实验二：ROS 基础与通信机制

1. 实验目的

- 理解 ROS 节点（Node）概念
- 掌握话题（Topic）发布-订阅机制
- 掌握服务（Service）请求-响应机制
- 理解 ROS Master 的作用

2. ROS 核心概念

2.1 基本组成

- **Node（节点）**：ROS 系统的基本运行单元，每个节点执行特定任务
- **Topic（话题）**：节点间通信的命名通道
- **Message Type（消息类型）**：定义话题上传输的数据格式
- **Master（主节点）**：提供节点间通信的协调服务（需要 `roscore`）

2.2 通信类型对比

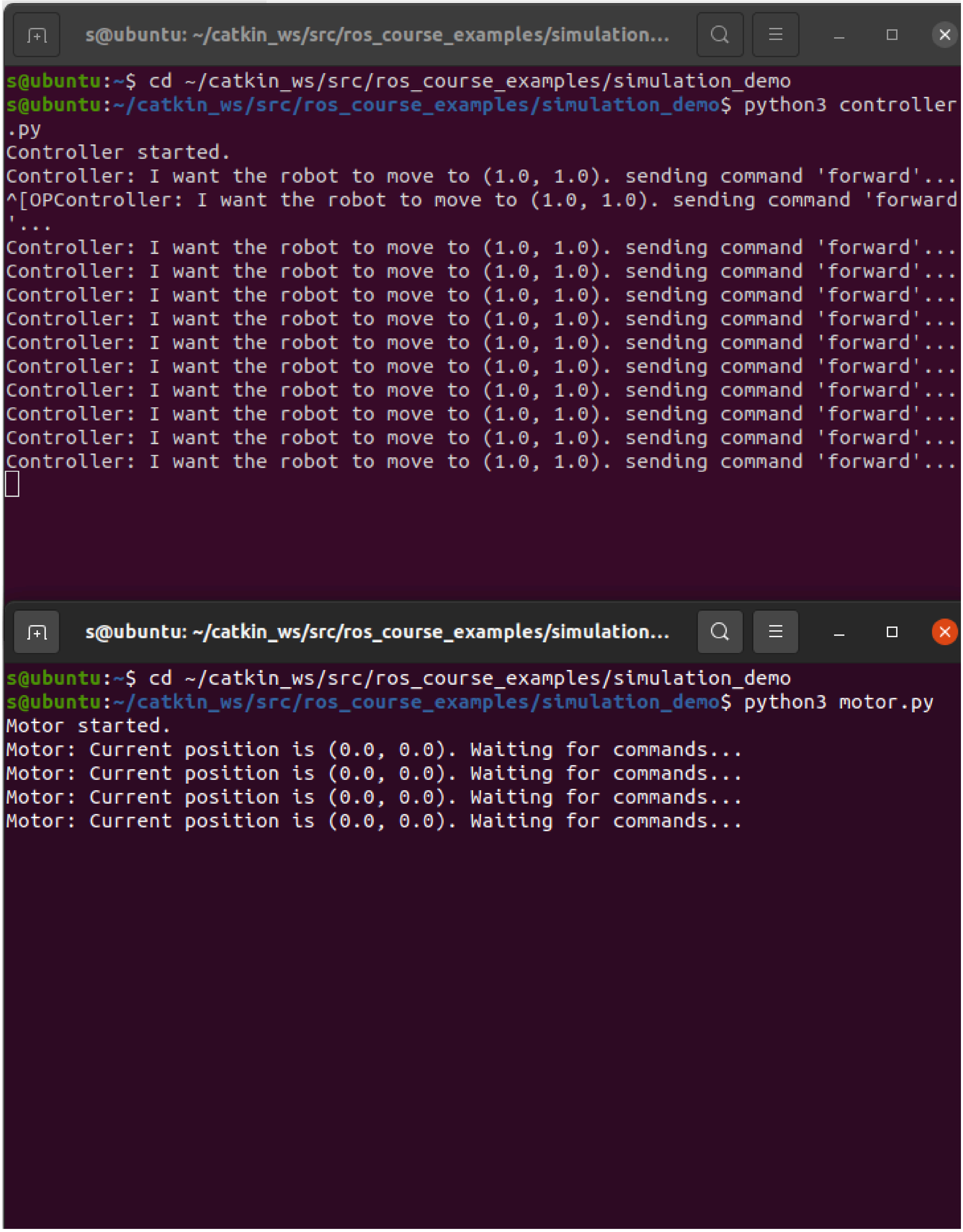
通信方式	特点	使用场景	比喻
Topic	异步，持续数据流	传感器数据、状态信息	广播电台
Service	同步，请求-响应	计算、查询、触发动作	打电话

3. 实验步骤

实验 2.1：体验无通信的孤立程序

```
cd ~/catkin_ws/src/ros_course_examples/simulation_demo
python3 controller.py

# 终端 2: 运行 Motor
cd ~/catkin_ws/src/ros_course_examples/simulation_demo
python3 motor.py
```

执行效果如图：



```
s@ubuntu:~$ cd ~/catkin_ws/src/ros_course_examples/simulation_demo
s@ubuntu:~/catkin_ws/src/ros_course_examples/simulation_demo$ python3 controller
.py
Controller started.
Controller: I want the robot to move to (1.0, 1.0). sending command 'forward'...
^[OPController: I want the robot to move to (1.0, 1.0). sending command 'forward
'...
Controller: I want the robot to move to (1.0, 1.0). sending command 'forward'...
Controller: I want the robot to move to (1.0, 1.0). sending command 'forward'...
Controller: I want the robot to move to (1.0, 1.0). sending command 'forward'...
Controller: I want the robot to move to (1.0, 1.0). sending command 'forward'...
Controller: I want the robot to move to (1.0, 1.0). sending command 'forward'...
Controller: I want the robot to move to (1.0, 1.0). sending command 'forward'...
Controller: I want the robot to move to (1.0, 1.0). sending command 'forward'...
Controller: I want the robot to move to (1.0, 1.0). sending command 'forward'...
Controller: I want the robot to move to (1.0, 1.0). sending command 'forward'...
Controller: I want the robot to move to (1.0, 1.0). sending command 'forward'...
```

```
s@ubuntu:~$ cd ~/catkin_ws/src/ros_course_examples/simulation_demo
s@ubuntu:~/catkin_ws/src/ros_course_examples/simulation_demo$ python3 motor.py
Motor started.
Motor: Current position is (0.0, 0.0). Waiting for commands...
Motor: Current position is (0.0, 0.0). Waiting for commands...
Motor: Current position is (0.0, 0.0). Waiting for commands...
Motor: Current position is (0.0, 0.0). Waiting for commands...
```

