实验6：ROS Gazebo Turtlebot3

姓名：周岱 学号：23049200610

一、理论知识

1. 简要说明ROS中的Gazebo是如何工作以及它在机器人系统中的重要性。

**工作原理：**

（1）仿真运行机制：

Gazebo 作为独立的物理仿真引擎，通过加载 URDF/Xacro 格式的机器人模型和 SDF 格式的环境模型，构建虚拟 3D 场景。其内部物理引擎（如 ODE、Bullet）计算机器人与环境的交互（碰撞、摩擦、重力等），实时更新机器人运动状态和传感器数据，并通过 gazebo\_ros 插件将这些信息转换为 ROS 话题（如 /odom 里程计、/scan 激光数据）发布。同时，Gazebo 接收 ROS 节点发布的控制指令（如 /cmd\_vel），驱动虚拟机器人执行运动，形成闭环控制。

（2）与 ROS 的通信机制：

通过 gazebo\_ros 功能包提供的接口，Gazebo 与 ROS 系统实现双向数据交互：一方面，Gazebo 将仿真中的机器人状态、传感器数据以 ROS 话题形式输出；另一方面，ROS 节点发布的控制指令、模型 spawn 请求等通过服务（Service）或话题传递给 Gazebo，实现对仿真环境的动态控制（如加载模型、修改物理参数）。

（3）多机器人与场景管理机制：

支持在同一仿真环境中加载多个机器人模型，通过 ROS 命名空间（Namespace）隔离不同机器人的话题和坐标变换（TF），避免冲突。同时，可通过 SDF 文件定义复杂环境（如障碍物、光照、地形），或通过 ROS 服务动态修改场景元素，模拟多样化的真实世界场景。

**在机器人系统中的重要性**：

（1）低成本开发验证：

无需依赖物理硬件，即可在虚拟环境中测试机器人算法（如导航、避障、SLAM），避免硬件损坏风险，降低开发成本，尤其适合初期原型设计和算法验证。

（2）高效场景复现：

可精确复现任意场景（如特定障碍物布局、传感器噪声环境），通过调整参数（如机器人初始位置、环境光照）重复测试，快速迭代优化算法，缩短调试周期。

（3）复杂功能测试：

支持模拟极端或危险场景（如碰撞、传感器失效），以及多机器人协作、动态环境交互等复杂任务，验证系统在边界条件下的稳定性和可靠性。

（4）无缝衔接 ROS 生态：

与 ROS 核心功能包（如 navigation、gmapping）完全兼容，仿真中验证的算法可直接迁移到真实机器人，减少 “仿真 - 实物” 差距，加速从虚拟开发到实际部署的流程。

（5）教学与培训支持：

提供可视化的虚拟环境，便于初学者理解机器人工作原理，或用于培训操作人员在安全环境中模拟复杂任务，降低实操门槛。

2. 以turtlebot3\_gazebo/models/turtlebot3\_burger/model.sdf文件为例，简要说明其中所包含的标签都有哪些，分别表示什么意思。

**核心标签及含义：**

（1）<model> 标签：

整个机器人模型的根标签，包含 name 属性（如 name="turtlebot3\_burger"），用于标识模型名称，在 Gazebo 中唯一区分不同模型。

（2）<link> 标签：

定义机器人的单个刚体部件（如底座、轮子、传感器），包含：

name：部件名称（如 base\_link、wheel\_left\_link）；

<pose>：该部件在父坐标系中的位置（x,y,z）和姿态（滚转，俯仰，偏航）；

<inertial>：定义质量、转动惯量等物理属性，用于物理引擎计算运动；

<collision>：定义碰撞检测边界（通常为简化的几何形状，如 <box>、<cylinder>）；

<visual>：定义可视化外观（如 <mesh> 引用 3D 模型文件，<material> 设置颜色 / 纹理）。

（3）<joint> 标签：

定义两个 <link> 之间的连接关系，包含：

name：关节名称（如 wheel\_left\_joint）；

type：关节类型（如 revolute 旋转关节、fixed 固定关节）；

<parent>/<child>：指定关节连接的父部件和子部件（如父为 base\_link，子为 wheel\_left\_link）；

<limit>：定义关节运动范围（如旋转角度 lower/upper、速度 velocity 限制）。

（4）<plugin> 标签：

加载 Gazebo 插件以扩展功能，TurtleBot3 中常用：

libgazebo\_ros\_diff\_drive.so：差速驱动插件，接收 /cmd\_vel 话题控制轮子转动发布 /odom 里程计数据；

libgazebo\_ros\_imu\_sensor.so：IMU 传感器插件，发布 /imu 话题（加速度、角速度数据）；

libgazebo\_ros\_laser.so：激光雷达插件，发布 /scan 话题（激光扫描数据）。

（5）<static> 标签：

布尔值（true/false），标识模型是否为静态（如 false 表示机器人为动态模型，可受物理力运动）。

（6）<self\_collide> 标签：

布尔值，控制模型自身部件是否发生碰撞检测（如 false 避免轮子与底座之间的碰撞计算）。

3. 以turtlebot3\_gazebo/worlds/ turtlebot3\_world.world文件为例，简要说明其中所包含的标签都有哪些，分别表示什么意思。

**核心标签及含义：**

（1）<sdf> 标签：

文件根标签，包含 version 属性（如 version="1.6"），指定 SDF 格式版本，确保 Gazebo 正确解析文件。

（2）<world> 标签：

定义整个仿真世界的根标签，name 属性（如 name="turtlebot3\_world"）标识世界名称，包含所有环境元素和物理规则。

（3）<include> 标签：

用于引入外部模型（如障碍物、地面、光源），通过 <uri> 指定模型路径（如 model://ground\_plane 引入默认地面，model://sun 引入光源），<name> 可重命名引入的模型。

（4）<model> 标签：

定义世界中的静态或动态模型（如障碍物、家具），包含：

name：模型唯一标识（如 name="box\_1" 表示一个立方体障碍物）；

<static>：设为 true 表示模型固定不动（如墙壁、桌子）；

<link>/<joint>：与机器人模型中的标签功能一致，定义模型的物理结构（如立方体的尺寸、质量）。

（5）<physics> 标签：

配置仿真世界的物理规则，包含：

name：物理引擎配置名称（如 name="default\_physics"）；

type：指定物理引擎（如 ode、bullet）；

<max\_step\_size>：物理计算的时间步长（越小精度越高，如 0.001 秒）；

<real\_time\_factor>：仿真时间与真实时间的比例（1.0 表示实时仿真）；

<gravity>：世界重力加速度（如 0 0 -9.8 表示沿 Z 轴负方向）。

（6）<light> 标签：

定义世界中的光源，包含：

name：光源名称（如 name="sun"）；

type：光源类型（如 directional 平行光、point 点光源）；

<pose>：光源位置和照射方向；

<diffuse>/<specular>：光源的漫反射和镜面反射颜色（RGB 值）。

（7）<scene> 标签：

配置仿真场景的可视化属性，如 <ambient> 环境光颜色（如 0.4 0.4 0.4 1）、<shadows> 是否开启阴影（true/false）。

4. 以turtlebot3\_gazebo/ launch /turtlebot3\_world.launch文件为例，简要说明其实现的功能逻辑。

**功能逻辑：**

（1）启动 Gazebo 仿真环境：

通过 <include> 标签引入 gazebo\_ros 包的 empty\_world.launch，初始化 Gazebo 物理引擎和 3D 可视化界面，加载指定的世界模型（如 turtlebot3\_world.world），包含地面、光源、障碍物等环境元素。

