Welcome

Thank you for choosing Freenove products!

About Battery

First, read the document **About_Battery.pdf** in the unzipped folder.
If you did not download the zip file, please download it and unzip it via link below.
https://github.com/Freenove/Freenove_Robot_Arm_Kit_for_Raspberry_Pi/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

Contents

Welcome 1
List 3
Calibration Graph 3
Sheet Metal Assembly Components 4
Machinery Parts 5
Acrylic Parts 6
Rubber Parts 6
Electronic Parts 7
Cables 8
Tools 8
Required but NOT Contained Parts 9
Preface 10
Introduction to Raspberry Pi 11
Chapter 0 Preparation 21
Install a System 21
Remote desktop & VNC 30
Chapter 1 Install Python Libraries (Required) 40
Chapter 1 Function Tests 44
Robot Arm Board for Raspberry Pi 44
Stepper Motor Test 47
Infrared Sensor Test 60
RGB LED Module Test 63
Buzzer Test 68
Servo Test 70
Chapter 2 Assembly of the Robot Arm 74
Numbering of the Assembly Components 74
Step 1 Mounting the Timing Pulleys to the Stepper Motor 76
Step 2 Mounting the Stepper Motors to the No.1 Assembly Part 77
Step 3 Mounting the Infrared Sensors 80
Step 4: Installing the Bottom Transmission Device 82
Step 5 Installing the Timing Belt 85
Step 6 Assembling the Base 87
Step 7 Mounting Assembly Components No.8 and No.17 89
Step 8 Mounting Assembly Components No.7 and No.9 90
Step 9: Assemble the Left and Right Transmission Devices 91
Step 10 Mounting Assembly Components No.10 and No.12 95
Step 11 Mounting Assembly Components No.11, No.4 and No.18 98
Step 12 Mounting Assembly Component No.20 102
Step 13 Mounting Raspberry Pi to Robot Arm Board 103
Step 14 Installing the Battery Holder 107
Step 15 Mouting Assembly Components No. 14 109
Step 16 Assembling the Pen Clip 111

Step 17 Installing the Pen Clip....115
Step 18 Assembling the Servo Clamp....117
Step 19 Installing the Servo Clamp....121
Step 20 Wiring of Stepper Motor and Sensors....123
Step 21 Installing the LED Module....125
Step 22 Adjustment of the Sensor Sensibility....127
Chapter 3 Installation, Launch, and Packaging of Robotic Arm Control Software....130
Libraries Installation....130
Robot Arm Controlling Software....135
Packaging of Robotic Arm Control Software....141
Chapter 4 Usage of the Software....142
Connecting to the Robot Arm....142
Configuring Parameters for Robot Arm....144
Robot Arm Control....160
Instructions Recording Mode....164
Drawing Mode....167
LED Module Control....175
Chapter 5 Introduction to the APP....179
Connecting to the Robot Arm....179
Configuring Parameters for Robot Arm....181
Robot Arm Control....198
Instructions Recording Mode....203
Drawing Mode....206
LED Module Control....214
Chapter 6 Communication Instructions....217
Formats of Communication Instructions....217
Instructions for Movements....217
Customized Instructions....217
Chapter 7 Core Code Introduction....222
Overall Flowchart....222
Code Introduction....223
What's Next?....270

List

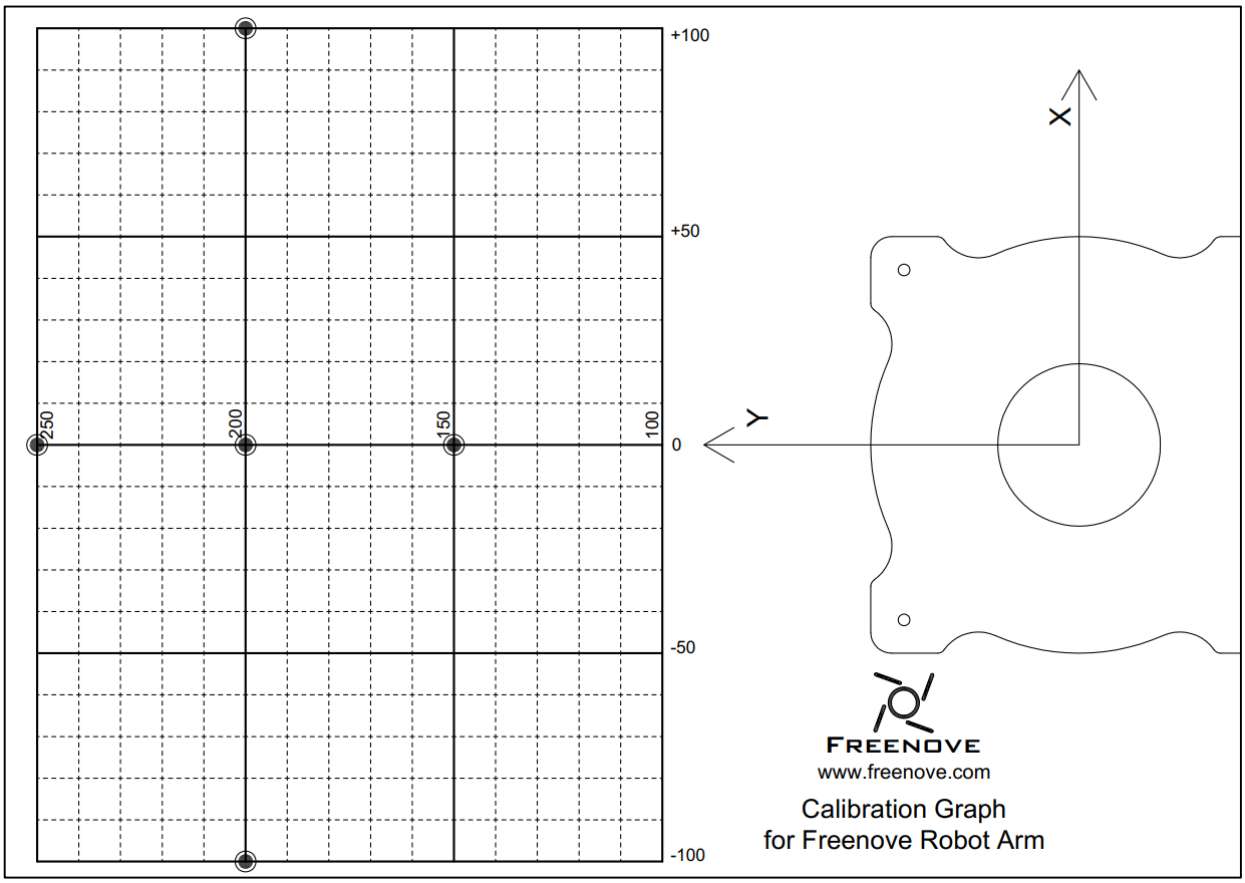
Before you begin, please check the list to make sure the materials are complete. If you find that your robot arm material is incomplete, please email us in time: support@freenove.com

Calibration Graph

Please check whether the calibration graph is included. If it is not there, please find the .pdf file in the material we provide and print it yourself.

Freenove_Robot_Arm_Kit_for_Raspberry_Pi/Calibration graph.pdf

Calibration graph

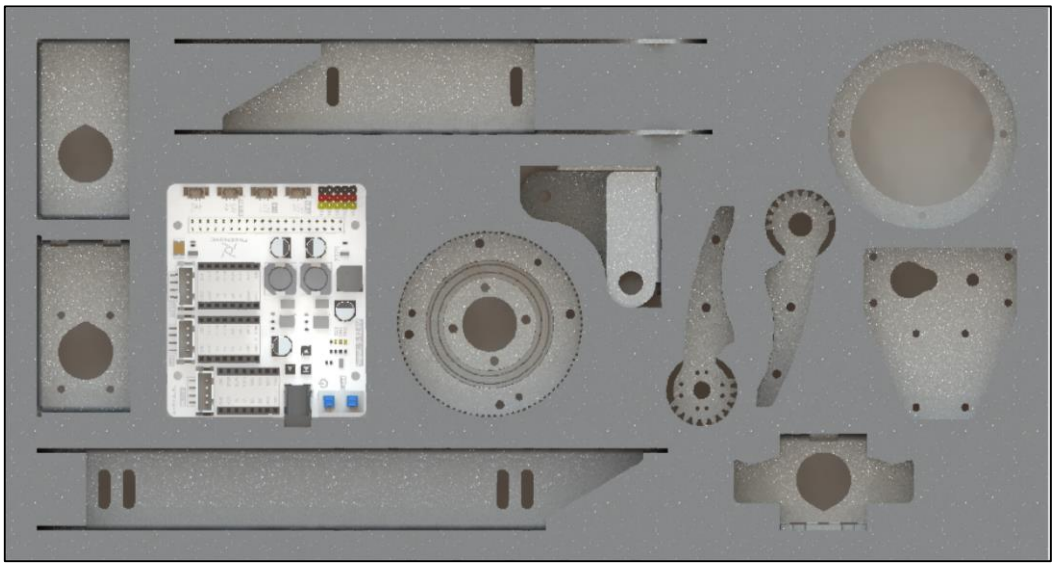


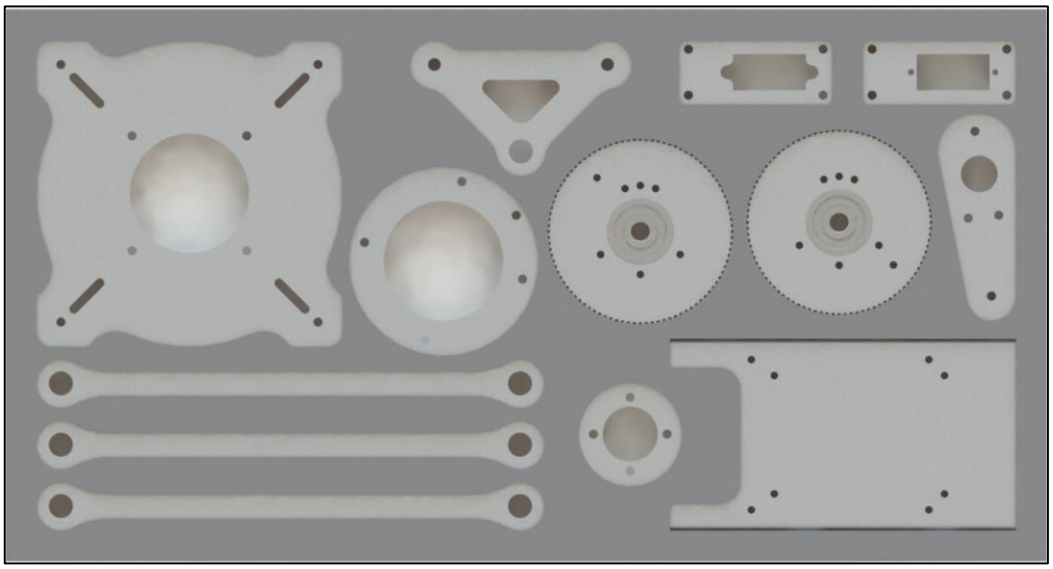
The recommended range of motion for the robot arm is between (-100,150) to (150,250).

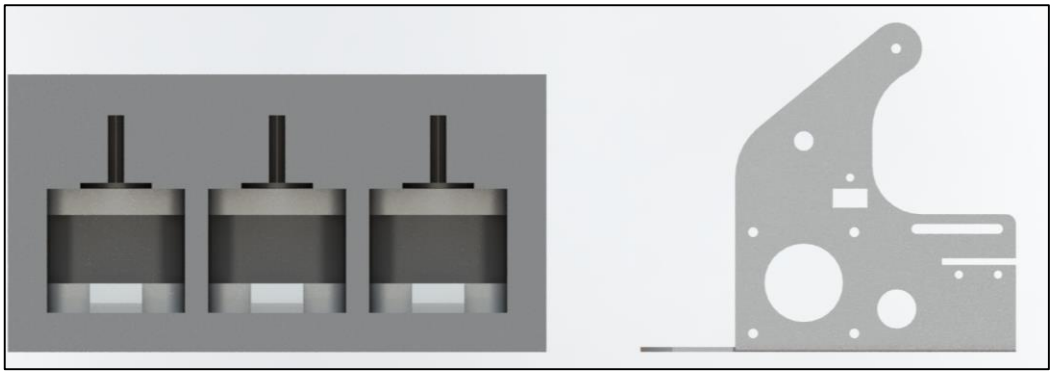
Sheet Metal Assembly Components

Here is the overall diagram of the materials. Please check if there is any piece missing or damaged upon unboxing.

Should any of these happen, please email our support at support@freenove.com.







Machinery Parts

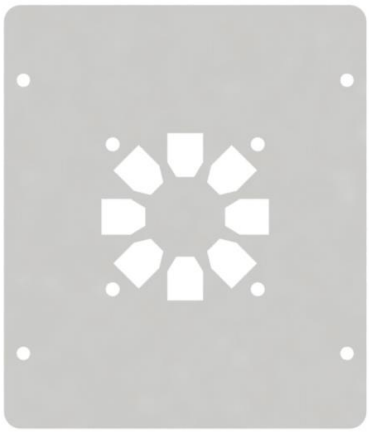
Fasteners

All fasteners come in a large bag, please open it and check whether they are complete.

Other Hardware Materials

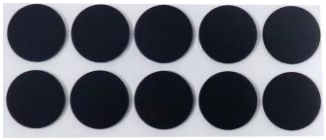
Tension Spring (0.5x5x25)	Spring Plunger	Timing Pulley -2GT-16 Gears	Plane Thrust Bearing (51106-30x47x11)
			

Acrylic Parts

Top Plate 	LED Module Mounting Plate 
Tailboard	

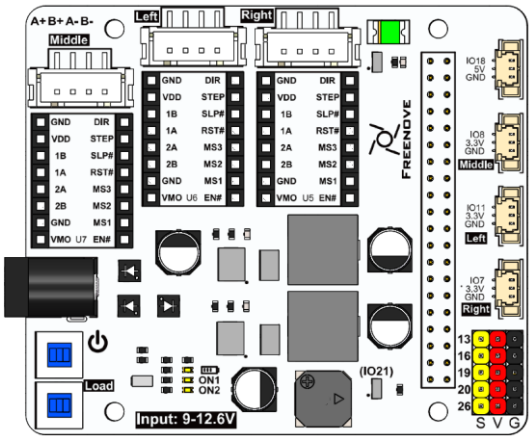
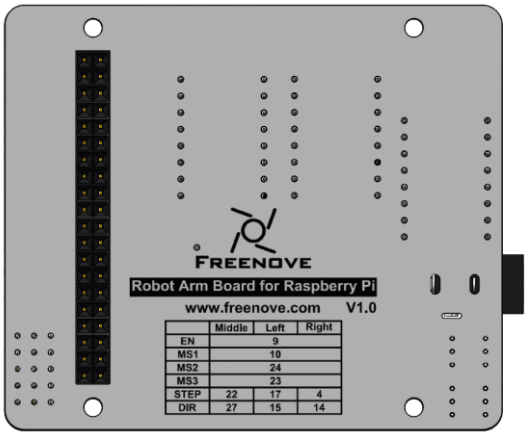
Rubber Parts

Foot Pad x1

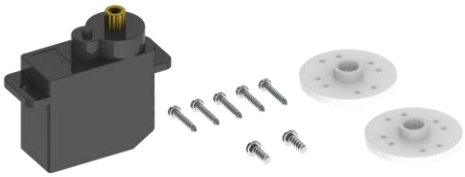
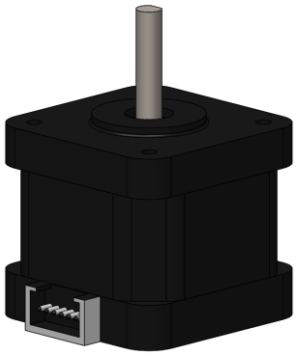


Electronic Parts

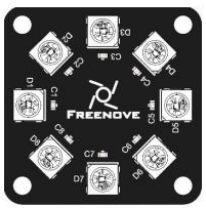
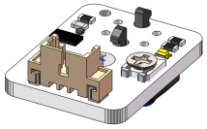
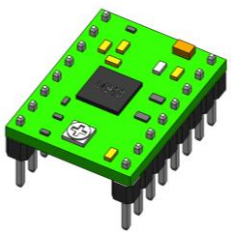
Robot Arm Board for Raspberry Pi

Top	Bottom
	

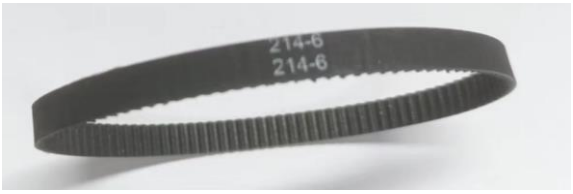
Transmission Parts

Servo Package X1	Stepper Motor X3
	

Electronic Module

LED Module	Battery Holder	Infrared Sensor Module	Stepper Motor Driving Module
			

Cables

20cm 3Pin Cable (Same Direction) 	
Cable For Stepper Motor 	
Power Cord 	
10cm 3Pin Cable To Jumper Wire 	2GT-214 Timing Belt 

Tools

Cross Screwdriver (3mm) X1 	Tweezers 
Allen Key 	5mm Open-End Wrench 
7mm Open-End Wrench 	8mm Open-End Wrench 

Required but NOT Contained Parts

Three 18650 lithium batteries without protected board.

The 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.

Raspberry Pi (Recommended model: Raspberry 4B / 3B+/ 3A+ /3B) x1

Raspberry PI 5 software pwm function is poor, it is recommended to use Raspberry PI 4.

Preface

Welcome to use Freenove Robot Arm Kit for Raspberry Pi. Following this tutorial, you can make a very cool robot arm with many functions.

This kit is based on Raspberry Pi, a popular control panel, so you can share and exchange your experience and design ideas with many enthusiasts all over the world. This kit contains all electronic components, modules, and mechanical components required for making the robot arm, all of which are packaged individually. There are detailed instructions for assembly and configuration 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 can help enthusiasts with little technical knowledge to make a robot arm. If you are very interested in Raspberry Pi, and want to learn how to program and build the circuit, please visit our website www.freenove.com or contact us to buy the kits designed for beginners: https://github.com/Freenove/Freenove_Ultrasonic_Starter_Kit_for_Raspberry_Pi

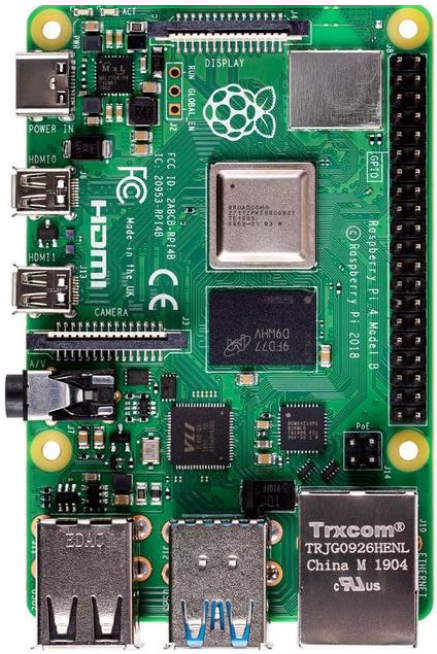
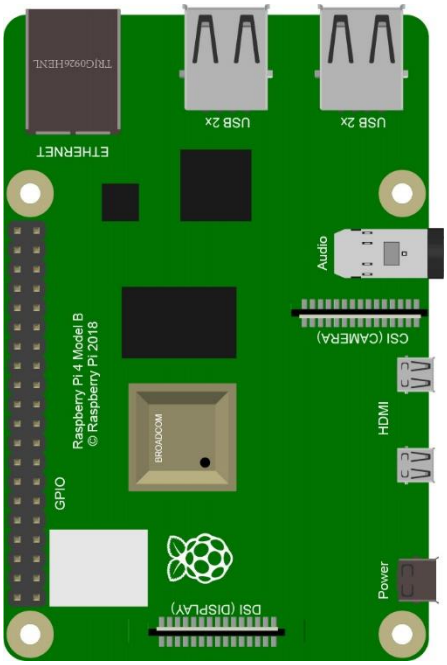
Introduction to Raspberry Pi

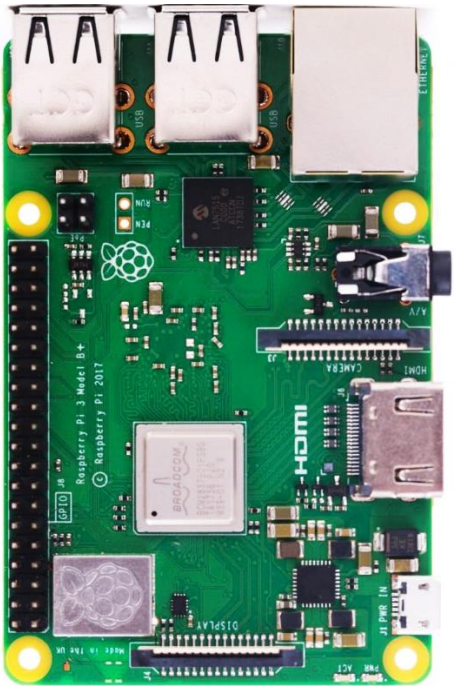
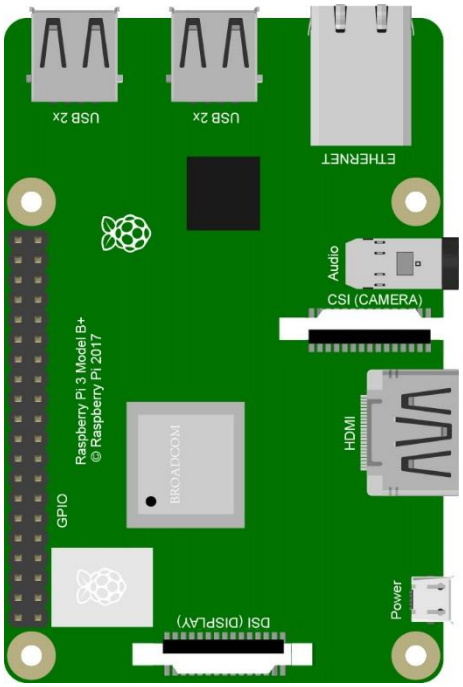
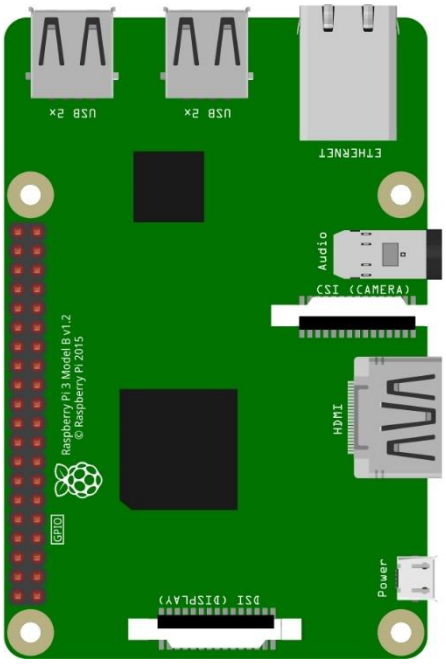
Raspberry Pi (or RPi, RPI, RasPi, which will be also referenced in this tutorial), a micro-computer with size of a card, quickly swept the world since it was launched. It is widely used in desktop workstation, media center, smart household, robots, and even the servers, etc. It can do almost everything, which continues to attract fans to explore it. Raspberry Pi is used to be running with Linux system and along with the release of windows 10 IoT, we can also run it with Windows. Raspberry Pi (with interfaces USB, network, HDMI, camera, audio, display and GPIO), as a microcomputer, can be run in command line mode and desktop system mode. Additionally, it is easy to operate just like Arduino, and you can even directly operate the GPIO of CPU.

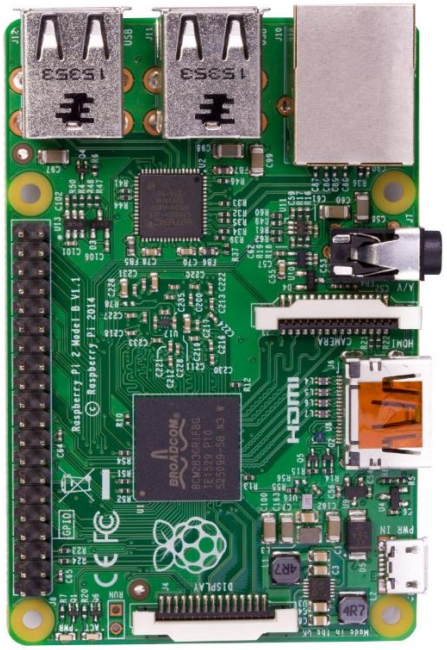
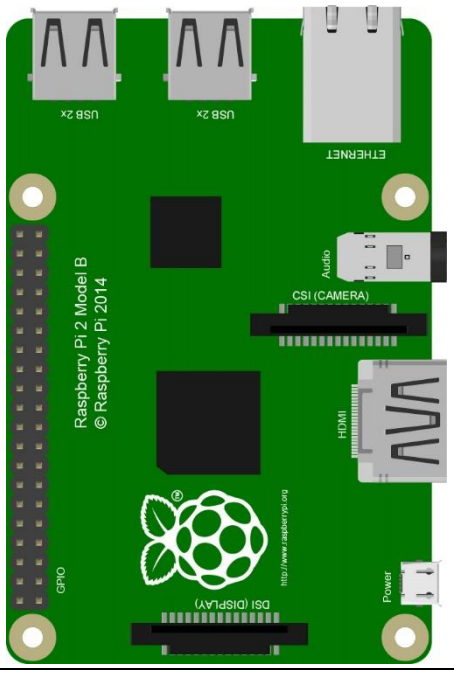
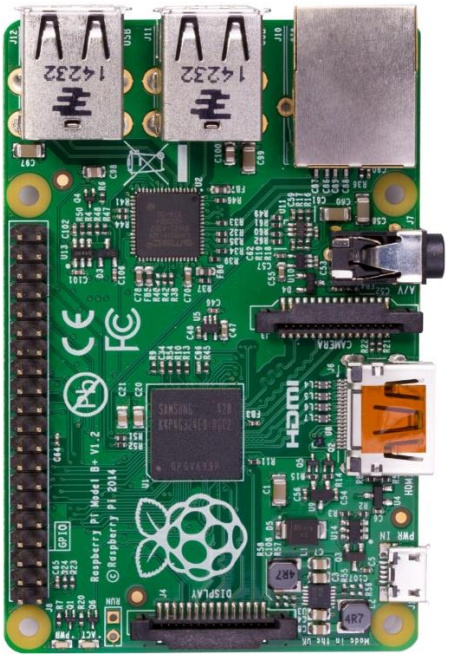
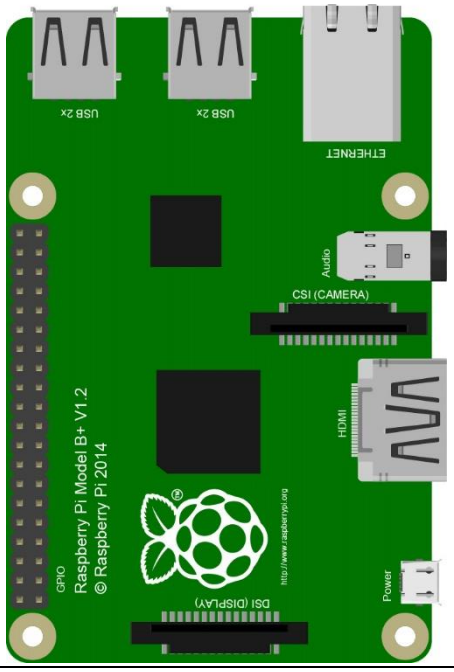
So far, at this writing, Raspberry Pi has advanced to its fifth generation product offering. Version changes are accompanied by increases in upgrades in hardware and capabilities. However, since the Raspberry Pi 5 has just been released, we are still using the Raspberry Pi 5 as the master system to learn.

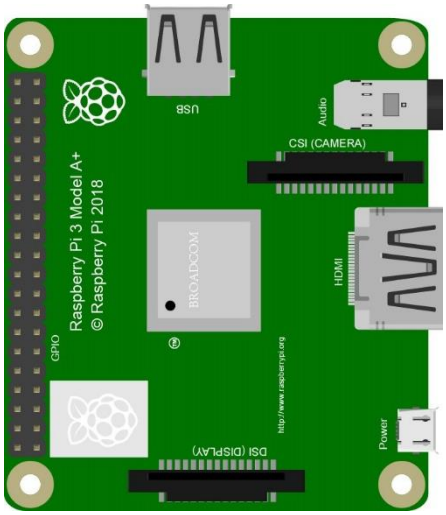
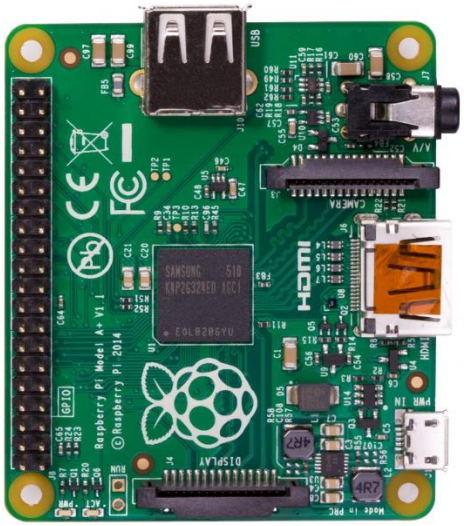
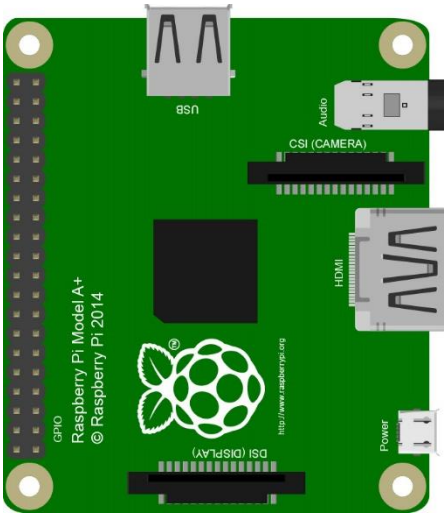
The A type and B type versions of the first generation products have been discontinued due to various reasons. What is most important is that other popular and currently available versions are consistent in the order and number of pins and their assigned designation of function, making compatibility of peripheral devices greatly enhanced between versions.

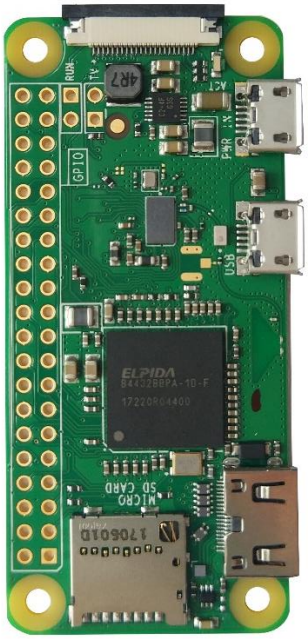
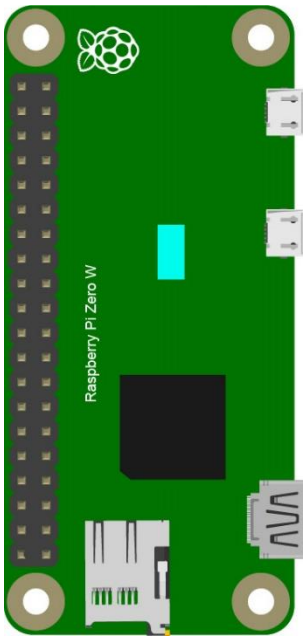
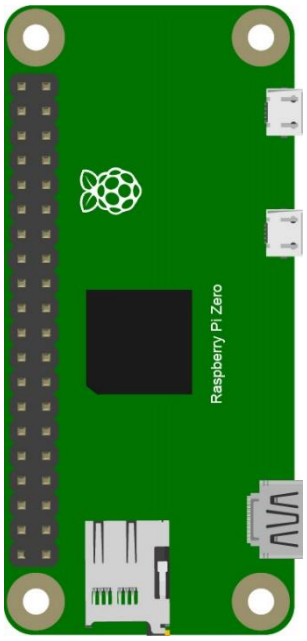
Below are the Raspberry Pi pictures and model pictures supported by this product.

Practicality picture of Raspberry Pi 4 Model B:	Model diagram of Raspberry Pi 4 Model B:
	

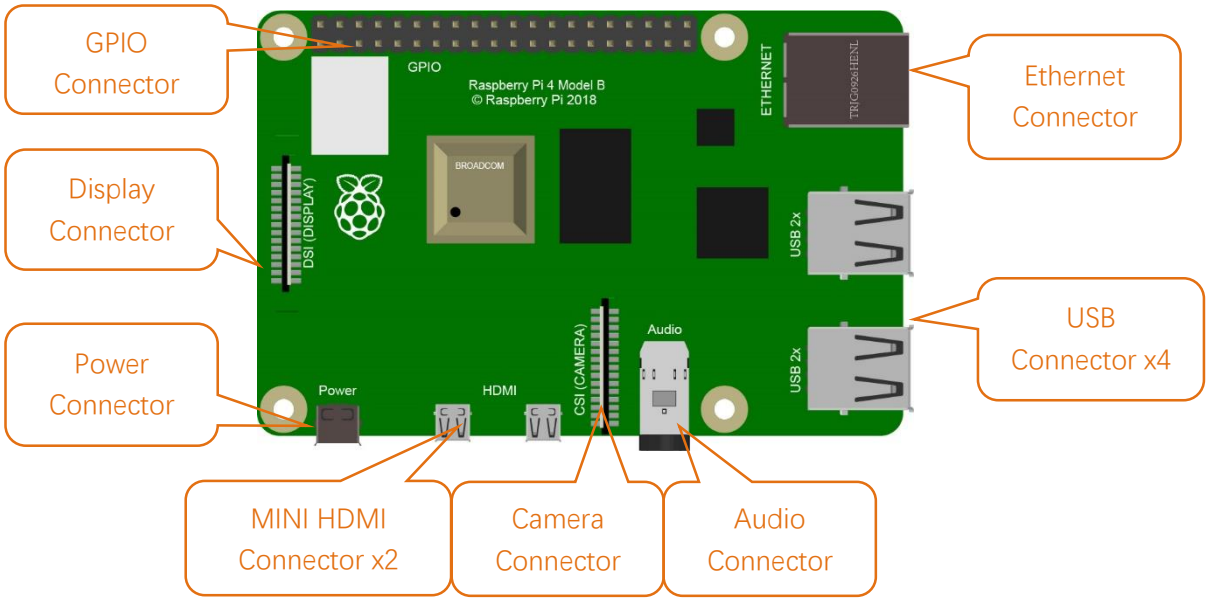
Practicality picture of Raspberry Pi 3 Model B+:	Model diagram of Raspberry Pi 3 Model B+:
	
Practicality picture of Raspberry Pi 3 Model B:	Model diagram of Raspberry Pi 3 Model B:
	

Practicality picture of Raspberry Pi 2 Model B:	Model diagram of Raspberry Pi 2 Model B:
	
Practicality picture of Raspberry Pi 1 Model B+:	Model diagram of Raspberry Pi 1 Model B+:
	

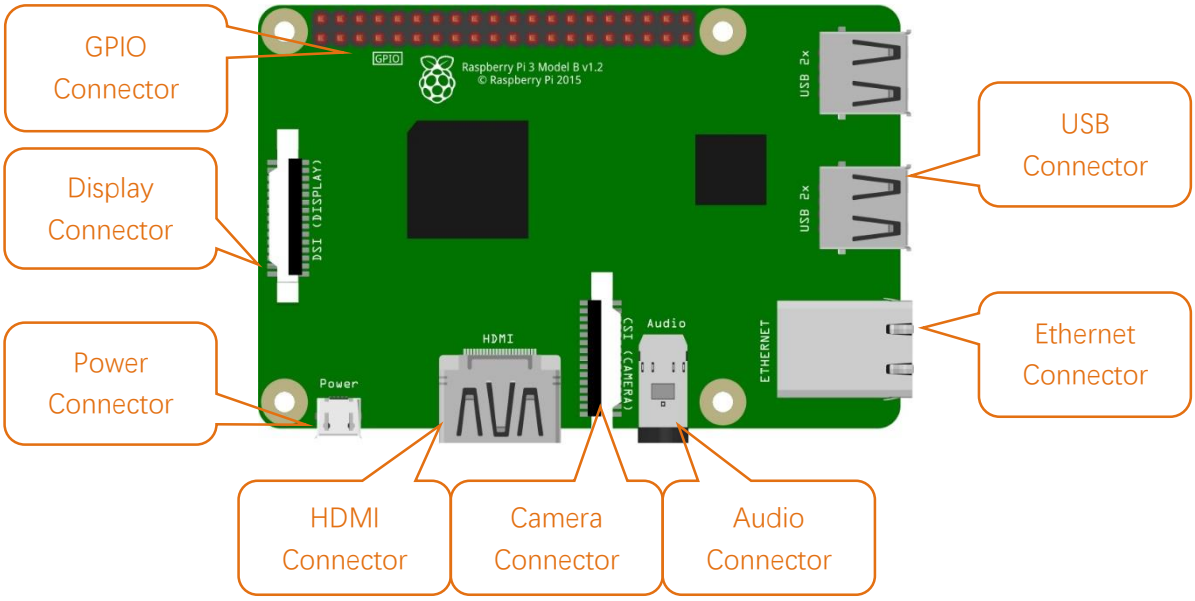
Practicality picture of Raspberry Pi 3 Model A+:	Model diagram of Raspberry Pi 3 Model A+:
	
Practicality picture of Raspberry Pi 1 Model A+:	Model diagram of Raspberry Pi 1 Model A+:
	

Practicality picture of Raspberry Pi Zero W:	Model diagram of Raspberry Pi Zero W:
	
Practicality picture of Raspberry Pi Zero:	Model diagram of Raspberry Pi Zero:
	

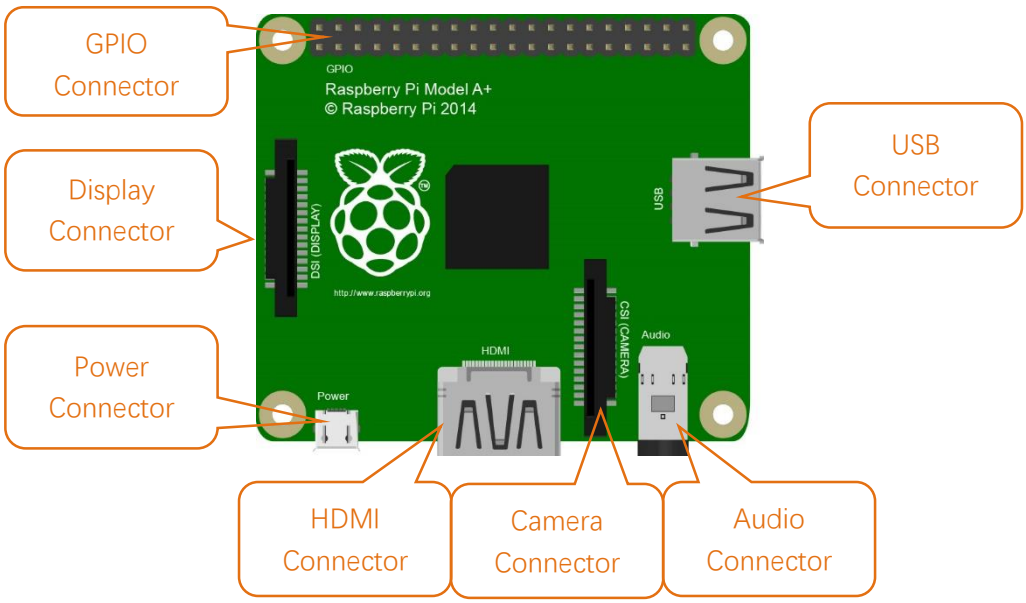
Hardware interface diagram of RPi 4B is shown below:



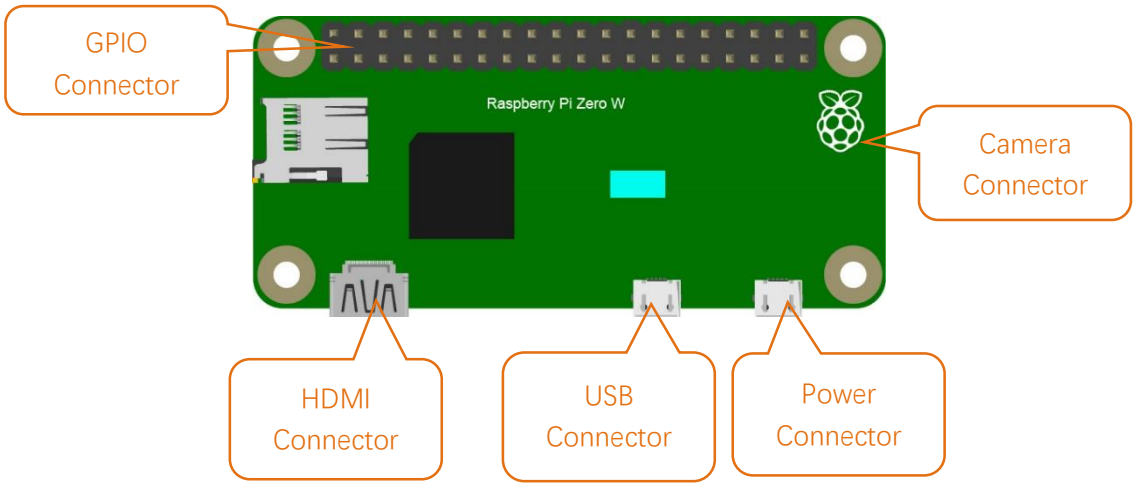
Hardware interface diagram of RPi 3B+/3B/2B/1B+ are shown below:



Hardware interface diagram of RPi 3A+/A+ is shown below:



Hardware interface diagram of RPi Zero/Zero W is shown below:

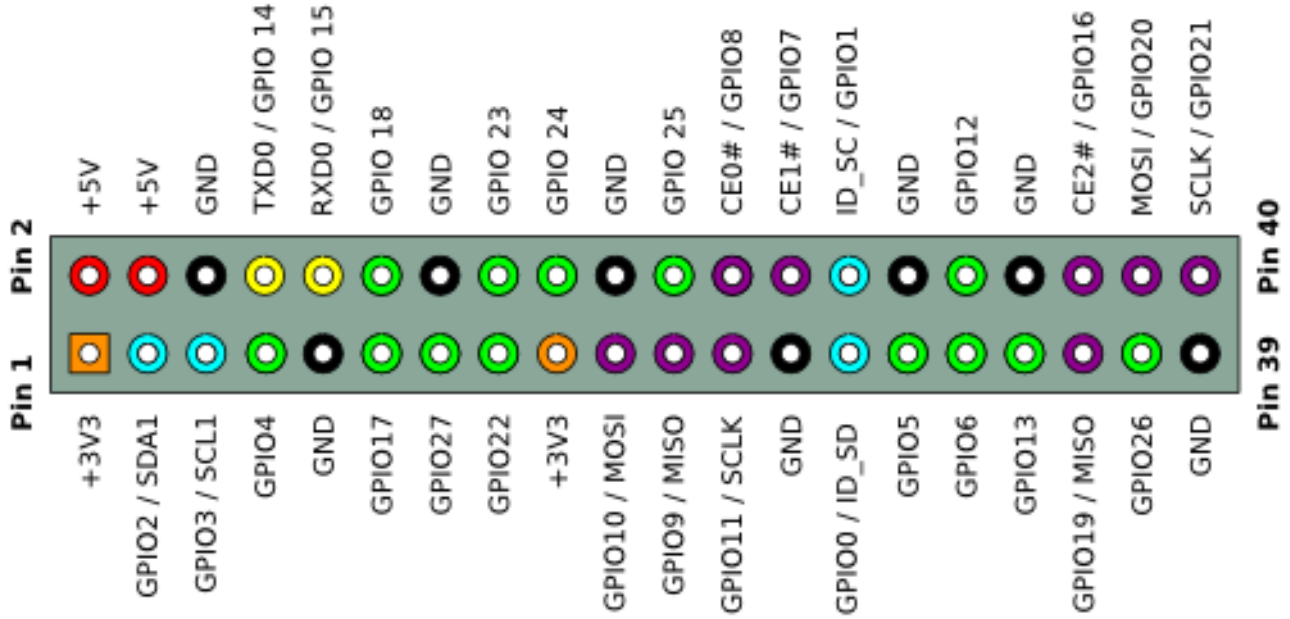


GPIO

GPIO: General purpose input/output. We will introduce the specific feature of the pins on the Raspberry Pi and how you can utilize them in all sorts of ways in your projects. Most RPi Module pins can be used as either an input or output, depending on your program and its functions. When programming the GPIO pins, there are three different ways to reference them: GPIO numbering, physical numbering, WiringPi GPIO Numbering.

BCM GPIO Numbering

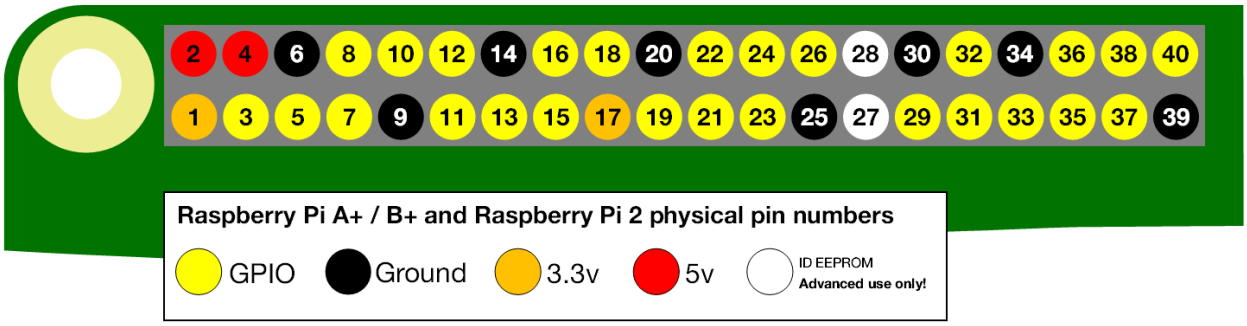
The Raspberry Pi CPU uses Broadcom (BCM) processing chips BCM2835, BCM2836 or BCM2837. GPIO pin numbers are assigned by the processing chip manufacturer and are how the computer recognizes each pin. The pin numbers themselves do not make sense or have meaning, as they are only a form of identification. Since their numeric values and physical locations have no specific order, there is no way to remember them, so you will need to have a printed reference or a reference board that fits over the pins.
Each pin is defined as below:



For more details about pin definition of GPIO, please refer to http://pinout.xyz/

PHYSICAL Numbering

Another way to refer to the pins is by simply counting across and down from pin 1 at the top left (nearest to the SD card). This is 'physical numbering', as shown below:



WiringPi GPIO Numbering

Different from the previous two types of GPIO serial numbers, RPi GPIO serial number of the WiringPi are numbered according to the BCM chip use in RPi.

wiringPi Pin	BCM GPIO	Name	Header	Name	BCM GPIO	wiringPi Pin	
—	—	3.3v	1 | 2	5v	—	—	
8	R1:0/R2:2	SDA	3 | 4	5v	—	—	
9	R1:1/R2:3	SCL	5 | 6	0v	—	—	
7	4	GPIO7	7 | 8	TxD	14	15	
—	—	0v	9 | 10	RxD	15	16	
0	17	GPIO0	11 | 12	GPIO1	18	1	
2	R1:21/R2:27	GPIO2	13 | 14	0v	—	—	
3	22	GPIO3	15 | 16	GPIO4	23	4	
—	—	3.3v	17 | 18	GPIO5	24	5	
12	10	MOSI	19 | 20	0v	—	—	
13	9	MISO	21 | 22	GPIO6	25	6	
14	11	SCLK	23 | 24	CE0	8	10	
—	—	0v	25 | 26	CE1	7	11	
30	0	SDA.0	27 | 28	SCL.0	1	31	
21	5	GPIO.21	29 | 30	0V			
22	6	GPIO.22	31 | 32	GPIO.26	12	26	
23	13	GPIO.23	33 | 34	0V			
24	19	GPIO.24	35 | 36	GPIO.27	16	27	
25	26	GPIO.25	37 | 38	GPIO.28	20	28	
		0V	39 | 40	GPIO.29	21	29	
wiringPi Pin	BCM GPIO	Name	Header	Name	BCM GPIO	wiringPi Pin	

For Pi B

For A+, B+, 2B, 3B, 3B+, 4B, Zero

(For more details, please refer to https://projects.drogon.net/raspberry-pi/wiringpi/pins/)

You can also use the following command to view their correlation.

```
gpio readall
```

```
pi@raspberrypi:~ $ gpio readall
 +-----+-----+---------+------+---+---Pi 3---+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 |     |     |    3.3v |      |   |  1 || 2  |   |      | 5v      |     |     |
 |   2 |   8 |   SDA.1 | ALT0 | 1 |  3 || 4  |   |      | 5V      |     |     |
 |   3 |   9 |   SCL.1 | ALT0 | 1 |  5 || 6  |   |      | 0v      |     |     |
 |   4 |   7 | GPIO. 7 |   IN | 1 |  7 || 8  | 1 | ALT5 | TxD     | 15  | 14  |
 |     |     |      0v |      |   |  9 || 10 | 1 | ALT5 | RxD     | 16  | 15  |
 |  17 |   0 | GPIO. 0 |   IN | 0 | 11 || 12 | 0 | IN   | GPIO. 1 | 1   | 18  |
 |  27 |   2 | GPIO. 2 |   IN | 0 | 13 || 14 |   |      | 0v      |     |     |
 |  22 |   3 | GPIO. 3 |   IN | 0 | 15 || 16 | 0 | IN   | GPIO. 4 | 4   | 23  |
 |     |     |    3.3v |      |   | 17 || 18 | 0 | IN   | GPIO. 5 | 5   | 24  |
 |  10 |  12 |    MOSI | ALT0 | 0 | 19 || 20 |   |      | 0v      |     |     |
 |   9 |  13 |    MISO | ALT0 | 0 | 21 || 22 | 0 | IN   | GPIO. 6 | 6   | 25  |
 |  11 |  14 |    SCLK | ALT0 | 0 | 23 || 24 | 1 | OUT  | CE0     | 10  | 8   |
 |     |     |      0v |      |   | 25 || 26 | 1 | OUT  | CE1     | 11  | 7   |
 |   0 |  30 |   SDA.0 |   IN | 1 | 27 || 28 | 1 | IN   | SCL.0   | 31  | 1   |
 |   5 |  21 | GPIO.21 |   IN | 1 | 29 || 30 |   |      | 0v      |     |     |
 |   6 |  22 | GPIO.22 |   IN | 1 | 31 || 32 | 0 | IN   | GPIO.26 | 26  | 12  |
 |  13 |  23 | GPIO.23 |   IN | 0 | 33 || 34 |   |      | 0v      |     |     |
 |  19 |  24 | GPIO.24 |   IN | 0 | 35 || 36 | 0 | IN   | GPIO.27 | 27  | 16  |
 |  26 |  25 | GPIO.25 |   IN | 0 | 37 || 38 | 0 | IN   | GPIO.28 | 28  | 20  |
 |     |     |      0v |      |   | 39 || 40 | 0 | IN   | GPIO.29 | 29  | 21  |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+---Pi 3---+---+------+---------+-----+-----+
```

For more details about wiringPi, please refer to http://wiringpi.com/ .

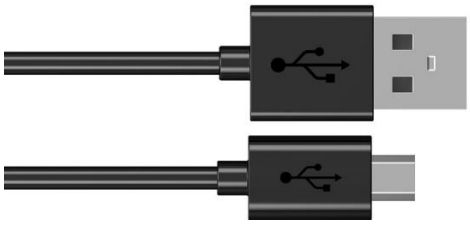
Chapter 0 Preparation

Install a System

Firstly, install a system for your RPi.

Component List

Required Components

Raspberry Pi 4B / 3B+/ 3B /3A+ (Recommended) 	5V/3A Power Adapter. Different versions of Raspberry Pi have different power requirements. 
Micro USB Cable x1 	Micro SD Card (TF Card) x1, Card Reader x1 

This robot also supports the following versions of the Raspberry Pi, but **additional accessories** need to be prepared by yourself.

Raspberry	Additional accessories
Raspberry Pi Zero W	Camera cable(>25cm) for zero w, 15 Pin 1.0mm Pitch to 22 Pin 0.5mm https://www.amazon.com/dp/B076Q595HJ/
Raspberry Pi Zero 1.3	wireless network adapter, Camera cable(>25cm) for zero w, 15 Pin 1.0mm Pitch to 22 Pin 0.5mm, OTG cable (USB Type micro B to USB Type A)
Raspberry Pi 2 Model B	wireless network adapter
Raspberry Pi 1 Model A+	wireless network adapter
Raspberry Pi 1 Model B+	wireless network adapter

Power requirements of various versions of Raspberry Pi are shown in following table:

Product	Recommended PSU current capacity	Maximum total USB peripheral current draw	Typical bare-board active current consumption
Raspberry Pi Model A	700mA	500mA	200mA
Raspberry Pi Model B	1.2A	500mA	500mA
Raspberry Pi Model A+	700mA	500mA	180mA
Raspberry Pi Model B+	1.8A	600mA/1.2A (switchable)	330mA
Raspberry Pi 2 Model B	1.8A	600mA/1.2A (switchable)	350mA
Raspberry Pi 3 Model B	2.5A	1.2A	400mA
Raspberry Pi 3 Model A+	2.5A	Limited by PSU, board, and connector ratings only.	350mA
Raspberry Pi 3 Model B+	2.5A	1.2A	500mA
Raspberry Pi 4 Model B	3.0A	1.2A	600mA
Raspberry Pi Zero W	1.2A	Limited by PSU, board, and connector ratings only.	150mA
Raspberry Pi Zero	1.2A	Limited by PSU, board, and connector ratings only	100mA

For more details, please refer to https://www.raspberrypi.org/help/faqs/#powerReqs

All these components are necessary for any of your projects to work. Among them, the power supply of at least 5V/2.5A, because a lack of a sufficient power supply may lead to many functional issues and even damage your RPi, we STRONGLY RECOMMEND a 5V/2.5A power supply. We also recommend using a SD Micro Card with a capacity 16GB or more (which, functions as the RPI's "hard drive") and is used to store the operating system and necessary operational files.

In future projects, the components list with a RPi will contain these required components, using only RPi as a representative rather than presenting details.

Optional Components

Under normal circumstances, there are two ways to login to Raspberry Pi: 1) Using a stand-alone monitor. 2) Using a remote desktop computer monitor "sharing" the PC monitor with your RPi.

Required Accessories for Monitor

If you want to use an independent monitor, mouse and keyboard, you also need the following accessories.

1. A Display with HDMI interface
2. A Mouse and a Keyboard with an USB interface

As to Pi Zero and Pi Zero W, you also need the following accessories:

1. A Mini-HDMI to HDMI Adapter and Cable.
2. A Micro-USB to USB-A Adapter and Cable (Micro USB OTG Cable).
3. A USB HUB.
4. USB to Ethernet Interface or USB Wi-Fi receiver.

For different Raspberry Pi, the optional items are slightly different. But they all aim to convert the interfaces to Raspberry Pi standards.

Item	Pi Zero	Pi Zero W	Pi A+	Pi 3A+	Pi B+/2B	Pi 3B/3B+/4B
Monitor	Yes	Yes	Yes	Yes	Yes	Yes
Mouse	Yes	Yes	Yes	Yes	Yes	Yes
Keyboard	Yes	Yes	Yes	Yes	Yes	Yes
Micro-HDMI to HDMI cable	Yes	Yes	No	No	No	No
Micro-USB to USB-A OTG cable	Yes	Yes	No	No	No	No
USB HUB	Yes	Yes	Yes	Yes	No	No
USB transferring to Ethernet interface	select one from two or select two from two	optional	select one from two or select two from two	optional	Internal Integration	Internal Integration
USB Wi-Fi receiver		Internal Integration		Internal Integration	optional	

Required Accessories for Remote Desktop

If you do not have an independent monitor, or if you want to use a remote desktop, first you need to login to Raspberry Pi through SSH, then open the VNC or RDP service. This requires the following accessories.

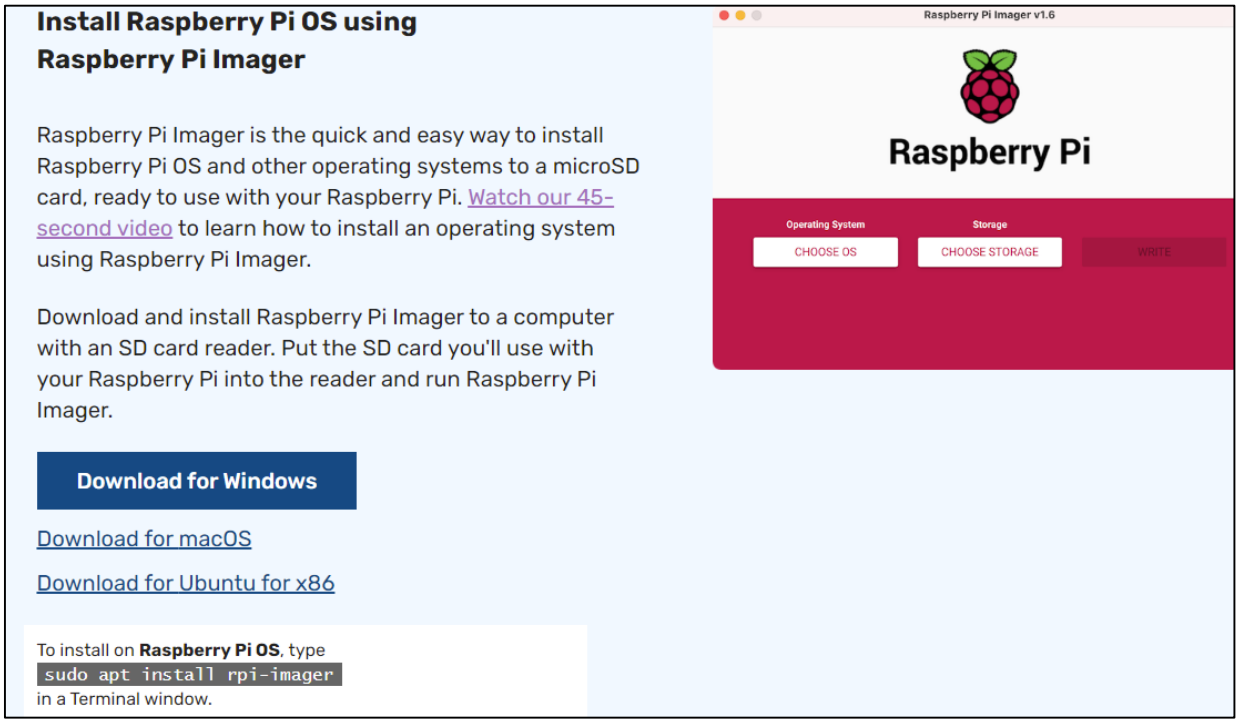
Item	Pi Zero	Pi Zero W	Pi A+	Pi 3A+	Pi B+/2B	Pi 3B/3B+/4B
Micro-USB to USB-A OTG cable	Yes	Yes	No	NO		
USB transferring to Ethernet interface	Yes	Yes	Yes			

Raspberry Pi OS

Install imager tool

Visit this website to install imager tool.

https://www.raspberrypi.com/software/

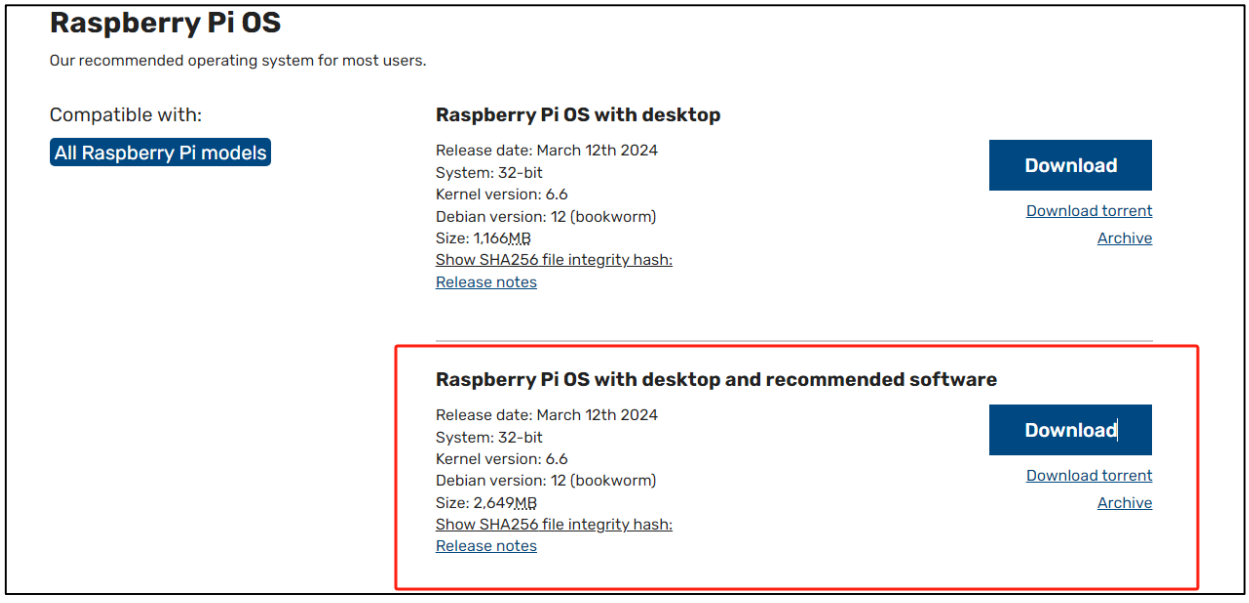


Download OS file

Due to the poor software pwm performance of Raspberry PI 5, we recommend using Raspberry PI 4 or below as the main control board of the robot arm.

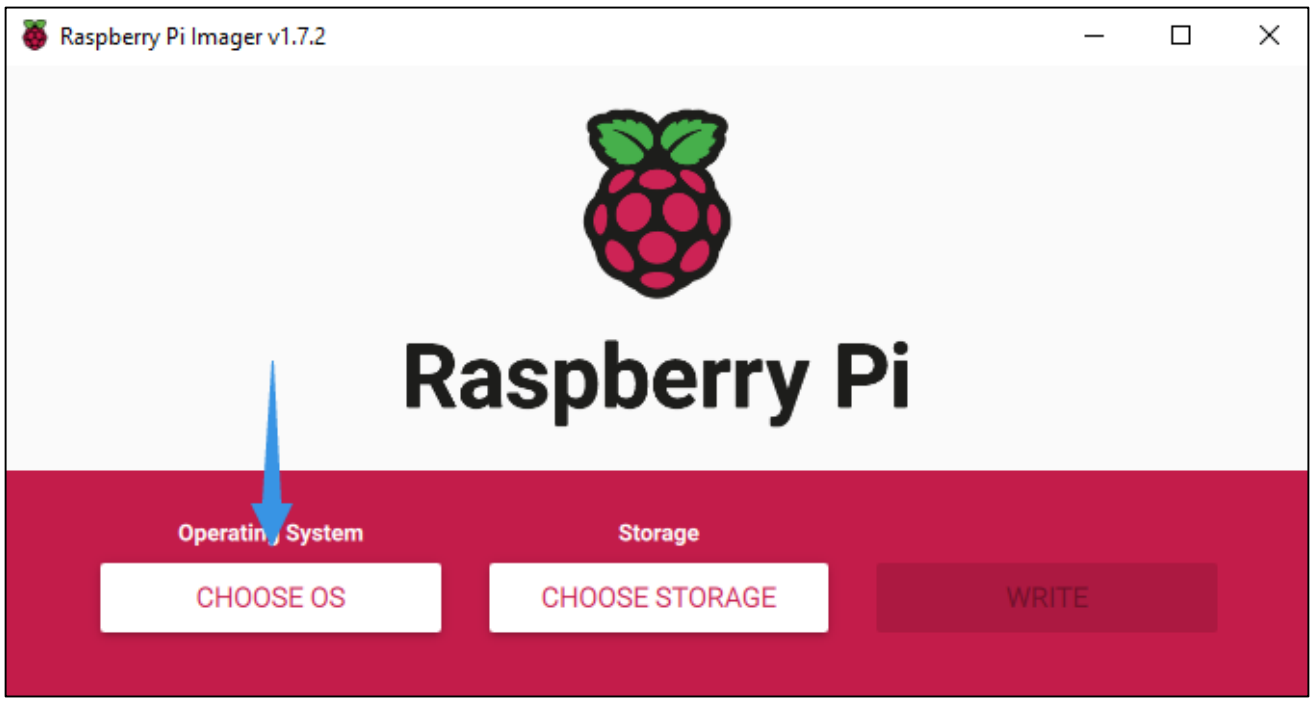
Visit following website to download the OS file.

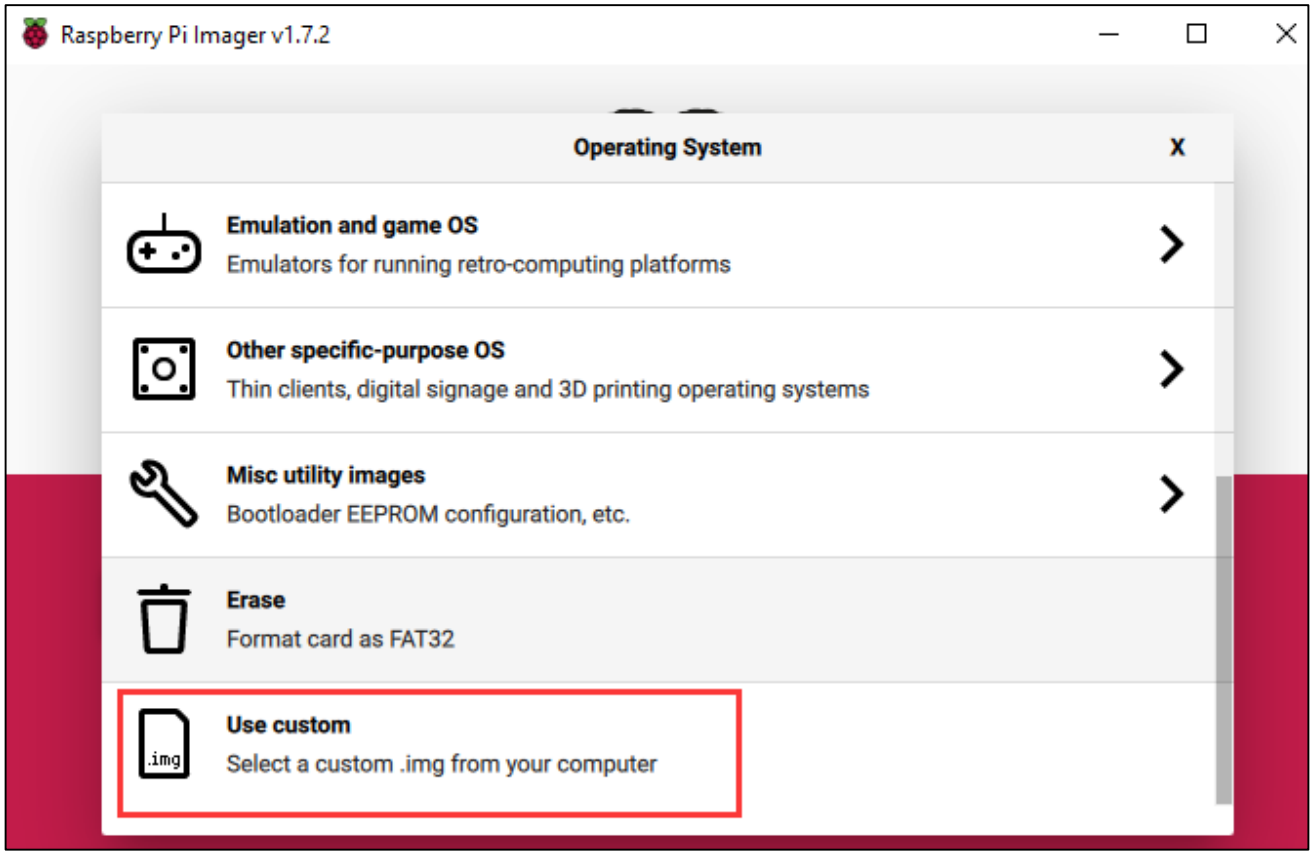
https://www.raspberrypi.com/software/operating-systems/



Write System to Micro SD Card

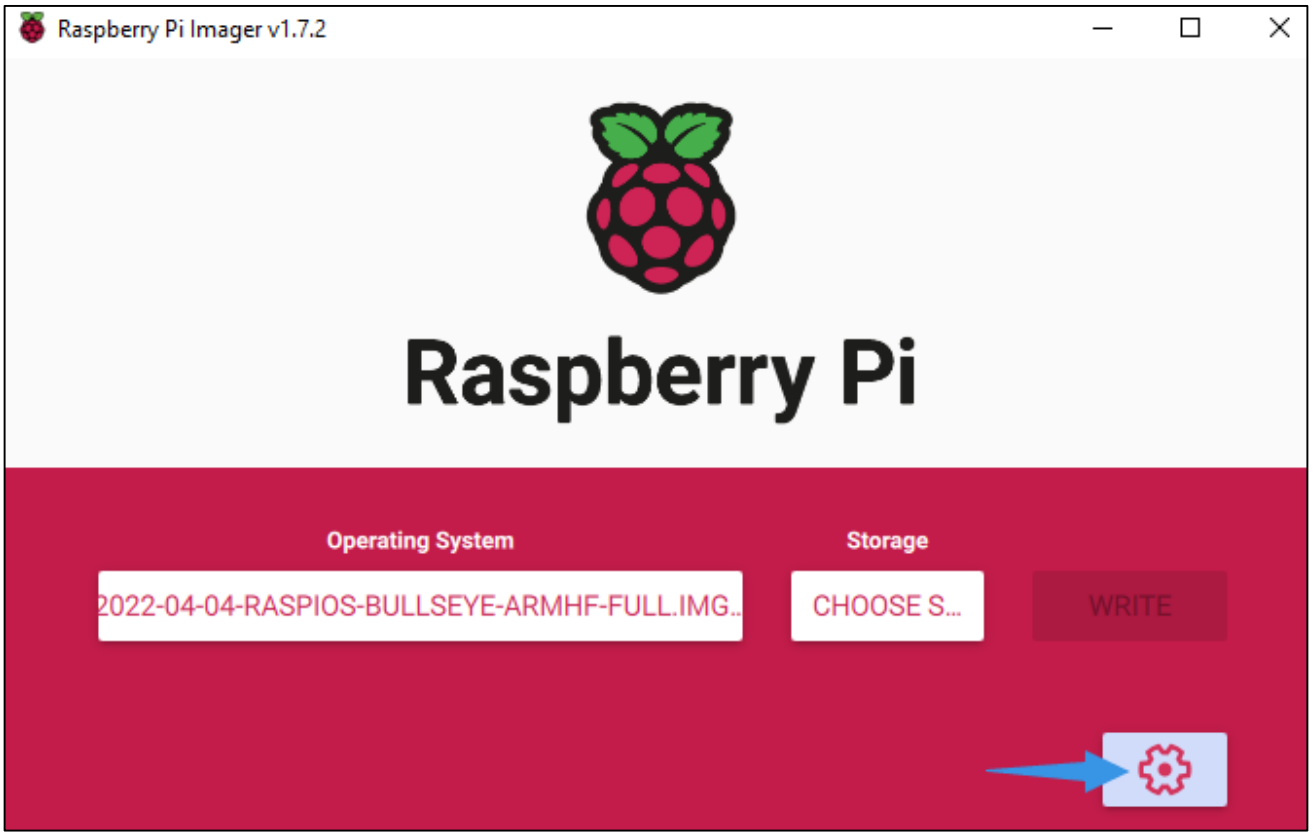
First, insert your Micro **SD card** into **card reader** and connect it to USB port of **PC**. Then open imager tool. Choose system that you just downloaded in Use custom.



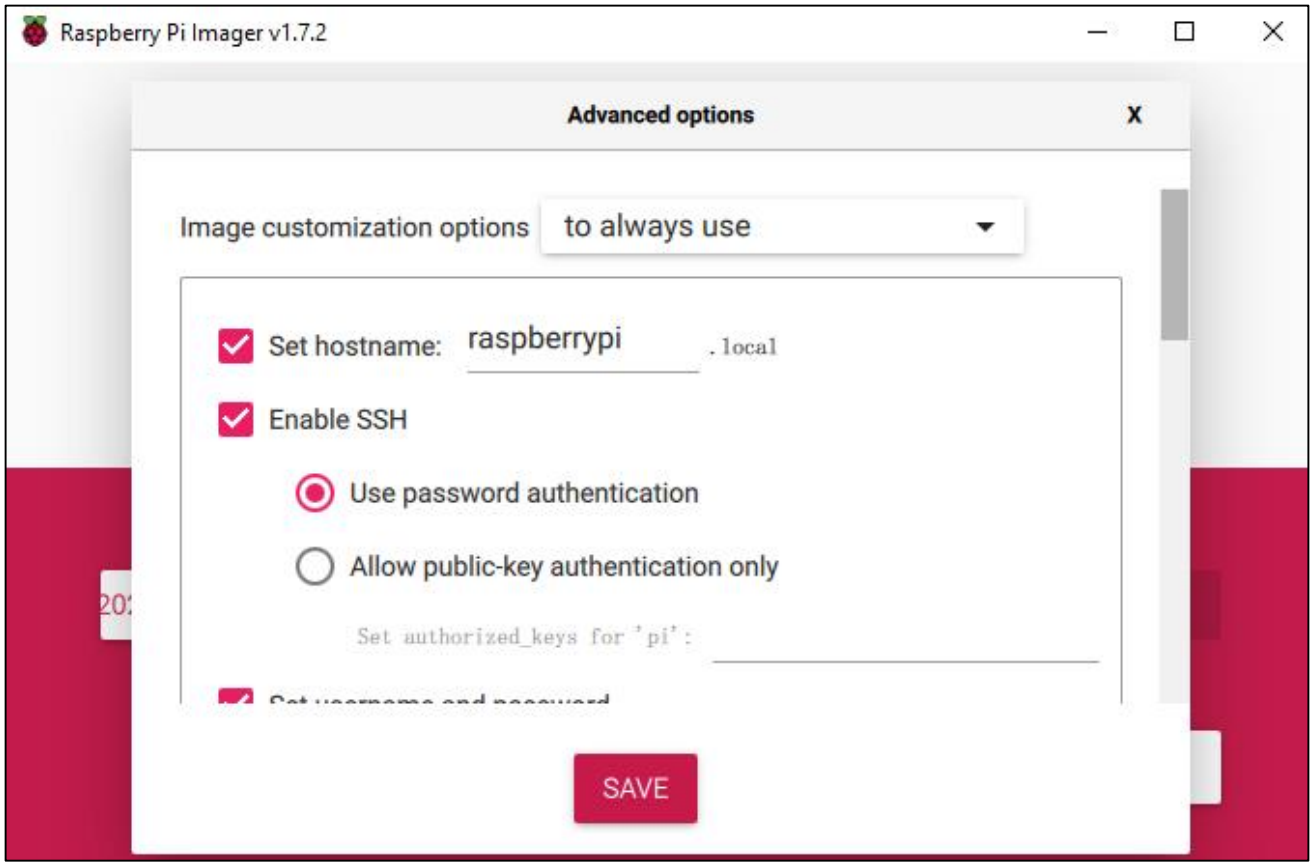


Choose the SD card.

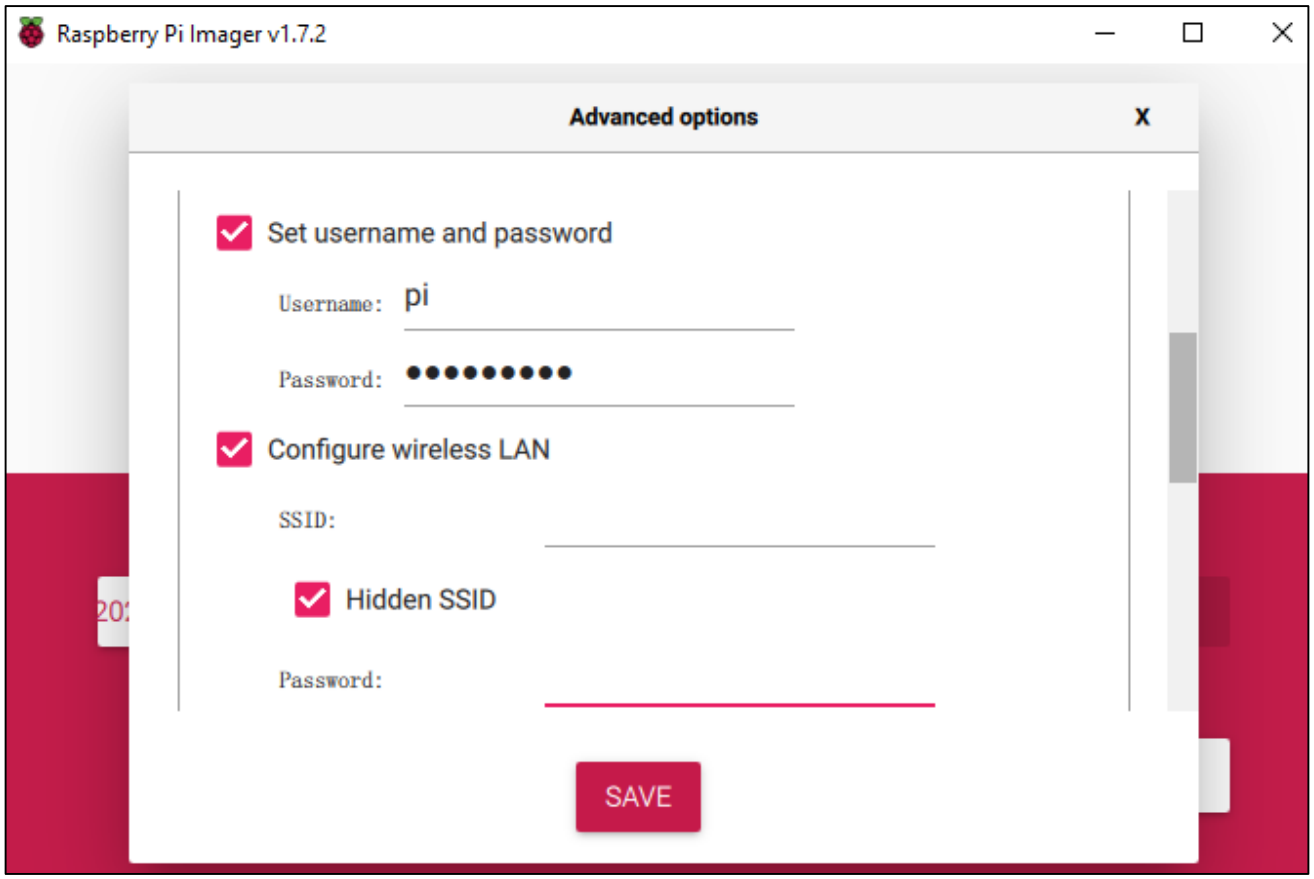
Image option.



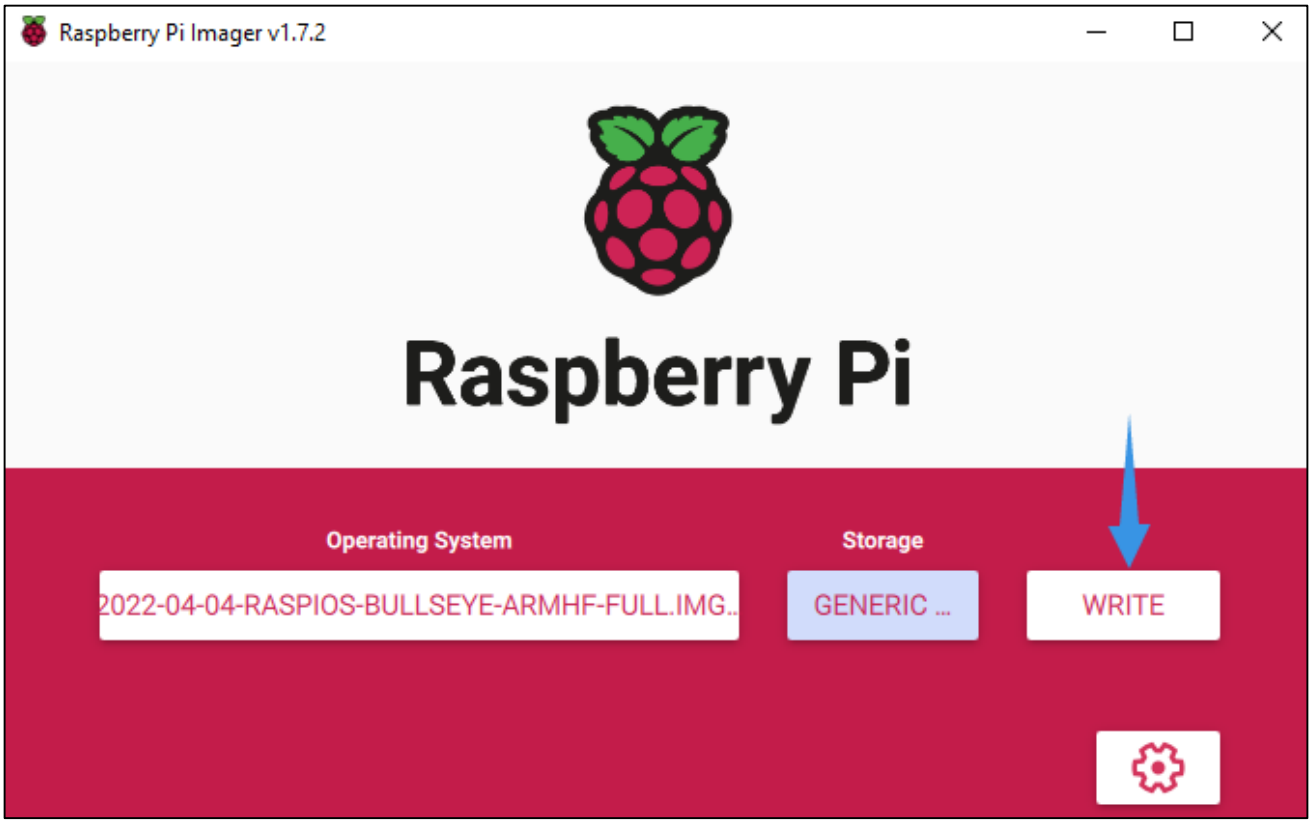
Enable SSH.



Configure WiFi and location. Here we set username as **pi**, password as **raspberry**



Finally WRITE.

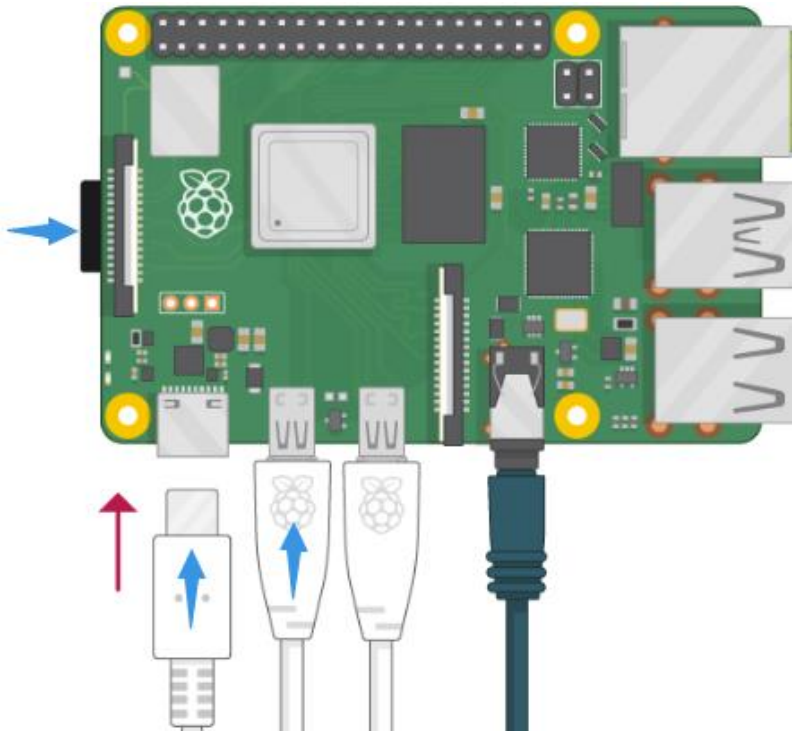


Start Raspberry Pi

If you don't have a spare monitor, please skip to next section.

If you have a spare monitor, please follow the steps in this section.

After the system is written successfully, put SD card into the SD card slot of RPi. Then connect your RPi to monitor through HDMI cable, attach your mouse and keyboard through the USB ports,



Later, after setup, you will need to enter your user name and password to login. The default user name: pi; password: raspberry. After login, you should see the following screen.

You can connect WiFi on the right corner if WiFi is connected successfully.

Now you can skip to VNC Viewer.

Remote desktop & VNC

After you log in Raspberry Pi, please use VNC Viewer to connect Raspberry Pi for this robot. Other remote ways may not support GUI. **If you have logged in Raspberry Pi please skip to VNC Viewer.**

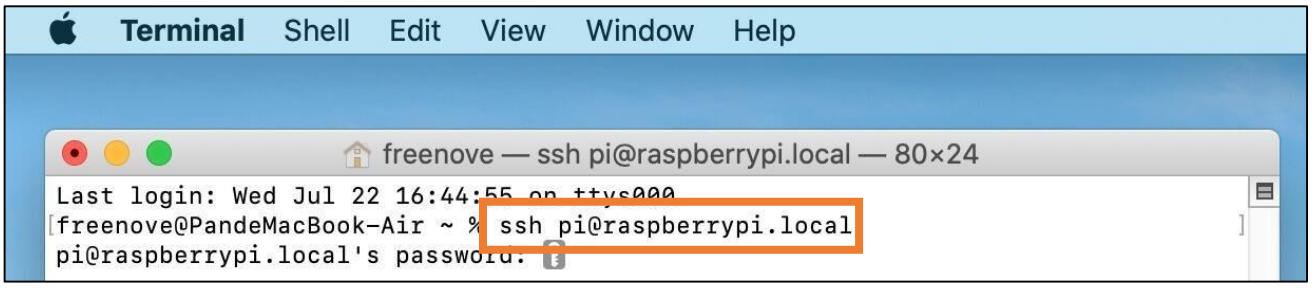
If you don't have a spare monitor, mouse and keyboard for your RPi, you can use a remote desktop to share a display, keyboard, and mouse with your PC. Below is how to use:
MAC OS remote desktop and Windows OS remote desktop.

MAC OS Remote Desktop

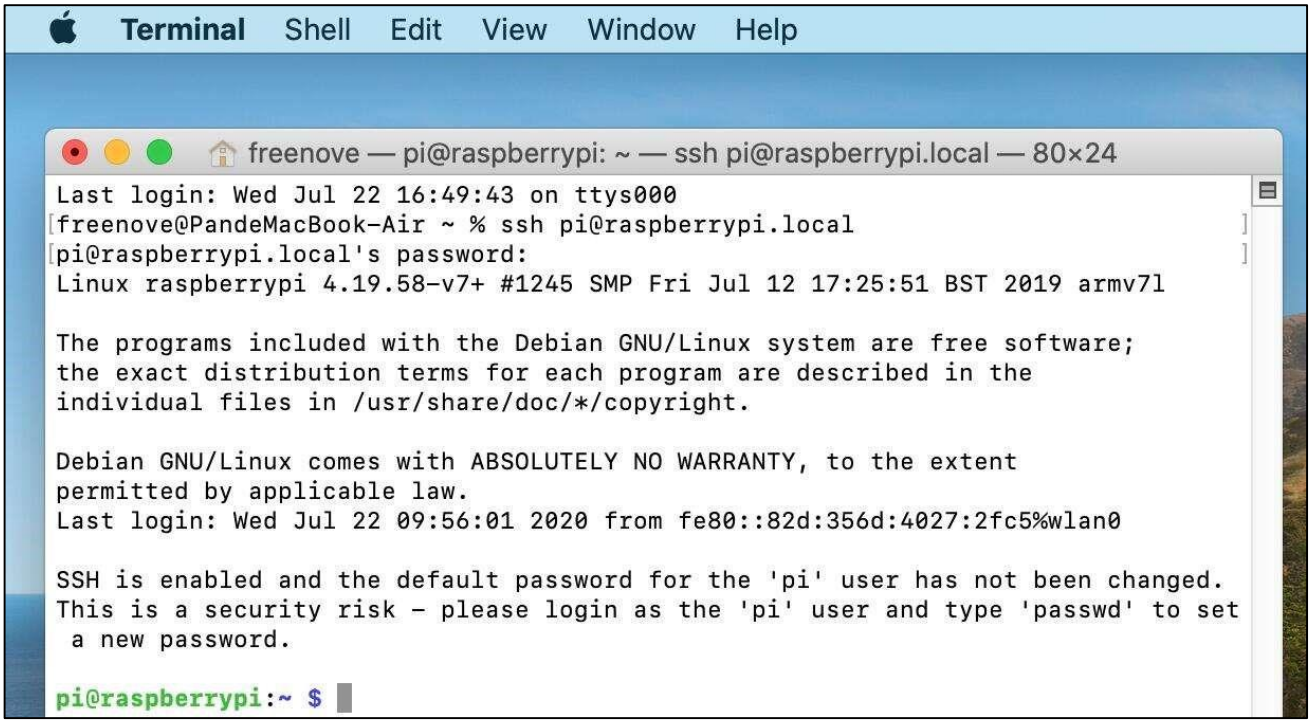
Open the terminal and type following command. **If this command doesn't work, please move to next page.**

```
ssh pi@raspberrypi.local
```

The password is **raspberry** by default, case sensitive.



You may need to type **yes** during the process.



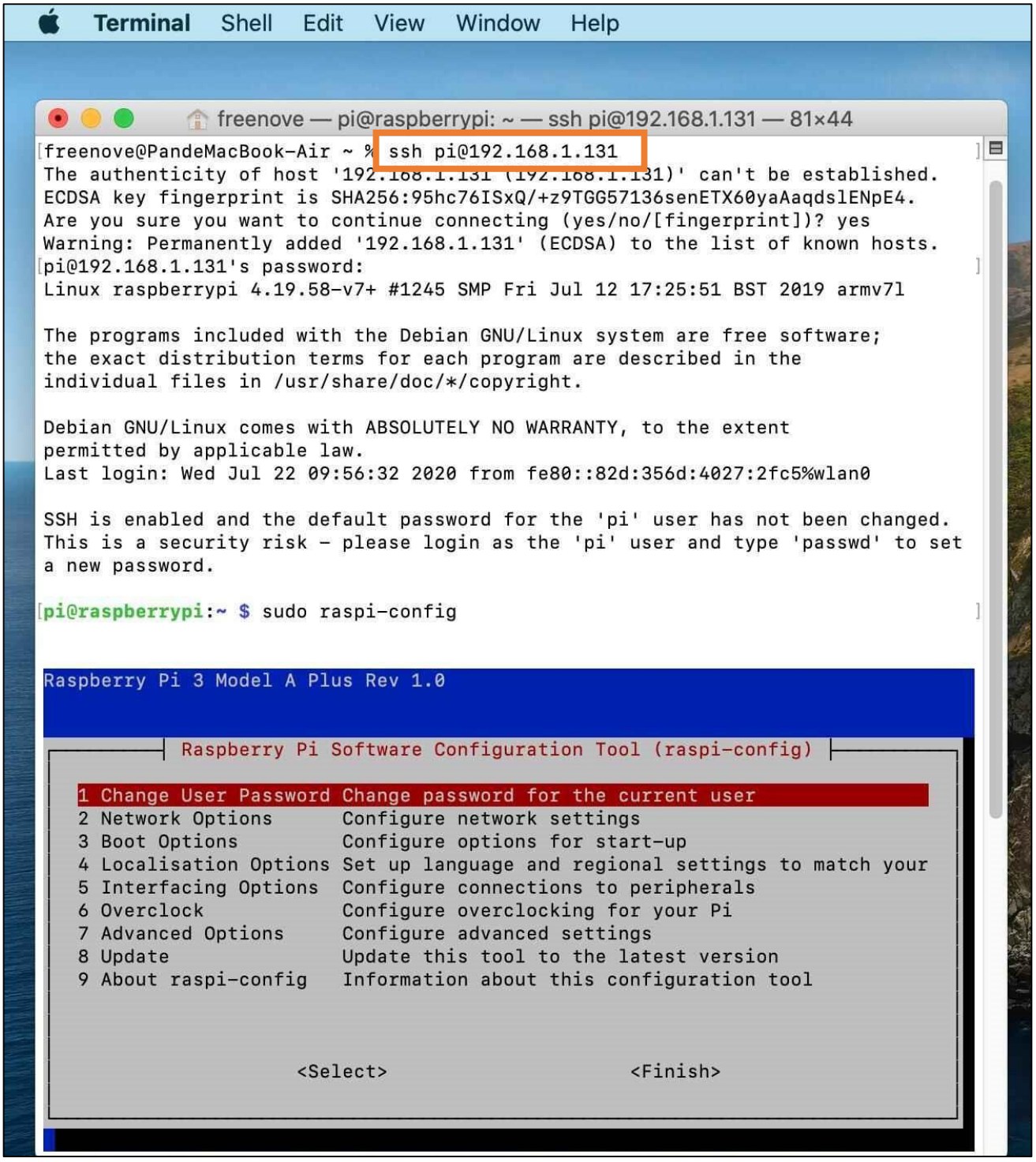
You can also use the IP address to log in Pi.

Enter **router** client to **inquiry IP address** named "raspberry pi". For example, I have inquired to **my RPi IP address, and it is "192.168.1.131".**

Open the terminal and type following command.

```
ssh pi@192.168.1.131
```

When you see pi@raspberrypi:~ $, you have logged in Pi successfully. Then you can skip to next section.



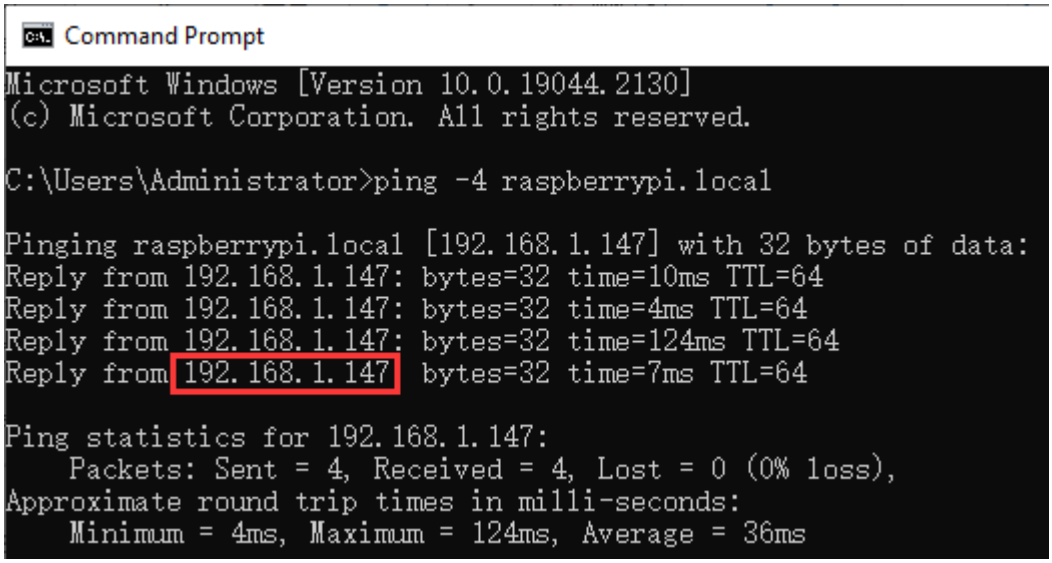
Then you can skip to VNC Viewer.

Windows OS Remote Desktop

If you are using win10, you can use follow way to login Raspberry Pi without desktop.

Press **Win+R**. Enter **cmd**. Then use this command to check IP:

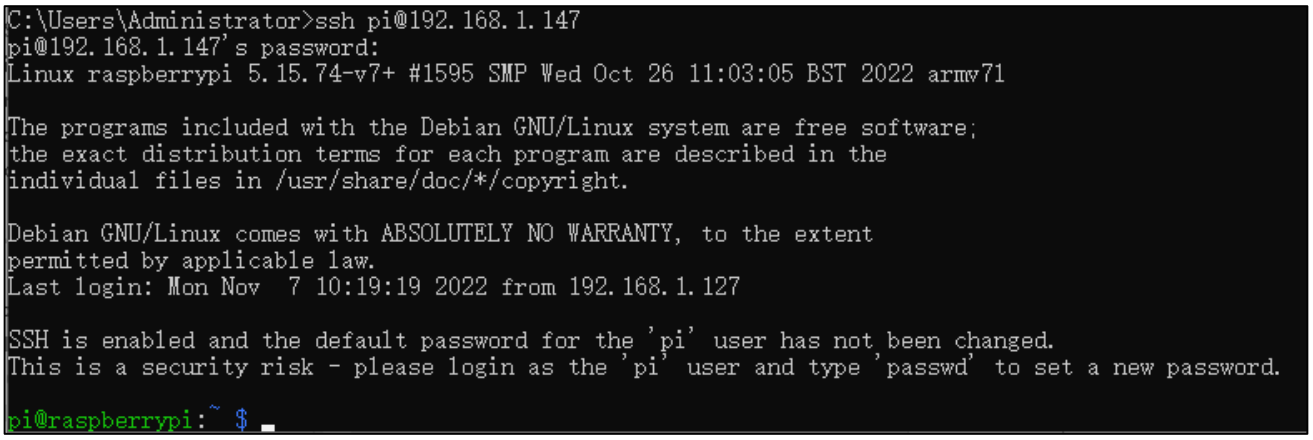
```
ping -4 raspberrypi.local
```



Then 192.168.1.147 is my Raspberry Pi IP.

Or enter **router** client to **inquiry IP address** named "**raspberrypi**". For example, I have inquired to **my RPi IP address, and it is "192.168.1.147".**

```
ssh pi@192.168.1.147
```

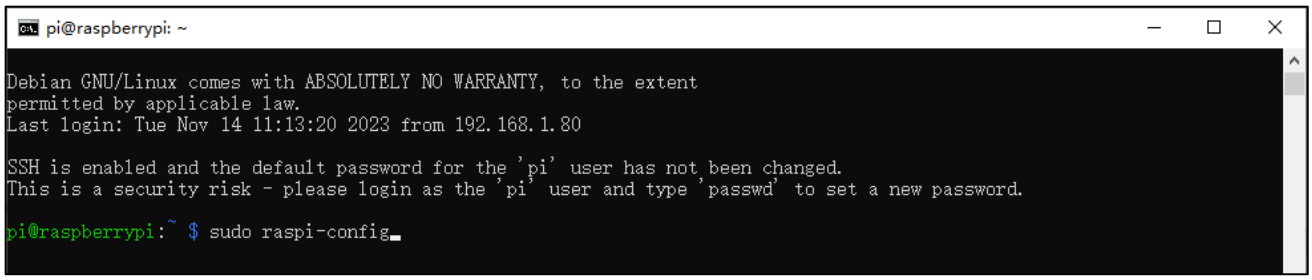


VNC Viewer & VNC

Open the Configuration Interface

Run the following command to open the configuration interface.

```
sudo raspi-config
```

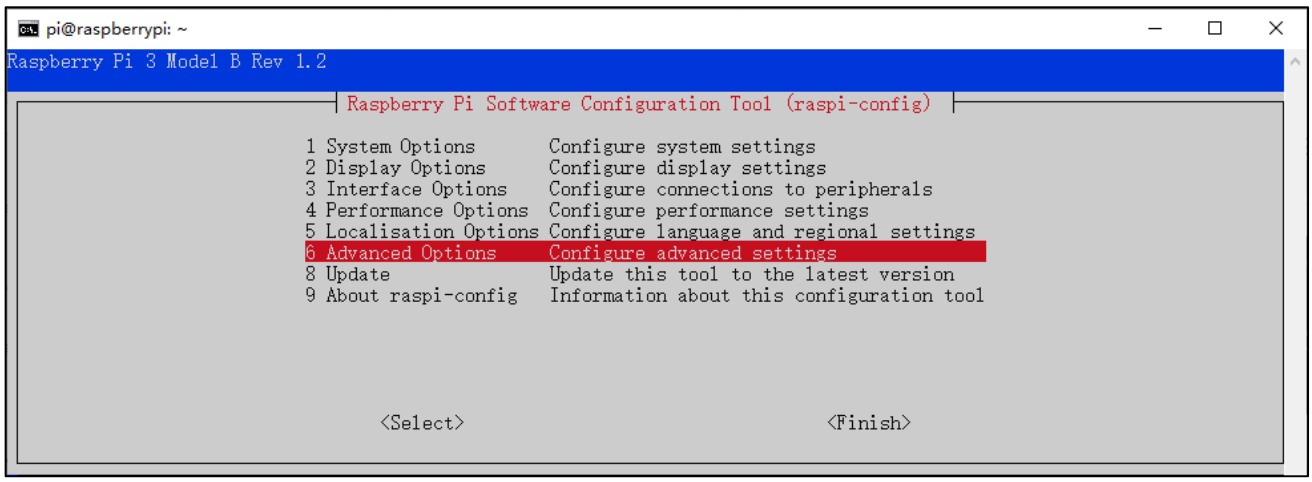


Debian Bookworm

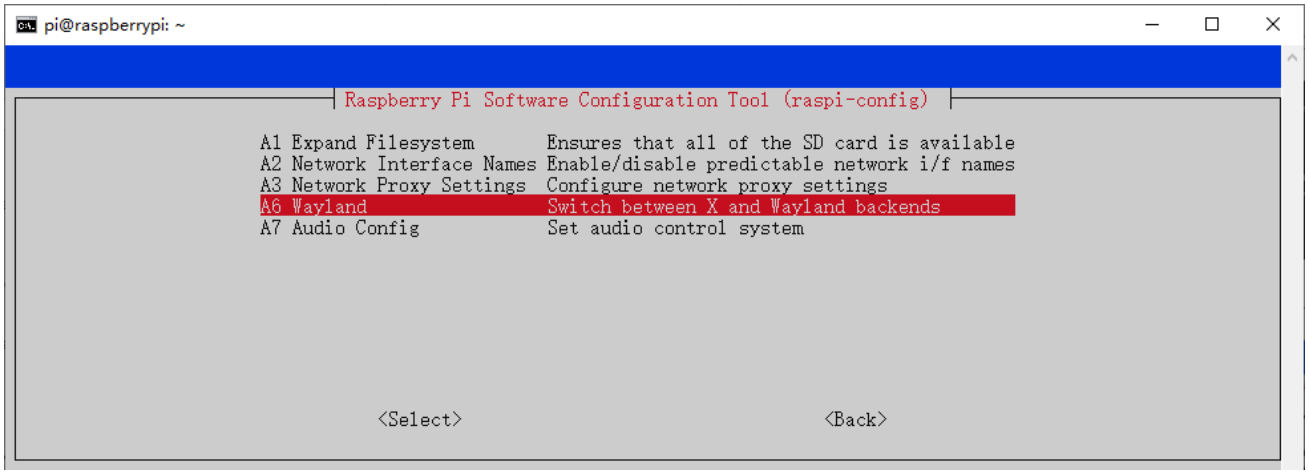
If your Raspberry Pi OS is Debian Bookworm system, please follow this section to operate. Here we take the version Raspberry Pi OS Full (64-bit) released on 2023-10-10 as an example.

If your RPi OS is not Debian Bookworm, you can skip this section.

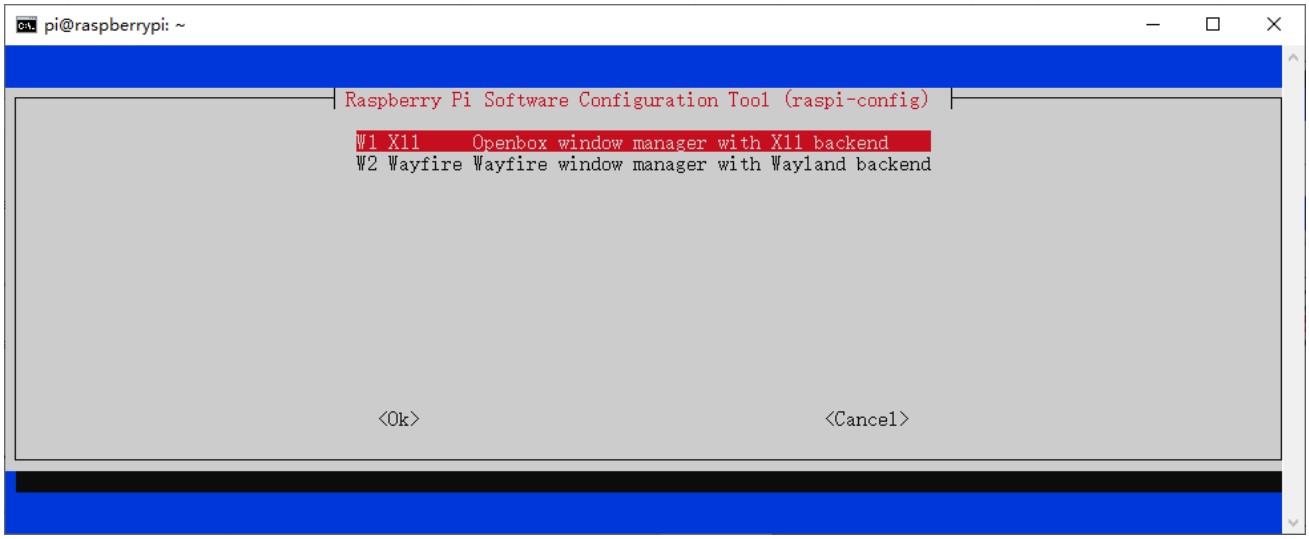
Select Advanced Options.

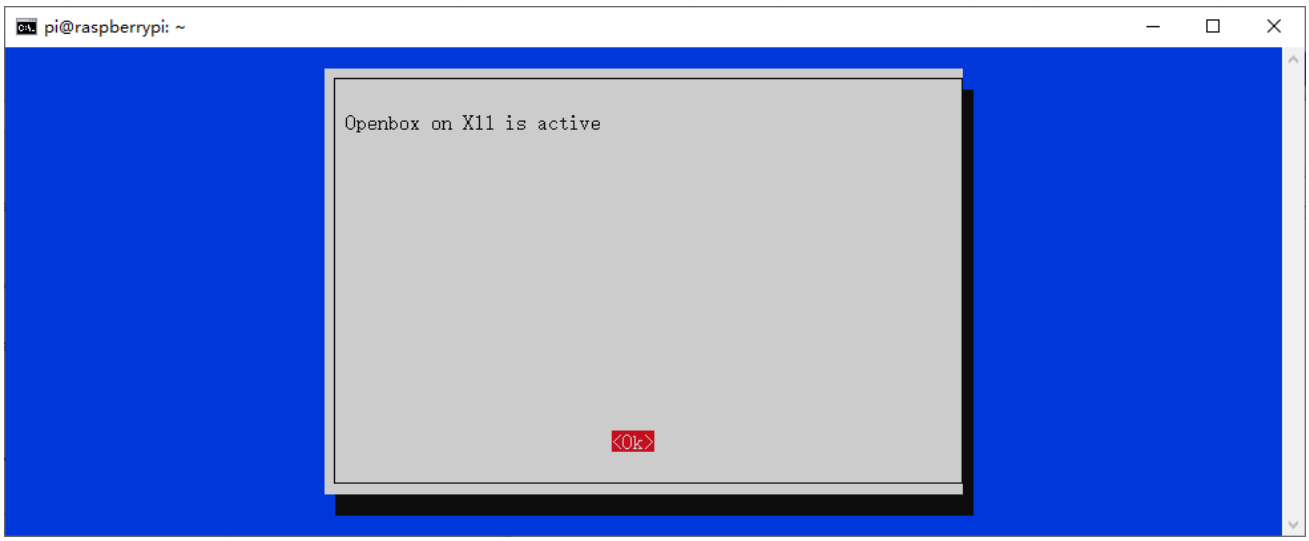


Select Wayland.



Select X11, press Enter and select OK.

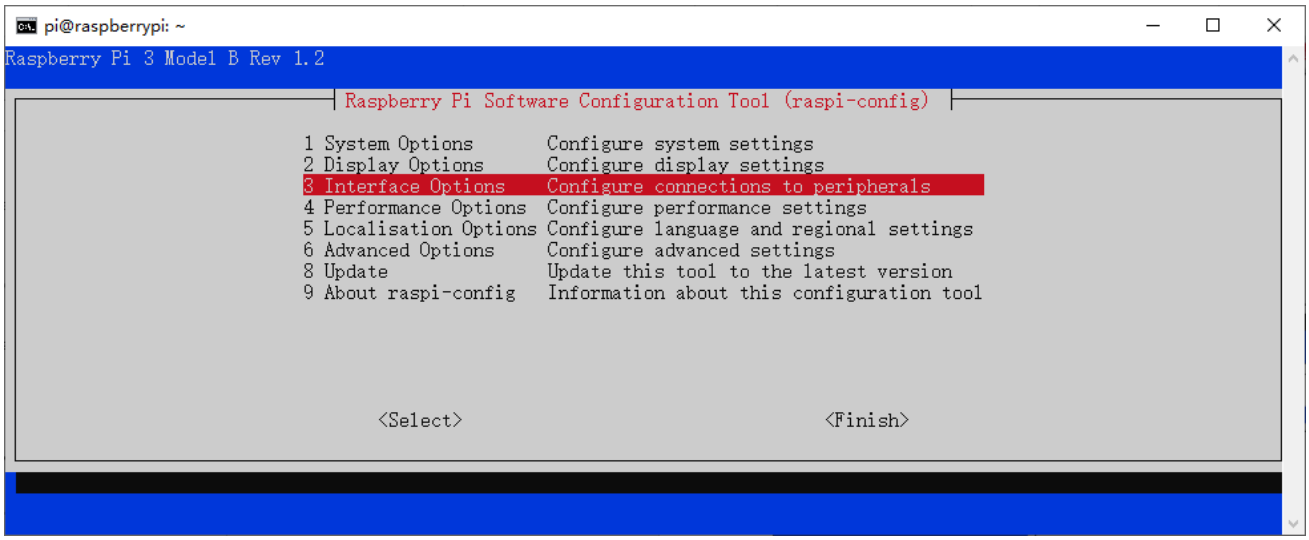




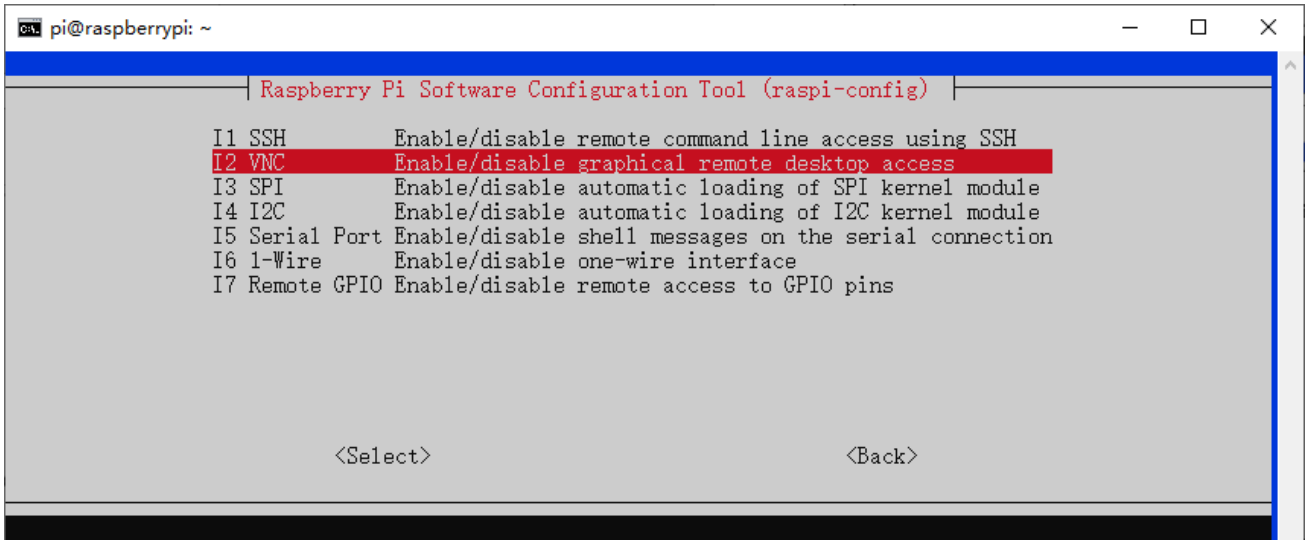
VNC Configuration

To use VNC Viewer, you need to enable it first.

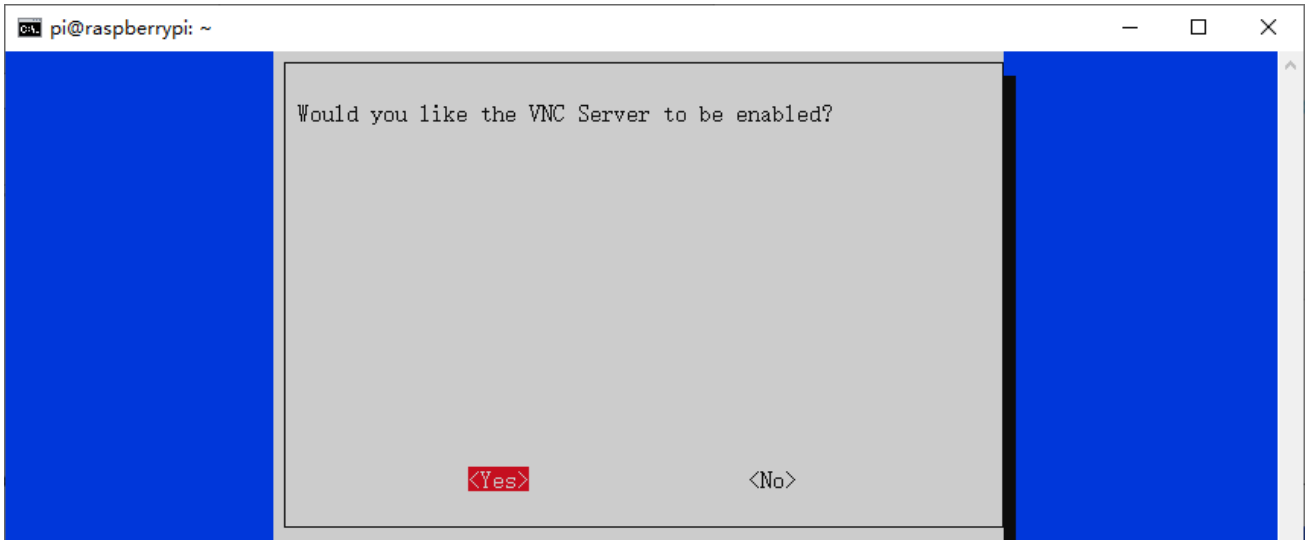
Select Interface Options.



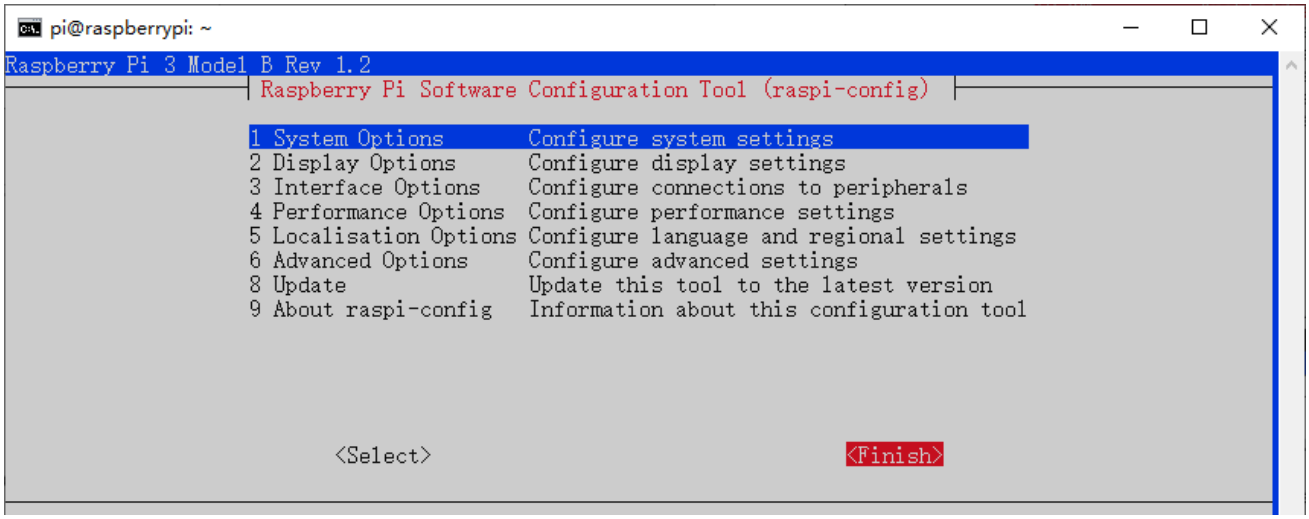
Select VNC.



