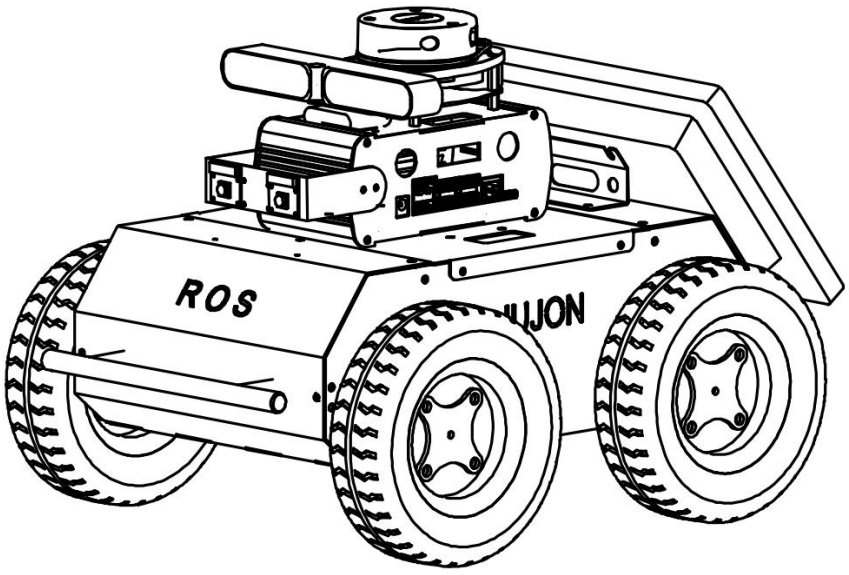
JUJON

# Ravelle

## 实验指导书 ［路威］




V 2.3

2021年8月12日星期四

# 版本信息说明

| 版本 | 变更描述 | 日期 | 编辑或审查 |
|---|---|---|---|
| V 1.0 | 创建实验指导书 | July 20, 2021 | 朱晓宇 |
| V 1.2 | 完善内容 | July 20, 2021 | 朱晓宇 |
| V 1.2 | 审核与修改 | July 21, 2021 | 杨雄 |
| V 1.5 | 完善内容 | July 21, 2021 | 朱晓宇 |
| V 1.5 | 审核与修改 | July 21, 2021 | 杨雄 |
| V 1.8 | 完善内容 | July 23, 2021 | 魏玉虎 |
| V 1.8 | 审核与修改 | July 23, 2021 | 杨雄 |
| V 2.0 | 完善内容 | July 24, 2021 | 朱晓宇 |
| V 2.0 | 审核与修改 | July 24, 2021 | 杨雄 |
| V 2.1 | 完善内容 | July 26, 2021 | 朱晓宇 |
| V 2.1 | 审核与修改 | July 26, 2021 | 杨雄 |
| V 2.2 | 完善内容 | July 27, 2021 | 魏玉虎 |
| V 2.2 | 审核与修改 | July 27, 2021 | 杨雄 |
| V 2.3 | 完善内容 | August 10, 2021 | 魏玉虎 |
| V 2.3 | 审核与修改 | August 10, 2021 | 杨雄 |
| V 2.3 | 审核与修改 | August 17, 2021 | 魏玉虎 |

# 目 录

# 目 录

# 目 录

# 目 录

# 目 录

# 目 录

# 目 录

# 目 录

# 目 录

# 目 录

# 目 录

# 第一章 路威套件开箱指南

## 1.1 实验目的

了解路威套件的使用方法，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，路威套件。

## 1.4 实验内容

### 1.4.1 开箱说明

开箱说明请参照路威用户手册。

### 1.4.2 Nomachine 远程连接

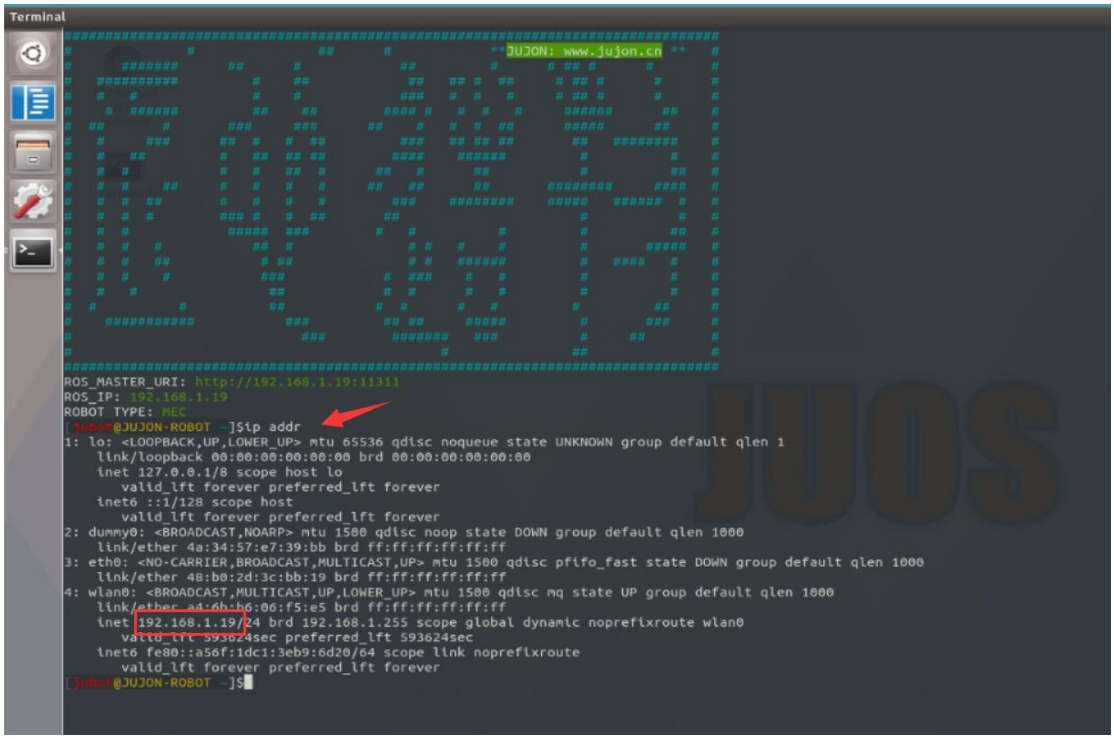
## 路威套件上连接 Nomachine 准备工作

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，首先需要插上路威套件的外接显示器、鼠标键盘，接好之后开机，显示器会显示操作系统界面。


JUOS是我们基于 Ubuntu 18.04 深度定制的 Linux 操作系统，对于路威ROS学习套

件有更好的契合。

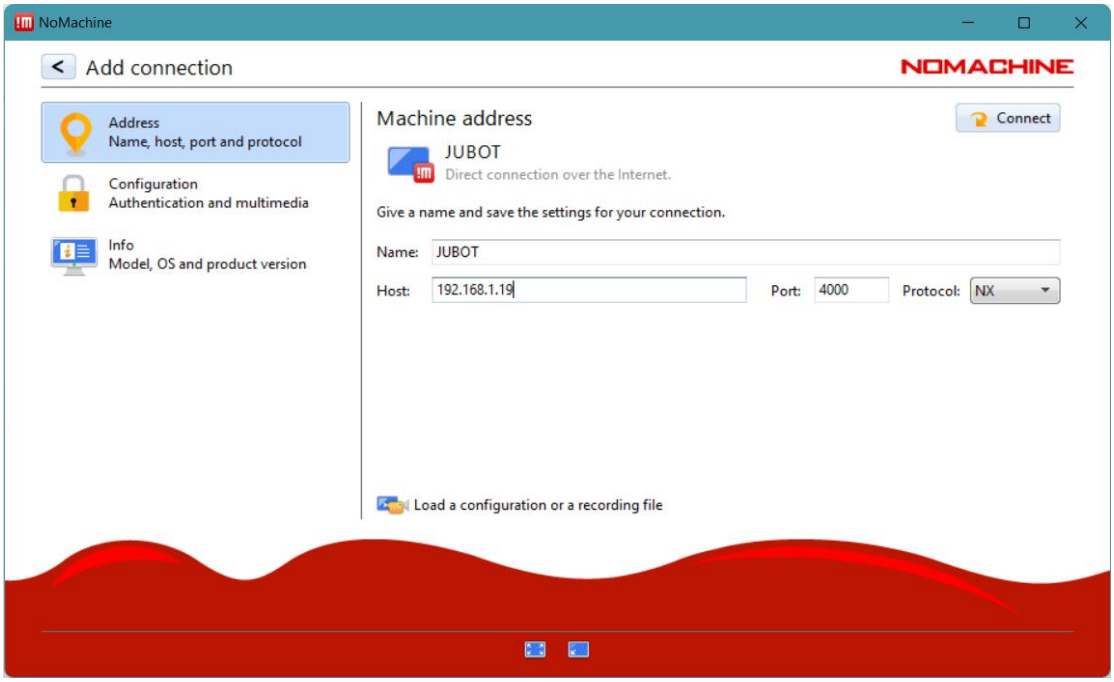
手动点击右上角 wifi 信号连接到无线网络，以后路威套件会自动连接到该网络。点击左侧终端按钮或按下 Ctrl+Alt+T 打开终端，输入 ip addr 得到 IP 地址：



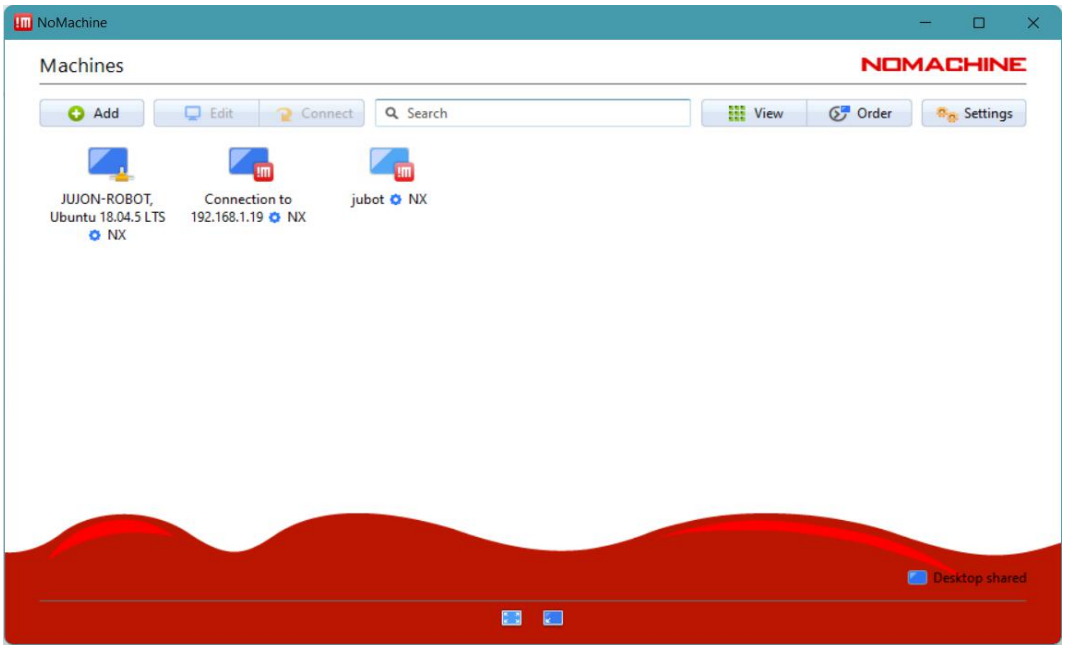

这里是 192.168.1.19，记住该地址，插入随机赠送的 hdmi 虚拟模块,然后拔掉 hdmi（不插拔一次PC机的hdmi线直接使用NoMachine连接机器人，屏幕分辨率可能会显示不正常）。

## 个人电脑上安装 Nomachine 并连接

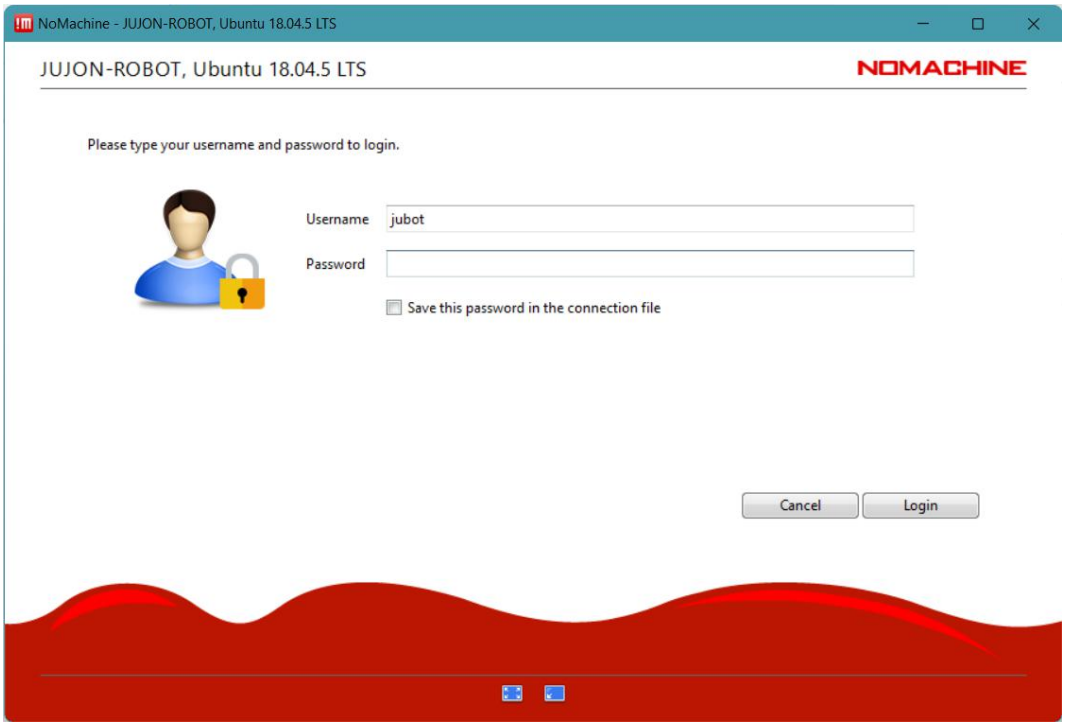
在个人电脑上安装Nomachine，下载地址：https://www.nomachine.com/ 安装完成后打开（保证路威套件和个人电脑在一个 wifi 网络下），点击add，然后输入对应的主机名，点击右上角的Connect，即可输入密码进行连接。




如果之前连接过设备，可能会出现下面的界面。




输入账号：jubot 密码：jubot，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 配置JUOS虚拟机

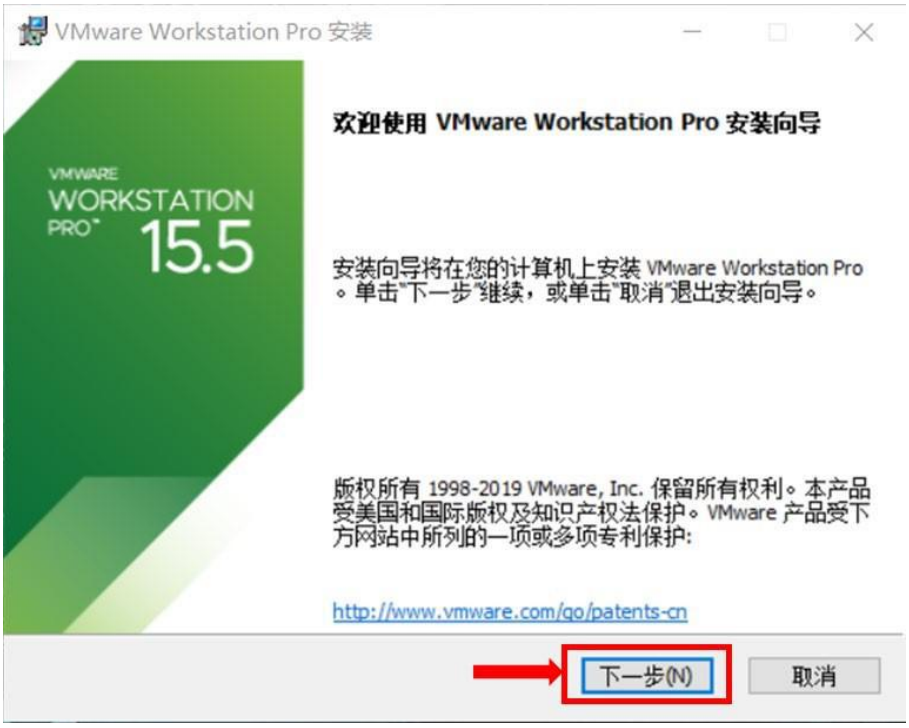
除了 NoMachine 连接的方式，为了实验的方便，我们还为用户准备了定制版虚拟机。对于没有 Linux 和 ROS 基础的用户，我们强力推荐使用我们的虚拟机环境开发，我们的环境已经安装好 ROS 和相关软件环境，使用相对简单，跟我们的教程也完全配套。等熟悉后可以再自行搭建开发环境。

**使用虚拟机的操作流程如下**

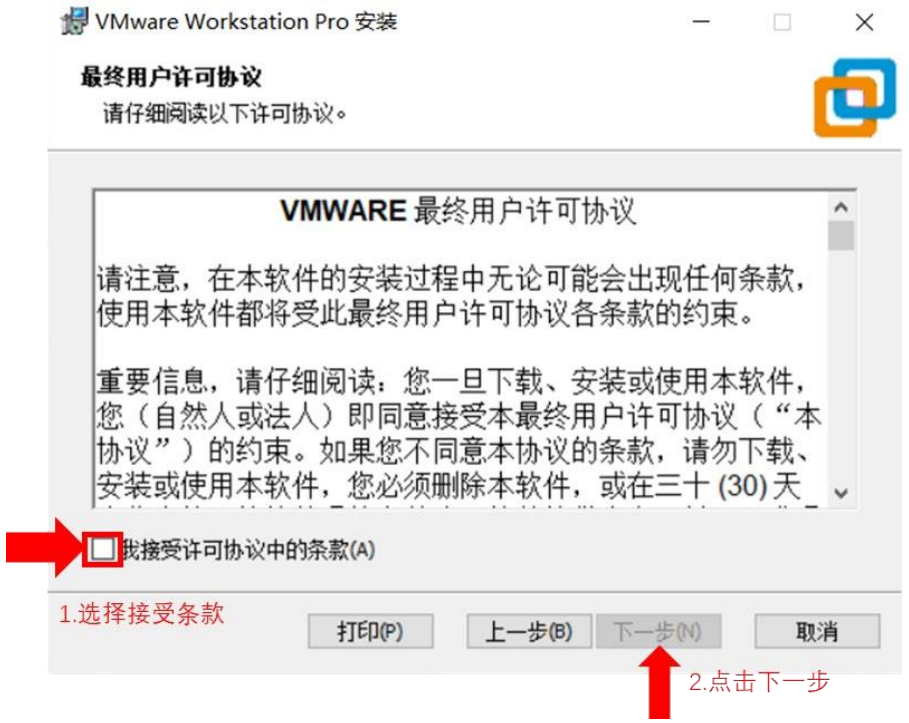
安装 VMware 软件

VMware Workstation 软件可以在 VMware 官网下载，资料包中也提供了VMware

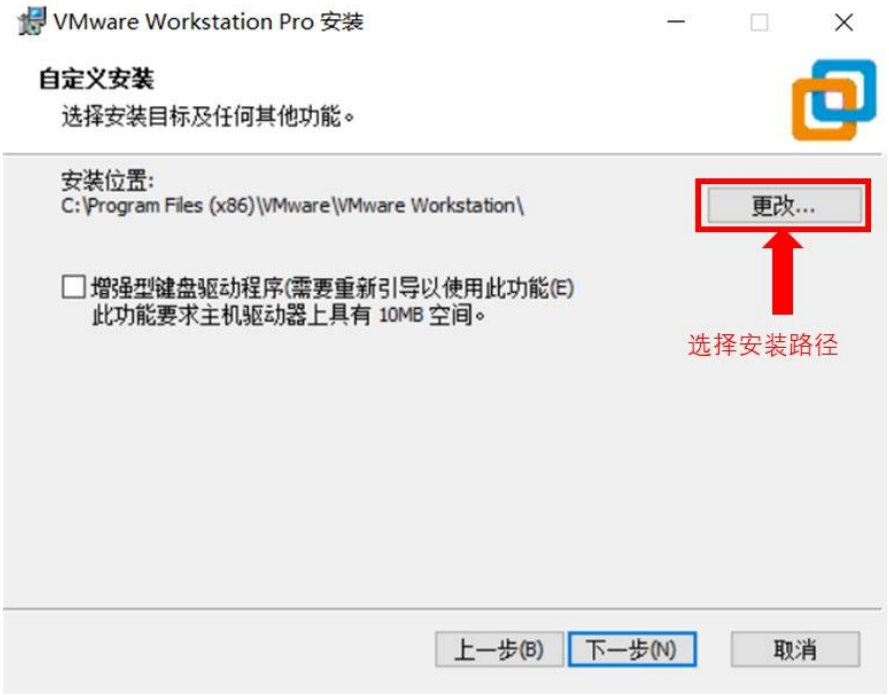
Workstation 软件，安装步骤如下。



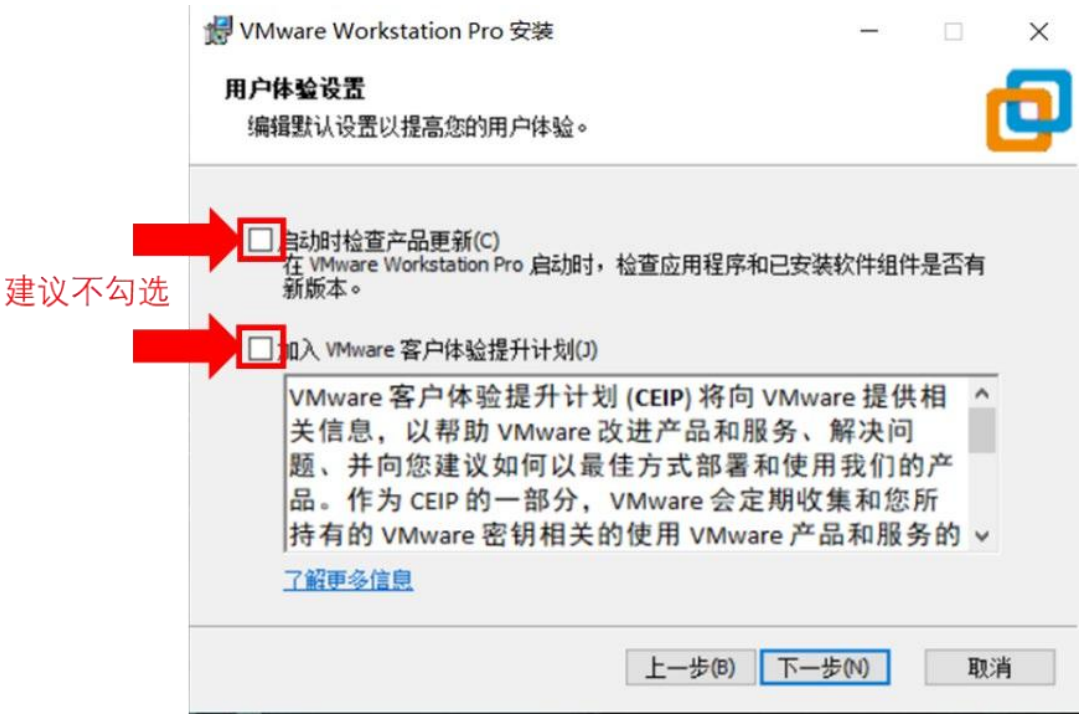

点击上图中的下一步，弹出如下图的页面：



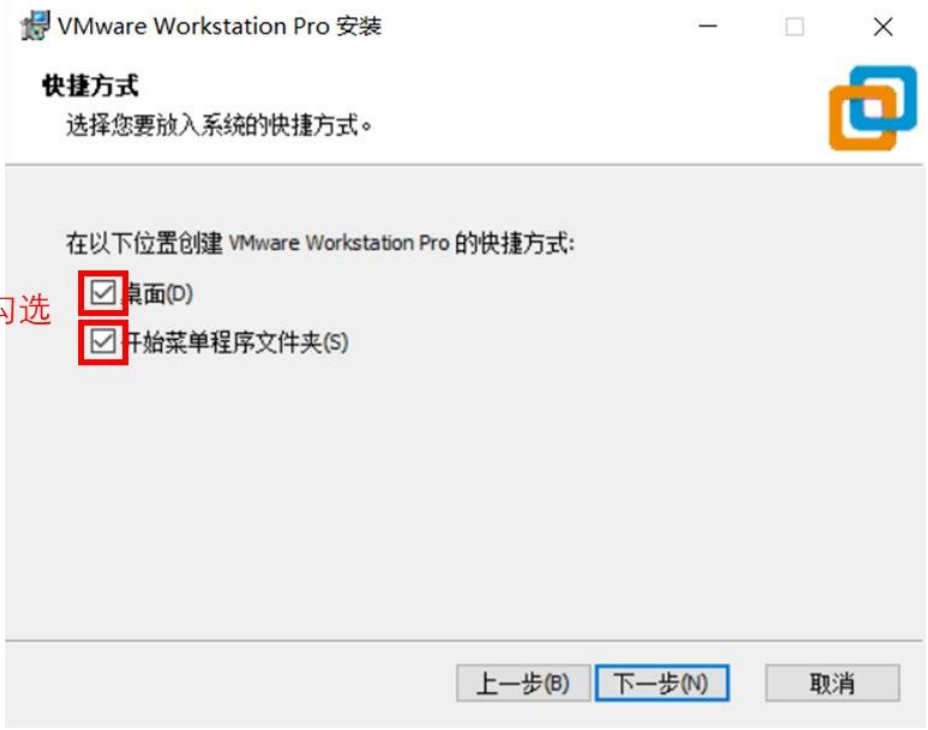

选择上图中的"我接受许可协议中的条款"，点击下一步。



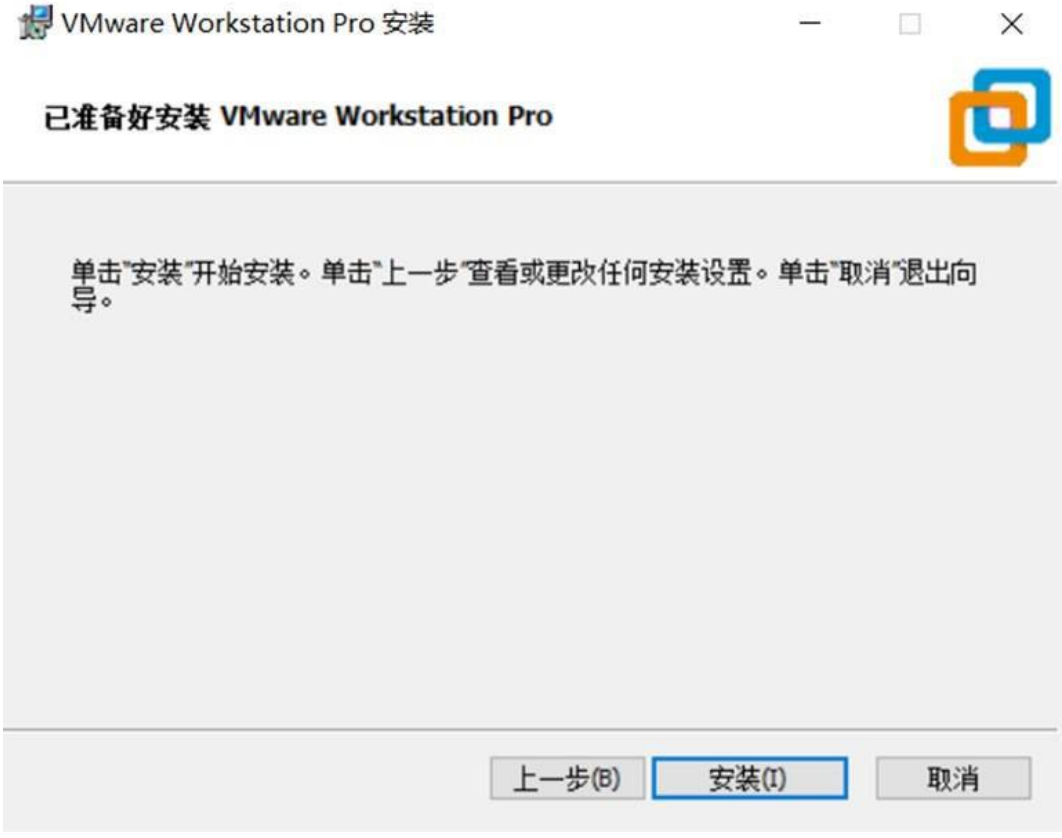

根自己需要更改安装路径，选择好路径后点击下一步继续安装。




更新检查错误上报，建议不要开启，注意这两个复选框默认都是勾选状态。



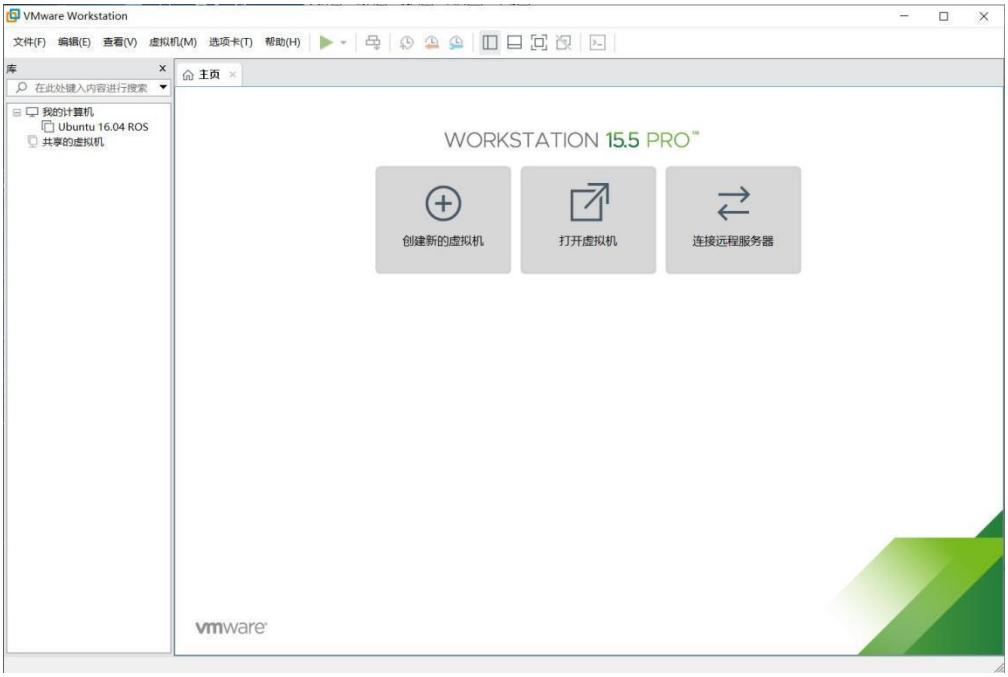

创建快捷方式，建议勾选。




点击安装，开始安装，耐心等待，即可完成安装。


第一次启动软件时，将弹出下图许可证界面，输入产品密钥。


当弹出下图界面时，到此VMware 虚拟机软件大致安装完成。



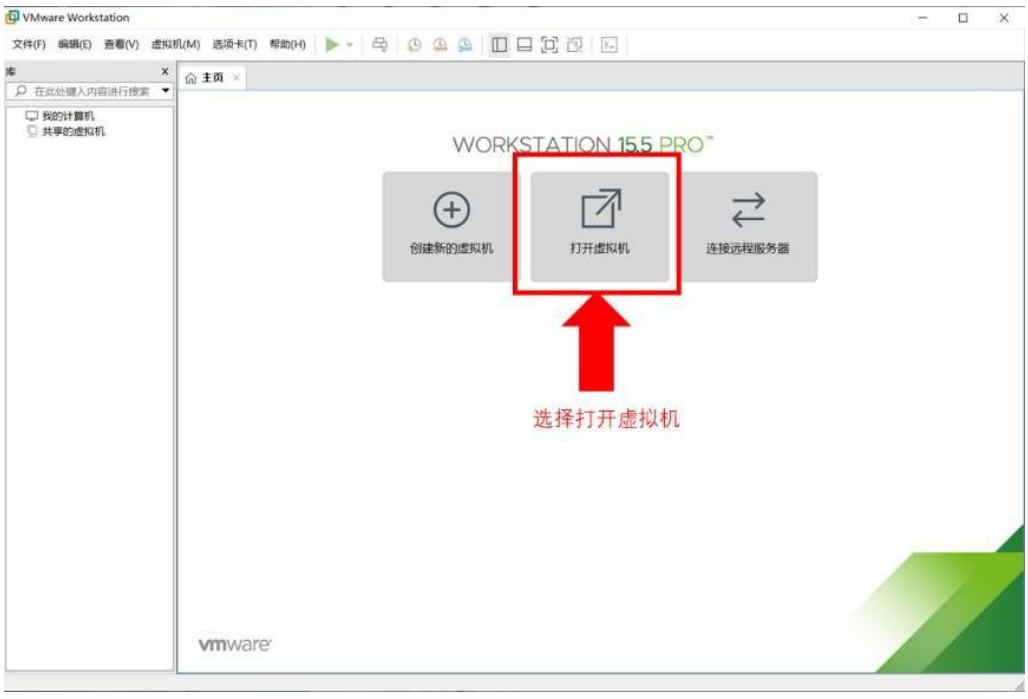

## 加载使用 JUOS 虚拟机

我们已为用户制作了一个安装有ROS 及相关工具的Ubuntu18.04 系统，用户可直接加载使用。系统存放在系统镜像文件夹下"JUOS_ROS_VM_V3.0.zip"，首先进行解压，解压后文件较大，所以建议解压时就选好路径，避免再次移动。

虚拟机默认用户名：jubot

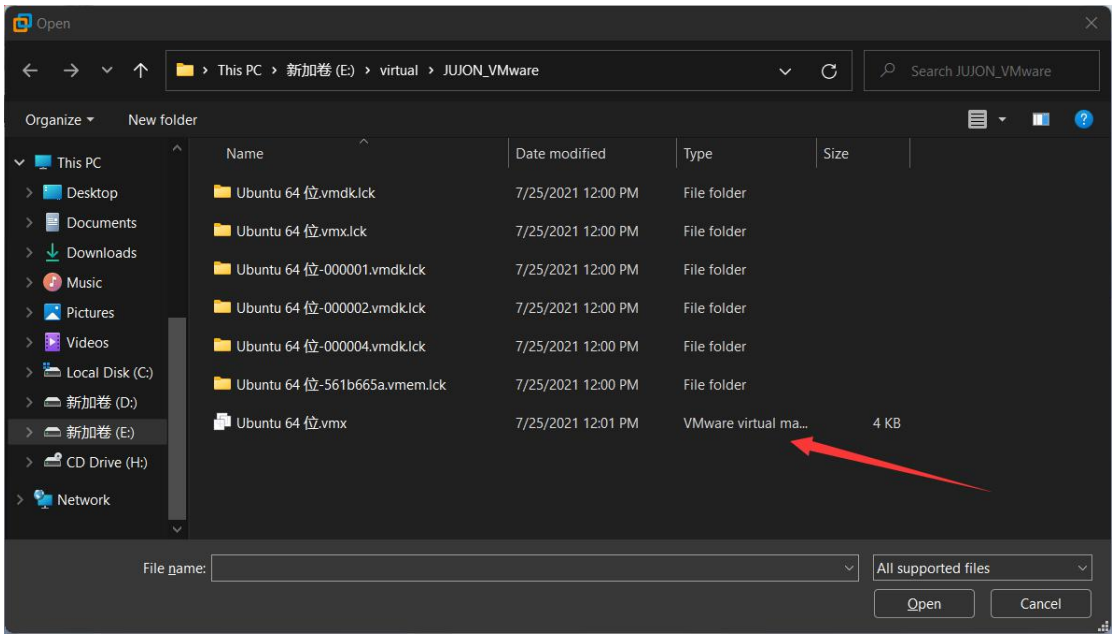
虚拟机默认密码：jubot

打开虚拟机安装软件VMware，点击下图的"打开虚拟机"选项。




会出现下图的路径选择界面，我们选择之前解压的路径，并选择"Ubuntu 64

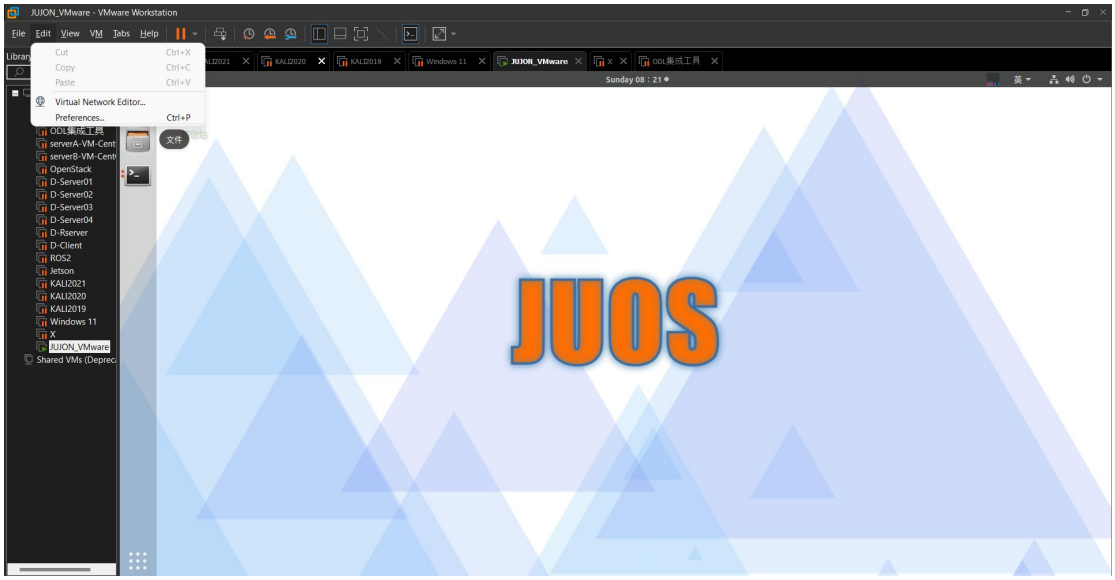
位.vmx"，也就是下图中标注的文件，然后点击打开。



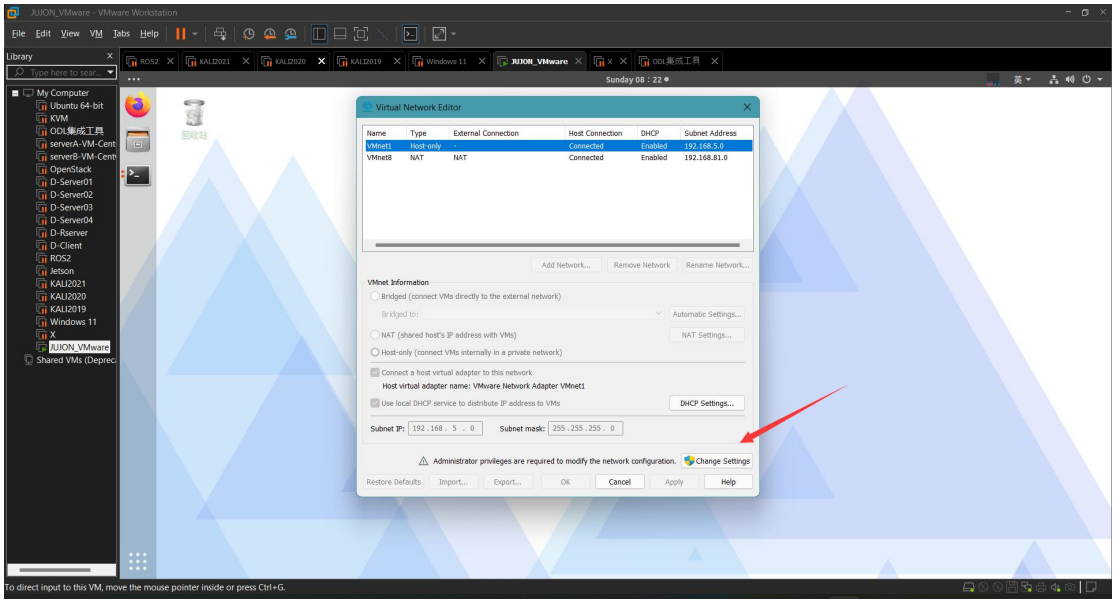

加载完成后，如下图。


**虚拟机网络配置**

启动虚拟机前，请先进行网络配置。点击"编辑"菜单栏下的"虚拟网络编辑器"如下图所示位置。



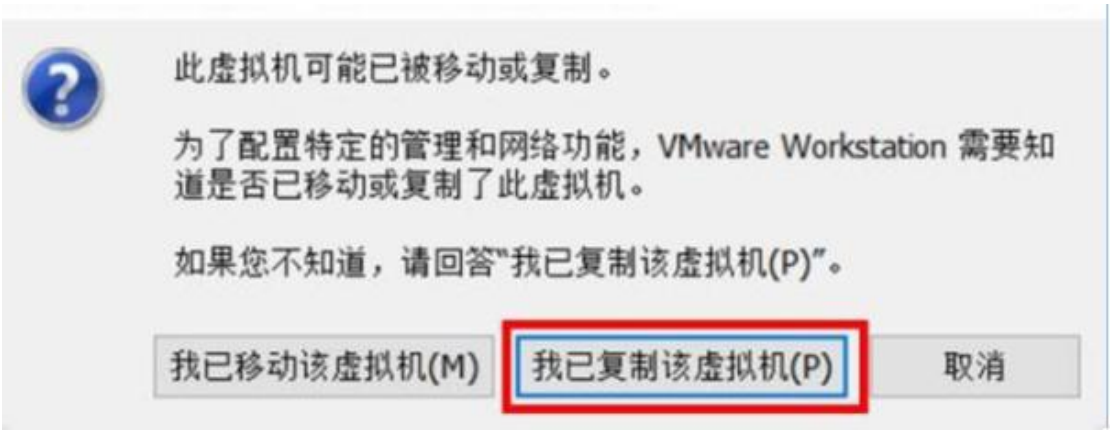

如果出现下图灰色显示，请点击下图"更改设置"图标，即可打开虚拟网络编辑器。



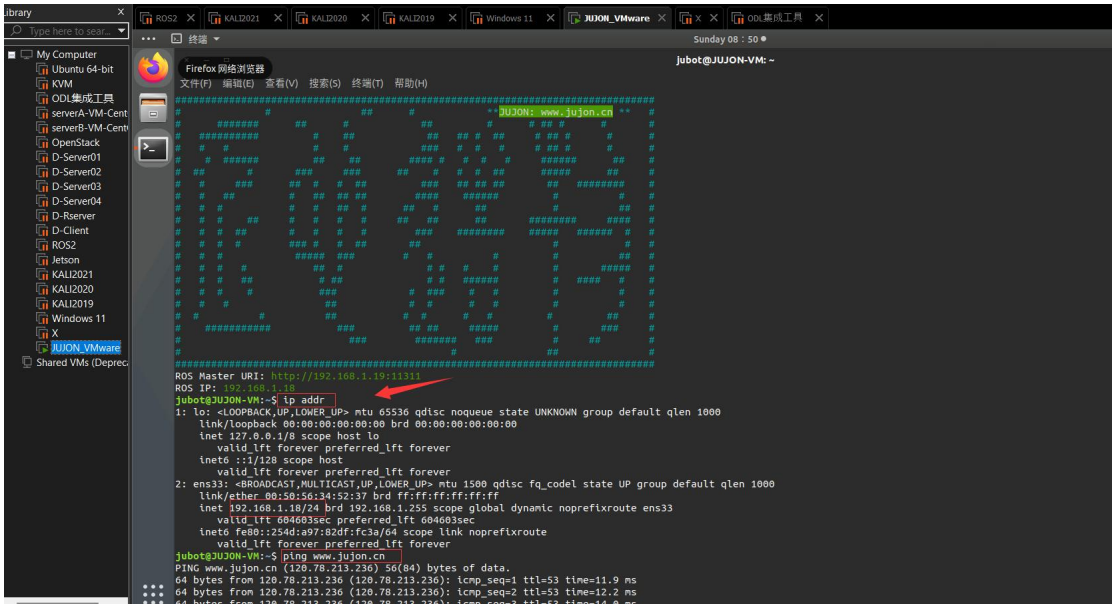

首先添加网络，如果已存在请忽略。虚拟机有三种网络连接模式，选择桥接模式。注意选择电脑实际使用的网卡设备，否则不能联网。一般台式机为普通有线网卡，笔记本为无线网卡，根据实际使用联网情况选择。


完成上述网络配置操作，即可开启此虚拟机。

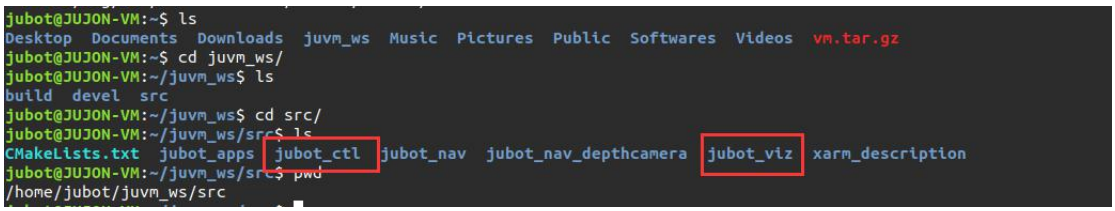

弹出下图的提示界面，选择"我已复制该虚拟机"。




可以通过打开终端输入 ip addr 命令查询IP 地址，并验证网络连接，验证IP 可以通过ping www.jujon.cn 的命令，看是否能够建立连接，详细参照下图。



使用文件管理器打开ROS 工作空间，有如下功能包，其中jubot_ctl 为手柄及键盘控制包，jubot_viz 为存放 RVIZ 配置文件。

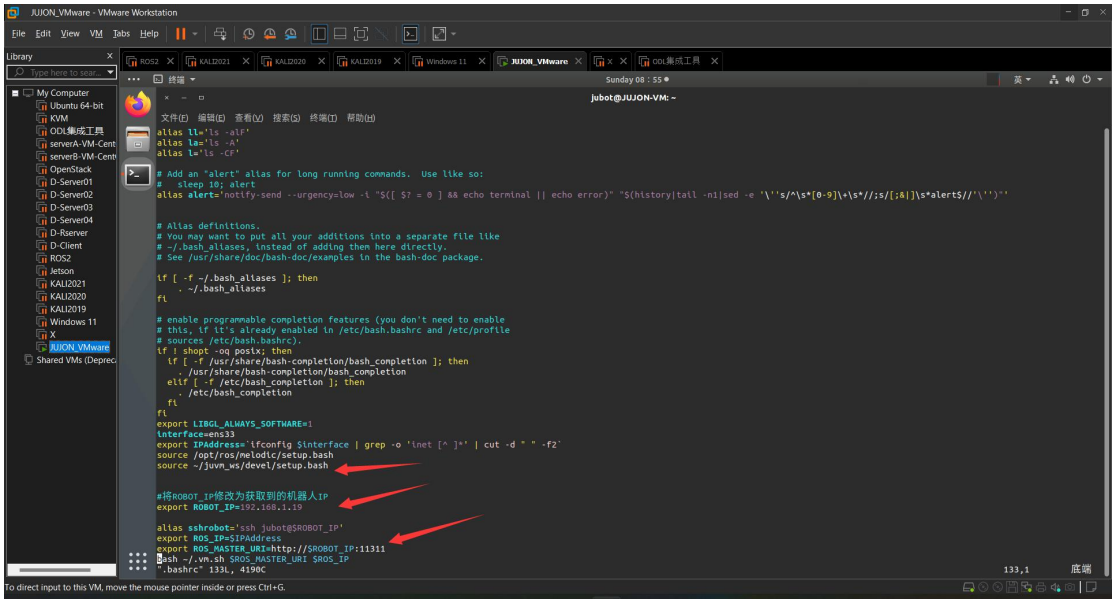



### 1.4.4 SSH连接路威套件

下文为我们提供的虚拟机为例进行说明。

在家目录下，用vim 编辑 .bashrc 配置文件

```
vim .bashrc
```




保存退出后，source .bashrc 文件使修改生效。

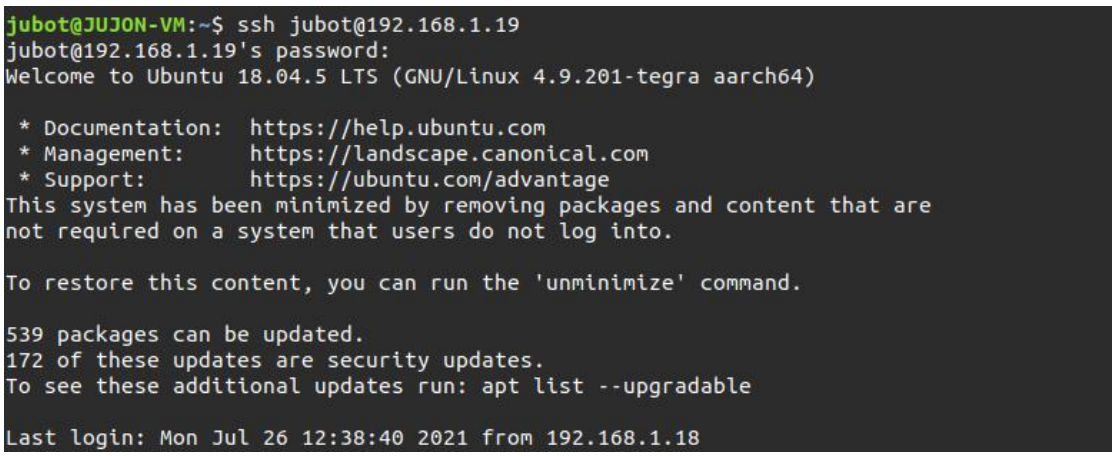
```
source .bashrc
```


系统环境变量即可自动配置完成ROS_IP,ROS_MASTER_URI 以及sshrobot 等参数。"sshrobot"为 SSH 登录机器人的快捷方式。

Nano 连接到 WIFI 后获取 IP 地址，开发主机可通过 SSH 方式连接到 Nano。命令如下，IP 地址更换为实际IP 地址。

```
ssh jubot@192.168.xx.xx
```

提示输入密码，注意密码不显示，输入完密码按回车即可。




路威套件默认账号：jubot，密码 jubot

为了 SSH 连接方便，我们把SSH 登录 Nano 命令映射为"sshrobot"特定字符， 用户只需输入该字符，即可 SSH 登录 Nano。

```
jubot@JUJON-VM:~$ sshrobot
The authenticity of host '192.168.1.19 (192.168.1.19)' can't be established.
ECDSA key fingerprint is SHA256:/H+OAYdF5RHy0xHZXUYGq6IMzh34Lg0VXHdfyhE5azQ.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.1.19' (ECDSA) to the list of known hosts.
jubot@192.168.1.19's password:
Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 4.9.201-tegra aarch64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage
This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

539 packages can be updated.
172 of these updates are security updates.
To see these additional updates run: apt list --upgradable

Last login: Sun Jul 25 18:50:08 2021 from 192.168.1.34
```

对机器人的实践和开发过程中，会频繁 SSH 登录机器人，使用快捷方式可以提高 SSH 登录效率。

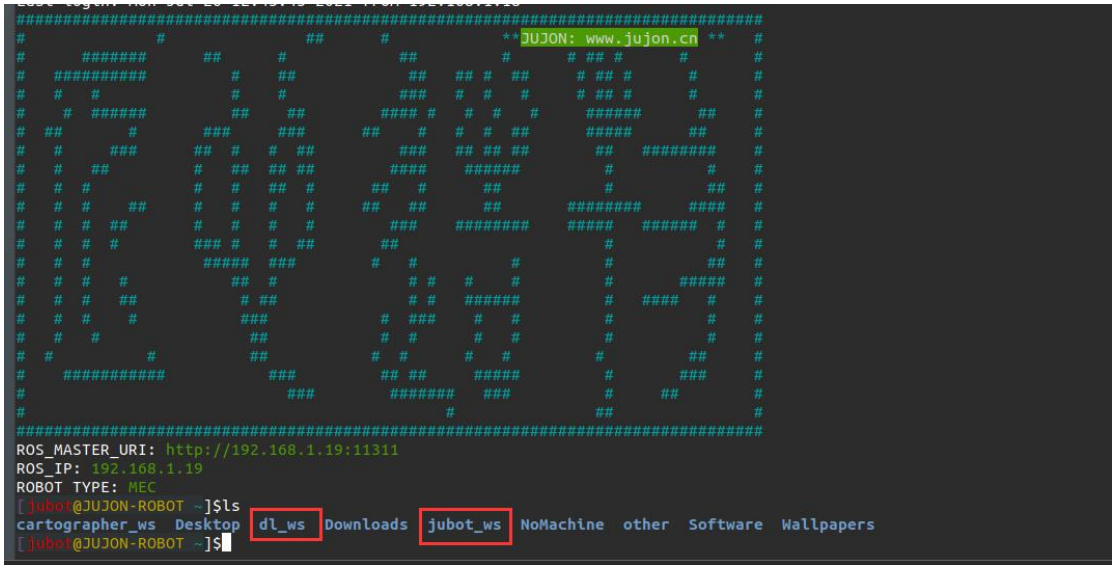
如果自己搭建的主机环境或部分老的虚拟机镜像没有sshrobot 快捷方式，可自行添加如下文本到".bashrc"文件下，注意修改为自己的 IP 地址。

alias sshrobot='ssh jubot@192.168.1.19'

```
export LIBGL_ALWAYS_SOFTWARE=1
interface=ens33
export IPAddress=`ifconfig $interface | grep -o 'inet [^ ]*' | cut -d " " -f2`
source /opt/ros/melodic/setup.bash
source ~/juvm_ws/devel/setup.bash


#将ROBOT_IP修改为获取到的机器人IP
export ROBOT_IP=192.168.1.19

alias sshrobot='ssh jubot@$ROBOT_IP'
export ROS_IP=$IPAddress
export ROS_MASTER_URI=http://$ROBOT_IP:11311
bash ~/.vm.sh $ROS_MASTER_URI $ROS_IP
```

### 1.4.5 路威套件软件框架与功能预览

路威套件的软件架构是基于 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：




一共有两个工作空间 robot_ws 和 dl_ws。

先看看 robot_ws:


- jubot_apps:安卓手机端移动控制/建图导航
- jubot_ctl:键盘/手柄控制
- jubot_cv:opencv视觉巡线/kcf目标追踪/yolo物体识别/二维码识别导航等
- jobot_driver:路威套件底盘驱动，运动控制，模型等代码
- jubot_multirobot:多机器人编队导航
- jobot_nav:多种算法雷达建图/导航
- jubot_nav_depthcamera:VSLAM深度相机建图/导航
- jubot_voice:六麦克风阵列语音导航/交互
- third_packages:各种第三方包依赖驱动

再看看 dl_ws:

```
[jubot@JUJON-ROBOT ~]$cd dl_ws/
[jubot@JUJON-ROBOT ~/dl_ws]$ls
darknet  tensorflow
[jubot@JUJON-ROBOT ~/dl_ws]$pwd
/home/jubot/dl_ws
[jubot@JUJON-ROBOT ~/dl_ws]$
```

- darknet:yolo深度学习网络
- tensorflow:训练模型/识别手写数字

## 1.5实验结果

熟悉路威套件的框架，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

1.能够创建自己的工作空间以及功能包。

2.了解 ROS 中典型的通讯机制。

3.了解典型 launch 文件的用法。

## 2.3 实验工具

个人电脑一台，路威套件。

## 2.4 实验内容

### 2.4.1 初识ROS 工作空间、功能包以及节点

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 -pv ~/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。

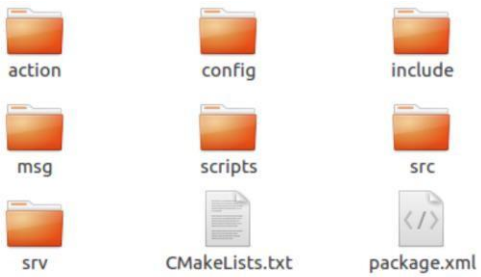
如果希望环境变量在所有终端中有效，则需要在终端配置文件中加入环境变量的设置：

```
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_ws$ cd src

接着使用 catkin_create_pkg 命令来创建一个名为 jujontest 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_ws/src$ catkin_create_pkg jujontest roscpp rospy std_msgs

含义是创建一个名为 jujontest 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 jujontest/CMakeLists.txt

Created folder jujontest/include/my_demo

Created folder jujontest/src

Successfully created files in

/home/jujon/ros_ws/src/jujontest

Please adjust the values in package.xml

这将会创建一个名为 jujontest 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 <sstream>
#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;
}
```

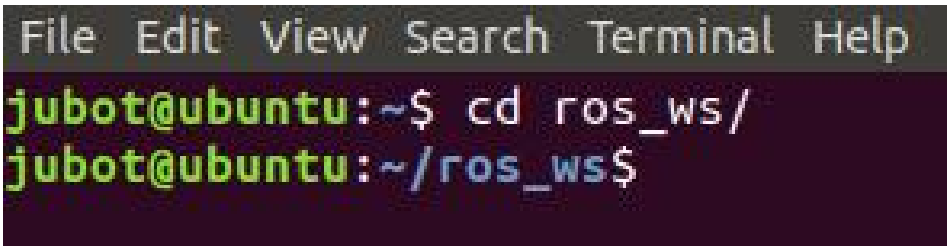
这是一个话题通讯中的话题发布者（关于通讯会在后面介绍，这里只看简单结构），名为 Talker.cpp 的一个 cpp 文件，是一个用 C++编写的节点，ROS 中不仅支持 C++也支持 Python 等多种语言，该节点经过初始化之后完成了发布一个名为 chatter 的话题，并发布消息类型为 String 的消息。

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。

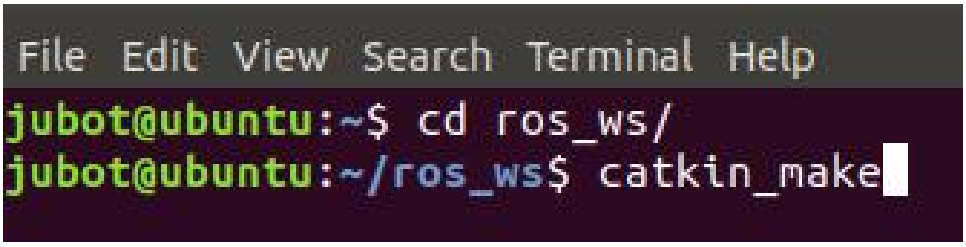
### 2.4.2 ROS 编译系统

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：

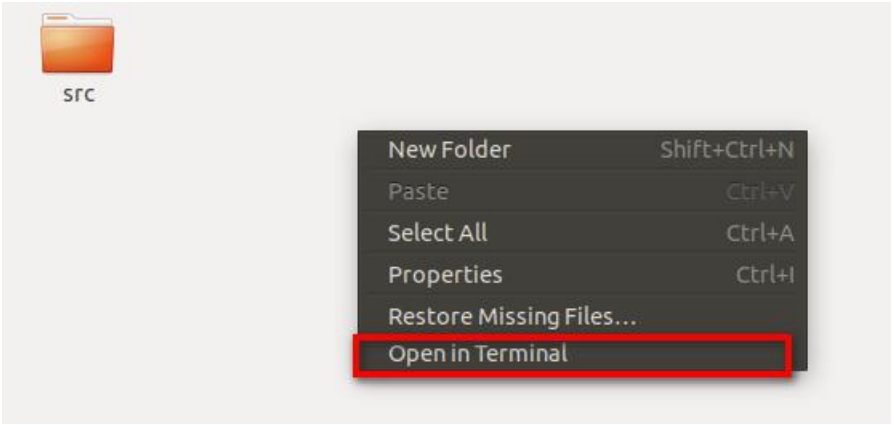



输入 cd <工作空间名>即可进入该路径：




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 format="2">
  <name>hello</name>
  <version>0.0.0</version>
  <description>The hello package</description>

  <!-- One maintainer tag required, multiple allowed, one person per tag -->
  <!-- Example:  -->
  <!-- <maintainer email="jane.doe@example.com">Jane Doe</maintainer> -->
  <maintainer email="upi@example.todo">your name</maintainer>


  <!-- One license tag required, multiple allowed, one license per tag -->
  <!-- Commonly used license strings: -->
  <!--   BSD, MIT, Boost Software License, GPLv2, GPLv3, LGPLv2.1, LGPLv3 -->
  <license>BSD</license>

  <!-- The *depend tags are used to specify dependencies -->
  <!-- Dependencies can be catkin packages or system dependencies -->
  <!-- Examples: -->
  <!-- Use depend as a shortcut for packages that are both build and exec depend
encies -->
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>roscpp</build_depend>
  <build_depend>rospy</build_depend>
  <build_depend>std_msgs</build_depend>
  <build_export_depend>roscpp</build_export_depend>
  <build_export_depend>rospy</build_export_depend>
  <build_export_depend>std_msgs</build_export_depend>
  <exec_depend>roscpp</exec_depend>
  <exec_depend>rospy</exec_depend>
  <exec_depend>std_msgs</exec_depend>
</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 3.0.2)
project(hello)
```

声明CMake API 版本以及项目名称

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

搜索依赖项（即 roscpp 等）的信息

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

搜索 catkin 中调用的头文件设置

```
add_executable(hello src/talter.cpp)
```

待生成可执行文件的名字设置

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

编译过程的linking library

```
add_dependencies(talker ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})
```

添加依赖项

### 2.4.3 ROS 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。

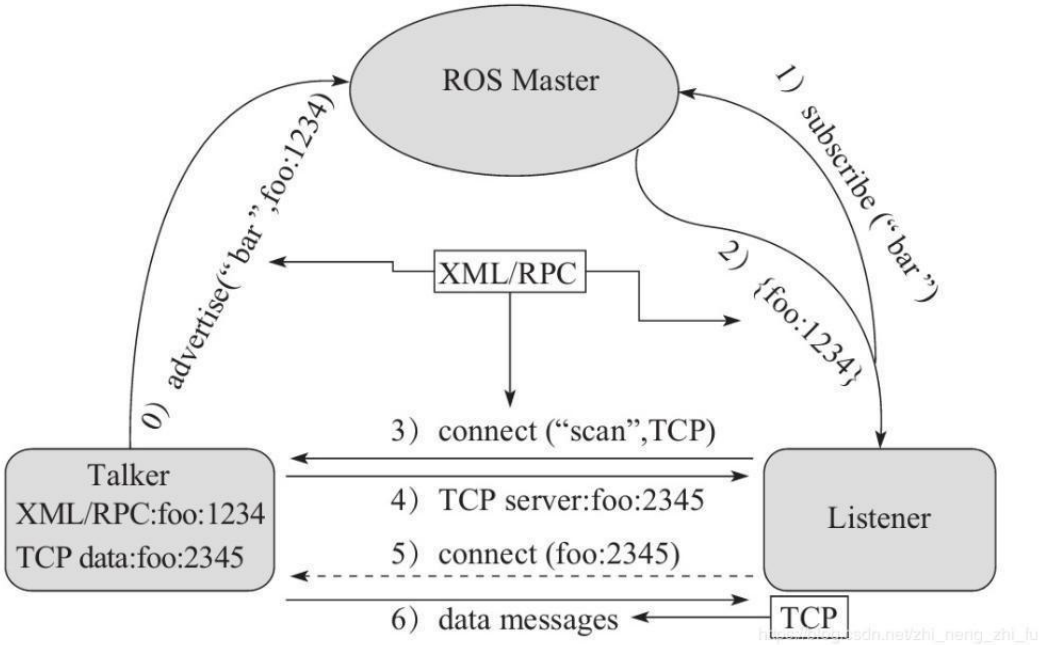
**节点管理器（ROS Master）**

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。

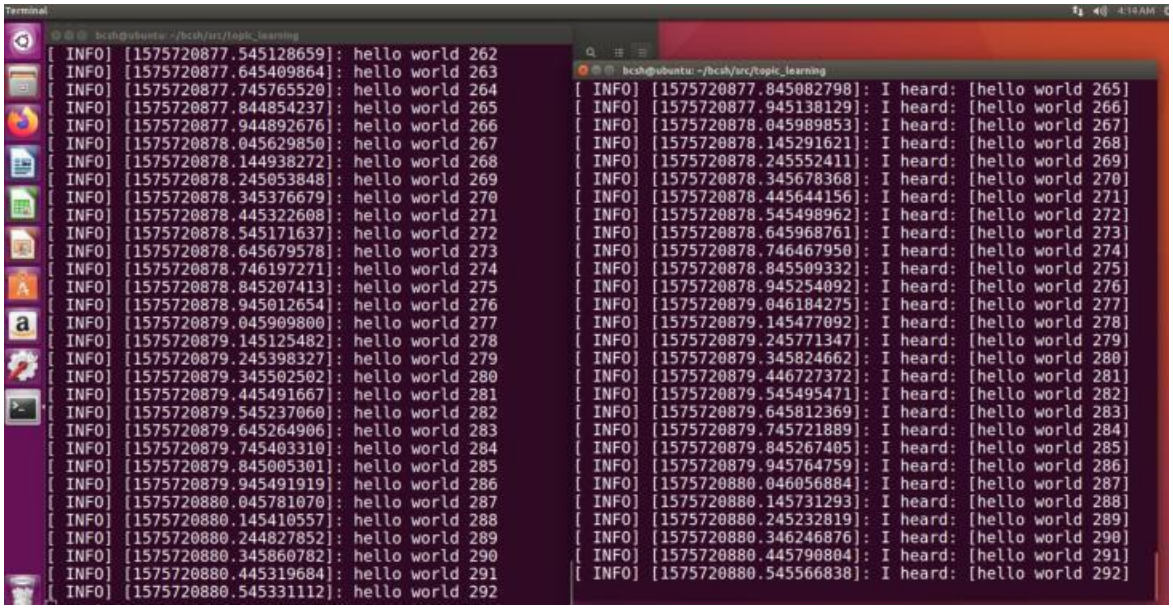
在该比喻中，Topic：找工作 ；Master：平台 M ；Talker：公司 A ；Listener：求职者 B。

## 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 jujontest talker` 启动 talker 节点以及启动 listener 节点：`rosrun jujontest listener`

可以看到如下结果：




我们来学习一下talker 和 listener 的源码：

**Talker：**

```cpp
/**
* 该例程将发布 chatter 话题，消息类型 String
*/
#include <sstream>
#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_msg
    // 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：

```cpp
/**
* 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

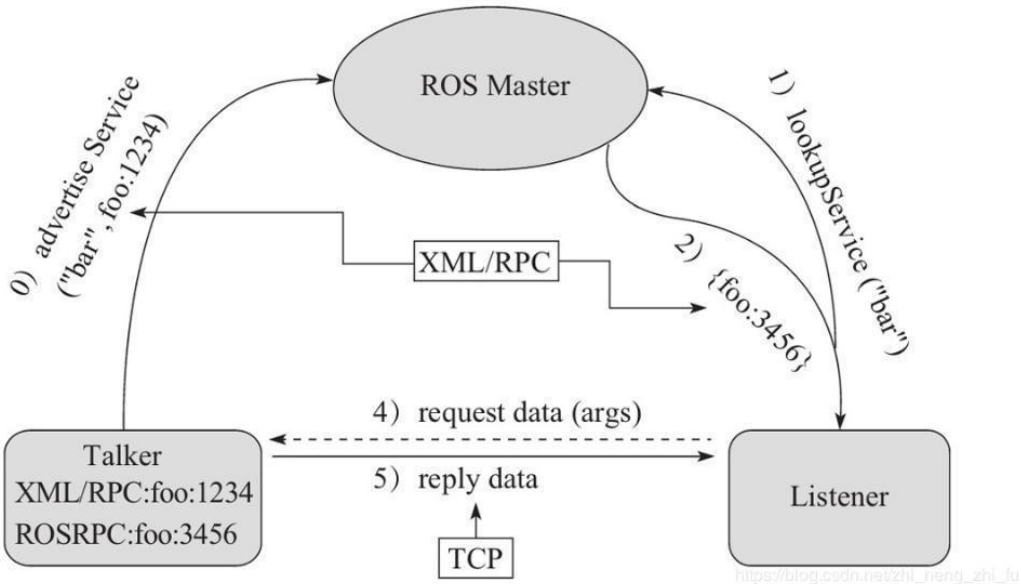
## 基本概念

**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：


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 turtlesimturtlesim_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。**

| | 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 路威套件控制功能的实现

## 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，路威套件。

## 3.4 实验内容

### 3.4.1 使用键盘控制路威套件运动

**检查机器人底盘能否正常运动**

打开新终端，输入 `roslaunch jubot_diver jubot_diver.launch` 进行底盘的连接。

```
[jubot@JUJON-ROBOT ~]$roslaunch jubot_driver jubot_driver.launch
... logging to /home/jubot/.ros/log/173a2008-ede2-11eb-979a-a46bb606f5e5/roslaunch-JUJON-ROBOT-15915.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://192.168.1.19:35359/

SUMMARY
========

PARAMETERS
 * /jubot_driver/Kd: 200
 * /jubot_driver/Ki: 0
 * /jubot_driver/Kp: 350
 * /jubot_driver/angular_correction_factor: 1.0
 * /jubot_driver/base_frame: base_footprint
 * /jubot_driver/baud_rate: 115200
 * /jubot_driver/control_rate: 50
 * /jubot_driver/imu_frame: base_imu_link
 * /jubot_driver/linear_correction_factor: 1.0
 * /jubot_driver/odom_frame: /odom
 * /jubot_driver/port_name: /dev/ttyTHS1
 * /jubot_driver/publish_odom_transform: True
 * /jubot_driver/version: XTARK-Mec&4WD
 * /rosdistro: melodic
 * /rosversion: 1.14.10
 * /use_sim_time: False

NODES
  /
    base_footprint_to_imu (tf/static_transform_publisher)
    jubot_driver (jubot_driver/jubot_driver)

auto-starting new master
process[master]: started with pid [15935]
ROS_MASTER_URI=http://192.168.1.19:11311
```

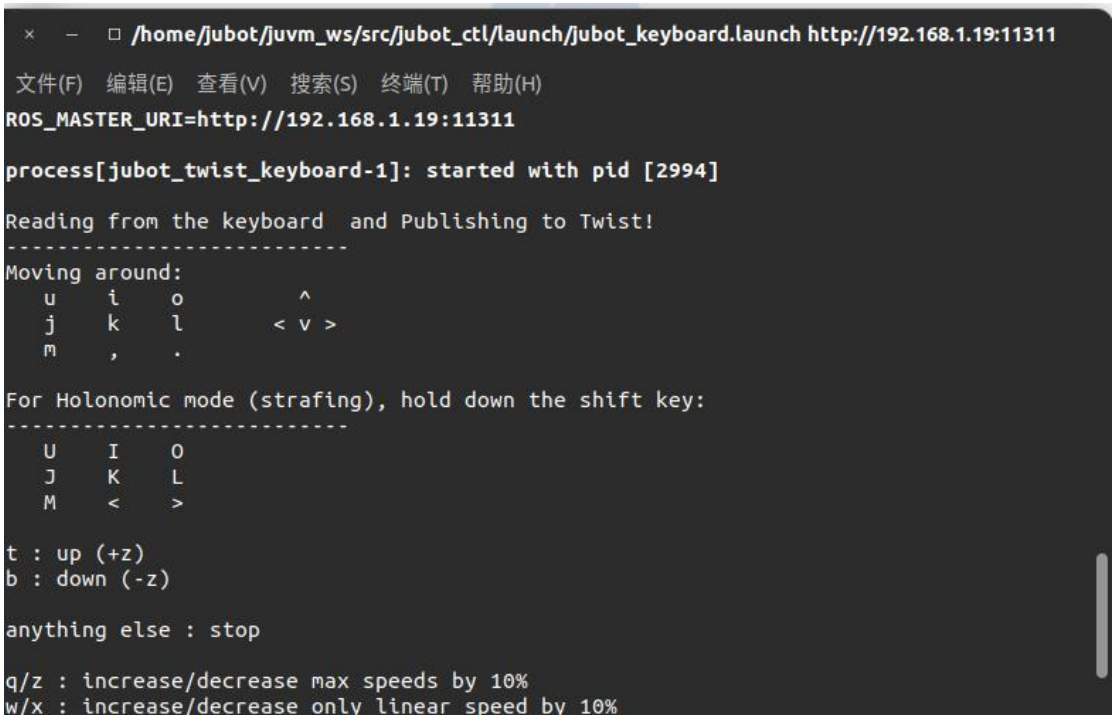
底盘一切连接正常打印日志内容如下（如果有黄色警告查看权限或者连接有没有问题）：

```
PARAMETERS
 * /jubot_driver/Kd: 200
 * /jubot_driver/Ki: 0
 * /jubot_driver/Kp: 350
 * /jubot_driver/angular_correction_factor: 1.0
 * /jubot_driver/base_frame: base_footprint
 * /jubot_driver/baud_rate: 115200
 * /jubot_driver/control_rate: 50
 * /jubot_driver/imu_frame: base_imu_link
 * /jubot_driver/linear_correction_factor: 1.0
 * /jubot_driver/odom_frame: /odom
 * /jubot_driver/port_name: /dev/ttyTHS1
 * /jubot_driver/publish_odom_transform: True
 * /jubot_driver/version: XTARK-Mec&4WD
 * /rosdistro: melodic
 * /rosversion: 1.14.10
 * /use_sim_time: False

NODES
  /
    base_footprint_to_imu (tf/static_transform_publisher)
    jubot_driver (jubot_driver/jubot_driver)

auto-starting new master
process[master]: started with pid [15935]
ROS_MASTER_URI=http://192.168.1.19:11311

setting /run_id to 173a2008-ede2-11eb-979a-a46bb606f5e5
process[rosout-1]: started with pid [15946]
started core service [/rosout]
process[jubot_driver-2]: started with pid [15949]
process[base_footprint_to_imu-3]: started with pid [15950]
1.000000 1.000000
[ INFO] [1627284092.468315252]: Send Time: 1627284092.468201
[ INFO] [1627284092.471516916]: Send End Time: 1627284092.471462
[ INFO] [1627284092.483559352]: Set PID P:[350], I:[0], D:[200]
[ INFO] [1627284092.487110495]: Robot Running!
```

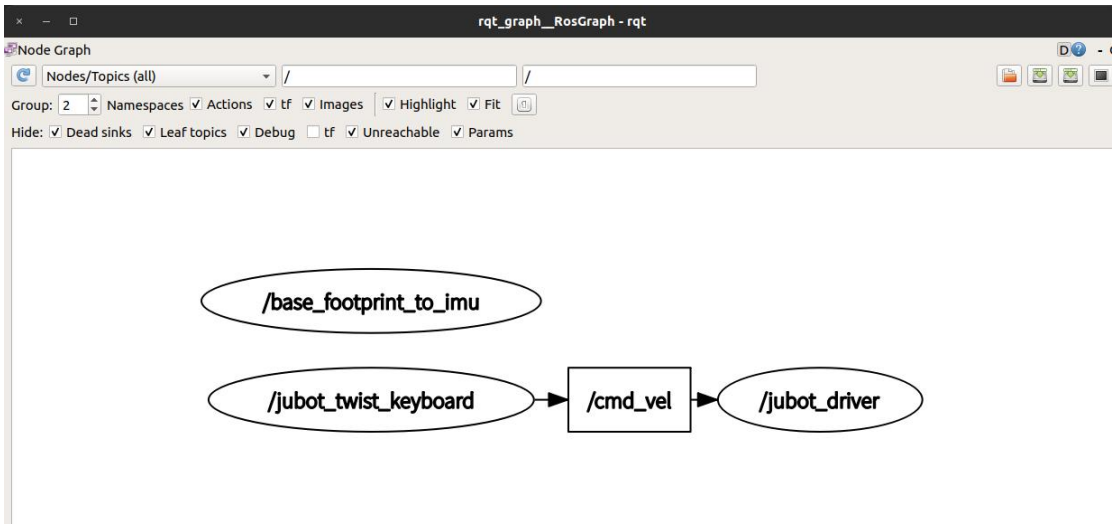
可以打开新终端或者在虚拟机上打开终端输入 `rostopic list` 来查看当前的节点发布的话题

