Dobot E6 Robot ROS Mobile Platform User Guide

Original Instructions

Issue: V1.0

Date: 2025-11-21

SHENZHEN DOBOT CORP LTD

Disclaimer

To the maximum extent permitted by applicable law, the products described (including its hardware, software and firmware, etc.) in this document are provided **AS IS**, which may have flaws, errors or faults. Dobot makes no warranties of any kind, express or implied, including but not limited to, merchantability, satisfaction of quality, fitness for a particular purpose and non-infringement of third party rights. In no event will Dobot be liable for any special, incidental, consequential or indirect damages resulting from the use of our products and documents.

Before using our product, please thoroughly read and understand the contents of this document and related technical documents that are published online, to ensure that the robot arm is used on the premise of fully understanding the robot arm and related knowledge. Please use this document with technical guidance from professionals. Even if follow this document or any other related instructions, damages or losses may happen in the using process. Dobot shall not be considered as a guarantee regarding all security information contained in this document.

The user has the responsibility to make sure following the relevant practical laws and regulations of the country, in order that there is no significant danger in the use of the robot arm.

SHENZHEN DOBOT CORP LTD

Address: Room 1003, Building 2, Chongwen Garden, Nanshan iPark, Liuxian Blvd, Nanshan District, Shenzhen, Guangdong Province, China

Website: www.dobot-robots.com

Preface

Purpose

This document describes the functions, technical specifications, installation guide of Dobot E6 Robot ROS Mobile Platform, making it easy for users to fully understand and use it.

Intended audience

This document is intended for:

- Customer
- Sales Engineer
- Installation and Commissioning Engineer
- Technical Support Engineer

Revision history

Date	Version	Revised content
2025/11/21	V1.0	The first release

Symbol conventions

The symbols that may be found in this document are defined as follows.

Symbol	Description
DANGER	Indicates a hazard with a high level of risk which, if not avoided, could result in death or serious injury.
WARNING	Indicates a hazard with a medium level or low level of risk which, if not avoided, could result in minor or moderate injury, robot arm damage.
NOTICE	Indicates a potentially hazardous situation which, if not avoided, could result in robot arm damage, data loss, or unanticipated result.
NOTE	Provides additional information to emphasize or supplement important points in the main text.

Contents

1. Introduction

The robot is a service robot developed using ROS. It not only features a control interface for its underlying development board but also comes equipped with a variety of extended sensors, such as a depth camera, ultrasonic sensor, smoke sensor, and temperature and humidity sensor. These sensors are like adding eyes, a nose, and a mouth to the robot, enhancing its functionality and enabling it to perform various tasks more intelligently.

The robot service robot can be equipped with an open-source kit designed for the development of various service robots. This kit includes a scalable mobile chassis, a human-machine interaction unit, a food delivery rack, and a disinfection kit. It can support the development of platforms for welcoming robots, food delivery robots, and medical service robots, as well as other types of service robots, such as those equipped with collaborative arms to become picking robots.

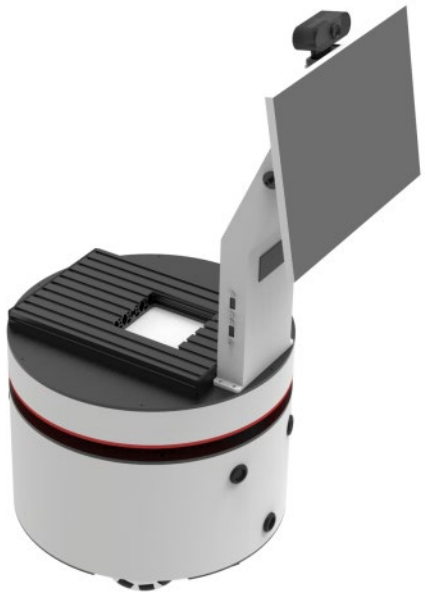


Figure 1.1 Generic chassis - Human-robot interface unit

ROS (Robot Operating System) is an open-source, general-purpose platform for developing robot software. The project originated from a collaboration between the STAIR project at Stanford University's Artificial Intelligence Laboratory and Willow Garage's Personal Robots Program in 2007. Over the years, ROS has become the most widely used general-purpose robot software development platform globally.

ROS is now maintained and managed by the Open Sour Robotics Foundation.

The biggest features of this operation platform are:

- Universality: At present, dozens of robot research and development institutions

and robot manufacturing companies around the world announce that the robots they have developed support ROS every year, which means that more and more robot companies around the world have taken ROS as the standard robot development platform.

- Open source: A large number of open source robot applications are provided on the official website of ROS for users to download and use. Robot developers can easily understand and engage in cutting-edge topics in the field of robot research.
- Reusability: The ROS modular development environment makes it possible to reuse programs, allowing robot developers to use programs that have already been developed by others while focusing on new features.
- Community: Currently, the official ROS forum has brought together a large number of robot developers from around the world. Through collaborative discussions and development efforts, they have made robot development work simpler and more convenient.

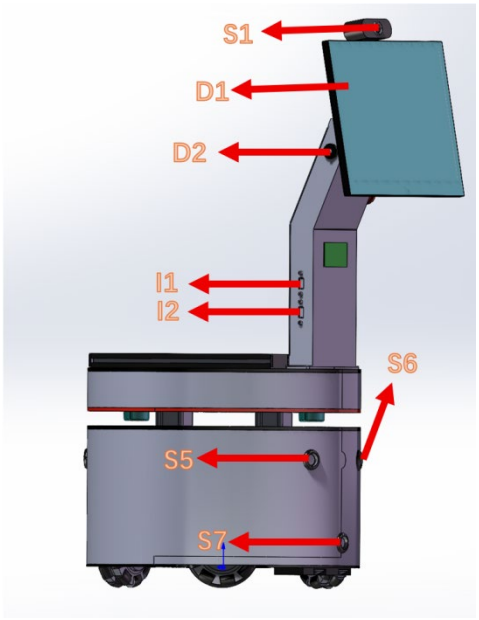
This document focuses on how to use this robot, covering hardware introductions, basic function usage, and comprehensive case studies that integrate these functions. We hope that users will learn more about the ROS platform while using the robot and develop their own functional applications. If any issues arise during use, please feel free to contact us promptly. We hope you enjoy the joy of technology with robot.

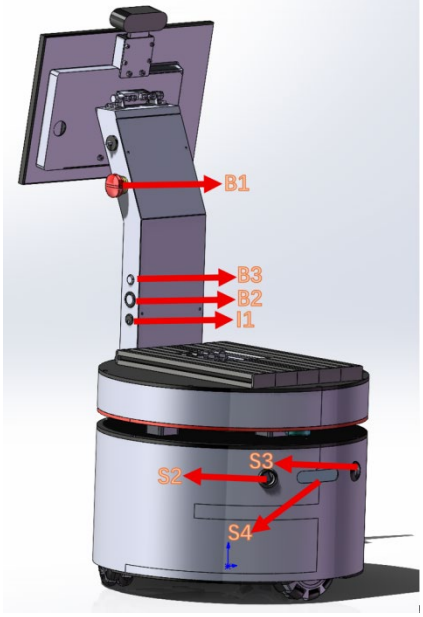
2. Robot specifications and Parameters

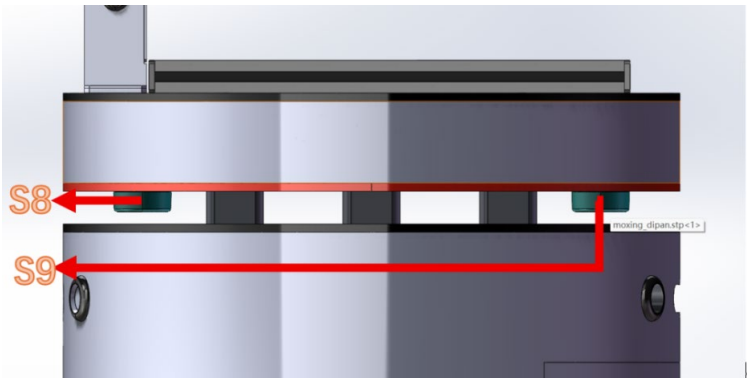
No.	Parameter name	Specifications	Quantity
1	Overall parameters	1. The platform has four degrees of freedom for mobility. It can move in the xy plane, rotate in the plane, perform horizontal rotation, and nod its head through three omnidirectional wheels. Each of the robot's three wheels is equipped with an independent spring pneumatic suspension system, which effectively reduces vibrations during movement. 2. Maximum speed of the robot: linear speed 0.6m/s, angular speed 0.8rad/s; 3. Large range positioning and navigation accuracy: 5cm; 4. Key points use visual secondary positioning to achieve high precision of 2mm; 5. sensors: depth camera*1; 2D LiDAR*2; visual intercom*1; auxiliary positioning camera*1; far-field microphone array*1; 13-inch touch screen LCD*1; ultrasonic sensors*3; temperature and humidity sensor*1; smoke sensor*1, carbon dioxide sensor*1, light intensity sensor*1; 6. Software system: ubuntu20.04+ROS, integrated learning system and development system, power failure reset of learning system, setting permission of development system; 7. Full power operation time: 3h; 8. Standby time: 6h; 9. Overall size: 377mm*377mm*798mm.	1
2	Robot controller	CPU: i5; Memory: 4G; Hard disk: 120G SSD. Interface: RS232*5, RS485*1, network port *2, support wifi, USB3.0*4, USB2.0*4. System: The EROS system is a dedicated AI robot system. It includes: (1) 3D visual simulation scenarios, (2) an integrated deep learning framework, Tensorflow, (3)	1

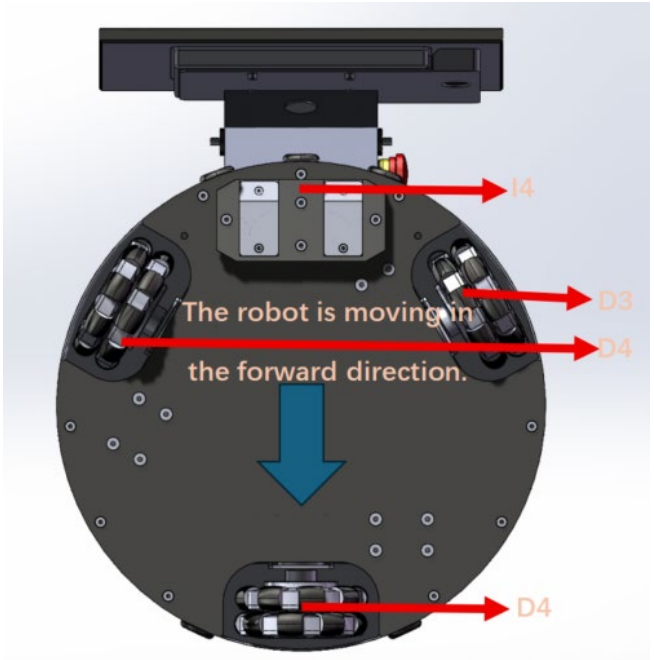
No.	Parameter name	Specifications	Quantity
		various commonly used sensor drivers, such as 2D cameras, 3D cameras, depth cameras, LiDAR, and IMU, (4) common robot localization and mapping algorithms, including gmapping and cartographer, (5) common path planning algorithms, such as A*, D*, DWA, and TEB.	
3	Touch LCD with speaker	Screen size: 13 inches; speaker: stereo full range; resolution: 1440 x 900, touch screen type: capacitive screen.	1
4	Drive and control integrated controller	• board integration: 4 channels of 120W DC servo motor drive; • interfaces: 4 motor drive interfaces, 4 motor encoder interfaces, 3 collision sensors with 5V power supply, 3 fall prevention sensors with 5V power supply, 3 ultrasonic sensors with 5V power supply, 1 temperature and humidity sensor with 5V power supply, 1 smoke alarm sensor with 5V power supply, 1 RS232 interface for communication with the host computer, 1 RS485 interface with 5V power supply, and 1 CAN interface with 5V power supply. • Power output: 12V/5A,5V/5A • Input voltage: 12-30V • Function: Drive the speed closed-loop control of more than 4 DC servo motors, receive speed control instructions through RS232, upload the distance of 3 ultrasonic sensors, the status of 3 collision sensors, the status of 3 anti-fall sensors, the status of 1 smoke sensor and the information of 1 temperature and humidity sensor.	1
5	Depth camera	Color image resolution: 1280x720, frame rate: 30fps or more; Depth image resolution: 1280x1024, frame rate: 7fps or more; Measurement range: 8 meters.	1
6	2D LiDAR	Measurement range: 360°; measurement distance:	2

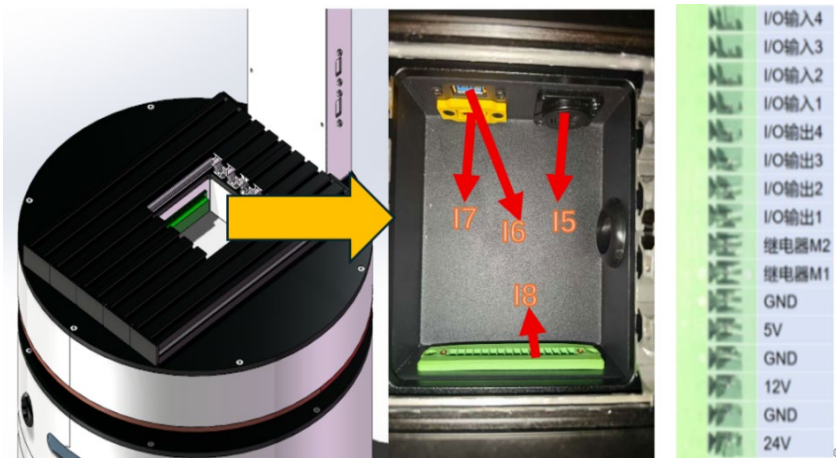
No.	Parameter name	Specifications	Quantity
		10000mm; Angle resolution: 1°	
7	Auxiliary positioning camera	Highest frame rate: 1280*720/30 frames; support for global shutter;	1
8	Interactive recognition camera	Resolution: 1920x1080 (1280 x720); Frame rate: 30fps; Function requirements: support autofocus, built-in microphone	1
10	Automatic charging dock	Input voltage: 25.4V; Charging current: 2A; Automatic charging guidance mode: QR code	1
11	DC servo motor	Encoder line: 500 lines; reduction ratio: 27:1 (planetary gear reduction); rated torque: 1.3Nm; rated speed: 230r/min; power: 30W, supply voltage: DC 24V, no-load current: 130ma.	3
12	Ultrasonic sensor	Working power supply: 5V; measurement distance: 750cm, resolution: 1cm, corresponding frequency: 15HZ, maximum beam angle: 60.	2
13	Battery + charger	Input voltage: 25.4V; output voltage: 21.6-25.2V; number of battery series: 6; battery capacity: 5200mAh, continuous current: 20A.	1
14	Temperature and humidity sensor	Temperature accuracy: ±0.8°; Temperature measurement range: -10° ~50°; Temperature resolution: 0.1℃; Humidity accuracy: ±3%RH; Humidity range: 0%RH-100%RH; Humidity resolution: 0.1%RH;	1
15	Accessories	Wireless mouse: 1 set, controller: 1 set, experiment guide: 1 copy, user guide: 1 copy.	1











No.	Name	Description
B1	E-stop button	Control the motor servo. Press the emergency stop switch to cut off the power of the motor encoder, and the motor loses the servo. Turn it open to control the motor normally. In case the robot loses control or flies, press this switch to stop robot in time.
B2	Switch button	The total switch for all parts of bobac3 is powered on. When the power switch is turned off, all parts stop working.
B3	Main switch button	The main start button for bobac3
D1	High resolution touch screen	The bobac3 screen allows for simple touch operations. Built-in speaker.
D2	Wifi antenna	Enhance the WiFi signal reception of bobac3.
D3	All-direction wheels 1	Driven by DC servo motor, built-in encoder and shock absorption structure.
D4	All-direction wheels 2	Driven by DC servo motor, built-in encoder and shock absorption structure.
D5	All-direction wheels 3	Driven by DC servo motor, built-in encoder and shock absorption structure.
S1	Auto-focus camera	For face recognition, expression analysis and other visual applications. Built-in microphone.

