

Table of Contents

Introduction	1.1
1 Product Introduction	1.2
1.1 Design Philosophy	1.2.1
1.2 Suitable Users	1.2.2
1.3 Application Scenario	1.2.3
1.4 Accessories Tools	1.2.4
2 Product Feature	1.3
2.1 Functional Parameters	1.3.1
2.2 Controller Parameters	1.3.2
2.3 Structural Parameters	1.3.3
2.4 Electronic Parameters	1.3.4
2.5 Coordinate System	1.3.5
3 User Notes	1.4
3.1 Safety Instructions	1.4.1
3.2 Transport and Storage	1.4.2
3.3 Maintenance and Care	1.4.3
3.4 FAQs	1.4.4
4 First Install and Use	1.5
4.1 Product Standard List	1.5.1
4.2 Product Unboxing Guide	1.5.2
4.3 Power-on Test Guide	1.5.3
5 Basic Application	1.6
5.1 System instruction manual	1.6.1
5.2 Application Use	1.6.2
1 myPanel	1.6.2.1
5.3 Firmware Use	1.6.3
1 firmware update info	1.6.3.1
1 how to burn firmware	1.6.3.2
6 SDK Development	1.7
6.1 Python	1.7.1
1 Environment Building	1.7.1.1
2 Introduction to API	1.7.1.2
3 TCP/IP Control	1.7.1.3
6.2 Robot Operating System 1 (ROS1)	1.7.2
1 Environment Building	1.7.2.1
2 ROS basics	1.7.2.2

3 Rviz use	1.7.2.3
4 Basic function case	1.7.2.4
6.3 Robot Operating System 2 (ROS2)	1.7.3
1 Environment Building	1.7.3.1
2 ROS2 basics	1.7.3.2
3 Rviz2 use	1.7.3.3
4 Basic function case	1.7.3.4
6.4 C plus plus (C++)	1.7.4
1 CPlus Download	1.7.4.1
2 build	1.7.4.2
3 API	1.7.4.3
4 example	1.7.4.4
7 Examples of Robots Using	1.8
8 Documents Download	1.9
8.1 Productin formation	1.9.1
8.2 Product Drawings	1.9.2
8.3 Software and Source Code	1.9.3
8.4 System Information	1.9.4
8.5 Publicity Material	1.9.5
9 About Us	1.10
9.1 Elephant Robotics	1.10.1
9.2 Contact us	1.10.2

Mercury A1



1 Profile

Mercury A1 features an exquisite structure and integrated design, with a maximum arm span of 450 millimeters, a maximum load of 1 kilogram, and a repeat positioning accuracy of ± 0.05 millimeters.

Weighing 3.5 kilograms, the Mercury A1 has a 1-kilogram load capacity and a working radius of 450 millimeters. This product boasts powerful functionality, easy operation, and the ability to collaborate safely with humans.

2 Features

- **Easy to operate and open-source**
 - Users can operate the robot via myBlockly and dragging teaching easily after quick-start learning.
 - It supports the development systems, such as ROS and moveIt.
- **Powerful performance and equipped with two screens**
 - Utilizes a brushless DC servo to achieve a repeat positioning accuracy of ± 0.05 millimeters.
 - The body is equipped with two screens and supports M5 ecological applications, effectively expanding coordinative application.
- **On-in-all design and safe collaborative work**
 - With a delicate structure, it optimizes space and integrates with applications in a coordinated manner.
 - Features kinematics self-interference detection, effectively avoiding motion collisions.

3 Application

Mercury A1 is not just a production tool; it is a tool to expand the boundaries of imagination. It can work with multiple types of end effectors to adapt to various applications, such as scientific research, education, and functional displays. The user experience is excellent.

4 Chapter Summary

The next section of this manual will guide you to the sub-sections that will give you a more complete understanding of our product specifications, control core parameters, mechanical structure parameters, electrical characteristics parameters, and coordinate system definitions.

Please feel free to select the following sections based on your interests and requirements:

[2.1-Machine specification](#)

In this section, we will describe the basic attributes of the product's industry consensus, such as robot description load, torque, positioning accuracy, size, functional support, and power parameters.

[2.2-Control core parameter](#)

Understand the main control parameters of the product, which is convenient for later custom development and use.

[2.3-Mechanical structure parameter](#)

In this part, we will introduce the important parameters of the mechanical structure of the product in detail, and provide customers with the corresponding 3D model download link, so that customers can better understand our products.

[2.4-Electrical characteristic parameter](#)

This chapter will provide customers with the electrical characteristic parameters of the product, which is convenient for customers to customize the development and use in the later period.

[2.5-Coordinate system](#)

This section describes the Angle and coordinate information of the product and explains the supported coordinate system controls. At the same time, the relevant parameters of the product are provided for the calculation of the corresponding coordinate system, such as DH parameters.

Please click on the respective links based on your interests to access more detailed information. If you have any questions or require further assistance, please do not hesitate to contact our customer support team. We are committed to providing you with support and guidance. Thank you for choosing our product, and we look forward to delivering an outstanding user experience for you!

5 Words of Thanks

We greatly appreciate you taking the time to read the Mercury A1 user manual. We hope this document helps you to better understand and effectively use this robot, thereby inspiring your creativity. Should you have any questions or require further assistance, please do not hesitate to contact our customer support team. We look forward to seeing the innovative projects you accomplish with myCobot 320 and welcome you to our rapidly growing community of developers.

If you have already read all the content in this chapter, please proceed to the next chapter.

[← Previous Chapter](#) | [Next Chapter →](#)

Product specification parameter

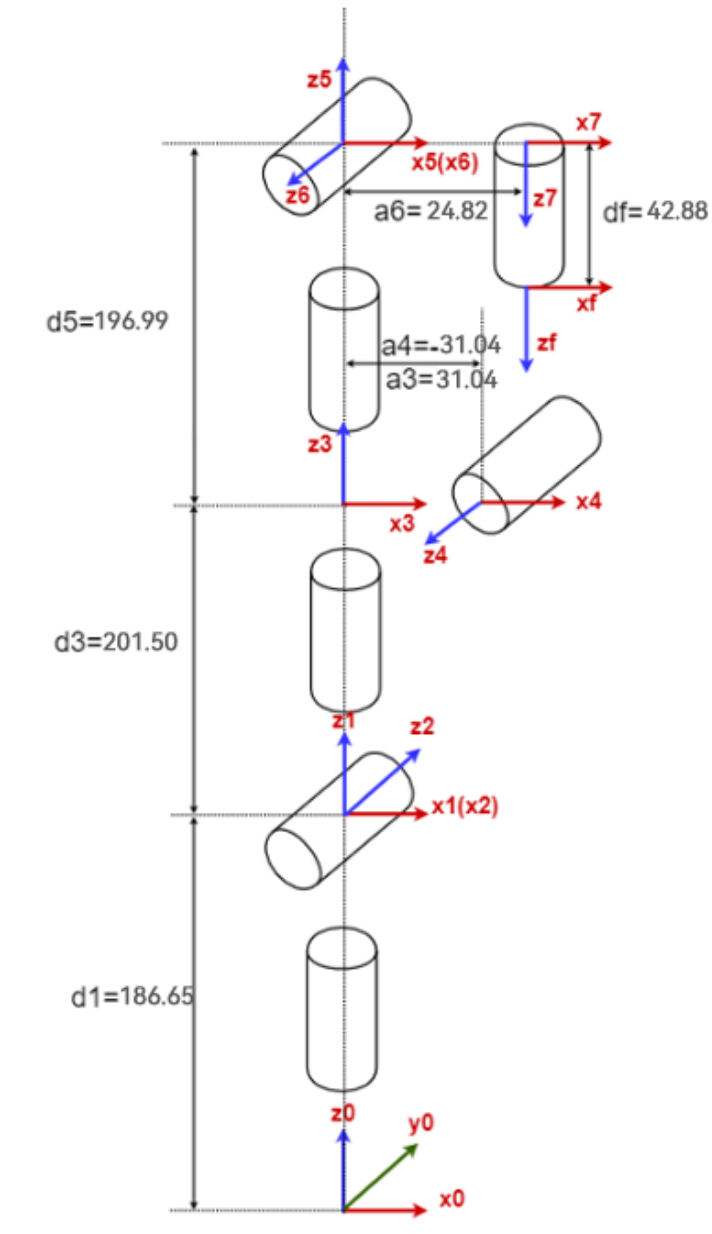


1 Manipulator parameter

Index	Parameter
Name	Elephant Consumer Robotic Arm
Model	MercuryA1
Product Size	98128640mm
Freedom	7
Working Radius	450mm
Efficient Load	1kg
Repeated Positioning Precision	±0.05mm
Weight	3.5kg
Power Input	DC24V/9.2A
Housing Material	Carbon Fiber
Communication Interface	WIFI/CAN bus/Bluetooth /USB/Serial Port
Service life	5000 hours

2 Structural Parameter

joint	joint minimum value °	joint maximum value °	joint maximum speed (°/s)	joint maximum acceleration (°/s ²)
J1	-175	175	150	200
J2	-65	115	150	200
J3	-175	175	150	200
J4	-180	10	150	200
J5	-175	175	150	200
J6	-20	173	150	200
J7	-180	180	150	200



3 Software basic function Support

Function/development environment	Usage situation
Free Mobile	Support
Articulated Movement	Support
Cartesian Movement	Support
Path Recording	Support
Wireless Control	Support
Emergency Stop	Support
Windows	Support
Linux	Support
MAC	Support
ROS 1	Support
Python	Support
C++	Support
C#	Support
JavaScript	Support
myblockly	Support
Arduino	Support
mystudio	Support
Serial Control Protocol	Support
TCP/IP	Support
MODBUS	Support

[← Previous Page](#) | [Next Page →](#)

Control core parameter



1 Master controller specification table

Index	Parameter
Main Control	M5-CM4STACK
Model	ESP32
Core Parameters	CM4104032, Processor: Broadcom BCM2711, Quad-core Cortex-A72 64-bit SoC, Clock Speed: 1.5 GHz, RAM: 4GB, Storage (eMMC): 32GB
2.0 inch LCD	Touch screen with ST7789V2 controller
HDMI	4K/p60 video output
Gigabit Ethernet Interface	RJ45 interface for connecting to a Gigabit Ethernet network, speed: 1Gbps
Speaker	2W audio output with AW88298 driver
WIFI	2.4 GHz, 5.0 GHz IEEE 802.11 b/g/n/ac
USB3.2Interface	Supports USB 3.2 protocol, using ASM3042 chip, high-speed PCIe bridge

2 Secondary controller 1 Specifications

Index	Parameter
Auxiliary Controller	Atom
Model	ESP32
Core Parameters	240MHz dual core, 600 DMIPS, 520KB SRAM, Wi-Fi, dual-mode Bluetooth
Auxiliary Controller Flash	4MB
LED Display	5X5 RGB

3 Secondary controller 2 Specifications

Index	Parameter
Auxiliary Controller	Pico
Model	ESP32
Core Parameters	240MHz dual core, 600 DMIPS, 520KB SRAM, Wi-Fi, dual-mode Bluetooth
Auxiliary Controller Flash	4MB

[← Previous Page](#) | [Next Page →](#)

Structural dimension parameter

! This chapter uses millimeters as the unit of distance and degrees as the unit of angle.

1 Product size and working space

When selecting the installation position of the robot, it is essential to consider the cylindrical space directly above and below the robot, and try to avoid moving the tool into these cylindrical spaces as much as possible. This is because doing so can cause the joints to rotate too quickly when the tool moves slowly, resulting in inefficient robot operation and making risk assessment difficult.

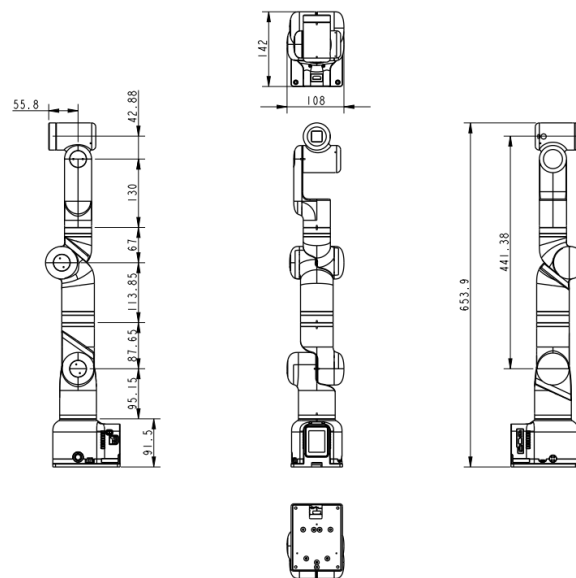


Figure 2.3.1 Product Dimensions

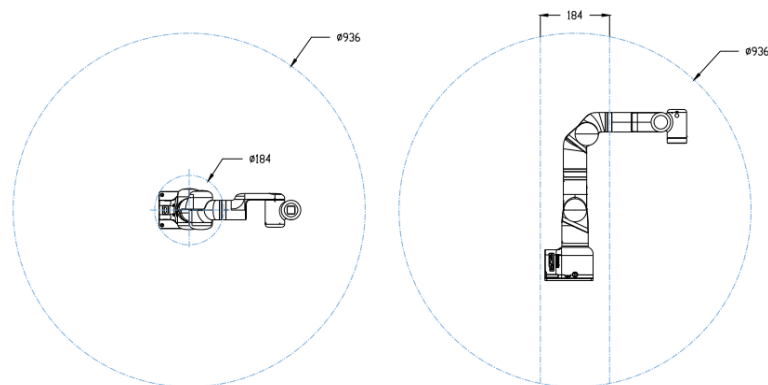


Figure 2.3.2 Product Workspace

2 Base Mounting Dimensions

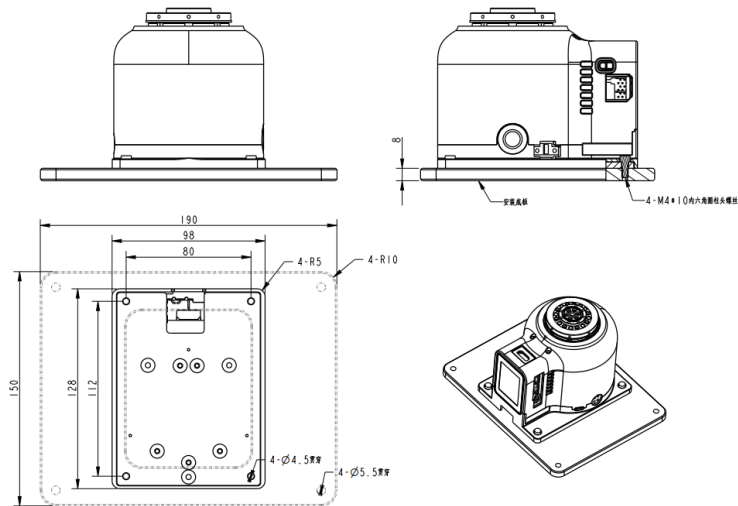


Figure 2.3.3 BaseMountingDimensions

3 End Flange Dimensions

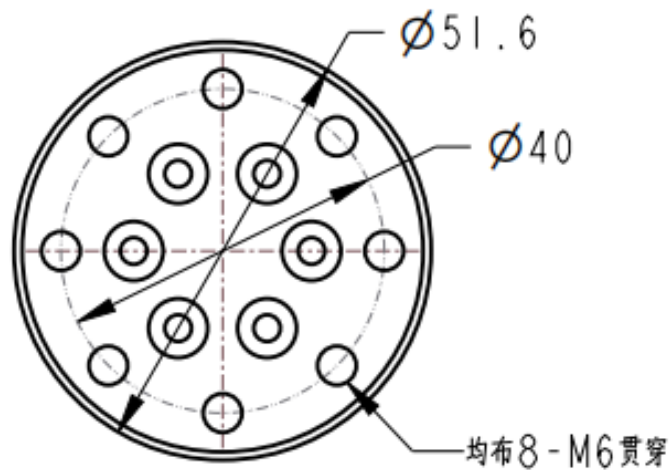


Figure 2.3.4 EndEffectorDimensions

4 3D model download

Product 3D model can be provided to provide reference materials for customers.

Download link:

https://download.elephantrobotics.com/Product_3d_files/myCobot_320_M5_2022_v1.2_230708.STEP

Electrical characteristic parameter

1 Base Interface Overview



Figure 2.4.1 Left side of the base

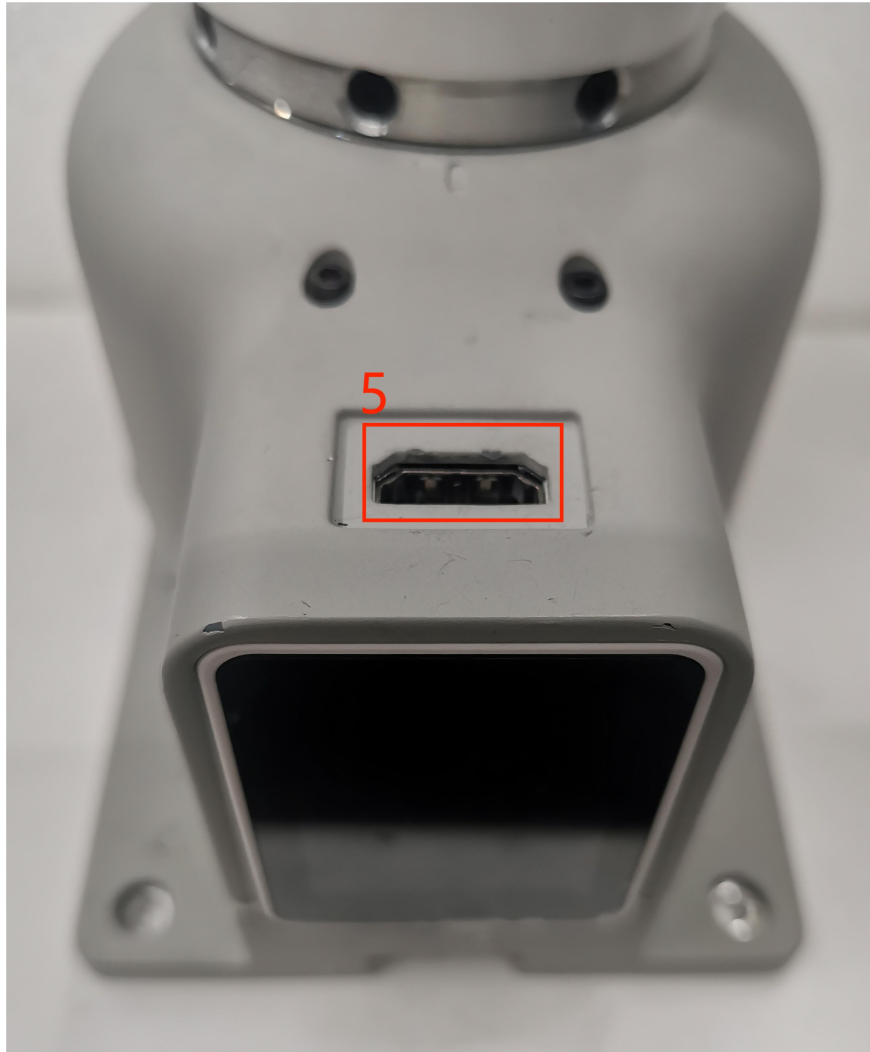


Figure 2.4.2 Front view of a base

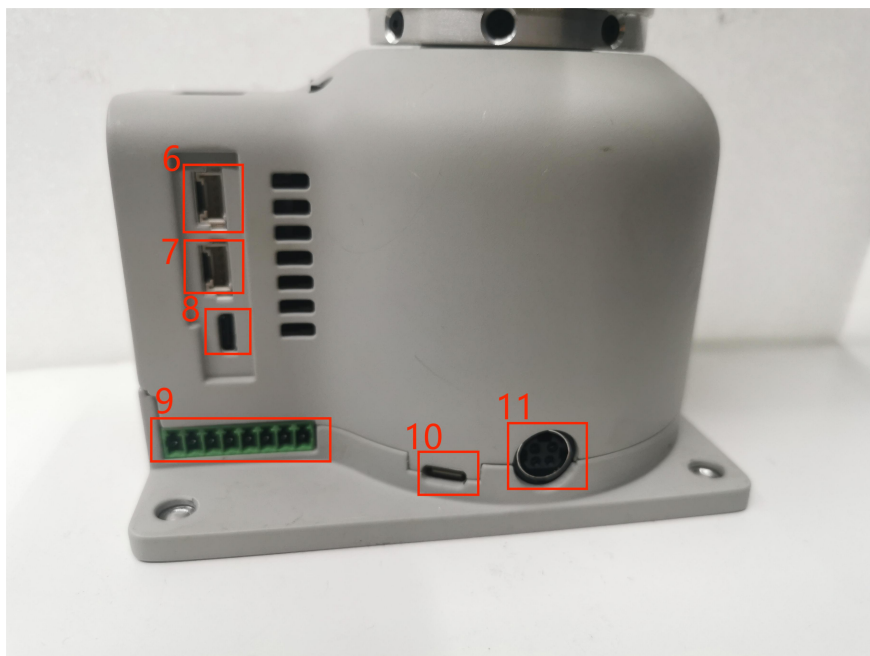


Figure 3 Right side of the base

1.1 Base Interface Description

Number	Interface	Definition	Function	Remark
1	Switch	Power Switch	Control power on/off	With indicator when powered
2	STOP	STOP	Emergency stop circuit interface	
3	USB3.2	USB3.2*2	External device or USB flash drive connection	
5	HDMI		Connects to the display	
6	Serial Port		Communication interface for transmitting and receiving serial data	
7	I2C Interface		Connects to I2C devices	
8	USB2.0	USB2.0	External device or USB flash drive connection	
9	Type C	Communication Interface	Communication with PC	For development use
10	Power Input Interface	DC24V Input	DC24V Input	

1 Power Switch: Controls the on/off of the main power input. When turned off, the controller is also powered off.

2 Emergency Stop Terminal: Connected to the emergency stop button box, used to control the robot's emergency stop.

Note: The emergency stop switch must be connected during robot operation, and ensure that the emergency stop switch circuit is in a connected state.

3 USB3.2 Interface: Interface for data connection using the USB 3.0 serial bus standard. Users can use the USB interface to copy program files or connect peripherals such as mice and keyboards.

5 HDMI Display Interface: Users can display the operating interface on other devices by connecting to the Micro HDMI display interface.

6 UART Serial Port: Communication interface for transmitting and receiving serial data.

7 I2C Interface: Interface used to connect I2C devices.

8 USB-C Interface: Interface for data connection using the USB 2.0 serial bus standard. Users can use the USB interface to copy program files or connect

peripherals such as mice and keyboards. This USB2.0 interface with OTG function is powered by the MP8759 and AW32901 controllers.

9 Type C : Used for communication with the PC, for development use.

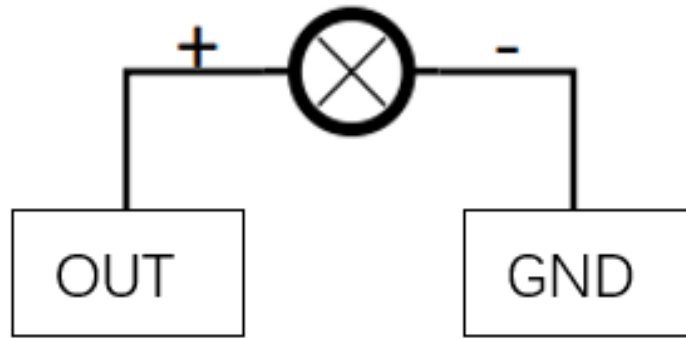
10 Power Input Interface: This interface is connected to the DC24V power adapter interface.

1.2 Base IO Interface Description

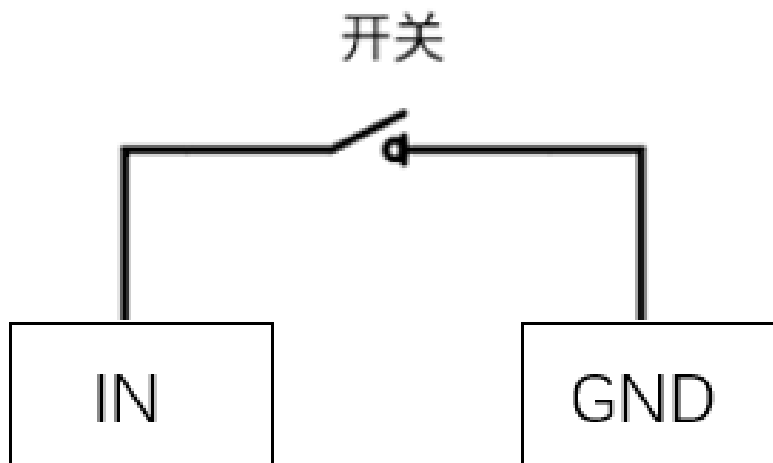
Number	Interface	Definition	Function	Remarks
4	DC/IO Interface	GND	GND	
		IN6	Digital input signal 1~6	Input is in NPN mode only
		IN5		
		IN4		
		IN3		
		IN2		
		IN1		
		24V	DC24V	DC24V input
9	DC/IO Interface	24V	DC24V	DC24V output
		OUT1	Digital output signal 1~6	Output is in PNP mode only
		OUT2		
		OUT3		
		OUT4		
		OUT5		
		OUT6		
		GND	GND	

1 Digital Input/Digital Output:

Includes 6 digital input signals and 6 digital output signals, used to interact with other devices as an important part of the automation system. It should be noted that the output signal is in PNP format, and the input signal is in NPN format. The external wiring diagram is as follows:



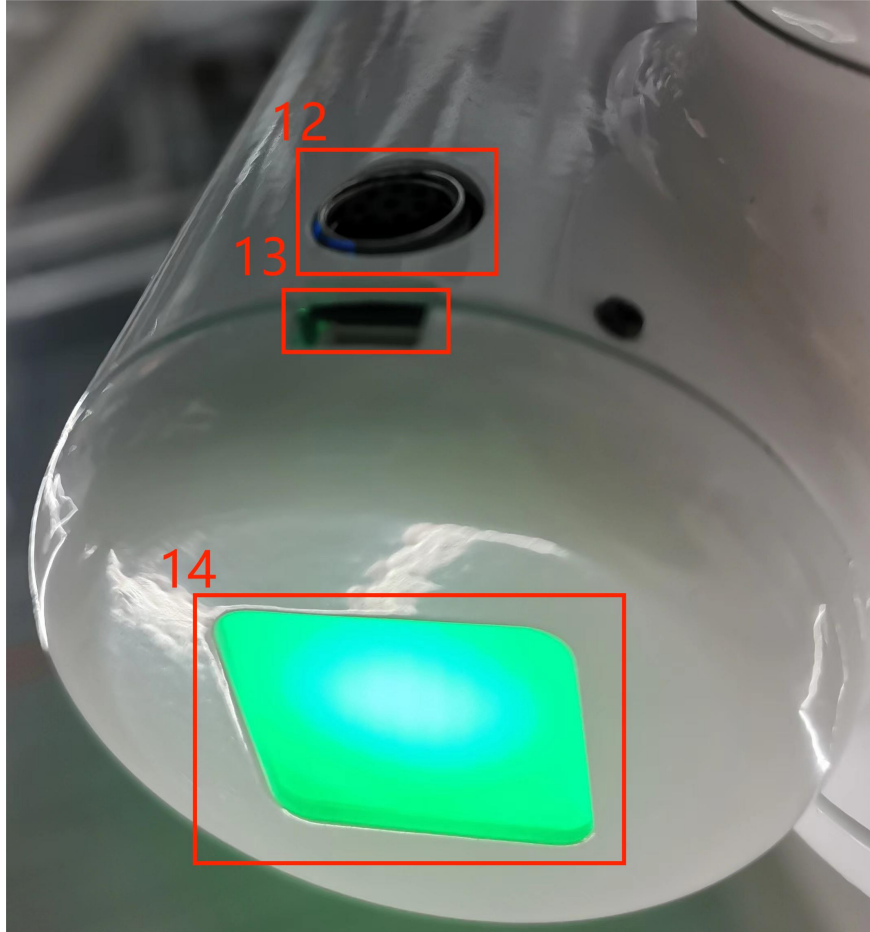
输出信号接线示例



输入信号接线示例

2 DC24V Output: Internal DC24V, available for user use.

2 End Interface Overview

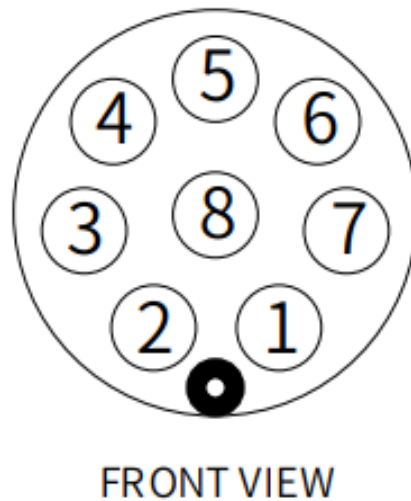


End effector side view

2.1 End Interface Description

Number	Interface	Definition	Function	Remarks
12	M8 Aviation Socket	End Tool IO Interface	Interact with external devices	
13	4-pinUSB Connector	External Interface	Connects to the camera	
14	Atom	Led + Buttons	Status viewing/ Drag-and-drop teaching	

1 M8 Aviation Socket I/O Diagram: The MercuryA1 robot provides one input and two outputs.



The definitions of each tool I/O port are shown in the table below. It should be noted that both input and output signals for the tool I/O are of the PNP type, and the wiring method is the same as for the bottom output interface.

Number	Signal	Explanation	Matching M8 Cable Color
1	GND	DC24V negative	White
2	OUT1	Tool output interface 1	brown
3	OUT2	Tool output interface 2	green
4	485A	reserved, undeveloped	yellow
5	24V	DC24V positive	Ash
6	IN1	Tool input interface 1	pink
7	IN2	unavailable	blue
8	485B	reserved, undeveloped	purple

2.1.2 Atom: Atom is used for the 5X5 RGB LED, displaying the status of the robotic arm and button functions (used during robot drag-and-drop teaching).

2.1.3 USB Connector:

[← Previous Page](#) | [Next Page →](#)

User Notes



This section is crucial for every user of this product and must be read carefully. It includes essential information on product use, transportation, storage, and maintenance to ensure safety and efficiency during operation. Additionally, it outlines liability for product failure or damage resulting from non-compliance with these guidelines. The User Notice is divided into subsections, each providing detailed guidance on different topics:

- [Safety Information](#)
Includes liability, safety warning signs, general safety rules, personal safety, and emergency response.
- [Transportation and Storage](#)
Describes packaging, transport, and long-term storage requirements, along with liability.
- [Maintenance and Care](#)
Offers guidance on routine maintenance to extend product lifespan.
- [FAQs and Solutions](#)
Provides a navigable guide for quickly resolving common issues.

By thoroughly reading this section, users will better understand how to use the product safely and efficiently, maximizing performance and lifespan.

If you have already read all the content in this chapter, please proceed to the next chapter.

[← Previous Chapter](#) | [Next Chapter →](#)

Safety Instructions

1 Synopsis

This chapter details general safety information for personnel performing installation, maintenance, and repair work on elephant robots. Read and understand the contents and precautions in this chapter before carrying, installing, and using it.

2 Hazard identification

The safety of cooperative robots is based on the proper configuration and use of robots. Furthermore, injury or damage caused by the operator may occur even if all safety instructions are followed. Therefore, it is very important to understand the safety risks of robot use in order to prevent them.

Table 1-1 to 3 lists the common security risks that may occur when robots are used:

Table 1-1 Risk level Security risks


<div></div>	
1	Personal injury or robot damage caused by improper handling of the robot.
2	If the robot is not fixed as required, for example, the screw is missing or the screw is not tight, or the locking capacity of the base is insufficient to support the robot to move at high speed, the robot will fall over, resulting in personal injury or robot damage.
3	The robot's safety function fails to play its role due to the incorrect configuration of safety functions or the lack of safety protection tools.

Table 1-2 Warning security risks



警告

1 When debugging the program, do not stay within the robot's motion range. Improper safety configuration may fail to avoid collisions, potentially causing personal injury.

2 The connection between robots and other equipment may introduce new hazards, necessitating a comprehensive risk assessment.

3 Be cautious of scratches and punctures caused by sharp surfaces, such as other equipment in the working environment or robot end effectors.

4 Robots are precision machines; stepping on them can cause damage. Improper placement during transportation may lead to vibration, affecting internal parts and causing damage. Therefore, ensure stability and mechanical structural integrity in all circumstances.

5 Failure to remove a clamped object before powering off the robot (when the clamping is not secure) may lead to dangers such as damage to the end effector or injuries if the clamped object falls due to power loss.

6 There is a risk of unexpected robot movement. Never stand under any axis of the robot under any circumstances!

7 Compared to ordinary mechanical equipment, robots have more degrees of freedom and a larger range of motion. Failure to stay within the range of motion may result in unexpected collisions.

Table 1-3 Potential safety hazards that may lead to electric shock



小心触电

1 Unknown hazards may arise when using non-original cables.

2 Electrical equipment contact with liquid may cause leakage hazard.

3 Electrical connection error may cause electric shock.

4 Be sure to switch off the power supply of the controller and related devices and remove the power plug before replacement. If the operation is carried out with power on, it may cause electric shock or failure.

3 Safety Precautions

The following safety rules should be followed when using the manipulator:

- The mechanical arm belongs to live equipment. Non-professionals are not allowed to change the circuit at will, otherwise it may cause damage to the equipment or human body.
- When operating mechanical arms, comply with local laws and regulations. The safety precautions and dangers, Warnings, and precautions described in this manual are only supplements to the local safety regulations.
- Please use the mechanical arm within the specified environment. Exceeding the specifications and load conditions of the mechanical arm will shorten the service life of the product and even damage the equipment.
- The person installing, operating and maintaining the myCobot arm, anyway, has to be rigorously trained on safety precautions and the right way to operate and maintain the robot.
- Anyway, don't use the product in humid environments for long periods of time. This product is a precision electronic component, which will damage the equipment in damp environment for a long time.
- Anyway, don't use the product in humid environments for long periods of time. This product is a precision electronic component, which will damage the equipment in damp environment for a long time.
- Highly corrosive cleaning is not suitable for cleaning the mechanical arm, and anodized parts are not suitable for immersion cleaning.
- Unconsciously, do not use the device without installing a base to avoid damaging the device or accidents, instead use the device in a fixed environment without obstacles.
- Do not use other power adapters for power supply. If the equipment is damaged due to the use of non-standard adapters, the after-sales service will not be included.
- Do not disassemble, disassemble, or unscrew the screws or shell of the manipulator. If disassembled, no warranty service can be provided.
- Personnel without professional training are not allowed to repair the faulty products and dismantle the mechanical arm without permission. If the products fail, please contact myCobot technical support engineers in time.
- If the product is discarded, please comply with the relevant laws to properly dispose of industrial waste and protect the environment.
- A child uses a device at some point, forcing someone to monitor the process and switch it off when it's finished.
- When the robot is moving, do not extend your hand into the movement range of the robot arm, for fear of collision.
- It is strictly prohibited to change, remove or modify the nameplate, description, icon and mark of the manipulator and related equipment.
- Please be careful in handling and installation. Put the robot gently according to the instructions on the packing case and place it correctly in the direction of the arrow. Otherwise, the machine may be damaged.
- **Do not burn other product drivers from Atom terminal, or burn firmware using unofficial recommendations. If the equipment is damaged due to**

the user burning other firmware, it will not be in the after-sales service.

- Power supply specifications: **Use the official power supply**
- USB Type-C usage specifications: **Do not connect to power strips**

If you have any questions or suggestions about the contents of this manual, please log in the official website of Elephant Robot and submit relevant information:

<https://www.elephantrobotics.com>

Please do not use the mechanical arm for the following purposes:

- Cost of healthcare in life-critical applications.
- Buying a bus can cause an explosion in an environment.
- Lent is used directly without a risk assessment.
- Cost of using a security function at a low level.
- Lo-fi does not conform to the use of robot performance parameters.

[← Previous Section](#) | [Next Chapter →](#)

Transport and Storage



During transportation, the original packaging should be used for transporting the Mercury A1. During transportation, it should be ensured that the Mercury A1 is stable as a whole and secured through appropriate measures.

During transportation and long-term storage, the environmental temperature should be maintained within the range of -20 to $+55^{\circ}\text{C}$, with humidity $\leq 95\%$ and no condensation.

As robots are precision machinery, when removing the Mercury A1 from its packaging, it should be handled with care. During transportation, if it cannot be placed steadily, it may cause vibration and damage to internal components of the robot.

After transportation is completed, the original packaging should be properly stored in a dry place for future repackaging and transportation needs.



危险

1 Not responsible for any damage incurred during shipping..

2 Make sure to strictly follow the installation instructions when installing the robot.

[← Previous Section](#) | [Next Chapter →](#)

Maintenance and Care

As a robotics manufacturer, we emphasize the importance of our customers being able to properly and safely maintain and upgrade their robotic equipment. Therefore, we provide a detailed maintenance and care guide, including common maintenance tasks and sections on repairing or upgrading hardware. Please read carefully for:

1 Recommended cycles.

Maintenance Item	Description	Recommended Frequency
Visual Inspection	Check for obvious damage, foreign object accumulation, or wear.	Daily
Structural Cleaning	Clean robot structural parts with a clean, dry cloth. Avoid moisture and corrosive cleaners.	Daily
Fastener Check	Check and tighten all bolts and connectors.	Daily
Lubrication	Lubricate joints and moving parts with manufacturer-recommended lubricant.	Every 3 months
Cable and Wiring Check	Inspect cables and wiring for damage or wear.	Monthly
Electrical Connection Check	Ensure all electrical connections are secure and undamaged.	Monthly
Software Update	Check and update control software and applications.	As updates are available
Software Data Backup	Regularly backup key software configurations and data.	Quarterly
Firmware Update	Regularly check and update firmware for new features and security patches.	As updates are available
Sensor and Device Check	Inspect sensors and other critical devices for proper operation.	Monthly
Emergency Stop Function Test	Regularly test the emergency stop function to ensure its reliability.	Monthly

Maintenance Item	Description	Recommended Frequency
Environmental Conditions Monitoring	Monitor temperature, humidity, dust, etc., in the working environment to meet robot operating specifications.	Continuous monitoring
Safety Configuration Review	Regularly check and confirm the robot's safety configurations, such as speed limits and work range settings.	Monthly
Preventive Maintenance Plan Execution	Perform regular checks and maintenance according to the manufacturer's maintenance plan.	According to manufacturer guidelines

2 Repair and Modification

We understand customers may need to upgrade or repair their robot hardware themselves. Before any upgrade, please read product specifications carefully and confirm with our officials if such operations are allowed. Unauthorized operations may lead to product failure and are not covered by the warranty.

Material Requirements

- **Officially Manufactured or Recommended Materials:** All parts and components for repairs and upgrades must be officially made or recommended by us.
- **Material Procurement:** Customers can purchase necessary materials through our official channels to ensure quality and compatibility.

Repair or Upgrade Process

- **Customer Repairs:** Customers are responsible for completing repairs, guided by our detailed manuals.
- **Follow Official Guidance:** Repairs must adhere strictly to our official guidance to prevent damage.

Liability and Warranty Policy

- **Liability Division:** Manufacturer provides official guidance and deals with defects; customers are responsible for following this guidance using official parts.
- **Warranty:** Valid only when repairs follow our guidance using official parts.

Considerations

- **Safety First:** Follow all safety guidelines before any repair or upgrade, including powering off and using proper protective gear.
- **Technical Support:** If issues arise during repairs, contact our technical support team for help.

Customers are strongly advised to follow these guidelines to ensure the safe and effective operation of their robot equipment.

[← Previous Section](#) | [Next Chapter →](#)

Common Problem

In this part, some common driver-related problems, software-related problems and hardware-related problems are listed.

[1 How To Ask Questions Gracefully](#)

[2 Drive Related](#)

[3 Software Problem](#)

[4 Hardware Problem](#)

If you have purchase intention or any parameter questions, please send an email to this mailbox.

[E-mail : "sales@elephantrobotics.com"](mailto:sales@elephantrobotics.com)

If the listed problems can't help you solve and you have more after-sales questions, please send an email to this mailbox.

[E-mail : "support@elephantrobotics.com"](mailto:support@elephantrobotics.com)

[← Previous Section](#) | [Next Chapter →](#)

First installation and use



- **Thank you for choosing our product**

Before we begin, we would like to sincerely thank you for choosing our product. We are committed to providing you with an excellent user experience.

- **First time use and problem handling**

This chapter will introduce in detail the initial use of the product after receiving it, and provide relevant information for solving problems to ensure that you have no worries during use.

- **Jump to each section**

- [4.1 Product Standard List](#)
- [4.2 Product Unboxing Guide](#)
- [4.3 Power-on Test Guide](#)

[← Previous Chapter](#) | [Next Chapter →](#)

Product Standard List

1 Product List Picture



Each product is numbered and detailed to ensure you can accurately refer to your listing.

2 Product Standard List Comparison Table

Serial number	Product
1	Mercury_A1
2	Product Brochure
3	Product kit, including USB-Type C, jumper, M6*35, cup head hexagon, fully threaded, stainless steel screws
4	24V, 9.2A supporting power supply
5	Aluminum alloy base
6	G-type clip*2
7	E-Stop Key

Note: After the packaging box is in place, please confirm that the robot packaging is intact. If there is any damage, please contact the logistics company and the supplier in your region in time. After unpacking, please check the actual contents in the box according to the item list.

[← Previous Page](#) | [Next Page →](#)

Product unboxing guide

1 Product unboxing graphic guide

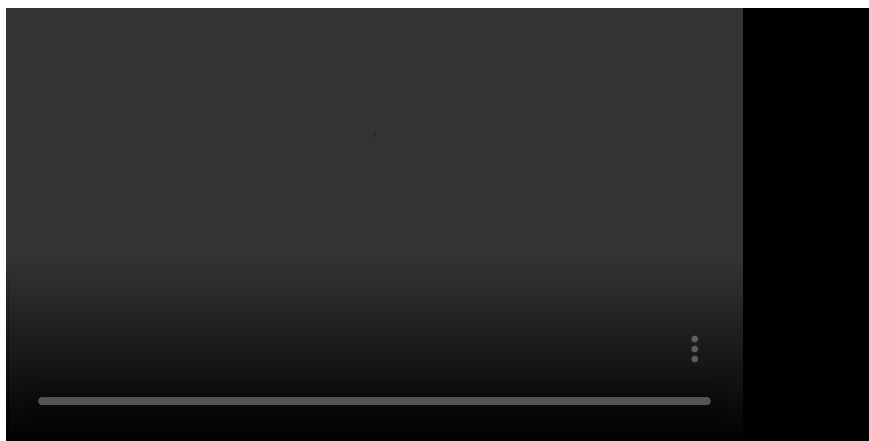
Why you need to follow the steps to remove the product

In this section we strongly recommend following the specified steps to remove the product. Not only does this help ensure that the product is not damaged during shipping, it also minimizes the risk of unexpected failure. Please read the following graphic guide carefully to ensure that your product is safe during the unboxing process.

- **1** Check whether the box is damaged. If there is any damage, please contact the logistics company and the supplier in your region in time.
- **2** Open the box and take out the product brochure, sponge packaging cover, Mercury robotic arm, supporting power supply, emergency stop switch, flat base, and accessory package.
- **3** Make sure each step is completed before proceeding to the next to prevent unnecessary damage or omissions.

Note: After removing the product, please carefully inspect the appearance of each item. Please check the actual items in the box against the item list.

