基于TurtleBot3的仿真slam建模与导航(Gazebo建模)

前言

- 快速配置环境可以直接看第八章，已经打包好。
  前面为分步骤过程，在第四章安装了Cartographer，但因为时间问题，后续没有用，可以跳过。

一、TurtleBot3简介与SLAM简介

1、什么是TurtleBot3？
TurtleBot3 是一个小型，低成本，完全可编程，基于 ROS 的移动机器人。它旨在用于教育，研究，产品原型和爱好应用的目的。TurtleBot3 的目标是大幅降低平台的尺寸和价格，而不会牺牲性能，功能和质量。由于提供了其他选项，如底盘，计算机和传感器，TurtleBot3 可以通过各种方式进行定制。TurtleBot3 应用 了SBC（单板计算机），深度传感器和 3D 打印的最新技术进步等技术。
1、什么是SLAM？
在研究机器人自动行驶的时候，人们注意到，为了实现自动驾驶功能，机器人必须实现对自身的定位和对周围环境的感知。在逐步探索中，研究者开始借用激光雷达、相机等先进的传感设备完成定位与环境感知功能。
同步定位与建图（Simultaneous Localization and Mapping，SLAM）是在上世纪80年代被提出的，起初发展的算法皆采用激光雷达作为定位与建图的工具，随着稀疏性问题的解决，相机也被引入SLAM领域，如今SLAM技术在向多传感器融合的方向发展，激光雷达、深度相机、IMU惯导等正成为SLAM技术的常见解决方案。

二、安装ROS

- 本文使用Ubuntu18.04安装ROS Melodic

1 . 更改ROS 源(国内源)

```
sudo sh -c '. /etc/lsb-release && echo "deb http://mirrors.ustc.edu.cn/ros/ubuntu/ $DI
```

2.设置密钥

```
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-key 421C365BD9FF1F7
```

3.更新软件包列表

```
sudo apt update
```

4.安装ROS Melodic

```
sudo apt-get install ros-melodic-desktop-full
```

5.初始化rosdep

```
sudo rosdep init
```

6.更新rosdep

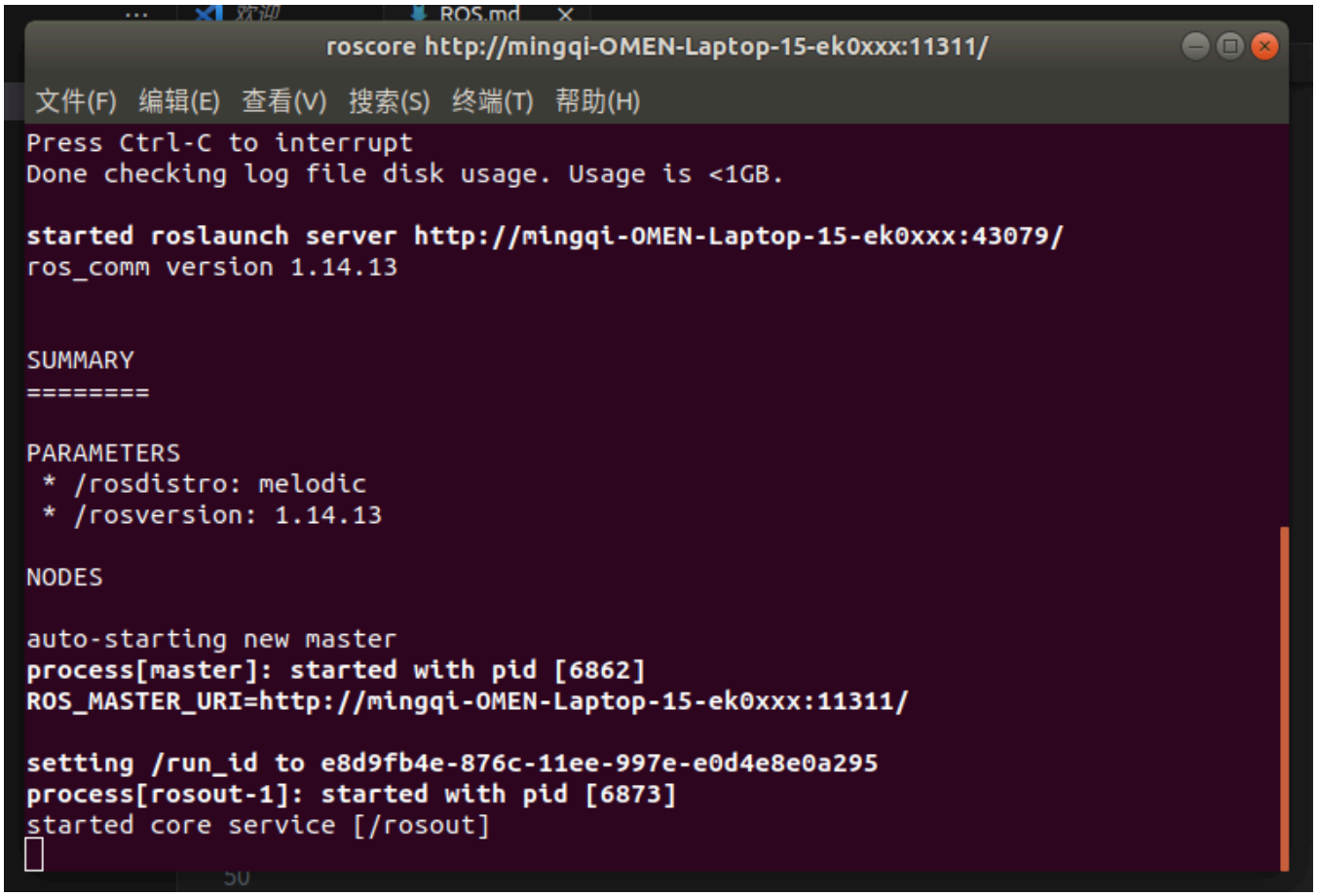
```
sudo rosdep update
```

7.将ros添加到环境变量

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

8.运行ROS检测是否安装成功

```
roscore
```



- 如图所示，这ROS Melodic安装成功

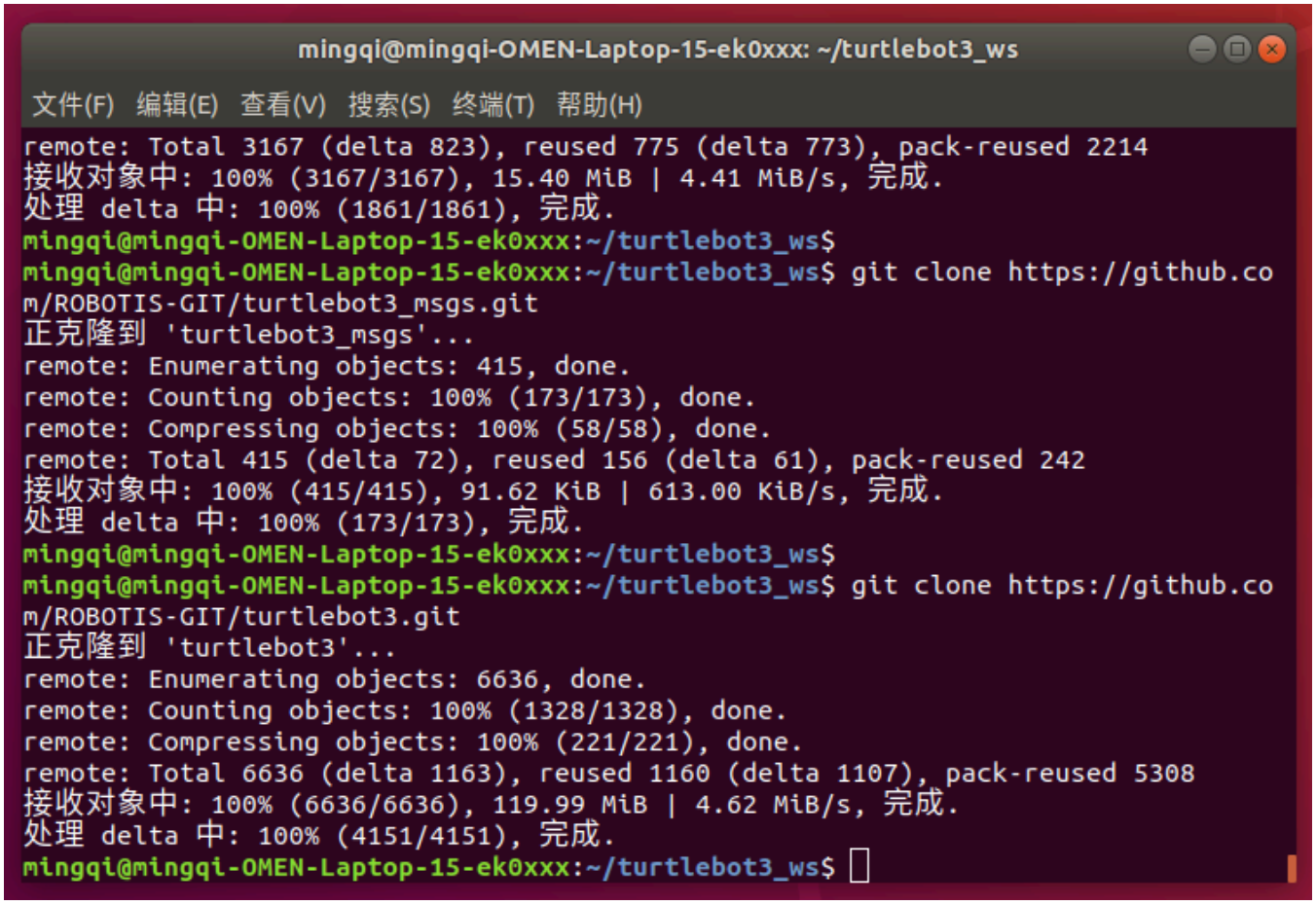
三、安装Turtlebot3

1.安装Turtlebot3依赖库

```
sudo apt-get install ros-melodic-joy ros-melodic-teleop-twist-joy ros-melodic-teleop-
```

