web browser, enter the IP address printed by the serial monitor in the address bar, and access it. Taking the Google browser as an example, here's what the browser prints out after successful access to ESP32's IP.


ESP32 OV2460
Not secure | 192.168.1.100
enter IP address
Toggle OV2640 settings
select pixel of the picture
adjust camera parameters
Resolution
VGA(640x480)
Quality
10
63
Brightness
-2
2
Contrast
-2
2
Saturation
-2
2
Special Effect
No Effect
AWB
AWB Gain
WB Mode
Auto
AEC SENSOR
AEC DSP
AE Level
-2
2
AGC
Gain Ceiling
2x
128x
BPC
WPC
Raw GMA
Lens Correction
H-Mirror
V-Flip
DCW (Downsize EN)
Color Bar
Face Detection
Face Recognition
set camera left and right, up and down
crop the picture
Get Still
Start Stream
turn on video

Click on Start Stream. The effect is shown in the image below.



Note: If sketch compilation fails due to ESP32 support package, follow the steps of the image to open the CameraWebServer. This sketch is the same as described in the tutorial above.



The following is the main program code. You need include other code files in the same folder when write your own code.

```
#include "esp_camera.h"
#include <WiFi.h>

//
// WARNING!!! PSRAM IC required for UXGA resolution and high JPEG quality
//            Ensure ESP32 Wrover Module or other board with PSRAM is selected
//            Partial images will be transmitted if image exceeds buffer size
//

// Select camera model
#define CAMERA_MODEL_WROVER_KIT // Has PSRAM
//#define CAMERA_MODEL_ESP_EYE // Has PSRAM
//#define CAMERA_MODEL_M5STACK_PSRAM // Has PSRAM
//#define CAMERA_MODEL_M5STACK_V2_PSRAM // M5Camera version B Has PSRAM
//#define CAMERA_MODEL_M5STACK_WIDE // Has PSRAM
//#define CAMERA_MODEL_M5STACK_ESP32CAM // No PSRAM
//#define CAMERA_MODEL_M5STACK_UNITCAM // No PSRAM
//#define CAMERA_MODEL_AI_THINKER // Has PSRAM
//#define CAMERA_MODEL_TTGO_T_JOURNAL // No PSRAM

#include "camera_pins.h"

const char* ssid = "********";
const char* password = "********";

void startCameraServer();

void setup() {
  Serial.begin(115200);
  Serial.setDebugOutput(true);
  Serial.println();

  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
```

```
43    config.pin_d7 = Y9_GPIO_NUM;
44    config.pin_xclk = XCLK_GPIO_NUM;
45    config.pin_pclk = PCLK_GPIO_NUM;
46    config.pin_vsync = VSYNC_GPIO_NUM;
47    config.pin_href = HREF_GPIO_NUM;
48    config.pin_sccb_sda = SIOD_GPIO_NUM;
49    config.pin_sccb_scl = SIOC_GPIO_NUM;
50    config.pin_pwdn = PWDN_GPIO_NUM;
51    config.pin_reset = RESET_GPIO_NUM;
52    config.xclk_freq_hz = 20000000;
53    config.pixel_format = PIXFORMAT_JPEG;
54
55    // if PSRAM IC present, init with UXGA resolution and higher JPEG quality
56    //                     for larger pre-allocated frame buffer.
57    if(psramFound()){
58      config.frame_size = FRAMESIZE_UXGA;
59      config.jpeg_quality = 10;
60      config.fb_count = 2;
61    } else {
62      config.frame_size = FRAMESIZE_SVGA;
63      config.jpeg_quality = 12;
64      config.fb_count = 1;
65    }
66
67  #if defined(CAMERA_MODEL_ESP_EYE)
68    pinMode(13, INPUT_PULLUP);
69    pinMode(14, INPUT_PULLUP);
70  #endif
71
72    // camera init
73    esp_err_t err = esp_camera_init(&config);
74    if (err != ESP_OK) {
75      Serial.printf("Camera init failed with error 0x%x", err);
76      return;
77    }
78
79    sensor_t * s = esp_camera_sensor_get();
80    // initial sensors are flipped vertically and colors are a bit saturated
81    if (s->id.PID == OV3660_PID) {
82      s->set_vflip(s, 1); // flip it back
83      s->set_brightness(s, 1); // up the brightness just a bit
84      s->set_saturation(s, -2); // lower the saturation
85    }
86    // drop down frame size for higher initial frame rate
```

```
  s->set_framesize(s, FRAMESIZE_QVGA);

#if defined(CAMERA_MODEL_M5STACK_WIDE) || defined(CAMERA_MODEL_M5STACK_ESP32CAM)
  s->set_vflip(s, 1);
  s->set_hmirror(s, 1);
#endif

  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");

  startCameraServer();

  Serial.print("Camera Ready! Use 'http://");
  Serial.print(WiFi.localIP());
  Serial.println("' to connect");
}

void loop() {
  // put your main code here, to run repeatedly:
  delay(10000);
}
```

Add procedure files and API interface files related to ESP32 camera.

```
#include "esp_camera.h"
#include <WiFi.h>

//
// WARNING!!! PSRAM IC required for UXGA resolution and high JPEG quality
//            Ensure ESP32 Wrover Module or other board with PSRAM is selected
//            Partial images will be transmitted if image exceeds buffer size
//

// Select camera model
#define CAMERA_MODEL_WROVER_KIT // Has PSRAM
//#define CAMERA_MODEL_ESP_EYE // Has PSRAM
//#define CAMERA_MODEL_M5STACK_PSRAM // Has PSRAM
//#define CAMERA_MODEL_M5STACK_V2_PSRAM // M5Camera version B Has PSRAM
//#define CAMERA_MODEL_M5STACK_WIDE // Has PSRAM
//#define CAMERA_MODEL_M5STACK_ESP32CAM // No PSRAM
//#define CAMERA_MODEL_M5STACK_UNITCAM // No PSRAM
```

```
//#define CAMERA_MODEL_AI_THINKER // Has PSRAM
//#define CAMERA_MODEL_TTGO_T_JOURNAL // No PSRAM

#include "camera_pins.h"
```

Enter the name and password of the router

```
const char *ssid_Router     = "********";  //input your wifi name
const char *password_Router = "********";  //input your wifi passwords
```

Initialize serial port, set baud rate to 115200; open the debug and output function of the serial.

```
  Serial.begin(115200);
  Serial.setDebugOutput(true);
  Serial.println();
```

Configure parameters including interface pins of the camera. Note: It is generally not recommended to change them.

```
  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sccb_sda = SIOD_GPIO_NUM;
  config.pin_sccb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.pixel_format = PIXFORMAT_JPEG;
```

ESP32 connects to the router and prints a successful connection prompt. If it has not been successfully connected, press the reset key on the ESP32-WROVER.

```
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
```

Open the video streams server function of the camera and print its IP address via serial port.

```
102   startCameraServer();
103
104   Serial.print("Camera Ready!  Use 'http://");
105   Serial.print(WiFi.localIP());
106   Serial.println("' to connect");
```

Configure the display image information of the camera.

