

移动机器人实验说明书

内容目录

[1.1.实验目的 （通过需要的实验目的去设计实验） 4](#__RefHeading___Toc2981_1076784552)

[1.2.实验原理 4](#__RefHeading___Toc2983_1076784552)

[1.3.实验环境 4](#__RefHeading___Toc2985_1076784552)

[1.3.1.机器人设备： 4](#__RefHeading___Toc2987_1076784552)

[1.3.1.1.Turtlebot3介绍： 4](#__RefHeading___Toc2989_1076784552)

[1.3.1.2.Turtlebot外形 5](#__RefHeading___Toc2991_1076784552)

[1.3.1.3.Turtlebot3特性： 5](#__RefHeading___Toc2993_1076784552)

[1.3.2.软件环境 7](#__RefHeading___Toc2995_1076784552)

[1.3.2.1.ROS操作系统来源 7](#__RefHeading___Toc2997_1076784552)

[1.3.2.2.ROS系统特性 7](#__RefHeading___Toc2999_1076784552)

[1.3.3.网络环境 8](#__RefHeading___Toc3001_1076784552)

[1.4.实验配置方法 8](#__RefHeading___Toc3003_1076784552)

[预备知识 8](#__RefHeading___Toc3005_1076784552)

[1.1.1.TurtleBot3 移动机器人装配（安装尺寸及运动性能参数、外形） 8](#__RefHeading___Toc3007_1076784552)

[1.1.2.硬件组成及系统逻辑结构图 （硬件分布 各个模块的功能 目前实验控制的架构软件控制模块的功能介绍等） 9](#__RefHeading___Toc3009_1076784552)

[1.1.2.1.核心板Intel® Joule™ SBC（single board computer） 9](#__RefHeading___Toc3011_1076784552)

[1.1.2.2.控制板OpenCR 10](#__RefHeading___Toc3013_1076784552)

[1.1.3.远程电脑的软件环境配置 10](#__RefHeading___Toc3015_1076784552)

[1.1.3.1.操作系统Ubuntu16.04LTS 10](#__RefHeading___Toc3017_1076784552)

[1.1.3.2.在Ubuntu中安装ROS 11](#__RefHeading___Toc3019_1076784552)

[1.1.3.3.安装Turtlebot3特定开源库 12](#__RefHeading___Toc3021_1076784552)

[1.1.3.4.网络设置 13](#__RefHeading___Toc3023_1076784552)

[1.1.4.Turtlebot3的软件配置方法 14](#__RefHeading___Toc3025_1076784552)

[1.1.4.1.安装Turtlebot3依赖包及USB设置 14](#__RefHeading___Toc3027_1076784552)

[1.1.4.2.网络设置 15](#__RefHeading___Toc3029_1076784552)

[1.1.5.硬件知识 15](#__RefHeading___Toc3031_1076784552)

[2.第二章ROS 概念和基本命令 16](#__RefHeading___Toc3033_1076784552)

[ROS概念和基本命令 16](#__RefHeading___Toc3035_1076784552)

[实验目的 16](#__RefHeading___Toc3037_1076784552)

[2.1.实验原理 16](#__RefHeading___Toc3039_1076784552)

[2.2.实验内容 16](#__RefHeading___Toc3041_1076784552)

[2.3.实验要求 16](#__RefHeading___Toc3043_1076784552)

[4.1.机器人连接实验 19](#__RefHeading___Toc3045_1076784552)

[4.1.1.实验目的 19](#__RefHeading___Toc3047_1076784552)

[4.1.2.实验步骤参考： 19](#__RefHeading___Toc3049_1076784552)

[准备好实验环境 19](#__RefHeading___Toc3051_1076784552)

[建立连接前准备 19](#__RefHeading___Toc3053_1076784552)

[代码流程 19](#__RefHeading___Toc3782_765870155)

[Bringup\_主机设置 [PC] 20](#__RefHeading___Toc3055_1076784552)

[Bringup\_Waffle设置 [Waffle] 21](#__RefHeading___Toc3057_1076784552)

[运行可视化软件RViz [PC] 21](#__RefHeading___Toc3059_1076784552)

[使用遥控控制机器人 [PC] 22](#__RefHeading___Toc3061_1076784552)

[实验常见问题： 24](#__RefHeading___Toc3063_1076784552)

[实验二 远程操作和SLAM 26](#__RefHeading___Toc3069_1076784552)

[实验准备： 26](#__RefHeading___Toc3071_1076784552)

[实验步骤 26](#__RefHeading___Toc3073_1076784552)

[SLAM实验[PC] 27](#__RefHeading___Toc3075_1076784552)

[自动导航Navigation 28](#__RefHeading___Toc3077_1076784552)

[提问：根据观察推断机器人路径规划的大致计算方法？ 29](#__RefHeading___Toc3079_1076784552)

[实验常见问题 29](#__RefHeading___Toc3081_1076784552)

[实验三 模拟控制和GAZEBO虚拟世界构建 30](#__RefHeading___Toc3083_1076784552)

[实验准备 30](#__RefHeading___Toc3085_1076784552)

[实验准备 33](#__RefHeading___Toc3087_1076784552)

[实验内容： 33](#__RefHeading___Toc3089_1076784552)

[1 获取图像 33](#__RefHeading___Toc3091_1076784552)

[Exp 4-1 follwer.py 34](#__RefHeading___Toc3093_1076784552)

[Exp4-2 follower\_opencv.py 37](#__RefHeading___Toc3095_1076784552)

[2.建立自己的软件包！ 38](#__RefHeading___Toc3097_1076784552)

[3识别引导线 40](#__RefHeading___Toc3099_1076784552)

[Exp4-3 follower\_line\_finder.py 42](#__RefHeading___Toc3101_1076784552)

[4-4循线跟踪 43](#__RefHeading___Toc3103_1076784552)

[Exp4-5 follower\_ proporation.py 44](#__RefHeading___Toc3105_1076784552)

[作业： 45](#__RefHeading___Toc3417_1076784552)

[Measurement Performance Specifications 55](#__RefHeading___Toc3109_1076784552)

1. 第一章 移动机器人基础知识
   1. 实验目的 （通过需要的实验目的去设计实验）

已知起始位置和已知的最终位置，在已知地图的周围规划一条路径，不能接触地图中的任何障碍物。

1）基于图的规划方法( Graph-based Plan Methods)

在这个模块中，我们将介绍经过网格的路径规划问题，在该规划方法中机器人只能出现在离散位置上。我们可以将这些情境建模为图，其中的节点对应于网格位置，边对应于相邻网格单元之间的路径。教材中的算法原理，可以用于规划起始节点和目标节点之间的路径，一些经典算法包括Voronoi图法等。

2）形位空间(Configuration Space, C-Space)的计算

这个模块，在于理解掌握形位空间的概念。它是一个数学工具，用于考虑机器人能够达到的位置集合。相应地，形位空间障碍的概念，指形位空间中由于存在障碍或其他实验装置，机器人无法到达的区域。这个公式有利于促进对路径规划问题的思考，允许通过形位空间点构造轨迹。通过离散连续的形位空间为图，就可以应用基于图的工具来解决我们的运动规划问题。

3）基于采样的路径规划法

在这个模块中，通过实验理解掌握基于样本的路径规划技术的概念。这涉及到在形位空间中随机采样点，然后在相邻的采样点之间形成无碰撞的边缘，形成一个捕获机器人形位空间结构的图。探索概率路线图和快速随机搜索树（Randomly Exploring Rapid Trees, RRTS）及其运动规划问题中的应用。

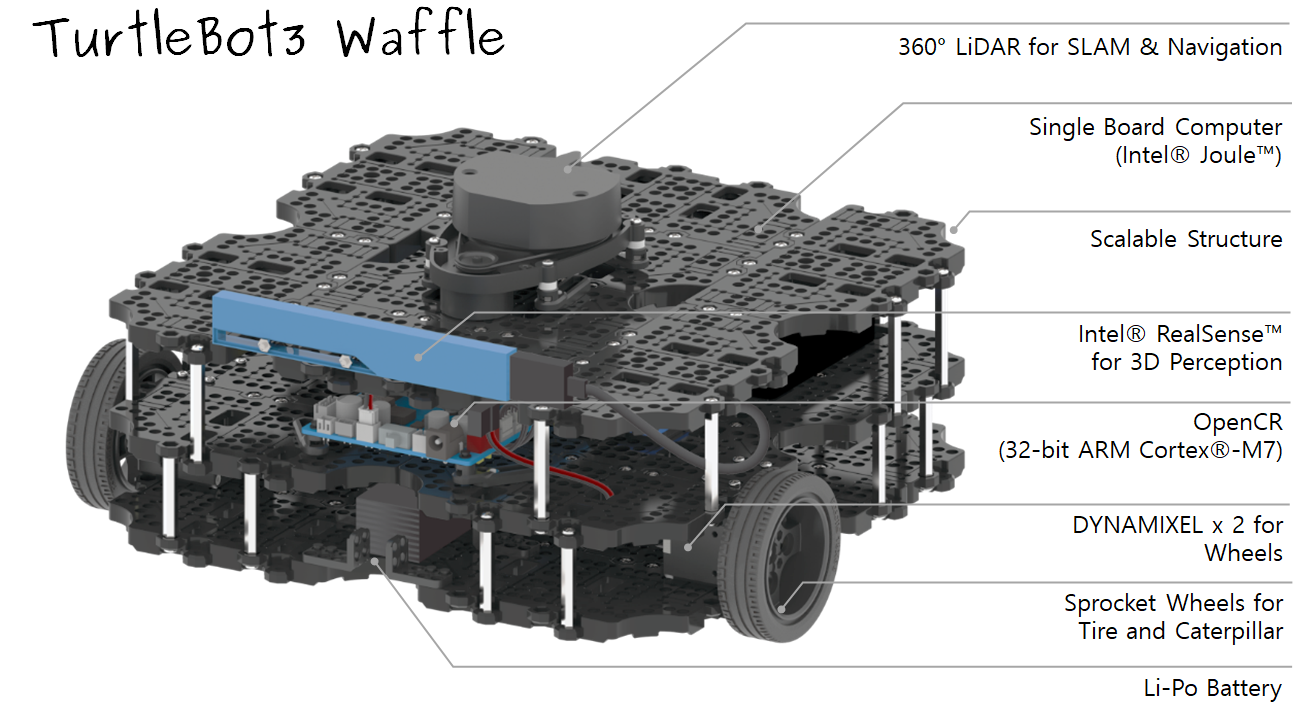
* 1. 实验原理

（待补充）

* 1. 实验环境
     1. 机器人设备：
        1. Turtlebot3介绍：

TurtleBot3是一款模块化，紧凑且可定制的新一代移动机器人。TurtleBot3的目标是在不牺牲功能，功能和质量的情况下大幅缩小平台的尺寸并降低价格。诸如底盘，计算机和传感器等可选部件都可用，TurtleBot3可以通过各种方式进行定制。 TurtleBot3愿意通过应用SBC（单板计算机）的最新技术进步，深度传感器和3D打印技术，成为制造商运动的中心。

* + - 1. Turtlebot外形



* + - 1. Turtlebot3特性：



1. 流行的开发平台

Turlebot是最流行的教育和研究用的开源机器人，新一代的Turtlebot3兼顾小型、廉价、全可编程的基于ROS的移动机器人。它是为教育、研究、业余爱好和产品原型开发的新一代产品。

1. 可承受的价格

Turtlebot是为了迎合学校、实验室和企业的预算限制的需求开发的产品，Turtlebot3是目前市面上装备有360度雷达距离传感器、能够使用SLAM功能的价格最实惠的移动机器人

1. 小尺寸携带便捷

TurtleBot3 Burger的尺寸仅为138mm x 178mm x 192mm（长x宽x高）。 它的大小约为前辈的四分之一。 想象一下，将TurtleBot3放在背包里，开发你的程序并在任何地方进行测试。

1. ROS标准

TurtleBot品牌由Open Robotics管理，Open Robotics负责开发和维护ROS。 如今，ROS已经成为全球所有机器人专家的首选平台。 TurtleBot可以与现有的基于ROS的机器人组件集成，但TurtleBot3可以成为一个负担得起的平台，为他们开始学习ROS。

1. 可扩展性

TurtleBot3鼓励用户使用一些替代选项来定制其机械结构：开源嵌入式电路板（作为控制板），计算机和传感器。 TurtleBot3 Burger是一款两轮差速驱动式平台，但它可以通过多种方式进行结构和机械定制：汽车，自行车，拖车等。 通过可扩展结构上的各种SBC，传感器和电机，将您的想法超越想象。

1. 移动机器人的模块化执行器

TurtleBot3能够通过在车轮关节中使用2个DYNAMIXEL获得精确的空间数据。 DYNAMIXEL XM系列可通过6种操作模式（XL系列：4种操作模式）中的一种操作：轮子的速度控制模式，转矩控制模式或关节的位置控制模式等。DYNAMIXEL甚至可用于制造移动机械手， 轻，但可以通过速度，扭矩和位置控制进行精确控制。 DYNAMIXEL是使TurtleBot3完美的核心组件。 组装，维护，更换和重新配置非常简单。

1. 为ROS开放控制板

控制板采用硬件方式开源，并以ROS通信的软件方式进行开源。 开源控制板OpenCR1.0功能强大，不仅可以控制DYNAMIXEL，还可以控制常用于基本识别任务的ROBOTIS传感器，而且成本效益更高。 各种传感器，如触摸传感器，红外传感器，彩色传感器和一些更多可用。 OpenCR1.0在电路板内部有一个IMU传感器，因此它可以增强对无数应用的精确控制。 该电路板具有3.3V，5V，12V电源，以加强可用的计算机设备阵容。

1. 强大的传感器阵容

TurtleBot3 Burger使用增强型360°LiDAR，9轴惯性测量单元和精密编码器来进行研究和开发。 TurtleBot3 Waffle配备了完全相同的360°LiDAR，另外还提供了一个强大的带有识别SDK的英特尔®Realsense™。这将是制造移动机器人的最佳硬件解决方案。

1. 开源

TurtleBot3的硬件，固件和软件都是开源的，这意味着欢迎用户下载，修改和共享源代码。 TurtleBot3的所有组件均采用注塑成型塑料制造以实现低成本，但3D CAD数据也可用于3D打印。 3D CAD数据通过全云3D CAD编辑器Onshape发布。 用户可以通过桌面PC，笔记本电脑甚至便携式设备上的网络浏览器进行访问。 Onshape允许绘制3D模型并将其与同事组装。 此外，对于想要自行开发OpenCR1.0开发板的用户来说，OpenCR1.0开发板的所有细节，如原理图，PCB Gerber文件，BOM和固件源代码都是在用户开放源代码许可证和ROS 社区。 您可以修改下载的源代码和硬件以与朋友分享。

* + 1. 软件环境
       1. ROS操作系统来源



ROS是为机器人设计的开源、基于元操作的系统。它满足着你对操作系统服务的期望，包括硬件抽象、低等级驱动控制、常用功能的实现，进程之间的消息传递和包管理。它还提供用于在多台计算机上获取、生成、写入和运行代码的工具和库。