No.	Name	Description
S2	Temperature and humidity sensor	Temperature and humidity can be detected.
S3	Smoke transducer	It can detect flammable gas
S4	Depth camera	RGB, depth and other data can be obtained, which can be used for common visual recognition, local obstacle avoidance and other applications.
S5	Ultrasonic sensor 1	It makes up for the defect that laser sensors cannot detect transparent substances such as glass.
S6	Ultrasonic sensor 2	
S7	Global shutter camera	Images with a maximum of 60 frames per second can be obtained for visual assisted positioning.
S8	Rear 2D LiDAR	Obtaining distance data between two planes is usually used for mapping, obstacle avoidance and other functions.
S9	Front 2D LiDAR	Obtaining distance data between two planes is usually used for mapping, obstacle avoidance and other functions.
I1	Charging port	Adapter charging port.
I2	USB3.0	USB devices can be accessed
I3	USB3.0	USB devices can be accessed
I4	Wireless charging port	When paired with the charging base, automatic charging can be achieved.
I5	LAN	Local LAN interface
I6	USB3.0	Can be connected to USB devices
I7	48V power supply output	Can provide 48V direct current
I8	IO port	Expansion terminal block

3. Introduction of robot Installation and Expansion Interface

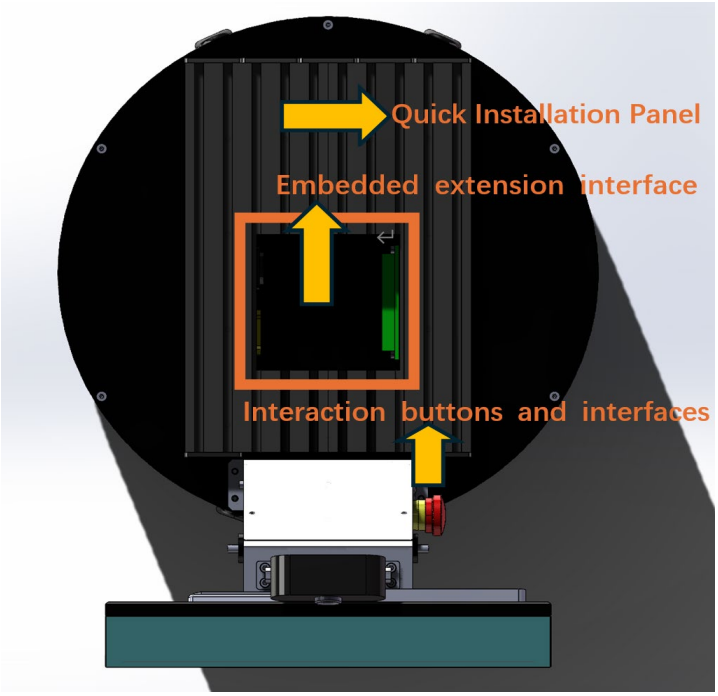
3.1 Description of robot expansion interface

All interactions and extension interfaces of the robot are integrated at the top, as shown in the figure below. It is mainly composed of three parts: interaction box, embedded extension interface and quick installation panel.

Among them, the interface and switch button on the interactive box are mainly used for normal startup, charging, and interactive devices such as mouse, keyboard, wireless arm, etc. which are used for general interaction between human and robot;

The built-in expansion interface is mainly used for the upper-level expansion platform of the robot, such as screen, control of the disinfection platform, common ground of various equipment, etc., which is used for the mutual control between the upper-level application equipment and the robot chassis;

The quick installation panel with profile panels can be quickly fixed on the platform by profile nuts, and the whole panel is profile panels. The installation position is flexible, and there is no need to drill screw holes to fix the upper equipment on the robot chassis.



The specific locations of each interface are shown in the following pictures, and the functions of each interface button are shown in the following table.

No.	Name	Quantity	Description
1	Expand network interface	1	Directly connected to the underlying robot controller, enabling expansion of network communication devices.
2	48V power supply interface	1	Provide 48V DC output
3	USB 3.0 expansion interface	1	The USB is directly connected to the underlying robot controller and is used when external expansion devices are connected, such as the end camera attached to the arm.
4	IO expansion interface	4	Multifunctional expansion interface, integrating 24V, 12V, and 5V power supply, 1 relay output, 4 NPN IO outputs, and 4 IO inputs. The power supply for the upper platform and some general control functions are provided through this expansion port.
5	Automatic charging port	1	In conjunction with the charging dock to achieve automatic charging, only a 54.6V lithium battery charger can be connected, and the current is less than 5A. Remember not to connect the power adapter
6	Charging port	1	The manual charging port can only be connected to the 54.6V lithium battery charger with a current of less than 5A. Remember not to connect the power adapter
7	Start button	1	Press the robot startup switch. Press "On" to turn it on. Press "Off" to turn it off.
8	Main power switch	1	The main power switch for the robot. If it is not activated after being pressed, please charge the robot.
9	E-stop button	1	Emergency braking of the robot
10	Antenna terminal	2	Antenna connector for WIFI connection.
11	Interconnected USB port	2	It is mainly used for devices such as mice, keyboards, wireless controllers, USB drives, etc.

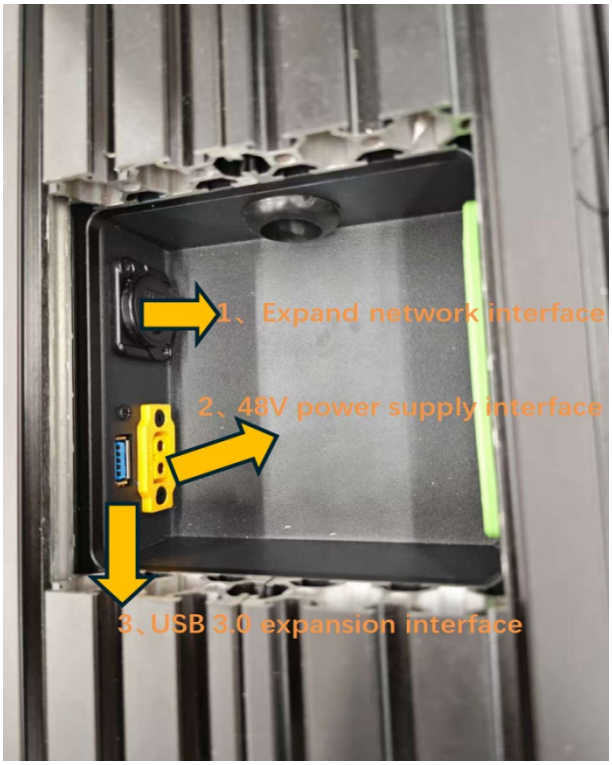


Figure 3.1 Built-in extension interface

Figure 3.2 Built-in extension interface

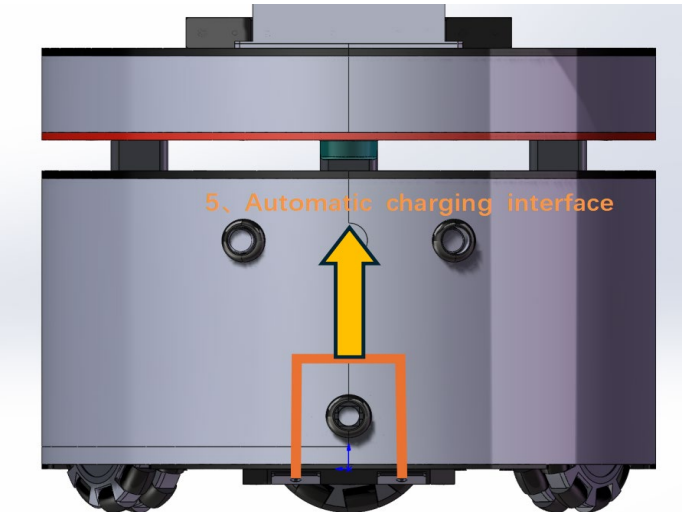


Figure 3.3 Bottom automatic charging port

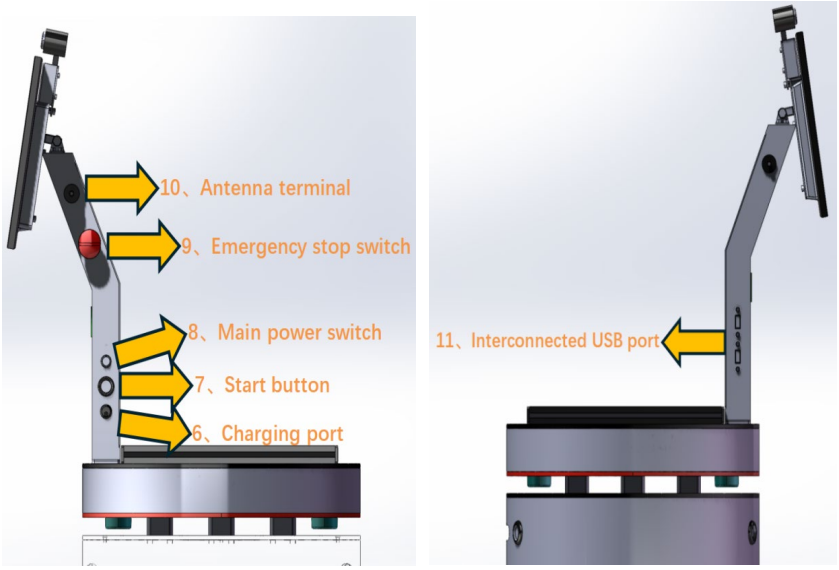
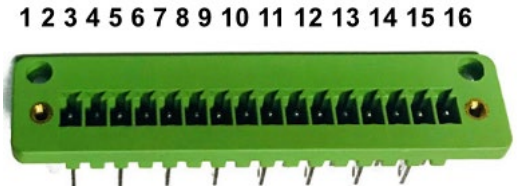


Figure 3.4 Interactive boxes

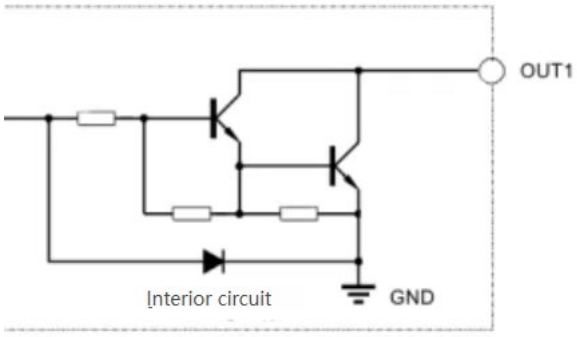
Expansion I/O interface



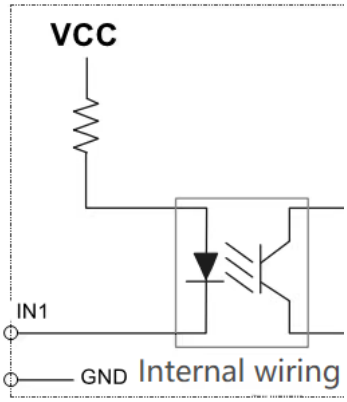
The specific interface pin is defined as follows:

No.	Signal	Interface type	Description
1	24V	Power supply	The output voltage is the internal battery voltage, 20V-25.4V, and the maximum output current is 10A.
2	GND	Power supply	Power ground
3	2V	Power supply	12V constant voltage output, maximum output current 7A
4	GND	Power supply	Power ground
5	5V	Power supply	5V constant voltage output, maximum output current 7A
6	GND	Power supply	Power ground
7	K+	Switch output	At the output end of the relay, K+ and K-are on during output, 5A/250VAC, 5A/30VDC
8	K-	Switch output	At the output end of the relay, K+ and K-are on during output, 5A/250VAC, 5A/30VDC
9	OUT1	IO output	NPN common ground output, maximum output current 250mA
10	OUT2	IO output	NPN common ground output, maximum output current 250mA
11	OUT3	IO output	NPN common ground output, maximum output current 250mA
12	OUT4	IO output	NPN common ground output, maximum output current 250mA
13	IN1	IO input	IO input, connected to GND trigger
14	IN2	IO input	IO input, connected to GND trigger
15	IN3	IO input	IO input, connected to GND trigger
16	IN4	IO input	IO input, connected to GND trigger

NPN common ground output internal electrical diagram:

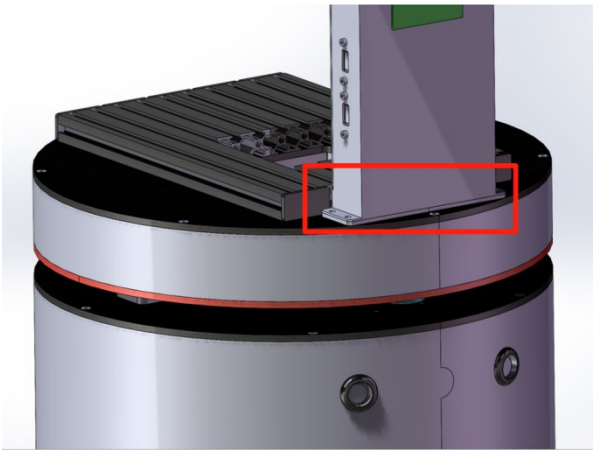


IO input internal electrical diagram:

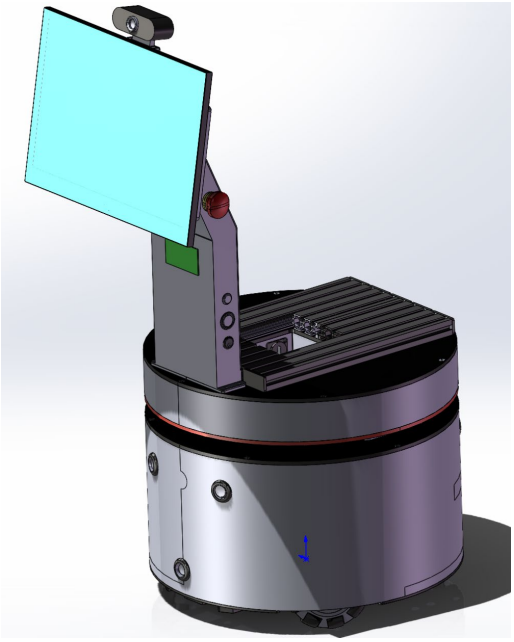


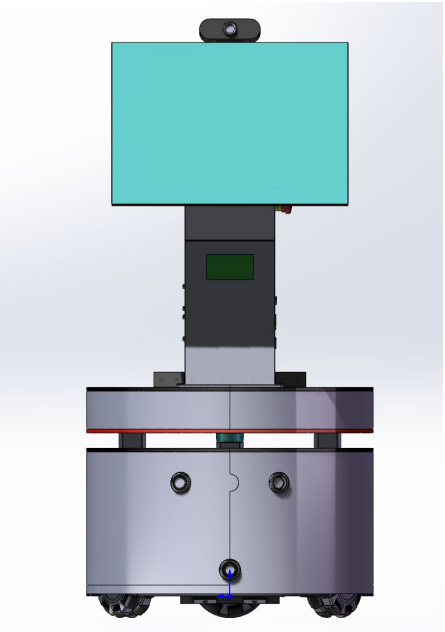
3.2 Installation and disassembly of the welcome kit

The intelligent interaction kit should be installed in advance before installation.



The welcome kit and chassis are securely fastened to the profile groove pit using 8 T-nut blocks and matching hex bolts. During installation, it automatically aligns, requiring only the tightening of the bolts. To disassemble, simply loosen the T-nut blocks and shake the kit to remove it.





4. Basic Operations

Starting with this chapter, the next section will run through all the functions of the robot.

> **NOTE**
>
> The robot is configured with Ubuntu 20.04. To open the terminal, use the 'Ctrl+Alt+T' shortcut and run the commands provided in the corresponding steps. When you are done or starting a new experiment, to prevent process conflicts, it is recommended to terminate the running program by pressing 'Ctrl+d' (it is not advisable to directly close the terminal while the program is running). **Refer to the Appendix** for the creation and modification of the necessary files.

4.1 Joystick control for vehicle

Function: After connecting to the controller interface, the robot can be controlled by publishing messages via the joystick, enabling movement control.

4.1.1 Start the chassis and controller nodes

Open a terminal (Ctrl+Alt+t) and enter the following command to open the chassis node:

```
Roslaunch rei_robot_base base.launch
```

Keep the chassis program running and open a new terminal (Ctrl+Alt+t) to enter the following command to enable the controller control function:

```
Roslaunch robot_joy robot_joy.launch
```

Long-press the MODE button to switch the controller mode.

Set the joystick mode to red light (MODE LED) to indicate that the controller is in control mode.

During operation, the user needs to press both the left and right joysticks simultaneously to switch the control mode of the controller between "wakeup" mode and "sleep" mode.

After waking up the controller, press the emergency stop button, and then you can control the movement of bobac3 robot through the joystick of the controller.

Instruction:
(1) Long-press the MODE button, and release it after the **MODE LED** starts flashing; this will switch the **MODE LED** to a steady red state.
(2) The controller will automatically enter the **sleep mode**. Press both the left and right joysticks simultaneously to switch to the **wake-up mode**.