```
jubot@JUJON-VM:~$ rostopic list
/cmd_vel
/imu
/jubot/aset
/jubot/avel
/jubot/bset
/jubot/bvel
/jubot/cset
/jubot/cvel
/jubot/dset
/jubot/dvel
/jubot_driver/parameter_descriptions
/jubot_driver/parameter_updates
/odom
/rosout
/rosout_agg
/tf
/voltage
```

下面打开一个新终端或者在虚拟机上打开终端，输入 `roslaunch jubot_ctl jubot_keyboard.launch` 打开键盘控制节点：




这时候就可以按照终端上的指示通过键盘上的 u i o j k l m , . 或者小键盘旁边的上下左右键等这些按键来控制路威套件移动了。这时候可以再使用 `rosrun rqt_graph rqt_graph` 查看会得到以下关系：

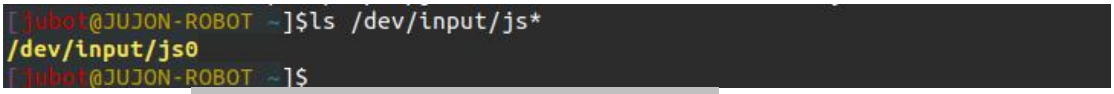



可以看出和上面的区别在于/jubot_twist_keyboard 节点发送给了/jubot_driver 这一节点一个名为/cmd_vel 的消息，说明我们的按键指令实际上就是在/jubot_twist_keyboard 节点内通过翻译转化成为/jubot_driver 驱动节点可以识别的/cmd_vel 消息，那么对于手柄来说我们可以大胆假设，可能也是通过一个节点将指令转换成/cmd_vel 消息发送给/jubot_driver驱动节点。

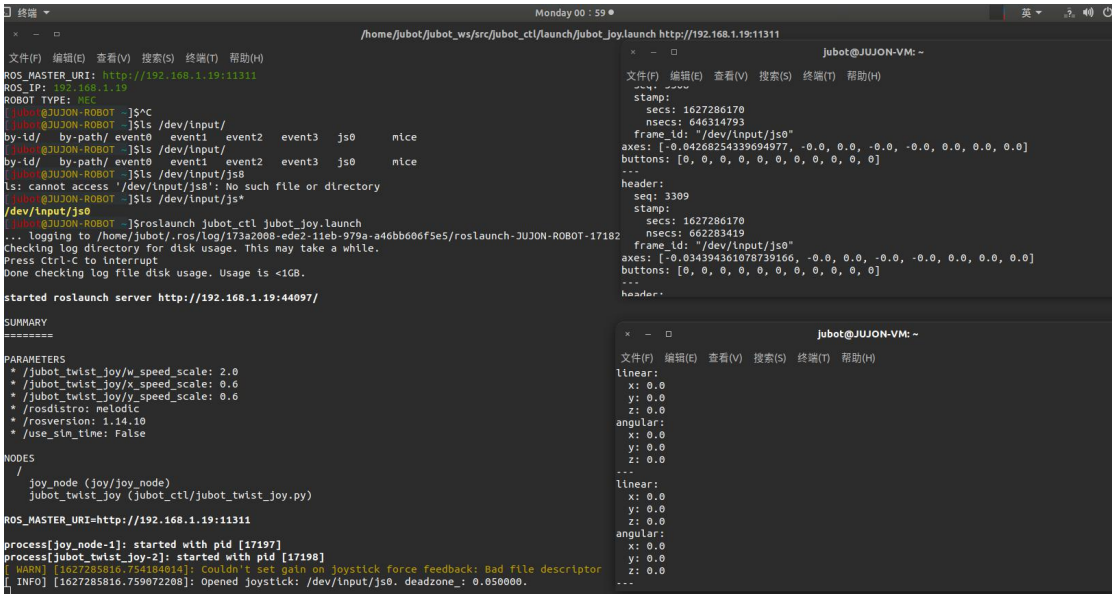
### 3.4.2 使用手柄控制路威套件

**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。



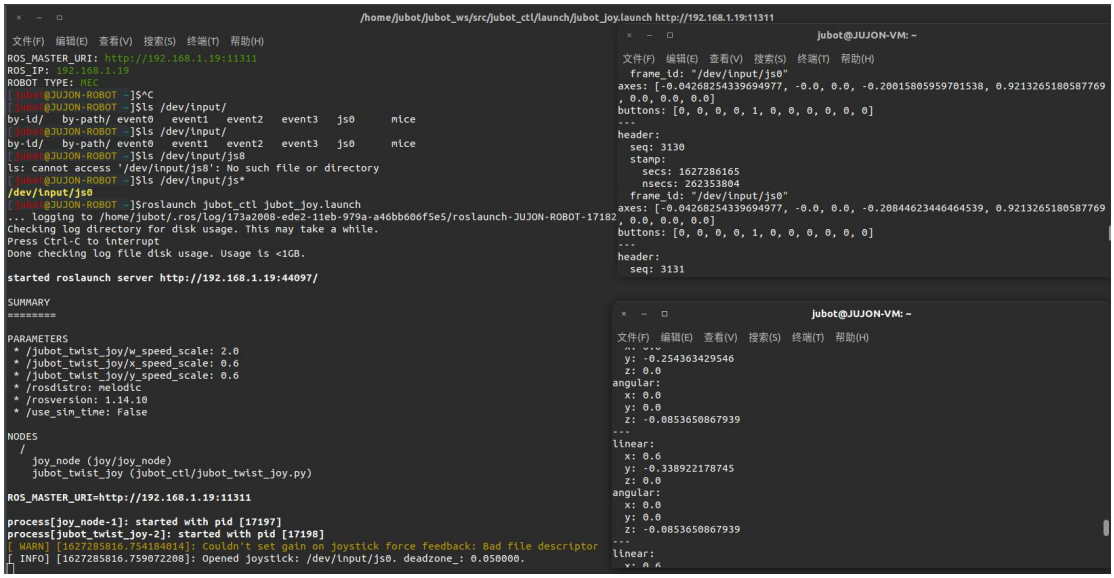

然后输入 `roslaunch jubot_ctl jubot_joy.launch`，重新打开两个个终端分别输入 `rostopic echo /joy` 和 `rostopic echo /cmd_vel` 分别监视手柄消息和速度的消息。



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 jubot_driver jubot_driver.launch` 进行底盘的连接，接着打开新终端输入 `roslaunch jubot_ctl jubot_joy.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。




### 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/jubot/jubot_ws/src/jubot_ctl/launch 下的 launch 文件 jubot_joy.launch

```
[jubot@JUJON-ROBOT ~/jubot_ws/src/jubot_ctl/launch]$pwd
/home/jubot/jubot_ws/src/jubot_ctl/launch
[jubot@JUJON-ROBOT ~/jubot_ws/src/jubot_ctl/launch]$ls
jubot_bringup_joy.launch  jubot_bringup_keyboard.launch  jubot_joy.launch  jubot_keyboard.launch
[jubot@JUJON-ROBOT ~/jubot_ws/src/jubot_ctl/launch]$cat jubot_joy.launch
<launch>
    <param name="use_sim_time" value="false"/>

    <!-- 启动手柄遥控节点 -->
    <node name="joy_node" pkg="joy" type="joy_node" output="screen" respawn="false"/>
    <node name="jubot_twist_joy" pkg="jubot_ctl" type="jubot_twist_joy.py" output="screen" respawn="false">
        <param name="x_speed_scale"   value="0.6" />
        <param name="y_speed_scale"   value="0.6" />
        <param name="w_speed_scale"   value="2.0" />
        <remap from="cmd_vel" to="cmd_vel"/>
    </node>
</launch>
[jubot@JUJON-ROBOT ~/jubot_ws/src/jubot_ctl/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，路威套件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：**

- dev 手柄的输入接口，这里是/dev/input/js0
- 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
- autorepeat_rate 操纵杆发送数据的频率

jubot_twist_joy 节点：

简介：改节点会订阅/joy 消息将其整理并发布为/cmd_vel 消息一些参数：

- axis_linear 用于线性运动控制的操纵杆轴。
- scale_linear 线性运动速度大小的控制。
- axis_angular 用于旋转运动控制的操纵杆轴。
- 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。

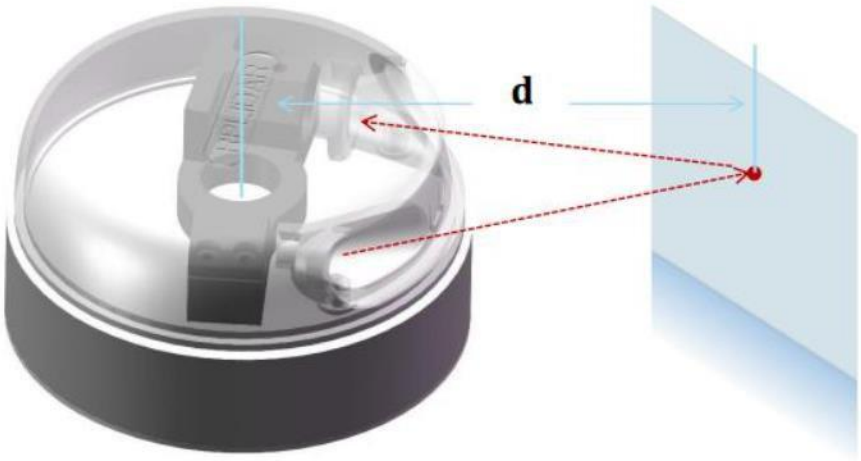
## 4.3 实验工具

个人电脑一台，路威套件及其配件。

## 4.4 实验内容

### 4.4.1 激光雷达测距原理

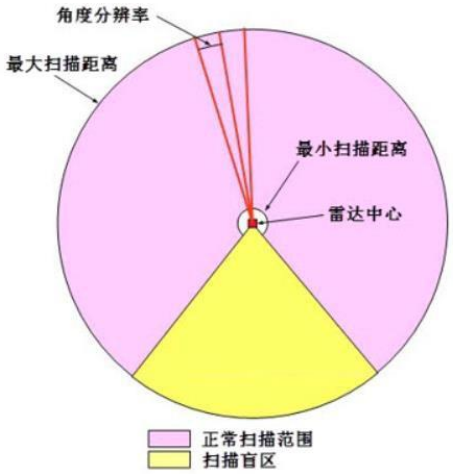
在激光雷达技术领域中，目前主要通过三角测距法与 TOF 方法来进行测距。在路威套件中我们所用的思岚激光雷达所采用的是三角测距法，以下简单介绍：




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.
```

**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

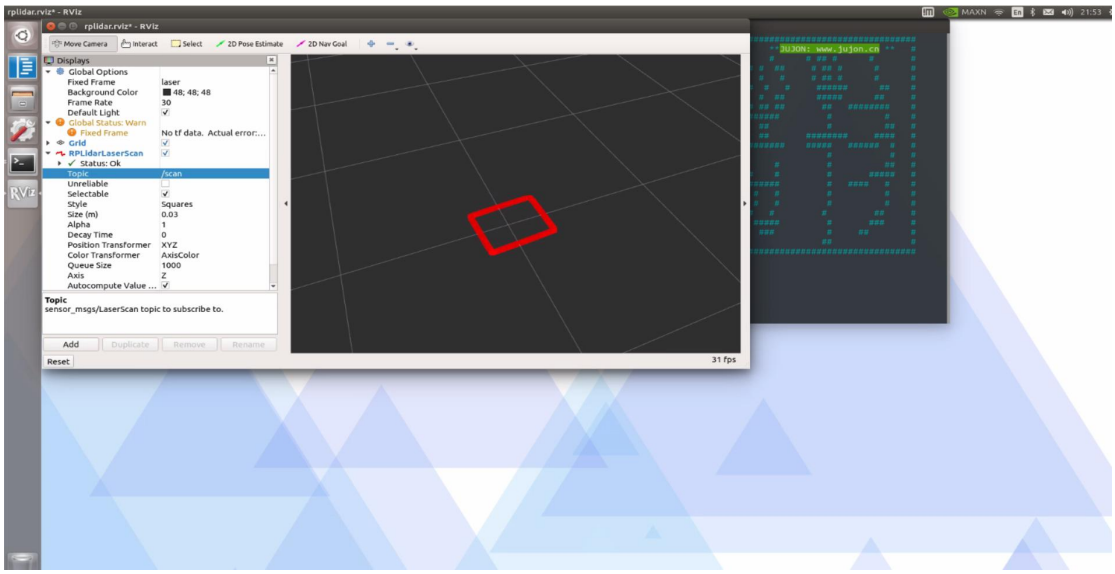
**intensities**： ranges 数据的强度，同样 360 个。

### 4.4.3 laser_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。

首先打开路威套件的激光雷达数据：

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



得到路威套件周围的障碍信息（这里为了展示效果，将路威套件围起来了）。

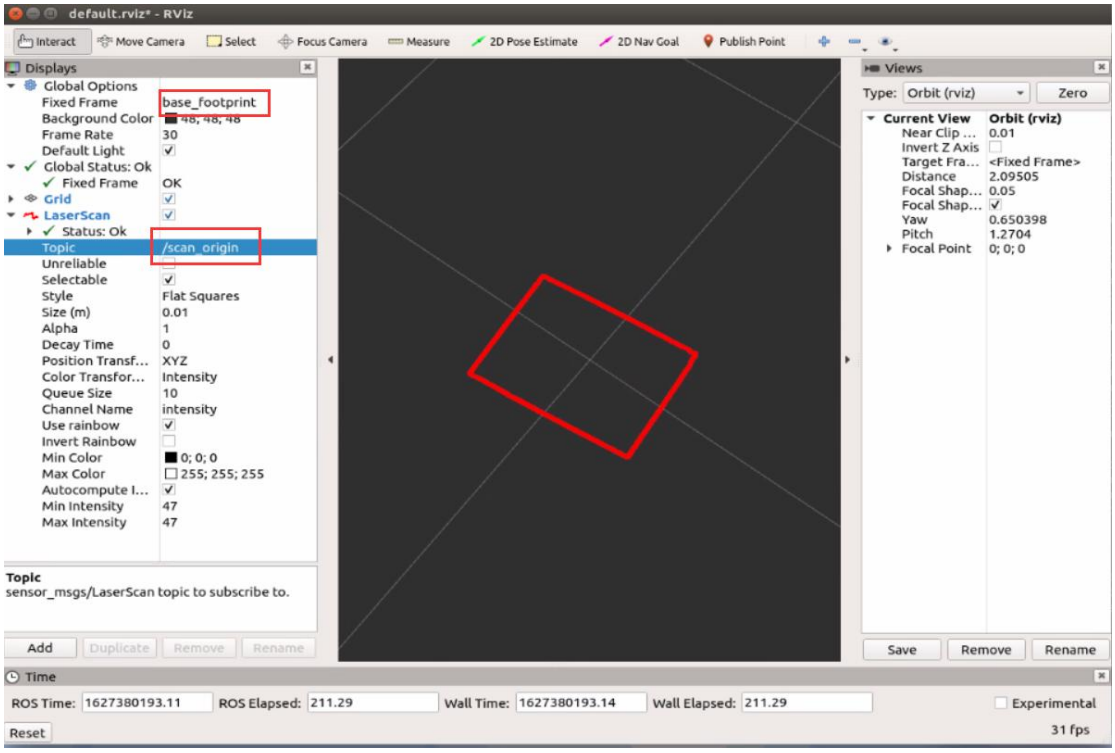
接着我们打开功能包 /home/jubot/jubot_ws/src/jubot_driver/config 下的 jubot_laserfilter.yaml 文件，在文件中修改包围盒的几何尺寸并保证 fixed_frame 一致：

```
[jubot@JUJON-ROBOT ~/jubot_ws/src/jubot_driver/config]$ls
camera_calib  cfg  jubot_laserfilter.yaml  jubot_params.yaml
[jubot@JUJON-ROBOT ~/jubot_ws/src/jubot_driver/config]$cat jubot_laserfilter.yaml
scan_filter_chain:
- name: box_filter
  type: laser_filters/LaserScanBoxFilter
  params:
    box_frame: base_footprint
    min_x: -0.12
    max_x: 0.12
    min_y: -0.12
    max_y: 0.12
    min_z: -0.2
    max_z: 0.2
[jubot@JUJON-ROBOT ~/jubot_ws/src/jubot_driver/config]$
```

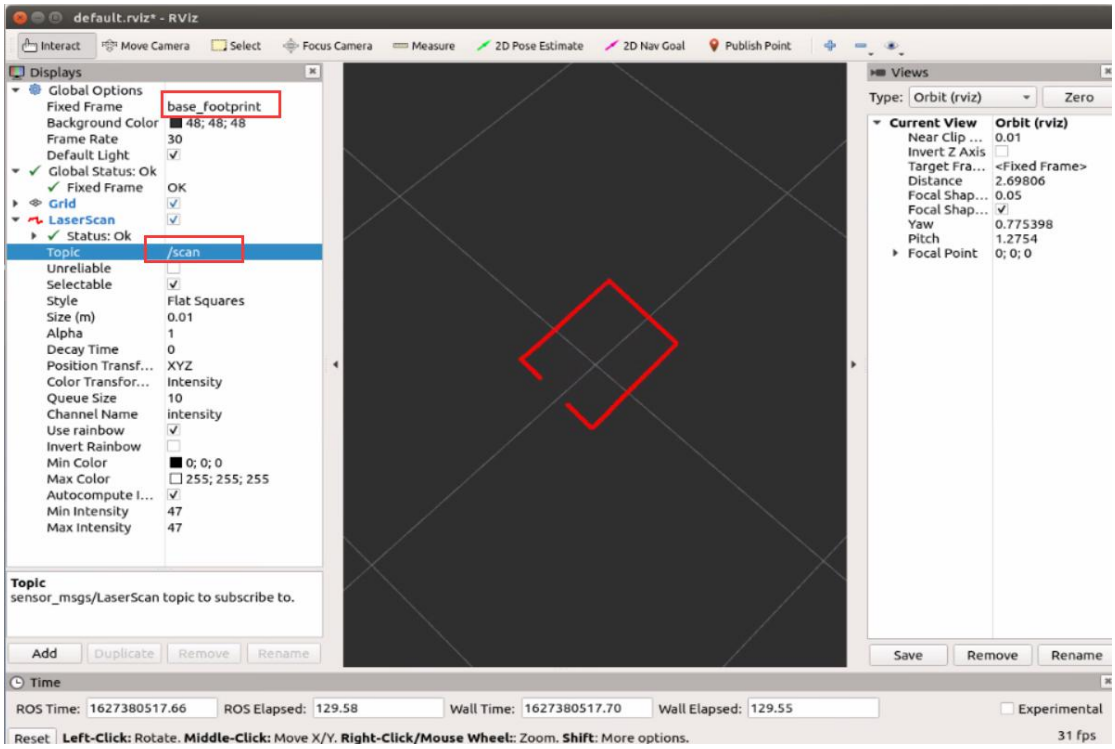
在该包围盒内的激光雷达数据将被屏蔽，保存之后打开新终端，输入：

`roslaunch jubot_driver jubot_lidar.launch`

使用 `rosrun rviz rviz` 打开 rviz观察：




目前并没有什么变化，点击 topic 后选择/scan,得到以下结果：




可以得到 /scan 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

根据实验步骤完成路威套件遇障停止的功能。

## 5.3 实验工具

个人电脑一台，路威套件及其配件。

## 5.4 实验内容

### 5.4.1 功能要求与分析

要求路威套件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 路威套件 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，让路威套件停止即可。（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 python 文件来实现按下一个按键后路威套件直线运动，核心代码段如下（该文件位置在~/jubot_ws/src/jubot_ctl/scripts下）：

```
if __name__ == "__main__":
    if os.name != 'nt':
        settings = termios.tcgetattr(sys.stdin)
    rospy.init_node('jubot_go_and_stop')
    pub = rospy.Publisher('cmd_vel', Twist, queue_size=10)
    sub = rospy.Subscriber("scan", 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("aaaa")
                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
```

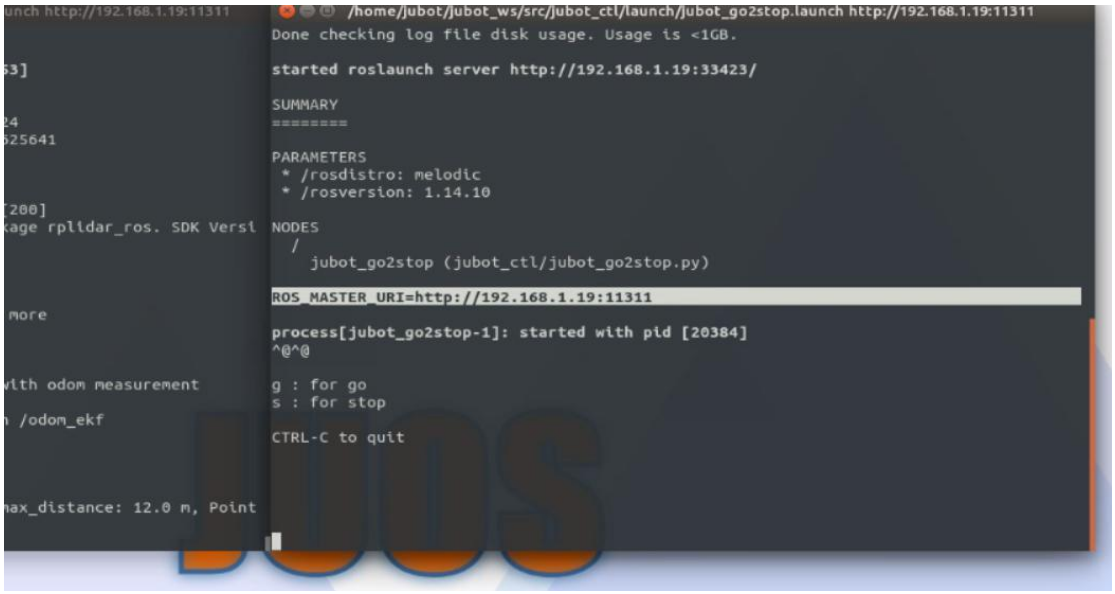
```
            if flag:
                twist.linear.x = 0
                print("STOP!!!!!!!!!!!!!!!\r\n")
            pub.publish(twist)
    except:
        print(e)
    finally:
        twist = Twist()
        twist.linear.x = 0.0; twist.linear.y = 0.0; twist.linear.z = 0.0
        twist.angular.x = 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

```
<launch>
    <node name="jubot_go2stop" pkg="jubot_ctl" type="jubot_go2stop.py" output="screen" respawn="false"/>
</launch>
```

接着就可以打开终端，先运行 `roslaunch jubot_driver jubot_bringup.launch` 连接地盘然后打开新终端，运行 `roslaunch jubot_ctl go_and_stop.launch`

即可得到如下界面：




这时只要按下键盘的 g，路威套件就会开始向前移动，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

完成了路威套件自主运动，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

能够理解 map、odom、base_footprint、base_link、laser 坐标系以及坐标系间的关系。

能够理解四轮差分运动模型的运动学分析。

能够理解 tf 框架的作用。

## 6.3 实验工具

个人电脑一台，路威套件。

## 6.4 实验内容

### 6.4.1 ROS 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_footprint 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，当然有些机器人是 base_link,其实是一个意思。机器人的定位问题最本质的就是求出 base_footprint 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_footprint 的 tf 是固定的。

## base_link 坐标系

机器人本体坐标系,但在路威套件中为了方便显示模型，我们以base_footprint坐标系为机器人本身位姿坐标系。

### 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，路威套件标准版采用的是四轮差速移动的运动方式。四轮差速底盘有四个独立的动力轮位于底盘两侧，通过给两侧轮子设定不同大小、方向的转速来实现机器人的前进、后退、以及转向的功能。

## 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。

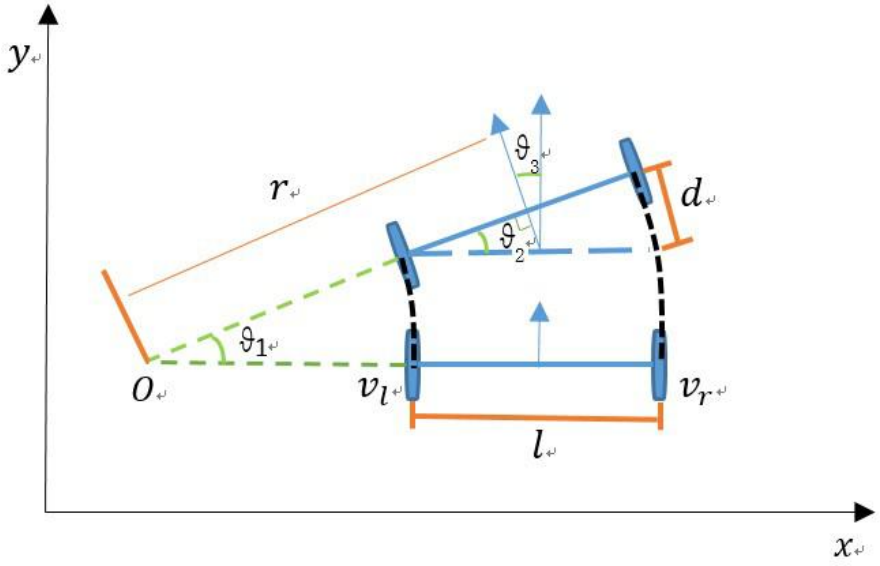
2. 其次，ROS 机器人不仅能实现手动遥控的任务，他还能自行的建图与导航，而实现这些功能的时候，必须要有底盘的里程计信息，所谓的里程计能够记录自己从运动开始到当前位置所产生的位移。这就需要有从各个电机的转速到机器人位移的变换关系。

## 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。所以我们可以建立抽象运动学模型如下：




如图是移动机器人在两个相邻时刻的位姿，其中$\theta_1$是两相邻时刻移动机器人绕圆弧运动的角度，$\theta_3$是两相邻时刻移动机器航向角（朝向角 head）的变化量。$l$是左右轮之间的间距，d是右轮比左轮多走的距离。r是移动机器人圆弧运动的半径。

单纯考虑机器人的 X 轴前进线速度，可以很直观的看到，移动机器人前进的速度就等于左右轮速度的平均：

$$V = \frac{V_r + V_l}{2}$$

对于机器人的航向角，如图所示，把两个时刻的机器人位置叠加在一起，可以清楚的看到移动机器人航向角变化量是$\theta_3$。从图中的几何关系可以得到：

$$\theta_1 = \theta_2 = \theta_3$$

也就是说移动机器人航向角变化了多少角度，它就绕其运动轨迹的圆心旋转了多少角度。这句话很好验证，我们让机器人做圆周运动，从起点出发绕圆心一圈回到起点处，在这过程中机器人累计的航向角为 360 度，同时它也确实绕轨迹圆心运动了 360 度，说明机器人航向角变化多少度，就绕圆心旋转了多少度。而这三个角度中,$\theta_2$很容易计算出来，由于相邻时刻时间很短，角度变化量$\theta_2$很小，有下面的近似公式：

$$\theta_2 \approx \sin\theta = \frac{d}{l} = \frac{(V_r - V_l)\cdot\Delta t}{l}$$

所以可以得到机器人绕圆心运动的角速度ω，它也是机器人航向角变化的速度：

$$r = \frac{v}{w} = \frac{l\cdot(V_r + V_l)}{2\cdot(V_r - V_l)}$$

从公式$r = \frac{v}{w} = \frac{l\cdot(V_r+V_l)}{2\cdot(V_r-V_l)}$可以发现当左轮速度等于右轮速度时，半径无穷大，即直线运动。最后将三个公式综合起来，可以得到左右轮速度和线速度角速度之间的关系如下：

$$V = \frac{V_r + V_l}{2}$$

机器人线速度（m/s）

$$W = \frac{V_r - V_l}{l}$$

机器人角速度（rad/s）

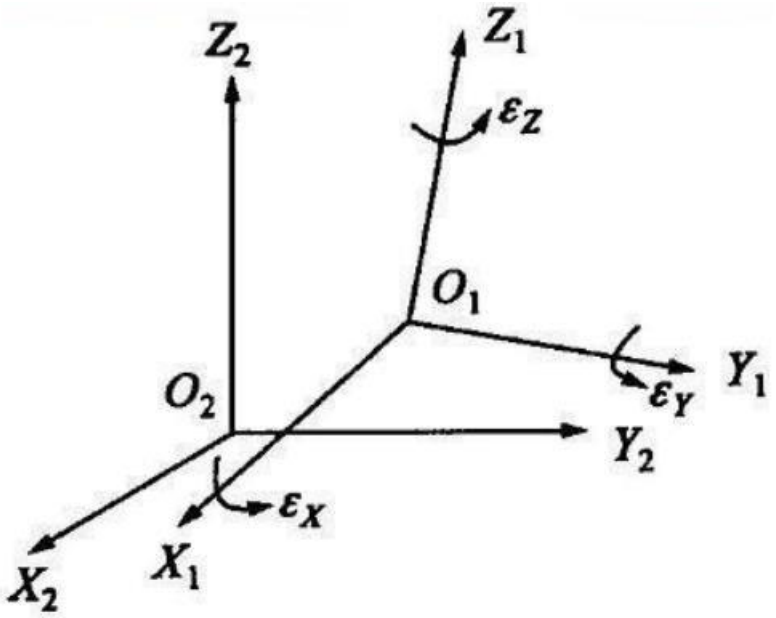
$$r = \frac{v}{w} = \frac{l\cdot(V_r + V_l)}{2\cdot(V_r - V_l)}$$

机器人旋转半径 r

至此，两轮差速运动模型运动学推导完成，通过以上公式，即可利用编码器获取的两侧轮子速度推导出机器人的角速度和线速度，进一步对此角速度以及线速度在时间上进行积分，即可得到机器人移动的距离，从而可以得到机器人里程计定位的坐标。

### 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。



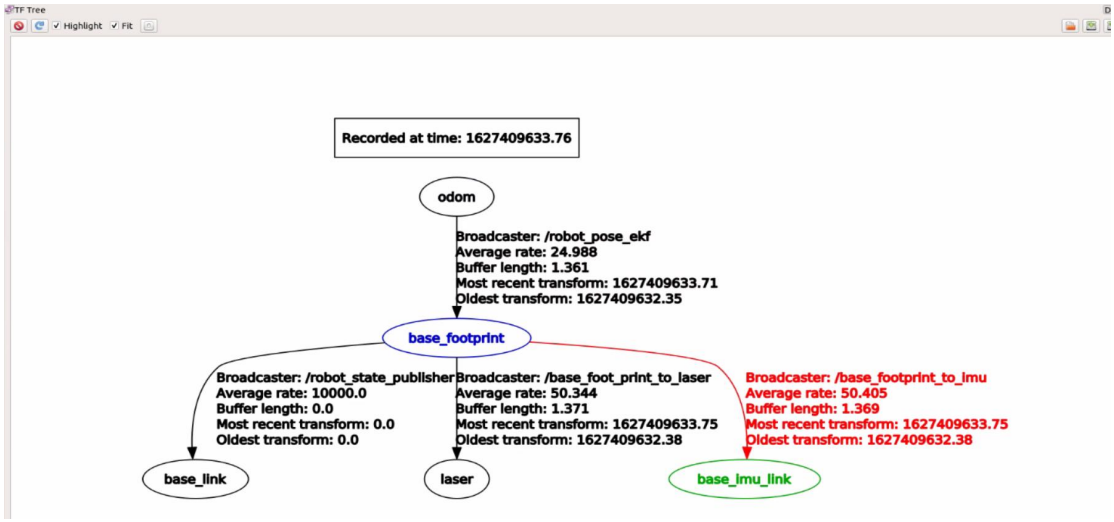

如图，坐标系01 可以经由坐标系02 的平移和旋转得到，其理论与原理不在此过多描述。

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：

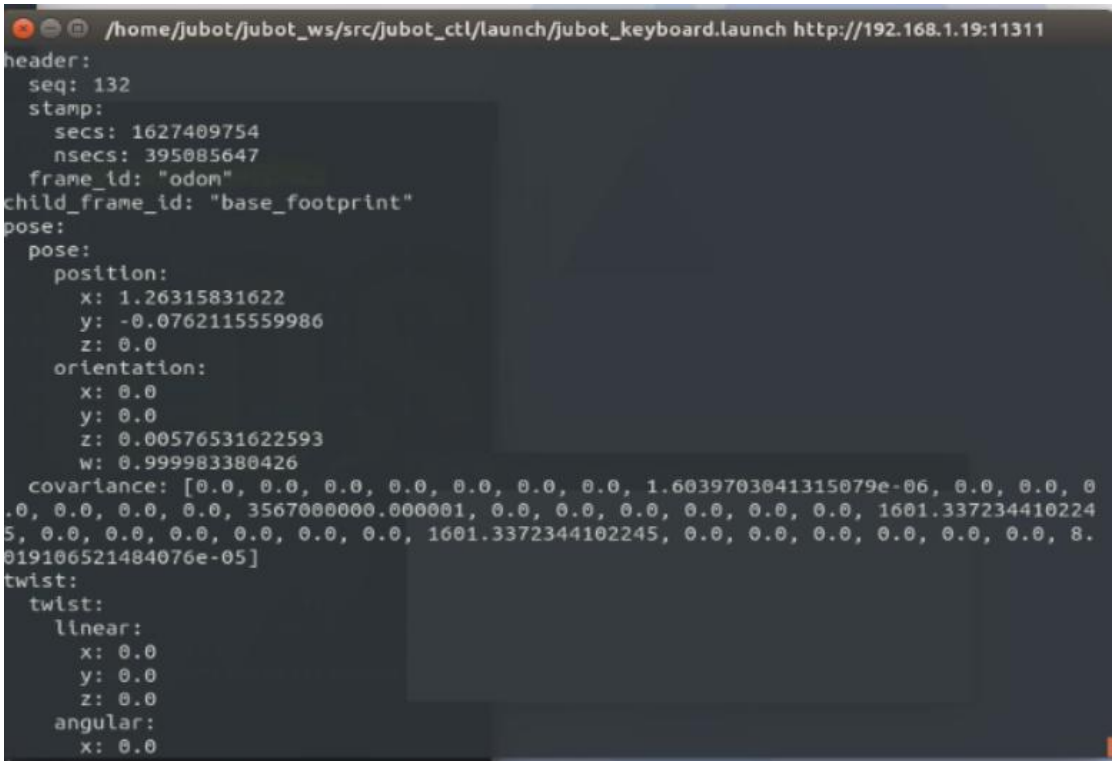
1. 监听 TF 变换
2. 接收并缓存系统中发布的所有坐标变换关系，并从中查询所需要的坐标变换关系。
3. 广播 TF 变换
4. 向系统中广播坐标系之间的坐标变换关系。系统中可能会存在多个不同部分的 TF 变换广播，每个广播都可以直接将坐标变换关系插入到 TF 树中，不用再进行同步。
5. 简单来说 tf 以树的形式记录着各个坐标系间的变换方程，在实际使用的时候可以直接计算出所需要的两个坐标系间的坐标关系。
6. 运行连接底盘程序之后可以新开启一个终端，输入

```
rosrun rqt_tf_tree rqt_tf_tree
```

得到当前的 tf 树如图：




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




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。

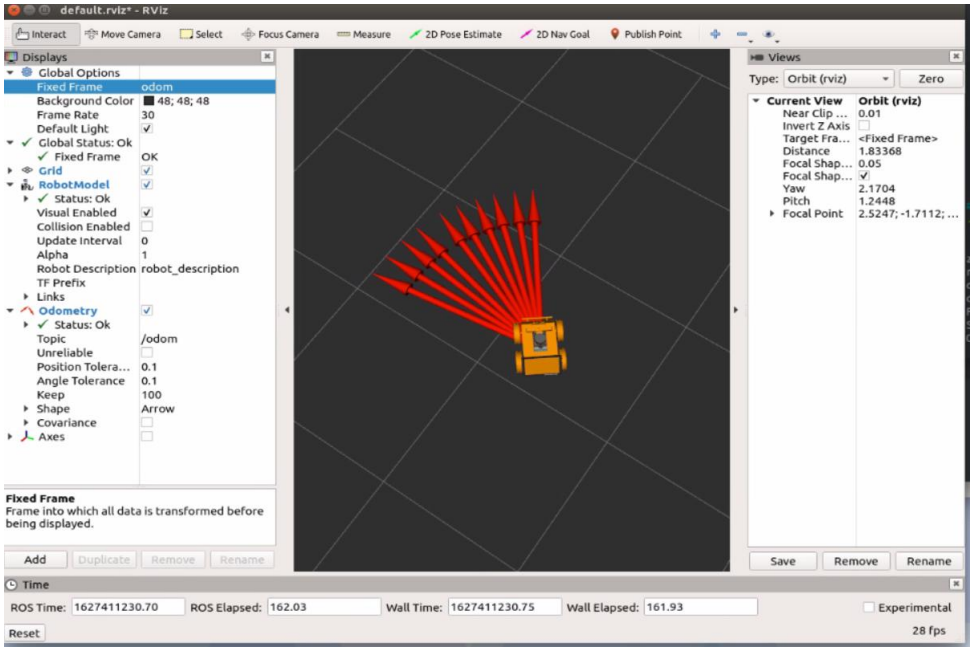
我们在 jubot_dirver 功能包中实现一个节点，来获取当前里程计与 tf 坐标变换信息。这个节点名称为get_pose_demo.py，源码如下所示：

```
#!/usr/bin/env python
import rospy
from tf_conversions import transformations
from * import transformations
from math import pi
import tf
class Robot:
    def init (self):
        self.tf_listener = tf.TransformListener()
        print("aaa")
        try:
            self.tf_listener.waitForTransform('/odom', '/base_footprint', rospy.Time(), rospy.Duration(1.0))
        except (tf.Exception, tf.ConnectivityException, tf.LookupException): return
    def get_pos(self):
        self.tf_listener = tf.TransformListener()
        try:
            self.tf_listener.waitForTransform('/odom', '/base_footprint', rospy.Time(), rospy.Duration(1.0))
        except (tf.Exception, tf.ConnectivityException, tf.LookupException):
            return
        try:
            (trans, rot) = self.tf_listener.lookupTransform('/odom', '/base_footprint', rospy.Time(0))
        except (tf.LookupException, tf.ConnectivityException, tf.ExtrapolationException):
            rospy.loginfo("tf Error")
            return None
        euler = transformations.euler_from_quaternion(rot)
        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 代表机器人机头朝向（左正右负）。首先打开终端启动底盘通信 `roslaunch jubot_driver jubot_bringup.launch`，然后打开新的终端输入 `rosrun jubot_driver get_pose_demo.py`，这样就获取到了当前的里程计信息。

```
[jubot@JUJON-ROBOT ~]$rosrun  jubot_driver get_pose_demo.py
(2.655497456176737, -2.166216106088109, 14.7168985649406)
(2.655497456176737, -2.166216106088109, 14.7168985649406)
(2.655497456176737, -2.166216106088109, 14.7168985649406)
(2.655497456176737, -2.166216106088109, 14.7168985649406)
(2.655497456176737, -2.166216106088109, 14.7168985649406)
(2.655497456176737, -2.166216106088109, 14.7168985649406)
(2.655497456176737, -2.166216106088109, 14.7168985649406)
(2.655497456176737, -2.166216106088109, 14.7168985649406)
(2.655497456176737, -2.166216106088109, 14.7168985649406)
(2.655497456176737, -2.166216106088109, 14.7168985649406)
(2.655497456176737, -2.166216106088109, 14.726898064794316)
(2.6554974561767364, -2.166216106088109, 14.726898064794316)
^C(2.6554974561767355, -2.166216106088109, 14.716901156000048)
```

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，并能推导出路威套件中 odom topic 中的内容来源。

## 6.6 实验报告

实验目的

实验要求

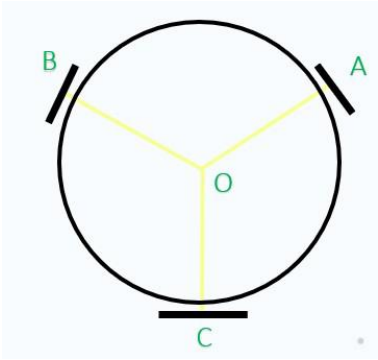
实验内容

实验总结

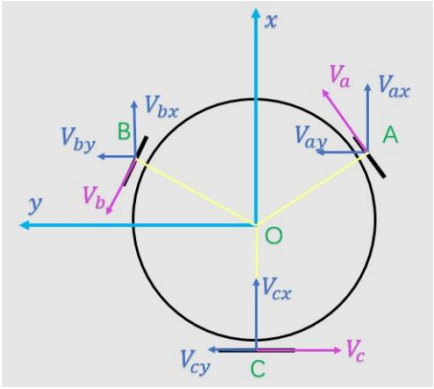
思考题

## 6.7 思考题

**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：

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

# 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，路威套件 。

## 7.4 实验内容

### 7.4.1 IMU 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。

### 7.4.2 IMU 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 jubot_driver jubot_bringup.launch
```

连接机器人的底盘，然后打开新终端输入 `rostopic info /imu` 得到以下信息：

```
[jubot@JUJON-ROBOT ~]$rostopic info  /imu
Type: sensor_msgs/Imu

Publishers:
 * /jubot_driver (http://192.168.1.19:39951/)

Subscribers:
 * /robot_pose_ekf (http://192.168.1.19:46573/)
```

表示该话题发布的消息为/sensor_msgs/Imu，话题的发布者为/jubot_driver，订阅者为/robot_pose_ekf。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：

```
[jubot@JUJON-ROBOT ~]$rosmsg info sensor_msgs/Imu
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 查看一下实际的数据是什么样的：

```
[jubot@JUJON-ROBOT ~]$rostopic echo /imu
header:
  seq: 39225
  stamp:
    secs: 1627454951
    nsecs: 202961162
  frame_id: "base_imu_link"
orientation:
  x: 0.0
  y: 0.0
  z: 0.0633130753927
  w: 0.997993714652
orientation_covariance: [1000000.0, 0.0, 0.0, 0.0, 1000000.0, 0.0, 0.0, 0.0, 0.05]
angular_velocity:
  x: 0.0
  y: -0.00106523301866
  z: -0.00106523301866
angular_velocity_covariance: [1000000.0, 0.0, 0.0, 0.0, 1000000.0, 0.0, 0.0, 0.0, 100
0000.0]
linear_acceleration:
  x: 0.678295898438
  y: 0.429467773438
  z: 9.857421875
linear_acceleration_covariance: [0.01, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
---
header:
  seq: 39226
  stamp:
```

### 7.4.3 robot_pose_ekf 实现 IMU 校准里程计

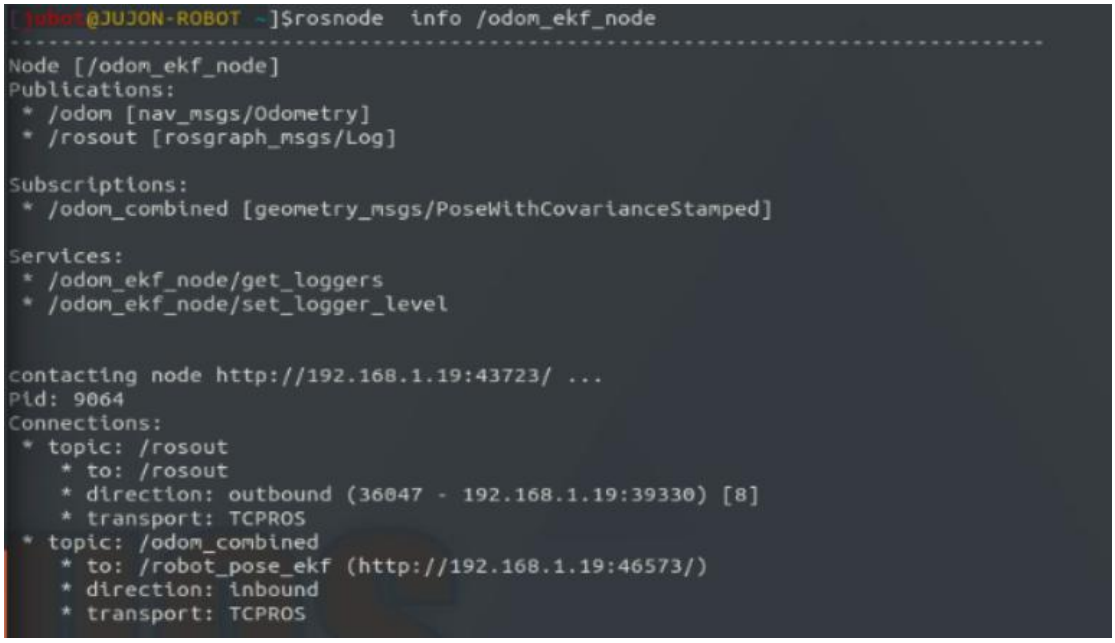
IMU 数据的使用方法有两种，一种简单的应用就是直接使用 IMU 数据中的航向角度来计算机器人的转角，这样的话，即使打滑或者抬起机器人转一定的角度，所得到的里程也能正确的反映出来。

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 /odom_combined`，来查看该话题：

```
[jubot@JUJON-ROBOT ~]$rostopic info /odom_combined
Type: geometry_msgs/PoseWithCovarianceStamped

Publishers:
 * /robot_pose_ekf (http://192.168.1.19:46573/)

Subscribers:
 * /odom_ekf_node (http://192.168.1.19:43723/)
```

输入 `rosnode info /odom_ekf_node` 查看节点信息：

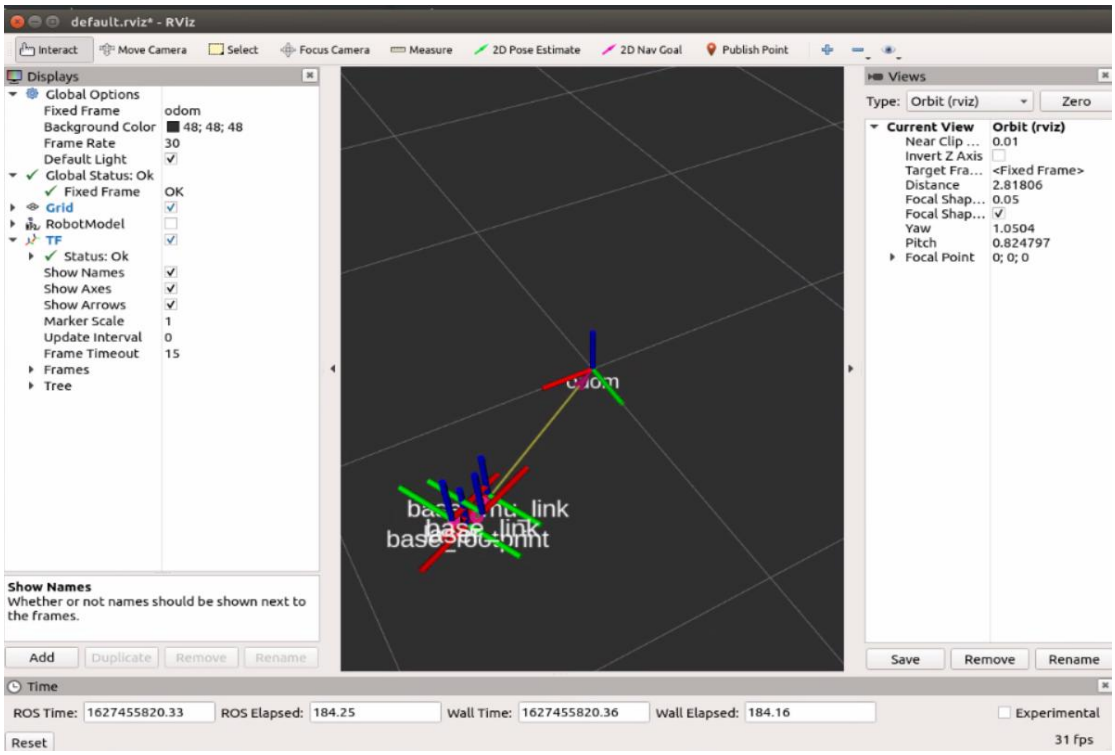



可以看到该节点会发布一个里程计的 topic。

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路威套件运动，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：

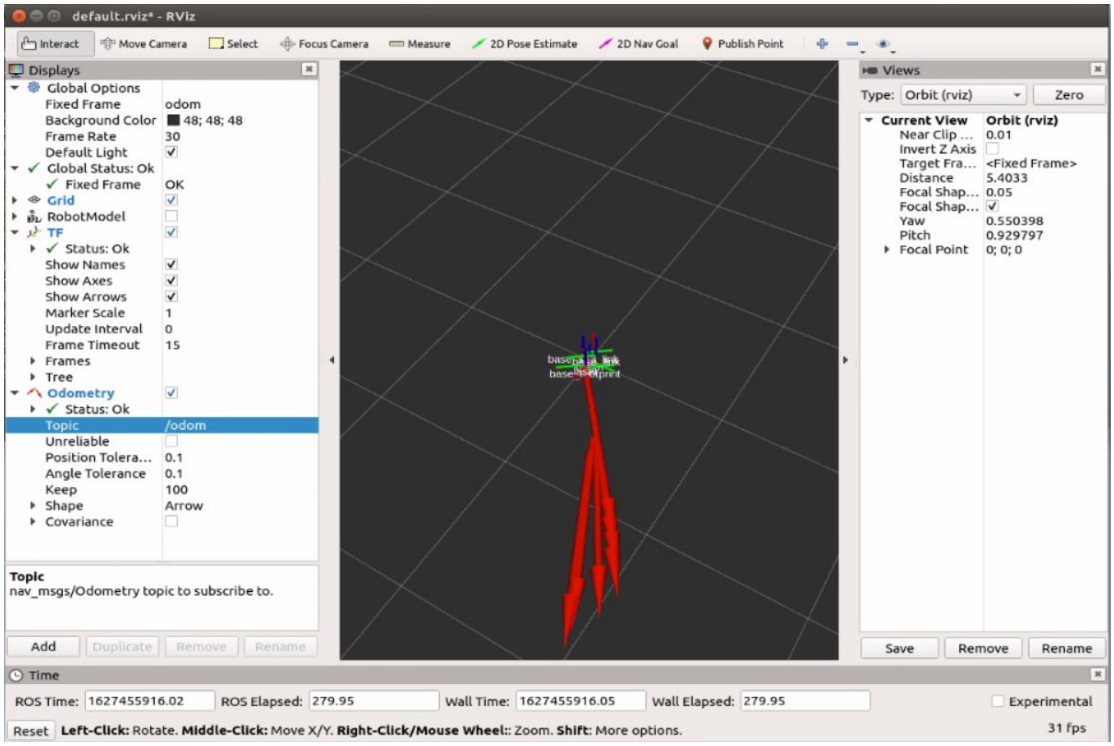
- 手动控制机器人走一圈然后回到之前的原点，通过观察模拟器(RViz)中里程与初始点的偏差
- 程序控制机器人行走(例如走一个方形)，通过观察最终机器人时间与最初原点的偏差

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。下面我们手动控制路威套件走一圈回到原点，可以在 rviz 中看到



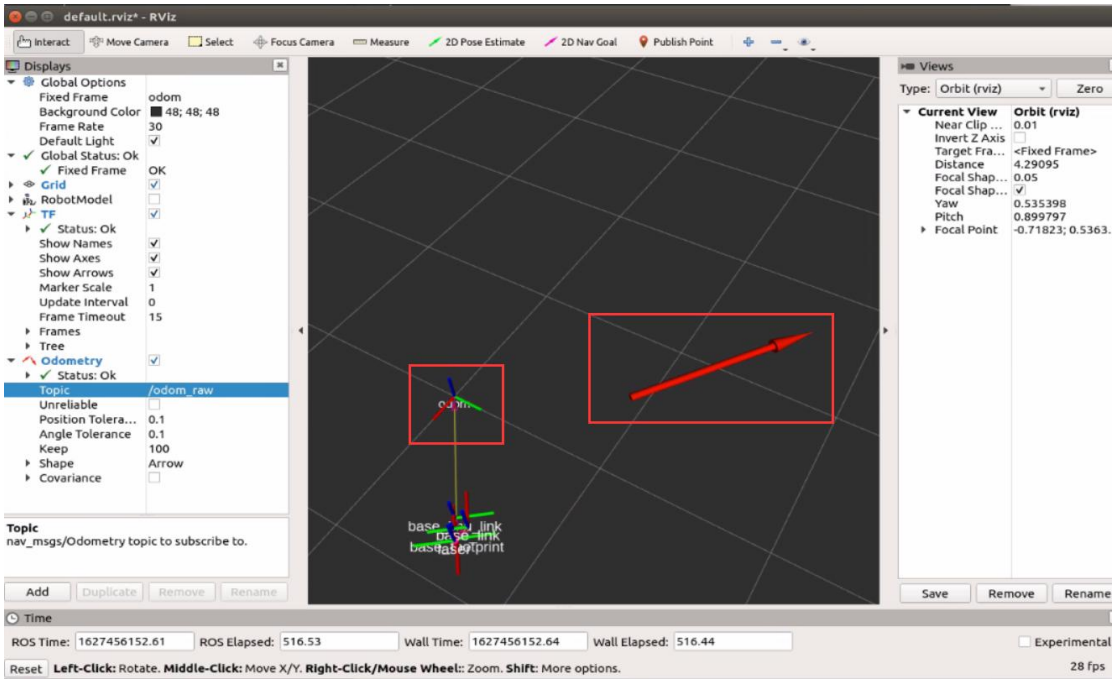

位置和姿态都存在一定误差。

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)，同上面我们控制路威套件走一圈回到原点。

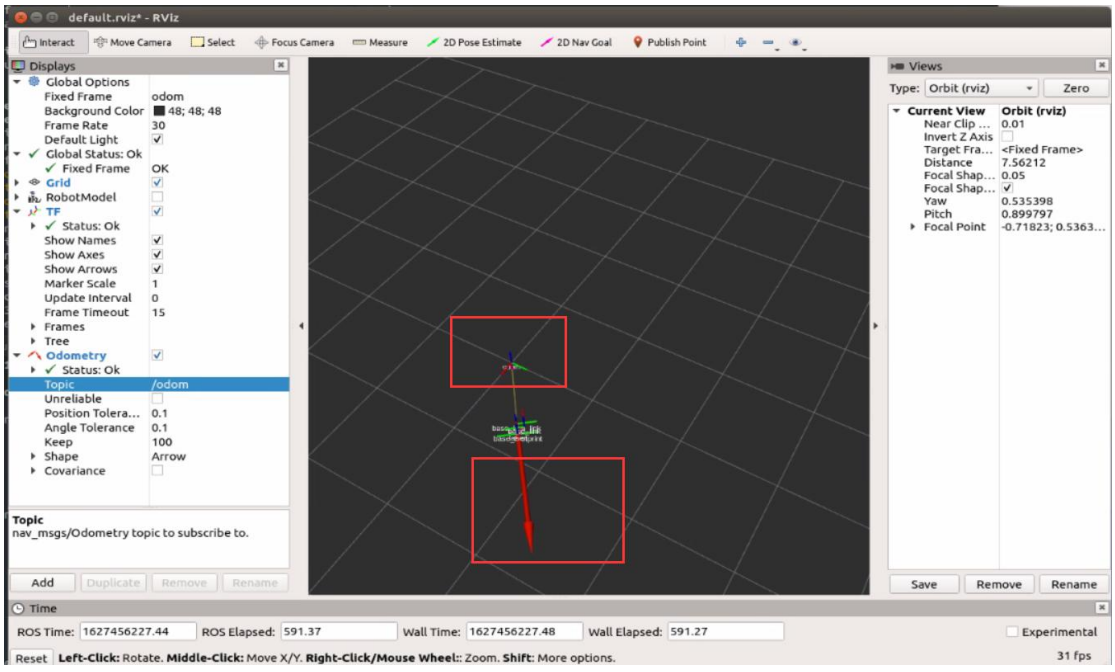



初始的时候坐标原点，轮式里程计（红色箭头），融合后的里程计（base_footprint）

的 tf 变换是重合的。




可以发现没走多远融合出来的里程和编码器里程就有了较大的差距。




我们继续控制使得路威套件回到原点。

## 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 基于里程计的运动标定

## 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，路威套件，米尺。

## 8.4 实验内容

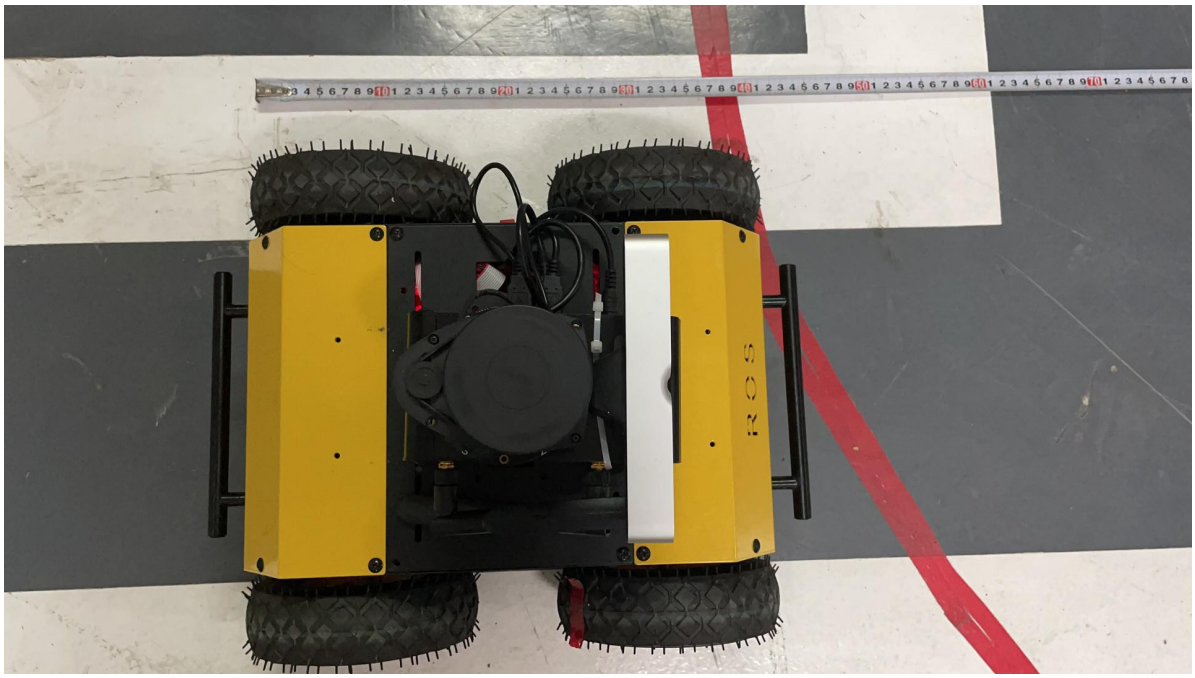
进行基于里程计的运动控制。

### 8.4.1 进行基于里程计的线速度标定

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，然后控制路威套件运动指定距离，这里我们设为 1 米。

下面来看一下具体的操作：

需提前用卷尺在底板上画一条 1 米长的直线，将卷成放在线旁，如下图所示。



新建终端，SSH 连接到机器人，运行线速度校准节点。

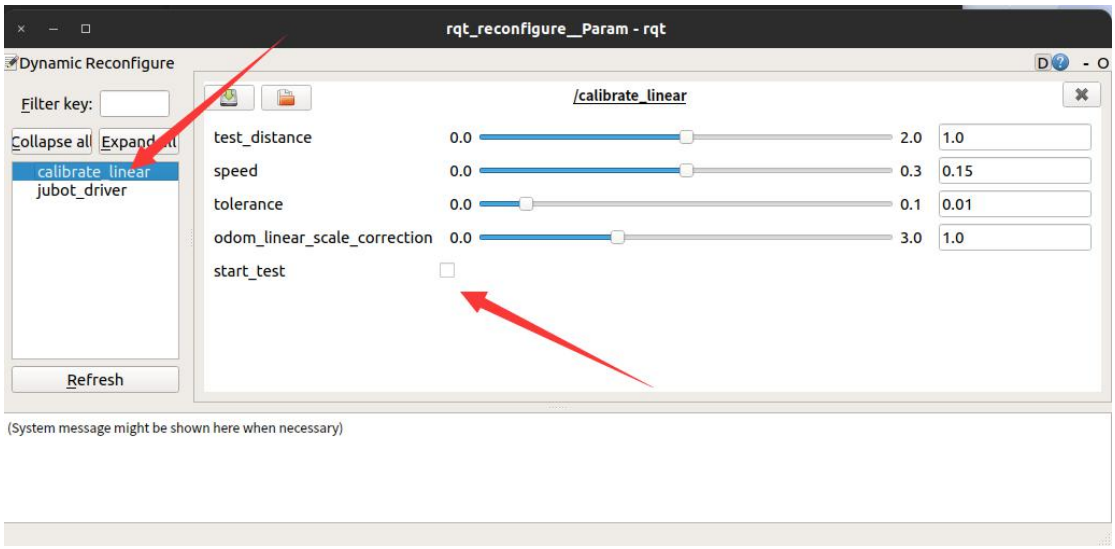
```
roslaunch jubot_driver jubot_calibrate_linear.launch
```

```
NODES
  /
    base_footprint_to_imu (tf/static_transform_publisher)
    calibrate_linear (jubot_driver/calibrate_linear.py)
    jubot_driver (jubot_driver/jubot_driver)

auto-starting new master
process[master]: started with pid [3000]
ROS_MASTER_URI=http://192.168.1.30:11311

setting /run_id to 32dcc822-fb4f-11eb-9591-a46bb6068b05
process[rosout-1]: started with pid [3011]
started core service [/rosout]
process[jubot_driver-2]: started with pid [3014]
process[calibrate_linear-3]: started with pid [3015]
process[base_footprint_to_imu-4]: started with pid [3020]
1.120000 0.890000
[ INFO] [1628760319.384697476]: Send Time: 1628760319.384552
[ INFO] [1628760319.387675839]: Send End Time: 1628760319.387580
[ INFO] [1628760319.399490227]: Set PID P:[250], I:[0], D:[150]
[ INFO] [1628760319.404369338]: Robot Running!
[INFO] [1628760324.840027]: Bring up rqt_reconfigure to control the test.
```

出现上述提示后，再新建一个终端，在虚拟机端运行 rqt 可视化工具。

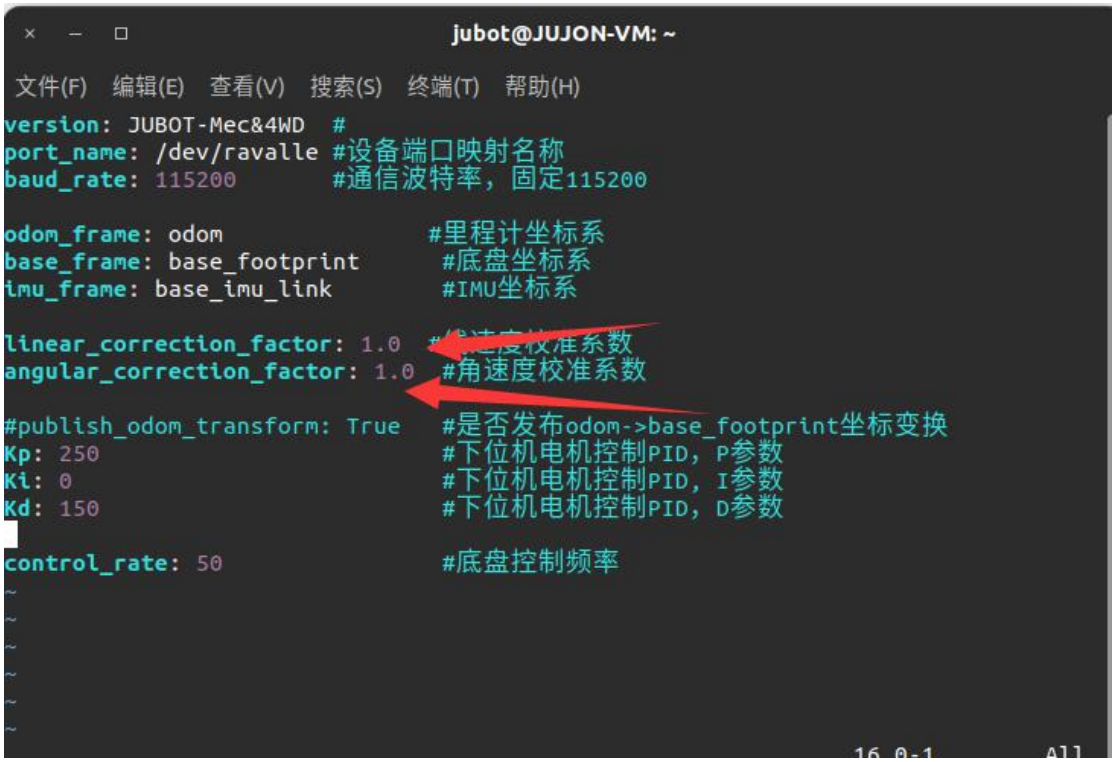
```
rosrun rqt_reconfigure rqt_reconfigure
```

点击左侧 calibrate_linear 选项，界面如下。test_distance 为测试距离，speed 为速度，tolerance 为允许误差，这三个参数一般为固定值不需要修改，odom_linear_scale_correction 为比例系数，是线速度校准的参数。



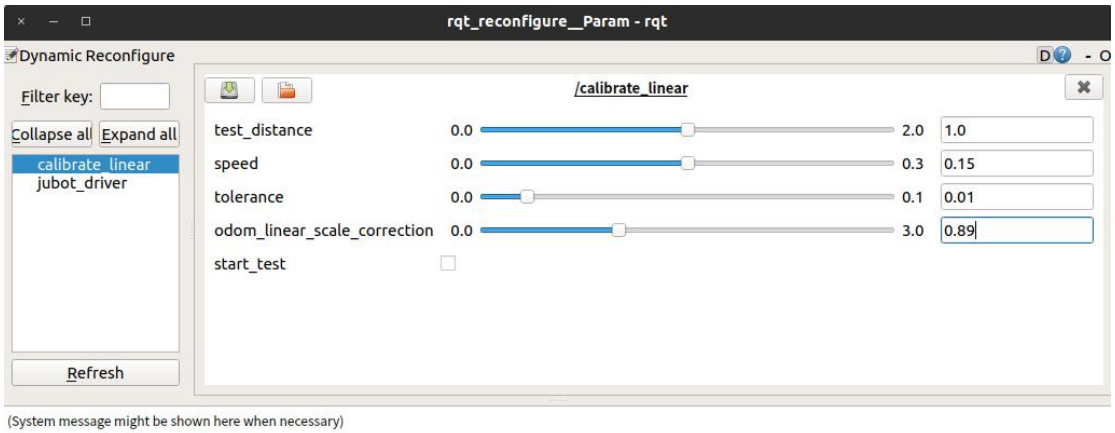

**校准流程**

步骤一，校准前先修改校正参数 linear_correction_factor 为 1.000，出厂默认为 1.000，如果用户之前已修改，请改回默认值 1.000 后再校准。




步骤二，点击 rqt_reconfigure 工具的 start_test 复选框，机器人开始自行向前行驶，目标距离为 1 米。

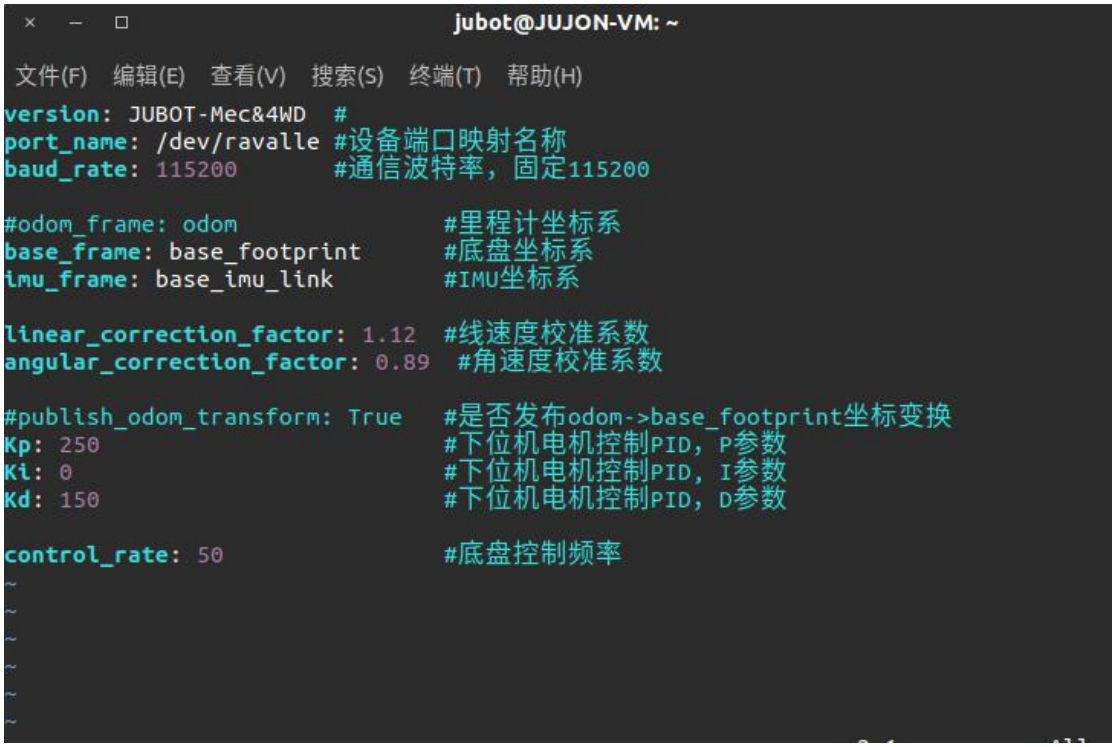
步骤三，机器人停止时，记录机器人实际的行驶距离，修改 odom_linear_scale_correction 比例参数，odom_linear_scale_correction 等于实际机器人行驶距离，可通过卷尺测量获得。




步骤四，修改后，再次重复步骤二，此时机器人行驶距离会更接近 1m。如果还存在偏差，适当修正 odom_linear_scale_correction 值后再次测试，直至机器人能准确行走 1m 的距离。如果机器人行驶距离超过 1m 可适当加大 odom_linear_scale_correction 参数再测试，反之减小 odom_linear_scale_correction 参数测试。最后记录下机器人准确行驶到 1m 时

odom_linear_scale_correction 参数值。

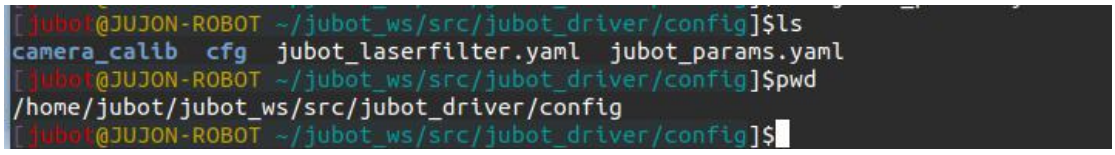
步骤五，使用 vi 修改线速度校正参数 linear_correction_factor，参数位于 jubot_driver 工具包 config 文件夹内的 jubot_params.yaml 配置文件中，修改 linear_correction_factor 为前面修正的 odom_linear_scale_correction 值。



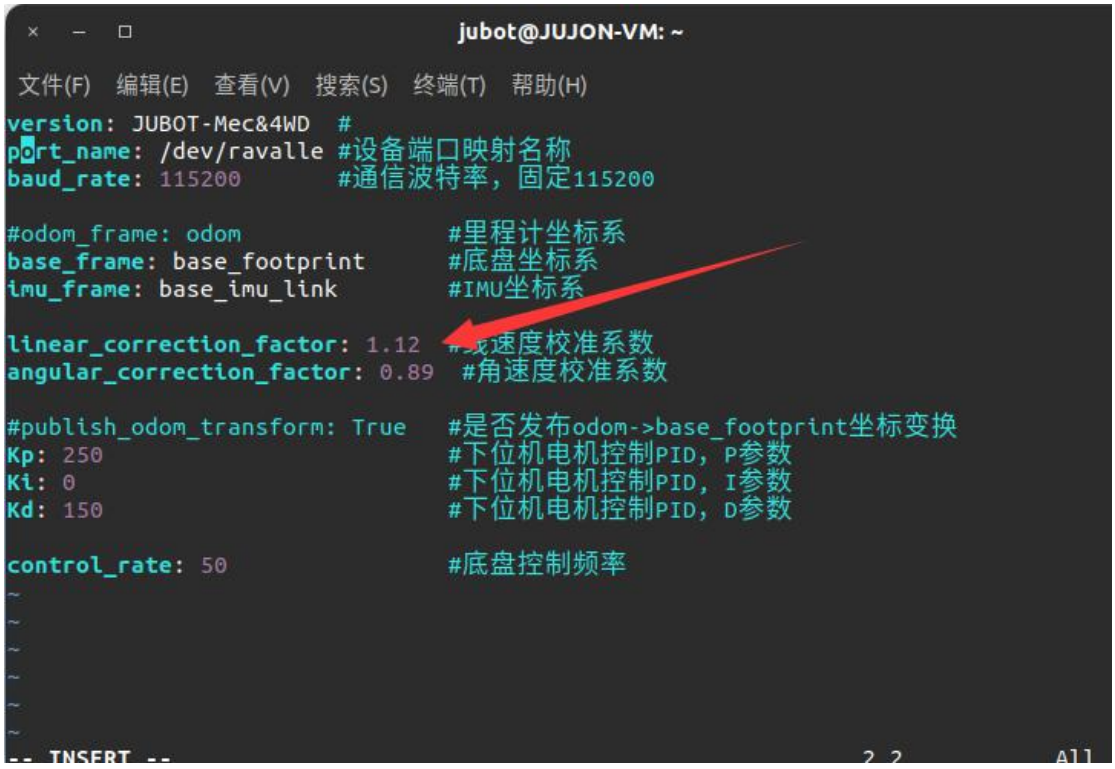

此时，机器人线速度校准完成，可以再次运行线速度校准程序，检验校准是否正确。特别说明，参数生效需要关闭并重新启动 jubot_driver 驱动包。

使用 vi 编辑器修改 jubot_params.yaml 文件方法如下。

新建终端，SSH 连接到机器人，进入 config 文件夹，然后利用 vi 编辑器，编辑 jubot_params.yaml 配置文件



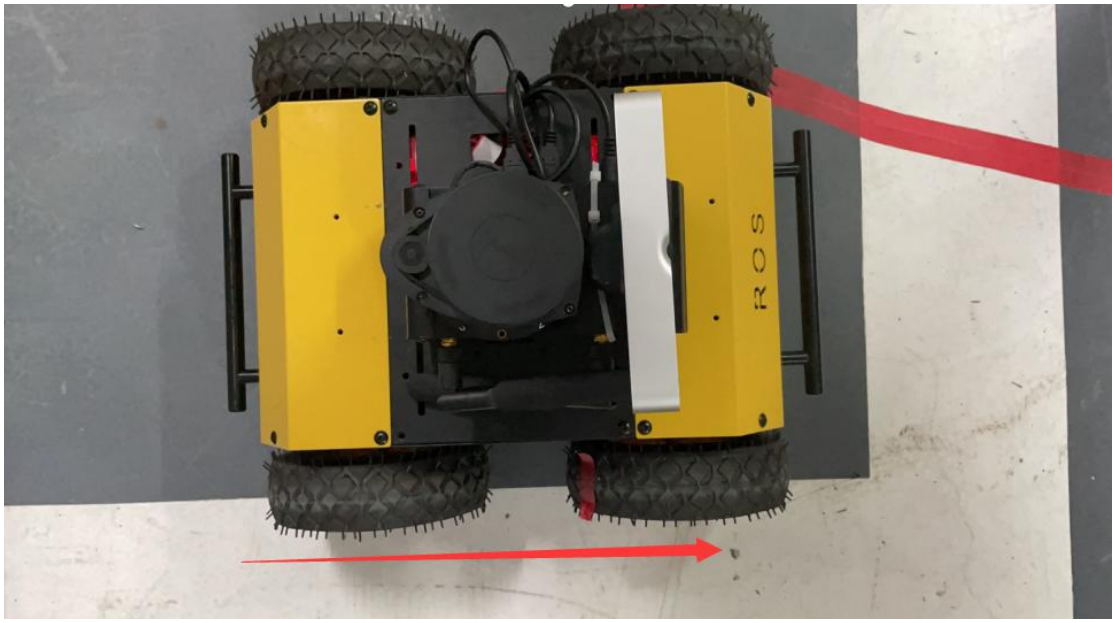

打开后，按键"i"进入 vi 编辑模式，光标移动到 linear_correction_factor 参数位置，即可修改参数值。




修改完成后，Esc 退出 vi 编辑模式进入命令模式，输入":wq"进行保存并退出操作。

### 8.4.2 进行基于里程计的角速度标定

将机器人放置在平整地面上，并且自定义一个正方向，机器人正直朝向正方向放置。




新建终端，SSH 连接到机器人，运行角速度校准节点。

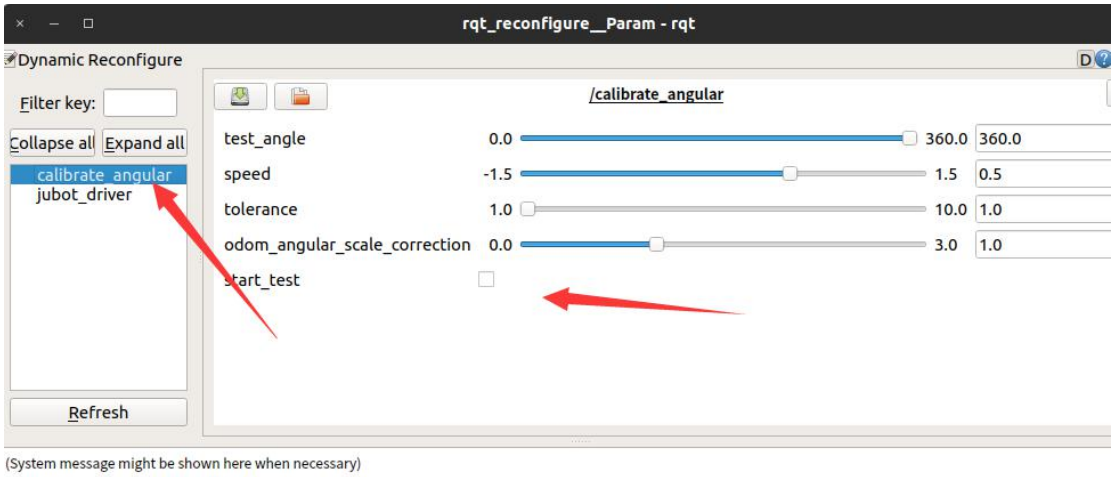
```
roslaunch jubot_driver jubot_calibrate_angular.launch
```