ROS是一个有许多祖先和贡献者的大型项目。机器人研究界的许多人都认为需要一个开放式的协作框架，并且已经为此目标创建了许多项目。

斯坦福大学在二十世纪二十年代中期进行了各种努力，涉及STanford人工智能机器人（STAIR）和个人机器人（PR）计划等集成的，体现的人工智能，为机器人应用创建了灵活的动态软件系统的内部原型。 2007年，附近有远见的机器人孵化器Willow Garage提供了大量资源来进一步扩展这些概念并创建经过充分测试的实现。无数的研究人员为核心ROS理念及其基本软件包贡献了他们的时间和专业知识，推动了这一努力。在整个过程中，该软件都是使用宽松的BSD开源许可进行公开开发的，并逐渐成为机器人研究界广泛使用的平台。

从一开始，ROS就是在多个机构和多个机器人开发的，其中包括许多从Willow Garage收到PR2机器人的机构。尽管所有贡献者将代码放置在同一台服务器上要简单得多，但多年来，“联合”模型已成为ROS生态系统的一大优势。任何团队都可以在自己的服务器上启动他们自己的ROS代码库，并且他们可以维护它的完全所有权和控制权。他们不需要任何人的许可。如果他们选择公开提供他们的存储库，他们可以获得应有的成就和荣誉，并从特定的技术反馈和改进中受益，例如所有开源软件项目。

* + - 1. ROS系统特性

ROS生态系统现在由全球数以万计的用户组成，致力于从桌面业余爱好项目到大型工业自动化系统等领域。

机器人操作系统（ROS）是编写机器人软件的灵活框架。它是一组工具，库和约定，旨在简化在各种机器人平台上创建复杂而强大的机器人行为的任务。

为什么？因为创建真正强大的通用机器人软件非常困难。从机器人的角度来看，对于人类来说微不足道的问题往往在任务和环境的实例之间大相径庭。处理这些变化非常困难，以至于没有任何一个人，实验室或机构可以希望自己做。

因此，ROS从根本上建立起来，鼓励协作机器人软件开发。例如，一个实验室可能拥有测绘室内环境的专家，并可以为制作地图提供​​世界级的系统。另一组可能有专家在使用地图进行导航，而另一组可能已经发现了一种计算机视觉方法，该方法对于识别杂物中的小物体非常有效。 ROS是专门为像这样的团体设计的，以便在本网站中描述的彼此协作和共同工作。

* + 1. 网络环境

ROS 是一种分布式计算环境。一个运行的 ROS 系统可以包括许多, 甚至数以百计的节点, 分布在多台机器上。根据系统的配置方式, 任何节点在任何时候都可能需要与任何其他节点通信。

因此, ROS 对网络配置有一定的要求:

•必须有完整的, 在所有端口上的所有对机器之间的双向连接。

•每个机器必须用所有其他机器可以识别的名称来做广播。

则在实验环境中较为便捷的网络配置方案为：给每一个机器人配置不同的节点名称，将计算主机和机器人通过同一无线网络路由器进行链接。

* 1. 实验配置方法

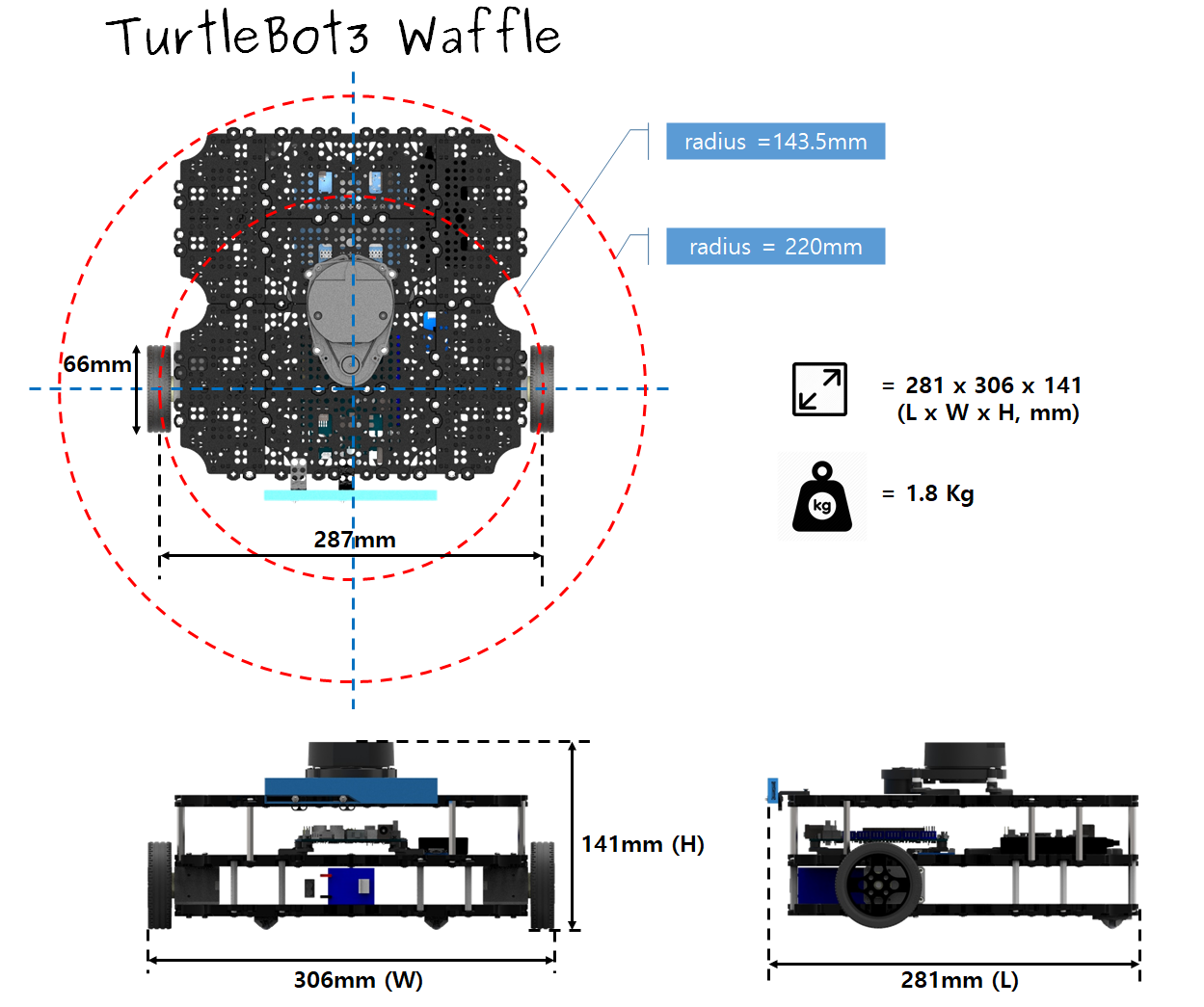
预备知识

在实验中，电脑的操作环境为LINUX系统，Linux是一套免费使用和自由传播的类Unix操作系统，是一个基于POSIX和UNIX的多用户、多任务、支持多线程和多CPU的操作系统。Linux同时具有字符界面和图形界面，在字符界面用户可以通过键盘输入相应的指令来进行操作，在图形界面用户也可以使用鼠标进行操作。实验之前，同学需要自行熟悉Linux的终端操作方法和基本命令的读识。附件包含Linux系统 bash指令概述。（请务必进行预期，否则学习会非常不畅快）

* + 1. TurtleBot3 移动机器人装配（安装尺寸及运动性能参数、外形）

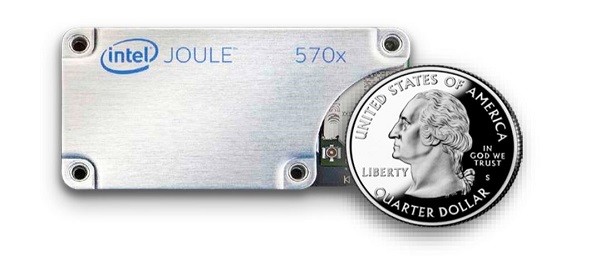
Turtlebot3新产品需要自行组装，根据随机器提供的组装说明书进行组装。

电子版组装教程见链接：http://emanual.robotis.com/docs/en/platform/turtlebot3/hardware\_setup/#hardware-setup



* + 1. 硬件组成及系统逻辑结构图 （硬件分布 各个模块的功能 目前实验控制的架构软件控制模块的功能介绍等）
       1. 核心板Intel® Joule™ SBC（single board computer）

Intel 对开发套件市场来说并不陌生，Joule作为新的单板计算机产品优点为外形紧凑，处理性能强大。其微型模块上系统 (SOM) 只有 24mm x 48mm 大小，但却囊括了众多功能，如支持 Intel® RealSense™ 技术的 4K 视频捕获和显示功能。该产品允许设备捕获景深 (DOF) 信息，让应用对环境及其中的任何对象都有一个空间感。我们很容易就能想象出它在以下领域的作用多么强大：自动驾驶车辆、无人机和机器人、汽车行业的驾驶员辅助应用、增强现实和虚拟现实领域、工业物联网…以及实现对当地环境的三维空间感认识会非常有用的许多其他应用。



Intel® Joule™ 570X 模块的功能：

* 高性能 64 位 1.7 GHz 四核 Intel® Atom™ T5700 处理器，突发频率可增强至 2.4 GHz
* 4GB LPDDR4 RAM 和 16GB eMMC 内存
* Intel® 高清显卡，支持 4K 视频捕获和显示
* 支持 MIMO 和蓝牙 4.1 的 802.11ac Wi-Fi
* USB 3.0、MPI\* CSI 和 DSI 接口，以及多个 GPIO、I2C 和 UART 接口
* 基于 Linux 的操作系统，专为物联网和智能设备而定制
* 增强对 Intel® RealSense 摄像头和数据库的支持
  + - 1. 控制板OpenCR

OpenCR1.0是TurtleBot3的主控制器板。 OpenCR1.0;针对ROS的开源控制模块，是针对ROS嵌入式系统开发的，提供完全开源的硬件和软件。一切关于控制板、原理图，PCB 、BOM和TurtleBot3的固件源代码等资料可以根据用户和ROS社区的开源许可证免费获取。

STM32F7系列是OpenCR1.0开发板中的主要芯片，该开发板基于功能强大的带有浮点单元的ARM Cortex-M7。 OpenCR1.0的开发环境从Arduino IDE和Scratch开放，适合年轻学生使用专家的固件开发。

OpenCR1.0提供数字和模拟输入/输出引脚，可以连接扩展板或各种传感器。此外，OpenCR1.0还具有各种通信接口：用于连接PC，UART，SPI，I2C和USB的USB，用于其他嵌入式设备。

使用SBC时，OpenCR1.0可以提供最佳的解决方案。它支持SBC和传感器的12V，5V，3.3V电源输出。它还支持电池和SMPS之间的热插拔电源输入。

* + 1. 远程电脑的软件环境配置
       1. 操作系统Ubuntu16.04LTS

ROS在Linux环境下运行，目前使用的版本号为Ubuntu16.04LTS。通过官方网站下载系统安装镜像之后根据指导手册进行安装。

指导手册链接：<https://www.ubuntu.com/download/desktop/install-ubuntu-desktop>

* + - 1. 在Ubuntu中安装ROS

声明：使用的Ubuntu版本为16.04(Xenia), ROS使用的版本号为Kinetic。

* 设定sources.list

将电脑设置为能够接受ROS包的状态。首先打开linux的命令窗口界面（ctrl+alt+T）键入：

$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb\_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'

* 设置ROS密钥

$ sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116

* 安装ROS

首先确定系统组件为最新版本

$ sudo apt-get update

在ROS中有很多不同的库和工具，在开始阶段有几种不用的默认安装设置可以选择，并可以分开安装ROS的不同组件，我们在这个阶段可以选择全组件完整安装（推荐）。在全组件安装中，软件包包含了ROS、rqt（机器人调试界面）、rviz（机器人可视化操作界面）、机器人通用库、2D/3D模拟器，导航和2D/3D感知组件。

$ sudo apt-get install ros-kinetic-desktop-full

安装过程中可能会遇到中断，一般是由于网络问题导致，只需要重新键入命令即可。

在全组件包安装完成之后，用户可以根据自身需求单独安装其他的组件，搜索可用组件，可以使用搜索命令，根据搜索得到的包名称进行安装:

$ apt-cache search ros-kinetic

$ sudo apt-get install ros-kinetic-PACKAGE

e.g.

$ sudo apt-get install ros-kinetic-slam-gmapping

* 初始化 rosdep

在开始使用ROS之前，需要初始化运行rosdep，可以让用户轻松地为要变异的源安装系统依赖项，并且在ROS某些核心组件中也需要该程序。

$ sudo rosdep init

$ rosdep update

* 建立ROS运行环境

建立运行环境的目的在于使得每一次运行新的程序时，ROS环境自动地从已经建立好的位置开始运行。设置的方法：

$ echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc \\将位置写入到ROS位置表中

$ source ~/.bashrc \\确认位置表

* 构建包的依赖关系

截至目前已经安装好运行核心ROS包的各种组件，为了能够自己创建和管理ROS工作空间，还可以另外分别安装其他你需要的各种工具。比如，rosinstall就是在你需要能快捷安装各种ROS包时候使用的命令行工具。

为了安装这些工具和构建ROS包的依赖关系，运行

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

至此便完成了ROS的安装过程，通过一步步的操作已经在电脑中建立了完备的ROS工具组，在这之后便可以自己安装需要的ROS开源库进行更高级的操作了。

* + - 1. 安装Turtlebot3特定开源库

目前我们已经有了完整的ROS工具，在这之上turtlebot是已经组装好的产品化的机器人，我们需要调用由开发者提供的开源库进行控制。开源库的作用便是建立了ROS工具和机器人的特定映射关系，下面安装Turtlebot3控制使用的依赖包（dependent packages）

**$ sudo apt-get install ros-kinetic-joy ros-kinetic-teleop-twist-joy ros-kinetic-teleop-twist-keyboard ros-kinetic-laser-proc ros-kinetic-rgbd-launch ros-kinetic-depthimage-to-laserscan ros-kinetic-rosserial-arduino ros-kinetic-rosserial-python ros-kinetic-rosserial-server ros-kinetic-rosserial-client ros-kinetic-rosserial-msgs ros-kinetic-amcl ros-kinetic-map-server ros-kinetic-move-base ros-kinetic-urdf ros-kinetic-xacro ros-kinetic-compressed-image-transport ros-kinetic-rqt-image-view ros-kinetic-gmapping ros-kinetic-navigation ros-kinetic-interactive-markers**