2.下载Turtlebot3资源包

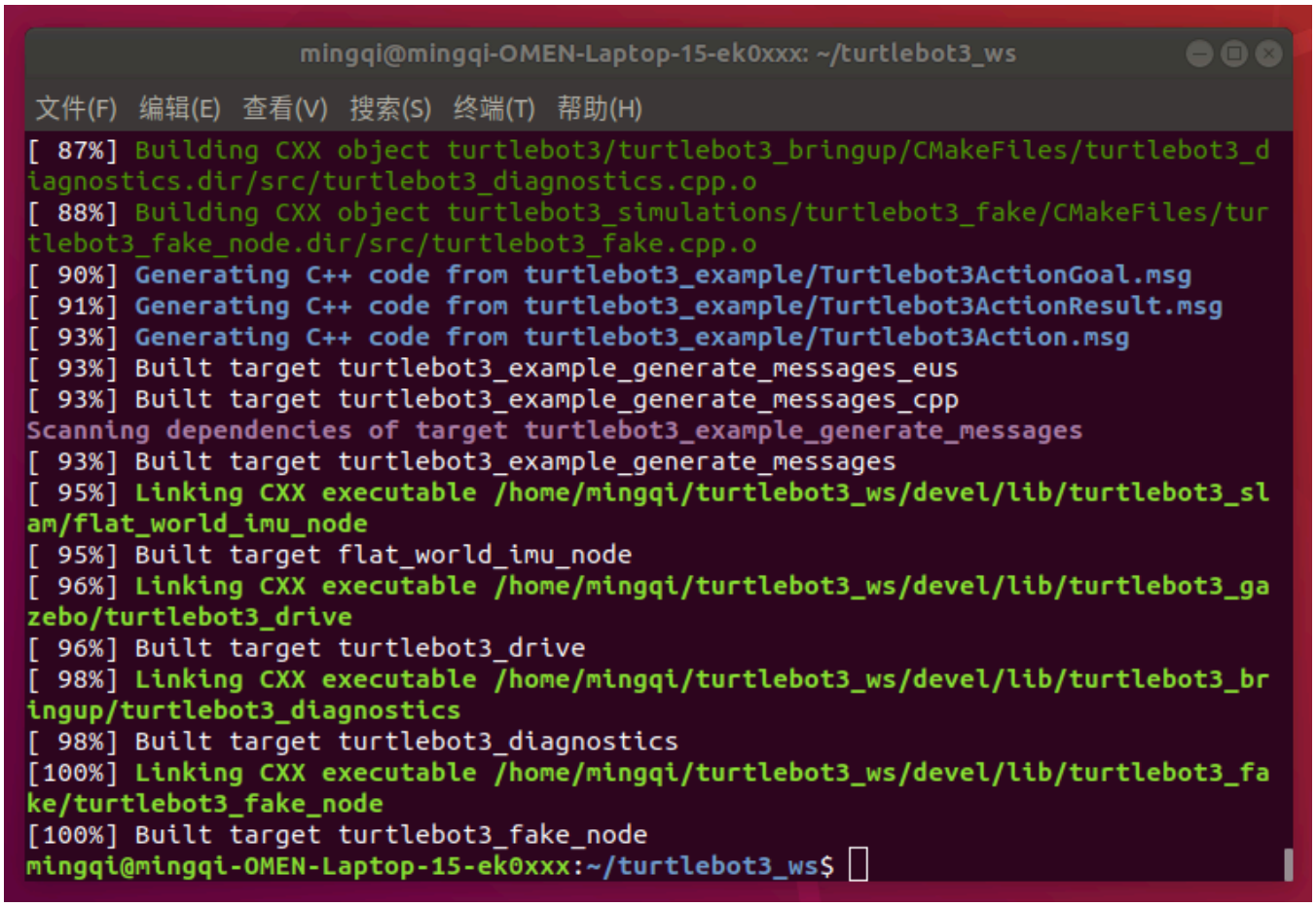
```
mkdir -p ~/turtlebot3_ws/src/
cd ~/turtlebot3_ws/src/
git clone https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git
git clone https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git
git clone https://github.com/ROBOTIS-GIT/turtlebot3.git
```



- git Turtlebot3资源包成功

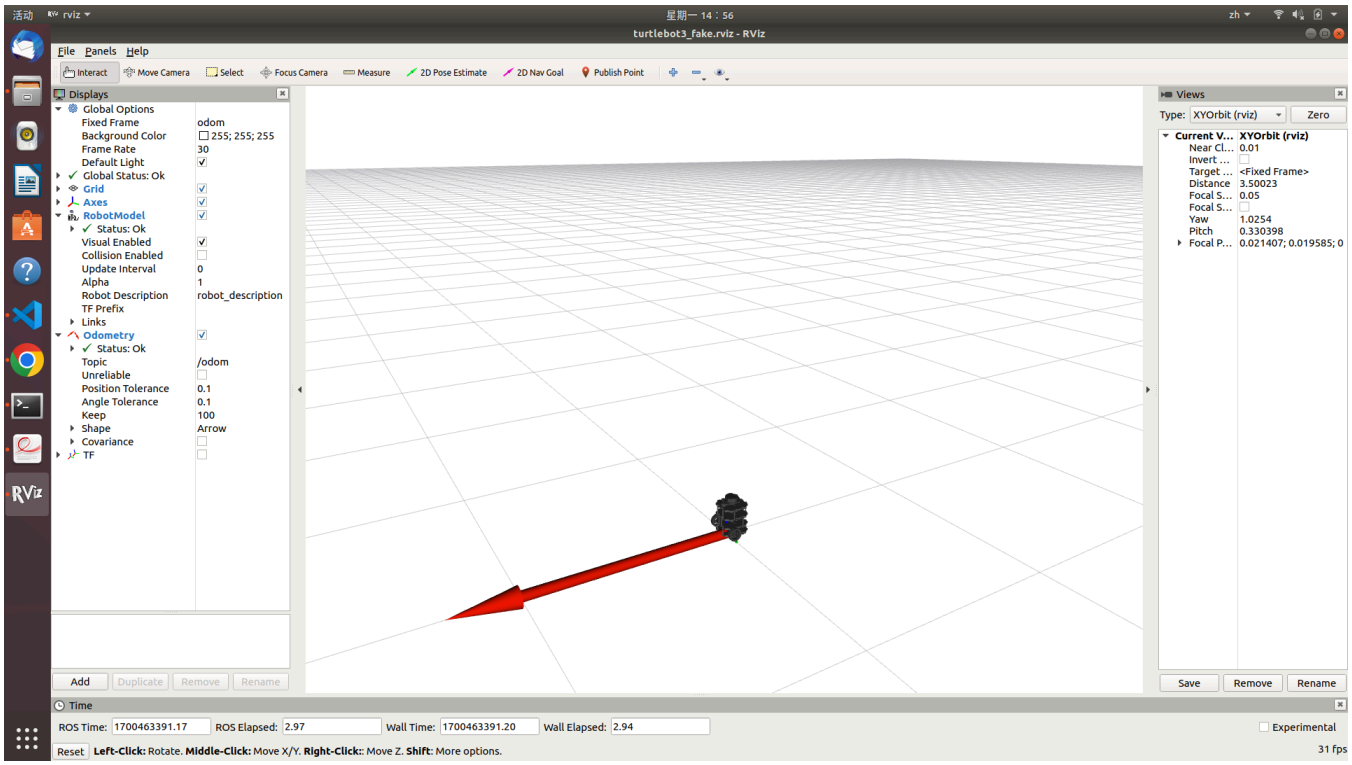
3.编译Turtlebot3资源包

```
cd ~/turtlebot3_ws
catkin_make
```



4.测试Turtlebot3是否安装成功

```
export TURTLEBOT3_MODEL=burger
source ~/turtlebot3_ws/devel/setup.bash
roslaunch turtlebot3_fake turtlebot3_fake.launch
```



5.把Turtlebot3加入环境变量

- 打开.bashrc文件

```
gedit ~/.bashrc
```

将下面两行**Shell**命令拷贝到文件最后

```
export TURTLEBOT3_MODEL=burger
source ~/turtlebot3_ws/devel/setup.bash
```

