8.ROS topic subscribers

The previous section introduced how to write a publisher node program in ros. This section introduces how to write a subscriber program.

8.1 Create a subscriber

The general creation steps are as follows:

- Initialize ROS nodes
- Create handle
- Subscribe to the topics you need
- Wait in a loop for topic messages, and enter the callback function after receiving the message
- Complete message processing in the callback function

The callback function is an important part of the subscriber. It processes the subscribed message, then parses the message data, and determines how to proceed with the subsequent program based on the parsed message data.

8.2 C++ version

8.2.1 Writing source code

In the src folder of the function package learn_topic, create a C++ file (the file suffix is .cpp), name it turtle_pose_subscriber.cpp, and paste the following content into turtle_pose_subscriber.cpp,

```
/*Create a small turtle to receive the current pose information*/
#include <ros/ros.h>
#include "turtlesim/Pose.h"
// After receiving the message, you will enter the message callback function, 
which will process the received data.
void turtle_poseCallback(const turtlesim::Pose::ConstPtr& msg)
{
    // Print received messages
    ROS_INFO("Turtle pose: x:%0.3f, y:%0.3f", msg->x, msg->y);
}
int main(int argc, char **argv)
{
    ros::init(argc, argv, "turtle_pose_subscriber");// Initialize ROS node
    ros::NodeHandle n;//Here is create handle
    // Create a subscriber. The topic of subscription is the topic of 
/turtle1/pose. poseCallback is the callback function.
    ros::Subscriber pose_sub = n.subscribe("/turtle1/pose", 
10,turtle_poseCallback);
    ros::spin(); // Loop waiting for callback function
    return 0;
}
```

8.2.2 Modify CMakelist.txt file

Configure in CMakelist.txt, under the build area, add the following content,

```
add_executable(turtle_pose_subscriber src/turtle_pose_subscriber.cpp)
target_link_libraries(turtle_pose_subscriber ${catkin_LIBRARIES})
```

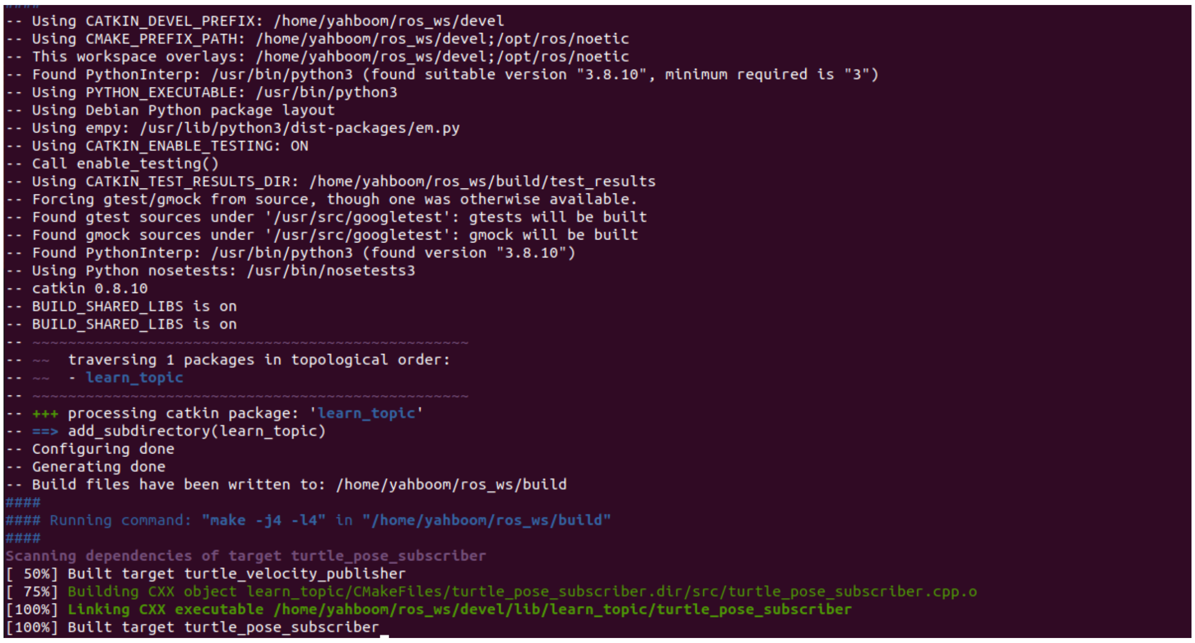
add_executable shows that the generated executable program file is turtle_pose_subscriber, and the compiled source code is turtle_pose_subscriber.cpp in the src directory.

target_link_libraries specifies the libraries that need to be linked when compiling and generating an executable file.