观察： 两个程序各自运行，无法相互通信。

关闭这两个程序吧。

实验 2.2：使用 ROS 实现节点通信

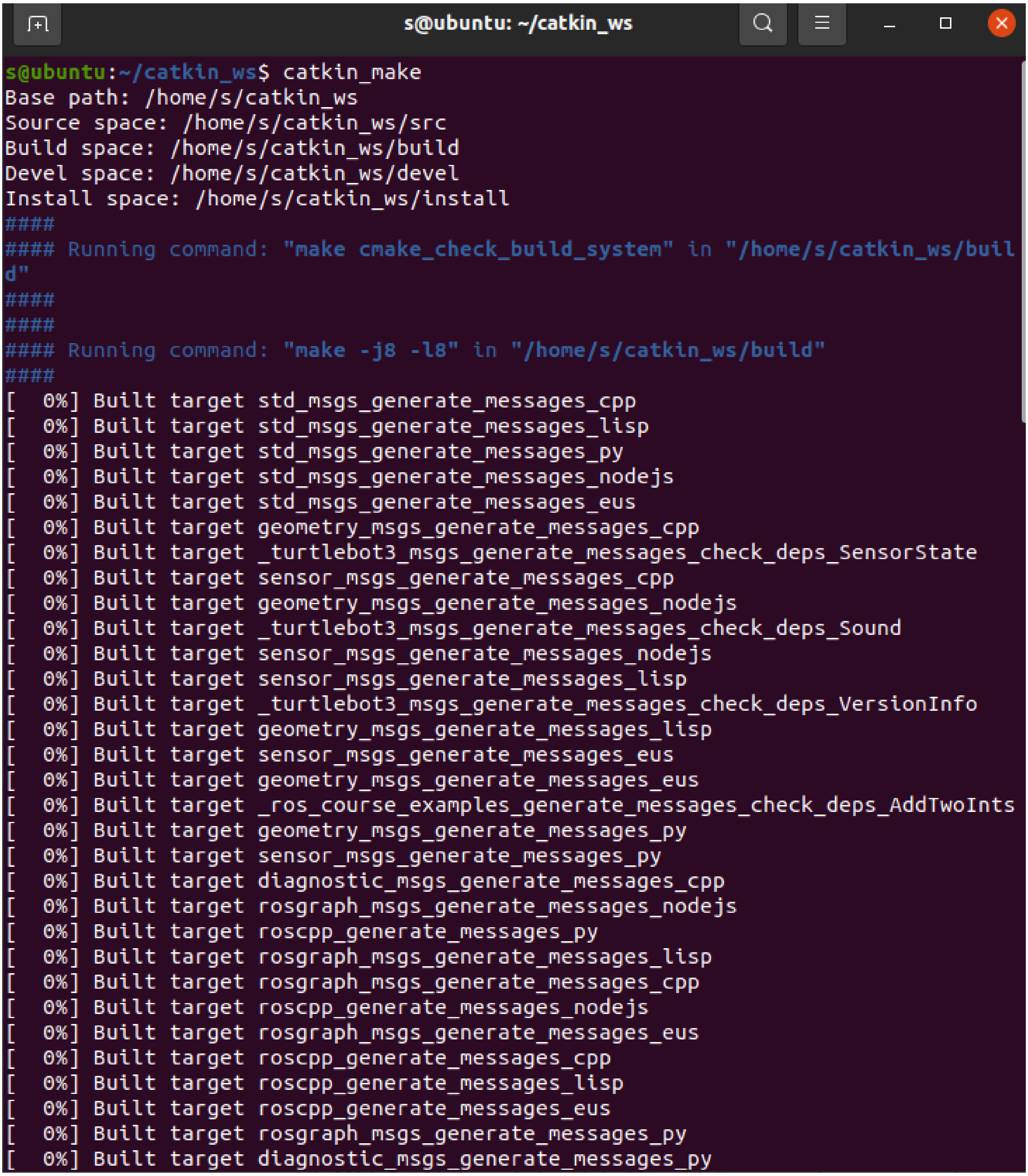
我们可以将代码封装成结点，然后基于ROS框架实现结点通信。

首先进行编译

在终端 1：编译ros工作空间

```
cd ~/catkin_ws
catkin_make
```

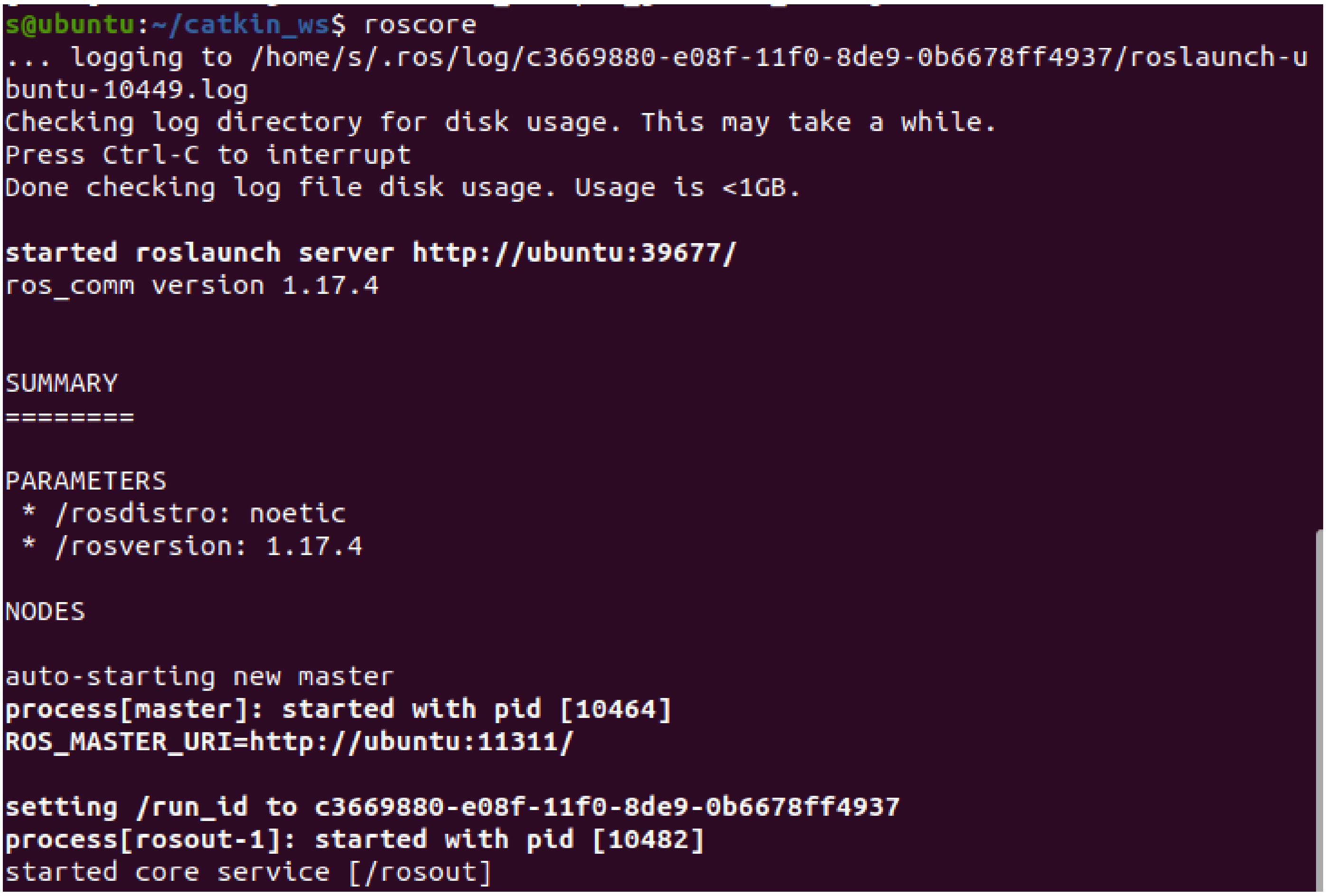
大概的示意图：



```
s@ubuntu:~/catkin_ws$ catkin_make
Base path: /home/s/catkin_ws
Source space: /home/s/catkin_ws/src
Build space: /home/s/catkin_ws/build
Devel space: /home/s/catkin_ws/devel
Install space: /home/s/catkin_ws/install
####
#### Running command: "make cmake_check_build_system" in "/home/s/catkin_ws/buil
d"
####
####
#### Running command: "make -j8 -l8" in "/home/s/catkin_ws/build"
####
[  0%] Built target std_msgs_generate_messages_cpp
[  0%] Built target std_msgs_generate_messages_lisp
[  0%] Built target std_msgs_generate_messages_py
[  0%] Built target std_msgs_generate_messages_nodejs
[  0%] Built target std_msgs_generate_messages_eus
[  0%] Built target geometry_msgs_generate_messages_cpp
[  0%] Built target _turtlebot3_msgs_generate_messages_check_deps_SensorState
[  0%] Built target sensor_msgs_generate_messages_cpp
[  0%] Built target geometry_msgs_generate_messages_nodejs
[  0%] Built target _turtlebot3_msgs_generate_messages_check_deps_Sound
[  0%] Built target sensor_msgs_generate_messages_nodejs
[  0%] Built target sensor_msgs_generate_messages_lisp
[  0%] Built target _turtlebot3_msgs_generate_messages_check_deps_VersionInfo
[  0%] Built target geometry_msgs_generate_messages_lisp
[  0%] Built target sensor_msgs_generate_messages_eus
[  0%] Built target geometry_msgs_generate_messages_eus
[  0%] Built target _ros_course_examples_generate_messages_check_deps_AddTwoInts
[  0%] Built target geometry_msgs_generate_messages_py
[  0%] Built target sensor_msgs_generate_messages_py
[  0%] Built target diagnostic_msgs_generate_messages_cpp
[  0%] Built target rosgraph_msgs_generate_messages_nodejs
[  0%] Built target roscpp_generate_messages_py
[  0%] Built target rosgraph_msgs_generate_messages_lisp
[  0%] Built target rosgraph_msgs_generate_messages_cpp
[  0%] Built target roscpp_generate_messages_nodejs
[  0%] Built target rosgraph_msgs_generate_messages_eus
[  0%] Built target roscpp_generate_messages_cpp
[  0%] Built target roscpp_generate_messages_lisp
[  0%] Built target roscpp_generate_messages_eus
[  0%] Built target rosgraph_msgs_generate_messages_py
[  0%] Built target diagnostic_msgs_generate_messages_py
```

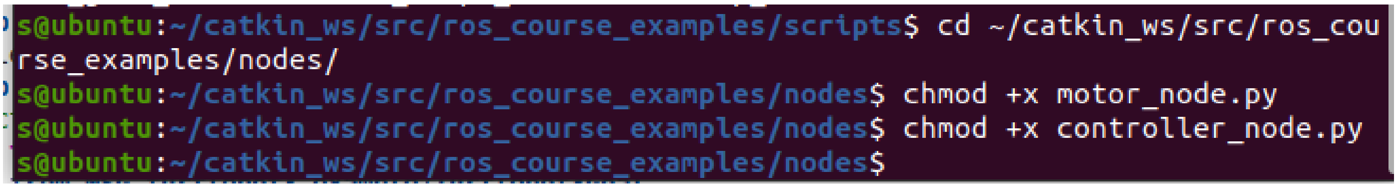
终端 1：启动 ROS Master

```
roscore
```



```
s@ubuntu:~/catkin_ws$ roscore
... logging to /home/s/.ros/log/c3669880-e08f-11f0-8de9-0b6678ff4937/roslaunch-u
buntu-10449.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:39677/
ros_comm version 1.17.4


SUMMARY
========

PARAMETERS
 * /rosdistro: noetic
 * /rosversion: 1.17.4

NODES

auto-starting new master
process[master]: started with pid [10464]
ROS_MASTER_URI=http://ubuntu:11311/

setting /run_id to c3669880-e08f-11f0-8de9-0b6678ff4937
process[rosout-1]: started with pid [10482]
started core service [/rosout]
```

```
cd ~/catkin_ws/src/ros_course_examples/nodes/
chmod +x motor_node.py
chmod +x controller_node.py
```



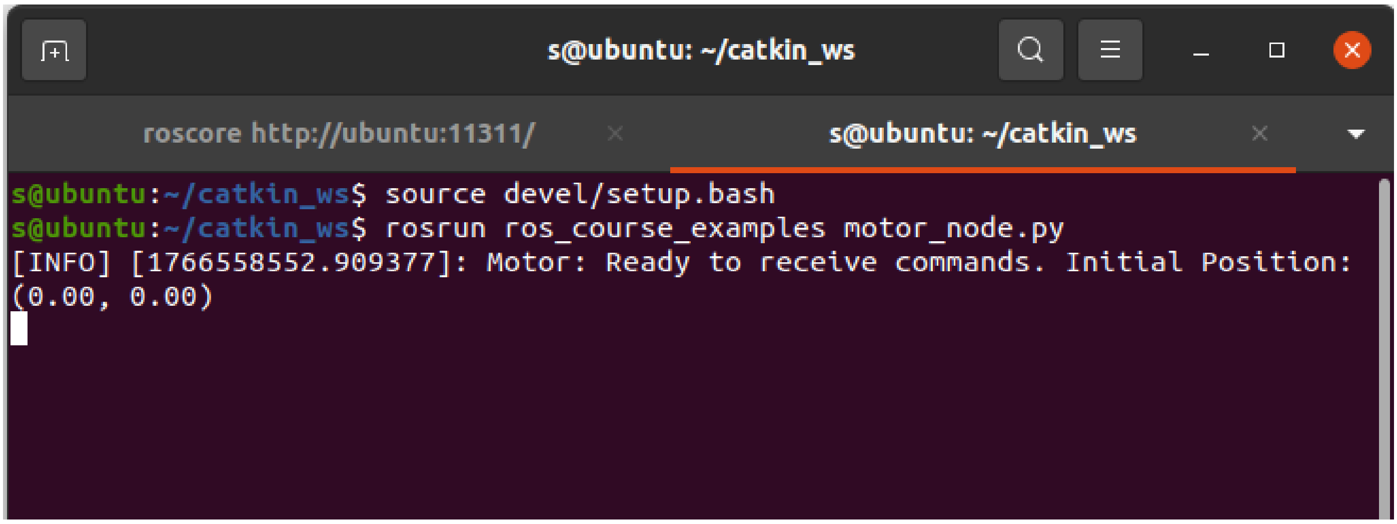
```
s@ubuntu:~/catkin_ws/src/ros_course_examples/scripts$ cd ~/catkin_ws/src/ros_cou
rse_examples/nodes/
s@ubuntu:~/catkin_ws/src/ros_course_examples/nodes$ chmod +x motor_node.py
s@ubuntu:~/catkin_ws/src/ros_course_examples/nodes$ chmod +x controller_node.py
s@ubuntu:~/catkin_ws/src/ros_course_examples/nodes$
```

启动通信演示

终端 2：启动电机结点

```
cd ~/catkin_ws
source devel/setup.bash
rosrun ros_course_examples motor_node.py
```

你可以发现这个结点（程序）正在等待信息，它的位置并不发生改变。

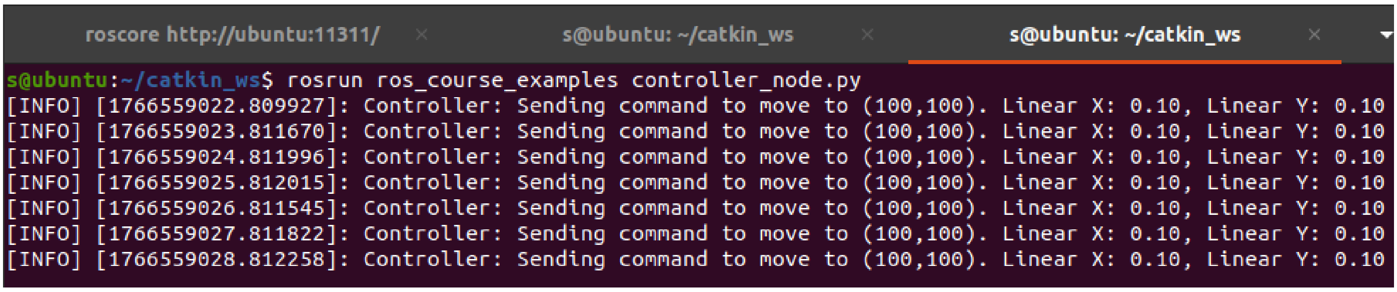


```
s@ubuntu:~/catkin_ws$ source devel/setup.bash
s@ubuntu:~/catkin_ws$ rosrun ros_course_examples motor_node.py
[INFO] [1766558552.909377]: Motor: Ready to receive commands. Initial Position:
(0.00, 0.00)
```