2 Product unboxing video guide



[← Previous Page](#) | [Next Page →](#)

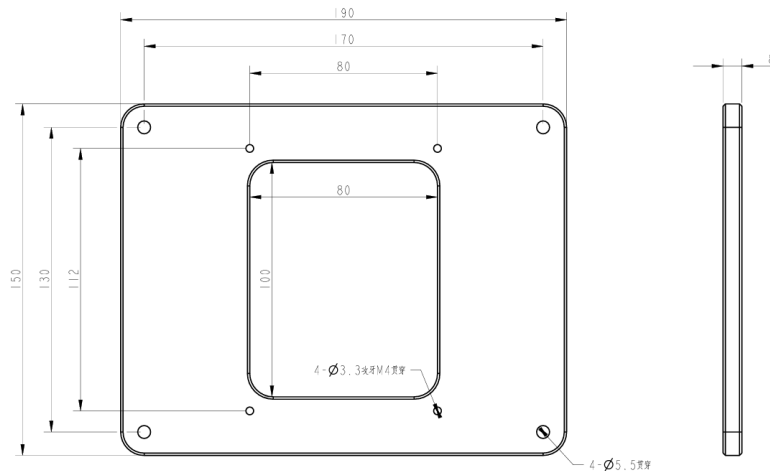
Boot detection guide

1 Structural installation and fixation

myCobot weights 3.5 kg. Due to the fact that the center of gravity will change along with the movement of the robot during utilization, robot is required to be fixed on a solid base at the beginning. A fixed base, or mobile base are both acceptable.

Base Interface Size

- The base fixing holes act as the interface between robot and other bases or planes. The specific hole size is shown in the figure below. There are 4 countersunk holes with a diameter of 4.5 mm, which can be fixed with M6 bolts.



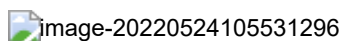
- There is a flange installed on the end. Please make sure there are corresponding threaded holes on the fixed base before installation.

Before the installation, please confirm:

- The environmental condition meets the requirements listed in Section 2.2.1.3.1 above.
- The installation position is no smaller than the working range of the robot, and there is enough space for installation, use, maintenance and repair.
- Put the base in a suitable position.
- Installation-related tools are prepared, such as screws, wrenches, etc.
- After confirming the above, please move the robot to the base installation table, adjust the robot position, and align the fixing holes of the robot base with the holes on the base mounting table. After aligning the holes, align the screws with the holes and tighten them.

Notice: When adjusting the position of the robot on the base installation table, do not pushing or pulling the robot directly on the base installation table to avoid scratches. When manually moving the robot, do not applying external force to the fragile parts of the robot body, so as to avoid unnecessary damage to the robot.

For more installation details, scan the code to watch the video:



2 External cable connection

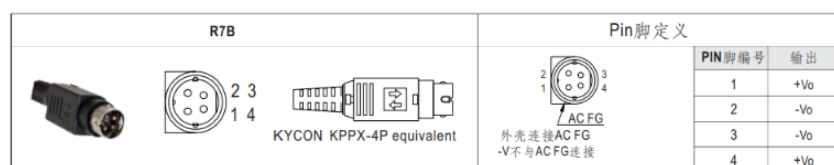
Before operation, confirm that you have read **Chapter Safety Instructions** to ensure safe operation. At the same time, connect the power adapter with the robotic arm, and fix the base of the robotic arm on the table.

myCobot must be powered on with an external power supply to provide sufficient power:

- Rated voltage: 24V
- Rated current: 9.4A
- Plug Type: R7B

■ Dc输出插头

● 标准插头: R7B



Note that you cannot just use the TypeC plugged into the M5Stack-basic for power. Use an officially adapted power supply to avoid damage to the robotic arm.

The use case diagram is shown in the following figure: (Please carefully align it with the use case diagram for connection)

Step 1:



Step 2:



Step 3:



Step 4:



Step 5:



Step 6:



3 Power on status display

Make sure that the power adapter and emergency stop switch are connected, press the power switch **Start button (round)**, then the **LED light panel of ATOM at the end of the robot arm will light**, and the **LED light panel at the bottom of the robot arm will light up**. **BASIC interface** will display **Atom communication status**.




4 Basic function detection

When performing this operation, please refer to section [5.1-4 Robot Information](#) for guidance. Prior to proceeding, ensure that you have followed the electrical connection instructions mentioned above and confirm that your device is securely installed. Failure to properly connect the cables or secure the device may result in accidents. Thank you for your cooperation.

[← Previous Page](#) | [Next Chapter →](#)

Basic Application

 <p>Hazard</p>	<p>This chapter primarily provides explanations for the basic functionality usage of the product and the use of basic software. This chapter is essential and should be read carefully. Please ensure a correct understanding of the described operations before proceeding with actual robot applications.</p>
---	---

For the Mercury A1 model, we have launched software specifically designed for robot application and maintenance for our users to use. Among them, RoboFlowLite and myStudio are necessary application layer integration software for users to use robots. This chapter will introduce in detail how to use these two softwares.

Software Description:

- [myPanel](#)

`myPanel` is an application that controls the Mercury robot arm by touching the cm4 on the base of the robot arm. Its functions are coordinate control, Angle control, drag teaching, etc. which is convenient for operators to interact with elephant robots and correctly use elephant robots

- [myStudio](#)

`myStudio` is a one-stop platform for using our company's robotic products. It supports users to download/burn/update robot embedded firmware; it has embedded Blockly visual programming application to facilitate users to control robots through visual programming; it supports rapid movement control of single joints or single coordinates of robots; it supports fast control of robot IO input and output; Supports real-time monitoring of robot status and other functions. **Under development, not online yet, please stay tuned**

[← Basic Application](#) | [System →](#)

System instruction manual

5.1.1 Robot system introduction

System

- Ubuntu is the most widely used linux operating system for personal desktop operating systems. For beginners, familiarity with the linux environment or some embedded hardware operating system is a good choice. The official ubuntu website has also released a dedicated operating system for the Raspberry PI.



- **Function introduction**
 - **myStudio**: Firmware burning software for updating and burning new firmware.
- **myBlockly**: Graphical programming software, can be directly by dragging building blocks to form running code, control the robot arm
- **ROS1 Shell**: Directly enter the compiled ROS1 environment, you can directly enter the corresponding instructions, run the corresponding ROS1 code
- **ROS2 Shell**: Directly enter the compiled ROS2 environment, you can directly enter the corresponding instructions, run the corresponding ROS2 code
- **Github-ElephantRobotics**: Elephant robotics official open source code repository
- **Home-ElephantRobotics**: Elephant robotics website
- **UserManual - CN/EN**: Robot use manual, contains everything about robot control
- **WiFi_ON/OFF**: WiFi switch, click to turn on/off the WiFi function
- **HotSpot_ON/OFF**: Hotspot switch, click to turn on/off the hotspot function, the hotspot name is **ElephantRobotics_A1_AP_XXXX**

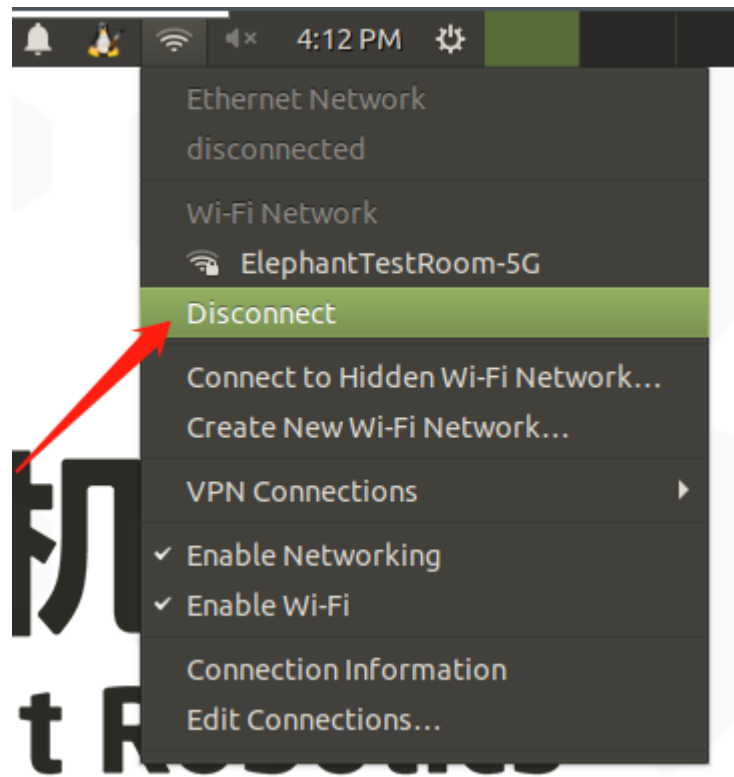
- **Language Support:** System language setting, click to enter the system language setting interface

5.1.2 System password description

- **account password & VNC password & SSH password & root account password**
 - Password: **Elephant**
- **How to change password**
 - change account password
 - Use the shortcut key `ctrl + alt + T` to open the terminal
 - Enter `passwd`
 - Enter the new password twice
 - change vnc password
 - Use the shortcut key `ctrl + alt + T` to open the terminal
 - Enter `vncpasswd`
 - Enter the new password twice
 - change ssh password
 - For SSH remote connection, the password of the administrator account is entered. You do not need to change the password separately
 - change root account password
 - Use the shortcut key `ctrl + alt + T` to open the terminal
 - Enter `sudo passwd`
 - Enter the new password twice

5.1.3 VNC

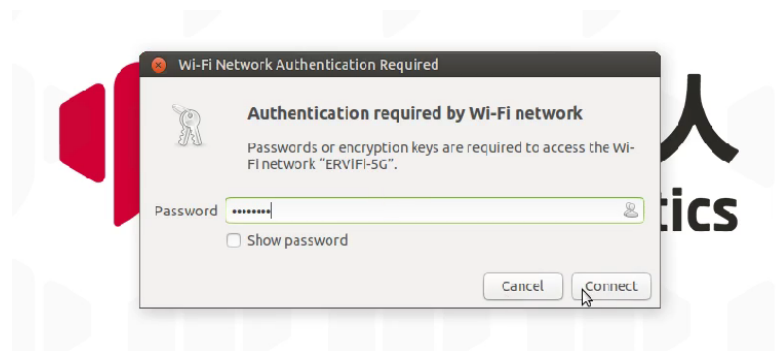
- **VNC Introduction**
 - Is a remote control software, generally used to remotely solve computer problems or software debugging
- **VNC Port**
 - When the robot arm and PC are connected to the same WiFi, the IP address of the robot arm is the port
- **Connect VNC**
 - There are two wireless connection modes. The first mode requires an external monitor to do some operations on the system. The specific steps are as follows: Click **“Disconnect”**, disconnect the default hotspot connection



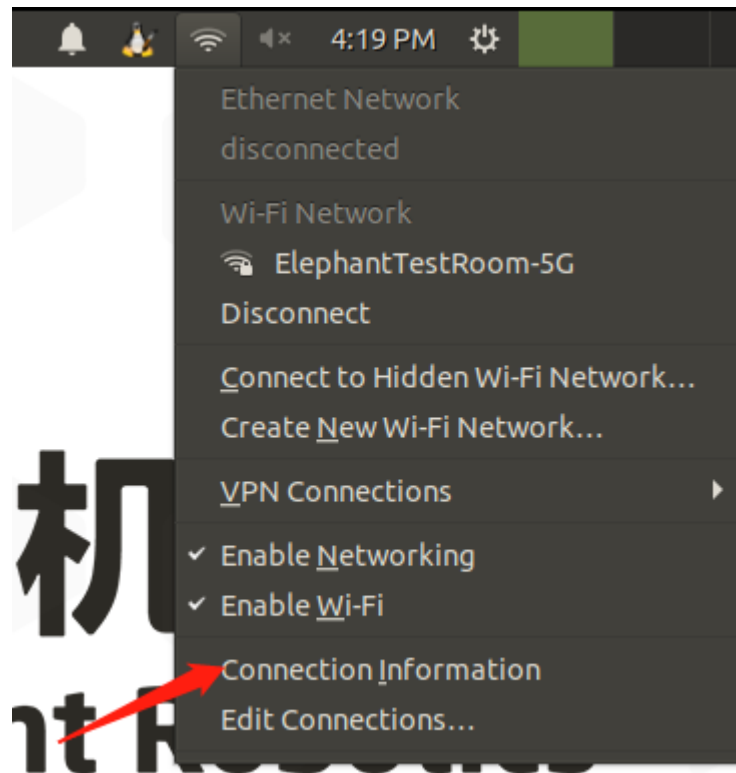
Click "**Enable Wi-Fi**" , and the currently available WiFi will appear



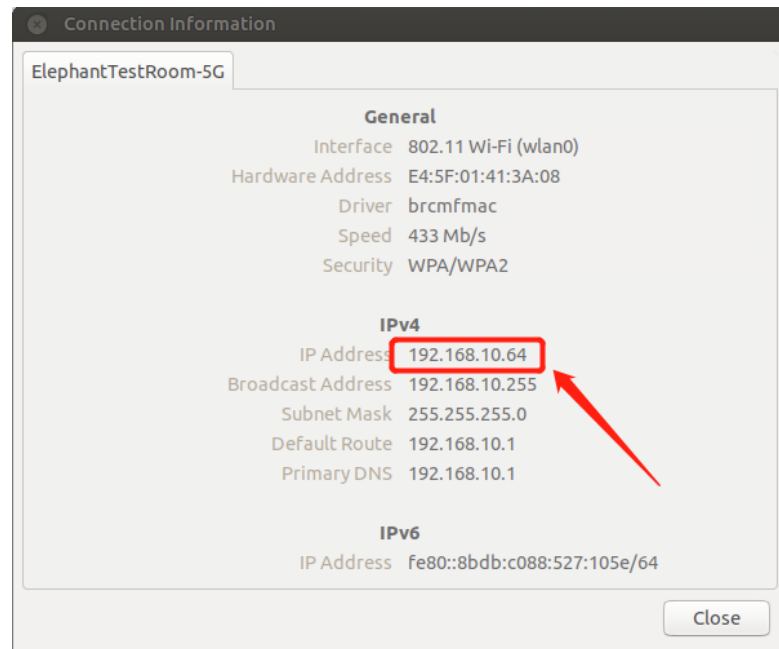
Click on the WiFi you need to connect to, enter the password



After connect successfully, click **"Connection Information"** to check IP address of the robot



As shown in the example, “**192.168.10.64**” is the current IP address of the robot

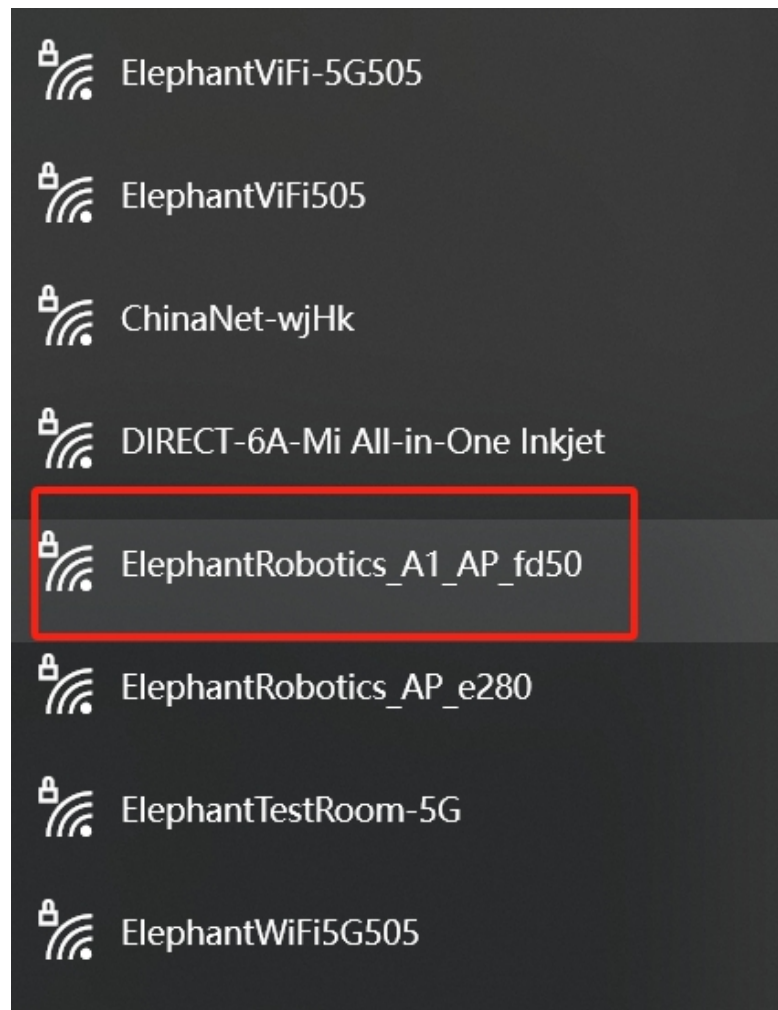


Connect your PC and the robot to the same WiFi, open the VNC viewer, enter the IP address (examples : input **192.168.10.64**) , enter the password **Elephant**, The user name is not specified by default. The following is an example of a successful connection :



- o The second method doesn't need to connect the monitor, directly connect the Ubuntu system hotspot with a PC for remote control, but this connection method does not have the function of Internet surfing, and can only remotely control the robot arm system. The specific steps are as follows:

Connect to the hotspot **ElephantRobotics_A1_AP_XXXX**, enter the password **Elephant**



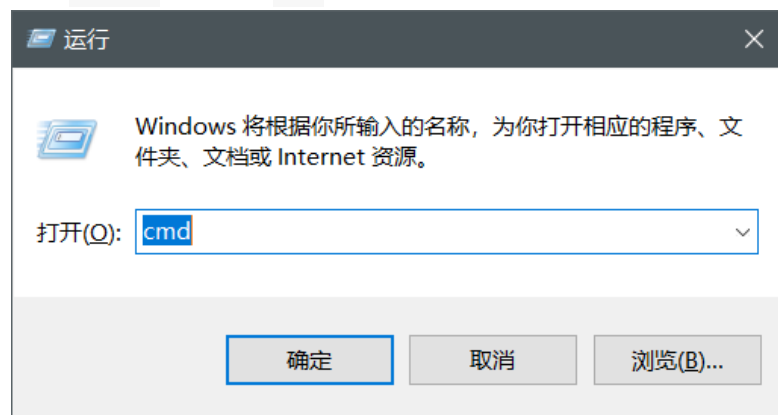
Open VNC viewer, input IP address **10.42.0.1** , enter, and then enter the password **Elephant**, The user name is not specified by default. The following is an example of a successful connection:



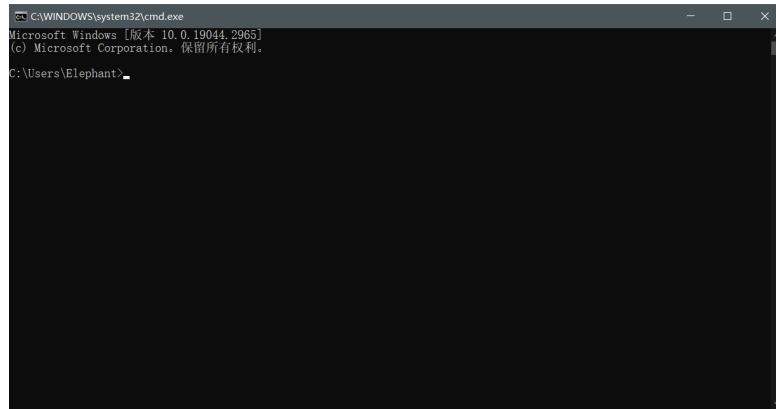
- **How can I improve connection fluency**
 - The fluency of the remote connection depends on the fluency of the connected WiFi. It is recommended to connect to a stable WiFi for remote control

5.1.4 SSH

- **SSH Introduction**
 - SSH is a network protocol used for encrypted logins between computers. If a user logs in from a local computer to another remote computer using SSH, we can assume that this login is secure, even if it is intercepted in the middle, the password will not be disclosed.
- **SSH Port**
 - The default port number is 22
- **SSH Connect**
 - Confirm the IP address of the robot by following instructions in **5.1.2 VNC**
 - Click `win + R` and enter `cmd`



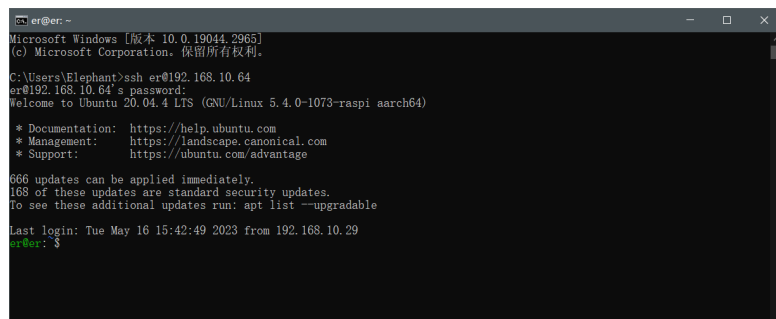
- o After input, click OK to open the shell interface



- o Enter `ssh er@IP address` , then press `Enter` (the IP address is mainly displayed by the robot arm, the figure is only an example)



- o Enter password `Elephant`



- o As shown in the above figure, the robot arm has been successfully connected by ssh
- How can I improve connection fluency
 - o The fluency of the remote connection depends on the fluency of the connected WiFi. It is recommended to connect to a stable WiFi for remote control

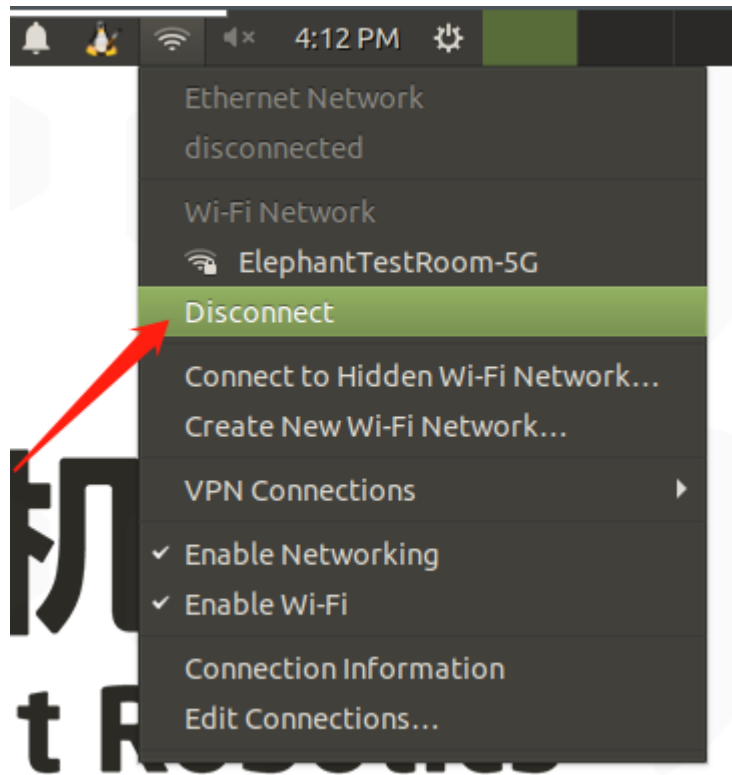
5.1.5 Network configuration

- Default AP usage
 - o Power on the robot, by default, the system will connect to the hotspot generated by the PI itself, the hotspot name is `ElephantRobotics_A1_AP_XXXX`, current IP address `10.42.0.1`, this hot spot does not have the function of web surfing, and the transmission rate and information is limited, so there will be some distortion and color

difference in the final imaging, and there will be a delay in communication transmission, which is a normal phenomenon

- **Connect to WLAN**

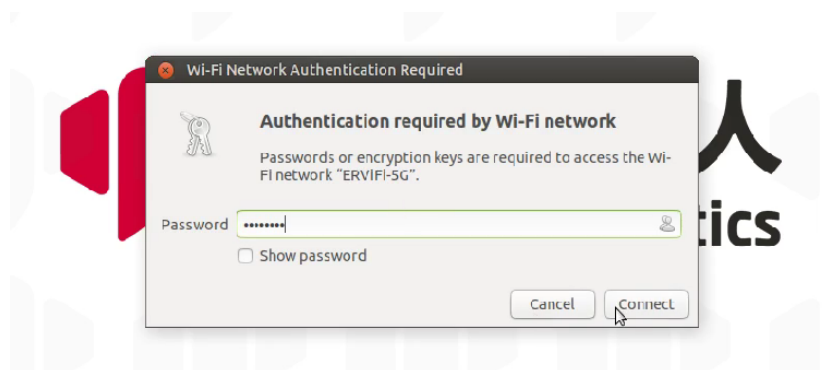
Click **"Disconnect"** to turn off default hotspot



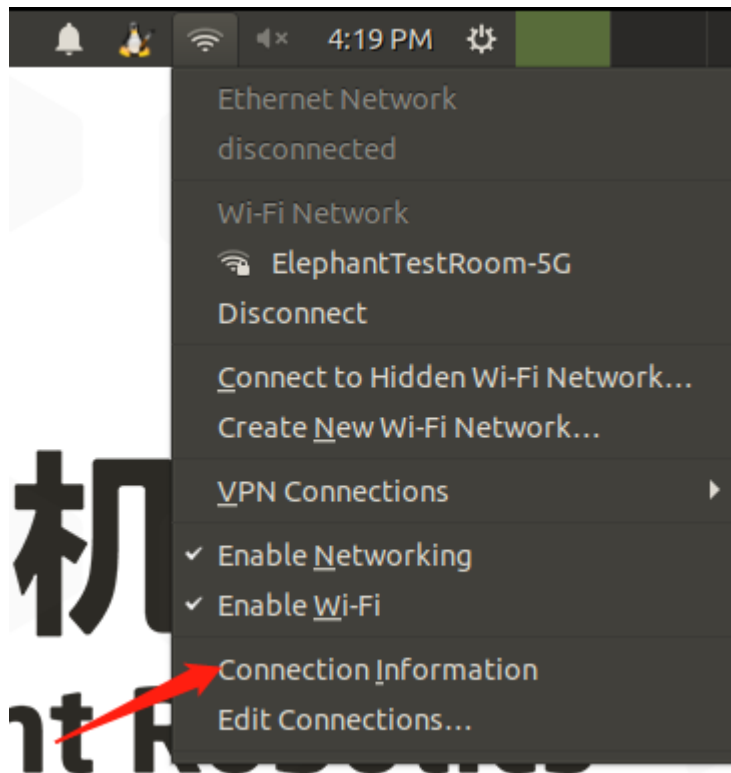
Click **"Enable Wi-Fi"** , wait for the currently available WiFi to be displayed



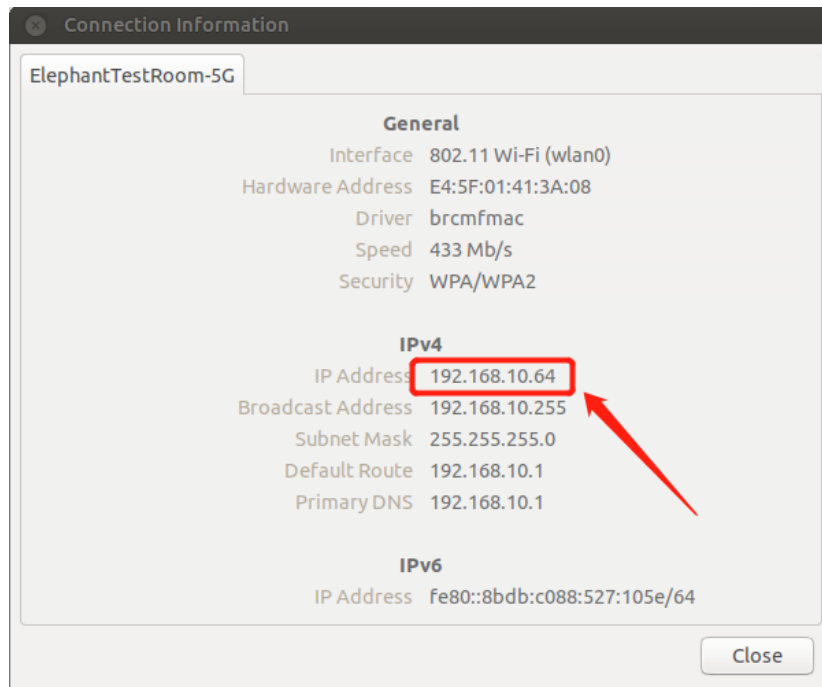
Click the WiFi you want to connect to and enter the WiFi password



After connect successfully, click **"Connection Information"** to check IP address



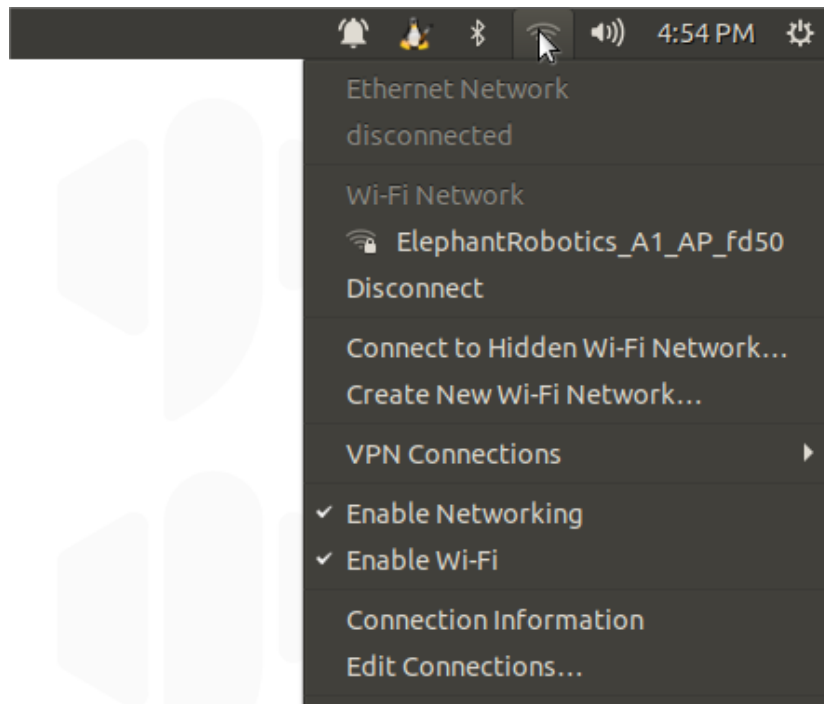
As shown in the example, “192.168.10.64”, It is the current IP address of the robot



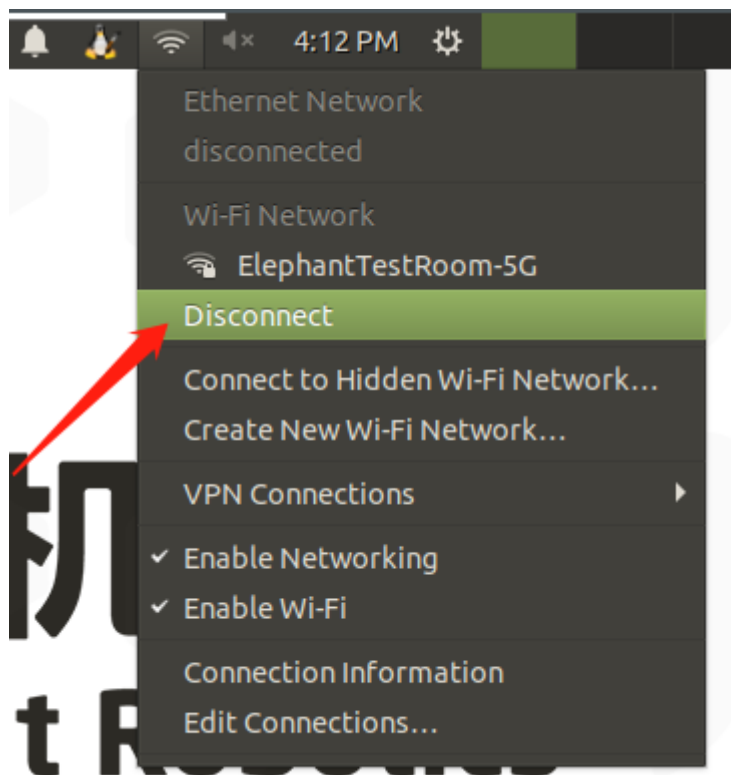
Wired network connection

Power on the robot, it is connected to the hotspot configured by the system by default:

ElephantRobotics_A1_AP_XXXX



Click “**Disconnect**”, disconnect from the default hotspot



How to set default IP address

[illegible]

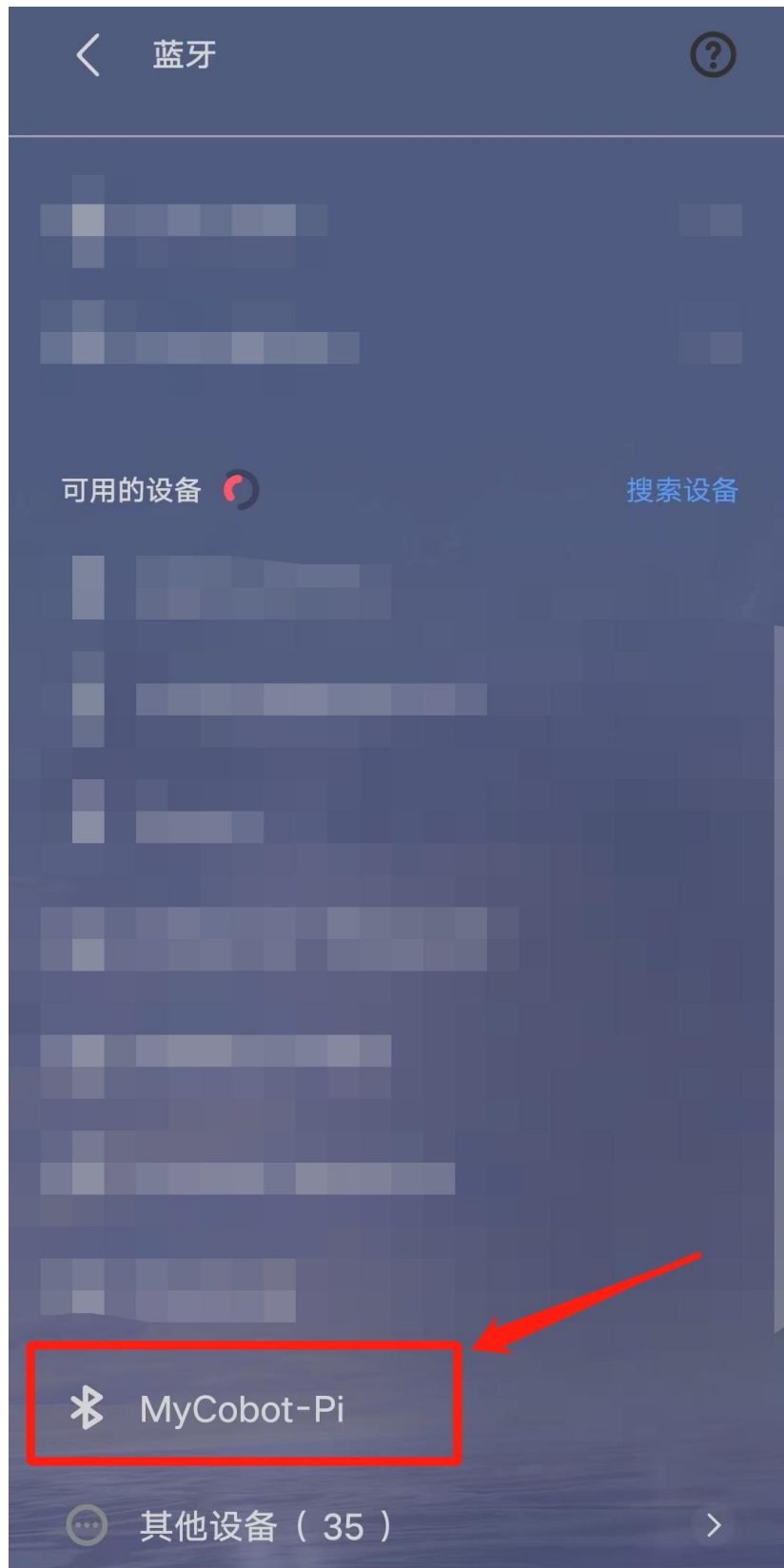
How to assign IP addresses automatically

```
er@er: ~  
File Edit View Search Terminal Help  
# Let NetworkManager manage all devices on this system  
network:  
  version: 2  
  renderer: NetworkManager
```

64

5.1.6 BT configuration

- The Bluetooth of the system is turned on by default. Use the PC/Phone directly , and you can search the robot's BT. The default name of the BT is **MyCobot-Pi**



Transfers files from PC/Phone to the robot system

- Select the file you want to transfer
- Operate in the robot system and click **Accept** to receive file



- o Wait for the BT transfer to complete



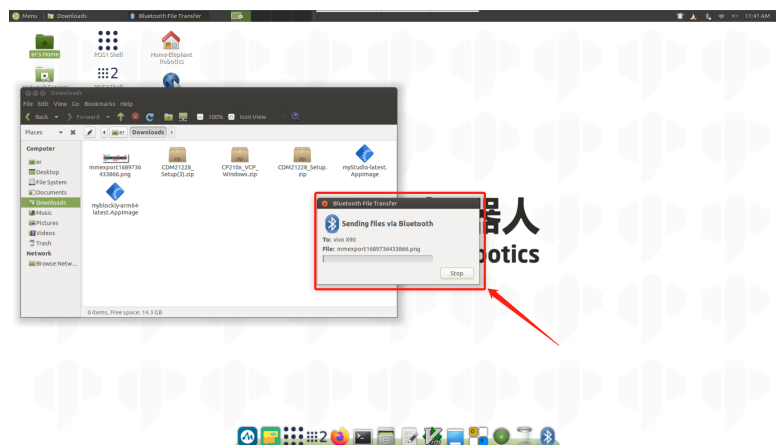
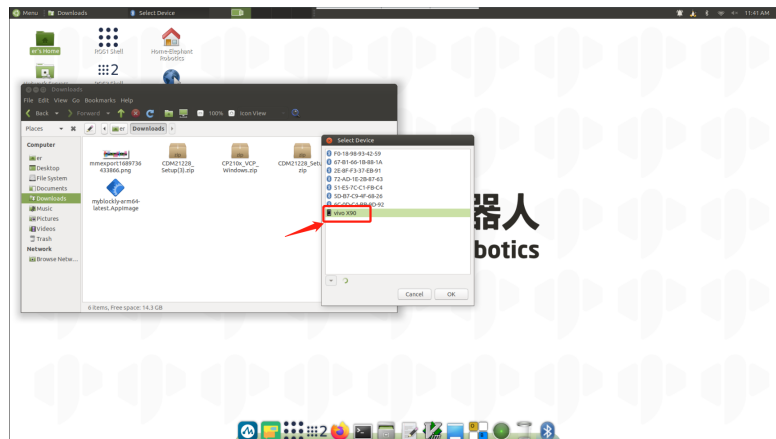
- o Can check received files in /home/er/Downloads folder



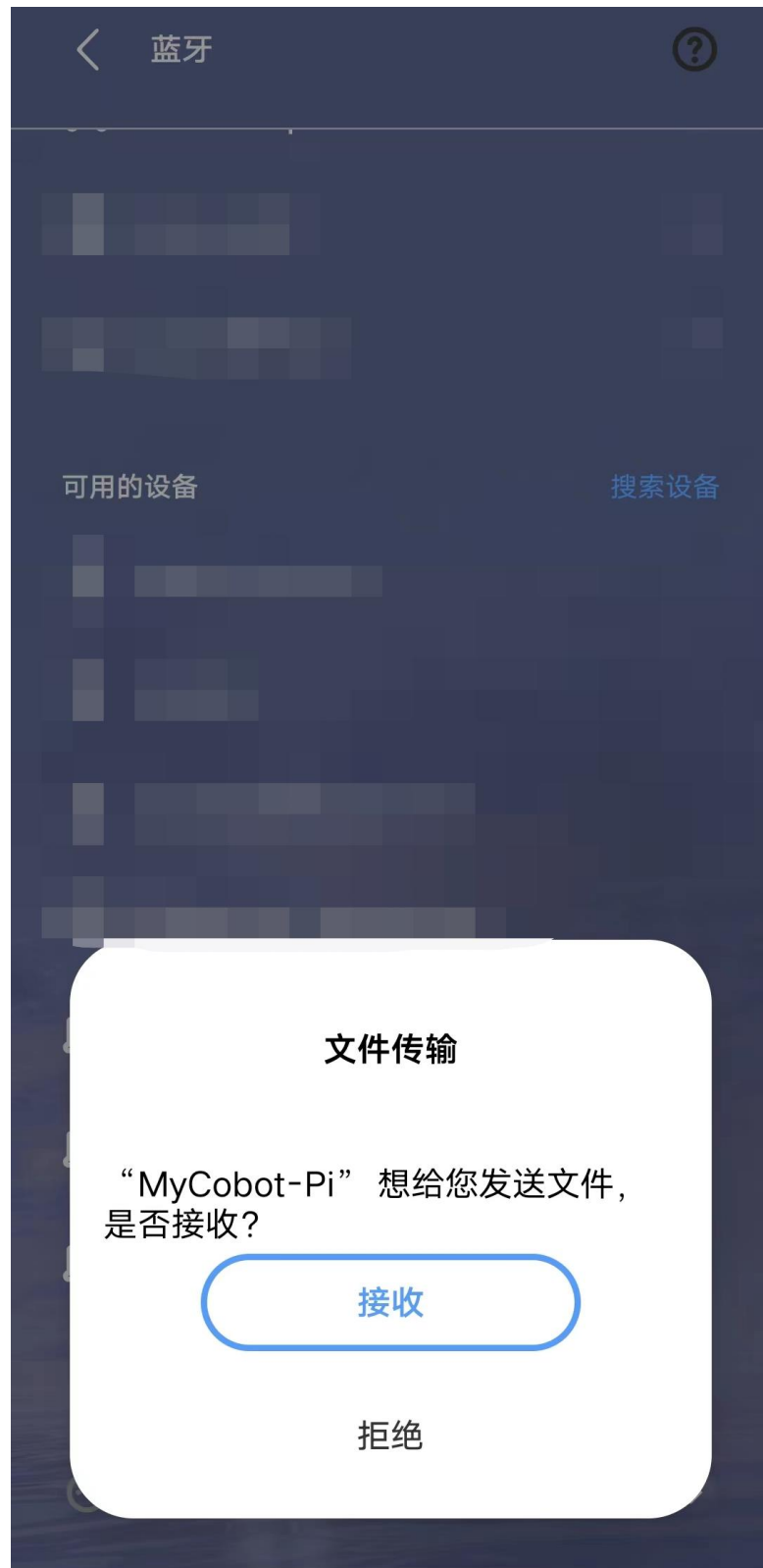
- Transfers files from the robot system to PC/Phone
 - o Click BT icon, select **Send Files to Device**



- o Choose PC/Phone



- o On PC/ mobile phone, allow the device to receive files



5.1.7 Language configuration

How to change language

Click **Language Support**, drag the language you want to the top and restart the system



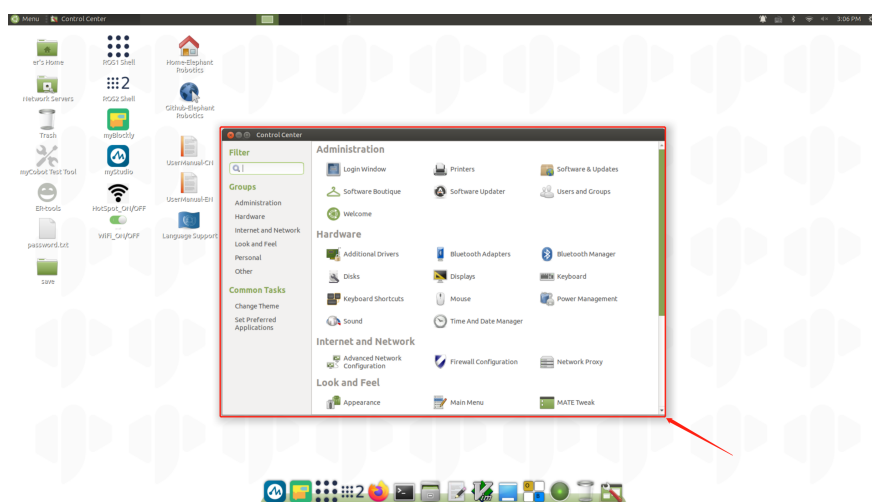
How to download language

Click **Language Support**, choose language, click to download, enter password **Elephant**

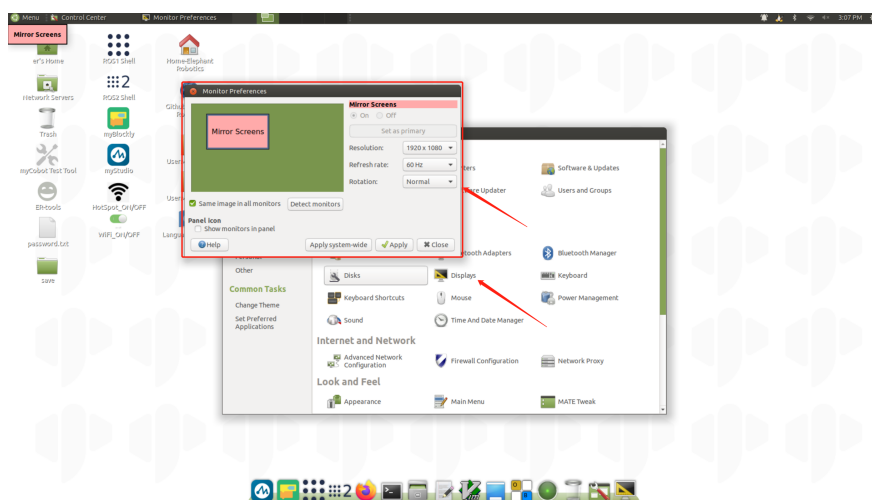


5.1.8 System resolution switch

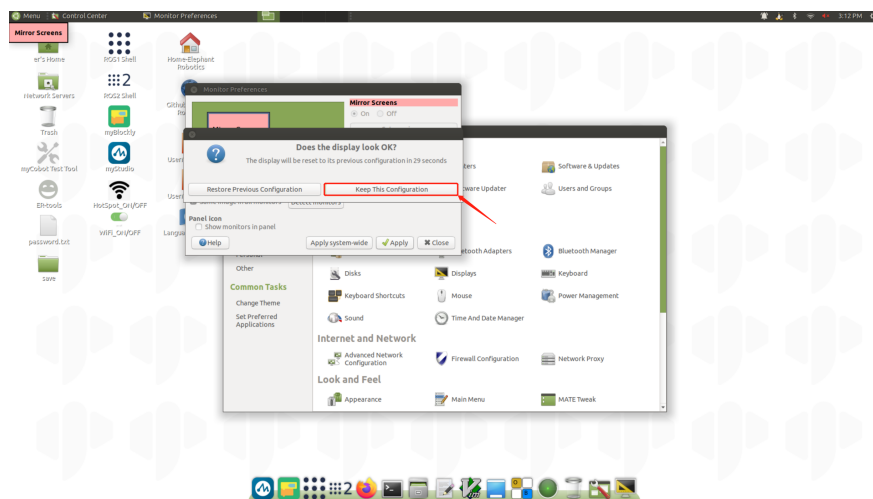
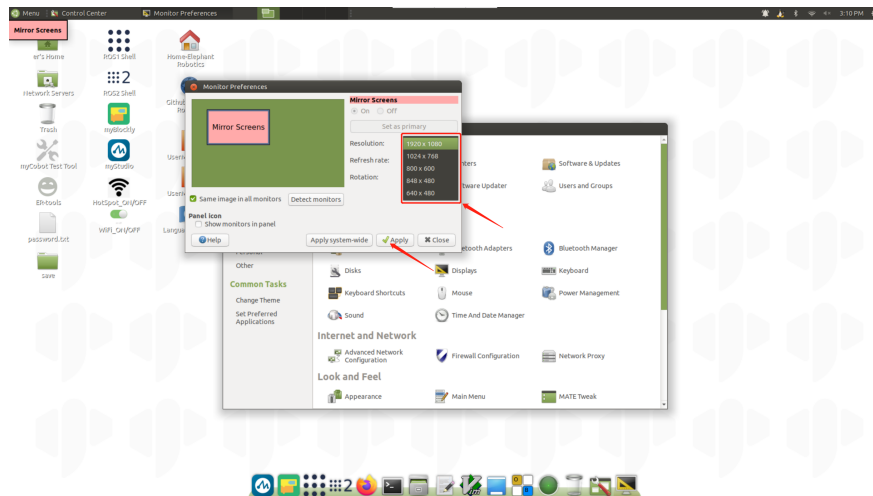
Click the icon in the upper right corner of the screen and select **System Settings** to enter the control panel



Choose Display



Toggle the selected resolution and click **Apply** to check and click **Keep this Configuration** to save the configuration



5.1.9 python

- **Python introduction**

Built-in installation **Python3.8**, no need to install it yourself

Installed libraries:

Package	Version
pymycobot	3.1.5
pyserial	3.5
numpy	1.23.5
opencv-contrib-python	4.7.0.72
rospkg	1.4.0
rospkg-modules	1.4.0

- **Try to program**

If you are new to the python programming language, you can follow the following video to program



One-stop service platform myStudio

[myblockly.](#)

Mercury Panel User Guide

1 Mercury Panel Introduction

Mercury Panel is an application that controls the Mercury robot arm by touching the cm4 on the base of the robot arm. Its functions are coordinate control, Angle control, drag teaching, etc.), which is convenient for operators to interact with elephant robots and correctly use elephant robots.

