

ROS 机器人平台智行 Mini

用户手册

版本信息

| 版本 | 日期 | 修订者 | 说明 |
|------|------------|------------|------|
| V1.0 | 2020-07-14 | 宋刚, 李强, 郭磊 | 初次编写 |

目录

| | |
|------------------------------------|----|
| 第一章. 智行 Mini 开箱指南 | 9 |
| 1.1 实验目的: | 9 |
| 1.2 实验要求: | 9 |
| 1.3 实验工具: | 9 |
| 1.4 实验内容: | 9 |
| 1.4.1 开箱说明 | 9 |
| 1.4.2 Nomachine 远程连接 | 9 |
| 1.4.3 智行 Mini 软件框架与功能预览 | 12 |
| 1.5 实验结果: | 14 |
| 1.6 实验报告: | 14 |
| 第二章. ROS 基本操作 | 15 |
| 2.1 实验目的: | 15 |
| 2.2 实验要求: | 15 |
| 2.3 实验工具: | 15 |
| 2.4 实验内容: | 15 |
| 2.4.1 ROS 工作空间、功能包以及节点的认识与创建 | 15 |
| 2.4.2 ROS 编译系统 | 19 |
| 2.4.3 ROS 通信机制之话题通信 | 20 |
| 2.4.4 ROS 通信机制之服务通信 | 24 |
| 2.4.5 参数服务器 | 26 |
| 2.4.6 launch 文件介绍 | 26 |
| 2.5 实验结果: | 28 |
| 2.6 实验报告: | 28 |
| 2.7 思考题: | 28 |
| 第三章. 智行 Mini 控制功能的实现 | 30 |
| 3.1 实验目的: | 30 |
| 3.2 实验要求: | 30 |
| 3.3 实验工具: | 30 |
| 3.4 实验内容: | 30 |
| 3.4.1 使用键盘控制智行 Mini 运动 | 30 |
| 3.4.2 使用手柄控制智行 mini | 32 |

| | |
|--------------------------------|----|
| 3.4.3 手柄控制节点分析 | 34 |
| 3.5 实验结果： | 35 |
| 3.6 实验报告： | 35 |
| 第四章.激光雷达驱动与滤波 | 36 |
| 4.1 实验目的： | 36 |
| 4.2 实验要求： | 36 |
| 4.3 实验工具： | 36 |
| 4.4 实验内容： | 36 |
| 4.4.1 激光雷达测距原理 | 36 |
| 4.4.2 LaserScan 消息解析 | 37 |
| 4.4.3 laser_filters 包的使用 | 38 |
| 4.5 实验结果： | 40 |
| 4.6 实验报告： | 40 |
| 第五章.激光雷达避障 | 41 |
| 5.1 实验目的： | 41 |
| 5.2 实验要求： | 41 |
| 5.3 实验工具： | 41 |
| 5.4 实验内容： | 41 |
| 5.4.1 功能要求与分析 | 41 |
| 5.4.2 功能的实现 | 41 |
| 5.5 实验结果： | 45 |
| 5.6 实验报告： | 45 |
| 第六章.里程计与坐标变换 | 46 |
| 6.1 实验目的： | 46 |
| 6.2 实验要求： | 46 |
| 6.3 实验工具： | 46 |
| 6.4 实验内容： | 46 |
| 6.4.1 ROS 机器人中一些坐标系的介绍 | 46 |
| 6.4.2 双轮差分运动模型的运动学解算 | 47 |
| 6.4.3 tf 变换的简单介绍 | 51 |
| 6.5 实验结果： | 56 |
| 6.6 实验报告： | 56 |

| | |
|----------------------------------|----|
| 6.7 思考题: | 56 |
| 第七章.轮式里程计与 imu 数据融合 | 58 |
| 7.1 实验目的: | 58 |
| 7.2 实验要求: | 58 |
| 7.3 实验工具: | 58 |
| 7.4 实验内容: | 58 |
| 7.4.1IMU 传感器使用的必要 | 58 |
| 7.4.2IMU 数据结构 | 58 |
| 7.4.3robot_pose_ekf 实现 IMU 校准里程计 | 62 |
| 7.5 实验结果: | 64 |
| 7.6 实验报告: | 64 |
| 第八章.基于里程计的运动控制 | 65 |
| 8.1 实验目的: | 65 |
| 8.2 实验要求: | 65 |
| 8.3 实验工具: | 65 |
| 8.4 实验内容: | 65 |
| 8.4.1 进行基于里程计的运动控制 | 65 |
| 8.5 实验结果: | 68 |
| 8.6 实验报告: | 68 |
| 第九章.Gmapping 建图 | 69 |
| 9.1 实验目的: | 69 |
| 9.2 实验要求: | 69 |
| 9.3 实验工具: | 69 |
| 9.4 实验内容: | 69 |
| 9.4.1 棚格地图的概念 | 69 |
| 9.4.2 建图的原理 | 70 |
| 9.4.3gmapping 建图的一些参数 | 71 |
| 9.4.4Gmapping 建图流程 | 73 |
| 9.5 实验结果: | 74 |
| 9.6 实验报告: | 75 |
| 第十章.Navigation 自主导航（上） | 76 |
| 10.1 实验目的: | 76 |

| | |
|---------------------------------------|------------|
| 10.2 实验要求: | 76 |
| 10.3 实验工具: | 76 |
| 10.4 实验内容: | 76 |
| 10.4.1 使用 navigation 包实现智行 mini 的导航功能 | 76 |
| 10.4.2 Navigation 导航原理 | 78 |
| 10.4.3 amcl 自主定位 | 79 |
| 10.4.4 move_base 功能使用 | 82 |
| 10.5 实验结果: | 94 |
| 10.6 实验报告: | 94 |
| 第十一章.Navigation 自主导航（下） | 95 |
| 11.1 实验目的: | 95 |
| 11.2 实验要求: | 95 |
| 11.3 实验工具: | 95 |
| 11.4 实验内容: | 95 |
| 11.4.1 运行流程 | 95 |
| 11.4.2 代码解读 | 96 |
| 11.5 实验结果: | 98 |
| 11.6 实验报告: | 98 |
| 第十二章.单目相机驱动 | 99 |
| 12.1 实验目的: | 99 |
| 12.2 实验要求: | 99 |
| 12.3 实验工具: | 99 |
| 12.4 实验内容 | 99 |
| 12.4.1 摄像头驱动实验 | 99 |
| 12.4.2 使用 opencv 视觉库处理图像信息 | 100 |
| 12.5 实验结果: | 106 |
| 12.6 实验报告: | 106 |
| 第十三章.单目相机参数标定实验 | 107 |
| 13.1 实验目的: | 107 |
| 13.2 实验要求: | 107 |
| 13.3 实验工具: | 107 |
| 13.4 实验内容 | 107 |

| | |
|------------------------------|-----|
| 13.4.1 摄像头标定实验 | 107 |
| 13.5 实验结果: | 111 |
| 13.6 实验报告: | 111 |
| 第十四章.颜色形状识别与跟踪实验 | 112 |
| 14.1 实验目的: | 112 |
| 14.2 实验要求: | 112 |
| 14.3 实验工具: | 112 |
| 14.4 实验内容 | 112 |
| 14.5 实验结果: | 115 |
| 14.6 实验报告: | 115 |
| 第十五章.基于 opencv 的人脸识别实验 | 116 |
| 15.1 实验目的: | 116 |
| 15.2 实验要求: | 116 |
| 15.3 实验工具: | 116 |
| 15.4 实验内容 | 116 |
| 15.4.1 人脸检测功能实现 | 116 |
| 15.4.2 人脸数据采集 | 118 |
| 15.4.3 人脸识别功能实现 | 120 |
| 15.5 实验结果: | 123 |
| 15.6 实验报告: | 123 |
| 第十六章.基于 kcf 的目标跟踪实验 | 124 |
| 16.1 实验目的: | 124 |
| 16.2 实验要求: | 124 |
| 16.3 实验工具: | 124 |
| 16.4 实验内容 | 124 |
| 16.4.1 目标跟踪功能实现 | 124 |
| 16.5 实验结果: | 128 |
| 16.6 实验报告: | 128 |
| 第十七章.卷积神经网络目标识别实验 | 129 |
| 17.1 实验目的: | 129 |
| 17.2 实验要求: | 129 |
| 17.3 实验工具: | 129 |

| | |
|-----------------------------|-----|
| 17.4 实验内容 | 129 |
| 17.4.1 训练自己的卷积神经网络框架 | 129 |
| 17.4.2 识别自定义目标 | 138 |
| 17.5 实验结果: | 141 |
| 17.6 实验报告: | 141 |
| 第十八章.语音阵列驱动实验 | 142 |
| 18.1 实验目的: | 142 |
| 18.2 实验要求: | 142 |
| 18.3 实验工具: | 142 |
| 18.4 实验内容 | 142 |
| 18.4.1 语音阵列介绍 | 142 |
| 18.4.2 语音数据采集 | 143 |
| 18.4.3 语音阵列数据采集 | 145 |
| 18.5 实验结果: | 151 |
| 18.6 实验报告: | 151 |
| 第十九章.基于百度 SDK 语音识别实验 | 152 |
| 19.1 实验目的: | 152 |
| 19.2 实验要求: | 152 |
| 19.3 实验工具: | 152 |
| 19.4 实验内容 | 152 |
| 19.4.1 百度语音 SDK | 152 |
| 19.4.2 基于百度 SDK 的语音识别 | 155 |
| 19.5 实验结果: | 158 |
| 19.6 实验报告: | 158 |
| 第二十章.语音控制实验 | 159 |
| 20.1 实验目的: | 159 |
| 20.2 实验要求: | 159 |
| 20.3 实验工具: | 159 |
| 20.4 实验内容 | 159 |
| 20.5 实验结果: | 161 |
| 20.6 实验报告: | 161 |
| 第二十一章.语音播报实验 | 162 |

| | |
|---------------------------|-----|
| 21.1 实验目的: | 162 |
| 21.2 实验要求: | 162 |
| 21.3 实验工具: | 162 |
| 21.4 实验内容 | 162 |
| 21.5 实验结果: | 163 |
| 21.6 实验报告: | 163 |
| 第二十二章. 统一部件组仿人视觉对抗 B 比赛例程 | 164 |
| 22.1 实验目的: | 164 |
| 22.2 实验要求: | 164 |
| 22.3 实验工具: | 164 |
| 22.4 实验内容 | 164 |
| 22.5 实验结果: | 175 |
| 22.6 实验报告: | 175 |
| 附录一.Ubuntu 系统下串口绑定 | 175 |

第一章.智行 Mini 开箱指南

1.1 实验目的:

了解智行 mini 的使用方法，以便进行后面的学习。

1.2 实验要求:

1. 按照实验步骤实现 Nomachine 的连接
2. 实现桌面功能的实验

1.3 实验工具:

个人电脑一台，智行 mini 及其配件

1.4 实验内容:

1.4.1 开箱说明

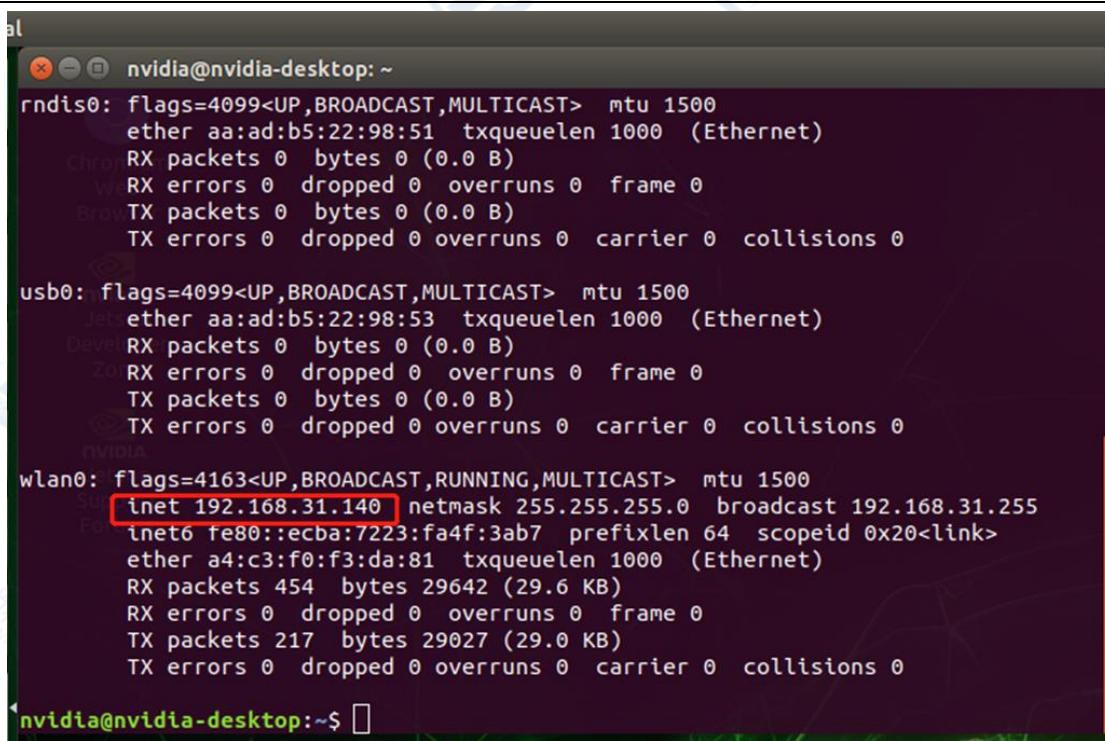
1.4.2 Nomachine 远程连接

智行 mini 上连接 Nomachine 准备工作

NoMachine 是一个可以将人工智能开放研究平台的桌面通过局域网分享出来的一个工具，首先需要插上智行 mini 的外接显示器、鼠标键盘，接好之后开机，显示器会显示操作系统界面：



手动点击右上角 wifi 信号连接到无线网络，以后智行 mini 会自动连接到该网络。点击左侧终端按钮或按下 Ctrl+Alt+T 打开终端，输入 ifconfig 得到 IP 地址：



```
nvidia@nvidia-desktop: ~
 0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
    ether aa:ad:b5:22:98:51  txqueuelen 1000  (Ethernet)
      RX packets 0  bytes 0 (0.0 B)
      RX errors 0  dropped 0  overruns 0  frame 0
      TX packets 0  bytes 0 (0.0 B)
      TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0

  1: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
    ether aa:ad:b5:22:98:53  txqueuelen 1000  (Ethernet)
      RX packets 0  bytes 0 (0.0 B)
      RX errors 0  dropped 0  overruns 0  frame 0
      TX packets 0  bytes 0 (0.0 B)
      TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0

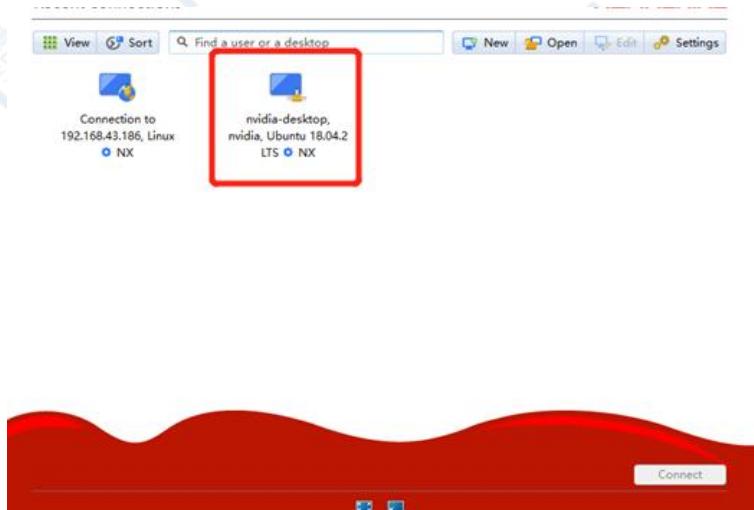
  wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet 192.168.31.140  netmask 255.255.255.0  broadcast 192.168.31.255
      inet6 fe80::ecba:7223:fa4f:3ab7  prefixlen 64  scopeid 0x20<link>
        ether a4:c3:f0:f3:da:81  txqueuelen 1000  (Ethernet)
          RX packets 454  bytes 29642 (29.6 KB)
          RX errors 0  dropped 0  overruns 0  frame 0
          TX packets 217  bytes 29027 (29.0 KB)
          TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0

nvidia@nvidia-desktop:~$
```

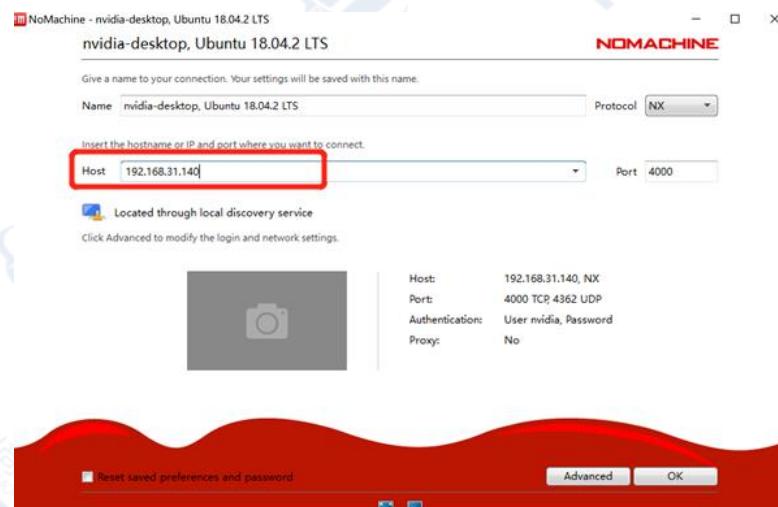
这里是 192.168.31.140，记住该地址，然后拔掉 hdmi，插入随机赠送的 hdmi 虚拟模块。

个人电脑上安装 Nomachine 并连接

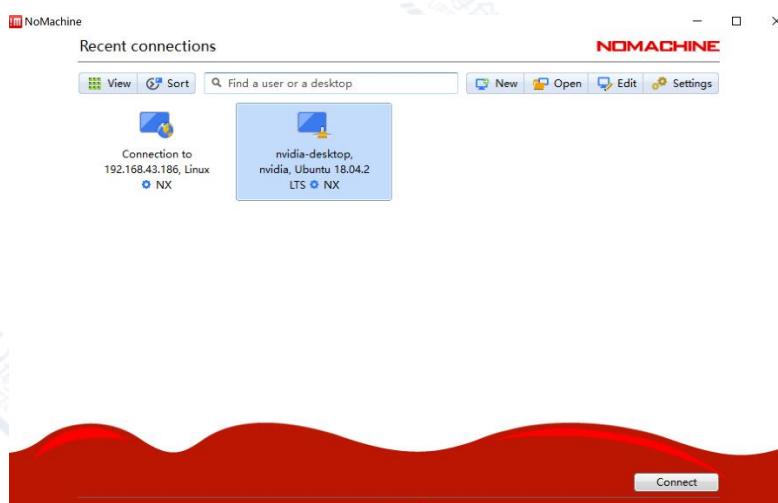
在个人电脑上安装 Nomachine，下载地址：<https://www.nomachine.com/> 安装完成后打开（保证智行 mini 和个人电脑在一个 wifi 网络下），点击这里，选择 edit connection



得到以下界面，在这里输入 snowman 的 IP 地址，点击 advanced，然后一路 OK：



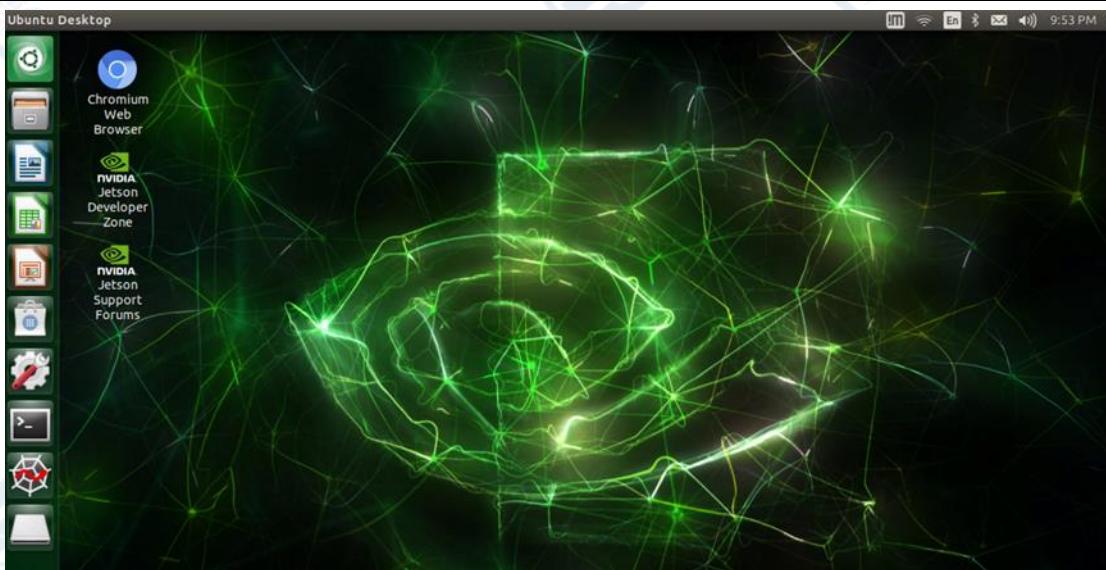
接着回到这里之后，左键双击这个：



输入账号：bcsh 密码：123456，点击 OK，后面根据提示全选 OK：

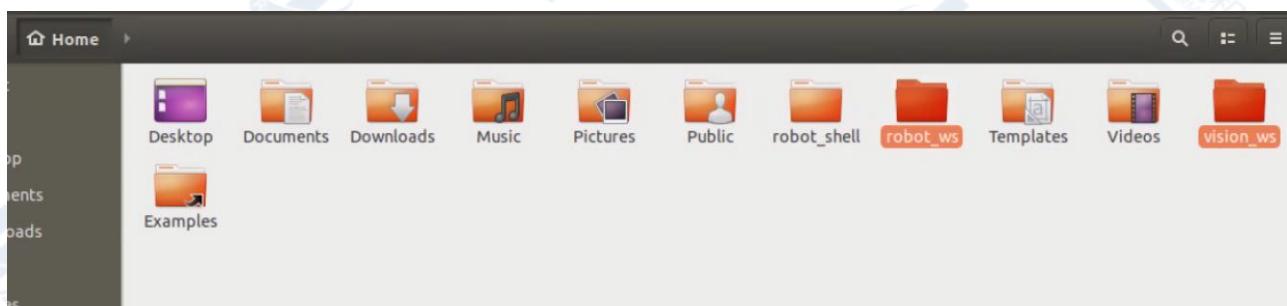


在个人电脑上看到此桌面即完成连接：



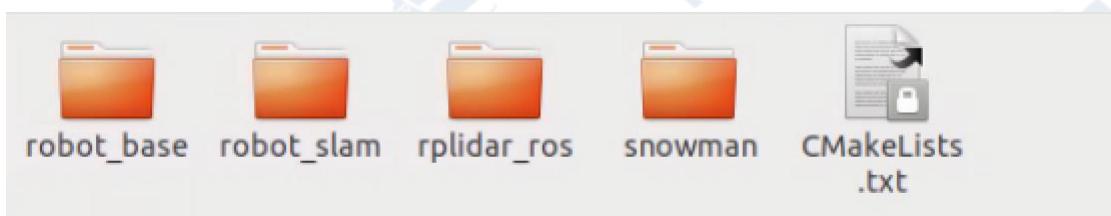
1.4.3 智行 Mini 软件框架与功能预览

智行 Mini 的软件架构是基于 ros 机器人操作系统的，软件架构由各种 ros 功能包组成，用于实现各种功能，源码位置在这里：



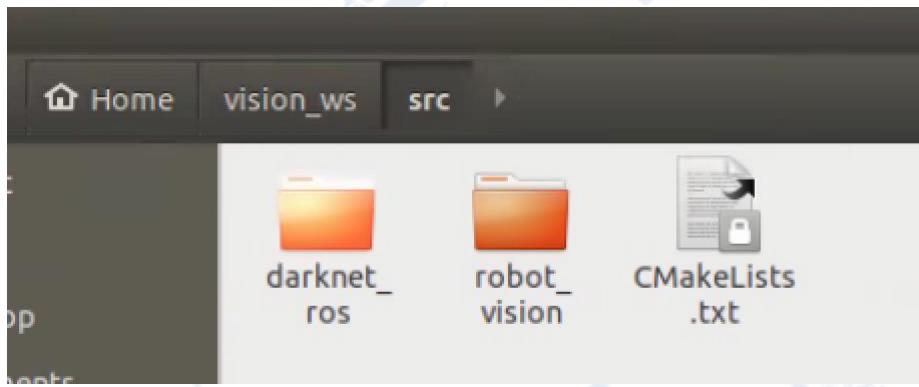
总共有两个工作空间 robot_ws 和 vision_ws。

先看看 robot_ws：



- Robot_base: 智行 Mini 底盘驱动，运动控制，模型等代码
- Robot_slam: 各种激光 slam 定位建图导航等
- Rplidar_ros: 激光雷达驱动
- Snowman: 语音交互代码

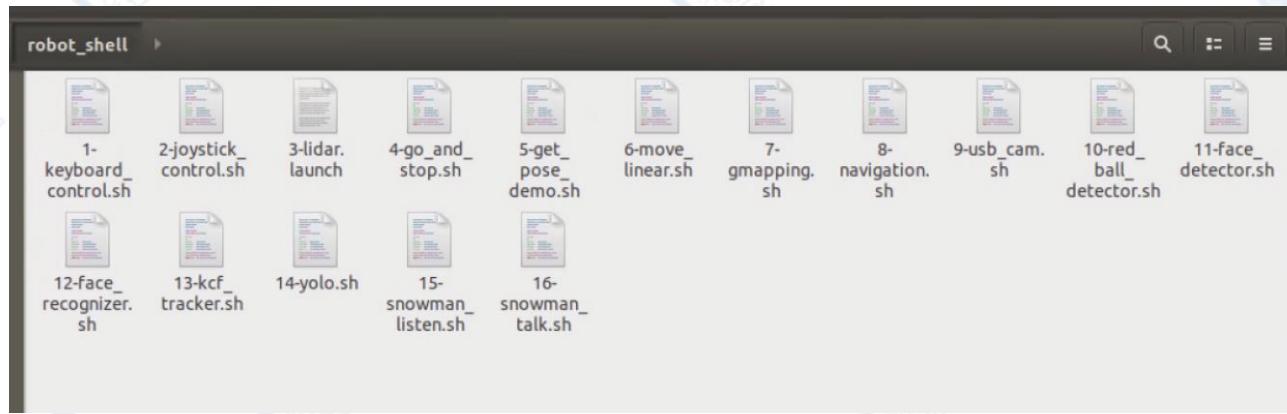
再看看 vision_ws:



- Robot_vision: 视觉功能源码
- Darknet_ros: 深度学习，AI 功能源码

同时，为了快速上手，我们也准备了一系列脚本，同学们可先尝试运行脚本感受 ros 机器人的强大功能。

脚本位置如下：



| 脚本列表 | 功能 | 对应课程 |
|------|----------------|--------|
| 1 | 键盘控制智行 Mini 运动 | 第三章 |
| 2 | 摇杆控制智行 Mini 运动 | 第三章 |
| 3 | 激光雷达驱动 | 第四章 |
| 4 | 激光雷达避障 | 第五章 |
| 5 | 里程计信息获取 | 第六章 |
| 6 | 直线运动控制 | 第八章 |
| 7 | Gmapping 建图 | 第九章 |
| 8 | 自主导航 | 第十，十一章 |
| 9 | 相机驱动 | 第十二章 |
| 10 | 红球追踪 | 第十四章 |

| | | |
|----|---------|---------|
| 11 | 人脸检测 | 第十五章 |
| 12 | 人脸识别 | 第十六章 |
| 13 | 目标跟踪 | 第十七章 |
| 14 | 目标检测与识别 | 第十八章 |
| 15 | 语音识别与控制 | 第十九,二十章 |
| 16 | 语音播报实验 | 第二十一章 |

1.5 实验结果:

熟悉智行 Mini 的框架，源码位置，案例预览等等

1.6 实验报告:

实验目的

实验要求

实验内容

实验总结

第二章.ROS 基本操作

2.1 实验目的:

了解 ROS 的基本操作

2.2 实验要求:

3. 能够创建自己的工作空间以及功能包
4. 了解 ROS 中典型的通讯机制
5. 了解典型 launch 文件的用法

2.3 实验工具:

个人电脑一台，智行 mini 及其配件

2.4 实验内容:

2.4.1ROS 工作空间、功能包以及节点的认识与创建

工作空间的认识:

工作空间是 ROS 的一个总的工程文件夹，工作空间可以作为一个独立的项目进行编译，存放 ROS 程序的源文件、编译文件和执行文件。一般包括以下文件夹：



其中：

src: 放置所有的源代码，配置文件等

build: 放置编译过程中产生的中间文件

devel: 放置编译生成的可执行文件

install: 放置用 install 指令安装成功后的结果

工作空间的创建:

首先使用系统命令创建工作空间目录，然后运行 ROS 工作空间的初始化命令即可完成创建过程：

```
mkdir -p ~/ros_workspace/src
```

直接创建二级文件夹 src,其中 ros_workspace 是工作空间的名字，可自己选择

```
cd ~/ros_workspace/
```

打开新创建的 ROS 工作空间,此时文件夹 ros_workspace 下只有一个 src 文件夹

```
catkin_make
```

初始化，之后会多出 devel 和 build 文件

```
source ~/ros_workspace/devel/setup.bash
```

将对应的工作空间的路径加入到环境变量 ROS_PACKAGE_PATH 中：

```
echo $ROS_PACKAGE_PATH
```

用于查看工作空间条件是否到了 ROS_PACKAGE_PATH 中。

如果希望环境变量在所有终端中有效，则需要在终端配置文件中加入环境变量的设置：

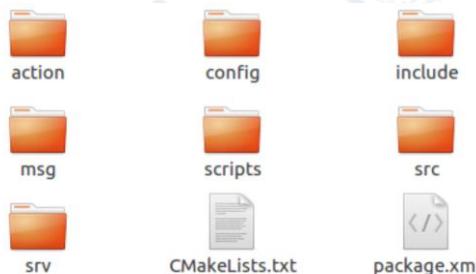
```
echo "source ~/WORKSPACE/devel/setup.bash">>>~/.bashrc
```

```
source ~/.bashrc
```

注意用自己命名的工作空间替换这里的 WORKSPACE

功能包的认识：

功能包是 ROS 软件中的基本单元，包含有 ROS 节点、库、配置文件等等，典型的功能包包含以下文件结构：



其中：

action: 放置功能包自定义的动作指令

config: 放置功能包中的配置文件，由用户创建，文件名可以不同。

include: 放置功能包中需要用到的头文件。

msg: 放置功能包自定义的消息类型。

scripts: 放置可以直接运行的 Python 脚本。

src: 放置需要编译的 C++ 代码。

srv: 放置功能包自定义的服务类型。

CMakeLists.txt: 编译器编译功能包的规则。

package.xml: 功能包清单。

功能包的创建：

ROS 提供直接创建功能包的命令 `catkin_create_pkg`, 使用方法如下:

首先切换到之前通过创建 catkin 工作空间教程创建的 catkin 工作空间中的 src 目录下:

```
~/ros_workspace$ cd src
```

接着使用 `catkin_create_pkg` 命令来创建一个名为 `learning_communication` 的新程序包, 这个程序包依赖于 `std_msgs`、`roscpp` 和 `rospy`: (分别是为了支持消息传递、C++ 编译、python 运行)

```
~/ros_workspace/src$ catkin_create_pkg learning_communication roscpp rospy std_msgs
```

含义是创建一个名为 `learning_communication` 的功能包, 该功能包依赖 `roscpp` `rospy` `std_msgs`, 创建成功后, 提示如下:

```
Created file learning_communication/CMakeLists.txt
```

```
Created folder learning_communication/include/my_demo
```

```
Created folder learning_communication/src
```

```
Successfully created files in
```

```
/home/nic/ros_workspace/src/learning_communication. Please adjust the values in
```

```
package.xml
```

这将会创建一个名为 `learning_communication` 的文件夹, 这个文件夹里面包含一个 `package.xml` 文件和一个 `CMakeLists.txt` 文件, 这两个文件都已经自动包含了部分你在执行 `catkin_create_pkg` 命令时提供的信息。

节点的认识

节点就是一些执行运算任务的进程, 它是一个能执行特定工作任务的工作单元, 并且能够相互通信, 从而实现一个机器人系统整体的功能。下面以一个典型的通信节点来举例。

```
#include <iostream>
#include "ros/ros.h"
#include "std_msgs/String.h"

int main(int argc, char **argv)
{
    // ROS节点初始化
    ros::init(argc, argv, "talker");

    // 创建节点句柄，方便调用
    ros::NodeHandle n;

    // 创建一个Publisher，发布名为chatter的topic，消息类型为std_msgs::String
    ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);

    // 设置循环的频率
    ros::Rate loop_rate(10);

    int count = 0;
    while (ros::ok())
    {
        // 初始化std_msgs::String类型的消息
        std_msgs::String msg;
        std::stringstream ss;
        ss << "hello world " << count;
        msg.data = ss.str();

        // 发布消息
        ROS_INFO("%s", msg.data.c_str());
        chatter_pub.publish(msg);

        // 循环等待回调函数
        ros::spinOnce();

        // 按照循环频率延时
        loop_rate.sleep();
        ++count;
    }
    return 0;
}
```

这是一个话题通讯中的话题发布者（关于通讯会在后面介绍，这里只看简单结构），名为 Talker.cpp 的一个 cpp 文件，是一个用 C++ 编写的节点，ROS 中不仅支持 C++ 也支持 Python 等多种语言，该节点经过初始化之后完成了发布一个名为 chatter 的话题，并发布消息类型为 String 的消息。

关于节点的编写可以在之后介绍通讯部分仿照案例自行编写。

2.4.2 ROS 编译系统

对于 ROS 工作空间中的 C++语言编写的功能包来说，可以通过 catkin_make 来进行编译，它可以一次性的编译整个工作空间中的所有功能包，使用方法是进入工作空间的路径输入 catkin_make：

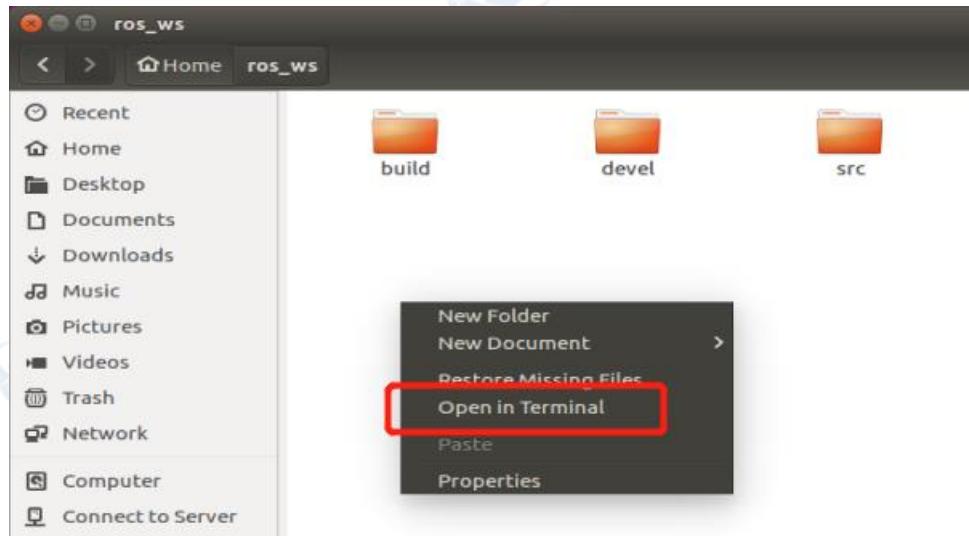
```
bcsh@ubuntu: ~
bcsh@ubuntu:~$ cd ros_ws/
```

输入 cd <工作空间名>即可进入该路径

```
bcsh@ubuntu: ~/ros_ws
bcsh@ubuntu:~$ cd ros_ws/
bcsh@ubuntu:~/ros_ws$ catkin_make
```

接着输入 catkin_make 即可

或者直接进入 ros_ws 文件夹右键，然后选择在终端打开后输入 catkin_make 亦可



catkin_make 的使用依赖于 Cmakelists.txt 以及 package.xml 文件

package.xml 文件主要指明该功能包在编译和运行时依赖于哪些其他 package，同时也包含该 package 的一些描述信息，如作者、版本等。典型内容如下：

```
<package>
  <name>hello_world_tutorial</name>
  <maintainer email="you@example.com">Your Name</maintainer>
  <description>
    A ROS tutorial.
  </description>
  <version>0.0.0</version>
  <license>BSD</license>

  <!-- Required by Catkin -->
  <buildtool_depend>catkin</buildtool_depend> 1

  <!-- Package Dependencies -->
  <build_depend>roscpp</build_depend>
  <run_depend>roscpp</run_depend> 2
</package>
```

其中 1、2 两处就分别表示改功能包编译和运行时都依赖 roscpp

Cmakelists.txt 文件则应具备以下部分：

```
cmake_minimum_required(VERSION 2.8.3)
project(learning_communication)
```

声明 CMake API 版本以及项目名称

```
cmake_minimum_required(VERSION 2.8.3)
project(catkin)

find_package(catkin REQUIRED COMPONENTS
    geometry_msgs
    roscpp
    rospy
    std_msgs
    message_generation
)
```

搜索依赖项 (即 roscpp 等) 的信息

```
include_directories(
    include
    ${catkin_INCLUDE_DIRS}
)
```

搜索 catkin 中调用的头文件

```
add_executable(talker src/talker.cpp)
```

设置待生成可执行文件的名字

```
target_link_libraries(talker ${catkin_LIBRARIES})
```

设置编译过程的 linking library

```
add_dependencies(talker ${PROJECT_NAME}_generate_messages_cpp)
```

添加依赖项

2.4.3ROS 通信机制之话题通信

基本概念

节点 (Node)

节点就是一些执行运算任务的进程。

消息 (Message)

节点之间通讯机制的实现就是通过发布和订阅消息，每一个消息都是严格的数据结构，支持标准数据类型、嵌套结构和数组，也可以根据需要自行定义。

话题 (Topic)

消息以一种发布/订阅 (Publish/Subscribe) 的方式传递。一个节点可以根据一个给定的话题发布消息 (发布者)，也可以关注某个话题并订阅特定类型的数据 (订阅者)，系统中可以同时有多个节点发布或者订阅同一个话题的消息。

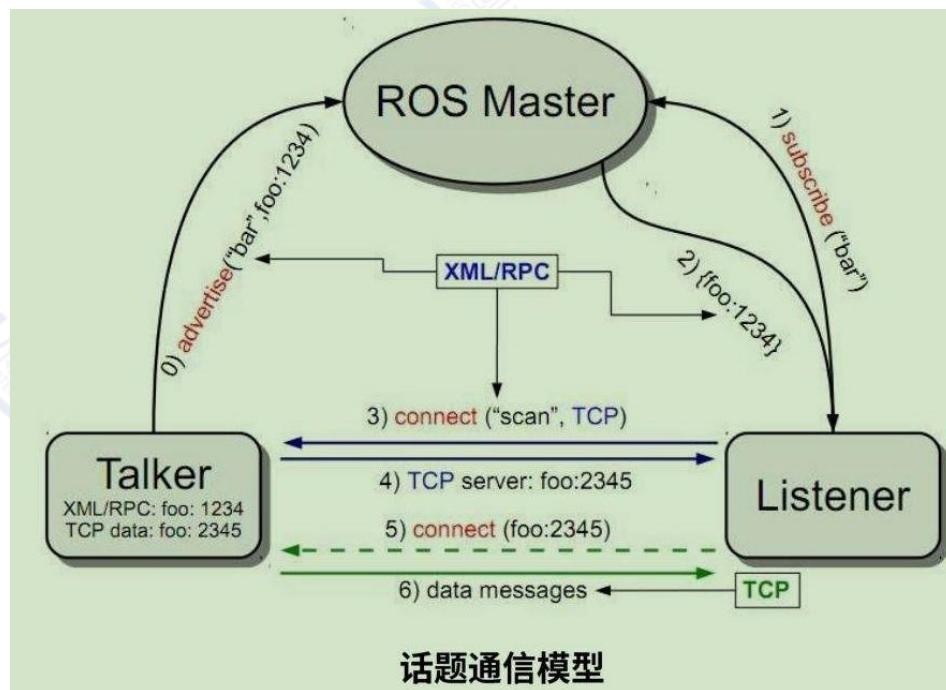
节点管理器 (ROS Master)

ROS Master 的功能是为了能够使所有的节点有条不紊的执行，它通过远程过程调用 (RPC) 提供登记列表和对其他图表的查找功能，帮助 ROS 节点之间相互查找、建立连接，同时还为系统提供参数服务器，管理全局参数。

机理

话题通讯机制在 ROS 中使用更为频繁，如图所示有两个节点，分别是发布者 Talker 和订阅者 Listener。

建立通信的详细过程如下



1. Talker 与 Listener 的注册:

两个节点分别发布、订阅同一个话题，启动顺序没有要求

2. ROS Master 进行信息匹配:

MASTER 根据发布者和订阅者的信息进行比对，一旦 ROS Master 找到了相互匹配的发布者和订阅者，就会进行匹配：通过 RPC 向 Listener 发送 Talker 的 RPC 地址信息。

3. Listener 发送连接请求

Listener 接收到 Master 发回的 Talker 地址信息，尝试通过 RPC 向 Talker 发送连接请求，传输订阅的话题名、消息类型以及通讯协议。

4. Talker 确定连接请求

Talker 接收到 Listener 发送的连接请求后，继续通过 RPC 向 Listener 确认连接信息，其中包含自身 TCP 地址信息。

5. Listener 尝试与 Talker 建立网络连接

Listener 接收到确认信息后，使用 TCP 尝试与 Talker 建立网络连接。

6. Talker 向 Listener 发布数据

成功建立连接后，Talker 开始向 Listener 发送话题消息数据。

通信方式的比喻

可以通过以下比喻来类比主题-消息通讯方式的建立。

1. 公司 A 发布招聘信息，求职者 B 想找工作
2. 求职平台 M 匹配成功，告诉求职者 B 公司 A 现在在招人，给其联系方式（RPC 地址）
3. 求职者 B 通过从平台 M 那里得到的公司 A 的联系方式，向公司递交请求
4. 公司收到这份求职信息，向求职者表示我们已经收到你的信息，并给了一个负责人的微信（TCP 地址）
5. 求职者收到确认信息后，添加了负责人的微信。
6. 公司负责人开始向求职者发送面试题目。

在该比喻中，Topic：找工作；Master：平台 M；Talker：公司 A；Listener：求职者 B。

实验

打开终端输入：

roscore

打开 ROS 节点管理器。

重新打开一个终端输入

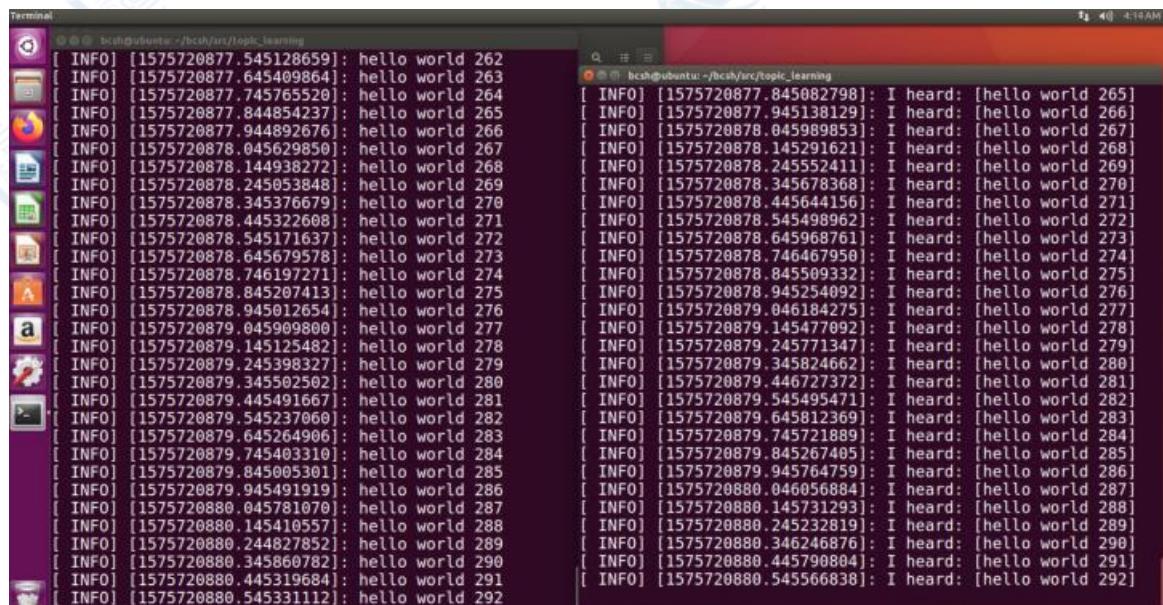
rosrun learn_communication talker

启动 talker 节点

以及启动 listener 节点：

rosrun learn_communication listener

可以看到如下结果：



```
[ INFO] [1575720877.545128659]: hello world 262
[ INFO] [1575720877.645409864]: hello world 263
[ INFO] [1575720877.745765529]: hello world 264
[ INFO] [1575720877.844854237]: hello world 265
[ INFO] [1575720877.944892676]: hello world 266
[ INFO] [1575720878.045629850]: hello world 267
[ INFO] [1575720878.144938272]: hello world 268
[ INFO] [1575720878.245053848]: hello world 269
[ INFO] [1575720878.34532608]: hello world 270
[ INFO] [1575720878.445322608]: hello world 271
[ INFO] [1575720878.545171637]: hello world 272
[ INFO] [1575720878.645679578]: hello world 273
[ INFO] [1575720878.746197271]: hello world 274
[ INFO] [1575720878.845207413]: hello world 275
[ INFO] [1575720878.945012654]: hello world 276
[ INFO] [1575720879.045909800]: hello world 277
[ INFO] [1575720879.145125482]: hello world 278
[ INFO] [1575720879.245398327]: hello world 279
[ INFO] [1575720879.345502502]: hello world 280
[ INFO] [1575720879.445491667]: hello world 281
[ INFO] [1575720879.545237060]: hello world 282
[ INFO] [1575720879.645264906]: hello world 283
[ INFO] [1575720879.745403310]: hello world 284
[ INFO] [1575720879.845005391]: hello world 285
[ INFO] [1575720879.945491919]: hello world 286
[ INFO] [1575720880.045781070]: hello world 287
[ INFO] [1575720880.145410557]: hello world 288
[ INFO] [1575720880.244827852]: hello world 289
[ INFO] [1575720880.345860782]: hello world 290
[ INFO] [1575720880.445319684]: hello world 291
[ INFO] [1575720880.545331112]: hello world 292
[ INFO] [1575720878.145291621]: I heard: [hello world 265]
[ INFO] [1575720878.245552411]: I heard: [hello world 266]
[ INFO] [1575720878.045989853]: I heard: [hello world 267]
[ INFO] [1575720878.145291621]: I heard: [hello world 268]
[ INFO] [1575720878.245552411]: I heard: [hello world 269]
[ INFO] [1575720878.345678368]: I heard: [hello world 270]
[ INFO] [1575720878.445644156]: I heard: [hello world 271]
[ INFO] [1575720878.545498962]: I heard: [hello world 272]
[ INFO] [1575720878.645968761]: I heard: [hello world 273]
[ INFO] [1575720878.746467950]: I heard: [hello world 274]
[ INFO] [1575720878.845509332]: I heard: [hello world 275]
[ INFO] [1575720878.945254092]: I heard: [hello world 276]
[ INFO] [1575720879.046184275]: I heard: [hello world 277]
[ INFO] [1575720879.145477092]: I heard: [hello world 278]
[ INFO] [1575720879.245771347]: I heard: [hello world 279]
[ INFO] [1575720879.345824662]: I heard: [hello world 280]
[ INFO] [1575720879.446727372]: I heard: [hello world 281]
[ INFO] [1575720879.545495471]: I heard: [hello world 282]
[ INFO] [1575720879.645812369]: I heard: [hello world 283]
[ INFO] [1575720879.745721889]: I heard: [hello world 284]
[ INFO] [1575720879.845267405]: I heard: [hello world 285]
[ INFO] [1575720879.945764759]: I heard: [hello world 286]
[ INFO] [1575720880.046056884]: I heard: [hello world 287]
[ INFO] [1575720880.145731293]: I heard: [hello world 288]
[ INFO] [1575720880.245232819]: I heard: [hello world 289]
[ INFO] [1575720880.346246876]: I heard: [hello world 290]
[ INFO] [1575720880.445790804]: I heard: [hello world 291]
[ INFO] [1575720880.545566838]: I heard: [hello world 292]
```

我们来学习一下 talker 和 listener 的源码

Talker:

```
/**  
 * 该例程将发布 chatter 话题，消息类型 String  
 */  
  
#include <iostream>  
#include "ros/ros.h"  
#include "std_msgs/String.h"  
int main(int argc, char **argv)  
{  
    // ROS 节点初始化  
    ros::init(argc, argv, "talker");  
    // 创建节点句柄  
    ros::NodeHandle n;  
    // 创建一个 Publisher，发布名为 chatter 的 topic，消息类型为 std_msgs::String  
    ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);  
    // 设置循环的频率  
    ros::Rate loop_rate(10);  
    int count = 0;  
    while (ros::ok())  
    {  
        // 初始化 std_msgs::String 类型的消息  
        std_msgs::String msg;  
        std::stringstream ss;  
        ss << "hello world " << count;  
        msg.data = ss.str();  
        // 发布消息  
        ROS_INFO("%s", msg.data.c_str());  
        chatter_pub.publish(msg);  
        // 循环等待回调函数  
        ros::spinOnce();  
        // 按照循环频率延时  
        loop_rate.sleep();  
        ++count;  
    }  
    return 0;  
}
```

可以看到在这个源码文件中，我们初始化了一个 ros 节点，创建了一个 ros 的消息发布器 chatter_pub，同时不断生成内容为 String 的消息，将消息填入消息发布器中，定时发送。

Listener:

```
/**  
 * 该例程将订阅 chatter 话题，消息类型 String  
 */  
  
#include "ros/ros.h"  
#include "std_msgs/String.h"
```

```
// 接收到订阅的消息后，会进入消息回调函数
void chatterCallback(const std_msgs::String::ConstPtr& msg)
{
    // 将接收到的消息打印出来
    ROS_INFO("I heard: [%s]", msg->data.c_str());
}

int main(int argc, char **argv)
{
    // 初始化 ROS 节点
    ros::init(argc, argv, "listener");

    // 创建节点句柄
    ros::NodeHandle n;

    // 创建一个 Subscriber，订阅名为 chatter 的 topic，注册回调函数 chatterCallback
    ros::Subscriber sub = n.subscribe("chatter", 1000, chatterCallback);

    // 循环等待回调函数
    ros::spin();

    return 0;
}
```

在这个文件中，我们初始化了一个节点，同时创建了一个消息接收器 sub，接收 talker 发送来的消息，并打印出来。

2.4.4 ROS 通信机制之服务通信

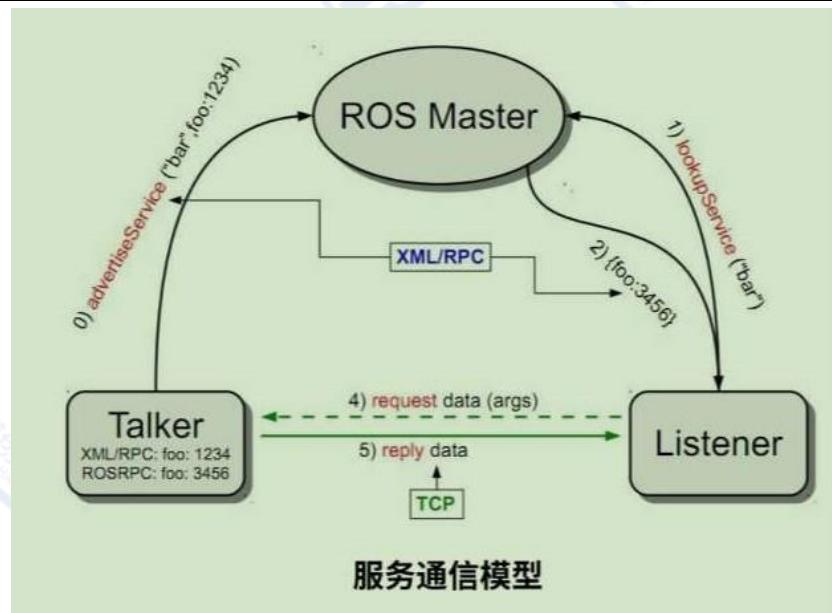
基本概念

服务 (Service)