8.2.3 Compile

Terminal input,

```
cd ~/ros_ws
catkin_make
```



```
-- Using CATKIN_DEVEL_PREFIX: /home/yahboom/ros_ws/devel
-- Using CMAKE_PREFIX_PATH: /home/yahboom/ros_ws/devel;/opt/ros/noetic
-- This workspace overlays: /home/yahboom/ros_ws/devel;/opt/ros/noetic
-- Found PythonInterp: /usr/bin/python3 (found suitable version "3.8.10", minimum required is "3")
-- Using PYTHON_EXECUTABLE: /usr/bin/python3
-- Using Debian Python package layout
-- Using empy: /usr/lib/python3/dist-packages/em.py
-- Using CATKIN_ENABLE_TESTING: ON
-- Call enable_testing()
-- Using CATKIN_TEST_RESULTS_DIR: /home/yahboom/ros_ws/build/test_results
-- Forcing gtest/gmock from source, though one was otherwise available.
-- Found gtest sources under '/usr/src/googletest': gtests will be built
-- Found gmock sources under '/usr/src/googletest': gmock will be built
-- Found PythonInterp: /usr/bin/python3 (found version "3.8.10")
-- Using Python nosetests: /usr/bin/nosetests3
-- catkin 0.8.10
-- BUILD_SHARED_LIBS is on
-- BUILD_SHARED_LIBS is on
-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-- ~~  traversing 1 packages in topological order:
-- ~~  - learn_topic
-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-- +++ processing catkin package: 'learn_topic'
-- ==> add_subdirectory(learn_topic)
-- Configuring done
-- Generating done
-- Build files have been written to: /home/yahboom/ros_ws/build
####
#### Running command: "make -j4 -l4" in "/home/yahboom/ros_ws/build"
####
Scanning dependencies of target turtle_pose_subscriber
[ 50%] Built target turtle_velocity_publisher
[ 75%] Building CXX object learn_topic/CMakeFiles/turtle_pose_subscriber.dir/src/turtle_pose_subscriber.cpp.o
[100%] Linking CXX executable /home/yahboom/ros_ws/devel/lib/learn_topic/turtle_pose_subscriber
[100%] Built target turtle_pose_subscriber
```

After the compilation is passed, you need to re-source the current environment variables to find or update the program. Enter in the terminal.

```
cd ~/ros_ws
source devel/setup.bash
```

