湖南工商大学

《智能机器人应用综合设计》课程报告

题　　目：	基于 ROS 的智能避障小车设计
姓　　名：	蒋海崇、黄沛豪、吴杭玻
学　　号：	2009090144、2009070236 2009090140
专　　业：	大数据与人工智能
班　　级：	数智 2002
指导教师：	姜林
职　　称：	副教授

前沿交叉学院

2023 年 2 月

课程报告评审表

等级 成绩 组成	优秀	良好	中等	及格	不及格
课程设计报告	Gazebo 小车仿真（20 分） 传感器感知与运动控制（20 分） 建图和导航（20 分） 智能应用案例（40 分）				
综合成绩评定：　　评阅老师（签章）： 年　　月　　日					

目　录

基于 ROS 的智能避障小车设计

1 ROS 简介

ROS 是英文 RobotOperatingSystem 的缩写，中文含义为机器人操作系统。ROS 是用于编写机器人软件的灵活框架，集成大量的工具、库和协议，提供类似操作系统所提供的功能，包括硬件抽象的描述、底层驱动、共用功能的执行、程序间消息的传递、程序发行包管理、简化复杂多样的机器人平台下的复杂任务创建与稳定行为控制。

ROS 的首要目标是提供一套统一的开源程序框架，用以在多样化的现实世界与仿真环境中实现对机器人的控制。ROS 的设计者将 ROS 表述为“ROS = Plumbing+Tools+Capabilities+Ecosystem”，即 ROS 是通讯机制、工具软件包、机器人高层技能以及机器人生态系统的集合体。

1.1 ROS 的发展史

在 2007 年 ROS 是斯坦福大学人工智能实验室与机器人技术公司 Willow Garage 合作的个人机器人项目 PRP（PersonalRobotsProgram）。2008 年之后由 WillowGarage 进行推动。随着 PR2 在 ROS 框架上不可思议的表现如：叠衣服，插插座，做早饭等功能如图，ROS 得到更多的关注。2010 年 WillowGarage 正式以开放源码的形式发布 ROS 框架，很快在机器人研究领域掀起了 ROS 开发与应用的热潮。

1.2 ROS 小车

EPRobot 智能小车是为本科、高职等不同人群计算机编程、机器人开发以及嵌入式系统开发等方向定制开发的学习平台。EPRobot 基于最新的树莓派 4B 卡片式电脑开发，搭载深度定制的底盘控制器，让小车的功能与性能达到极致。首先，让我们先了解一下小车的结构和组成。 图 1 为 EPRobotV1 版本小车，该版本为开放式框架，不带语音识别与播报模块。

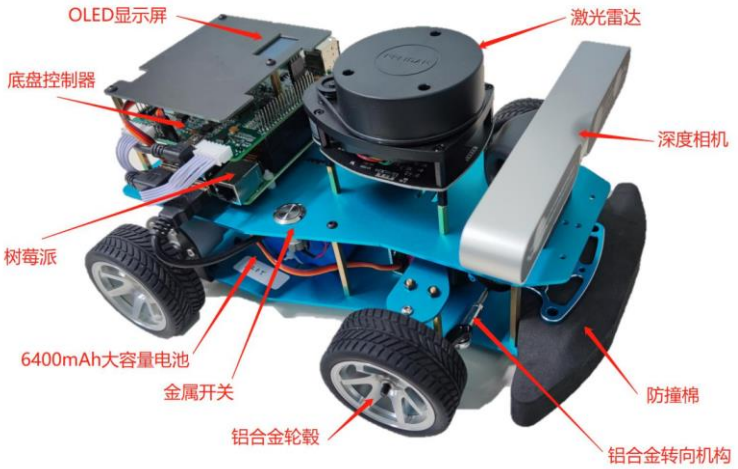

图 1 EPRobotV1 小车结构图

小车主要由树莓派 4B、底盘控制器、激光雷达、深度相机、2 个配有编码器的电机以及底盘组成。下面几张图介绍了小车的接线情况。从图 4 中可看出，主板左侧提供了 2 路 USB 的 5V 供电口，可用于为扩展迷你路由器、5V 扬声器等设备供电，没有通讯功能。还提供了 1 路 12V 电池输出，也可为外接 12V 的设备供电。从图 5 中可以看出，主板后方连接了风扇，其位于树莓派的下方，用于在树莓派高速运算时的散热，小车出厂程序并未启用该风扇，用户可自行修改底层控制器的程序来增加风扇控制，该接口支持转速调节。用户可使用树莓派的 3.5mm 音频口扩展扬声器，搭配主板的 USB 供电使用。从图 6 可以看出，有一根 USB 数据线连接了 STM32 底板的 miniUSB 口和树莓派的 USB 数据口，该线用于树莓派与激光雷达的数据交互。由于激光雷达的调速接口位于 STM32 底板，所以树莓派必须要通过底板与激光雷达进行通信。接在树莓派的 USB 口上的黑色小模块是无线手柄的收发器。树莓派多余的 USB 口可用于扩展麦克风阵列。

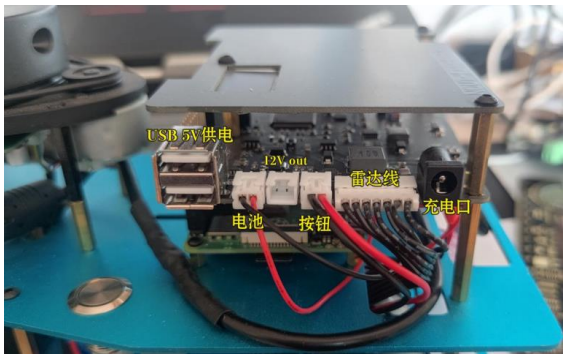

图 2 EPRobotV1 小车左侧接线图

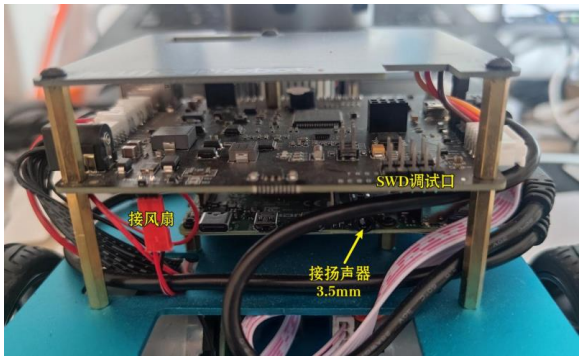

图 3 EPRobotV1 小车后方接线图

EPRobot 小车的大脑采用最新版本的树莓派 4B（内存 4GB）平台，运算能力和资源配置大幅提高。它内置了 Ubuntu18.04 操作系统，并安装了 ROS-melodic 版本的机器人开发框架，不仅可以进行嵌入式 Linux 系统以及 ROS 系统的实践与开发，同样能够非常方便快捷的实现各种机器人控制的算法与程序。

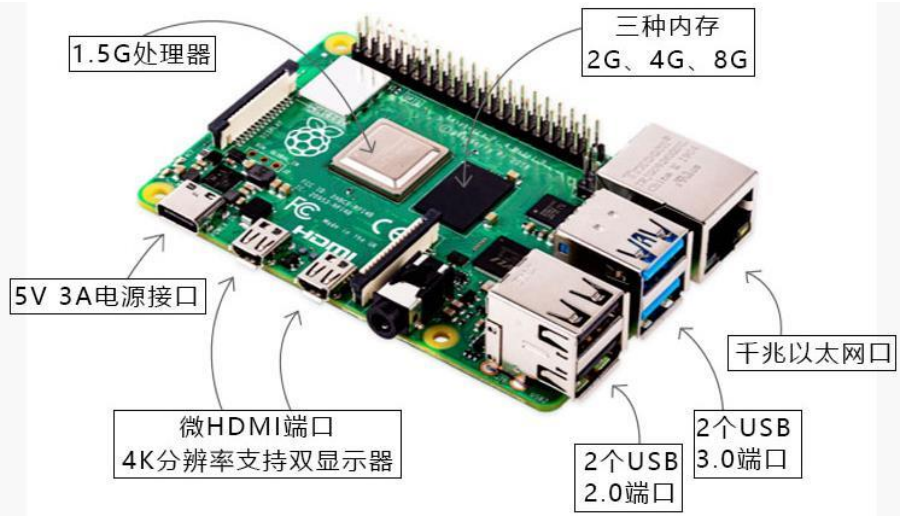

图 4 树莓派 4B 资源介绍

EPRobot 采用后轮驱动、前轮转向的阿克曼底盘结构，运动学模型最接近真实汽车结构，可以轻松模拟自动驾驶的各种功能。小车搭载的激光雷达用于实现快速实时壁障，摄像头用于图像识别和循迹，能够驾驭多种应用场景，如竞速小车、深度学习、机器视觉等。EPRobot 根据智能车的控制特点深度定制基于 ARMCortex-M4 单片机的底盘控制器，将单片机大部分资源进行利用，可在此平台完成单片机相关课程的所有实验教学，同时能够进行 RT-Thread 等微型实时操作系统的开发与教学；底盘控制器使用 PID 算法和 MFAC 无模型自适应控制算法精确控制小车的速度和转向角度。