（2）加载机器人模型描述：

通过 <param> 标签加载 TurtleBot3 的 URDF/Xacro 模型文件（如 turtlebot3\_burger.urdf.xacro），将机器人结构（关节、连杆、传感器）以 robot\_description 参数形式发布到 ROS 参数服务器，供后续节点解析。

（3）启动机器人状态发布节点：

启动 joint\_state\_publisher\_gui 节点，用于发布机器人关节状态（支持手动调整关节角度，仅仿真时使用）；

启动 robot\_state\_publisher 节点，根据 robot\_description 和关节状态，计算并发布机器人各部件的坐标变换（TF），为导航、感知提供坐标参考。

（4）在 Gazebo 中生成机器人：

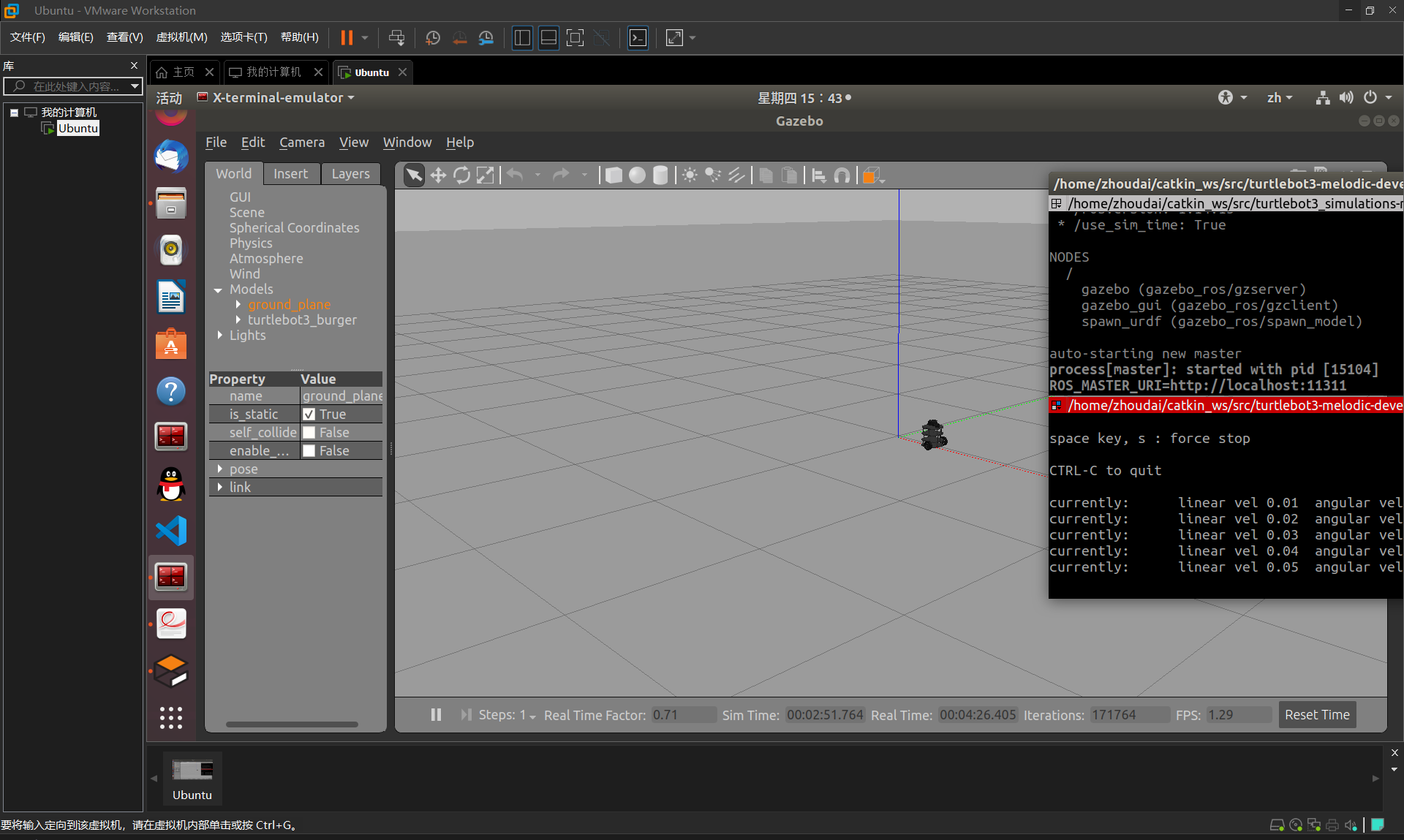
通过 spawn\_model 节点（来自 gazebo\_ros 包），从参数服务器读取 robot\_description，在 Gazebo 仿真环境中生成 TurtleBot3 实体模型，并设置初始位置（如 x=0, y=0, z=0）。