Key Mapping		Explanation
Left Joystick (L)	Up/Down	Forward/Backward
	Left/right	Left/Right Translation
Right Joystick (R)	Left/right	Left/Right Rotation
Left Shoulder Button (L2)		Acceleration
Right Shoulder Button (R2)		Deceleration
A		Increase Maximum Linear Velocity
B		Decrease Maximum Acceleration
X		Increase Maximum Angular Velocity
Y		Decrease Maximum Angular Velocity

4.2 Map construction and navigation

4.2.1 Start the slam node

The default SLAM algorithm is GMapping algorithm. You can change the SLAM algorithm to cartographer algorithm by setting the "slam_methods" parameter of bobac3_slam.launch file

Open a new terminal and enter:

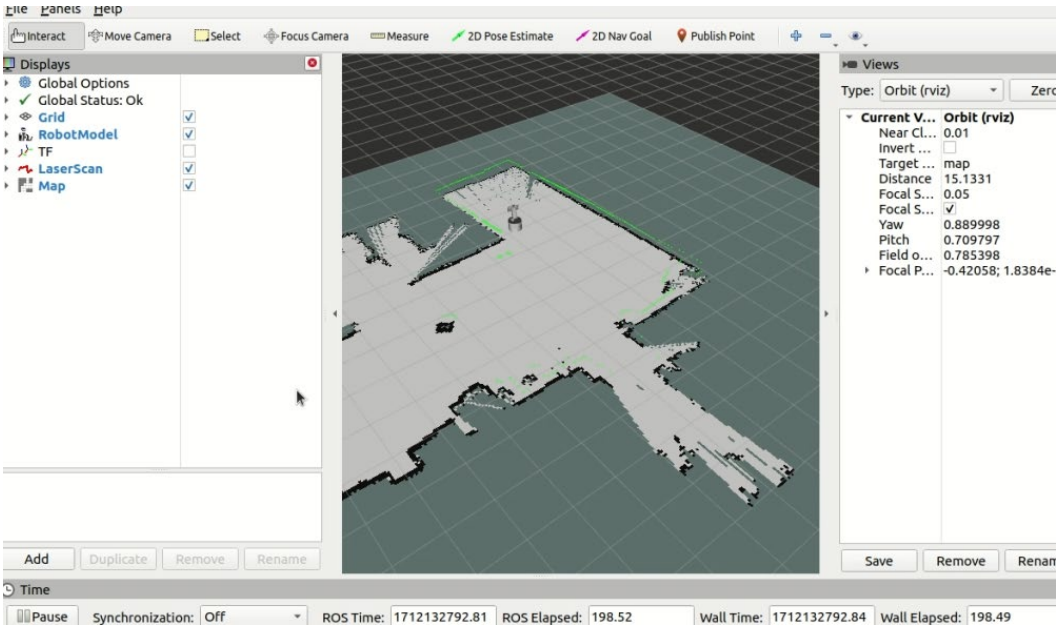
```
# Default GMapping algorithm
Roslaunch bobac3_slam bobac3_slam.launch
```

Or temporarily modified to Cartographer algorithm

```
#Cartographer algorithm
roslaunch bobac3_slam bobac3_slam.launch slam_methods:=cartographer
```

You can see that the display will open rviz. Remember the current position of bobac3 robot when it opens slam. When robot uses the map constructed this time for navigation, this position is the starting point of the robot navigation without any other changes.

The controller can control the robot to move in a map environment and build a complete two-dimensional map.



NOTICE

Permanent modification of the default algorithm: Modify the *default* of the slam_methods parameter in file bobac3_ws/src/bobac3_slam/launch /bobac3_slam.launch.

```
<launch>
  <!-- Arguments -->
  <arg name="slam_methods" default="gmapping" doc="slam type [gmapping, cartographer]"/>

  <arg name="open_slam_rviz" default="true"/>

  <include file="$(find bobac3_description)/launch/display.launch">
    <arg name="open_rviz" value="false"/>
  </include>

```

4.2.2 Save the map

Once the map is built, you need to save the map for navigation.

Keep the slam program running (do not close it), open a new terminal (Ctrl+Alt+t), and enter the following commands in sequence to save the map (map_file specifies the map file name, default is "map"):

```
# The default format of the map created by the gmapping algorithm
roslaunch map_manager map_save.launch map_file:=test
# When using cartographer algorithm to build a map, you need to change the cartographer parameter to save the map
roslaunch map_manager map_save.launch map_file:=test map_type:=cartographer
```

When the interface shows the prompt Done, it means that the map is created and all open terminals can be closed.

NOTICE

You can also directly modify the default values of the "map_type" and "map_file" parameters in the "~/bobac3_ws/src/map_manager /launch/map_save.launch" file to modify the default values of map saving. If you do not modify these two parameters, the map will be saved to the "~/bobac3_ws/src/map_manager/maps" directory after running.

4.2.3 Autonomous navigation

Autonomous navigation is actually the robot's autonomous path planning on the map we have, and then reaching the target point. In order to carry out navigation, it is necessary to load the map first. In the previous section, we have built and saved the map, so how to use the map created by ourselves?

The "map_file_name" parameter is the name of the saved map file.

The default positioning algorithm is AMCL algorithm. The positioning algorithm can be modified to cartographer algorithm by setting the "location_method" parameter of bobac3_nav_2d.launch file.

First move the robot to the starting position (and starting point of navigation) where the map was built, and then enter the following instructions at the terminal:

```
# Use the amcl positioning algorithm
Roslaunch bobac3_navigation bobac3_nav_2d.launch map_file_name:=test
```

Or

```
# Use the cartographer positioning algorithm
Roslaunch bobac3_navigation bobac3_nav_2d.launch location_method:=cartographer
map_file_name:=test
```

It will open up an rviz interface, where the robot model is displayed, and the map that was just saved;

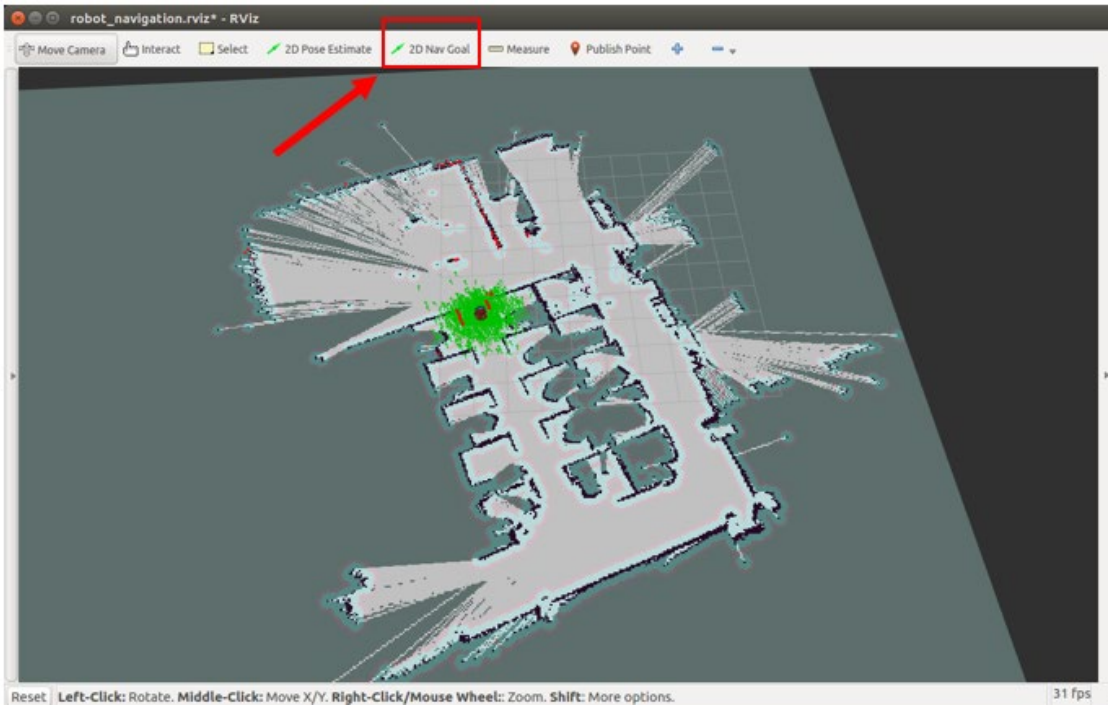


Figure 4.1 Autonomous mapping

The "2D Nav Goal" button in the toolbar above is used to give a navigation goal for a given robot.

Click the "2D Nav Goal" button, then move the mouse to the white area on the map where the robot needs to reach. Hold down the right mouse button and drag to select the robot's orientation. Once you are satisfied, release the right mouse button. The robot will then follow the route planned by the path planner, as shown in Figure 4.3. Green represents the planned path, black objects on the map indicate obstacles, gray areas show the expanded layer of obstacles, and red arrows indicate the destination and the direction after reaching it. If the target point is on a black or gray area, the robot will not move, as it is considered an obstacle and cannot be reached.

NOTICE

Similarly, you can change the default map by modifying the default value of the "map_file" parameter in ~/bobac3_ws/src/bobac3_navigation/launch/bobac3_nav_2d.launch. If you don't modify it, the map named "home" will be loaded by default.

4.3 Secondary Positioning

Function: The robot will use the bottom camera to identify the AR marker on the charging dock. It acquires the marker's position and then controls the robot to move directly facing the AR marker, finally stopping at a precise position directly in front of it.

Turn on the robot's emergency stop. Place the charging dock within the field of view of the robot's bottom camera (confirm the location of the bottom camera) as shown in the figure below:

4.3.1 Open related nodes

Open a **new robot terminal** and enter the following command to launch the chassis node:

```
roslaunch rei_robot_base base.launch
```

Open a **new robot terminal** and enter the following command to launch the bottom camera node:

```
roslaunch ar_pose ar_base.launch
```

Open another **new robot terminal** and enter the following command to launch the robot model:

```
roslaunch bobac3_description display.launch
```

Open another **new robot terminal** and enter the following command to run the relative movement node:

```
roslaunch relative_move relative_move.launch
```

Open another **new robot terminal** and enter the following command to start the AR marker recognition and tracking program:

```
rosservice call /track "ar_id: 0
goal_dist: 0.2"
```

"ar_id: 0": 0 represents the ID number of the AR marker on the charging dock. goal_dist: 0.2" specifies that the robot will stop at a distance of 200 mm in front of the charging dock.

4.4 Automatic recharge

Autonomous recharge is the process in which the robot returns to the charging dock to charge itself when the battery is low or the task is completed.

It needs to be coordinated with autonomous navigation and QR code guidance.

4.4.1 Place the charging dock

Place the charging dock in a position where the robot can navigate to, against the wall, and connect the charger.

4.4.2 Obtain the navigation coordinates of charging dock

Refer to 4.2.3 to open the navigation function:

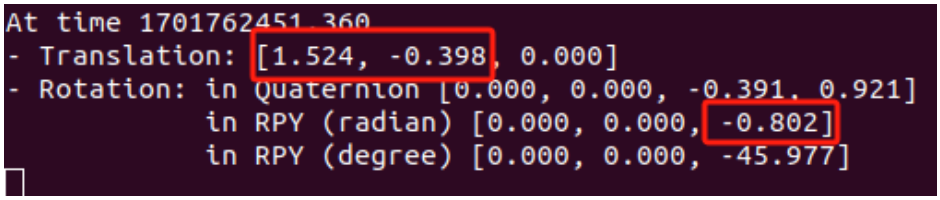
```
roslaunch bobac3_navigation bobac3_nav_2d.launch
```

Open the controller control node (refer to 5.1 for the control method of the controller):

```
roslaunch robot_joy robot_joy.launch
```

Control the robot to move so that the bottom camera is facing the QR code of the charging dock about 30~40 cm in front. Another terminal is opened to query the current posture of the robot (/base_footprint is the origin of the robot, /odom is the coordinate system of the robot odometer):

```
rosrun tf tf_echo /odom /base_footprint
```



4.4.3 Run automatic recharge

Open the terminal and refer to 5.2.3 to open the navigation function:

```
roslaunch bobac3_navigation bobac3_nav_2d.launch
```

Open another **new robot terminal** and enter the following command to run the AR marker recognition:

```
roslaunch ar_pose ar_base.launch
```

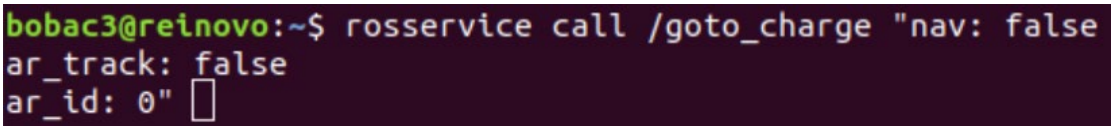
Open another **new robot terminal** and enter the following command to execute the relative movement node:

```
roslaunch relative_move relative_move.launch
```

Open a **new robot terminal** and enter the following instructions to run autonomous recharge:

```
roslaunch auto_charging auto_charging.launch
```

After all nodes run successfully, open a new terminal and enter the following instructions to set the parameters corresponding to the charging service and start charging (it is recommended to use TAB to complete the content and modify the parameters):



- When the nav parameter is set to true, navigate to the target point position.

Pose represents the target point for navigation.

- track_id: The ID of the AR marker on the charging dock (default is marker 0).
- track_dist: The distance between the robot's center and the charging dock.

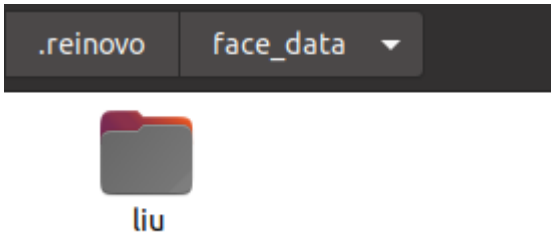
- docking_dist: The additional distance to move towards the dock after reaching the specified track_dist.

At this point, the robot will navigate itself to the charging dock and recharge.

4.5 Face detection and recognition

4.5.1 Add new faces

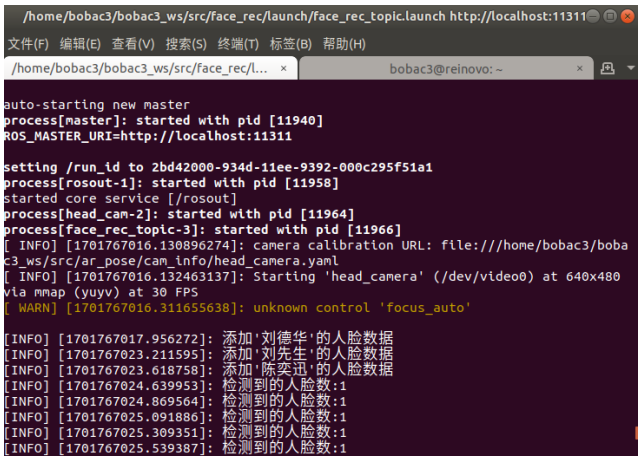
Enter the /home/bobac3/.reinovo/face_data directory and create a new directory (named after the face that recognizes it), and put the corresponding face data (take a picture of a face without anyone else in it) into the newly created directory.



4.5.2 Subscribe to topics to get results in real time

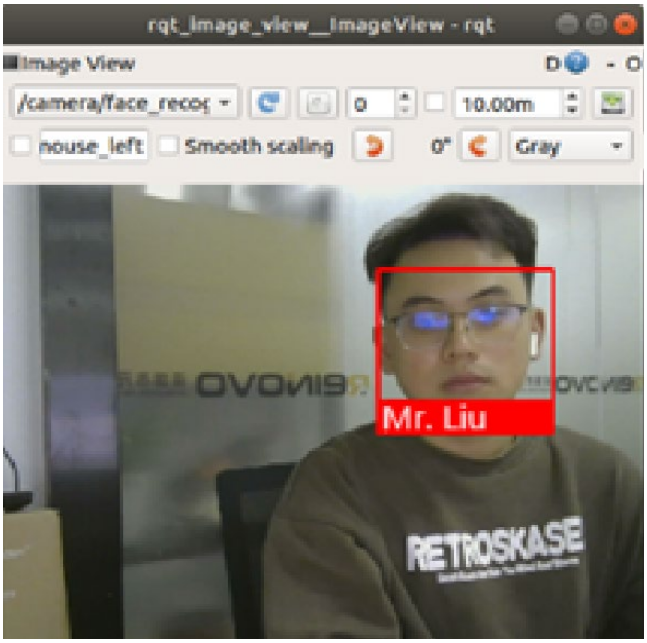
Open the terminal to run

```
roslaunch face_rec face_rec_topic.launch
```



Open a new terminal and subscribe to the topic /camera/face_recognition:

```
rqt_image_view
```



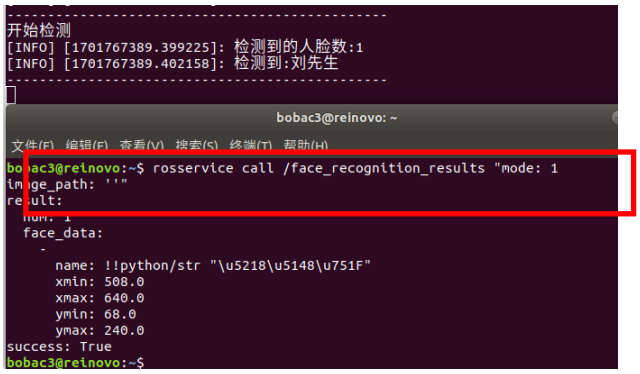
4.5.3 Call the service to obtain the result

Open the terminal to run

```
roslaunch face_rec face_rec_service.launch
```

Open another terminal and send a service request. When mode is 1, the current camera picture is identified:

```
rosservice call /face_recognition_results <Use TAB to complete>
```



Open another terminal and send a service request. When mode is 2, the specified image can be identified:

5. Bobac3 Hardware Detection

5.1 Controller

5.1.1 Wake up the controller

Insert the receiver end of the controller into the USB port of the robot. If the controller is in sleep state (the two lights on the top of the controller are all off), click the mode key or the round button of the start key on the front of the controller to activate it. The default activation state is the green light is on.