Applicable equipment:

-Mercury A1

Be sure to download the updated atom and basic firmware before use

2 How to use it

We have installed the panel in the system, and there is a shortcut on the desktop as shown in the figure.



Click the icon to run panel, and a message box will pop up in the upper right corner of the desktop (if it is opened, click to close it).



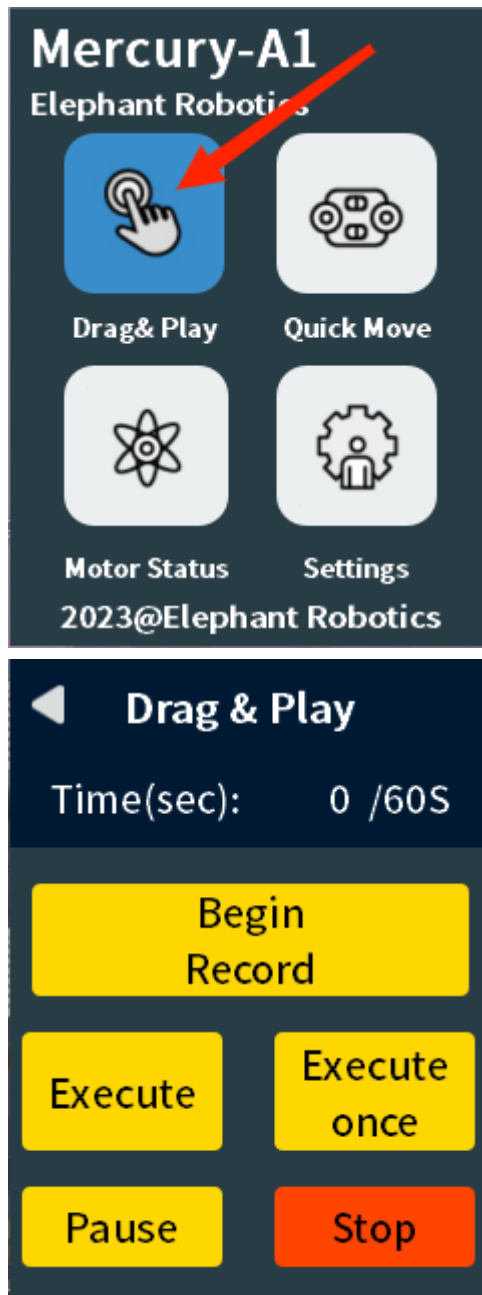
Click to close panel again, and a message box pops up in the upper right corner of the desktop.



When the panel is run, the user can intuitively see the button that can be clicked and the corresponding comment. The default startup is the power-on state (the rgb light at the end of the power-on state will be on). The gray triangle in the upper left corner of the interface is to return to the previous interface, and the x in the upper right corner is to close the interface.

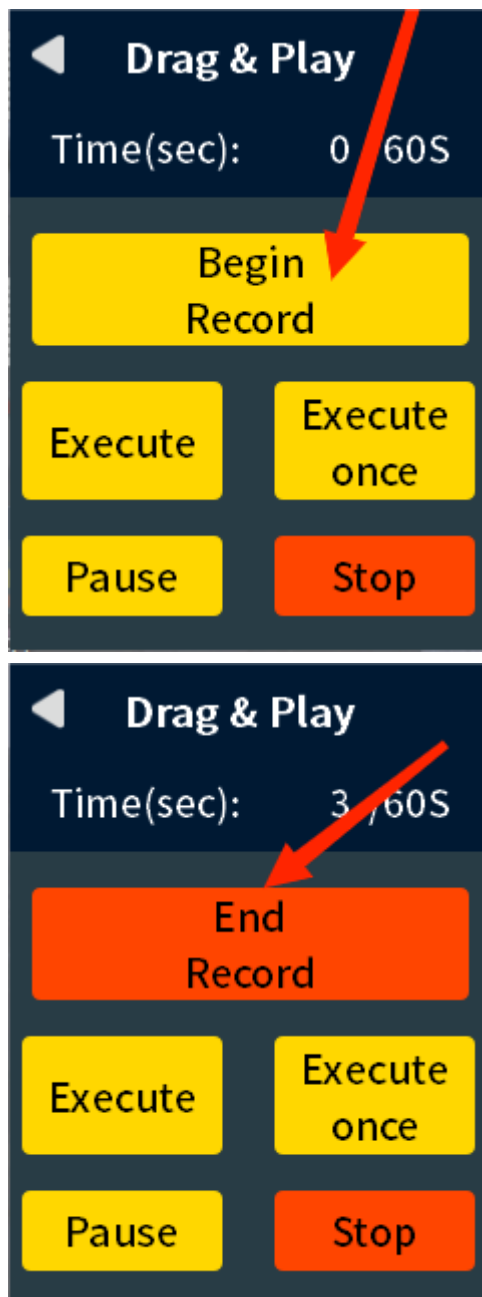
2.1 Drag & Play

Click drag to teach in the main interface to enter the drag to teach interface.

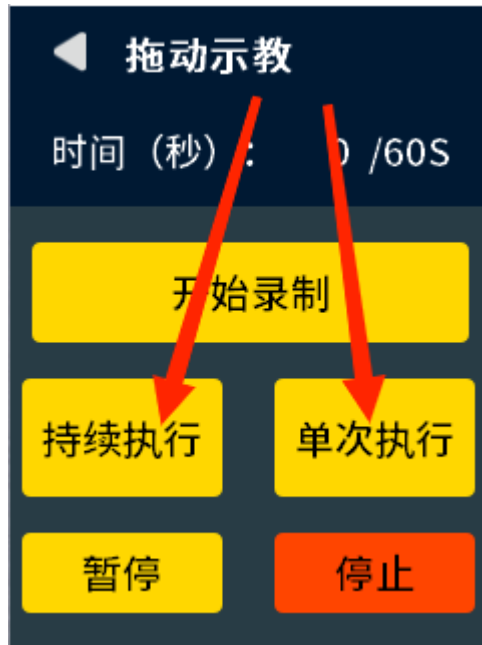


Click to start recording and pop-up window will remind you that the robot arm is about to be relaxed. Click OK to relax the robot arm and start dragging the recording teaching point. When recording, the recording time will be displayed on the top in real time (the recording time shall not exceed 60 seconds, otherwise there will be a pop-up warning), click to end the recording after recording, and the mechanical arm will be locked.

If you click Record in case of power failure, the system prompts you to power on first. Click OK.



Click Single execution to play a recorded action; Click Continue to play the recording action in a loop.



Click Stop during the movement of the robot arm, the robot arm stops moving, and the recorded point is cleared.

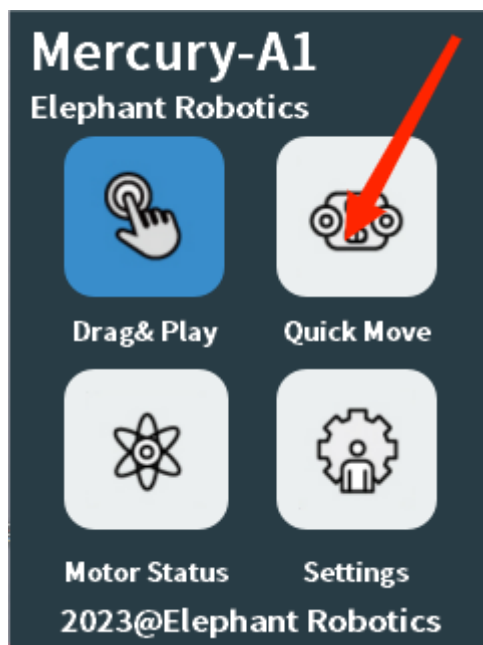


Click pause during the movement of the mechanical arm, the mechanical arm stops moving, the pause button changes to resume, and the mechanical arm continues to move along the previous break point.



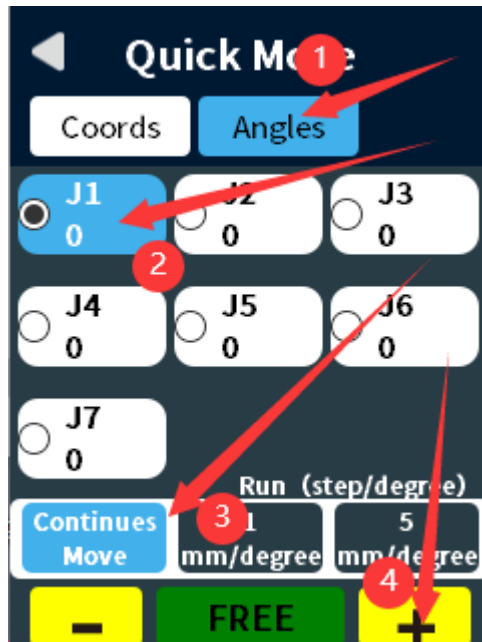
2.2 Motion control

Click "Quick Move" in the main interface to enter the "Quick Move" interface and select Angle control or coordinate control.

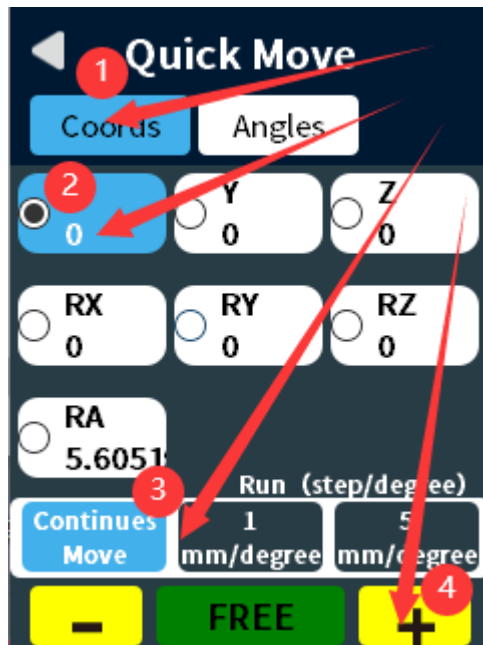


Angle control: Select the joint to be operated, and the control is divided into continuous motion and stepping motion. Select continuous under run, this is continuous movement, after pressing the button, until the release button will stop (after reaching the limit will also stop); Select 1 mm/degree or 5 mm/degree (step value) under run, then it is a step movement, click + to increase the corresponding step value, click - to reduce the corresponding step value, the step

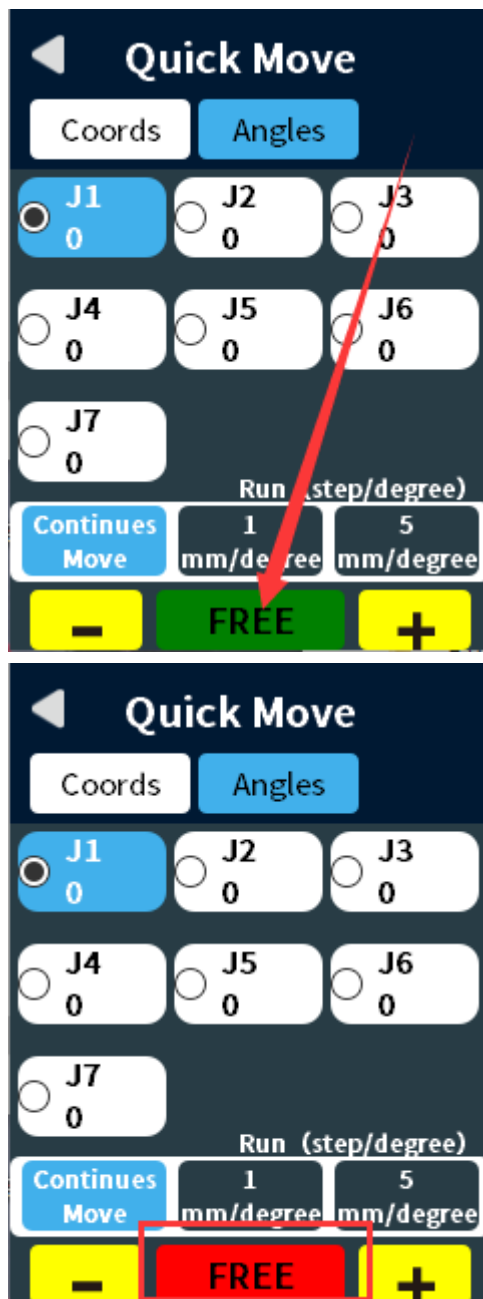
movement will stop when it reaches the point (for example, joint 1 is at an Angle of 30 with an incremental value of 50, press the button on the side of the minus sign, then the robot arm will stop when it moves to -20).



Coordinate control: Adjust the robot arm to a proper attitude, such as joint 4: -90° , before performing coordinate control. Select the axis to operate, the control is divided into continuous motion and step motion. Select continuous under run, this is continuous movement, after pressing the button, until the release button will stop (after reaching the limit will also stop); Select 1 mm/degree or 5 mm/degree (step value) under run, then it is a step movement, click + to increase the corresponding step value, click - to reduce the corresponding step value, the step movement will stop when it reaches the point (for example, the X-axis is now 30, the incremental value is 50, press the button on the side of the plus sign, then the robot arm will stop when it moves to 80). RA is the redundant arm Angle.

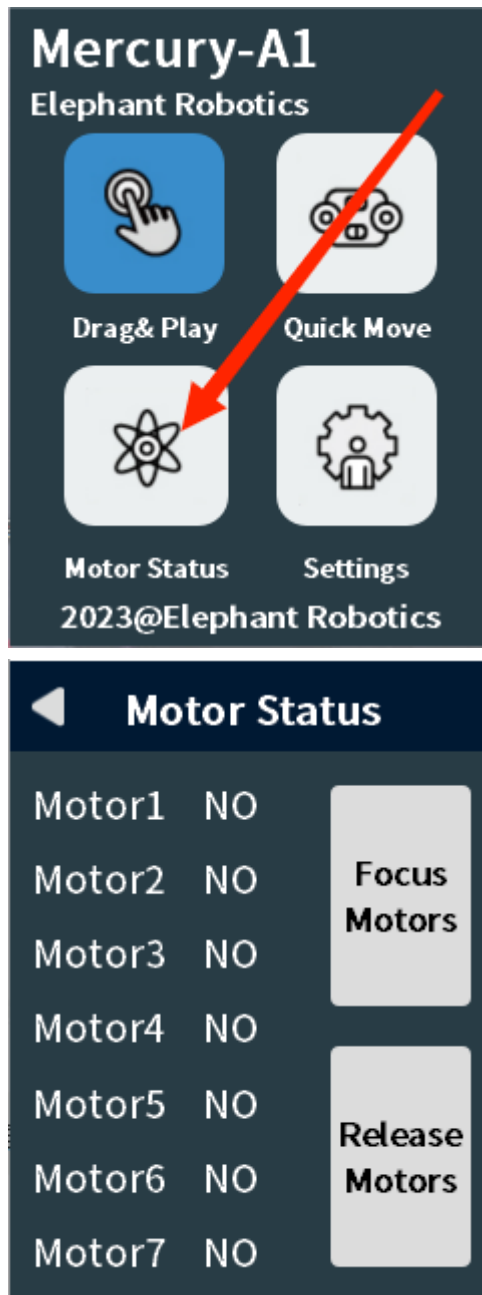


Free movement: Press the free movement button to enter the free movement mode, and the rgb light of atom changes color. Keep pressing the rgb button at the end of the robot arm to relax, and the whole robot arm can move freely and release the whole lock of the rgb button at the end. Click Free Move again to exit Free Move mode.

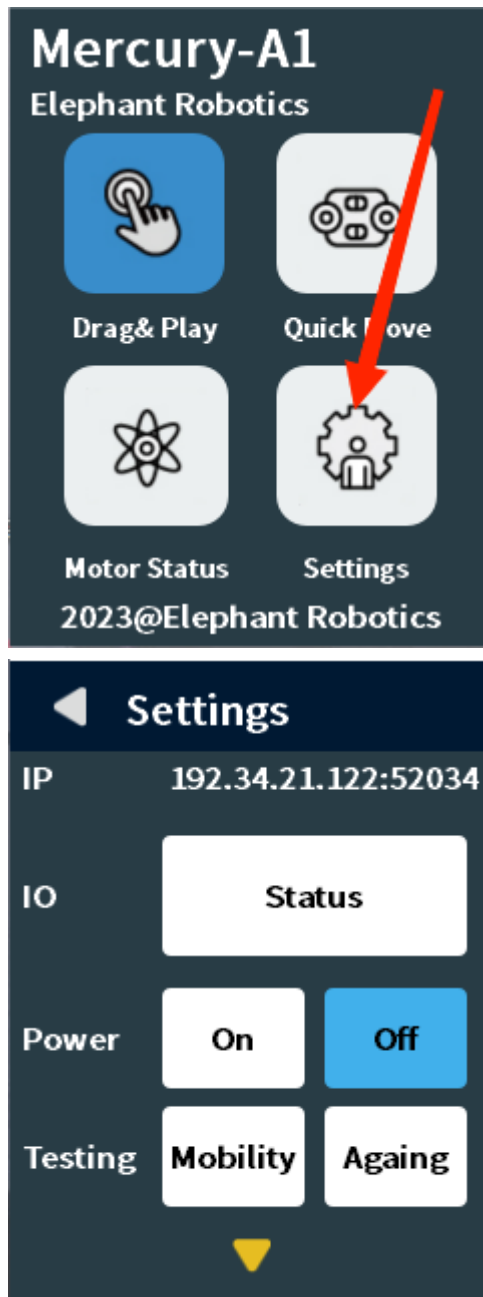


2.3 Overall operating state

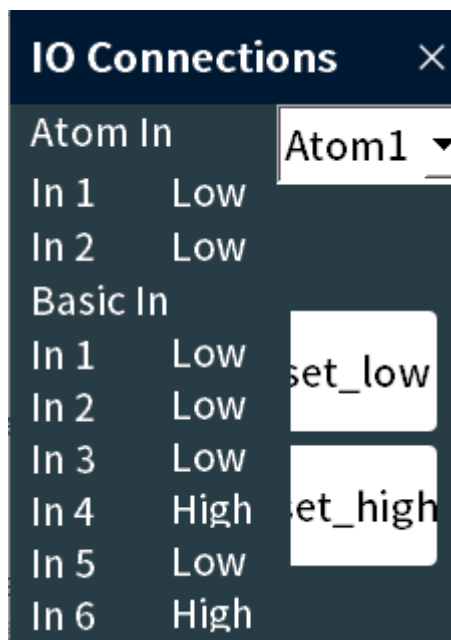
Click the motor status on the main interface to enter the motor status interface and check the motor connection status. The whole mechanical arm can be relaxed and locked by all locking and all relaxing.



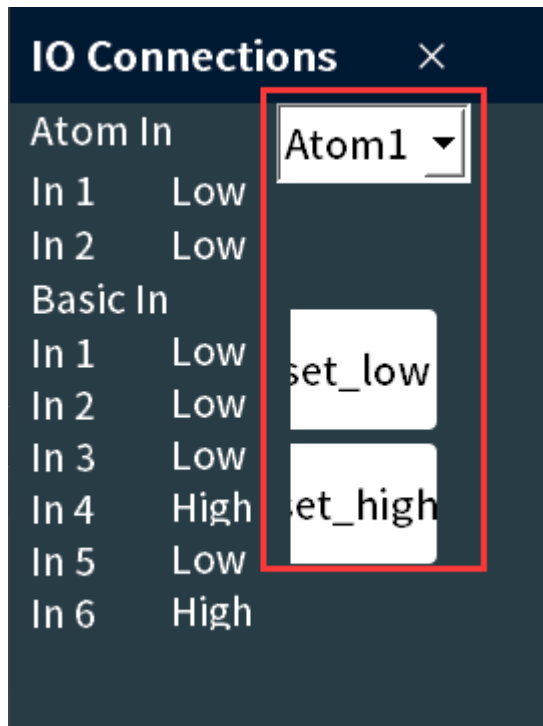
Click Settings on the main screen to enter the Settings screen.



On this screen you can click IO status to view the IO status corresponding to atom and basic.



IO status interface setting pin high and low level has not been developed.

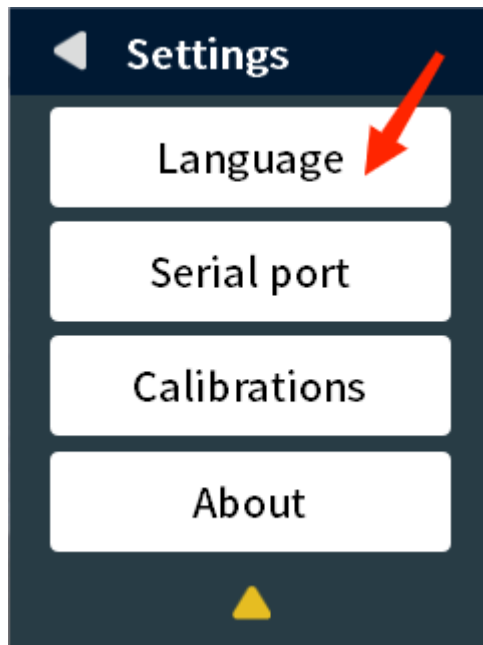


Click Quick test -> Multi-action, the robot arm starts a round of movement of multiple points, the button changes to stop, click stop, the robot arm stops moving; Click Quick Test -> Aging, the robot arm starts to cycle a group of large point movements, the button changes to stop, click stop, the robot arm stops after completing the current round of action.

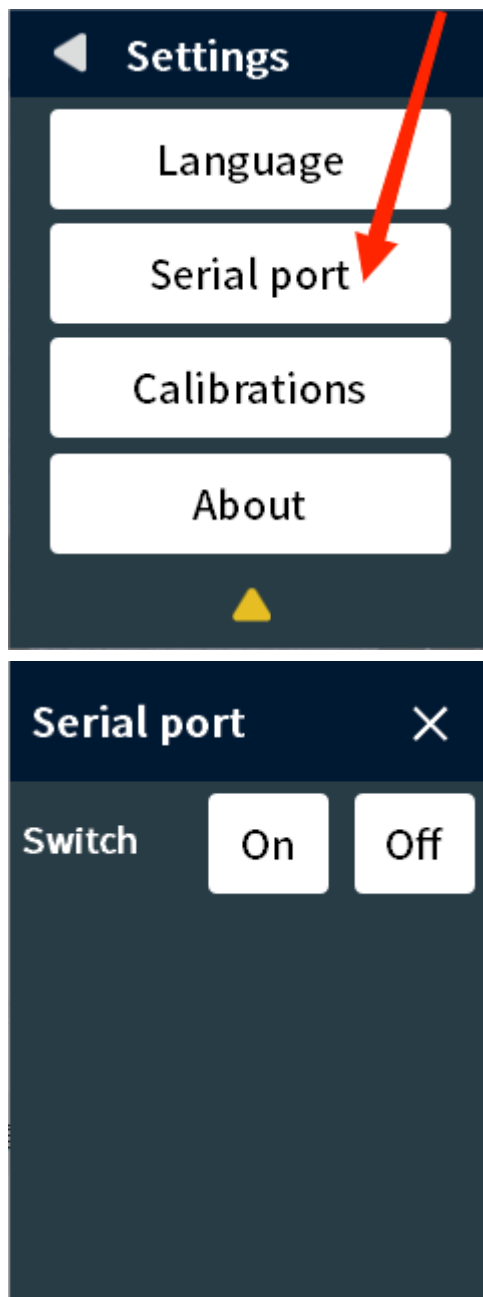
Click the yellow triangle and turn the page.



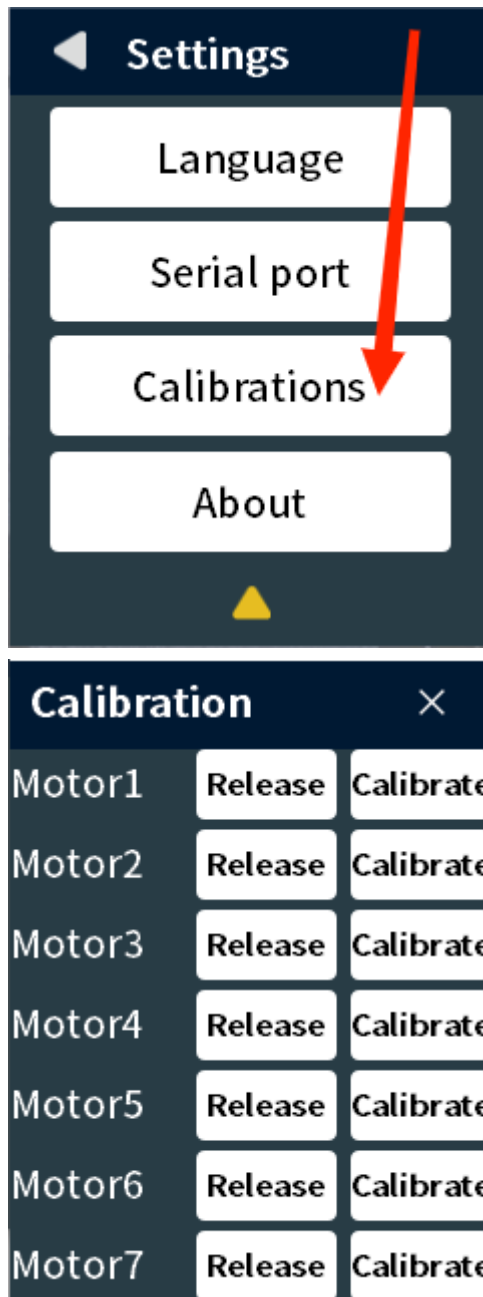
Tap Language to enter the language Settings screen, you can select the theme language of the application.



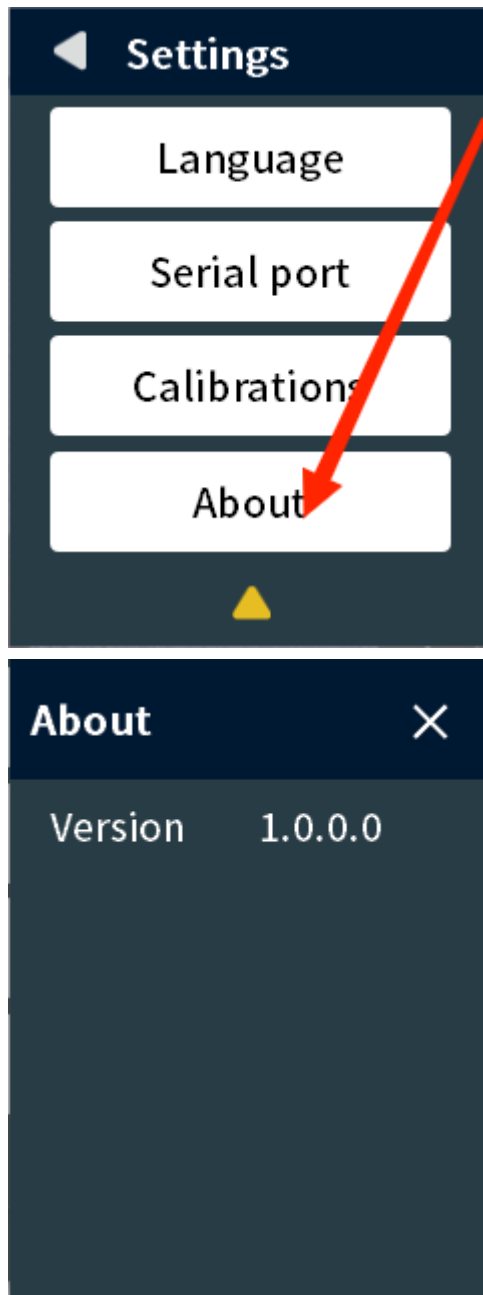
Tap the serial port to go to the serial port control screen. Click serial port connection -> Open to open the serial port; Serial port connection -> Off, close the serial port.



Click Calibrate to enter the calibration screen. You can perform a psychic calibration for each motor: click on the corresponding joint of the power off to relax the mechanical arm, drag the corresponding joint to the new zero position, click on the corresponding joint lock, and the calibration is completed.



Click About to enter the About screen to view the application version.



2.3 Mechanical arm status protection prompts

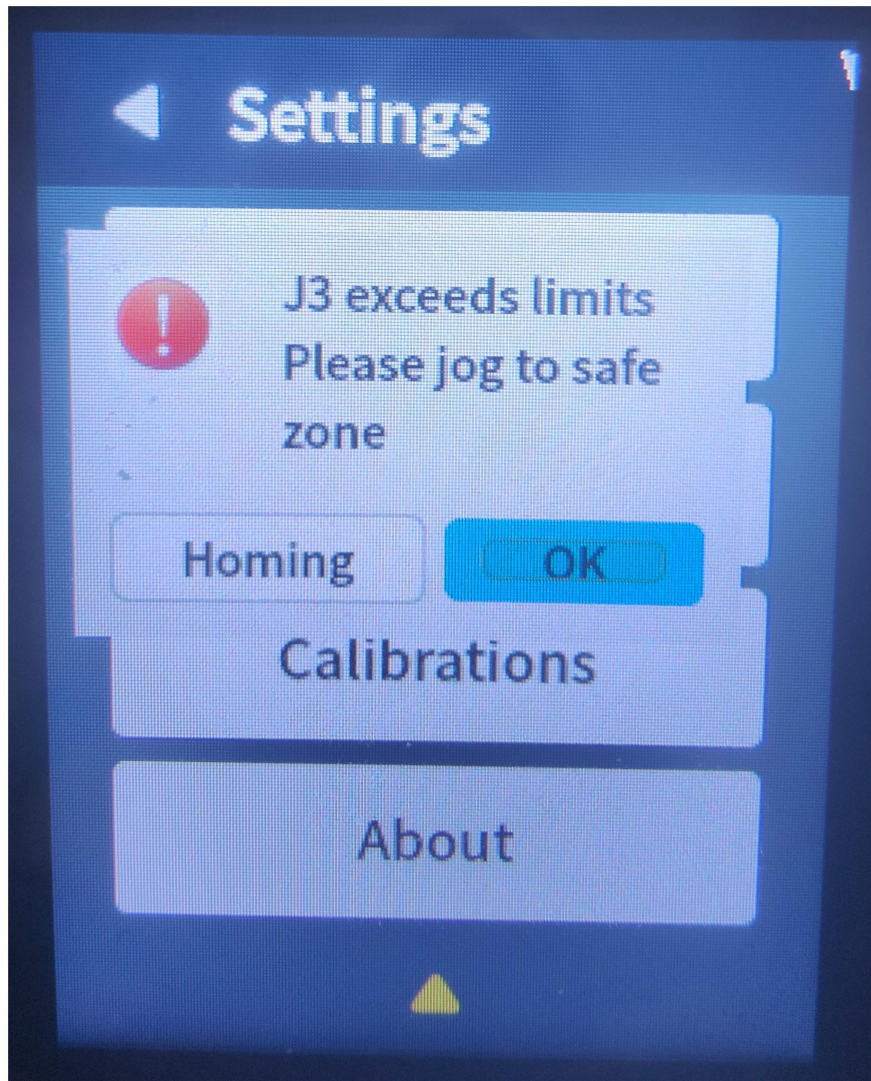
There will be a protection mechanism detection during the operation of Panel, and a pop-up prompt will be displayed after it is triggered.

For example:

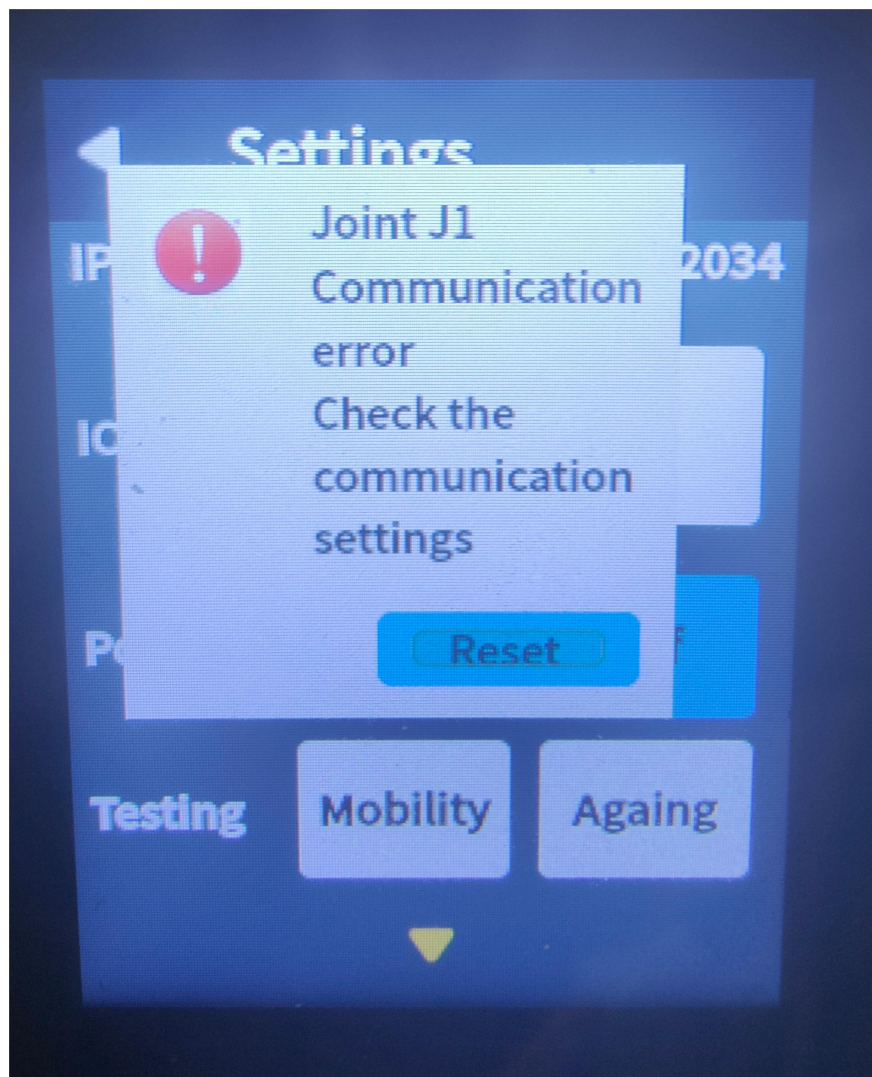
Joint out of limit, motor voltage is insufficient, voltage is too high, motor drive temperature is high, motor temperature is too high, motor current is too large, motor communication error, motor short circuit.

When the limit is exceeded, click the return to zero on the overlimit reminder pop-up window. At this time, the pop-up window prompts to wait for the robot arm to return to zero. After the movement stops, click the OK on the pop-up window to

continue to control the robot arm. If you click OK directly, you cannot control the robot arm.



When an error is reported outside the limit, the popup window has only one button to "recover", and then the robot arm cannot be controlled and needs to be powered down again.



pico firmware update instructions

Version	Firmware function description (function items)	Version iteration record	Firmware usage instructions	Applicable devices
V1.0	Robot control: 1. Single/multi-angle/coordinate control 2. Obtain angle/coordinates 3. Clamp, pump control			1. This version Requires use of 2020 devices

[← Previous Page](#) | [Next Page →](#)

How to burn firmware

1. Enter the `/home/er/A1_flash` folder
2. Run the burning script `A1_flash.sh` to burn the file

```
sudo bash /home/er/A1_flash/A1_flash.sh
```

If you want to get the latest version of firmware, please go to [Latest Firmware Release Address](#) to download.

Update firmware: Save the latest downloaded firmware to the `/home/er/A1_flash` folder, replace the `firmware.bin` file, and then execute the burning script `A1_flash.sh` .

[← Previous Page](#)

Environment configuration

pymycobot is a python library developed by Elephant Robot and is used for robot control.