Select Yes.

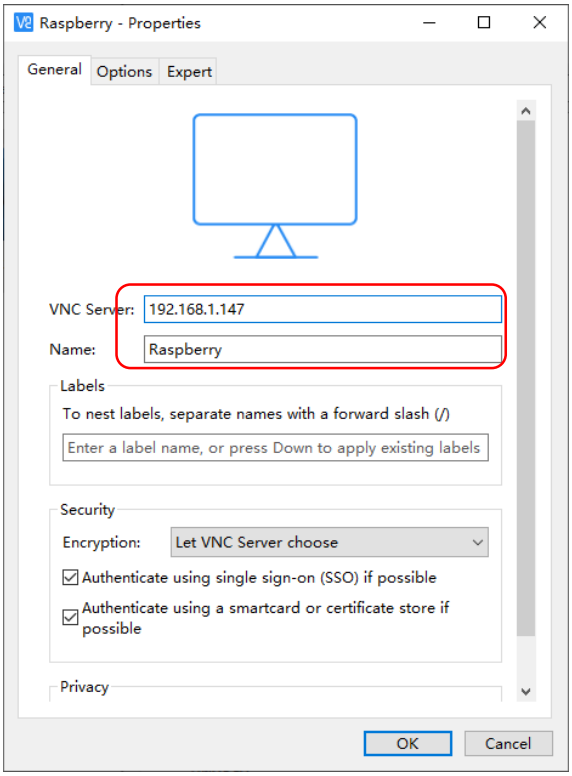


Select Finish.

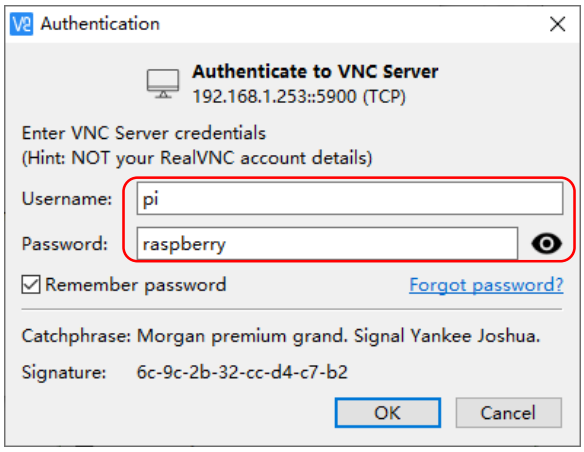


Reboot your Raspberry Pi and the settings will take effect.

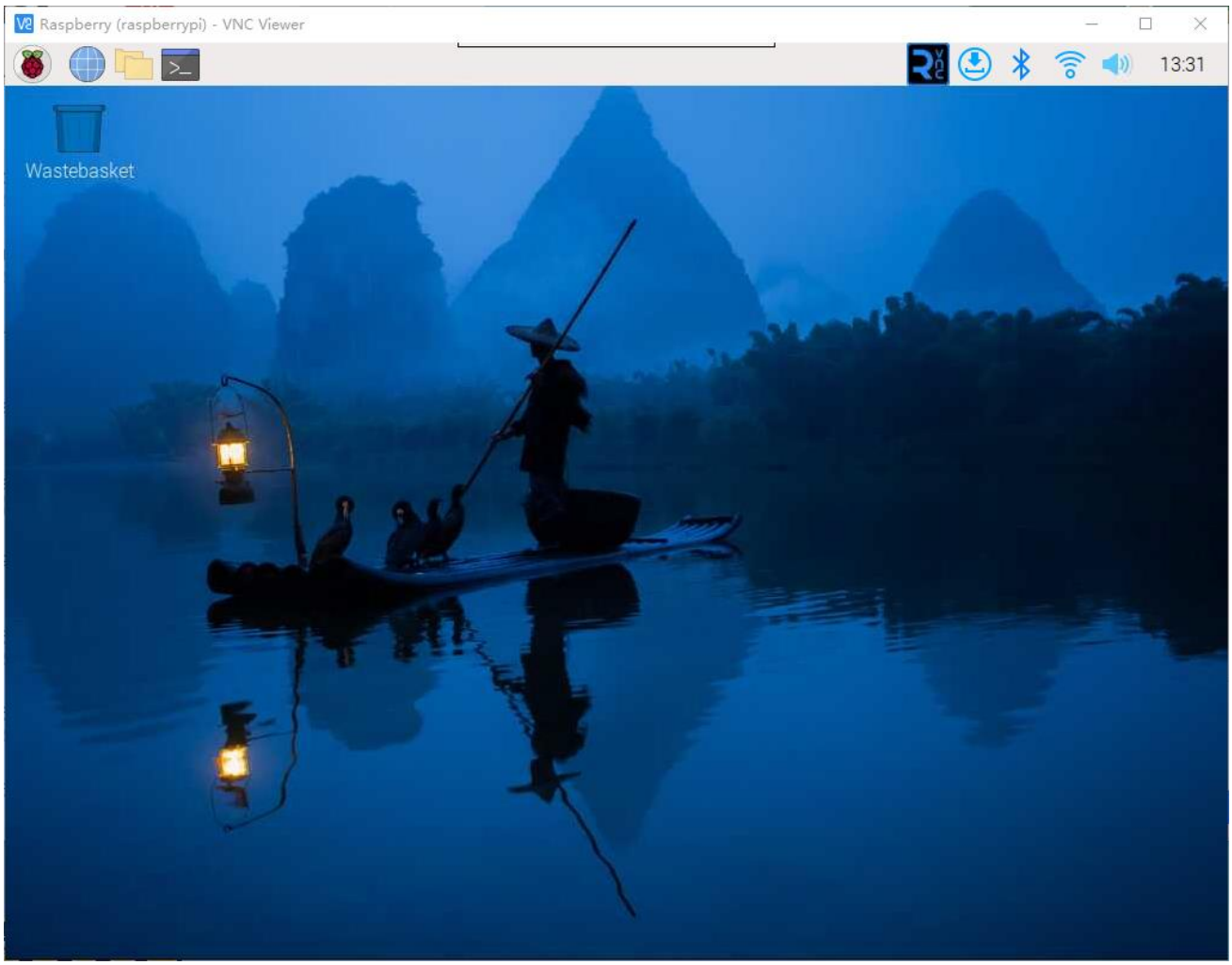
Then download and install VNC Viewer according to your computer system by clicking following link:
https://www.realvnc.com/en/connect/download/viewer/
After installation is completed, open VNC Viewer. And click File → New Connection. Then the interface is shown below.



Enter IP address of your Raspberry Pi and fill in a Name. Click OK.
Then on the VNC Viewer panel, double-click new connection you just created, and the following dialog box pops up.



Enter username: **pi** and Password: **raspberry**. Click OK.



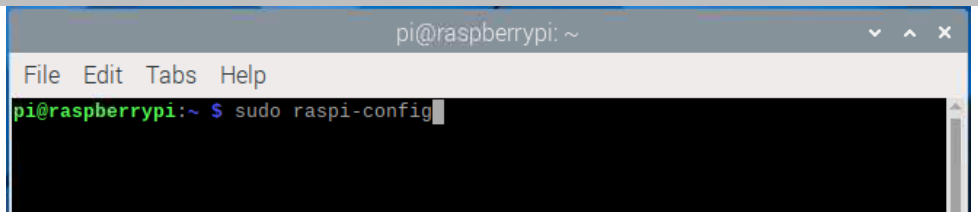
Here, you have logged in to Raspberry Pi successfully by using VNC Viewer

If the resolution ratio is not great or there is just a little window, you can set a proper resolution ratio via steps below.

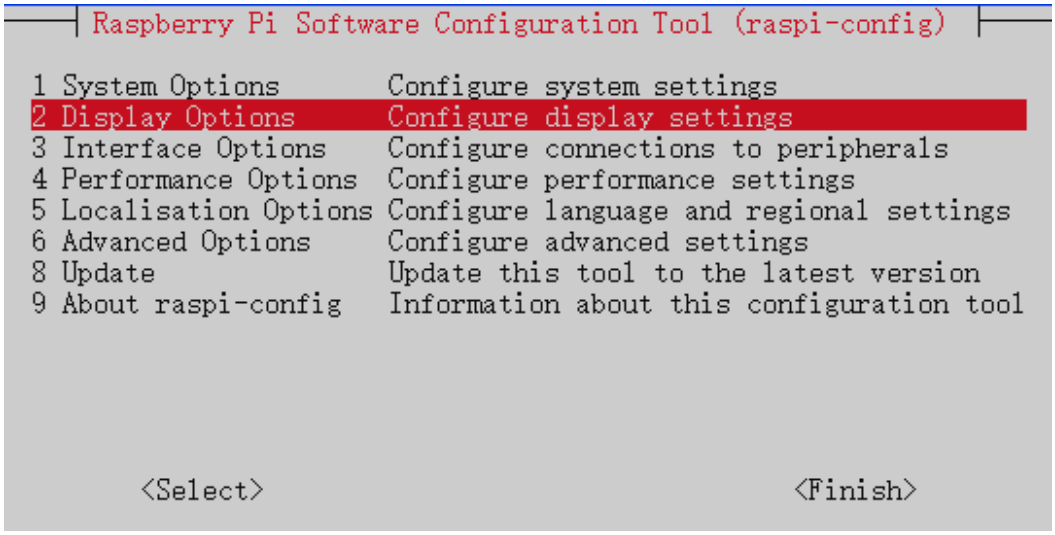
Click the Terminal icon.

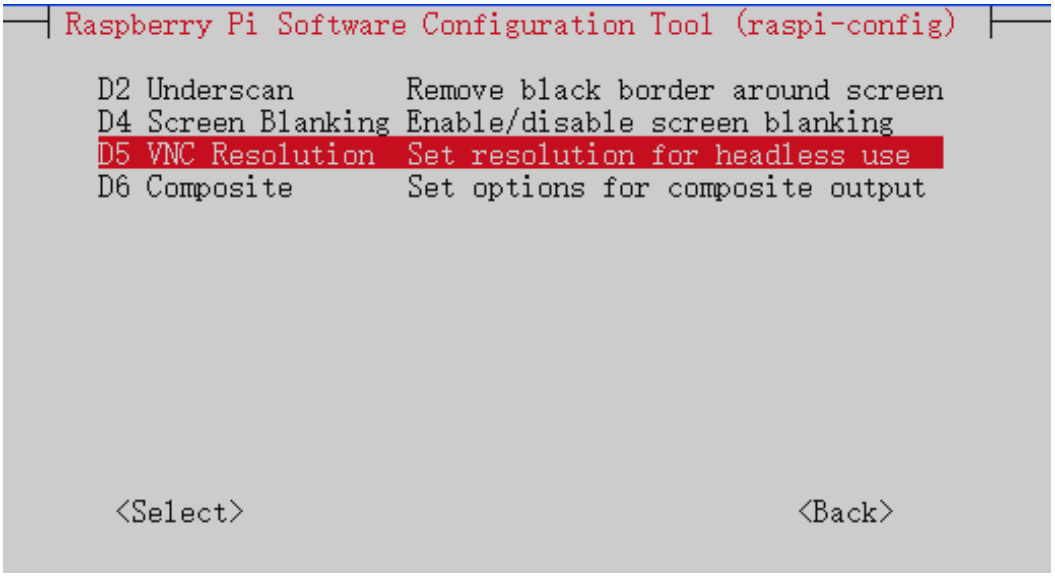
Run the command in the Terminal.

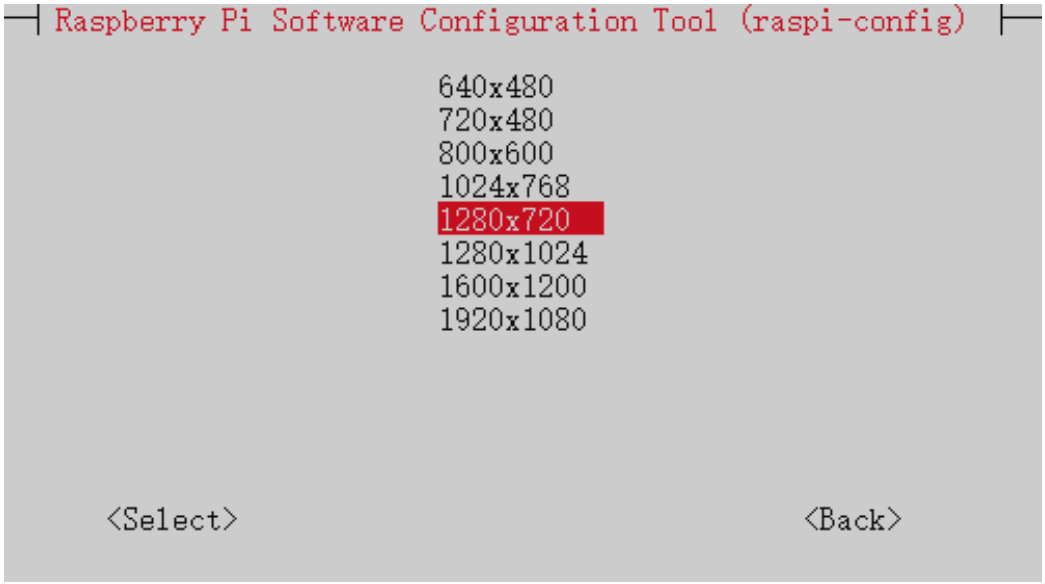
```
sudo raspi-config
```



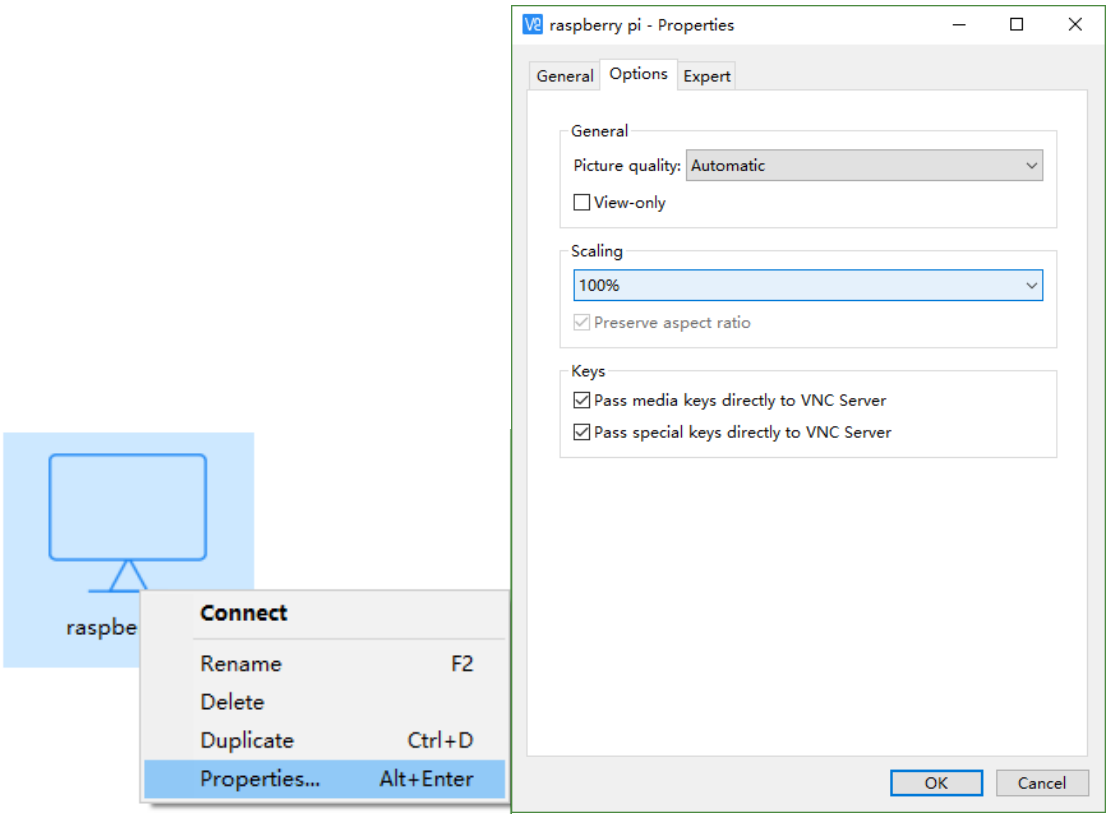
Select **Display Options→VNV Resolution→Proper resolution ratio (set by yourself) →OK→Finish→Yes**. Then reboot Raspberry Pi.







In addition, your VNC Viewer window may zoom your Raspberry Pi desktop. You can change it. On your VNC View control panel, click right key. Select Properties->Options label->Scaling. Then set proper scaling.



Here, you have logged in to Raspberry Pi successfully by using VNC Viewer and operated proper setting.

Raspberry Pi 4B/3B+/3B integrates a Wi-Fi adaptor. If you did not connect Pi to WiFi. You can connect it to wirelessly control the robot.

Chapter 1 Install Python Libraries (Required)

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

In this chapter, we will do some foundational preparation work: Start your Raspberry Pi and install some necessary libraries. And in next chapter, we will assemble the robot arm.

Note:

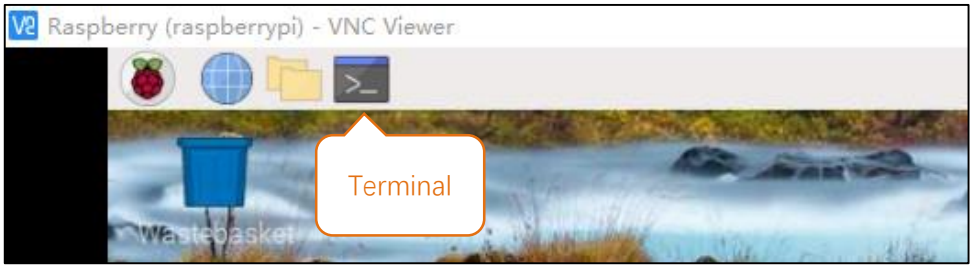
1. **Make sure Raspberry Pi OS with Desktop (64-bit) is used.**

2, The installation of libraries takes much time. **You can power Raspberry Pi with a power supply Cable**.

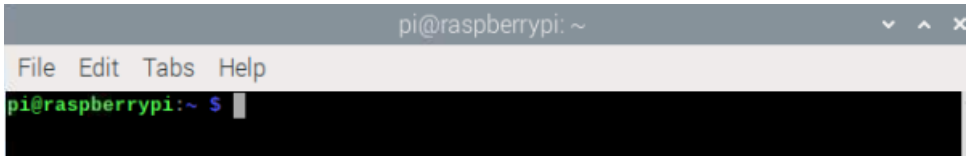
3, If you are using **remote desktop** to login Raspberry Pi, you need to use **VNC viewer**. If you use the 32-bit version, VNC viewer may not be able to use.

Step 1 Obtain the Code

Start the Raspberry Pi and open the terminal. You can click the terminal as shown below, or press "CTAL+ALT+T" on the desktop.



The terminal is shown below:



Type following command to get robot arm code and place it in user directory "Pi".

Please execute commands below one by one in turn.

```
cd ~
git clone https://github.com/Freenove/Freenove_Robot_Arm_Kit_for_Raspberry_Pi.git
```

Downloading takes much time. Please wait with patience.

You can also find and download the code by visiting our official website (http://www.freenove.com) or our GitHub repository (https://github.com/freenove).

Please note that all codes for this robot arm is written with **Python3**. If executed under python 2, errors may occur.

Set Python3 as default python

First, check the default python on your raspberry Pi. Press Ctrl-Z to exit.

```
pi@raspberrypi:~ $ python --version
```

If it is python3, you can skip this section.

If it is python2, you need to execute the following commands to set default python to python3.

1. Enter directory /usr/bin

```
cd /usr/bin
```

2. Delete the old python link.

```
sudo rm python
```

3. Create new python links to python.

```
sudo ln -s python3 python
```

4. Check python. Press Ctrl-Z to exit.

```
python –-version
```

```
pi@raspberrypi:~ $ cd /usr/bin
pi@raspberrypi:/usr/bin $ sudo rm python
pi@raspberrypi:/usr/bin $ sudo ln -s python3 python
```

If you want to set python2 as default python in **other projects**, just repeat the above command and change python3 to python2.

```
pi@raspberrypi:~ $ cd /usr/bin
pi@raspberrypi:/usr/bin $ sudo rm python
pi@raspberrypi:/usr/bin $ sudo ln -s python2 python
```

Shortcut Key

Now, we will introduce several shortcuts that are very **useful** and **commonly used** in terminal.

1. **up and down arrow keys**. History commands can be quickly brought back by using up and down arrow keys, which are very useful when you need to reuse certain commands.

When you need to type command, pressing "↑" (the Up key) will go backwards through the command history and pressing "↓" (the Down Key) will go forwards through the command history.

2. **Tab key**. The Tab key can automatically complete the command/path you want to type. When there are multiple commands/paths conforming to the already typed letters, pressing Tab key once won't have any result. And pressing Tab key again will list all the eligible options. However, when there is only one eligible option, the command/path will be completely typed as soon as you press the Tab key..

As shown below, under the '~'directory, enter the Documents directory with the "cd" command. After typing "cd D", press Tab key, there is no response. Press Tab key again, all the files/folders that begin with "D" is listed. Continue to type the character "oc", then press the Tab key, and then "Documents" is typed automatically.

```
pi@raspberrypi:~ $ cd D
Desktop/   Documents/ Downloads/
pi@raspberrypi:~ $ cd Doc
```

```
pi@raspberrypi:~ $ cd D
Desktop/   Documents/ Downloads/
pi@raspberrypi:~ $ cd Documents/
```

Step 2 Configuration

Additional supplement

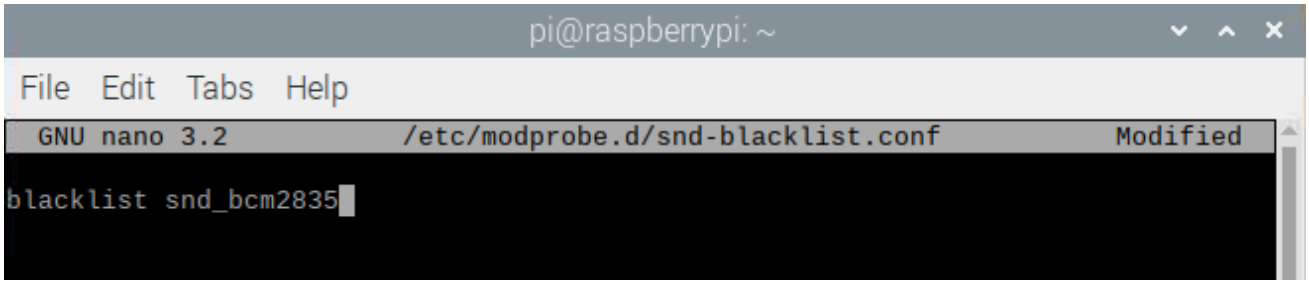
Raspbery Pi, other than 4B and 400, needs to disable the audio module; otherwise the LED will not work properly.

1. Create a new snd-blacklist.conf and open it for editing

```
sudo nano /etc/modprobe.d/snd-blacklist.conf
```

Add following content: After adding the contents, you need to press Ctrl+O, Enter, Ctrl+X.

```
blacklist snd_bcm2835
```



2. We also need to edit config file.

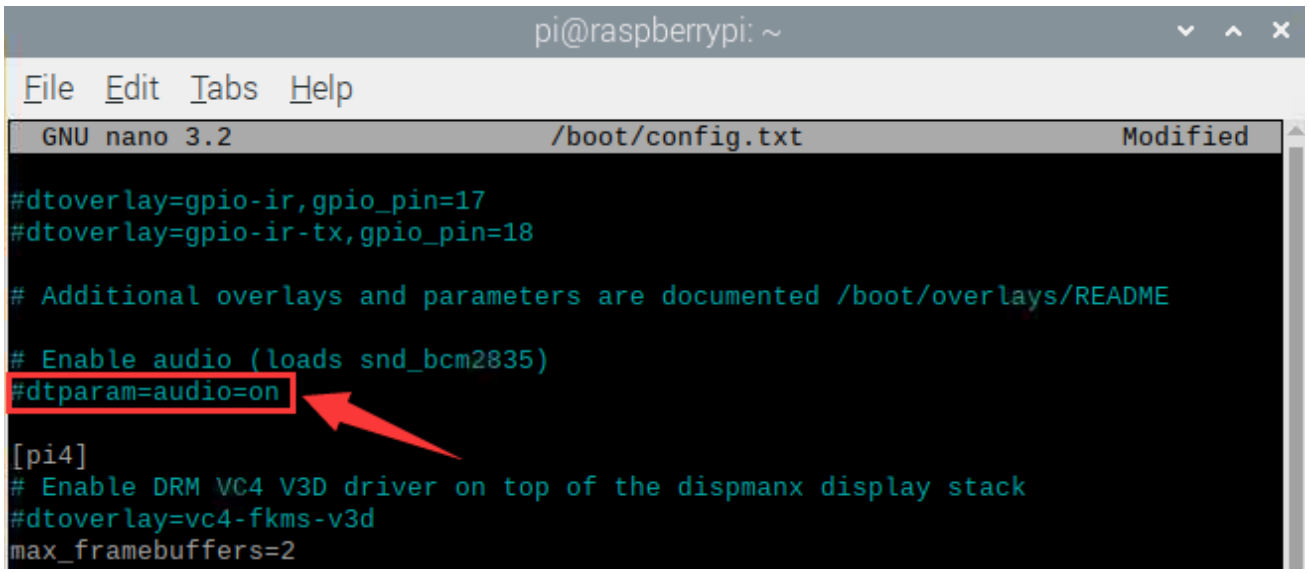
```
sudo nano /boot/config.txt
```

Find the contents of the following two lines (with Ctrl + W you can search):

```
# Enable audio (loads snd_bcm2835)
dtparam=audio=on
```

Add # to comment out the second line. Press Ctrl+O, Enter, Ctrl+X.

```
# Enable audio (loads snd_bcm2835)
# dtparam=audio=on
```



It will take effect after a reboot, and you can restart the pi after executing the next section.

If you want to restart the audio module, just restore the content modified in the above two steps.

Step 3 Run the Installation Program

1. Execute following commands to enter directory of “setup.py”.

```
cd ~/Freenove_Robot_Arm_Kit_for_Raspberry_Pi/Server
```

2. Run setup.py

```
sudo python setup.py
```

This program will automatically install the pigpio, rpi_ws281x, etc. Please **reboot** the Raspberry Pi after the installation completes, as shown below.

```
Now the installation is successful.

Please restart raspberry pi
```

After the installation completes, reboot the Raspberry Pi.

If the installation fails, please check your network and try again.

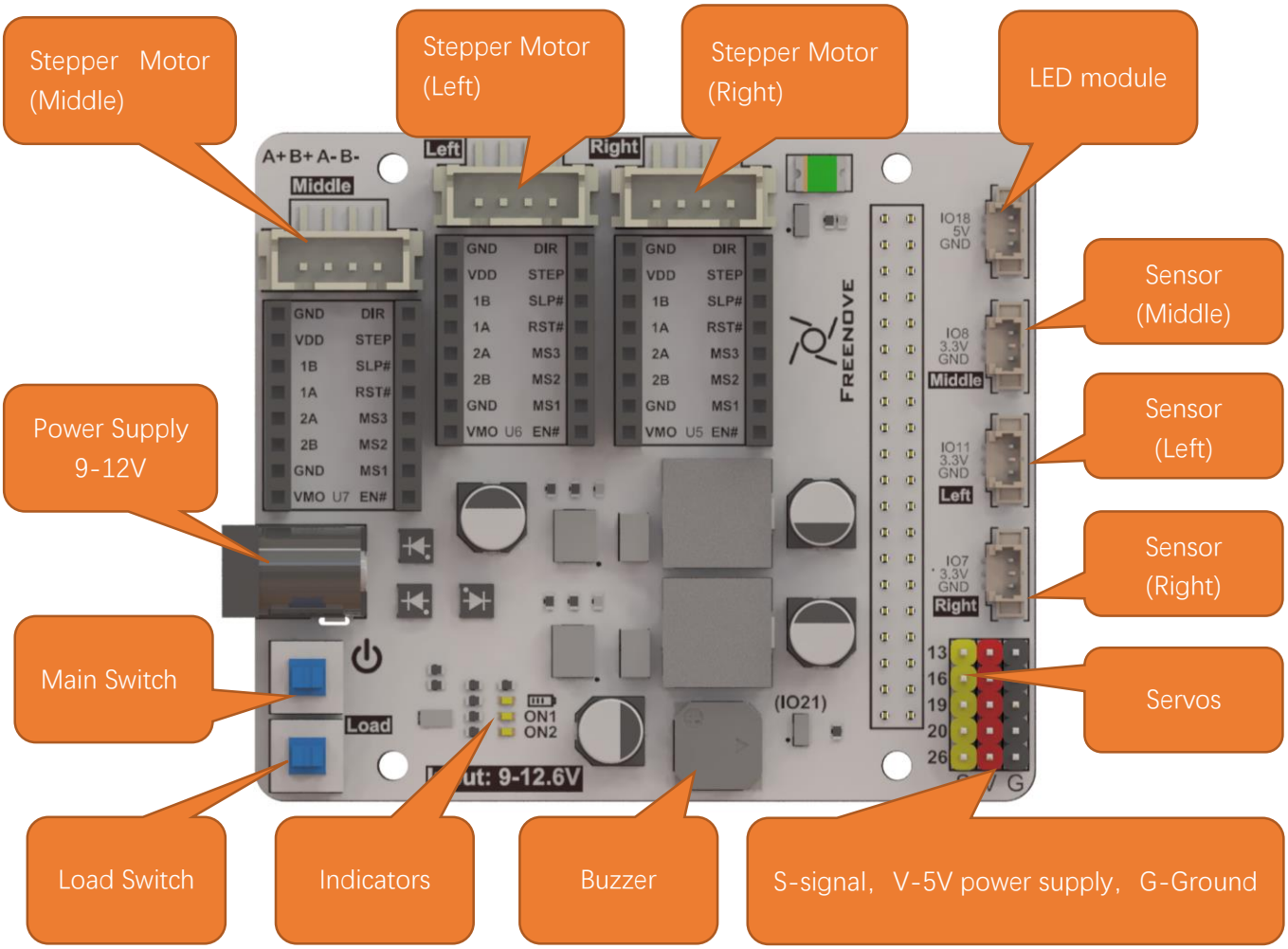
```
sudo python setup.py
```

Chapter 1 Function Tests

Robot Arm Board for Raspberry Pi

Introduction to the Robot Board

Take out the robot board as shown below from the product box.
In this chapter, we will test all modules with this robot board and the Raspberry Pi.
The board is connected to the Raspberry Pi through the IO Port on the board. The positioning holes on the board are suitable for the Raspberry Pi. The features and functions are as follows.



Installing Batteries and Robot Board

I Install batteries

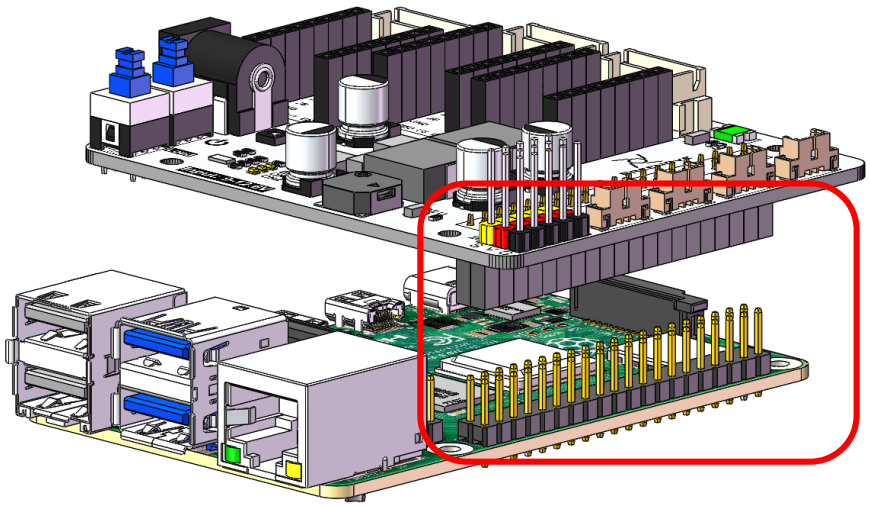
You can power the robot arm with external power supply or via the battery holder included.

- If you choose the former method, ensure that the voltage of the external supply remains within the range of 9-12.6V, with a current rating of at least 5A
- If you use the battery holder included, please prepare three flat-top Li-ion 18650 batteries.

Put the 18650 batteries into the battery holder. During installation, please ensure that the orientation of the battery aligns with the indications on the battery box, as shown in the diagram below.

II Installing the Robot Board

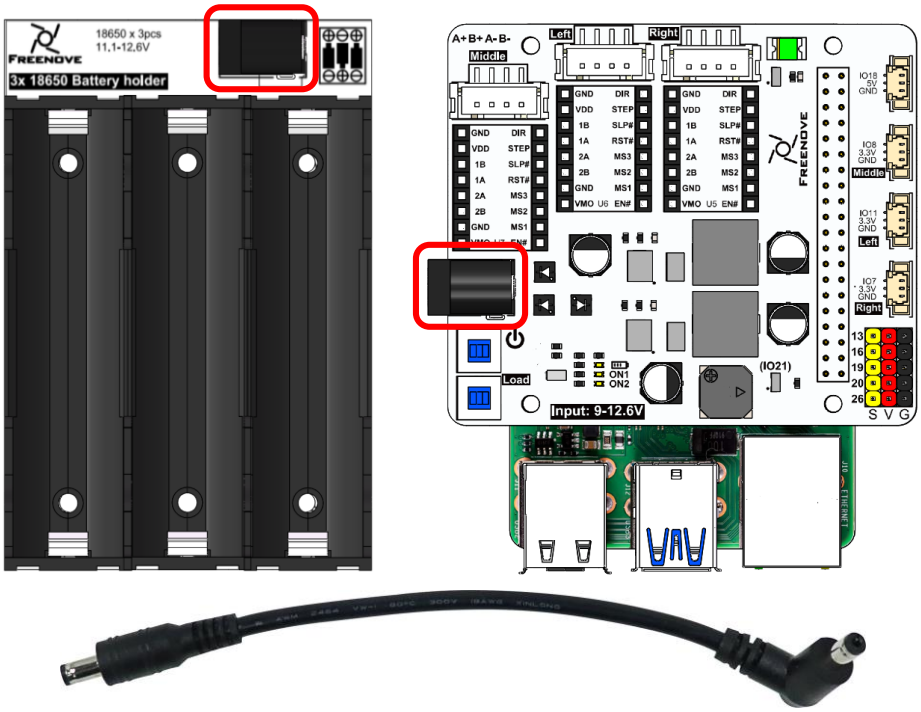
Assemble the robot board to the Raspberry Pi. Pay attention to the orientation of both boards, as shown below.



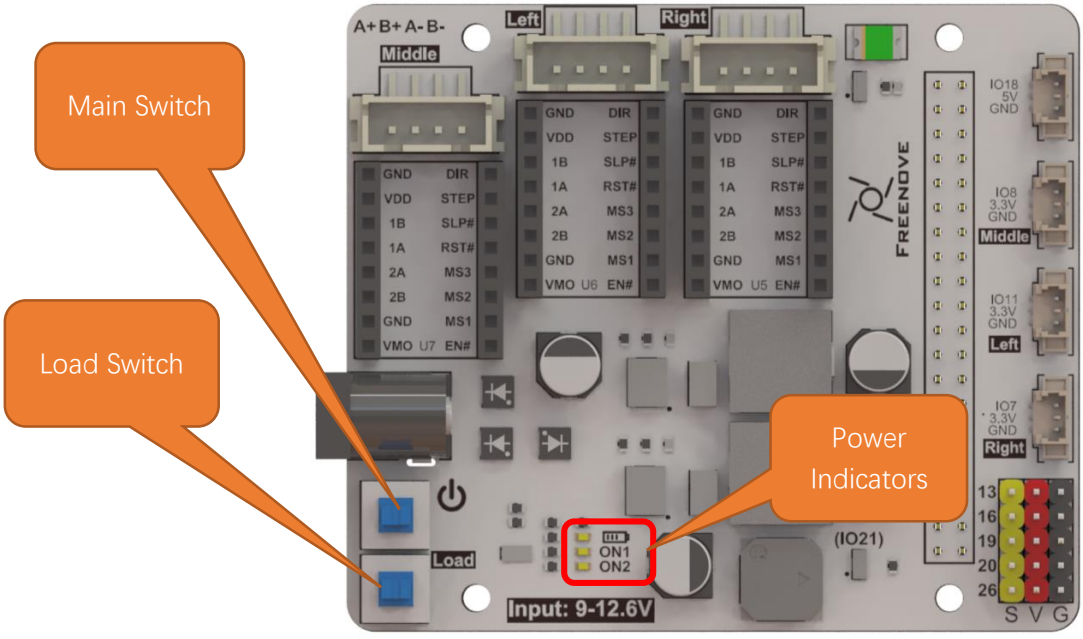
Please carefully verify that the pins of the Raspberry Pi align precisely with the female headers on the mainboard. If there is any misalignment, please reinstall them accordingly.

III Robot Board Test

Connect the robot board and the battery holder with the Dual DC power cable.



After conencting power supply, press the main switch, and you can see the power indicator ON1 light up. Then it will power the Raspberry Pi. The robot board also powers the LED module, buzzer and sensors.



Press the Load switch and ON2 will light up. It powers the A4988 moduld and the motors.

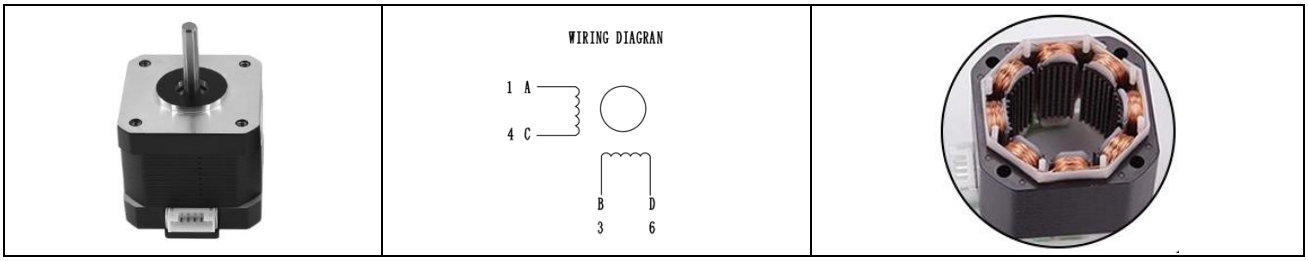
As you can see, there is another indicator above ON1. **It will light off when the total voltage of the three Li-ion batteries is lower than 10.5V, indicating a lower power supply. At this point, please charge the batteries.**

Stepper Motor Test

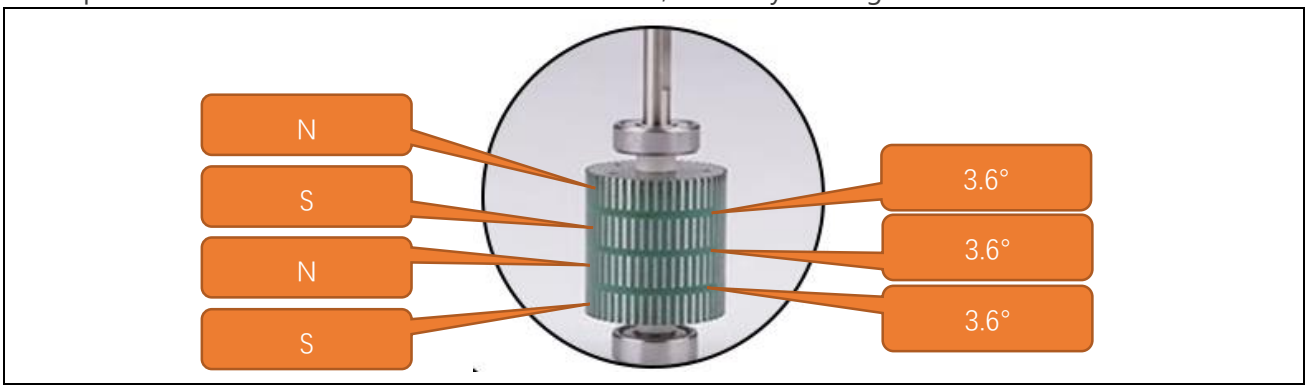
If you are not interested in the principles of stepper motors and their driving modules, feel free to skip ahead to the debugging section.

Introduction to the Stepper Motors

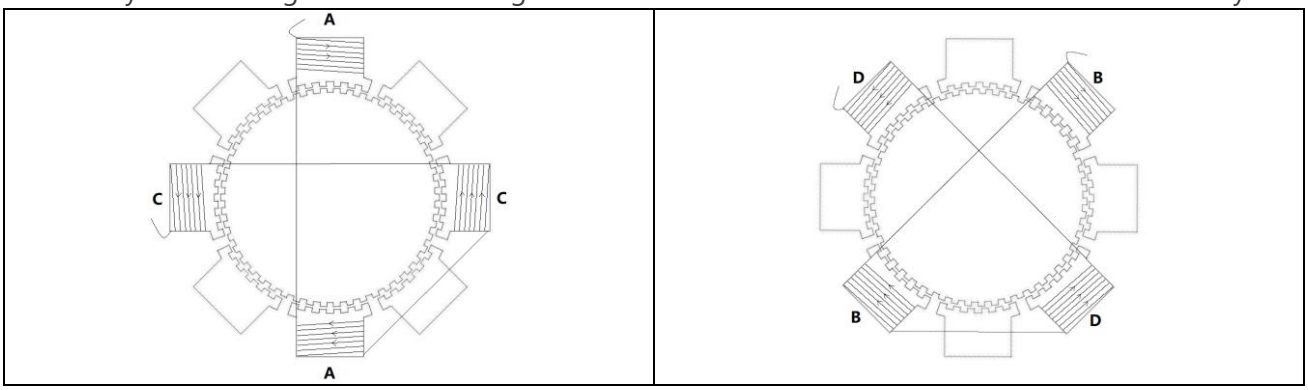
The main body of the stepper motor consists of a stator and a rotor.
The stator is composed of eight stator windings, which are divided into four groups, A, B, C, and D. Among them, groups A and C are connected together and the same goes to B and D.



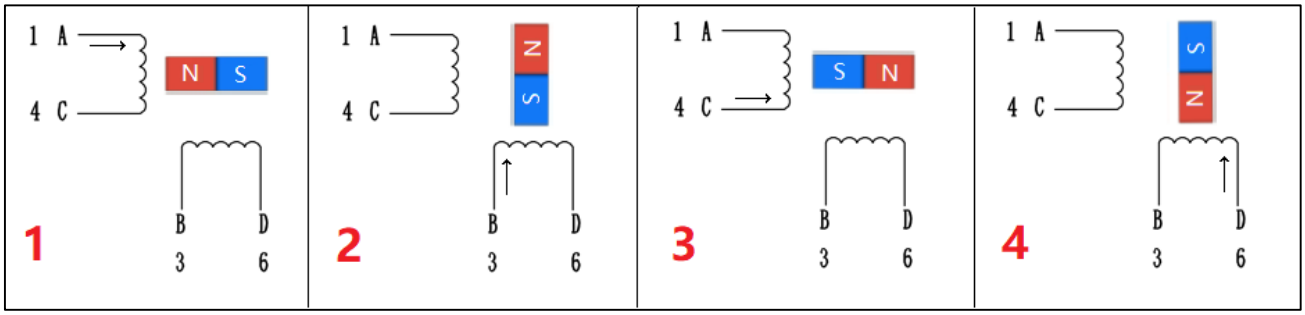
The rotor encompasses four sets of permanent magnets, with the N and S poles arranged alternately. These poles rotate around the axis of the motor rotor, offset by 3.6 degrees from each other.



Phase A and C together form a single phase in the motor system. When energized, Phase A generates a North pole electromagnetic field directed towards the motor rotor, while Phase C generates a corresponding South pole electromagnetic field. Alternatively, if Phase A produces a South pole electromagnetic field towards the motor rotor, Phase C will generate a North pole electromagnetic field towards the motor rotor. Similarly, Phases B and D operate in a similar manner to Phases A and C, collectively contributing to the electromagnetic fields that drive the motor's motion and functionality.



As shown in the above diagram, A and C form one set of motor windings, while B and D form another set. If you want the magnet to make one full rotation, pay attention to the direction of the current flow in the motor windings below.



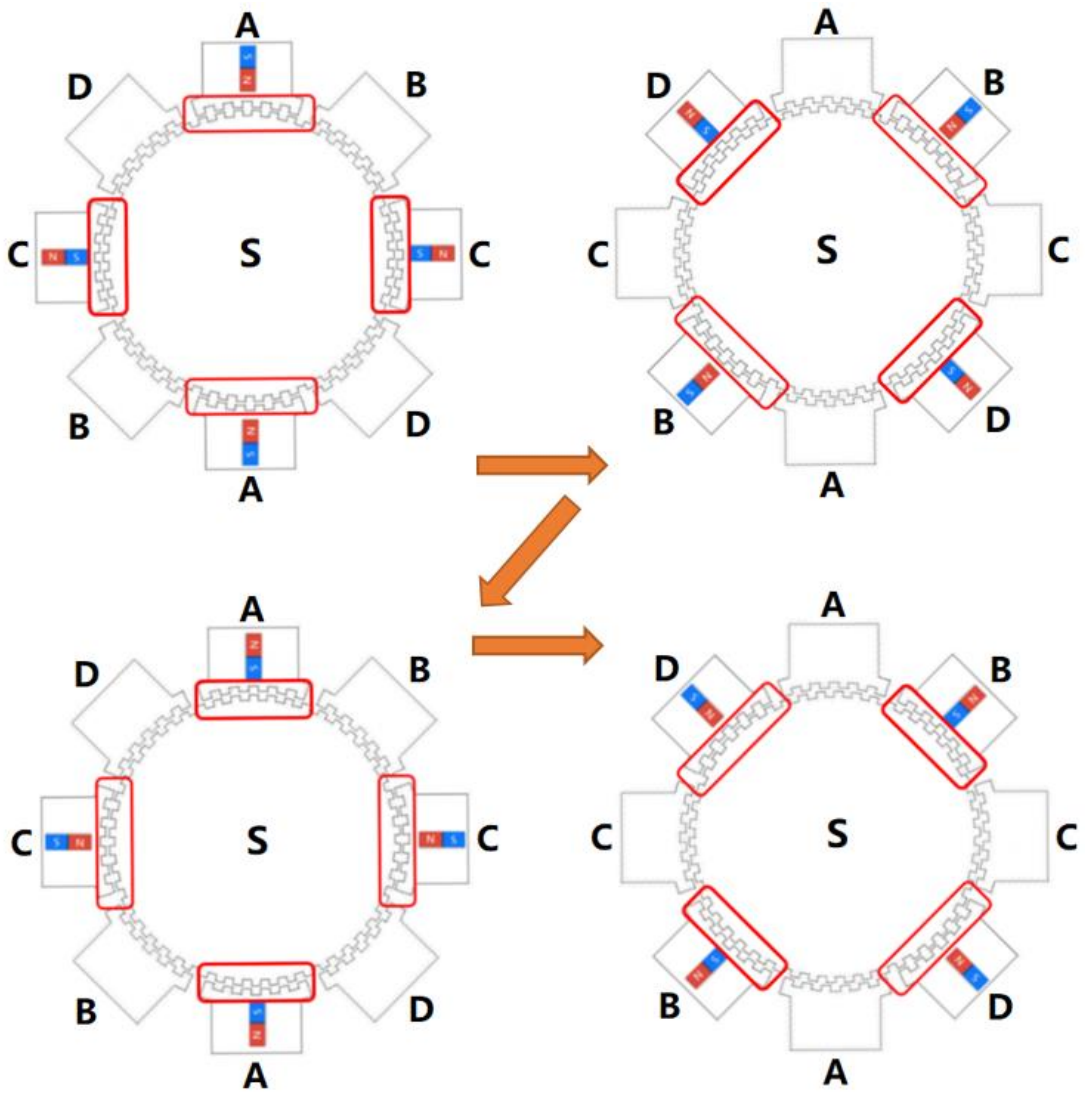
Observing from the S pole of the motor rotor, we proceed with the following steps:

Step 1: Apply positive voltage to A, negative to C, and leave B and D unpowered.
- A generates an N pole magnetic field toward the motor rotor, attracting the rotor gears to face the gears of group A.
- C generates a magnetic field with a South pole directed towards the motor rotor, causing repulsion between the rotor gears and the C gears, offsetting each other.
- B and D do not produce a magnetic field, thus not interfering with the rotation of the motor rotor.

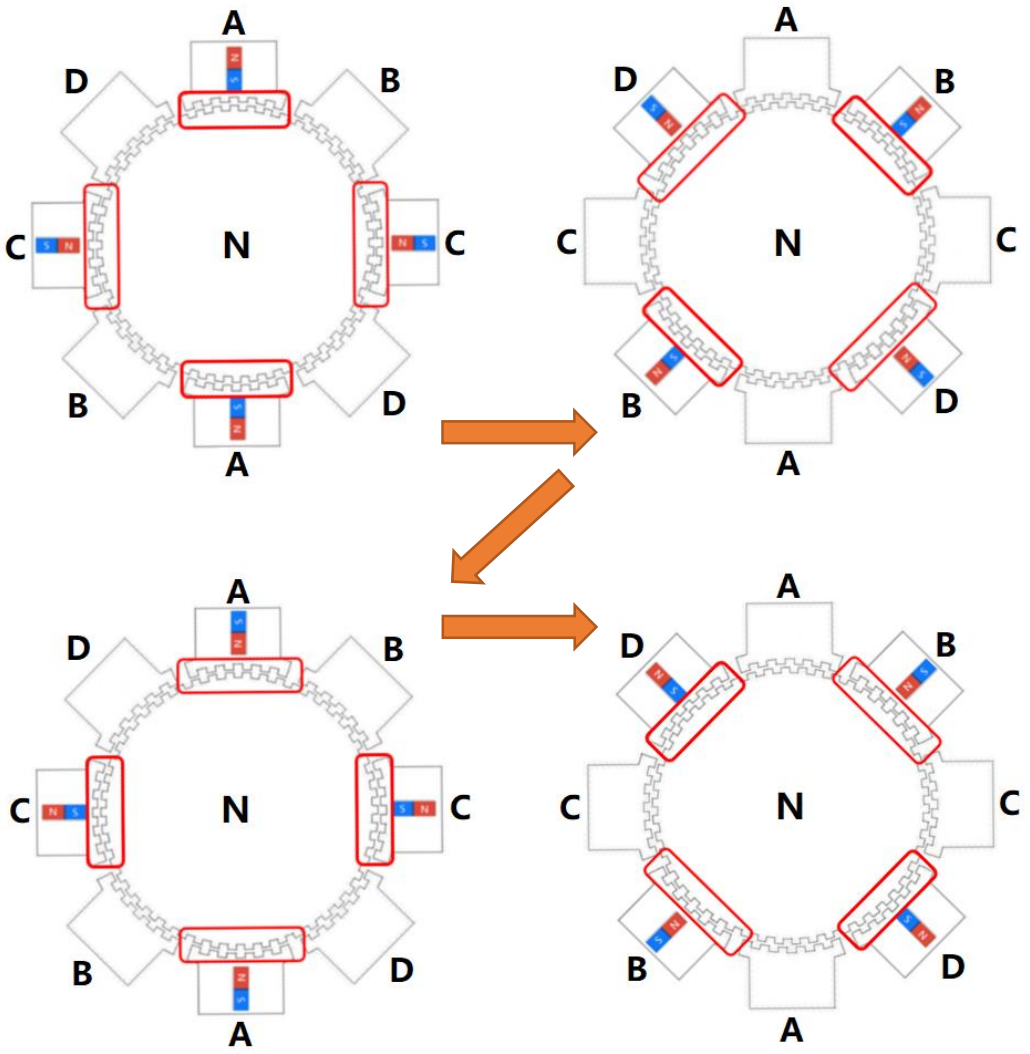
Step 2: Apply positive voltage to B, negative to D, and leave A and C unpowered.
- B generates an N pole magnetic field toward the motor rotor, attracting the rotor gears to face the gears of group B.
- D generates a magnetic field with a South pole directed towards the motor rotor, causing repulsion between the rotor gears and the D gears, offsetting each other.
- A and C do not produce a magnetic field, thus not interfering with the rotation of the motor rotor.

Step 3: Apply positive voltage to C, negative to A, and leave B and D unpowered.
- C generates an N pole magnetic field toward the motor rotor, attracting the rotor gears to face the gears of group C.
- A generates a magnetic field with a South pole directed towards the motor rotor, causing repulsion between the rotor gears and the A gears, offsetting each other.
- B and D do not produce a magnetic field, thus not interfering with the rotation of the motor rotor.

Step 4: Apply positive voltage to D, negative to B, and leave A and C unpowered.
- D generates an N pole magnetic field toward the motor rotor, attracting the rotor gears to face the gears of group D.
- B generates a magnetic field with a South pole directed towards the motor rotor, causing repulsion between the rotor gears and the B gears, offsetting each other.
- A and C do not produce a magnetic field, thus not interfering with the rotation of the motor rotor.



Similarly, when observing from the motor rotor's N pole, careful examination of the gear's condition reveals that it exhibits attraction and repulsion in the opposite direction compared to the S pole. This characteristic enhances the uniformity of torque in the stepper motor.



With each iteration of this cycle, the stepper motor's rotor rotates clockwise by a distance of one tooth. The motor rotor comprises 50 teeth, with each tooth spanning an angle of 360°/50 = 7.2°. Therefore, in the aforementioned steps, with each step executed, the motor rotates by 7.2°/4 = 1.8°.
To complete one full rotation of the motor, it would require 360°/1.8° = 200 steps.

The direction of motor rotation depends on whether we execute the cycle steps in the forward or reverse order.

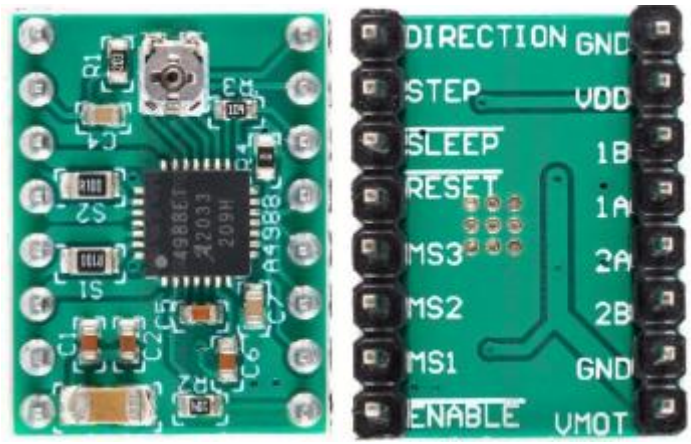
Following the steps from top to bottom, if we sequentially energize the motor's A, B, C, D phases, the stepper motor will rotate in one direction. Conversely, following the steps from bottom to top and sequentially energizing the motor's A, B, C, D phases will result in the stepper motor rotating in the opposite direction.

CW	CCW	A	B	C	D
↓	↑	+		-	
			+		-
		-		+	
			-		+

Description of the Stepper Motor Driver Module

In our daily life, stepper motors are commonly found in a variety of devices such as 3D printers and typewriters, showcasing their remarkable precision and versatility. However, as previously discussed, merely controlling the current direction of a stepper motor results in a limited precision of 1.8°, which contradicts our understanding of its capabilities.

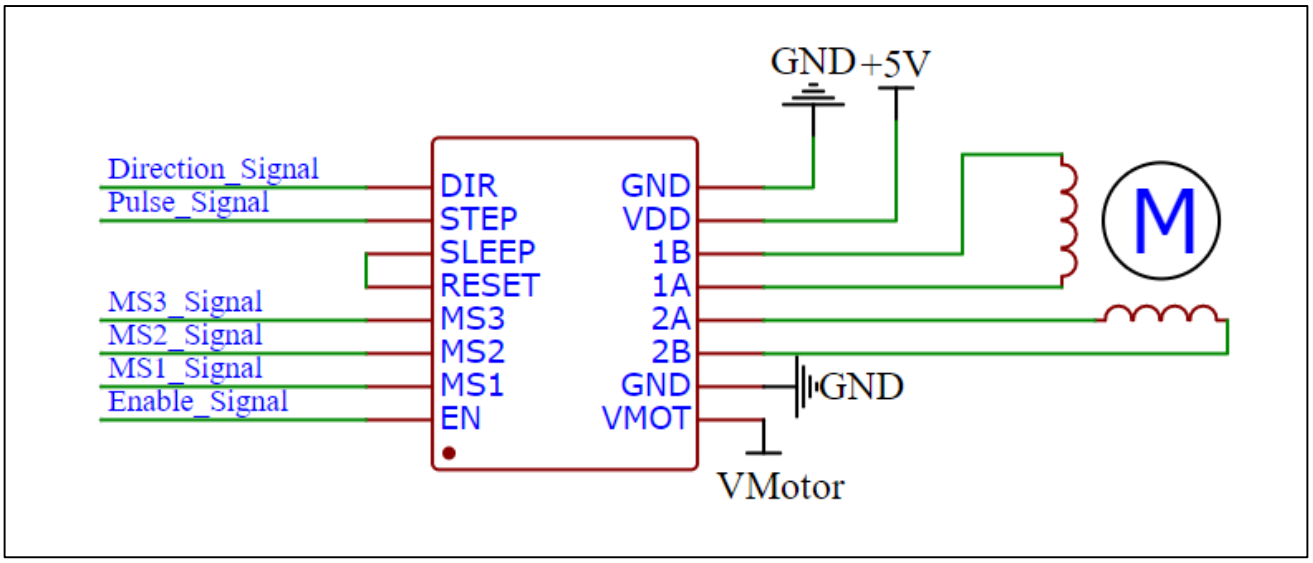
This section delves into the principles and applications of stepper motor driver modules, elucidating how they contribute to enhancing stepper motor precision. Within our robotic arm system, we rely on the A4988 module as the stepper motor driver. This module boasts a wide voltage input range of 8-35V and can deliver a maximum current output of 2A.



Below is a brief introduction to each pin of the stepper motor driver module:

Pins	Description	Pin	Description
GND	Ground pin, providing the reference voltage for the module.	DIR	This determines the direction of rotation of the motor.
VDD	This pin supplies power to the internal logic of the module. (3.3-5V)	STEP	Pulses input pin.
1B	This pin connects the stepper motor coils (4-C) A-	SLEEP#	Putting this pin at logic low puts the driver into a low-power sleep mode
1A	This pin connects the stepper motor coils (1-A) A+	RESET#	Reset pin, valid in low
2A	This pin connects the stepper motor coils (2-B) B+	MS3	These pins set the microstep resolution of the driver.
2B	This pin connects the stepper motor coils (6-D) B-	MS2	
GND	Negative of external load power supply	MS1	
VMOT	Positive of external load power supply. (9-12.6V)	ENABLE#	This pin enables or disables the outputs of the stepper motor coils.

The connection of the A4988 driver is as shown below:

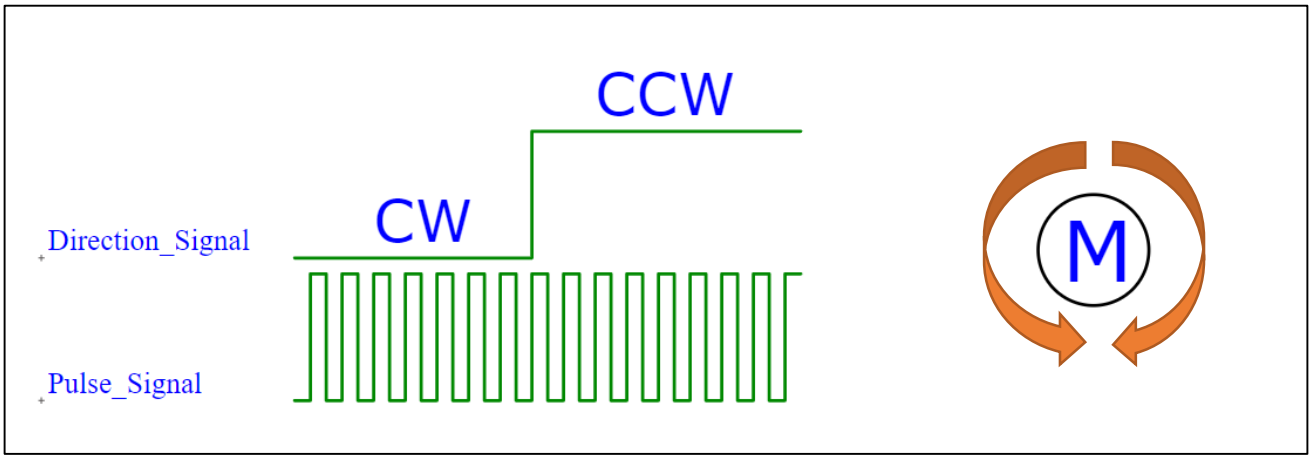


The A4988 module can enable or disable a stepper motor by controlling the level of the Enable_Signal pin. When the Enable_Signal pin is set to high, the A4988 module will block the output current for 1A, 1B, 2A, and 2B, resulting in the stepper motor losing current and the internal coils losing their magnetic field. This lack of magnetic field constraint allows the motor rotor to rotate freely.

Conversely, when the Enable_Signal is set to low, the A4988 module will activate the output current for 1A, 1B, 2A, and 2B. This provides the stepper motor with current, causing the internal coils to generate a magnetic field that holds the motor rotor in place, preventing free rotation.

The driver can also determine the rotation direction of the stepper modules by controlling the level of the Direction_Signal pin.

- When the Direction_Signal is at a high level, inputting pulse signals via the Pulse_Signal will cause the stepper motor to rotate in one direction.
- When the Direction_Signal is at a low level, inputting pulse signals via the Pulse_Signal will cause the stepper motor to rotate in the opposite direction.



If we want the motor to rotate once, we need to know the microstepping resolution of A4988 first.

MS1	MS2	MS3	Microstep Resolution	Excitation Mode	Number of pulses per revolution	Resolution of each step
L	L	L	Full Step	2 Phase	200	1.8°
H	L	L	Half Step	1-2 Phase	400	0.9°
L	H	L	Quarter Step	W1-2 Phase	800	0.45°
H	H	L	Eighth Step	2W1-2 Phase	1600	0.225°
H	H	H	Sixteenth Step	4W1-2 Phase	3200	0.1125°

For instance, if we intend to rotate the stepper motor one full revolution, as illustrated in the left diagram below:

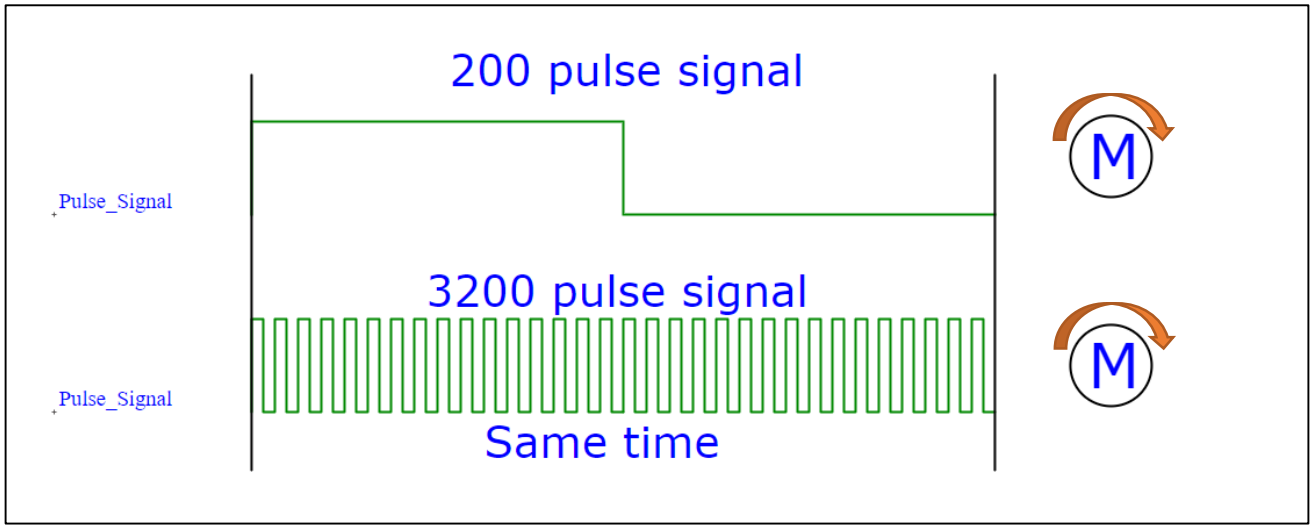
- When MS1, MS2, and MS3 are all set to a low level, it requires 200 pulse signals from the Pulse_Signal input.
- When MS1, MS2, and MS3 are all set to a high level, it necessitates 3200 pulse signals from the Pulse_Signal input.

Expanding on this, we can regulate the frequency of the Pulse_Signal to adjust the motor's rotational speed, as shown in the right diagram below:

For example, if we aim to complete one revolution of the stepper motor within 1 second:

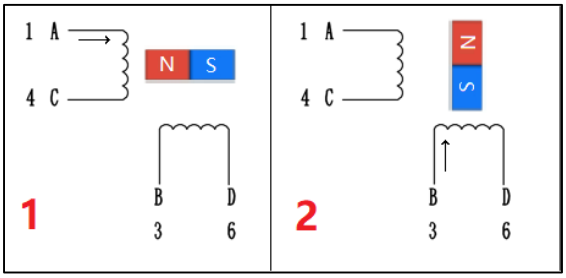
- When MS1, MS2, and MS3 are all set to a low level, we need to input 200 pulse signals via the Pulse_Signal within 1 second. Thus, the frequency (F) of the Pulse_Signal is 200Hz. Each pulse has a period (T) of 1s/200Hz = 0.005s, equating to 5 milliseconds per pulse.

- When MS1, MS2, and MS3 are all set to a high level, we need to input 3200 pulse signals via the Pulse_Signal within 1 second. Hence, the frequency (F) of the Pulse_Signal is 3200Hz. Each pulse has a period (T) of 1s/3200Hz = 0.0003125s, corresponding to 0.3125 milliseconds per pulse.



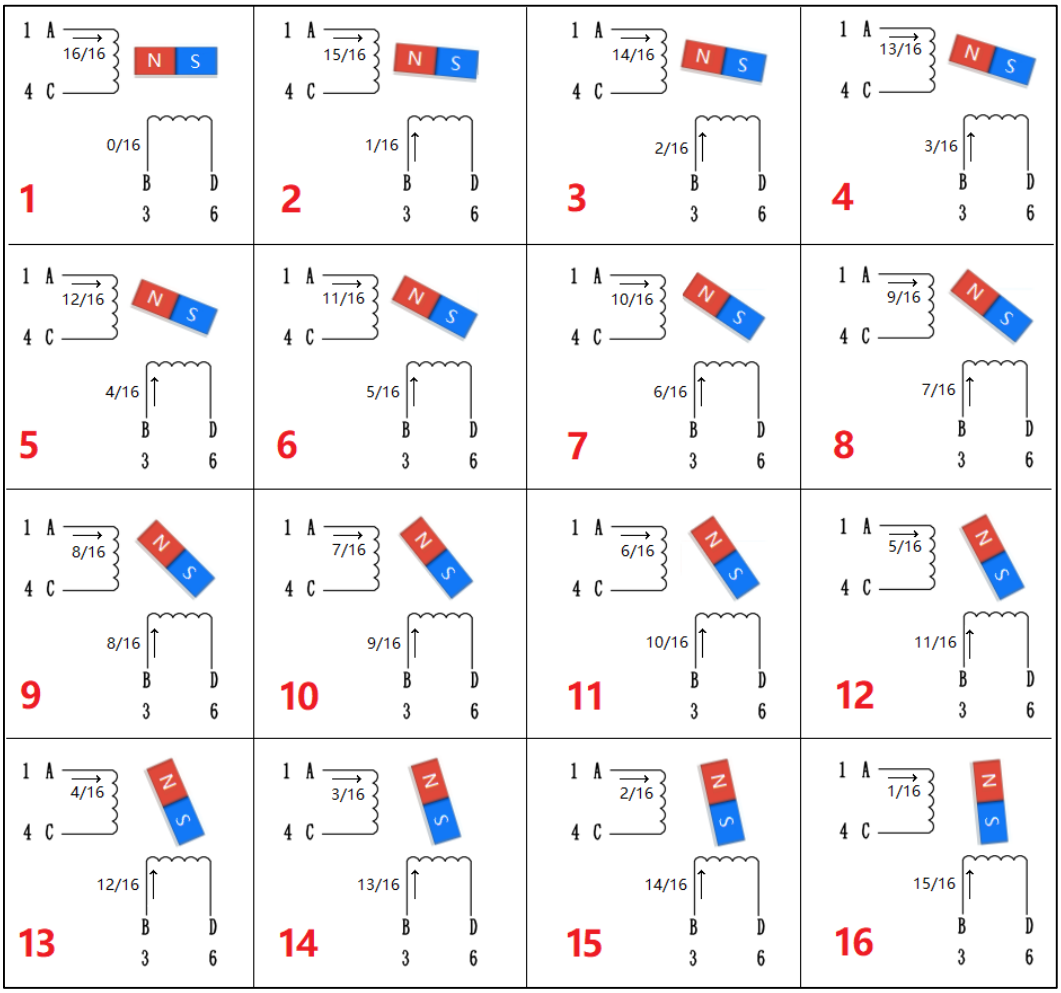
The A4988 module has the capability to regulate the output current of the stepper motor connection terminals, thereby enhancing the precision of the stepper motor's rotational angle.

For instance, when MS1, MS2, and MS3 are all set to a low level, assuming that terminals A and C are energized while B and D are not, sending a single pulse signal to the A4988 module will result in a change in the output terminals of the stepper motor. The updated state would see terminals A and C unpowered, while B and D remain powered, as depicted in the diagram below.



At this point, the rotor of the motor moves a distance equivalent to 1/4 of a gear tooth. The precision of the motor's rotation angle is calculated as 360°/50 teeth/4 = 1.8° per step.

When MS1, MS2, and MS3 are all set to a high level, the A4988 module controls the output current values of terminals A, C, and B, D by receiving 16 pulse signals from the Pulse_Signal each time, as illustrated in the diagram below.

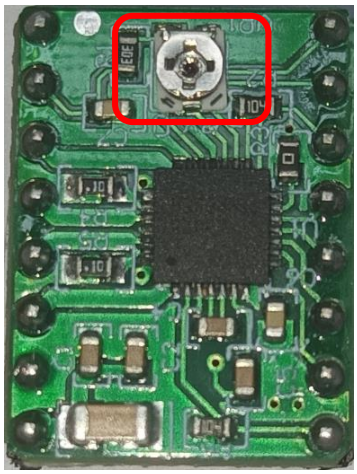


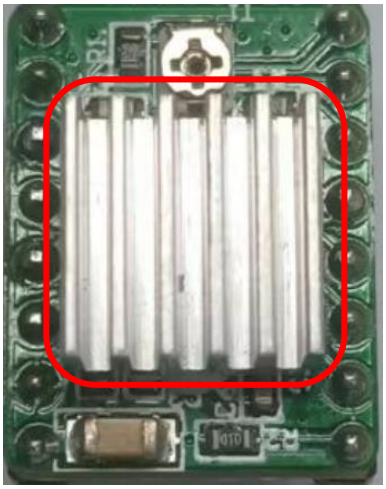
When MS1, MS2, and MS3 are all set to a high level, it requires 16 pulses to move the motor rotor a distance equivalent to 1/4 of a gear tooth. The precision of the motor's rotation angle is calculated as 360°/50 teeth/4/16 = 0.1125° per step.

Adjust the Stepper Motor Driver Module

In our circuit board setup, the A4988 module is powered via the DC interface. We recommend using an external power supply with a voltage ranging from 9 to 12.6V.

The A4988 module can deliver a maximum current output of around 2A, whereas the motor we provide is rated at 1.5A per phase. Before proceeding with debugging, it's crucial to inspect the adjustable resistor on the A4988 module. The default configuration is illustrated in the diagram below.

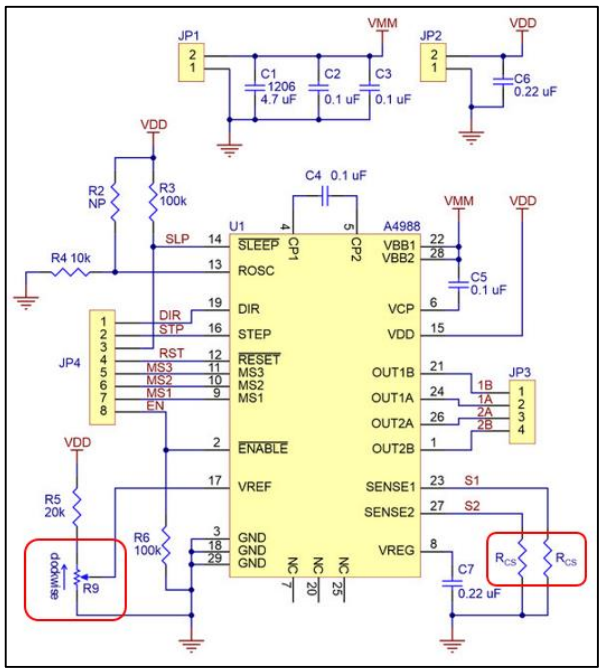




In its default state, the adjustable resistor is connected to the Vref pin of the A4988 chip, allowing control of the maximum output current of the A4988 module by adjusting the voltage level at Vref.

Before use, check whether the A4988 module has already been fitted with a heatsink, as shown in the diagram above. If not, locate the heatsink and remove the adhesive backing before affixing it to the A4988 module as depicted. If the heatsink is not attached, it is advisable to limit the A4988 module's output current to below 1.2A to prevent potential damage caused by overheating.

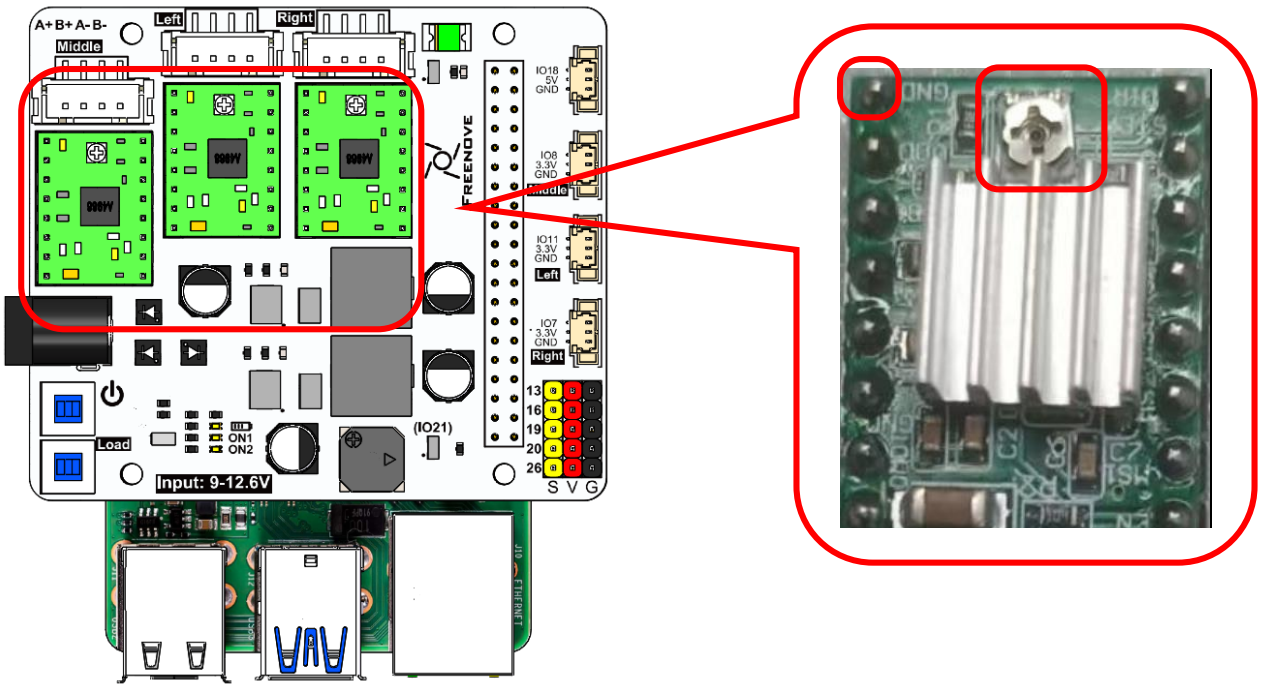
Turning the adjustable resistor clockwise increases the voltage level, thereby raising the maximum output current of the A4988 module.
Conversely, rotating the adjustable resistor counterclockwise decreases the voltage level, resulting in a reduction of the A4988 module's maximum output current.



The formula to calculate the reference voltage (Vref) for the A4988 module is: Vref = Imax * Rcs * 8. In our A4988 module, the resistance value (Rcs) for each module is 0.1 ohms. The calculation formula for the maximum output current (Imax) can be simplified as Imax = Vref / 0.8.

Mount the three stepper motor driver modules onto the Robot Arm Board. Be cautious about the orientation of the stepper motor driver modules to prevent incorrect insertion, as it could harm the circuit board. Avoid connecting the stepper motors at this stage.

Once installed, utilize a screwdriver to **clockwise rotate** the resistors on the three stepper motor driver modules to the position illustrated in the diagram below. The Vref value at this position should range between 0.7V and 0.8V. The A4988 module can deliver a current of 0.875-1A.



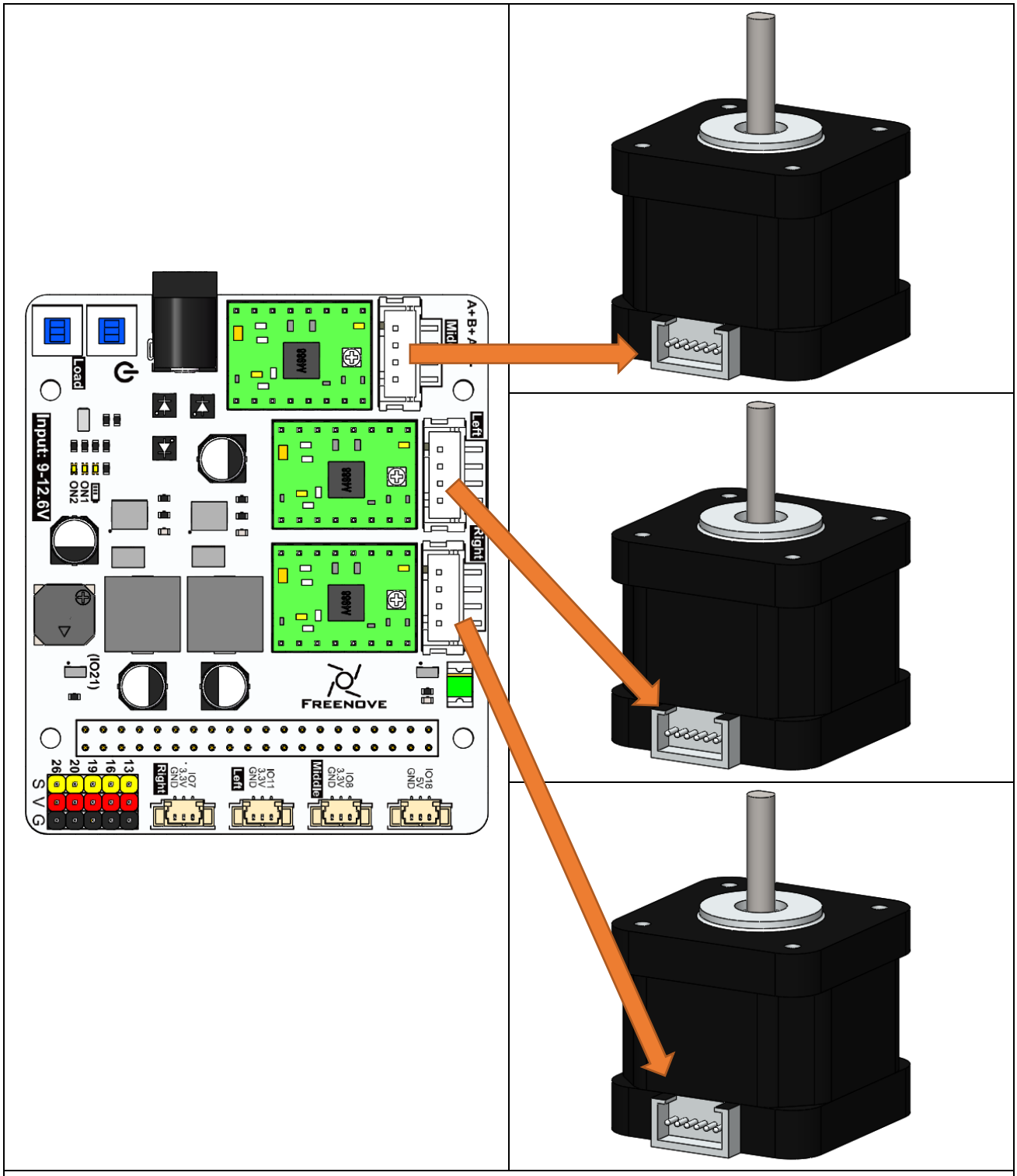
The stepper motor has a maximum rated current of 1.5A per phase. According to the formula Imax = Vref / 0.8, the Vref value should not exceed 1.2V to prevent potential damage to the stepper motor. We recommend maintaining the Vref value between 0.7V and 1V.

If you have a voltmeter available, you can place the negative probe on the GND pin of the A4988 module and the positive probe on the adjustable resistor. By adjusting the resistor, you can precisely tune the Vref value, ensuring optimal performance of the A4988 module.

Please note:
1. If the A4988 modules do not have heatsink attached, the maximum output current should not exceed 1.2A. Running at high currents for prolonged periods without a heatsink can cause the chip to overheat and potentially damage itself.
2. After continuous operation of the A4988 module and the motor for some time, the temperature of the module typically ranges between 40°C and 50°C. Avoid touching the A4988 module with your hands as it may be hot.

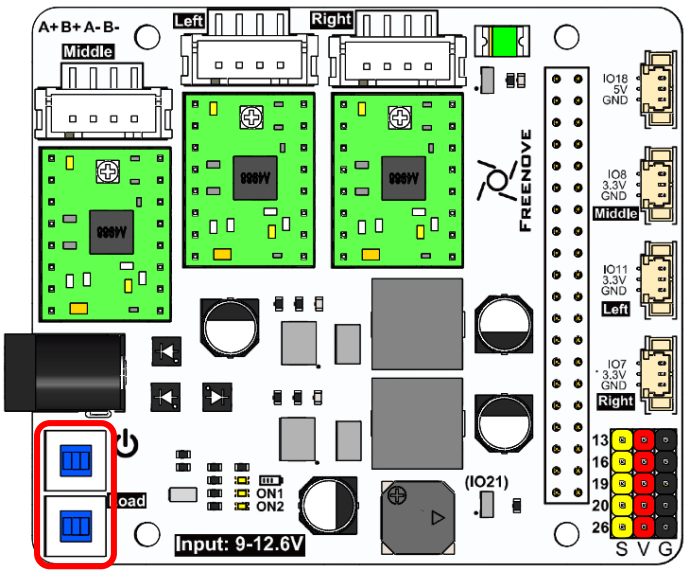
Running Code to Test Stepper Motor

1. Connect the the stepper motors to the robot board with cables come with this kit. (Power cable is not shown in the diagrams below.)



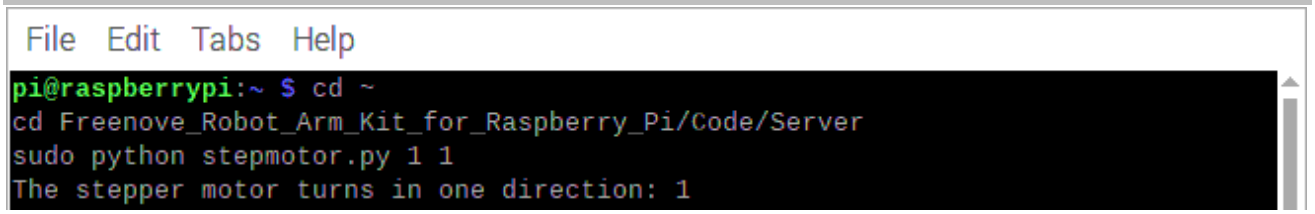
Cables for stepper motor

2. Turn ON the power switches.

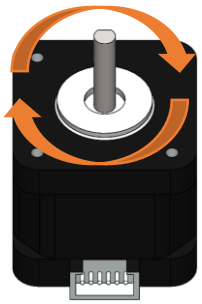


3. Open the Terminal on Raspberry Pi, type the following commands and press Enter.

```
cd ~
cd Freenove_Robot_Arm_Kit_for_Raspberry_Pi/Server/Code
sudo python stepmotor.py 1 1
```



4. The stepper motor should rotate continuously in the clockwise direction. **Please note that if the motor rotates in the counterclockwise direction, please halt the assembly of the robotic arm and contact us via email at: support@freenove.com**.



5. Here, we're appending parameters to the command. The first parameter indicates the motor number, ranging from 1 to 3. The second parameter denotes the motor's direction of rotation. When set to 1, the motor rotates clockwise; when set to 0, the motor rotates counterclockwise.

You can try the following commands

Commands	Descriptions
sudo python stepmotor.py 1 0	Contorol Stepper Motor 1 to rotate in the counterclockwise direction
sudo python stepmotor.py 2 1	Contorol Stepper Motor 2 to rotate in the clockwise direction
sudo python stepmotor.py 3 0	Contorol Stepper Motor 3 to rotate in the counterclockwise direction

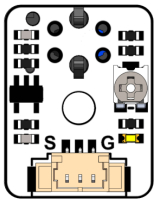
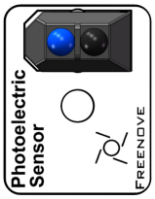
6. Press Ctrl+C to exit the program.

Infrared Sensor Test

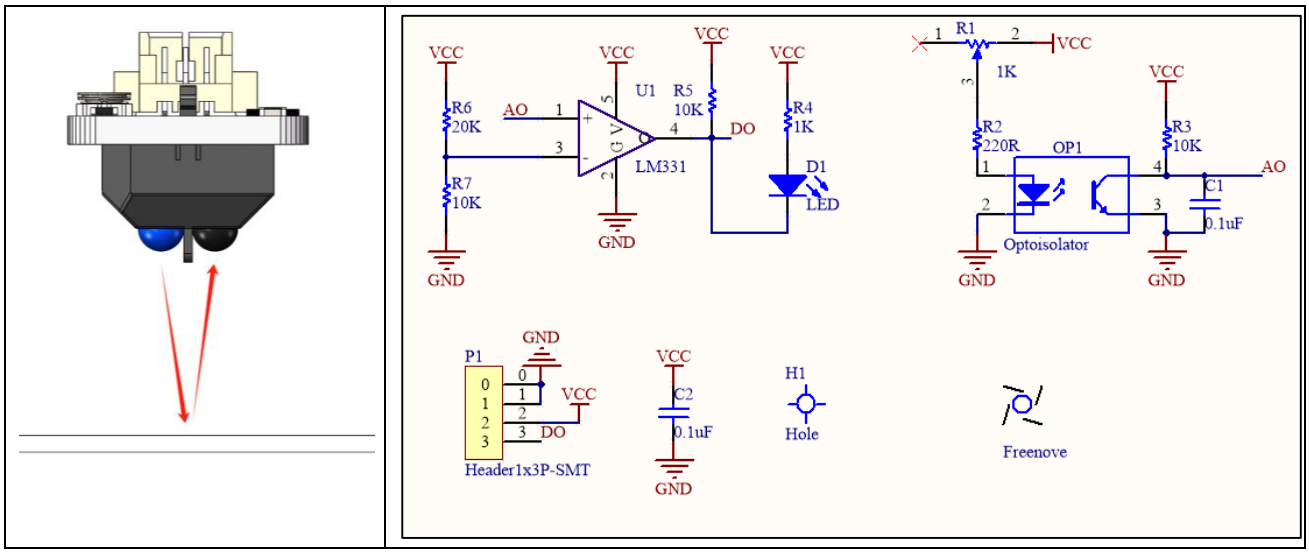
Whenever the power supply to the stepper motor is interrupted and subsequently restored, the Raspberry Pi system loses track of the motor's current angular position. To mitigate this issue, we have implemented a solution utilizing an infrared sensor. This sensor serves to accurately detect and record the stepper motor's original position, ensuring continuous awareness of its whereabouts within the system.

Introductions to the Principle of the Sensor

Please find the infrared sensor as shown in the following image.

Top	Bottom
	

The infrared sensor module comprises both an infrared emitting diode and an infrared receiving diode. When the module is powered by a 3.3V-5V source, the infrared emitting diode emits infrared light signals outward. These signals are then reflected back upon encountering objects and subsequently detected by the infrared receiving diode, as depicted in the diagram on the left below.



The schematic diagram of the infrared sensor is shown in the above right image.

When the infrared sensor is close to an object, the emitted infrared light signals reflect back from the object's surface. This reflection triggers a low voltage level at AO and DO, lighting up indicator light D1 to signify object detection.

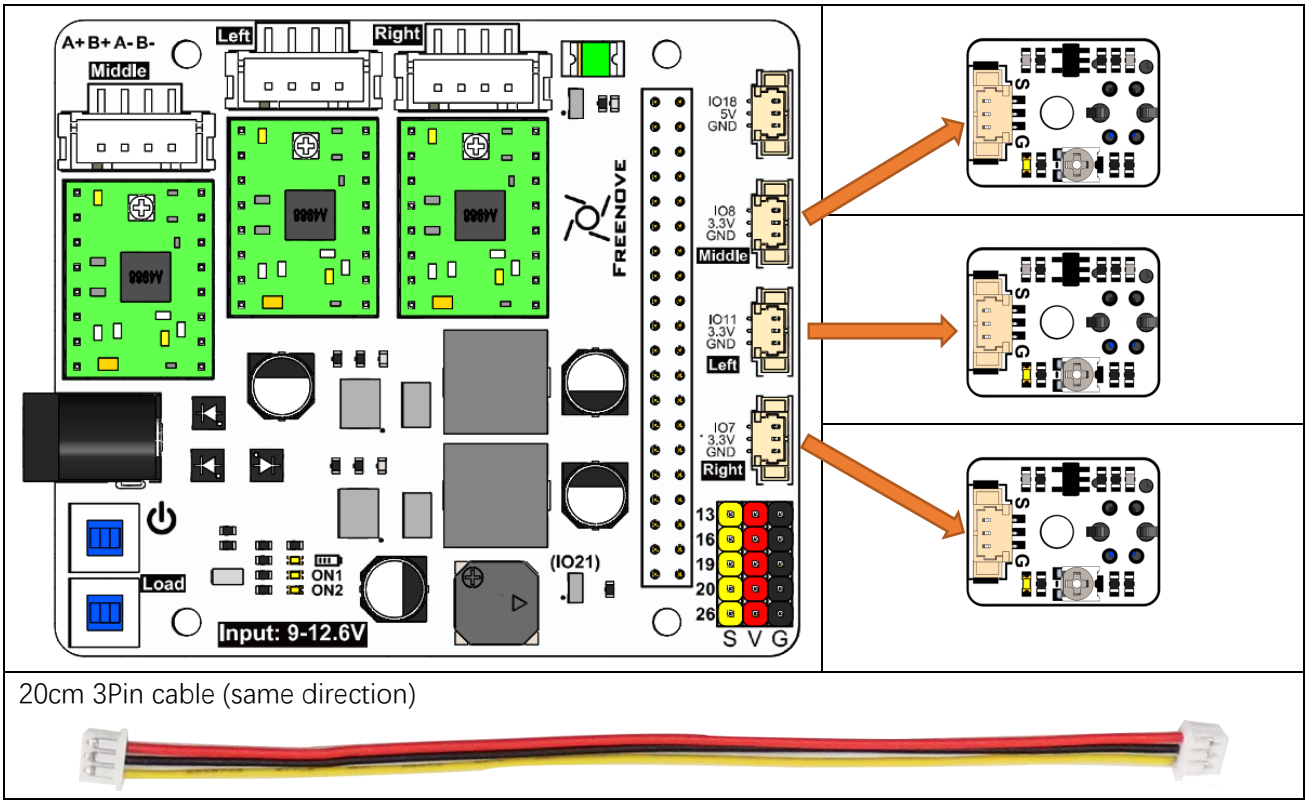
Conversely, when the sensor is distant from an object, the reflected infrared signals are weak, resulting in high voltage levels at AO and DO, causing indicator light D1 to turn off.

Furthermore, infrared signals are absorbed by black objects, preventing their reflection back to the sensor.

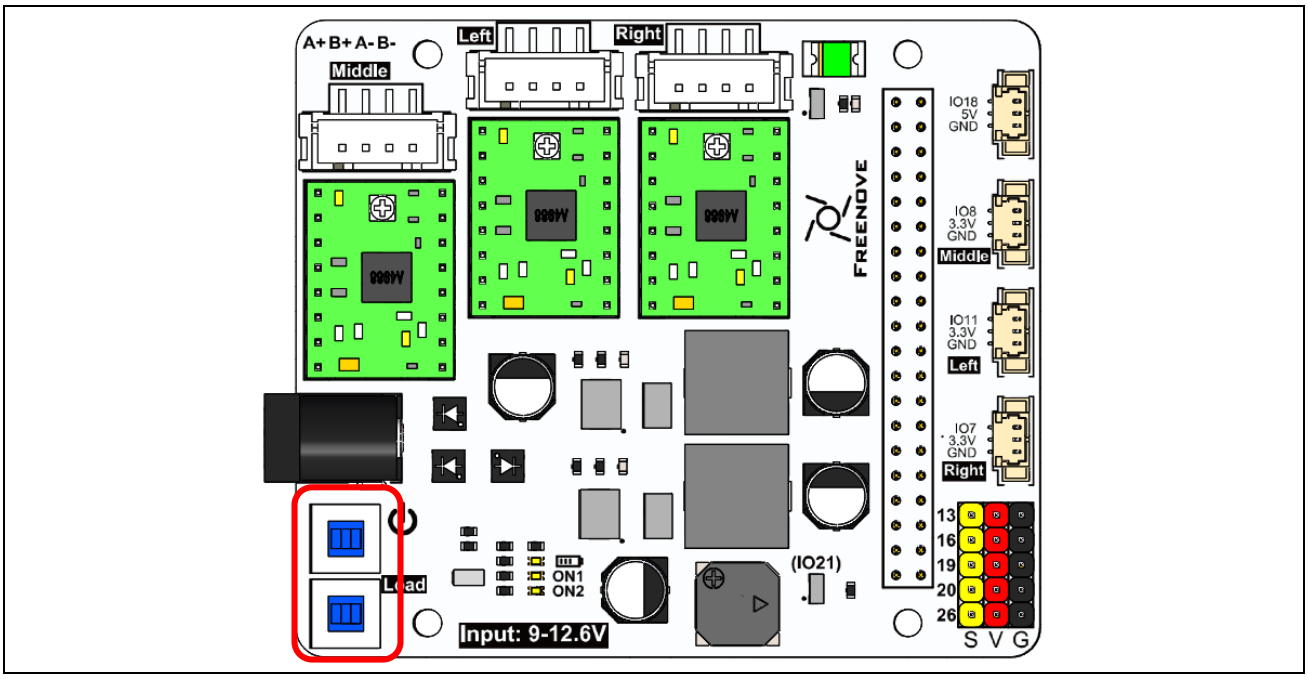
The infraredsensor receives infrared signals?	AO	DO	Status of the indicator
Yes	LOW	LOW	ON
No	HIGH	HIGH	OFF

Running Code to Test the Sensors

1. Connect the three sensors to the robot board with three 3P cables in the kit. (Power cable is not shown in the diagrams below.)

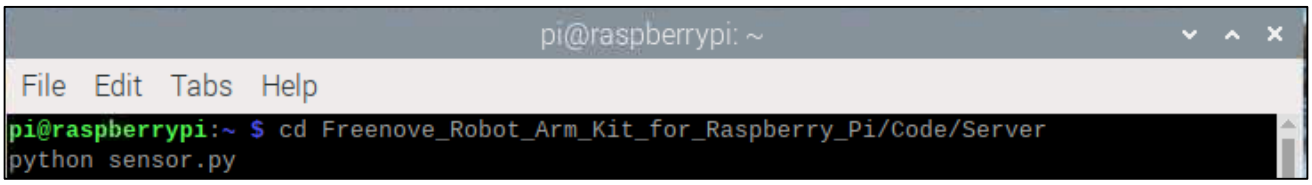


2. Turn ON the power switches.

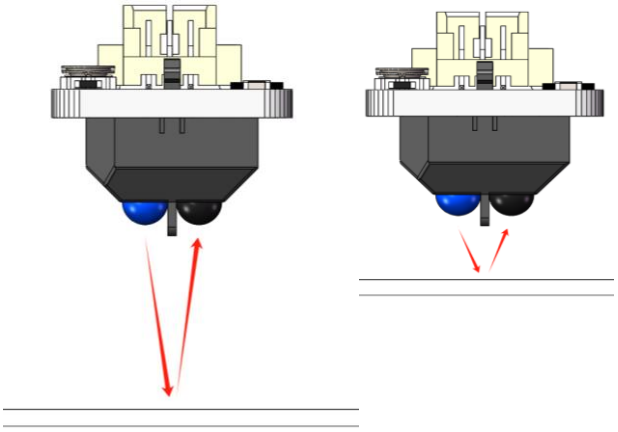
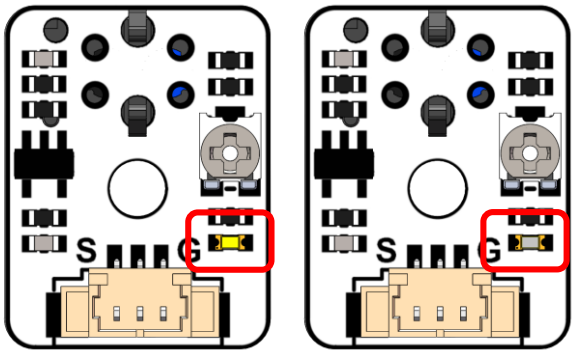


3. Run the following commans on the Raspberry Pi Terminal.

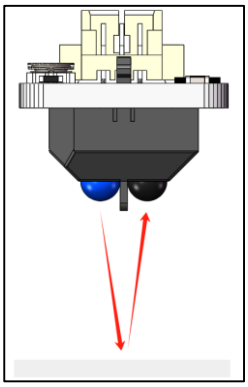
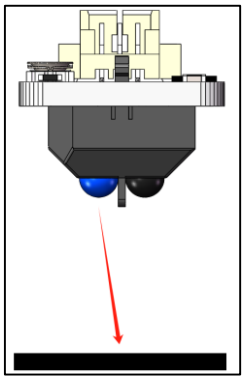
```
cd Freenove_Robot_Arm_Kit_for_Raspberry_Pi/Server/Code
python sensor.py
```



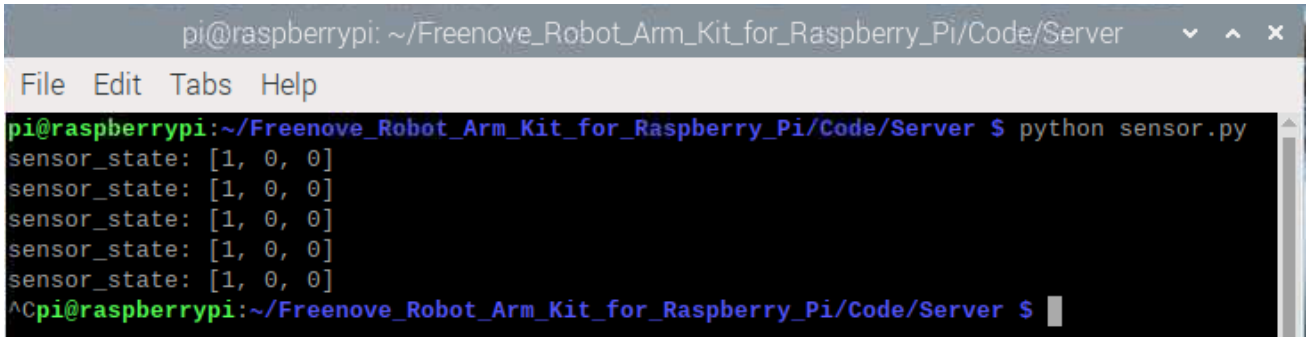
4. Align the sensor with objects and move it up and down. Notice the indicator light changing with each movement. Once the light blinks once, a prompt message will appear on the console."

Move the sensor up and down	Phenomenon
	

Note: Avoid using objects with black surface, otherwise the infrared lights signals will be absorbed and cannot return to the sensor.

5. Press Ctrl+C to exit the program.



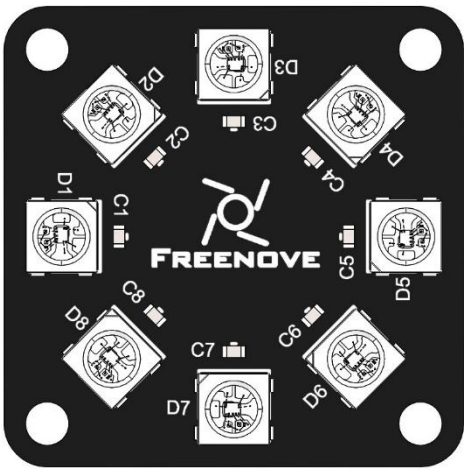
RGB LED Module Test

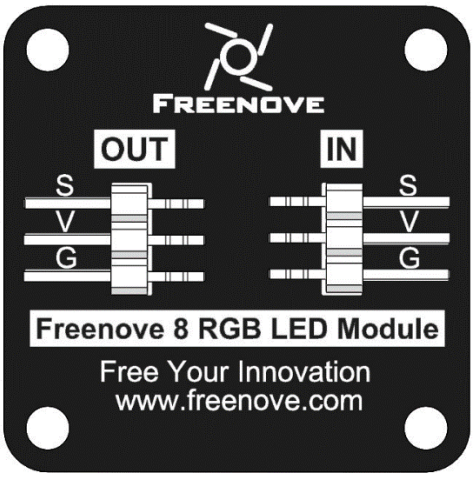
LedPixel Module

The LedPixel is as below.

It consists of 8 WS2812, each of which requires only one pin to control and supports cascade. Each WS212 has integrated 3 LEDs, red, green and blue respectively, and each of them supports 256-level brightness adjustment, which means that each WS2812 can emit 2^24=16,777,216 different colors.

You can use only one data pin to control eight LEDs on the module. As shown below:





Pin description:

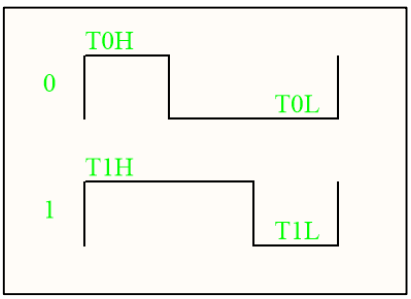
(IN)		(OUT)	
symbol	Function	symbol	Function
S	Input control signal	S	Output control signal
V	Power supply pin, +3.5V~5.5V	V	Power supply pin, +3.5V~5.5V
G	GND	G	GND

The color of each LED consists of red, green, and blue components, with each color occupying 8 bits of data. Therefore, to control the color of an LED, we need to send 24 bits of color data to the LED through pins.

G7	G6	G5	G4	G3	G2	G1	G0	R7	R6	R5	R4	R3	R2	R1	R0	B7	B6	B5	B4	B3	B2	B1	B0

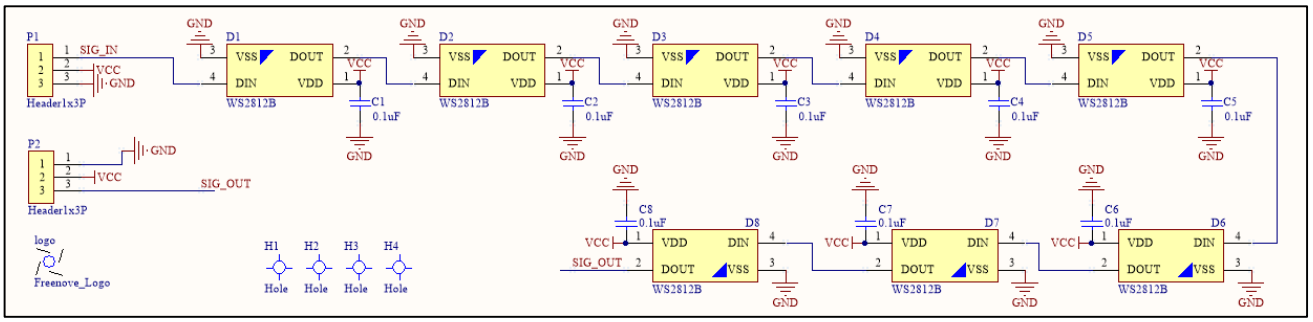
Any of the 24-bit color data can be represented by 1 and 0.

The timing waveform diagram is shown below.



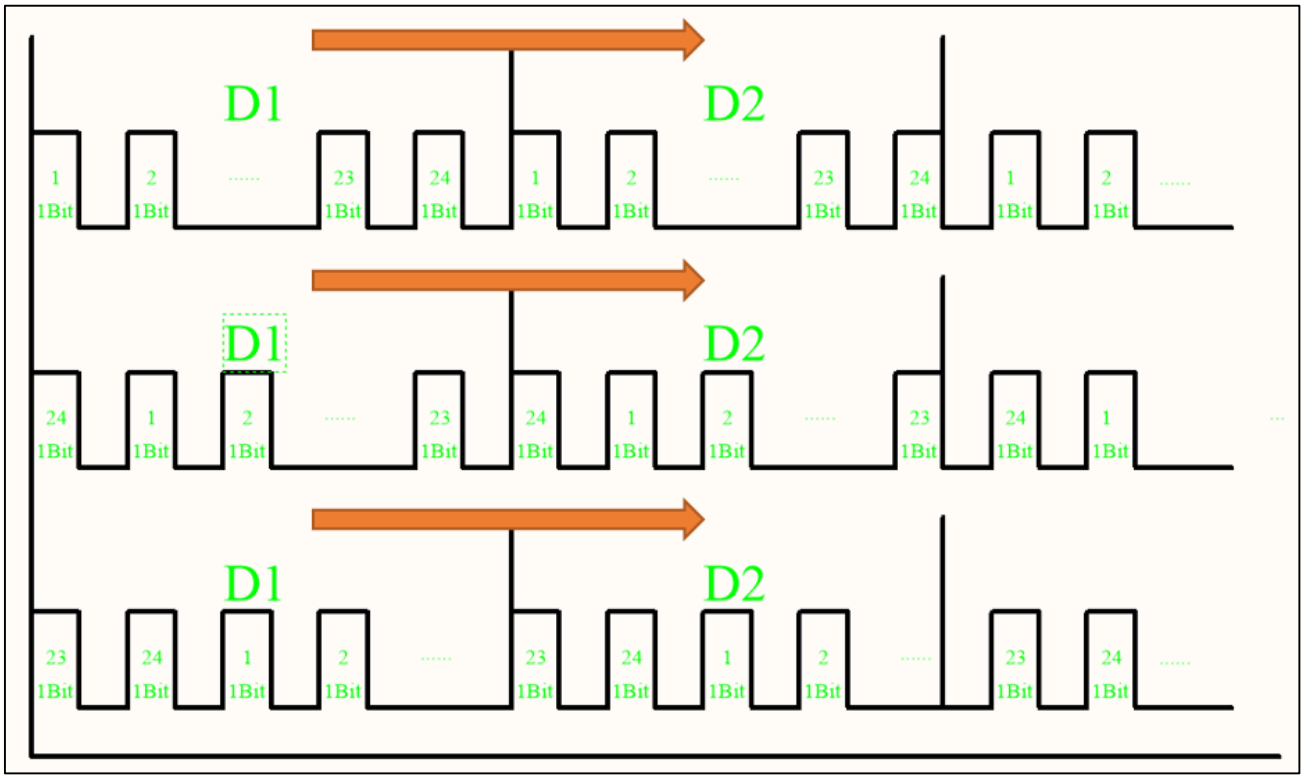
T0H	0 code, high voltage time	0.4us	+-150ns
T0L	0 code, low voltage time	0.85us	+-150ns
T1H	1 code, high voltage time	0.85us	+-150ns
T1L	1 code, low voltage time	0.4us	+-150ns
RST	Low voltage time	Over 50us	

The schematic diagram is as shown below.



The previous schematic reveals that the Raspberry Pi connects to the DIN pin of the LEDs through SIG_IN and sends the data to the LED.

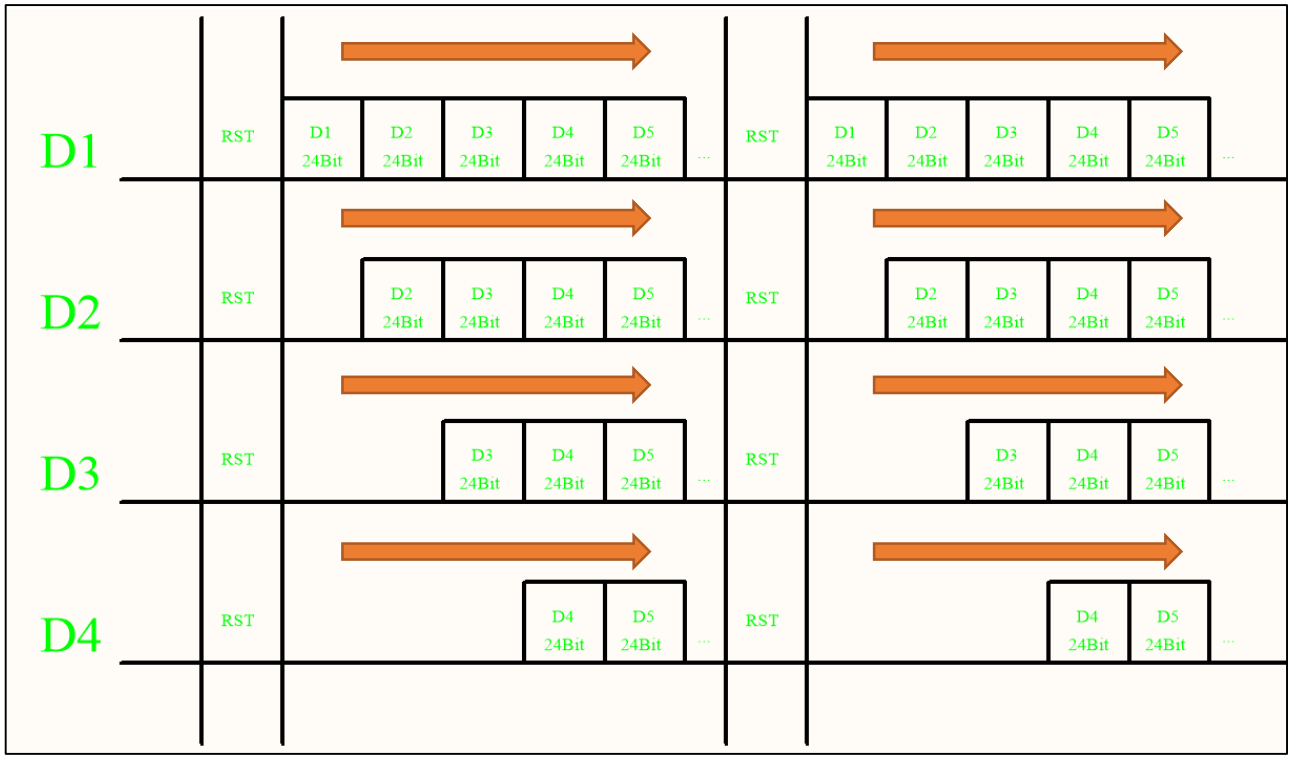
The LED data is input via the DIN pin of and output from the DOUT pin, as shown below.



Every time the DIN pin of a RGB LED receives 1 bit of data, it shifts the 24 bits of data within the LED it back one bit, and the excess bit of data is then sent to the next LED via the DOUT pin.

Therefore, to control serveral LEDs with Raspberry Pi, it needs to send the 24-bit color data of the LEDs continuously.

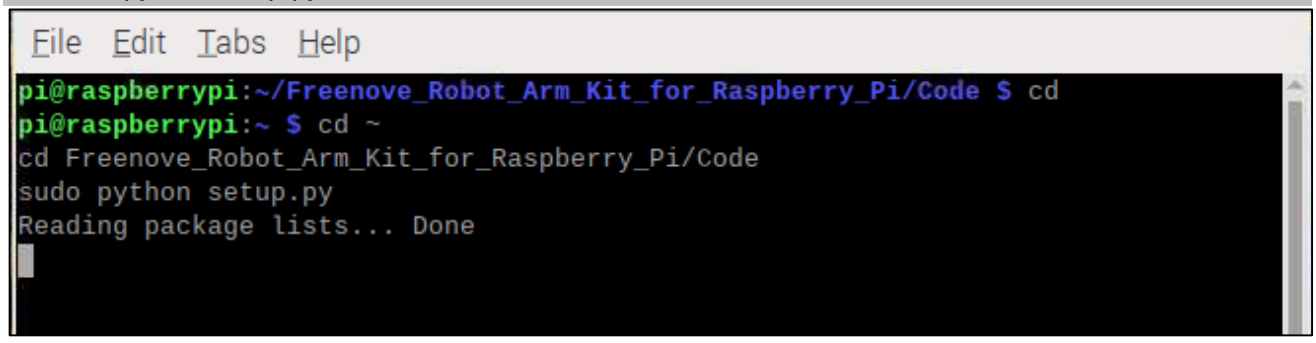
To control 8 RGB LEDs, the Raspberry Pi needs to send 8 groups of 24-bit color data to the LEDs. Fist, send the 24-bit color data of D7, then send those of D6, and repeat in this way until finally send the 24-bit data of D1.



rpi-ws281x-python

In our tutorial, we use the rpi-ws281x-python library to drive the LED module.
Please run the following commands to install the library. If you have installed it, you can skip this step.

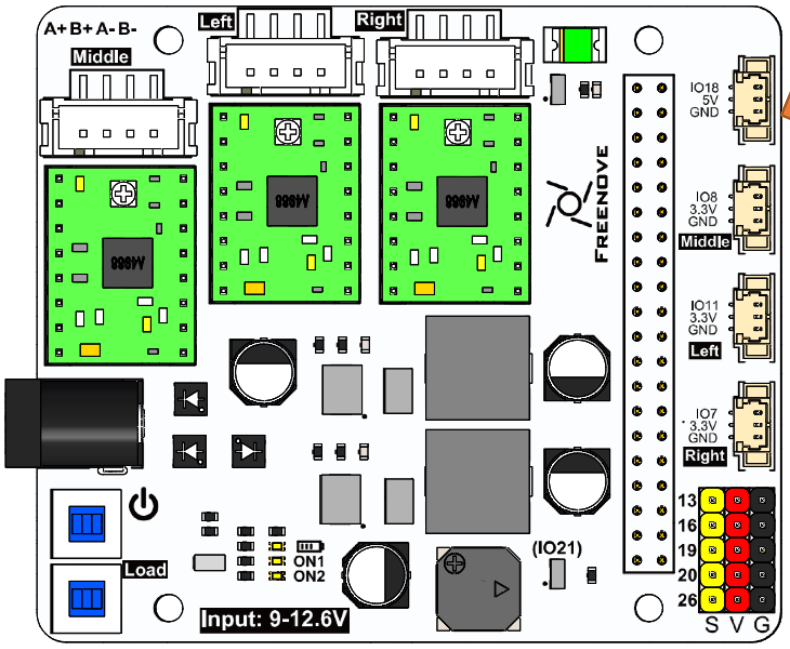
```
cd ~
cd Freenove_Robot_Arm_Kit_for_Raspberry_Pi/Code
sudo python setup.py
```

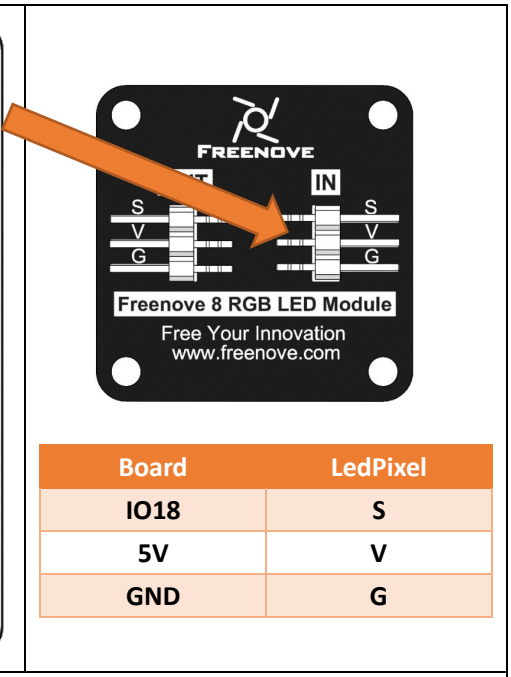


Wait for the library to install.