四、安装Cartographer

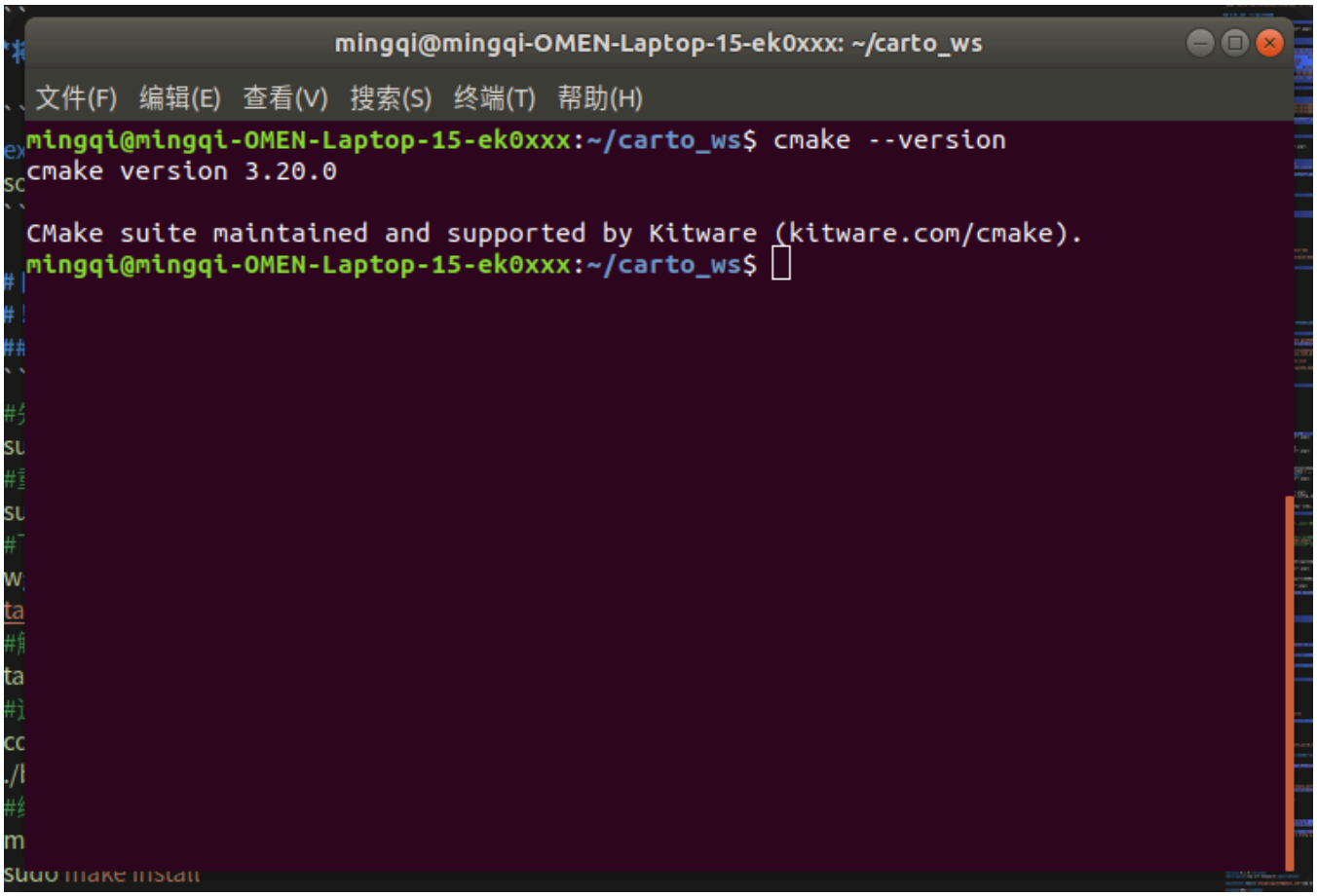
！！安装**Cartographer**，需要先升级**Cmake**

1、升级Cmake到3.2.0

```
#先卸载3.10的CMake
sudo apt remove cmake
#重新安装依赖
sudo apt-get install build-essential libssl-dev
#下载Cmake3.2.0
wget https://github.com/Kitware/CMake/releases/download/v3.20.0/cmake-3.20.0.tar.gz
#解压缩
tar -zxvf cmake-3.20.0.tar.gz
#运行bootstrap脚本
cd cmake-3.20.0
./bootstrap
#编译安装
make
sudo make install
```

- 检测Cmake安装是否成功

```
cmake --version
```



- 安装成功

2、安装Cartographer

！！按照Cartographer官网教程安装

- 安装一些工具：wstool, rosdep和Ninja

```
sudo apt-get update
sudo apt-get install -y python-wstool python-rosdep ninja-build stow
```

- 安装好这些工具后,创建一个cartographer_ros工作区

```
#创建文件夹
mkdir carto_ws
cd carto_ws
#下载源码
wstool init src
wstool merge -t src https://raw.githubusercontent.com/cartographer-project/cartographe
wstool update -t src
```

- 更改代码，防止后面报错
  打开**cartographer/package.xml**将第**46**行 `<depend>libabsl-dev</depend>` 进行注释
- 安装依赖项

```
sudo rosdep init
rosdep update
rosdep install --from-paths src --ignore-src --rosdistro=${ROS_DISTRO} -y
```

- 安装abseil-cpp 库

```
src/cartographer/scripts/install_abseil.sh

 //由于版本冲突，您可能需要卸载ROS abseil-cpp
 sudo apt-get remove ros-${ROS_DISTRO}-abseil-cpp
```

- 编译

```
 catkin_make_isolated --install --use-ninja
```

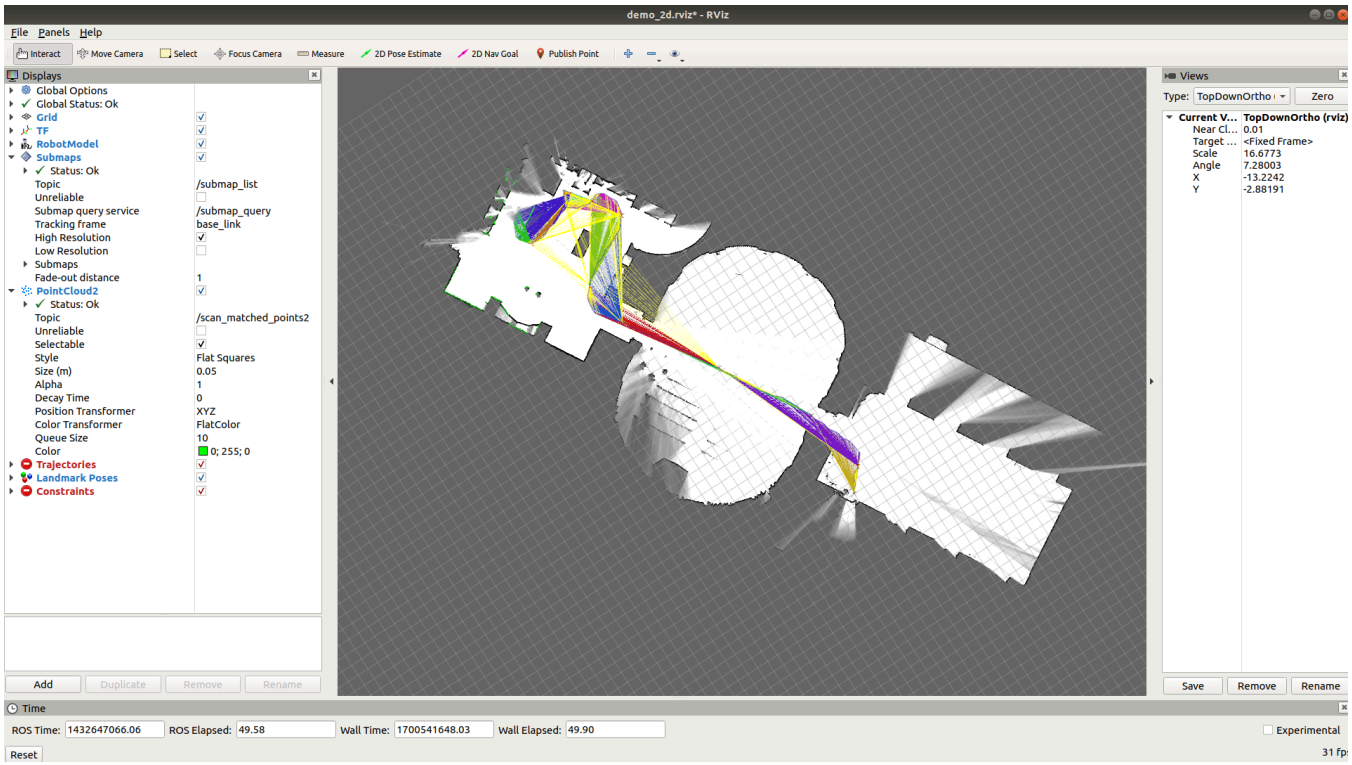
- 将 cartographer_ws 添加到环境变量中

```
 echo 'source ~/carto_ws/install_isolated/setup.bash' >> ~/.bashrc
source ~/.bashrc
```

3.官方demo验证

- 2D-Demo与结果

```
#下载demo包
wget -P ~/Downloads https://storage.googleapis.com/cartographer-public-data/bags/backpa
#测试
roslaunch cartographer_ros demo_backpack_2d.launch bag_filename:=${HOME}/Downloads/cart
```



五、Gazebo仿真环境搭建

！！经过测试默认安装的**Gazebo9**在建立模型放置窗户与门时，会闪退，所以更新为**Gazebo11**

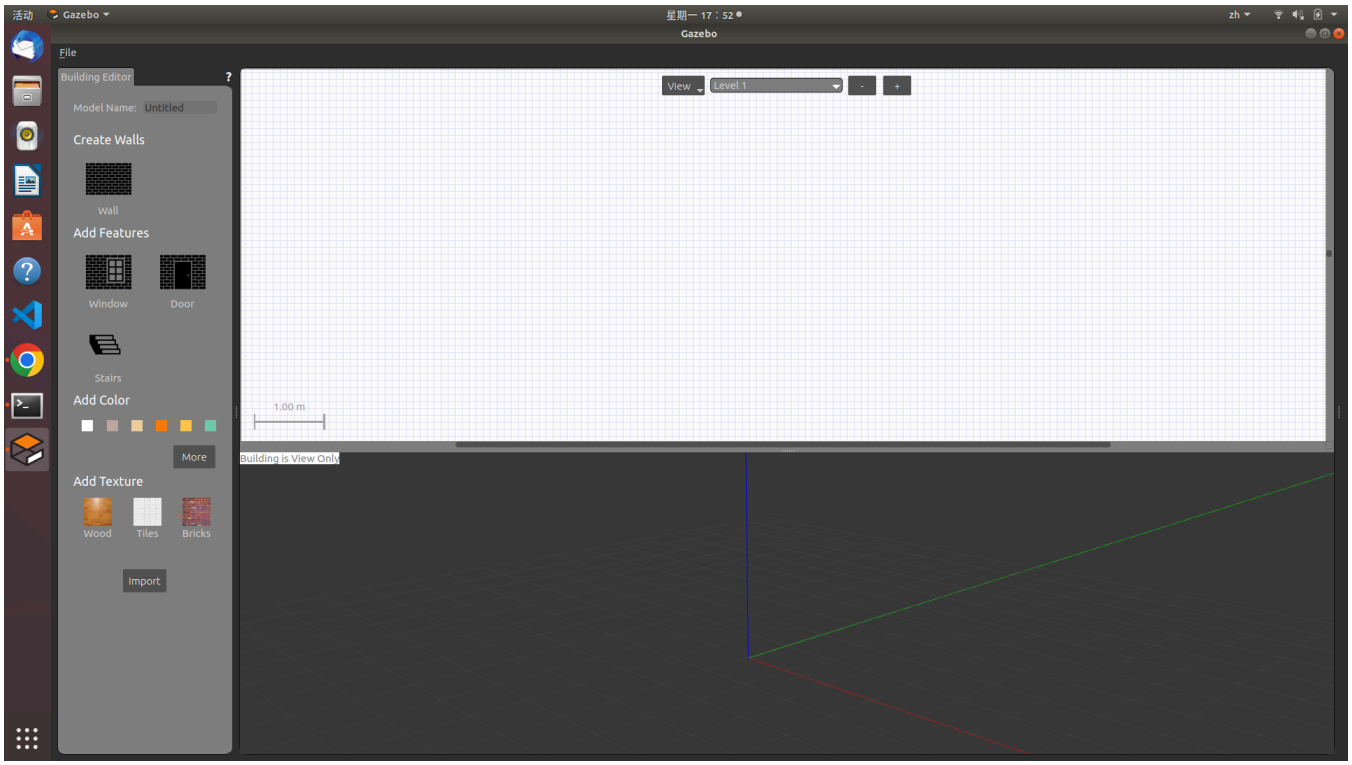
```
 # 查看一下Gazebo的插件
dpkg -l | grep gazebo
# 卸载全部插件
sudo apt-get remove gazebo9 gazebo9-common gazebo9-plugin-base libgazebo9:amd64 libgaze
# 配置镜像
sudo sh -c 'echo "deb http://packages.osrfoundation.org/gazebo/ubuntu-stable `lsb_relea
#查看文件写入是否正确
cat /etc/apt/sources.list.d/gazebo-stable.list
#设置Key
wget https://packages.osrfoundation.org/gazebo.key -O - | sudo apt-key add -
#更新
sudo apt-get update
#下载、安装Gazebo 11
sudo apt-get install gazebo11
sudo apt-get install libgazebo11-dev
#安装Gazebo的Ros插件
sudo apt install ros-melodic-gazebo11-*
#检查是否安装成功，直接打开Gazebo即可
gazebo
```