对于双向的同步传输模式，话题通讯机制就不是很适用，对此我们采用名为服务的同步传输模式，与话题机制不同的是，ROS 中只允许有一个节点提供指定命名的服务。

机理

服务是带有一种应答的通信机制，通信原理如图所示，与话题通信相比，其减少了 Listener 与 Talker 之间的 RPC 通信。



1. Talker 与 Listener 的注册:

Talk 而通过 RPC 向 ROS Master 注册发布者的信息，其中包含有所提供的服务名；Listener 通过 RPC 向 ROS Master 注册订阅者的信息，包括需要查找的服务名。

2. ROS Master 进行信息匹配:

MASTER 根据 Listener 的订阅信息从注册列表中进行查找，如果没有找到匹配的 Talker，则等待 Talker 加入；如果找到匹配的 Talker 的信息，则通过 PRC 向 Listener 发送 Talker 的信息。

3. Listener 与 Talker 建立网络连接

Listener 接收到确认信息后，使用 TCP 尝试与 Talker 建立网络连接，并且发送服务的请求数据。

4. Talker 向 Listener 发布服务应答数据

Talker 接收到服务请求数据后，开始执行服务功能，执行完成后，向 Listener 发送数据。

通信方式的比喻

可以借助以下比喻来更好的理解服务消息机制

1. 学校门口有人脸识别功能的机器（提供进门的服务），和一个同学 A 要进学校（查找进门）
2. 学校门口保安得知学生想要进学校，告知这个同学可以通过人脸识别机器获取进门资格。
3. 学生 A 得知后去找机器去刷脸（向机器发送自己的脸的图像信号）
4. 机器收到信号，进行识别，识别后返回结果给同学 A（是本校生，准许进入）

在该比喻中：Service: 进门；Master: 校门口保安；Talker: 能实现人脸识别功能的机器；Listener:

想进门的学生 A

2.4.5 参数服务器

roscore 在启动时，除了启动 Mater Node, 用来管理 Node， 沟通各节点之间的消息和服务外。 还会创建一个 Parameter Server。它用来设置与查询参数。Parameter Server 可以存储: strings, integers, , booleans, lists, 字典， iso8601 数据， base64-encoded 数据。

参数服务器维护一个存储着各种参数的字典，字典就是为了方便读写一些不常改变的参数，给它们加上索引，这个索引是唯一的。

关于其一般的用法如下图所示：

rosparam

列出当前所有参数
`$ rosparam list`

显示某个参数的值
`$ rosparam get param_key`

设置某个参数的值
`$ rosparam set param_key param_value`

保存参数到文件
`$ rosparam dump file_name`

从文件读取参数
`$ rosparam load file_name`

删除参数参数
`$ rosparam delete param_key`

其中 rosparam load 后面的文件必须遵从 yaml 格式

2.4.6 launch 文件介绍

在前面我们介绍了使用如 `rosrun learning_communication server` 之类的命令来启动 ros 节点，但每当启动一个 ros 节点或工具的时候，都需要打开一个新的终端运行一个命令，当系统中的节点数量不断增加时，就会变得越来越麻烦，launch 文件就是用来解决这个问题的，它不仅可以以此同时启动多个节点，还可以同时启动 ROS Master 节点管理器，并且可以实现每个节点的各种配置，为多个节点的操作提供很大的便利。

这里提供一个简单的 launch 文件，结合文件来介绍 launch 文件内各个部分的含义：

```
<launch>
    <!-- Turtlesim Node-->
    <node pkg="turtlesim" type="turtlesim_node" name="sim"/>
    <node pkg="turtlesim" type="turtle_teleop_key" name="teleop"
output="screen"/>
    <!-- Axes -->
    <param name="scale_linear" value="2" type="double"/>
    <param name="scale_angular" value="2" type="double"/>

    <node pkg="learning_tf" type="turtle_tf_broadcaster"
          args="/turtle1" name="turtle1_tf_broadcaster" />
    <node pkg="learning_tf" type="turtle_tf_broadcaster"
          args="/turtle2" name="turtle2_tf_broadcaster" />
</launch>
```

首先，最基本的 launch 文件必须含有根元素 launch，然后在中间添加其他内容

```
<launch>
    ...
    ...
</launch>
```

launch 文件的核心在启动 ros 节点，如：

```
<node pkg="turtlesim" type="turtlesim_node" name="sim"/>
```

表示启动一节点，其中含有三个要素，pkg 定义节点所在功能包的名称（示例中为 turtlesim），type 定义节点的可执行文件的名称（示例中为 turtlesim_node），这两个要素也可以用 rosrun turtlesim turtlesim_node 来实现。最后的 name 属性用来定义节点运行的名称，将覆盖节点中 init() 赋予节点的名称。如示例中：

```
<node pkg="learning_tf" type="turtle_tf_broadcaster"
      args="/turtle1" name="turtle1_tf_broadcaster" />
```

就是表示将功能包 learning_tf 里的节点 turtle_tf_broadcaster 重命名为 turtle1_tf_broadcaster 这是最常用的三个属性。

在某些情况下，我们还会用到其他一些属性：

像这句话中的 args="/turtle1" 表示将发布的 topic 名字变为/turtle1，此外：

output = "screen"：将节点的标准输出打印到终端屏幕，默认输出为日志文档。

respawn = "true"：复位属性，该节点停止时，会自动启动，默认为 false。

required = "true"：必要节点，当该节点终止时，launch 文件内的其他节点也被终止。

ns = "namespace"：命名空间，为节点内的相对名称添加命名空间前缀。

args = "arguments"：节点需要的输入参数。

实际应用中的 launch 文件往往会更复杂，使用的标签也更多。像示例中的：

```
<param name="scale_linear" value="2" type="double"/>
```

表示将 scale_linear 这个参数的值，设置为 2，并且加载到参数服务器中。

更多的设置需要自己在实际应用中慢慢的掌握

2.5 实验结果：

对 ROS 基本操作有一定的了解、完成了自己的工作空间以及功能包、对不同的通信方式有所掌握、对 launch 文件有一定的理解。

2.6 实验报告：

实验目的

实验要求

实验内容

实验总结

思考题

2.7 思考题：

1. 为什么要打开多个终端来输入指令？

答：当一个终端在运行时是无法再执行新的指令的。

2. 新安装 ROS, 第一次登陆初始密码是什么？ROS 的默认用户名密码是什么？

答：用户名是“admin”，并且没有密码（点击“Enter”键）。您可以进入 system-password 修改密码。

3. 简述 ros 中 Topic(话题) 和 Service (服务) 的区别

答：

| | Topic | Service |
|------|----------------------------|---------------------|
| 通信方式 | 异步通信 | 同步通信 |
| 实现原理 | TCO/IP | TCP/IP |
| 通信模式 | Publish-Subscribe | Request-Reply |
| | Publisher-Subscriber (多对多) | Client-Server (多对一) |

| | | |
|------|-------------------------|----------------------|
| | 接受者受到数据会触发回调 (callback) | 远程过程调用 (RPC) 服务器端的服务 |
| 应用场景 | 连续、高频的数据发布 | 偶尔调用的功能呢个、具体的任务 |

Topic 发布一个消息后，就直接去执行后面的程序；而 Service 调用一个服务，会一直等待结果。

4. 试举例 ros 中两种通讯机制的应用场景

答：话题通讯机制：激光雷达、里程计发布数据

服务通讯机制：开关传感器、拍照、逆解计算

5. 在话题通讯机制功能包的实验中，在节点建立连接之后，关掉 ROS Master 会对数据传输有影响吗？其他节点还能加入这两个节点之间的网络吗？

答：对现有节点之间的传输没有影响，但是新的节点无法再加入两者的网络中。

第三章.智行 Mini 控制功能的实现

3.1 实验目的:

了解手柄控制机器人运动功能实现的详细逻辑

3.2 实验要求:

复习话题节点相关概念，并在本课程中加以理解

能够实现用键盘和手柄控制机器人移动

3.3 实验工具:

个人电脑一台，智行 mini 及其配件

3.4 实验内容:

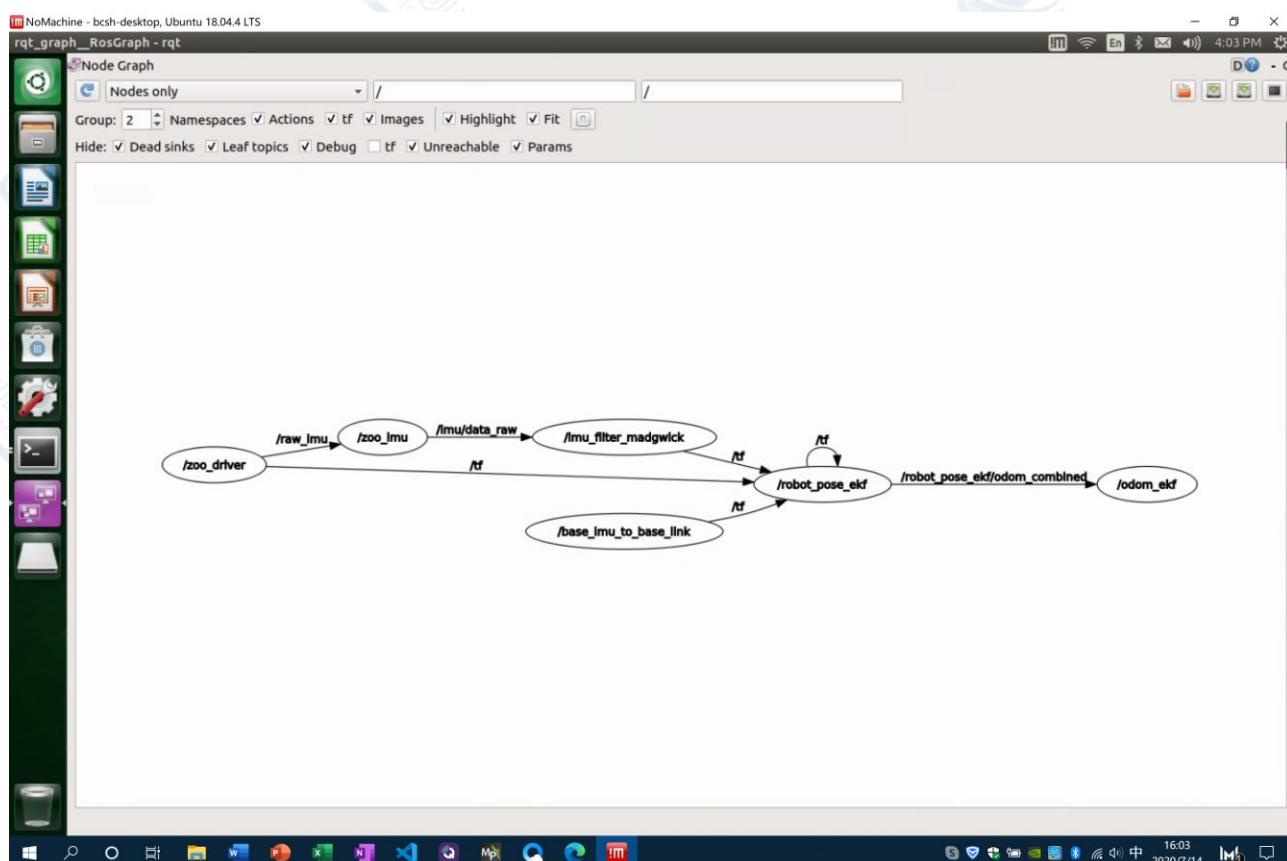
3.4.1 使用键盘控制智行 Mini 运动

检查机器人底盤能否正常运动

打开新终端，输入 `roslaunch zoo_bringup bringup_with_imu.launch` 进行底盤的连接

```
[ INFO] [1594713394.263704115]: end sleep
[ INFO] [1594713394.276762558]: robot version:v1.0.2 build time:20180730-m0e0
[ INFO] [1594713394.291803370]: subscribe cmd topic on [cmd_vel]
[ INFO] [1594713394.300279100]: advertise odom topic on [wheel_odom]
[ INFO] [1594713394.322775265]: RobotParameters: 115 250 3960 10 320 2700 0 10 2
50 40 40 200 69
[ INFO] [1594713394.610596549]: Initializing Odom sensor
[ WARN] [1594713394.626724874]: Calibrating accelerometer and gyroscope, make sure robot is stationary and level.
[ INFO] [1594713394.642479601]: Odom sensor activated
[ INFO] [1594713394.647180857]: Kalman filter initialized with odom measurement
[ INFO] [1594713394.691421]: Publishing combined odometry on
[ WARN] [1594713402.339557807]: Still waiting for data on topic /imu/data_raw...
[ INFO] [1594713410.597530620]: Calibrating accelerometer and gyroscope complete
.
[ INFO] [1594713410.597629944]: Bias values can be saved for reuse.
[ INFO] [1594713410.597687289]: Accelerometer: x: 0.217433, y: -0.258877, z: -1.
389072
[ INFO] [1594713410.597734737]: Gyroscope: x: -0.010785, y: 0.048404, z: 0.01723
6
[ INFO] [1594713410.629894044]: First IMU message received.
[ INFO] [1594713410.630537489]: Initializing Imu sensor
[ INFO] [1594713410.694075676]: Imu sensor activated
```

可以打开新终端输入 `rqt_graph` 来查看当前的节点以及消息之间的关系，得到下图：（图中椭圆内代表节点，方框内代表节点之家传递的消息）

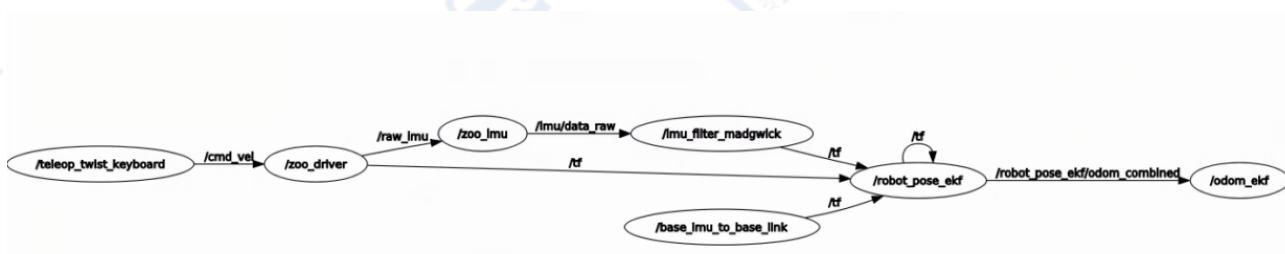


下面打开一个新终端，输入 `roslaunch zoo_control keyboard_teleop.launch` 打开键盘控制节点：

```

process[teleop_twist_keyboard-1]: started with pid [12482]
Reading from the keyboard and Publishing to Twist!
-----
Moving around:
 u i o
 j k l
 m , .
For Holonomic mode (strafing), hold down the shift key:
-----
 U I O
 J K L
 M < >
t : up (+z)
b : down (-z)
anything else : stop
q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%
CTRL-C to quit
currently: speed 0.2 turn 1.0
  
```

这时候就可以按照终端上的指示通过键盘上的 uiujklm,.等这些按键来控制小车移动了。这时候可以再使用 rqt_graph 查看会得到以下关系：



可以看出和上面的区别在于/teleop_twist_keyboard 节点发送给了/zoo_driver 这一节点一个名为/cmd_vel 的消息，说明我们的按键指令实际上就是在/teleop_twist_keyboard 节点内通过翻译转化成为/zoo_driver 驱动节点可以识别的/cmd_vel 消息，那么对于手柄来说我们可以大胆假设，可能也是通过一个节点将指令转换成/cmd_vel 消息发送给/zoo_driver 驱动节点。

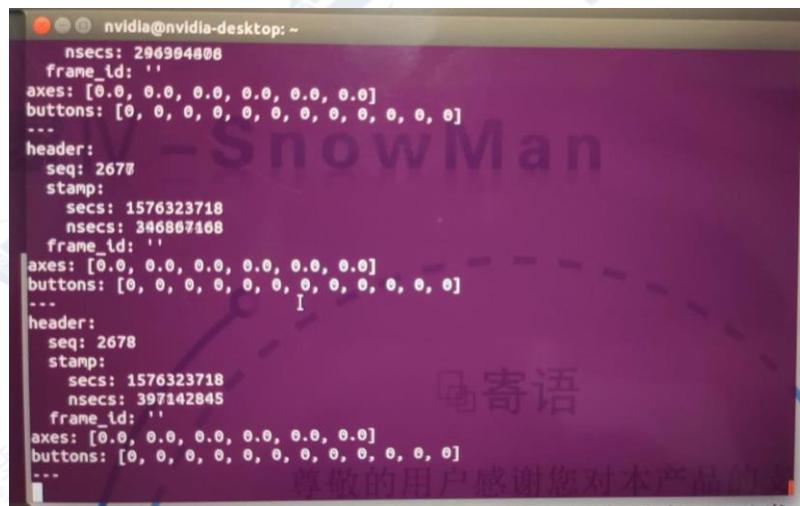
3.4.2 使用手柄控制智行 mini

检查手柄是否能正常使用

将手柄接收端插入 USB 端口，打开一个终端输入 `ls /dev/input/js*`，一般会输出 js0 或者 js1，输出哪个说明手柄接收端口是哪个。

然后输入 `roslaunch zoo_control joystick.launch`，重新打开两个个终端分别输入 `rostopic echo /joy` 和 `rostopic echo /cmd_vel` 分别监视手柄消息和速度的消息。

手柄消息显示如图：

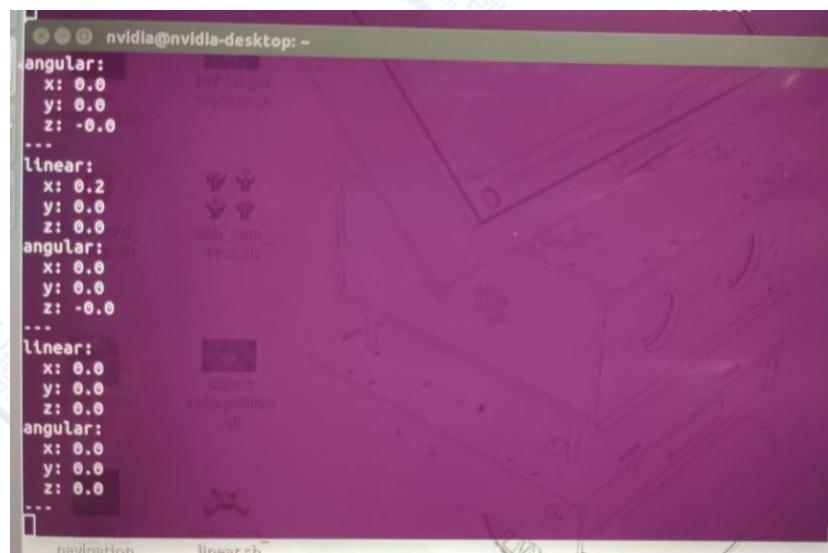


```

nvidia@nvidia-desktop: ~
  nsecs: 296994806
  frame_id: ''
axes: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
buttons: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
...
header:
  seq: 2677
  stamp:
    secs: 1576323718
    nsecs: 346867168
  frame_id: ''
axes: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
buttons: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
...
header:
  seq: 2678
  stamp:
    secs: 1576323718
    nsecs: 397142845
  frame_id: ''
axes: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
buttons: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
...

```

速度消息显示如图：



按下手柄上下左右方向键，axes 数组倒数第 1、2 个数会有变化（1 或-1），滑动方向摇杆数组第 1，2 个数会有变化。（使用时使用的是方向摇杆），滑动摇杆/按下方向键，观察/cmd_vel 是否有数字，有数字则手柄的测试到此结束，关闭所有正在运行的终端，下面可以正式通过手柄来遥控机器人了。

打开一个终端，输入 **roslaunch zoo_bringup bringup_with_imu.launch** 进行底盘的连接，接着打开新终端输入 **roslaunch zoo_robot joystick.launch** 打开手柄控制节点，如果之前的测试都没问题，那么现在就可以正常的使用手柄来控制机器人的移动了。

可以打开 **rostopic echo /cmd_vel** 监视速度的消息，只要/cmd_vel 消息不为 0，对应的机器人就会运动。

```

z: -0.0
...
linear:
  x: 0.2
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: -0.0
...
linear:
  x: 0.2
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: -0.0
...
linear:
  x: 0.0586208224297
  y: 0.0
  z: 0.0

```

3.4.3 手柄控制节点分析

在第(一、二)步的实验中我们发现键盘控制节点可以直接完成键盘指令到/cmd_vel 消息的转化，而手柄则需要两个节点的转化，下面我们依次来观察这两个转化过程。

手柄控制中消息的发布

我们先打开目录/home/nvidia/ros_ws/src/mini4_bot/launch 下的 launch 文件 joystick.launch

```

<launch>
  <arg name="joy_config" default="joystick" />
  <arg name="joy_dev" default="/dev/input/js0" />
  <arg name="config_filepath" default="$(find mini4_bot)/config/${arg joy_config}.yaml" />

  <node pkg="joy" type="joy_node" name="joy_node">
    <param name="dev" value="$(arg joy_dev)" />
    <param name="deadzone" value="0.3" />
    <param name="autorepeat_rate" value="20" />
  </node>

  <node pkg="teleop_twist_joy" name="teleop_twist_joy" type="teleop_node" output="screen">
    <rosparam command="Load" file="$(arg config_filepath)" />
  </node>
</launch>

```

发现在其中会打开两个节点，但这两个标准节点都是在安装 ROS 时已经通过 apt 功能安装过的，智行 mini 中仅有可以使用的可执行文件，想要看源码需要在 ROSwiki 上查找。这里仅做简单介绍。

joy_node 节点：

简介：该节点通过接受手柄的按键信息，将发布消息/joy,该消息包含了操作杆每个按钮和轴当前的状态。

一些参数：

a): **dev** 手柄的输入接口，这里是/dev/inout/js0

b): deadzone 死区设置，0.3 表示将操纵杆移动到 30%的偏移值其数值才会发生改变

c): autorepeat_rate 操纵杆发送数据的频率

teleop_twist_joy 节点：

简介：该节点会订阅/joy 消息将其整理并发布为/cmd_vel 消息

一些参数：

a): axis_linear 用于线性运动控制的操纵杆轴。

b): scale_linear 线性运动速度大小的控制

c): axis_angular 用于旋转运动控制的操纵杆轴。

d): scale_angular 用于角速度大小的控制

3.5 实验结果：

可以通过键盘和手柄控制机器人的移动，并了解其消息的传递

3.6 实验报告：

实验目的

实验要求

实验内容

实验总结

第四章.激光雷达驱动与滤波

4.1 实验目的:

了解激光雷达原理，发布数据的类型以及使用方法

4.2 实验要求:

能够学会用 laser_filters 包屏蔽部分激光雷达数据

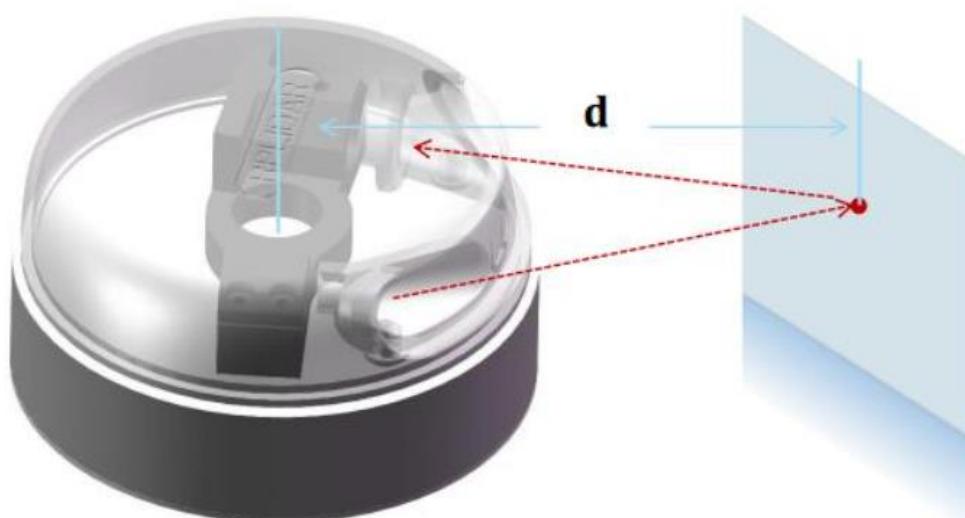
4.3 实验工具:

个人电脑一台，智行 mini 及其配件

4.4 实验内容:

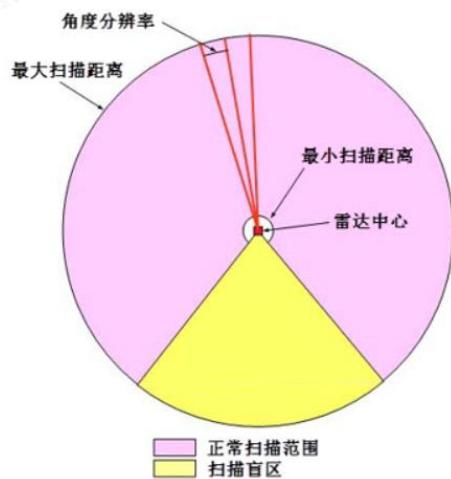
4.4.1 激光雷达测距原理

在激光雷达技术领域中，目前主要通过三角测距法与 TOF 方法来进行测距。在执行 mini 中我们所用的思岚激光雷达所采用的是三角测距法，以下简单介绍：



如图所示，由发射装置和接收装置所获得的时间差，以及两装置之间的距离就可以计算出障碍点到激光雷达的具体距离了。

激光雷达工作时会先在当前位置发出激光并接收反射光束，解析得到距离信息，而后激光发射器会转过一个角度分辨率对应的角度再次重复这个过程。限于物理及机械方面的限制，激光雷达通常会有一部分“盲区”。使用激光雷达返回的数据通常可以描绘出一幅极坐标图，极点位于雷达扫描中心，0-360° 整周圆由扫描区域及盲区组成。在扫描区域中激光雷达在每个角度分辨率对应位置解析出的距离值会被依次连接起来，这样，通过极坐标表示就能非常直观地看到周围物体的轮廓，激光雷达扫描范围示意图可以参见下图。



目前，移动机器人的研究中已经大量使用激光雷达辅助机器人的避障导航，通常激光雷达都会提供 ROS 驱动，将消息/LaserScan 发布到话题/Scan 中。

4.4.2 LaserScan 消息解析

LaserScan 消息结构如下所示

```

Header header          # timestamp in the header is the acquisition time of
# the first ray in the scan.
#
# in frame frame_id, angles are measured around
# the positive Z axis (counterclockwise, if Z is up)
# with zero angle being forward along the x axis

float32 angle_min      # start angle of the scan [rad]
float32 angle_max      # end angle of the scan [rad]
float32 angle_increment # angular distance between measurements [rad]

float32 time_increment   # time between measurements [seconds] - if your scanner
# is moving, this will be used in interpolating position
# of 3d points
float32 scan_time        # time between scans [seconds]

float32 range_min        # minimum range value [m]
float32 range_max        # maximum range value [m]

float32[] ranges          # range data [m] (Note: values < range_min or > range_max should be discarded)
float32[] intensities    # intensity data [device-specific units]. If your
# device does not provide intensities, please leave
# the array empty.

```

其中：

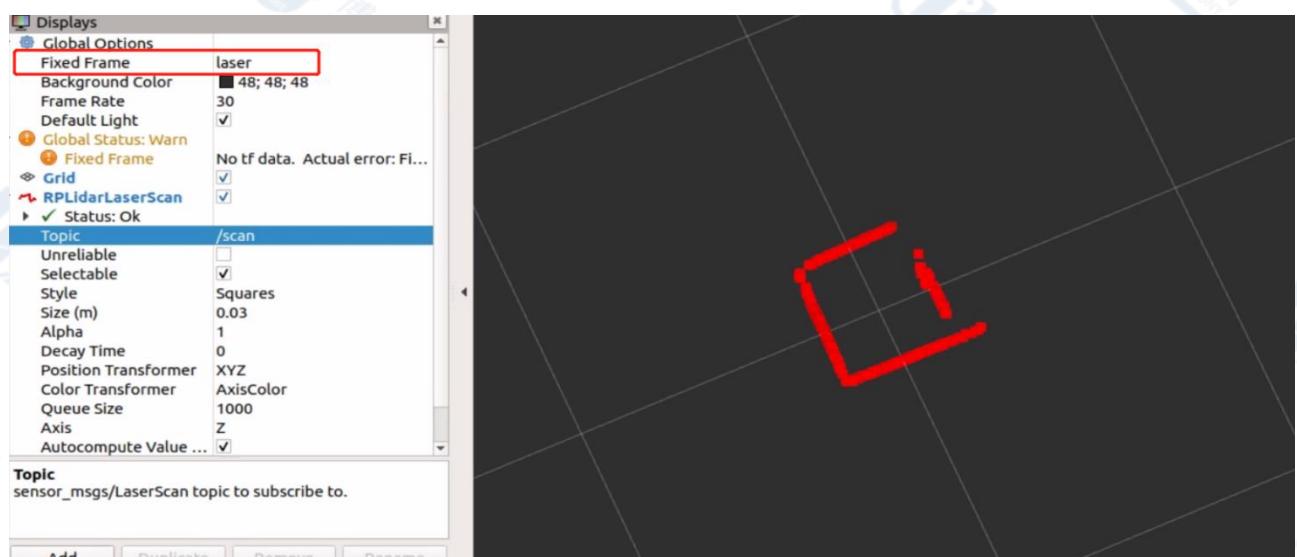
angle-min: 扫描起始角度
angle-max: 扫描终止角度
angle-increment: 两次测量的角度差
time-increment: 两次测量的时间间隔
scan-time : 完成一次扫描的时间
range-min: 测距的最小值
range-max: 测距的最大值
ranges: 转一周的测量数据, 一共 360 个
intensities: 随影的 ranges 数据的置信度, 同样 360 个

4.4.3laser_filters 包的使用

在我们实际使用激光雷达的时候, 激光雷达一般放置在机器人一个平台上, 除非放置在机器人最上方的平台, 否则中部或底部的话都会都支柱在激光雷达的扫描范围内, 而我们肯定是不希望有这些干扰的, laser_filters 包就提供了将一些干扰性的激光数据去除的功能。

首先打开智行 mini 的激光雷达数据:

开启终端输入: `roslaunch rplidar_ros view_rplidar.launch` 打开激光雷达数据



得到智行 mini 周围的障碍信息 (这里为了展示效果, 将智行 mini 围起来了)。

接着我们打开功能包 `zoo_robot/params` 下的 `box_filter.yaml` 文件, 在文件中修改包围盒的几何尺寸并保证 `fixed_frame` 一致:

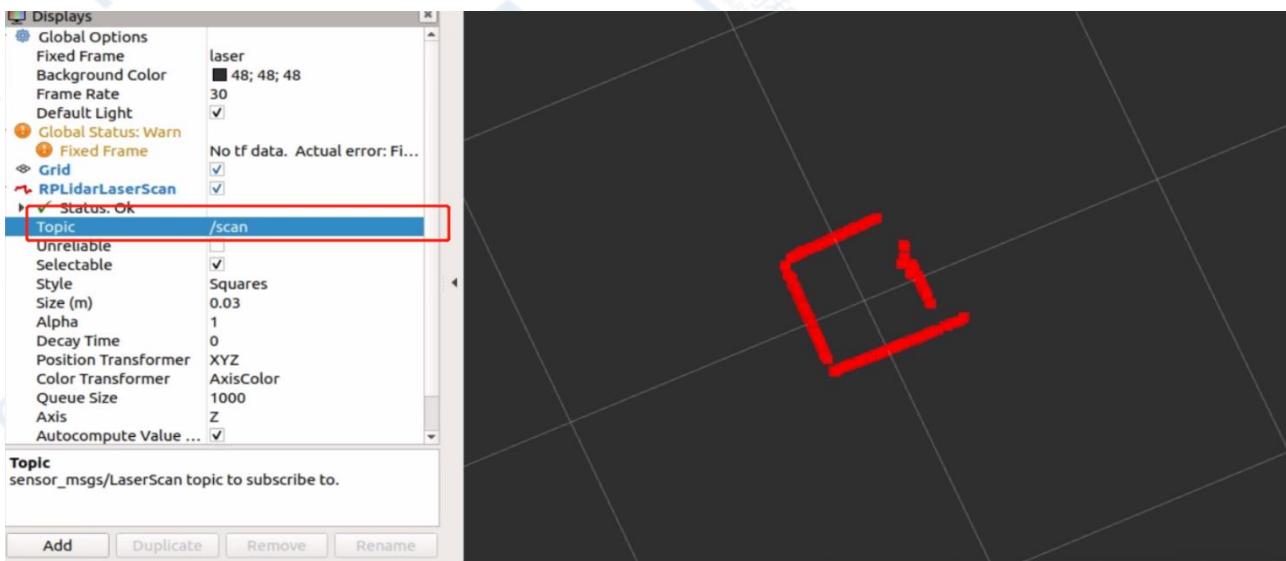
```

scan_filter_chain:
- name: box_filter
  type: laser_filters/LaserScanBoxFilter
  params:
    box_frame: laser
    min_x: -0.20
    max_x: 0.20
    min_y: -0.20
    max_y: 0.20
    min_z: -0.1
    max_z: 0.1
  
```

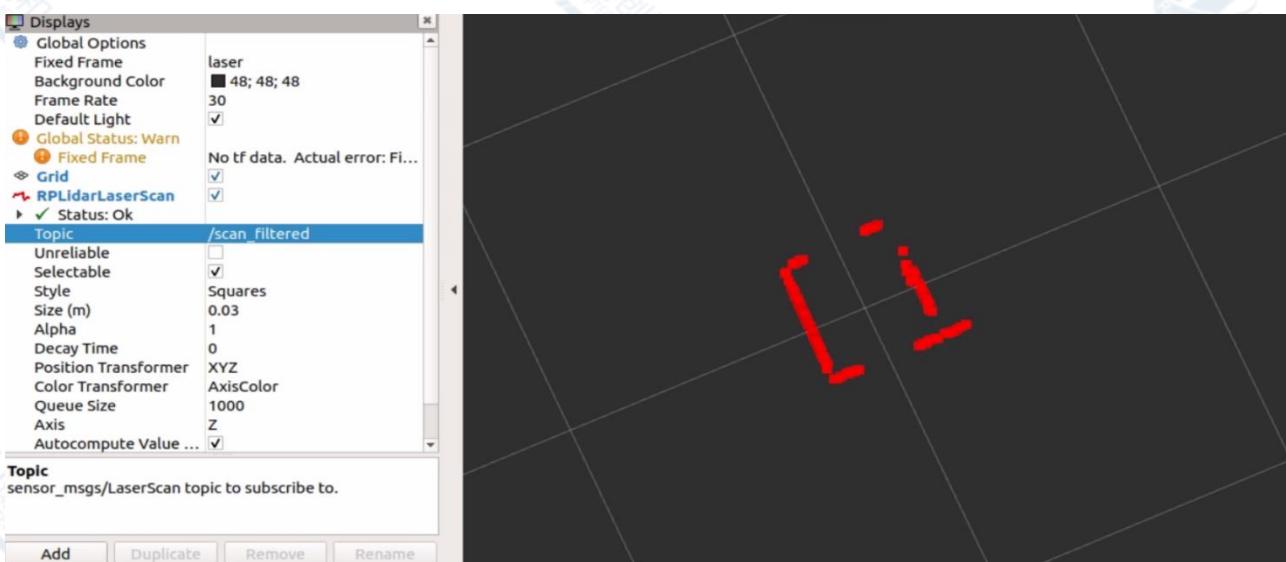
在该包围盒内的激光雷达数据将被屏蔽，保存之后打开新终端，输入：

`roslaunch zoo_robot box_filter.launch`

观察 rviz：



目前并没有什么变化，点击 topic 后的/scan 发现新增一个下拉菜单，选择/scan_filtered, 得到以下结果：



可以得到 scan_filtered 得到的结果即为屏蔽干扰后激光雷达的数据，后续我们实现 slam，避障等功能的时候，应该使用屏蔽后的数据。

4.5 实验结果：

能够以 box 屏蔽为例，在实际激光雷达数据以及模拟激光雷达数据中使用 laser_filters 包

4.6 实验报告：

实验目的

实验要求

实验内容

实验总结

第五章.激光雷达避障

5.1 实验目的:

巩固激光雷达的工作原理，利用其发布的消息加以实际应用

5.2 实验要求:

根据实验步骤完成智行 mini 遇障停止的功能

5.3 实验工具:

个人电脑一台，智行 mini 及其配件

5.4 实验内容:

5.4.1 功能要求与分析

要求智行 mini 能够接受指令后开始前进，当发现前方有障碍时停止。分析此功能，发现其需要完成的任务有三个，一是能够接受指令后开始移动；二是能够判断前方是否有障碍物；三是前方遇到障碍物时停止。下面依次进行分析，首先第一点接受指令开始移动，与第三节中讲述的键盘控制类似，可以仿照键盘控制程序来进行：当接收到某一键盘指令时发布持续发布/cmd_vel 的消息让智行 mini 向前移动。第二点判断前方是否有障碍物就可以通过激光雷达获得的数据来确定，可以依据第四节所讲述的功能包，让激光雷达仅接受前方的数据即可。至于第三点，则只要考虑当激光雷达获取的前方的距离数据小于一定值时发布新的/cmd_vel 消息，让小车停止即可。（所针对的一，二，三点可能有其他实现方法，希望发挥自己的创意）。下面以此为例，完成该功能。

5.4.2 功能的实现

首先编写一个简单的 py 文件来实现按下一个按键后小车直线运动，代码如下：

```
#!/usr/bin/env python
import rospy
```

```
from geometry_msgs.msg import Twist
from sensor_msgs.msg import LaserScan
import sys, select, os
if os.name == 'nt':
    import msvcrt
else:
    import tty, termios
check_forward_dist = 0.5
flag = 0
flagshow = 0

msg = ""

g : for go
s : for stop

CTRL-C to quit

"""

def getKey():
    if os.name == 'nt':
        return msvcrt.getch()

    tty.setraw(sys.stdin.fileno())
    rlist, _, _ = select.select([sys.stdin], [], [], 0.1)
    if rlist:
        key = sys.stdin.read(1)
    else:
        key = ''

    termios.tcsetattr(sys.stdin, termios.TCSADRAIN, settings)
    return key

def vels(target_linear_vel, target_angular_vel):
    return "currently:\tllinear vel %s\t angular vel %s " % (target_linear_vel*0.2,target_angular_vel*0.2)

def callback(data):
    global flag
    global flagshow
    if flagshow:

        print(data.ranges[179], 'flag:', flag)
        print("\r")
```

```
if data.ranges[179]<check_forward_dist:  
    flag = 1  
    flagshow = 0  
  
if __name__=="__main__":  
    if os.name != 'nt':  
        settings = termios.tcgetattr(sys.stdin)  
    rospy.init_node('turtlebot3_teleop')  
    pub = rospy.Publisher('cmd_vel', Twist, queue_size=10)  
    sub = rospy.Subscriber("scan_filtered", LaserScan, callback,queue_size = 1)  
    status = 0  
    target_linear_vel = 0.0  
    target_angular_vel = 0.0  
    try:  
        print(msg)  
        while(1):  
            key = getKey()  
            if key == 'g' :  
                flagshow = 1  
                #flag = 0  
                target_linear_vel = 1  
                status = status + 1  
                print(vels(target_linear_vel,target_angular_vel))  
  
            elif key == ' ' or key == 's' :  
                flagshow = 0  
                target_linear_vel = 0.0  
  
                target_angular_vel = 0.0  
  
                print(vels(target_linear_vel, target_angular_vel))  
            else:  
                if (key == '\x03'):  
                    break  
                if status == 20 :  
                    print(msg)  
                    status = 0  
                twist = Twist()  
                control_linear_vel = 0.2*target_linear_vel  
                twist.linear.x = control_linear_vel; twist.linear.y = 0.0; twist.linear  
.z = 0.0  
                twist.angular.x = 0.0; twist.angular.y = 0.0; twist.angular.z = 0.0  
                if flag:  
                    twist.linear.x = 0  
                print("STOP!!!!!!!!!!!!!!\r\n")
```

```
    pub.publish(twist)

except:
    print(e)

finally:
    twist = Twist()
    twist.linear.x = 0.0; twist.linear.y = 0.0; twist.linear.z = 0.0
    twist.angular.x = 0.0; twist.angular.y = 0.0; twist.angular.z = 0.0
    pub.publish(twist)

if os.name != 'nt':
    termios.tcsetattr(sys.stdin, termios.TCSADRAIN, settings)
```

完成该 py 文件后用第四节讲述的方法给该文件赋予权限，然后在 launch 文件夹下编写 go_and_stop.launch 文件如下：

```
1. <launch>
2.
3. <node pkg="zoo_robot" type="go_and_stop.py" name="go_and_stop" output="screen">
4.   <param name="speed" value="0.2" type="double"/>
5.   <param name="turn" value="1.0" type="double"/>
6. </node>
7.
8. </launch>
```

接着就可以打开终端，先运行 **roslaunch zoo_robot robot_lidar.launch** 连接地盘

然后打开新终端，运行 **roslaunch zoo_robot go_and_stop.launch**

即可得到如下界面：

```
① ② ③ /home/nvidia/ros_ws/src/minibot/launch/go_and_stop.launch http://localhost:11311

PARAMETERS
  * /go_and_stop/speed: 0.2
  * /go_and_stop/turn: 1.0
  * /rosdistro: melodic
  * /rosversion: 1.14.3

NODES
/
  go_and_stop (minibot/go_and_stop.py)

ROS_MASTER_URI=http://localhost:11311

process[go_and_stop-1]: started with pid [13476]

g : for go
s : for stop
CTRL-C to quit
```

这时只要按下键盘的 g, 智行 mini 就会开始向前移动, 当前方半米出现障碍物时停止移动。s 为急停按钮。

5.5 实验结果:

完成了小车自主运动, 遇障停止的功能。

5.6 实验报告:

实验目的

实验要求

实验内容

实验总结

第六章.里程计与坐标变换

6.1 实验目的:

理解 odom 的形成与双轮差分运动模型的运动分析，对坐标变换有所了解。

6.2 实验要求:

6. 能够理解 map、odom、base_link、base_laser 坐标系以及坐标系间的关系
7. 能够理解双轮差分运动模型的运动学分析
8. 能够理解 tf 框架的作用

6.3 实验工具:

个人电脑一台，智行 mini 及其配件

6.4 实验内容:

6.4.1ROS 机器人中一些坐标系的介绍

map 坐标系

地图坐标系，顾名思义，一般设该坐标系为固定坐标系（fixed frame），一般与机器人所在的世界坐标系一致。map 坐标系是一个世界固定坐标系，其 Z 轴指向上方。相对于 map 坐标系的移动平台的姿态，不应该随时间显著移动。map 坐标是不连续的，这意味着在 map 坐标系中移动平台的姿态可以随时发生离散的跳变。

典型的设置中，定位模块基于传感器的监测，不断的重新计算世界坐标中机器人的位姿，从而消除偏差，但是当新的传感器信息到达时可能会跳变。

map 坐标系作为长期的全局参考是很有用的，但是跳变使得对于本地传感和执行器来说，其实是一个不好的参考坐标。

odom 坐标系

odom 坐标系是一个世界固定坐标系。在 odom 坐标系中移动平台的位姿可以任意移动，没有任何界

限。这种移动使得 `odom` 坐标系不能作为长期的全局参考。然而，在 `odom` 坐标系中的机器人的姿态能够保证是连续的，这意味着在 `odom` 坐标系中的移动平台的姿态总是平滑变化，没有跳变。

在一个典型设置中，`odom` 坐标系是基于测距源来计算的，如车轮里程计，视觉里程计或惯性测量单元。

`odom` 坐标系作为一种精确，作为短期的本地参考是很有用的，但偏移使得它不能作为长期参考。

注意 `odom` 坐标系的引入是为了消除由于器件、结构等方面的原因，通过运动反馈获得的里程信息中出现的累计误差。在一开始 `map` 和 `odom` 的坐标系是重合的，但随着机器人的运动，一些误差的产生 `odom` 坐标系就会不再与 `map` 重合了，而利用一些校正传感器求 `map`→`odom` 的坐标变换过程（tf），也就是对里程计的累积误差进行消除的过程。

`odom` 还要与 `odom topic` 作区分，`odom` 是坐标系，而 `odom topic` 则是 `odom-->base_link` 的 tf 关系，也即机器人在 `odom` 坐标系下的位姿坐标。该位姿坐标的具体求法在后续会介绍。

base_link 坐标系

机器人本体坐标系，是用于表示机器人本身位姿的坐标系，一般与机器人中心重合，当然有些机器人（PR 2）是 `base_footprint`，其实是一个意思。机器人的定位问题最本质的就是求出 `base_link` 在 `map` 坐标系下的坐标和姿态。

laser 坐标系

激光雷达的坐标系，与激光雷达的安装点有关，其与 `base_link` 的 tf 是固定的。

6.4.2 双轮差分运动模型的运动学解算

底盘说明

轮式移动机器人主要有两轮差速移动和多轮全向移动两种方式，在满足一些移动要求的条件下，智行 mini 采用的是两轮差速移动的运动方式。两轮差速底盘有两个独立的动力轮位于底盘两侧，通过给两个轮子设定不同大小、方向的转速来实现机器人的前进、后退、以及转向的功能。除了两个驱动轮外，为了保持稳定，一般还会配有一到两个辅助支撑的万向轮，智行 mini 中配有的是一个球形轮。

底盘任务分析

进行运动学分析之前我们要清楚对于差动轮的使用我们需要有哪些对应关系：

1. 首先，机器人需要根据我们所给出的指令来执行相应的动作，例如向前，向左这类的运动指令。而与之对应的动作是由电机完成的，那么首先，我们就需要有机器人的速度到两个电机转速的对应关系。

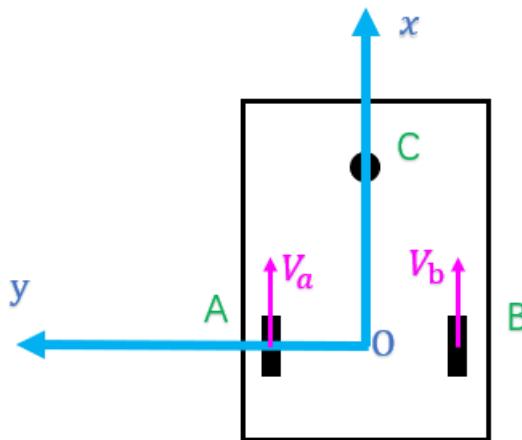
2. 其次，ROS 机器人不仅能实现手动遥控的任务，他还能自行的建图与导航，而实现这些功能的时候，必须要有底盘的里程计信息，所谓的里程计能够记录自己从运动开始到当前位置所产生的位移。这就需要有从各个电机的转速到机器人位移的变换关系

底盘运动学分析

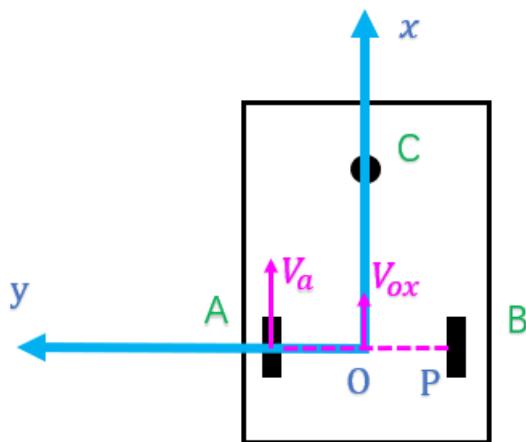
差动轮所选用的车轮为一般的标准轮，在进行运动学建模前，我们作出如下假设：

1. 轮子的平面总是和地面保持垂直，并且在任何时候，轮子与地面之间只有一个单独的接触点。
2. 该接触点无滑动，即只存在纯滚动。

所以我们可以建立运动学模型如下：



其中 A、B 分别为两个动力轮，C 为辅助的全向轮平面内没有约束，A、B 轮的速度分别设为 V_a 、 V_b ，并且方向如图为正并设机器人角速度逆时针为正。我们以 A、B 轮的中点作为机器人坐标系的原点，并建立机器人的随体坐标系如图。由运动学性质可知，在两个个轮子的运动下整体的运动等同于单个轮子作用下运动的矢量和，故我们先就一个轮子来分析，首先假设 B 轮不动，只有 A 轮运动，如下：



A 轮有一个速度 V_a , B 轮不动, 说明 B 轮与地面接触点静止, 由运动学原理可知, 该接触点就是机器人该瞬时的速度瞬心, 设为 P, 设 OP 距离为 L 则有:

$$\frac{V_{ox}}{L} = \frac{V_{ax}}{2L}$$

$$V_{oy} = 0$$

$$\dot{\theta} = -\frac{V_a}{2L}$$

同理我们可以得到当 A 轮不动, 只有 B 轮运动时的运动学方程:

$$\frac{V_{ox}}{L} = \frac{V_{bx}}{2L}$$

$$V_{oy} = 0$$

$$\dot{\theta} = \frac{V_b}{2L}$$

由上述的式子可以解得:

$$\begin{cases} V_{ox} = \frac{1}{2}(V_a + V_b) \\ V_{oy} = 0 \\ \dot{\theta} = \frac{1}{2L}(V_b - V_a) \end{cases}$$

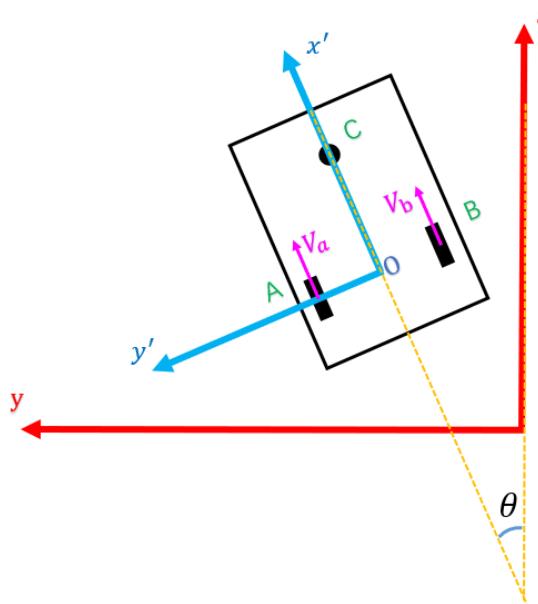
对于这个式子的理解, 我们可以举个例子, 当 A 轮的速度是 0.2m/s, B 轮的速度是-0.1m/s 时, 那么机器人沿 x 方向的速度就是 $\frac{1}{2}(0.2 - 0.1)$ m/s 即 0.05m/s, 而该瞬时角速度为 $\frac{1}{2L}(-0.1 - 0.2)$ 为-0.15/L, 可知该瞬时机器人既有向前为 0.05m/s 的速度, 也会有大小为 0.15/L 沿顺时针转动的角速度。

上述的式子为各个车轮速度到机器人速度的变换关系, 我们称之为运动学正解方程, 而由运动学正解方程我们可以直接解得运动学逆解方程 (从机器人速度到各个车轮速度的变换关系):

$$\begin{cases} V_a = V_{ox} + L\dot{\theta} \\ V_b = V_{ox} - L\dot{\theta} \end{cases}$$

利用该方程，我们就能实现底盘的任务一：实现机器人速度道两个电机转速的对应。

那么对于任务二，我们又应该如何去实现呢，各个电机的转速到机器人位移没有直接的变换关系，但我们可以先分析出各个电机转速到机器人的三个移动速度的变换，再通过在每个时间间隔内对速度积分的方式求得三个位移。不过这里要注意的是，我们不能直接使用上述求出的随体坐标系下的正解方程来作为速度变换关系，而应该使用两个电机的转速到机器人的绝对速度的变换。我们设定机器人初始位置对应的随体坐标系为绝对坐标系，在某一时刻，机器人的随体坐标系已经与绝对坐标系有了大小为 θ 的夹角，如图：



两坐标系间变换关系为：

$$\begin{cases} x = x' \cos \theta + y' \sin \theta \\ y = -x' \sin \theta + y' \cos \theta \end{cases}$$

