DIABLO ROS2

语言：**English** / **中文**

The diablo sdk based on serial communication, you can get started quickly through `ros2`. If you want to develop without ROS, you can also modify `CMakeLists` in ROS to compile only the source code.We will constantly update the package of `ros2`, hoping to be helpful for your robot development.

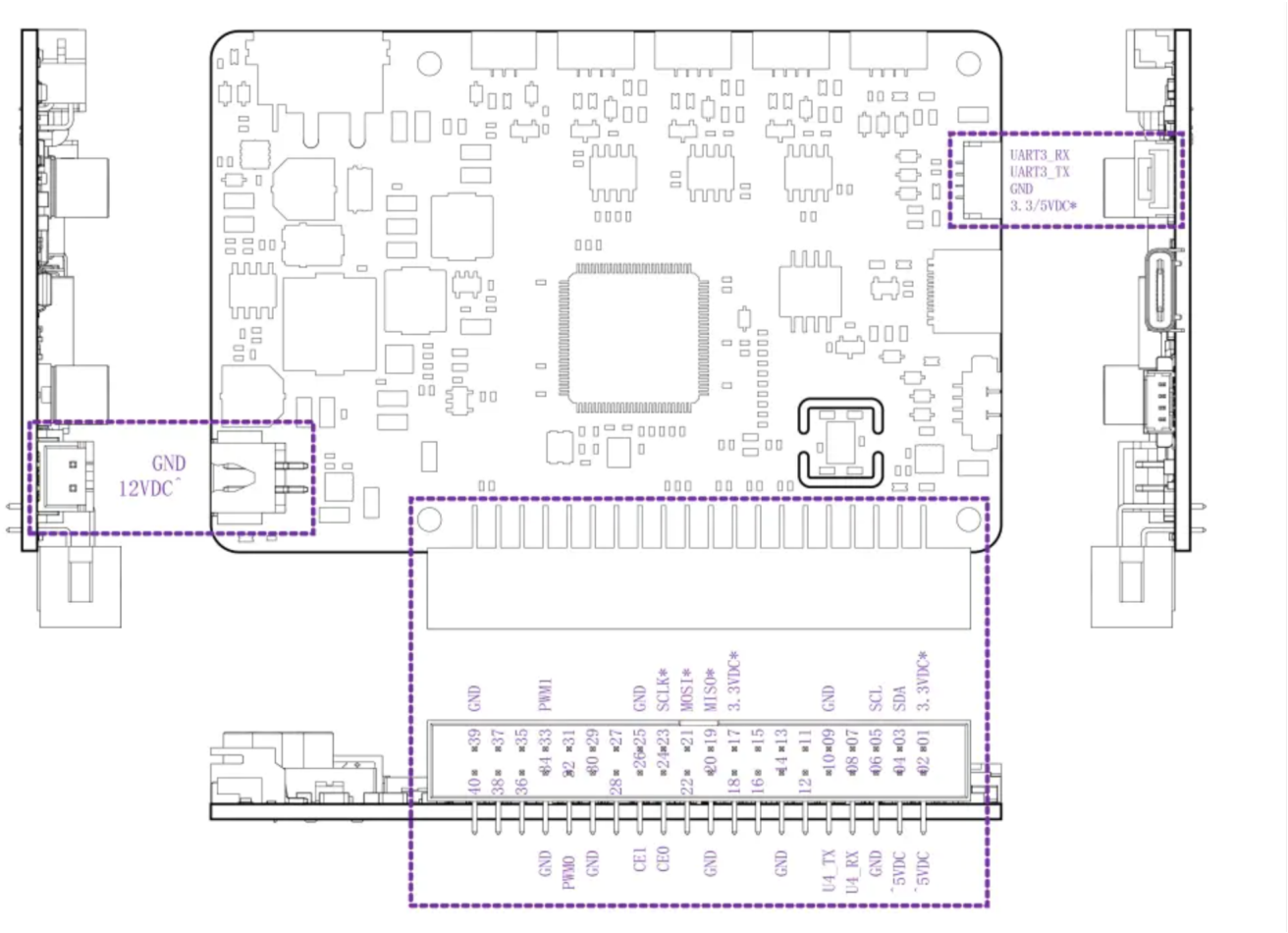
Basic Information

- `X3pi` The default user is `root` and the password `root`
- `Raspberry pi` The default user is `diablo` and the password `diablo123`
- `X3pi` The default serial port number of IO is `/dev/ttyS3`

  > `Raspberry pi` The default serial port number of IO is `/dev/ttyAMA0`
  >
  > You can switch hardware by modifying Hal.init("/dev/ttyS3") and recompiling

- `ROS_DOMAIN_ID=5` , You can use `export ROS_ DOMAIN_ Id=5` connect and control the `Diablo` function node in the LAN.