1.打开Gazebo

```
gazebo
```

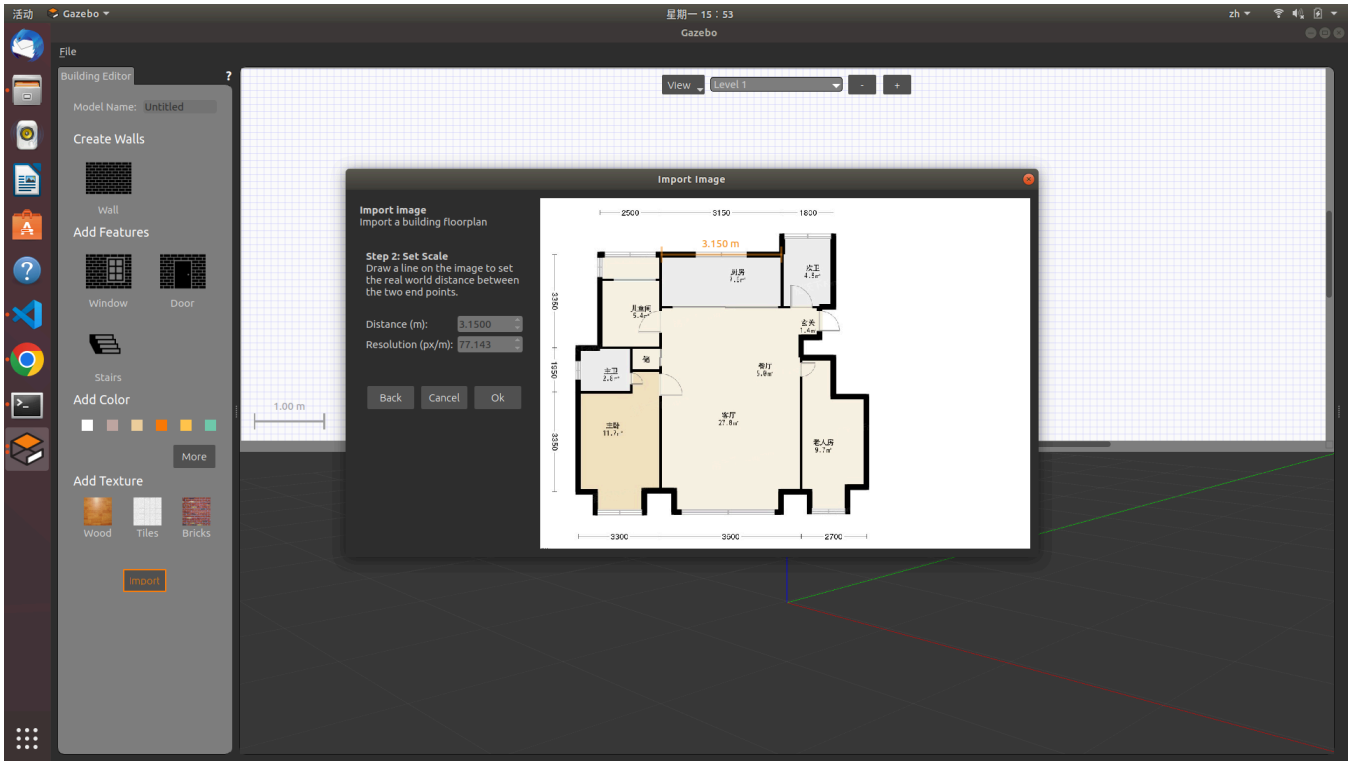
2. 打开建筑编辑器

点击上方“Edit”中“Buiding Edit ”进入建筑编辑器。



3.用户型图作为参考

点击左下角“import”，选择一张户型照片导入。



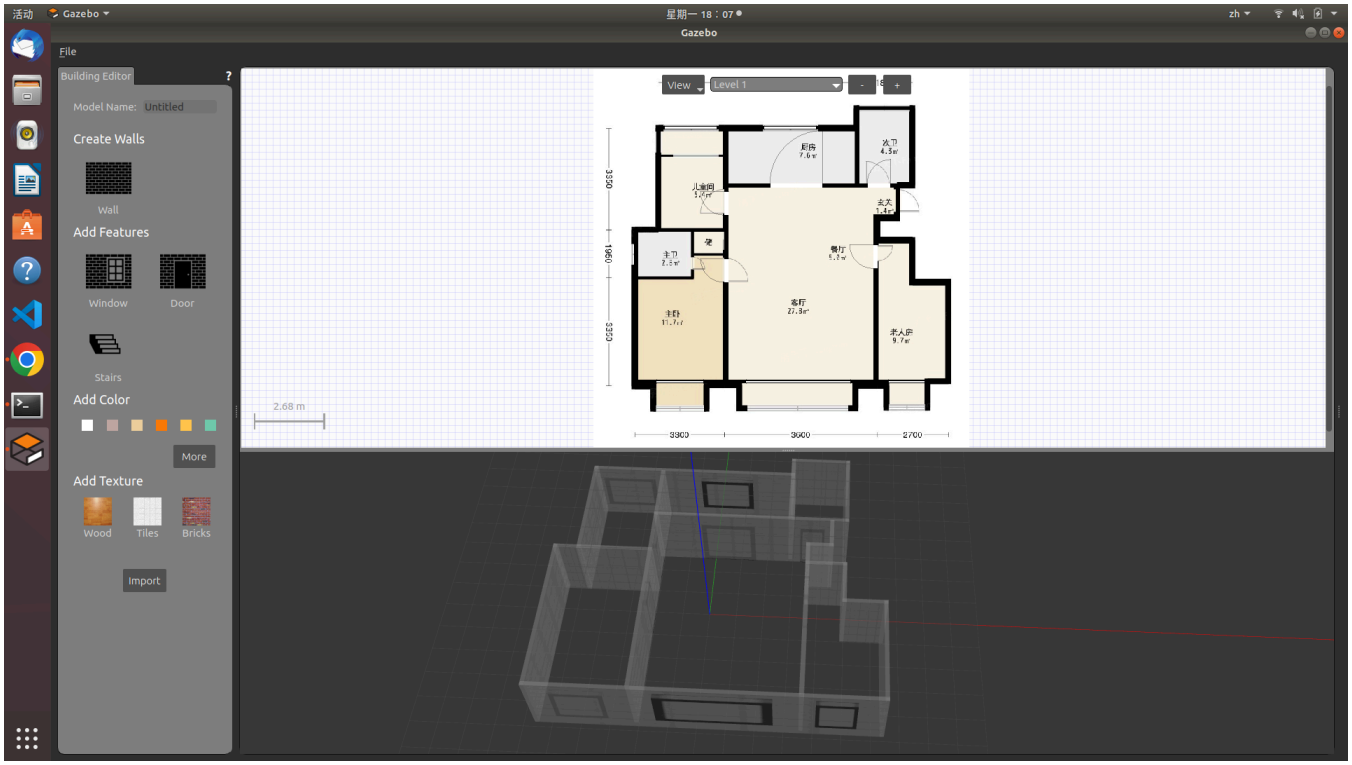
户型图如下

4.创建模型

- 点击左边“ Wall ”，对着房屋轮廓和内部墙壁描边，下方的3D视图会同步显示墙壁信息。
- 再按照户型图进行门窗的设置，点击左边的“Window ”和“ Door ”，在相应墙壁位置选择即可。
- **可以双击2D视图里的墙壁进行详细参数设置。**

5.保存模型

如图，完成模型绘制。



点击左上角“File” → “Save” 保存, 保存为model.config和model.sdf文件。

6.生成.world文件

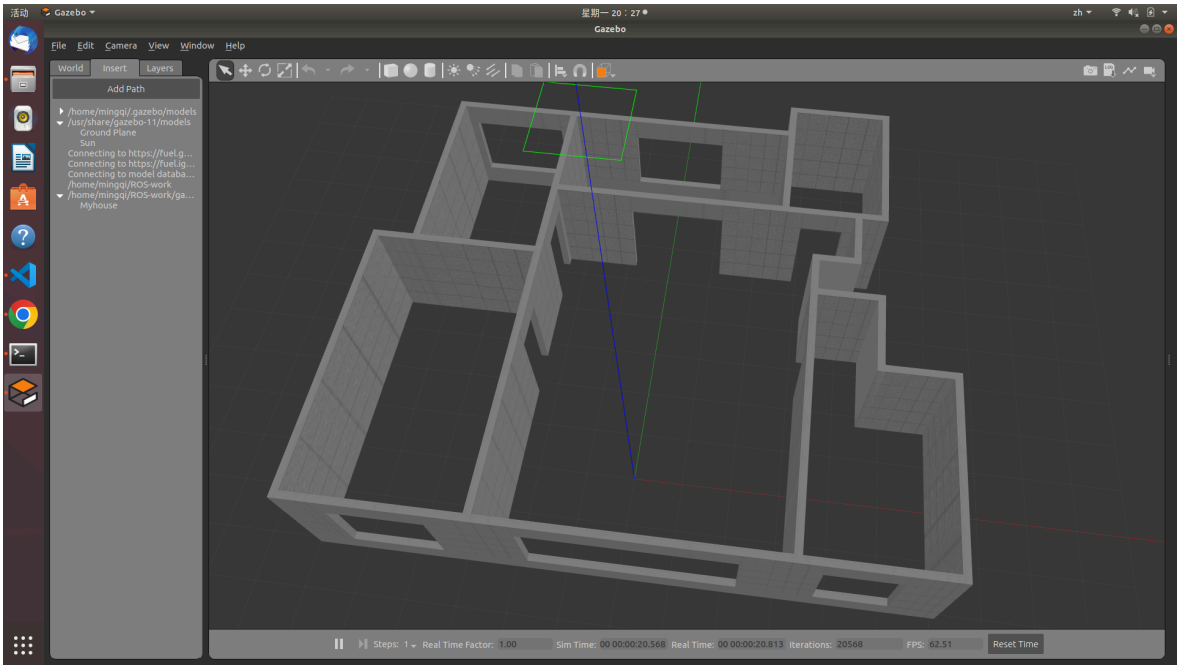
在gazebo_mode文件中有三个模型，但因为我的电脑性能不行，导入就卡顿，所以用了gazebo的库。