The set_vflip() function sets whether the image is flipped 180°, with 0 for no flip and 1 for flip 180°.

The set_brightness() function sets the brightness of the image, with values ranging from -2 to 2.

The set_saturation() function sets the color saturation of the image, with values ranging from -2 to 2.

```
79   sensor_t * s = esp_camera_sensor_get();
80   // initial sensors are flipped vertically and colors are a bit saturated
81   if (s->id.PID == OV3660_PID) {
82     s->set_vflip(s, 1); // flip it back
83     s->set_brightness(s, 1); // up the brightness just a bit
84     s->set_saturation(s, -2); // lower the saturation
85   }
```

Modify the resolution and sharpness of the images captured by the camera. The sharpness ranges from 10 to 63, and the smaller the number, the sharper the picture. The larger the number, the blurrier the picture. Please refer to the table below.

```
58   config.frame_size = FRAMESIZE_UXGA;
59   config.jpeg_quality = 10;
60   config.fb_count = 2;
```

Reference

Image resolution	Sharpness	Image resolution	Sharpness
FRAMESIZE_QQVGA	160x120	FRAMESIZE_VGA	640x480
FRAMESIZE_QQVGA2	128x160	FRAMESIZE_SVGA	800x600
FRAMESIZE_QCIF	176x144	FRAMESIZE_XGA	1024x768
FRAMESIZE_HQVGA	240x176	FRAMESIZE_SXGA	1280x1024
FRAMESIZE_QVGA	320x240	FRAMESIZE_UXGA	1600x1200
FRAMESIZE_CIF	400x296	FRAMESIZE_QXGA	2048x1536

Chapter 15 Dog

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

Environment Installation

Install python3

Download and install Python3 package.

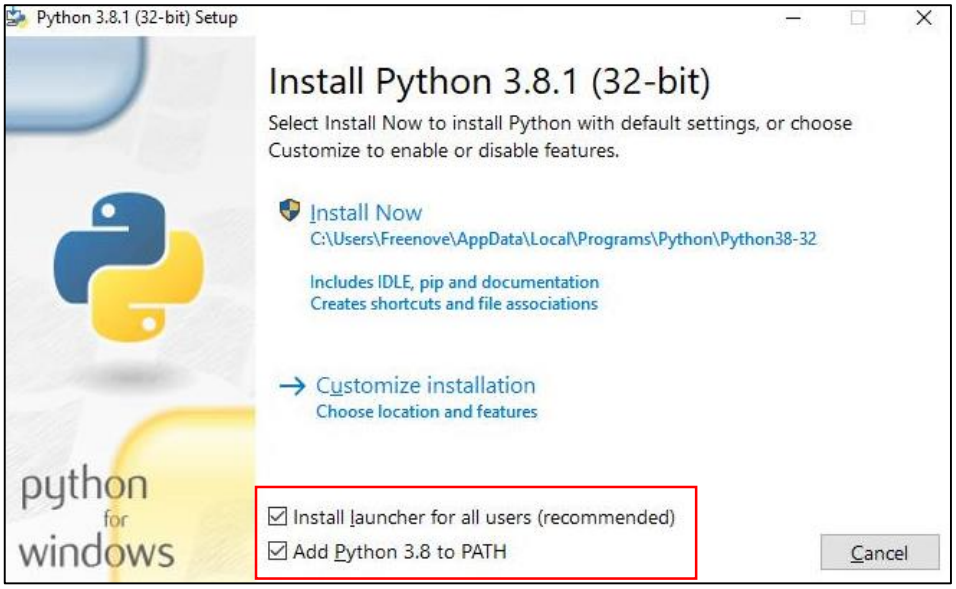
https://www.python.org/downloads/release/python-381/

Files

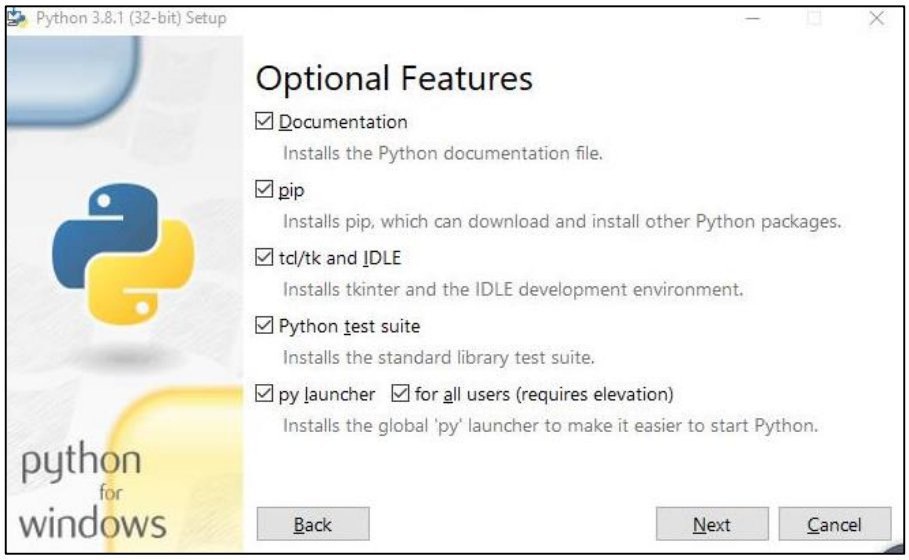
Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		f215fa2f55a78de739c1787ec56b2bcd	23978360	SIG
XZ compressed source tarball	Source release		b3fb85fd479c0bf950c626ef80cacb57	17828408	SIG
macOS 64-bit installer	macOS	for OS X 10.9 and later	d1b09665312b6b1f4e11b03b6a4510a3	29051411	SIG
Windows help file	Windows		f6bbf64cc36f1de38fbf61f625ea6cf2	8480993	SIG
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64	4d091857a2153d9406bb5c522b211061	8013540	SIG
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64	3e4c42f5ff8fcdbe6a828c912b7afdb1	27543360	SIG
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64	662961733cc947839a73302789df6145	1363800	SIG
Windows x86 embeddable zip file	Windows		980d5745a7e525be5abf4b443a00f734	7143308	SIG
Windows x86 executable installer	Windows		2d4c7de97d6fcd8231fc3decbf8abf79	26446128	SIG
Windows x86 web-based installer	Windows		d21706bdac544e7a968e32bbb0520f51	1325432	SIG

Choose the one matching to your computer version to download and install.

