机器人操作系统结课报告

姓名：秦旗峰

学号：2023302143029

专业：导航工程（智能导航实验班）

课程：机器人操作系统与应用

武汉大学

2025.05

目录

1 任务背景.....3
2 实现步骤与技术路线.....3
3 代码解释.....4
3.1 SetGoal.srv——自定义服务文件.....4
3.2 TurtleBot3Navigation.action——自定义动作文件.....5
3.3 goal_service_server.cpp——动作客户端与服务接收端.....5
3.4 navigation_action_server.cpp——动作服务端.....6
3.5 robot_navigation.launch——一键启动.....8
4 运行结果展示.....9
5 反思与总结.....13
6 课程建议.....13
参考： 14

1 任务背景

ROS 作为开源机器人操作系统，提供了通信、仿真和算法开发的标准化框架。本项目基于 Gazebo 的 TurtleBot3 仿真实验环境，实现自主导航功能。

本次结课任务要求通过 Gazebo 仿真环境，设计服务-动作(Service-Action)的通信框架，使得 TurtleBot3 能够通过服务接收目标点指令后利用动作服务器自动导航到指定位置并反馈运行信息。

2 实现步骤与技术路线

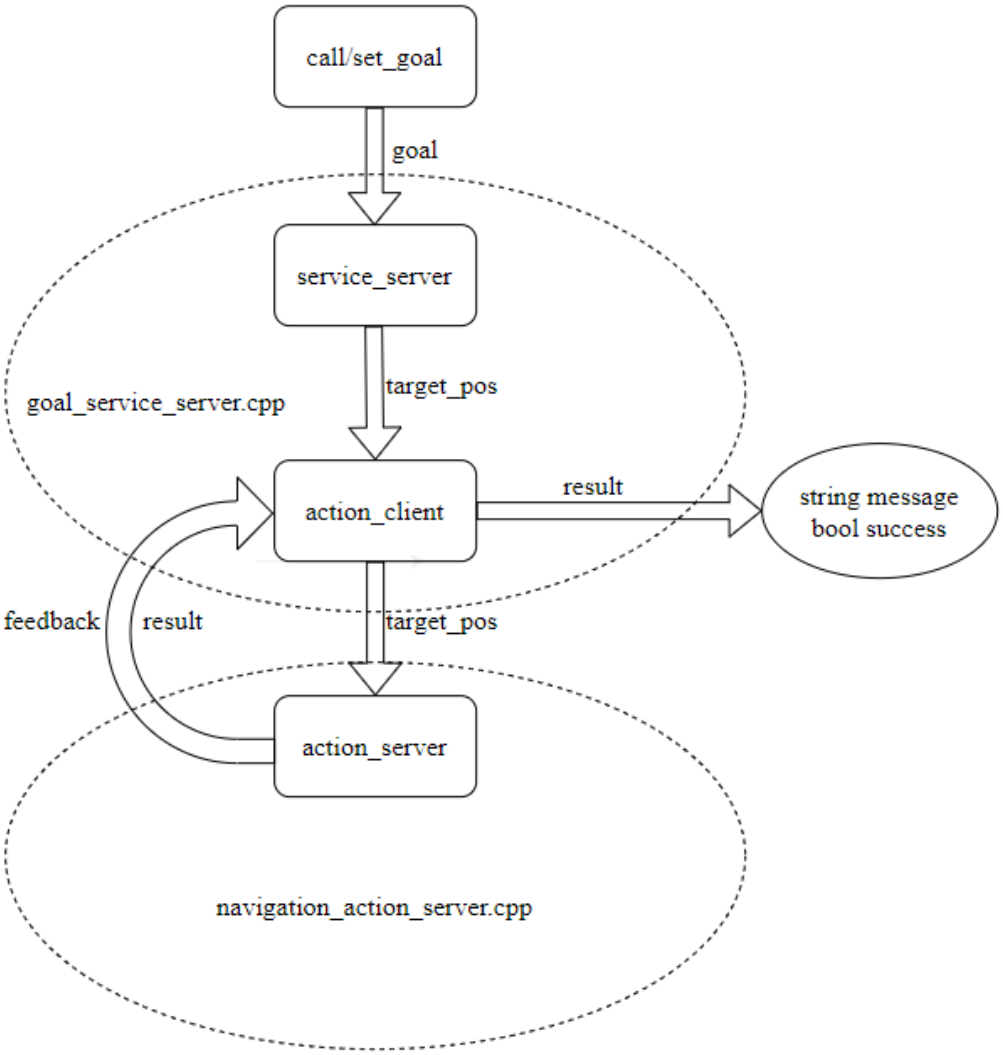

图 2.1 技术路线框图示意

• **实现步骤：**

1.创建 service 服务端；

2.创建 action 客户端；

3.service 服务端接收到 goal 后传递给 action 客户端的 target_pos；

4.创建 action 服务端；

5.创建 tf 监听者；

6.创建 cmd_vel 速度发布者；

7.tf 监听 odom 和 base_footprint 坐标关系；

8.当 action 服务端接收到来自客户端的 target_pos 后计算距离和角度；

9.按照计算的距离、角度以及速度发布逻辑发布机器人运行速度；

10.tf 监听实时坐标变换，获取 current_pose 并实时 feedback；

11.当 distance_to_goal<0.1 时，发布 0 速度，机器人静止并输出运行结果。

3 代码解释

3.1 SetGoal.srv——自定义服务文件

SetGoal.srv 是一份自定义服务类型文件，服务的请求(request)是终端输入的目标点坐标 geometry_msgs/Point goal，服务的回应(response)是运行结果的 bool success 和 string message；

```
geometry_msgs/Point goal
---
bool success
string message
```

图 3.1 SetGoal.srv 文件

geometry_msgs/Point 类型是一个几何三维坐标，包含 x，y，z 三个分量，用于保存终端调用服务的目标点坐标；bool success 类型返回运行是否成功的 bool 值，string message 作为服务结果的消息，服务成功后返回 message= “Navigation completed”。

3.2 TurtleBot3Navigation.action——自定义动作文件

TurtleBot3Navigation.action 定义了 tb3(TurtleBot3)的行动过程，包括目标值 geometry_msgs/PoseStamped target_pos；结果 bool success 和 string message；反馈 geometry_msgs/Pose current_pose 和 float32 distance_to_goal。

```
geometry_msgs/PoseStamped target_pos
---
bool success
string message
---
geometry_msgs/Pose current_pose
float32 distance_to_goal
```

图 3.2 TurtleBot3Navigation.action 文件

目标值 target_pos 属于 geometry_msgs/PoseStamped 类型，服务端在终端接收到命令后将目标点坐标传递给 PoseStamped 中的三维坐标信息。在 tb3 运动过程中，也是通过 target_pos 中的 frame_id 确定坐标系统关系以及 orientation 四元数确定角度关系。结果的定义与 SetGoal.srv 相同。动作过程反馈值为机器人当前的坐标信息和四元数信息 current_pose，参与实时的角度计算与距离计算，distance_to_goal 则是判断是否到达目标点的重要信息。