在系统安装完所有依赖包之后，创建一个工作空间

$ cd ~/catkin\_ws/src/ \\创建文件夹

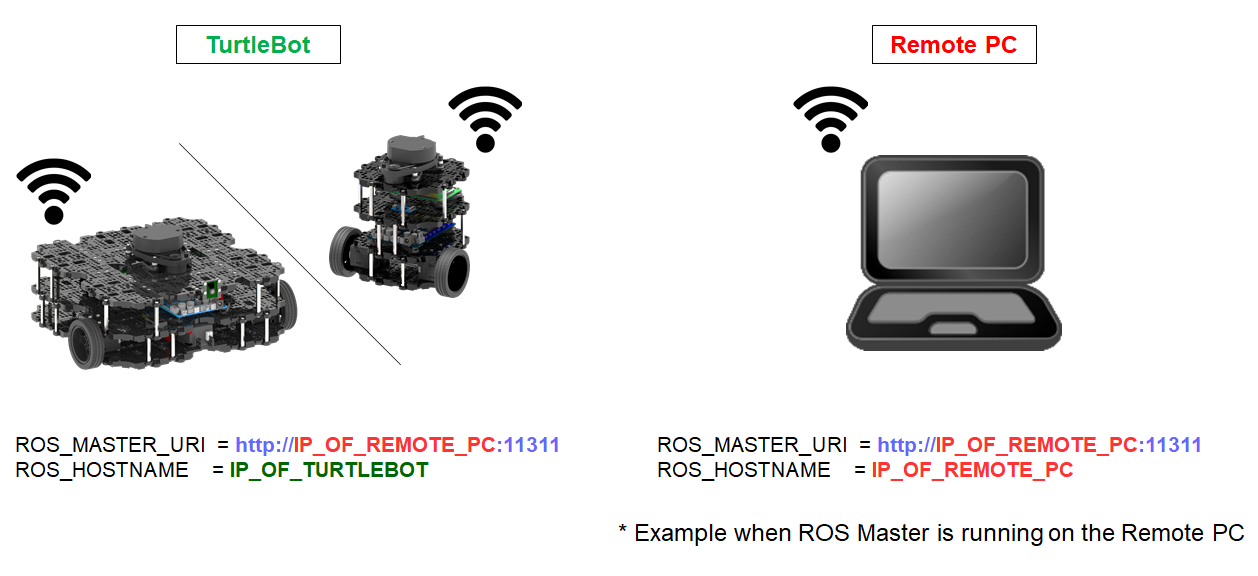
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3\_msgs.git

$ git clone https://github.com/ROBOTIS-GIT/turtlebot3.git \\从github上加载软件包

$ cd ~/catkin\_ws && catkin\_make \\进入文件夹 使用make函数

如果运行catkin\_make之后，没有报错则说明Turtlebot3的运行环境已经设置好了。

* + - 1. 网络设置

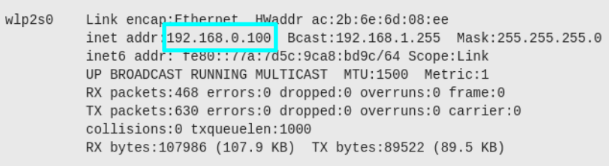


ROS系统的正常运行需要远程主机和机器人可以进行双向通信，在前面的网络环境介绍中已提到，我们需要用IP地址来进行节点识别。

首先将远程主机和机器人接入同一无线局域网内，此时可以通过以下命令查看分配到的网络IP地址

$ ifconfig

此时得到



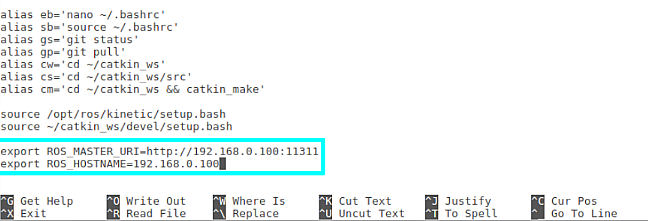
此时键入以下命令：

$ gedit ~./bashrc

然后按‘ alt+/ ‘ 滑到列表最下面，对本地主机（localhost）地址进行编辑，将上一步得到的IP地址写入主机地址行。界面如下图所示，其中，

$ export ROS\_MASTER\_URI=http:// 192.168.0.11 : 11311 \\11311为ROS环境默认主机代号

$ export ROS\_HOSTNAME = 192.168.0.11



关闭窗口之后，在终端输入命令以确认设置

$ source ~./bashrc

* + 1. Turtlebot3的软件配置方法

实验前已经为机器人配置了linux系统环境，其系统版本为Alternative Ubuntu 16.04 for Intel® Joule™ 。安装系统后，在Ubuntu下安装ROS，具体流程与在电脑上安装ROS一样，此处不作介绍。

* + - 1. 安装Turtlebot3依赖包及USB设置

与远程电脑的区别在于移动机器人上安装了不同的依赖包，用来控制不同的底层组件，其代码如下：

$ cd ~/catkin\_ws/src

$ git clone <https://github.com/ROBOTIS-GIT/hls_lfcd_lds_driver.git>

$ git clone https://github.com/ROBOTIS-GIT/turtlebot3\_msgs.git

$ git clone https://github.com/ROBOTIS-GIT/turtlebot3.git

然后删除了几个在PC端使用的依赖包

$cd ~/catkin\_ws/src/turtlebot3

$ sudo rm -r turtlebot3\_description/ turtlebot3\_teleop/ turtlebot3\_navigation/ turtlebot3\_slam/ turtlebot3\_example/

安装Turtlebot用依赖包

$ sudo apt-get install ros-kinetic-rosserial-python ros-kinetic-tf

然后重启Joule之后构建包

$ cd ~/catkin\_ws && catkin\_make

USB设置使得OpenCR1.0不需要根权限也能使用USB端口

$ rosrun turtlebot3\_bringup create\_udev\_rules

注：至此的操作只需要在第一次配置机器人时需要，完成操作后不需要更改依赖包等设置。

* + - 1. 网络设置

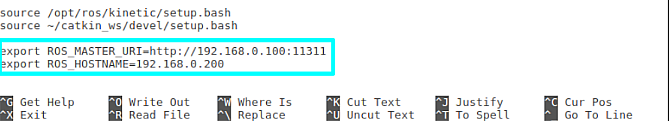
同样使用以下命令行得到机器人分配到的网络IP地址

$ ifconfig

根据得到的网络地址，同样是打开bashrc表单

$ gedit ~/.bashrc

在最后两行中MASTER\_URI输入远程主机的IP地址，在HOSTNAME中输入机器人的IP地址，这样便在机器人中设置好了网络名称。



同样，在完成修改之后需呀确定修改表单

$ source ~/.bashrc

注：每一次进行试验或接入均需要检查

* + 1. 硬件知识

实验之前了解SBC和控制板以及驱动轮的信息，其规格见附录2。

1. 第二章ROS 概念和基本命令

ROS概念和基本命令

实验目的

* 1. 实验原理
  2. 实验内容



* 1. 实验要求

1. 实验三 Turtle 小实验

实验目的

本实验目的在于使用上一章中的内容，对ROS内置的乌龟机器人进行控制，更具体的了解ROS的具体架构和各个工具的使用方法。

实验原理

实验内容

实验结果

实验要求

1. 实验四 Tutlebot实战
   1. 机器人连接实验
      1. 实验目的

当准备工作做好之后，我们便可以开始着手于对机器人的控制了。首先需要做的是建立远程主机和Tultlebot3的网络连接，最终达到能双向通讯的效果。

* + 1. 实验步骤参考：

准备好实验环境

网络：实验环境需要在联网的情况下进行,且需要连接相同的无线网络。这样做的目的在于ROS系统在使用过程中移动机器人需要访问并传递信息到远程电脑建立的程序核心服务器(roscore),信息不经过互联网所以只能在局域网中进行连接。

控制：Turtlebot开始运行前需要连接显示屏，(roslaunch)启动模块控件。

空间：无随机障碍环境。

电源：调试过程请使用12V电源转换器连接，使用SLAM实验使用电池。同时使用时拔掉转换器会引起故障，建议调试完成后只接电池重启运行。

建立连接前准备

首先在远程电脑上进行测试，在开始连接之前需要确定：

* 本代码在远程电脑上运行，如果在Turtlebot上输入代码请不要运行roscore程序
* 确保远程主机和机器人IP网络地址设置正确
* 当电池电量低于11V时，电池会发出持续的警报声，而且部分控件无法工作，此时应该将电池充电。

代码流程

在准备工作做好之后，建议将bashrc表单中设定机器人型号名称，正确的设定在后续可视化操作中才能调用正确的三维模型。

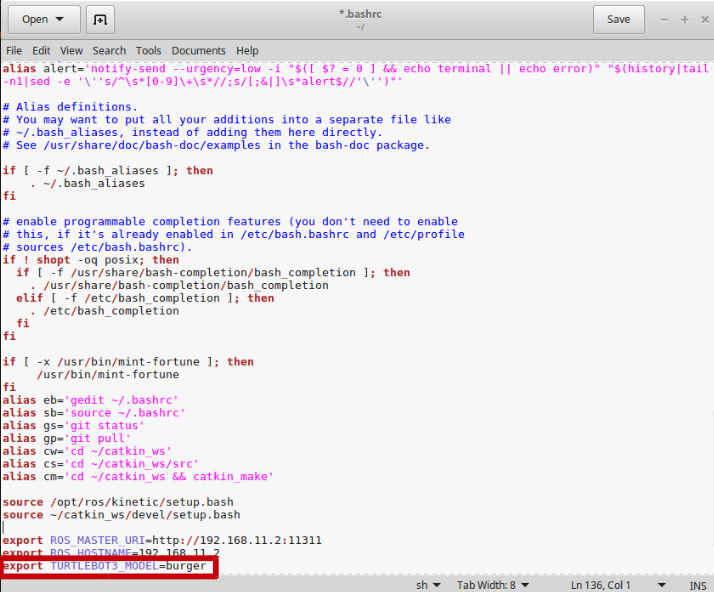
调用更改 表单命令行：

$ gedit ~/.bashrc

这样能使用gedit工具修改表单，如果无法正确调用表单需要在命令之前加上 sudo 以提供根权限。

修改为：

$ export TURTLEBOT3\_MODEL=waffle



同样，在每一次修改bashrc表单之后都需要重复操作确认表单命令已完成更新。

$ source ~/.bashrc

Bringup\_主机设置 [PC]

在远程主机终端启动程序，使用ROS的第一步就是运行：

$ roscore

此时你会看到以下代码：

... logging to ~/.ros/log/9cf88ce4-b14d-11df-8a75-00251148e8cf/roslaunch-machine\_name-13039.log

Checking log directory for disk usage. This may take awhile.

Press Ctrl-C to interrupt

Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://machine\_name:33919/

ros\_comm version 1.4.7

SUMMARY

======

PARAMETERS

\* /rosversion

\* /rosdistro

NODES

auto-starting new master

process[master]: started with pid [13054]

ROS\_MASTER\_URI=http://machine\_name:11311/

setting /run\_id to 9cf88ce4-b14d-11df-8a75-00251148e8cf

process[rosout-1]: started with pid [13067]

started core service [/rosout]

以上代码表述了从加载程序文件然后计算磁盘和内存空间，建立了一个尾号为11311的主服务器地址，并开始服务。

如果roscore没有成功初始化，说明网络设置有问题，可以回到网络设置说明检查主机地址设置是否正确。

Bringup\_Waffle设置 [Waffle]

在机器人桌面终端调用Turtlebot3 应用程序

$ roslaunch turtlebot3\_bringup turtlebot3\_robot.launch

注，此处出现常见问题见本节错误提示[Error-2]

如果想单独调用激光传感器（lide sensor）, Intel® RealSense™ R200 和核心驱动程序，可以分别打开三个终端运行以下命令：

$ roslaunch turtlebot3\_bringup turtlebot3\_lidar.launch

$ roslaunch turtlebot3\_bringup turtlebot3\_realsense.launch

$ roslaunch turtlebot3\_bringup turtlebot3\_core.launch

注：在使用Realsense相机之前请阅读相关说明 [Intel® RealSense™](http://emanual.robotis.com/docs/en/platform/turtlebot3/appendix_realsense/" \l "installation)

注：如果终端提示lost sync with device 说明传感器没有连接良好。

在命令运行良好的情况下，可以观察到终端信息不断刷新，传感器信号被成功传输到主服务器上，此时便完成了远程主机和机器人的成功连接。

此时可以使用第二章中所学的内容来了解程序各个节点的连接方式

运行可视化软件RViz [PC]

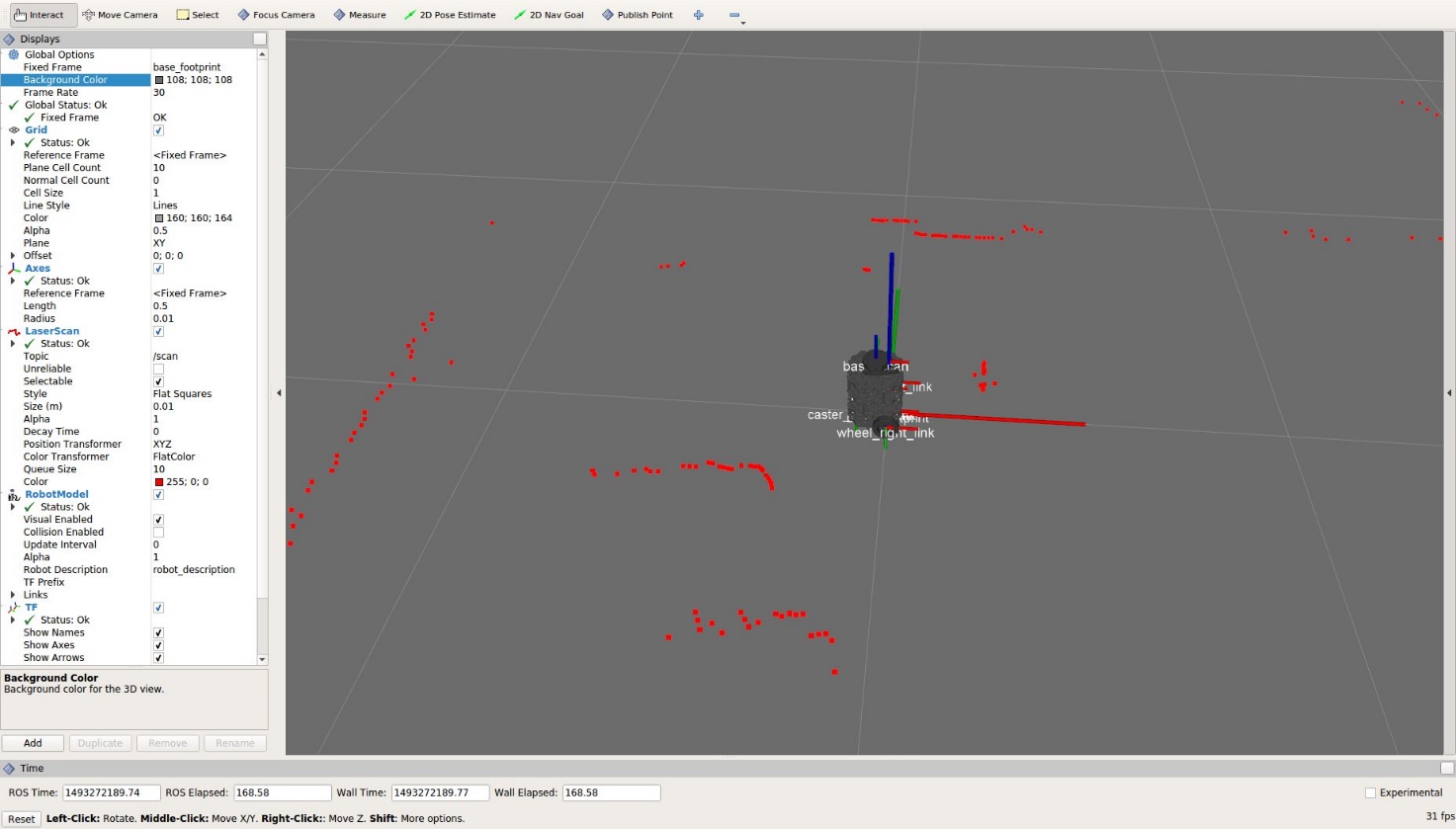
Rviz是ROS环境下的3D可视化工具，通过这个工具可以在电脑中模拟机器人在世界中的位置，并通过读取传感器数据获得虚拟环境信息。