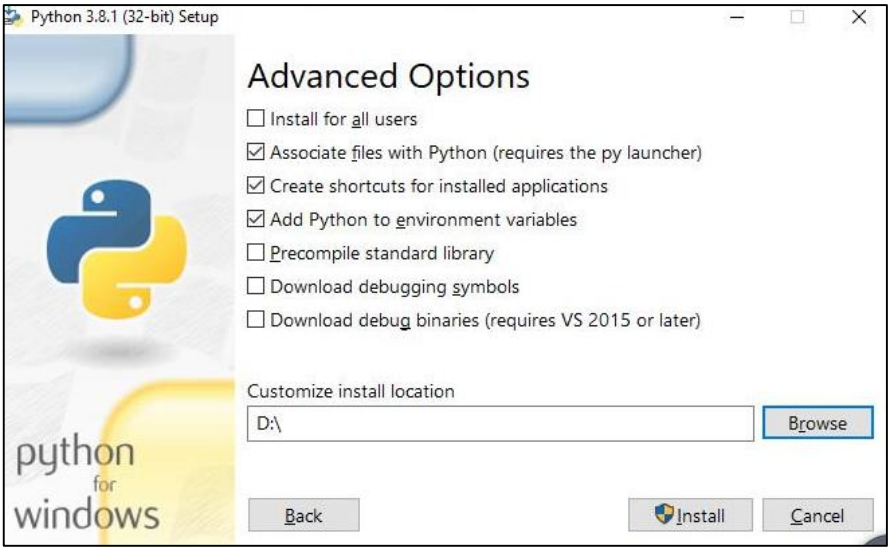
Please note that "Add Python 3.8. to PATH" MUST be check.



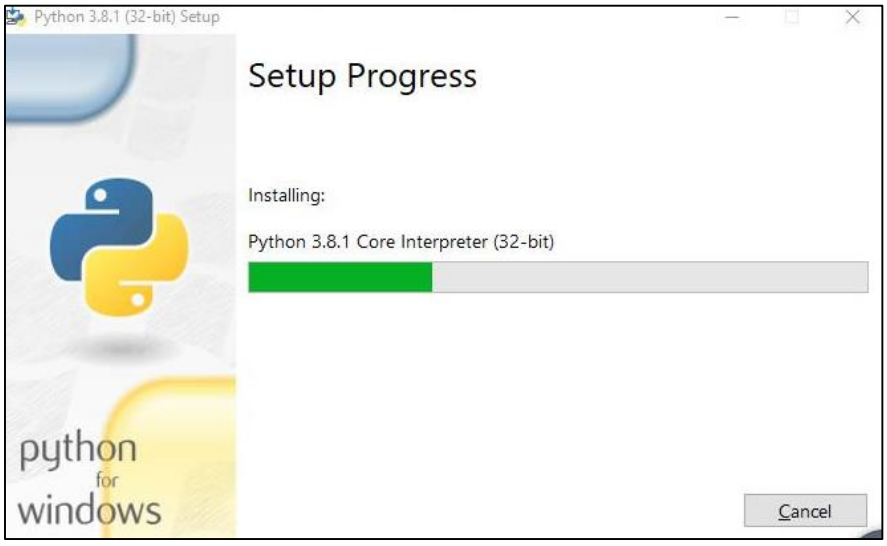
Check all the options and then click "Next".



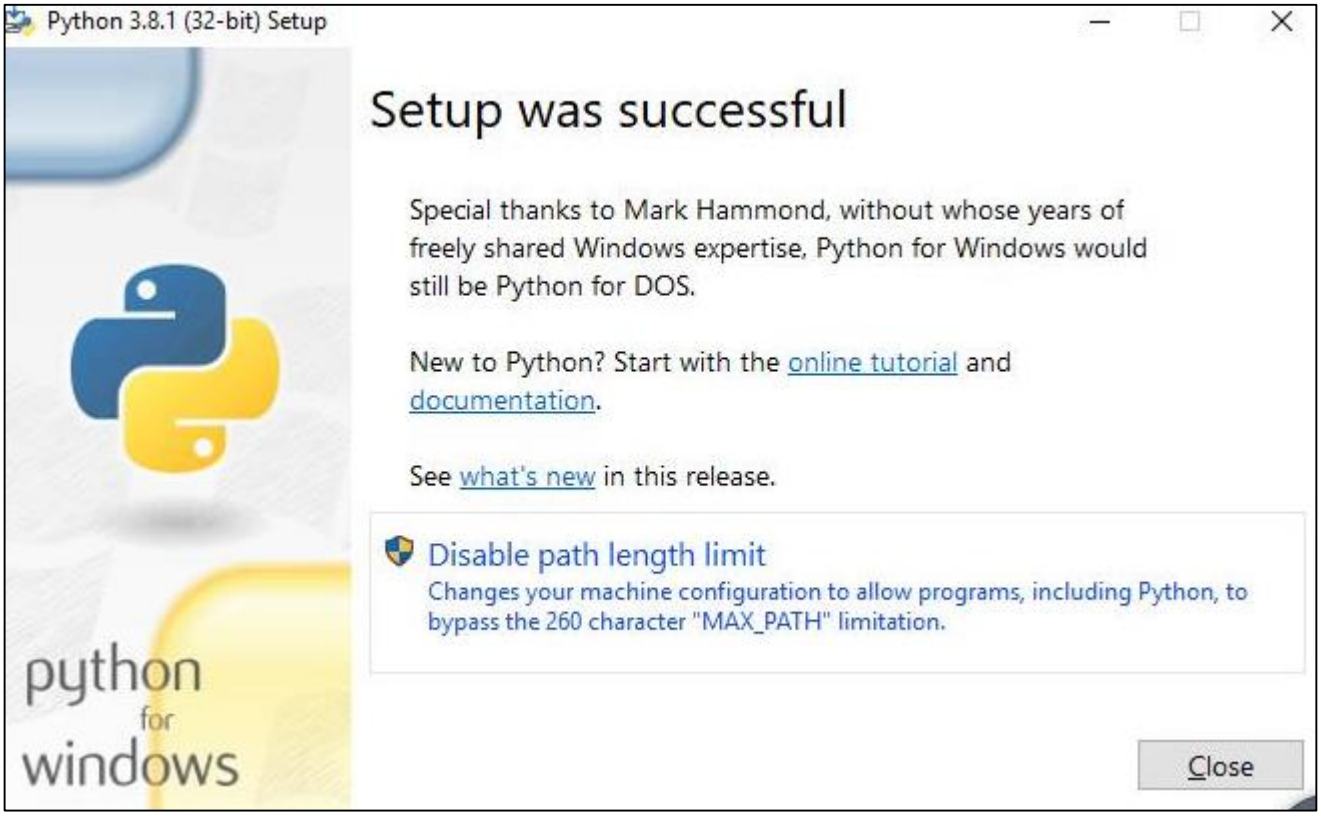
Here you can select the installation path of Python. We install it at D drive. If you are a novice, you can select the default path.



Wait for it to finish installing.



Now the installation is finished.



Install Freenove Board

You can download it directly by clicking the link below, and unzip it manually.