```
NODES
  /
    base_footprint_to_imu (tf/static_transform_publisher)
    calibrate_angular (jubot_driver/calibrate_angular.py)
    jubot_driver (jubot_driver/jubot_driver)

auto-starting new master
process[master]: started with pid [3718]
ROS_MASTER_URI=http://192.168.1.30:11311

setting /run_id to 18274064-fb51-11eb-94a3-a46bb6068b05
process[rosout-1]: started with pid [3729]
started core service [/rosout]
process[jubot_driver-2]: started with pid [3737]
process[calibrate_angular-3]: started with pid [3738]
process[base_footprint_to_imu-4]: started with pid [3743]
1.120000 0.890000
[ INFO] [1628761132.591142290]: Send Time: 1628761132.591013
[ INFO] [1628761132.593881699]: Send End Time: 1628761132.593799
[ INFO] [1628761132.605962945]: Set PID P:[250], I:[0], D:[150]
[ INFO] [1628761132.609906797]: Robot Running!
[INFO] [1628761137.364433]: Bring up rqt_reconfigure to control the test.
```

出现上述提示后，再新建一个终端，在虚拟机端运行 rqt 可视化工具。温馨提示，再虚拟机端运行 ROS 图形化工具，IP 环境变量需要修改正确，否则工具不能启动。

```
rosrun rqt_reconfigure rqt_reconfigure
```

点击左侧 calibrate_angular 选项，界面如下。test_angular 为测试角度，speed 为转动速度，tolerance 为允许误差，这三个参数一般为固定值不需要修改，odom_angular_scale_correction 为比例参数，是这次角速度校准的参数。




**校准流程**

步骤一，校准前先修改校正参数 angular_correction_factor 为 1.000，出厂默认为 1.000，如果用户之前已修改，请改回默认值 1.000 后再校准。

步骤二，点击 rqt_reconfigure 工具的 start_test 复选框，机器人开始转动，目标旋转角度为 360 度。

步骤三，机器人停止时，记录机器人实际的旋转角度，修改

odom_angular_scale_correction 比例参数，odom_angular_scale_correction 计算公式如下。

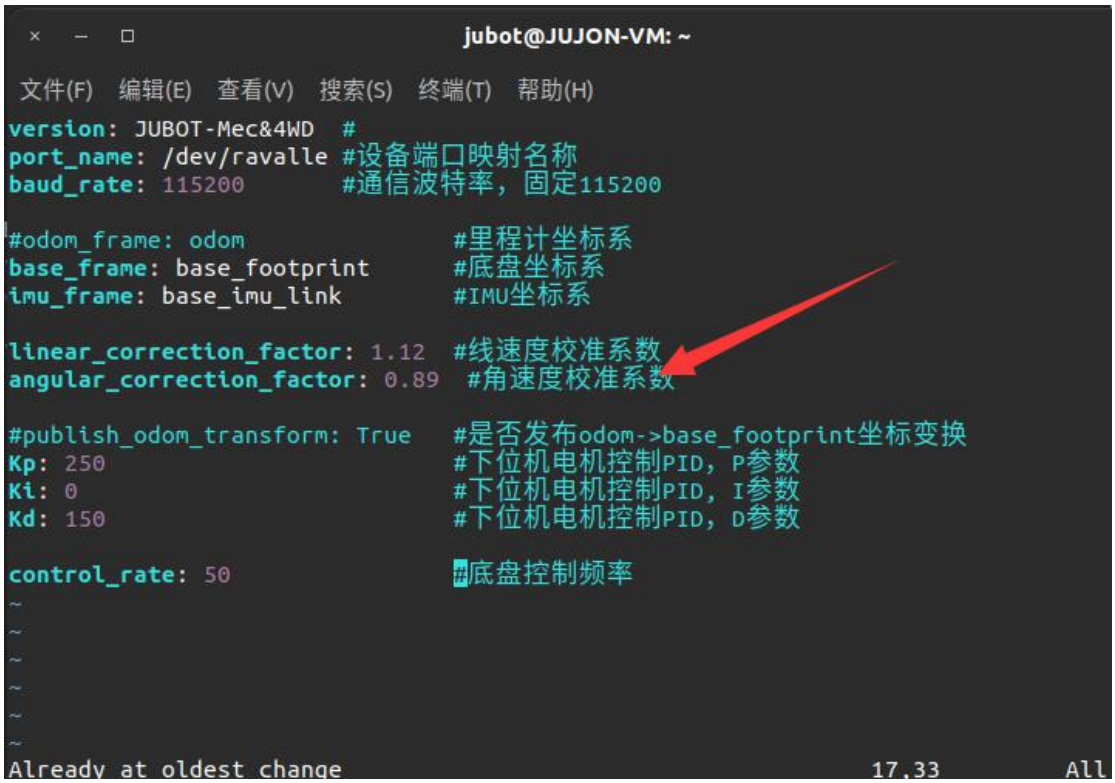
odom_angular_scale_correction = 实际旋转角度 ÷ 360 度　　步骤四，修改后，再次重复步骤二，此时机器人旋转角度会更接近 360 度。如果

还存在偏差，适当修正 odom_angular_scale_correction 值后再次测试，直至机器人能准确旋转 360 度。如果机器人旋转超过 360 度可适当加大

odom_angular_scale_correction 参数再测试，反之减小

odom_angular_scale_correction 参数再测试。最后记录下机器人准确旋转 360 度时 odom_angular_scale_correction 参数值。

步骤五，使用 vi 修改角速度校正参数 angular_correction_factor，参数位于 jubot_driver 工具包 config 文件夹内的 jubot_params.yaml 配置文件中。修改 angular_correction_factor 为前面修正的 odom_angular_scale_correction 值。




此时，机器人角速度校准完成，可以再次运行角速度校准程序，检验校准是否正确。特别说明，参数生效需要关闭并重新启动 jubot_driver 驱动包。

### 8.4.3 电机转速 PID 参数调节

ROS 机器人的轮子转动通过 PID 进行控制，我们已经默认配置了一套合适的

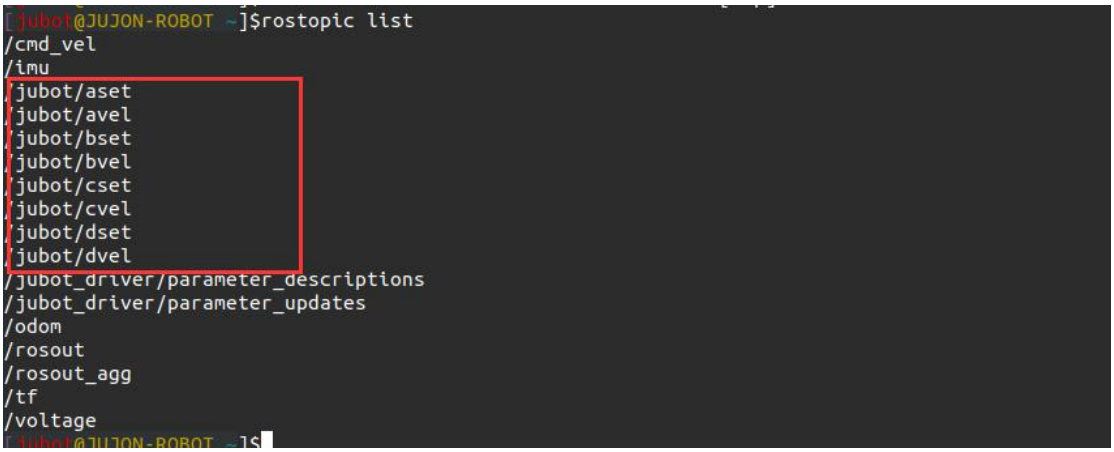
PID 参数。从学习角度考虑，用户也可以自行进行 PID 参数配置。温馨提示，建议用户调试后修改回默认参数，否则可能因为不合理的参数导致算法与机器人运动不匹配，出现奇怪现象。

机器人可通过 ROS 实现图像化调节电机运行 PID 参数，观察电机响应曲线，直观了解 PID 控制原理，对我们的调试有重要的作用，请跟随下面的教程操作。

首先通过 SSH 指令连接到 ROS 机器人，运行 jubot_driver 驱动包。

```
roslaunch jubot_driver jubot_driver.launch
```

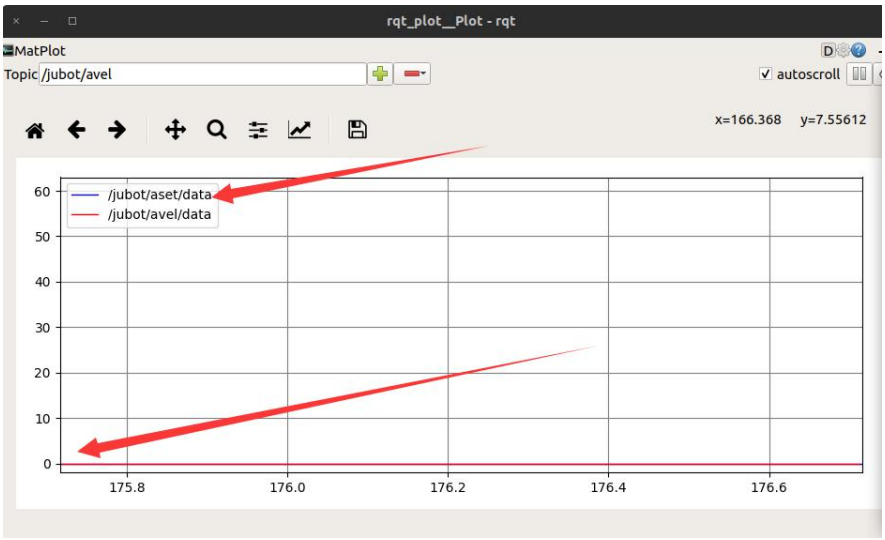
再新建一个终端，在虚拟机端运行 `rostopic list` 命令查看底盘驱动的话题消息。下图所示为电机目标速度和实际速度。




再新建一个终端，在虚拟机端运行 rosrun rqt_plot rqt_plot，打开图形化工具。

```
rosrun rqt_plot rqt_plot
```

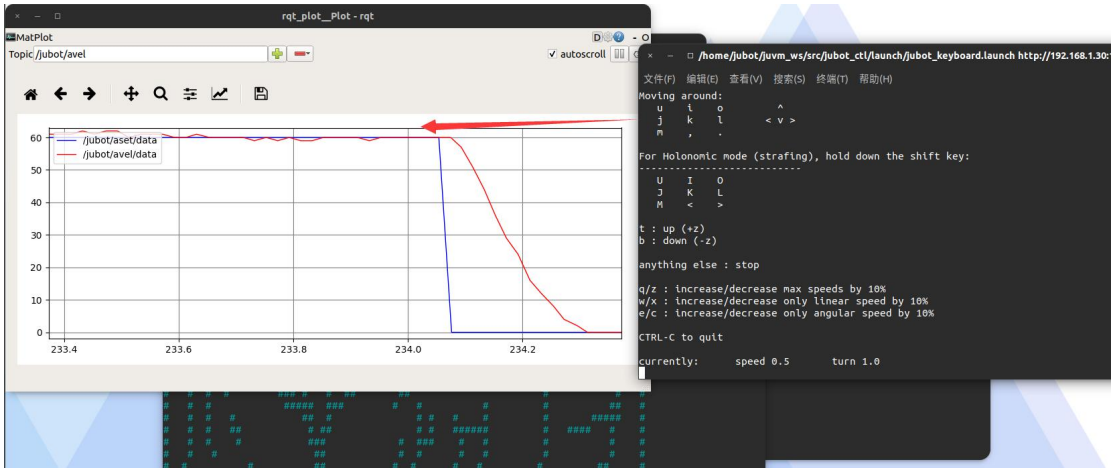
选择一组电机速度数据，例如 a 电机，将 aset 和 avel 两个数据添加到示波器，示波器实时显示两个数据。




然后新建终端，利用 SSH 指令连接到 ROS 机器人，运行键盘控制节点。

```
roslaunch jubot_ctl jubot_keyboard.launch
```

使用键盘操控机器人前进、后退等操作，可观察电机 A 曲线的变化，可看到电机实际速度对设定值的跟随效果。




用户也可修改电机控制的 PID 参数，再新建一个终端，在虚拟机端运行可视化工具。

```
rosrun rqt_reconfigure rqt_reconfigure
```

修改 Kp、Ki、Kd 的值来调节电机的运行效果，可结合目标值与实际值波形调节。


前面的调整为动态参数调整，不会更新到机器人参数文件，下次启动即失效。如果感觉自行整定的 PID 参数效果更好，可将整定好的参数写进机器人配置文件。（一般不建议）

再新建一个终端利用 SSH 指令连接到 ROS 机器人，进入 jubot_driver/config 文件夹，修改 jubot_params.yaml 配置文件的 PID 参数。

```
#publish_odom_transform: True    #是否发布odom->base_footprint坐标变换
Kp: 250                          #下位机电机控制PID，P参数
Ki: 0                            #下位机电机控制PID，I参数
Kd: 150                          #下位机电机控制PID，D参数
```

修改后退出并重新运行 jubot_driver.launch 驱动包和键盘指令，此时机器人的 PID 数值就是我们整定后的数值。

再次提醒，ROS 机器人的轮子转动通过 PID 进行控制，我们已经默认配置了一套合适的 PID 参数。从学习角度考虑，用户也可以自行进行 PID 参数配置，但建议用户调试完后修改回默认参数，否则可能因为不合理的参数导致算法与机器人运动不匹配，出现奇怪现象。

## 8.5实验结果

完成基于里程计的运动标定

## 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 SLAM 建图(Gmapping&Hector)

## 9.1 实验目的

理解栅格地图的概念，建图的原理，多种 slam 建图算法的一些参数含义。

## 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 多种slam 建图算法的一些参数的含义使用 多种slam 建图算法实际建图。

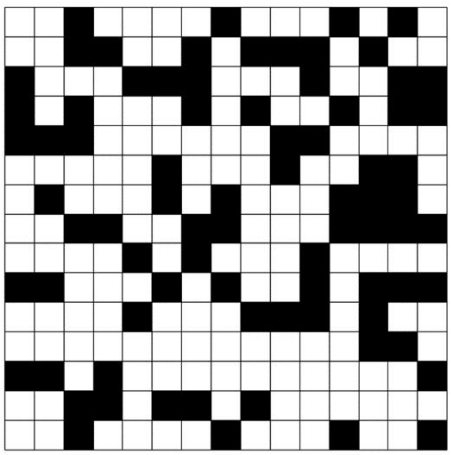
## 9.3 实验工具

个人电脑一台，路威及其配件。

## 9.4 实验内容

### 9.4.1 占据栅格地图的概念

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。

路威套件 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，故路威套件 不适用于特别大的环境场景。

### 9.4.3 Gmapping 建图算法理论概念


GMapping 的理论概念较为简单，它是基于 Rao-Blackwellized 粒子滤波（RBPF）算法实现的开源 SLAM 算法，GMapping 算法将定位与建图的过程分离， 先通过粒子滤波算法做定位，并通过粒子与已产生地图进行scanMatch，然后不断矫正里程计误差并添加新的 scan 做为地图。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。

同其他建图算法相比，GMapping 算法的优点与缺点也比较显著。

GMapping 算法的优点在于可以实时构建室内地图，在构建小场景地图所需的计算量较小且精度较高。相比 Hector SLAM 对激光雷达频率要求低、鲁棒性高

Hector 在机器人快速转向时很容易发生错误匹配，建出的地图发生错位，原因主要是优化算法容易陷入局部最小值）；

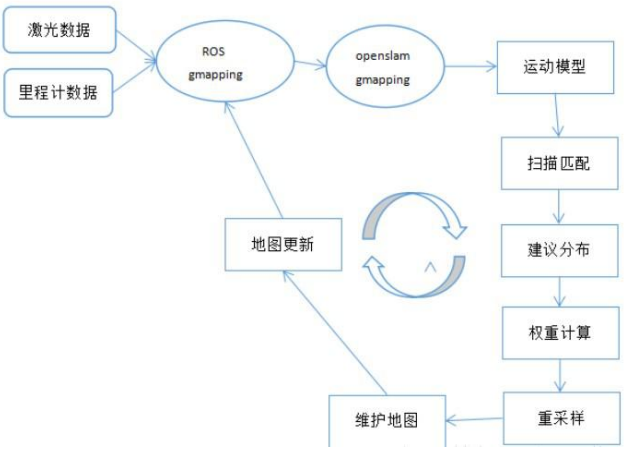
而相比 Cartographer 在构建小场景地图时，Gmapping 不需要太多的粒子并且没有回环检测因此计算量小于 Cartographer 而精度并没有差太多。Gmapping 有效利用了车轮里程计信息，这也是 Gmapping 对激光雷达频率要求低的原因：里程计可以提供机器人的位姿先验。而 Hector 和 Cartographer 的设计初衷不是为了解决平面移动机器人定位和建图，Hector 主要用于救灾等地面不平坦的情况，因此无法使用里程计。而 Cartographer 是用于手持激光雷达完成 SLAM 过程，也就没有里程计可以用。

GMapping 算法的缺点是随着场景增大所需的粒子增加，因为每个粒子都携带一幅地图，因此在构建大地图时所需内存和计算量都会增加。因此不适合构建大场景地图。并且没有回环检测，因此在回环闭合时可能会造成地图错位，虽然增加粒子数目可以使地图闭合但是以增加计算量和内存为代价。所以不能像 Cartographer 那样构建大的地图，虽然论文生成几万平米的地图，但实际我们使用中建的地图没有几千平米时就会发

生错误。

Gmapping 和 Cartographer 一个是基于滤波框架 SLAM 另一个是基于优化框架的 SLAM，两种算法都涉及到时间复杂度和空间复杂度的权衡。Gmapping 牺牲空间复杂度保证时间复杂度，这就造成 Gmapping 不适合构建大场景地图，试想一下要构建 200x200 米的环境地图，栅格分辨率选择 5 厘米，每个栅格占用一字节内存，那么一个粒子携带的地图就需要 16M 内存，如果是 100 个粒子就需要 1.6G 内存。如果地图变成 500 乘 500 米，粒子数为 200 个，可能电脑就要崩溃了。翻看 Cartographer 算法，优化相当于地图中只用一个粒子，因此存储空间比较 Gmapping 会小很多倍，但计算量大，一般的笔记本很难跑出来好的地图，甚至根本就跑不动。优化图需要复杂的矩阵运算，这也是谷歌为什么还要弄个ceres 库出来的原因。

GMapping 的算法流程如下图所示：




由图上可以看出，在 ROS 中的 slam_gmapping 软件包依赖于开源的 openslam_gmapping 算法包，也就是说，在 ROS 中的 gmapping 软件包是对 openslam_gmapping 算法包的ROS 封装实现。

GMapping 相关源代码及 WIKI 地址：

Gmapping ROS Wiki: http://wiki.ros.org/gmapping

slam_gmapping 软件包:

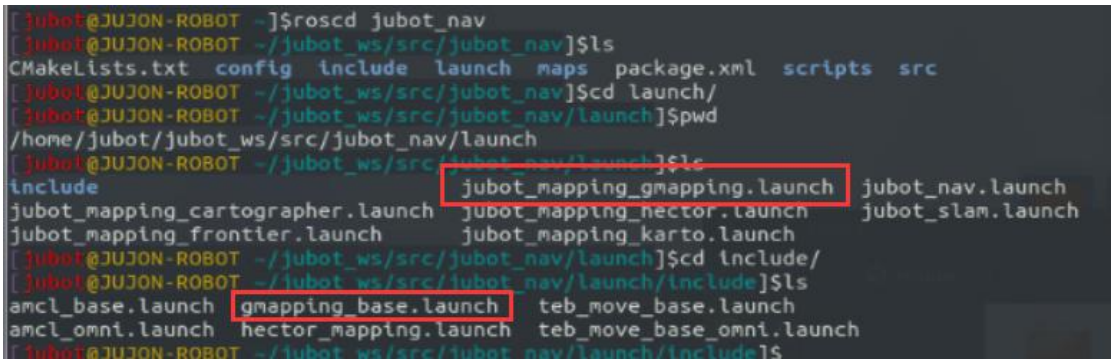
https://github.com/ros-perception/slam_gmapping

openslam_gmapping 开源算法:

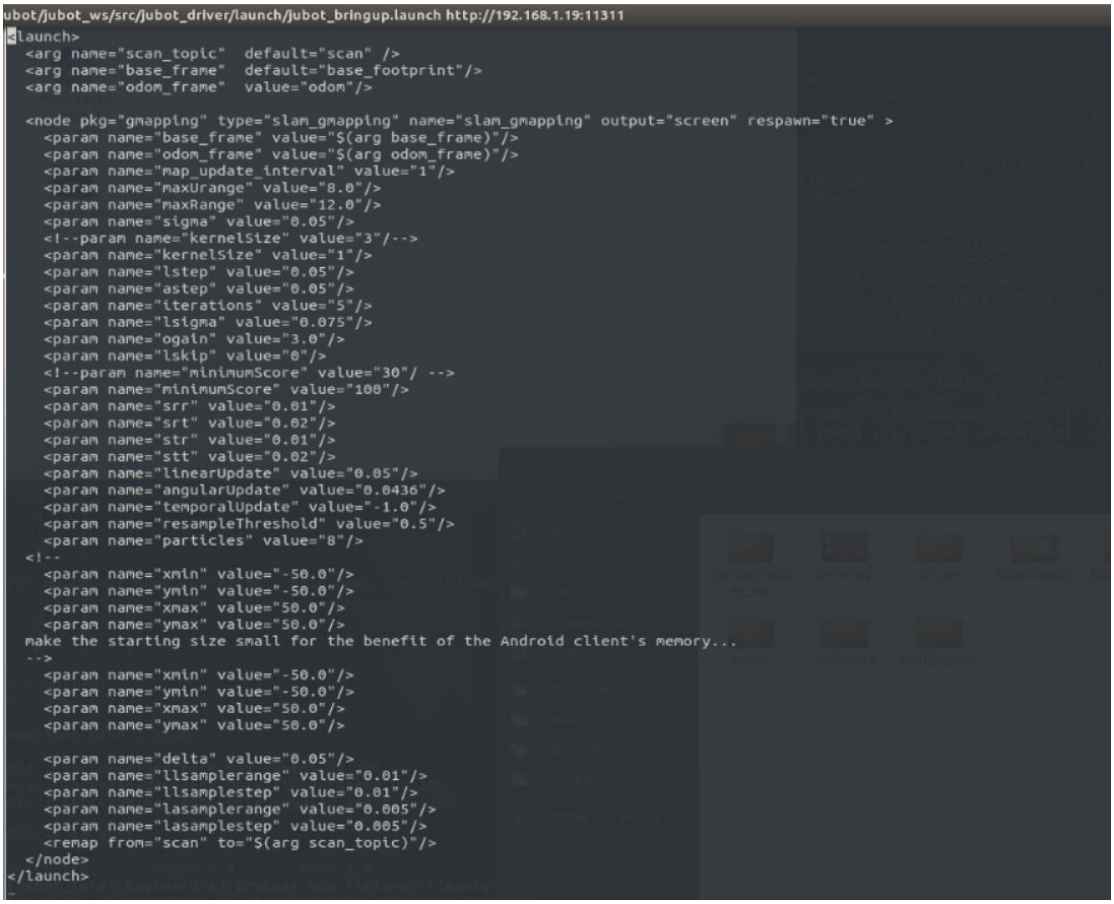
https://github.com/ros-perception/openslam_gmapping


## gmapping 参数

在该位置打开文件：




得到：




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 scanmach 的影响，如果 scanmatch 没有成功的话，不会更新地图。

**maxUrange：**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maxUrange < 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

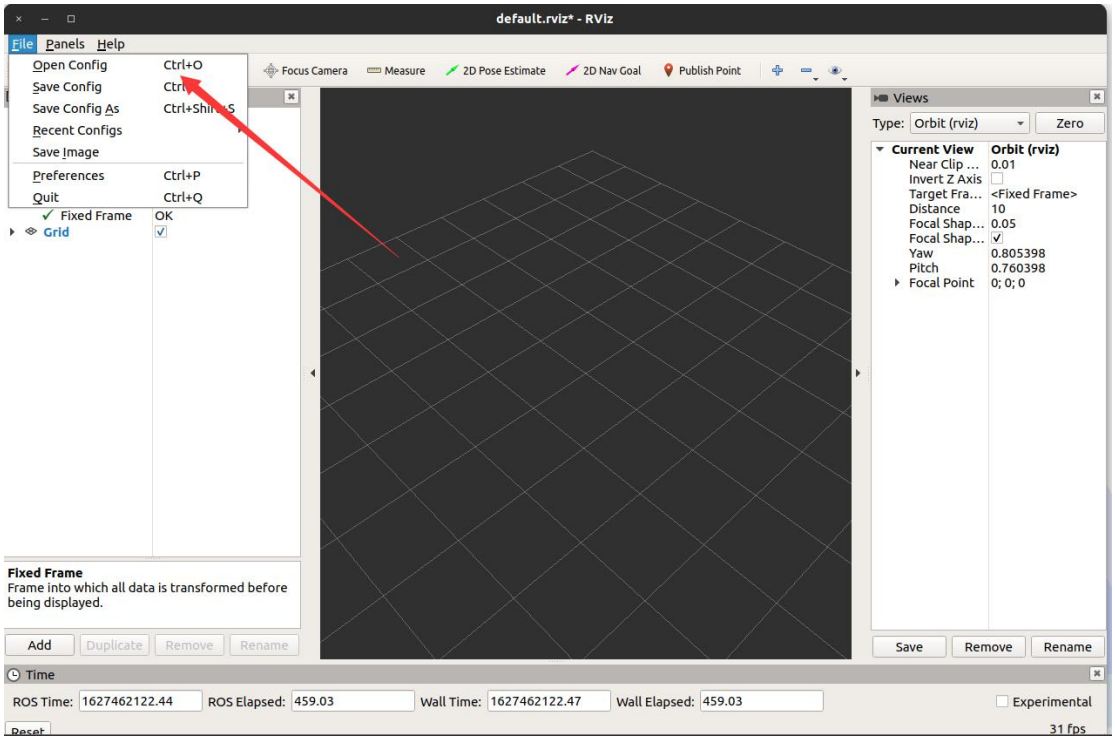
首先，打开终端输入：

`roslaunch jubot_nav jubot_slam.launch slam_methods:=gmapping` ，连接底盘，连接激光雷达，发布底盘到雷达的 TF 变换。

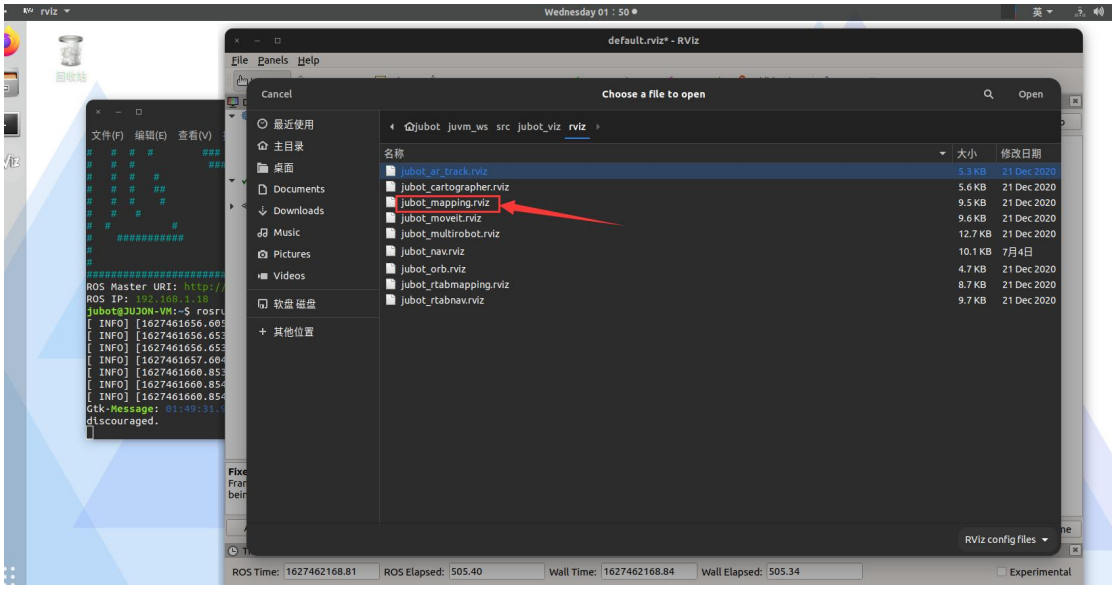
为了方便我们已经把rviz的配置保存在了虚拟机端。

在虚拟机端运行Rviz可视化工具：`rosrun rviz rviz`

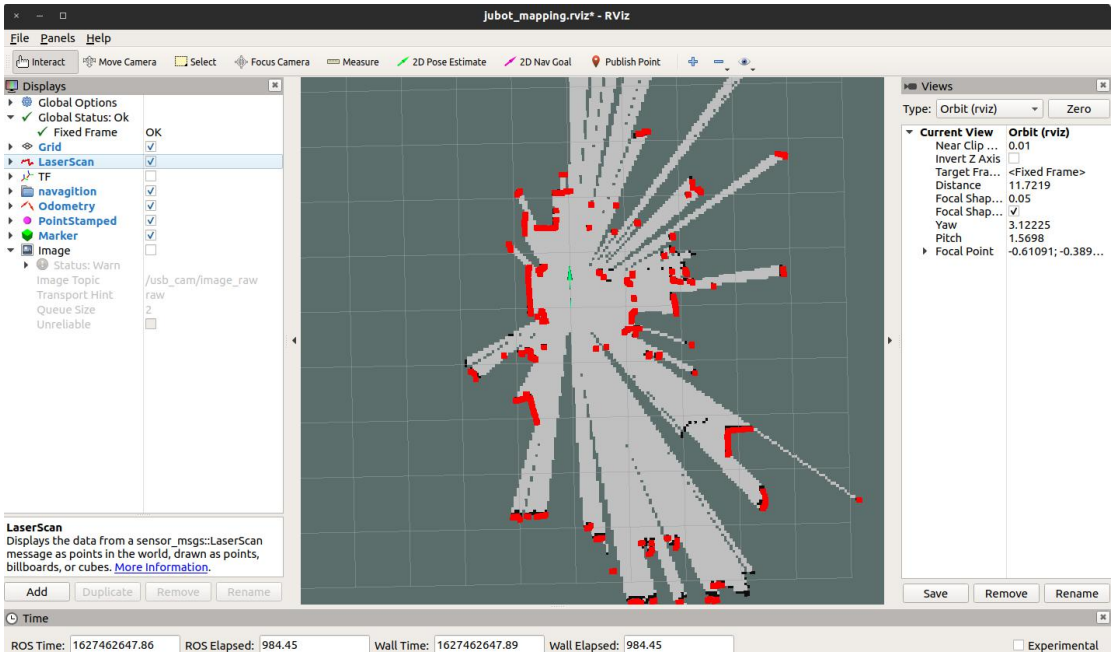
选择"File"菜单下的" Open Config"选项。菜单位置在整个界面的左上角。




选择建图所标识的jubot_mapping.rviz




打开配置文件后显示如下：




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。

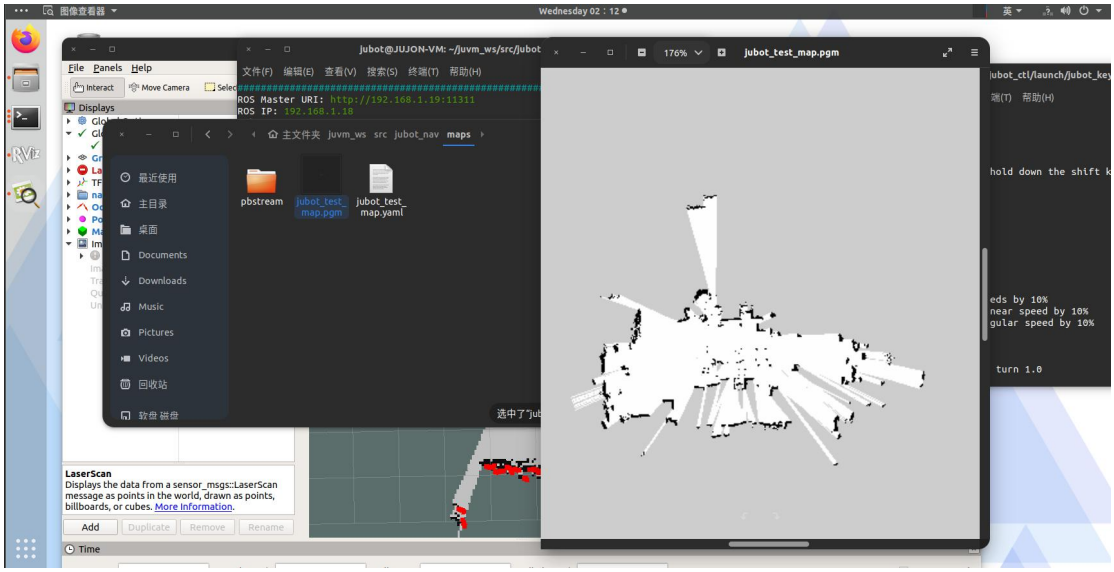
得到满意的地图后，在新终端内通过以下指令：

```
rosrun map_server map_saver -f jubot_test_map
```

保存地图，地图位置如下：

```
jubot@JUJON-VM:~/juvm_ws/src/jubot_nav$ ls
CMakeLists.txt  config  include  launch  maps  package.xml  scripts  src
jubot@JUJON-VM:~/juvm_ws/src/jubot_nav$ cd maps/
jubot@JUJON-VM:~/juvm_ws/src/jubot_nav/maps$ ls
pbstream
jubot@JUJON-VM:~/juvm_ws/src/jubot_nav/maps$ rosrun map_server map_saver -f jubo
t_test_map
[ INFO] [1627463366.992706185]: Waiting for the map
[ INFO] [1627463368.745613618]: Received a 1984 X 1984 map @ 0.050 m/pix
[ INFO] [1627463368.745727207]: Writing map occupancy data to jubot_test_map.pgm
[ INFO] [1627463368.837259248]: Writing map occupancy data to jubot_test_map.yam
l
[ INFO] [1627463368.837445275]: Done

jubot@JUJON-VM:~/juvm_ws/src/jubot_nav/maps$ ls
jubot_test_map.pgm  jubot_test_map.yaml  pbstream
jubot@JUJON-VM:~/juvm_ws/src/jubot_nav/maps$
```

可以在保存地图的位置查看地图




## 9.4.5 Hector 算法建图讲解及实践

hector 功能包使用高斯牛顿方法，不需要里程计数据，只根据激光信息便可构建地图。因此，该功能包可以很好地在空中机器人、手持构图设备及特种机器人中运行。

## 9.4.6 Hector 建图算法理论概念

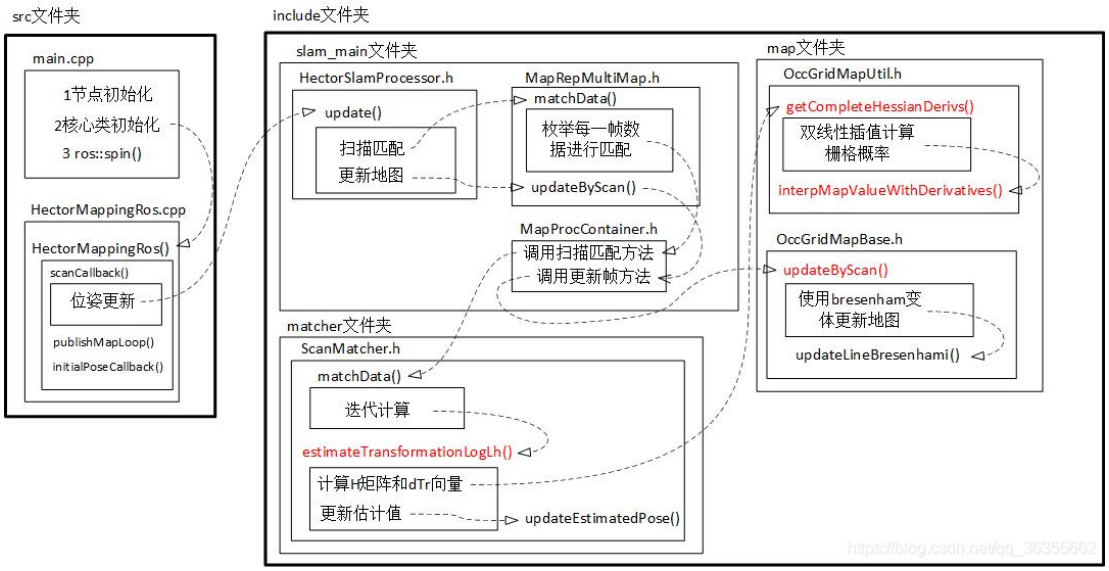
Hector slam 利用高斯牛顿方法解决 scan-matching 问题，对传感器要求较高。

Hector slam 的优点在于不需要使用里程计，所以使得空中无人机及地面路威套件在不平坦区域建图存在运用的可行性；利用已经获得的地图对激光束点阵进行优化，估计激光点在地图的表示,和占据网格的概率；利用高斯牛顿方法解决 scan-matching 问题，获得激光点集映射到已有地图的刚体变换（x,y,theta）；为避免局部最小而非全局最优，使用多分辨率地图；导航中的状态估计加入惯性测量系统（IMU），

利用 EKF 滤波；

Hector slam 也有很大的局限性，它需要雷达的更新频率较高，测量噪声小。所以在制图过程中，需要 robot 速度控制在比较低的情况下，建图效果才会比较理想，这也是它没有回环（loop close）的一个后遗症；且在里程计数据比较精确的时候，无法有效利用里程计信息。

## Hector 的算法流程如下图所示：



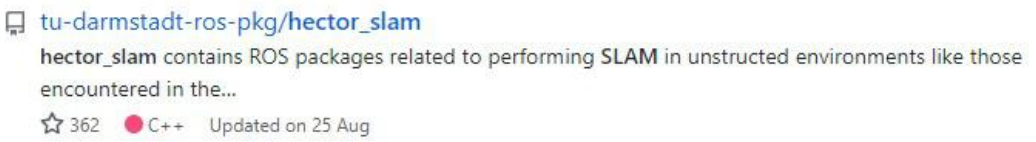

上图不仅是 Hector SLAM 的算法流程图，也是 Hector 源码的结构图，用户在后期如果要学习 Hector SLAM 算法源码，也可借鉴上图的结构进行代码梳理与学习。

HectorSLAM 相关源代码及 WIKI 地址：

Hector Mapping ROS Wiki: http://wiki.ros.org/hector_mapping

Hector_slam 软件包:

https://github.com/tu-darmstadt-ros-pkg/hector_slam




### 9.4.7 Hector 软件包的 ROS 框架

在 ROS 中，机器人使用 hector_mapping 软件包实现了 Hector 算法建图，在路威套件中，已通过二进制方法安装了 hector_mapping 软件包，用户无需手动安装，如需学习算法源码，可以通过上文提供的 github 地址直接下载算法包进行学习。

hector_mapping 节点所需要的 topic 数据及其发布的 topic 数据如下：

## Hector_mapping 收听话题

- scan(sensor_msgs/LaserScan)

用于创建地图的激光雷达数据

- syscommand(std_msgs/String)

系统命令。如果字符串为"reset"，则将地图和机器人姿态重置为其初始状态。

## Hector_mapping 发布话题

- map(nav_msgs/OccupancyGrid)

算法建立的地图话题。

- slam_out_pose(geometry_msgs/PoseStamped)

算法估计的机器人姿态，无协方差。

- poseupdate

估计的机器人姿态与不确定性的高斯估计。

## Hector_mapping 参数

在该位置打开文件：

```
[jubot@JUJON-ROBOT ~]$ls
cartographer_ws  dl_ws      jubot_shell  NoMachine  Software
Desktop          Downloads  jubot_ws     other      Wallpapers
[jubot@JUJON-ROBOT ~]$roscd jubot_nav
[jubot@JUJON-ROBOT ~/jubot_ws/src/jubot_nav]$ls
CMakeLists.txt  config  include  launch  maps  package.xml  scripts  src
[jubot@JUJON-ROBOT ~/jubot_ws/src/jubot_nav]$cd launch/
[jubot@JUJON-ROBOT ~/jubot_ws/src/jubot_nav/launch]$ls
include                                jubot_mapping_hector.launch
jubot_mapping_cartographer.launch      jubot_mapping_karto.launch
jubot_mapping_frontier.launch          jubot_nav.launch
jubot_mapping_gmapping.launch          jubot_slam.launch
[jubot@JUJON-ROBOT ~/jubot_ws/src/jubot_nav/launch]$cd include/
[jubot@JUJON-ROBOT ~/jubot_ws/src/jubot_nav/launch/include]$ls
amcl_base.launch  gmapping_base.launch    teb_move_base.launch
amcl_omni.launch  hector_mapping.launch   teb_move_base_omni.launch
[jubot@JUJON-ROBOT ~/jubot_ws/src/jubot_nav/launch/include]$pwd
/home/jubot/jubot_ws/src/jubot_nav/launch/include
[jubot@JUJON-ROBOT ~/jubot_ws/src/jubot_nav/launch/include]$
```

得到：

```
<launch>
  <arg name="tf_map_scanmatch_transform_frame_name" default="scanmatcher_frame"/>
  <arg name="base_frame" default="base_footprint"/>
  <arg name="odom_frame" default="odom"/>
  <arg name="pub_map_odom_transform" default="true"/>
  <arg name="scan_subscriber_queue_size" default="5"/>
  <arg name="scan_topic" default="scan"/>
  <arg name="map_size" default="2048"/>

  <node pkg="hector_mapping" type="hector_mapping" name="hector_mapping" output="screen">

    <!-- Frame names -->
    <param name="map_frame" value="map" />
    <param name="base_frame" value="$(arg base_frame)" />
    <param name="odom_frame" value="$(arg odom_frame)" />

    <!-- Tf use -->
    <param name="use_tf_scan_transformation" value="true"/>
    <param name="use_tf_pose_start_estimate" value="false"/>
    <param name="pub_map_odom_transform" value="$(arg pub_map_odom_transform)"/>

    <!-- Map size / start point -->
    <param name="map_resolution" value="0.050"/>
    <param name="map_size" value="$(arg map_size)"/>
    <param name="map_start_x" value="0.5"/>
    <param name="map_start_y" value="0.5" />
    <param name="map_multi_res_levels" value="2" />

    <!-- Map update parameters -->
    <param name="update_factor_free" value="0.4"/>
    <param name="update_factor_occupied" value="0.9" />
    <param name="map_update_distance_thresh" value="0.4"/>
    <param name="map_update_angle_thresh" value="0.06" />
    <param name="laser_z_min_value" value = "-1.0" />
    <param name="laser_z_max_value" value = "1.0" />

    <!-- Advertising config -->
    <param name="advertise_map_service" value="true"/>

    <param name="scan_subscriber_queue_size" value="$(arg scan_subscriber_queue_size)"/>
    <param name="scan_topic" value="$(arg scan_topic)"/>
```

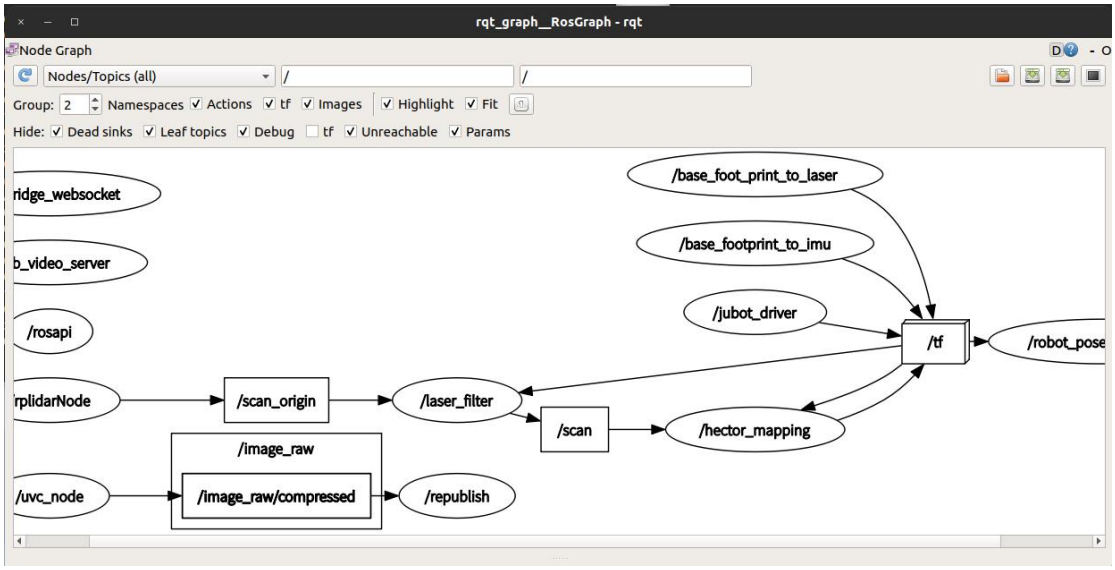
这些都是 Hector 中的一些参数，下面将针对其中的部分参数作简单介绍：

- base_frame 机器人基坐标系名称。
- map_frame 地图坐标系名称
- odom_frame 里程计坐标系名称
- map_resolution 地图分辨率[m]。这是网格单元边缘的长度。
- map_size 地图的大小[每轴像元数]。该地图为正方形，并具有（map_size * map_size）个网格单元。
- map_start_x map 框架的原点[0.0，1.0]在 x 轴上相对于网格图的位置。 0.5 在中间。
- map_start_y map 框架的原点[0.0，1.0]在 y 轴上相对于网格图的位置。 0.5 在中间。
- map_update_distance_thresh 进行地图更新的阈值[m]。自地图更新发生之前的最后一次更新以来，平台必须以米为单位行进这么长的距离，或者经历
- map_update_angle_thresh 参数所描述的角度变化。
- map_update_angle_thresh 执行地图更新的阈值[rad]。自地图更新发生之前的最后一次更新以来，平台必须经历此行程参数所描述的角度变化，直到
- map_update_distance_thresh 参数指定的角度为止。
- map_pub_period 地图发布周期[s]。
- map_multi_res_levels 地图多分辨率网格级别的数量。
- update_factor_free 映射更新修饰符，用于更新[0.0，1.0]范围内的空闲单

元。值 0.5 表示没有变化。

- update_factor_occupied 映射更新修饰符，用于更新[0.0，1.0]范围内的占用单元。值 0.5 表示没有变化。
- laser_min_dist 系统要使用的激光扫描端点的最小距离[m]。比此值更近的扫描端点将被忽略。
- laser_max_dist 系统要使用的激光扫描端点的最大距离[m]。比此值更远的扫描端点将被忽略。
- laser_z_min_value 系统要使用的激光扫描端点相对于激光扫描仪框架的最小高度[m]。低于此值的扫描端点将被忽略。
- laser_z_max_value 系统要使用的激光扫描端点相对于激光扫描仪框架的最大高度[m]。高于此值的扫描端点将被忽略。
- pub_map_odom_transform 确定是否应由系统发布map-> odom 变换。
- output_timing 输出定时信息，用于通过 ROS_INFO 处理每次激光扫描。
- scan_subscriber_queue_size 扫描订阅服务器的队列大小。如果日志文件以比实时速度更快的速度回放到 hector_mapping，则应将其设置为较高的值（例如 50）。
- pub_map_scanmatch_transform 确定是否将scanmatcher 映射映射转换发布到 tf。帧名称由" tf_map_scanmatch_transform_frame_name"参数确定。
- tf_map_scanmatch_transform_frame_name 如前面的参数中所述，将
- scanmatcher 发布到映射转换时的帧名称。

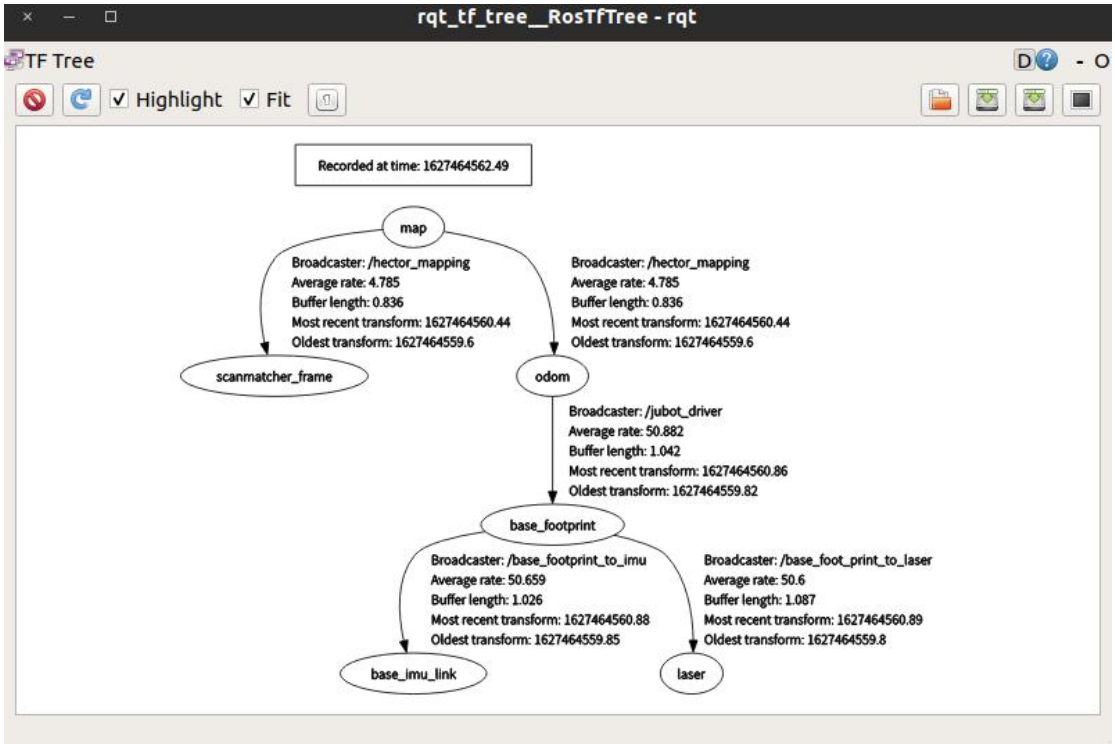
使用 rosrun rqt_graph rqt_graph 查看Hector_mapping 节点话题结构图




如图所示，hector_mapping 节点的话题结构是相对简单的，其订阅了/scan 话题来获取激光雷达的扫描数据，还订阅了TF 话题，来获取激光雷达与底盘的 TF 位置和发布机器人的地图 SLAM 定位信息。Hector_mapping 节点发布/map 话题，就是

hector 算法所建立的地图。可以看到，hector 算法建图只需要激光雷达的数据，不需要里程计的数据参与，所以 hector 建图算法很适合用在崎岖的复杂场景或无人机地图的建立。

使用 `rosrun rqt_tf_tree rqt_tf_tree` 查看Hector_mapping 的TF 坐标发布




由图中可以看到，hector_mapping 节点发布了从 map 到 odom 的坐标变换， 这

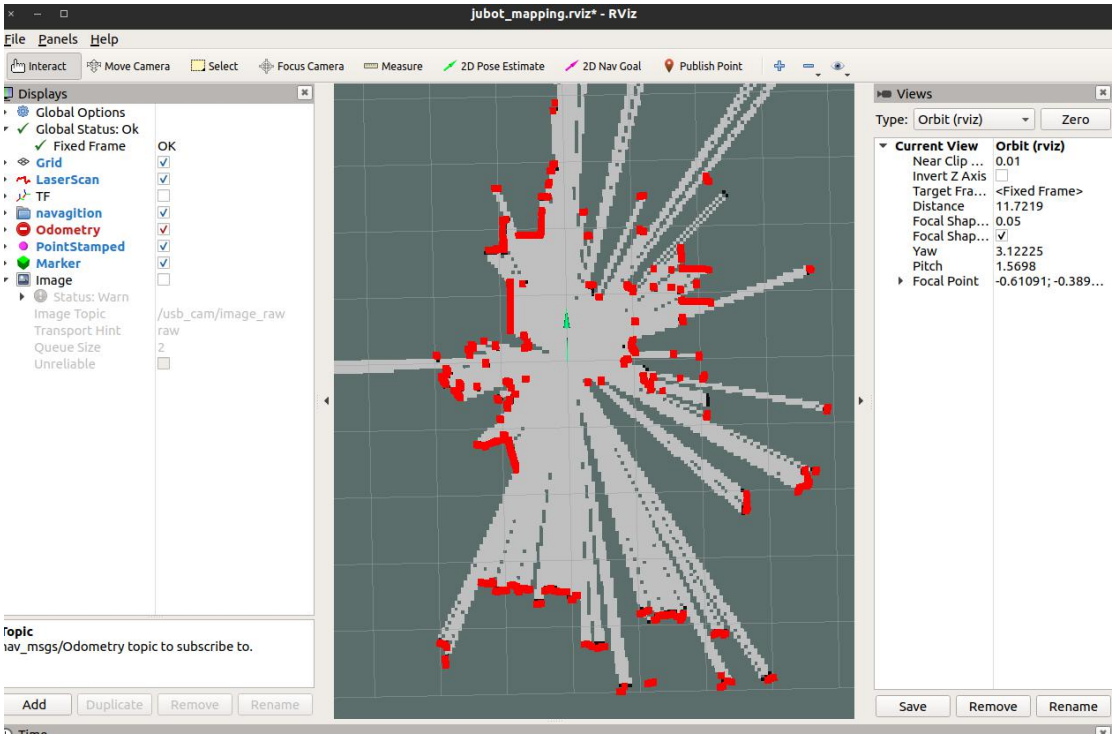
就意味着 hector_mapping 节点提供了机器人在 map 坐标系中的定位。由此，完成 hector_mapping 建图定位的流程。

### 9.4.8 Hector 建图流程

`roslaunch jubot_nav jubot_slam.launch slam_methods:=hector` ，连接底盘，连接激光雷达，发布底盘到雷达的 TF 变换。

按照上节所示操作，在虚拟机端运行Rviz可视化工具：`rosrun rviz rviz`

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：

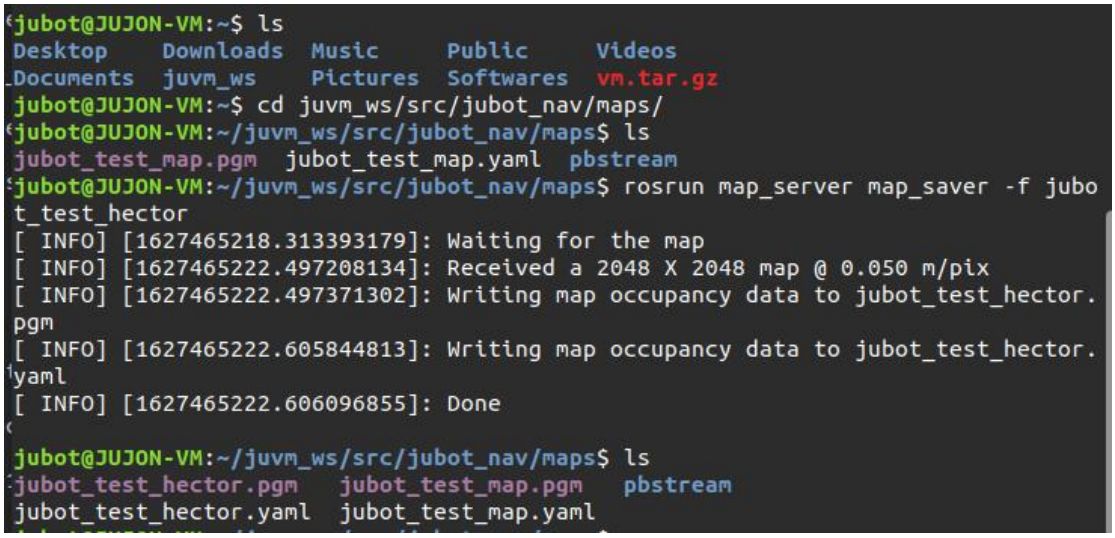
| 地图颜色 | 代表意义 |
| --- | --- |
| 红色 | 激光雷达探测到的障碍点（障碍物轮廓点阵） |
| 灰色 | 还没有探索到的未知区域 |
| 白色 | 已经探明的不存在静态障碍物的区域 |
| 黑色 | 静态障碍物轮廓 |

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：

```
rosrun map_server map_saver -f jubot_test_map
```

保存地图，地图位置如下：




查看如上节Gmapping所示。

## 9.5 实验结果

使用Gmapping & Hector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 SLAM 建图(Karto&Cartographer)

## 10.1 实验目的

理解栅格地图的概念，建图的原理，多种 slam 建图算法的一些参数含义。

## 10.2 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 多种slam 建图算法的一些参数的含义使用多种slam 建图算法实际建图。

## 10.3 实验工具

个人电脑一台，路威及其配件。

## 10.4 实验内容

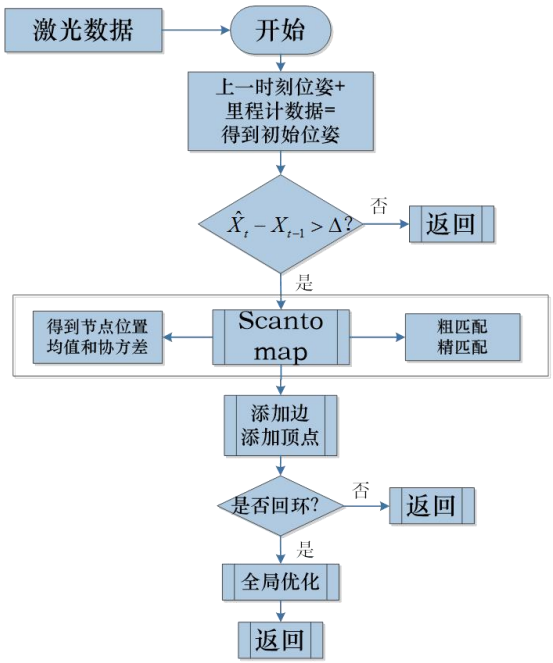
### 10.4.1 Karto 建图算法理论概念


Karto_SLAM 是基于图优化的思想，用高度优化和非迭代 cholesky 分解进行稀疏系统解耦作为解。图优化方法利用图的均值表示地图，每个节点表示机器人轨迹的一个位置点和传感器测量数据集，每个新节点加入，就会进行计算更新。

Karto_SLAM 的 ROS 版本，其中采用的稀疏点调整（the Spare Pose Adjustment(SPA)）与扫描匹配和闭环检测相关。landmark 越多,内存需求越大,然而图优化方式相比其他方法在大环境下制图优势更大，因为他仅包含点的图(robot

pose)，求得位姿后再求map。

Karto SLAM 的算法程序框架如下图所示：




从上图中可以看出，其实流程还是相当简单的，slam 传统软实时的运行机制，每进入一帧数据，即进行处理，然后返回。

KartoSLAM 相关源代码及 WIKI 地址：

- KartoSLAM ROS Wiki: http://wiki.ros.org/slam_karto
- slam_karto 软件包: https://github.com/ros-perception/slam_karto
- open_karto 开源算法: https://github.com/ros-perception/open_karto

### 10.4.2 karto 软件包的 ROS 框架讲解

在 ROS 中，机器人使用slam_karto 软件包实现了 KartoSLAM 算法建图，在路威套件中，已通过二进制方法安装了 slam_karto 软件包，用户无需手动安装，如需学习算法源码，可以通过上文提供的 github 地址直接下载算法包进行学习。

slam_karto 节点所需要的 topic 数据及其发布的 topic 数据如下：

#### slam_karto 收听话题

- tf(tf/tfMessage)

  slam_karto 软件包通过查询 tf 变化，获取机器人基座标系与雷达的变换关系， 以及获取机器人里程计的定位信息。

- scan(sensor_msgs/LaserScan)

  用于创建地图的激光雷达数据

## slam_karto 发布话题

- map(nav_msgs/OccupancyGrid)

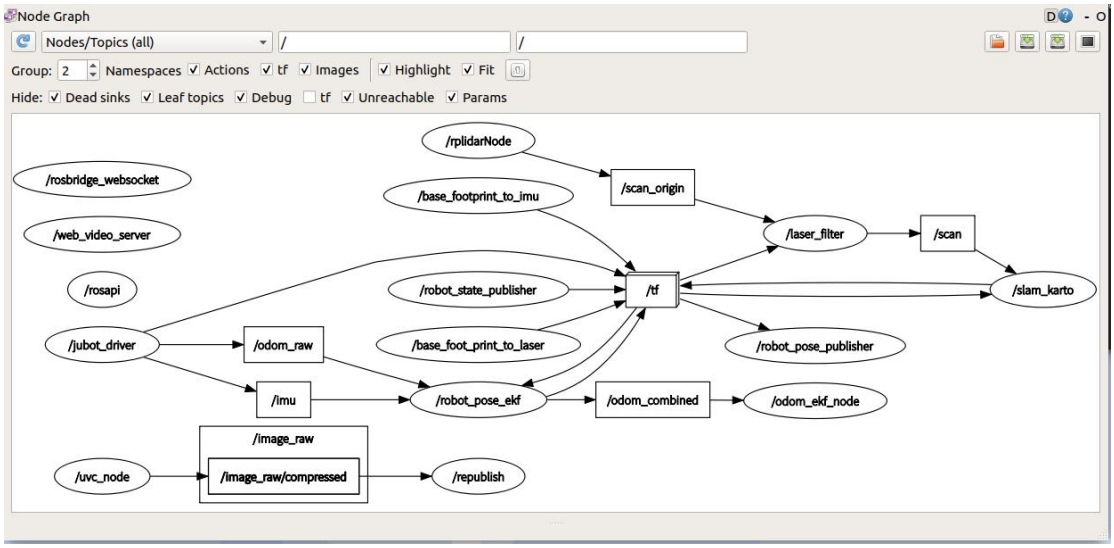
## slam_karto 参数

- odom_frame 里程计坐标系名称。
- map_frame 地图坐标系名称。
- base_frame 机器人基坐标系名称。
- throttle_scans 算法在每多少次的扫描中处理一次数据（将其设置为更高的数字以跳过更多扫描）。
- map_update_interval 两次更新之间的时间间隔（以秒为单位）。降低此数字会更频繁地更新占用网格，以增加计算负荷为代价。
- Resolution 地图的分辨率（每个占用格块的米数）
- Delta 地图的分辨率（每个占用栅格块的米数）。与分辨率相同。定义为与 gmapping 的参数名称兼容。
- transform_publish_period TF 发布之间的时间间隔（以秒为单位）。要禁用广播 TF，请设置为 0。
- use_scan_matching 设置为 true 时，建图算法将使用扫描匹配算法。在大多数实际情况下，应将其设置为 true，以便建图算法可以校正测距法和扫描数据中的噪声和错误。在某些模拟环境中，其中模拟的扫描和测距数据非常准确，扫描匹配算法可能会产生较差的结果。在这种情况下，请将其设置为 false 可改善结果。
- use_scan_barycenter 使用扫描端点的重心定义扫描之间的距离。
- minimum_travel_distance 设置两次扫描之间的最小距离。如果新扫描的位置大于上一次扫描的 minimumTravelDistance，则算法将使用新扫描的数据。否则，如果它也不能满足航向要求的最低要求，它将放弃新扫描。出于性能原因，通常最好仅在机器人移动了合理数量的情况下才进行过程扫描。
- minimum_travel_heading 设置两次扫描之间的最小航向变化。如果新扫描的数据大于上一次扫描的 minimum_travel_heading，则映射器将使用新扫描的数据。否则，如果新扫描也未达到最小行进距离要求，它将被丢弃。出于性能原因，通常最好仅在机器人移动了合理数量的情况下才进行过程扫描。

- scan_buffer_size 设置为扫描匹配而存储的扫描链的长度。
- scan_buffer_maximum_scan_distance 设置为匹配而存储的扫描链中第一次扫描和最后一次扫描之间的最大距离。
- link_match_minimum_response_fine 仅当相关响应值大于此值时，才链scan。
- link_scan_maximum_distance 设置链接扫描之间的最大距离。无论相关响应值如何，相距较远的 scan 帧都不会链接。
- loop_search_maximum_distance 与当前位置的距离小于此距离的扫描将被视为匹配闭环。
- do_loop_closing 启用/禁用闭环匹配。
- loop_match_minimum_chain_size 当闭环检测找到候选对象时，它必须是大量链接扫描中的一部分。如果扫描链小于此值，则我们不会尝试闭环。
- loop_match_maximum_variance_coarse 要考虑可行的解决方案， 可能的环路闭合的协方差值必须小于该值。这适用于粗略搜索。
- loop_match_minimum_response_coarse 如果响应大于此值，则以粗略分辨率启动循环关闭搜索。
- loop_match_minimum_response_fine 如果响应大于此值，则以较高分辨率启动循环关闭搜索。
- correlation_search_space_dimension 设置匹配器使用的搜索网格的大小。搜索网格的大小为 correlation_search_space_dimension x
- correlation_search_space_dimension
- correlation_search_space_resolution 设置相关网格的分辨率（网格单元的大小）。
- correlation_search_space_smear_deviation X 和 Y 中的该值会平滑点读数，以创建更平滑的响应。
- loop_search_space_dimension 匹配器使用的搜索网格的大小。
- loop_search_space_resolution 相关网格的分辨率（网格单元的大小）。
- loop_search_space_smear_deviation X 和Y 中的该值会平滑点读数，以创建更平滑的响应。
- distance_variance_penalty 扫描匹配时偏离里程表的惩罚差异。惩罚是乘数（小于 1.0），它是被测扫描位置和测距姿势的增量的函数。

- angle_variance_penalty 同
- distance_variance_penalty 定义。
- fine_search_angle_offset 精细搜索期间要搜索的角度范围。
- coarse_search_angle_offset 粗略搜索期间要搜索的角度范围。
- coarse_angle_resolution 粗略搜索期间要搜索的角度分辨率。
- minimum_angle_penalty 角度惩罚乘数的最小值，因此得分不会变得太小。
- minimum_distance_penalty距离罚分乘数的最小值， 因此分数不会变得太小。
- use_response_expansion如果最初没有找到合适的匹配项，是否增加搜索空间

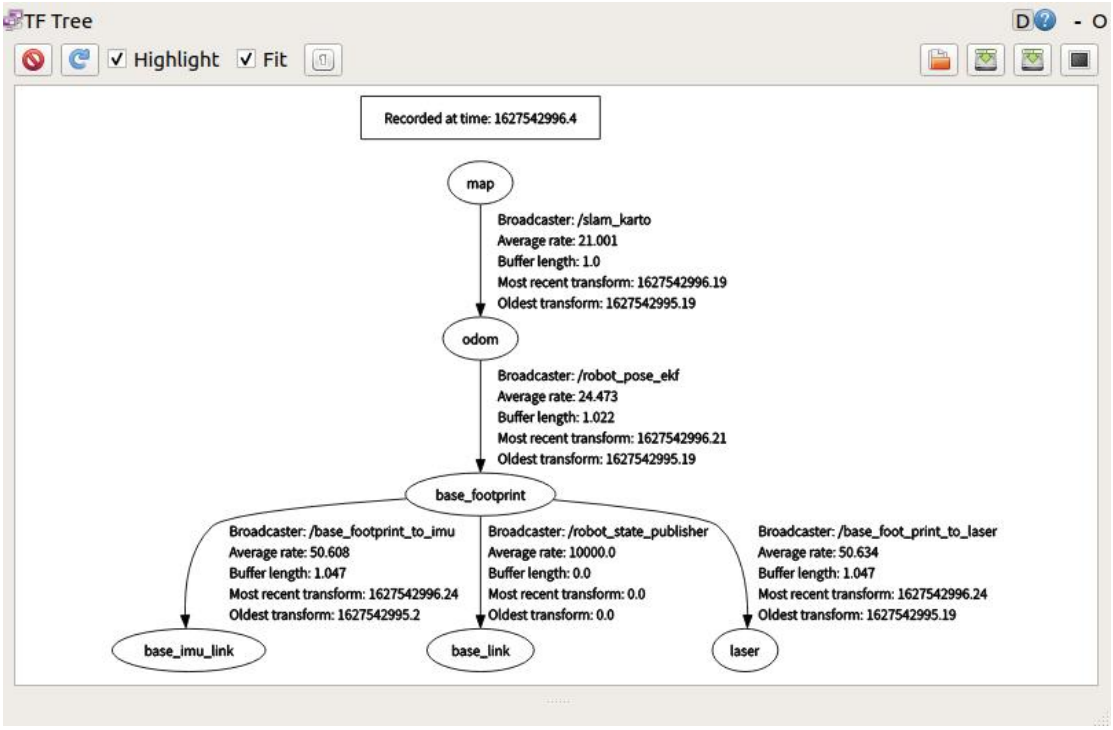
## slam_karto 节点话题结构图




如图所示，slam_karto 节点的话题结构是相对简单的，其订阅了/scan 话题来获取激光雷达的扫描数据，还订阅了 TF 话题，来获取机器人的里程计定位信息。

slam_karto 节点发布/map 话题，就是 kartoSLAM 算法所建立的地图。

## slam_karto 的 TF 坐标发布



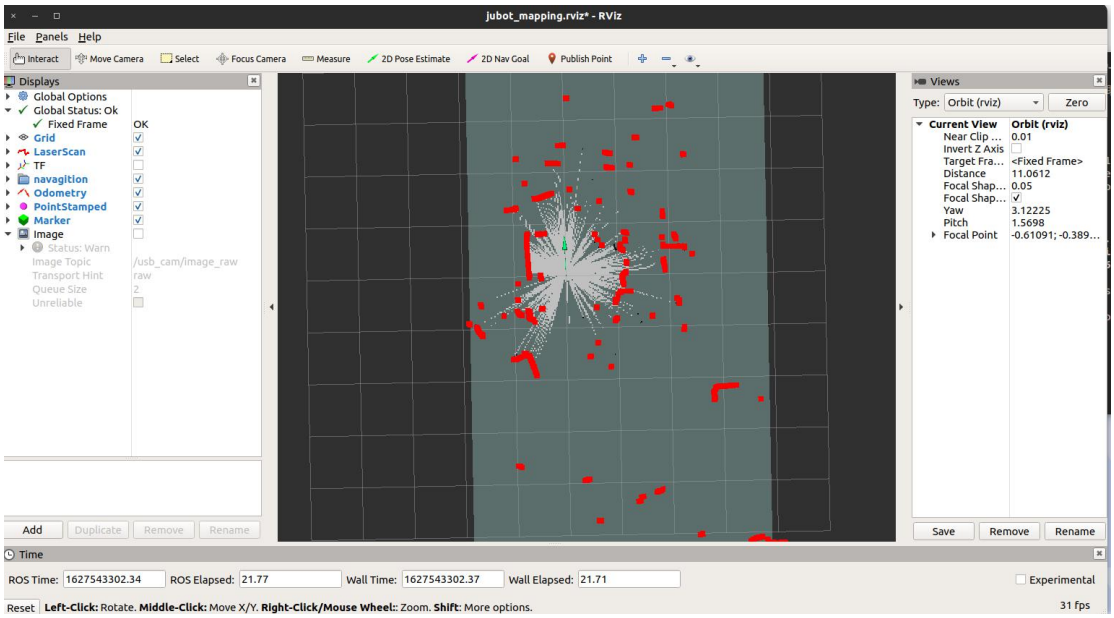

由图中可以看到，slam_karto 节点发布了从map 到 odom 的坐标变换，这就意味着 slam_karto 节点提供了机器人在map 坐标系中的定位。由此，完成 slam_karto 建图的流程。

### 10.4.3 slam_karto 建图流程

`roslaunch jubot_nav jubot_slam.launch slam_methods:=karto` ,连接底盘,连接激光雷达，发布底盘到雷达的 TF 变换。

按照上节所示操作，在虚拟机端运行Rviz可视化工具：`rosrun rviz rviz`

同上节所示，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，在新终端内进入到下图表示位置，然后通过以下指令：