终端 3：启动控制器结点 接下来，让我们控制它

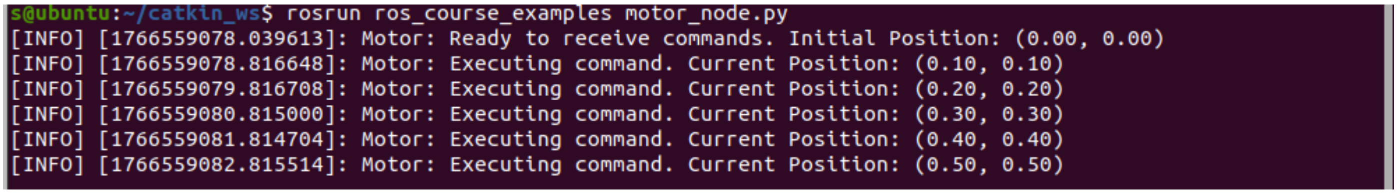
```
cd ~/catkin_ws
source devel/setup.bash
rosrun ros_course_examples controller_node.py
```

观察终端，我们可以发现它正在不断的发送指令：



```
s@ubuntu:~/catkin_ws$ rosrun ros_course_examples controller_node.py
[INFO] [1766559022.809927]: Controller: Sending command to move to (100,100). Linear X: 0.10, Linear Y: 0.10
[INFO] [1766559023.811670]: Controller: Sending command to move to (100,100). Linear X: 0.10, Linear Y: 0.10
[INFO] [1766559024.811996]: Controller: Sending command to move to (100,100). Linear X: 0.10, Linear Y: 0.10
[INFO] [1766559025.812015]: Controller: Sending command to move to (100,100). Linear X: 0.10, Linear Y: 0.10
[INFO] [1766559026.811545]: Controller: Sending command to move to (100,100). Linear X: 0.10, Linear Y: 0.10
[INFO] [1766559027.811822]: Controller: Sending command to move to (100,100). Linear X: 0.10, Linear Y: 0.10
[INFO] [1766559028.812258]: Controller: Sending command to move to (100,100). Linear X: 0.10, Linear Y: 0.10
```

返回查看终端2（电机结点）



```
s@ubuntu:~/catkin_ws$ rosrun ros_course_examples motor_node.py
[INFO] [1766559078.039613]: Motor: Ready to receive commands. Initial Position: (0.00, 0.00)
[INFO] [1766559078.816648]: Motor: Executing command. Current Position: (0.10, 0.10)
[INFO] [1766559079.816708]: Motor: Executing command. Current Position: (0.20, 0.20)
[INFO] [1766559080.815000]: Motor: Executing command. Current Position: (0.30, 0.30)
[INFO] [1766559081.814704]: Motor: Executing command. Current Position: (0.40, 0.40)
[INFO] [1766559082.815514]: Motor: Executing command. Current Position: (0.50, 0.50)
```

观察到： Controller 发送速度指令，Motor 接收并更新位置。

但是，如果每启动一个程序都输入一行指令不会太复杂了吗？

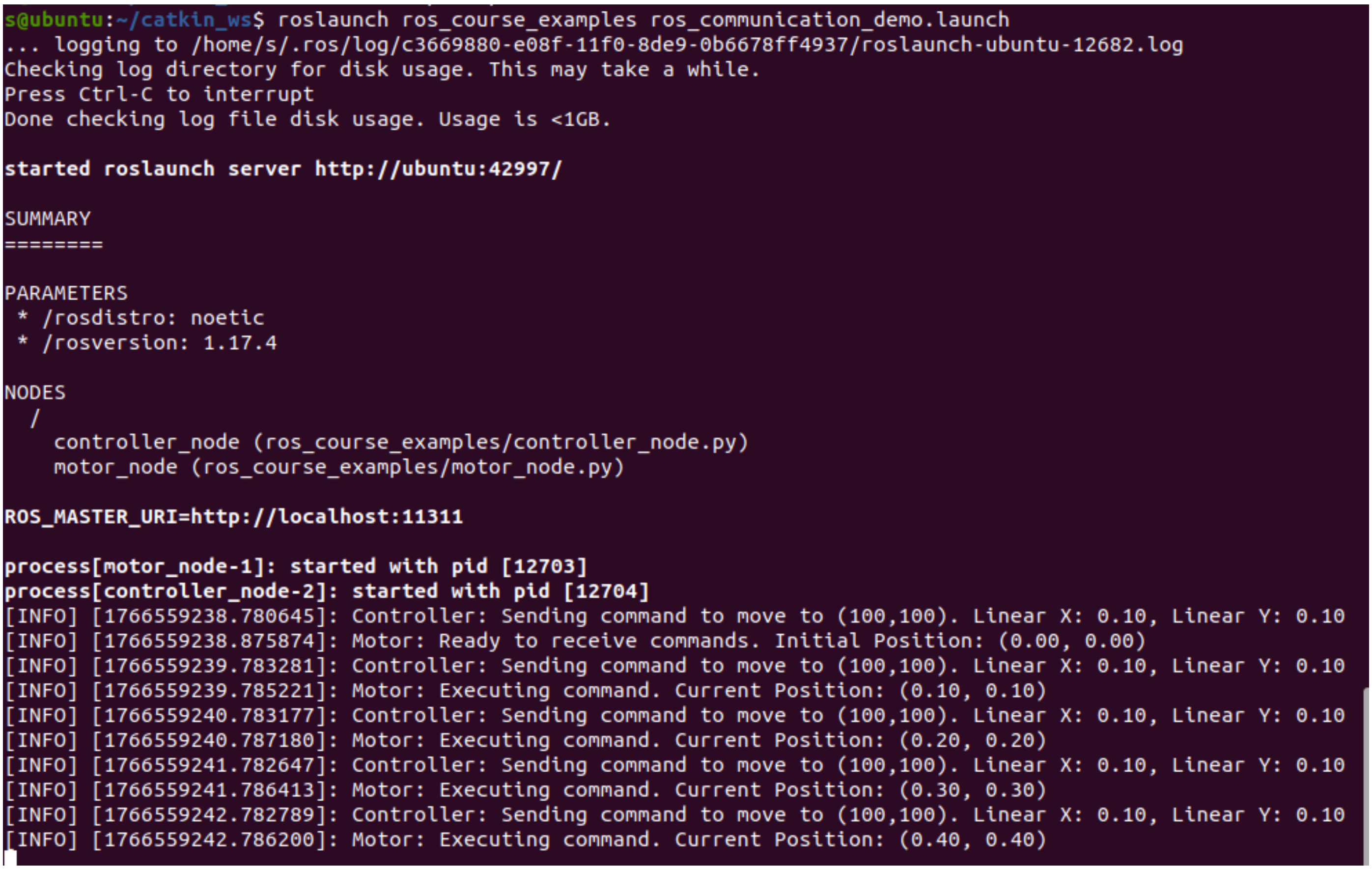
ROS框架中也有自己的“脚本”，可以一键启动多个结点。

关闭终端2和终端3，在终端3中执行：

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

```
roslaunch ros_course_examples ros_communication_demo.launch
```

你可以观察到，这两个结点都已经启动了。



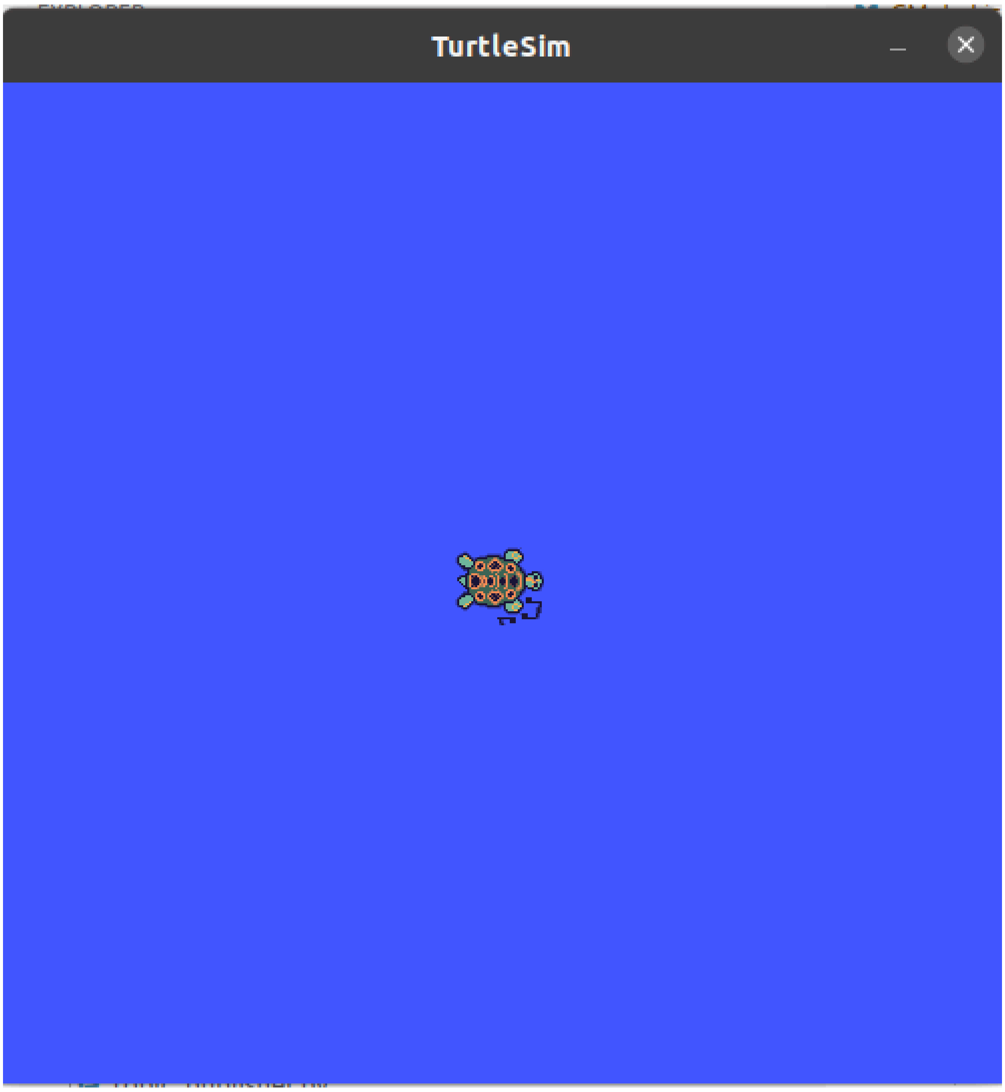
```
s@ubuntu:~/catkin_ws$ roslaunch ros_course_examples ros_communication_demo.launch
... logging to /home/s/.ros/log/c3669880-e08f-11f0-8de9-0b6678ff4937/roslaunch-ubuntu-12682.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ubuntu:42997/

SUMMARY
========

PARAMETERS
 * /rosdistro: noetic
 * /rosversion: 1.17.4

NODES
  /
    controller_node (ros_course_examples/controller_node.py)
    motor_node (ros_course_examples/motor_node.py)

ROS_MASTER_URI=http://localhost:11311

process[motor_node-1]: started with pid [12703]
process[controller_node-2]: started with pid [12704]
[INFO] [1766559238.780645]: Controller: Sending command to move to (100,100). Linear X: 0.10, Linear Y: 0.10
[INFO] [1766559238.875874]: Motor: Ready to receive commands. Initial Position: (0.00, 0.00)
[INFO] [1766559239.783281]: Controller: Sending command to move to (100,100). Linear X: 0.10, Linear Y: 0.10
[INFO] [1766559239.785221]: Motor: Executing command. Current Position: (0.10, 0.10)
[INFO] [1766559240.783177]: Controller: Sending command to move to (100,100). Linear X: 0.10, Linear Y: 0.10
[INFO] [1766559240.787180]: Motor: Executing command. Current Position: (0.20, 0.20)
[INFO] [1766559241.782647]: Controller: Sending command to move to (100,100). Linear X: 0.10, Linear Y: 0.10
[INFO] [1766559241.786413]: Motor: Executing command. Current Position: (0.30, 0.30)
[INFO] [1766559242.782789]: Controller: Sending command to move to (100,100). Linear X: 0.10, Linear Y: 0.10
[INFO] [1766559242.786200]: Motor: Executing command. Current Position: (0.40, 0.40)
```