2 编程的准备工作

2.1 ROS 系统的安装、使用与开发

2.1.1 内容提要

1.将 Ubuntu 设置为国内源

2.ROS 系统的安装

3.安装测试

4.ROS 系统的使用

5.ROS 系统的开发

2.1.2 ROS 系统的安装

首先将 Ubuntu 更新国内源，这样，下载速度会更快。

sudo apt-get up<ickey>date

sudo apt-get up.bash" >> ~/.bashrc

//使环境变量生效，终端中输入:

$ source ~/.bashrc

sudo apt-get install python-rosinstall

安装好后输入以下命令测试：

(1) 启动

roscore

(2) 启动节点

rosrun turtlesim turtlesim_node

(3) 运行

rosrun turtlesim turtle_teleop_key

2.1.3 ROS 系统的使用

在开始使用 ROS 之前需要初始化 rosdep。rosdep 可以方便在需要编译某些源码的时候为其安装一些系统依赖，同时也是某些 ROS 核心功能组件所必需用到的工具。之后进行环境配置 echo"source /opt/ros/Kinetic/setup.bash">> ~/.bashrcsource~/.bashrc。环境配 置好后安装 rosinstall。rosinstall 是 ROS 中一个独立分开的常用命令行工具，它可以方便通过一条命令就可以给某个 ROS 软件包下载很多源 码树。这样 ROS 就安装好了。运行 ROS 之前要进行工作环境配置：管理 环境、创建 ROS 工作空间、之后在工作空间中编写代码以及安装需要的功 能包就可以完成相应的 ROS 运行了。例如启动主节点服务器之后运行 rosrun turtlesim turtlesim _ node 指令就可以运行小海龟的界面，如下图所示：

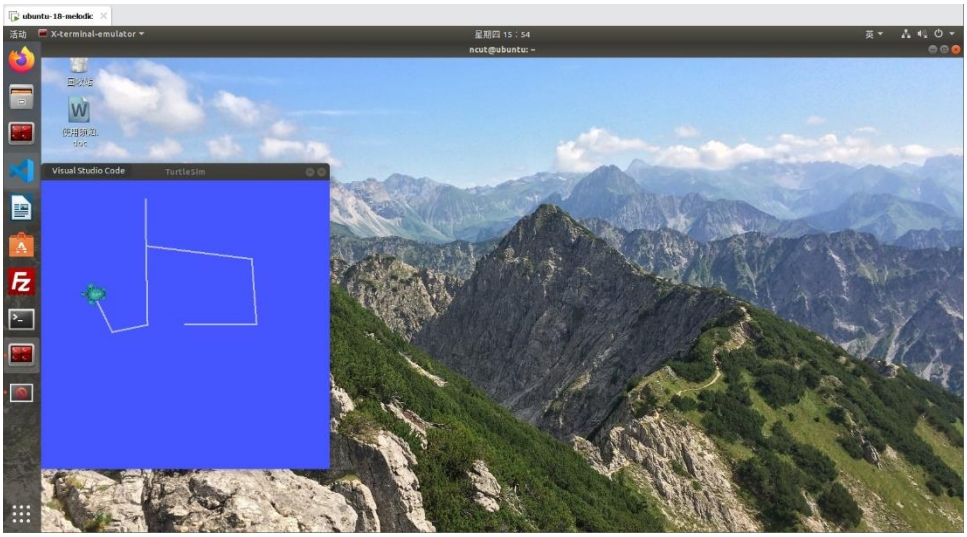

图 5 小乌龟运行实例

2.1.4 ROS 系统的开发

1. 新建工作环境

2.首先新建一个目录（文件夹）作为之后的工作环境，在此我们命名为 catkin_ws，具体命令如下（-p 不一定需要）：

$ mkdir -p ~/catkin_ws

3.正常情况下文件夹便成功建立了。之后我们需要在次文件夹中新建一 个 src 文件夹

$ mkdir -p ~/catkin_ws/src

4.我们切换到 src 文件夹下，预编译工作空间，为了生成 CMakeLists.txt 文件，具体命令如下：

$ cd ~/catkin_ws/src

$ catkin_init_workspace

5.切换到工作空间（在这里是 catkin_ws）根目录，编译后完成整个 工作空间便搭建完成了。

$ cd ~/catkin_ws

$ catkin_make

6.配置 ROS 环境

将新建的工作空间添加到 ROS 环境中，这样每次打开一个新的终端，系统 便会调用一个 bashrc 文件，对环境进行配置。其中/home/uav/catkin_ws 为工作空间的完整路径。

$ echo "source /home/uav/catkin_ws/devel/setup.bash" >> ~/.bashrc

之后再开一个新的终端，运行文件.bashrc

$ source ~/.bashrc

7.创建一个 ROS 综合功能包

8.在 catkin_ws/src 文件夹中建一个功能包,功能包文件名为 beiginner_tutorials25

$ cd ~/catkin_ws/src

$ catkin_create_pkg beginner_tutorials std_msgs roscpp rospy

之后在 src 中创建 CPP 文件或 PY 文件，进行程序编写，之后更改 Cmakelist.txt 文件，编译后即可进行我们创建功能包的使用。

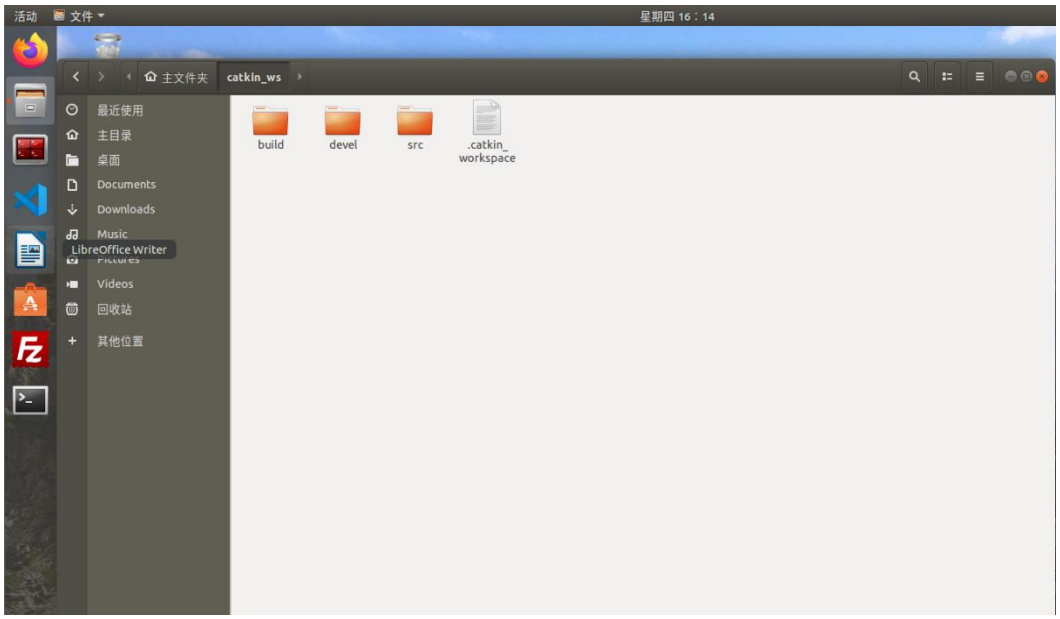

图 6　环境创建实例

3 课程设计任务与要求

本课程设计任务是应用机器人小车基本模型，基本组件的，实现小车的运动控制，雷达导航，视觉导航，以及定位巡航的应用算法，进行实际问题的开发，独立设计和开发一个能够完成一定功能的智能机器人小车的系统。

具体任务为：

（1）以 Gazebo 对机器人小车进行物理仿真，并创建机器人 URDF 模型。要求在 rviz 中显示，通过键盘对小车进行运动控制。要求：小车模型必须自主设计，若有任意 2 个模型完全相同，则相同者均被判为 0 分。

（2）实现传感器感知及运动控制，要求使用传感器感知的数据（如：语音，图像，视频）对小车进行运动控制，至少实现 2 种。

（3）在虚拟环境中实现对小车的建图和导航，要求导航与 SLAM 的同步仿真。

（4）智能应用案例开发，要求在小车运动或导航过程中，遇到某条件触发，给出相应智能响应。如：a）小车定点到某地，发现地面有垃圾，可以进行图片拍摄或语音播报，或屏幕文字显示等；b)小车行进中不断通过摄像机扫描周围物体，若遇特殊物品，或是特殊标记的物品，能进行播报。总之，该部