- ganzebo库安装

```
 # 进入.gazebo文件夹，用于存放Gazebo模型和相关配置文件
cd ~/.gazebo/
# 安装Git工具
sudo apt install git
# 从https://gitee.com/dva7777/gazebo_models.git克隆Gazebo模型
git clone https://gitee.com/dva7777/gazebo_models.git
# 将克隆的gazebo_models文件夹重命名为models，并放到.gazebo文件夹下
mv gazebo_models/ ./models
```

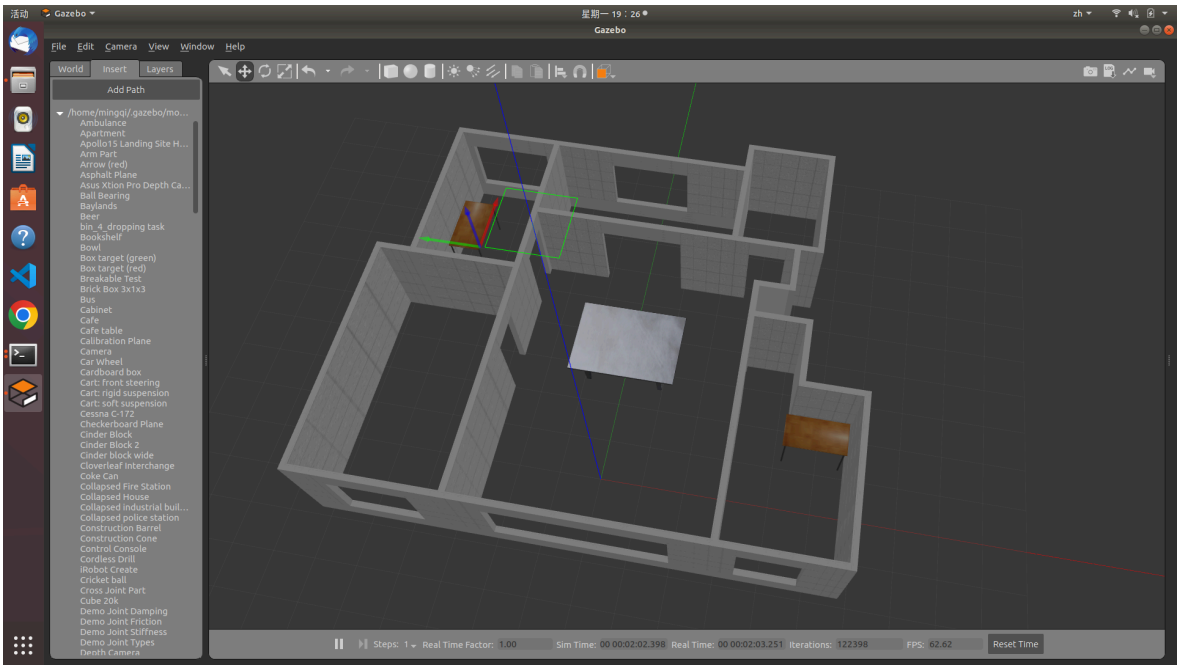
- 放置房间模型

  再次打开gazebo ，左上角insert 选项卡里就会找到自己的模型，找到并选择自己刚才建好的模型放置到右边窗口。



- 放置部分家具

  在insert选项下，选择添加自己想要的家具等模型，来丰富自己的world。可以通过上方按钮，来平移、旋转模型选择合适的摆放位置。



- 点击“File”中“Save world”，保存到 .world文件 下，并命名为Myhouse.world。

六、配置ROS工作工作空间

1.创建ROS工作空间

```
#创建文件夹
mkdir -p ~/roshomework/src
cd roshomework/src
# ROS的工作空间初始化命令
catkin_init_workspace
cd ..
# 编译整个工作空间
catkin_make
#配置文件中加入环境变量
echo "source ~/roshomework/devel/setup.bash" >> ~/.bashrc
```

*可以将**ros-w**的压缩包解压，把所有文件放入**oshomework/src**中。省略以下部分至下一章节*

在**roshomework/src**建立三个文件夹，分别为**launch**、**src**和**worlds**。再创建一个包清单的**package.xml**文件

```
touch ~/roshomework/src/package.xml
mkdir ~/roshomework/src/launch
mkdir ~/roshomework/src/src
mkdir ~/roshomework/src/words
```

2.导入.world文件

- 创建gazebo_world文件夹

```
mkdir ~/roshomework/src/worlds/gazebo_world
```

- 将第四章-6中生成的Myhouse.world放入gazebo_world文件夹中

3.编写launch文件

- 参考turtlebot3_world.launch文件编写
- 新建.launch文件

```
touch ~/roshomework/src/launch/ turtlebot3_world.launch
```

- 将下面程序复制到turtlebot3_world.launch文件中

```
<launch>
  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle, wa
  <arg name="x_pos" default="0.0"/>
  <arg name="y_pos" default="0.0"/>
  <arg name="z_pos" default="0.0"/>

  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find  ros-w)/worlds/gazebo_world/Myhouse.world"/>
    <arg name="paused" value="false"/>
    <arg name="use_sim_time" value="true"/>
    <arg name="gui" value="true"/>
    <arg name="headless" value="false"/>
    <arg name="debug" value="false"/>
  </include>

  <param name="robot_description" command="$(find xacro)/xacro --inorder $(find turtlebot

  <node pkg="gazebo_ros" type="spawn_model" name="spawn_urdf"  args="-urdf -model turtlebo
</launch>
```

4. 编写package.xml文件

- 将下列程序复制到package.xml中

```
<?xml version="1.0"?>
<package format="2">
  <name>ros-w</name>
  <version>0.1.0</version>
  <description>zmq wrh homework</description>

  <maintainer email="zhangmingqi318@gmail.com">zmq wrh</maintainer>

  <license>GPL</license>

  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>rospy</build_depend>
  <build_export_depend>rospy</build_export_depend>
  <exec_depend>rospy</exec_depend>
  <exec_depend>gazebo_ros</exec_depend>

  <export>
       <gazebo_ros gazebo_media_path="${prefix}/worlds"/>
  </export>
</package>
```

5.更改Turtlebot3类型

- 打开.bashrc文件

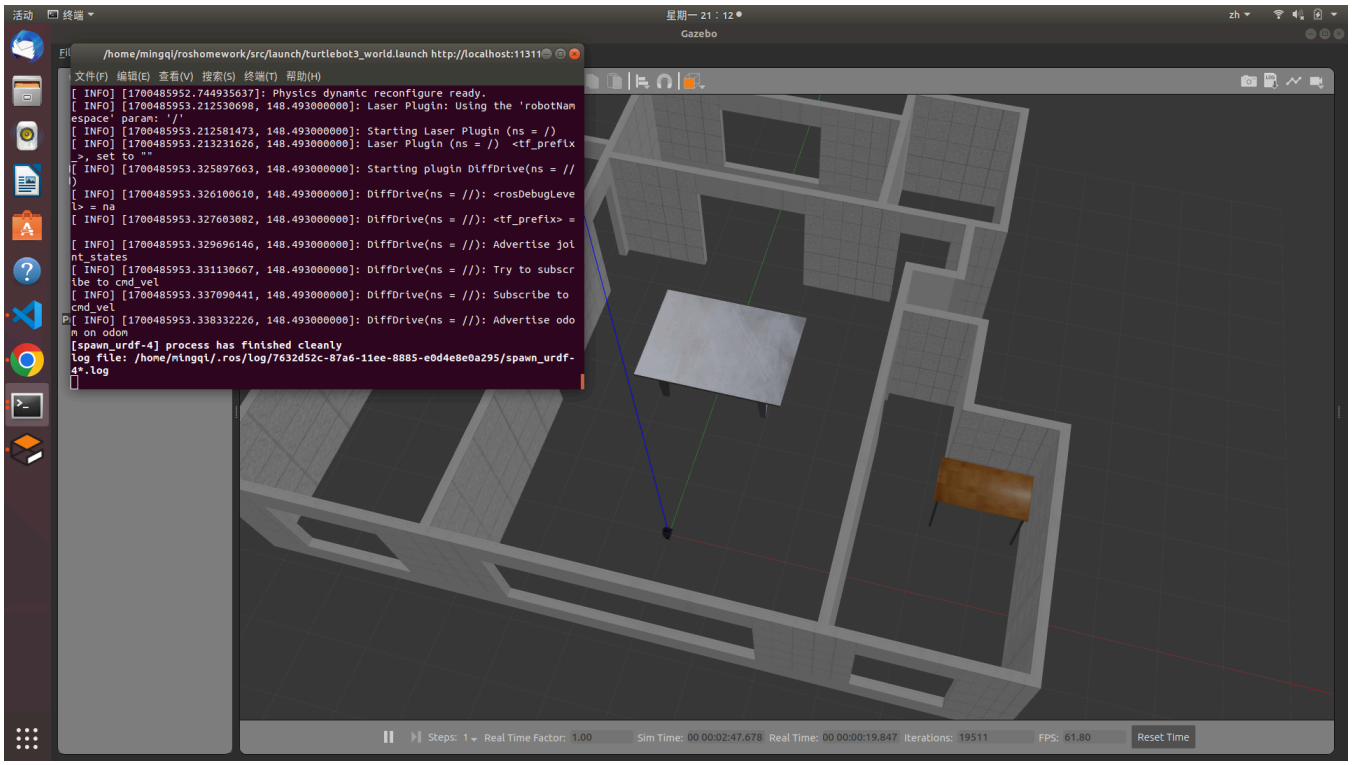
```
gedit ~/.bashrc
```

将下TURTLEBOT3_MODEL=burger改为export TURTLEBOT3_MODEL=waffle

```
#原为burger
export TURTLEBOT3_MODEL=burger
#改为waffle
export TURTLEBOT3_MODEL=waffle
```