3.3 goal_service_server.cpp——动作客户端与服务接收端

goal_service_server.cpp 是服务的提供者，接收到服务请求后向动作服务器发布移动指令的请求，实时输出当前点到目标点的距离，在服务完成后，告知请求执行完毕(Navigation completed)。

在代码中，定义了 GoalServiceServer 类来搭建动作的客户端和服务接收端，其中 service_注册“set_goal”服务并调用动作服务器，action_client_为动作的客

```
class GoalServiceServer {
private:
    ros::ServiceServer service_;
    actionlib::SimpleActionClient<burger::TurtleBot3NavigateToGoalAction> action_client_;    //动作客户端
```

图 3.3 GoalServiceServer 类的定义

户端，接收到来自 service_的服务请求后，与动作的服务端建立联系，发布目标点坐标。

```
public:
    GoalServiceServer(ros::NodeHandle* nh) :
        action_client_("navigate_to_goal", true) {
        service_ = nh->advertiseService("set_goal", &GoalServiceServer::handleGoalRequest, this);
        ROS_INFO("等待服务启动");
        action_client_.waitForServer();
        ROS_INFO("连接到服务器，准备出发");
    }
```

图 3.4 构造函数

handleGoalRequest 是服务请求的回调函数，当接收到服务请求后转递目标点的坐标信息(包括坐标系统与坐标)，向动作的服务端发布目标，等待结果实时订阅反馈信息并输出反馈。当动作服务器启动时，输出服务器连接信息。

```
bool handleGoalRequest(
    burger::SetGoal::Request& req,
    burger::SetGoal::Response& res
) {
    burger::TurtleBot3NavigateToGoalGoal goal;
    goal.target_pos.header.frame_id = "odom";  // 使用 odom 坐标系
    goal.target_pos.pose.position = req.goal;
    action_client_.sendGoal(
        goal,
        actionlib::SimpleActionClient<burger::TurtleBot3NavigateToGoalAction>::SimpleDoneCallback(),
        actionlib::SimpleActionClient<burger::TurtleBot3NavigateToGoalAction>::SimpleActiveCallback(),
        boost::bind(&GoalServiceServer::feedbackCallback, this, _1)
    );
    bool finished = action_client_.waitForResult(ros::Duration(60.0));
    if (finished) {
        auto result = action_client_.getResult();
        res.success = true;
        res.message = "Navigation completed.";
    } else {
        res.success = false;
        res.message = "Navigation timed out.";
    }
    return true;
}
void feedbackCallback(const burger::TurtleBot3NavigateToGoalFeedbackConstPtr& feedback) {
    float distance_to_goal = feedback->distance_to_goal;
    ROS_INFO("距离目标点还有 %.2f 米", distance_to_goal);
}
```

