Table of Contents

Introduction 1.1

1 Product Introduction 1.2

- 1.1 Design Philosophy 1.2.1
- 1.2 Application Scenario 1.2.2

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

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 SDK Development 2.1

- 5.1 Python 2.1.1
  - 1 Environment Building 2.1.1.1
  - 2 Introduction to API 2.1.1.2
  - 3 python demo 2.1.1.3
  - 3 Drag to teach 2.1.1.4
  - 4 Use Cases 2.1.1.5
- 5.2 Robot Operating System 1 (ROS1) 2.1.2
  - 1 Environment Building 2.1.2.1
  - 2 ROS basics 2.1.2.2
  - 3 Rviz use 2.1.2.3
  - 4 Basic function case 2.1.2.4
- 5.3 Robot Operating System 2 (ROS2) 2.1.3
  - 1 Environment Building 2.1.3.1
  - 2 ROS2 basics 2.1.3.2
  - 3 Rviz2 use 2.1.3.3
  - 4 Basic function case 2.1.3.4

6 Examples of Robots Using 2.2
7 About Us 3.1
7.1 Elephant Robotics 3.1.1
7.2 Contact us 3.1.2

Mercury_X1_docs

1.1 Design Concept

Mercury X1 is a wheeled dual-arm humanoid robot with 19 degrees of freedom for embodied intelligence applications. It is composed of Mercury B1 dual-arm robot and a high-performance mobile base. The whole machine is equipped with a new self-developed Liyuan series harmonic module. The new modular design module comes standard with electromagnetic brakes and hollow wiring. It has the characteristics of high precision, high inertia and low weight. It is also equipped with NVIDIA Jetson Xavier main control, which works together with four independent secondary control units. The mobile base is equipped with high-performance lidar, ultrasonic sensors and 2D vision and other rich perceptions; it is driven by a direct drive motor with a maximum operating speed of 1.2m/s; the maximum climbing height is 2CM; the maximum climbing angle is 15 degrees. The maximum battery life of the whole machine is up to 8 hours, meeting the needs of personal and commercial applications.

1.2 Application scenarios

1.2.1 Scientific research scenario

One-stop scientific research on embodied intelligent humanoid robot - Mercury B1.

Thanks to the application of self-developed control algorithms and self-developed modules, X1 can be used for research and applications in multiple directions such as dual-arm collaborative control, dual-arm motion planning, and humanoid embodied intelligence applications. It is the first choice model for humanoid robot application research. .

1.2.2 Education scene

Robot education and teaching set, which can be used with 2D or 3D vision modules, can realize image recognition, model training, robot control, motion planning, robot space calibration, vision and robot hand-eye calibration in many disciplines and different fields in humanoid embodied intelligent education. Knowledge learning.

1.2.3 Service scenario

The powerful performance and internal performance of the Mercury series, coupled with the elegant appearance design, make the Mercury robot not only used in education and scientific research scenarios, but also in commercial service scenarios. Through different end accessories, it can be matched with various daily furniture and home appliances. The robot can serve actual company displays, business exhibitions, industry exchanges and other application environments, showing the cool applications of robot waiters.

1.2.4 Entertainment scene

Paired with a customized end effector to simulate human movement development, the Mercury robot can realize human-like movement applications and carry out creative development of personal applications.

1.2.5 VR/Aloha remote operation scene

Paired with VR/Aloha and other equipment to realize remote control robot applications

Product Specifications

1 Machine Parameters

Metrics	Parameter
Name	Mercury X1 wheeled dual-arm humanoid robot
Model	Mercury X1
Height of the whole machine	1.2m
Degrees of freedom	19
Maximum battery life	8h
Maximum load of the robotic arm	1kg
Repeatability of the robotic arm	±0.05mm
Net weight of the whole machine	20kg
Operating voltage	DC24V/9.2A
Chassis drive motor	High-performance direct-drive motor
Maximum operating speed	1.2m/s
Maximum Climb Angle	15°
Storage space	15L
Master	6-Core Arm v8.2 64-bit CPU，384-Core Volta™ GPU
Master Computing Power	21 TOPS
Chassis CPU	4-Core ARM Cortex-A57 CPU
Chassis GPU	128 NVIDIA CUDA® cores
screen	9-inch touch screen
Mobile Site Sensor	LiDAR, Ultrasonic Radar, 2D Vision
3D Camera	Obi Nakako Deeyea
Microphone	Linear 4 microphone, 5 meters 180° pickup
IO	24V 6 Input, 6 Output2A
Communication Methods	CAN bus/WIFI/network port/Bluetooth/USB/serial port

2 Software Basic Function Support

Function/Development Environment	Usage
Free Movement	support
Joint Movement	support
Cartesian Movement	support
Track 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 Port Control Protocol	support
TCP/IP	support
MODBUS	support

← 上一页| 下一页 →

Control core parameters

1 Main controller 1 specification data sheet

Metrics	Parameter
Master	Jetson Xavier
Master Model	Jetson Xavier NX
CPU	6-core NVIDIA Carmel ARM®v8.2 64-bit CPU 6MB L2 + 4MB L3
GPU	384-core NVIDIA Volta™ GPU with 48 Tensor cores
AI Performance	21 TOPS
Storage	16 GB eMMC 5.1
CSI Camera	2 CSI cameras
Network	10/100/1000 BASE-T Ethernet
USB port	1 x USB 3.2 2.0 (10 Gbps) 2 x USB 2.0 port
Other I/O	2 UART serial ports

2 Master Controller 2 Specification Parameter Table

Metrics	Parameter
Master	Left Arm Master
Master Model	ESP32
Core parameter	240MHz dual core. 600 DMIPS, 520KB SRAM. Wi-Fi, dual mode Bluetooth
Auxiliary control Flash	4MB
LED display	5X5 RGB