分可以充分想象开发应用程序，只要合理即可。

4 实验操作

4.1 任务一：小车模型的设计

4.1.1 小车模型设计

创建一个四轮圆柱状机器人模型，机器人参数如下，底盘为圆柱状，半径 10cm，高 8cm，四轮由两个驱动轮和两个万向支撑轮组成，两个驱动轮半径为 3.25cm,轮胎宽度 1.5cm，两个万向轮为球状，半径 0.75cm，底盘离地间距为 1.5cm(与万向轮直径一致)。

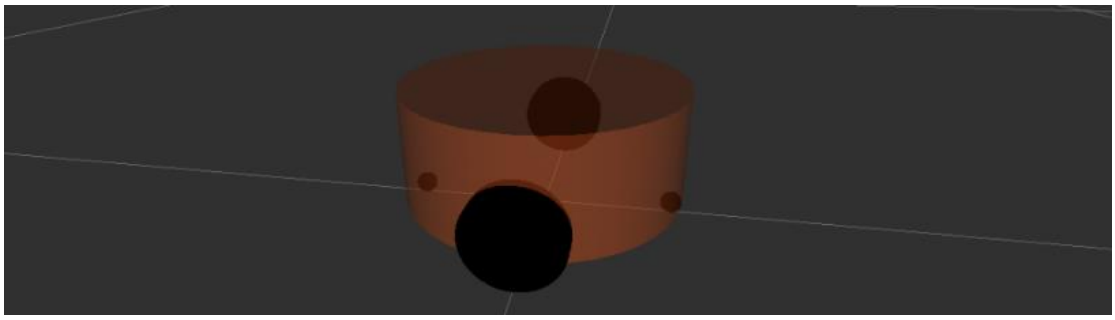

图 7　小车模型示意图

4.1.2 实现流程:

创建机器人模型可以分步骤实现：

1. 新建 urdf 文件，并与 launch 文件集成
2. 搭建底盘
3. 在底盘上添加两个驱动轮
4. 在底盘上添加两个万向轮

4.1.3 URDF 工具

在 ROS 中，提供了一些工具来方便 URDF 文件的编写，比如:

- check_urdf 命令可以检查复杂的 urdf 文件是否存在语法问题
- urdf_to_graphiz 命令可以查看 urdf 模型结构，显示不同 link 的层级关

系

当然，要使用工具之前，首先需要安装，安装命令:sudo apt install liburdfdom-tools

（1）check_urdf 语法检查

进入 urdf 文件所属目录，调用:check_urdf urdf 文件，如果不抛出异常，说明文件合法,否则非法

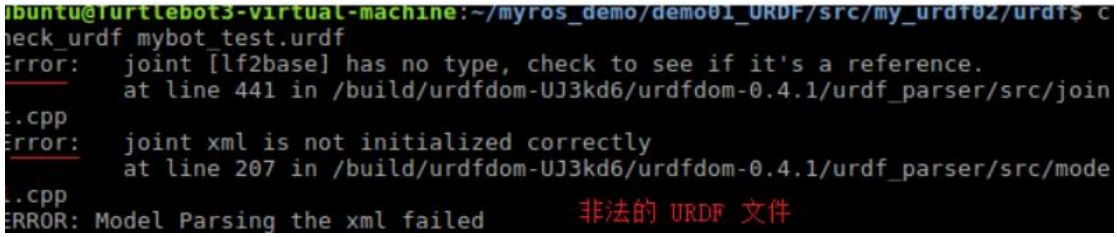

图 8　非法的 URDF 文件

图 9 合法的 URDF 文件

（2）urdf_to_graphiz 结构查看

进入 urdf 文件所属目录，调用:urdf_to_graphiz urdf 文件，当前目录下会生成 pdf 文件

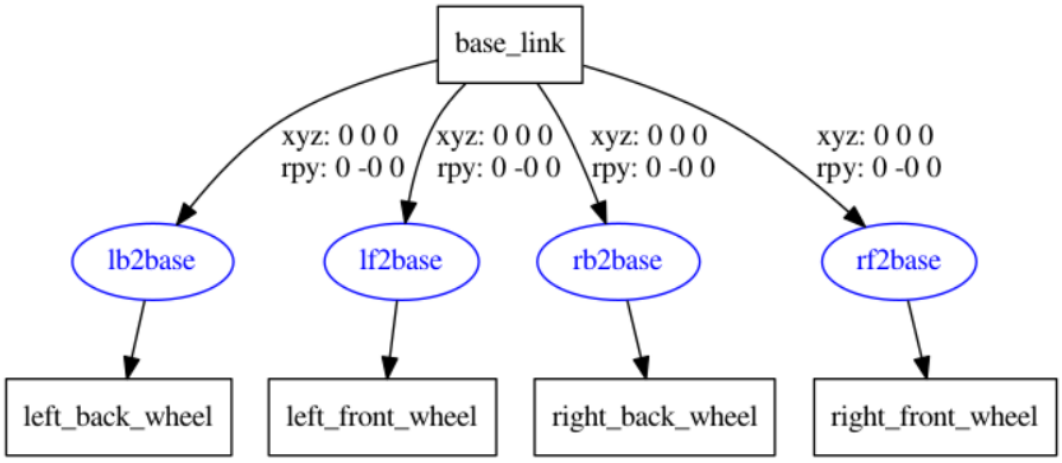

图 10　urdf 文件结构图

4.2 任务二：实现传感器感知及运动控制

4.2.1 ros 组件

（1）ros_control

ros_control:是一组软件包，它包含了控制器接口，控制器管理器，传输和硬件接口。ros_control 是一套机器人控制的中间件，是一套规范，不同的机器人平台只要按照这套规范实现，那么就可以保证 与 ROS 程序兼容，通过这套规范，实现了一种可插拔的架构设计，大大提高了程序设计的效率与灵活性。

（2）Gazebo

Gazebo 是一款 3D 动态模拟器，能够在复杂的室内和室外环境中准确有效地模拟机器人群。与游戏引擎提供高保真度的视觉模拟类似，Gazebo 提供高保真度的物理模拟，其提供一整套传感器模型，以及对用户和程序非常友好的交互方式。

如下图：

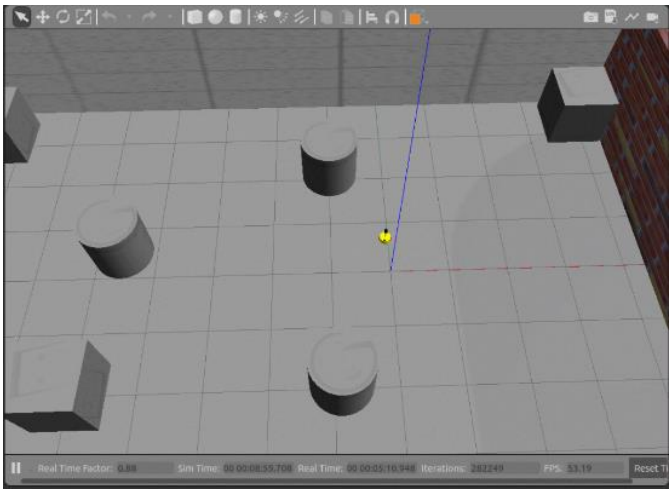

图 11　Gazebo 模拟环境

（3）Rviz

rviz 是一款三维可视化工具，很好的兼容了各种基于 ROS 软件框架的机器人平台。 在 rviz 中，可以使用 XML 对机器人、周围物体等任何实物进行尺寸、质量、位置、材质、关节等属性的描述，并且在界面中呈现出来。

如下图：

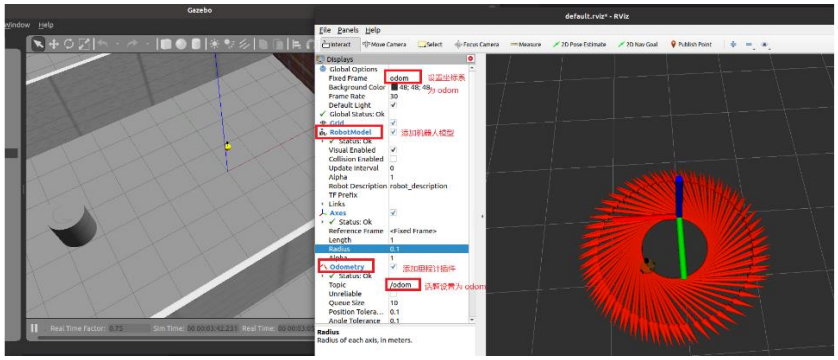

图 12　rviz 示意图

4.2.2 雷达信息仿真以及显示

通过 Gazebo 模拟激光雷达传感器，并在 Rviz 中显示激光数据。

雷达仿真基本流程:

1、已经创建完毕的机器人模型，编写一个单独的 xacro 文件，为机器人模型添加雷达配置；

2、将此文件集成进 xacro 文件；

3、启动 Gazebo，使用 Rviz 显示雷达信息。

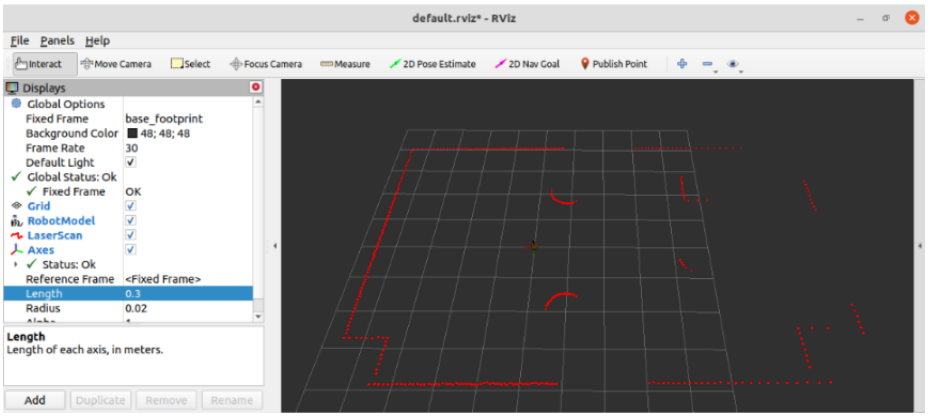

图 13　雷达信息仿真及显示

4.2.3 摄像头信息仿真以及显示

通过 Gazebo 模拟摄像头传感器，并在 Rviz 中显示摄像头数据。

摄像头仿真基本流程:

1、已经创建完毕的机器人模型，编写一个单独的 xacro 文件，为机器人模型添加摄像头配置；

2、 将此文件集成进 xacro 文件；

3、 启动 Gazebo，使用 Rviz 显示摄像头信息。

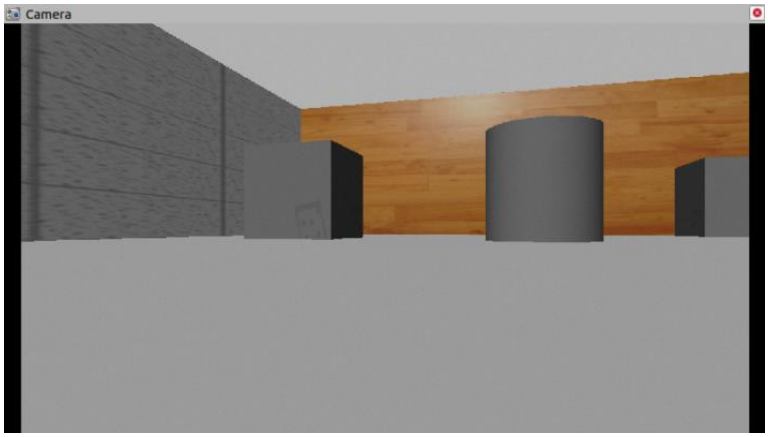

图 14　摄像头信息仿真及显示

4.2.4 kinect 信息仿真以及显示

通过 Gazebo 模拟 kinect 摄像头，并在 Rviz 中显示 kinect 摄像头数据。kinect 摄像头仿真基本流程:

1、已经创建完毕的机器人模型，编写一个单独的 xacro 文件，为机器人模型添加 kinect 摄像头配置；

2、将此文件集成进 xacro 文件；

3、启动 Gazebo，使用 Rviz 显示 kinect 摄像头信息。

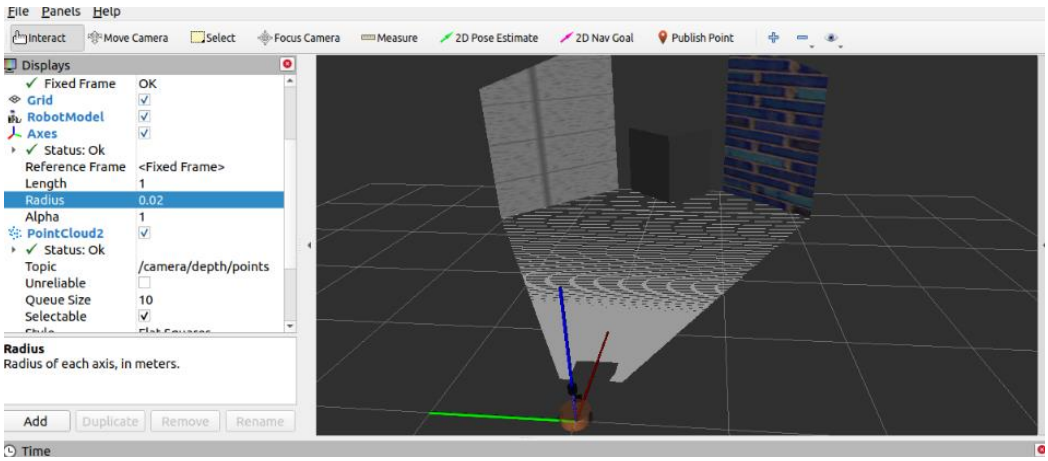

图 15　kinect 信息仿真及显示

4.2.5 小车模型运动

控制机器人模型在 rviz 中做圆周运动实现流程：

1. 安装 Arbotix
2. 创建新功能包，准备机器人 urdf、xacro 文件
3. 添加 Arbotix 配置文件
4. 编写 launch 文件配置 Arbotix
5. 启动 launch 文件并控制机器人模型运动

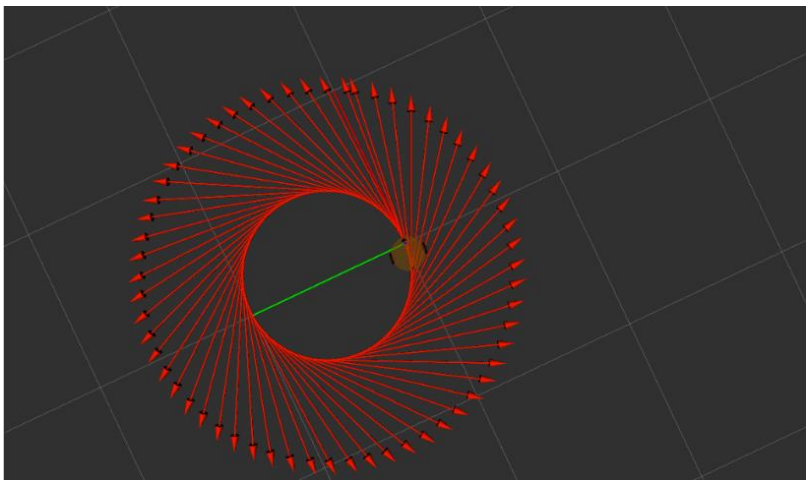

图 16　小车运动示意图

4.3 任务三：实现对小车的建图和导航

导航是机器人系统中最重要的模块之一，比如现在较为流行的服务型室内机器人，就是依赖于机器人导航来实现室内自主移动的。

即时定位与地图构建（Simultaneous Localization and Mapping，SLAM）技术可以很精确地实现环境的地图构建，定位以及多点导航。目前 SLAM 技术可以分为激光 SLAM 和视觉 SLAM，激光 SLAM 采用的传感器为激光雷达，而视觉激光则采用深度摄像头。激光 SLAM 技术较为成熟、误差少，且足以满足当前环境的使用，所以本项目采用了此项技术。

当机器人有了 SLAM 的功能，就相当于机器人有了一双眼睛。其可以扫描未知环境从而获取地图，以及实时获取自己的当前位置；可以自行路径规划到达指定地图上的地点。如智能扫地机就可以通过 SLAM 实现精准的地面避障遍历，

又自行回到充电地点。在无人驾驶以及无人机上亦同样适用，其需要实现一个共同的目的：避障、定位、路径规划。为了让机器人更加智能化，可在实现此项技术上再加上手机 APP 的控制和语音识别控制。

4.3.1 SLAM 建图

SLAM 算法有多种，当前我们选用 gmapping 建图。gmapping 是 ROS 开源社区中较为常用且比较成熟的 SLAM 算法之一，gmapping 可以根据移动机器人里程计数据和激光雷达数据来绘制二维的栅格地图。