Main control pin description



Installation

Installation method	Supported platform[s]	Development Docs	Official website
Source	Linux , ros-foxy	DIABLO Manual	Direct drive

You can compile our SDK source code on most Linux devices. Or compile our provided ros package directly on devices that support ros-foxy, please note that you need to reconfigure the serial port mapping before compiling to switch hardware devices.

Quick Start

Preparations

Before connecting the robot, you need to do the following preparations:

Display

The development board supports the HDMI display interface. Connect the development board and the display via an HDMI cable to support graphical desktop display.

Network connection

The development board supports two network interfaces: wired Ethernet and wireless WiFi. Users can achieve network connection functions through any interface.

Login to the system

The development board supports two system versions, Ubuntu 20.04 Desktop and Server. Users can freely choose to use them according to their personal habits to get a freer experience. If the user is more familiar with command line interactive operations, the Ubuntu Server version without a graphical desktop can be used.

Before this, the robot has been deployed with the ROS2 driver package at the factory, and the user does not need to deploy it by himself. Skip the creation of ros_ws. If the hardware device is switched, it needs to be redeployed

1. Create ros project workspace

```
#make sure you have build all dependence.

sudo apt-get install python3-colcon-common-extensions
mkdir -p ~/diablo_ws/src
cd ~/diablo_ws/src

#clone API source code
git clone -b basic https://github.com/DDTRobot/diablo_ros2.git

cd ~/diablo_ws
colcon build
source install/setup.bash

#before starting the node , please check of serial port in diablo_ctrl.cpp is
correct.
ros2 run diablo_ctrl diablo_ctrl_node

#run controller python script
ros2 run diablo_teleop teleop_node
```

2、Full compilation

```
#make sure you have built all dependencies.

sudo apt-get install python3-colcon-common-extensions python3-pip
sudo pip3 install rosdep
sudo rosdep init
rosdep update
mkdir -p ~/diablo_ws/src
cd ~/diablo_ws/src

#clone API source code
git clone https://github.com/DDTRobot/diablo_ros2.git
cd ~/diablo_ws
rosdep install -i --from-path src --rosdistro foxy -y
```

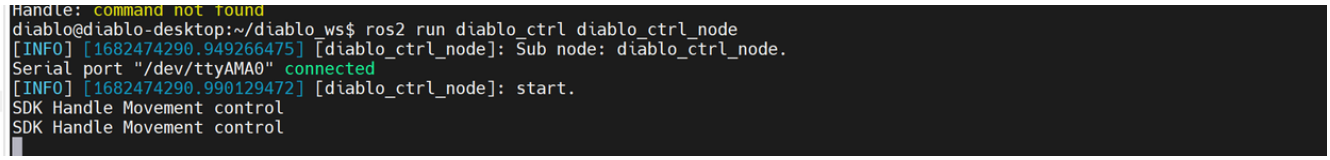
```
colcon build
source install/setup.bash

#before starting the node , please check if the serial port in diablo_ctrl.cpp is
correct.
ros2 run diablo_ctrl diablo_ctrl_node

#run controller python script
ros2 run diablo_teleop teleop_node
```

Example

1.Open the first terminal and enter the following command to obtain control authority:

```
ros2 run diablo_ctrl diablo_ctrl_node
```

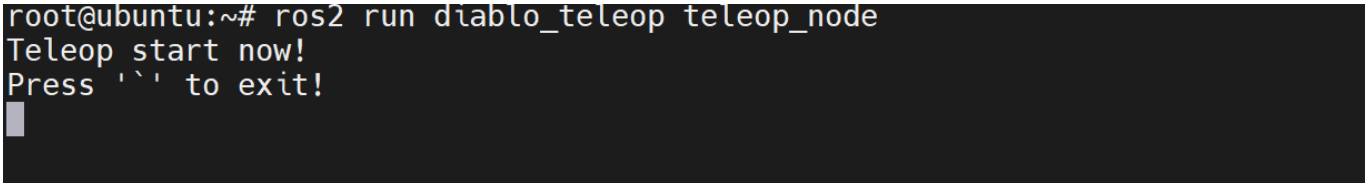
After successful execution, the terminal will output the following information:



Open another terminal: Enter the following command for keyboard control:

```
ros2 run diablo_teleop teleop_node
```

After successful execution, the terminal will output the following information:



Note: Before operating, you need to turn off the remote control handle, otherwise, pressing the keyboard will have no response.