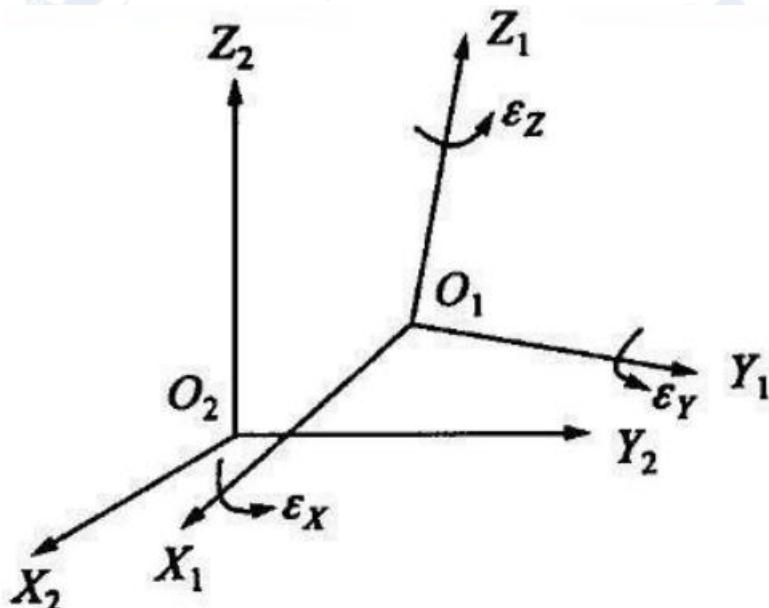
故有方程：

$$\begin{cases} V_{ox} = \frac{1}{2}(V_a + V_b) \cos \theta \\ V_{oy} = -\frac{1}{2}(V_a + V_b) \sin \theta \\ \dot{\theta} = \frac{1}{2L}(V_b - V_a) \end{cases}$$

这个方程组就是两个电机转速到地面坐标系下机器人运动速度的变换关系，计算里程时，需要在每一个时间间隔内对速度求积分，累加的和就是当前时刻的里程数据，也即发送到主机上的 odom topic 的内容。到此，关于双轮差分的运动模型结算就全部完成了。

6.4.3 tf 变换的简单介绍

坐标变换是机器人学中一个非常基础也是非常重要的概念。机器人本体和机器人的工作环境中往往存在大量的组件元素，在机器人设计和机器人应用中都会涉及不同组件的位置和姿态，这就需要引入坐标系以及坐标变换的概念。



如图，坐标系 O₁ 可以经由坐标系 O₂ 的平移和旋转得到，其理论与原理不在此过多描述。

TF 是一个让用户随时间跟踪多个坐标系的功能包，它使用树形数据结构，根据时间缓冲并维护多个坐标系之间的坐标变换关系，可以帮助开发者在任意时间、在坐标系间完成点、向量等坐标的变换。想要使用 TF 功能包，总体来说需要以下两个步骤：

1. 监听 TF 变换

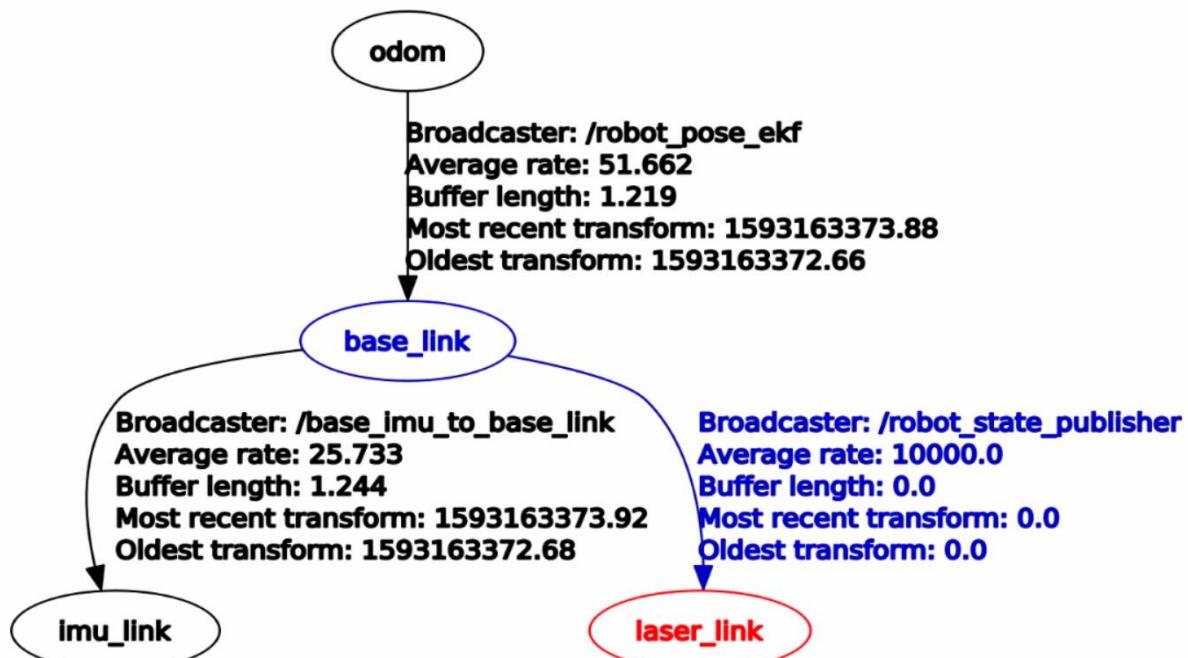
接收并缓存系统中发布的所有坐标变换关系，并从中查询所需要的坐标变换关系。

2. 广播 TF 变换

向系统中广播坐标系之间的坐标变换关系。系统中可能会存在多个不同部分的 TF 变换广播，每个广播都可以直接将坐标变换关系插入到 TF 树中，不用再进行同步。

简单来说 tf 以树的形式记录着各个坐标系间的变换方程，在实际使用的时候可以直接计算出所需要的两个坐标系间的坐标关系。

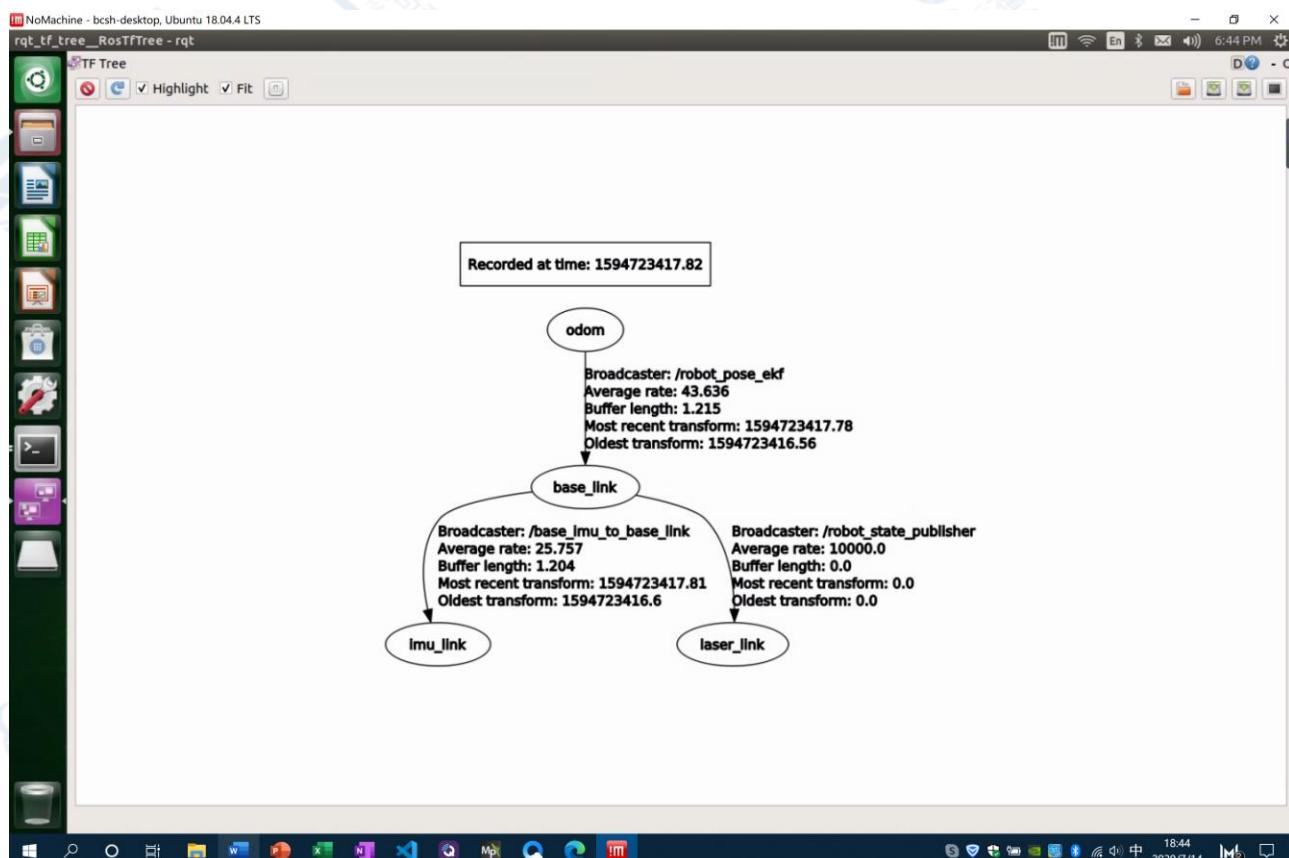
运行连接地盘程序之后可以新开启一个终端，输入 osrun rqt_tf_tree rqt_tf_tree 得到当前的 tf 树如图：



接下来我们通过智行 mini 来观察如何获取里程计和 TF 变换。

首先打开终端输入 `roslaunch zoo_robot robot_lidar.launch` 启动底盘通信。

然后打开新的终端输入 `rosrun rqt_tf_tree rqt_tf_tree`



得到 tf 变换。然后打开新的终端输入 `rostopic echo /odom`

```
bcsh@bcsh-desktop: ~
seq: 100
stamp:
  secs: 1594723531
  nsecs: 683897706
  frame_id: "/odom"
  child_frame_id: "base_link"
  pose:
    pose:
      position:
        x: 0.0
        y: 0.0
        z: 0.0
      orientation:
        x: 0.000547040539884
        y: 0.00179345524695
        z: -0.0105638027442
        w: 0.999942443512
      covariance: [2.5460030883550644e-07, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 2.5460030
3550644e-07, 0.0, 0.0, 0.0, 0.0, 0.0, 9765000000.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0
, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 7
542283374195e-12]
twist:
  twist:
    linear:
```

可以看到里程计信息，里程计信息组成如下图所示

```
# This represents an estimate of a position and velocity in free space.
# The pose in this message should be specified in the coordinate frame given by header.frame_id.
# The twist in this message should be specified in the coordinate frame given by the child_frame_id
Header header
string child_frame_id
geometry_msgs/PoseWithCovariance pose
geometry_msgs/TwistWithCovariance twist
```

其中 pose 代表机器人当前的位置和姿态，而 twist 代表机器人当前速度信息。

我们在 zoo_robot 功能包中实现一个节点，来获取当前里程计与 tf 坐标变换信息。这个节点名称为 get_pose_demo.py，源码如下所示：

```
#!/usr/bin/env python
import rospy

from tf_conversions import transformations
from math import pi
import tf

class Robot:
    def __init__(self):
        self.tf_listener = tf.TransformListener()
        try:
```

```
    self.tf_listener.waitForTransform('/odom', '/base_link', rospy.Time(),  
rospy.Duration(1.0))  
    except (tf.Exception, tf.ConnectivityException, tf.LookupException):  
        return  
  
    def get_pos(self):  
        try:  
            (trans, rot) = self.tf_listener.lookupTransform('/odom', '/base_link',  
rospy.Time(0))  
        except (tf.LookupException, tf.ConnectivityException, tf.ExtrapolationException):  
            rospy.loginfo("tf Error")  
            return None  
        euler = transformations.euler_from_quaternion(rot)  
        #print euler[2] / pi * 180  
  
        x = trans[0]  
        y = trans[1]  
        th = euler[2] / pi * 180  
        return (x, y, th)  
  
if __name__ == "__main__":  
    rospy.init_node('get_pos_demo', anonymous=True)  
    robot = Robot()  
    r = rospy.Rate(100)  
    r.sleep()  
    while not rospy.is_shutdown():  
        print robot.get_pos()  
        r.sleep()
```

如代码所示，本程序的内容为监听里程计坐标系到机器人本体坐标系 base_link 的坐标变换，这样就等于获取到里程计信息，同时打包成 (x, y, yaw) 的数组形式打印出来。其中 x 代表 ros 坐标系下机器人前后位置变化（前正后负），y 代表 ros 坐标系下机器人左右位置变化（左正右负），yaw 代表机器人的机头朝向（左正右负）。

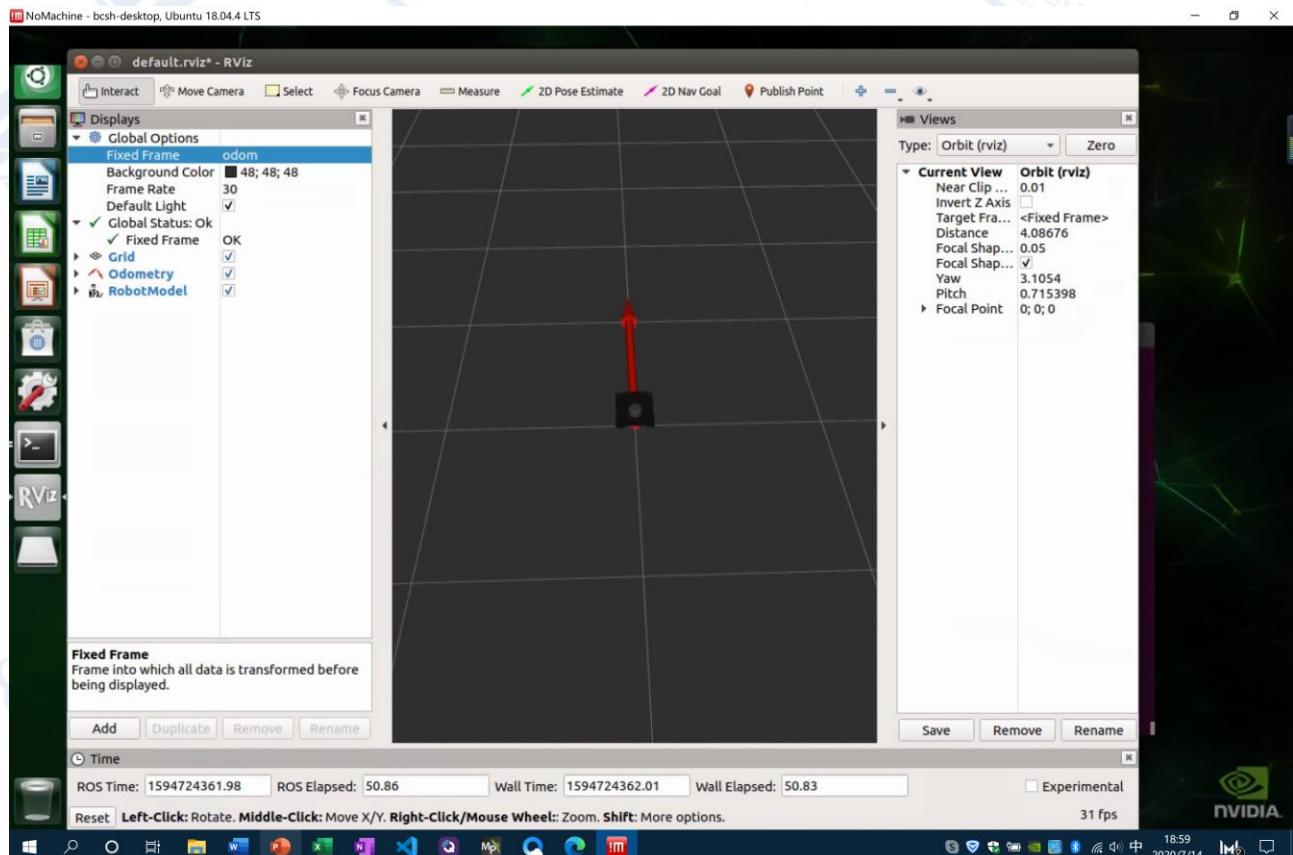
首先打开终端启动底盘通信

然后打开新的终端输入 `rosrun zoo_robot get_pos_demo.py`

这样就获取到了当前的里程计信息。

```
(0.0, 0.0, -0.06984788398830769)
(0.0, 0.0, -0.06984788398830769)
(0.0, 0.0, -0.06984788398830769)
(0.0, 0.0, -0.06984788398830769)
(0.0, 0.0, -0.07009556297265741)
(0.0, 0.0, -0.06958066552588348)
(0.0, 0.0, -0.06958066552588348)
(0.0, 0.0, -0.06958066552588348)
(0.0, 0.0, -0.06935438668807796)
(0.0, 0.0, -0.06935438668807796)
(0.0, 0.0, -0.06992787050076707)
(0.0, 0.0, -0.06992787050076707)
(0.0, 0.0, -0.06906685249547535)
(0.0, 0.0, -0.06906685249547535)
(0.0, 0.0, -0.06893572831839936)
(0.0, 0.0, -0.06893572831839936)
(0.0, 0.0, -0.06893572831839936)
(0.0, 0.0, -0.06893572831839936)
(0.0, 0.0, -0.06983013094796549)
(0.0, 0.0, -0.06983013094796549)
(0.0, 0.0, -0.06983013094796549)
(0.0, 0.0, -0.06983013094796549)
```

使用手柄或者键盘控制底盘运动，观察里程计变化。另外可在 rviz 中观察里程计变化：



红色箭头即为里程计。

6.5 实验结果：

理解 ROS 机器人中常见的一些坐标系，坐标系之间的关系，并能推导出智行 mini 中 odom topic 中的内容来源。

6.6 实验报告：

实验目的

实验要求

实验内容

实验总结

思考题

6.7 思考题：

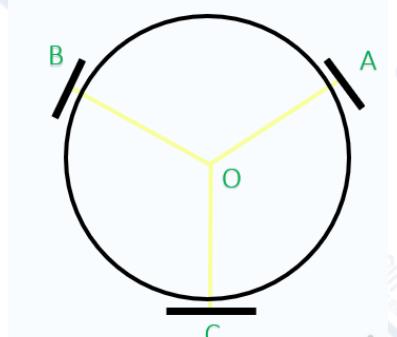
4. 请解释一下随体坐标系正解方程中的 $V_{oy}=0$ 是什么意思？

答：在这个式子 V_{oy} 一直是 0，说明沿 y 方向的速度永远为 0，代表机器人只能够前后移动，以及转弯，而无法左右平移，这就是两轮差速移动的运动限制。

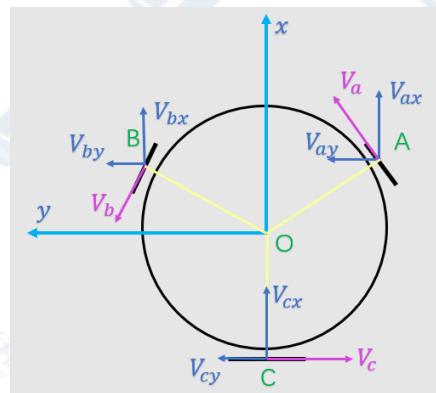
5. 对于任务 2，为什么我们不能直接使用随体坐标系下的正解方程进行结算而任务 1 却可以直接用随体坐标系下的逆解方程进行结算？

答：对于任务 1 我们对机器人所发出的命令例如向前运动，左转都是站在机器人的视角发出的指令，即该指令是在机器人随体坐标系下的发出的指令，所以可以直接使用随体坐标系下的逆解方程，但对于任务 2，关于机器人的位移，我们就应该跳出机器人的随体坐标系来看待机器人，所谓的位移应该是在地面坐标系下产生的绝对位移，因此我们需要使用地面坐标系下的正解方程进行结算。

6. 尝试对三轮全向移动机器人（底盘结构如下）进行随体坐标系的正逆解分析以及绝对坐标系下的正解分析。



答：按如图设定轮子移动方向以及坐标方向，可解得：



随体坐标系下的正解方程为：

$$\begin{cases} V_{ox} = \frac{\sqrt{3}}{3}(V_a - V_b) \\ V_{oy} = -\frac{2}{3}V_c + \frac{1}{3}(V_a + V_b) \\ \dot{\theta} = \frac{1}{3L}(V_a + V_b + V_c) \end{cases}$$

随体坐标系下的逆解方程为：

$$\begin{cases} v_a = \frac{\sqrt{3}}{2}V_{ox} + \frac{1}{2}V_{oy} + \dot{\theta}L \\ v_b = -\frac{\sqrt{3}}{2}V_{ox} + \frac{1}{2}V_{oy} + \dot{\theta}L \\ v_c = -V_{oy} + \dot{\theta}L \end{cases}$$

绝对坐标系下的正解方程为：

$$\begin{cases} V_{ox} = \frac{2}{3}V_c \sin \theta + \frac{1}{3}V_a(\sqrt{3}\cos \theta - \sin \theta) + \frac{1}{3}V_b(-\sqrt{3}\cos \theta - \sin \theta) \\ V_{oy} = -\frac{2}{3}V_c \sin \theta + \frac{1}{3}V_a(\sqrt{3}\sin \theta + \cos \theta) + \frac{1}{3}V_b(-\sqrt{3}\sin \theta + \cos \theta) \\ \dot{\theta} = \frac{1}{3L}(V_a + V_b + V_c) \end{cases}$$

第七章.轮式里程计与 imu 数据融合

7.1 实验目的:

理解 IMU 传感器在机器人定位中发挥的作用并理解传感器数据融合的方式。

7.2 实验要求:

能够弄清楚 IMU 的数据格式，了解数据含义

能够使用 robot_pose_ekf 实现 IMU 校准里程计

7.3 实验工具:

个人电脑一台，智行 mini 及其配件

7.4 实验内容:

7.4.1IMU 传感器使用的必要

在机器人的实际运动中常常会出现轮子打滑的过程，当机器人以较快的速度运动时突然停止，或是机器人的突然转向，机器人都会出现打滑。而出现打滑时，里程计所计算的坐标与机器人的实际坐标之间就会产生误差了，也就是我们上一节所说的 odom 坐标系与 map 坐标系不再重合。即使是以很小的速度去加减速、转弯，机器人也会不可避免的出现一些误差，而当这些误差累积到一定程度就会对机器人的定位产生严重的影响。因此必须采用一些辅助的传感器去帮助校正，IMU 传感器就是较为常用的一种。

7.4.2IMU 数据结构

IMU 为惯性测量单元，一般包含了三个单轴的加速度计和三个单轴的陀螺仪，简单理解通过加速度二次积分就可以得到位移信息、通过角速度积分就可以得到三个角度，实际上要比这个复杂许多。

下面来具体看一下 IMU 发布的话题以及消息，首先运行 `roslaunch zoo_robot robot_lidar.launch` 连接机器人的底盘，然后打开新终端输入 `rostopic info /imu/data_raw` 得到以下信息：

```
bcsh@bcsh-desktop:~$ rostopic info /imu/data_raw
Type: sensor_msgs/Imu

Publishers:
* /zoo_imu (http://bcsh-desktop:42893/)

Subscribers:
* /imu_filter_madgwick (http://bcsh-desktop:40165/)
```

表示该话题发布的消息为/sensor_msgs/Imu，话题的发布者为/zoo_imu,订阅者为 imu_filter_madgwick

下面来看一看消息的具体结构，输入 `rosmsg info sensor_msgs/Imu`，得到以下结果：

```
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
geometry_msgs/Quaternion orientation
  float64 x
  float64 y
  float64 z
  float64 w
  float64[9] orientation_covariance
geometry_msgs/Vector3 angular_velocity
  float64 x
  float64 y
  float64 z
  float64[9] angular_velocity_covariance
geometry_msgs/Vector3 linear_acceleration
  float64 x
  float64 y
  float64 z
  float64[9] linear_acceleration_covariance
```

其中 orientation 代表四元数，表示当前机器人姿态数据，由磁罗盘测出，angular_velocity 代表机器人当前角速度，由陀螺仪测出，linear_acceleration 代表加速度，由加速度计测出。

在底盘连接成功的基础上可以运行 `rostopic echo /imu/data_raw` 查看一下实际的数据是什么样的：

```
Z: 9.873857711
linear_acceleration_covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
---
header:
  seq: 9081
  stamp:
    secs: 1594725109
    nsecs: 108160728
  frame_id: "imu_link"
orientation:
  x: 0.0
  y: 0.0
  z: 0.0
  w: 0.0
orientation_covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
angular_velocity:
  x: 0.00147716054739
  y: 0.00236893030826
  z: -0.00110487376537
angular_velocity_covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
linear_acceleration:
  x: 0.008114128
  y: -0.037294993
  z: 9.756650211
```

通过对原始数据处理得到一个 imu/data 数据类型为 sensor_msgs/Imu，查看该话题：

接可以看到 imu_filter_madgwick 会订阅该 topic，即该 topic 作为输入滤波得到最终数据(发布 imu/data topic 类型同样为 sensor_msgs/Imu)，输入 rostopic info /imu/data 查看下一节点：

```
nvidia@nvidia-desktop:~$ rostopic info /imu/data
Type: sensor_msgs/Imu

Publishers:
  * /imu_filter_madgwick (http://nvidia-desktop:46441/)

Subscribers:
  * /robot_pose_ekf (http://nvidia-desktop:43011/)
```

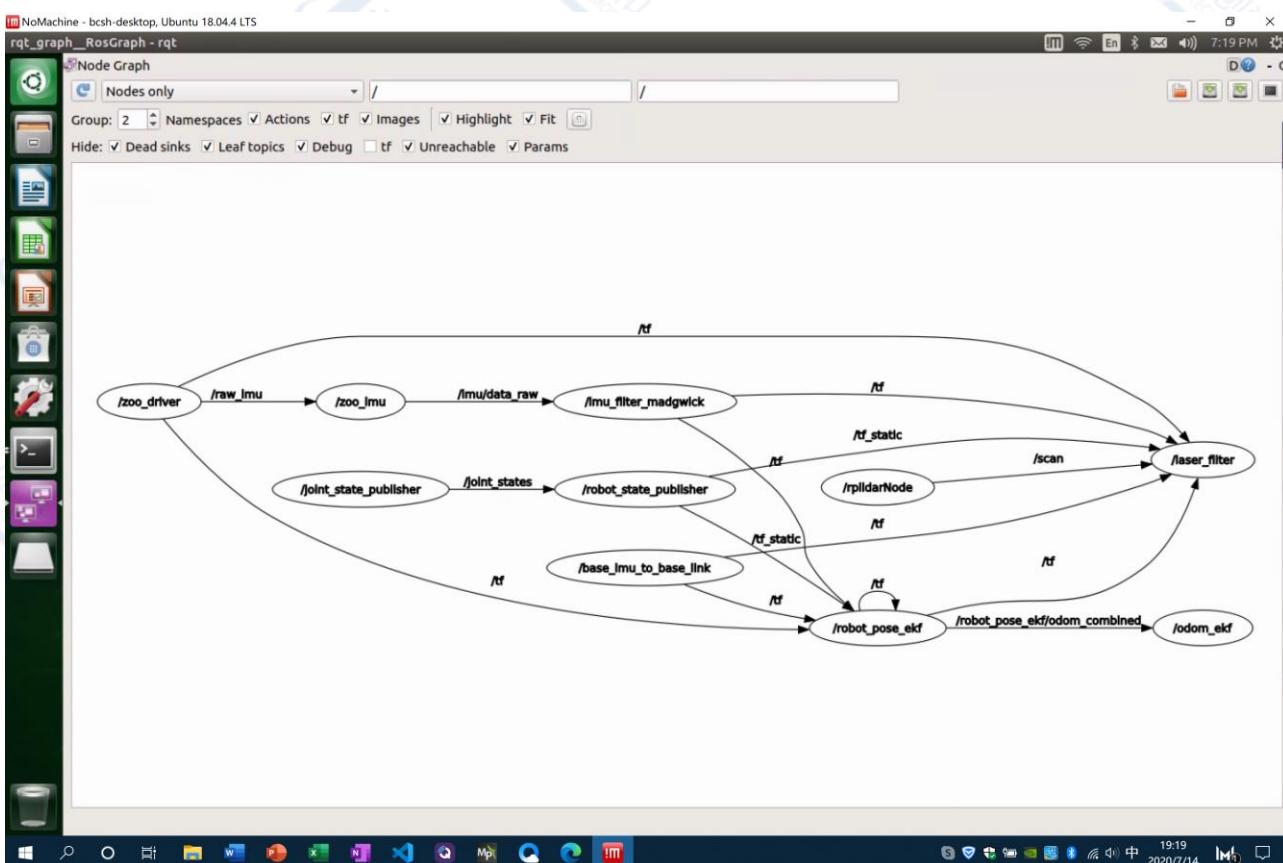
我们查看现在的话题中的 IMU 消息：rostopic echo /imu/data

```
---
```

```
header:  
  seq: 81529  
  stamp:  
    secs: 1593160996  
    nsecs: 504169721  
  frame_id: "imu_link"  
orientation:  
  x: 8.93460067388e-05  
  y: -0.00135748970669  
  z: 0.908536723256  
  w: 0.417802790481  
orientation_covariance: [1e-06, 0.0, 0.0, 0.0, 1e-06, 0.0, 0.0, 0.0, 0.0, 1e-06]  
angular_velocity:  
  x: -0.00356152880378  
  y: 0.00529744243446  
  z: -0.00632082667678  
angular_velocity_covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]  
linear_acceleration:  
  x: -0.077667122  
  y: -0.109716868  
  z: 9.794462711  
linear_acceleration_covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

可以发现现在的波动就比较小了，静止的时候接近于0。

我们可以输入 `rqt graph` 查看节点之间的逻辑关系，来理一理目前的数据流：



就这样，IMU 数据经由/zoo_driver 节点读取后发布消息并经过一系列的滤波操作，最终由/robot_pose_ekf 订阅，而/robot_pose_ekf 包将在下面介绍。

7.4.3 robot_pose_ekf 实现 IMU 校准里程计

IMU 数据的使用方法有两种，一种简单的应用就是直接使用 IMU 数据中的航向角度来计算机器人的转角，这样的话，即使打滑或者抬起机器人转一定的角度，所得到的里程也能正确的反映出来。

而我们下面要介绍的，是使用 ROS 的官方包/robot_pose_ekf 通过扩展的卡尔曼滤波来融合里程、IMU 以及机器人自身姿态信息。我们输入 `rostopic info /robot_pose_ekf/odom_combined` 来查看该话题：

```
nvidia@nvidia-desktop:~$ rostopic info /robot_pose_ekf/odom_combined
Type: geometry_msgs/PoseWithCovarianceStamped

Publishers:
* /robot_pose_ekf (http://nvidia-desktop:39331/)

Subscribers:
* /odom_ekf (http://nvidia-desktop:44173/)
```

输入 `rosnode info /odom_ekf` 查看节点信息：

```
nvidia@nvidia-desktop:~$ rosnode info /odom_ekf
Node [/odom_ekf]
Publications:
* /odom [nav_msgs/Odometry]
* /rosout [rosgraph_msgs/Log]

Subscriptions:
* /robot_pose_ekf/odom_combined [geometry_msgs/PoseWithCovarianceStamped]

Services:
* /odom_ekf/get_loggers
* /odom_ekf/set_logger_level

contacting node http://nvidia-desktop:44173/ ...
Pid: 11160
Connections:
* topic: /rosout
  * to: /rosout
  * direction: outbound
  * transport: TCPROS
* topic: /robot_pose_ekf/odom_combined
  * to: /robot_pose_ekf (http://nvidia-desktop:39331/)
  * direction: inbound
  * transport: TCPROS
```

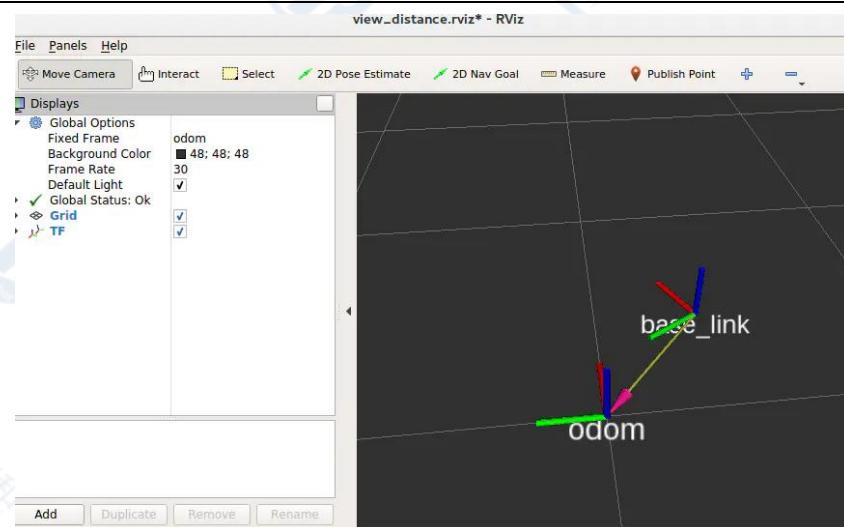
可以看到该节点会发布一个里程计的 topic。

具体的卡尔曼滤波的原理不做过多的解释，本节主要介绍的是 IMU 数据的一个信息流，以及各个节点大致的功能，最终获得能够在 IMU 传感器的参与下获得一个经过校准的 odom 数据。可以看到，轮式里程计数据为 wheel_odom, 里程计 odom 为融合 imu 之后的里程。同学们可以使用键盘或者手柄控制智行 Mini 运动，观察哪个得到的结果更加精准。这里有两种方法可以做测试：

- 手动控制机器人走一圈然后回到之前的原点，通过观察模拟器(RViz)中里程与初始点的偏差
- 程序控制机器人行走(例如走一个方形)，通过观察最终机器人时间与最初原点的偏差

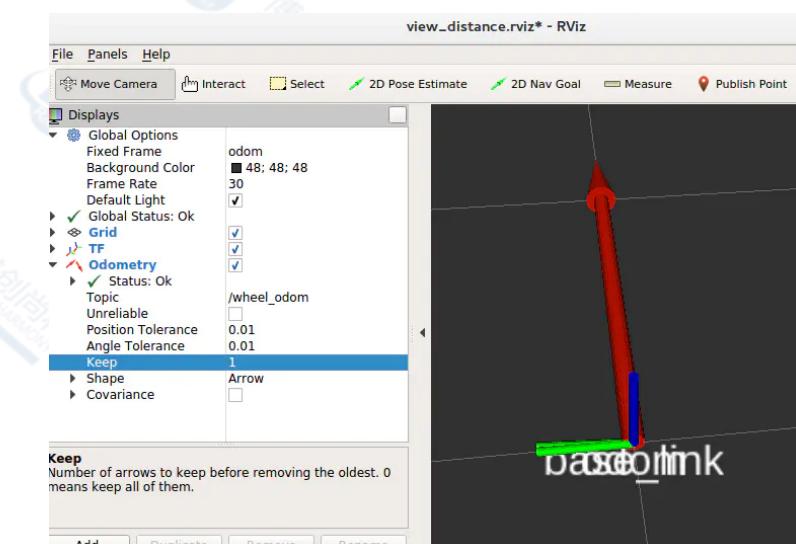
下面我们使用的都是第一种，标记个原点，控制机器人随机行走一段时间后再控制其回到标记的原点。

下面我们手动控制智行 Mini 走一圈回到原点，可以在 rviz 中看到

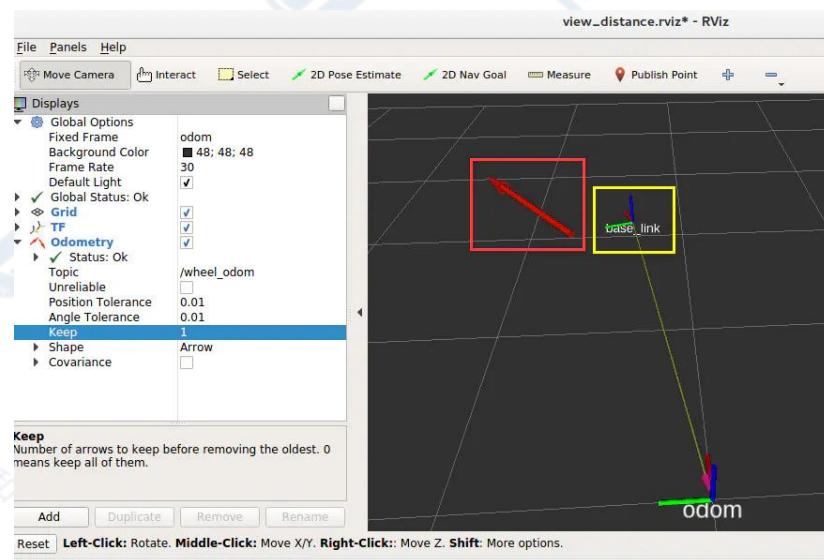


位置和姿态都存在一定误差。

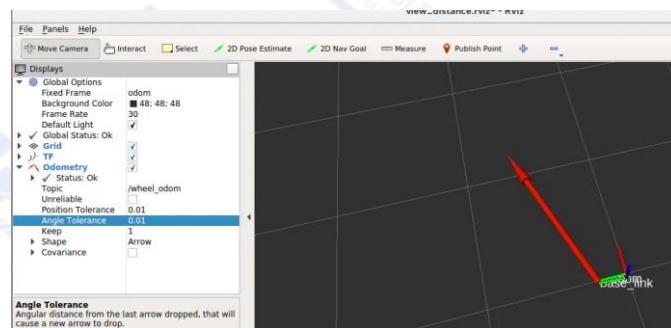
作为对比我们在使用融合 IMU 的时候，把使用编码器计算出来的里程计显示出来作对比(大的红色箭头就是编码器里程计)，同上面我们控制智行 Mini 走一圈回到原点。



初始的时候坐标原点，轮式里程计（红色箭头），融合后的里程计（base_link）的 tf 变换是重合的。



可以发现没走多远融合出来的里程（黄色框）和编码器里程（红色框）就有了较大的差距。



我们继续控制使得智行 Mini 回到原点。

7.5 实验结果：

理解 IMU 数据的信息流，理解 robot_pose_ekf 包的作用。

7.6 实验报告：

实验目的

实验要求

实验内容

实验总结

第八章.基于里程计的运动控制

8.1 实验目的:

理解里程计标定的数据，根据里程计控制机器人运行指定距离。

8.2 实验要求:

按照步骤所示，完成里程计的编程

8.3 实验工具:

个人电脑一台，智行 mini 及其配件，米尺

8.4 实验内容:

8.4.1 进行基于里程计的运动控制

在本节内容中，我们将实现以下内容

读取融合后的里程计信息，然后控制智行 Mini 运动指定距离，这里我们设为 1 米。

下面来看一下具体的操作：

我们在 zoo_robot 文件夹内编写名为 move_linear.py 的 python 文件，输入以下代码

```
#!/usr/bin/env python

import rospy
from geometry_msgs.msg import Twist, Point
from math import copysign, sqrt, pow
import tf

class MoveLinear():
    def __init__(self):
        # Give the node a name
        rospy.init_node('move_linear', anonymous=False)
        # Set rospy to execute a shutdown function when terminating the script
        rospy.on_shutdown(self.shutdown)
        # How fast will we check the odometry values?
        self.rate = rospy.get_param('~rate', 20)
```

```
r = rospy.Rate(self.rate)
# Set the distance to travel
self.test_distance = rospy.get_param('~test_distance', 1.0) # meters
self.speed = rospy.get_param('~speed', 0.15) # meters per second
self.tolerance = rospy.get_param('~tolerance', 0.01) # meters
self.start_test = rospy.get_param('~start_test', True)
# Publisher to control the robot's speed
self.cmd_vel = rospy.Publisher('/cmd_vel', Twist, queue_size=5)
# The base frame is base_footprint for the TurtleBot but base_link for Pi R
obot
self.base_frame = rospy.get_param('~base_frame', '/base_link')
# The odom frame is usually just /odom
self.odom_frame = rospy.get_param('~odom_frame', '/odom')
# Initialize the tf listener
self.tf_listener = tf.TransformListener()
# Give tf some time to fill its buffer
rospy.sleep(2)
# Make sure we see the odom and base frames
self.tf_listener.waitForTransform(self.odom_frame, self.base_frame, rospy.T
ime(), rospy.Duration(60.0))
rospy.loginfo("Bring up rqt_reconfigure to control the test.")
self.position = Point()
# Get the starting position from the tf transform between the odom and base
frames
self.position = self.get_position()
x_start = self.position.x
y_start = self.position.y
move_cmd = Twist()
while not rospy.is_shutdown():
    # Stop the robot by default
    move_cmd = Twist()
    if self.start_test:
        # Get the current position from the tf transform between the odom a
nd base frames
        self.position = self.get_position()
        # Compute the Euclidean distance from the target point
        distance = sqrt(pow((self.position.x - x_start), 2) +
                        pow((self.position.y - y_start), 2))
        # How close are we?
        error = distance - self.test_distance
        # Are we close enough?
        if not self.start_test or abs(error) < self.tolerance:
            self.start_test = False
        else:
            # If not, move in the appropriate direction
```

```
        move_cmd.linear.x = copysign(self.speed, -1 * error)
    else:
        self.position = self.get_position()
        x_start = self.position.x
        y_start = self.position.y
        self.cmd_vel.publish(move_cmd)
        r.sleep()
    # Stop the robot
    self.cmd_vel.publish(Twist())

def get_position(self):
    # Get the current transform between the odom and base frames
    try:
        (trans, rot) = self.tf_listener.lookupTransform(self.odom_frame, self.
base_frame, rospy.Time(0))
    except (tf.Exception, tf.ConnectivityException, tf.LookupException):
        rospy.loginfo("TF Exception")
        return
    return Point(*trans)

def shutdown(self):
    # Always stop the robot when shutting down the node
    rospy.loginfo("Stopping the robot...")
    self.cmd_vel.publish(Twist())
    rospy.sleep(1)

if __name__ == '__main__':
    try:
        MoveLinear()
        rospy.spin()
    except:
        rospy.loginfo("Calibration terminated.")
```

根据代码及注释可以得到，本案例通过订阅 tf 变换获取当前位置，与目标位置做一个判断，距离足够近的时候则停止运动，否则就前进。同时我们将这个脚本 chmod 可执行，然后编写 launch 文件 move_linear.launch：

```
<launch>
    <node pkg="zoo_robot" type="move_linear.py" respawn="false" name="move_linear" output="screen">
    </node>
</launch>
```

首先启动 **roslaunch zoo_robot robot_lidar.launch** 连接机器人的底盘，然后打开新终端启动程序 roslaunch zoo_robot move_linear.launch，机器人会以 0.15m/s 的速度向前行驶 1m 的距离。

8.5 实验结果：

能够完成里程计的运动控制。

8.6 实验报告：

实验目的

实验要求

实验内容

实验总结

第九章.Gmapping 建图

9.1 实验目的:

理解栅格地图的概念，建图的原理，Gmapping 建图的一些参数含义。

9.2 实验要求:

能够理解栅格地图的概念

能够理解建图的原理

能够理解 gmapping 建图的一些参数的含义

使用 gmapping 实际建图

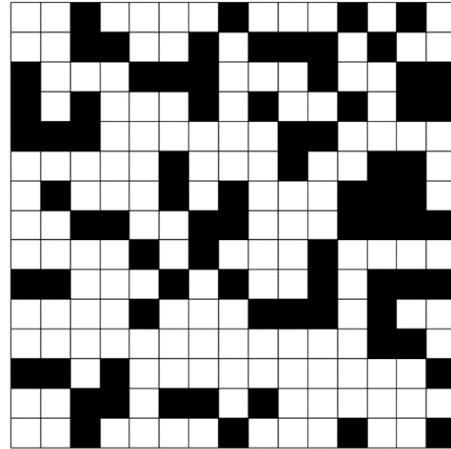
9.3 实验工具:

个人电脑一台，智行 mini 及其配件

9.4 实验内容:

9.4.1 栅格地图的概念

占据栅格地图是一种在 SLAM 地图中相较而言创建和维护更简单的地图表达形式，并且使用栅格地图更容易实现导航功能，因此占据栅格地图在目前二维 SLAM 中应用较为广泛。其基本原理是将机器人所在的空间划分为若干个大小相同且相互独立的栅格块，如图。而对于每一个栅格块，我们给予 $(0, 1)$ 之间的数来表示在这个栅格内有多大的概率存在障碍。很显然，对栅格的划分越细，其地图的精度自然也就越高。但同时栅格越多，计算机的计算量也就越大，这是栅格地图主要的缺点，也因此，栅格地图大多使用在空间面积较小的环境中。



栅格地图也是对 SLAM 建图支持度相当高的一种地图模式。

9.4.2 建图的原理

2D 激光 SLAM 的建图的输入一般是里程计数据、IMU 数据、2D 激光雷达数据，得到的输出为覆盖栅格地图以及机器人的运动轨迹。目前的算法主要由两个部分执行：一部分通过数据融合、特征处理等方式将提取得的传感器数据转化为可用的数据模型；另一部分则通过概率算法等手段来对这些模型进行处理，生成环境地图。

但一般来说这种直接根据传感器数据得出的地图会在某些情况下不可取。因为但凡传感器都是有误差的，而在建图的时候上一帧数据对下一帧的处理影响非常大，这就导致一旦出现误差就会在建图过程中一直累积下去。如果环境非常大，到最后就会导致所获得的的地图完全不可用。因此，回环检测是 SLAM 建图算法中非常重要的一部分。

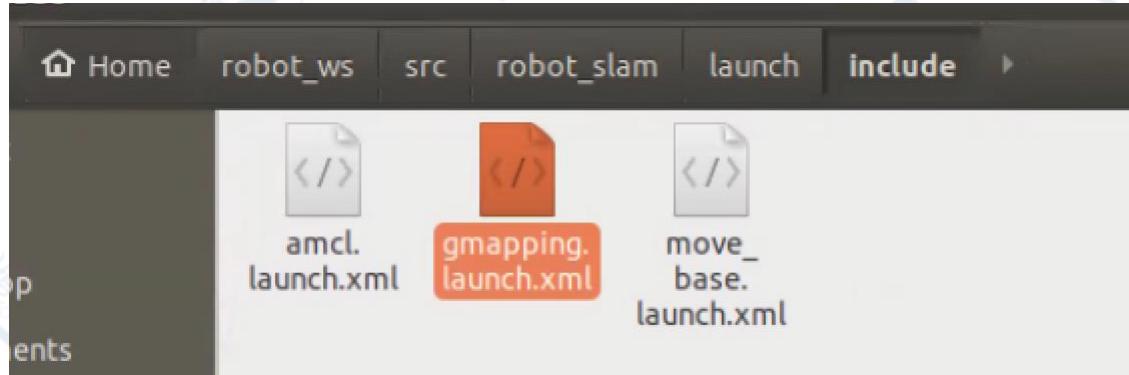
回环检测的基本原理就是将获得的一帧数据以及自己的位置信息与之前某一帧数据或是已计算出的地图进行匹配，找到能够成功匹配的帧，建立位姿约束关系，并以此约束关系为基准再去调整其他的数据，从而起到减小累积误差的效果。在目前的算法中，回环检测也不仅仅是只有帧间配对， scan-to-map 以及 map-to-map 才是用的比较多的方式。他们分别表示用当前帧数据去匹配之前某些帧建立的地图和用当前某些帧建立的地图去匹配之前建立的地图。相较而言 map-to-map 的匹配方式更加准确但计算量更大，同时匹配难度也更大。所以对于不同的应用场景来说，选择哪种算法应当根据当前场景的特点来确定。

智行 mini 上使用的 gmapping 包是基于滤波 SLAM 框架的常用开源算法，它基于 RBpf 粒子滤波算法，即将定位和建图分离，先进行定位再建图， Gmapping 在 RBpf 算法上做了两个主要的改进：改进提议分布和选择性重采样。但是它只适用于小环境下的建图定位，计算量小且精度较高。因为这种算法有效的利用了里程计的数据，所以并没有使用回环检测。在室内小环境下，即便相比目前开源算法中效果最好的 Cartographer 精度也没有下降太多，反而计算量由于没有回环检测而大大减小。不过随着场景增大，在没有

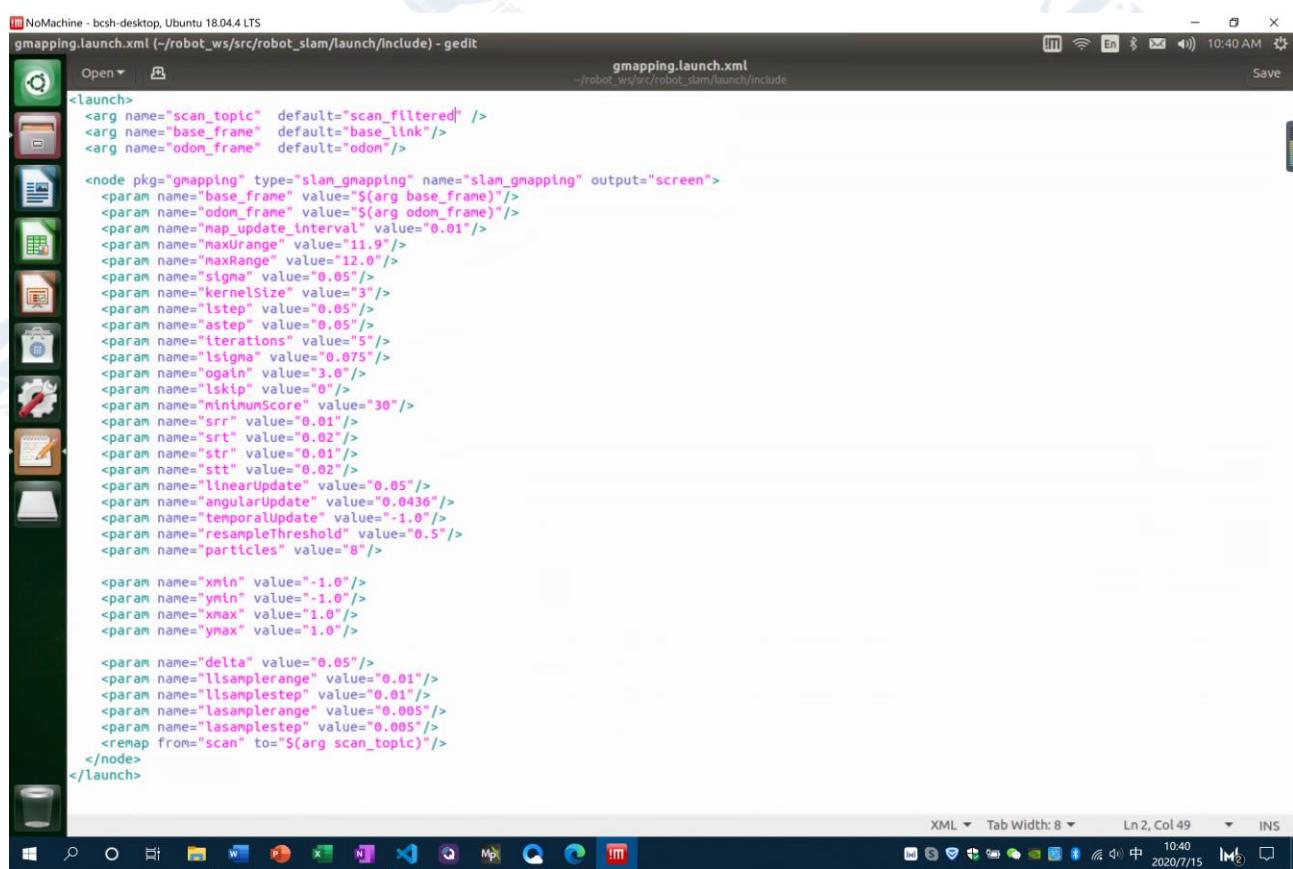
回环检测的情况下回环闭合时就可能会导致地图错位，故智行 mini 不适用于特别大的环境场景。

9.4.3 gmapping 建图的一些参数

在该位置打开文件：



得到：



```

<Launch>
<arg name="scan_topic" default="scan_filtered" />
<arg name="base_frame" default="base_link"/>
<arg name="odom_frame" default="odom"/>

<node pkg="gmapping" type="slam_gmapping" name="slam_gmapping" output="screen">
<param name="base_frame" value="$(arg base_frame)"/>
<param name="odom_frame" value="$(arg odom_frame)"/>
<param name="map_update_interval" value="0.01"/>
<param name="maxRange" value="11.9"/>
<param name="sigma" value="0.05"/>
<param name="kernelSize" value="3"/>
<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="gain" value="3.0"/>
<param name="lskip" value="0"/>
<param name="minimumScore" value="30"/>
<param name="srr" value="0.01"/>
<param name="srt" value="0.02"/>
<param name="str" value="0.01"/>
<param name="stt" value="0.02"/>
<param name="linearUpdate" value="0.05"/>
<param name="angularUpdate" value="0.0436"/>
<param name="temporalUpdate" value="-1.0"/>
<param name="resampleThreshold" value="0.5" />
<param name="particles" value="8" />

<param name="xmin" value="-1.0"/>
<param name="ymin" value="-1.0"/>
<param name="xmax" value="1.0"/>
<param name="ymax" value="1.0"/>

