机器人操作系统

11912060110

11912060110

目录

1. 介绍

1.1 系统环境

- 虚拟机：VMware 16.x
- 操作系统：ubuntu 18.04
- 机器人操作系统：melodic

1.2

1.2.1 安装

注册码网上搜

1.3

1.3.1 安装

1.3.2 换源

打开类似 软件和更新 的应用，选择国内源

1.3.3 安装增强工具(虚拟机)

```
sudo apt install -y open-vm-tools open-vm-tools-desktop
```

1.4

1.4.1 配置Ubuntu软件仓库

配置你的Ubuntu软件仓库（repositories）以允许使用“restricted”“universe”和“multiverse”存储库。你可以根据Ubuntu软件仓库指南来完成这项工作。

1.4.2 设置sources.list

```
sudo sh -c '. /etc/lsb-release && echo "deb http://mirrors.tuna.tsinghua.edu.cn/ros/ubuntu/ `lsb_release -cs` main" > /etc/apt/sources.list.d/ros-latest.list'
```

1.4.3 设置密钥

```
sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

1.4.4 安装

首先更新软件源

```
sudo apt update
```

然后根据需求安装机器人操作系统

- 桌面完整版（推荐）：：包含 ROS、rqt、rviz、机器人通用库、2D/3D 模拟器、导航以及 2D/3D 感知包。

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

- 桌面版：包含 ROS，rqt，rviz 和机器人通用库

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

- ROS-基础包：包含 ROS 包，构建和通信库。没有图形界面工具。

```
sudo apt install ros-melodic-ros-base
```

- 单独的包：你也可以安装某个指定的ROS软件包（使用软件包名称替换掉下面的PACKAGE）：

```
sudo apt install ros-melodic-PACKAGE
```

1.4.5 初始化 rosdep

安装国内版 `rosdep`

```
sudo apt install -y python3-pip
pip3 install rosdepc -i https://pypi.mirrors.ustc.edu.cn/simple
```

执行初始化

```
sudo rosdepc init
rosdepc update
```

1.4.6 设置环境

- bash

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

- zsh

```
echo "source /opt/ros/melodic/setup.zsh" >> ~/.zshrc
source ~/.zshrc
```

1.4.7 安装开发依赖

```
sudo apt-get install python-rosinstall python-rosinstall-generator python-wstool build-essential
```

最后更新: 2021-12-05

2. 内容

2.1 实验一

2.1.1 通过键盘控制小乌龟运动实验过程与结果截图

1. 打开一个新终端，启动主节点

```
roscore
```

2. 打开一个新终端，启动 `turtlesim` 节点

```
rosrun turtlesim turtlesim_node
```

3. 打开一个新终端，运行按键控制

```
rosrun turtlesim turtle_teleop_key
```

2.1.2 通过 `rostopic pub` 命令控制小乌龟运动过程与结果截图

列出所有发布(Published)和订阅(Subscribed)的主题及其类型的详细信息

```
rostopic list -v
```

将得到以下信息

```
Published topics:
 * /turtle1/color_sensor [turtlesim/Color] 1 publisher
 * /rosout [rosgraph_msgs/Log] 1 publisher
 * /rosout_agg [rosgraph_msgs/Log] 1 publisher
 * /turtle1/pose [turtlesim/Pose] 1 publisher
Subscribed topics:
 * /turtle1/cmd_vel [geometry_msgs/Twist] 1 subscriber
 * /rosout [rosgraph_msgs/Log] 1 subscribe
```

我们可以得知 `* 话题名称 [消息类型] 发布或订阅者的数量`

使用 `rosmsg` 查看消息的详细信息

```
rosmsg show geometry_msgs/Twist
```

将得到以下信息

```
geometry_msgs/Vector3 linear
    float64 x
    float64 y
    float64 z
geometry_msgs/Vector3 angular
    float64 x
    float64 y
    float64 z
```

我们需要传递三个线速度和三个角速度

```
rostopic pub -1 /turtle1/cmd_vel geometry_msgs/Twist -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'
```

以上命令会发送一条消息给 `turtlesim`，告诉它以 `2.0` 大小的线速度和 `1.8` 大小的角速度移动。

- 这条命令将消息发布到指定的话题：