Running Code to Test the RGB LED Module

1. Use the 3P to jumper wires to connect the RGB LED module. (The power cable is not shown in the diagrams below.)

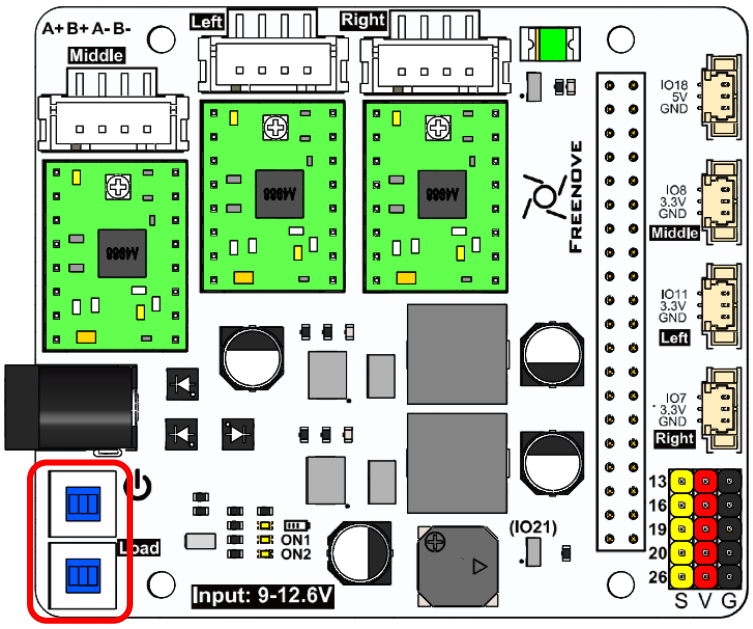




Board	LedPixel
IO18	S
5V	V
GND	G

10cm 3Pin cable to Jumper wire.

2. Turn ON the power switch.



3. Run the commands on Raspberry Pi Terminal.

```
cd ~
cd Freenove_Robot_Arm_Kit_for_Raspberry_Pi/Server/Code
sudo python ledPixel.py 1
```

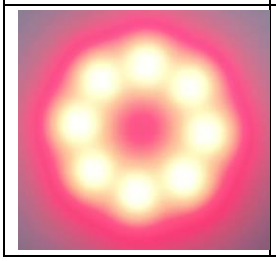
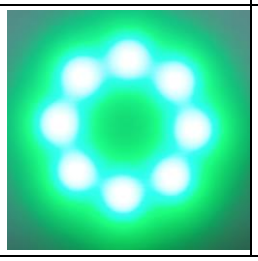
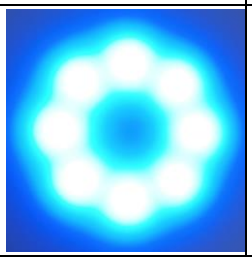
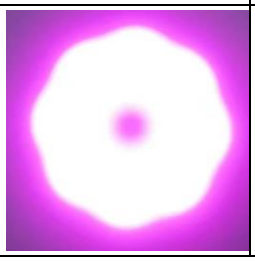
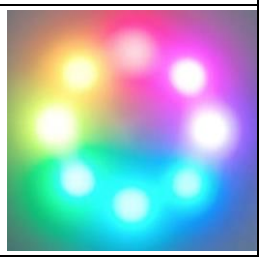
```
File  Edit  Tabs  Help
pi@raspberrypi:~ $ cd ~
cd Freenove_Robot_Arm_Kit_for_Raspberry_Pi/Code/Server
sudo python ledPixel.py 1
```

4. Press Ctrl+C to exit the program.

Here we add parameters to the command. The first parameter indicates the mode of the LED module. There are 6 modes in total, so the range of this parameter is 1-6. The second, third, and fourth parameters represent the data of red, green, and blue colors respectively, and the range is 0-255.
You can try the following commands.

Commands	Explanation
sudo python ledPixel.py 1	Customize the color of the RGB LED. Without parameters, the color is red by default.
sudo python ledPixel.py 1 0 0 255	Customize the color, The parameters set the red and green to 0 and the blue to 255, so the LED module only emits blue light.
sudo python ledPixel.py 2	Repeatedly emits red, green and blue. Paramters are not needed.
sudo python ledPixel.py 3	Running water effect. Without parameters, the color is red by default.
sudo python ledPixel.py 3 0 255 0	Running water effect. The parameters set the red and blue to 0 and the green to 255, so the LED module only emits green light.
sudo python ledPixel.py 4	Gradient light, slowly changing in rainbow colors. Parameters are not needed.
sudo python ledPixel.py 5	Rainbow lights, repeatedly turning the rainbow colors. Parameters are not needed.
sudo python ledPixel.py 6	Breathing light. Withouth parameters, the color is red by default.
sudo python ledPixel.py 6 255 0 255	Breathign light. The parameters set the red and blue to 255 and the green to 255, so the LED module only emits red and blue light, which looks pinkish purple.

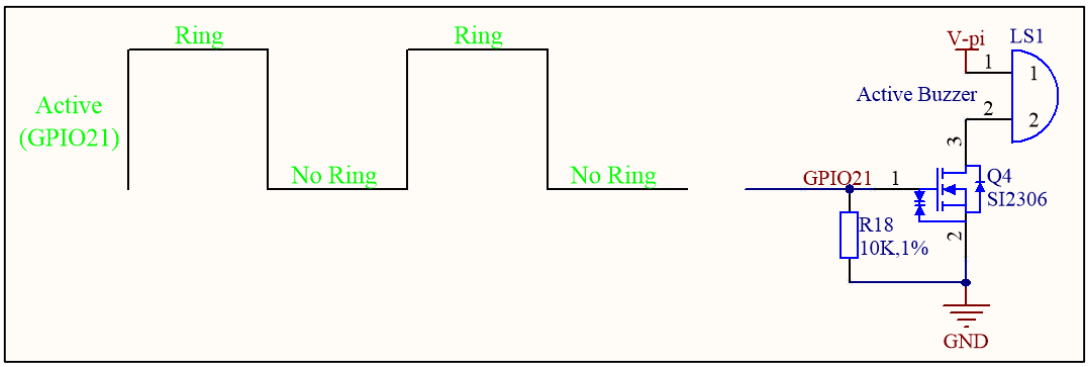
The colors displayed are as below:

Red	Green	Blue	Pinkish purple	Rainblow lights
				

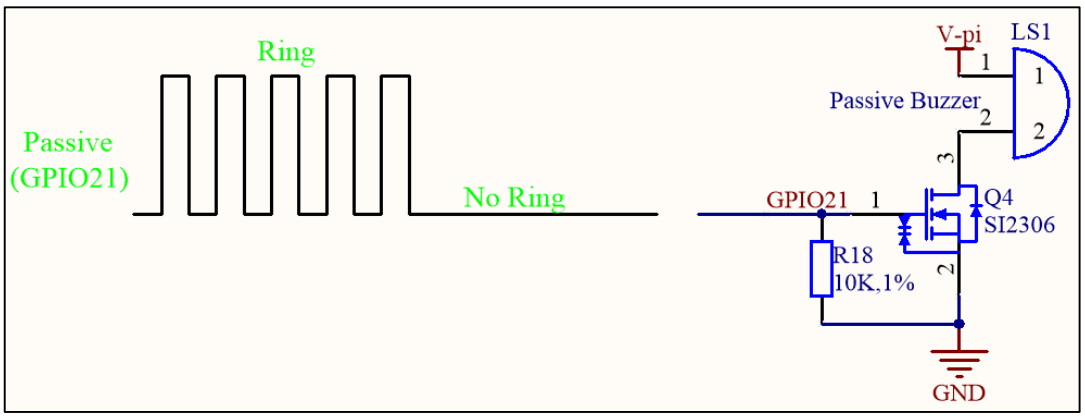
Buzzer Test

Introduction to Buzzer

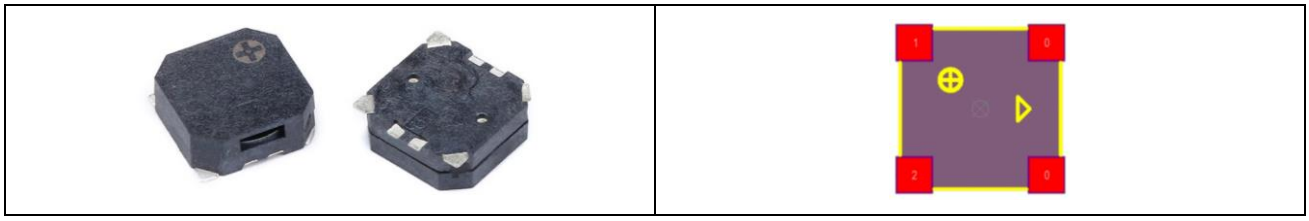
Buzzer can be classified into active buzzer and passive buzzer according to the driving method.
Active buzzer has an internal oscillation source, which can continuously emit sound when connected to a DC voltage, with a fixed frequency. With the circuit below, when GPIO21 outputs a high level, the active buzzer will sound, and when GPIO21 outputs a low level, the active buzzer will not sound.



Passive buzzers do not have an internal oscillation source; they require an external pulse signal to produce sound. By adjusting the frequency of the pulse signal provided to the connected pin, different frequencies of sound can be emitted. In the circuit below, when GPIO21 outputs pulse signals of different frequencies, the passive buzzer can produce different sounds. When GPIO21 outputs a low level, the passive buzzer does not emit any sound.

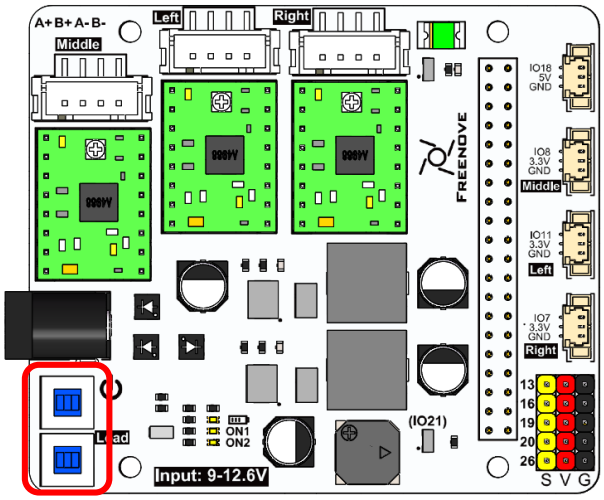


In this kit, we use the MLT-8530 passive buzzer as the sound-producing device. As shown in the diagram below, the buzzer pins 1 and 2 correspond to LS1-1 and LS1-2 in the schematic diagram above. Pin 0 is only used to fix the part.



Running Code to Test the Buzzer

1. Turn ON the power switch.

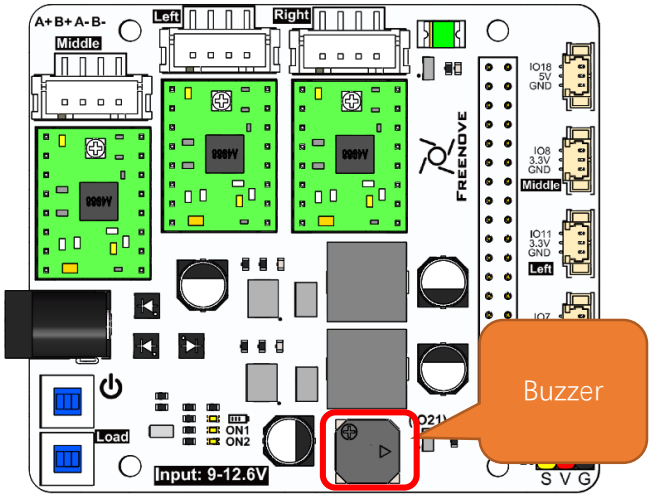


2. Run the commands on Raspberry Pi Terminal.

```
cd ~
cd Freenove_Robot_Arm_Kit_for_Raspberry_Pi/Server/Code
sudo python buzzer.py
```

```
pi@raspberrypi:~ $ cd ~
cd Freenove_Robot_Arm_Kit_for_Raspberry_Pi/Code/Server
sudo python buzzer.py
```

3. The onboard buzzer emits three brief beeps.



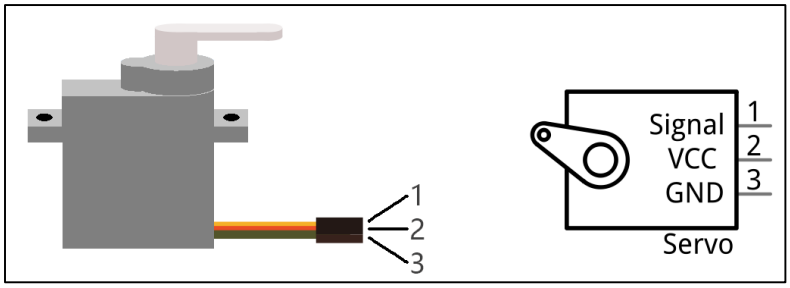
You can try the following commands.

Commands	Explanations
sudo python buzzer.py	The buzzers makes three beeps with a frequency opf 2kHz, each lasting 100ms.
sudo python buzzer.py 2500	The buzzers makes three beeps with a frequency opf 2.5kHz, each lasting 100ms.
sudo python buzzer.py 2000 300	The buzzers makes three beeps with a frequency opf 2kHz, each lasting 300ms.
sudo python buzzer.py 2000 100 5	The buzzers makes five beeps with a frequency opf 2kHz, each lasting 100ms.

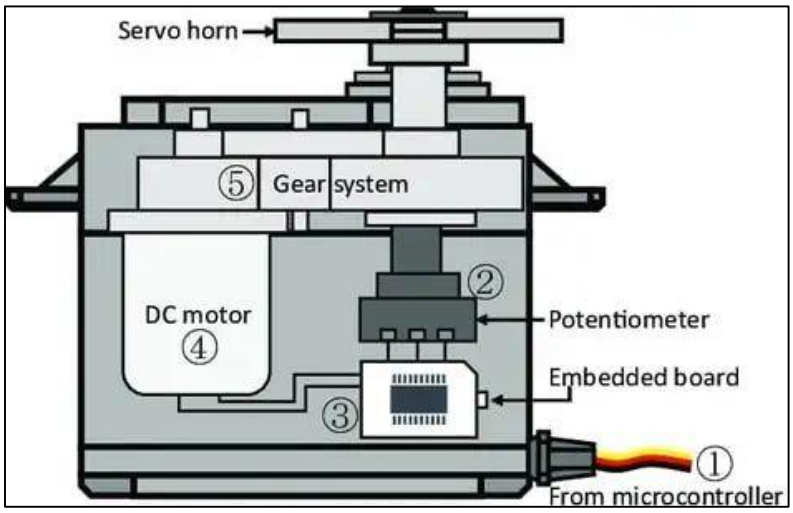
Servo Test

Introduction to the Servo

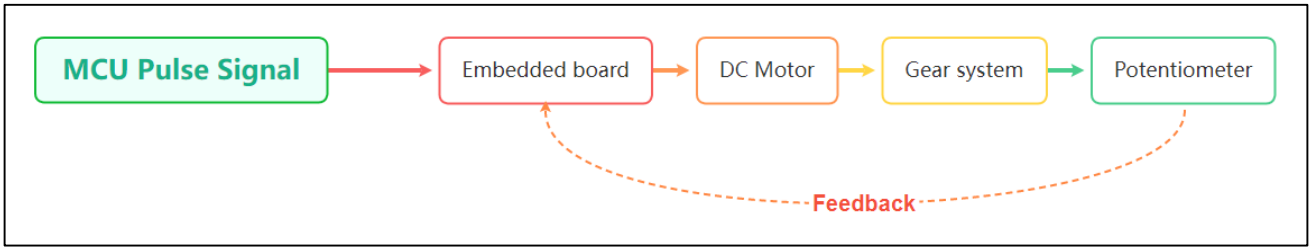
In this kit, we use a servo motor as the power source for the robotic arm clamp. The servo motor is shown in the diagram below."



The servo has three wires, among which, the red one resprensents positive of power supply, brown/black one indicates GND, and the orange/yellow one is the signal cable.



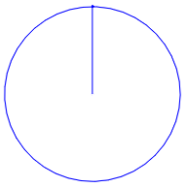
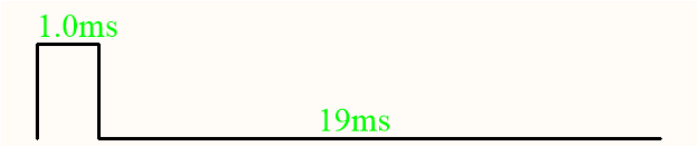
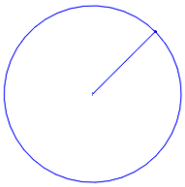
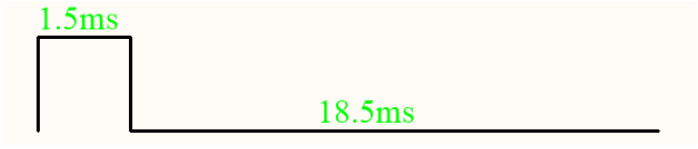
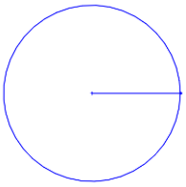
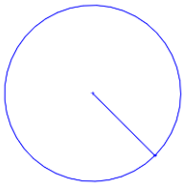
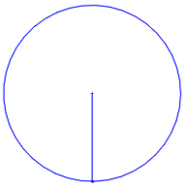
When the servo motor receives an angle signal through its wires, the onboard control board processes this signal and regulates the motor's rotation. The motor's rotation in turn drives the gears, which subsequently rotate the potentiometer. The ADC value of the potentiometer is then sent back to the control board. Based on this ADC value, the control board determines whether the motor has reached the desired angle and adjusts its rotation direction accordingly. This process is depicted in the diagram below.



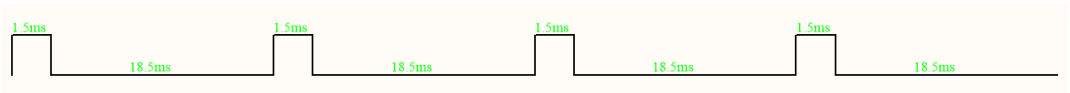
The clock signal period of the servo's internal control board is 20ms, so if we want to control the rotation of the servo to the target angle, our clock signal period must also be 20ms. f = 1s / t = 1s / 20ms = 50Hz.

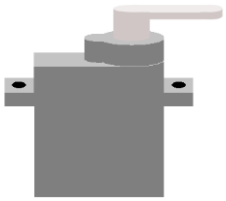
The servo controls the rotation angle with 50Hz pulse signals.

The pulse signals are as shown below:

Signals	Servo rotation angle
	0 degree 
	45 degrees 
	90 degrees 
	135 degrees 
	180 degrees 

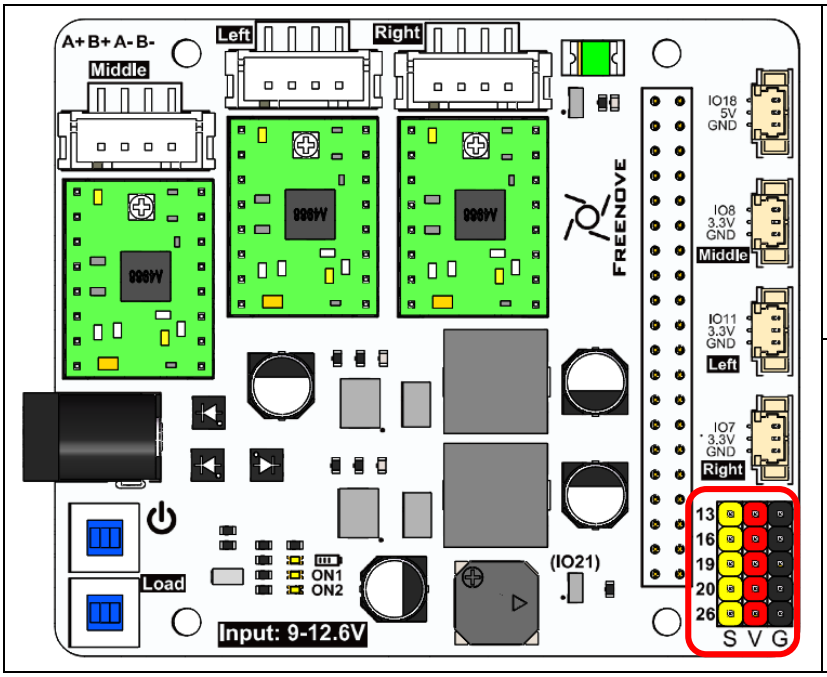
As depicted in the diagram above, to control the servo motor to rotate to a 90-degree position, the servo signal wire needs to first output a 1.5ms high-level signal, followed by an 18.5ms low-level signal, repeating this process in a cycle.



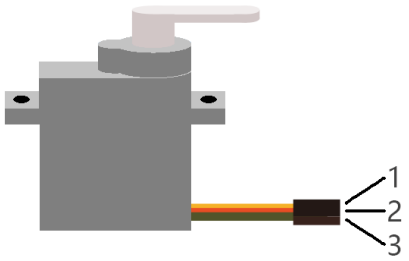


Running Code to Test the Servo

1. Connect the servos to the pins marked below. (The power cable is not shown in the diagrams below.)



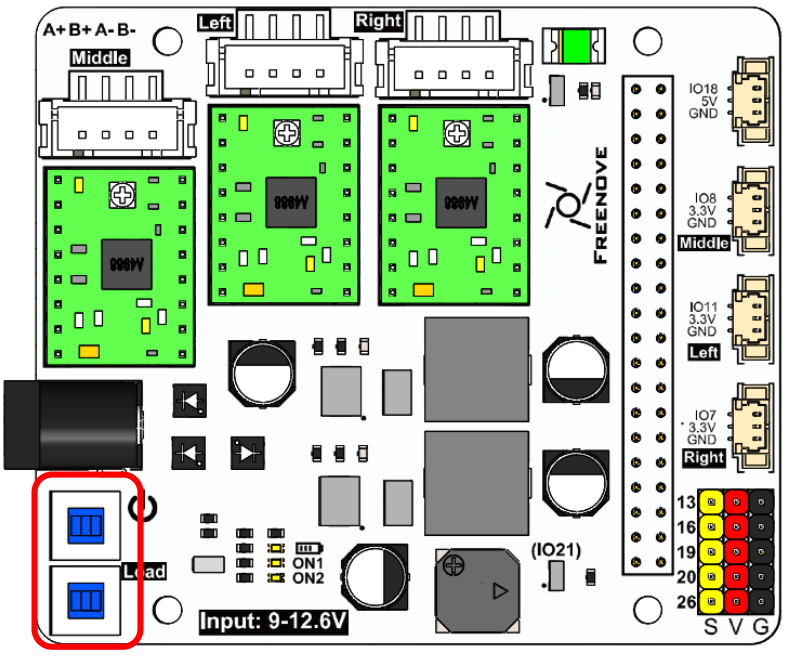
Simulation drawing of the servo



Wiring of the servo

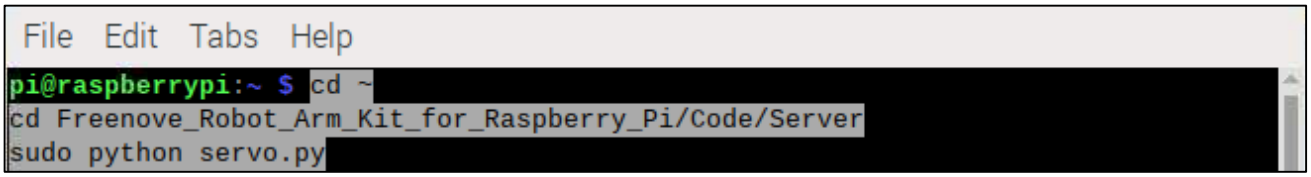
Board	Servo
S	1
V	2
G	3

2. Turn ON the power switch.

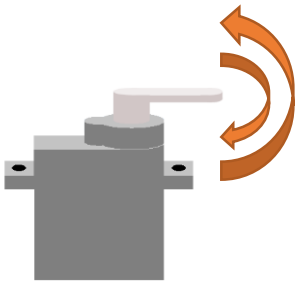


3. Run the following commands on Raspberry Pi terminal.

```
cd ~
cd Freenove_Robot_Arm_Kit_for_Raspberry_Pi/Server/Code
sudo python servo.py
```



4. The servo motor repeatedly moves from the 0-degree position to the 180-degree position and then back to the 0-degree position.



Here are some commands for your reference.

Commands	Explanations
sudo python servo.py	Simultaneously controls five channels of servos to rotate back and forth repeatedly.
sudo python servo.py 0	Controls the servo of channel 0 (GPIO13) to rotate to 90 degress and stay at that position.
sudo python servo.py 0 90	Controls the servo of channel 0 (GPIO13) to rotate to 90 degress and stay at that position.
sudo python servo.py 0 150	Controls the servo of channel 0 (GPIO13) to rotate to 150 degress and stay at that position.
sudo python servo.py 1 45	Controls the servo of channel 1 (GPIO16) to rotate to 45 degress and stay at that position.

There are two parameters in the commands. The first one is used to select the pin that controls the servo, whose range is 0-4; and the second one is to control the angle of rotation of the servo, the range is 0-180 degrees.

Commands	Index	GPIO Number
sudo python servo.py 0 90	0	GPIO13
sudo python servo.py 1 90	1	GPIO16
sudo python servo.py 2 90	2	GPIO19
sudo python servo.py 3 90	3	GPIO20
sudo python servo.py 4 90	4	GPIO26

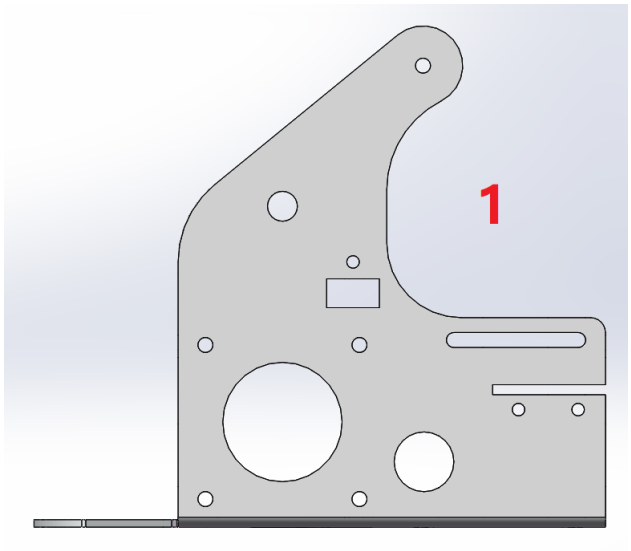
Chapter 2 Assembly of the Robot Arm

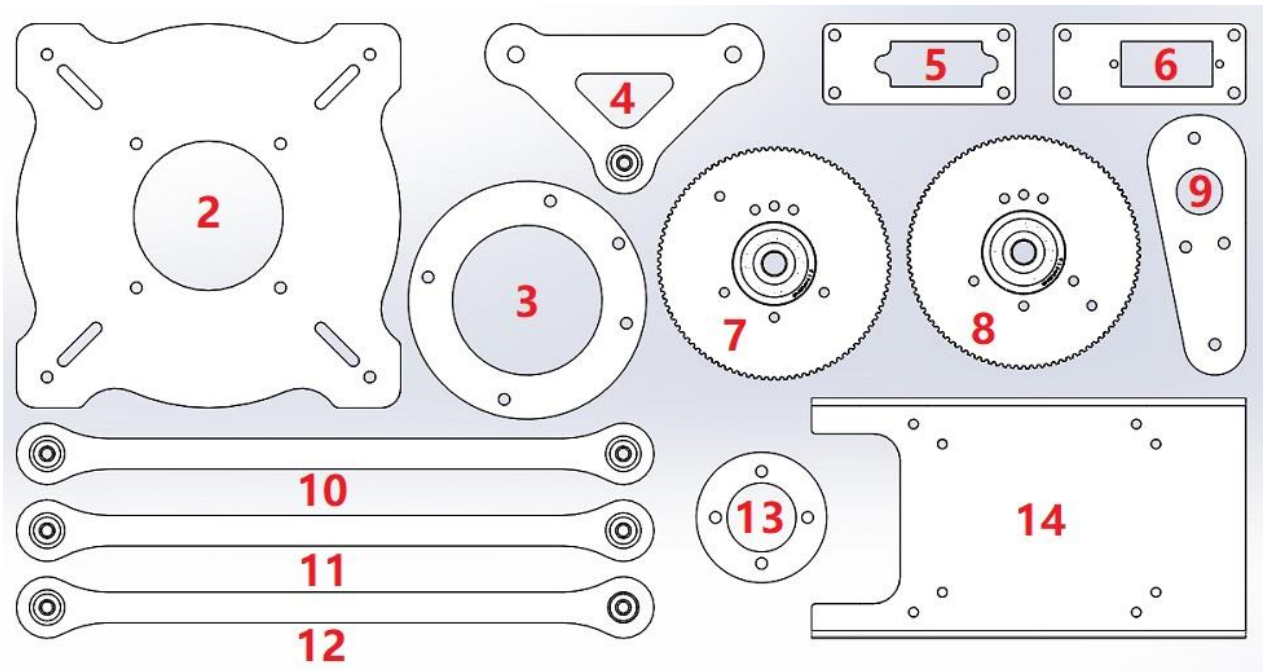
If you have any concerns, please feel free to contact us at support@freenove.com

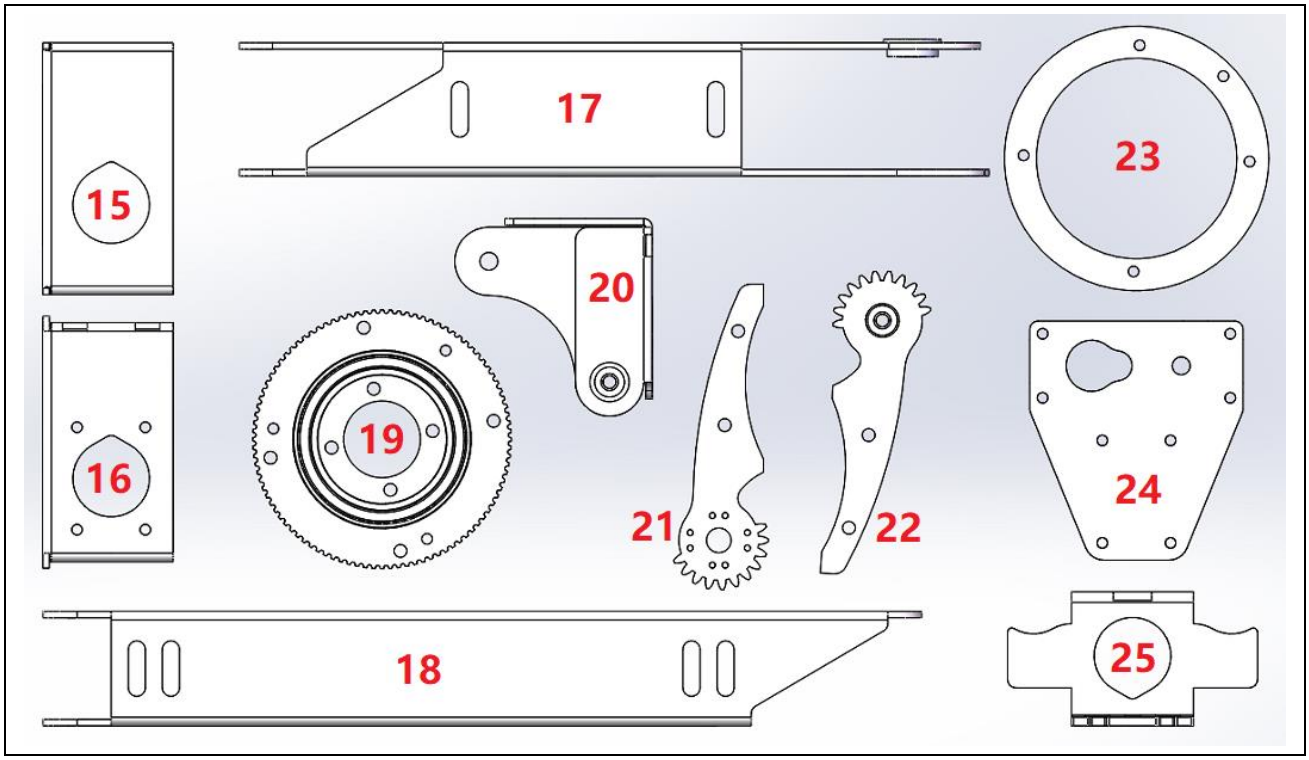
It is recommended to assemble and use the robot arm according to the tutorial. Otherwise, there may be installation errors, device damage, etc.

Numbering of the Assembly Components

Before assembling the arm, we number the assembly parts of the robot arm to facilitate the assembly.



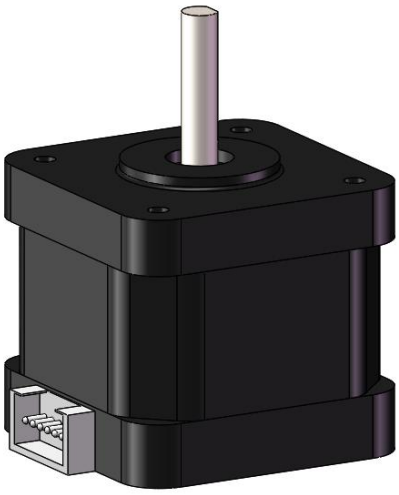
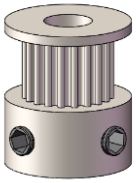




If there are wrong/missing parts in your kit, please contact us: support@freenove.com

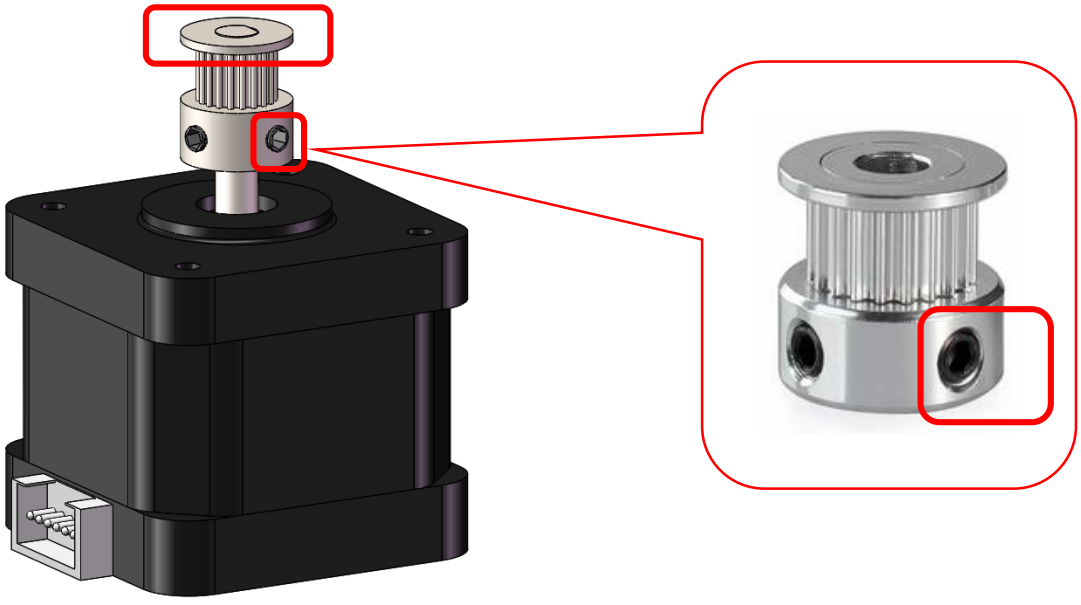
Step 1 Mounting the Timing Pulleys to the Stepper Motor

Materials Needed

Stepper Motor X3	Timing Pulley -2GT-16 X3
	 Please Note That Each Timing Pulley Has 2 Bolts.
Allen Key 	

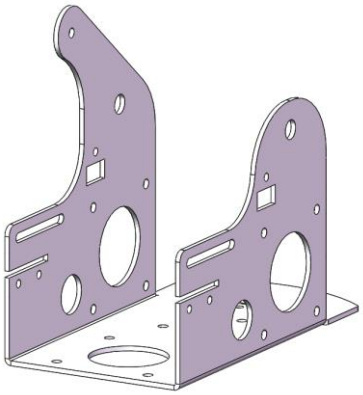
Installation Steps

1. Mount the timing pulleys to the rotating axle of the stepper motors. Make sure the timing pulley is flush with the axle.
2. Adjust the bolts on the timing pulley with the Allen key to fix the part.



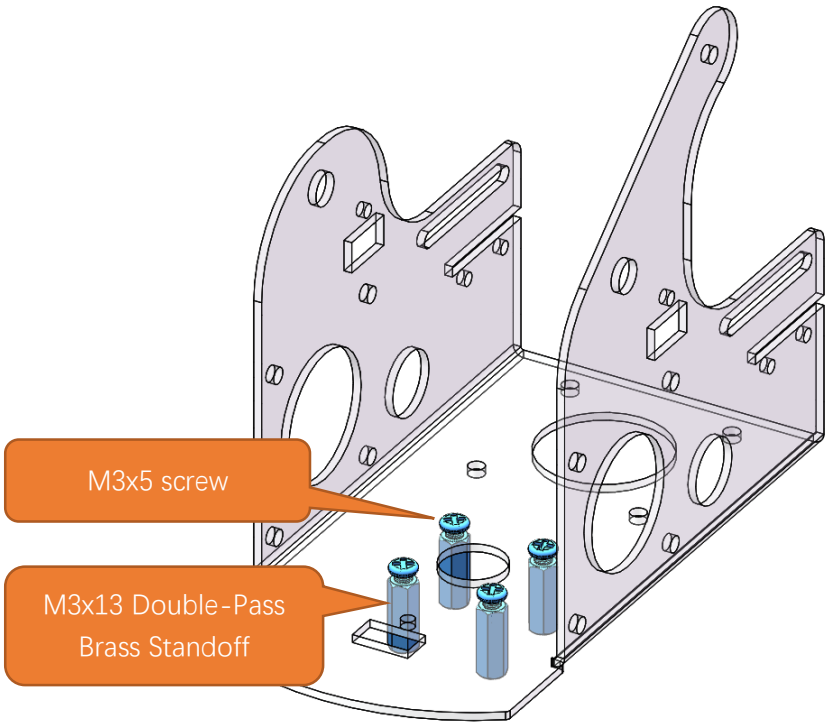
Step 2 Mounting the Stepper Motors to the No.1 Assembly Part

Materials Needed

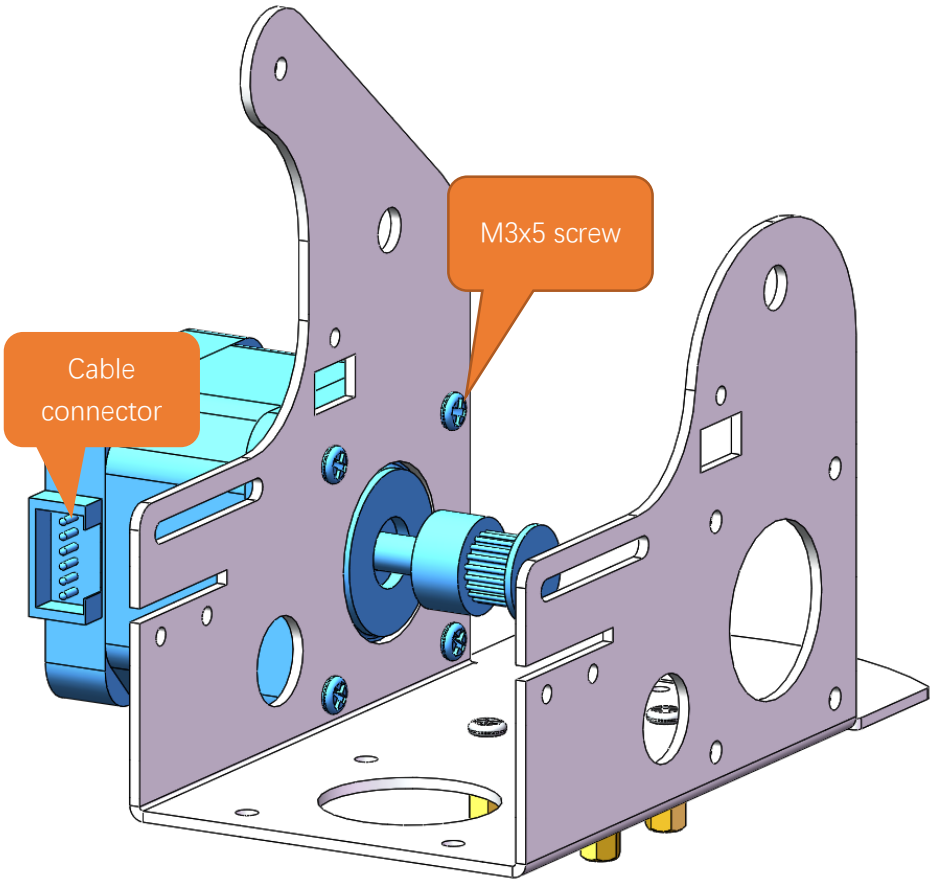
No.1 Assembly Component 	Stepper Motor X3 	
M3x5 Screws X16 	M3x13 Double-Pass Brass Standoff X4 	Cross Screwdriver (3mm) X1 

Installation Steps

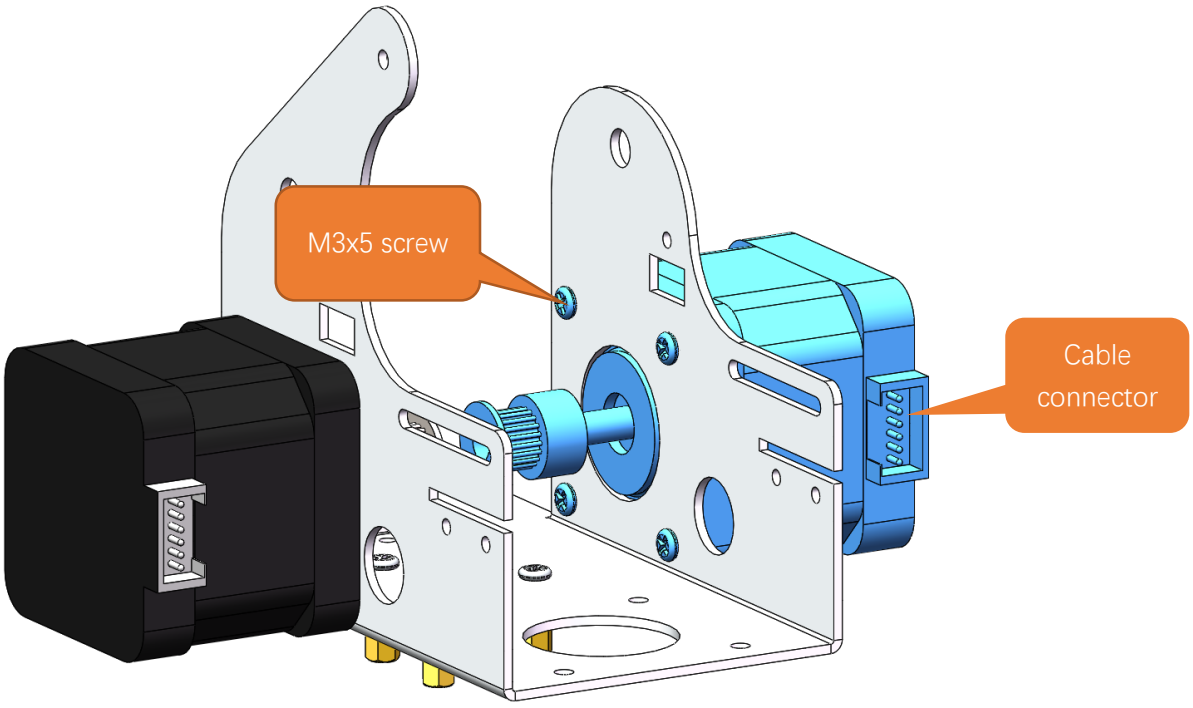
1. Use four M3x5 screws to fasten the four M3x13 double-pass brass standoffs onto the No.1 assembly component. (Infrared sensos are not shown in the diagrams below.)



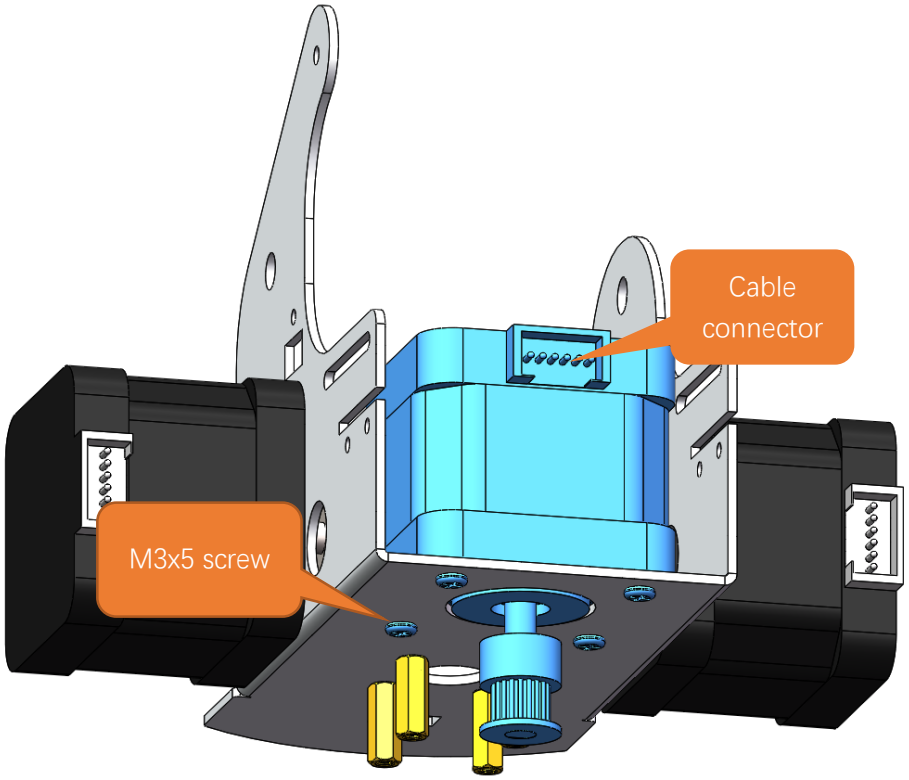
2. Fix one stepper motor to the left of the No.1 assembly part with four M3x5 screws. Pay attention to the position of the cable connector of the motor.



3. Fix another stepper motor to the right of the No.1 assembly part with four M3x5 screws. Pay attention to the position of the cable connector of the motor.

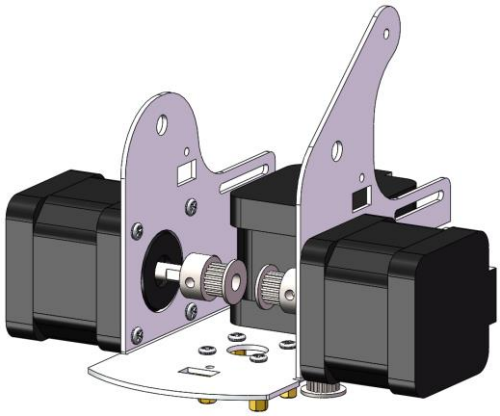
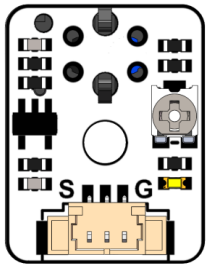


4. Fix the third stepper motor to the middle of the No.1 assembly component with four M3x5 screws. Pay attention to the position of the cable connector of the motor.



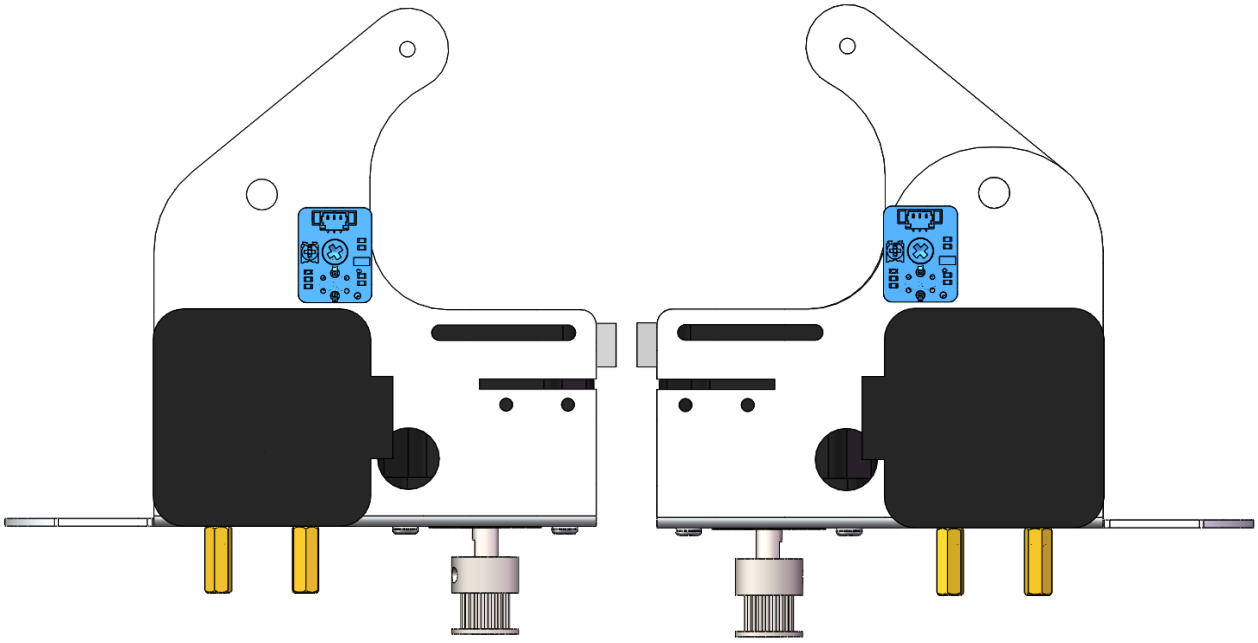
Step 3 Mounting the Infrared Sensors

Materials need

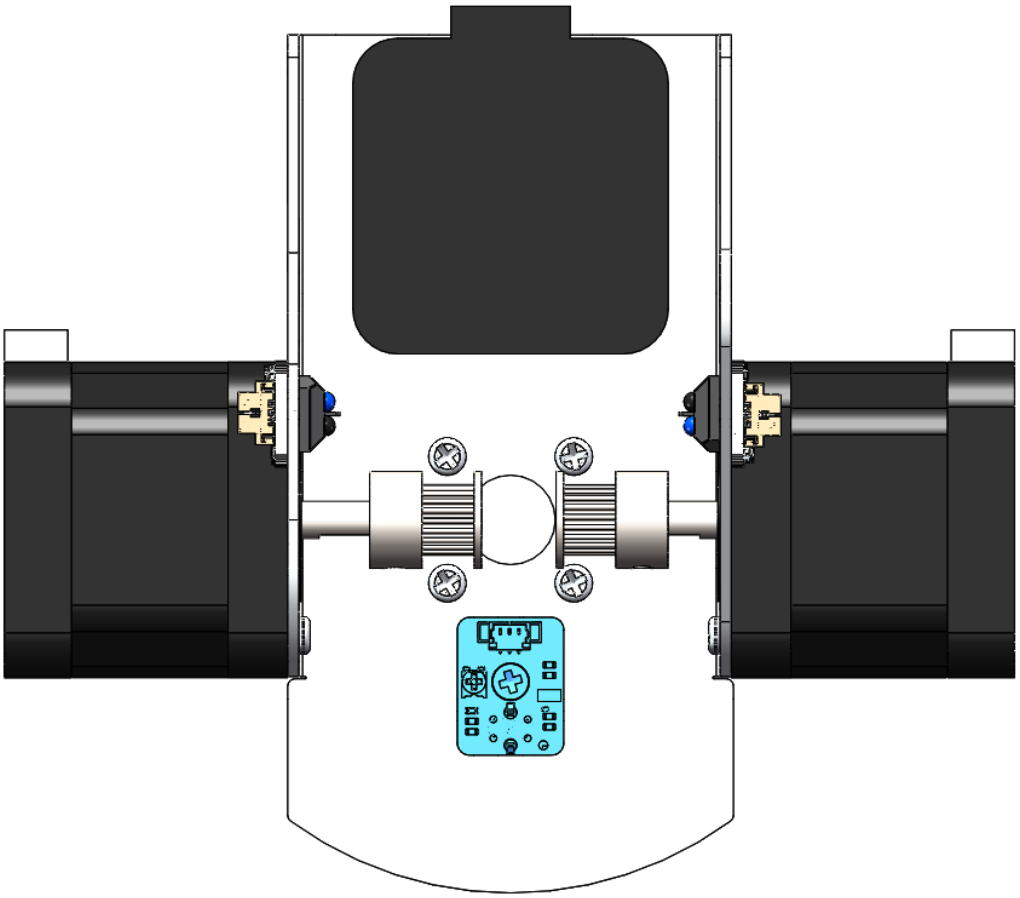
Robot Arm Assembly 	Infrared Sensors X3 
M3x3x5 Screws X3 	Cross Screwdriver (3mm) X1 

Installation Steps

Mount two infrated sensors to the No.1 assembly part wih two M3x3x5 screws, as shown below.

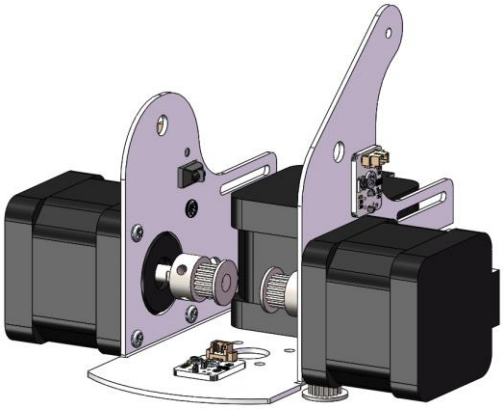
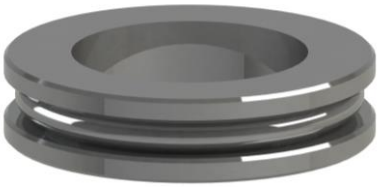
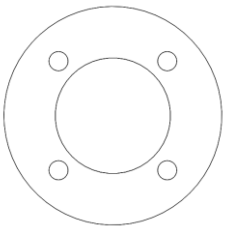
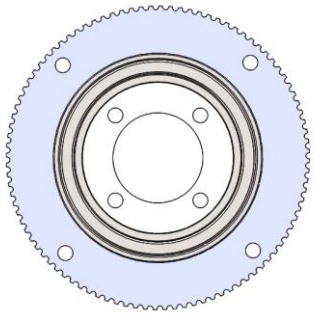
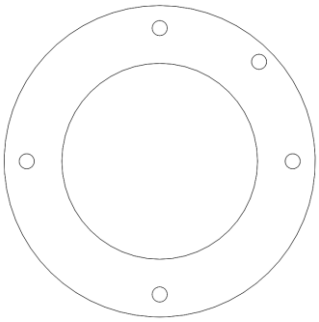
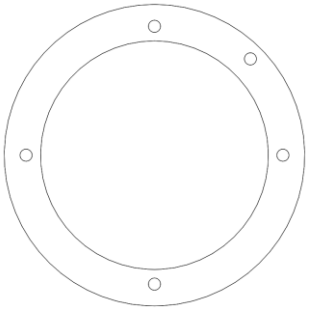


Mount the other infrared sensor to No.1 assembly component with an M3x3x5 screw, as shown below.



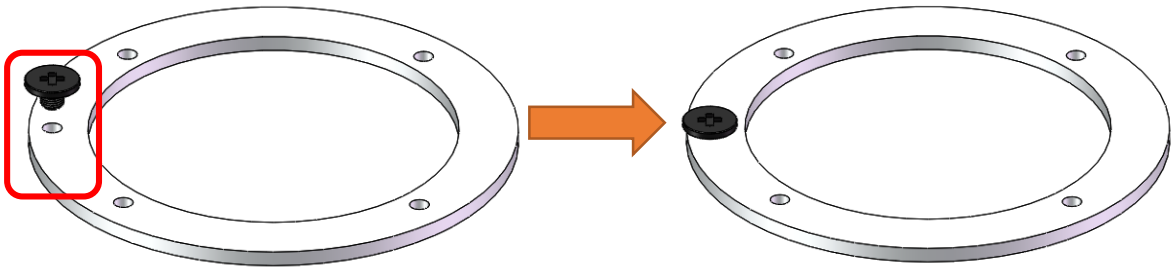
Step 4: Installing the Bottom Transmission Device

Material needed

Robot Arm Assembly 	Plane Thrust Bearing (51106-30x47x11) 
M3x12 Screws X4 	M3x3x7 Screws X1 
No.13 Assembly Component 	No. 19 Assembly Component 
No. 3 Assembly Component X1 	No. 23 Assembly Component 
Allen Key 	Cross Screwdriver (3mm) X1 

Installataion Steps

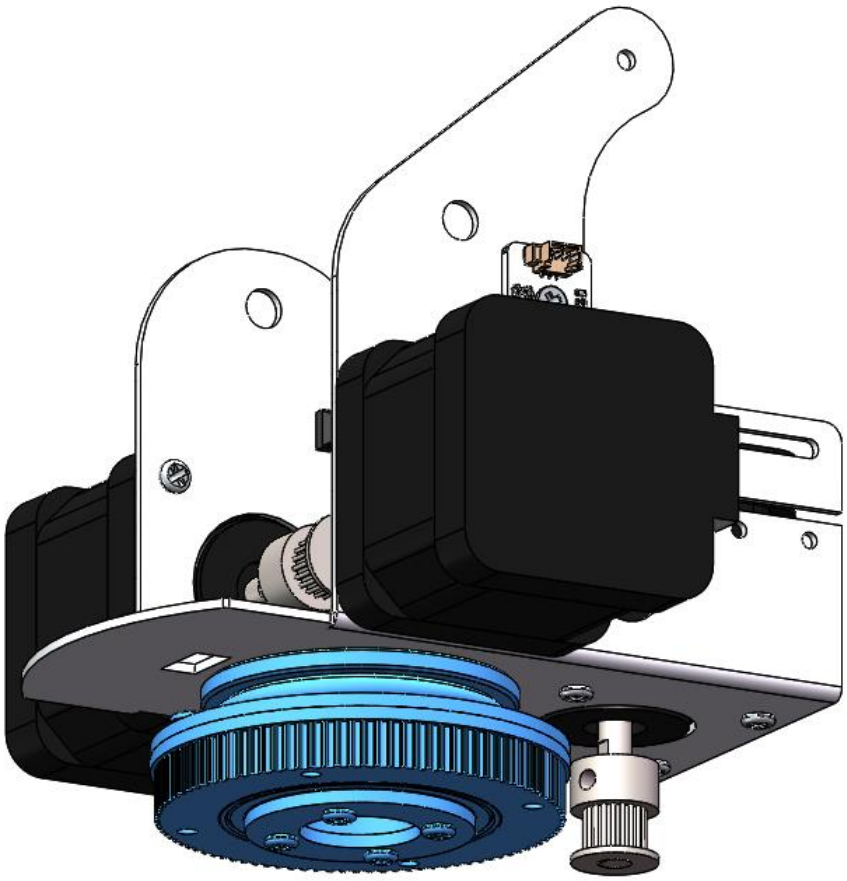
1. Mount the M3x3x7 screw to the No.23 assembly component, as shown below.



The No.23 assembly component is similar to the No.3 one; please do not mix them up.

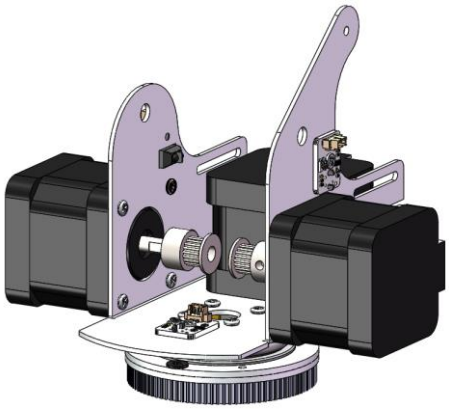
2. Use 4 M3x12 screws to secure the plane thrust bearing and Assembly Components No.23, No.3, and No.19 onto the double-pass copper standoff of Assembly Component No.1.

After the assembly is completed, it should look as shown in the figure below.



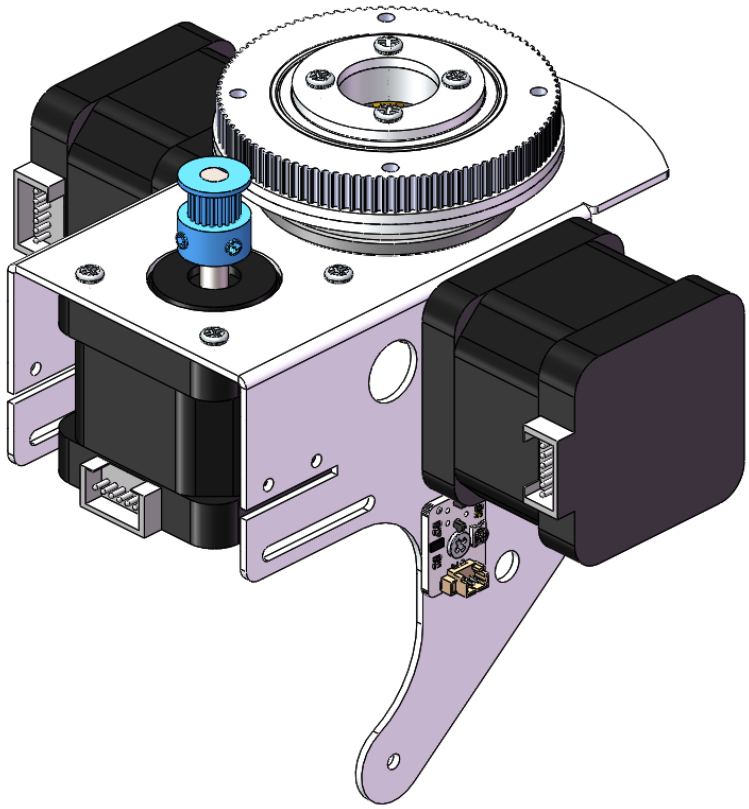
Step 5 Installing the Timing Belt

Materials needed

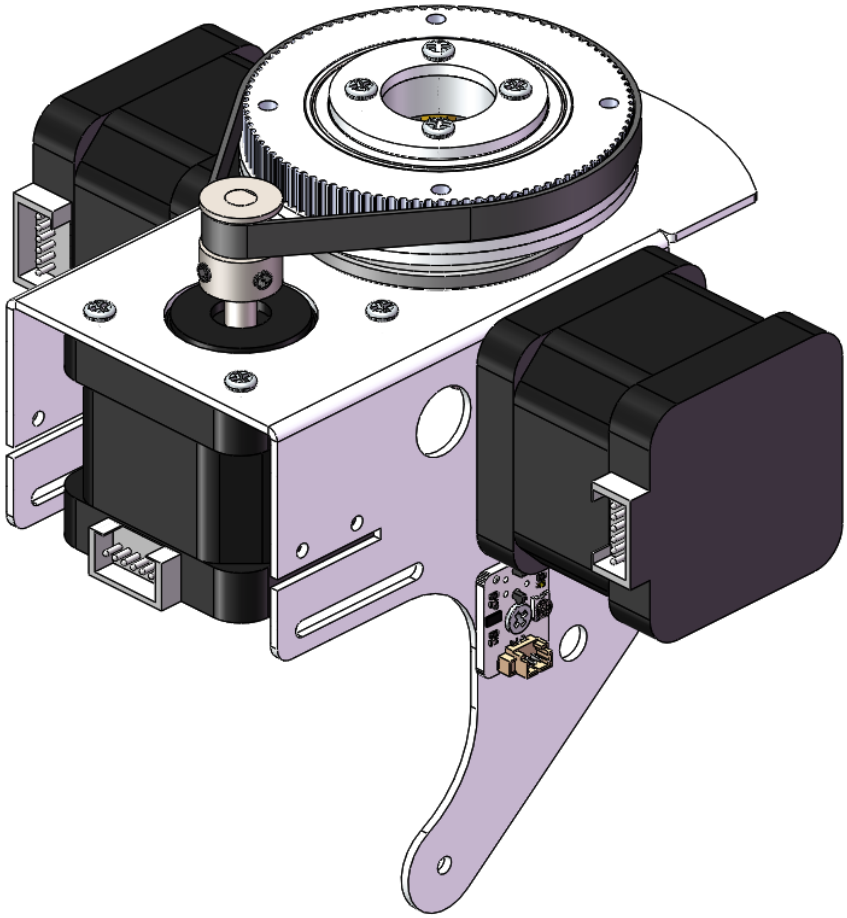
Robot Arm Assembly	2GT-214 Timing Belt
	
Allen Key	
	

Installation Steps

1. Use the Allen key to loosen the bolts on the 2GT timing pulley.

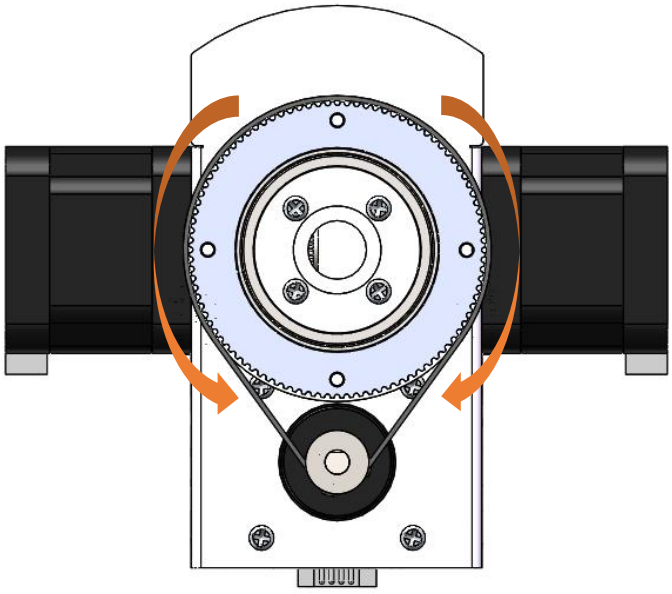


2. Position the timing belt over the 2GT timing pulley and the No.19 assembly component.



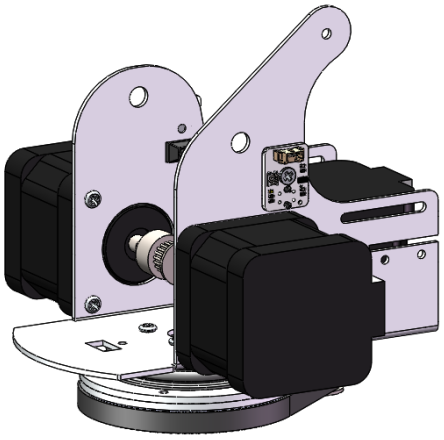
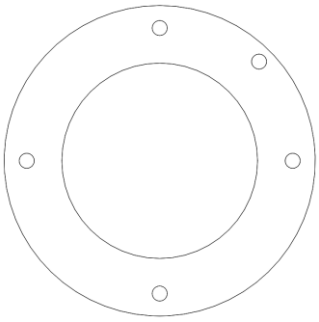
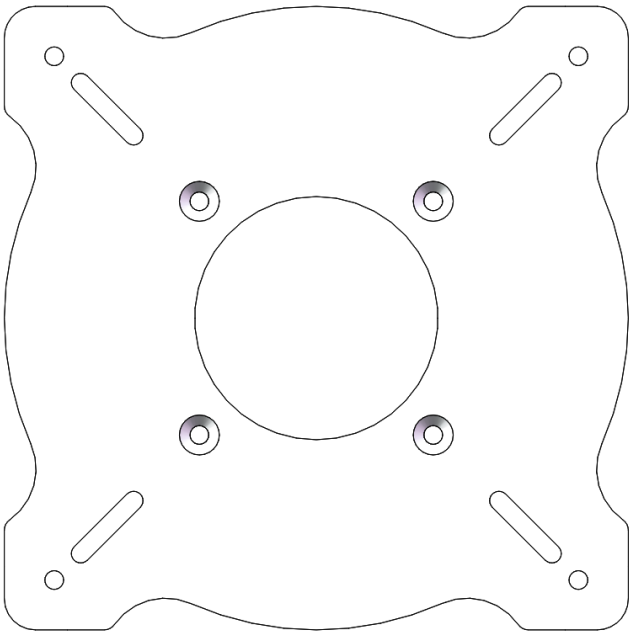
Tip: When installing the timing belt, it is easier to install if you rotate the large gear.

3. Gently rotate the large gear back and forth by hand several times. This motion will allow the timing belt to guide the 2GT timing pulley to the optimal position.
After the pulley has settled into place, use an Allen key to tighten the bolts on the 2GT timing pulley, ensuring it is securely fastened to the shaft.



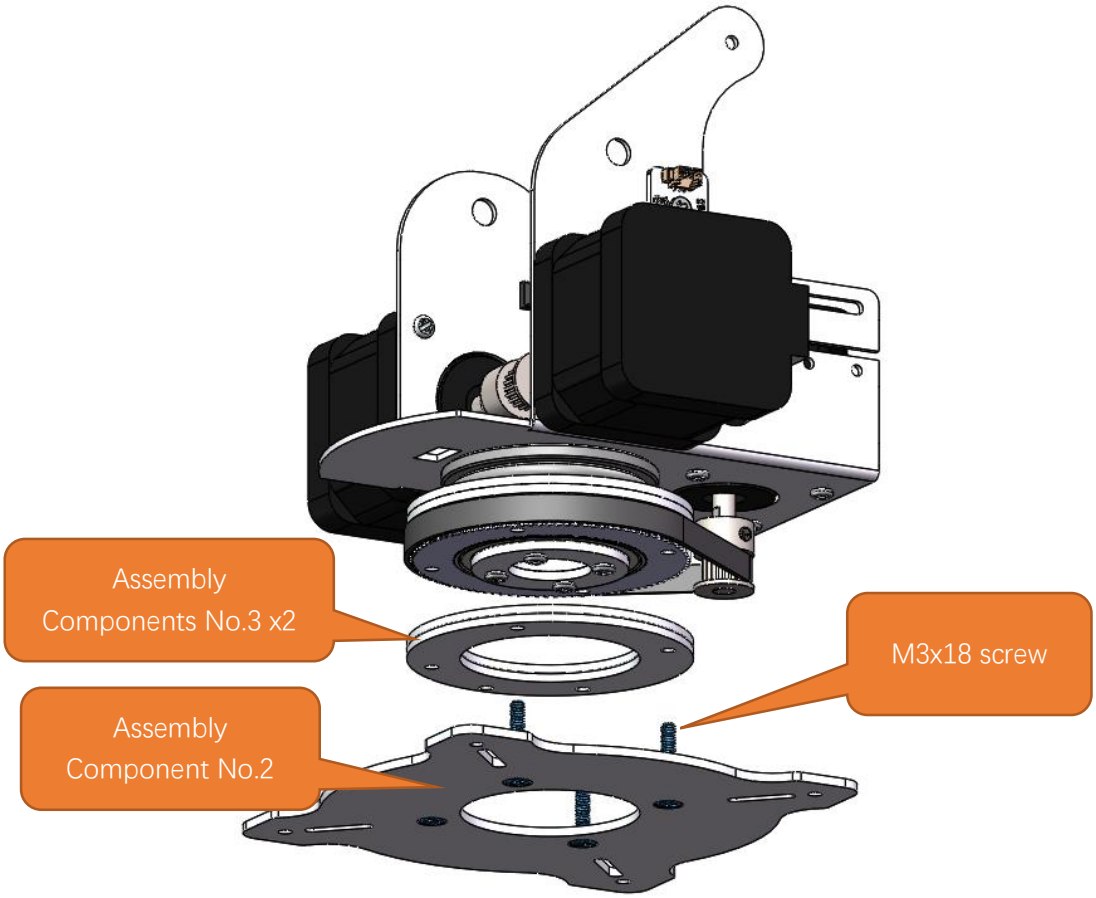
Step 6 Assembling the Base

Material needed

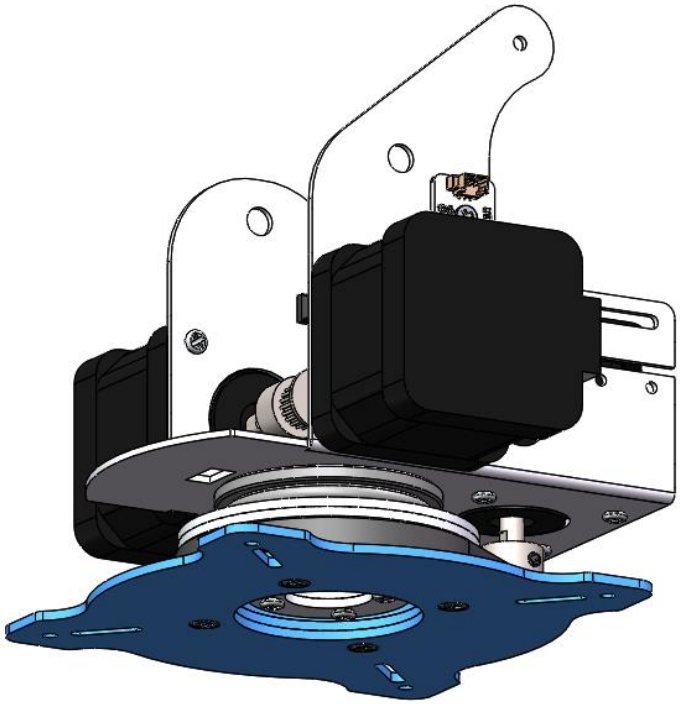
Robot Arm Assembly 	No. 3 Assembly Component X2 
No.2 Assembly Component 	
M3x18 Screws X4 	Cross Screwdriver (3mm) X1 

Installation Steps

Secure two Assembly Components No. 3 and one Assembly Component No. 2 to the large gear using four M3x18 screws.

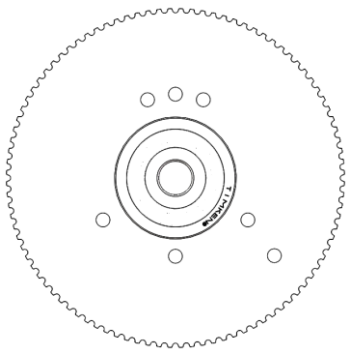
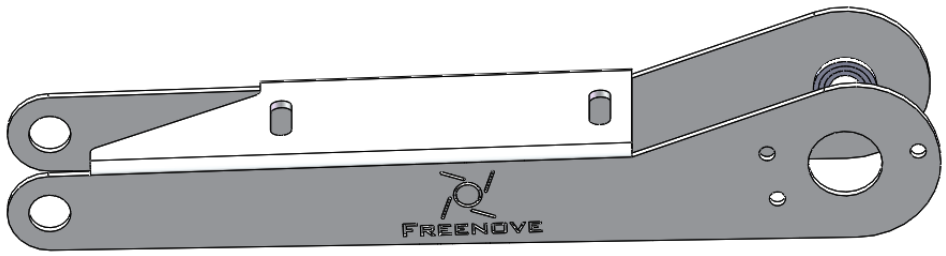


After the assembly is completed, it should look as shown in the figure below.



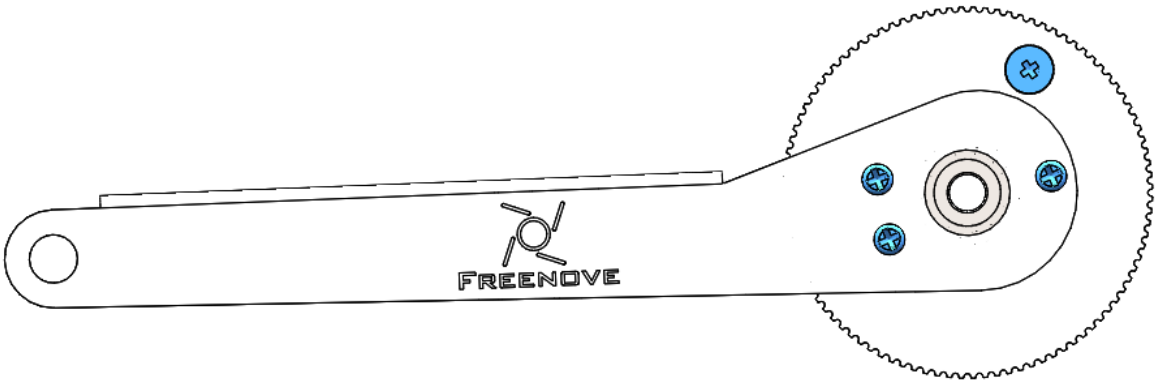
Step 7 Mounting Assembly Components No.8 and No.17

Materials needed

M3x5 Screw X3 	M3x3x7 Screw X1 
No.8 Assembly Component 	Cross Screwdriver (3mm) X1 
Assembly Component No. 17 	

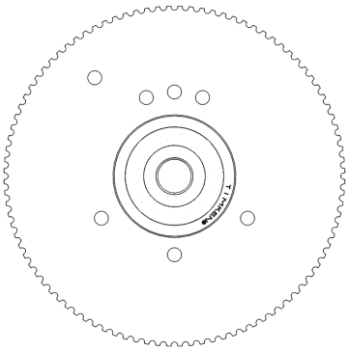
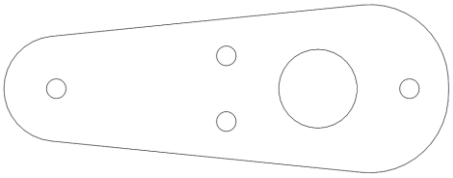
Assembly steps

Attach the Assembly Component No.8 to the Component No.17 with three M3x5 screws, and install one M3x7 black screw into the Component No.8. Ensure the correct alignment and orientation of the components with respect to each other.



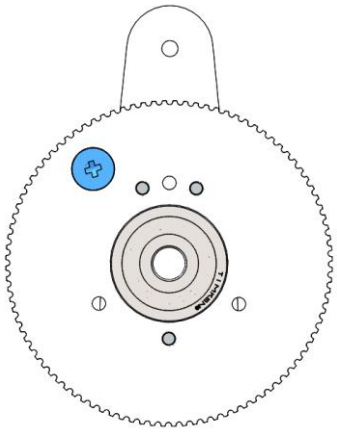
Step 8 Mounting Assembly Components No.7 and No.9

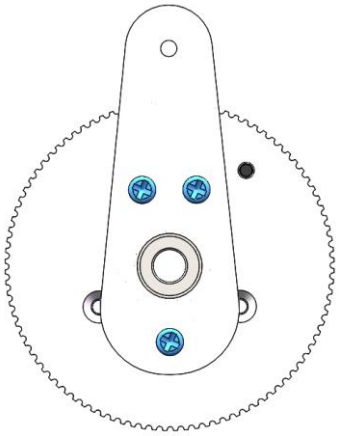
Materials needed

M3x8 Screw X3 	M3x3x7 Screw X1 
Assembly Component No.7 	Assembly Component No.9 
Cross Screwdriver (3mm) x1 	

Installation Steps

Use three M3x8 screws to secure the Component No.9 onto the No.7 one, and apply one M3x3x7 black screw to fasten it onto the Component No.7. Carefully check the alignment and the correct orientation of the components in relation to one another.

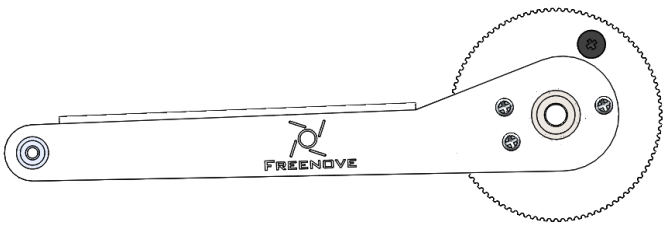
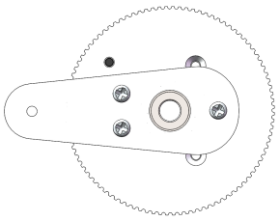
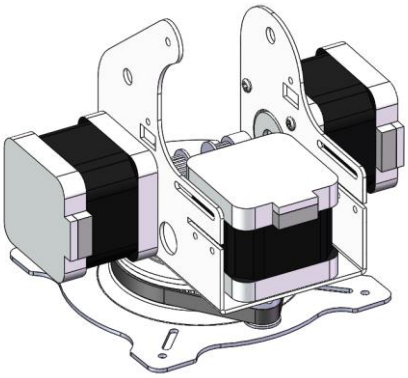




Please note that the M3x3x7 black screw and the Assembly Component No. 9 are not on the same plane as the No.7 part.

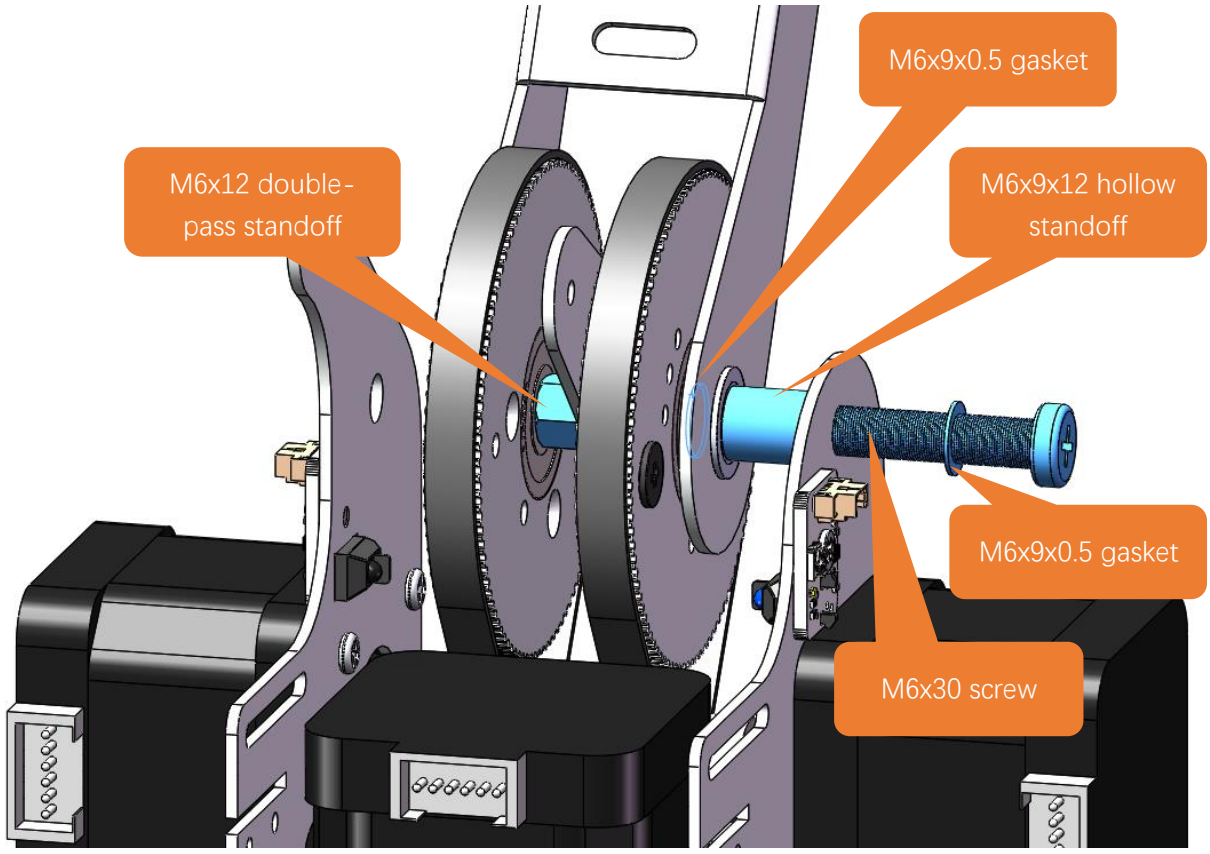
Step 9: Assemble the Left and Right Transmission Devices

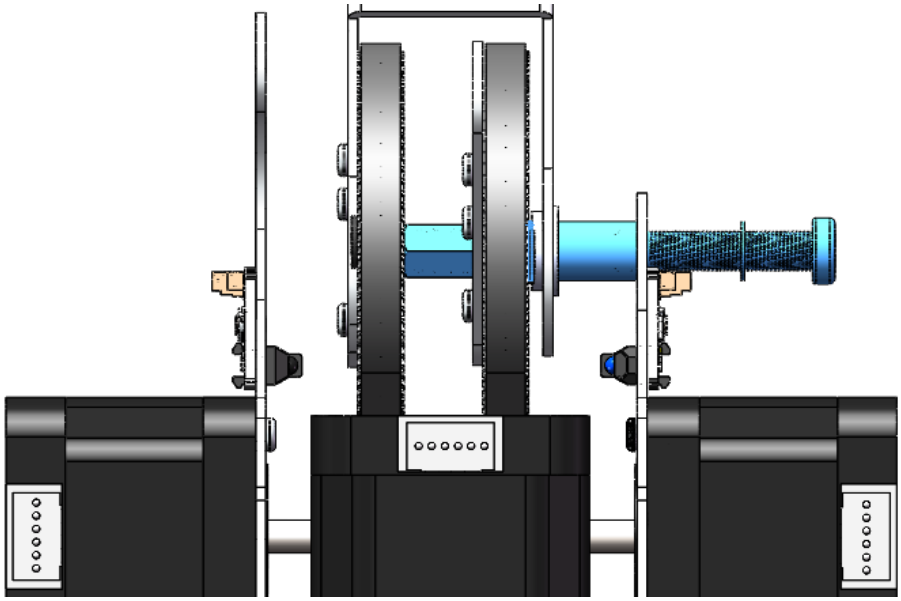
Materials needed

Assembly Part 1 	Assembly Part 2 	
M6x9x12 Hollow Column X1 M6x9x14 Hollow Columnx1 M6x12 Double-Pass Brass Standoff X1 	M6x30 Screw x2 	M6x9x0.5 Flat Gasket x3 
Robot Arm Assembly 	2GT-214 Timing Belt x2 	
Cross Screwdriver (3mm) x1 	8mm Open-end Wrench 	Allen Key 

Installation Steps

1. Secure the Assembly Part 1 and Assembly Part 2 to the right side of the main body with one M6x30 screw, one M6x12 double-pass brass standoff, one M6x9x12 hollow standoff, and two M6x9x0.5 gasket. The timing belt should be placed around the large gear on the right side, as shown in the figure below.

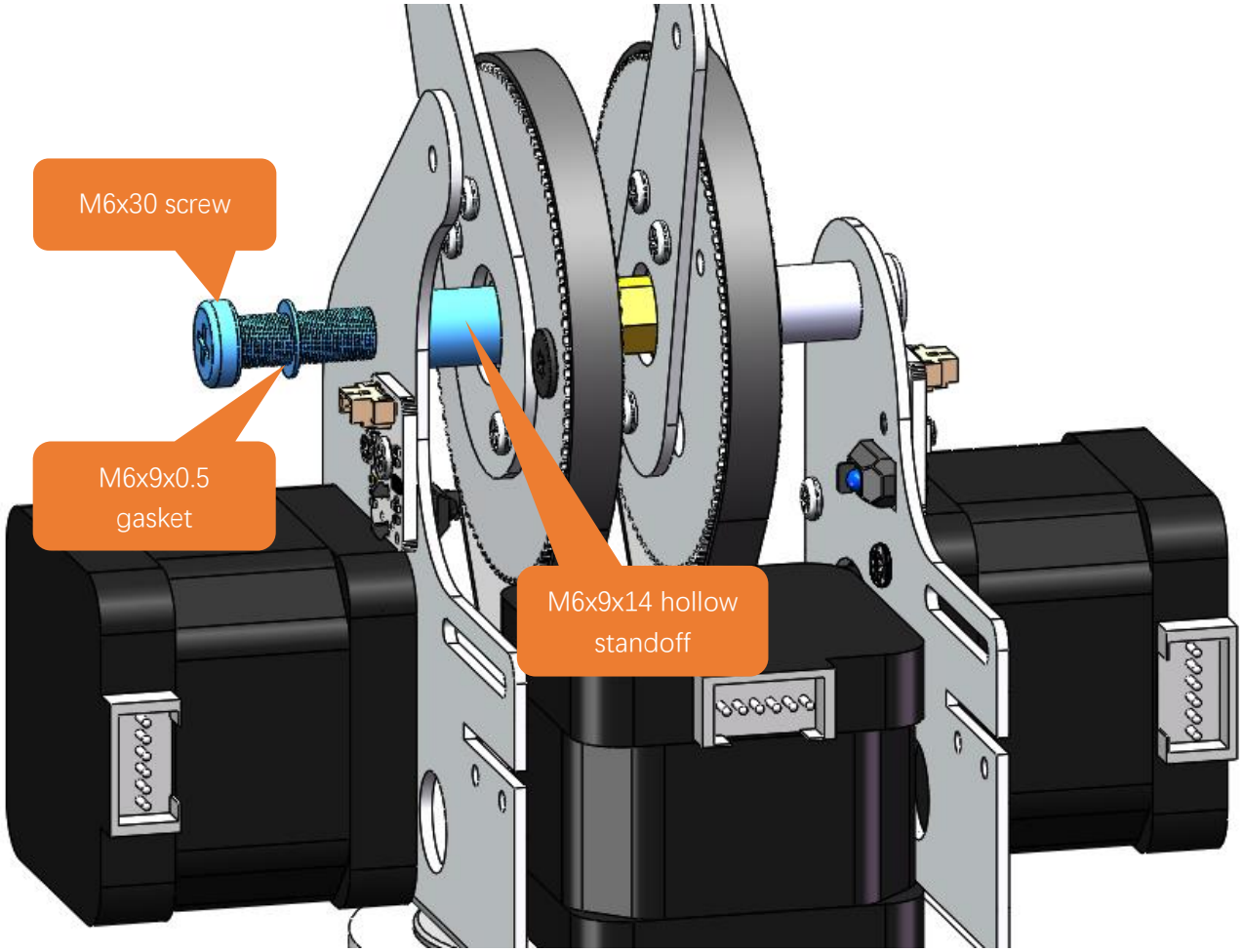




Notes:

1. Before starting the assembly process, ensure that the timing belt is already fitted around the large gear of Assembly Part 2.
2. Pay attention to the orientation of the body, Assembly Part 1, and Assembly Part 2 to avoid installation errors.
3. This step is relatively complex, and irrelevant parts have been hidden in the simulation diagram to avoid obstructing the view of the simulation.
4. Do not tighten the M6x30 screws first.

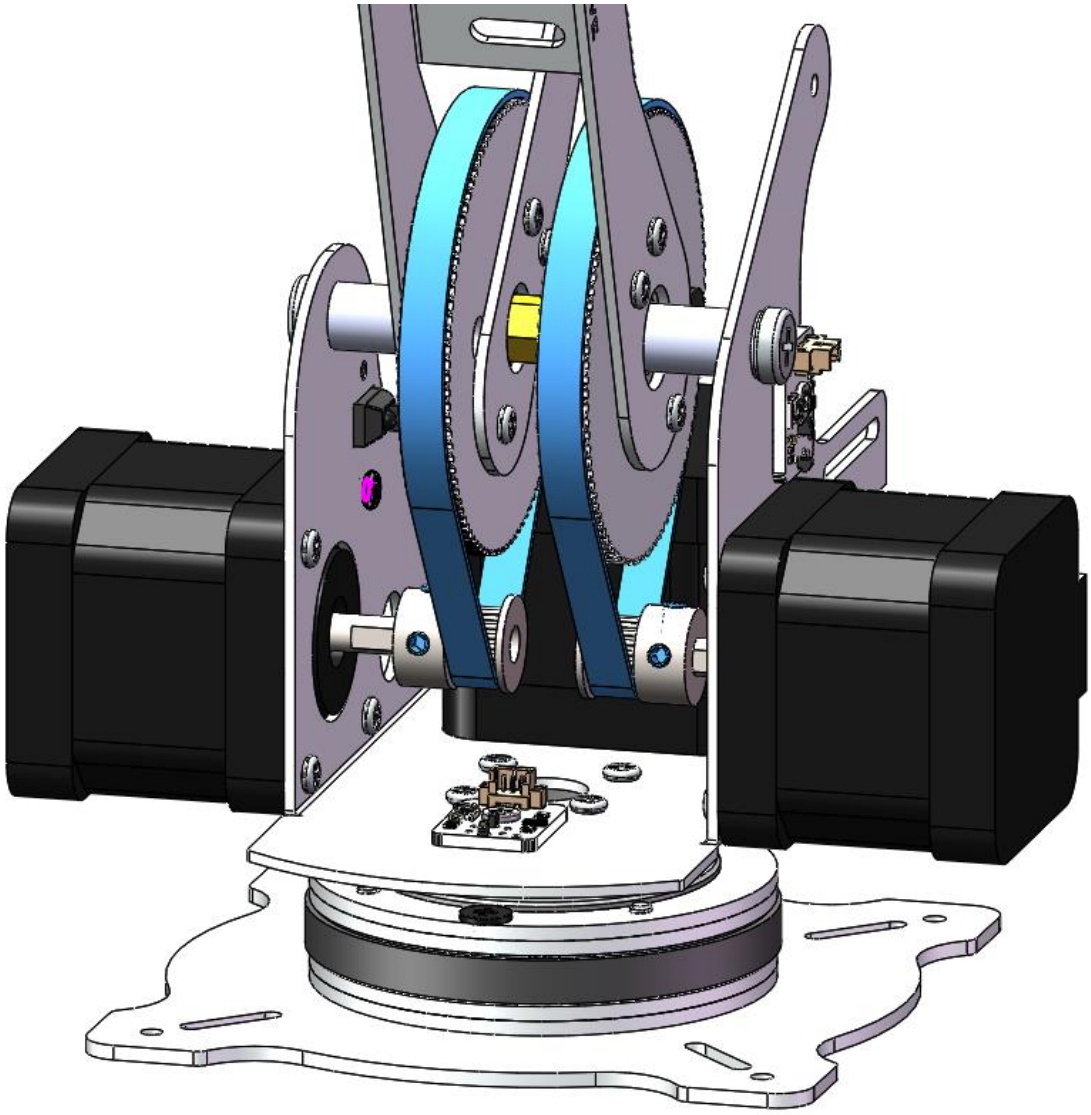
2. Mount Assembly Part 1 to the left side of the main body wtih one M6x30 screw, one M6x9x14 hollow standoff, and one M6x9x0.5 gasket, as shown in the figure below.



Notes:

1. Before starting the assembly process, ensure that the timing belt is already fitted around the large gear of Assembly Part 1.
2. Pay attention to the orientation of the body, Assembly Part 1, and Assembly Part 2 to avoid installation errors.
3. This step is relatively complex, and irrelevant parts have been hidden in the simulation diagram to avoid obstructing the view of the simulation.
4. Tighten two M6x30 screws.

3. Use an Allen key to adjust the bolts on the timing pulley, allowing the pulley to move left and right. Slide the timing belt onto the timing pulley and rotate the large gear back and forth to ensure proper positioning of the belt. Once the belt is correctly seated, tighten the bolts to secure the timing pulley in place.



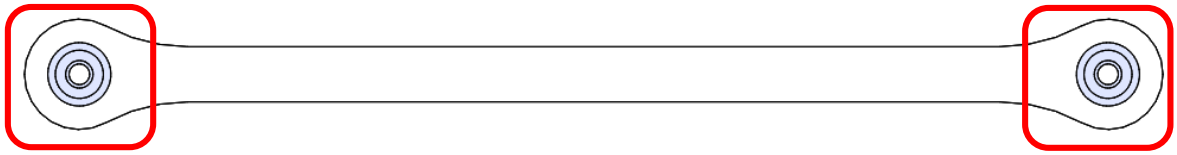
Two approaches can be considered for installing the timing belt:

1. **Loosening and Positioning Approach**
   - Start by using an Allen key to loosen the bolts on the timing pulley, allowing for some movement.
   - First, position the timing belt around the large gear of the assembly part.
   - Next, slide the other end of the timing belt onto the timing pulley.
   - Once both ends of the belt are in place, tighten the bolts to secure the timing pulley.
2. **Pre-Routing and Adjustment Approach:**
   - Begin by placing the timing belt on the timing pulley.
   - Then, route one end of the timing belt to the large gear of the assembly part.
   - After most of the belt has been positioned around the large gear, rotate the large gear on the assembly part.
   - This action will guide the timing belt into place on both the large gear and the timing pulley.

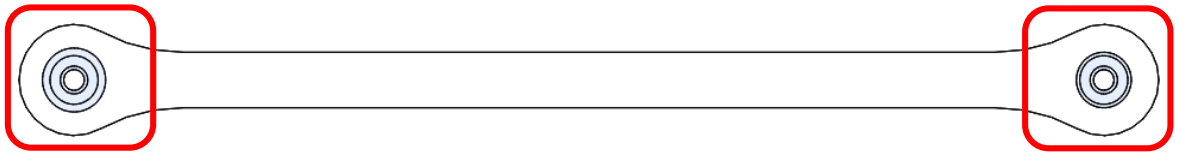
Step 10 Mounting Assembly Components No.10 and No.12

Before we begin, let's clarify the differences between Assembly Components n 10, 11, and 12:

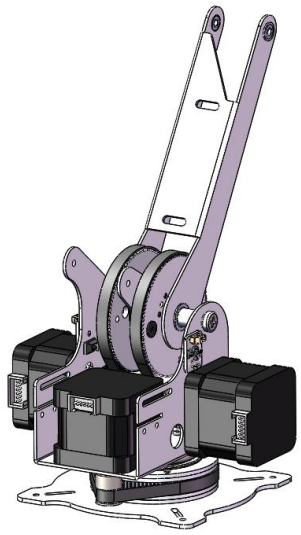
Components 10 and 11: The two bearings orient in the same direction.



Component 12: The two bearings orient to opposite directions.

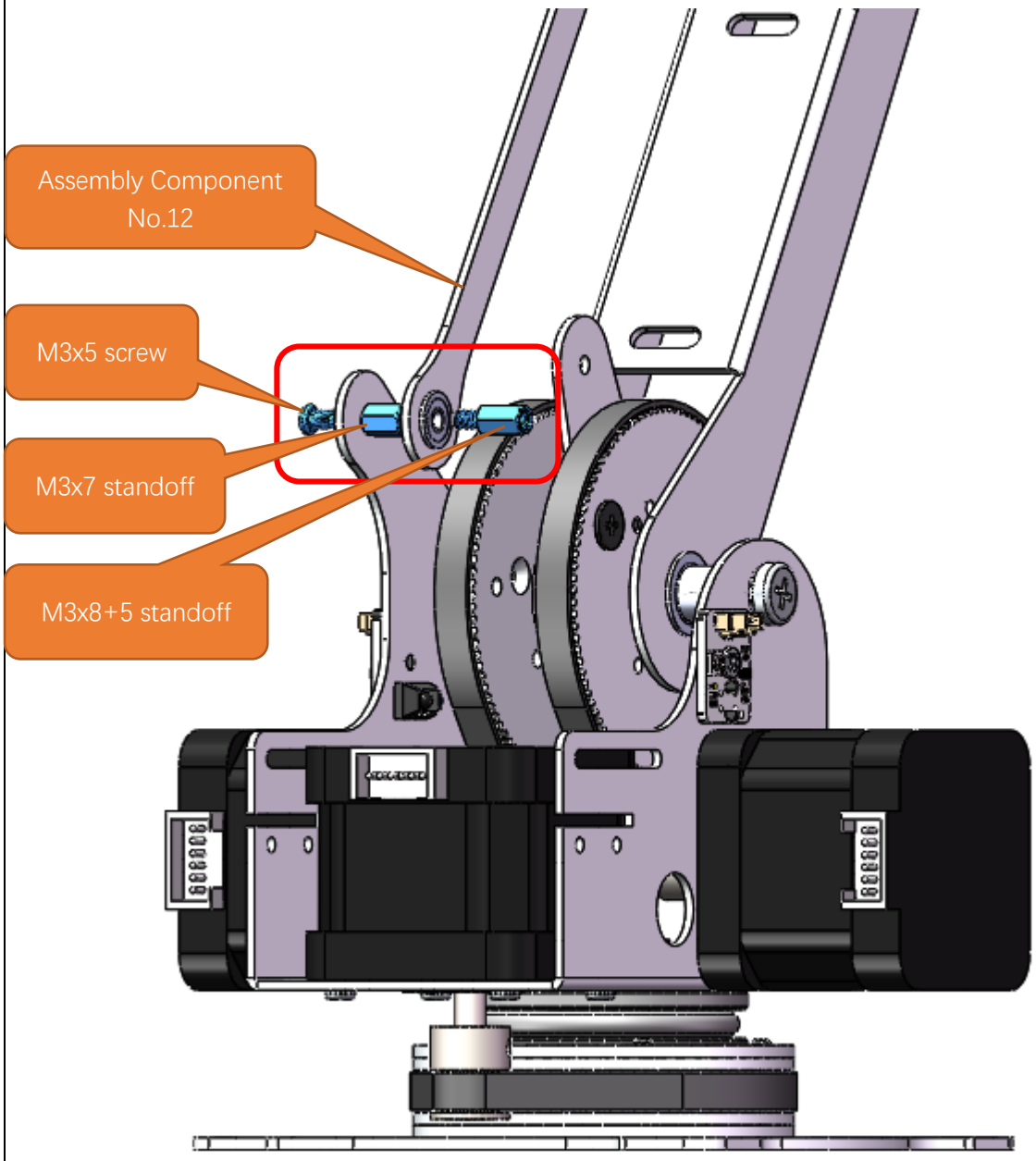


Material Needed

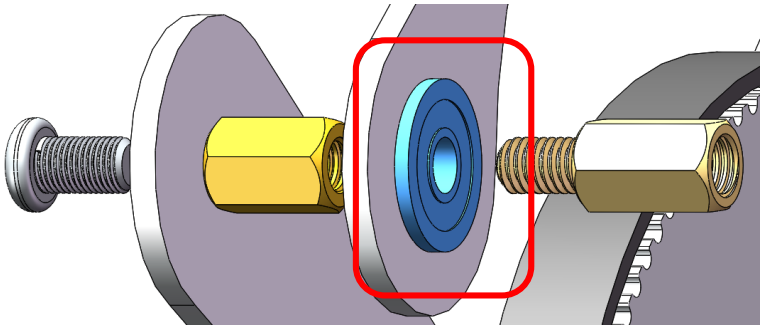
M3x5 screws x3	M3x7 standoff x1、M3x8+5 standoff x1,M3x9 standoff x1	
		
Robot Arm Assembly	Assembly Component No.10	Assembly Component No.12
		

Installation Steps

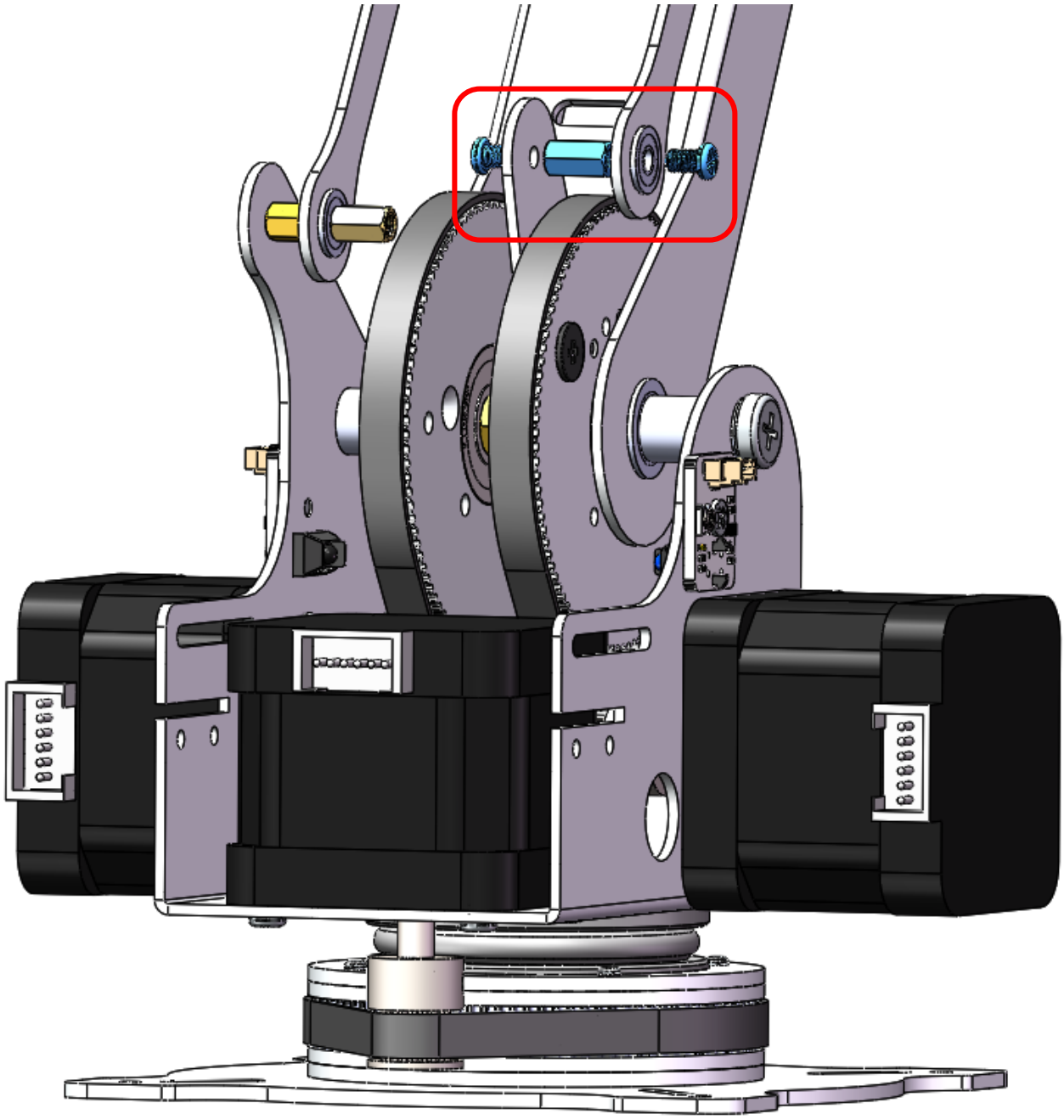
1. Affix Assembly Component No. 12 to the left side of the robotic arm's main structure using a M3x5 screw, an M3x7 standoff, and an M3x8+5standoff, as depicted in the accompanying diagram.



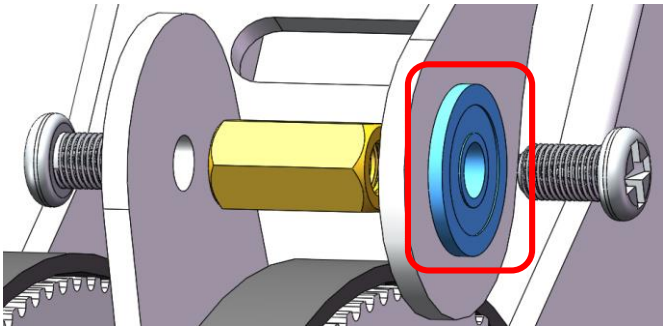
Please note that the end of Assembly Component No. 12 with the protrusion should face to the right, as illustrated in the figure below.



2. Affix Assembly Component No. 10 to the right side of the robotic arm's main structure using two M3x5 screws and an M3x9 standoff, as depicted in the accompanying diagram.



Please note that the end of Assembly Component No. 10 with the protrusion should face to the right, as illustrated in the figure below.

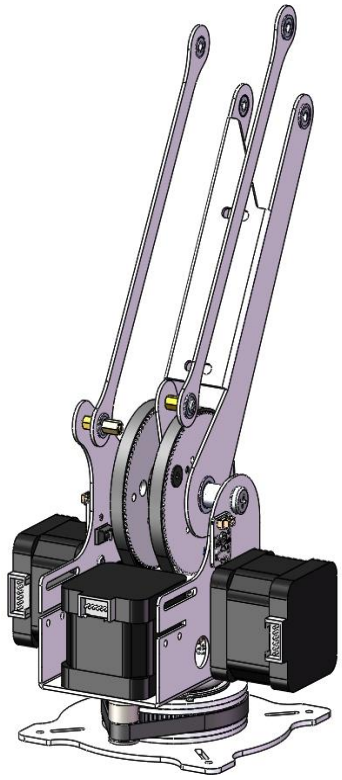


Step 11 Mounting Assembly Components No.11, No.4 and No.18

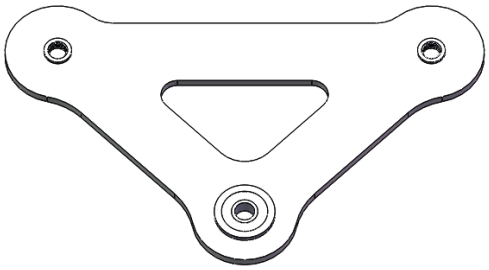
Materials Needed

M3x5 Screw x4	M3x8 Screw x1	M3x5x0.5 Gasket x1
		

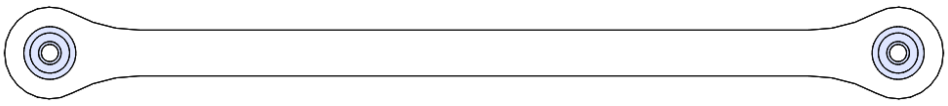
Robot Arm Assembly



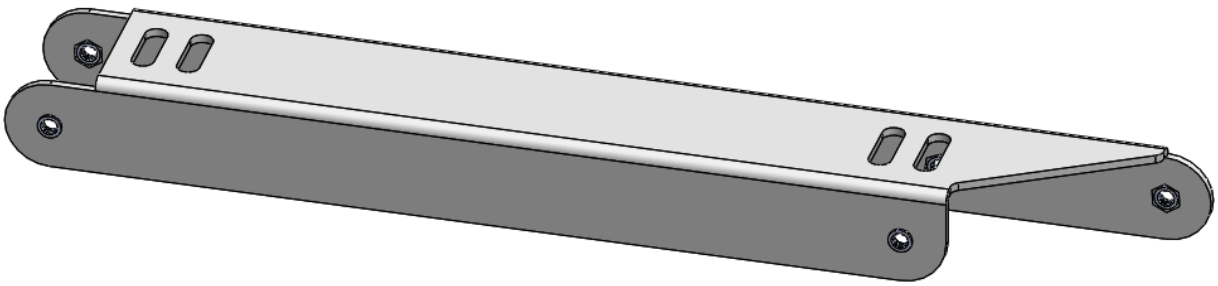
Assembly Component No.4



Assembly Component No.11

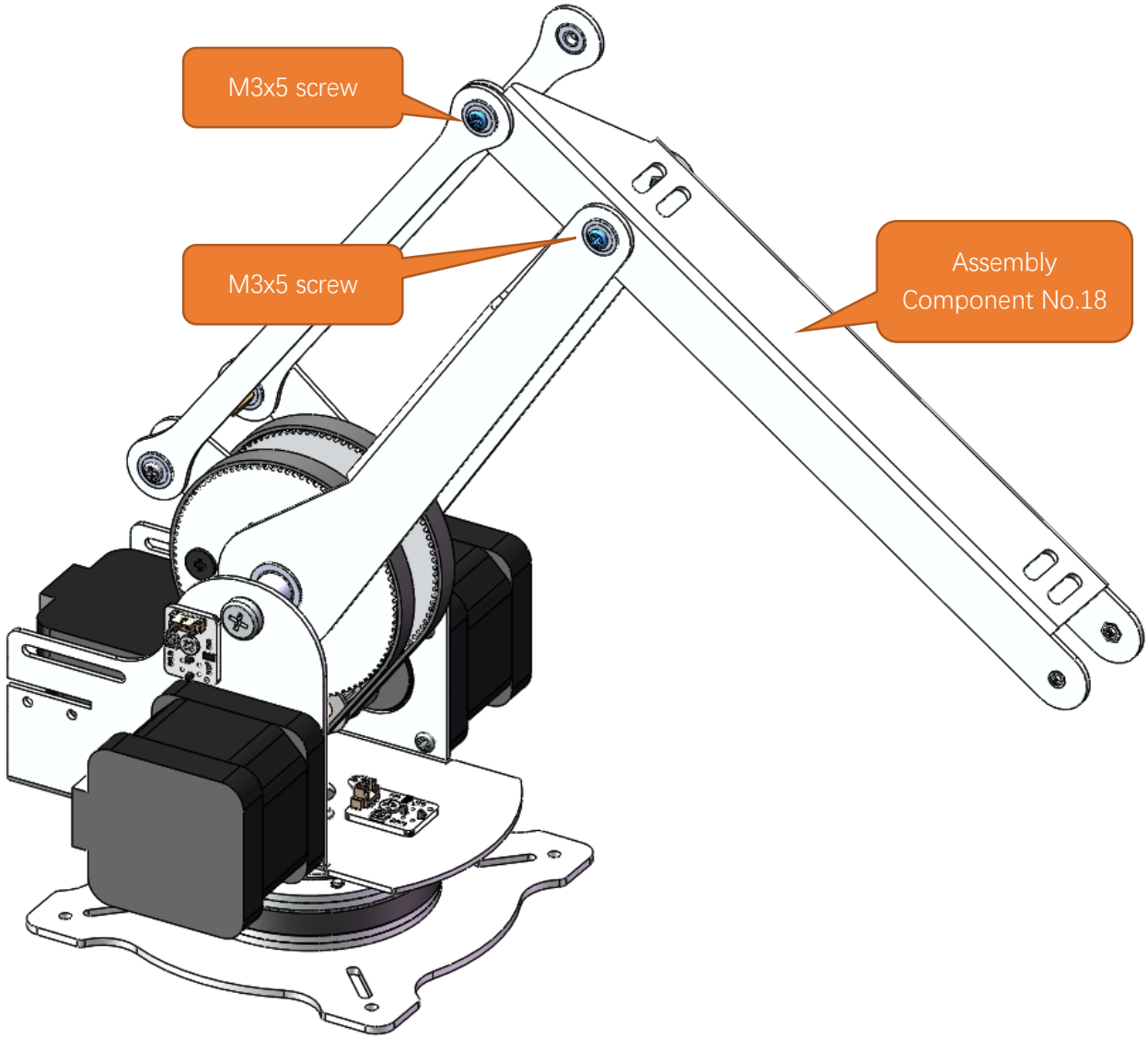


Assembly Component No.18



Installation Steps

1. Mount Assembly Component No.18 to the robot arm assembly with two M3x5 screws.



2. Mount Component No. 4 to the robotic arm assembly with one M3x5 screw, one M3x8 screw, and one M3x5x0.5 gasket. **Please ensure the correct orientation of Component No. 4.**

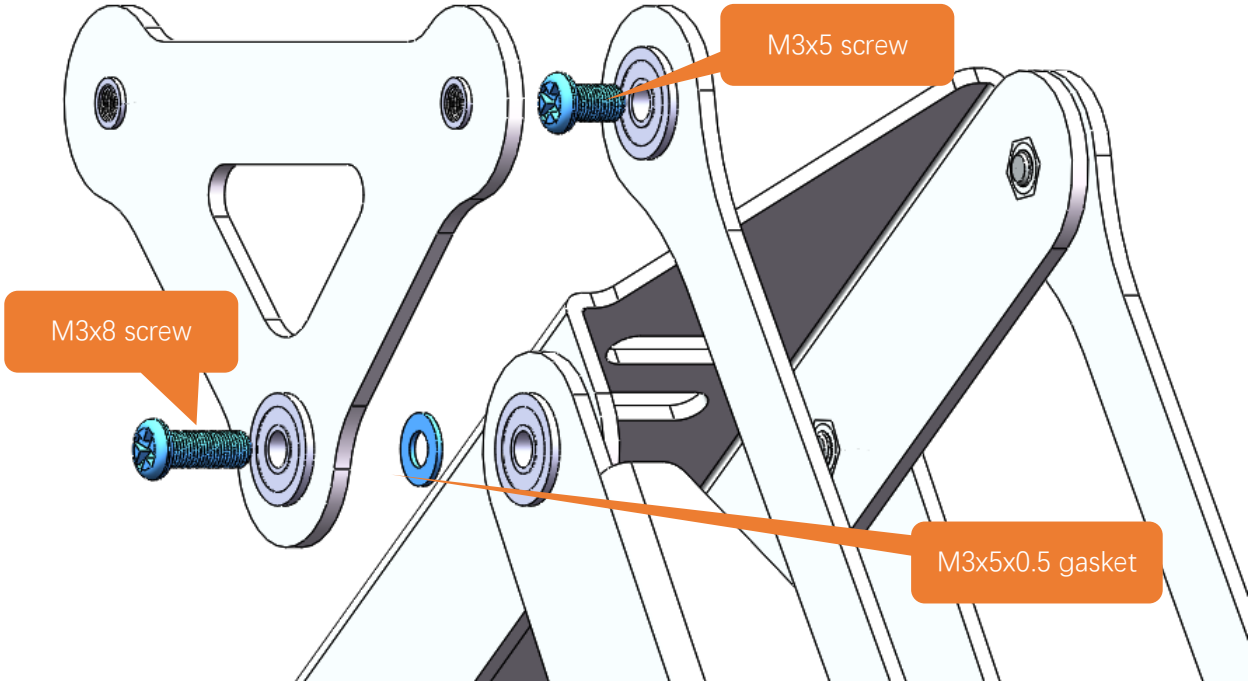
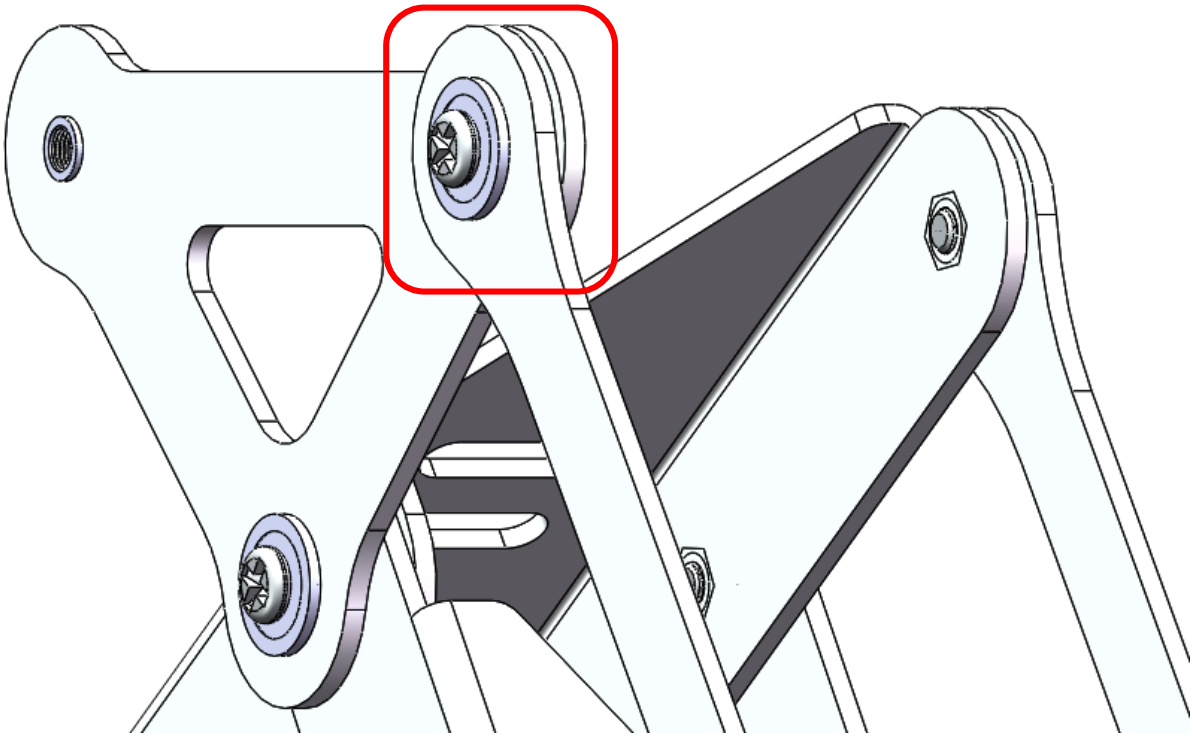
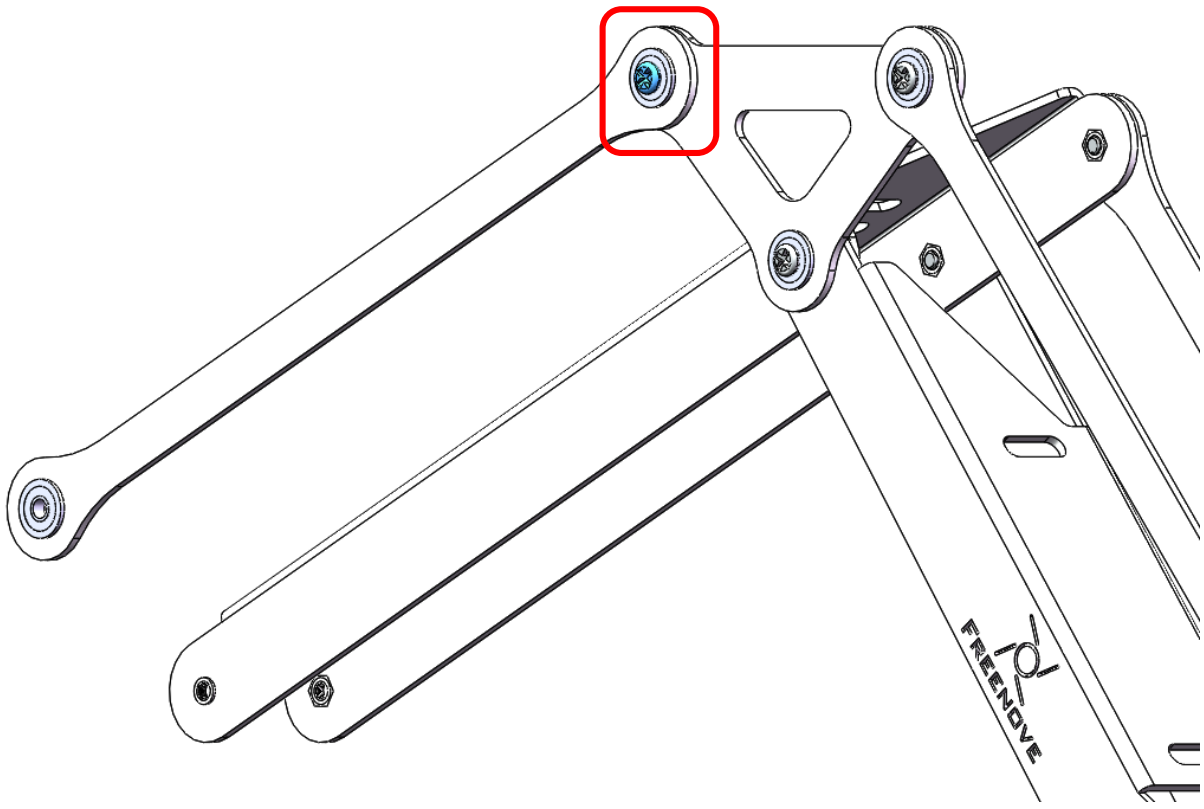


Diagram of the completed assembly). Please pay attention to the installation of the positions circled in the figure below.