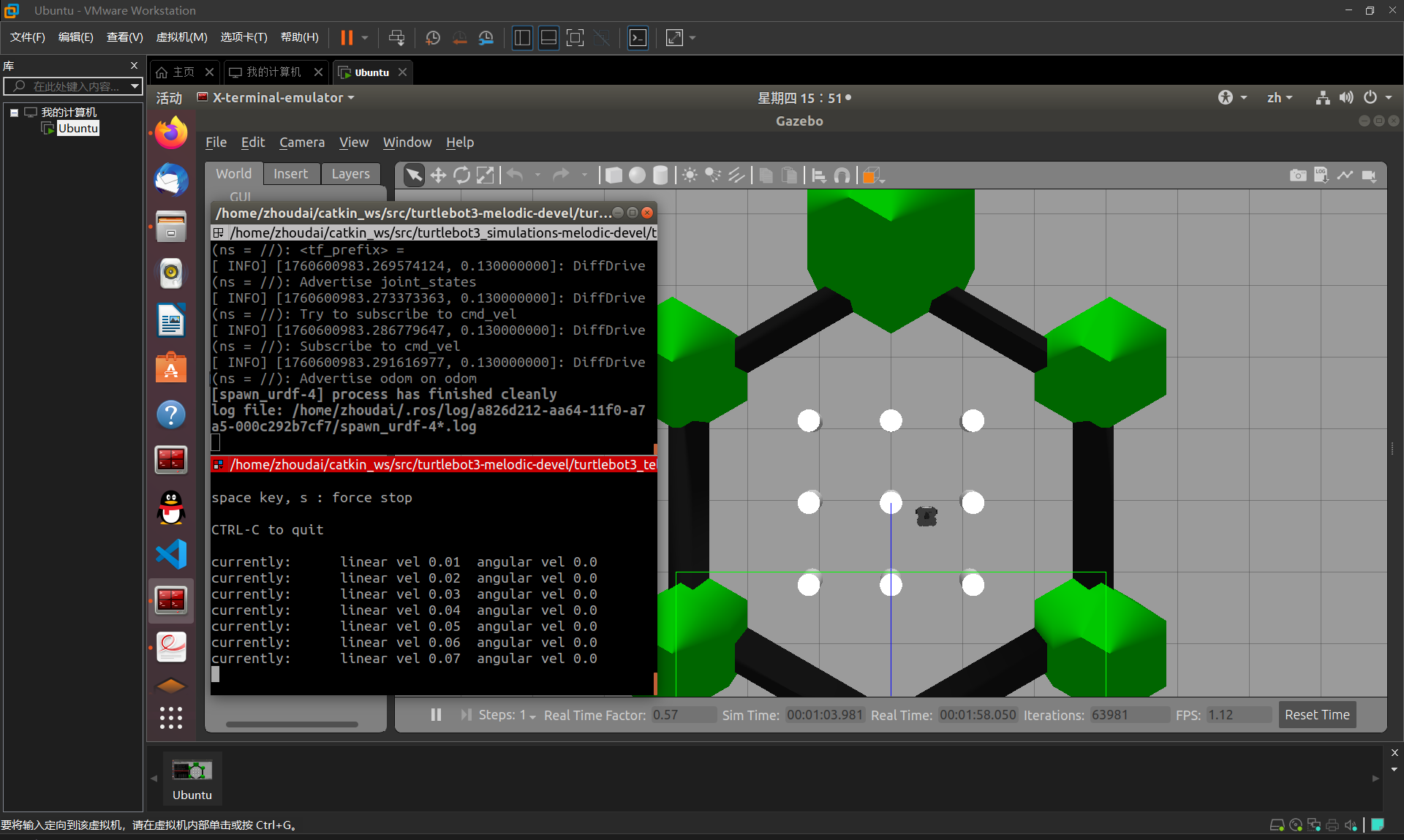
（5）配置机器人与 ROS 的交互：

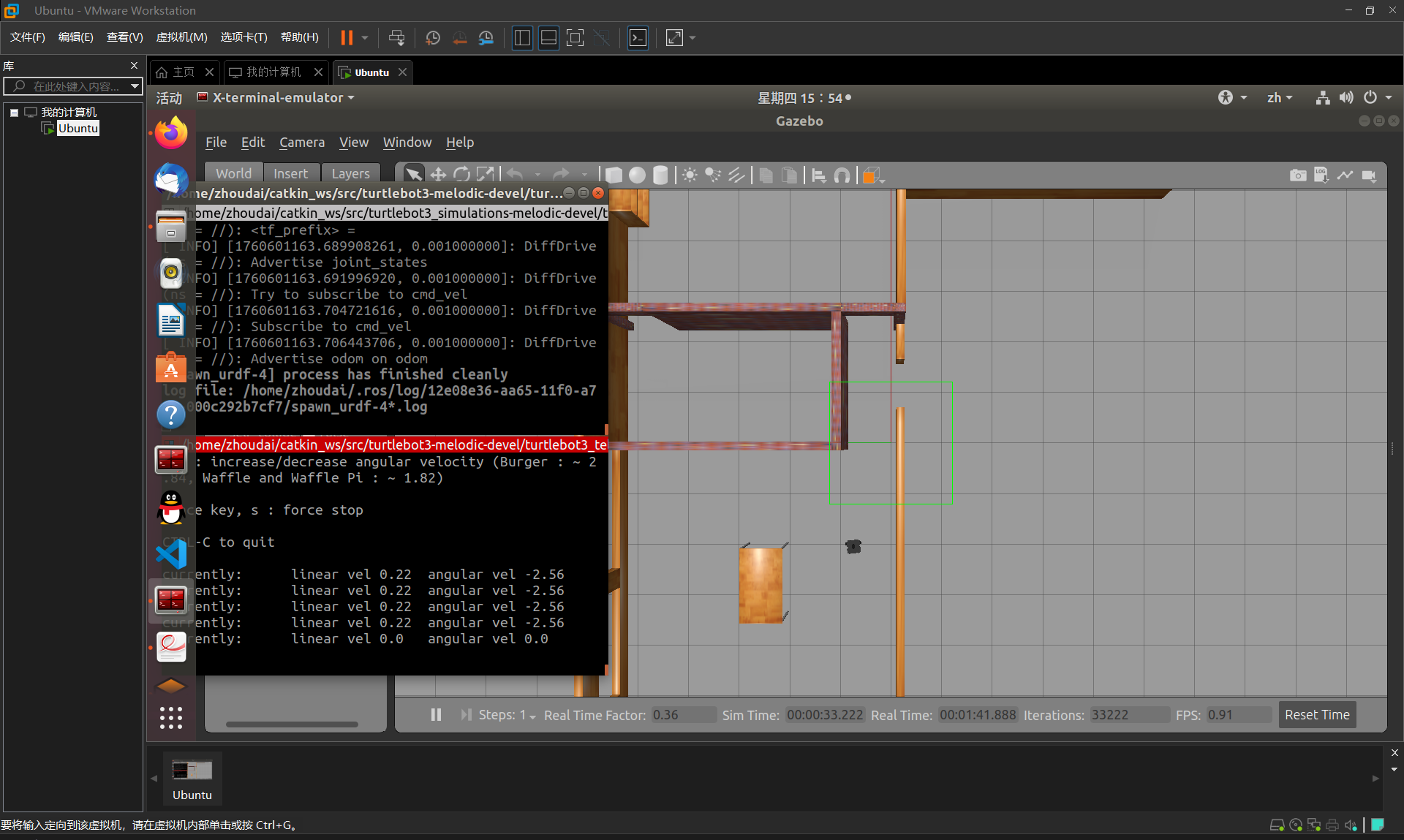
通过 Gazebo 插件（如差速驱动插件、激光雷达插件），将仿真中机器人的传感器数据（如 /scan 激光数据、/odom 里程计）发布为 ROS 话题，并接收 ROS 控制指令（如 /cmd\_vel）驱动机器人运动，实现 “感知 - 控制” 闭环。

