RoArm-M1 Tutorial V: ROS2 Serial Communication Node

From Waveshare Wiki
Jump to: navigation, search

RoArm-M1 Tutorial Directory

- RoArm-M1 Tutorial I: How To Use (/wiki/RoArm-M1_Tutorial_I:_How_To_Use)
- RoArm-M1 Tutorial II: Secondary Development Tutorial (/wiki/RoArm-M1_Tutorial_II:_Secondary_Development_Tutorial)
- RoArm-M1 Tutorial III: VMware ROS2 Getting Started Tutorial (/wiki/RoArm-M1_Tutorial_III:_VMware_ROS2_Getting_Started_Tutorial)
- RoArm-M1 Tutorial IV: URDF Model Tutorial (/wiki/RoArm-M1_Tutorial_IV:_URDF_Model_Tutorial)
- RoArm-M1 Tutorial V: ROS2 Serial Communication Node
- RoArm-M1 Tutorial VI: How to assemble RoArm-M1 (/wiki/RoArm-M1_Tutorial_VI:_How_to_assemble_RoArm-M1)
- RoArm-M1 Tutorial VII: Assembly Graphics Tutorial (/wiki/RoArm-M1_Tutorial_VII:_Assembly_Graphics_Tutorial)
- RoArm-M1 Tutorial VIII: Use of rqt in ROS2 (/wiki/RoArm-M1_Tutorial_VIII:_Use_of_rqt_in_ROS2)
- RoArm-M1 Tutorial IV: URDF Model Tutorial (/wiki/RoArm-M1_Tutorial_IV:_URDF_Model_Tutorial)
- RoArm-M1 Main Page (/wiki/RoArm-M1)

RoArm-M1 ROS2 Serial Communication Node Tutorial

- This tutorial is for learning RoArm-M1 ROS2 serial communication node.

Introduction

In ROS2, Node, Topic, Subscription, and Publisher are the core concepts used to implement communication in a distributed system. Each node in ROS2 is responsible for a single, modular task, and each node can send and receive data to and from other nodes through topics, services, etc. In our ROS2 demos, two ROS2 packages are provided, both located in roarm_ws/src, where roarm is the package related to the URDF model, and serial_ctrl is the package related to the serial communication node. serial_ctrl node is used to send the pose information published by the URDF package to the robotic arm through the serial port, thus controlling the robotic arm action. The attitude information refers to the absolute angle degree of each servo of the arm. In this tutorial, we will introduce the program structure of this node and how this node works.

Demo Introduction

The main demo of the serial communication node is in serial_ctrl, and the file name is serial_ctrl_py.py, which is compiled in Python language. Here we introduce the code in this file.

```
import rclpy
from rclpy.node import Node
import array

from sensor_msgs.msg import JointState
from std_msgs.msg import Float64

import json
import serial
```

The first three lines are responsible for ROS2's Python interface and node classes.
The middle two lines are the data format of attitude information.
The last two lines are a library for serial communication and a library for encoding and decoding JSON data (serial communication uses JSON data format).

```
ser = serial.Serial("/dev/ttyUSB0",115200)
```

Instantiate the serial port communication object. The serial port device we use here is USB0. If you port this demo to other devices, you need to change the device and baud rate here.

```
class MinimalSubscriber(Node):
    def __init__(self):
        super().__init__('serial_ctrl')

        self.position = []

        self.subscription = self.create_subscription(
            JointState,
            'joint_states',
            self.listener_callback,
            10)
        self.subscription  # prevent unused variable warning

    def posGet(self, radInput, direcInput, multiInput):
        if radInput == 0:
            return 2047
        else:
            getPos = int(2047 + (direcInput * radInput / 3.1415926 * 2048 * multiInput) + 0.5)
            return getPos

    def listener_callback(self, msg):
        a = msg.position

        join1 = self.posGet(a[0], -1, 1)
        join2 = self.posGet(a[1], -1, 3)
        join3 = self.posGet(a[2], -1, 1)
        join4 = self.posGet(a[3],  1, 1)
        join5 = self.posGet(a[4], -1, 1)

        data = json.dumps({'T':3,'P1':join1,'P2':join2,'P3':join3,'P4':join4,'P5':join5,'S1':0,'S2':0,'S3':0,'S4':0,'S5':0,'A1':60,'A2':60,'A3':60,'A4':60,'A5
':60})

        ser.write(data.encode())

        print(data)
```