3 Main controller 3 specification parameter table

Metrics	Parameter
Master	Right arm master
Master Model	ESP32
Core parameter	240MHz dual core. 600 DMIPS, 520KB SRAM. Wi-Fi, dual mode Bluetooth
Auxiliary control Flash	4MB
LED display	5X5 RGB

4 Auxiliary controller 1 specification data sheet

Metrics	Parameter
Auxiliary control	Jetson Nano
Auxiliary control model	Jetson Nano 4G
AI Performance	472 GFLOPS
GPU	128-core NVIDIA Maxwell™ architecture GPU
GPU Max Frequency	921MHz
CPU	Quad-core ARM® Cortex®-A57 MPCore processor
CPU Max Frequency	1.43GHz
Memory	4GB 64-bit LPDDR4

25.6GB/s| CSI Camera | 1 CSI Camera | | USB* | 1x USB3.0(5 Gbps)
2x USB 2.0 | | Mechanical |69.6mmx45mm
260-pin SO-DIMM connector |

← 上一页| 下一页 →

Structural Dimension Parameters

❗ > This chapter is in millimeters of distance and degrees of angle.

1 Product Dimensions & Workspace

When choosing a robot mounting location, it is important to consider the cylindrical space directly above and below the robot, and avoid moving the tool towards the cylindrical space as much as possible. Because this will cause the joint to rotate too fast when the tool movement is slow, resulting in low robot efficiency and difficult risk assessment.

2 Arm end flange dimensions

End size

← 上一页| 下一页 →

Electrical Characteristic Parameters

1 Overview of the dock interface

Figure 1 Front view of the base

Figure 2 Left view of the base

Figure 3 Right view of the base

Figure 4 Interior view of the base

1.1 Dock Interface Description

Numbered	Interface	Define	Features	Note
1	Power input interface	DC24V input	DC24V input	
2	Switch	Power switch	Control the input power on and off	With light (light on)
3	E-stop interface	STOP	Emergency stop loop interface	
4	Ethernet port			
5	USB3.0	USB3.0*2	It can be connected to an external device or a USB flash drive	

1 Power input interface: This interface is connected to the DC24V power adapter interface

2 Power Switch: Control the main power input on and off, when off, the controller is also powered off

3 Emergency stop circuit terminal: connected with the emergency stop button box, it can be used to control the emergency stop of the robot

> **Note**: The emergency stop switch must be connected to the robot in use, and ensure that the emergency stop switch circuit is connected.

4 Ethernet

5 USB3.0 interface: the interface for data connection with serial bus standard 3.0, users can use the USB interface to copy program files, and can also use the USB interface to connect mouse, keyboard and other peripherals

2 Overview of End Interfaces

Figure 5 Diagram of the end of the left arm

Figure 5 Diagram of the end of the right arm

2.1 End Interface Description

Numbered	Interface	Define	Features	Note
6	4pin USB terminal	External interface	Connect the camera	
7	M8 Aviation Socket	End Tool IO Interface	Interact with external devices	

1 As shown in the diagram is the M8 aviation socket I/O diagram, the Mercury X1 robot provides one input and two outputs.

The definition of each tool I/O port is shown in the following table, it should be noted that the tool I/O is PNP type in both input and output, and the wiring method is the same as that of the bottom output interface.

Numbered	Signal	Explanation	M8 line 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	Powder
7	IN2	With input interface 2	Blue
8	485B	Reserved, undeveloped	Purple

2.1.3 USB terminal: used to connect the camera

← 上一页| 下一页 →

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

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

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

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.

4 Disclaimer

Please read and understand the following disclaimer carefully before using the product:

- **Safe Use:** This product is designed for specific application scenarios. Ensure that all safety guidelines and operating manuals are followed during use. Users should be properly trained in the use of the product and understand and comply with all relevant safety regulations.
- **LIMITATION OF LIABILITY:** THE MANUFACTURER SHALL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM THE USE OR MISUSE OF, OR ANY MATTER RELATING TO THE PRODUCTS. This disclaimer does not cover or exclude liability whose exclusion is not permitted by law.
- **Technical Support:** Please read the product documentation carefully during installation and use, and seek technical support from the manufacturer if necessary. For technical support issues, please refer to the official documentation provided by the manufacturer or contact the relevant support channel.
- **Software Updates:** The manufacturer may provide updates to product firmware or software. Users should regularly check and apply these updates to ensure product performance and security.
- **Periodic Maintenance:** Users should inspect and maintain the product following the regular maintenance guidelines provided by the manufacturer. Regular maintenance and inspections help ensure long-term product performance.
- **Customization and Modifications:** Products may not be customized, modified, or altered without the express permission of the manufacturer. Any unauthorized modifications may void the product's warranty and may have unpredictable effects on safety and performance.
- **Legal Compliance:** Users should ensure that their use complies with all applicable laws and regulations. In some areas, the use of the product may be restricted by specific regulations.

By using the Mercury A1 seven-axis collaborative robot, you agree to and accept these disclaimers. The manufacturer reserves the right to change product specifications, features, and disclaimers without prior notice.

← Previous Section | Next Chapter →

Transport and preservation

1 Logistics Transportation Requirements

Temperature	0 ° C ~ 50 ° C
Relative humidity	20% ~ 70%
Direction during transportation	The robot's head faces up, and his arms are down
External conditions during transportation	Fixed wooden frames outside to prevent squeezing
Wooden frame size	57 *57*120

2 Equipment Storage

Temperature	0 ° C ~ 50 ° C
Relative humidity	20% ~ 70%
Direction during transportation	The robot's head faces up, and his arms are down
Overlapping requirements	not overlapping
Storage environment	indoor
Other environmental requirements	-Fain away from dust, oil fume, salt, iron dumbs, etc. -Stark from flammable, corrosive liquid and gas. -Don’t contact water. -It without passing impact and vibration. -For away from strong electromagnetic interference sources.