```
rosrun map_server map_saver -f jubot_test_karto
```

保存地图，地图位置如下：

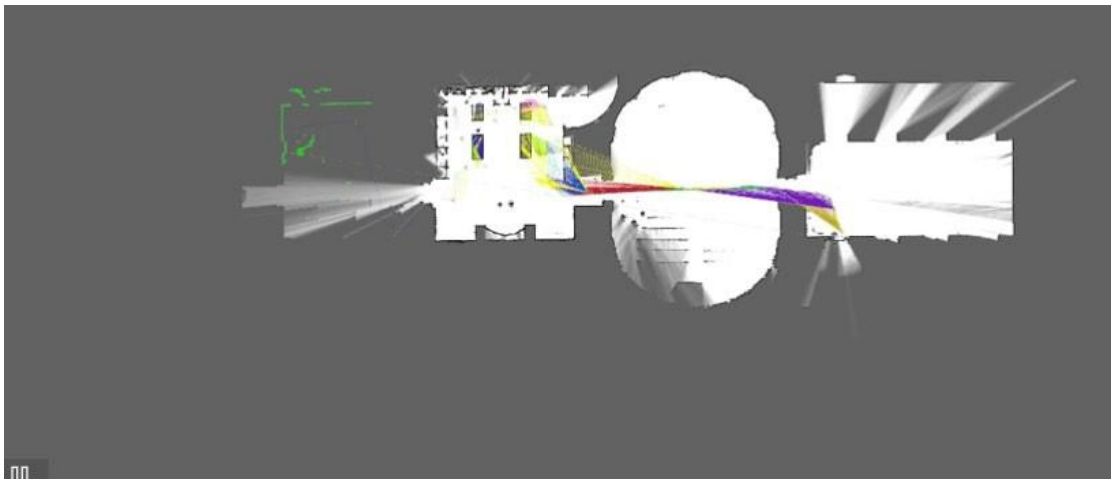
```
jubot@JUJON-VM:~/juvm_ws/src/jubot_nav/maps$ ls
jubot_test_karto.pgm   jubot_test_map.pgm   pbstream
jubot_test_karto.yaml  jubot_test_map.yaml
jubot@JUJON-VM:~/juvm_ws/src/jubot_nav/maps$ pwd
/home/jubot/juvm_ws/src/jubot_nav/maps
jubot@JUJON-VM:~/juvm_ws/src/jubot_nav/maps$
```

### 10.4.4 Cartographer 算法理论概念

本节使用 Google Cartographer 算法进行机器人建图，Cartographer 是基于图优化的方法建图算法，它与 Karto 都是图优化框架，但有诸多不同，例如 Karto 采取的是spa 图优化方法，而 Cartographer 采用的是google 的 ceres 构建 problem 优化，Karto 的前后端是单线程进行，而 cartographer 采取的是多线程后端优化。

而Cartographer 也支持多传感器融合建图，可以处理来自激光雷达、IMU、里程计

等传感器的数据并给予这些数据进行地图的构建。



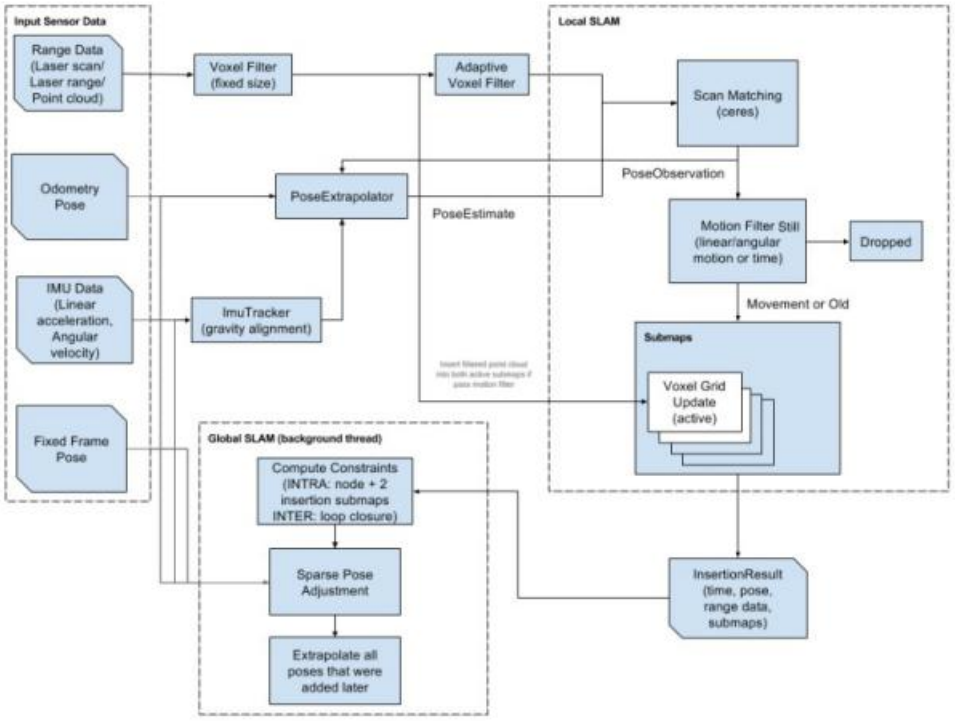
Cartographer 主要理论是通过闭环检测来消除构图过程中产生的累积误差。用于闭环检测的基本单元是submap。

一个 submap 是由一定数量的 laser scan 构成。将一个 laser scan 插入其对应的 submap 时,会基于submap 已有的laser scan 及其它传感器数据估计其在该 submap 中的最佳位置。

submap 的创建在短时间内的误差累积被认为是足够小的。然而随着时间推移，越来越多的 submap 被创建后，submap 间的误差累积则会越来越大。因此需要通过闭环检测适当的优化这些 submap 的位姿进而消除这些累积误差，这就将问题转化成一个位姿优化问题。当一个 submap 的构建完成时，也就是不会再有新的 laser scan 插入到该 submap 时，该 submap 就会加入到闭环检测中。闭环检测会考虑所有的已完成创建的submap。

当一个新的 laser scan 加入到地图中时，如果该 laser scan 的估计位姿与地图中某个 submap 的某个 laser scan 的位姿比较接近的话，那么通过某种scan match 策略就会找到该闭环。

Cartographer 中的 scan match 策略通过在新加入地图的 laser scan 的估计位姿附近取一个窗口，进而在该窗口内寻找该 laser scan 的一个可能的匹配，如果找到了一个足够好的匹配，则会将该匹配的闭环约束加入到位姿优化问题中。Cartographer 的重点内容就是融合多传感器数据的局部 submap 创建以及用于闭环检测的 scan match 策略的实现。




cartographer 整体可以分为两个部分：

一是 Local SLAM 部分，也被称为 Frontend; 该部分的主要任务是建立维护 Submaps,但问题是该部分建图误差会随着时间累积。该部分相关的参数定义在 /src/cartographer/configuration_files/trajectory_builder_2d.lua 和 /src/cartographer/configuration_files/trajectory_builder_3d.lua 中。

二是 Global SLAM 部分，也称为Backend. 该部分的主要任务是进行 Loop Closure。如前所述，闭环检测本质上也是一个优化问题，文中将其表达成了 pixel-accurate match 的形式，采用 Branch-and-Bound Approach （BBA）的方法来解决。具体怎么用 Branch-and-Bound 方法，可以参见论文（Real-Time Loop Closure in 2D LIDAR SLAM）。如果是 3D 情况下，该部分还负责根据 IMU 数据找出重力的方向。

总体而言，Local Slam 部分负责生成较好的子图，而 Global Slam 部分进行全局优化，将不同的子图以最匹配的位姿 tie 在一起。

可以看出，从传感器出发，Laser 的数据经过两个滤波器后进行 Scan Matching，用来构建子图 submap，而新的 Laser Scan 进来后也会插入到已经维护着的子图的适当位置。如何决定插入的最优位姿，就是通过 Ceres Scan Matching 来实现的。估计出来的最优位姿也会与里程计和 IMU 数据融合，用来估计下一时刻的位姿。

## 10.4.5 Cartographer 软件包的 ROS 框架

Google 开源的代码包含两个部分：cartographer 和 cartographer_ros。cartographer 主要负责处理来自雷达、IMU 和里程计的数据并基于这些数据进行地图的构建，是 cartographer 理论的底层实现。cartographer_ros 则基于 ros 的通信机制获取传感器的数据并将它们转换成cartographer 中定义的格式传递给 cartographer 处理，与此同时也将 cartographer 的处理结果发布用于显示或保存，是基于 cartographer 的上层应用。ceres-solver 是一个非线性优化的库。我们暂时可以不用管它，只需要知道当我们解决优化问题的时候可以调用这个库来求解即可。

在路威套件中，cartographer 的源码位于机器人端的~/cartographer_ws文件夹中。

```
[jubot@JUJON-ROBOT ~]$ls
cartographer_ws  Desktop  dl_ws  Downloads  jubot_ws  NoMachine  other  Software  Wallpapers
[jubot@JUJON-ROBOT ~]$cd cartographer_ws/
[jubot@JUJON-ROBOT ~/cartographer_ws]$ls
abseil-cpp  build_isolated  devel_isolated  install_isolated  src
[jubot@JUJON-ROBOT ~/cartographer_ws]$
```

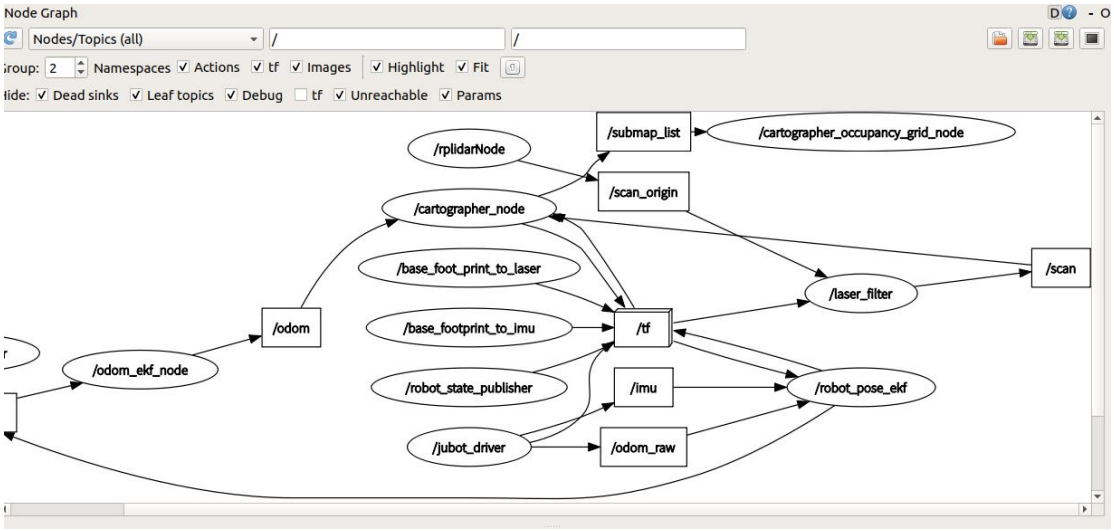
1.common：定义了基本数据结构和一些工具的使用接口。例如，四舍五入取整的函数、时间转化相关的一些函数、数值计算的函数、互斥锁工具等。

2.sensor：定义了传感器数据的相关数据结构。

3.transform：定义了位姿的数据结构及其相关的转化。如 2d\3d 的刚体变换等。

4.mapping：定义了上层应用的调用借口以及局部 submap 构建和基于闭环检测的位姿优化等的接口。这个文件也是算法的核心。其中mapping_2d 和 mapping_3d 是对mapping 接口的不同实现。

5. io：定义了一些与 IO 相关的工具，用于存读取数据、记录日志等。



在运行 cartographer_ros 建图算法时，需启动两个 cartographer 的节点，第一个是 cartographer_node 节点，该节点为 cartographer 算法核心节点，收听了 scan 和 odom 数据并输出 submap_list 数据，第二个节点是 cartographer_occupancy_grid_node 节点，将 cartographer 输出的 submap_list 融合成ROS 标准的map 栅格地图数据话题并发布。

## cartographer_node 收听话题

- **echoes**(sensor_msgs/MultiEchoLaserScan)多回波雷达数据。
- **scan**(sensor_msgs/LaserScan) 用于创建地图的激光雷达数据。
- **points2**(sensor_msgs/PointCloud2) 点云传感器数据。
- **imu**(sensor_msgs/Imu) IMU 传感器数据
- **odom**(nav_msgs/Odometry)里程计传感器数据

## cartographer_node 发布话题

- **scan_matched_points**(sensor_msgs/PointCloud2)匹配好的 2D 点云数据，用来 scan-to-submap matching。
- **submap_list**(cartographer_ros_msgs/SubmapList)发布构建好的 submap。

## cartographer_node 提供的 Service

- **submap_query**(cartographer_ros_msgs/SubmapQuery)提供查询submap 的服务，获取到 request 的 submap
- **start_trajectory**(cartographer_ros_msgs/StartTrajectory)开始维护一条轨迹
- **finish_trajectory**(cartographer_ros_msgs/FinishTrajectory)结束一个给定 ID 的轨迹
- **write_state**(cartographer_ros_msgs/WriteState)将当前状态写入文件保存
- **get_trajectory_states**(cartographer_ros_msgs/GetTrajectoryStates)获取指定 trajectory 的状态

## cartorgrapher_occupancy_grid_node 收听的话题

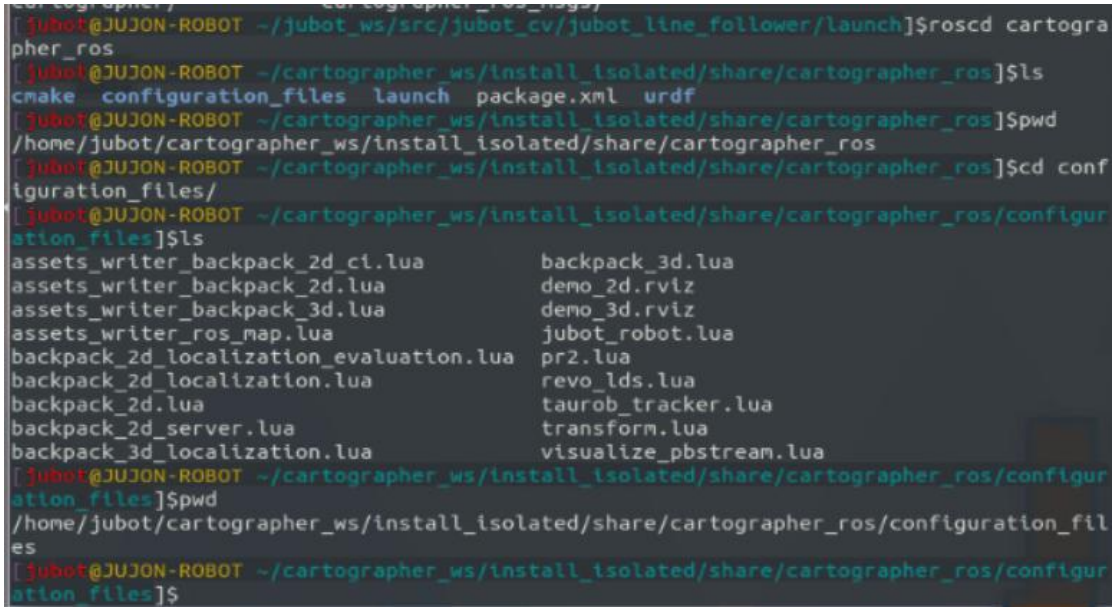
- **submap_list** (cartographer_ros_msgs/SubmapList) 由 cartographer_node 提供的 submap 数据

## cartorgrapher_occupancy_grid_node 收听的话题

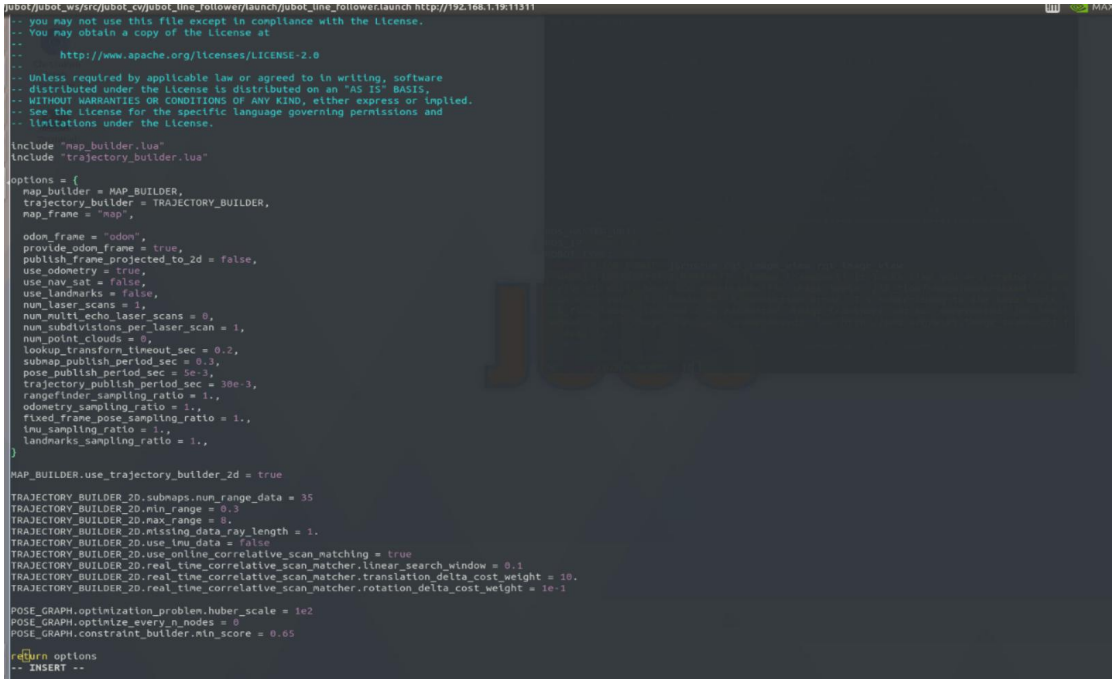
- map (nav_msgs/OccupancyGrid) ROS 标准格式的栅格地图。

## cartographer 建图算法参数

在路威套件中，cartographer 的配置参数位于机器人端 `~/cartographer_ws/install_isolated/share/cartographer_ros/configuration_files/jubot_robot.lua` 文件。用户可以通过 `roscd cartographer_ros` 命令直接定位到该文件夹，文件夹如下图：




配置文件内容如下图：



## 核心参数：

- map_frame 地图坐标系名称
- tracking_frame 路径追踪的坐标系的名称
- published_frame 发布坐标系变换的坐标系名称
- odom_frame 里程计坐标系名称
- provide_odom_frame 是否发布里程计坐标系变换
- publish_frame_projected_to_2d 如果启用，则已发布的 pose 将限制为纯 2D 姿势（无滚动，俯仰或 z 偏移）。
- use_odometry 是否使用odom 数据
- use_nav_sat 如果启用，请在主题"fix"上订阅 sensor_msgs
- num_laser_scan 订阅的激光雷达主题的数量
- num_multi_echo_laser_scans 订阅的多回波雷达主题的数量
- lookup_transform_timeout_sec 使用 TF 查找变换的超时时间
- submap_publish_period_sec 发布子图的时间间隔，单位是秒
- pose_publish_period_sec 发布 pose 的时间间隔，比如：5e-3 频率是 200Hz
- trajectory_publish_period_sec 以秒为单位发布轨迹标记的间隔，例如， 30e-3 持续 30 毫秒。

**采样比率相关参数：**

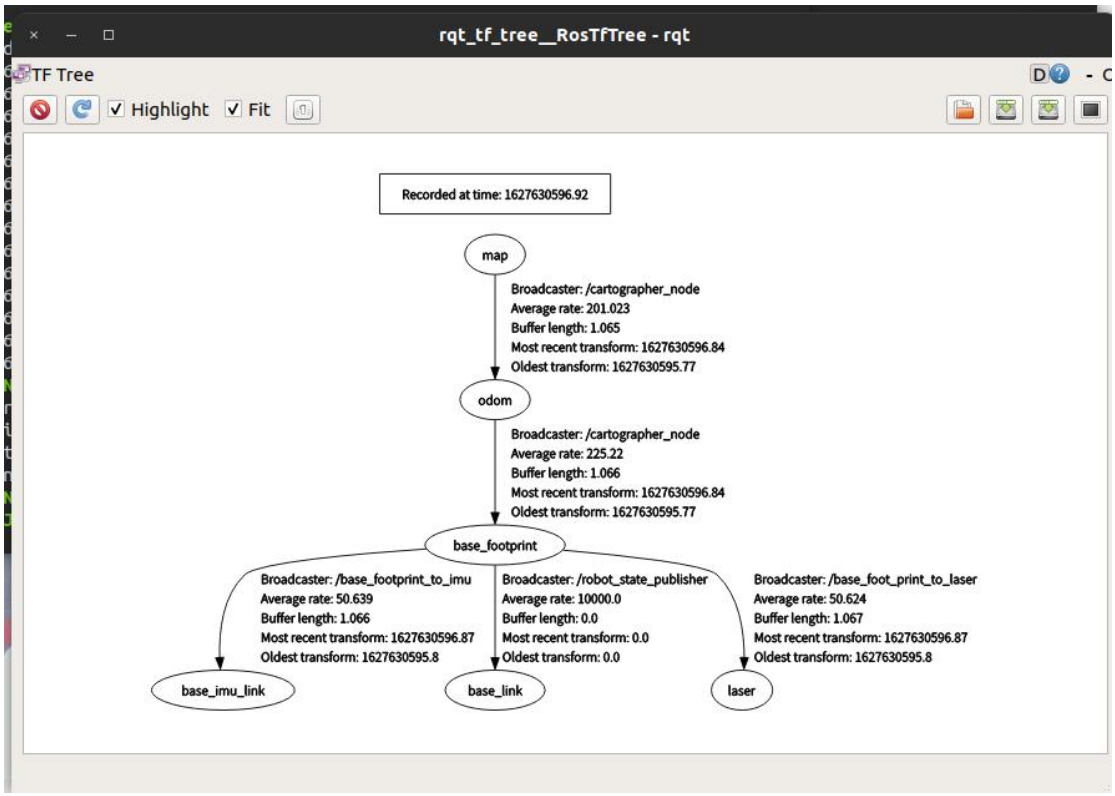
- rangefinder_sampling_ratio 激光雷达消息的固定采样频率
- odometry_sampling_ratio 里程计消息的固定采样频率
- fixed_frame_sampling_ratio 固定坐标系消息的固定采样频率
- imu_sampling_ratio IMU 消息的固定采样频率
- landmarks_sampling_ratio 路标消息的固定采样频率

cartographer 可配置参数较多，在此不一一列举，如要详细学习cartographer 算法，可以参照以下链接，通过官方文档学习：

Cartographer Github: https://github.com/cartographer-project/cartographer

Cartographer 官 方 Doc: https://google-cartographer.readthedocs.io/en/latest/configuration.html

## cartographer 的 TF 坐标发布

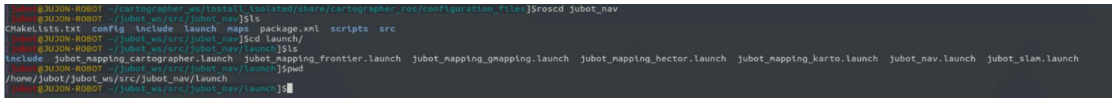



由图中可以看到，cartographer 节点发布了从map 到odom 的坐标变换，这就意味着 cartographer 节点提供了机器人在map 坐标系中的定位。

### 10.4.6 在 ROS 中部署 Cartographer 建图算法

在路威套件中，cartographer 建图算法的启动文件位于机器人端

`~/jubot_ws/src/jubot_nav/launch/jubot_mapping_cartographer.launch`




内容如下：

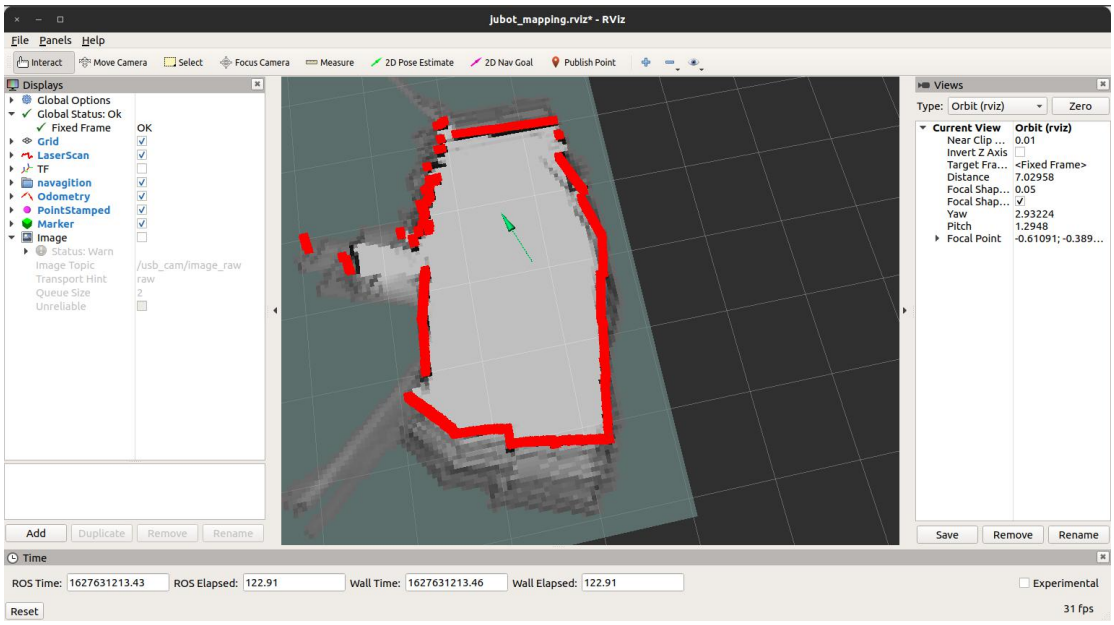
```
<?xml version="1.0"?>
<launch>
  <param name="/use_sim_time" value="false" />
  <arg name="resolution" default="480p"/>

  <include file="$(find jubot_driver)/launch/jubot_bringup.launch"/>
  <include file="$(find jubot_driver)/launch/jubot_camera.launch">
        <arg name="resolution" value="$(arg resolution)"/>
  </include>

  <node name="cartographer_node" pkg="cartographer_ros"
      type="cartographer_node" args="
          -configuration_directory $(find cartographer_ros)/configuration_files
          -configuration_basename jubot_robot.lua"
      output="screen">
    <remap from="scan" to="/scan" />
  </node>

  <node name="cartographer_occupancy_grid_node" pkg="cartographer_ros"
      type="cartographer_occupancy_grid_node" args="-resolution 0.05" />

</launch>
```

## 10.4.7 Cartographer 建图流程

`roslaunch jubot_nav jubot_slam.launch slam_methods:=cartographer`，连接底盘,连接激光雷达，发布底盘到雷达的 TF 变换。

按照上节所示操作，在虚拟机端运行Rviz可视化工具：`rosrun rviz rviz`

同上节所示，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：

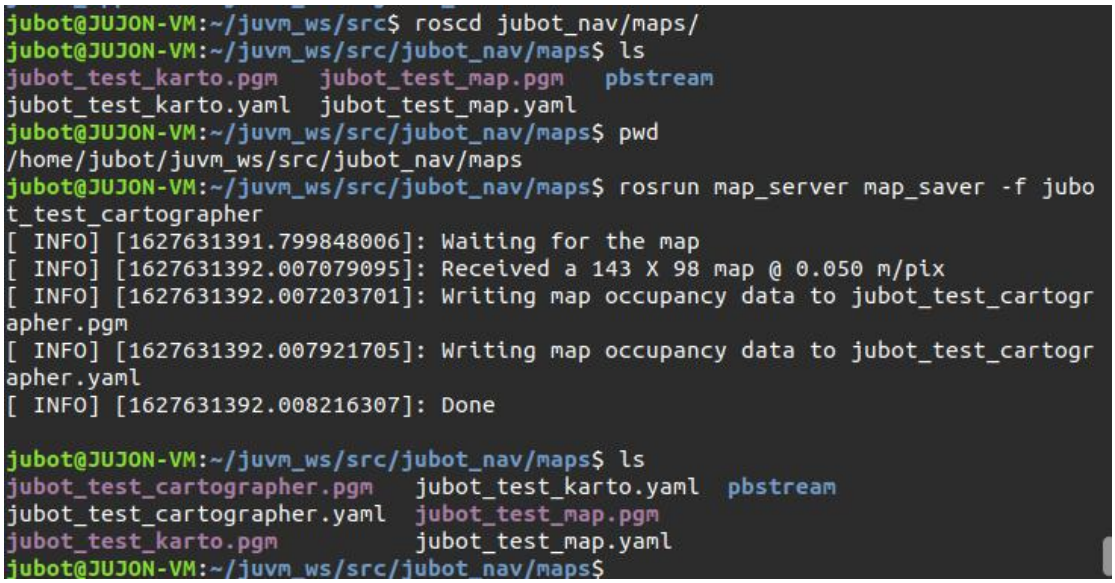
| 地图颜色 | 代表意义 |
| --- | --- |
| 红色 | 激光雷达探测到的障碍点（障碍物轮廓点阵） |
| 灰色 | 还没有探索到的未知区域 |
| 白色 | 已经探明的不存在静态障碍物的区域 |
| 黑色 | 静态障碍物轮廓 |

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，在虚拟机端进入到下图位置内通过以下指令：

```
rosrun map_server map_saver -f jubot_test_cartographer
```

保存地图，地图位置如下：




## 10.5 实验结果

使用 Carto & Cartographer 建立机器人所处环境的地图

## 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（上）

## 11.1 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。

## 11.2 实验要求

能够实现 路威套件 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。

## 11.3 实验工具

个人电脑一台，路威套件及其配件。

## 11.4 实验内容

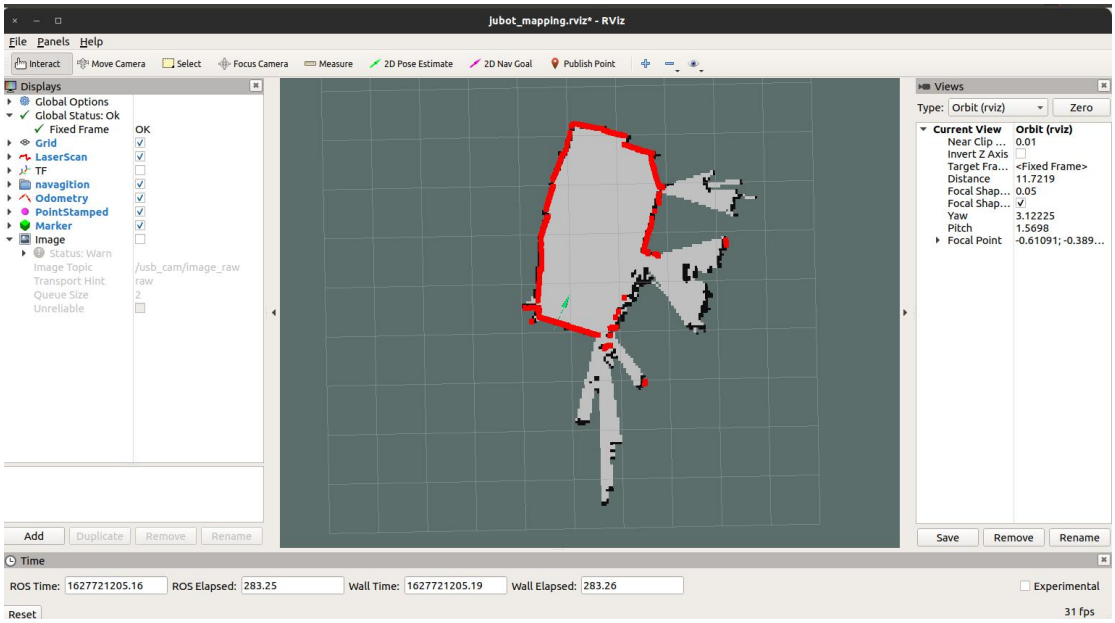
### 11.4.1 使用 Navigation 包实现路威套件的导航功能

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。

温习下前面一节，首先打开终端，输入

```
roslaunch jubot_nav jubot_slam.launch slam_methods:=gmapping
```

我们使用gmapping建一个地图



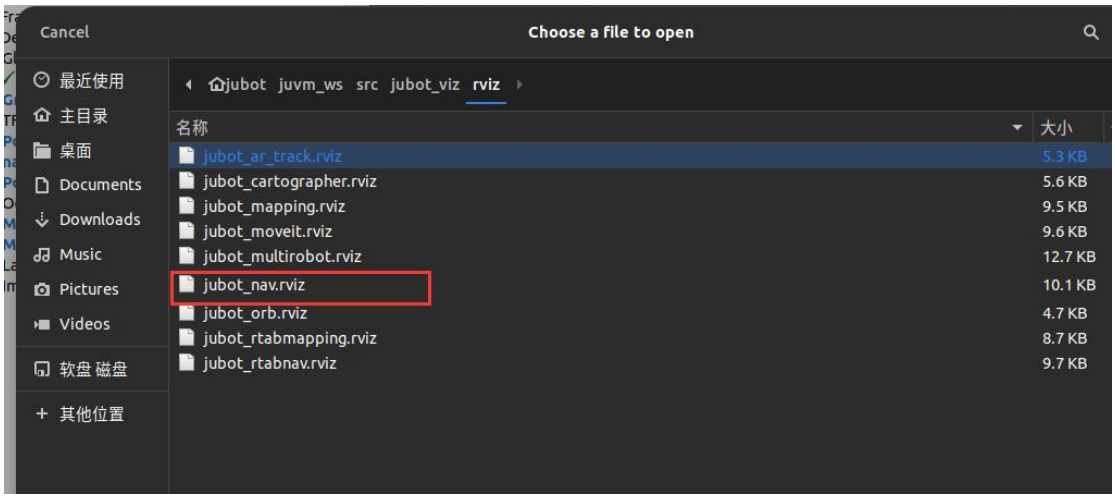

然后保存下来（注意这次我们直接保存在了机器人上，而不是虚拟机）：

```
[jubot@JUJON-ROBOT ~/other]$cd map/
[jubot@JUJON-ROBOT ~/other/map]$ls
[jubot@JUJON-ROBOT ~/other/map]$rosrun map_server map_saver -f nav
[ INFO] [1627721108.929211651]: Waiting for the map
[ INFO] [1627721109.207200547]: Received a 1984 X 1984 map @ 0.050 m/pix
[ INFO] [1627721109.207337894]: Writing map occupancy data to nav.pgm
[ INFO] [1627721109.552027471]: Writing map occupancy data to nav.yaml
[ INFO] [1627721109.552281175]: Done

[jubot@JUJON-ROBOT ~/other/map]$ls
nav.pgm  nav.yaml
[jubot@JUJON-ROBOT ~/other/map]$pwd
/home/jubot/other/map
[jubot@JUJON-ROBOT ~/other/map]$
```

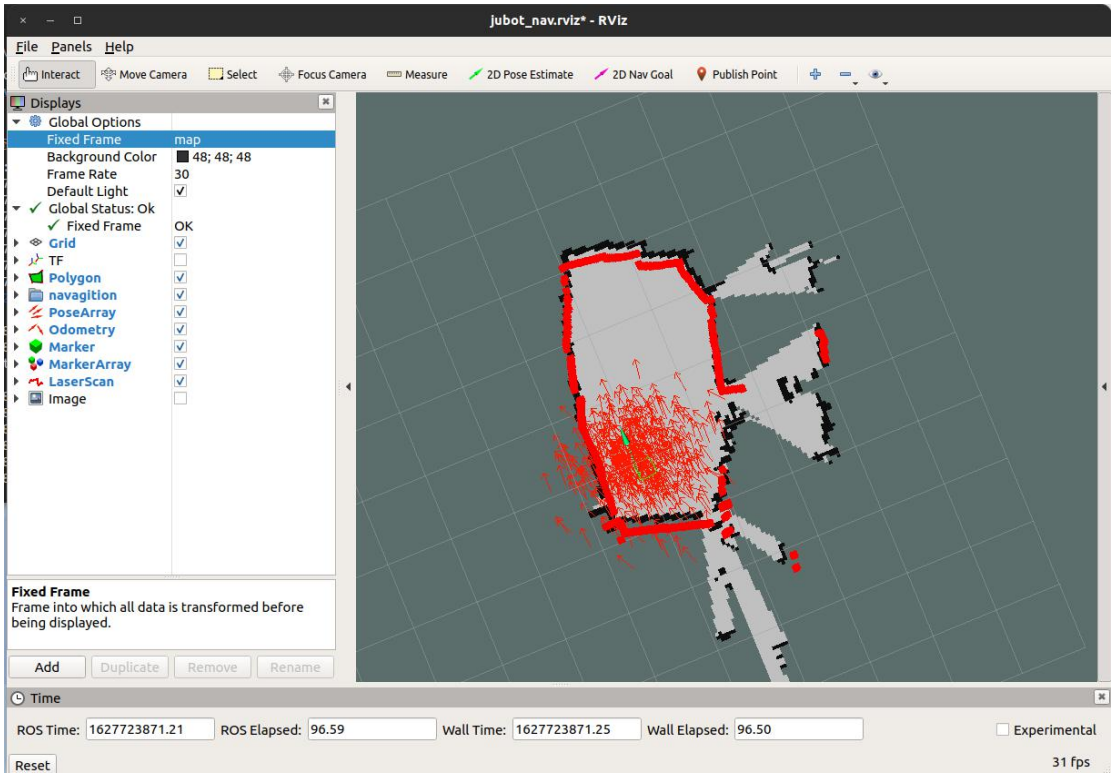
使用下面命令，注意地图存放路径为绝对路径。

```
roslaunch jubot_nav jubot_nav.launch map_file:=/home/jubot/other/map/nav.yaml
```


然后打开新终端，输入 `rosrun rviz rviz`，选择已经配置好的rviz文件：

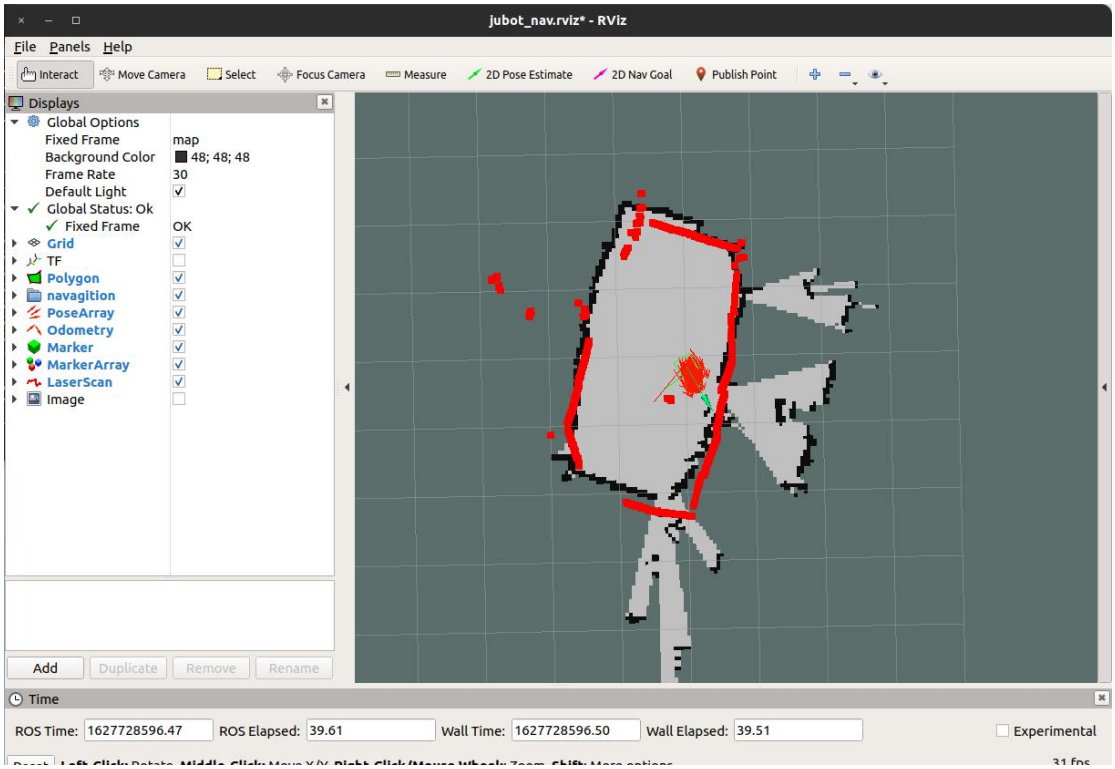



接下来就可以在 rviz 下看到机器人在环境场景中的一些信息了，如图：




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。

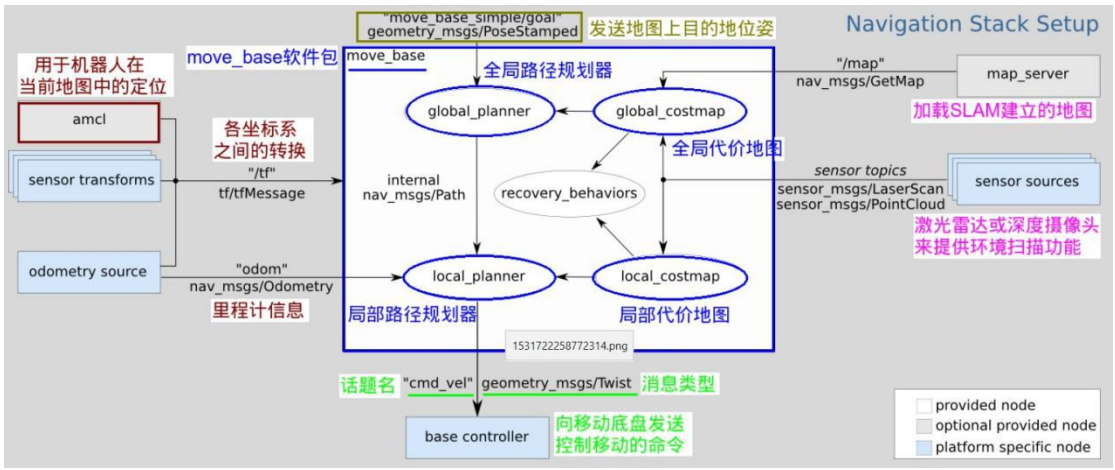
完成调整初始点之后就可以点击 2D Nav Goal 那个箭头，然后在地图中点击想要让机器人去的地方并调整朝向，机器人就会自动规划路径，然后朝着目标点行驶了，在行驶过程中，机器人也会自动去避障， 比如一个人突然出现在原先规划的路径上时，机器人也会重新规划局部路线，绕过这个人，展示效果如图：




## 11.4.2 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：

```
<!-- 启动Map server功能包，发布地图 -->
<arg name="map_file" default="$(find jubot_nav)/maps/jubot_test.yaml"/>
<node name="map_server" pkg="map_server" type="map_server" args="$(arg map_file)" />

<arg name="initial_pose_x" default="0.0"/>
<arg name="initial_pose_y" default="0.0"/>
<arg name="initial_pose_a" default="0.0"/>

<!-- 启动AMCL 自适应蒙特卡洛定位算法包 -->
<group if="$(eval arg('robot_type') == 'MEC')">
        <include file="$(find jubot_nav)/launch/include/amcl_omni.launch">
                <arg name="initial_pose_x" value="$(arg initial_pose_x)"/>
                <arg name="initial_pose_y" value="$(arg initial_pose_y)"/>
                <arg name="initial_pose_a" value="$(arg initial_pose_a)"/>
        </include>

        <!-- 启动路径规划算法包 -->
        <include file="$(find jubot_nav)/launch/include/teb_move_base_omni.launch"/>
</group>
<group if="$(eval arg('robot_type') == '4WD')">
        <include file="$(find jubot_nav)/launch/include/amcl_base.launch">
                <arg name="initial_pose_x" value="$(arg initial_pose_x)"/>
                <arg name="initial_pose_y" value="$(arg initial_pose_y)"/>
                <arg name="initial_pose_a" value="$(arg initial_pose_a)"/>
        </include>

        <!-- 启动路径规划算法包 -->
        <include file="$(find jubot_nav)/launch/include/teb_move_base.launch"/>
</group>

<include file="$(find rosbridge_server)/launch/rosbridge_websocket.launch"/>
<node name="robot_pose_publisher" pkg="robot_pose_publisher" type="robot_pose_publisher"/>
<arg name="debug" default="false"/>
<node pkg="world_canvas_server" type="world_canvas_server" name="world_canvas_server" args="$(arg debug)">
        <param name="start_map_manager" value="true"/>
        <param name="auto_save_map" value="false"/>
</node>
<node pkg="world_canvas_server" type="map_manager.py" name="map_manager"/>
```

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

### 11.4.3 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 意为失能此功能，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。被保存的位姿将会用在连续的运动上来初始化滤波器。-1.0 失能。

**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：--η λe^(-λz)，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η λe^(-λz)， 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（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，不明确是什么意思）.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：Zh * prob(dist,σ) +avg，Zh 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, likehood_field, likehood_field_prob（和 likehood_field 一样但是融合了beamskip 特征--官网的注释），默认是"likehood_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(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）

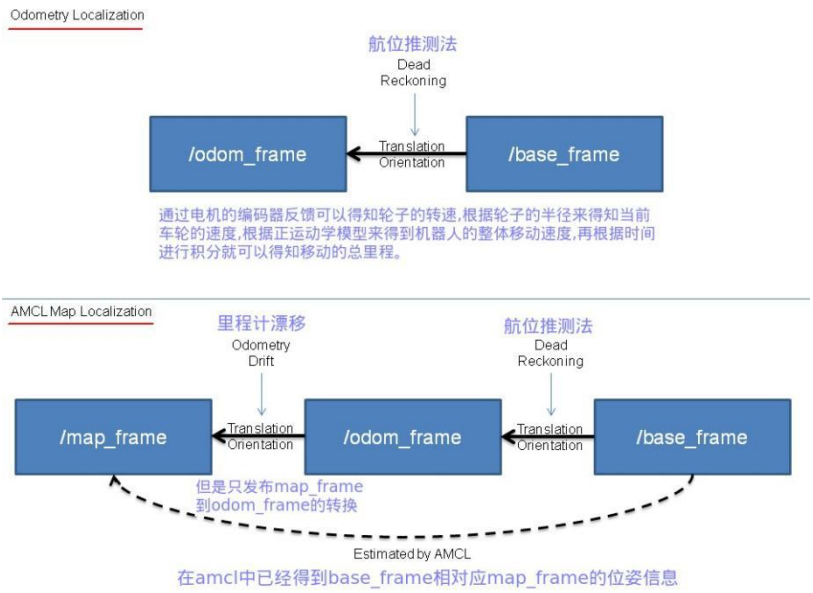
**odom_frame_id** (string, default: "odom")：里程计默认使用的坐标系。

**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。

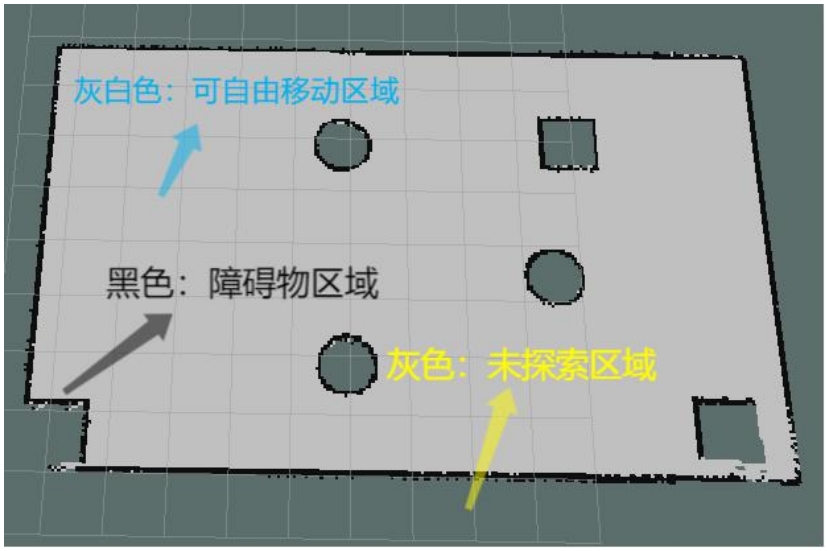
另外必须要注意的是，amcl 发布的 tf 变换是 map 到 odom 的变换，用于修正轮子打滑等问题导致的odom 不准确。




### 11.4.4 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 SLAM建图算法（例如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。

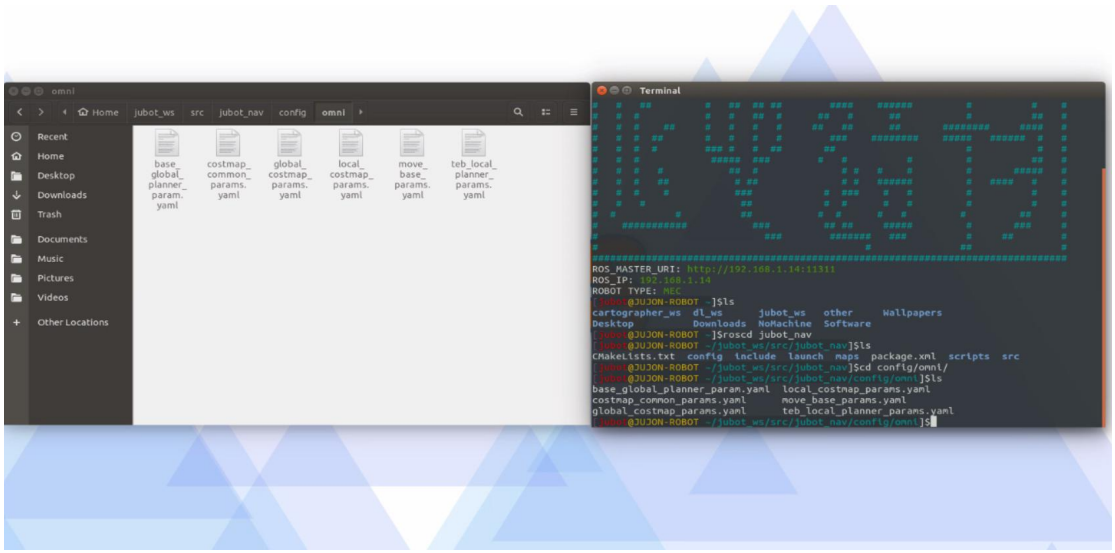



costmap 简单来说就是为了在这张地图上进行各种加工，方便我们后面进行路径规划而存在的，通过使用 costmap_2d 这个软件包来实现的，该软件包在原始地图上实现了两张新的地图。一个是 local_costmap， 另外一个就是 global_costmap，根据名字大家就可以知道了，两张 costmap 一个是为局部路径规划准备的，一个是为全局路径规划准备的。无论是 local_costmap 还是 global_costmap，都可以配置多个图层， 包括下面几种：

- **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 、

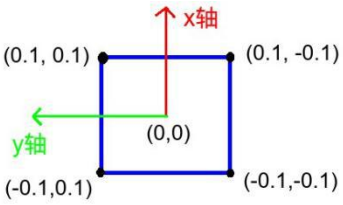
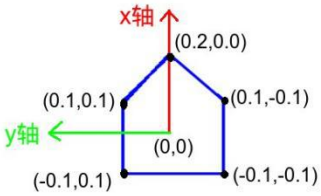
global_costmap_params.yaml 、

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：

```
footprint: [[-0.12, -0.12], [-0.12, 0.12],[0.12, 0.12], [0.12, -0.12]]
#robot_radius: 0.12
obstacle_layer:
  enabled: true
  max_obstacle_height: 0.6
  min_obstacle_height: 0.0
  obstacle_range: 2.0
  raytrace_range: 0.5
  #raytrace_range: 5.0
  inflation_radius: 0.20
  combination_method: 1
  observation_sources: laser_scan_sensor
  track_unknown_space: true

  origin_z: 0.0
  z_resolution: 0.1
  z_voxels: 10
  unknown_threshold: 15
  mark_threshold: 0
  publish_voxel_map: true
  footprint_clearing_enabled: true
```

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：

<table>
<tr>
<td>正方形的机器人</td>
<td> 
</td>
<td>footprint参数设置<br>顺时针<br>footprint: [[0.1,0.1],[0.1,-0.1],[-0.1,-0.1],[-0.1,0.1]]<br>逆时针<br>footprint: [[0.1,0.1],[-0.1,0.1],[-0.1,-0.1],[0.1,-0.1]]</td>
</tr>
<tr>
<td>五边形机器人</td>
<td> 
</td>
<td>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：<br>footprint: [[0.1,0.1],[0.2,0.0],[0.1,-0.1],[-0.1,-0.1],[-0.1,0.1]]</td>
</tr>
</table>

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来参与到路径规划，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。

```
footprint: [[-0.12, -0.12], [-0.12, 0.12],[0.12, 0.12], [0.12, -0.12]]
#robot_radius: 0.12
obstacle_layer:
  enabled: true
  max_obstacle_height: 0.6
  min_obstacle_height: 0.0
  obstacle_range: 2.0
  raytrace_range: 0.5
  #raytrace_range: 5.0
  inflation_radius: 0.20
  combination_method: 1
  observation_sources: laser_scan_sensor
  track_unknown_space: true

  origin_z: 0.0
  z_resolution: 0.1
  z_voxels: 10
  unknown_threshold: 15
  mark_threshold: 0
  publish_voxel_map: true
  footprint_clearing_enabled: true
```

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:  10.0  # exponential rate at which the obstacle cost drops off (default: 10)
  inflation_radius:     0.20 # max. distance from an obstacle at which costs are incurred for planning paths.

static_layer:
  enabled:              true
  map_topic:            "/map"
```

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*cost_scaling_factor*(distance_from_obstacle-inscribed_radius))*(costmap_2d::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: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：


global_frame:在局部代价地图中的全局坐标系，一般需要设置为 odom

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。

**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。

teb_local_planner:应用较好的局部规划算法，**路威套件默认使用的此算法。**

move_base_params.yaml **各参数的意义如下：**

```
shutdown_costmaps: false

controller_frequency: 4.0
controller_patience: 3.0 # 3.0

planner_frequency: 1.0
planner_patience: 3.0

oscillation_timeout: 5.0
oscillation_distance: 0.1

# Planner selection
base_global_planner: "global_planner/GlobalPlanner"
base_local_planner: "teb_local_planner/TebLocalPlannerROS"

max_planning_retries: 1

recovery_behavior_enabled: true
clearing_rotation_allowed: true

recovery_behaviors:
  - name: 'conservative_reset'
    type: 'clear_costmap_recovery/ClearCostmapRecovery'
    #type: 'move_slow_and_clear/MoveSlowAndClear'
  #- name: 'aggressive_reset'
  #  type: 'clear_costmap_recovery/ClearCostmapRecovery'
  #- name: 'super_reset'
  #  type: 'clear_costmap_recovery/ClearCostmapRecovery'
  - name: 'clearing_rotation'
    type: 'rotate_recovery/RotateRecovery'

conservative_reset:
  reset_distance: 1.0
  #layer_names: [static_layer, obstacle_layer, inflation_layer]
  layer_names: [obstacle_layer]

aggressive_reset:
  reset_distance: 3.0
  #layer_names: [static_layer, obstacle_layer, inflation_layer]
  layer_names: [obstacle_layer]

super_reset:
  reset_distance: 5.0
  #layer_names: [static_layer, obstacle_layer, inflation_layer]
  layer_names: [obstacle_layer]

move_slow_and_clear:
  clearing_distance: 0.5
  limited_trans_speed: 0.1
  limited_rot_speed: 0.4
  limited_distance: 0.3
```

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：**

```
# Planner selection
base_global_planner: "global_planner/GlobalPlanner"

base_local_planner: "teb_local_planner/TebLocalPlannerROS"

GlobalPlanner:                                  # Also see: http://wiki.ros.org/global_planner
  old_navfn_behavior: false                     # Exactly mirror behavior of navfn, use defaults for other boo
  use_quadratic: true                           # Use the quadratic approximation of the potential. Otherwise,
  use_dijkstra: true                            # Use dijkstra's algorithm. Otherwise, A*, default true
  use_grid_path: false                          # Create a path that follows the grid boundaries. Otherwise, u

  allow_unknown:  true                          # Allow planner to plan through unknown space, default true
                                                #Needs to have track_unknown_space: true in the obstacle / vox
  planner_window_x: 0.0                         # default 0.0
  planner_window_y: 0.0                         # default 0.0
  default_tolerance: 0.0                        # If goal in obstacle, plan to the closest point in radius def
```

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a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,偏向于直线穿越网格,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点.

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.

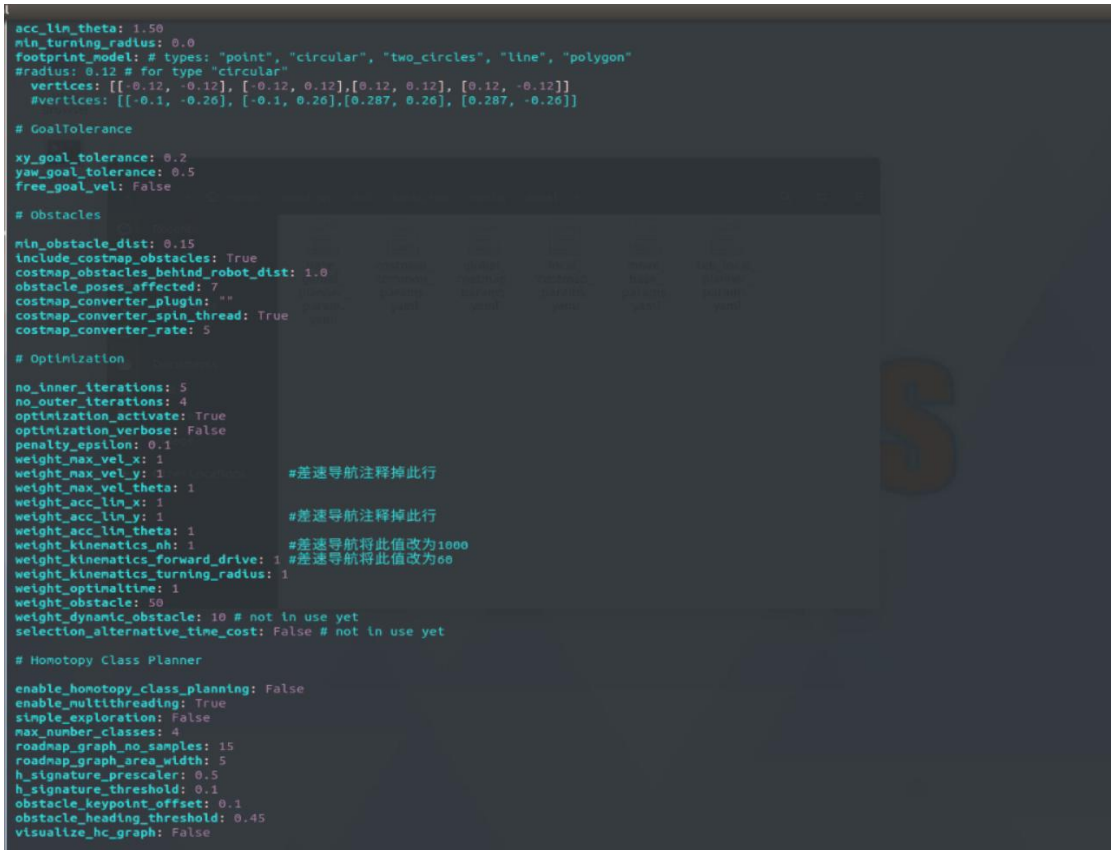
publish_potential：是否发布 costmap 的势函数.

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 teb_local_planner， teb_local_planner将navigation里的base_local_planner替换，故其作用机制和base_local_planner一样为ros的plugin机制。

**teb_local_planner_params.yaml 文件各参数意义：**

```
TebLocalPlannerROS:

 odom_topic: odom
 #odom_topic: /robot_pose_ekf/odom_combined
 map_frame: odom
 #map_frame: /odom_combined

 # Trajectory

 teb_autosize: True
 dt_ref: 0.45
 dt_hysteresis: 0.1
 global_plan_overwrite_orientation: True
 max_global_plan_lookahead_dist: 3.0
 feasibility_check_no_poses: 5

 # Robot

 max_vel_x: 0.7
 max_vel_y: 0.5    #差速导航注释掉此行
 max_vel_x_backwards: 0.35
 max_vel_theta: 2.0
 acc_lim_x: 0.85
 acc_lim_y: 0.85   #差速导航注释掉此行
 acc_lim_theta: 1.50
 min_turning_radius: 0.0
 footprint_model: # types: "point", "circular", "two_circles", "line", "polygon"
 #radius: 0.12 # for type "circular"
   vertices: [[-0.12, -0.12], [-0.12, 0.12],[0.12, 0.12], [0.12, -0.12]]
   #vertices: [[-0.1, -0.26], [-0.1, 0.26],[0.287, 0.26], [0.287, -0.26]]

 # GoalTolerance

 xy_goal_tolerance: 0.2
 yaw_goal_tolerance: 0.5
 free_goal_vel: False

 # Obstacles

 min_obstacle_dist: 0.15
 include_costmap_obstacles: True
 costmap_obstacles_behind_robot_dist: 1.0
 obstacle_poses_affected: 7
 costmap_converter_plugin: ""
 costmap_converter_spin_thread: True
 costmap_converter_rate: 5

 # Optimization

 no_inner_iterations: 5
 no_outer_iterations: 4
 optimization_activate: True
 optimization_verbose: False
 penalty_epsilon: 0.1
```

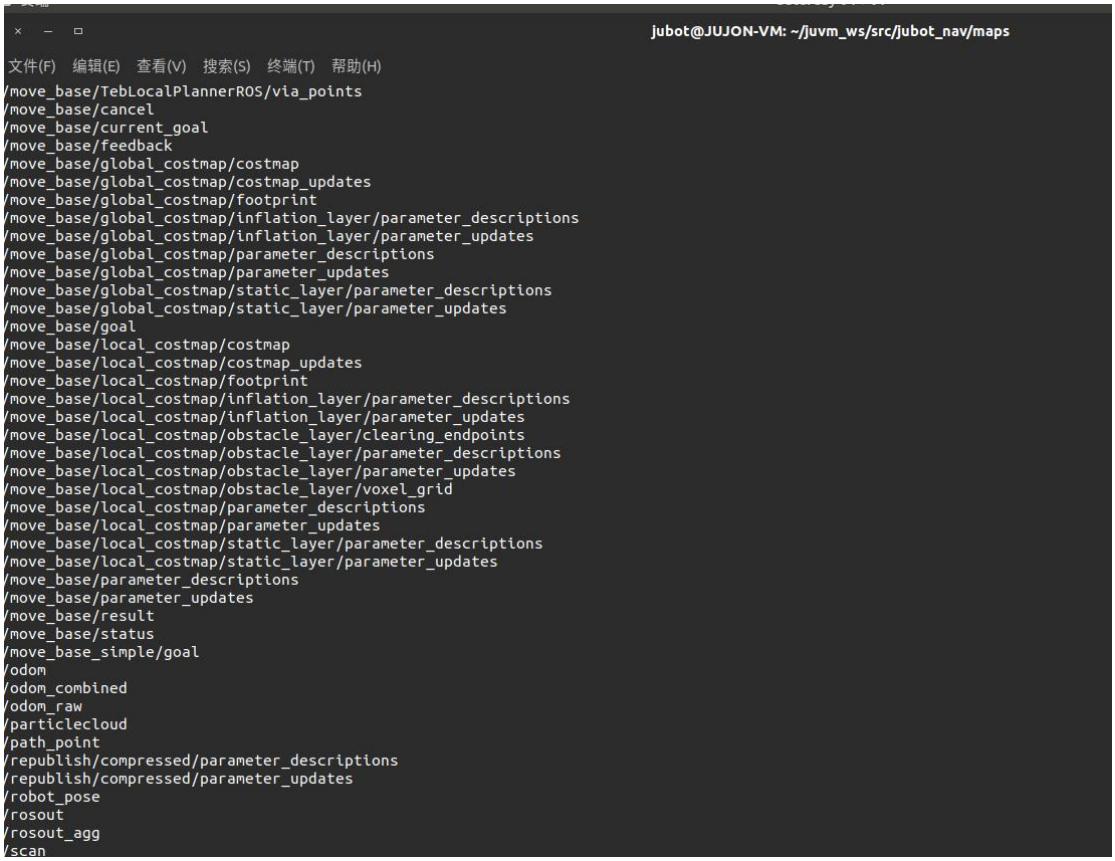



因 TEB local planner 算法参数较多，在此只对部分重要参数进行解释，详细参数定义及解释请参考TEBLocalPlanner 的官方 wiki-http://wiki.ros.org/teb_local_planner

- odom_topic 里程计话题消息
- map_frame 地图坐标系名称
- max_vel_x 机器人的最大 X 轴线速度
- max_vel_y 机器人的最大Y 轴线速度
- max_vel_x_backwards 机器人倒车的最大 X 轴线速度
- max_vel_theta 机器人最大旋转速度
- acc_lim_x 机器人 X 轴最大线加速度
- acc_lim_y 机器人 Y 轴最大线加速度
- acc_lim_theta 机器人最大旋转角加速度
- min_turning_radius 机器人最小转弯半径
- footprint_model 机器人形状类型，可选 point，circular，line， polygon 等，默认为 polygon 矩形
- vertices 通过点集描述矩形机器人的形状。

- xy_goal_tolerance 机器人到达目标点的 X、Y 坐标容差，机器人实际坐标与目标坐标小于这个值，则规划器认为机器人已到达目标点。
- yaw_goal_tolerance 机器人到达目标姿态的 Yaw 轴朝向容差，机器人实际朝向与目标姿态朝向小于这个值，则规划器认为机器人已经到达目标姿态。
- free_goal_vel 消除目标速度限制，使机器人可以最大速度到达目标。
- min_obstacle_dist 与障碍物的最小期望距离。
- weight_max_vel_x 满足最大允许 X 轴线速度的优化权重
- weight_max_vel_y 满足最大允许Y 轴线速度的优化权重
- weight_max_vel_theta 满足最大允许Yaw 轴角速度的优化权重
- weight_acc_lim_x 满足最大允许 X 轴线加速度的优化权重
- weight_acc_lim_y 满足最大允许Y 轴线加速度的优化权重
- weight_acc_lim_theta 满足最大允许Yaw 轴角加速度的优化权重
- weight_kinematics_nh 用于满足非完整运动学的优化权重（此参数必须很高，因为运动学方程构成了一个等式约束，即使值 1000 也不意味着由于与其他成本相比较小的"原始"成本值而导致的矩阵条件不佳）。
- weight_kinematics_forward_driver 优化权重，用于迫使机器人仅选择前进方向（正向速度）。较小的值（例如 1.0）仍然允许向后行驶。大约1000 的值几乎可以防止向后驱动（但不能保证）。

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：




## 11.5 实验结果

能够让路威套件自主导航到指定位置，理解 costmap，planner 中的各种参数含义

## 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 Navigation 自主导航（下）

## 12.1实验目的

实现通过代码自主发布航点，实现自主导航。

## 12.2实验要求

熟悉导航原理以及各个参数的作用，实现复杂地图的自主运动规划。

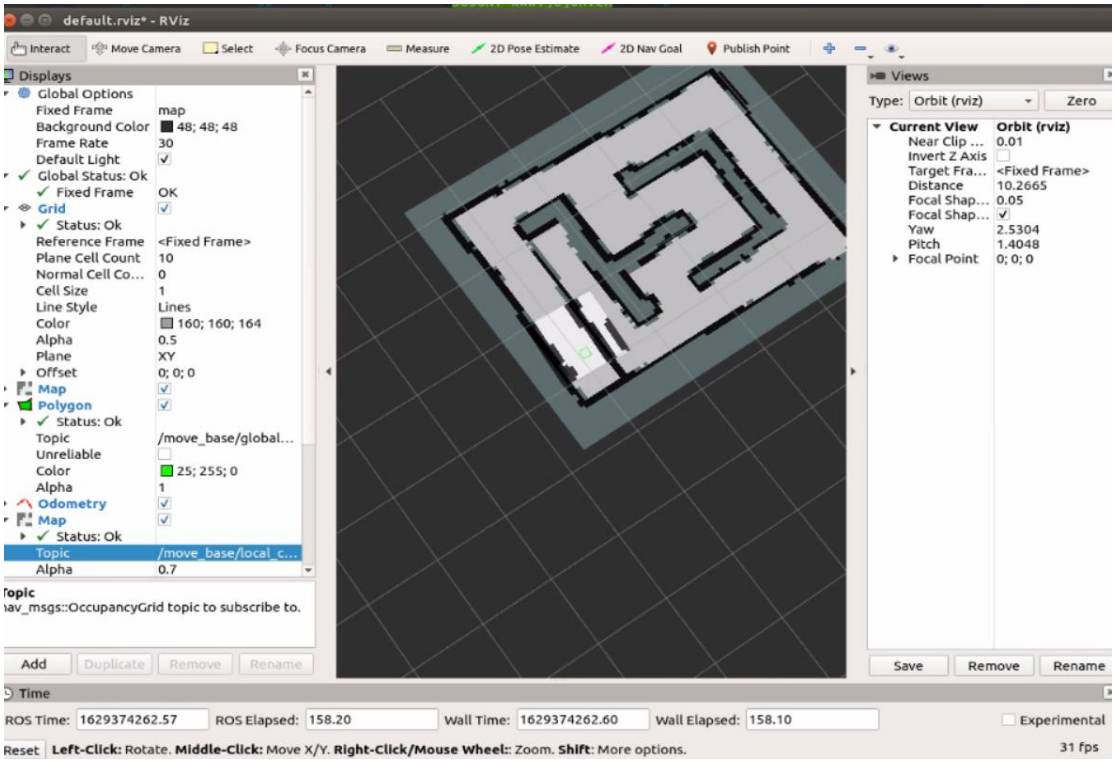
## 12.3实验工具

个人电脑一台，路威套件。

## 12.4实验内容

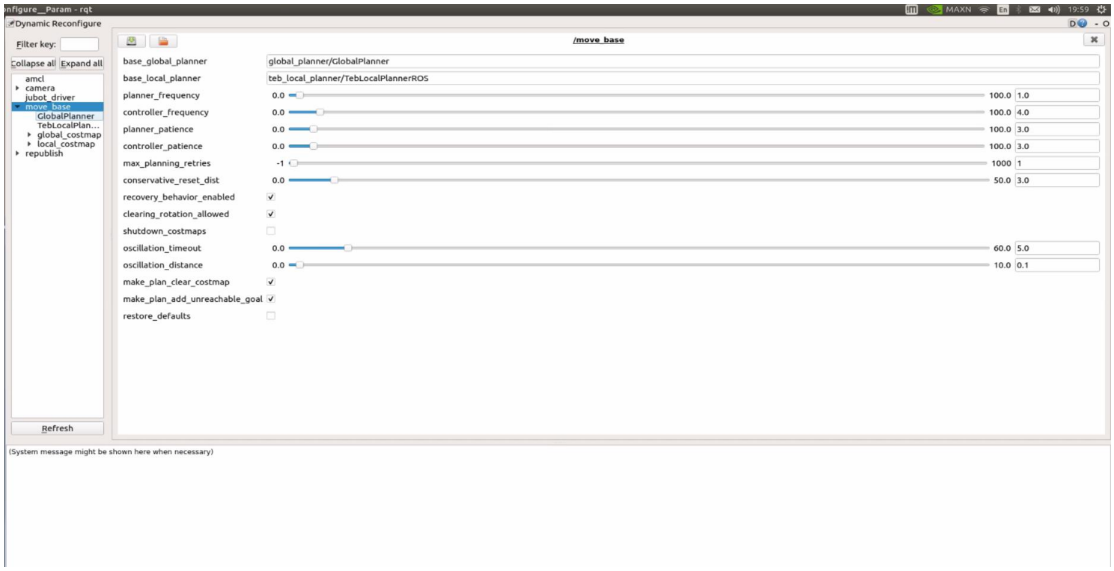
### 12.4.1运行流程

首先建造一个较为复杂的地图，打开终端，输入 `roslaunch jubot_nav jubt_nav.launch` 运行导航功能，然后打开新终端,输入 `rosrun rviz rviz`，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：




如上图所示我们可以建造一些复杂的地图，然后运行

`rosrun rqt_reconfigure rqt_reconfigure` 即可进行可视化的调参，使得我们的导航变得精准。




## 12.5 实验结果

能够让路威套件自主导航到指定位置，理解 costmap，planner 中的各种参数含义。熟练使用rqt_reconfigure工具，掌握导航算法中的各个参数。

## 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 单目相机驱动

## 13.1 实验目的

可以在 ros 系统中使用单目 usb 摄像头。

## 13.2 实验要求

掌握 ros 中 usb 摄像头驱动，ros 中 usb 相机数据格式。

## 13.3 实验工具

个人电脑一台，路威套件。

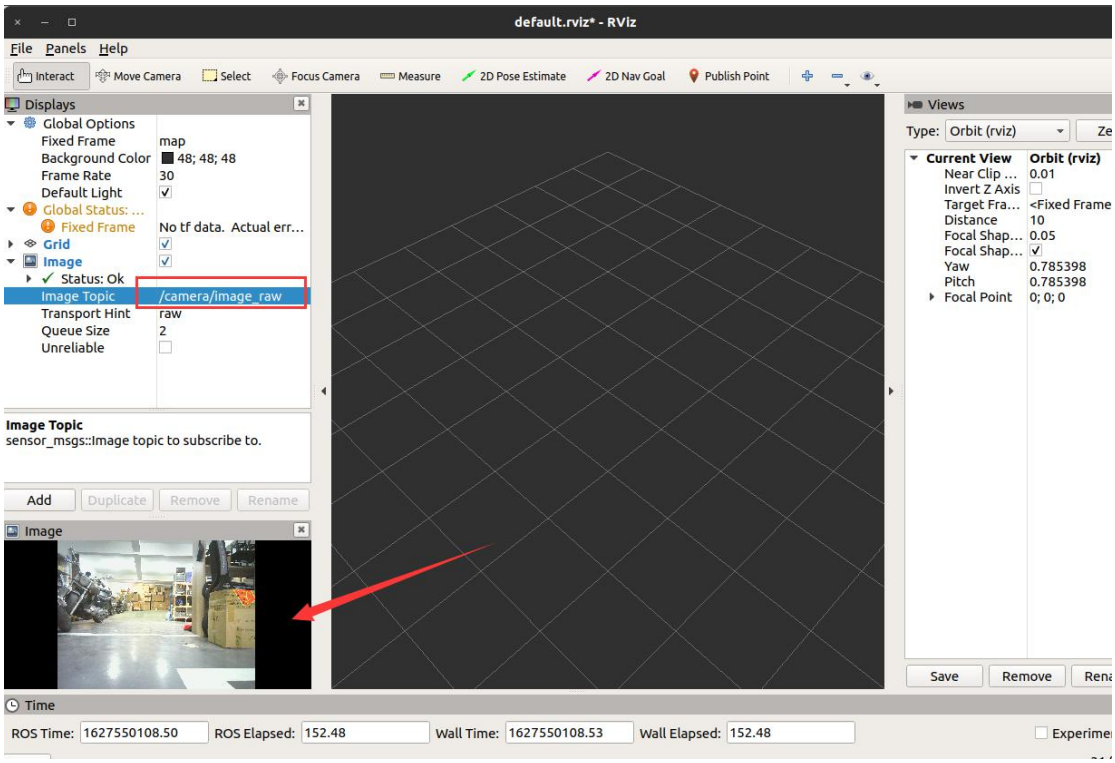
## 13.4 实验内容

### 13.4.1 摄像头消息基本概念

首先连接到我们的路威套件，打开新的终端输入：

```
roslaunch jubot_driver jubot_camera.launch
```

然后再在虚拟机端运行 `rosrun rviz rviz`




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 /camera/image_raw
```

```
jubot@JUJON-VM:~/juvm_ws/src$ rostopic info /camera/image_raw
Type: sensor_msgs/Image

