ROS tutorial for UR5e

Eva Cheung

October 26, 2023

Contents

ROS

The ROS version that had been used for development is : Noetic, with 20.04 Ubuntu

1 Environmental setup

1.1 Dual or Multi Robot setup

The command for both robot are universal. One need to figure out a way to allocate those command to the individual robot. This is achieved by adding **node name** to the command. The node name is specific to the IP address of the robot, setup through catkin_ws, within the file location: **RobotConfig/catkin_ws/src/Universal_Robot_Driver/ur_robot_driver/launch**

File location	**Purpose**
/3_paste.launch	Generate namespace "pasting_robot" when launched. Allow single robot control
/3_Assembly.launch	Generate namespace "assembly_robot" when launched. Allow single robot control.
/3_dual.launch	Generate two namespaces: "assembly_robot" and "pasting_robot". Allow dual robot control
/1_pp_robot_bringup.launch	Altered robot IP and port address based on ur5e_bringup.launch file to allow finner control
/1_d_robot_bringup.launch	Altered robot IP and port address based on ur5e_bringup.launch file to allow finner control

Table 1: An Overview of workflow

1.2 Remote Control

The UR5 e-series normally have two states: (1)Manual (2)Remote. Sometimes the robot may have an extra 'automatic mode', which acts as a compulsory intermediate mode in order to change between local and remote mode. Each mode provide different level of access to the pendant, generally speaking, manual mode means you can move the robot around using free-drive, edit the program, plane and other features. In comparison, remote mode forbidden most access such as program and feature edit.

UR5e does not allow TCP/IP communication in Manual Mode. One **MUST** change to Remote Mode before performing any ROS commands.

Figure 1

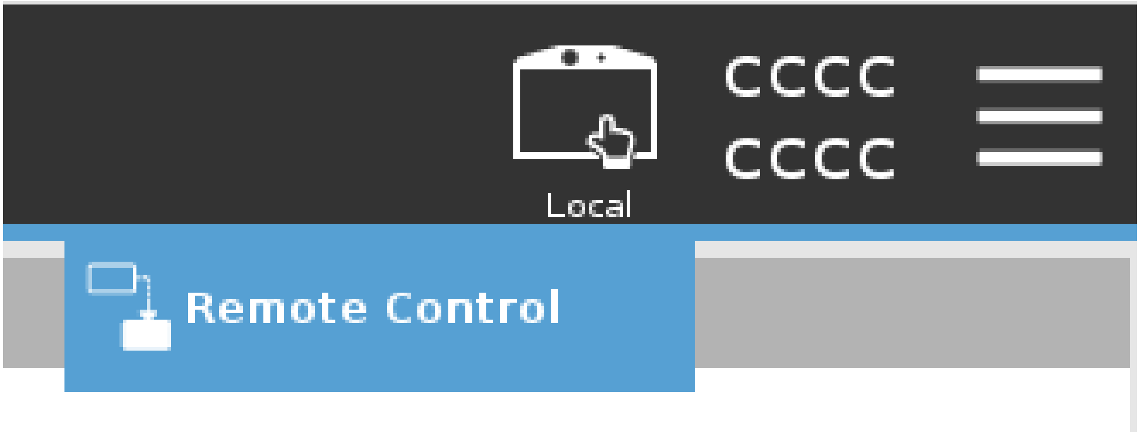


Figure 1: Location to change operational mode. e

2 Network configuration

Network Configuration
Things to watch out for

1. Make sure there is no collision between the robot and the factory network.
2. The robot network is on the same subnet as the robot PC
3. Make sure the IP configuration is completed in catkin work space as mentioned in table 1

3 Catkin_ws

Here the catkin_ws is downloaded from Universal Robot ROS Driver. This driver contain libraries. The installed version has commit id of: **"395c0541b20d0da2cd480e2ad85b2100410fb043"**. Follow the instruction and build the catkin_ws

3.1 Modify Catkin_ws

After modifying the catki_ws, there are steps should be followed to ensure it compiles correctly. In the terminal:

1. 'cd catkin_ws'
2. 'catkin_make'

If relocating the catkin_ws is required and necessary:

1. Make sure to only migrate the "src" folder to the new desired location. Hence, you DO NOT NEED the build and devel folders
2. In the bashrc file
   - If you cannot find it, in the terminal: 'cd'
   - 'gedit .bashrc'
   - At the bottom you should see the bashrc file calling "**./rosrc**". The rosrc file is the file you need to edit
3. Make sure the noetic ROS location is sourced correctly in the rosrc file

4 Communication

The selected communication between robot and PC uses DashboardServices through ROS packages from Universal Robot ROS Driver. The dashboard service from Universal Robot uses port **29999**.