← Previous | Next Chapter →

Care and maintenance

as a robot manufacturer, we attach importance to ensuring that customers can maintain and safely maintain and upgrade their robotic equipment.To this end, we provide the following detailed maintenance and maintenance guidelines, including common maintenance items and maintenance or upgrade hardware. Please read it carefully.

1 Common maintenance items and recommendation cycles

Maintenance Project	Description	Recommended cycle
Visual inspection	Check whether the robot has obvious damage, foreign body accumulation or wear	Daily
Structural cleaning	Use clean and dry fabrics to clean the robot structure components to avoid moisture and erosion cleaner	Daily
Fastener components check	Check and fasten all bolts and connectors	Daily
Lubrication	Lubrication of joints and mobile components, use the lubricant recommended by the manufacturer	every 3 months
Cable and wiring check	Check the cable and wiring to ensure no damage or wear	month
Electrical connection check	Make sure that all electrical connections are firm, non - corrosive or damaged	Monthly
Software Update	Check and update the control software and applications	When there is update each time
Software Data Backup	Regular backup key software configuration and data	Season
Firmware update	regular inspection and update firmware to obtain the latest functions and security patch	each time it is updated
Sensor and device check	Check the sensor and other key devices to ensure normal work	monthly
Emergency stop function test	Regular test emergency stop function to ensure its reliability	Month
Environmental conditions monitoring	Monitoring working environment temperature, humidity, dust, etc. to ensure the operating specifications of the robot.	Continuous monitoring

Maintenance Project	Description	Recommended cycle
Safety configuration review	regular inspection and confirmation of the safety configuration of the robot, such as speed limit and working scope settings	Month
Practice of Preventive Maintenance Plan	Perform regular inspections and maintenance according to the manufacturer's maintenance plan	According to the manufacturer guide

2 Guide to change the robot hardware independently

We understand that customers may have the need to upgrade or repair robotic hardware by themselves.Before performing any upgrade operation, please read the relevant parameters of the product in detail and confirm whether it is allowed to perform such operations with our official personnel.The product failure without official permission may lead to product failure and not within the scope of warranty.

Material requirements

official manufacturing or recommended materials: All the accessories and components required for maintenance and upgrades must be made or clearly recommended by our official.This includes but not limited to electronic components, sensors, motors, connectors and any other replaceable components.

Material acquisition: Customers can purchase the maintenance and upgrade materials required for purchasing through our official channels.This ensures the quality and compatibility of the accessories.

repair or upgrade process

Customer maintenance: Customers should be responsible for completing maintenance work.We will provide detailed maintenance guidance and manuals to guide customers to complete the maintenance steps.

follow official guidance: maintenance operations should strictly follow the official guidance we provide.Any operation from official guidance may cause equipment damage.

liability and warranty policy

Division of responsibilities:

manufacturer: provide official guidance, official manufacturing or recommended materials for maintenance and upgrading, and deal with problems caused by manufacturing defects.

Customer: Responsible for completing maintenance in accordance with official guidance and using official accessories.

Warranty Policy:

Maintenance is effective: Only when the maintenance operation follows our guidance and uses official accessories, the warranty is valid.

warranty is invalid: If the customer fails to operate in accordance with the official guidance, or uses unofficial accessories for maintenance or upgrade, any damage will not be within the scope of warranty.

Precautions

Safety First: Before performing any maintenance or upgrade operation, make sure to follow all security guides, including power off and appropriate protection equipment.

technical support: If you encounter problems during maintenance, it is recommended to stop operation and contact our technical support team to seek help.

We strongly recommend that customers follow these guidelines strictly to ensure the safety and effective operation of robotic equipment.Improper maintenance operations can cause device damage and affect the warranty state.For further guidance or support, please contact our professional and technical team in time.

← 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"

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"

← Previous Section | Next Chapter →

First installation and use

1

- **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 unboxing guide

1 Product Unboxing Graphic Guide

Why you need to follow these steps to move your 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 guidelines carefully to ensure your product is safe during the unboxing process.

- **1** Check the box for damage. If there is any damage, please contact the logistics company and supplier in your area in time.
- **2** Open the packaging box and take out the instructions for use, sponge packaging cover, mercury_x1 robot, supporting power supply, and delivery list.
- **3** Please make sure each step is completed before proceeding to the next step 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.

←Previous page |Next page→

Start detection guide

External cable connections

Please read the **Chapter Safety Instructions** carefully before operation to ensure safe operation. At the same time, connect the power adapter to the base.

Power status display

Confirm that the power adapter is connected, press the power switch **start button (round)**, and the **LED light on the base will flash**.

Basic function detection

Please refer to the 5.1-4 Robot Information chapter for guidance during operation. Before proceeding, be sure to follow the electrical connection instructions above and verify that the device is securely installed. Failure to properly connect cables or secure equipment may result in an accident. thank you for your cooperation.

←Previous page |Next chapter→

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`.

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:

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` :

7.1.1-7

- 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: 7.1.1-7
- Right click on the selection that the red arrow points, and create a new python file.

7.1.1-7

- Type name for the new file.

7.1.1-7

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 defa

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

```python
from pymycobot import Mercury

ml = Mercury('/dev/left_arm')
mr = Mercury('/dev/right_arm')

ml.power_on()
mr.power_on()

print(ml.get_angles())
print(mr.get_angles())
```

← System introduction | Python API →

6.1 Python API

[toc]

6.1.1 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

ml = Mercury('/dev/left_arm')
mr = Mercury('/dev/right_arm')

ml.power_on()
mr.power_on()

print(ml.get_angles())
print(mr.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`) Arm joint range of motion | Joint Id | range | | ---- | ---- | | 1 | -178 ~ 178 | | 2 | -74 ~ 130 | | 3 | -178 ~ 178 | | 4 | -180 ~ 10 | | 5 | -178 ~ 178 | | 6 | -20 ~ 273 | | 7 | -180 ~ 180 |

    Body joint range of motion. Joint 11 is the chin camera. Joint 12 is the neck. Joint 13 is the lumbar joint | Joint Id | range | | ---- | ---- | | 11 | -60 ~ 0 | | 12 | -138 ~ 188 | | 13 | -118 ~ 118 |

  - `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 : faile; 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
  - 1: clicked

`set_color(r, g, b)`

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

11. Drag Teaching

`drag_teach_save()`

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

`drag_teach_pause()`

- **function:** Pause sampling

`drag_teach_execute()`

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

`drag_teach_clean()`

- **function:** clear sample.