Linux

The system has Python 3.8.10 installed by default at the factory, and the `pymycobot` control library has been installed, so users do not need to install it themselves.

pymycobot installation

You can install pymycobot by entering commands through the terminal

```
pip install pymycobot
```

pymycobot uninstall

You can uninstall pymycobot by entering commands through the terminal

```
pip uninstall pymycobot
```

pymycobot update

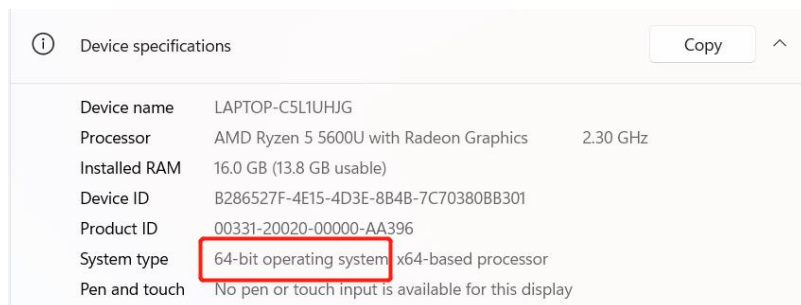
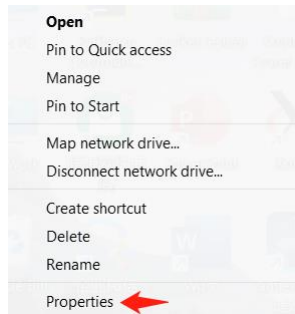
You can update pymycobot by entering commands through the terminal

```
pip install pymycobot -U
```

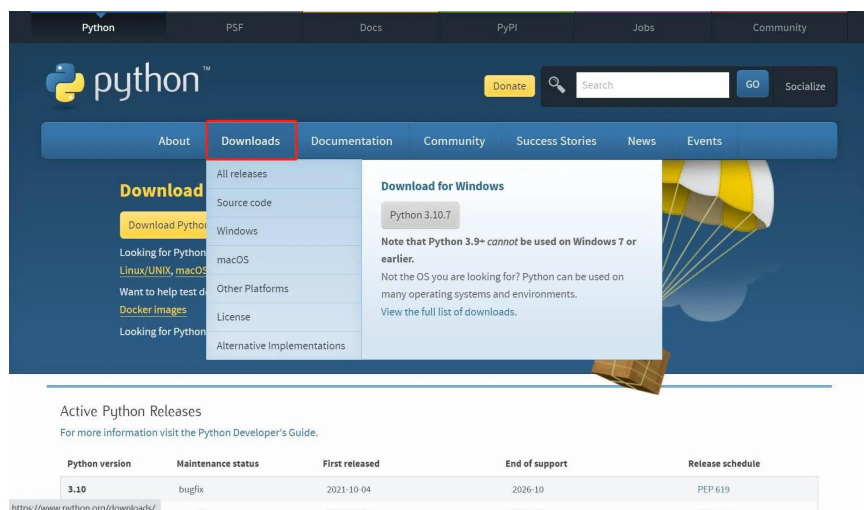
Windows

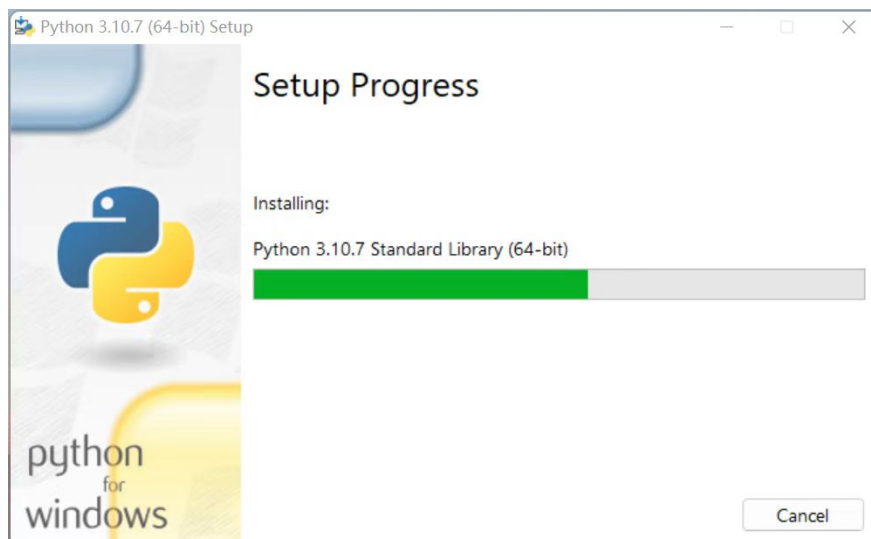
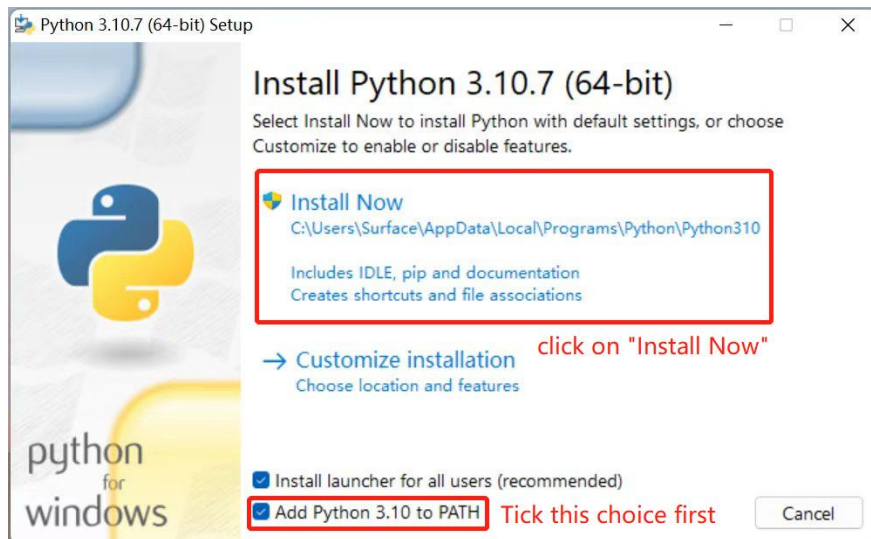
1.1 Installing Python

Notice: Before installation, check the operation system of PC. Press right button on the **My Computer** icon and then select **Properties**. Install the corresponding Python.

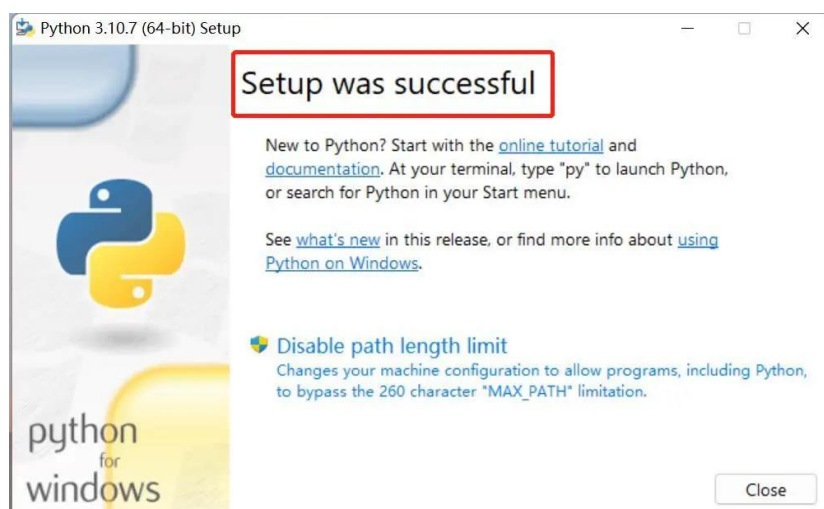


- Go to <http://www.python.org/download/> to download Python.
- Click on **Downloads**, and then download begins. Tick **Add Python 3.10 to PATH**. Click on **Install Now**, and then installation begins.





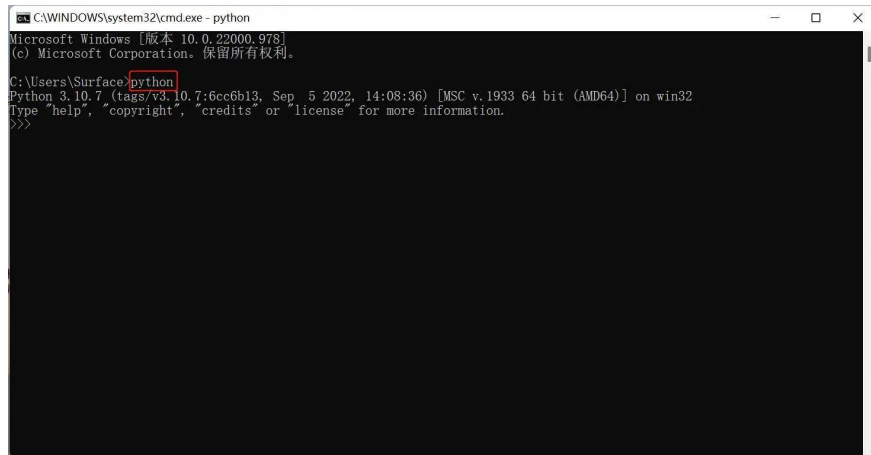
- Download and installation complete.



1.2 Running Python

Open the command prompt window (Win+R, input `cmd` and press `Enter`). Type `Python` .

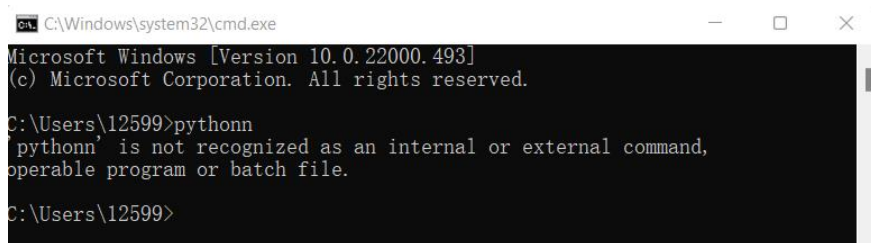
Successful Installation:



This on-screen instruction means that Python is successfully installed. The prompt `>>>` means Python interactive environment. If you input a Python code to get the execution result immediately.

Error Report:

If a wrong instruction is typed, for example "pythonn", the system may report an error.



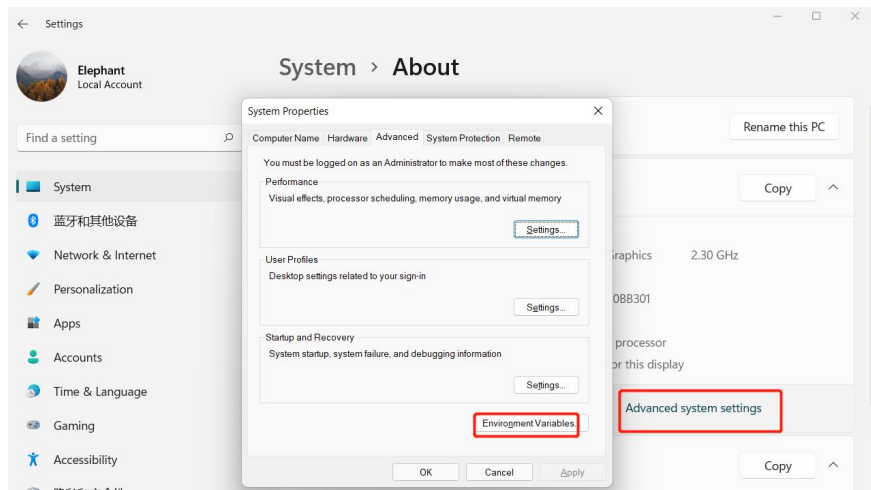
Notice: Generally, the error results from lack of environment configuration. Refer to **1.3 Environment Configuration** to solve problems.

1.3 Environment Variable Configuration

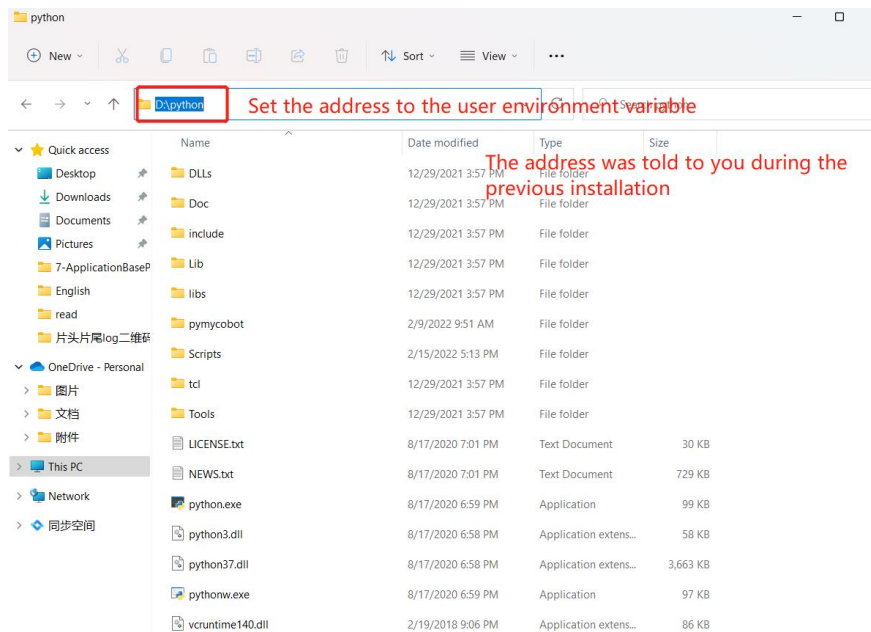
Windows follows the path set by a Path environment variable in search of **python.exe** . Otherwise, an error will be reported. If you fail to tick `Add Python 3.9 to PATH` during installation, you need to manually add the path where python.exe is located into environment variable or download python again. Remember to tick `Add Python 3.9 to PATH` .

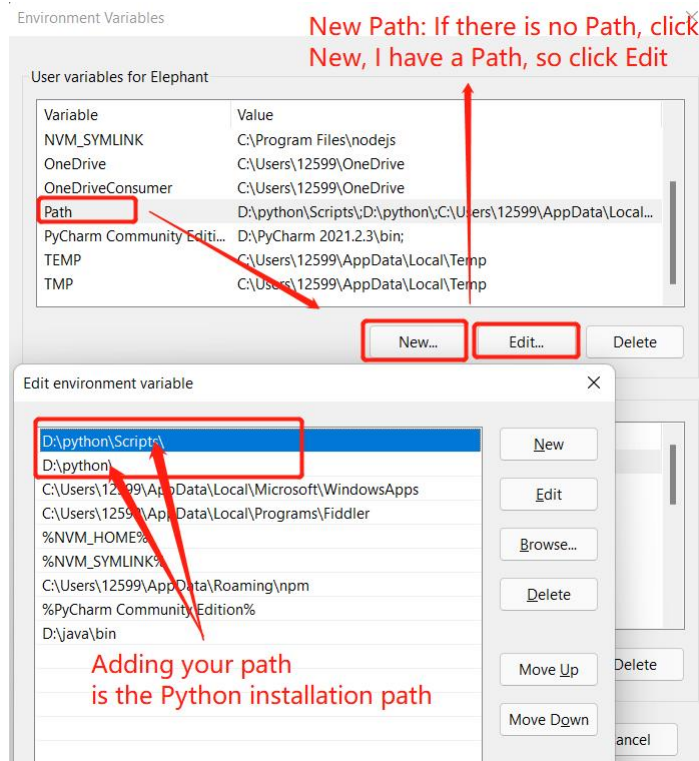
Follow the steps below to add python into environment variable manually.

- Right click on `My Computer` icon -->Properties ->Advanced System Settings ->Environment Variables



- The environment variables include user variables and system variables. For user variables, users can utilize their own downloaded programs via `cmd` command. Write the absolute path of the target program into the user variables.





- After the configuration, open the command prompt window (Win+R; input `cmd` and press `Enter`), and type `Python`.

```

C:\Windows\system32\cmd.exe - python
Microsoft Windows [Version 10.0.22000.493]
(c) Microsoft Corporation. All rights reserved.

C:\Users\12599>python
Python 3.7.9 (tags/v3.7.9:13c9474c7, Aug 17 2020, 18:58:18) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
  
```

2 Installation of PyCharm

PyCharm is a powerful python editor with the nature of cross-platform. Follow the steps below to download and install PyCharm.

Go to [PyCharm](#) to download PyCharm.

2.1 Download and Installation

Official website view:



Version: 2022.2.3
Build: 222.4345.23
11 October 2022

[System requirements](#)
[Installation instructions](#)
[Other versions](#)
[Third-party software](#)

Download PyCharm

[Windows](#) [macOS](#) [Linux](#)

Professional

For both Scientific and Web Python development. With HTML, JS, and SQL support.

[Download](#)

Free 30-day trial available

Community

For pure Python development

[Download](#)

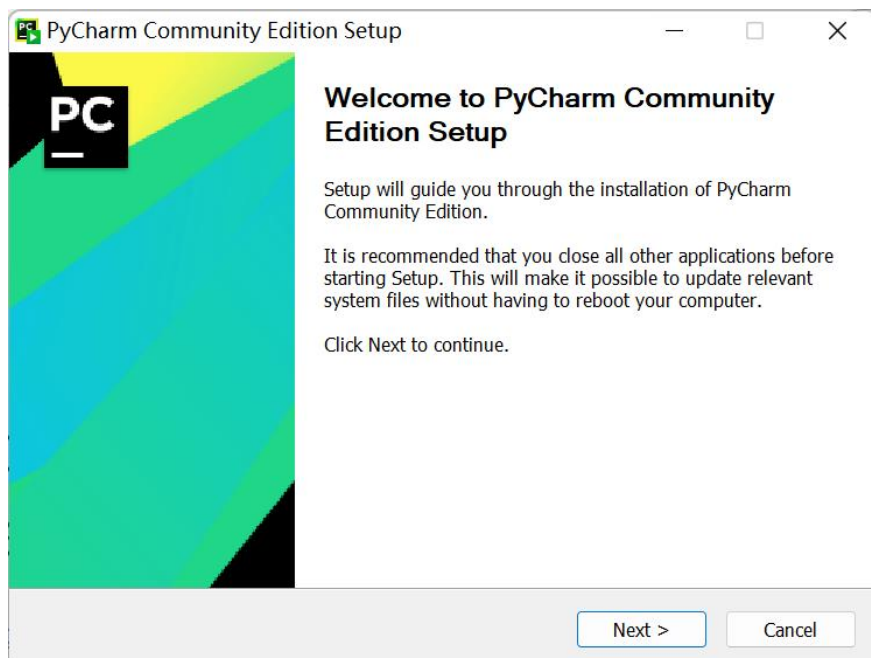
Free, built on open-source



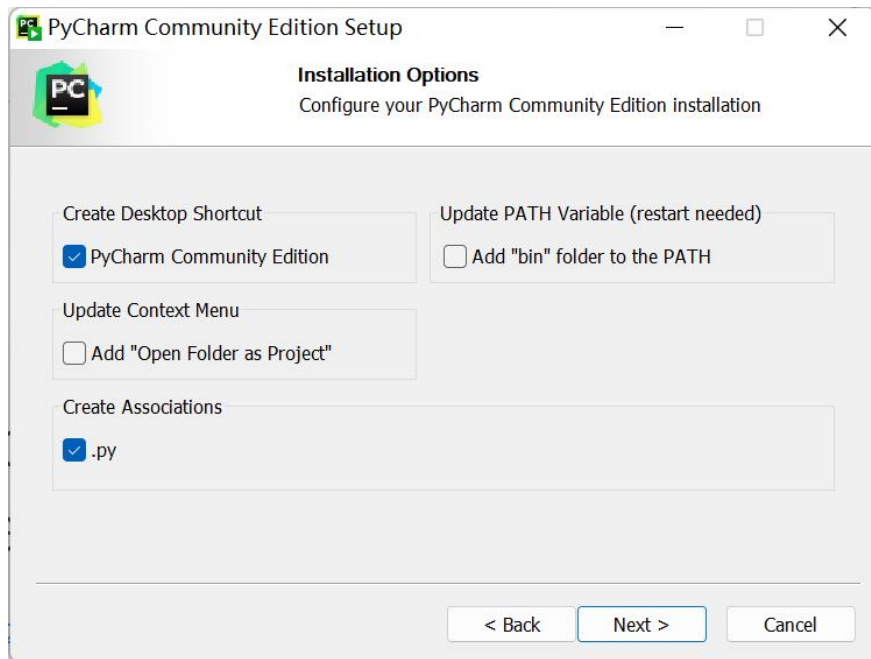
Get the Toolbox App to download PyCharm and its future updates with ease

It is recommended to install the free version.

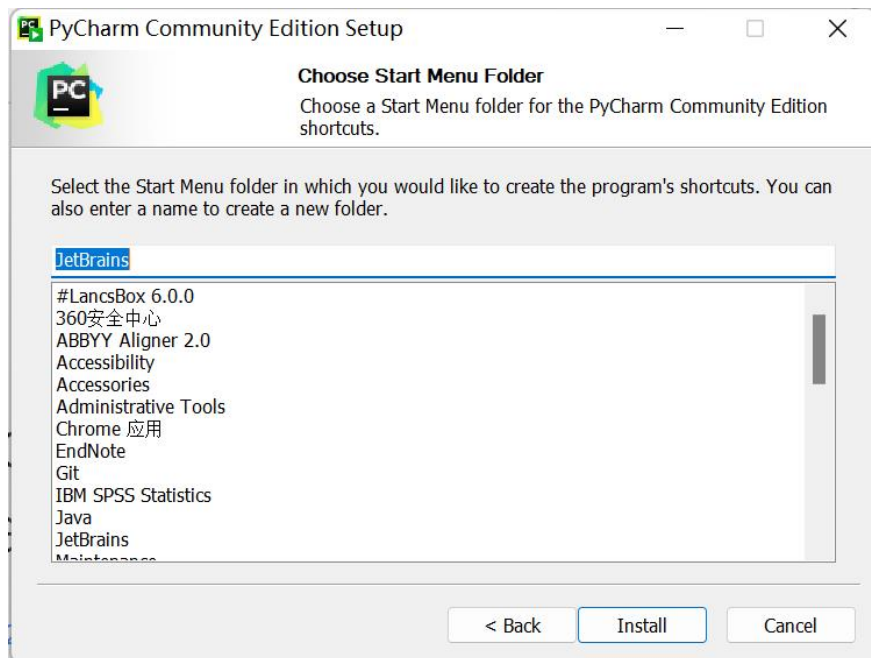
- Click on **Next** :



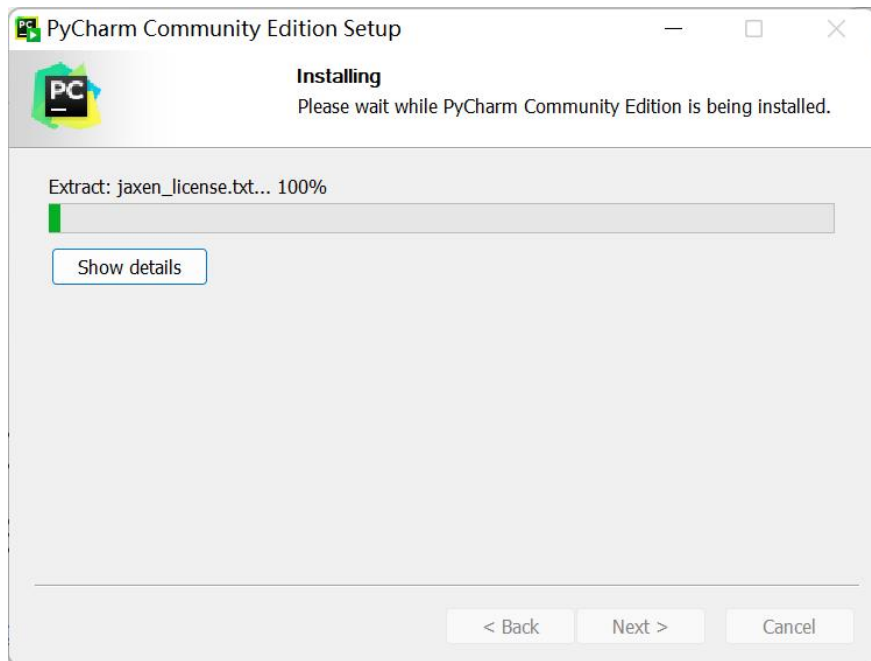
- Select options according to your needs and then select **Next** :



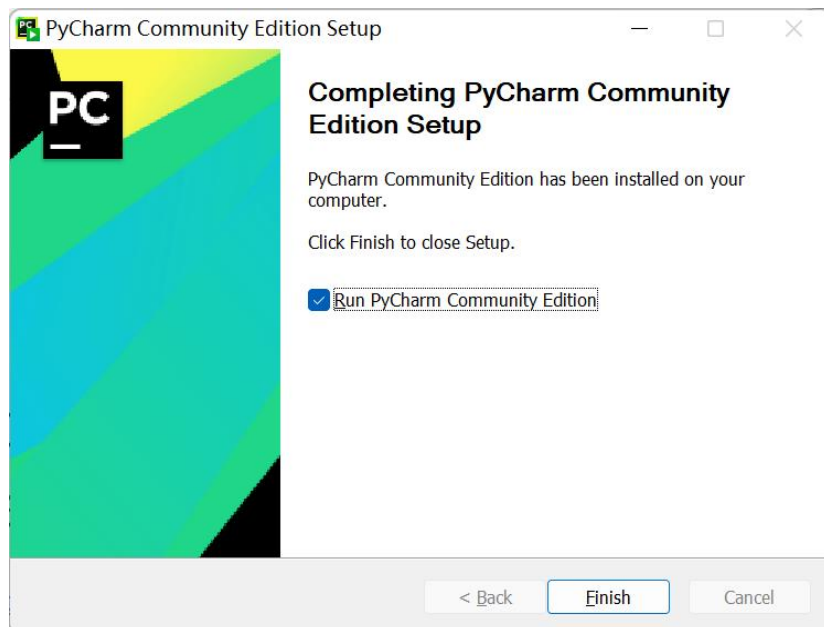
- Tap Install :



- Installing:



- Tap **Finish**

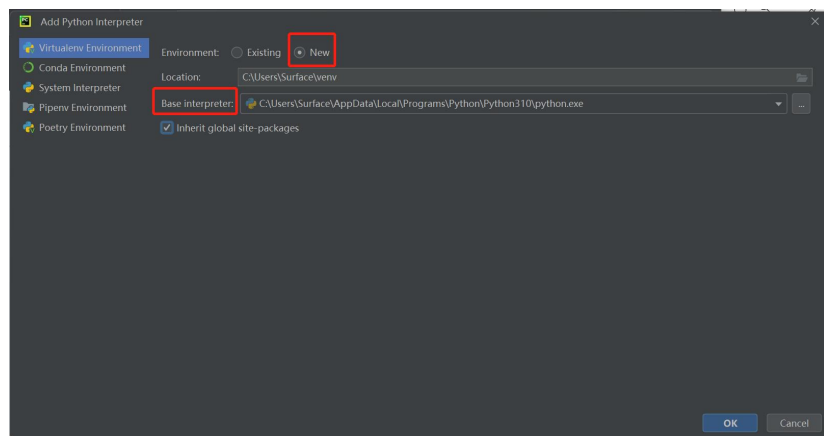
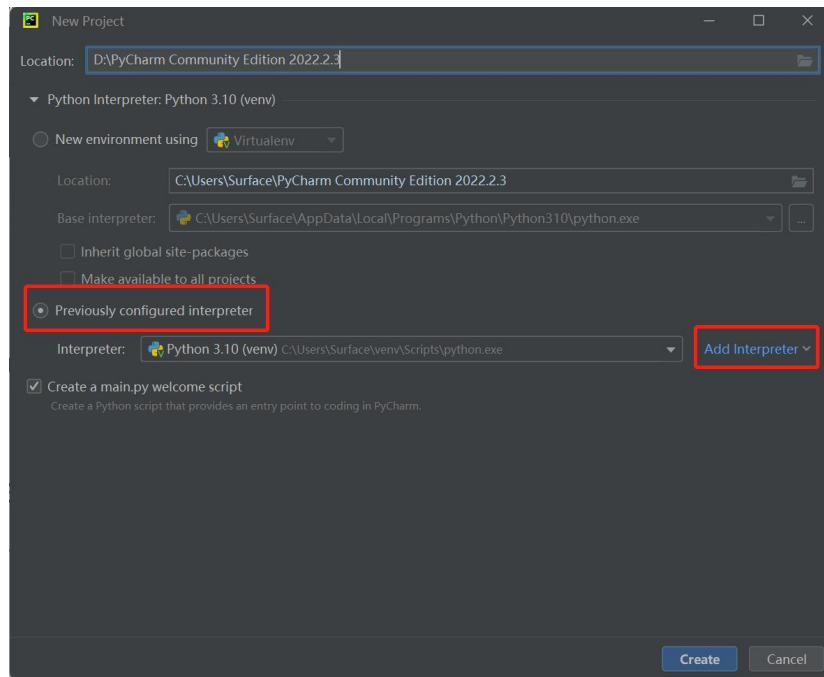


2.2 Create a new project

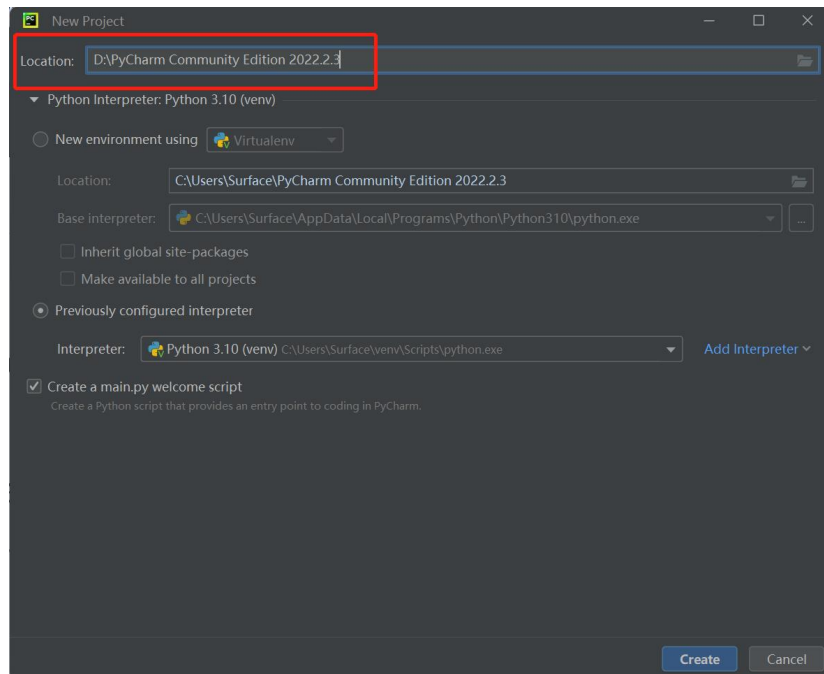
- Click **+New Project** :




- The **Interpreter** is used to interpret python programs. Select **Add Interpreter** -> **New** to add base interpreter.



- **Location** refers to the place where to save python file. Choose a file to put your programs.



- Click on `Create` and a sample appears: 
- Right click on the selection that the red arrow points, and create a new python file.



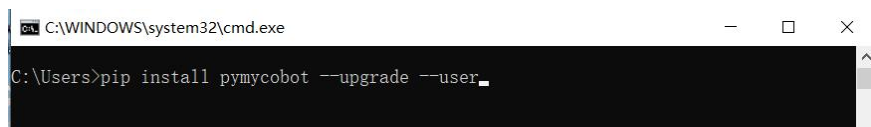
- Type name for the new file.



3 Preparations

- pymycobot installation. Type `pip install pymycobot --upgrade --user` via terminal (Win+R) `cmd` command.

```
pip install pymycobot --upgrade --user
```



- Source code installation. Open a terminal (Win+R, input `cmd`), and type the command below to install.

```
git clone https://github.com/elephantrobotics/pymycobot.git <your-path>

#Fill in your installation address in <your-path>, do not choose the current default

cd <your-path>/pymycobot

#Go to the pymycobot folder of the downloaded package.

#Run one of the following commands according to your python version.

# Install

python2 setup.py install

# or

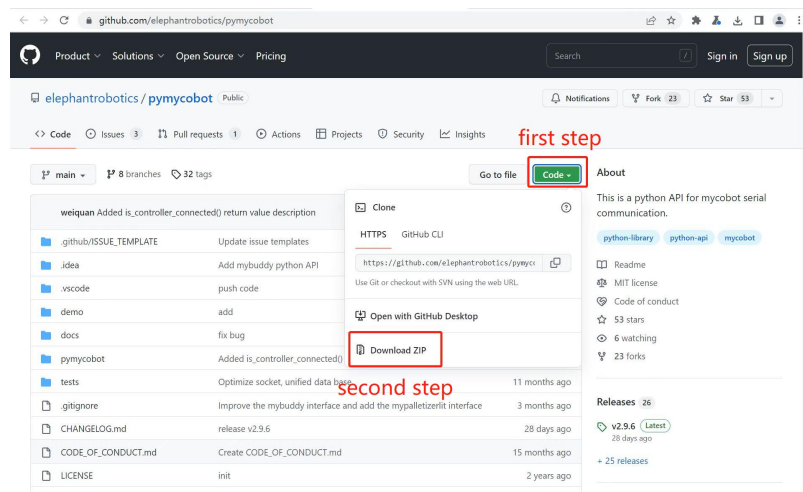
python3 setup.py install
```

- Update pymycobot

```
pip install pymycobot --upgrade
```

Notice:

1. If no red wavy line appears below the codes, pymycobot is successfully installed.
2. if a red wavy line appears, got to the address <https://github.com/elephantrobotics/pymycobot> to download pymycobot manually and put it into python library.



Basic usage of Python

```
from pymycobot import Mercury

mc = Mercury('/dev/ttyAMA1')

print(mc.get_angles())
```

[← System introduction](#) | [Python API](#) [→](#)

6.1 Python API

[toc]

6.1.2 API usage instructions

API (Application Programming Interface), also known as Application Programming Interface functions, are predefined functions. When using the following function interfaces, please import our API library at the beginning by entering the following code, otherwise it will not run successfully:

```
# Example
from pymycobot import Mercury

mc = Mercury('/dev/ttyAMA1')

print(mc.get_angles())
```

1. System Status

`get_system_version()`

- **function:** get system version
- **Return value:** system version

`get_robot_type()`

- **function:** get robot id
- **Return value:** Definition Rule: Actual machine model. For example, the Mercury A1 model is 4500

`get_atom_version()`

- **function:** Get the end version number
- **Return value:** End parameters(float)

`get_robot_status()`

- **function:** Upper computer error security status
- **Return value:** 0 - Normal. other - Robot triggered collision detection

2. Overall Status

`power_on()`

- **function:** atom open communication (default open)

- **Attentions:** After executing poweroff or pressing emergency stop, it takes 7 seconds to power on and restore power

- **Return value:**

- 1 - Power on completed.
- 0 - Power on failed

`power_off()`

- **function:** Power off of the robotic arm

- **Return value:**

- 1 - Power on completed.
- 0 - Power on failed

`is_power_on()`

- **function:** judge whether robot arms is powered on or not

- **Return value:**

- 1 : power on
- 0 : power off
- -1 : error

`release_all_servos()`

- **function:** release all robot arms

- **Attentions:** After the joint is disabled, it needs to be enabled to control within 1 second

- **Parameters:** `data` (optional) : The way to relax the joints. The default is damping mode, and if the 'data' parameter is provided, it can be specified as non damping mode (1- Undamping).

- **Return value:**

- 1 - release completed.
- 0 - release failed

`focus_all_servos()`

- **function:** Turn on robot torque output

- **Return value:**

- 1 : complete
- 0 : failed
- -1 : error

3.MDI Mode and Operation

`get_angles()`

- **function:** get the degree of all joints

- **Return value:** `list` a float list of all degree

`get_angle()`

- **function:** Get single joint angle
- **Parameters:** `joint_id` (int): 1 ~ 7
- **Return value:** Array of angles corresponding to joints

`send_angle(id, degree, speed)`

- **function:** send one degree of joint to robot arm
- **Parameters:**
 - `id` : Joint id(`genre.Angle`), range int 1-7
 - `degree` : degree value(`float`) | Joint Id | range | | ---- | ---- | | 1 | -178 ~ 178 | | 2 | -74 ~ 130 | | 3 | -178 ~ 178 | | 4 | -180 ~ 10 | | 5 | -178 ~ 178 | | 6 | -20 ~ 273 | | 7 | -180 ~ 180 |
 - `speed` : the speed and range of the robotic arm's movement 1~100

`send_angles(angles, speed)`

- **function:** Send all angles to all joints of the robotic arm
- **Parameters:**
 - `angles` : a list of degree value(`List[float]`), length 7
 - `speed` : (`int`) 1 ~ 100

`get_coords()`

- **function:** Obtain robot arm coordinates from a base based coordinate system
- **Return value:** a float list of coord:[x, y, z, rx, ry, rz]

`send_coord(id, coord, speed)`

- **function:** send one coord to robot arm
- **Parameters:**
 - `id` :send one coord to robot arm, 1-6 corresponds to [x, y, z, rx, ry, rz]
 - `coord` : coord value(`float`) | Coord Id | range | | ---- | ---- | | 1 | -466 ~ 466 | | 2 | -466 ~ 466 | | 3 | -240 ~ 531 | | 4 | -180 ~ 180 | | 5 | -180 ~ 180 | | 6 | -180 ~ 180 |
 - `speed` : (`int`) 1-100

`send_coords(coords, speed, mode)`

- **function::** Send overall coordinates and posture to move the head of the robotic arm from its original point to your specified point
- **Parameters:**
 - `coords` : a list of coords value [x,y,z,rx,ry,rz] ,length6
 - `speed` (`int`) : 1 ~ 100

pause()

- **function:** Control the instruction to pause the core and stop all movement instructions

is_paused()

- **function:** Check if the program has paused the move command
- **Return value:**
 - 1 - paused
 - 0 - not paused
 - -1 - error

resume()

- **function:** resume the robot movement and complete the previous command

stop()

- **function:** stop all movements of robot
- **Return value:**
 - 1 - stopped
 - 0 - not stop
 - -1 - error

is_in_position(data, flag)

- **function :** judge whether in the position.
- **Parameters:**
 - data: Provide a set of data that can be angles or coordinate values. If the input angle length range is 7, and if the input coordinate value length range is 6
 - flag data type (value range 0 or 1)
 - 0 : angle
 - 1 : coord
- **Return value:**
 - 1 - true
 - 0 - false
 - -1 - error

is_moving()

- **function:** judge whether the robot is moving
- **Return value:**
 - 1 moving
 - 0 not moving
 - -1 error

4. JOG Mode and Operation

`jog_angle(joint_id, direction, speed)`

- **function:** jog control angle
- **Parameters:**
 - `joint_id` : Represents the joints of the robotic arm, represented by joint IDs ranging from 1 to 7
 - `direction(int)` : To control the direction of movement of the robotic arm, input `0` as negative value movement and input `1` as positive value movement
 - `speed` : 1 ~ 100

`jog_coord(coord_id, direction, speed)`

- **function:** jog control coord.
- **Parameters:**
 - `coord_id : (int)` Coordinate range of the robotic arm: 1~6
 - `direction : (int)` To control the direction of machine arm movement, `0` - negative value movement, `1` - positive value movement
 - `speed` : 1 ~ 100

`jog_increment_angle(joint_id, increment, speed)`

- **function:** Single joint angle increment control
- **Parameters:**
 - `joint_id` : 1-7
 - `increment` : Incremental movement based on the current position angle
 - `speed` : 1 ~ 100

`jog_increment_coord(coord_id, increment, speed)`

- **function:** Single joint angle increment control
- **Parameters:**
 - `joint_id` : axis id 1 - 6.
 - `increment` : Incremental movement based on the current position coord
 - `speed` : 1 ~ 100

5. Coordinate controlled attitude deviation angle

`get_solution_angles()`

- **function:** Obtain the value of zero space deflection angle
- **Return value:** Zero space deflection angle value

`set_solution_angles(angle, speed)`

- **function:** Obtain the value of zero space deflection angle

- **Parameters:**

- `angle` : Input the angle range of joint 1, angle range -90 to 90
- `speed` : 1 - 100.

6. Joint software limit operation

`get_joint_min_angle(joint_id)`

- **function:** Read the minimum joint angle
- **Parameters:**
 - `joint_id` : Enter joint ID (range 1-7)
- **Return value:** `float` Angle value

`get_joint_max_angle(joint_id)`

- **function:** Read the maximum joint angle
- **Parameters:**
 - `joint_id` : Enter joint ID (range 1-7)
- **Return value:** `float` Angle value

`set_joint_min(id, angle)`

- **function:** Set minimum joint angle limit
- **Parameters:**
 - `id` : Enter joint ID (range 1-7)
 - `angle` : Refer to the limit information of the corresponding joint in the `send_angle()` interface, which must not be less than the minimum value

`set_joint_max(id, angle)`

- **function:** Set minimum joint angle limit
- **Parameters:**
 - `id` : Enter joint ID (range 1-7)
 - `angle` : Refer to the limit information of the corresponding joint in the `send_angle()` interface, which must not be greater than the maximum value

7. Joint motor control

`is_servo_enable(servo_id)`

- **function:** Detecting joint connection status
- **Parameters:** `servo id` 1-7
- **Return value:**
 - `1` : Connection successful
 - `0` : not connected
 - `-1` : error

is_all_servo_enable()

- **function:** Detect the status of all joint connections
- **Return value:**
 - 1 : Connection successful
 - 0 : not connected
 - -1 : error

set_servo_calibration(servo_id)

- **function:** The current position of the calibration joint actuator is the angle zero point
- **Parameters:**
 - servo_id : 1 - 7

release_servo(servo_id)

- **function:** Set the specified joint torque output to turn off
- **Parameters:**
 - servo_id : 1 ~ 7
- **Return value:**
 - 1 : release successful
 - 0 : release failed
 - -1 : error

focus_servo(servo_id)

- **function:** Set the specified joint torque output to turn on
- **Parameters:** servo_id : 1 ~ 7
- **Return value:**
 - 1 : focus successful
 - 0 : focus failed
 - -1 : error

set_break (joint_id, value)

- **function:** Set break point
- **Parameters:**
 - joint_id : int. joint id 1 - 7
 - value : int. 0 - disable, 1 - enable
- **Return value:** 0 : failure; 1 : success

get_servo_speeds()

- **function:** Get the movement speed of all joints
- **Return value:** unit step/s

`get_servo_currents()`

- **function:** Get the movement current of all joints
- **Return value:** 0 ~ 5000 mA

`get_servo_status()`

- **function:** Get the movement status of all joints
- **Return value:** a value of 0 means no error

`servo_restore(joint_id)`

- **function:** Clear joint abnormalities
- **Parameters:**
 - `joint_id` (int): joint id 1 - 7

8. Robotic arm end IO control

`set_digital_output(pin_no, pin_signal)`

- **function:** set IO statue
- **Parameters**
 - `pin_no` (int): Pin number
 - `pin_signal` (int): 0 / 1

`get_digital_input(pin_no)`

- **function:** read IO statue
- **Parameters:** `pin_no` (int)
- **Return value:** signal

9. Robotic arm end gripper control

`set_gripper_state(flag, speed, _type_1=None)`

- **function:** Adaptive gripper enable
- **Parameters:**
 - `flag` (int) : 0 - open 1 - close, 254 - release
 - `speed` (int) : 1 ~ 100
 - `_type_1` (int) :
 - 1 : Adaptive gripper (default state is 1)
 - 2 : A nimble hand with 5 fingers
 - 3 : Parallel gripper
 - 4 : Flexible gripper

set_gripper_value(gripper_value, speed, gripper_type=None)

- **function:** Set the gripper value
- **Parameters:**
 - `gripper_value (int)` : 0 ~ 100
 - `speed (int)` : 1 ~ 100
 - `gripper_type (int)` :
 - `1` : Adaptive gripper (default state is 1)
 - `2` : A nimble hand with 5 fingers
 - `3` : Parallel gripper
 - `4` : Flexible gripper

set_gripper_calibration()

- **function:** Set the current position of the gripper to zero

set_gripper_enabled(value)

- **function:** Adaptive gripper enable setting
- **Parameters:**
 - `value` 1: Enable 0: Release

set_gripper_mode(mode)

- **function:** Set gripper mode
- **Parameters:**
 - `value` :
 - 0: Transparent transmission mode
 - 1: normal mode

get_gripper_mode()

- **function:** Get gripper mode
- **Return value:**
 - 0: Transparent transmission mode
 - 1: normal mode

10. Button function at the end of the robot arm

is_btn_clicked()

- **function:** Get the status of the button at the end of the robot arm
- **Return value:**
 - 0: no clicked

- o 1: clicked

set_color(r, g, b)

- **function:** Set the color of the end light of the robotic arm
- **Parameters:**
 - o `r (int)` : 0 ~ 255
 - o `g (int)` : 0 ~ 255
 - o `b (int)` : 0 ~ 255

11. Drag Teaching

drag_tech_save()

- **function:** Start recording and dragging teaching points.
 - o Note: In order to display the best sports effect, the recording time should not exceed 90 seconds

drag_tech_pause()

- **function:** Pause sampling

drag_tech_execute()

- **function:** Start dragging the teach-in point, executing it only once.

12. Cartesian space coordinate parameter setting

set_tool_reference(coords)

- **function:** Set tool coordinate system.
- **Parameters:** `coords : (list) [x, y, z, rx, ry, rz]`.
- **Return value:** NULL

get_tool_reference(coords)

- **function:** Get tool coordinate system.
- **Return value:** `coords : (list) [x, y, z, rx, ry, rz]`

set_world_reference(coords)

- **function:** Set world coordinate system.
- **Parameters:** `coords : (list) [x, y, z, rx, ry, rz]`.
- **Return value:** NULL

`get_world_reference()`

- **function:** Get world coordinate system.
- **Return value:** `list` [x, y, z, rx, ry, rz].

`set_reference_frame(rftype)`

- **function:** Set base coordinate system.
- **Parameters:** `rftype` : 0 - base 1 - tool.

`get_reference_frame()`

- **function:** Set base coordinate system.
- **Return value:**
 - `0` - base
 - `1` - tool.

`set_movement_type(move_type)`

- **function:** Set movement type.
- **Parameters:**
 - `move_type` : 1 - moveI, 0 - moveJ.