8.2.4 Running the program

Run roscore

```
roscore
```

Run the little turtle node

```
rosrun turtlesim turtlesim_node
```

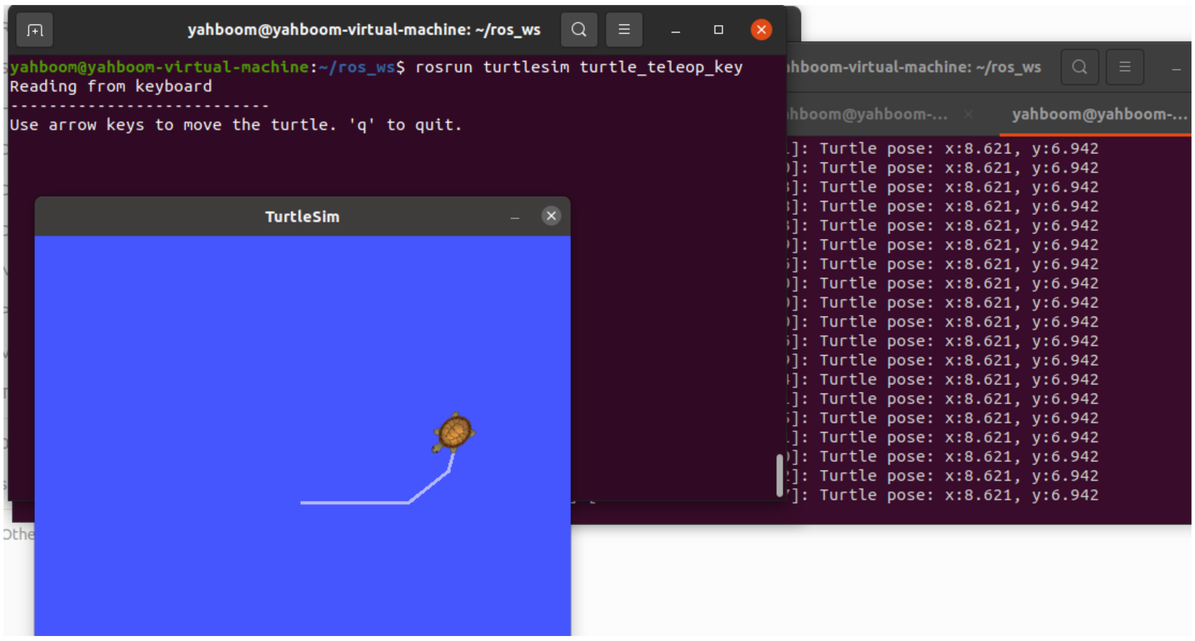
Run the turtle speed control node

```
rosrun turtlesim turtle_teleop_key
```

Run the subscriber node to continuously receive the pose data sent by the little turtle.

```
rosrun learn_topic turtle_pose_subscriber
```

After all programs are running, click on the terminal running the turtle speed control node and press [up, down, left, and right] on the keyboard to control the movement of the turtle. The terminal running the subscriber will print the xy coordinate value of the turtle to the terminal in real time.As shown below,



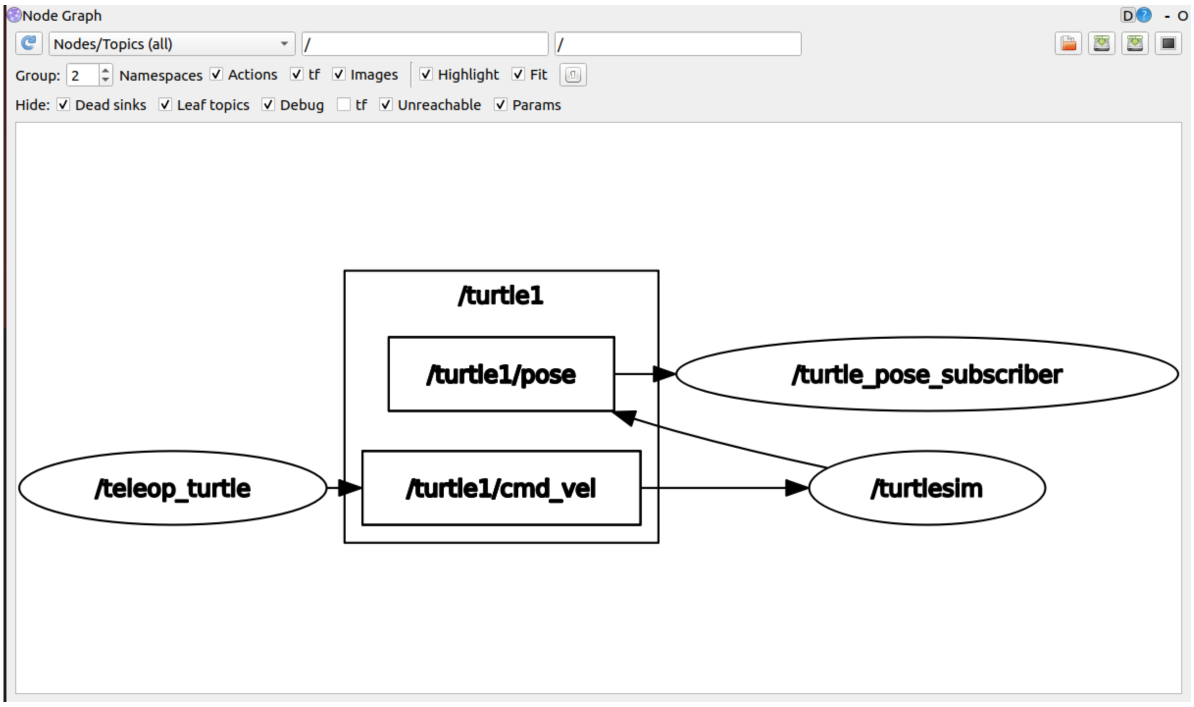
```
yahboom@yahboom-virtual-machine:~/ros_ws$ rosrun turtlesim turtle_teleop_key
Reading from keyboard
---------------------------
Use arrow keys to move the turtle. 'q' to quit.
```

```
]: Turtle pose: x:8.621, y:6.942
]: Turtle pose: x:8.621, y:6.942
]: Turtle pose: x:8.621, y:6.942
]: Turtle pose: x:8.621, y:6.942
]: Turtle pose: x:8.621, y:6.942
]: Turtle pose: x:8.621, y:6.942
]: Turtle pose: x:8.621, y:6.942
]: Turtle pose: x:8.621, y:6.942
]: Turtle pose: x:8.621, y:6.942
]: Turtle pose: x:8.621, y:6.942
]: Turtle pose: x:8.621, y:6.942
]: Turtle pose: x:8.621, y:6.942
]: Turtle pose: x:8.621, y:6.942
]: Turtle pose: x:8.621, y:6.942
]: Turtle pose: x:8.621, y:6.942
]: Turtle pose: x:8.621, y:6.942
]: Turtle pose: x:8.621, y:6.942
]: Turtle pose: x:8.621, y:6.942
]: Turtle pose: x:8.621, y:6.942
```