实验 2.3：TurtleSim 通信实验

可以关闭这个程序了（**注意不要关闭roscore的终端**）

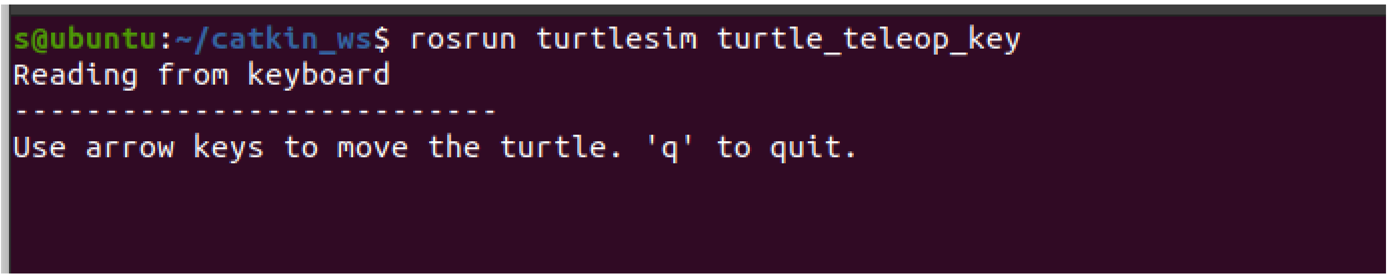
终端 2：启动 Turtle 仿真

```
rosrun turtlesim turtlesim_node
```



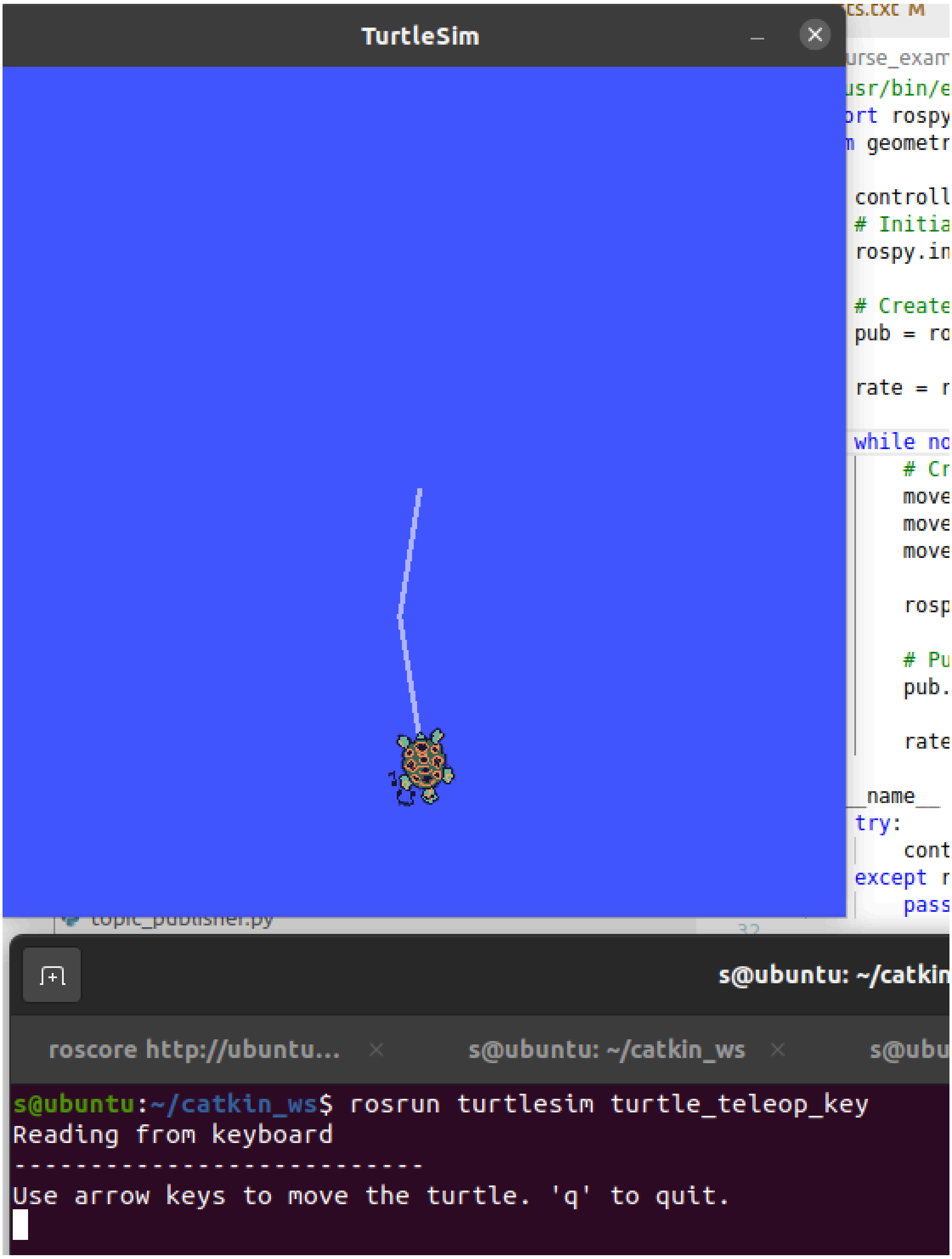
终端 3：启动键盘控制

```
rosrun turtlesim turtle_teleop_key
```



```
s@ubuntu:~/catkin_ws$ rosrun turtlesim turtle_teleop_key
Reading from keyboard
---------------------------
Use arrow keys to move the turtle. 'q' to quit.
```

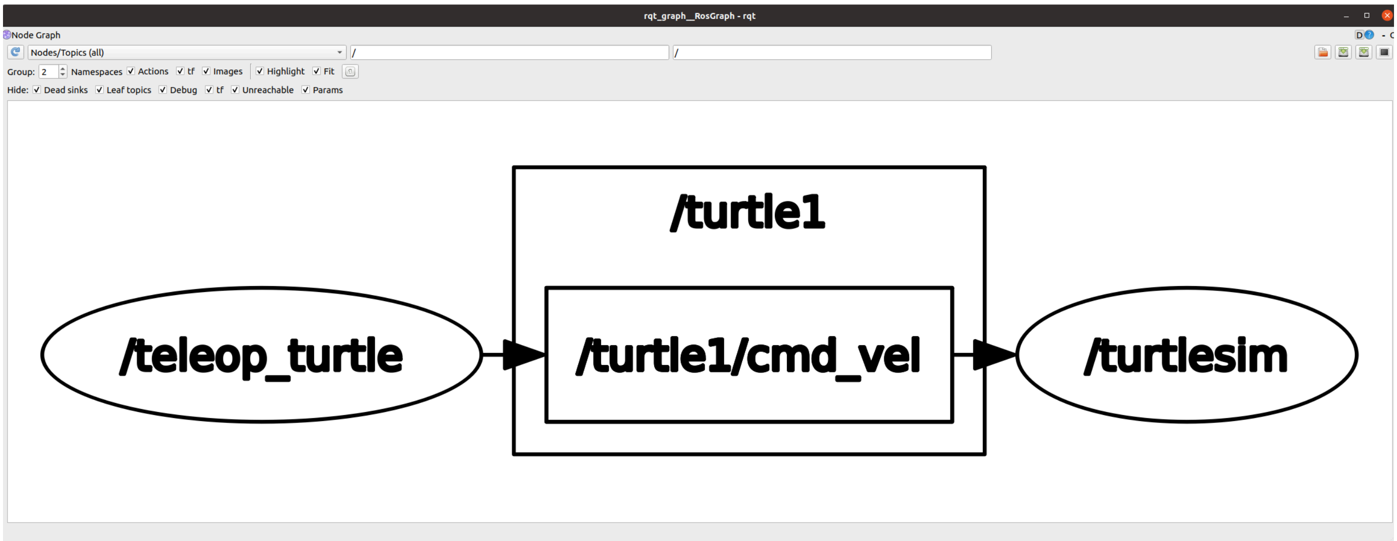
在终端3中按下方向键可以控制turtle运动。



```
s@ubuntu:~/catkin_ws$ rosrun turtlesim turtle_teleop_key
Reading from keyboard
---------------------------
Use arrow keys to move the turtle. 'q' to quit.
```

终端 4：可视化通信图

```
rqt_graph
```



rqt_graph__RosGraph - rqt

Node Graph

Nodes/Topics (all) / /

Group: 2 Namespaces ☑ Actions ☑ tf ☑ Images ☑ Highlight ☑ Fit

Hide: ☑ Dead sinks ☑ Leaf topics ☑ Debug ☑ tf ☑ Unreachable ☑ Params

通信流程分析：

1. turtle_teleop_key 节点监听键盘输入
2. 发布速度命令到 /turtle1/cmd_vel 话题
3. 发布到话题中的信息就是线速度和角速度，消息的类型是 geometry_msgs/Twist
4. turtlesim_node 订阅 /turtle1/cmd_vel
5. 接收速度命令并执行运动