图 3.5 回调函数

为了防止因为不确定因素导致的错误而陷入死循环，我设置了最高运行时间为 60s，若超出运行时长，则提示“Navigation timed out”。

3.4 navigation_action_server.cpp——动作服务端

navigation_action_server.cpp 动作服务端的主要作用是接收来自客户端的请求后发布速度指令，控制机器人小车运行到指定目标点(target_pose_)。

程序中需要创建动作服务端 action_server_，坐标变换存储器 tf_buffer_，坐标变换监听者 tf_listener_，速度发布者 vel_pub_。

```
class NavigationActionServer {
private:
    actionlib::SimpleActionServer<burger::TurtleBot3NavigateToGoalAction> action_server_;    //action的服务器
    tf2_ros::Buffer tf_buffer_;    // 用于存储坐标变换
    tf2_ros::TransformListener tf_listener_;    //用于监听坐标变换
    ros::Publisher vel_pub_;    // 发布速度指令
    geometry_msgs::PoseStamped target_pose_;    // 目标位置
```

图 3.6 NavigationActionServer 类的定义

```
public:
    NavigationActionServer(ros::NodeHandle* nh) :     //构造函数
        action_server_(*nh, "navigate_to_goal", false),
        tf_listener_(tf_buffer_) {
        action_server_.registerGoalCallback(boost::bind(&NavigationActionServer::handleGoal, this));    //bind函数，回调函数为handleGoal
        action_server_.start();
        vel_pub_ = nh->advertise<geometry_msgs::Twist>("cmd_vel", 10);
    }
```

图 3.7 构造函数

当接收到客户端的请求时，会首先在 action_server_中注册动作类型，后调用回调函数 handleGoal，在回调函数中完成坐标转换，距离角度计算和速度发布。

```
void handleGoal() {
    auto goal = action_server_.acceptNewGoal();
    target_pose_ = goal->target_pos;
    ROS_INFO("接收到新的坐标点: (%.2f, %.2f)",
             target_pose_.pose.position.x,
             target_pose_.pose.position.y);

    ros::Rate rate(10);
    while (ros::ok()) {
        geometry_msgs::TransformStamped transform;
        try {
            transform = tf_buffer_.lookupTransform("odom", "base_footprint", ros::Time(0));    //建立odom和base_footprint之间的坐标变换
        } catch (tf2::TransformException& ex) {
            ROS_WARN("坐标转换出错: %s", ex.what());
            ros::Duration(0.1).sleep();
            continue;
        }

        double dx = target_pose_.pose.position.x - transform.transform.translation.x;
        double dy = target_pose_.pose.position.y - transform.transform.translation.y;
        double distance = sqrt(dx*dx + dy*dy);
        double yaw_target = atan2(dy, dx);

        tf2::Quaternion q_current(
            transform.transform.rotation.x,
            transform.transform.rotation.y,
            transform.transform.rotation.z,
            transform.transform.rotation.w
        );
        double yaw_current = tf2::getYaw(q_current);     //获取当前的偏航角

        double angle_error = yaw_target - yaw_current;
        angle_error = atan2(sin(angle_error), cos(angle_error));

        burger::TurtleBot3NavigateToGoalFeedback feedback;    //实时反馈当前坐标
        feedback.current_pose.position.x = transform.transform.translation.x;
        feedback.current_pose.position.y = transform.transform.translation.y;
        feedback.distance_to_goal = distance;
        action_server_.publishFeedback(feedback);

        geometry_msgs::Twist cmd_vel;
```

图 3.8 回调函数（接下图）

接收到目标点作为动作的目标值，运动过程中需要实时监听 odom 坐标系与 tb3 的 base_footprint 坐标系的转换关系，用于计算当前位置到目标点的距离与角度关系并将其赋值给 feedback 信息。在接下来的步骤中，我们将利用这些距离与角度信息完成速度的控制与发布。

```cpp
    geometry_msgs::Twist cmd_vel;

    if (distance < 0.1) {
        // 到达目标点
        cmd_vel.linear.x = 0.0;
        cmd_vel.angular.z = 0.0;
        vel_pub_.publish(cmd_vel);
        action_server_.setSucceeded();

        ROS_INFO("到达目标点!");

        break;
    } else if (fabs(angle_error) > 0.1) {    //优先调整角度
        cmd_vel.linear.x = 0.0;
        cmd_vel.angular.z = 0.3 * angle_error;
        ROS_INFO("调整角度: 角度误差=%.2f rad, 角速度=%.2f", angle_error, cmd_vel.angular.z);
    } else {
        cmd_vel.linear.x = std::min(0.3 * distance, 0.3);
        cmd_vel.angular.z = 0.3 * angle_error;
        ROS_INFO("向前进: 线速度=%.2f, 角速度=%.2f", cmd_vel.linear.x, cmd_vel.angular.z);
    }

    vel_pub_.publish(cmd_vel);
    rate.sleep();
  }
}
```