执行步骤如下：

1) 先启动 Gazebo 仿真环境
2) 然后再启动地图绘制的 launch 文件:

   roslaunch test launch map
3) 启动键盘键盘控制节点，用于控制机器人运动建图：

   rosrun teleop_twist_keyboard teleop_twist_keyboard.py
4) 在 rviz 中添加组件，显示栅格地图
5) 最后，就可以通过键盘控制 gazebo 中的机器人运动，同时，在 rviz 中可以显示 gmapping 发布的栅格地图数据了，下一步，还需要将地图单独保存。

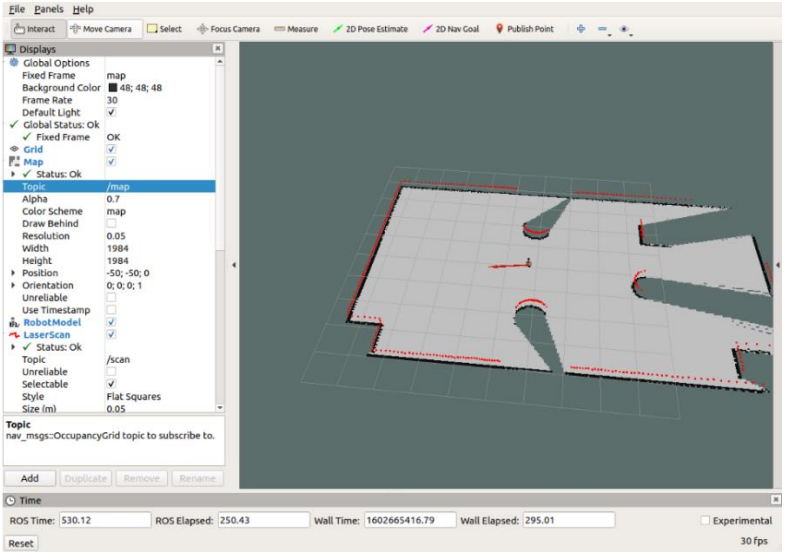

图 17　SLAM 建图

4.3.2 小车定位

定位就是推算机器人自身在全局地图中的位置，当然，SLAM 中也包含定位算法实现，不过 SLAM 的定位是用于构建全局地图的，是属于导航开始之前的阶段，而当前定位是用于导航中，导航中，机器人需要按照设定的路线运动，通过定位可以判断机器人的实际轨迹是否符合预期。在 ROS 的导航功能包集 navigation 中提供了 amcl 功能包，用于实现导航中的机器人定位。

AMCL(adaptive Monte Carlo Localization) 是用于 2D 移动机器人的概率定位系统，它实现了自适应（或 KLD 采样）蒙特卡洛定位方法，可以根据已有地图使用粒子滤波器推算机器人位置。

amcl 已经被集成到了 navigation 包，navigation 安装前面也有介绍，命令如下: sudo apt install ros-<ROS 版本>-navigation