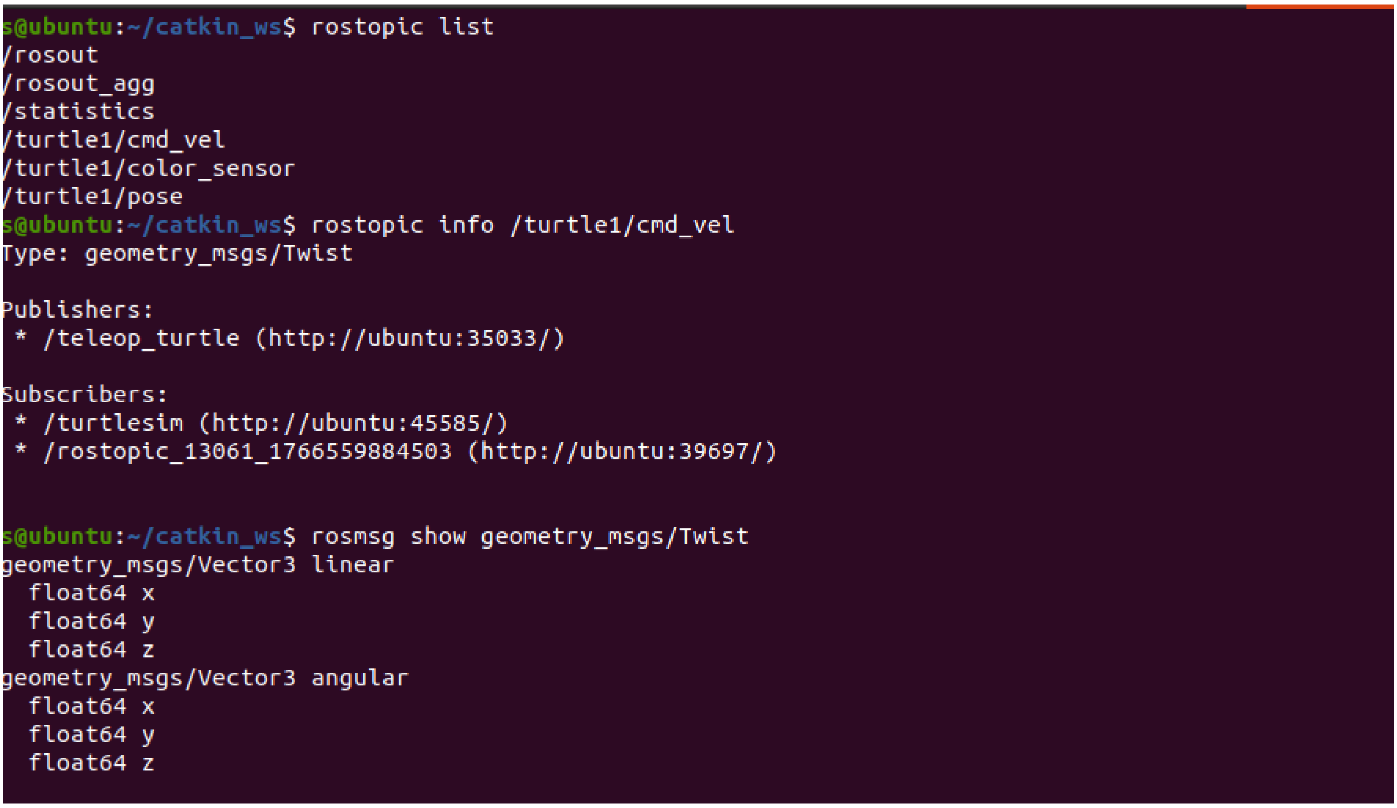
实验 2.4：查看话题信息

以上这些信息都可以通过ros指令来观察到。

```
# 列出所有话题
rostopic list
```

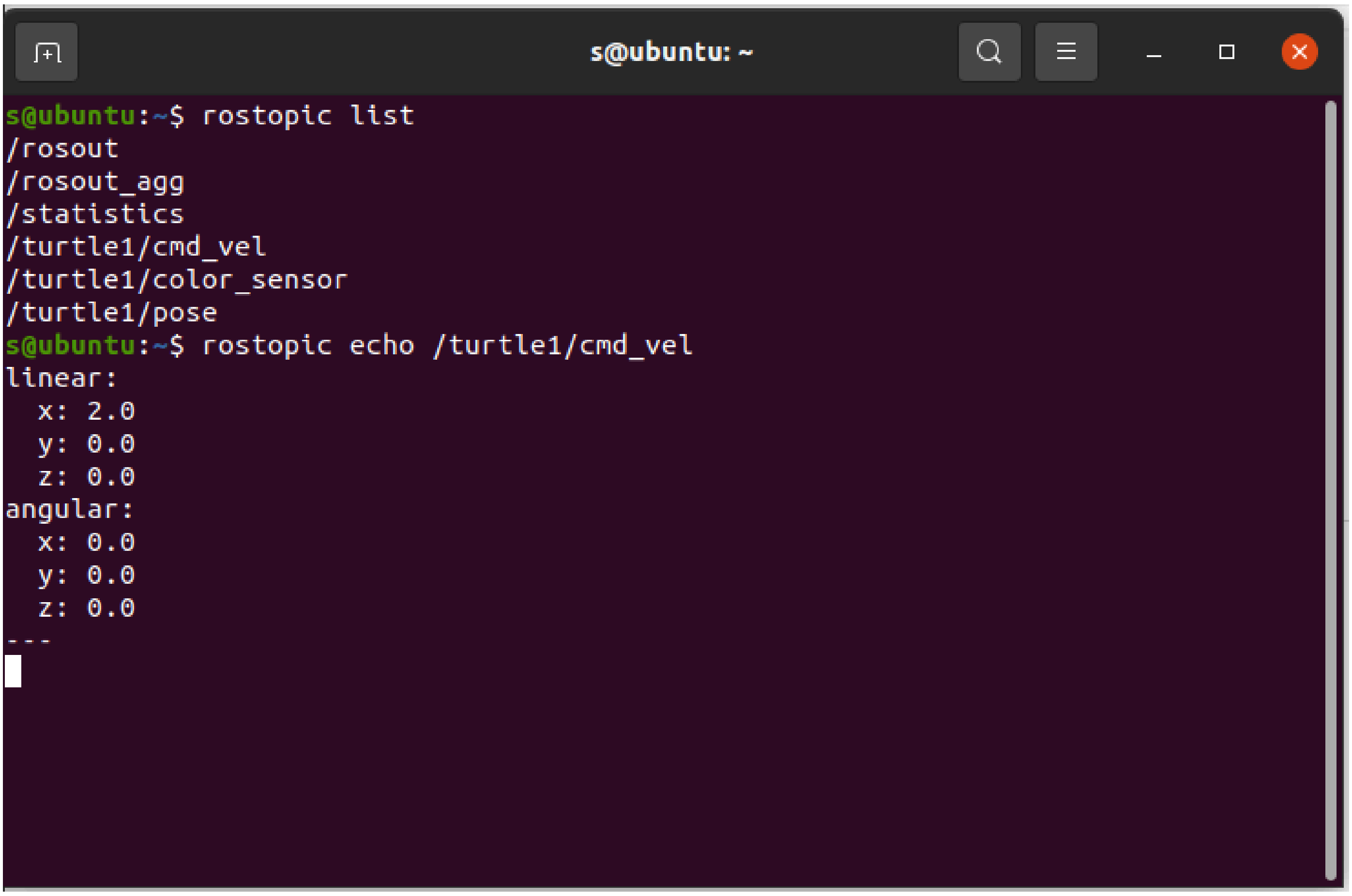
```
# 查看话题信息
rostopic info /turtle1/cmd_vel
```

```
# 查看消息类型定义
rosmsg show geometry_msgs/Twist
```



```
s@ubuntu:~/catkin_ws$ rostopic list
/rosout
/rosout_agg
/statistics
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
s@ubuntu:~/catkin_ws$ rostopic info /turtle1/cmd_vel
Type: geometry_msgs/Twist

Publishers:
 * /teleop_turtle (http://ubuntu:35033/)

Subscribers:
 * /turtlesim (http://ubuntu:45585/)
 * /rostopic_13061_1766559884503 (http://ubuntu:39697/)


s@ubuntu:~/catkin_ws$ rosmsg show geometry_msgs/Twist
geometry_msgs/Vector3 linear
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 angular
  float64 x
  float64 y
  float64 z
```

```
# 查看话题数据
rostopic echo /turtle1/cmd_vel
```



```
s@ubuntu:~$ rostopic list
/rosout
/rosout_agg
/statistics
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
s@ubuntu:~$ rostopic echo /turtle1/cmd_vel
linear:
  x: 2.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---
```

实验三：Gazebo 仿真环境

1. 实验目的

- 掌握 Gazebo 仿真器的使用
- 学会加载和保存仿真世界
- 理解 World 文件结构
- 掌握机器人模型的加载

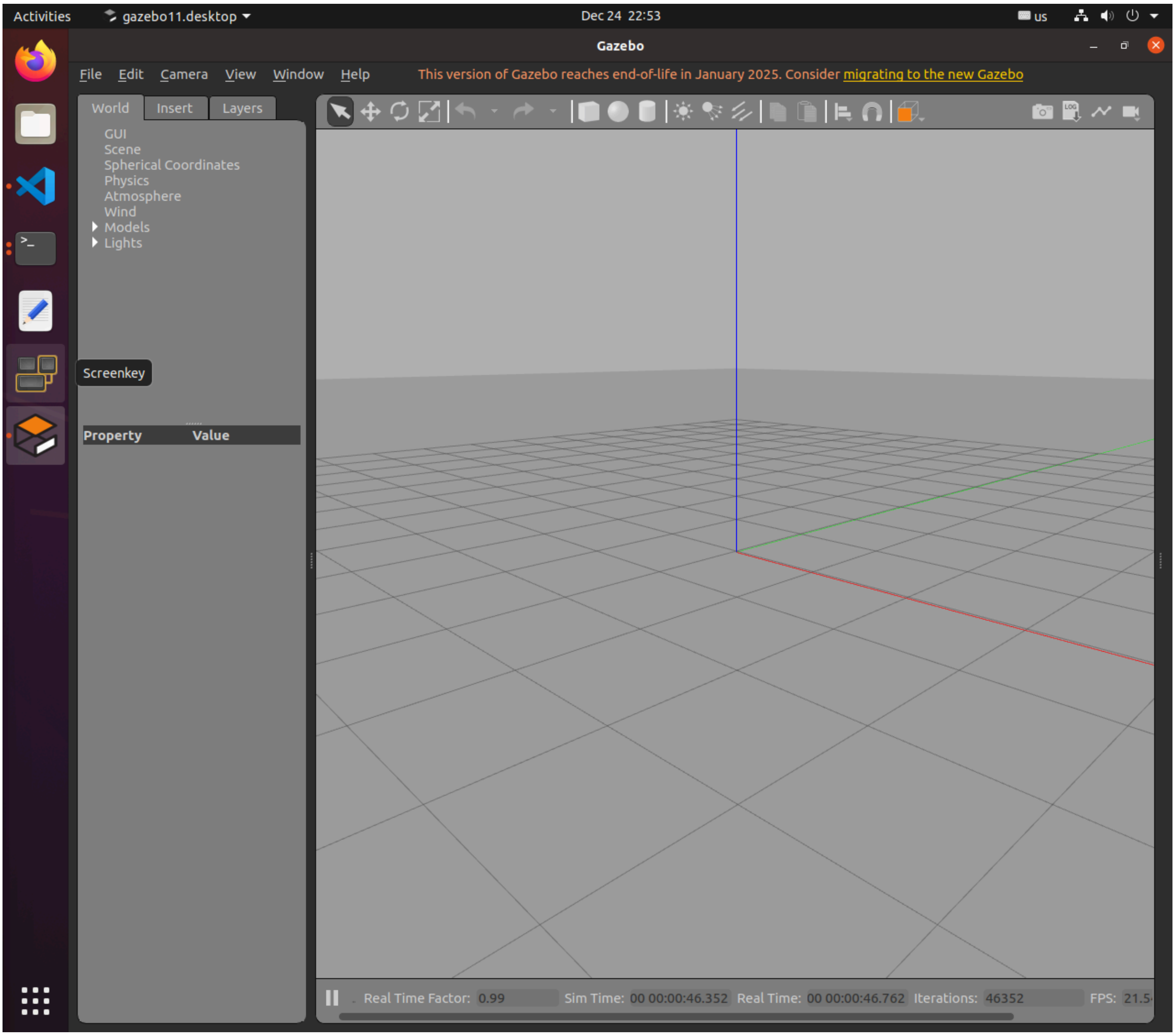
2. Gazebo 核心功能

- **物理引擎**： 模拟真实物理规律（重力、碰撞、摩擦）
- **传感器仿真**： 激光雷达、相机、IMU 等
- **ROS 集成**： 与 ROS 无缝通信
- **可视化**： 3D 场景渲染