6.测试工作空间是否建立

```
roslaunch ros-w turtlebot3_world.launch
```



七、程序编写

1、编写键盘控制程序

- 在 ~/roshomework/src/src中创建控制小车的Python的程序（参考turtlebot3_teleop_key程序）

```
touch ~/roshomework/src/src/rosw_teleop_key.py
```

- 下列代码复制进入.py文件中

```
#!/usr/bin/env python

import rospy
from geometry_msgs.msg import Twist
import sys, select, os
if os.name == 'nt':
 import msvcrt, time
else:
 import tty, termios

BURGER_MAX_LIN_VEL = 0.22
BURGER_MAX_ANG_VEL = 2.84

WAFFLE_MAX_LIN_VEL = 0.26
WAFFLE_MAX_ANG_VEL = 1.82

LIN_VEL_STEP_SIZE = 0.01
ANG_VEL_STEP_SIZE = 0.1

msg = """
Control Your TurtleBot3!
---------------------------
Moving around:
       w
  a    s    d
       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
"""

e = """
Communications Failed
"""
```

```
def getKey():
   if os.name == 'nt':
       timeout = 0.1
       startTime = time.time()
       while(1):
           if msvcrt.kbhit():
               if sys.version_info[0] >= 3:
                   return msvcrt.getch().decode()
               else:
                   return msvcrt.getch()
           elif time.time() - startTime > timeout:
               return ''

   tty.setraw(sys.stdin.fileno())
   rlist, _, _ = select.select([sys.stdin], [], [], 0.1)
   if rlist:
       key = sys.stdin.read(1)
   else:
       key = ''

   termios.tcsetattr(sys.stdin, termios.TCSADRAIN, settings)
   return key

def vels(target_linear_vel, target_angular_vel):
   return "currently:\tlinear vel %s\t angular vel %s " % (target_linear_vel,target_angula

def makeSimpleProfile(output, input, slop):
   if input > output:
       output = min( input, output + slop )
   elif input < output:
       output = max( input, output - slop )
   else:
       output = input

   return output

def constrain(input, low, high):
   if input < low:
     input = low
```

```
    elif input > high:
      input = high
    else:
      input = input

    return input

def checkLinearLimitVelocity(vel):
    if turtlebot3_model == "burger":
      vel = constrain(vel, -BURGER_MAX_LIN_VEL, BURGER_MAX_LIN_VEL)
    elif turtlebot3_model == "waffle" or turtlebot3_model == "waffle_pi":
      vel = constrain(vel, -WAFFLE_MAX_LIN_VEL, WAFFLE_MAX_LIN_VEL)
    else:
      vel = constrain(vel, -BURGER_MAX_LIN_VEL, BURGER_MAX_LIN_VEL)

    return vel

def checkAngularLimitVelocity(vel):
    if turtlebot3_model == "burger":
      vel = constrain(vel, -BURGER_MAX_ANG_VEL, BURGER_MAX_ANG_VEL)
    elif turtlebot3_model == "waffle" or turtlebot3_model == "waffle_pi":
      vel = constrain(vel, -WAFFLE_MAX_ANG_VEL, WAFFLE_MAX_ANG_VEL)
    else:
      vel = constrain(vel, -BURGER_MAX_ANG_VEL, BURGER_MAX_ANG_VEL)

    return vel

if __name__=="__main__":
    if os.name != 'nt':
        settings = termios.tcgetattr(sys.stdin)

    rospy.init_node('turtlebot3_teleop')
    pub = rospy.Publisher('cmd_vel', Twist, queue_size=10)

    turtlebot3_model = rospy.get_param("model", "burger")

    status = 0
    target_linear_vel   = 0.0
    target_angular_vel  = 0.0
```

```
    control_linear_vel  = 0.0
    control_angular_vel = 0.0

    try:
        print(msg)
        while not rospy.is_shutdown():
            key = getKey()
            if key == 'w' :
                target_linear_vel = checkLinearLimitVelocity(target_linear_vel + LIN_VEL_S
                status = status + 1
                print(vels(target_linear_vel,target_angular_vel))
            elif key == 'x' :
                target_linear_vel = checkLinearLimitVelocity(target_linear_vel - LIN_VEL_S
                status = status + 1
                print(vels(target_linear_vel,target_angular_vel))
            elif key == 'a' :
                target_angular_vel = checkAngularLimitVelocity(target_angular_vel + ANG_VE
                status = status + 1
                print(vels(target_linear_vel,target_angular_vel))
            elif key == 'd' :
                target_angular_vel = checkAngularLimitVelocity(target_angular_vel - ANG_VE
                status = status + 1
                print(vels(target_linear_vel,target_angular_vel))
            elif key == ' ' or key == 's' :
                target_linear_vel   = 0.0
                control_linear_vel  = 0.0
                target_angular_vel  = 0.0
                control_angular_vel = 0.0
                print(vels(target_linear_vel, target_angular_vel))
            else:
                if (key == '\x03'):
                    break

            if status == 20 :
                print(msg)
                status = 0

            twist = Twist()
```

```
            control_linear_vel = makeSimpleProfile(control_linear_vel, target_linear_vel,
            twist.linear.x = control_linear_vel; twist.linear.y = 0.0; twist.linear.z = 0.0

            control_angular_vel = makeSimpleProfile(control_angular_vel, target_angular_vel
            twist.angular.x = 0.0; twist.angular.y = 0.0; twist.angular.z = control_angular

            pub.publish(twist)

    except:
        print(e)

    finally:
        twist = Twist()
        twist.linear.x = 0.0; twist.linear.y = 0.0; twist.linear.z = 0.0
        twist.angular.x = 0.0; twist.angular.y = 0.0; twist.angular.z = 0.0
        pub.publish(twist)

    if os.name != 'nt':
        termios.tcsetattr(sys.stdin, termios.TCSADRAIN, settings)
```

为Python赋权，防止后期报错

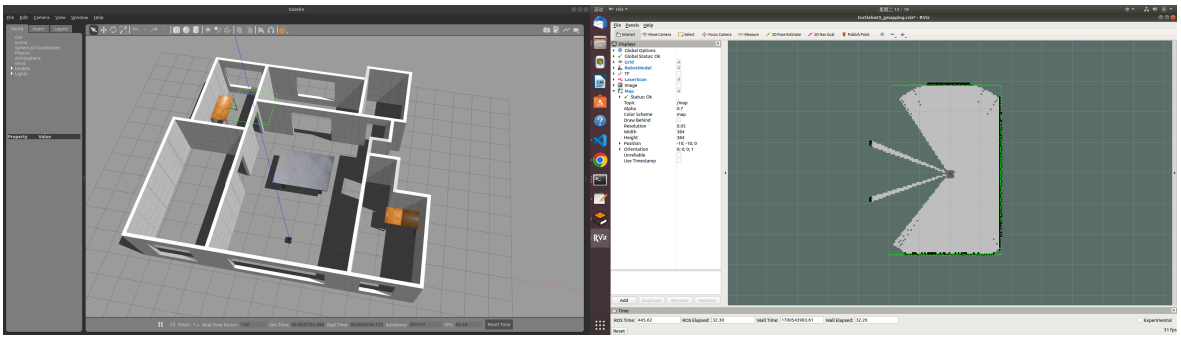
```
chmod 777 ~/roshomework/src/src/rosw_teleop_key.py
```

2、slam建模

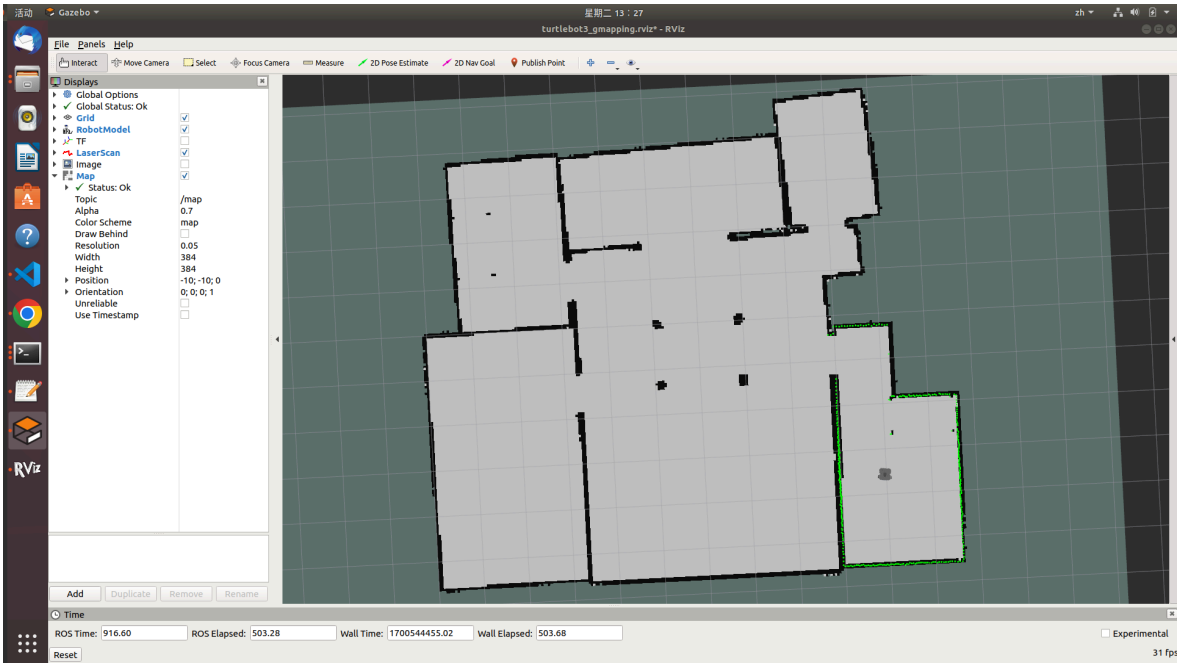
借用turtlebot3_slam程序

```
#打开模型
roslaunch ros-w turtlebot3_world.launch
#新建终端 运行键盘控制程序
rosrun ros-w rosw_teleop_key.py
#新建终端 运行turtlebot3_slam.launch来slam建模
roslaunch turtlebot3_slam turtlebot3_slam.launch
```