NOTE! Port 29999 require the robot to be in **Remote Control Mode**

Figure 2 shows the 2 avaliable routes for using dashboard services in this project:

1. Direct connection through sockets and ports to the dashboard services
2. Connection through ROS, with pre-determined unique identifier for each robot ip address

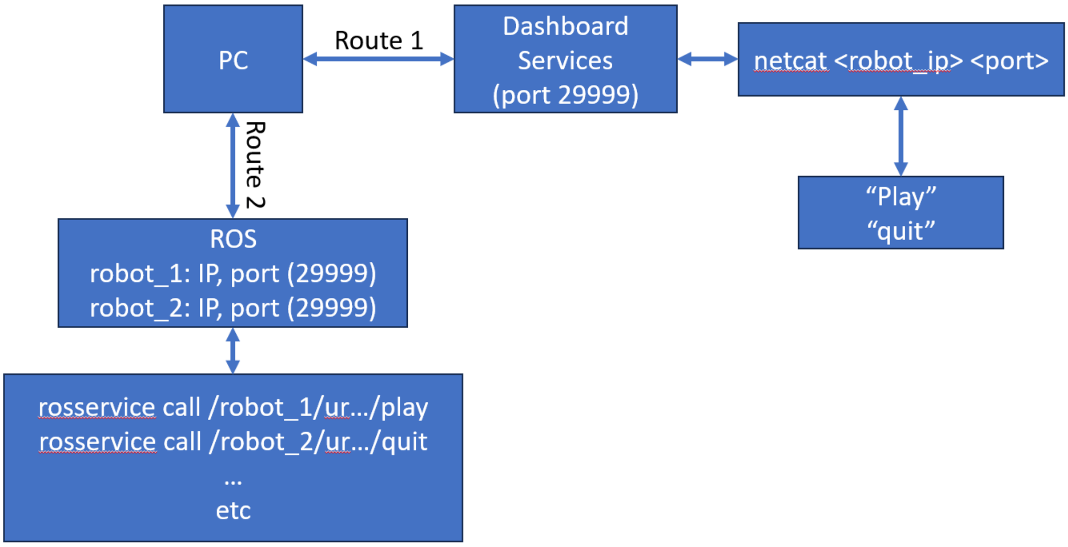


Figure 2: The avaliable routes for using Dashboard services. Both routes require the robot to be in Remote Control Mode

4.1 Example usage - Route 1, no need to launch ROS

In the terminal:
netcat <robot_ip> <port>
netcat 192.168.2.30 29999

- play
- power_off
- getVariable <variable_name>

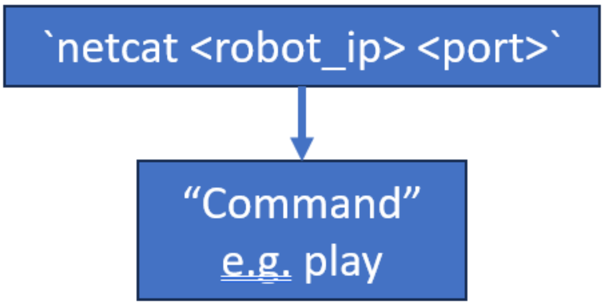


Figure 3: The route to use dashboardservice through socket in terminal

4.2 Example usage - Route 2

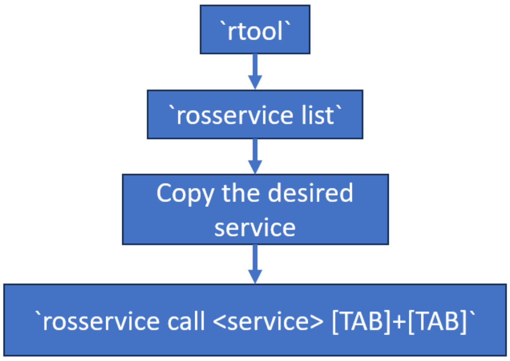


Figure 4: The route to use ROS service in terminal

NOTE, always remember to launch ROS before using any rosservices or rosmsgs.