MinimalSubscriber inherits from Node and is used to define a ROS2 node:
__init__(self) defines the node name serial_ctrl.
create_subscription() needs to define the data type of the subscribed topic: JointState, and the subscribed topic name: joint_states. The callback function self.listener_callback after receiving the subscription data, saves the message queue length 10.
posGet() function is for entering the angle, direction, and reduction ratio in radians.
listener_callback function is for entering the subscribed data and converting the angle information in radians into the position data of the servo, packaging it in JSON format, and sending it to the robotic arm via the serial port.

```
def main(args=None):
    rclpy.init(args=args)
    minimal_subscriber = MinimalSubscriber()

    rclpy.spin(minimal_subscriber)

    # Destroy the node explicitly
    # (optional - otherwise it will be done automatically
    # when the garbage collector destroys the node object)
    minimal_subscriber.destroy_node()
    rclpy.shutdown()
```

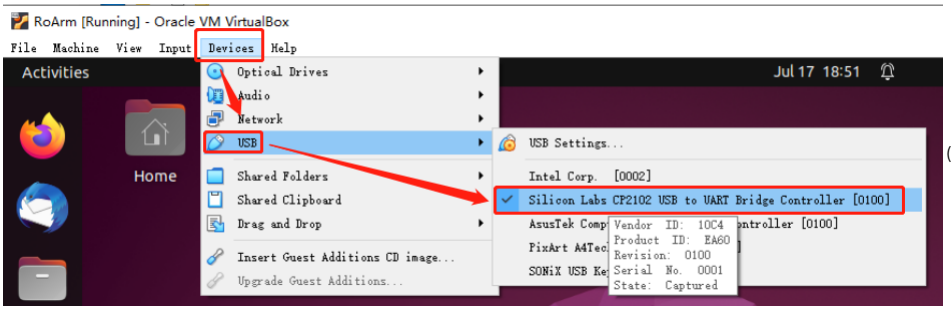
In the main() function, first, initialize rclpy, then create a MinimalPublisher, rclpy.spin() enters the spin lock, and the node will be closed when the lock is exited.

Compile

```
cd roarm_ws
colcon build
```

Running Serial Communication Nodes

First, power on the robotic arm, and connect the robotic arm to the computer via the USB cable. VMware will query if the new device is to be connected to a host or a virtual machine, select the virtual machine.



(/wiki/File:RoArm-M1_Tutorial_V_07.png)

Open a new terminal:

```
sudo chmod 777 /dev/ttyUSB0
If an error is reported here, it is likely that your VM is not connected to the robotic arm.
cd roarm_ws
Configuration source files
source install/setup.bash
```

To start rviz2, enter the following command in the terminal:

```
ros2 launch roarm roarm.launch.py
```

The rviz2 window will appear, showing a model of this robotic arm.
Without closing the above terminal, open a new one and run the following command:

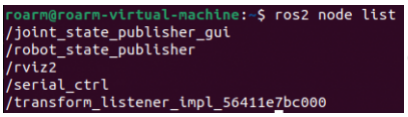
```
cd roarm_ws
source install/setup.bash
```

Use the following commands to run the serial communication nodes. The first serial_ctrl is the package in the roarm/src directory, and the second is the serial_ctrl node inside the serial_ctrl package:

```
ros2 run serial_ctrl serial_ctrl
```

At this point, you should have four windows open, a terminal window running roarm.launch.py, a terminal window running the serial_ctrl node, a robotic arm model window in rviz2, and a control panel window. Adjust the position of these windows so that you can see the robotic arm model window and make sure that the terminal running the serial_ctrl node is active, next you can control the movement of the robotic arm model in rviz2 by dragging and dropping the sliders in the control panel, the real robotic arm product will follow along.
Open a new window again, and run the following commands to view all the running nodes list:

```
ros2 node list
```

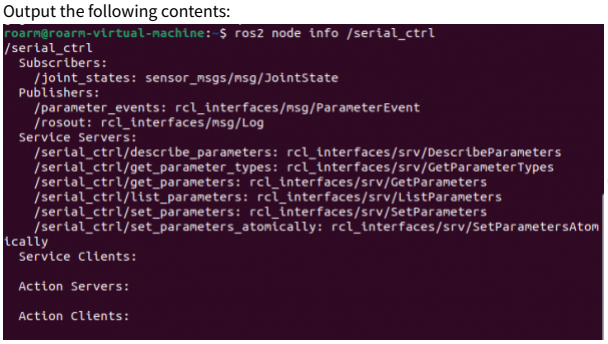


(/wiki/File:RoArm_Tutorial_V_17.png)

You can see the names of the nodes that are running on the system, use the ros2 node info <node_name> command to get more information about the node, it will return a series of subscriber, publisher, service, and action information.
Here we are looking at information about the serial communication node /serial_ctrl, run the following command:

```
ros2 node info /serial_ctrl
```

Output the following contents:



(/wiki/File:RoArm_Tutorial_V_18.png)

Since then, we can control the movement of the robotic arm by running the serial communication node. Then, we do further learning based on the subsequent tutorials.