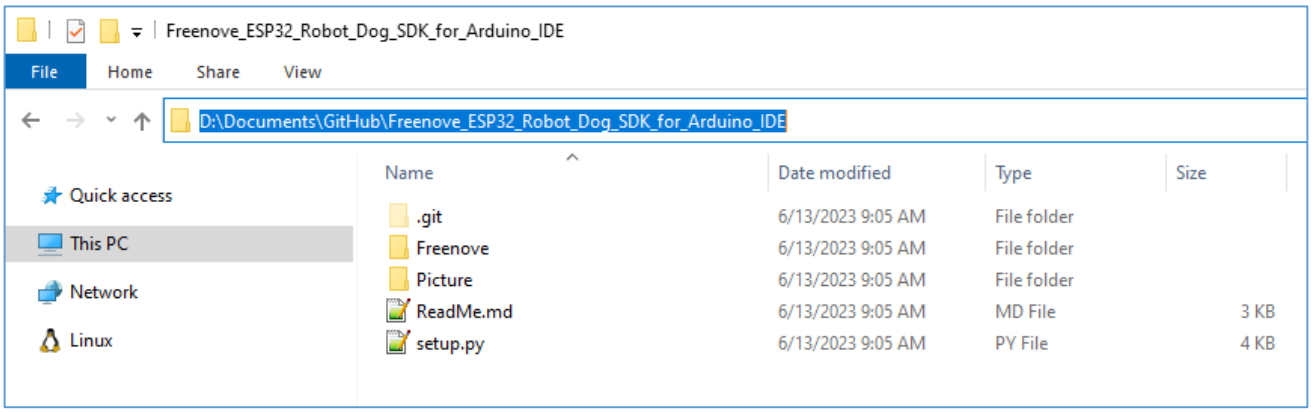
	https://github.com/Freenove/Freenove_ESP32_Robot_Dog_SDK_for_Arduino_IDE/archive/master.zip

Or download via the command

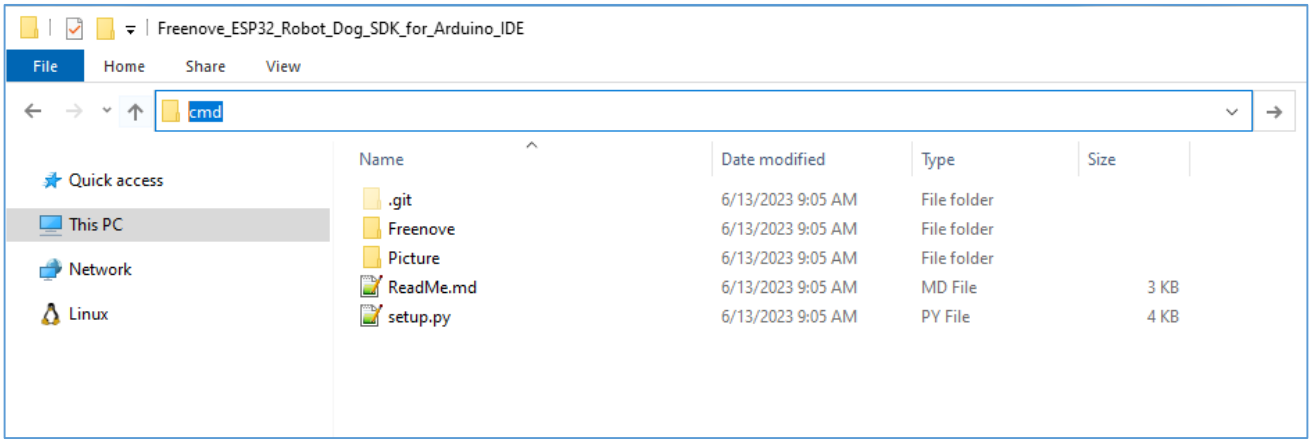
	git clone https://github.com/Freenove/ Freenove_ESP32_Robot_Dog_SDK_for_Arduino_IDE.git

Here we take window installation as an example. Same steps for Linux and Mac.

Find the unzipped file directory, as shown in the figure below.

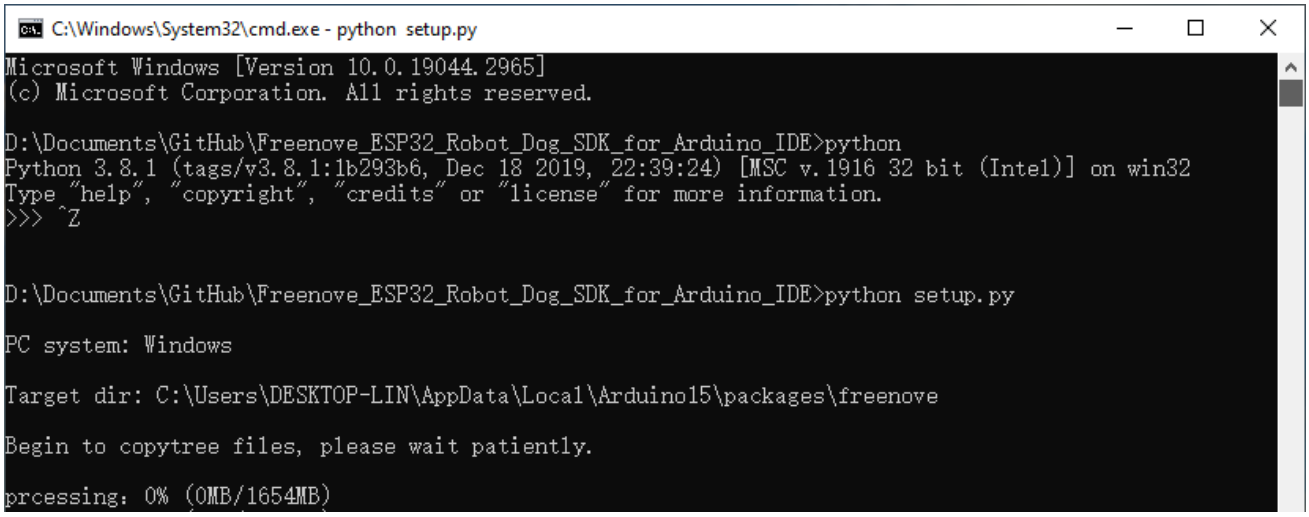


Type **cmd** in the address bar and press Enter. If you are on a Linux or Mac computer, you can use the terminal to enter the file directory.

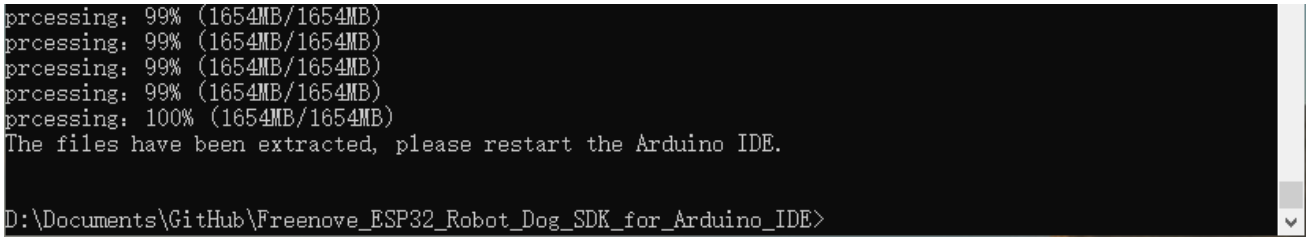


Type **python setup.py** in the cmd.