Publishers:
 * /republish (http://192.168.1.19:33451/)

Subscribers:
 * /rviz_1627549955362319289 (http://192.168.1.18:43609/)
```

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
```

```
jubot@JUJON-VM:~/juvm_ws/src$ 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。

**header**：消息头，包含图像的序号、时间戳和绑定坐标系。

**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。

**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。

**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。

**is_bigendian**：图像数据的大小端存储模式。

**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×3=1280×3=3840 字节。

**data**：存储图像数据的数组，大小为 step×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：3840×720= 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接受不了的，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，图像在传输前往往会进行压缩处理，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。

如图为压缩后的实际带宽数据。

```
jubot@JUJON-VM:~/juvm_ws/src$ rostopic bw /camera/image_raw/compressed
subscribed to [/camera/image_raw/compressed]
average: 499.75KB/s
        mean: 67.71KB min: 67.67KB max: 67.75KB window: 2
average: 585.58KB/s
        mean: 67.68KB min: 67.58KB max: 67.81KB window: 11
average: 625.02KB/s
        mean: 67.68KB min: 67.57KB max: 67.81KB window: 21
average: 619.98KB/s
        mean: 67.68KB min: 67.57KB max: 67.81KB window: 30
average: 474.92KB/s
        mean: 67.68KB min: 67.57KB max: 67.81KB window: 30
^[^Aaverage: 384.87KB/s
        mean: 67.68KB min: 67.57KB max: 67.81KB window: 30
average: 323.52KB/s
        mean: 67.68KB min: 67.57KB max: 67.81KB window: 30
average: 279.01KB/s
```

### 13.4.2 摄像头图像显示

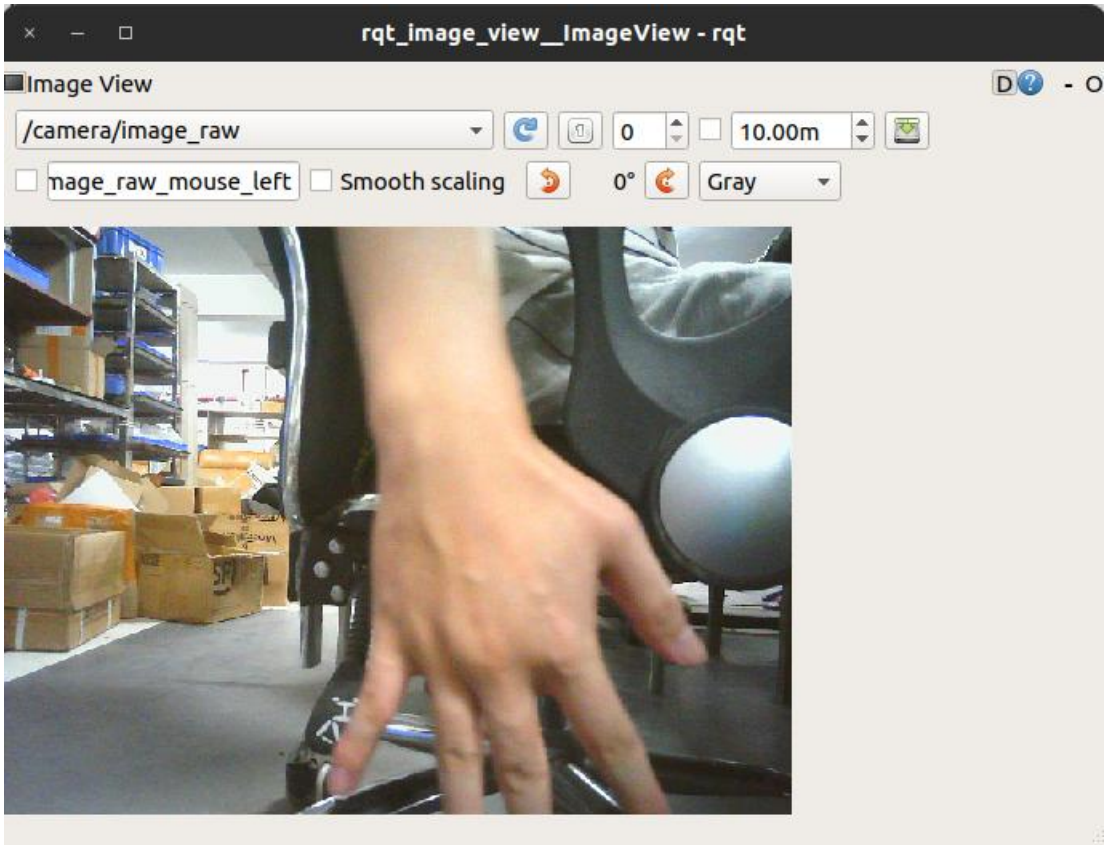
#### rqt_image_view 查看图像

通过 SSH 命令连接到机器人，运行摄像头节点。

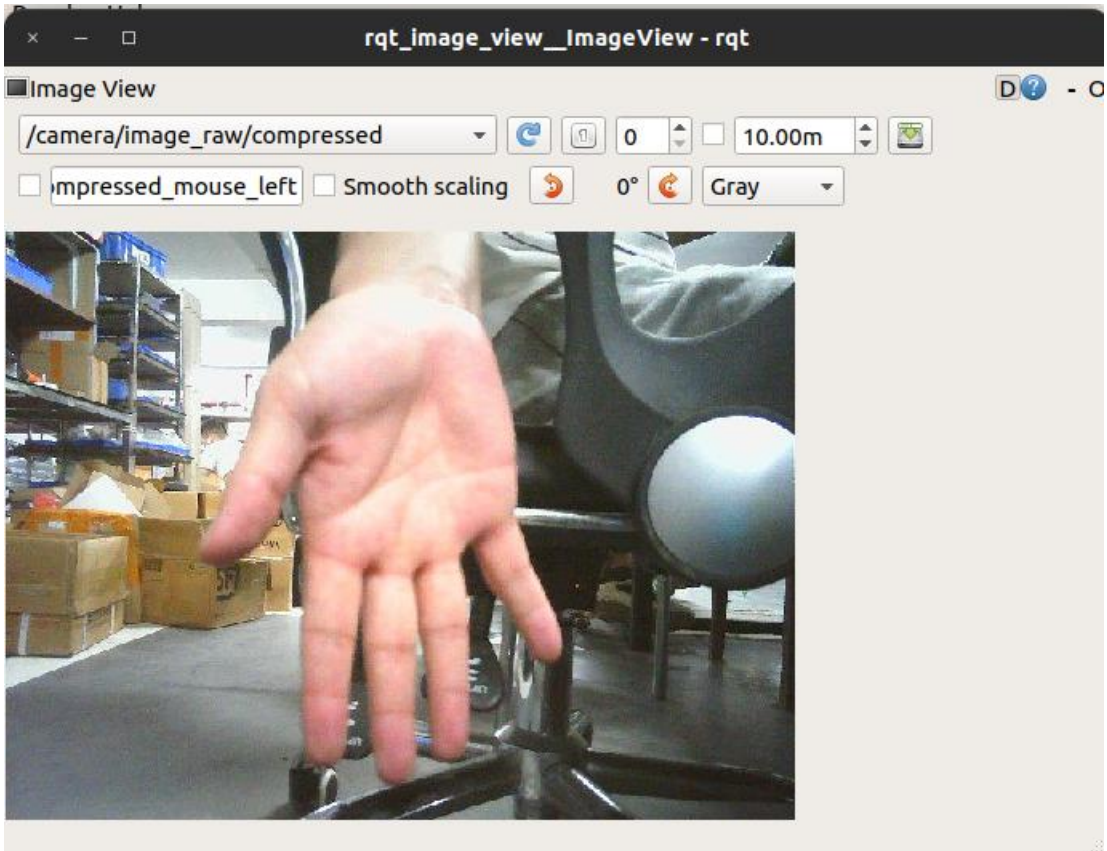
```
roslaunch jubot_driver jubot_camera.launch
```

再新建一个终端，在虚拟机端运行摄像头显示工具。

```
rosrun rqt_image_view rqt_image_view
```




选择中/camera/image_raw，即可看到摄像头未压缩的原始数据画面，可以明显观察到原始数据图像卡顿，帧率较低。选择压缩后的图像后，图像显示流畅。



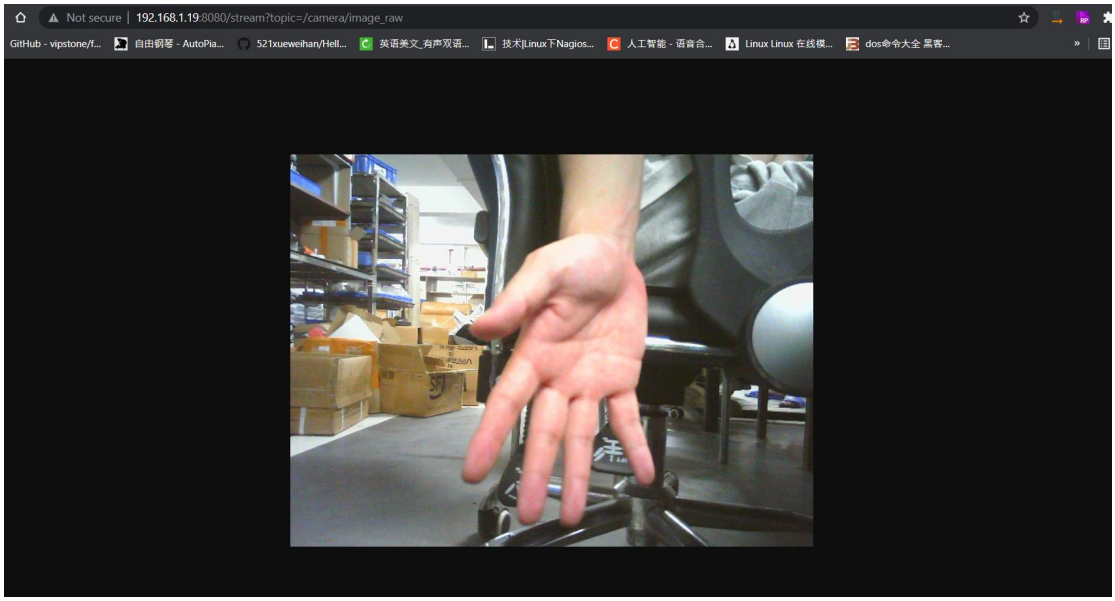
**通过 WEB 端的显示摄像头信息**

摄像头启动后，会启动一个基于 WEB 的图像服务器，用户可在浏览器中输入网址实现机器人摄像头数据的浏览器显示。具体使用方法如下。

通过 SSH 命令连接到机器人，运行摄像头节点。

在任意同网段浏览器中输入如下网址，注意 IP 地址修改为Jetson nano实际的 IP 地址。

http://192.168.1.19:8080/stream?topic=/camera/image_raw




**摄像头分辨率调节**

机器人搭载的摄像头支持多种分辨率格式，可以启动时设置参数来控制摄像头的分辨率。启动方法如下：

```
roslaunch jubot_driver jubot_camera.launch resolution:=480p
```

resolution 可选参数如下：

480p：摄像头以 640x480 分辨率启动

720p：摄像头以 1280x720 分辨率启动

1080p：摄像头以 1920x1080 分辨率启动

如不带 resolution 参数启动，摄像头默认以 480p 分辨率运行，分辨率越高，图像处理例程的 demo 运行速度相应越慢。

在运行其他使用到摄像头的 demo 中，如 opencv 图像处理例程以及 jubot_line_follower 例程，均可以使用在启动 launch 文件时带 resolution 参数的方法来控制摄像头的使用分辨率。

## 13.5 实验结果

了解ros 中 usb 相机数据格式，能够驱动 usb 摄像头，在 rqt_image_view 和WEB端显示图像信息。

## 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 单目相机参数标定实验

## 14.1 实验目的

可以在 ros 系统中标定单目 usb 摄像头。

## 14.2 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。

## 14.3 实验工具

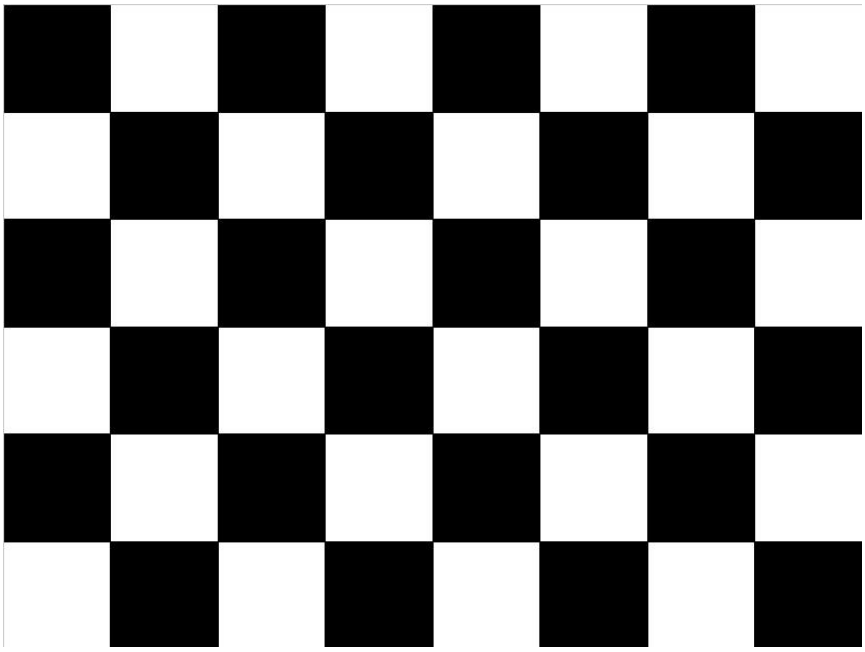
个人电脑一台，路威套件。

## 14.4 实验内容

### 14.4.1 摄像头标定实验

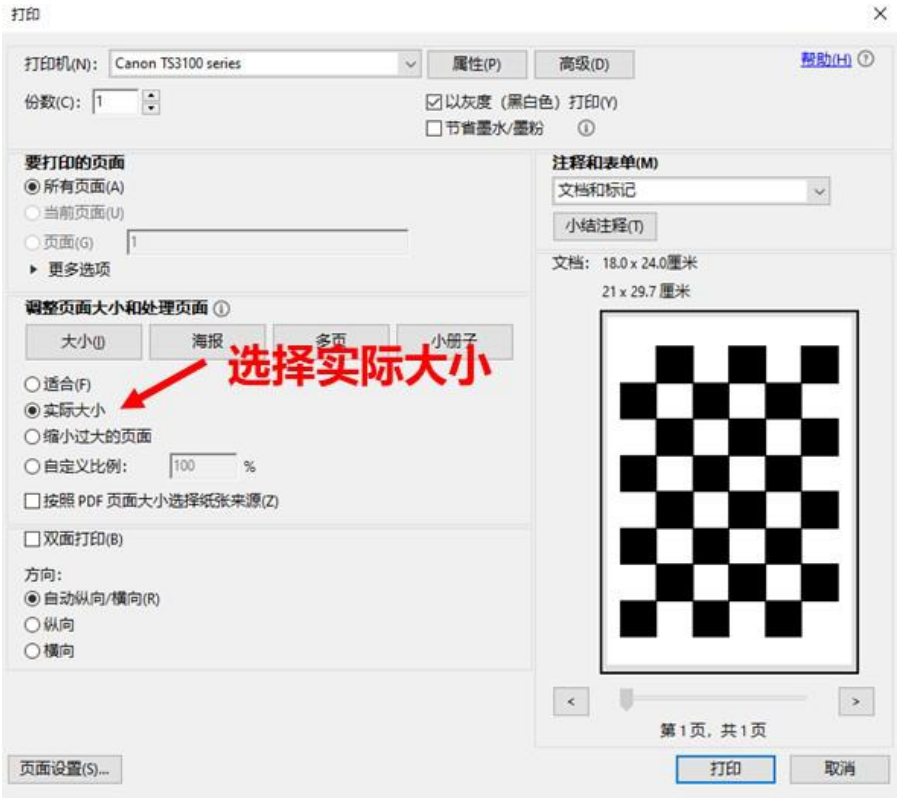
摄像头这种精密仪器对光学器件的要求较高，由于摄像头内部与外部的一些原因，生成的物体图像往往会发生畸变，为了避免数据源造成的误差，需要针对摄像头的参数进行标定。ROS 官方提供了用于双目和单目摄像头标定的功能包——camera_calibration。

标定需到棋盘格图案的标定靶，可以在我们提供的资料当中找到，请你将该标定靶打印出来贴到平面硬纸板上以备使用。



标定需要原始的标定板大小，所以标定板打印时需要原始尺寸打印， 打印时选择"

实际大小”。

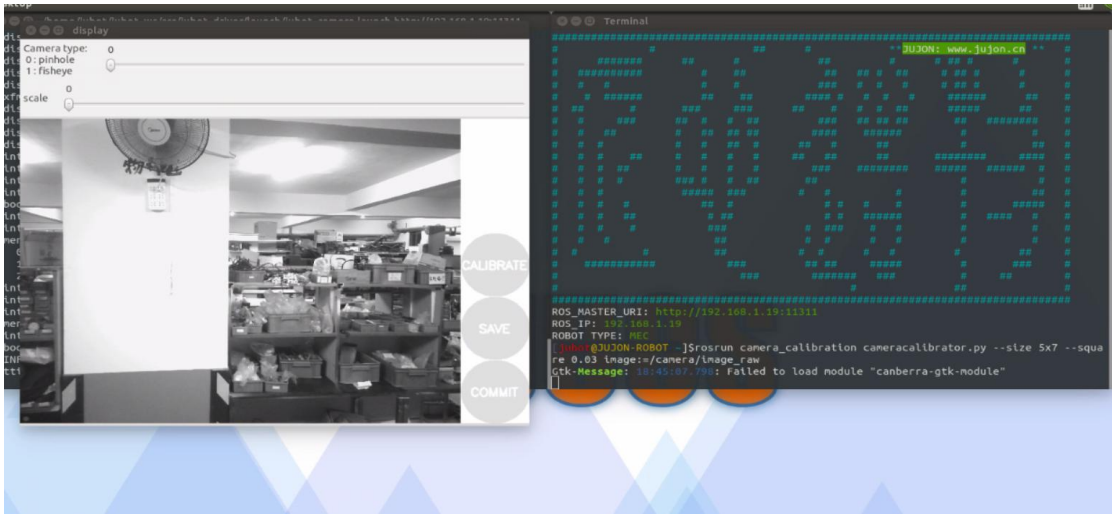



一切就绪后准备开始标定摄像头。首先使用以下命令启动 USB 摄像头：

```
roslaunch jubot_driver jubot_camera.launch
```

然后使用以下命令启动标定程序：

```
rosrun camera_calibration cameracalibrator.py --size 5x7 --square 0.03 image:=/camera/image_raw
```




**cameracalibrator.py 标定程序需要以下几个输入参数：**

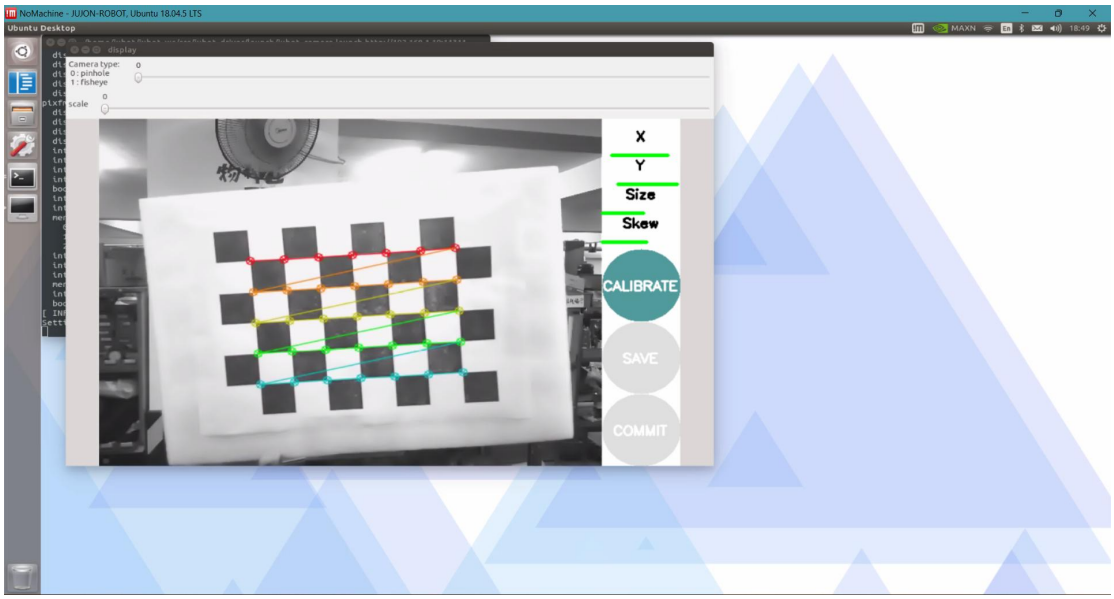
1）size：标定棋盘格的内部角点个数，这里使用的棋盘一共有 6 行，每行有 8 个

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。

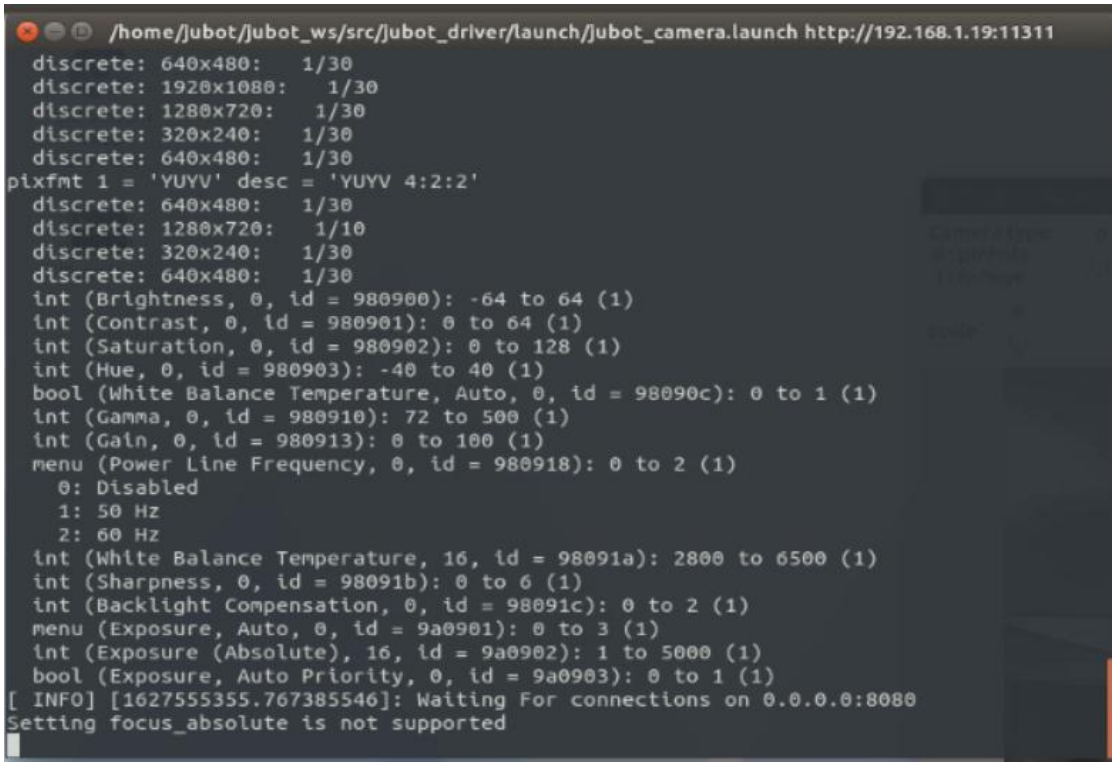
1）X：标定靶在摄像头视野中的左右移动。

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。

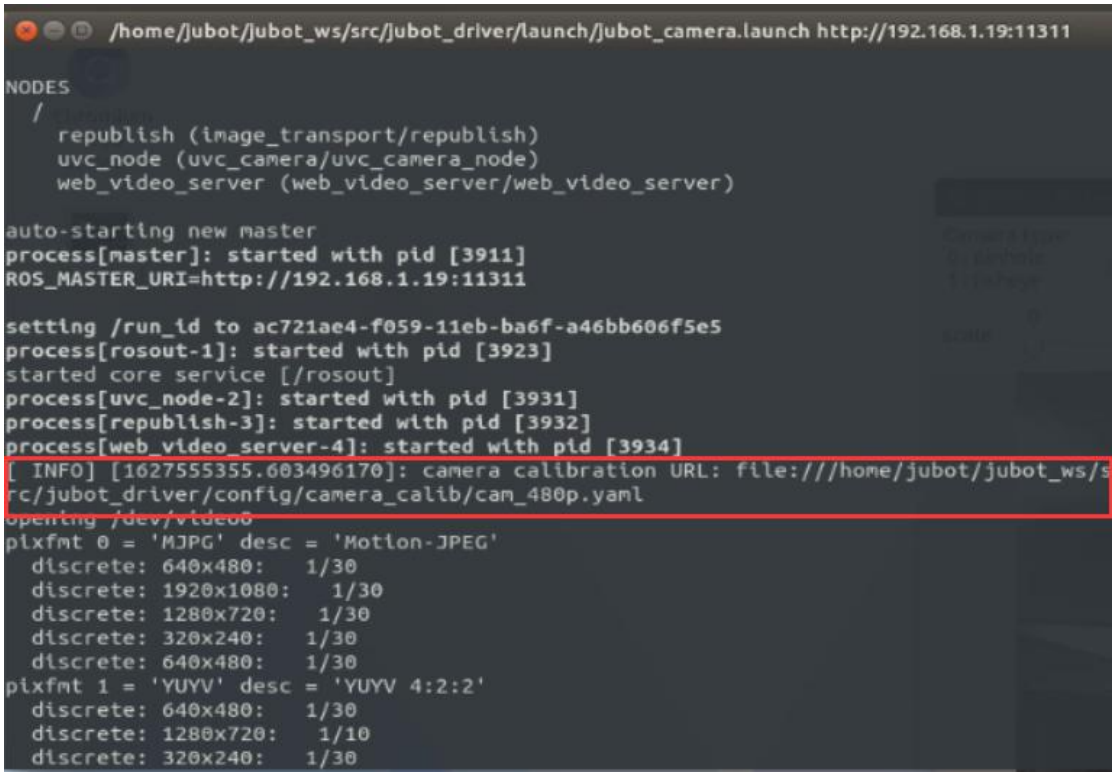
不断在视野中移动标定靶，直到"CALIBRATE"按钮变色，表示标定程序的参数采集完成。点击"CALIBRATE"按钮，标定程序开始自动计算摄像头的标定参数，这个过程需要等待一段时间，界面可能会变成灰色无响应状态，注意千万不要关闭。参数计算完成后界面恢复，而且在终端中会有标定结果的显示。



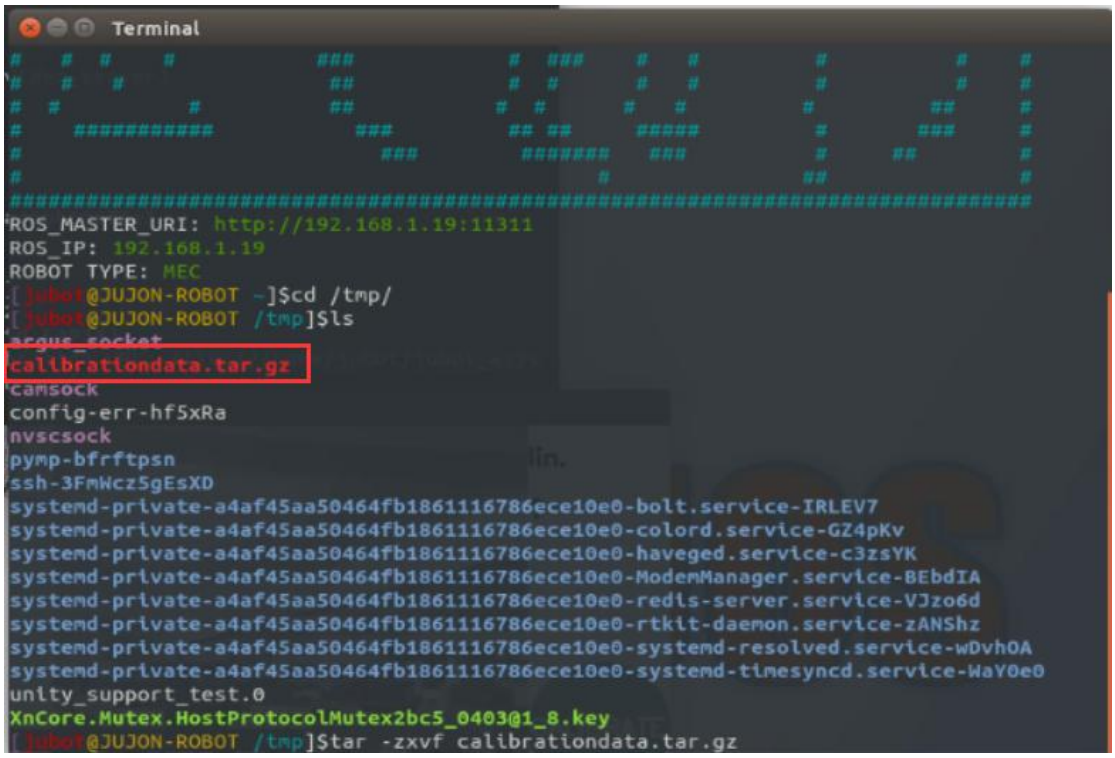

点击界面中的"SAVE"按钮，标定参数将被保存到默认的文件夹下，并在终端中看到该路径。

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

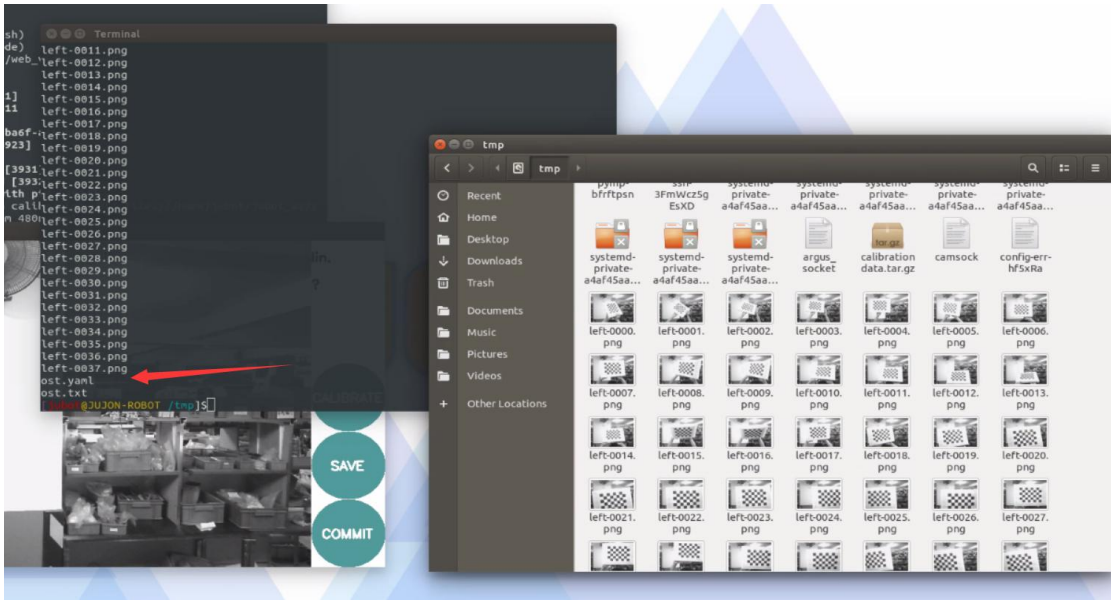
最后，点击 COMMIT 按钮将标定参数结果保存到机器人默认文件夹，在摄像头启动的终端窗口中可以看到如下信息，说明标定结果文件已经保存到我们的配置文件夹下，下次启动相机节点时，会自动调用。相机标定工具会自动退出关闭。



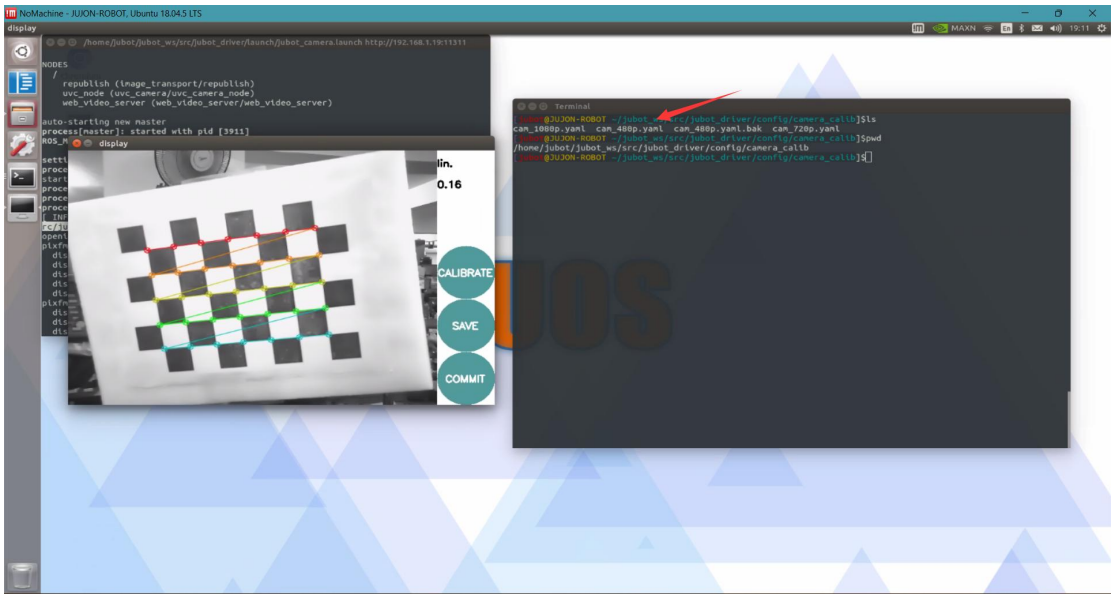

也可以在点击"COMMIT"按钮后，提交数据并退出程序。然后打开/tmp 文件夹，就可以看到标定结果的压缩文件 calibrationdata.tar.gz；




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 重名名为 cam_480p.yaml 放在以下文件夹内




再重新启动jubot_camera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。

## 14.5 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。

## 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

## 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。

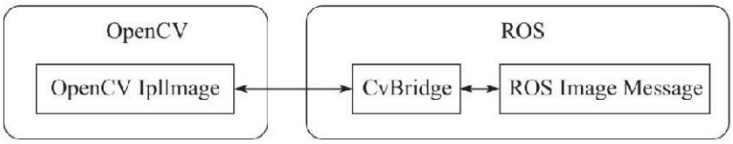
## 15.3 实验工具

个人电脑一台，路威套件。

## 15.4 实验内容

### 15.4.1 OpenCV介绍

基于 OpenCV 的人脸识别和物体跟踪：OpenCV 是图像处理中的"利器"，ROS 中的 cv_bridge 功能包为两者提供了接口，赋予 ROS 应用强大的图像处理能力，可以轻松实现人脸识别、物体跟踪等多种功能。



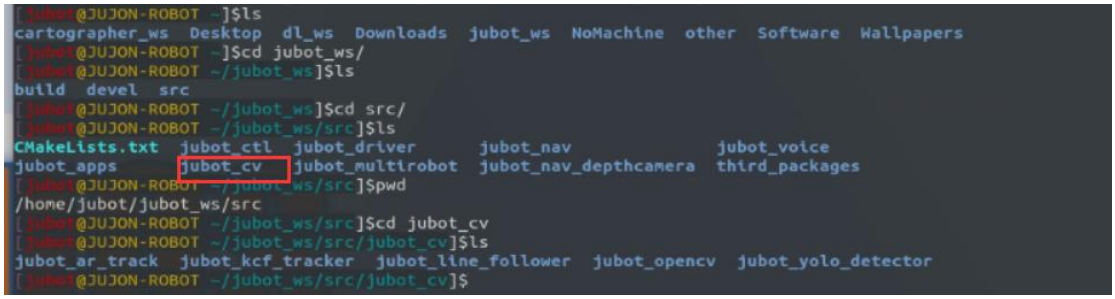

OpenCV 库（Open Source Computer Vision Library）是一个基于 BSD 许可发行的跨平台开源计算机视觉库，可以运行在 Linux、Windows 和mac OS 等操作系统 上。OpenCV 由一系列 C 函数和少量 C++类构成，同时提供 C++、Python、Ruby、MATLAB 等语言的接口，实现了图像处理和计算机视觉方面的很多通用算法，而且对非商业应用和商业应用都是免费的。同时 OpenCV 可以直接访问硬件摄像头，并且还提供一个简单的 GUI 系统——HighGui。

基于 OpenCV 库，我们可以快速开发机器视觉方面的应用，而且 ROS 中已经集成了OpenCV 库和相关的接口功能包，使用以下命令即可安装：

ROS 为开发者提供了与 OpenCV 的接口功能包——cv_bridge。开发者可以通过该功能包将 ROS 中的图像数据转换成 OpenCV 格式的图像，并且调用 OpenCV 库进行各种图像处理；或者将 OpenCV 处理过后的数据转换成 ROS 图像，通过话题进行发布，实现各节点之间的图像传输。

## 文件目录结构

ROS 机器人图像处理相关功能包位于 jubot_cv 文件夹下，具体路径和文件结构如下。




主要功能包介绍如下表。

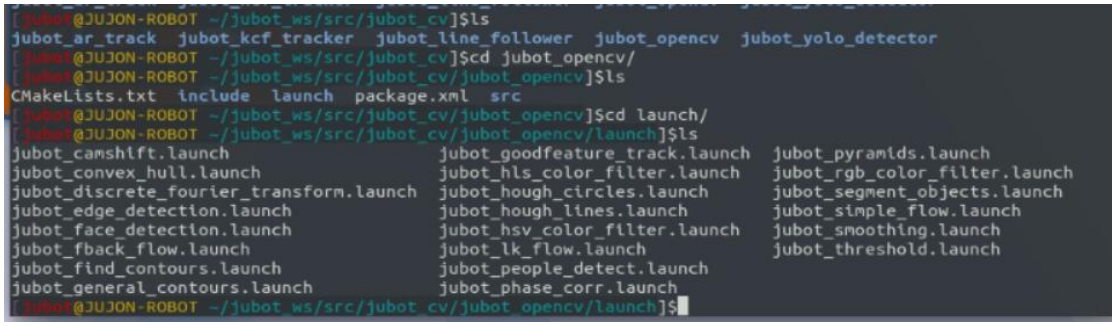
jubot_opencv　OpenCV 基础例程

jubot_line_follower 机器人寻红线行驶

jubot_ar_track　二维码识别跟踪

## OpenCV 图像处理示例

机器人可使用 OpenCV 进行图像处理，例如人脸检测，边缘检测，光流等，具体清单如下图。官方提供较多示例，如下示例我们都经过测试，用户可以运行对应的 launch 文件进行探索。本小节以人脸检测为例进行说明。



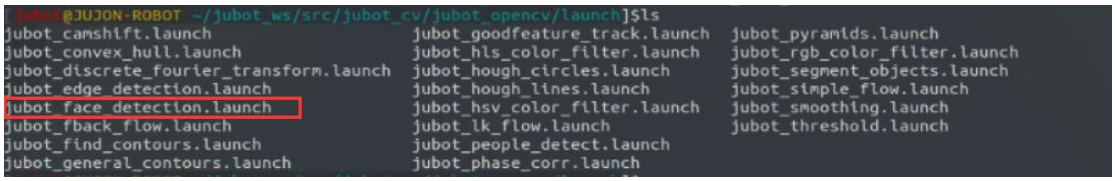

每个文件功能介绍如下:

| |
|---|
| jubot_camshift.launch Camshift 目标追踪算法 |
| jubot_convex_hull.launch Convex Hull 寻找凸多边形算法 |
| jubot_discrete_fourier_transform.launch 离散傅里叶变换算法 |
| jubot_edge_detection.launch 边缘检测算法 |
| jubot_face_detection.launch 人脸检测算法 |
| jubot_fback_flow.launch opencv 光流算法 |
| jubot_find_contours.launch 轮廓检测 |
| jubot_general_contours.launch 一般轮廓检测 |

| |
|---|
| jubot_goodfeature_track.launch 特征点追踪算法 |
| jubot_hls_color_filter.launch HLS 颜色空间滤波 |
| jubot_hough_circles.launch 霍夫圆环检测 |
| jubot_hough_lines.launch 霍夫直线检测 |
| jubot_hsv_color_filter.launch HSV 颜色空间滤波 |
| jubot_lk_flow.launch LK 光流算法 |
| jubot_people_detect.launch 人体检测算法 |
| jubot_phase_corr.launch 相位相关位移检测 |
| jubot_pyramids.launch 图像金字塔采样算法 |
| jubot_rgb_color_filter.launch RGB 颜色空间滤波 |
| jubot_segment_objects.launch 前景物体检测算法 |
| jubot_simple_flow.launch 简单光流算法 |
| jubot_smoothing.launch 图像平滑算法 |
| jubot_threshold.launch 图像阈值滤波算法 |

### 15.4.2 人脸检测例程

首先通过 SSH 命令连接到机器人，进入 opencv 例程文件夹，通过 ls 命令查看例程名称，人脸识别例程 launch 文件为"jubot_face_detection.launch"。




再新建一个终端，SSH 连接到机器人，运行 USB 摄像头节点。

```
roslaunch jubot_driver jubot_camera.launch
```

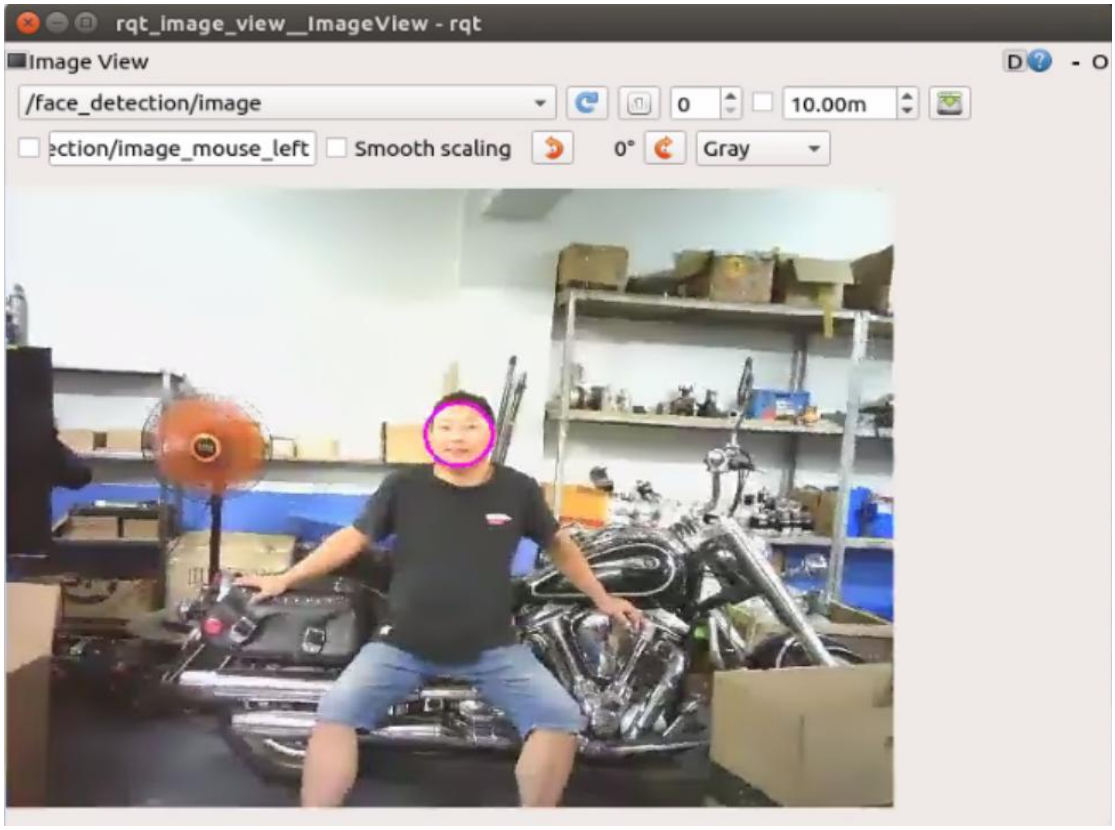
再新建一个终端，SSH 连接到机器人，运行人脸识别例程。

```
roslaunch jubot_opencv jubot_face_detection.launch
```

再新建一个终端，在虚拟机端运行摄像头显示节点。

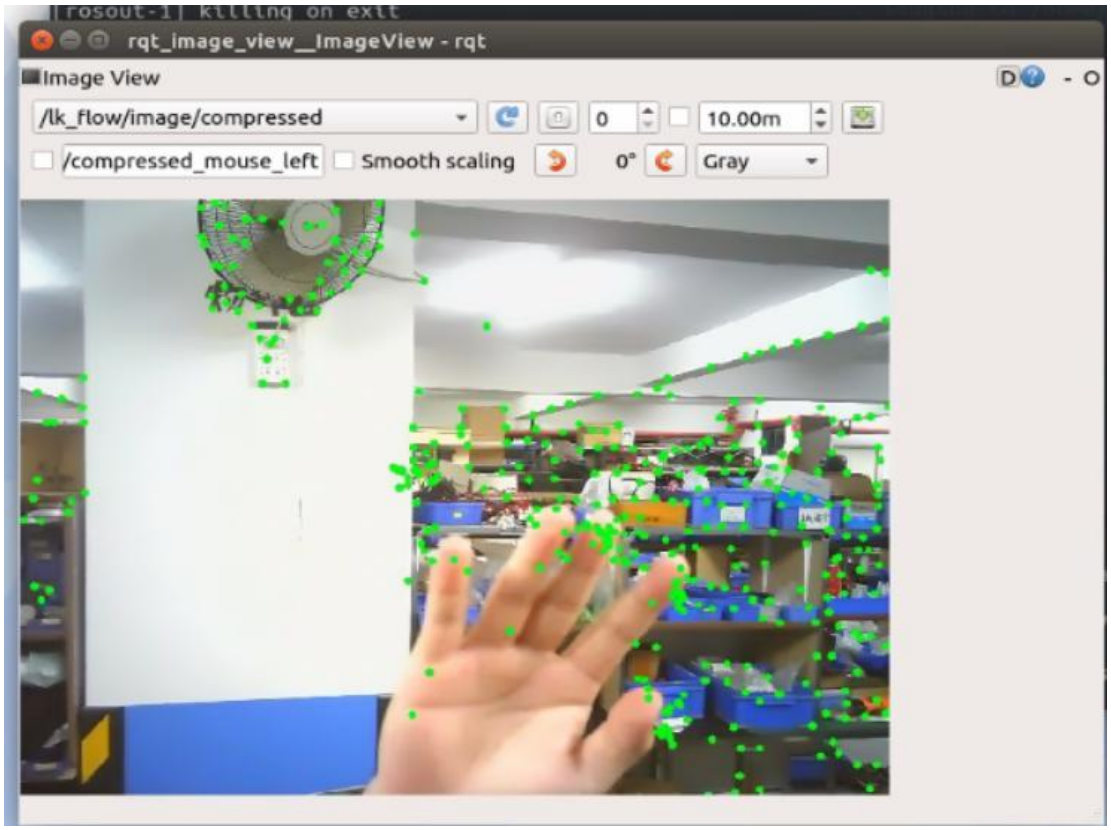
```
rosrun rqt_image_view rqt_image_view
```

选中/face_detection/image，即可看到摄像头画面，画面中的人脸会用粉红色的圆圈圈出。




同样的，我们可以测试其他算法，例如LK光流算法

```
roslaunch jubot_opencv jubot_lk_flow.launch
```




更多例程，大家可以通过运行 opencv 中不同的 launch 文件自行尝试。

源码 wiki: http://wiki.ros.org/opencv_apps

## 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 OpenCV 视觉巡线行驶

## 16.1 实验目的

学习基于ROS 与单目相机的视觉巡线。

## 16.2 实验要求

掌握基于ROS 与 OpenCV 的视觉巡线。

## 16.3 实验工具

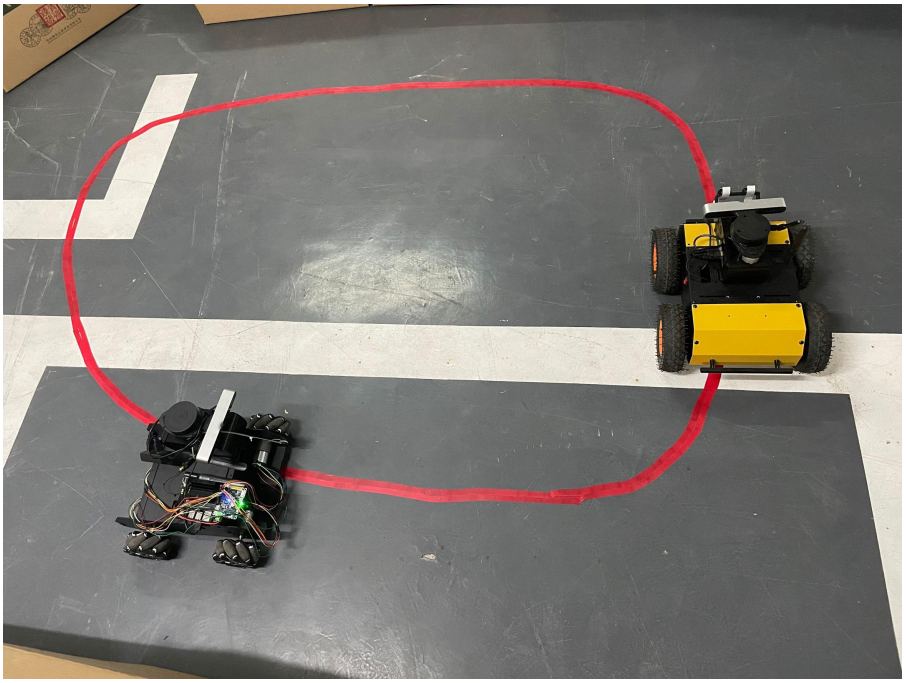
个人电脑一台，路威套件。

## 16.4 实验内容

### 16.4.1 OpenCV 视觉巡线

ROS 机器人实现了基于摄像头的循迹行驶功能，默认为红线，其它颜色线用户可自行修改调试颜色参数。

测试前请先将机器人放置到测试场地，摄像头看到红线，机器人逆时针放置在循迹场地上。

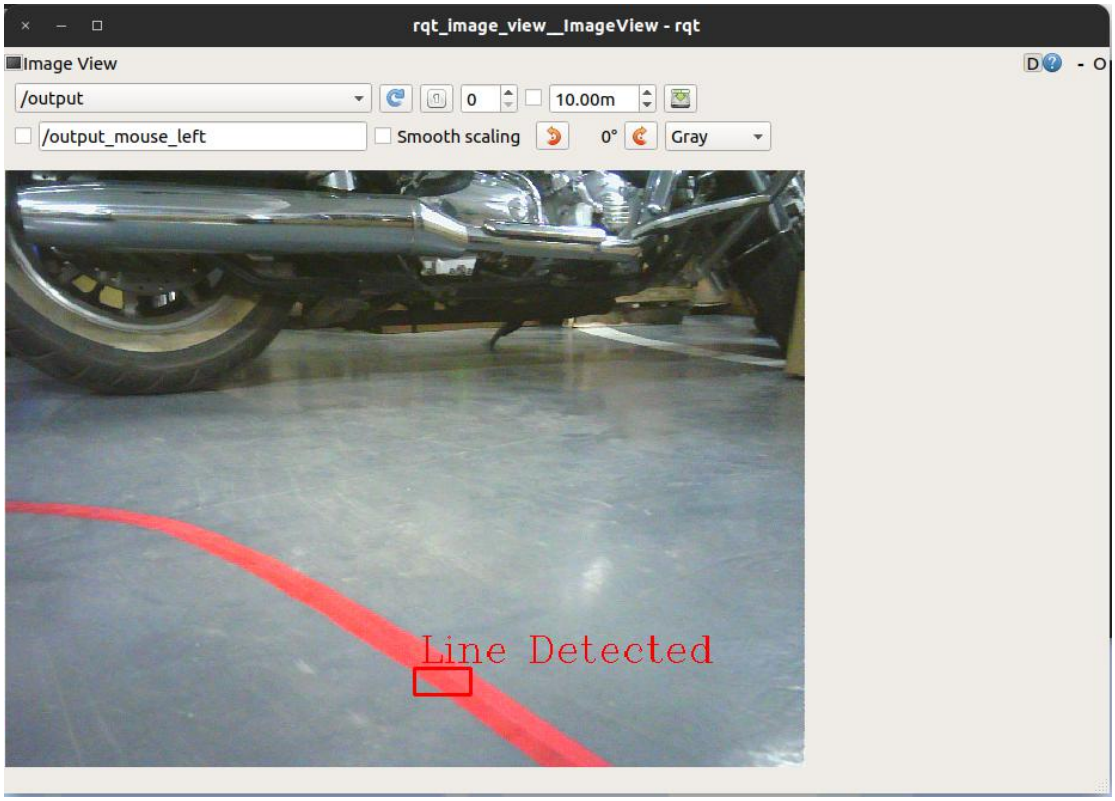


通过 SSH 命令连接到机器人，运行寻线程序，机器人即可沿红线行驶。

```
roslaunch jubot_line_follower jubot_line_follower.launch
```

再新建一个终端，在虚拟机端运行rqt_image_view 软件，选择"/output"话题即可看到机器人摄像头图像，并看到标定的红线图标。

```
rosrun rqt_image_view rqt_image_view
```

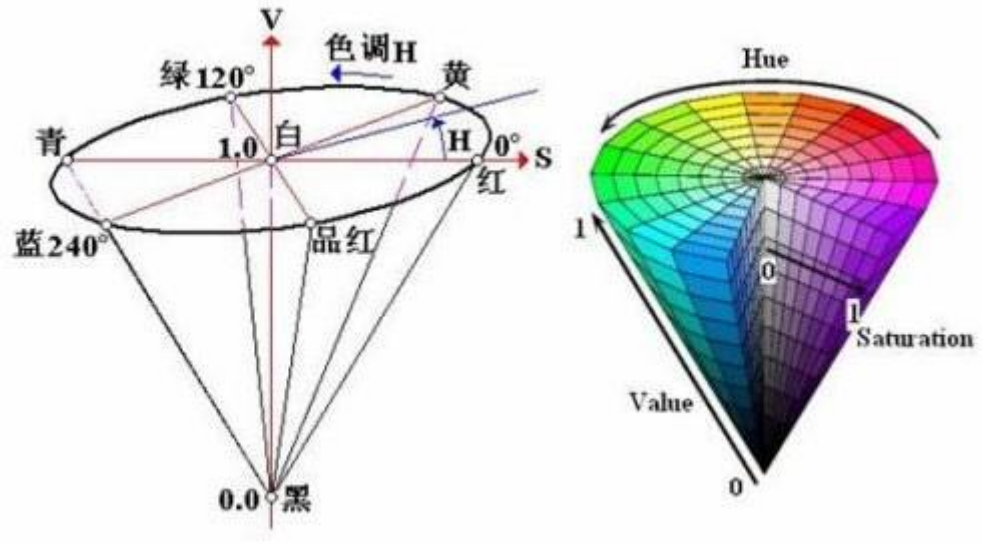


源码路径在机器人端

## 巡线颜色调试

摄像头颜色是基于 HSV 颜色空间设定的，用户可根据自己需要设定巡线颜色。

HSV 是一种将 RGB 色彩空间中的点在倒圆锥体中的表示方法。HSV 即色相(Hue)、饱和度(Saturation)、明度(Value)，又称 HSB(B 即 Brightness)。色相是色彩的基本属性，就是平常说的颜色的名称，如红色、黄色等。饱和度（S）是指色彩的纯度，越高 色彩越纯，低则逐渐变灰，取 0-100%的数值。明度（V），取 0-max(计算机中 HSV 取值范围和存储的长度有关)。HSV 颜色空间可以用一个圆锥空间模型来描述。圆锥的顶点处，V=0，H 和 S 无定义，代表黑色。圆锥的顶面中心处 V=max，S=0，H 无定义，代表白色。




设定巡线颜色时建议将机器人轮胎悬空调试，避免机器人乱跑。设定方法如下。

通过 SSH 命令连接到机器人，运行寻线程序。

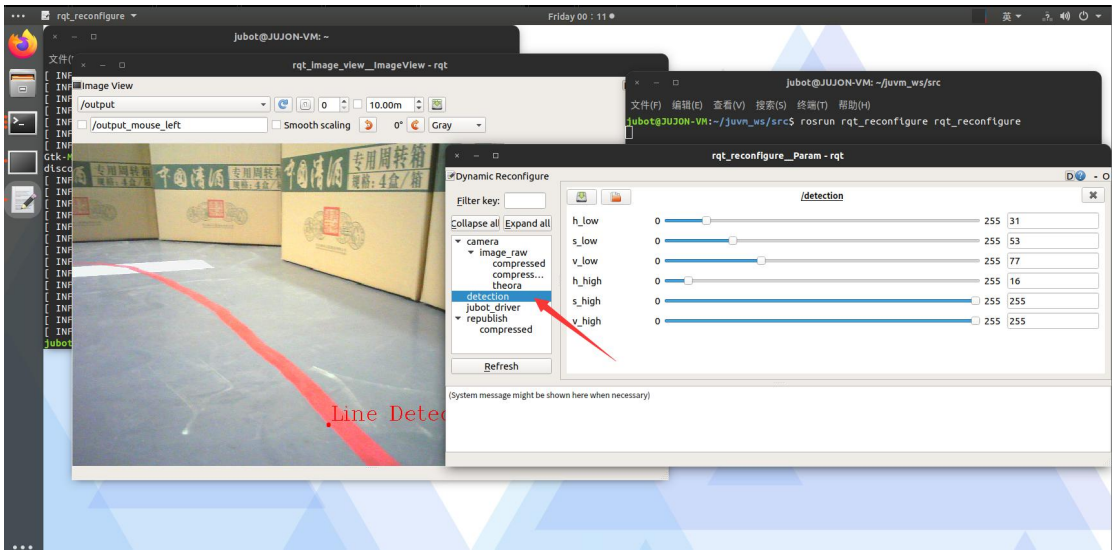
```
roslaunch jubot_line_follower jubot_line_follower.launch
```

再新建一个终端，在虚拟机端运行rqt_image_view 软件，选择"/output"话题即可看到机器人摄像头图像。

```
rosrun rqt_image_view rqt_image_view
```

再新建一个终端，在虚拟机端运 rqt_reconfigure 软件，可动态调整巡线颜色参数。

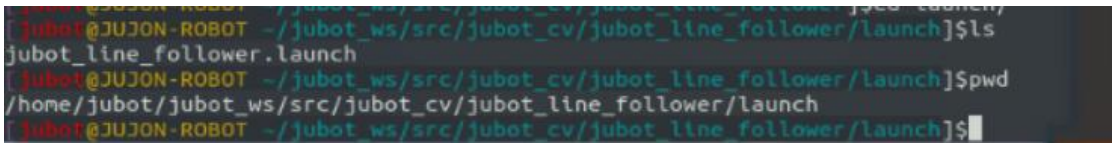
```
rosrun rqt_reconfigure rqt_reconfigure
```

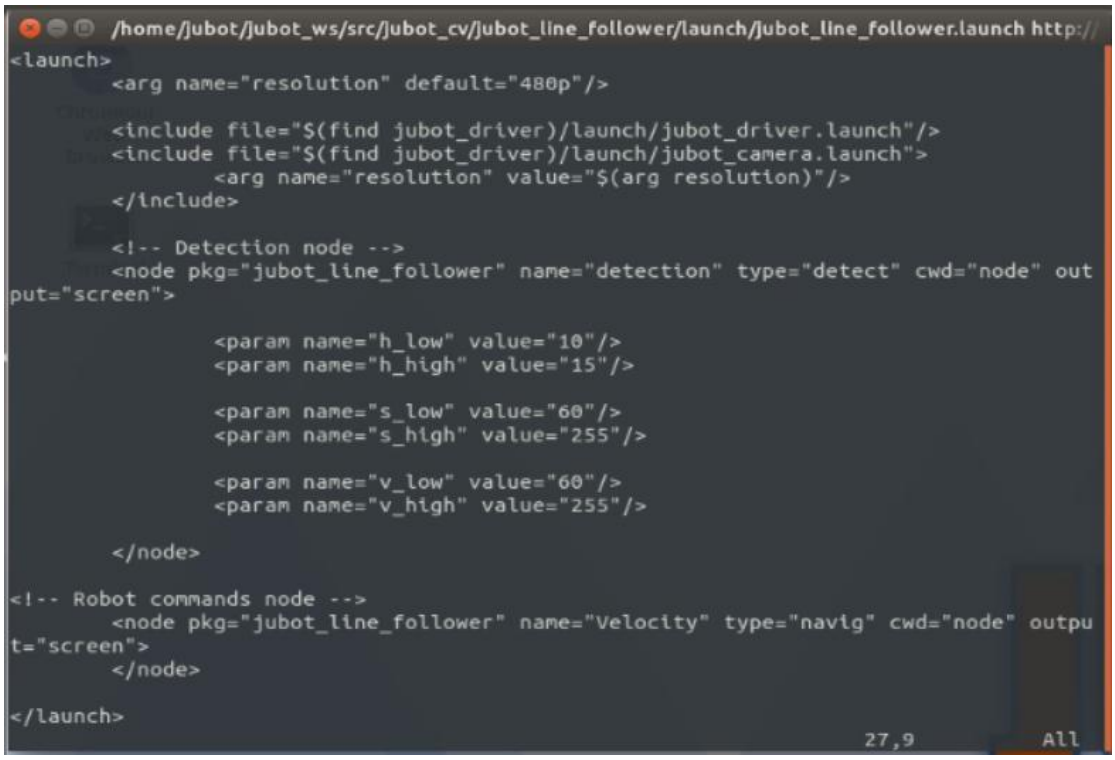



| | 黑 | 灰 | 白 | 红 | | 橙 | 黄 | 绿 | 青 | 蓝 | 紫 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| hmin | 0 | 0 | 0 | 0 | 156 | 11 | 26 | 35 | 78 | 100 | 125 |
| hmax | 180 | 180 | 180 | 10 | 180 | 25 | 34; | 77 | 99 | 124 | 155 |
| smin | 0 | 0 | 0 | 43 | | 43 | 43 | 43 | 43 | 43 | 43 |
| smax | 255 | 43 | 30 | 255 | | 255 | 255 | 255 | 255 | 255 | 255 |
| vmin | 0 | 46 | 221 | 46 | | 46 | 46 | 46 | 46 | 46 | 46 |
| vmax | 46 | 220 | 255 | 255 | | 255 | 255 | 255 | 255 | 255 | 255 |

颜色参数为 HSV 颜色空间，H，S，V 每个参数具有上下两个阈值，通过调节可设定不同的颜色。可以把该颜色的胶带粘在白纸上，放置在机器人前面测试。

调整完成后，可打开 jubot_line_follower.launch 文件修改参数，改为调整后的值，下次启动即为巡线该颜色。






## 16.5 实验结果

能够使用 OpenCV 视觉库实现视觉巡线功能。

## 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 OpenCV 视觉二维码检测

## 17.1 实验目的

学习基于 ROS 与单目相机的人脸识别功能。

## 17.2 实验要求

掌握基于 ROS 与 OpenCV 的视觉二维码检测功能。

## 17.3 实验工具

个人电脑一台，路威套件。

## 17.4 实验内容

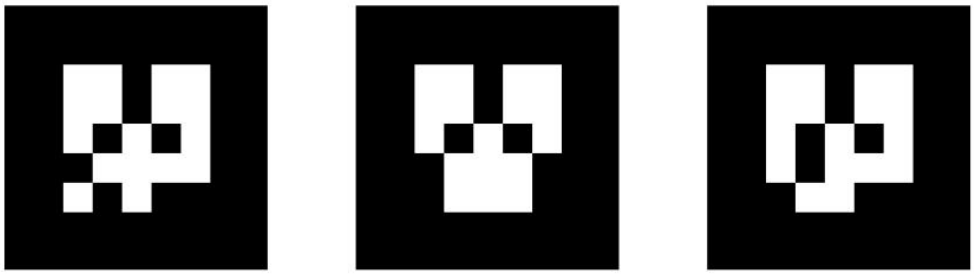
### 17.4.1 OpenCV 视觉二维码实验

二维码技术在生活和工业场景中应用越来越广泛，无论是商场购物还是共享单车，二维码作为一种入口标志已经得到广泛应用。ROS 中提供了多种二维码识别的功能包，本小节介绍二维码识别的功能包 ar_track_alvar。

ar_track_alvar 的主要功能是生成大小，分辨率和数据/ ID 编码不同的 AR 标签，识别并跟踪各个 AR 标签的姿态。识别和跟踪由多个标签组成的"捆绑包"的状态。这样可以实现更稳定的姿态估计，对遮挡的鲁棒性以及对多边物体的跟踪。使用相机图像自动计算捆绑包中标签之间的空间关系，从而用户不必手动测量和输入 XML 文件中的标签位置即可使用捆绑包功能。

功能包具有自适应阈值处理能力，可处理各种照明条件，基于光流的跟踪以实现更稳定的姿态估计，以及一种改进的标签识别方法，该方法不会随着标签数量的增加而显着降低速度。

下图是生成的编码器分别为 1,2,3 的 AR 标签，下面介绍 AR 标签的详细操作流程。

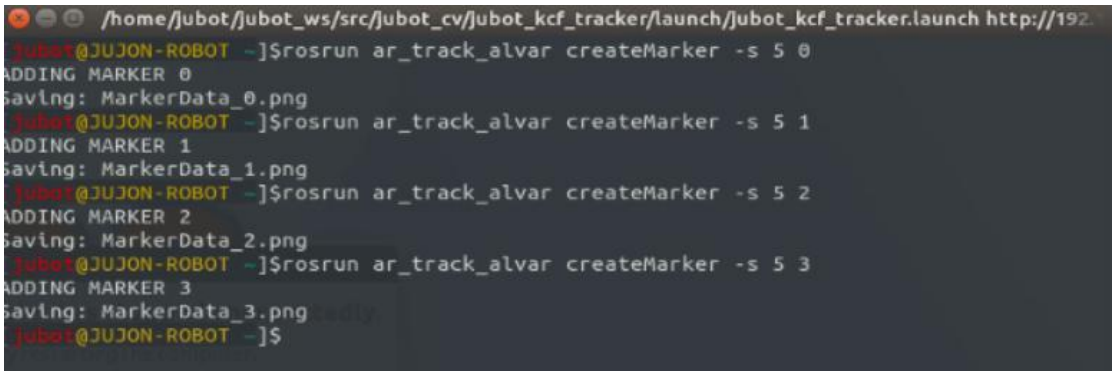


源码地址，http://wiki.ros.org/ar_track_alvar

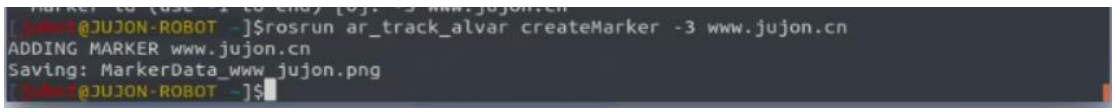
**生成二维码标签**

ar_track_alvar 功能包提供了二维码 AR 标签的生成功能，可以使用如下命令创建标号为 0 的二维码标签，保存在当前目录下。参数 5 是标签的尺寸参数，0 是标签的标号，可以是 0~65535 之间的任意数字。

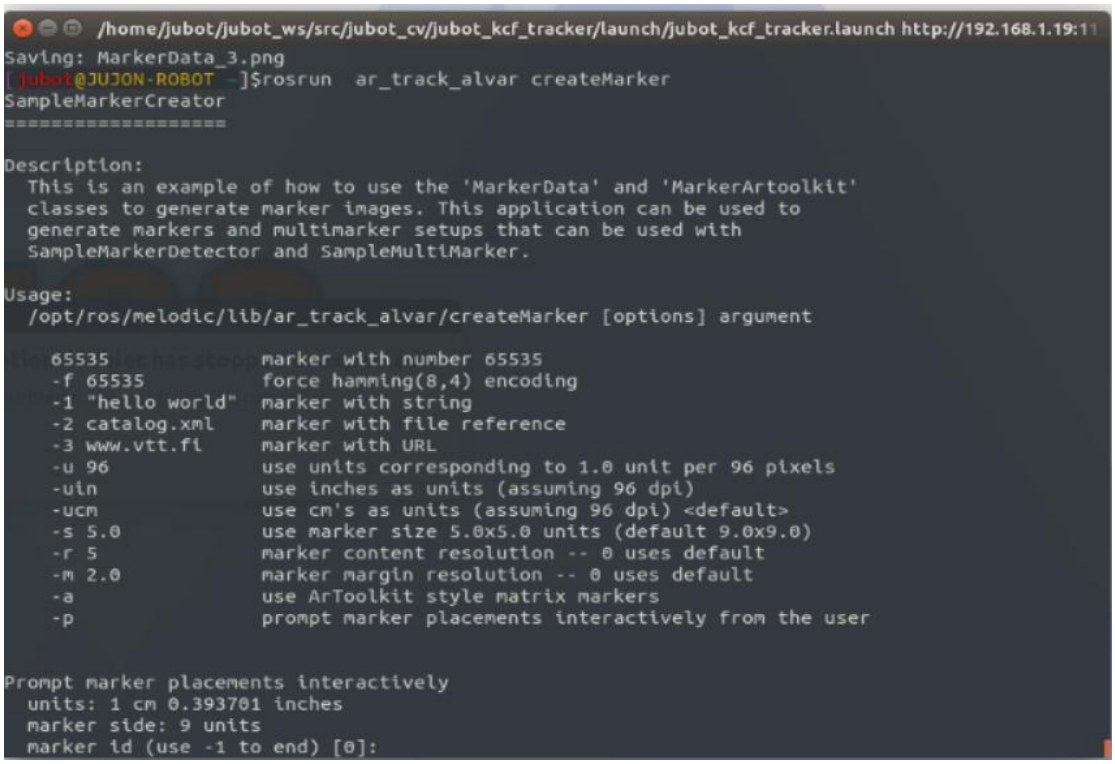
```
rosrun ar_track_alval createMarker -s 5 0
rosrun ar_track_alval createMarker -s 5 1
rosrun ar_track_alval createMarker -s 5 2
rosrun ar_track_alval createMarker -s 5 3
```



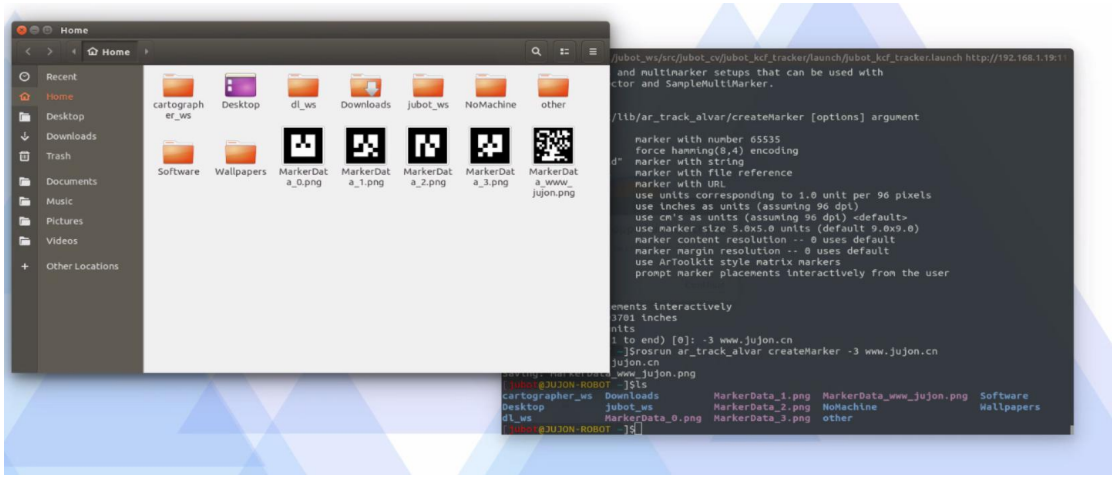

createMarker 工具还有很多参数可以进行配置，使用以下命令即可看到使用帮助，不仅可以使用数字标号生成二维码标签，也可以使用字符串、文件名、网址等，还可以使用-s 参数设置生成二维码的尺寸。