<param name="delta" value="0.05"/>
<param name="llssamplerange" value="0.01"/>
<param name="llsamplestep" value="0.01"/>
<param name="lasamplerange" value="0.005"/>
<param name="lasamplestep" value="0.005"/>
<remap from="scan" to="$(arg scan_topic)"/>
</node>
</Launch>

```

这些都是 gmapping 中的一些参数，下面将针对其中的部分参数作简单介绍：

map_update_interval: 是 gmapping 过程中每隔几秒更新一次地图，该值变小的话表明更新地图的频率加快，会增加消耗更多当前系统的 CPU 计算资源。同时地图更新也受 scanmatch 的影响，如果 scanmatch 没有成功的话，不会更新地图。

maxRange: 是雷达可用的最大有效测距值，maxRange 是雷达的理论最大测距值，一般情况下设置 TEL: 010-8211 4870 /4890/4887

maxUrang < 雷达的现实实际测距值 <= maxRange。

下面这些参数一般不用修改保持默认值即可：

sigma (float, default: 0.05), endpoint 匹配标准差

kernelSize (int, default: 1), 用于查找对应的 kernel size

lstep (float, default: 0.05), 平移优化步长

astep (float, default: 0.05), 旋转优化步长

iterations (int, default: 5), 扫描匹配迭代步数

lsigma (float, default: 0.075), 用于扫描匹配概率的激光标准差

ogain (float, default: 3.0), 似然估计为平滑重采样影响使用的 gain

lskip (int, default: 0), 每次扫描跳过的光束数.

minimumScore: 最小匹配得分, 这个参数很重要, 它决定了对激光的一个置信度, 越高说明对激光匹配算法的要求越高, 激光的匹配也越容易失败而转去使用里程计数据, 而设的太低又会使地图中出现大量噪声, 所以需要权衡调整。

srr,srt,str,stt 四个参数是运动模型的噪声参数, 一般设置为默认值不用修改。

linearUpdate: 机器人移动多远距离, 进行一次 scanmatch 匹配。

angularUpdate: 机器人旋转多少弧度, 进行一次 scanmatch 匹配。

temporalUpdate: 如果最新扫描处理比更新慢, 则处理 1 次扫描, 该值为负数时候关闭基于时间的更新, 该参数很重要, 需要重点关注。

resampleThreshold: 基于重采样门限的 Neff

particles:gmapping 算法中的粒子数, 因为 gmapping 使用的是粒子滤波算法, 粒子在不断地迭代更新, 所以选取一个合适的粒子数可以让算法在保证比较准确的同时有较高的速度。

xmin,ymin,xmax,ymax: 初始化的地图大小。

delta: 创建的地图分辨率, 默认是 0.05m。

llsamplerange: 线速度移动多长距离进行似然估计。

llsamplestep: 线速度移动多少步长进行似然估计。

lasamplerange: 每转动多少弧度用于似然估计。

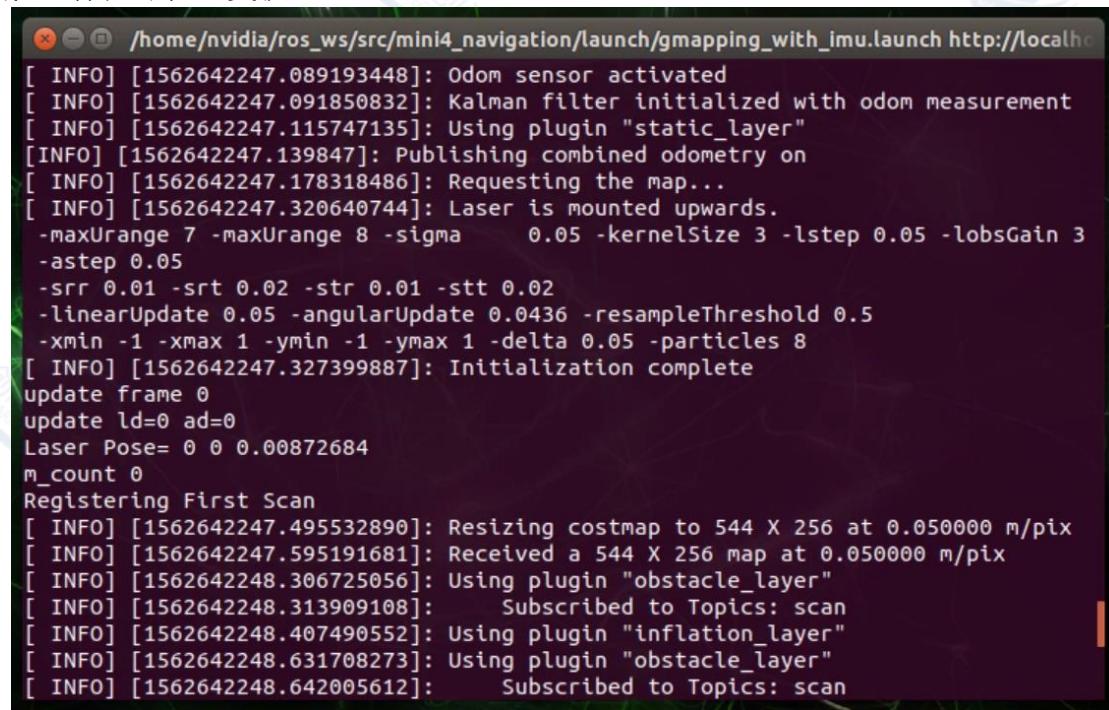
lasamplestep: 用于似然估计的弧度采样步长是多少。

transform_publish_period: 多长时间发布一次 tf 转换 (map->odom 转换), 单位是秒。

occ_thresh: 在 gmapping 过程中占有栅格地图的阈值, 只有当前单元格的占有率超过该值才认为是被占有。如果设置该值为 1 的话就会导致任何障碍物都不会认为是占有。

9.4.4 Gmapping 建图流程

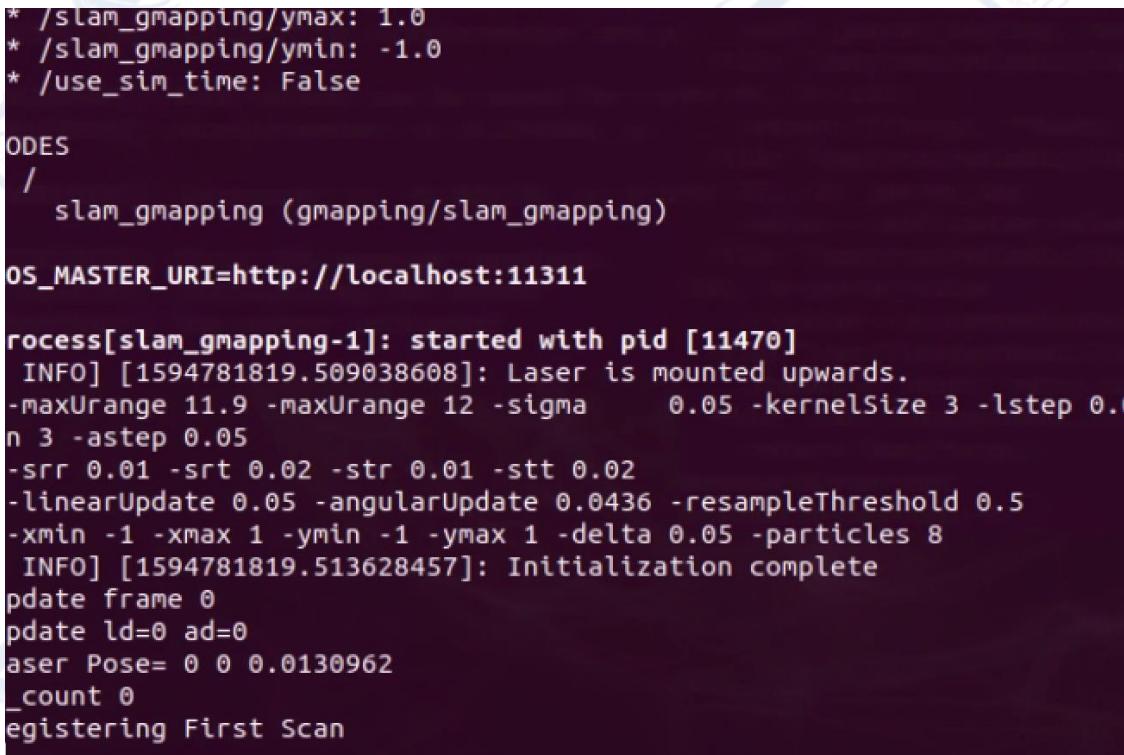
首先，打开终端输入：**roslaunch zoo_robot robot_lidar.launch**，连接底盘，连接激光雷达，发布底盘到雷达的TF变换。



```
/home/nvidia/ros_ws/src/minil4_navigation/launch/gmapping_with_imu.launch http://localhost:11311

[ INFO] [1562642247.089193448]: Odom sensor activated
[ INFO] [1562642247.091850832]: Kalman filter initialized with odom measurement
[ INFO] [1562642247.115747135]: Using plugin "static_layer"
[ INFO] [1562642247.139847]: Publishing combined odometry on
[ INFO] [1562642247.178318486]: Requesting the map...
[ INFO] [1562642247.320640744]: Laser is mounted upwards.
  -maxUrange 7 -maxUrange 8 -sigma      0.05 -kernelSize 3 -lstep 0.05 -lobsGain 3
  -astep 0.05
  -srr 0.01 -srt 0.02 -str 0.01 -stt 0.02
  -linearUpdate 0.05 -angularUpdate 0.0436 -resampleThreshold 0.5
  -xmin -1 -xmax 1 -ymin -1 -ymax 1 -delta 0.05 -particles 8
[ INFO] [1562642247.327399887]: Initialization complete
update frame 0
update ld=0 ad=0
Laser Pose= 0 0 0.00872684
m_count 0
Registering First Scan
[ INFO] [1562642247.495532890]: Resizing costmap to 544 X 256 at 0.050000 m/pix
[ INFO] [1562642247.595191681]: Received a 544 X 256 map at 0.050000 m/pix
[ INFO] [1562642248.306725056]: Using plugin "obstacle_layer"
[ INFO] [1562642248.313909108]: Subscribed to Topics: scan
[ INFO] [1562642248.407490552]: Using plugin "inflation_layer"
[ INFO] [1562642248.631708273]: Using plugin "obstacle_layer"
[ INFO] [1562642248.642005612]: Subscribed to Topics: scan
```

打开新终端输入：**roslaunch robot_slam gmapping.launch**



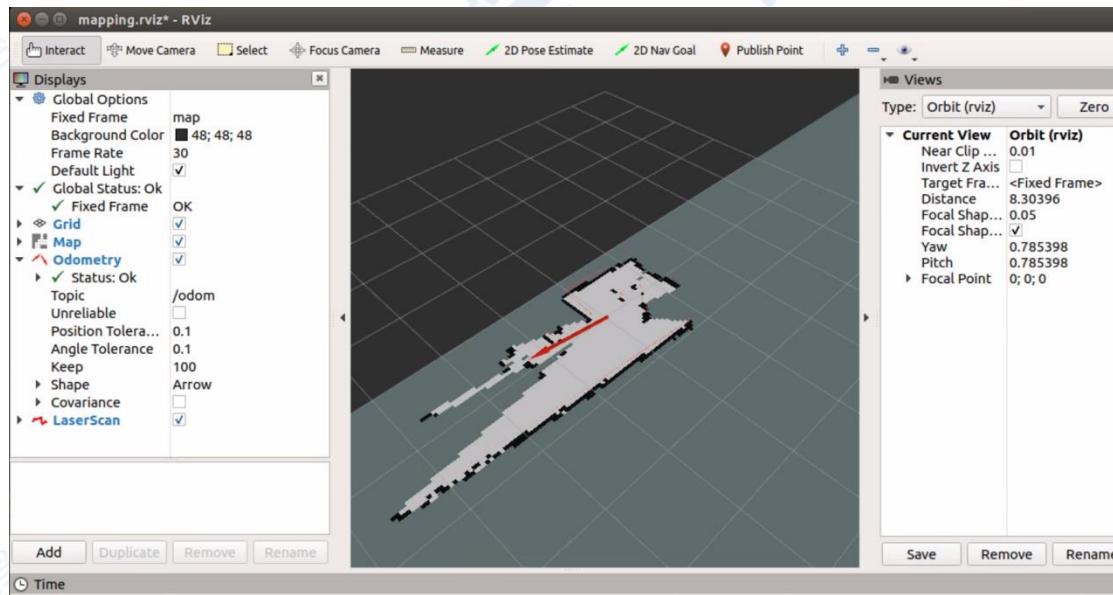
```
* /slam_gmapping/ymax: 1.0
* /slam_gmapping/ymin: -1.0
* /use_sim_time: False

ODES
/
  slam_gmapping (gmapping/slam_gmapping)

ROS_MASTER_URI=http://localhost:11311

process[slam_gmapping-1]: started with pid [11470]
[ INFO] [1594781819.509038608]: Laser is mounted upwards.
  -maxUrange 11.9 -maxUrange 12 -sigma      0.05 -kernelSize 3 -lstep 0.05
  -n 3 -astep 0.05
  -srr 0.01 -srt 0.02 -str 0.01 -stt 0.02
  -linearUpdate 0.05 -angularUpdate 0.0436 -resampleThreshold 0.5
  -xmin -1 -xmax 1 -ymin -1 -ymax 1 -delta 0.05 -particles 8
[ INFO] [1594781819.513628457]: Initialization complete
update frame 0
update ld=0 ad=0
Laser Pose= 0 0 0.0130962
m_count 0
registering First Scan
```

打开新终端输入：**roslaunch robot_slam view_mapping.launch**，在 Rviz 下查看建图效果

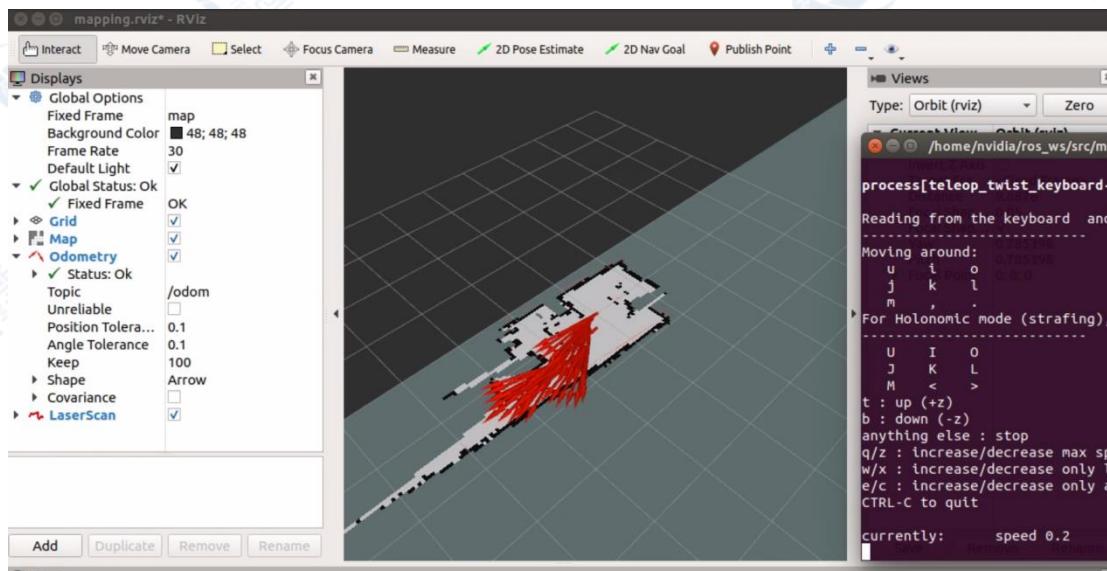


可以看到开始扫描出地图

Rviz 窗口含义如下：

| 地图颜色 | 代表意义 |
|------|----------------------|
| 红色 | 激光雷达探测到的障碍点（障碍物轮廓点阵） |
| 灰色 | 还没有探索到的未知区域 |
| 白色 | 已经探明的不存在静态障碍物的区域 |
| 黑色 | 静态障碍物轮廓 |

启动键盘或者手柄控制，控制机器人移动完成建图。



得到满意的地图后，在新终端内通过以下指令：**roslaunch robot_slam save_map.launch** 保存地图，地图位置如下：

9.5 实验结果：

使用 Gmapping 建立机器人所处环境的地图。

9.6 实验报告：

实验目的

实验要求

实验内容

实验总结

第十章.Navigation 自主导航（上）

10.1 实验目的：

理解 navigation 导航包的原理以及一些参数的使用方法与含义。

10.2 实验要求：

能够实现智行 mini 的导航功能

能够理解导航功能包的实现原理

能够掌握各参数的含义，以及其使用方法

10.3 实验工具：

个人电脑一台，智行 mini 及其配件

10.4 实验内容：

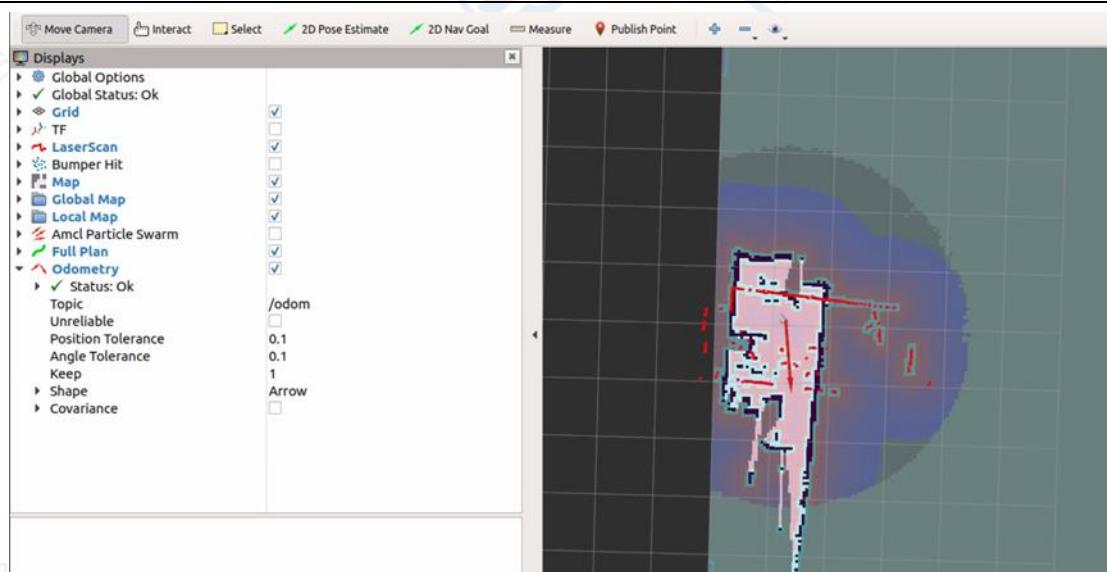
10.4.1 使用 navigation 包实现智行 mini 的导航功能

在上一节的最后我们刚刚得到了我们所创建的环境的地图，下面我们紧接着实验的流程，先来进行一下 navigation 导航包的实际使用。

首先打开终端，输入 **roslaunch zoo_robot robot_lidar.launch** 启动底盘，

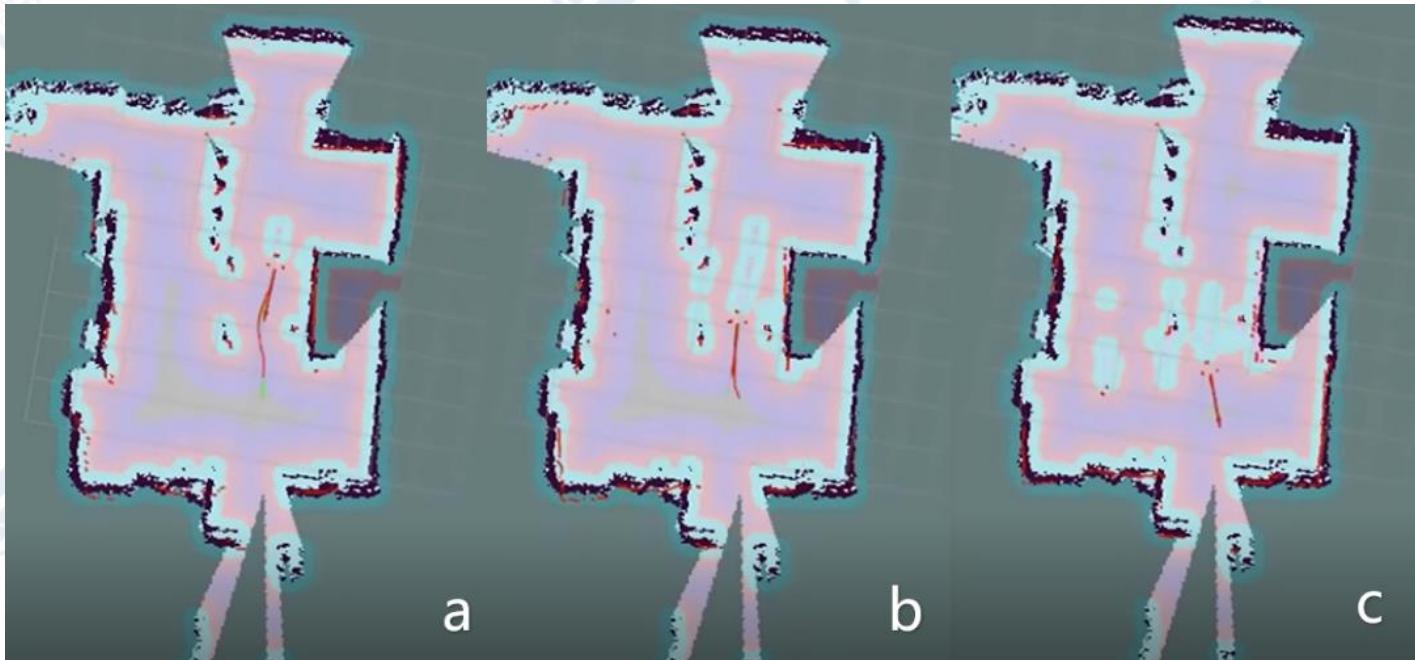
然后打开新的终端，输入 **roslaunch robot_slam navigation.launch** 运行导航功能，

然后打开新终端，输入 **roslaunch robot_slam view_nav.launch**，就可以在 rviz 下看到机器人在环境场景中的一些信息了，如图：



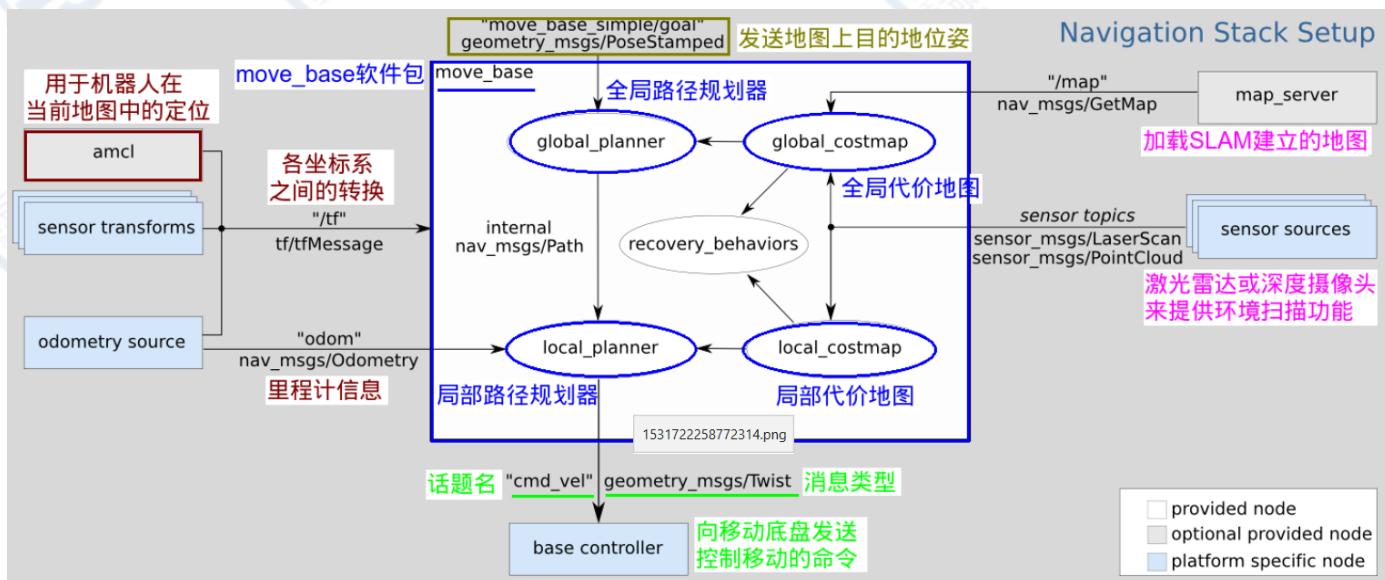
来仔细观察一下这个界面，上面我们会使用到的两个图标是两个绿色的箭头，分别是 2D Pose Estimate 和 2D Nav Goal，界面的右侧则是显示的信息，图中有红色箭头代表目前机器人朝向，红色点形成的框代表激光雷达识别到的障碍物，黑色点形成的框，这里是加载的上一节所保存下来的地图。这些含义在上一节也有所说明。那么在这张图中就能发现，红色的识别的障碍物框和黑色的加载的障碍物框并不重合，有很大的偏移，这就是我们在运行导航时要注意的一点，当启动导航时，默认的机器人的坐标是与使用 Gmapping 建图时的起点一致，所以这里我们有两种解决方法：第一种就是在我们建图的时候要记清楚开始点，然后在运行导航的时候将机器人摆回到初始点（包括位置和姿态）；第二种就是点击 2D Pose Estimate 那个绿色的箭头，然后点击地图中一点调整姿态再点击一次，让红色框和黑色框尽量重合即可。两种方法的目的都是一样的，都是确保所建立的地图的坐标系与导航的坐标系为同一坐标系。

完成调整初始点之后就可以点击 2D Nav Goal 那个箭头，然后在地图中点击想要让机器人去的地方并调整朝向，机器人就会自动规划路径，然后朝着目标点行驶了，在行驶过程中，机器人也会自动去避障，比如一个人突然出现在原先规划的路径上时，机器人也会重新规划局部路线，绕过这个人，展示效果如图：



10.4.2 Navigation 导航原理

导航功能的实现主要是依靠 navigation 功能包集来完成的，navigation 是 2D 的导航包集，它通过接收里程计数据、tf 坐标变换树以及传感器数据，为移动机器人输出目标位置以及安全速度。其导航逻辑如图所示



导航功能的核心是 move_base 节点，它接收来自里程计消息、机器人姿态位置、地图数据等信息，在节点内会进行全局规划以及局部规划。其中局部规划是为了能在导航过程中随时根据环境的改变来改变自己的路径，达到自动避障的效果。

导航功能的实现首先要有的三个因素就是地图、导航的起点、终点目标，并在导航过程中不断根据里程计、激光雷达等传感器数据来确定自己的位置。在 navigation 导航功能中，首先会根据代价地图规划处起点到终点的路线，然后结合里程计信息以及激光雷达的数据判断当前位置并规划处当前位置附近的局部路线以达到避障的效果。最终将局部规划的路线以速度指令的形式输出。

我们来分析一下 navigation 这个 launch 文件：

```
<launch>
  <param name="use_sim_time" value="false" />
  <arg name="map_name" default="my_lab.yaml" />
  <node name="map_server" pkg="map_server" type="map_server" args="$(find robot_slam)/maps/${arg map_name}"/>
  <include file="$(find robot_slam)/launch/include/move_base.launch.xml" />
  <include file="$(find robot_slam)/launch/include/amcl.launch.xml" />
</launch>
```

可以看到启动了三个核心节点

- **Map_server:** 加载地图到地图服务器
- **Amcl:** 基于地图和激光雷达，里程计实现概率定位
- **Move_base:** 导航与路径规划节点

10.4.3 amcl 自主定位

Amcl 是基于现有地图与激光雷达实现概率自定位的节点，用于确认并输出机器人在当前地图中的位置。

首先来介绍一下 amcl 的订阅与发布：

Amcl 订阅的话题：

| | |
|--------------|---|
| /scan | 激光雷达输出的话题，在进行 amcl 定位时，这个是必不可少需要订阅的话题，因为我们要知道目标周围的环境状态。 |
| /tf | 订阅了各坐标系转换的话题，用于查询各坐标系的转换 |
| /initialpose | 用于（重新）初始化粒子滤波器的平均值和协方差，简单来理解就是先预估计一下机器人的初始位姿 |
| /amcl/map | 当在 launch 文件中设置了 use_map_topic 为 true 时，amcl 则订阅该话题获取地图，然后使用基于激光来进行定位，当然设置 use_map_topic 为 false 时不订阅该话题也是可以的 |

- Amcl 发布的话题：

| | |
|---------------|--|
| amcl_pose | 机器人在地图上带有协方差的位姿估计，这个是话题是整个粒子滤波定位的最终输出结果，该话题输出的位姿信息是根据全局坐标系/map 的坐标转换后的位置 |
| particlecloud | 在粒子滤波器维护下的一组粒子位姿估 |

| | |
|----|---|
| | 计，可以直接在 rviz 中显示，查看粒子的收敛效果 |
| tf | 发布从 odom 坐标系到 map 坐标系的转换，当然该 odom 坐标系可以使用 odom_frame_id 参数来重新映射为自定义的坐标系名称 |

接下来我们来看看 amcl 自主定位有哪些可以调整的参数：

滤波器可以设定的参数：

min_particles (int, default: 100): 滤波器中的最少粒子数，值越大定位效果越好，但是相应的会增加主控平台的计算资源消耗。

~max_particles (int, default: 5000): 滤波器中最多粒子数，是一个上限值，因为太多的粒子数会导致系统资源消耗过多。

~kld_err (double, default: 0.01): 真实分布与估计分布之间的最大误差。

~kld_z (double, default: 0.99): 上标准分位数 (1-p)，其中 p 是估计分布上误差小于 kld_err 的概率，默认 0.99。

~update_min_d (double, default: 0.2 meters): 在执行滤波更新前平移运动的距离，默认 0.2m(对于里程计模型有影响，模型中根据运动和地图求最终位姿的似然时丢弃了路径中的相关所有信息，已知的只有最终位姿，为了规避不合理的穿过障碍物后的非零似然，这个值建议不大于机器人半径，否则因更新频率的不同可能产生完全不同的结果)。

~update_min_a (double, default: π/6.0 radians): 执行滤波更新前旋转的角度。

~resample_interval (int, default: 2): 在重采样前需要滤波更新的次数。

~transform_tolerance (double, default: 0.1 seconds): tf 变换发布推迟的时间，为了说明 tf 变换在未来时间内是可用的。

~recovery_alpha_slow (double, default: 0.0 (disabled)): 慢速的平均权重滤波的指数衰减频率，用作决定什么时候通过增加随机位姿来 recover，默认 0 (disable)，可能 0.001 是一个不错的值。

~recovery_alpha_fast (double, default: 0.0 (disabled)): 快速的平均权重滤波的指数衰减频率，用作决定什么时候通过增加随机位姿来 recover，默认 0 (disable)，可能 0.1 是个不错的值。

~initial_pose_x (double, default: 0.0 meters): 初始位姿均值 (x)，用于初始化高斯分布滤波器。
(initial_pose_参数决定撒出去的初始位姿粒子集范围中心)。

~initial_pose_y (double, default: 0.0 meters): 初始位姿均值 (y)，用于初始化高斯分布滤波器。
(同上)

~initial_pose_a (double, default: 0.0 radians): 初始位姿均值 (yaw)，用于初始化高斯分布滤波器。
(粒子朝向)

~initial_cov_xx (double, default: 0.5*0.5 meters): 初始位姿协方差 (x*x)，用于初始化高斯分布滤波器。
(initial_cov_参数决定初始粒子集的范围)

~initial_cov_yy (double, default: 0.5*0.5 meters): 初始位姿协方差 (y*y)，用于初始化高斯分布滤波器。
(同上)

~initial_cov_aa (double, default: (π/12)*(π/12) radian): 初始位姿协方差 (yaw*yaw)，用于初始化高斯分布滤波器。
(粒子朝向的偏差)

~gui_publish_rate (double, default: -1.0 Hz): 扫描和路径发布到可视化软件的最大频率，设置参数为 -1.0 意味着禁用此功能， 默认 -1.0。

~save_pose_rate (double, default: 0.5 Hz): 存储上一次估计的位姿和协方差到参数服务器的最大速率。被保存的位姿将被用在连续的运动上来初始化滤波器。-1.0 禁用。

~use_map_topic (bool, default: false): 当设置为 true 时，AMCL 将会订阅 map 话题，而不是调用服务返回地图。也就是说当设置为 true 时，有另外一个节点实时的发布 map 话题，也就是机器人在实时的

进行地图构建，并供给 amcl 话题使用；当设置为 false 时，通过 map server，也就是调用已经构建完成的地图。

~first_map_only (bool, default: false): 当设置为 true 时，AMCL 将仅仅使用订阅的第一个地图，而不是每次接收到新的时更新为一个新的地图。

可以设置的所有激光模型参数：

laser_min_range (double, default: -1.0): 最小扫描范围，参数设置为-1.0 时，将会使用激光上报的最小扫描范围。

~laser_max_range (double, default: -1.0): 最大扫描范围，参数设置为-1.0 时，将会使用激光上报的最大扫描范围。

~laser_max_beams (int, default: 30): 更新滤波器时，每次扫描中多少个等间距的光束被使用（减小计算量，测距扫描中相邻波束往往不是独立的可以减小噪声影响，太小也会造成信息量少定位不准）。

~laser_z_hit (double, default: 0.95): 模型的 z_hit 部分的混合权值，默认 0.95(混合权重 1.具有局部测量噪声的正确范围--以测量距离近似真实距离为均值，其后 laser_sigma_hit 为标准偏差的高斯分布的权重)。

~laser_z_short (double, default: 0.1): 模型的 z_short 部分的混合权值，默认 0.1 (混合权重 2.意外对象权重 (类似于一元指数关于 y 轴对称 0~测量距离 (非最大距离) 的部分: $-\eta\lambda e^{(-\lambda z)}$), 其余部分为 0, 其中 η 为归一化参数, λ 为 laser_lambda_short,z 为 t 时刻的一个独立测量值 (一个测距值, 测距传感器一次测量通常产生一系列的测量值)), 动态的环境, 如人或移动物体)。

~laser_z_max (double, default: 0.05): 模型的 z_max 部分的混合权值，默认 0.05 (混合权重 3.测量失败权重 (最大距离时为 1, 其余为 0) , 如声呐镜面反射, 激光黑色吸光对象或强光下的测量, 最典型的是超出最大距离)。

~laser_z_rand (double, default: 0.05): 模型的 z_rand 部分的混合权值，默认 0.05 (混合权重 4.随机测量权重--均匀分布 (1 平均分布到 0~最大测量范围) , 完全无法解释的测量, 如声呐的多次反射, 传感器串扰)。

~laser_sigma_hit (double, default: 0.2 meters): 被用在模型的 z_hit 部分的高斯模型的标准差，默认 0.2m。

~laser_lambda_short (double, default: 0.1): 模型 z_short 部分的指数衰减参数，默认 0.1 (根据 $\eta\lambda e^{(-\lambda z)}$, λ 越大随距离增大意外对象概率衰减越快)。

~laser_likelihood_max_dist (double, default: 2.0 meters): 地图上做障碍物膨胀的最大距离，用作 likelihood_field 模型 (likelihood_field_range_finder_model 只描述了最近障碍物的距离，(目前理解应该是在这个距离内的障碍物膨胀处理,但是算法里又没有提到膨胀, 不明是什么意思).这里算法用到上面的 laser_sigma_hit。似然域计算测量概率的算法是将 t 时刻的各个测量 (舍去达到最大测量范围的测量值) 的概率相乘, 单个测量概率: $Z_h * prob(dist,\sigma) + avg$, Z_h 为 laser_z_hit,avg 为均匀分布概率, dist 最近障碍物的距离, prob 为 0 为中心标准方差为 σ (laser_sigma_hit) 的高斯分布的距离概率)。

~laser_model_type (string, default: "likelihood_field"): 激光模型类型定义，可以是 beam, likelihood_field, likelihood_field_prob (和 likelihood_field 一样但是融合了 beamskip 特征--官网的注释)，默认是“likelihood_field”。

可以设置的里程计模型参数：

~odom_model_type (string, default: "diff"): odom 模型定义，可以是"diff", "omni", "diff-corrected", "omni-corrected"，后面两个是对老版本里程计模型的矫正，相应的里程计参数需要做一定的减小。

~odom_alpha1 (double, default: 0.2): 指定由机器人运动部分的旋转分量估计的里程计旋转的期望噪声，默认 0.2 (旋转存在旋转噪声)。

~odom_alpha2 (double, default: 0.2): 机器人运动部分的平移分量估计的里程计旋转的期望噪声，默认 0.2 (旋转中可能出现平移噪声)。

~odom_alpha3 (double, default: 0.2): 机器人运动部分的平移分量估计的里程计平移的期望噪声，如

果你自认为自己机器人的里程计信息比较准确那么就可以将该值设置的很小。

~odom_alpha4 (double, default: 0.2): 机器人运动部分的旋转分量估计的里程计平移的期望噪声，你设置的这 4 个 alpha 值越大说明里程计的误差越大。

~odom_alpha5 (double, default: 0.2): 平移相关的噪声参数（仅用于模型是“omni”的情况，就是当你的机器人是全向移动时才需要设置该参数，否则就设置其为 0.0）

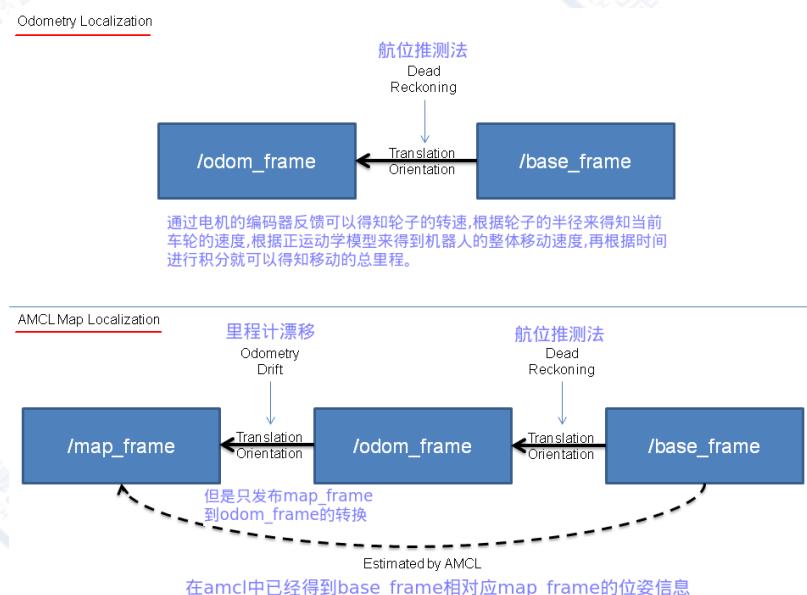
~odom_frame_id (string, default: "odom"): 里程计默认使用的坐标系。

~base_frame_id (string, default: "base_link"): 机器人的基坐标系。

~global_frame_id (string, default: "map"): 由定位系统发布的坐标系名称。

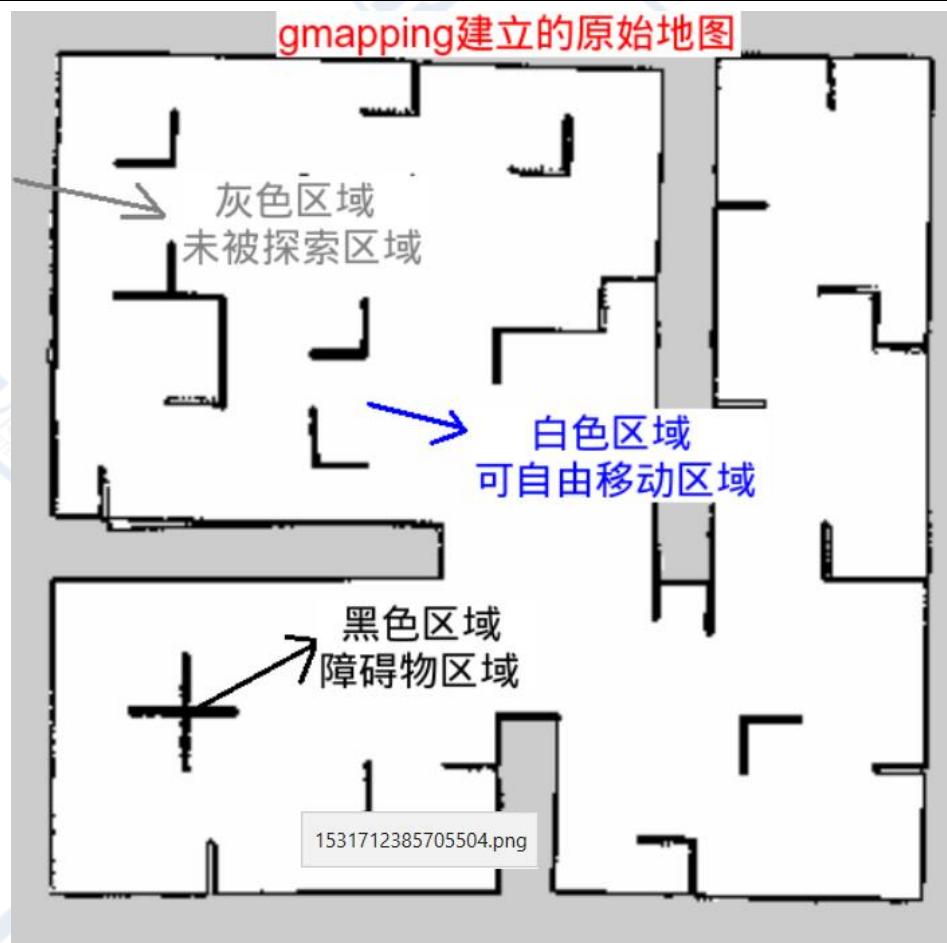
~tf_broadcast (bool, default: true): 设置为 false 阻止 amcl 发布全局坐标系和里程计坐标系之间的 tf 变换。

另外必须要注意的是，amcl 发布的 tf 变换是 map 到 odom 的变换，用于修正轮子打滑等问题导致的 odom 不准确。



10.4.4 move_base 功能使用

在机器人进行路径规划时，我们需要明白规划算法是依靠什么在地图上来计算出来一条路径的。依靠的是 gmapping 扫描构建的一张环境全局地图，但是仅仅依靠一张原始的全局地图是不行的。因为这张地图是静态的，无法随时来更新地图上的障碍物信息。在现实环境中，总会有各种无法预料到的新障碍物出现在当前地图中，或者旧的障碍物现在已经从环境地图中被移除掉了，那么我们就需要来随时更新这张地图。同时由于默认的地图是一张黑白灰三色地图，即只会标出障碍物区域、自由移动区域和未被探索区域。机器人在这样的地图中进行路径规划，会导致规划的路径不够安全，因为我们的机器人在移动时需要与障碍物之间保持一定的安全缓冲距离，这样机器人在当前地图中移动时就更安全了。



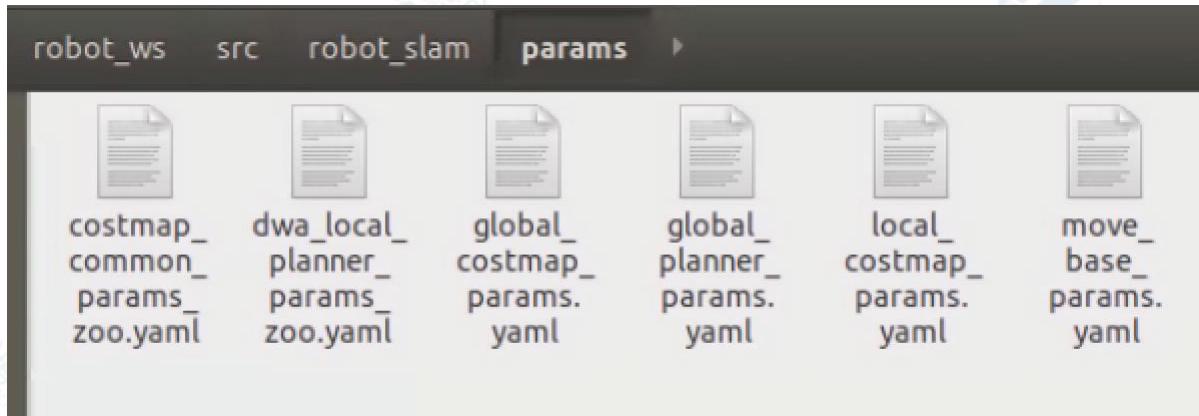
costmap 简单来说就是为了在这张地图上进行各种加工，方便我们后面进行路径规划而存在的，通过使用 costmap_2d 这个软件包来实现的，该软件包在原始地图上实现了两张新的地图。一个是 local_costmap，另外一个就是 global_costmap，根据名字大家就可以知道了，两张 costmap 一个是为了局部路径规划准备的，一个是为全局路径规划准备的。无论是 local_costmap 还是 global_costmap，都可以配置多个图层，包括下面几种：

- Static Map Layer: 静态地图层，基本上不变的地图层，通常都是 SLAM 建立完成的静态地图。
- Obstacle Map Layer: 障碍地图层，用于动态的记录传感器感知到的障碍物信息。
- Inflation Layer: 膨胀层，在以上两层地图上进行膨胀（向外扩张），以避免机器人的撞上障碍物。
- Other Layers: 你还可以通过插件的形式自己实现 costmap，目前已有 Social Costmap Layer、Range Sensor Layer 等开源插件。

在启动 move_base 节点时，首先会加载 costmap_common_params.yaml 到 global_costmap 和 local_costmap 两个命名空间中，因为该配置文件是一个通用的代价地图配置参数，即 local_costmap 和

global_costmap 都需要配置的参数。然后 local_costmap_params.yaml 是专门为了局部代价地图配置的参数，global_costmap_params.yaml 是专门为全局代价地图配置的参数。

我们可以进入一下文件夹来查看有关 costmap 的相关参数：



进入该文件夹下可以发现 costmap_common_params.yaml、global_costmap_params.yaml、local_costmap_params.yaml 文件都在此处，打开 costmap_common_params.yaml：、costmap_common_params.yaml 参数含义如下：

```
max_obstacle_height: 0.60 # assume something like an arm is mounted on top of the robot

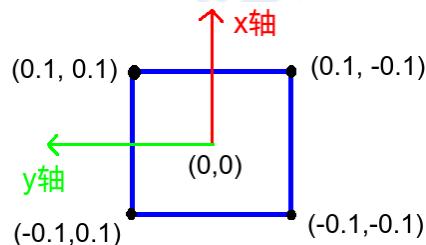
# Obstacle Cost Shaping (http://wiki.ros.org/costmap\_2d/hydro/inflation)
#robot_radius: 0.16 # distance a circular robot should be clear of the obstacle (kobuki: 0.18)
footprint: [[0.16, 0.16], [0.16, -0.16], [-0.16, -0.16], [-0.16, 0.16]]
# footprint: [[x0, y0], [x1, y1], ... [xn, yn]] # if the robot is not circular
|
map_type: voxel

obstacle_layer:
  enabled: true
  max_obstacle_height: 0.6
  origin_z: 0.0
  z_resolution: 0.2
  z_voxels: 2
  unknown_threshold: 15
  mark_threshold: 0
  combination_method: 1
  track_unknown_space: true #true needed for disabling global path planning through unknown space
  obstacle_range: 2.5
  raytrace_range: 3.0
  origin_z: 0.0
  z_resolution: 0.2
  z_voxels: 2
  publish voxel map: false
```

robot_radius: 设置机器人的半径，单位是米。如果机器人是圆形的，就可以直接设置该参数。

如果机器人不是圆形的那就需要使用 footprint 这个参数，该参数是一个列表，其中的每一个坐标代表机器人上的一点，设置机器人的中心为[0, 0]，根据机器人不同的形状，找到机器人各凸出的坐标点即可，具体可参考下图来设置：

正方形的机器人



footprint参数设置

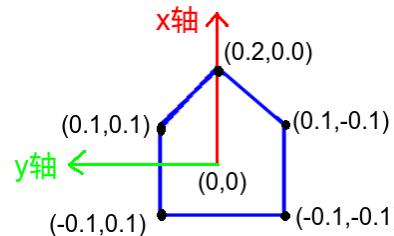
顺时针

footprint: [[0.1,0.1],[0.1,-0.1],[-0.1,-0.1],[-0.1,0.1]]

逆时针

footprint: [[0.1,0.1],[-0.1,0.1],[-0.1,-0.1],[0.1,-0.1]]

五边形机器人



顺时针或逆时针设置参数都是可以的,这里使用顺时针来设置参数 :

footprint: [[0.1,0.1],[0.2,0.0],[0.1,-0.1],[-0.1,-0.1],[-0.1,0.1]]

obstacle_layer:配置障碍物图层

enabled:是否启用该层

combination_method:只能设置为 0 或 1, 用来更新地图上的代价值, 一般设置为 1;

track_unknown_space:如果设置为 false, 那么地图上代价值就只分为致命碰撞和自由区域两种, 如果设置为 true, 那么就分为致命碰撞, 自由区域和未知区域三种。意思是说假如该参数设置为 true 的话, 就意味着地图上的未知区域也会被认为是可以自由移动的区域, 这样在进行全局路径规划时, 可以把一些未探索的未知区域也参与到路径规划, 如果你需要这样的话就将该参数设置为 false。不过一般情况未探索的区域不应该当作可以自由移动的区域, 因此一般将该参数设置为 true;

obstacle_range:设置机器人检测障碍物的最大范围, 意思是说超过该范围的障碍物, 并不进行检测, 只有靠近到该范围内才把该障碍物当作影响路径规划和移动的障碍物;

raytrace_range:在机器人移动过程中, 实时清除代价地图上的障碍物的最大范围, 更新可自由移动的空间数据。假如设置该值为 3 米, 那么就意味着在 3 米内的障碍物, 本来开始时是有的, 但是本次检测却没有了, 那么就需要在代价地图上来更新, 将旧障碍物的空间标记为可以自由移动的空间。

```

observation_sources: scan
scan:
  data_type: LaserScan
  topic: scan
  inf_is_valid: true
  marking: true
  clearing: true
  min_obstacle_height: 0.05
  max_obstacle_height: 0.35

```

observation_sources:设置导航中所使用的传感器，这里可以用逗号形式来区分开很多个传感器，例如激光雷达，碰撞传感器，超声波传感器等，我们这里只设置了激光雷达；

laser_scan_sensor:添加的激光雷达传感器

sensor_frame:激光雷达传感器的坐标系名称；

data_type:激光雷达数据类型；

topic:该激光雷达发布的话题名；

marking:是否可以使用该传感器来标记障碍物；

clearing:是否可以使用该传感器来清除障碍物标记为自由空间；

```

inflation_layer:
  enabled: true
  cost_scaling_factor: 2.5 # exponential rate at which the obstacle cost drops off (default: 10)
  inflation_radius: 1.2 # max. distance from an obstacle at which costs are incurred for planning paths.

static_layer:
  enabled: true

```

inflation_layer:膨胀层，用于在障碍物外标记一层危险区域，在路径规划时需要避开该危险区域

enabled:是否启用该层；

cost_scaling_factor:膨胀过程中应用到代价值的比例因子，代价地图中到实际障碍物距离在内切圆半径到膨胀半径之间的所有 cell 可以使用如下公式来计算膨胀代价：

$$\exp(-1.0 * \text{cost_scaling_factor} * (\text{distance_from_obstacle} - \text{inscribed_radius})) * \\ (\text{costmap_2d}::\text{INSCRIBED_INFLATED_OBSTACLE} - 1)$$

公式中 `costmap_2d::INSCRIBED_INFLATED_OBSTACLE` 目前指定为 254，注意：由于在公式中 `cost_scaling_factor` 被乘了一个负数，所以增大比例因子反而会降低代价

inflation_radius:膨胀半径，膨胀层会把障碍物代价膨胀直到该半径为止，一般将该值设置为机器人底盘的直径大小。如果机器人经常撞到障碍物就需要增大该值，若经常无法通过狭窄地方就减小该值。

Static_layer:静态地图层，即 SLAM 中构建的地图层

enabled:是否启用该地图层；

打开 `global_costmap_params.yaml` 文件：

`global_costmap_params.yaml` 参数含义如下：

