RealMan Robotrm_exampleUser Manual V1.0

RealMan Intelligent Technology (Beijing) Co., Ltd.

Revision History:

No.	Date	Comment
V1.0	11/22/2023	Draft

Content

1. rm_example Package Description

rm_bringup package is used for realizing some basic robotic arm functions. And we can also refer to the code to realize other robotic arm functions. This package is introduced in detail in the following aspects.

1. Package use.
2. Package architecture description.
3. Package topic description.

Through the introduction of the three parts, it can help you:

1. Understand the package use.
2. Familiar with the file structure and function of the package.
3. Familiar with the topic related to the package for easy development and use.

Source code address:https://github.com/RealManRobot/ros2_rm_robot.git.

2. rm_example Package Use

2.1 Changing the work coordinate system.

First, we need to run the underlying driver node of the robotic arm rm_driver.

```
rm@rm-desktop:~$ ros2 launch rm_driver rm_<arm_type>_driver.launch.py
```

In practice, the above <arm_type> needs to be replaced by the actual model of the robotic arm. The available models of the robotic arm are 65, 63, eco65, and 75.

For example, the launch command of 65 robotic arm:

```
rm@rm-desktop:~$ ros2 launch rm_driver rm_65_driver.launch.py
```

After successfully launching the node, execute the following commands to replace the node in the work coordinate system.

```
rm@rm-desktop:~$ ros2 run rm_example rm_change_work_frame
```

The following command pops up to indicate successful replacement.

```
[INFO] [1701416861.664450164] [changeframe]: *******Switching the tool coordinate system succeeded
```

Enter the following commands in the end for verification, first subscribe to the current work coordinate system topic.

```
rm@rm-desktop:~$ ros2 topic echo /rm_driver/get_curr_workFrame_result
```

Then publish the request of the current coordinate system.

```
rm@rm-desktop:~$ ros2 topic pub --once /rm_driver/get_curr_workFrame_cmd std_msgs/msg/Empty "{}"
```

You can see the following interface pop up at the end.

```
rm@rm-desktop:~$ ros2 topic echo /rm_driver/get_curr_workFrame_result
data: Base
---
```

2.2. Get the current state message of the robotic arm.

First, we need to run the underlying driver node of the robotic arm rm_driver.

```
rm@rm-desktop:~$ ros2 launch rm_driver rm_<arm_type>_driver.launch.py
```

In practice, the above <arm_type> needs to be replaced by the actual model of the robotic arm. The available models of the robotic arm are 65, 63, eco65, and 75.

For example, the launch command of 65 robotic arm:

```
rm@rm-desktop:~$ ros2 launch rm_driver rm_65_driver.launch.py
```

After successfully launching the node, execute the following commands to obtain the current state of the robotic arm node.

```
rm@rm-desktop:~$ ros2 run rm_example rm_get_state
```

The following command pops up to indicate successful replacement.

```
[INFO] [1701417669.964643809] [get_state]: joint state is: [-0.001000, 0.002000, 0.000000, 0.000000, 0.000000, -0.001000, -0.001000]
[INFO] [1701417669.964801453] [get_state]: pose state is: [0.000025, 0.000000, 0.850500, 0.000000, 0.000000, 0.000000]
[INFO] [1701417669.964838239] [get_state]: arm_err is: 0
[INFO] [1701417669.964852780] [get_state]: sys_err is: 0
```

What is displayed in the interface is the current angle message of the robotic arm, as well as the current end coordinate position and Euler angle posture message of the robotic arm.

2.3 MoveJ motion of the robotic arm

Through the following commands, we can control the joint MoveJ motion of the robotic arm.

First, we need to run the underlying driver node of the robotic arm rm_driver.

```
rm@rm-desktop:~$ ros2 launch rm_driver rm_<arm_type>_driver.launch.py
```

In practice, the above <arm_type> needs to be replaced by the actual model of the robotic arm. The available models of the robotic arm are 65, 63, eco65, and 75.

For example, the launch command of 65 robotic arm:

```
rm@rm-desktop:~$ ros2 launch rm_driver rm_65_driver.launch.py
```

After successfully launching the node, execute the following commands to control the movement of the robotic arm.

```
rm@rm-desktop:~$ ros2 launch rm_example rm_<dof>_movej.launch.py
```

dof represents the current degree of freedom message of the arm, and the parameters can be selected as 6dof and 7dof

For example, when starting the 7-axis robotic arm, the following commands are needed.

```
rm@rm-desktop:~$ ros2 launch rm_example rm_7dof_movej.launch.py
```

After successful running, the joint of the robotic arm rotates and the interface displays as follows.

```
[INFO] [launch]: All log files can be found below /home/rm/.ros/log/2023-12-01-16-31-59-669068-rm-desktop-9816
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [movej_demo-1]: process started with pid [9817]
[movej_demo-1] [INFO] [1701419519.880260228] [Movej_demo]: arm_dof is 7
[movej_demo-1]
[movej_demo-1] [INFO] [1701419520.691271905] [Movej_demo]: *******Movej succeeded
[movej_demo-1]
```

2.4 MoveJ_P motion of the robotic arm

Through the following commands, we can control the joint MoveJ_P motion of the robotic arm.