"rtool" is launching: 'roslaunch ur_robot_driver 3_dual.launch' in the terminal. See figure 5, the highlighted part shows what a successful launch should look like. For single robot control, one can do: **roslaunch ur_robot_driver <launch_file>**, where launch files are avaliable in table 1 e.g. 3_Assembly.launch

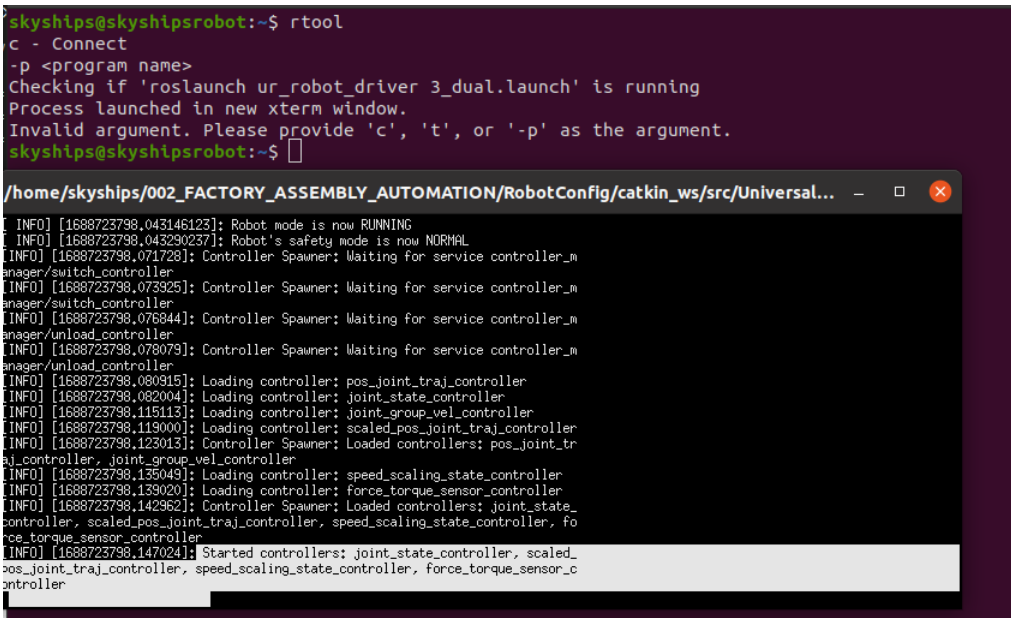


Figure 5: ROS launch using rtool. This is not part of the ROS package, this is a local .bash file. "Started Controller ..." indicate successful execution (highlighted part)

4.3 Command Summary

UR_ROS_Interface illustrated the list of available commands and their corresponding libraries & type.

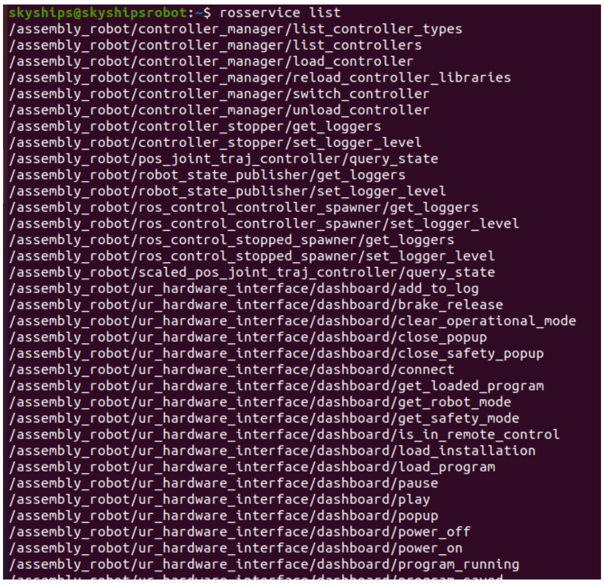
Interface Style	Category	Terminal Command
services	**rosservice**	• rosservice list • rosservice info <selected_service> • rosservice call <selected_service> [TAB]+[TAB]
	rossrv	• rossrv show <Type>, where type is generated from rosservice info • rossrv list
message/action	**rostopic**	• rostopic list • rostopic info <selected_topic> • rostopic echo <selected_topic> • rostopic pub <selected_topic>
	rosmsg	• rosmsg show <Type>, where type is generated from rostopic info • rosmsg list