图 3.9 回调函数（接上图）

机器人的速度控制逻辑为：优先调整角度，再调整距离。当接收到新的坐标点时，首先计算 angle_error，即当机器人当前时刻自身的角度与和目标点连线角度的差异，当角度差异较大时通过发布角速度信息调整角度，角度较小时，发布线速度指令，如此逐渐调整来完成到达目标点的目的。当机器人当前坐标与目标点之间的距离小于 0.1 个单位时，我们认为其到达了目标点，返回运行结果并停止发布 0 速度。

3.5 robot_navigation.launch——一键启动

goal_service_server.cpp | navigation_action_server.cpp | robot_navigation.launch

src > burger > launch > robot_navigation.launch

```xml
<launch>

  <arg name="model" default="burger" />
  <env name="TURTLEBOT3_MODEL" value="$(arg model)" />

  <!-- 启动Gazebo仿真环境 -->
  <include file="$(find turtlebot3_gazebo)/launch/turtlebot3_empty_world.launch">
    <arg name="model" value="$(arg model)" />
  </include>

  <!-- 启动SLAM -->
  <include file="$(find turtlebot3_slam)/launch/turtlebot3_slam.launch">
    <arg name="slam_methods" value="gmapping" />
    <arg name="open_rviz" value="false" />
  </include>

  <!-- 启动服务端和动作服务器-->
  <node pkg="burger" type="goal_service_server" name="goal_service_server" output="screen" />
  <node pkg="burger" type="navigation_action_server" name="navigation_action_server" output="screen" />
</launch>
```

图 3.10 robot_navigation.launch 文件

ROS 中的 launch 文件可以一键启动多个节点。在 robot_navigation.launch 文

件中，将 Gazebo 仿真环境、机器人 slam(本实验并没有用到)、goal_service_server (动作客户端与服务接收端)和 navigation_action_server(动作服务端)一键启动。

4 运行结果展示

首先，我们需要打开两个终端，将路径改为我们的工作空间 demo01_ws 并配置环境属性(在命令行键入：source devel/setup.bash);

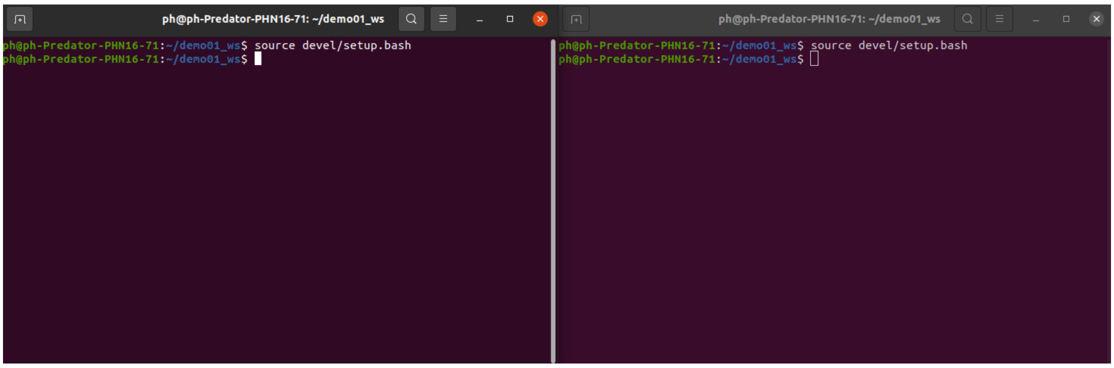

图 4.1 打开终端并配置属性

在其中一个终端输入 roslaunch burger robot_navigation.launch 以启动 launch 文件，Gazebo 和其他节点相继启动。