```
rostopic pub
```

- 这一选项会让rostopic只发布一条消息，然后退出：

```
-1
```

- 这是要发布到的话题的名称：

```
/turtle1/cmd_vel
```

- 这是发布到话题时要使用的消息的类型：

```
geometry_msgs/Twist
```

- 这一选项（两个破折号）用来告诉选项解析器，表明之后的参数都不是选项。如果参数前有破折号（-）比如负数，那么这是必需的。

```
--
```

- 如前所述，一个turtlesim/Velocity消息有两个浮点型元素：linear和angular。在本例中，'[2.0, 0.0, 0.0]'表示linear的值为x=2.0, y=0.0, z=0.0，而'[0.0, 0.0, 1.8]'是说angular的值为x=0.0, y=0.0, z=1.8。这些参数实际上使用的是YAML语法，在YAML命令行文档中有描述。

```
'[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'
```

```
rostopic pub /turtle1/cmd_vel geometry_msgs/Twist -r 1 -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, -1.8]'
```

这将以1 Hz的速度发布velocity指令到velocity话题上。

画一个正方形

```
rosservice call turtle1/teleport_relative -- 2 0
rosservice call turtle1/teleport_relative -- 0 1.57075
rosservice call turtle1/teleport_relative -- 2 0
rosservice call turtle1/teleport_relative -- 0 1.57075
rosservice call turtle1/teleport_relative -- 2 0
rosservice call turtle1/teleport_relative -- 0 1.57075
rosservice call turtle1/teleport_relative -- 2 0
```

2.1.3 通过键盘控制omove移动机器人的运动实验过程与结果截图

1. 移动机器人的NUC控制器插上显示屏和键盘鼠标，开机后连接路由器wifi或手机热点；
2. 通过 `ifconfig` 或者 `ip addr` 命令，查看当前控制器的IP地址
3. 保持机器人开机状态，拆除显示屏和键鼠。实验过程中不要关闭wifi热点，保持网络顺畅。
4. 在自己电脑上通过命令 `ssh omove@$ip` 远程登录到控制器，`$ip` 指第2步查询的IP地址，提示输入密码，输入密码 `1`，回车确认。
5. 在控制器中，启动 `omove_driver` 功能包中的 `omove_driver.launch` 文件。

```
roslaunch omove_driver omove_driver.launch
```

6. 在控制器中，启动键盘控制节点 `teleop_twist_keyboard` 通过键盘按键控制移动机器人运动。

```
rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

2.1.4 通过 `rostopic pub` 命令控制omove移动机器人运动过程与结果截图

通过 `rostopic pub` 命令的方式控制移动机器人运动。

```

```

最后更新: 2021-12-05

2.2 实验二

2.2.1 ROS工作空间与环境管理

```
mkdir -p ~/catkin_ws/src      #创建工作空间catkin_ws和子目录src，自定义空间名
cd ~/catkin_ws                #进入到工作空间catkin_ws
catkin_make                   #编译工作空间catkin_ws
```

2.2.2 ROS程序包创建

```
cd ~/catkin_ws/src                          #进入程序包目录
catkin_create_pkg lab std_msgs rospy roscpp #创建程序包lab
cd lab                                      #进入程序包lab
```

2.2.3 ROS编程话题发布与订阅的代码和分析

```
mkdir scripts
touch scripts/talker.py scripts/listener.py    # 创建空白文件
chmod +x scripts/talker.py scripts/listener.py # 添加执行权限
```

编辑 `scripts/talker.py`

```
gedit scripts/talker.py
```

scripts/talker.py

```
#!/usr/bin/env python
# license removed for brevity
import rospy
from std_msgs.msg import String
def talker():
    pub = rospy.Publisher('chatter', String, queue_size=10)
    rospy.init_node('talker', anonymous=True)
    rate = rospy.Rate(10)  # 10hz
    while not rospy.is_shutdown():
        hello_str = "hello world %s" % rospy.get_time()
        rospy.loginfo(hello_str)
        pub.publish(hello_str)
        rate.sleep()
if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

编辑 `scripts/listener.py`

```
gedit scripts/listener.py
```

scripts/listener.py

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import String
def callback(data):
    rospy.loginfo(rospy.get_caller_id() + "I heard %s", data.data)
def listener():

    # In ROS, nodes are uniquely named. If two nodes with the same
    # name are launched, the previous one is kicked off. The
    # anonymous=True flag means that rospy will choose a unique
    # name for our 'listener' node so that multiple listeners can
    # run simultaneously.
    rospy.init_node('listener', anonymous=True)
    rospy.Subscriber("chatter", String, callback)
    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()
if __name__ == '__main__':
    listener()
```