```
global_costmap:  
  global_frame: map  
  robot_base_frame: base_link  
  update_frequency: 1.0  
  publish_frequency: 0.5  
  static_map: true  
  resolution: 0.05  
  transform_tolerance: 0.5  
  plugins:  
    - {name: static_layer,  
      type: "costmap_2d::StaticLayer"}  
    - {name: obstacle_layer,  
      type: "costmap_2d::VoxelLayer"}  
    - {name: inflation_layer,  
      type: "costmap_2d::InflationLayer"} >
```

global_frame:全局代价地图需要在哪个坐标系下运行；

robot_base_frame:在全局代价地图中机器人本体的基坐标系，就是机器人上的根坐标系。通过 `global_frame` 和 `robot_base_frame` 就可以计算两个坐标系之间的变换，得知机器人在全局坐标系中的坐标了。

update_frequency:全局代价地图更新频率，一般全局代价地图更新频率设置的比较小；

static_map:配置是否使用 `map_server` 提供的地图来初始化，一般全局地图都是静态的，需要设置为 `true`；

rolling_window:是否在机器人移动过程中需要滚动窗口，始终保持机器人在当前窗口中心位置；

transform_tolerance:坐标系间的转换可以忍受的最大延时；

plugins:在 `global_costmap` 中使用下面三个插件来融合三个不同图层，分别是 `static_layer`、`obstacle_layer` 和 `inflation_layer`，合成一个 `master_layer` 来进行全局路径规划。

打开 `local_costmap_params.yaml` 文件：`local_costmap_params.yaml` 参数含义如下：

```
local_costmap:  
  global_frame: odom  
  robot_base_frame: base_link  
  update_frequency: 1.0  
  publish_frequency: 2.0  
  static_map: false  
  rolling_window: true  
  width: 3.5  
  height: 3.5  
  resolution: 0.05  
  transform_tolerance: 0.5  
  plugins:  
    - {name: obstacle_layer, type: "costmap_2d::VoxelLayer"}  
    - {name: inflation_layer, type: "costmap_2d::InflationLayer"}]
```

global_frame:在局部代价地图中的全局坐标系，一般需要设置为 `odom_frame`，但是由于 stdr 没有这个坐标系，我就拿/`map_static` 来代替了；

robot_base_frame:机器人本体的基坐标系；

update_frequency:局部代价地图的更新频率；

publish_frequency:局部代价地图的发布频率；

static_map:局部代价地图一般不设置为静态地图，因为需要检测是否在机器人附近有新增的动态障碍物；

rolling_window:使用滚动窗口，始终保持机器人在当前局部地图的中心位置；

width:滚动窗口的宽度，单位是米；

height:滚动窗口的高度，单位是米；

resolution:地图的分辨率，该分辨率可以从加载的地图相对应的配置文件中获取到；

transform_tolerance:局部代价地图中的坐标系之间转换的最大可忍受延时；

plugins:在局部代价地图中，不需要静态地图层，因为我们使用滚动窗口来不断的扫描障碍物，所以就需要融合两层地图 (`inflation_layer` 和 `obstacle_layer`) 即可，融合后的地图用于进行局部路径规划；

`move_base` 是导航功能的核心，在导航过程中会根据起点、目标点以及地图信息规划处全局路线，但有了全局路线还不够，为了避免与一些移动的障碍物发生碰撞比如行人，还需要对所处的一个局部环境进行局部路径规划，规划的路径要尽可能的服从全局路径，只是为了暂时性的避开障碍物。若配置路径规划的参数就要配置 `move_base` 相关的参数，在 `move_base` 中有多种路径规划器算法可选，我们需要告诉

move_base 路径规划器使用哪种算法。一般来说，全局路径的规划插件包括：

navfn:ROS 中比较旧的代码实现了 dijkstra 和 A*全局规划算法。

global_planner:重新实现了 Dijkstra 和 A*全局规划算法，可以看作 navfn 的改进版。

parrot_planner:一种简单的算法实现全局路径规划算法。

局部路径的规划插件包括：

base_local_planner:实现了 Trajectory Rollout 和 DWA 两种局部规划算法。

dwa_local_planner:实现了 DWA 局部规划算法，可以看作是 base_local_planner 的改进版本。

move_base_params.yaml 各参数的意义如下：

```
shutdown_costmaps: false
controller_frequency: 5.0
controller_patience: 3.0

planner_frequency: 1.0
planner_patience: 5.0

oscillation_timeout: 10.0
oscillation_distance: 0.2

# local planner - default is trajectory rollout
base_local_planner: "dwa_local_planner/DWAPlannerROS"

base_global_planner: global_planner/GlobalPlanner #navfn/NavfnROS #alternatives: , carrot_planner/CarrotPlanner
```

shutdown_costmaps:当 move_base 在不活动状态时, 是否关掉 costmap.

controller_frequency:向底盘控制移动话题 cmd_vel 发送命令的频率.

controller_patience:在空间清理操作执行前, 控制器花多长时间等有效控制下发.

planner_frequency:全局规划操作的执行频率. 如果设置为 0.0, 则全局规划器仅在接收到新的目标点或者局部规划器报告路径堵塞时才会重新执行规划操作.

planner_patience:在空间清理操作执行前, 留给规划器多长时间来找出一条有效规划.

oscillation_timeout:执行修复机制前, 允许振荡的时长.

oscillation_distance:来回运动在多大距离以上不会被认为是振荡.

base_local_planner:指定用于 move_base 的局部规划器名称.

base_global_planner:指定用于 move_base 的全局规划器插件名称.

global_planner_params.yaml 文件用于规划全局路径, 各参数意义如下:

```
GlobalPlanner:  
    old_navfn_behavior: false  
    use_quadratic: true  
    use_dijkstra: true  
    use_grid_path: false  
  
    allow_unknown: true  
  
    planner_window_x: 0.0  
    planner_window_y: 0.0  
    default_tolerance: 0.5  
  
    publish_scale: 100  
    planner_costmap_publish_frequency: 0.0  
  
    lethal_cost: 253  
    neutral_cost: 66  
    cost_factor: 0.55  
    publish_potential: true
```

old_navfn_behavior: 若在某些情况下, 想让 global_planner 完全复制 navfn 的功能, 那就设置为 true。

use_dijkstra: 设置为 true, 将使用 dijkstra 算法, 否则使用 A*算法.

use_quadratic: 设置为 true, 将使用二次函数近似函数, 否则使用更加简单的计算方式, 这样节省硬件计算资源.

use_grid_path: 如果设置为 true, 则会规划一条沿着网格边界的路径, 倾向于直线穿越网格, 否则将使用梯度下降算法, 路径更为光滑点.

allow_unknown: 是否允许规划器规划穿过未知区域的路径, 只设计该参数为 true 还不行, 还要在 costmap_commons_params.yaml 中设置 track_unknown_space 参数也为 true 才行。

default_tolerance: 当设置的目的地被障碍物占据时, 需要以该参数为半径寻找到最近的点作为新目的地.

visualize_potential: 是否显示从 PointCloud2 计算得到的势区域.

lethal_cost: 致命代价值, 默认是设置为 253, 可以动态来配置该参数.

neutral_cost: 中等代价值, 默认设置是 50, 可以动态配置该参数.

cost_factor: 代价地图与每个代价值相乘的因子.

publish_potential: 是否发布 costmap 的势函数.

局部路径规划参数相当重要, 因为它是直接控制机器人的移动底盘运动的插件, 它负责来向移动底盘的/cmd_vel 话题中发布控制命令。机器人移动的效果好不好, 这个局部路径规划可是影响最大的。

在这里我们使用 dwa_local_planner, 它是一个能够驱动底盘移动的控制器, 该控制器连接了路径规划器和机器人. 使用地图, 规划器产生从起点到目标点的运动轨迹, 在移动时, 规划器在机器人周围产生一个函数, 用网格地图表示。控制器的工作就是利用这个函数来确定发送给机器人的速度 dx, dy, dtheta

DWA 算法的基本思想

1. 在机器人控制空间离散采样(dx, dy, dtheta)
2. 对每一个采样的速度进行前向模拟, 看看在当前状态下, 使用该采样速度移动一小段时间后会发生什么.
3. 评价前向模拟得到的每个轨迹, 是否接近障碍物, 是否接近目标, 是否接近全局路径以及速度等等. 舍弃非法路径
4. 选择得分最高的路径, 发送对应的速度给底座

dwa_local_planner_params.yaml 文件各参数意义:

```
# Robot Configuration Parameters - Kobuki
max_vel_x: 0.25
min_vel_x: -0.25

max_vel_y: 0
min_vel_y: 0

max_trans_vel: 0.35 # choose slightly less than the base's capability
min_trans_vel: 0.001 # this is the min trans velocity when there is negligible rotational velocity
trans_stopped_vel: 0.05

# Warning!
# do not set min_trans_vel to 0.0 otherwise dwa will always think translational velocities
# are non-negligible and small in place rotational velocities will be created.

max_rot_vel: 1.4 # choose slightly less than the base's capability
min_rot_vel: 0.8 # this is the min angular velocity when there is negligible translational velocity
rot_stopped_vel: 0.1

acc_lim_x: 1 # maximum is theoretically 2.0, but we
acc_lim_theta: 1.5
acc_lim_y: 0      # diff drive robot
```

max_vel_x: x 方向最大速度的绝对值

min_vel_x: x 方向最小值绝对值, 如果为负值表示可以后退.

max_vel_y:y 方向最大速度的绝对值.

min_vel_y:y 方向最小速度的绝对值.

max_trans_vel:平移速度最大值绝对值

min_trans_vel:平移速度最小值的绝对值

max_rot_vel:最大旋转速度的绝对值.

min_rot_vel:最小旋转速度的绝对值.

acc_lim_x:x 方向的加速度绝对值

acc_lim_y:y 方向的加速度绝对值, 该值只有全向移动的机器人才需配置.

acc_lim_th:旋转加速度的绝对值.

```
# Goal Tolerance Parameters
yaw_goal_tolerance: 0.2
xy_goal_tolerance: 0.15
# latch_xy_goal_tolerance: false

# Forward Simulation Parameters
sim_time: 2.0      # 1.7
vx_samples: 10      # 3
vy_samples: 1        # diff drive robot, there is only one sample
vtheta_samples: 20    # 20
```

yaw_goal_tolerance:到达目标点时偏航角允许的误差, 单位弧度.

xy_goal_tolerance:到达目标点时, 在 xy 平面内与目标点的距离误差.

latch_xy_goal_tolerance:设置为 true, 如果到达容错距离内, 机器人就会原地旋转, 即使转动是会跑出容错距离外.

sim_time:向前仿真轨迹的时间.

sim_granularity:步长, 轨迹上采样点之间的距离, 轨迹上点的密集程度.

vx_samples:x 方向速度空间的采样点数.

vy_samples:y 方向速度空间采样点数.

vtheta_samples:旋转方向的速度空间采样点数.

controller_frequency:发送给底盘控制移动指令的频率.

```
# Trajectory Scoring Parameters
path_distance_bias: 32.0      # 32.0      - weighting for how much it should stick to the global path plan
goal_distance_bias: 24.0      # 24.0      - weighting for how much it should attempt to reach its goal
occdist_scale: 0.4            # 0.01     - weighting for how much the controller should avoid obstacles
forward_point_distance: 0.325 # 0.325    - how far along to place an additional scoring point
stop_time_buffer: 0.2          # 0.2      - amount of time a robot must stop in before colliding for a valid traj.
scaling_speed: 0.25           # 0.25     - absolute velocity at which to start scaling the robot's footprint
max_scaling_factor: 0.2        # 0.2      - how much to scale the robot's footprint when at speed.

# Oscillation Prevention Parameters
oscillation_reset_dist: 0.05 # 0.05     - how far to travel before resetting oscillation flags

# Debugging
publish_traj_pc : true
publish_cost_grid_pc: true
global_frame_id: odom
```

path_distance_bias:定义控制器与给定路径接近程度的权重.

goal_distance_bias:定义控制器与局部目标点的接近程度的权重.

occdist_scale:定义控制器躲避障碍物的程度.

stop_time_buffer:为防止碰撞, 机器人必须提前停止的时间长度.

scaling_speed:启动机器人底盘的速度.

max_scaling_factor:最大缩放参数.

publish_cost_grid:是否发布规划器在规划路径时的代价网格. 如果设置为 true, 那么就会在 ~/cost_cloud 话题上发布 sensor_msgs/PointCloud2 类型消息.

oscillation_reset_dist:机器人运动多远距离才会重置振荡标记.

prune_plan:机器人前进是否清除身后 1m 外的轨迹.

在按照之前所讲述的运行导航功能的实验中, 输入 rostopic list 可以查看当前存在的话题, 可以看到有很多 planner 和 costmap:

```
/move_base/DWAPlannerROS/cost_cloud
/move_base/DWAPlannerROS/global_plan
/move_base/DWAPlannerROS/local_plan
/move_base/DWAPlannerROS/parameter_descriptions
/move_base/DWAPlannerROS/parameter_updates
/move_base/DWAPlannerROS/trajectory_cloud
/move_base/GlobalPlanner/parameter_descriptions
/move_base/GlobalPlanner/parameter_updates
/move_base/GlobalPlanner/plan
/move_base/GlobalPlanner/potential
/move_base/cancel
/move_base/current_goal
/move_base/feedback
/move_base/global_costmap/costmap
/move_base/global_costmap/costmap_updates
/move_base/global_costmap/footprint
/move_base/global_costmap/inflation_layer/parameter_descriptions
/move_base/global_costmap/inflation_layer/parameter_updates
/move_base/global_costmap/obstacle_layer/clearing_endpoints
/move_base/global_costmap/obstacle_layer/parameter_descriptions
/move_base/global_costmap/obstacle_layer/parameter_updates
/move_base/global_costmap/parameter_descriptions
/move_base/global_costmap/parameter_updates
```

10.5 实验结果：

能够让智行 mini 自主导航到指定位置，理解 costmap，planner 中的各种参数含义

10.6 实验报告：

实验目的

实验要求

实验内容

实验总结

第十一章.Navigation 自主导航（下）

11.1 实验目的：

实现通过代码自主发布航点，实现自主导航

11.2 实验要求：

熟悉 action 原理，发布代码与 move_base 服务器进行通信，实现自主运动规划

11.3 实验工具：

个人电脑一台，智行 mini 及其配件

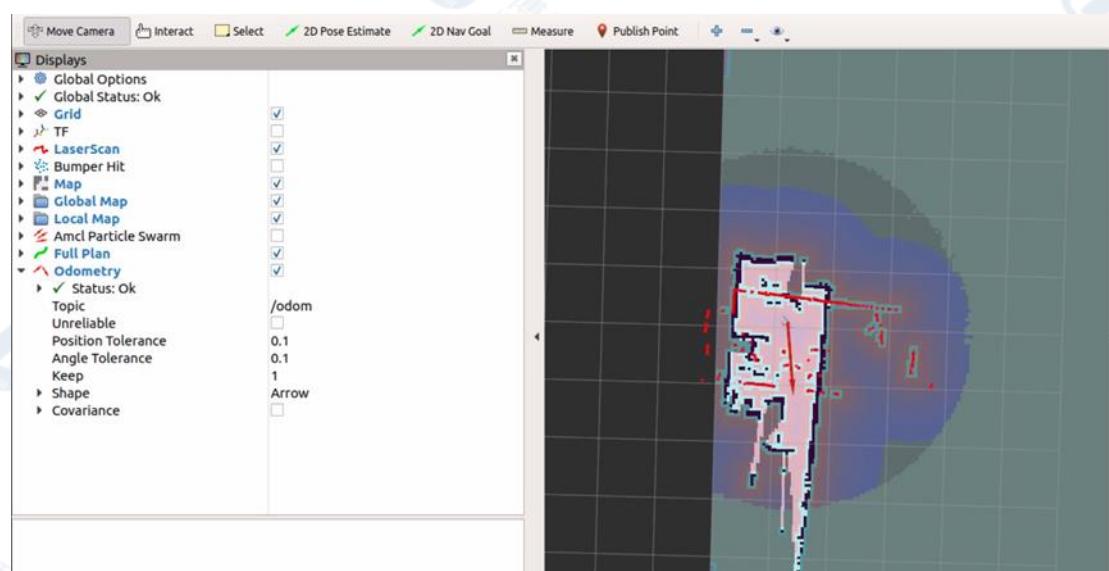
11.4 实验内容：

11.4.1 运行流程

首先打开终端，输入 **roslaunch zoo_robot robot_lidar.launch** 启动底盘，

然后打开新的终端，输入 **roslaunch robot_slam navigation.launch** 运行导航功能，

然后打开新终端，输入 **roslaunch robot_slam view_nav.launch**，就可以在 rviz 下看到机器人在环境场景中的一些信息了，如图：



按照上一节的内容，我们这里要通过 2d Nav Goal 选择目标点了，这里我们选择启动脚本，通过代码发布航点。

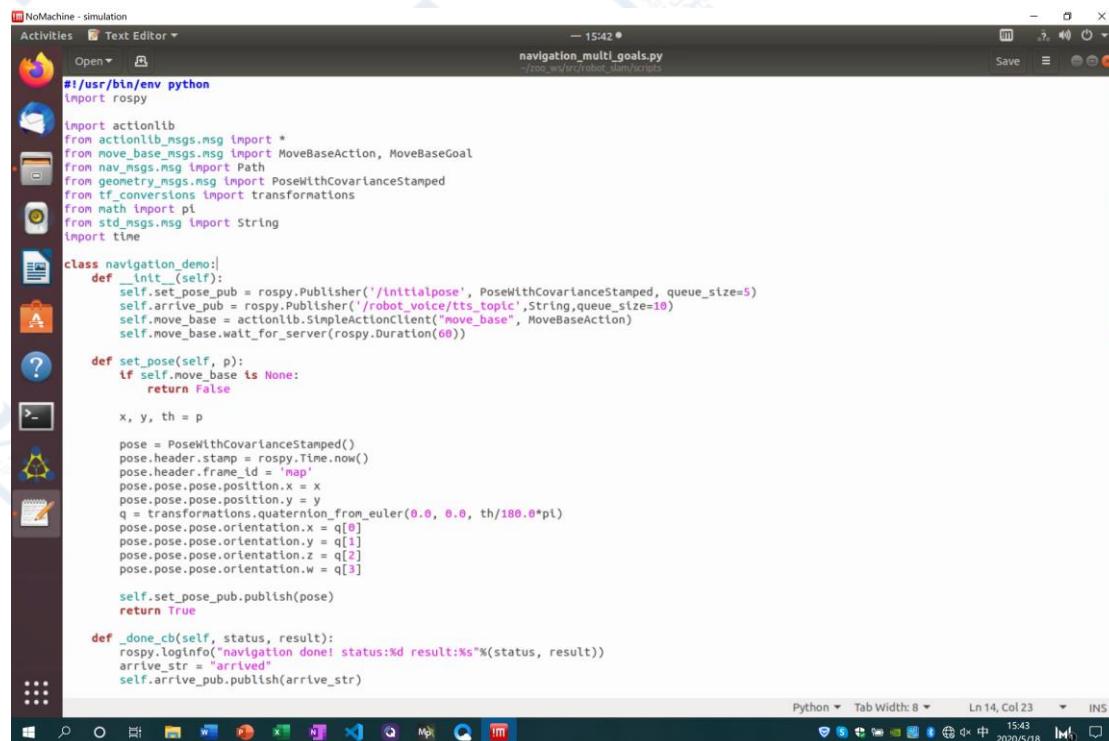
打开新的终端输入 `roslaunch robot_slam navigation_multi_demo.launch`，输入 1 确认，就会开始自动发布航点，智行 Mini 也会自动开始运动。

11.4.2 代码解读

打开我们运行自主导航的 launch 文件可以发现：

```
<?xml version="1.0"?>
<launch>
  <node pkg="robot_slam" type="navigation_multi_goals.py" respawn="false" name="navigation_multi_goals" output="screen">
    <param name="goalListX" value="4.0, 7.0" />
    <param name="goalListY" value="-1.0, -2.0" />
    <param name="goalListYaw" value="0.0, 0.0" /><!--degree-->
  </node>
</launch>
```

运行了一个名为 `navigation_multi_demo.launch` 的 python 脚本，我们找到这个脚本



```
#!/usr/bin/env python
import rospy

from actionlib_msgs.msg import *
from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal
from nav_msgs.msg import Path
from geometry_msgs.msg import PoseWithCovarianceStamped
from tf_conversions import transformations
from math import pi
from std_msgs.msg import String
import time

class navigation_demo:
    def __init__(self):
        self.set_pose_pub = rospy.Publisher('/initialpose', PoseWithCovarianceStamped, queue_size=5)
        self.arrive_pub = rospy.Publisher('/robot_voice/tts_topic', String, queue_size=10)
        self.move_base = actionlib.SimpleActionClient("move_base", MoveBaseAction)
        self.move_base.wait_for_server(rospy.Duration(60))

    def set_pose(self, p):
        if self.move_base is None:
            return False

        x, y, th = p

        pose = PoseWithCovarianceStamped()
        pose.header.stamp = rospy.Time.now()
        pose.header.frame_id = 'map'
        pose.pose.pose.position.x = x
        pose.pose.pose.position.y = y
        q = transformations.quaternion_from_euler(0.0, 0.0, th/180.0*pi)
        pose.pose.pose.orientation.x = q[0]
        pose.pose.pose.orientation.y = q[1]
        pose.pose.pose.orientation.z = q[2]
        pose.pose.pose.orientation.w = q[3]

        self.set_pose_pub.publish(pose)
        return True

    def _done_cb(self, status, result):
        rospy.loginfo("navigation done! status:%d result:%s"%(status, result))
        arrive_str = "arrived"
        self.arrive_pub.publish(arrive_str)

if __name__ == '__main__':
    navigation_demo()
```

接下来介绍代码功能：

我们将导航功能封装成了一个类，接下来分函数介绍

`init` 函数：

```
def __init__(self):
    self.set_pose_pub = rospy.Publisher('/initialpose', PoseWithCovarianceStamped, queue_size=5)
    self.arrive_pub = rospy.Publisher('/robot_voice/tts_topic', String, queue_size=10)
    self.move_base = actionlib.SimpleActionClient("move_base", MoveBaseAction)
    self.move_base.wait_for_server(rospy.Duration(60))
```

类的初始化函数，初始化导航订阅与发布的主题

`set_pose` 函数：

```

def set_pose(self, p):
    if self.move_base is None:
        return False

    x, y, th = p

    pose = PoseWithCovarianceStamped()
    pose.header.stamp = rospy.Time.now()
    pose.header.frame_id = 'map'
    pose.pose.position.x = x
    pose.pose.position.y = y
    q = transformations.quaternion_from_euler(0.0, 0.0, th/180.0*pi)
    pose.pose.orientation.x = q[0]
    pose.pose.orientation.y = q[1]
    pose.pose.orientation.z = q[2]
    pose.pose.orientation.w = q[3]

    self.set_pose_pub.publish(pose)
    return True
  
```

发布导航目标点的封装，目标点由 position 和 orientation 组成。Position 表示目标点的三维坐标（参考 ros 坐标系），orientation 代表目标点的朝向。

_done_cb, _active_cb, _feedback_cb 函数：

```

def _done_cb(self, status, result):
    rospy.loginfo("navigation done! status:%d result:%s"%(status, result))
    arrive_str = "arrived"
    self.arrive_pub.publish(arrive_str)

def _active_cb(self):
    rospy.loginfo("[Navi] navigation has been activated")

def _feedback_cb(self, feedback):
    rospy.loginfo("[Navi] navigation feedback\r\n%s"%feedback)
  
```

回调函数，反馈导航过程中的信息（抵达终点，开始导航）。

goto 函数：

```

def goto(self, p):
    rospy.loginfo("[Navi] goto %s"%p)
    arrive_str = "going to next point"
    self.arrive_pub.publish(arrive_str)
    goal = MoveBaseGoal()

    goal.target_pose.header.frame_id = 'map'
    goal.target_pose.header.stamp = rospy.Time.now()
    goal.target_pose.pose.position.x = p[0]
    goal.target_pose.pose.position.y = p[1]
    q = transformations.quaternion_from_euler(0.0, 0.0, p[2]/180.0*pi)
    goal.target_pose.pose.orientation.x = q[0]
    goal.target_pose.pose.orientation.y = q[1]
    goal.target_pose.pose.orientation.z = q[2]
    goal.target_pose.pose.orientation.w = q[3]

    self.move_base.send_goal(goal, self._done_cb, self._active_cb, self._feedback_cb)
    result = self.move_base.wait_for_result(rospy.Duration(60))
    if not result:
        self.move_base.cancel_goal()
        rospy.loginfo("Timed out achieving goal")
    else:
        state = self.move_base.get_state()
        if state == GoalStatus.SUCCEEDED:
            rospy.loginfo("reach goal %s succeeded!"%p)
    return True
  
```

发送目标点给 move_base，由 move_base 全程接管路径规划，运动控制，这里只发布命令，接收回调。

cancel 函数：

```
def cancel(self):
    self.move_base.cancel_all_goals()
    return True
```

取消导航。

```
goalListX = rospy.get_param('~goalListX', '2.0, 2.0')
goalListY = rospy.get_param('~goalListY', '2.0, 4.0')
goalListYaw = rospy.get_param('~goalListYaw', '0, 90.0')

goals = [[float(x), float(y), float(yaw)] for (x, y, yaw) in zip(goalListX.split(","), goalListY.split(","), goalListYaw.split(","))]
```

在这里，读取我们要到达的目标点合集，在 launch 文件中可见，也可以进行修改，去新的地点。

```
goal_1 = goals[0]
navi.goto(goal_1)
time.sleep(3)

goal_2 = goals[1]
navi.goto(goal_2)
time.sleep(2)
```

在这里发布了两个航点，不再赘述。

这里必须要说明的一点是，我们编辑的航点发布文件与 move_base 之间是依靠 ros 中的高级通信机制 actionlib 进行通讯的，我们只用到了它很少的一部分功能，想要实现更复杂逻辑的同学务必更加掌握 actionlib 这种通信方式。

11.5 实验结果：

能够让智行 mini 基于 actionlib 自动发布多个航点，实现自主巡航功能。

11.6 实验报告：

实验目的

实验要求

实验内容

实验总结

第十二章.单目相机驱动

12.1 实验目的:

可以在 ros 系统中使用单目 usb 摄像头

12.2 实验要求:

掌握 ros 中 usb 摄像头驱动，ros 中 usb 相机数据格式，opencv 处理图像数据

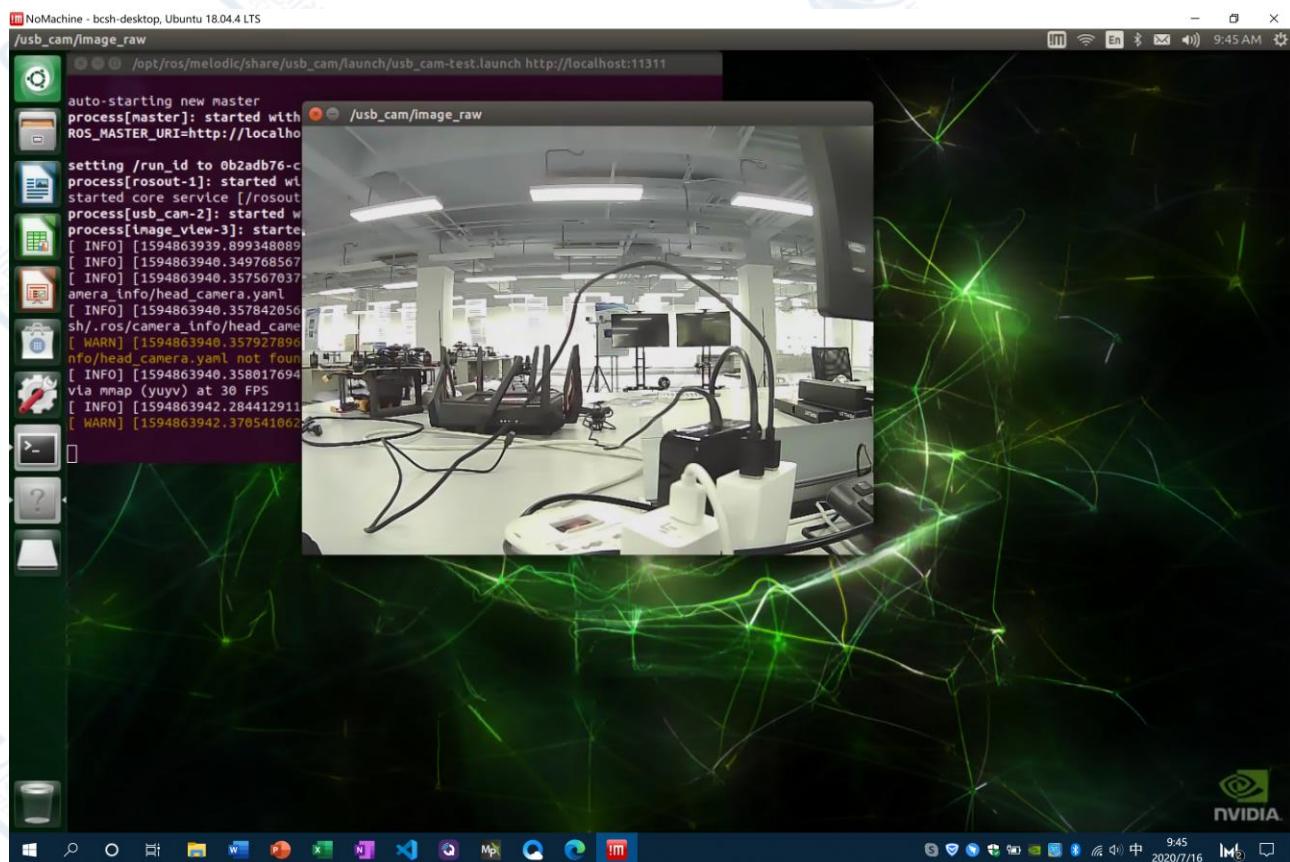
12.3 实验工具:

个人电脑一台，智行 mini 及其配件

12.4 实验内容

12.4.1 摄像头驱动实验

首先连接到我们的智行 Mini，打开新的终端输入 `roslaunch usb_cam usb_cam-test.launch`



可以看到图像显示了出来

使用以下命令查看当前系统中的图像话题信息: `rostopic info /usb_cam/image_raw`

```
bcsh@bcsh-desktop:~$ rostopic info /usb_cam/image_raw
Type: sensor_msgs/Image

Publishers:
* /usb_cam (http://bcsh-desktop:37857/)

Subscribers:
* /image_view (http://bcsh-desktop:36053/)
```

```
bcsh@bcsh-desktop:~$
```

可以看到, 图像话题的消息类型是 `sensor_msgs/Image`, 这是 ROS 定义的一种摄像头原始图像的消息类型, 可以使用以下命令查看该图像消息的详细定义: `rosmsg show sensor_msgs/Image`

```
bcsh@bcsh-desktop:~$ rosmsg show sensor_msgs/Image
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
uint32 height
uint32 width
string encoding
uint8 is_bigendian
uint32 step
uint8[] data
```

该类型图像数据的具体内容如下。

- 1) `header`: 消息头, 包含图像的序号、时间戳和绑定坐标系。
- 2) `height`: 图像的纵向分辨率, 即图像包含多少行的像素点, 这里使用的摄像头为 720。
- 3) `width`: 图像的横向分辨率, 即图像包含多少列的像素点, 这里使用的摄像头为 1280。
- 4) `encoding`: 图像的编码格式, 包含 RGB、YUV 等常用格式, 不涉及图像压缩编码。
- 5) `is_bigendian`: 图像数据的大小端存储模式。
- 6) `step`: 一行图像数据的字节数量, 作为数据的步长参数, 这里使用的摄像头为 $width \times 3 = 1280 \times 3 = 3840$ 字节。
- 7) `data`: 存储图像数据的数组, 大小为 $step \times height$ 字节, 根据该公式可以算出这里使用的摄像头产生一帧图像的数据大小是: $3840 \times 720 = 2764800$ 字节, 即 2.7648MB。一帧 720×1280 分辨率的图像数据量就是 2.76MB, 如果按照 30 帧/秒的帧率计算, 那么一秒钟摄像头产生的数据量就高达 82.944MB! 这个数据量在实际应用中是受不了的, 尤其是在远程传输图像的场景中, 图像占用的带宽过大, 会对无线网络造成很大压力。实际应用中, 图像在传输前往往进行压缩处理, ROS 也设计了压缩图像的消息类型——`sensor_msgs/CompressedImage`, 这个消息类型相比原始图像的定义要简洁不少, 除了消息头外, 只包含图像的压缩编码格式“`format`”和存储图像数据的“`data`”数组。图像压缩编码格式包含 JPEG、PNG、BMP 等, 每种编码格式对数据的结构已经进行了详细定义, 所以在消息类型的定义中省去了很多不必要的信息。

12.4.2 使用 opencv 视觉库处理图像信息

在获取到 ros 图像信息后, 我们要使用 opencv 来完成一系列操作, 那么怎么样把图像数据与 opencv 相

结合呢，我们建立这样一个 python 脚本 ros_opencv.py 输入以下代码：

```
#!/usr/bin/env python

import rospy
import cv2
from cv_bridge import CvBridge, CvBridgeError
from sensor_msgs.msg import Image
from geometry_msgs.msg import Twist
import numpy as np
import time

class ROS2OPENCV(object):
    def __init__(self, node_name):
        self.node_name = node_name
        rospy.init_node(node_name)
        rospy.on_shutdown(self.shutdown)
        self.window_name = self.node_name
        self.rgb_topic = rospy.get_param("~rgb_image_topic", "/usb_cam/image_raw")
        self.depth_topic = rospy.get_param("~depth_image_topic", "/depth/image_raw")
    )
        self.move = rospy.get_param("~if_move", False)
        self.rgb_image_sub = rospy.Subscriber(self.rgb_topic, Image, self.rgb_image_callback)
        self.depth_image_sub = rospy.Subscriber(self.depth_topic, Image, self.depth_image_callback)
        self.cmd_pub = rospy.Publisher('/cmd_vel', Twist, queue_size=1)
        self.bridge = CvBridge()

        self.frame = None
        self.frame_width = None
        self.frame_height = None
        self.frame_size = None
        self.drag_start = None
        self.selection = None
        self.track_box = None
        self.detect_box = None
        self.display_box = None
        self.marker_image = None
        self.processed_image = None
        self.display_image = None
        self.target_center_x = None
        self.cps = 0
        self.cps_values = list()
        self.cps_n_values = 20
```

```
self.linear_speed = 0.0
self.angular_speed = 0.0

#####rgb-image callback function#####
def rgb_image_callback(self, data):
    start = time.time()
    frame = self.convert_image(data)
    if self.frame is None:
        self.frame = frame.copy()
        self.marker_image = np.zeros_like(frame)
        self.frame_size = (frame.shape[1], frame.shape[0])
        self.frame_width, self.frame_height = self.frame_size
        cv2.imshow(self.window_name, self.frame)
        cv2.setMouseCallback(self.window_name, self.onMouse)
        cv2.waitKey(3)
    else:
        self.frame = frame.copy()
        self.marker_image = np.zeros_like(frame)
        processed_image = self.process_image(frame)
        self.processed_image = processed_image.copy()
        self.display_selection()
        self.display_image = cv2.bitwise_or(self.processed_image, self.marker_image)

    ####show track-box####
    if self.track_box is not None and self.is_rect_nonzero(self.track_box):
        tx, ty, tw, th = self.track_box
        cv2.rectangle(self.display_image, (tx, ty), (tx+tw, ty+th), (0, 0, 0), 2)
    ####calcate move cmd####
    self.target_center_x = int(tx+tw/2)
    offset_x = (self.target_center_x - self.frame_width/2)
    target_area = tw*th
    if self.move:
        if target_area<100:
            linear_vel = 0.0
        elif target_area >110:
            linear_vel = 0.0
        else:
            linear_vel = 0.0
        angular_vel = float(0.01*(offset_x))
        if angular_vel>0.1:
            angular_vel = 0.1
        if angular_vel<-0.1:
            angular_vel = -0.1
        self.update_cmd(linear_vel,angular_vel)
```

```
####show detect-box#####
    elif self.detect_box is not None and self.is_rect_nonzero(self.detect_box):
        dx, dy, dw, dh = self.detect_box
        cv2.rectangle(self.display_image, (dx, dy), (dx+dw, dy+dh), (255, 5
0, 50), 2)
    ####calcate time and fps#####
    end = time.time()
    duration = end - start
    fps = int(1.0/duration)
    self.cps_values.append(fps)
    if len(self.cps_values)>self.cps_n_values:
        self.cps_values.pop(0)
    self.cps = int(sum(self.cps_values)/len(self.cps_values))
    font_face = cv2.FONT_HERSHEY_SIMPLEX
    font_scale = 0.5
    if self.frame_size[0] >= 640:
        vstart = 25
        voffset = int(50+self.frame_size[1]/120.)
    elif self.frame_size[0] == 320:
        vstart = 15
        voffset = int(35+self.frame_size[1]/120.)
    else:
        vstart = 10
        voffset = int(20 + self.frame_size[1] / 120.)
    cv2.putText(self.display_image, "CPS: " + str(self.cps), (10, vstart),
font_face, font_scale,(255, 255, 0))
    cv2.putText(self.display_image, "RES: " + str(self.frame_size[0]) + "X"
+ str(self.frame_size[1]), (10, voffset), font_face, font_scale, (255, 255, 0))

    ####show result#####
    cv2.imshow(self.window_name, self.display_image)
    cv2.waitKey(3)

#####depth-image callback function#####
def depth_image_callback(self, data):
    dept_frame = self.convert_depth_image(data)

#####convert ros-image to cv2-image#####
def convert_image(self, ros_image):
    try:
        cv_image = self.bridge.imgmsg_to_cv2(ros_image, "bgr8")
        cv_image = np.array(cv_image, dtype=np.uint8)
        return cv_image
    except CvBridgeError, e:
```

```
print e

#####convert ros-depth-image to cv2-gray-image#####
def convert_depth_image(self, ros_image):
    try:
        depth_image = self.bridge.imgmsg_to_cv2(ros_image, "passthrough")
        depth_image = np.array(depth_image, dtype=np.float32)
        return depth_image
    except CvBridgeError, e:
        print e

#####process image#####
def process_image(self, frame):
    return frame

#####process depth image#####
def process_depth_image(self, frame):
    return frame

#####mouse event#####
def onMouse(self, event, x, y, flags, params):
    if self.frame is None:
        return
    if event == cv2.EVENT_LBUTTONDOWN and not self.drag_start:
        self.track_box = None
        self.detect_box = None
        self.drag_start = (x, y)
    if event == cv2.EVENT_LBUTTONUP:
        self.drag_start = None
        self.detect_box = self.selection
    if self.drag_start:
        xmin = max(0, min(x, self.drag_start[0]))
        ymin = max(0, min(y, self.drag_start[1]))
        xmax = min(self.frame_width, max(x, self.drag_start[0]))
        ymax = min(self.frame_height, max(y, self.drag_start[1]))
        self.selection = (xmin, ymin, xmax-xmin, ymax-ymin)

#####display selection box#####
def display_selection(self):
    if self.drag_start and self.is_rect_nonzero(self.selection):
        x, y, w, h = self.selection
        cv2.rectangle(self.marker_image, (x, y), (x + w, y + h), (0, 255, 255),
2)
```

```
#####calculate if rect is zero#####
def is_rect_nonzero(self, rect):
    try:
        (_,_,w,h) = rect
        return ((w>0)and(h>0))
    except:
        try:
            ((_,_),(w,h),a) = rect
            return (w > 0) and (h > 0)
        except:
            return False

#####shut down#####
def shutdown(self):
    rospy.loginfo("Shutting down node")
    cv2.destroyAllWindows()

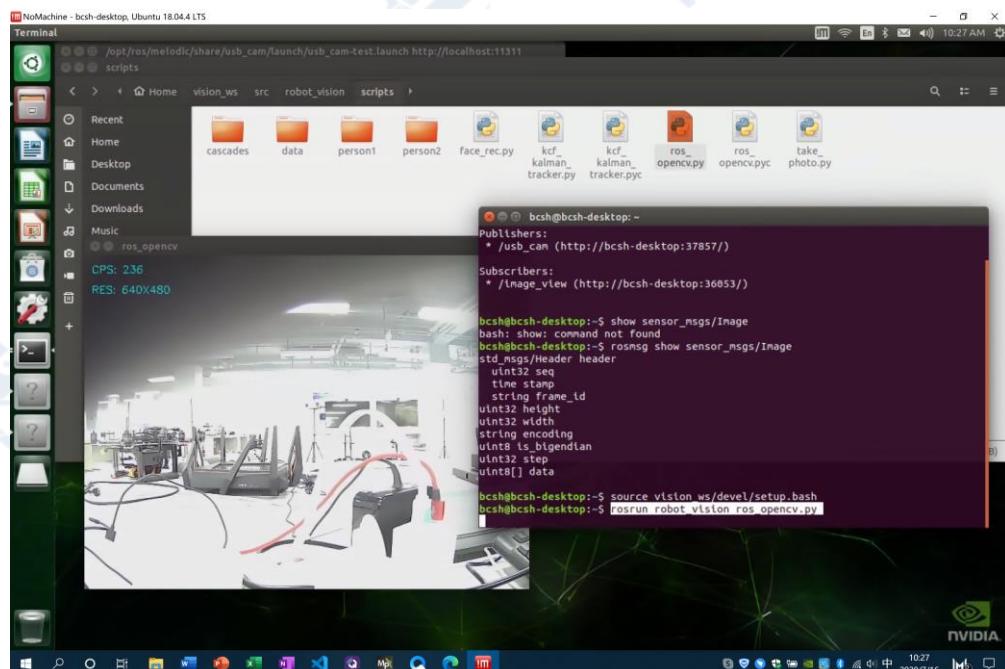
#####update cmd#####
def update_cmd(self, linear_speed, angular_speed):
    self.linear_speed = linear_speed
    self.angular_speed = angular_speed
    move_cmd = Twist()
    move_cmd.linear.x = linear_speed
    move_cmd.angular.z = angular_speed
    self.cmd_pub.publish(move_cmd)

if __name__ == '__main__':
    try:
        ROS2OPENCV("ros_opencv")
        rospy.spin()
    except KeyboardInterrupt:
        print "Shutting down cv_bridge_test node"
        cv2.destroyAllWindows()
```

在这个代码里面，我们完成了以下几个功能：

- 1) 初始化 ros 节点
- 2) 在这个节点里接收/usb_cam/image_raw 消息，转化为 opencv 可处理的图像矩阵
- 3) 将这个功能封装成 python 基类，留出图像处理接口，用于后续功能实现
- 4) 发布/cmd_vel 消息，将图像处理与运动控制相结合
- 5) 将处理帧率等消息打印在图像上。

接下来我们来运行这个节点，打开新的终端输入 `rosrun robot_vision ros_opencv.py`
得到以下结果：



可以看到在获取到图像的同时，在左上角显示出了处理的帧率和分辨率。后续想实现其他基于 opencv 的图像处理，都可以继承此基类，重写 process_image 函数，完成视觉任务。请大家务必掌握这个 python 脚本，它是后续我们所有图像操作的基础。

12.5 实验结果：

能够驱动 usb 摄像头，并使得图像数据可以对接 opencv 使用 opencv 来处理器图像。

12.6 实验报告：

实验目的

实验要求

实验内容

实验总结

第十三章.单目相机参数标定实验

13.1 实验目的:

可以在 ros 系统中标定单目 usb 摄像头

13.2 实验要求:

掌握 ros 中 usb 摄像头标定方法，标定内外参数矩阵与畸变

13.3 实验工具:

个人电脑一台，智行 mini 及其配件

13.4 实验内容

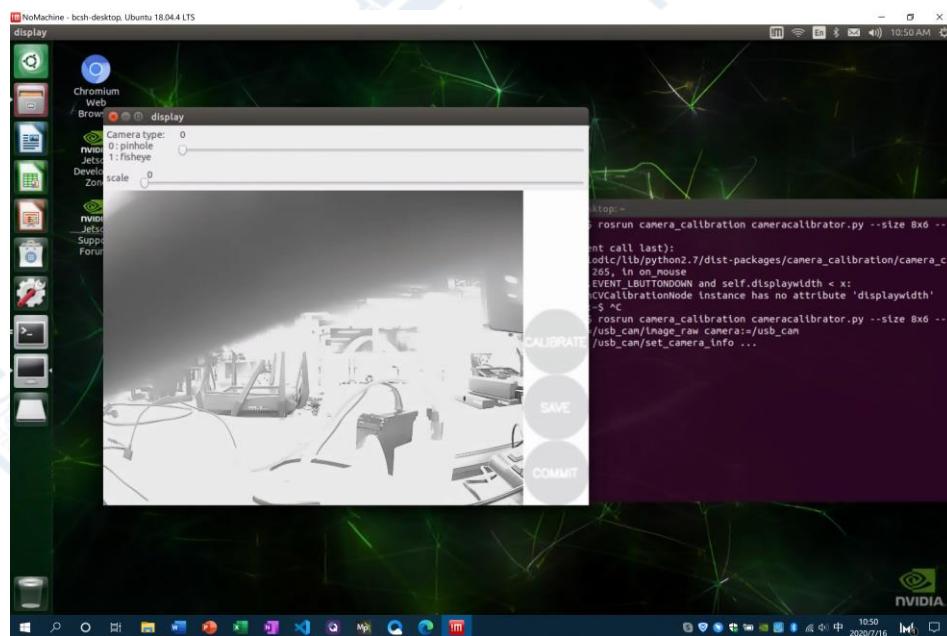
13.4.1 摄像头标定实验

摄像头这种精密仪器对光学器件的要求较高，由于摄像头内部与外部的一些原因，生成的物体图像往往会发生畸变，为了避免数据源造成的误差，需要针对摄像头的参数进行标定。ROS 官方提供了用于双目和单目摄像头标定的功能包——camera_calibration。

标定需要用到图 7-6 所示棋盘格图案的标定靶，可以在源码中找到 (robot_vision/doc/checkerboard.pdf)，请你将该标定靶打印出来贴到平面硬纸板上以备使用。

一切就绪后准备开始标定摄像头。首先使用以下命令启动 USB 摄像头：roslaunch robot_vision usb_cam.launch

然后使用以下命令启动标定程序：rosrun camera_calibration cameracalibrator.py --size 8x6 --square 0.024 image:=/usb_cam/image_raw camera:=/usb_cam



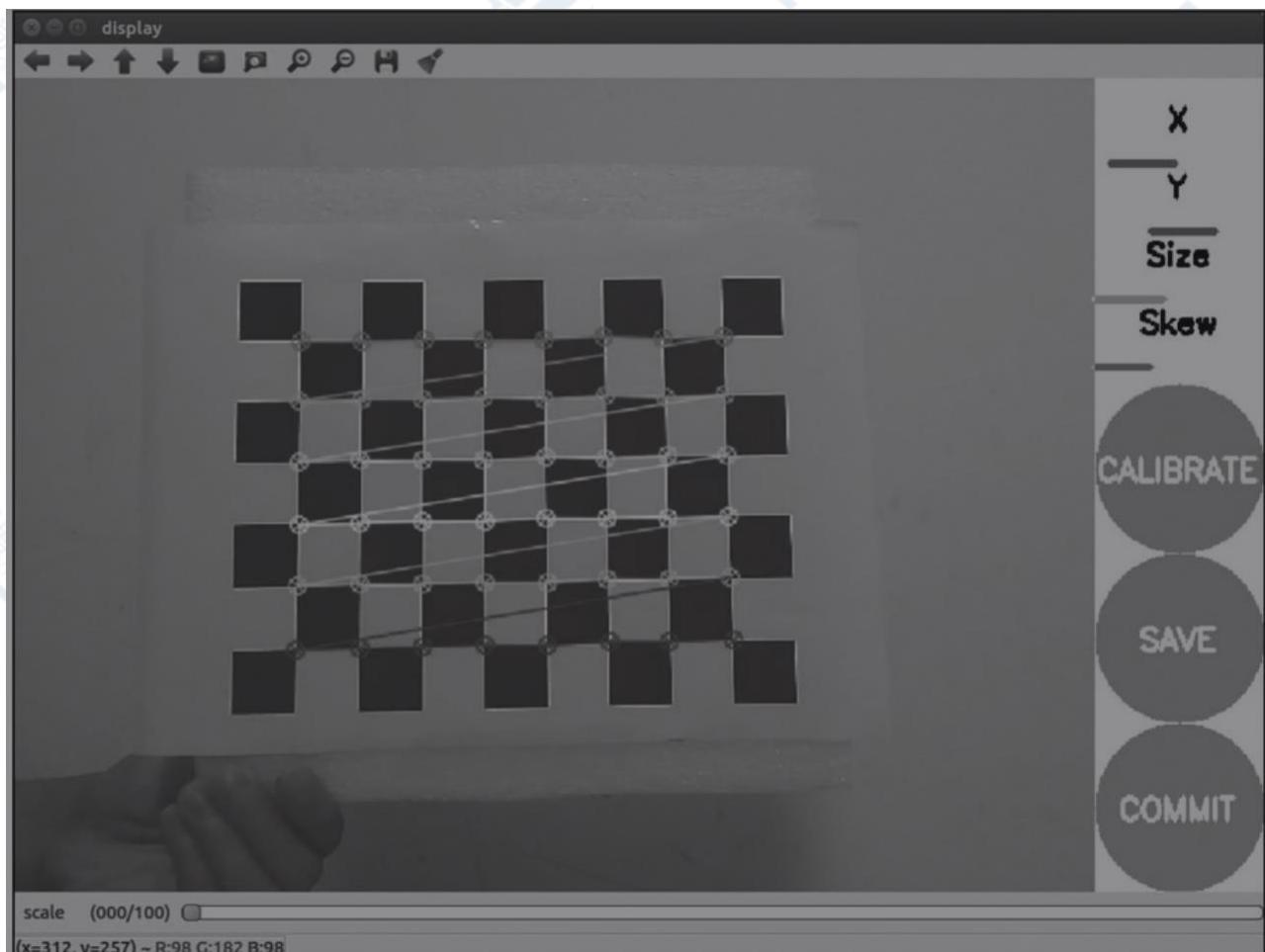
cameracalibrator.py 标定程序需要以下几个输入参数。

- 1) size: 标定棋盘格的内部角点个数, 这里使用的棋盘一共有 6 行, 每行有 8 个内部角点。
- 2) square: 这个参数对应每个棋盘格的边长, 单位是米。
- 3) image 和 camera: 设置摄像头发布的图像话题。

根据使用的摄像头和标定靶棋盘格尺寸, 相应修改以上参数, 即可启动标定程序。

标定程序启动成功后, 将标定靶放置在摄像头视野范围内, 应该可以看到如下图形界面。





在没有标定成功前，右边的按钮都为灰色，不能点击。为了提高标定的准确性，应该尽量让标定靶出现在摄像头视野范围内的各个区域，界面右上角的进度条会提示标定进度。

- 1) X: 标定靶在摄像头视野中的左右移动。
- 2) Y: 标定靶在摄像头视野中的上下移动。
- 3) Size: 标定靶在摄像头视野中的前后移动。
- 4) Skew: 标定靶在摄像头视野中的倾斜转动。

不断在视野中移动标定靶，直到“CALIBRATE”按钮变色，表示标定程序的参数采集完成。点击“CALIBRATE”按钮，标定程序开始自动计算摄像头的标定参数，这个过程需要等待一段时间，界面可能会变成灰色无响应状态，注意千万不要关闭。参数计算完成后界面恢复，而且在终端中会有标定结果的显示。

```
('D = ', [0.03172285924641212, -0.04513796090551495, 0.004146197393863798, 0.006204520963568228, 0.0])
('K = ', [514.4538649142406, 0.0, 368.31123198655376, 0.0, 514.428903410344, 224.58090408672336, 0.0, 0.0, 1.0])
('R = ', [1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0])
('P = ', [516.4229736328125, 0.0, 372.8872462120489, 0.0, 0.0, 520.513671875, 226.4447006538976, 0.0, 0.0, 0.0, 1.0, 0.0])
None
# oST version 5.0 parameters