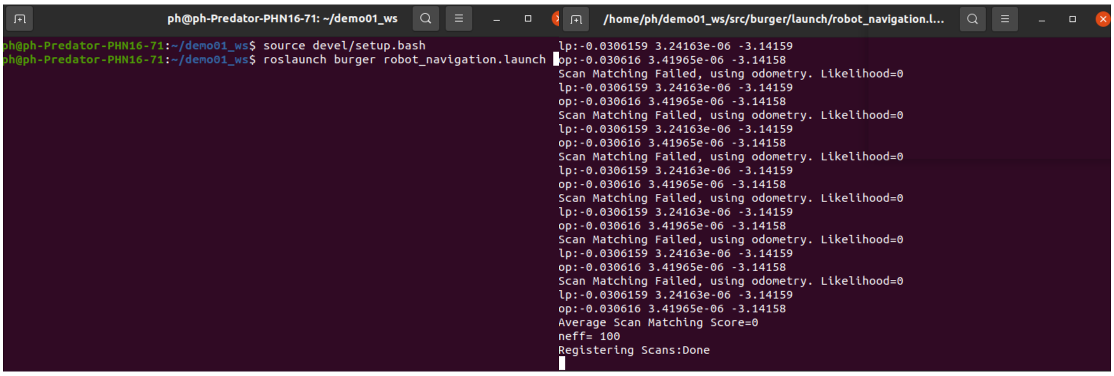

图 4.2 启动 launch 文件

此时可以在 Gazebo 仿真环境中看到 TurtleBot3 机器人静止在坐标系原点，被启动的终端持续输出 Scan 结果。

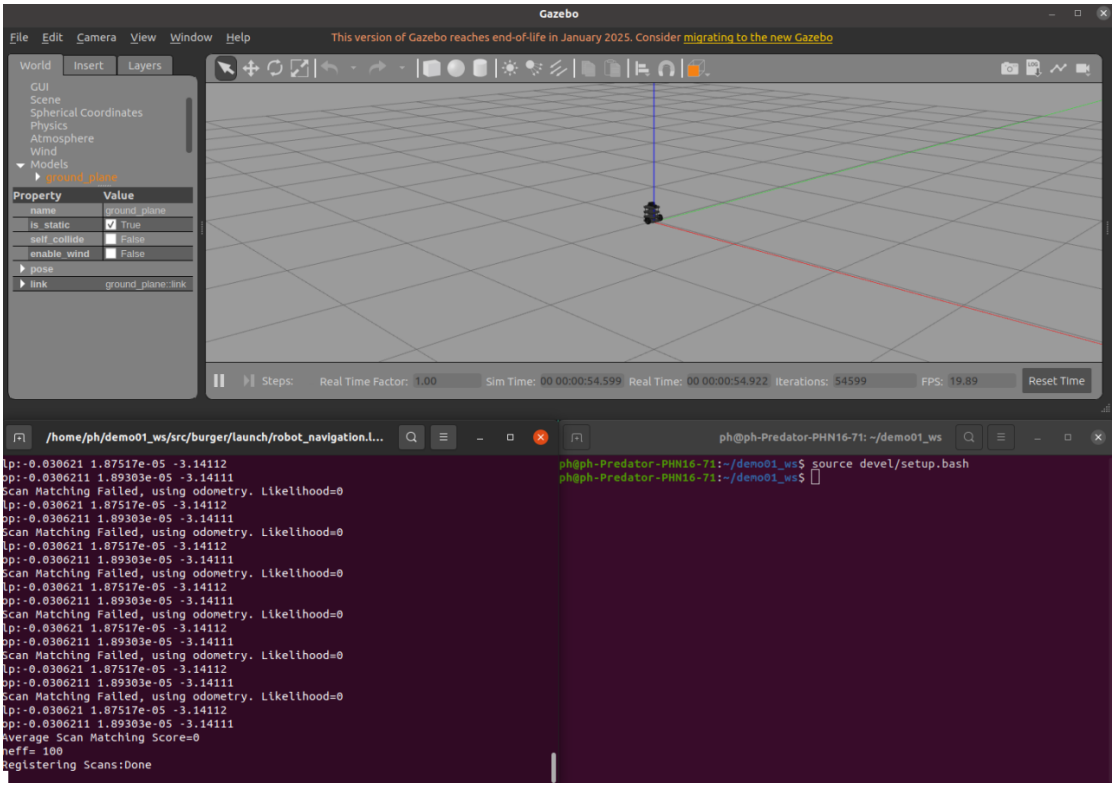

图 4.3 机器人静止在原点

这时可以在另一个终端请求服务，输入 rosservice call /set_goal“goal: x:1.0 y:1.0 z:1.0”，客户端接收到请求后向动作服务端发出请求，机器人开始运动并实时反馈运动信息。