Table 2: Class and function for dashboard_services.py

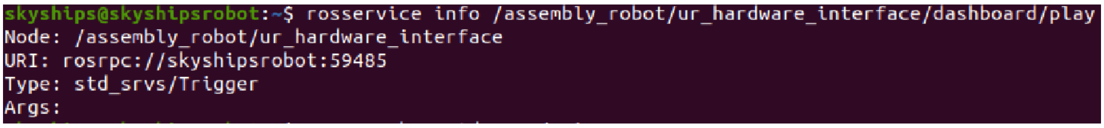
4.4 ROS Commands in detail with examples

Rosservice

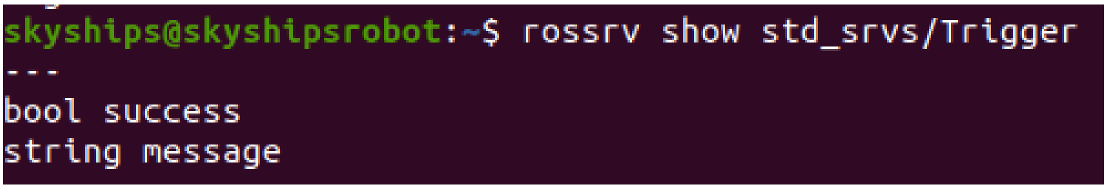
rosservice shows all the currently used/avaliable services that can be called. Interface "srv" has 3 steps: (1) Connect, (2) Request and (3) Result. It is simply a blocking call.



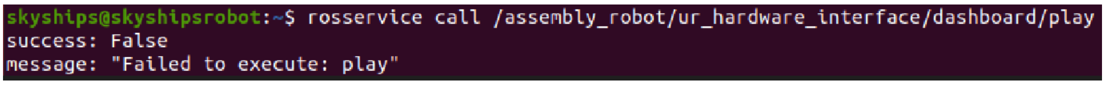
(a) Demonstrate a list of available rosservice



(b) Shows the information of the selected rosservice, the empty **arg** meaning it is not expecting any arguments when called



(c) Using the "Type" returned from rosservice info in figure 6b to illustrate what will be returned when std_srvs/Trigger is used. The library to install



(d) Example of calling rosservice with command "play", one can use [TAB] + [TAB] to complete the command to ensure it is not expecting any arguments

Figure 6: The usage of rosservice

In order to add or change any of the services or message in ROS, please head to folder location shown in figure 7 and its corresponding file location. Check the python ROS beginner course in "The Construct Sim" for step by step walk through

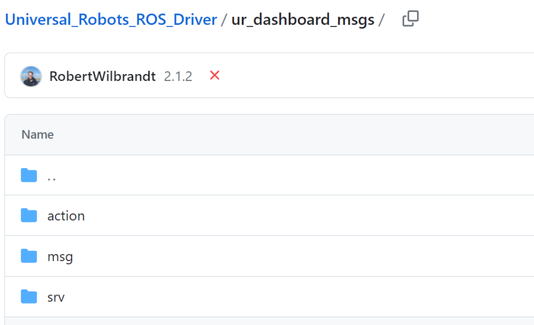


Figure 7: Folder location for customized services, messages and action

```python
''' Power on '''
import rospy
from std_srvs.srv import Trigger, TriggerRequest

srv_name = "/assembly_robot/ur_hardware_interface/dashboard/power_on"
# connect
rospy.wait_for_service(srv_name, timeout=5)
client = rospy.ServiceProxy(srv_name, Trigger)
# reques/talk
request = TriggerRequest()
# result
result = client(request)

print(result.message)
```

(a) Using rosservice with no required input argument, power on command

```python
# ''' Load '''
import rospy
from ur_dashboard_msgs.srv import Load,LoadRequest

program = 'scre_1.urp'
srv_name = '/assembly_robot/ur_hardware_interface/dashboard/load_program'

rospy.wait_for_service(srv_name, timeout=5)
client = rospy.ServiceProxy(srv_name, Load)

request = LoadRequest()
request.filename = program

result = client(request)

# print(result.success)
load_fail = result.success

# logic
if load_fail == False:
    print('ahahahhahah')
```

(b) Using rosservice with required input argument "filename", load program command

Figure 8: Python example for rosservice usage

To summarise figure 6:

1. There is a ros service called **"/assembly_robot/ur_hardware_interface/dashboard/play"** (Figure 6a)

2. **No input arguments** required when the service is being called. (Figure 6b)

3. This ros service uses **standard srv libraries** with message type "Trigger". (Figure 6b)

   – in python: **from std_srv.srv import Trigger, TriggerRequest**

4. Upon execution of the command, there are two variables that can be accessed. (1) success, (2) message (Figure 6c and 6d)

   – in python: **result.message** will return string. With result being the product of using ros service. See figure 8a for example usage.