5.1.2 Check the device input

Enter the following command at the terminal:

```
ls /dev/input
```

As shown in Figure 3.1 below, the device input list contains devices with the word js, indicating that the system recognizes the controller.

Figure 5.1 List of input devices

A js serial number corresponds to only one controller device. When multiple controller devices are connected, the system will number them according to the power-up order of the USB port when starting up.

5.1.3 Corresponding number of test controller

If multiple js are found, jstest can be used to determine the js serial number corresponding to the controller.

Enter the following command at the terminal:

```
sudo jstest /dev/input/js0
```

Turn the joystick of the controller. If there is data on the terminal, it indicates that this controller corresponds to the corresponding serial number. If there is no response, continue to test the next js number. After determining the corresponding js number, write it down for later use.

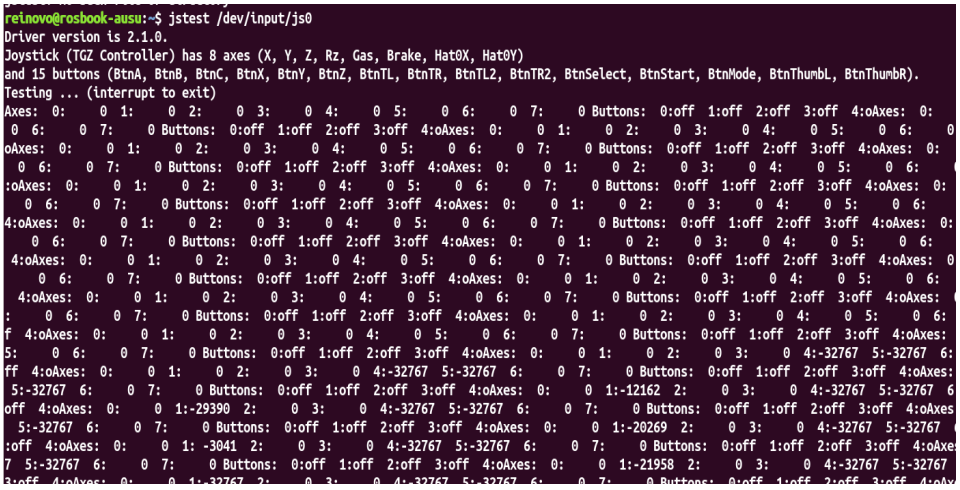


Figure 5.2 Correct sample of controller test

Use the roscd command on the terminal to enter robot_joy/launch:

```
Roscd robot_joy/launch
```

Modify robot_joy.launch:

```
<param name="button_max_angular_increase" value="2" type="int" />
<param name="button_max_angular_reduce" value="3" type="int" />

<param name="linear_vel_max" value="1.0" type="double" />
<param name="angular_vel_max" value="2.0" type="double" />
<param name="linear_vel" value="0.4" type="double" />
<param name="angular_vel" value="0.5" type="double" />
</node>
<node respawn="true" pkg="joy" type="joy_node" name="joy_node">
<param name="dev" type="string" value="/dev/input/js0" />
  <param name="deadzone" value="0.12" />
</node>
</launch>
```

Figure 5.3 bobac3_joy.launch

Find the parameter "dev" and fill in the previously determined js number.

If you only have one controller, this step only needs to be operated once, confirm the js number, under normal circumstances, there is no need to operate again.

5.1.4 Control nodes with the controller

Enter the following command at the terminal:

```
roslaunch robot_joy robot_joy.launch
```

This function subscribes to the key button information of the controller: /joy, and the message type is sensor_msgs/Joy. Publishes the speed message: /cmd_vel, and the topic type is geometry_msgs/Twist.

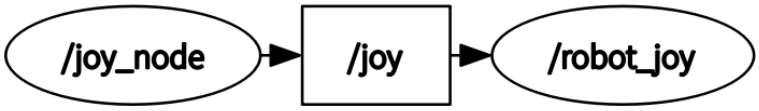


Figure 5.4 Controller node communication diagram

The specific message members are shown in Figure 5.5 and Figure 5.6.

```
bobac3@reinovo:~$ rosmsg show geometry_msgs/Twist
geometry_msgs/Vector3 linear
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 angular
  float64 x
  float64 y
  float64 z
```

Figure 5.5 geometry_msgs/Twist information

```
bobac3@reinovo:~$ rosmsg show sensor_msgs/Joy
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
float32[] axes
int32[] buttons
```

Figure 5.6 sensor_msgs/Joy information

Control mode: The left joystick of the controller controls the speed command of moving forward, backward, left and right, and the right joystick controls the speed command of

turning left and right. The above is the observation node. For the specific steps of controlling the controller to control the car, please see 4.1.

5.2 Detection of chassis sensors and wheels

5.2.1 Robot chassis node start

To get the sensor data, you need to start the robot's chassis node.

Open the terminal (Ctrl+Alt+t) and enter the following command:

```
roslaunch rei_robot_base bobac3_base.launch
```

5.2.2 View sensor information

After the operation is successful, a topic named /car_data will be sent out. By monitoring this topic, you can obtain real-time data of each motor speed and sensor. Keep the program of bobac3_base running, open a new terminal (Ctrl+Alt+t), and enter the following instructions:

```
Rostopic echo/car_data
```

As shown in Figure 5.7:

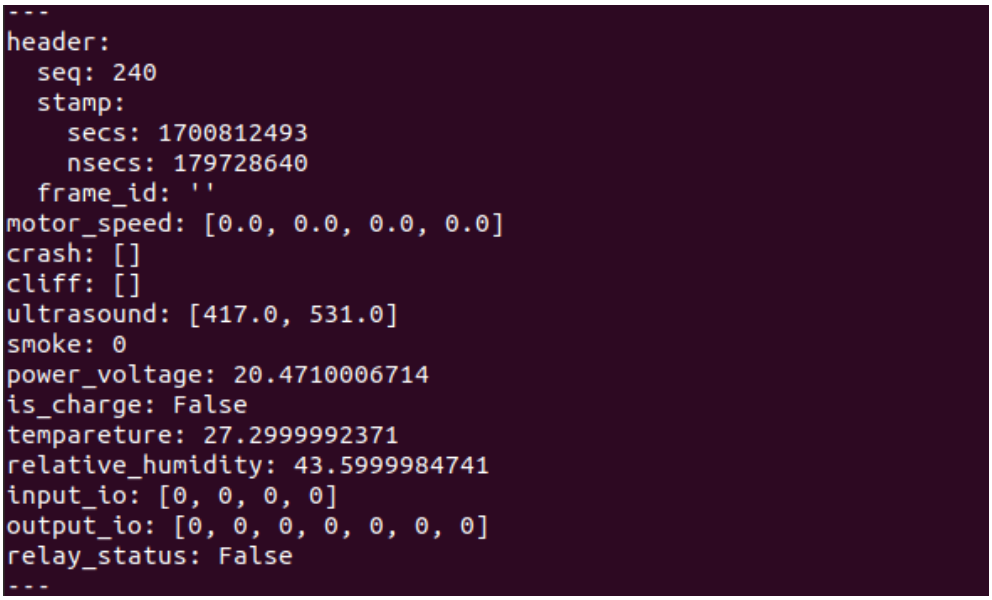


Figure 5.7 car_data data

Look down from above:

- Motor_speed is the speed of the three motors. When you press the emergency stop and push the robot by hand, you will see the corresponding data change.
- Ultrasound: two ultrasonic sensor messages.
- Smoke: Smoke sensor, which has two state values, 0 and 1. In a normal environment, it is displayed as 1. When the smoke sensor detects combustible gas, the state value is 0.
- Power voltage: Check the battery voltage message, you can see the current voltage of the battery here. When the voltage is less than 21V, it needs to be charged immediately.
- Is_charge: Detect whether it is charging, only for detecting whether it is charging from the bottom.

- Temperature is a measure of the current temperature of the environment.
- Humidity is the detection of the current humidity in the environment.
- input_io: Input IO information
- output_io: Output IO information
- relay_status: Relay status

5.2.3 Robot bottom wheel detection

After launching the robot base node and opening the sensor data monitoring terminal, press the emergency stop button. Then, manually rotate the wheels on the base. You can observe the motor_speed values (which represent the rotational speeds of the three motors) on terminal information in the View sensor information. If the data changes, it indicates that the wheels are functioning normally.

5.3 Turn on the LiDAR

5.3.1 Check whether the LiDAR is recognized

Enter the command at the terminal:

```
ls /dev/*lidar
```

/dev/output /dev/front_lidar indicates that the front lidar is detected, and /dev/rear_lidar indicates that the rear LiDAR is detected.

5.3.2 Start the LiDAR node

Enter the following command at the terminal to enable the LiDAR node:

```
Roslaunch rei_ydlidar_nodelet nodelet_2_lidar.launch
```

If there is no error, the startup is successful.

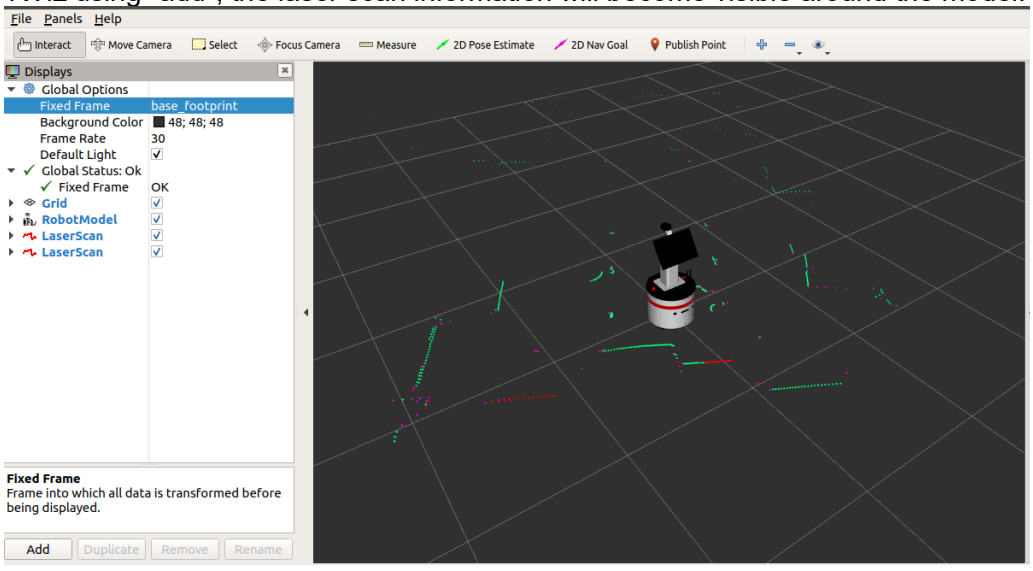
5.3.3 Display LiDAR data in rviz

Keep the lidar in step 5.3.2 Start the LiDAR node running.

Open a terminal and enter the following command to open the 3d model of the robot:

```
roslaunch bobac3_description display.launch
```

At this point, the robot's 3D model will be displayed. By adding the LiDAR data in RViz using "add", the laser scan information will become visible around the model.



5.4 Depth camera image view

The terminal runs the following command:

```
Roslaunch berxel_camera berxel_base_camera.launch
```

After the operation is successful, keep the program running and open a new terminal to enter the following instructions to open rviz to view the image:

```
rviz
```

Follow these steps to see the image:

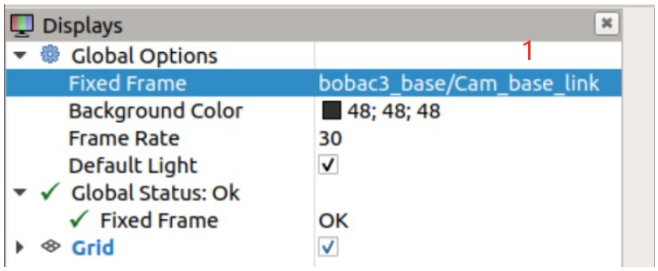


Figure 5.8 Modify fixed_frame

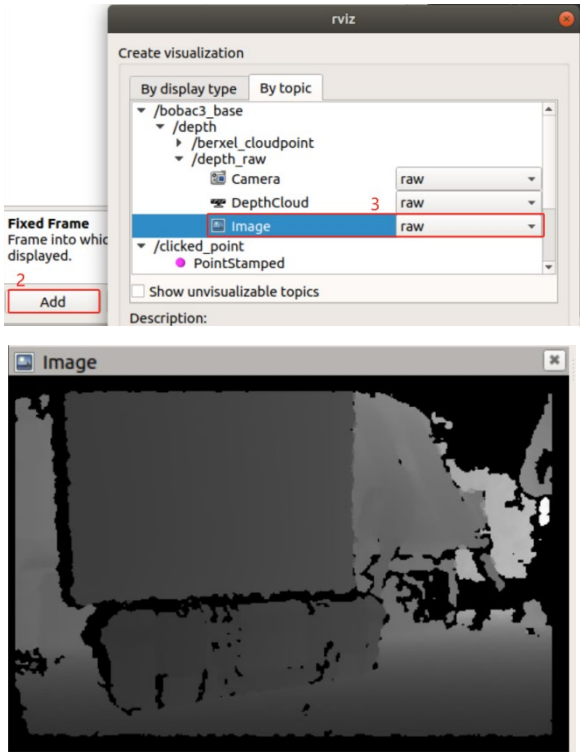
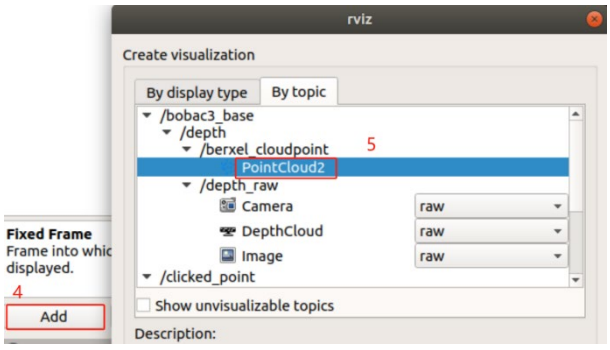


Figure 5.9 Add and display the depth image



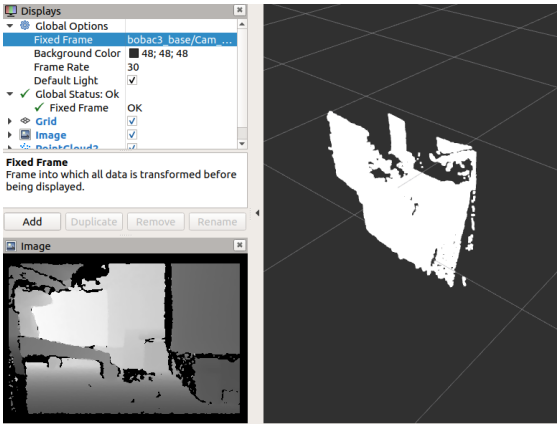


Figure 5.10 Point cloud display

5.5 USB camera image viewing

The robot is equipped with two USB cameras, one at the bottom and one at the head. The bottom camera is used for landmark recognition, and the head camera is used for face recognition and interaction.

5.5.1 Open and view the bottom camera

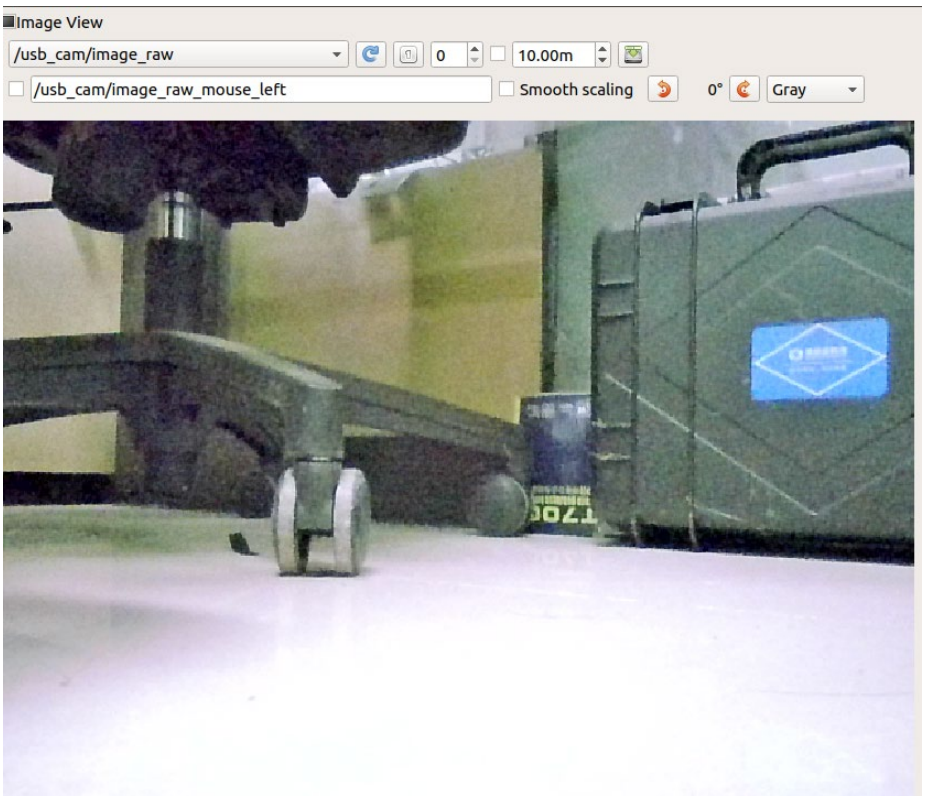
Open a terminal and enter the following command to open the camera:

```
$ roslaunch ar_pose usbcam_base.launch
```

Open another terminal and enter the following command:

```
$ rqt_image_view
```

As shown in the figure below, select the topic name to be displayed as: /usb_cam/image_raw.