```
rosrun ar_track_alval createMarker
```




用户可以使用上述命令自己生成二维码，也可以使用我们已经生成好的二维码，位于"配套文件/二维码标签"文件夹中找到，可以将二维码打印出来备用。也可以自己生成二维码拿出来使用。

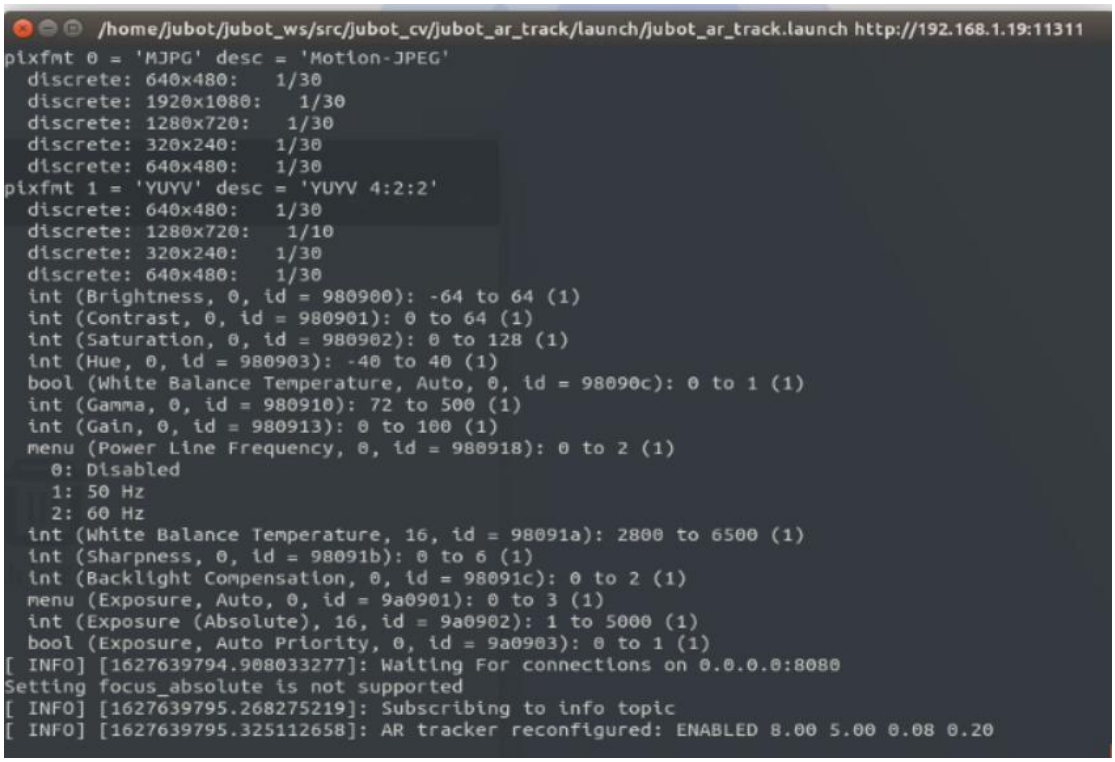



**识别二维码标签**

新建一个终端，SSH 连接到机器人，运行二维码识别程序。

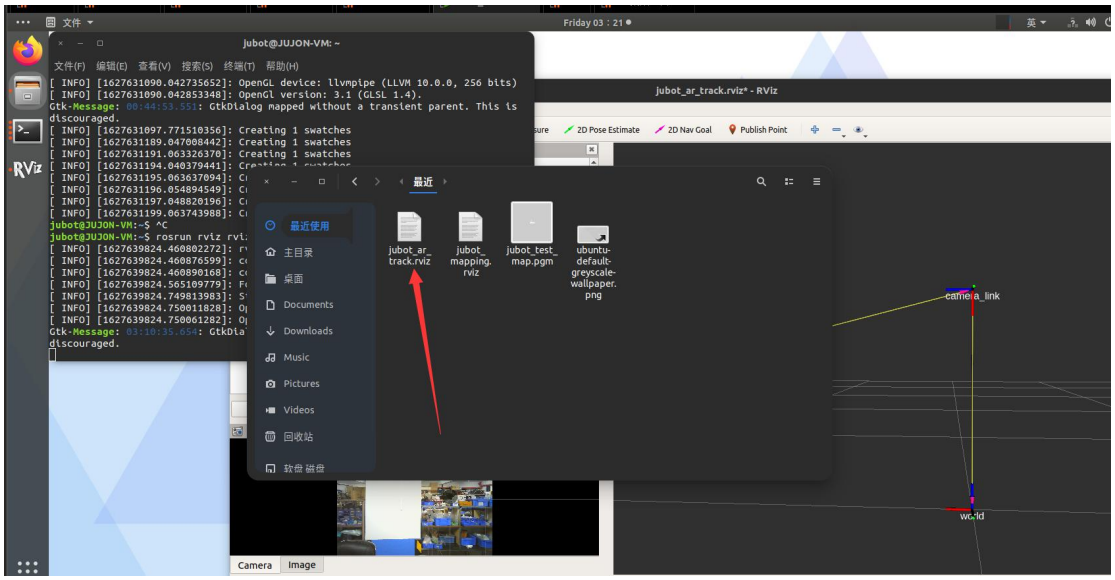
```
roslaunch jubot_ar_track jubot_ar_track.launch
```

等待启动完成，出现如下界面。

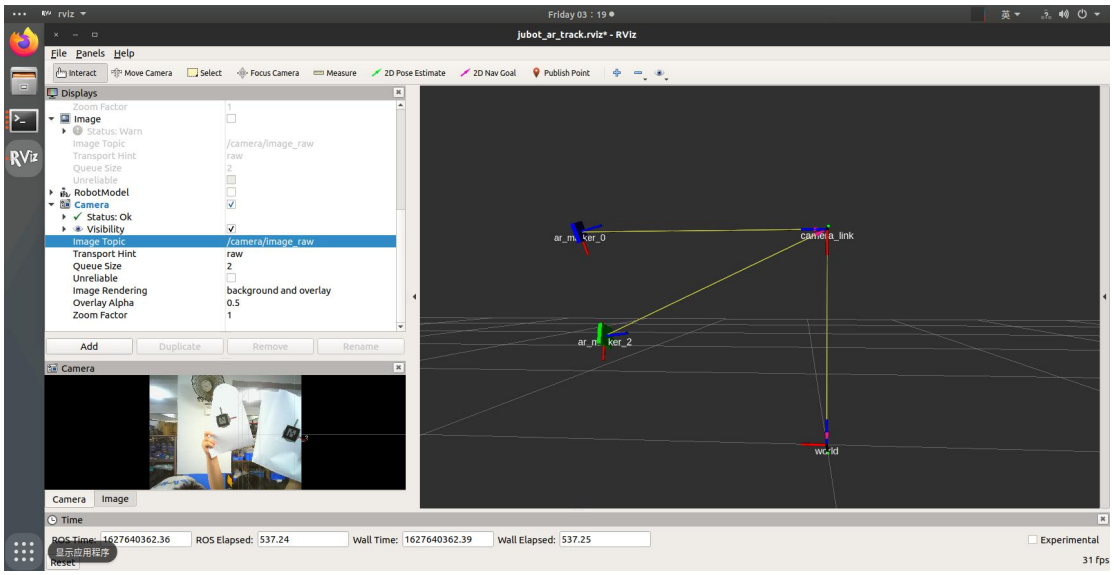



再新建一个终端，在虚拟机端运行 RVIZ 可视化工具。

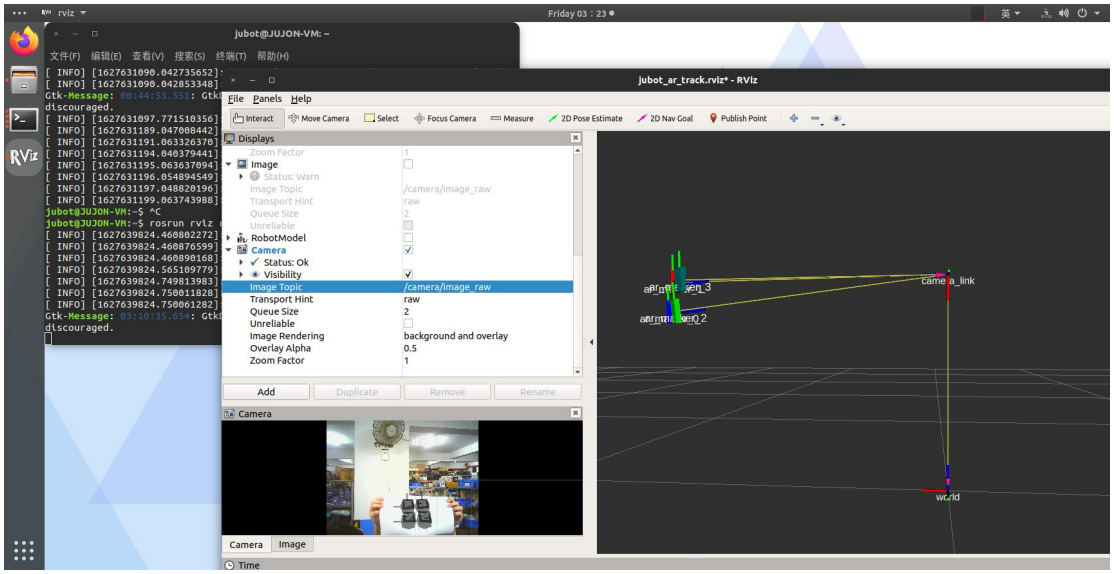
RVIZ 加载二维码识别的配置文件"jubot_ar_track.rviz"。



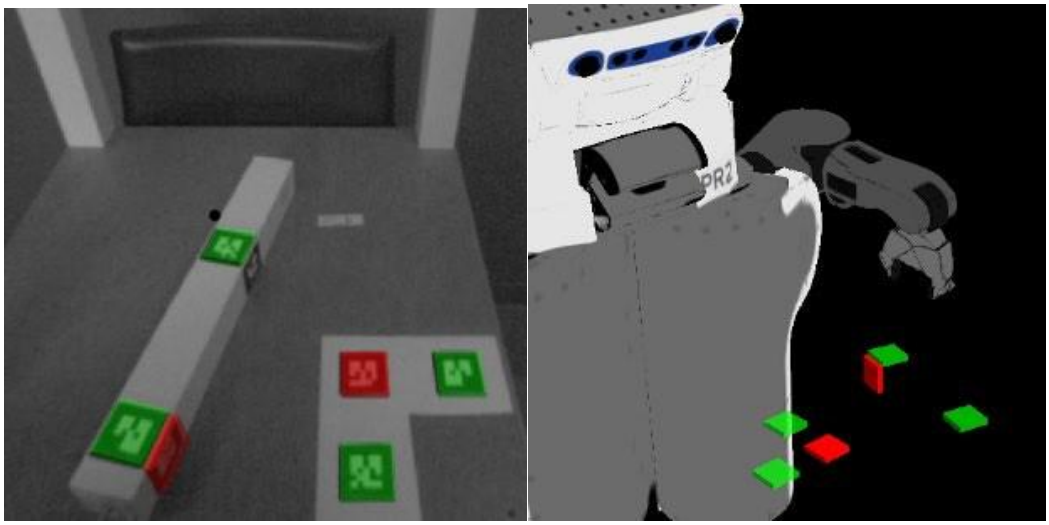

运行成功后可以在打开的 RVIZ 界面中看到摄像头图像，现在将二维码标签放置到摄像头的视野范围内，可以看到下图所示的识别结果。




图像中的二维码上会出现坐标轴，代表识别到的二维码姿态。ar_track_alvar 功能包不仅可以识别图像中的二维码，而且可以确定二维码的空间位姿。在使用摄像头的情况下，因为二维码尺寸已知，所以根据图像变化可以计算二维码的姿态，还可以计算二维码相对摄像头的空间位置。改功能包可以同时识别多个二维码图像。




二维码标签在机器人应用中使用较多，获取这些数据后我们就可以实现进一步的应用了，例如可以实现导航中的二维码定位、引导机器人跟随运动等功能。



## 17.5 实验结果

能够使用 OpenCV 视觉二维码检测。

## 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 基于 KCF 的目标跟踪

## 18.1 实验目的

学习基于ROS 与单目相机的目标跟踪功能。

## 18.2 实验要求

掌握 OpenCV 的 KCF 跟踪算法，卡尔曼滤波算法。

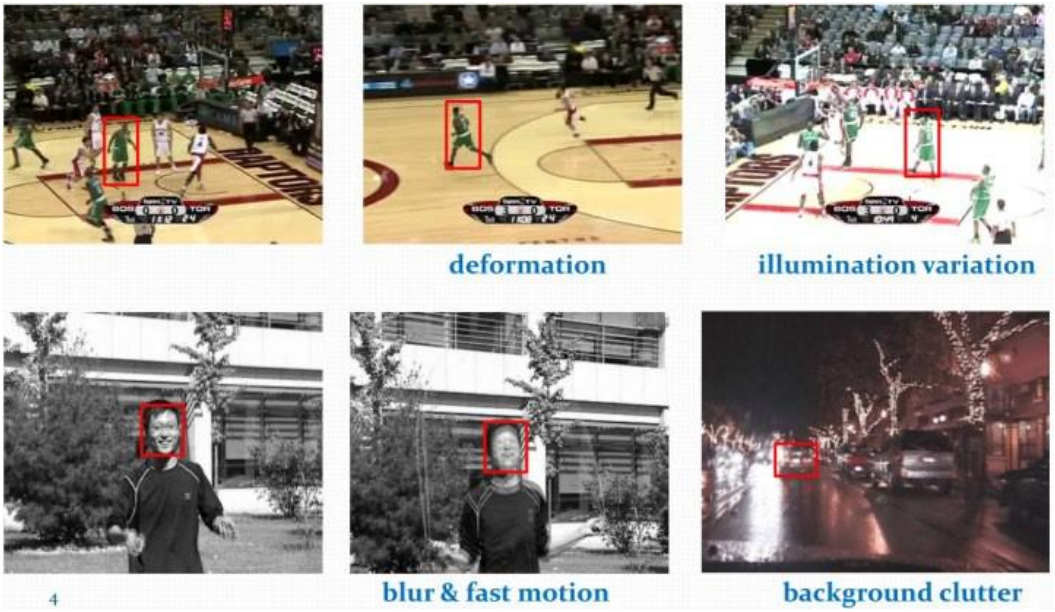
## 18.3 实验工具

个人电脑一台，路威套件。

## 18.4 实验内容

### 18.4.1 KCF 算法简介

目标跟踪大致可分为单目标跟踪与多目标跟踪。目标跟踪解决的问题是：第一帧给出目标矩形框，然后从后续帧开始目标跟踪算法能够跟踪该目标矩形框。通常来 说，目标跟踪有几大难点：外观变形，光照变化，快速运动和运动模糊，背景相似干扰：




平面外旋转，平面内旋转，尺度变化，遮挡和出视野等情况：


对于目标跟踪方法的分类，大致可分为生成模型方法和判别模型方法，目前比较流行的是判别类方法。生成类方法为在当前帧对目标区域建模，下一帧寻找与模型最相似的区域就是预测位置，例如：卡尔曼滤波，粒子滤波，mean-shift 等。判别类算法的经典套路为图像特征+机器学习，当前帧以目标区域为正样本，背景区域为负样 本，机器学习训练分类器，下一帧用训练好的分类器找最优区域，例如：Struck， TLD 等。与生成类方法最大的区别，是分类器训练过程中用到了背景信息，这样分类器专注区分前景和背景，判别类方法普遍都比生成类好。判别类方法的最新发展就是相关滤波类方法和深度学习类方法。相关滤波方法例如：DCF，KCF，ECO 等。深度学习方法例如：MDNet，TCNN，SiamFC 等。

KCF 全称为 Kernel Correlation Filter 核相关滤波算法。是在 2014 年由 Joao F. Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista 提出来的，算法出来之后也算是轰动一时，这个算法不论是在跟踪效果还是跟踪速度上都有十分亮眼的表现，所以引起了一大批的学者对这个算法进行研究以及工业界也在陆续把这个算法应用在实际场景当中。

KCF 是一种判别式跟踪方法，这类方法一般都是在跟踪的过程中训练一个目标检测器，使用目标检测器去检测下一帧预测位置是否是目标，然后再使用新检测结果去更新训练集进而更新目标检测器。而在训练目标检测器时一般选取目标区域为正样本，目标周围区域为负样本。

KCF 的主要贡献可概括为：

- 使用目标周围区域的循环矩阵采集正负样本，利用岭回归训练目标检测器，并成功的利用循环矩阵在傅里叶空间可对角化的性质将矩阵的运算转化为向量的Hadamad 积，即元素的点乘，大大降低了运算量，提高了运算速度，使算法满足实时性要求。
- 将线性空间的岭回归通过核函数映射到非线性空间，在非线性空间通过求解一个对偶问题和某些常见的约束，同样的可以使用循环矩阵傅里叶空间对角化简化计算。
- 给出了一种将多通道数据融入该算法的途径。

KCF 算法相关论文以及官方示例源码地址：

https://www.robots.ox.ac.uk/~joao/circulant/index.html

### 18.4.2 KCF 物体跟踪算法使用

在路威套件中，搭载了基于 KCF 的物体追踪功能 Demo，程序包源码位于机器人端 ~/jubot_ws/src/jubot_cv/jubot_kcf_tracker/ 路径下。

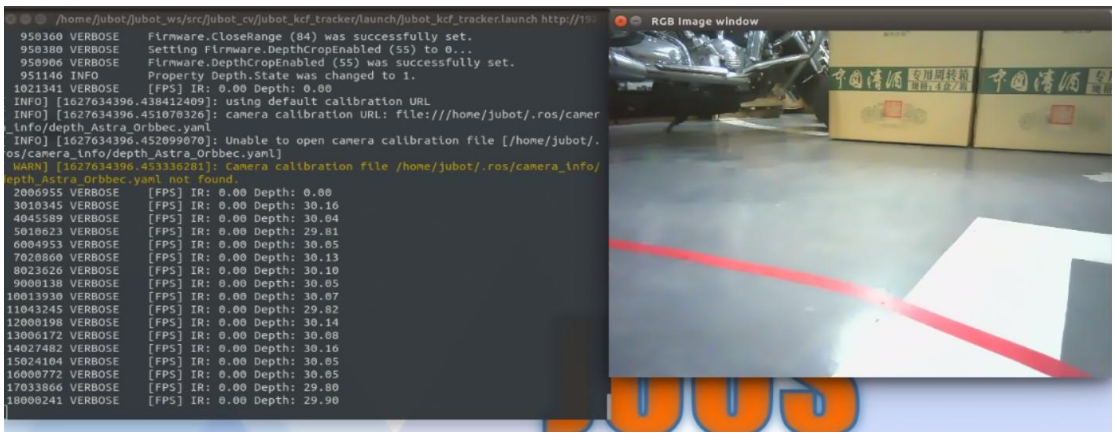

因为 KCF 算法会打开显示窗口用于跟踪目标框的拖选，所以，需要通过 NoMachine 远程桌面连接机器人运行 KCF 算法；如通过 SSH 方法连接启动 KCF 算法，会出现无法打开可视化窗口的错误。

所以，首先，通过 NoMachine 远程桌面连接机器人，并打开终端：


输入以下命令，启动 KCF 跟踪算法：

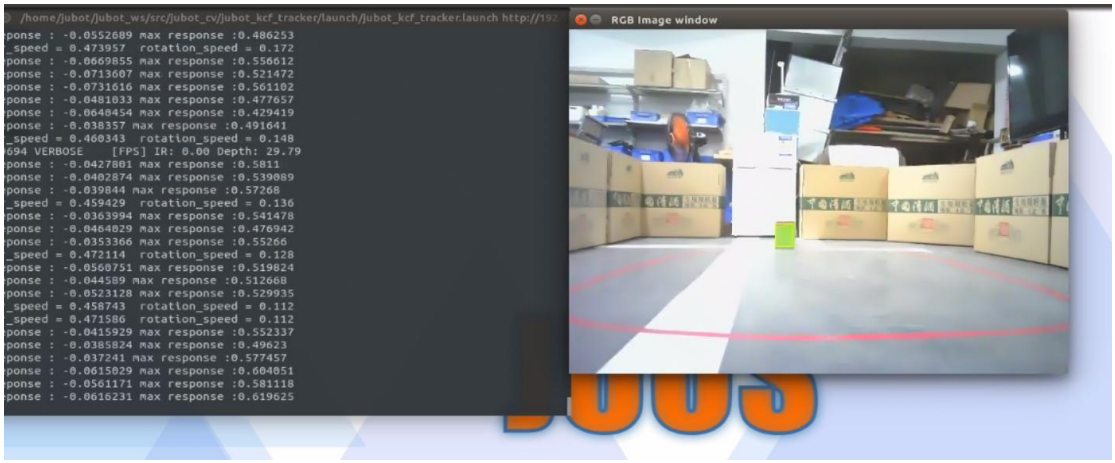
```
roslaunch jubot_kcf_tracker jubot_kcf_tracker.launch
```

启动成功后，会出现一摄像头画面：

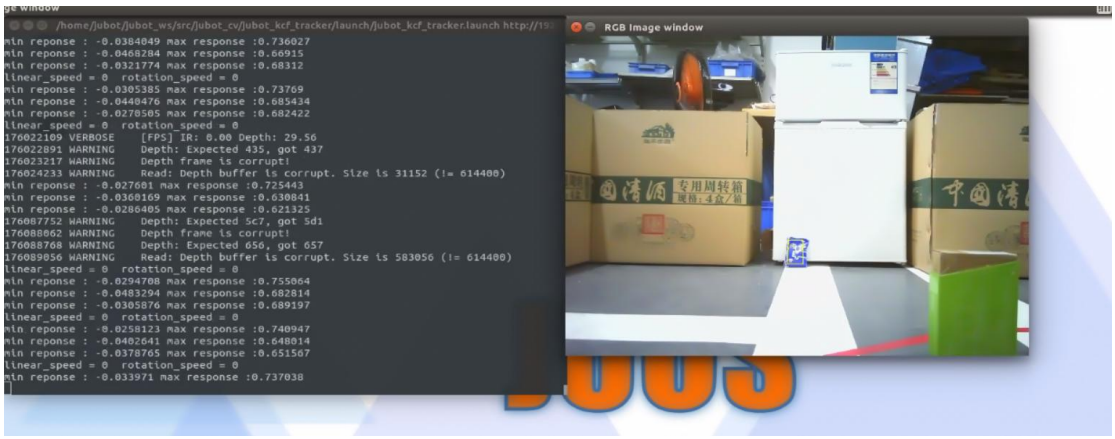



在此画面中，可以用鼠标框选想要跟踪的物体，KCF 算法可以跟踪任意物体，特征越明显，物体表面越平整，跟踪效果越好，例如：

跟踪物体A：




跟踪物体B：




选中要跟踪的目标之后，机器人将会利用深度相机来获取物体离机器人的距离， 并利用KCF 算法使物体保持在画面正中心，机器人将会跟物体保持 1.5m 左右距离跟踪物体运动。

**无深度相机的机器人套装，机器人不会跟随物体运动。**

**KCF 跟踪可以跟随物体前进与转弯，无法后退。**

## 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

# 第十九章 VSLAM 预备知识

## 19.1 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。

## 19.2 实验要求

理解深度相机与VSLAM的基本定义。

## 19.3 实验工具

个人电脑一台，路威套件。

## 19.4 实验内容

### 19.4.1 深度相机定义

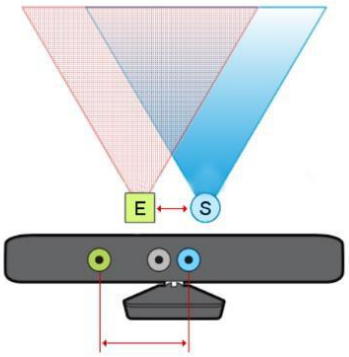
随着机器视觉，自动驾驶等颠覆性的技术逐步发展，采用 3D 相机进行物体识别，行为识别，场景 建模的相关应用越来越多，可以说深度相机就是终端和机器人的眼睛，那么什么是深度相机呢，跟之前的普通相机（2D）想比较，又有哪些差别？ 深度相机又称之为 3D 相机，顾名思义，就是通过该相机能检测出拍摄空间的景深距离，这也是与普通摄像头最大的区别。

普通的彩色相机拍摄到的图片能看到相机视角内的所有物体并记录下来，但是其所记录的数据不包含这些物体距离相机的距离。仅仅能通过图像的语义分析来判断哪些物体离我们比较远，哪些比较近，但是并没有确切的数据。而 深度相机则恰恰解决了该问题，通过深度相机获取到的数据，我们能准确知道图像中每个点离摄像头距离，这样加上该点在 2D 图像中的(x,y)坐标，就能获取图像中每 个点的三维空间坐标。通过三维坐标就能还原真实场景，实现场景建模等应用。

## 19.4.2 深度相机分类

### 结构光深度相机

结构光（Structured-light），其基本原理是，通过近红外激光器，将具有一定结构特征的光线投射到被拍摄物体上，再由专门的红外摄像头进行采集。这种具备一定结构的光线，会因被摄物体的不同深度区域反射，而采集不同的图像相位信息，然后通过运算单元将这种结构的变化换算成深度信息，以此来获得三维结构。简单来说就是，通过光学手段获取被拍摄物体的三维结构，再将获取到的信息进行更深入的计算。通常采用特定波长的不可见的红外激光作为光源，它发射出来的光经过 一定的编码投影在物体上，通过一定算法来计算返回的编码图案的畸变来得到物体的位置和深度信息。根据编码图案不同一般有：条纹结构光，代表传感器 enshape ； 编码结构光，代表传感器 Mantis Vision，Realsense(F200);散斑结构光 ，代表传感器奥比中光，微软 Kinect，英特尔 RealSense 等。




结构光深度相机原理示意图（注意E端发射的带图案的光源）

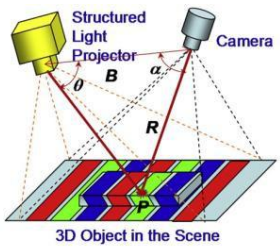



Illustration of structured light.

**结构光的优点：**

方案成熟，相机基线可以做的比较小，方便小型化。

资源消耗较低，单帧 IR 图就可计算出深度图，功耗低。

主动光源，夜晚也可使用。

在一定范围内精度高，分辨率高，分辨率可达 1280x1024,帧率可达60FPS。

**结构光的缺点：**

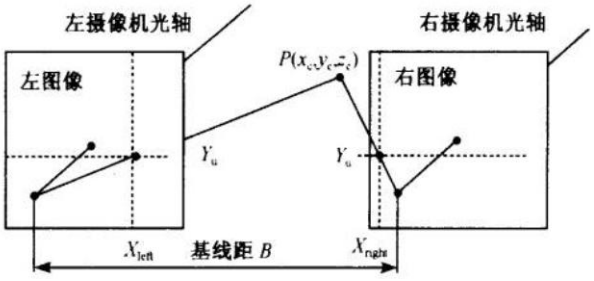
容易受环境光干扰，室外体验差。

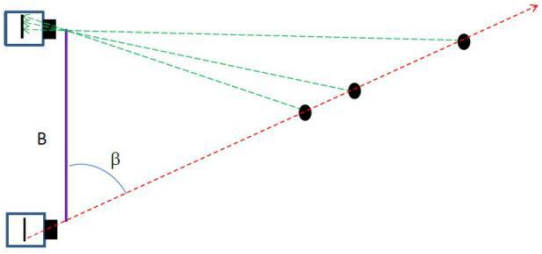
随检测距离增加，精度会变差。

## 双目视觉深度相机

双目立体视觉(Binocular Stereo Vision)是机器视觉的一种重要形式，他是基于视差原理并利用成像设备 从不同的位置获取被测物体的两幅图像，通过计算图像对应点间的位置偏差，来获取物体三维几何信息的 方法。

当然完整的双目深度计算非常复杂，主要涉及到左右相机的特征匹配，计算会非常消耗资源。




**双目相机的优点：**

硬件要求低，成本也低。普通 CMOS 相机即可。

室内外都适用。只要光线合适，不要太昏暗。

**双目相机缺点：**

对环境光照非常敏感。光线变化导致图像偏差大，进而会导致匹配失败或精度低。

不适用单调缺乏纹理的场景。双目视觉根据视觉特征进行图像匹配，没有特征会导致匹配失败。

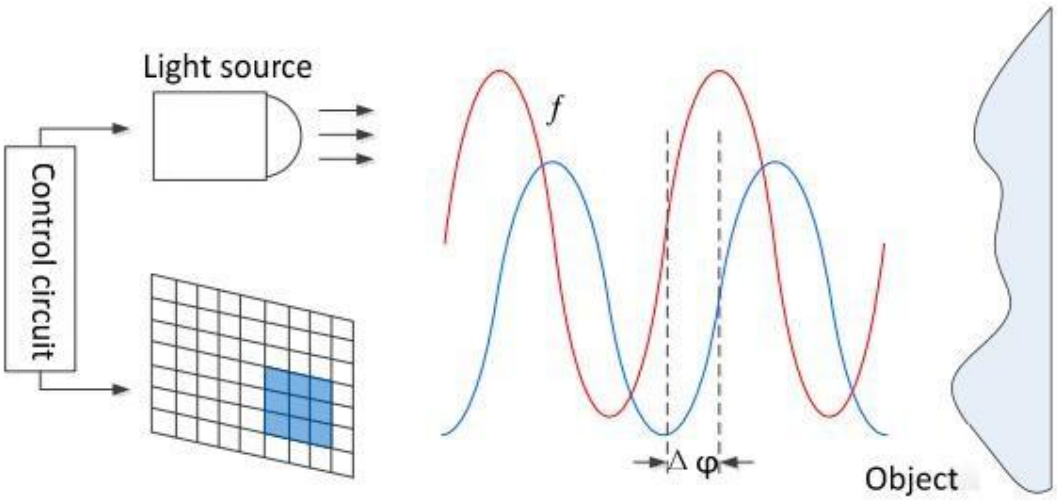
计算复杂度高。该方法是纯视觉的方法，对算法要求高，计算量较大。

基线限制了测量范围。测量范围和基线（两个摄像头间距）成正比，导致无法小型化。

## TOF 深度相机

TOF（Time Of Flight），顾名思义是测量光飞行时间来取得距离，具体而言就是通过给目标连续发射激光脉冲，然后用传感器接收从反射光线，通过探测光脉冲的飞行往返时间来得到确切的目标物距离。因为光速非常快，通过直接测光飞行时间实际不可行，一般通过检测一定手段调制后的光波的相位偏移来实现。TOF 法根据调制方法的不同，一般可以分为两种：脉冲调制（Pulsed Modulation）和连续波调制（Continuous Wave Modulation）。脉冲调制需要非常高精度时钟进行测量，且需要发出高频高强度激光，目前大多采用检测相位偏移办法来实现 TOF 功能。简单来说就是，发出一道经过处理的光，碰到物体以后会反射回来，捕捉来回的时间，因为已知光速和调制光的波长，所以能快速准确计算出到物体的距离。

因为 TOF 并非基于特征匹配，这样在测试距离变远时，精度也不会下降很快，目前无人驾驶以及一些高端的消费类 Lidar 基本都是采用该方法来实现。




**TOF 的优点主要有：**

检测距离远。在激光能量够的情况下可达几十米。

受环境光干扰比较小。

**TOF 的缺点主要有：**

对设备要求高，特别是时间测量模块，资源消耗大。

该方案在检测相位偏移时需要多次采样积分，运算量大，边缘精度低。

限于资源消耗和滤波，帧率和分辨率都没办法做到较高。

### 19.4.3 路威套件深度相机参数

路威套件上搭载的深度相机是基于奥比中光 Astra Pro 方案的定制版相机，该相机具备 1080P RGB 普通相机功能和基于结构光原理深度相机功能，并且具有双立体声麦克风。配备高端 ISP 芯片，可自动根据环境光调节快门优化图像，非常适合机器人视觉图像处理。

具体参数可参考下表：

| 内容 | 参数 |
| --- | --- |
| RGB 像素 | 1080P |
| 深度分辨率 | 640*480/320*240 |
| 深度最大帧率 | 30FPS |
| 视频分辨率 | 1280*720 |
| 视频最大帧率 | 30FPS |
| 麦克风 | 双立体麦克风 |
| 视场角（FOV） | H 58.4° x V 45.7° |
| 精度 | 1m: ±3mm |
| 工作范围 | 0.6-8m |
| 功耗 | <2.5W |
| 工作温度 | 10℃-40℃ |

### 19.4.4 路威套件深度相机文件目录说明

在 jubot_nav_depthcamera 软件扩展包 launch 文件夹中，主要包含了以下 launch 启动文件。

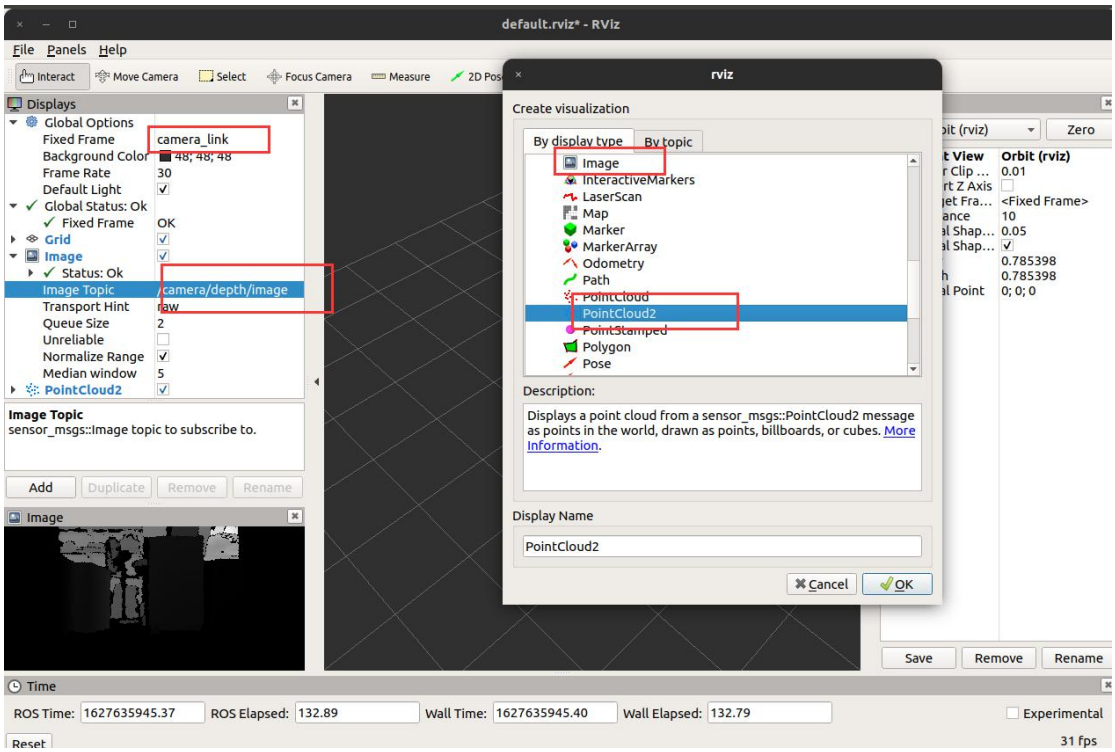
| 文件名称 | 说明 |
| --- | --- |
| jubot_ORBSLAM.launch | ORBSLAM 启动文件。 |
| jubot_ORBSLAM2.launch | ORBSLAM2 算法启动文件。 |
| jubot_RTABSLAM_Mapping.launch | RTAB-SLAM 深度相机建图启动文件 |
| jubot_RTABSLAM_Mapping_UseLidar.launch | RTAB-SLAM 深度相机融合激光雷达建图启动文件 |
| jubot_RTABSLAM_Navigation.launch | RTAB-SLAM 深度相机导航启动文件 |
| jubot_RTAB_SLAM_Navigation_UseLidar.launch | RTAB-SLAM 深度相机融合激光雷达导航启动文件 |
| jubot_mapping_frontier.launch | 深度相机模拟激光雷达自动探索建图启动文件 |
| jubot_mapping_gmapping.launch | 深度相机模拟激光雷达 GMapping 算法建图启动文件 |
| jubot_mapping_karto.launch | 深度相机模拟激光雷达 Karto 算法建图启动文件 |
| jubot_nav.launch | 深度相机模拟激光雷达导航算法启动文件 |
| jubot_slam.launch | 建图算法入口文件 |

## 查看点云图

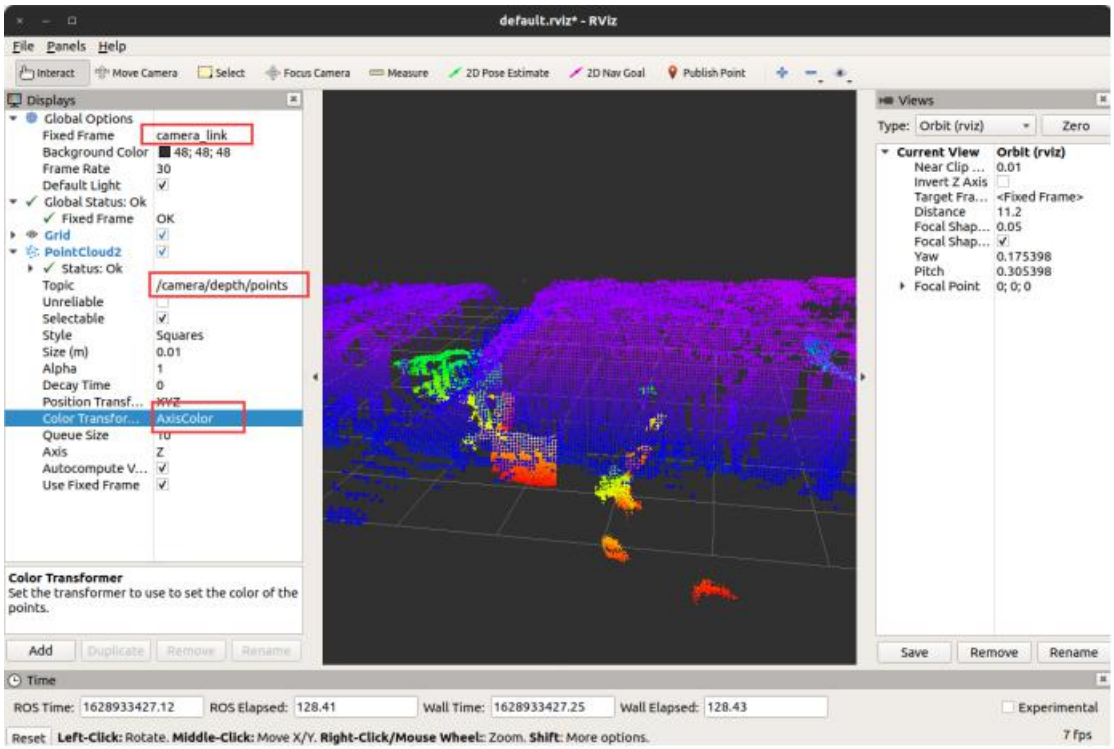
在机器人端执行如下命令，运行深度相机驱动。

```
roslaunch jubot_nav_depthcamera jubot_depthcamera.launch
```

在虚拟机端打开一个终端，运行 `rosrun rviz rviz` ,打开rviz 显示工具。




此时可以看到点云图像显示。




## 查看深度图像

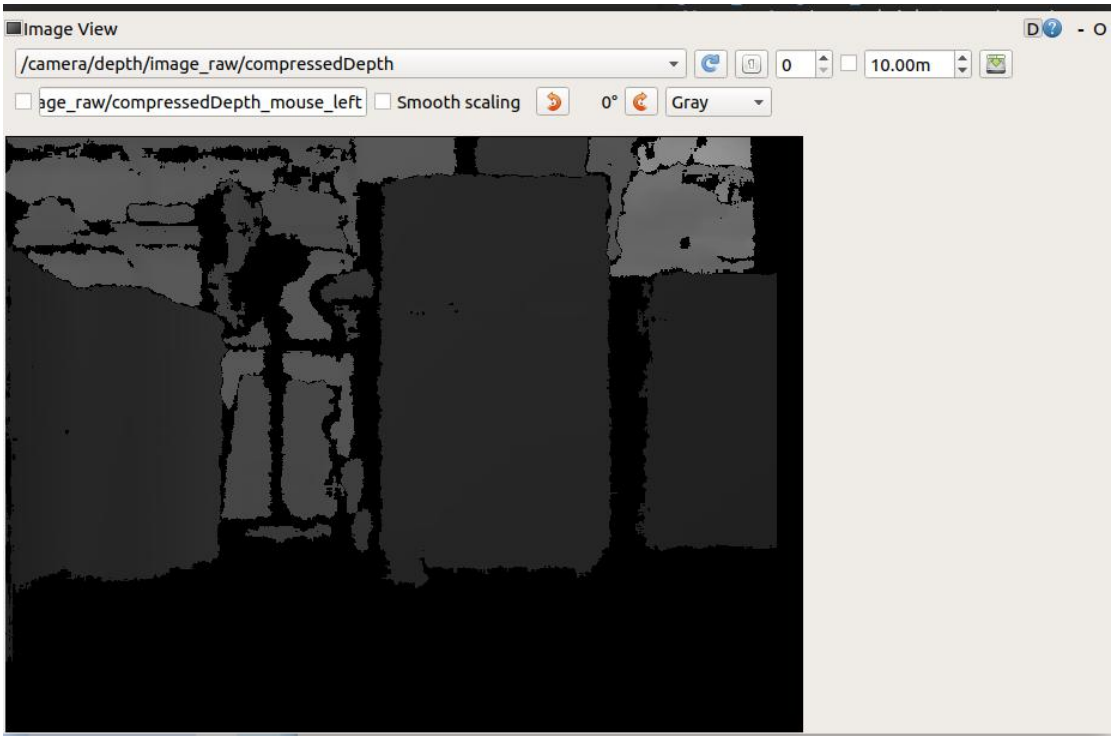
在机器人端执行如下命令，运行深度相机驱动。

```
roslaunch jubot_nav_depthcamera jubot_depthcamera.launch
```

在虚拟机端新建一个终端，运行

```
rosrun rqt_image_view rqt_image_view
```

此时，即可显示深度相机的深度图像。




**深度相机模拟激光雷达**

使用深度相机模拟激光雷达，进行建图导航操作。

通过获取深度相机单行扫描数据作为二位平面扫描数据使用。

具体使用方法如下。

**VSLAM 建图**

在机器人端执行如下命令，运行建图算法。

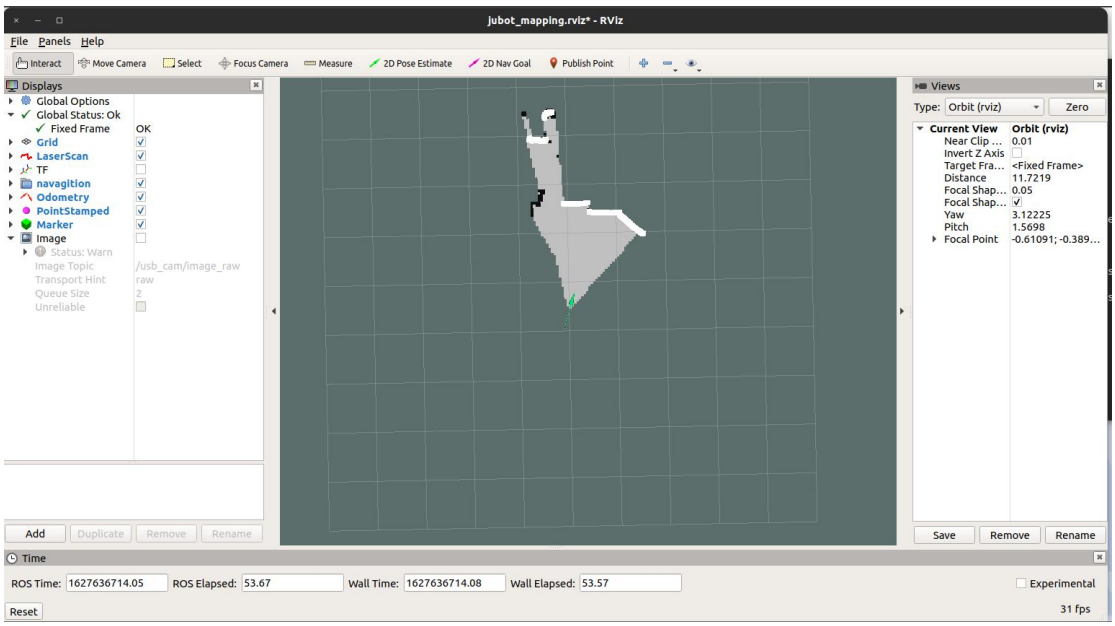
slam_methods 参数为建图算法参数，可选 gmapping,karto

其余操作与机器人使用手册中的建图章节完全一致。

gmapping建图：

```
roslaunch jubot_nav_depthcamera jubot_slam.launch slam_methods:=gmapping
```

在虚拟机端运行Rviz可视化工具：`rosrun rviz rviz` 同雷达建图所示，在 Rviz 下查看建图效果：

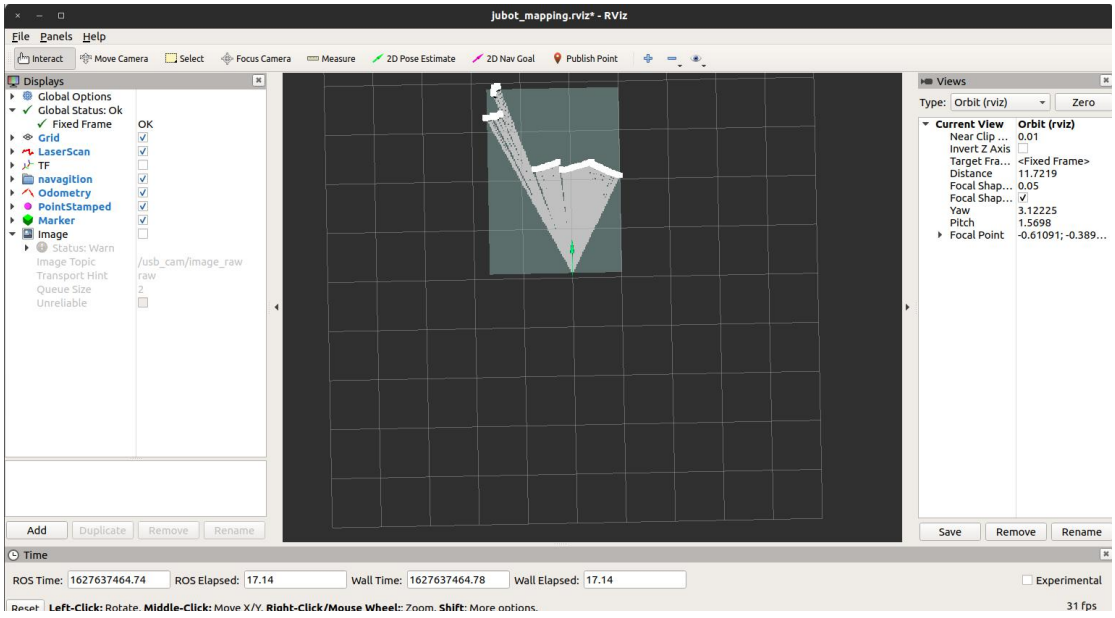



启动键盘或者手柄控制，控制机器人移动完成建图。得到满意的地图后，在虚拟机端内通过以下指令：`rosrun map_server map_saver -f jubot_vslam_gmapping` 保存地图。

karto建图：

`roslaunch jubot_nav_depthcamera jubot_slam.launch slam_methods:=gmapping`

在虚拟机端运行Rviz可视化工具：`rosrun rviz rviz` 同雷达建图所示，在 Rviz 下查看建图效果：




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，在虚拟机端内通过以下指令：`rosrun map_server map_saver -f jubot_vslam_karto` 保存地图。

**VSLAM 导航**

在机器人端执行如下命令，运行建图算法。

```
roslaunch jubot_nav_depthcamera jubot_nav.launch
```

其余操作与机器人使用手册中的导航章节完全一致。请读者自行尝试。

## 19.5 实验结果

能够理解多种深度相机的原理，能够在机器人端获取到点云与深度图像信息。

## 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 ORB-SLAM & ORB-SLAM2 视觉 SLAM 算法

## 20.1 实验目的

可以在 ROS 系统中依靠ORB-SLAM & ORB-SLAM2 算法实现VSLAM建图导航。

## 20.2 实验要求

理解掌握 ORB-SLAM & ORB-SLAM2 的基本使用。

## 20.3 实验工具

个人电脑一台，路威套件。

## 20.4 实验内容

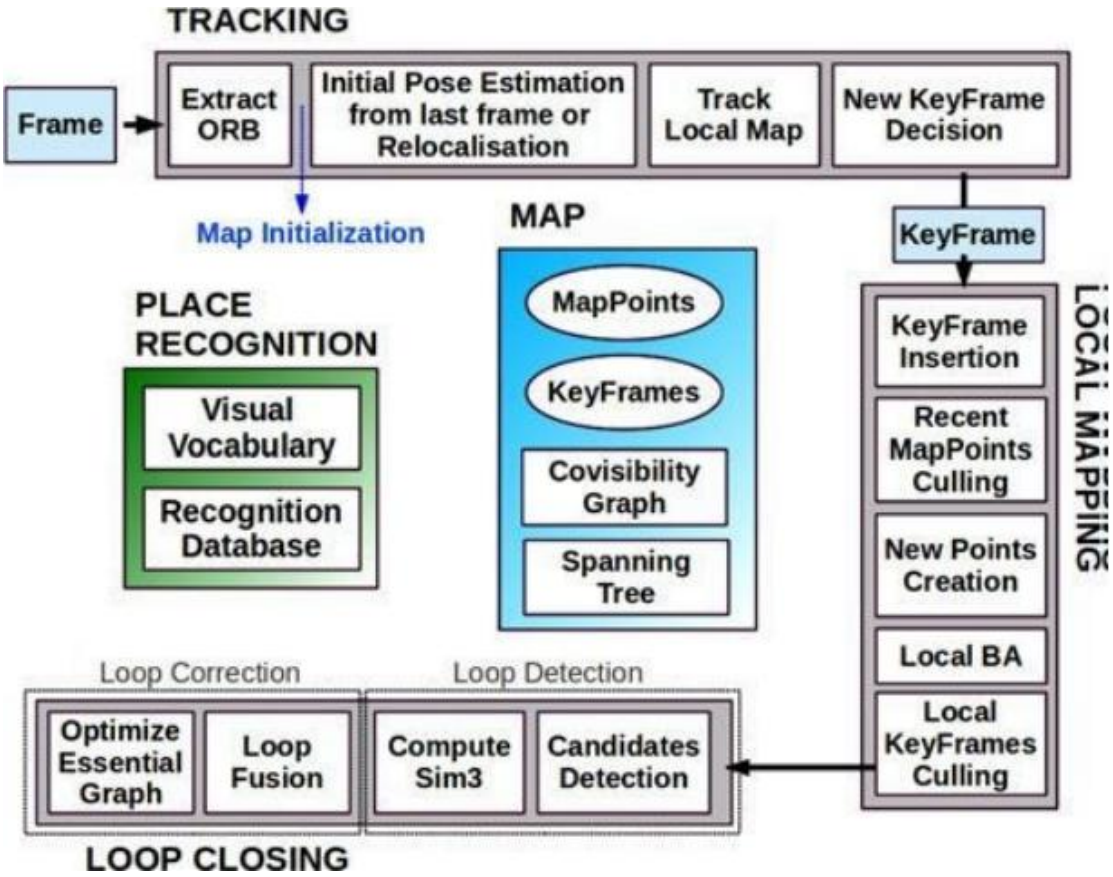
### 20.4.1 ORB-SLAM简介

ORB-SLAM 是西班牙 Zaragoza 大学的 Raúl Mur-Arta 编写的视觉 SLAM 系统。它是一个完整的 SLAM 系统，包括视觉里程计、跟踪、回环检测，是一种完全基于稀疏特征点的单目 SLAM 系统，同时还有单目、双目、RGBD 相机的接口。其核心是使用 ORB (Orinted FAST and BRIEF) 作为整个视觉 SLAM 中的核心特征。

ORB-SLAM 基本延续了 PTAM 的算法框架,但对框架中的大部分组件都做了改进,归纳起来主要有 4 点:

- ORB-SLAM 选用了 ORB 特征，基于 ORB 描述量的特征匹配和重定位，都比PTAM 具有更好的视角不变性。此外，新增三维点的特征匹配效率更高，因此能更及时地扩展场景。扩展场景及时与否决定了后续帧是否能稳定跟踪。
- ORBSLAM 加入了循环回路的检测和闭合机制，以消除误差累积。系统采用与重定位相同的方法来检测回路(匹配回路两侧关键帧上的公共点)，通过方位图 (Pose Graph) 优化来闭合回路。
- PTAM 需要用户指定 2 帧来初始化系统，2 帧间既要有足够的公共点，又要有足够的平移量. 平移运动为这些公共点提供视差 (Parallax)，只有足够的视差才能三角化出精确的三维位置。ORB-SLAM 通过检测视差来自动选择初始化的 2 帧。
- PTAM 扩展场景时也要求新加入的关键帧提供足够的视差，导致场景往往难以扩展. ORB-SLAM 采用一种更鲁棒的关键帧和三维点的选择机制——先用宽松的判断条件

尽可能及时地加入新的关键帧和三维点，以保证后续帧的鲁棒跟踪；再用严格的判断条件删除冗余的关键帧和不稳定的三维点，以保证 BA 的效率和精度。




ORB-SLAM 它是由三大块、三个流程同时运行的。第一块是跟踪，第二块是建图， 第三块是闭环检测。

1. 跟踪（Tracking）

这一部分主要工作是从图像中提取 ORB 特征，根据上一帧进行姿态估计，或者进行通过全局重定位初始化位姿，然后跟踪已经重建的局部地图，优化位姿，再根据一些规则确定新关键帧。

2. 建图（LocalMapping）

这一部分主要完成局部地图构建。包括对关键帧的插入，验证最近生成的地图点并进行筛选，然后生成新的地图点，使用局部捆集调整（Local BA），最后再对插入的关键帧进行筛选，去除多余的关键帧。

3. 闭环检测（LoopClosing）

这一部分主要分为两个过程，分别是闭环探测和闭环校正。闭环检测先使用 WOB 进行探测，然后通过 Sim3 算法计算相似变换。闭环校正，主要是闭环融合和 Essential

Graph 的图优化。

**ORB-SLAM 算法优点：**

一个代码构造优秀的视觉 SLAM 系统，非常适合移植到实际项目。采用 g2o 作为后端优化工具，能有效地减少对特征点位置和自身位姿的估计误差。采用 DBOW 减少了寻找特征的计算量，同时回环匹配和重定位效果较好。重定位：

比如当机器人遇到一些意外情况之后，它的数据流突然被打断了，在 ORB-SLAM 算法下，可以在短时间内重新把机器人在地图中定位。

使用了类似「适者生存」的方案来进行关键帧的删选，提高系统追踪的鲁棒性和系统的可持续运行。

提供最著名的公共数据集（ KITTI 和 TUM 数据集）的详尽实验结果，以显示其性能。可以使用开源代码，并且还支持使用 ROS。 (Github: slightech/MYNT-EYE-ORB-SLAM2-Sample）

**ORB-SLAM 算法缺点：**

构建出的地图是稀疏点云图。只保留了图像中特征点的一部分作为关键点，固定在空间中进行定位，很难描绘地图中的障碍物的存在。

初始化时最好保持低速运动，对准特征和几何纹理丰富的物体。

旋转时比较容易丢帧，特别是对于纯旋转，对噪声敏感，不具备尺度不变性。

如果使用纯视觉 slam 用于机器人导航，可能会精度不高，或者产生累积误差，漂移，尽管可以使用 DBoW 词袋可以用来回环检测。最好使用 VSLAM+IMU 进行融合， 可以提高精度上去，适用于实际应用中机器人的导航。

### 20.4.2 ORB-SLAM 算法 Demo 运行

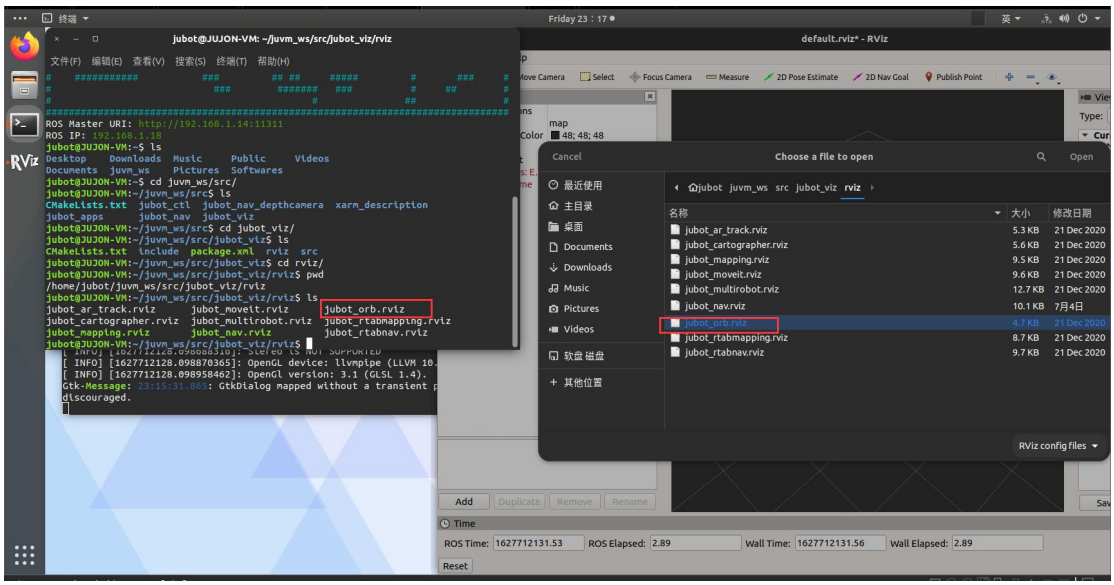
通过 SSH 连接到机器人端，执行如下命令，运行 ORB_SLAM 示例。

```
roslaunch jubot_nav_depthcamera jubot_ORBSLAM.launch
```

运行起来后需要等待加载一段时间，看到下面界面即启动成功：

```
  int (White Balance Temperature, 16, id = 98091a): 2800 to 6500 (1)
  int (Sharpness, 0, id = 98091b): 0 to 6 (1)
  int (Backlight Compensation, 0, id = 98091c): 0 to 2 (1)
  menu (Exposure, Auto, 0, id = 9a0901): 0 to 3 (1)
  int (Exposure (Absolute), 16, id = 9a0902): 1 to 5000 (1)
  bool (Exposure, Auto Priority, 0, id = 9a0903): 0 to 1 (1)

ORB-SLAM Copyright (C) 2014 Raul Mur-Artal
This program comes with ABSOLUTELY NO WARRANTY;
This is free software, and you are welcome to redistribute it
under certain conditions. See LICENSE.txt.
[ INFO] [1628233885.262038425]: Waiting For connections on 0.0.0.0:8080

Loading ORB Vocabulary. This could take a while.
Setting focus_absolute is not supported
Vocabulary loaded!

Camera Parameters:
- fx: 609.286
- fy: 609.342
- cx: 351.427
- cy: 237.732
- k1: -0.3492
- k2: 0.1363
- p1: 0
- p2: 0
- fps: 30
- color order: RGB (ignored if grayscale)

ORB Extractor Parameters:
- Number of Features: 1000
- Scale Levels: 8
- Scale Factor: 1.2
- Fast Threshold: 20
- Score: FAST

Motion Model: Enabled
```

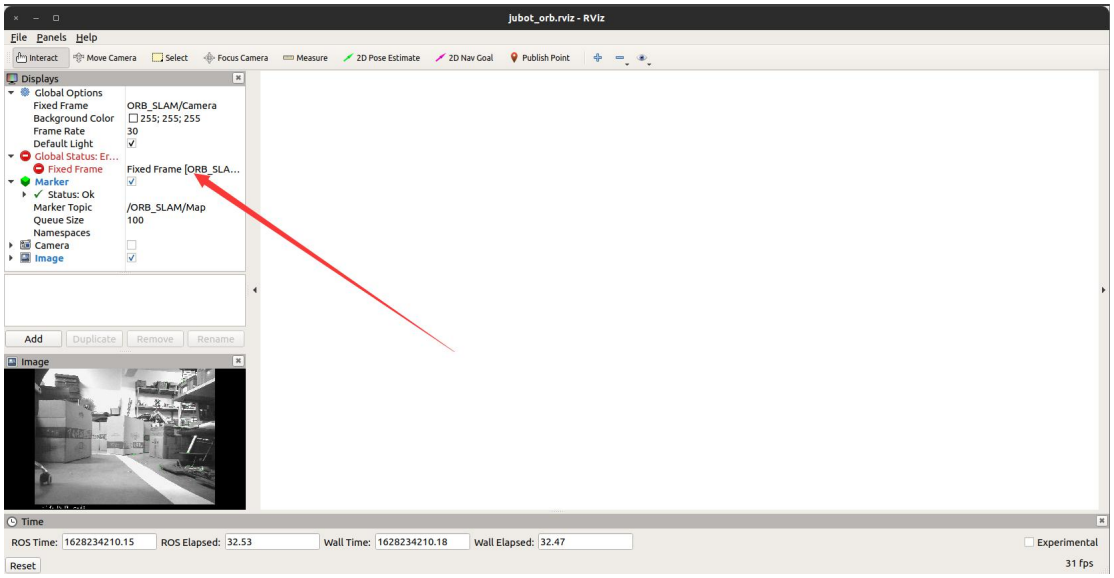
启动成功后在虚拟机端再打开一个终端，运行 `rosrun rviz rviz` ，启动可视化界面，并打开 juvm_ws/src/jubot_viz/rviz/文件夹下的 jubot_orb.rviz 显示配置文件，此时，可以看到 ORB_SLAM 算法界面。



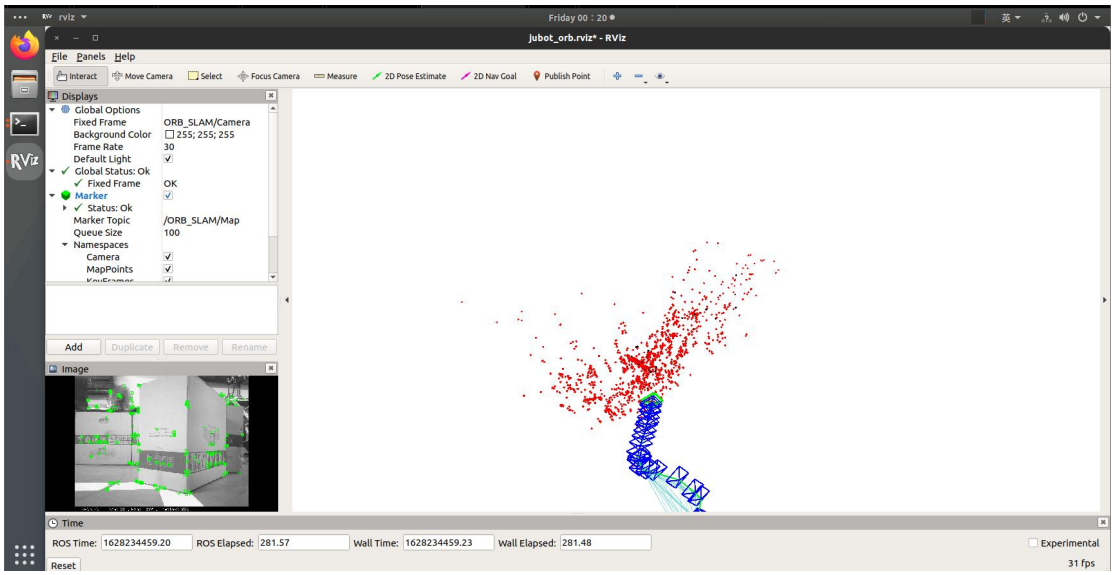

在虚拟机端可以运行 `roslaunch jubot_ctl jubot_keyboard.launch` 启动键盘节点控制机器人移动，移动时因为受制于处理器处理速度及相机帧率，尽量降低移动速度，否则建图效果较差。

刚启动时，ORB-SLAM 算法需要完成初始化步骤，在完成初始化之前，RVIZ 显示全白，需要用户将机器人朝向一个方向，多角度移动机器人，即让机器人多个视角拍摄一个方向，最好选择纹理丰富的视角。完成初始化后，即可看到 ORB-SLAM 算法图像。

一进入可能会报下面错误，暂时不予理会




使用手机app或者键盘控制路威套件移动上面错误即可消失。




## 20.4.3 ORB-SLAM2 算法介绍

ORB-SLAM2 是基于单目，双目和 RGB-D 相机的一套完整的 SLAM 方案。它能够实现地图重用，回环检测和重新定位的功能。无论是在室内的小型手持设备，还是到工厂环境的无人机和城市里驾驶的汽车，ORB-SLAM2 都能够在标准的 CPU 上进行实时工作。ORB-SLAM2 在后端上采用的是基于单目和双目的光束法平差优化（BA）的方式，这个方法允许米制比例尺的轨迹精确度评估。此外，ORB-SLAM2 包含一个轻量级的定位模式，该模式能够在允许零点漂移的条件下，利用视觉里程计来追踪未建图的区域并且匹配特征点。

项目地址：（https://github.com/raulmur/ORB_SLAM2）

### 20.4.4 ORB-SLAM2 算法 Demo 运行

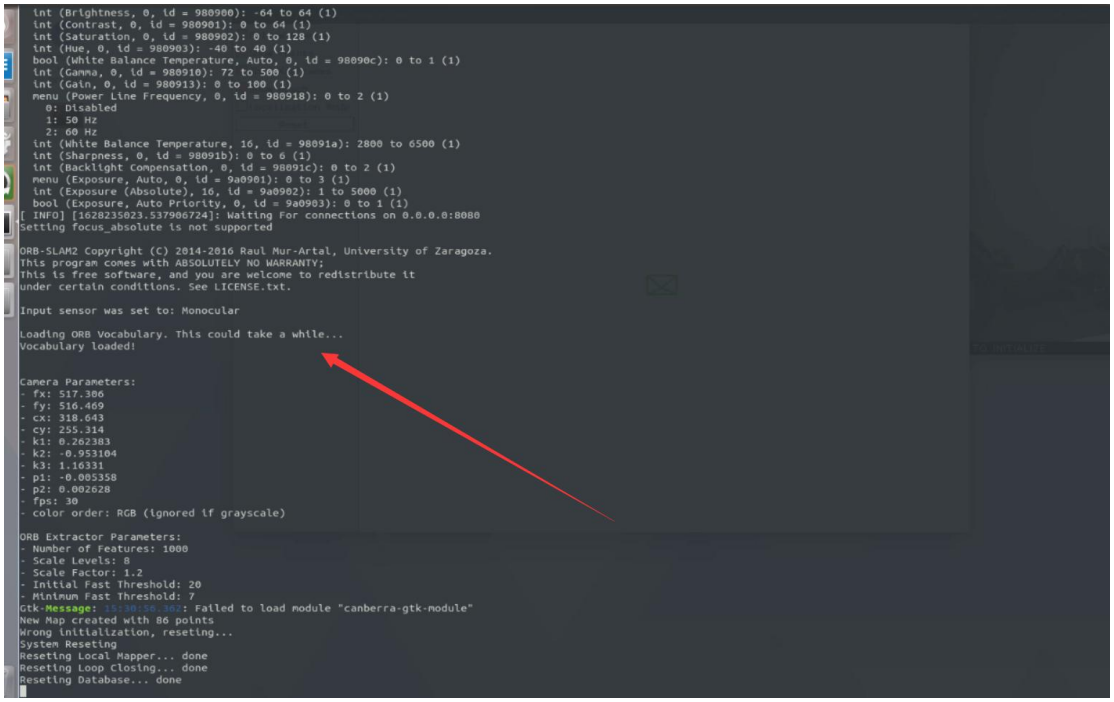
ORB-SLAM2 算法与 ORB-SLAM 算法一个很大的不同点就是，ORB-SLAM2 算法在设计时降低了对 ROS 的依赖，也就是说，ORB-SLAM2 并不想对 ROS 进行很深的绑定与依赖，所以，相对于 ORB-SLAM 来说，ORB-SLAM2 对 ROS 的支持与利用没有那么完善，比如说 ORB-SLAM2 没有通过 ROS 话题来发布算法中得到的数据。

为了方便大家快速进行 ORB-SLAM2 效果的验证与简单使用，我们采用了 ORB-SLAM2 的 ROS 启动的方法来启动该算法，与 ORB-SLAM 不同的是，因为 ORB-SLAM2 对 ROS 的接口并不是那么多，所以无法利用 ROS 的 Rviz 工具来显示算法数据，而是采用了 ORB-SLAM2 自带的一个图像显示界面来显示算法效果，所以，在运行 ORB- SLAM2 时，我们需要通过远程桌面的方法来进行。

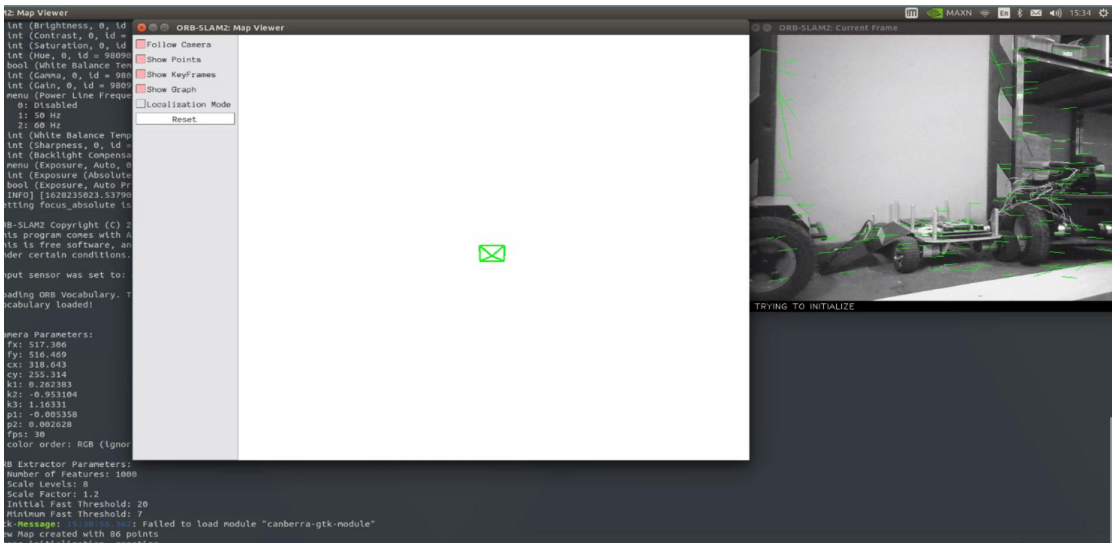
首先，通过 NoMachine 连接到机器人远程桌面，并打开终端，运行如下命令启动 ORB- SLAM2 ：