`get_movement_type()`

- **function:** Get movement type.
- **Return value:**
 - `1` - moveI
 - `0` - moveJ

`set_end_type(end)`

- **function:** Get end coordinate system
- **Parameters:**
 - `end (int)` : `0` - flange, `1` - tool

`get_end_type()`

- **function:** Obtain the end coordinate system
- **Return value:**
 - `0` - flange
 - `1` - tool

13. Circular motion

`write_move_c(transpoint, endpoint, speed)`

- **function:** Arc trajectory motion

- Parameters: `transpoint(list)` : Arc passing through point coordinates
`endpoint (list)` : Arc endpoint coordinates `speed(int)` : 1 ~ 100

14. Set bottom IO input/output status

`set_basic_output(pin_no, pin_signal)`

- function:** Set Base IO Output
- Parameters:**
 - `pin_no (int)` Pin port number, range 1 ~ 6
 - `pin_signal (int)`: 0 - low. 1 - high

`get_basic_input(pin_no)`

- function:** Read base IO input, range 1 ~ 6
- Parameters:**
 - `pin_no (int)` pin number
- Return value:** 0 - low. 1 - high

15. Set up 485 communication at the end of the robotic arm

`tool_serial_restore()`

- function:** 485 factory reset

`tool_serial_ready()`

- function:** Set up 485 communication
- Return value:** 0 : not set 1 : Setup completed

`tool_serial_available()`

- function:** Set up 485 communication
- Return value:** 0-Normal 1-Robot triggered collision detection

`tool_serial_read_data()`

- function:** Read fixed length data. Before reading, read the buffer length first.
After reading, the data will be cleared
- Parameters:** `data_len (int)`: The number of bytes to be read, range 1 ~ 45
- Return value:** 0 : not set 1 : Setup completed

`tool_serial_write_data()`

- function:** End 485 sends data, Data length range is 1 ~ 45 bytes
- Return value:** 0-Normal 1-Robot triggered collision detection

`tool_serial_flush()`

- **function:** Clear 485 buffer
- **Return value:** 0-Normal 1-Robot triggered collision detection

`tool_serial_peek()`

- **function:** View the first data in the buffer, the data will not be cleared
- **Return value:** 1 byte data

`tool_serial_set_baud(baud)`

- **function:** Set 485 baud rate, default 115200
- **Parameters:** baud (int): baud rate
- **Return value:** NULL

`tool_serial_set_timeout(max_time)`

- **function:** Set 485 timeout in milliseconds, default 30ms
- **Parameters:** max_time (int): timeout
- **Return value:** NULL

remote control

Mercury A1 supports socket remote control. We provide a developed python socket server script, which can realize remote control of the robot using python.

Server

The server script is the Server_A1.py file saved on the system desktop. Run this file to open the socket server.

```
python Server_A1.py
```

Client

The client is required to be in the same network segment as the server. [Self Hotspot] (../5-BasicApplication/5.1-SystemUsageInstructions/5.1-SystemUsageInstructions.md#513-vnc) will be turned on by default when the system is powered on, client computer You can connect to the machine's own hotspot or connect to the same WIFI as the machine.

Client usage

pymycobot contains the API of the socket client. The client is created:

```
from pymycobot import MercurySocket

mc = MercurySocket("10.42.0.1", 9000)

print(mc.get_angles())
```

The usage of socket API is basically the same as that of serial communication API, and the interface is universal. [Learn more about API](#)

[← Python API](#) | [System →](#)

ROS

ROS is an open-source meta-operating system used for robots. It provides an operating system with expected services, including hardware abstraction, low-level device control, implementation of common functions, messages transferred between processes, and package management. It also provides the tools and library functions needed to obtain, compile, write, and run codes across computers.

The "graph" of ROS runtime is a loosely coupled peer-to-peer process network based on a ROS communication infrastructure. ROS implements several different communication methods, including a services mechanism based on synchronous RPC-style communication, a topics mechanism based on asynchronous streaming media data, and a parameter server for data storage.

ROS is not a real-time framework, but it can be embedded in real-time programs. Willow Garage's PR2 robot uses a system called `pr2_etherCAT` to send or receive ROS messages in real time. ROS may also be seamlessly integrated with Orocos real-time toolkits.

ROS Logo :



1 Design goals and characteristics of ROS

Many people ask "what are differences between ROS and other robot software platforms?" This question is difficult to answer. Because ROS is not a framework that integrates most functions or features. In fact, the main goal of ROS is to provide code reuse support for robot R&D. ROS is a framework (i.e. nodes) for distributed processes, which are encapsulated in program and function packages that can be easily shared and distributed. ROS also supports a federated system similar to a code repository, and this system also enables the collaboration and distribution of a project. This design enables the development and realization of a project to be decided completely independently from the file system to the user interface (no limit by ROS). At the same time, all projects can be integrated with the basic tools of ROS.

To support the primary goals of sharing and collaboration, the ROS framework has several other features:

- **Lean:** ROS is designed to be as lean as possible so that codes written for ROS can be used with other robot software frameworks. The inevitable conclusion from this is that ROS can be easily integrated into other robot software platforms: ROS can already be integrated with OpenRAVE, Orocos and Player.

- ROS insensitive libraries: The preferred development model of ROS is written with clean library functions that do not depend on ROS.
- Language independence: The ROS framework can simply be implemented in any modern programming language. ros has implemented Python version, C++ version and Lisp version. It also has experimental libraries for Java and Lua versions.
- Loose coupling: The function modules in ROS are encapsulated in independent function packages or meta-function packages, which are easy to share. The modules in the function package are run in units of nodes. With ROS standard IO used as the interface, developers does not need to pay attention to the internal implementation of the module, as long as they understand the interface rules, they can achieve reuse and point-to-point loose coupling between modules.
- Convenient testing: ROS has a built-in unit/integration testing framework called rostest, which can easily install or uninstall test modules.
- Scalable: ROS is applicable to large runtime systems and large development processes.
- Free and open-source: It has many developers and many function packages.

2 Why ROS is used?

Through ROS, we can realize the simulated control of robot arms in a virtual environment.

we will visualize robot arms through **rviz**, operates our robot arms in a variety of ways, and plan and executes the action path of robot arms through **Movelt** to achieve the effect of freely controlling the robot arms.

We will learn how to control our products through the platform in ros in the following chapters.

Movelt

Movelt is currently the most advanced software for movement operations of robot arms and has been used for more than 100 robots. It integrates the latest achievements in motion planning, control, 3D perception, motion control, control and navigation, provides an easy-to-use platform for developing advanced robot applications, and provides an integrated software platform for design, and integration assessment of new robot products in the fields of industry, commerce, R&D, etc.

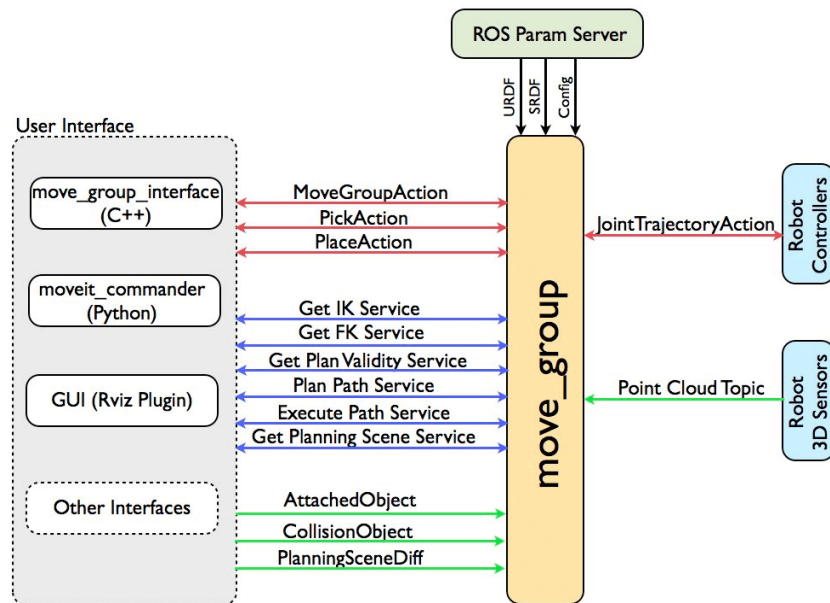
Movelt Logo :



1 Introduction

MoveIt is an integrated development platform in ROS, which consists of a variety of functional packages for manipulating robot arms, including motion planning, operation, control, inverse kinematics, 3D perception, collision detection, etc.

The following figure shows the high-level structure of the main node **move_group** provided by MoveIt. It is like a combiner: all the individual components are integrated together, providing a series of actions and services for users to use.



2 User interface

The user may access the operations and services provided by **move_group** in three ways:

- In C++, you may use **move_group** easily by using **move_group_interface** package.
- In Python, use the **moveit_commander** package.
- Via GUI: use Rviz (ROS visualization tool) of Motion-commander.

move_group can be configured using the ROS parameter server, from which the robot's URDF and SRDF can also be obtained.

3 Configuration

move_group is a ROS node. It uses the ROS parameter server to obtain three kinds of information:

- URDF - **move_group** looks for the **robot_description** parameter in the ROS parameter server to get the robot's URDF.

- SRDF - `move_group` looks for the `robot_description_semantic` parameter in the ROS parameter server to get the robot's SRDF. SRDF is typically created by the user using an MoveIt Setup Assistant.
 - MoveIt configuration - `move_group` will look in the ROS parameter server for additional MoveIt-specific configurations, including joint constraint, kinematics, motion planning, perception, and other information. The configuration files for these components are automatically generated by the MoveIt Setup Assistant and stored in the configuration directory of the robot's corresponding MoveIt configuration package. For the use of configuration assistant, please refer to: [MoveIt Setup Assistant](#)
-

[← Previous Section](#) | [Next Page →](#)

Raspberry Pi system environment

The Raspberry Pi version comes with an Ubuntu (V-20.04) system and a built-in development environment, so you don't need to build and manage it, Just update the mercury_ros package.

mercury_ros is a ROS1 package launched by Elephant Robot for its Mercury series robotic arms.

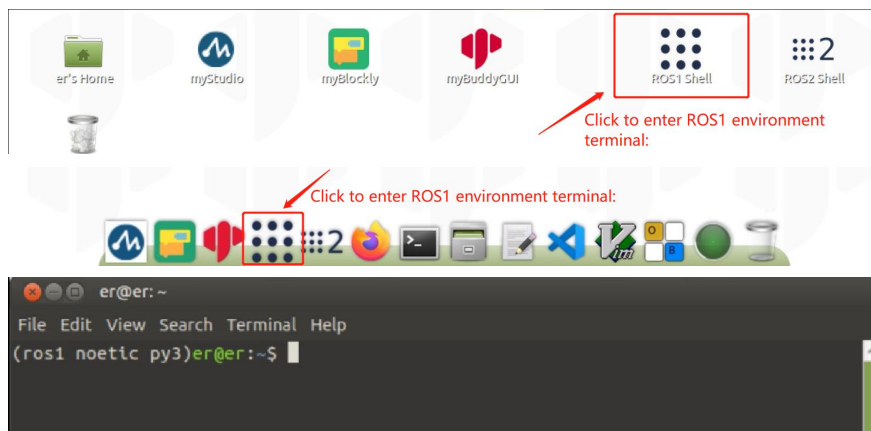
ROS1 Project address: http://github.com/elephantrobotics/mercury_ros

Robotic arm API driver library address:

<https://github.com/elephantrobotics/pymycobot>

1 Update the mercury_ros package

In order to ensure that users can use the latest official package in time (new users do not need to update), they can go to the /home/er/catkin_ws/src folder through the file manager and click the ROS1 Shell icon on the desktop or the corresponding icon in the lower bar of the desktop to open the ROS1 environment terminal:



Then run the command to update:

```
# Clone the code on github
cd ~/catkin_ws/src
git clone https://github.com/elephantrobotics/mercury_ros.git # Please check the attar
cd ~/catkin_ws      # Back to work area
catkin_make # Build the code in the workspace
source devel/setup.bash # add environment variable
```

Note: If the mercury_ros folder already exists in the /home/er/catkin_ws/src (equivalent to ~/catkin_ws/src) directory, you need to delete the original mercury_ros before executing the above command. Among them, ubuntu in the directory path is the user name of the virtual machine. If it is inconsistent, please modify it.

So far, the ROS1 environment construction has been completed. You can learn [the basics of ROS](#) or [ROS use cases](#).

[← Previous Page](#) | [Next Page →](#)

1 ROS project structure

1.1 catkin workspace

Catkin workspace is the directory where catkin software packages are created, modified, and compiled. Catkin's workspace can be intuitively described as a warehouse, which contains various ROS project projects to facilitate system organization, management and calling.

- **Create workspace:**

```
mkdir -p ~/catkin_ws/src # Create a folder
cd ~/catkin_ws/src      # Enter the folder
catkin_init_workspace    # Initialize the current directory into a ROS workspace
cd ..                   # Return to the parent directory
catkin_make              # Build the code in the workspace
```

The structure of catkin is very clear. It includes three paths: src, build, and devel. It may also include others under some compilation options. But these three folders are the default for the catkin compilation system. Their specific functions are as follows:

```
src/: ROS catkin software package (source code package)

build/: cache information and intermediate files of catkin (CMake)

devel/: Generated target files (including header files, dynamic link libraries, static
```

A simple workspace looks like this:

```
workspace_folder/      -- WORKSPACE
  src/                  -- SOURCE SPACE
    CMakeLists.txt      -- 'Toplevel' CMake file, provided by catkin
  package_1/
    CMakeLists.txt      -- CMakeLists.txt file for package_1
    package.xml         -- Package manifest for package_1
    ...
  package_n/
    CMakeLists.txt      -- CMakeLists.txt file for package_n
    package.xml         -- Package manifest for package_n
```


1.2 ROS software package

Package is not only a software package on Linux, but also the basic unit of catkin compilation. The object we use catkin_make to compile is each ROS package.

```
+--PACKAGE
+-- CMakeLists.txt
+-- package.xml
+-- src/
+-- include/
+-- scripts/
+-- msg/
+-- srv/
+-- urdf/
+-- launch/
```

- **CMakeLists.txt**: Defines the package name, dependencies, source files, target files and other compilation rules of the package. It is an essential component of the package.
- **package.xml**: Describes the package name, version number, author, dependencies and other information of the package, which is an indispensable component of the package.
- **src/**: stores ROS source code, including C++ source code (.cpp) and Python module (.py)
- **include/**: stores the header files corresponding to the C++ source code
- **scripts/**: stores executable scripts, such as shell scripts (.sh), Python scripts (.py)
- **msg/**: stores messages in custom format (.msg)
- **srv/**: stores services in custom formats (.srv)
- **urdf/**: stores the robot's model description (.urdf or .xacro) and 3D model files (.sda, .stl, .dae, etc.)
- **launch/**: stores launch files (.launch or .xml)

Create your own package:

- Command format:

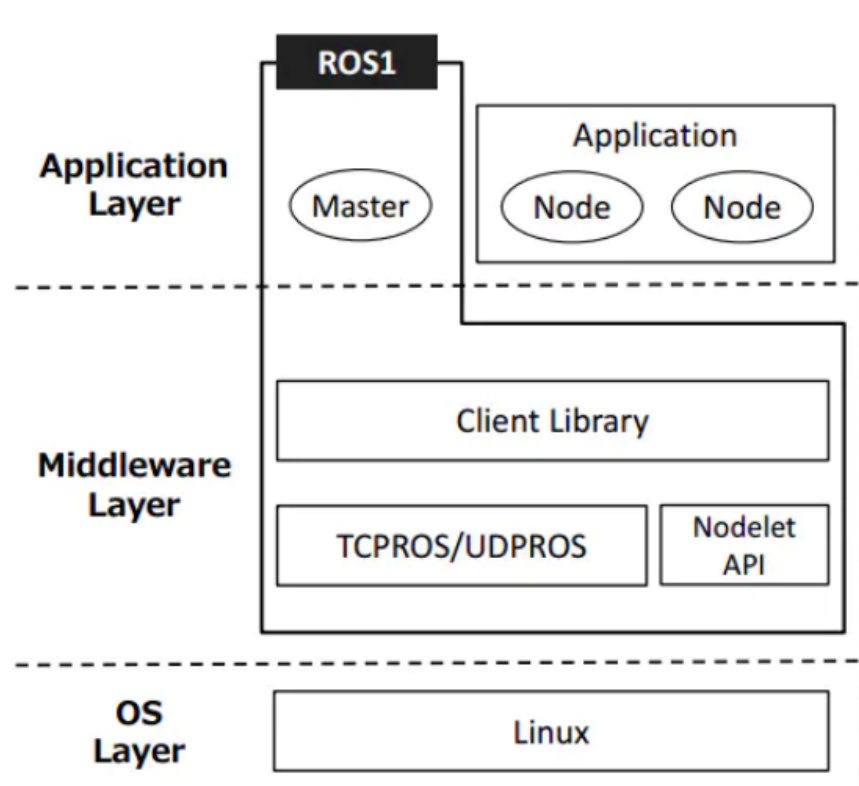
The catkin_create_pkg command will ask you to enter package_name. If necessary, you can also add some other dependent software packages later:

```
catkin_create_pkg <package_name> [depend1] [depend2] [depend3]
```

- For example:

```
catkin_create_pkg beginner_tutorials std_msgs rospy roscpp
```

2 ROS communication architecture



2.1 Master and node

1 Master

Node manager. Each node must register with the master before starting and manage the communication between nodes.

2 roscore

Starting the master will also start roscout (log management) and parameter server (parameter manager)

3 nodes

ROS processes and instances of running executable files in pkg.

```
$roslaunch [pkg_name] [node_name] #Start
$roslaunch list #List currently running node information
$roslaunch info [node_name] #Display detailed information of a node
$roslaunch kill [node_name] #End a node
```

4 launch

Start the master and multiple nodes.

```
$roslaunch [pkg_name] [file_name.launch]
```

2.2 Service and Topic

We provide some services and topics for interacting with mycobot.

1 Service

Enter in the command line:

```
source ~/catkin_ws/devel/setup.bash # Add environment variables
roslaunch mycobot_320_communication communication_service.launch
```

Support parameters:

- port: concatenate serial string
- baud: baud rate

Open a new command line:

```
# Display active service information
rosservice list

# /get_joint_angles
# /get_joint_coords
# /set_joint_angles
# /set_joint_coords
# /switch_gripper_status
# /switch_pump_status
```

Related commands and instructions:

command	Detailed description
rosservice list	Display active service information
rosservice info [service name]	Display information about the specified service
rosservice type [service name]	Show service type
rosservice find [service name]	Find a service for a specified service type
rosservice uri [service name]	show ROSRPC URI service
rosservice args [service name]	show service parameters
rosservice call [service name] [parameters]	Request service with input parameters

2 Topic

Enter in the command line:

```
source ~/catkin_ws/devel/setup.bash
roslaunch mycobot_320_communication communication_topic.launch
```

Support parameters:

- port: concatenate serial string
- baud: baud rate

Open a new command line:

```
# Display active service information
rostopic list

#/mycobot/angles_goal
#/mycobot/coords_goal
#/mycobot/angles_real
#/mycobot/coords_real
#/mycobot/pump_status
#/mycobot/gripper_status
```

Related commands and instructions:

Command	Detailed description
rostopic list	Display active topic list
rostopic echo [topic name]	Display the message content of the specified topic in real time
rostopic find [type name]	Display threads with messages of the specified type
rostopic type [topic name]	Displays the message type of the specified topic
rostopic bw [topic name]	Display the message bandwidth of the specified topic (bandwidth)
rostopic hz [topic name]	Display the message data publishing cycle of the specified topic
rostopic info [topic name]	Display information about the specified topic
rostopic pub [topic name] [message type] [parameters]	Post a message with the specified topic name

The difference between service and topic:

	service	topic
Synchronization	Asynchronous	Synchronous
communication model	pub/sub	server/client
underlying protocol	ROSTCP/ROSUDP	ROSTCP/ROSUDP
Feedback Mechanism	No	Yes
buffer	Yes	No
Real-time	Weak	Strong
Node Relationship	Many-to-Many	One-to-Many
Applicable Scenarios	Data Transmission	Logical Processing

you can go to [service](#) and [topic](#) learn more about the use of these two features

2.3 Introduction to msg and srv

- msg: The msg file is a simple text file describing the fields of a ROS message. They are used to generate source code for messages in different languages (c++ or python, etc.).

- **srv**: srv files are used to describe services. It consists of two parts: the request (request) and the response (response). msg files are stored in the msg directory of the package, and srv files are stored in the srv directory.

1 rosmmsg

rosmmsg is a command line tool for displaying information about ROS message types.

rosmmsg demo:

```
rosmmsg show      # Show message description
rosmmsg info      # Display message information
rosmmsg list      # list all messages
rosmmsg md5       # Display md5 encrypted message
rosmmsg package   # Display all messages under a feature pack
rosmmsg packages  # List feature packs that contain messages
```

- rosmmsg list will list all msgs in the current ROS
- rosmmsg packages List all packages containing messages
- rosmmsg package List all msgs under a package

```
//rosmmsg package # Package names
rosmmsg package turtlesim
```

- rosmmsg show Show message description

```
//rosmmsg show # message name
rosmmsg show turtlesim/Pose
# result:
float32 x
float32 y
float32 theta
float32 linear_velocity
float32 angular_velocity
```

- rosmmsg info Works the same as rosmmsg show
- rosmmsg md5 A check algorithm to ensure the consistency of data transmission

2 rossrv

rossrv is a command-line tool for displaying information about ROS service types, and uses a syntax that is highly similar to rosmmsg.

```

rossrv show      # Display service message details
rossrv info      # Display information about service messages
rossrv list      # List all service information
rossrv md5       # Display md5 encrypted service messages
rossrv package   # Display all service messages under a package
rossrv packages  # Show all packages that contain service messages

```

- `rossrv list` Will list all srv messages in the current ROS
- `rossrv packages` List all packages that contain service messages
- `rossrv package` List all msgs under a package

```

//rossrv package # Package names
rossrv package turtlesim

```

- `rossrv show` Show message description

```

//rossrv show # message name
rossrv show turtlesim/Spawn
# result:
float32 x
float32 y
float32 theta
string name
---
string name

```

- `rossrv info` The effect is the same as `rossrv show`
- `rossrv md5` Use md5 checksum (encryption) for service data

3 Introduction to URDF

- Unified Robot Description Format, Unified Robot Description Format, abbreviated as URDF. The `urdf` package in ROS contains a C++ parser for URDF, and URDF files describe robot models in XML format. *URDF cannot be used alone, it needs to be combined with Rviz or Gazebo. URDF is just a file that needs to be rendered into a graphical robot model in Rviz or Gazebo.

3.1 urdf file description

Code example:

Only part of the code is intercepted here for display:

```

<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro" name="firefighter" >

<xacro:property name="width" value=".2" />

<link name="base">
  <visual>
    <geometry>
      <mesh filename="package://mercury_description/urdf/mercury_a1/base.dae"/>
    </geometry>
    <origin xyz = "0.0 0 0" rpy = " 0 0 0"/>
  </visual>
  <collision>
    <geometry>
      <mesh filename="package://mercury_description/urdf/mercury_a1/base.dae"/>
    </geometry>
    <origin xyz = "0.0 0 0" rpy = " 0 0 0"/>
  </collision>
</link>

<link name="joint1">
  <visual>
    <geometry>
      <mesh filename="package://mercury_description/urdf/mercury_a1/joint1.dae"/>
    </geometry>
    <origin xyz = "0.0 0 -0.08" rpy = " 0 0 0"/>
  </visual>
  <collision>
    <geometry>
      <mesh filename="package://mercury_description/urdf/mercury_a1/joint1.dae"/>
    </geometry>
    <origin xyz = "0.0 0 -0.08" rpy = " 0 0 0"/>
  </collision>
</link>

<link name="joint2">
  <visual>
    <geometry>
      <mesh filename="package://mercury_description/urdf/mercury_a1/joint2.dae"/>
    </geometry>
    <origin xyz = "0.0 0 0.0" rpy = " 3.14159 0 -1.5708" />
  </visual>
  <collision>
    <geometry>
      <mesh filename="package://mercury_description/urdf/mercury_a1/joint2.dae"/>
    </geometry>

```



```

    <origin xyz = "0.0 0 0.0 " rpy = " 3.14159 0 -1.5708"/>
  </collision>
</link>

<joint name="joint1_to_base" type="revolute">
  <axis xyz="0 0 1"/>
  <limit effort = "1000.0" lower = "-3.1066" upper = "3.1066" velocity = "0"/>
  <parent link="base"/>
  <child link="joint1"/>
  <origin xyz= "0 0 0.175" rpy = "0 0 0"/>
</joint>

<joint name="joint2_to_joint1" type="revolute">
  <axis xyz="0 0 1"/>
  <limit effort = "1000.0" lower = "-1.4311" upper = "2.2689" velocity = "0"/>
  <parent link="joint1"/>
  <child link="joint2"/>
  <!-- <origin xyz= "0 0 0" rpy = "1.5708 3.14159 0"/> -->
  <origin xyz= "0 0 0" rpy = "-1.5708 0 0"/>
</joint>

</robot>

```

It can be seen that the urdf file is not complicated, it is mainly composed of two parts, `link` and `joint`, which are repeated continuously.

3.2 link section

The link element describes a rigid body with inertial, visual features, and collision properties

3.2.1 Attributes

name: The name used to describe the link itself

3.2.2 element

- `<inertial>` (optional)
 - Inertia properties of connecting rods
 - `<origin>` (optional, defaults to identity if not specified)
 - Defines the reference coordinate of the inertial reference system relative to the connecting rod coordinate system. The coordinate must be defined at the center of gravity of the connecting rod, and its coordinate axis may not be parallel to the main axis of inertia.
 - xyz (optional, defaults to zero vector) Represents the offset in the x , y , z x,y,zx,y,z directions, in meters.

- rpy(optional: defaults to identity if not specified) Indicates the rotation of the coordinate axis in the RPY direction, in radians.
 - `<mass>` Mass properties of connecting rods
 - `<inertia>` 3×3 rotational inertia matrix, consisting of six independent quantities: `ixx`, `ixy`, `ixz`, `iyy`, `iyz`, `izz`.
- `<visual>` (optional)
 - Visual properties of the connecting rod. It is used to specify the shape of the link display (rectangle, cylinder, etc.). There can be multiple visual elements in the same link, and the shape of the link is formed by two elements. In general, the model is more complex and can be drawn through solidworks to generate stl calls, and simple shapes such as adding end effectors can be directly written. At the same time, the position of the geometry can be adjusted according to the gap between the theoretical model and the actual model.
 - `<name1>` (optional) The name of the connecting rod geometry.
 - `<origin>` (optional, defaults to identity if not specified)
 - The geometry coordinate system relative to the coordinate system of the connecting rod.
 - xyz (optional: defaults to zero vector) Represents the offset in the x , y , z x,y,zx,y,z directions, in meters.
 - rpy (optional: defaults to identity if not specified) Indicates the rotation of the coordinate axis in the RPY direction, in radians.
- `<geometry>` (required)
 - The shape of the visualization, which can be one of the following:
 - `<box>` A rectangle with elements including length, width, and height. The origin is in the center.
 - `<cylinder>` Cylinder, elements include radius and length. center of origin.
 - `<sphere>` Sphere, element containing the radius. The origin is in the center.
 - `<mesh>` The grid, as determined by the file, also provides a scale to define its boundaries. Collada .dae files are recommended, .stl files are also supported, but must be a local file.
- `<material>` (optional)
 - Visualize the component's material. It can be defined outside the link tag, but it must be inside the robot tag. When defining outside the link tag, the name of the link must be quoted.
 - `<color>` (optional) Color, consisting of red/green/blue/alpha, in the range [0,1].
 - `<texture>` (optional) Material properties, defined by the file.
- `<collision>` (optional)
 - Collision properties of the link. Collision properties differ from visual properties of connecting rods, and simple collision models are often used to simplify calculations. The same link can have multiple collision attribute labels, and the collision attribute representation of the link is composed of the set of geometric shapes defined by it.
 - `<name>` (optional) Specifies the name of the connecting rod geometry
 - `<origin>` (optional, defaults to identity if not specified)

- The reference coordinate system of the collision component is relative to the reference coordinate system of the link coordinate system.
- xyz (optional, default zero vector) Represents the offset in the x , y , z x,y,zx,y,z directions, in meters.
- rpy (optional, defaults to identity if not specified) Indicates the rotation of the coordinate axis in the RPY direction, in radians.
- `<geometry>` Same as the geometry element description above

Detailed elements and the role of each element can go to [official documentation](#) to view

3.3 joint part

The joint section describes the kinematics and dynamics of the joint and specifies safety limits for the joint.

3.3.1 properties of joint:

name:

Specifies a unique name for the joint

type:

Specifies the type of joint, where type can be one of the following:

- revolute - A hinged joint that rotates along an axis, the range of which is specified by the upper and lower bounds.
- Continuous - A continuous hinged joint that rotates around an axis with no upper and lower bounds.
- Prismatic - A sliding joint that slides along an axis, the range of which is specified by upper and lower limits. +Fixed - this is not really a joint because it cannot move. All degrees of freedom are locked. This type of joint does not require axes, calibration, dynamics, limits or safety_controller.
- Floating - This joint allows motion in all 6 degrees of freedom.
- Plane - This joint allows movement in a plane perpendicular to the axis.

3.3.2 elements of joint

- `<origin>` (optional, defaults to identity if not specified) In the transformation from parent link to child link, the joint is located at the origin of the child link. Modifying this parameter can adjust the position of the connecting rod. It can be used to adjust the error between the actual model and the theoretical model, but it is not recommended to modify it greatly, because this parameter affects the connecting rod stl The position of , easily affects the collision detection effect.
 - xyz (optional: default to zero vector) Represents the offset in the x , y , z x,y,zx,y,z axis directions, in meters.

- rpy (optional: default to zero vector) Represents the angle of rotation around a fixed axis: roll is around the x-axis, pitch is around the y-axis, and yaw is around the z-axis, expressed in radians.
- `<parent>` (required)
 - The name of the parent link is a mandatory attribute.
 - link The name of the parent link is the name of the link in the robot structure tree.
- `<child>` (required)
 - The name of the child link is a mandatory attribute.
 - link The name of the child link is the name of the link in the robot structure tree.
- `<axis>` (optional: defaults to (1,0,0))
 - The joint's axis is in the joint's coordinate system. This is the axis of rotation (revolute joint), the axis of movement of the prismatic joint, and the standard plane of the planar joint. This axis is specified in the joint coordinate system. Modifying this parameter can adjust the axis around which the joint rotates. It is often used to adjust the rotation direction. If the model rotation is opposite to the actual one, just multiply by -1. Fixed and floating joints do not need this element.
 - xyz(required) x , y , z x, y, z components representing axis vectors, as normalized vectors.
- `<calibration>` (optional)
 - The reference point of the joint, used to correct the absolute position of the joint.
 - rising (optional) When the joint is moving forward, the reference point triggers a rising edge.
 - falling (optional) When the joint is moving forward, the reference point triggers a falling edge.
- `<dynamics>` (optional)
 - This element is used to specify the physical properties of the joint. Its value is used to describe the modeling performance of the joint, especially during simulation.

`<limit>` (Required when the joint is a rotation or translation joint)

- This element is a joint kinematics constraint.
- lower (optional, default to 0) Specify the attribute of the lower bound of the joint's motion range (the unit of the revolute joint is radians, and the unit of the prismatic joint is meters). This attribute is ignored for continuous joints.
- upper (optional, defaults to 0) Specify the attribute of the upper bound of the joint's motion range (the unit of the revolute joint is radians, and the unit of the prismatic joint is the meter). This attribute is ignored for continuous joints.
- effort (required) This property specifies the maximum force at which the joint will run.
- velocity (required) This property specifies the maximum speed of the joint runtime.

`<mimic>` (optional)

- This tag is used to specify a defined joint to mimic an existing joint. The value of this joint can be calculated using the following formula: $\text{value} = \text{multiplier} * \text{other_joint_value} + \text{offset}$
- `joint(required)` The name of the joint to mimic.
- `multiplier(optional)` Specify the multiplier factor in the above formula.
- `offset(optional)` Specify the offset term in the above formula. Default value is 0

`<safety_controller>` (optional)

- This element is a security control limit. The data under this element will be read into `move_group`, but it is invalid in practice. `Move_group` will skip this limit and directly read the parameter content under limit. At the same time, setting this element may cause planning failure.
- `soft_lower_limit (optional, defaults to 0)` This attribute specifies the lower bound of the joint security control boundary, which is the starting limit point of the joint security control. This value needs to be greater than the lower value in the above limit.
- `soft_upper_limit (optional, defaults to 0)` This attribute specifies the upper bound of the joint security control boundary, which is the starting limit point of the joint security control. This value needs to be less than the upper value in the above limit.
- `k_position(optional, defaults to 0)` This attribute is used to describe the relationship between position and velocity.
- `k_velocity(required)` This property is used to describe the relationship between force and velocity.

Detailed elements and the role of each element can go to

<http://wiki.ros.org/urdf/XML/joint> to view.

4 Commonly used command tools

In ROS, there are many commonly used command line tools, which can help you develop, debug, manage ROS nodes, etc. The following are some commonly used ROS command line tools:

4.1 Compile workspace

```
cattin_make
```

4.2 roscore

Start the ROS master node. Before running a ROS node, you usually need to start roscore first

```
roscore
```

4.3 rosrun

Run the specified ROS node.

```
roslaunch package_name node_name
```

4.4 roslaunch

Use the Launch file to start one or more ROS nodes.

```
roslaunch package_name launch_file.launch
```

4.5 rosnodetool

View running ROS node information.

```
roslaunch list
roslaunch info node_name
```

4.6 rostopic

View information about running ROS topics.

```
rostopic list
rostopic echo topic_name
```

4.7 rosservice

View and call ROS services.

```
rosservice list
rosservice call service_name
```

4.8 rosparam

Get and set ROS parameters.

```
rosparam get parameter_name  
rosparam set parameter_name value
```

4.9 rosmmsg

View ROS message types.

```
rosmmsg show message_type
```

4.10 rosdep

Install dependencies of ROS packages.

```
rosdep install package_name
```

4.11 Environment variables

View the ROS_PACKAGE_PATH environment variable

```
echo $ROS_PACKAGE_PATH
```

[← Previous Page](#) | [Next Page →](#)

Brief introduction and use of rviz

rviz is a 3D visualization platform in ROS. On one hand, it can realize the graphical display of external information, and on the other hand, it can also release control information to an object through rviz, realizing the monitoring and control of a robot.

1 Installation of rviz and the introduction to its interface

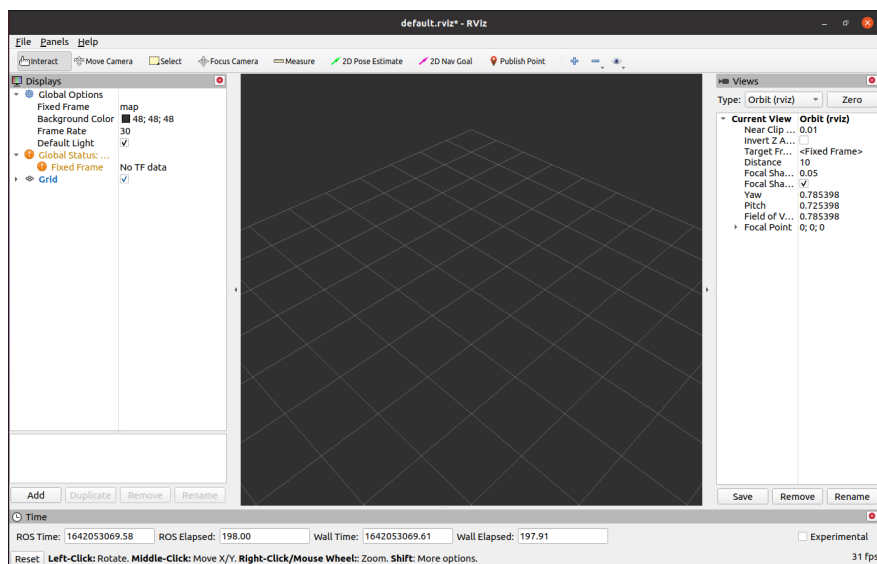
Open a new terminal (shortcut key: `Ctrl+Alt+T`) and enter the following command:

```
roscore
```

Then open a new terminal (shortcut key: `Ctrl+Alt+T`) and input the following command to open rviz.

```
roslaunch rviz rviz  
  
# or  
  
rviz
```

Open rviz, and the following interface will be displayed:



1.1 Introduction of all areas

- There is a list of monitors on the left. The monitor is a device that draws something in a 3D world and may have some options available in the display list.

- On the top is a toolbar, which allows the user to use various function buttons to select tools with multiple functions.
- The middle part is the 3D view: It is a main screen where various data can be viewed in three dimensions. The background color, fixed frame, grid, etc. of the 3D view can be set in detail in the Global Options and Grid items displayed on the left.
- Below is the time display area, including system time and ROS time.
- The right side is the observation angle setting area where different observation angles can be set.

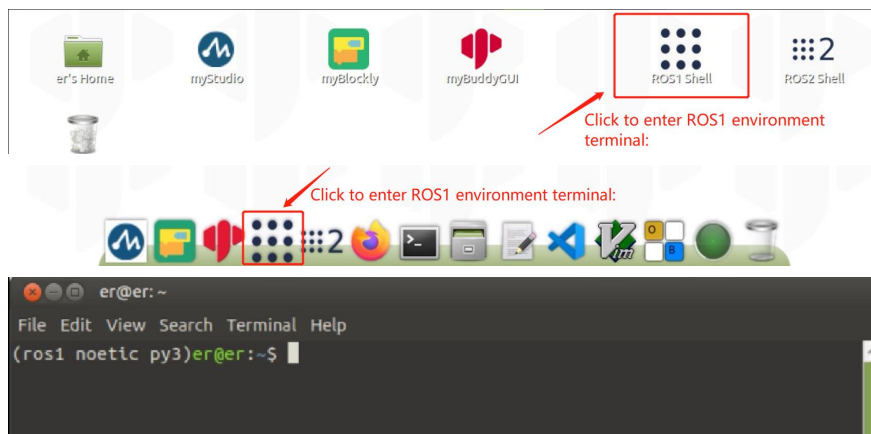
We only give a rough introduction in this part. If you want to know more details, go to [User Guide](#).

2 Simple use

Start using launch file

This example is built on what you have already done [Environment building](#) and you have successfully copied the company's code from GitHub to your virtual machine.

Click the `ROS1 Shell` icon on the desktop or the corresponding icon in the lower bar of the desktop to open the ROS1 environment terminal:



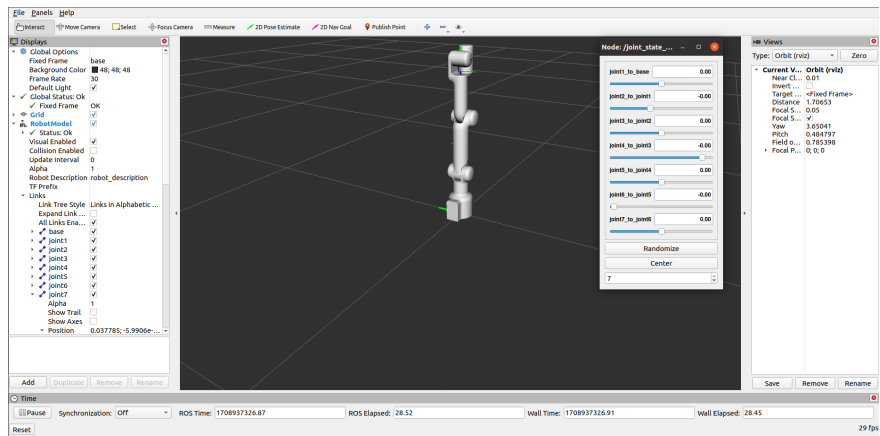
Input the command to **configure the ROS environment**.

```
cd ~/catkin_ws/
source devel/setup.bash
```

再输入:

```
roslaunch mercury_a1 test.launch
```

Open rviz, and then you will obtain the following result:



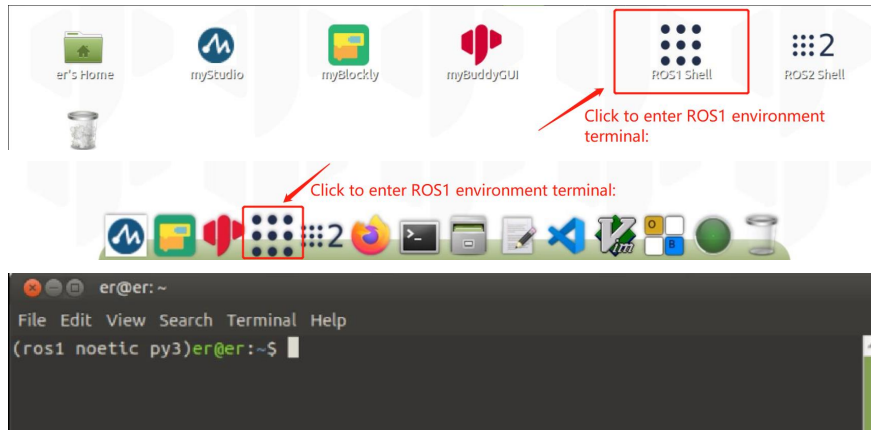
If you want to know more information about rviz, go to [Official documents](#).

[← Previous Page](#) | [Next Page →](#)

Control of the robotic arm

1 Slider Control

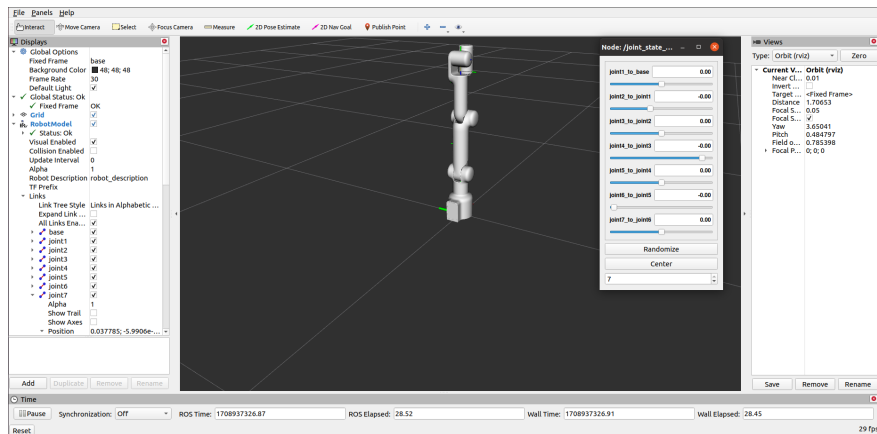
Click the `ROS1 Shell` icon on the desktop or the corresponding icon in the lower bar of the desktop to open the ROS1 environment terminal:



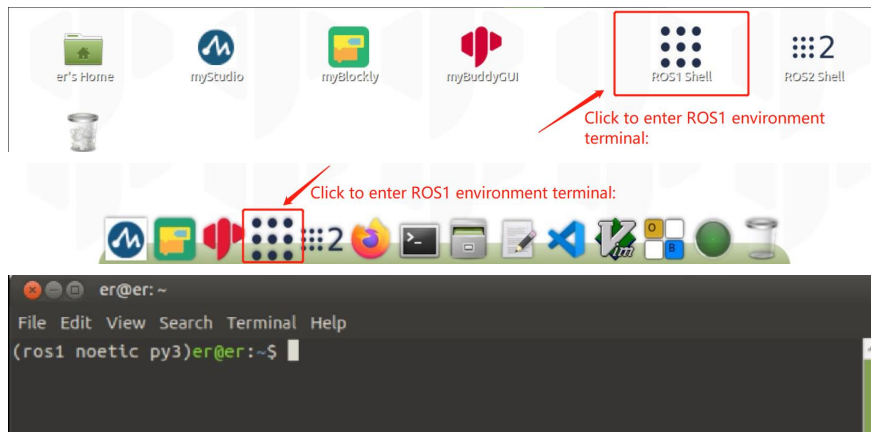
Then run the command:

```
roslaunch mercury_a1 slider_control.launch
```

rviz and a slider component will be opened, and you will see the following interface:



Then you can **control the model in rviz to make it move by dragging the slider**. If you want the real mycobot to move with the model, you need to open another ROS1 environment terminal:



Then run the command:

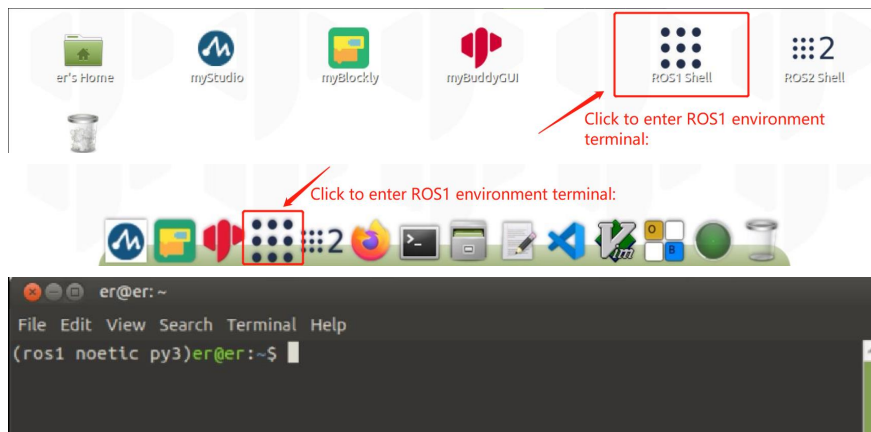
```
roslaunch mercury_a1 slider_control.py
```

Note: Since the robot arm will move to the current position of the model when the command is input, make sure that the model in rviz does not appear to be worn out before you use the command.