First, we need to run the underlying driver node of the robotic arm rm_driver.

```
rm@rm-desktop:~$ ros2 launch rm_driver rm_<arm_type>_driver.launch.py
```

In practice, the above <arm_type> needs to be replaced by the actual model of the robotic arm. The available models of the robotic arm are 65, 63, eco65, and 75.

For example, the launch command of 65 robotic arm:

```
rm@rm-desktop:~$ ros2 launch rm_driver rm_65_driver.launch.py
```

After successfully launching the node, execute the following commands to control the movement of the robotic arm.

```
rm@rm-desktop:~$ ros2 run rm_example movejp_demo
```

After successful execution, the interface appears as follows, and the robotic arm will move to the specified pose.

```
[INFO] [1701421702.758055237] [Movejp_demo_node]: *******MoveJP succeeded
```

2.5 MoveL motion of the robotic arm

Through the following commands, we can control the joint MoveL motion of the robotic arm.

First, we need to run the underlying driver node of the robotic arm rm_driver.

```
rm@rm-desktop:~$ ros2 launch rm_driver rm_<arm_type>_driver.launch.py
```

In practice, the above <arm_type> needs to be replaced by the actual model of the robotic arm. The available models of the robotic arm are 65, 63, eco65, and 75.

For example, the launch command of 65 robotic arm:

```
rm@rm-desktop:~$ ros2 launch rm_driver rm_65_driver.launch.py
```

After successfully launching the node, execute the following commands to control the movement of the robotic arm.

```
rm@rm-desktop:~$ ros2 run rm_example movel_demo
```

After successful execution, the interface appears as follows, and the robotic arm performs two motions. Firstly, it moves to the specified pose through MoveJP, and then performs joint motion through MoveL.

```
[INFO] [1701422229.234605201] [Movel_demo_node]: *******MoveJP succeeded

[INFO] [1701422230.095942637] [Movel_demo_node]: *******MoveL succeeded
```

3. rm_example Package Architecture Description

3.1 Overview of package files

The current rm_driver package is composed of the following files.

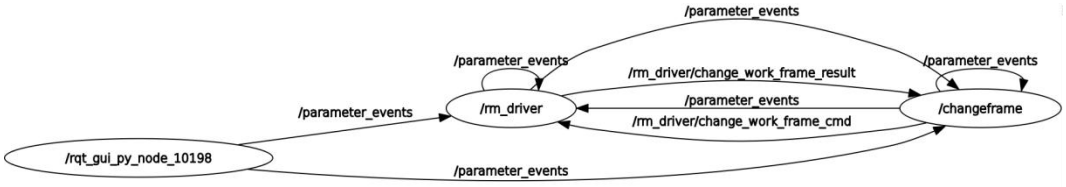
```
├── CMakeLists.txt                    # compilation rule file
├── include
│   └── rm_example
├── launch
│   ├── rm_6dof_movej.launch.py                    # 6 degrees of freedom MoveJ movement launch file
│   └── rm_7dof_movej.launch.py                    #7 degrees of freedom MoveJ movement launch file
├── package.xml
└── src
    ├── api_ChangeWorkFrame_demo.cpp              # source file to change the work
```

```
coordinate system
    ├── api_Get_Arm_State_demo.cpp        # source file to get the robotic arm's state
    ├── api_MoveJ_demo.cpp                 #MoveJ motion source file
    ├── api_MoveJP_demo.cpp               #MoveJP motion source file
    └── api_MoveL_demo.cpp                 #MoveL motion source file
```

4. rm_example Topic Description

4.1 rm_change_work_frame topic description

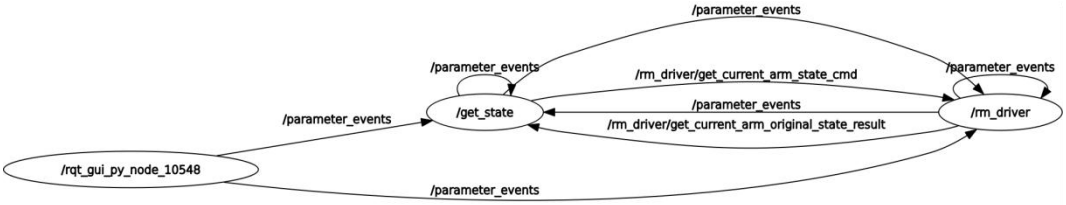
The following is the data communication diagram of this node:



You can see that the main communication topics between /changeframe node and /rm_driver are /rm_driver/change_work_frame_result and /rm_driver/change_work_frame_cmd. / rm_driver/change_work_frame_cmd is the release of the switch request and the switch target coordinates, and /rm_driver/change_work_frame_result is the switch result.