首先声明调用的模型为waffle ，然后启动Turtlebot远程应用，启动Rviz。

$ export TURTLEBOT3\_MODEL=waffle

$ roslaunch turtlebot3\_bringup turtlebot3\_remote.launch

$ rosrun rviz rviz -d `rospack find turtlebot3\_description`/rviz/model.rviz



此时可以在程序窗口观察到调用的3D模型，窗口内出现的红点即为激光传感器的探测信息，描述了周围环境的光学反射点，通过对应的观察可以识别出椅子墙壁等物件。

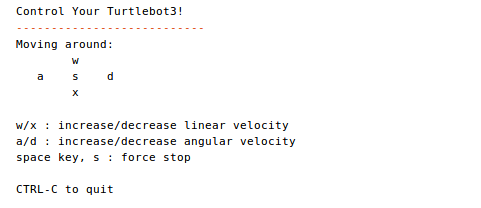
使用遥控控制机器人 [PC]

Turtlebot提供了多种遥控方案，如 Keyboard / Andriod /Xbox /Ps手柄……在完成连接的基础上，我们可以通过远程主机的键盘对小机器人进行运动控制。在开始运行之前请保证机器人在宽敞的平面上运行，如果在桌面上运行请确保机器人不会从桌面上掉落。

这里我们首先在PC上运行无线操作依赖包中的键盘驱动程序。

$ roslaunch turtlebot3\_teleop turtlebot3\_teleop\_key.launch

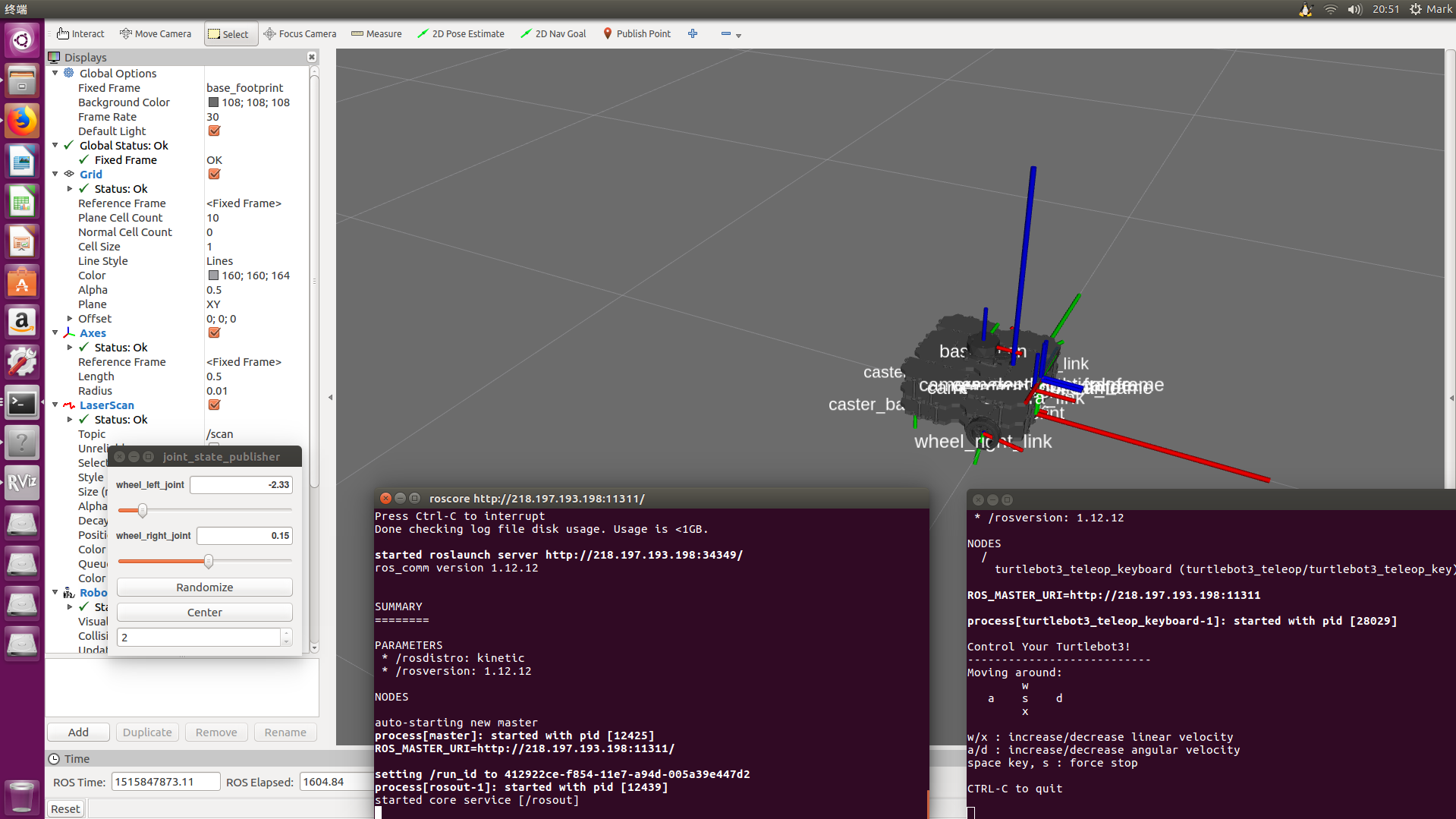
如果程序成功运行，终端界面会如下图所示：



注:本命令常见问题请查阅[Error-2]

注意，在键盘控制过程中可以发现，键盘设定的为两个驱动轮的角速度，所以在实际控制当中，改变方向之后小车并不会往这个方向直行而是继续在路径切线上左偏或右偏，这样的运行效果是小车采用绕圈的行动方式。

在小车运行过程中，上一小节中运行的激光距离传感器也同时在采集数据，运行时产生的点云扫描结果如下 图所示。



实验常见问题：

[ERROR-1]

$ roscore

 提示无法建立服务器连接;

解决办法：

第一次建立网络连接请确定主机和移动机器人服务器地址设定正确，设定网络连接命令：

$ sudo gedit ~/.bashrc

注：更改索引文件 在设定主机HOST地址填入正确的网络地址

（ifconfig查看远程电脑网络地址 具体操作请重做 设定ROS Network步骤）

$ source ~/.bashrc

%%重设索引文件

[ERROR-2]

$ roslaunch teleop\_twist\_joy teleop.launch

**OR**

$ roslaunch turtlebot3\_bringup turtlebot3\_robot.launch

出现rospack无法找到的情况，首先检查拼写。拼写无误情况下出现错误提示请运行：

$ source ～/catkin\_ws/devel/setup.bash

重新加载catkin\_ws工作空间设置文件，并查找ROS依赖包，比如bringup包

$ rospack find turtlebot3\_bringup

终端会返回rospack地址，表示已经找到依赖包。

这个时候再运行

$ roslaunch turtlebot3\_bringup turtlebot3\_robot.launch

命令才可以正确启动。

[Error 待补充]

实验二 远程操作和SLAM

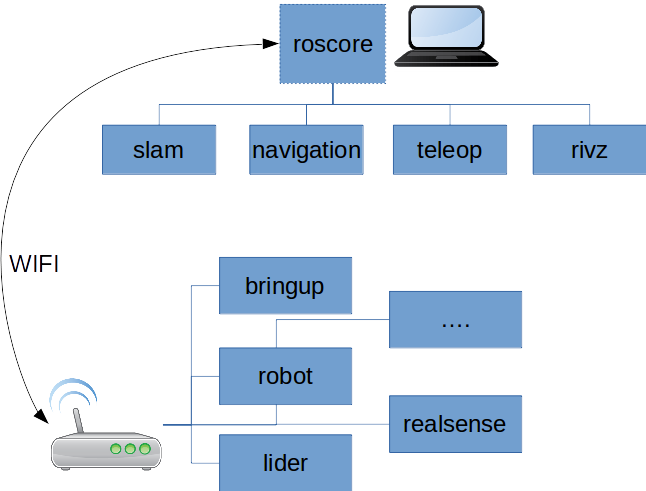
实验准备：

同实验一。并将远程主机和机器人运行至正常连接并启动传感器。

实验步骤

SLAM (simultaneous localization and mapping),也称为CML (Concurrent Mapping and Localization), 即时定位与地图构建，或并发建图与定位。问题可以描述为：将一个机器人放入未知环境中的未知位置，是否有办法让机器人一边逐步描绘出此环境完全的地图，所谓完全的地图（a consistent map）是指不受障碍行进到房间可进入的每个角落。 SLAM最早由Smith、Self和Cheeseman于1988年提出。 由于其重要的理论与应用价值，被很多学者认为是实现真正全自主移动机器人的关键。

Turtlebot3相对于前代机器人的最知名的功能就是加入了SLAM模块，本实验旨在使用提供的SLAM程序了解和使用Waffle机器人的地图实时构建功能。



上图表示了在SLAM和Navigation实验中所需要的主要程序模块及结构，其中roscore创建连接服务器，Turtlebot建立连接用以交换数据。建立连接完成之后turtlebot内启动激光雷达等部件，视使用需求使用robot总装模块和lider core等独立模块。

SLAM实验[PC]

建立连接之后进入SLAM程序

首先声明使用的模型为waffle机器人，并启动SLAM例程。

$ export TURTLEBOT3\_MODEL=waffle

$ roslaunch turtlebot3\_slam turtlebot3\_slam.launch

此时可以观察到终端窗口能够实时刷新传感器发送过来的数据。此时可以在RViz中打开模拟界面。

$ rosrun rviz rviz -d `rospack find turtlebot3\_slam`/rviz/turtlebot3\_slam.rviz

并启动键盘控制（或手柄控制，如果条件允许）

$ roslaunch turtlebot3\_teleop turtlebot3\_teleop\_key.launch

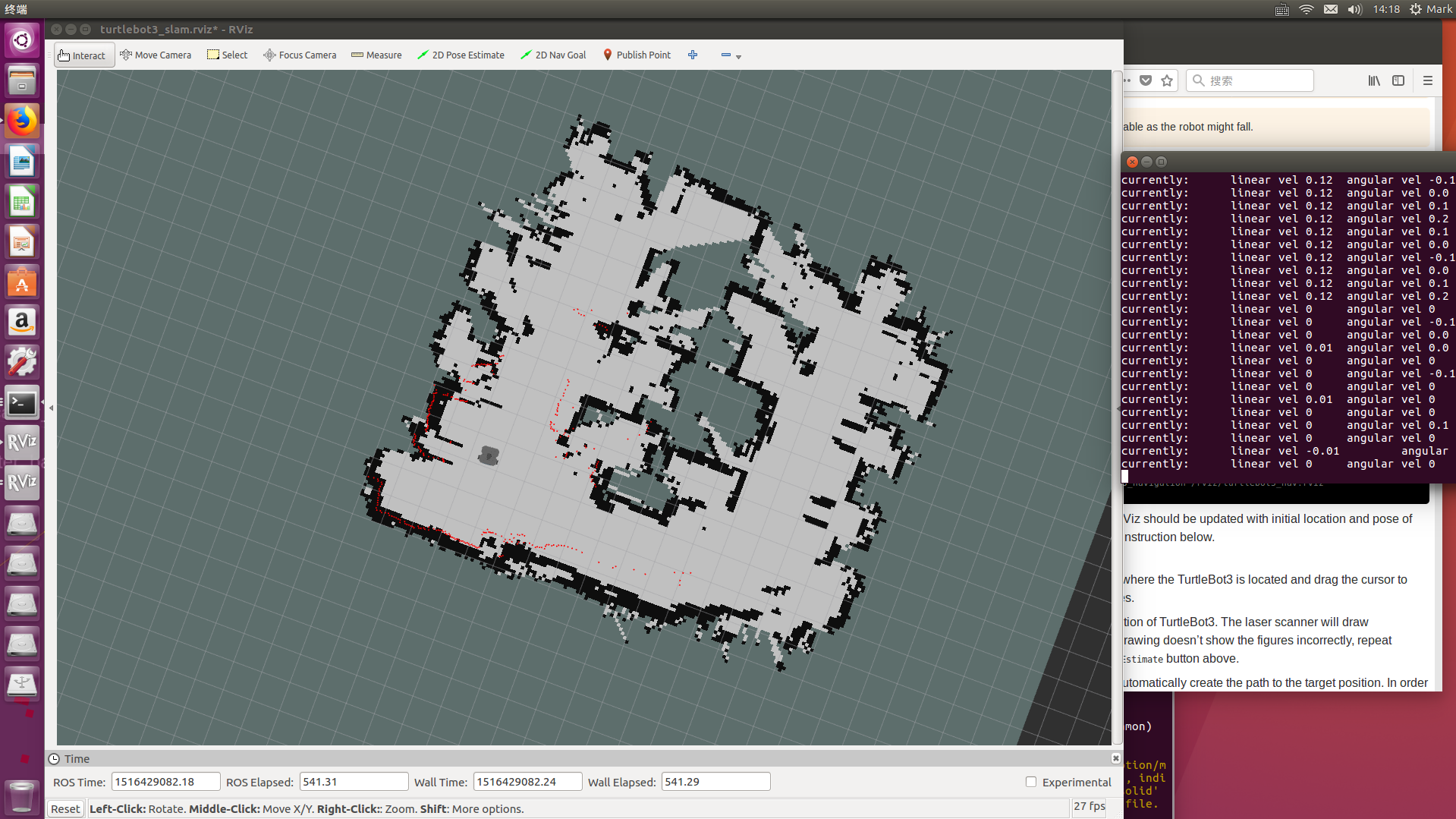
保证移动机器人有充足控件运行的前提下，通过键盘控制机器人运动，此时可以从RViz程序界面观察到由激光传感器发送过来的环境信息正在随着机器人位置的移动逐步建立了环境的外形，并通过多个位置的测量使环境信息更加丰富准确。

