7.ROS topic publisher

As mentioned in the previous section, a node program has both publishing and subscribing messages. In this section, we will explain how to declare a publisher in the node program and publish topic messages. We proceed based on the previously established workspace ros_ws and learn_topic function packages.

7.1 Create a publisher

General creation steps are as follows:

- Initialize ROS nodes
- Create handle
- Register node information with ROS Master, including the published topic name, message type in the topic, and queue length
- Create and initialize message data
- Send messages cyclically with a certain frequency

7.2 C++ version

7.2.1 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_velocity_publisher.cpp, and paste the following content into turtle_velocity_publisher.cpp.

```
/*Create a small turtle speed publisher*/
#include <ros/ros.h>
#include <geometry_msgs/Twist.h>
int main(int argc, char **argv)
{
    ros::init(argc, argv, "turtle_velocity_publisher");//ROS node initialization
    ros::NodeHandle n;//Here is create handle
    //Create a Publisher, publish a topic named /turtle1/cmd_vel, the message
type is geometry_msgs::Twist, and the queue length is 10
    ros::Publisher turtle_vel_pub = n.advertise<geometry_msgs::Twist>
    ("/turtle1/cmd_vel", 10);
    ros::Rate loop_rate(10);//Set the frequency of the loop
    while (ros::ok())
    {
        //Initialize the message to be published, the type must be consistent
with Publisher
        geometry_msgs::Twist turtle_vel_msg;
        turtle_vel_msg.linear.x = 0.8;
        turtle_vel_msg.angular.z = 0.6;
        turtle_vel_pub.publish(turtle_vel_msg);// Publish speed news
        //Print published speed content
        ROS_INFO("Publsh turtle velocity command[%0.2f m/s, %0.2f rad/s]",
        turtle_vel_msg.linear.x, turtle_vel_msg.angular.z);
        loop_rate.sleep();//Delay according to cycle frequency
    }
    return 0;
}
```

7.2.2 Modify CMakelist.txt file

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

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

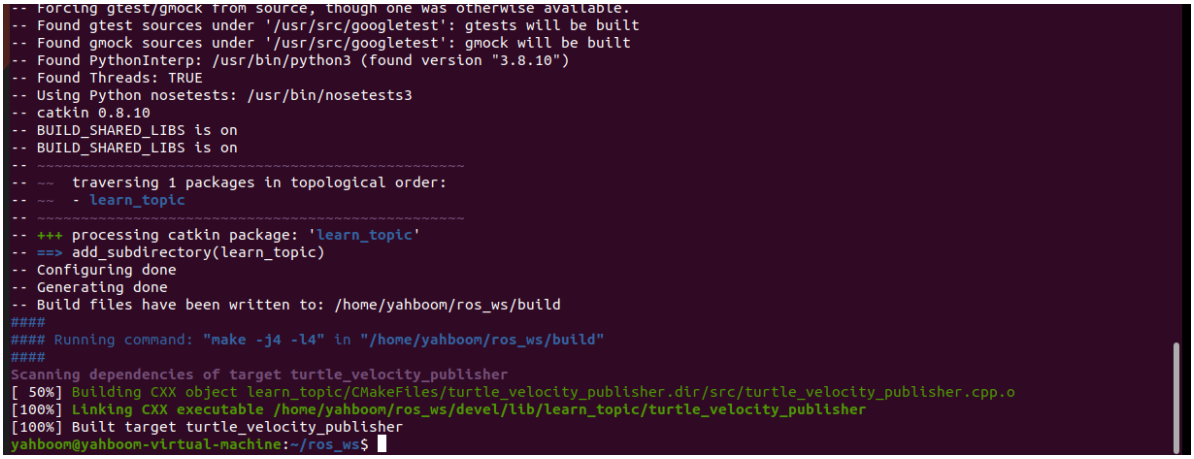
add_executable shows that the generated executable program file is turtle_velocity_publisher, and the compiled source code is turtle_velocity_publisher.cpp in the src directory.