4.2 rm_get_state topic description

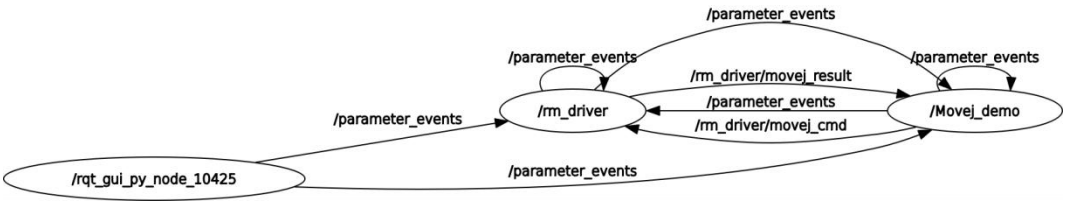
The following is the data communication diagram of this node:



You can see that the main communication topics between the /get_state node and /rm_driver are /rm_driver/get_current_arm_state_cmd and /rm_driver/get_current_arm_original_state_result. / rm_driver/get_current_arm_state_cmd is the request to get the current state of the arm, and /rm_driver/get_current_arm_original_state_result is the switch result.

4.3 movej_demo topic description

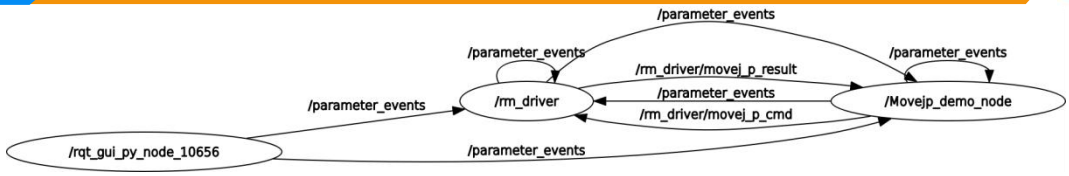
The following is the data communication diagram of this node:



You can see that the main communication topics between the /Movej_demo node and /rm_driver are /rm_driver/movej_cmd and /rm_driver/movej_result. / rm_driver/movej_cmd is the request to control the motion of the robotic arm, which will release the radian information of each joint that needs to be moved, and /rm_driver/movej_result is the motion result.

4.4 movejp_demo topic description

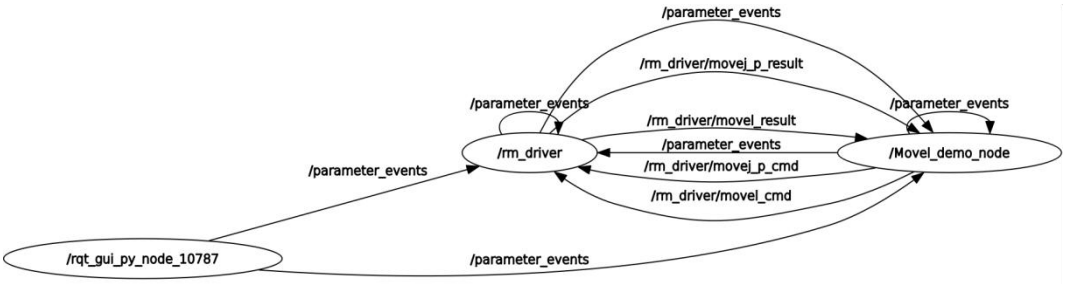
The following is the data communication diagram of this node:



You can see that the main communication topics between the /Movejp_demo_node node and /rm_driver are /rm_driver/movej_p_cmd and rm_driver/movej_p_result. / rm_driver/movej_p_cmd is the request to control the motion planning of the robotic arm, which will publish the coordinates of the target point to be moved, and /rm_driver/movej_p_result is the motion result.

4.5 movel_demo topic description

The following is the data communication diagram of this node:



You can see that the main communication topics between the /Movel_demo_node node and /rm_driver are /rm_driver/movej_p_cmd and /rm_driver/movej_p_result, and /rm_driver/movel_cmd and /rm_driver/movel_result. / rm_driver/movej_p_cmd is the request to control the motion planning of the robotic arm, which will publish the coordinates of the target point to be moved first, /rm_driver/movej_p_result is the motion result. After reaching the first point, we reach the second point through linear motion, we can publish the pose of the second point through /rm_driver/movel_cmd, and the topic of /rm_driver/movel_result is the result of the motion.