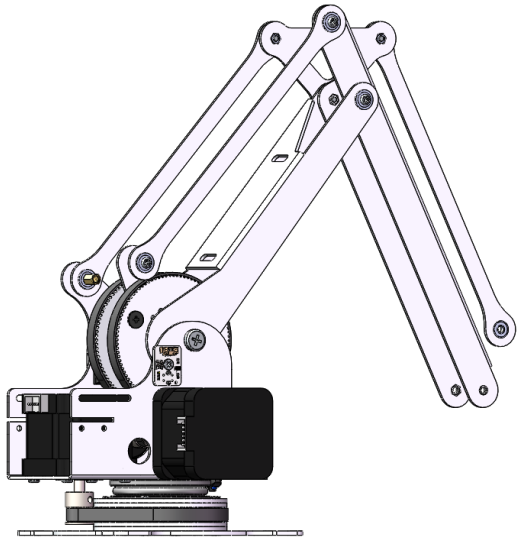
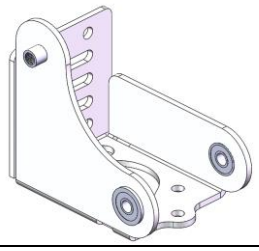


3. Attach Assembly Component No. 11 to Component No. 4 with one M3x5 screw. **Please ensure the correct orientation of Component No. 11.**



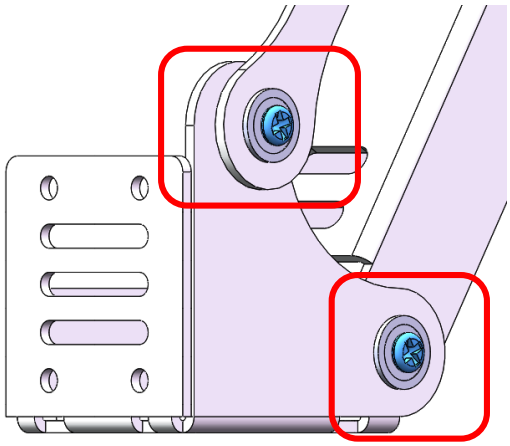
Step 12 Mounting Assembly Component No.20

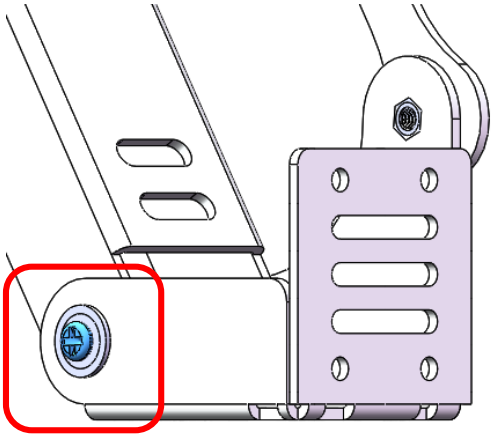
Materials Needed

Robot Arm Assembly	Assembly Component No.20
	
	M3x5 Screws x3 

Installation Steps

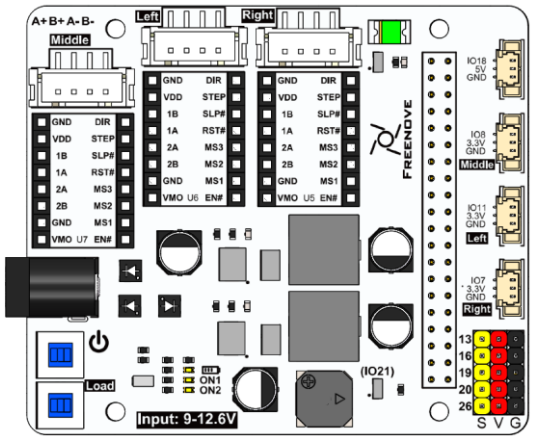
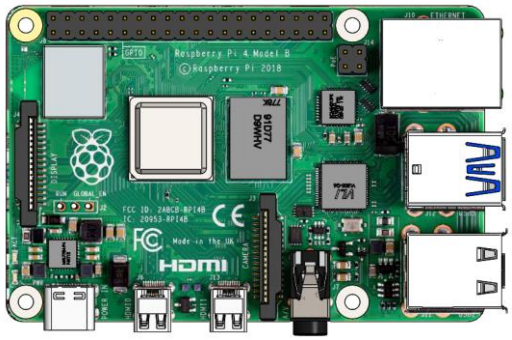
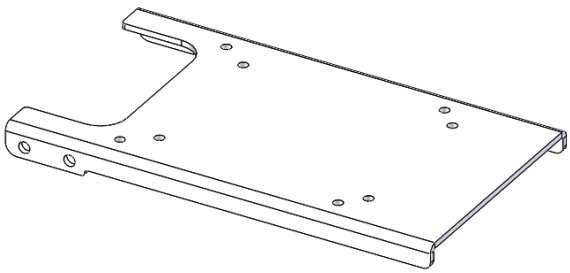
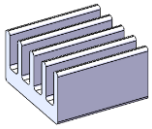
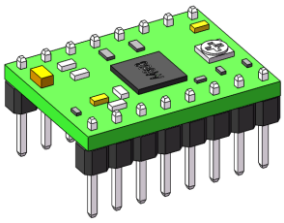
Attach the Assembly Component No. 20 to Components No.11 and No.18 with three M3x5 screws.





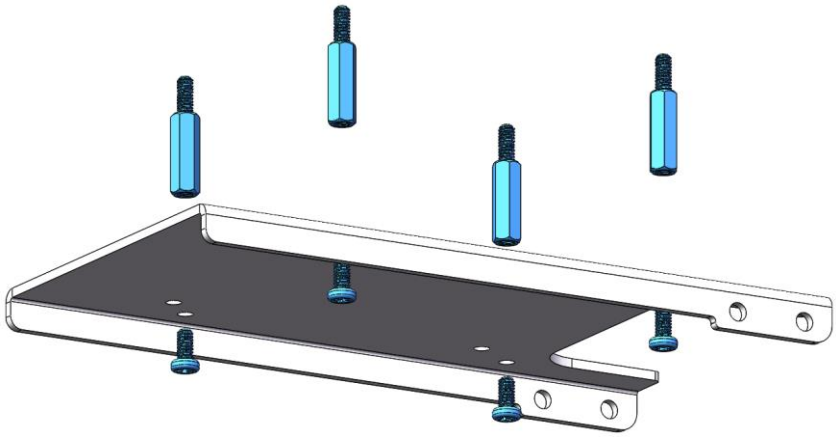
Step 13 Mounting Raspberry Pi to Robot Arm Board

Materials Needed

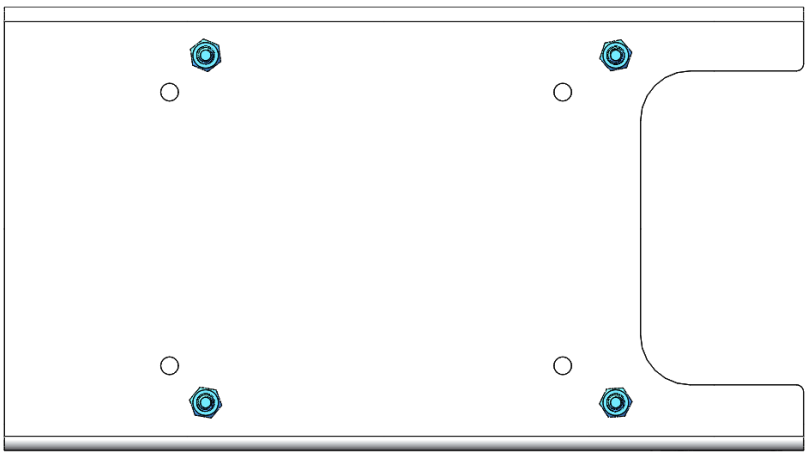
M2.5x8 Screws x4 	M2.5x13+6 Brass Standoffs x4 	M2.5x11+6 Brass Standoffs x4 	M2.5x25 Brass Standoff x4 
Robot Arm Board 	Raspberry Pi (Not included in the kit. Please prepare one yourself.) 		
	Cross Screwdriver (3mm) x1 		
Assembly Component No.14 	Heat Sinks x3 		
	Stepper Motor Driver Modules x3 		

Installation Steps

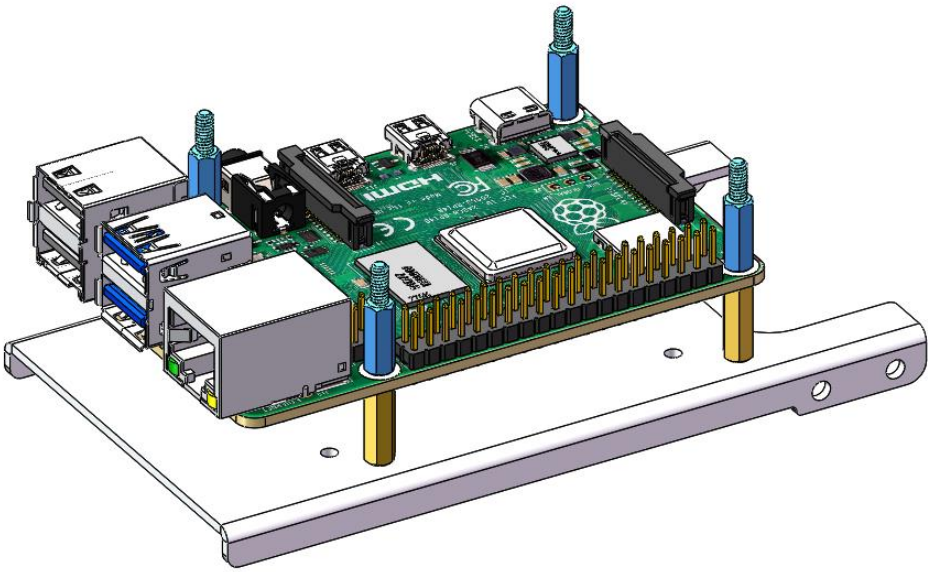
Fix four M2.5x13+6 brass standoffs to Assembly Component NO.14 with four M2.5x8 screws.



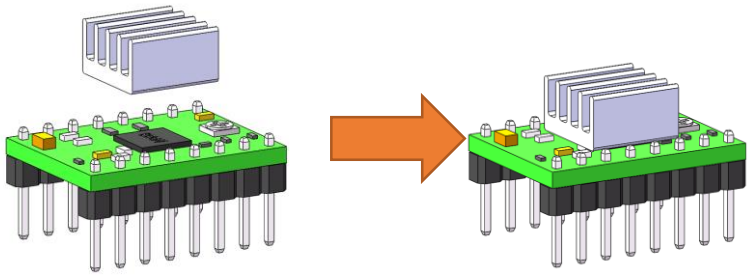
Please note: Ensure that you do not install the screws into the wrong holes. The hole positions are shown in the figure below.



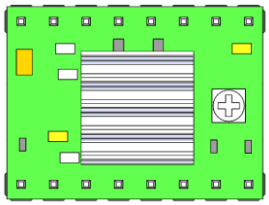
Mount the Raspberry Pi with four M2.5x11+6 standoffs. Pay attention to the orientation of the Raspberry Pi.



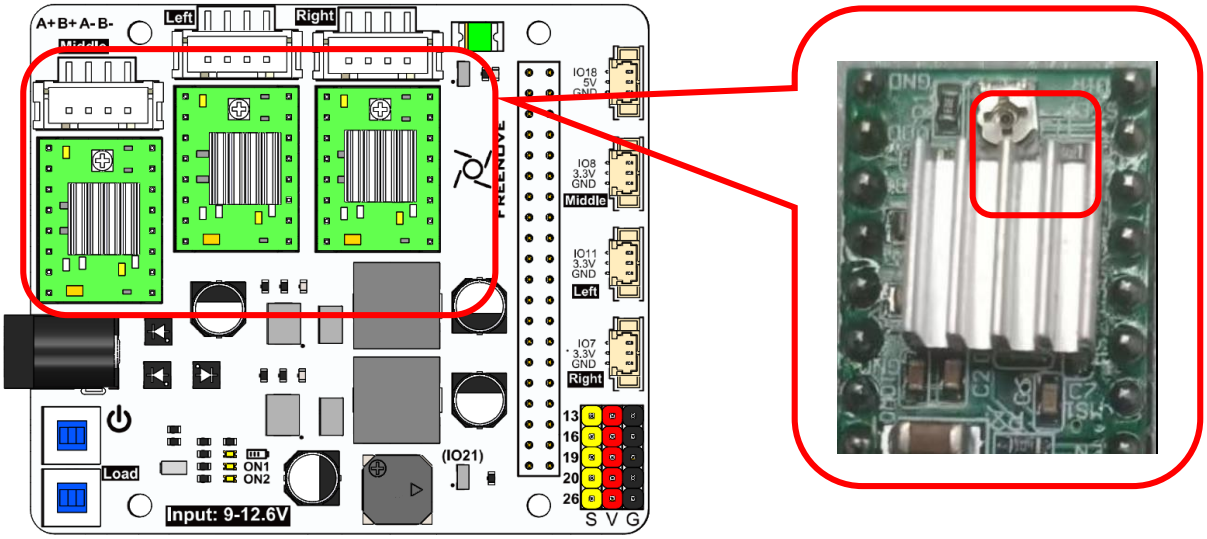
Peel off the adhesive backing from the back of the heatsink and adhere it to the chip on the stepper motor driver module. Please be careful not to touch the pins on either side.



Top View



Plug the three stepper motor driver module onto Robot Arm Board.

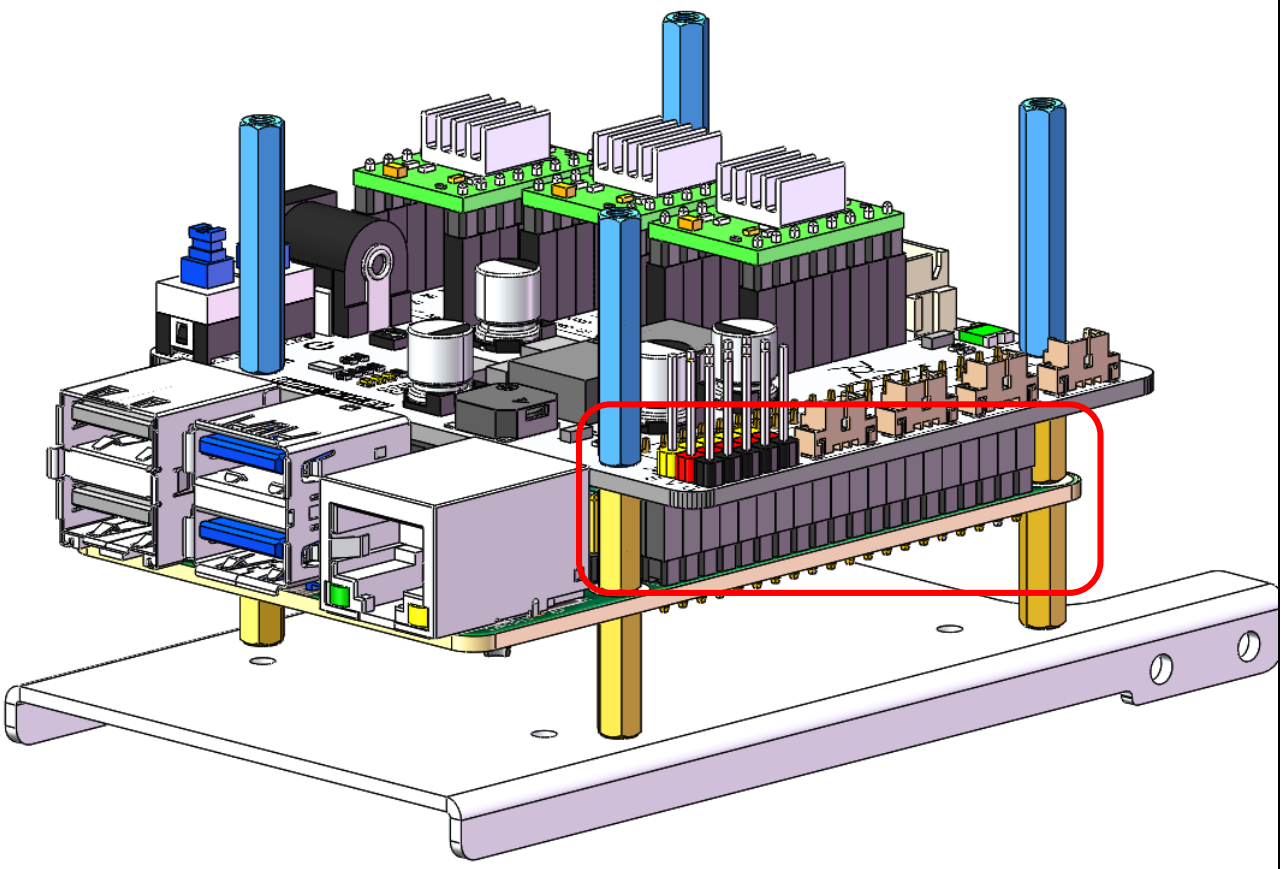


Note:

1. The orientation of the stepper motor driver module must not be reversed, as it may damage the circuit board.

2. Using a screwdriver, rotate the resistors on all three stepper motor driver modules to the position shown in the diagram above.

Mount the Robot Arm Board to Raspberry Pi with four M2.5x8 screws.



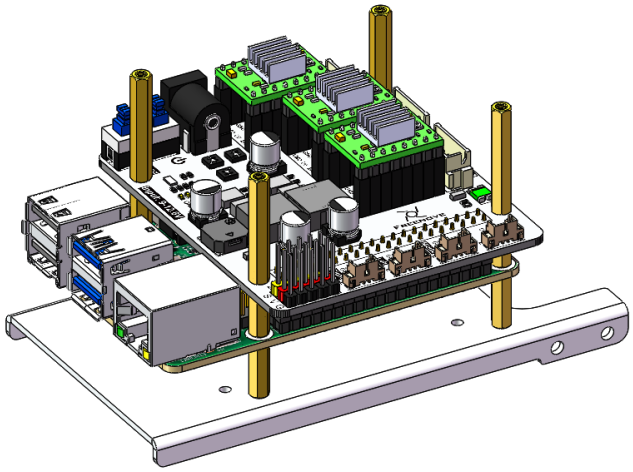
Notes:

1. **Please pay attention to the orientation of the Robot Arm Board.**
2. **Ensure that the pins between the Robot Arm Board and the Raspberry Pi are connected one-to-one.**

Step 14 Installing the Battery Holder

Materials Needed

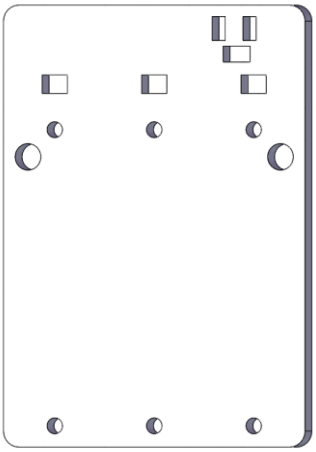
Assembly Component No.14



Battery Holder

Cross Recessed Countersunk Screw(M3x12)

Tailboard x1



Installation Steps

Attached the battery holder, tailed acrylic board to the robot arm assembly with four M3x12 screws.

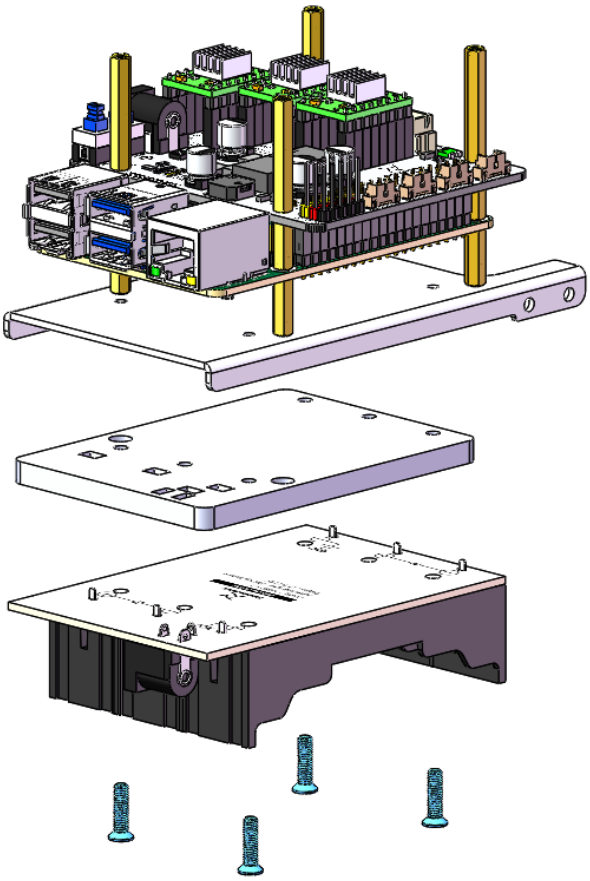
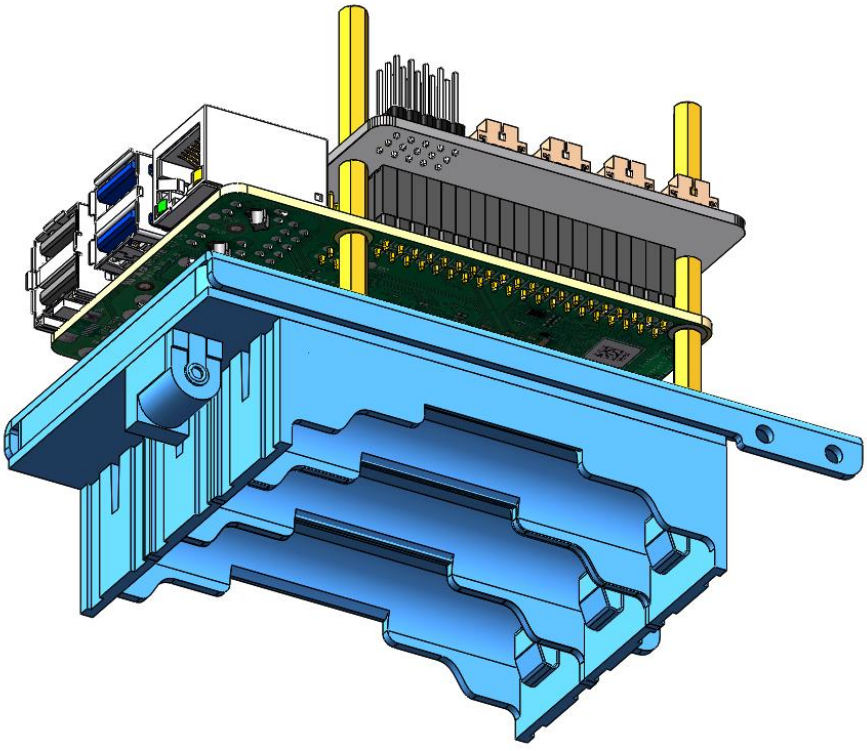


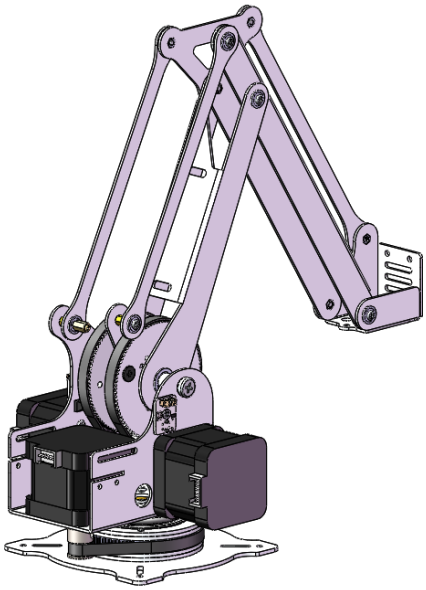
Diagram of the completed assembly



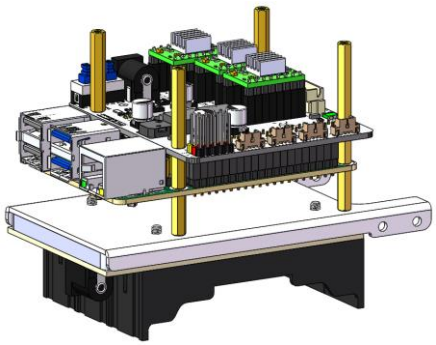
Step 15 Mouting Assembly Components No. 14

Materials Needed

Robot Arm Assembly



Assembly Component No.14

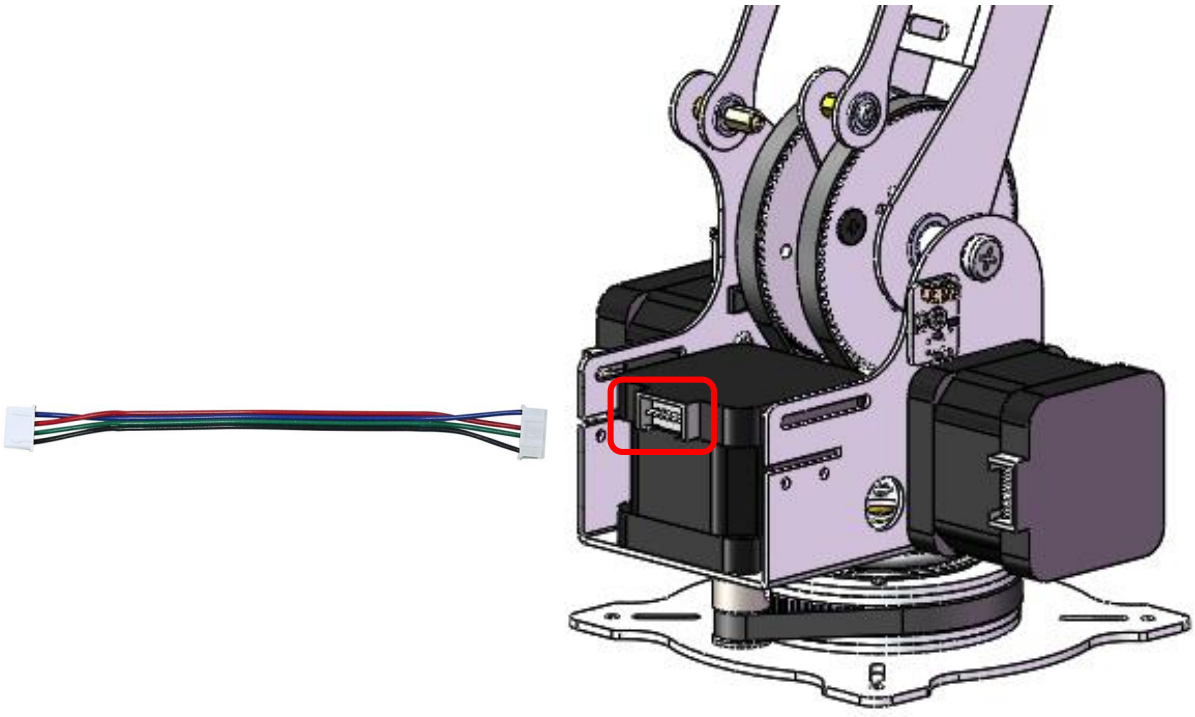


M3x5 Screws x4

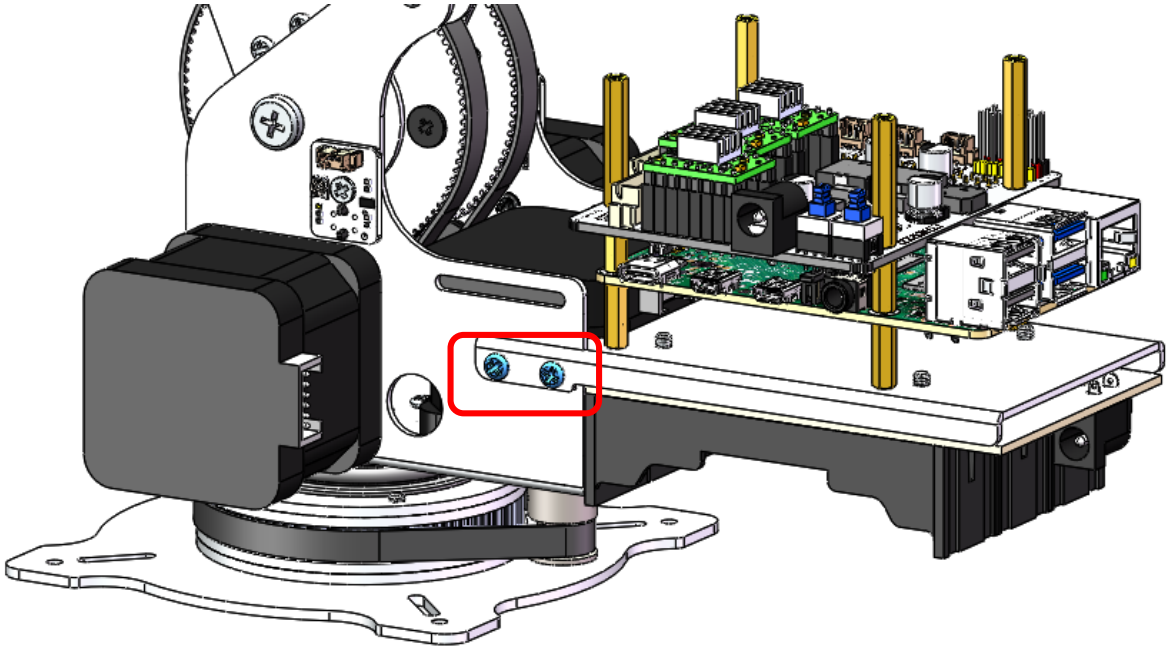
Cable for Stepper Motor x1

Installation Steps

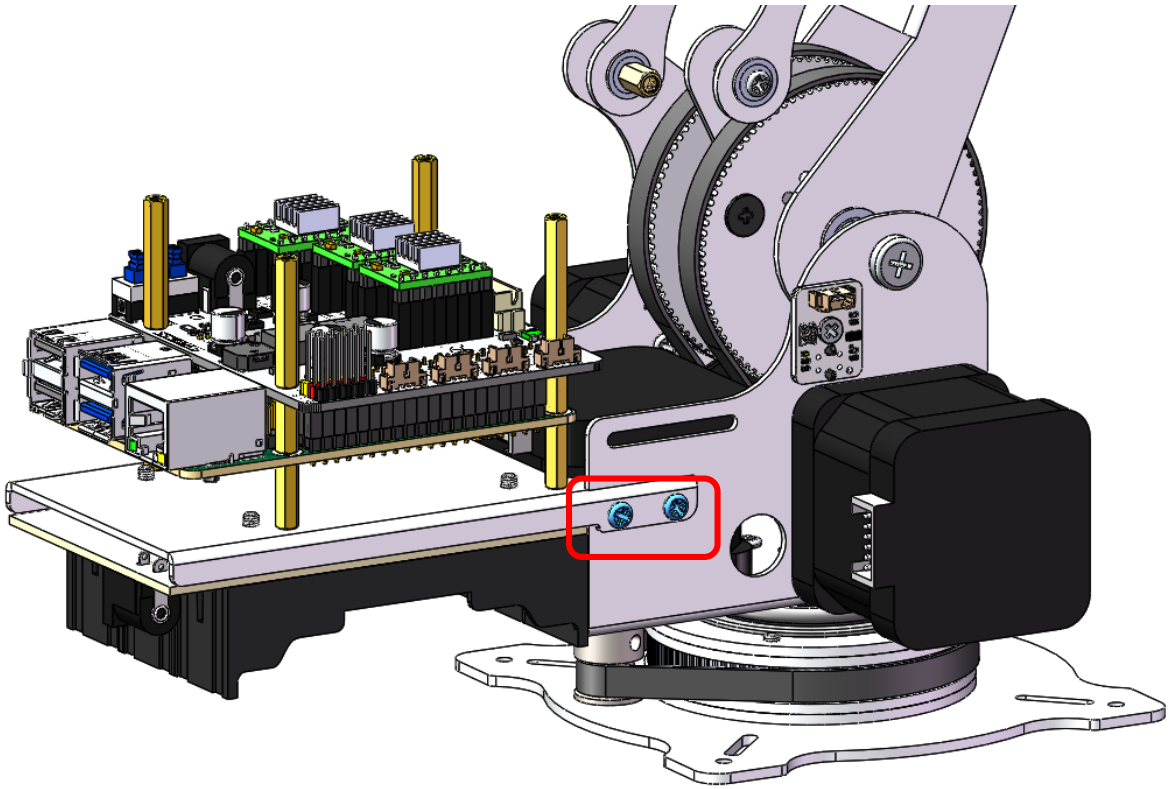
For easier wiring in the future, we recommend that you first connect a wire to the stepper motor.



Use two M3x5 screws to secure the left side of Assembly Component No. 14 onto the robotic arm assembly.

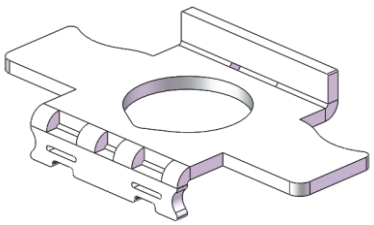
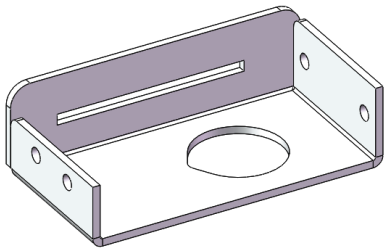
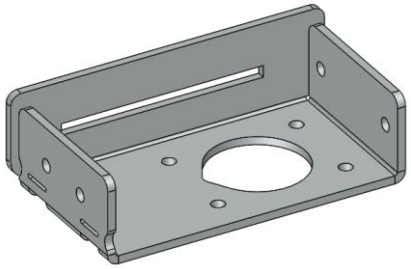


Use two M3x5 screws to secure the right side of Assembly Component No. 14 onto the robotic arm assembly.



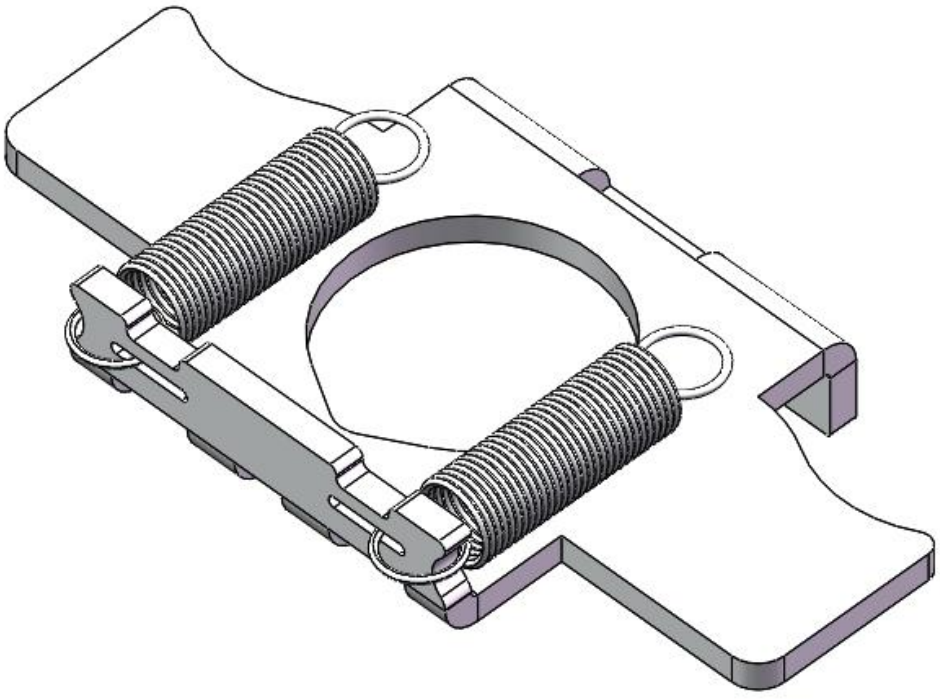
Step 16 Assembling the Pen Clip

Materials Needed

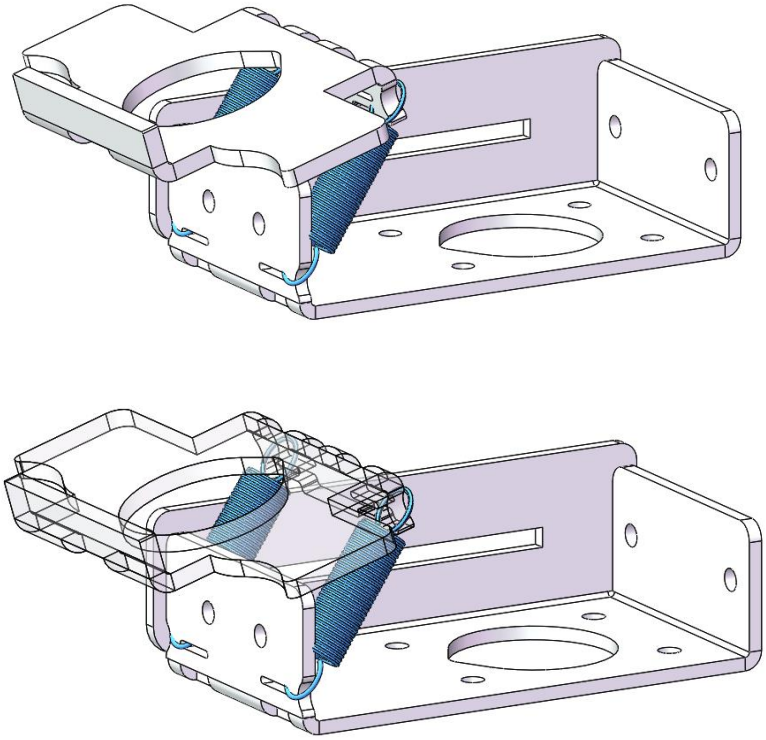
M3x3x5 Screw x4 	Assembly Component No.25 
0.5x5x25 Tension Spring x2 	Cross screwdriver (3mm) x1 
Assembly Component No.15 	Assembly Component No.16 

Installation Steps

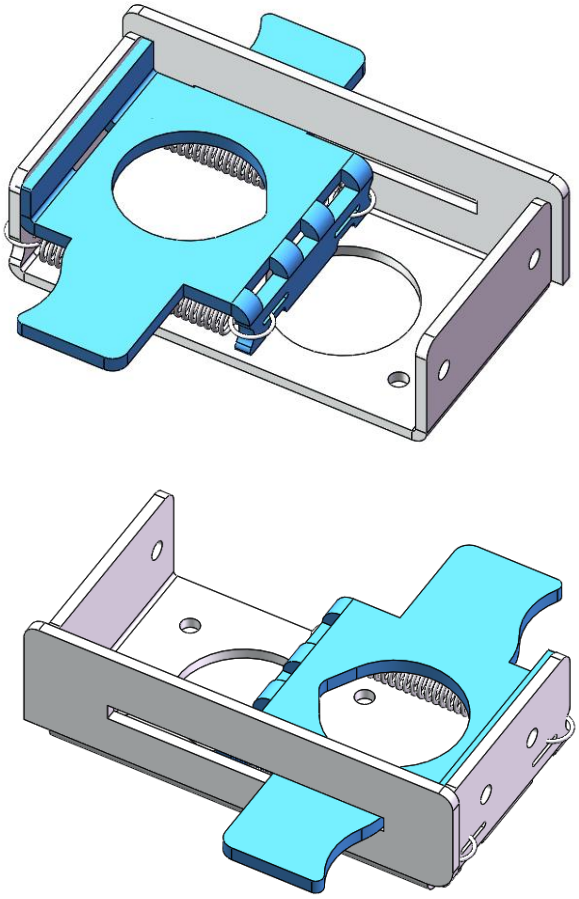
Secure one end of two tension springs to Component No. 25 as shown in the figure below.



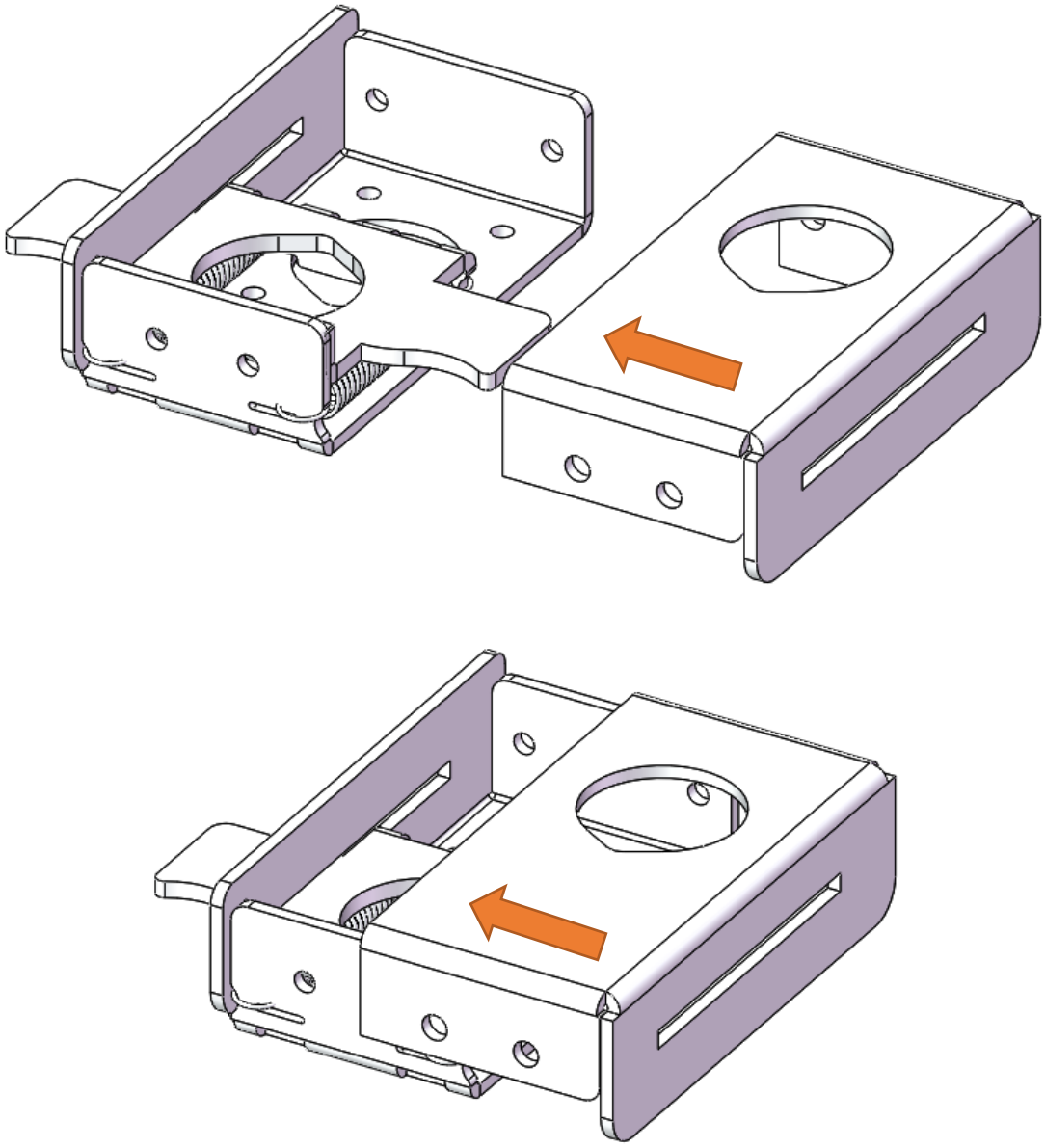
Secure one end of two tension springs to Component No. 16 as shown in the figure below.

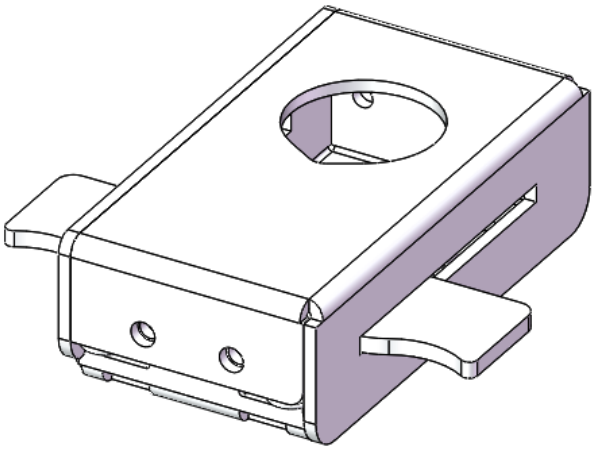


Mount Component No.25 to Component No.16, as shown below.

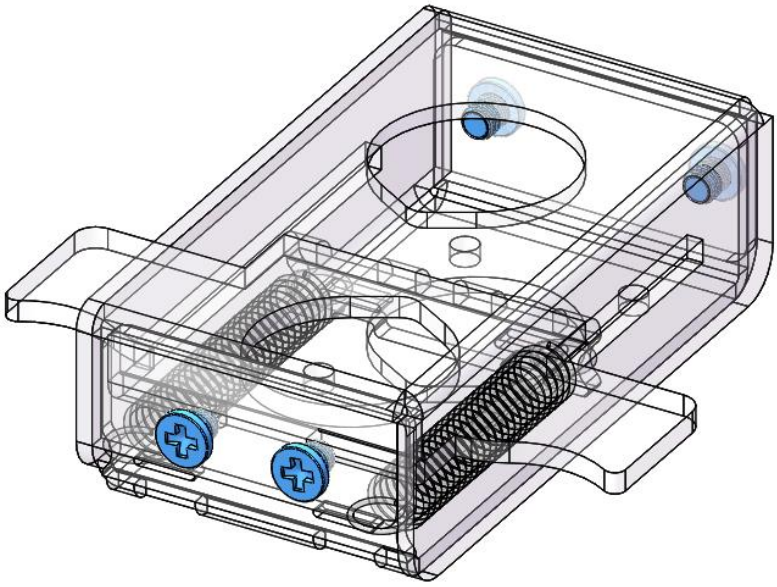


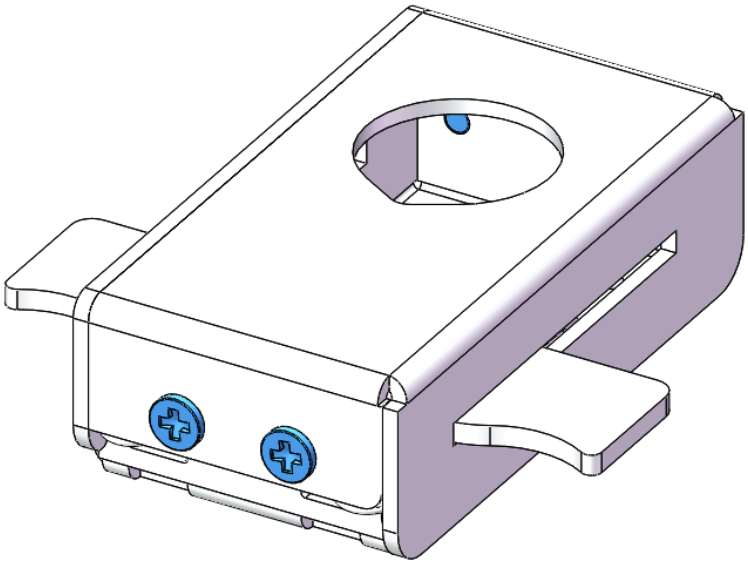
As shown in the figure below, slide Assembly Component No.15 from the right side of Component No. 16 by translating it horizontally.





Fix Components No.15 and No.16 with four M3x3x5 screws.

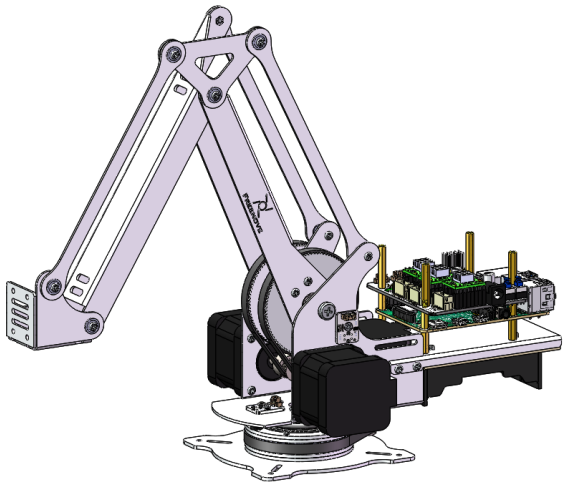
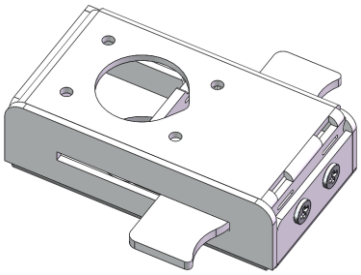




Step 17 Installing the Pen Clip

We have provided two types of clamping fixtures, please install the appropriate fixture according to your personal needs. Note that only one type of fixture is supported for installation at a time.

Materials Needed

Robot Arm Assembly 		Pen Clip 
5mm Open-end Wrench 	7mm Open-end Wrench 	Cross Screwdriver (3mm) x1 
M3x5x0.5 Gasket x4 	M6x9x0.5 Gasket x4 	Spring Plunger x2 

Installation Steps

As shown in the figure below, place four M3x5x0.5 gaskets between the spring plunger and the pen clip. Use a 5mm wrench to install the springplunfer onto the pen clip.

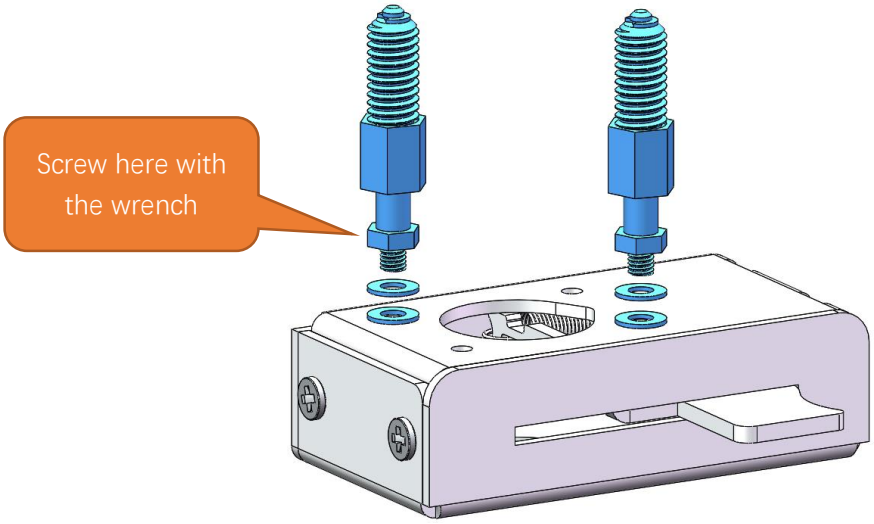
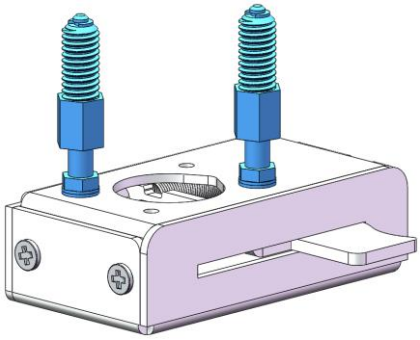
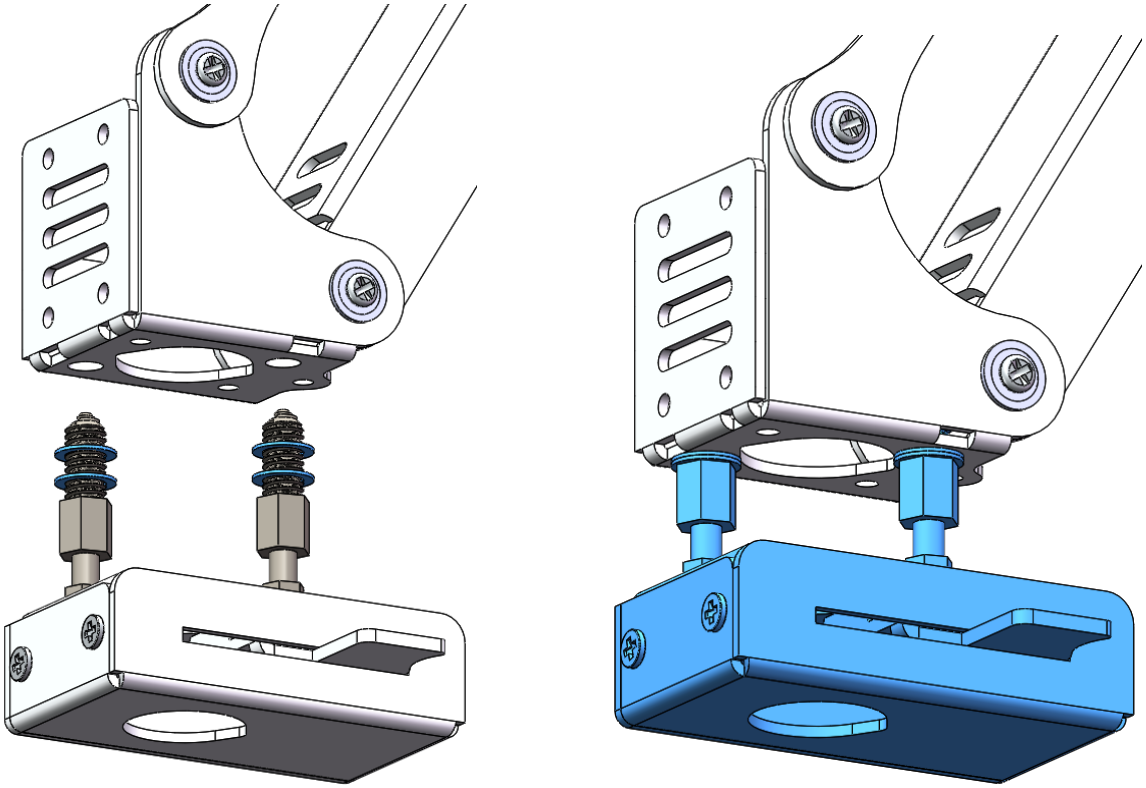


Diagram of the completed assembly



As illustrated in the figure below, space four M6x9x0.5 gasket between the spring plunger and the end of the robotic arm assembly, and use a 7mm wrench to fix them.



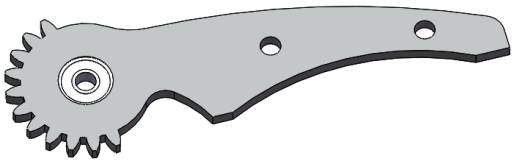
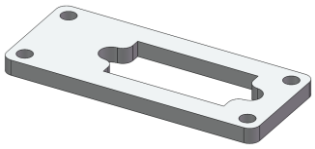
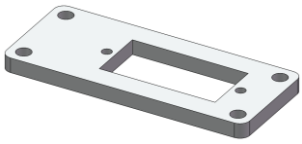
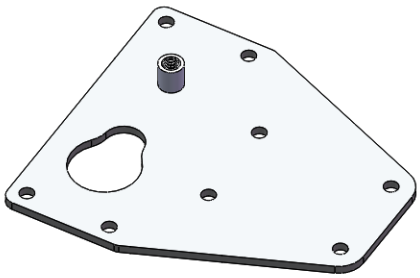
Step 18 Assembling the Servo Clamp

We have provided two types of clamping fixtures, please install the appropriate fixture according to your personal needs. Note that only one type of fixture is supported for installation at a time.

If you have assembled the pen clip, please skip this chapter and the following chapter.

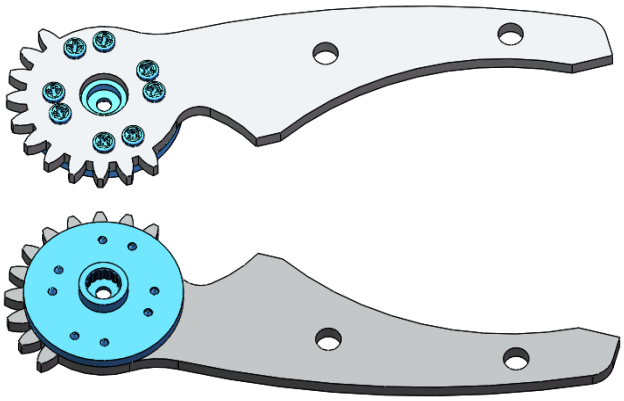
If you want to use the servo clamp, please disassemble the pen clip first.

Materials Needed

Assembly Conponent No.21 	Assembly Conponent No.22 
Assembly Conponent No.5 	Assembly Conponent No.6 
Assembly Conponent No.24 	Servo package x1 
M2x7 Screws x2、M2 Nuts x2 	M3x8 Screws x4 
M1.4x5 Screws x8 	Cross Screwdriver (3mm) x1 

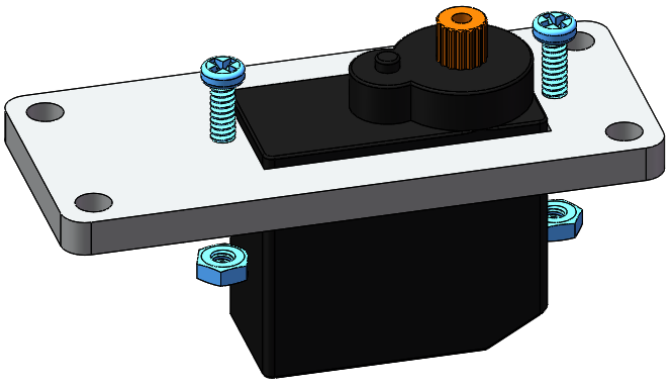
Installation Steps

Fix the disk servo horn to Assembly Component No.21 with four M1.4x5 screws. Pay attention to the orientation of the component. The figure below shows the front and back views of the installed component.

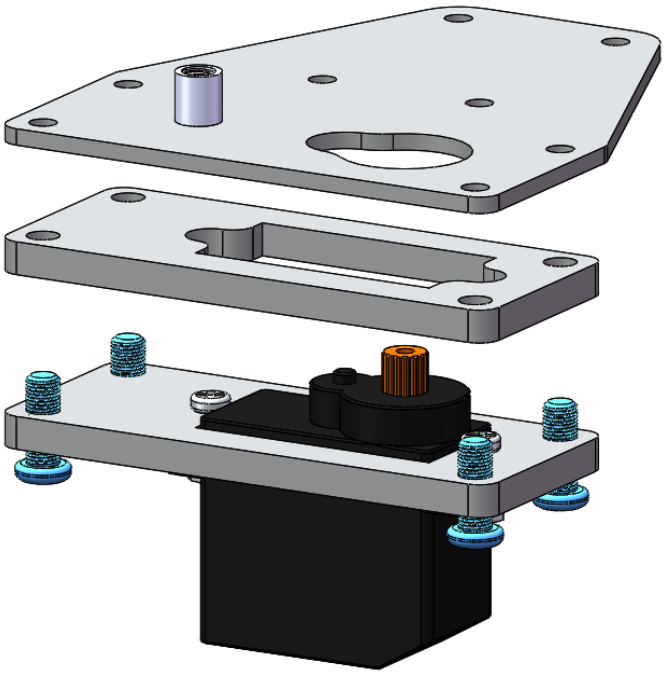


Note: Do not use the M1.4*7 screws in the servo bag even if there is any.

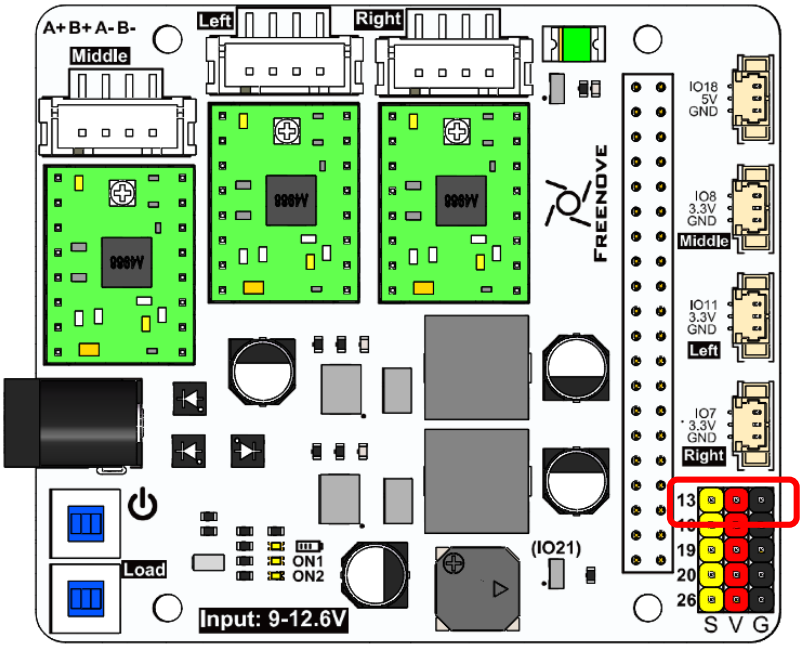
Attached the servo to Assembly Component No.6 with two M2x7 screws and M2 nuts.



Attached Components No.5, No.6 and No.24 together with four M3x8 screws.

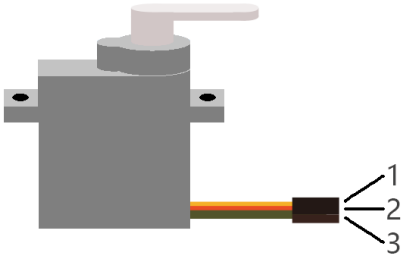


Connect the servo wire to the GPIO13 pin marked below. Connect power supply and turn ON both two switches.



Note: S connects to the signal wire of the servo (orange cable), V to positive (red), and G to negative (black).

Simultated Image of the Servo



Servo wiring

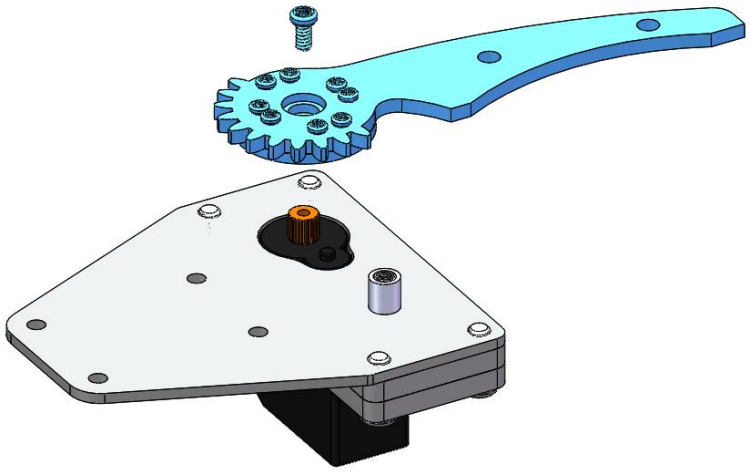
Board	Servo
13	1
V	2
G	3

Run the commands on Raspberry Pi Terminal

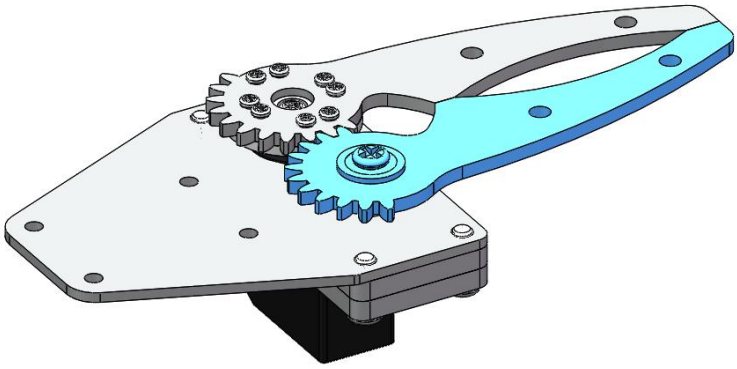
```
cd ~
cd Freenove_Robot_Arm_Kit_for_Raspberry_Pi/Server/Code/
sudo python servo.py 0 0
```

You can see the servo rotates to the position of 0 degree.

Keep the servo powered. Attached the Component No.21 to the servo with a M2x4 screw.



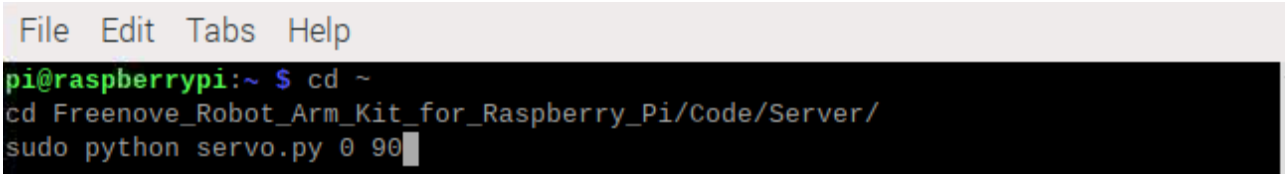
Attach Component No.22 to Component No.22 with an M3x5 screw, as shown below.



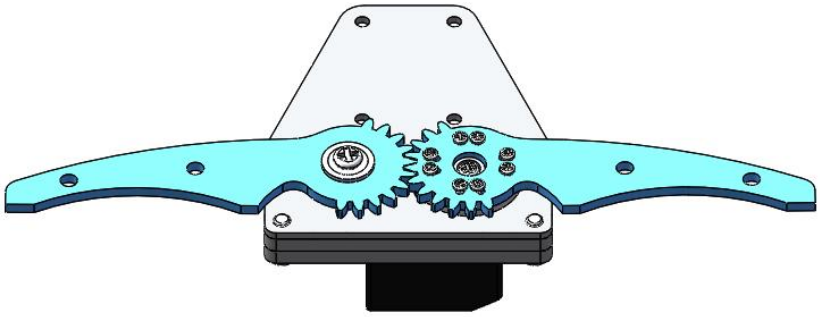
Verify assembly.

Run the commands on Raspberry Pi terminal.

```
cd ~
cd Freenove_Robot_Arm_Kit_for_Raspberry_Pi/Server/Code/
sudo python servo.py 0 90
```



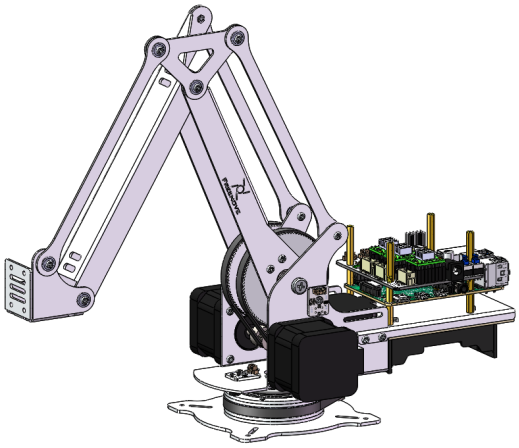
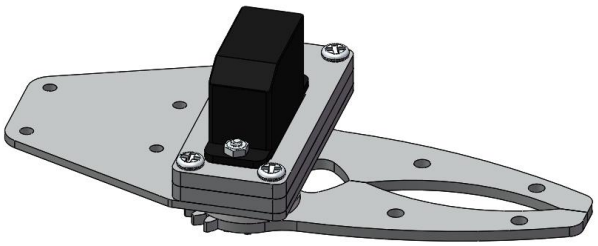
This results in the servo rotating to 90 degrees.



Step 19 Installing the Servo Clamp

Here are two mounting options for the servo clamp. You can choose the most suitable method based on your specific application scenario.

Materials Needed

Robot Arm Assembly 	Servo Clamp 
M3x5 Screws x4 	Cross Screwdriver (3mm) x1 

Mounting Options 1

As shown in the figure below, use two M3x5 screws to secure the servo clamp onto the robotic arm assembly.

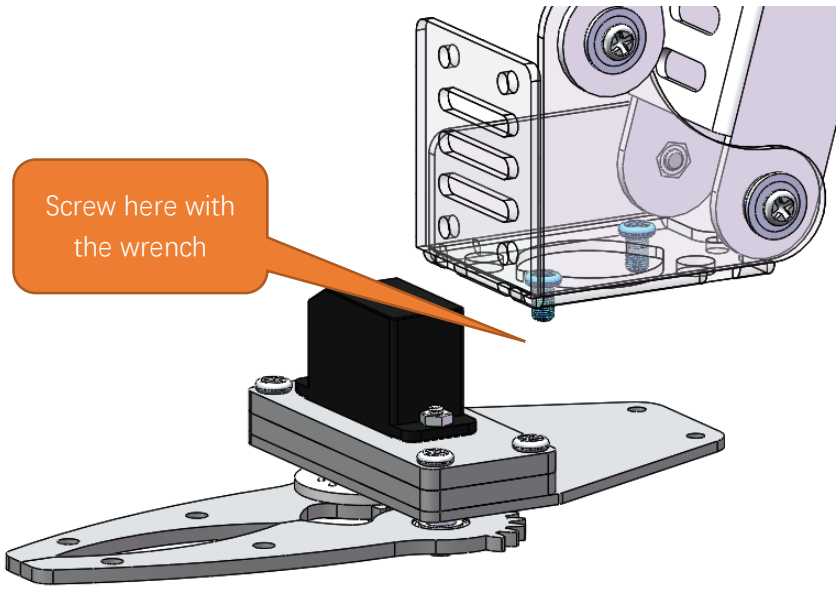
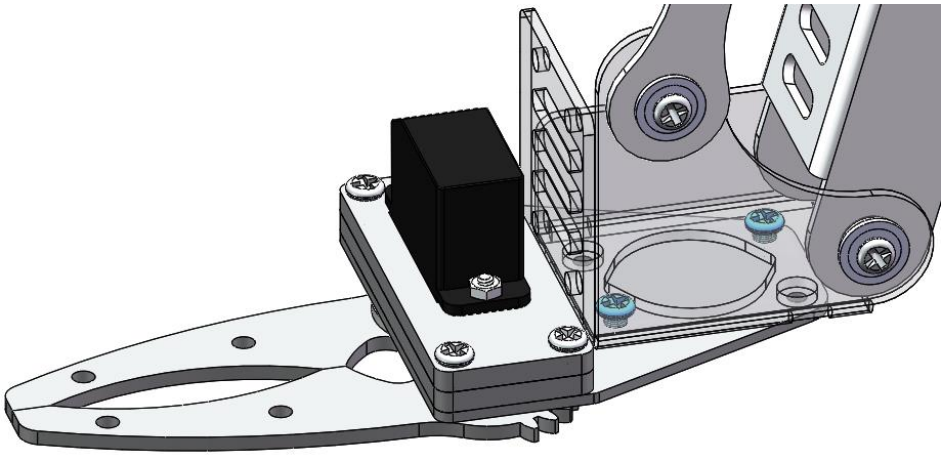
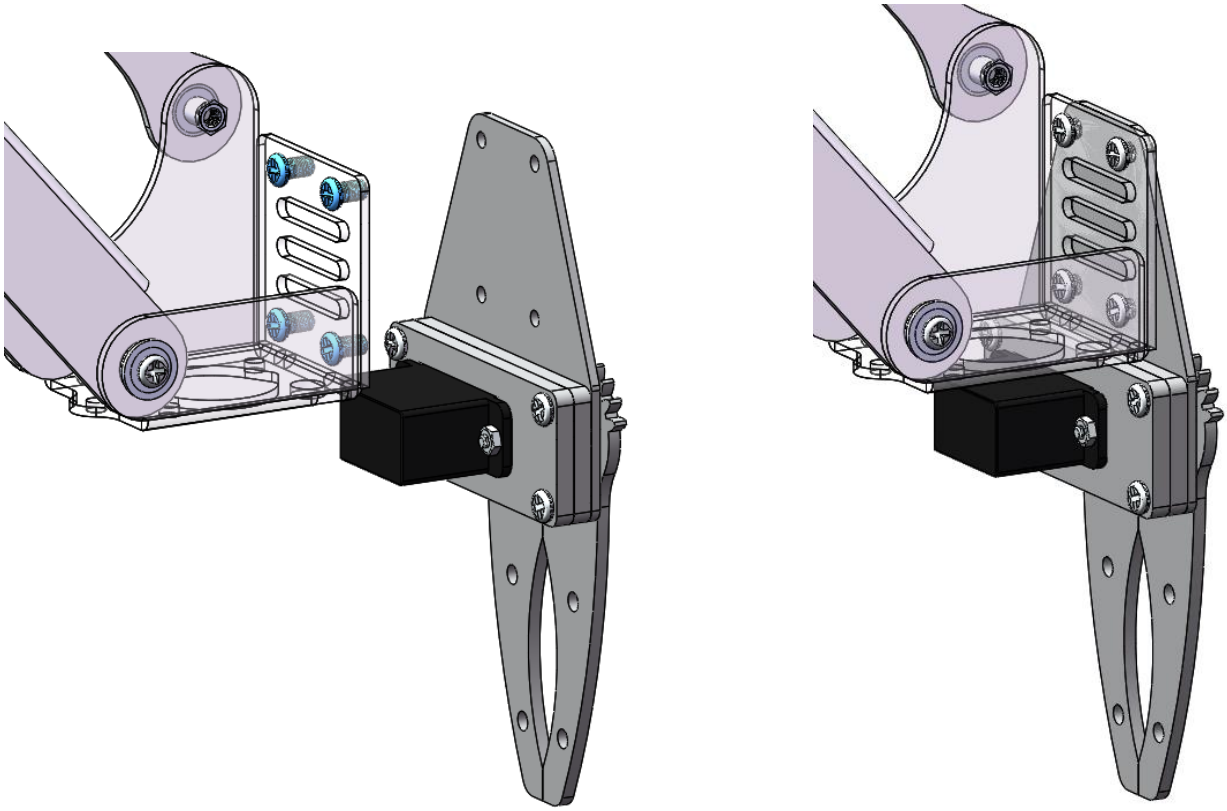


Diagram of completed assembly



Mounting Options 2

As depicted in the figure below, install the servo clamp onto the end of the robotic arm assembly with four M3x5 screws.



Step 20 Wiring of Stepper Motor and Sensors

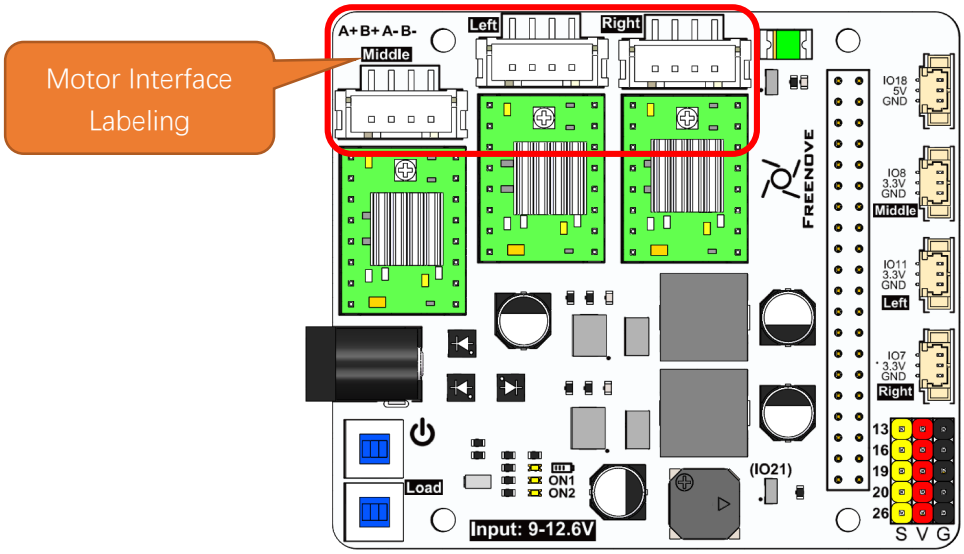
Materials Needed

20cm 3Pin cable (same direction)

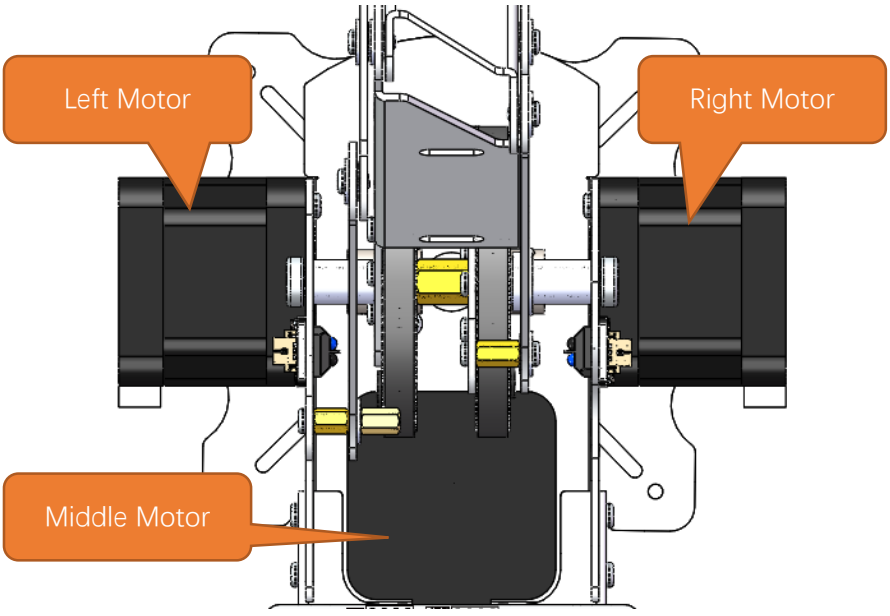
Cable for Stepper Motor x2

Installation Steps

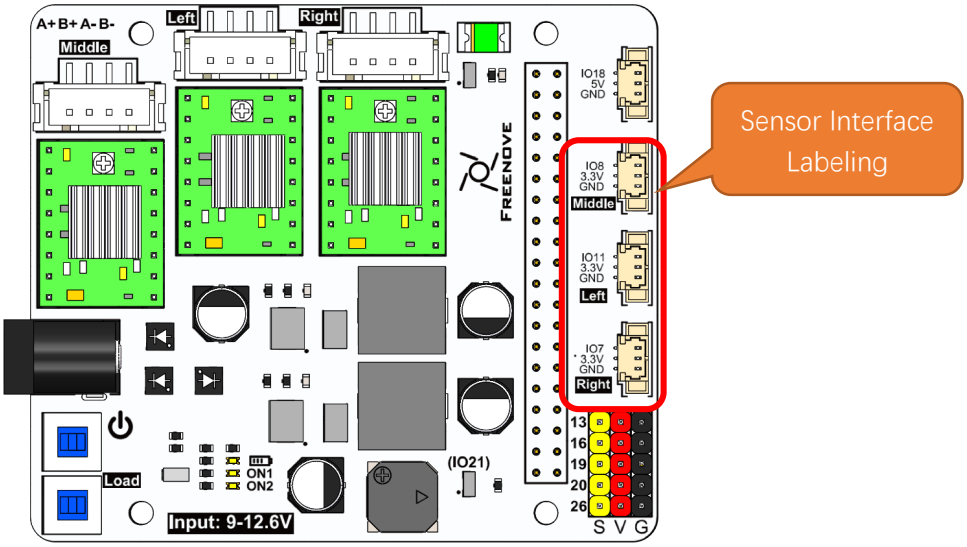
the Stepper Motors to Robot Arm Board with the cables.



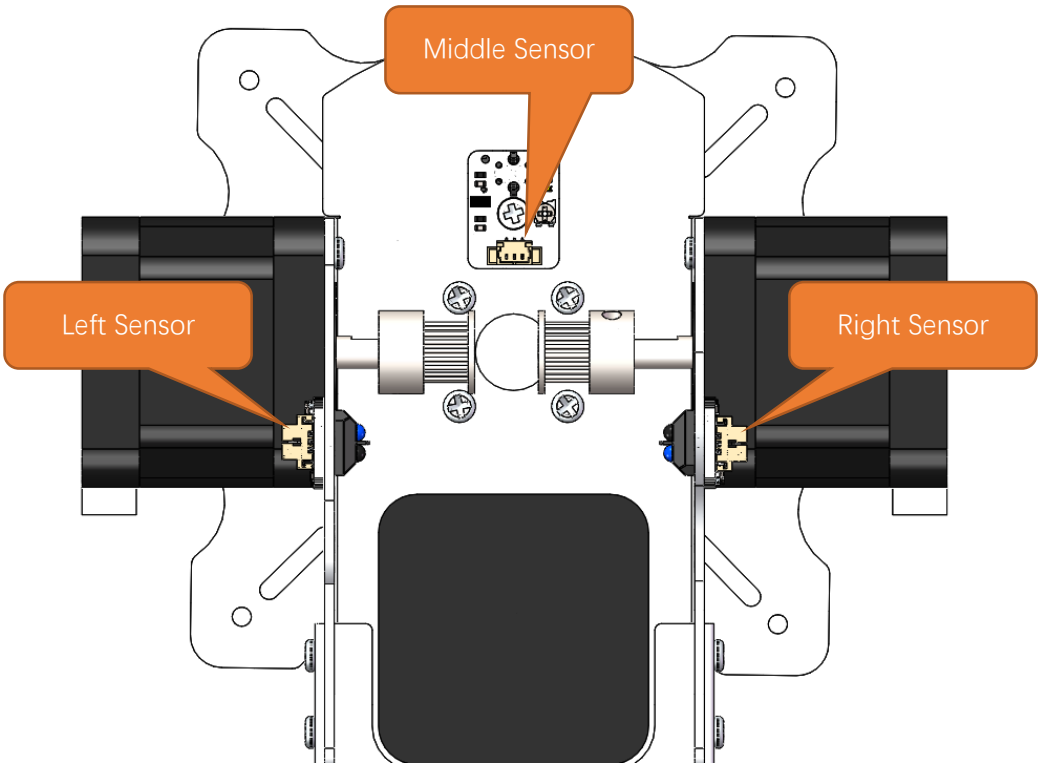
Please be aware that the motor interfaces on the board are clearly labeled; ensure that you do not mistakenly connect the wires to the wrong terminals.



Conenect the sensors to Robot Arm Board with three 20cm 3Pin Cable.

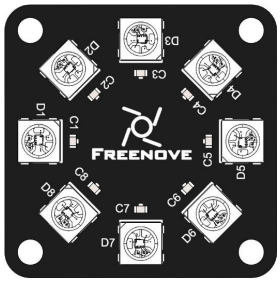
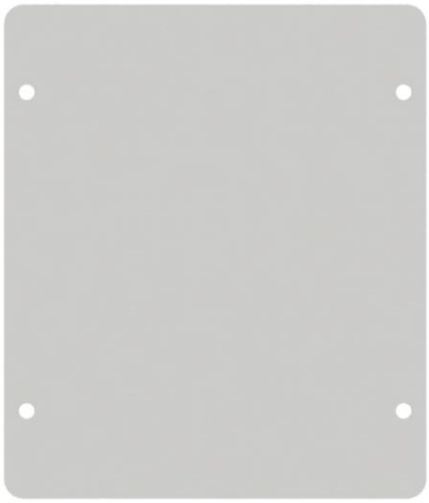
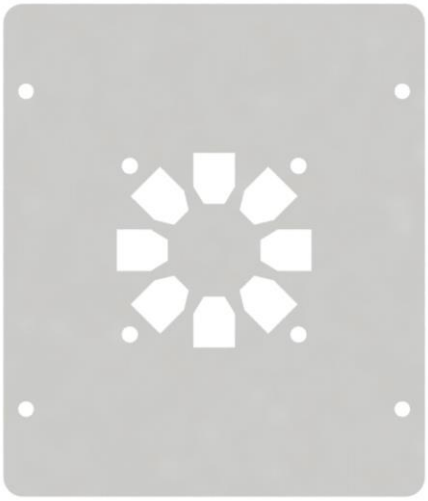


Please note that the corresponding sensor interfaces are already labeled on the board—ensure that you do not mistakenly connect the wires to the incorrect terminals.



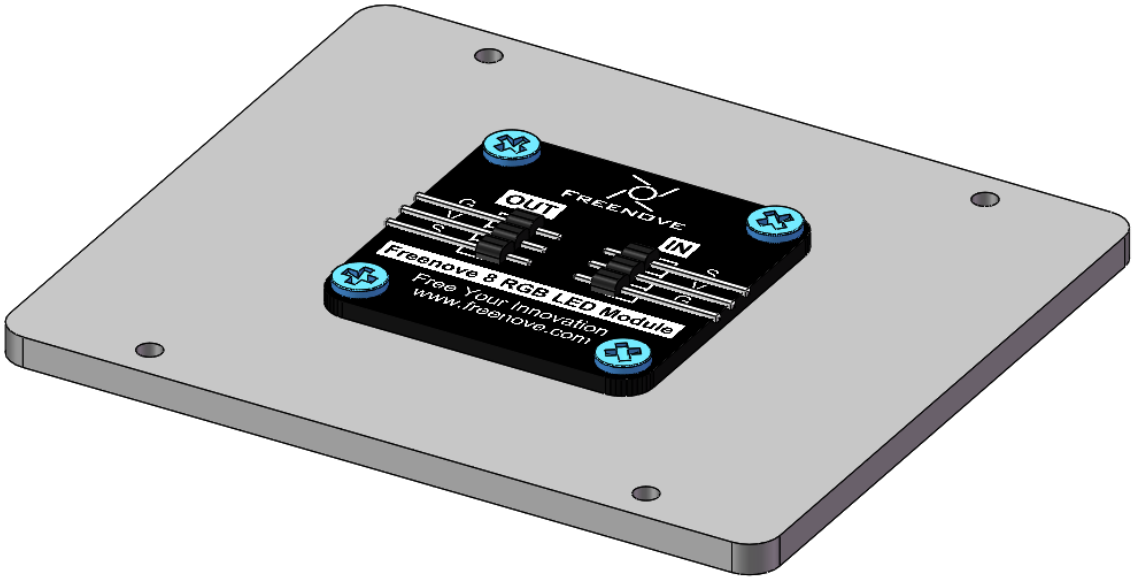
Step 21 Installing the LED Module

Material Needed

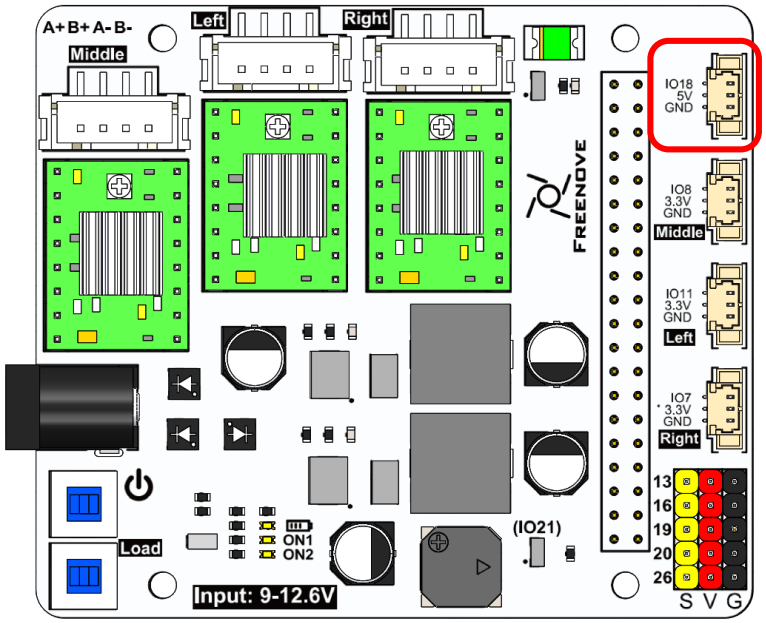
LEDPixel x1 	Top Acrylic Plate 	LED Module Mounting Plate 
M3x3x5 Screws x4 	M2.5x8 Screws x4 	Cross Screwdriver (3mm) x1 
		10cm 3Pin Cable to Jumper Wire. 

Installation Steps

Attach the LEDpixel module to the fixing board with four M3x3x5 screws, as shown below.



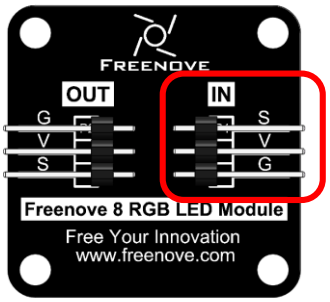
Connect the LEDPixel and Robot Arm Board with the 10cm 3Pin cable to Jumper wire



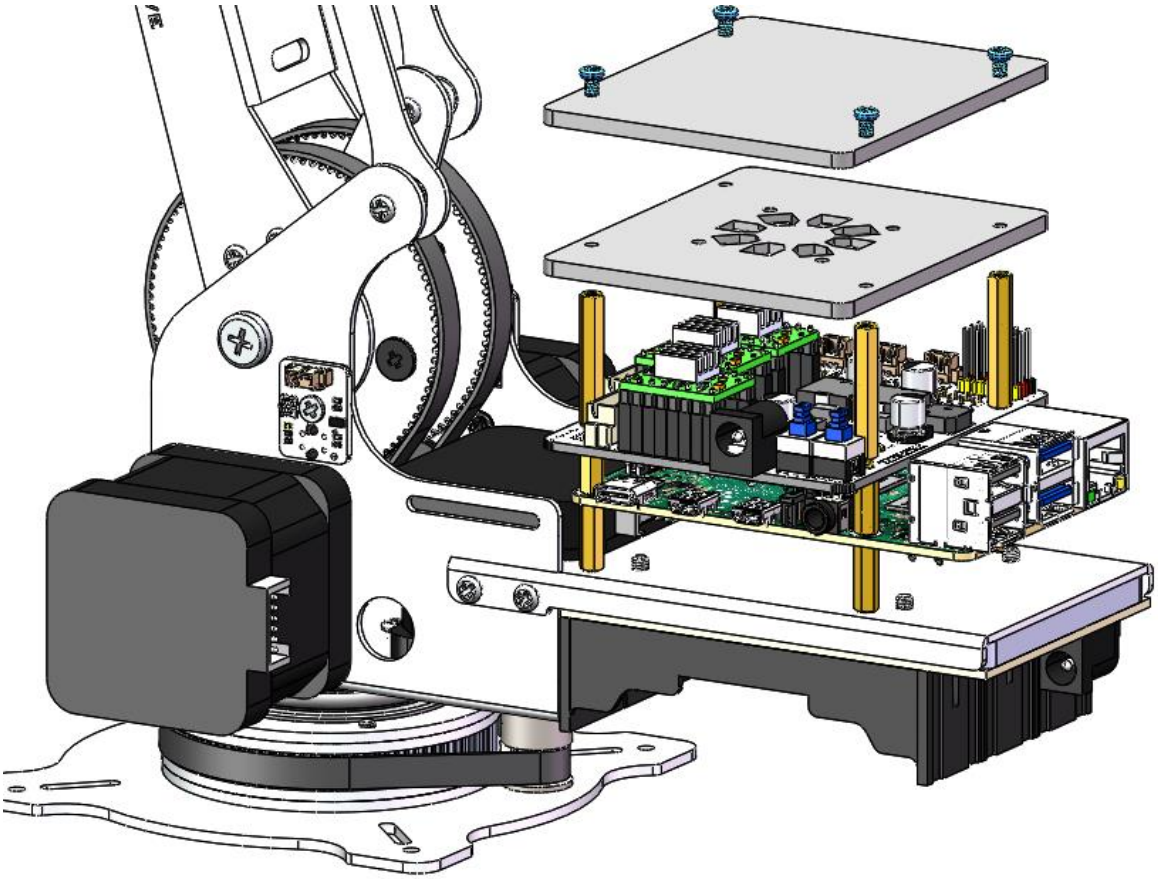
Cable sequence

Board	LedPixel
IO18	S
5V	V
GND	G

LEDPixel



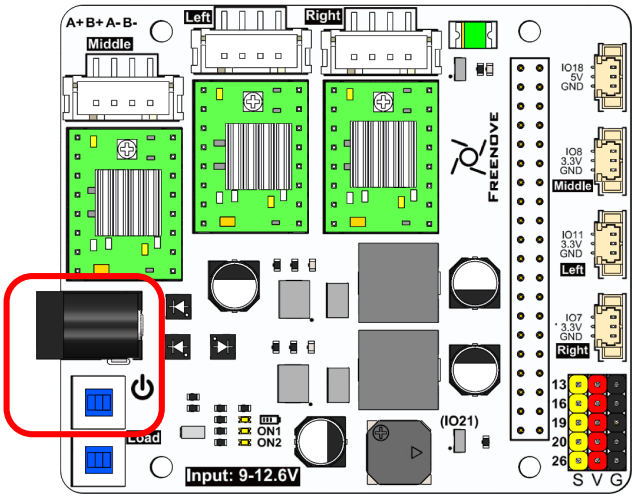
Use four M2.5x8 screws to mount the acrylic top plate and the LED mounting plate onto four M2.5x25 standoffs.



Step 22 Adjustment of the Sensor Sensibility

Before using the robotic arm, it is necessary to calibrate the sensors to ensure the smooth and proper functioning of the mechanical system.

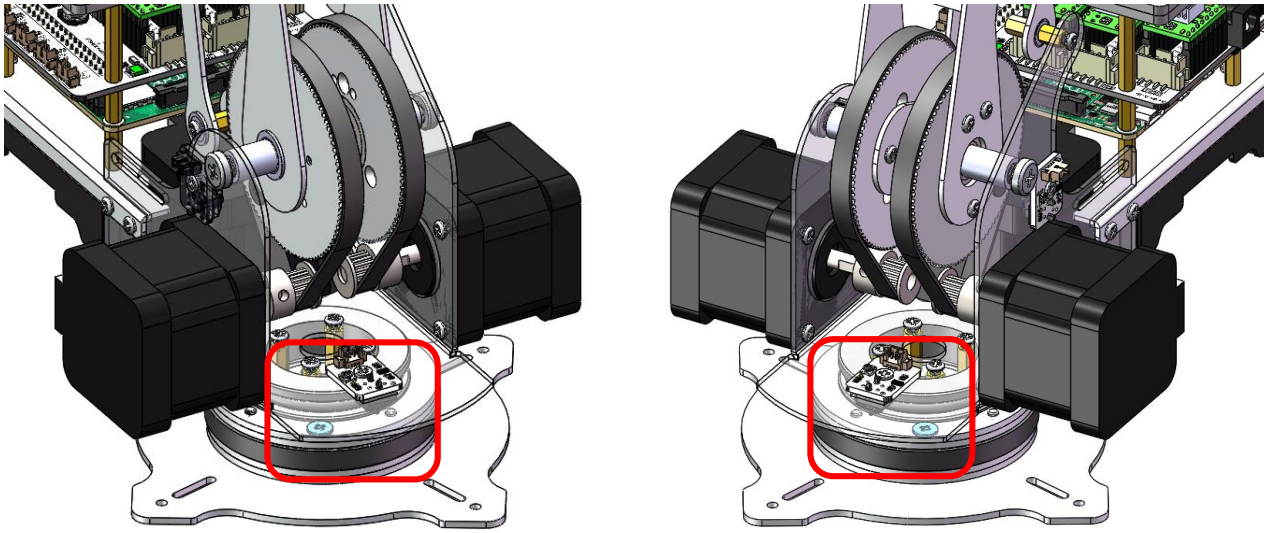
Power the robot board. Press the switch circled below, and you'll see the indicator ON1 light up.



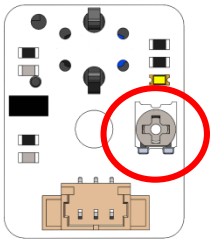
The indicators of the three sensors on the robot arm will also light up.

Adjust the middle sensor.

Rotate the robotic arm's main body so that the sensor passes back and forth over the M3x3x7 black screws. Observe the indicator light on the sensor during this process.

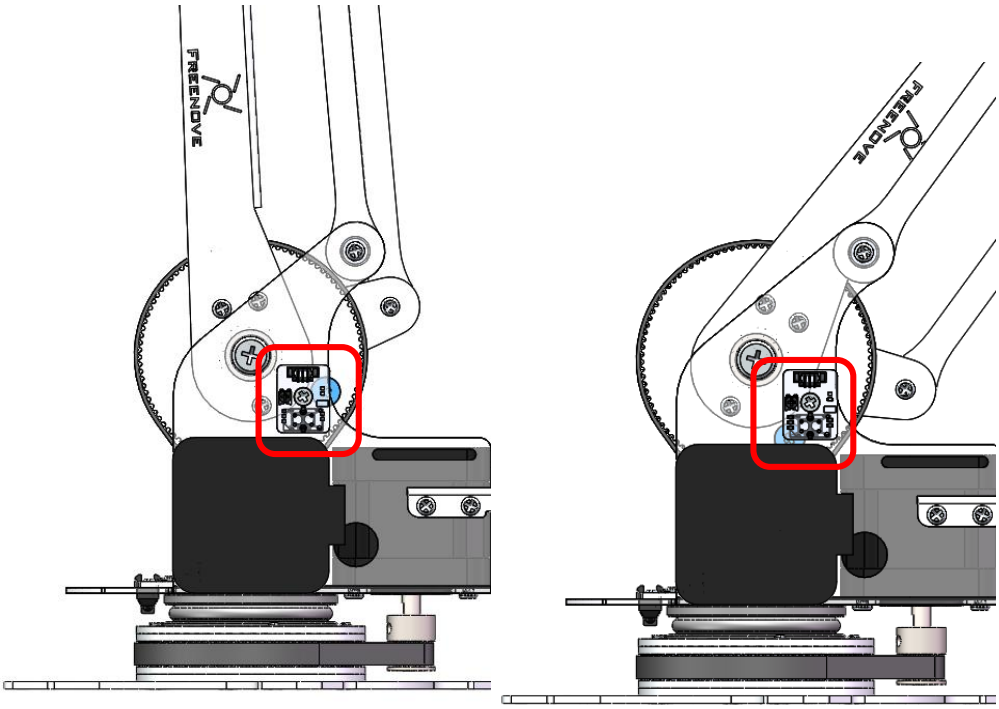


When the sensor is positioned above the black screws, the indicator light on the sensor should go out. If the light stays ON, adjust the dial on the sensor using a screwdriver. **Do not rotate the dial by more than half a turn.**

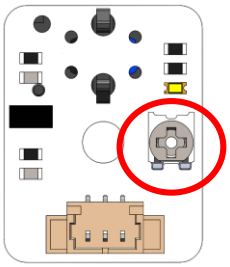


Adjust the sensor on the left side.

Rotate Assemblyl Component No.17 so that the sensor moves back and forth over the M3x3x7 black screws. Observe the indicator light on the sensor as you do so.

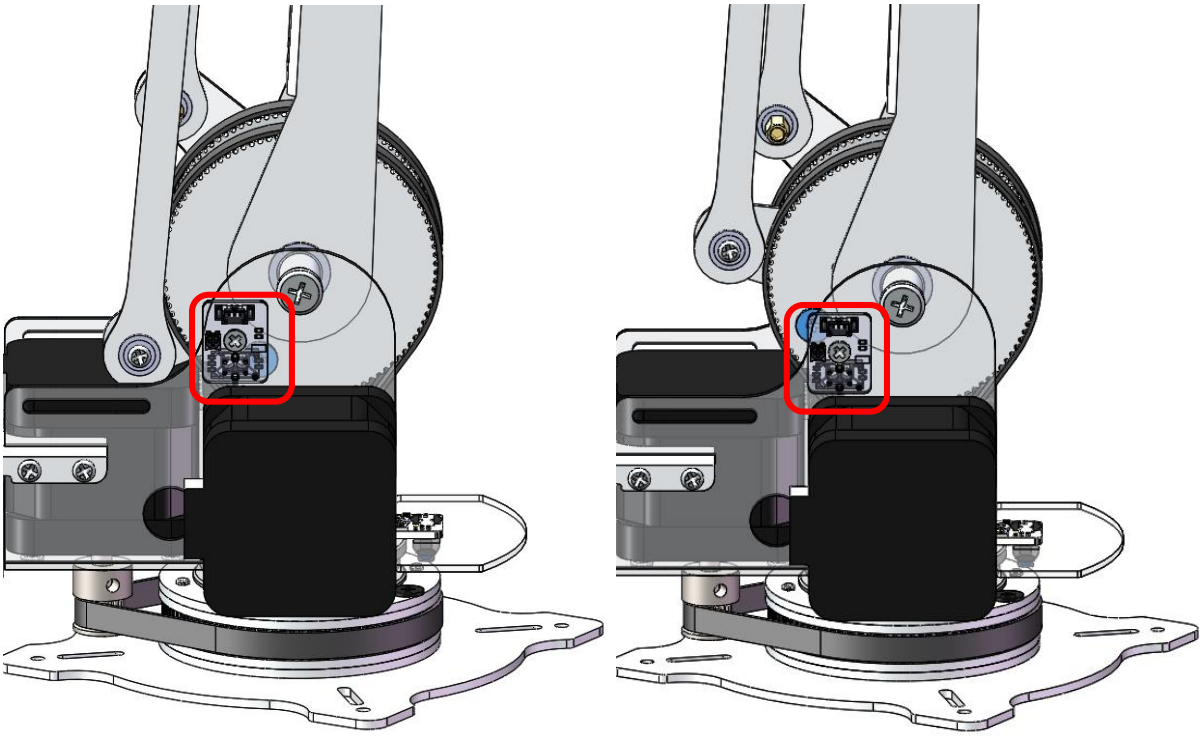


When the sensor is positioned above the black screws, the indicator light on the sensor should go out. If the light stays ON, adjust the dial on the sensor using a screwdriver. **Do not rotate the dial by more than half a turn.**

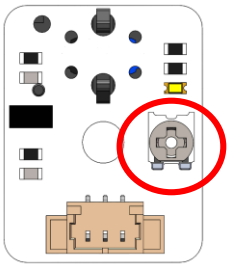


Adjust the sensor on the left side.

Rotate Assemblyl Component No.18 so that the sensor moves back and forth over the M3x3x7 black screws. Observe the indicator light on the sensor as you do so.



When the sensor is positioned above the black screws, the indicator light on the sensor should go out. If the light stays ON, adjust the dial on the sensor using a screwdriver. **Do not rotate the dial by more than half a turn.**



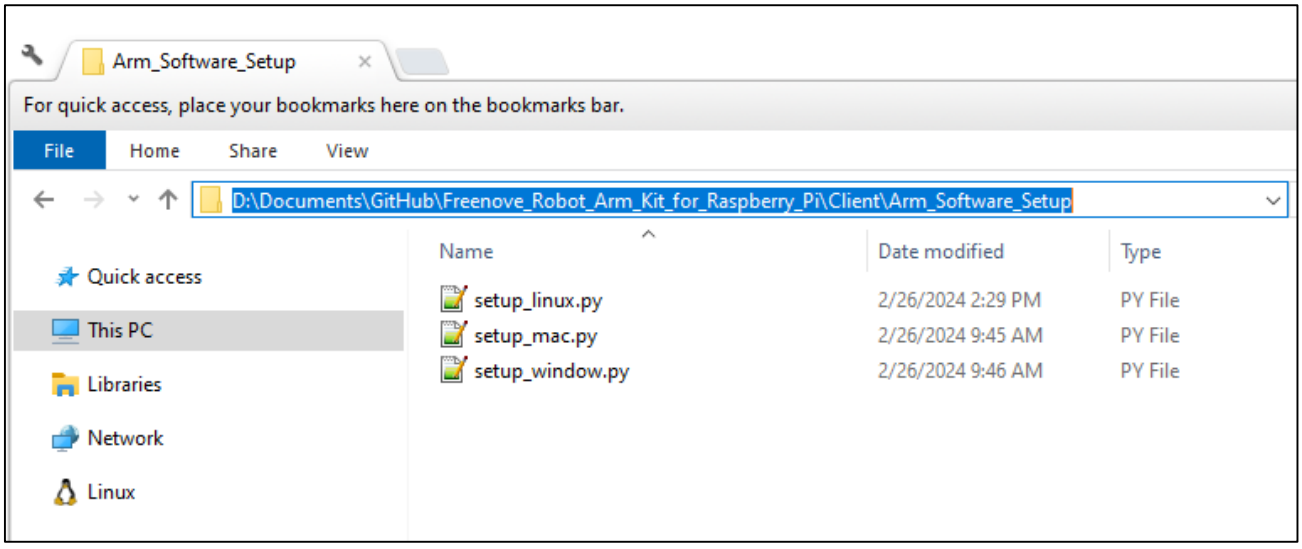
Chapter 3 Installation, Launch, and Packaging of Robotic Arm Control Software

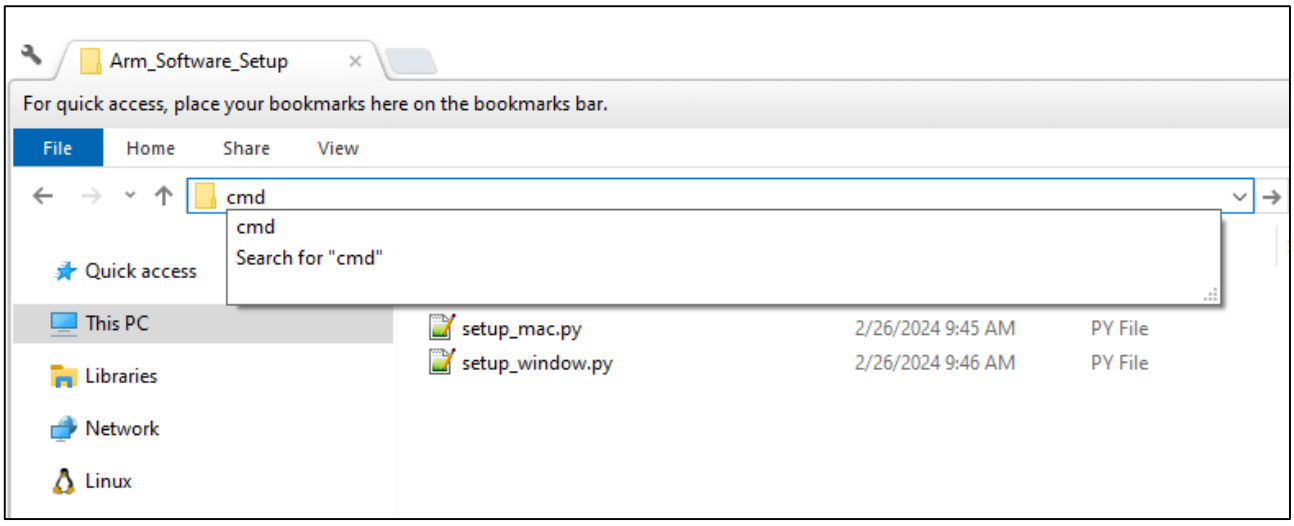
Libraries Installation

We offer quick installation methods for libraries across three major platforms. Please select and install the appropriate libraries based on your computer's operating system platform.

Windows

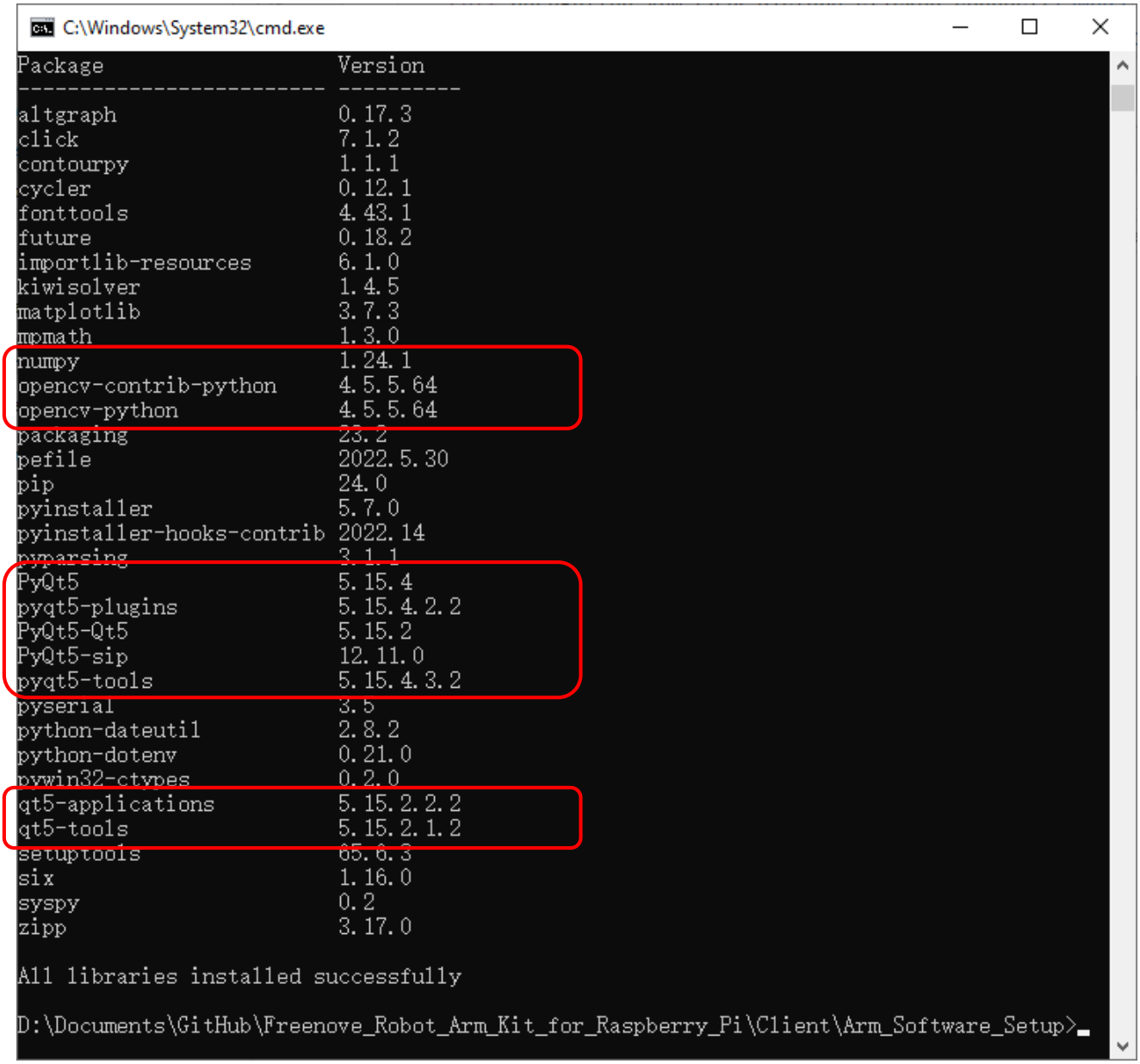
Open the folder “Freenove_Robot_Arm_Kit_for_Raspberry_Pi/Client/Arm_Software_Setup” and type in “cmd” on the search bar.