[image]
width
640
height
480
[narrow_stereo]
camera matrix
514.453865 0.000000 368.311232
0.000000 514.428903 224.580904
0.000000 0.000000 1.000000

distortion
0.031723 -0.045138 0.004146 0.006205 0.000000

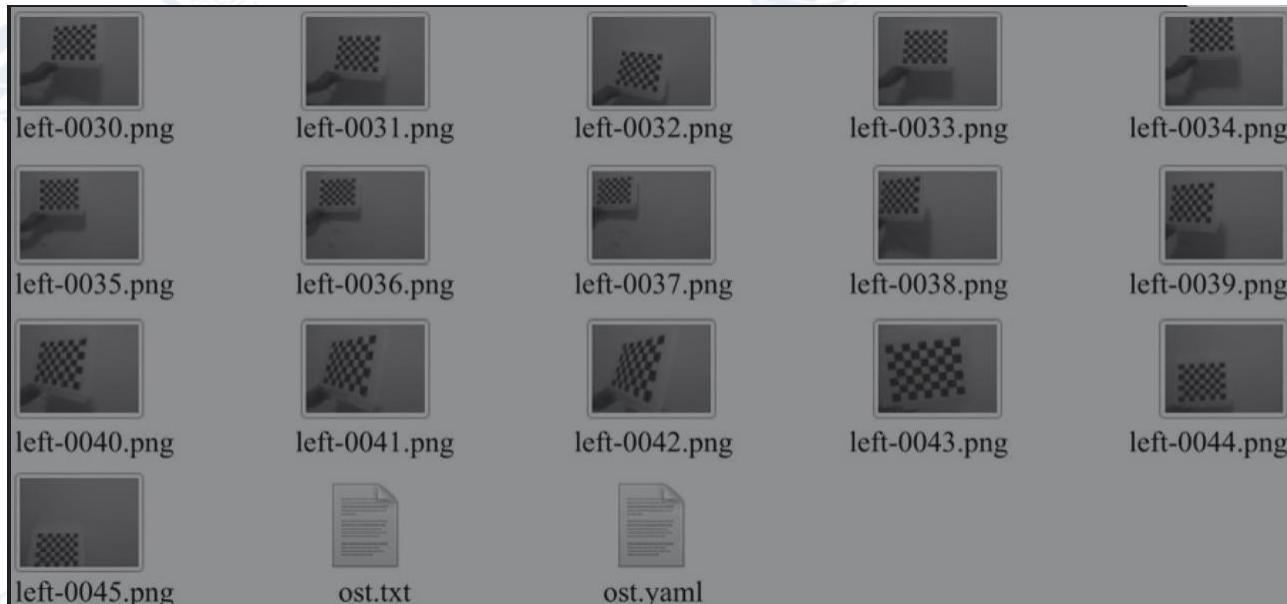
rectification
1.000000 0.000000 0.000000
0.000000 1.000000 0.000000
0.000000 0.000000 1.000000

projection
516.422974 0.000000 372.887246 0.000000
0.000000 520.513672 226.444701 0.000000
0.000000 0.000000 1.000000 0.000000
```

点击界面中的“SAVE”按钮，标定参数将被保存到默认的文件夹下，并在终端中看到该路径。

```
('Wrote calibration data to', '/tmp/calibrationdata.tar.gz')
```

点击“COMMIT”按钮，提交数据并退出程序。然后打开/tmp 文件夹，就可以看到标定结果的压缩文件 calibrationdata.tar.gz；解压该文件后的内容如下图所示，从中可以找到 ost.yaml 命名的标定结果文件，将该文件复制出来，重新命名就可以使用了。



我们将 ost.yaml 重命名为 head_camera.yaml 放在以下文件夹内

```
/home/bcsh/.ros/camera_info/head_camera.yaml
```

然后再重新启动 usb_cam 即可看到标定后的图像，得到内外参数矩阵同时消除了畸变。

13.5 实验结果：

能够使用 camera_calibration 功能包标定摄像头，去除畸变，得到内外参数矩阵。

13.6 实验报告：

实验目的

实验要求

实验内容

实验总结

第十四章.颜色形状识别与跟踪实验

14.1 实验目的:

可以在 ros 系统中基于 opencv 实现颜色与形状识别

14.2 实验要求:

掌握 opencv 中图像一些基本操作函数，霍夫变换形状识别

14.3 实验工具:

个人电脑一台，智行 mini 及其配件

14.4 实验内容

本实验着重介绍 opencv 图像变换的一些函数，以及颜色识别和形状识别的方法。首先在 robot_vision 的 scripts 文件夹内编写脚本 red_ball_detector.py，脚本内容如下所示：

```
#!/usr/bin/env python

import rospy
import cv2
import numpy as np
from ros_opencv import ROS2OPENCV
from std_msgs.msg import String

class RedBallDetector(ROS2OPENCV):
    def __init__(self, node_name):
        super(RedBallDetector, self).__init__(node_name)
        self.detect_box = None
        self.initRange()

    def process_image(self, frame):
        src = frame.copy()
        ###convert rgb to hsv###
        hsv = cv2.cvtColor(src, cv2.COLOR_BGR2HSV)

        ###create inrange mask(yellow and red)###
        res = src.copy()
        mask_red1 = cv2.inRange(hsv, self.red_min, self.red_max)
```

```
mask_red2 = cv2.inRange(hsv, self.red2_min, self.red2_max)
mask = cv2.bitwise_or(mask_red1, mask_red2)

res = cv2.bitwise_and(src, src, mask=mask)
h,w = res.shape[:2]
blured = cv2.blur(res,(5,5))
ret, bright = cv2.threshold(blured,10,255,cv2.THRESH_BINARY)
###open and close calculate###
gray = cv2.cvtColor(bright,cv2.COLOR_BGR2GRAY)
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))
opened = cv2.morphologyEx(gray, cv2.MORPH_OPEN, kernel)
closed = cv2.morphologyEx(opened, cv2.MORPH_CLOSE, kernel)
cv2.imshow("gray", closed)
circles = cv2.HoughCircles(closed, cv2.HOUGH_GRADIENT, 1, 20, param1=50, param2=30, minRadius=0, maxRadius=0)
if circles is not None:
    circles = np.uint16(np.around(circles))
    for i in circles[0, : ]:
        cv2.circle(frame, (i[0], i[1]), i[2], (0, 0, 255), 2)
        cv2.circle(frame, (i[0], i[1]), 2, (0, 0, 255), 2)
processed_image = frame.copy()
return processed_image

def initRange(self):
    self.red_min = np.array([0, 128, 46])
    self.red_max = np.array([5, 255, 255])
    self.red2_min = np.array([156, 128, 46])
    self.red2_max = np.array([180, 255, 255])

if __name__ == '__main__':
    try:
        node_name = "red_ball_detector"
        RedBallDetector(node_name)
        rospy.spin()
    except KeyboardInterrupt:
        print "Shutting down face detector node."
cv2.destroyAllWindows()
```

这里可以看出，我们实现了一个扩展类，这个扩展类继承了第十二章 ros 与 opencv 集合的基类，在这个拓展类中，我们通过一系列函数变换，能够提取出摄像头视野中的红色圆形，特别需要注意的是以下几个函数：

```
hsv = cv2.cvtColor(src, cv2.COLOR_BGR2HSV)
```

这个函数 cvtColor 将图像从 BGR 空间转化为 HSV 空间，HSV 是颜色空间模型，在这个空间里颜色是连续的更容易区分

```
self.red_min = np.array([0, 128, 46])
self.red_max = np.array([5, 255, 255])
```

```
self.red2_min = np.array([156, 128, 46])
self.red2_max = np.array([180, 255, 255])
```

```
mask_red1 = cv2.inRange(hsv, self.red_min, self.red_max)
mask_red2 = cv2.inRange(hsv, self.red2_min, self.red2_max)
```

这个函数 `inRange` 是二值化函数，将在颜色阈值内的像素值变为黑色(0)，阈值外的变为白色(255)，这样就区分出了某个特定颜色，同学们可以调整这个红色的阈值达到更好的效果。

```
res = cv2.bitwise_and(src, src, mask=mask)
```

这个函数 `bitwise_and` 是与函数，起到将两张图像黑色区域合并的效果

```
blured = cv2.blur(res,(5,5))
```

这个函数是滤波函数，滤除噪点

```
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))
opened = cv2.morphologyEx(gray, cv2.MORPH_OPEN, kernel)
closed = cv2.morphologyEx(opened, cv2.MORPH_CLOSE, kernel)
```

这个函数是连续的开闭运算函数，`MORPH_OPEN` 代表开运算，`MORPH_CLOSE` 代表闭运算，开运算和闭运算能起到滤除噪点，使得集合区域更加平滑的作用。

```
circles = cv2.HoughCircles(closed, cv2.HOUGH_GRADIENT, 1, 20, param1=50, param2=30, minRadius=0, maxRadius=0)
```

这个函数是霍夫变换函数，在一张灰度图中识别圆。

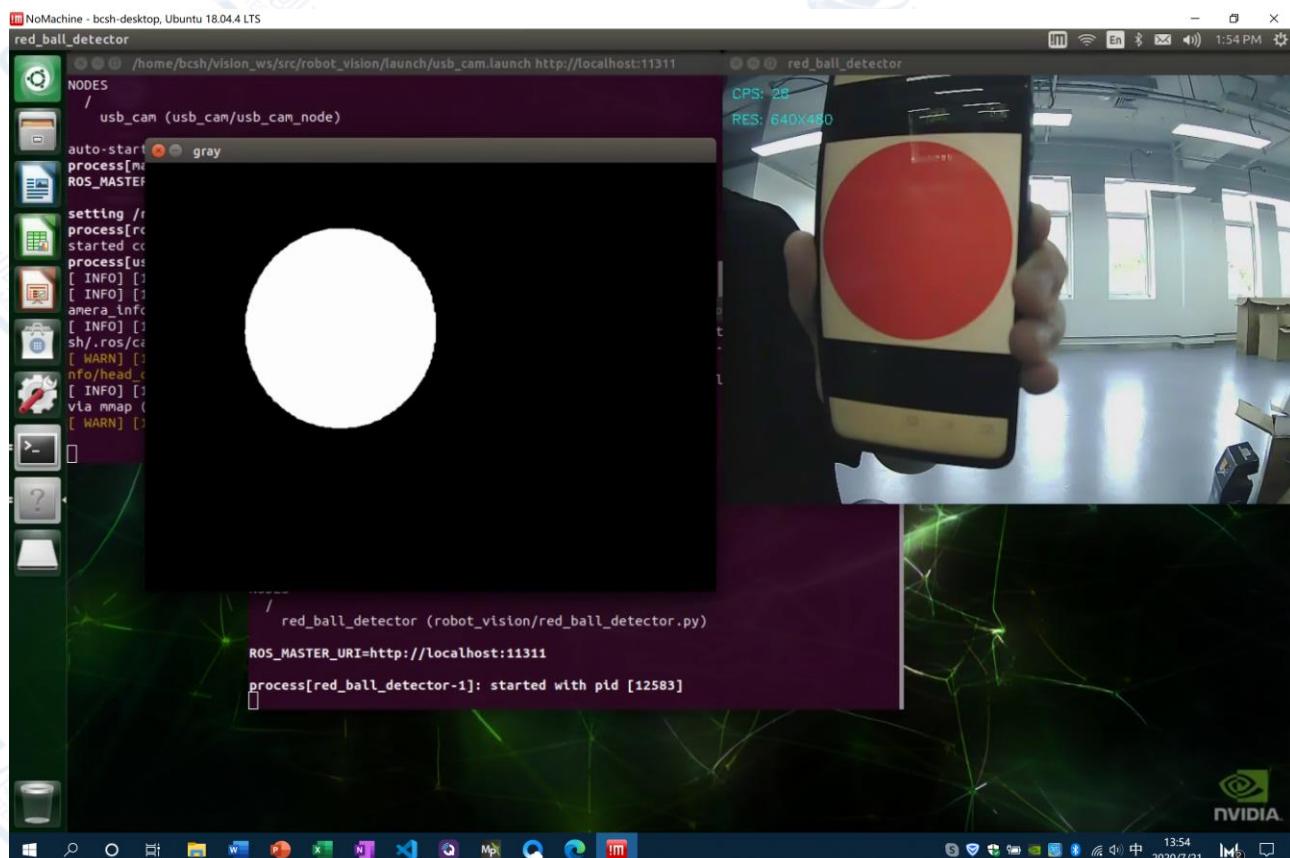
```
cv2.circle(frame, (i[0], i[1]), 2, (0, 0, 255), 2)
```

这个函数实现了在一张图上指定位置画一个圆的功能。

接下来我们来看看实现效果：

首先打开终端输入 `roslaunch robot_vision usb_cam.launch` 打开摄像头

然后打开终端输入 `roslaunch robot_vision red_ball_detector.launch` 进行红色圆识别。



正确识别出了圆形。

14.5 实验结果：

能够熟练掌握 opencv 图像处理常用函数，进行颜色识别，形状识别。

14.6 实验报告：

实验目的

实验要求

实验内容

实验总结

第十五章.基于 opencv 的人脸识别实验

15.1 实验目的:

学习基于 ros 与单目相机的人脸识别功能

15.2 实验要求:

掌握基于 ros 与 opencv 的人脸检测，人脸数据保存，人脸识别功能

15.3 实验工具:

个人电脑一台，智行 mini 及其配件

15.4 实验内容

15.4.1 人脸检测功能实现

要想正确识别出是谁，首先要具备提取出人脸的能力，及正确识别出人脸所在位置。

2001 年，Viola 和 Jones 提出了基于 Haar 特征的级联分类器对象检测算法，并在 2002 年由 Lienhart 和 Maydt 进行改进，为快速、可靠的人脸检测应用提供了一种有效方法。OpenCV 已经集成了该算法的开源实现，利用大量样本的 Haar 特征进行分类器训练，然后调用训练好的瀑布级联分类器 cascade 进行模式匹配。

接下来我们来看看如何通过 opencv 来实现人脸检测。在 robot_vision 中新建一个 face_detector.py 文件，并输入以下代码：

```
#!/usr/bin/env python

import rospy
import cv2
from ros_opencv import ROS2OPENCV
import os
import sys

class FaceDetector(ROS2OPENCV):
    def __init__(self, node_name):
        super(FaceDetector, self).__init__(node_name)
        self.detect_box = None
        self.result = None
        self.count = 0
```

```
self.face_cascade = cv2.CascadeClassifier('/home/bcsh/vision_ws/src/robot_vision/scripts/cascades/haarcascade_frontalface_default.xml')

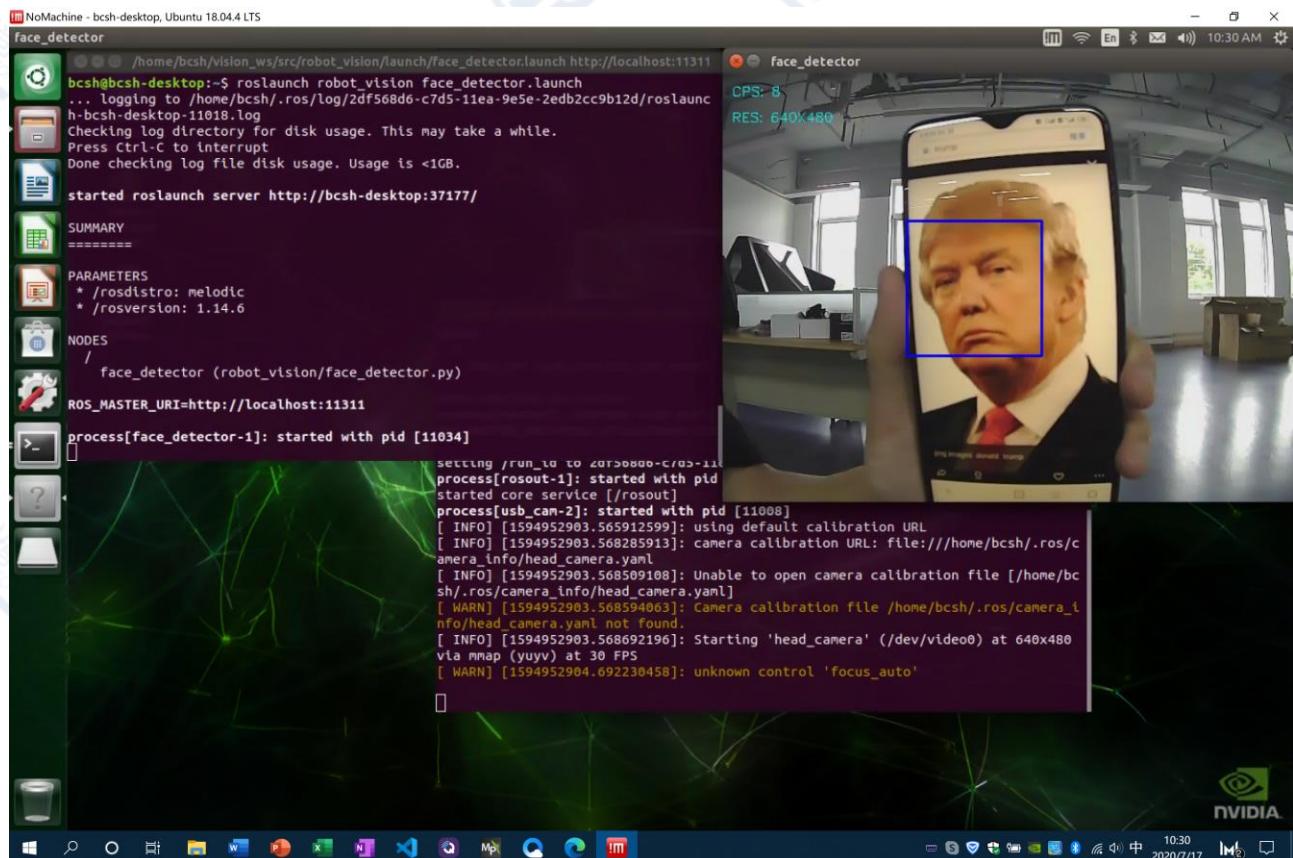
def process_image(self, frame):
    src = frame.copy()
    gray = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY)
    faces = self.face_cascade.detectMultiScale(gray, 1.3, 5)
    result = src.copy()
    self.result = result
    for (x, y, w, h) in faces:
        result = cv2.rectangle(result, (x, y), (x+w, y+h), (255, 0, 0), 2)
    return result

if __name__ == '__main__':
    try:
        node_name = "face_detector"
        FaceDetector(node_name)
        rospy.spin()
    except KeyboardInterrupt:
        print "Shutting down face detector node."
cv2.destroyAllWindows()
```

从以上代码我们不难看出，这里我们生成了一个继承类，继承了之前 ROS 与 Opencv 接口类，在这个类里面我们重写了 `process_image` 函数，使得该函数可以完成人脸识别功能。核心函数为 `detectMultiScale` 函数，这个函数实现了将视频中的人脸提取出来，反馈值为 `faces`，`faces` 是由多个数组组成，每个数组代表人脸在当前图像中的位置 `(x, y, w, h)` 分别代表人脸框的左上角点的坐标，人脸框的宽度和长度。

我们首先打开终端输入 `roslaunch robot_vision usb_cam.launch` 启动摄像头

然后打开新的终端输入 `roslaunch robot_vision face_detector.launch` 启动人脸检测



可以看到正确的检测出了人脸。

15.4.2 人脸数据采集

完成人脸数据检测后，我们想要进行人脸的识别，在识别正确的人脸之前，我们必须完成训练，训练就要求我们有人脸数据。接下来我们就来实现人脸数据的采集。

首先新建一个 `take_photo.py` 的脚本，在这个脚本中输入以下代码

```
#!/usr/bin/env python

import rospy
import cv2
from ros_opencv import ROS2OPENCV
import sys, select, os
if os.name == 'nt':
    import msvcrt
else:
    import tty, termios

def getKey():
    if os.name == 'nt':
        return msvcrt.getch()
    tty.setraw(sys.stdin.fileno())
    rlist, _, _ = select.select([sys.stdin], [], [], 0.1)
    if rlist:
```

```
key = sys.stdin.read(1)
else:
    key = ''
termios.tcsetattr(sys.stdin, termios.TCSADRAIN, settings)
return key

class TakePhoto(ROS2OPENCV):
    def __init__(self, node_name):
        super(TakePhoto, self).__init__(node_name)
        self.detect_box = None
        self.result = None
        self.count = 0
        self.person_name = rospy.get_param('~person_name', 'name_one')
        self.face_cascade = cv2.CascadeClassifier('/home/bcsh/vision_ws/src/robot_vision/scripts/cascades/haarcascade_frontalface_default.xml')
        self.dirname = "/home/bcsh/vision_ws/src/robot_vision/scripts/data/" + self.person_name + "/"
        if (not os.path.isdir(self.dirname)):
            os.makedirs(self.dirname)

    def process_image(self, frame):
        key = getKey()
        src = frame.copy()
        gray = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY)
        faces = self.face_cascade.detectMultiScale(gray, 1.3, 5)
        result = src.copy()
        self.result = result
        for (x, y, w, h) in faces:
            result = cv2.rectangle(result, (x, y), (x+w, y+h), (255, 0, 0), 2)
            f = cv2.resize(gray[y:y+h, x:x+w], (200, 200))
            if self.count<20:
                if key == 'p' :
                    cv2.imwrite(self.dirname + '%s.pgm' % str(self.count), f)
                    self.count += 1
        return result

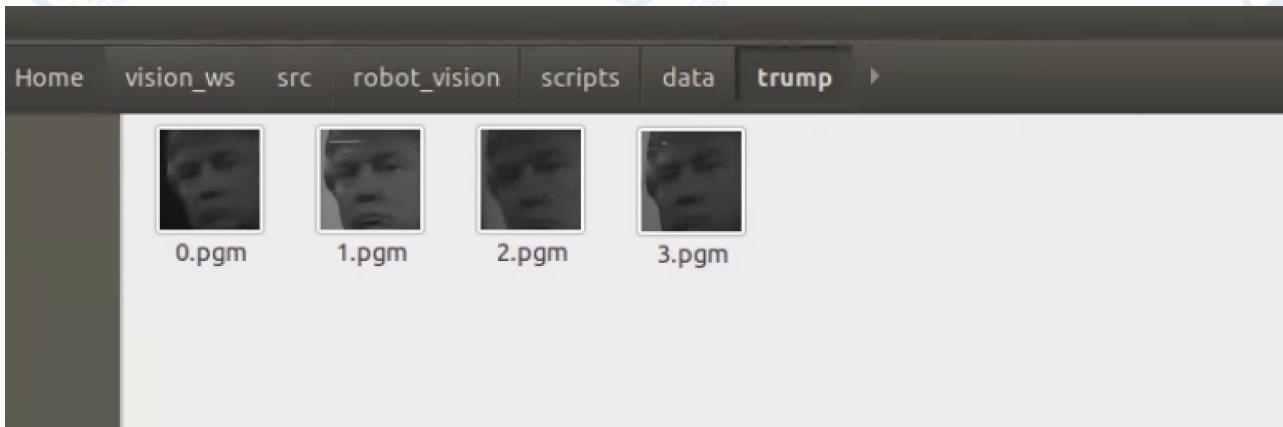
if __name__ == '__main__':
    if os.name != 'nt':
        settings = termios.tcgetattr(sys.stdin)
    try:
        node_name = "take_photo"
        TakePhoto(node_name)
        rospy.spin()
```

```
except KeyboardInterrupt:  
    print "Shutting down face detector node."  
cv2.destroyAllWindows()
```

可以看到，这是我们人脸检测程序的升级版本，它主要实现了，检测到人脸之后，按下“p”键即可自动存储人脸图像。我们来看看这个脚本的 launch 文件：

```
<launch>  
  <node name="take_photo" type="take_photo.py" pkg="robot_vision" output="screen">  
    <param name="person_name" value="trump"/>  
  </node>  
</launch>
```

如上所示，`person_name` 参数即代表了要存储的这个人的名字，想换个人就直接修改这个参数即可。接下来我们先打开终端启动相机驱动，然后打开新的终端输入 `roslaunch robot_vision take_photo.launch`，把鼠标点击在这个终端，然后人脸出现在屏幕上，在不同的方位检测到人脸后按下 p 拍照，即可存储人脸图像。存储后的图像在这里：



这里建议大家多换几个角度拍照，识别出来的效果会更好。

15.4.3 人脸识别功能实现

采集足够的人脸数据之后，我们要开始完成人脸识别功能，新建 `face_recognize.py` 脚本，输入以下代码：

```
#!/usr/bin/env python  
  
import rospy  
import os  
import sys  
import cv2  
import numpy as np  
from ros_opencv import ROS2OPENCV  
  
def read_images(path, sz=None):  
    c = 0  
    X, y = [], []  
    names = []  
    for dirname, dirnames, filenames in os.walk(path):
```

```
for subdirname in dirnames:
    subject_path = os.path.join(dirname, subdirname)
    for filename in os.listdir(subject_path):
        try:
            if (filename == ".directory"):
                continue
            filepath = os.path.join(subject_path, filename)
            im = cv2.imread(os.path.join(subject_path, filename), cv2.IMREAD_GRAYSCALE)
            if (im is None):
                print("image" + filepath + "is None")
            if (sz is not None):
                im = cv2.resize(im, sz)
            X.append(np.asarray(im, dtype=np.uint8))
            y.append(c)
        except:
            print("unexpected error")
            raise
    c = c+1
    names.append(subdirname)
return [names, X, y]

class FaceRecognizer(ROS2OPENCV):
    def __init__(self, node_name):
        super(FaceRecognizer, self).__init__(node_name)
        self.detect_box = None
        self.result = None
        self.names = None
        self.X = None
        self.Y = None
        self.face_cascade = cv2.CascadeClassifier('/home/bcsh/vision_ws/src/robot_vision/scripts/cascades/haarcascade_frontalface_default.xml')
        self.read_dir = "/home/bcsh/vision_ws/src/robot_vision/scripts/data"
        [names, X, Y] = read_images(self.read_dir)
        Y = np.asarray(Y, dtype=np.int32)
        self.names = names
        self.X = X
        self.Y = Y
        self.model = cv2.face_EigenFaceRecognizer.create()
        self.model.train(np.asarray(X), np.asarray(Y))

    def process_image(self, frame):
        src = frame.copy()
        gray = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY)
        faces = self.face_cascade.detectMultiScale(gray, 1.3, 5)
```

```
result = src.copy()
self.result = result
for (x, y, w, h) in faces:
    result = cv2.rectangle(result, (x, y), (x+w, y+h), (255, 0, 0), 2)
    roi = gray[x:x+w, y:y+h]
    try:
        roi = cv2.resize(roi, (200,200), interpolation=cv2.INTER_LINEAR)
        [p_label, p_confidence] = self.model.predict(roi)
        print"confidence: %s"%(p_confidence)
        if p_confidence>9500:
            cv2.putText(result, names[p_label], (x, y-20), cv2.FONT_HERSHEY_SIMPLEX, 1, 255, 2)
    except:
        continue
return result

if __name__ == "__main__":
    try:
        node_name = "face_recognizer"
        FaceRecognizer(node_name)
        rospy.spin()
    except KeyboardInterrupt:
        print "Shutting down face detector node."
cv2.destroyAllWindows()
```

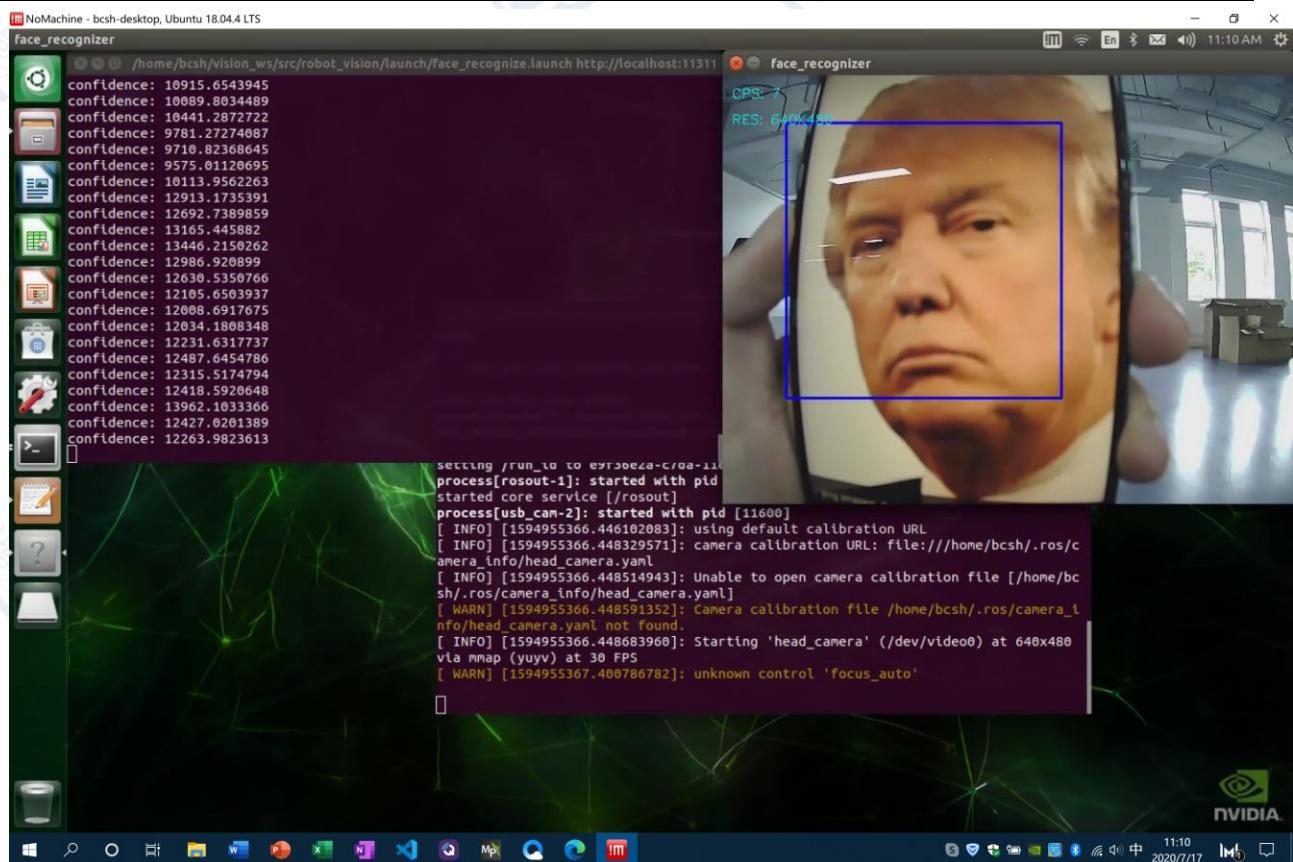
在这段代码中，我们首先读取存储的图像数据，人名，打包成对应的数组，然后使用 model.train 函数完成人脸识别模型的训练，随后在重写的 process_image 函数中先检测人脸，然后用我们训练好的模型识别人脸。特别需要注意这段代码：

```
try:
    roi = cv2.resize(roi, (200,200), interpolation=cv2.INTER_LINEAR)
    [p_label, p_confidence] = self.model.predict(roi)
    print"confidence: %s"%(p_confidence)
    if p_confidence>9500:
        cv2.putText(result, names[p_label], (x, y-20), cv2.FONT_HERSHEY_SIMPLEX, 1, 255, 2)
```

model.predict 函数实现了人脸的预测，它返回了两个参数：

p_label: 人脸编号，即人脸是谁

p_confidence: 识别置信度，置信度越高，说明识别效果越可靠。我们先来运行一下这个节点，打开新的终端输入 rosrun robot_vision face_recognize.launch



可以看到，在这里我们打印出了致信度，大家可以修改这个置信度阈值（这里我们是 9500），超过这个阈值即可判断识别正确。

人脸识别是机器视觉中的重要应用，请同学们多训练多调参数达到更好的识别效果。

15.5 实验结果：

能够使用 opencv 视觉库实现人脸识别功能。

15.6 实验报告：

实验目的

实验要求

实验内容

实验总结

第十六章.基于 kcf 的目标跟踪实验

16.1 实验目的:

学习基于 ros 与单目相机的目标跟踪功能

16.2 实验要求:

掌握 opencv 的 kcf 跟踪算法，卡尔曼滤波算法

16.3 实验工具:

个人电脑一台，智行 mini 及其配件

16.4 实验内容

16.4.1 目标跟踪功能实现

在 robot_vision 功能包中创建脚本 kcf_kalman_tracker.py。输入以下内容：

```
#!/usr/bin/env python

import rospy
import cv2
import numpy as np
from ros_opencv import ROS2OPENCV

class KcfKalmanTracker(ROS2OPENCV):
    def __init__(self, node_name):
        super(KcfKalmanTracker, self).__init__(node_name)
        self.tracker = cv2.TrackerKCF_create()
        self.detect_box = None
        self.track_box = None
        #####init kalman#####
        self.kalman = cv2.KalmanFilter(4, 2)
        self.kalman.measurementMatrix = np.array([[1,0,0,0],[0,1,0,0]],np.float32)
        self.kalman.transitionMatrix = np.array([[1,0,1,0],[0,1,0,1],[0,0,1,0],[0,0,0,1]],np.float32)
        self.kalman.processNoiseCov = np.array([[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]],np.float32)*0.03
        self.measurement = np.array((2,1),np.float32)
```

```
self.prediction = np.array((2,1),np.float32)

def process_image(self, frame):
    try:
        if self.detect_box is None:
            return frame
        src = frame.copy()
        if self.track_box is None or not self.is_rect_nonzero(self.track_box):
            self.track_box = self.detect_box
        if self.tracker is None:
            raise Exception("tracker not init")
        status = self.tracker.init(src, self.track_box)
        if not status:
            raise Exception("tracker initial failed")
    else:
        self.track_box = self.track(frame)
    except:
        pass
    return frame

def track(self, frame):
    status, coord = self.tracker.update(frame)
    center = np.array([[np.float32(coord[0]+coord[2]/2)],[np.float32(coord[1]+coord[3]/2)]])
    self.kalman.correct(center)
    self.prediction = self.kalman.predict()
    cv2.circle(frame, (int(self.prediction[0]),int(self.prediction[1])),4,(255,60,100),2)
    round_coord = (int(coord[0]), int(coord[1]), int(coord[2]), int(coord[3]))
    return round_coord

if __name__ == '__main__':
    try:
        node_name = "kcfkalmantracker"
        KcfKalmanTracker(node_name)
        rospy.spin()
    except KeyboardInterrupt:
        print "Shutting down face detector node."
cv2.destroyAllWindows()
```

由以上代码可知，先生成了一个继承类继承继承了 ROS_OPENCV 基类的功能与函数，然后通过这条代码生成了 opencv 库中的 KCF 跟踪器

```
self.tracker = cv2.TrackerKCF_create()
```

接下来通过以下代码来生成卡尔曼滤波器及其参数

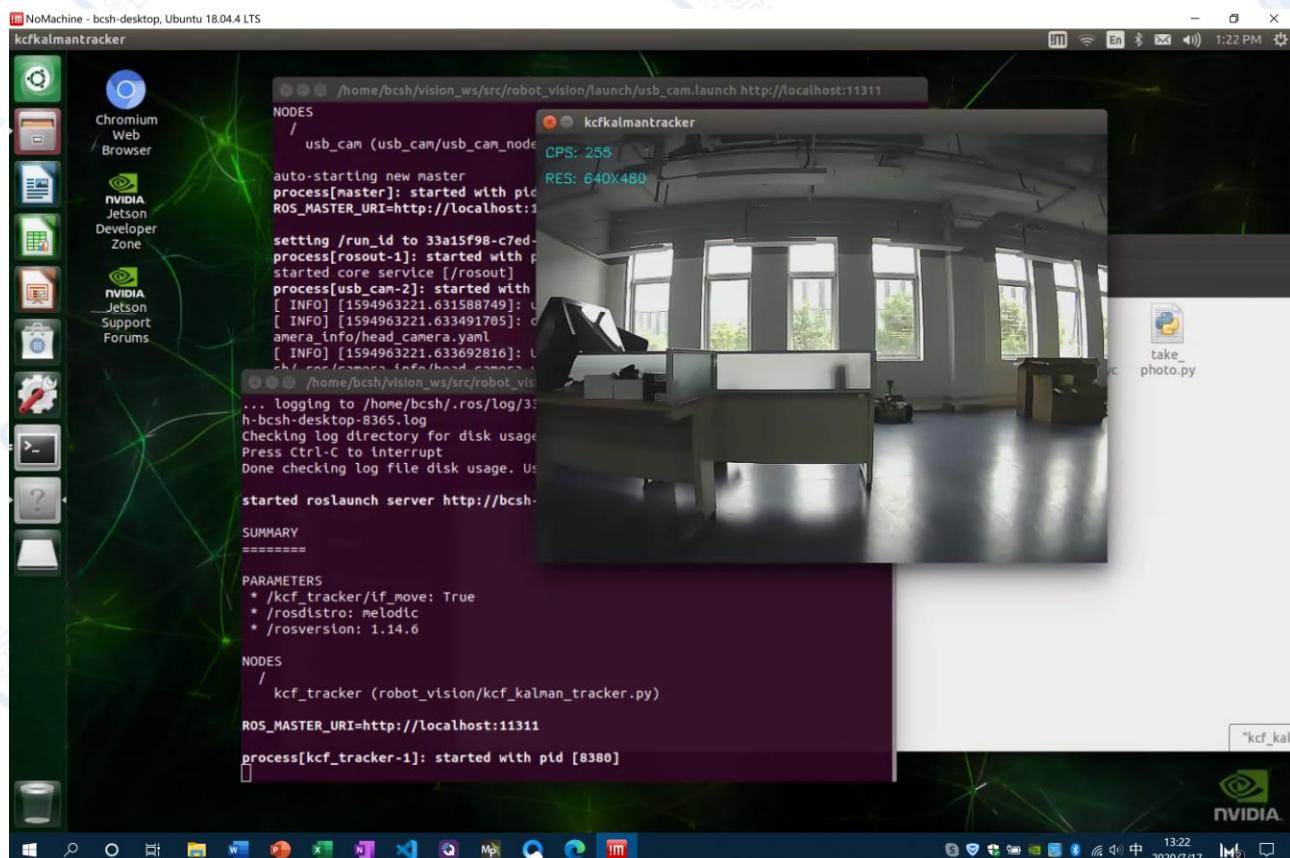
```
self.kalman = cv2.KalmanFilter(4, 2)
    self.kalman.measurementMatrix = np.array([[1,0,0,0],[0,1,0,0]],np.float32)
    self.kalman.transitionMatrix = np.array([[1,0,1,0],[0,1,0,1],[0,0,1,0],[0,0,0,1]],np.float32)
    self.kalman.processNoiseCov = np.array([[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]],np.float32)*0.03
    self.measurement = np.array((2,1),np.float32)
    self.prediction = np.array((2,1),np.float32)
```

在 track 函数中，我们通过 kcf 跟踪器实现目标的跟踪，然后通过卡尔曼滤波器锁定目标中心。

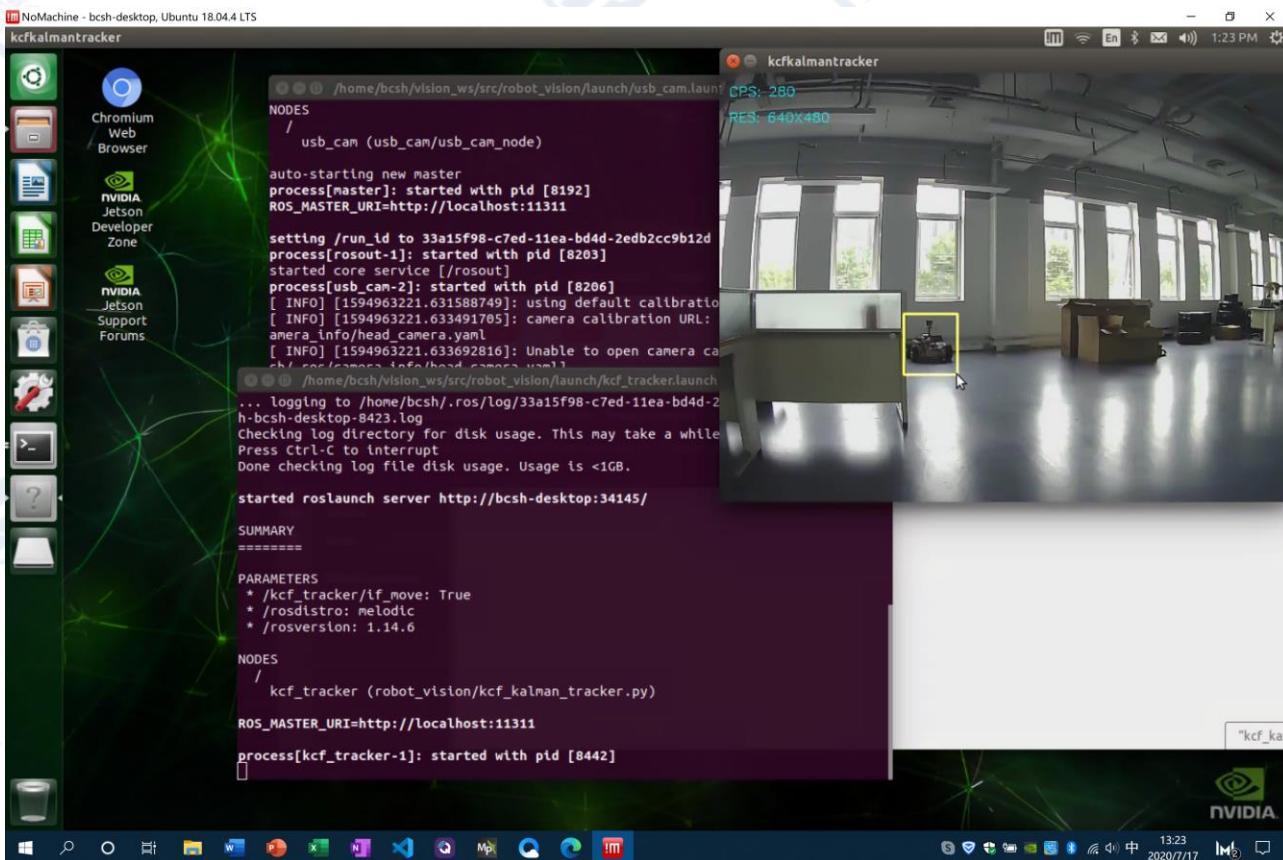
```
self.prediction = self.kalman.predict()
cv2.circle(frame, (int(self.prediction[0]),int(self.prediction[1])),4,(255,60,100),2)
```

可以看到 prediction 即为输出的目标点。

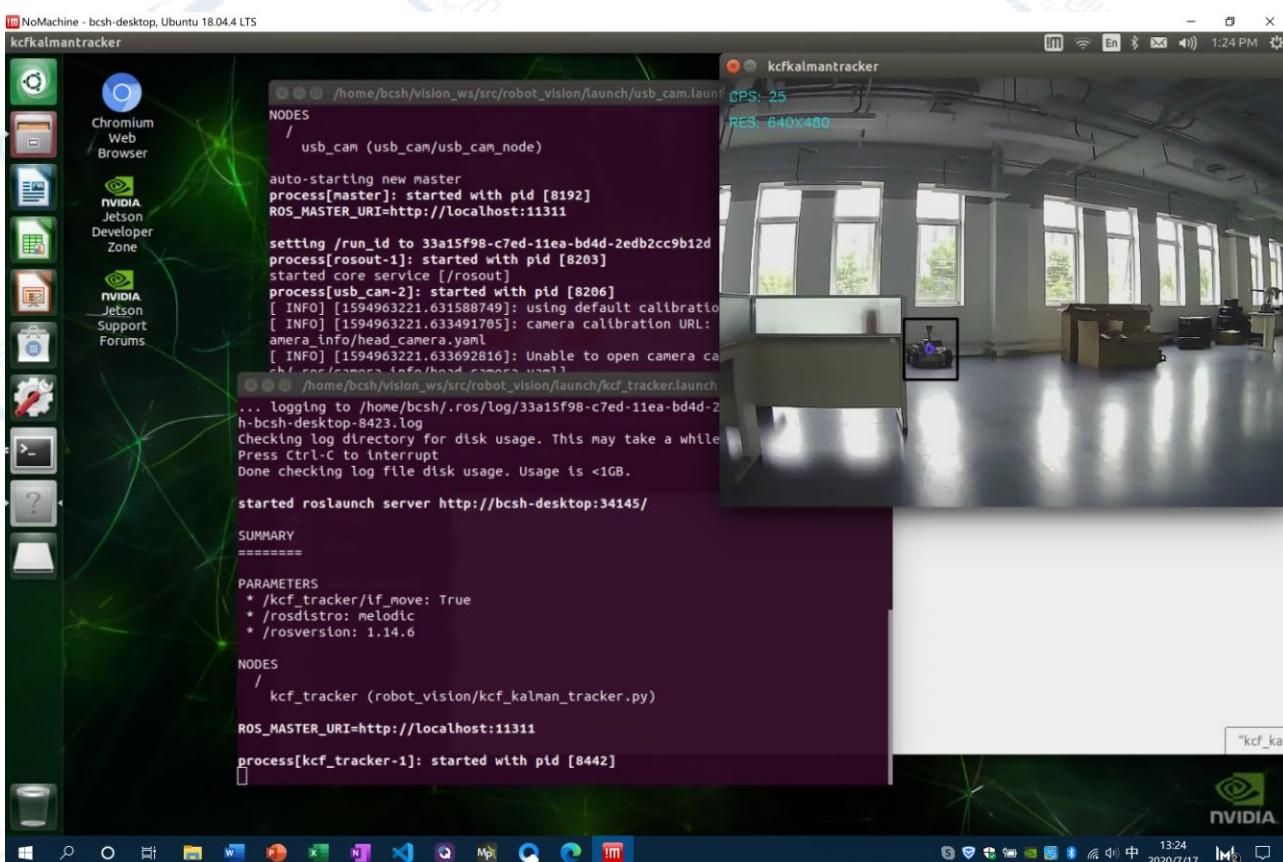
连接我们的智行 Mini，打开摄像头驱动后，打开新的终端输入 rosrun robot_vision kcf_tracker.launch



使用鼠标左键框选出要跟踪的目标



可以看到跟踪效果：



这时可以控制小车运动，完成跟踪效果。

16.5 实验结果：

能够使用 opencv 视觉库实现框选目标跟踪功能。

16.6 实验报告：

实验目的

实验要求

实验内容

实验总结

第十七章.卷积神经网络目标识别实验

17.1 实验目的:

可以在 ros 系统中依靠深度学习算法实现目标识别

17.2 实验要求:

掌握 ros 中 darknet_ros 框架，通过 yolo 算法实现目标识别

17.3 实验工具:

个人电脑一台，智行 mini 及其配件

17.4 实验内容

在这个案例中，我们将从神经网络数据训练与目标检测两个方面介绍。

17.4.1 训练自己的卷积神经网络框架

智行 Mini 中布署了 tiny-YOLO v3 算法进行目标的检测。

YOLO 系列的目标检测算法可以说是目标检测史上的宏篇巨作，YOLOv3 创造性的提出了 one-stage。也就是将物体分类和物体定位在一个步骤中完成。yolo 直接在输出层回归 bounding box 的位置和 bounding box 所属类别，从而实现 one-stage。通过这种方式，在 GeForce1070Ti 下可实现 20 帧每秒的运算速度。顾名思义，YOLOv3 是 YOLO 算法中的第三代。YOLO 算法从最开始的 YOLOv1 一路演化过来在很多细节与网络结构方面都有较大的改变。

在本篇中，从实现的角度切入，带领大家一步步实现训练自己的数据集，让大家先对目标检测的方法和工程环境有定性的认识。**在设置的过程中会尽量把步骤写详细，请大家在做的每一步确认与本文的内容完全一致，如需要自定义的地方会有注明。**

训练的基本流程为：

- 准备训练数据集

为了识别目标，我们需要采集目标相关的视觉信息。方法就是用相机为目标拍一系列照片，将包含目标的一组照片整理到一个文件夹中，就形成了“数据集”。

- 准备训练计算机

YOLO 模型十分复杂，需要配备独立显卡的终端设备才可以运行。显卡算力越高，程序计算得越快，推荐使用 GTX 1050Ti 及以上的显卡。运行 Ubuntu 16.04 x86_64 系统最佳。需要安装 CUDA10.1 及以上版本，及 cudnn7.2.1 以上版本，注意 CUDA 和 cudnn 的版本对应。

- 准备训练环境

我们通过 darknet 来实现 yolo 的训练过程，所以需要告诉 yolo 我们的一系列配置文件与数据集存放地址，然后运行 darknet 的程序运行训练过程。

接下来我们开始配置训练环境。

我们现在假设您的电脑安装好了 ubuntu 16.04 cuda cudnn (必须的三项)。

首先，在您的个人计算机上打开终端输入以下指令：

```
sudo apt-get install git
```

在这一步我们首先安装 git 代码管理工具

然后继续输入：

```
git clone https://github.com/pjreddie/darknet.git
```

显示如下：

```
Cloning into 'darknet'...
remote: Enumerating objects: 5901, done.
remote: Total 5901 (delta 0), reused 0 (delta 0), pack-reused 5901
Receiving objects: 100% (5901/5901), 6.16 MiB | 96.00 KiB/s, done.
Resolving deltas: 100% (3916/3916), done.
```

显示 100% 后就下载完成了。我们来看 darknet 的文件结构：

在终端里面输入：

```
cd darknet
tree -L 1
```

显示如下：

```
.
├── cfg
├── data
├── examples
├── include
├── LICENSE
├── LICENSE.fuck
├── LICENSE.gen
├── LICENSE/gpl
├── LICENSE.meta
├── LICENSE.mit
├── LICENSE.v1
├── Makefile
├── python
├── README.md
├── scripts
└── src
```

其中有几个比较重要的文件和文件夹，它们是：

| 文件 (夹) | 说明 |
|-----------|--------------------------------------|
| backup | 训练后的网络模型都保存在这里 |
| cfg | 所有的训练环境设置相关 |
| data | 存放数据集的地方 |
| Makefile | darknet的重要编译配置文件，比如是否能够用GPU加速的设置就在这里 |

代码是由 pjreddie 大神用 C 写的，所以只有源代码还不能运行，我们需要对其进行编译，这个源代码是一个 makefile 组织的工程（所以代码根目录才会有 Makefile 文件）。编译之前，需要对 Makefile 文件进行设置：

在命令行输入：

```
cd ~/darknet
gedit Makefile
```

gedit 命令会打开 Makefile 文件，Makefile 的前几行意思如下：

```
GPU=0 # 是否使用GPU? 0: 否, 1: 是
CUDNN=0 # 是否使用cudnn? 0: 否, 1: 是
OPENCV=0 # 是否使用OpenCV? 0: 否, 1: 是
OPENMP=0 # 是否使用OpenNMP? 0: 否, 1: 是
DEBUG=0 # 是否使用debug模式? 0: 否, 1: 是
```

我们需要使用 GPU，这样会使用独立显卡进行计算，加快模型计算速度。另外也需要 cudnn，这是英伟达公司为自家显卡设计的专为深度神经网络的加速模块，也需要使用 OpenCV 做图像处理，所以设置如下：

```
GPU=1
CUDNN=1
OPENCV=1
OPENMP=0
DEBUG=0
```

更改后按保存后关闭文件。打开命令行，输入：

```
cd ~/darknet  
make
```

程序将开始编译，编译完成后，我们再来看它的目录：输入：

```
tree -L 1
```

显示如下：

```
.  
├── backup  
├── build  
├── cfg  
├── darknet  
├── data  
├── examples  
├── include  
├── libdarknet.a  
├── libdarknet.so  
├── LICENSE  
├── LICENSE.fuck  
├── LICENSE.gen  
├── LICENSE.gpl  
├── LICENSE.meta  
├── LICENSE.mit  
├── LICENSE.v1  
├── Makefile  
├── obj  
├── python  
├── README.md  
├── results  
├── scripts  
└── src
```

增加了一些文件，这时可以看是否存在 libdarknet.so 和 darknet 文件，如果有，就证明编译成功了。
接下来准备要训练的数据集。

深度学习要完成对目标的检测，首先要知道哪些是目标。这就需要我们在训练前先在数据集中标注出来目标的位置。Yolo_mark 是一个开源的图像标注工具，我们可以利用它生成符合 YOLO 格式的标注文件。

首先下载 YOLO_mark，打开命令行，输入：

```
cd ~  
git clone https://github.com/AlexeyAB/Yolo\_mark.git
```

同样的，我们介绍下 Yolo_mark 的工程目录：

```
cd ~/Yolo_mark  
tree -L 1
```

显示如下：

```
.  
├── CMakeCache.txt  
├── CMakeFiles  
├── cmake_install.cmake  
├── CMakeLists.txt  
├── LICENSE  
├── linux_mark.sh  
├── main.cpp  
├── Makefile  
├── README.md  
├── x64  
└── yolo_mark  
    ├── yolo_mark.sln  
    └── yolo_mark.vcxproj
```

Yolo_mark 是一个 cmake 工程，同样，我们来看其中需要的重要关注的文件（夹）：

| 文件（夹） | 说明 |
|------------------|---------------|
| linux_mark.sh | 标注软件的设置与启动脚本 |
| x64.Release.data | 默认的数据集设置与存放目录 |

注：上表中，文件夹名称中的'!'代表子目录。

它的编译步骤如下：

```
cmake .  
make
```

在启动标注程序之前，需要对标注程序启动脚本进行设置。打开 linux_mark.sh：

```
gedit linux_mark.sh
```

看到文件其实是执行 yolo_mark 程序，我们需要设置其后的参数，第一个参数是数据集存放的地址，第二个参数是设置文件索引的地址，第三个参数是目标种类的表格文件位置。所以我们需要把这三个参数做如下修改：

```
echo      Example how to start marking bouded boxes for tra  
ining set Yolo v2
```

```
./yolo_mark /home/${USER_NAME}/darknet/data/coco/images/tr  
ain /home/${USER_NAME}/darknet/data/coco/train.txt /hom  
e/${USER_NAME}/darknet/data/obj.names
```