Wait for it to finish installation.

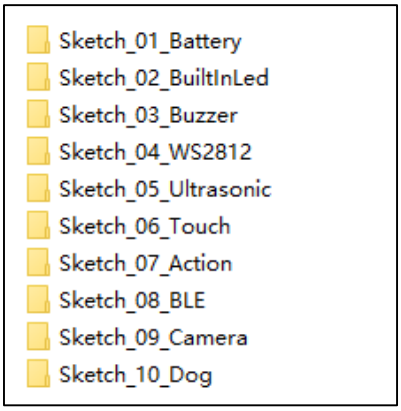


Restart Arduino IDE as per the prompt.

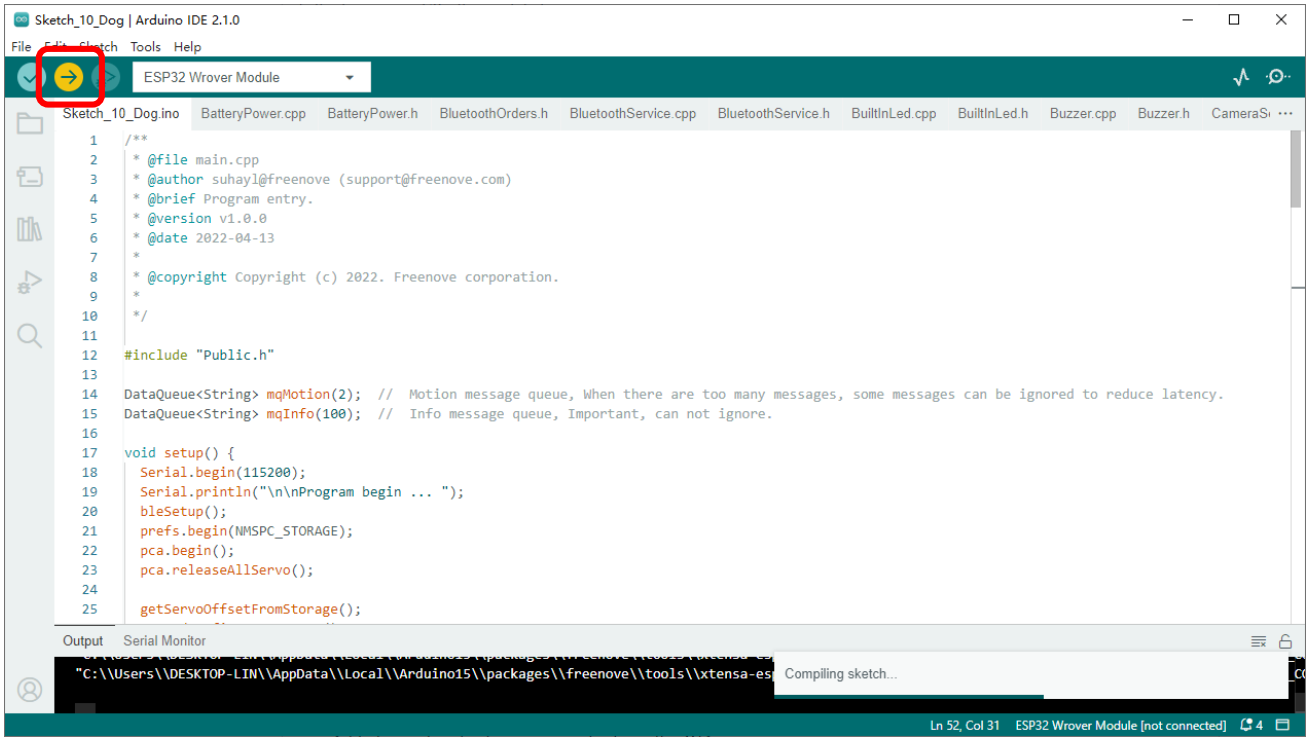


Sketch

Open "Sketch_10_Dog" folder in **"Freenove_Robot_Dog_Kit_for_ESP32\Sketches"** and then double-click "Sketch_10_Dog.ino".

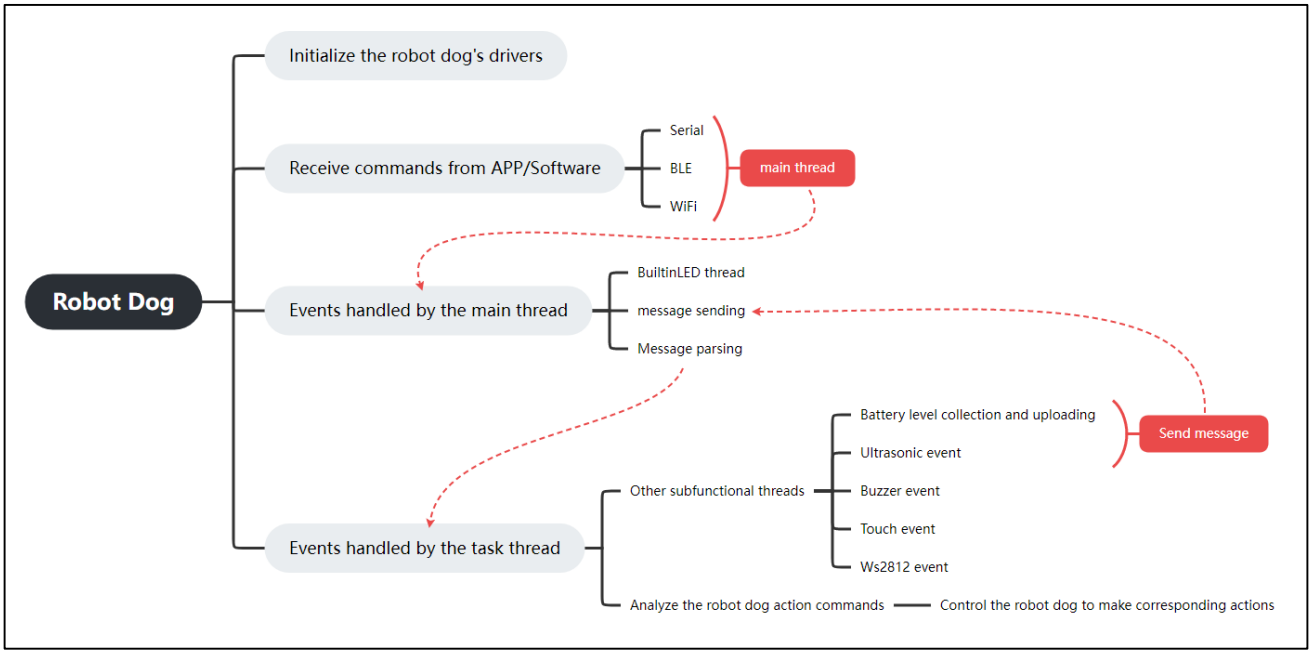


Sketch_10_Dog



Upload the sketch to esp32 and you can control the robot dog to make any action as shown in Chapter 3.