Operation Instructions:

```
w: Control the robot to move forward. (-1.0 to +1.0 meters/second); (-1.6 to +1.6
meters/second Low-speed mode::High-speed mode::
s: Control the robot to move backward. (-1.0 to +1.0 meters/second); (-1.6 to +1.6
meters/second Low-speed mode::High-speed mode::
a: Control the robot to turn left. (-5.0 to +5.0 radians/second) Arbitrarily
mode::
d: Control the robot to turn right. (-5.0 to +5.0 radians/second) Arbitrarily
mode::
q: Control the robot to tilt left. (-0.2 to +0.2 radians/second) Standing mode::
```

```
e: Control the robot to tilt right. (-0.2 to +0.2 radians/second) Standing mode::
r: Adjust the tilt angle of the body to horizontal. Standing mode::
z: Switch the robot to standing mode.
x: Switch the robot to crawling mode.
v: Used to improve the control mode of the robot. (0 ~ 1) Position mode 0:
b: Used to improve the control mode of the robot. (-0.25 to +0.25 meters/second)
Position mode 1:
n: Used for the control mode of the robot's head pitch. (0 to 1) Position mode 0:
m: Used for the control mode of the robot's head pitch. (-0.3 to +0.3
radians/second) Position mode 1:
h: The minimum height in standing mode. Position mode
k: The medium height in standing mode. Position mode
j: The maximum height in standing mode. Position mode
u: Control the robot to look up. Position mode
i: Adjust the body to horizontal. Position mode
o: Control the robot to look down. Position mode
f: Moonwalk. Dance mode
g: End of moonwalk. Dance mode
c: Jump mode.
: Exit the virtual remote control.
```

diablo ROS API Description

1.diablo_ctrl_node

The diablo_ctrl_node node includes the topics it publishes and the topics it subscribes to so that other nodes can interact with it correctly.

2.Subscribable Topic Interfaces

Topic Names:

```
/diablo/MotionCmd
/diablo/sensor/Battery
/diablo/sensor/Body_state
/diablo/sensor/Imu
/diablo/sensor/ImuEuler
/diablo/sensor/Motors
```

3.Control API

The diablo control is encapsulated in the form of ros2 topics so that clients can control the robot's operation services.

Details

Functional Overview: Entering standing mode, controlling standing posture, controlling height, moving forward and sideways, pitch, roll.

Topic: `/diablo/MotionCmd`

Msg Types: `motion_msgs/msg/MotionCtrl`

Command Example:

```
ros2 topic pub /diablo/MotionCmd motion_msgs/msg/MotionCtrl "{mode_mark: false,
value: {forward: 0.0, left: 0.0, up: 0.0, roll: 0.0, pitch: 0.0, leg_split: 0.0},
mode: {pitch_ctrl_mode: false, roll_ctrl_mode: false, height_ctrl_mode: false,
stand_mode: false, jump_mode: false, split_mode: false}}"
```

Field Description:

For the mode_mark in the MotionCtrl message, setting it to true means you are sending a mode setting command, not a direct movement control command. In the mode setting command, you can set the robot to enter a specific operating mode, such as standing mode, jumping mode, etc. For example: To set the robot to enter the standing mode and change the height control, you can send the following command:

```
msg->mode_mark = true;
 msg->mode.stand_mode = true;
 msg->value.up = 1.0;  //Change the up value to change the height
 motion_publisher->publish(msg);
```

Assuming no mode setting is performed, which means mode_mark is set to false. For more control examples, refer to: teleop.py

---

4.Sensor Information

Sensor information includes the acquisition and invocation of IMU, power system, and data from six motors.

Details

IMU Data

Functional Overview: Acquire rapid 50hz IMU quaternion, angular velocity, acceleration, and other data.

Topic: `/diablo/sensor/Imu`

Msg Type: `sensor_msgs/msg/Imu`

Command Example: `ros2 topic echo /diablo/sensor/Imu`

IMU Euler Data

Functional Overview: Acquire rapid 50hz IMU Euler pitch, roll, yaw, and other data.

