

# 智能机器人系统实验报告

专 业 班 级

计算机创新实验 18-1 班

学生姓名及学号

2018213106 刘嘉伟

任 课 教 师

方宝富老师

实验指导教师

方宝富老师

实 验 地 点

D502

2020 ~2021 学年第二学期

# 实验 1 熟悉 ROS 操作系统、学习 ROS 常用指令

## 一、实验目的

掌握 ROS 的安装以及常用命令，具体内容如下：

- (1) 学会在 ubuntu18.04 系统下安装 ROS。
- (2) 了解 ROS 基础命令。
- (3) 运行海龟示例。

## 二、实验设备

硬件环境：DELL 笔记本

系统环境：VmWare、Ubuntu18.04、ROS Melodic

## 三、实验内容

### 1、安装 ROS

#### (a) 设置安装源

```
$ sudo sh -c echo deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main  
/etc/apt/sources.list.d/ros-latest.list
```

#### (b) 设置 Key

```
$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-key  
C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

#### (c) 安装

```
$ sudo apt update  
$ sudo apt install ros-melodic-desktop-full
```

#### (d) 配置环境变量

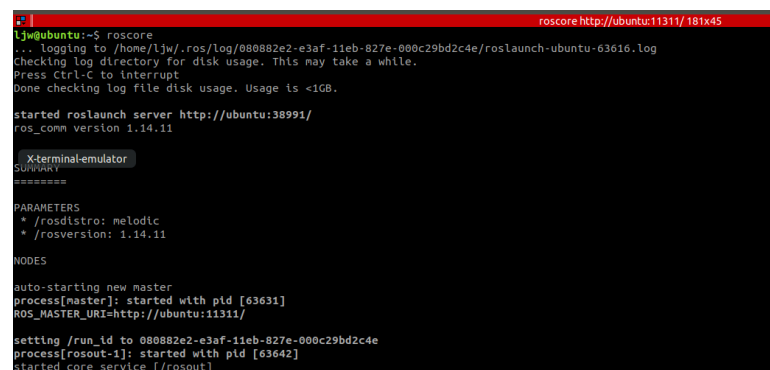
```
$ echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc  
$ source ~/.bashrc
```

#### (e) 安装依赖

```
$ sudo apt install python-rosinstall python-rosinstall-generator python-wstool build-essential
```

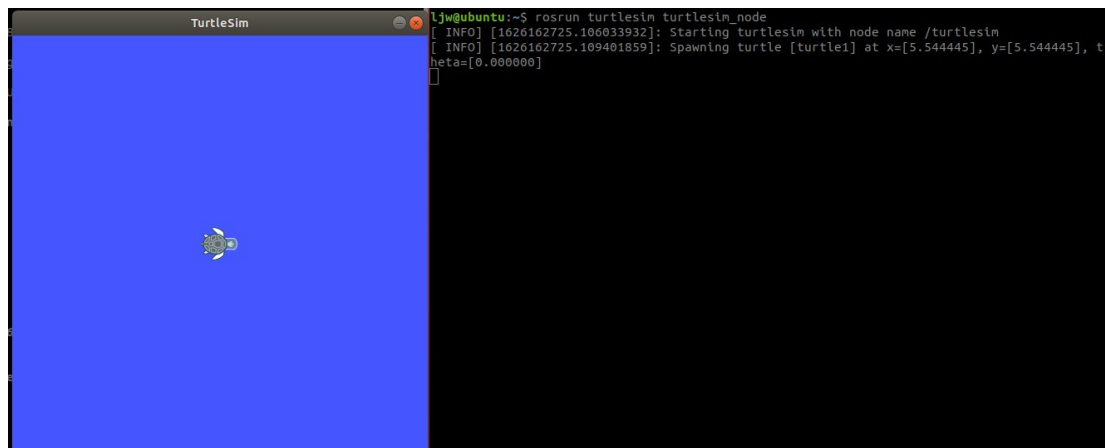
### 2、ROS 常用指令以及小海龟示例

#### ● roscore 命令



```
ljj@ubuntu:~$ roscore  
... logging to /home/ljj/.ros/log/080882e2-e3af-11eb-827e-000c29bd2c4e/roslaunch-ubuntu-63616.log  
Checking log directory for disk usage. This may take a while.  
Press Ctrl-C to interrupt  
Done checking log file disk usage. Usage is <1GB.  
  
started roslaunch server http://ubuntu:38991/  
ros_comm version 1.14.11  
  
X-terminal-emulator  
SUMMARY  
=====  
  
PARAMETERS  
* /rostdistro: melodic  
* /rosversion: 1.14.11  
  
NODES  
  
auto-starting new master  
process[master]: started with pid [63631]  
ROS_MASTER_URI=http://ubuntu:11311/  
  
setting /run_id to 080882e2-e3af-11eb-827e-000c29bd2c4e  
process[roscout-1]: started with pid [63642]  
started core service [/roscout]
```

- rosrn 命令



- rosnod 命令

```
ljwt@ubuntu:~$ rosnod list
/rosout
ljwt@ubuntu:~$
```

- roscd 命令

```
ljwt@ubuntu:~$ roscd turtlesim/
ljwt@ubuntu:/opt/ros/melodic/share/turtlesim$ pwd
/opt/ros/melodic/share/turtlesim
ljwt@ubuntu:/opt/ros/melodic/share/turtlesim$
```

- rostopic 命令

```
ljwt@ubuntu:/opt/ros/melodic/share/turtlesim$ rostopic list
/rosout
/rosout_agg
ljwt@ubuntu:/opt/ros/melodic/share/turtlesim$
```

- rosservice 命令

```
ljwt@ubuntu:/opt/ros/melodic/share/turtlesim$ rosservice list
/rosout/get_loggers
/rosout/set_logger_level
ljwt@ubuntu:/opt/ros/melodic/share/turtlesim$
```

- roslaunch 命令

```
ljwt@ubuntu:/opt/ros/melodic/share/turtlesim$ roslaunch gazebo_ros empty_world.launch
... logging to /home/ljwt/.ros/log/080882e2-e3af-11eb-827e-000c29bd2c4e/roslaunch-ubuntu-64
545.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ubuntu:41173/

SUMMARY
=====

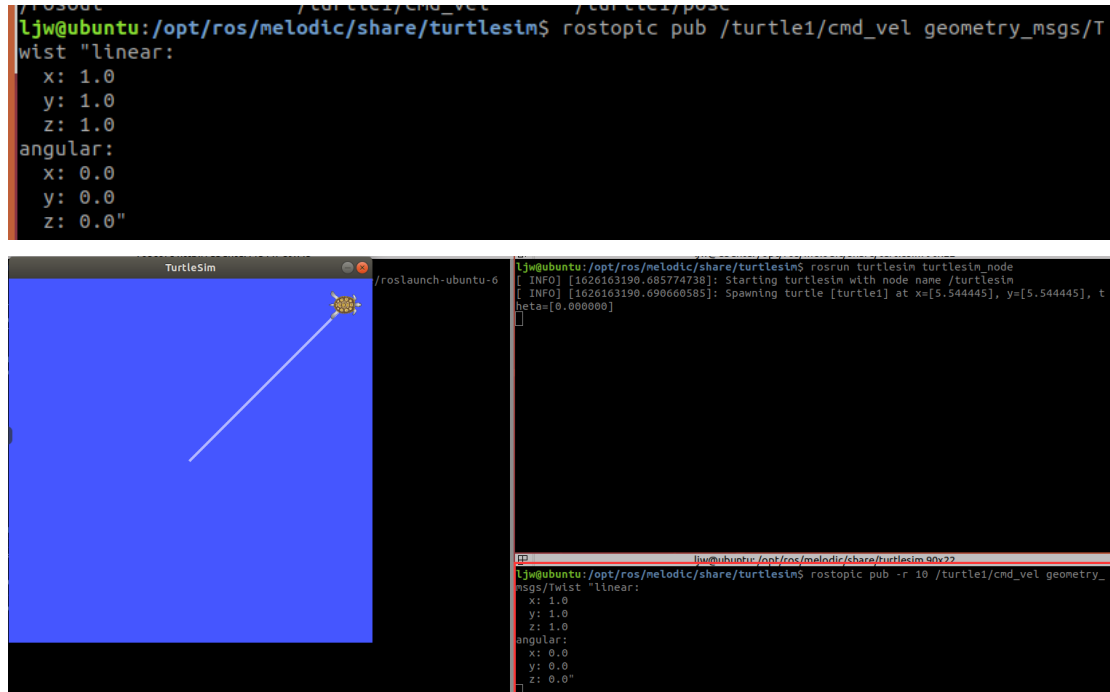
PARAMETERS
* /gazebo/enable_ros_network: True
* /rostdistro: melodic
* /rosversion: 1.14.11
* /use_sim_time: True