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

7.2.3 Compile

Terminal input,

```
cd ~/ros_ws
catkin_make
```



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
```

7.2.4 Running the program

Open roscore,

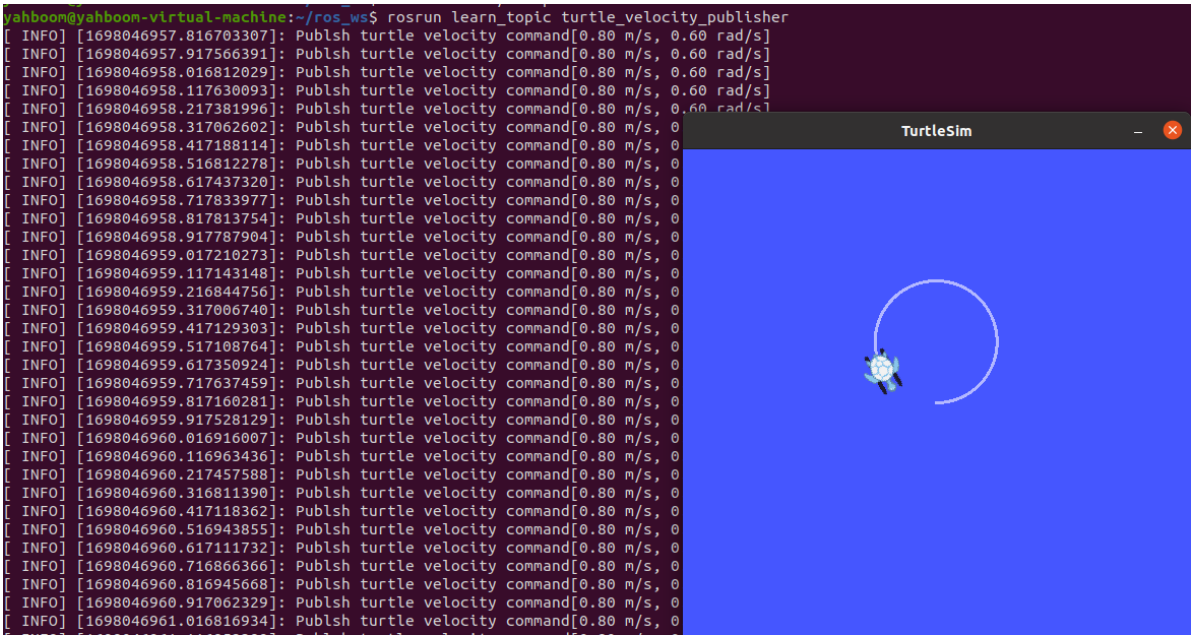
```
roscore
```

Run the little turtle node program,

```
rosrun turtlesim turtlesim_node
```

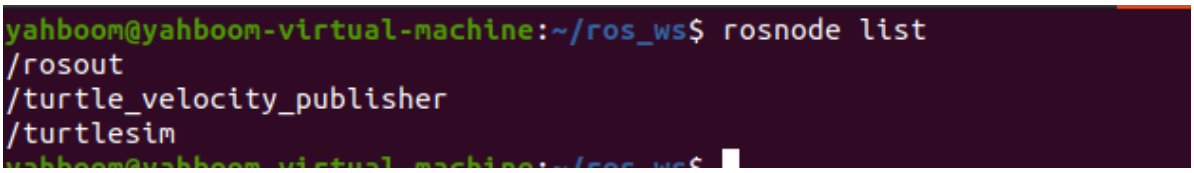
Run the publisher node program and continue to send speed to the little turtle.