Please note: If your robot dog has not been calibrated, please navigate back to Chapter 2 to calibrate the robot dog first.



The following is the code:

```
#include "Public.h"

DataQueue<String> mqMotion(2);  //  Motion message queue, When there are too many messages, some messages can be ignored to reduce latency.
DataQueue<String> mqInfo(100);  //  Info message queue, Important, can not ignore.

void setup() {
  Serial.begin(115200);
  Serial.println("\n\nProgram begin ... ");
  bleSetup();
  prefs.begin(NMSPC_STORAGE);
  pca.begin();
  pca.releaseAllServo();
  getServoOffsetFromStorage();
  getLedConfigFromStorage();
  cs.begin();
  setupAdc();
  setupBuzzer();
  setupSonar();
  setupRGBLED();
  setupBuiltInLed();
  setupTouchPad();
  Serial.println("setup finished!\n\n");

  startTask(TASK_MOTION_SERVICE);
  startTask(TASK_SECONDRAY);  // loop2

```

```
27    if (!cs.isCameraNormal) {
28      setMelodyToQueue(MELODY_CAM_FAILURE);
29    }
30    setMelodyToQueue(MELODY_POWER_UP);
31    standUp();
32  }
33
34  void loop() {
35    task_CommandService(NULL);
36    task_BleUploadService(NULL);
37    task_showBuiltInLed(NULL);
38    vTaskDelay(20);
39    static uint32_t lastT = 0;
40    if (millis() - lastT > 1000) {
41      // Serial.printf("Total heap: %d\r\n", ESP.getHeapSize());
42      // Serial.printf("Free heap: %d\r\n", ESP.getFreeHeap());
43      lastT = millis();
44    }
45  }
46
47  void loopSecondary(void *pvParameters) {
48    while (1) {
49      task_BatteryPowerListener(NULL);
50      task_showRGBLeds(NULL);
51      task_AutoWalking(NULL);
52      task_BuzzerService(NULL);
53
54      task_TouchPad(NULL);
55      vTaskDelay(50);
56    }
57    vTaskDelete(xTaskGetCurrentTaskHandle());
58  }
59
60  void serialEventRun() {
61    static String serialInputString = "";
62    while (Serial.available()) {
63      char inChar = (char)Serial.read();
64      serialInputString += inChar;
65      if (inChar == '\n') {
66        enterMessageQueue(serialInputString);
67        serialInputString = "";
68      }
69    }
70  }
```

```
71
72  void onBleReceived(BLECharacteristic *pCharacteristic) {
73    std::string rxValue = pCharacteristic->getValue();
74    static String bleInputString = "";
75    if (rxValue.length() > 0) {
76      for (int i = 0; i < rxValue.length(); i++) {
77        bleInputString += rxValue[i];
78        if (rxValue[i] == '\n') {
79          enterMessageQueue(bleInputString);
80          bleInputString = "";
81        }
82      }
83    }
84  }
85
86  void onWiFiCmdReceived(WiFiClient *client) {
87    static String wifiInputString = "";
88    while (client->available()) {
89      char rv[1024];
90      int ret = client->read((uint8_t *)rv, sizeof(rv));
91      for (int i = 0; i < ret; i++) {
92        wifiInputString += rv[i];
93        if (rv[i] == '\n') {
94          Serial.print(wifiInputString);
95          enterMessageQueue(wifiInputString);
96          wifiInputString = "";
97        }
98      }
99    }
100 }
101
102 void onWiFiCmdTrasmit(WiFiClient *client) {
103   if (!mqTx.isEmpty()) {
104     client->write(mqTx.out().c_str());
105   }
106 }
107
108 void enterMessageQueue(String msg) {
109   Serial.print("msg : ");
110   Serial.print(msg);
111   switch (msg.charAt(0)) {
112     case ACTION_INSTALLATION:
113     case ACTION_CALIBRATE:
114     case ACTION_UP_DOWN:
```

```
    case ACTION_BODY_HEIGHT:
    case ACTION_MOVE_ANY:
    case ACTION_TWIST:
    case ACTION_DANCING:
      mqMotion.enterForced(msg);
      controlTask(TASK_MOTION_SERVICE, TASK_RESUME);
      break;
    default:
      mqInfo.enterForced(msg);
      break;
  }
}
```

Message queue. Each time instructions are received, they will be stored in a queue and waiting for the main thread to parse and execute it one by one.

```
DataQueue<String> mqMotion(2);  //  Motion message queue, When there are too many messages, some messages can be ignored to reduce latency.
DataQueue<String> mqInfo(100);  //  Info message queue, Important, can not ignore.
```

Initialize the robot dog's drivers.

```
  Serial.begin(115200);
  Serial.println("\n\nProgram begin ... ");
  bleSetup();
  prefs.begin(NMSPC_STORAGE);
  pca.begin();
  pca.releaseAllServo();
  getServoOffsetFromStorage();
  getLedConfigFromStorage();
  cs.begin();
  setupAdc();
  setupBuzzer();
  setupSonar();
  setupRGBLED();
  setupBuiltInLed();
  setupTouchPad();
  Serial.println("setup finished!\n\n");
```

Receive commands by Serial.

```
void serialEventRun() {
  static String serialInputString = "";
  while (Serial.available()) {
    char inChar = (char)Serial.read();
    serialInputString += inChar;
    if (inChar == '\n') {
      enterMessageQueue(serialInputString);
      serialInputString = "";
    }
```

```
  }
}
```

Receive commands by BLE.

```
void onBleReceived(BLECharacteristic *pCharacteristic) {
  std::string rxValue = pCharacteristic->getValue();
  static String bleInputString = "";
  if (rxValue.length() > 0) {
    for (int i = 0; i < rxValue.length(); i++) {
      bleInputString += rxValue[i];
      if (rxValue[i] == '\n') {
        enterMessageQueue(bleInputString);
        bleInputString = "";
      }
    }
  }
}
```

Receive commands by WiFi.

```
void onWiFiCmdReceived(WiFiClient *client) {
  static String wifiInputString = "";
  while (client->available()) {
    char rv[1024];
    int ret = client->read((uint8_t *)rv, sizeof(rv));
    for (int i = 0; i < ret; i++) {
      wifiInputString += rv[i];
      if (rv[i] == '\n') {
        Serial.print(wifiInputString);
        enterMessageQueue(wifiInputString);
        wifiInputString = "";
      }
    }
  }
}
```