5.5.2 Turn on and view the head camera

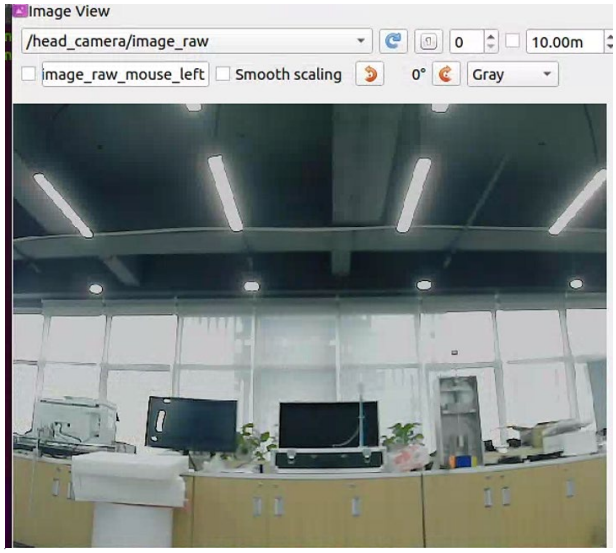
Open a terminal and enter the following command to open the camera:

```
$ roslaunch ar_pose usbcam_head.launch
```

Open another terminal and enter the following command:

```
$ rqt_image_view
```

As shown in the figure below, select the topic name to be displayed as: /usb_cam/image_raw.



5.6 Turn on the simulated disinfection lamp

The disinfection lamp of the disinfection robot is controlled by the robot expansion IO1.

Open a terminal and enter the following command to open the chassis:

```
$ roslaunch rei_robot_base bobac3_base.launch
```

Open a terminal again and enter the following command to call the service to turn on the light:

```
bobac3@reinovo:~$ rosservice call /set_io "{all_off: false, all_on: false, io: [1, 0, 0, 0, 0, 0, 0], state: true}"
```

To turn off the service light, open a terminal and enter the following command:

```
bobac3@reinovo:~$ rosservice call /set_io "{all_off: false, all_on: false, io: [1, 0, 0, 0, 0, 0, 0], state: false}"
```

5.7 Turn on and off the atomizing sterilizer

The sterilizer is controlled by a robot extension relay.

Open a terminal and enter the following command to open the chassis:

```
$ roslaunch rei_robot_base bobac3_base.launch
```

Open a terminal again and enter the following instructions to call the service to open the relay and open the sterilizer

```
$ rosservice call /set_relay "data: true"
```

Open the terminal and enter a command to turn off the sterilizer

```
$ rosservice call /set_relay "data: false"
```

6. Comprehensive Application

The previous foundational modules have introduced the core functionalities of the bobac3 robot. Through hands-on experimentation, you should now be familiar with the operation and principles of its various sensors. This chapter builds upon that knowledge, guiding you to integrate these hardware components within the ROS framework to develop complex algorithms for multi-task applications.

This chapter combines autonomous navigation, chassis control, and voice recognition to create a comprehensive voice-interaction application for the robot.

6.1 Voice integration application

6.1.1 Voice command control of the robot

Previous experiments have demonstrated the voice chat functionality, which primarily utilized the voice module alone. The voice module can now be integrated with other systems to enable comprehensive applications.

The following experiment will involve controlling the bobac3 robot's movement through voice commands. Examples include: "move forward", "move backward", "turn left" and "turn right".

Ensure the robot is connected to the network, then open a terminal and enter the following command to launch the bobac3 voice chat system:

```
Roslaunch robot_audio voice_interaction.launch
```

Keep this program running, open a new terminal (Ctrl+Alt+T), and enter the following command to launch the robot's base node:

```
Roslaunch rei_robot_base base.launch
```

Open the robot relative motion node:

```
roslaunch relative_move relative_move.launch
```

After all systems are launched, release the emergency stop button and use the following voice command to switch the bobac3 robot into control mode.

```
User: "Yuanbao, Yuanbao"
Bobac3 Robot: "Yes, what can I do?"
User: "Please move forward 0.1 meters."
Bobac3 Robot: "Okay, moving forward 0.1 meters."
User: "Yuanbao, Yuanbao"
Bobac3 Robot: "Yes, what is it?"
User: "Please move left 0.1 meters."
Bobac3 Robot: "Okay, moving forward 0.5 meters."
```

No problem if it misrecognizes. Just wake it up again and re-interact. If it prompts "Say it again", no wake-up is needed.

Recognizable directional commands: forward, backward, turn left, turn right, move left, move right.

Recognizable distance units: meter, decimeter, centimeter, radian, degree.

6.1.2 Intelligent voice navigation

Similar to the voice control functionality described above, this voice navigation feature replaces the manual mouse operation for setting the robot's navigation target points. However, prior to this, we need to assign specific location names to corresponding coordinate points on the map.

1. First, follow the steps in Section 4.2 to construct and load the map. This step can be skipped if already completed during previous operations.
2. Next, define location names for specific positions on the map according to your requirements. For example, identify and label points such as the doorway, office, dining area, etc. Afterwards, we need to capture the precise coordinate data for these labeled points.

Move the robot to the intended navigation starting point, then open a terminal and enter the following command to launch the navigation function:

```
Roslaunch bobac3_navigation bobac3_nav_2d.launch
```

Press the E-stop button to immobilize the robot. While keeping the current program running, open a new terminal (Ctrl+Alt+T) and enter the following command to monitor the /move_base_simple/goal topic:

```
Rostopic echo /move_base_simple/goal
```

Return to the rviz interface and use the2d_nav_goal tool to mark the target pose for your desired location name. The corresponding coordinate information will then be visible in the second terminal we opened earlier. For example: to designate point A on the map as the "kitchen", use the 2d_nav_goal tool to set the target pose for the robot at that point. This means when the robot reaches this target position, it will be considered to have arrived at the kitchen. Record the corresponding coordinates (x, y) and quaternion values (z, w) as shown in Figure 5.3.

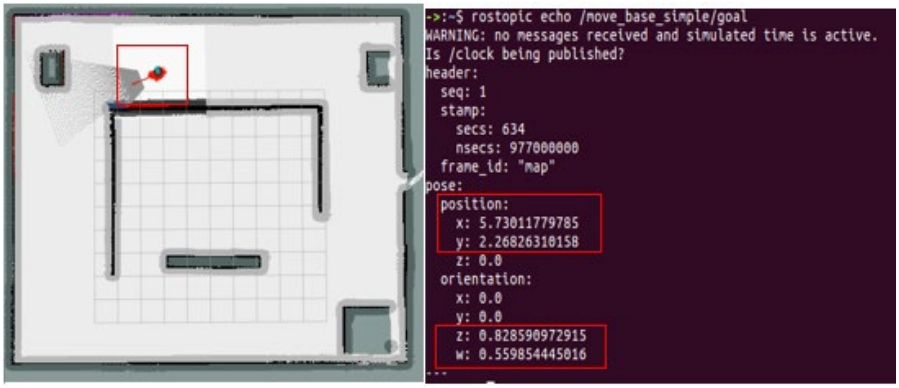


Figure 6.1 Acquiring coordinate data

Navigate to the home directory and use Ctrl+H to display hidden files. Open the configuration file position_info.txt located at /home/bobac3/.ros/AIUI/dist. Configure the **waypoint names** according to the format shown in the image below, and modify the waypoint coordinates in the configuration file by entering the values in the order of x, y, z, w. The description for each location is added afterward.

```
1 Finance Office  8.04  -0.33  -0.75 0.665  This is the Finance Office, the place for
financial settlement
2 R&D Department  4.079 0.619 0.699 0.715  This is the R&D Department, the place for
research and development
3 Production Department  1.075 2.95 0.707 0.707  This is the Production Department, the
place for manufacturing
4 Debugging Area  -2.5   2.28 0.72 0.69    This is the Debugging Area, the place for
testing and debugging
5 Reception Desk    -3.58  -3.72 0.998 0.06  This is the Reception Desk, welcome for you
```

After configuring the first waypoint, use the 2d_nav_goal tool again to mark the next location. Repeat this process until all required locations are configured. Save and close the position_info.txt file, then terminate all previously running programs.

3. Configuring voice recognition points

To ensure the robot understands our navigation intent upon hearing the predefined location names, we need to upload these names to the voice server. Navigate to the path /home/bobac3/.ros/AIUI/dist, open the pos_name.txt file, and list all navigation point names in the specified format.

```
1 {"name": "Kitchen", "alias": "Place for cooking"}
2 {"name": "Living Room", "alias": "Place for receiving guests"}
3 {"name": "Bedroom", "alias": "Place for sleeping"}
4 {"name": "Reception Desk", "alias": "Place for reception"}
5 {"name": "R&D Department", "alias": "Place for research"}
6 {"name": "Production Department", "alias": "Place for production and processing"}
7 {"name": "Finance Office", "alias": "Place for cost control"}
8 {"name": "Service Hall", "alias": "Place for handling affairs"}
9 {"name": "Registration Office", "alias": "Place for registration"}
10 {"name": "Exhibition Center", "alias": "Place for exhibitions"}
11 {"name": "Administrative Center", "alias": "Place for administration"}
12 {"name": "Debugging Area", "alias": "Place for debugging"}
```

4. Compiling the workspace

The following step is to compile the workspace so that the changes made in position_info.txt and pos_name.txt can take effect. In the terminal, enter the following commands sequentially:

```
Cd ~/bobac3_ws
catkin_make
```

5. Program execution

After the compilation is complete, move the robot to the navigation starting point. Then, open a terminal and enter the following command to launch the program:

```
Roslaunch bobac3_navigation bobac3_nav_2d.launch
```

Then, in another terminal, enter the following command:

```
Roslaunch robot_audio voice_interaction.launch
```

① Use the voice_up_sync service to upload the voice recognition point to the server.

```
rosservice call /voice_up_sync
```

② Voice navigation

After activation, wake up the robot and say to the bobac3 robot: "Take me to..." (or "Navigate to..."). The destination must be a waypoint defined in the position_info.txt configuration file; otherwise, it will not be recognized.

For example, if you say to the bobac3 robot: "Take me to the balcony", the robot will move to the target point and then play the voice description corresponding to that waypoint.

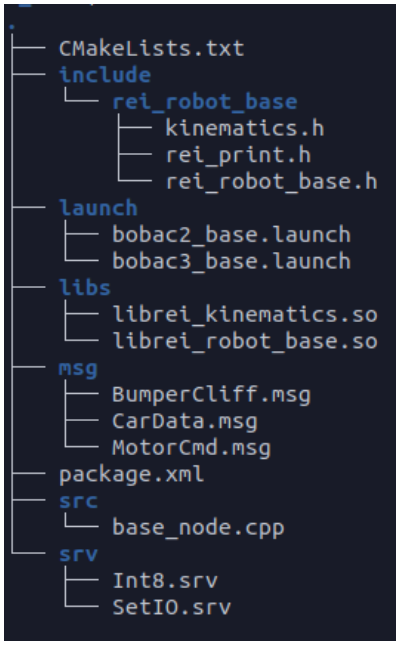
7. Robot ROS Development API

7.1 rei_robot_base

Function: Communicates with the lower-level controller to read motor speed, ultrasonic sensor data, temperature and humidity, smoke, and I/O status. Controls motor operation, digital I/O, and relays.

Node name: robot_base_node

7.1.1 File structure



File name	Function
kinematics.h	Kinematics solver header file
rei_print.h	Information printing header file
rei_robot_base.h	Base communication header file
bobac3_base.launch	Launch file
librei_robot_base.so	Base communication encapsulated library
librei_robot_kinematics.so	Kinematics solver encapsulated library

File name	Function
BumperCliff.msg	Bumper/cliff sensor custom message
CarData.msg	Lower-level controller data custom message
MotorCmd.msg	Motor speed control custom message
base_node.cpp	ROS node
Int8.srv	Buzzer control custom service file
SetIO.srv	Output I/O control custom service file

7.1.2 ROS API

Published TF

(~robot_base_frame_id)<——(~odom_frame_id) (Published when publish_odom_tf is set to true)

Subscribed topic

Topic name (message type)	Function	Parameter description
cmd_vel(geometry_msgs/Twist)	Velocity control	http://docs.ros.org/en/melodic/api/geometry_msgs/html/msg/Twist.html
motor_cmd(rei_robot_base/MotorCmd)	Motor speed control	std_msgs/Header header: Message header float64[] motor_expect_speed: Target speed for each motor

Published Topic

Topic name (message type)	Function	Parameter description
car_data(rei_robot_base/CarData)	Lower-level controller data	std_msgs/Header header: Message header float64[] motor_speed: Current speed of each motor (r/min) int8[] crash: Values from each collision sensor int8[] cliff: Values from each cliff sensor float32[] ultrasound: Data from each ultrasonic sensor int8 smoke: Smoke sensor value float32 power_voltage: Battery voltage (V) bool is_charge: Charging status float32 temperature: Temperature value (°C) float32 relative_humidity: Relative humidity value (RH) int8[4] input_io: Status of each input IO int8[7] output_io: Status of each output IO bool relay_status: Relay status
odom(nav_msgs/Odometry)	Wheel odometry data (published when publish_odom is set to true)	http://docs.ros.org/en/melodic/api/nav_msgs/html/msg/Odometry.html

Services

Topic name (message type)	Function	Parameter description
pub_odom(std_srvs/SetBool)	Control whether to publish the odometry topic	http://docs.ros.org/en/api/std_srvs/html/srv/SetBool.html data=true: Enable publishing/ON data=false: Disable publishing/OFF
pub_odom_tf(std_srvs/SetBool)	Control whether to publish the odometry tf	/
set_relay(std_srvs/SetBool)	Control relay switching	
set_io(rei_robot_base/SetIO)	Control output IO	bool all_off: When true, all IO outputs 0 (effective only when all_on is false) bool all_on: When true, all IO outputs 1 (effective only when all_off is false) int8[] io: Effective only when both all_on and all_off are false. Select the IO port number(s) to change (one or multiple; there are only 7 output IOs, numbering starts from 1) bool state: The state to set for the IO ports listed in int8[] io --- bool success: Returns true if successful string message
clear_odom(std_srv/Empty)	Reset wheel odometry data	http://docs.ros.org/en/api/std_srvs/html/srv/Empty.html
set_buzzer(rei_robot_base	Control the buzzer	int8 data: Buzzer sounds when

Topic name (message type)	Function	Parameter description
/Int8)		value > 0 --- bool success string message
set_extra_motor(rei_robot_base/Int8)	Control the additional expansion motor (valid for oryxBot robot)	int8 data: 0=Stop, 1=Forward, 2=Reverse --- bool success string message

Configuration parameters

Parameter name	Description
robot_model	string. Robot model. Options: bobac2, bobac3, oryxbot, fox
port	string. Communication serial port
baud	int. Default 115200. Baud rate
data_bit	int. Default 8. Data bits
stop_bit	int. Default 1. Stop bits
slave	int. Default 1. Slave address
range_1_frame_id	string. Default ultrasound1_link. Frame ID for ultrasonic sensor 1. Valid when robot_model is a bobac series.
range_2_frame_id	string. Default ultrasound2_link. Frame ID for ultrasonic sensor 2. Valid when robot_model is a bobac series.
range_3_frame_id	string. Default ultrasound3_link. Frame ID for ultrasonic sensor 3. Valid when robot_model is bobac2.