编辑 `CMakeLists.txt`

```
gedit CMakeLists.txt
```

在文件末尾添加以下内容

CMakeLists.txt

```
catkin_install_python(PROGRAMS
  scripts/talker.py
  scripts/listener.py
  DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
)
```

编译工程

```
cd ~/catkin_ws
catkin_make
```

加载环境

```
source devel/setup.bash
```

运行主节点

```
roscore
```

运行订阅者

```
cd ~/catkin_ws
source devel/setup.bash
rosrun lab listener.py
```

运行发布者

```
cd ~/catkin_ws
source devel/setup.bash
rosrun lab talker.py
```

2.2.4 编程控制omove移动机器人自主运动

```
touch scripts/run.py
chmod +x scripts/run.py
```

scripts/run.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import rospy
from geometry_msgs.msg import Twist
def run():
    pub = rospy.Publisher('cmd_vel', Twist, queue_size=10)
    rospy.init_node('run', anonymous=True)
    rate = rospy.Rate(10)  # 10hz
    count = 0
    while not rospy.is_shutdown():
        twist = Twist()
        if 0 == count:
            rospy.loginfo("前进")
        elif 20 == count:
            rospy.loginfo("左移")
        elif 40 == count:
            rospy.loginfo("后退")
        elif 60 == count:
            rospy.loginfo("右移")
        if count <= 20:
            twist.linear.x = 1
        elif 20 < count and count <= 40:
            twist.linear.y = 1
        elif 40 < count and count <= 60:
            twist.linear.x = -1
        elif 60 < count and count <= 80:
            twist.linear.y = -1
        else:
            count = -1
        pub.publish(twist)
        rate.sleep()
        count += 1
if __name__ == '__main__':
```

```
    try:
        run()
    except rospy.ROSInterruptException:
        pass
```

CMakeLists.txt

```
catkin_install_python(PROGRAMS
  scripts/run.py
  scripts/talker.py
  scripts/listener.py
  DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
)
```

上传代码

```
scp ~/catkin_ws/src/lib omove@$ip:~/omve_ws/src
```

连接控制器然后编译

```
ssh omove@$ip
cd ~/omove_ws
catkin_make
```

启动驱动节点

```
roslaunch omove_driver omove_driver.launch
```

运行程序

```
rosrun lab run.py
```

最后更新: 2021-12-05

2.3 实验三

2.3.1 笔记本摄像头和omove移动机器人视觉获取图像步骤和截图

装usb_cam程序包

```
sudo apt-get install ros-melodic-usb-cam
```

启动测试文件，驱动笔记本电脑摄像头

> Note

启动前，请检查右下角电脑的摄像头是否与虚拟机连接

```
roslaunch usb_cam usb_cam-test.launch
```

> Note

如果摄像头打开后又关闭，请尝试将虚拟机的USB兼容性为 `3.0`

启动omove深度视觉驱动文件

```
roslaunch realsense2_camera rs_camera.launch
```

通过rqt工具获取图像

```
rosrun rqt_image_view rqt_image_view
```

查看图像消息格式

```
rosrun rqt_msg rqt_msg
```

2.3.2 激光雷达消息获取

启动omove激光雷达和omove小车驱动文件

```
roslaunch omove_driver omove_driver_rplidar.launch
```

查看激光雷达消息格式

```
rosrun rqt_msg rqt_msg
```

2.3.3 omove移动机器人视觉传感器、激光雷达和IMU消息格式查看

```
rosmsg show sensor_msgs/Image
rosmsg show sensor_msgs/LaserScan
rosmsg show sensor_msgs/Imu
```

2.3.4 ROS图像和opencv图像的转换原理、方法

- ROS图像转换为opencv图像

```
cv_image = bridge.imgmsg_to_cv2(image_message, desired_encoding="passthrough")
```

- opencv图像转换ROS图像

```
cv_image = cv2_to_imgmsg(cv_image, encoding="passthrough")
```

最后更新: 2021-12-05

2.4 实验四

```
roslaunch realsense2_camera rs_camera.launch
```

2.4.1 颜色识别

scripts/ros-opencv-color.py