```
rosrun learn_topic turtle_velocity_publisher
```



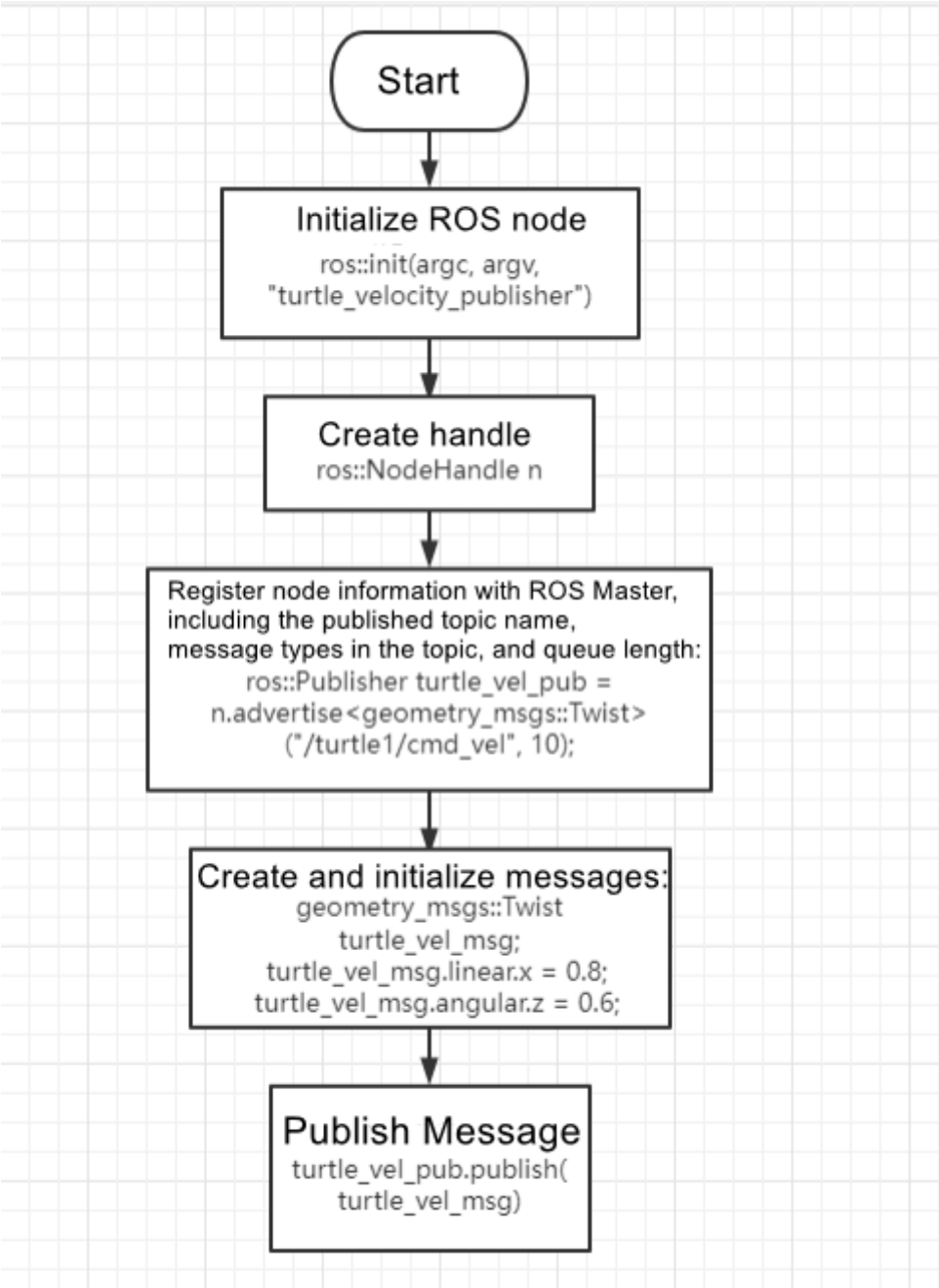
As shown in the figure above, after receiving the published message, the little turtle will move at the specified speed. We can check which nodes are running through rosnode list and enter in the terminal,