Run the following command and wait for the libraries to install automatically. Once the installation is complete, you should see a confirmation similar to the illustration shown below.

```
python setup_window.py
```

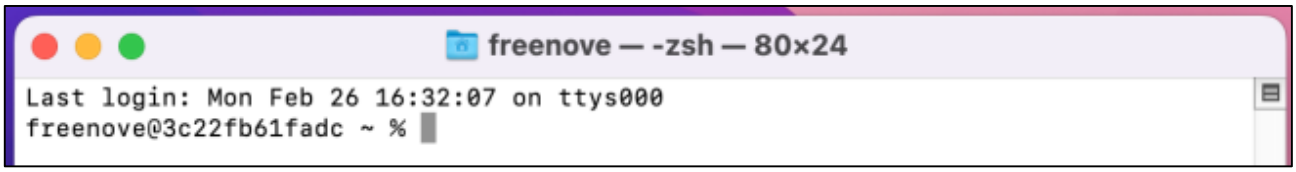


Please note:

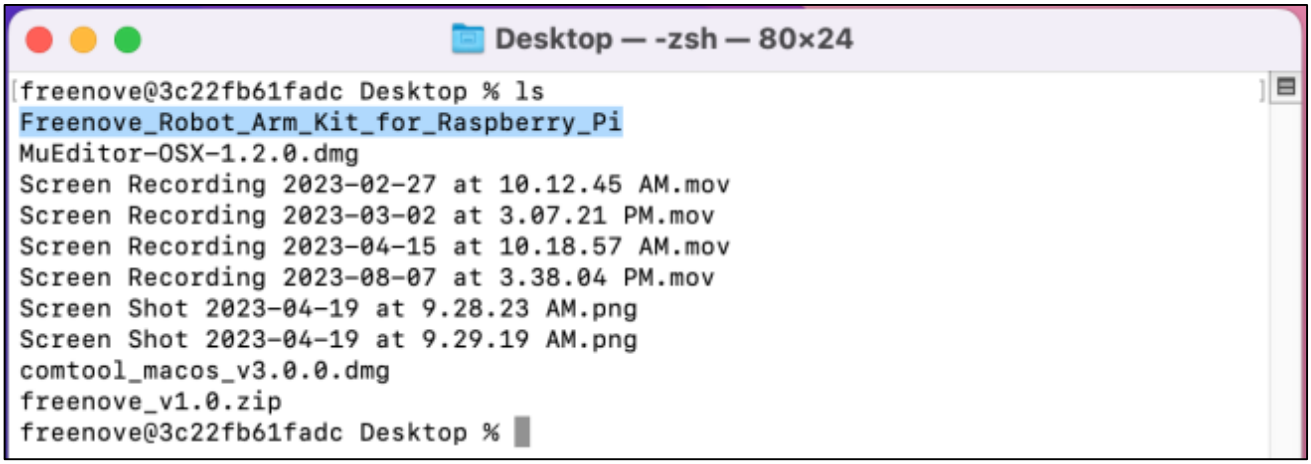
1. **Proxy Settings**: Proxy configurations can cause the installation of libraries to fail. If you have a proxy enabled on your computer, please disable it and then attempt the installation again.
2. **Network Connectivity**: Poor network conditions may also lead to installation failures. Ensure that you have a stable internet connection, and if necessary, retry the installation several times until it is successful.

Mac

Open the Terminal.

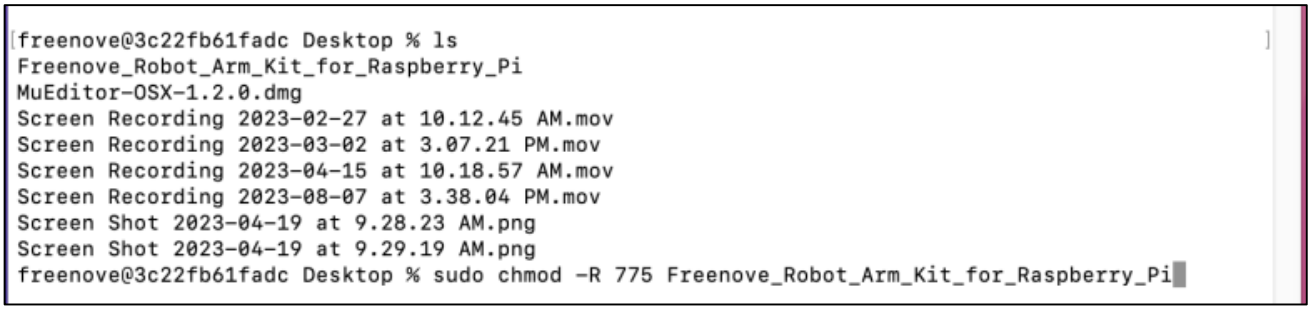


Find the "Freenove_Robot_Arm_Kit_for_Raspberry_Pi" on your Mac. We place it on the desktop, as shown below:



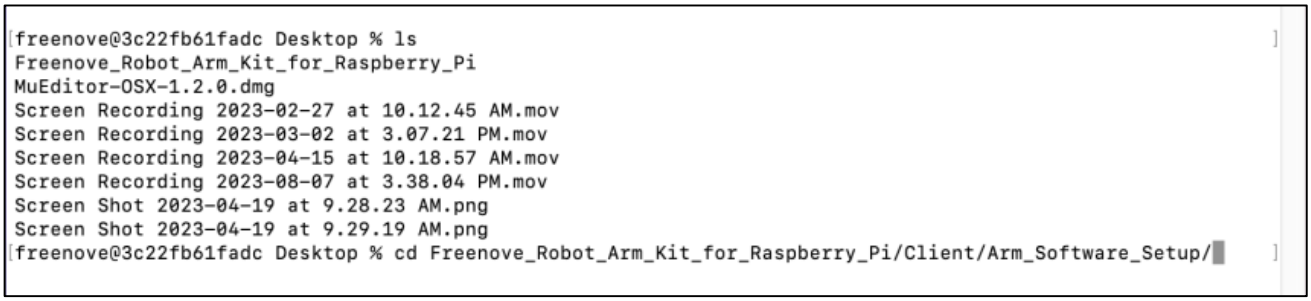
Run the following command to modify the permission, allowing the code to run normally.

```
sudo chmod –R 775 Freenove_Robot_Arm_Kit_for_Raspberry_Pi
```



Run the following command to enter the directory.

```
cd Freenove_Robot_Arm_Kit_for_Raspberry_Pi/Client/Arm_Software_Setup
```

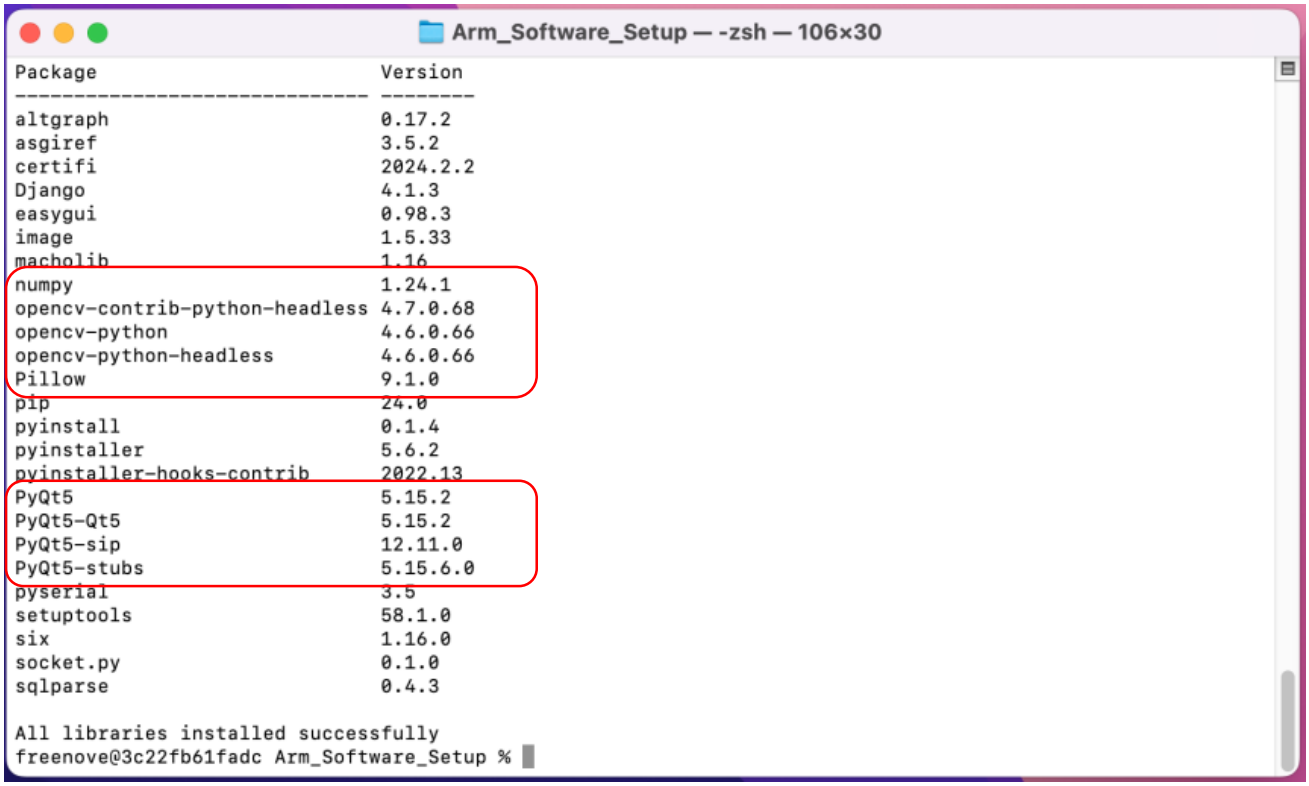


Run the command to install the libraries.

```
sudo python3 setup_mac.py
```

```
freenove@3c22fb61fadc Arm_Software_Setup %
freenove@3c22fb61fadc Arm_Software_Setup % ls
setup_linux.py  setup_mac.py    setup_window.py
freenove@3c22fb61fadc Arm_Software_Setup % sudo python setup_mac.py
```

Once the installation is complete, you should see a confirmation similar to the illustration shown below.

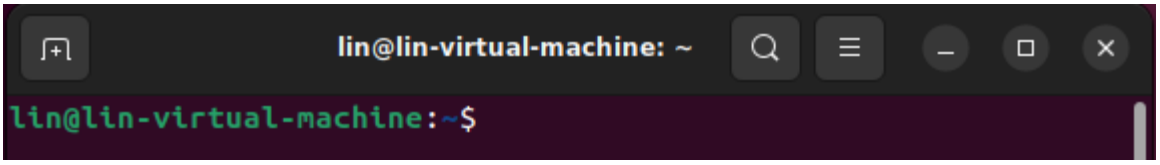


Please note:

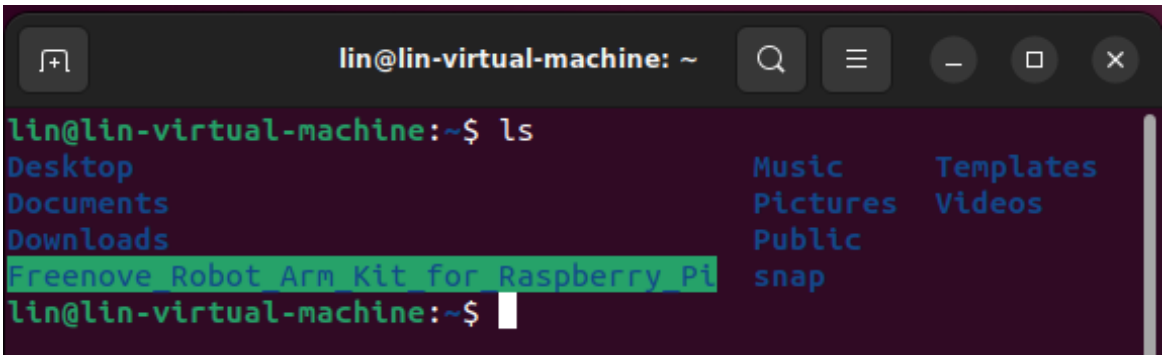
1. **Proxy Settings**: Proxy configurations can cause the installation of libraries to fail. If you have a proxy enabled on your computer, please disable it and then attempt the installation again.
2. **Network Connectivity**: Poor network conditions may also lead to installation failures. Ensure that you have a stable internet connection, and if necessary, retry the installation several times until it is successful.

Linux

Open the Terminal.



Find the "Freenove_Robot_Arm_Kit_for_Raspberry_Pi" folder on your computer. We place it under the home directory, as shown below:

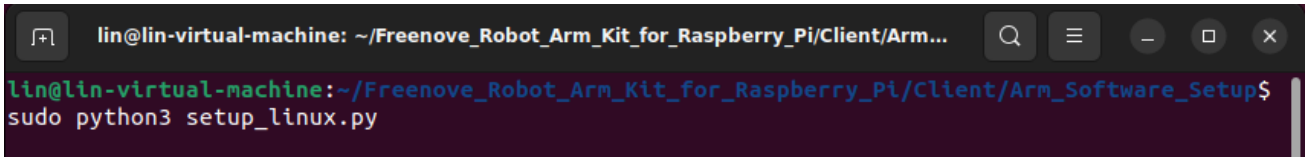


Run the command to enter the folder.

```
cd Freenove_Robot_Arm_Kit_for_Raspberry_Pi/Client/Arm_Software_Setup
```

Run the command to install the libraries.

```
sudo python3 setup_linux.py
```



Once the installation is complete, you should see a confirmation similar to the illustration shown below.

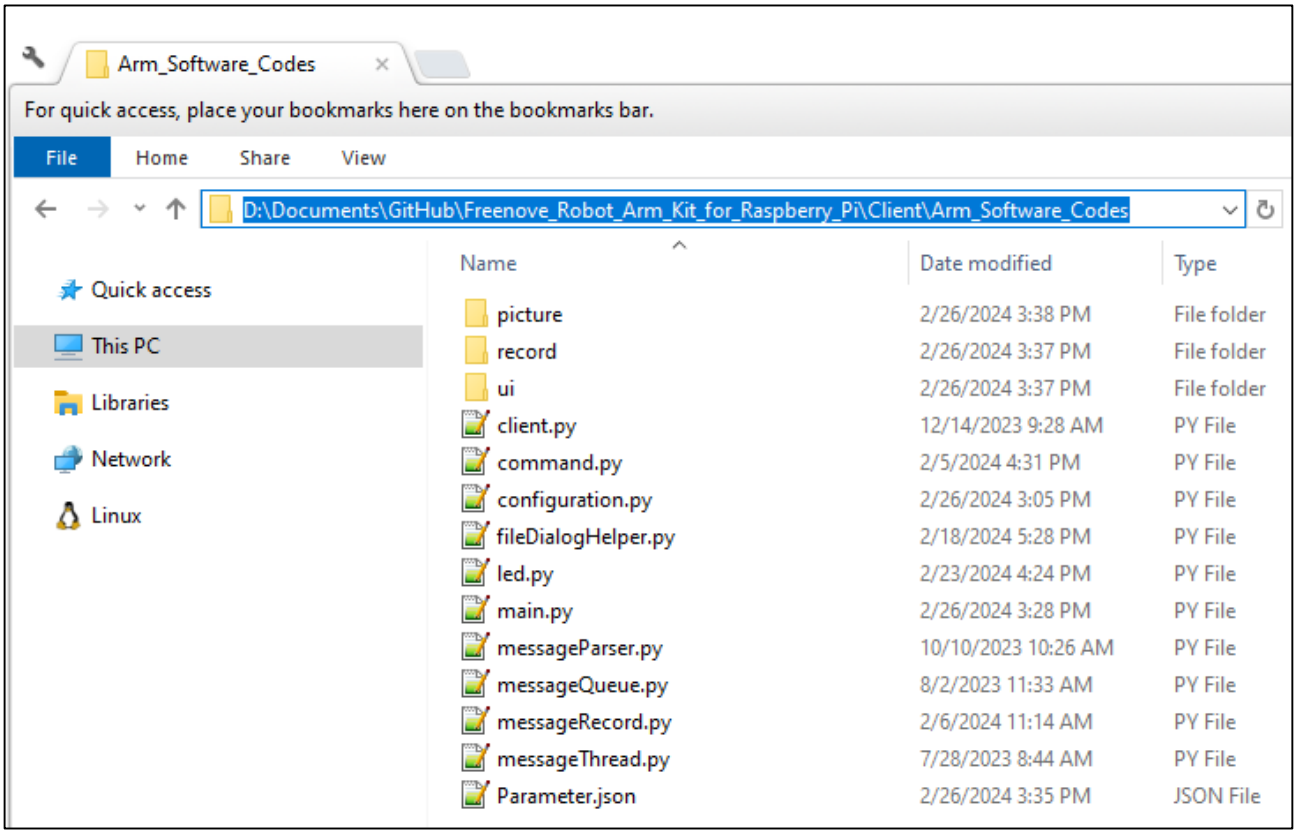
Please note:

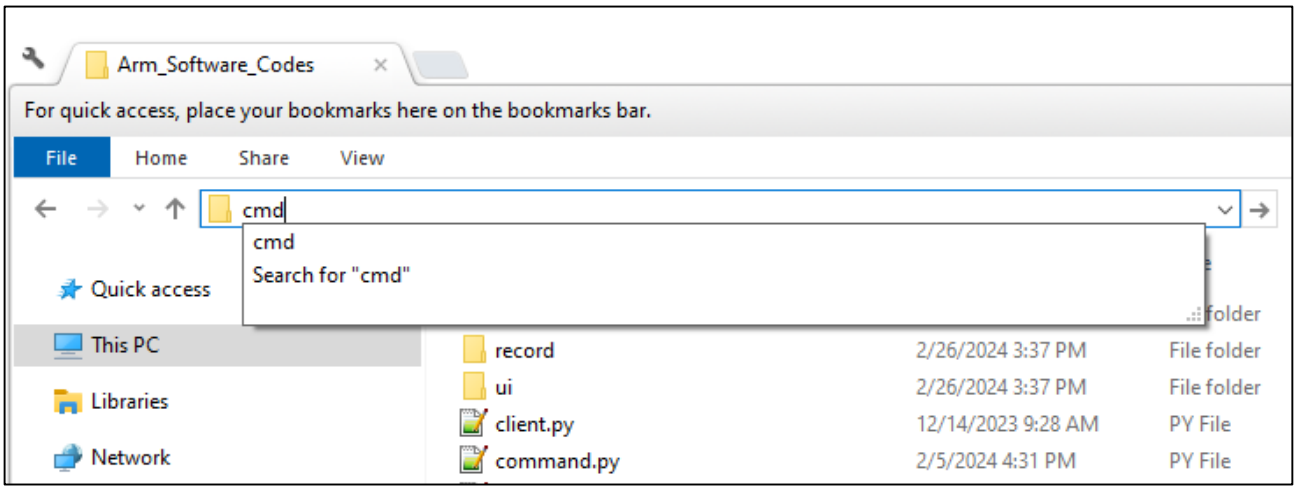
1. **Proxy Settings**: Proxy configurations can cause the installation of libraries to fail. If you have a proxy enabled on your computer, please disable it and then attempt the installation again.
2. **Network Connectivity**: Poor network conditions may also lead to installation failures. Ensure that you have a stable internet connection, and if necessary, retry the installation several times until it is successful.

Robot Arm Controlling Software

Windows

Open the folder “Freenove_Robot_Arm_Kit_for_Raspberry_Pi/Client/Arm_Software_Codes”, type in “cmd” on the search bar.

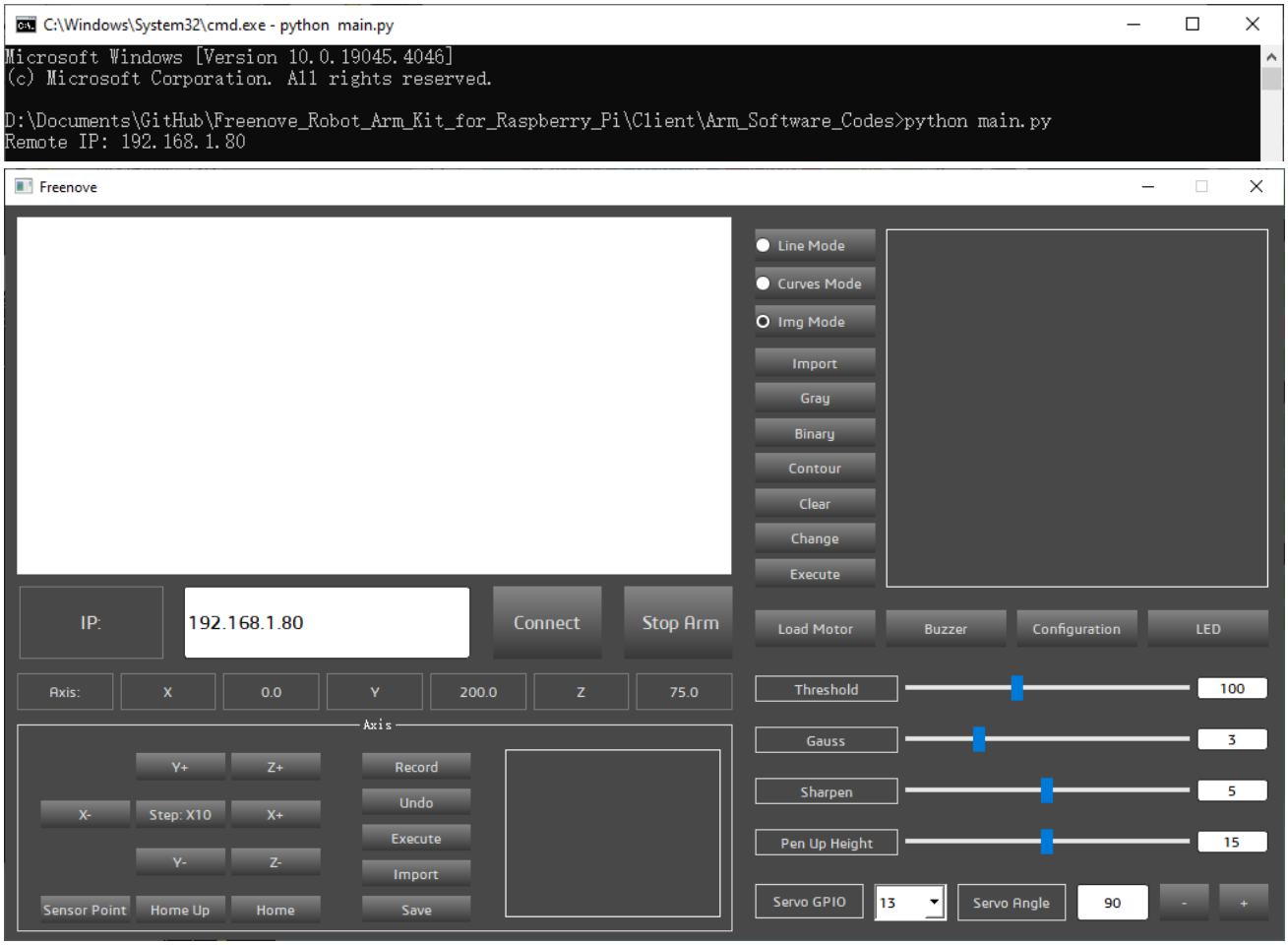




Run the command to start the software.

```
python main.py
```

The interfaces as shown below:



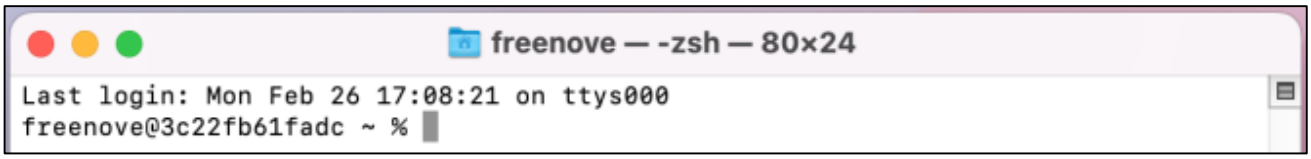
Please note the following:

1. The use of a proxy or other network tools may result in the robotic arm control software obtaining an incorrect network IP address.
2. Make sure your computer and Raspberry Pi are connected to the same local network to avoid communication failures.

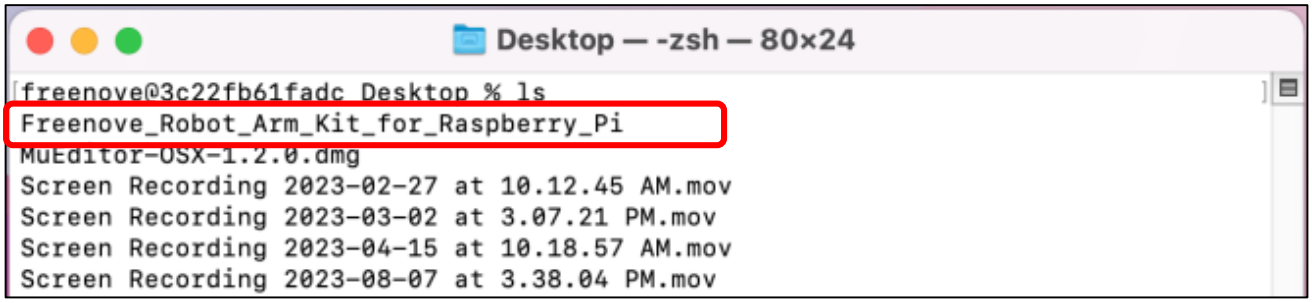
Once you have learned how to open the software for controlling the robotic arm, please proceed to the next step by clicking here.

Mac

Open the Terminal.

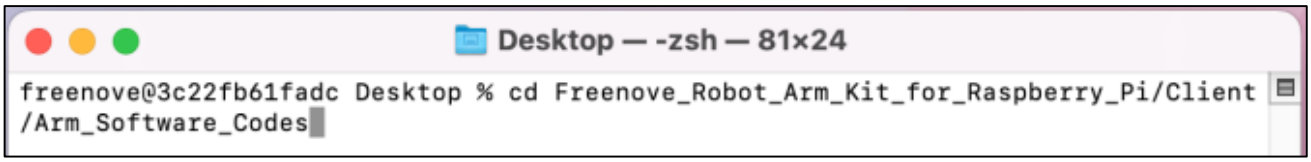


Find the folder "Freenove_Robot_Arm_Kit_for_Raspberry_Pi" on your computer. We place in on the desktop, as shown below.



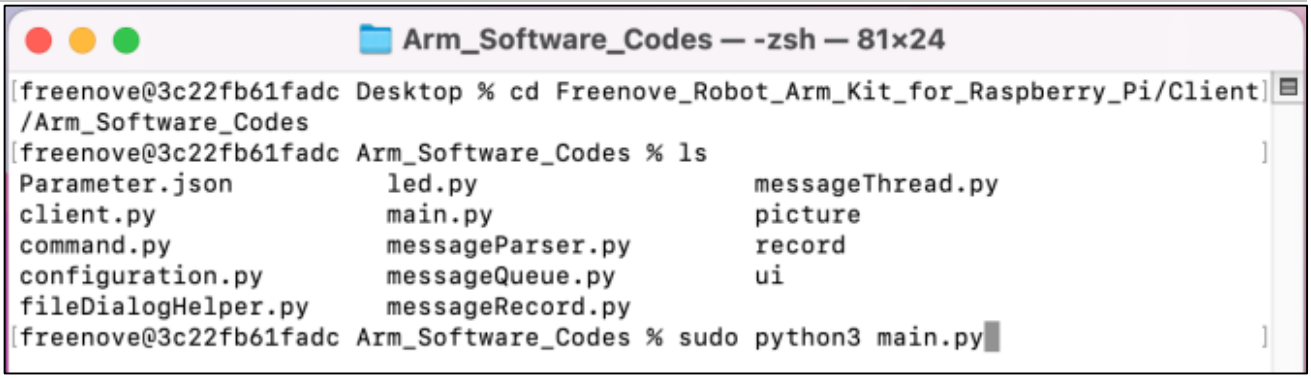
Enter the folder with the command:

```
cd Freenove_Robot_Arm_Kit_for_Raspberry_Pi/Client/Arm_Software_Codes
```

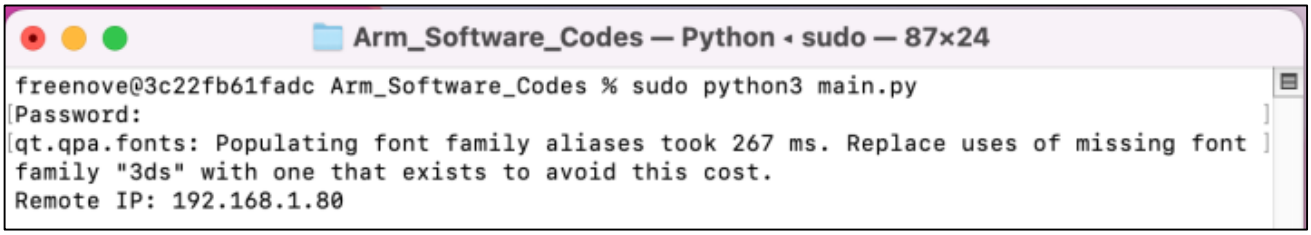


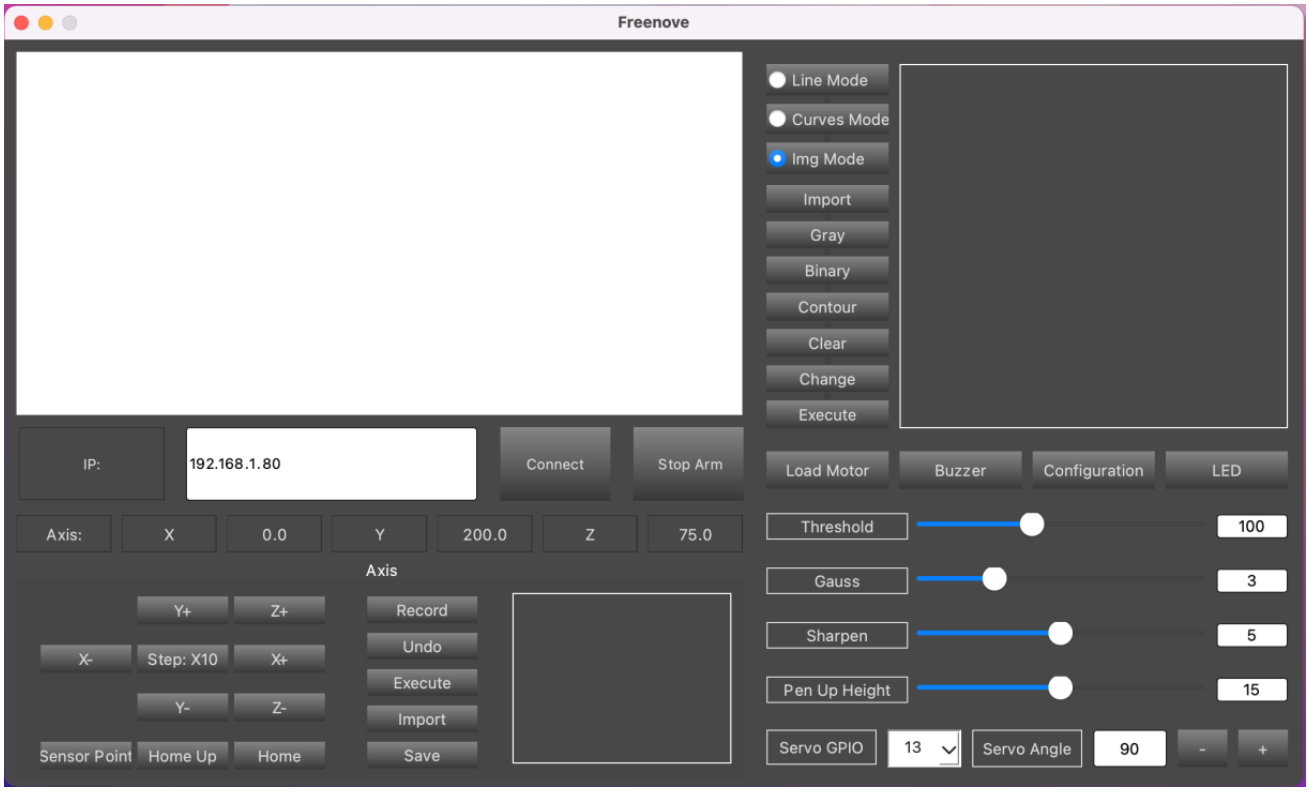
Run the command to start the software.

```
sudo python3 main.py
```



The interfaces are as shown below.





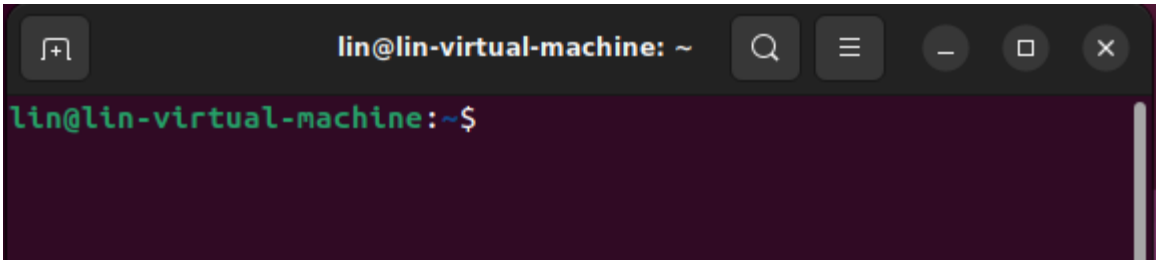
Please note the following:

1. The use of a proxy or other network tools may result in the robotic arm control software obtaining an incorrect network IP address.
2. Make sure your computer and Raspberry Pi are connected to the same local network to avoid communication failures.

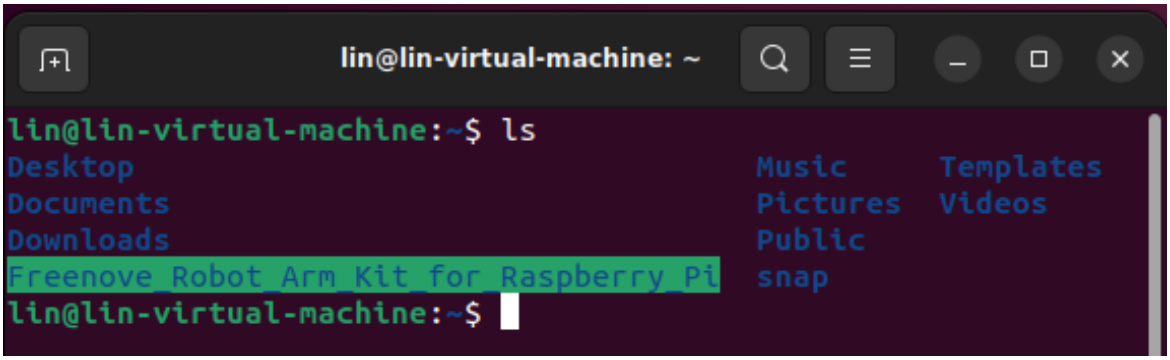
Once you have learned how to open the software for controlling the robotic arm, please proceed to the next step by clicking here.

Linux

Open the Terminal.

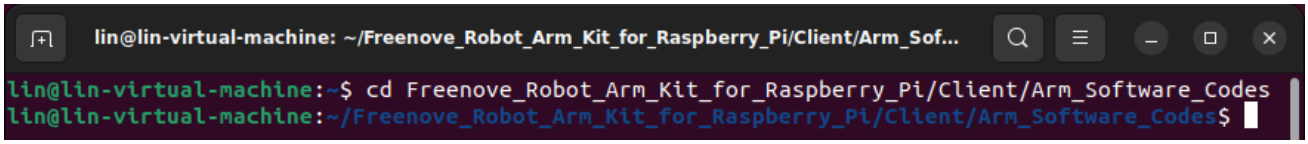


Open the folder "Freenove_Robot_Arm_Kit_for_Raspberry_Pi". We place it under the home directory, as shown below:



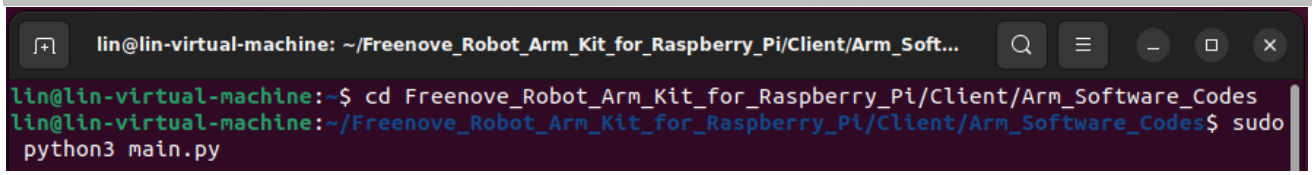
Enter the folder with the following command.

```
cd Freenove_Robot_Arm_Kit_for_Raspberry_Pi/Client/Arm_Software_Codes
```

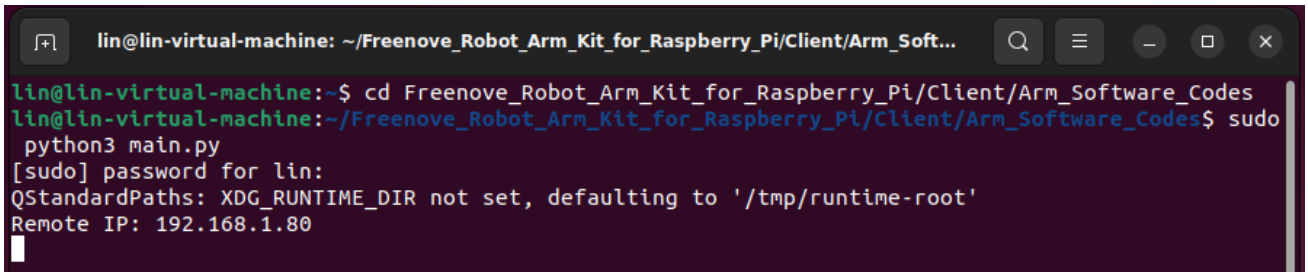


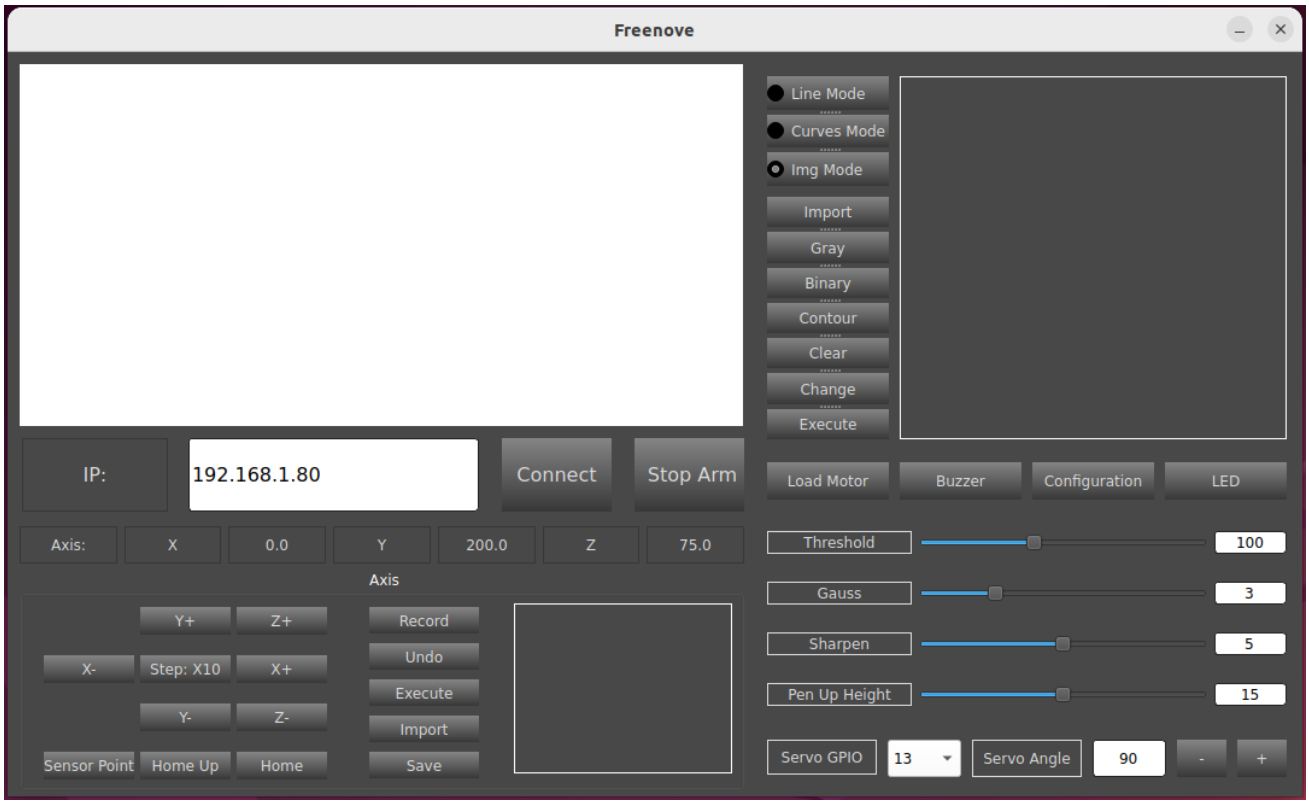
Run the command to start the software.

```
sudo python3 main.py
```



The interfaces are as shown below.





Please note the following:

1. The use of a proxy or other network tools may result in the robotic arm control software obtaining an incorrect network IP address.
2. Make sure your computer and Raspberry Pi are connected to the same local network to avoid communication failures.

Once you have learned how to open the software for controlling the robotic arm, please proceed to the next step by clicking here.

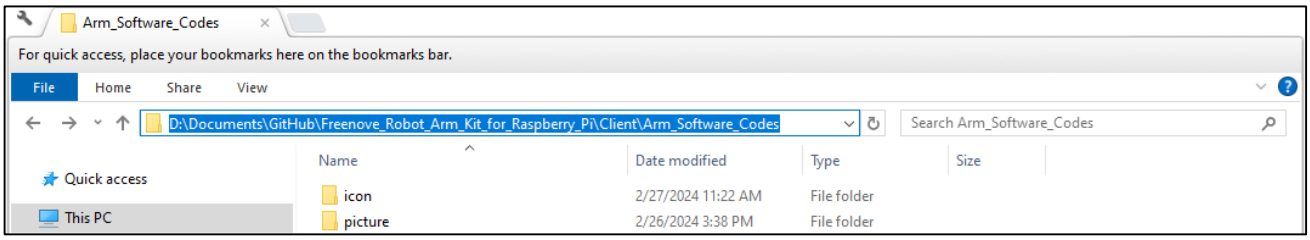
Packaging of Robotic Arm Control Software

If you prefer to run programs directly without having to run the code each time, you can execute the following packaging command to bundle the entire software into a single executable file.

This example will be based on the Windows operating system, but the operation is analogous for other systems.

1. Enter the directory where the software code locates.

Freenove_Robot_Arm_Kit_for_Raspberry_Pi/Client/Arm_Software_Codes



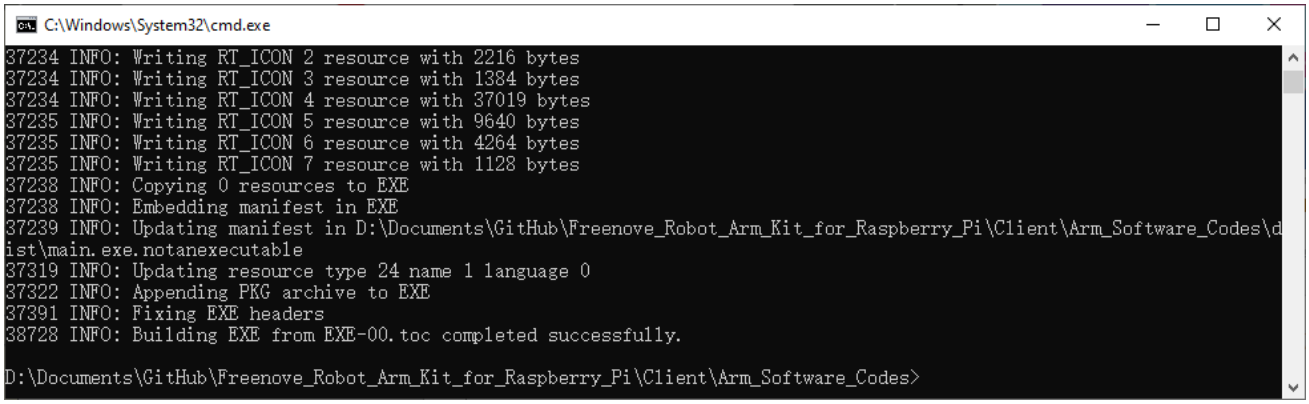
2. Type in cmd on the search bar to enter the terminal.

3. Run the following command and wait for the packaging process to complete.

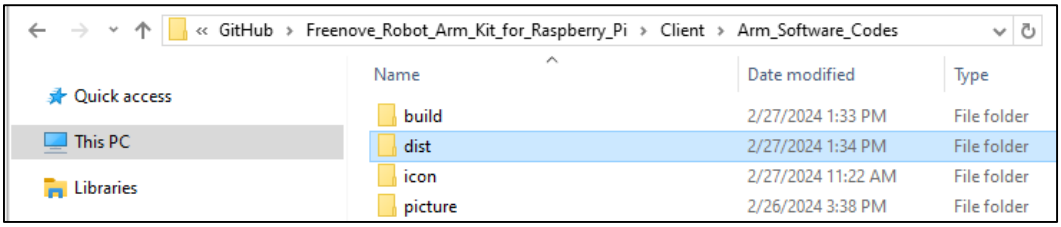
```
pyinstaller -F main.py
```

Or this command:

```
sudo pyinstaller -F main.py
```



4. Once the packaging is complete, the executable file will be saved in the 'dist' folder.

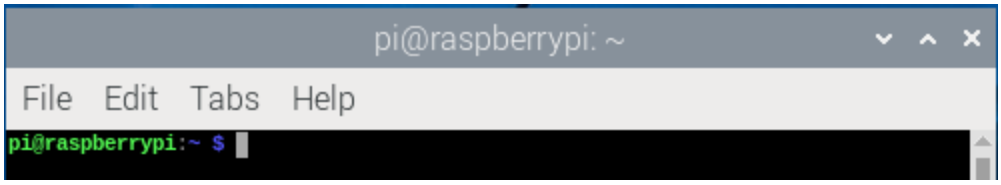


Note: If the packaging fails, you can delete the 'dist' folder and try the process again. Repeat the attempt several times until it is successful.

Chapter 4 Usage of the Software

Connecting to the Robot Arm

1. Open Raspberry Pi Terminal.



2. Enter the folder **Freenove_Robot_Arm_Kit_for_Raspberry_Pi/Server/Code.**

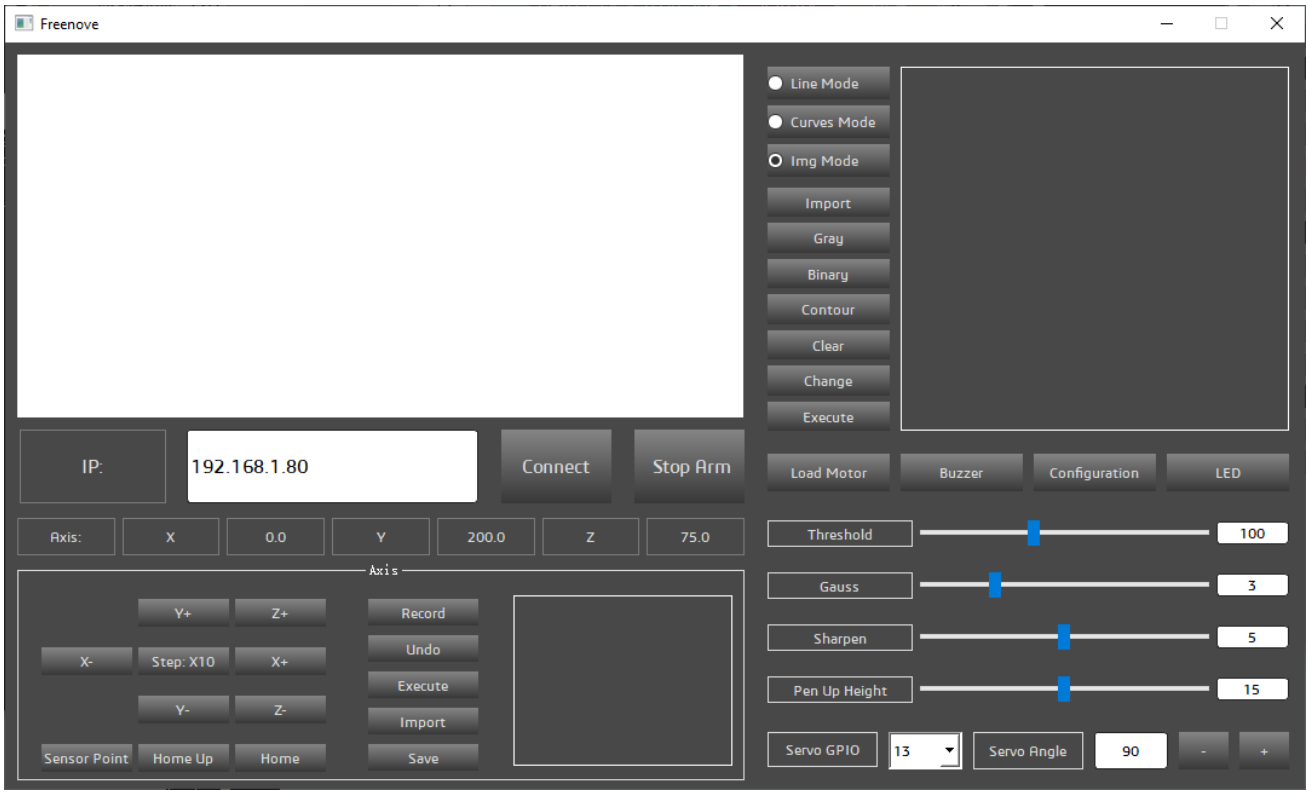
```
cd Freenove_Robot_Arm_Kit_for_Raspberry_Pi/Server/Code
```

3. Run the server.

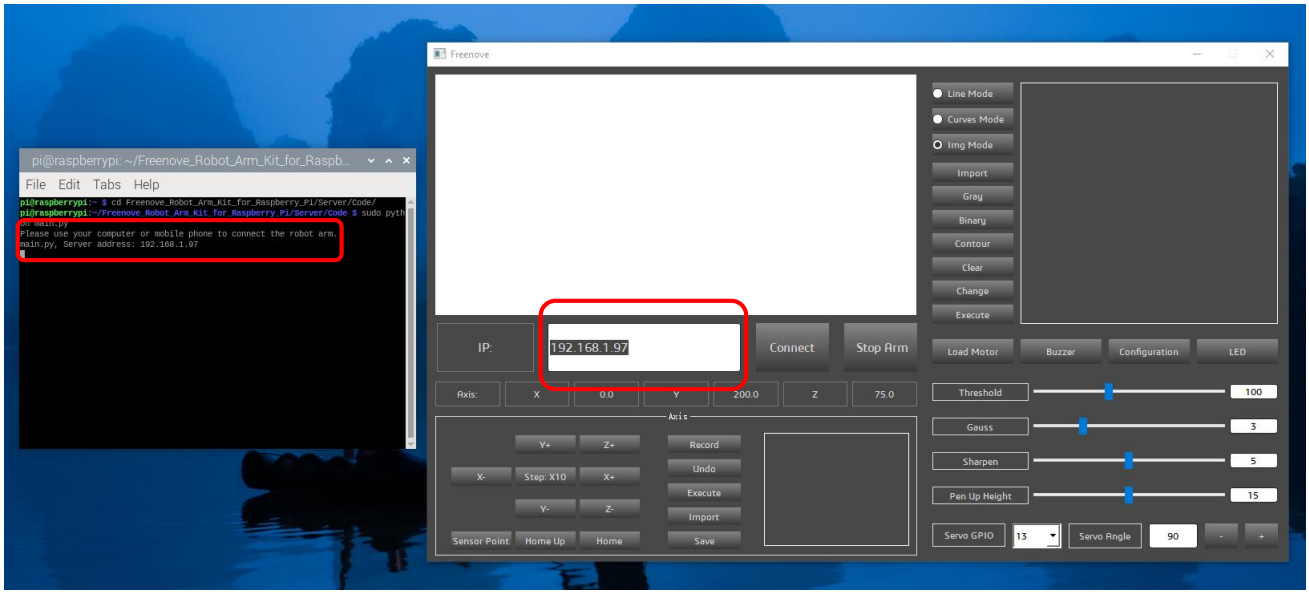
```
sudo python main.py
```

You can see the IP address of your Raspberry Pi printed.

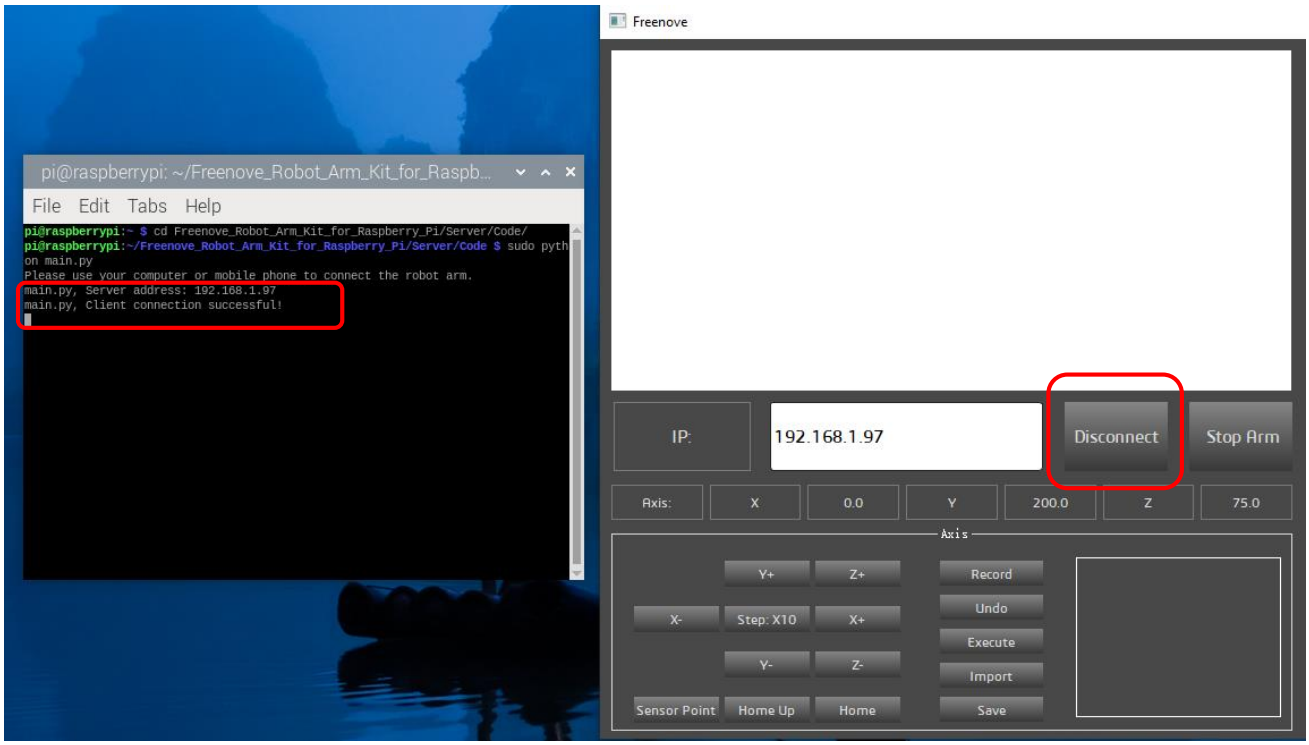
4. Make sure your computer and the Raspberry Pi are connected to the same local network. Open the software.



5. Enter the IP address of your Raspberry Pi printed above to the bar on the software.

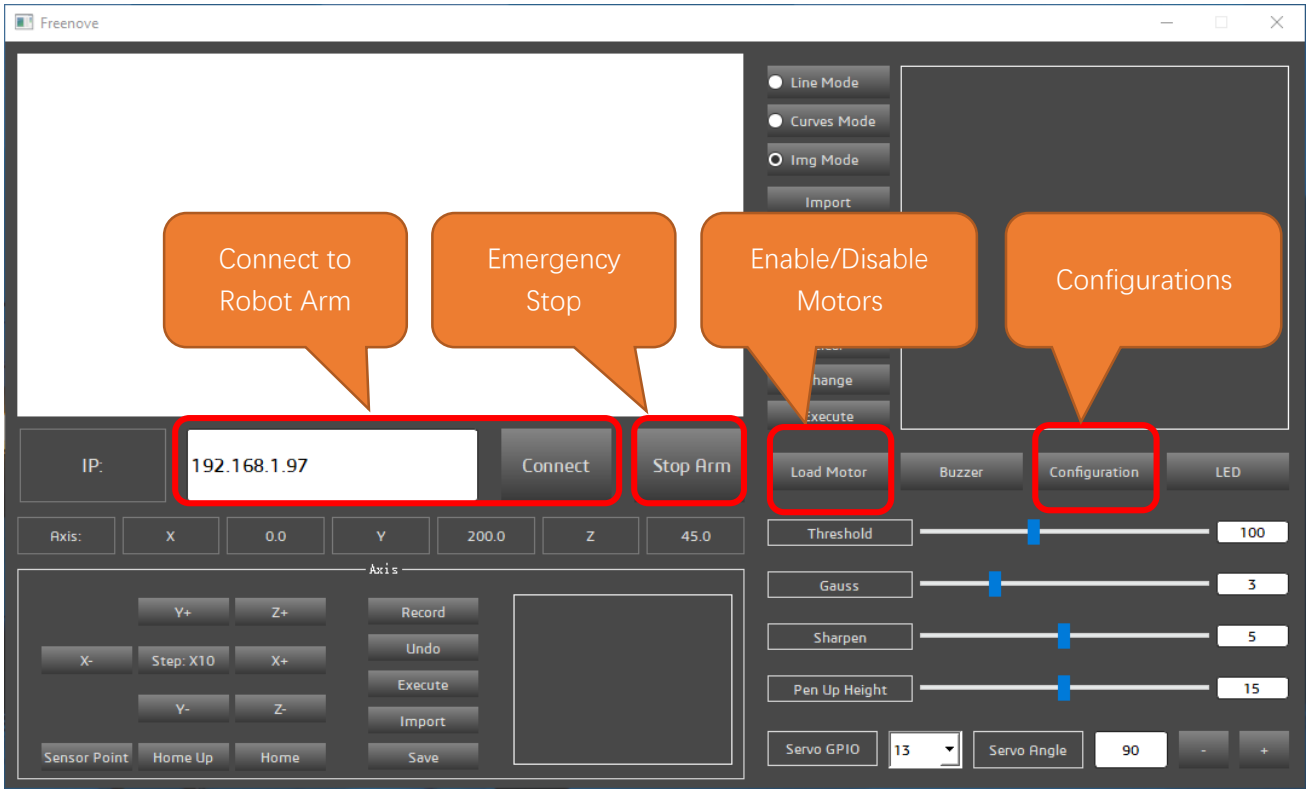


6. Click the Connect button and the software will establish communication with Raspberry Pi via WiFi.



Configuring Parameters for Robot Arm

Before using the robotic arm, please keep the following precautions in mind. They will help you use the mechanical arm more effectively:



Precautions for Use

1. **Initial Calibration**: The robotic arm does not contain calibration parameters when used for the first time. Therefore, a calibration configuration operation is necessary to ensure optimal performance. The calibration configuration information is recorded in a JSON file.

2. **Environment Changes**: If there is a change in the working environment of the robotic arm, please perform the calibration configuration again to adapt to the new environment.

3. **No Repeated Calibration**: If the working environment of the robotic arm does not change, the arm only requires a single calibration to function properly. There is no need to repeat the calibration configuration operation each time.

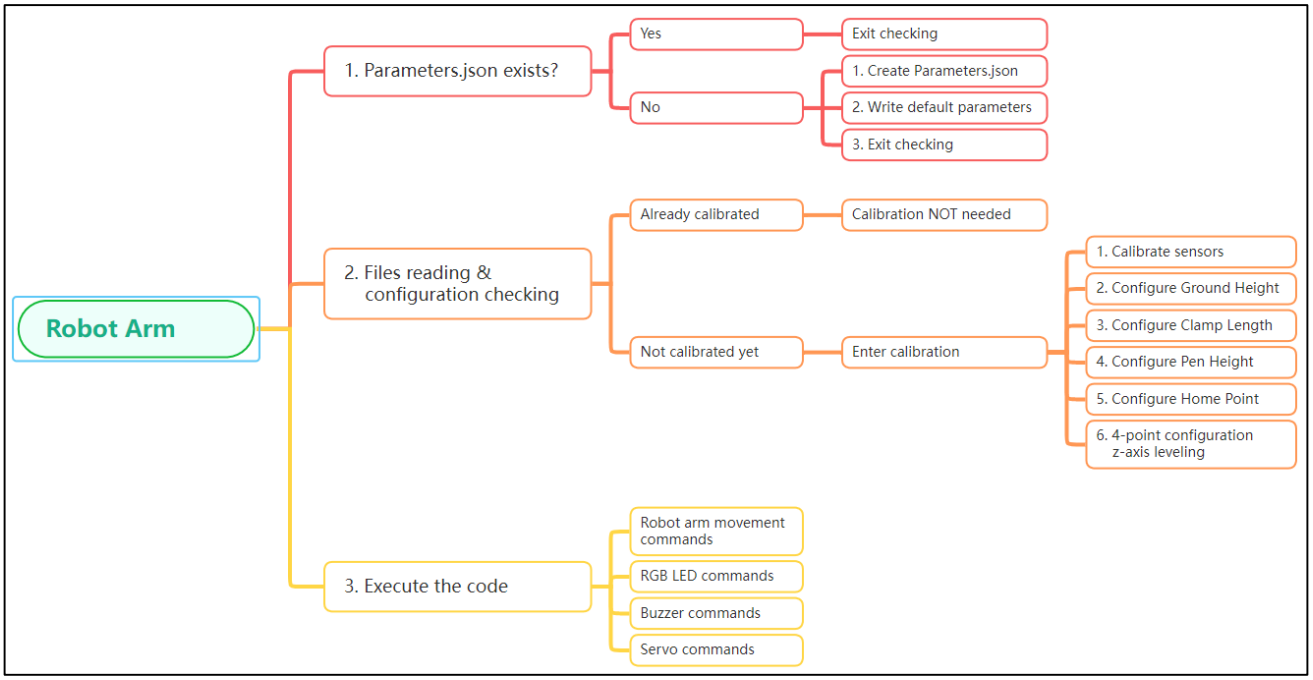
4. **Emergency Stop**: In case of abnormal operation of the robotic arm, you can force stop the arm using the "Stop Arm" button in the software. This will forcefully shut down all threads of the robotic arm and disable the motors, protecting both the motors and the circuit board. Alternatively, you can physically cut off the power supply to the motors by pressing the "Load" switch on the Raspberry Pi robotic arm's mainboard.

5. **Sensor Position Check**: Before using the software to control the robotic arm each time, ensure that the arm is set to the midpoint position detected by the sensor.

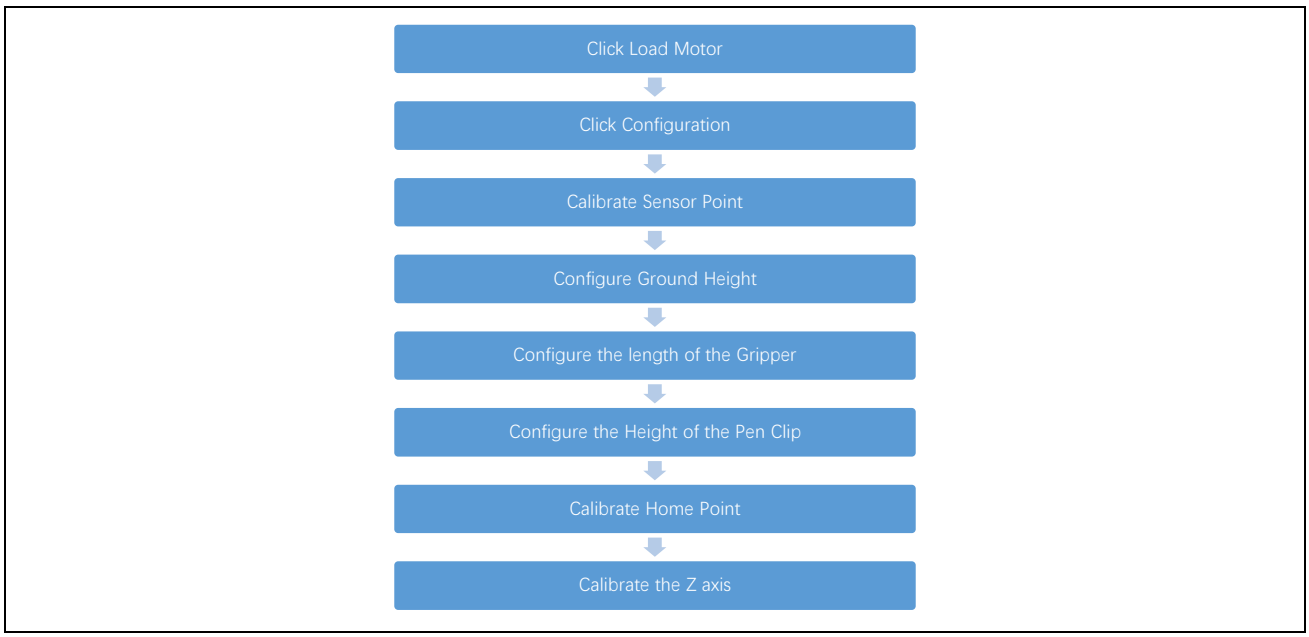
6. **Motor Activation for Calibration**: The calibration configuration operation requires the cooperation of the motors; thus, if the motors are not enabled first, it will not be possible to open the calibration configuration interface.

Robotic Arm Configuration Parameters

Each time the robotic arm is powered up, it undergoes a series of complex checks, as illustrated in the figure below.

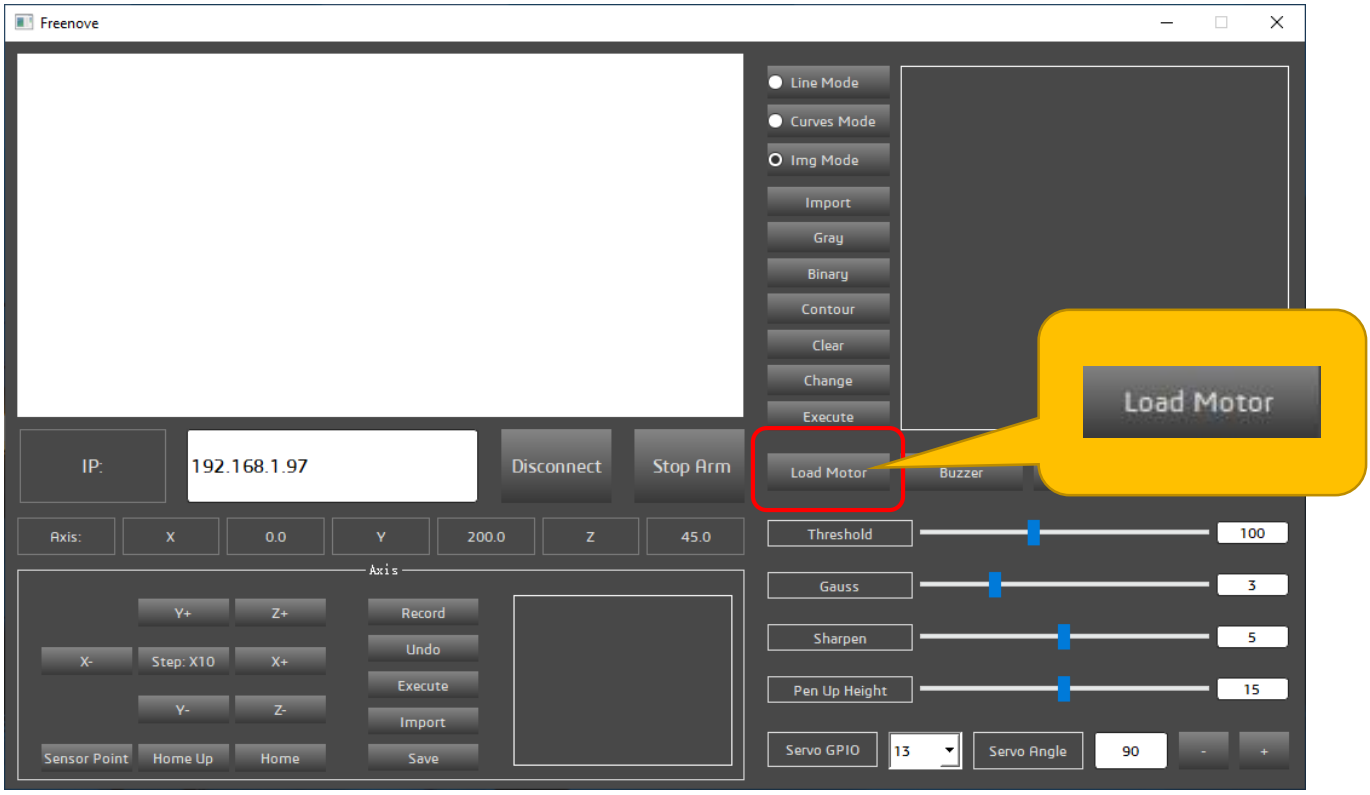


If this is the first time the robotic arm is being powered on, it's essential to perform a calibration configuration to ensure optimal operation. Below is the configuration process for the robotic arm, which we will explain step by step, following the flowchart provided.



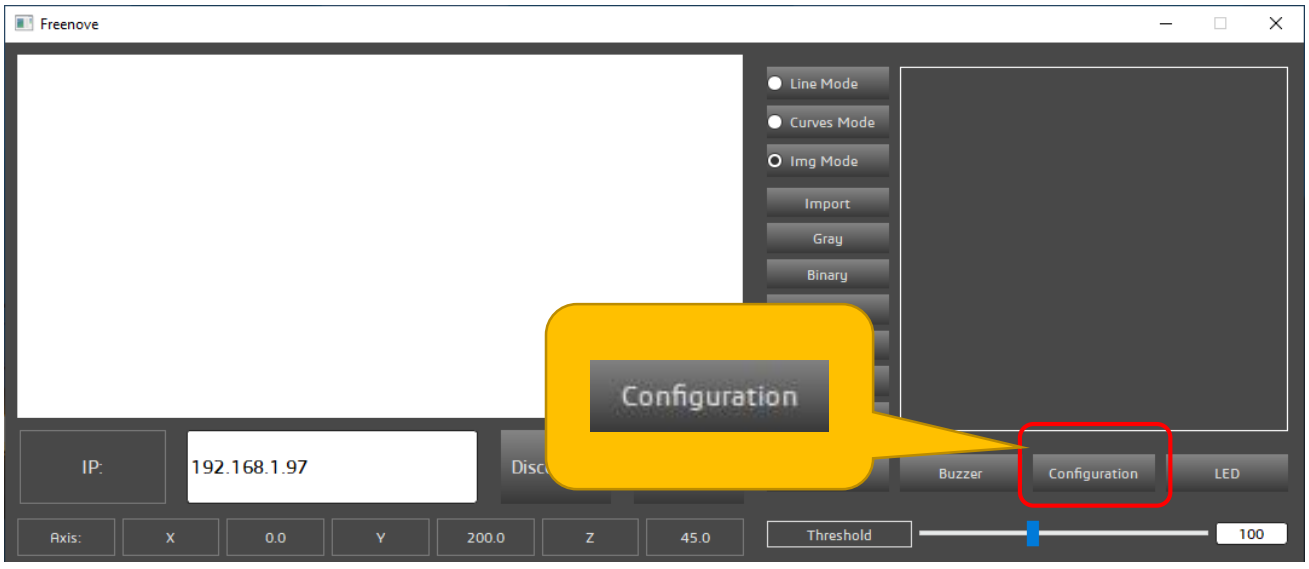
Enable Motors

In its default state, the robotic arm is in a condition where the motors are disabled to ensure safety. Therefore, the first step is to enable the motors by clicking the "Load Motor" button in the software. This action will energize the motors and prepare them for the calibration sequence that follows.

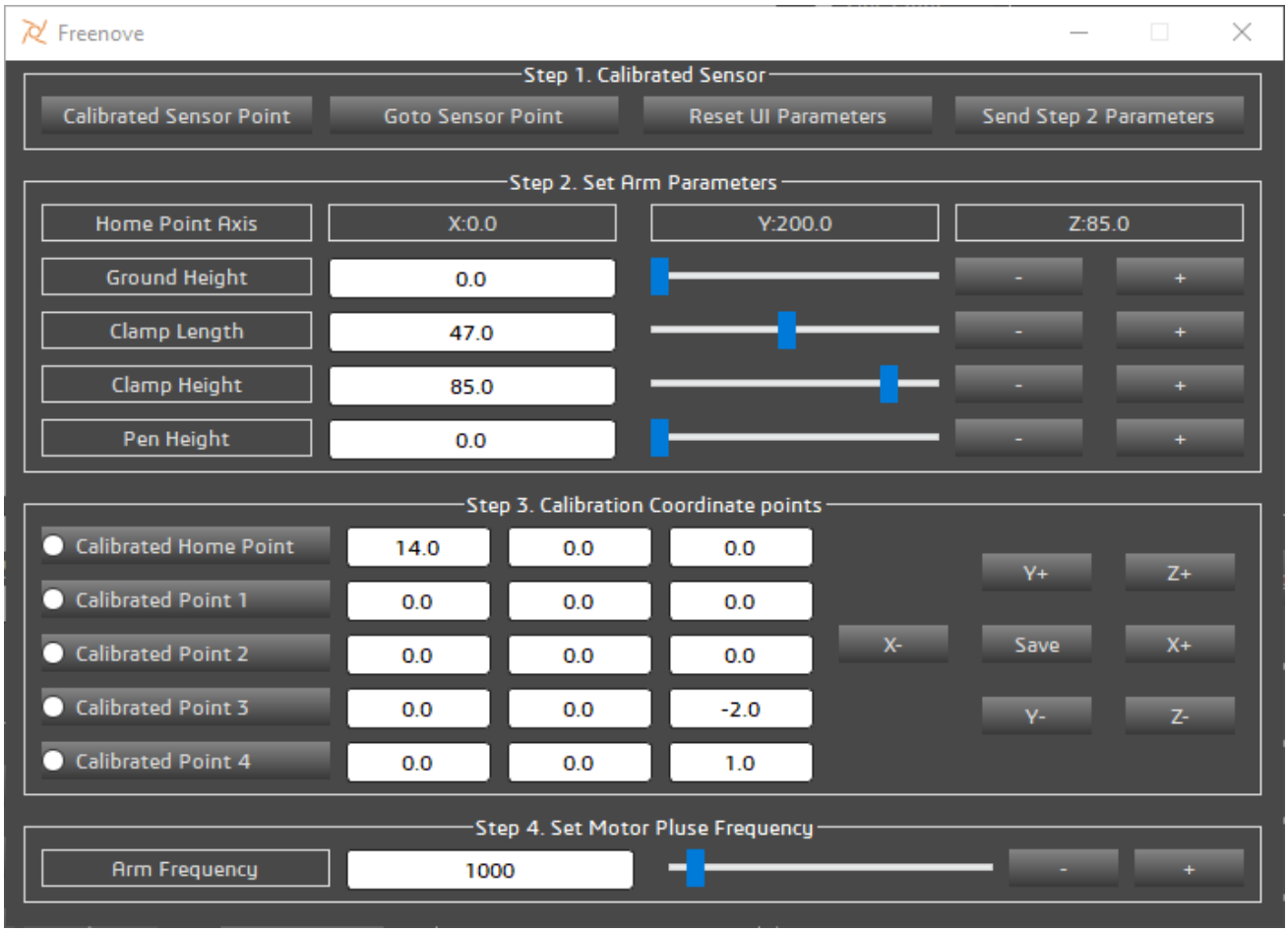


Open the Configuration Interface

Ensure that you have clicked "Load Motor" first, as this is a prerequisite for enabling the motors. Once the motors are enabled, you can proceed to click on "Configuration." Doing so will open the robotic arm's configuration interface.



The interface is as illustrated below.

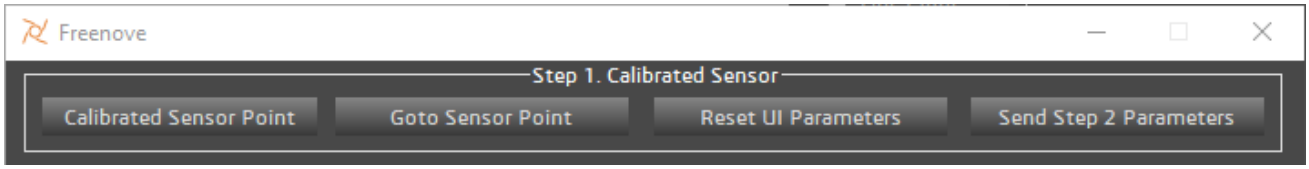


As illustrated in the above figure, the configuration interface has been divided into four sections. Below, we will explain the configuration of the robotic arm step by step, covering each of these sections.

Step 1. Calibrated Sensor

The sensors on the robotic arm are crucial for assisting the arm in positioning its own range of motion. Without them, the robotic arm would not be able to determine its true coordinate position. Therefore, for the robotic arm to function properly and stably, the calibration of the sensors is extremely important.

At the first use of the robot arm, please click the Calibrated Sensor Point button to calibrate the sensors.



Calibrated Sensor Point is used for calibrating the sensors on the robotic arm. This process involves moving the arm back and forth three times across the sensor. The purpose of this movement is to calculate the offset value where the arm rotates to the center of the sensor. By determining this offset, the system can adjust the arm's movements to ensure it accurately positions itself at the sensor's midpoint.

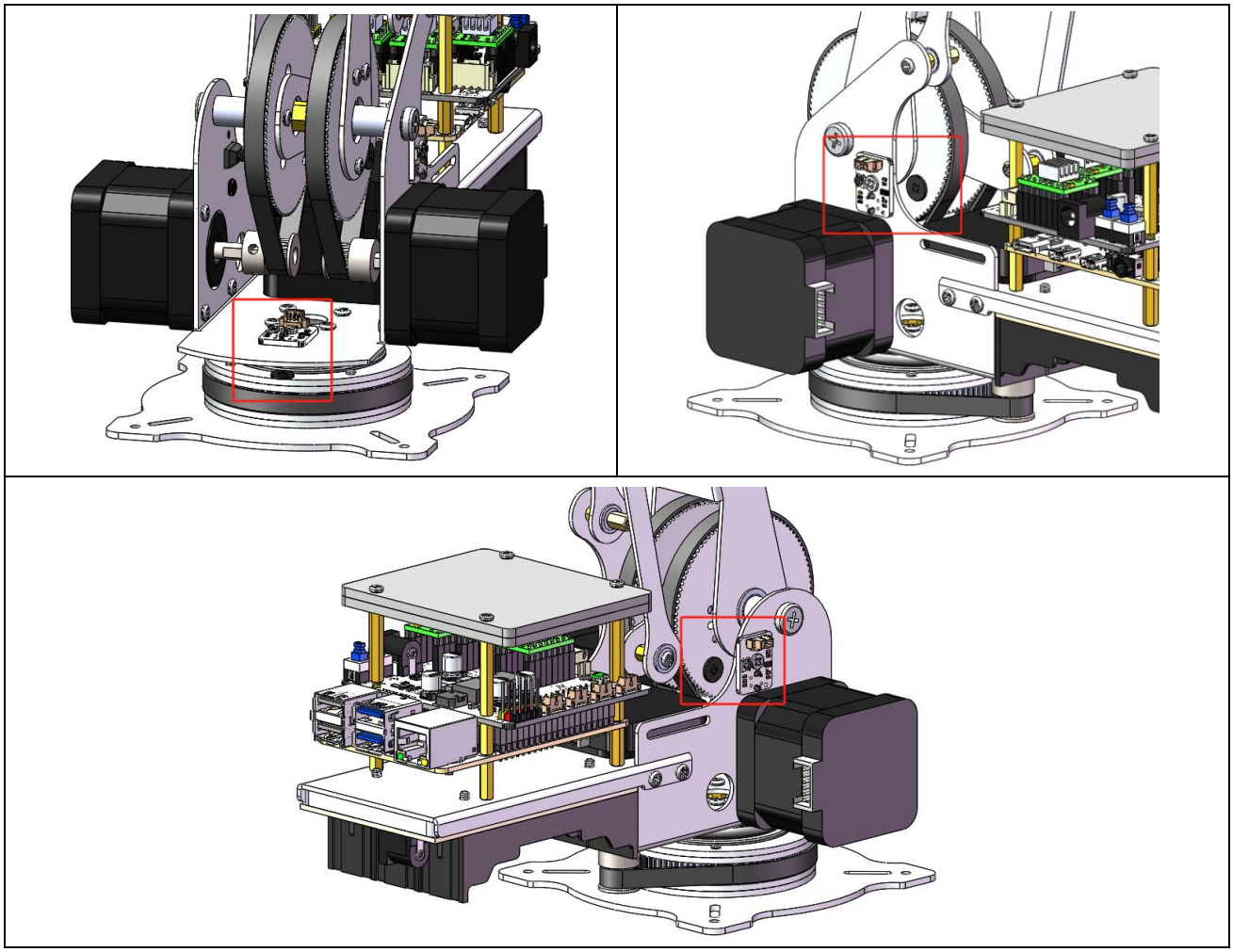
Goto Sensor Point is designed to return the robotic arm to the center position of the sensor. The behavior of this function depends on whether the sensors have been calibrated:

-Uncalibrated Condition: If the sensors have not been calibrated and you click on "Goto Sensor Point," the robotic arm will move to the edge position of the sensor.

-Calibrated Condition: Once the sensors have been calibrated and you click on "Goto Sensor Point," the robotic arm will accurately move to the center position of the sensor.

-For the initial setup of the robotic arm, please click "**Calibrated Sensor Point**", rather than "Goto Sensor Point".

-From the second time onwards, after the sensors have been calibrated, you can directly use the "Goto Sensor Point" function.

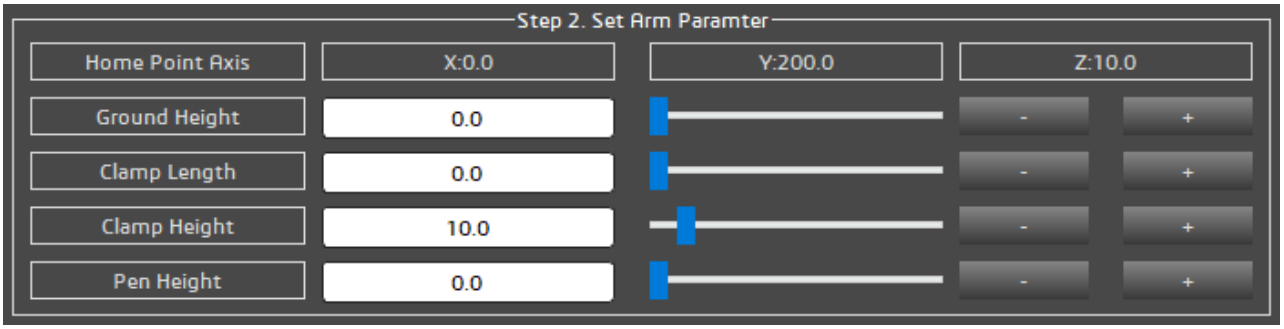


Reset UI Parameters is to restore all parameters on the user interface to their default settings.

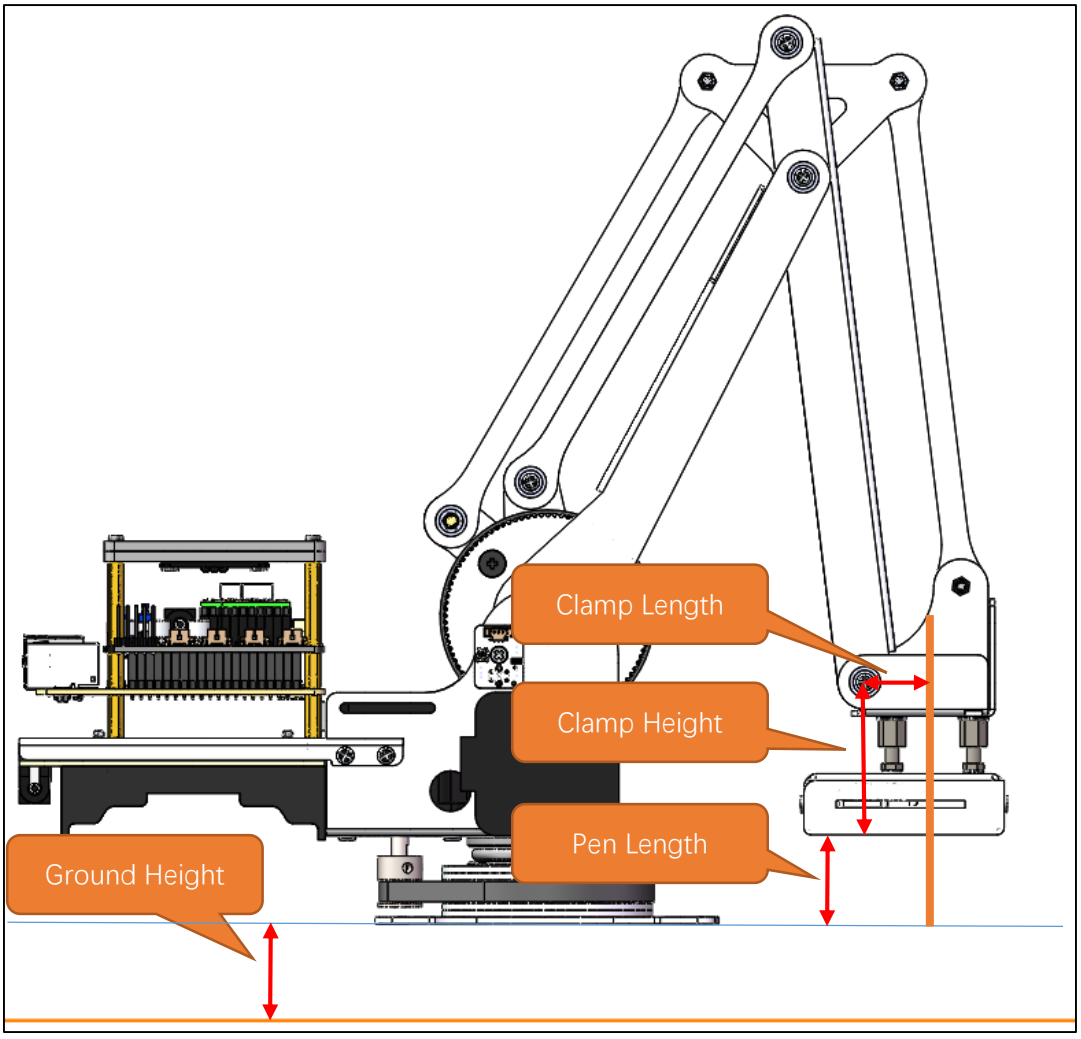
Send Step 2 Parameters is used to transmit the configuration information from Step 2 of the setup process to the robotic arm.

Step 2. Set Arm Parameters

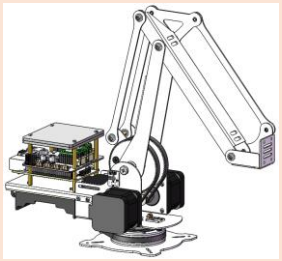
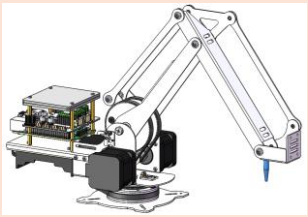
Building upon the first step, we proceed to configure Step 2. You can adjust these values using sliders or buttons within the user interface. These parameters are crucial for the proper functioning of the robotic arm, so it's important to set them carefully. Please refer to the recommended values provided in the documentation.

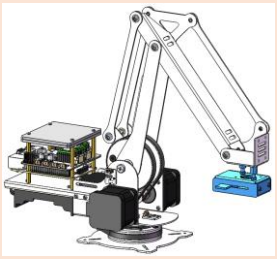
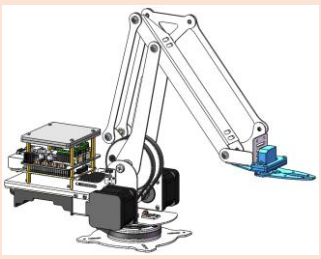
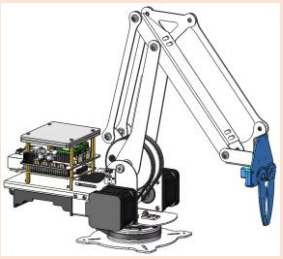


Ground Height refers to the height of the base of the robotic arm from the plane of the coordinate system.
Clamp Length is the length of the end effector, such as a clamp or clamp, at the end of the robotic arm.
Clamp Height is the height of the end effector from the ground.
Pen Height is the distance from the ground to the pen clip when the robotic arm is equipped with a pen clip.



The robotic arm operates in four distinct states based on its end effector configuration: No End Effector, Pen clip Installed, Clamp Installed (Horizontal), and Clamp Installed (vertical).

No End Effector	Locating Post
	

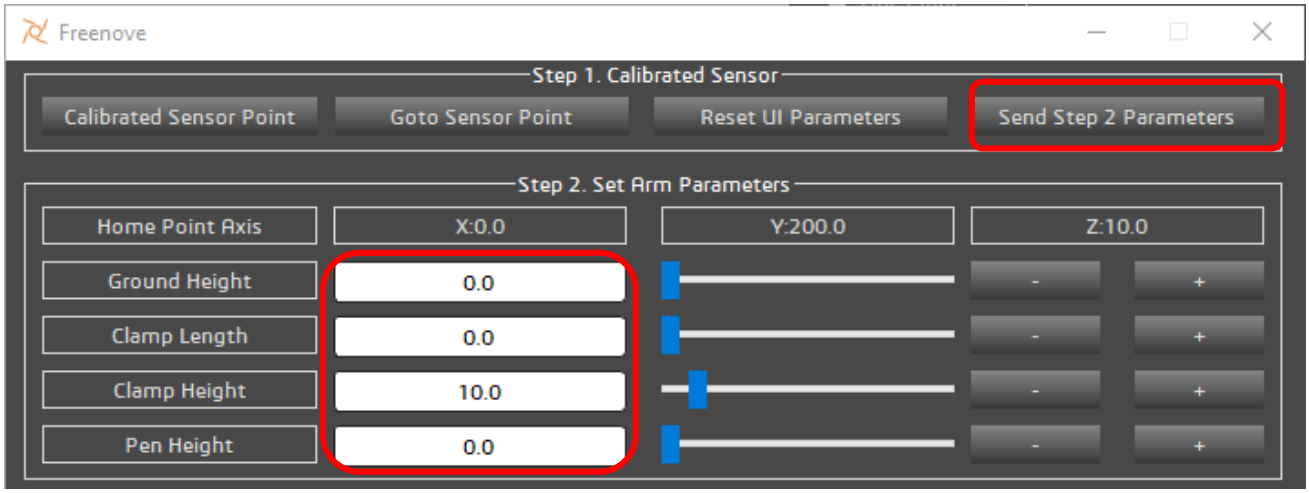
pen clip installed	Clamp Installed (Horizontal)	Clamp Installed (vertical)
		

The recommended parameters are as follows:

Mode	Ground Height	Clamp Length	Clamp Height	Pen Height
No End Effector	0mm	0mm	10mm	0mm
Locating Post	0mm	0mm	10mm	30mm
Pen Clip Installed	0mm	15mm(10.5+ radius of pen)	45mm	30mm(Recommended value for pen height)
Clamp Installed (Horizontal)	0mm	70mm	24mm	0mm
Clamp Installed (vertical)	0mm	47mm	85mm	0mm

No End Effector Mode

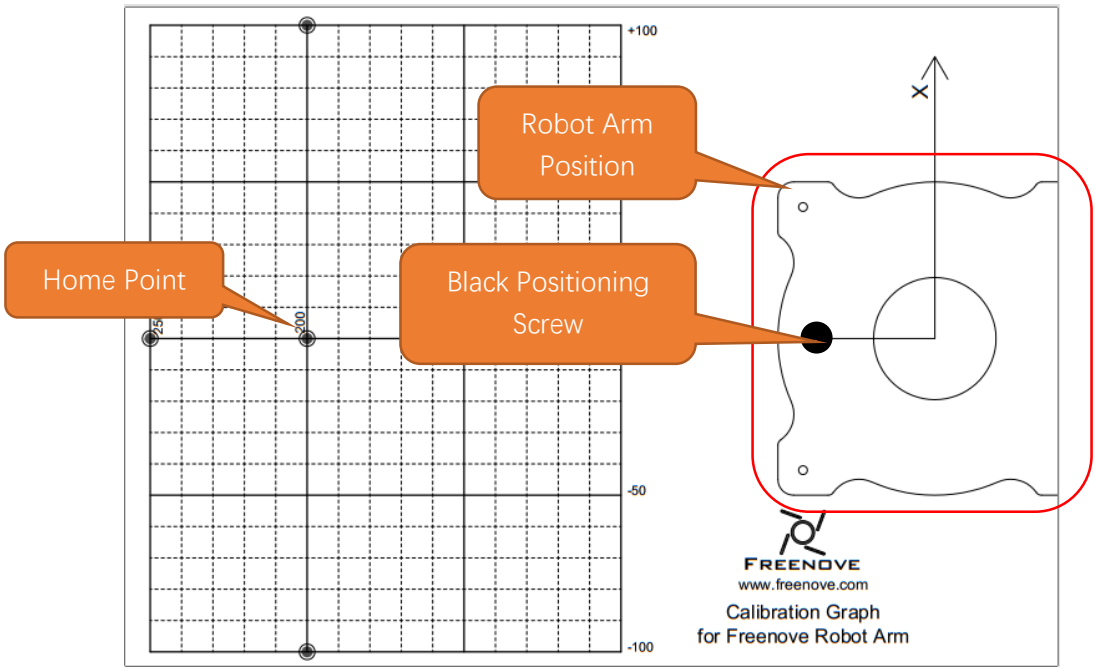
After you have finished configuring the parameters for Step 2, proceed to click on the "Send Step 2 Parameters" button located in Step 1. For this example, let's assume you are working in the "No End Effector" mode, which means the robotic arm is not equipped with any additional tools or clamps. The configurations are as shown below.



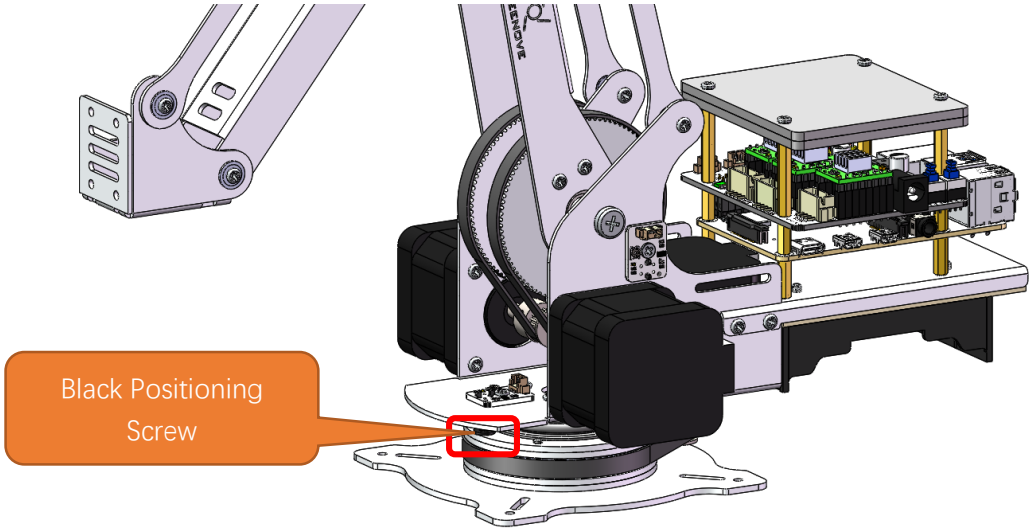
Home Point Calibration

Once Step 2 of the configuration is complete, the robotic arm should be able to operate normally. However, you might notice that the arm's movements are not as precise as expected. This lack of accuracy can be attributed to mechanical deviations. To address these deviations, we can configure to calculate the mechanical angle deviations and apply compensatory adjustments to the angles used by the control system.

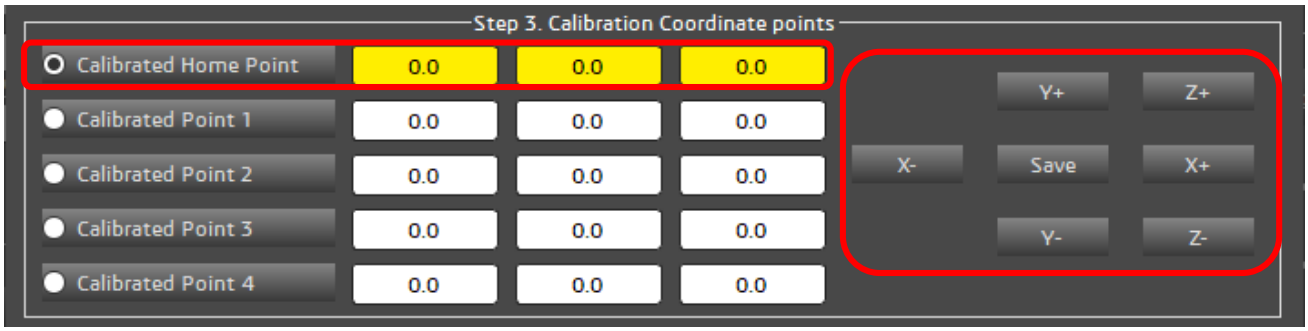
To proceed with the calibration process, you will need to locate the calibration paper. If you cannot find the calibration paper, you can print it out using an A4 sheet from a printer. The location of the calibration paper file is **Freenove_Robot_Arm_Kit_for_Raspberry_Pi/Calibration graph.pdf**



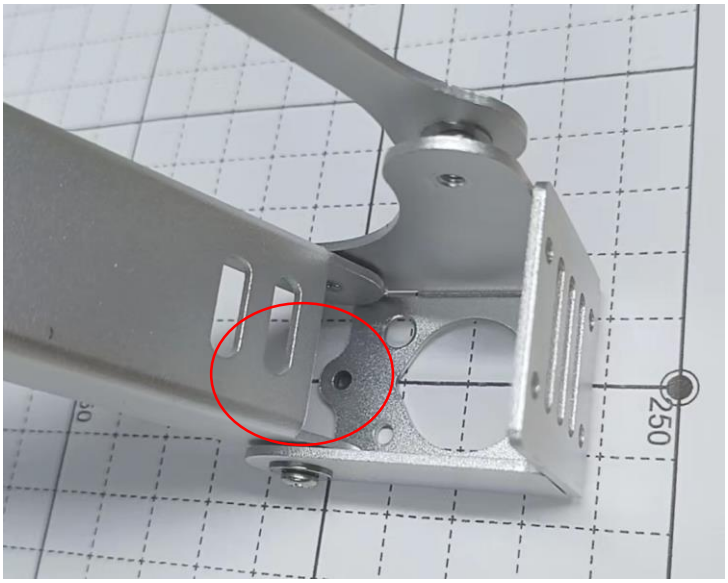
Place the robot arm to the Calibration graph. Align the base to the diagram. Pay attention to the position of the black screw.



Click on "Calibrated Home Point" under Step 3. Use the buttons on the right to adjust the position of the robotic arm, guiding it to move to the (0, 200) coordinates marked on the calibration paper. Once the arm is correctly positioned, click the "Save" button to record this calibrated position.

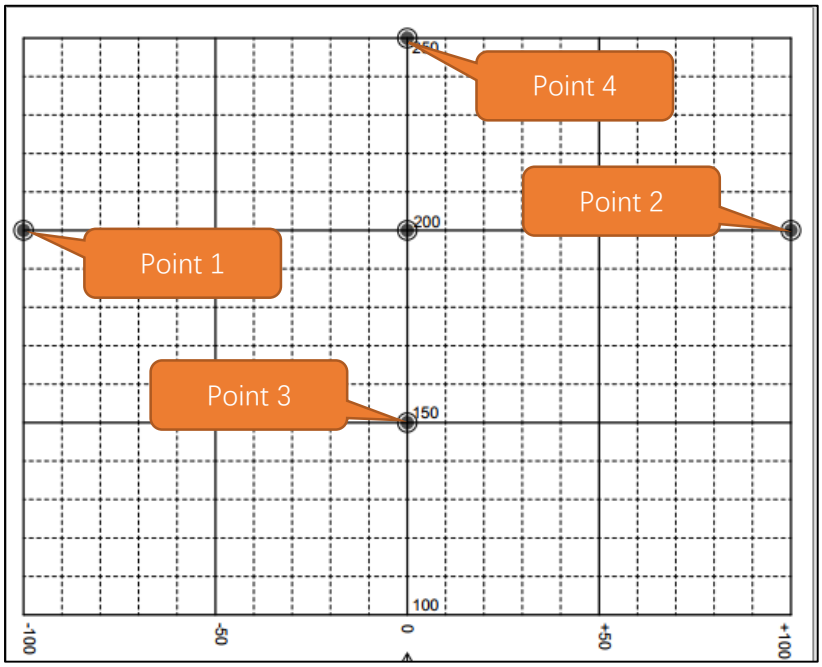


Position the robotic arm's end so that the alignment hole at the tip moves to the (0, 200) coordinates on the calibration paper.

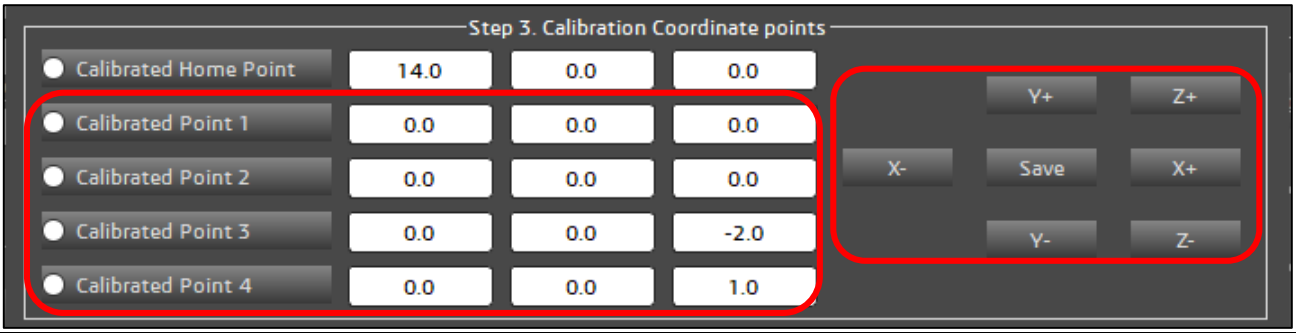


Z Axis Calibration

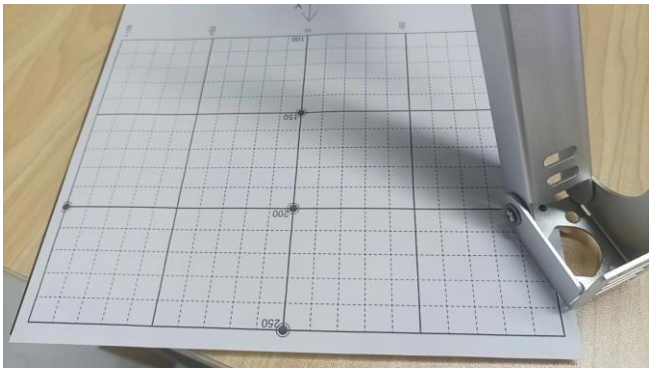
Structural imperfections can sometimes lead to minor inaccuracies in the z-axis coordinate of the robotic arm when it moves to different positions on the calibration paper. To correct for this, we utilize the Calibrated Points 1-4 as part of Step 3.

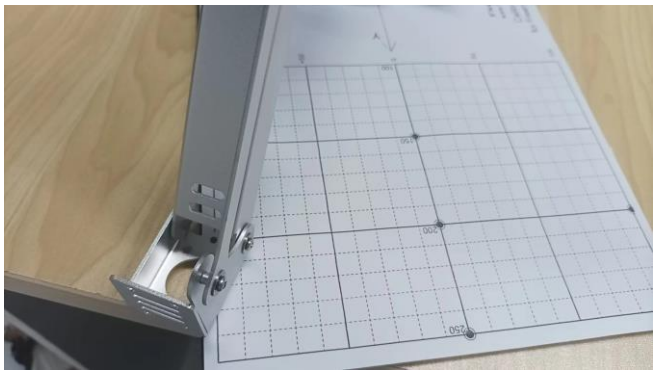
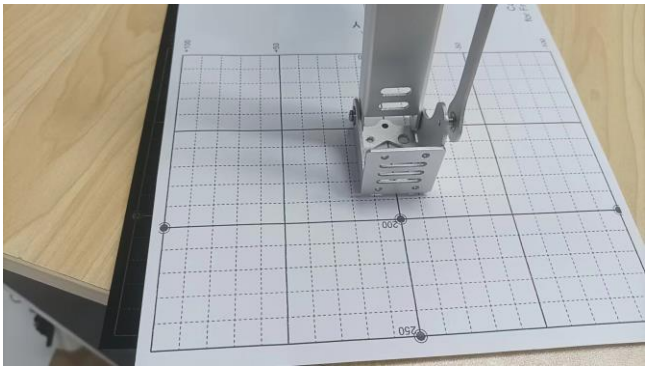


Select the corresponding calibration point, fine-tune the robotic arm through the buttons on the right, move the robotic arm to the corresponding point, and then click Save. Calibrate these 4 points from top to bottom.



Point 1



Point 2 
Point 3 
Point 4 

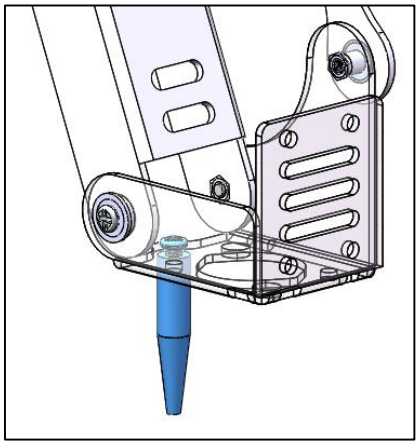
At this point, the calibration of the robotic arm is completed. You can choose to close this interface and return to the main interface to control the robotic arm.

Locating Post Mode

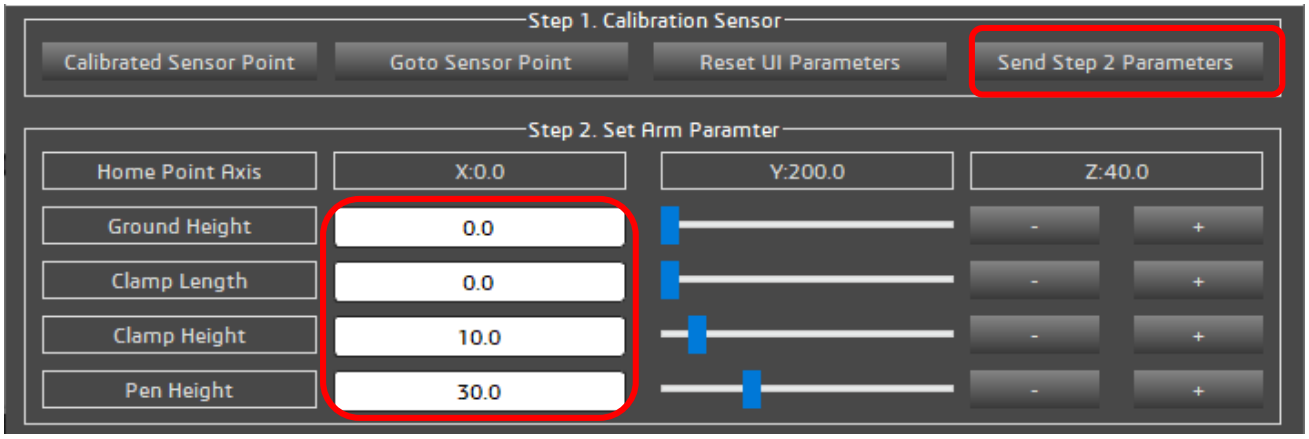
Installing the Locating Post

After you have finished configuring the parameters for Step 2, proceed to click on the "Send Step 2 Parameters" button located in Step 1. For this example, let's assume you are working in the " Locating Post " mode, means that the arm needs to be fitted with a locating Post.
Please find the Locating Post for the M6x30 from the Machinery Parts.

Using one M3x5 screw, secure it to the end of the arm. As shown in the following picture.

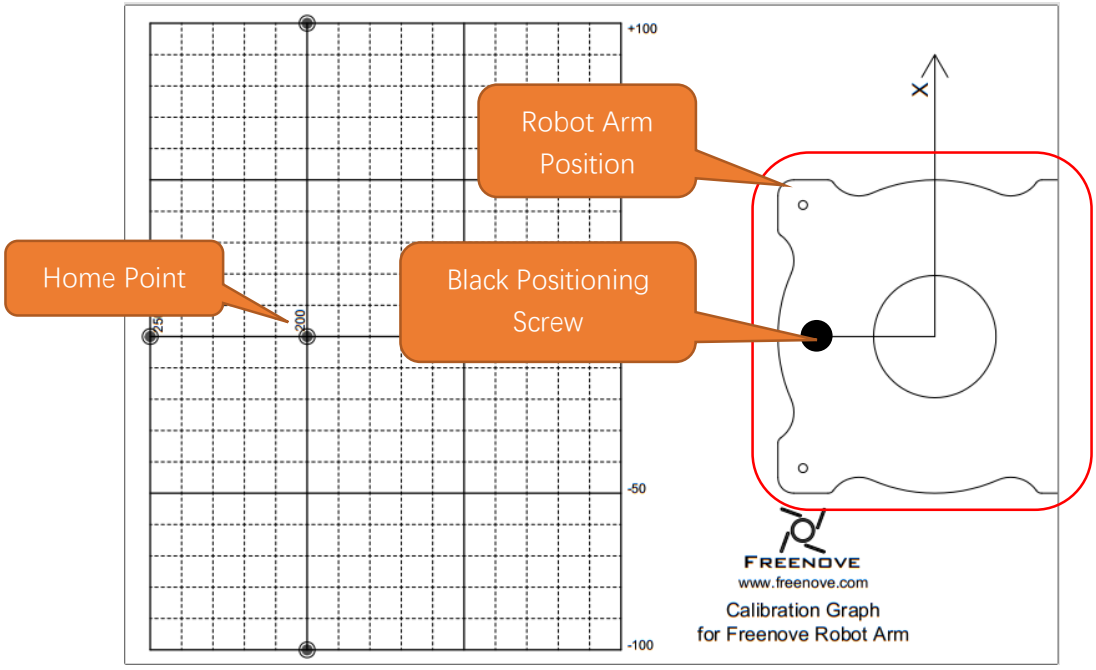


The configurations are as shown below.

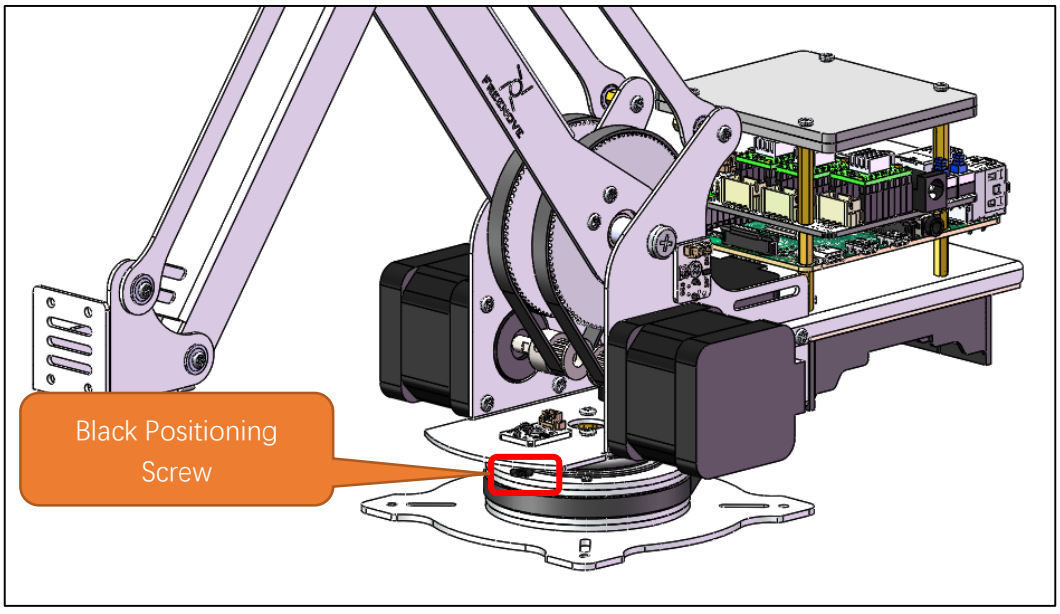


Home Point Calibration

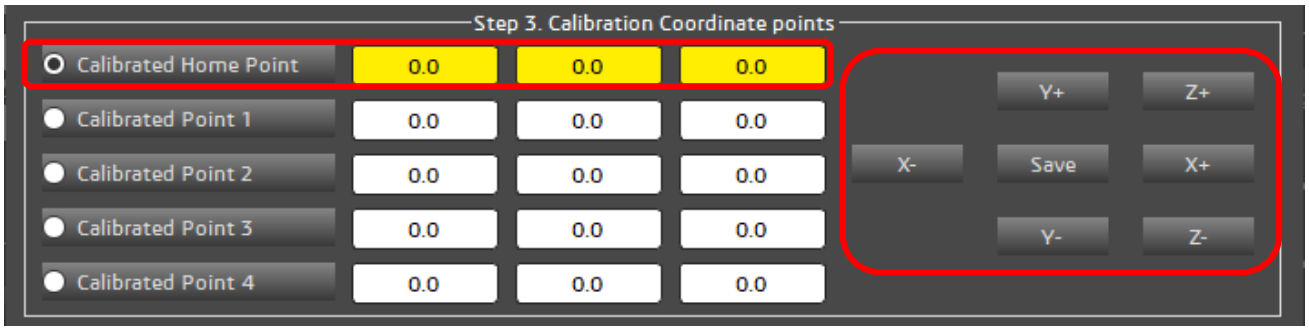
Once Step 2 of the configuration is complete, the robotic arm should be able to operate normally. However, you might notice that the arm's movements are not as precise as expected. This lack of accuracy can be attributed to mechanical deviations. To address these deviations, we can configure to calculate the mechanical angle deviations and apply compensatory adjustments to the angles used by the control system.
To proceed with the calibration process, you will need to locate the calibration paper. If you cannot find the calibration paper, you can print it out using an A4 sheet from a printer. The location of the calibration paper file is **Freenove_Robot_Arm_Kit_for_Raspberry_Pi/Calibration graph.pdf**



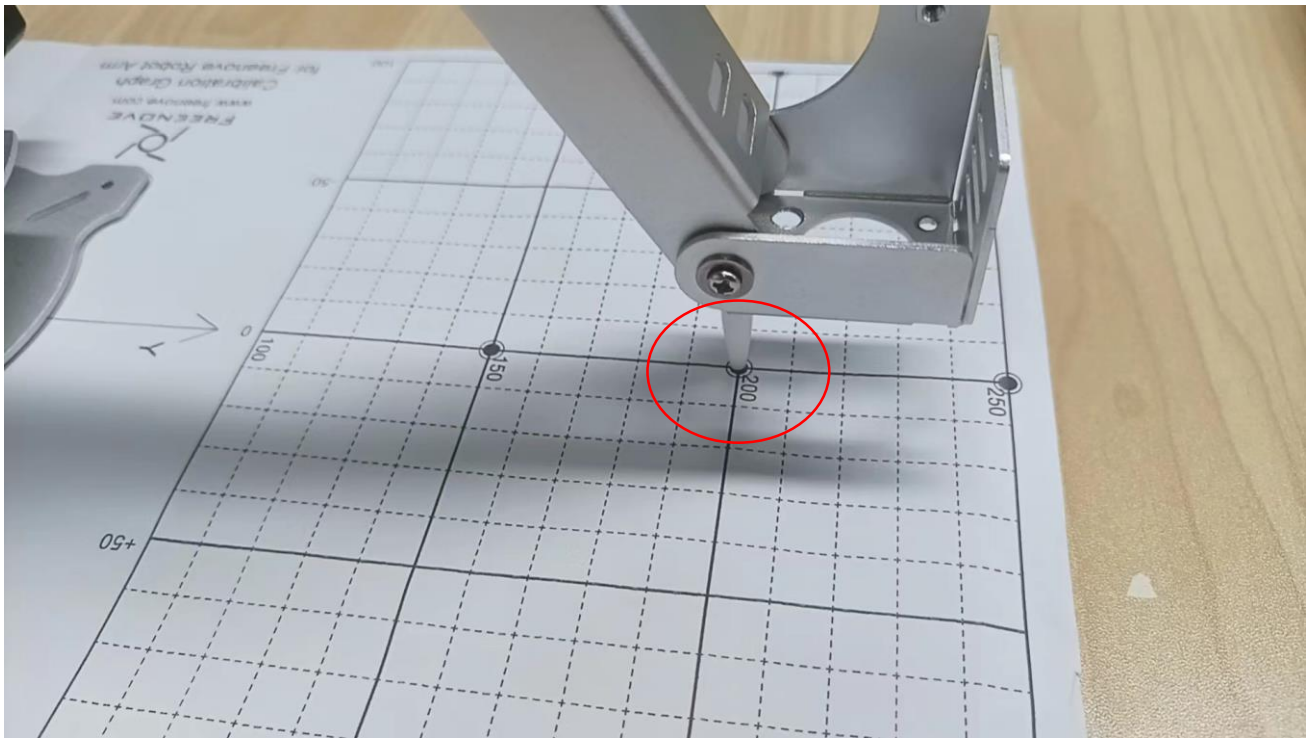
Place the robot arm to the Calibration graph. Align the base to the diagram. Pay attention to the position of the black screw.