完成地形测量之后可以得到扫描地形图一个，打开新的终端，输入保存地图命令：

$ rosrun map\_server map\_saver -f ~/map

此时在主文件夹中 /home/<username> 会出现保存到的地图文件 map.pgm和map.yaml

下图为实例测量C210办公室的地形图，可以观察到比较明确的办公室布置。



注：扫描过程中需要注意不要使机器人与环境产生接触，不要人为在机器人附近活动，不要将机器人速度提到太快，太快的移动速递会导致环境探测信号量减少。按照规范操作，以便得到正确的地形数据。

自动导航Navigation

完成地形探测之后可以调用自动导航功能。导航功能的原理是利用已经测量好的实验地形，将机器人与原初始点相对位置和姿态确定好之后，将自然条件下的实时探测行进问题转变为在虚拟空间中对目标位置进行轨迹规划的问题。

在开始导航实验之前确认已经运行过Bringup任务，同样在新的终端中声明使用的机型waffle

$ export TURTLEBOT3\_MODEL=waffle

$ roslaunch turtlebot3\_navigation turtlebot3\_navigation.launch map\_file:=$HOME/map.yaml

以上程序便完成了读取在上一节中测量好的地图数据 map.yaml

接下来启动RViz可视化程序执行导航任务。

$ rosrun rviz rviz -d `rospack find turtlebot3\_navigation`/rviz/turtlebot3\_nav.rviz

在开始导航之前，RViz需要确定机器人的初始位置。初始位置的正确建立即产生了正确的从实体空间到地图空间的映射，校准方法如下：

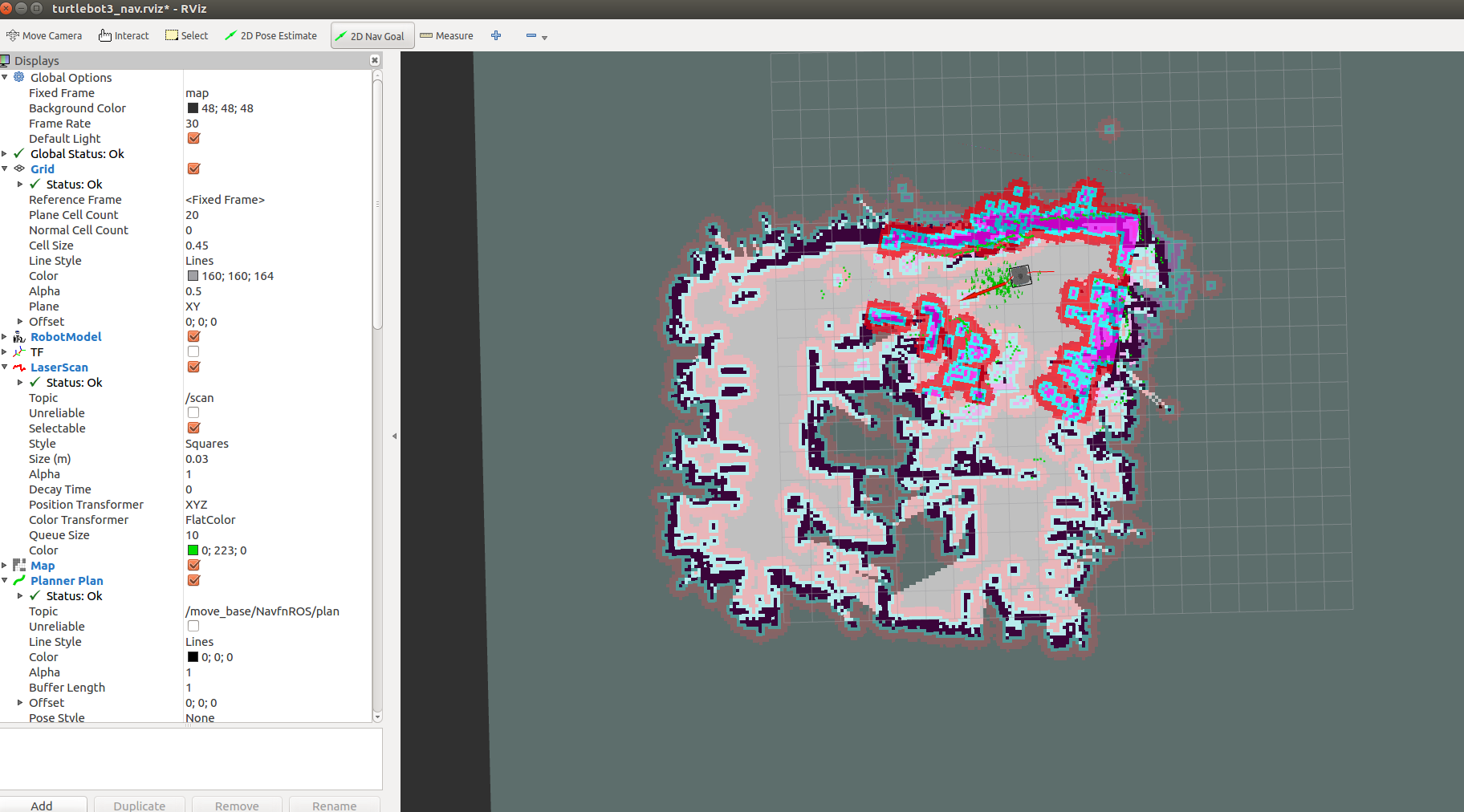
* 点击工具栏 2D Pose Estimate 按键。
* 在地图上点击机器人相对于环境的正确位置并拖动箭头以确定机器人的姿态，可以通过多次调整确定最合适的初始位置设定。

完成设定后的绿色箭头便表示了机器人相对于地图所处的位置和初始姿态，机器人上的激光传感器同时会测量周围环境，可以通过实时测量的环境信息对初始位置进行校对（红点）。如果发现位置不正确可以重复定位操作。

完成定位工作之后，需要设定目标位置和目标姿态，设定方法如下：

* 点击工具栏 2D Nav Goal 按键
* 点击一个确定的点并拖动箭头以设定目标位置和机器人最后停止的方向。

实验过程中要求操作者确定好初始位置和目的位置及姿态之后，机器人便会自行计算运行路径并移动至目标点。



提问：根据观察推断机器人路径规划的大致计算方法？

实验常见问题

本次实验建立在完整的连接和已有的控制命令中，不需要操作者参与计算，操作较为简单，可以在熟悉操作的情况下没有障碍的完成时实验。

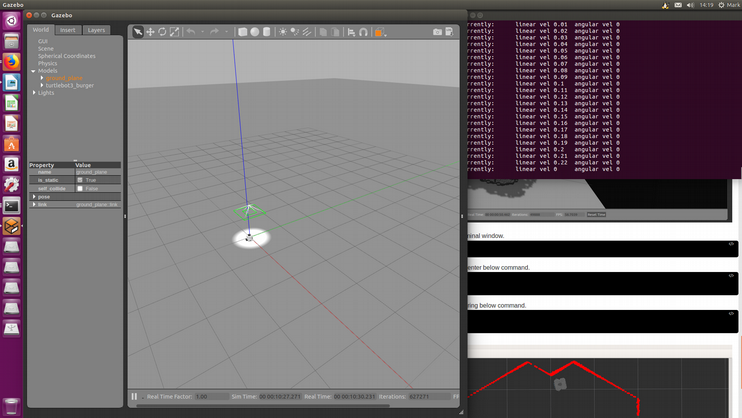
主要会出现的情况有：如果没有确定好初始位置开始进行扫描测量地形，其地图初始点未知，在导航实验中如果不能从原出发点开始运行，会产生地图定位不准的问题，所以在一定程度上也会影响到内部程序对轨迹规划的运算，因为在导航功能中小车的轨迹规划是同时计算原测量地图和实时雷达扫描障碍物的最优解。同时，在实验中杂乱的周边环境对导航功能也会产生较大影响。所以总的来说，小车的运行原理和计算能力对实验场地提出了一定要求。

实验三 模拟控制和GAZEBO虚拟世界构建

实验准备

本实验基于ubuntu系统的Gazebo模拟软件，在电脑上运行模拟程序。只需要远程电脑一台和正确的网络连接环境。

Simulation 步骤参考：[http://emanual.robotis.com/docs/en/platform/turtlebot3/simulation/#gazebo-3d](http://emanual.robotis.com/docs/en/platform/turtlebot3/simulation/" \l "gazebo-3d)

Gazebo软件介绍：Ubuntu环境下的模拟软件，可以建立空间和实体，使用turtlebot提供的调用程序可以使用键盘控制模拟机器人在模拟空间中的运行和探测。其软件界面和操作效果如下图所示：

首先需要做的事下载克隆模拟依赖包，一下命令便描述了安装包应该下载的位置和来源，最后经过构建程序（catkin\_make）完成安装。

$ cd ~/catkin\_ws/src/

# 进入工作空间下的资源文件夹(/src)

$ git clone <https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git>

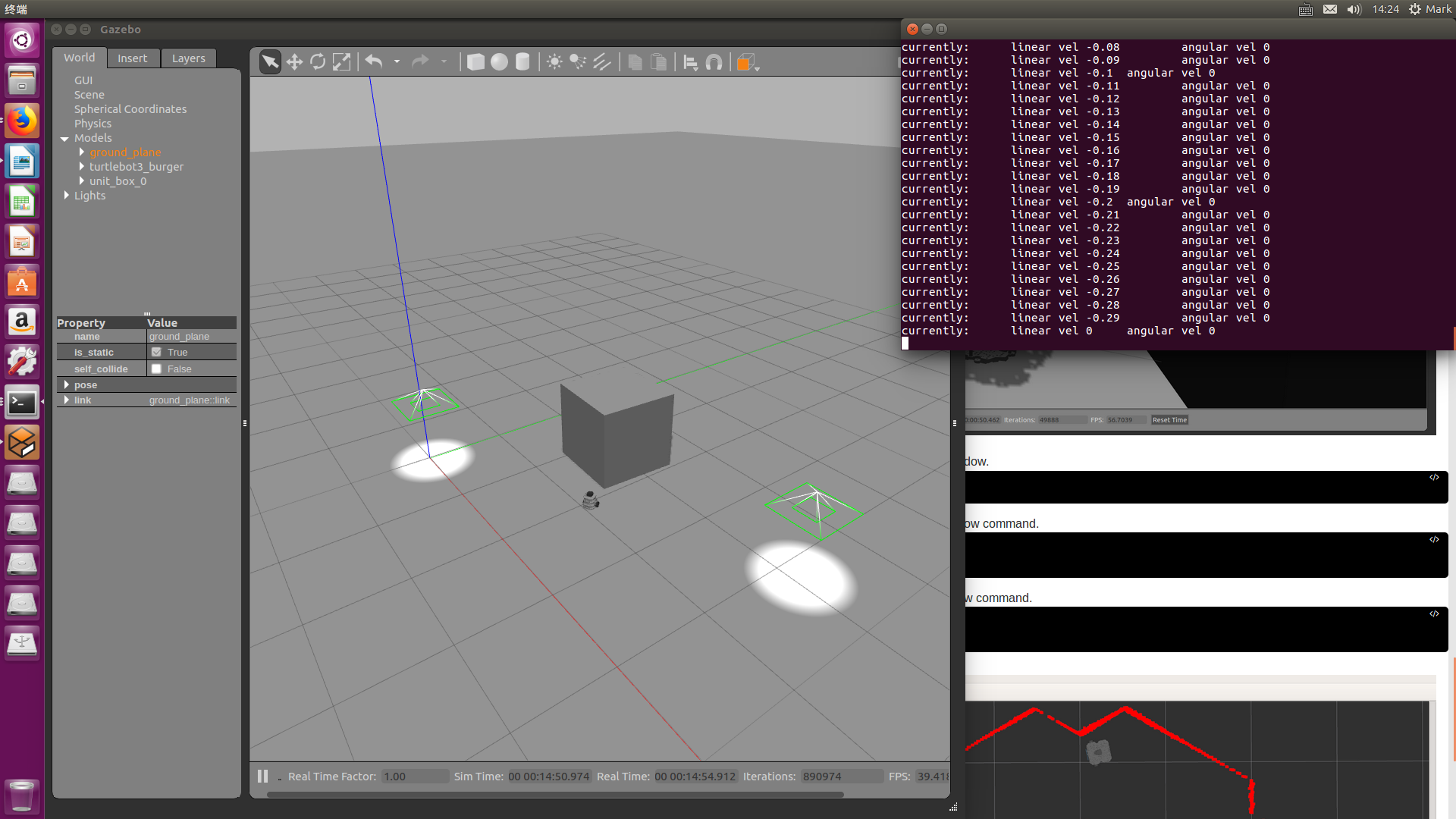
# git 是链接github的工具 clone 为copy指令 目的在于下载链接的内容

$ cd ~/catkin\_ws && catkin\_make

# 进入工作空间主文件夹，make新下载的软件包（可以理解为整理软件内容 注册相关信息以便调用）

这里使用的方法是在ROS中建立一个虚假节点TurtleBot3 fake node ，不需要实体机器人也可以进行通信和控制，你甚至可以通过无线控制节点在RViz中对虚拟机器人进行控制。

同样在运行之前先声明需要加载的3D模型

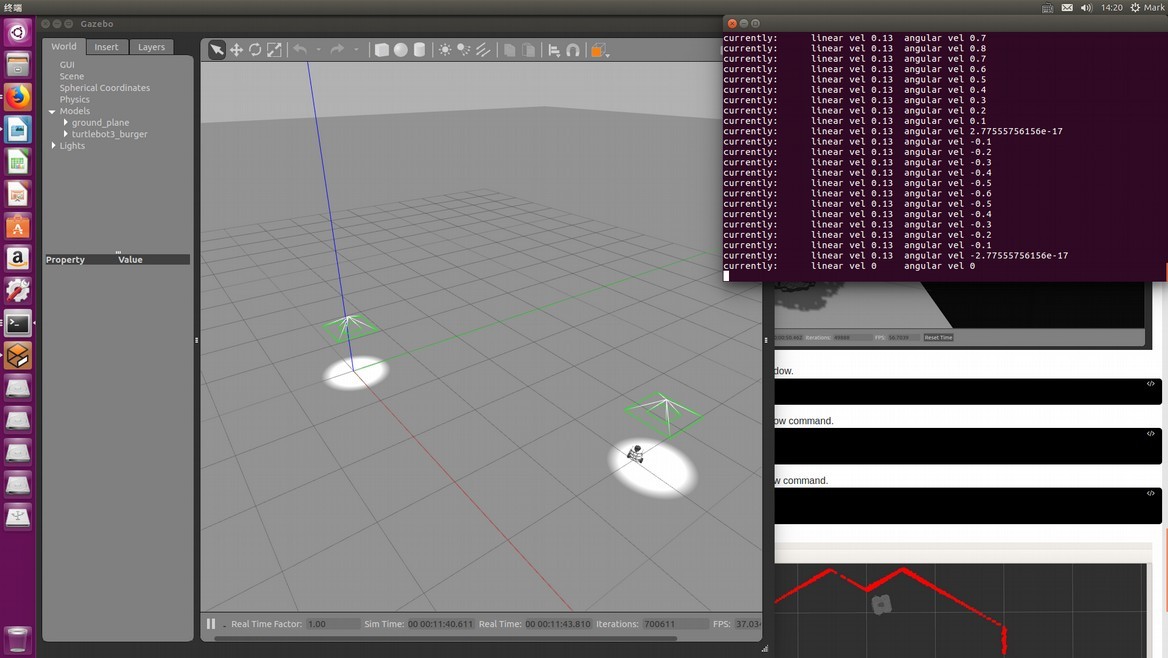


$ export TURTLEBOT3\_MODEL= waffle

$ roslaunch turtlebot3\_fake turtlebot3\_fake.launch

然后建立无线控制节点

$ roslaunch turtlebot3\_teleop turtlebot3\_teleop\_key.launch

如上图所示，使用键盘控制程序可以操作小车在已建立好的世界平面内进行运动。模拟效果相当直观简洁，控制操作同操作实体一样，而且在速度和运动空间上有更大自由，能达到较高的速度，同时还可以自己建立虚拟实体（如下图所示），且可以设定实体大小位置质量固定性等参数，模拟小车与实体接触的不同情况。