yahboom@yahboom-virtual-machine: ~/ros_ws
TurtleSim
hboom-virtual-machine: ~/ros_ws
hboom@yahboom-...
yahboom@yahboom-...

We can use the rqt_graph tool to view the communication between nodes and terminal input.

```
rqt_graph
```



The inside of the ellipse represents the node name, the inside of the rectangle represents the topic name, and the arrow represents the direction of topic message transmission. The subscriber node program turtle_pose_subscriber we wrote is subscribed to the topic message of /turtle1/pose.This is consistent with the code `ros::Subscriber pose_sub = n.subscribe("/turtle1/pose", 10, turtle_poseCallback);`, and then in the callback function, print the data, the code is as follows,

```
void turtle_poseCallback(const turtlesim::Pose::ConstPtr& msg)
{
    // Print received messages
    ROS_INFO("Turtle pose: x:%0.3f, y:%0.3f", msg->x, msg->y);
}
```

8.2.5 Program flow chart

8.3 Python version

8.3.1 Writing source code

Create a new python file (file suffix .py) in the scripts folder under the function package learn_topic, name it turtle_pose_subscriber.py, copy and paste the following program code into the turtle_pose_subscriber.py file,

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import rospy
from turtlesim.msg import Pose
def poseCallback(msg):
    rospy.loginfo("Turtle pose: x:%0.3f, y:%0.3f", msg.x, msg.y)
def turtle_pose_subscriber():
    rospy.init_node('turtle_pose_subscriber', anonymous=True)# ROS node
initialization
    # Create a Subscriber, subscribe to the topic named /turtle1/pose, and
register the callback function poseCallback
    rospy.Subscriber("/turtle1/pose", Pose, poseCallback)
    rospy.spin()# Loop waiting for callback function
if __name__ == '__main__':
    turtle_pose_subscriber()
```

The python program does not need to be compiled, but it needs to add executable permissions and enter it in the terminal.