Click on "Calibrated Home Point" under Step 3. Use the buttons on the right to adjust the position of the robotic arm, guiding it to move to the (0, 200) coordinates marked on the calibration paper. Once the arm is correctly positioned, click the "Save" button to record this calibrated position.

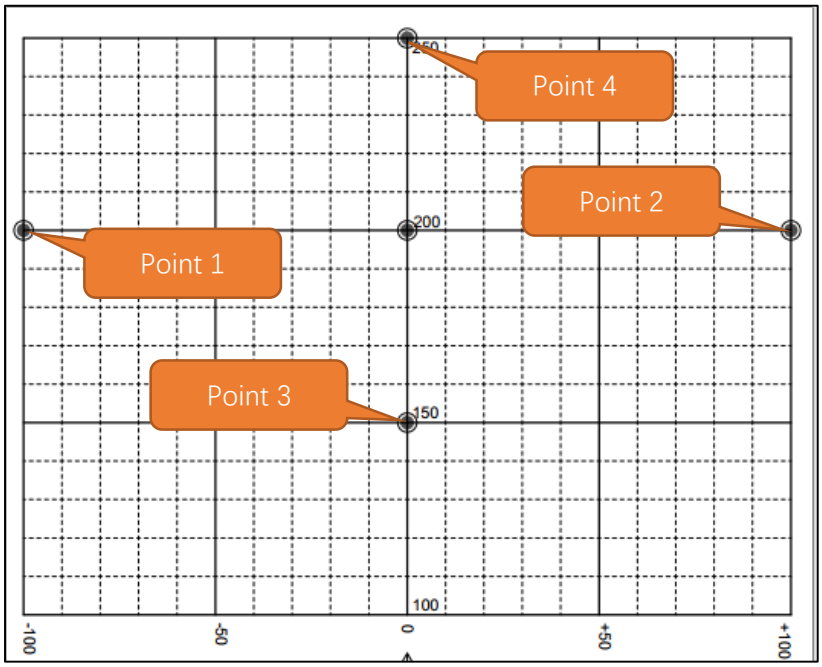


Position the robotic arm's end so that the alignment hole at the tip moves to the (0, 200) coordinates on the calibration paper.

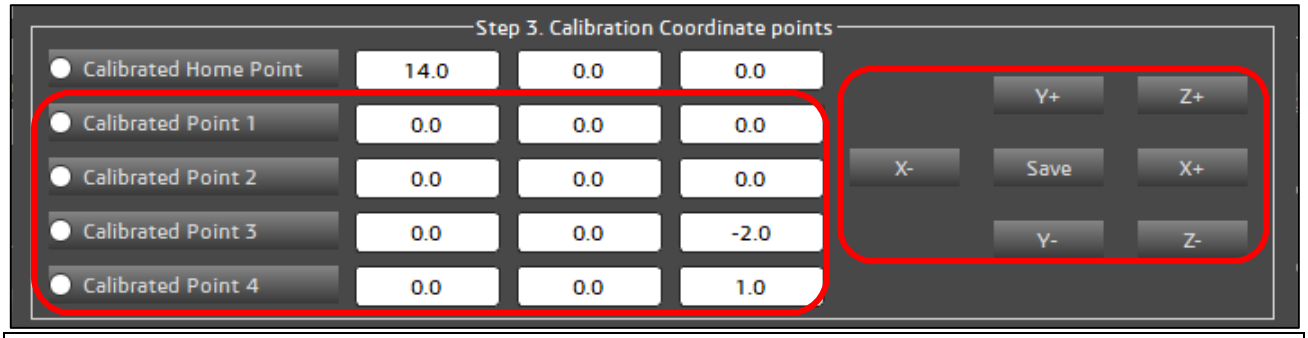


Z Axis Calibration

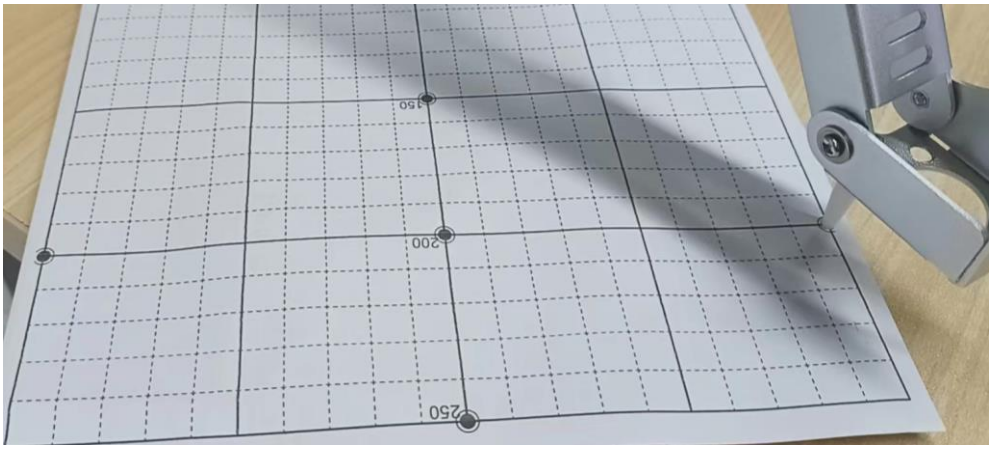
Structural imperfections can sometimes lead to minor inaccuracies in the z-axis coordinate of the robotic arm when it moves to different positions on the calibration paper. To correct for this, we utilize the Calibrated Points 1-4 as part of Step 3.



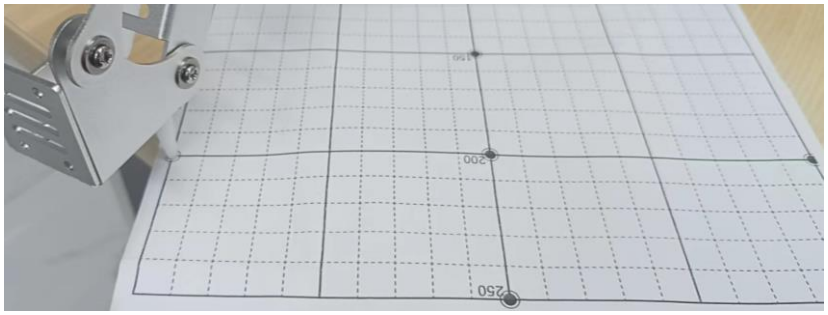
Select the corresponding calibration point, fine-tune the robotic arm through the buttons on the right, move the robotic arm to the corresponding point, and then click Save. Calibrate these 4 points from top to bottom.



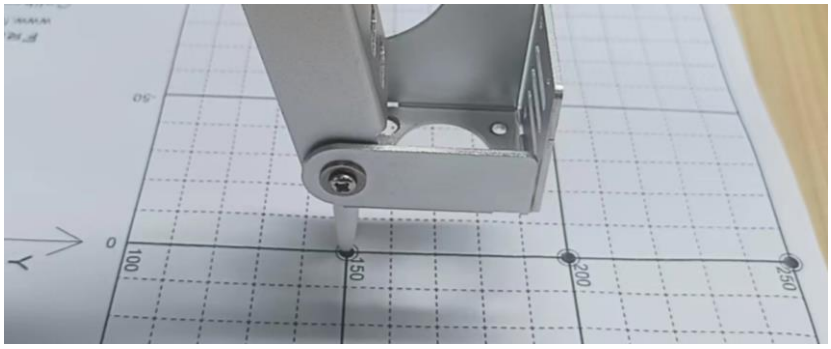
Point 1



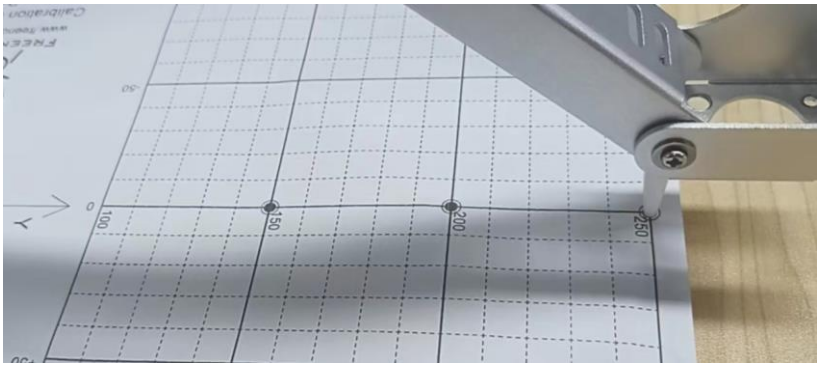
Point 2



Point 3



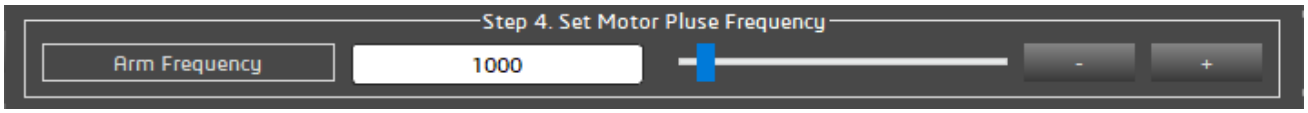
Point 4



At this point, the calibration of the robotic arm is completed. You can choose to close this interface and return to the main interface to control the robotic arm.

Motor Pulse Frequency Setting

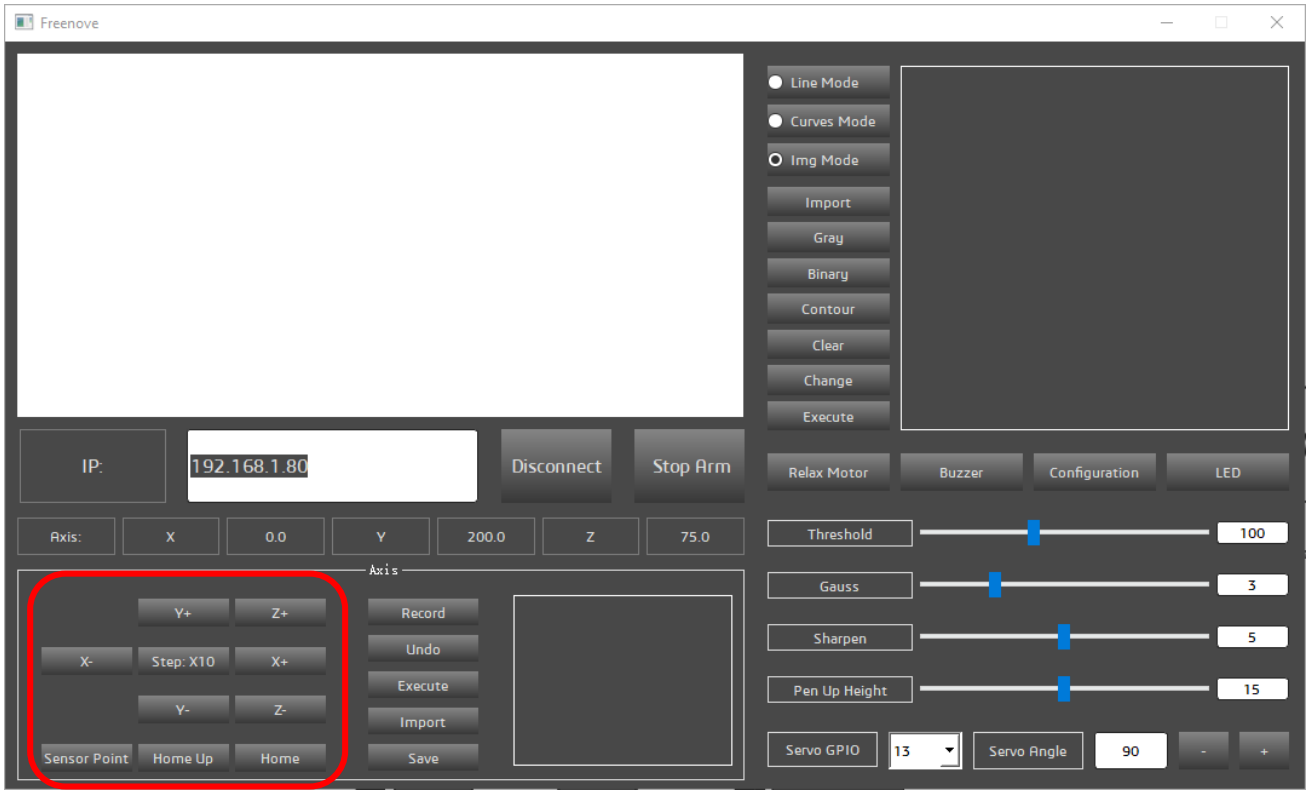
As previously mentioned, the rotation speed of stepper motors is determined by the frequency of the pulse signals. Higher frequencies result in faster motor speeds, while lower frequencies lead to slower speeds. By default, the pulse frequency for stepper motors is set at 1000 Hz (pulses per second). Generally, it is not recommended to change this default setting.



Robot Arm Control

Robot Arm Movements

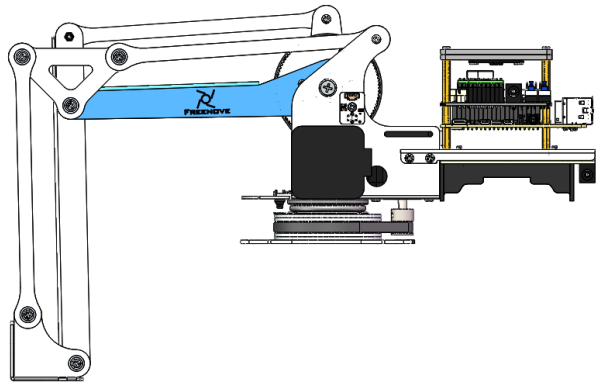
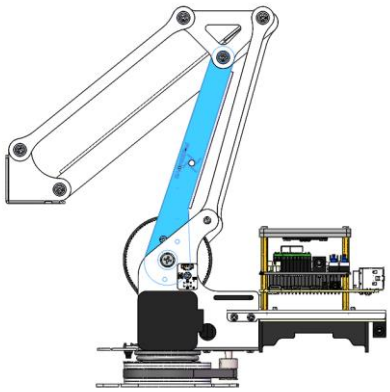
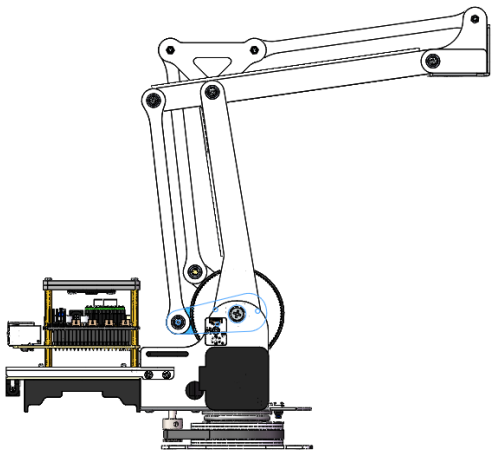
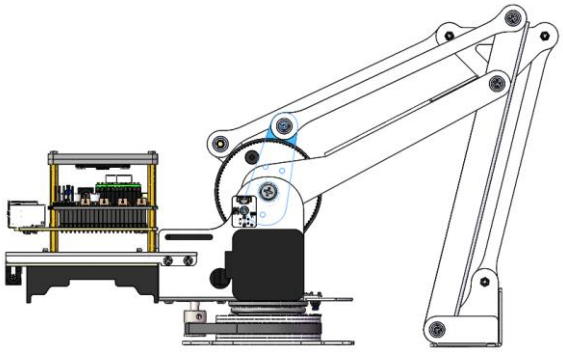
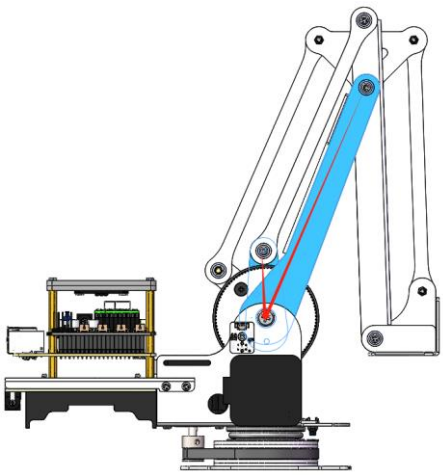
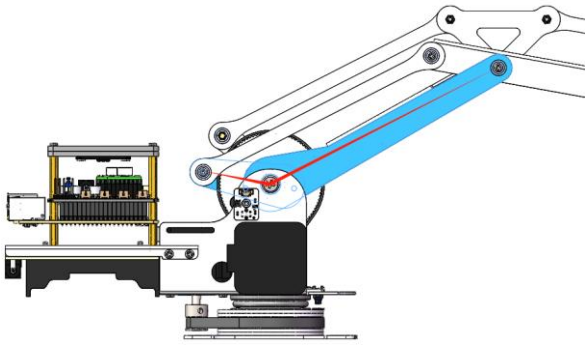
You can move the robot arm by clicking the buttons marked below.



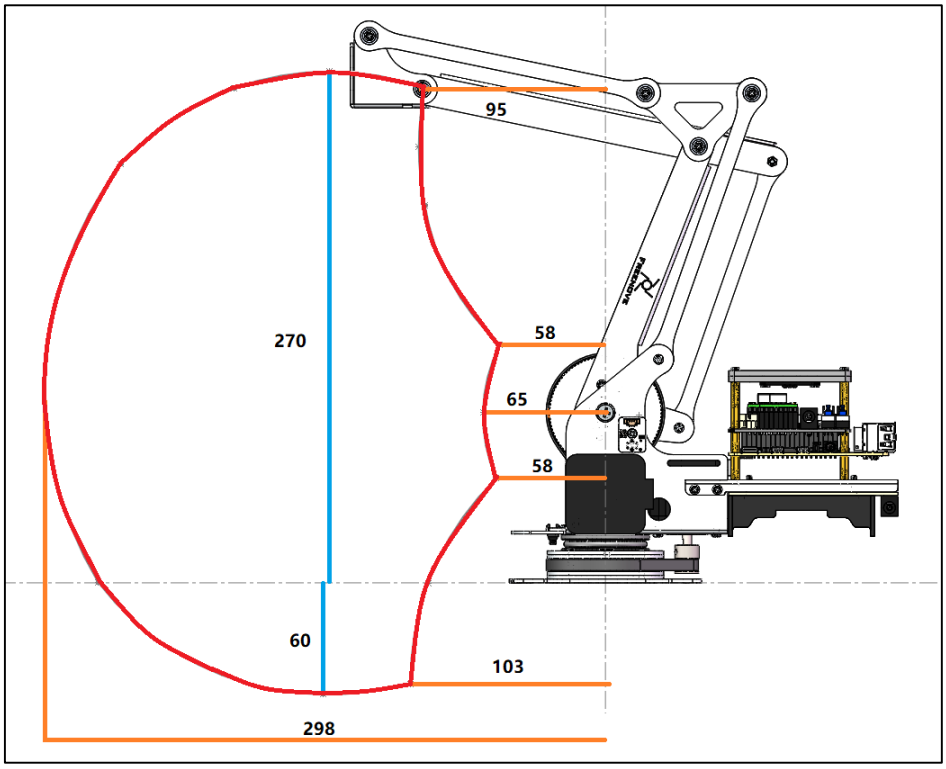
Sensor Point	After every time you click "Load Motor," or when the stepper motor experiences a missed step action, please click the designated button. This button will return the robotic arm to the middle position of the sensor, effectively resetting the robotic arm's positional coordinate information.
Step: X10	This feature is designed to adjust the distance that the robotic arm moves with each action. By pressing the X, Y, or Z buttons, the robotic arm will move 10mm. When you click this button, it allows you to toggle between 1mm and 10mm as the step size for each movement.
Home	Control the robot arm to move to the Home point coordinates
Home Up	Control the robot arm to move above the Home point coordinates
X、Y、Z	Control the robotic arm to move up, down, left, right, and back.

Rotation range

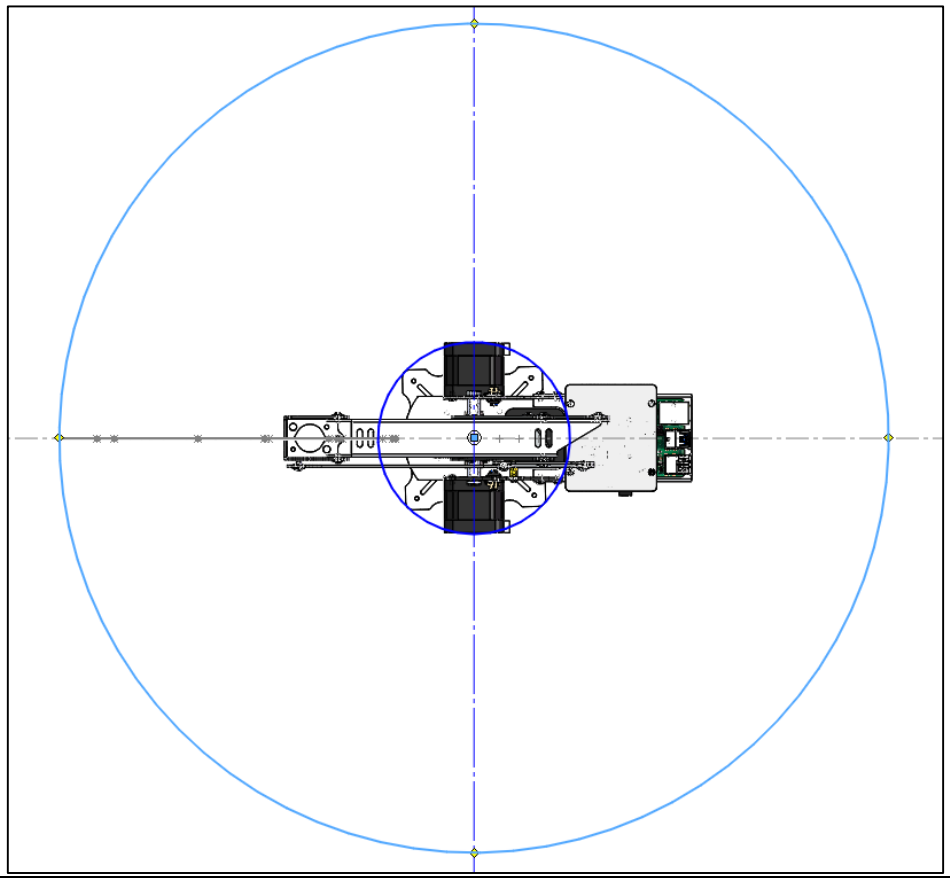
Due to the physical constraints of the structure of the robot arm, there are certain coordinates that the robot arm cannot reach. Therefore, we have implemented an angular limit range for the movement of the robotic arm, as shown in the figure below.

Rotation limit Angle of the left-motor 1: 0°	Rotation limit Angle of the left-motor 2: 110°
	
Rotation limit Angle of the right-motor 1: -12°	Rotation limit Angle of the right-motor 2: 98°
	
Minimum Angle between the two sides: 26°	Maximum Angle between the two sides: 150°
	

The motion range of the robot arm is shown in the figure below.

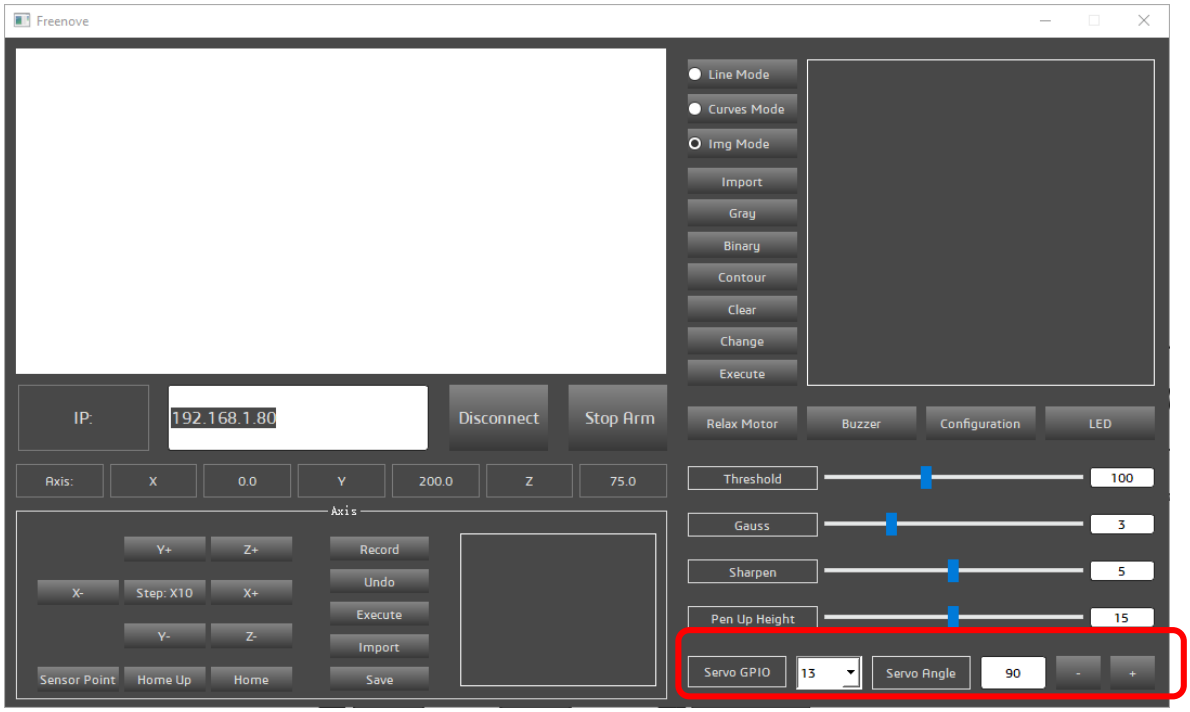


The motion range of the robot arm is shown in the figure below.

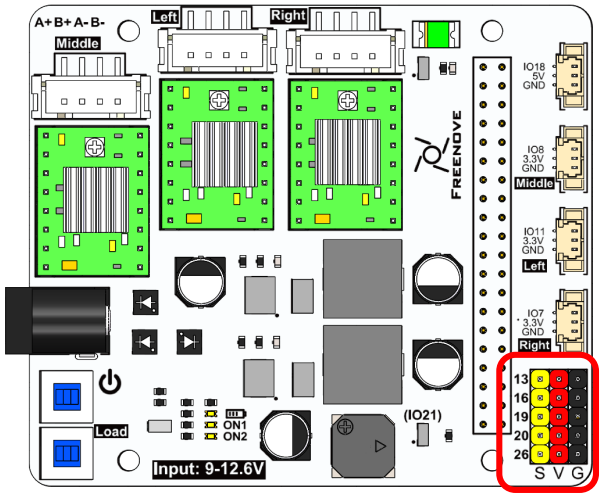


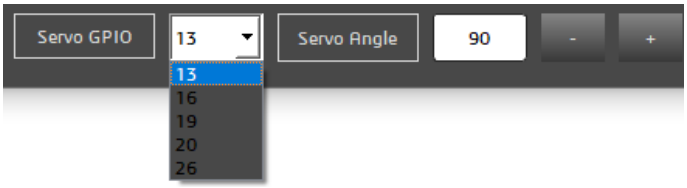
Robot Clamp Control

Before using the software to control the robot clamp, please ensure that the clamp is properly installed.
As shown in the figure below, you can use the buttons depicted in the diagram to control the clamp of the robotic arm.



Before you begin controlling the servo with the software, it's important to verify which interface it is connected to. For instance, if you have connected the servo to GPIO pin 13, you should select 13 in the software. Similarly, if your servo is connected to GPIO pin 16, choose 16 in the software.

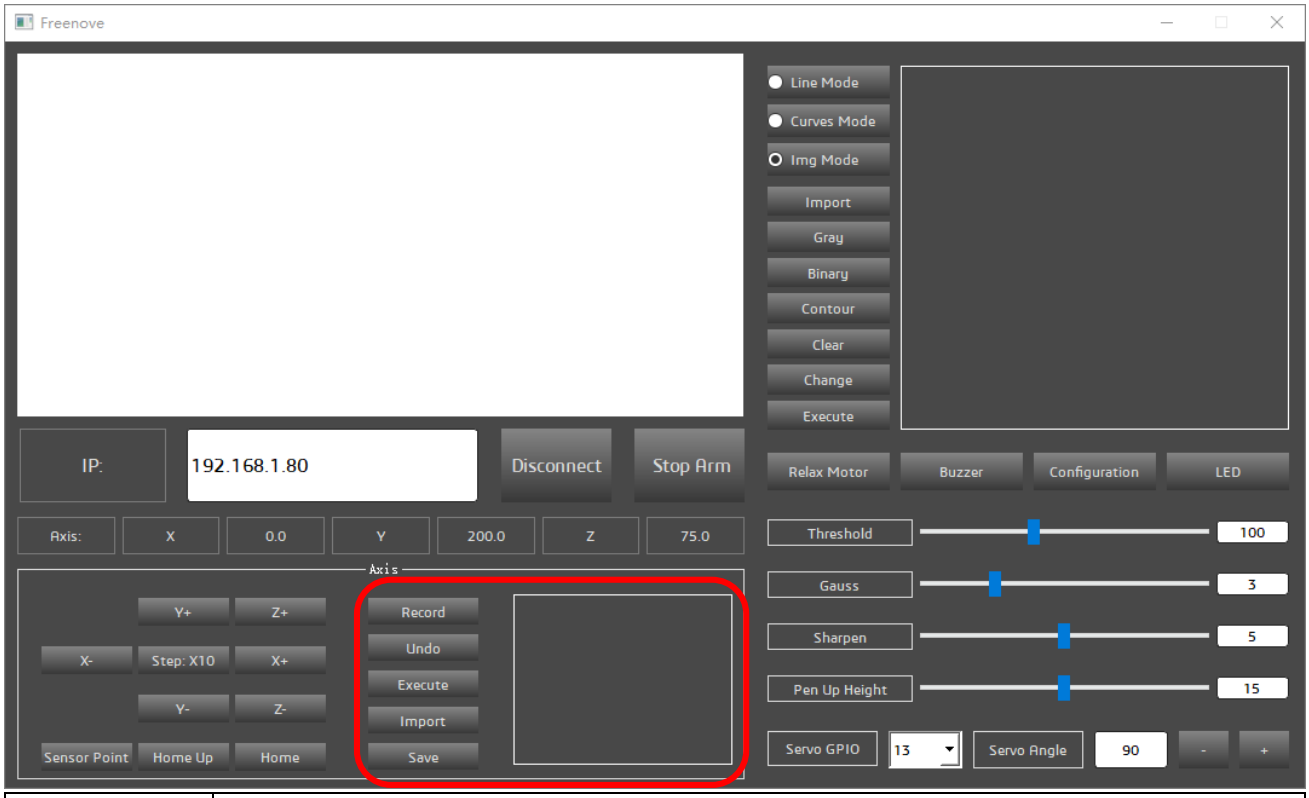




When operating the clamp via the software, the functionality of the control buttons is straightforward:
Clicking the "+" button will cause the clamp to open, expanding its jaws to release any held objects.
Clicking the "-" button will make the clamp close, clamping down on whatever is held between its jaws.
It is important to be mindful of the angle at which you use the clamp to pick up items. For instance, if the clamp has rotated to 30 degrees and the item is securely grasped, it is not advisable to continue tightening the clamp. Doing so could result in the servo exerting more power than necessary to hold the item, which could potentially damage both the servo and the object being gripped.

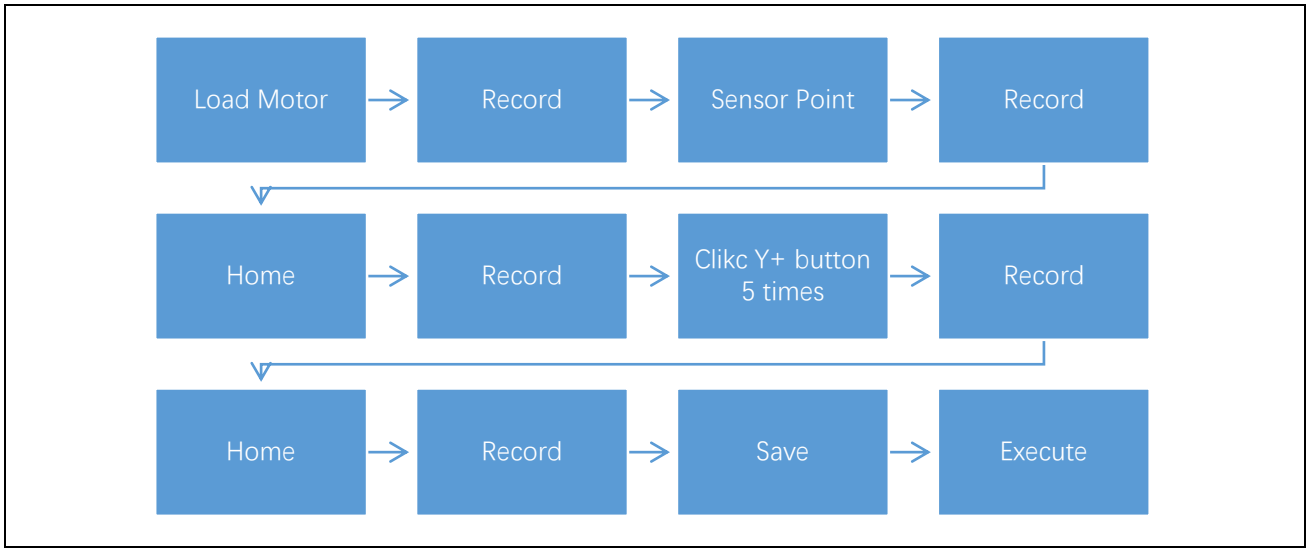
Instructions Recording Mode

To enhance the interactivity and fun of using the robotic arm, we've incorporated an instruction recording mode within the software.



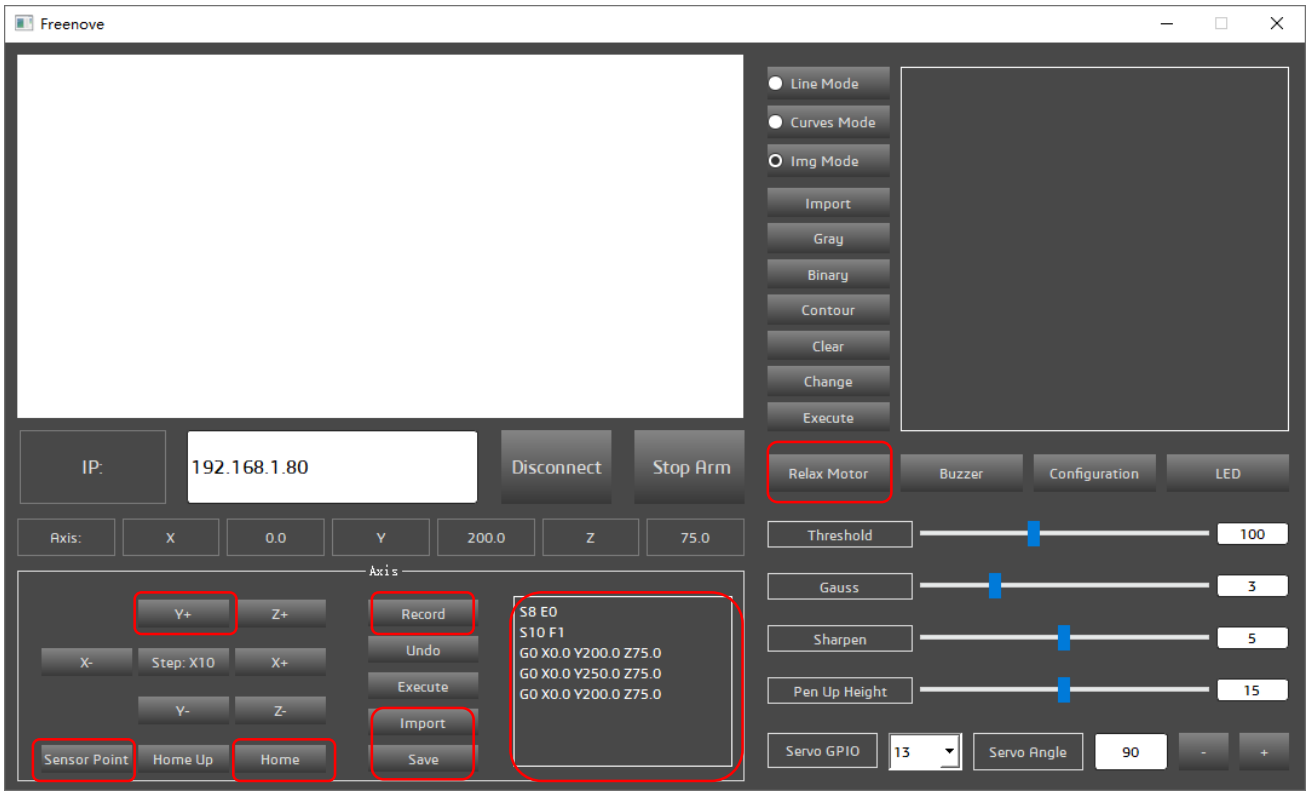
Record	Each time Record is clicked, the software will record the last executed operation instruction and display it in the display area on the right.
Undo	Each time Undo is clicked, the software will cancel the last command recorded in the display area on the right
Execute	Each time Execute is clicked, the robot arm will execute the contents of the right display area from top to bottom.
Import	Each time Import is clicked, the user selects a local file, and the software loads the instructions in the local file into the display area on the right.
Save	Each time Save is clicked, the software writes the contents of the display area on the right to a local file.

For example, to program the robotic arm to perform sensor calibration, move to the Home Point, then to Point 4, and finally return to the Home Point, you would follow these steps:

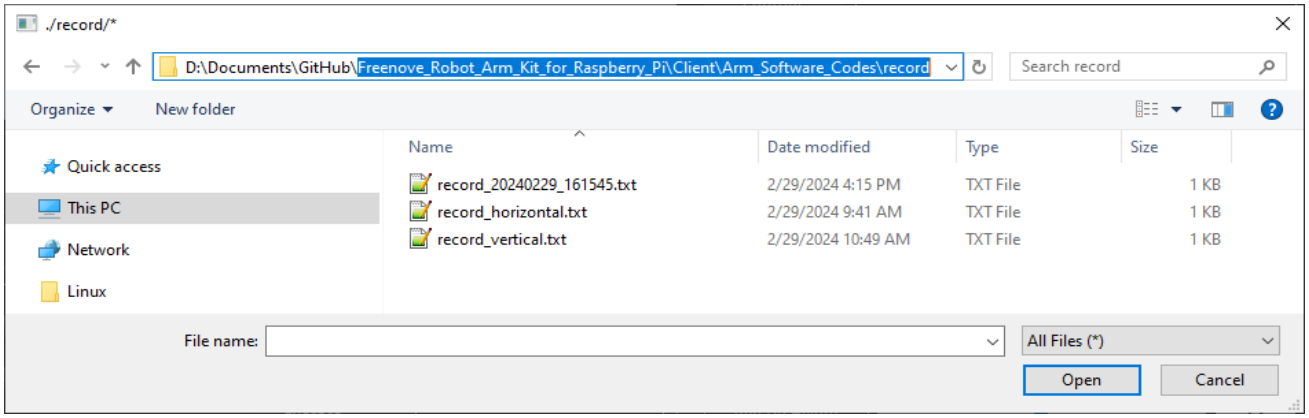


Each time Save is clicked, the content in the display area on the right will be saved in the form of date + time under the directory

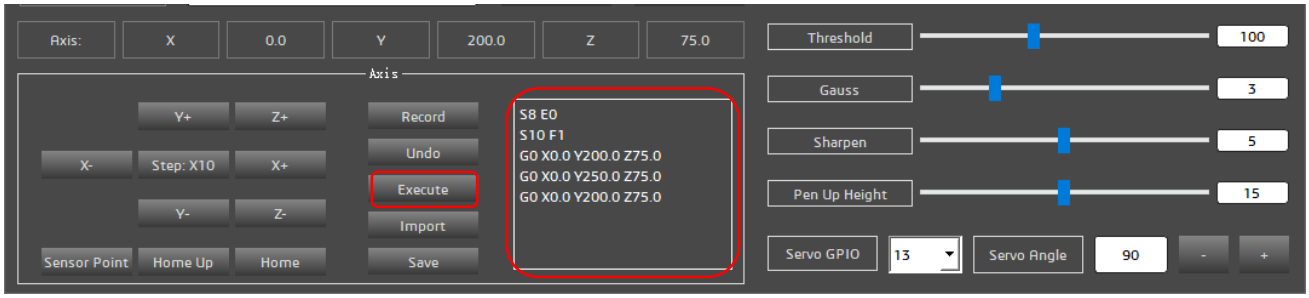
Freenove_Robot_Arm_Kit_for_Raspberry_Pi\Client\Arm_Software_Codes\record



Similarly, each time Import is clicked, please upload the txt files under the directory **Freenove_Robot_Arm_Kit_for_Raspberry_Pi\Client\Arm_Software_Codes\record**



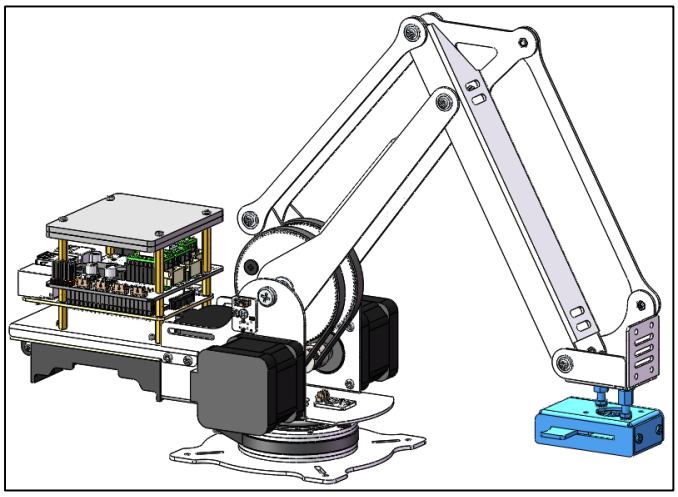
Each time Execute is clicked, the content displayed on the right is sent to the robotic arm to be executed. Please note that if the robot arm has not returned to the sensor center since it was started, please click Sensor Point before clicking Execute. If the robotic arm has not been centered and you attempt to execute commands without centering, the arm will use its buzzer to alert you. The number of beeps corresponds to the number of commands in the display area that are queued for execution.



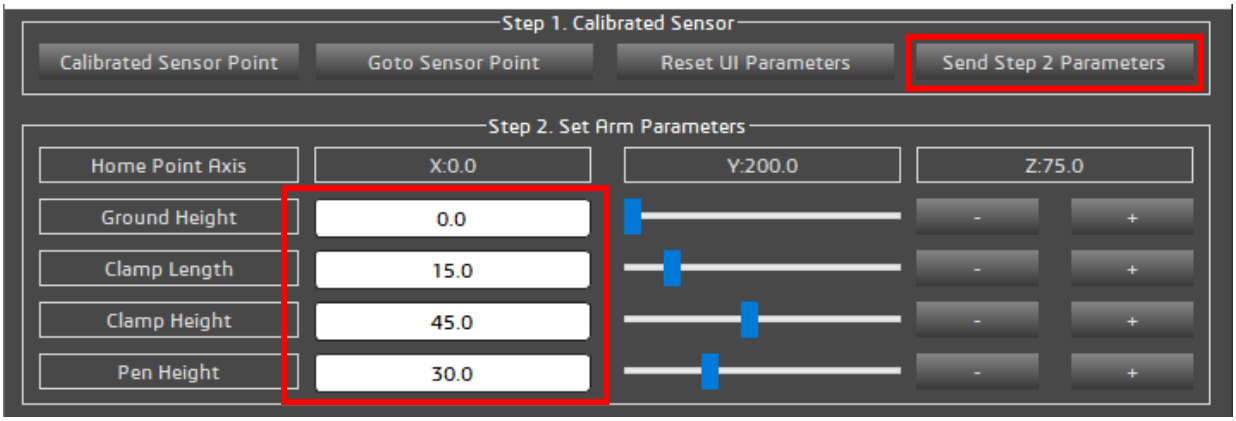
Drawing Mode

To make the experience with the robotic arm more engaging and entertaining, we utilize it for an attempt at drawing.

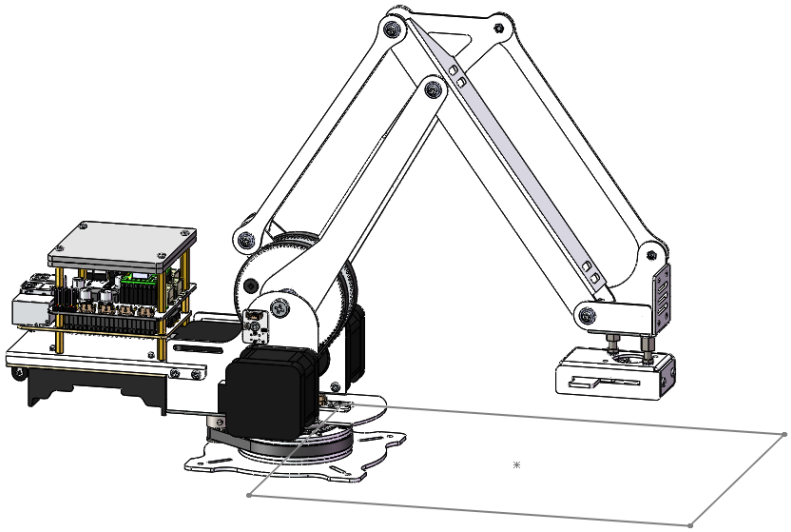
Firstly, ensure that the robotic arm is laid flat on a table surface and that the pen clip is properly installed.



Open the Configuration interface of the software. Configure the settings according to the provided configuration information, and then click on "Send Step 2 Parameters."

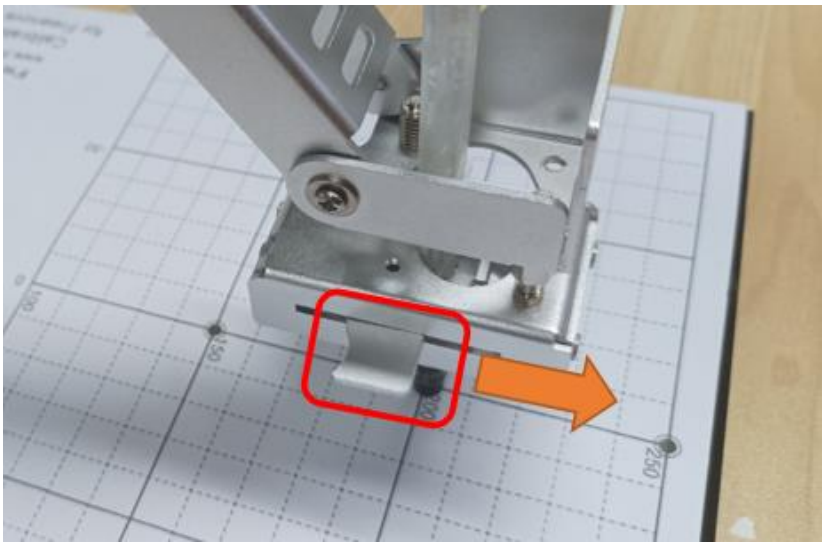


You can see that the pen clip stays 30.0mm above the calibration paper.

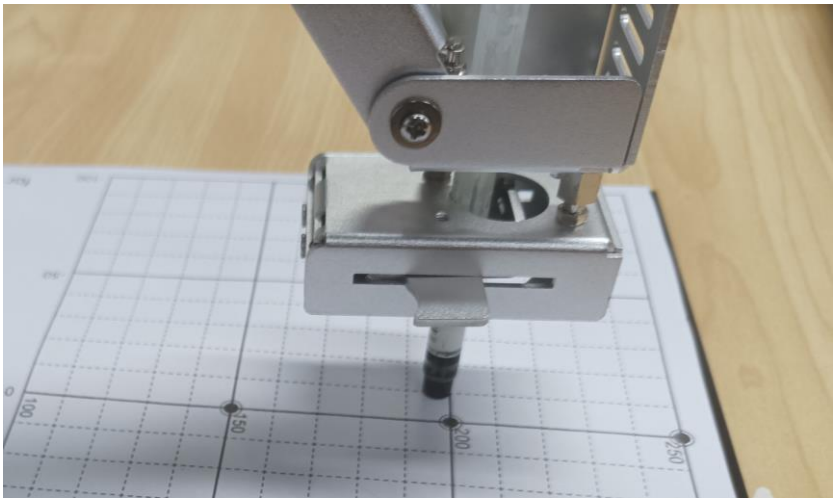


Manipulate the spring-loaded clip of the pen holder to insert the pen into the clip. Please ensure that the pen is positioned to naturally stand vertically against the paper, just making contact with it.

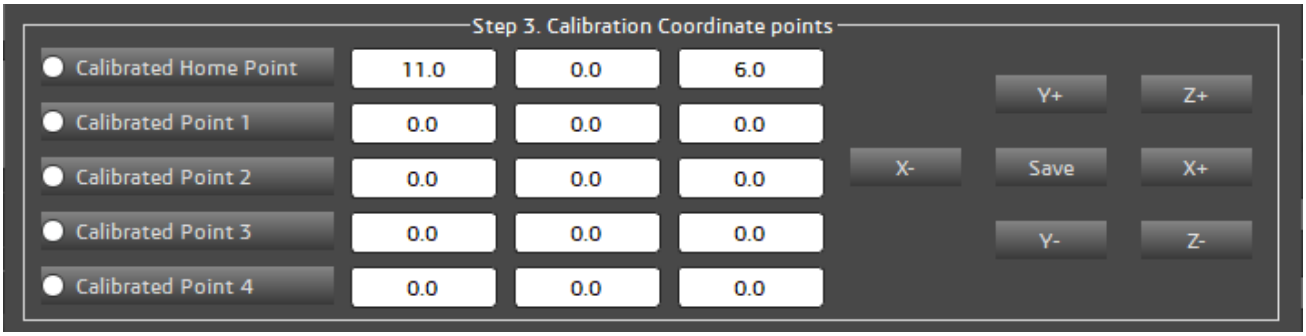
In actual use, please use the pen tip to calibrate, the result is more accurate.



If the pen clip at the end of your robotic arm is not at the Home Point position, as shown in the figure below,

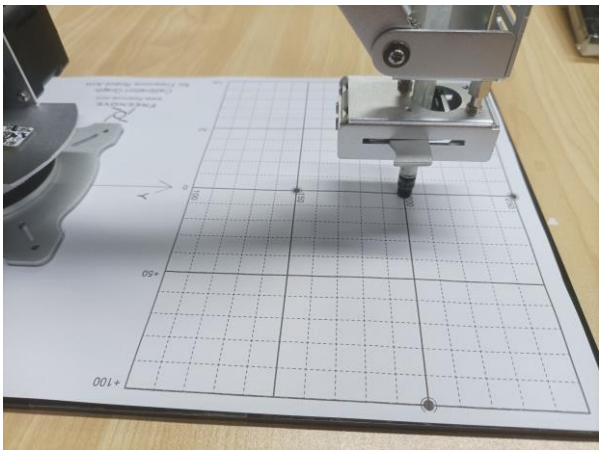


then you need to readjust Home Point and Points 1-4 so that the end of the pen clip of the robotic arm can run parallel to the paper.

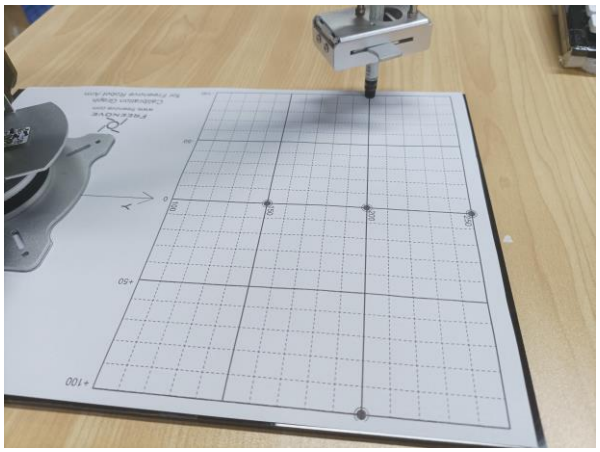


The actual configuration is as shown in the figure below.

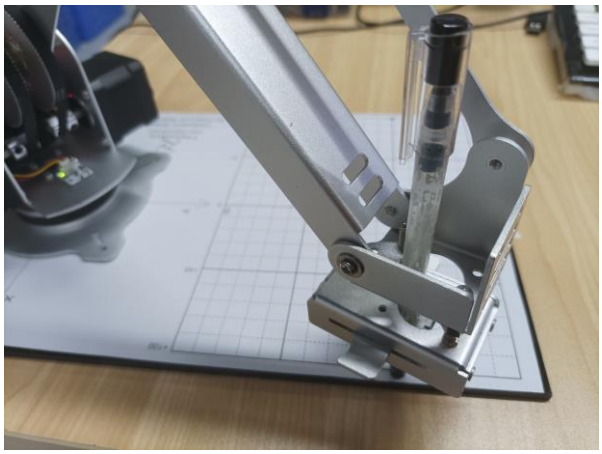
Home Point



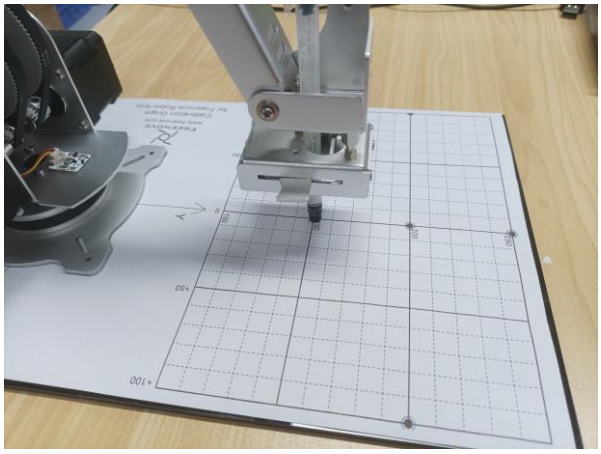
Point 1



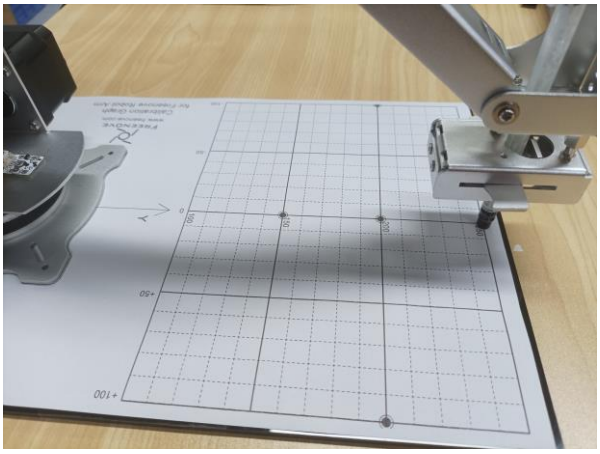
Point 2



Point 3

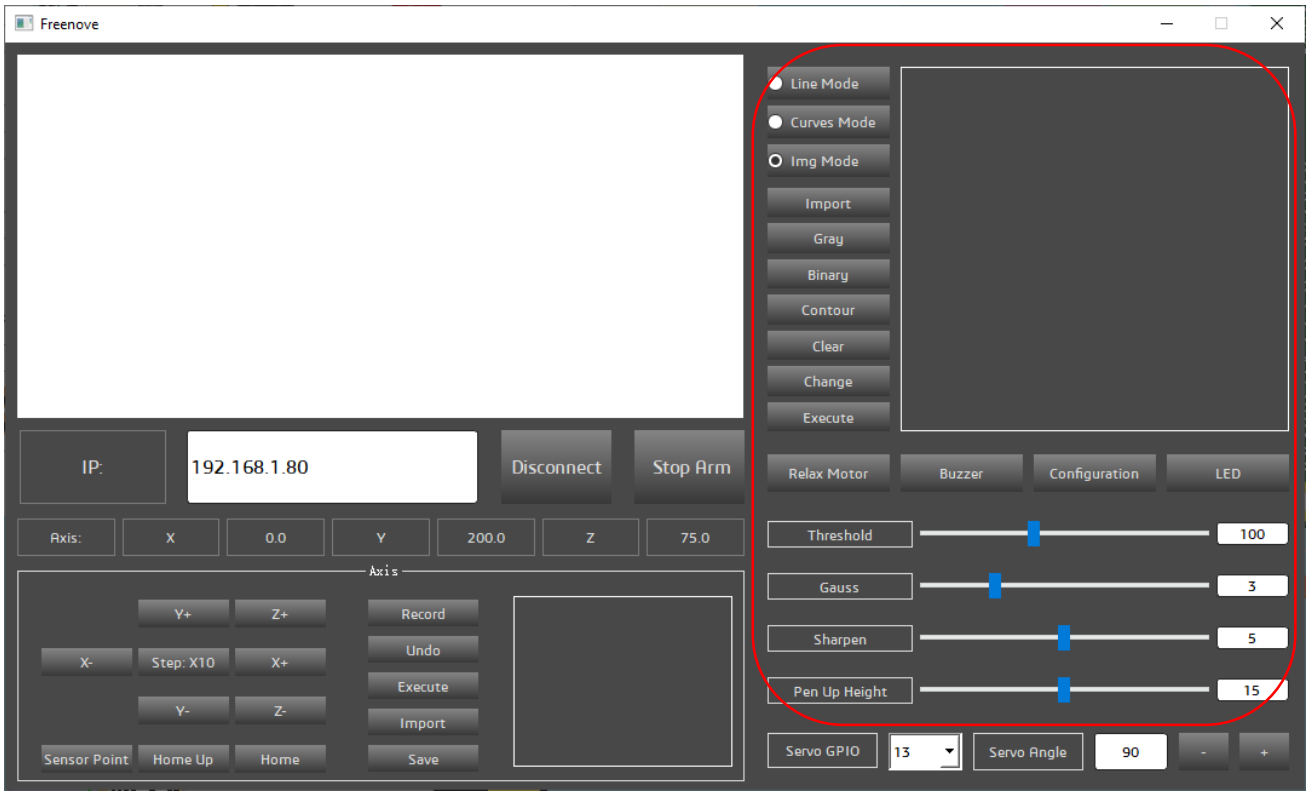


Point 4



After the recalibration is complete, close the Configuration screen and return to the main interface.

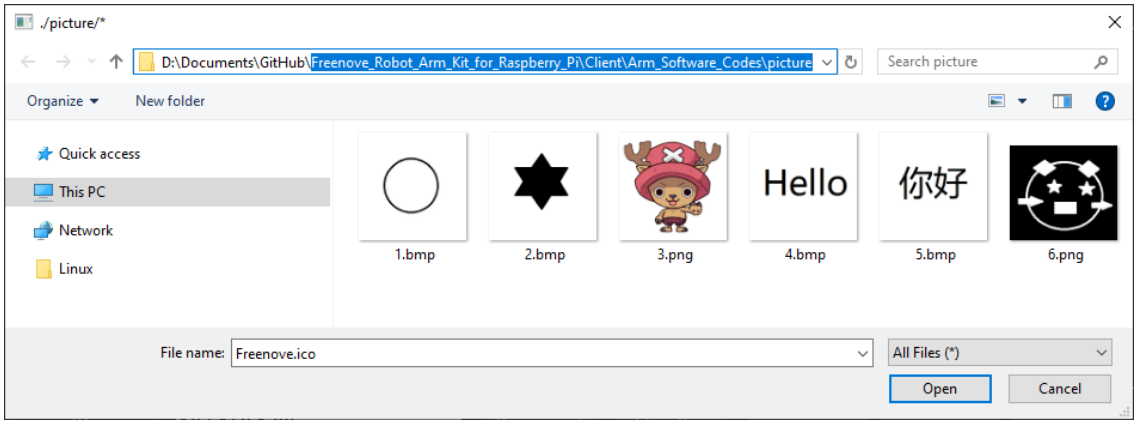
The buttons of the drawing function is as illustrated below.



We can open the local folder by clicking the Import button and select the appropriate picture to load into the software. Usually we put the picture undet the directory

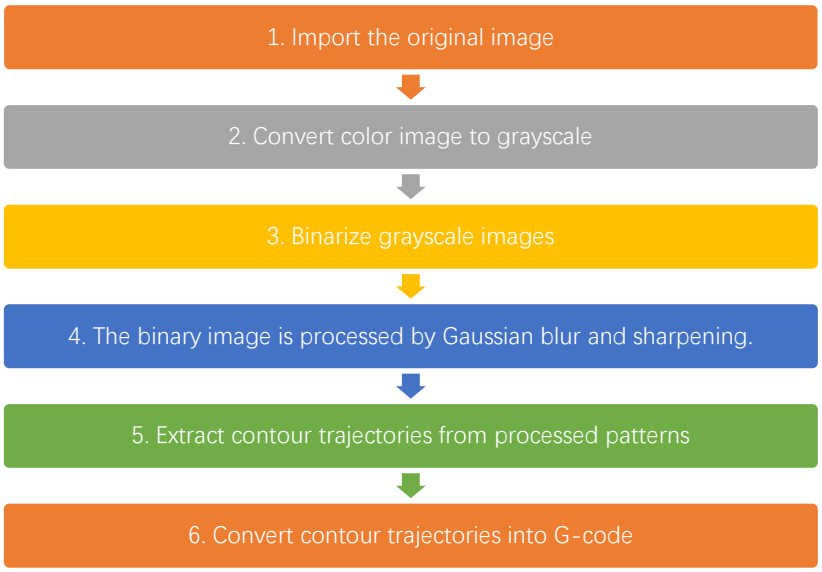
Freenove_Robot_Arm_Kit_for_Raspberry_Pi\Client\Arm_Software_Codes\picture

Select the file “4.bmp” and click Open.

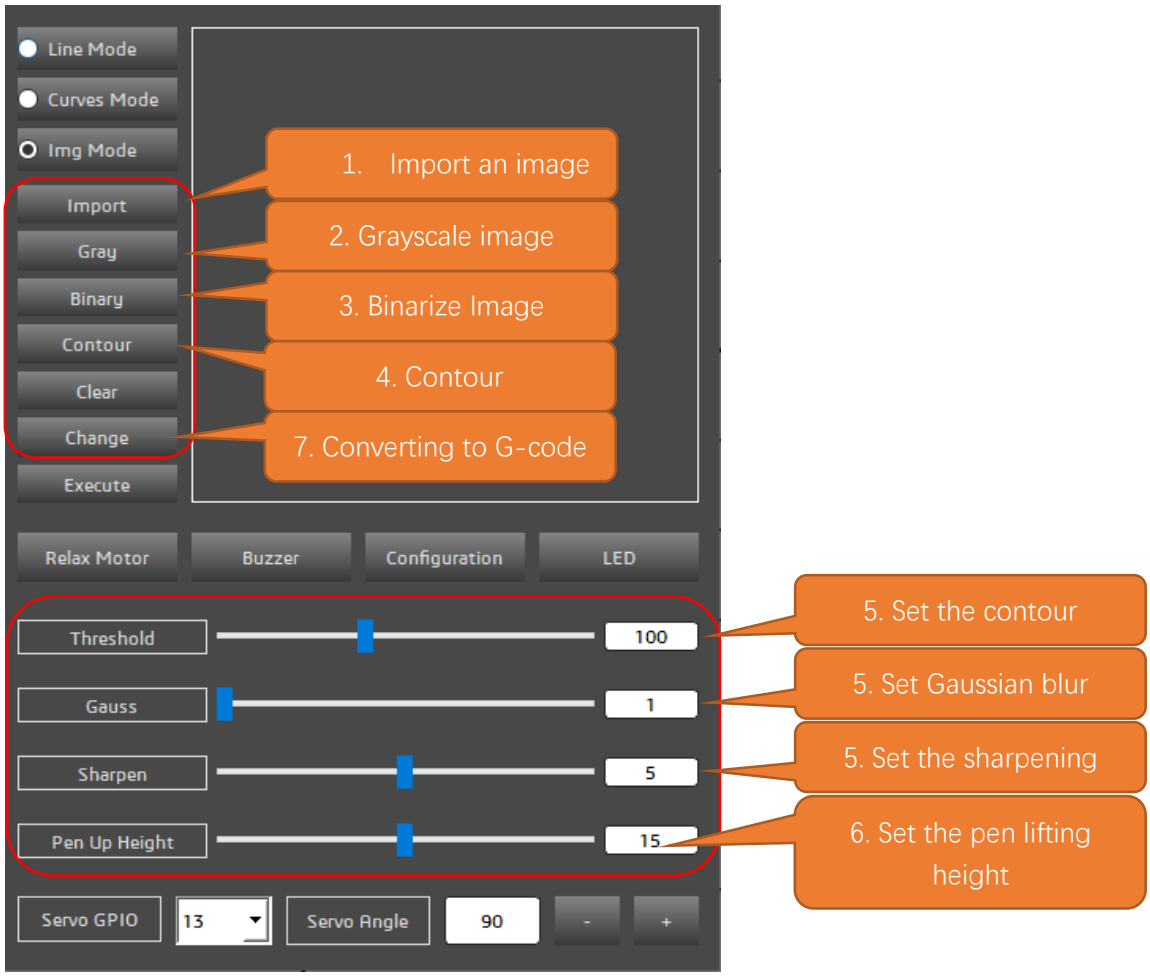


And you can see the picture is loaded to the software.

The original image cannot be used directly by the robotic arm. It requires a series of data processing and transformations to be converted into a format that the arm can utilize, as illustrated in the figure below.

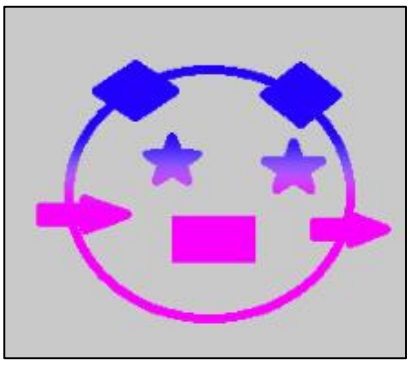


Please follow the prompts and perform the operations on the software.

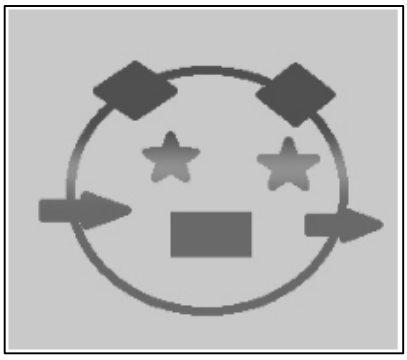


The effect of image processing is shown in the figure below.

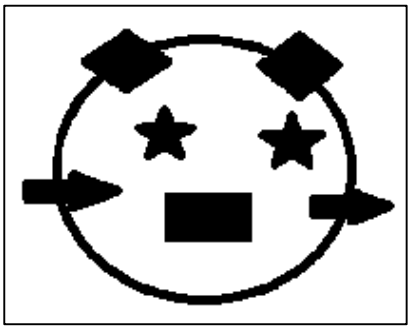
1. Import an image



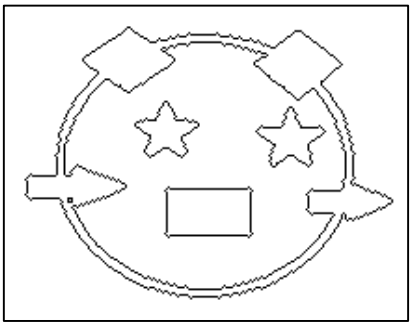
2. Convert the image to grayscale image



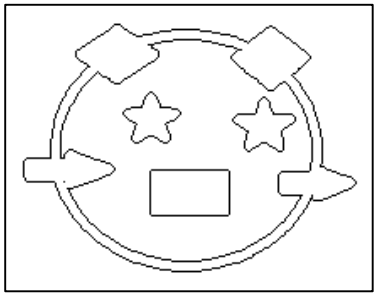
3. Binarize the grayscale image



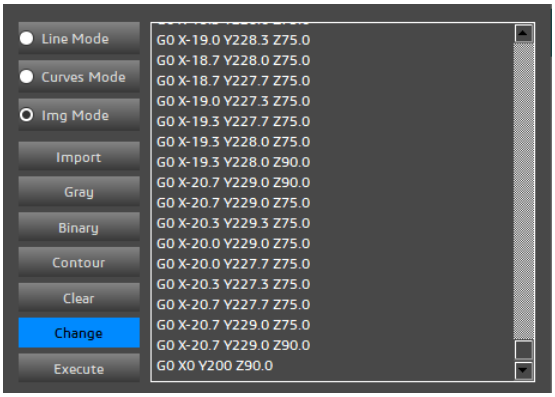
4. Convert the binary image to contour image



5. Adjust the contour threshold, Gaussian blur value, and sharpening value until the image effect is suitable.



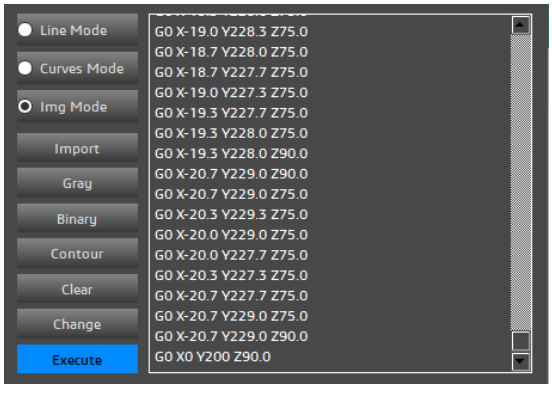
6. Click Change.



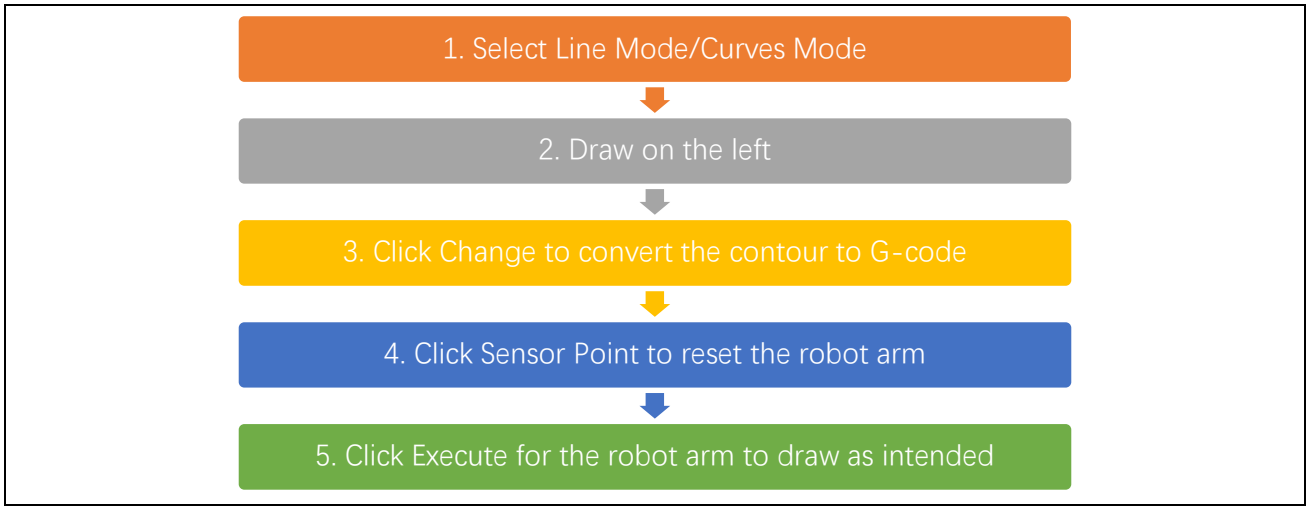
7. Click Sensor Point on the bottom left of the software.

This step is important!

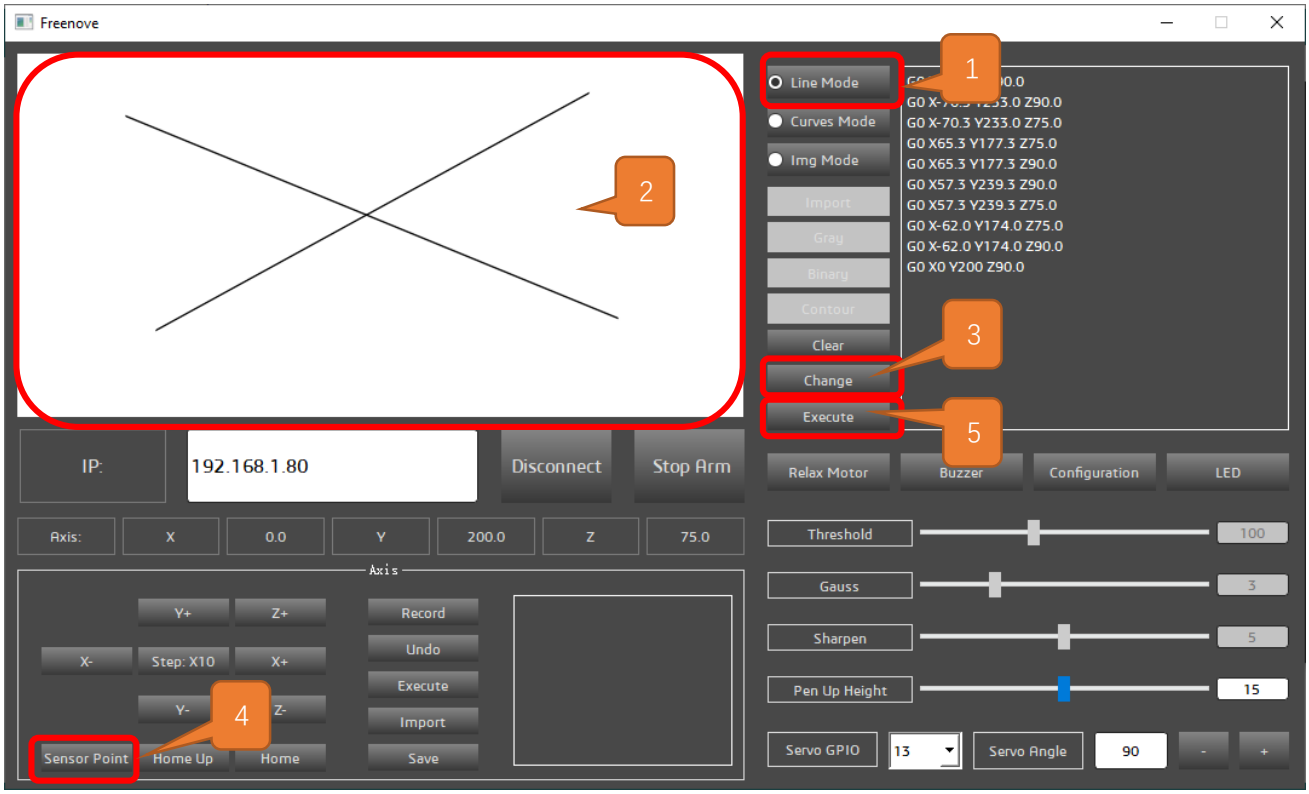
8. Clikc Execute



You can also choose other modes and draw freely in the left margin, as shown below.



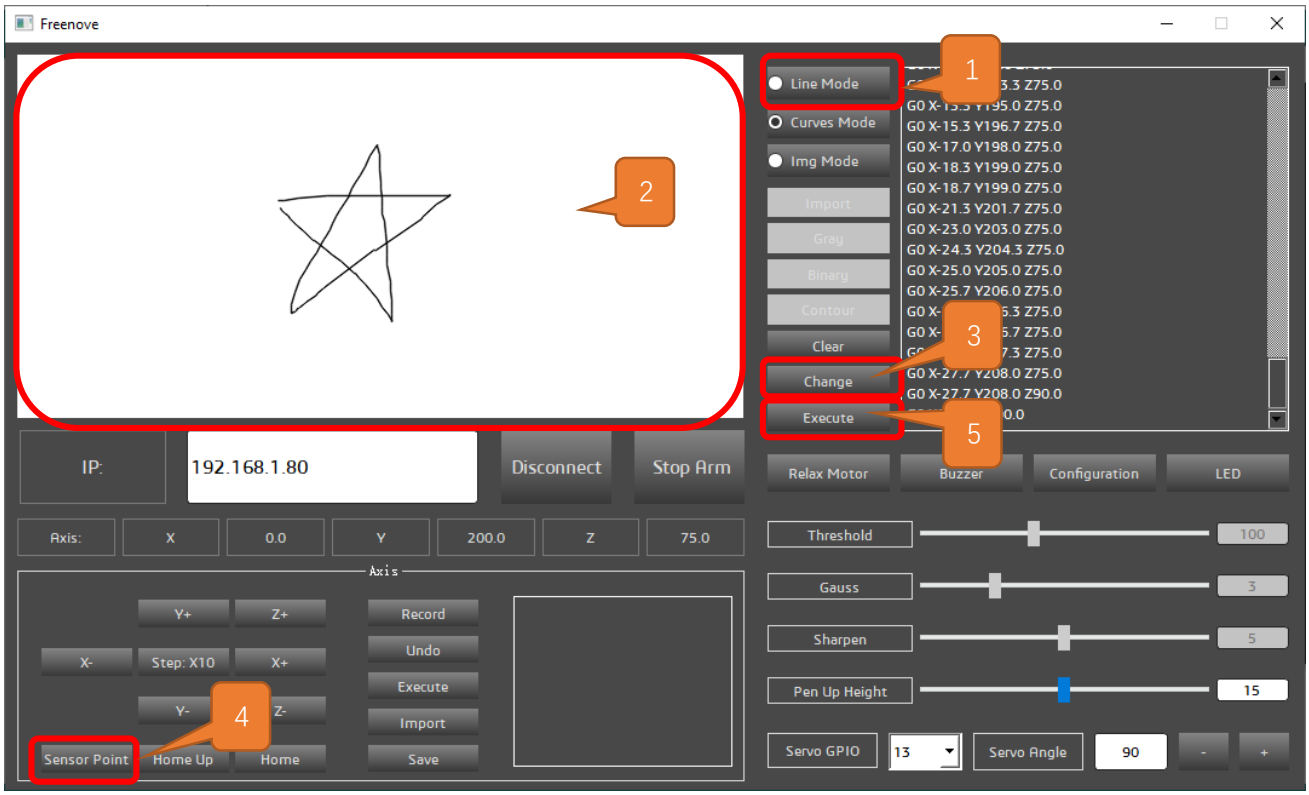
The steps for operating in the draw straight line mode are illustrated as shown in the figure below.



Please note that we have incorporated a buzzer alert mechanism for incorrect operations within the robotic arm.

Therefore, if your robotic arm has not undergone the Sensor Point operation after connecting to WiFi, and you omit Step 4 of the drawing function and directly proceed to Step 5, the program will not execute, but rather the buzzer will sound continuously as warnings.

Curve pattern



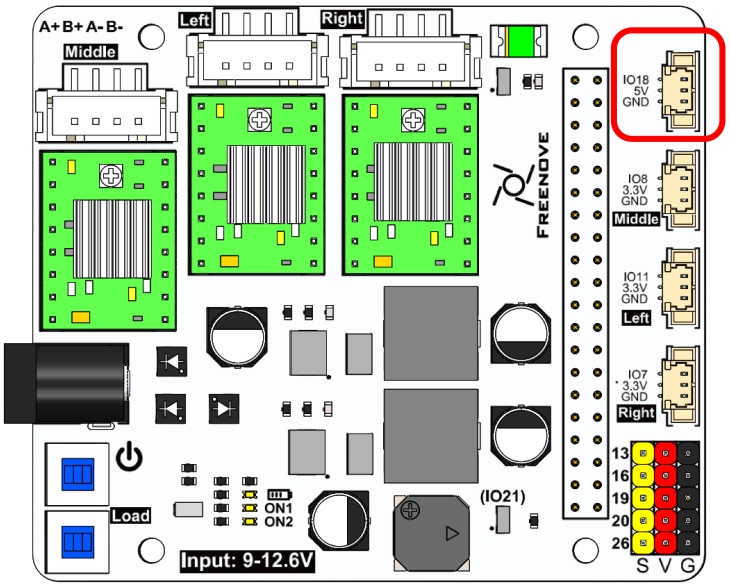
Please note that we have incorporated a buzzer alert mechanism for incorrect operations within the robotic arm.

Therefore, if your robotic arm has not undergone the Sensor Point operation after connecting to WiFi, and you omit Step 4 of the drawing function and directly proceed to Step 5, the program will not execute, but rather the buzzer will sound continuously as warnings.

LED Module Control

There is an LED module assembled on the robot arm, which can also be controlled via the software.
First and foremost, please ensure that your RGB LED module is properly connected to the Robot Arm Board.
Pay close attention to the wiring sequence on the LED module before installing it; incorrect wiring may result in the lights not illuminating.

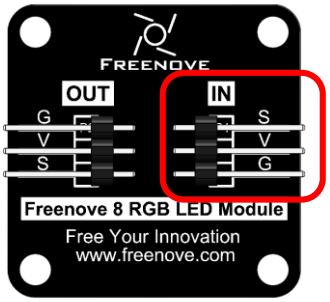
Connect the LED module to the robot arm board with a 10cm 3Pin cable to Jumper wire.



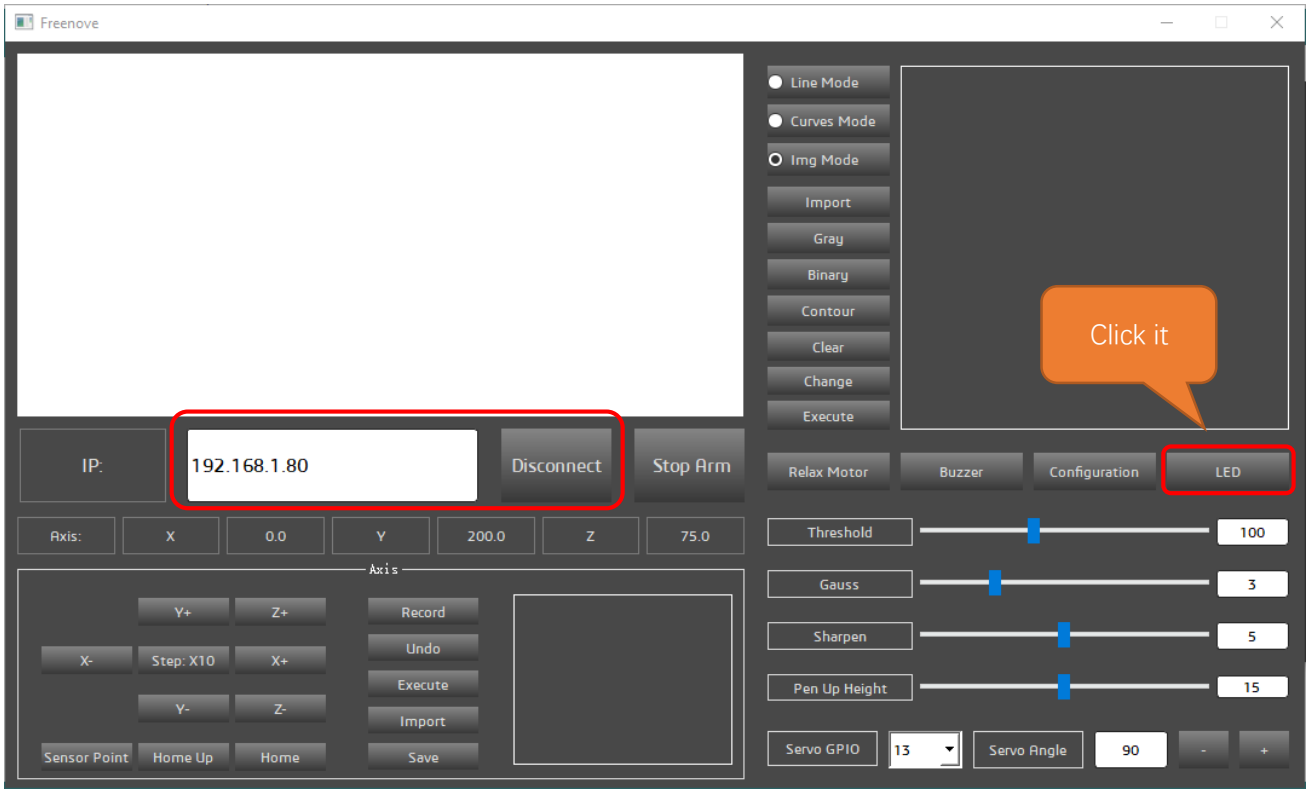
Wiring sequence

Board	LedPixel
IO18	S
5V	V
GND	G

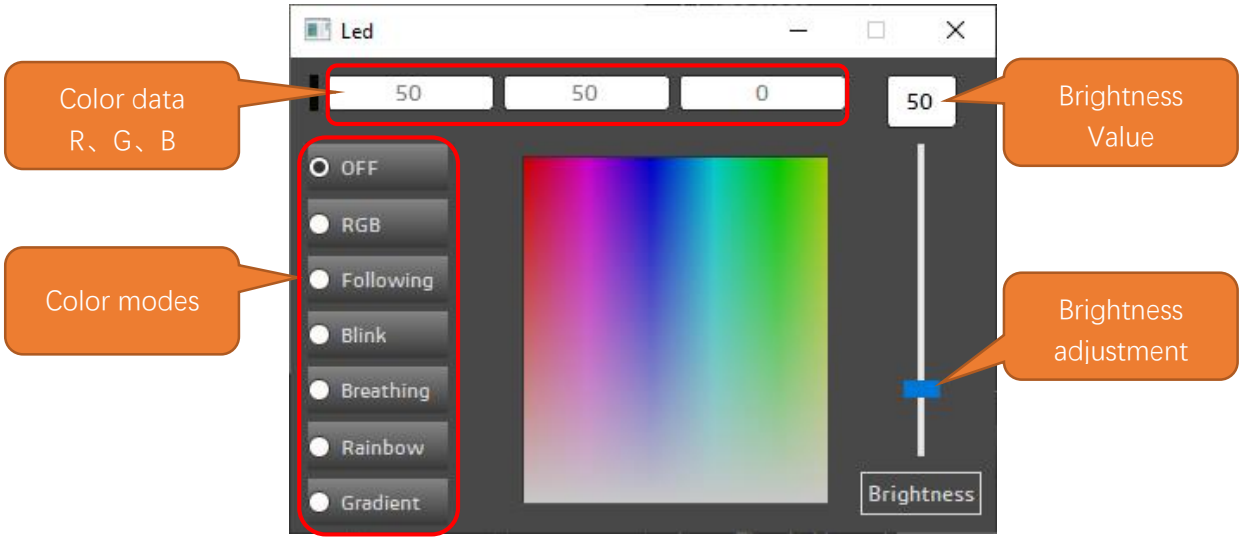
LedPixel



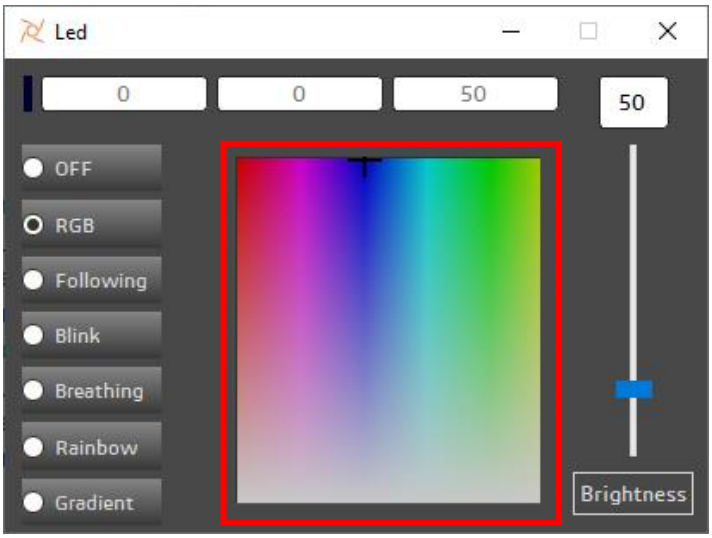
Open the software, connect it to Raspberry Pi and click the LED button.



The interface is as shown below.

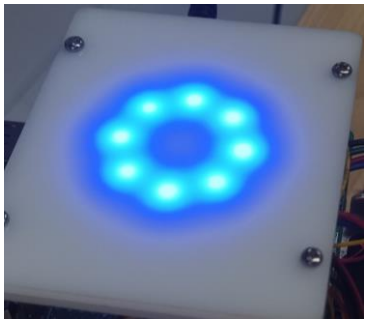


You can use your mouse to move across the color palette and select your preferred color. As you move the mouse, the color data displayed at the top of the interface will change according to your selection. Simultaneously, the light ring on the robotic arm will change in real-time to match the color you have chosen.

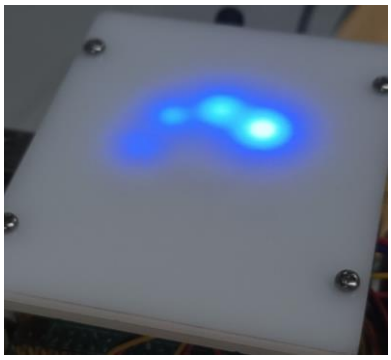


The LED module has six displaying modes. Its brightness can be adjusted by sliding the slider on the left.

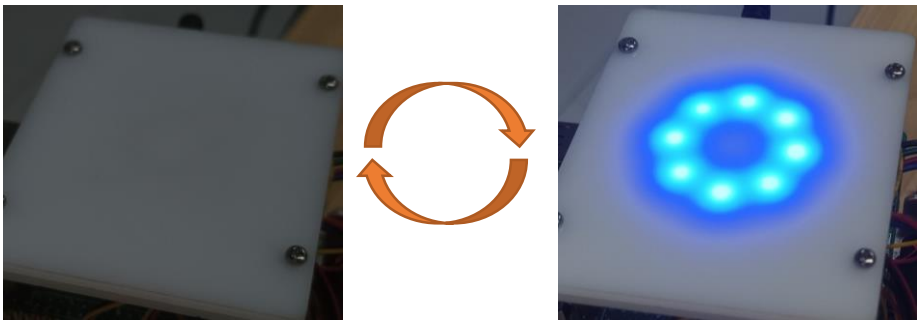
1. RGB Mode: In this mode, all the RGB lights will display a single color. You can change the color and brightness of the lights by using the color palette and sliders.



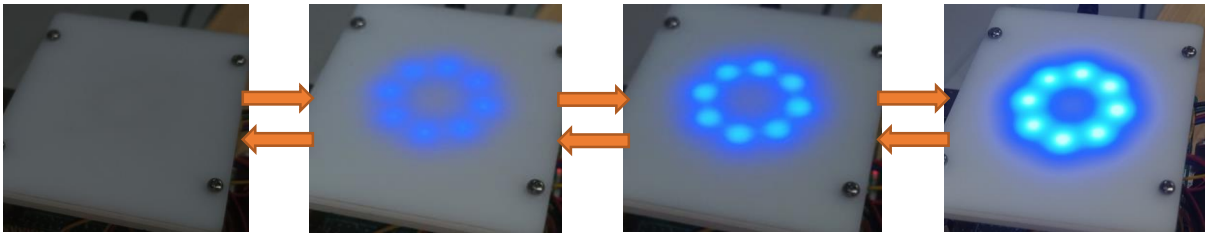
2. Following Mode: The RGB lights will continuously. You can use the color palette and sliders to change their color and brightness.



3. Blink Mode: All lights continuously blink. You can use the color palette and sliders to change their color and brightness.

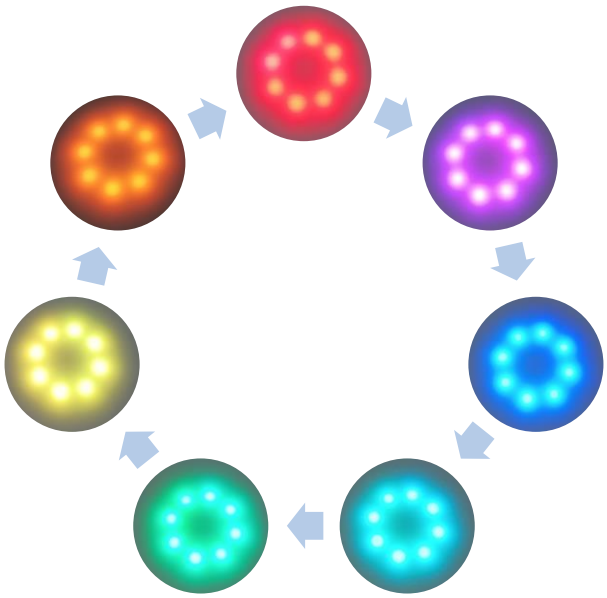


4. Breathing Mode: All lights gradually turn from dim to bright and then from bright to dim, like they are breathing. You can use the color palette and sliders to change their color and brightness.



5. Rainbow Mode: The whole LED module emits colors like rainbow and rotate slowly. In this mode, the color and brightness cannot be changed.

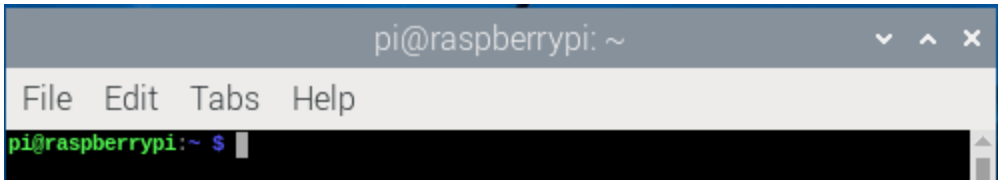
6. Gradient Mode: The whole LED module emits a single color and slowly change into other colors. In this mode, the color and brightness cannot be changed.



Chapter 5 Introduction to the APP

Connecting to the Robot Arm

Open Raspberry Pi Terminal.



Enter the folder **Freenove_Robot_Arm_Kit_for_Raspberry_Pi/Server**

```
cd Freenove_Robot_Arm_Kit_for_Raspberry_Pi/Server/Code
```

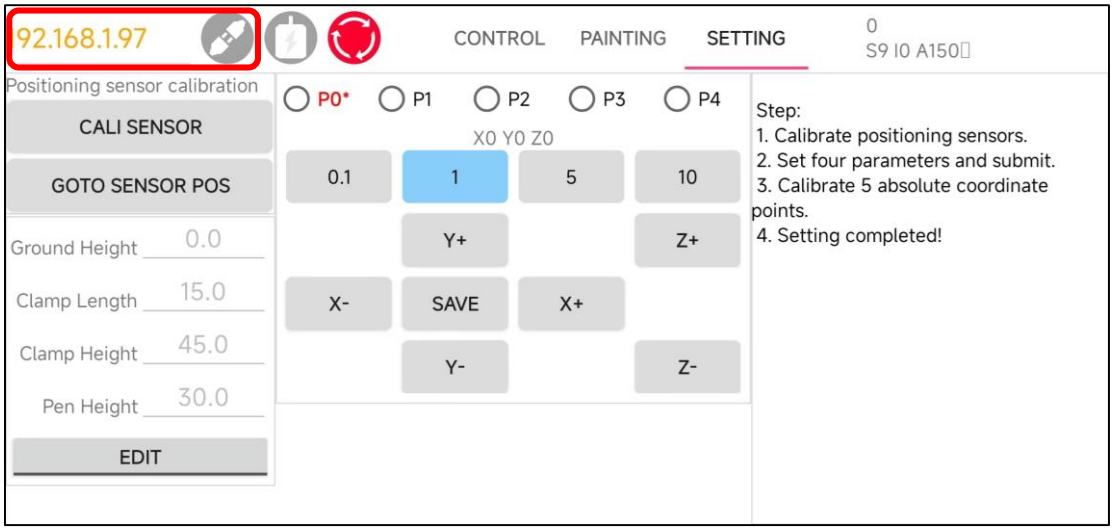
Run the server code.

```
sudo python main.py
```

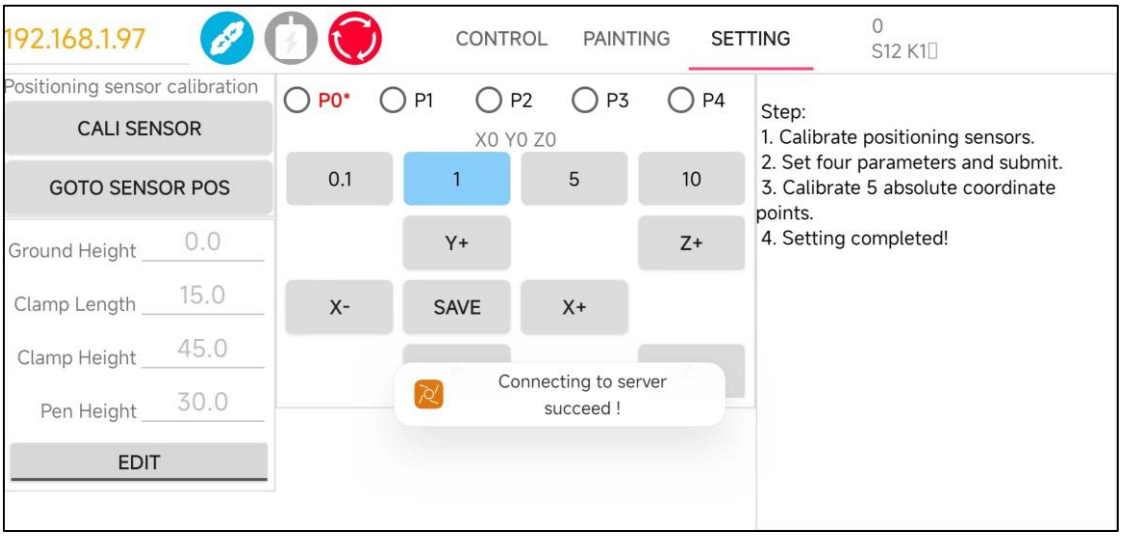
You can see the IP address of your Raspberry Pi printed.

Make sure your phone and the Raspberry Pi are connected to the same local network. Open the Freenove APP.

Enter the IP address printed on Raspberry Pi Terminal.

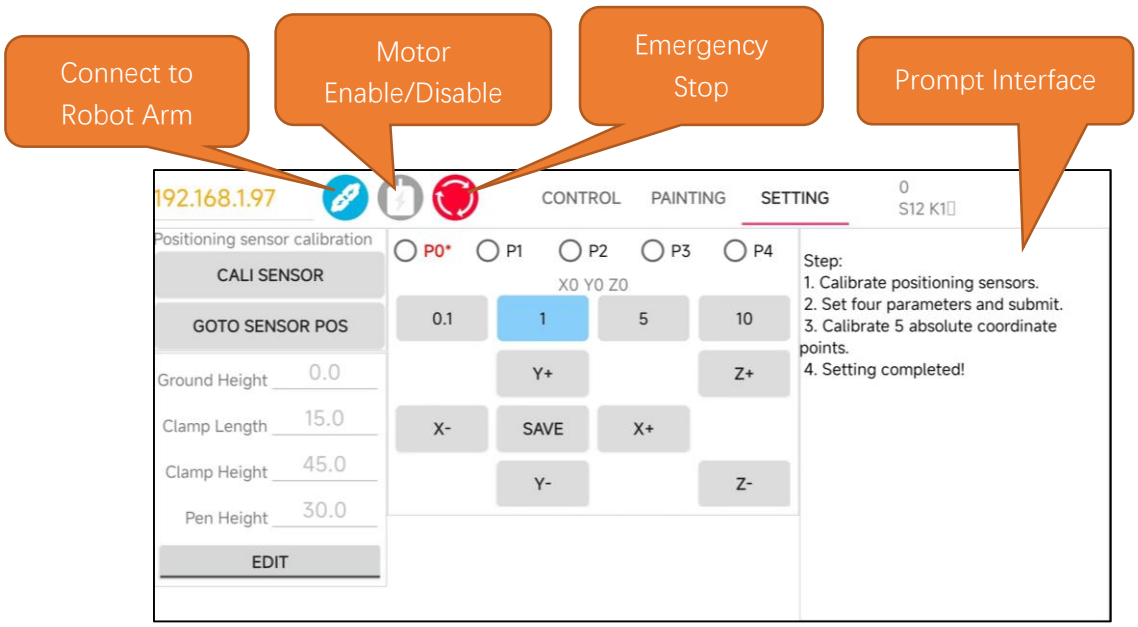


Click the Connection button and the app will establish communication with Raspberry Pi via WiFi.



Configuring Parameters for Robot Arm

Before using the robotic arm, please keep the following precautions in mind. They will help you use the mechanical arm more effectively:



Precautions for Use

1. **Initial Calibration**: The robotic arm does not contain calibration parameters when used for the first time. Therefore, a calibration configuration operation is necessary to ensure optimal performance. The calibration configuration information is recorded in a JSON file.

2. **Environment Changes**: If there is a change in the working environment of the robotic arm, please perform the calibration configuration again to adapt to the new environment.

3. **No Repeated Calibration**: If the working environment of the robotic arm does not change, the arm only requires a single calibration to function properly. There is no need to repeat the calibration configuration operation each time.

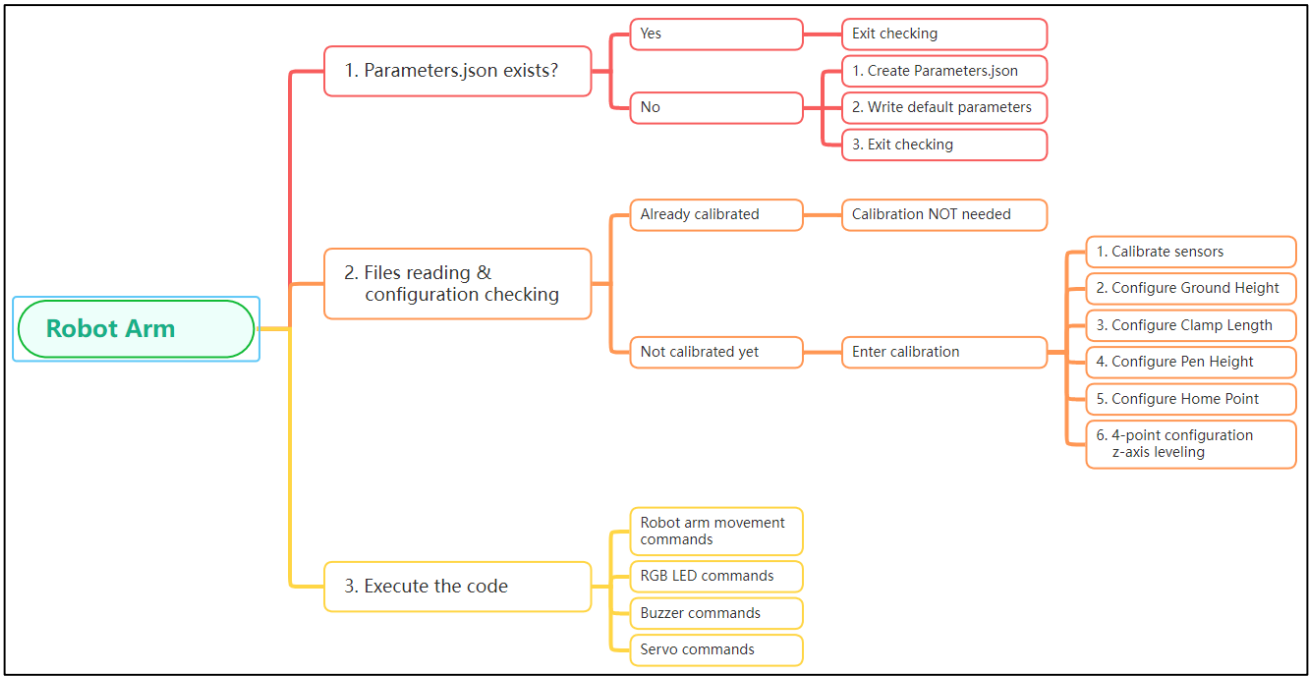
4. **Emergency Stop**: In case of abnormal operation of the robotic arm, you can force stop the arm using the "Stop Arm" button in the software. This will forcefully shut down all threads of the robotic arm and disable the motors, protecting both the motors and the circuit board. Alternatively, you can physically cut off the power supply to the motors by pressing the "Load" switch on the Raspberry Pi robotic arm's mainboard.

5. **Sensor Position Check**: Before using the software to control the robotic arm each time, ensure that the arm is set to the midpoint position detected by the sensor.

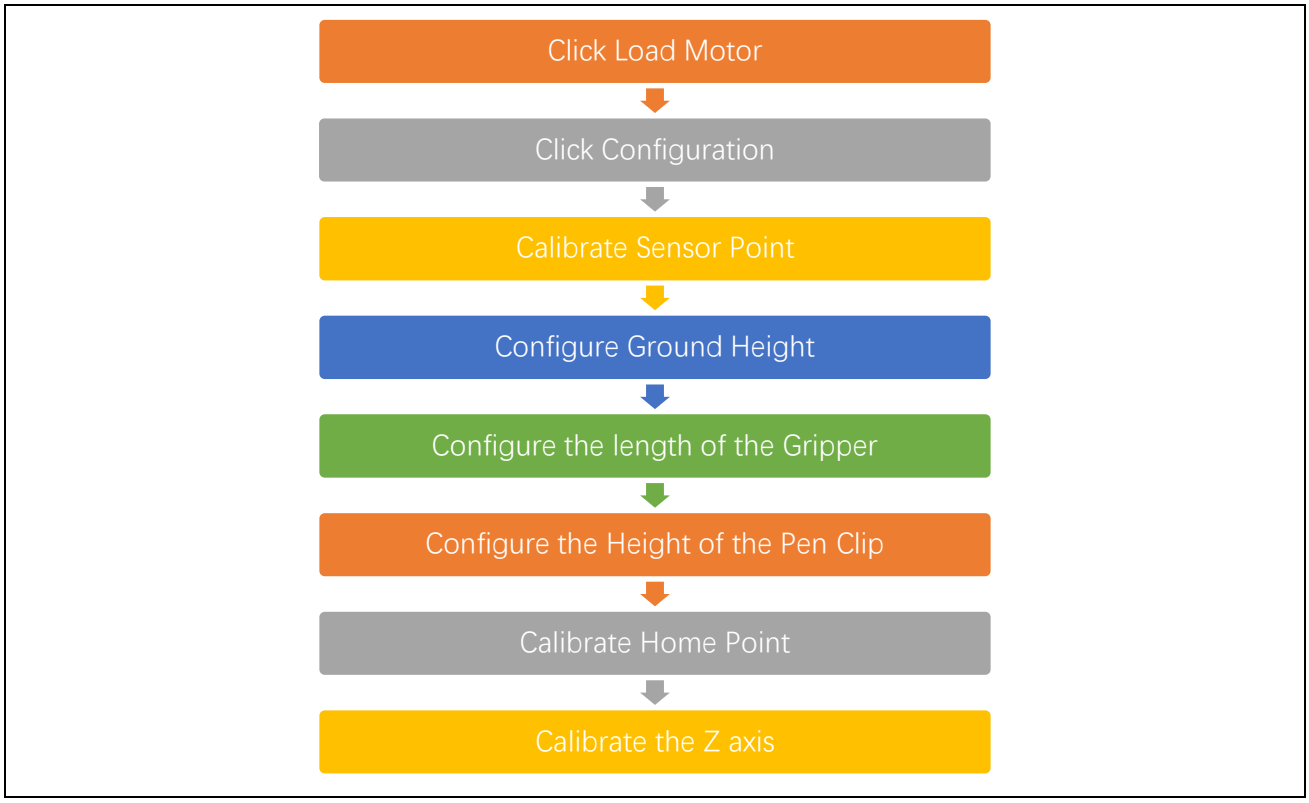
6. **Motor Activation for Calibration**: The calibration configuration operation requires the cooperation of the motors; thus, if the motors are not enabled first, it will not be possible to open the calibration configuration interface.

Robotic Arm Configuration Parameters

Each time the robotic arm is powered up, it undergoes a series of complex checks, as illustrated in the figure below.

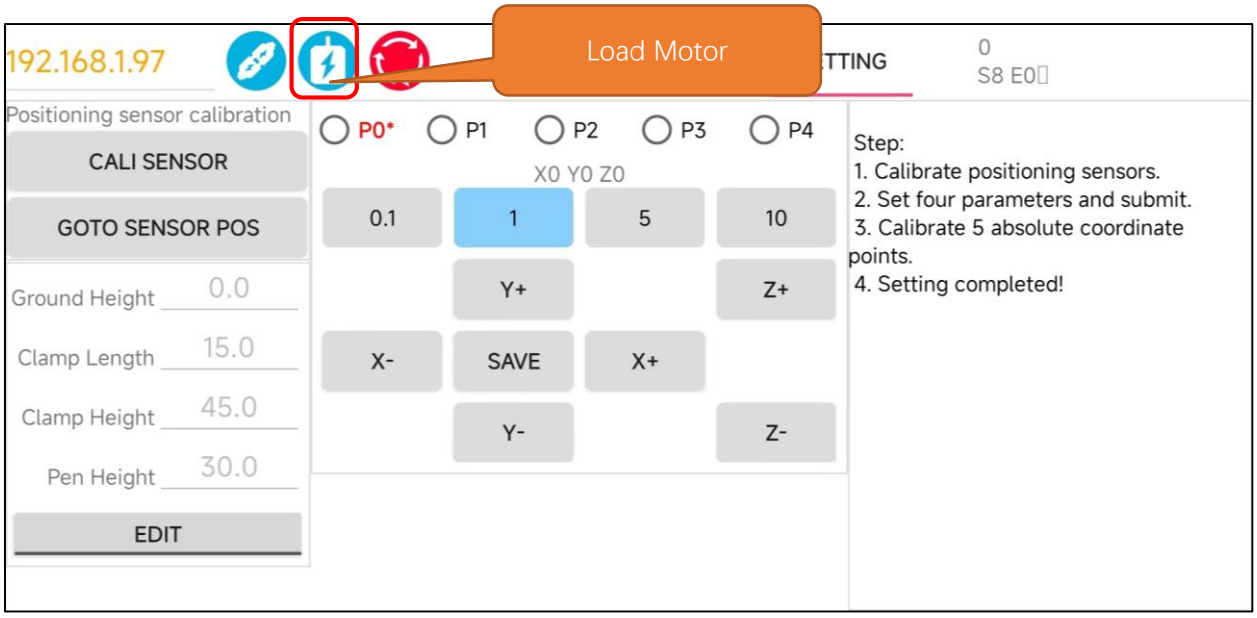


If this is the first time the robotic arm is being powered on, it's essential to perform a calibration configuration to ensure optimal operation. Below is the configuration process for the robotic arm, which we will explain step by step, following the flowchart provided.



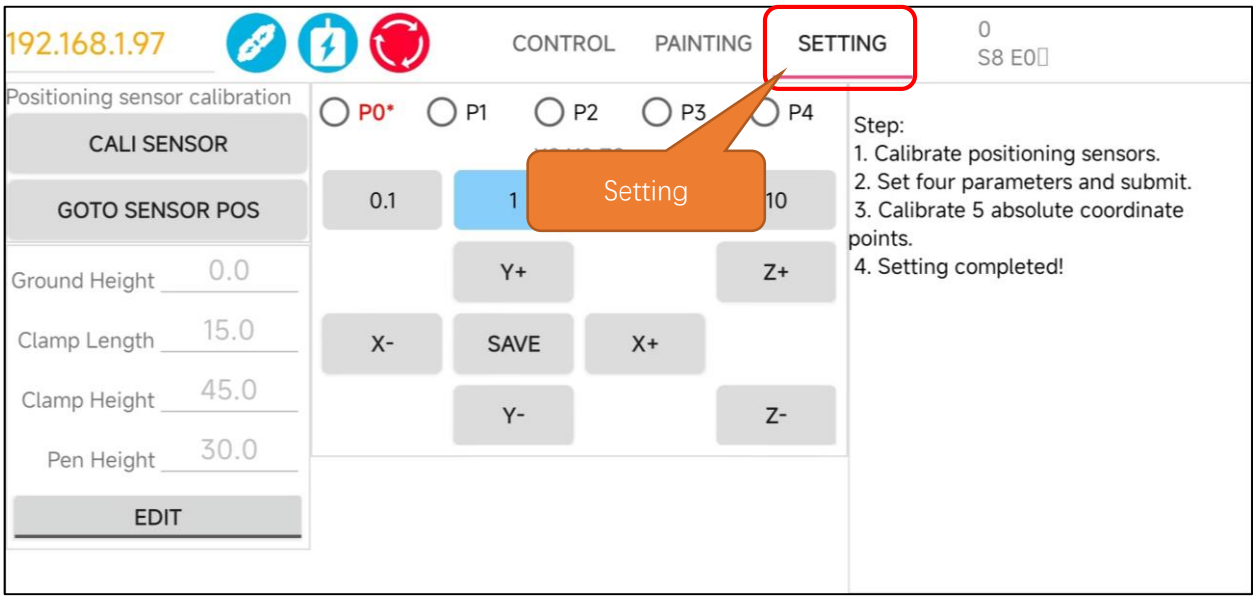
Enable Motors

In its default state, the robotic arm is in a condition where the motors are disabled to ensure safety. Therefore, the first step is to enable the motors by clicking the "Load Motor" button in the software. This action will energize the motors and prepare them for the calibration sequence that follows.