Parameter name	Description
BC_1_frame_id	string. Default bumper_cliff1_link. Frame ID for bumper-cliff sensor 1. Valid when robot_model is bobac2.
BC_2_frame_id	tring. Default bumper_cliff2_link. Frame ID for bumper-cliff sensor 2. Valid when robot_model is bobac2.
BC_3_frame_id	string. Default bumper_cliff3_link. Frame ID for bumper-cliff sensor 3. Valid when robot_model is bobac2.
odom_frame_id	string. Default odom. Frame ID for the odometry coordinate frame.
robot_base_frame_id	string. Default base_footprint. Frame ID for the robot base coordinate frame.
publis_odom	bool. Default false. Whether to publish the odometry topic.
publis_odom_tf	bool. Default false. Whether to publish the odometry tf.
kinematics_mode	int. Chassis kinematic model. 2: Two-wheel differential, 3: Three-wheel omni-directional, 5: Four-wheel Mecanum wheel omni-directional.
wheel_radius	double. Wheel radius. Unit: m.
wheel_separation_x	double. Not used when kinematics_mode is 2. Represents wheel separation when kinematics_mode is 3, and wheel separation in the x-direction when kinematics_mode is 5.
wheel_separation_y	double. Represents wheel separation when kinematics_mode is 2. Not used when kinematics_mode is 3. Represents wheel separation in the y-direction when kinematics_mode is 5.
max_vx	double. Maximum allowed linear velocity in the x-direction. Unit: m/s. It is recommended not to exceed 0.4 m/s.
max_vy	double. Maximum allowed linear velocity in the y-direction. Unit: m/s. It is recommended not to exceed 0.4 m/s.

Parameter name	Description
max_vth	double. Maximum allowed angular velocity. Unit: rad/s. It is recommended not to exceed 0.8 rad/s.

Others

For definitions and functionalities of the classes BaseKinematics and RobotBaseCom used in this package, please refer to their corresponding header files:

BaseKinematics——kinematics.h

RobotBaseCom——rei_robot_base.h

The source code files can be viewed or downloaded from the master branch at:

https://gitee.com/reinovo/rei_robot_base.git

7.1.3 Operating requirements

To successfully run the programs in this package, the following conditions must be met: stable communication with the lower-level controller and normal power operation.

7.1.4 Chassis communication protocol

The chassis communication utilizes the Modbus protocol for data exchange.

Scope

This communication protocol applies to data exchange between the drive control board and the industrial computer, specifying their physical connection, communication link, and application technical specifications.

Referenced standards

Modbus-RTU protocol

Overview

- Byte format

Asynchronous serial communication. Each byte consists of 8 data bits, no parity bit, 1 stop bit(1). Asynchronous communication at 115200 bps. The drive control board address is 01.

- Frame format

Adhere to the standard Modbus-RTU protocol. A typical frame format is as follows:

1. Master sends query command

Start gap	Device address	Function code	Start register address	Number of registers	CRC Checksum	End gap
longer than 3.5 character times	1 byte	1 byte	2 bytes, high byte first	2 bytes, high byte first	2 bytes, high byte first	longer than 3.5 character times

Slave response

Start gap	Device address	Function code	Start register address	Data	CRC Checksum	End gap
longer than 3.5 character times	1 byte	1 byte	2 bytes, high byte first	N byte	2 bytes, high byte first	longer than 3.5 character times

Example: The master reads HoldDataReg[1]. The operation is as follows:

01 03 00 01 00 01 D5 CA

Slave Address | Function Code | Data Address | Number of Registers | CRC Checksum

Upon receiving this data stream, the drive control board calculates a CRC checksum to verify its integrity. If the data is validated, it sends a formatted response back to the master.

Response:

01 03 02 0017 F8 4A

Slave Address | Function Code | Byte Count | Data (2 bytes) | CRC Checksum

2. Master writes to slave (Single register write)

Start gap	Device address	Function code	Start register address	Data	CRC Checksum	End gap
longer than 3.5 character times	1 byte	1 byte	2 bytes, high byte first	N byte	2 bytes, high byte first	longer than 3.5 character times

Example:

01 06 00 01 00 17 98 04

Slave Address | Function Code | Data Address | Data | CRC Checksum

If the local address is 01, the microcontroller receives this data stream, calculates the CRC checksum to verify its integrity. If the data is validated, the result is: HoldDataReg[1] = 0x0017. This completes a write operation by the Modbus master to the slave, achieving successful communication.

NOTE: In Modbus, "coils" and "registers" essentially refer to "bit variables" and "16-bit variables" respectively. The term "coil" originates from the protocol's initial development by Schneider Electric for its PLCs, hence many terms are related to PLC terminology. **This communication specification uses only holding registers; coils are not enabled.** Additionally, regarding the Modbus verification method, the corresponding C language implementation is provided at the end of the document.

3. Application

Read Wheel Speed (for each wheel)

Function: Read the wheel speed command. Returns 4 double values. One double value (8 bytes) is formed by 4 registers (high byte first), corresponding to the speed of wheels 1-4. The unit is revolutions per minute (RPM).

Modbus function code: 04

Register starting address: 30001 (Input registers)

Data length: 16 (Registers)

Data type: double

Frame format:

Device Address	Function Code	Register address high	Register address low	Quantity high	Quantity low	CRC high	CRC low
1	4	30001		16		/	

Drive control board response:

Frame format:

Device Address	Function Code	Quantity high	Quantity low	Motor 1 speed	Motor 2 speed	Motor 3 speed	Motor 4 speed	CRC high	CRC low
1	4	16		/	/	/	/	/	

Data definition:

Wheel 1 Speed: Double-precision data, using 4 input registers, i.e., 8 bytes of data, high byte first, unit is rpm.

Wheel 2 Speed: Double-precision data, using 4 input registers, i.e., 8 bytes of data, high byte first, unit is rpm.

Wheel 3 Speed: Double-precision data, using 4 input registers, i.e., 8 bytes of data, high byte first, unit is rpm.

Wheel 4 Speed: Double-precision data, using 4 input registers, i.e., 8 bytes of data, high byte first, unit is rpm.

Read battery charging status

Function: Determine whether the robot is charging.

Modbus function code: 04

Register starting address: 30017 (Input register)

Data Length: 1

Data Type: 16-bit unsigned integer

Frame format:

Device Address	Function Code	Register address high	Register address low	Quantity high	Quantity low	CRC high	CRC low
1	4	30017		1		/	

Drive control board response:

Frame format:

Device Address	Function Code	Quantity high	Quantity low	Charging status	CRC high	CRC low
1	4	1		0 or 1	/	

Data definition:

Charging status: 16-bit unsigned integer, occupying 1 input register (i.e., 2 bytes of data), high byte first. It indicates whether the battery is currently charging, where 0 represents not charging and 1 represents charging.

Read power supply voltage

Function: Read the value of the power supply voltage.

Modbus function code: 04

Register starting address: 30018 (Input register)

Data length: 1

Data type: 16-bit unsigned integer

Frame format:

Device Address	Function Code	Register address high	Register address low	Quantity high	Quantity low	CRC high	CRC low
1	4	30018		1		/	

Drive control board response:

Frame format:

Device Address	Function Code	Quantity high	Quantity low	Power supply voltage value	CRC high	CRC low
1	4	1		/	/	

Data definition:

Power supply voltage value: 16-bit unsigned integer, occupying 1 input register (i.e., 2 bytes of data), high byte first. It corresponds to the current power supply voltage value, in units of mV.

Read IO Status

Function: Read the status of the extended input IO or the bumper-cliff sensors (when using the bobac2 robot).

Modbus function code: 04

Register starting address: 30019 (Input register)

Data length: 1

Data type: 16-bit unsigned integer

Frame format:

Device Address	Function Code	Quantity high	Quantity low	Quantity high	Quantity low	CRC high	CRC low
1	4	30019		1		/	

Drive control board response:

Frame format:

Device Address	Function Code	Quantity high	Quantity low	IO status	CRC high	CRC low
1	4	1		/	/	

Data definition:

IO status: 16-bit unsigned integer, occupying 1 input register (i.e., 2 bytes of data), high byte first. The lower 4 bits represent the status of the four corresponding channel IOs. The bit status is 1 when the IO input is connected to GND, and 0 when the IO input is floating. The device input IOs are typically connected to sensors with NPN outputs or switch output sensors.

IO status: When the device is a bobac2 robot, the lower 6 bits (bits 0-5) correspond to the IO status of the following sensors, respectively: Cliff 2, Bumper 2, Cliff 3, Bumper 3, Cliff 1, Bumper 1.

Read ultrasonic sensor status

Function: Read the status of the ultrasonic sensors, returning 3 integers corresponding to the distance values (in mm) from ultrasonic sensors 1-3.

Modbus function code: 04

Register starting address: 30020

Data length: 3

Data type: 16-bit unsigned integer

Frame format:

Device Address	Function Code	Quantity high	Quantity low	Quantity high	Quantity low	CRC high	CRC low
1	4	30020		3		/	

Drive control board response:

Frame format:

Device Address	Function Code	Quantity high	Quantity low	Ultrasonic 1-3	CRC high	CRC low
1	4	3		/	/	

Data definition:

Ultrasonic 1-3: Return the distance measured by the corresponding ultrasonic sensor, in millimeters (mm). The data type is a 16-bit unsigned integer. It should be read at 200ms intervals.

Read smoke sensor status

Function: Determine if the smoke sensor has been triggered.

Modbus function code: 04

Register starting address: 30023 (Input register)

Data length: 1

Data Type: 16-bit unsigned integer

Frame format:

Device Address	Function Code	Quantity high	Quantity low	Quantity high	Quantity low	CRC high	CRC low
1	4	30023		1		/	

Drive control board response:

Frame format:

Device Address	Function Code	Quantity high	Quantity low	Smoke status	CRC high	CRC low
1	4	1		0 or 1	/	

Data definition:

Smoke sensor status: 16-bit unsigned integer, occupying 1 input register (i.e., 2 bytes of data), high byte first. It indicates the current status of the smoke sensor, where 0 represents Normal and 1 represents Triggered.

Read temperature and humidity sensor status

Function: Read the data from the temperature and humidity sensor.

Modbus function code: 04

Register starting address: 30024 (Input register)

Data length: 4

Data type: float

Frame format:

Device Address	Function Code	Register address high	Register address low	Quantity high	Quantity low	CRC high	CRC low
1	4	30024		4		/	

Drive control board response:

Frame format:

Device Address	Function Code	Quantity high	Quantity low	Temperature	Humidity	CRC high	CRC low
1	4	4		/	/	/	

Data definition:

Temperature status: Float data type, occupying 2 input registers (i.e., 4 bytes of data), high byte first. It corresponds to the current temperature value in degrees Celsius (°C).

Humidity status: Float data type, occupying 2 input registers (i.e., 4 bytes of data), high byte first. It corresponds to the current humidity value, expressed as a percentage of relative humidity. Relative humidity refers to the ratio of the partial pressure of water vapor in the air to the saturated vapor pressure of water at the same temperature.

Set Individual Wheel Speed

Function: Set the speed command for the wheels. Write 4 double-precision values, where each value is composed of 4 registers (8 bytes), with the high byte first. These values correspond to the rotational speed of wheels 1-4, in revolutions per minute (rpm).

Modbus function code: 16

Register starting address: 40001 (Holding register)

Data length: 16

Data type: double

Frame format:

Device address	Function code	Register address high	Register address low	Quantity high	Quantity low	Motor 1 speed	Motor 2 speed	Motor 3 speed	Motor 4 speed	CRC high	CRC low
1	16	40001		16							

Data definition:

Wheel 1 Speed: Double-precision data, occupying 4 input registers (i.e., 8 bytes of data), high byte first. Unit is rpm.

Wheel 2 Speed: Double-precision data, occupying 4 input registers (i.e., 8 bytes of data), high byte first. Unit is rpm.

Wheel 3 Speed: Double-precision data, occupying 4 h input registers (i.e., 8 bytes of data), high byte first. Unit is rpm.

Wheel 4 Speed: Double-precision data, occupying 4 input registers (i.e., 8 bytes of data), high byte first. Unit is rpm.

Set buzzer playback

Function: Play the buzzer by writing an unsigned integer corresponding to the delay time for the buzzer to sound.

Modbus function code: 06

Register starting address: 40017 (Holding register)

Data length: 1

Data type: 16-bit unsigned integer

Frame format:

Device address	Function code	Register address high	Register address low	Playback duration	CRC high	CRC low
1	6	40017				

Data definition:

Playback time: 16-bit unsigned integer, occupying 1 input register (i.e., 2 bytes of data), high byte first. This corresponds to the buzzer delay time, with a unit of 100ms.

Set drive control board relay output

Function: Configure the relay output of the drive control board.

Modbus function code: 06

Register starting address: 40018 (holding register)

Data length: 1

Data type: 16-bit unsigned integer

Frame format:

Device address	Function code	Register address high	Register address low	Relay status	CRC high	CRC low
1	6	40018		0 or 1		

Data definition:

Relay status: 16-bit unsigned integer accepting only input 0 or 1, 1 input register (2 bytes of data), high byte first, where 1 indicates relay closure.

Set IO output

Function: Configure extended IO output to control the output level of 7 extended IO channels and one relay output.

Modbus function code: 06

Register starting address: 40019 (holding register)

Data length: 1

Data type: 16-bit unsigned integer

Frame format:

Device address	Function code	Register address high	Register address low	IO status	CRC high	CRC low
1	6	40019				

Data definition:

IO status: 16-bit unsigned integer, 1 input register (2 bytes of data), high byte first.

The lower 7 bits represent the output level of the extended IO, which is NPN common-ground output.

Set extended motor output

Function: Set extended motor output to control the stop, forward rotation, and reverse rotation of the extended motor.

Modbus function code: 06

Starting register address: 40020 (holding register)

Data length: 1

Data type: 16-bit unsigned integer

Frame format:

Device address	Function code	Register address high	Register address low	Motor status	CRC high	CRC low
1	6	40020		0-3		

Data definition:

Motor status: A 16-bit unsigned integer that only accepts the values 0, 1, or 3, occupying 1 holding register (i.e., 2 bytes of data), high byte first.

0: Motor stop; 1: Motor forward; 2: Motor reverse; Other values are invalid.

Appendix:

The C language function for CRC calculation is as follows:


```
/*********************************************
->Function name: CRCVerify()
->Function: Verifies the CRC of received and transmitted commands
-> Input Parameters: Verify the pointer to data
Number of bytes to verify
-> Return value: None
*********************************************/
WORDCRCVerify(BYTE*pMsg,WORDusDataLen)
{
 BYTEucCRCHi=0xFF; /*highbyteofCRCinitialized*/
 BYTEucCRCLo=0xFF; /*lowbyteofCRCinitialized*/
 WORDuIndex=0; /*willindexintoCRClookuptable*/
 while(usDataLen--) /*passthroughmessagebuffer*/
 {
 uIndex=ucCRCHi^*pMsg++; /*calculatetheCRC*/
 ucCRCHi=ucCRCLo^auchCRCHi[uIndex];
 ucCRCLo=auchCRCLo[uIndex];
 }
 return(ucCRCHi<<8|ucCRCLo);
}/*The return value is the check value for the RTU message frame */
staticBYTEauchCRCHi[]={
0x00,0xC1,0x81,0x40,0x01,0xC0,0x80,0x41,0x01,0xC0,0x80,0x41,0x00,0xC1,0x81,0x40
,0x01,0xC0,0x80,0x41,0x00,0xC1,0x81,0x40,0x00,0xC1,0x81,0x40,0x01,0xC0,0x80,0x4
1,0x01,0xC0,0x80,0x41,0x00,0xC1,0x81,0x40,0x00,0xC1,0x81,0x40,0x01,0xC0,0x80,0x
41,0x00,0xC1,0x81,0x40,0x01,0xC0,0x80,0x41,0x01,0xC0,0x80,0x41,0x00,0xC1,0x81,0
x40,0x01,0xC0,0x80,0x41,0x00,0xC1,0x81,0x40,0x00,0xC1,0x81,0x40,0x01,0xC0,0x80,
0x41,0x00,0xC1,0x81,0x40,0x01,0xC0,0x80,0x41,0x01,0xC0,0x80,0x41,0x00,0xC1,0x81
,0x40,0x00,0xC1,0x81,0x40,0x01,0xC0,0x80,0x41,0x01,0xC0,0x80,0x41,0x00,0xC1,0x8
1,0x40,0x01,0xC0,0x80,0x41,0x00,0xC1,0x81,0x40,0x00,0xC1,0x81,0x40,0x01,0xC0,0x
80,0x41,0x01,0xC0,0x80,0x41,0x00,0xC1,0x81,0x40,0x00,0xC1,0x81,0x40,0x01,0xC0,0
x80,0x41,0x00,0xC1,0x81,0x40,0x01,0xC0,0x80,0x41,0x01,0xC0,0x80,0x41,0x00,0xC1,
0x81,0x40,0x00,0xC1,0x81,0x40,0x01,0xC0,0x80,0x41,0x01,0xC0,0x80,0x41,0x00,0xC1
,0x81,0x40,0x01,0xC0,0x80,0x41,0x00,0xC1,0x81,0x40,0x00,0xC1,0x81,0x40,0x01,0xC
0,0x80,0x41,0x00,0xC1,0x81,0x40,0x01,0xC0,0x80,0x41,0x01,0xC0,0x80,0x41,0x00,0x
C1,0x81,0x40,0x01,0xC0,0x80,0x41,0x00,0xC1,0x81,0x40,0x00,0xC1,0x81,0x40,0x01,0
xC0,0x80,0x41,0x01,0xC0,0x80,0x41,0x00,0xC1,0x81,0x40,0x00,0xC1,0x81,0x40,0x01,
0xC0,0x80,0x41,0x00,0xC1,0x81,0x40,0x01,0xC0,0x80,0x41,0x01,0xC0,0x80,0x41,0x00
,0xC1,0x81,0x40};
staticcharauchCRCLo[]={
```