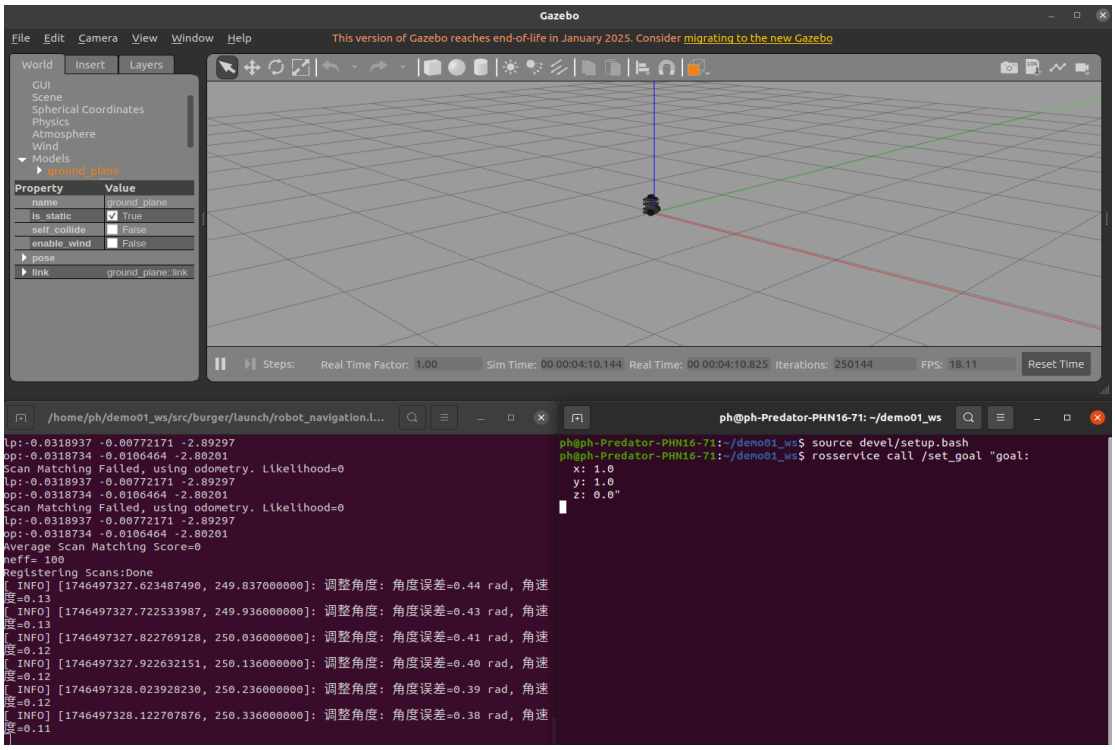

图 4.4 收到请求后开始运动

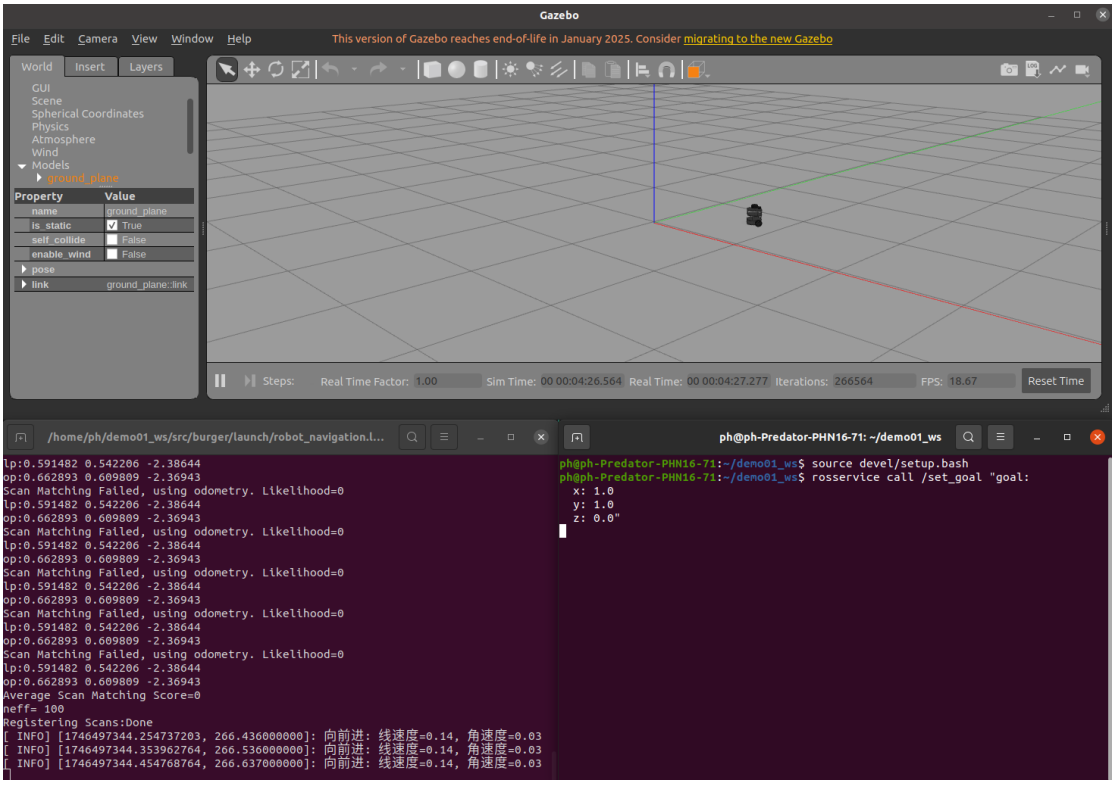

图 4.5 机器人正在运动

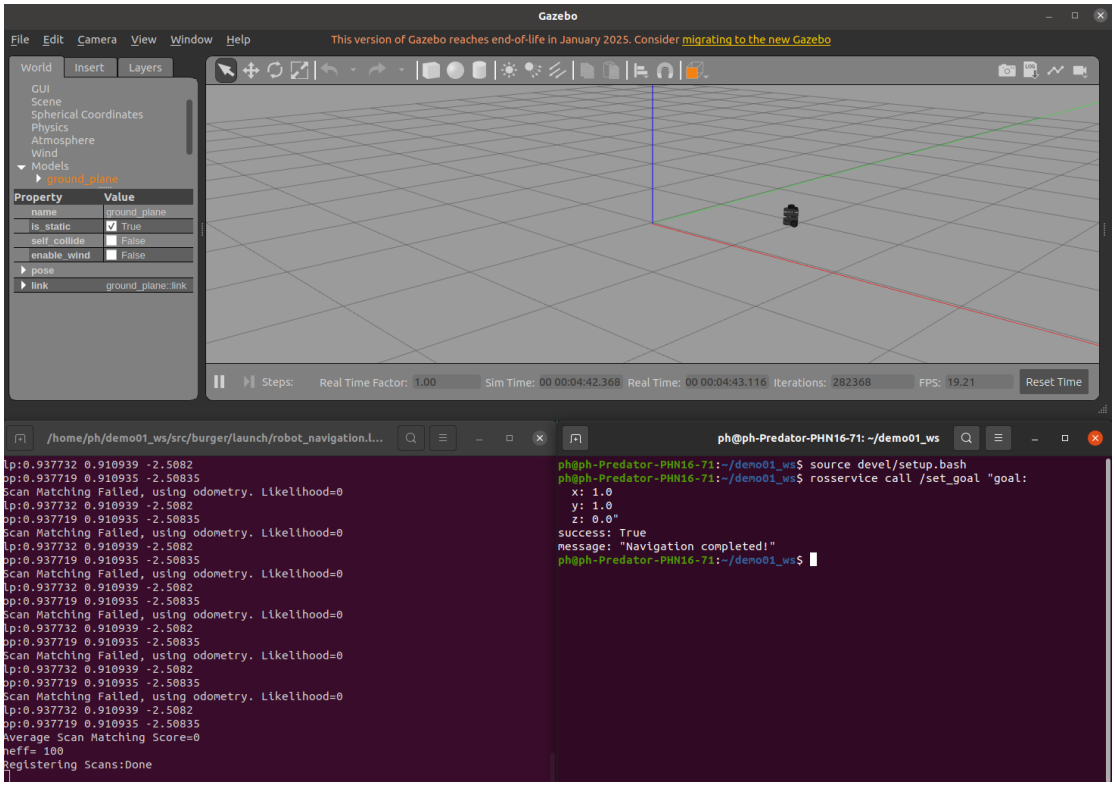

图 4.6 机器人运动到目标点后输出结果信息

再次输入 rosservice call /set_goal“goal: x:3.0 y:4.0 z:0.0”加以重复验证：

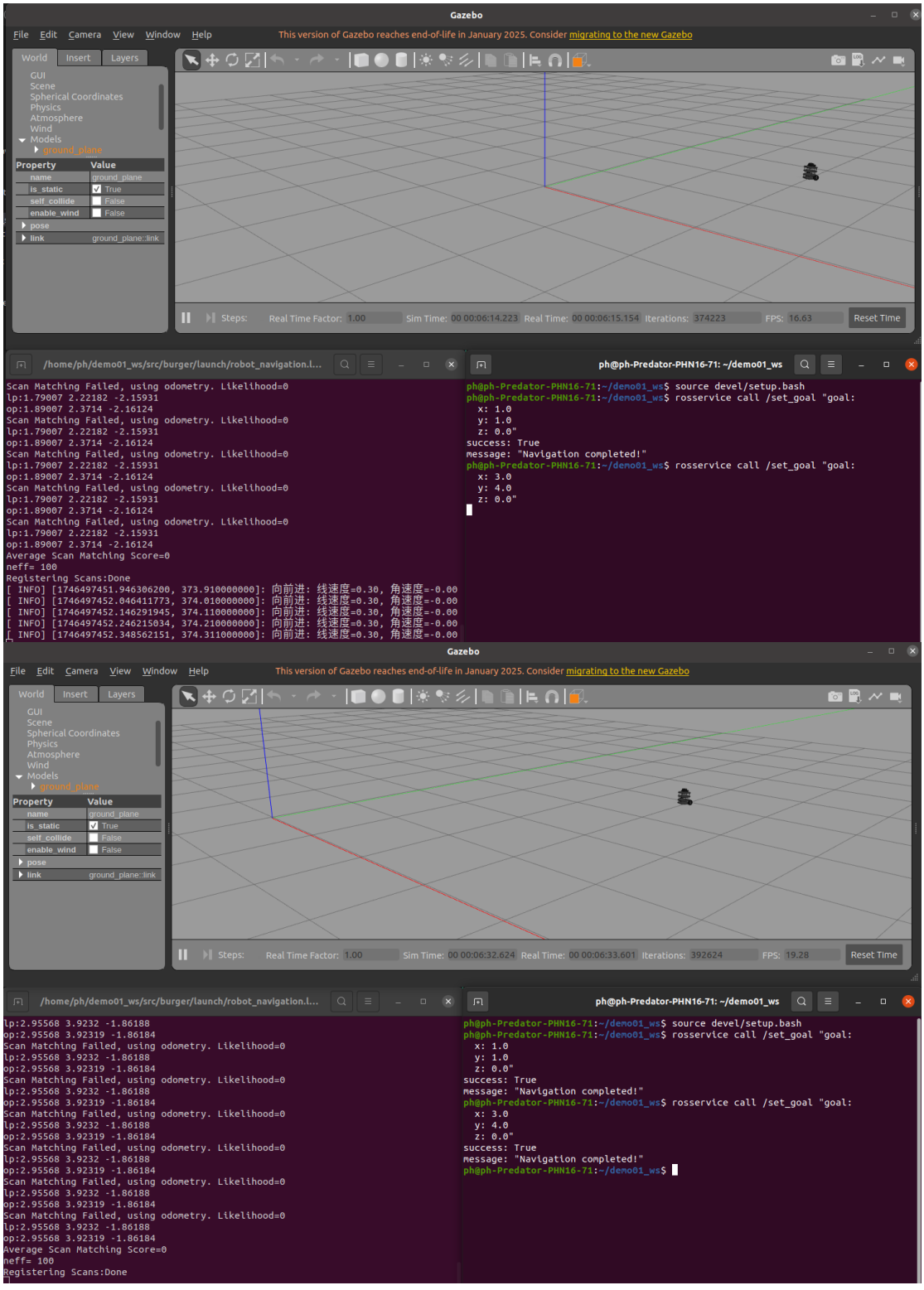

图 4.7 设置新的目标点重复验证

运行结果正确，更换多个不同目标点，机器人均能运行到指定点。

5 反思与总结

为了实现本次的结课任务，我认真复习了上课所学的 service、tf、launch 和 action，并在哔站和博客上学习了一些大佬们的 ROS 教程，为我能顺利完成打下基础。

本次任务可以拆解为三层，第一层是 service 服务端，我们只需要编写服务端程序，等待终端命令行请求服务再执行相关文件；第二层是 action 客户端，这一层与 service 