Do not drag the slider quickly after connecting the robot arm to prevent damage to the robot arm.

2 Model Following

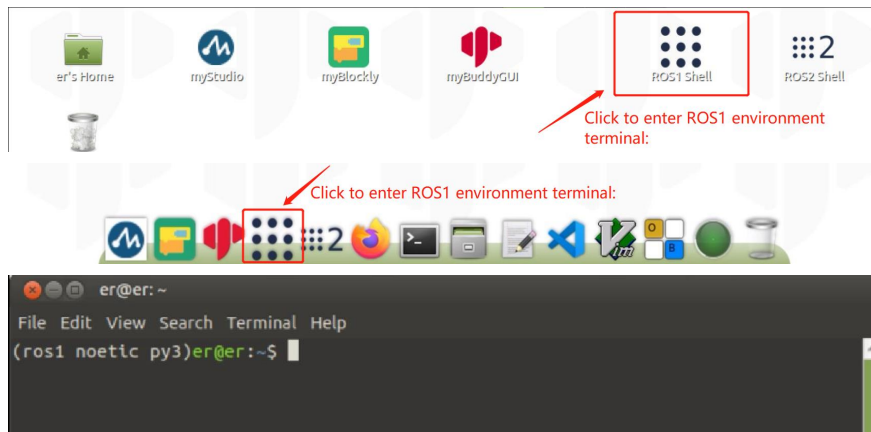
In addition to the above controls, we can also let the model move by following the real robot arm. Open a ROS1 environment terminal:



Then run the command:

```
roslaunch mercury_a1 follow_display.py
```

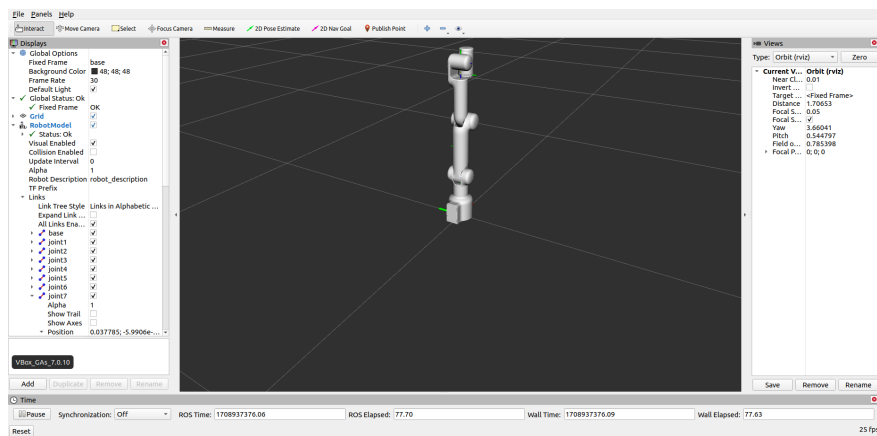
Then open another ROS1 environment terminal:



Then run the command:

```
roslaunch mercury_a1 mercury_follow.launch
```

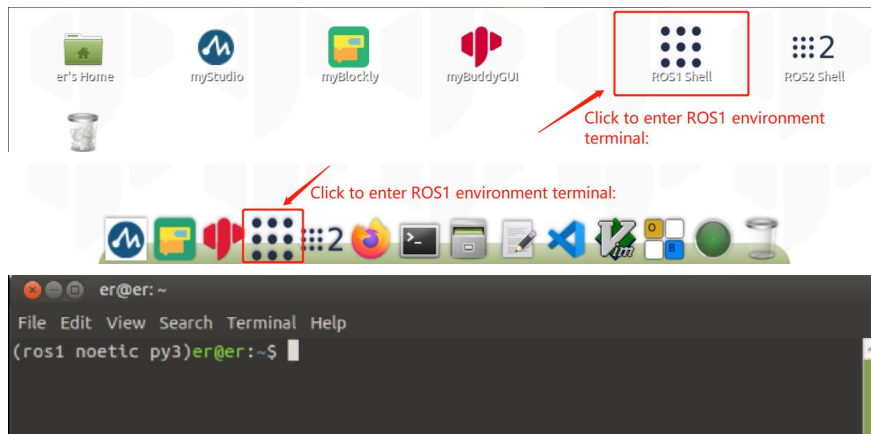
It will open rviz to show the model following effect.



3 GUI control

On the basis of the previous contents, this package also **provides a simple GUI control interface**. This method is used for interaction between real robot arms. Connect to mycobot.

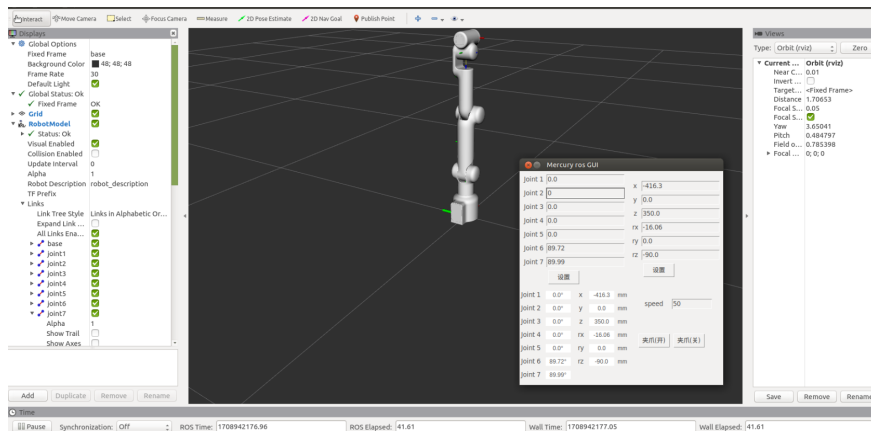
Open a ROS1 environment terminal:



Then run the command:

```
roslaunch mercury_a1 simple_gui.launch
```

Running effect:

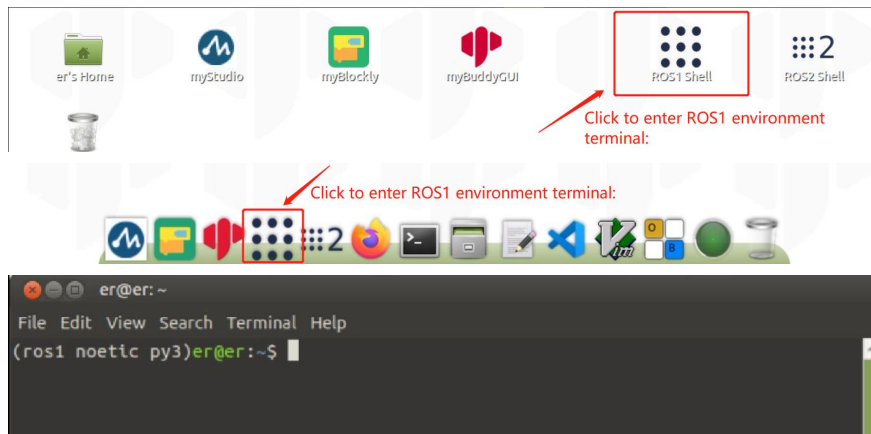


Note: Before using the gripper switch button, make sure the adaptive gripper is connected to the end of the robot arm.

4 Keyboard control

Keyboard control is added in `mercury_a1` package, and real-time Synchronization is performed in rviz. This function depends on pythonApi, so be sure to connect with the real robot arm.

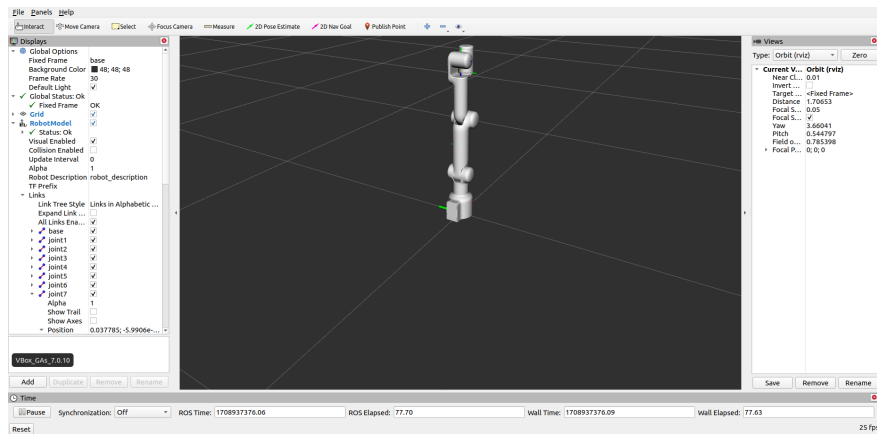
Open a ROS1 environment terminal:



Then run the command:

```
roslaunch mercury_a1 teleop_keyboard.launch
```

Running effect is as follows:



Mercury A1 information will be output in the command line as follows:

SUMMARY

=====

PARAMETERS

```
* /mercury_services/ baud: 115200
* /mercury_services/port: /dev/ttyAMA1
* /robot_description: <?xml version="1....
* /roscdistro: noetic
* /rosversion: 1.15.14
```

NODES

```
/
  mercury_services (mercury_a1_communication/mercury_services.py)
  real_listener (mercury_a1/listen_real.py)
  robot_state_publisher (robot_state_publisher/robot_state_publisher)
  rviz (rviz/rviz)
```

ROS_MASTER_URI=http://localhost:11311

```
process[robot_state_publisher-1]: started with pid [14764]
process[rviz-2]: started with pid [14765]
process[mecharm_services-3]: started with pid [14766]
process[real_listener-4]: started with pid [14782]
[INFO] [1646649869.148017]: start ...
[INFO] [1646649869.156531]: /dev/ttyAMA1,115200
```

Mercury Status

Joint Limit:

```
joint 1: -165 ~ +165
joint 2: -100 ~ +100
joint 3: -165 ~ +165
joint 4: -175 ~ +4
joint 5: -165 ~ +165
joint 6: -1 ~ +180
joint 7: -165 ~ +165
```

Connect Status: True

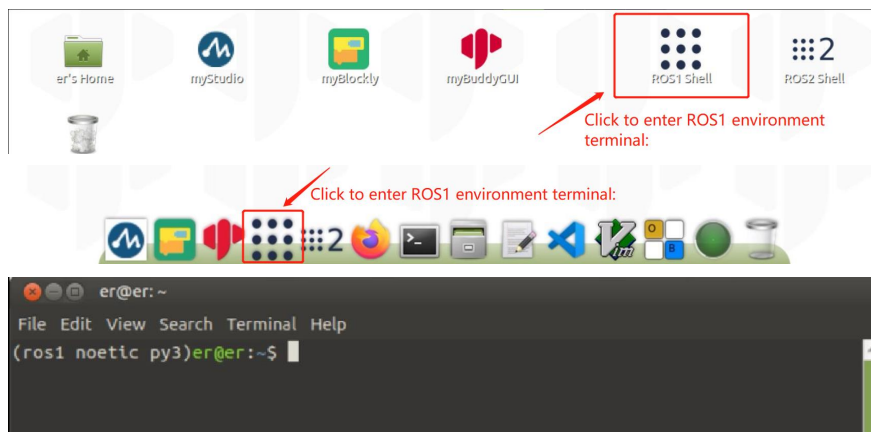
Servo Infomation: unknown

Servo Temperature: unknown

Atom Version: unknown

[INFO] [1646649869.427899]: ready

Then open another ROS1 environment terminal:



Then run the command:

```
roslaunch mercury_a1 teleop_keyboard.py
```

You will see the following output in the command line:

```
Mercury Teleop Keyboard Controller
-----
Moving options(control coordinations [x,y,z,rx,ry,rz]):

    w(x+)

a(y-)    s(x-)    d(y+)

    z(z-)    x(z+)

u(rx+)    i(ry+)    o(rz+)
j(rx-)    k(ry-)    l(rz-)

Gripper control:
    g - open
    h - close

Other:
    1 - Go to init pose
    2 - Go to home pose
    3 - Resave home pose
    q - Quit

currently:    speed: 50    change percent: 5
```

In this terminal, you can control the state of the robot arm and move it using the keys in the command line.

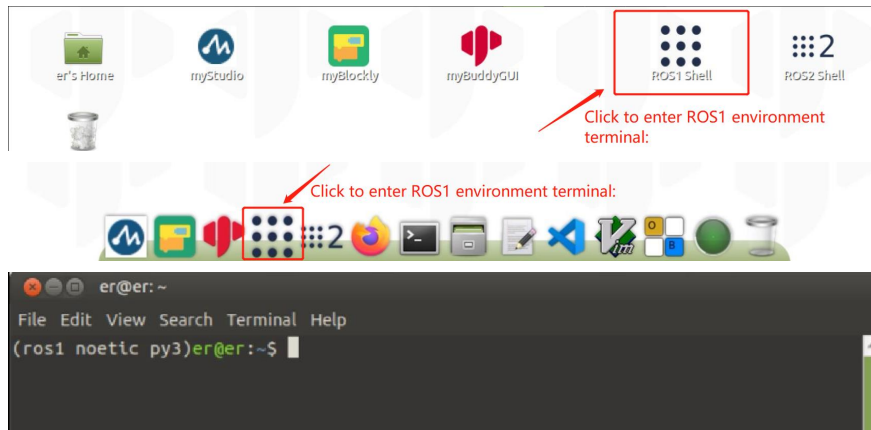
Parameters supported by this script:

- `_speed`: the movement speed of the robot arm
- `_change_percent`: movement distance percentage

5 Moveit use

`mercury_a1` has integrated the Moveit section.

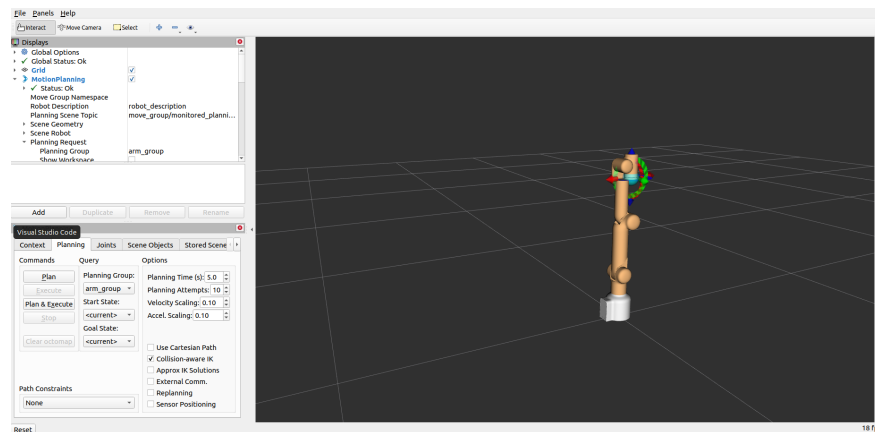
Click the `ROS1 Shell` icon on the desktop or the corresponding icon in the lower bar of the desktop to open the ROS1 environment terminal:



Then run the command:

```
roslaunch mercury_a1_moveit mercury_a1.launch
```

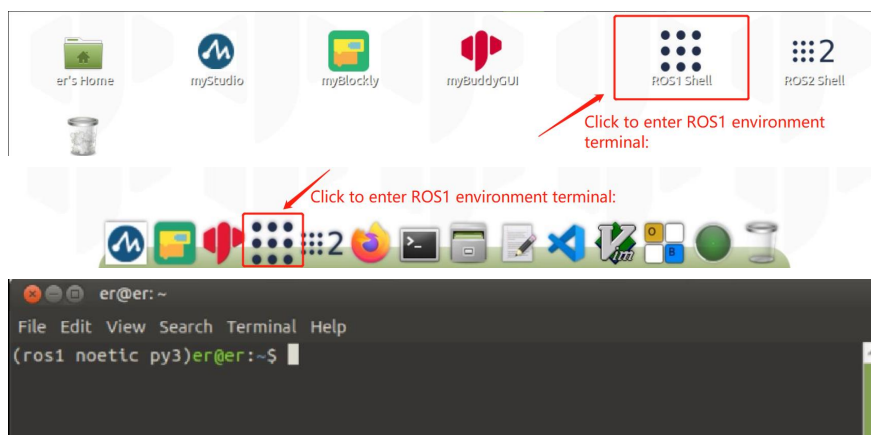
The operation effect is as follows:



Can be planned and executed to demonstrate the effect:



If you need to let the real robot arm execute the plan synchronously, you need to open another ROS1 environment terminal:



Then run the command:

```
rosrun mercury_a1_moveit sync_plan.py
```

Then plan and execute again to demonstrate the effect:



[← Previous Page](#) | [Next Section →](#)

ROS2 introduction

The predecessor of ROS2 is ROS, and ROS is the Robot Operating System (Robot Operating System). But ROS itself is not an operating system, but a software library and toolset. The emergence of Ros solved the communication problem of each component of the robot. Later, more and more robot algorithms were integrated into ROS. ROS2 inherited ROS, which is more powerful and better than ROS.

1 Design goals and features of ROS2

ROS2 has the historical mission of changing the era of intelligent robots. At the beginning of the design, it was considered to meet the needs of various robot applications.

- **Multi-Robot Systems:** In the future, robots will not be independent individuals, and communication and collaboration between robots are also required. ROS2 provides standard methods and communication mechanisms for the application of multi-robot systems.
- **Cross-platform:** Robot application scenarios are different, and the control platforms used will also be very different. In order to allow all robots to run ROS2, ROS2 can run on Linux, Windows, MacOS, and RTOS across platforms.
- **Real time:** Robot motion control and many behavior strategies require the robot to be real-time. For example, the robot must reliably detect pedestrians in front of it within 100ms, or complete kinematics and dynamics calculations within 1ms. ROS2 is a real-time like this Basic requirements are provided.
- **Productization:** A large number of robots have entered our lives, and there will be more and more in the future, ROS2 can not only be used in the robot research and development stage, but also can be directly installed in the product and go to the consumer market. This also poses a huge challenge to the stability and robustness of ROS2.
- **Project management:** Robot development is a complex system engineering. The project management tools and mechanisms for the whole process of design, development, debugging, testing, and deployment will also be reflected in ROS2, making it easier for us to develop a robot.

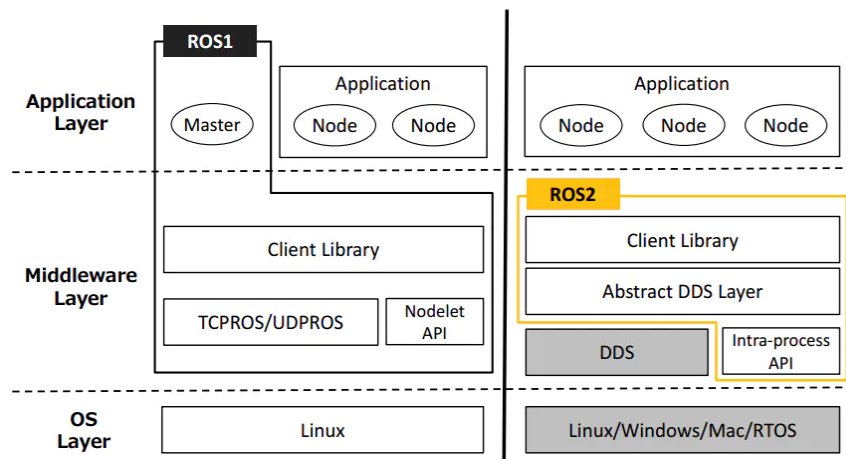
2 Release Version

The release version and maintenance cycle corresponding to ROS2 and Ubuntu.

ROS2 version	release date	Maintenance deadline	Ubuntu version
Dashing	2019.5	2021.5	Ubuntu 18.04 (Bionic Beaver)
Eloquent	2019.11	2020.11	Ubuntu 18.04 (Bionic Beaver)
Foxy	2020.6	2023.5	Ubuntu 20.04(Focal Fossa)
Galactic	2021.5	2022.11	Ubuntu 20.04(Focal Fossa)
Humble	2022.5	2027.5	Ubuntu 22.04(Jammy Jellyfish)

3 Comparison of ROS and ROS2

ROS2 redesigned the system architecture. The architecture changes between the two generations of ROS are as follows:



- **OS Layer:** In ROS2, it can be built on linux or other systems, even bare metal without an operating system.
- **Middleware Layer:** The communication system of ROS1 is based on TCPROS/UDPROS, while the communication system of ROS2 is based on DDS. DDS is a standard solution for data publishing/subscribing in distributed real-time systems.
- **Application Layer:** ROS1 relies on ROS Master, while in ROS2, a discovery mechanism called "Discovery" is used between nodes to help establish connections with each other.

ROS has designed a complete set of communication mechanisms (topics, services, parameters, actions) to simplify robot development. Through this mechanism, the various components of the robot can be connected. This mechanism has designed a node called Ros Master, and the communication of all other components must go through the master node. Once the master node hangs up, it will cause the communication of the entire robot system to collapse! Therefore, the instability of Ros cannot be used to make some high-risk robots such as automatic driving. In addition, there are the following disadvantages:

- Communication based on TCP has poor real-time performance and high system overhead
- Unfriendly to python3 support
- Messaging mechanism is not compatible
- No encryption mechanism, low security

ROS2 first removes the master node that exists in ROS. After removing the master node, each node can discover each other through the DDS node, each node is equal, and can realize one-to-one, one-to-many, and many-to-many communication. After using DDS for communication, reliability and stability have been enhanced.

Compared with **ROS** that only supports Linux systems, **ROS2** also supports windows, mac, and even RTOS platforms

[← Previous Section](#) | [Next Page →](#)

Raspberry Pi system environment

The Raspberry Pi version comes with Ubuntu (V-20.04) system and built-in **ROS2 Galactic** development environment. There is no need to build and manage it. You only need to update the `mercury_ros2` package.

`mercury_ros2` is a ROS2 package launched by Elephant Robot for its Mercury series robotic arms.

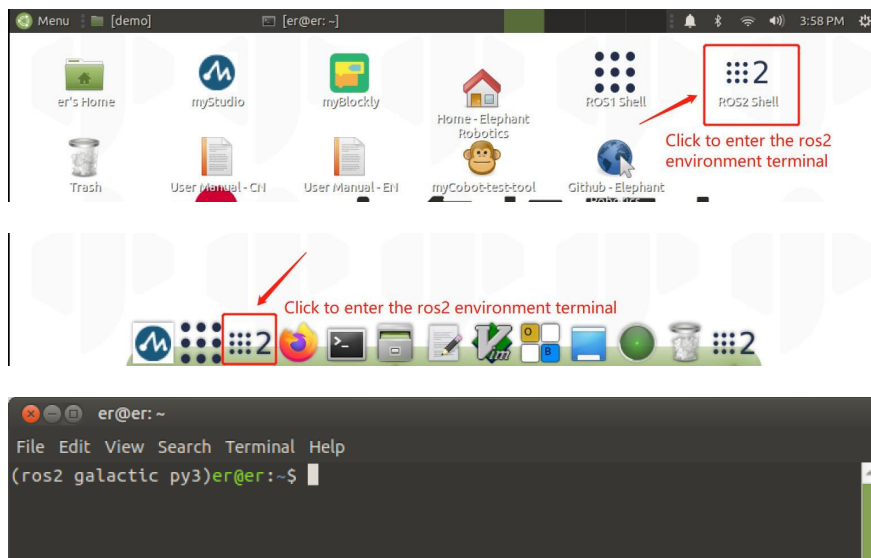
ROS2 project address: http://github.com/elephantrobotics/mercury_ros2

Robotic arm API driver library address:

<https://github.com/elephantrobotics/pymycobot>

1 Update mercury_ros2 package

In order to ensure that users can use the latest official packages in a timely manner, you can enter the `/home/er/colcon_ws/src` folder through the file manager, click the `ROS2 Shell` icon on the desktop or the corresponding icon in the lower bar of the desktop to open the ROS2 environment terminal:



Then run the command update:

```
# Clone the code on github
cd ~/colcon_ws/src
git clone https://github.com/elephantrobotics/mercury_ros2.git # Please check the atte
cd ~/colcon_ws # Back to work area
colcon build --symlink-install # Build the code in the workspace, --symlink-install: A
source install/setup.bash # add environment variables
```

Note: If the `mercury_ros2` folder already exists in the `/home/er/colcon_ws/src` (equivalent to `~/colcon_ws/src`) directory, you need to delete the original `mercury_ros2` before executing the above command. Among them, `er` in the directory path is the user name of the system.

So far, the ROS2 environment construction has been completed. You can learn [the basics of ROS2](#) or [ROS2 use cases](#)

[← Previous Page](#) | [Next Page →](#)

1 ROS2 project structure

1.1 colcon workspace

The colcon workspace is the directory where software packages are created, modified, and compiled. Colcon's workspace can be intuitively described as a warehouse, which contains various ROS project projects to facilitate system organization, management and calling.

- **Create workspace:**

```
mkdir -p ~/colcon_ws/src # Create folder
cd ~/colcon_ws/ # Enter the folder
colcon build # Build the code in the workspace.
```

Note: colcon supports option `--symlink-install`. This allows for faster iteration by changing installed files by changing files in the source space (such as Python files or other uncompiled resources). Avoid the need to recompile every time you modify your python script.

```
colcon build --symlink-install
```

A ROS workspace is a directory with a particular structure. Commonly there is a `src` subdirectory. Inside that subdirectory is where the source code of ROS packages will be located. Typically the directory starts otherwise empty.

colcon does out of source builds. By default it will create the following directories as peers of the `src` directory:

```
src/: colcon package for ROS2 (source code package)

build/: The location where intermediate files are stored. For each package, a subfolder
        will be created.

install/: The installation location of each package. By default, each package will be
          installed in a subfolder of this directory.

log/: Contains various logging information about each colcon call.
```

The directory structure of a ROS2 workspace is as follows:

```

Workspace --- Customized workspace.

|--- build: The directory where intermediate files are stored. A separate subdire
|--- install: Installation directory, a separate subdirectory will be created for
|--- log: Log directory, used to store log files.
|--- src: Directory used to store function package source code.

|-- C++ function package
    |-- package.xml: package information, such as: package name, version, aut
    |-- CMakeLists.txt: Configure compilation rules, such as source files, de
    |-- src: C++ source file directory.
    |-- include: header file directory.
    |-- msg: message interface file directory.
    |-- srv: Service interface file directory.
    |-- action: action interface file directory.

|-- Python function package
    |-- package.xml: package information, such as: package name, version, aut
    |-- setup.py: similar to CMakeLists.txt of C++ function package.
    |-- setup.cfg: Function package basic configuration file.
    |-- resource: resource directory.
    |-- test: stores test-related files.
    |-- Directory with the same name of the function package: Python source f

```

1.2 ROS2 package

Package is not only a software package on Linux, but also the basic unit of colcon compilation. The object we use `colcon build` to compile is each ROS2 package.

Create your own package:

- The command syntax for creating a software package using Python is:

```
ros2 pkg create --build-type ament_python <package_name>
```

- For example:

```
ros2 pkg create --build-type ament_python --node-name my_node my_package
```

2 Basic tool commands

In this chapter, you will learn about the common command tools of ROS2.

2.1 Topics

ROS 2 breaks complex systems down into many modular nodes. Topics are a vital element of the ROS graph that act as a bus for nodes to exchange messages. Topics are one of the main ways in which data is moved between nodes and therefore between different parts of the system.

Specific reference: [Official Tutorials](#)

- **topics help**

```
ros2 topics -h
```

- **Start turtlesim and keyboard control**

```
ros2 run turtlesim turtlesim_node  
ros2 run turtlesim turtle_teleop_key
```

- **Node Relationship Diagram**

```
rqt_graph
```

- **Learn about topic-related commands**

```
ros2 topics -h
```

- **topics list**

```
ros2 topic list  
ros2 topic list -t # Display the corresponding message type
```

- **View topic content**

```
ros2 topic echo <topic_name>  
ros2 topic echo /turtle1/cmd_vel
```

- **Display topic-related information, type**

```
ros2 topic info <topic_name>  
# Output /turtle1/cmd_vel topic related information  
ros2 topic info /turtle1/cmd_vel
```

- **Display interface related information**

```
ros2 interface show <msg_type>
# Output geometry_msgs/msg/Twist interface related information
ros2 interface show geometry_msgs/msg/Twist
```

- **Issue an order**

```
ros2 topic pub <topic_name> <msg_type> '<args>'
# Issue speed command
ros2 topic pub --once /turtle1/cmd_vel geometry_msgs/msg/Twist "{linear: {x: 2.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}"
# Issue speed commands at a certain frequency
ros2 topic pub --rate 1 /turtle1/cmd_vel geometry_msgs/msg/Twist "{linear: {x: 2.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}"
```

- **See how often topics are posted**

```
ros2 topic hz <topic_name>
# Output /turtle1/cmd_vel publish frequency
ros2 topic pub --rate 1 /turtle1/cmd_vel geometry_msgs/msg/Twist "{linear: {x: 2.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}"
```

2.2 Nodes

Each node in ROS should be responsible for a single, module purpose (e.g. one node for controlling wheel motors, one node for controlling a laser range-finder, etc). Each node can send and receive data to other nodes via topics, services, actions, or parameters. A full robotic system is comprised of many nodes working in concert. In ROS 2, a single executable (C++ program, Python program, etc.) can contain one or more nodes.

Specific reference: [Official Tutorials](#)

- **nodes help**

```
ros2 nodes -h
```

- **Start turtlesim and keyboard control**

```
ros2 run turtlesim turtlesim_node
ros2 run turtlesim turtle_teleop_key
```

- **View the node list**

```
ros2 node list
```

- **View Node Relationship Diagram**

```
rqt_graph
```

- **Remapping**

```
ros2 run turtlesim turtlesim_node --ros-args --remap __node:=my_turtle
ros2 node list
```

- **View node information**

```
ros2 node info <node_name>
ros2 node info /my_turtle
```

2.3 Services

Services are another method of communication for nodes in the ROS graph. Services are based on a call-and-response model, versus topics' publisher-subscriber model. While topics allow nodes to subscribe to data streams and get continual updates, services only provide data when they are specifically called by a client.

Specific reference: [Official Tutorials](#)

- **services help**

```
ros2 service -h
```

- **Start turtlesim and keyboard control**

```
ros2 run turtlesim turtlesim_node
ros2 run turtlesim turtle_teleop_key
```

- **View the service list**

```
ros2 service list
# Display service list and message type
ros2 service list -t
```

- **View the message types received by the service**

```
ros2 service type <service_name>
ros2 service type /clear
```

- **Find services that use a certain message type**

```
ros2 service find <type_name>
ros2 service find std_srvs/srv/Empty
```

- **View Service Message Type Definitions**

```
ros2 interface show <type_name>.srv
ros2 interface show std_srvs/srv/Empty.srv
```

- **Call the service command to clear the walking track**

```
ros2 service call <service_name> <service_type>
ros2 service call /clear std_srvs/srv/Empty
```

- **Spawn a new turtle**

```
ros2 service call /spawn turtlesim/srv/Spawn "{x: 2, y: 2, theta: 0.2, name: 'turtle2'}
```

2.4 Parameters

A parameter is a configuration value of a node. You can think of parameters as node settings. A node can store parameters as integers, floats, booleans, strings, and lists. In ROS 2, each node maintains its own parameters. For more background on parameters, please see the concept document.

Specific reference: [Official Tutorials](#)

- **parameters help**

```
ros2 param -h
```

- **Start turtlesim and keyboard control**

```
ros2 run turtlesim turtlesim_node
ros2 run turtlesim turtle_teleop_key
```

- **View service list**

```
ros2 param list
```

- **Get the parameter value**

```
ros2 param get <node_name> <parameter_name>
ros2 param get /turtlesim background_g
```

- **Set parameter values**

```
ros2 param set <node_name> <parameter_name> <value>
ros2 param set /turtlesim background_r 150
```

- **Export parameter values**

```
ros2 param dump <node_name>
ros2 param dump /turtlesim
```

- **Import parameters independently**

```
ros2 param load <node_name> <parameter_file>
ros2 param load /turtlesim ./turtlesim.yaml
```

- **Start the node and import parameters at the same time**

```
ros2 run <package_name> <executable_name> --ros-args --params-file <file_name>
ros2 run turtlesim turtlesim_node --ros-args --params-file ./turtlesim.yaml
```

2.5 Actions

Actions are one of the communication types in ROS 2 and are intended for long running tasks. They consist of three parts: a goal, feedback, and a result.

Actions are built on topics and services. Their functionality is similar to services, except actions are preemptable (you can cancel them while executing). They also provide steady feedback, as opposed to services which return a single response.

Actions use a client-server model, similar to the publisher-subscriber model (described in the topics tutorial). An “action client” node sends a goal to an “action server” node that acknowledges the goal and returns a stream of feedback and a result.

Specific reference: [Official Tutorials](#)

- **action help**

```
ros2 action -h
```

- **Start turtlesim and keyboard control**

```
ros2 run turtlesim turtlesim_node
ros2 run turtlesim turtle_teleop_key
```

Press G|B|V|C|D|E|R|T to achieve rotation, press F to cancel

- **View the server and client of the node action**

```
ros2 node info /turtlesim
```

- **View action list**

```
ros2 action list  
ros2 action list -t # show action type
```

- **view action info**

```
ros2 action info <action>  
ros2 action info /turtle1/rotate_absolute
```

- **View action message content**

```
ros2 interface show turtlesim/action/RotateAbsolute
```

- **Send action target information**

```
ros2 action send_goal <action_name> <action_type>  
ros2 action send_goal /turtle1/rotate_absolute turtlesim/action/RotateAbsolute "{theta:  
# With feedback information  
ros2 action send_goal /turtle1/rotate_absolute turtlesim/action/RotateAbsolute "{theta:
```

2.6 RQt

RQt is a graphical user interface framework that implements various tools and interfaces in the form of plugins. One can run all the existing GUI tools as dockable windows within RQt! The tools can still run in a traditional standalone method, but RQt makes it easier to manage all the various windows in a single screen layout.

Specific reference: [Official Tutorials](#)

You can run any RQt tools/plugins easily by:

```
rqt
```

- **rqt help**

```
rqt -h
```

- **Start turtlesim and keyboard control**


```
ros2 run turtlesim turtlesim_node
ros2 run turtlesim turtle_teleop_key
```

- Action Type Browser: / Plugins -> Actions -> Action Type Browser
- parameter reconfiguration: / Plugins -> configuration -> Parameter Reconfigure
- Node graph: /Node Graph
- control steering: /Plugins -> Robot Tools -> Robot Steering
- service invocation: /Plugins -> Services -> Service Caller
- Service Type Browser: Plugins -> Services -> Service Type Browser
- message release: Plugins -> Topics -> Message Publisher
- Message Type Browser: Plugins -> Topics -> Message Type Browser
- topic list: Plugins -> Topics -> Topic Monitor
- draw a graph: Plugins -> Visualization -> Plot
- **View logs: rqt_console**

```
ros2 run rqt_console rqt_console
ros2 run turtlesim turtlesim_node
ros2 topic pub -r 1 /turtle1/cmd_vel geometry_msgs/msg/Twist "{linear: {x: 2.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}
```

2.7 TF2

tf2 is the transform library, which lets the user keep track of multiple coordinate frames over time. tf2 maintains the relationship between coordinate frames in a tree structure buffered in time and lets the user transform points, vectors, etc. between any two coordinate frames at any desired point in time.

Specific reference: [Official Tutorials](#)

Let's start by installing the demo package and its dependencies.

```
sudo apt-get install ros-foxy-turtle-tf2-py ros-foxy-tf2-tools ros-foxy-tf-transformations
```

- **follow**
- launch starts 2 little turtles, the first little turtle automatically follows the second one

```
ros2 launch turtle_tf2_py turtle_tf2_demo.launch.py
```

- Control the movement of the first little turtle through the keyboard

```
ros2 run turtlesim turtle_teleop_key
```

- **View TF tree**

```
ros2 run tf2_tools view_frames.py  
evince frames.pdf
```

- **View the relationship between two coordinate systems**

```
ros2 run tf2_ros tf2_echo [reference_frame] [target_frame]  
ros2 run tf2_ros tf2_echo turtle2 turtle1
```

- **View TF relationships on rviz**

```
ros2 run rviz2 rviz2 -d $(ros2 pkg prefix --share turtle_tf2_py)/rviz/turtle_rviz.rviz
```

2.8 URDF

URDF is the Unified Robot Description Format for specifying robot geometry and organization in ROS.

Specific reference: [Official Tutorials](#)

- **Complete syntax**

```

<robot>

  # describe:

  # Parameters: name=""

  # Child node:

    <link>

      # Description:

      # Parameters: name=""

      # Child node:

        <visual>

          # describe:

          # Parameters:

          # child nodes:

            <geometry>

              # description

              # parameters

              # Child node:

                <cylinder />

                  # Description:

                  # Parameters:

                    # length="0.6"

                    # radius="0.2"

                <box />

                  # description

                  # Parameters:size="0.6 0.1 0.2"

                <mesh />

                  # Description

                  #Parameters: filename="package://urdf_tutorial/mes

            <collision>

              # Description: collision element, prioritized

              # parameters

              # child node

                <geometry>

            <inertial>

              # description

              # parameters

              # Child nodes:

                <mass />

                  # description: mass

                  # Parameters: value=10

                <inertia />

                  # Description: Inertia

                  # Parameters: i+"Cartesian product of xyz" (9 in t

            <origin />

              # Description:

              # Parameters:

                # rpy="0 1.5 0"

                # xyz="0 0 -0.3"

```

```

        <material />

        # Description
        # Parameters: name="blue"

    <joint>
        # Description
        # Parameters:
            # name=""
            # type=""
            # fixed
            # prismatic
        # child node
        <parent />
            # Description
            # Parameters: link=""
        <child />
            # Description:
            # Parameters: link=""
        <origin />
            # Description:
            # Parameters: xyz="0 -0.2 0.25"
        <limit />
            # Description
            # Parameters:
                # effort="1000.0"    maximum effort
                # lower="-0.38"      Joint upper limit (radians)
                # upper="0"          Joint lower limit (radians)
                # velocity="0.5"     Maximum velocity
        <axis />
            # Description: Press ? axis rotation
            # Parameters: xyz="0 0 1", along the Z axis

    <material>
        # Description:
        # Parameters: name="blue"
        # child node:
        <color />
            # description:
            # Parameters: rgba="0 0 0.8 1"

```

- **Install dependent libraries**

```

sudo apt install ros-foxy-joint-state-publisher-gui ros-foxy-joint-state-publisher
sudo apt install ros-foxy-xacro

```

- **Download the source code**

```
cd ~/dev_ws  
git clone -b ros2 https://github.com/ros/urdf_tutorial.git src/urdf_tutorial
```

- **Compiling the source code**

```
colcon build --packages-select urdf_tutorial
```

- **Running the example**

```
ros2 launch urdf_tutorial display.launch.py model:=urdf/01-myfirst.urdf
```

2.9 Launch

The launch system in ROS 2 is responsible for helping the user describe the configuration of their system and then execute it as described. The configuration of the system includes what programs to run, where to run them, what arguments to pass them, and ROS-specific conventions which make it easy to reuse components throughout the system by giving them each a different configuration. It is also responsible for monitoring the state of the processes launched, and reporting and/or reacting to changes in the state of those processes.

Launch files written in Python, XML, or YAML can start and stop different nodes as well as trigger and act on various events.

Specific reference: [Official Tutorials](#)

Setup

Create a new directory to store your launch files:

```
mkdir launch
```

Write the launch file

Let's put together a ROS 2 launch file using the turtlesim package and its executables. As mentioned above.

Copy and paste the complete code into the launch/turtlesim_mimic_launch.py file:



```

from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    return LaunchDescription([
        Node(
            package='turtlesim',
            namespace='turtlesim1',
            executable='turtlesim_node',
            name='sim'
        ),
        Node(
            package='turtlesim',
            namespace='turtlesim2',
            executable='turtlesim_node',
            name='sim'
        ),
        Node(
            package='turtlesim',
            executable='mimic',
            name='mimic',
            remappings=[
                ('/input/pose', '/turtlesim1/turtle1/pose'),
                ('/output/cmd_vel', '/turtlesim2/turtle1/cmd_vel'),
            ]
        )
    ])

```

Run the ros2 launch file

To run the launch file created above, enter into the directory you created earlier and run the following command:

The syntax format is:

```
ros2 launch <package_name> <launch_file_name>
```

```
cd launch
ros2 launch turtlesim_mimic_launch.py
```

- **launch help**

```
ros2 launch -h
```

- **running node**

```
ros2 launch turtlesim multisim.launch.py
```

- **Check the parameters of the launch file**

```
ros2 launch turtlebot3_fake_node turtlebot3_fake_node.launch.py -s
ros2 launch turtlebot3_fake_node turtlebot3_fake_node.launch.py --show-arguments
ros2 launch turtlebot3_bringup robot.launch.launch.py -s
```

- **Run the launch file with parameters**

```
ros2 launch turtlebot3_bringup robot.launch.launch.py usb_port:=/dev/opencr
```

- **Run the node and debug**

```
ros2 launch turtlesim turtlesim_node.launch.py -d
```

- **Only output node description**

```
ros2 launch turtlesim turtlesim_node.launch.py -p
```

- **running components**

```
ros2 launch composition composition_demo.launch.py
```

2.10 Run

run is used to run a single node, component program

- **run help**

```
ros2 run -h
```

- **running node**

```
ros2 run turtlesim turtlesim_node
```

- **Run node with parameters**

```
ros2 run turtlesim turtlesim_node --ros-args -r __node:=turtle2 -r __ns:=/ns2
```

- **Run component container**

```
ros2 run rclcpp_components component_container
```

- **running components**

```
ros2 run composition manual_composition
```

2.11 Package

A package can be considered a container for your ROS 2 code. If you want to be able to install your code or share it with others, then you'll need it organized in a package. With packages, you can release your ROS 2 work and allow others to build and use it easily.

Package creation in ROS 2 uses ament as its build system and colcon as its build tool. You can create a package using either CMake or Python, which are officially supported, though other build types do exist.