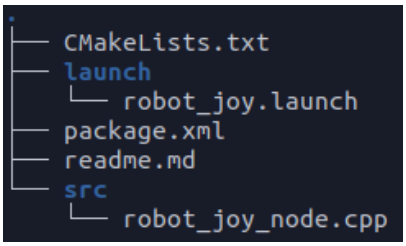
```
0x00,0xC0,0xC1,0x01,0xC3,0x03,0x02,0xC2,0xC6,0x06,0x07,0xC7,0x05,0xC5,0xC4,0x
04,0xCC,0x0C,0x0D,0xCD,0x0F,0xCF,0xCE,0x0E,0x0A,0xCA,0xCB,0x0B,0xC9,0x09,0x
08,0xC8,0xD8,0x18,0x19,0xD9,0x1B,0xDB,0xDA,0x1A,0x1E,0xDE,0xDF,0x1F,0xDD,0x1
D,0x1C,0xDC,0x14,0xD4,0xD5,0x15,0xD7,0x17,0x16,0xD6,0xD2,0x12,0x13,0xD3,0x11,
0xD1,0xD0,0x10,0xF0,0x30,0x31,0xF1,0x33,0xF3,0xF2,0x32,0x36,0xF6,0xF7,0x37,0xF5
,0x35,0x34,0xF4,0x3C,0xFC,0xFD,0x3D,0xFF,0x3F,0x3E,0xFE,0xFA,0x3A,0x3B,0xFB,0x
39,0xF9,0xF8,0x38,0x28,0xE8,0xE9,0x29,0xEB,0x2B,0x2A,0xEA,0xEE,0x2E,0x2F,0xEF,
0x2D,0xED,0xEC,0x2C,0xE4,0x24,0x25,0xE5,0x27,0xE7,0xE6,0x26,0x22,0xE2,0xE3,0x
23,0xE1,0x21,0x20,0xE0,0xA0,0x60,0x61,0xA1,0x63,0xA3,0xA2,0x62,0x66,0xA6,0xA7,0
x67,0xA5,0x65,0x64,0xA4,0x6C,0xAC,0xAD,0x6D,0xAF,0x6F,0x6E,0xAE,0xAA,0x6A,0x6
B,0xAB,0x69,0xA9,0xA8,0x68,0x78,0xB8,0xB9,0x79,0xBB,0x7B,0x7A,0xBA,0xBE,0x7E,
0x7F,0xBF,0x7D,0xBD,0xBC,0x7C,0xB4,0x74,0x75,0xB5,0x77,0xB7,0xB6,0x76,0x72,0x
B2,0xB3,0x73,0xB1,0x71,0x70,0xB0,0x50,0x90,0x91,0x51,0x93,0x53,0x52,0x92,0x96,0
x56,0x57,0x97,0x55,0x95,0x94,0x54,0x9C,0x5C,0x5D,0x9D,0x5F,0x9F,0x9E,0x5E,0x5A,
0x9A,0x9B,0x5B,0x99,0x59,0x58,0x98,0x88,0x48,0x49,0x89,0x4B,0x8B,0x8A,0x4A,0x4
E,0x8E,0x8F,0x4F,0x8D,0x4D,0x4C,0x8C,0x44,0x84,0x85,0x45,0x87,0x47,0x46,0x86,0x
82,0x42,0x43,0x83,0x41,0x81,0x80,0x40};
```


7.2 robot_joy

Function: Joystick speed control

Node name: robot_joy_node

7.2.1 File structure



7.2.2 ROS API

Subscribed Topic

Topic name (Message type)	Function	Parameter description
joy(sensor_msgs/Joy)	Joystick key values	

Published Topic

Topic name (Message type)	Function	Parameter description
cmd_vel(geometry_msgs/Twist)	Speed control	http://docs.ros.org/en/melodic/api/geometry_msgs/html/msg/Twist.html

Configuration parameters

Parameter name	Parameter description
axis_linear_x	int type, default value 1, axis number for x-direction control keys (forward/backward)
axis_linear_y	int type, default value 0, axis number for y-direction control keys (left/right movement)
axis_angular	int type, default value 3, axis number for yaw angle control keys (left/right turn)

axis_accelerator	int type, default value 5, axis number for acceleration key
axis_deceleration	int type, default value 2, axis number for deceleration key
button_max_linear_increase	int type, default value 0, button for increasing maximum linear velocity
button_max_angular_increase	int type, default value 2, button for increasing maximum angular velocity
button_max_linear_reduce	int type, default value 1, button for decreasing maximum linear velocity
button_max_angular_reduce	int type, default value 3, button for decreasing minimum angular velocity
linear_vel_max	double type, default value 0.8, maximum achievable linear velocity (velocity will be limited to this value regardless of acceleration)
angular_vel_max	double type, default value 3.0, maximum achievable angular velocity (velocity will be limited to this value regardless of acceleration)
linear_vel	double type, default value 0.3, initial maximum linear velocity
angular_vel	double type, default value 0.6, initial maximum angular velocity

7.2.3 Operating requirements

To successfully run the programs in this package, you need to:

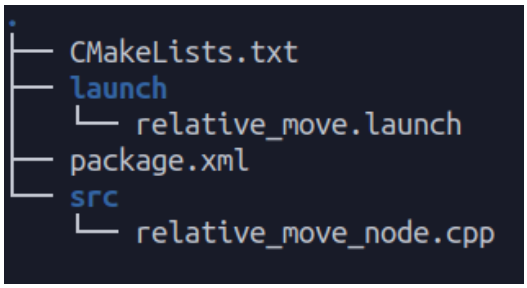
Insert the joystick receiver and turn on the joystick power.

7.3 relative_move

Function: Chassis relative movement

Node name: relative_move_node

7.3.1 File structure



7.3.2 ROS API

Subscribed Topic

Topic name (message type)	Function	Parameter description
scan(sensor_msgs/LaserScan)	2D LiDAR Data	http://docs.ros.org/en/melodic/api/sensor_msgs/html/msg/LaserScan.html

Published Topic

Topic name (message type)	Function	Parameter description
cmd_vel(geometry_msgs/Twist)	Speed control	http://docs.ros.org/en/melodic/api/geometry_msgs/html/msg/Twist.html

Services

Service name (Service type)	Function	Parameter description
relative_move(relative_move/SetRelativeMove)	Robot relative movement	geometry_msgs/Pose2D goal: The direction and distance for the movement. string global_frame: The global reference frame (typically /odom, or set to /map when used with navigation). bool finishStopObstacle : Whether to directly end when stopping due to obstacles (generally

		occurs after obstacle avoidance is enabled) --- bool success string message
stop_rel_move	Stop relative movement	std_sr vs/Empty

Configuration parameters

Parameter name	Parameter description	Default value
base_frame	string type, robot base coordinate frame.	base_footprint
odom_model_type	string type, chassis motion model; Differential: diff Omnidirectional: omni	diff
x_p	double type, PID adjustment in the x-direction	0.5
x_i		0.0
x_d		0.2
y_p	double type, PID adjustment in the y-direction	0.5
y_i		0.0
y_d		0.2
theta_p	double type, rotational PID adjustment	1.0
theta_i		0.0
theta_d		0.3

7.3.3 Operating conditions

To successfully run the programs in this package, the following is required:

Launch bobac3_base.launch from rei_robot_base, with the parameter publish_odom set to true.

7.4 ar_track_alvar

Function: AR marker recognition

7.4.1 ROS API

Offical document: http://wiki.ros.org/ar_track_alvar

Configuration parameters

Two new configuration parameters have been added.

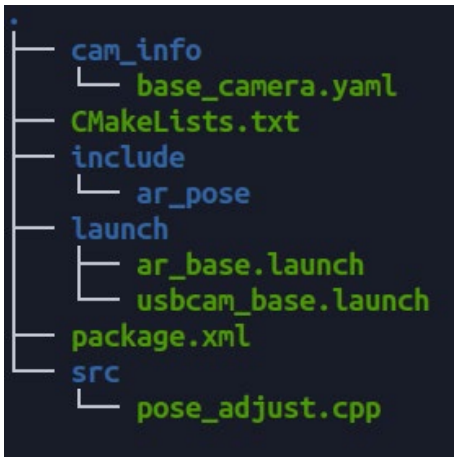
Parameter	Parameter description
special_detect	bool type, default value false. Determine whether to only recognize AR marker with specified IDs.
special_id	int type, default value 0. AR marker ID. Effective only when special_detect is true.

7.5 ar_pose

Function: AR marker tracking

Node name: pose_adjust

7.5.1 File structure



Among them,

base_camera.yaml is the camera intrinsic parameter calibration file

usbcam_base.launch is the launch file for visual positioning camera

ar_base.launch is the master launch file for this function, which also publishes a world——>usb_cam TF transformation.

7.5.2 ROS API

Subscribed Topic

Topic name (message type)	Function	Parameter description
ar_pose_marker(ar_track_alvar_msgs/AlvarMarkers)	AR marker information	Header header: Message header AlvarMarker[] markers：Information of all detected AR markers

Published Topic

Topic name (message type)	Function	Parameter description
cmd_vel(geometry_msgs/Twist)	Speed control	http://docs.ros.org/en/melodic/api/geometry_msgs/html/msg/Twist.html

Services

Service name (service type)	Function	Parameter description
ar_track(bobac3_msgs/track)	AR marker tracking	int8 ar_id: AR ID being tracked. float32 goal_dist: The tracking distance --- string message bool success

7.5.3 Operating requirements

To run this package successfully, you must:

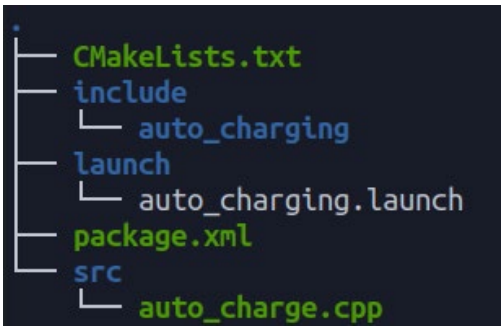
Install or compile the ar_track_alvar package

7.6 auto_charging

Function: Auto charging

Node name: auto_charge

7.6.1 File structure



Among them,

The auto_charging.launch file is the launch file for the automatic charging feature and it simultaneously initiates the ar_base.launch.

7.6.2 ROS API

Services

Service name (service type)	Function	Parameter description
goto_charge(auto_charging/SetCharge)	Auto charging	bool nav: whether to navigate to the charging location bool ar_track: whether the robot is aligned with the charging docker int8 ar_id: The id of AR marker --- string message bool success

Clients

Service name (service type)	Function	Parameter description
ar_track(ar_pose/track)	AR marker tracking	int8 ar_id: AR id being tracked float32 goal_dist: The tracking distance --- string message bool success

Actionlib Clients

move_base: Autonomous navigation action

Configuration parameters

Parameter name	Parameter description
anchor_x	double type, default 0, charging point coordinate x
anchor_y	double type, default 0, charging point coordinate y
anchor_theta	double type, default 0, charging point yaw angle

7.6.3 Operating conditions

To successfully run the programs in this package, the following is required:

The ar_pose.launch in ar_pose must be able to run successfully.

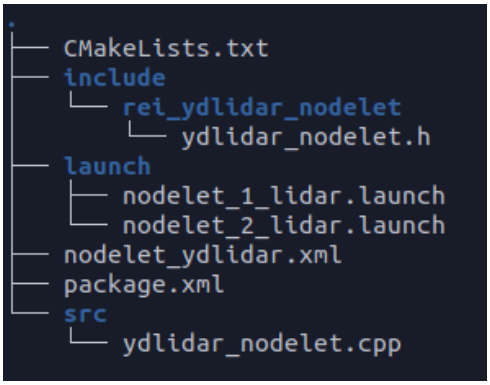
7.7 rei_ydlidar_nodelet

Function: Acquiring 2D LiDAR data

This package utilizes nodelet to reduce the time loss caused by data transmission during dual LiDAR data fusion.

Plugin name: rei_ydlidar/ReiYdlidars

7.7.1 File structure



Among them,

nodelet_1_lidar.launch: Start the front LiDAR

nodelet_2_lidar.launch: Start both the front and rear LiDARs

7.7.2 ROS API

Published Topic

Topic name (message type)	Function	Parameter description
~/scan(sensor_msgs/LaserScan)	LiDAR topics	http://docs.ros.org/en/melodic/api/sensor_msgs/html/msg/LaserScan.html

Configuration parameters

Parameter name	Parameter description
Port	string type, default value /dev/ydlidar, LiDAR serial port number or IP address
ignore_array	string type, default value " ", laser angles to be ignored
frame_id	string type, default value laser_frame, LiDAR coordinate frame
baudrate	int type, default value 230400, baud rate or network port number

Parameter name	Parameter description
lidar_type	int type, default value 1, LiDAR type, 0 TYPE_TOF, 1 TYPE_TRIANGLE, 2 TYPE_TOF_NET
device_type	int type, default value 0, device communication type, 0 YDLIDAR_TYPE_SERIAL, 1 YDLIDAR_TYPE_TCP, 2 YDLIDAR_TYPE_UDP
sample_rate	int type, default value 9, measurement frequency, unit: kilo-times/second
abnormal_check_count	int type, default value 4, defines the number of retry attempts for handling abnormal data during device startup
resolution_fixed	bool type, default value true, fixed angular resolution
reversion	bool type, default value true, invert LiDAR
inverted	bool type, default value true, reverse LiDAR, false for clockwise, true for counterclockwise
auto_reconnect	bool type, default true, auto-reconnect to the LiDAR
isSingleChannel	bool type, default false, whether the LiDAR is single-channel
intensity	bool type, default false, whether the LiDAR supports intensity data
support_motor_dtr	bool type, default false, whether the LiDAR can be started/stopped via Serial DTR
invalid_range_is_inf	bool type, default false, whether to set invalid distances as inf
angle_max	float type, default 180.0, maximum valid angle in degrees
angle_min	float type, default -180.0, minimum valid angle in degrees
range_max	float type, default 16.0, maximum valid range in meters

Parameter name	Parameter description
range_min	float type, default 0.1, minimum valid range in meters
frequency	float type, default 10.0, scanning frequency in Hz

7.7.3 Operating conditions

To successfully run the programs in this package, the following condition must be met:

The LiDAR must be functioning properly and communication must be stable.

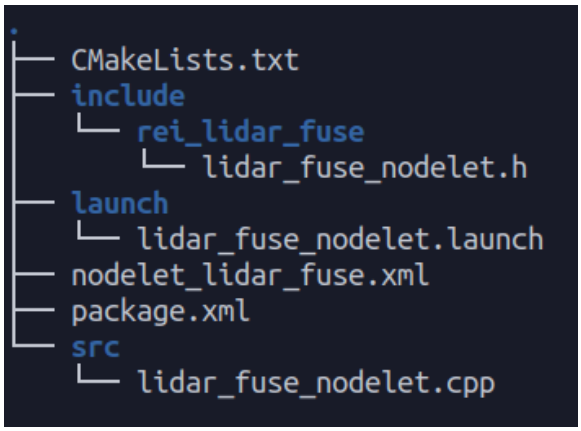
7.8 rei_lidar_fuse

Function: Dual LiDAR fusion

Subscribe to LiDAR data using the nodelet method.