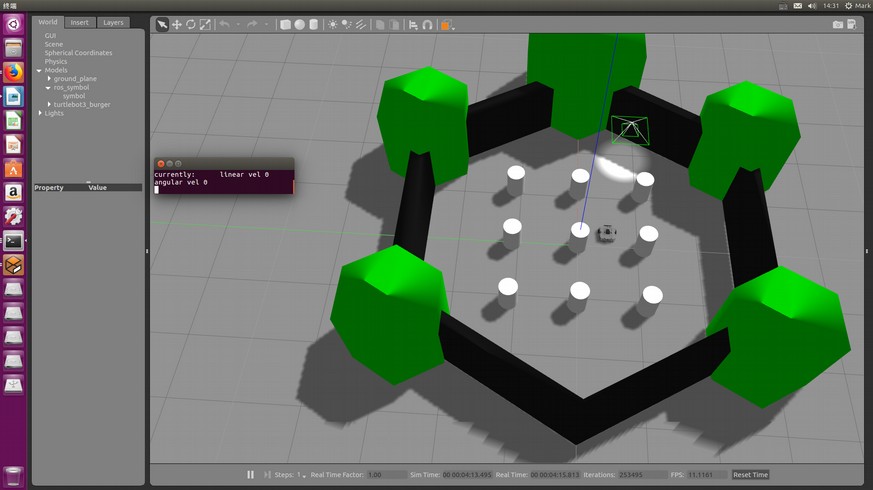
如下图所示，在软件中可以建立空白空间或者调用已经建立好的运动空间：

$ export TURTLEBOT3\_MODEL=waffle

$ roslaunch turtlebot3\_gazebo turtlebot3\_empty\_world.launch

或者

$ roslaunch turtlebot3\_gazebo turtlebot3\_world.launch



为了能够控制机器人同样需要在新的终端中创建使用键盘控制的远程控制节点

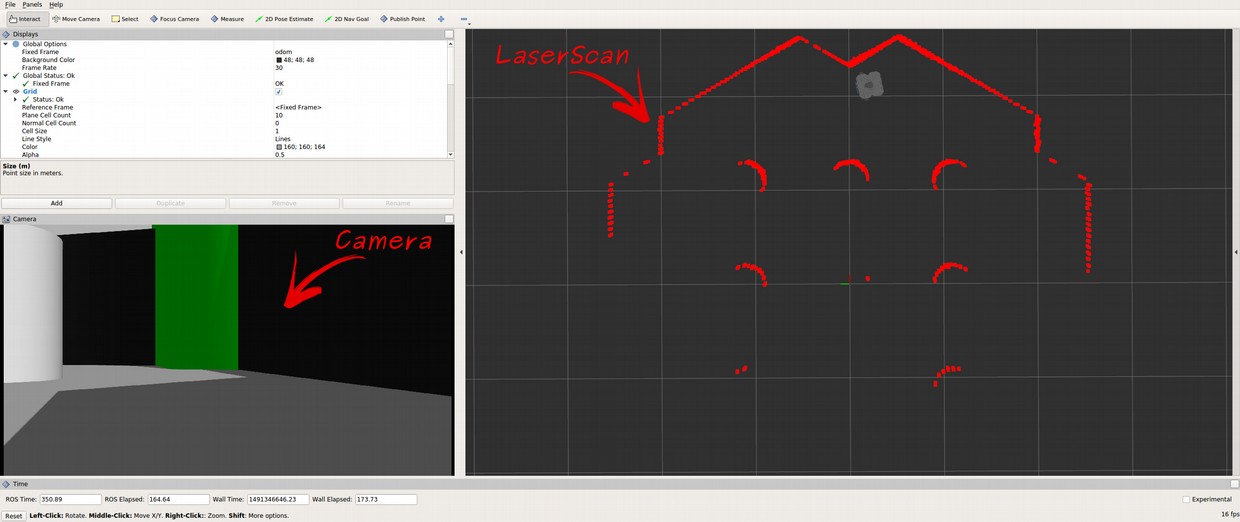
$ roslaunch turtlebot3\_teleop turtlebot3\_teleop\_key.launch

在建立好的控件里面，turtlebot已经提供了软件接口，可以只接调用键盘控制程序控制Gazebo软件下的模拟器，对模拟环境下的使用键盘进行控制。

还可以调用已经提供的命令接入模拟的激光传感器进行探测环境的点云可视化。模拟环境下的激光雷达探测环境界面如下图所示

$ export TURTLEBOT3\_MODEL=waffle

$ roslaunch turtlebot3\_gazebo turtlebot3\_gazebo\_rviz.launch



同时如果需要在虚拟环境中实现导航功能可以在新的终端中输入命令：

$ export TURTLEBOT3\_MODEL=waffle

$ roslaunch turtlebot3\_gazebo turtlebot3\_simulation.launch

模拟器具有良好的开放性，可以自己建立运行环境对不同环境进行模拟，可以很好的扩展机器人操作的视野，对解决真实环境下的机器人探测导航等问题提供生动具体的认识。

实验四 视觉循线跟踪实验

实验准备

前几章的内容集中在演示机器人的运作过程，使得同学们对于移动机器人有一个初步的了解。目前实现的功能实际上是开环系统，发送命令之后并没有反馈调节过程。在本章实验中，我们将会使用机器人上的传感器来建立一个闭环的控制系统，实现计算误差并对控制系统进行反馈调节，以减小个方面参数的误差。

本章中将创建一个可以使用相机来跟随地面上的轨迹的机器人。使用到**Opencv**库来实现图像处理功能。整个实验流程分下面几步工作：

* 从相机提取实时的图像信息，压缩后转发信息给Opencv
* 建立图像蒙板来获取路线中心
* 发送命令给执行器，使得机器人保持在路线的中心上

通过这三步就能建立一个带有反馈的移动机器人控制系统，教程将在模拟器的环境下进行，实验任务要求同学能够结合turtlebot实体进行控制。

实验内容：

1 获取图像

在ROS系统中，图像信息往往是通过sensor\_msg/Image 的数据类型进行传递的。为了获得图像信息我们需要对图像信息的发布标题(Topic)进行订阅(Subscribe)， 不同的机器人可能会使用不同的标题进行信息发布，这里我们将对标题进行查阅以确定收取的信息。

让我们开始实验，首先需要启动三个终端(Ctrl-Alt-T) : 一个运行roscore、一个启动gazebo模拟环境、一个用来进行相关的交互命令。

首先打开终端运行roscore：

user@hostname$ roscore

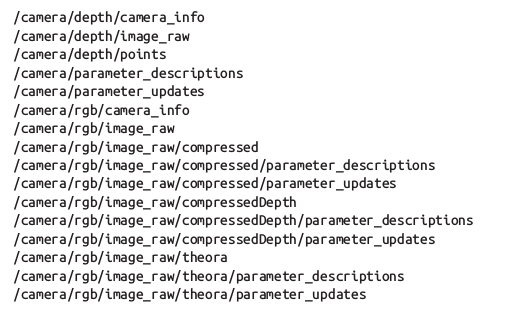
接着运行虚拟环境：

user@hostname$ roslaunch turtlebot\_gazebo turtlebot\_world.launch

在新的终端中，我们使用shell命令，如果是首次接触，我们需要了解机器人到底发布了哪些消息。所以需要先列表：

user@hostname$ rostopic list

终端会出现很多topic，根据名称可以看到与图像相关的有一下内容：

根据图示内容可以看到相机包含camera/depth/的深度信息(由深度相机提供信息，kinect相机包含双目深度视觉模块)，以及camera/rgb/的彩色图像信息，以及各种参数(info, description, updates)。我们需要处理的是彩色图像信息，有camera/rgb/image\_raw 和 …/image\_raw/compressed 两个topic提供。/camera/rgb/image\_raw/theora则传送视频信息，一般作为监控视频使用能有效降低传输数据量，但有一定延时性并大量占用处理进程空间，一般不作为图像处理对象。

这里要说明的是，如果程序处理过程是在机器人主机上实现的话，一般会直接对原图信息进行处理，也就是建立一个Subsceriber去接受/camera/rgb/image\_raw，本实验中因为运行环境为有一定带宽限制的WIFI链接，所以将建立一个Subscriber订阅消息，使用/camera/rgb/image\_raw/compressed 作为图像处理对象。

此时我们可以在你建立的package的scripts文件夹中建立一个新的python脚本文件，接下来可以自行编写程序来接受图像信息了：

(本例中使用未压缩图像，请注意与后面程序进行区分！)

### Exp 4-1 follwer.py

 程序在接受Image之后调用image\_callback函数，但是什么都没有做喔：）

为了能够直接对python脚本进行调用，进入文件目录，在终端中可以使用以下方法：

$ chmod +x follower.py

$ ./follower.py

说明：chmod 命令用于修改文件权限Modifies the permissions of a file，对于 已经有权限的文件，可以加上 ‘./’的前缀直接运行。

注：对于一般的python文件也可以通过符合bash格式的语句进行调用，如：

$ python follower.py

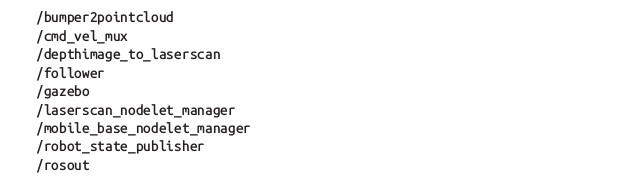
本程序不会有任何输出，因为在callback函数中没有对msg进行处理，不过通过建立基本的Subscriber，相信大家也对程序的基本结构有了一定的了解。

接下来我们来测试ROS系统的工具：

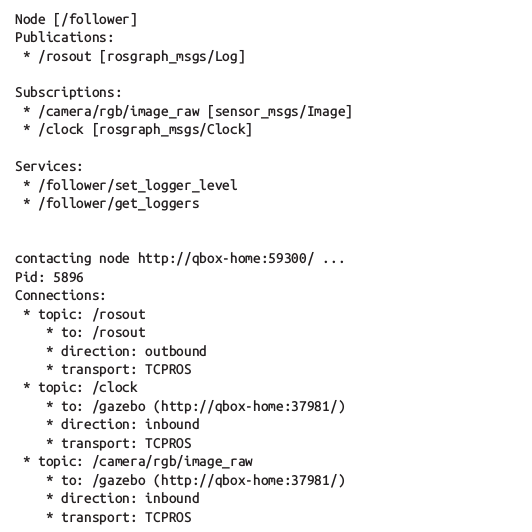
$ rosnode list

rosnode说明：

终端会返回一系列ROS节点列表，通过列表可以了解建立了多少节点，在整个ROS信息流中有哪些节点是活跃的。

其中/follower节点便是我们通过rospy.init\_node('follower')这一行命令建立的 ，其他节点分别由roslaunch 命令、roscore命令建立。通过使用info命令我们可以看到发布节点的具体信息：

$ rosnode info /follower



以上便是一个较为完整的节点信息，其中包含了节点所发布方向的下级节点/rosout，本节点所接受的上级节点 /camera/rgb/image\_raw，/clock，以及他们使用的数据类型[rosgraph\_msgs/Log、sensor\_msgs/Image、rosgraph\_msgs/Clock等]

通过这一步操作同学们可以简单的了解到如何建立和接受另外节点的消息的方法，接下来得步骤会更加有趣。为了了解我们建立的follower.py 程序到底在干什么，首先我们来看节点接受消息的速度如何，一个简单的命令就可以搞定：

$ rostopic hz /camera/rgb/image\_raw

rostopic hz 前面章节中已经介绍过了，用来检测消息标题的发布频率，我们可以看到类似一下图片的消息流：



通过测试 可知：图像是以20帧的速率被/camera/rgb/image\_raw 发布出来。

现在我们知道follower已经在接受数据消息了，那么接下来将会对数据进行处理，常用的方法是调用Opencv库（如果想自己写图像处理算法也是相当欢迎的！），在Opencv中有众多开源的高效的机器视觉算法内容，为了从ROS到Opencv建立良好的数据链接，我们可以使用cv\_bridge package，在这个package中包含着从sensor\_msgs/Image 转换到Opencv对象的数据格式转换函数。

Exp4-2 旨在教你如何将图像消息转换成Opencv消息对象，然后在频幕上显示出来。

### Exp4-2 follower\_opencv.py

#!/usr/bin/env python

import rospy

from sensor\_msgs.msg import CompressedImage # 本例代入的是压缩图像喔 ！

import cv2, cv\_bridge # 导入opencv模块和 协议模块

class Follower: # 设定一个名为Follower的类（也可以是别的代码风格）

def \_\_init\_\_(self):

self.bridge = cv\_bridge.CvBridge()

self.image\_sub = rospy.Subscriber(

'/camera/rgb/image\_raw/compressed',

CompressedImage,self.image\_callback)

# 以上括号内分别是 接受的node名称和 数据类型 ，以及触发的回调函数

def image\_callback(self,msg):

image = self.bridge.compressed\_imgmsg\_to\_cv2(msg,desired\_encoding="bgr8")

#类参数bridge转换函数

print image.size, image.shape

#本句目的输出图像大小已经形状

cv2.namedWindow("window2",cv2.WINDOW\_NORMAL)

#简历名为window2的窗口 窗口类为 cv2内置的NORMAL窗口

cv2.imshow("window2",image[:,:,0])

# 显示二维图像

cv2.waitKey(3)

#以上函数每次接受消息都会被调用

rospy.init\_node('follower',anonymous = True)

# anonymous 同步选项 每公布一条消息就接受一个消息

follower = Follower()

#调用类 函数会被运行

rospy.spin()

通过运行以上程序，你可以在桌面上见到图像传输窗口，前提是你已经运行了roscore和gazebo虚拟环境：） 此时gazebo正在不断计算图像信息并发布到信息流中，被你建立的follower节点接受，并通过imshow()函数显示出来。