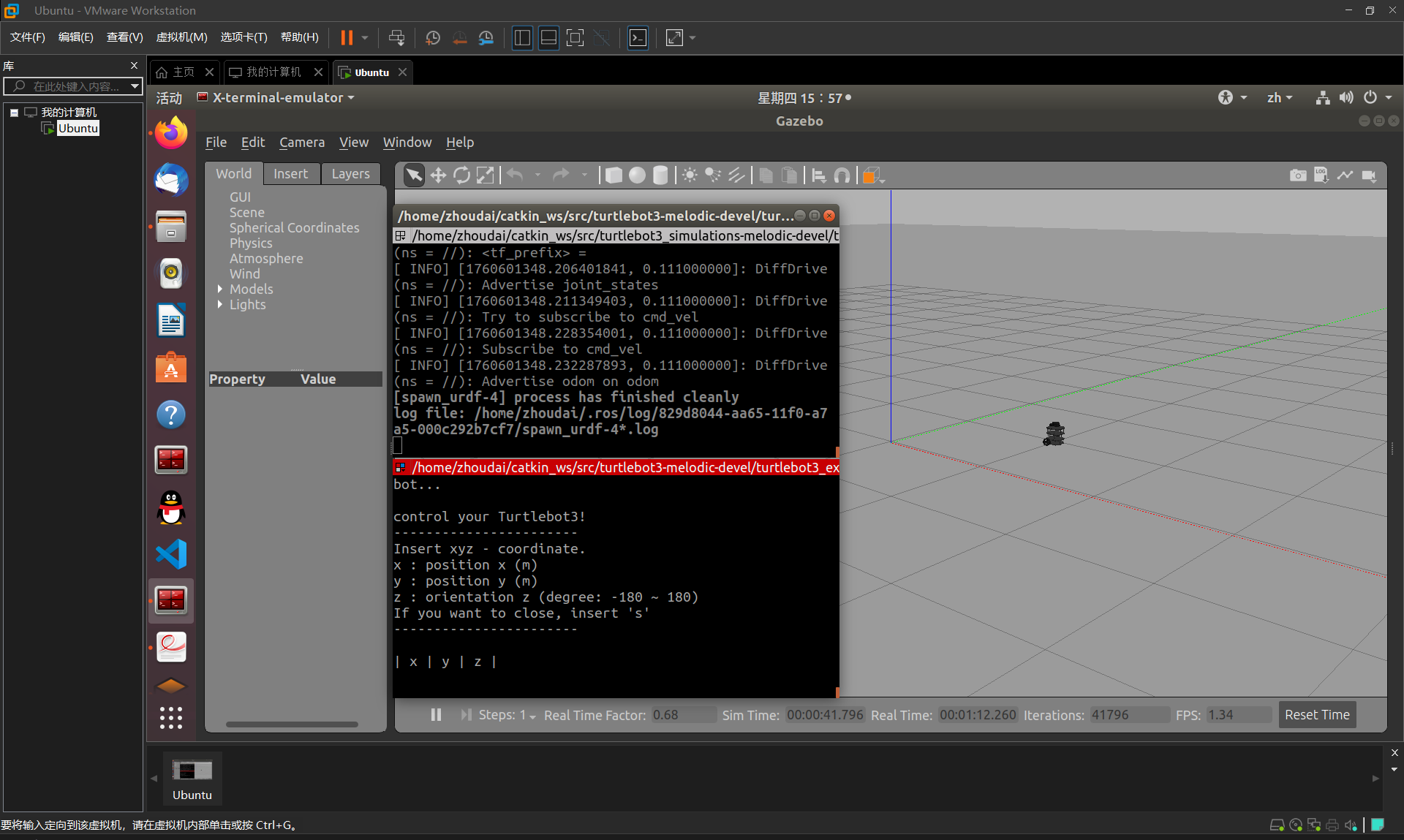
二、实践项目：要求记录关键步骤、遇到的问题及解决办法、成功实现的截图

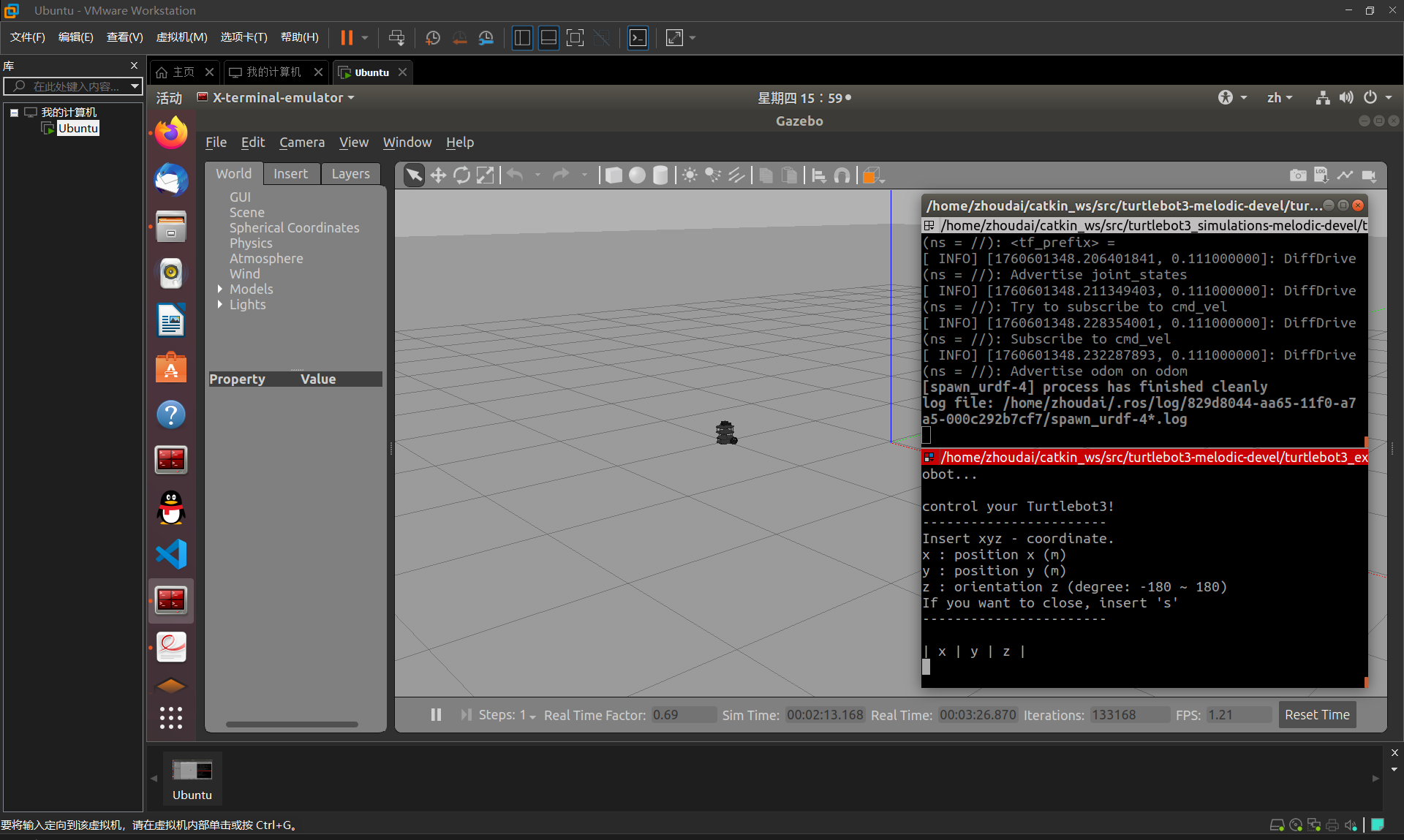
1. 成功运行Turtlebot3官方文档Gazebo Simulation（[TurtleBot3](https://emanual.robotis.com/docs/en/platform/turtlebot3/simulation/#gazebo-simulation)）示例代码并截图