3. 实验步骤

3.1 启动空白世界

```
gazebo
```

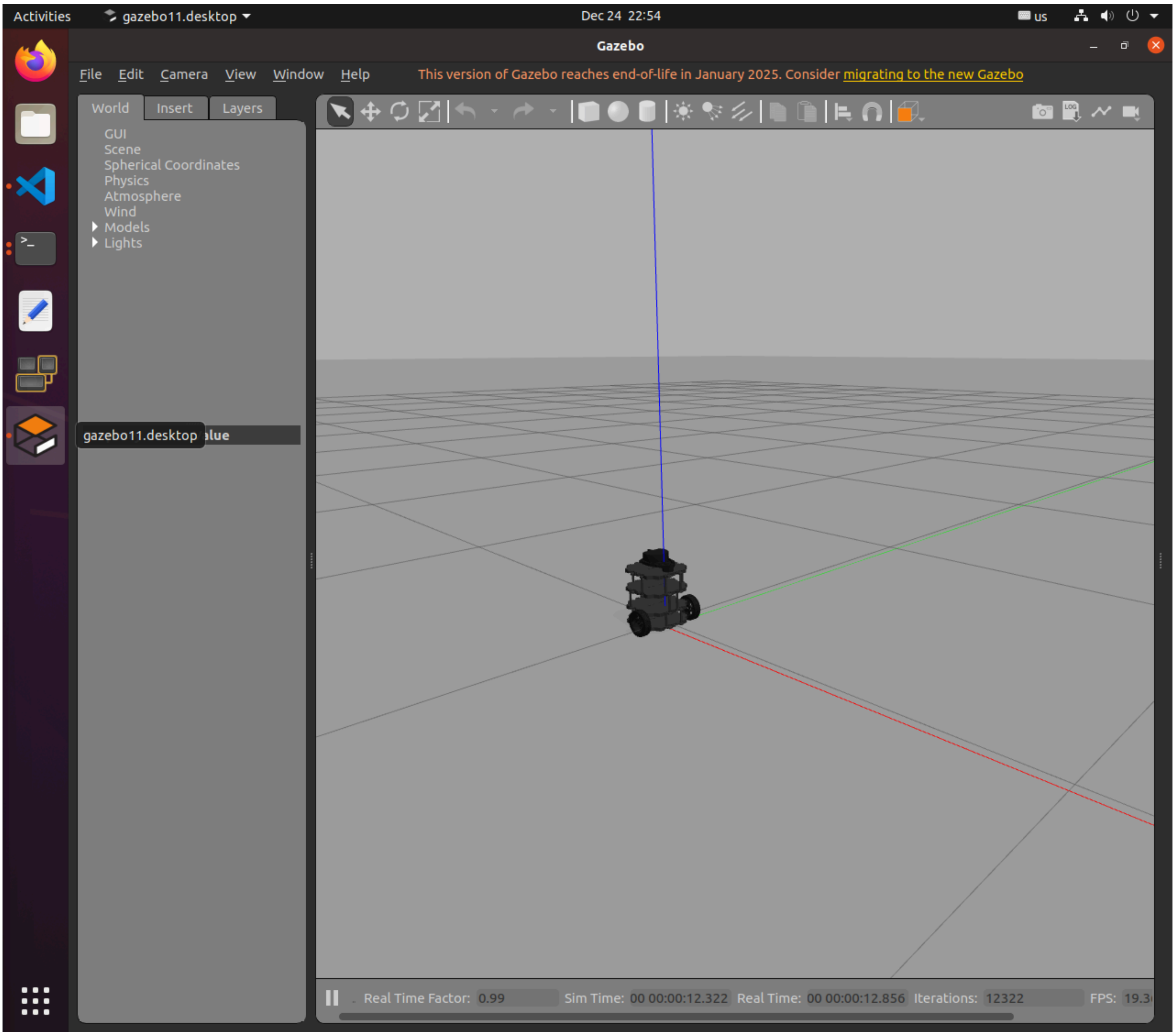


This version of Gazebo reaches end-of-life in January 2025. Consider migrating to the new Gazebo

Real Time Factor: 0.99 Sim Time: 00 00:00:46.352 Real Time: 00 00:00:46.762 Iterations: 46352 FPS: 21.5

或使用 ROS 启动

```
source ~/catkin_ws/devel/setup.bash
export TURTLEBOT3_MODEL=burger
roslaunch turtlebot3_gazebo turtlebot3_empty_world.launch
```



This version of Gazebo reaches end-of-life in January 2025. Consider migrating to the new Gazebo

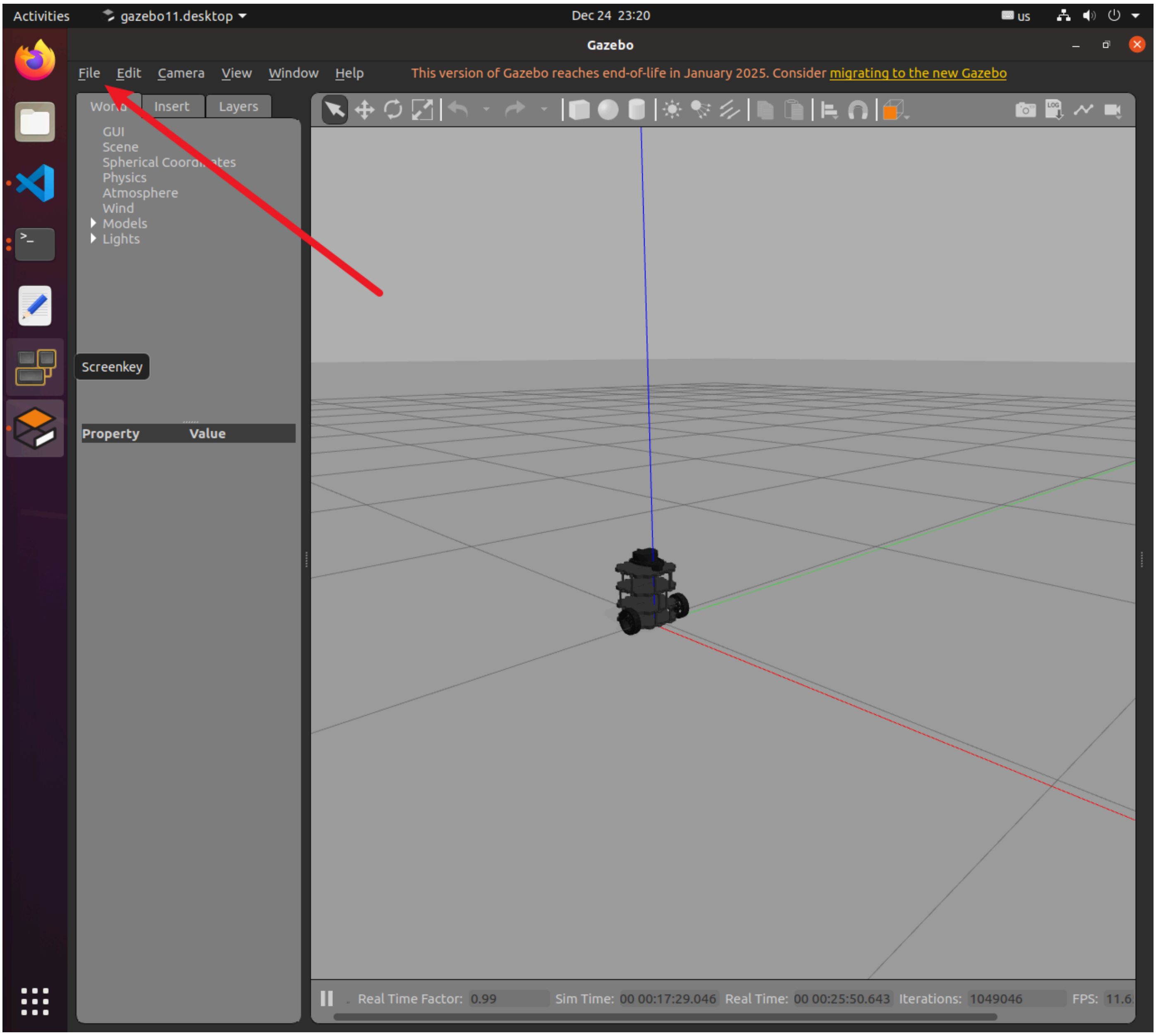
Real Time Factor: 0.99 Sim Time: 00 00:00:12.322 Real Time: 00 00:00:12.856 Iterations: 12322 FPS: 19.3

3.2 构建自定义场景

1. **插入模型**： 从左侧面板拖拽物体到场景
2. **调整参数**：

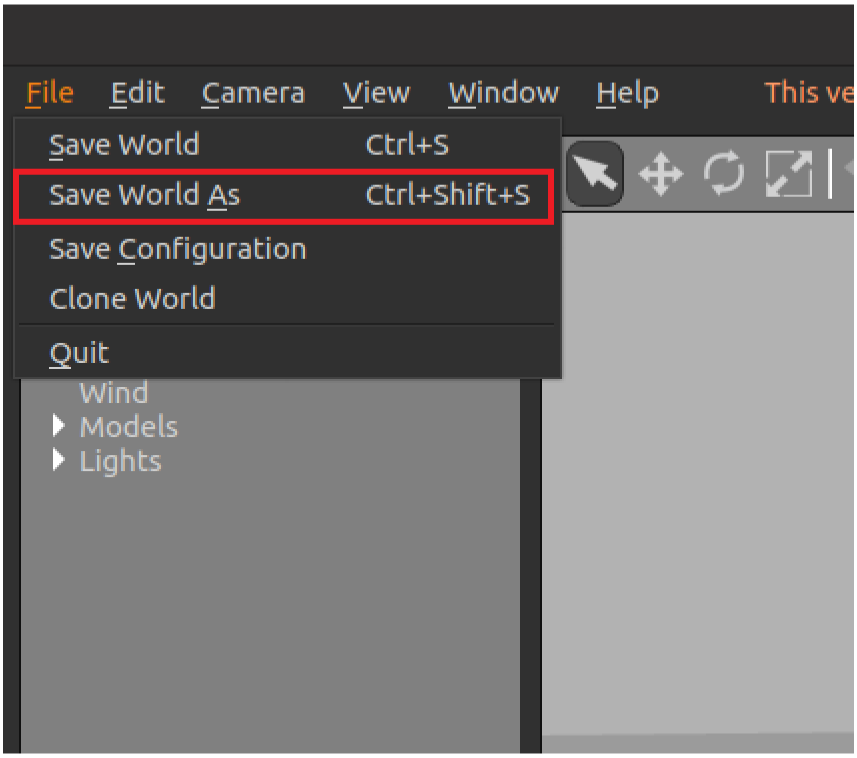
- Position (x, y, z)：位置坐标
- Orientation (roll, pitch, yaw)：姿态角度
- Scale：缩放大小

1. **保存世界**： `File → Save World As → my_world.world`



This version of Gazebo reaches end-of-life in January 2025. Consider migrating to the new Gazebo

Real Time Factor: 0.99 Sim Time: 00 00:17:29.046 Real Time: 00 00:25:50.643 Iterations: 1049046 FPS: 11.6