NODES
```

## 四、动手实现

动手实现 1: 通过手动给/turtle1/cmd\_vel 话题发送消息, 控制小海龟移动。

方法: 通过 rostopic 的 pub 命令来向/turtle1/cmd\_vel 发送运动消息, 就能使海龟运动。

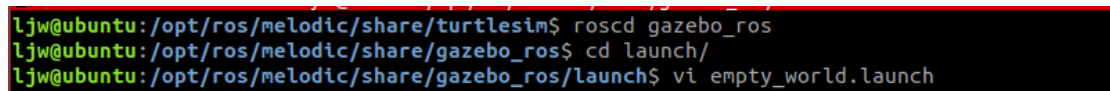


The screenshot displays a ROS environment with three windows. The top terminal window shows the command `rostopic pub /turtle1/cmd_vel geometry_msgs/Twist "linear: {x: 1.0, y: 1.0, z: 1.0}, angular: {x: 0.0, y: 0.0, z: 0.0}"` being executed. The middle window is the TurtleSim visualization, showing a blue square environment with a small orange turtle icon and a white line indicating its path. The bottom terminal window shows the command `rostopic pub -r 10 /turtle1/cmd_vel geometry_msgs/Twist "linear: {x: 1.0, y: 1.0, z: 1.0}, angular: {x: 0.0, y: 0.0, z: 0.0}"` being executed, along with ROS log messages indicating the start of the turtlesim node and the spawning of the turtle.

动手实现 2: 当使用 roslaunch 启动节点后, 再运行 roscore 会报错, 因为只能同时开启一个 roscore, 而 roslaunch 会自动打开 roscore, 所以如果再运行一个 roscore 将会报错。

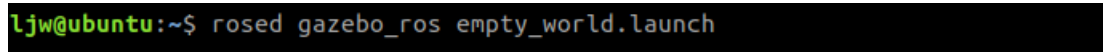
动手实现 3:

通过 roscd 进入文件夹下修改 empty\_world.launch



The screenshot shows a terminal window with the following commands: `roscd gazebo_ros`, `cd launch/`, and `vi empty_world.launch`.

通过 rosed 直接修改 empty\_world.launch



The screenshot shows a terminal window with the command `rostd edit empty_world.launch` being executed.

## 实验 2 熟悉 ROS 话题通信架构，实现话题通信过程

### 一、实验目的

- (1) 了解 ROS 节点间的话题通信过程
- (2) 编写节点实现话题通信

### 二、实验设备

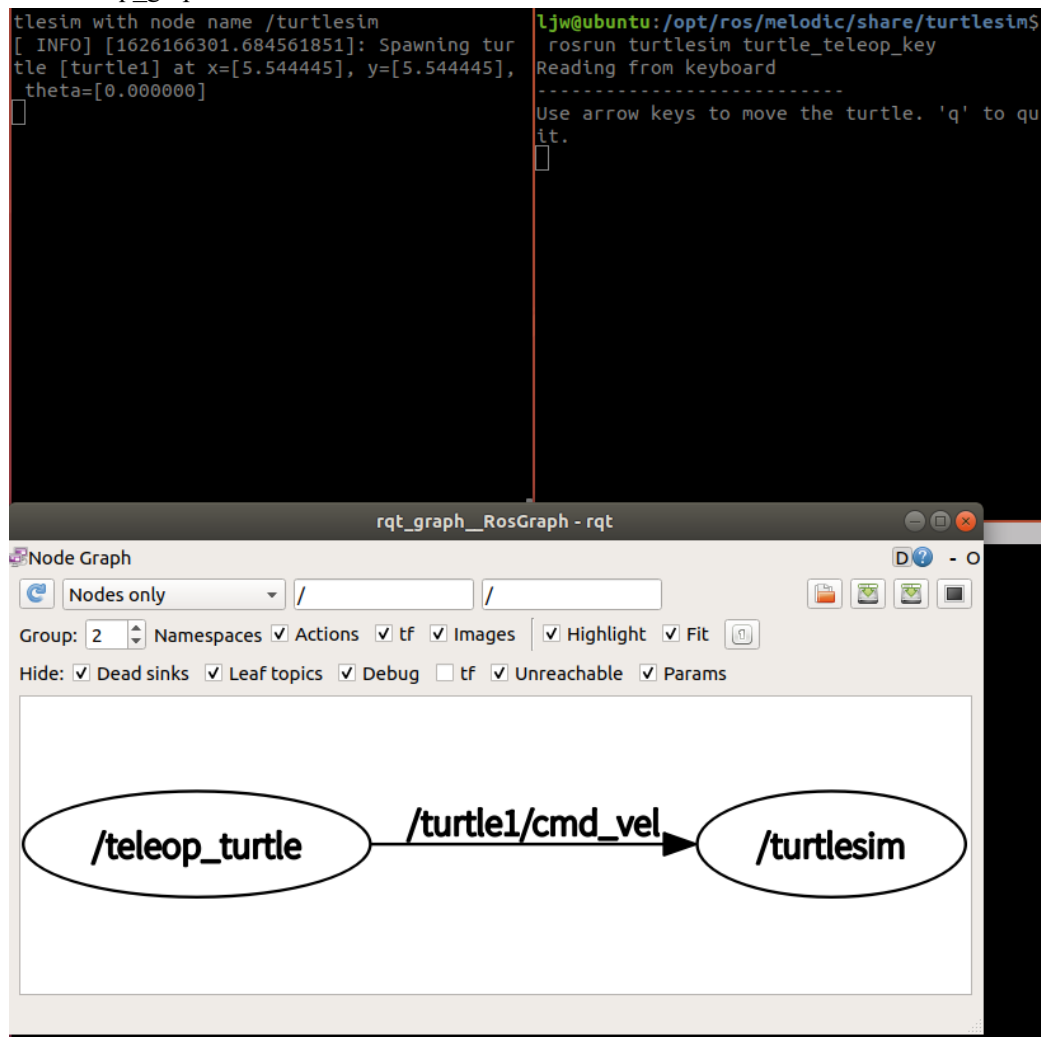
硬件环境：DELL 笔记本

系统环境：VmWare、Ubuntu18.04、ROS Melodic

### 三、实验内容

- (1) 了解 ROS 节点间的通信过程：

- 使用 `rqt_graph` 查看 ROS 节点关系



- 使用 rostopic 查看话题信息

```
ljw@ubuntu:~$ rostopic info /turtle1/cmd_vel
Type: geometry_msgs/Twist

Publishers:
* /teleop_turtle (http://ubuntu:37227/)

Subscribers:
* /turtlesim (http://ubuntu:44991/)
```

## (2) 编写节点实现话题通信

- C++版本

### talker.cpp

```
1  #include "ros/ros.h"
2  #include "std_msgs/String.h"
3  #include <sstream>
4
5  int main(int argc, char *argv[])
6  {
7      //设置中文编码
8      setlocale(LC_ALL, "");
9
10     ros::init(argc, argv, "talker");
11     ros::NodeHandle nh;
12
13     //初始化一个发布对象
14     ros::Publisher pub = nh.advertise<std_msgs::String>("chatter", 10);
15     //存储发布的消息
16     std_msgs::String msg;
17
18     //一个前缀
19     std::string front_msg = "hello,你好呀! ";
20
21     //发布计数
22     int count = 0;
23
24     //以rate的频率发送消息
25     ros::Rate rate(1);
26
27     ros::Duration(3).sleep();
28     while(ros::ok())
29     {
30         //拼接字符串啊
31         std::stringstream ss;
32
33         ss << front_msg << count;
34         //给消息赋值并发布
35         msg.data = ss.str();
36         pub.publish(msg);
37
38         //消息提示
39         ROS_INFO("发布的消息是: %s", ss.str().c_str());
40
41         //按周期发布
42         count++;
43         rate.sleep();
44         ros::spinOnce();
45     }
46     return 0;
47 }
```

## listener.cpp

```
1  #include "ros/ros.h"
2  #include <sstream>
3  #include "std_msgs/String.h"
4
5  void doMsg(const std_msgs::String::ConstPtr &msg_p) {
6      ROS_INFO("收到的数据是: %s", msg_p->data.c_str());
7  }
8
9  int main(int argc, char *argv[])
10 {
11     //设置中文编码
12     setlocale(LC_ALL, "");
13     ros::init(argc, argv, "listener");
14     ros::NodeHandle nh;
15     //获取订阅对象
16     ros::Subscriber sub = nh.subscribe("chatter", 10, doMsg);
17
18     //回调函数
19     ros::spin();
20     return 0;
21 }
```

## ● Python 版本

### talker.py

```
1  #!/usr/bin/env python
2  import rospy
3  from std_msgs.msg import String
4
5  def talker():
6      pub = rospy.Publisher('chatter', String, queue_size=10)
7      rospy.init_node('talker', anonymous=True)
8      rate = rospy.Rate(10) # 10hz
9      while not rospy.is_shutdown():
10         hello_str = "hello world %s" % rospy.get_time()
11         rospy.loginfo(hello_str)
12         pub.publish(hello_str)
13         rate.sleep()
14
15
16 if __name__ == '__main__':
17     try:
18         talker()
19     except rospy.ROSInterruptException:
20         pass
```

### listener.py