服务端紧密相连，当 service 接收到服务请求后，service 将请求传递给 action，由 action 客户端向 action 服务端请求动作服务；第三层就是 action 服务端，接收来自 action 客户端的指令，发布速度指令，完成任务。

实验中遇到了很多困难，我们需要通过 tf 坐标变换计算距离和方向，但是不知道哪些坐标系可以为我们所用。我在仔细分析了 tf 树和查阅一些博客之后，才得知可以建立机器人 odom 和 base_footprint 坐标系的关系可以实现这个过程。另一个问题就是 action 类型的消息不熟练，许多函数在最初进行任务时不知道可以调用，这个在博客上资料也很少，但是问了 AI。

编程过程中我遇到了机器人在目标点周围打转转的情况，我多次优化了速度控制逻辑，才采用先调整角度后调整线速度的策略(运动路径上存在障碍就不适用了)，并调整了速度大小，采用较小的速度发布，不至于让小车运行过快导致偏差变大。

最后的问题就是 BURGER 模型缺失的问题，在前几周我在我的 Ubuntu 系统上安装了 mini-conda 和深度学习的包，一些 python 插件和环境变量可能被做了修改，导致我的机器人模型一直无法正常显示，这个问题我还在努力解决。因此我在代码写完之后，程序的调试、修改和运行是在彭昊同学电脑上完成。

这些问题都能暴露出自身的一些不足，对 Ubuntu 系统不够熟练、逻辑并没有那么清晰等，我在今后的学习生涯中努力改进自己，超越自己。

6 课程建议

机器人课程是一个很有意思的课。我对课程的建议如下：

1.增加课时 2.依托 vscode 进行讲解和编程，这样同学们能够更好地掌握编

译方式，编程时也有更多提示 3.增加对 Linux 系统常用操作的教学计划占比。

这些就是我的一些建议，感谢陈老师和孙老师的辛勤付出。

参考：

https://blog.csdn.net/qq_51013517/article/details/143106187

https://blog.csdn.net/Travis_X/article/details/130431652

https://www.bilibili.com/video/BV1pQ5bz3E8B

https://blog.csdn.net/weixin_43134049/article/details

http://www.autolabor.com.cn/book/ROSTutorials