Specific reference: [Official Tutorials](#)

Creating a workspace

Create a new directory for every new workspace. The name doesn't matter, but it is helpful to have it indicate the purpose of the workspace. Let's choose the directory name `ros2_ws`, for "development workspace":

```
mkdir -p ~/ros2_ws/src  
cd ~/ros2_ws/src
```

- **pkg help**

```
ros2 pkg -h
```

- **List Feature Packs**

```
ros2 pkg executable turtlesim
```

- **Output a function package executable program**

```
ros2 pkg executable turtlesim
```

- **Create a Python package**

Make sure you are in the `src` folder before running the package creation command.

```
cd ~/ros2_ws/src
```

The command syntax for creating a new package in ROS 2 is:


```
ros2 pkg create --build-type ament_python <package_name>
# you will use the optional argument --node-name which creates a simple Hello World ty
ros2 pkg create --build-type ament_python --node-name my_node my_package
```

- **Build a package**

Putting packages in a workspace is especially valuable because you can build many packages at once by running `colcon build` in the workspace root. Otherwise, you would have to build each package individually.

```
# Return to the root of your workspace:
cd ~/ros2_ws
# Now you can build your packages:
colcon build
```

- **Source the setup file**

To use your new package and executable, first open a new terminal and source your main ROS 2 installation.

Then, from inside the `ros2_ws` directory, run the following command to source your workspace:

```
source install/setup.bash
```

Now that your workspace has been added to your path, you will be able to use your new package's executables.

- **Use the package**

To run the executable you created using the `--node-name` argument during package creation, enter the command:

```
ros2 run my_package my_node
```

[← Previous Page](#) | [Next Page →](#)

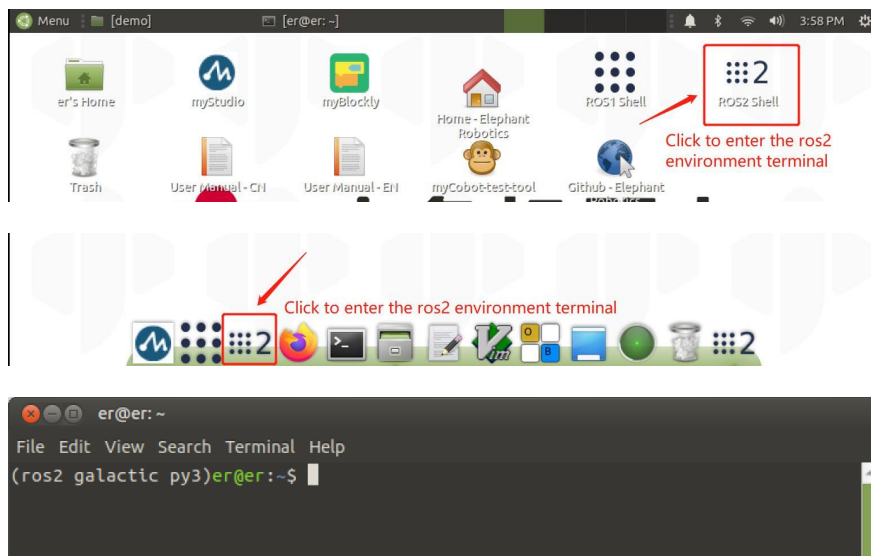
Brief introduction and use of rviz2

Rviz2 is a visualization tool for displaying messages in the robot environment, providing a 3D perspective to view the robot's status and activities. It can help developers better understand the current status and activities of the robot, as well as other visual messages. Rviz2 provides a series of visualization tools that can help developers better understand the status and activities of robots, such as visual coordinate systems, laser scanning messages, point cloud messages, robot models, etc. Using Rviz2, robotic systems can be easily viewed and debugged to better achieve robotic goals.

1 Introduction to rviz2

The successful installation of ros2 indicates that rviz2 is also successfully installed together, because the installation of ros2 includes rviz2.

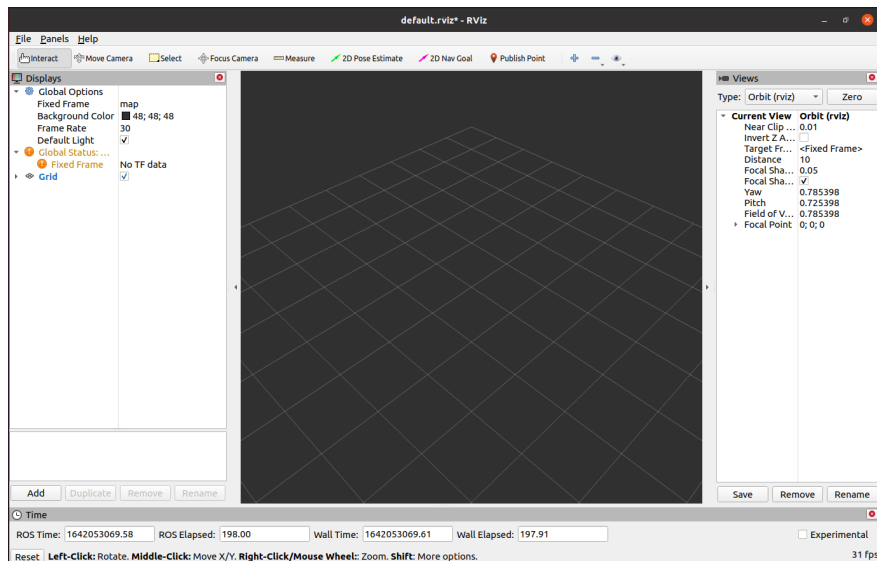
Click the ROS2 Shell icon on the desktop or the corresponding icon in the lower bar of the desktop to open the ROS2 environment terminal:



enter the command to open rviz2

```
ros2 run rviz2 rviz2  
# or  
rviz2
```

Open rviz2 and display the following interface:



1.1 Introduction of each area

- On the left is the list of monitors, a monitor is something that draws something in the 3D world and may have some options available in the display list. Including functions such as adding, deleting, copying, renaming plug-ins, displaying plug-ins, and setting plug-in properties.
- Above is the toolbar, which allows users to use various function buttons to select tools with multiple functions
- The middle part is the 3D view: it is the main screen where various data can be viewed in 3D. The background color, fixed frame, grid, etc. of the 3D view can be set in detail in the Global Options and Grid items displayed on the left.
- Below is the time display area, including system time and ROS time.
- The right side is the observation angle setting area, and different observation angles can be set.

We only give a rough introduction in this part. If you want to know more detailed content, you can go to the [user guide](#) to view it.

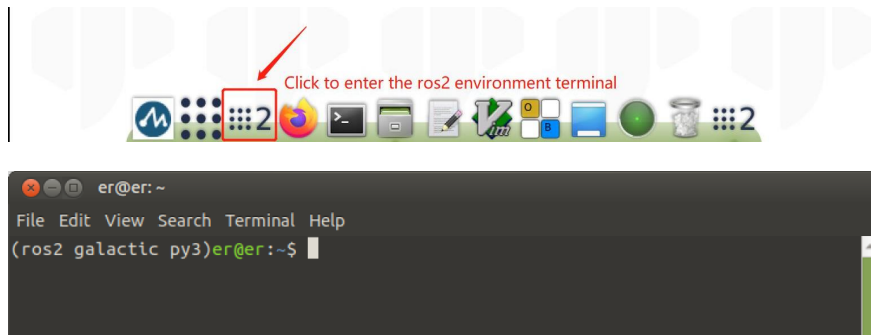
2 Simple use

Launch through launch file

This example is based on the fact that you have completed [Environment Setup](#) and successfully copied the company's code from GitHub.

Click the `ROS2 Shell` icon on the desktop or the corresponding icon in the lower bar of the desktop to open the ROS2 environment terminal:





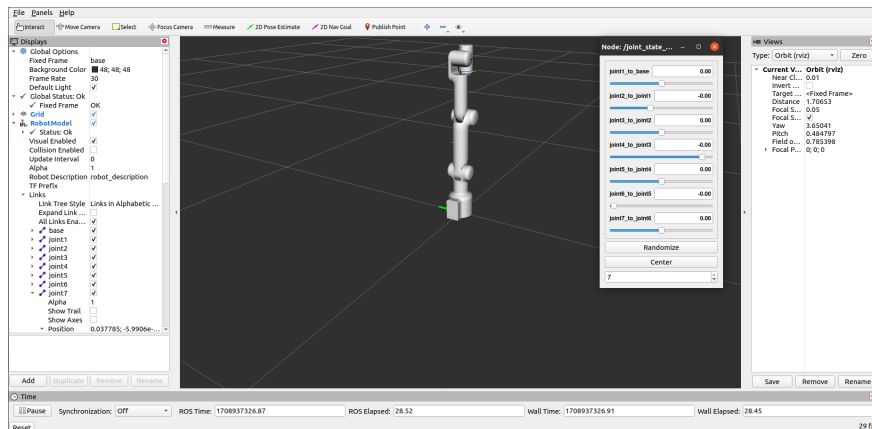
Enter the following command to configure **ROS2** environment.

```
cd ~/colcon_ws/
colcon build --symlink-install
source install/setup.bash
```

Enter again:

```
ros2 launch mercury_a1 test.launch.py
```

Open rviz2 and get the following result:



If you want to know more information about rviz, you can go to the [official documentation](#) to view it.

[← Previous Page](#) | [Next Page →](#)

Control of the robotic arm

1 Slider Control

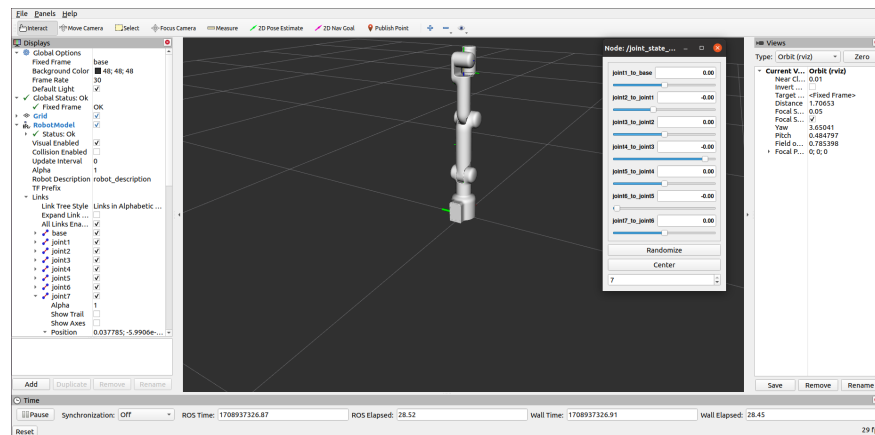
Click the `ROS2 Shell` icon on the desktop or the corresponding icon in the lower bar of the desktop to open the ROS1 environment terminal:



Then run the command:

```
ros2 launch mercury_a1 slider_control.launch.py
```

It will **open rviz** and a **slider component** and you will see something like this:



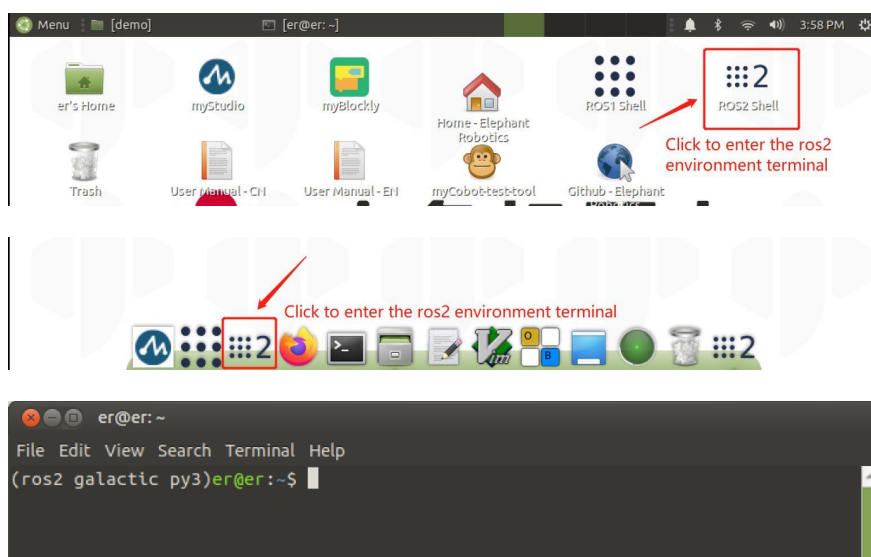
Then you can **control the movement of the model in rviz2 by dragging the slider**. The real Mercury A1 will move with it.

Please note: Since the robot arm will move to the current position of the model when the command is input, please make sure that the model in rviz2 does not appear to be worn out before you use the command Do not quickly drag the slider after connecting the robotic arm to prevent damage to the robotic arm

2 Model Follow

In addition to the above controls, we can also **make the model follow the movement of the real robotic arm** .

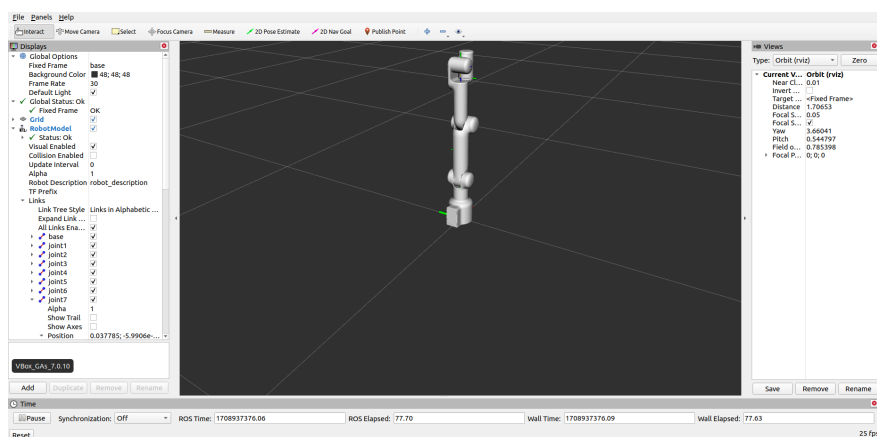
Click the `ROS2 Shell` icon on the desktop or the corresponding icon in the lower column of the desktop to open the ROS2 environment terminal:



Then run the command:

```
ros2 launch mercury_a1 mercury_follow.launch.py
```

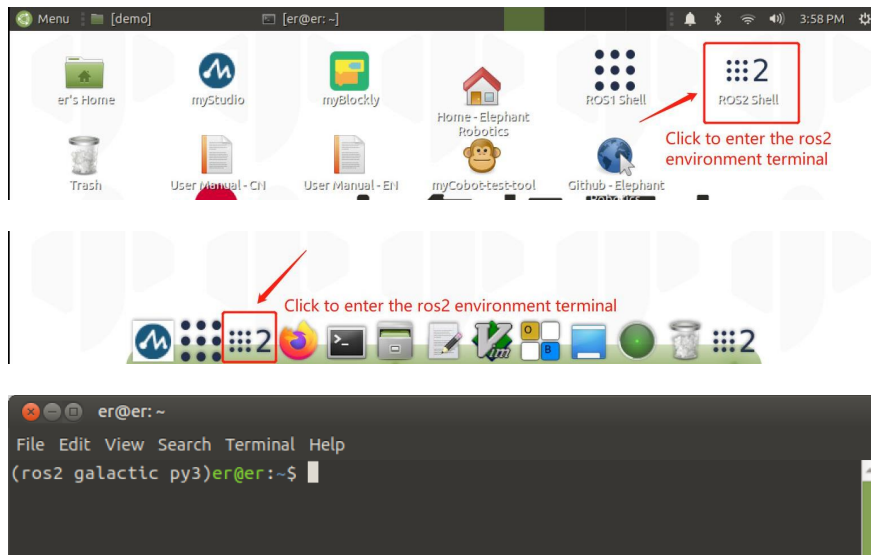
It will open rviz to show the model following effect .



3 GUI Control

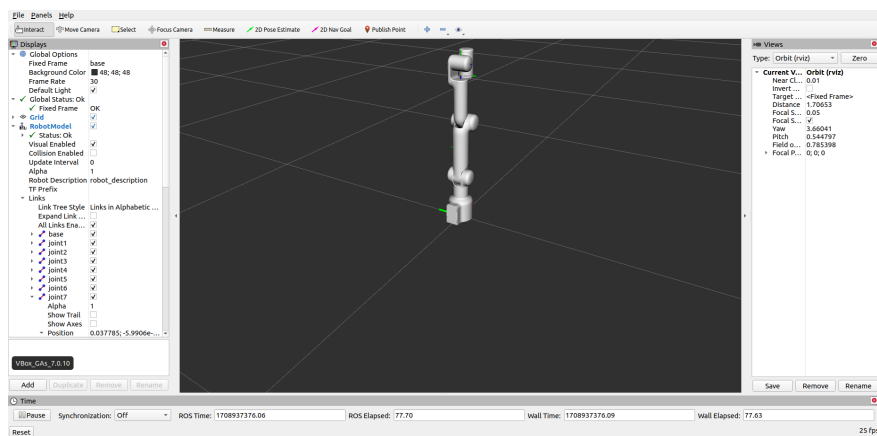
On the basis of the previous, this package also **provides a simple GUI control interface** . This method means that the real robotic arms are linked with each other, please connect to mycobot.

Click the `ROS2 Shell` icon on the desktop or the corresponding icon in the lower column of the desktop to open the ROS2 environment terminal:

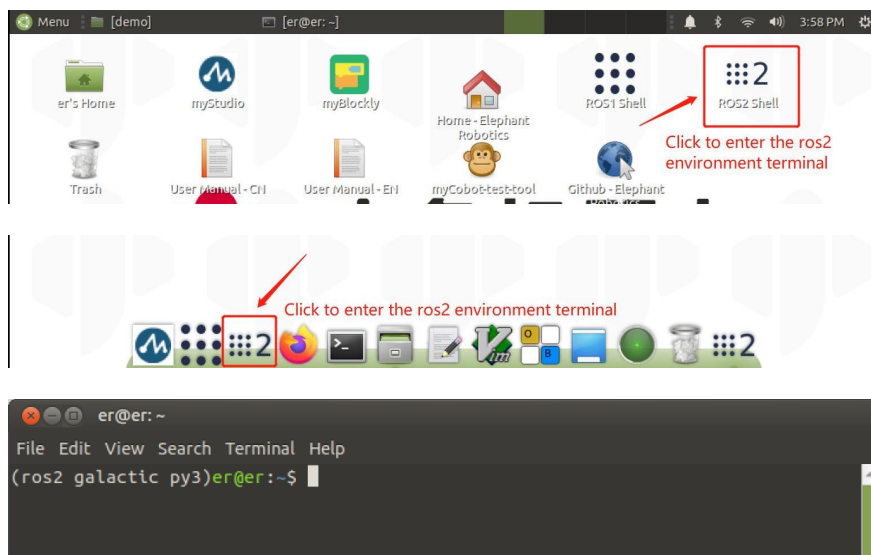


Then run the command:

```
ros2 launch mercury_a1 simple_gui.launch.py
```

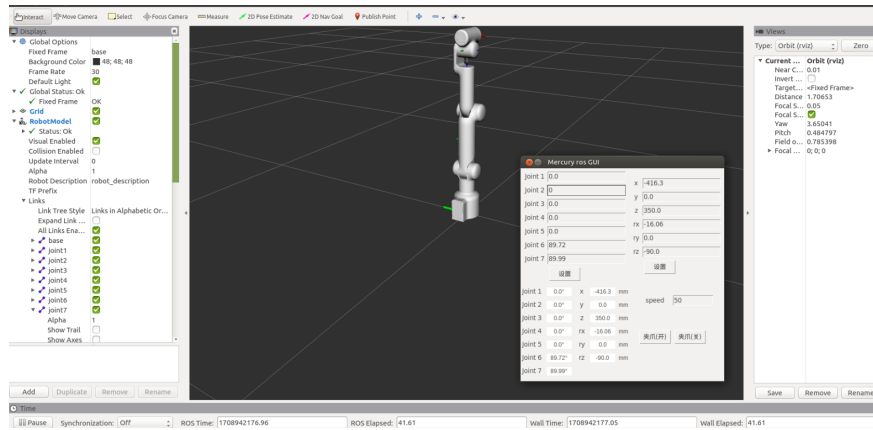


Next, open another ROS2 environment terminal:



Then run the command:

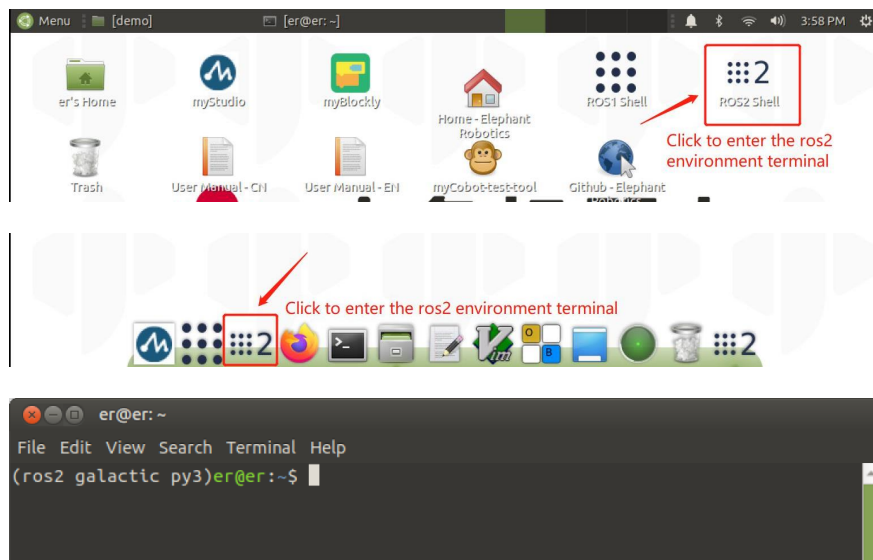
```
ros2 run mercury_a1 simple_gui
```



4 Keyboard Control

The function of keyboard control is added in the package of `mercury_a1`, and it is synchronized in real time in rviz. This function relies on pythonApi, so make sure to connect with the real robotic arm.

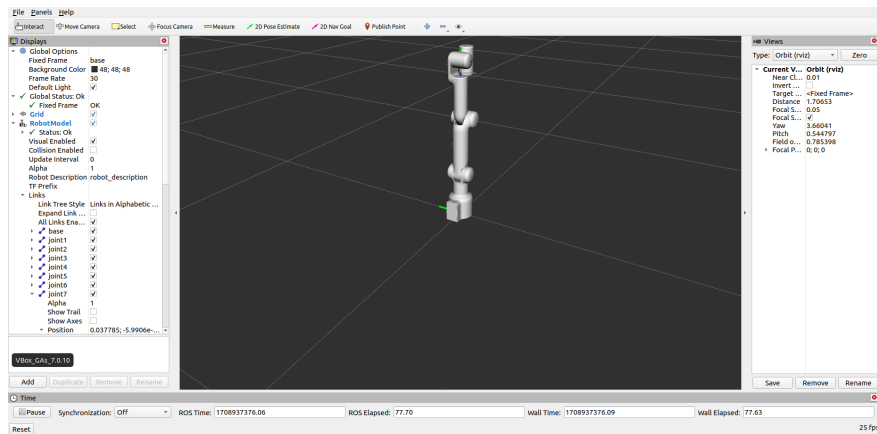
Click the `ROS2 Shell` icon on the desktop or the corresponding icon in the lower column of the desktop to open the ROS2 environment terminal:



Then run the command:

```
ros2 launch mercury_a1 teleop_keyboard.launch.py
```

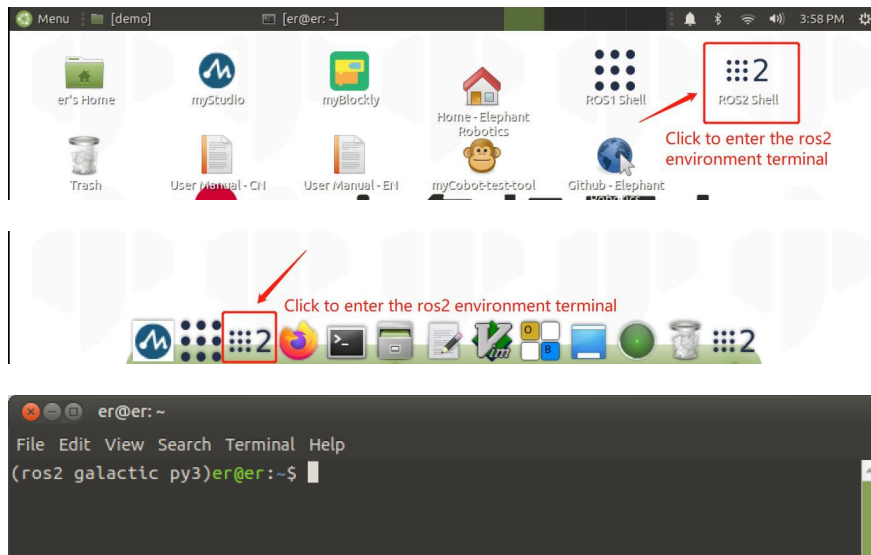
The operation effect is as follows:



The command line will output Mercury A1 information, as follows:

```
[INFO] [launch]: All log files can be found below /home/er/.ros/log/2023-09-05-10-43-1
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [robot_state_publisher-1]: process started with pid [6293]
[INFO] [rviz2-2]: process started with pid [6295]
[INFO] [follow_display-3]: process started with pid [6297]
[robot_state_publisher-1] Parsing robot urdf xml string.
[robot_state_publisher-1] Link link1 had 1 children
[robot_state_publisher-1] Link link2 had 1 children
[robot_state_publisher-1] Link link3 had 1 children
[robot_state_publisher-1] Link link4 had 1 children
[robot_state_publisher-1] Link link5 had 1 children
[robot_state_publisher-1] Link link6 had 1 children
[robot_state_publisher-1] Link link7 had 0 children
[robot_state_publisher-1] [INFO] [1659667392.703132973] [robot_state_publisher]: got s
[robot_state_publisher-1] [INFO] [1659667392.703485410] [robot_state_publisher]: got s
[robot_state_publisher-1] [INFO] [1659667392.703545198] [robot_state_publisher]: got s
[robot_state_publisher-1] [INFO] [1659667392.703571119] [robot_state_publisher]: got s
[robot_state_publisher-1] [INFO] [1659667392.703587512] [robot_state_publisher]: got s
[robot_state_publisher-1] [INFO] [1659667392.703603744] [robot_state_publisher]: got s
[robot_state_publisher-1] [INFO] [1659667392.703619685] [robot_state_publisher]: got s
[robot_state_publisher-1] [INFO] [1659667392.703619685] [robot_state_publisher]: got s
[rviz2-2] [INFO] [1659667393.588026632] [rviz2]: Stereo is NOT SUPPORTED
[rviz2-2] [INFO] [1659667393.588472253] [rviz2]: OpenGL version: 3.1 (GLSL 1.4)
[rviz2-2] [INFO] [1659667393.766777360] [rviz2]: Stereo is NOT SUPPORTED
[rviz2-2] Parsing robot urdf xml string.
[follow_display-3] [INFO] [1659667394.310152595] [follow_display]: port:/dev/ttyAMA1,
```

Next, open another ROS2 environment terminal:



Then run the command:

```
ros2 run mercury_a1 teleop_keyboard
```

You will see the following output on the command line:

```
Mercury Teleop Keyboard Controller
-----
Moving options(control coordinations [x,y,z,rx,ry,rz]):
    w(x+)

    a(y-)    s(x-)    d(y+)

    z(z-) x(z+)

    u(rx+)  i(ry+)  o(rz+)
    j(rx-)  k(ry-)  l(rz-)

Gripper control:
    g - open
    h - close

Other:
    1 - Go to init pose
    2 - Go to home pose
    3 - Resave home pose
    q - Quit

currently:    speed: 60    change percent: 10
```

In this terminal, you can control the state of the robot arm and move the robot arm through the keys in the command line.

Note: The coordinate control operation can only be performed after the 2 command is executed.

[← Previous Page](#) | [Next Section →](#)

C++

<<<<<<< HEAD 使用c++语言，您可以通过我们公司开发的c++动态库，进行自由开发（坐标控制、角度控制、io控制、夹爪控制等），控制我们公司已经研发出来的部分机器人。



C++是什么？

C++是C语言的继承，它既可以进行C语言的过程化程序设计，又可以以抽象数据类型为特点的基于对象的程序设计，还可以进行以继承和多态为特点的面向对象的程序设计。

C++擅长面向对象程序设计的同时，还可以进行基于过程的设计，因而C++就适应的问题规模而论，大小由之。

C++不仅拥有计算机高效运行的实用性特征，同时还致力于提高大规模程序的编程质量与程序设计语言的问题描述能力。

编程开发

部分集成开发环境（IDE）

===== Using c++ language, you can freely develop (coordinate control, Angle control, io control, claw control, etc.) through the c++ dynamic library developed by our company to control some of the robots that our company has developed.



What is C++?

C++ is the inheritance of C language, it can not only carry out procedural programming of C language, but also carry out object-based programming characterized by abstract data types, but also carry out object-oriented programming characterized by inheritance and polymorphism.

While C++ is good at object-oriented programming, it can also do process-based programming, so the size of C++ in terms of the size of the problem it ADAPTS to depends on it.

C++ not only has the practical characteristics of efficient operation of computers, but also aims to improve the programming quality of large-scale programs and the problem description ability of programming language.

Programming development

Partial Integrated Development Environment (IDE)



b4cbb522d469c9fae1bce398ab2eea

533b152ea9

Visual Studio (Visual C++)

Dev C++

C++ Builder

<<<<<< HEAD

编译器

=====

compiler



b4cbb522d469c9fae1bce398ab2eea

533b152ea9

Ultimate++

Digital Mars

C-Free

MinGW

<<<<<<< HEAD

C++ 环境搭建

1 确认开发目标

Mercurycpp是一个和Mercury进行串口通讯的接口程序，调用的是我们公司自己开发的Mercury库，里面有简单的使用案例。如果您想通过C++进行自由开发，控制我们公司已经研发出来的机器人，那么它是您的选择。

支持的机械臂型号：MercurycppA1

运行Mercurycpp所需要的软件：vs2019、qt5.12.10、vsaddin（qt插件）。

2 Windows环境配置

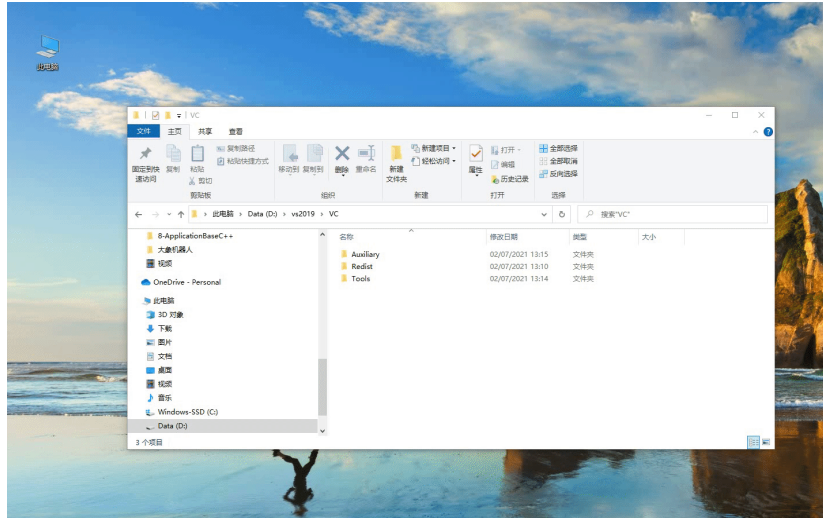
2.1 安装vs2019

- 下载：
首先在官网下载[vs2019](#)。
- 安装：
安装完成后，会出现下图所示界面，主要选择“通用Windows平台开发、使用C++的桌面开发、ASRNET和Web开发”这3个（此处只是建议，具体可根据自己的需求选择，vs2019安装时间较长）。



- 环境变量配置：
此电脑--》右键 属性--》高级系统设置--》环境变量--》看系统变量处，点击新建--》变量名：VCINSTALLDIR 变量值：找到Redist所在目录(如：

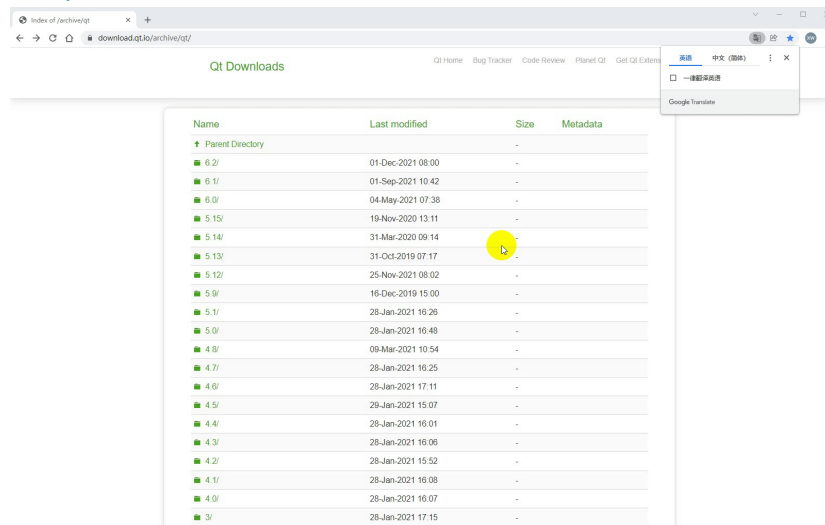
D:\vs2019\VC) ， 具体如下图所示：



2.2 安装qt5.12.10

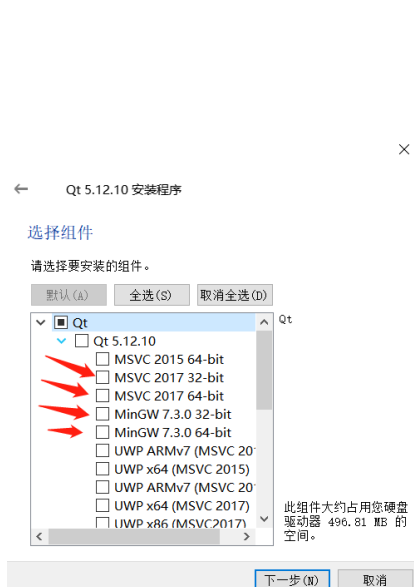
- 下载：

下载qt5.12.10及以上版本都可以，具体操作如下图：



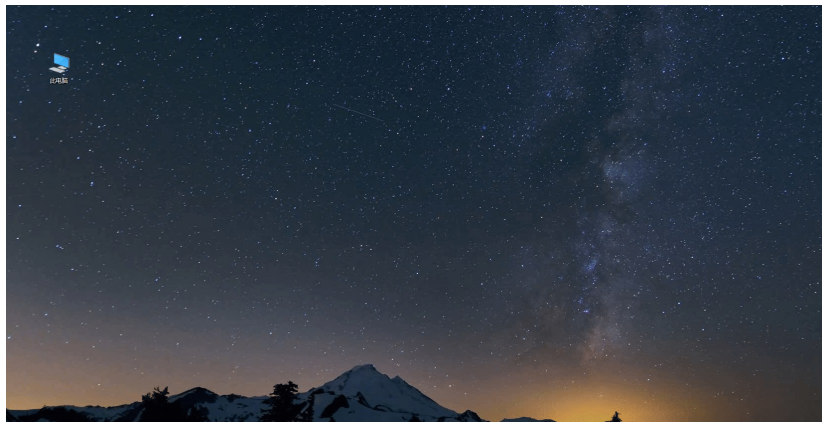
- 安装：

首先登录qt账号，没有就先注册。接下来会出现选择组件的界面，windows上选择MinGW和MSVC即可，具体如下图所示：



- 环境变量配置：

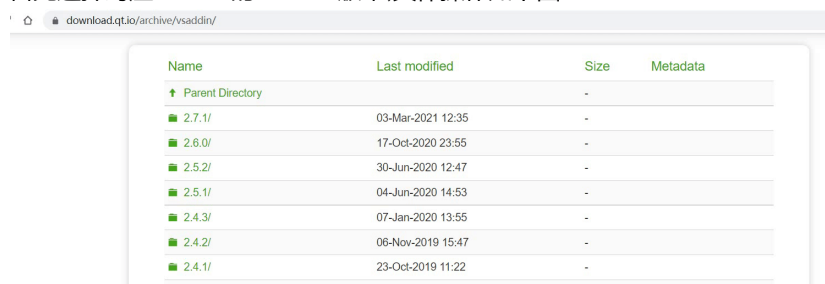
此电脑--》右键 属性--》高级系统设置--》环境变量--》看系统变量处，点击新建--》变量名：QTDIR 变量值：msvc2017_64所在目录（如：D:\qt5.12.10\5.12.10\msvc2017_64，具体看自己电脑的安装路径），具体如下图所示：



2.3 安装qt插件vsaddin

- 下载：

首先选择对应vs2019的vsaddin版本,具体操作如下图：

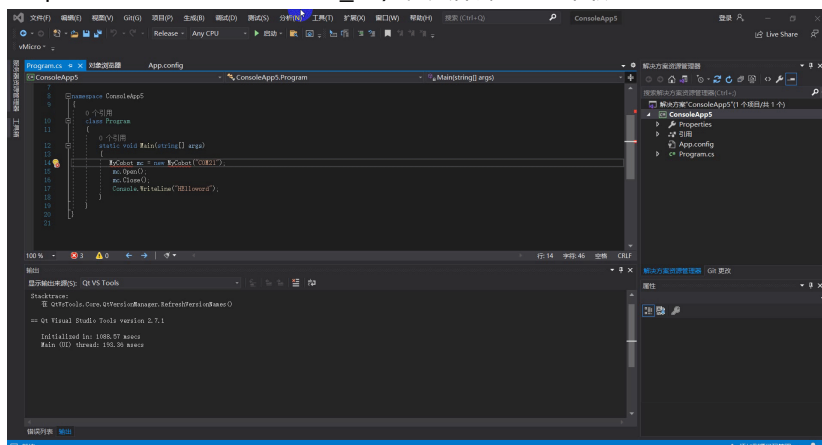


- 安装：直接安装即可

- 配置：

vs2019菜单栏 扩展--》QT VS ToolS--》QT Versions--》add new qtversion Path选择msvc2017_64所在路径（如：

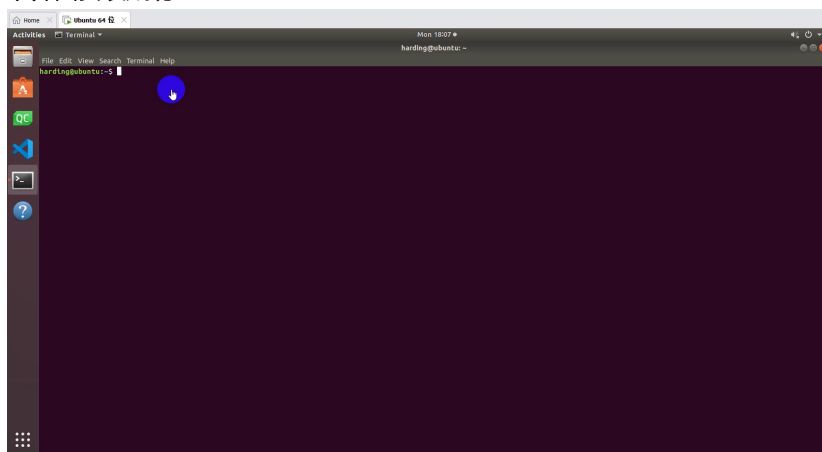
D:\qt5.12.10\5.12.10\msvc2017_64) ，具体操作如下图所示：



3 Linux环境配置

3.1 安装qt5.12.10

- 下载：
下载地址和Windows一样，选择linux上的安装包即可，具体可看上面的8.1.2.2。
- 安装：
命令行安装：运行“安装包名称”，如果没有执行权限，加执行权限：sudo chmod +x “安装包名称”，然后进入图形界面，和Windows一样；
图形界面安装：和Windows一样。
建议直接普通用户权限安装qt，安装成功后可以执行qmake --version，出现如下界面安装成功：



- 配置：

打开配置文件，普通用户安装qt：vi ~/.bashrc，root用户安装qt：vi ~/.profile。在配置文件中添加：

**export QTDIR="qt所在目录" (如:
export
QTDIR=\$HOME/Qt/5.12.10/gcc_64)
， 具体如下图所示:**

C++ environment setup

1 Identify development goals

MercuryCpp is an interface program for serial communication with Mercury, which calls the Mercury library developed by our company, which has a simple use case. If you want to freely develop through c++ and control the robots that our company has developed, then it is your choice.

Supported arm model: mercury

Software required to run MercuryCpp: vs2019, qt5.12.10, vsaddin (qt plugin).

2 Configure the Windows environment

2.1 Install vs2019

- Download:

First download it on the official website[vs2019](#).

- Installation:

After the installation is complete, the interface shown in the following figure will appear, mainly select "universal Windows platform development, desktop development using C++, ASRNET and Web development" these three (here is only a suggestion, you can choose according to your needs, vs2019 takes a long time to install).