Divide the commands into robot dog action commands and other commands according to the types and send them into the corresponding message queue.

```
void enterMessageQueue(String msg) {
  Serial.print("msg : ");
  Serial.print(msg);
  switch (msg.charAt(0)) {
    case ACTION_INSTALLATION:
    case ACTION_CALIBRATE:
    case ACTION_UP_DOWN:
    case ACTION_BODY_HEIGHT:
    case ACTION_MOVE_ANY:
    case ACTION_TWIST:
```

```
    case ACTION_DANCING:
      mqMotion.enterForced(msg);
      controlTask(TASK_MOTION_SERVICE, TASK_RESUME);
      break;
    default:
      mqInfo.enterForced(msg);
      break;
  }
}
```

Events handled by the main thread.

```
void loop() {
  task_CommandService(NULL);
  task_BleUploadService(NULL);
  task_showBuiltInLed(NULL);
  vTaskDelay(20);
  static uint32_t lastT = 0;
  if (millis() - lastT > 1000) {
    // Serial.printf("Total heap: %d\r\n", ESP.getHeapSize());
    // Serial.printf("Free heap: %d\r\n", ESP.getFreeHeap());
    lastT = millis();
  }
}
```

Events handled by the task thread.

```
void loopSecondary(void *pvParameters) {
  while (1) {
    task_BatteryPowerListener(NULL);
    task_showRGBLeds(NULL);
    task_AutoWalking(NULL);
    task_BuzzerService(NULL);

    task_TouchPad(NULL);
    vTaskDelay(50);
  }
  vTaskDelete(xTaskGetCurrentTaskHandle());
}
```

Development

Code Repository

The main purpose of this tutorial is to explain how to assemble and use the robot dog. If you want to learn more about the code, please visit:
https://github.com/Freenove/Freenove_ESP32_Dog_Firmware
There is a more detailed explanation of the code in the code repository, please download and learn by yourself.

Communication Command

If you do not want to understand the underlying code of the robot dog, but want to develop a control platform software for the robot dog, you can refer to the following communication commands, which explain each control command of the robot dog in detail.

Ways of Communication

The robot support three ways of communication, namely, serial port, Bluetooth Low Energy (BLE) and TCP Socket communication, as follows:

Ways	Description
Serial Port	Baud rate 115200
BLE	Bluetooth GATT
TCP Socket	Port 5000 is the command transmission port, and port 8000 is the video transmission port.

Note: All commands are valid once they are sent. It is recommended not to send too many commands repeatedly in a short period as the processing capacity of the robot dog is limited. All commands only need to be sent once.

Communication Command Format

1. Each command consists of four parts: command word, delimiter, parameter and terminator, among which, the number of command word is one and the parameters is variable, from 0 to n, depending on the specific command.
   A. The first character of each command is the command word, used to distinguish the major category of the command, such as "A".
   B. The character "#" is the delimiter between the command word and the parameter, used to separate the string.
   C. Each command is terminated with "\n", which is used to separate each command.
   Example: “A#10#20#30#40#50#\n”
2. Parse of Commands

When parsing commands, first separate the commands with "\n", and then separate the command word and parameters of each command with“#”. The characters after "\n" are divided to the next command for parsing.

Command Words

We have defined the following 18 command words.

```
#define ACTION_UP_DOWN                  'A'
#define ACTION_BODY_HEIGHT              'B'
#define ACTION_RGB                      'C'
#define ACTION_BUZZER                   'D'
#define ACTION_TWIST                    'E'
#define ACTION_MOVE_ANY                 'F'
#define ACTION_CAMERA                   'G'
#define ACTION_ULTRASONIC               'H'
#define ACTION_GET_VOLTAGE              'I'
#define ACTION_CALIBRATE                'J'
#define ACTION_SET_NVS                  'K'
#define ACTION_INSTALLATION             'L'
#define ACTION_AUTO_WALKING             'M'
#define ACTION_NETWORK                  'N'
#define ACTION_DANCING                  'O'
#define ACTION_SET_ROBOT                'R'
#define ACTION_TEST                     'T'
#define ID_CHECK                        'W'
```

Explanation of Communication Protocol Instructions

ACTION_UP_DOWN

Control the robot to stand up or lie down with one parameter. When lying down, the servos will be unloaded to reduce battery power consumption

App command	Action
A#0#\n	Switch between standing up and lying down
A#1#\n	Stand up
A#2#\n	Lie down

No return,

ACTION_BODY_HEIGHT

Set the robot dog's body to any height. This function has not been added to the app yet, but the robot dog can receive this command.

App command	Action
B#50#\n	Make the robot stand up and set the height to 50 cm.

ACTION_RGB

Set the running mode and color of the LEDs on the robot dog.

App command	Action
C#mode#red#green#blue#\n	Set the RGB LEDs to “mode” mode and specify the color values of the three channels（0-255）
C#1#255#0#0#\n	

The value range and meaning of mode, among which, in modes 2 and 5, the color cannot be changed. The parameters can be any value in the range of 0-255.	`#define LED_MODE_OFF 0`
	`#define LED_MODE_RGB 1`
	`#define LED_MODE_FOLLOWING 2`
	`#define LED_MODE_BLINK 3`
	`#define LED_MODE_BREATHING 4`
	`#define LED_MODE_RAINBOW 5`

Introduction to “mode” function

Macro Definition	Parameter	Description
LED_MODE_OFF	0	Turn OFF LEDs
LED_MODE_RGB	1	Static LEDs display. Color and brightness can be adjusted through the parameters.
LED_MODE_FOLLOWING	2	The LEDs alternately display four colors of red, green, yellow and blue.
LED_MODE_BLINK	3	LEDs blink. Color and brightness can be adjusted through the parameters.
LED_MODE_BREATHING	4	LEDs ON and OFF like breathing. Color and brightness can be adjusted through the parameters.
LED_MODE_RAINBOW	5	LEDs show the color of rainbow and change slowly.

ACTION_BUZZER

Set the frequency of the buzzer.

App command	Action
D#freq#\n	“freq” refers to the frequency of the buzzer.
D#2000#\n	Make the buzzer sound at the frequency of 2000
D#0#\n	Stop the buzzer.

ACTION_TWIST

Make the robot twist itself.

App command	Action
E#pitch#roll#yaw#\n	
E#20#0#0#\n	The robot is tilted by 20 degrees in the pitch direction.
E#10#10#10#\n	The robot is tilted by 10 degrees in three directions.

Unit: degree, ranging from 0 to 20 degrees.

