

# 5 Subscribers

---

## 5 Subscribers

### 5.1 Subscribers

#### 5.2 Create a Subscriber

##### 5.2.1 Creation steps

##### 5.2.2 C++ language implementation

##### 5.2.3 Python language implementation

## 5.1 Subscribers

The subscriber receives the data published by the publisher, and then enters its callback function and processes the received data in the callback function. Its core content is a callback function, and each subscriber subscribes to a topic that has a callback function.

## 5.2 Create a Subscriber

### 5.2.1 Creation steps

1. initialize the ROS node
2. create a handle
3. subscribe to the topic you need
4. wait for the topic message in a loop, and enter the callback function after receiving the message
5. complete the message processing in the callback function.

### 5.2.2 C++ language implementation

1. In the "Publisher" tutorial, create a new c++ file under the src folder of the created function package and name it turtle\_pose\_subscriber.cpp
2. Copy and paste the program code below into the turtle\_pose\_subscriber.cpp file

```
/*Create a small turtle's current pose information to receive*/
#include <ros/ros.h>
#include "turtlesim/Pose.h"
// After receiving the message, it will enter the message callback function, and
the callback function will process the received data
void turtle_poseCallback(const turtlesim::Pose::ConstPtr & msg){
    // print received message
    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 the ROS node

    ros::NodeHandle n; //here is to create handle

    // Create a subscriber, the subscribed topic is the topic of /turtle1/pose,
    and poseCallback is the callback function
```

```

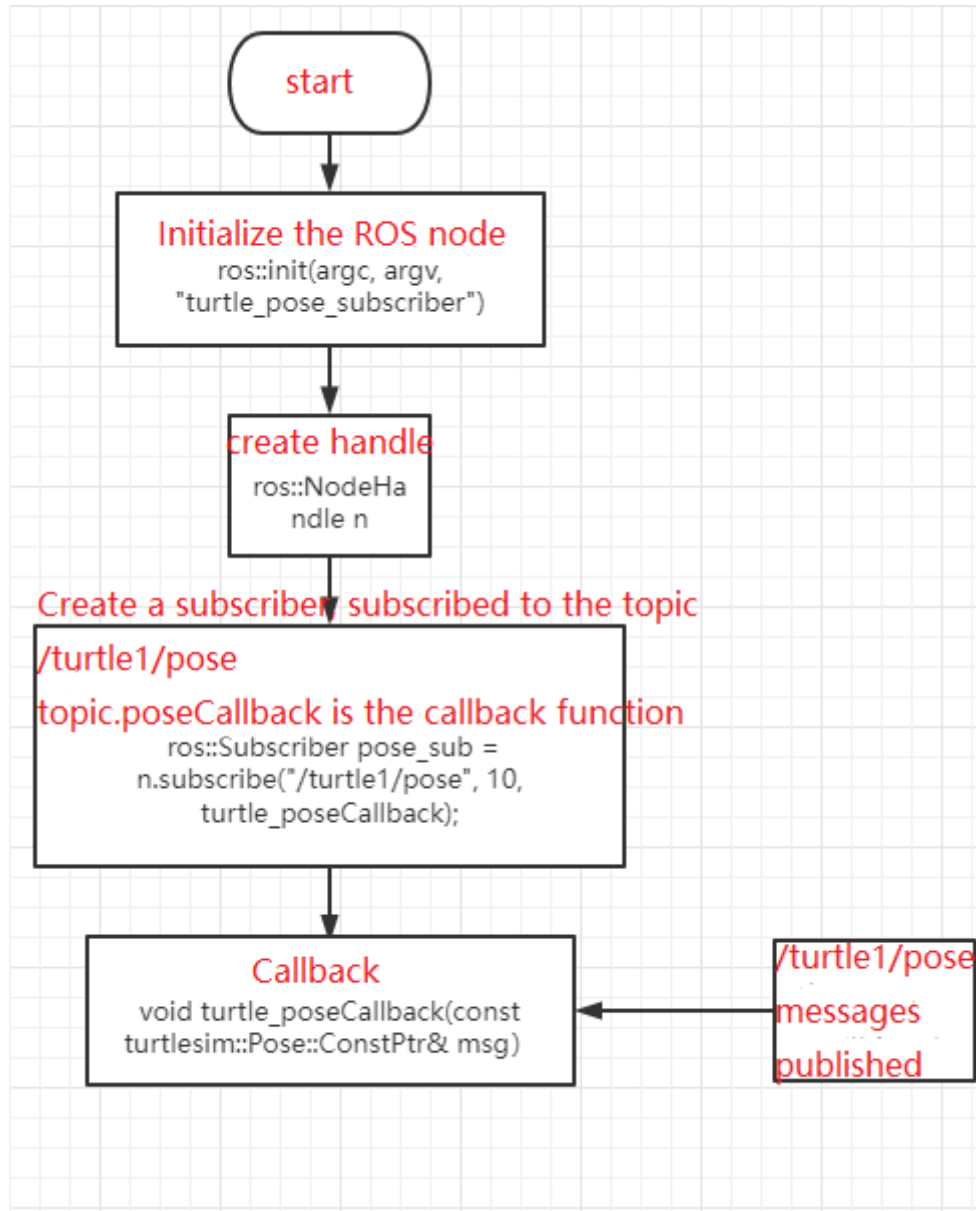
ros::Subscriber pose_sub = n.subscribe("/turtle1/pose", 10,
turtle_poseCallback);

ros::spin(); // loop waiting for callback function

return 0;
}

```

3. Program flow chart, which can be viewed corresponding to the content of 5.2.1



4. 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})

```

5. Compile the code in the workspace directory

```

cd ~/catkin_ws
catkin_make
source devel/setup.bash #Environment variables need to be configured,
otherwise the system cannot find the running program

```

6. run the program

- run roscore