```
1  #!/usr/bin/env python
2  import rospy
3  from std_msgs.msg import String
4  def talker():
5      pub = rospy.Publisher('chatter', String, queue_size=10)
6      rospy.init_node('talker', anonymous=True)
7      rate = rospy.Rate(10) # 10hz
8      while not rospy.is_shutdown():
9         hello_str = "hello world %s" % rospy.get_time()
10         rospy.loginfo(hello_str)
11         pub.publish(hello_str)
12         rate.sleep()
13
14 if __name__ == '__main__':
15     try:
16         talker()
17     except rospy.ROSInterruptException:
18         pass
```

## 四、动手实践

动手实现 1: 利用实验一所学知识, 查看 `turtle1/cmd_vel` 话题的消息类型 `Twist` 的内容是什么。

```
ljw@ubuntu:~$ rosmmsg show geometry_msgs/Twist
geometry_msgs/Vector3 linear
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 angular
  float64 x
  float64 y
  float64 z
```

动手实现 2: 利用所学知识, 编写 `launch` 文件, 通过 `roslaunch` 同时启动 `talker` 与 `listener` 节点。

**launch 文件:**

```
demo01_transport.cpp  talker_listener.launch x
src > exp2 > launch > talker_listener.launch
1 <launch>
2   <node pkg="exp2" type="talker" name="mytalker" output="screen" />
3   <node pkg="exp2" type="listener" name="mylistener" output="screen" />
4 </launch>
```

**实验结果:**

```
ljw@ubuntu:~/rosexp_ws$ roslaunch exp2 talker_listener.launch
... logging to /home/ljw/.ros/log/3a4754e0-e3b9-11eb-827e-000c29bd2c4e/roslaunch-ubuntu-74
294.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ubuntu:39283/

SUMMARY
=====

PARAMETERS
* /rostdistro: melodic
* /rosversion: 1.14.11

NODES
/
  mylistener (exp2/listener)
  mytalker (exp2/talker)

ROS_MASTER_URI=http://localhost:11311

process[mytalker-1]: started with pid [74317]
process[mylistener-2]: started with pid [74318]
[ INFO] [1626167082.428795387]: 发布的消息是: hello,你好呀! 0
[ INFO] [1626167082.429161800]: 收到的数据是: hello,你好呀! 0
[ INFO] [1626167082.431514863]: 发布的消息是: hello,你好呀! 1
[ INFO] [1626167082.431694683]: 收到的数据是: hello,你好呀! 1
[ INFO] [1626167083.432467908]: 发布的消息是: hello,你好呀! 2
[ INFO] [1626167083.432690975]: 收到的数据是: hello,你好呀! 2
[ INFO] [1626167084.432197586]: 发布的消息是: hello,你好呀! 3
[ INFO] [1626167084.432872109]: 收到的数据是: hello,你好呀! 3
[ INFO] [1626167085.432079891]: 发布的消息是: hello,你好呀! 4
[ INFO] [1626167085.432554793]: 收到的数据是: hello,你好呀! 4
[ INFO] [1626167086.432235676]: 发布的消息是: hello,你好呀! 5
[ INFO] [1626167086.432662656]: 收到的数据是: hello,你好呀! 5
[ INFO] [1626167087.432756587]: 发布的消息是: hello,你好呀! 6
[ INFO] [1626167087.433555011]: 收到的数据是: hello,你好呀! 6
```



动手实现 3: 当话题通信时, 使用实验一学习的命令, 查看话题的信息, 比如 `roscall`、`rostopic`、`rosservice`。

**roscall:**

```
ljw@ubuntu:~/rosexp_ws$ roscall /rosout
```

```
-----
Node [/rosout]
Publications:
* /rosout_agg [rosgraph_msgs/Log]

Subscriptions:
* /rosout [unknown type]

Services:
* /rosout/get_loggers
* /rosout/set_logger_level
```

```
contacting node http://ubuntu:38829/ ...
Pid: 73773
```

**rostopic:**

```
ljw@ubuntu:~/rosexp_ws$ rostopic info /chatter
```

```
Type: std_msgs/String
```

```
Publishers:
* /mytalker (http://ubuntu:36501/)
```

```
Subscribers:
* /mylistener (http://ubuntu:33885/)
```

**rosservice:**

```
ljw@ubuntu:~/rosexp_ws$ rosservice info /mylistener/get_loggers
```

```
Node: /mylistener
URI: rosrpc://ubuntu:59835
Type: roscpp/GetLoggers
Args:
```

## 实验三：熟悉 ROS 服务通信架构，实现服务通信过程

### 1、实验目的

- (1) 了解 ROS 节点间的服务通信过程
- (2) 编写节点实现服务通信

### 2、实验设备

硬件环境：DELL 笔记本

系统环境：VmWare、Ubuntu18.04、ROS Melodic

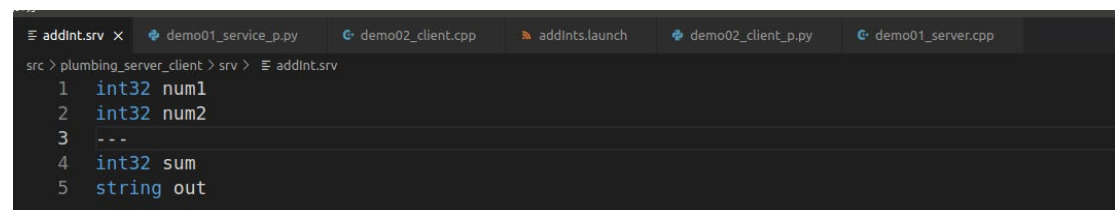
### 3、实验内容

- (1) 了解服务通信过程

```
ljlw@ubuntu:~$ rosservice info /turtle1/set_pen
Node: /turtlesim
URI: rosrpc://ubuntu:40521
Type: turtlesim/SetPen
Args: r g b width off
```

- (2) 编写节点实现服务通信

- **srv 消息文件**



```
addint.srv x demo01_service_p.py demo02_client.cpp addints.launch demo02_client_p.py demo01_server.cpp
src > plumbing_server_client > srv > addint.srv
1 int32 num1
2 int32 num2
3 ---
4 int32 sum
5 string out
```

- **C++版本**

**Service.cpp**

```

1  #include "ros/ros.h"
2  #include "plumbing_server_client/addInt.h"
3
4  /*
5   服务端实现：解析客户端提交的数据，并运算产生相应
6   1、包含头文件
7   2、初始化ROS节点
8   3、创建节点句柄
9   4、创建一个服务对象
10  5、处理请求并产生响应
11  6、spin()
12  */
13
14  bool doNums(plumbing_server_client::addIntRequest &request,
15             plumbing_server_client::addIntResponse &response) {
16
17      //1.处理请求
18      int num1 = request.num1;
19      int num2 = request.num2;
20      ROS_INFO("收到的请求数据:num1 = %d, num2 = %d", num1, num2);
21      //2.组织响应
22      int sum = num1 + num2;
23      response.sum = sum;
24      response.out = std::string(std::to_string(num1) + "+" + std::to_string(num2));
25      ROS_INFO("求和结果: sum = %d", sum);
26      ROS_INFO("求和的算式为: %s", response.out.c_str());
27      return true;
28  }

```

```

30  int main(int argc, char *argv[])
31  {
32      setlocale(LC_ALL, "");
33
34      ros::init(argc, argv, "heiShui"); //节点名称保证唯一
35      ros::NodeHandle nh;
36      ros::ServiceServer service = nh.advertiseService("addInts", doNums);
37      ROS_INFO("服务器端启动");
38      ros::spin();
39      return 0;
40  }
41

```

## Client.cpp

```

int main(int argc, char *argv[])
{
    setlocale(LC_ALL, "");
    ros::init(argc, argv, "daBao");

    ros::NodeHandle nh;
    ros::ServiceClient client = nh.serviceClient<plumbing_server_client::addInt>("addInts");

    plumbing_server_client::addInt ai;
    ai.request.num1 = atoi(argv[1]);
    ai.request.num2 = atoi(argv[2]);

    //调用判断服务器状态函数
    //函数1
    client.waitForExistence();
    bool flag = client.call(ai);

    if(flag) {
        ROS_INFO("响应成功!");
        ROS_INFO("响应结果 = %d", ai.response.sum);
        ROS_INFO("返回的算式为: %s", ai.response.out.c_str());
    } else {
        ROS_INFO("处理失败.....");
    }
    return 0;
}

```

## ● Python 版本

### Service.py