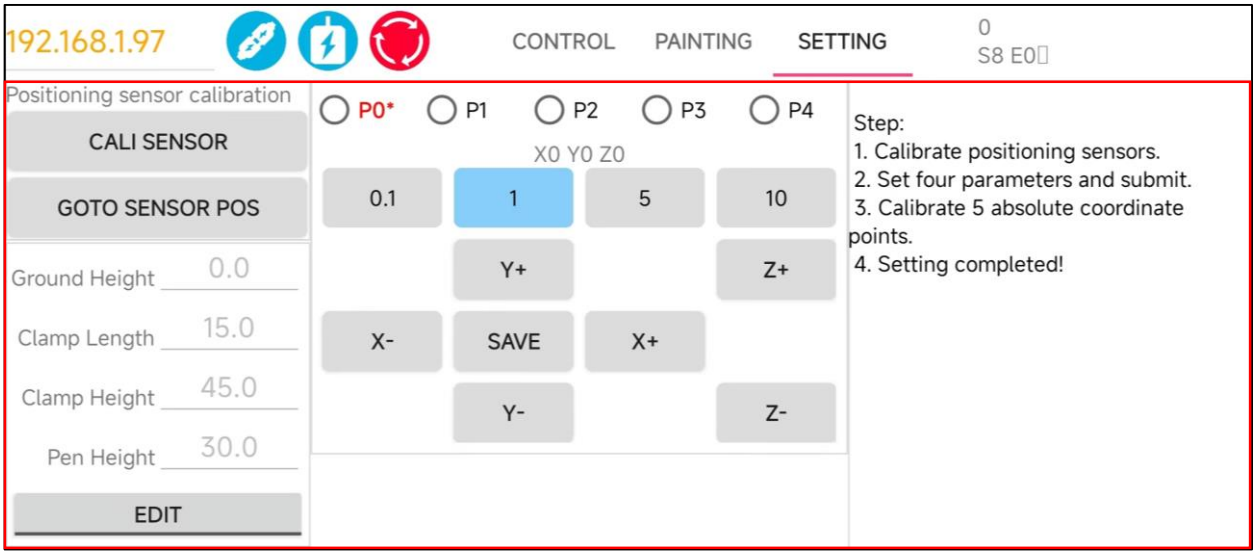


Open the Configuration Interface

Ensure that you have clicked "Load Motor" first, as this is a prerequisite for enabling the motors. Once the motors are enabled, you can proceed to click on "Setting." Doing so will open the robotic arm's configuration interface.



The interface is as shown below.

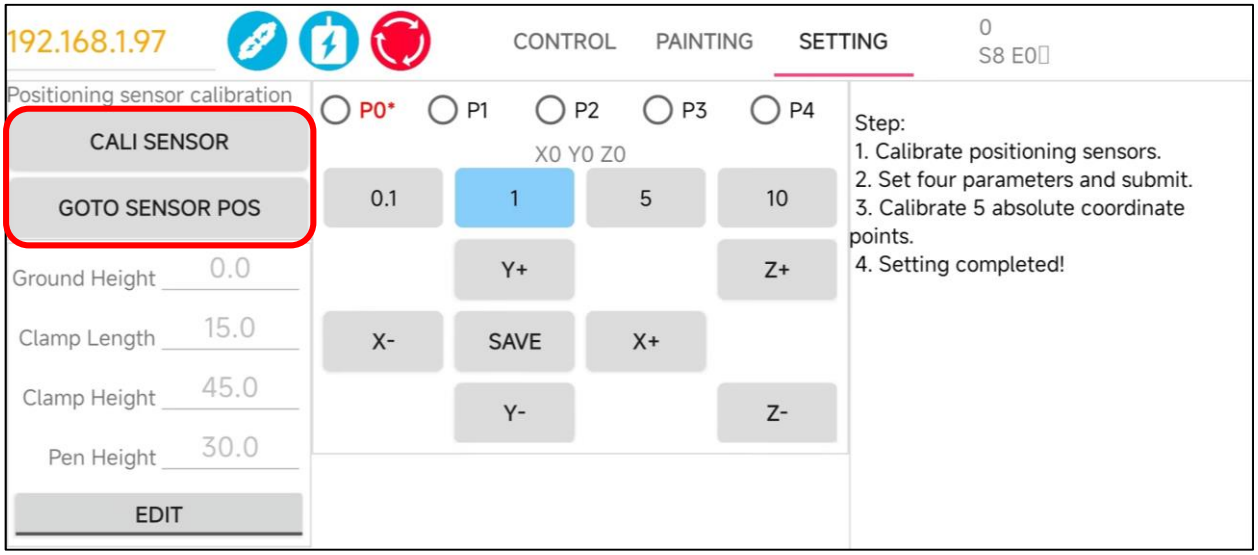


Below, we will explain the configuration of the robotic arm step by step, covering each of these sections.

Step 1. Calibrated Sensor

The sensors on the robotic arm are crucial for assisting the arm in positioning its own range of motion. Without them, the robotic arm would not be able to determine its true coordinate position. Therefore, for the robotic arm to function properly and stably, the calibration of the sensors is extremely important.

At the first use of the robot arm, please click the Calibrated Sensor Point button to calibrate the sensors.



Calibrated Sensor Point is used for calibrating the sensors on the robotic arm. This process involves moving the arm back and forth three times across the sensor. The purpose of this movement is to calculate the offset value where the arm rotates to the center of the sensor. By determining this offset, the system can adjust the arm's movements to ensure it accurately positions itself at the sensor's midpoint.

Goto Sensor Point is designed to return the robotic arm to the center position of the sensor. The behavior of this function depends on whether the sensors have been calibrated:

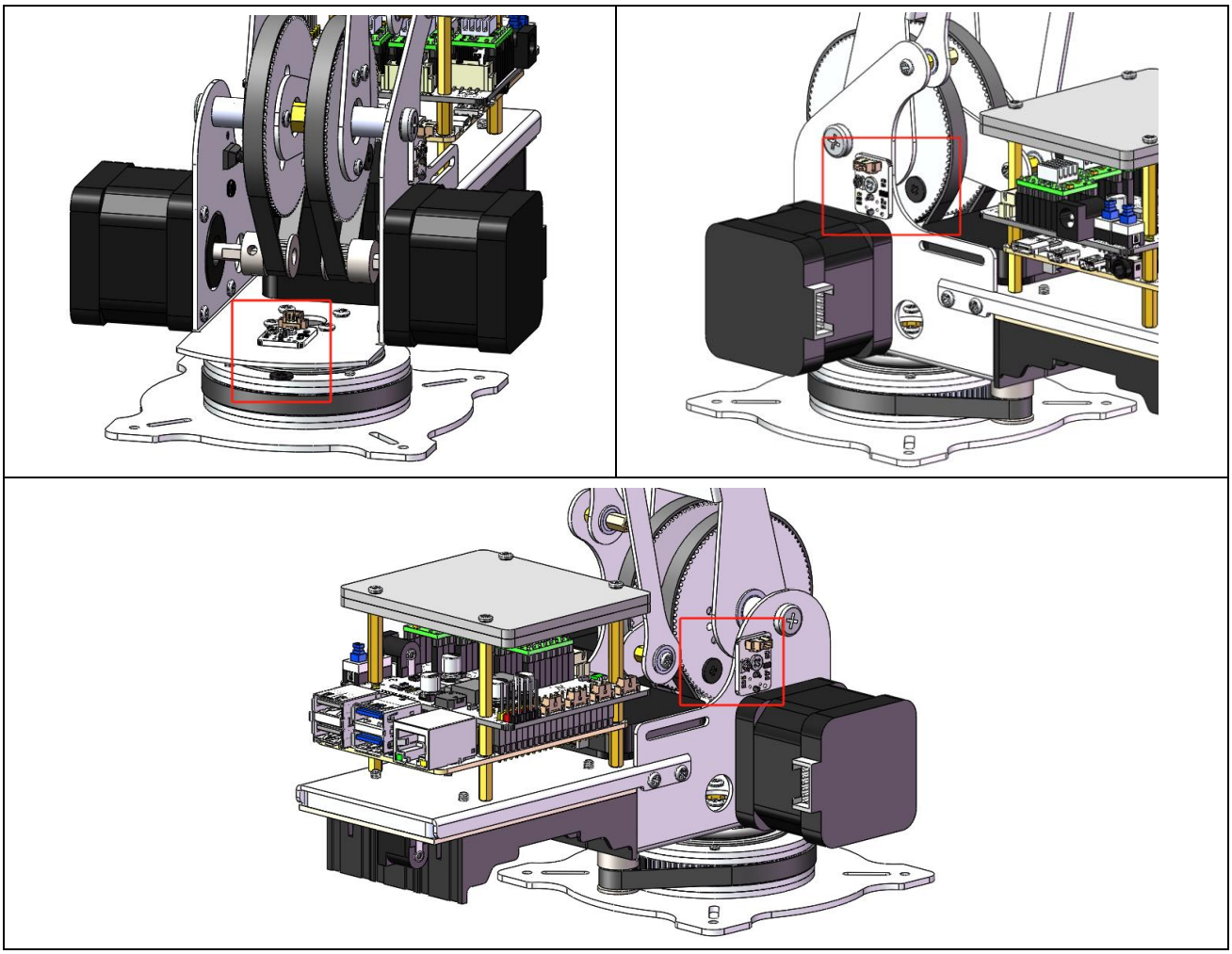
-Uncalibrated Condition: If the sensors have not been calibrated and you click on "Goto Sensor Point," the

robotic arm will move to the edge position of the sensor.

-Calibrated Condition: Once the sensors have been calibrated and you click on "Goto Sensor Point," the robotic arm will accurately move to the center position of the sensor.

-For the initial setup of the robotic arm, please click "**Calibrated Sensor Point**", rather than "Goto Sensor Point".

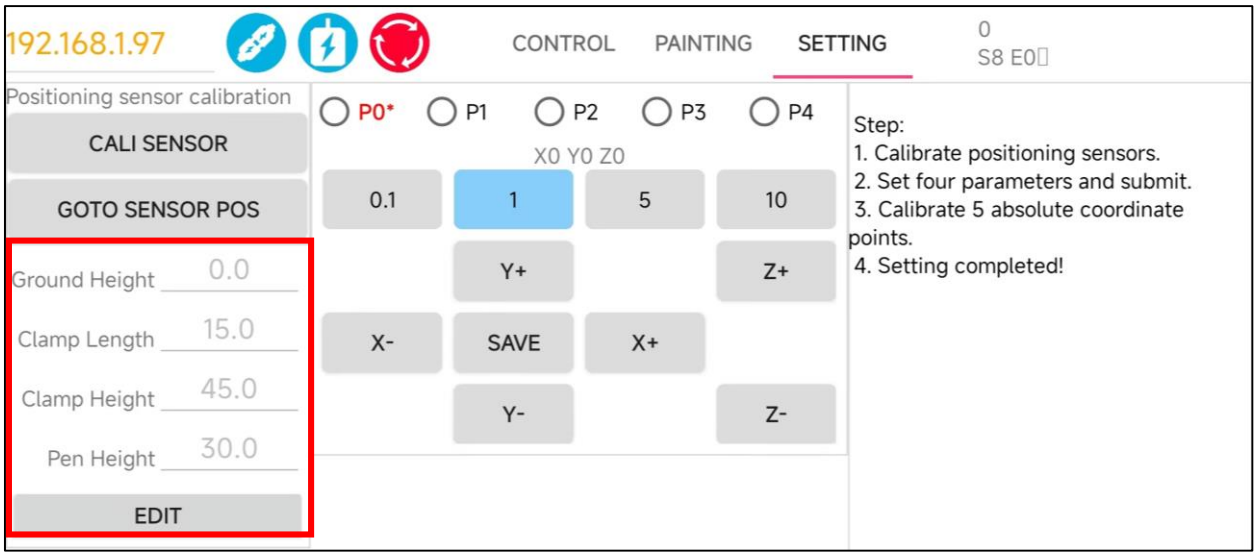
-From the second time onwards, after the sensors have been calibrated, you can directly use the "Goto Sensor Point" function.



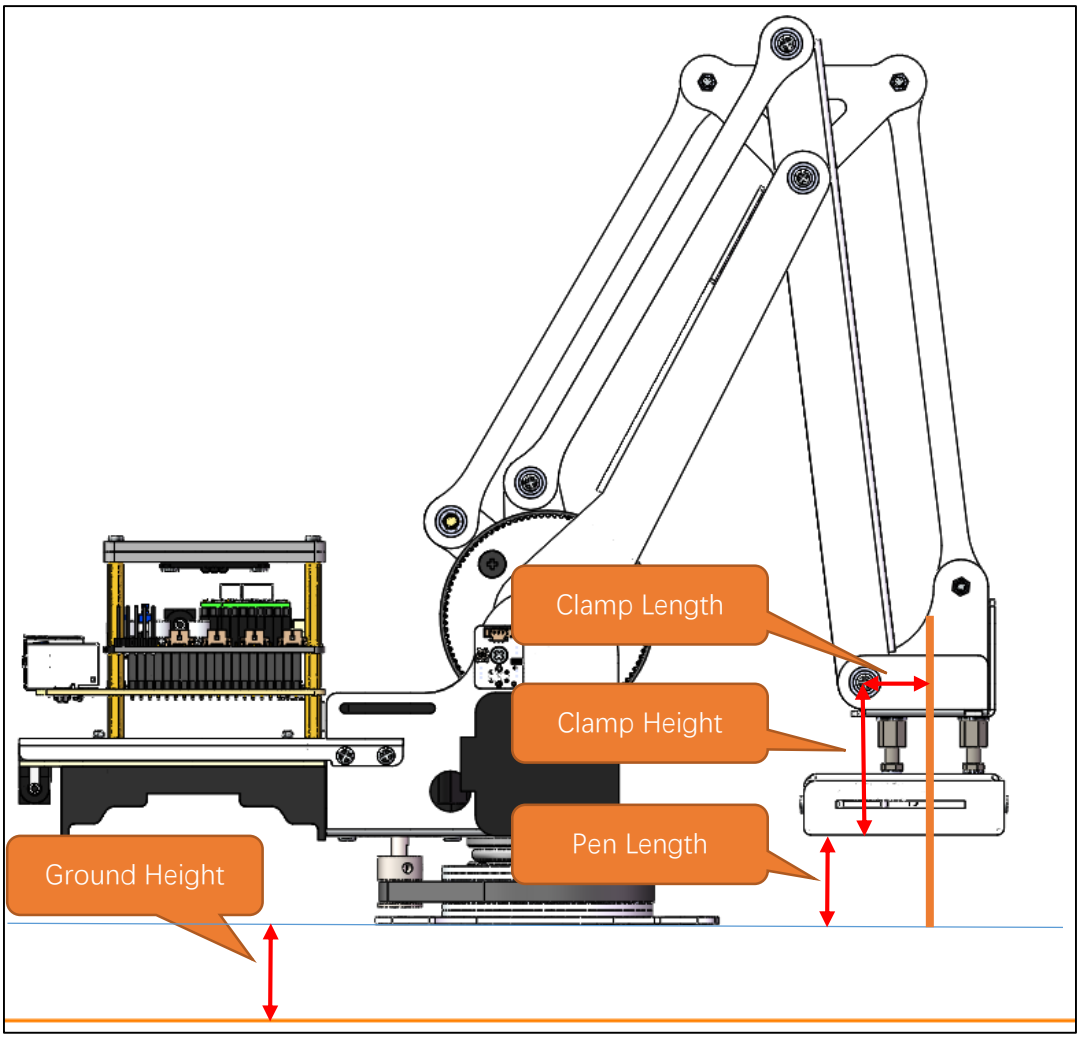
Step 2. Set Arm Parameters

Building upon the first step, we proceed to configure These parameters are crucial for the proper functioning of the robotic arm, so it's important to set them carefully. Please refer to the recommended values provided in the documentation.

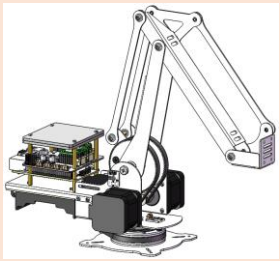
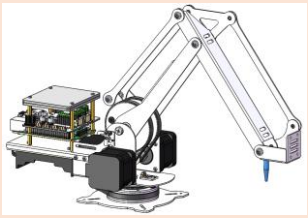
Click Edit to modify the values marked below.

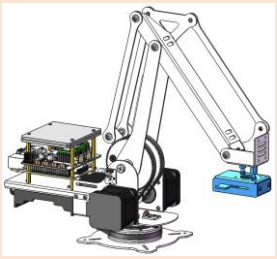
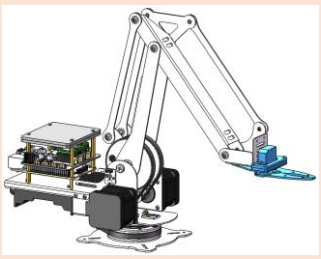
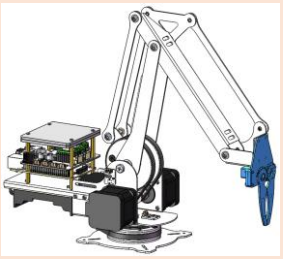


Ground Height refers to the height of the base of the robotic arm from the plane of the coordinate system.
Clamp Length is the length of the end effector, such as a clamp or clamp, at the end of the robotic arm.
Clamp Height is the height of the end effector from the ground.
Pen Height is the distance from the ground to the pen clip when the robotic arm is equipped with a pen clip.


Clamp Length
Clamp Height
Pen Length
Ground Height

The robotic arm operates in four distinct states based on its end effector configuration: No End Effector, Pen clip Installed, Clamp Installed (Horizontal), and Clamp Installed (vertical).

No End Effector	Locating Post
	

pen clip installed	Clamp Installed (Horizontal)	Clamp Installed (vertical)
		

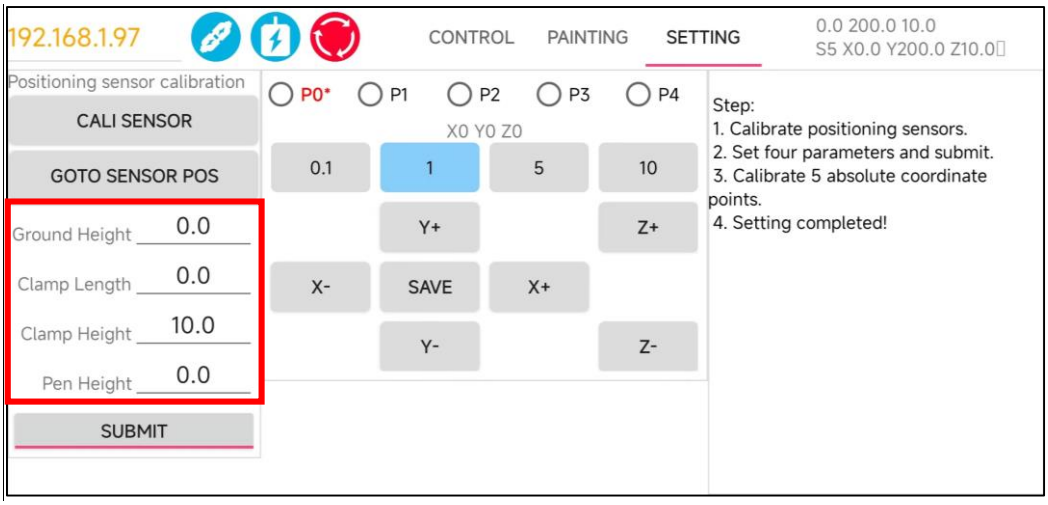
The recommended parameters are as follows:

Mode	Ground Height	Clamp Length	Clamp Height	Pen Height
No End Effector	0mm	0mm	10mm	0mm
Locating Post	0mm	0mm	10mm	30mm
Pen Clip Installed	0mm	15mm(10.5+ radius of pen)	45mm	30mm(Recommended value for pen height)
Clamp Installed (Horizontal)	0mm	70mm	24mm	0mm
Clamp Installed (vertical)	0mm	47mm	85mm	0mm

No End Effector Mode

Tap on the "Submit" button to send the data to the robot arm.

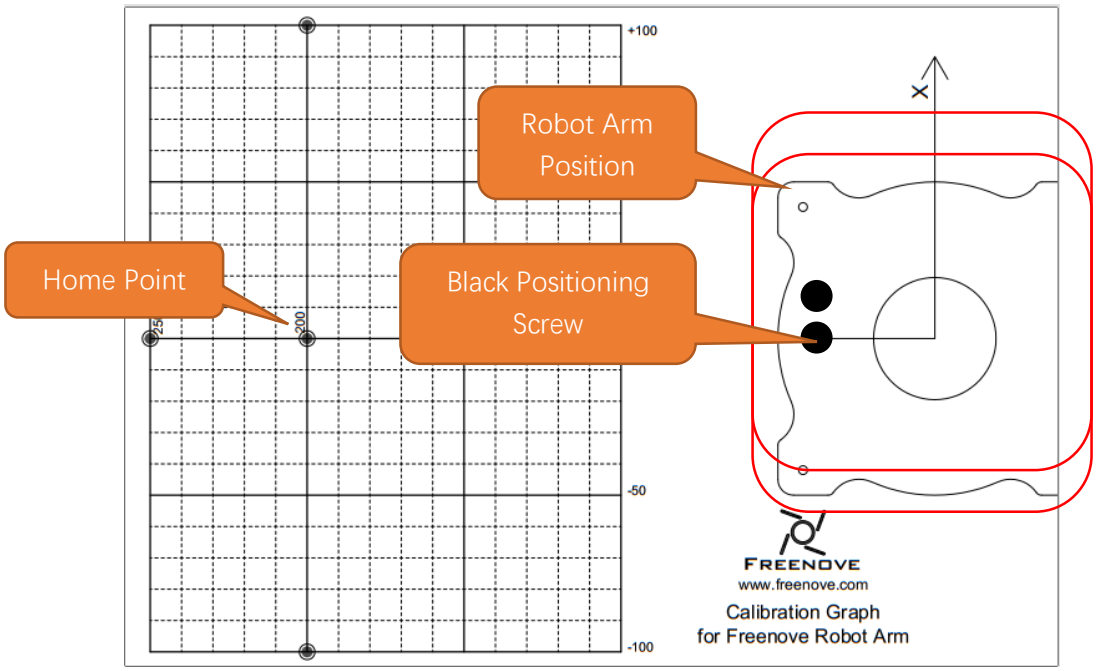
For this example, let's assume you are working in the "No End Effector" mode, which means the robotic arm is not equipped with any additional tools or clamps. The configurations are as shown below.



Home Point Calibration

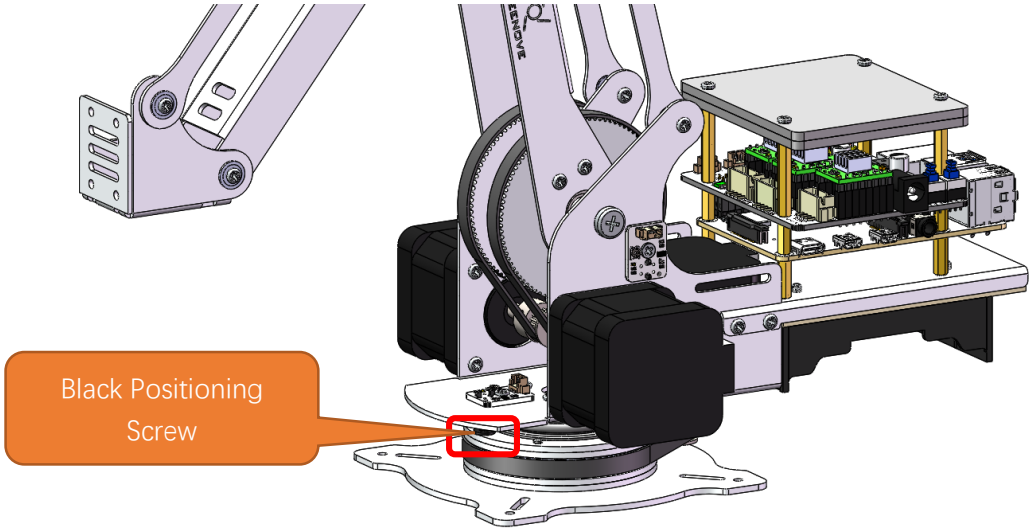
Once Step 2 of the configuration is complete, the robotic arm should be able to operate normally. However, you might notice that the arm's movements are not as precise as expected. This lack of accuracy can be attributed to mechanical deviations. To address these deviations, we can configure to calculate the mechanical angle deviations and apply compensatory adjustments to the angles used by the control system.

To proceed with the calibration process, you will need to locate the calibration paper. If you cannot find the calibration paper, you can print it out using an A4 sheet from a printer. The location of the calibration paper file is **Freenove_Robot_Arm_Kit_for_Raspberry_Pi/Calibration graph.pdf**

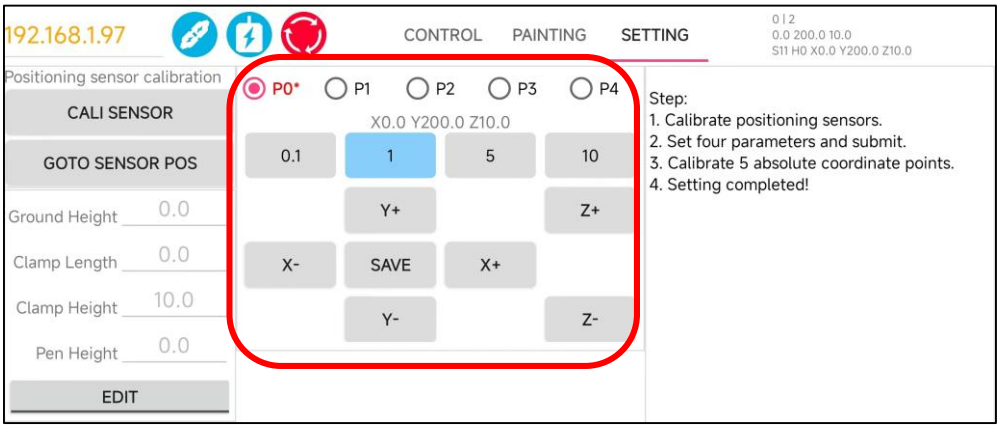


Place the robot arm to the Calibration graph. Align the base to the diagram. Pay attention to the position of

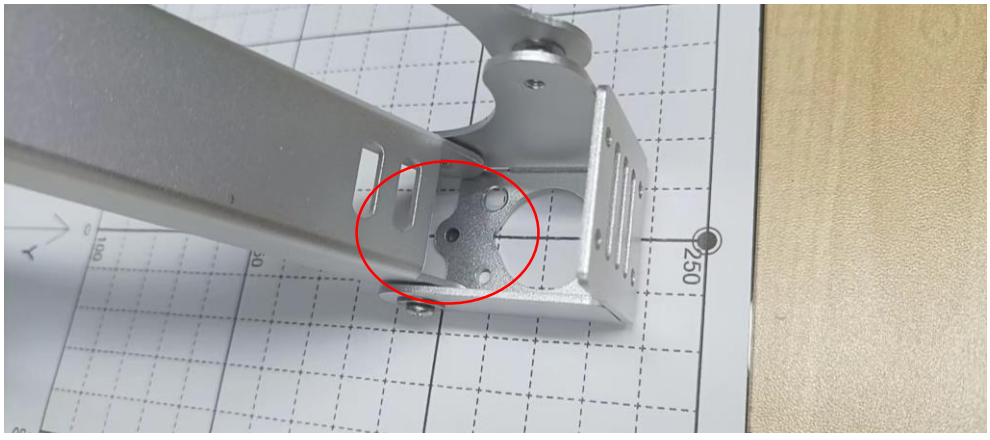
Place the robot arm to the Calibration graph. Align the base to the diagram. Pay attention to the position of the black screw.



Tap P0*. Use the buttons on the right to adjust the position of the robotic arm, guiding it to move to the (0, 200) coordinates marked on the calibration paper. Once the arm is correctly positioned, click the "Save" button to record this calibrated position.

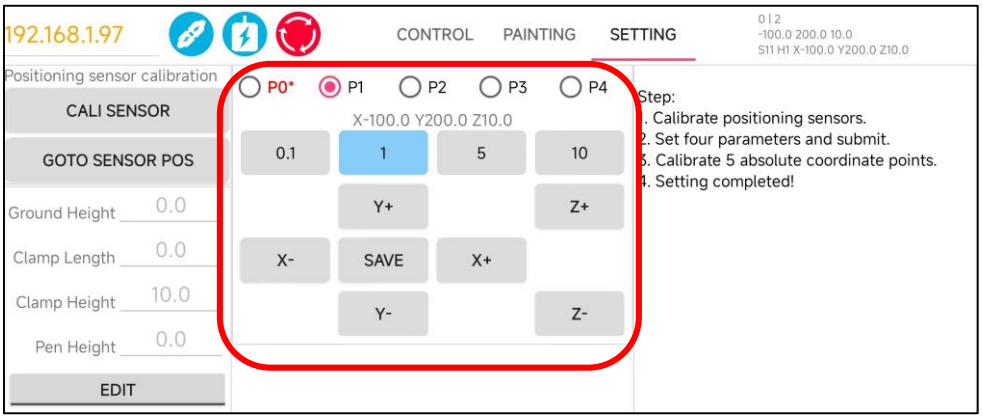


Position the robotic arm's end so that the alignment hole at the tip moves to the (0, 200) coordinates on the calibration paper.

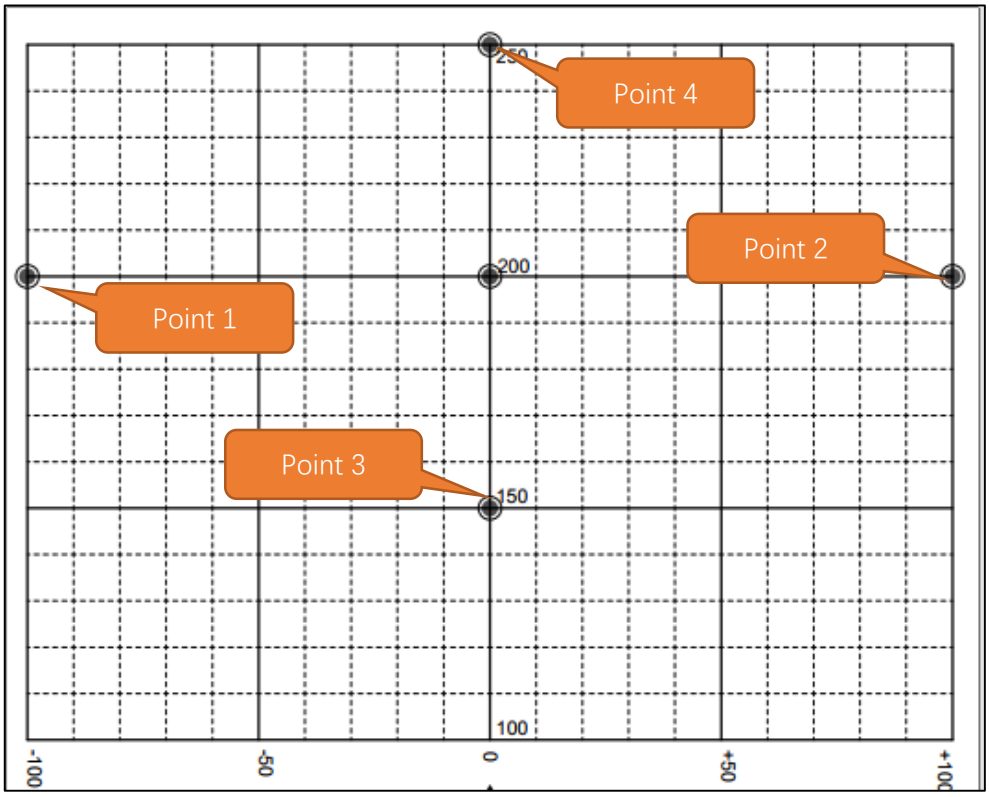


Z Axis Calibration

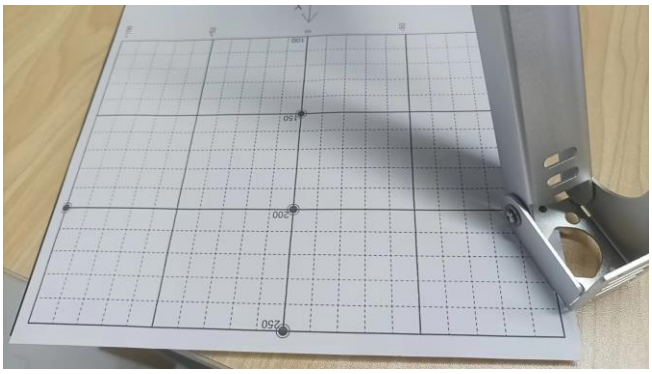
Structural imperfections can sometimes lead to minor inaccuracies in the z-axis coordinate of the robotic arm when it moves to different positions on the calibration paper. To correct for this, we utilize the Calibrated Points 1-4.



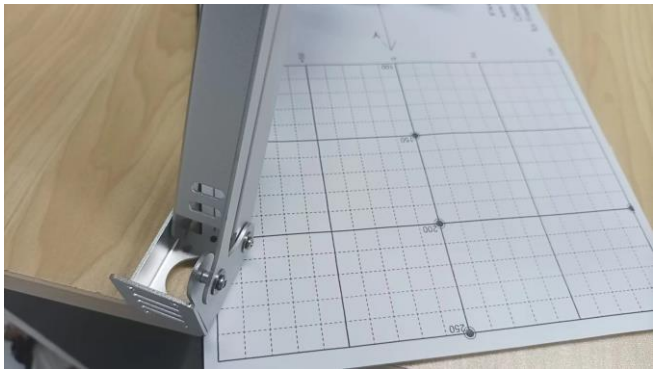
The corresponding potins of P1-P4 on the calibration graph is as shown below.



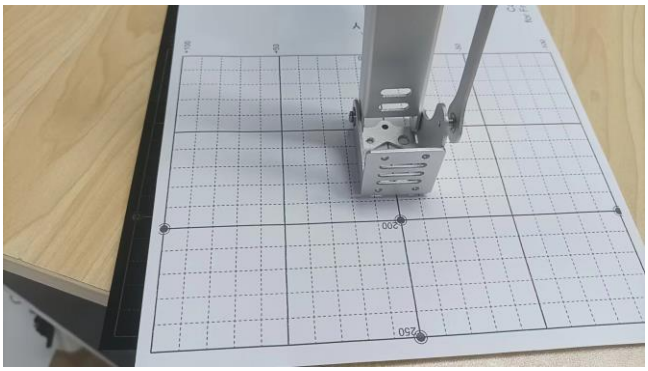
Point 1



Point 2



Point 3



Point 4

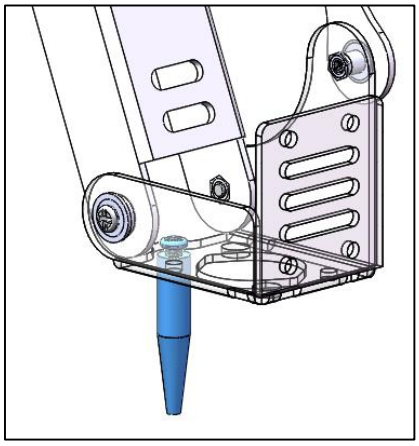
At this point, the calibration of the robotic arm is completed. You can choose to close this interface and return to the main interface to control the robotic arm.

Locating Post Mode

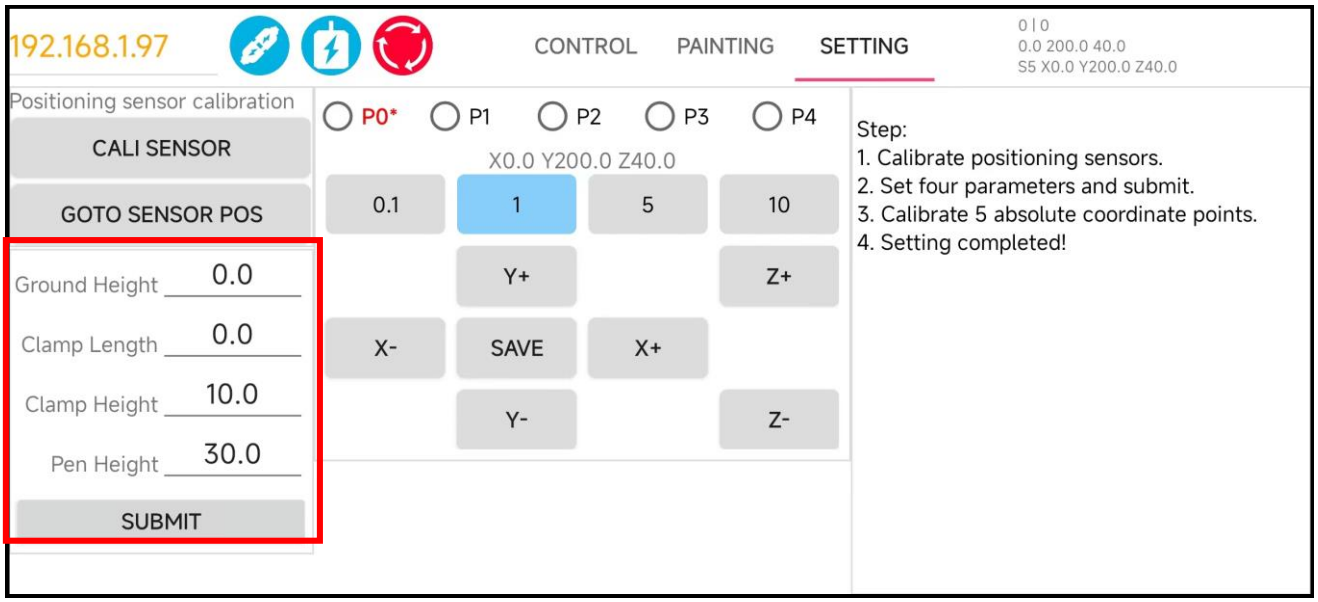
Installing the Locating Post

After you have finished configuring the parameters for Step 2, proceed to click on the "Send Step 2 Parameters" button located in Step 1. For this example, let's assume you are working in the " Locating Post " mode, means that the arm needs to be fitted with a locating Post.
Please find the Locating Post for the M6x30 from the Machinery Parts.

Using one M3x5 screw, secure it to the end of the arm. As shown in the following picture.

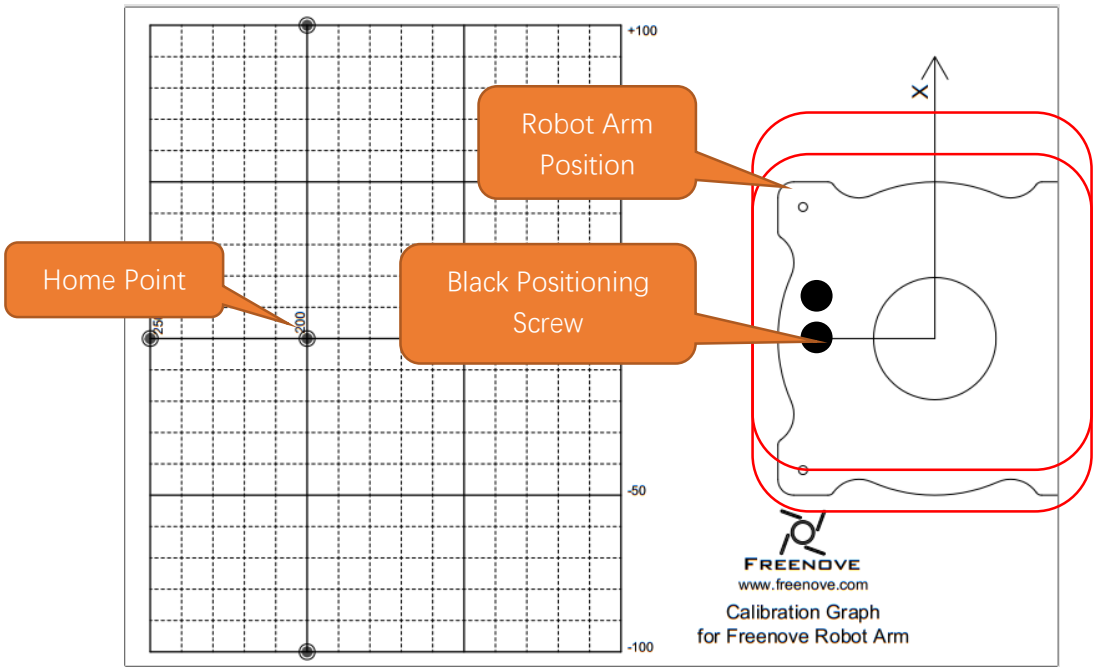


Tap on the "Submit" button to send the data to the robot arm.
The configurations are as shown below.

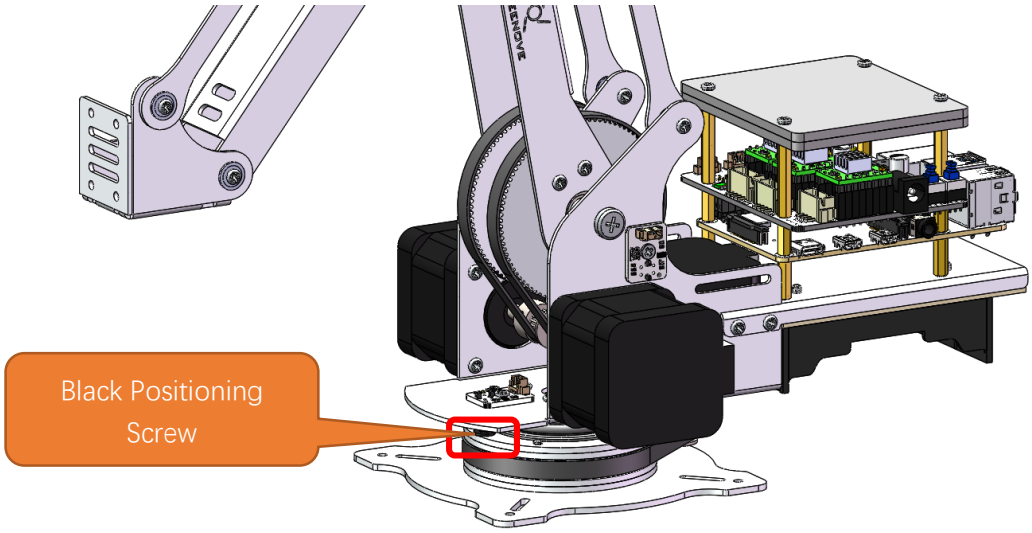


Home Point Calibration

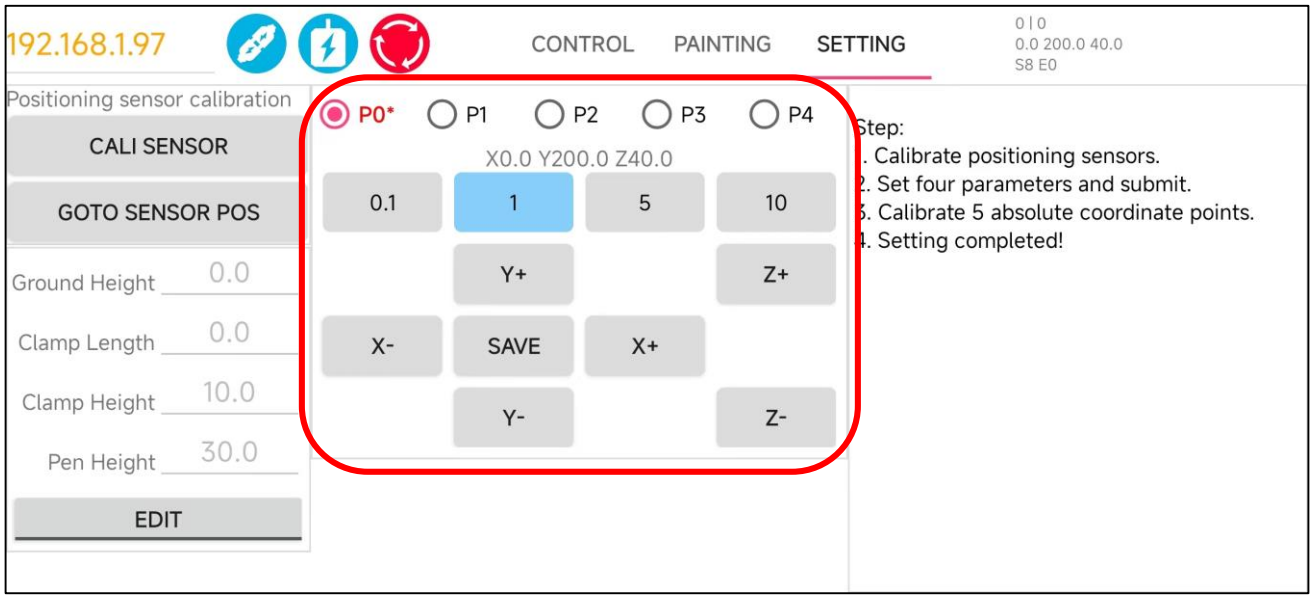
Once Step 2 of the configuration is complete, the robotic arm should be able to operate normally. However, you might notice that the arm's movements are not as precise as expected. This lack of accuracy can be attributed to mechanical deviations. To address these deviations, we can configure to calculate the mechanical angle deviations and apply compensatory adjustments to the angles used by the control system.
To proceed with the calibration process, you will need to locate the calibration paper. If you cannot find the calibration paper, you can print it out using an A4 sheet from a printer. The location of the calibration paper file is **Freenove_Robot_Arm_Kit_for_Raspberry_Pi/Calibration graph.pdf**



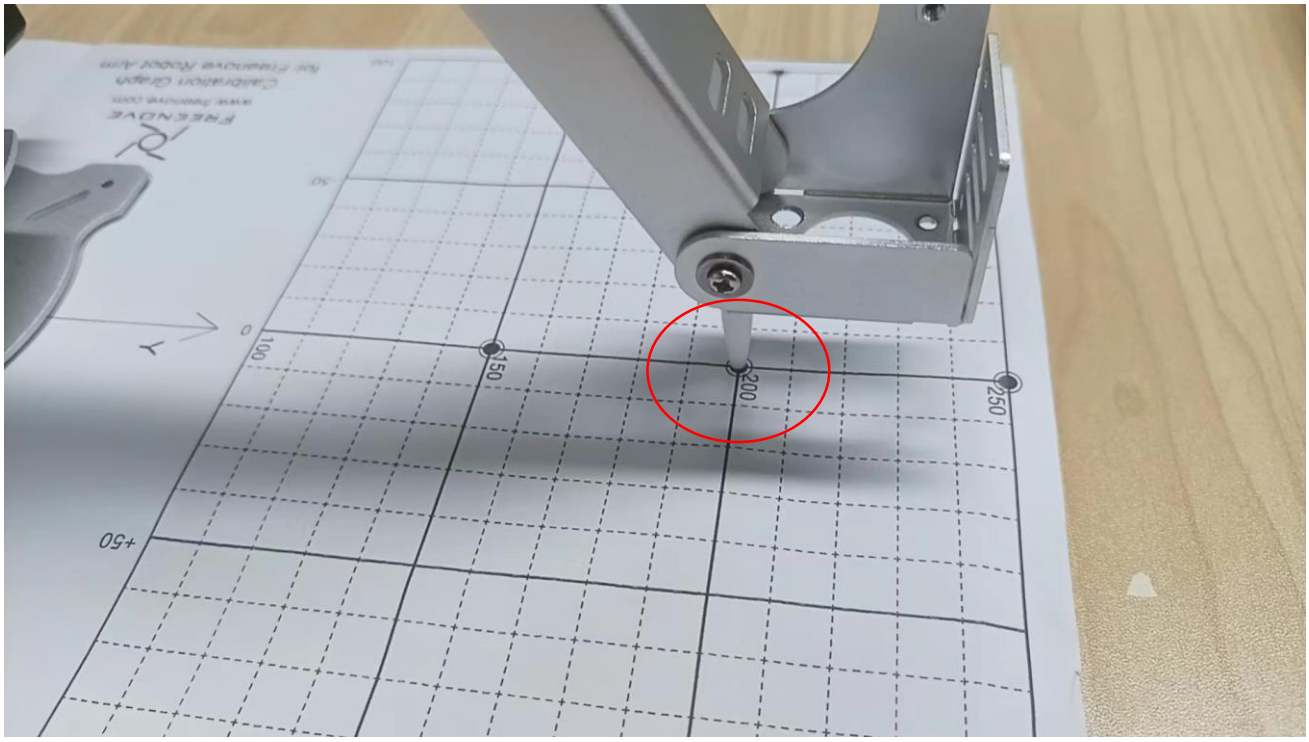
Place the robot arm to the Calibration graph. Align the base to the diagram. Pay attention to the position of
Place the robot arm to the Calibration graph. Align the base to the diagram. Pay attention to the position of
the black screw.



Tap P0*. Use the buttons on the right to adjust the position of the robotic arm, guiding it to move to the (0, 200) coordinates marked on the calibration paper. Once the arm is correctly positioned, click the "Save" button to record this calibrated position.

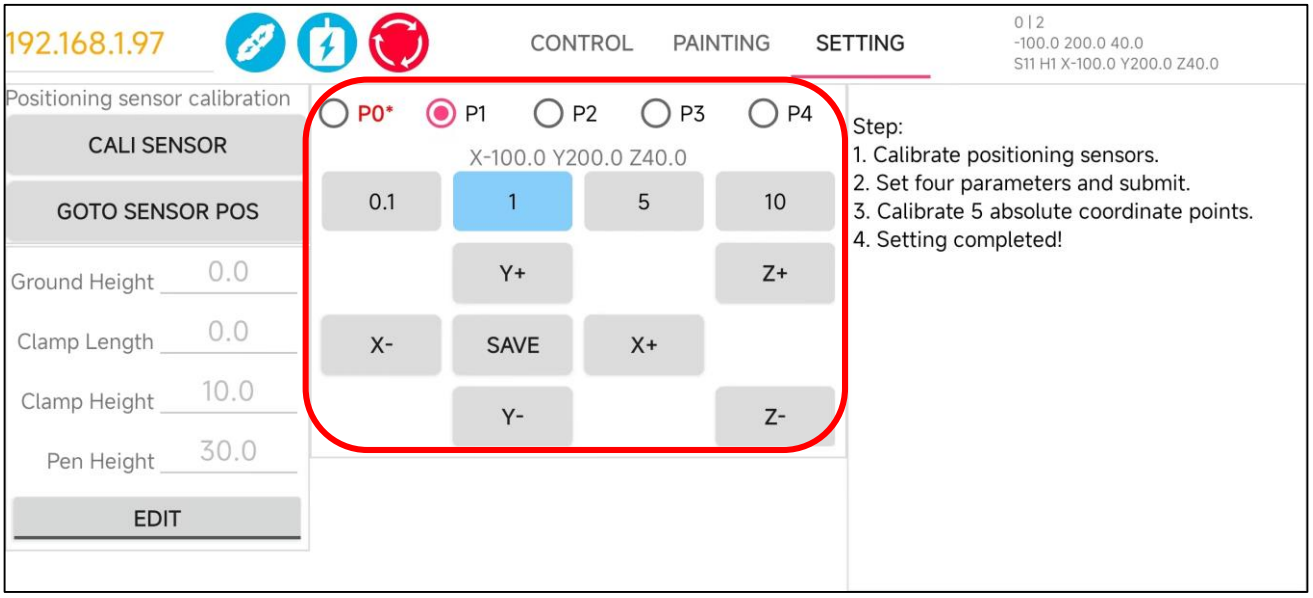


Position the robotic arm's end so that the alignment hole at the tip moves to the (0, 200) coordinates on the calibration paper.

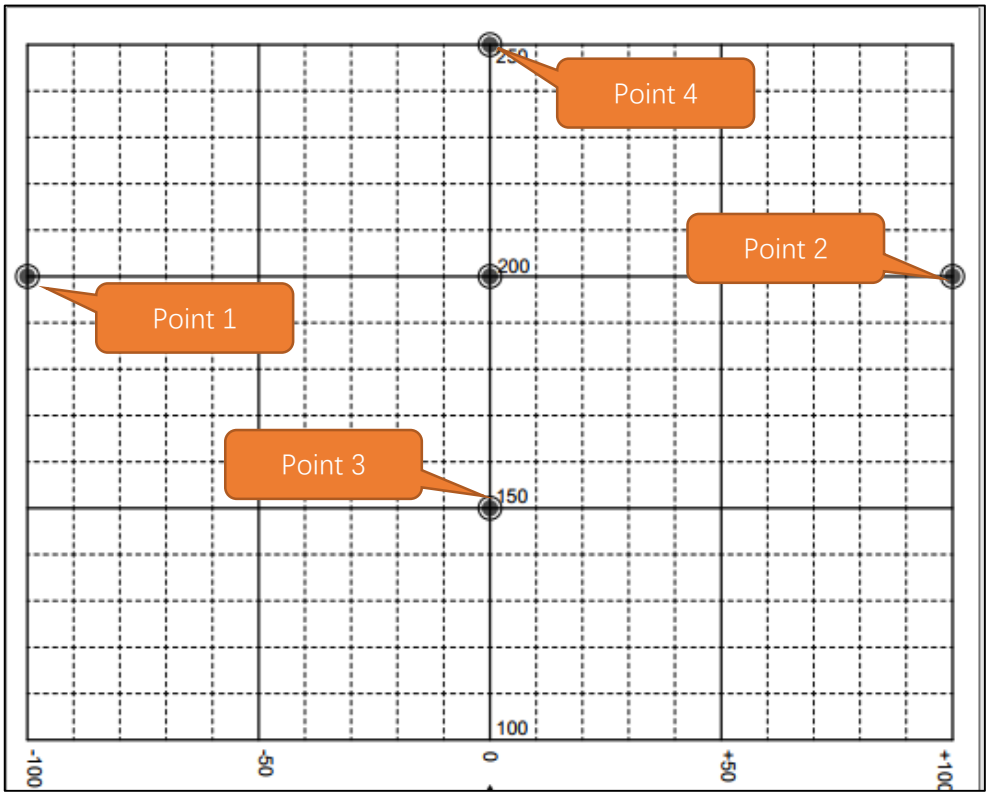


Z Axis Calibration

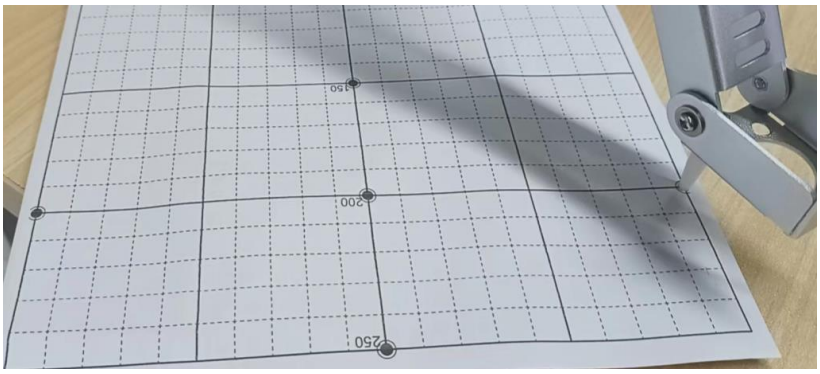
Structural imperfections can sometimes lead to minor inaccuracies in the z-axis coordinate of the robotic arm when it moves to different positions on the calibration paper. To correct for this, we utilize the Calibrated Points 1-4.



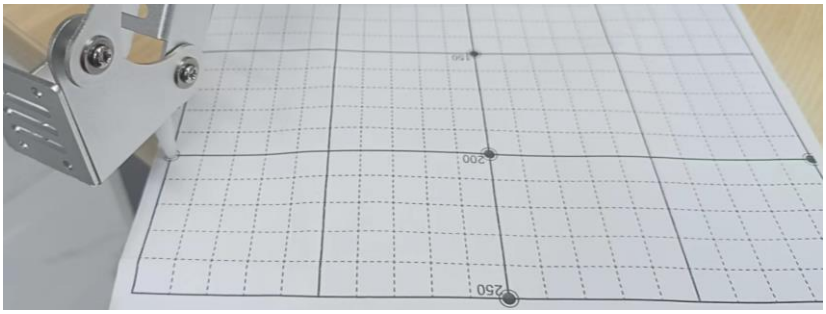
The corresponding potins of P1-P4 on the calibration graph is as shown below.



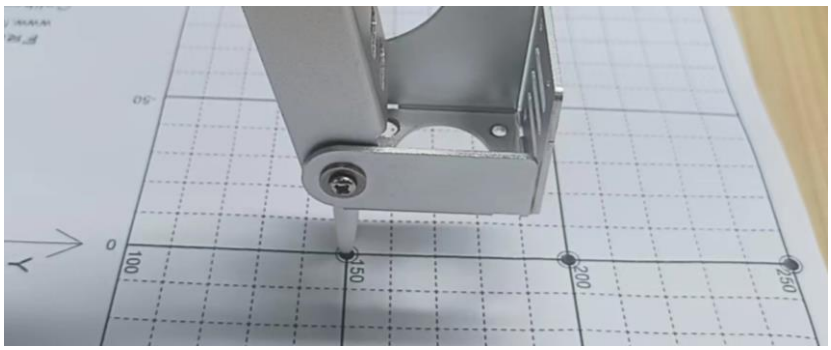
Point 1



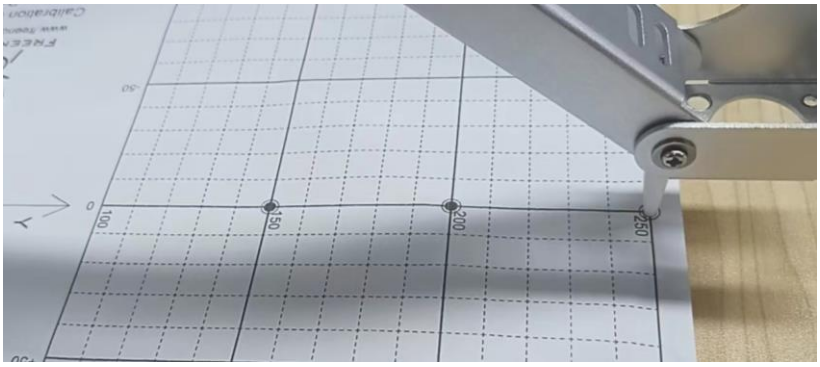
Point 2



Point 3



Point 4



At this point, the calibration of the robotic arm is completed. You can choose to close this interface and return to the main interface to control the robotic arm.

Robot Arm Control

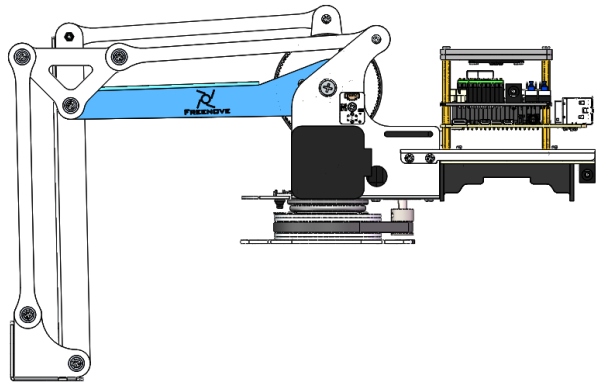
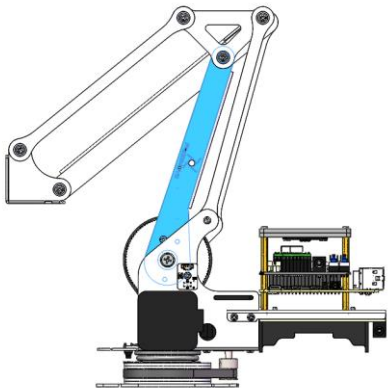
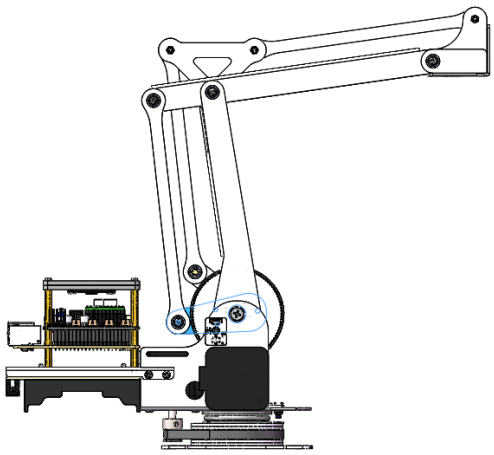
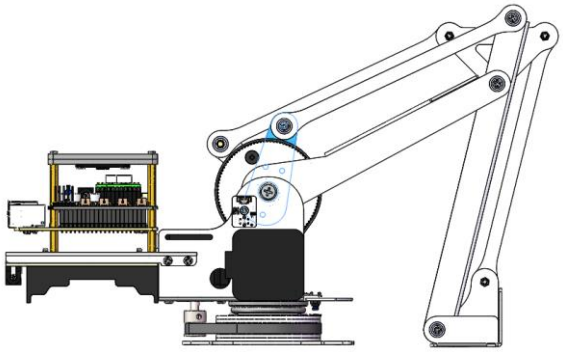
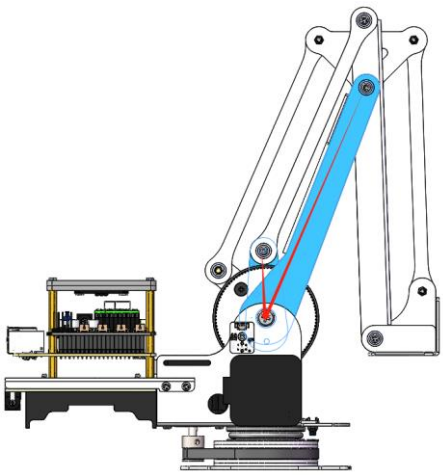
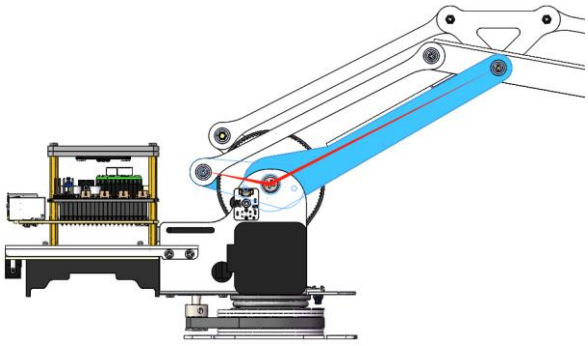
Robot Arm Movements

Click Control to jump to the robotic arm control interface. As shown in the figure below, you can control the movement of the robotic arm through the buttons in the figure below.

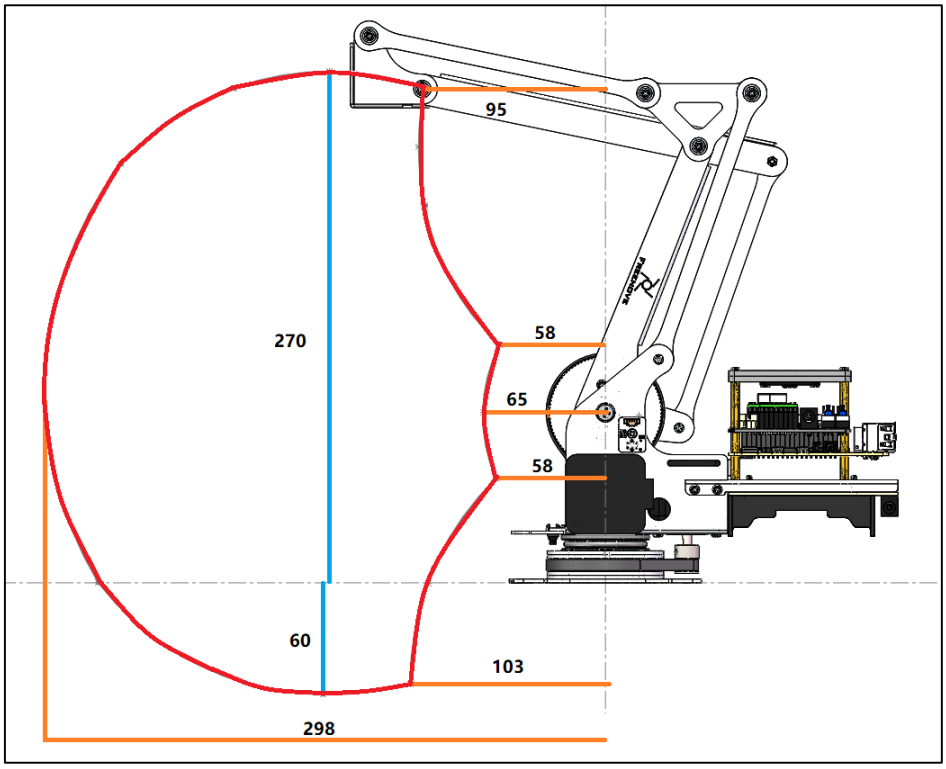
Sensor Point	After every time you click "Load Motor," or when the stepper motor experiences a missed step action, please click the designated button. This button will return the robotic arm to the middle position of the sensor, effectively resetting the robotic arm's positional coordinate information.
0.1、1、5、10	The robot arm will move with a step size corresponding to the number.
Home	Control the robot arm to move to the Home point coordinates.
Home Up	Control the robot arm to move above the Home point coordinates.
X、Y、Z	Control the robotic arm to move up, down, left, right, and back.

Rotation range

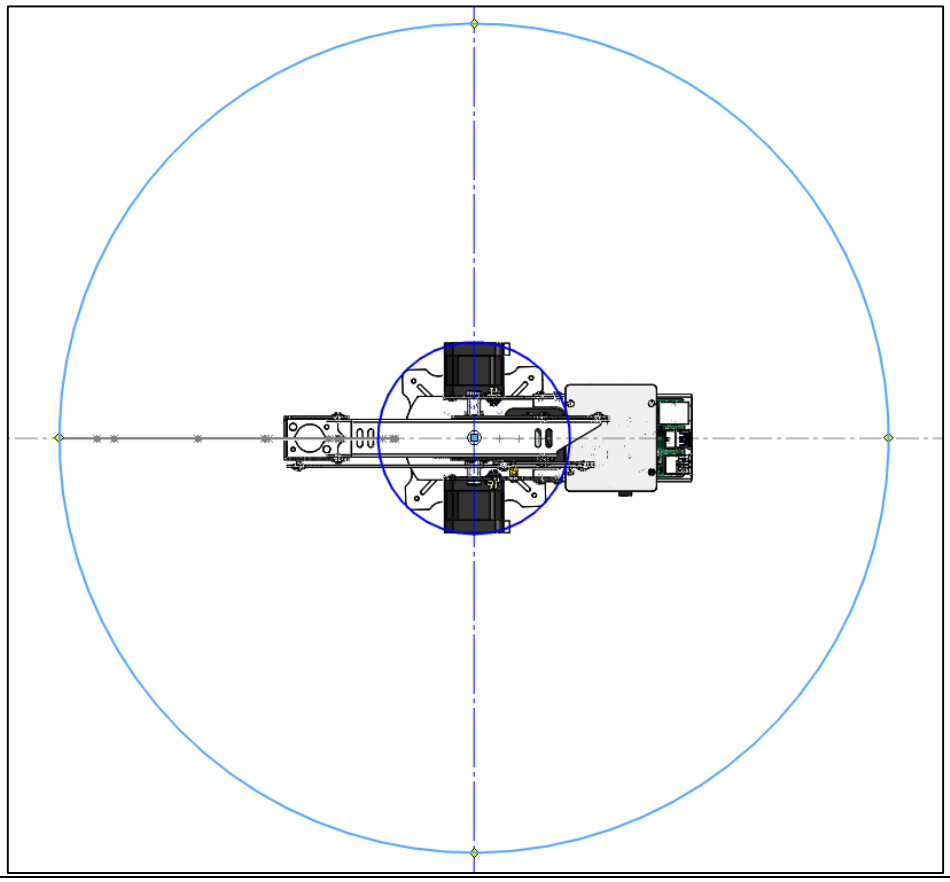
Due to the physical constraints of the structure of the robot arm, there are certain coordinates that the robot arm cannot reach. Therefore, we have implemented an angular limit range for the movement of the robotic arm, as shown in the figure below.

Rotation limit Angle of the left-motor 1: 0°	Rotation limit Angle of the left-motor 2: 110°
	
Rotation limit Angle of the right-motor 1: -12°	Rotation limit Angle of the right-motor 2: 98°
	
Minimum Angle between the two sides: 26°	Maximum Angle between the two sides: 150°
	

The motion range of the robot arm is shown in the figure below.

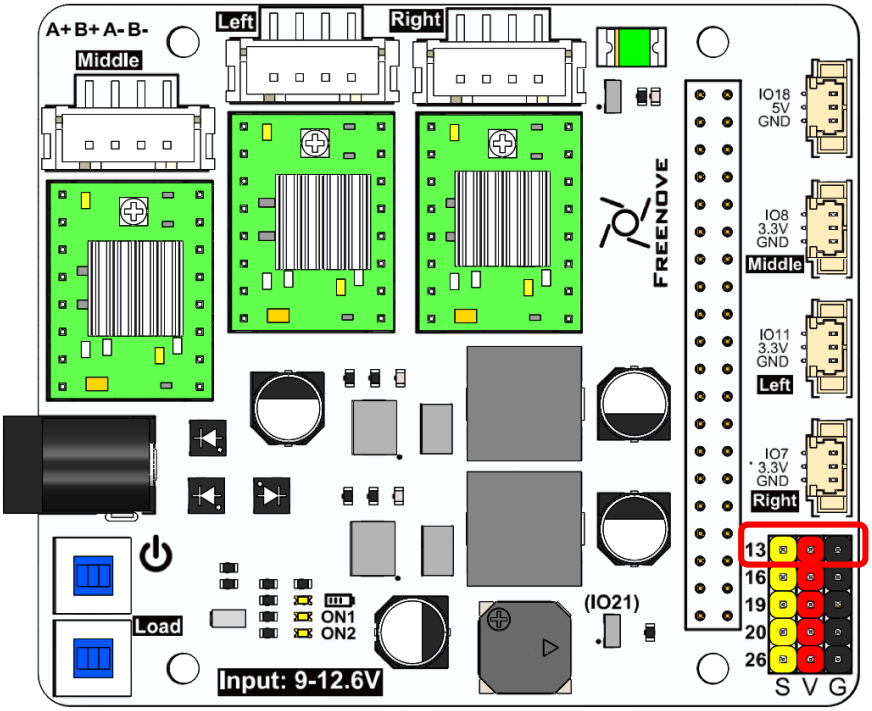


The motion range of the robot arm is shown in the figure below.

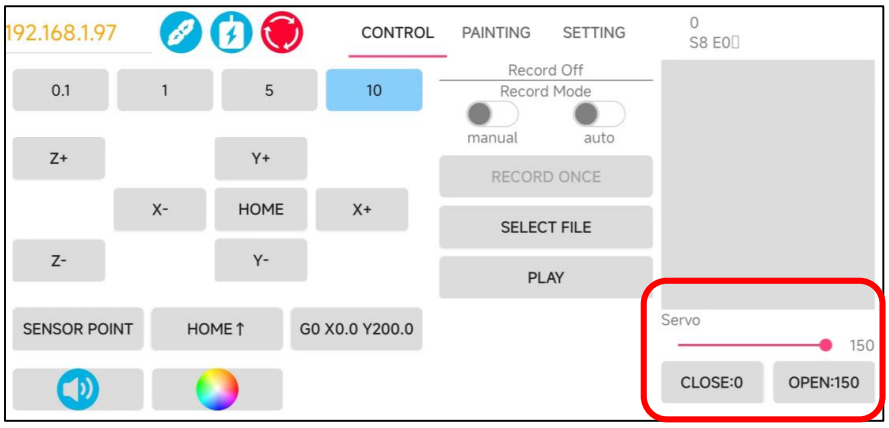


Robot Clamp Control

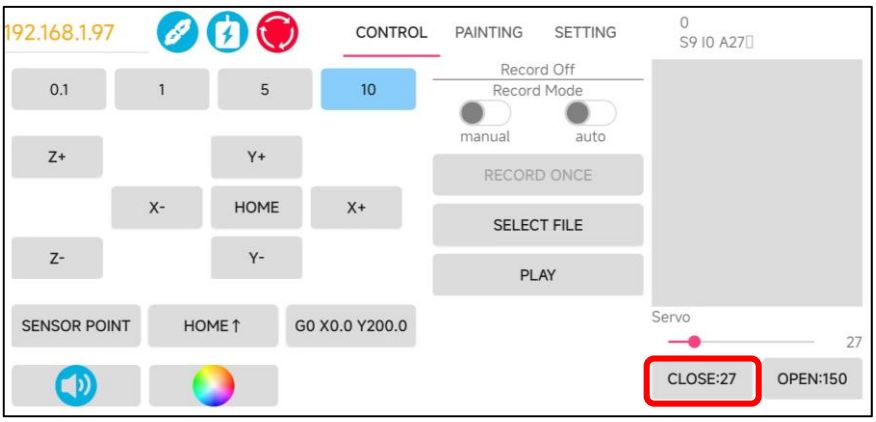
Before using the software to control the robot clamp, please ensure that the clamp is properly installed. Please note that when controlled with Phone APP, the GPIO pin of servo is fixed. Please connect the servo to GPIO13.



As shown in the figure below, you can control the clamp of the robotic arm through the buttons or slider as shown in the figure below.



When operating the clamp via the software, the functionality of the control buttons is straightforward:
Clicking the "+" button will cause the clamp to open, expanding its jaws to release any held objects.
Clicking the "-" button will make the clamp close, clamping down on whatever is held between its jaws.
If you want to change the value on the Close or Open button, please slide the slider to the appropriate angle position, and then long press the corresponding button, as shown in the figure below.



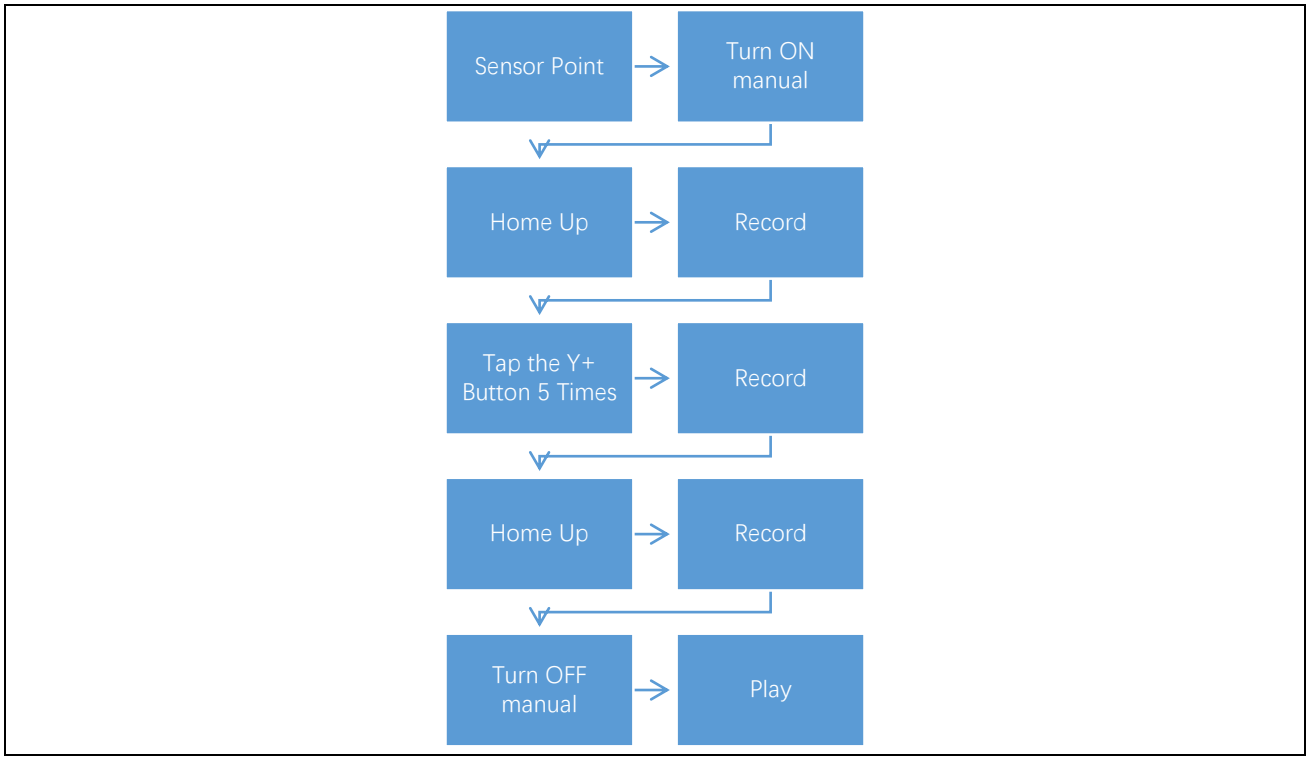
It is important to be mindful of the angle at which you use the clamp to pick up items. For instance, if the clamp has rotated to 30 degrees and the item is securely grasped, it is not advisable to continue tightening the clamp. Doing so could result in the servo exerting more power than necessary to hold the item, which could potentially damage both the servo and the object being gripped.

Instructions Recording Mode

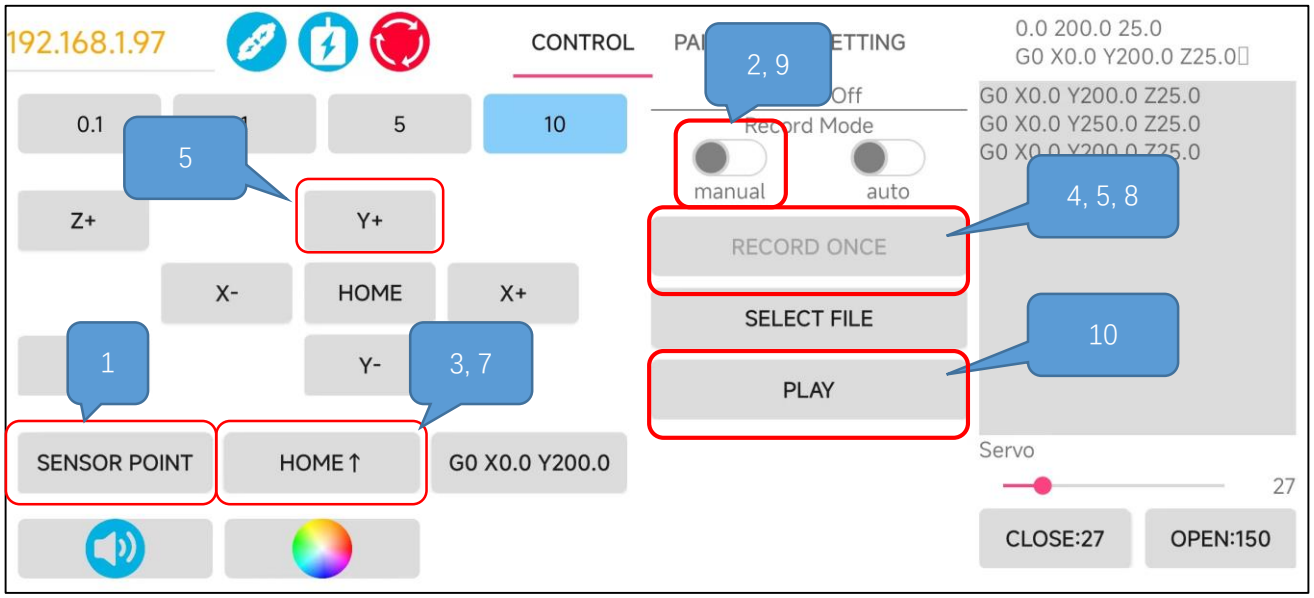
To enhance the interactivity and fun of using the robotic arm, we've incorporated an instruction recording mode within the software.

Manual	Manual Recording Mode: In this mode, you actively record operational commands by using the Record Once feature. Each time you click, the most recent App operation command is recorded and displayed on the right side of the interface.
Auto	Automatic Recording Mode: When this mode is active, any operation performed within the App is automatically recorded and displayed on the right side without the need for manual clicks.
Record Once	Each time the button is clicked, the previous App operation instruction is recorded and displayed on the right.
Select File	time it is clicked, the user selects a local file, and the software loads the instructions in the local file into the display area on the right.
Play	Each time it is clicked, the operation command displayed on the right is sent to the robot arm for execution.

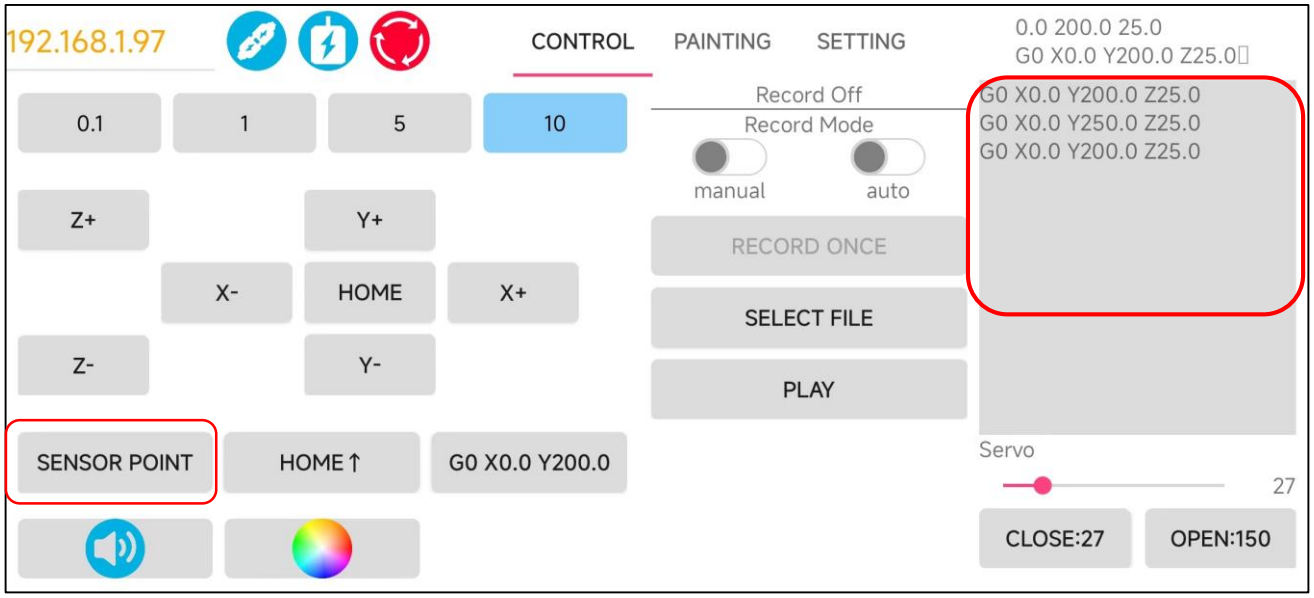
For example, to program the robotic arm to perform sensor calibration, move to the Home Point, then to Point 4, and finally return to the Home Point, you would follow these steps:



Refer to the steps above, follow the order below to operate.



Each time Play is tapped, the content displayed on the right is sent to the robotic arm to be executed. Please note that if the robot arm has not returned to the sensor center since it was started, please click Sensor Point before tapping Play. If the robotic arm has not been centered and you attempt to execute commands without centering, the arm will use its buzzer to alert you. The number of beeps corresponds to the number of commands in the display area that are queued for execution.

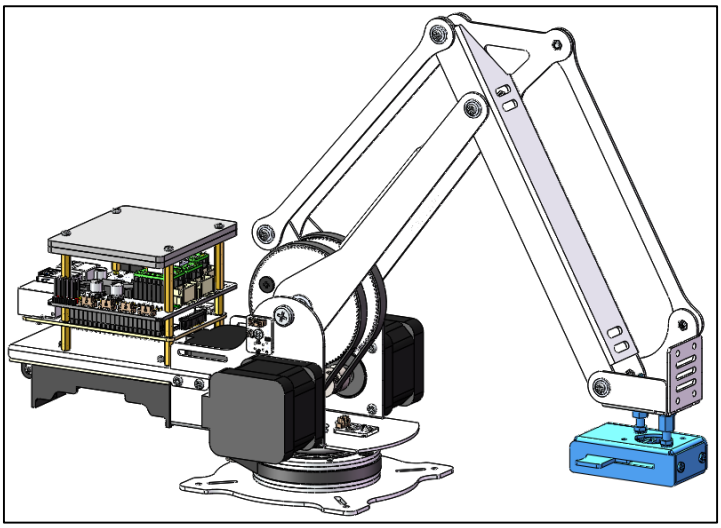


Each tiem Recore Mode is disabled, the contents shown on the right will be stored to local. Similarly, each time Select File is tapped, the App will open the directory /Document/Freenove.

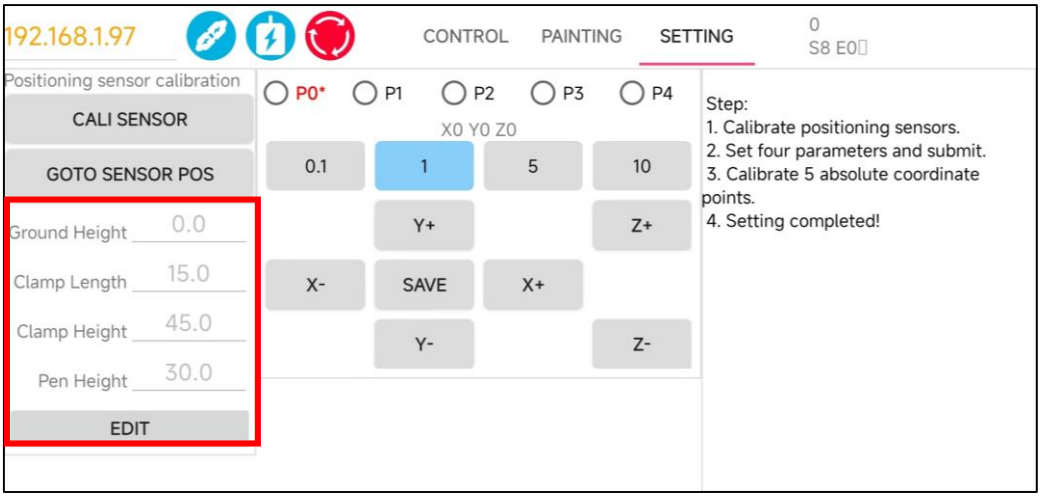
Drawing Mode

To make the experience with the robotic arm more engaging and entertaining, we utilize it for an attempt at drawing.

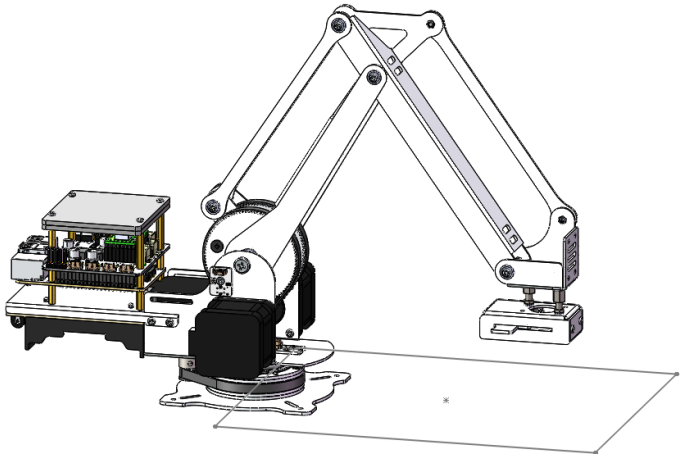
Firstly, ensure that the robotic arm is laid flat on a table surface and that the pen clip is properly installed.



Open the Setting interface. Tap on Edit and configure as shown below. Tap on Submit.

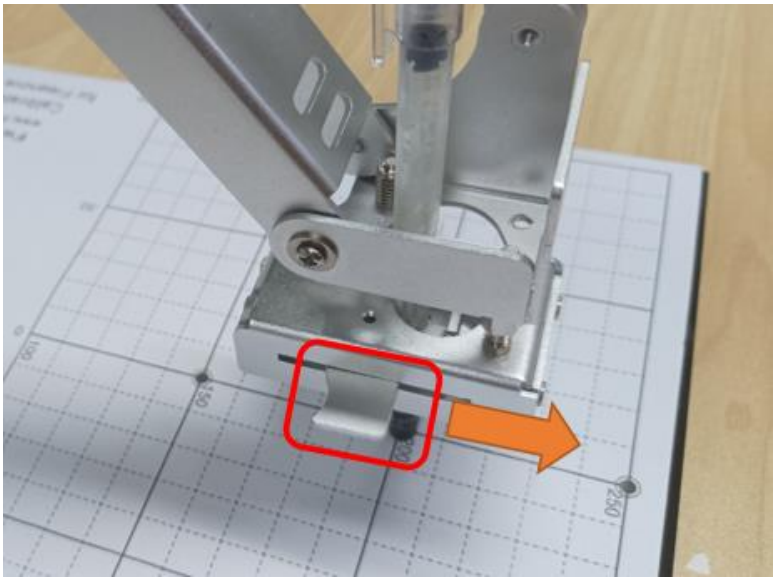


You can see that the pen clip stays 30.0mm above the calibration paper.

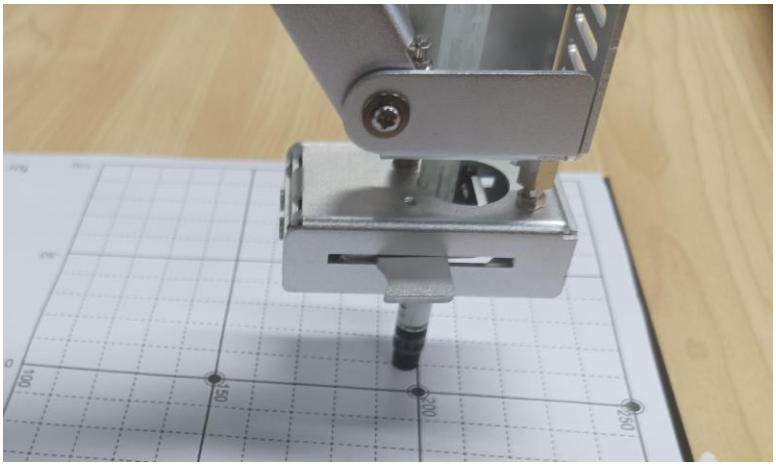


Manipulate the spring-loaded clip of the pen holder to insert the pen into the clip. Please ensure that the pen is positioned to naturally stand vertically against the paper, just making contact with it.

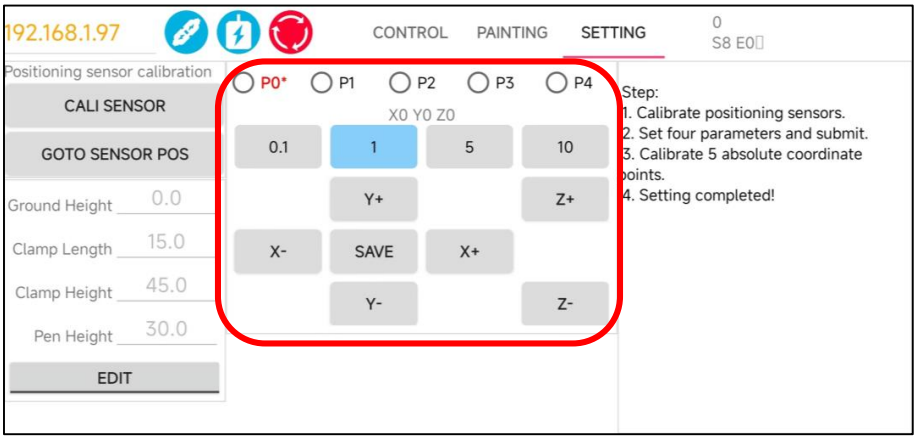
In actual use, please use the pen tip to calibrate, the result is more accurate.



If the pen clip at the end of your robotic arm is not at the P0* position, as shown in the figure below,

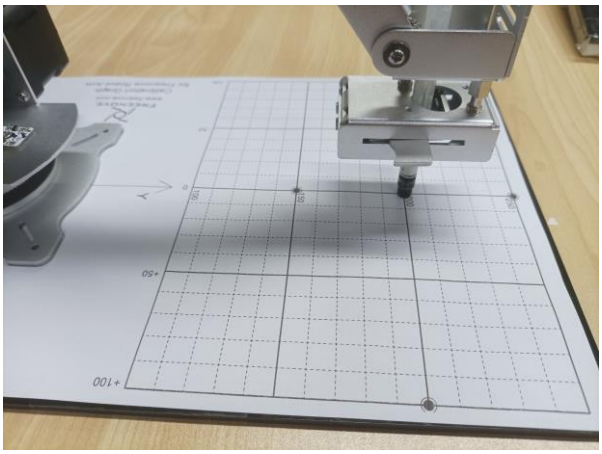


then you need to readjust P0* and Points 1-4 so that the end of the pen clip of the robotic arm can run parallel to the paper.

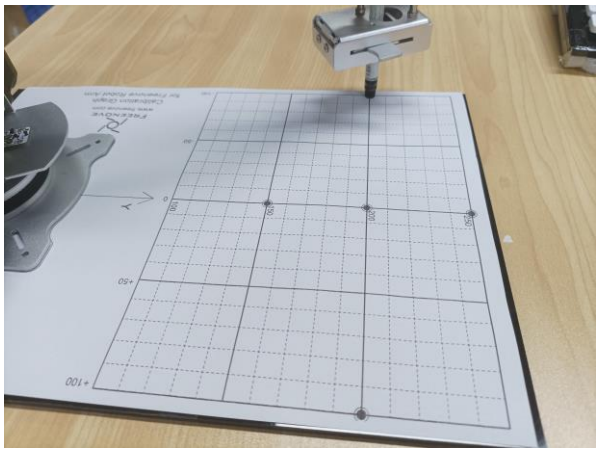


The configuration is as shown below.

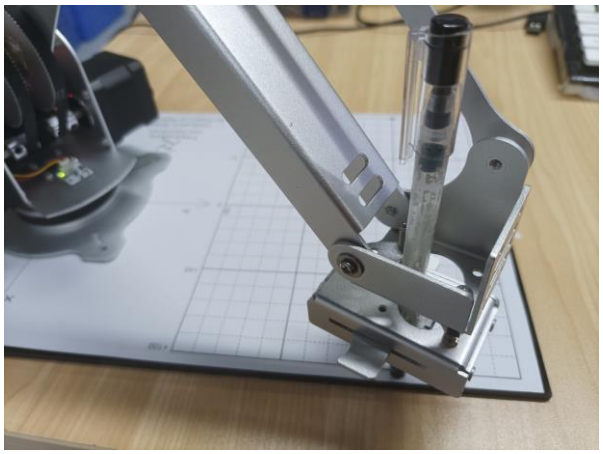
P0*



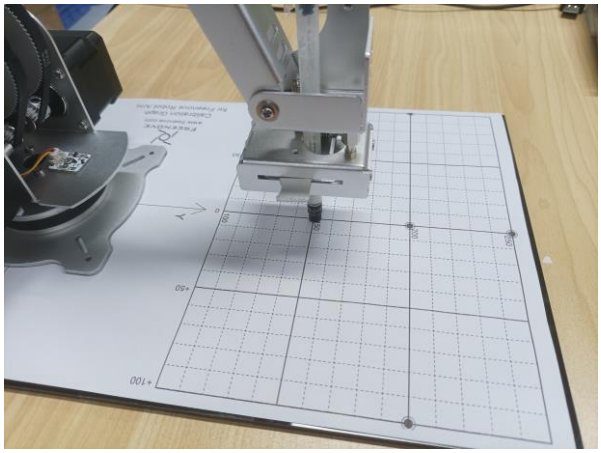
P1



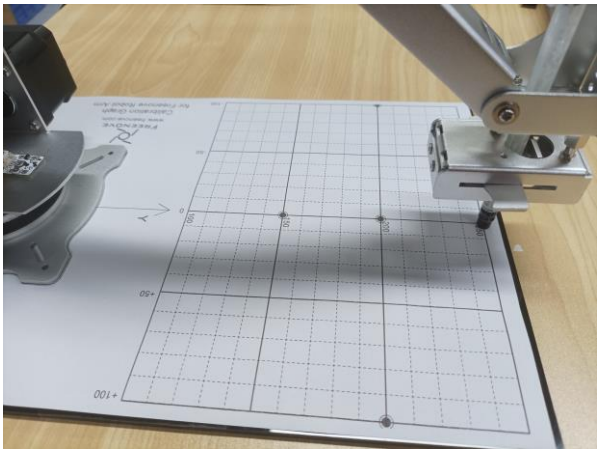
P2



P3

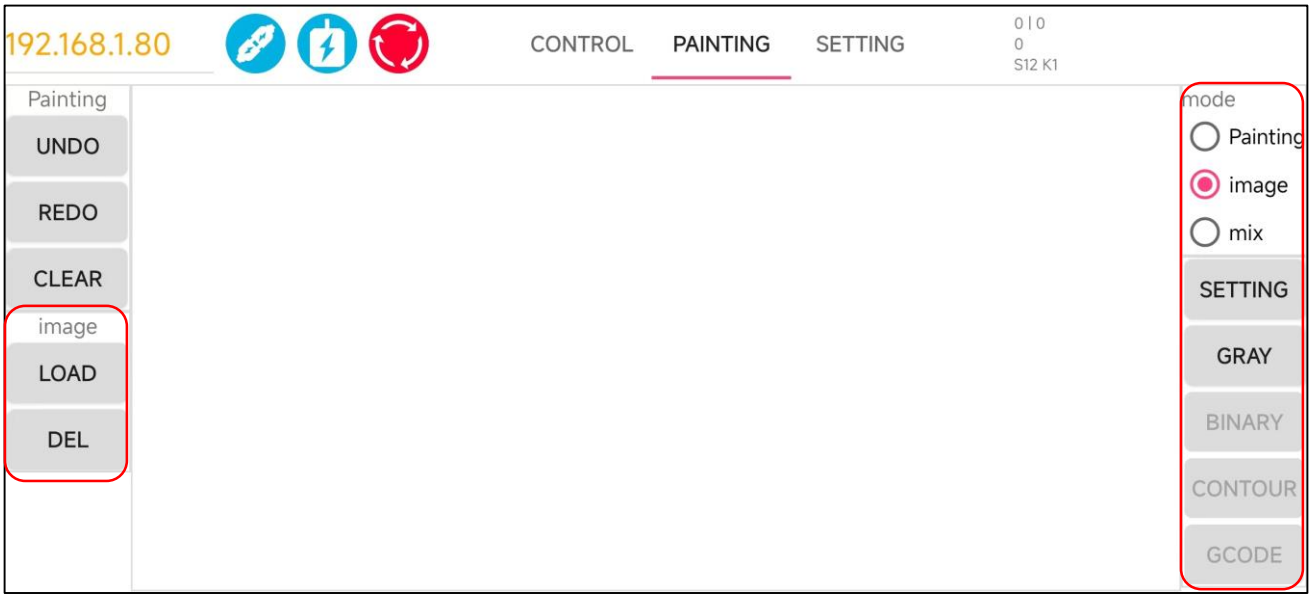


P4



After finishing calibration, tap on Painting to start drawing.

The buttons for painiting are as shown below.



Freehand Mode	In this mode, you can freely draw any content on the app.
Input: Painting → Canvas: Painting → Output: Gcode	
Image Mode	By choosing this mode, you have the option to import any image into the app.
Input: Image → Canvas: Image → Output: Gcode	
Mixed Mode	Selecting this mode allows you to import an image while drawing additional content directly on the interface
Input: Mix → Canvas: Painting + Image → Output: Gcode	

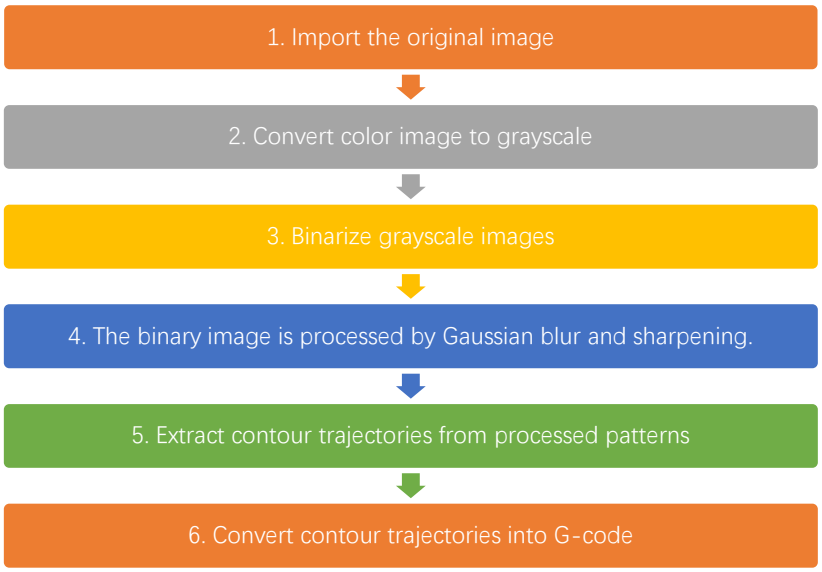
Here we take the Image Mode as an example.

1. Select the Image Mode.

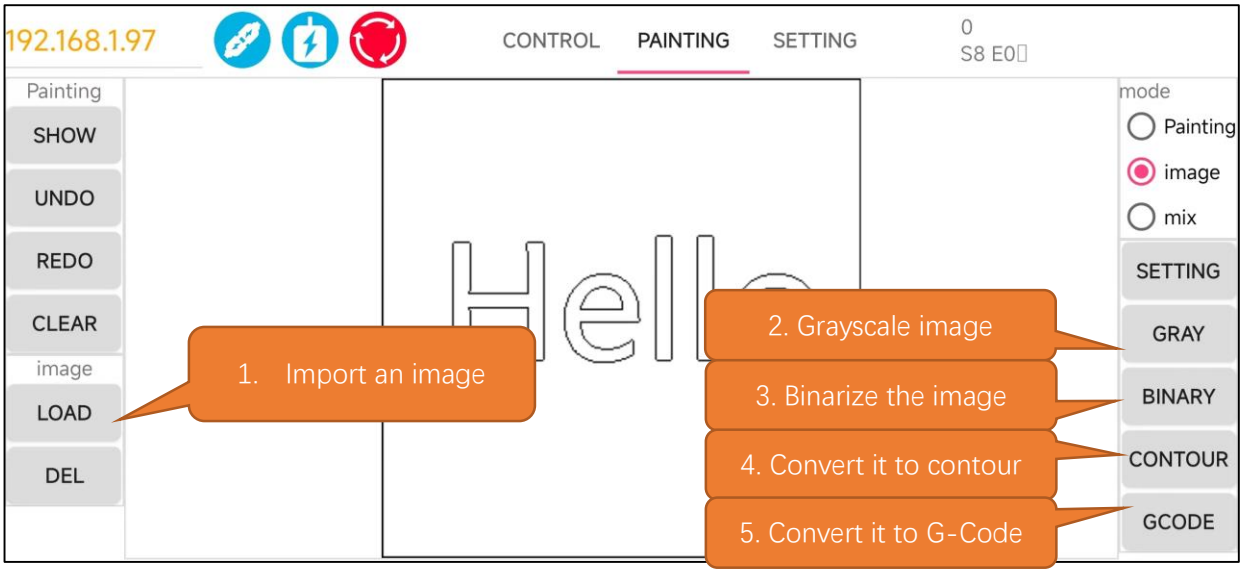
2. Tap the Load button on the left to open your local folder and browse through your files. From there, you can choose the desired image to load into the app. For demonstration purposes, let's select an image named "Hello."

3. If at any point you need to remove the image from the display area, you can use the Del button to delete the image, clearing the workspace for a new image or a different project.

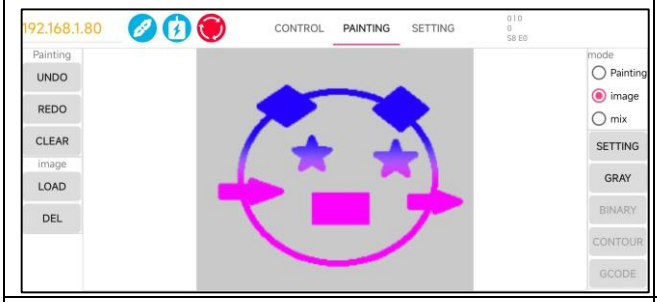
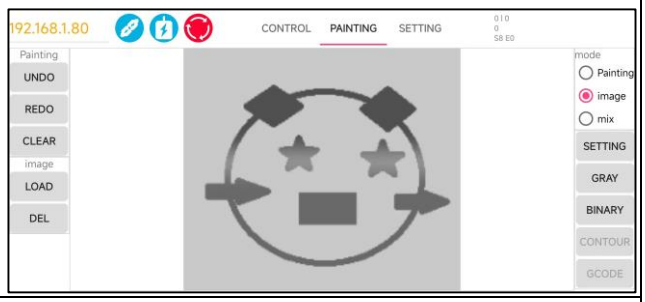
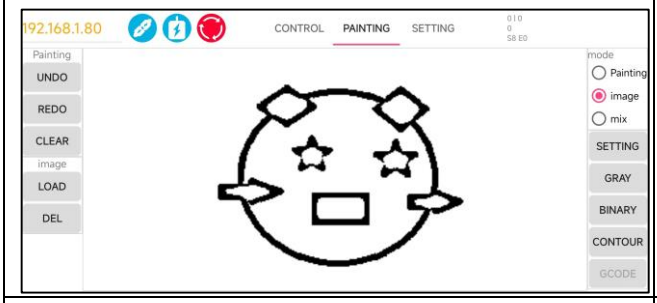
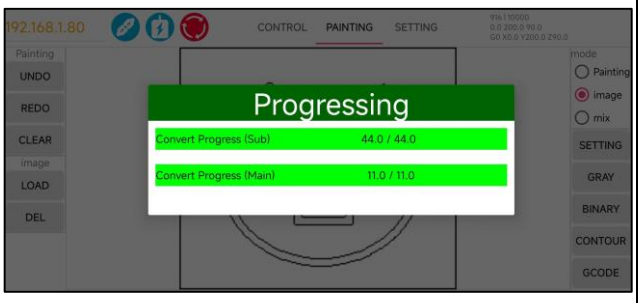
The original image cannot be used directly by the robotic arm. It requires a series of data processing and transformations to be converted into a format that the arm can utilize, as illustrated in the figure below.



Please operate on the app according to the sequence.



The effect of image processing is shown in the figure below.

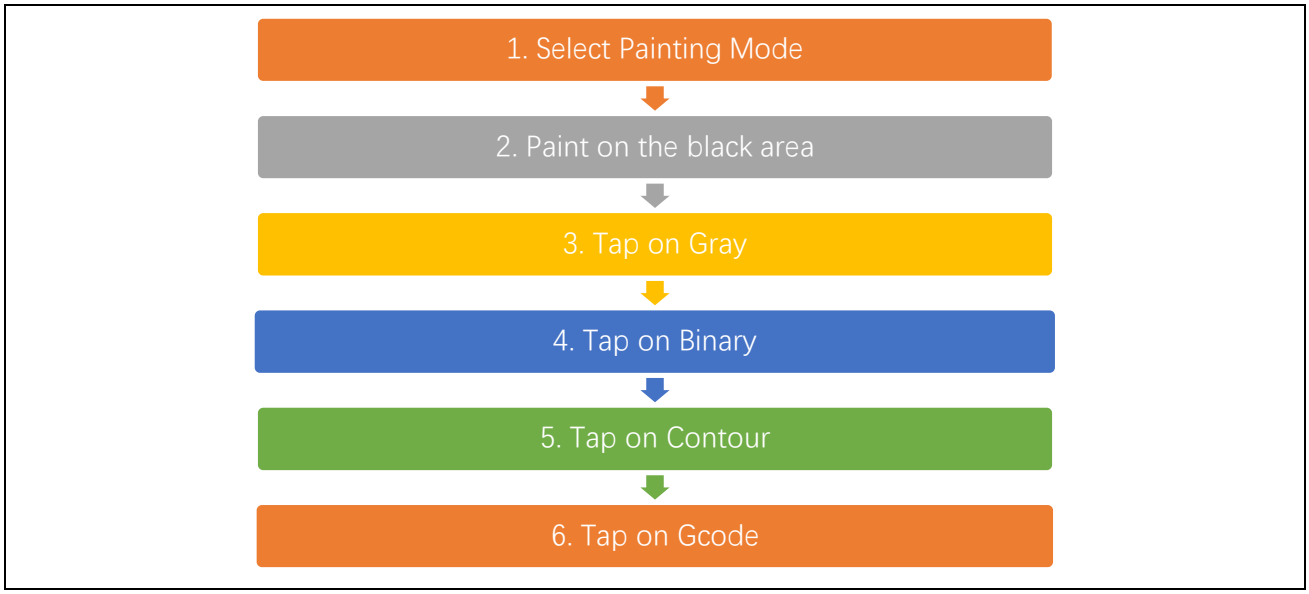
1. Import an image 	2. Convert the image to grayscale image 
3. Binarize the grayscale image 	4. Convert the binary image to contour image 
5. Click Gcode 	Notes: 1. When you click Contour, you may sometimes encounter display abnormalities. Please click it again until it resolves a normal image. 2. Before clicking Gcode, please make sure that your robotic arm has executed Sensor Point before. If not, please go to the Control interface and click Sensor Point once to update the coordinate position of the robotic arm to avoid operation errors.

Please take note of the following when using the image modes within the app:

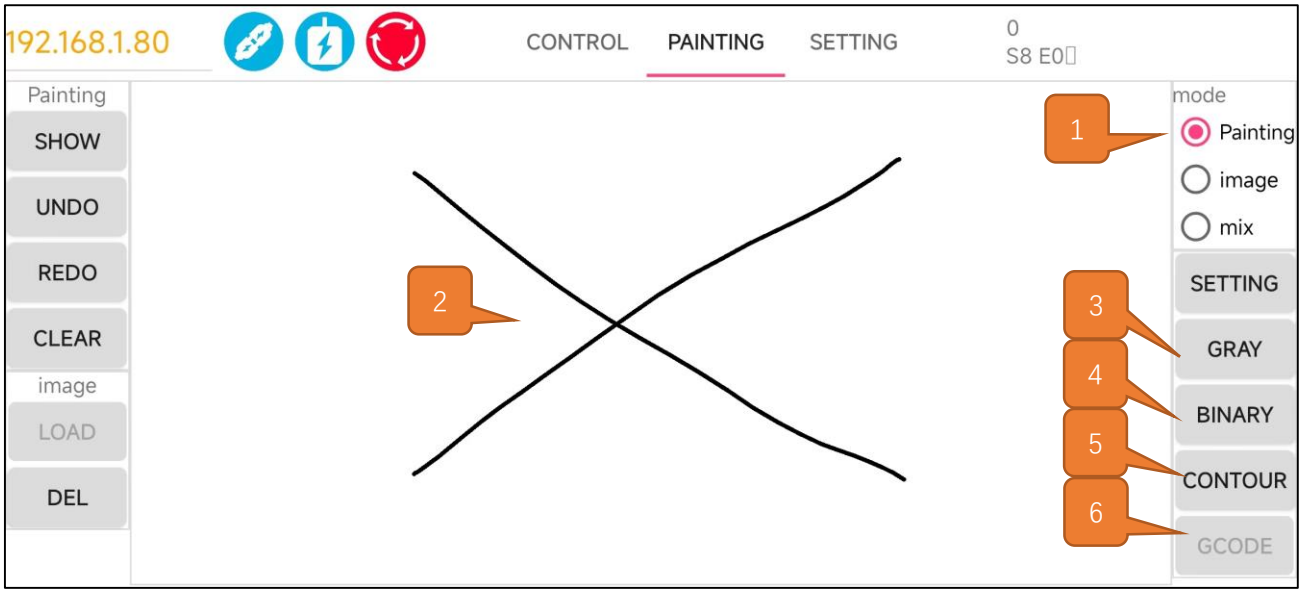
1. Image Mode: The app will automatically resize the imported image to fit the screen size of your mobile device.

2. Mixed Mode: In this mode, the app overlays the "Painting" layer on top of the "Image" layer. The "Image" layer will be stretched to match the size of the "Painting" layer. This may result in the image appearing distorted, as shown in the figure below. This distortion is a normal occurrence due to the layering and resizing process.

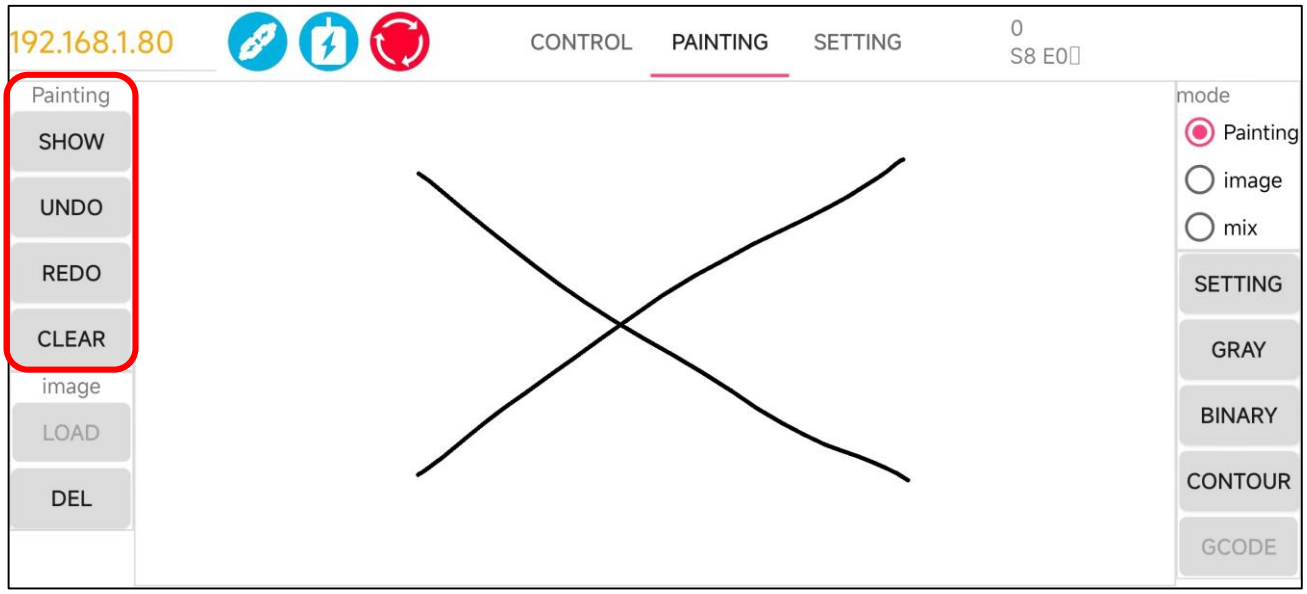
You can also select the Painting mode to paint as you like on the black area.



The operation of Pating Mode is similar to that of Image Mode. The process of Painting mode is as show below.



As illustrated below, there are four button on the left of the canvas.



Button Name	Functions
Show	**Show the original contents of Pating.**
Undo	**Undo the last action**
Redo	**Restore the previous step**
Clear	**Clear the original Painting content**

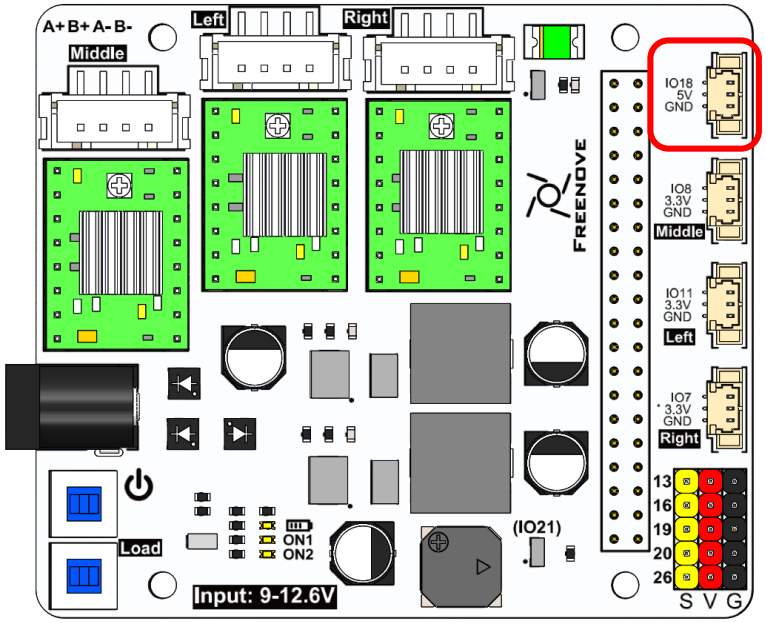
If the canvas content is modified after each click on "Show," please re-click "Gray -> Binary -> Contour."

Please note that if your robotic arm has not performed Sensor Point operations after connecting to WiFi and enabling the motors, clicking on "Gcode" will not cause the robotic arm to execute the corresponding actions. Instead, it will trigger a beeper alert. This is because the robotic arm has not undergone coordinate calibration, and at this moment, it is unaware of precise position of its end effector.

LED Module Control

There is an LED module assembled on the robot arm, which can also be controlled via the software. First and foremost, please ensure that your RGB LED module is properly connected to the Robot Arm Board. Pay close attention to the wiring sequence on the LED module before installing it; incorrect wiring may result in the lights not illuminating.

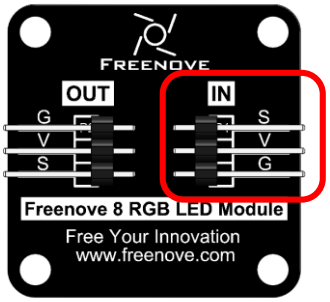
Connect the LED module to the robot arm board with a 10cm 3Pin cable to Jumper wire.