```
roslaunch jubot_nav_depthcamera jubot_ORBSLAM2.launch
```

同样要等待加载一段时间

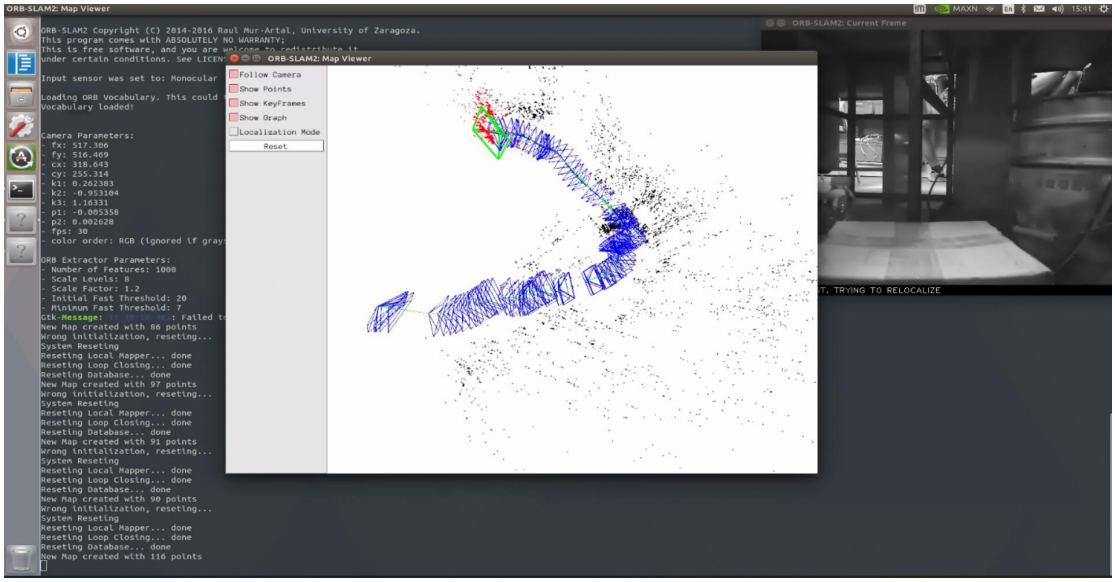



等待算法启动后，可以看到下面的图




算法打开了两个窗口，一个是算法当前处理视角图片，包含了特征点信息，后方白色窗口为机器人关键点地图与机器人姿态显示。

与 ORB-SLAM 算法相同，刚启动时，ORB-SLAM2 算法需要完成初始化步骤，在完成初始化之前，窗口显示全白，需要用户将机器人朝向一个方向，多角度移动机器人，即让机器人多个视角拍摄一个方向，最好选择纹理丰富的视角。完成初始化后，即可看到 ORB-SLAM2 算法对机器人的定位。




## 20.5 实验结果

能够理解 ORB-SLAM & ORB-SLAM2 原理，能够获取到ORB-SLAM & ORB-SLAM2信息。

## 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 RTAB-SLAM 视觉 SLAM 算法

## 21.1 实验目的

可以在 ROS 系统中依靠 RTAB-SLAM 算法实现VSLAM建图导航。

## 21.2 实验要求

理解 RTAB-SLAM 概念掌握 RTAB-SLAM 的基本使用。

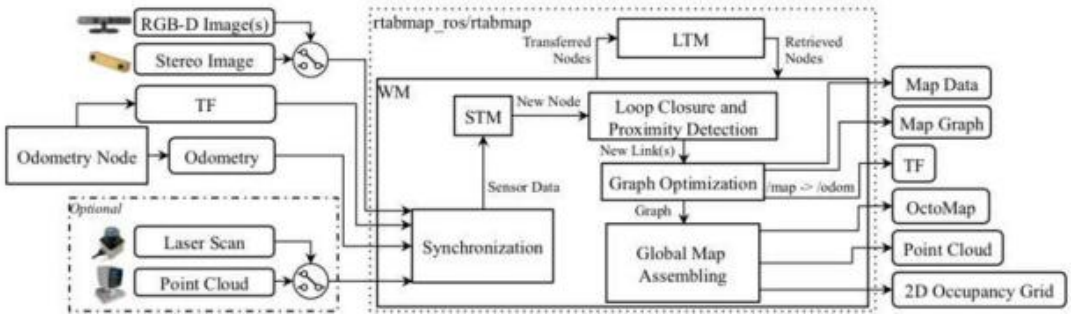
## 21.3 实验工具

个人电脑一台，路威套件。

## 21.4 实验内容

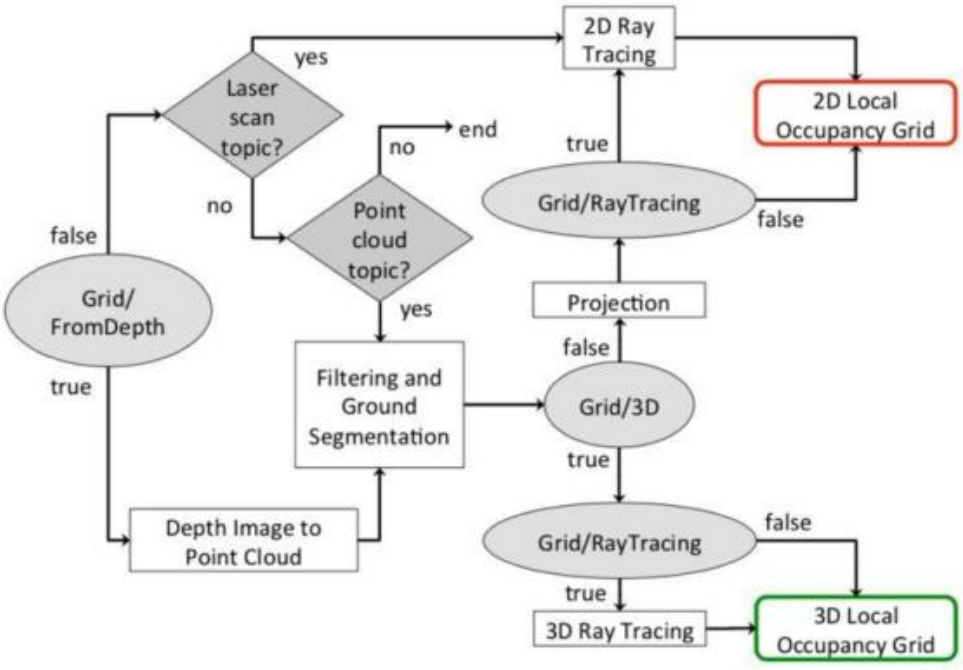
### 21.4.1 RTAB-SLAM 简介

RTAB-Map，（for Real-Time Appearance-Based Mapping）用于基于外观的实时建图[Labbe and Michaud, 2013,Labbe and Michaud, 2017]，是一个通过内存管理方法实现回环检测的开源库。从限制地图的大小以使得回环检测始终在固定的时间限制内处理，从而满足长期和大规模环境在线建图要求。 从 2013 年开始并于 2013 年作为开源库发布，RTAB-Map 已经扩展到完整的基于图的 SLAM 方法[Stachniss et al., 2016]，被用于各种设置和应用[Laniel et al., 2017, Foresti et al., 2016, Chen et al., 2015, Goebel, 2014]。 因此 RTAB-Map 已经发展成为一个跨平台的独立 C++ 库和一个 ROS 包，且由以下实际需求驱动着：在线处理、鲁棒而低漂移的里程计、鲁棒的定位、实用的地图生成和开发、多会话的建图（又称为机器人绑架问题或初始化状态问题）。

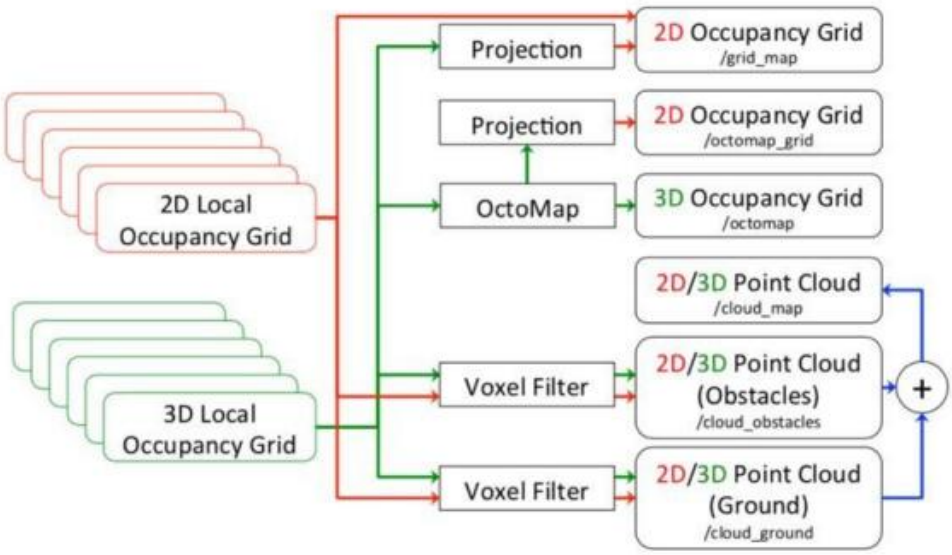



以上为 RTAB-Map ROS 节点的框图。所需输入是：TF，用于定义传感器相对于机器人底座的位置；来自任何来源的里程计（可以是 3DoF 或 6DoF）；其中一种相机输入（一

个或多个 RGB-D 图像，或双目立体图像），且带有相应的校准消息。可选输入：2D 激光的雷达扫描，或 3D 激光的点云。然后，来自这些输入的所有消息被同步并传递给 graph-SLAM 算法。输出的是：Map Data，包含最新添加的节点（带有压缩传感器数据）和 Graph; Map Graph，没有任何数据的纯图;TF，矫正过的里程计; 可选的OctoMap（3D 占用栅格地图）; 可选的稠密点云地图; 可选的 2D 占用栅格地图。



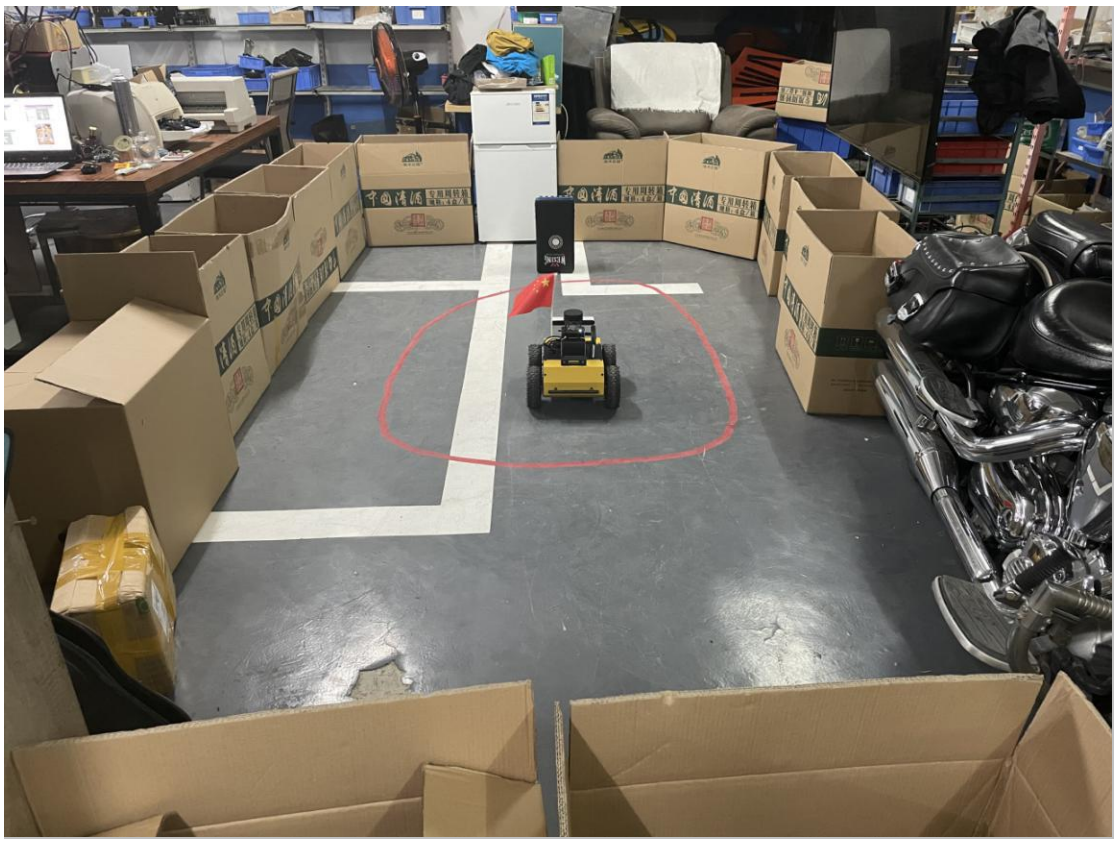

STM 的局部占用栅格地图创建。依赖的参数（由椭圆显示），可选用激光扫描和点云输入（由棱形显示），局部占用栅格地图可以是 2D 或 3D。




生成 3D 占用栅格地图（OctoMap）及其 2D 地图。

### 21.4.2 使用RTAB-SLAM建图

首先选择一块合适的场地，测试场地如下：

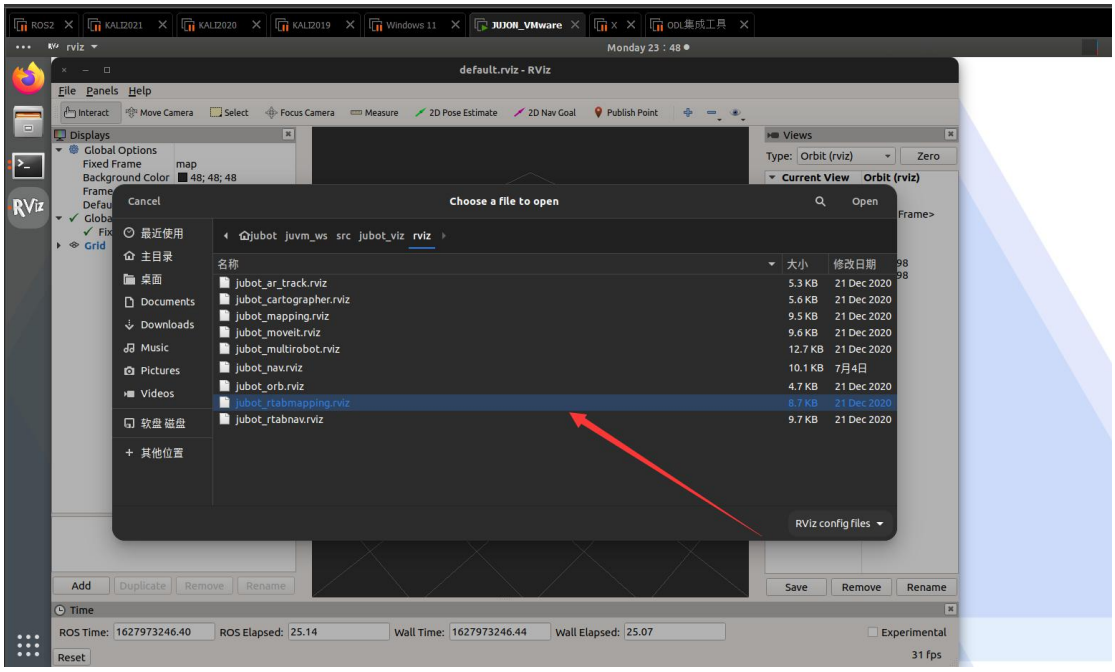


运行下面命令启动：

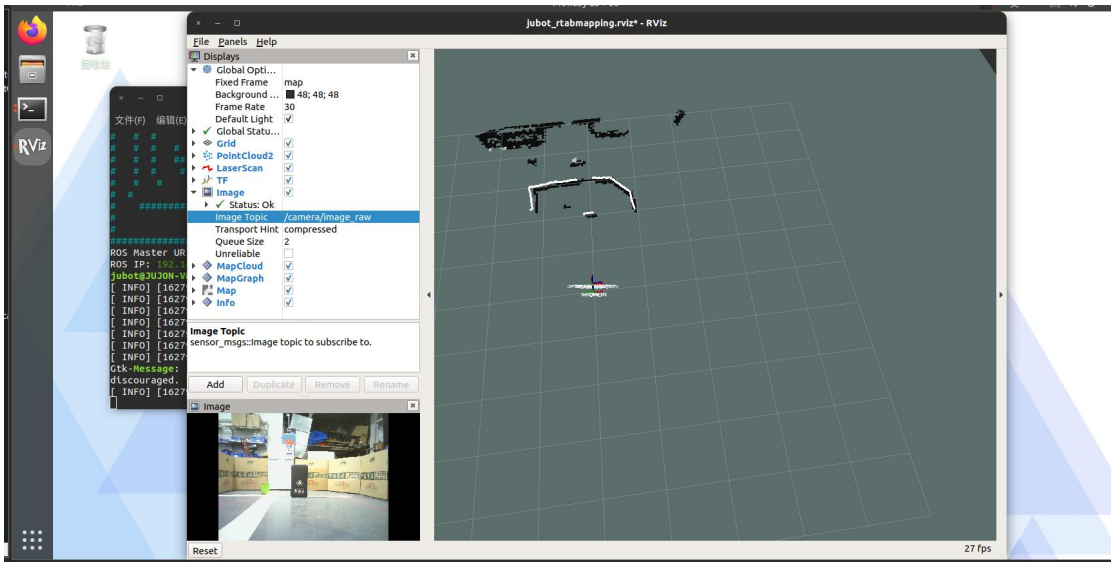
```
roslaunch jubot_nav_depthcamera jubot_RTABSLAM_Mapping.launch
```

启动成功后在虚拟机端再打开一个终端，运行 `rosrun rviz rviz` ，启动可视化界面，

并打开 juvm_ws/src/jubot_viz/rviz/文件夹下的 jubot_rtabmapping.rviz 显示配置文件，此时，可以看到 RTAB-SLAM 算法建图界面。

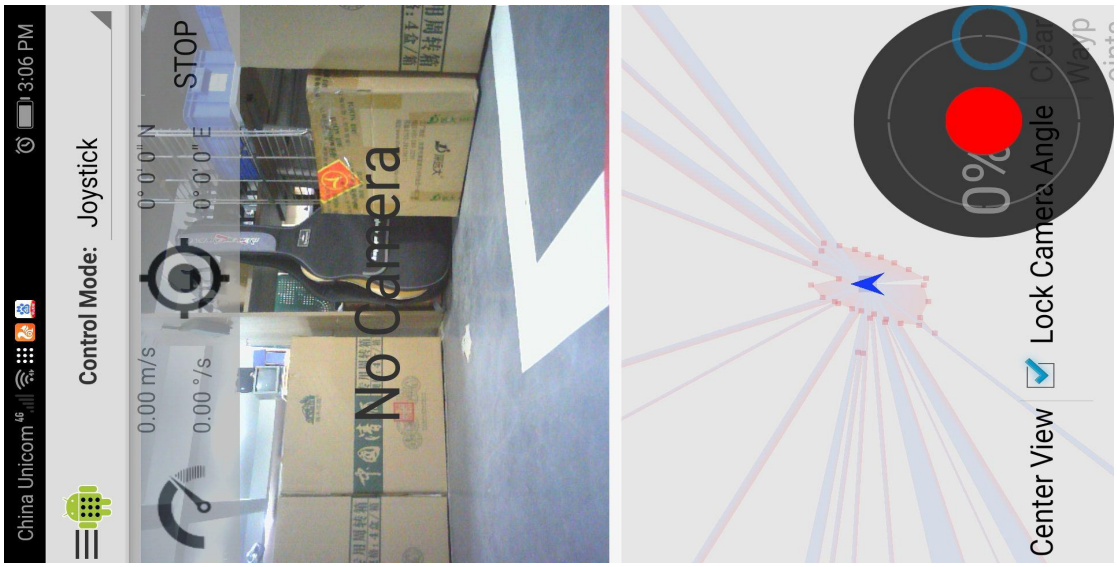



启动后：

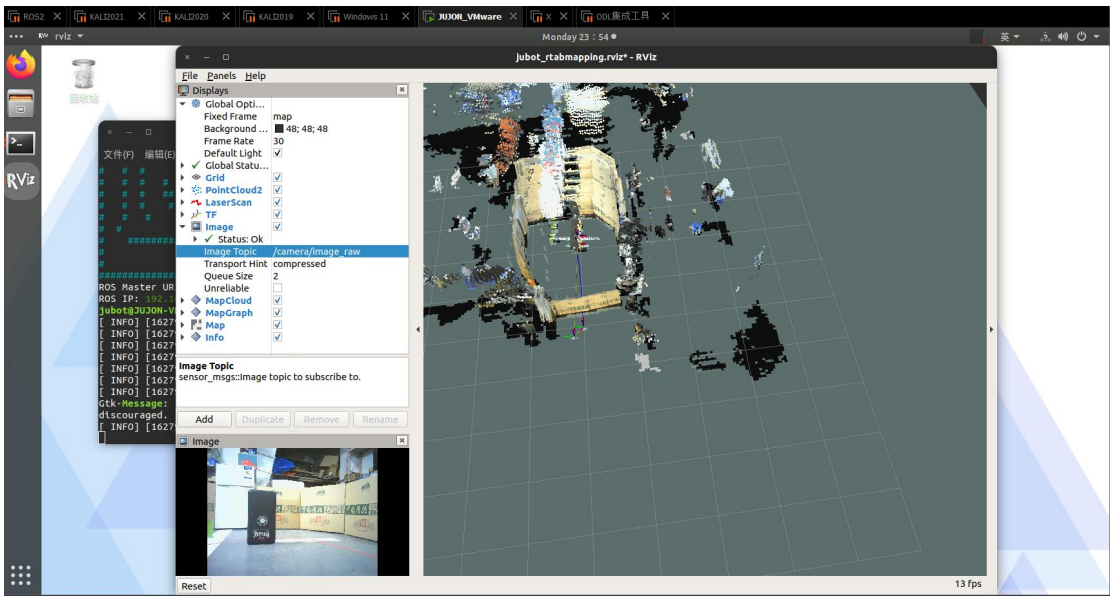



此时，可在虚拟机端可以运行

`rosrun teleop_twist_keyboard teleop_twist_keyboard` 或启动键盘/手柄遥控节点控制机器人移动，建立场景 3D 地图。也可以在手机端控制机器人移动。




RTAB-SLAM 建图带有闭环修正环节，如发现建图偏差较大，可控制机器人多走几遍，RTAB 算法可自动完成闭环修正。建图效果如下。




当完成建图后，可以直接 ctrl+c 退出建图节点，算法将会自动保存地图。（RTAB-SLAM 算法地图保存为*.db 格式，自动保存路径~/.ros/rtabmap.db），无需手动保存地图。

```
564646630 VERBOSE    Setting Firmware.Stream0Mode (5) to 0...
564650429 VERBOSE    Firmware.Stream0Mode (5) was successfully set.
564650464 VERBOSE    Setting Firmware.Stream1Mode (6) to 0...
564652978 VERBOSE    Firmware.Stream1Mode (6) was successfully set.
564653009 VERBOSE    Setting Firmware.Stream2Mode (7) to 0...
564654585 VERBOSE    Firmware.Stream2Mode (7) was successfully set.
564654615 VERBOSE    Shutting down USB depth read thread...
564654633 VERBOSE    Shutting down USB image read thread...
564654798 VERBOSE    Shutting down USB events thread...
564654885 VERBOSE    Device closed successfully
564654917 VERBOSE    Shutting down Scheduler thread...
564656390 VERBOSE    Shutting down USB depth read thread...
564656419 VERBOSE    Shutting down USB image read thread...
564656429 VERBOSE    Device closed successfully
rtabmap: Saving database/long-term memory... (located at /home/jubot/.ros/rtabmap.db)
rtabmap: 2D occupancy grid map saved.
rtabmap: Saving database/long-term memory...done! (located at /home/jubot/.ros/rtabmap.db, 224 MB)
[camera/camera_nodelet_manager-5] escalating to SIGTERM
[camera/camera_nodelet_manager-5] escalating to SIGKILL
[rosout-1] killing on exit
[master] killing on exit
Shutdown errors:
 * process[camera/camera_nodelet_manager-5, pid 12161]: required SIGKILL. May still be running.
shutting down processing monitor...
... shutting down processing monitor complete
done
[jubot@JUJON-ROBOT ~]$
```

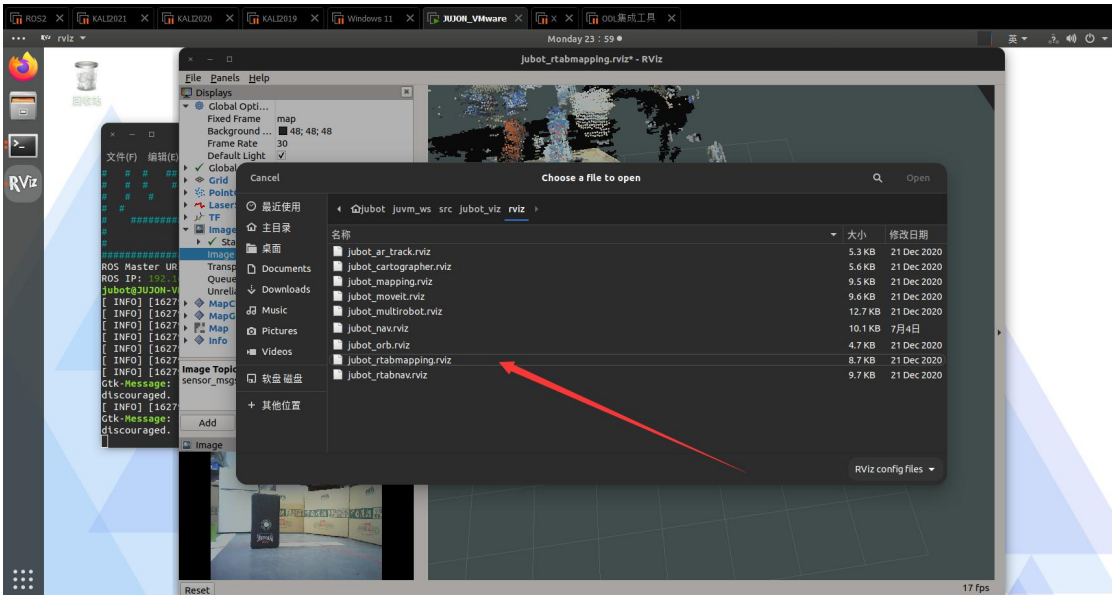
## 21.4.3 深度相机+激光雷达融合建图

此例程为深度相机与激光雷达融合建图，相对于单独深度相机建图，融合建图效果更优，精度更高。

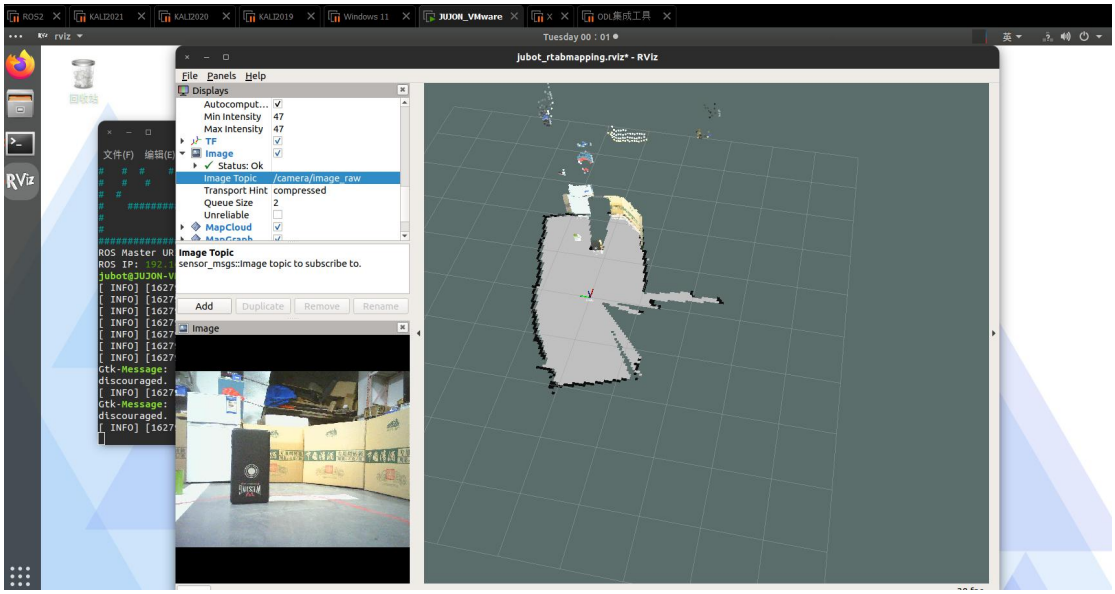
通过 SSH 连接到机器人，执行如下命令，启动 RTAB-SLAM 深度相机+激光雷达融合建图。

```
roslaunch jubot_nav_depthcamera jubot_RTABSLAM_Mapping_UseLidar.launch
```

启动成功后在虚拟机端再打开一个终端，运行 `rosrun rviz rviz`，启动可视化界面，并打开 juvm_ws/src/jubot_viz/rviz/文件夹下的 jubot_rtabmapping.rviz 显示配置文件，此时，可以看到 RTAB-SLAM 算法建图界面。



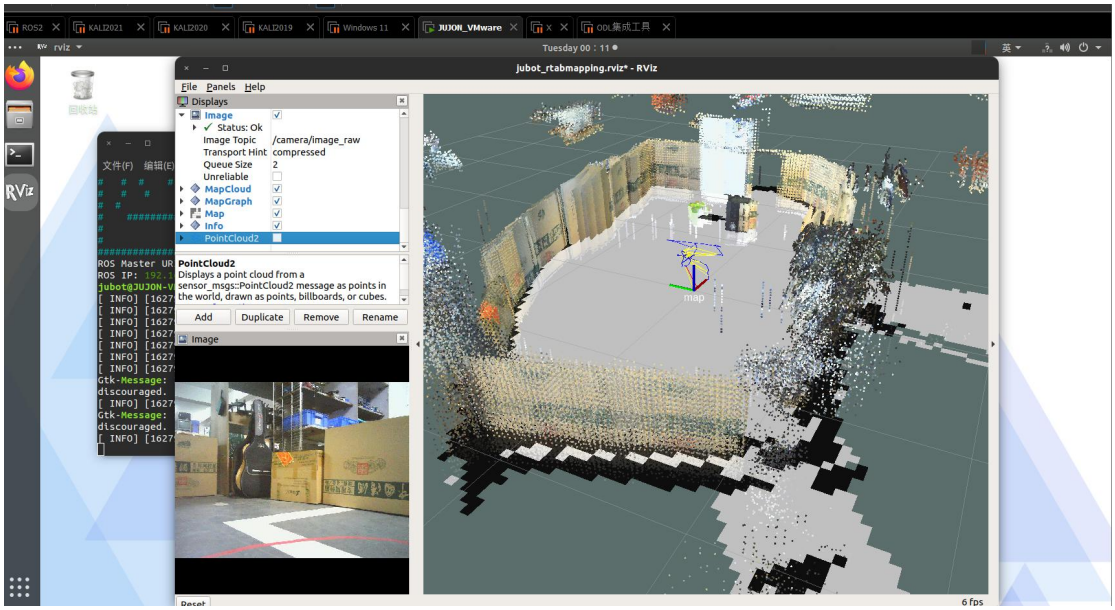

启动后：




此时，可在虚拟机端可以运行。

`roslaunch jubot_ctl jubot_keyboard.launch` 或启动键盘/手柄遥控节点控制机器人移动，建立场景 3D 地图。

RTAB-SLAM 建图带有闭环修正环节，如发现建图偏差较大，可控制机器人多走几遍，RTAB 算法可自动完成闭环修正。建图效果如下。




当完成建图后，可以直接 ctrl+c 退出建图节点，算法将会自动保存地图。（RTAB-SLAM 算法地图保存为*.db 格式，自动保存路径~/.ros/rtabmap.db），无需手动保存地图。

```
[robot_pose_ekf-3] killing on exit
[jubot_driver-2] killing on exit
947833581 INFO       Destroying stream 'IR'...
947834471 VERBOSE    'IR' stream destroyed.
947834715 INFO       Destroying stream 'Depth'...
947836379 VERBOSE    'Depth' stream destroyed.
947836550 VERBOSE    Shutting down Sensor commands.txt thread...
947836598 VERBOSE    Setting Firmware.Stream0Mode (5) to 0...
947839294 VERBOSE    Firmware.Stream0Mode (5) was successfully set.
947839349 VERBOSE    Setting Firmware.Stream1Mode (6) to 0...
947841721 VERBOSE    Firmware.Stream1Mode (6) was successfully set.
947841770 VERBOSE    Setting Firmware.Stream2Mode (7) to 0...
947844084 VERBOSE    Firmware.Stream2Mode (7) was successfully set.
947844137 VERBOSE    Shutting down USB depth read thread...
947844163 VERBOSE    Shutting down USB image read thread...
947844909 VERBOSE    Shutting down USB events thread...
947844979 VERBOSE    Device closed successfully
947845015 VERBOSE    Shutting down Scheduler thread...
947846600 VERBOSE    Shutting down USB depth read thread...
947846653 VERBOSE    Shutting down USB image read thread...
947846680 VERBOSE    Device closed successfully
rtabmap: Saving database/long-term memory... (located at /home/jubot/.ros/rtabmap.db)
rtabmap: 2D occupancy grid map saved.
rtabmap: Saving database/long-term memory...done! (located at /home/jubot/.ros/rtabmap.db, 345 MB)
[camera/camera_nodelet_manager-5] escalating to SIGTERM
[camera/camera_nodelet_manager-5] escalating to SIGKILL
[rosout-1] killing on exit
[master] killing on exit
Shutdown errors:
 * process[camera/camera_nodelet_manager-5, pid 17938]: required SIGKILL. May still be running.
shutting down processing monitor...
... shutting down processing monitor complete
done
[jubot@JUJON-ROBOT ~]$
```

## 21.4.4 深度相机导航

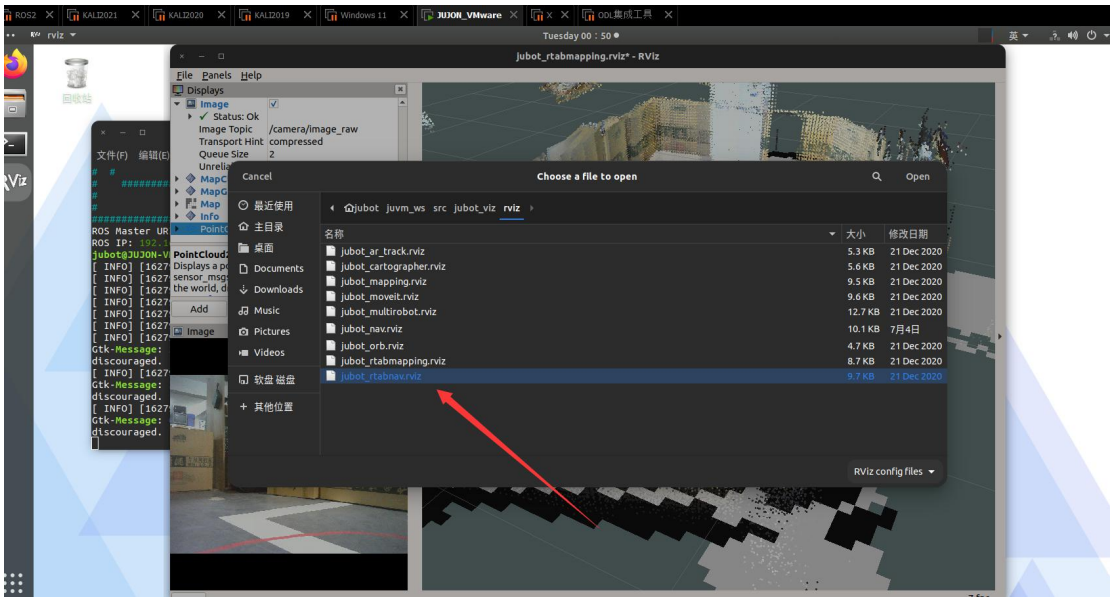
此例程为单独利用深度相机进行 RTAB-SLAM 算法导航，不启动激光雷达，同时用深度相机模拟激光雷达数据提高算法效果。

在使用 RTAB-SLAM 导航时，算法会自动加载~/.ros/rtabmap.db 地图，无需手动设置。

通过 SSH 连接到机器人，执行如下命令，启动 RTAB-SLAM 深度相机导航。

```
roslaunch jubot_nav_depthcamera jubot_RTABSLAM_Navigation.launch
```

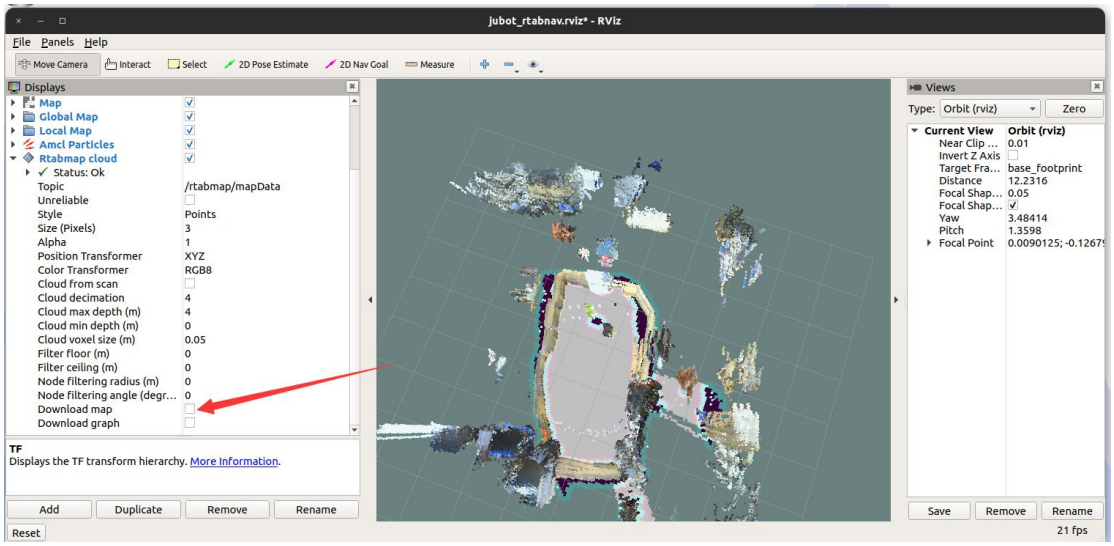
启动成功后在虚拟机端再打开一个终端，运行 `rosrun rviz rviz` ，启动可视化界面，并打开 juvm_ws/src/jubot_viz/rviz/文件夹下的 jubot_rtabnav.rviz



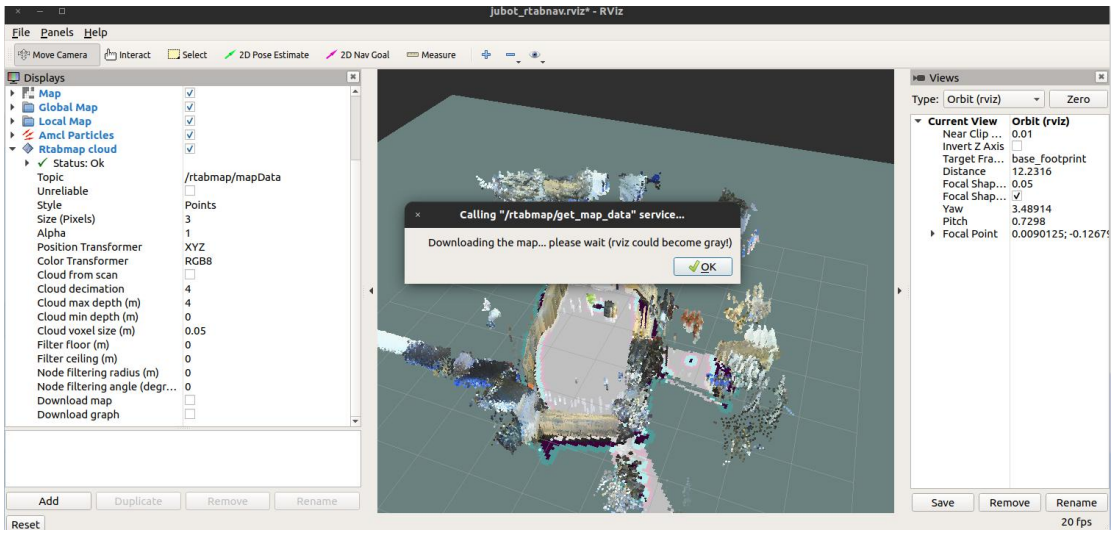

显示配置文件，此时，可以看到 RTAB-SLAM 导航界面。

启动导航后，RTAB-SLAM 算法可以自动匹配机器人当前位置，无需手动标定机器人位姿。

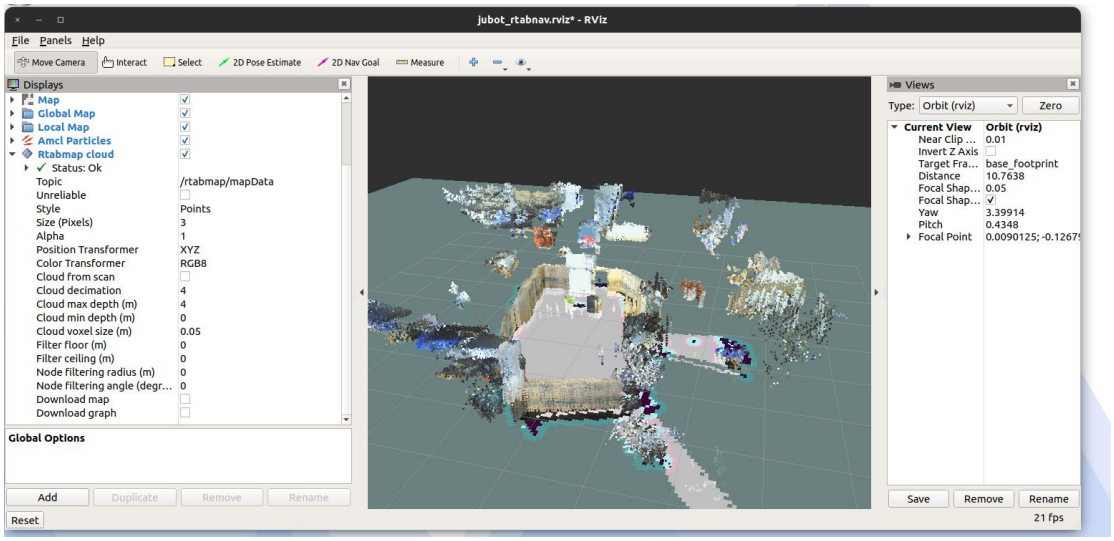
在默认启动后，机器人所建立的 3D 地图因存放在机器人端，所以 RVIZ 没有显示建立的 3D 地图，可以通过电机 Download Map 按钮，使 rviz 下载 3D 地图，显示 3D 地图。如下所示：




点击后，rviz 开始从机器人端下载 3D 地图文件，需要一段时间，rviz 画面也有可能变灰，耐心等待下载完成。




下载完成后：




此时，即可利用 2D Nav Goal 工具指定目标点，机器人即可前往目标点。

## 深度相机+激光雷达融合导航

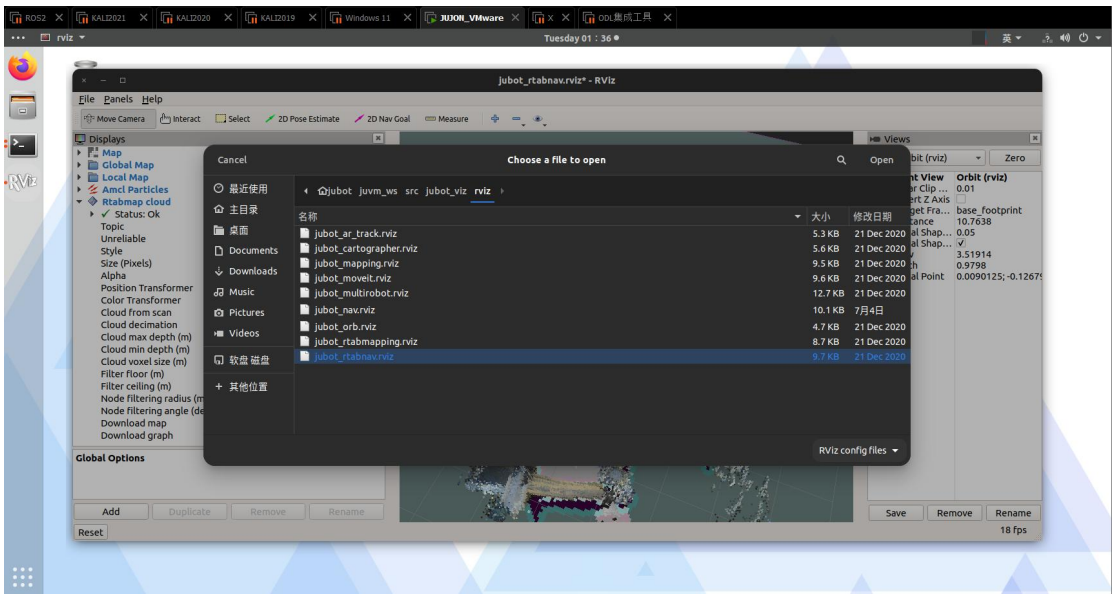
此例程为利用深度相机与激光雷达融合进行 RTAB-SLAM 算法导航。

在使用 RTAB-SLAM 导航时，算法会自动加载~/.ros/rtabmap.db 地图，无需手动设置。

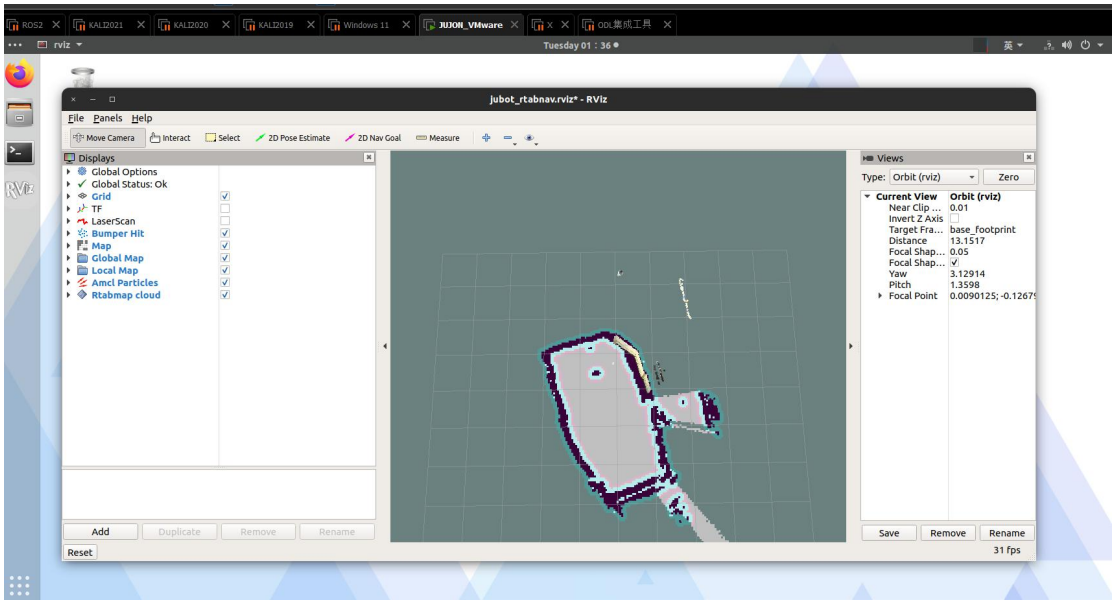
通过 SSH 连接到机器人，执行如下命令，启动 RTAB-SLAM 深度相机导航。

```
roslaunch jubot_nav_depthcamera jubot_RTABSLAM_Navigation_UseLidar.launch
```

启动成功后在虚拟机端再打开一个终端，运行 `rosrun rviz rviz` ，启动可视化界面，并打开 juvm_ws/src/jubot_viz/rviz/文件夹下的 jubot_rtabnav.rviz

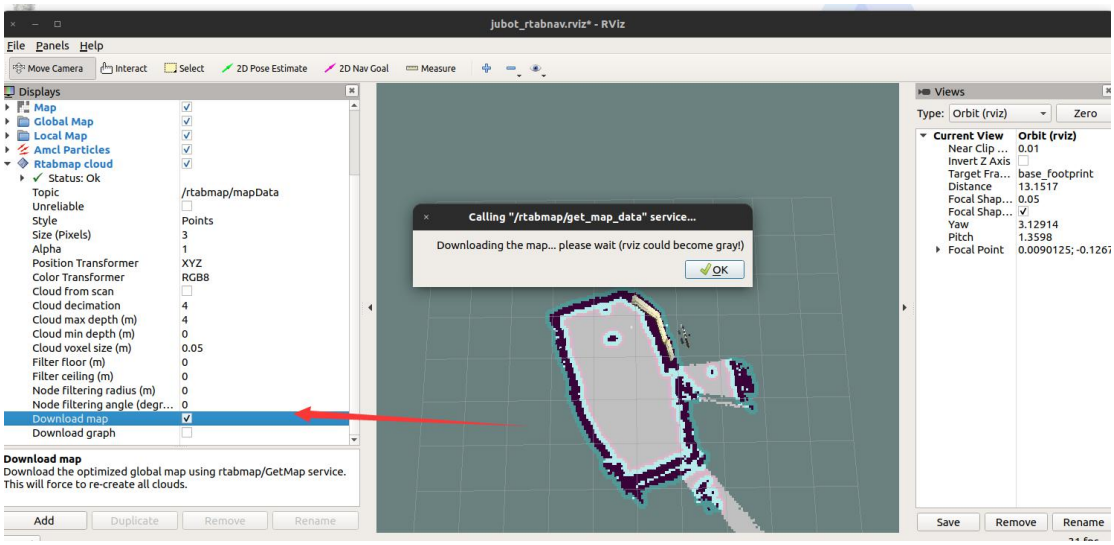



显示配置文件，此时，可以看到 RTAB-SLAM 导航界面。



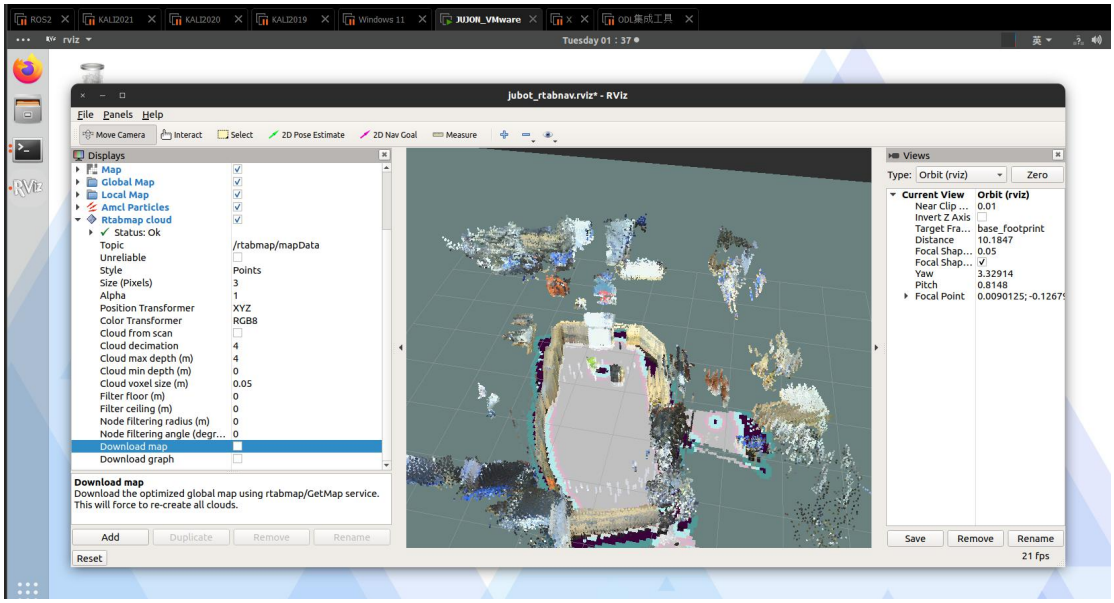

启动导航后，RTAB-SLAM 算法可以自动匹配机器人当前位置，无需手动标定机器人位姿。

在默认启动后，机器人所建立的 3D 地图因存放在机器人端，所以 RVIZ 没有显示建立的 3D 地图，可以通过点击 Download Map 按钮，使 rviz 下载 3D 地图，显示 3D 地图。如下所示：

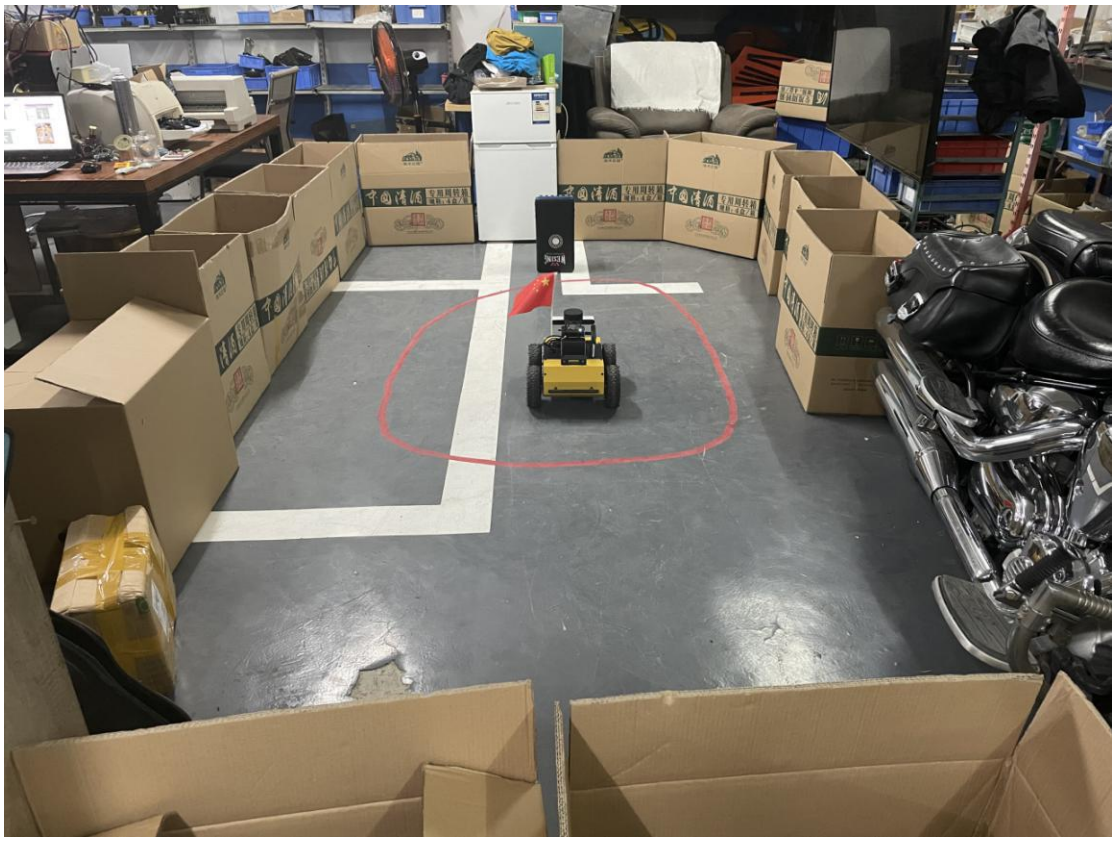



点击后，rviz 开始从机器人端下载 3D 地图文件，需要一段时间，rviz 画面也有可能变灰，耐心等待下载完成。

下载完成后：




对比我们真实场地，可以看出建图效果较为良好。



此时，即可利用 2D Nav Goal 工具指定目标点，机器人即可前往目标点。

## 21.5 实验结果

能够理解 RTAB-SLAM 算法原理，能够使用 RTAB 算法进行建图导航，并与雷达节点进行融合。

## 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 卷积神经网络(YOLO)

## 22.1 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。

## 22.2 实验要求

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

## 22.3 实验工具

个人电脑一台，路威套件。

## 22.4 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。

### 22.4.1 YOLO 网络简介

YOLO（you only look once, YOLO）是基于一款小众的深度学习框架——

darknet 的目标检测开源项目，darknet 短小精悍，虽然功能和复用性不如当前大火的深度学习框架 Tensorflow 和 Caffe，但由于其源码都是用纯 C 语言和 CUDA 底层编写的， 所以它的特点让它在 YOLO 项目中大放光彩：速度快，充分发挥多核处理器和 GPU 并行运算的功能。所以，YOLO 的快速检测正好适合我们这种需要实时检测视频帧的项目；此外，它的准确度也非常高，在尺寸中等偏小的物体上有非常高的准确率，这得益于它的训练方式,但在大尺寸的物体，比如占到了整个图片百分之 60 的物体，识别率则不尽如人意。

YOLO 是一个目标检测算法项目，而目标检测的本质，就是识别与回归，而处理图像用的最多的就是卷积神经网络 CNN，所以，YOLO 本质上，就是一个实现了回归功能的深度卷积神经网络。

**YOLO 特征提取方式**

既然是训练卷积神经网络，就要提取图片信息的特征。相对于 FAST R-CNN 目标检测模型中使用region proposal（候选区域）特征提取，YOLO 选择了对于图片的全局区域进行训练，速度加快的同时，能够更好的区分目标和背景，但是对于大物体而言，背景也有可能被算进目标的一部分，所以这就是它对中小物体效果特别好，而大背景却不尽如人意的原因。

### YOLO 网络预测方式

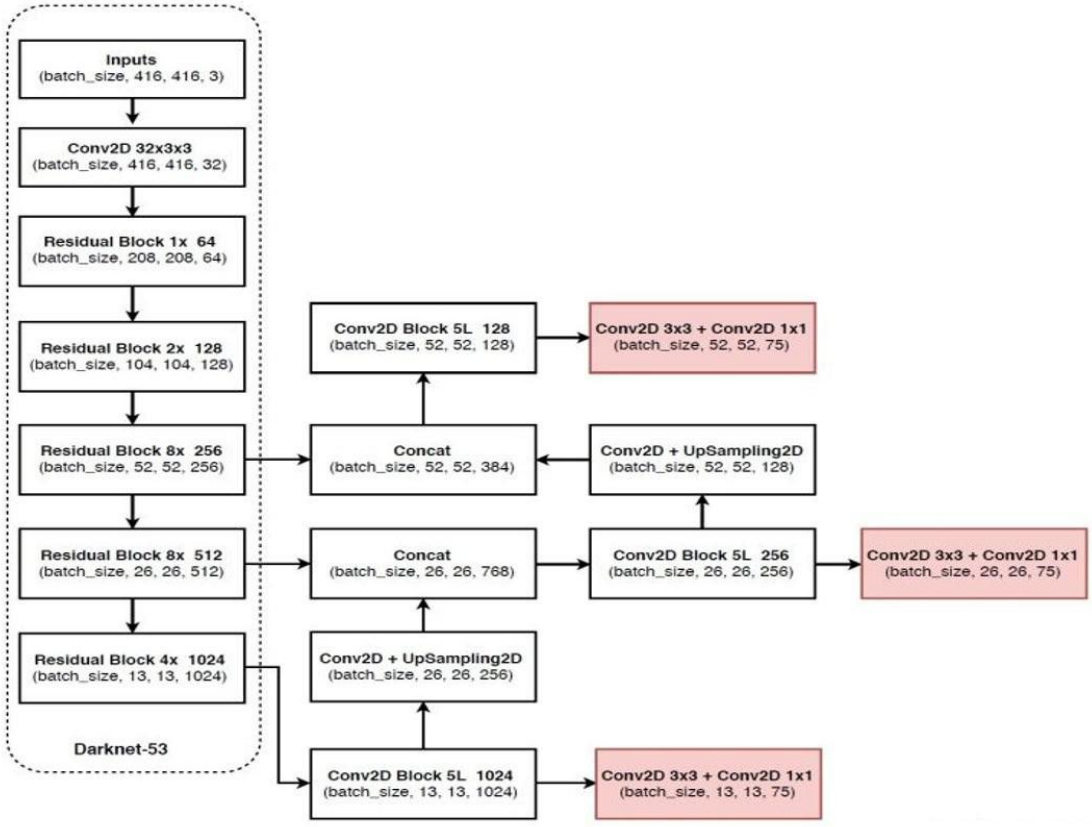
YOLO 在预测图片上采用的是端对端的检测，将整个图片的分为 S*S 个区域，而如果一个物体的中心落在某个区域上，则对应的网络会对它进行检测。

而对于每个网络，都有一个 bounding box ，就是预测区域，每次预测时有四个坐标参数，左上角的 xi ， yi ，宽度和高度 tw ，th，以及一个置信度。这个置信度就是逻辑回归的产物。置信度判断这个 bounding box 是否会被忽略，如果不会被忽略，则又会进行多标签分类的逻辑回归，从而贴上标签。

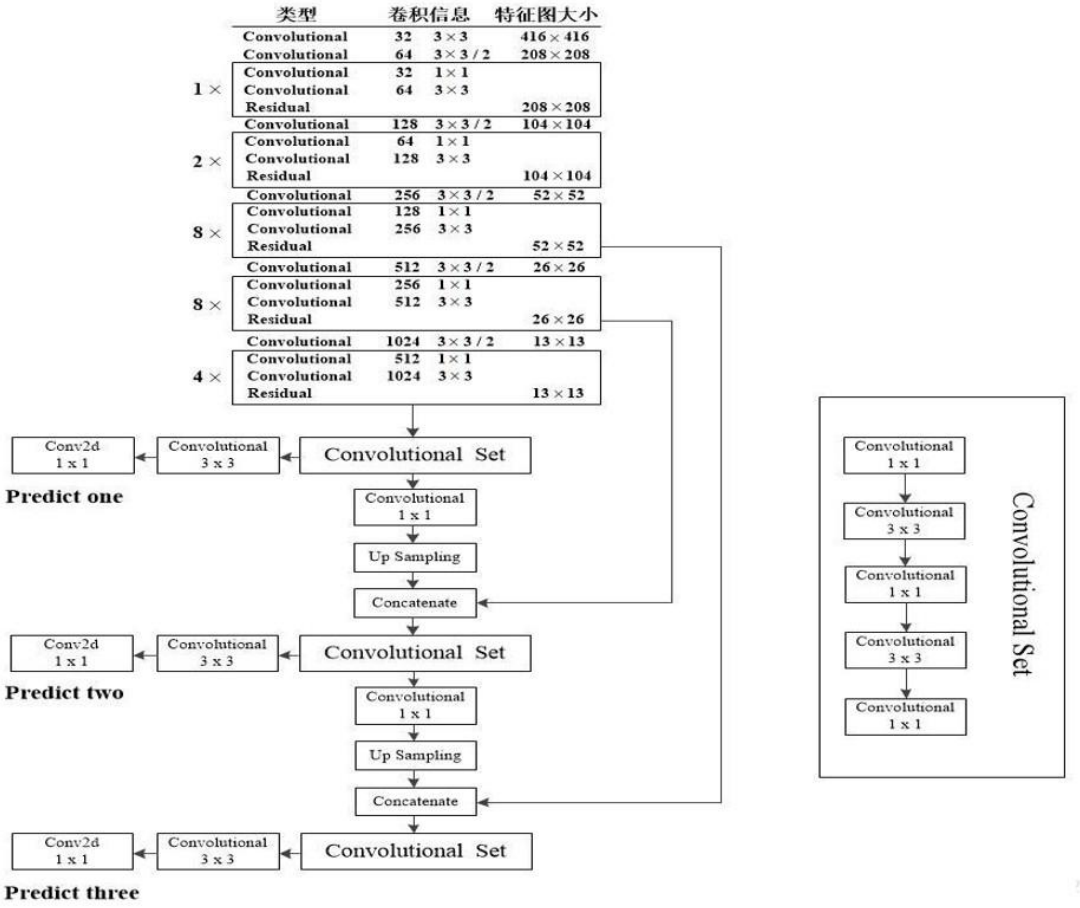
### YOLO 网络模型

YOLO 最重要的内容就是一个庞大而丰富的深度卷积神经网络模型，它一共有 53 个全连接卷积层，所以作者在 Github 上又将该项目称为 Darknet-53，但实际上卷积层不止 53 层，因为特征提取也用到了大量的卷积核。

### YOLOV3 网络结构图




下面这张图呈现了Darknet 在测试时的网络连接状态：

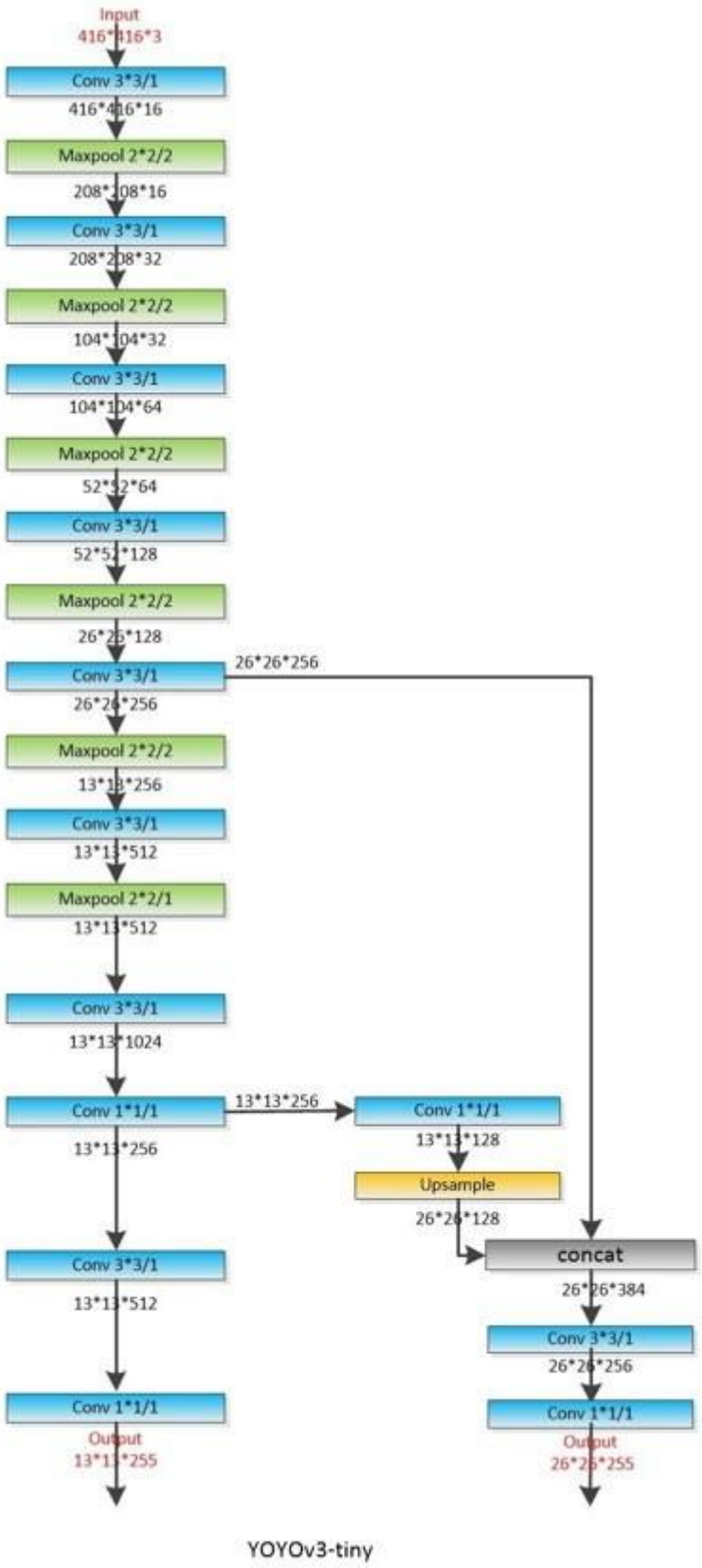



(1)一张 416*416 大小的图片输入，会经过很多层的深度卷积（图片中略过了层数）一直降维到 52,26 和 13。

(2)在 52,26 和 13 维分别有三个全卷积特征提取器，对应的是右边的 Convolutional Set，这就是特征提取器的内部卷积核结构，1*1 的卷积核用于降维，3*3 的卷积核用于提取特征，多个卷积核交错达到目的。每个全卷积特征层是有连接的， 在图中为 Concatenate 标志，意味着当前特征层的输入有来自于上一层的输出的一部分。每个特征层都有一个输出 Predict，即预测结果，最后根据置信度大小对结果进行回归，得到最终的预测结果。

YOLO-Tiny

对于速度要求比较高的项目，YOLO-tiny 才是我们的首要选择。如下图这个网络在 YOLOv3 的基础上去掉了一些特征层，只保留了 2 个独立预测分支，具体的结构图如下：




YOYOv3-tiny

## 22.4.2 在 ROS 中运行 YOLO 物体识别

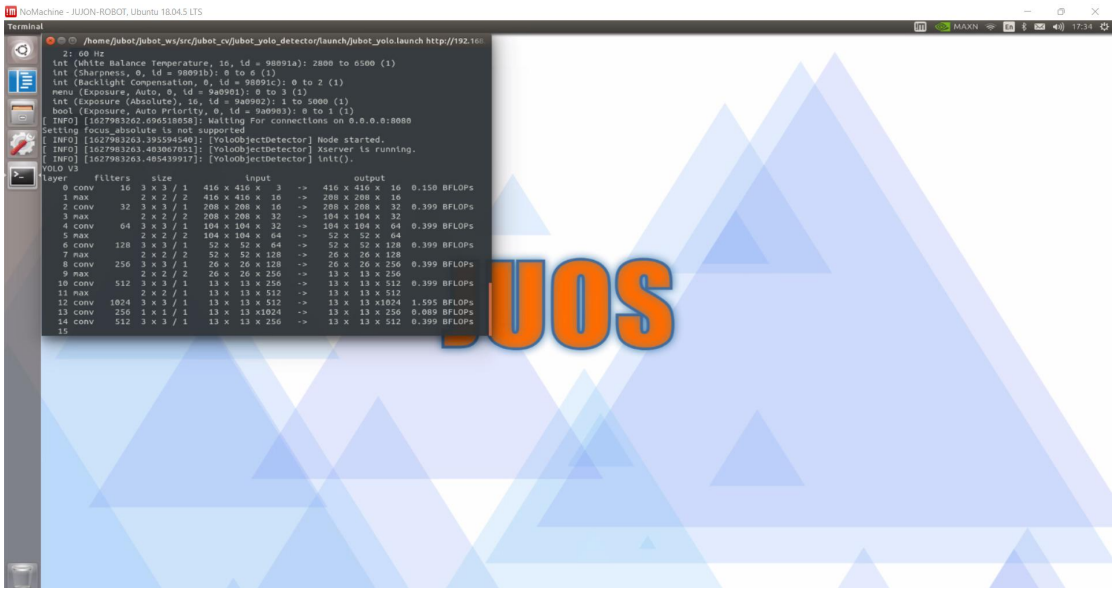
在路威套件中，提供了在 ROS 框架中运行 YOLO 物体识别网络的功能包，该功能

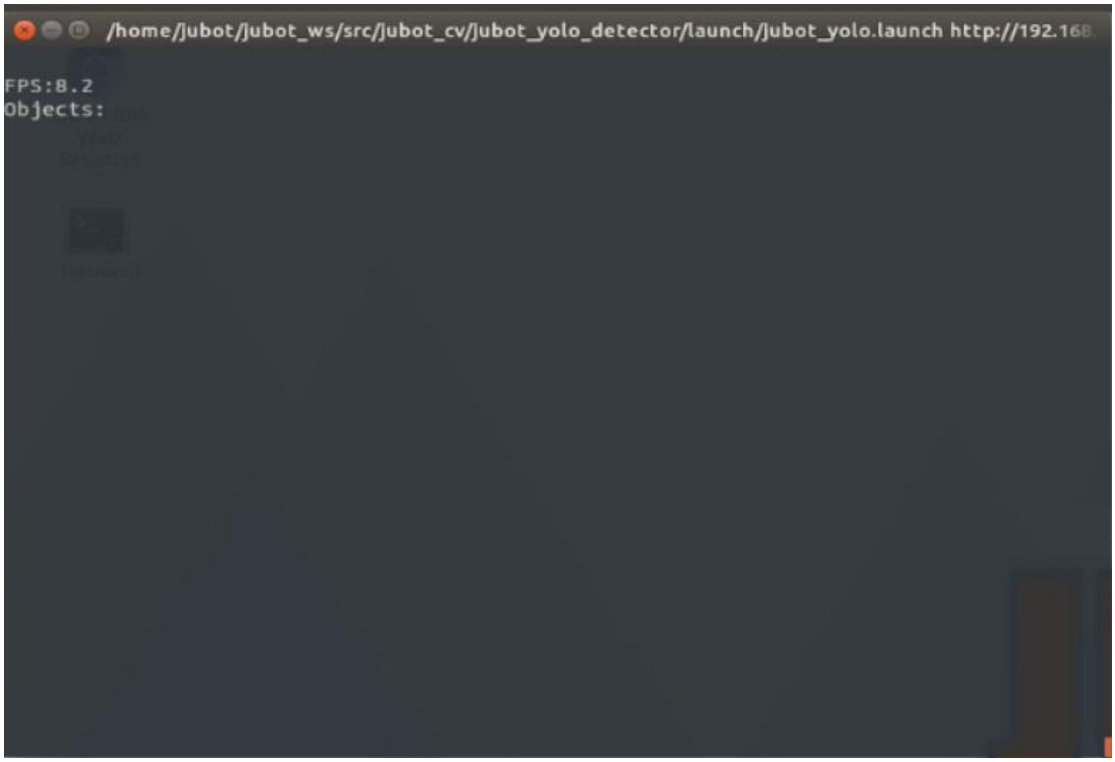
包实现了将摄像头实时图像利用 YOLO 网络进行物体识别的功能，并将识别到的物体通过 ROS 话题发布，可以非常方便的进行功能的开发及移植，赋予用户自己应用物体识别的能力。该功能包位于机器人端~/jubot_ws/src/jubot_cv/jubot_yolo_detector/路径下。

使用方法如下：

```
roslaunch jubot_yolo_detector jubot_yolo.launch
```

通过 SSH 连接到机器人端，在终端中输入如下命令运行 YOLO 物体识别功能包：

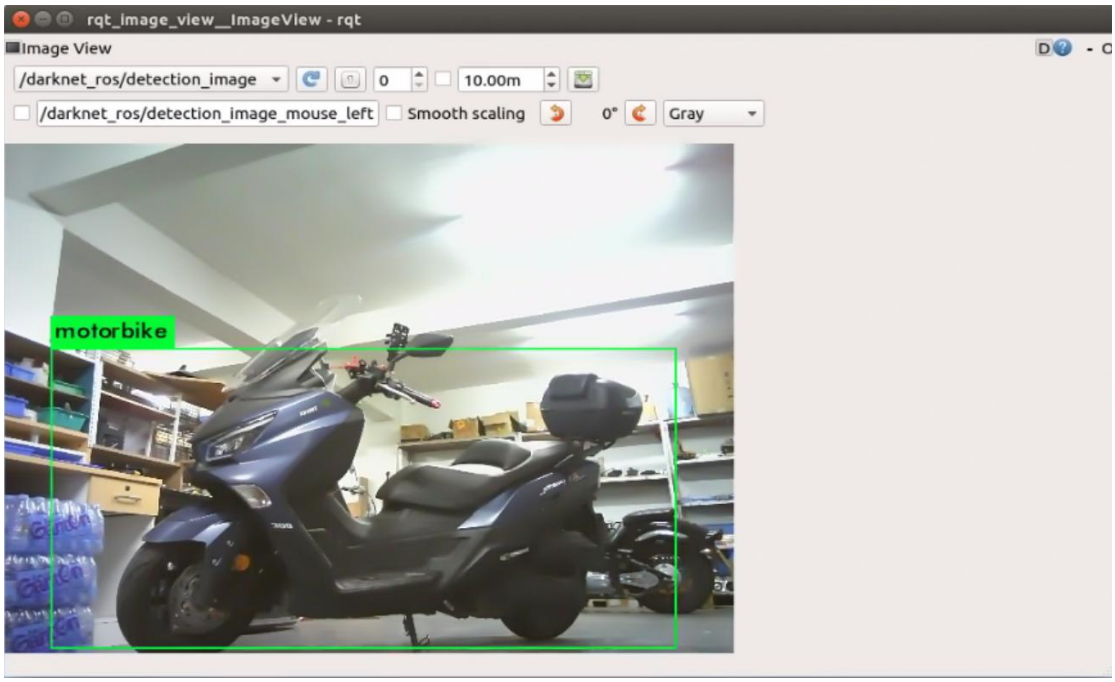


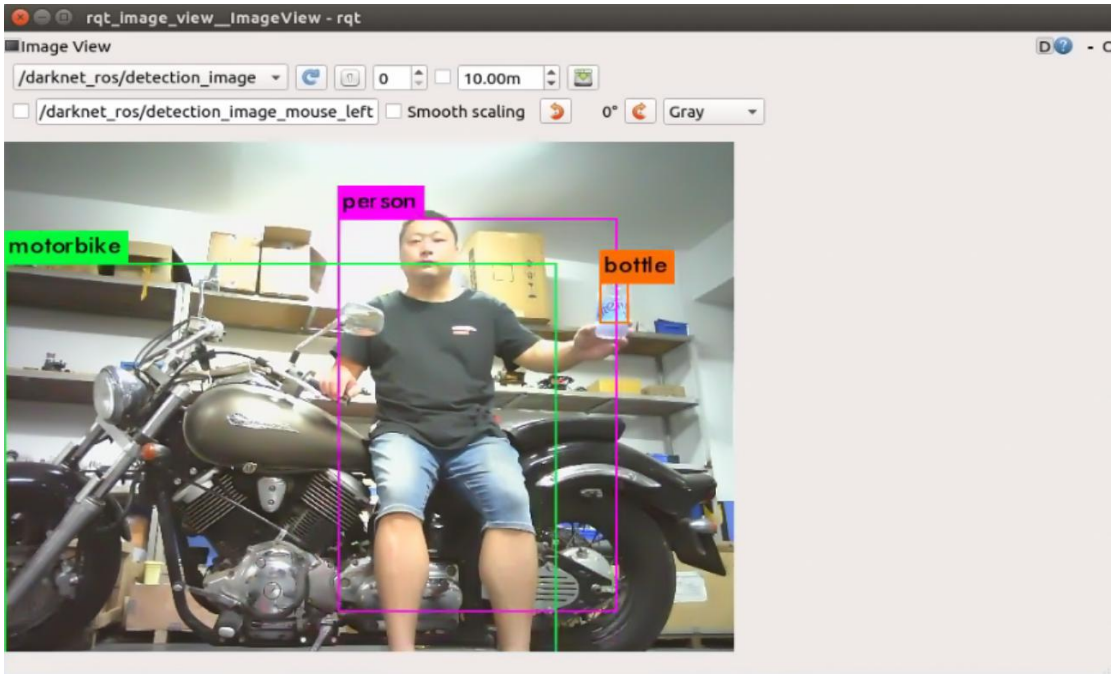


再在虚拟机端打开一终端，输入如下命令，启动 rqt_image_viewer 工具，显示实时识别图像：

```
rosrun rqt_image_view rqt_image_view
```

选择图像话题/darknet_ros/detection_image，画面中即可实时显示当前识别的图像：




除了通过终端提示和图像来获取识别结果以外，也可通过 ROS 话题来获取识别结果，以此可以非常方便的将识别功能集成用户自己的二次开发的需求中，查看话题消息

方法如下：

通过 SSH 连接到机器人端，在终端中利用 rostopic 命令获取消息内容：

```
rostopic echo /darknet_ros/bounding_boxes
```

数据如下：

```
  frame_id: "detection"
image_header:
  seq: 1271
  stamp:
    secs: 1627986876
    nsecs: 697787022
  frame_id: "camera_link"
bounding_boxes:
  -
    probability: 0.395895391703
    xmin: 273
    ymin: 408
    xmax: 444
    ymax: 480
    id: 0
    Class: "person"
  -
    probability: 0.388960957527
    xmin: 316
    ymin: 419
    xmax: 398
    ymax: 469
    id: 0
    Class: "person"
---
header:
  seq: 77
  stamp:
    secs: 1627986881
    nsecs: 499999194
  frame_id: "detection"
image_header:
  seq: 1384
  stamp:
    secs: 1627986881
    nsecs: 217565817
  frame_id: "camera_link"
bounding_boxes:
```

可以看到，节点将识别到的物体的概率、在图像中的位置以及物体名称等信息都发布到了话题中，用户也可在程序中订阅该话题，来获取物体识别的信息。

### 22.4.3 更换 YOLO 模型

在前一小节所述功能包中，默认启动 YOLOV3 网络模型，路威套件内置了四种网络模型，分别是 YOLOV2、YOLOV2-Tiny、YOLOV3、YOLOV3-Tiny。其中，

Tiny 模型是YOLO 的简化模型，运行速度较快，但相应的识别准确度有一定程度上的下降，经测试，在 Jetson Nano 上，YOLOV2 与 YOLOV3 识别帧率仅为 1-3 帧每秒，而YOLOV2-Tiny 以及 YOLOV3-Tiny 可达到 10 帧每秒。

用户可通过如下方法，更换功能包启动的网络模型：

通过 SSH 连接到机器人端，通过如下命令定位到 jubot_yolo_detector 功能包的启动文件： `roscd jubot_yolo_detector/launch`

然后，利用 vi 编辑器编辑该路径下的 jubot_yolo.launch 文件：

```
^C[jubot@JUJON-ROBOT ~]$roscd jubot_yolo_detector/launch
[jubot@JUJON-ROBOT ~/jubot_ws/src/jubot_cv/jubot_yolo_detector/launch]$ls
jubot_yolo.launch
[jubot@JUJON-ROBOT ~/jubot_ws/src/jubot_cv/jubot_yolo_detector/launch]$pwd
/home/jubot/jubot_ws/src/jubot_cv/jubot_yolo_detector/launch
[jubot@JUJON-ROBOT ~/jubot_ws/src/jubot_cv/jubot_yolo_detector/launch]$
```

```
<?xml version="1.0" encoding="utf-8"?>

<launch>
  <!-- Console launch prefix -->
  <arg name="launch_prefix" default=""/>
  <arg name="image" default="/camera/image_raw" />

  <!-- Config and weights folder. -->
  <arg name="yolo_weights_path"          default="$(find darknet_ros)/yolo_network_config/weights"/>
  <arg name="yolo_config_path"           default="$(find darknet_ros)/yolo_network_config/cfg"/>

  <!-- ROS and network parameter files -->
  <arg name="ros_param_file"             default="$(find jubot_yolo_detector)/config/ros.yaml"/>

  <arg name="network_param_file"         default="$(find jubot_yolo_detector)/config/yolov3-tiny.yaml"/>

  <!-- Load parameters -->
  <rosparam command="load" ns="darknet_ros" file="$(arg ros_param_file)"/>
  <rosparam command="load" ns="darknet_ros" file="$(arg network_param_file)"/>