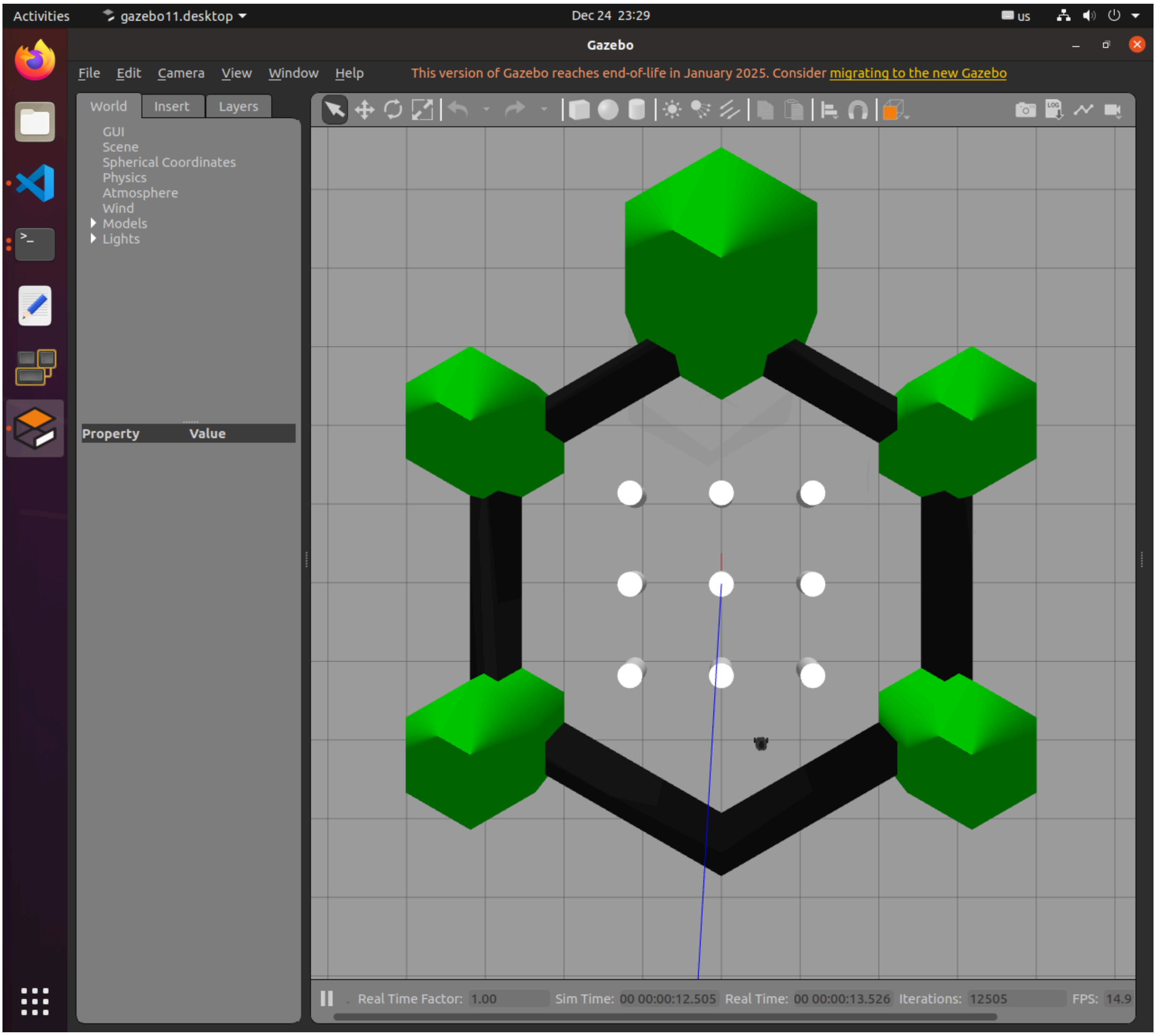


3.3 加载自定义世界

```
gazebo my_world.world
```

方法 2：ROS launch 文件

```
source ~/catkin_ws/devel/setup.bash
export TURTLEBOT3_MODEL=burger
roslaunch turtlebot3_gazebo turtlebot3_world.launch
```



This version of Gazebo reaches end-of-life in January 2025. Consider migrating to the new Gazebo

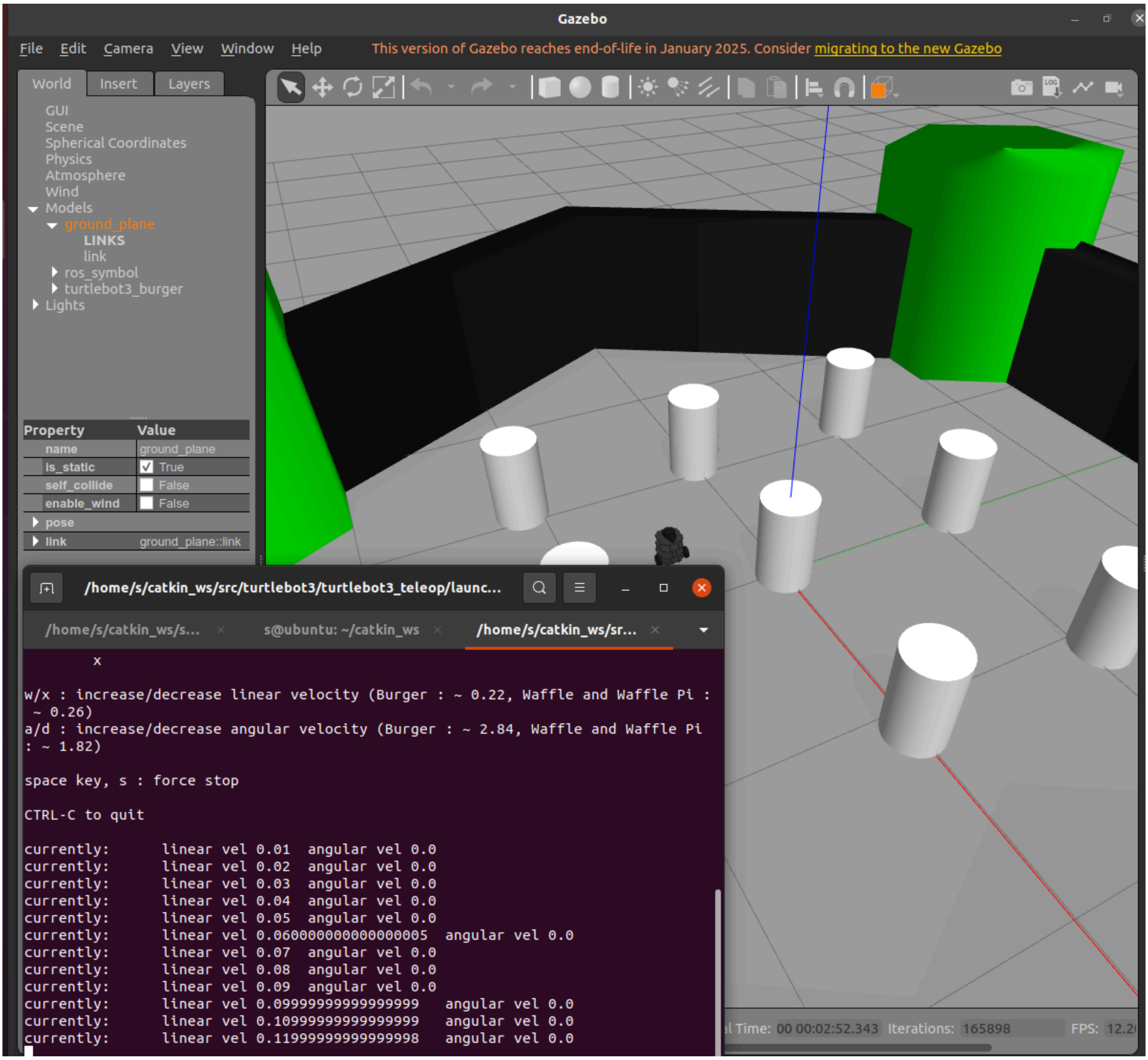
Real Time Factor: 1.00 Sim Time: 00 00:00:12.505 Real Time: 00 00:00:13.526 Iterations: 12505 FPS: 14.9

3.4 控制机器人运动

```
source ~/catkin_ws/devel/setup.bash
export TURTLEBOT3_MODEL=burger
roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

操作说明：

- `W/A/S/D` 或 `方向键`：控制移动
- `X`：停止
- `Q/Z`：增加/减少速度



Property	Value
name	ground_plane
is_static	☑ True
self_collide	□ False
enable_wind	□ False
pose	
link	ground_plane::link

```
        x

w/x : increase/decrease linear velocity (Burger : ~ 0.22, Waffle and Waffle Pi :
 ~ 0.26)
a/d : increase/decrease angular velocity (Burger : ~ 2.84, Waffle and Waffle Pi
: ~ 1.82)

space key, s : force stop

CTRL-C to quit

currently:      linear vel 0.01  angular vel 0.0
currently:      linear vel 0.02  angular vel 0.0
currently:      linear vel 0.03  angular vel 0.0
currently:      linear vel 0.04  angular vel 0.0
currently:      linear vel 0.05  angular vel 0.0
currently:      linear vel 0.060000000000000005  angular vel 0.0
currently:      linear vel 0.07  angular vel 0.0
currently:      linear vel 0.08  angular vel 0.0
currently:      linear vel 0.09  angular vel 0.0
currently:      linear vel 0.09999999999999999   angular vel 0.0
currently:      linear vel 0.10999999999999999   angular vel 0.0
currently:      linear vel 0.11999999999999998   angular vel 0.0
```

实验四：RViz 可视化

1. 实验目的

- 掌握 RViz 可视化工具的使用
- 学会添加和配置显示项
- 理解传感器数据的可视化表示

- 掌握界面交互和视角控制

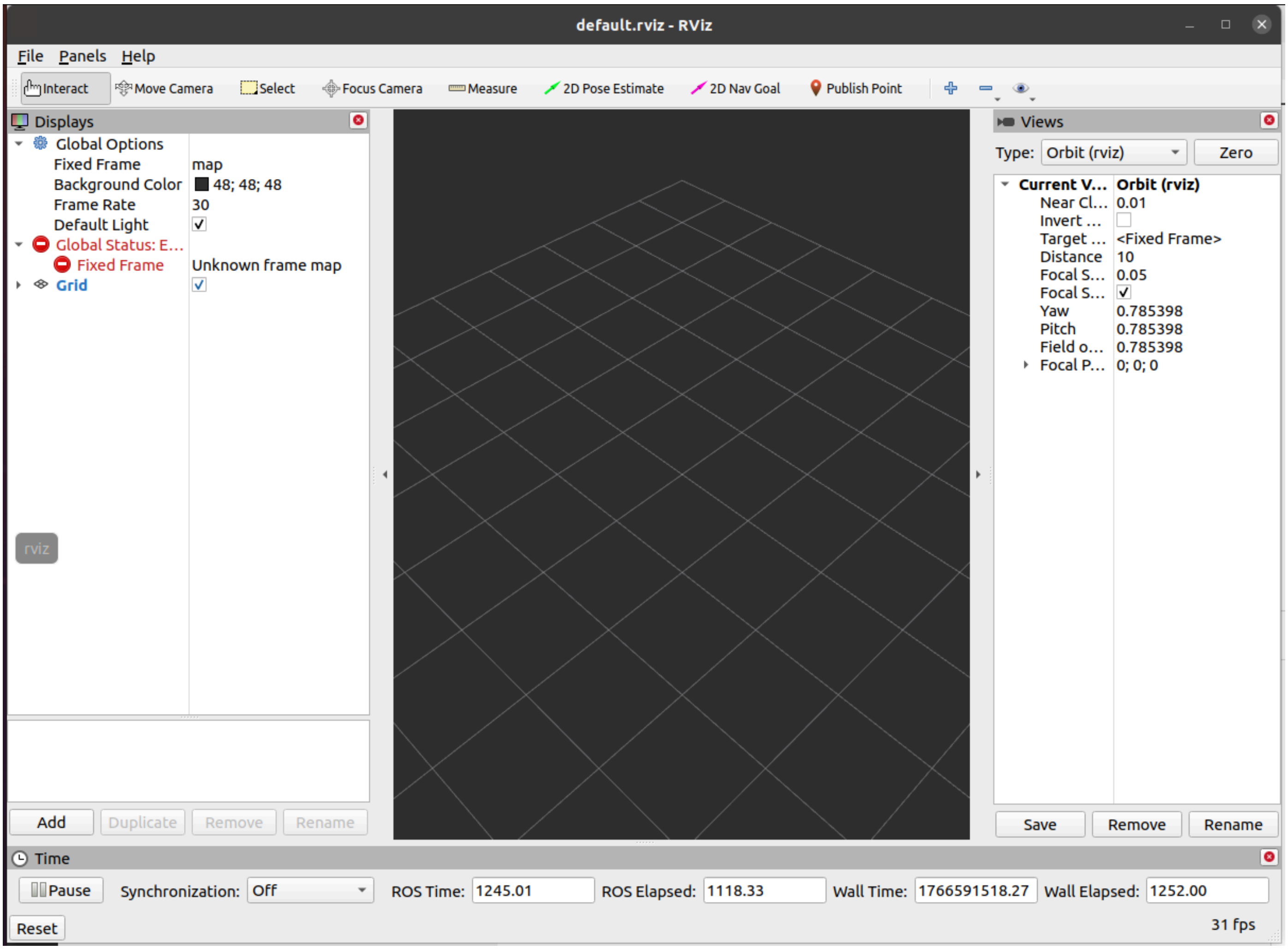
2. RViz 可显示的数据类型

- **Robot Model**： 3D 机器人模型
- **LaserScan**： 激光雷达扫描数据
- **PointCloud2**： 3D 点云数据
- **TF**： 坐标系变换关系
- **Image**： 相机图像
- **Odometry**： 里程计轨迹
- **Path**： 规划路径
- **Map**： 占据栅格地图

3. RViz 启动方式介绍

直接启动