```
pause
```

这其实是为了方便后面存放数据集，注意路径必须是绝对路径，里边的\${USER_NAME}需要替换成大家的登录用户名。

先在 darknet 文件夹中建立存放位置：

```
cd ~/darknet/data  
mkdir -p coco/images  
mkdir -p coco/labels
```

再把之前准备好的数据集拷贝到/home/\${USER_NAME}/darknet/data/coco/images 文件夹下。

第二个参数是在运行标注软件后会在该文件里自动生成文件索引，不需要手动设置内容。

接下来我们设置目标类别文件，我们输入：

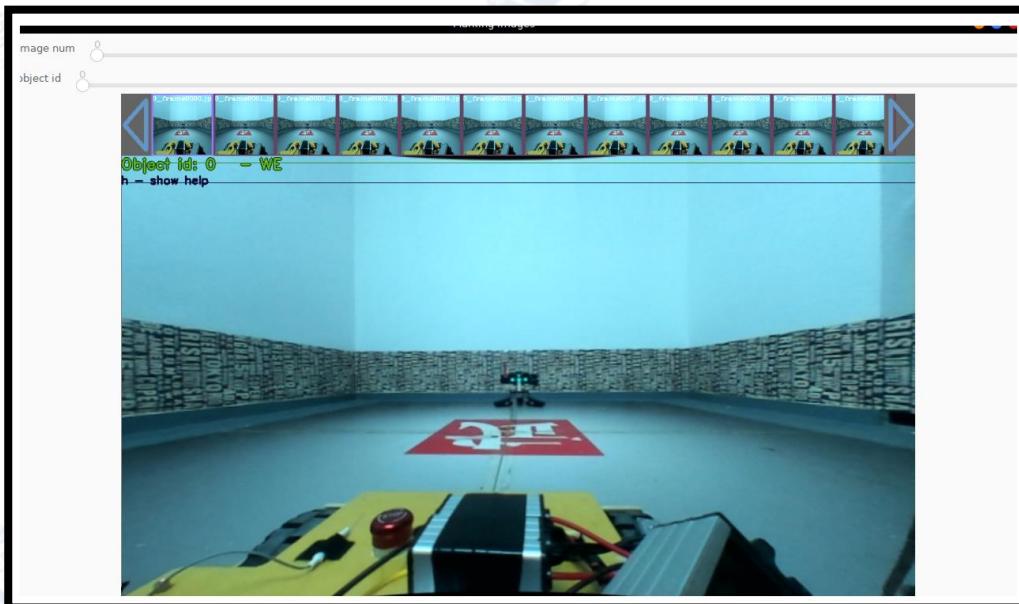
```
gedit ~/darknet/data/coco/obj.names
```

这里面的每一行有一个单词，代表一个品类的名称。这里要写入我们告诉算法我们要检测的目标的名称，删除 obj.names 里的内容，并且输入目标

分类的名称（本例中以“WE”为例），保存。

现在我们打开标注软件：

```
cd ~/Yolo_mark  
sh linux_mark.sh
```



在软件的主界面的上方显示的是总体图像的数量和类别的数量，其下方是数据集中各图片的预览，再接下来绿的的文字显示目标的名称，接下来

就是标注的主界面，在这里，我们将对每张图片进行标注。

在标注的过程中我们需要用到一些快捷键，希望大家能在使用的时候慢慢记住这些快捷键，会极大增加标注的效率：

鼠标操作

| 鼠标按键 | 说明 |
|--------|-------|
| 鼠标左键拖动 | 画标注框 |
| 鼠标右键拖动 | 移动标注框 |

键盘操作

| 快捷键 | 说明 |
|------|--------------------------|
| 方向右键 | 下一张图片 |
| 方向左键 | 上一张图片 |
| r | 删除选中的标注框（当鼠标在其上悬停时为选中状态） |
| c | 将该图片上的所有标注删除 |
| p | 复制上一张的标注 |
| o | 跟随目标 |
| Esc | 关闭标注程序 |
| n | 单框标注模式 |
| 0-9 | 类别选取序号 |
| m | 显示坐标 |
| w | 标注线宽 |
| k | 隐藏目标名称 |
| h | 帮助 |

由上表可知，我们拖动鼠标左键将目标框在其中，注意通过其他的鼠标和键盘的快捷键保证标注的框是能够将目标完整包含的最小框。如果目标不规则，用框选取可能会暴露出大量的背景信息，则需要适当减小框的尺寸。

当我们把数据集全部标注完成后，按 Esc 退出软件，恭喜，最苦最累的工作我们已经做完了，接下来真正进入到数据训练的步骤。

将我们之前的 `~/darknet/data/coco/images/train` 文件夹下的标签全部选中，剪切到 `~/darknet/data/coco/labels/train` 中。

下载 tiny-yolov3 的权重文件 `yolov3-tiny.weights` 和预训练好的权重 `darknet53.conv.74`:

```
cd ~/darknet
wget https://pjreddie.com/media/files/yolov3-tiny.weights
wget https://pjreddie.com/media/files/darknet53.conv.74
```

用 darknet 中已经预配置好的环境，现在可以测试一下效果：

```
./darknet detect cfg/yolov3-tiny.cfg yolov3-tiny.weights data/dog.jpg
# 第一个参数是神经网络结构描述文件，第二个参数是权重文件，第三个是待检测的图像
```

接下来进行 darknet 环境配置：

我们在~/darknet/cfg/文件夹下建立文件 my_we.data:

```
cd ~/darknet
touch my_we.data
gedit my_we.data
```

在文件中写入：

```
classes = 1
train = /home/${USER_NAME}/darknet/data/coco/my_we.txt
valid = /home/${USER_NAME}/darknet/data/coco/my_we.txt
names = data/obj.names
backup = backup
eval = coco
```

其中 classes 就是要分类的总类别，我们此处只检测 WE，所以是 1。

将 /home/\${USER_NAME}/darknet/cfg/yolov3-tiny.cfg 复制一份，重命名为 yolov3-tiny-train.cfg，并修改：

```
mv /home/${USER_NAME}/darknet/cfg/yolov3-tiny.cfg yolov3-tiny-train.cfg
gedit /home/${USER_NAME}/darknet/cfg/yolov3-tiny-train.cfg
```

将文件的前 7 行改为：

```
[net]
# Testing
# batch=1
# subdivisions = 1
# Training
batch = 64
subdivisions = 2
```

注意修改\${USER_NAME}，如果显存比较小，可以把 batch = 64 改成 32 或 16
准备完成！接下来就开始我们的训练

```
cd ~/darknet
./darknet detector train cfg/my_we.data cfg/yolov3-tiny-train.cfg darknet53.conv.74
```

上图中的 000000 代表训练了多少步，yolo 中是默认 10000 步以前每 1000 步更新显示一次，10000 步以后每 10000 步更新显示一次。Class 代表当前

训练分类的正确率，在 Class 的正确率达到 0.9（参考值）的时候可以按 Ctrl + C 停止训练，可以检查在 backup 中已经保存了训练的权重值。

这个过程将会较长（在 1070Ti 的显卡上大概需要中运行 10 个小时以达到 0.9 以上的正确率），请大家耐心等待结果。

```
15 conv      18 1 x 1 / 1    13 x 13 x 512    ->    13 x 13 x 18  0.003 BFLOPs
16 yolo
17 route 13
18 conv    128 1 x 1 / 1    13 x 13 x 256    ->    13 x 13 x 128  0.011 BFLOPs
19 upsample           2x    13 x 13 x 128    ->    26 x 26 x 128
20 route 19 8
21 conv    256 3 x 3 / 1    26 x 26 x 384    ->    26 x 26 x 256  1.196 BFLOPs
22 conv    18 1 x 1 / 1    26 x 26 x 256    ->    26 x 26 x 18   0.006 BFLOPs
23 yolo
Loading weights from yolov3-tiny.conv.15...Done!
Learning Rate: 0.001, Momentum: 0.9, Decay: 0.0005
Resizing
644
Loaded: 0.298004 seconds
Region 16 Avg IOU: 0.154911, Class: 0.542475, Obj: 0.385543, No Obj: 0.507530, .5R: 0.000000, .75R: 0
000000, count: 9
Region 23 Avg IOU: 0.196190, Class: 0.678789, Obj: 0.582617, No Obj: 0.492592, .5R: 0.000000, .75R: 0
000000, count: 4
Region 16 Avg IOU: 0.406785, Class: 0.456425, Obj: 0.330342, No Obj: 0.509826, .5R: 0.500000, .75R: 0
000000, count: 4
Region 23 Avg IOU: 0.261887, Class: 0.593361, Obj: 0.530446, No Obj: 0.491866, .5R: 0.000000, .75R: 0
000000, count: 7
Region 16 Avg IOU: 0.297758, Class: 0.340539, Obj: 0.442540, No Obj: 0.507202, .5R: 0.000000, .75R: 0
000000, count: 7
Region 23 Avg IOU: -nan, Class: -nan, Obj: -nan, No Obj: 0.492202, .5R: -nan, .75R: -nan, count: 0
Region 16 Avg IOU: 0.191769, Class: 0.644537, Obj: 0.251000, No Obj: 0.505752, .5R: 0.000000, .75R: 0
000000, count: 8
Region 23 Avg IOU: 0.233697, Class: 0.597563, Obj: 0.395009, No Obj: 0.492591, .5R: 0.125000, .75R: 0
000000, count: 8
Region 16 Avg IOU: 0.160983, Class: 0.604025, Obj: 0.436177, No Obj: 0.507331, .5R: 0.000000, .75R: 0
000000, count: 4
```

到此，我们的神经网络已经训练好了。

17.4.2 识别自定义目标

我们所有的程序都是在 ROS 框架下进行开发的。ROS 系统就是一个节点通信框架，在机器人（无人机、无人车、仿人形机器人等）开发中各个运动关节，各个功能节点在 ROS 中都可以抽象成为一个个节点（node），各个节点之间控制指令的传递与状态的反馈就连接形成了一个协调统一的运动系统。所以如果想要在能够在 ROS 框架下使用 YOLO，就要将 darknet 与 ROS 结合形成一个功能包，所幸已经有大牛将 YOLO 写成了非常好用 ROS 包——darknet_ros。

我们在打开 aviator 的机载处理器，打开终端输入

```
roscore
```

```
tree -L 2
```

目录树如下：

```
. └── darknet      # yolo所在目录
    ├── cfg
    ├── data
    ├── examples
    ├── include
    ├── LICENSE
    ├── LICENSE.fuck
    ├── LICENSE.gen
    ├── LICENSE.gpl
    ├── LICENSE.meta
    ├── LICENSE.mit
    ├── LICENSE.v1
    ├── Makefile
    ├── python
    ├── README.md
    ├── scripts
    └── src
└── darknet_ros   # darknet_ros主要功能都在这里
    ├── CHANGELOG.rst
    ├── CMakeLists.txt
    ├── config
    ├── doc
    ├── include
    ├── launch
    ├── package.xml
    ├── src
    ├── test
    └── yolo_network_config
└── darknet_ros_msgs  # darknet_ros的自定义信息
    ├── action
    ├── CHANGELOG.rst
    ├── CMakeLists.txt
    ├── msg
    └── package.xml
└── jenkins-pipeline
└── LICENSE
└── README.md
```

darknet_ros_msgs 中定义了两种消息类型，分别是 BoundingBox.msg 和 BoundingBoxes.msg。我们知道在 yolo 算法中是可能在同一张图片上发现一组目标的位置的，所以设置这两个消息类型一个是为了传递一个目标位置的信息，另一个是为了传递一张图片上的目标位置信息。

darknet_ros 文件夹是一个标准的 ROS 包，我们需要关注的是 src 文件夹，关键的 ros 消息收发规则都在这里实现。config 文件夹中包含了一系列关于网络信息的设置和 darknet_ros 节点相关的设置。

在之前中我们提到 yolo 把训练好的模型保存在 backup 文件夹下，我们把在训练计算机上最新的权重文件和 ~/darknet/cfg/yolov3-tiny.cfg 这两个文件分别复制到智行 Mini ~/vision_ws/src/darknet_ros/darknet_ros/yolo_network_config 文件夹下的 cfg 和 weights 文件夹下。

在~/vision_ws/src/darknet_ros/darknet_ros/config 文件夹中新建 yolov3_tiny_we.yaml，在其中写入：

```
yolo_model:  
  
    config_file:  
        name: yolov3-tiny.cfg  
    weight_file:  
        name: yolov3-tiny-we.weights  
    threshold:  
        value: 0.3  
    detection_classes:  
        names:  
            - WE
```

注意上面的 config_file 和 weight_file 的名字就写我们刚刚复制两个文件的文件名。threshold 是激活值，就是说神经网络在判断一个位置是目标的概率大于这个阀值的时候就认为这个区域是目标，修改这个值是把双刃剑，当这个值设小的时候查全率高，查准率低，反之则查全率低，查准率高。

在~/visison_ws/src/darknet_ros/darknet_ros/launch 中将 darknet_ros.launch 复制一份，命名为 we.launch，修改里面的

```
<rosparam command="load" ns="darknet_ros" file="$(find darknet_ros)/config/yolov2-tiny.yaml"/>为  
<rosparam command="load" ns="darknet_ros" file="$(find  
darknet_ros)/config/yolov3_tiny_we.yaml"/>
```

以指定我们刚刚新建的 yolov3_tiny_we.yaml。

最后，再修改~/vision_ws/src/darknet_ros/darknet_ros/cfg/ros.yaml 文件中的 camera_reading::topic 一行，这行指定了我们的 darknet_ros 节点接收来自哪个节点的图像消息，aviator 中的摄像头发布的消息为 /usb_cam/image_raw。

到此，我们的 darknet_ros 环境已经配置好了。

最后我们来看一下如何运行：

首先，我们需要开启摄像头：

```
1. rosrun usb_cam usb_cam-test.launch
```

然后启动目标检测：

```
1. rosrun darknet_ros we.launch
```

启动成功后，在预览窗口中显示预测的目标位置。

我们可以通过 rostopic list 和 rosmsg list 看到 darknet_ros 发布的主题和消息都有哪些。如果需要修改，则查看~/vision_ws/src/darknet_ros/darknet_ros/src/中的代码。

17.5 实验结果：

能够使用 Yolo 深度学习算法实现目标识别。

17.6 实验报告：

实验目的

实验要求

实验内容

实验总结

第十八章.语音阵列驱动实验

18.1 实验目的:

在 ros 系统中测试 respeaker mic array 语音阵列。

18.2 实验要求:

掌握远场语音阵列驱动操作，实现语音录制，LED 控制等操作

18.3 实验工具:

个人电脑一台，智行 mini 及其配件

18.4 实验内容

在这个案例中，我们将从麦克风驱动与阵列驱动两部分介绍。

18.4.1 语音阵列介绍

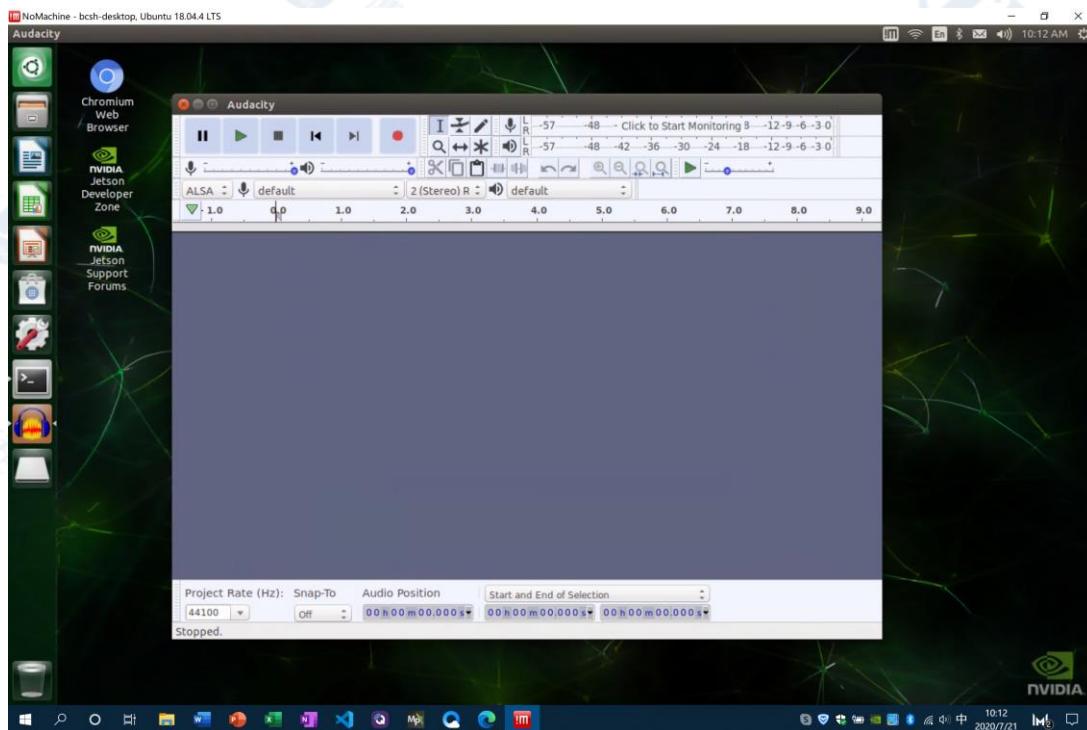


Seeed 的 ReSpeaker Mic Array v2.0 是基于 XVSM-2000 的 ReSpeaker Mic Array v1.0 的升级版本。v2.0 是基于 XMOS 的 XVF-3000 开发。它可以堆叠(连接)到 ReSpeaker Core 的顶部，显著改善语音交互体验。该主板集成了 4 个 PDM 麦克风，有助于将 ReSpeaker 的声学 DSP 性能提高到更高的水平。

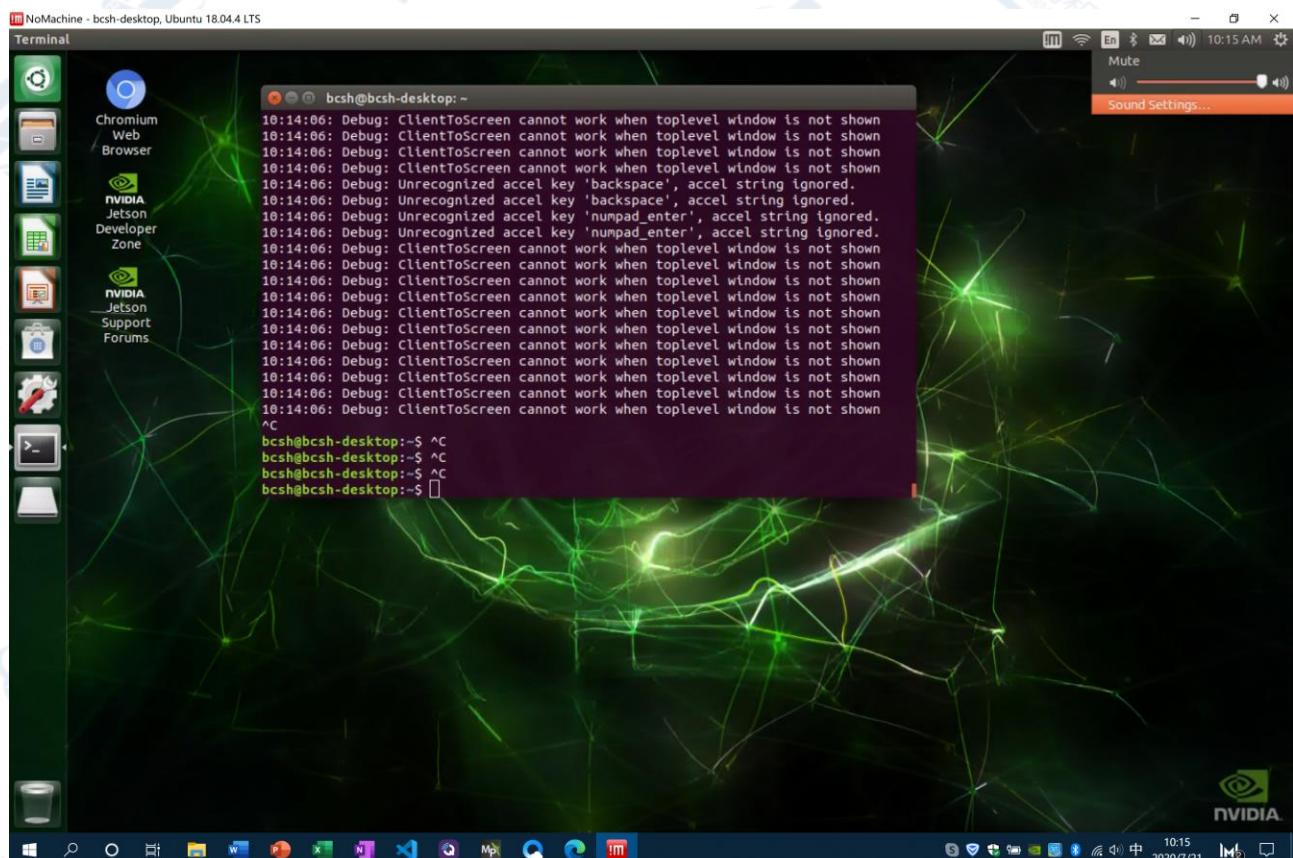
在智行 mini 的系统中，我们已经安装配置好了它的全部驱动。

18.4.2 语音数据采集

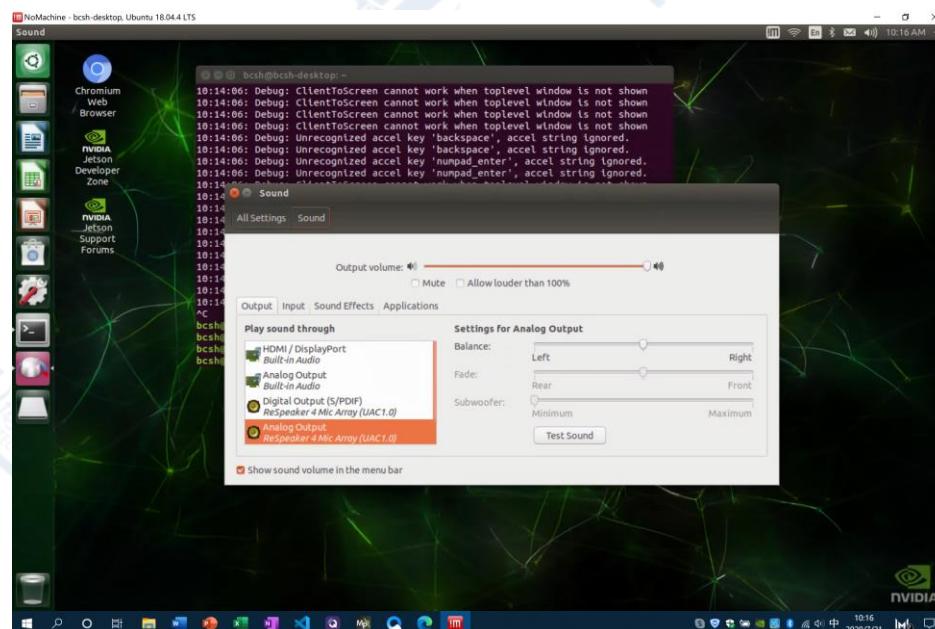
在智行 mini 系统中，我们安装了录音软件 audacity, 打开终端输入 audacity



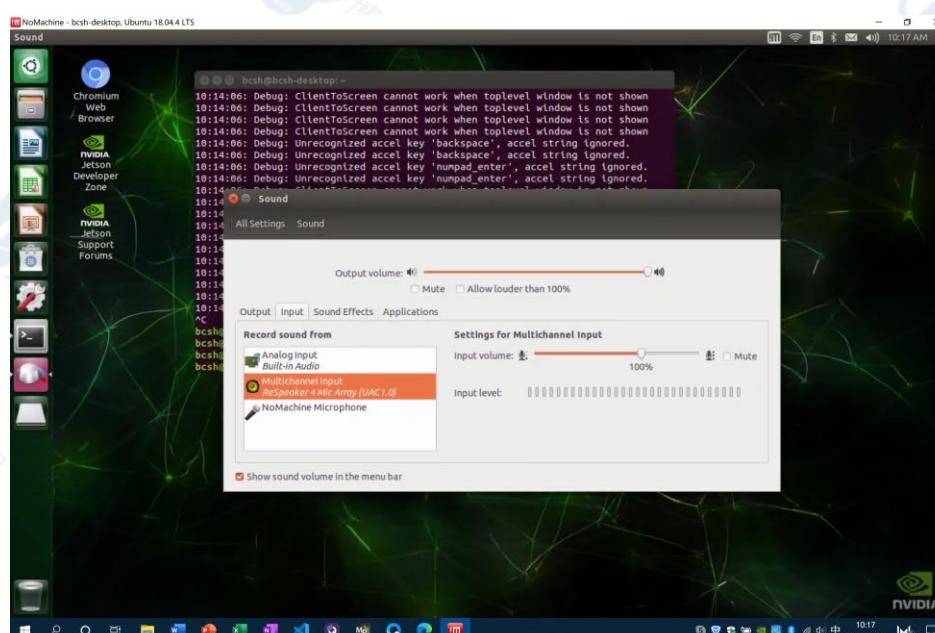
点击系统右上角音量按键，打开 sound settings



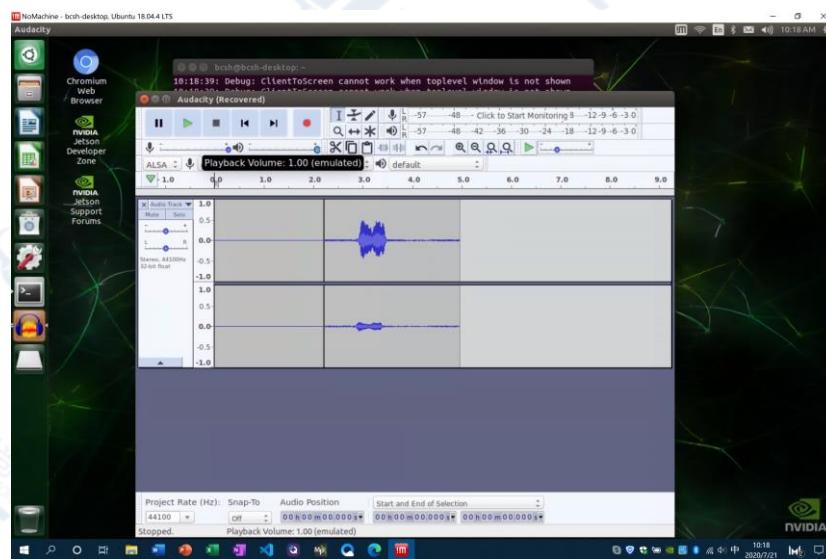
输出 output 选择语音阵列



输入 input 也选择语音阵列



点击 record 按钮录音



然后可以点击 play 播放录音，则证明麦克风与音箱驱动一切正常。

18.4.3 语音阵列数据采集

我们不仅可以通过语音板录音，还可以提取声源，控制 LED，接下来我们在 snowman 的 scripts 文件

夹内新建脚本 respeaker_interface.py，在这个脚本中输入以下内容：

```
#!/usr/bin/env python

import usb.core
import usb.util
import rospy
import time
import struct

try:
    from pixel_ring import usb_pixel_ring_v2
except IOError as e:
    print e
    raise RuntimeError("Check the device is connected and recognized")

PARAMETERS = {
    'AECFREEZEONOFF': (18, 7, 'int', 1, 0, 'rw', 'Adaptive Echo Canceler updates inhibit.', '0 = Adaptation enabled', '1 = Freeze adaptation, filter only'),
}
```

```
'AECNORM': (18, 19, 'float', 16, 0.25, 'rw', 'Limit on norm of AEC filter coefficients'),  
'AECPATHCHANGE': (18, 25, 'int', 1, 0, 'ro', 'AEC Path Change Detection.', '0 = false (no path change detected)', '1 = true (path change detected)'),  
'RT60': (18, 26, 'float', 0.9, 0.25, 'ro', 'Current RT60 estimate in seconds'),  
'HPFONOFF': (18, 27, 'int', 3, 0, 'rw', 'High-pass Filter on microphone signals.', '0 = OFF', '1 = ON - 70 Hz cut-off', '2 = ON - 125 Hz cut-off', '3 = ON - 180 Hz cut-off'),  
'RT60ONOFF': (18, 28, 'int', 1, 0, 'rw', 'RT60 Estimation for AES. 0 = OFF 1 = ON'),  
'AECSILENCELEVEL': (18, 30, 'float', 1, 1e-09, 'rw', 'Threshold for signal detection in AEC [-inf .. 0] dBov (Default: -80dBov = 10log10(1x10-8))'),  
'AECSILENCEMODE': (18, 31, 'int', 1, 0, 'ro', 'AEC far-end silence detection status. ', '0 = false (signal detected) ', '1 = true (silence detected)'),  
'AGCONOFF': (19, 0, 'int', 1, 0, 'rw', 'Automatic Gain Control. ', '0 = OFF ', '1 = ON'),  
'AGCMAXGAIN': (19, 1, 'float', 1000, 1, 'rw', 'Maximum AGC gain factor. ', '[0 .. 60] dB (default 30dB = 20log10(31.6))'),  
'AGCDESIREDELEVEL': (19, 2, 'float', 0.99, 1e-08, 'rw', 'Target power level of the output signal. ', '[-inf .. 0] dBov (default: -23dBov = 10log10(0.005))'),  
'AGCGAIN': (19, 3, 'float', 1000, 1, 'rw', 'Current AGC gain factor. ', '[0 .. 60] dB (default: 0.0dB = 20log10(1.0))'),  
'AGCTIME': (19, 4, 'float', 1, 0.1, 'rw', 'Ramps-up / down time-constant in seconds.'),  
'CNIONOFF': (19, 5, 'int', 1, 0, 'rw', 'Comfort Noise Insertion.', '0 = OFF', '1 = ON'),  
'FREEZEONOFF': (19, 6, 'int', 1, 0, 'rw', 'Adaptive beamformer updates.', '0 = Adaptation enabled', '1 = Freeze adaptation, filter only'),  
'STATNOISEONOFF': (19, 8, 'int', 1, 0, 'rw', 'Stationary noise suppression.', '0 = OFF', '1 = ON'),  
'GAMMA_NS': (19, 9, 'float', 3, 0, 'rw', 'Over-subtraction factor of stationary noise. min .. max attenuation'),  
'MIN_NS': (19, 10, 'float', 1, 0, 'rw', 'Gain-floor for stationary noise suppression.', '[-inf .. 0] dB (default: -16dB = 20log10(0.15))'),  
'NONSTATNOISEONOFF': (19, 11, 'int', 1, 0, 'rw', 'Non-stationary noise suppression.', '0 = OFF', '1 = ON'),  
'GAMMA_NN': (19, 12, 'float', 3, 0, 'rw', 'Over-subtraction factor of non- stationary noise. min .. max attenuation'),  
'MIN_NN': (19, 13, 'float', 1, 0, 'rw', 'Gain-floor for non-stationary noise suppression.', '[-inf .. 0] dB (default: -10dB = 20log10(0.3))'),
```

```
'ECHOONOFF': (19, 14, 'int', 1, 0, 'rw', 'Echo suppression.', '0 = OFF', '1 = ON'),  
'GAMMA_E': (19, 15, 'float', 3, 0, 'rw', 'Over-subtraction factor of echo (direct and early components). min .. max attenuation'),  
'GAMMA_ETAIL': (19, 16, 'float', 3, 0, 'rw', 'Over-subtraction factor of echo (tail components). min .. max attenuation'),  
'GAMMA_ENL': (19, 17, 'float', 5, 0, 'rw', 'Over-subtraction factor of non-linear echo. min .. max attenuation'),  
'NLATTENONOFF': (19, 18, 'int', 1, 0, 'rw', 'Non-Linear echo attenuation.', '0 = OFF', '1 = ON'),  
'NLAEC_MODE': (19, 20, 'int', 2, 0, 'rw', 'Non-Linear AEC training mode.', '0 = OFF', '1 = ON - phase 1', '2 = ON - phase 2'),  
'SPEECHDETECTED': (19, 22, 'int', 1, 0, 'ro', 'Speech detection status.', '0 = false (no speech detected)', '1 = true (speech detected)'),  
'FSBUPDATED': (19, 23, 'int', 1, 0, 'ro', 'FSB Update Decision.', '0 = false (FSB was not updated)', '1 = true (FSB was updated)'),  
'FSBPATHCHANGE': (19, 24, 'int', 1, 0, 'ro', 'FSB Path Change Detection.', '0 = false (no path change detected)', '1 = true (path change detected)'),  
'TRANSIENTONOFF': (19, 29, 'int', 1, 0, 'rw', 'Transient echo suppression.', '0 = OFF', '1 = ON'),  
'VOICEACTIVITY': (19, 32, 'int', 1, 0, 'ro', 'VAD voice activity status.', '0 = false (no voice activity)', '1 = true (voice activity)'),  
'STATNOISEONOFF_SR': (19, 33, 'int', 1, 0, 'rw', 'Stationary noise suppression for ASR.', '0 = OFF', '1 = ON'),  
'NONSTATNOISEONOFF_SR': (19, 34, 'int', 1, 0, 'rw', 'Non-stationary noise suppression for ASR.', '0 = OFF', '1 = ON'),  
'GAMMA_NS_SR': (19, 35, 'float', 3, 0, 'rw', 'Over-subtraction factor of stationary noise for ASR.', '[0.0 .. 3.0] (default: 1.0)'),  
'GAMMA_NN_SR': (19, 36, 'float', 3, 0, 'rw', 'Over-subtraction factor of non-stationary noise for ASR.', '[0.0 .. 3.0] (default: 1.1)'),  
'MIN_NS_SR': (19, 37, 'float', 1, 0, 'rw', 'Gain-floor for stationary noise suppression for ASR.', '[-inf .. 0] dB (default: -16dB = 20log10(0.15))),  
'MIN_NN_SR': (19, 38, 'float', 1, 0, 'rw', 'Gain-floor for non-stationary noise suppression for ASR.', '[-inf .. 0] dB (default: -10dB = 20log10(0.3))),  
'GAMMAVAD_SR': (19, 39, 'float', 1000, 0, 'rw', 'Set the threshold for voice activity detection.', '[-inf .. 60] dB (default: 3.5dB 20log10(1.5))),  
# 'KEYWORDDETECT': (20, 0, 'int', 1, 0, 'ro', 'Keyword detected. Current value so needs polling.'),  
'DOAANGLE': (21, 0, 'int', 359, 0, 'ro', 'DOA angle. Current value. Orientation depends on build configuration.)  
}
```

```
class RespeakerInterface(object):  
    VENDOR_ID = 0x2886  
    PRODUCT_ID = 0x0018  
    TIMEOUT = 100000  
  
    def __init__(self):  
        self.dev = usb.core.find(idVendor=self.VENDOR_ID,  
                               idProduct=self.PRODUCT_ID)  
        if not self.dev:  
            raise RuntimeError("Failed to find Respeaker device")  
        rospy.loginfo("Initializing Respeaker device")  
        self.dev.reset()  
        self.pixel_ring = usb_pixel_ring_v2.PixelRing(self.dev)  
        self.set_led_think()  
        time.sleep(5) # it will take 5 seconds to re-recognize as audio device  
        self.set_led_trace()  
        rospy.loginfo("Respeaker device initialized (Version: %s)" % self.version)  
  
    def __del__(self):  
        try:  
            self.close()  
        except:  
            pass  
        finally:  
            self.dev = None  
  
    def write(self, name, value):  
        try:  
            data = PARAMETERS[name]  
        except KeyError:  
            return  
        if data[5] == 'ro':  
            raise ValueError('{} is read-only'.format(name))  
        id = data[0]  
        # 4 bytes offset, 4 bytes value, 4 bytes type  
        if data[2] == 'int':  
            payload = struct.pack(b'iii', data[1], int(value), 1)  
        else:  
            payload = struct.pack(b'ifi', data[1], float(value), 0)  
        self.dev.ctrl_transfer(  
            usb.util CTRL_OUT | usb.util CTRL_TYPE_VENDOR | usb.util CTRL_RECIPIENT  
_DEVICE,  
            0, 0, id, payload, self.TIMEOUT)  
  
    def read(self, name):
```

```
try:
    data = PARAMETERS[name]
except KeyError:
    return
id = data[0]
cmd = 0x80 | data[1]
if data[2] == 'int':
    cmd |= 0x40
length = 8
response = self.dev.ctrl_transfer(
    usb.util CTRL_IN | usb.util CTRL_TYPE_VENDOR | usb.util CTRL_RECIPIENT_
DEVICE,
    0, cmd, id, length, self.TIMEOUT)
response = struct.unpack(b'ii', response.tostring())
if data[2] == 'int':
    result = response[0]
else:
    result = response[0] * (2.**response[1])
return result

def set_led_think(self):
    self.pixel_ring.set_brightness(10)
    self.pixel_ring.think()

def set_led_trace(self):
    self.pixel_ring.set_brightness(20)
    self.pixel_ring.trace()

def set_led_color(self, r, g, b, a):
    self.pixel_ring.set_brightness(int(20 * a))
    self.pixel_ring.set_color(r=int(r*255), g=int(g*255), b=int(b*255))

def set_vad_threshold(self, db):
    self.write('GAMMAVAD_SR', db)

def is_voice(self):
    return self.read('VOICEACTIVITY')

@property
def direction(self):
    return self.read('DOAANGLE')

@property
def version(self):
    return self.dev.ctrl_transfer(
```

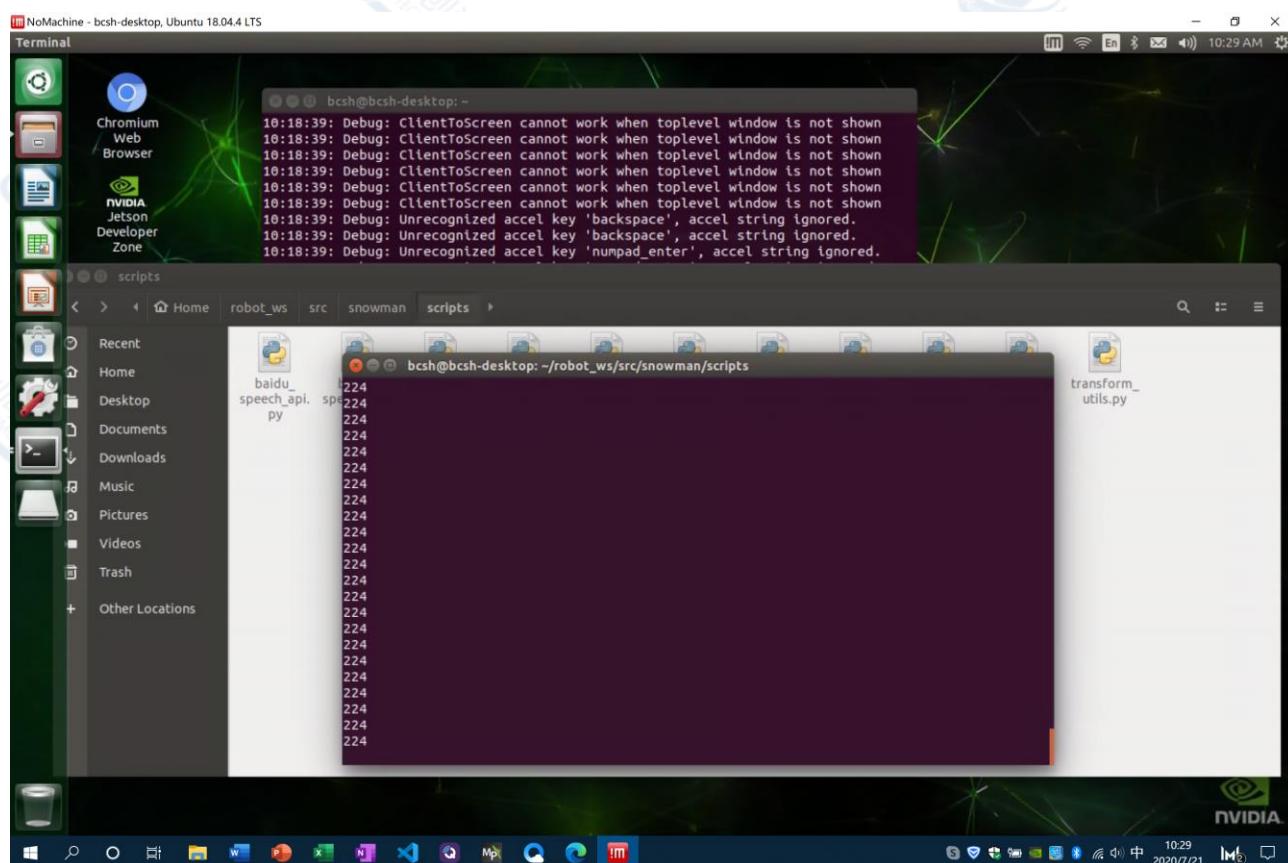
```
usb.util CTRL_IN | usb.util CTRL_TYPE_VENDOR | usb.util CTRL_RECIPIENT_DEVICE,
0, 0x80, 0, 1, self.TIMEOUT)[0]

def close(self):
    """
    close the interface
    """
    usb.util.dispose_resources(self.dev)

if __name__ == '__main__':
    audio_interface = RespeakerInterface()
    while True:
        direction = audio_interface.direction
        print(direction)
```

在这个脚本中，我们实现了语音驱动的数据接口以及 LED 灯控制，同学们可以读取 python 代码实现其他 LED 控制方式。我们来运行一下这个脚本

```
bcs@bcs-h-desktop:~/robot_ws/src/snowman/scripts$ python respeaker_interface.py
```



可以观察到语音板灯光变化，以及终端打印出来的拾取到的声音角度。

18.5 实验结果：

能够熟练使用 python 代码驱动语音阵列。

18.6 实验报告：

实验目的

实验要求

实验内容

实验总结

第十九章.基于百度 SDK 语音识别实验

19.1 实验目的:

基于语音阵列实现语音识别。

19.2 实验要求:

掌握百度语音识别 api，完成语音识别

19.3 实验工具:

个人电脑一台，智行 mini 及其配件

19.4 实验内容

在这个案例中，我们将介绍如何通过百度 sdk 实现语音识别。

19.4.1 百度语音 SDK

百度 AI 开放平台集成了一系列语音识别，语音播报技术，官方网站如下：

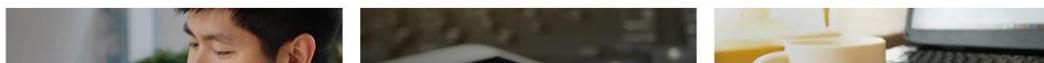
<https://ai.baidu.com/tech/speech?track=cp:ainsem|pf:pc|pp:chanpin-yuyin|pu:yuyin-yuyinshibie-pinpai|ci|kw:10003657>



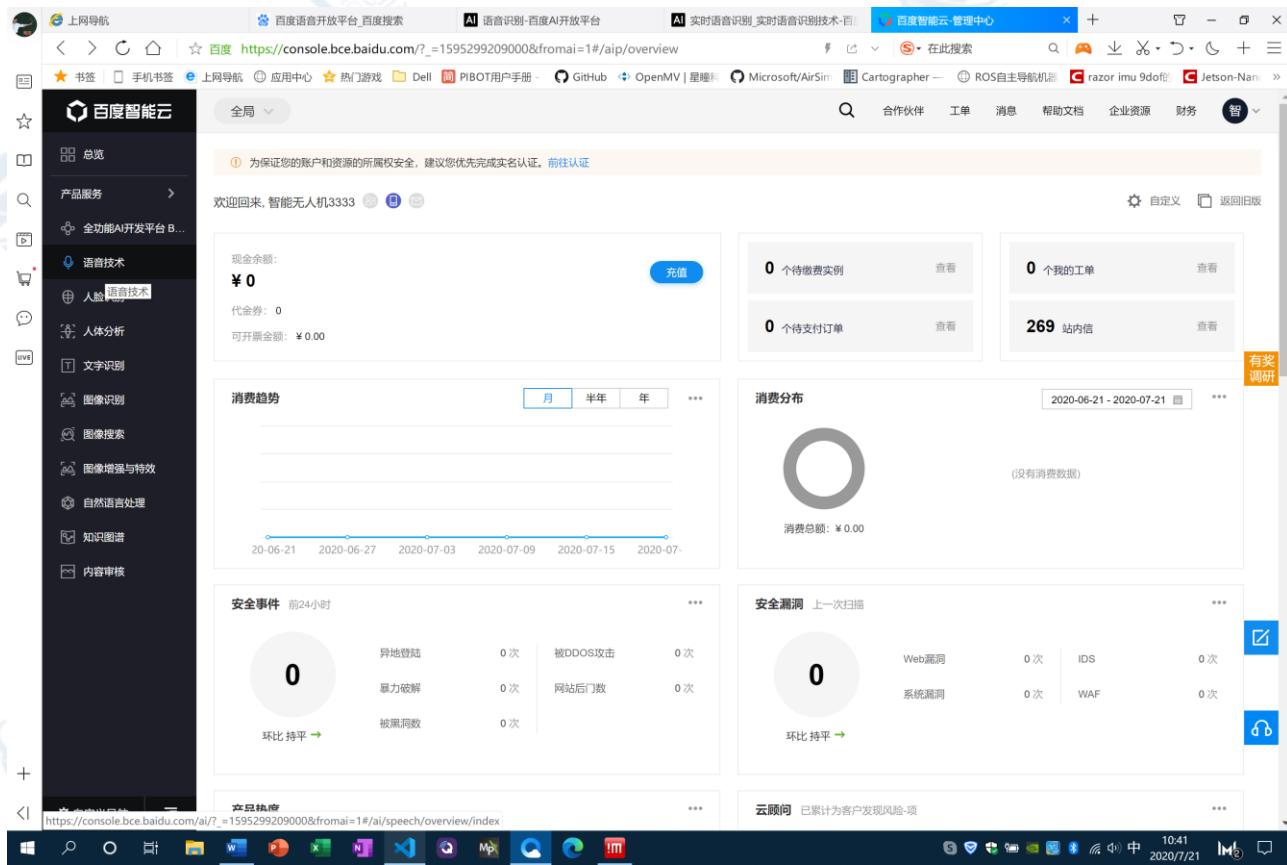
产品列表 应用场景 特色优势 产品定价 支持交流 相关推荐



产品列表

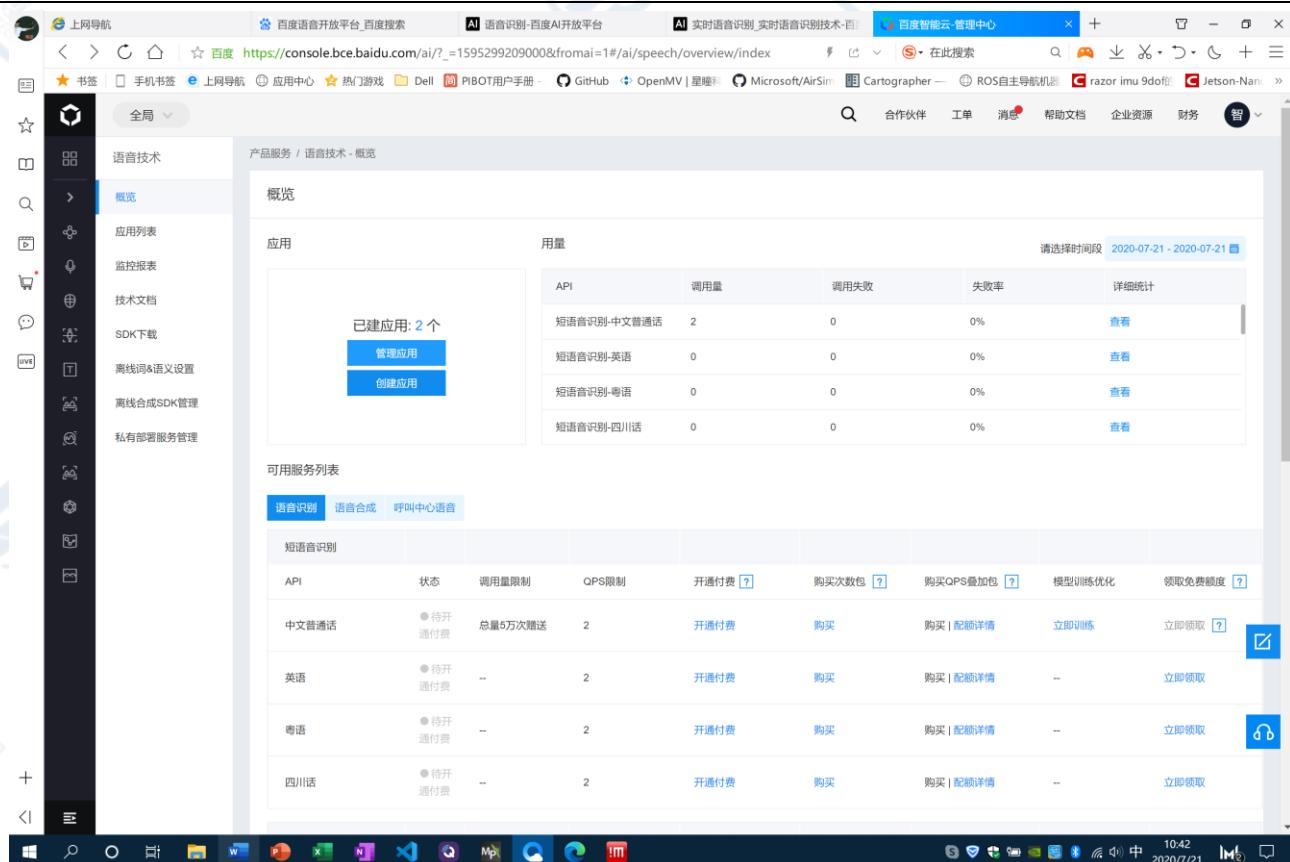


同学们可以自行注册自己的账号。
登陆后进入控制台点击语音技术



同学们可以自行注册自己的账号。
登陆后进入控制台点击语音技术

创建并管理自己的语音应用：



概览

应用

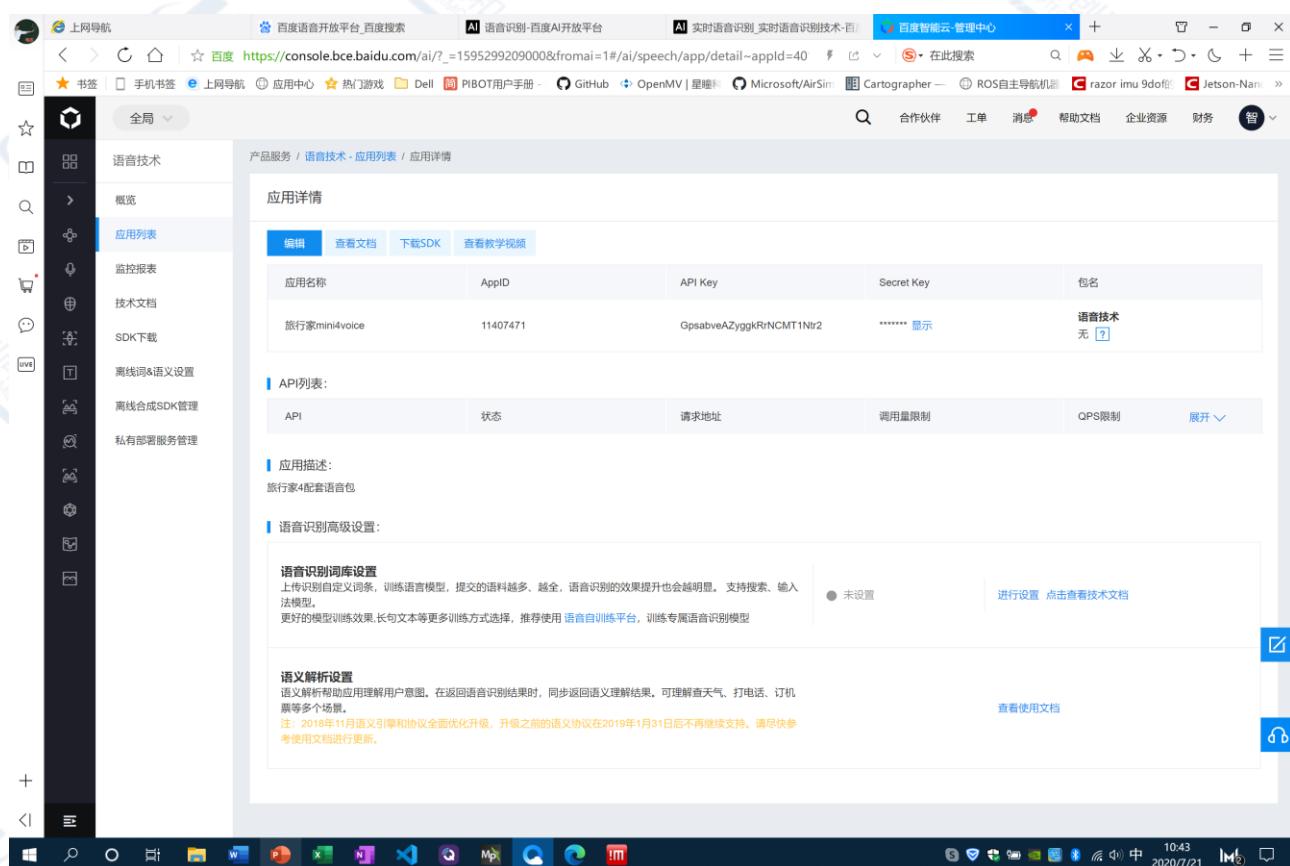
| API | 调用量 | 调用失败 | 失败率 | 详细统计 |
|-------------|-----|------|-----|--------------------|
| 短语音识别-中文普通话 | 2 | 0 | 0% | 查看 |
| 短语音识别-英语 | 0 | 0 | 0% | 查看 |
| 短语音识别-粤语 | 0 | 0 | 0% | 查看 |
| 短语音识别-四川话 | 0 | 0 | 0% | 查看 |

可用服务列表

语音识别

| API | 状态 | 调用量限制 | QPS限制 | 开通付费 | 购买次数包 | 购买QPS叠加包 | 模型训练优化 | 领取免费额度 |
|-------|-------|---------|-------|------|-------|-----------|--------|--------|
| 中文普通话 | 待开通付费 | 总量5万次赠送 | 2 | 开通付费 | 购买 | 购买 配购详情 | 立即训练 | 立即领取 |
| 英语 | 待开通付费 | -- | 2 | 开通付费 | 购买 | 购买 配购详情 | -- | 立即领取 |
| 粤语 | 待开通付费 | -- | 2 | 开通付费 | 购买 | 购买 配购详情 | -- | 立即领取 |
| 四川话 | 待开通付费 | -- | 2 | 开通付费 | 购买 | 购买 配购详情 | -- | 立即领取 |

这里是我们的应用账户



应用详情

API列表:

| API | 状态 | 请求地址 | 调用量限制 | QPS限制 | 展开 |
|---------------|----------|--------------------------|----------|-------|--------------------|
| 旅行家mini4voice | 11407471 | GpsabveAZyggkRrNCMT1Ntr2 | ***** 显示 | 语音技术无 | 展开 |

语音识别高级设置:

语音识别词库设置

上传识别自定义词条，训练语言模型，提交的语料越多、越全，语音识别的效果提升也会越明显。支持搜索、输入法模型。

更好的模型训练效果，长句文本等更多训练方式选择，推荐使用[语音自训练平台](#)，训练专属语音识别模型。

语义解析设置

语义解析帮助应用理解用户意图，在返回语音识别结果时，同步返回语义理解结果，可理解查天气、打电话、订机票等多个场景。

注：2018年11月语义引擎和协议全面优化升级，升级之前的语义协议在2019年1月31日后不再继续支持，请尽快参考使用文档进行更新。

大家记住 AppId, API Key, Secret Key 这三个 ID，可以在程序中替换为自己的。

19.4.2 基于百度 SDK 的语音识别

在 snowman 的 scripts 中创建脚本 respeaker_audio.py，输入以下代码