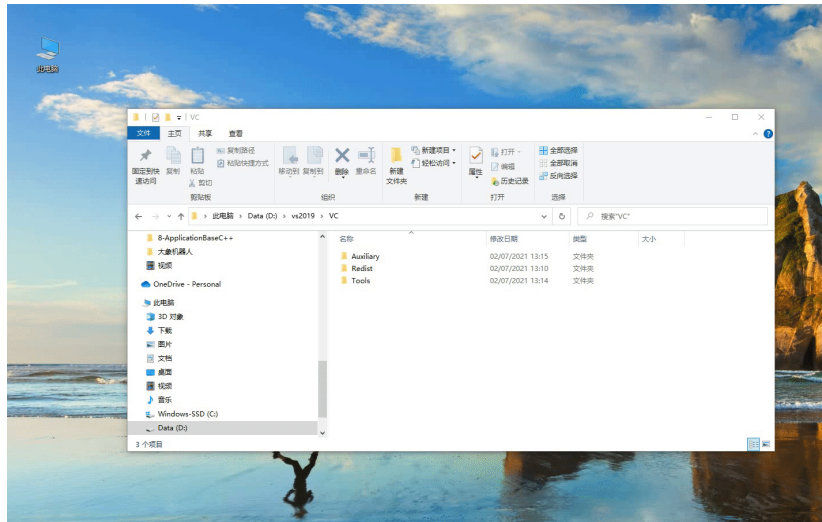


- Environment variable configuration:

This computer -- "right click properties --" Advanced system Settings --

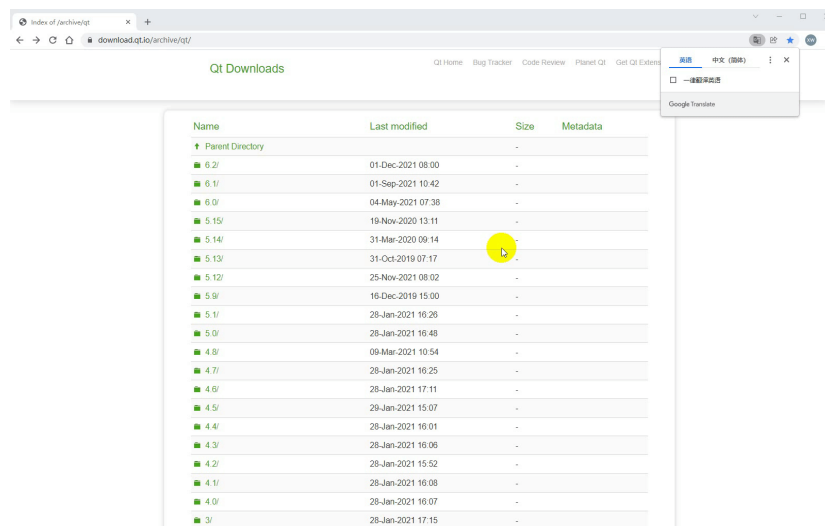
"Environment variables --" look at the system variables, click New -- "variable

name: VCINSTALLDIR Variable value: Find the directory where Redist is located (such as: D:\vs2019\VC), as shown below:



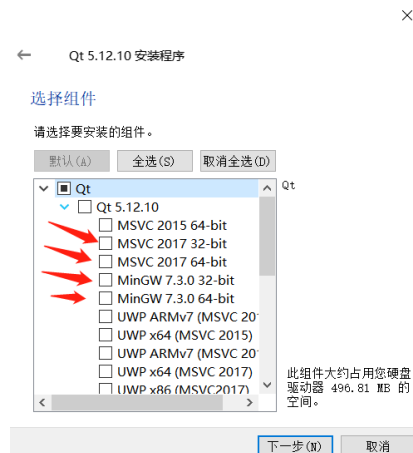
2.2 Install qt5.12.10

- Download:
Download [qt5.12.10](#) The above versions can be used. The specific operation is as follows:

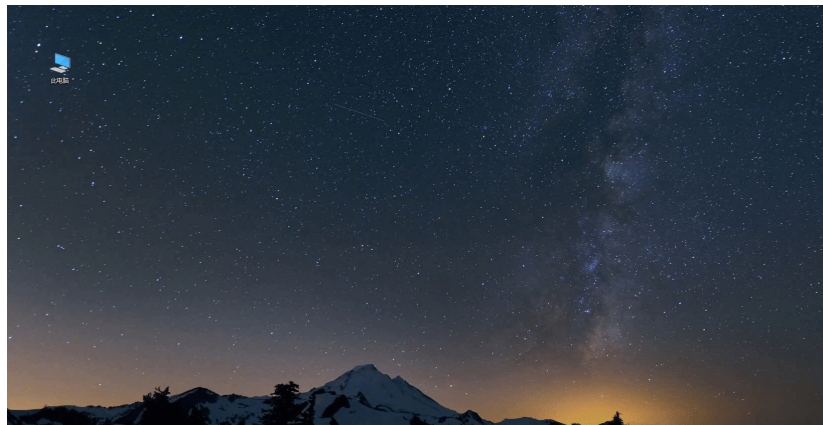


- Installation:
First login to the qt account, do not register first. Next, the interface for selecting components will appear, MinGW and MSVC can be selected on

windows, as shown below:

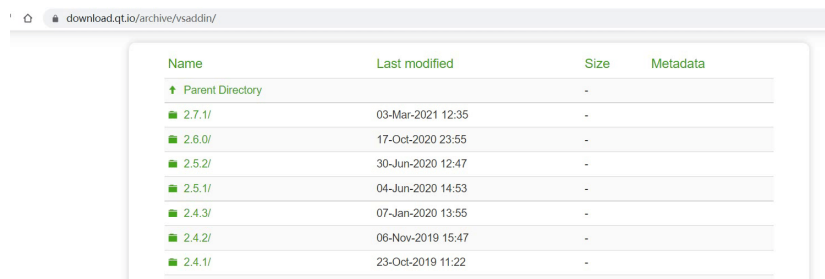


- Environment variable configuration:
This computer -- "Right click properties --" Advanced System Settings --
"Environment variables --" look at the system variables, click new -- "variable
name: QTDIR Variable value: msvc2017_64 directory (such as:
D:\qt5.12.10\5.12.10\msvc2017_64, see the installation path of your
computer), as shown in the following figure:



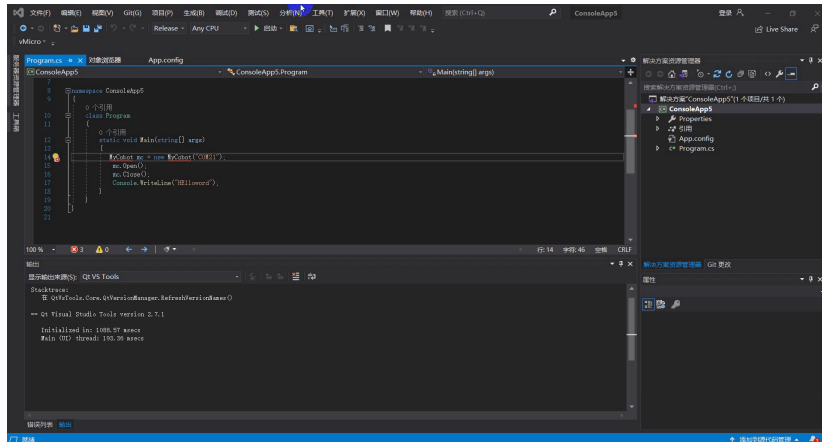
2.3 Install the qt plugin vsaddin

- Download:
First select the corresponding vs2019 [vsaddin](#) Version, the specific operation
is as follows:



- Installation: Directly install
- Configuration:
vs2019 menu bar extension -- QT VS Tools-- QT Versions-- add new

qtversion Path Select the path where msvc2017_64 resides (for example: D:\qt5.12.10\5.12.10\msvc2017_64), the specific operation is shown as follows:



3 Configure the Linux environment

3.1 Installing qt5.12.10

- Download:

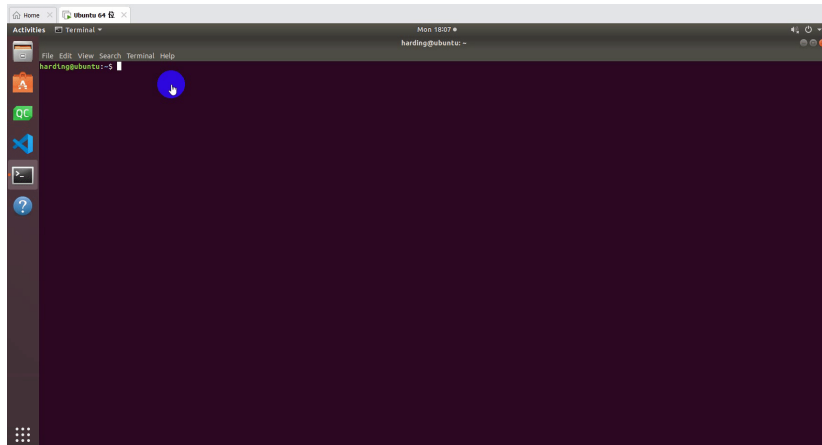
The download address is the same as for Windows, select the installation package on linux, see 8.1.2.2 above.

- Installation:

Command line installation: Run./ "Installation package name", if there is no execution permission, add the execution permission: sudo chmod +x "Installation package name", and then enter the graphical interface, the same as Windows;

GUI installation: same as Windows.

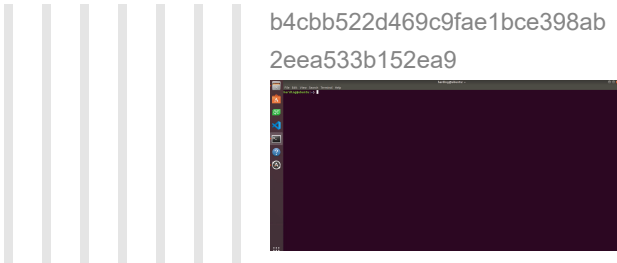
You are advised to install qt as a common user. After the installation is successful, run qmake --version. The following information is displayed:



- Configuration:

Open the configuration file, and install qt: vi ~/.bashrc as a common user and qt: vi ~/.profile as a root user. Add export QTDIR="directory where qt is

located" (for example, export QTDIR=\$HOME/Qt/5.12.10/gcc_64) to the configuration file, as shown in the following figure:



<<<<<<< HEAD

MercuryCpp 编译运行

1 .下载

1.1 源代码下载

在github上下载[MycobotCpp](#)。

1.2 动态库下载

[Dependency library download](#)(如需下载最新版本, 请选择“**Windows**”或“**Linux**”, Windows操作系统需下载后缀。zip, Linux操作系统需下载。tar.gz)

2 在Linux上运行

将serial文件夹和mercurylib文件夹拷贝到er/目录下。Serial是串行通信库, mercurylib是c++库。我自己的代码是用main.cpp编写的。

2.1 编译和构建

- mkdir build && cd build
- cmake ..
- cmake --build .

2.2 运行

- 命令行运行: ./MercuryLIB(在本例中, 从构建目录)

注意

如果你不使用cmake编译，就像你直接在MFC中使用它一样，按照如下所示进行配置：

MercuryCpp Compile run

1 Download

1.1 Source code download

Download it on github[MycobotCpp](#)。

1.2 Dynamic library download

[Dependency library download](#)(To download the latest version, select **Windows** or **Linux**, suffix.zip is the required library for Windows,.tar.gz is the required library for Linux)

2 Run on Linux

Copy the serial folder and mercurylib folder to the er/ directory. serial is the serial communication library, and mercurylib is the c++ library. My own code is written in main.cpp.

2.1 Compile and build

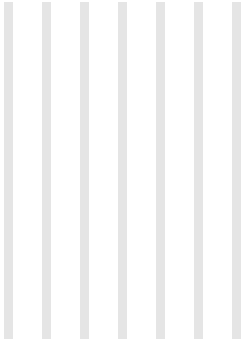
- mkdir build && cd build
- cmake ..
- cmake --build .

2.2 Run

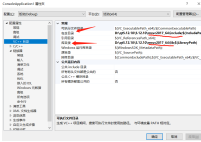
- Command line run:./MercuryLIB (in this case, from the build directory)

Look out

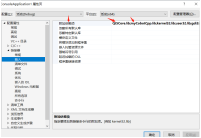
If you do not compile with cmake, as if you are using it directly in MFC, configure as shown below:



b4cbb522d469c9fae1bce398ab2eea



533b152ea9



<<<<<<< HEAD

Mercury API

使用以下函数接口时，请先进入我们的API库，否则无法成功运行。有关库的下载和导入，请参见Mercurycpp编译和运行。

下面的API使用坐标、角度控制和速度参考范围如下：

坐标范围 (X, + - 466 - mm, Y: + - 466 - mm, Z: - 240 ~ 531 mm, RX/RX/RZ: + - 180 °)

X,Y,Z 速度: 1~200; **RX,RY,RZ: 速度** 1~40

角度范围 (J1: + - 178 °, J2: - 74 ° ~ 130 ° (机械臂放在工作台上，不超过110°), J3: + - 178 °, J4: - 180 ° ~ 10 °, J5: + - 178 °, J6: - 20 ° ~ 273 °, J7: + - 180 °)

速度: 1~150

1 实例化Mercury

1.1 Mercury(char* portid, int baud)

功能：实例化Mercury

返回值：Mercury类型，Mercury对象的单例实例

参数说明：参数1:串口名 参数2:波特率

注意：在调用下面的API时，不需要在单独实例化，调用此API即可

2 机器人整体运行状态 Overall Status

2.1 PowerOn()

功能：机械臂上电

返回值：无

参数说明：无

注意：机械臂上电后，无法手动移动机械臂

2.2 PowerOff()

功能：机械臂掉电

返回值：无

参数说明：无

注意：机械臂上电后，如果想要手动移动机械臂，可以调用此API

2.3 isPoweredOn()

功能：控制核心连接状态查询

返回值：int类型,0--未上电, 1--上电

参数说明：无

2.4 releaseAllServos()

功能：机器人关闭力矩输出

返回值：无

参数说明：无

2.5 FocusAllServos()

功能：机器人打开力矩输出

返回值：无

参数说明：无

3 输入程序控制模式 MDI Mode and Robot Control (Manual Data Input)

3.1 GetAngle()

功能：获取单关节角度

返回值：double类型,对应关节的角度。

参数说明：参数1：关节号(1-7)

3.2 GetAngles();

功能：读取所有角度

返回值：double*类型，指向一个存放1-7关节角度的double数组

参数说明：无

3.3 WriteAngle(int joint, double value, int speed)

功能：发送单独角度

返回值：无

参数说明：参数1：关节号(1-7)，参数2：角度，参数3：速度 (1-100)

3.4 WriteAngles(double* angles, int speed)

功能：发送全部关节角度

返回值: 无

参数说明: 参数1: 全部角度, 参数2: 速度(0-100)

3.5 GetCoords()

功能: 获取全部坐标

返回值: double*类型, 指向一个double数组

参数说明: 无

3.6 WriteCoord(int axis, double value, int speed)

功能: 发送单参数坐标

返回值: 无

参数说明: 参数1: 坐标号 (1-6 (X-RZ)), 参数2: 坐标 (X: +-466mm, Y: +-466mm, Z: -290~641mm, RX/RZ: +-180°), 参数3: 速度(0-100)

3.7 WriteCoords(double* coords, int speed)

功能: 发送全部坐标

返回值: 无

参数说明: 参数1: 坐标, 参数2: 速度(0-100)

3.8 ProgramPause()

功能: 控制核心暂停进行的指令, 可以停止所有的移动指令

返回值: 无

参数说明: 无

3.9 isProgramPaused()

功能: 检测程序是否暂停了移动的指令

返回值: int类型, 0-未停止/1-停止

参数说明: 无

3.10 ProgramResume()

功能: 程序恢复移动指令, 继续完成上一个指令

返回值: 无

参数说明: 无

3.11 TaskStop()

功能: 控制核心停止所有的移动指令, 程序停止

返回值: 无

参数说明: 无

3.12 checkRunning()

功能: 检测机器人是否在运动

返回值: int类型, 1-正在运动/0-未运动

参数说明: 无

3.13 IsInPosition(double* coords, bool value)

功能: 检查机械臂是否到达指定点(角度或坐标)

返回值: bool类型, 返回false——指定的点没有到达, 返回true——指定的点已经到达

参数说明: 参数1: 指向包含所有角度或坐标的数组的指针 参数2: 0或1 (坐标为1, 角度为0)

4 JOG运行与操作 JOG mode and operation

4.1 JogCoord(int axis, int direction, int speed)

功能: 使机械臂朝坐标轴方向运动

返回值: 无

参数说明: 参数1: 坐标号 (1-6, x y z rx ry rz), 参数2: 方向(1--正方向, 0--反方向), 参数3: 速度 (范围: 0-100)

注意: 此API会让机械臂一直朝坐标轴正反方向运动, 到达限位后或者中途调用TaskStop才会停止运动

4.2 JogAngle(int joint, int direction, int speed)

功能: 使某个关节运动直至TaskStop或是到达限位

返回值: 无

参数说明: 参数1: 关节号 (1-7), 参数2: 方向 (1--正方向, 0--反方向), 参数3: 速度 (范围: 0-100)

注意: 此API会让机械臂关节一直朝正反方向运动, 到达限位后或者中途调用TaskStop才会停止运动

4.3 JogIncrement(int joint, double increment, int speed)

功能: 使某个关节运动设定的角度增量

返回值: 无

参数说明: 参数1: 关节号 (1-7) , 参数2: 关节增量值, 速度 (范围: 0-100)

注意: 机械臂进行步进运动: 比如当前关节1坐标为-100, 增量值为50, 运动完后, 关节1将为50

5 坐标控制姿态偏转角Coordinate Control Attitude Deflection Angle

5.1 GetSolutionAngles()

功能: 获取零空间偏转角数值

返回值: double类型零空间偏转角数值

参数说明: 无

5.2 SetSolutionAngles(double angle ,int speed)

功能: 设置零空间偏转角数值

返回值: 无

参数说明: 参数1: 零空间偏转角数值 (范围+-90°) ;参数2: 速度 (范围0-100)

6 底座basicIO控制 basic IO Control

6.1 setBasicOutput(int pin_number, int pin_signal)

功能: 设置底座IO输出

返回值: 无

参数说明: 参数1: 引脚号 (basic输出引脚号) , 参数2: 状态 (0--低电平, 1--高电平)

6.2 getBasicInput(int pin_number)

功能: 读取底座IO输出

返回值: 引脚状态, 0--低电平, 1--高电平

参数说明: 参数1: :basic输入引脚号

7 设置软件关节限位 Software Joint Limit

7.1 getJointMin(int joint)

功能: 读取关节最小角度

返回值: double类型, 关节最小能运行到的角度

参数说明：参数1：关节号 (1-6)

7.2 getJointMax(int joint)

功能：读取关节最大角度

返回值：double类型，关节最大能运行到的角度

参数说明：参数1：关节号 (1-6)

7.3 setJointMin(int joint, double angle)

功能：设置关节最小角度

返回值：无

参数说明：参数1：关节号 (1-6) ， 参数2：最小角度（关节最小能运行到的角度）

7.4 setJointMax(int joint, double angle)

功能：设置关节最大角度

返回值：无

参数说明：参数1：关节号 (1-6) ,参数2：最大角度（关节最大能运行到的角度）

8 拖动示教 Drag and play

8.1 DragTechSave()

功能：启动拖动示教录制点位

返回值：无

参数说明：无

8.2 DragTechExecute()

功能：执行示教点位

返回值：无

参数说明：无

8.3 DragTechPause()

功能：暂停点位录制

返回值：无

参数说明：无

注意：暂停之后机械臂处于放松状态

9 笛卡尔空间坐标参数设定 Cartesian space coordinate parameter setting

9.1 setToolReference(double x,double y, double z, double rx, double ry, double rz)

功能：设置工具坐标系

返回值：无

参数说明：参数1：x，参数2：y，参数3：z，参数4：rx，参数5：ry，参数6：rz (参数范围：x(-2800~280.00)(10xmm)，y (-2800~280.00)(10xmm)，z(-2800~280.00)(10xmm)，rx (-3140~3140)(1000xRd)，ry(-3140~3140)(1000xRd)，rz (-3140~3140)(1000xRd))

9.2 getToolReference()

功能：获取工具坐标系

返回值：double *类型，指向存放6个double类型数据 (x,y,z,rx,ry,rz) 的数组

参数说明：无

9.3 setWorldReference(double x, double y, double z, double rx, double ry, double rz)

功能：设置世界坐标系

返回值：无

参数说明：参数1：x，参数2：y，参数3：z，参数4：rx，参数5：ry，参数6：rz (参数范围：x(-2800~280.00)(10xmm)，y (-2800~280.00)(10xmm)，z(-2800~280.00)(10xmm)，rx (-3140~3140)(1000xRd)，ry(-3140~3140)(1000xRd)，rz (-3140~3140)(1000xRd))

9.4 getWorldReference()

功能：获取世界坐标系

返回值：double *类型，指向存放6个double类型数据 (x,y,z,rx,ry,rz) 的数组

参数说明：无

9.5 setMovementType(int movement)

功能：设置移动类型

返回值：无

参数说明：参数1：1-moveI, 0-moveJ

9.6 getMovementType()

功能：获取移动类型

返回值：int类型， 1-moveI, 0-moveJ

参数说明：无

9.7 setEndType(int endtype)

功能：设置末端坐标系

返回值：无

参数说明：参数1:0-法兰 1-工具

9.8 getEndType()

功能：获取末端坐标系

返回值：int类型， 0-法兰 1-工具

参数说明：无

10 圆弧运动

10.1 WriteMovC(float transpointx, float transpointy, float transpointz, float transpointrx, float transpointry, float transpointrz, float endpointx , float endpointy , float endpointz, float endpointrx, float endpointry, float endpointrz, int speed)

功能：圆弧轨迹运动

返回值：无

参数说明：transpoint为圆弧途径点，endpoint为圆弧结束点。参数1: transpointx, 参数2: transpointy, 参数3: transpointz, 参数4: transpointrx, 参数5: transpointry, 参数6: transpointrz, 参数7: endpointx , 参数8: endpointy , 参数9: endpointz , 参数10: endpointrx, 参数11: endpointry, 参数12: endpointrz, 参数13: 速度 (1-100)

11 关节电机状态 servo state value

11.1 GetServoSpeeds()

功能：获取关节运行速度，运行时才有速度，单位：步/s

返回值：int*类型的指针，指向一个存放1-7关节速度大小的int[7]数组，范围：+-3000

参数说明：无

11.2 GetServoCurrents()

功能：获取关节电流

返回值：int*类型的指针，指向一个存放1-7关节电流大小的int[7]数组，范围：0-5000

参数说明：无

11.3 GetServoStatus()

功能：获取关节状态,0-->各关节正常

返回值：int*类型的指针，指向一个存放1-7关节状态的int[7]数组，状态：0--ok, 1-欠压, 2-过温, 3-过流

参数说明：无

Mercury API

When using the following function interface, please first enter our API library, otherwise it will not run successfully. For the download and import of libraries, please see Mercurycpp compile and run.

The following API uses coordinates, Angle control, and speed reference ranges as follows:

coordinates (X, + - 466 - mm, Y: + - 466 - mm, Z: - 240 ~ 531 mm, RX/RX/RZ: + - 180 °)

X,Y,Z speed: 1~200; RX,RX,RZ: Speed 1~40

Angle (J1: + - 178 °, J2: - 74 ° ~ 130 ° (robotic arm on the table, not more than 110 °), J3: + - 178 °, J4: - 180 ° ~ 10 °, J5: + - 178 °, J6: - 20 ° ~ 273 °, J7: + - 180 °)

Speed: 1~150

1 Instantiate Mercury

1.1 Mercury(char* portid, int baud)

Function: Instantiate Mercury

Return value: Mercury type, singleton instance of Mercury object

Parameter description: Parameter 1: serial port name Parameter 2: baud rate

Note: When calling the following API, you do not need to instantiate it separately, just call this API

2 Overall running Status of the robot

2.1 PowerOn()

Function: The mechanical arm is powered on

Returned value: None

Parameter Description: None

Note: After the robot arm is powered on, the robot arm cannot be moved manually

2.2 PowerOff()

Function: mechanical arm power off

Returned value: None

Parameter Description: None

Note: After the robot arm is powered on, if you want to move the robot arm manually, you can call this API

2.3 isPoweredOn()

Function: Query core connection status

Returned value: int type, 0-- not powered on, 1-- powered on

Parameter Description: None

2.4 releaseAllServos()

Function: Robot off torque output

Returned value: None

Parameter Description: None

2.5 FocusAllServos()

Function: Robot opens torque output

Returned value: None

Parameter Description: None

3 MDI Mode and Robot Control (Manual Data Input) MDI mode and Robot control (manual data input)

3.1 GetAngle()

Function: Obtain single joint Angle

Return value: Type double, corresponding to the Angle of the joint.

Parameter Description: Parameter 1: Joint number **(1-7)**

3.2 GetAngles();

Function: Read all angles

Return value: Type double*, pointing to a double array of 1-7 joint angles

Parameter Description: None

3.3 WriteAngle(int joint, double value, int speed)

Function: Send individual Angle

Returned value: None

Parameter description: Parameter 1: joint number **(1-7)**, parameter 2: Angle, parameter 3: speed

3.4 WriteAngles(double* angles, int speed)

Function: Send all joint angles

Returned value: None

Parameter description: Parameter 1: a pointer to the full array of angles, parameter 2: speed

3.5 GetCoords()

Function: Get all coordinates

Return value: Type double*, pointing to a double array

Parameter Description: None

3.6 WriteCoord(int axis, double value, int speed)

Function: Send single parameter coordinates

Returned value: None

Parameter description: Parameter 1: coordinate number **(1-6 (X-RZ))**, parameter 2: coordinate, parameter 3: speed

3.7 WriteCoords(double* coords, int speed)

Function: Send all coordinates

Returned value: None

Parameter description: Parameter 1: a pointer to the array holding all coordinates, parameter 2: speed

3.8 ProgramPause()

Function: Control the core to suspend the instruction, you can stop all the moving instructions

Returned value: None

Parameter Description: None

3.9 isProgramPaused()

Function: Detect whether the program has paused the moving instruction

Return value: int type, 0- not stopped / 1- stopped

Parameter Description: None

3.10 ProgramResume()

Function: The program resumes the moving instruction and continues to complete the previous instruction

Returned value: None

Parameter Description: None

3.11 TaskStop()

Function: The control core stops all moving instructions, and the program stops

Returned value: None

Parameter Description: None

3.12 checkRunning()

Function: Detect whether the robot is moving

Return value: int type, 1- moving / 0- not moving

Parameter Description: None

*IsInPosition(double coords, bool is_linear = true) Function: Check whether the robot arm has reached the specified point (Angle or coordinate)
Return value: bool type, return false-- the specified point has not been reached,*

return true-- the specified point has been reached

*Parameter Description: Parameter 1: a pointer to an array containing all angles or coordinates parameter 2:0 or 1 *(coordinates are 1, angles are 0)*

4 JOG mode and operation JOG mode and operation

4.1 JogCoord(int axis, int direction, int speed)

Function: Make the mechanical arm move in the axis direction

Returned value: None

Parameter description: Parameter 1: Coordinate number (**1-6, x y z rx ry rz**), parameter 2: Direction (**1-- positive direction, 0-- negative direction**), parameter 3: Speed (**range: 0-100**)

Note: This API will keep the robot arm moving in the positive and negative direction of the axis, until it reaches the limit or calls TaskStop in the middle of the motion

4.2 JogAngle(int joint, int direction, int speed)

Function: Make a joint move until TaskStop or reach the limit

Returned value: None

Parameter description: Parameter 1: Joint number (**1-7**), Parameter 2: Direction (**1-- positive direction, 0-- reverse direction**), parameter 3: Speed (**range: 0-100**)

Note: This API will keep the robot arm joint moving in a positive and negative direction until it reaches the limit or calls TaskStop in the middle

4.3 JogIncrement(int joint, double increment, int speed)

Function: Make a joint movement set Angle increment

Returned value: None

Parameter description: Parameter 1: Joint number (**1-7**) , Parameter 2: joint increment value, speed (**Range: 0-100**)

Note: step movement of the robot arm: for example, the current joint 1 coordinate is -100, the increment value is 50, and after the movement, joint 1 will be 50

5 Coordinate Control Attitude Deflection Angle. Coordinate control attitude deflection Angle

5.1 GetSolutionAngles()

Function: Obtain zero space deflection Angle value

Return value: double null space deflection Angle value

Parameter Description: None

5.2 SetSolutionAngles(double angle,int speed)

Function: Set zero space deflection Angle value

Returned value: None

Parameter Description: Parameter 1: zero space deflection Angle value (**range +-90°**); Parameter 2: Speed (**range 0-100**)

6 Base basicIO Control basicIO Control

6.1 setBasicOutput(int pin_number, int pin_signal)

Function: Set the base I/O output

Returned value: None

Parameter description: Parameter 1: Pin number (**basic output pin number**), parameter 2: Status (**0-- low, 1-- high**)

6.2 getBasicInput(int pin_number)

Function: Read the base I/O output

Return value: Pin status, 0-- low, 1-- high

Parameter Description: Parameter 1: :basic Enter the pin number

7 Set the Software Joint Limit

7.1 getJointMin(int joint)

Function: Read the minimum joint Angle

Return value: Type double, the minimum Angle to which the joint can run

Parameter Description: Parameter 1: Joint number (**1-6**)

7.2 getJointMax(int joint)

Function: Read the maximum Angle of joint

Return value: Type double, the maximum Angle to which the joint can run

Parameter Description: Parameter 1: Joint number **(1-6)**

7.3 setJointMin(int joint, double angle)

Function: Set the minimum joint Angle

Returned value: None

Parameter description: Parameter 1: Joint number **(1-6)**, parameter 2: Minimum Angle (minimum Angle to which the joint can operate)

7.4 setJointMax(int joint, double angle)

Function: Set the maximum joint Angle

Returned value: None

Parameter description: Parameter 1: Joint number **(1-6)**, parameter 2: Maximum Angle (maximum Angle to which the joint can operate)

8 Drag the teaching Drag and play

8.1 DragTechSave()

Function: Start and drag the teaching recording point

Returned value: None

Parameter Description: None

8.2 DragTechExecute()

Function: Perform teaching point position

Returned value: None

Parameter Description: None

8.3 DragTechPause()

Function: Pause point recording

Returned value: None

Parameter Description: None

Note: The robot arm is in a relaxed state after the pause

9 Cartesian space coordinate parameter setting

9.1 setToolReference(double x,double y, double z, double rx, double ry, double rz)

Function: Set the tool coordinate system

Returned value: None

Parameter description: Parameter 1: x, parameter 2: y, parameter 3: z, parameter 4: rx, parameter 5: ry, parameter 6: rz(**Parameter range: x(-2800~280.00)(10xmm), y (-2800~280.00)(10xmm), z(-2800~280.00)(10xmm), rx (-3140~3140) (1000xRd), Ry (3140 ~ 3140), 1000 (XRD), rz (3140 ~ 3140), 1000 (XRD))**)

9.2 getToolReference()

Function: Get tool coordinate system

Return value: Type double *, pointing to an array containing six doubles (x,y,z,rx,ry,rz)

Parameter Description: None

9.3 setWorldReference(double x, double y, double z, double rx, double ry, double rz)

Function: Set the world coordinate system

Returned value: None

Parameter description: Parameter 1: x, parameter 2: y, parameter 3: z, parameter 4: rx, parameter 5: ry, parameter 6: rz(**Parameter range: x(-2800~280.00)(10xmm), y (-2800~280.00)(10xmm), z(-2800~280.00)(10xmm), rx (-3140~3140) (1000xRd), Ry (3140 ~ 3140), 1000 (XRD), rz (3140 ~ 3140), 1000 (XRD))**)

9.4 getWorldReference()

Function: Get the world coordinate system

Return value: Type double *, pointing to an array containing six doubles (x,y,z,rx,ry,rz)

Parameter Description: None

9.5 setMovementType(int movement)

Function: Set the movement type

Returned value: None

Parameter Description: Parameters 1:1-movel, 0-moveJ

9.6 getMovementType()

Function: Get movement type

Return value: int type, 1-movel, 0-moveJ

Parameter Description: None

9.7 setEndType(int endtype)

Function: Set the end coordinate system

Returned value: None

Parameter Description: Parameter 1:0- Flange 1- tool

9.8 getEndType()

Function: Get the end coordinate system

Returned value: int type, 0- flange 1- tool

Parameter Description: None

10 Circular motion

10.1 WriteMovC(float transpointx, float transpointy, float transpointz, float transpointrx, float transpointry, float transpointrz, float endpointx , float endpointy , float endpointz, float endpointrx, float endpointry, float endpointrz, int speed)

Function: arc trajectory movement

Returned value: None

Parameter description: transpoint is the arc path point, endpoint is the endpoint of the arc. Parameter 1: transpointx, parameter 2: transpointy, parameter 3: transpointz, parameter 4: transpointrx, parameter 5: transpointry, parameter 6: transpointrz, parameter 7: endpointx, parameter 8: endpointy, parameter 9: endpointz, parameter 10: endpointrx, parameter 11: endpointry, parameter 12: endpointrz, parameter 13: Speed

11 servo state value of joint motor

11.1 GetServoSpeeds()

Function: Get the joint running speed, running speed, unit: step /s

Return value: A pointer of type int* to an int[7] array containing the size of 1-7 joint velocities in the range +-3000

Parameter Description: None

11.2 GetServoCurrents()

Function: Obtain joint current

Return value: A pointer of type int* to an array of int[7] containing 1-7 joint currents in the range 0-5000

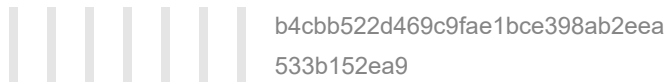
Parameter Description: None

11.3 GetServoStatus()

Function: Obtain joint status,0--> All joints are normal

Return value: A pointer of type int* to an array of int[7] holding the states of the 1-7 joint, 0--ok, 1- undervoltage, 2- overtemperature, 3- overcurrent

Parameter Description: None



Robot arm usage scenario case

This chapter presents classic robotic arm use cases to demonstrate the application of the product in representative scenarios. This includes typical applications of robotic arms in different fields, highlighting the product's versatility and applicability. Through these cases, users can gain an in-depth understanding of the flexibility and performance of the robotic arm in practical applications, providing a reference for their application in specific scenarios.

1. Painting case:

```
from pymycobot.mycobot import MyCobot
import time
import math

#Create a MyCobot instance and specify the serial port and baud rate
mc = MyCobot('COM3',115200)

#Send the target coordinate point to move the robot arm to the specified position
mc.send_coords([52.9, -64.4, 409.7, -91.23, -0.25, -89.81], 50, 0)
# Pause 2 seconds
time.sleep(2)

#Send the target coordinate point to move the robot arm to another specified position
mc.send_coords([21.5, 145.5, 233.6, -89.72, 19.19, 13.45], 50, 0)
# Pause 2 seconds
time.sleep(2)

# Send the target coordinate point cyclically to make the robot arm move in a circular
for i in range(1, 361):
    x = 21.5 + 30 * math.cos(i / 180.0 * math.pi)
    y = 145.5 + 30 * math.sin(i / 180.0 * math.pi)
    mc.send_coords([x, y, 233.6, -89.72, 19.19, 13.45], 100, 0)
    # Pause 0.7 seconds
    time.sleep(0.7)

#Send the target coordinate point to return the robot arm to the initial position
mc.send_coords([52.9, -64.4, 409.7, -91.23, -0.25, -89.81], 50, 0)
```

2. Dancing case:

```
from pymycobot.mycobot import MyCobot
import time

if __name__ == '__main__':
    #Create a MyCobot instance and specify the serial port and baud rate
    mc = MyCobot('COM3',115200)

    # Set the start time
    start = time.time()
    # Let the robotic arm reach the specified position
    mc.send_angles([-1.49, 115, -153.45, 30, -33.42, 137.9], 80)
    # Determine whether it reaches the specified location
    while not mc.is_in_position([-1.49, 115, -153.45, 30, -33.42, 137.9], 0):
        # Let the robot arm resume movement
        mc.resume()
        # Let the robot arm move for 0.5s
        time.sleep(0.5)
        # Pause the robot arm movement
        mc.pause()
        # Determine whether the move has timed out
        if time.time() - start > 3:
            break
    # Set start time
    start = time.time()
    # Let the exercise continue for 30 seconds
    while time.time() - start < 30:
        # Let the robotic arm reach the position quickly
        mc.send_angles([-1.49, 115, -153.45, 30, -33.42, 137.9], 80)
        # Set the color of the light to [0,0,50]
        mc.set_color(0, 0, 50)
        time.sleep(0.7)
        # Let the robotic arm reach the position quickly
        mc.send_angles([-1.49, 55, -153.45, 80, 33.42, 137.9], 80)
        # Set the color of the light to [0,50,0]
        mc.set_color(0, 50, 0)
        time.sleep(0.7)
```

3. Wood block transport case:

```
from pymycobot import PI_PORT, PI_BAUD
import time
def gripper_test(mc):
    Print("Start check IO part of api\n")
    # Detect whether the gripper is moving
    flag = mc.is_gripper_moving()
    Print("Is gripper moving: {}".format(flag))
    time.sleep(1)

    # Set the current position to (2048).
    # Use it when you are sure you need it.
    # Gripper has been initialized for a long time. Generally, there
    # is no need to change the method.
    # mc.set_gripper_ini()
    # Set joint point 1 and let it rotate to the position 2048
    mc.set_encoder(1, 2048)
    time.sleep(2)
    # Set six joint positions and let the robotic arm rotate to this position at a speed of 70
    mc.set_encoders([1024, 1024, 1024, 1024, 1024, 1024], 20)
    # mc.set_encoders([2048, 2900, 2048, 2048, 2048, 2048], 20)
    # mc.set_encoders([2048, 3000, 3000, 3000, 2048, 2048], 50)
    time.sleep(3)
    # Get the position information of joint point 1
    Print(mc.get_encoder(1))
    # Set the clamping jaw to rotate to the position 2048
    mc.set_encoder(7, 2048)
    time.sleep(3)
    # Set the gripper to rotate to the 1300 position
    mc.set_encoder(7, 1300)
    time.sleep(3)

    # Let the gripper reach the 2048 state at a speed of 70. 2048 will report an error, so
    mc.set_gripper_value(255, 70)
    time.sleep(3)
    # Let the gripper reach the 1500 state at a speed of 70. 1500 will report an error, so
    mc.set_gripper_value(255, 70)
    time.sleep(3)

    num=5
    while num>0:
        # Set the state of the clamping claw to quickly open the claw at a speed of 70
        mc.set_gripper_state(0, 70)
        time.sleep(3)
        #Set the state of the clamping claw and let it quickly close the claws at a speed of 70
```

```
        mc.set_gripper_state(1, 70)
    time.sleep(3)
    num-=1

    # Get the value of the gripper
    Print("")
    Print(mc.get_gripper_value())
    # mc.release_all_servos()

if __name__ == "__main__":
    #Create a MyCobot instance and specify the serial port and baud rate
    mc = MyCobot('COM3',115200)

    mc.set_encoders([2048, 2048, 2048, 2048, 2048, 2048], 20)
    time.sleep(3)
    gripper_test(mc)
```

About Us



This chapter primarily provides information about the company introduction and development history, offering users additional background on the team and robots. The "Contact Us" section furnishes detailed contact information, including email formats, instructions for describing problems, and templates for reproducing steps. This enables users to communicate effectively with the team and resolve issues. Overall, the objective of this chapter is to establish an efficient communication channel with users while presenting information about the company and team.

- [9.1 Elephant Robotics](#)
- [9.2 Contact us](#)

[← Previous Page](#)

ElephantRobot

1 Company Profile

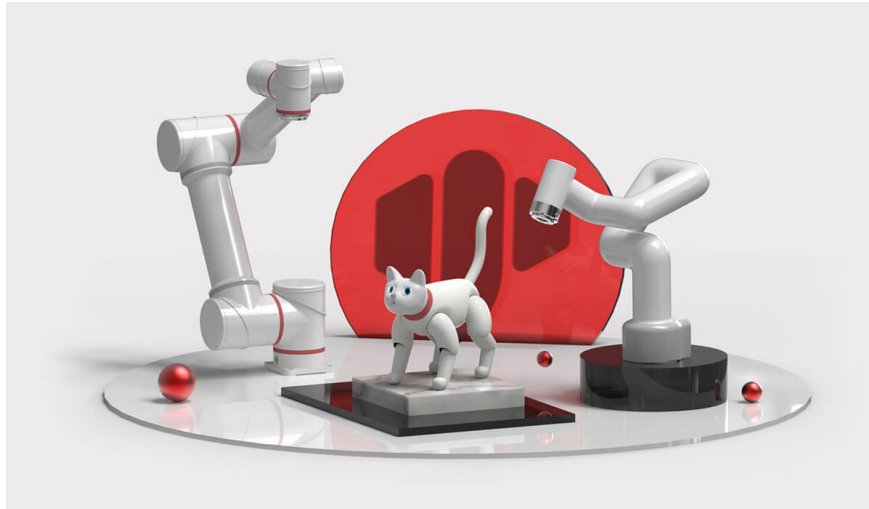


Elephant Robotics is based in Shenzhen, China, and is a high-tech enterprise focusing on robot R&D, design, and automation solutions.

The company is dedicated to providing highly flexible collaborative robots, easy-to-learn operating systems, and intelligent automation solutions for robotics education and scientific research institutions, commercial scenarios, and industrial production. Elephant Robotics has earned unanimous recognition and praise from several factories of the world's top 500 companies in countries such as South Korea, Japan, the United States, Germany, Italy, and Greece.

Adhering to the vision of "Enjoy Robots World," Elephant Robotics advocates collaborative work between humans and robots, aiming to make robots valuable assistants in work and life. The company seeks to help people liberate themselves from simple, repetitive, and monotonous tasks, leveraging the benefits of human-machine collaboration to enhance work efficiency and create a better new life.

In the future, Elephant Robotics aspires to propel the robot industry's development through cutting-edge technology and collaboratively usher in a new era of automation and intelligence.



2 Development Process

- 2016.08 ----- Elephant Robot Co., Ltd. was officially established.
- 2016.08 ----- Entered HAX incubator and obtained SOSV seed round investment.
- 2016.08 ----- Started developing Elephant S industrial collaborative robot.
- 2017.01 ----- Awarded "Top 10 Most Innovative Companies in China at CES."
- 2017.04 ----- Attended Hannover Industrial Fair and Korea Automation Exhibition.
- 2017.07 ----- The two founders were selected as "30 Business Elites Under 30" by Forbes Asia.
- 2017.10 ----- Launched the fifth generation single-arm industrial collaborative robot Elephant S.
- 2018.04 ----- Obtained angel round investment from "Cloud Angel Fund."
- 2018.06 ----- First public appearance at the 2018 Hannover World Industrial Fair.
- 2018.06 ----- Received the "Intelligent Manufacturing Entrepreneurship MBA Award" from Cheung Kong Graduate School of Business.
- 2018.06 ----- Received the "X-elerator Award" from Tsinghua SEM.
- 2018.11 ----- Won second place in the Shenzhen Division of the Asian Intelligent Hardware Competition.
- 2018.11 ----- Won the "Most Investment Enterprise Award" at the Golden Globe Award.
- 2019.03 ----- Won the "Leadership Award" at the Golden Globe Award.
- 2019.04 ----- March 2019 Catbot won the "Industrial Robot Innovation Award."
- 2019.09 ----- Attended Huawei European Ecosystem Conference (HCE) and officially became a member of Huawei's ecological partners.

2019.11 ----- Elephant Robot and Harbin Institute of Technology attended the IROS International Intelligent Robots and Systems Conference.

2019.12 ----- Elephant Robot-South China University of Technology's "Intelligent Robot Joint Development Laboratory" was officially unveiled.

2019.12 ----- Won the 2019 "Innovation Technology Award" from Gaogong.

2019.12 ----- Won the "Top Ten Fast-Growing Enterprises" of Gaogong in 2019.

2019.12 ----- Won the Shenzhen Equipment Industry-Industrial Robot Segment-"New Enterprise Award."

2019.12 ----- Launched the world's first bionic robotic cat, MarsCat.

2020.05 ----- The founder won the 2019 Shenzhen Robot Emerging Figure Award.

2020.10 ----- Launched myCobot, the world's lightest and smallest six-axis collaborative robot.

2021.03 ----- Launched myCobotPro 320, the smallest collaborative robot for scientific research.

2021.05 ----- MarsCat received competing reports from Xinhua Finance, China Daily, Nanjing Daily, Harbin Daily, and other media.

2021.07 ----- Released the smallest composite robot chassis – the little elephant mobile robot myAGV.

2021.09 ----- Launched the world's first fully wrapped four-axis robotic arm - the little elephant palletizing robotic arm myPalletizer.

2022.01 ----- Obtained a series of reports from 36 Chlorine and Geek Park on the role of Elephant Robot in the light consumer robot industry.

2022.02 ----- MarsCat and myCobot appeared in the Spring Festival Gala live video broadcast and participated in Shenzhen Satellite TV's special New Year program.

2022.05 ----- Launched the most compact small six-axis robotic arm mechArm, capable of artificial intelligence robot education.

2022.06 ----- Combined with Unity engine, based on myCobot robot, launched artificial intelligence robot practical introduction book + books (international courses).

2022.07 ----- Released metaCat, a simulation companion robot cat in the artificial intelligence era.

2022.07 ----- Released mybuddy, the smallest dual-arm collaborative robot in history.

2022.08 ----- Won the "Top Ten Non-Industrial Technology Innovation Awards."

2022.08 ----- The founder won the "2022 Shenzhen Robot Emerging Figure Award."

2022.11 ----- First runner-up in iFLYTEK AI Developer Competition real-time engagement (real-time interaction) track.

2022.11 ----- Best Robot Award at 2022 World Acoustic Expo 1024 Science and Technology Expo.

2022.12 ----- CCTV report.

3 Related Links

- Official website: <https://www.elephantrobotics.com>
- Purchase links:
 - Taobao: <https://shop504055678.taobao.com>
 - Shopify: <https://shop.elephantrobotics.com/>
- Video:
 - Bilibili: [Elephant Robot's personal space-Elephant Robot's personal homepage-Bilibili Video](#)
 - YouTube: [Elephant Robotics - YouTube](#)
- Facebook: <https://www.facebook.com/mycobotcreator/>
- LinkedIn: <https://www.linkedin.com/company/18319865>
- Twitter: <https://twitter.com/CobotMy>
- Discord: <https://discord.gg/2MAherp7nt>
- Hackster: <https://www.hackster.io/elephant-robotics>

[← Previous Page](#) | [Next Page →](#)

Contact Us

Our working hours are on Chinese working days, from 10 AM to 6 PM
Beijing time.

If you have any other problems, contact us via the ways below.

Email :

If you have purchase intention or any parameter questions, please send an email
to this mailbox.

E-mail :

sales@elephantrobotics.com

If the listed problems can't help you solve and you have more after-sales
questions, please send an email to this mailbox.

E-mail :

support@elephantrobotics.com

We will give a reply within 1-2 business days;

WeChat: We provide one-to-one service only for those users who have
purchased
myCobot via WeChat.



[← Previous Page](#)