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:** `oords` : (`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 - movel, 0 - moveJ.

`get_movement_type()`

- **function:** Get movement type.
- **Return value:**
  - `1` - movel
  - `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

6.1.2 Base control interface

To call the Python interface below, you need to first start the ROS server.

```
roslaunch turn_on_mercury_robot navigation.launch
```

Next, open another terminal and enter the following command.

```
cd ~/mercury_x1_ros/src/turn_on_mercury_robot/scripts/
python cobotx_socket_server_json.py
```

`go_straight(speed=0.25, exercise_duration=5)`

- **function:** Control the forward movement of the base
- **Parameters**:
  - `speed` : Movement speed. Unit: meters/second
  - `exercise_duration` : Exercise duration。

`go_back(speed=0.25, exercise_duration=5)`

- **function:** Control the base to move backwards
- **Parameters**:
  - `speed` : Movement speed. Unit: meters/second

- `exercise_duration` : Exercise duration。

`turn_left(speed=0.25, exercise_duration=5)`

- **function:** Control the left turn movement of the base
- **Parameters**:
  - `speed` : Movement speed. Unit: meters/second
  - `exercise_duration` : Exercise duration。

`turn_right(speed=0.25, exercise_duration=5)`

- **function:** Control the right turn movement of the base
- **Parameters**:
  - `speed` : Movement speed. Unit: meters/second
  - `exercise_duration` : Exercise duration。

`stop()`

- **function:** Control the base to stop the current movement.

`init_position(position_x, position_y, orientation_z, orientation_w, covariance)`

- **function:** Set navigation starting position.
- *Parameters**:
  - `position_x` :
  - `position_y` :
  - `orientation_z` :
  - `orientation_w` :
  - `covariance` :

`goto_position(position_x, position_y, orientation_z, orientation_w, covariance)`

- **function:** Navigating to the target point.
- *Parameters**:
  - `position_x` :
  - `position_y` :
  - `orientation_z` :
  - `orientation_w` :
  - `covariance` :

`cancel_navigation()`

- **function:** Cancel navigation.

`get_software_version()`

- **function:** Get the base server version number.

`get_base_ros_version()`

- **function:** Get the base ROS project version number.

`get_battery_state()`

- **function:** Get battery level.

Case 1

Set the color of the end lights to blue

```
from pymycobot import Mercury

ml = Mercury("/dev/ttyTHS0")
mr = Mercury("/dev/ttyACM0")
# Robot powered on
ml.power_on()
mr.power_on()

ml.set_color(0,0,255)
mr.set_color(0,0,255)
```

Case 2

Angle control

```
from pymycobot import Mercury
import time
ml = Mercury("/dev/ttyTHS0")
mr = Mercury("/dev/ttyACM0")
# Robot powered on
ml.power_on()
mr.power_on()
# Single angle control
ml.send_angle(1, 90, 40)
mr.send_angle(1, 90, 40)
time.sleep(3)
ml.send_angle(1, 0, 40)
mr.send_angle(1, 0, 40)
time.sleep(3)

# All angle controls
ml.send_angles([0, 0, 90, 0, 0, 90, 0], 40)
mr.send_angles([0, 0, 90, 0, 0, 90, 0], 40)
time.sleep(3)
ml.send_angles([0, 0, 0, 0, 0, 90, 0], 40)
mr.send_angles([0, 0, 0, 0, 0, 90, 0], 40)
```

Drag Teaching

drag_teach_save()

Start recording and dragging teaching points.

- Note: In order to display the best sports effect, the recording time should not exceed 90 seconds

drag_teach_pause()

Pause sampling.Pause sampling and enable the robotic arm again

drag_teach_execute()

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

Case

```
from pymycobot import Mercury
import time
ml = Mercury("/dev/ttyTHS0")
mr = Mercury("/dev/ttyACM0")

ml.power_on()
mr.power_on()
# Left arm begins trajectory recording
ml.drag_teach_save()
# Recording duration is 10 seconds
time.sleep(10)
# Stop recording
ml.drag_teach_pause()
time.sleep(1)
# Start executing the recorded motion trajectory, only once
ml.drag_teach_execute()
```

Case 1

Set the color of the end lights to blue

```
from pymycobot import Mercury

ml = Mercury("/dev/ttyTHS0")
mr = Mercury("/dev/ttyACM0")
# Robot powered on
ml.power_on()
mr.power_on()

ml.set_color(0,0,255)
mr.set_color(0,0,255)
```

Case 2

Angle control

```
from pymycobot import Mercury
import time
ml = Mercury("/dev/ttyTHS0")
mr = Mercury("/dev/ttyACM0")
# Robot powered on
ml.power_on()
mr.power_on()
# Single angle control
ml.send_angle(1, 90, 40)
mr.send_angle(1, 90, 40)
time.sleep(3)
ml.send_angle(1, 0, 40)
mr.send_angle(1, 0, 40)
time.sleep(3)

# All angle controls
ml.send_angles([0, 0, 90, 0, 0, 90, 0], 40)
mr.send_angles([0, 0, 90, 0, 0, 90, 0], 40)
time.sleep(3)
ml.send_angles([0, 0, 0, 0, 0, 90, 0], 40)
mr.send_angles([0, 0, 0, 0, 0, 90, 0], 40)
```

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 transfered 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 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.