```
#!/usr/bin/env python
import rospy
import cv2
import numpy as np
from cv_bridge import CvBridge, CvBridgeError
from sensor_msgs.msg import Image
class rosopencvcolor():
    def __init__(self):
        rospy.init_node('ros_opencv_color')
        self.cv_bridge = CvBridge()
        rospy.Subscriber('/camera/color/image_raw', Image, self.color_img_cb)
        self.lane_img_pub = rospy.Publisher('/lane_image', Image, queue_size=5)
    def color_img_cb(self, data):
        try:
            cv_img = self.cv_bridge.imgmsg_to_cv2(data, 'bgr8')
        except CvBridgeError as e:
            print e
        height, width, channels = cv_img.shape
        lower = np.array([0,0,140], dtype = "uint8")
        upper = np.array([255, 255, 255], dtype = "uint8")
        mask = cv2.inRange(cv_img, lower, upper)
        extraction = cv2.bitwise_and(cv_img, cv_img, mask = mask)
        if not rospy.is_shutdown():
            self.lane_img_pub.publish(self.cv_bridge.cv2_to_imgmsg(np.hstack([cv_img, extraction]), "bgr8"))
            cv2.imshow("Image window", np.hstack([cv_img, extraction]))
            cv2.imshow('test', cv_img)
            cv2.waitKey(1)
if __name__ == '__main__':
    rosopencvcolor()
    rospy.spin()
    cv2.destroyAllWindows()
```

2.4.2 轮廓识别

scripts/ros-opencv-findcontours.py

```
#!/usr/bin/env python
import os
import math
import rospy
import cv2
import numpy as np
from cv_bridge import CvBridge, CvBridgeError
from sensor_msgs.msg import Image
class rosopencvfindcontours():
    def __init__(self):
        rospy.init_node('ros_opencv_findcontours')
        self.cv_bridge = CvBridge()
        rospy.Subscriber('/camera/color/image_raw', Image, self.color_img_cb)
        self.lane_img_pub = rospy.Publisher('/lane_image_findcontours', Image, queue_size=5)
    def color_img_cb(self, data):
        try:
            cv_img = self.cv_bridge.imgmsg_to_cv2(data, 'bgr8')
        except CvBridgeError as e:
            print e
        ret, thresh = cv2.threshold(cv2.cvtColor(cv2.GaussianBlur(cv_img, (5, 5), 0), cv2.COLOR_BGR2GRAY), 127, 255, cv2.THRESH_BINARY)
        img_, contours, hierarchy = cv2.findContours(thresh, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
        cv2.drawContours(cv_img, contours, -1, (255, 120, 0), 3)
        cv2.imshow('Image', np.hstack((cv2.cvtColor(thresh, cv2.COLOR_GRAY2BGR), cv_img)))
        cv2.waitKey(1)
if __name__ == '__main__':
    rosopencvfindcontours()
    rospy.spin()
    cv2.destroyAllWindows()
```

最后更新: 2021-12-05

2.5 实验五

安装gazebo（如安装完整版ROS系统，则已经安装gazebo，可跳过此步）

```
sudo apt-get update
sudo apt-get install ros-melodic-gazebo*
```

2.5.1 gazebo仿真环境创建

1. 菜单栏选择Edit（编辑）>Building Editor（模拟建筑物编辑器），或使用快捷键Ctrl+B打开环境编辑器
2. 选择调色板区域Wall（墙壁），在界面右上侧二维视图区域单击鼠标左键开始绘制墙壁，移动鼠标可以拖动墙壁，会以橙色高量显示墙壁和其长度，再次单击鼠标左键确认墙壁的终点。可以选择继续移动鼠标以上一面墙壁的终点开始绘制下一面墙壁，也可以单击鼠标右键结束此面墙壁的绘制。
3. 以相同的方法绘制一个封闭仿真环境，绘制完成后可以对墙壁进行编辑和调整，鼠标左键双击需要编辑的墙壁可以弹出检查器，在检查器中可以对墙壁的位置、长度、厚度和高度等参数进行编辑。这里将外墙长度设置为4m，内外墙高度都设置为1m，厚度默认为0.15m。
4. 给墙壁添加材质和纹理特征，选择调色板区域下方Add Texture（添加质地）中的Bricks（砖）选项，然后鼠标左键单击三维视图区域的外墙，即可以为外墙添加砖的纹理和质地。
5. 在菜单栏选择File（文件）>Save As（另存为），另存当前编辑好的仿真环境文件，选择一个保存路径并取名为 `my_building`（可自定义名称），保存完成之后可以看到生成一个名为 `my_building` 的文件夹，包含 `model.config` 和 `model.sdf` 两个文件。
6. 新建一个环境，导入之前所创建的建筑物，定好原点，保存为 `my_world.sdf`

> Note

将 `my_building` 文件夹复制到用户home目录的 `.gazebo/models` 路径下，用户即可在Gazebo界面的面板Insert选项卡中看到此模型文件，可直接选中插入到场景窗口中。

2.5.2 Turtlebot3 Burger机器人运动仿真

安装依赖包