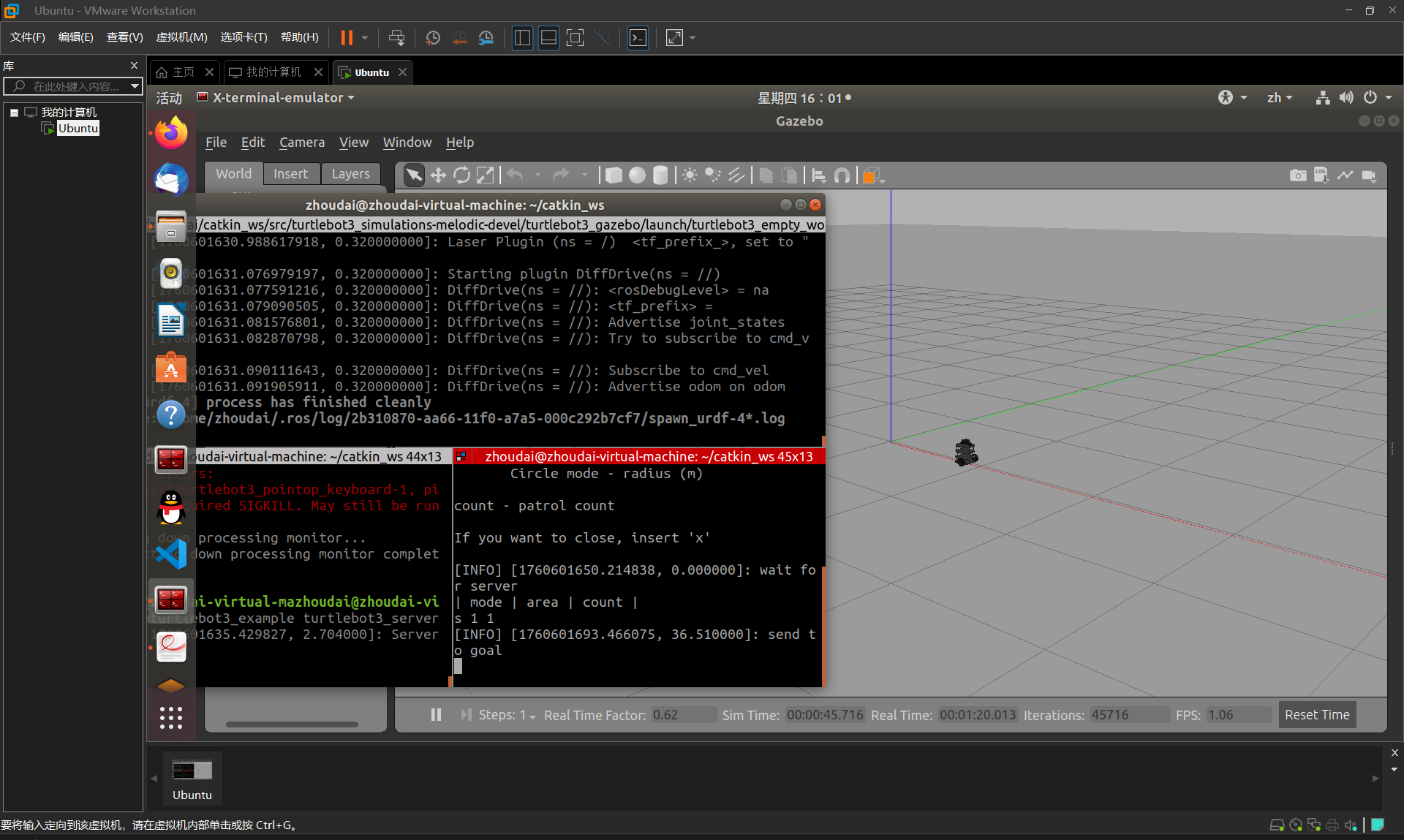




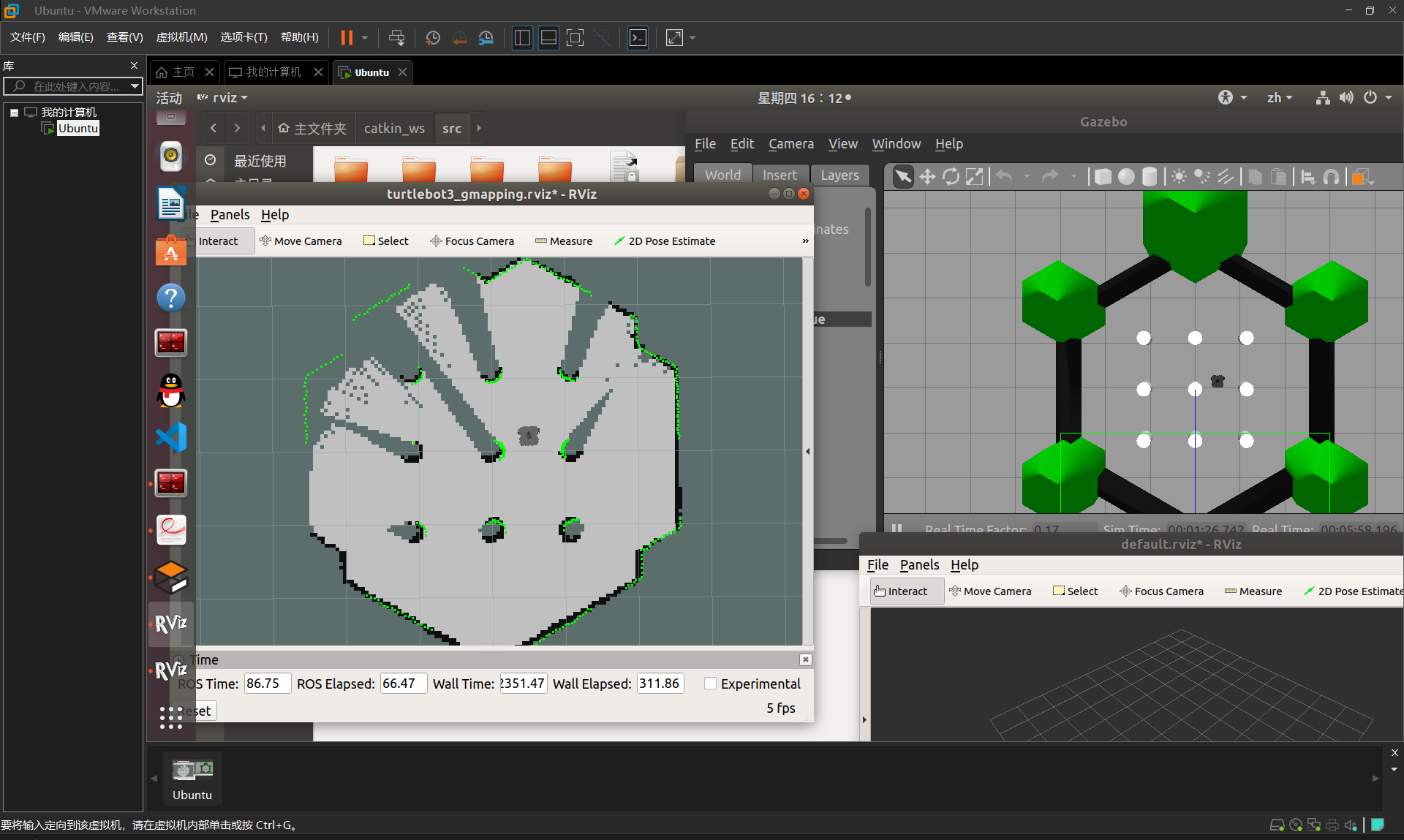


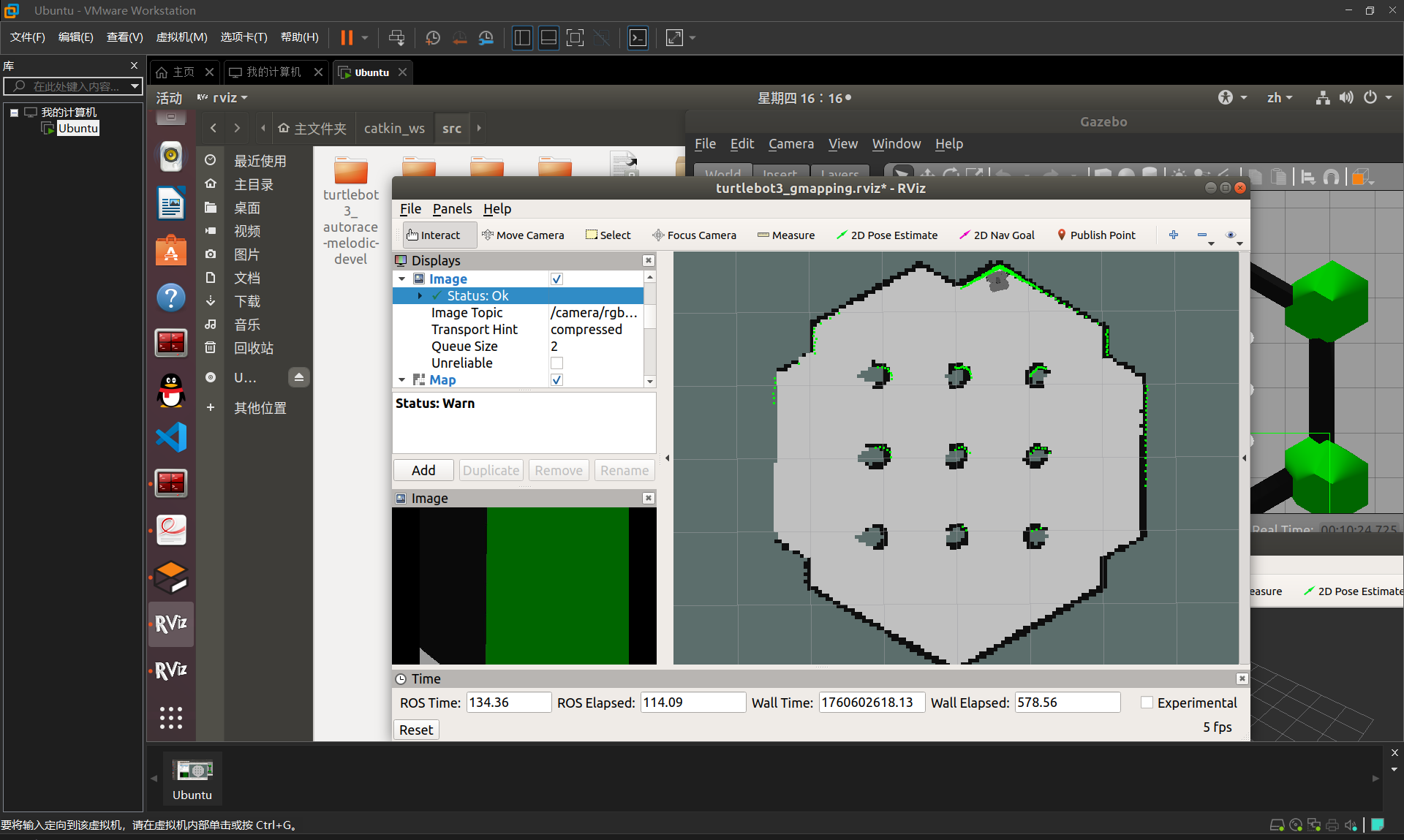


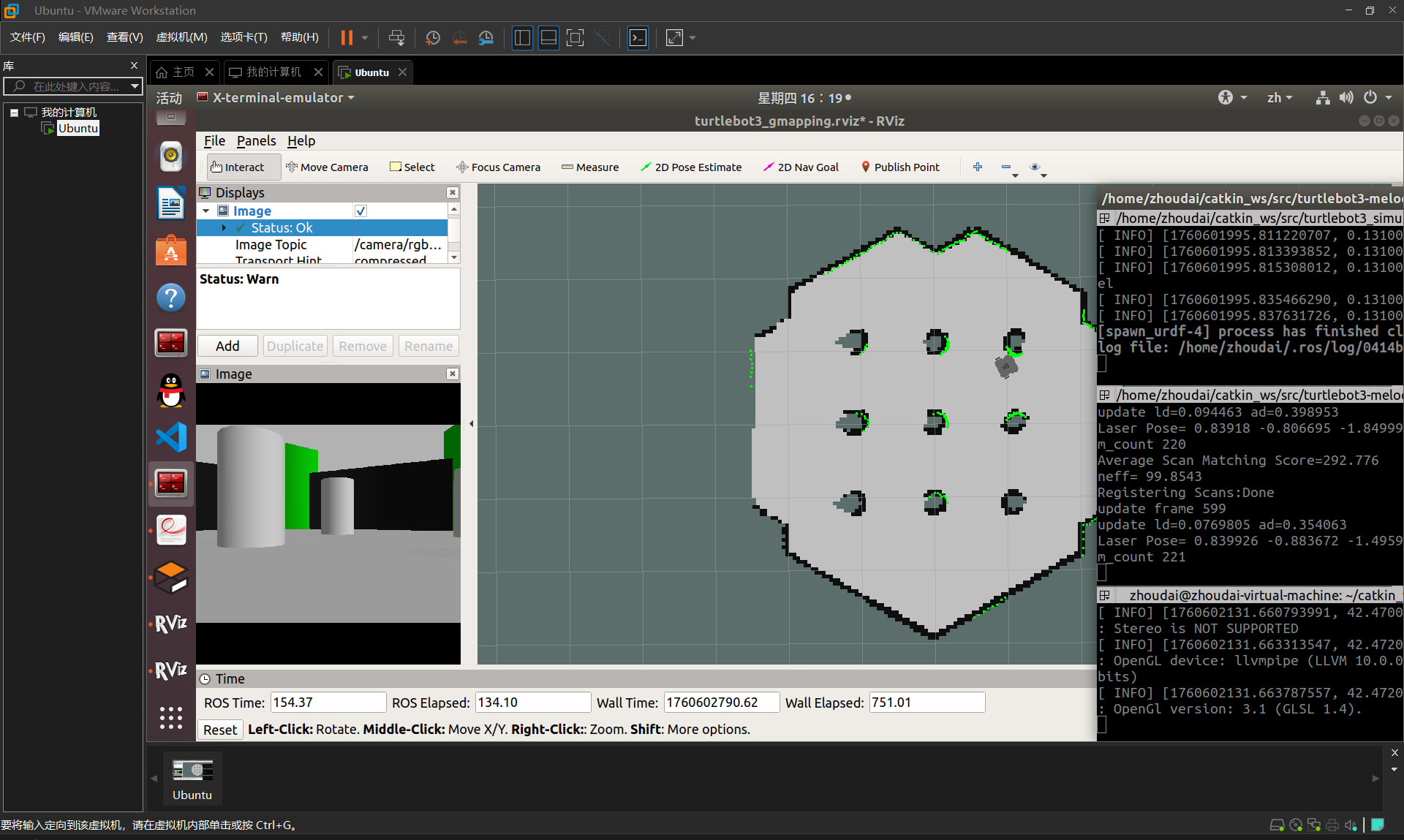