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

← Previous Section | Next Page →

Linux system environment

The system comes with Ubuntu (V-20.04) system when leaving the factory, with a built-in development environment. There is no need to set up and manage it. Just update the `mercury_x1_ros` package.

`mercury_x1_ros` is a ROS1 package launched by Elephant Robot for its Mercury X1 series robotic arms.

ROS1 Project address: http://github.com/elephantrobotics/mercury_x1_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 packages in a timely manner, you can enter the `/home/er/` folder through the file manager, open the ROS1 environment terminal, and then run the update command:

```
# Clone the code on github
git clone https://github.com/elephantrobotics/mercury_x1_ros.git # Please check the at
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_x1_ros` folder already exists in the `/home/er` directory, you need to delete the original `mercury_x1_ros` first and then execute the above command. Among them, `er` in the directory path is the user name of the system. If there are any inconsistencies, please modify them.

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 rosout (log management) and parameter server (parameter manager)

3 nodes

ROS processes and instances of running executable files in pkg.

```
$rosrun [pkg_name] [node_name] #Start
$rosnode list #List currently running node information
$rosnode info [node_name] #Display detailed information of a node
$rosnode 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 toservice 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 rosmsg

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

rosmsg demo:

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

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

```
//rosmsg package  # Package names
rosmsg package turtlesim
```

- rosmsg show Show message description

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

- rosmsg info Works the same as rosmsg show
- rosmsg 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 rosmsg.

```
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 soildwork 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.
    - `<namel>` (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, zx, 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: `value = multiplier * other_joint_value + 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

```
caktin_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.

```
rosrun 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 rosnode

View running ROS node information.

```
rosnode list
rosnode 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 rosmsg

View ROS message types.

```
rosmsg 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 Introduction to rviz interface

Open a new ROS1 terminal (shortcut key: `Ctrl+Alt+T`) and enter the following command:

```
roscore
```

Then open a new ROS1 terminal (shortcut key: `Ctrl+Alt+T`) and input the following command to open rviz.

```
rosrun 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.

If you want to know more information about rviz, go to Official documents.

← Previous Page | Next Page →

Mercury X1 Controls

Here we mainly introduce how to control the movement of Mercury X1 through a series of related instructions.

1. Chassis underlying communication

First, start the underlying communication and map construction program of the chassis. Open the ROS1 environment terminal and run the command:

```
roslaunch turn_on_mercury_robot mapping.launch
```