```
3 import rospy
4 from plumbing_server_client.srv import *
5 """
6     服务端：解析客户端请求，产生响应。
7
8     1、导包
9     2、初始化 ROS 节点；
10    3、创建服务端对象；
11    4、处理逻辑（回调函数）；
12    5、spin()
13 """
14 # 参数：封装了请求数据
15 # 返回值：响应数据
16 def doNum(request):
17     num1 = request.num1
18     num2 = request.num2
19
20     sum = num1 + num2
21     response = addIntResponse()
22     response.sum = sum
23     response.out = str(num1) + "+" + str(num2)
24     rospy.loginfo("服务器解析的数据 num1 = %d, num2 = %d, 响应的结果: sum = %d", num1, num2, sum)
25     rospy.loginfo("收到的算式为 : " + response.out)
26     return response
27
28 if __name__ == "__main__":
29     rospy.init_node("heiShui")
30     server = rospy.Service("addInts", addInt, doNum)
31     rospy.loginfo("服务器已经启动了")
32     rospy.spin()
```

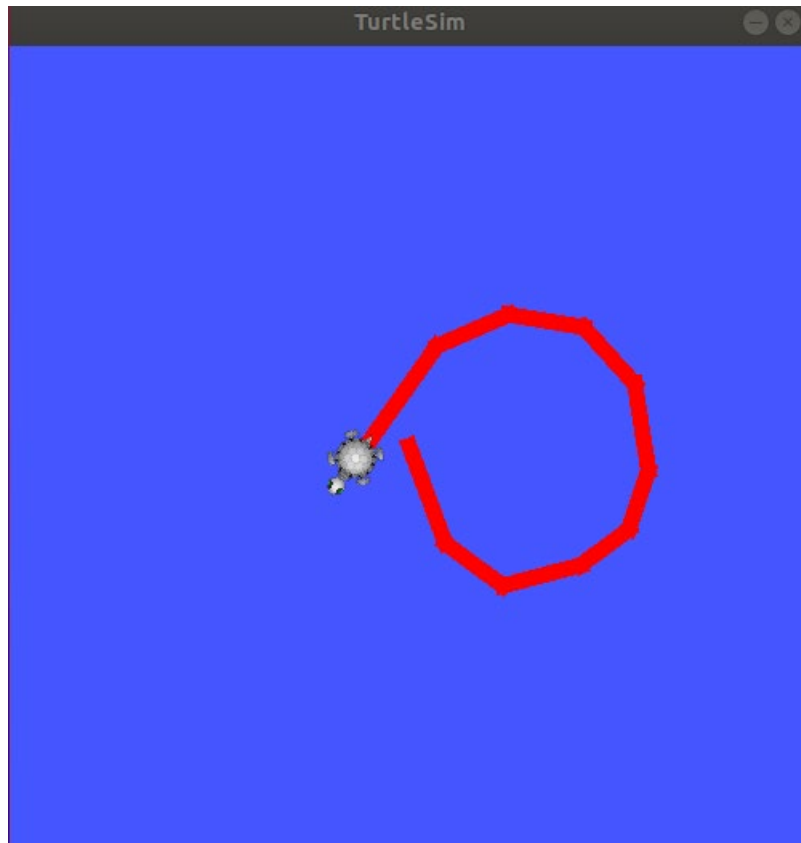
### Client.py

```
4 import rospy
5 from plumbing_server_client.srv import *
6 import sys
7
8 """
9     服务端：组织并提交请求，处理服务端响应
10
11     1、导包
12     2、初始化 ROS 节点；
13     3、创建客户端对象；
14     4、组织请求数据，并发送请求；
15     5、处理响应
16 """
17
18
19 if __name__ == "__main__":
20     if len(sys.argv) != 3:
21         rospy.loginfo("传入的参数个数不对")
22         sys.exit(1)
23     rospy.init_node("erHei")
24     client = rospy.ServiceProxy("addInts", addInt)
25     num1 = int(sys.argv[1])
26     num2 = int(sys.argv[2])
27     client.wait_for_service()
28     response = client.call(num1, num2)
29     rospy.loginfo("响应的数据:%d", response.sum)
30     rospy.loginfo("返回的算式为" + response.out)
```

## 4、动手实现

1、手动调用/turtle1/set\_pen 服务设置画笔为红色，宽度为 10，然后 控制海龟移动查看效果。

```
ljw@ubuntu:~$ rosservice call /turtle1/set_pen "{r: 255, g: 0, b: 0, width: 10, 'off': 0}"
```



2、 尝试改写服务通信的服务功能，比如拼接客户端传来的参数并返回。

在实验内容中已经写进代码里了，我在 `srv` 文件中加了一个 `string` 类型的变量，用来拼接运算的字符串，并将拼接结果返回给 `client`，由 `client` 输出结果。

3、 思考是否能够将 `server` 与 `listener` 使用 `launch` 启动？若可以请实现。  
可以。

```
1 <launch>
2   <node pkg="plumbing_server_client" type="demo01_server" name="server" output="screen" />
3   <node pkg="plumbing_server_client" type="demo02_client" name="client" args="1 2" output="screen" />
4 </launch>
```

结果：

```
ljw@ubuntu:~/demo03_ws$ roslaunch plumbing_server_client addInts.launch
... logging to /home/ljw/.ros/log/d10f10be-e3bb-11eb-827e-000c29bd2c4e/roslaunch-ubuntu-83
906.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ubuntu:45641/

SUMMARY
=====

PARAMETERS
* /roscpp: melodic
* /rosversion: 1.14.11

NODES
/
  client (plumbing_server_client/demo02_client)
  server (plumbing_server_client/demo01_server)