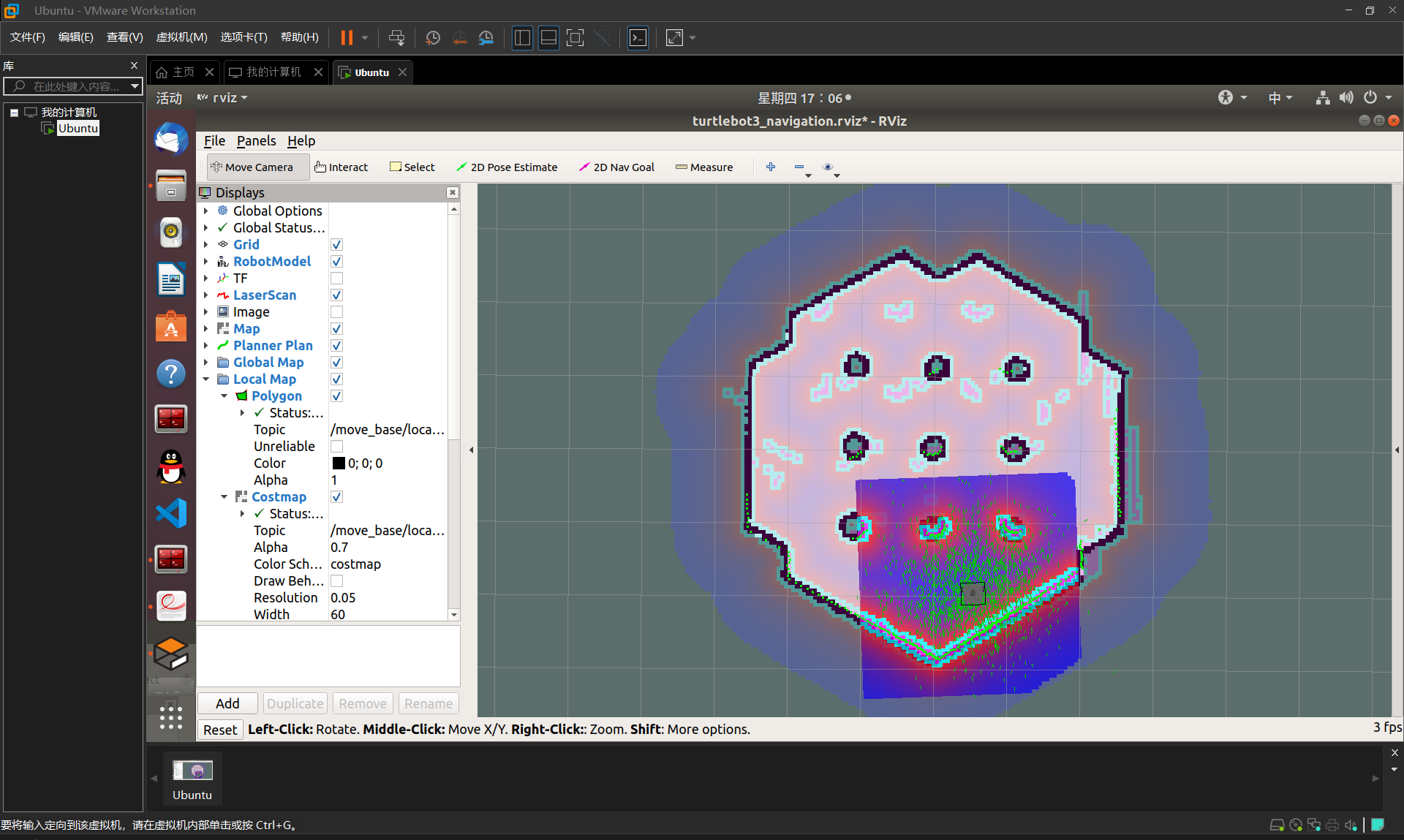
2. 成功运行Turtlebot3官方文档SLAM Simulation（[TurtleBot3](https://emanual.robotis.com/docs/en/platform/turtlebot3/slam_simulation/)）示例代码并截图