```
rosnode list
```



/turtle_velocity_publisher is the program we write, compile, and run. The node name here is consistent with `ros::init(argc, argv, "turtle_velocity_publisher")` in the code.

7.2.5 Program flow chart



7.3 Python version

7.3.1 Writing source code

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

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# This routine will publish to the turtle1/cmd_vel topic, message type
geometry_msgs::Twist
import rospy
from geometry_msgs.msg import Twist
def turtle_velocity_publisher():
    rospy.init_node('turtle_velocity_publisher', anonymous=True) # ROS node
initialization
```

```
    # Create a small turtle speed publisher and publish a topic named
/turtle1/cmd_vel. The message type is geometry_msgs::Twist, and 8 represents the
message queue length.
    turtle_vel_pub = rospy.Publisher('/turtle1/cmd_vel', Twist, queue_size=8)
    rate = rospy.Rate(10) #Set the frequency of the loop
    while not rospy.is_shutdown():
        # Initialize geometry_msgs::Twist type message
        turtle_vel_msg = Twist()
        turtle_vel_msg.linear.x = 0.8
        turtle_vel_msg.angular.z = 0.6
        # release the news
        turtle_vel_pub.publish(turtle_vel_msg)
        rospy.loginfo("linear is :%0.2f m/s, angular is :%0.2f rad/s",
        turtle_vel_msg.linear.x, turtle_vel_msg.angular.z)
        rate.sleep()# Delay according to cycle frequency
if __name__ == '__main__':
    try:
        turtle_velocity_publisher()
    except rospy.ROSInterruptException:
        pass
```

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_velocity_publisher.py
```

7.3.2 Run

Open roscore,

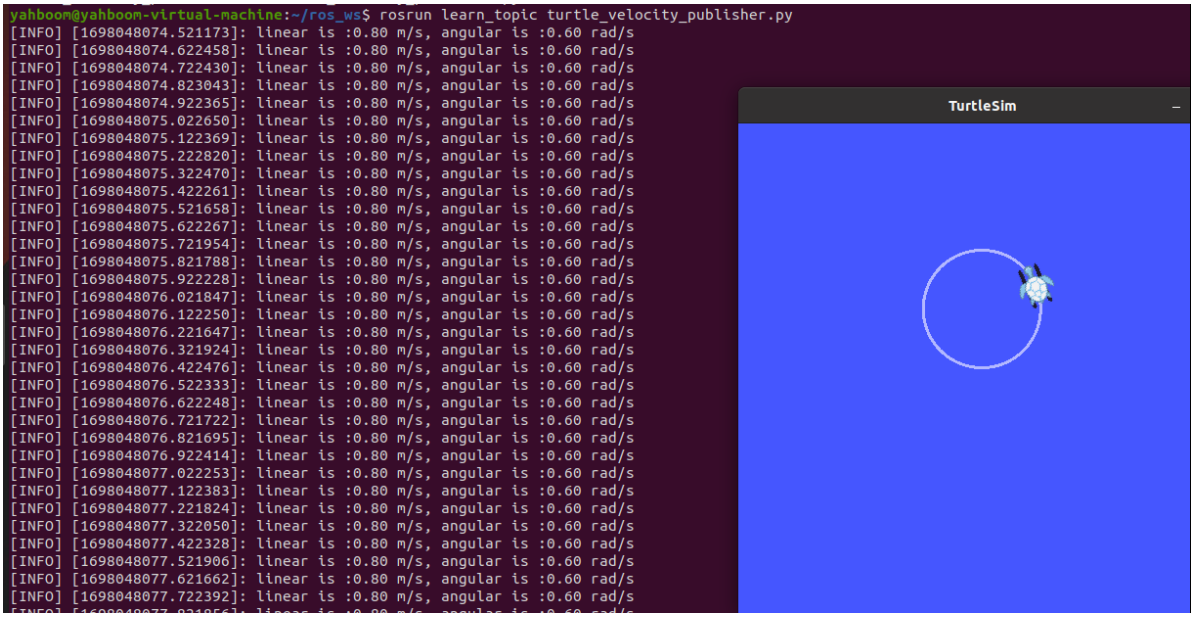
```
roscore
```

Run the little turtle node,

```
rosrun turtlesim turtlesim_node
```

Run the publisher node program and continue to send speed to the little turtle.

```
rosrun learn_topic turtle_velocity_publisher.py
```



7.3.3 Program flow chart