- 通过键盘操控小车构建地图，如下图



- 构建完地图



- 启动**map_server**，保存地图（**pgm**和**yaml**）

```
rosrun map_server map_saver -f /home/mingqi/roshomework/src/map/map
```

为地图赋权，防止后期报错

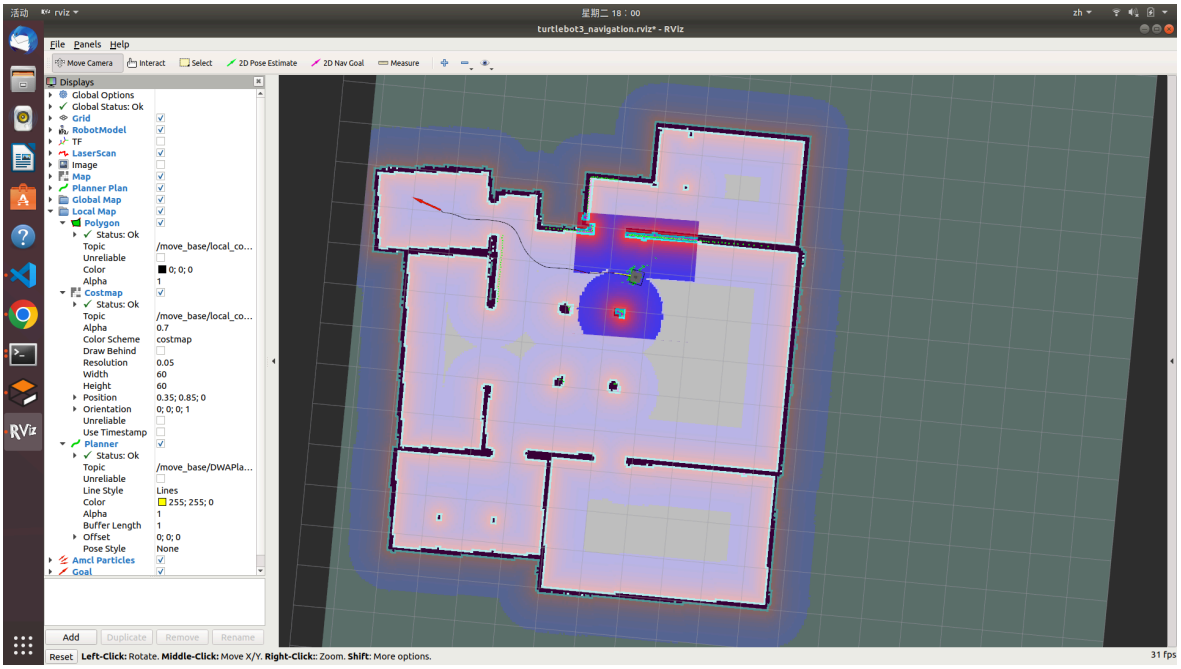
```
chmod 777 ~/roshomework/src/map/map.yaml
```

3、slam自主导航

借用**turtlebot3_slam**程序

```
#加载仿真环境
  roslaunch ros-w turtlebot3_world.launch
#加载导航节点
  roslaunch turtlebot3_navigation turtlebot3_navigation.launch map_file:=/home/mingqi/ros
```

- 配置好rviz,使用2D Pose Estimate设定好小车起始地点，再使用2D Nav Goal设定小车目标地点，小车会规划好路径，自动运行到目标位置，实现自主导航功能。



4、整合launch文件

整合launch文件，一共分为五个：
1.仿真环境（turtlebot3_world.launch）
2.仿真环境+键盘控制（turtlebot3_control.launch）
3.仿真环境+键盘控制+激光rviz（turtlebot3_control_laser.launch）
4.仿真环境+slam建模（turtlebot3_slam.launch）
5.仿真环境+slam导航（turtlebot3_guidance.launch）

- **1.仿真环境（turtlebot3_world.launch）**
  上文已经配置好

```
turtlebot3_world.launch
```

节点图如下：

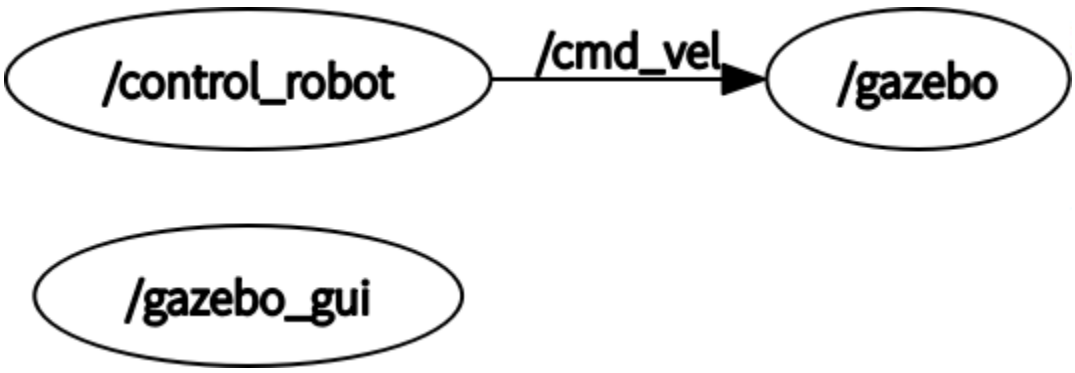
- **2.仿真环境+键盘控制（turtlebot3_control.launch）**
  加入键盘控制节点

```
#新建launch文件
touch ~/roshomework/src/launch/turtlebot3_control.launch
#授权
chmod 777 ~/roshomework/src/launch/turtlebot3_control.launch
#文本编辑器打开
gedit ~/roshomework/src/launch/turtlebot3_control.launch
```

将下面程序复制到**turtlebot3_control.launch**文件中

```
<launch>
  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle, wa
  <arg name="x_pos" default="0.0"/>
  <arg name="y_pos" default="0.0"/>
  <arg name="z_pos" default="0.0"/>

  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find  ros-w)/worlds/gazebo_world/Myhouse.world"/>
    <arg name="paused" value="false"/>
    <arg name="use_sim_time" value="true"/>
    <arg name="gui" value="true"/>
    <arg name="headless" value="false"/>
    <arg name="debug" value="false"/>
  </include>

  <param name="robot_description" command="$(find xacro)/xacro --inorder $(find turtlebot

  <node pkg="gazebo_ros" type="spawn_model" name="spawn_urdf"  args="-urdf -model turtlebo
 <node pkg="ros-w" type="rosw_teleop_key.py" name="control_robot" output="screen"  />
</launch>
```

节点图如下：



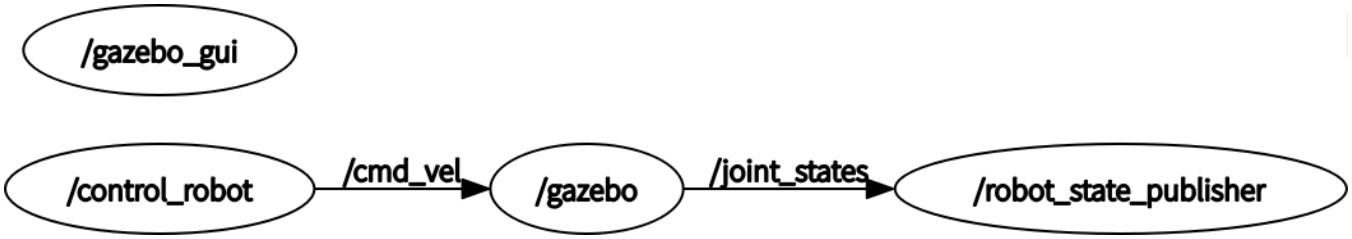
- **3.仿真环境+键盘控制+激光rviz（turtlebot3_control_laser.launch）**
  加入键盘控制节点

```
#新建launch文件
touch ~/roshomework/src/launch/turtlebot3_control_laser.launch
#授权
chmod 777  ~/roshomework/src/launch/turtlebot3_control_laser.launch
#文本编辑器打开
gedit ~/roshomework/src/launch/turtlebot3_control_laser.launch
```

将下面程序复制到**turtlebot3_control_laser.launch**文件中

```
<launch>
  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle, wa
  <arg name="x_pos" default="0.0"/>
  <arg name="y_pos" default="0.0"/>
  <arg name="z_pos" default="0.0"/>

  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find  ros-w)/worlds/gazebo_world/Myhouse.world"/>
    <arg name="paused" value="false"/>
    <arg name="use_sim_time" value="true"/>
    <arg name="gui" value="true"/>
    <arg name="headless" value="false"/>
    <arg name="debug" value="false"/>
  </include>

  <include file="$(find turtlebot3_gazebo)/launch/turtlebot3_gazebo_rviz.launch">
  </include>

  <param name="robot_description" command="$(find xacro)/xacro --inorder $(find turtlebot

  <node pkg="gazebo_ros" type="spawn_model" name="spawn_urdf"  args="-urdf -model turtlebo
 <node pkg="ros-w" type="rosw_teleop_key.py" name="control_robot" output="screen"  />
</launch>
```