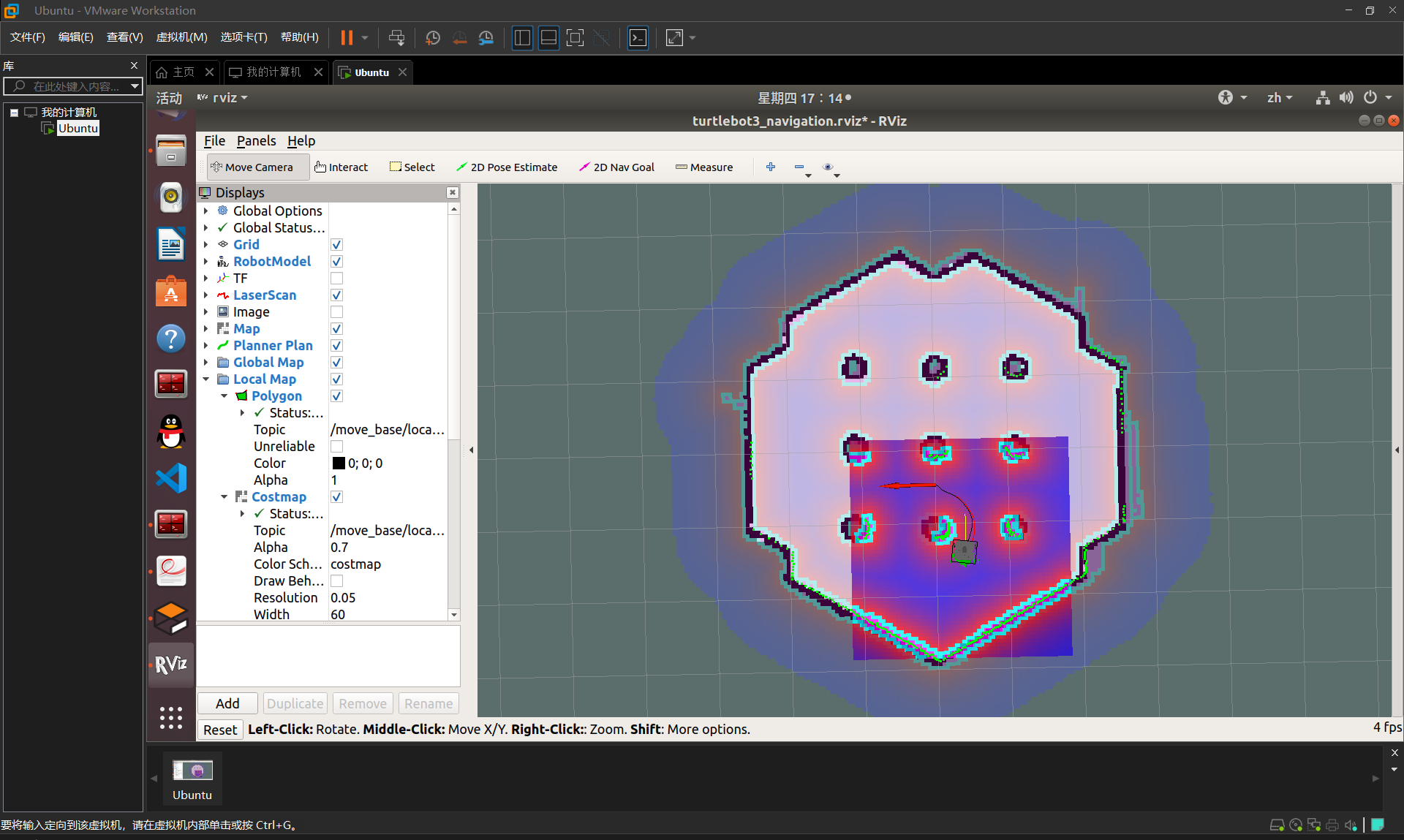




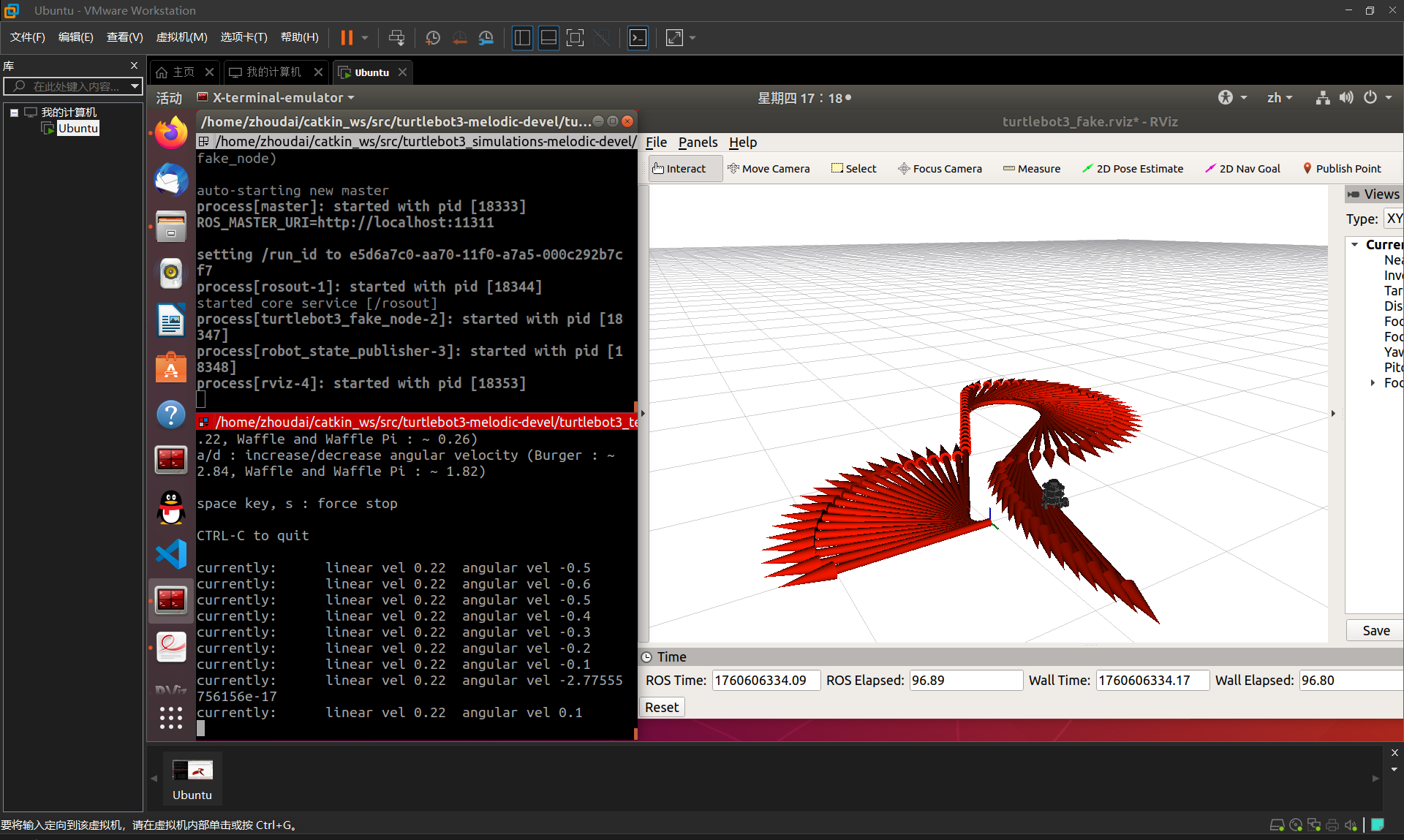


3. 成功运行Turtlebot3官方文档Navigation Simulation（[TurtleBot3](https://emanual.robotis.com/docs/en/platform/turtlebot3/nav_simulation/)）示例代码并截图





4. 成功运行Turtlebot3官方文档Fake node Simulation（[TurtleBot3](https://emanual.robotis.com/docs/en/platform/turtlebot3/fakenode_simulation/)）示例代码并截图



5. 以Turtlebot3为基础，对示例代码进行修改（参考更多的launch文件），在Gazebo仿真环境中实现自定义任务的仿真。

**任务描述：**

任务是在 ROS Melodic 环境中，基于 TurtleBot3 机器人实现一个 “逃跑 - 跟随” 双机器人系统，具体描述如下：

（1）核心目标：

通过键盘手动控制一个机器人（逃跑者，robot1）移动，另一个机器人（跟随者，robot2）自动追踪逃跑者的位置并保持设定距离跟随。

（2）实现方式：