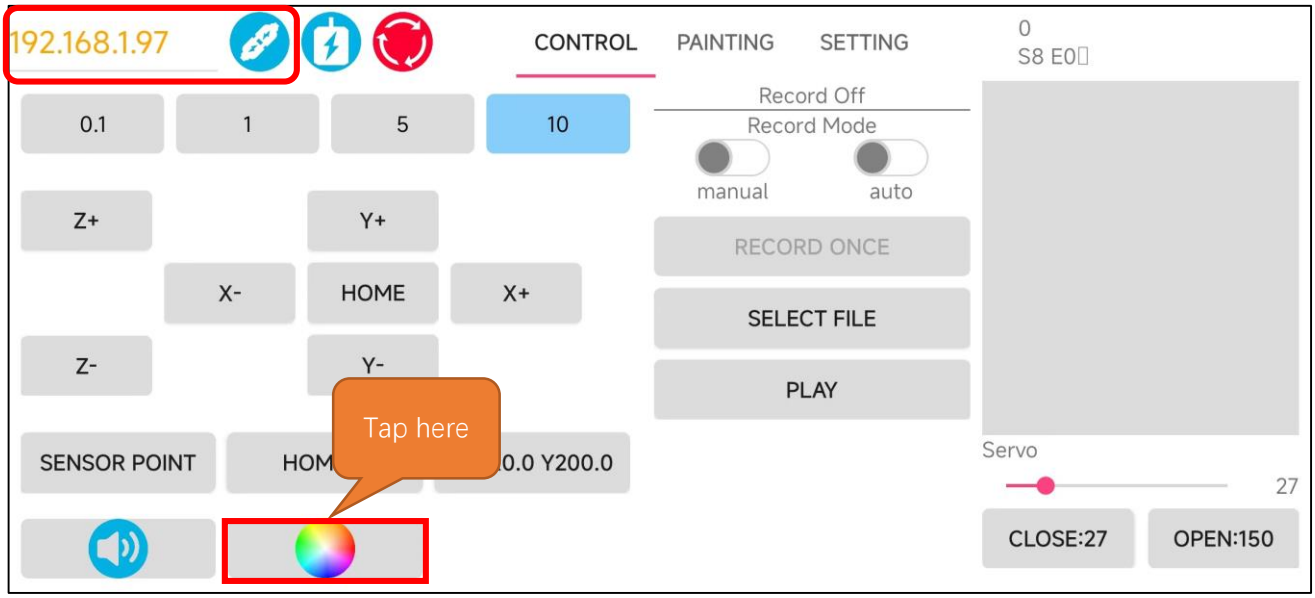
Wiring Sequence

Board	LedPixel
IO18	S
5V	V
GND	G

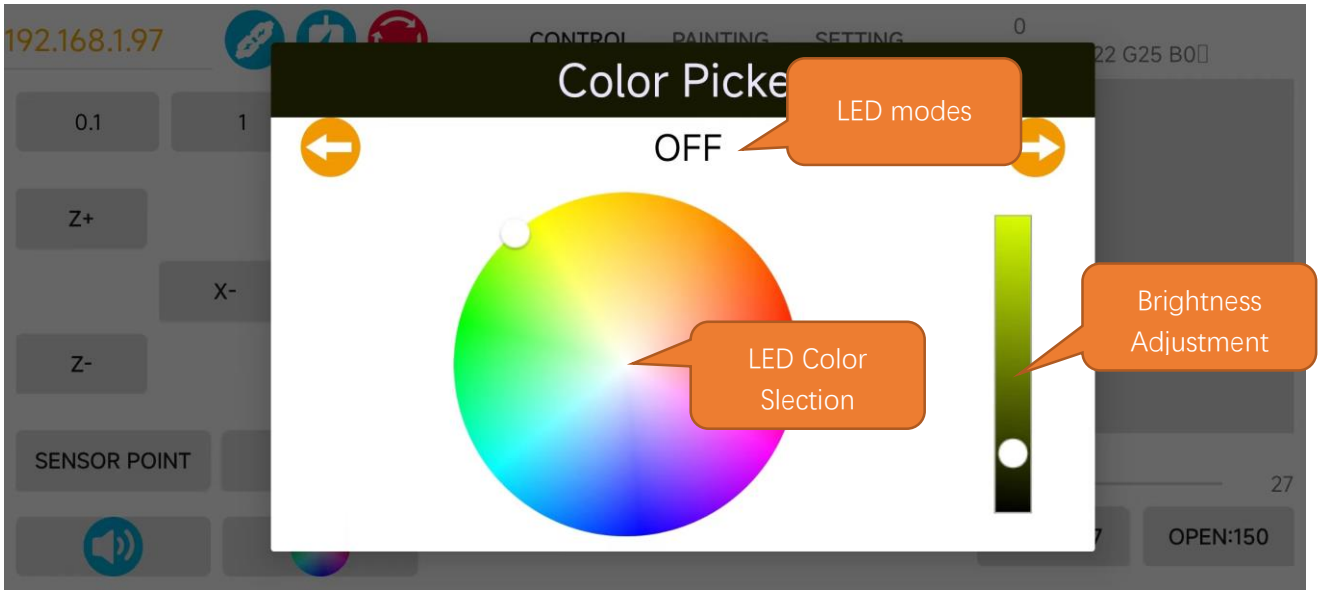
LedPixel



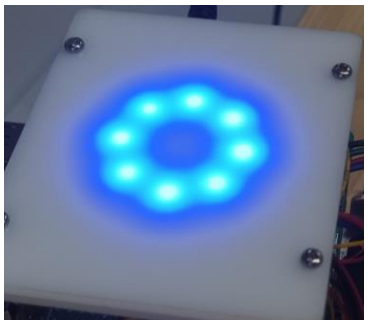
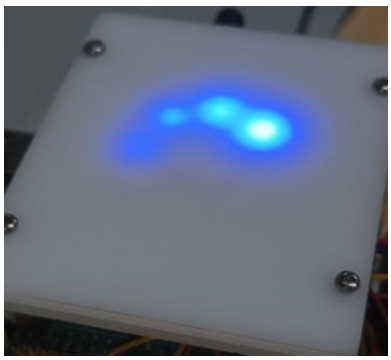
Open the Freenove APP, connect to WiFi and click the LED button.



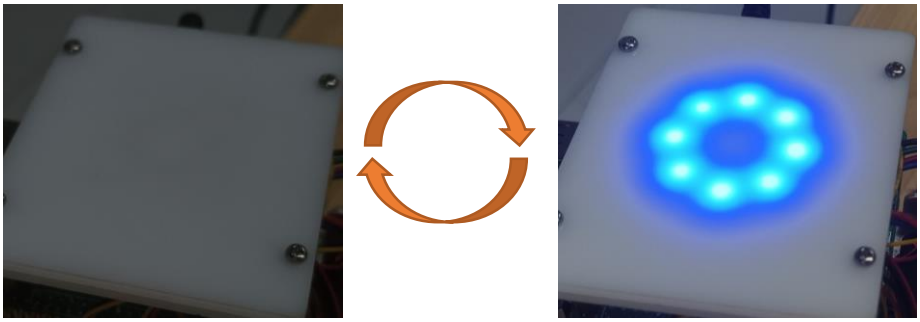
The interface is as below.



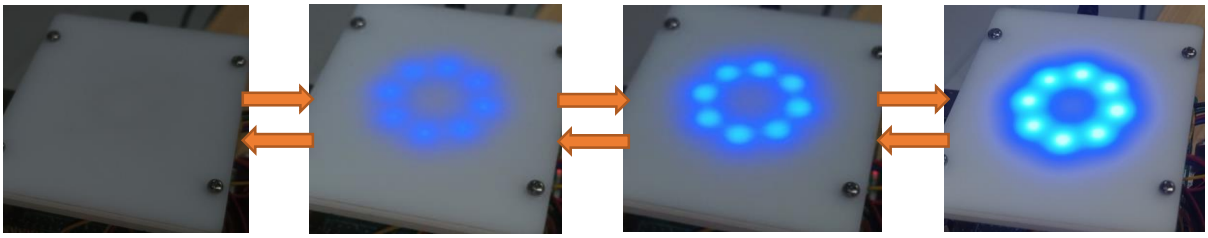
You can switch between the LED modes of the LED modes using the left and right arrow keys. Select the color for the lights using the color palette, and adjust the brightness of the lights with the slider. The RGB lights have six different display modes.

RGB Mode: In this mode, all the RGB lights will display a single color. You can change the color and brightness of the lights by using the color palette and sliders. 
Following Mode: The RGB lights will continuously. You can use the color palette and sliders to change their color and brightness. 

Blink Mode: All lights continuously blink. You can use the color palette and sliders to change their color and brightness.

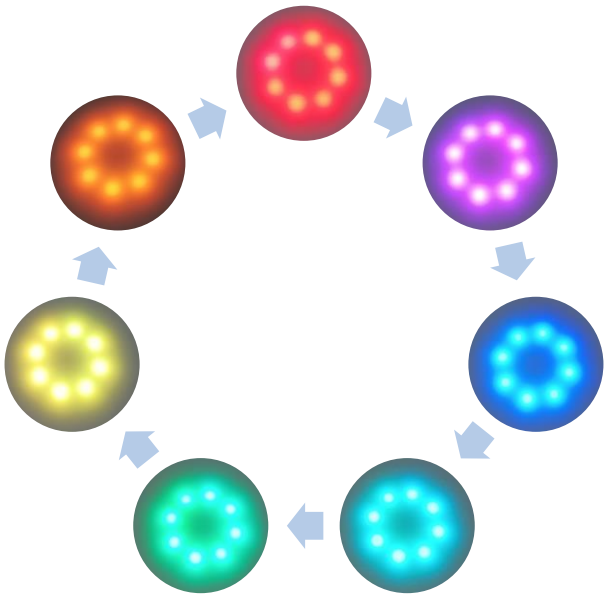


Breathing Mode: All lights gradually turn from dim to bright and then from bright to dim, like they are breathing. You can use the color palette and sliders to change their color and brightness.



Rainbow Mode: The whole LED module emits colors like rainbow and rotate slowly. In this mode, the color and brightness cannot be changed.

Gradient Mode: The whole LED module emits a single color and slowly change into other colors. In this mode, the color and brightness cannot be changed.



Chapter 6 Communication Instructions

Formats of Communication Instructions

In the operation of a robotic arm, each command is constructed using a series of command **characters** and their corresponding **parameters**. These command characters are delineated by space characters (" ") to distinguish one from another. To signify the conclusion of an instruction, a combination of a carriage return and a line feed ("\r\n") is employed.

Instruction Explanation	Instructions
Control the robot arm to return to its sensor point.	S10 F1\r\n
Move the end effector of the robot arm to the coordinates (0, 200, 75).	G0 X0.0 Y200.0 Z75.0\r\n
Set the RGB light to display in mode 1 with the color value (0, 255, 0).	S1 M1 R0 G255 B0\r\n

Instructions for Movements

The moving instructions consist of two commands, as shown below.

Instruction Explanations	Instructions
Move the robot arm to the target position in 3D space.	G0 X0.0 Y200.0 Z75.0\r\n
Pause the robot arm's movement for x milliseconds.	G4 T150\r\n

G0 is the instruction to move the robot arm, where XYZ corresponds to the coordinates in the coordinate system.

G4 is used to introduce a delay in the robot arm's operation, with T150 indicating a pause of 150 milliseconds.

Customized Instructions

LED Module Instructions

S1 indicates the instructions for the LED module. There are 7 modes: OFF, RGB, Following, Blink, Breathing, Rainbow, and Gradient. The last three parameters indicate the red, green and blue color values, ranging from 0 to 255.

Instruction Explanations	Instructions
Set the RGB light to display in Mode 1 with the color green (0,255,0).	S1 M1 R0 G255 B0\r\n
Set the RGB light to display in Mode 2 with the color red (255,0,0).	S1 M2 R255 G0 B0\r\n
Set the RGB light to display in Mode 3 with the color blue (0,0,255).	S1 M3 R0 G0 B255\r\n

Buzzer Instructions

S2 indicates an instruction related to the buzzer. The buzzer used in this robot is a passive buzzer, which operates within a frequency range of 1-4 kHz.

Instruction Explanations	Instructions
Make the buzzer emit a sound at a frequency of 2 kHz.	S2 D2000
Turn off the buzzer's sound emission.	S2 D0

the Ground Clearance Setting Instructions

S3 is used to set the height of the robotic arm's base from the ground. This value is set to the height of the padding only when the arm's base is elevated. Typically, when the robotic arm is placed and operates on a horizontal surface, this value is set to 0mm.

Instruction Explanations	Instructions
Set the ground clearance of the robot arm to 0.0mm.	S3 O0.0\r\n

End Effector Length Setting Instructions

S4 is used to set the length of the end effector of the robot arm. Our robotic arm is equipped with two types of grippers and has different mounting methods. Therefore, it is necessary to configure the gripper's length value according to the actual situation.

Instruction Explanations	Instructions
Set the gripper length to 45.0mm	S4 L45.0\r\n

Home Position Setting Instructions

S5 is used to set the coordinates of the Home point at the end of the robot arm. XYZ corresponds to the coordinate position in the coordinate system.

Instruction Explanations	Instructions
Set the Home position of the robotic arm's end effector.	S5 X0.0 Y200.0 Z75.0\r\n

Frequency Setting Instructions

S6 indicates the instructions for pulse frequency setting. The range of pulse frenqency of this robot arm is 100-16000, and it is set as 1000 by default.

Instruction Explanations	Instructions
1000Hz Set the pulse frequency for the robot arm's stepper motors	S6 Q1000\r\n
App/Software → Robot Arm	

Stepper Motor Microstep Setting Instructions

S7 is used to set the microstep resolution of the robotic arm's stepper motor. When the resolution is set to 1, the stepper motor has the maximum torque but the lowest precision. Conversely, when the resolution is set to 5, the stepper motor has the minimum torque but the highest precision. The default resolution is 5.

Instruction Explanations	Instructions
Set the resolution to 5	S7 W5\r\n
App/Software → Robot Arm	

Motor Enable/Diasble Instructions

S8 is used to control the power supply to the stepper motor. When powered, the motor generates torque, locking the motor in place. When the power is cut, the motor loses torque, allowing it to rotate freely.

Instruction Explanations	Instructions
Enable the stepper motor.	S8 E0\r\n
Disable the stepper motor.	S8 E1\r\n
App/Software → Robot Arm	

Servo Control Instructions

S9 is used to control the movement of servos. "Ix" represents the servo index number, which ranges from 0 to 4. "Ax" denotes the angle to which the servo should rotate.

Instruction Explanations	Instructions
Control Servo 0 to rotate to 90 degrees	S9 I0 A90\r\n
Control Servo 1 to rotate to 150 degrees	S9 I1 A150\r\n
App/Software → Robot Arm	

Sensor Calibration Instrutions

S10 is employed to re-establish the reference coordinate system for the robot arm. Whenever there is an instance of step loss or when a previously disabled motor is reactivated, it is imperative to initiate a sensor calibration command first.

Instruction Explanations	Instructions
Recalibrate the sensor center position of the robot arm.	S10 F0\r\n
Move the robot arm to the center positon of the sensors.	S10 F1\r\n
App/Software → Robot Arm	

Coordinates Calibration Instructions

S11 command is designed to assist in calibrating the robotic arm, enhancing its precision and accuracy during operation. There are three calibration commands in total. The robot arm will only perform calibration at a specific point upon receiving a start signal for calibration. If the start signal is not received, the process signal command and the end signal command will have no effect.

This command allows for the calibration of up to five points. The 0^{th} coordinate represents the Home position of the robot arm, while coordinates 1 through 4 correspond to the four points listed on the calibration paper.

Instruction Explanations	Instructions
Begin the calibration of the 0^{th} point	S11 C0\r\n
Control the robot arm to move to the correct positon after calibration	S11 H0 X0.0 Y200.0 Z75.0\r\n
Finish the calibration of the 0^{th} point	S11 J0 X0.0 Y200.0 Z75.0\r\n
App/Software → Robot Arm	

Instructions for Inquring the Motion Command Queue

S12 is utilized to check the number of remaining motion commands that have not yet been executed within the robot arm's control system.

Instruction Explanations	Instructions
Start the feedback thread for robot arm motion command inquiry.	S12 K1\r\n
Stop the feedback thread for robot arm motion command inquiry.	S11 K0\r\n
App/Software → Robot Arm	
The robotic arm sends a feedback to the App/Software every 0.5 seconds, reporting the current number of remaining motion commands (where 'x' represents the current number of remaining commands).	S12 Kx\r\n
Robot Arm → App/Software	

Emergency Stop Insctructions

S13 is used to shut down the robot arm's threads and motors. It is typically employed in situations where the robotic arm exhibits abnormal motion, such as when it becomes stuck but continues to operate. This scenario can easily lead to damage to the robotic arm. Utilizing this command allows for an emergency stop of the robotic arm and releases the motor torque, thereby protecting the arm from harm.

Instruction Explanations	Instructions
Force shutdown of robot arm threads and motor current	S13 N1\r\n
App/Software → Robot Arm	

Chapter 7 Core Code Introduction

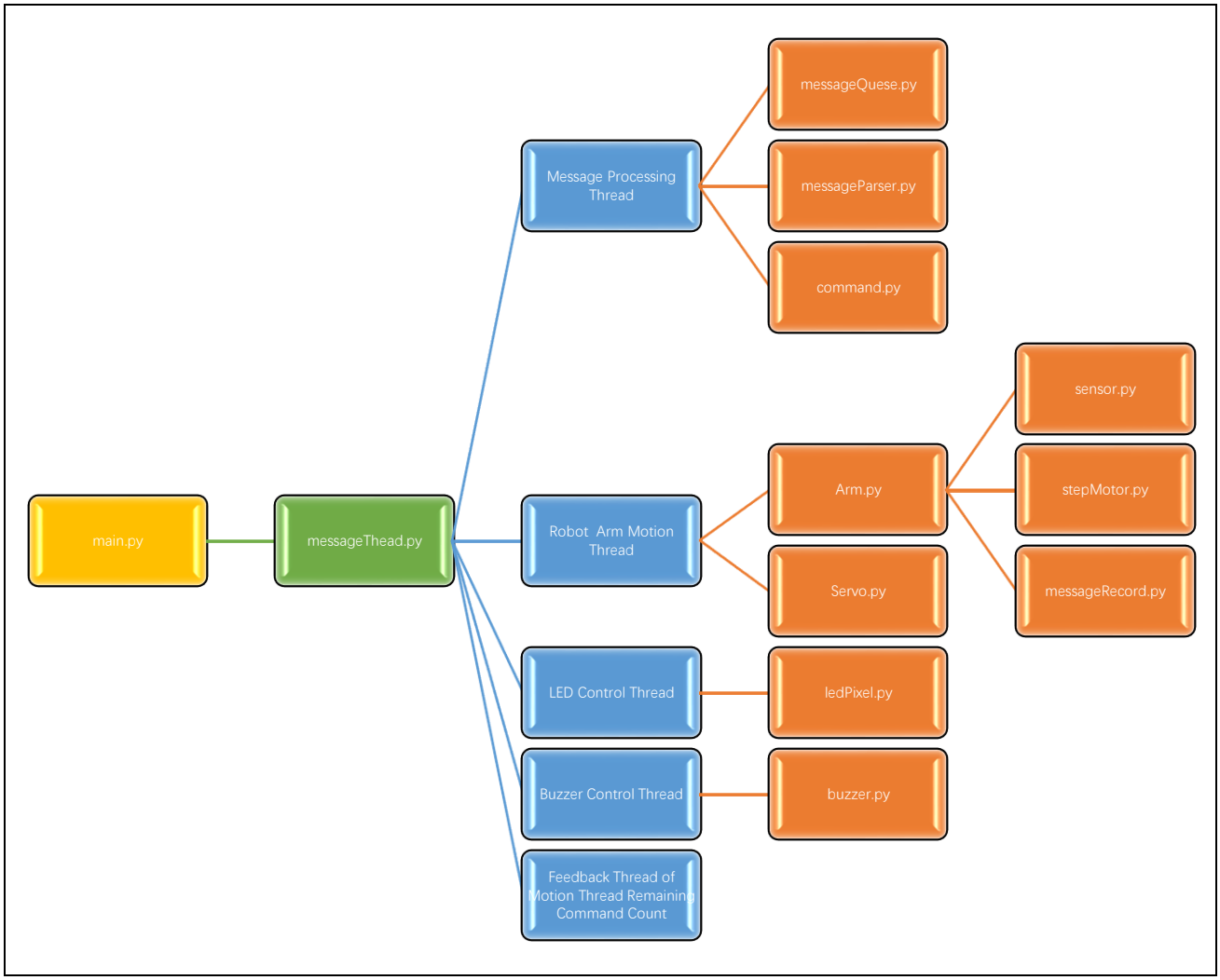
Overall Flowchart

Upon executing the main.py script located in the **Freenove_Robot_Arm_Kit_for_Raspberry_Pi/Server/Code** directory, a server thread is launched by the main.py through the messageThread.py module. This thread sets up a network server using the socket to monitor for and accept connections from devices intending to interface with the robot arm.

Subsequently, the server thread generates four additional threads to handle different operations. The first of these is the message processing thread to process commands from the App/Software. This thread employs the messageQueue.py and messageParser.py modules to efficiently parse these commands. Once a command has been deciphered, it is directed to the appropriate processing thread for further action.

The remaining three threads are assigned to specific functions: the robot arm motion thread, the LED control thread, and the buzzer control thread. Each thread is dedicated to processing events related to its assigned domain.

The sequence of operations for the robot arm is outlined below:



Code Introduction

Next, we will explain the code files one by one.

messageThread.py

We use this file to create and destroy thread objects. The code is as shown below.

```
import threading
import time
import inspect
import ctypes


def _async_raise(tid, exctype):
    """raises the exception, performs cleanup if needed"""
    tid = ctypes.c_long(tid)
    if not inspect.isclass(exctype):
        exctype = type(exctype)
    res = ctypes.pythonapi.PyThreadState_SetAsyncExc(tid, ctypes.py_object(exctype))
    if res == 0:
        raise ValueError("invalid thread id")
    elif res != 1:
        ctypes.pythonapi.PyThreadState_SetAsyncExc(tid, None)
        raise SystemError("PyThreadState_SetAsyncExc failed")

def create_thread(thread):
    return threading.Thread(target=thread)

def stop_thread(thread):
    for i in range(5):
        _async_raise(thread.ident, SystemExit)

def test():
    while True:
        print('-------')
        time.sleep(1)

if __name__ == "__main__":
    t = create_thread(test)
    t.start()
    time.sleep(5)
    print("main thread sleep finish")
    stop_thread(t)
```

Import function modules.

```
1  import threading
2  import time
3  import inspect
4  import ctypes
```

Create a function called create_thread that accepts a parameter thread and returns a threading.Thread object. It is used to create thread objects.

```
19 def create_thread(thread):
20     return threading.Thread(target=thread)
```

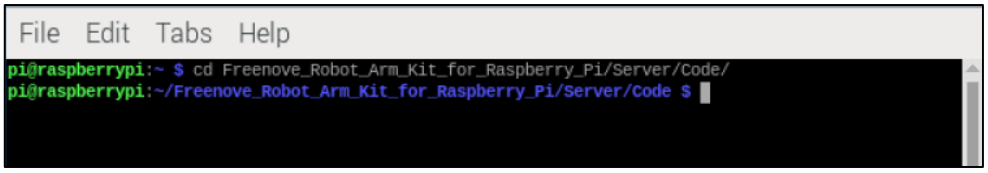
Create a function called stop_thread that accepts one parameter thread. It is used to close threads.

```
22 def stop_thread(thread):
23     for i in range(5):
24         _async_raise(thread.ident, SystemExit)
```

In the code, we create a test function to test whether the above code runs normally. We let it print a prompt message every 1 second.

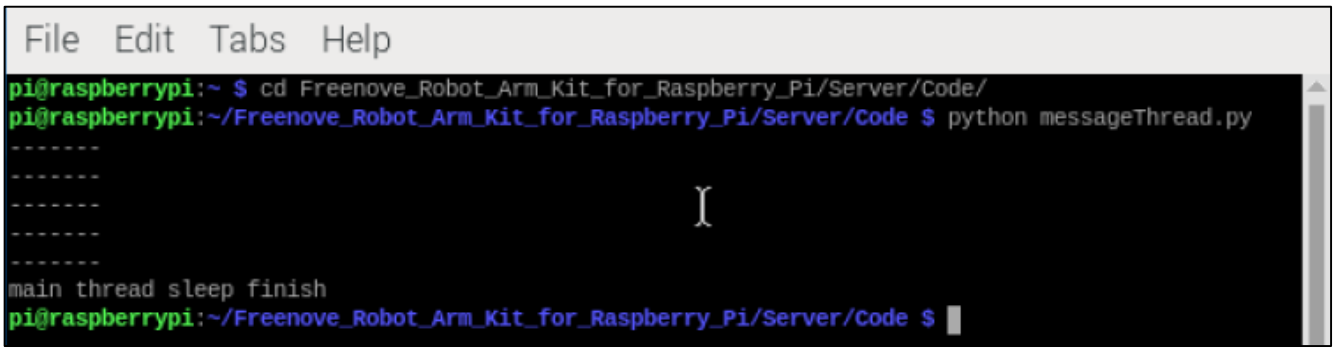
```
26 def test():
27     while True:
28         print('-------')
29         time.sleep(1)
```

Open a terminal and jump to the directory where the messageThread.py file is located.



Enter python messageThread.py and it will run the code below. In the code, the main process first creates a sub-thread test. This sub-thread prints a prompt message every 1 second. After 5 seconds, the main process closes the sub-thread test.

```
31 if __name__ == "__main__":
32     t = create_thread(test)
33     t.start()
34     time.sleep(5)
35     print("main thread sleep finish")
36     stop_thread(t)
```



messageQueue.py

This file is applied to access data. Each time the robot arm receives data, it calls this file to access the data in a unified format and execute the instructions one by one to make the system more stable. The code is as below.

```
1  class MessageQueue:
2      def __init__(self):
3          self.items = []
4
5      def put(self, item):
6          self.items.append(item) #Data is sent to the message queue
7
8      def get(self):
9          if self.empty():
10             return None
11         return self.items.pop(0) #Returns the first data
12
13     def gets(self):
14         if self.empty():
15             return None
16         return self.items
17
18     def delete(self, index):    #Delete the data corresponding to the index number
19         if self.empty() == False:
20             del self.items[index]
21
22     def len(self):
23         return len(self.items)   #Returns the length of the message queue
24
25     def empty(self):
26         return self.len() == 0   #Return True if the message queue is empty
27
28     def clear(self):
29         self.items.clear()       #Clear message queue
30
31     def end(self):
32         data = self.items[-1]    #Gets data at the end of the message queue
33         return data
34
35 if __name__ == '__main__':
36     myQueue = MessageQueue()              #Request a message queue object
37     myQueue.clear()                    #Clear the message queue
38     for i in range(10):
39         myQueue.put(str(i))              #Feed the message queue 0-9
```

```
40      while myQueue.empty() is not True:
41          print(myQueue.get())            #Get the contents of the message queue and print it
42      print(myQueue.empty())
```

Write a message queue object, which is implemented through the list variable self.itens.

```
1   class MessageQueue:
2       def __init__(self):
3           self.items = []
```

Add a piece of data to the message queue.

```
5       def put(self, item):
6           self.items.append(item) #Data is sent to the message queue
```

Use the get() function to get a data from the message queue. Use the gets() function to get all data from the message queue.

```
8       def get(self):
9           if self.empty():
10              return None
11          return self.items.pop(0) #Returns the first data
12
13      def gets(self):
14          if self.empty():
15              return None
16          return self.items
```

Delete the data corresponding to the specified index number in the message queue.

```
18      def delete(self, index):    #Delete the data corresponding to the index number
19          if self.empty() == False:
20              del self.items[index]
```

Get the number of data in the message queue and return it.

```
22      def len(self):
23          return len(self.items)   #Returns the length of the message queue
```

Determine whether the message queue is empty.

```
25      def empty(self):
26          return self.len() == 0   #Return True if the message queue is empty
```

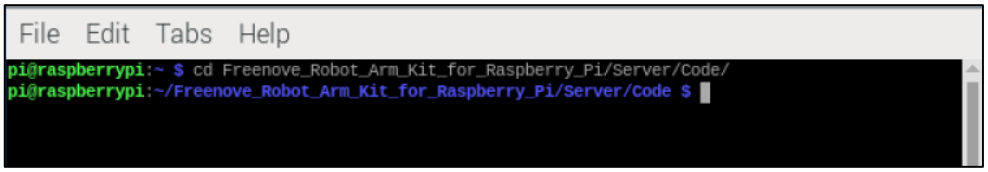
Clear all data in the message queue.

```
28      def clear(self):
29          self.items.clear()      #Clear message queue
```

Get the last data in the message queue.

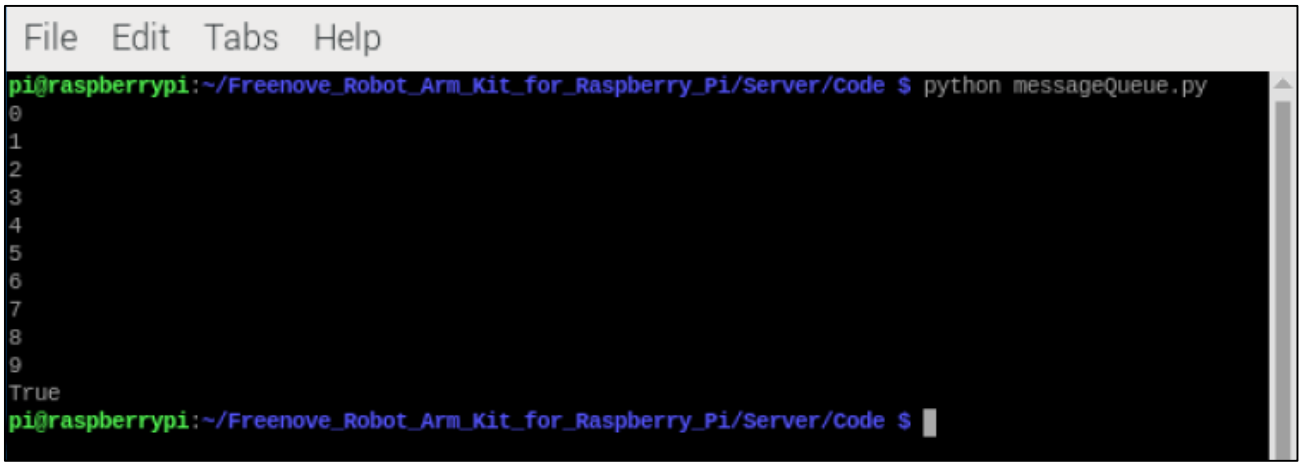
```
31      def end(self):
32          data = self.items[-1]   #Gets data at the end of the message queue
33          return data
```

Open a terminal and jump to the directory where the messageQuese.py file is located.



Enter python messageQuese.py and it will run the code below.

```
35  if __name__ == '__main__':
36      myQueue = MessageQueue()            #Request a message queue object
37      myQueue.clear()                     #Clear the message queue
38      for i in range(10):
39          myQueue.put(str(i))             #Feed the message queue 0-9
40      while myQueue.empty() is not True:
41          print(myQueue.get())            #Get the contents of the message queue and print it
42      print(myQueue.empty())
```



messageParser.py

file is used to parse commands from an app or software. The code is as follows.

```
1   class MessageParser:
2       def __init__(self):
3           self.inputCommandArray = []
4           self.commandArray = []
5           self.stringParameter = []
6           self.floatParameter = []
7           self.intParameter = []
8           self.commandChar = None
9
10      def parser(self, msg):
11          self.clearParameters()                  #Clear the command array
12          inputStringTemp = msg.strip()           #Delete beginning and end Spaces
13          self.inputCommandArray = inputStringTemp.split(" ")#Separate instructions by space and
    store them in the list
14          for i in self.inputCommandArray:        #Parses the list data and stores it in different lists
15              self.commandArray.append(i[0:1])
16              self.stringParameter.append(i[1:])
17          self.commandChar = self.commandArray[0][0:1]  #Takes the first byte of the command
18          if self.commandChar == "G" or self.commandChar == "S":
```

```
            self.floatParameter = [float(x) for x in self.stringParameter]  #Convert string to float type
            self.intParameter = [round(x) for x in self.floatParameter]     #Convert float to int type
        else:
            print("messageParser.py, error command.")

    def clearParameters(self):
        self.inputCommandArray.clear()
        self.commandArray.clear()
        self.stringParameter.clear()
        self.floatParameter.clear()
        self.intParameter.clear()
        self.commandChar = None

if __name__ == '__main__':
    msg = MessageParser()
    msg.parser("G1.0 X1.0 Y2.3 Z4.55")
    print(msg.commandArray)
    print(msg.stringParameter)
    print(msg.floatParameter)
    print(msg.intParameter)
    print(msg.commandChar)
```

Write a message parsing object specifically used to parse app/software instructions.

```
class MessageParser:
    def __init__(self):
        self.inputCommandArray = []
        self.commandArray = []
        self.stringParameter = []
        self.floatParameter = []
        self.intParameter = []
        self.commandChar = None
```

The instruction parsing function sends the content that needs to be parsed into the function through parameters. After parsing the instruction, the content in different formats is stored in different variables.

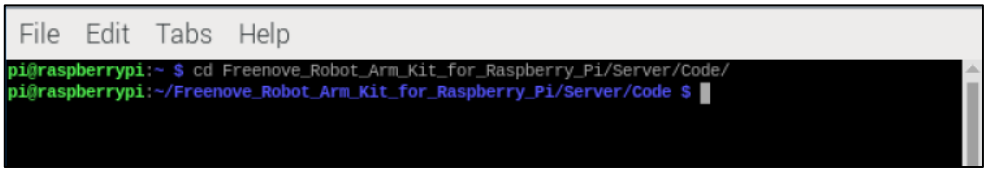
```
    def parser(self, msg):
        self.clearParameters()                  #Clear the command array
        inputStringTemp = msg.strip()           #Delete beginning and end Spaces
        self.inputCommandArray = inputStringTemp.split(" ")#Separate instructions by space and
store them in the list
        for i in self.inputCommandArray:        #Parses the list data and stores it in different lists
            self.commandArray.append(i[0:1])
            self.stringParameter.append(i[1:])
        self.commandChar = self.commandArray[0][0:1]  #Takes the first byte of the command
        if self.commandChar == "G" or self.commandChar == "S":
            self.floatParameter = [float(x) for x in self.stringParameter]  #Convert string to float type
            self.intParameter = [round(x) for x in self.floatParameter]     #Convert float to int type
```

```
21          else:
22              print("messageParser.py, error command.")
```

Clear parsed content.

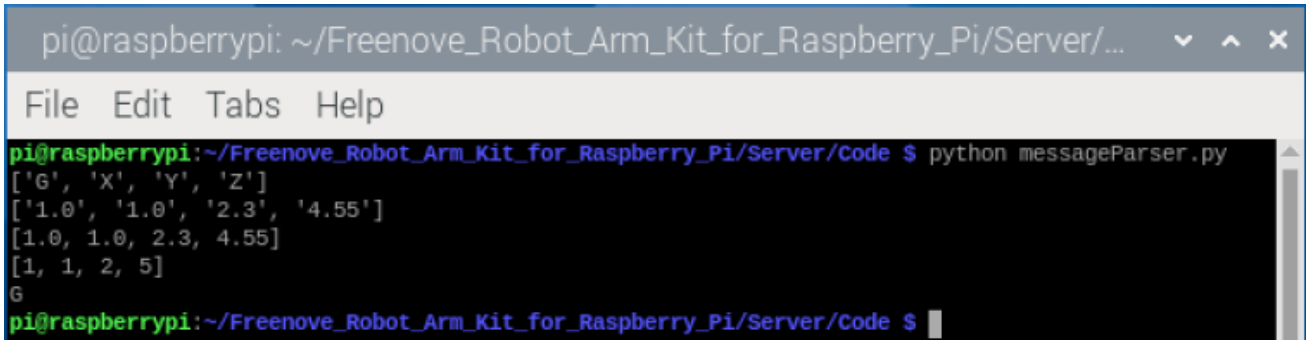
```
24      def clearParameters(self):
25          self.inputCommandArray.clear()
26          self.commandArray.clear()
27          self.stringParameter.clear()
28          self.floatParameter.clear()
29          self.intParameter.clear()
30          self.commandChar = None
```

Open a terminal and jump to the directory where the messageParser.py file is located.



Enter python messageParser.py and it will run the code below.

```
32  if __name__ == '__main__':
33      msg = MessageParser()
34      msg.parser("G1.0 X1.0 Y2.3 Z4.55")
35      print(msg.commandArray)
36      print(msg.stringParameter)
37      print(msg.floatParameter)
38      print(msg.intParameter)
39      print(msg.commandChar)
```



Command.py

This file is utilized for defining the specifications of communication content between the robot arm and the app/software.

```
class Command:
    def __init__(self):
        self.MOVE_ACTION =          'G'
        self.AXIS_X_ACTION =        'X'
        self.AXIS_Y_ACTION =        'Y'
        self.AXIS_Z_ACTION =        'Z'
        self.DELAY_T_ACTION =       'T'
        self.DECOLLATOR_CHAR =      ' '

        self.CUSTOM_ACTION =        'S'
        self.WS2812_MODE =          'M'
        self.WS2812_RED     =       'R'
        self.WS2812_GREEN   =       'G'
        self.WS2812_BLUE    =       'B'
        self.BUZZER_ACTION =        'D'
        self.GROUND_HEIGHT =        'O'
        self.CLAMP_LENGTH =         'L'
        self.ARM_FREQUENCY =        'Q'
        self.ARM_MSX =              'W'
        self.ARM_ENABLE =           'E'
        self.ARM_SERVO_INDEX =      'I'
        self.ARM_SERVO_ANGLE =      'A'
        self.ARM_SENSOR_POINT =     'F'
        self.ARM_CALIBRATION_START =    'C'
        self.ARM_CALIBRATION_POINT =    'H'
        self.ARM_CALIBRATION_END    =   'J'
        self.ARM_QUERY =            'K'
        self.ARM_STOP =             'N'
```

MessageRecord.py

This file is used to store messages to a local file. The code is as follows.

```
1   import json
2   import os
3
4   class MessageRecord:
5       # Format: 'String' : data (integer/floating point)
6       def __init__(self):
7           self.jsonData = {
...             ......
28          }
29          self.fileInit()
30
31      #Find if the target exists in the json
32      def findData(self, targetValue):
33          if targetValue in self.jsonData:
34              return True
35          else:
36              return False
37
38      #Delete a file in a specified path
39      def deleteFile(self, filePath):
40          try:
41              os.remove(filePath)
42          except OSError as e:
43              print(f"The file '{e}' was not successfully deleted.")
44
45      #Local json file initialization
46      def fileInit(self):
47          if os.path.exists("Parameter.json") is True:
48              with open("Parameter.json", "r", encoding='utf-8') as fp:
49                  data = json.load(fp)
50                  if len(data) == len(self.jsonData):
51                      self.readJsonFile()
52                  else:
53                      self.deleteFile("Parameter.json")
54                      self.writeJsonFile()
55          else:
56              self.writeJsonFile()
57
58      #Read the local json file and store it in self.jsonData
59      def readJsonFile(self):
60          with open("Parameter.json", "r", encoding='utf-8') as fp:
```

```
61              self.jsonData = json.load(fp)
62
63      #Transcode self.jsonData into json format and write it to a local json file
64      def writeJsonFile(self):
65          with open("Parameter.json", "w", encoding='utf-8') as fp:
66              json.dump(self.jsonData, fp, indent=4)
67
68      #Reads the message for the specified object
69      def readJsonObject(self, objectName):
70          try:
71              return self.jsonData[str(objectName)]
72          except:
73              print(f"Read the Json: '{objectName}' does not exist." )
74
75      #Writes the message for the specified object to the local json
76      def writeJsonObject(self, objectName, data):
77          try:
78              self.jsonData[str(objectName)] = data
79              self.writeJsonFile()
80          except:
81              print(f"Write the Json: '{objectName}' does not exist." )
82
83  if __name__ == '__main__':
...         ......
```

Import json and os objects.

```
1   import json
2   import os
```

。Write a local file storage object to store some important messages in local files.

```
4   class MessageRecord:
5       # Format: 'String' : data (integer/floating point)
6       def __init__(self):
7           self.jsonData = {
...             ......
28          }
29          self.fileInit()
```

Check if target object exists in json.

```
31      #Find if the target exists in the json
32      def findData(self, targetValue):
33          if targetValue in self.jsonData:
34              return True
35          else:
36              return False
```

Delete the designated file.

```
38      #Delete a file in a specified path
```

```
39      def deleteFile(self, filePath):
40          try:
41              os.remove(filePath)
42          except OSError as e:
43              print(f"The file '{e}' was not successfully deleted.")
```

Determine whether there is a json file in the local file, and whether the number in the json file is consistent with the number of self.jsonData. If they are consistent, read the json file content and assign it to self.jsonData. If the numbers are inconsistent or the json file does not exist, create a new json file and write the default parameters.

```
45      #Local json file initialization
46      def fileInit(self):
47          if os.path.exists("Parameter.json") is True:
48              with open("Parameter.json", "r", encoding='utf-8') as fp:
49                  data = json.load(fp)
50                  if len(data) == len(self.jsonData):
51                      self.readJsonFile()
52                  else:
53                      self.deleteFile("Parameter.json")
54                      self.writeJsonFile()
55          else:
56              self.writeJsonFile()
```

These two functions are used to read and write back and forth between json files and self.jsonData.

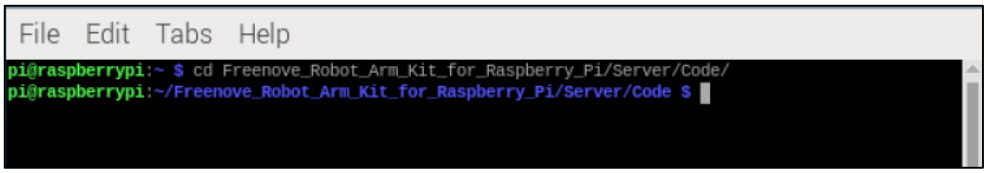
```
58      #Read the local json file and store it in self.jsonData
59      def readJsonFile(self):
60          with open("Parameter.json", "r", encoding='utf-8') as fp:
61              self.jsonData = json.load(fp)
62
63      #Transcode self.jsonData into json format and write it to a local json file
64      def writeJsonFile(self):
65          with open("Parameter.json", "w", encoding='utf-8') as fp:
66              json.dump(self.jsonData, fp, indent=4)
```

The readJsonObject function is used to read the parameters of the specified object from json. The writeJsonObject function is used to write the parameters of the specified object into json and update it to the local file.

```
68      #Reads the message for the specified object
69      def readJsonObject(self, objectName):
70          try:
71              return self.jsonData[str(objectName)]
72          except:
73              print(f"Read the Json: '{objectName}' does not exist." )
74
75      #Writes the message for the specified object to the local json
76      def writeJsonObject(self, objectName, data):
77          try:
```

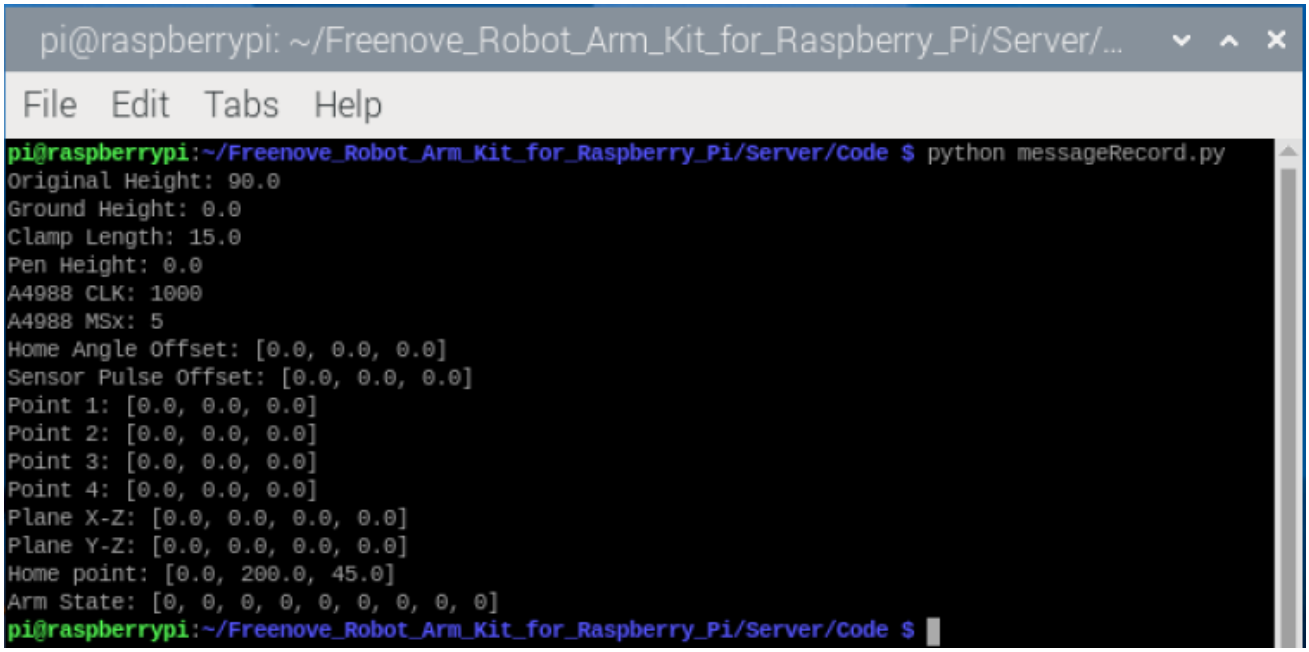
```
78              self.jsonData[str(objectName)] = data
79              self.writeJsonFile()
80          except:
81              print(f"Write the Json: '{objectName}' does not exist." )
```

Open a terminal and jump to the directory where the messageRecord.py file is located.



Run python messageRecord.py and it will run the following code.

```
83  if __name__ == '__main__':
...     ......
```



sensor.py

This file is used to obtain sensor status information. The code is as below.

```
import pigpio

class TCRT5000:
    def __init__(self):
        self.TCRT5000_PIN = [8, 11, 7]
        self.tcrt5000 = pigpio.pi()
        self.tcrt5000.set_mode(self.TCRT5000_PIN[0], pigpio.INPUT)
        self.tcrt5000.set_mode(self.TCRT5000_PIN[1], pigpio.INPUT)
        self.tcrt5000.set_mode(self.TCRT5000_PIN[2], pigpio.INPUT)
    def readTCRT5000S1(self):
        return self.tcrt5000.read(self.TCRT5000_PIN[0])
    def readTCRT5000S2(self):
        return self.tcrt5000.read(self.TCRT5000_PIN[1])
    def readTCRT5000S3(self):
        return self.tcrt5000.read(self.TCRT5000_PIN[2])
    def readTCRT5000ALL(self):
        s1 = self.readTCRT5000S1()
        s2 = self.readTCRT5000S2()
        s3 = self.readTCRT5000S3()
        return [s1,s2,s3]

if __name__ == '__main__':
    import os
    import time
    os.system("sudo pigpiod")
    time.sleep(1)
    current_state = [0,0,0]
    sensor_count = [0,0,0]
    sensor = TCRT5000()
    try:
        while True:
            sensor_state = sensor.readTCRT5000ALL()
            print("sensor_state:", sensor_state)
            if current_state[0] != sensor_state[0]:
                time.sleep(0.1)
                sensor_state = sensor.readTCRT5000ALL()
                if current_state[0] != sensor_state[0]:
                    current_state[0] = sensor_state[0]
                    sensor_count[0] = sensor_count[0] + 1
                if sensor_count[0] >= 2:
                    sensor_count[0] = 0
```

```
42                  print("Sensor 1 is triggered.")
43              elif current_state[1] != sensor_state[1]:
44                  time.sleep(0.1)
45                  sensor_state = sensor.readTCRT5000ALL()
46                  if current_state[1] != sensor_state[1]:
47                      current_state[1] = sensor_state[1]
48                      sensor_count[1] = sensor_count[1] + 1
49                  if sensor_count[1] >= 2:
50                      sensor_count[1] = 0
51                      print("Sensor 2 is triggered.")
52              elif current_state[2] != sensor_state[2]:
53                  time.sleep(0.1)
54                  sensor_state = sensor.readTCRT5000ALL()
55                  if current_state[2] != sensor_state[2]:
56                      current_state[2] = sensor_state[2]
57                      sensor_count[2] = sensor_count[2] + 1
58                  if sensor_count[2] >= 2:
59                      sensor_count[2] = 0
60                      print("Sensor 3 is triggered.")
61              else:
62                  time.sleep(0.3)
63      except KeyboardInterrupt:  # When 'Ctrl+C' is pressed, the child program destroy() will be
64  executed.
65          pass
```

Import pigpio obeject to control the pins of Raspberry Pi.

```
1   import pigpio
```

Write a sensor object and set all sensor pins to input mode.

```
3   class TCRT5000:
4       def __init__(self):
5           self.TCRT5000_PIN = [8, 11, 7]
6           self.tcrt5000 = pigpio.pi()
7           self.tcrt5000.set_mode(self.TCRT5000_PIN[0], pigpio.INPUT)
8           self.tcrt5000.set_mode(self.TCRT5000_PIN[1], pigpio.INPUT)
9           self.tcrt5000.set_mode(self.TCRT5000_PIN[2], pigpio.INPUT)
```

Read the sensor value and return it.

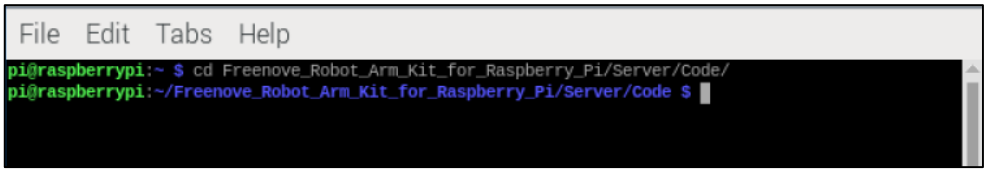
```
10      def readTCRT5000S1(self):
11          return self.tcrt5000.read(self.TCRT5000_PIN[0])  #
12      def readTCRT5000S2(self):
13          return self.tcrt5000.read(self.TCRT5000_PIN[1])
14      def readTCRT5000S3(self):
15          return self.tcrt5000.read(self.TCRT5000_PIN[2])
```

Read the data of the three sensors and return them.

```
16      def readTCRT5000ALL(self):
17          s1 = self.readTCRT5000S1()
```

```
18          s2 = self.readTCRT5000S2()
19          s3 = self.readTCRT5000S3()
20          return [s1,s2,s3]
```

Open the terminal and jump to the directory where the sensor.py file is located.



Run sudo python sensor.py and it will run the following code.

```
22  if __name__ == '__main__':
...     ......
```

When the sensor recognizes the black screw on the robot arm, the indicator light on the sensor goes out and the value 1 is returned.

When the sensor cannot recognize the black screw on the robot arm, the indicator light on the sensor lights up and a value of 0 is returned.

stepMotor.py

This file is used to control the stepper motor. The code is as below.

```
1   import pigpio
2   import time
3   import math
4   import threading
5   import os
6   import sensor
7
8   class StepMotor:
9       def __init__(self):
...         ......
26      def initA4988(self):
...         ......
48      def setA4988Enable(self, enable):
...         ......
51      def setA4988MSx(self, ms1, ms2, ms3):
...         ......
55      def readA4988Msx(self):
...         ......
67      def setA4988MsxMode(self, mode):
...         ......
86      def setA4988ClkFrequency(self, frequency):
...         ......
89      def myDelay(self, second):
...         ......
97      def motorRun(self, motor_number, direction, pulse_count, pulse_frequency):
...         ......
130     def  gotoMidSensorPoint1(self):
...         ......
184     def  gotoMidSensorPoint2(self):
...         ......
238     def  gotoMidSensorPoint3(self):
...         ......
291     def  caliSensorPoint(self):
...         ......
338     def  gotoSensorPoint(self, pulse_count):
...         ......
428     def pulseCountToAngle(self, pulse_count):
...         ......
432     def angleToPulseCount(self, angle):
...         ......
439     def angleToStepMotorParameter(self, targetAngle):
```

```
...         ......
451     def moveStepMotorToTargetAngle(self, targetAngle):
...         ......
482 if __name__ == '__main__':
483     import sys
484     os.system("sudo pigpiod")
485     time.sleep(1)
486     motor = StepMotor()
487     motor.setA4988Enable(0)
488     try:
489         if len(sys.argv)!=3:
490             while True:
491                 print("The stepper motor turns in one direction.")
492                 motor1 = threading.Thread(target=motor.motorRun, args=(1,0,200,1000,))
493                 motor2 = threading.Thread(target=motor.motorRun, args=(2,0,200,1000,))
494                 motor3 = threading.Thread(target=motor.motorRun, args=(3,0,200,1000,))
495                 motor1.start()
496                 motor2.start()
497                 motor3.start()
498                 motor1.join()
499                 motor2.join()
500                 motor3.join()
501                 print("The stepper motor turns in the other direction.")
502                 motor1 = threading.Thread(target=motor.motorRun, args=(1,1,200,1000,))
503                 motor2 = threading.Thread(target=motor.motorRun, args=(2,1,200,1000,))
504                 motor3 = threading.Thread(target=motor.motorRun, args=(3,1,200,1000,))
505                 motor1.start()
506                 motor2.start()
507                 motor3.start()
508                 motor1.join()
509                 motor2.join()
510                 motor3.join()
511         elif len(sys.argv)==3:
512             motor_number = int(sys.argv[1])
513             direction = int(sys.argv[2])
514             while True:
515                 print("The stepper motor turns in one direction:" , direction)
516                 motor_threading = threading.Thread(target=motor.motorRun,
    args=(motor_number,direction,3200,16000,))
517                 motor_threading.start()
518                 motor_threading.join()
519     except KeyboardInterrupt:  # When 'Ctrl+C' is pressed, the child program destroy() will be
    executed.
520         motor.setA4988Enable(1)
```

Import function modules.

```
1   import pigpio
    import time
    import math
    import threading
    import os
    import sensor
```

Write a stepper motor object that allows us to control the stepper motor.

```
8   class StepMotor:
9       def __init__(self):
...         ......
```

From the stepper motor test section in Chapter 1, we know that the Raspberry Pi does not directly connect to the stepper motor, but controls it through the stepper motor driver module. The stepper motor module we use is A4988, so we write a function to initialize the pins connected to the A4988 module.

```
26      def initA4988(self):
...         ......
```

Write a function to manage the power supply to the stepper motor through its driver module. The motor should be energized and operational when `enable` is set to 0, and it should be de-energized and inactive when `enable` is set to 1.

```
48      def setA4988Enable(self, enable):
...         ......
```

The “setA4988MSx” function is used to adjust the microstepping of the stepper motor by controlling the MS1 and MS2 pins. The `readA4988Msx` function is used to read the current microstepping setting of the stepper motor. The `setA4988MsxMode` function further encapsulates the “setA4988MSx” function. The `mode` parameter ranges from 1 to 5. When the parameter is 1, the stepper motor operates without microstepping, requiring 200 pulses to complete one rotation, with each pulse corresponding to an angle of 1.8 degrees. When the parameter is 5, the stepper motor is set to 16 microsteps, requiring 3200 pulses for one rotation, with each pulse corresponding to an angle of 0.1125 degrees. The default setting for the robotic arm is 16 microsteps.

```
51      def setA4988MSx(self, ms1, ms2, ms3):
..          ......
55      def readA4988Msx(self):
...         ......
67      def setA4988MsxMode(self, mode):
...         ......
```

Set the pulse frequency of the stepper motor that determines how quickly the Raspberry Pi sends control signals, which in turn affects the speed at which the robotic arm rotates. A higher pulse frequency results in faster control signal transmission and quicker arm movement, while a lower frequency slows down both the signal sending and the arm's rotational speed. It is recommended that the `frequency` parameter be set within the range of 100 to 16000 Hz.

```
86      def setA4988ClkFrequency(self, frequency):
...         ......
```

The function of myDelay function is the same as that of time.sleep function, with the unit being seconds.

```
89    def myDelay(self, second):
...       ......
```

The stepper motor operation function is used to control the rotation of the stepper motor. `motor_number` indicates the identifier of the motor, with a range of 1-3. `direction` signifies the direction of rotation of the motor, with a range of 0 and 1. `pulse_count` represents the number of pulses; with 16 microstepping, a pulse count of 3200 will result in one complete rotation of the motor. `pulse_frequency` denotes the pulse frequency; the higher the frequency, the faster the stepper motor rotates, and the lower the frequency, the slower it rotates, with a range of 100-16000.

```
97    def motorRun(self, motor_number, direction, pulse_count, pulse_frequency):
...       ......
```

The `gotoMidSensorPointx` function is designed to calculate the number of pulses required for the stepper motor to rotate from the edge of the sensor to the center of the sensor, and it returns this value.

The `caliSensorPoint` function calls `gotoMidSensorPointx` to rotate the robot arm to the center position of the sensor and returns the pulse counts for each motor as it moves to the center.

The `gotoSensorPoint` function is used to directly rotate the robotic arm to a position that is `pulse_count` pulses away from the edge of the sensor.

Typically, we first call the `caliSensorPoint` function to rotate the robotic arm to the center of the sensor and record the returned value. Later, if we want the robotic arm to directly rotate to the center position of the sensor, we simply call the `gotoSensorPoint` function with the return value of `caliSensorPoint` as the argument.

```
130   def  gotoMidSensorPoint1(self):
...       ......
184   def  gotoMidSensorPoint2(self):
...       ......
238   def  gotoMidSensorPoint3(self):
...       ......
291   def  caliSensorPoint(self):
...       ......
338   def  gotoSensorPoint(self, pulse_count):
...       ......
```

The large gear on the robot arm has 96 teeth, and the synchronous gear on the stepper motor has 16 teeth, giving a gear ratio of 6:1.

With the default microstepping of 16 for the robot arm, the number of pulses required for the joint to rotate one full circle is calculated as 200 (pulses per motor step) times 16 (microsteps) times 6 (gear ratio), which equals 19,200 pulses.

Therefore, to calculate how many degrees are corresponded by a certain number of pulses (`pulse_count`), we simply divide `pulse_count` by 19,200 pulses and then multiply by 360 degrees to get the corresponding angle in degrees.

Similarly, to calculate the pulse value needed for a certain angle of rotation, we divide the angle value by 360 degrees and then multiply by 19,200 to obtain the pulse value corresponding to that angle.

```
428   def pulseCountToAngle(self, pulse_count):
429       a4988PII = self.readA4988Msx()
430       angle = pulse_count * 360 / 6 / 200 / a4988PII
431       return angle
```

```
    def angleToPulseCount(self, angle):
        a4988Pll = self.readA4988Msx()
        if angle == 0:
            pulse_count = 0
        else:
            pulse_count = (angle / 360) * 6 * 200 * a4988Pll
        return pulse_count
```

Calling the 'angleToStepMotorParameter' function can calculate the number of pulses and the direction of rotation required by the stepper motor to rotate the robot arm to the target angle.

```
    def angleToStepMotorParameter(self, targetAngle):
        valueAngle = [(self.lastAngle[i] - targetAngle[i]) for i in range(3)]
        direction = [0,0,0]
        pulse_count = [0,0,0]
        for i in range(3):
            if valueAngle[i] >= 0:
                direction[i] = 0
            else:
                direction[i] = 1
            pulse_count[i] = self.angleToPulseCount(math.fabs(valueAngle[i]))
        return direction, pulse_count
```

The `moveStepMotorToTargetAngle` function is the core function of the entire robot arm. Each time we have the arm to rotate by a certain angle, the code converts these angle values into pulse counts.
However, for stepper motors, pulse counts are integers, and sometimes the calculated values are floating-point numbers. If we simply truncate the decimal part, over time this could lead to significant inaccuracies in the movement of the robot arm.
Therefore, we use `pulse_int_value` to store the integer part of the pulse count resulting from the angle-to-pulse conversion, and `pulse_margin` to store the decimal part, along with the direction of rotation.
When this function is called again, it adds the previous `pulse_margin` to the calculation, ensuring that the decimal part of the data is not lost. This allows the robotic arm to accurately move to each desired position.

```
    def moveStepMotorToTargetAngle(self, targetAngle):
        direction, pulse_count = self.angleToStepMotorParameter(targetAngle)
        self.lastAngle = targetAngle.copy()
        pulse_int_value = [0,0,0]
        for i in range(3):
            if direction[i] == self.pulse_margin_dir[i]:
                pulse_int_value[i] = round(pulse_count[i] + self.pulse_margin[i])
                self.pulse_margin[i] = pulse_count[i] + self.pulse_margin[i] - pulse_int_value[i]
            else:
                pulse_int_value[i] = round(pulse_count[i] - (self.pulse_margin[i]))
                self.pulse_margin[i] = pulse_count[i] + self.pulse_margin[i] - pulse_int_value[i]
        self.pulse_margin_dir = direction.copy()
        buflist = pulse_count.copy()
        buflist.sort(reverse=True)
        maxdata = buflist[0]
```

```
466         #print("stepmotor.py,", targetAngle, pulse_count, pulse_int_value, self.pulse_margin)
467         if maxdata != 0:
468             self.motor1 = threading.Thread(target=self.motorRun, args=(1, direction[0],
    pulse_int_value[0], self.A4988ClkFrequency[0],))
469             self.motor2 = threading.Thread(target=self.motorRun, args=(2, direction[1],
    pulse_int_value[1], self.A4988ClkFrequency[1],))
470             self.motor3 = threading.Thread(target=self.motorRun, args=(3, direction[2],
    pulse_int_value[2], self.A4988ClkFrequency[2],))
471             self.motor1.start()
472             self.motor2.start()
473             self.motor3.start()
474             try:
475                 self.motor1.join()
476                 self.motor2.join()
477                 self.motor3.join()
478             except:
479                 #print("Stepmotor.py, Motor thread is False.")
480                 pass
```

Enter sudo python stepMotor.py, and it will run the code below. The three motors on the robotic arm rotate back and forth in small increments.

```
482 if __name__ == '__main__':
483     import sys
484     os.system("sudo pigpiod")
485     time.sleep(1)
486     motor = StepMotor()
487     motor.setA4988Enable(0)
488     try:
489         if len(sys.argv)!=3:
490             while True:
491                 print("The stepper motor turns in one direction.")
492                 motor1 = threading.Thread(target=motor.motorRun, args=(1,0,200,1000,))
493                 motor2 = threading.Thread(target=motor.motorRun, args=(2,0,200,1000,))
494                 motor3 = threading.Thread(target=motor.motorRun, args=(3,0,200,1000,))
495                 motor1.start()
496                 motor2.start()
497                 motor3.start()
498                 motor1.join()
499                 motor2.join()
500                 motor3.join()
501                 print("The stepper motor turns in the other direction.")
502                 motor1 = threading.Thread(target=motor.motorRun, args=(1,1,200,1000,))
503                 motor2 = threading.Thread(target=motor.motorRun, args=(2,1,200,1000,))
504                 motor3 = threading.Thread(target=motor.motorRun, args=(3,1,200,1000,))
505                 motor1.start()
```

```
506                 motor2.start()
507                 motor3.start()
508                 motor1.join()
509                 motor2.join()
510                 motor3.join()
511         elif len(sys.argv)==3:
512             motor_number = int(sys.argv[1])
513             direction = int(sys.argv[2])
514             while True:
515                 print("The stepper motor turns in one direction:" , direction)
516                 motor_threading = threading.Thread(target=motor.motorRun,
    args=(motor_number,direction,3200,16000,))
517                 motor_threading.start()
518                 motor_threading.join()
      except KeyboardInterrupt:  # When 'Ctrl+C' is pressed, the child program destroy() will be
519 executed.
520         motor.setA4988Enable(1)
```

```
pi@raspberrypi:~/Freenove_Robot_Arm_Kit_for_Raspberry_Pi/Server/Code $ sudo pyth
on stepmotor.py
The stepper motor turns in one direction.
The stepper motor turns in the other direction.
The stepper motor turns in one direction.
The stepper motor turns in the other direction.
```

Please be aware: In the context of the stepper motor testing section in Chapter 1, the stepper motor is not mounted on the robo arm. At that stage, it is possible to use commands with parameters to control motor to spin continuously in a single direction. However, in the current chapter, with the stepper motor now integrated into the robot arm, employing commands with parameters that result in continuous rotation could cause mechanical structural conflicts. Consequently, do not use parameterized commands to avoid any potential damage to the robot arm.

arm.py

This file is used to control the robot arm. The code is as below.

```
1   from stepmotor import StepMotor
2   import math
3
4   class Arm:
5       def __init__(self):
...         ......
32      #mapping function
33      def map(self, value, fromLow, fromHigh, toLow, toHigh):
...         ......
35      #range limiting function
36      def constrain(self, value, min, max):
...         ......
42      #Determine whether a point is within a line segment formed by two other points
43      def point_is_between_line(self, p1, p2, p3):
...         ......
70      #Given that point P3(x3,y3) is located on the line connected by points P1(x1, y1, z1) and P2*(x2,
        y2, z2), find the z value of P3
71      def calculate_z_coordinate(self, start_axis, end_axis, axis):
...         ......
83      #Solve equation
84      def solve_quadratic(self, a, b, c):
...         ......
91      #Calculate the x coordinates of the intersection points according to the slope of the line, the
        intercept of the line and the radius of the circle
92      def find_intersections(self, k_value, b_value, r_value):
...         ......
102     #Calculate the coordinates of the intersection points by the coordinates, the radius of the circle
103     def calculate_axis(self, start_axis, end_axis, radius):
...         ......
149     #Determine whether a point is range of the ball
150     def is_point_inside_sphere(self, x, y, z, radius):
...         ......
158     #Based on the coordinates, radius of the circle, calculate the coordinate positions of the two
        points intersecting the circle, if the line does not intersect the circle, return [0, 0]
159     def calculate_valid_axis(self, start_axis, end_axis, radius):
...         ......
191     #Adjust the z-axis according to the X-axis
192     def setPlaneXZ(self, x1, x2, zz1, zz2):
...         ......
197     #Adjust the z-axis according to the Y-axis
198     def setPlaneYZ(self, y1, y2, zz1, zz2):
```

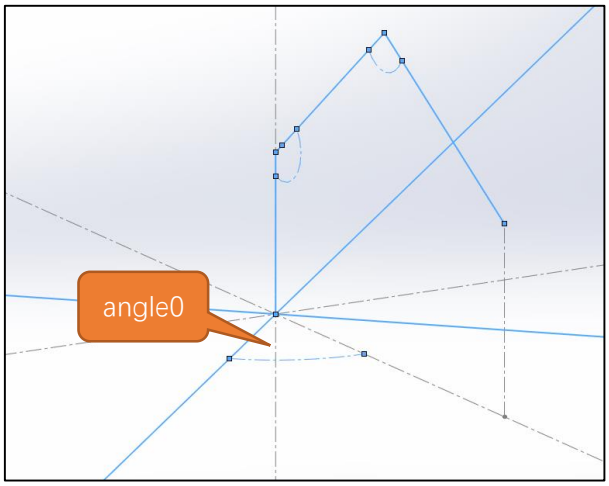
```
...     ......
203     #Radians are converted to angles
204     def radianToAngle(self, radian):
...     ......
206     #Angles are converted to radians
207     def angleToRadian(self, angle):
...     ......
209     #Triangulation: cosA =  (b*b+c*c-a*a) /2bc
210     def sidesToAngle(self, a, b, c):
...     ......
218     #Set the arm clamp length
219     def setClampLength(self, length):
...     ......
221     #Set the arm clamp height
222     def setClampHeight(self, height):
...     ......
224     #Set the height of the rotating shaft of the mechanical arm from the bottom surface
225     def setOriginHeight(self, height):
...     ......
227     #Set the height between the bottom of the robot arm and the ground
228     def setGroundHeight(self, height):
...     ......
230     #Set the height of the pen at the end of the robot arm
231     def setPenHeight(self, height):
...     ......
233     #Set the stepper motor pulse frequency
234     def setFrequency(self, frequency):
...     ......
237     #Set the stepper motor subdivision mode
238     def setMsxMode(self, mode):
...     ......
240     #Set stepper motor enable and disable
241     def setArmEnable(self, enable):
...     ......
243     #Set the robot arm to calibrate the sensor center position
244     def setArmToSensorPoint(self):
...     ......
249     #Move the arm to the center of the sensor
250     def setArmToSensorPointNoAdjust(self, pulse_count):
...     ......
254     #Set the stepper motor calibration offset Angle
255     def setArmOffseAngle(self, offsetAngle):
...     ......
257     #Convert coordinates to angles
```

```
258     def coordinateToAngle(self, axis):
...         ......
276     #Convert angles to coordinates
277     def angleToCoordinata(self, angle):
...         ......
303     #Control the robot arm to move to the corresponding coordinates
304     def moveStepMotorToTargetAxis(self, axis, mode=0):
...         ......
355 if __name__ == '__main__':
356     import os
357     import time
358     os.system("sudo pigpiod")
359     time.sleep(1)
360     arm = Arm()
361     arm.setArmEnable(0)
362     print(arm.setArmToSensorPoint())
363     arm.setArmEnable(1)
```

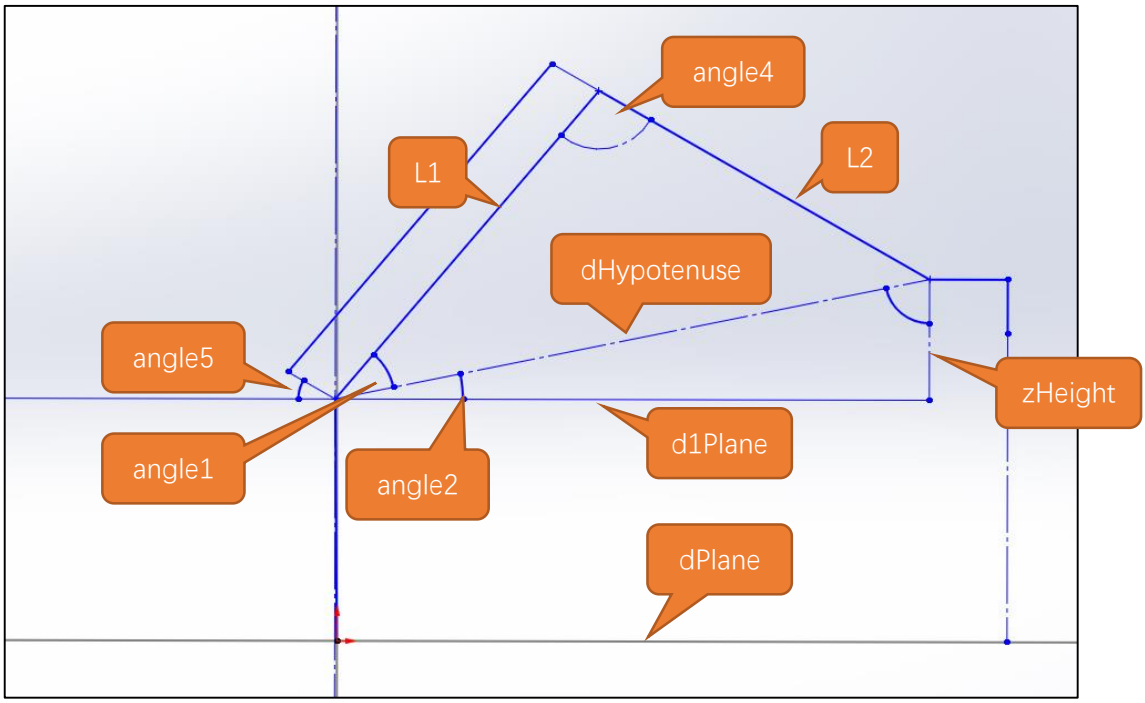
We can use the coordinateToAngle function to convert the end coordinates into the rotation angles of the three motors of the robotic arm.

```
258     #Convert coordinates to angles
259     def coordinateToAngle(self, axis):
260         tanxy = math.atan2(axis[1], axis[0])
261         angle0 = self.radianToAngle(tanxy)
262         dPlane = math.fabs(math.sqrt(math.pow(axis[0],2) + math.pow(axis[1],2)))
263         d1Plane = dPlane - self.CLAMP_LENGTH
264         zHeight = axis[2] - self.ORIGINAL_HEIGHT - self.GROUND_HEIGHT + self.PEN_HEIGHT
265         dHypotenuse = math.sqrt(math.pow(d1Plane,2) + math.pow(zHeight,2))
266         #cosA =  (b*b+c*c-a*a) /(2*b*c), a=b, cosA = c / (2b)
267         angle1 = self.sidesToAngle(self.L2_LENGTH, self.L1_LENGTH, dHypotenuse)
268         angle2 = self.radianToAngle(math.atan2(math.fabs(zHeight), d1Plane))
269         if zHeight > 0:
270             angle3 = angle1 + angle2
271         else:
272             angle3 = angle1 - angle2
273         angle4 = 180 - (2 * angle1)
274         angle5 = 180 - (angle3 + angle4)
275         angle = [angle0, angle3, angle5]
276         return angle
```

1. Calculate angle0 based on the x, y coordinates of the end of the robot arm, and convert the three-dimensional model into a two-dimensional model.



2. Calculate the values of d1Plane and zHeight, and based on these two values, calculate the value of dHypotenuse based on the Pythagorean Theorem (a*a+b*b=c*c).
3. Calculate the value of angle1 from the three-sided formula. (cosA = (b*b+c*c-a*a)/2bc)
4. According to the cotangent function in trigonometric functions, directly find the value of angle2.
5. If the coordinates of the end of the robot arm are lower than the height of d1Plane, then angle3=angle1-angle2, otherwise, then angle3=angle1+angle2.
6. Since L1=L2, the triangle formed by dHypotenuse, L1, and L2 is an isosceles triangle. Therefore, angle4 =180-(2*angle1)
7. Finally, as can be seen from the figure, angle5+angle3+angle4=180°, solve angle5.



Similarly, based on the model mentioned above, we can deduce the function `angleToCoordinate()` which converts angles to coordinates.

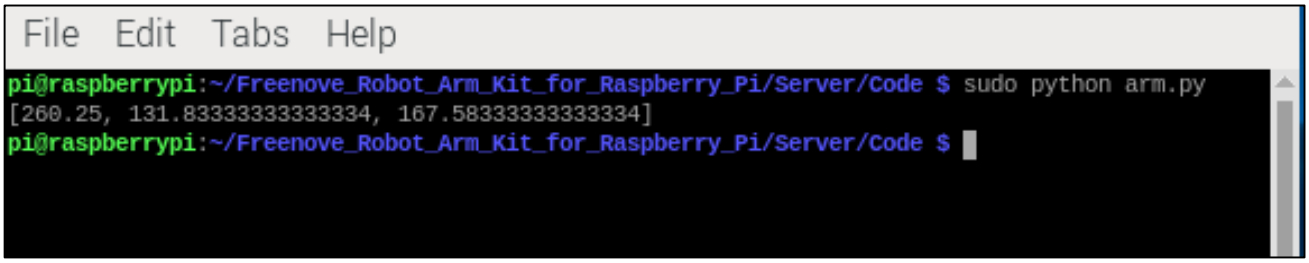
```
276     #Convert angles to coordinates
277     def angleToCoordinata(self, angle):
...         ......
```

The moveStepMotorToTargetAxis function is used to control the movement of the robotic arm's end effector to a target coordinate position.

```
303     #Control the robot arm to move to the corresponding coordinates
304     def moveStepMotorToTargetAxis(self, axis, mode=0):
...         ......
```

Run sudo python arm.py and the following code will be executed. The robot arm will perform a sensor calibration action and print out the pulse values.

```
355 if __name__ == '__main__':
356     import os
357     import time
358     os.system("sudo pigpiod")
359     time.sleep(1)
360     arm = Arm()
361     arm.setArmEnable(0)
362     print(arm.setArmToSensorPoint())
363     arm.setArmEnable(1)
```



servo.py

This file is used to control servos.

```
import pigpio
import time

class Servo:
    def __init__(self):
        self.SERVO_CHANNEL_PIN = [13,16,19,20,26]
        self.servo_index = -1
        self.PwmServo = pigpio.pi()
        self.initServo(self.servo_index)
    def constrain(self, value, min, max):
        if value > max:
            value = max
        if value < min:
            value = min
        return value
    def initServo(self, index):
        if self.servo_index != index:
            self.servo_index = self.constrain(index, 0, len(self.SERVO_CHANNEL_PIN))
            mode = self.PwmServo.get_mode(self.SERVO_CHANNEL_PIN[self.servo_index])
            freq = self.PwmServo.get_PWM_frequency(self.SERVO_CHANNEL_PIN[self.servo_index])
            pwm_range = self.PwmServo.get_PWM_range(self.SERVO_CHANNEL_PIN[self.servo_index])
            if (mode != pigpio.OUTPUT) or (freq != 50) or (pwm_range != 20000):
                self.PwmServo.set_mode(self.SERVO_CHANNEL_PIN[self.servo_index],pigpio.OUTPUT)
                self.PwmServo.set_PWM_frequency(self.SERVO_CHANNEL_PIN[self.servo_index],50)
                self.PwmServo.set_PWM_range(self.SERVO_CHANNEL_PIN[self.servo_index], 20000)
    def setServoAngle(self, index, angle):
        if index < len(self.SERVO_CHANNEL_PIN):
            self.initServo(index)
            angle = self.constrain(angle, 0, 180)
            servo_duty = 500+(2000/180)*angle
            self.PwmServo.set_PWM_dutycycle(self.SERVO_CHANNEL_PIN[self.servo_index], servo_duty)
            return servo_duty
    def relaxServo(self, index):
        if index < len(self.SERVO_CHANNEL_PIN):
            self.PwmServo.set_PWM_dutycycle(self.SERVO_CHANNEL_PIN[self.servo_index], 20000)

# Main program logic follows:
if __name__ == '__main__':
    import os
    import sys
    os.system("sudo pigpiod")
```

```
42      time.sleep(1)
43      servo = Servo()
44      print("")
45      try:
46          while True:
47              if len(sys.argv)==1:
48                  for j in range(180):
49                      for i in range(5):
50                          servo.setServoAngle(i, j)
51                      time.sleep(0.01)
52                  for j in range(180):
53                      for i in range(5):
54                          servo.setServoAngle(i, 180-j)
55                      time.sleep(0.01)
56              elif len(sys.argv)==2:
57                  index = servo.constrain(int(sys.argv[1]), 0, 4)
58                  servo.setServoAngle(index, 90)
59                  time.sleep(0.1)
60              elif len(sys.argv)==3:
61                  index = servo.constrain(int(sys.argv[1]), 0, 4)
62                  angle = servo.constrain(int(sys.argv[2]), 0, 180)
63                  servo.setServoAngle(index, angle)
64                  time.sleep(0.1)
65      except KeyboardInterrupt:
66          for i in range(5):
67              servo.relaxServo(i)
68          time.sleep(0.5)
69          servo.PwmServo.stop()
70          print ("\nEnd of program")
```

Import the pigpio object to control the pins of Raspberry Pi.

```
1   import pigpio
2   import time
```

Servo interface initialization function. The range of `index` is 0-4, corresponding to these 5 GPIOs: 13, 16, 19, 20, 23.

```
    def initServo(self, index):
        if self.servo_index != index:
            self.servo_index = self.constrain(index, 0, len(self.SERVO_CHANNEL_PIN))
            mode = self.PwmServo.get_mode(self.SERVO_CHANNEL_PIN[self.servo_index])
            freq = self.PwmServo.get_PWM_frequency(self.SERVO_CHANNEL_PIN[self.servo_index])
            pwm_range = self.PwmServo.get_PWM_range(self.SERVO_CHANNEL_PIN[self.servo_index])
            if (mode != pigpio.OUTPUT) or (freq != 50) or (pwm_range != 20000):
                self.PwmServo.set_mode(self.SERVO_CHANNEL_PIN[self.servo_index],pigpio.OUTPUT)
                self.PwmServo.set_PWM_frequency(self.SERVO_CHANNEL_PIN[self.servo_index],50)
                self.PwmServo.set_PWM_range(self.SERVO_CHANNEL_PIN[self.servo_index], 20000)
```

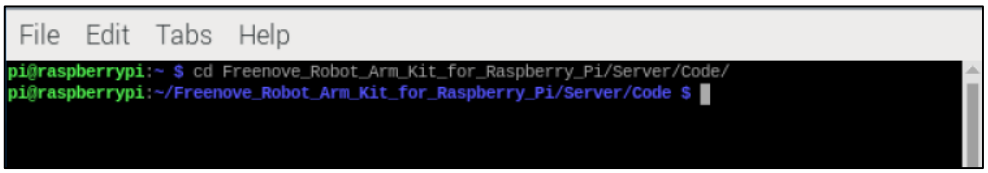
The setServoAngle function is used to control the specified servo to rotate to any angle. `index` is the servo index number, ranging from 0 to 4. `angle` is the angle to which the servo should rotate, ranging from 0 to 180 degrees.

```
    def setServoAngle(self, index, angle):
        if index < len(self.SERVO_CHANNEL_PIN):
            self.initServo(index)
            angle = self.constrain(angle, 0, 180)
            servo_duty = 500+(2000/180)*angle
            self.PwmServo.set_PWM_dutycycle(self.SERVO_CHANNEL_PIN[self.servo_index], servo_duty)
            return servo_duty
```

The servo disable function, when called, releases the torque on the specified servo. `index` is the servo index number, with a range of 0 to 4.

```
    def relaxServo(self, index):
        if index < len(self.SERVO_CHANNEL_PIN):
            self.PwmServo.set_PWM_dutycycle(self.SERVO_CHANNEL_PIN[self.servo_index], 20000)
```

Open a terminal and jump to the directory where the servo.py file is located.



Run sudo python servo.py and it will run the following code.

```
# Main program logic follows:
if __name__ == '__main__':
    ......
```

The following commands are for your reference.

Commands	Explanations
sudo python servo.py	Simultaneously controls five channels of servos to rotate back and forth repeatedly.
sudo python servo.py 0	Controls the servo of channel 0 (GPIO13) to rotate to 90 degress and stay at that position.
sudo python servo.py 0 90	Controls the servo of channel 0 (GPIO13) to rotate to 90 degress and stay at that position.
sudo python servo.py 0 150	Controls the servo of channel 0 (GPIO13) to rotate to 150 degress and stay at that position.
sudo python servo.py 1 45	Controls the servo of channel 1 (GPIO16) to rotate to 45 degress and stay at that position.

ledPixel.py

This file is used to control the LED module. The code is as below.

```
import time
from rpi_ws281x import *

# Define functions which animate LEDs in various ways.
class LedPixel:
    def __init__(self):
        self.LedMod= 0
        self.color=[0,0,0]
        #Control the sending order of color data
        self.ORDER = "RGB"
        # LED strip configuration:
        LED_COUNT      = 8      # Number of LED pixels.
        LED_PIN        = 18      # GPIO pin connected to the pixels (18 uses PWM!).
        LED_FREQ_HZ    = 800000  # LED signal frequency in hertz (usually 800khz)
        LED_DMA        = 10      # DMA channel to use for generating signal (try 10)
        LED_BRIGHTNESS = 255     # Set to 0 for darkest and 255 for brightest
        LED_INVERT     = False   # True to invert the signal (when using NPN transistor level shift)
        LED_CHANNEL    = 0       # set to '1' for GPIOs 13, 19, 41, 45 or 53
        # Create NeoPixel object with appropriate configuration.
        self.strip = Adafruit_NeoPixel(LED_COUNT, LED_PIN, LED_FREQ_HZ, LED_DMA, LED_INVERT,
LED_BRIGHTNESS, LED_CHANNEL)
        # Intialize the library (must be called once before other functions).
        self.strip.begin()
    #Returns the correct color data based on the color type
    def LED_TYPR(self, R, G, B):
        Led_type = ["GRB", "GBR", "RGB", "RBG", "BRG", "BGR"]
        color = [Color(G, R, B), Color(G, B, R), Color(R, G, B), Color(R, B, G), Color(B, R, G), Color(B, G, R)]
        if self.ORDER in Led_type:
            return color[Led_type.index(self.ORDER)]
    #Set the lights one by one and display
    def colorWipe(self, color, wait_ms=50, interval_ms=1000):
        self.strip.setBrightness(255)
        colorShow = self.LED_TYPR(color[0], color[1], color[2])
        for i in range(self.strip.numPixels()):
            self.strip.setPixelColor(i, colorShow)
            self.strip.show()
            time.sleep(wait_ms/1000.0)
        time.sleep(interval_ms/1000.0)
    #Custom RGB
    def RGBLed(self, color, wait_ms=100):
        self.strip.setBrightness(255)
```

```
41          colorShow = self.LED_TYPR(color[0], color[1], color[2])
42          for i in range(8):
43              self.strip.setPixelColor(i, colorShow)
44          self.strip.show()
45          time.sleep(wait_ms/1000.0)
46      #Following lights
47      def followingLed(self, color, wait_ms=60):
48          self.strip.setBrightness(255)
49          ledShow = [self.LED_TYPR(color[0]//8, color[1]//8, color[2]//8),
50                  self.LED_TYPR(color[0]//4, color[1]//4, color[2]//4),
51                  self.LED_TYPR(color[0]//2, color[1]//2, color[2]//2),
52                  self.LED_TYPR(color[0]//1, color[1]//1, color[2]//1)]
53          ledNum = self.strip.numPixels()
54          for z in range(ledNum):
55              for i in range(ledNum):
56                  self.strip.setPixelColor(i, Color(0,0,0))
57              for j in range(len(ledShow)):
58                  self.strip.setPixelColor((z+j)%ledNum, ledShow[j])
59              self.strip.show()
60              time.sleep(wait_ms/1000.0)
61      #Blink lights
62      def blinkLed(self, color, wait_ms=300):
63          self.strip.setBrightness(255)
64          colorShow = self.LED_TYPR(color[0], color[1], color[2])
65          for i in range(8):
66              self.strip.setPixelColor(i, colorShow)
67          self.strip.show()
68          time.sleep(wait_ms/1000.0)
69          for i in range(8):
70              self.strip.setPixelColor(i, Color(0, 0, 0))
71          self.strip.show()
72          time.sleep(wait_ms/1000.0)
73      #Breathing lights
74      def breathLight(self, color, wait_ms=15):
75          colorShow = self.LED_TYPR(color[0], color[1], color[2])
76          for i in range(8):
77              self.strip.setPixelColor(i, colorShow)
78          for i in range(100):
79              self.strip.setBrightness(i)
80              self.strip.show()
81              time.sleep(wait_ms/1000.0)
82          for i in range(100):
83              self.strip.setBrightness(100-i)
84              self.strip.show()
```

```
85              time.sleep(wait_ms/1000.0)
86      #Color palette
87      def wheel(self, pos):
88          """Generate rainbow colors across 0-255 positions."""
89          if pos<0 or pos >255:
90              r=g=b=0
91          elif pos < 85:
92              r=pos * 3
93              g=255 - pos * 3
94              b=0
95          elif pos < 170:
96              pos -= 85
97              r=255 - pos * 3
98              g=0
99              b=pos * 3
100         else:
101             pos -= 170
102             r=0
103             g=pos * 3
104             b=255 - pos * 3
105         return self.LED_TYPR(r,g,b)
106     #Rainbow light
107     def rainbowCycle(self, wait_ms=3, iterations=1):
108         self.strip.setBrightness(50)
109         """Draw rainbow that uniformly distributes itself across all pixels."""
110         for j in range(256*iterations):
111             for i in range(self.strip.numPixels()):
112                 self.strip.setPixelColor(i, self.wheel((int(i * 256 / self.strip.numPixels()) + j) & 255))
113             self.strip.show()
114             time.sleep(wait_ms/1000.0)
115     #Gradient rainbow light
116     def gradualChange(self, wait_ms=10, iterations=1):
117         self.strip.setBrightness(50)
118         """Draw rainbow that fades across all pixels at once."""
119         for j in range(256*iterations):
120             for i in range(self.strip.numPixels()):
121                 self.strip.setPixelColor(i, self.wheel((i+j) & 255))
122             self.strip.show()
123             time.sleep(wait_ms/1000.0)
124     #Rotating lights can be customed
125     def rotateLed(self, color, number=2, wait_ms=50):
126         self.strip.setBrightness(255)
127         colorShow = self.LED_TYPR(color[0], color[1], color[2])
128         ledNum = self.strip.numPixels()
```

```
129         for z in range(ledNum):
130             for i in range(ledNum):
131                 self.strip.setPixelColor(i, Color(0,0,0))
132             for j in range(number):
133                 self.strip.setPixelColor((z+j*(ledNum//number))%ledNum, colorShow)
134             self.strip.show()
135             time.sleep(wait_ms/1000.0)
136     #The colored light displays the function, which needs to be executed using a thread loop. data
    contains four parameters, 0:mode, 1:R, 2:G, and 3:B
137     def light(self, data):
138         self.LedMod=data[0]
139         for i in range(3):
140             self.color[i]=int(data[i+1]%256)
141         if self.LedMod==0:                                  #close
142             self.color = [0,0,0]
143             self.colorWipe(self.color)
144         elif self.LedMod==1:                                #RGB
145             self.RGBLed(self.color)
146         elif self.LedMod==2:                                #Following
147             self.followingLed(self.color)
148         elif self.LedMod==3:                                #Blink
149             self.blinkLed(self.color)
150         elif self.LedMod==4:                                #Breathing
151             self.breathLight(self.color)
152         elif self.LedMod==5:                                #Rainbow
153             self.rainbowCycle()
154         elif self.LedMod==6:                                #Gradual
155             self.gradualChange()
156         elif self.LedMod==7:                                #The two lights rotate symmetrically
157             self.rotateLed(self.color, 2, 50)
158         elif self.LedMod==8:                                #The four lights rotate symmetrically
159             self.rotateLed(self.color, 4, 100)
160         elif self.LedMod>=9 or self.LedMod < 0:
161             self.LedMod = 0
162             print("parameter error! Press Ctrl+c to exit and re-enter the parameters, please.")
163             time.sleep(1)
164 # Main program logic follows:
165 if __name__ == '__main__':
166     import sys
167     led=LedPixel()
168     led.strip.setBrightness(255)
169     print ('Program is starting ... ')
170     col=[Color(255,0,0), Color(0,255,0), Color(0,0,255)]
171     try:
```

```
172         if len(sys.argv)<5 and len(sys.argv)>=2:
173             strParameter = sys.argv[1:]
174             intParameter = [int(strParameter[i]) for i in range(len(strParameter))]
175             parameter = [intParameter[0], 128, 0, 0]
176             while True:
177                 led.light(parameter)
178         elif len(sys.argv)<2:
179             print("Enter a parameter, 'sudo python ledPixel.py 1 0 0 255'")
180             print("or 'sudo python ledPixel.py 1'")
181         else:
182             strParameter = sys.argv[1:5]
183             intParameter = [int(strParameter[i]) for i in range(len(strParameter))]
184             while True:
185                 led.light(intParameter)
186     except KeyboardInterrupt:  # When 'Ctrl+C' is pressed, the child program destroy() will be
    executed.
187         for i in range(8):
188             led.strip.setPixelColor(i, Color(0, 0, 0))
189         led.strip.show()
```

Import the RGB LED object to control the LED module.

```
    mport time
    from rpi_ws281x import *
```

Designate colors to the LEDs one by one.

```
    #Set the lights one by one and display
    def colorWipe(self, color, wait_ms=50, interval_ms=1000):
        self.strip.setBrightness(255)
        colorShow = self.LED_TYPR(color[0], color[1], color[2])
        for i in range(self.strip.numPixels()):
            self.strip.setPixelColor(i, colorShow)
            self.strip.show()
            time.sleep(wait_ms/1000.0)
        time.sleep(interval_ms/1000.0)
```

Make the eight LEDs to emit designated colors at the same time.

```
    #Custom RGB
    def RGBLed(self, color, wait_ms=100):
        self.strip.setBrightness(255)
        colorShow = self.LED_TYPR(color[0], color[1], color[2])
        for i in range(8):
            self.strip.setPixelColor(i, colorShow)
        self.strip.show()
        time.sleep(wait_ms/1000.0)
```

Display a rotating effect like flowing water, with the color customizable to any desired hue.

```
#Following lights
def followingLed(self, color, wait_ms=60):
    self.strip.setBrightness(255)
    ledShow = [self.LED_TYPR(color[0]//8, color[1]//8, color[2]//8),
           self.LED_TYPR(color[0]//4, color[1]//4, color[2]//4),
           self.LED_TYPR(color[0]//2, color[1]//2, color[2]//2),
           self.LED_TYPR(color[0]//1, color[1]//1, color[2]//1)]
    ledNum = self.strip.numPixels()
    for z in range(ledNum):
        for i in range(ledNum):
            self.strip.setPixelColor(i, Color(0,0,0))
        for j in range(len(ledShow)):
            self.strip.setPixelColor((z+j)%ledNum, ledShow[j])
        self.strip.show()
        time.sleep(wait_ms/1000.0)
```

The eight RGB lights blink, displaying a specified color.

```
#Blink lights
def blinkLed(self, color, wait_ms=300):
    self.strip.setBrightness(255)
    colorShow = self.LED_TYPR(color[0], color[1], color[2])
    for i in range(8):
        self.strip.setPixelColor(i, colorShow)
    self.strip.show()
    time.sleep(wait_ms/1000.0)
    for i in range(8):
        self.strip.setPixelColor(i, Color(0, 0, 0))
    self.strip.show()
    time.sleep(wait_ms/1000.0)
```

The eight RGB lights transition from dark to bright and then back to dark, with the color customizable to any desired shade.

```
#Breathing lights
def breathLight(self, color, wait_ms=15):
    colorShow = self.LED_TYPR(color[0], color[1], color[2])
    for i in range(8):
        self.strip.setPixelColor(i, colorShow)
    for i in range(100):
        self.strip.setBrightness(i)
        self.strip.show()
        time.sleep(wait_ms/1000.0)
    for i in range(100):
        self.strip.setBrightness(100-i)
        self.strip.show()
        time.sleep(wait_ms/1000.0)
```

Color palette function that takes a value from 0 to 255 and returns a tuple containing the red, green, and blue color components.

```
#Color palette
def wheel(self, pos):
    """Generate rainbow colors across 0-255 positions."""
    if pos<0 or pos >255:
        r=g=b=0
    elif pos < 85:
        r=pos * 3
        g=255 - pos * 3
        b=0
    elif pos < 170:
        pos -= 85
        r=255 - pos * 3
        g=0
        b=pos * 3
    else:
        pos -= 170
        r=0
        g=pos * 3
        b=255 - pos * 3
    return self.LED_TYPR(r,g,b)
```

The eight RGB lights display colors like a rainbow and slowly rotate. The colors cannot be set manually.

```
#Rainbow light
def rainbowCycle(self, wait_ms=3, iterations=1):
    self.strip.setBrightness(50)
    """Draw rainbow that uniformly distributes itself across all pixels."""
    for j in range(256*iterations):
        for i in range(self.strip.numPixels()):
            self.strip.setPixelColor(i, self.wheel((int(i * 256 / self.strip.numPixels()) + j) & 255))
        self.strip.show()
        time.sleep(wait_ms/1000.0)
```

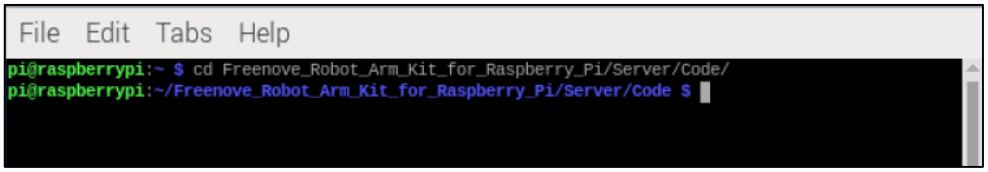
The eight RGB lights gradually transition from one color to another simultaneously, and this process cycles continuously, with the colors not being adjustable.

```
#Gradient rainbow light
def gradualChange(self, wait_ms=10, iterations=1):
    self.strip.setBrightness(50)
    """Draw rainbow that fades across all pixels at once."""
    for j in range(256*iterations):
        for i in range(self.strip.numPixels()):
            self.strip.setPixelColor(i, self.wheel((i+j) & 255))
        self.strip.show()
        time.sleep(wait_ms/1000.0)
```

The eight RGB lights display only a subset of the lights, with a symmetrical rotation pattern, and the color value can be specified to any desired color.

```
#Rotating lights can be customed
def rotateLed(self, color, number=2, wait_ms=50):
    self.strip.setBrightness(255)
    colorShow = self.LED_TYPR(color[0], color[1], color[2])
    ledNum = self.strip.numPixels()
    for z in range(ledNum):
        for i in range(ledNum):
            self.strip.setPixelColor(i, Color(0,0,0))
        for j in range(number):
            self.strip.setPixelColor((z+j*(ledNum//number))%ledNum, colorShow)
        self.strip.show()
        time.sleep(wait_ms/1000.0)
```

Open the terminal and jump to the directory where the ledPixel.py file is located.



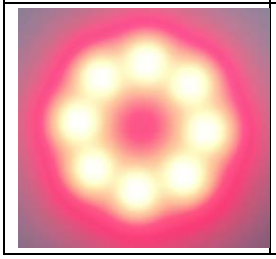
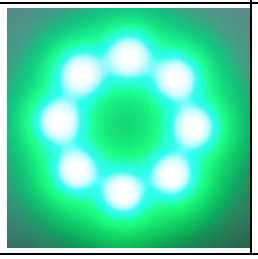
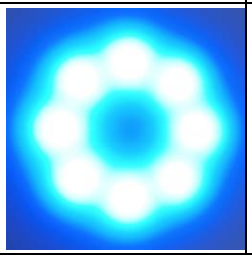
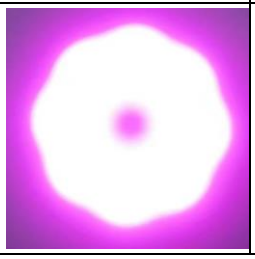
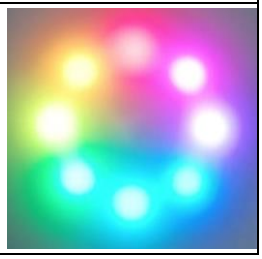
Run sudo python ledPixel.py and it will run the following code.

```
# Main program logic follows:
if __name__ == '__main__':
    ......
```

You can try the following commands.

Commands	Explanation
sudo python ledPixel.py 1	Customize the color of the RGB LED. Without parameters, the color is red by default.
sudo python ledPixel.py 1 0 0 255	Customize the color, The parameters set the red and green to 0 and the blue to 255, so the LED module only emits blue light.
sudo python ledPixel.py 2	Repeatedly emits red, green and blue. Paramters are not needed.
sudo python ledPixel.py 3	Running water effect. Without parameters, the color is red by default.
sudo python ledPixel.py 3 0 255 0	Running water effect. The parameters set the red and blue to 0 and the green to 255, so the LED module only emits green light.
sudo python ledPixel.py 4	Gradient light, slowly changing in rainbow colors. Parameters are not needed.
sudo python ledPixel.py 5	Rainbow lights, repeatedly turning the rainbow colors. Parameters are not needed.
sudo python ledPixel.py 6	Breathing light. Withouth parameters, the color is red by default.
sudo python ledPixel.py 6 255 0 255	Breathign light. The parameters set the red and blue to 255 and the green to 255, so the LED module only emits red and blue light, which looks pinkish purple.

The colors displayed are as below:

Red	Green	Blue	Pinkish purple	Rainblow lights
				

buzzer.py

This file is used to control the buzzer. The code is as below.

```
1  import pigpio
2  import time
3
4  class Buzzer:
5      def __init__(self):
6          self.BUZZER_PIN = 21
7          self.PwmBuzzer = pigpio.pi()
8          self.initBuzzer()
9
10     def initBuzzer(self):
11         self.PwmBuzzer.set_mode(self.BUZZER_PIN, pigpio.OUTPUT)
12         self.PwmBuzzer.set_PWM_frequency(self.BUZZER_PIN, 2000)
13         self.PwmBuzzer.set_PWM_range(self.BUZZER_PIN, 100)
14         self.PwmBuzzer.set_PWM_dutycycle(self.BUZZER_PIN, 0)
15
16     def buzzerRun(self, frequency=2000):
17         if frequency != 0:
18             self.PwmBuzzer.set_PWM_dutycycle(self.BUZZER_PIN, 50)
19             self.PwmBuzzer.set_PWM_frequency(self.BUZZER_PIN, frequency)
20         else:
21             self.PwmBuzzer.set_PWM_dutycycle(self.BUZZER_PIN, 0)
22             self.PwmBuzzer.set_PWM_frequency(self.BUZZER_PIN, 0)
23
24     def buzzerRunXms(self, frequency=2000, delayms=100, times=1):
25         for i in range(times):
26             self.buzzerRun(frequency)
27             time.sleep(float(delayms/1000))
28             self.buzzerRun(0)
29             time.sleep(float(delayms/1000))
30         self.buzzerRun(0)
31
32 # Main program logic follows:
```

```
33  if __name__ == '__main__':
34      import os
35      import sys
36      os.system("sudo pigpiod")
37      time.sleep(1)
38      B=Buzzer()
39      try:
40          if len(sys.argv)==1:
41              B.buzzerRunXms(2000,100,3)
42          elif len(sys.argv)==2:
43              B.buzzerRunXms(int(sys.argv[1]),100,3)
44          elif len(sys.argv)==3:
45              B.buzzerRunXms(int(sys.argv[1]),int(sys.argv[2]),3)
46          elif len(sys.argv)==4:
47              B.buzzerRunXms(int(sys.argv[1]),int(sys.argv[2]),int(sys.argv[3]))
48          elif len(sys.argv)>4:
49              print("Too many parameters.")
50              B.buzzerRunXms(1000,300,2)
51      except KeyboardInterrupt:
52          B.buzzerRun(0)
53          print('quit')
```

Buzzer initialization function.

```
10      def initBuzzer(self):
11          self.PwmBuzzer.set_mode(self.BUZZER_PIN, pigpio.OUTPUT)
12          self.PwmBuzzer.set_PWM_frequency(self.BUZZER_PIN, 2000)
13          self.PwmBuzzer.set_PWM_range(self.BUZZER_PIN, 100)
14          self.PwmBuzzer.set_PWM_dutycycle(self.BUZZER_PIN, 0)
```

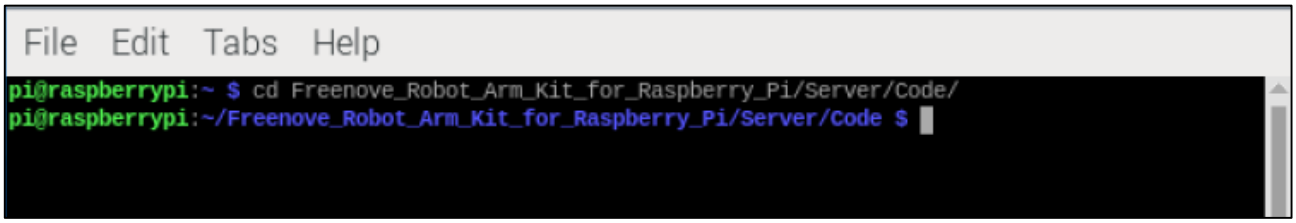
This function makes the buzzer to sound at a specified frequency within the range of 1-4000Hz.

```
16      def buzzerRun(self, frequency=2000):
17          if frequency != 0:
18              self.PwmBuzzer.set_PWM_dutycycle(self.BUZZER_PIN, 50)
19              self.PwmBuzzer.set_PWM_frequency(self.BUZZER_PIN, frequency)
20          else:
21              self.PwmBuzzer.set_PWM_dutycycle(self.BUZZER_PIN, 0)
22              self.PwmBuzzer.set_PWM_frequency(self.BUZZER_PIN, 0)
```

Buzzer sound function. `frequency` represents the sound frequency, `delayms` indicates the duration of each beep, and `times` specifies the number of beeps.

```
24      def buzzerRunXms(self, frequency=2000, delayms=100, times=1):
25          for i in range(times):
26              self.buzzerRun(frequency)
27              time.sleep(float(delayms/1000))
28              self.buzzerRun(0)
29              time.sleep(float(delayms/1000))
30          self.buzzerRun(0)
```

Open the terminal and jump to the directory where the buzzer.py file is located.



Run sudo python buzzer.py and it will run the following code.

```
22  if __name__ == '__main__':
...     ......
```

You can try the following commands.

Commands	Explanations
sudo python buzzer.py	The buzzers makes three beeps with a frequency opf 2kHz, each lasting 100ms.
sudo python buzzer.py 2500	The buzzers makes three beeps with a frequency opf 2.5kHz, each lasting 100ms.
sudo python buzzer.py 2000 300	The buzzers makes three beeps with a frequency opf 2kHz, each lasting 300ms.
sudo python buzzer.py 2000 100 5	The buzzers makes five beeps with a frequency opf 2kHz, each lasting 100ms.

main.py

```
1   import time
2   import socket
3   import fcntl
4   import struct
5   import messageThread
6   import messageQueue
7   import messageParser
8   import messageRecord
9   import command
10  import arm
11  import buzzer
12  import servo
13  import ledPixel
14
15  class ArmServer:
16      def __init__(self):
...         ......
64      #Set the message receiving thread
65      def setThreadingReceiveState(self, state):
...         ......
77      #Set the robot arm to move thread
78      def setThreadingArmState(self, state):
...         ......
89      #Set the light thread
90      def setThreadingLedState(self, state):
...         ......
101     #Set the buzzer thread
102     def setThreadingBuzzerState(self, state):
...         ......
113     #Set the feedback thread
114     def setThreadingFeedbackState(self, state):
...         ......
125     #Buzzer instructions are generated according to the parameters and sent to the message queue
126     def setRobotBuzzer(self, frequency, delayms, times):
...         ......
133     #Generate light instructions according to the parameters and send them to the message queue
134     def setRobotLED(self, mode, r, g, b):
...         ......
141     #The move instruction is generated according to the parameters and sent to the message queue
142     def setRobotAction(self, axis):
...         ......
148     #Get the IP address of the Raspberry PI
```

```
149     def get_interface_ip(self):
...         ......
152     #socket sending function
153     def serverSend(self,data):
...         ......
155     #Opening the socket Server
156     def turn_on_server(self):
...         ......
172     #Disable the socket server
173     def turn_off_server(self):
...         ......
184     #Check the installation and operation status of mechanical arm
185     def safetyOperationInspection(self):
...         ......
245     #Message receiving thread
246     def threadingReceiveInstruction(self):
...         ......
362     #The moving action thread
363     def threadingRobotAction(self):
...         ......
642     #Light thread
643     def threadingRobotLed(self):
...         ......
651     #Buzzer thread
652     def threadingRobotBuzzer(self):
...         ......
666     #Feedback thread
667     def threadingRobotActionFeedback(self):
...         ......
674     #Server check thread
675     def threadingCheckServer(self):
...         ......
695 if __name__ == '__main__':
...         ......
```

Write an object to control the robot arm and initialize it.

```
15  class ArmServer:
16      def __init__(self):
...         ......
```

Thread start and stop functions. When the `state` parameter is True, the thread is started. When the `state` parameter is False, the thread is stopped.

```
64      #Set the message receiving thread
65      def setThreadingReceiveState(self, state):
...         ......
77      #Set the robot arm to move thread
```

```
78      def setThreadingArmState(self, state):
...         ......
89      #Set the light thread
90      def setThreadingLedState(self, state):
...         ......
101     #Set the buzzer thread
102     def setThreadingBuzzerState(self, state):
...         ......
113     #Set the feedback thread
114     def setThreadingFeedbackState(self, state):
...         ......
```

Generate the corresponding command based on the parameters and place it into the respective message queue, waiting for the associated thread to execute the code in the order of the queue.

```
125     #Buzzer instructions are generated according to the parameters and sent to the message queue
126     def setRobotBuzzer(self, frequency, delayms, times):
...         ......
133     #Generate light instructions according to the parameters and send them to the message queue
134     def setRobotLED(self, mode, r, g, b):
...         ......
141     #The move instruction is generated according to the parameters and sent to the message queue
142     def setRobotAction(self, axis):
...         ......
```

This function obtains the IP address of Raspberry Pi.

```
148     #Get the IP address of the Raspberry PI
149     def get_interface_ip(self):
...         ......
```

This functions sends communication data.

```
152     #socket sending function
153     def serverSend(self, data):
...         ......
```

Start the robot arm server thread. Establish socket communication and print the IP address, then create message reception thread, motion execution thread, RGB light thread, and buzzer thread.

```
155     #Opening the socket Server
156     def turn_on_server(self):
...         ......
```

Close the robot arm server thread. Terminate all threads and disconnect the socket connection.

```
172     #Disable the socket server
173     def turn_off_server(self):
...         ......
```

The safety check function for the robotic arm is designed to ensure that the arm is fully set up, thereby minimizing the risk of accidental operation.

```
184     #Check the installation and operation status of mechanical arm
185     def safetyOperationInspection(self):
...         ......
```

The robot arm's reception thread is used to receive commands from the App/Software. It then performs a preliminary parsing of the commands and sends the parsed instructions to the respective message queues for their corresponding functions.

```
245     #Message receiving thread
246     def threadingReceiveInstruction(self):
...         ......
```

The robot arm's motion thread is utilized to control the movement of the arm and the rotation of the servos. When data is detected in the message queue, it parses each message and executes the content derived from the parsing.

```
362     #The moving action thread
363     def threadingRobotAction(self):
...         ......
```

The RGB light thread is used to control the patterns and colors of the lights. When data is detected in the message queue, it retrieves the latest instruction from the queue, parses it, and executes it.

```
642     #Light thread
643     def threadingRobotLed(self):
...         ......
```

The buzzer thread is designed to control the buzzer. When data is detected in the message queue, it parses each message and executes the content derived from the parsing.

```
651     #Buzzer thread
652     def threadingRobotBuzzer(self):
...         ......
```

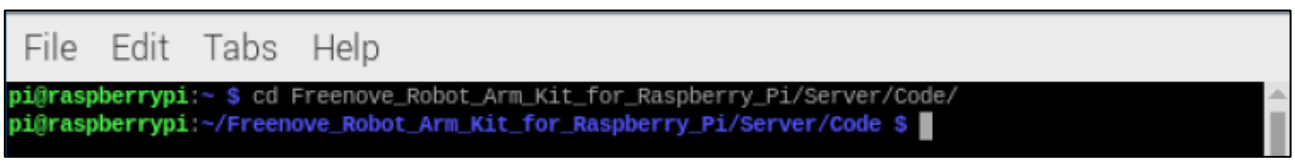
The motion instruction remaining count feedback thread is activated upon receiving a query start command from the App/Software. It queries the current number of remaining motion instructions every 0.5 seconds and feeds this information back to the App/Software. When a query stop command is received from the App/Software, this thread is terminated.

```
666     #Feedback thread
667     def threadingRobotActionFeedback(self):
...         ......
```

The server connection check thread. When executed, this thread will check whether the server has been started, if there are any users connected, if any users have disconnected, and whether to shut down the server.

```
674     #Server check thread
675     def threadingCheckServer(self):
...         ......
```

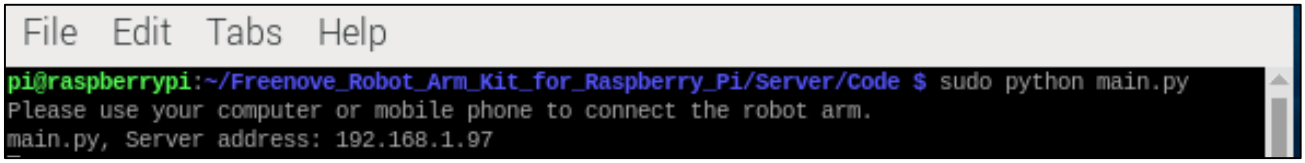
Open the terminal and jump to the directory where the main.py file is located.



Run sudo python main.py and it will run the following code.

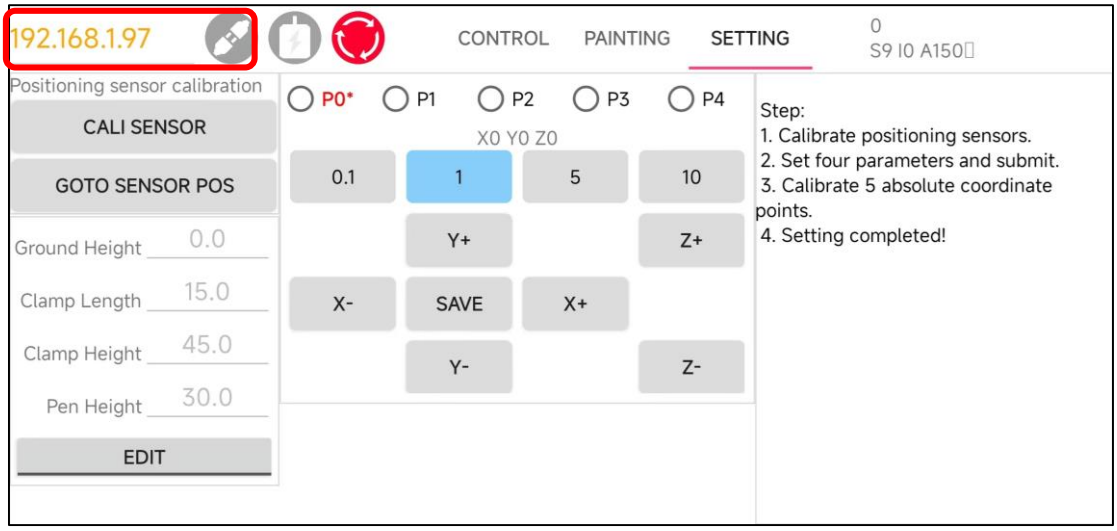
```
695 if __name__ == '__main__':
...     ......
```

The IP address of the Raspberry Pi will be printed on the Terminal.

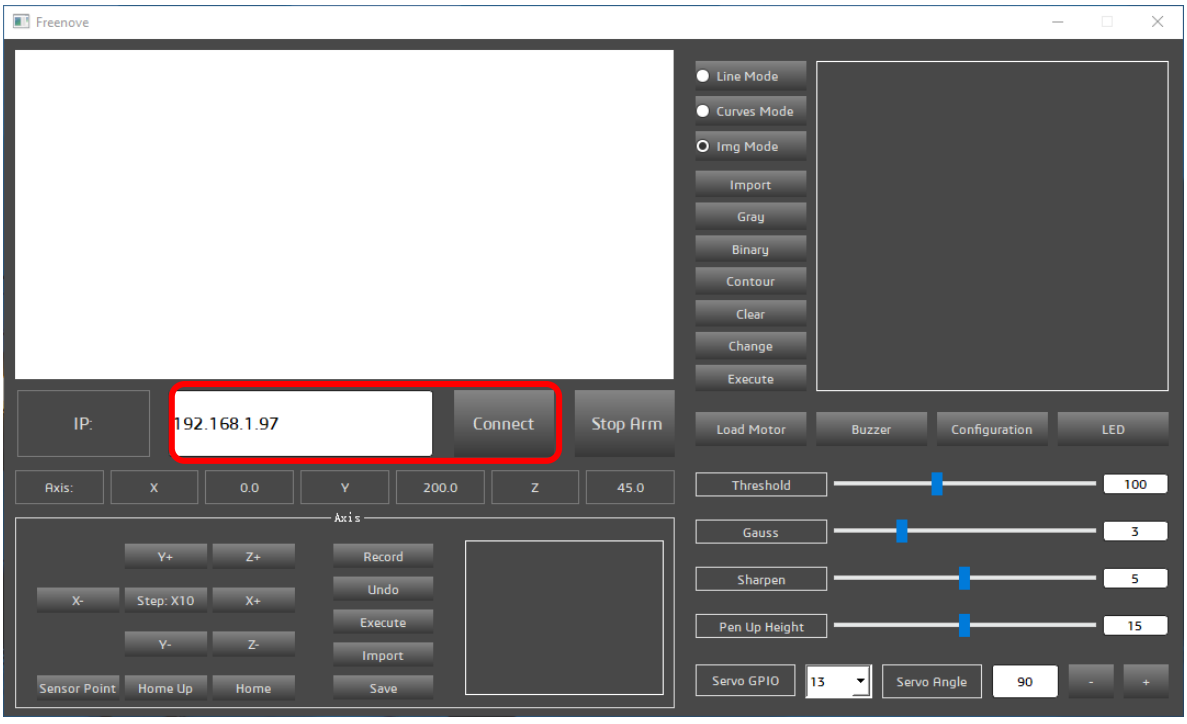


You can connect the APP/Software to the printed IP address of Raspberry Pi.

App



Software



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.