ACTION_MOVE_ANY

F#alpha#stepLength#gama#spd#\n

Make the robot move sideways for alpha degrees counterclockwise in its forward direction while rotating alpha degrees at each step, with the step length of stepLength and the speed of spd.

```
/**
 * @brief The command for walking, in any direction, at any step length and any speed, with any rotation angle.
 *
 * @param alpha The moving direction, with the x-axis direction as 0 degrees, counterclockwise as positive and clockwise as negative. Unit: degree [0-360]. The x-direction means moving forward .
 * @param stepLength The length of each step (<=20)
 * @param gama The rotation angle. Rotate in place with Rotate in place, with counterclockwise as positive and clockwise as negative, and the unit is degrees. [0-360].
 * @param spd The moving speed. Unit: mm / 10ms [1, 8]
 */
```

App command	Action
F#0#20#0#5#\n	Moving forward at the speed of 5 and step length of 20mm.
F#90#10#-10#5#\n	The robot moves sideways to the left while rotating 10 degrees to the right at each step, with a step length of 10 and speed of 5.
F#0#0#0#5#\n	Stop moving

This command also only needs to be sent once. Once commanded, the robot keeps walking until it receives a new command, such as a command to stop walking or to twist its body.

ACTION_CAMERA

This is the command sent to the app to determine whether the camera is malfunctioned. The phone app receives and parses the command.

Robot command	Action

G#100#\n	The camera works fine, not need to process.
G#101#\n	The camera is malfunctioned, prompting via Toast.

If the camera is faulty, the robot will periodically send reminders to the app, and the app will send a prompt to the user after receiving the message.

ACTION_ULTRASONIC

Once the obstacle avoidance mode is turned ON, the robot will actively send the current distance measured by the ultrasonic wave to the app and the app needs to parse this command.

The App can also actively send this command for the robot dog to print out the distance measured by the ultrasonic wave through the serial port. Note that the command does not contain parameters, and there is currently no Action.

App command	Action
H#\n	The robot obtains the distance measured by the ultrasonic module once, and prints it out through serial port.

Robot command	Action
H#dist#\n	"dist" refers to the distance measured by the ultrasonic module
H#101#\n	The current measured distance is 101 cm.

ACTION_GET_VOLTAGE

The robot actively sends the current voltage value and battery percentage to the app every 3 seconds without the need of the app to inquire.

The unit of voltage is mV.

Robot command	Action
I#voltage#percent#\n	"voltage" refers to the current voltage of the batteries, and "percent" is the battery percentage.
I#8400#100%\nI	The current voltage is 8400mV, 100%.

ACTION_CALIBRATE

Calibrate the robot. This command must contain only five parameters.

Parameters	Action
J#legn#SSC#x#y#z#\n	
Legn	Leg number, numbered counterclockwise from left front to right front, 0,1,2,3
S: Select S: Set C: Confirm	1: Set the position of the legs without overlaying calibration information. 2: Confirm the calibration information. 3: Select one leg to calibrate.
x、y、z	They refer to the coordinates of the tiptoe of the robot dog, with the calibration point as the relative value of the origin. If the leg does not need calibration, then it is 0 0 0.

Examples:

App command	Action

J#2#3#0#0#0#\n	Leg 2 is selected to calibrate and the other legs are unloaded to prevent interference.
J#2#1#10#20#-10#\n	Set leg 2 to the relative position of 10 20 -10
J#2#2#10#20#-10#\n	Confirm and save the calibration information of 10 20 -20 to the robot.

ACTION_SET_NVS

Save data. This command is a debug command without the need to be sent by the app.

App command	Action
K#1#\n	Save the mode and color of the LEDs
K#2#\n	Clear all data in NVS.

ACTION_INSTALLATION

Put the robot into an installation or calibration pose.

App command	Action
L#1#\n	Put the robot in the installation pose, and place all servos at 90 degrees
L#2#\n	Put the robot in the calibration pose without overlaying the calibration information.

ACTION_AUTO_WALKING

Put the robot into automatic obstacle avoidance mode.

App command	Action
M#0#\n	Turn OFF obstacle avoidance mode.
M#1#\n	Turn ON obstacle avoidance mode.

ACTION_NETWORK

The robot's network related commands.

App sending

App command	Action
N#0#\n	The robot starts to scan WIFI and returns the results one by one in the following form: N#0#SSID#\n
N#1#ssid#psd#\n	The robot uses the password psd to connect to the WIFI hotspot named SSID and returns the result: Success: N#103#\n Failure: N#104#\n Later, regardless of success or failure, the WIFI status is sent again. (see the robot sending below)
N#2#\n	Disconnect the robot's WIFI connection, and then send the WIFI status.
N#3#\n	Establish a WIFI hotspot on the robot with the SSID as "FreenoveDog-RobotId" and the password as 12345678. Without this it will not success, The following results will be returned: N#301#AP_SSID#IP#Password#\n The above parameters are hotspot name, robot IP address and hotspot password respectively.
N#4#\n	Turn OFF the robot's hotspot.
N#5#\n	Obtain the robot WIFI and camera status and return them once.

Robot sending:

Robot command	Action
N#0#SSID#\n	Return the robot's WIFI scanning results one by one.
N#101#SSID#IP#\n	Robot's WIFI status: WIFI is connected. The parameters are the WIFI SSID and IP address of the robot in STA mode
N#102#\n	Robot's WIFI status: WIFI is not connected.
N#103#\n	The robot connects to WIFI successfully.
N#104#\n	The robot fails to connect to WIFI.
N#301#AP_SSID#IP#Password#\n	The robot's WIFI status: AP has been established. The parameters are the robot's WIFI SSID, IP address and password in AP mode.
N#302#\n	The robot's WIFI hotspot has been turned OFF.
N#5#\n	Obtain the robot WIFI and camera status and return them once.

ACTION_DANCING

Make the robot dance at some fixed actions.

App command	Action
O#n#\n	n refers to the fixed action modes.
O#0#\n	Say hello
O#1#\n	Push up
O#2#\n	Stretch itself
O#3#\n	Turn around
O#4#\n	Sit down
O#5#\n	Dance

The value and meanings of parameter n are as follows

```
#define DANCE_SAY_HELLO         0
#define DANCE_PUSH_UP           1
#define DANCE_STRETCH_SELF      2
#define DANCE_TURN_AROUND       3
#define DANCE_SIT_DOWN          4
#define DANCE_DANCING           5
```

ACTION_SET_ROBOT

Reserved.

ACTION_TEST

Reserved. Develop to use.

ID_CHECK

Check the basic information of the robot, which is used for verification in the future.

App command	Action
W#0#\n	Obtain the firmware version and robot name. The result returned are as follows: W#0#100#FREENOVE-DOG#\n among which, 100 refers to the version of V1.0.0 and the name of FREENOVE-DOG.
W#1#\n	Obtain the robot's firmware version with the result returned as follows: W#1#100#\n among which, 100 refers to the version of V1.0.0
W#2#\n	Obtain the robot's name with the result returned as follows: W#2#FREENOVE-DOG#\n referring to the name as FREENOVE-DOG
W#3#\n	Obtain the internal code with the current result returned as follows W#3#FNK006201#\n
W#4#FREENOVE#\n	Fixed command to check whether the source of the controller is valid.

What's Next?

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, 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.
http://www.freenove.com/

Thank you again for choosing Freenove products.