执行步骤如下：

1) 先启动 Gazebo 仿真环境(此过程略)；
2) 启动键盘控制节点：

```
rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

3) 启动上一步中集成地图服务、amcl 与 rviz 的 launch 文件；
4) 在启动的 rviz 中，添加 RobotModel、Map 组件，分别显示机器人模型与地图，添加 posearray 插件，设置 topic 为 particlecloud 来显示 amcl 预估的当前机器人的位姿，箭头越是密集，说明当前机器人处于此位置的概率越高；
5) 通过键盘控制机器人运动，会发现 posearray 也随之而改变。

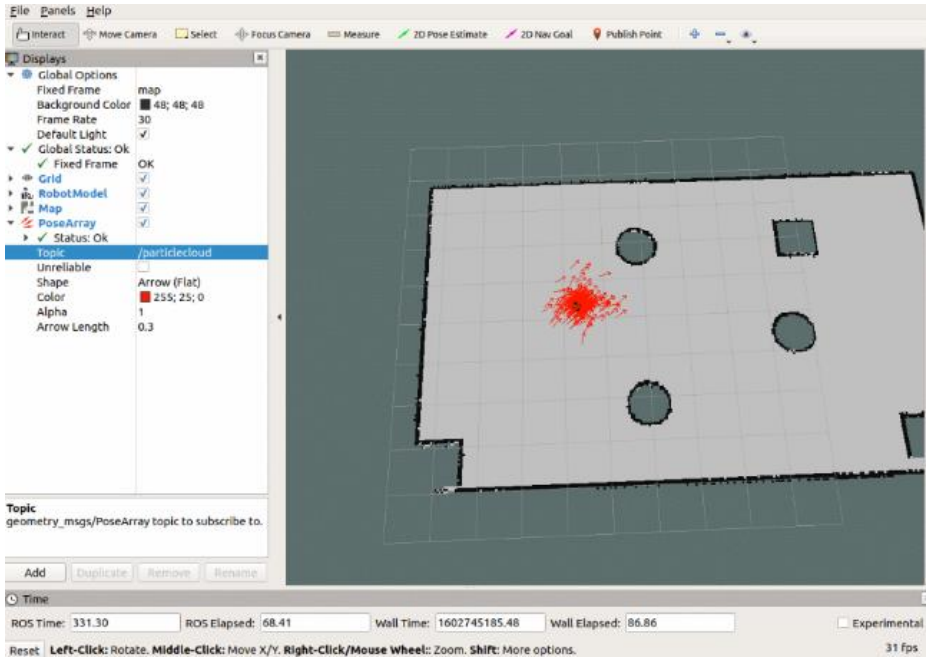

图 18　SLAM 小车定位

4.3.3 路径规划

路径规划是导航中的核心功能之一，在 ROS 的导航功能包集 navigation 中提供了 move_base 功能包，用于实现此功能。

move_base 功能包提供了基于动作(action)的路径规划实现，move_base 可以根据给定的目标点，控制机器人底盘运动至目标位置，并且在运动过程中会连续反馈机器人自身的姿态与目标点的状态信息。如前所述(7.1)move_base 主要由全局路径规划与本地路径规划组成。

move_base 已经被集成到了 navigation 包，navigation 安装前面也有介绍，命令如下: sudo apt install ros-<ROS 版本>-navigation

路径规划算法在 move_base 功能包的 move_base 节点中已经封装完毕了，但是还不可以直接调用，因为算法虽然已经封装了，但是该功能包面向的是各种类型支持 ROS 的机器人，不同类型机器人可能大小尺寸不同，传感器不同，速度不同，应用场景不同....最后可能会导致不同的路径规划结果，那么在调用路径规划节点之前，我们还需要配置机器人参数。具体实现如下:

1) 先编写 launch 文件模板
2) 编写配置文件

3) 集成导航相关的 launch 文件

4) 测试

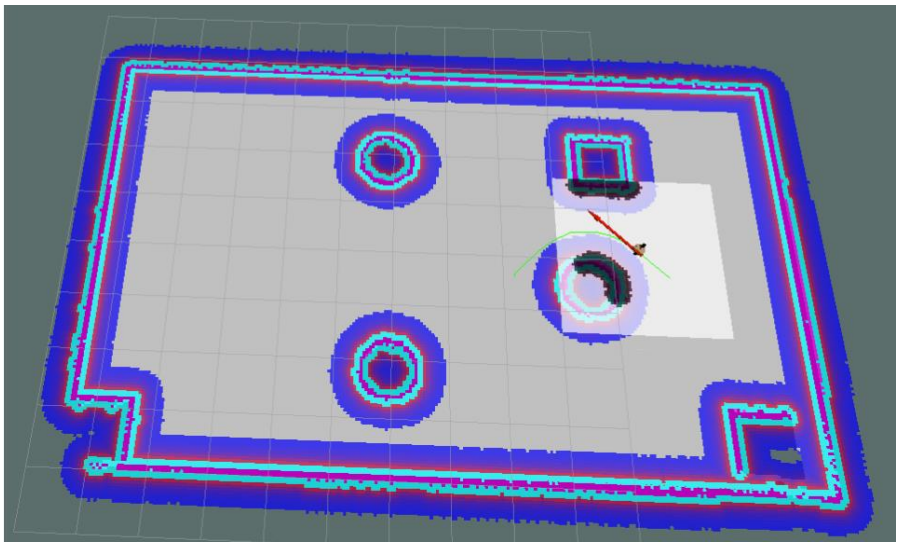

图 19 SLAM 路径规划

4.4 任务四：小车智能规避障碍方案设计

4.4.1 算法设计

移动机器人已逐渐渗透到各个领域，随着移动机器人的研究不断深入，如何让机器人避障也是一大热点和难点。本文介绍了如何使用深度学习和迁移学习的方法，只通过少量标签样本数据训练的模型能够获得较好的图像分类能力，实现障碍识别。利用迁移学习方法可以使用已有网络结构对不同领域进行处理，借助少量训练集快速训练出可用模型。基于 ROS 机器人操作系统进行避障功能的设计与实现，有利于提高软件的通用性，便于与其他功能结合，高效构建完整的机器人系统。

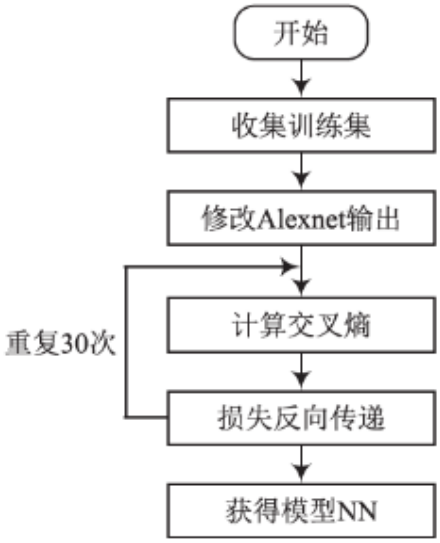

图20　SLAM路径规划

通过收集图片并将图片按照标签BLOCK和FREE进行分类，使用Python 的PyTorch 深度学习张量库训练图片分类器。为了在使用较少训练集的条件下达到较好效果，使用预处理模型AlexNet 将原先有大量标签的最后一层替换成仅有BLOCK 和FREE 的特定特征，最终得到基于卷积神经网络的图像分类器。系统架构如图所示。

摄像头采集的图像作为神经网络的输入，根据神经网络的输出结果计算移动机器人的目标线速度和角速度，并通过USB 线发送给底盘控制器，控制机器人执行避障动作。

通过深度学习训练得到基于卷积神经网络的图像分类器，使用迁移学习技术缩短分类器的训练时间，基于ROS操作系统进行避障功能的设计与实现。分析预设概率值对避障效果的影响，在合适的线速度、角速度与预设概率值的条件下测得避障成功率约为93%。

4.4.2 实验步骤

本智能方案主要由collision_avoidance、mobile_base_nodelet_managet、mobile_base 和diagnostic_aggregator 节点组成。collision_avoidance 节点负责获取实时图像、调用神经网络处理图像和发送运动控制指令信息给

mobile_base_nodelet_managet 节点。mobile_base_nodelet_managet 节点负责解决ROS 数据交互存在的延时和堵塞问题，可实现多路数据交互，把信息发送到 mobile_base 节点。mobile_base 节点是控制机器人运动的关键，diagnostic_aggregator 节点是诊断节点。关键节点如图所示。

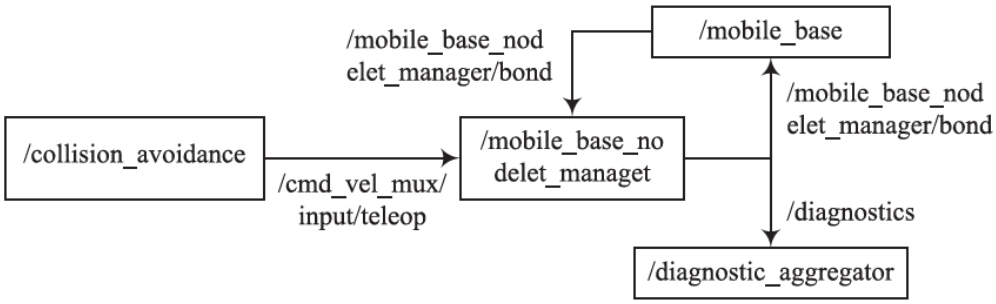

图 21　关键节点图

collision_avoidance 节点和 mobile_base_nodelet_managet 节点间使用 cmd_vel_mux 话题通信，消息内容是移动机器人的目标线速度和角速度。collision_avoidance 节点发布消息，mobile_base_nodelet_managet 节点读取消息，并根据优先级分配电机运行权限，保证多路控制时系统的稳定性。优先级和超时时间可通过修改YAML文件相关参数设置。

cmd_vel_mux 话题采用Twist 类型的消息格式，它是用于控制电机的向量结构体，由线速度和角速度组成，可利用笛卡尔直角坐标系实现精确表达，采用Twist 类型消息格式，其数据类型均为float 64 型。collision_avoidance 节点将移动机器人的目标线速度和角速度封装为Twist 类型消息，发布到 cmd_vel_mux 话题中。mobile_base_nodelet_managet 节点从话题队列中取出电机控制消息并按优先级依次执行。如图所示为对应的程序流程。

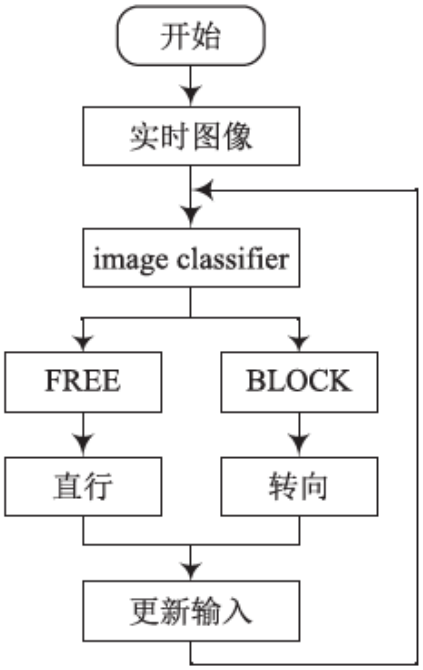

图22　小车运动流程图

4.4.3 系统测试

神经网络会返回 1 个表示移动机器人是否受阻的概率值。当返回的概率值大于预设概率值时，表示移动机器人受阻，移动机器人将左转弯直到避开障碍物，否则继续前进。预设概率值将对避障性能产生直接影响。本测试设定角速度为 1 rad/s，线速度为 0.2 m/s，通过改变预设概率值研究在特定角速度和线速度下预设概率值对移动机器人避障性能的影响，测试结果见下表所列。

表 1　不同概率预设值对避障距离的影响

预设概率值	触发距离 /cm	边　距 /cm
≥ 0.8	发生混乱	发生混乱
0.7	41	10
0.6	40	10
0.5	38	9
0.4	41	9
0.3	45	10
0.2	45	13
0.1	50	17

在一定范围内，随着预设概率值递增，移动机器人沿着墙壁走直线和触发避障的距离都随之变小。设定不同的预设概率值，移动机器人沿墙壁直线运动所需的最小边距不同，触发避障的距离也不同。根据具体应用场景和应用需求对预设概率值做适当调整即可满足不同的用户需求。

过低的预设概率值可能使移动机器人在离障碍物较远时开始转向；而过高的预设概率值会使移动机器人在过障碍物时因物体只占识别图像的某一部分，达不到预设概率值而认为前方无障碍并发生碰撞。此外，过高的预设概率值会使移动机器人在触发避障的临界点抖动加剧。因此根据移动机器人的外形设定合适的预设概率值尤为重要。通过多次测试，测得最适合移动机器人的预设概率值范围为0.4~0.5，避障成功率约为93%，避障效果较好。

小车在gazebo仿真环境避障效果如下图所示

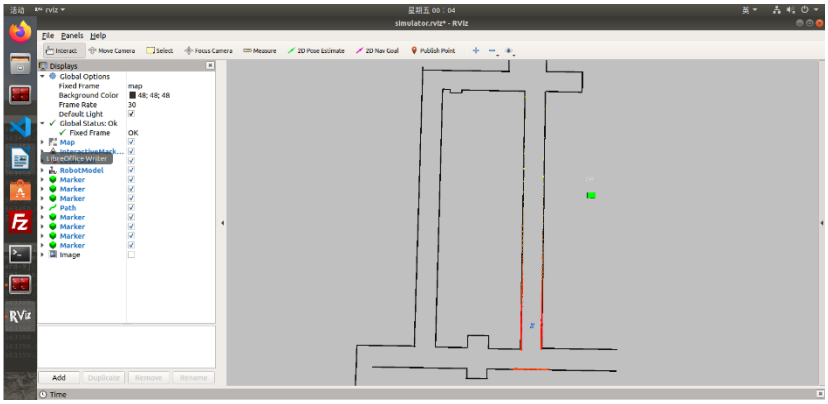

图 23　小车运动避障示意图

5 总体感悟

通过此次的ROS课程设计制作，使我们更加认识到了动手能力和理论知识的重要性，而理论与实践的结合更是重中之重。当然，我们也深刻地认识到我们的不足，由于自身理论知识的欠缺和动手能力的不佳在工作中频频受阻，走了好多弯路，虽然在制作过程中不可避免地遇到很多问题，但是最后还是圆满解决了这些问题，实现了整个系统设计与最后调试，相关指标达到预期的要求，很好地完成了本次设计任务。

为实现视觉避障功能，文中设计了一种基于ROS 和深度学习的智能避障

移动机器人。本系统使用深度学习训练得到基于卷积神经网络的图像分类器，使用迁移学习技术缩短分类器的训练时间，基于ROS 操作系统进行避障功能的设计与实现，并快速构建扩展性强、性能稳定的机器人系统。系统测试结果表明，本系统具有较好的避障能力，对不同环境和障碍物均有良好的适应性。

6 参考文献

[1]自主移动机器人室内定位方法研究综述[J]. 高云峰,周伦,吕明睿,刘文涛. 传感器与微系统. 2013(12)

[2]小型室内机器人定位技术的研究与应用[J]. 陈立钢. 绿色环保建材. 2020(07)

[3]基于 ROS 系统移动机器人 SLAM 算法的研究与实现[J]. 李贺喜,李富强,牛童立,李康,杜边境. 技术与市场. 2020(07)

7 附录：

（1） 小车 urdf 模型