Plugin name: rei_lidar_fuse/LidarFuse

7.8.1 File structure



7.8.2 ROS API

Subscribed Topic

Topic name (message type)	Function	Parameter description
<~front_topic>(sensor_msgs/LaserScan)	Front LiDAR topic	http://docs.ros.org/en/melodic/api/sensor_msgs/html/msg/LaserScan.html
<~rear_topic>(sensor_msgs/LaserScan)	Rear LiDAR topic	/

Published Topic

Topic name (message type)	Function	Parameter description
fuse_scan(sensor_msgs/LaserScan)	Fused LiDAR data	http://docs.ros.org/en/melodic/api/sensor_msgs/html/msg/LaserScan.html

Configuration parameters

Parameter name	Parameter description
frame_id	string type, default "laser_link", coordinate frame of the fused LiDAR data
front_topic	string type, default /front_lidar/scan, topic of the front LiDAR
rear_topic	string type, default /rear_lidar/scan, topic of the rear LiDAR
angle_max	float type, default 3.1415, maximum valid angle in radians
angle_min	float type, default -3.1415, minimum valid angle in radians
range_max	float type, default 7.0, maximum valid range in meters
range_min	float type, default 0.25, minimum valid range in meters

angle_increment	float type, default 0.014, angular resolution in radians
scan_time	ffloat type, default 0.02, scan time interval in seconds

7.8.3 Operating requirements

To successfully run the programs in this package, the following condition must be met:

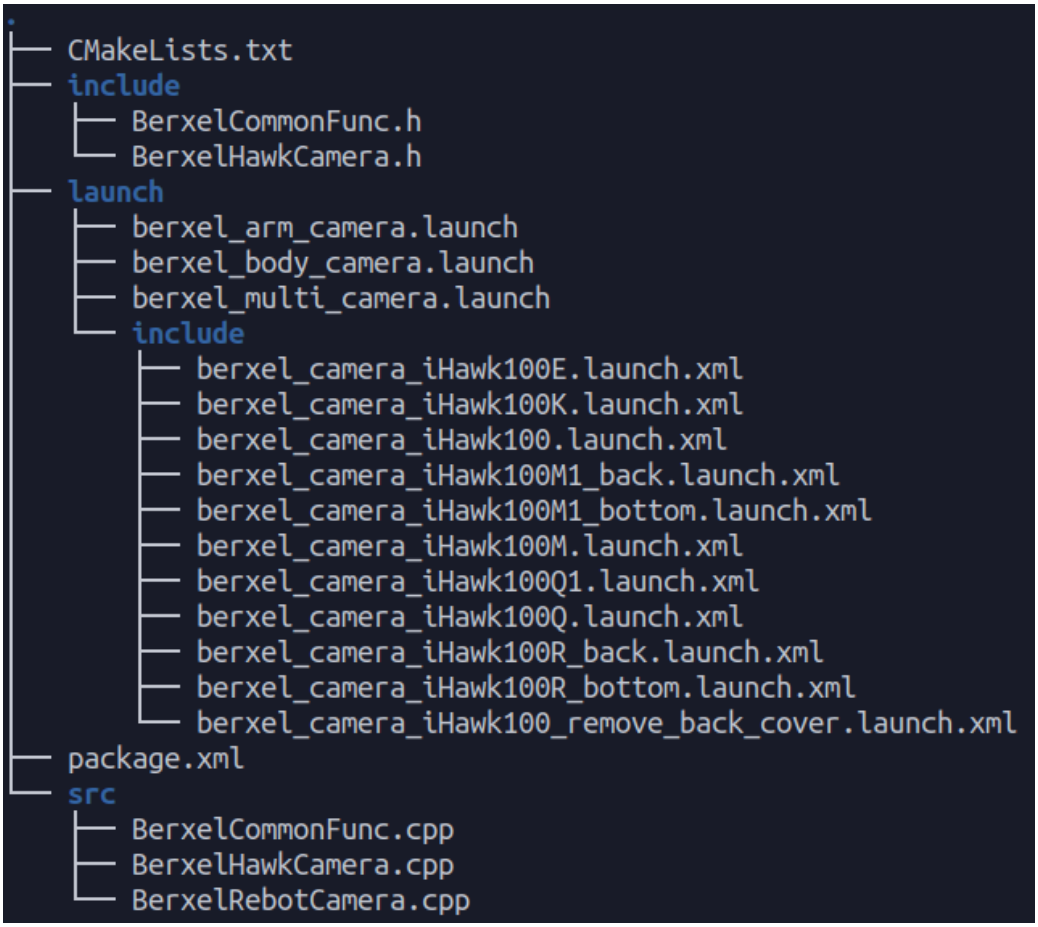
None.

7.9 berxel_camera

Function: Depth camera driver package

Node name: berxel_camera

7.9.1 File structure



Among them,

berxel_arm_camera.launch: Arm-mounted depth camera launch file

berxel_body_camera.launch: Chassis-mounted depth camera launch file

berxel_multi_camera.launch: Dual depth camera launch file

7.9.2 ROS API

Published Topic

Topic name (message type)	Function	Parameter description
~/rgb/camera_info(sensor_msgs/CameraInfo)	Color camera intrinsics	http://docs.ros.org/en/melodic/api/sensor_msgs/html/msg/CameraInfo.html
~/depth/camera_info(sensor_msgs/CameraInfo)	Depth camera intrinsics	/
~/ir/camera_info(sensor_msgs/CameraInfo)	Infrared camera intrinsics	/
~/rgb/rgb_raw(sensor_msgs/Image)	Color image	http://docs.ros.org/en/melodic/api/sensor_msgs/html/msg/Image.html
~/depth/depth_raw(sensor_msgs/Image)	Depth image	http://docs.ros.org/en/melodic/api/sensor_msgs/html/msg/Image.html
~/ir/ir_raw(sensor_msgs/Image)	Infrared image	http://docs.ros.org/en/melodic/api/sensor_msgs/html/msg/Image.html
~/berxel_cloudpoint(sensor_msgs/PointCloud2)	Point cloud data	http://docs.ros.org/en/melodic/api/sensor_msgs/html/msg/PointCloud2.html

Configuration parameters

Parameter name	Parameter description
stream_flag	int type, default 1, stream mode, 1 single stream, 2 mixed stream, 3 HD mixed mode (infrared not supported in HD

	mode), 4 QVGA
stream_type	int type, default 1, stream type, 1 color, 2 depth, 3 color+depth, 4 infrared with flood illumination, 5 color+infrared, 6 depth+infrared
color_width	int type, default 400, color stream resolution
color_height	int type, default 640, color stream resolution
depth_width	int type, default 400, depth stream resolution
depth_height	int type, default 640, depth stream resolution
ir_width	int type, default 400, infrared stream resolution
ir_height	int type, default 640, infrared stream resolution
depth_fps	int type, default 640, depth frame rate (VGA and QVGA support 5, 10, 15, 20, 25, 30; HD mode only supports 10)
Registration	int type, default 0, registration switch: 0: off, 1: on
publish_depth_original_raw	int type, default 0, depth data publishing mode: 0: depth, 1: depth_original_raw
publish_cloud_point	int type, default 0, point cloud publishing switch: 0: off, 1: on
color_cloud_point	int type, default 0, colored point cloud: 0: off, 1: on
ordered_cloud	int type, default 0, point cloud type: 0: unorganized point cloud, 1: organized point cloud
time_stamp	int type, default 0, timestamp type: 0: use device timestamp, 1: use ROS timestamp

NOTE: When the stream_flag is set to HD mode, the color and depth resolution of the depth camera used on this robot must be set to 1280×800.

7.9.3 Operating requirements

To successfully run the programs in this package, the following conditions must be met:

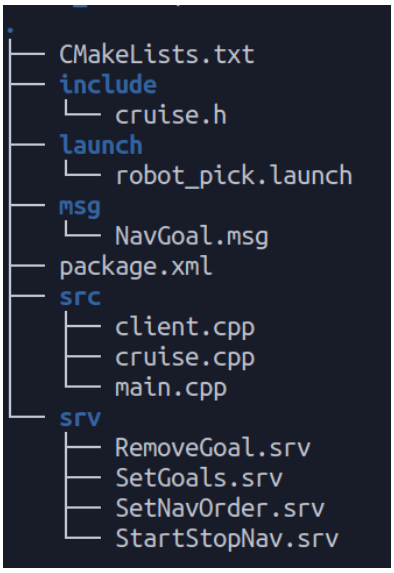
- The depth camera must be functioning properly and communication must be stable.
- It cannot be launched simultaneously with the LiDAR.

7.10 robot_cruise

Function: Picking task scheduling

Node names: cruise_node, cruise_client

7.10.1 File structure



The client.cpp is a client used to invoke various services in cruise_node, designed for adding and configuring picking tasks.

7.10.2 ROS API

Published Topic(cruise_node)

Topic name (message type)	Function	Parameter description
goals_markers(visualization_msgs/MarkerArray)	Visualize target points in RViz	http://docs.ros.org/en/melodic/api/visualization_msgs/html/msg/MarkerArray.html

Services(cruise_node)

Service name (service type)	Function	Parameter description
set_goals(robot_cruise/SetGoals)	Set target point	string mode: Target point setting mode, insert: add new target points, reset: delete previous target points and then add new target points robot_cruise/NavGoal[] goals: Information of the target points to be added (string point_name: Navigation point name int8 level: Navigation point priority, 0: if navigation to this point fails, skip and continue to the next point, 1: if navigation to this point fails, stop the entire task geometry_msgs/PoseStamped target_pose: Coordinate information of the navigation point) --- bool isSuccess
start_stop_nav(robot_cruise/StartStopNav)	Start, pause, stop task	string mode: Action mode, start: begin, pause: suspend, stop: terminate int8 level: Command priority, 0: wait for the previous task to complete before executing this command, 1: execute this command immediately --- bool isSuccess
set_goals_order(robot_cruise/SetNavOrder)	Set navigation point execution order	int16 loop_time: Number of cycles, 0 indicates infinite looping string[] goal_order: Navigation point

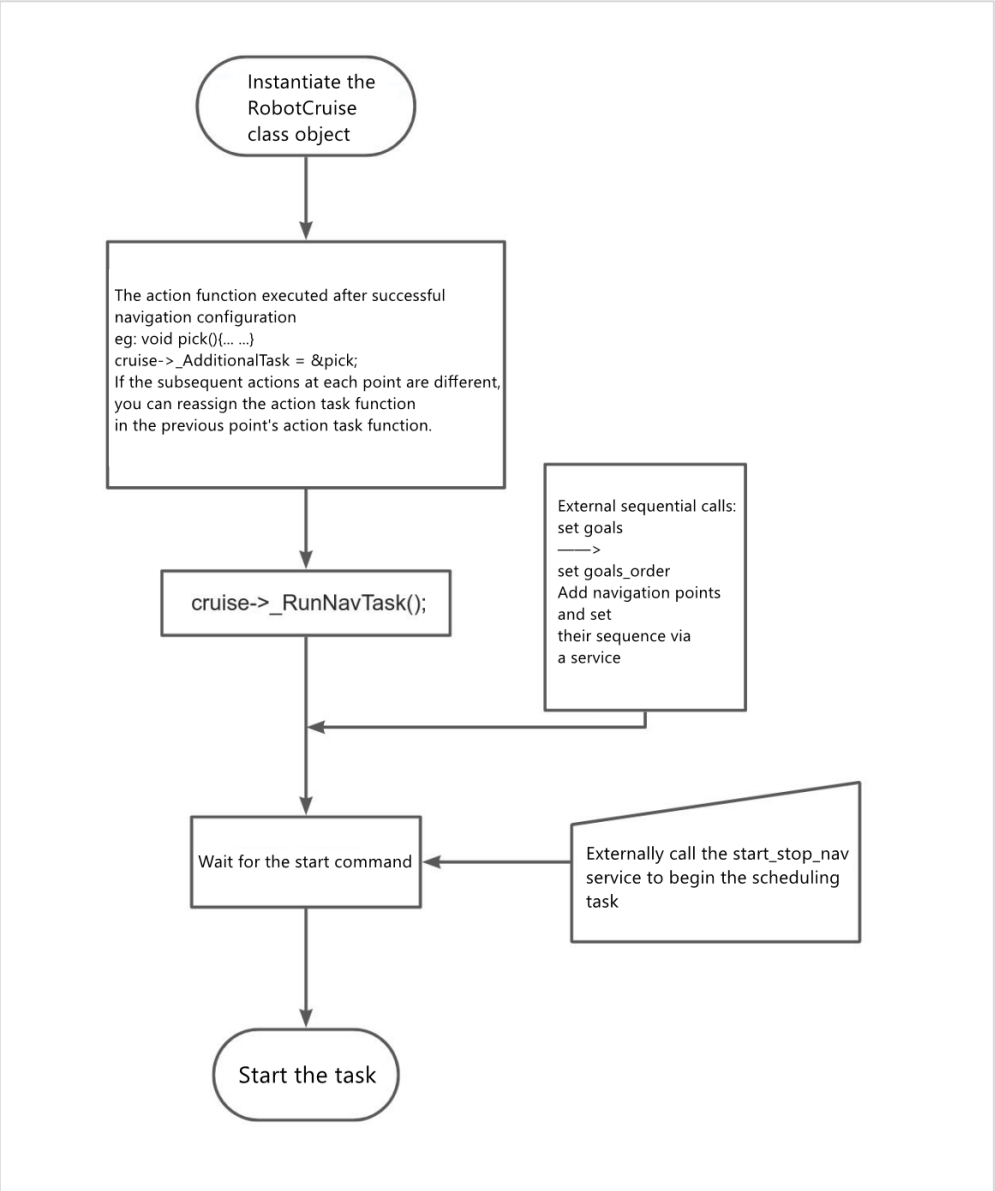
Service name (service type)	Function	Parameter description
		sequence --- bool isSuccess
remove_goalsr(robot_cruise/RemoveGoal)	Delete navigation points	string mode: Deletion mode, all: delete all, not_all: delete specific points string[] goal_names: Names of navigation points to be deleted, valid when mode is not_all --- bool isSuccess

Others

This node offers sophisticated functionality designed for diverse navigation scheduling tasks. By mastering the RobotCruise class within this node, you can subsequently configure multi-point navigation missions according to your specific requirements.

RobotCruise class

Invocation logic:



For detailed source code explanations, please refer to: https://gitee.com/reinovo/robot_cruise.git.

8. Precautions and Maintenance

The Bobac3 robot is a mobile service robot designed to ensure continuous energy supply, ensuring its uninterrupted operation. It features a charging interface, and the battery typically lasts up to 4 hours. When not in use, press the emergency stop button and use a power adapter or a mobile charging station to recharge the robot. The adapter's indicator light has two states: red and green. A red light indicates that the robot is charging, while a green light signifies that the battery is fully charged. However, avoid overcharging the robot when it is fully charged. Keep the robot powered when idle and charge it regularly. When not in use, try to charge it once a month or twice a month. Below are the usage precautions.

1. When using, please handle with care, especially when removing the module, be careful not to pull the internal wires, and ensure that the ground is clean for robot use.
2. Do not place other objects or shielding on the bobac3 robot.
3. Do not change or plug and unplug the lines privately, and check whether the external sensor and receiver connection lines are inserted into the control machine.
4. Check whether the emergency stop button is in normal condition.
5. Check if the screws are loose.
6. Check whether there is any foreign matter on each sensor, especially metal foreign matter is prohibited.
7. Ensure that there is no obstacle on the radar to ensure the normal operation of the radar.
8. Please handle with care during transportation.
9. When using the controller to control the robot, do not collide with sharp or uneven objects
10. Use the human body following laboratory to choose a place where the foreground and background are clearly distinguished as far as possible, control the robot, please follow the control gesture rules (surrender gesture)
11. When using voice-related features, make sure the robot is connected to the Internet
12. Keep a proper distance from the robot when it is moving
13. In case of special circumstances, stop the robot movement and press the emergency stop button directly. After turning off the program, the emergency stop button can be suspended.
14. After long-term idling, please charge the robot before use. If not used, please keep the battery sufficient for storage. Do not use non-compatible adapter to charge the robot.

15. Do not connect the charging port to the power adapter.

9. Appendix

Commonly used commands in Ubuntu system:

- sudo: Authorize
- rm: Delete
- mkdir: Create a directory
- touch: Create a file
- cat: View and concatenate file contents
- ps: Display status of running processes
- grep: Search for specified strings
- kill: Terminate processes
- nohup: Run processes in the background
- ls command: ls -a displays all files and directories, including hidden ones
- pwd command: Used to show the current path
- cd command: Used to change the current directory
- apt-get update command: Get the latest package information
- apt-get upgrade command: Update the software package
- apt-get dist-upgrade command: Update the software package

These commands are highly useful in the Ubuntu system, helping you manage and utilize your system more efficiently.

How to use Vim editor

1. Three modes in Vim:
   - Command mode: Control the cursor, perform operations like copy, paste, delete, and query files.
   - Edit mode: Enter and modify text.
   - Last-Line mode: Save and exit documents, and configure the editing environment.

2. Press a, A, or i in Vim's Command Mode to switch to Edit mode.
3. In command mode, actions like deletion, copying, and pasting are available, but direct text editing is disabled. Pressing any of the keys i, I, o, O, a, A, r, or R switches to Edit mode. In Linux systems, when pressing these keys, "INSERT" or "REPLACE" will appear at the bottom-left of the screen to indicate active editing. Press 『Esc』 to return to command mode.
4. Pressing “I” in Command mode keeps the cursor at its current position, inserting text before the cursor. Pressing “a” moves the cursor one position forward and inserts text before it.
5. In command mode, pressing "o" allows you to enter edit mode and start editing on a new line. Regardless of the current cursor position, it will automatically jump to the next line and insert a new line for editing.
6. In Last-Line Mode, use “wq” or “Shift + zz”” to save and exit. To force quit without saving, enter “q!” in Last-Line Mode.