2.建立自己的软件包！

终于各位可以真正做一点整齐的事情了！下面我们来为了实验建立一个软件包，package用来装你将创立的文件。（creat a package的具体内容和技术细节请见说明）

显然只是看看地平线是不够的，我们需要进入本章实验的正题：循线跟踪。为了完成着一个目标我们需要简历一个包含多个模块的launch文件，就像名字一样，作为程序的发射器，\*.launch管理着一个功能模块所需要的软件包和各种参数。一下是本实验中的主程序文件：course.launch

<launch>

<arg name="model" default="$(env TURTLEBOT3\_MODEL)" doc="model type [burger, waffle, waffle\_pi]"/>

<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">

####################### need change #########################

<arg name="world\_name" value="$(find yourpack)/world/course.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.py $(find turtlebot3\_description)/urdf/turtlebot3\_$(arg model).urdf.xacro" />

<node pkg="robot\_state\_publisher" type="robot\_state\_publisher" name="robot\_state\_publisher">

<param name="publish\_frequency" type="double" value="30.0" />

</node>

<node name="spawn\_urdf" pkg="gazebo\_ros" type="spawn\_model" args="-urdf -model turtlebot3 -x $(arg x\_pos) -y $(arg y\_pos) -z $(arg z\_pos) -param robot\_description" />

</launch>

course.launch说明：

<model>获取MODEL名称设定初始位置

<include>调用文件所包含的底层模块

<gazebo/empty>驱动空世界 设定世界名为course 并设定世界course.world/在该文件中引用 worldmeterial/接着引用地图course.png作为地图。（该过程有兴趣的同学可以自己摸索文件包的各项文件信息，可以自己更改地图形状喔）

<robot\_description> 设定turtlebot模型描述文件

<robot\_state\_publisher> 建立node发送状态信息（scan/tf）

<spawn\_urdf> 根据机器人描述文件建立robote\_model进行显示 缺少会无法建立可视模型。

以上程序并不能正确运行，请根据自己建立的package名称修改注释符中的语句：）

接下来就可以开启虚拟环境了！

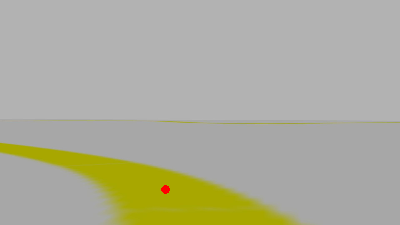
$ roslaunch ‘yourpack\_name’ course.launch

此时我们便打开了一个虚拟的循线实验环境，其界面如下图所示。

循线作为一个移动机器人的基本功能，在移动机器人的开发上面具有十分重要的作用，小到CGV小车，大到无人驾驶，都会考虑到与地面标志相关的应用环境，在车道线或者地面指引线上运动，是学习移动机器人的必经之路。所以接下来请同学们认真学习本实验的例程，并作出自己的改动的改进。

3识别引导线

循线任务的第一步是识别地面线路。我们将使用Opencv库来对图像进行处理，上一节中我们已经能够获得图像消息，并建立了自己的launch文件，本节的目标是改进follower的功能，下图是一般情况下的图像窗口界面。

有很多方法可以用来检测路线，本课程中因为处理对象是非常均匀的路线，而且环境中也没有太多的干扰信息，所以采用一种较为简单的方法做处理：蒙版->切片->求中心。通过程序的阅读和编写，希望同学可以学习到如何接受节点消息并将信息流从ROS转换到python库中进行处理的方法，通过这样的训练可以给你一些启发并能够使用更多优秀功能更强大的Opencv库函数。

第一步，建立颜色蒙版。简单的方法就是对图想抽取像素，对我们需要的颜色进行筛选。不过不同于RGB三原色的图像模式，在ROS中我们使用的是HSV图像，其中包含三个参数（hue）颜色、(saturation)饱和度，和(value)亮度。HSV制式相对与RGB有更好的抗干扰性，降低环境亮度和阴暗变化的影响。经过图像格式转化之后我们可以设置颜色阈值来挑选出图像上的路径线路（黄色部分）。以下代码将会实现这样的操作。请在scripts文件夹下，建立新文件follower\_color\_filter.py

#!/usr/bin/env python

import rospy ,cv2, cv\_bridge , numpy

from sensor\_msgs.msg import CompressedImage

# 导入模块

class Follower:

def \_\_init\_\_(self):

self.bridge = cv\_bridge.CvBridge()

self.image\_sub = rospy.Subscriber('/camera/rgb/image\_raw/compressed',CompressedImage,self.image\_callback)

# 函数变量分别为：接受节点 、 接受消息类型、 回调函数

def image\_callback(self,msg):

image = self.bridge.compressed\_imgmsg\_to\_cv2(msg)

hsv = cv2.cvtColor(image, cv2.COLOR\_BGR2HSV)

##### 进行消息-> 变量image -> hsv格式变量 的两步转换

lower\_yellow = numpy.array([20,50,150])

upper\_yellow = numpy.array([50,255,190])

##### 建立蒙版参量 参量使用指针格式（inRange函数的要求）

mask = cv2.inRange(hsv, lower\_yellow, upper\_yellow)

masked = cv2.bitwise\_and(image,image,mask=mask)

##### 使用蒙版进行二值化 bitwise

cv2.namedWindow("window2",cv2.WINDOW\_NORMAL)

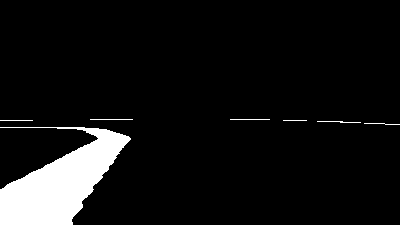
cv2.imshow("window2",mask) #进行显示

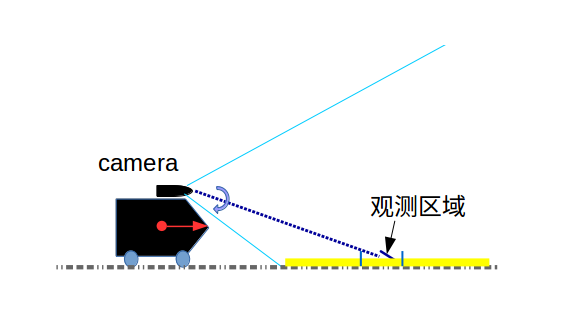
cv2.waitKey(3)

rospy.init\_node('follower',anonymous = True)

follower = Follower()

rospy.spin()

输出如上图所示图像，黄色部分被选取出来(1)，其他部分被设置为黑色(0)。这只是第一步二值化操作，选出需要的对象之后，我们需要跟踪线，如何确定我们需要跟踪的对象呢？ 我们只需要选取高度为20个(或30个)像素点的区域作为计算对象就可以了。着背后的原理是，图像反映的是摄像头所看见的内容，其中地平线一下的才是地面标志线，对图像进行有效的切片可以帮助我们确定跟踪对象(黄线)的合理距离，我们可以只考虑距离机器人1米的黄线，对这一块的对象进行跟踪就能满足功能需求。 示意图如下：

然后对选取合适的区域进行切片，并标记中心点，经过计算的中心点偏离视频中轴的距离将作为Error参数进行输出。当系统能够测算出Error之后，学习过经典控制理论的同学，相信你就知道我们就可以简历一个有反馈调节的控制系统了：）

例程Exp4-3 在follower\_color\_filter 的基础上完成了上面所描述的功能：

### Exp4-3 follower\_line\_finder.py

#!/usr/bin/env python

import rospy ,cv2, cv\_bridge , numpy

from sensor\_msgs.msg import CompressedImage

class Follower:

def \_\_init\_\_(self):

self.bridge = cv\_bridge.CvBridge()

self.image\_sub = rospy.Subscriber(

'/camera/rgb/image\_raw/compressed',

CompressedImage,

self.image\_callback )

def image\_callback(self,msg):

image = self.bridge.compressed\_imgmsg\_to\_cv2(msg,

desired\_encoding='bgr8')

hsv = cv2.cvtColor(image, cv2.COLOR\_BGR2HSV)

lower\_yellow = numpy.array([20,50,150])

upper\_yellow = numpy.array([50,255,190])

mask = cv2.inRange(hsv, lower\_yellow, upper\_yellow)

# 同Exp4-2相同 不多做介绍

h,w,d= image.shape

top = 3\*h/4

bot = top + 20

# 在图像的下3/4处进行切片 注意：image纵向向下为x正向 横向向右为y正向

mask[0:top,:] = 0

mask[bot:h,:] = 0

M = cv2.moments(mask)

#MOMENTS Opencv中用于计算图像中心的函数类 可以查看官网：）

if M['m00']>0:

cx = int(M['m10']/M['m00'])

cy = int(M['m01']/M['m00'])

cv2.circle(image,(cx,cy),20,(0,0,255),-1)

# 对切片之后的图像计算中心，并标记小圆圈

cv2.namedWindow("window2",cv2.WINDOW\_NORMAL)

cv2.imshow("window2",image)

cv2.namedWindow("window",cv2.WINDOW\_NORMAL)

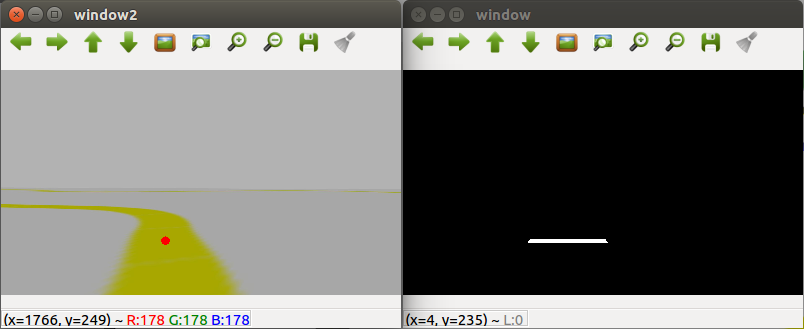
cv2.imshow("window",mask)

cv2.waitKey(3)

rospy.init\_node('follower',anonymous = True)

follower = Follower()

rospy.spin()

运行之后的结果如图所示，为了更好的观察图像的效果，可以在新的终端中运行第二章中的teleop函数，对小车进行控制，了解其动态效果。

4-4循线跟踪

前面小节中已经完成了识别标记的工作，接下来只需要简历负反馈的控制系统，便可以让小车跟踪地面线路运动了。本例只包含PID算法中的比例增益环节，希望同学通过学习，可以加入PID算法对小车进行更有效的控制。(请自行查阅增量式PID算法的相关内容)

### Exp4-5 follower\_ proporation.py

#!/usr/bin/env python

import rospy ,cv2, cv\_bridge , numpy

from sensor\_msgs.msg import CompressedImage

from geometry\_msgs.msg import Twist

class Follower:

def \_\_init\_\_(self):

self.bridge = cv\_bridge.CvBridge()

self.image\_sub = rospy.Subscriber('/camera/rgb/image\_raw/compressed',CompressedImage,self.image\_callback)

self.cmd\_vel\_pub = rospy.Publisher('cmd\_vel',Twist,queue\_size =1)

self.twist =Twist()

def image\_callback(self,msg):

image = self.bridge.compressed\_imgmsg\_to\_cv2(msg,desired\_encoding='bgr8')

hsv = cv2.cvtColor(image, cv2.COLOR\_BGR2HSV)

lower\_yellow = numpy.array([20,50,150])

upper\_yellow = numpy.array([50,255,190])

mask = cv2.inRange(hsv, lower\_yellow, upper\_yellow)

h,w,d= image.shape

top = 5\*h/6

bot = top + 20

mask[0:top,:] = 0

mask[bot:h,:] = 0

#cut the image to a blade

M = cv2.moments(mask)

#class MOMENTS

if M['m00']>0:

cx = int(M['m10']/M['m00'])

cy = int(M['m01']/M['m00'])

cv2.circle(image,(cx,cy),20,(0,0,255),-1)

err = cx-w/2

self.twist.linear.x = 0.2

self.twist.angular.z = -float(err)/1000

self.cmd\_vel\_pub.publish(self.twist)

cv2.namedWindow("window2",cv2.WINDOW\_NORMAL)

cv2.imshow("window2",image)

cv2.waitKey(3)

rospy.init\_node('follower',anonymous = True)

follower = Follower()

rospy.spin()

### 作业：

1.简历自己的package完成本节实验内容

2.对最后的程序逐行注释，并对实验结果进行说明

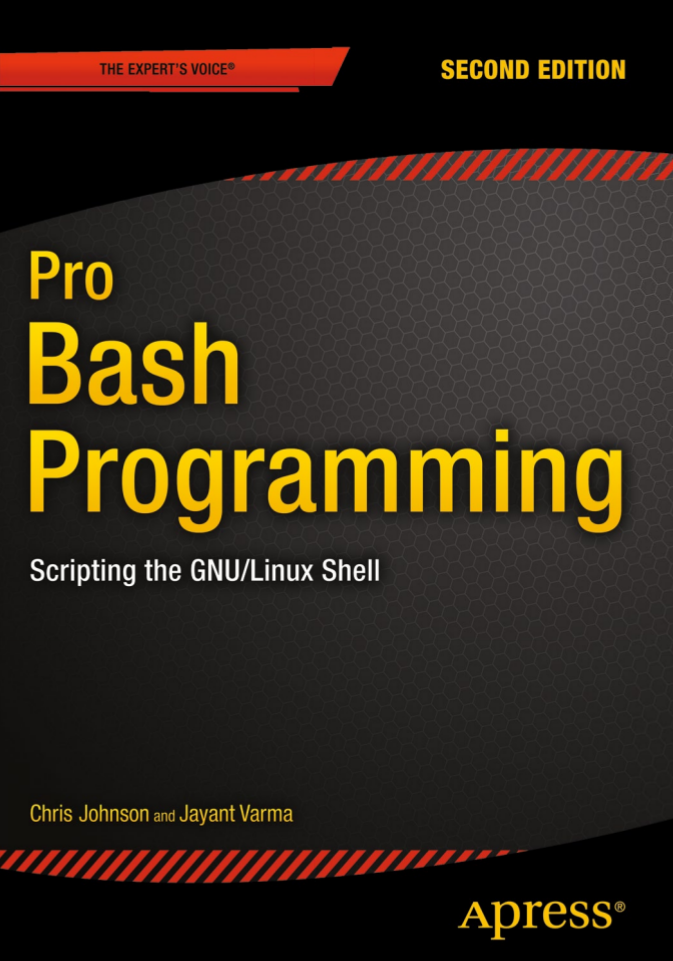
3.撰写实验报告

加分项：

* 修改Exp4-1的接受topic和数据类型
* follower\_proportional.py建立PID控制模型
* 绘制自己的地图并修改.world文件

综合实验设计

附录1 参考资料

实验参考资料见：

[https://github.com/markdole/Hust\_Exp\_File](https://github.com/markdole/Hust_Exp_File/blob/master/2015_Book_ProBashProgramming.pdf)