ROS_MASTER_URI=http://localhost:11311

process[server-1]: started with pid [83924]
process[client-2]: started with pid [83925]
[ INFO] [1626169070.596038278]: 参数个数:5
[ INFO] [1626169070.596157292]: 参数1:/home/ljw/demo03_ws/devel/lib/plumbing_server_client
/demo02_client
[ INFO] [1626169070.596185094]: 参数2:1
[ INFO] [1626169070.596191267]: 参数3:2
[ INFO] [1626169070.596195597]: 参数4: __name:=client
[ INFO] [1626169070.596199982]: 参数5: __log:=/home/ljw/.ros/log/d10f10be-e3bb-11eb-827e-00
0c29bd2c4e/client-2.log
[ INFO] [1626169070.599006896]: 服务器端启动
[ INFO] [1626169070.605454371]: 收到的请求数据: num1 = 1, nums2 = 2
[ INFO] [1626169070.605540169]: 求和结果: sum = 3
[ INFO] [1626169070.605582862]: 求和的算式为: 1+2
[ INFO] [1626169070.605779844]: 响应成功!
[ INFO] [1626169070.606596416]: 响应结果 = 3
[ INFO] [1626169070.606610319]: 返回的算式为: 1+2
[client-2] process has finished cleanly
log file: /home/ljw/.ros/log/d10f10be-e3bb-11eb-827e-000c29bd2c4e/client-2*.log
```

# 实验 4 了解 ROS 中常用组件：launch 启动文件、Qt 工具箱和 rviz 三维可视化环境

## 1、实验目的

了解 ROS 中常用组件，具体内容如下：

- (1) 学会使用 launch 启动文件同时启动多个节点。
- (2) 学会使用 Qt 工具箱中的常用工具。
- (3) 学会使用 rviz 相关插件。

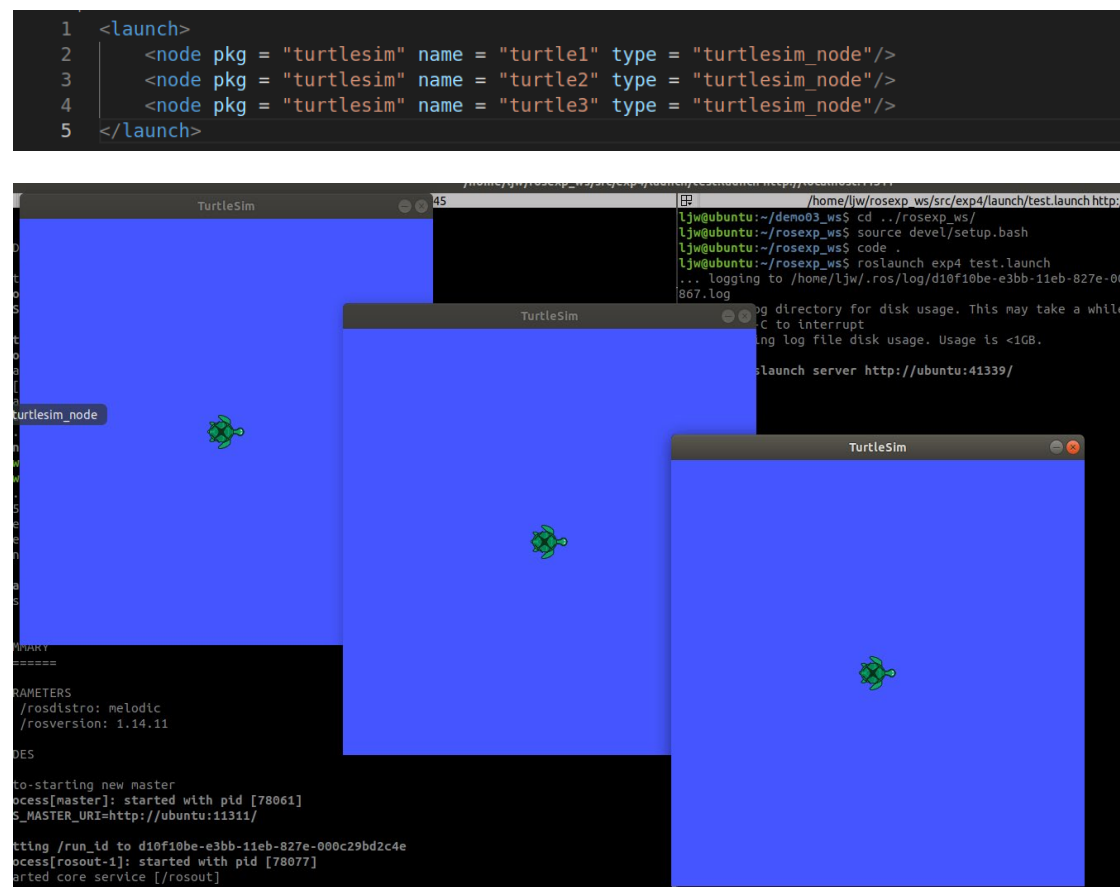
## 2、实验设备

硬件环境：DELL 笔记本

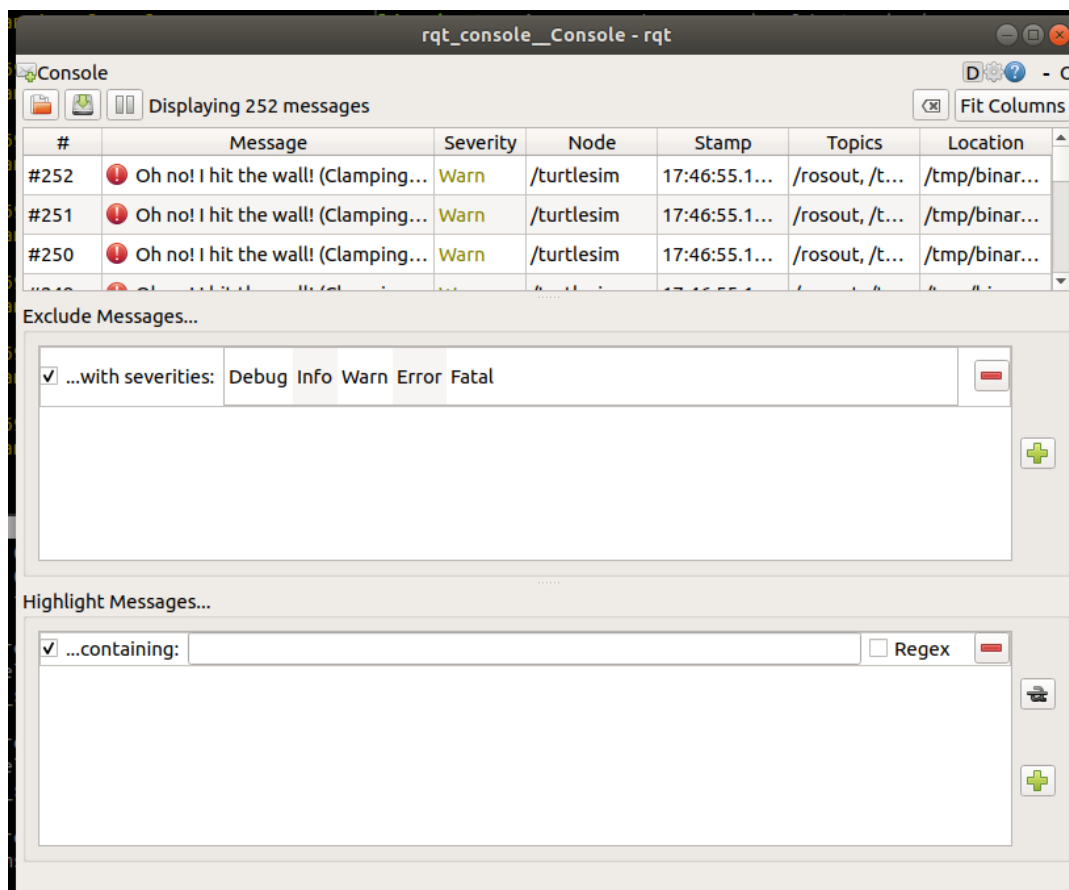
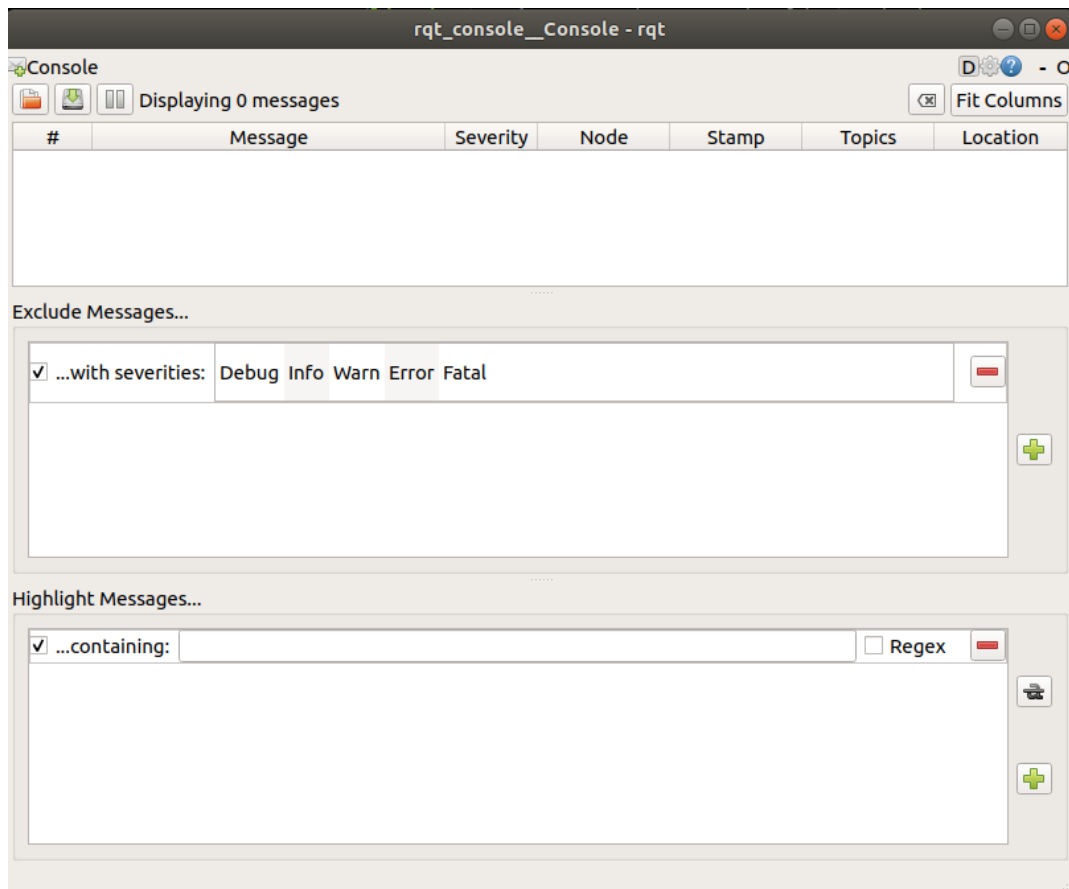
系统环境：VmWare、Ubuntu18.04、ROS Melodic

## 3、实验内容

- (1) launch 启动文件

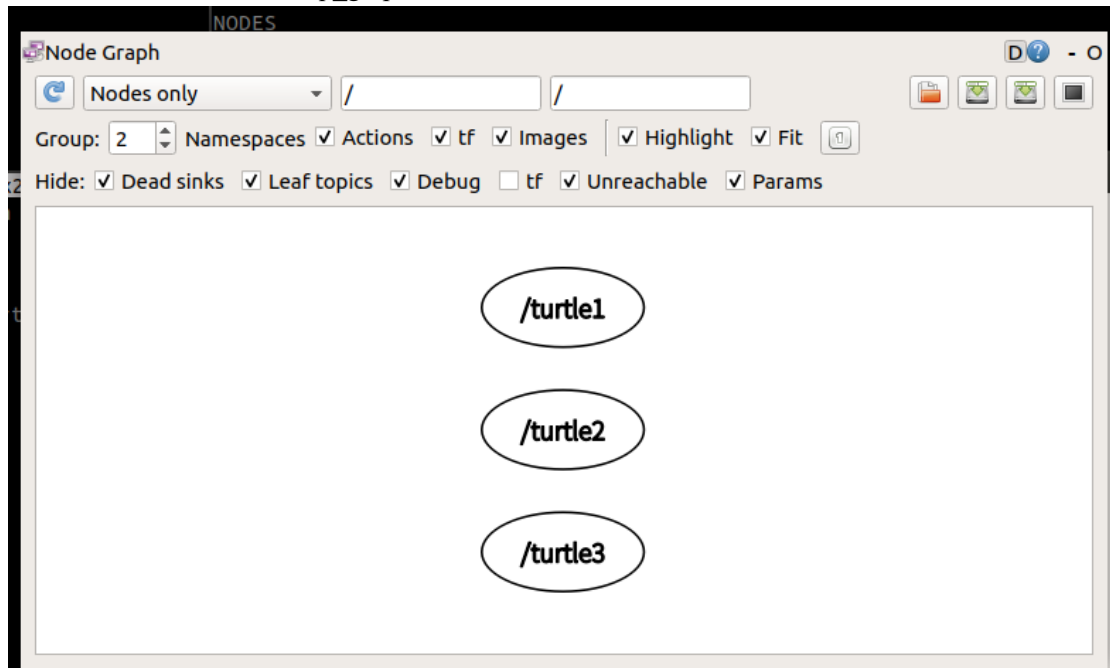


## (2) Qt 工具箱

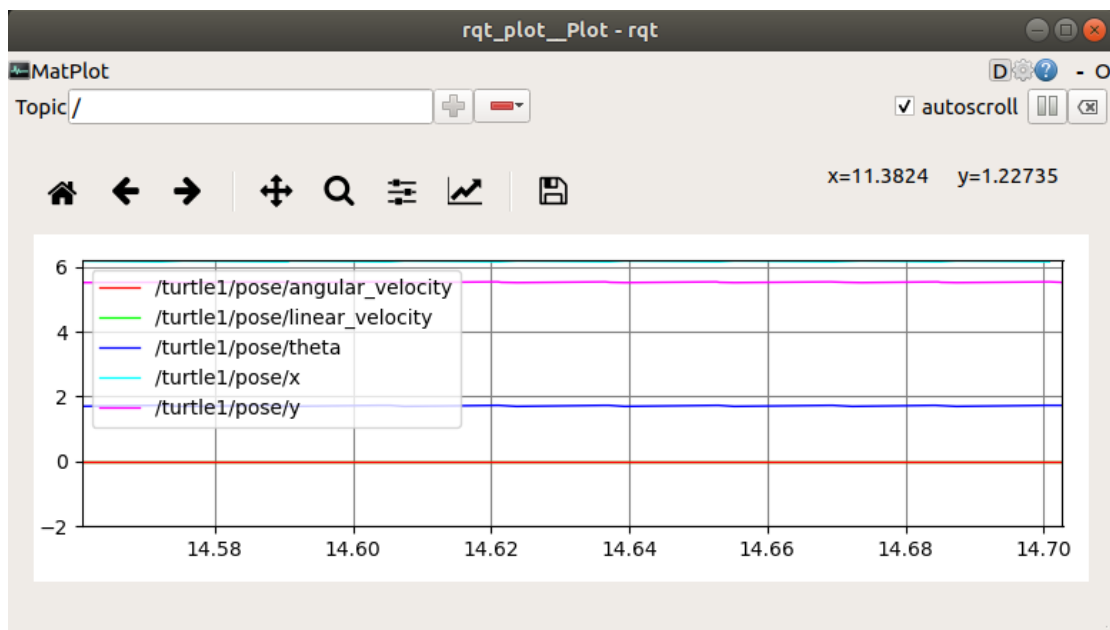




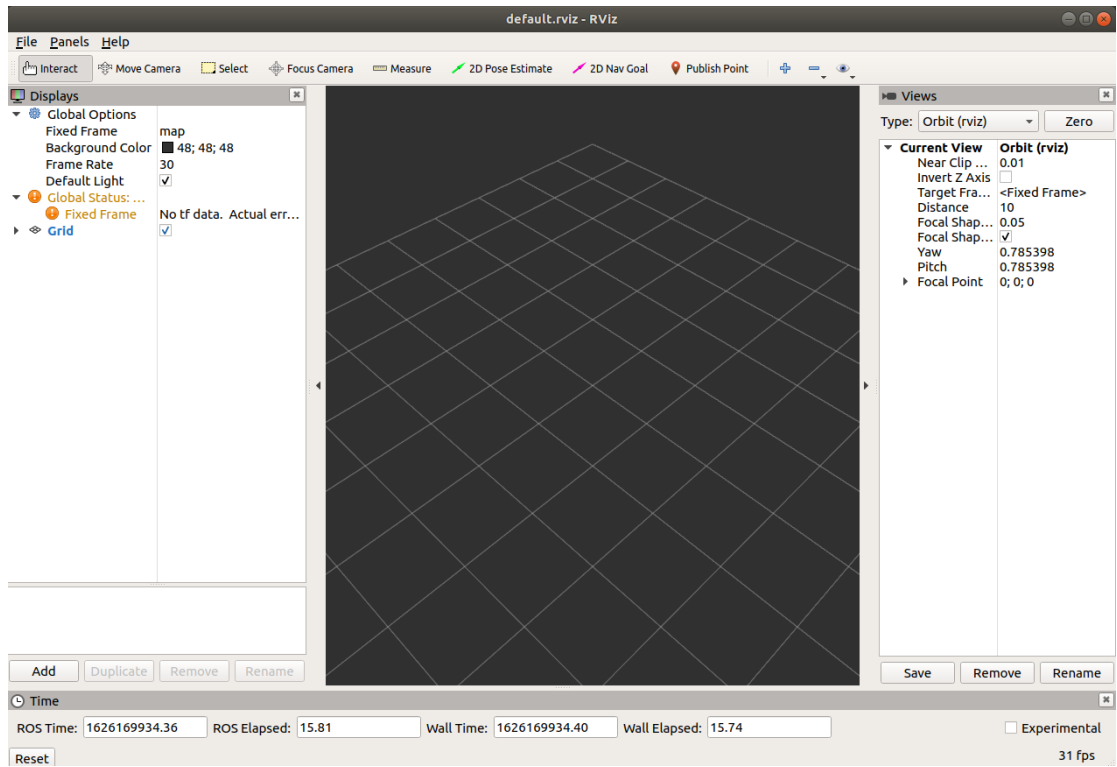
(3) 节点可视化工具: rqt\_graph



(4) 数据绘图工具: rqt\_plot

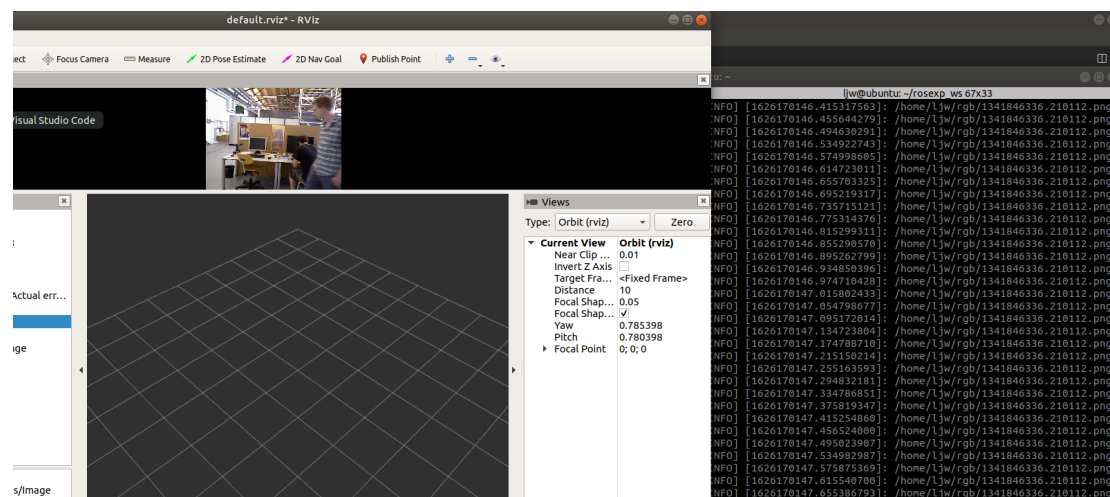


(5) rviz 三维可视化环境



## 4、动手实现