```
#!/usr/bin/env python

import pyaudio
from baidu_speech_api import BaiduVoiceApi
import json
import sys
import os
from aip.speech import AipSpeech
from contextlib import contextmanager

reload(sys)
sys.setdefaultencoding( "utf-8" )

CHUNK = 1024
RECORD_SECONDS = 5
APP_ID = '11407471'
API_KEY = 'GpsabveAZyggkRrNCMT1Ntr2'
SECRET_KEY = 'm3XhjR3T9F3eM9257Euc4W1SfHZaddxh'

@contextmanager
def ignore_stderr(enable=True):
    if enable:
        devnull = None
        try:
            devnull = os.open(os.devnull, os.O_WRONLY)
            stderr = os.dup(2)
            sys.stderr.flush()
            os.dup2(devnull, 2)
            try:
                yield
            finally:
                os.dup2(stderr, 2)
                os.close(stderr)
        finally:
            if devnull is not None:
                os.close(devnull)
    else:
        yield

class RespeakerAudio(object):
    def __init__(self, channel=0, supress_error=True):
        with ignore_stderr(enable=supress_error):
```

```
        self.pyaudio = pyaudio.PyAudio()
        self.channels = None
        self.channel = channel
        self.device_index = None
        self.rate = 16000
        self.bitwidth = 2
        self.bitdepth = 16
        #####find device#####
        count = self.pyaudio.get_device_count()
        for i in range(count):
            info = self.pyaudio.get_device_info_by_index(i)
            name = info["name"].encode("utf-8")
            chan = info["maxInputChannels"]
            if name.lower().find("respeaker") >= 0:
                self.channels = chan
                self.device_index = i
                break
        if self.device_index is None:
            info = self.pyaudio.get_default_input_device_info()
            self.channels = info["maxInputChannels"]
            self.device_index = info["index"]
        self.channel = min(self.channels - 1, max(0, self.channel))
        self.stream = self.pyaudio.open(
            rate = self.rate,
            format=self.pyaudio.get_format_from_width(self.bitwidth),
            channels=1,
            input=True,
            input_device_index=self.device_index,
        )
        self.aipSpeech = AipSpeech(APP_ID, API_KEY, SECRET_KEY)
        self.baidu = BaiduVoiceApi(appkey=API_KEY, secretkey=SECRET_KEY)

    def stop(self):
        self.stop()
        try:
            self.stream.stop_stream()
            self.stream.close()
        except:
            pass
        finally:
            self.stream = None
        try:
            self.pyaudio.terminate()
        except:
            pass
```

```
def generator_list(self, list):
    for l in list:
        yield l

def record(self):
    self.stream.start_stream()
    print("* recording")
    frames = []
    for i in range(0, int(self.rate/CHUNK*RECORD_SECONDS)):
        data = self.stream.read(CHUNK)
        frames.append(data)
    print("done recording")
    self.stream.stop_stream()
    print("start to send to baidu")
    text = self.baidu.server_api(self.generator_list(frames))
    #print (text)
    if text:
        try:
            text = json.loads(text)
            for t in text['result']:
                print(t)
                #self.voice_publisher.publish(str(text))
            return(str(t))
        except KeyError:
            return("get nothing")
    else:
        print("get nothing")
        return("get nothing")

if __name__ == '__main__':
    snowman_audio = RespeakerAudio()
    while True:
        text = snowman_audio.record()
        #print (text)
```

通过以上代码，我们实现了基于百度 SDK 与远场语音阵列的语音识别技术，这里

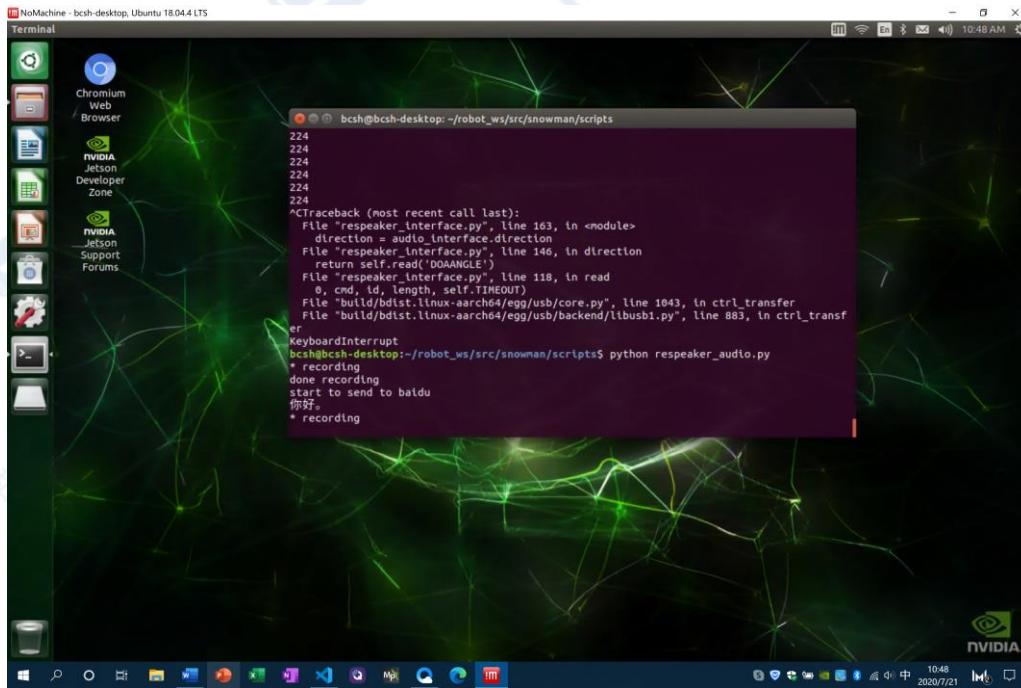
```
APP_ID = '11407471'
API_KEY = 'GpsabveAZyggkRrNCMT1Ntr2'
SECRET_KEY = 'm3XhjR3T9F3eM9257Euc4W1sfHZaddxh'
```

的三个 ID 可以替换成你们自己账户的，调用更加自由。

然后我们运行一下这个脚本

```
bcsh@bcsh-desktop:~/robot_ws/src/snowman/scripts$ python respeaker_audio.py
```

出现 recording 的时候说话，可以看到实现了语音识别。



19.5 实验结果：

能够熟练使用百度 SDK 实现语音识别。

19.6 实验报告：

实验目的

实验要求

实验内容

实验总结

第二十章.语音控制实验

20.1 实验目的:

基于语音阵列实现语音识别并通过语音控制实现智行 Mini 控制。

20.2 实验要求:

掌握百度语音识别 api，完成语音识别并控制智行 Mini 运动

20.3 实验工具:

个人电脑一台，智行 mini 及其配件

20.4 实验内容

在这个案例中，我们将介绍如何通过百度 sdk 实现语音识别并控制智行 mini 运动。

首先我们在 scripts 文件夹内编写 snowmannode.py 文件，输入以下代码

```
#!/usr/bin/env python
#coding:utf-8

import rospy
import string
import sys
from respeaker_interface import RespeakerInterface
from respeaker_audio import RespeakerAudio
from std_msgs.msg import String
from geometry_msgs.msg import Twist

reload(sys)
sys.setdefaultencoding( "utf-8" )

class SnowManNode(object):
    def __init__(self, node_name):
        self.node_name = node_name
        rospy.init_node(node_name)
        rospy.on_shutdown(self.shutdown)
        self.respeaker_interface = RespeakerInterface()
        self.respeaker_audio = RespeakerAudio()
        self.ask_pub = rospy.Publisher('/snowman/ask', String, queue_size=5)
        self.cmd_pub = rospy.Publisher('/cmd_vel', Twist, queue_size=5)
```

```
self.rate = rospy.Rate(100)
while not rospy.is_shutdown():
    text = self.respeaker_audio.record()
    if text.find("前进") >= 0:
        print("forward")
        cmd_msg = Twist()
        cmd_msg.linear.x = 0.3
        cmd_msg.angular.z = 0.0
        self.cmd_pub.publish(cmd_msg)
    elif text.find("后退") >= 0:
        print("backward")
        cmd_msg = Twist()
        cmd_msg.linear.x = -0.3
        cmd_msg.angular.z = 0.0
        self.cmd_pub.publish(cmd_msg)
    elif text.find("左转") >= 0:
        print("left")
        cmd_msg = Twist()
        cmd_msg.linear.x = 0.0
        cmd_msg.angular.z = 0.3
        self.cmd_pub.publish(cmd_msg)
    elif text.find("右转") >= 0:
        print("right")
        cmd_msg = Twist()
        cmd_msg.linear.x = 0.0
        cmd_msg.angular.z = -0.3
        self.cmd_pub.publish(cmd_msg)
    print(text)
    self.rate.sleep()

def shutdown(self):
    self.cmd_pub.publish(Twist())
    self.respeaker_interface.close()
    self.respeaker_audio.stop()

if __name__ == '__main__':
    snowmanNode = SnowManNode("snowman_node")
```

可以看到，在这个脚本文件中，我们生成了一个节点，在这个节点中包含了之前两节介绍的语音驱动

和语音识别类，通过语音识别的结果，若包含“前进”，则智行 mini 向前运动，同理还有后退，左转，右转。同学们可以先打开终端输入 `roslaunch zoo_robot robot_lidar.launch` 启动底盘，然后输入 `roslaunch snowman mini4_listen.launch` 启动语音控制，然后等出现 record 后说出指令词，观察智行 mini 运动。

20.5 实验结果：

能够熟练使用百度 SDK 实现语音识别与语音控制。

20.6 实验报告：

实验目的

实验要求

实验内容

实验总结

第二十一章.语音播报实验

21.1 实验目的:

基于语音阵列实现语音合成与语音播报。

21.2 实验要求:

掌握百度语音播报 api，完成语音合成与播报

21.3 实验工具:

个人电脑一台，智行 mini 及其配件

21.4 实验内容

在 snowman 的 scripts 文件夹内新建 mini4_talk.py,输入以下代码

```
#!/usr/bin/env python
#coding:utf-8
from aip.speech import AipSpeech
import os
import rospy

APP_ID = '11407471'
API_KEY = 'GpsabveAZyggkRrNCMT1Ntr2'
SECRET_KEY = 'm3XhjR3T9F3eM9257Euc4W1SfHZaddxh'

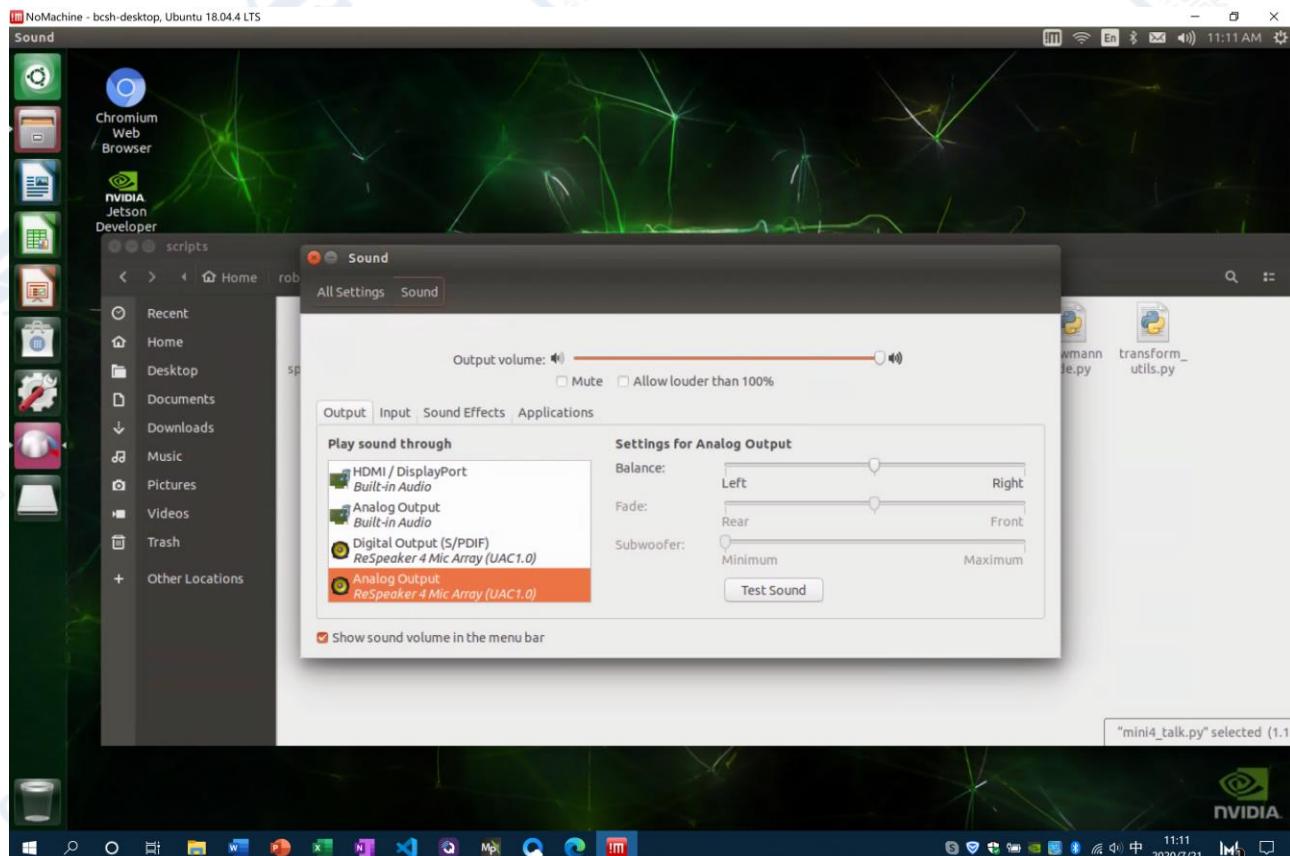
aipSpeech = AipSpeech(APP_ID, API_KEY, SECRET_KEY)

string = ' 北京博创尚和科技有限公司是国家高新技术企业，核心技术团队由北航机器人研究所、中科院自动化所毕业的博士、硕士组成。深耕教育领域多年，以人工智能和机器人技术为核心，为各类高校和中小学提供智能机器人产品、实验室解决方案及课程服务，已与全国 600 余所高校和大量中小学在科研、教学、工程实践、机器人竞赛、创客空间建设等方面合作，得到了业内人士的普遍认可和好评。'
rospy.init_node('mini4_talk', anonymous=True)
result = aipSpeech.synthesis(string, 'zh', 1, {
    'vol': 5, 'per': 5,
})

# 识别正确返回语音二进制 错误则返回 dict 参照下面错误码
```

```
if not isinstance(result, dict):
    with open('faster.mp3', 'wb') as f:
        f.write(result)
    os.system("mpg123 faster.mp3")
```

然后注意 sound settings 中 output 选择语音板



通过以下方式运行此程序：

```
bcsh@bcsh-desktop:~/robot_ws/src/snowman/scripts$ python mini4_talk.py
```

可以听到音箱中播报出了我们这段文字，语音合成同样是使用了百度 api，同学们可以自行替换自己的 ID。也可以修改上述代码完成更多应用。

21.5 实验结果：

能够熟练使用百度 SDK 实现语音合成。

21.6 实验报告：

实验目的

实验要求

实验内容

实验总结

第二十二章. 统一部件组仿人视觉对抗 B 比赛例程

22.1 实验目的:

融合智行 mini 大部分实验形成一个基于导航，语音，视觉的的例程。

22.2 实验要求:

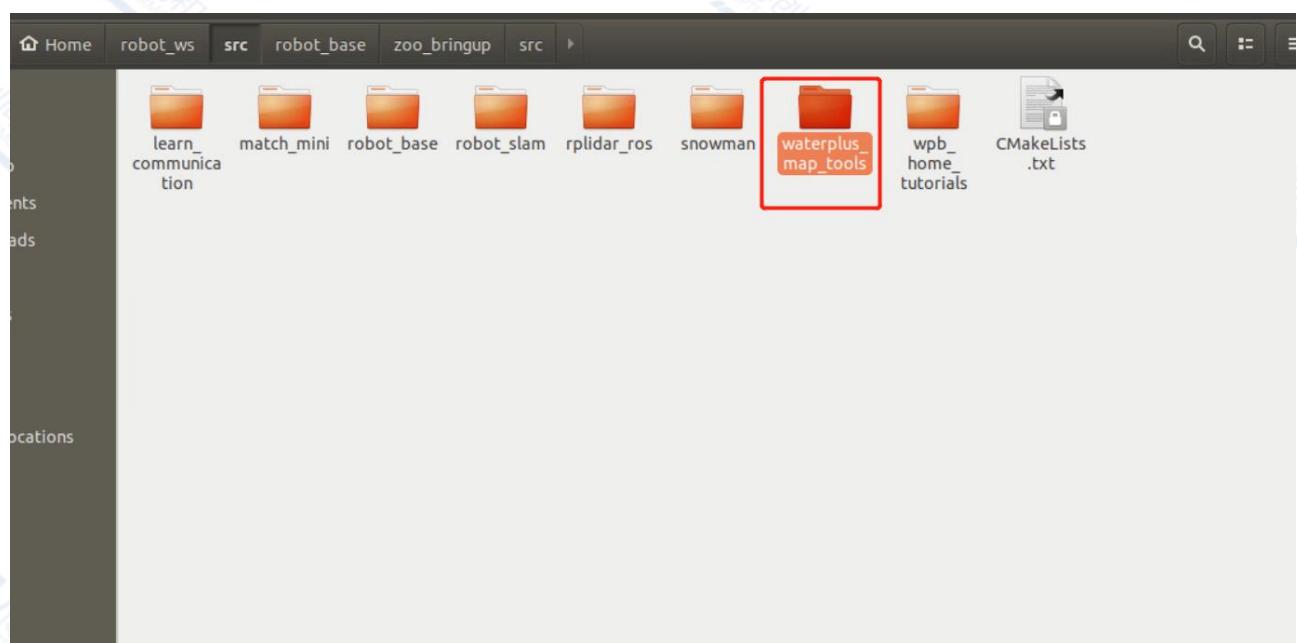
掌握语音控制智行 mini 进行自主导航
掌握人脸识别后对人脸进行判断并追踪

22.3 实验工具:

个人电脑一台，智行 mini 及其配件

22.4 实验内容

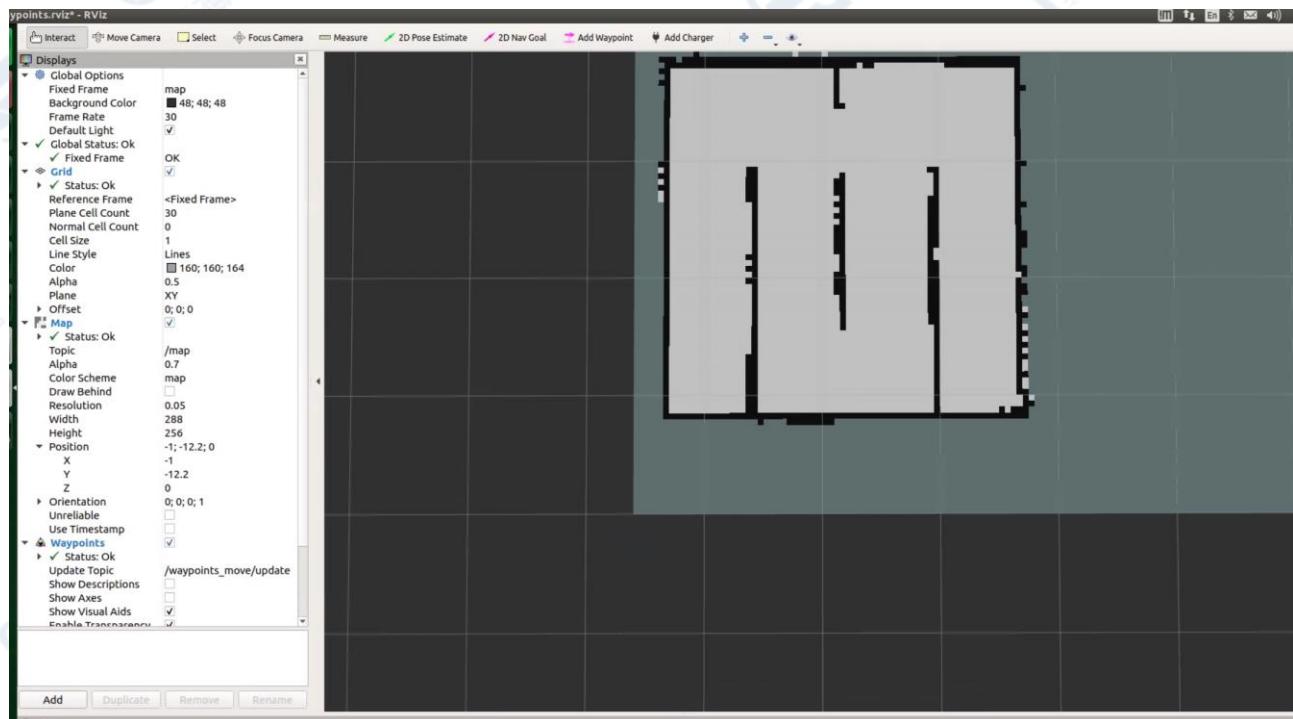
在开始实验之前，这里先介绍一下此次实验用到的 Rviz 插件 waterplus_map_tools，它可以在我们建完的地图上进行航点标注，那么之后的自主导航全部依靠于这些我们标注过的航点来实验智行 mini 在语音控制下的自主导航，如图，这里我们已经把功能包放在了 robot_ws 这个工程目录下：



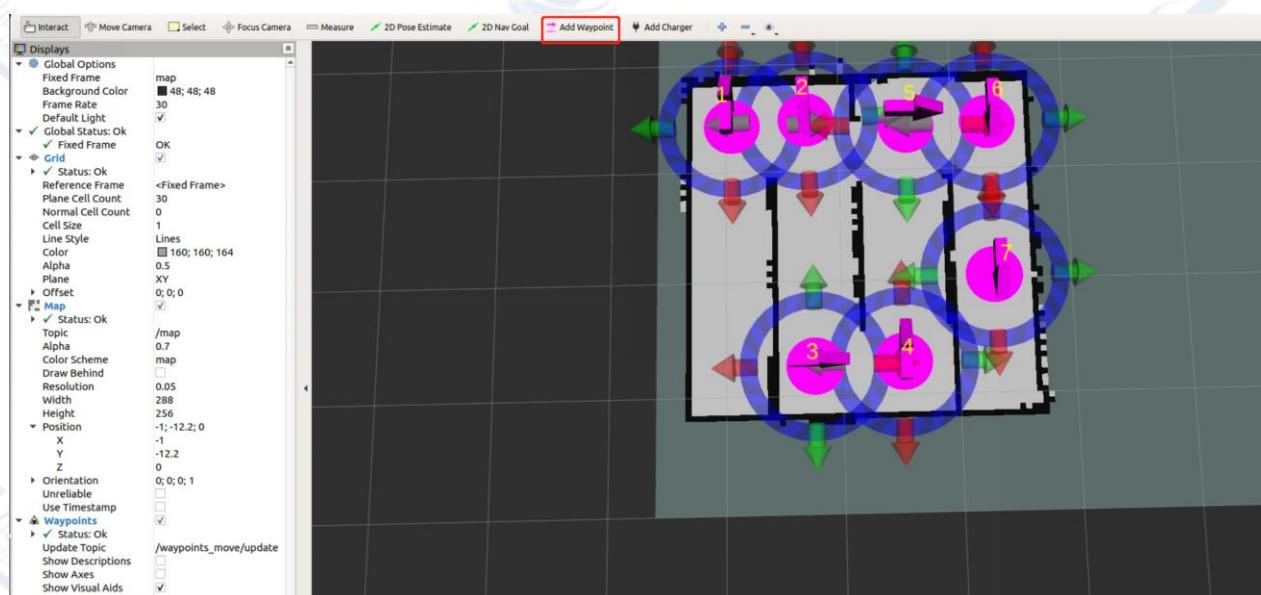
首先第一步，就是我们的 gmapping 建图，这个操作跟上述实验中的操作是一致的，这里不做讲解，同学们可以根据上述实验第九章 Gmapping 建图即可完成这一步。

接下来呢，就是运用我们上述提到的 Rviz 插件 waterplus_map_tools 进行航点标注，运行下面命令打开我们刚建好的图：

```
bcsh@bcsh-desktop: ~
bcsh@bcsh-desktop:~$ rosrun waterplus_map_tools add_waypoint.launch
```

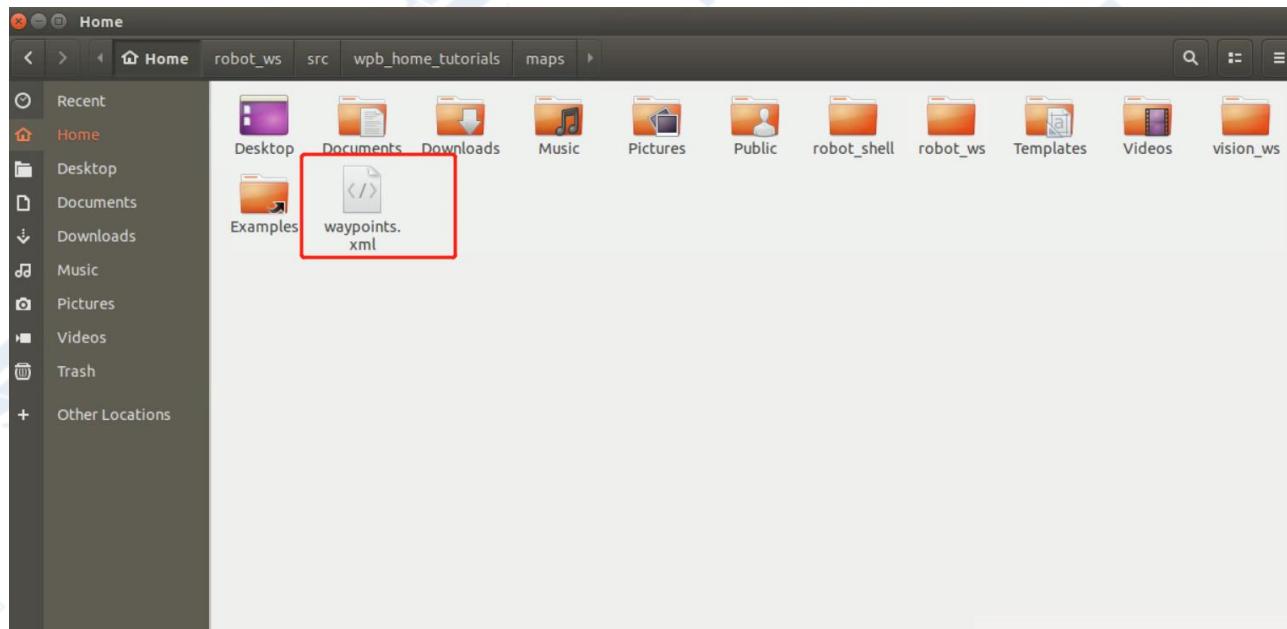


上图我们可以看到，Rviz 加载了我们刚才建好的地图，点击 Add Waypoint 按钮，按照起点到终点的路劲顺序标注。

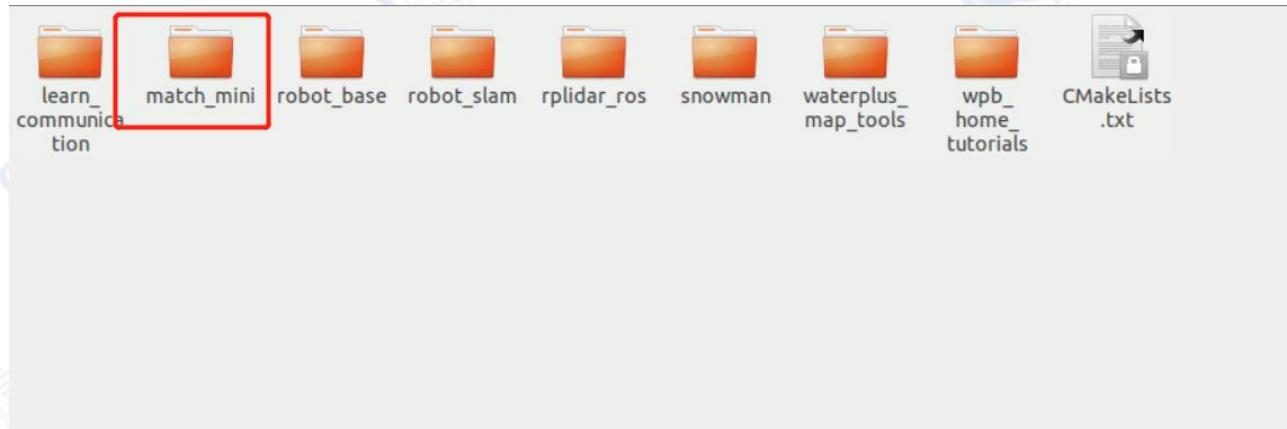


标注完成之后，运行如下命令，即可看到将已标注好的航点信息保存到 home/waypoint.xml 中

```
bcsh@bcsh-desktop:~$ rosrun waterplus_map_tools wp_saver
```



航点标注完成之后，就是编写自主导航程序了，大家可以看到，在我们的 robot_ws 目录里我们已经建立了一个 match_mini 的 ros 功能包，这个里就是单独放的基于比赛的代码



在 match_mini/script 目录下建立 auto_slam.py 的 python 脚本，写入如下代码：

```
#!/usr/bin/env python
import rospy

import actionlib
import roslaunch
from actionlib_msgs.msg import *
from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal
from nav_msgs.msg import Path
from std_msgs.msg import String
from geometry_msgs.msg import PoseWithCovarianceStamped
from tf_conversions import transformations
from xml.dom.minidom import parse
```

```
from math import pi
import tf

class navigation_demo:
    def __init__(self):
        self.set_pose_pub = rospy.Publisher('/initialpose',
        PoseWithCovarianceStamped, queue_size=5)

        self.move_base = actionlib.SimpleActionClient("move_base",
MoveBaseAction)
        self.move_base.wait_for_server(rospy.Duration(60))

        self.tf_listener = tf.TransformListener()
        self.get_point = rospy.Publisher('get_pos', String, queue_size=5)

        self.plist = []
        self.success_count = 0

    def set plist(self, plist):
        self.plist = plist

    def set_pose(self, p):
        if self.move_base is None:
            return False

        x, y, th = p

        pose = PoseWithCovarianceStamped()
        pose.header.stamp = rospy.Time.now()
        pose.header.frame_id = 'map'
        pose.pose.pose.position.x = x
        pose.pose.pose.position.y = y
        q = transformations.quaternion_from_euler(0.0, 0.0, th / 180.0 * pi)
        pose.pose.pose.orientation.x = q[0]
        pose.pose.pose.orientation.y = q[1]
        pose.pose.pose.orientation.z = q[2]
        pose.pose.pose.orientation.w = q[3]

        self.set_pose_pub.publish(pose)
        return True

    def _done_cb(self, status, result):
        rospy.loginfo("navigation done! status:%d result:%s" % (status, result))
```

```
def _active_cb(self):
    rospy.loginfo("[Navi] navigation has been activated")

def _feedback_cb(self, feedback):
    rospy.loginfo("[Navi] navigation feedback\r\n%s" % feedback)

def goto(self, p):
    goal = MoveBaseGoal()

    goal.target_pose.header.frame_id = 'map'
    goal.target_pose.header.stamp = rospy.Time.now()
    goal.target_pose.pose.position.x = p[0]
    goal.target_pose.pose.position.y = p[1]
    goal.target_pose.pose.position.z = p[2]
    # q = transformations.quaternion_from_euler(0.0, 0.0, p[2]/180.0*pi)
    goal.target_pose.pose.orientation.x = p[3]
    goal.target_pose.pose.orientation.y = p[4]
    goal.target_pose.pose.orientation.z = p[5]
    goal.target_pose.pose.orientation.w = p[6]

    self.move_base.send_goal(goal, self._done_cb, self._active_cb,
                           self._feedback_cb)

    result = self.move_base.wait_for_result(rospy.Duration(60))
    print(result)

    state = self.move_base.get_state()
    if state == GoalStatus.SUCCEEDED:
        self.success_count += 1
        if len(self.plist) == self.success_count:
            rospy.loginfo("arrived goal point")
            self.get_point.publish("1")
            self.isSendVoice = False
    return True

def cancel(self):
    self.move_base.cancel_all_goals()
    return True

def callback(msg):
    doc = parse("/home/bcsh/waypoints.xml")
    root_element = doc.documentElement
    points = root_element.getElementsByTagName("Waypoint")
```

```
plist = []

rospy.loginfo("set pose...")
navi = navigation_demo()

for p in points:
    point = [0] * 7
    point[0] = float(p.getElementsByTagName("Pos_x")[0].childNodes[0].data)
    point[1] = float(p.getElementsByTagName("Pos_y")[0].childNodes[0].data)
    point[2] = float(p.getElementsByTagName("Pos_z")[0].childNodes[0].data)
    point[3] = float(p.getElementsByTagName("Ori_x")[0].childNodes[0].data)
    point[4] = float(p.getElementsByTagName("Ori_y")[0].childNodes[0].data)
    point[5] = float(p.getElementsByTagName("Ori_z")[0].childNodes[0].data)
    point[6] = float(p.getElementsByTagName("Ori_w")[0].childNodes[0].data)
    plist.append(point)

print(plist)

rospy.loginfo("goto goal...")
navi.set_plist(plist)

for waypoint in plist:
    # print(waypoint)
    navi.goto(waypoint)

if __name__ == "__main__":
    rospy.init_node('auto_slam_node', anonymous=True)
    rospy.Subscriber("auto_slam", String, callback)

    rospy.spin()
    r = rospy.Rate(0.2)
    r.sleep()
```

上述代码中我们可以看到，在主函数中我们订阅了一个 `auto_slam` 的话题，并接收其消息，在 `callback` 这个回调函数中，当我们接收到来自于 `auto_slam` 的消息后，读取刚刚标注好的航点文件 `waypoint.xml`，将其中的航点信息读取出来然后遍历执行导航到每一个航点，直到智行 mini 导航到最后一个航点。那我们可以看到在 `goto` 这个函数中，记录每一次航点状态的回调，到最后一个航点的时候，向 `get_pos` 这个话题发送一个消息，使其播放语音：我已到位，等待命令，实施抓捕（语音控制下面会讲到）

```
def goto(self, p):
    goal = MoveBaseGoal()

    goal.target_pose.header.frame_id = 'map'
    goal.target_pose.header.stamp = rospy.Time.now()
    goal.target_pose.pose.position.x = p[0]
    goal.target_pose.pose.position.y = p[1]
    goal.target_pose.pose.position.z = p[2]
    # q = transformations.quaternion_from_euler(0.0, 0.0, p[2]/180.0*pi)
    goal.target_pose.pose.orientation.x = p[3]
    goal.target_pose.pose.orientation.y = p[4]
    goal.target_pose.pose.orientation.z = p[5]
    goal.target_pose.pose.orientation.w = p[6]

    self.move_base.send_goal(goal, self._done_cb, self._active_cb, self._feedback_cb)

    result = self.move_base.wait_for_result(rospy.Duration(60))
    print(result)

    state = self.move_base.get_state()
    if state == GoalStatus.SUCCEEDED:
        self.success_count += 1
        if len(self.plist) == self.success_count:
            rospy.loginfo("arrived goal point")
            self.get_point.publish("1")
            self.isSendVoice = False
    return True
```

自主导航这一部分，我们就完成了，那么接下来介绍一下手动导航以及语音播报是如何使用的。

首先，我们在 `match_mini/src` 目录下建立一个 `nav_result_callback.cpp` 文件，用于监听手动导航后到达目标点的一个状态，然后向 `get_pos` 节点发送消息，使其播报语音：我已到位，等待命令，实施抓捕。

```
# include <ros/ros.h>
# include <std_msgs/String.h>
# include <sstream>
# include <string>

# include <codecvt>
# include <move_base_msgs/MoveBaseAction.h>
# include <move_base_msgs/MoveBaseGoal.h>

ros::Publisher pointState;
int play_state = 0;
void state_callback(const move_base_msgs::MoveBaseActionResult & msg)
{
    std::cout << "status is " << msg.status.status << std::endl;
```

```
if (msg.status.status == 3)
{
    std::cout << "the goal was achieved successfully!" << std::endl;

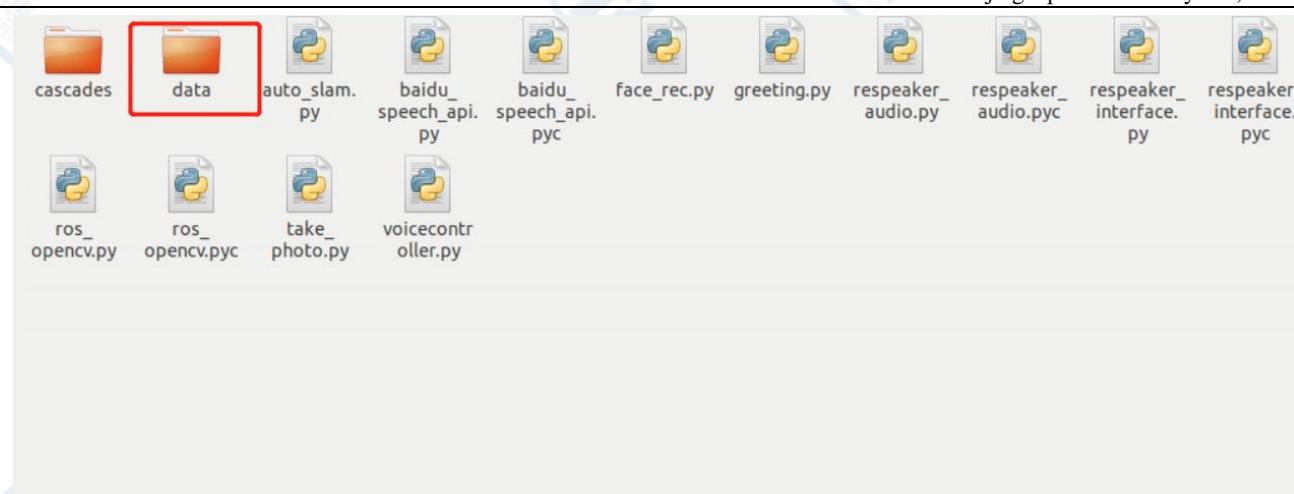
    std_msgs::String msg;
    std::stringstream ss;
    ss << "1";
    msg.data = ss.str();
    if(play_state == 0)
    {
        play_state = 1;
        pointState.publish(msg);
    }
}

int main(int argc, char ** argv)
{
    ros::init(argc, argv, "nav_result_callback");
    ros::NodeHandle n;

    ros::Subscriber goal_sub = n.subscribe("move_base/result", 10, state_callback);
    pointState = n.advertise< std_msgs::String > ("/get_pos", 10);

    ros::spin();
    return 0;
}
```

至此，导航部分完成，接下来我们进行人脸训练识别的部分，上述基于视觉的实验中人脸识别部分的人脸采集与此一致，在实验过程中，我们还是沿用之前的实验进行人脸采集，采集完成之后把文件夹拷贝到比赛的功能包中：



接下来就是我们的人脸识别，这里我们打开 face_rec.py 文件：

```
def callback(msg):
    global face_path
    global face_name
    if msg.data == "liwei":
        face_name = "liwei"
    if msg.data == "yaom":
        face_name = "yaom"
    face_rec()

if __name__ == "__main__":
    rospy.init_node('face_detector')
    rospy.Subscriber("auto_face", String, callback)

    rospy.spin()
```

可以看到主函数中订阅了 auto_face 的话题，在接收到消息 msg 为一个人名字符串，来区分嫌疑犯一和嫌疑犯二，在识别到确定的嫌疑犯，通过发布 cmd_vel 消息，控制智行 mini 移动到嫌疑犯的前面

```
for (x, y, w, h) in faces:
    result = cv2.rectangle(result, (x, y), (x + w, y + h), (255, 0, 0), 2)
    roi = gray[x:x + w, y:y + h]
    try:
        roi = cv2.resize(roi, (200, 200), interpolation=cv2.INTER_LINEAR)
        [p_label, p_confidence] = model.predict(roi)
        cv2.putText(result, names[p_label], (x, y - 20), cv2.FONT_HERSHEY_SIMPLEX, 1, 255, 2)

        if p_confidence > 9500 and names[p_label] == face_name:

            cv2.putText(result, self.names[p_label], (x, y - 20), cv2.FONT_HERSHEY_SIMPLEX, 1, 255, 2)
            offset_x = ((x + w) / 2 - 240)
            target_area = w * h
            linear_vel = 0
            angular_vel = 0

            print(target_area)
            if target_area < 100:
                linear_vel = 0.0
            elif target_area > 110:
                linear_vel = 0.1
            else:
                linear_vel = 0.0

            if offset_x > 0:
                angular_vel = -0.1

            if offset_x < 0:
                angular_vel = 0.1
            update_cmd(linear_vel, angular_vel)
```

最后就是我们的语音控制，可以看到，在 match_mini/script 目录下有 voice_controller.py 文件：

```
class VoiceController(object):
    def __init__(self, node_name):
        self.node_name = node_name
        rospy.init_node(node_name)
        rospy.on_shutdown(self.shutdown)
        self.respeaker_interface = RespeakerInterface()
        self.respeaker_audio = RespeakerAudio()
        self.ask_pub = rospy.Publisher('cmd_msg', String, queue_size=5)

    def shutdown(self):
        self.respeaker_interface.close()
        self.respeaker_audio.stop()

def callback(msg):
    os.system("mpg123 /home/bcsh/robot_ws/src/match_mini/voice/zhuabu.mp3")

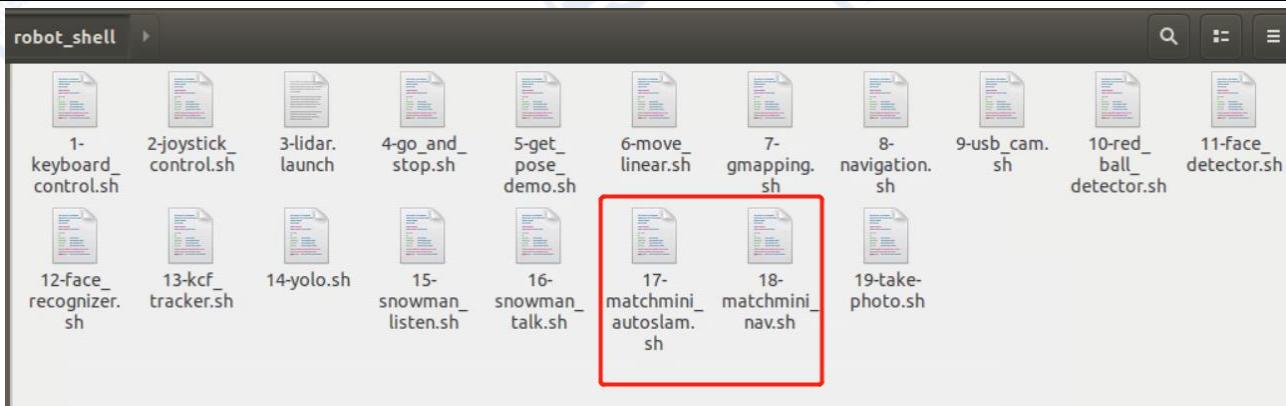
if __name__ == '__main__':
    voice_controller = VoiceController("voice_controller")
    auto_slam = rospy.Publisher('auto_slam', String, queue_size=10) # 话题的名称chatter
    auto_face = rospy.Publisher('auto_face', String, queue_size=10) # 话题的名称chatter
    rospy.Subscriber("get_pos", String, callback, queue_size=10)
    rate = rospy.Rate(100)

    isPub = False
    while not rospy.is_shutdown():
        text = voice_controller.respeaker_audio.record()
        if text.find("开始导航") >= 0 and isPub is not True:
            auto_slam.publish("start")
            isPub = True
        if text.find("抓") >= 0:
            print("send liwei to auto_face")
            auto_face.publish("liwei")
        direction = voice_controller.respeaker_interface.direction
        print(text)
        print(direction)
        rate.sleep()
```

上述代码中，在主函数中创建了 2 个话题，即接受到语音指令开始导航，以及接受到语音指令进行抓捕（这个同学们可以自由发挥，可以通过不同的方式区分和控制抓捕嫌疑人，这里的指令只做师范，具体同学们在调试的过程中更改语音指令），以及一个话题订阅，接收 get_pos 发来的消息，接收到消息后播报语音。

至此，统一部件组仿人视觉对抗 B 比赛例程给大家介绍完了，样例仅供参考。

在这里为大家写好了 2 个可执行文件，运行即可执行自主导航和手动导航



22.5 实验结果：

能够根据规则，在规定的场地里进行通过语音控制进行自动导航以及人脸识别跟踪

22.6 实验报告：

实验目的

实验要求

实验内容

实验总结

附录一.Ubuntu 系统下串口绑定

如何查看端口号？

```
ls /dev/tty*
```

为什么要绑定端口号？

当机载电脑连接外部设备大于等于两个时，由于插拔顺序（上电顺序）不同会导致同一设备的端口号与之前不一致，从而导致在程序内在进行读取端口操作时会出错。

绑定端口号的方法：

本方法根据设备所插入的 USB 端口路径进行绑定，可以解决上述问题。但是，按照本方法绑定设备后，不可再更改设备的 USB 端口，因此需要一直插入相同的 USB 端口。

1、查询设备路径，红色线表示的 ID_PATH 就是该设备的路径

```
nvidia@tegra-ubuntu:~$ lsusb
Bus 002 Device 009: ID 0bda:0411 Realtek Semiconductor Corp.
Bus 002 Device 008: ID 2109:0813 VIA Labs, Inc.
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 017: ID 10c4:ea60 Cygnal Integrated Products, Inc. CP210x UART Bridge / myAVR mySmartUSB light
Bus 001 Device 027: ID 17ef:6093 Lenovo
Bus 001 Device 026: ID 1c4f:0082 SiGma Micro
Bus 001 Device 025: ID 0bda:5411 Realtek Semiconductor Corp.
Bus 001 Device 024: ID 10c4:ea60 Cygnal Integrated Products, Inc. CP210x UART Bridge / myAVR mySmartUSB light
Bus 001 Device 023: ID 2109:2813 VIA Labs, Inc.
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

2、新建一个映射规则

```
sudo vi /etc/udev/rules.d/rplidar.rules
```

3、输入对应内容

```
SUBSYSTEM=="tty",ENV{ID_PATH}=="platform-3530000.xhci-usb-0:3:1.0", MODE="0666",
SYMLINK+="rplidar"
```

```
SUBSYSTEM=="tty",ENV{ID_PATH}=="platform-3530000.xhci-usb-0:3:1.0",MODE=="0666",SYMLINK+="rplidar"
~

~

~

~

~

DEVLINKS=/dev/serial/by-path/platform-3530000.xhci-usb-0:3:1.0-port0 /dev/rplidar
DEVNAME=/dev/ttyUSB0
DEVPATH=/devices/3530000.xhci/usb1/1-3/1-3:1.0/ttyUSB0/ttyUSB0
ID_BUS=usb
ID_MM_CANDIDATE=1
ID_MODEL=CP2102_USB_to_UART_Bridge_Controller
ID_MODEL_ENC=CP2102\0x20USB\0x20to\0x20UART\0x20Bridge\0x20Controller
ID_MODEL_FROM_DATABASE=CP210x UART Bridge / myAVR mySmartUSB light
ID_MODEL_ID=ea60
ID_PATH=platform-3530000.xhci-usb-0:3:1.0
ID_PATH_TAG=platform-3530000_xhci-usb-0_3_1_0
ID_REVISION=0100
ID_SERIAL=Silicon_Labs_CP2102_USB_to_UART_Bridge_Controller_0001
ID_SERIAL_SHORT=0001
ID_TYPE=generic
ID_USB_DRIVER=cp210x
ID_USB_INTERFACES=:ff0000:
ID_USB_INTERFACE_NUM=00
ID_VENDOR=Silicon_Labs
ID_VENDOR_ENC=Silicon\0x20Labs
ID_VENDOR_FROM_DATABASE=Cygnal Integrated Products, Inc.
ID_VENDOR_ID=10c4
MAJOR=188
MINOR=0
SUBSYSTEM=tty
SYSTEMD_WANTS=gpsdctl@ttyUSB0.service
TAGS=:systemd:
USEC_INITIALIZED=225044255
net.ifnames=0
```

这里 rplidar 就是我们自定义的端口号。

4、重新插拔设备使之生效（一定要重新插拔），并输入对应指令查看。

```
ls -l /dev |grep ttyUSB
```

```
nvidia@tegra-ubuntu:~$ ls -l /dev |grep ttyUSB
lrwxrwxrwx 1 root root 7 Oct 10 11:48 gps0 -> ttyUSB0
lrwxrwxrwx 1 root root 7 Oct 10 11:52 gps1 -> ttyUSB1
lrwxrwxrwx 1 root root 7 Oct 10 11:48 rplidar -> ttyUSB0
crw-rw-rw- 1 root dialout 188, 0 Oct 10 11:48 ttyUSB0
crw-rw-rw- 1 root dialout 188, 1 Oct 10 11:57 ttyUSB1
lrwxrwxrwx 1 root root 7 Oct 10 11:52 ultrasonic -> ttyUSB1
```

上图则代表绑定端口号成功。针对激光雷达（原本对应端口号为 tty/USB0）和超声波（原本对应端口号 tty/USB1），现自定义端口号分别为/tty/rplidar 和/tty/ultrasonic

说明：

- a、如果程序中需要对激光雷达端口进行操作的话，可以使用 tty/USB0 或者 tty/rplidar。
- b、按照此方法绑定端口号后，不能改变设备所插入 USB 端口。
- c、删除对应的.rules 文件即可取消绑定。