节点图如下：



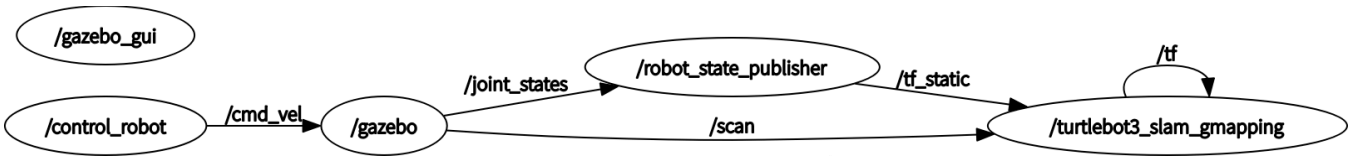
- **4.仿真环境+slam建模（turtlebot3_slam.launch）**
  加入键盘控制节点

```
#新建launch文件
touch ~/roshomework/src/launch/turtlebot3_slam.launch
#授权
chmod 777 ~/roshomework/src/launch/turtlebot3_slam.launch
#文本编辑器打开
gedit ~/roshomework/src/launch/turtlebot3_slam.launch
```

将下面程序复制到**turtlebot3_slam.launch**文件中

```
<launch>
  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle, wa
  <arg name="x_pos" default="0.0"/>
  <arg name="y_pos" default="0.0"/>
  <arg name="z_pos" default="0.0"/>

  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find  ros-w)/worlds/gazebo_world/Myhouse.world"/>
    <arg name="paused" value="false"/>
    <arg name="use_sim_time" value="true"/>
    <arg name="gui" value="true"/>
    <arg name="headless" value="false"/>
    <arg name="debug" value="false"/>
  </include>
  <include file="$(find turtlebot3_slam)/launch/turtlebot3_slam.launch">
  </include>

  <param name="robot_description" command="$(find xacro)/xacro --inorder $(find turtlebot

  <node pkg="gazebo_ros" type="spawn_model" name="spawn_urdf"  args="-urdf -model turtlebo
 <node pkg="ros-w" type="rosw_teleop_key.py" name="control_robot" output="screen"  />
</launch>
```

节点图如下：



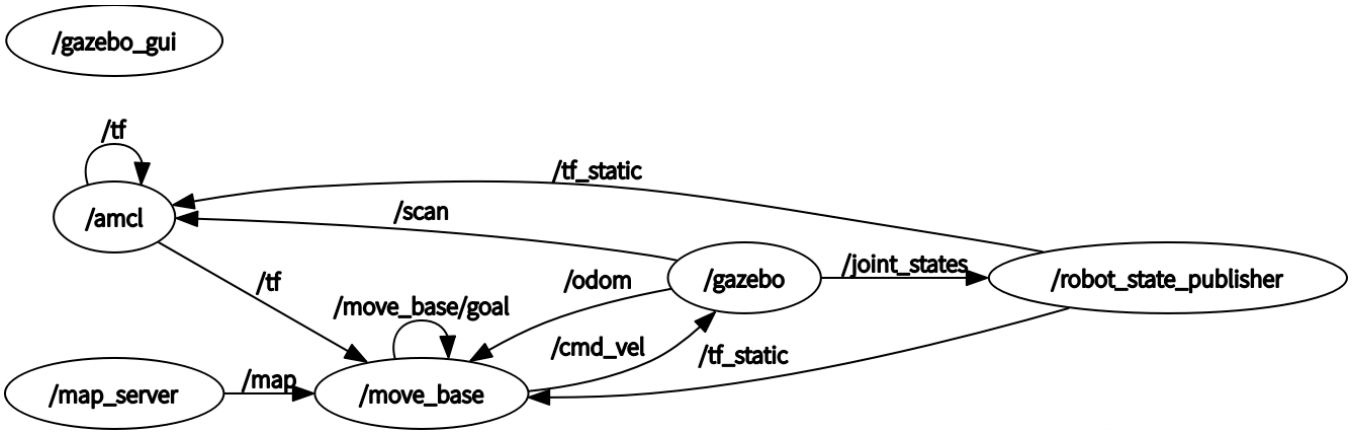
- **5.仿真环境+slam导航（turtlebot3_guidance.launch）**
  加入键盘控制节点

```
#新建launch文件
touch ~/roshomework/src/launch/turtlebot3_guidance.launch
#授权
chmod 777 ~/roshomework/src/launch/turtlebot3_guidance.launch
#文本编辑器打开
gedit ~/roshomework/src/launch/turtlebot3_guidance.launch
```

将下面程序复制到**turtlebot3_guidance.launch**文件中

```
<launch>
  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle, wa
  <arg name="x_pos" default="0.0"/>
  <arg name="y_pos" default="0.0"/>
  <arg name="z_pos" default="0.0"/>

  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find  ros-w)/worlds/gazebo_world/Myhouse.world"/>
    <arg name="paused" value="false"/>
    <arg name="use_sim_time" value="true"/>
    <arg name="gui" value="true"/>
    <arg name="headless" value="false"/>
    <arg name="debug" value="false"/>
  </include>

  <include file="$(find turtlebot3_navigation)/launch/turtlebot3_navigation.launch">
    <arg name="map_file" value="$(find  ros-w)/map/map.yaml"/>
  </include>

  <param name="robot_description" command="$(find xacro)/xacro --inorder $(find turtlebot3

  <node pkg="gazebo_ros" type="spawn_model" name="spawn_urdf"  args="-urdf -model turtlebo
</launch>
```

节点图如下：



八、简易安装OR测试

*所有代码都可以从我的**Github**中下载：**https://github.com/zhuanshunjishi/ROS-work***
*或在在**Gitee**中下载：**https://gitee.com/MingQi_Ya/ROS-work***

1.安装Turtlebot3

- 因为在后来用Turtlebot3作为载体。
  按照第三章安装**Turtlebot3**

2.安装gazebo模型库

- 因为在后来的仿真模型（Myhouse.world）中用到了gazebo模型库的模型。

```
 # 进入.gazebo文件夹，用于存放Gazebo模型和相关配置文件
cd ~/.gazebo/
# 安装Git工具
sudo apt install git
# 从https://gitee.com/dva7777/gazebo_models.git克隆Gazebo模型
git clone https://gitee.com/dva7777/gazebo_models.git
# 将克隆的gazebo_models文件夹重命名为models，并放到.gazebo文件夹下
mv gazebo_models/ ./models
```

3.安装功能包

- 在后期程序中需要gmapping包和dwa_local_planner包，现在安装防止后期报错。

```
sudo apt-get install ros-melodic-gmapping
sudo apt-get install ros-melodic-dwa-local-planner
```

4.创建ROS工作空间

```
#创建文件夹
mkdir -p ~/roshomework/src
cd roshomework/src
# ROS的工作空间初始化命令
catkin_init_workspace
cd ..
# 编译整个工作空间
catkin_make
#配置文件中加入环境变量
echo "source ~/roshomework/devel/setup.bash" >> ~/.bashrc
```

5.下载github文件

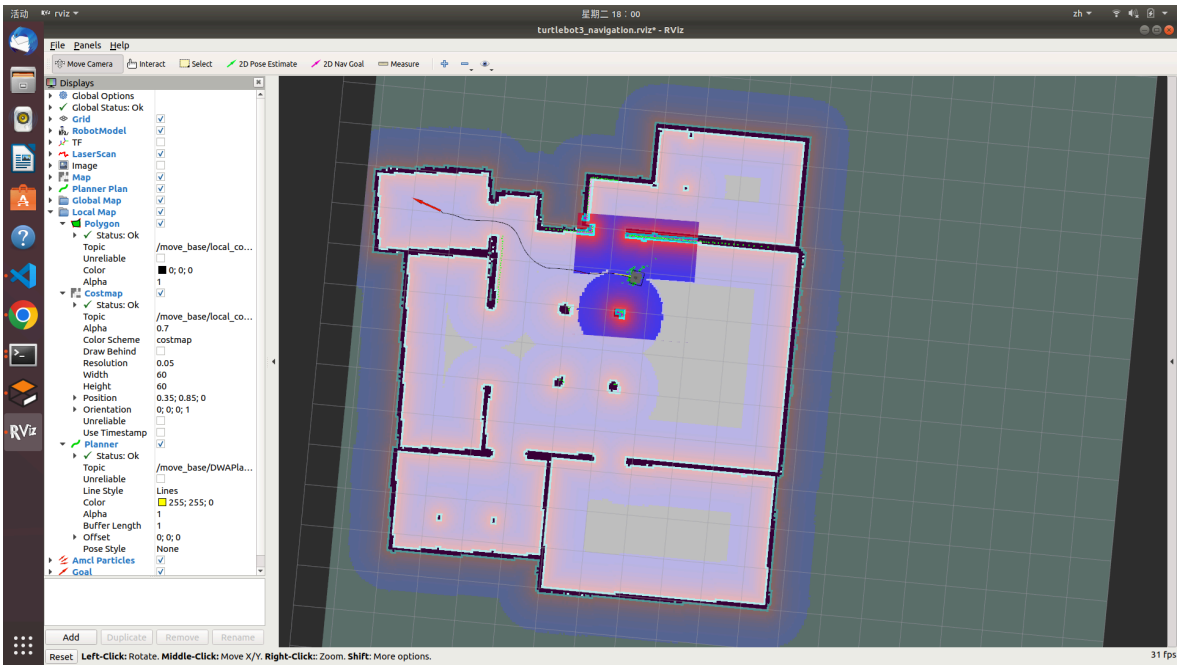
```
#创建文件夹
cd ~/roshomework/src
#git下载文件
 git clone https://github.com/zhuanshunjishi/ROS-work.git
 #如果上面现在不了，可以用gitee上面的
 git clone https://gitee.com/MingQi_Ya/ROS-work.git
 #移动文件
 mv ~/roshomework/src/ROS-work/* ~/roshomework/src
 sudo rm -r ~/roshomework/src/ROS-work
 #更新环境变量
 source ./.bashrc
```

6、slam自主导航

已经保存好map文件，不需要从新slam建立map文件

```
#运行slam导航launch
  roslaunch ros-w turtlebot3_guidance.launch
```

- 配置好rviz,使用2D Pose Estimate设定好小车起始地点，再使用2D Nav Goal设定小车目标地点，小车会规划好路径，自动运行到目标位置，实现自主导航功能。



参考连接

- Ubuntu18.04安装ROS Melodic（详细，亲测安装完成，有清晰的截图步骤）
- Cartographer ROS
- gazebo仿真环境搭建+配置+小车运动仿真