```
rviz
```



Global Options
Fixed Frame map
Background Color ■ 48; 48; 48
Frame Rate 30
Default Light ☑
Global Status: E...
Fixed Frame Unknown frame map
Grid ☑

Type: Orbit (rviz) Zero
Current V... Orbit (rviz)
Near Cl... 0.01
Invert ... □
Target ... <Fixed Frame>
Distance 10
Focal S... 0.05
Focal S... ☑
Yaw 0.785398
Pitch 0.785398
Field o... 0.785398
Focal P... 0; 0; 0

Synchronization: Off ROS Time: 1245.01 ROS Elapsed: 1118.33 Wall Time: 1766591518.27 Wall Elapsed: 1252.00

31 fps

3.1 RViz 界面基础操作

视角控制

- 🖱️ **鼠标左键拖拽：** 旋转视角
- 🖱️ **鼠标滚轮：** 缩放视图
- 🖱️ **Shift + 左键拖拽：** 平移视图
- 🖱️ **Shift + 滚轮：** 上下平移
- 🖱️ **鼠标中键拖拽：** 平移（某些系统）

界面布局说明

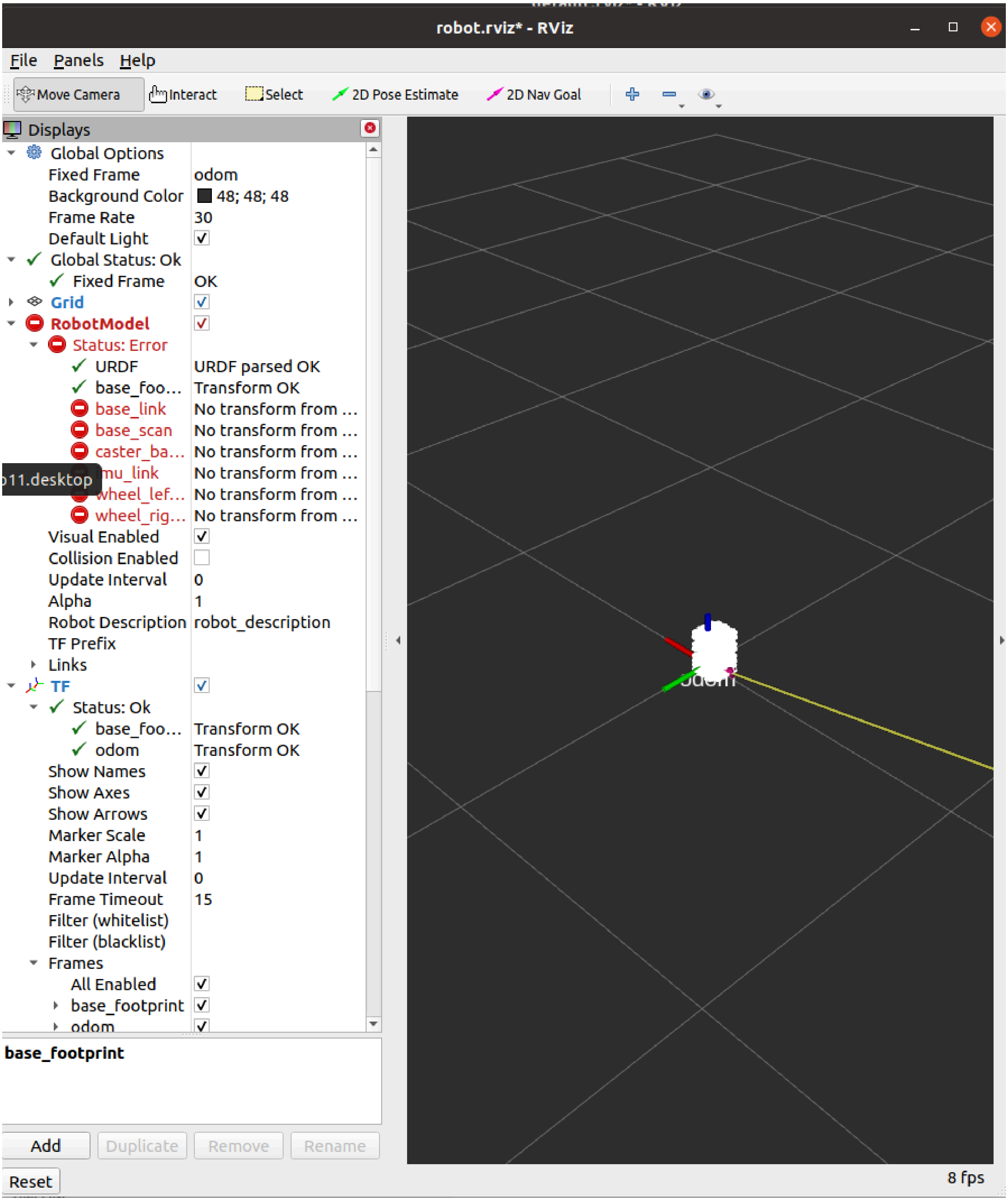
```
┌─────────────────────────────────────────┐
│  菜单栏 (File / View / Panels)          │
├──────────┬─────────────────────┬────────┤
│          │                     │        │
│ Displays │   3D 视图窗口        │ 工具栏 │
│ 面板     │  (主显示区域)        │ Views  │
│          │                     │ Time   │
└──────────┴─────────────────────┴────────┘
```

4. 实验步骤

4.1 启动仿真和可视化

```
# 终端 1：启动 gazebo
roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

```
# 终端 2：启动 RViz
roslaunch turtlebot3_rviz_launchers view_robot.launch
```



Global Options	
Fixed Frame	odom
Background Color	■ 48; 48; 48
Frame Rate	30
Default Light	☑
✓ Global Status: Ok	
✓ Fixed Frame	OK
Grid	☑
RobotModel	☑
Status: Error	
✓ URDF	URDF parsed OK
✓ base_foo…	Transform OK
base_link	No transform from …
base_scan	No transform from …
caster_ba…	No transform from …
mu_link	No transform from …
wheel_lef…	No transform from …
wheel_rig…	No transform from …
Visual Enabled	☑
Collision Enabled	□
Update Interval	0
Alpha	1
Robot Description	robot_description
TF Prefix	
Links	
TF	☑
✓ Status: Ok	
✓ base_foo…	Transform OK
✓ odom	Transform OK
Show Names	☑
Show Axes	☑
Show Arrows	☑
Marker Scale	1
Marker Alpha	1
Update Interval	0
Frame Timeout	15
Filter (whitelist)	
Filter (blacklist)	
Frames	
All Enabled	☑
base_footprint	☑
odom	☑

base_footprint

4.2 添加显示项（Display）详细步骤

示例1：添加激光雷达数据

1. 点击左下角 **"Add"** 按钮
2. 在弹出窗口中选择 **"By display type"** 标签
3. 找到并双击 **"LaserScan"**
4. 在左侧 Displays 面板中展开 **"LaserScan"**

5. 配置参数：

```
Topic: /scan           # 点击下拉选择 /scan
Size (m): 0.05         # 调整点的大小
Style: Points          # 显示样式
Color Transformer: Intensity  # 颜色映射
```

6. 观察红色扫描点显示障碍物位置

示例2：添加里程计轨迹

1. 点击 **"Add"** → 选择 **"Odometry"**
2. 配置参数：

```
Keep: 100              # 保留 100 条历史轨迹
Shape: Arrow           # 显示为箭头
Color: 0; 0; 255       # 蓝色 (RGB)
Alpha: 1.0             # 不透明度
Shaft Length: 0.3      # 箭头长度
```

3. 机器人移动时会显示蓝色轨迹线

示例3：添加坐标系（TF）

1. 点击 **"Add"** → 选择 **"TF"**
2. 配置参数：

```
✓ Show Axes           # 显示坐标轴
✓ Show Arrows         # 显示箭头
Marker Scale: 0.5     # 调整坐标轴大小
Update Interval: 0    # 更新频率（0=最快）
```

3. 观察红绿蓝箭头（代表 X/Y/Z 轴）

4.3 常用显示项配置表

显示类型	推荐话题	作用	关键参数
RobotModel	(默认)	显示机器人3D模型	`Robot Description: robot_description`
LaserScan	`/scan`	激光雷达扫描数据	`Size: 0.05`, `Style: Points`
Odometry	`/odom`	里程计轨迹	`Keep: 100`, `Shape: Arrow`
Path	`/move_base/global_plan`	全局规划路径	`Color: 255;255;0` (黄色)
TF	(无需设置)	坐标系关系	`Show Names: ✓`, `Marker Scale: 0.5`
Image	`/camera/rgb/image_raw`	相机图像	`Transport Hint: raw`
Map	`/map`	占据栅格地图	`Color Scheme: map`

4.4 调整 Fixed Frame（参考坐标系）

什么是 Fixed Frame?

- RViz 中所有数据的显示都需要一个参考坐标系
- 不同场景需要选择不同的 Fixed Frame

选择建议

场景	Fixed Frame	效果
观察机器人运动	`odom`	视角跟随机器人
查看全局地图	`map`	固定不动的世界视角
调试传感器	`base_link`	视角锁定在机器人上

设置方法

1. 在顶部 **"Global Options"** 展开

2. 点击 **"Fixed Frame"** 下拉菜单
3. 选择 `odom` 或 `map`

⚠️ **注意**： 如果 Fixed Frame 设置错误，所有显示项会变灰色或不显示。

4.5 查看 TF 树

方法1：在 RViz 中查看

- 添加 TF 显示项
- 勾选 "Show Names" 和 "Show Axes"
- 观察坐标系之间的层级关系

方法2：生成 TF 树图（推荐）

```
#
rosrun tf2_tools view_frames.py

# 等待 5 秒后自动生成 frames.pdf
evince frames.pdf  # 或用其他 PDF 阅读器打开
```

查看 TF 变换

TF 树结构示例

```
└─ odom
    └─ base_footprint
        └─ base_link
            ├─ camera_link
            │   └─ camera_rgb_frame
            ├─ imu_link
            └─ base_scan (laser_link)
```

6. 常见问题排查

问题	原因	解决方法
❌ 显示项变红色/灰色	话题未发布	`rostopic list` 检查话题是否存在

问题	原因	解决方法
❌ 看不到机器人模型	Fixed Frame 错误	改为 `odom` 或 `base_link`
❌ 激光数据不显示	Topic 选择错误	确认为 `/scan`
❌ TF 显示 "No transform"	TF 树不完整	检查 `rosrun tf view_frames`

课后练习建议

- 多练习 Linux 命令行操作
- 尝试修改示例代码参数，观察效果
- 使用 `rqt_graph` 和 `rostopic` 工具分析系统
- 阅读 ROS Wiki 官方文档

推荐资源

- ROS Wiki
- Gazebo Tutorials
- TF Tutorials