在 Gazebo 仿真环境中加载两个 TurtleBot3（Burger 模型）机器人，分别作为逃跑者和跟随者。

逃跑者通过键盘控制节点（turtlebot3\_teleop\_key）接收 WASD 键输入，输出速度指令（/robot1/cmd\_vel）。

跟随者通过自定义节点（follower\_node.py），利用 TF 坐标变换监听逃跑者的实时位置，通过比例控制算法生成速度指令（/robot2/cmd\_vel），实现自动追踪。

（3）关键技术：

多机器人命名空间（ns）和 TF 前缀（tf\_prefix）隔离，避免话题和坐标冲突。

统一坐标系根节点（world），确保两个机器人的 TF 坐标链连通，实现相对位置计算。

速度控制逻辑：跟随者根据与逃跑者的距离和角度动态调整线速度和角速度。

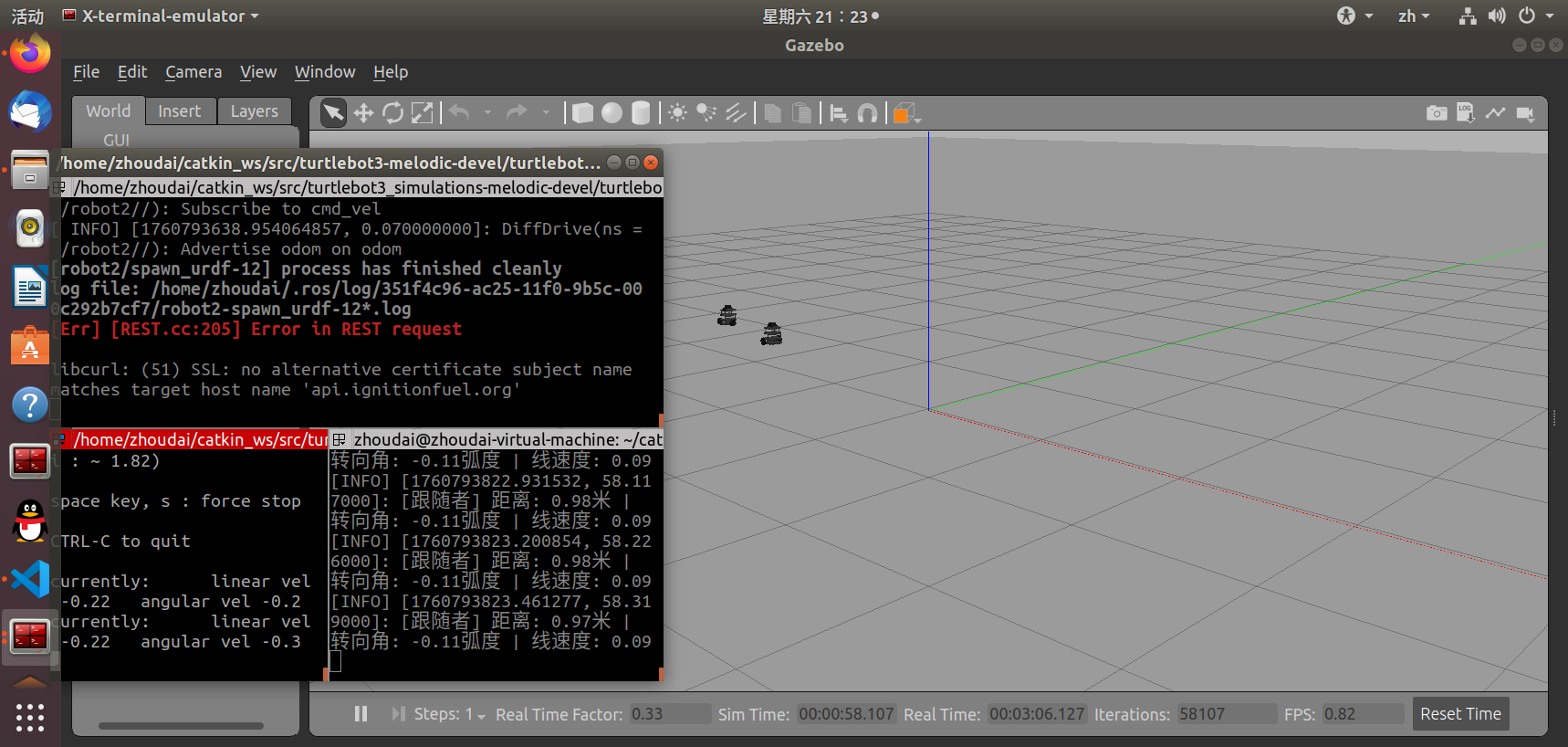
（4）操作流程：

启动双机器人仿真环境，加载 Gazebo 世界和机器人模型。

启动键盘控制节点，手动操作逃跑者移动。

启动跟随者节点，自动追踪逃跑者，形成 “逃跑 - 跟随” 交互。

**截图：**



三、心得体会

本次实验都非常有趣，让我快速了解gazebo，slam，navigation等，可以很快地复现github上的许多有趣的案例，多亏了老师提供的压缩包，否则自己从github克隆下来的用不了。自定义任务做了一个双机器人跟随的任务，顺便复习一下TF。