```
    <link name="front_wheel">
        <visual>
            <geometry>
                <sphere radius="0.0075" />
            </geometry>
            <origin xyz="0 0 0" rpy="0 0 0" />
            <material name="black">
                <color rgba="0.0 0.0 0.0 1.0" />
            </material>
        </visual>
    </link>

    <joint name="front_wheel2base_link" type="continuous">
        <parent link="base_link" />
        <child link="front_wheel" />
        <origin xyz="0.0925 0 -0.0475" />
        <axis xyz="1 1 1" />
    </joint>

    <link name="back_wheel">
        <visual>
            <geometry>
                <sphere radius="0.0075" />
            </geometry>
            <origin xyz="0 0 0" rpy="0 0 0" />
            <material name="black">
                <color rgba="0.0 0.0 0.0 1.0" />
            </material>
        </visual>
    </link>

    <joint name="back_wheel2base_link" type="continuous">
        <parent link="base_link" />
        <child link="back_wheel" />
        <origin xyz="-0.0925 0 -0.0475" />
        <axis xyz="1 1 1" />
</joint>
```

（2） 小车运动

```
<robot name="my_car_move" xmlns:xacro="http://wiki.ros.org/xacro">

    <!-- 传动实现:用于连接控制器与关节 -->
    <xacro:macro name="joint_trans" params="joint_name">
        <!-- Transmission is important to link the joints and the
controller -->
        <transmission name="${joint_name}_trans">
            <type>transmission_interface/SimpleTransmission</type>
            <joint name="${joint_name}">

<hardwareInterface>hardware_interface/VelocityJointInterface</hardwareInte
rface>
            </joint>
            <actuator name="${joint_name}_motor">

<hardwareInterface>hardware_interface/VelocityJointInterface</hardwareInte
rface>
                <mechanicalReduction>1</mechanicalReduction>
            </actuator>
        </transmission>
    </xacro:macro>

    <!-- 每一个驱动轮都需要配置传动装置 -->
    <xacro:joint_trans joint_name="left_wheel2base_link" />
    <xacro:joint_trans joint_name="right_wheel2base_link" />

    <!-- 控制器 -->
    <gazebo>
        <plugin name="differential_drive_controller"
filename="libgazebo_ros_diff_drive.so">
            <rosDebugLevel>Debug</rosDebugLevel>
            <publishWheelTF>true</publishWheelTF>
            <robotNamespace>/</robotNamespace>
            <publishTf>1</publishTf>
            <publishWheelJointState>true</publishWheelJointState>
            <alwaysOn>true</alwaysOn>
            <updateRate>100.0</updateRate>
            <legacyMode>true</legacyMode>
            <leftJoint>left_wheel2base_link</leftJoint> <!-- 左轮
-->
```

```
                    <rightJoint>right_wheel2base_link</rightJoint> <!-- 右
轮 -->
                    <wheelSeparation>${base_link_radius *
2}</wheelSeparation> <!-- 车轮间距 -->
                    <wheelDiameter>${wheel_radius * 2}</wheelDiameter>
<!-- 车轮直径 -->
                    <broadcastTF>1</broadcastTF>
                    <wheelTorque>30</wheelTorque>
                    <wheelAcceleration>1.8</wheelAcceleration>
                    <commandTopic>cmd_vel</commandTopic> <!-- 运动
控制话题 -->
                    <odometryFrame>odom</odometryFrame>
                    <odometryTopic>odom</odometryTopic> <!-- 里程计话
题 -->
                    <robotBaseFrame>base_footprint</robotBaseFrame> <!--
根坐标系 -->
                </plugin>
            </gazebo>

</robot>
```

（3） 雷达信息仿真以及显示

```
    <robot name="my_sensors" xmlns:xacro="http://wiki.ros.org/xacro">

        <!-- 雷达 -->
        <gazebo reference="laser">
          <sensor type="ray" name="rplidar">
            <pose>0 0 0 0 0 0</pose>
            <visualize>true</visualize>
            <update_rate>5.5</update_rate>
            <ray>
              <scan>
                <horizontal>
                  <samples>360</samples>
                  <resolution>1</resolution>
                  <min_angle>-3</min_angle>
                  <max_angle>3</max_angle>
                </horizontal>
              </scan>
              <range>
                <min>0.10</min>
```

```
                    <max>30.0</max>
                    <resolution>0.01</resolution>
                  </range>
                  <noise>
                    <type>gaussian</type>
                    <mean>0.0</mean>
                    <stddev>0.01</stddev>
                  </noise>
                </ray>
                <plugin name="gazebo_rplidar"
        filename="libgazebo_ros_laser.so">
                    <topicName>/scan</topicName>
                    <frameName>laser</frameName>
                </plugin>
              </sensor>
            </gazebo>

        </robot>
```

（4）Gazebo 仿真摄像头

```
<robot name="my_sensors" xmlns:xacro="http://wiki.ros.org/xacro">
  <!-- 被引用的 link -->
  <gazebo reference="camera">
    <!-- 类型设置为 camara -->
    <sensor type="camera" name="camera_node">
      <update_rate>30.0</update_rate> <!-- 更新频率 -->
      <!-- 摄像头基本信息设置 -->
      <camera name="head">
        <horizontal_fov>1.3962634</horizontal_fov>
        <image>
          <width>1280</width>
          <height>720</height>
          <format>R8G8B8</format>
        </image>
        <clip>
          <near>0.02</near>
          <far>300</far>
        </clip>
        <noise>
          <type>gaussian</type>
          <mean>0.0</mean>
```

```
            <stddev>0.007</stddev>
          </noise>
        </camera>
        <!-- 核心插件 -->
        <plugin name="gazebo_camera" filename="libgazebo_ros_camera.so">
          <alwaysOn>true</alwaysOn>
          <updateRate>0.0</updateRate>
          <cameraName>/camera</cameraName>
          <imageTopicName>image_raw</imageTopicName>
          <cameraInfoTopicName>camera_info</cameraInfoTopicName>
          <frameName>camera</frameName>
          <hackBaseline>0.07</hackBaseline>
          <distortionK1>0.0</distortionK1>
          <distortionK2>0.0</distortionK2>
          <distortionK3>0.0</distortionK3>
          <distortionT1>0.0</distortionT1>
          <distortionT2>0.0</distortionT2>
        </plugin>
      </sensor>
    </gazebo>
</robot>
```

（5）SLAM 建图

```
<launch>
<param name="use_sim_time" value="true"/>
      <node pkg="gmapping" type="slam_gmapping" name="slam_gmapping"
output="screen">
        <remap from="scan" to="scan"/>
        <param name="base_frame" value="base_footprint"/><!--底盘坐标系-->
        <param name="odom_frame" value="odom"/> <!--里程计坐标系-->
        <param name="map_update_interval" value="5.0"/>
        <param name="maxUrange" value="16.0"/>
        <param name="sigma" value="0.05"/>
        <param name="kernelSize" value="1"/>
        <param name="lstep" value="0.05"/>
        <param name="astep" value="0.05"/>
        <param name="iterations" value="5"/>
        <param name="lsigma" value="0.075"/>
        <param name="ogain" value="3.0"/>
        <param name="lskip" value="0"/>
        <param name="srr" value="0.1"/>
```

```
        <param name="srt" value="0.2"/>
        <param name="str" value="0.1"/>
        <param name="stt" value="0.2"/>
        <param name="linearUpdate" value="1.0"/>
        <param name="angularUpdate" value="0.5"/>
        <param name="temporalUpdate" value="3.0"/>
        <param name="resampleThreshold" value="0.5"/>
        <param name="particles" value="30"/>
        <param name="xmin" value="-50.0"/>
        <param name="ymin" value="-50.0"/>
        <param name="xmax" value="50.0"/>
        <param name="ymax" value="50.0"/>
        <param name="delta" value="0.05"/>
        <param name="llsamplerange" value="0.01"/>
        <param name="llsamplestep" value="0.01"/>
        <param name="lasamplerange" value="0.005"/>
        <param name="lasamplestep" value="0.005"/>
    </node>