Output the following information:

```
... logging to /home/er/.ros/log/34bcf3be-0606-11ef-8293-e8fb1c355a09/roslaunch-er-des
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://er-desktop:43117/

SUMMARY
========

PARAMETERS
 * /LD14/disable_max: [180]
 * /LD14/disable_min: [120]
 * /LD14/flag_parted: True
 * /if_akm_yes_or_no: no
 * /lsm10_v2/baud_rate: 460800
 * /lsm10_v2/disable_max: [260]
 * /lsm10_v2/disable_min: [100]
 * /lsm10_v2/frame_id: laser
 * /lsm10_v2/max_distance: 30.0
 * /lsm10_v2/min_distance: 0.0
 * /lsm10_v2/scan_topic: scan
 * /lsm10_v2/serial_port: /dev/wheeltec_lidar
 * /lsm10_v2/truncated_mode: 1
 * /lsn10/disable_max: [10, 40, 70, 180]
 * /lsn10/disable_min: [0, 30, 60, 90]
 * /lsn10/max_distance: 30.0
 * /lsn10/min_distance: 0.0
 * /lsn10/truncated_mode: 0
 * /mercury_robot/gyro_frame_id: gyro_link
 * /mercury_robot/odom_frame_id: odom
 * /mercury_robot/robot_frame_id: base_footprint
 * /mercury_robot/serial_baud_rate: 115200
 * /mercury_robot/usart_port_name: /dev/wheeltec_con...
 * /robot_description: <?xml version="1....
 * /robot_pose_ekf/base_footprint_frame: base_footprint
 * /robot_pose_ekf/freq: 30.0
 * /robot_pose_ekf/imu_used: True
 * /robot_pose_ekf/odom_data: odom
 * /robot_pose_ekf/odom_used: True
 * /robot_pose_ekf/output_frame: odom
 * /robot_pose_ekf/sensor_timeout: 2.0
 * /robot_pose_ekf/vo_used: False
 * /rosdistro: noetic
 * /rosversion: 1.16.0
 * /rplidarNode/angle1_end: 50.0
 * /rplidarNode/angle1_start: 40.0
 * /rplidarNode/angle2_end: 140.0
```

```
 * /rplidarNode/angle2_start: 130.0
 * /rplidarNode/angle3_end: 230.0
 * /rplidarNode/angle3_start: 220.0
 * /rplidarNode/angle4_end: 320.0
 * /rplidarNode/angle4_start: 310.0
 * /rplidarNode/angle_end: 360.0
 * /rplidarNode/angle_start: 0.0
 * /rplidarNode/distance_max: 30.0
 * /rplidarNode/distance_min: 0.0
 * /rplidarNode/is_parted: False
 * /slam_gmapping/angularUpdate: 0.0436
 * /slam_gmapping/astep: 0.05
 * /slam_gmapping/base_frame: base_footprint
 * /slam_gmapping/delta: 0.05
 * /slam_gmapping/iterations: 5
 * /slam_gmapping/kernelSize: 3
 * /slam_gmapping/lasamplerange: 0.005
 * /slam_gmapping/lasamplestep: 0.005
 * /slam_gmapping/linearUpdate: 0.05
 * /slam_gmapping/llsamplerange: 0.01
 * /slam_gmapping/llsamplestep: 0.01
 * /slam_gmapping/lsigma: 0.075
 * /slam_gmapping/lskip: 0
 * /slam_gmapping/lstep: 0.05
 * /slam_gmapping/map_update_interval: 0.01
 * /slam_gmapping/maxRange: 5.0
 * /slam_gmapping/maxUrange: 4.0
 * /slam_gmapping/minimumScore: 30
 * /slam_gmapping/odom_frame: odom
 * /slam_gmapping/ogain: 3.0
 * /slam_gmapping/particles: 8
 * /slam_gmapping/resampleThreshold: 0.5
 * /slam_gmapping/sigma: 0.05
 * /slam_gmapping/srr: 0.01
 * /slam_gmapping/srt: 0.02
 * /slam_gmapping/str: 0.01
 * /slam_gmapping/stt: 0.02
 * /slam_gmapping/temporalUpdate: -1.0
 * /slam_gmapping/xmax: 5.0
 * /slam_gmapping/xmin: -5.0
 * /slam_gmapping/ymax: 4.0
 * /slam_gmapping/ymin: -4.0

NODES
  /
    base_to_camera (tf/static_transform_publisher)
    base_to_gyro (tf/static_transform_publisher)
    base_to_laser (tf/static_transform_publisher)
```

```
    base_to_link (tf/static_transform_publisher)
    joint_state_publisher (joint_state_publisher/joint_state_publisher)
    lsm10_v2 (lsm10_v2/lsm10_v2)
    mercury_robot (turn_on_mercury_robot/mercury_robot_node)
    robot_pose_ekf (robot_pose_ekf/robot_pose_ekf)
    robot_state_publisher (robot_state_publisher/robot_state_publisher)
    save_map (world_canvas_msgs/save)
    slam_gmapping (gmapping/slam_gmapping)

auto-starting new master
process[master]: started with pid [7403]
ROS_MASTER_URI=http://localhost:11311

setting /run_id to 34bcf3be-0606-11ef-8293-e8fb1c355a09
process[rosout-1]: started with pid [7414]
started core service [/rosout]
process[lsm10_v2-2]: started with pid [7421]
process[save_map-3]: started with pid [7422]
process[slam_gmapping-4]: started with pid [7427]
port = /dev/wheeltec_lidar, baud_rate = 460800
open_port /dev/wheeltec_lidar  OK !
process[mercury_robot-5]: started with pid [7434]
process[base_to_link-6]: started with pid [7437]
process[base_to_laser-7]: started with pid [7447]
[ INFO] [1714380923.257953390]: Data ready
[ INFO] [1714380923.271190031]: tringai_robot serial port opened
process[base_to_camera-8]: started with pid [7455]
process[base_to_gyro-9]: started with pid [7461]
process[joint_state_publisher-10]: started with pid [7464]
process[robot_state_publisher-11]: started with pid [7476]
process[robot_pose_ekf-12]: started with pid [7488]
[ INFO] [1714380923.815804921]: output frame: odom
[ INFO] [1714380923.822399818]: base frame: base_footprint
[ INFO] [1714380924.077547181]: Initializing Odom sensor
[ INFO] [1714380924.582680406]: Odom sensor activated
[ INFO] [1714380925.085858055]: Initializing Imu sensor
[ INFO] [1714380925.087556508]: Kalman filter initialized with odom measurement
[ INFO] [1714380925.091287721]: Imu sensor activated
[ INFO] [1714380925.206262818]: Laser is mounted upwards.
 -maxUrange 4 -maxUrange 5 -sigma     0.05 -kernelSize 3 -lstep 0.05 -lobsGain 3 -aste
 -srr 0.01 -srt 0.02 -str 0.01 -stt 0.02
 -linearUpdate 0.05 -angularUpdate 0.0436 -resampleThreshold 0.5
 -xmin -5 -xmax 5 -ymin -4 -ymax 4 -delta 0.05 -particles 8
[ INFO] [1714380925.222595401]: Initialization complete
update frame 0
update ld=0 ad=0
Laser Pose= 0.0368556 0.000564671 3.14
m_count 0
```

```
Registering First Scan
update frame 4089
update ld=0.0500112 ad=4.53872e-06
Laser Pose= 0.0383474 0.001601 3.14
```

2. Load URDF model

After the underlying communication program is started, the Mercury X1 URDF model is loaded through the launch file, a ROS1 environment terminal is opened, and then the command is run.

```
roslaunch turn_on_mercury_robot slider_control.launch
```

The effect diagram is as follows:

You can then control the movement of the joint model in rviz by dragging the slider.

3.Joint control

After the URDF model is successfully loaded, if you want the real Mercury X1 to move along with it, you need to open another ROS1 environment terminal and run the command:

Then run the command:

```
rosrun turn_on_mercury_robot slider_control.py
```

Please note: Since the robot arm will move to the current position of the model when the command is entered, please make sure that the model in rviz does not have mold penetration before you use the command Do not quickly drag the slider after connecting the robotic arm to prevent damage to the robotic arm

4 Chassis Control

After the joint control program is started, if you want the chassis car in the real Mercury X1 to run together, you need to start the keyboard control program of the chassis car, open a ROS1 environment terminal, and then run the command:

```
rosrun turn_on_mercury_robot mercury_keyboard.py
```

← 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 →

Linux system environment

The system 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_x1_ros2` package.

`mercury_x1_ros2` is a ROS2 package launched by Elephant Robot for its Mercury X1 series robotic arms.