```
sudo apt-get install -y ros-melodic-joy ros-melodic-teleop-twist-joy ros-melodic-teleop-twist-keyboard ros-melodic-laser-proc ros-melodic-rgbd-launch ros-melodi
```

创建ROS工作空间并下载Turtlebot移动机器人软件包

```
mkdir -p ~/turtlebot_ws/src && cd ~/turtlebot_ws/src
git clone https://hub.fastgit.org/ROBOTIS-GIT/turtlebot3_msgs.git
git clone https://hub.fastgit.org/ROBOTIS-GIT/turtlebot3.git
git clone https://hub.fastgit.org/ROBOTIS-GIT/turtlebot3_simulations.git
cd ~/turtlebot_ws && catkin_make
```

运行仿真功能包并观察效果

```
cd ~/turtlebot_ws
source devel/setup.bash
export TURTLEBOT3_MODEL=burger     #选择导入burger model
roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

键盘控制仿真环境中的机器人运动

```
source devel/setup.bash
export TURTLEBOT3_MODEL=burger
rosrun turtlebot3_teleop turtlebot3_teleop_key
```

修改仿真环境模型为自己创建的仿真环境

1. 将之前创建的 `my_world.sdf` 文件复制到 `~/turtlebot_ws/src/turtlebot3_simulations/turtlebot3_gazebo/worlds/` 目录下，打开 `~/turtlebot_ws/src/turtlebot3_simulations/turtlebot3_gazebo/launch/turtlebot3_world.launch` 文件，将其中加载仿真环境的命令改为：

```
<arg name="world_name" value="$(find turtlebot3_gazebo)/worlds/my_world.sdf"/>
```

2. 在此文件中修改加载Turtlebot3模型在仿真环境中的初始位置：

```
<arg name="x_pos" default="0.0"/>
<arg name="y_pos" default="0.0"/>
<arg name="z_pos" default="0.0"/>
```

3. 再次运行仿真功能包并观察效果

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

2.5.3 Turtlebot3 Burger机器人地图构建和自动导航过程

1. 保持仿真环境和键盘控制窗口运行，打开一个新的终端窗口运行slam文件：

```
source devel/setup.bash
export TURTLEBOT3_MODEL=burger
roslaunch turtlebot3_slam turtlebot3_slam.launch
```

2. 键盘控制机器人在环境中运动，创建满意的地图后用命令保存地图：

```
source devel/setup.bash
rosrun map_server map_saver -f ~/turtlebot_ws/src/turtlebot3/turtlebot3_navigation/maps/my_map
```

3. 打开 `~/turtlebot_ws/src/turtlebot3/turtlebot3_navigation/launch/turtlebot3_navigation.launch` 文件，将需要加载的地图改为之前保存的地图：

```
<arg name="map_file" default="$(find turtlebot3_navigation)/maps/my_map.yaml"/>
```

4. 保持仿真环境运行，关闭其余窗口运行的文件，运行自动导航文件：

```
source devel/setup.bash
export TURTLEBOT3_MODEL=burger
roslaunch turtlebot3_navigation turtlebot3_navigation.launch
```

鼠标左键选择rviz工具栏的 `2D Pose Estimate` 重定位仿真环境中机器人位置和地图中一致，然后选择rviz工具栏中 `2D Nav Goal` 选项，长按鼠标左键在地图上为机器人指定一个导航目标点。松开鼠标后，在短时间内规划出了一条最优的路径，机器人开始向目标点运动。

最后更新: 2021-12-05

2.6 实验六

最后更新: 2021-12-05