Topic: /diablo/sensor/ImuEuler

Msg Type: `ception_msgs/msg/IMUEuler`

Command Example: `ros2 topic echo /diablo/sensor/ImuEuler`

5.Motor API

The motor API is used to obtain the status of the six motors and leg length.

Details

Functional Overview: Read the status information of the six motors and the length of the left and right legs.

Topic: `/diablo/sensor/Motors`

Msg Type: `motion_msgs/msg/LegMotors`

Command Example: `ros2 topic echo /diablo/sensor/Motors`

Field Description:

- header: Standard metadata for higher-level timestamped data types. This field is used to convey timestamp and coordinate frame information;
- enc_rev: Number of revolutions;
- pos: Position, unit rad;
- vel: Angular velocity, unit rad/s;
- iq: Current, unit A;
- leg_length: Leg length, unit m;

6.Power System API

The power system API can only view information data such as battery information.

Details

Functional Overview: Subscribe to the Topic to feedback power information.

Topic: `/diablo/sensor/Body_state`

Msg Type: `sensor_msgs/msg/Battery`

Command Example: `ros2 topic echo /diablo/sensor/Battery`

Field Description:

- header: Standard metadata for higher-level timestamped data types. This field is used to convey timestamp and coordinate frame information;
- voltage: Voltage, unit V;
- current: Bus current, unit A;
- percentage: Remaining battery level

7.Robot Status Information

Robot status information feedback such as exceptions, uninitialized, etc., can be used for self-checking faults, warnings, and other issues.

Details

Functional Overview: Subscribe to the Topic to feedback power information.

Topic: `/diablo/sensor/Body_state`

Msg Type: `motion_msgs/msg/RobotStatus`

Command Example: `ros2 topic echo /diablo/sensor/Battery`

Field Description:

- header: Standard metadata for higher-level timestamped data types. This field is used to convey timestamp and coordinate frame information;
- ctrl_mode_msg: Robot control status
- robot_mode_msg: Robot mode status
- error_msg: Robot error status code
- warning_msg: Robot warning status code

Status Codes

- ctrl_mode_msg = 1; SDK control authority mode
- robot_mode_msg = 2; Crawling state
- robot_mode_msg = 3; Standing state
- robot_mode_msg = 4; Transition from standing state to crawling state
- robot_mode_msg = 5; Transition from crawling state to standing state
- error_msg = 2064384; Disconnection of six motors, if there are other status codes, please contact DDT FAE for assistance and diagnosis, the same applies to warning_msg status codes.

FAQ

- [1 Problem]

```
[INFO] [1684144302.531810930] [diablo_ctrl_node]: start.
OSDK Serial Receive Timeout Occured!
OSDK Serial Receive Timeout Occured!
OSDK Serial Receive Timeout Occured!
OSDK Serial Receive Timeout Occured!
OSDK Serial Receive Timeout Occured!
OSDK Serial Receive Timeout Occured!
OSDK Serial Receive Timeout Occured!
OSDK Serial Receive Timeout Occured!
OSDK Serial Receive Timeout Occured!
OSDK Serial Receive Timeout Occured!
```

- [1 Solution]

- Make sure you cloned the diablo_ros2 from the basic branch on GitHub
- Check if the 40p cable connecting the development board and the robot control board is connected properly or not.

- [2 Problem]

```
diablo@diablo-desktop:~$ ros2 run diablo_ctrl diablo_ctrl_node
[INFO] [1684144042.624726365] [diablo_ctrl_node]: Sub node: diablo_ctrl_node.
terminate called after throwing an instance of 'VulcanSerial::Exception'
  what():  /home/diablo/diablo_ws/src/diablo_ros2/diablo_common/diablo_utils/src/SerialPort/SerialPort.cpp:137: Could not open device /dev/ttyS3.
s the device name correct and do you have read/write permission?
diablo@diablo-desktop:~$ cd diablo_ws/
```

- [2 Solution]

- Solution: Change Hal.init("/dev/ttys3") in diablo_ctrl.cpp to the correct port of your device, save and recompile.

Contents

The following is the ros2 node Directory:

- Robot ception node

  > Robot carrier sensor

- SDK source code and common fuction
- Robot firing sdk and ctrl

  > Start sdk ctrl
  >
  > Capture keyboard event

- Ros custom msgs

  > Robot basic movement ctrl msg

- Ros rviz2 and gazebo example

  > rviz2 rviz2 gazebo simulation
  >
  > rviz2 Qt teleop UI
  >
  > motor angle 2 urdf angle