Bash Programming：

[2015\_Book\_ProBashProgramming.pdf](https://github.com/markdole/Hust_Exp_File/blob/master/2015_Book_ProBashProgramming.pdf)

请参阅第一节的内容，学习如何在终端中输入基本命令，了解语句格式和使用方法。

Bash语言指引参考资料：

<https://bash.cyberciti.biz/guide/Main_Page>

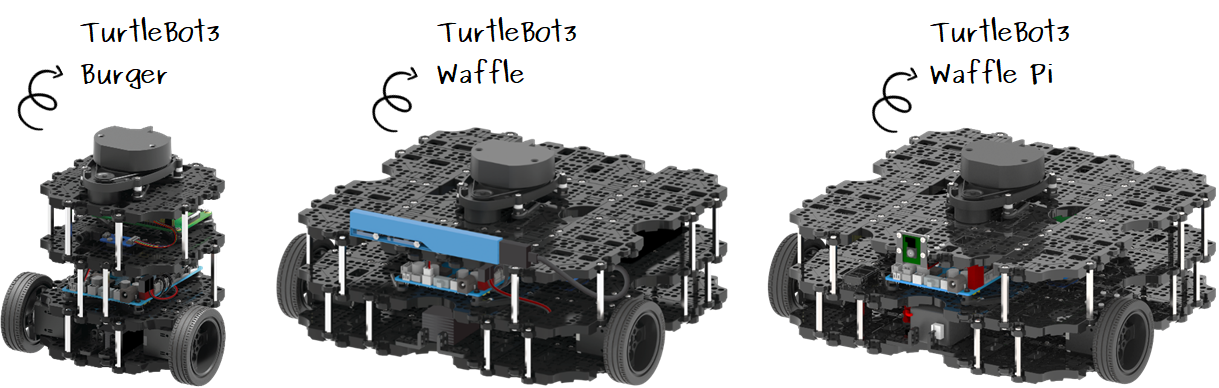
Shell 脚本30分钟入门指南（中文）：

<https://github.com/qinjx/30min_guides/blob/master/shell.md>

实验源代码见：

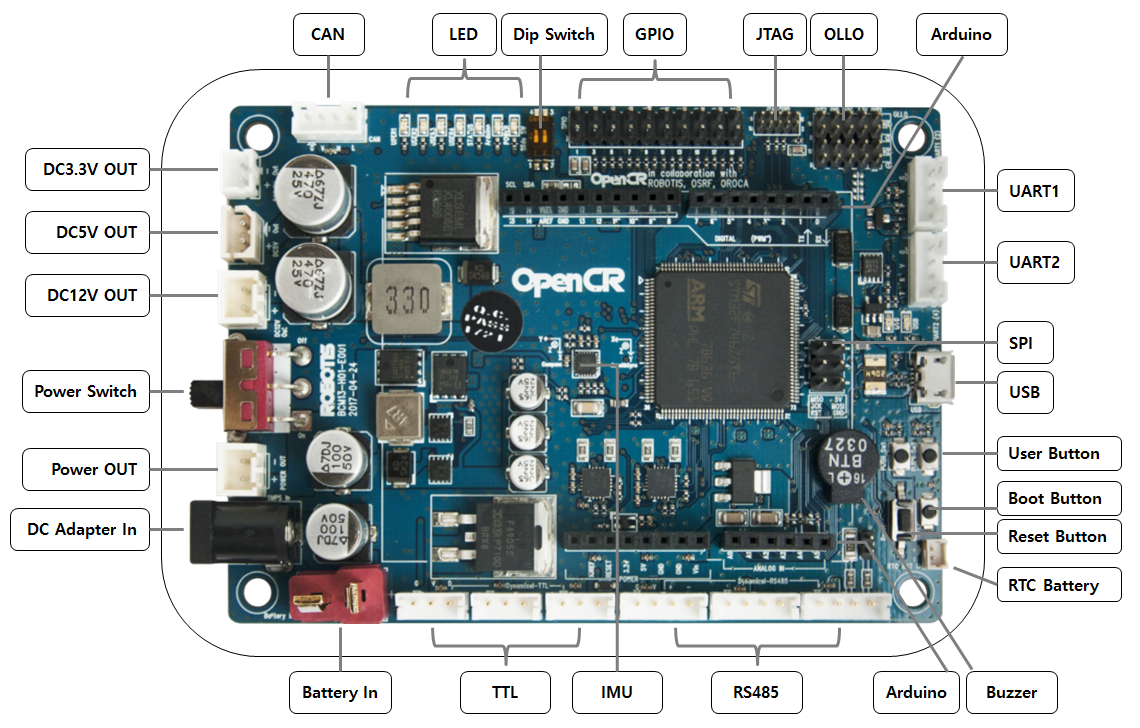
<https://github.com/markdole/Hust_exp6.git>

附录1



| **Items** | **Burger** | **Waffle** | **Waffle Pi** |
| --- | --- | --- | --- |
| Maximum translational velocity | 0.22 m/s | 0.26 m/s | 0.26 m/s |
| Maximum rotational velocity | 2.84 rad/s (162.72 deg/s) | 1.82 rad/s (104.27 deg/s) | 1.82 rad/s (104.27 deg/s) |
| Maximum payload | 15kg | 30kg | 30kg |
| Size (L x W x H) | 138mm x 178mm x 192mm | 281mm x 306mm x 141mm | 281mm x 306mm x 141mm |
| Weight (+ SBC + Battery + Sensors) | 1kg | 1.8kg | 1.8kg |
| Threshold of climbing | 10 mm or lower | 10 mm or lower | 10 mm or lower |
| Expected operating time | 2h 30m | 2h | 2h |
| Expected charging time | 2h 30m | 2h 30m | 2h 30m |
| SBC (Single Board Computers) | Raspberry Pi 3 Model B | Intel® Joule™ | Raspberry Pi 3 Model B |
| MCU | 32-bit ARM Cortex®-M7 with FPU (216 MHz, 462 DMIPS) | 32-bit ARM Cortex®-M7 with FPU (216 MHz, 462 DMIPS) | 32-bit ARM Cortex®-M7 with FPU (216 MHz, 462 DMIPS) |
| Remote Controller | - | - | RC-100B + BT-410 Set (Bluetooth 4, BLE) |
| Actuator | Dynamixel XL430-W250 | Dynamixel XM430-W210 | Dynamixel XM430-W210 |
| LDS(Laser Distance Sensor) | 360 Laser Distance Sensor LDS-01 | 360 Laser Distance Sensor LDS-01 | 360 Laser Distance Sensor LDS-01 |
| Camera | - | Intel® Realsense™ R200 | Raspberry Pi Camera Module v2.1 |
| IMU | Gyroscope 3 Axis Accelerometer 3 Axis Magnetometer 3 Axis | Gyroscope 3 Axis Accelerometer 3 Axis Magnetometer 3 Axis | Gyroscope 3 Axis Accelerometer 3 Axis Magnetometer 3 Axis |
| Power connectors | 3.3V / 800mA 5V / 4A 12V / 1A | 3.3V / 800mA 5V / 4A 12V / 1A | 3.3V / 800mA 5V / 4A 12V / 1A |
| Expansion pins | GPIO 18 pins Arduino 32 pin | GPIO 18 pins Arduino 32 pin | GPIO 18 pins Arduino 32 pin |
| Peripheral | UART x3, CAN x1, SPI x1, I2C x1, ADC x5, 5pin OLLO x4 | UART x3, CAN x1, SPI x1, I2C x1, ADC x5, 5pin OLLO x4 | UART x3, CAN x1, SPI x1, I2C x1, ADC x5, 5pin OLLO x4 |
| Dynamixel ports | RS485 x 3, TTL x 3 | RS485 x 3, TTL x 3 | RS485 x 3, TTL x 3 |
| Audio | Several programmable beep sequences | Several programmable beep sequences | Several programmable beep sequences |
| Programmable LEDs | User LED x 4 | User LED x 4 | User LED x 4 |
| Status LEDs | Board status LED x 1 Arduino LED x 1 Power LED x 1 | Board status LED x 1 Arduino LED x 1 Power LED x 1 | Board status LED x 1 Arduino LED x 1 Power LED x 1 |
| Buttons and Switches | Push buttons x 2, Reset button x 1, Dip switch x 2 | Push buttons x 2, Reset button x 1, Dip switch x 2 | Push buttons x 2, Reset button x 1, Dip switch x 2 |
| Battery | Lithium polymer 11.1V 1800mAh / 19.98Wh 5C | Lithium polymer 11.1V 1800mAh / 19.98Wh 5C | Lithium polymer 11.1V 1800mAh / 19.98Wh 5C |
| PC connection | USB | USB | USB |
| Firmware upgrade | via USB / via JTAG | via USB / via JTAG | via USB / via JTAG |
| Power adapter (SMPS) | Input : 100-240V, AC 50/60Hz, 1.5A @max Output : 12V DC, 5A | Input : 100-240V, AC 50/60Hz, 1.5A @max Output : 12V DC, 5A | Input : 100-240V, AC 50/60Hz, 1.5A @max Output : 12V DC, 5A |

附录2



| **Items** | **Specifications** |
| --- | --- |
| Microcontroller | STM32F746ZGT6 / 32-bit ARM Cortex®-M7 with FPU (216MHz, 462DMIPS) |
| Sensors | Gyroscope 3Axis, Accelerometer 3Axis, Magnetometer 3Axis (MPU9250) |
| Programmer | ARM Cortex 10pin JTAG/SWD connector USB Device Firmware Upgrade (DFU) Serial |
| Extension pins | 32 pins (L 14, R 18) \*Arduino connectivity Sensor module x 4 pins Extension connector x 18 pins |
| Communication circuits | USB (Micro-B USB connector/USB 2.0/Host/Peripheral/OTG) TTL ([B3B-EH-A](http://www.jst-mfg.com/product/pdf/eng/eEH.pdf) / Dynamixel) RS485 ([B4B-EH-A](http://www.jst-mfg.com/product/pdf/eng/eEH.pdf) / Dynamixel) UART x 2 ([20010WS-04](http://www.alldatasheet.com/datasheet-pdf/pdf/147797/YEONHO/20010WS-04000.html)) CAN ([20010WS-04](http://www.alldatasheet.com/datasheet-pdf/pdf/147797/YEONHO/20010WS-04000.html)) |
| LEDs and buttons | LD2 (red/green) : USB communication User LED x 4 : LD3 (red), LD4 (green), LD5 (blue) User button x 2 |
| Powers | External input source 5 V (USB VBUS), 7-24 V (Battery or SMPS) Default battery : LI-PO 11.1V 1,800mAh 19.98Wh Default SMPS: 12V 5A External output source 12V@1A([SMW250-02](http://www.alldatasheet.com/datasheet-pdf/pdf/148144/YEONHO/SMW250-02P.html)), 5V@4A([5267-02A](http://www.molex.com/molex/products/datasheet.jsp?part=active/0022035025_PCB_HEADERS.xml&channel=Products&Lang=en-US)), 3.3V@800mA([20010WS-02](http://www.alldatasheet.com/datasheet-pdf/pdf/147795/YEONHO/20010WS-02000.html)) External battery Port for RTC (Real Time Clock) ([Molex 53047-0210](http://www.molex.com/molex/products/datasheet.jsp?part=active/0530470210_PCB_HEADERS.xml)) Power LED: LD1 (red, 3.3 V power on) Reset button x 1 (for power reset of board) Power on/off switch x 1 |
| Dimensions | 105(W) X 75(D) mm |
| Mass | 60g |



| **Items** | [**XL430-W250**](http://support.robotis.com/en/product/actuator/dynamixel_x/xl_series/xl430-w250.htm)**(for Burger)** | [**XM430-W210**](http://support.robotis.com/en/product/actuator/dynamixel_x/xm_series/xm430-w210.htm)**(for Waffle)** |
| --- | --- | --- |
| Microcontroller <td colspan=2> ST CORTEX-M3 (STM32F103C8 @ 72Mhz, 32bit) </td> |  |  |
| Position Sensor <td colspan=2> Contactless Absolute Encoder (12bit, 360°) </td> |  |  |
| Motor | Cored Motor | Coreless Motor |
| Baud Rate | 9600 bps ~ 4.5 Mbps |  |
| Control Modes | Velocity, Position, Extended Position, PWM | Velocity, Position, Extended Position, PWM, Current, Current-base Position |
| Gear Ratio | 258.5 : 1 | 212.6 : 1 |
| Stall Torque | 1.0 N.m (@ 9V, 1A) | 2.7 N.m (@ 11.1V, 2.1A) |
|  | 1.4 N.m (@ 11.1V, 1.3A) | 3.0 N.m (@ 12V, 2.3A) |
|  | 1.5 N.m (@ 12V, 1.4A) | 3.7 N.m (@ 14.8V, 2.7A) |
| No Load Speed | 47rpm (@ 9V) | 70rpm (@ 11.1V) |
|  | 57rpm (@ 11.1V) | 77rpm (@ 12V) |
|  | 61rpm (@ 12V) | 95rpm (@ 14.8V) |
| Communication | TTL Level Multi Drop Bus | TTL Level / RS485 Multi Drop Bus |
| Material | Engineering Plastic | Full Metal Gear, Metal Body, Engineering Plastic |
| Standby Current | 52mA | 40mA |



* 360 Laser Distance Sensor LDS-01 is a 2D laser scanner capable of sensing 360 degrees that collects a set of data around the robot to use for SLAM (Simultaneous Localization and Mapping) and Navigation.
* The LDS-01 is used for TurtleBot3 Burger, Waffle and Waffle Pi models.
* It supports USB interface(USB2LDS) and is easy to install on a PC.
* It supports UART interface for embedded baord.

| * **Items** | **Specifications** |
| --- | --- |
| Operating supply voltage | 5V DC ±5% |
| Light source | Semiconductor Laser Diode(λ=785nm) |
| LASER safety | IEC60825-1 Class 1 |
| Current consumption | 400mA or less (Rush current 1A) |
| Detection distance | 120mm ~ 3,500mm |
| Interface | 3.3V USART (230,400 bps) 42bytes per 6 degrees, Full Duplex option |
| Ambient Light Resistance | 10,000 lux or less |
| Sampling Rate | 1.8kHz |
| Dimensions | 69.5(W) X 95.5(D) X 39.5(H)mm |
| Mass | Under 125g |

### [Measurement Performance Specifications](http://emanual.robotis.com/docs/en/platform/turtlebot3/appendix_lds_01/" \l "measurement-performance-specifications)

| **Items** | **Specifications** |
| --- | --- |
| Distance Range | 120 ~ 3,500mm |
| Distance Accuracy (120mm ~ 499mm) | ±15mm |
| Distance Accuracy(500mm ~ 3,500mm) | ±5.0% |
| Distance Precision(120mm ~ 499mm) | ±10mm |
| Distance Precision(500mm ~ 3,500mm) | ±3.5% |
| Scan Rate | 300±10 rpm |
| Angular Range | 360° |
| Angular Resolution | 1° |