```
cd ~/ros_ws/src/learn_topic/scripts
sudo chmod a+x turtle_pose_subscriber.py
```

8.3.2 Run

Open roscore,

```
roscore
```

Run the little turtle node,

```
rosrun turtlesim turtlesim_node
```

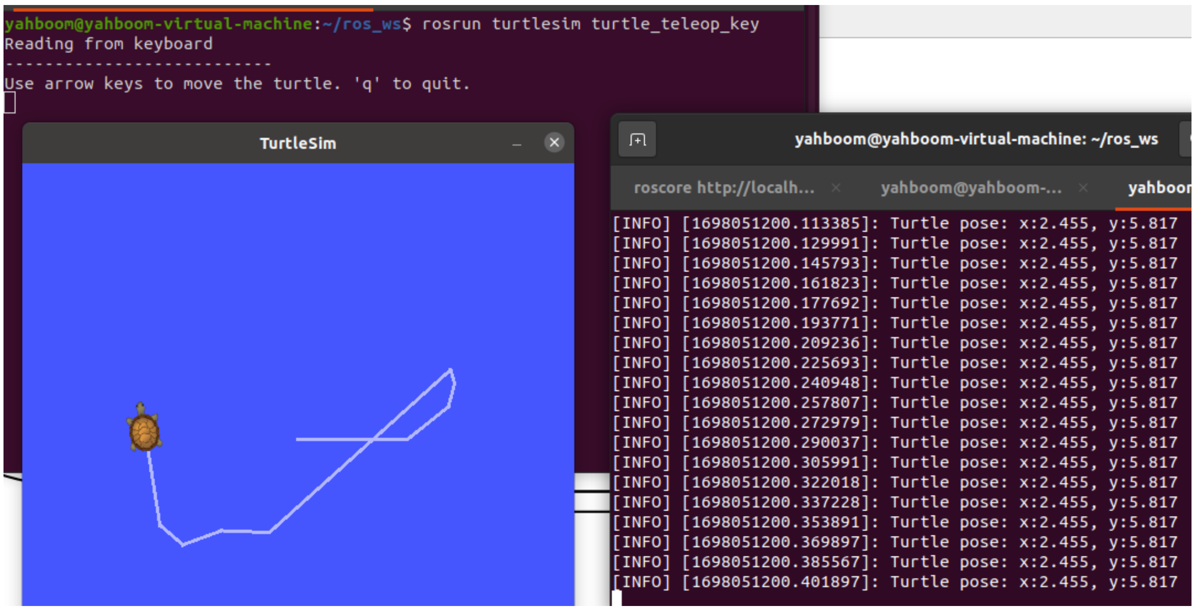
Run the turtle speed control node,

```
rosrun turtlesim turtle_teleop_key
```

Run the subscriber node and continue to receive the pose data sent by the little turtle.