```
roscore
```

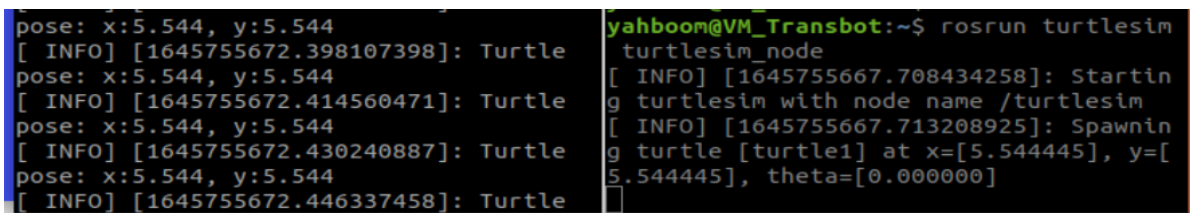
- Run the turtle node

```
roslaunch turtlesim turtlesim_node
```

- Run the subscription to continuously receive the pose data sent by the turtle

```
roslaunch learning_topic turtle_pose_subscriber
```

7. run the screenshot



The screenshot shows two terminal windows. The left window displays a series of turtle pose messages: 'pose: x:5.544, y:5.544' followed by '[ INFO] [1645755672.398107398]: Turtle pose: x:5.544, y:5.544' and similar messages. The right window shows the output of running 'roslaunch turtlesim turtlesim\_node', including 'Starting turtlesim with node name /turtlesim' and 'Spawning turtle [turtle1] at x=[5.544445], y=[5.544445], theta=[0.000000]'.

8. program operation instructions

- After running the node of the turtle, the turtle will continuously send its own pose information, and the topic name sent is

/turtle1/pose

- After turtle\_pose\_subscriber runs, it will receive the data messages sent by the turtle, and then print the information in the callback function.

### 5.2.3 Python language implementation

1. In the function package directory, create a new folder scripts, and then create a new python file(file suffix .py) in the scripts folder, named turtle\_pose\_subscriber.py
2. 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, register
    the callback function poseCallback
    rospy.Subscriber("/turtle1/pose", Pose, poseCallback)
```

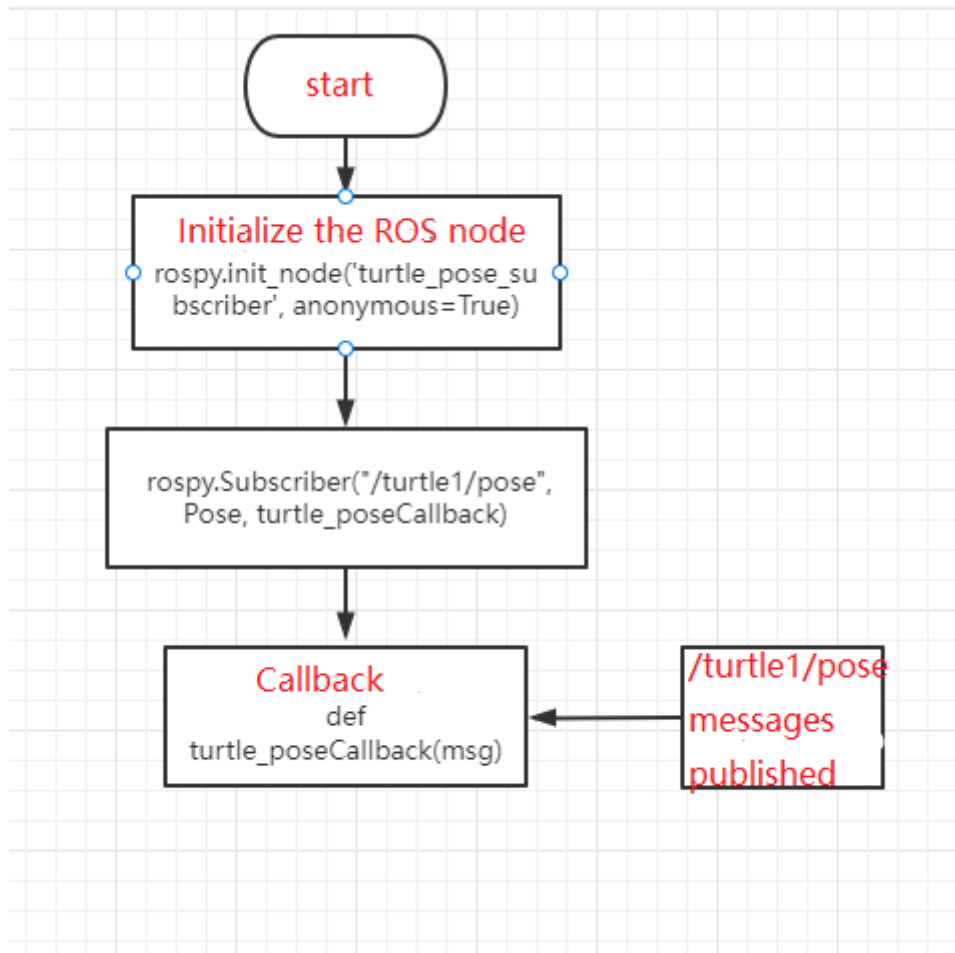
```

rospy.spin() # loop waiting for the callback function

if __name__ == '__main__':
    turtle_pose_subscriber()

```

### 3. program flow chart



### 4. run the program

- run roscore

```
roscore
```

- Run the turtle node

```
roslaunch turtlesim turtlesim_node
```

- Run the subscriber to continuously receive the pose data sent by the turtle

```
roslaunch learning_topic turtle_pose_subscriber.py
```

### 5. Refer to 5.2.2 for operation effect and program description.