ROS2 project address: http://github.com/elephantrobotics/mercury_x1_ros2

Robotic arm API driver library address:
https://github.com/elephantrobotics/pymycobot

1 Update mercury_x1_ros2 package

In order to ensure that users can use the latest official packages in a timely manner, you can enter the `/home/er` folder through the file manager, open the ROS2 environment terminal, and then run the command update:

```
# Clone the code on github
git clone https://github.com/elephantrobotics/mercury_x1_ros2.git # Please check the a
cd ~/mercury_x1_ros2      # Back to work area
colcon build --symlink-install # Build the code in the workspace, --symlink-install: A
source install/setup.bash # add environment variables



# Compile the function package separately:
# If you only compile "turn_on_mercury_robot", you need to execute the command:
colcon build --packages-select turn_on_mercury_robot
```

Note: If the `mercury_x1_ros2` folder already exists in the `/home/er` directory, you need to delete the original `mercury_x1_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 colocn 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 subfolde

install/: The installation location of each package. By default, each package will be

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
# Issue speed commands at a certain frequency
ros2 topic pub --rate 1 /turtle1/cmd_vel geometry_msgs/msg/Twist "{linear: {x: 2.0, y:
```

- **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:
```

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 grap: /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
```

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-transformat
```

- **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
```

Writer 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:

```python
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 launc 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

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.

If you want to know more information about rviz, you can go to the official documentation to view it.

← Previous Page | Next Page →

Mercury X1 Controls

Here we mainly introduce how to control the movement of Mercury X1 through a series of related instructions.

1. Chassis underlying communication

First, start the underlying communication and map construction program of the chassis, and load the URDF model of Mercury X1.

Open the ROS2 environment terminal and run the command:

```
ros2 launch slam_gmapping slam_gmapping.launch.py
```

You can then control the movement of the joint model in rviz2 by dragging the slider.

2.Joint control

After the URDF model is successfully loaded, if you want the real Mercury X1 arms to move with it, you need to open another ROS2 environment terminal and run the command:

```
ros2 run mercury_x1_control slider_control
```

Please note: Since the robot arm will move to the current position of the model when the command is entered, please make sure that the model in rviz does not have mold penetration before you use the command Do not quickly drag the slider after connecting the robotic arm to prevent damage to the robotic arm

3 Chassis Control

After the joint control program is started, if you want the chassis car in the real Mercury X1 to move with it, you need to start the keyboard control program of the chassis car, open a ROS2 environment terminal, and then run the command:

```
ros2 run mercury_x1_control mercury_keyboard
```

← Previous Page | Next Section →

7 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.