Rostopic

rostopic shows all the available ROS messgage and action that can be currently displayed or called. Interface type "msg" only has one step, which simply return the observed/published information. Interface type "action" is a non-blocking call contain 4 major steps: (1) goal, (2) status, (3) feedback, (4) result

```
skyships@skyshipsrobot:~$ rostopic list
/assembly_robot/joint_group_vel_controller/command
/assembly_robot/joint_states
/assembly_robot/pos_joint_traj_controller/command
/assembly_robot/pos_joint_traj_controller/follow_joint_trajectory/cancel
/assembly_robot/pos_joint_traj_controller/follow_joint_trajectory/feedback
/assembly_robot/pos_joint_traj_controller/follow_joint_trajectory/goal
/assembly_robot/pos_joint_traj_controller/follow_joint_trajectory/result
/assembly_robot/pos_joint_traj_controller/follow_joint_trajectory/status
/assembly_robot/pos_joint_traj_controller/state
/assembly_robot/scaled_pos_joint_traj_controller/command
/assembly_robot/scaled_pos_joint_traj_controller/follow_joint_trajectory/cancel
/assembly_robot/scaled_pos_joint_traj_controller/follow_joint_trajectory/feedback
/assembly_robot/scaled_pos_joint_traj_controller/follow_joint_trajectory/goal
/assembly_robot/scaled_pos_joint_traj_controller/follow_joint_trajectory/result
/assembly_robot/scaled_pos_joint_traj_controller/follow_joint_trajectory/status
/assembly_robot/scaled_pos_joint_traj_controller/state
/assembly_robot/speed_scaling_factor
/assembly_robot/ur_hardware_interface/io_states
/assembly_robot/ur_hardware_interface/robot_mode
/assembly_robot/ur_hardware_interface/robot_program_running
/assembly_robot/ur_hardware_interface/safety_mode
```

(a) Demonstrate a list of available rostopic

```
skyships@skyshipsrobot:~$ rostopic info /assembly_robot/ur_hardware_interface/io_states
Type: ur_msgs/IOStates

Publishers:
 * /assembly_robot/ur_hardware_interface (http://skyshipsrobot:39705/)

Subscribers: None
```

(b) Shows the information of the selected rostopic

```
skyships@skyshipsrobot:~$ rosmsg show ur_msgs/IOStates
ur_msgs/Digital[] digital_in_states
  uint8 pin
  bool state
ur_msgs/Digital[] digital_out_states
  uint8 pin
  bool state
ur_msgs/Digital[] flag_states
  uint8 pin
  bool state
ur_msgs/Analog[] analog_in_states
  uint8 CURRENT=0
  uint8 VOLTAGE=1
  uint8 pin
  uint8 domain
  float32 state
ur_msgs/Analog[] analog_out_states
  uint8 CURRENT=0
  uint8 VOLTAGE=1
  uint8 pin
  uint8 domain
  float32 state
```

(c) Use 'Type' obtained from figure 9b to illustrate what will be returned when called

Figure 9: The usage of rostopic

To summarise figure 9:

1. There is a ROS node called **"/assembly_robot/ur_hardware_interface/io_state"** (Figure 6a)

2. This ROS service uses **ur_msgs** libraries with **message** type "IOStates". (Figure 6b)

   – in Python: **from ur_msgs.msg import IOStates**

3. The list of all nodes publishing on the topic (Publisher), very useful if you don't know from where some data is coming from. No other topics that are subscribing to the node (Figure 9b)

4. When used, the ROS message returns digital_in_states, digital_out_states, flag_states, analog_in_states, analog_out_states. Each of which has a list of information that one can specify, e.g., **digital_in_states[2].state** returns the **state of digital input 2**. See Figure 10 for Python examples.

Python example for rostopics:

```python
def call_back(msg):
    # this is pin 3
    data = msg.digital_out_states[3].state
```

(a) rostopic usage python example, callback code. With msg being the monitored data

```python
io_state = '/pickAndPlace_robot/ur_hardware_interface/io_states'
io_state_msg = IOStates()
rospy.Subscriber(io_state, IOStates ,call_back, queue_size=1)
rospy.spin()
```

(b) rostopic usage python example, subscribing with callback node

Figure 10: Python example for rostopic (type: message)