    <node pkg="joint_state_publisher" name="joint_state_publisher"
type="joint_state_publisher" />
    <node pkg="robot_state_publisher" name="robot_state_publisher"
type="robot_state_publisher" />

    <node pkg="rviz" type="rviz" name="rviz" />
    <!-- 可以保存 rviz 配置并后期直接使用-->
    <!--
    <node pkg="rviz" type="rviz" name="rviz" args="-d $(find
my_nav_sum)/rviz/gmapping.rviz"/>
    -->
</launch>
```

（6）SLAM 定位

```
<launch>
<node pkg="amcl" type="amcl" name="amcl" output="screen">
  <!-- Publish scans from best pose at a max of 10 Hz -->
  <param name="odom_model_type" value="diff"/><!-- 里程计模式为差分 -->
  <param name="odom_alpha5" value="0.1"/>
  <param name="transform_tolerance" value="0.2" />
  <param name="gui_publish_rate" value="10.0"/>
  <param name="laser_max_beams" value="30"/>
```

```
  <param name="min_particles" value="500"/>
  <param name="max_particles" value="5000"/>
  <param name="kld_err" value="0.05"/>
  <param name="kld_z" value="0.99"/>
  <param name="odom_alpha1" value="0.2"/>
  <param name="odom_alpha2" value="0.2"/>
  <!-- translation std dev, m -->
  <param name="odom_alpha3" value="0.8"/>
  <param name="odom_alpha4" value="0.2"/>
  <param name="laser_z_hit" value="0.5"/>
  <param name="laser_z_short" value="0.05"/>
  <param name="laser_z_max" value="0.05"/>
  <param name="laser_z_rand" value="0.5"/>
  <param name="laser_sigma_hit" value="0.2"/>
  <param name="laser_lambda_short" value="0.1"/>
  <param name="laser_lambda_short" value="0.1"/>
  <param name="laser_model_type" value="likelihood_field"/>
  <!-- <param name="laser_model_type" value="beam"/> -->
  <param name="laser_likelihood_max_dist" value="2.0"/>
  <param name="update_min_d" value="0.2"/>
  <param name="update_min_a" value="0.5"/>

  <param name="odom_frame_id" value="odom"/><!-- 里程计坐标系 -->
  <param name="base_frame_id" value="base_footprint"/><!-- 添加机器人基坐标
系 -->
  <param name="global_frame_id" value="map"/><!-- 添加地图坐标系 -->

  <param name="resample_interval" value="1"/>
  <param name="transform_tolerance" value="0.1"/>
  <param name="recovery_alpha_slow" value="0.0"/>
  <param name="recovery_alpha_fast" value="0.0"/>
</node>
</launch>
```

（7）路径规划

```
<launch>
    <!-- 设置地图的配置文件 -->
    <arg name="map" default="nav.yaml" />
    <!-- 运行地图服务器，并且加载设置的地图-->
    <node name="map_server" pkg="map_server" type="map_server" args="$(find
mycar_nav)/map/$(arg map)"/>
```

```
    <!-- 启动 AMCL 节点 -->
    <include file="$(find mycar_nav)/launch/amcl.launch" />

    <!-- 运行 move_base 节点 -->
    <include file="$(find mycar_nav)/launch/path.launch" />
    <!-- 运行 rviz -->
    <node pkg="rviz" type="rviz" name="rviz" args="-d $(find
mycar_nav)/rviz/nav.rviz" />

</launch>
```

（8）小车避障

A：

```
#超声波测距函数
def Distance_Ultrasound():
    GPIO.output(TRIG,GPIO.LOW)       #输出口初始化置 LOW（不发射）
    time.sleep(0.000002)
    GPIO.output(TRIG,GPIO.HIGH)      #发射超声波
    time.sleep(0.00001)
    GPIO.output(TRIG,GPIO.LOW)       #停止发射超声波
    while GPIO.input(ECHO) == 0:
        emitTime = time.time()       #记录发射时间
    while GPIO.input(ECHO) == 1:
        acceptTime = time.time()     #记录接收时间
    totalTime = acceptTime - emitTime                    #计算总时间
    distanceReturn = totalTime * 340 / 2 * 100     #计算距离（单位：cm）
    return   distanceReturn              #返回距离

#避障函数
def Obstacle_Avoidance():
    while True:
        dis= Distance_Ultrasound()
        print("距离 ",dis,"cm")
        if dis<30:                       #距离小于 30cm 时启动避障程序
            while dis<30:
                Back_time(0.5)           #距离小于 30cm 时后退 0.5s
                dis=Distance_Ultrasound()
                print("距离 ",dis,"cm")
            Left_time(1.5)               #左转 1.5s
            Forward()                    #继续前进
        time.sleep(0.5)
```

B：

```
import RPi.GPIO as GPIO          #引入 RPi.GPIO 库函数命名为 GPIO
import time                      #引入计时 time 函数

GPIO.setwarnings(False)

GPIO.setmode(GPIO.BCM)           #将 GPIO 编程方式设置为 BCM 模式，基于插座引脚编号

#接口定义
TRIG = 5                         #将超声波模块 TRIG 口连接到树莓派 Pin29
ECHO = 6                          #将超声波模块 ECHO 口连接到树莓派 Pin31
INT1 = 17                        #将 L298 INT1 口连接到树莓派 Pin11
INT2 = 18                        #将 L298 INT2 口连接到树莓派 Pin12
INT3 = 27                        #将 L298 INT3 口连接到树莓派 Pin13
INT4 = 22                        #将 L298 INT4 口连接到树莓派 Pin15

#输出模式
GPIO.setup(TRIG,GPIO.OUT)
GPIO.setup(ECHO,GPIO.IN)
GPIO.setup(INT1,GPIO.OUT)
GPIO.setup(INT2,GPIO.OUT)
GPIO.setup(INT3,GPIO.OUT)
GPIO.setup(INT4,GPIO.OUT)

#一直前进函数
def Forward():
    GPIO.output(INT1,GPIO.LOW)
    GPIO.output(INT2,GPIO.HIGH)
    GPIO.output(INT3,GPIO.LOW)
    GPIO.output(INT4,GPIO.HIGH)

#后退指定时间函数
def Back_time(time_sleep):
    GPIO.output(INT1,GPIO.HIGH)
    GPIO.output(INT2,GPIO.LOW)
    GPIO.output(INT3,GPIO.HIGH)
    GPIO.output(INT4,GPIO.LOW)
    time.sleep(time_sleep)

#左转指定时间函数
```

```
def Left_time(time_sleep):
    GPIO.output(INT1,GPIO.LOW)
    GPIO.output(INT2,GPIO.LOW)
    GPIO.output(INT3,GPIO.LOW)
    GPIO.output(INT4,GPIO.HIGH)
    time.sleep(time_sleep)

#停止函数
def Stop():
    GPIO.output(INT1,GPIO.LOW)
    GPIO.output(INT2,GPIO.LOW)
    GPIO.output(INT3,GPIO.LOW)
    GPIO.output(INT4,GPIO.LOW)

#超声波测距函数
def Distance_Ultrasound():
    GPIO.output(TRIG,GPIO.LOW)      #输出口初始化置 LOW（不发射）
    time.sleep(0.000002)
    GPIO.output(TRIG,GPIO.HIGH)     #发射超声波
    time.sleep(0.00001)
    GPIO.output(TRIG,GPIO.LOW)      #停止发射超声波
    while GPIO.input(ECHO) == 0:
        emitTime = time.time()      #记录发射时间
    while GPIO.input(ECHO) == 1:
        acceptTime = time.time()    #记录接收时间
    totalTime = acceptTime - emitTime          #计算总时间
    distanceReturn = totalTime * 340 / 2 * 100     #计算距离（单位：cm）
    return   distanceReturn             #返回距离

#避障函数
def Obstacle_Avoidance():
    while True:
        dis= Distance_Ultrasound()
        print("距离  ",dis,"cm")
            if dis<30:                  #距离小于 30cm 时启动避障程序
              while dis<30:
                  Back_time(0.5)        #距离小于 30cm 时后退 0.5s
                  dis=Distance_Ultrasound()
                  print("距离  ",dis,"cm")
              Left_time(1.5)            #左转 1.5s
              Forward()                 #继续前进
```

```
        time.sleep(0.5)

print("超声波避障系统运行中，按 Ctrl+C 退出...")
        try:
            Forward()                   #初始状态为前进
            Obstacle_Avoidance()
        except KeyboardInterrupt:
            Stop()
```