```
10 int main(int argc, char** argv)
11 {
12     ros::init(argc, argv, "image_publisher");
13     ros::NodeHandle nh;
14     image_transport::ImageTransport it(nh);
15     image_transport::Publisher pub = it.advertise("camera/image", 10);
16
17     std::ifstream ins;
18     string base_dir = "/home/ljw/";
19     string file_dir = base_dir + "rgb.txt";
20     ins.open(file_dir.c_str());
21
22     string name;
23     ros::Rate rate(25);
24     while(ros::ok()) {
25         ins >> name;
26         string photo_dir = base_dir + "rgb/" + name;
27         ROS_INFO("%s", photo_dir.c_str());
28         cv::Mat image = cv::imread(photo_dir, CV_LOAD_IMAGE_COLOR);
29         sensor_msgs::ImagePtr msg = cv_bridge::CvImage(std_msgs::Header(), "bgr8", image).toImageMsg();
30         pub.publish(msg);
31         rate.sleep();
32         ros::spinOnce();
33     }
34
35     ins.close();
36     return 0;
37 }
```



# 实验 5 ROS 简单应用：激光 SLAM

## 1、实验目的

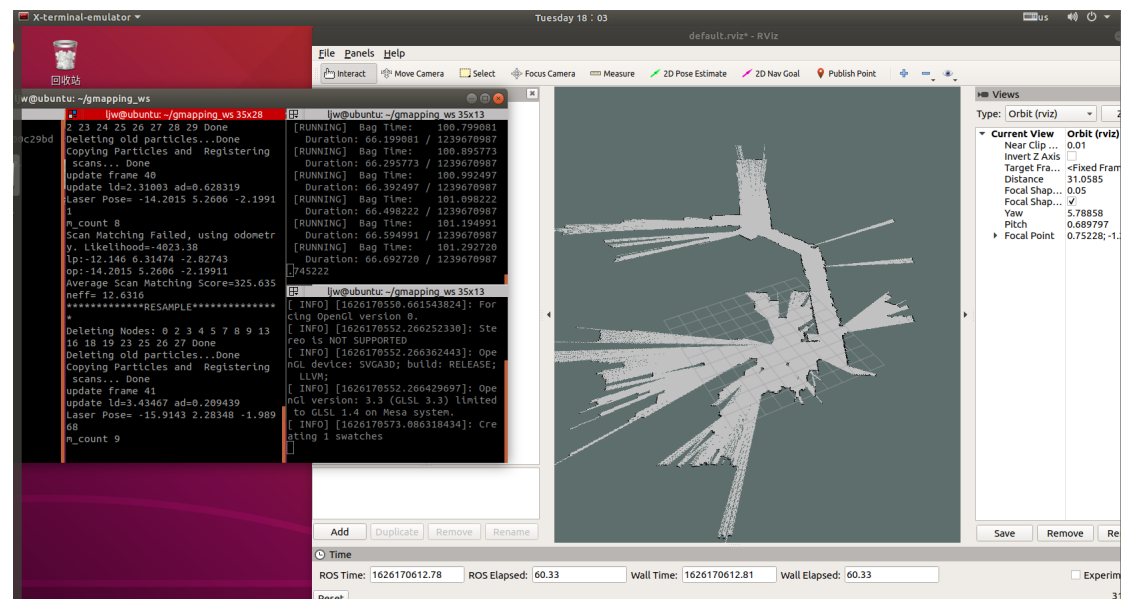
- (1) 了解机器人和 SLAM 之间关系
- (2) 了解激光 SLAM
- (3) 了解 gmapping

## 2、实验设备

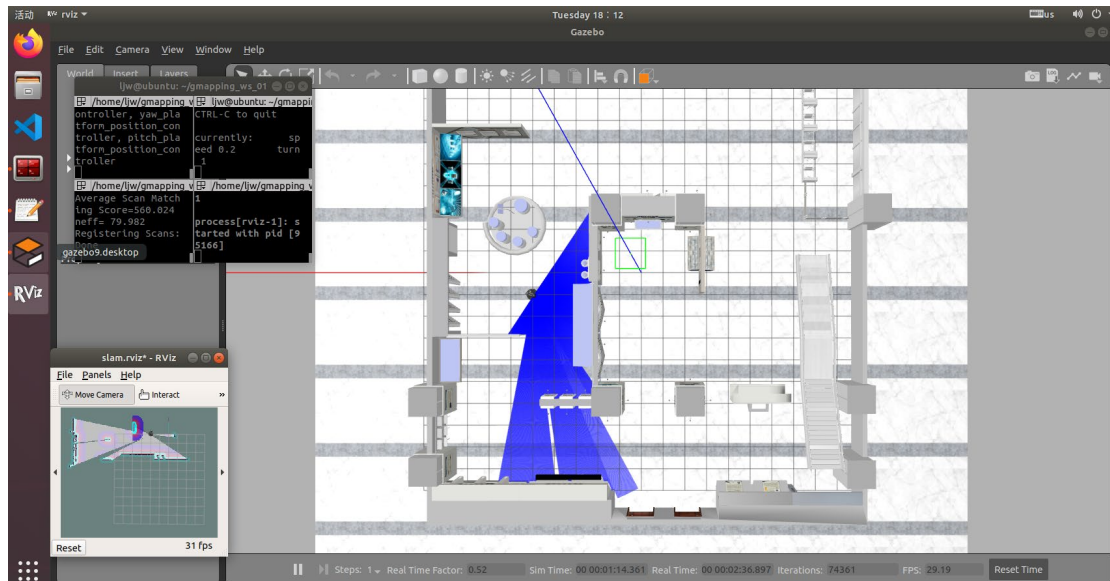
硬件环境：DELL 笔记本

系统环境：VmWare、Ubuntu18.04、ROS Melodic

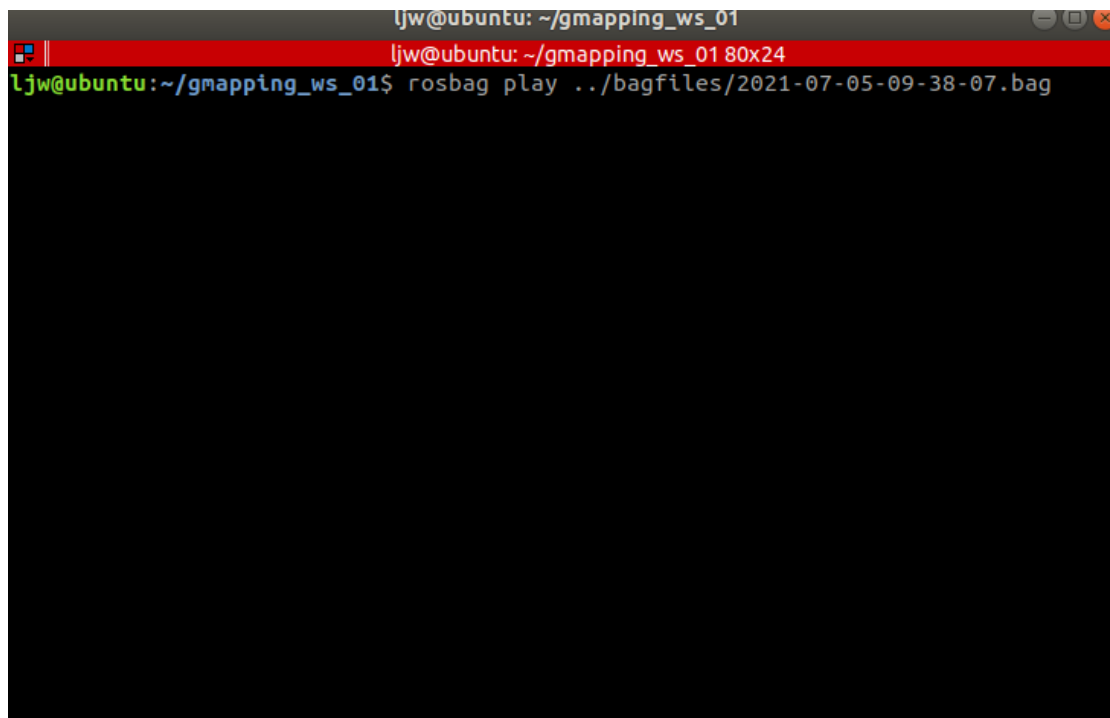
## 3、实验内容

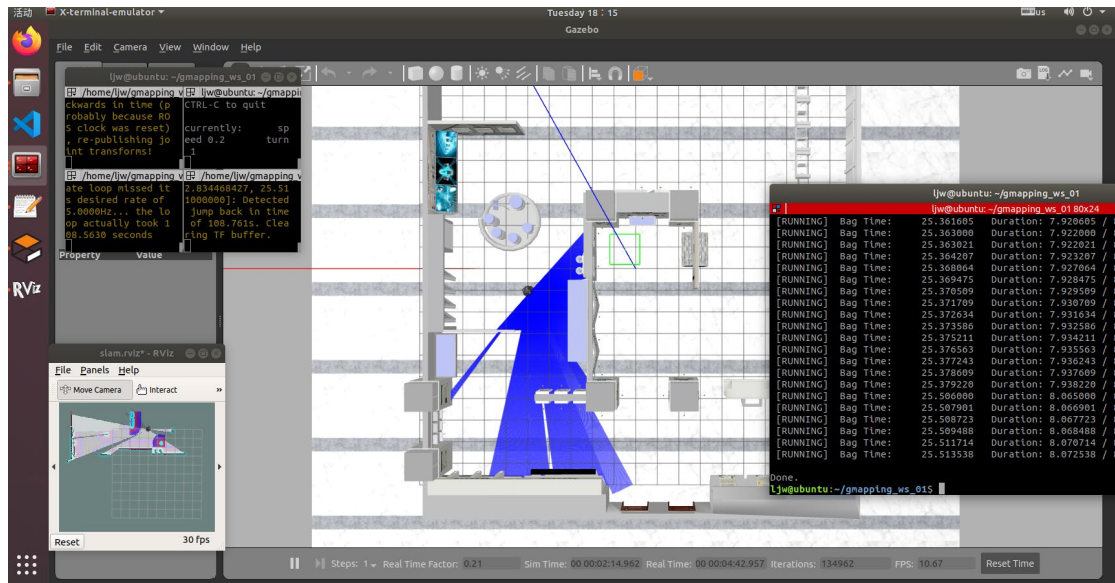


## 4、动手实现



使用 bag 文件重现运动轨迹





# 实验 6 ROS 简单应用：视觉 SLAM

## 1、实验目的

- (1) 了解视觉 SLAM
- (2) 了解 ORB-SLAM

## 2、实验设备

硬件环境：DELL 笔记本

系统环境：VmWare、Ubuntu18.04、ROS Melodic

## 3、动手实现

尝试了解 ORB-SLAM2 相关工作原理，并尝试编译并运行 ORB-SLAM2 源码。

- ORB-SLAM2 工作原理

ORB-SLAM2 算法使用 ORB 算子进行特征点的提取与描述，其算法原理来自于文章《ORB an efficient alternative to SIFT or SURF》。

ORB 算法相比于 SIFT 具有相似的匹配能力，运算量更小，受噪声的影响更小，能够用于实时运算。作者的主要目的是将该算法更为广泛地应用到各种图像处理场景中，譬如用没有 GPU 的低功耗设备实现全景拼接和图像跟踪，或者减少在 PC 上执行基于特征的物体检测运算时间。在这些应用中该算法表现和 SIFT 一样好（优于 SURF），同时运行速度比 SIFT 快两个数量级。ORB 算法基于 FAST 角点检测算法和 BRIEF 描述子，因此称为 ORB (Oriented FAST and Rotated BRIEF)。这两种方法都具有高性能低运算量的特点。

### ORB 算法流程：

- (1) 特征点的检测

ORB 采用 FAST (features from accelerated segment test) 算法来检测特征点。FAST 核心思想就是找出那些卓尔不群的点，即拿一个点跟它周围的点比较，如果它和其中大部分的点都不一样就可以认为它是一个特征点。

FAST 具体计算过程：

1. 从图片中选取一个像素点 P，下面我们将判断它是否是一个特征点。我们首先把它的密度（即灰度值）设为  $I_p$ 。
2. 设定一个合适的阈值  $t$ ：当 2 个点的灰度值之差的绝对值大于  $t$  时，我们认为这 2 个点不相同。
3. 考虑该像素点周围的 16 个像素。
4. 现在如果这 16 个点中有连续的  $n$  个点都和点不同，那么它就是一个角点。这里  $n$  设定为 12。
5. 我们现在提出一个高效的测试，来快速排除一大部分非特征点的点。该测试仅仅检查在位置 1、9、5 和 13 四个位置的像素（首先检查 1 和 9，看它们是否和点相同。如果是，再检查 5 和 13）。如果是一个角点，那么上述四个像素点中至少有 3 个应该和点相同。如果

都不满足，那么不可能是一个角点。

### (2) 特征点的描述

得到特征点后我们需要以某种方式  $F$  描述这些特征点的属性。这些属性的输出我们称之为该特征点的描述子 (Feature Descriptor)。ORB 采用 BRIEF 算法来计算一个特征点的描述子。BRIEF 算法的核心思想是在关键点  $P$  的周围以一定模式选取  $N$  个点对，把这  $N$  个点对的比较结果组合起来作为描述子。

### (3) 特征点的匹配

ORB 算法最大的特点就是计算速度快。这首先得益于使用 FAST 检测特征点，FAST 的检测速度正如它的名字一样是出了名的快。再次是使用 BRIEF 算法计算描述子，该描述子特有的 2 进制串的表现形式不仅节约了存储空间，而且大大缩短了匹配的时间。

## ● 编译运行 ORB-SLAM2 源码

### (a) 安装 Pangolin

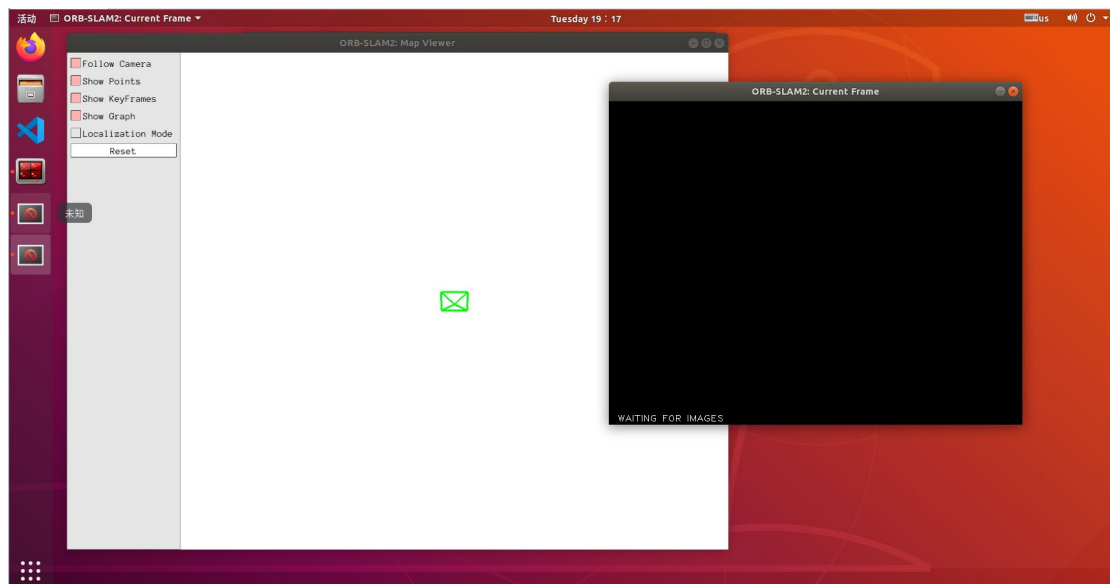
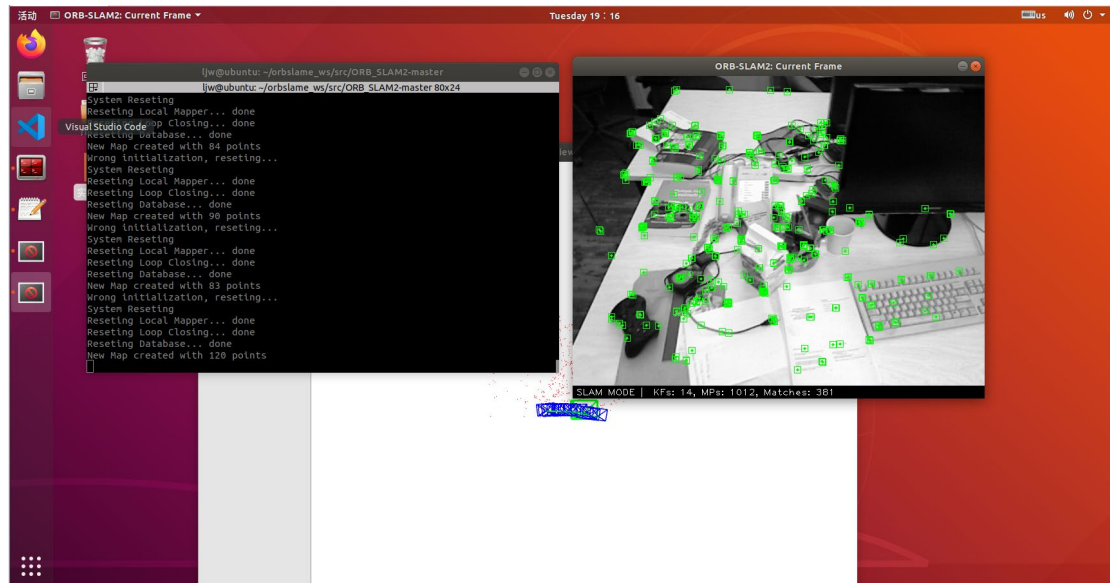
```
$ sudo apt-get install libglew-dev
$ git clone https://github.com/stevenlovegrove/Pangolin.git
$ cd Pangolin
$ mkdir build
$ cd build
$ cmake ..
$ make -j
```

### (b) 安装 ORB\_SLAM2