Mobile grabbing wooden block case

```
from uvc_camera import UVCCamera
from arm_control import *
from marker_utils import *
import stag
from mercury_ros_api import MapNavigation

# Gripper tool length
Tool_LEN = 175
# Distance from camera center to flange
Camera_LEN = 78
np.set_printoptions(suppress=True, formatter={'float_kind': '{:.2f}'.format})
# Camera configuration file
camera_params = np.load("src/camera_params.npz")
mtx, dist = camera_params["mtx"], camera_params["dist"]
# QR code size
MARKER_SIZE = 32
# Set left arm port
ml = Mercury("/dev/ttyTHS0")



#Convert rotation matrix to Euler angles
def CvtRotationMatrixToEulerAngle(pdtRotationMatrix):
    pdtEulerAngle = np.zeros(3)
    pdtEulerAngle[2] = np.arctan2(pdtRotationMatrix[1, 0], pdtRotationMatrix[0, 0])
    fCosRoll = np.cos(pdtEulerAngle[2])
    fSinRoll = np.sin(pdtEulerAngle[2])
    pdtEulerAngle[1] = np.arctan2(-pdtRotationMatrix[2, 0],
                                  (fCosRoll * pdtRotationMatrix[0, 0]) + (fSinRoll *
    pdtEulerAngle[0] = np.arctan2((fSinRoll * pdtRotationMatrix[0, 2]) - (fCosRoll *
                                  (-fSinRoll * pdtRotationMatrix[0, 1]) + (fCosRoll *
    return pdtEulerAngle



#Convert Euler angles to rotation matrix
def CvtEulerAngleToRotationMatrix(ptrEulerAngle):
    ptrSinAngle = np.sin(ptrEulerAngle)
    ptrCosAngle = np.cos(ptrEulerAngle)
    ptrRotationMatrix = np.zeros((3, 3))
    ptrRotationMatrix[0, 0] = ptrCosAngle[2] * ptrCosAngle[1]
    ptrRotationMatrix[0, 1] = ptrCosAngle[2] * ptrSinAngle[1] * ptrSinAngle[0] - ptrS
    ptrRotationMatrix[0, 2] = ptrCosAngle[2] * ptrSinAngle[1] * ptrCosAngle[0] + ptrS
    ptrRotationMatrix[1, 0] = ptrSinAngle[2] * ptrCosAngle[1]
    ptrRotationMatrix[1, 1] = ptrSinAngle[2] * ptrSinAngle[1] * ptrSinAngle[0] + ptrC
    ptrRotationMatrix[1, 2] = ptrSinAngle[2] * ptrSinAngle[1] * ptrCosAngle[0] - ptrC
    ptrRotationMatrix[2, 0] = -ptrSinAngle[1]
    ptrRotationMatrix[2, 1] = ptrCosAngle[1] * ptrSinAngle[0]
    ptrRotationMatrix[2, 2] = ptrCosAngle[1] * ptrCosAngle[0]
    return ptrRotationMatrix
```

```
# Coordinates are converted into homogeneous transformation matrices, (x, y, z, rx, ry
def Transformation_matrix(coord):
    position_robot = coord[:3]
    pose_robot = coord[3:]
    #Convert Euler angles to rotation matrix
    RBT = CvtEulerAngleToRotationMatrix(pose_robot)
    PBT = np.array([[position_robot[0]],
                    [position_robot[1]],
                    [position_robot[2]]])
    temp = np.concatenate((RBT, PBT), axis=1)
    array_1x4 = np.array([[0, 0, 0, 1]])
    # Splice the two arrays row by row
    matrix = np.concatenate((temp, array_1x4), axis=0)
    return matrix


def Eyes_in_hand_left(coord, camera):
    # Camera coordinates
    Position_Camera = np.transpose(camera[:3])
    # Robotic arm coordinate matrix
    Matrix_BT = Transformation_matrix(coord)
    #Hand-eye matrix
    Matrix_TC = np.array([[0, -1, 0, Camera_LEN],
                          [1, 0, 0, 0],
                          [0, 0, 1, -Tool_LEN],
                          [0, 0, 0, 1]])
    #Object coordinates (camera system)
    Position_Camera = np.append(Position_Camera, 1)
    #Object coordinates (base coordinate system)
    Position_B = Matrix_BT@Matrix_TC@Position_Camera
    return Position_B


# Wait for the end of the robot arm operation
def waitl():
    time.sleep(0.2)
    while (ml.is_moving()):
        time.sleep(0.03)


# Get object coordinates (camera system)
def calc_markers_base_position(corners: NDArray, ids: T.List, marker_size: int, mtx: N
    if len(corners) == 0:
        return []
    # Obtain the rotation vector and translation vector of the object through the QR
    rvecs, tvecs = solve_marker_pnp(corners, marker_size, mtx, dist)
```

```
    for i, tvec, rvec in zip(ids, tvecs, rvecs):
        tvec = tvec.squeeze().tolist()
        rvec = rvec.squeeze().tolist()
        rotvector = np.array([[rvec[0], rvec[1], rvec[2]]])
        #Convert rotation vector to rotation matrix
        Rotation = cv2.Rodrigues(rotvector)[0]
        #Convert rotation matrix to Euler angles
        Euler = CvtRotationMatrixToEulerAngle(Rotation)
        #Object coordinates (camera system)
        target_coords = np.array([tvec[0], tvec[1], tvec[2], Euler[0], Euler[1], Eule
    return target_coords


if __name__ == "__main__":
    # Navigate to target point
    flag_feed_goalReached = MapNavigation.moveToGoal(1.8811798181533813, 1.2514267325
    # Determine whether the navigation point has been reached
    if flag_feed_goalReached:
        #Set camera id
        camera = UVCCamera(5, mtx, dist)
         #Open camera
        camera.capture()
        # Set the left arm observation point
        origin_anglesL = [-44.24, 15.56, 0.0, -102.59, 65.28, 52.06, 23.49]
        #Set the gripper movement mode
        ml.set_gripper_mode(0)
        # Set tool coordinate system
        ml.set_tool_reference([0, 0, Tool_LEN, 0, 0, 0])
        # Set the end coordinate system to the tool
        ml.set_end_type(1)
        # Set movement speed
        sp = 40
        # Move to the observation point
        ml.send_angles(origin_anglesL, sp)
        # Wait for the robot arm movement to end
        waitl()
         # Refresh camera interface
        camera.update_frame()
        # Get the current frame
        frame = camera.color_frame()
        # Get the angle and id of the QR code in the screen
        (corners, ids, rejected_corners) = stag.detectMarkers(frame, 11)
        # Get the coordinates of the object (camera system)
        marker_pos_pack = calc_markers_base_position(corners, ids, MARKER_SIZE, mtx,
        # Get the current coordinates of the robotic arm
        cur_coords = np.array(ml.get_base_coords())
        #Convert angle value to radian value
        cur_bcl = cur_coords.copy()
```

```
cur_bcl[-3:] *= (np.pi / 180)
 # Convert object coordinates (camera system) to (base coordinate system) thr
fact_bcl = Eyes_in_hand_left(cur_bcl, marker_pos_pack)
target_coords = cur_coords.copy()
target_coords[0] = fact_bcl[0]
target_coords[1] = fact_bcl[1]
target_coords[2] = fact_bcl[2] + 50
# The robotic arm moves above the QR code
ml.send_base_coords(target_coords, 30)
# Wait for the robot arm movement to end
waitl()
 # Open the gripper
ml.set_gripper_value(100, 100)
# The robot arm moves downward along the z-axis
ml.send_base_coord(3, fact_bcl[2], 10)
# Wait for the robot arm movement to end
waitl()
# Close jaws
ml.set_gripper_value(20, 100)

# Wait for the jaws to close
time.sleep(2)

# Lift the gripper
ml.send_base_coord(3, fact_bcl[2] + 50, 10)
```

elephantrobot

1 Company Profile

Elephant Robotics is based in Shenzhen, China. It is a high-tech enterprise focusing on robot R&D, design and automation solutions.

We are committed 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. Its product quality and smart solutions have been unanimously recognized and praised by several factories from the world's top 500 companies such as South Korea, Japan, the United States, Germany, Italy, and Greece.

Elephant Robots adheres to the vision of "Enjoy Robots World" and advocates collaborative work between humans and robots, making robots a good helper for humans in work and life, helping people liberate themselves from simple, repetitive, and boring work and giving full play to the advantages of human-machine collaboration. , thereby improving work efficiency and helping mankind create a better new life.

In the future, Elephant Robot hopes to promote the development of the robot industry through a new generation of cutting-edge technology, and join hands with customers and partners to open a new era of automation and intelligence.

2 Development History

2016.08 -----Elephant Robot Co., Ltd. was officially established

2016.08 -----Enter HAX incubator and obtain 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 -----Attend Hannover Industrial Fair and Korea Automation Exhibition

2017.07 -----The two founders were selected as one of Forbes Asia's "30 Business Elites Under 30"

2017.10 -----The fifth generation single-arm industrial collaborative robot Elephant S is launched

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 "Leading Figure 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 -----The world’s first bionic robotic cat, MarsCat, is launched

2020.05 -----The founder won the 2019 Shenzhen Robot Emerging Figure Award

2020.10 -----The world’s lightest and smallest six-axis collaborative robot myCobot is launched

2021.03 -----The smallest collaborative robot for scientific research, myCobotPro 320, is launched

2021.05 ----- MarsCat has 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 -----The world's first fully wrapped four-axis robotic arm - the little elephant palletizing robotic arm myPalletizer is launched

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 -----The most compact small six-axis robotic arm mechArm is launched, 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 two-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 enganement (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 link
  - 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
- X (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
Mercury via WeChat.

← Previous Page