```
rosrun learn_topic turtle_pose_subscriber.py
```

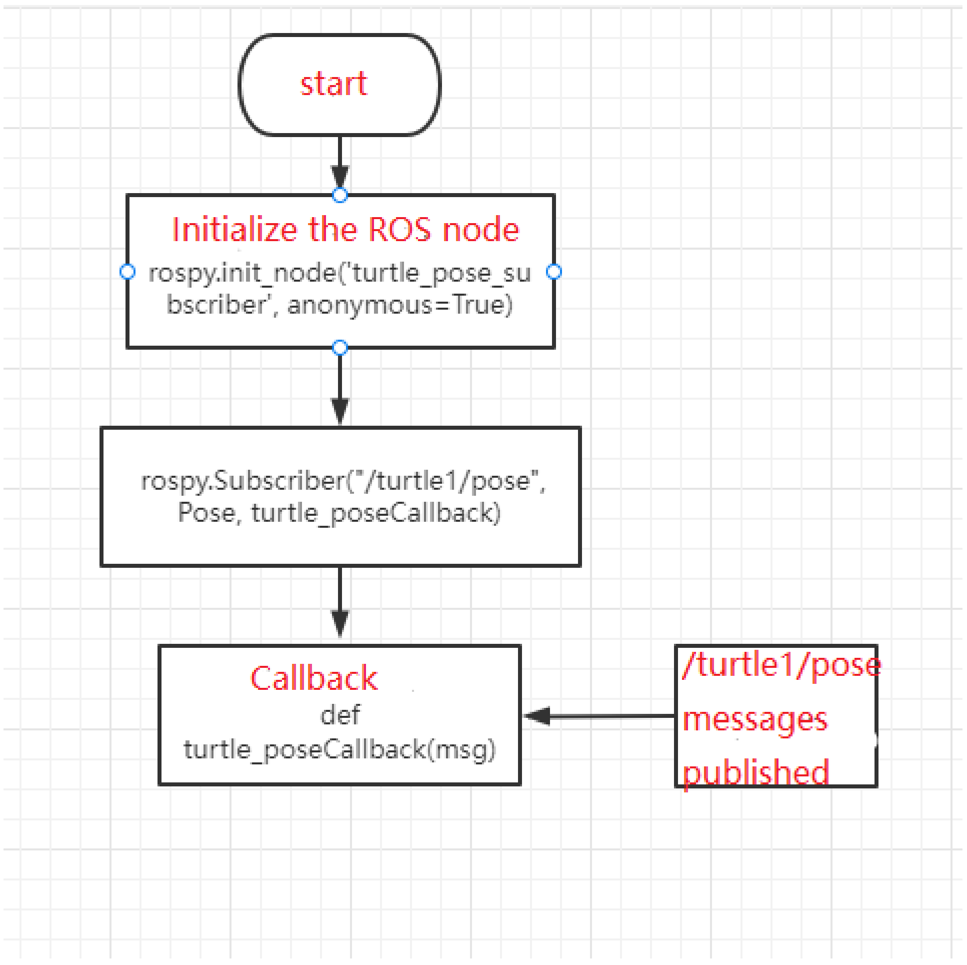
After all programs are running, click on the terminal running the turtle speed control node and press [up, down, left, and right] on the keyboard to control the movement of the turtle. The terminal running the subscriber will print the xy coordinate value of the turtle to the terminal in real time.As shown below,



```
yahboom@yahboom-virtual-machine:~/ros_ws$ rosrun turtlesim turtle_teleop_key
Reading from keyboard
---------------------------
Use arrow keys to move the turtle. 'q' to quit.
```

```
[INFO] [1698051200.113385]: Turtle pose: x:2.455, y:5.817
[INFO] [1698051200.129991]: Turtle pose: x:2.455, y:5.817
[INFO] [1698051200.145793]: Turtle pose: x:2.455, y:5.817
[INFO] [1698051200.161823]: Turtle pose: x:2.455, y:5.817
[INFO] [1698051200.177692]: Turtle pose: x:2.455, y:5.817
[INFO] [1698051200.193771]: Turtle pose: x:2.455, y:5.817
[INFO] [1698051200.209236]: Turtle pose: x:2.455, y:5.817
[INFO] [1698051200.225693]: Turtle pose: x:2.455, y:5.817
[INFO] [1698051200.240948]: Turtle pose: x:2.455, y:5.817
[INFO] [1698051200.257807]: Turtle pose: x:2.455, y:5.817
[INFO] [1698051200.272979]: Turtle pose: x:2.455, y:5.817
[INFO] [1698051200.290037]: Turtle pose: x:2.455, y:5.817
[INFO] [1698051200.305991]: Turtle pose: x:2.455, y:5.817
[INFO] [1698051200.322018]: Turtle pose: x:2.455, y:5.817
[INFO] [1698051200.337228]: Turtle pose: x:2.455, y:5.817
[INFO] [1698051200.353891]: Turtle pose: x:2.455, y:5.817
[INFO] [1698051200.369897]: Turtle pose: x:2.455, y:5.817
[INFO] [1698051200.385567]: Turtle pose: x:2.455, y:5.817
[INFO] [1698051200.401897]: Turtle pose: x:2.455, y:5.817
```

8.3.3 Program flow chart


start
Initialize the ROS node
rospy.init_node('turtle_pose_subscriber', anonymous=True)
rospy.Subscriber("/turtle1/pose", Pose, turtle_poseCallback)
Callback
def turtle_poseCallback(msg)
/turtle1/pose messages published