```
$ cd ~/catkin_ws/src/
$ git clone https://github.com/raulmur/ORB_SLAM2.git ORB_SLAM2
$ cd ORB_SLAM2
$ chmod +x build.sh
$ ./build.sh
$ chmod +x build_ros.sh
$ export
ROS_PACKAGE_PATH=${ROS_PACKAGE_PATH}:/catkin_ws/src/ORB_SLAM2/Examples/ROS
$ ./build_ros.sh
```

实验结果：





由于虚拟机无法连接笔记本的内置摄像头，所以无法实时检测场景。

遇到的问题：

(1) ORB\_SLAM2/src/System.cc: error: 'usleep' was not declared in this scope usleep(5000);

解决办法：在所有缺失 usleep 的文件中添加一个头文件 `#include<unistd.h>`

(2) CMakeFiles/RGBD.dir/build.make:197: recipe for target './RGBD' failed

解决办法：修改 home/catkin\_ws/src/ORB\_SLAM2/Examples/ROS/ORB\_SLAM2/文件夹下的 CMakeLists.txt 文件。在 set (LIBS xxxxx 的后面加上 `-lboost_system`

(3) OPENCV 版本不匹配

解决办法：修改 CmakeLists.txt，将 OPENCV 版本修改为 3.2.0（系统自带 3.2.0 版本）

```
find_package(OpenCV 3.2.0 QUIET)
if(NOT OpenCV_FOUND)
    find_package(OpenCV 2.4.3 QUIET)
    if(NOT OpenCV_FOUND)
        message(FATAL_ERROR "OpenCV > 2.4.3 not found.")
    endif()
endif()
```