  <!-- Start darknet and ros wrapper -->

  <include file="$(find jubot_driver)/launch/jubot_camera.launch"/>
  <node pkg="darknet_ros" type="darknet_ros" name="darknet_ros" output="screen" launch-prefix="$(arg launch_prefix)">
    <param name="weights_path"          value="$(arg yolo_weights_path)" />
    <param name="config_path"           value="$(arg yolo_config_path)" />
    <remap from="camera/rgb/image_raw"  to="$(arg image)" />
  </node>

</launch>
~
```

红框所框，即为功能包导入的 YOLO 模型文件，通过更改此处的文件名，并重新运行该功能包，即可调用不同的YOLO 模型文件，以实现不同场景下的最优的识别效果，可选的文件名如下：

- yolov3.yaml YOLO-V3 模型，识别率较高，帧率 1-3 帧
- yolov3-tiny.yaml YOLO-V3Tiny 模型，识别速度较高，帧率 10
- 帧左右。
- yolov2.yaml YOLO-V2 模型，帧率 1-3 帧。
- yolov2-tiny.yaml YOLO-V2Tiny 模型，帧率 10 帧左右。

更改想要调用的文件名并保存退出后，重新运行功能节点，即可使用所指定的 YOLO 模型。

## 22.5 实验结果

能够使用 Yolo 深度学习算法实现目标识别。

## 22.6 实验报告

实验目的

实验要求

实验内容

实验总结

# 第二十三章 训练自己的卷积神经网络框架(YOLO)

## 23.1 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。

## 23.2 实验要求

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

## 23.3 实验工具

个人电脑一台，路威套件。

## 23.4 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。

### 23.4.1 训练自己的卷积神经网络框架

能够使用 Yolo 深度学习算法训练自己的卷积神经网络框架。

路威套件中布署了 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 18.04 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 18.04 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-reu
sed 5901
Receiving objects: 100% (5901/5901), 6.16 MiB | 96.00 Ki
B/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
cd ~/Yolo_mark
tree -L 1
```

同样的，我们介绍下 Yolo_mark 的工程目录:

```
.
├── 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
```

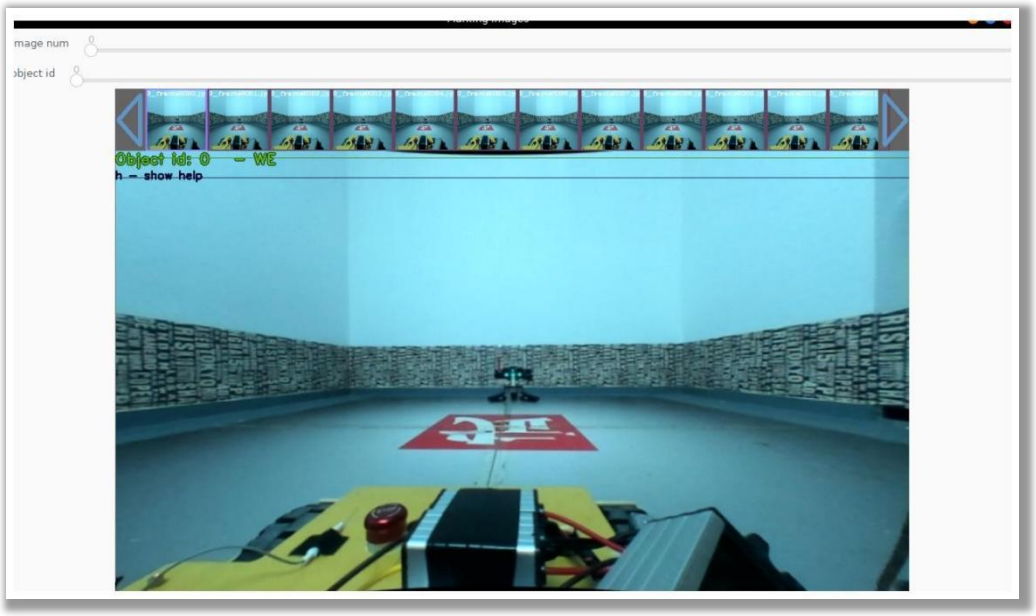
再把之前准备好的数据集拷贝到/home/${USER_NAME}/darknet/data/coco/images文件夹下。第二个参数是在运行标注软件后会在该文件里自动生成文件索引，不需要手动设置内容。 接下来我们设置目标类别文件，我们输入：

```
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 d
ata/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/${USER_NAME}/darknet/cfg/yolov3-tiny.cfg yolov3-t
iny-train.cfg
    gedit /home/${USER_NAME}/darknet/cfg/yolov3-tiny-trai
n.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-tr
ain.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 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
oading weights from yolov3-tiny.conv.15...Done!
earning Rate: 0.001, Momentum: 0.9, Decay: 0.0005
esizing
44
oaded: 0.298004 seconds
egion 16 Avg IOU: 0.154911, Class: 0.542475, Obj: 0.385543, No Obj: 0.507530, .5R: 0.000000, .75R: 0
000000,  count: 9
egion 23 Avg IOU: 0.196190, Class: 0.678789, Obj: 0.582617, No Obj: 0.492592, .5R: 0.000000, .75R: 0
000000,  count: 4
egion 16 Avg IOU: 0.406785, Class: 0.456425, Obj: 0.330342, No Obj: 0.509826, .5R: 0.500000, .75R: 0
000000,  count: 4
egion 23 Avg IOU: 0.261887, Class: 0.593361, Obj: 0.530446, No Obj: 0.491866, .5R: 0.000000, .75R: 0
000000,  count: 7
egion 16 Avg IOU: 0.297758, Class: 0.340539, Obj: 0.442540, No Obj: 0.507202, .5R: 0.000000, .75R: 0
000000,  count: 7
egion 23 Avg IOU: -nan, Class: -nan, Obj: -nan, No Obj: 0.492202, .5R: -nan, .75R: -nan,  count: 0
egion 16 Avg IOU: 0.191769, Class: 0.644537, Obj: 0.251000, No Obj: 0.505752, .5R: 0.000000, .75R: 0
000000,  count: 8
egion 23 Avg IOU: 0.233697, Class: 0.597563, Obj: 0.395009, No Obj: 0.492591, .5R: 0.125000, .75R: 0
000000,  count: 8
egion 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。

### 23.4.2 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

```
roscd darknet_ros 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路威套件

~/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：

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

然后启动目标检测：

```
roslaunch 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。

## 23.5 实验结果

能够使用 Yolo 深度学习算法训练自己的数据集并实现目标识别。

## 23.6 实验报告

实验目的

实验要求

实验内容

实验总结

# 第二十四章 卷积神经网络(TensorFlow)

## 24.1 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。

## 24.2 实验要求

掌握通过 TensorFlow 算法实现目标识别。

## 24.3 实验工具

个人电脑一台，路威套件。

## 24.4 实验内容

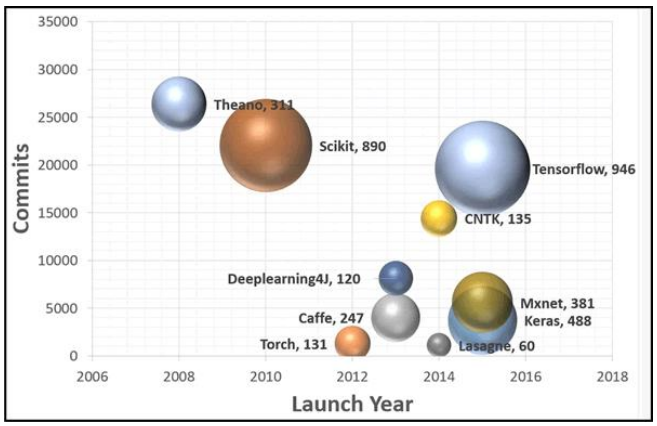
在这个案例中，我们将从神经网络数据训练与目标检测两个方面介绍。

### 24.4.1 TensorFlow 简介

TensorFlow 是一个开源的、基于 Python 的机器学习框架，它由 Google 开发，并在图形分类、音频处理、推荐系统和自然语言处理等场景下有着丰富的应用， 是目前最热门的机器学习框架。除了 Python，TensorFlow 也提供了 C/C++、Java、Go、R 等其它编程语言的接口。

TensorFlow 是由 Google Brain 团队为深度神经网络（DNN）开发的功能强大的开源软件库，于 2015 年 11 月首次发布，在 Apache 2.x 协议许可下可用。截至今天，短短的两年内，其 GitHub 库大约 845 个贡献者共提交超过 17000 次，这本身就是衡量 TensorFlow 流行度和性能的一个指标。

下图列出了当前流行的深度学习框架，从中能够清楚地看到 TensorFlow 的领先地位：



开源深度学习库 TensorFlow 允许将深度神经网络的计算部署到任意数量的 CPU 或 GPU 的服务器、PC 或移动设备上，且只利用一个 TensorFlow API。你可能会问，还有很多其他的深度学习库，如 Torch、Theano、Caffe 和 MxNet，那 TensorFlow 与其他深度学习库的区别在哪里呢？包括 TensorFlow 在内的大多数深度学习库能够自动求导、开源、支持多种 CPU/GPU、拥有预训练模型，并支持常用的 NN 架构，如递归神经网络（RNN）、卷积神经网络（CNN）和深度置信网络（DBN）。

**TensorFlow 则还有更多的特点，如下：**

- 支持所有流行语言，如 Python、C++、Java、R 和 Go。
- 可以在多种平台上工作，甚至是移动平台和分布式平台。
- 它受到所有云服务（AWS、Google 和 Azure）的支持。
- Keras——高级神经网络 API，已经与 TensorFlow 整合。
- 与 Torch/Theano 比较，TensorFlow 拥有更好的计算图表可视化。
- 允许模型部署到工业生产中，并且容易使用。
- 有非常好的社区支持。
- TensorFlow 不仅仅是一个软件库，它是一套包括 TensorFlow，TensorBoard 和 TensorServing 的软件。

**谷歌 research 博客列出了全球一些使用 TensorFlow 开发的有趣项目：**

- Google 翻译运用了 TensorFlow 和 TPU（Tensor Processing Units）。
- Project Magenta 能够使用强化学习模型生成音乐，运用了 TensorFlow。
- 澳大利亚海洋生物学家使用了 TensorFlow 来发现和理解濒临灭绝的海牛。
- 一位日本农民运用 TensorFlow 开发了一个应用程序，使用大小和形状等物理特性对黄瓜进行分类。

使用 TensorFlow 的项目还有很多。本教程旨在让用户理解 TensorFlow 在深度学习模型中的应用，使用户可以对深度神经网络模型用于数据集并开发有用的应用程序有一个基本的概念。

在路威套件中，已经安装了 TensorFlow2.0，用户可通过 Python3 直接调用 TensorFlow，并且支持 Jetson Nano GPU 加速，因此，路威套件也是一款非常好用的TensorFlow 深度学习学习平台。

### 24.4.2 手写数字识别（MNIST）例程

在 Jetson Nano 路威套件中，搭载了基于TensorFlow 的手写数字识别例程， 在

本小节中，将会手把手的教给用户，如何利用TensorFlow 搭建并训练一个手写数字识别的神经网络，并且给出了一个调用路威套件上的摄像头，利用训练好的神经网络，来识别训练集中手写数字的简单应用 Demo。

本小节所述例程，位于机器人端~/dl_ws/tensorflow/MNIST_Demo/路径下。

- MNIST_data 文件夹，存储手写数字MNIST 数据集。
- scripts 文件夹，存储例程源码。
- train_model 文件夹，存储训练模型。

接下来，让我们手把手的训练一个TensorFlow 神经网络模型！

当我们开始学习编程的时候，第一件事往往是学习打印"Hello World"。就好比编程入门有 Hello World，机器学习入门有 MNIST。

MNIST 是一个入门级的计算机视觉数据集，它包含各种手写数字图片：

它也包含每一张图片对应的标签，告诉我们这个是数字几。比如，上面这四张图片的标签分别是 5，0，4，1。

在此教程中，我们将训练一个机器学习模型用于预测图片里面的数字。我们的目的不是要设计一个世界一流的复杂模型 -- 尽管我们会在之后给你源代码去实现一流的预测模型 -- 而是要介绍下如何使用TensorFlow。所以，我们这里会从一个很简单的数学模型开始，它叫做 Softmax Regression。

对应这个教程的实现代码很短，而且真正有意思的内容只包含在三行代码里面。但是，去理解包含在这些代码里面的设计思想是非常重要的：TensorFlow 工作流程和机器学习的基本概念。因此，这个教程会很详细地介绍这些代码的实现原理。

在路威套件中，搭载的是 TensorFlow2.0，与TensorFlow1.x 的 API 接口有些许不同，同时 2.0 版本也兼容 1.x 版本的API。在本例程中，将会使用TensorFlow2.0 的 1.x 兼容模式API 来运行，1.x 版本API 更加贴近算法原理，可以更好的去学习机器学习网络的搭建过程。

### 24.2.3 使用摄像头识别手写数字

在本节教程中，将会给大家演示如何读取前一节所训练保存的神经网络模型，并使用路威套件上的摄像头实时识别所拍到的手写数字。

本节的源码文件为 scripts/test_camera.py，读取模型与训练模型的步骤雷同，在此不再进行逐行解读，其核心代码片段有。

读取并恢复网络模型：

```
def weight_variable(shape):
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)

def bias_variable(shape):
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)

def conv2d(x, W):
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max_pool(x):
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],strides=[1, 2, 2, 1], padding='SAME')

def network(x):
    x_image = tf.reshape(x, [-1,28,28,1]) #-1 means arbitrary
    h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)  #conv1
    h_pool1 = max_pool(h_conv1)                               #max_pool1

    h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)  #conv2
    h_pool2 = max_pool(h_conv2)                               #max_pool2

    h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
    h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1) #fc1

    h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)               #dropout

    y_predict=tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2) + b_fc2) #fc2 output
    return y_predict

keep_prob = tf.placeholder("float")
W_conv1 = weight_variable([5, 5, 1, 32])
b_conv1 = bias_variable([32])
W_conv2 = weight_variable([5, 5, 32, 64])
b_conv2 = bias_variable([64])
W_fc1 = weight_variable([7 * 7 * 64, 1024])
b_fc1 = bias_variable([1024])
```

读取摄像头图像，并截取 ROI 区域图像传入神经网络：

```
saver.restore(sess, "../train_model/model_save.ckpt")

cap = open_cam_usb(0,640,480)

while(1):
    ret, frame = cap.read()
    cv2.rectangle(frame,(270,200),(340,270),(0,0,255),2)
    cv2.imshow("capture", frame)
    roiImg = frame[200:270,270:340]
    img = cv2.resize(roiImg,(28,28))
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    np_img = img.astype(np.float32)

    netoutput = network(np_img)
    predictions = sess.run(netoutput,feed_dict={keep_prob: 0.5})

    predicts=predictions.tolist()

    label=predicts[0]
    result=label.index(max(label))
    print('result num:')
    print(result)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

运行方法如下：

首先，通过 VNC 远程桌面连接机器人，打开终端，将终端目录切换到

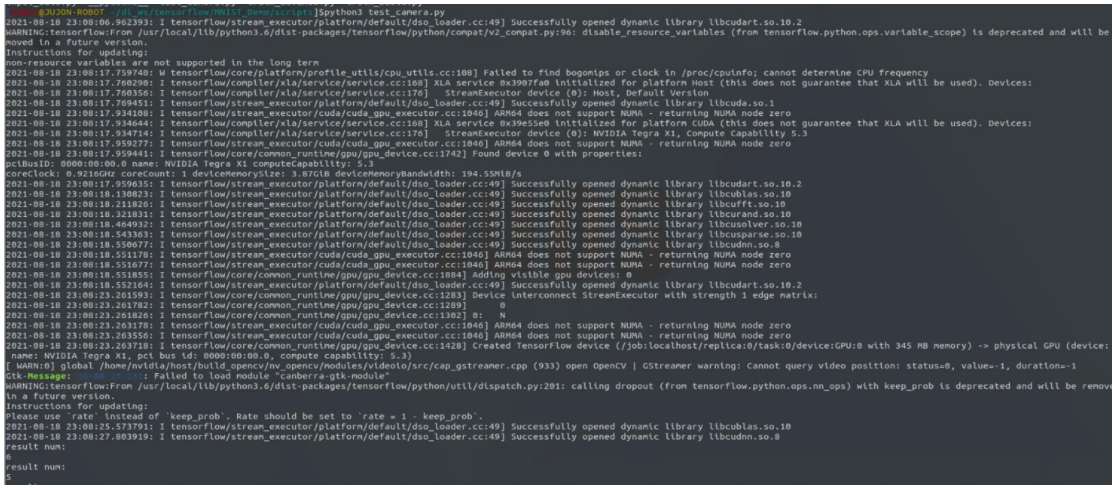
~/dl_ws/tensorflow/MNIST_Demo/scripts/

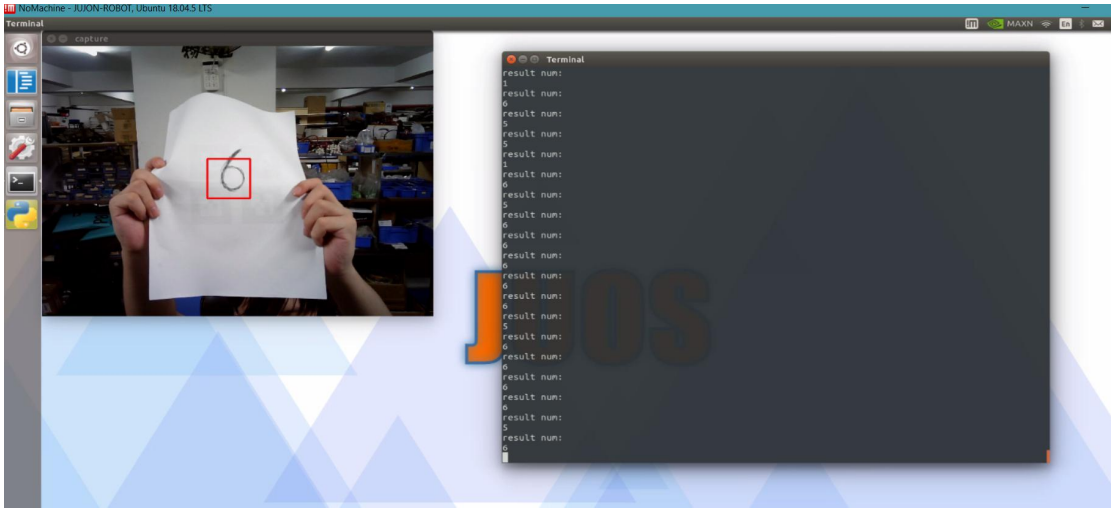
```
[jubot@JUJON-ROBOT ~/dl_ws/tensorflow/MNIST_Demo/scripts]$pwd
/home/jubot/dl_ws/tensorflow/MNIST_Demo/scripts
[jubot@JUJON-ROBOT ~/dl_ws/tensorflow/MNIST_Demo/scripts]$ls
input_data.py   __pycache__   test_camera.py  train_advance.py  train_basic.py
[jubot@JUJON-ROBOT ~/dl_ws/tensorflow/MNIST_Demo/scripts]$
```

输入以下命令，启动摄像头手写数字识别例程：

~/dl_ws/tensorflow/MNIST_Demo/scripts$python3 test_camera.py

运行效果如下：





将摄像头红框 ROI 区域对准要识别的手写数字图片，在终端窗口中，即可输出神经网络识别结果。

至此，就完成了从 0 训练一个手写数字识别神经网络，并使用摄像头运行的整套流程。用户可以根据本教程所提供的算法的简单实现，进一步学习TensorFlow，在路威套件平台上进一步开发更加完善、更加强大的功能。

## 24.5 实验结果

能够使用 TensorFlow 深度学习算法训练自己的数据集并实现目标识别。

## 24.6 实验报告

实验目的

实验要求

实验内容

实验总结

# 第二十五章 训练自己的卷积神经网络框架(TensorFlow)

## 25.1 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。

## 25.2 实验要求

掌握通过 TensorFlow 算法实现训练自己的卷积神经网络。

## 25.3 实验工具

个人电脑一台，路威套件。

## 25.4 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。

### 25.4.1 搭建手写数字识别网络

在本节教程中，我们将会使用 Python3 交互式编程来进行，所以，在开始之前， 我们首先打开机器人，并通过 SSH 指令连接到机器人端。

切换到~/dl_ws/tensorflow/MNIST_Demo/scripts 例程源码文件夹路径下：

```
[jubot@JUJON-ROBOT ~]$cd ~/dl_ws/tensorflow/MNIST_Demo/scripts
[jubot@JUJON-ROBOT ~/dl_ws/tensorflow/MNIST_Demo/scripts]$ls
input_data.py  __pycache__  test_camera.py  train_advance.py  train_basic.py
[jubot@JUJON-ROBOT ~/dl_ws/tensorflow/MNIST_Demo/scripts]$pwd
/home/jubot/dl_ws/tensorflow/MNIST_Demo/scripts
[jubot@JUJON-ROBOT ~/dl_ws/tensorflow/MNIST_Demo/scripts]$
```

通过如下命令，进入Python3 交互式编程

```
~/dl_ws/tensorflow/MNIST_Demo/scripts$ python3
```

```
[jubot@JUJON-ROBOT ~/dl_ws/tensorflow/MNIST_Demo/scripts]$python3
Python 3.6.9 (default, Oct  8 2020, 12:12:24)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

**MNIST 数据集**

首先，我们需要导入MNIST 数据集，MNIST 数据集的官网是Yann LeCun's website。在这里，我们提供了 input_data.py 文件用于自动下载和安装这个数据集，我们可以利用这个程序，将数据集导入，在程序中输入：

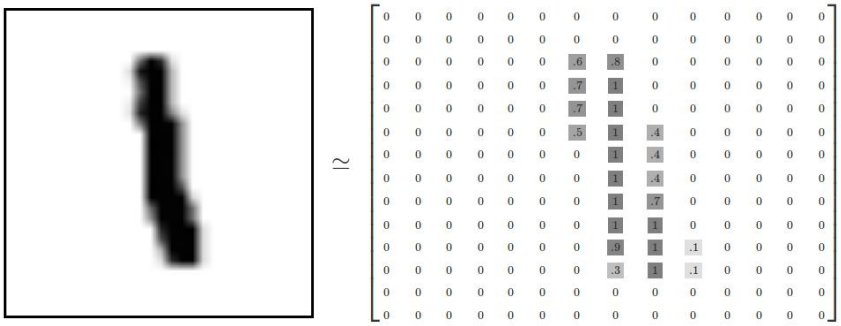
```
>>> import input_data
>>> mnist = input_data.read_data_sets("../MNIST_data/", one_hot =True)
```

```
[jubot@JUJON-ROBOT ~/dl_ws/tensorflow/MNIST_Demo/scripts]$python3
Python 3.6.9 (default, Oct  8 2020, 12:12:24)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import input_data
2021-08-18 23:23:25.843815: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library libcudart.so.10.2
>>> mnist = input_data.read_data_sets("../MNIST_data/", one_hot =True)
Extracting ../MNIST_data/train-images-idx3-ubyte.gz
Extracting ../MNIST_data/train-labels-idx1-ubyte.gz
Extracting ../MNIST_data/t10k-images-idx3-ubyte.gz
Extracting ../MNIST_data/t10k-labels-idx1-ubyte.gz
>>>
```

导入的数据集被分成两部分：60000 行的训练数据集（mnist.train）和 10000 行的测试数据集（mnist.test）。这样的切分很重要，在机器学习模型设计时必须有一个单独的测试数据集不用于训练而是用来评估这个模型的性能，从而更加容易把设计的模型推广到其他数据集上（泛化）。

正如前面提到的一样，每一个 MNIST 数据单元有两部分组成：一张包含手写数字的图片和一个对应的标签。我们把这些图片设为"**xs**"，把这些标签设为"**ys**"。训练数据集和测试数据集都包含 **xs** 和 **ys**，比如训练数据集的图片**mnist.train.images** ，训练数据集的标签是 **mnist.train.labels**。

每一张图片包含 28 像素 X28 像素。我们可以用一个数字数组来表示这张图片：

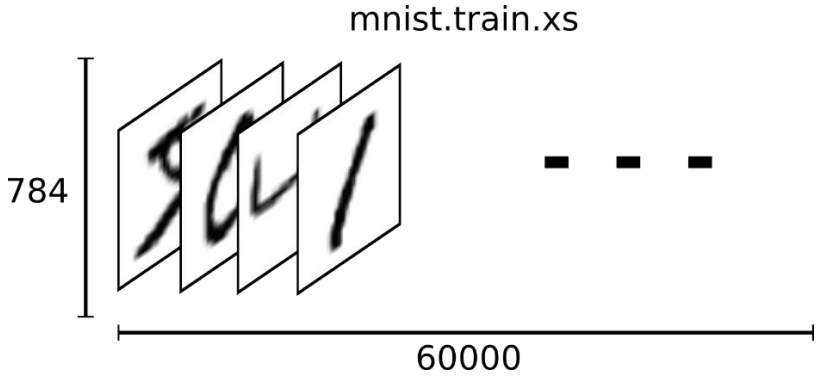


我们把这个数组展开成一个向量，长度是 28x28 = 784。如何展开这个数组（数字间的顺序）不重要，只要保持各个图片采用相同的方式展开。从这个角度来 看，MNIST 数据集的图片就是在 784 维向量空间里面的点，并且拥有比较复杂的结构（提醒：此类数据的可视化是计算密集型的）。

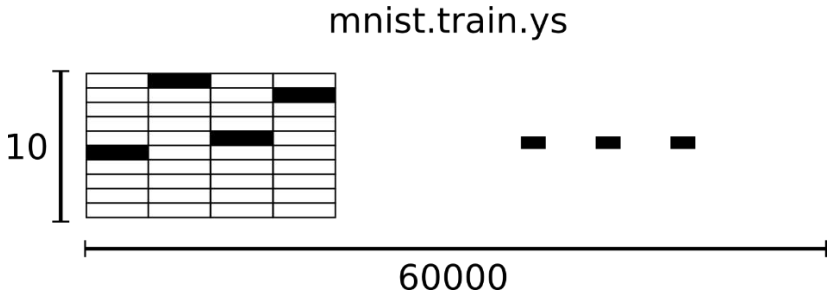
展平图片的数字数组会丢失图片的二维结构信息。这显然是不理想的，最优秀的计算机视觉方法会挖掘并利用这些结构信息，我们会在后续教程中介绍。但是在这个教程中我们忽略这些结构，所介绍的简单数学模型，softmax 回归(softmax regression)，不会利用这些结构信息。

因此，在MNIST 训练数据集中，mnist.train.images 是一个形状为 [60000,784]

的张量，第一个维度数字用来索引图片，第二个维度数字用来索引每张图片中的像素点。在此张量里的每一个元素，都表示某张图片里的某个像素的强度值，值介于 0 和 1 之间。




相对应的MNIST 数据集的标签是介于 0 到 9 的数字，用来描述给定图片里表示的数字。为了用于这个教程，我们使标签数据是"one-hot vectors"。 一个 one-hot 向量除了某一位的数字是 1 以外其余各维度数字都是 0。所以在此教程中，数字 n 将表示成一个只有在第 n 维度（从 0 开始）数字为 1 的 10 维向量。比如，标签 0 将表示成([1,0,0,0,0,0,0,0,0,0])。因此， mnist.train.labels 是一个 [60000,10] 的数字矩阵。




现在，我们准备好可以开始构建我们的模型啦！

## Softmax 回归介绍

我们知道 MNIST 的每一张图片都表示一个数字，从 0 到 9。我们希望得到给定图片代表每个数字的概率。比如说，我们的模型可能推测一张包含 9 的图片代表数字 9 的概率是 80%但是判断它是 8 的概率是 5%（因为 8 和 9 都有上半部分的小圆）， 然后给予它代表其他数字的概率更小的值。

这是一个使用 softmax 回归（softmax regression）模型的经典案例。softmax 模型可以用来给不同的对象分配概率。即使在之后，我们训练更加精细的模型时，最后一

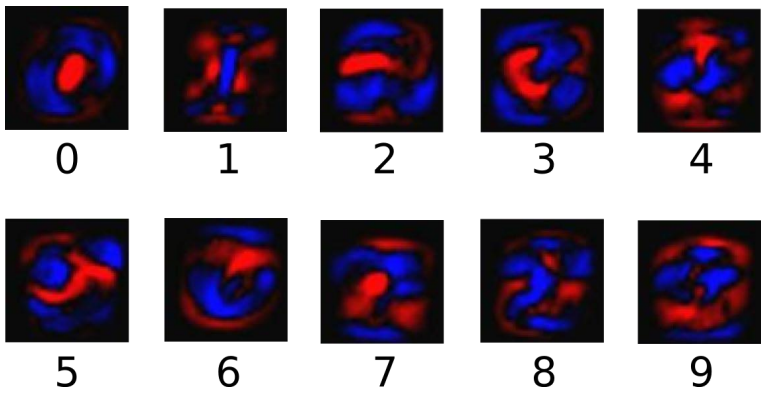
步也需要用 softmax 来分配概率。

softmax 回归（softmax regression）分两步：

**第一步**

为了得到一张给定图片属于某个特定数字类的证据（evidence），我们对图片像素值进行加权求和。如果这个像素具有很强的证据说明这张图片不属于该类，那么相应的权值为负数，相反如果这个像素拥有有利的证据支持这张图片属于这个类，那么权值是正数。

下面的图片显示了一个模型学习到的图片上每个像素对于特定数字类的权值。红色代表负数权值，蓝色代表正数权值。




我们也需要加入一个额外的偏置量（bias），因为输入往往会带有一些无关的干扰量。因此对于给定的输入图片 x 它代表的是数字 i 的证据可以表示为

$$evidence_i = \sum_j W_{i,j} x_j + b_i$$

其中$W_i$代表权重，$b_i$代表数字 i 类的偏置量，j 代表给定图片 x 的像素索引用于像素求和。然后用 softmax 函数可以把这些证据转换成概率 y：

$$\mathrm{y} = \mathrm{softmax(evidence)}$$

这里的 softmax 可以看成是一个激励（activation）函数或者链接（link）函数，把我们定义的线性函数的输出转换成我们想要的格式，也就是关于 10 个数字类的概率分布。因此，给定一张图片，它对于每一个数字的吻合度可以被 softmax 函数转换成为一个概率值。softmax 函数可以定义为：

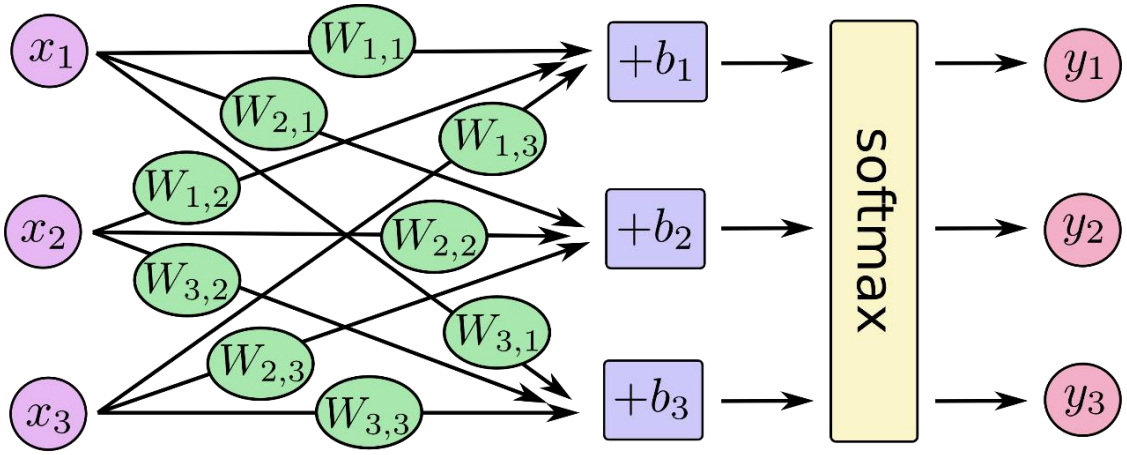
$$\mathrm{softmax(x)} = \mathrm{normalize(exp(x))}$$

展开等式右边的子式，可以得到：

$$\mathrm{softmax(x)}_i = \frac{exp(x_i)}{\sum_j exp(x_j)}$$

但是更多的时候把 softmax 模型函数定义为前一种形式：把输入值当成幂指数求值，再正则化这些结果值。这个幂运算表示，更大的证据对应更大的假设模型（hypothesis）里面的乘数权重值。反之，拥有更少的证据意味着在假设模型里面拥有更小的乘数系数。假设模型里的权值不可以是 0 值或者负值。Softmax 然后会正则化这些权重值，使它们的总和等于 1，以此构造一个有效的概率分布。（更多的关于 Softmax 函数的信息，可以参考Michael Nieslen 的书里面的这个部分，其中有关于 softmax 的可交互式的可视化解释。）

对于 softmax 回归模型可以用下面的图解释，对于输入的 xs 加权求和，再分别加上一个偏置量，最后再输入到 softmax 函数中：




如果把它写成一个等式，我们可以得到：

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left( \begin{matrix} W_{1,1}x_1 + W_{1,2}x_1 + W_{1,3}x_1 + b_1 \\ W_{2,1}x_2 + W_{2,2}x_2 + W_{2,3}x_2 + b_2 \\ W_{3,1}x_3 + W_{3,2}x_3 + W_{3,3}x_3 + b_3 \end{matrix} \right)$$

我们也可以用向量表示这个计算过程：用矩阵乘法和向量相加。这有助于提高计算效率。（也是一种更有效的思考方式）

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left( \begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

更进一步，可以写成更加紧凑的方式：

$$y = \text{softmax}(w_x + b)$$

**实现回归模型**

为了用 python 实现高效的数值计算，我们通常会使用函数库，比如 NumPy，会把类似矩阵乘法这样的复杂运算使用其他外部语言实现。不幸的是，从外部计算切换回 Python 的每一个操作，仍然是一个很大的开销。如果你用 GPU 来进行外部计算， 这样的开销会更大。用分布式的计算方式，也会花费更多的资源用来传输数据。

TensorFlow 也把复杂的计算放在 python 之外完成，但是为了避免前面说的那些开销，它做了进一步完善。Tensorflow 不单独地运行单一的复杂计算，而是让我们可以先用图描述一系列可交互的计算操作，然后全部一起在Python 之外运行。（这样类似的运行方式，可以在不少的机器学习库中看到。）

使用 TensorFlow 之前，首先导入它：

```
>>> import tensorflow.compat.v1 as tf
>>> tf.disable_v2_behavior()
```

我们通过操作符号变量来描述这些可交互的操作单元，可以用下面的方式创建一个：

```
>>> x = tf.placeholder("float", [None, 784])
```

x 不是一个特定的值，而是一个占位符 placeholder，我们在 TensorFlow 运行计算时输入这个值。我们希望能够输入任意数量的 MNIST 图像，每一张图展平成 784 维的向量。我们用 2 维的浮点数张量来表示这些图，这个张量的形状是[None，784 ]。（这里的 None 表示此张量的第一个维度可以是任何长度的。）

我们的模型也需要权重值和偏置量，当然我们可以把它们当做是另外的输入（使用占位符），但TensorFlow 有一个更好的方法来表示它们：Variable 。 一个 Variable 代表一个可修改的张量，存在在 TensorFlow 的用于描述交互性操作的图中。它们可以用于计算输入值，也可以在计算中被修改。对于各种机器学习应用，一般都会有模型参数，可以用 Variable 表示。

```
>>> W = tf.Variable(tf.zeros([784,10]))
>>> b = tf.Variable(tf.zeros([10]))
```

我们赋予 tf.Variable 不同的初值来创建不同的Variable：在这里，我们都用全为零的张量来初始化 W 和 b。因为我们要学习W 和 b 的值，它们的初值可以随意设置。

注意，W 的维度是[784，10]，因为我们想要用 784 维的图片向量乘以它以得到一

个 10 维的证据值向量，每一位对应不同数字类。b 的形状是[10]，所以我们可以直接把它加到输出上面。

现在，我们可以实现我们的模型啦。只需要一行代码！

```
>>> y = tf.nn.softmax(tf.matmul(x,W) + b)
```

首先，我们用 tf.matmul(X，W)表示 x 乘以 W，对应之前等式里面的，这里 x 是一个 2 维张量拥有多个输入。然后再加上 b，把和输入到 tf.nn.softmax 函数里面。

至此，我们先用了几行简短的代码来设置变量，然后只用了一行代码来定义我们的模型。TensorFlow 不仅仅可以使 softmax 回归模型计算变得特别简单，它也用这种非常灵活的方式来描述其他各种数值计算，从机器学习模型对物理学模拟仿真模型。一旦被定义好之后，我们的模型就可以在不同的设备上运行：计算机的 CPU，GPU，甚至是手机！

**训练模型**

为了训练我们的模型，我们首先需要定义一个指标来评估这个模型是好的。其实，在机器学习，我们通常定义指标来表示一个模型是坏的，这个指标称为成本（cost）或损失（loss），然后尽量最小化这个指标。但是，这两种方式是相同的。

一个非常常见的，非常漂亮的成本函数是"交叉熵"（cross-entropy）。交叉熵产生于信息论里面的信息压缩编码技术，但是它后来演变成为从博弈论到机器学习等其他领域里的重要技术手段。它的定义如下：

$$H_{y'}(y) = -\sum y'_i \log(y_i)$$

y 是我们预测的概率分布，y’是实际的分布（我们输入的 one-hot vector)。比较粗糙的理解是，交叉熵是用来衡量我们的预测用于描述真相的低效性。更详细的关于交叉熵的解释超出本教程的范畴，但如果要深入学习机器学习，你很有必要好好理解它。

为了计算交叉熵，我们首先需要添加一个新的占位符用于输入正确值：

```
>>> y_ = tf.placeholder("float", [None,10])
```

然后我们可以用$-\sum y'_i \log(y_i)$计算交叉熵:

```
>>> cross_entropy = -tf.reduce_sum(y_*tf.log(y))
```

首先，用 tf.log 计算 y 的每个元素的对数。接下来，我们把 y_ 的每一个元素和 tf.log(y_) 的对应元素相乘。最后，用 tf.reduce_sum 计算张量的所有元素的总和。（注意，这里的交叉熵不仅仅用来衡量单一的一对预测和真实值，而是所有 100 幅

图片的交叉熵的总和。对于 100 个数据点的预测表现比单一数据点的表现能更好地描述我们的模型的性能。

现在我们知道我们需要我们的模型做什么啦，用TensorFlow 来训练它是非常容易的。因为 TensorFlow 拥有一张描述你各个计算单元的图，它可以自动地使用反向传播算法(backpropagation algorithm)来有效地确定你的变量是如何影响你想要最小化的那个成本值的。然后，TensorFlow 会用你选择的优化算法来不断地修改变量以降低成本。

```
>>>train_step=tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)
```

在这里，我们要求 TensorFlow 用梯度下降算法（gradient descent algorithm）以 0.01 的学习速率最小化交叉熵。梯度下降算法（gradient descent algorithm）是一个简单的学习过程，TensorFlow 只需将每个变量一点点地往使成本不断降低的方向移动。当然 TensorFlow 也提供了其他许多优化算法：只要简单地调整一行代码就可以使用其他的算法。

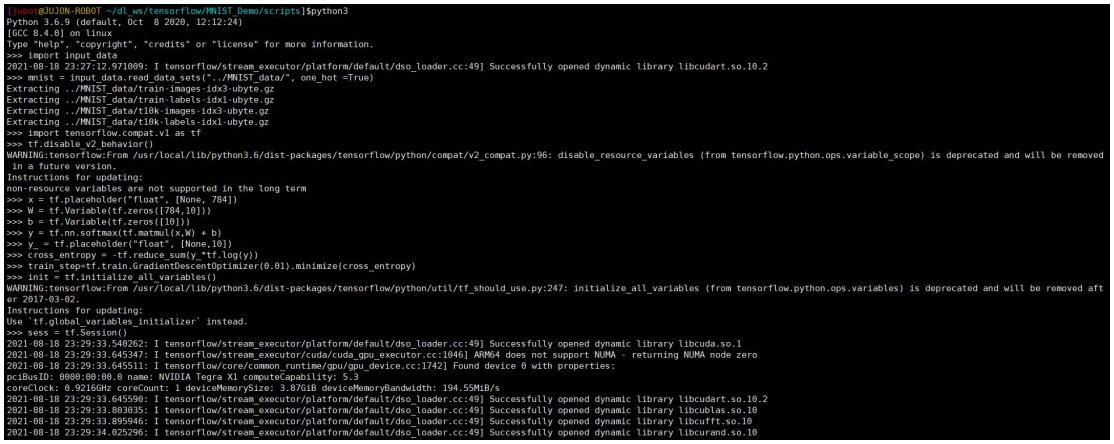
TensorFlow 在这里实际上所做的是，它会在后台给描述你的计算的那张图里面增加一系列新的计算操作单元用于实现反向传播算法和梯度下降算法。然后，它返回给你的只是一个单一的操作，当运行这个操作时，它用梯度下降算法训练你的模型， 微调你的变量，不断减少成本。

现在，我们已经设置好了我们的模型。在运行计算之前，我们需要添加一个操作来初始化我们创建的变量：

```
>>>init = tf.initialize_all_variables()
```

现在我们可以在一个 Session 里面启动我们的模型，并且初始化变量：

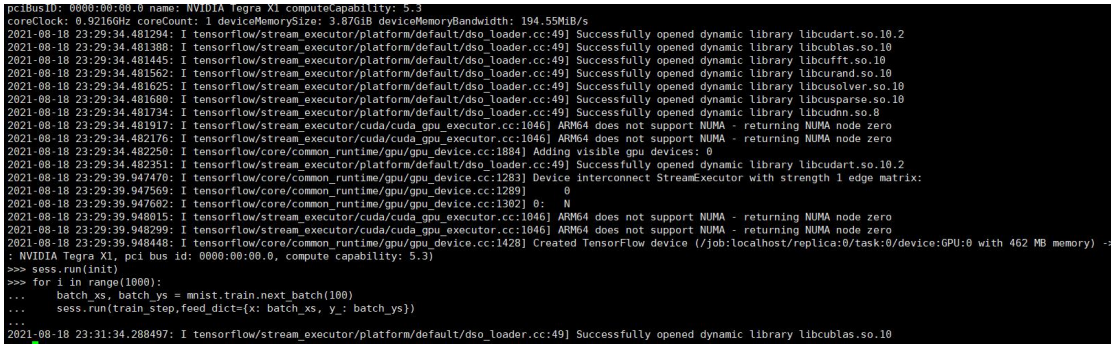
```
>>> sess = tf.Session()
```




```
>>> sess.run(init)
```

然后开始训练模型，这里我们让模型循环训练 1000 次！

```
>>> for i in range(1000):
>>>     batch_xs, batch_ys = mnist.train.next_batch(100)
>>>     sess.run(train_step,feed_dict={x: batch_xs, y_: batch_ys})
```




该循环的每个步骤中，我们都会随机抓取训练数据中的 100 个批处理数据点，然后我们用这些数据点作为参数替换之前的占位符来运行 train_step。

使用一小部分的随机数据来进行训练被称为随机训练（stochastic training）- 在这里更确切的说是随机梯度下降训练。在理想情况下，我们希望用我们所有的数据来进行每一步的训练，因为这能给我们更好的训练结果，但显然这需要很大的计算开销。所以，每一次训练我们可以使用不同的数据子集，这样做既可以减少计算开销， 又可以最大化地学习到数据集的总体特性。

**评估我们的模型**

那么我们的模型性能如何呢？

首先让我们找出那些预测正确的标签。tf.argmax 是一个非常有用的函数，它能给出某个 tensor 对象在某一维上的其数据最大值所在的索引值。由于标签向量是由

0,1 组成，因此最大值 1 所在的索引位置就是类别标签，比如 tf.argmax(y,1)返回的是模型对于任一输入 x 预测到的标签值，而 tf.argmax(y_,1) 代表正确的标签，我们可以用 tf.equal 来检测我们的预测是否真实标签匹配(索引位置一样表示匹配)。

```
>>>correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
```

这行代码会给我们一组布尔值。为了确定正确预测项的比例，我们可以把布尔值转换成浮点数，然后取平均值。例如，[True, False, True, True] 会变成[1,0,1,1] ，取平均值后得到 0.75。

```
>>>accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
```

最后，我们计算所学习到的模型在测试数据集上面的正确率。

```
>>>print(sess.run(accuracy,feed_dict={x:mnist.test.images,y_:mnist.test.labels}))
```

```
: NVIDIA Tegra X1, pci bus id: 0000:00:00.0, compute capability: 5.3)
>>> sess.run(init)
>>> for i in range(1000):
...     batch_xs, batch_ys = mnist.train.next_batch(100)
...     sess.run(train_step,feed_dict={x: batch_xs, y_: batch_ys})
...
2021-08-18 23:31:34.288497: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library libcublas.so.10
>>> correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
>>> accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
>>> print(sess.run(accuracy,feed_dict={x:mnist.test.images,y_:mnist.test.labels}))
0.9174
>>>
```

这个最终结果值应该大约是 90%。

这个结果好吗？并不太好。事实上，这个结果是很差的。这是因为我们仅仅使用了一个非常简单的模型。不过，做一些小小的改进，我们就可以得到 97%的正确率。最好的模型甚至可以获得超过 99.7%的准确率！（想了解更多信息，可以看看这个关于各种模型的性能对比列表。）

比结果更重要的是，我们从这个模型中学习到的设计思想。不过，如果你仍然对这里的结果有点失望，可以查看下一个教程，在那里你可以学习如何用TensorFlow 构建更加复杂的模型以获得更好的性能！

本节教程的完整代码，位于 scripts/train_basic.py，用户也可直接执行该代码文件来运行。

```
~/dl_ws/tensorflow/MNIST_Demo/scripts$ python3 train_basic.py
```

```
2021-08-18 23:29:34.481917: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:1046] ARM64 does not support NUMA - returning NUMA node zer
2021-08-18 23:29:34.482176: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:1046] ARM64 does not support NUMA - returning NUMA node zer
2021-08-18 23:29:34.482250: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1884] Adding visible gpu devices: 0
2021-08-18 23:29:34.482351: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library libcudart.so.1
2021-08-18 23:29:39.947470: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1283] Device interconnect StreamExecutor with strength 1 edge m
2021-08-18 23:29:39.947569: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1289]      0
2021-08-18 23:29:39.947602: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1302] 0:   N
2021-08-18 23:29:39.948015: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:1046] ARM64 does not support NUMA - returning NUMA node zer
2021-08-18 23:29:39.948299: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:1046] ARM64 does not support NUMA - returning NUMA node zer
2021-08-18 23:29:39.948448: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1428] Created TensorFlow device (/job:localhost/replica:0/task:
: NVIDIA Tegra X1, pci bus id: 0000:00:00.0, compute capability: 5.3)
>>> sess.run(init)
>>> for i in range(1000):
...     batch_xs, batch_ys = mnist.train.next_batch(100)
...     sess.run(train_step,feed_dict={x: batch_xs, y_: batch_ys})
...
2021-08-18 23:31:34.288497: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library libcublas.so.1
>>> correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
>>> accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
>>> print(sess.run(accuracy,feed_dict={x:mnist.test.images,y_:mnist.test.labels}))
0.9174
>>> exit()
[jubot@JUJON-ROBOT ~/dl_ws/tensorflow/MNIST_Demo/scripts]$ls
input_data.py  __pycache__  test_camera.py  train_advance.py  train_basic.py
[jubot@JUJON-ROBOT ~/dl_ws/tensorflow/MNIST_Demo/scripts]$
```

## 25.4.2 搭建一个进阶的手写数字识别网络

在本节教程中，我们将在上一节的基础上，进一步搭建一个更深层次的深度卷积神经网络用来进行手写数字识别。

首先，进入 Python3 交互式编程界面。

切换到~/dl_ws/tensorflow/MNIST_Demo/scripts 例程源码文件夹路径下：

```
[jubot@JUJON-ROBOT ~/dl_ws/tensorflow/MNIST_Demo/scripts]$pwd
/home/jubot/dl_ws/tensorflow/MNIST_Demo/scripts
[jubot@JUJON-ROBOT ~/dl_ws/tensorflow/MNIST_Demo/scripts]$ls
input_data.py  __pycache__  test_camera.py  train_advance.py  train_basic.py
[jubot@JUJON-ROBOT ~/dl_ws/tensorflow/MNIST_Demo/scripts]$
```

通过如下命令，进入Python3 交互式编程

```
~/dl_ws/tensorflow/MNIST_Demo/scripts$ python3
```

```
[jubot@JUJON-ROBOT ~/dl_ws/tensorflow/MNIST_Demo/scripts]$python3
Python 3.6.9 (default, Oct  8 2020, 12:12:24)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

导入 MNIST 的数据集：

```
>>> import input_data
```

```
>>> mnist = input_data.read_data_sets("../MNIST_data/", one_hot =True)
```

以及 TensorFlow 库：

```
>>> import tensorflow.compat.v1 as tf
```

```
>>> tf.disable_v2_behavior()
```

```
[jubot@JUJON-ROBOT ~/dl_ws/tensorflow/MNIST_Demo/scripts]$python3
Python 3.6.9 (default, Oct  8 2020, 12:12:24)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import input_data
2021-08-18 23:35:01.331181: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library libcudart.so.10.2
>>> mnist = input_data.read_data_sets("../MNIST_data/", one_hot =True)
Extracting ../MNIST_data/train-images-idx3-ubyte.gz
Extracting ../MNIST_data/train-labels-idx1-ubyte.gz
Extracting ../MNIST_data/t10k-images-idx3-ubyte.gz
Extracting ../MNIST_data/t10k-labels-idx1-ubyte.gz
>>> import tensorflow.compat.v1 as tf
>>> tf.disable_v2_behavior()
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/compat/v2_compat.py:96: disable_resource_variables (from tensorflow.python.ops.va
 in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
>>>
```

## 运行 TensorFlow 的 InteractiveSession

Tensorflow 依赖于一个高效的 C++后端来进行计算。与后端的这个连接叫做 session。一般而言，使用 TensorFlow 程序的流程是先创建一个图，然后在 session 中启动它。

这里，我们使用更加方便的 InteractiveSession 类。通过它，你可以更加灵活地构建你的代码。它能让你在运行图的时候，插入一些计算图，这些计算图是由某些操作(operations)构成的。这对于工作在交互式环境中的人们来说非常便利，比如使用 IPython。如果你没有使用 InteractiveSession，那么你需要在启动 session 之前构建整个计算图，然后启动该计算图。

```
>>> sess = tf.InteractiveSession()
```

```
>>> tf.disable_v2_behavior()
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/compat/v2_compat.py:96: disable_resource_variables (from tensorflow.python.ops.variable_scope)
 in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
>>> sess = tf.InteractiveSession()
2021-08-18 23:36:51.991390: W tensorflow/core/platform/profile_utils/cpu_utils.cc:108] Failed to find bogomips or clock in /proc/cpuinfo; cannot determine CPU frequency
2021-08-18 23:36:51.991885: I tensorflow/compiler/xla/service/service.cc:168] XLA service 0x27f3dbe0 initialized for platform Host (this does not guarantee that XLA will be us
2021-08-18 23:36:51.991954: I tensorflow/compiler/xla/service/service.cc:176]   StreamExecutor device (0): Host, Default Version
2021-08-18 23:36:52.045178: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library libcuda.so.1
2021-08-18 23:36:52.130089: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:1046] ARM64 does not support NUMA - returning NUMA node zero
2021-08-18 23:36:52.130450: I tensorflow/compiler/xla/service/service.cc:168] XLA service 0x26f277f0 initialized for platform CUDA (this does not guarantee that XLA will be us
2021-08-18 23:36:52.130512: I tensorflow/compiler/xla/service/service.cc:176]   StreamExecutor device (0): NVIDIA Tegra X1, Compute Capability 5.3
2021-08-18 23:36:52.130923: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:1046] ARM64 does not support NUMA - returning NUMA node zero
2021-08-18 23:36:52.131043: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1742] Found device 0 with properties:
pciBusID: 0000:00:00.0 name: NVIDIA Tegra X1 computeCapability: 5.3
coreClock: 0.9216GHz coreCount: 1 deviceMemorySize: 3.87GiB deviceMemoryBandwidth: 194.55MiB/s
2021-08-18 23:36:52.131122: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library libcudart.so.10.2
2021-08-18 23:36:52.154076: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library libcublas.so.10
2021-08-18 23:36:52.172510: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library libcufft.so.10
2021-08-18 23:36:52.211039: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library libcurand.so.10
2021-08-18 23:36:52.221476: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library libcusolver.so.10
2021-08-18 23:36:52.225905: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library libcusparse.so.10
2021-08-18 23:36:52.226585: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library libcudnn.so.8
2021-08-18 23:36:52.226916: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:1046] ARM64 does not support NUMA - returning NUMA node zero
2021-08-18 23:36:52.227170: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:1046] ARM64 does not support NUMA - returning NUMA node zero
2021-08-18 23:36:52.227246: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1884] Adding visible gpu devices: 0
2021-08-18 23:36:52.227399: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library libcudart.so.10.2
2021-08-18 23:36:55.190544: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1283] Device interconnect StreamExecutor with strength 1 edge matrix:
2021-08-18 23:36:55.190639: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1289]      0
2021-08-18 23:36:55.190676: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1302] 0:   N
2021-08-18 23:36:55.191072: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:1046] ARM64 does not support NUMA - returning NUMA node zero
2021-08-18 23:36:55.191351: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:1046] ARM64 does not support NUMA - returning NUMA node zero
2021-08-18 23:36:55.191500: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1428] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 635 MB memory
: NVIDIA Tegra X1, pci bus id: 0000:00:00.0, compute capability: 5.3)
>>>
```

## 计算图

为了在 Python 中进行高效的数值计算，我们通常会使用像 NumPy 一类的库，将一些诸如矩阵乘法的耗时操作在 Python 环境的外部来计算，这些计算通常会通过其它语言并用更为高效的代码来实现。

但遗憾的是，每一个操作切换回 Python 环境时仍需要不小的开销。如果你想在 GPU

或者分布式环境中计算时，这一开销更加可怖，这一开销主要可能是用来进行数据迁移。

TensorFlow 也是在 Python 外部完成其主要工作，但是进行了改进以避免这种开销。其并没有采用在 Python 外部独立运行某个耗时操作的方式，而是先让我们描述一个交互操作图，然后完全将其运行在 Python 外部。这与 Theano 或 Torch 的做法类似。

因此 Python 代码的目的是用来构建这个可以在外部运行的计算图，以及安排计算图的哪一部分应该被运行。详情请查看基本用法中的计算图表一节。

**占位符**

我们通过为输入图像和目标输出类别创建节点，来开始构建计算图。

```
>>> x = tf.placeholder("float", shape=[None, 784])
>>> y_ = tf.placeholder("float", shape=[None,10])
```

这里的 x 和 y 并不是特定的值，相反，他们都只是一个占位符，可以在 TensorFlow 运行某一计算时根据该占位符输入具体的值。

输入图片 x 是一个 2 维的浮点数张量。这里，分配给它的 shape 为[None, 784]，其中 784 是一张展平的 MNIST 图片的维度。None 表示其值大小不定，在这里作为第一个维度值，用以指代 batch 的大小，意即 x 的数量不定。输出类别值 y_也是一个 2 维张量，其中每一行为一个 10 维的 one-hot 向量,用于代表对应某一 MNIST 图片的类别。

虽然placeholder 的 shape 参数是可选的，但有了它，TensorFlow 能够自动捕捉因数据维度不一致导致的错误。

**构建一个多层卷积网络**

在上一小节所搭建的神经网络模型在 MNIST 上只有 90%正确率，实在太糟糕。在这个小节里，我们用一个稍微复杂的模型：卷积神经网络来改善效果。这会达到大概 99.2%的准确率。虽然不是最高，但是还是比较让人满意。

**权重初始化**

为了创建这个模型，我们需要创建大量的权重和偏置项。这个模型中的权重在初始化时应该加入少量的噪声来打破对称性以及避免 0 梯度。由于我们使用的是 ReLU 神经元，因此比较好的做法是用一个较小的正数来初始化偏置项，以避免神经元节点输出恒为 0 的问题（dead neurons）。为了不在建立模型的时候反复做初始化操作，我们定义两个函数用于初始化。

```
>>> def weight_variable(shape):
>>>     initial = tf.truncated_normal(shape,stddev=0.1)
>>>     return tf.Variable(initial)

>>> def bias_variable(shape):
```

```
>>>     initial = tf.constant(0.1, shape=shape)
>>>     return tf.Variable(initial)
```

```
>>> x = tf.placeholder("float", shape=[None, 784])
>>> y_ = tf.placeholder("float", shape=[None,10])
>>> def weight_variable(shape):
...     initial = tf.truncated_normal(shape,stddev=0.1)
...     return tf.Variable(initial)
...
>>> def bias_variable(shape):
...     initial = tf.constant(0.1, shape=shape)
...     return tf.Variable(initial)
...
>>>
```

**卷积和池化**

TensorFlow 在卷积和池化上有很强的灵活性。我们怎么处理边界？步长应该设多大？在这个实例里，我们会一直使用 vanilla 版本。我们的卷积使用 1 步长（stride size），0 边距（padding size）的模板，保证输出和输入是同一个大小。我们的池化用简单传统的 2x2 大小的模板做max pooling。为了代码更简洁，我们把这部分抽象成一个函数。

```
>>> def conv2d(x, W):
>>>     return tf.nn.conv2d(x, W, strides=[1,1,1,1],padding='SAME')
```

```
...
>>> def conv2d(x, W):
...     return tf.nn.conv2d(x, W, strides=[1,1,1,1],padding='SAME')
...
```

```
>>> def max_pool_2x2(x):
>>>     return tf.nn.max_pool(x,ksize=[1,2,2,1],strides=[1,2,2,1],padding='
SAME')
```

```
>>> def max_pool_2x2(x):
...     return tf.nn.max_pool(x,ksize=[1,2,2,1],strides=[1,2,2,1],padding='SAME')
...
>>>
```

**第一层卷积**

现在我们可以开始实现第一层了。它由一个卷积接一个max pooling 完成。卷积在每个 5x5 的patch 中算出 32 个特征。卷积的权重张量形状是[5, 5, 1, 32]，前两个维度是 patch 的大小，接着是输入的通道数目，最后是输出的通道数目。 而对于每一个输出通道都有一个对应的偏置量。

```
>>> W_conv1 = weight_variable([5, 5, 1, 32])
>>> b_conv1 = bias_variable([32])
```

为了用这一层，我们把 x 变成一个 4d 向量，其第 2、第 3 维对应图片的宽、高，最后一维代表图片的颜色通道数(因为是灰度图所以这里的通道数为 1，如果是

rgb 彩色图，则为 3)。

```
>>> x_image = tf.reshape(x, [-1,28,28,1])
```

我们把 x_image 和权值向量进行卷积，加上偏置项，然后应用 ReLU 激活函数，最后进行max pooling。

```
>>> h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
>>> h_pool1 = max_pool_2x2(h_conv1)
```

```
>>> W_conv1 = weight_variable([5, 5, 1, 32])
>>> b_conv1 = bias_variable([32])
>>> x_image = tf.reshape(x, [-1,28,28,1])
>>> h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
>>> h_pool1 = max_pool_2x2(h_conv1)
>>>
```

**第二层卷积**

为了构建一个更深的网络，我们会把几个类似的层堆叠起来。第二层中，每个 5x5 的 patch 会得到 64 个特征。

```
>>> W_conv2 = weight_variable([5, 5, 32, 64])
>>> b_conv2 = bias_variable([64])
>>> h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
>>> h_pool2 = max_pool_2x2(h_conv2)
```

```
>>> W_conv2 = weight_variable([5, 5, 32, 64])
>>> b_conv2 = bias_variable([64])
>>> h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
>>> h_pool2 = max_pool_2x2(h_conv2)
```

**密集连接层**

现在，图片尺寸减小到 7x7，我们加入一个有 1024 个神经元的全连接层，用于处理整个图片。我们把池化层输出的张量 reshape 成一些向量，乘上权重矩阵，加上偏置，然后对其使用 ReLU。

```
>>> W_fc1 = weight_variable([7 * 7 * 64, 1024])
>>> b_fc1 = bias_variable([1024])
>>> h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
>>> h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)
```

```
>>> W_fc1 = weight_variable([7 * 7 * 64, 1024])
>>> b_fc1 = bias_variable([1024])
>>> h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
>>> h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)
>>>
```

Dropout

为了减少过拟合，我们在输出层之前加入 dropout。我们用一个 placeholder 来代表一个神经元的输出在 dropout 中保持不变的概率。这样我们可以在训练过程中启用dropout，在测试过程中关闭 dropout。 TensorFlow 的tf.nn.dropout 操作除了可以屏蔽神经元的输出外，还会自动处理神经元输出值的 scale。所以用 dropout 的时候可以不用考虑scale。

```
>>> keep_prob = tf.placeholder("float")
>>> h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
```

```
>>> keep_prob = tf.placeholder("float")
>>> h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/util/dispatch.py:201: calling dropout (f
future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
>>>
```

**输出层**

最后，我们添加一个 softmax 层，就像前面的单层 softmax regression 一样。

```
>>> W_fc2 = weight_variable([1024, 10])
>>> b_fc2 = bias_variable([10])
>>> y_conv=tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)
```

```
>>> W_fc2 = weight_variable([1024, 10])
>>> b_fc2 = bias_variable([10])
>>> y_conv=tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)
>>>
```

**训练和评估模型**

这个模型的效果如何呢？

为了进行训练和评估，我们使用与之前简单的单层SoftMax 神经网络模型几乎相同的一套代码，只是我们会用更加复杂的 ADAM 优化器来做梯度最速下降，在 feed_dict 中加入额外的参数 keep_prob 来控制 dropout 比例。然后每 100 次迭代输出一次日志。

```
>>> cross_entropy = -tf.reduce_sum(y_*tf.log(y_conv))
>>> train_step = tf.train.AdamOptimizer(1e-3).minimize(cross_entropy)
>>> correct_prediction = tf.equal(tf.argmax(y_conv,1), tf.argmax(y_,1))
>>> accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
>>> sess.run(tf.initialize_all_variables())
>>> for i in range(5000):
>>> batch = mnist.train.next_batch(50)
```

```
>>> if i%100 == 0:
>>> train_accuracy = accuracy.eval(feed_dict={
>>> x:batch[0], y_: batch[1], keep_prob: 1.0})
>>> print ("step %d, training accuracy %g"%(i, train_accuracy))
>>> train_step.run(feed_dict={x: batch[0], y_: batch[1], keep_prob: 0.5})
>>>
>>>
>>> print ("test accuracy %g"%accuracy.eval(feed_dict={
>>>     x: mnist.test.images[0:200], y_: mnist.test.labels[0:200],
keep_prob:1.0}))
```

以上代码，在最终测试集上的准确率大概是 99.2%。

**保存训练好的模型文件**

当训练好一个神经网络模型文件，并想要把它用在实际应用上时，我们就要去将训练好的神经网络模型文件保存下来，这样在其他应用上，我们就可以不用再重复训练网络模型，而是直接导入现有的网络模型使用，保存的方法就是导入 TensorFlow 模型保存对象，然后保存模型，方法如下。

```
>>> saver = tf.train.Saver()
>>> saver.save(sess , "../train_model/model_save.ckpt")
```

这样，我们训练好的模型文件，就保存到了train_model 文件夹下，之后，就可以进行网络的调用。本节教程的完整代码，位于 scripts/train_advance.py，用户也可直接执行该代码文件来运行。

```
~/dl_ws/tensorflow/MNIST_Demo/scripts$ python3 train_advance.py
```

### 25.2.3 使用摄像头识别手写数字

在本节教程中，将会给大家演示如何读取前一节所训练保存的神经网络模型，并使用路威套件上的摄像头实时识别所拍到的手写数字。

本节的源码文件为 scripts/test_camera.py，读取模型与训练模型的步骤雷同，在此不再进行逐行解读，其核心代码片段有。

读取并恢复网络模型：

```
    return tf.Variable(initial)

def conv2d(x, W):
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max_pool(x):
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],strides=[1, 2, 2, 1], padding='SAME')

def network(x):
    x_image = tf.reshape(x, [-1,28,28,1]) #-1 means arbitrary
    h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)  #conv1
    h_pool1 = max_pool(h_conv1)                               #max_pool1

    h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)  #conv2
    h_pool2 = max_pool(h_conv2)                               #max_pool2

    h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
    h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1) #fc1

    h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)               #dropout

    y_predict=tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2) + b_fc2) #fc2 output
    return y_predict

keep_prob = tf.placeholder("float")
W_conv1 = weight_variable([5, 5, 1, 32])
b_conv1 = bias_variable([32])
W_conv2 = weight_variable([5, 5, 32, 64])
b_conv2 = bias_variable([64])
W_fc1 = weight_variable([7 * 7 * 64, 1024])
b_fc1 = bias_variable([1024])
W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])
sess=tf.InteractiveSession()
saver = tf.train.Saver()
saver.restore(sess, "../train_model/model_save.ckpt")

cap = open_cam_usb(0,640,480)
```

读取摄像头图像，

```
b_conv1 = bias_variable([32])
W_conv2 = weight_variable([5, 5, 32, 64])
b_conv2 = bias_variable([64])
W_fc1 = weight_variable([7 * 7 * 64, 1024])
b_fc1 = bias_variable([1024])
W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])
sess=tf.InteractiveSession()
saver = tf.train.Saver()
saver.restore(sess, "../train_model/model_save.ckpt")

cap = open_cam_usb(0,640,480)

while(1):
    ret, frame = cap.read()
    cv2.rectangle(frame,(270,200),(340,270),(0,0,255),2)
    cv2.imshow("capture", frame)
    roiImg = frame[200:270,270:340]
    img = cv2.resize(roiImg,(28,28))
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    np_img = img.astype(np.float32)

    netoutput = network(np_img)
    predictions = sess.run(netoutput,feed_dict={keep_prob: 0.5})

    predicts=predictions.tolist()

    label=predicts[0]
    result=label.index(max(label))
    print('result num:')
    print(result)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

并截取 ROI 区域图像传入神经网络：

运行方法如下：

首先，通过 VNC 远程桌面连接机器人，打开终端，将终端目录切换到

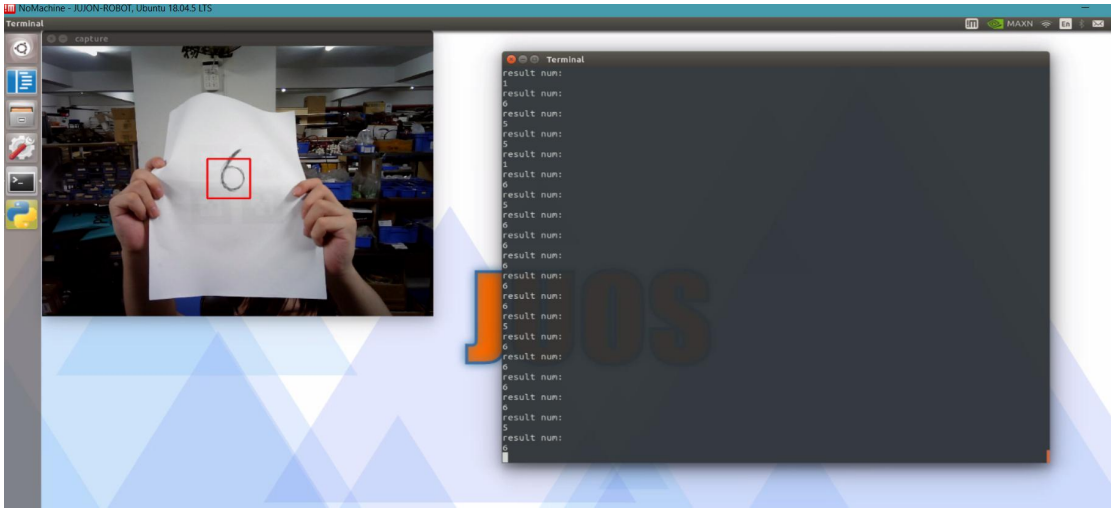
```
~/dl_ws/tensorflow/MNIST_Demo/scripts/
```

```
[jubot@JUJON-ROBOT ~/dl_ws/tensorflow/MNIST_Demo/scripts]$ls
input_data.py   __pycache__   test_camera.py  train_advance.py  train_basic.py
[jubot@JUJON-ROBOT ~/dl_ws/tensorflow/MNIST_Demo/scripts]$pwd
/home/jubot/dl_ws/tensorflow/MNIST_Demo/scripts
[jubot@JUJON-ROBOT ~/dl_ws/tensorflow/MNIST_Demo/scripts]$
```

输入以下命令，启动摄像头手写数字识别例程：

```
~/dl_ws/tensorflow/MNIST_Demo/scripts$python3 test_camera.py
```

运行效果如下：




将摄像头红框 ROI 区域对准要识别的手写数字图片，在终端窗口中，即可输出神经网络识别结果。

至此，就完成了从 0 训练一个手写数字识别神经网络，并使用摄像头运行的整套流程。用户可以根据本教程所提供的算法的简单实现，进一步学习TensorFlow，在路威套件平台上进一步开发更加完善、更加强大的功能。

## 25.5 实验结果

能够使用 TensorFlow 深度学习算法训练自己的数据集并实现目标识别。

## 25.6 实验报告

实验目的

实验要求

实验内容

实验总结

# 第二十六章 语音阵列驱动测试

## 26.1 实验目的

在 Jetson Nano 系统中测试科大讯飞6麦克风语音阵列。

## 26.2 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。

## 26.3 实验工具

个人电脑一台，路威套件。

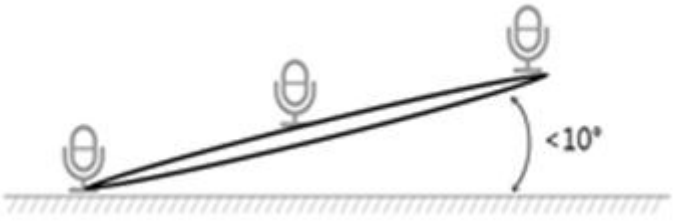
## 26.4 实验内容

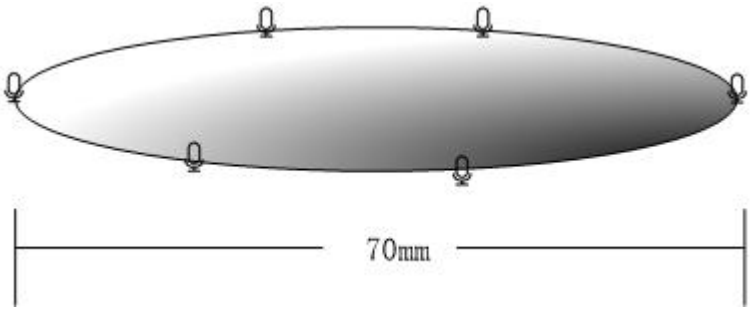
在这个案例中，我们将从麦克风驱动与阵列驱动两部分介绍。

### 26.4.1 语音阵列介绍

麦克风阵列是由一定数目的声学传感器（一般为麦克风）组成，对声场的空间特性进行采样并处理的系统。其主要作用有声源定位，抑制背景噪声、干扰、混响、回声，信号提取与分离。声源定位是指利用麦克风阵列计算声源距离阵列的角度和距离，基于TDOA（Time Difference Of Arrival，到达时间差）实现对目标声源的跟踪；信号的提取与分离是指在期望方向上有效地形成一个波束，仅拾取波束内的信号，从而达到同时提取声源和抑制噪声的目的；此外利用麦克风阵列提供的信息基于深度神经网络可实现有效的混响去除，从而极大程度上提升了真实应用场景中语音交互的效果。

六麦环形阵列构型如下：



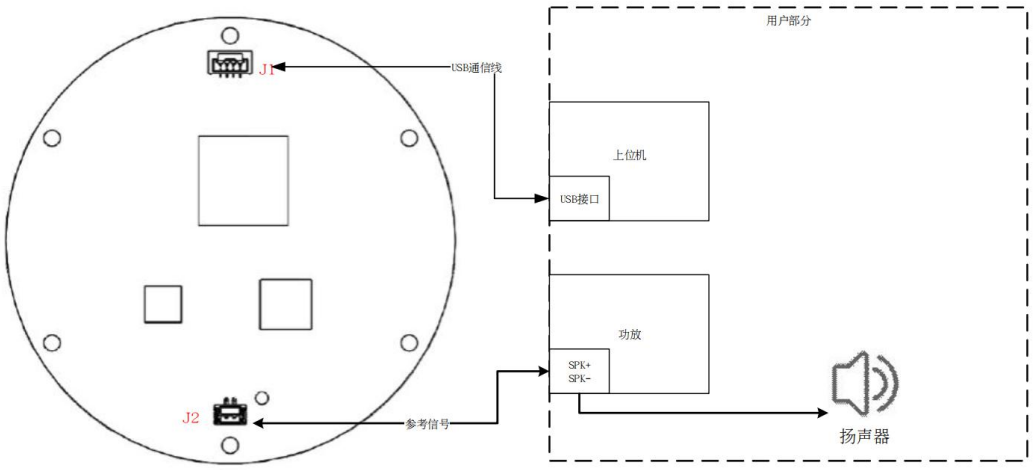



六麦环形阵列呈圆形布局，其中6个麦克风均匀分布在圆周，麦克风按圆形等距摆放，半径为35mm，如图2-1所示。圆平面和水平面之间可以有一定夹角，但夹角不能超过10°。麦克风阵列的零度方向必须和产品的正面朝向保持一致。科大讯飞麦克风阵列采用平面式分布结构，包含6个麦克风，可实现360度等效拾音，唤醒定位分辨率为1度。

麦克风正面结构如下：

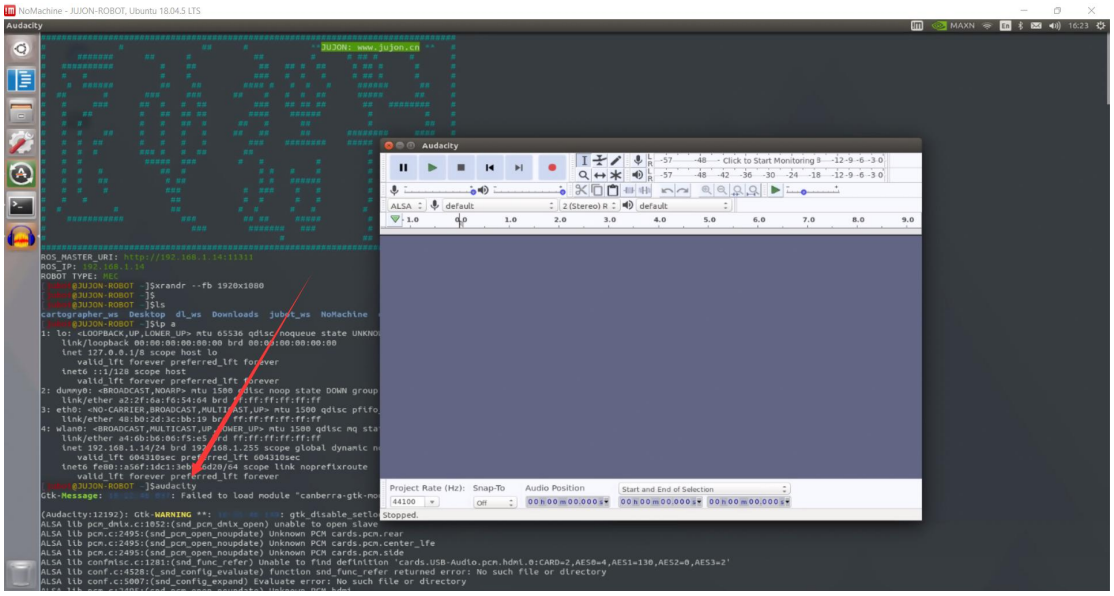

麦克风反面结构如下：


接线图如下：

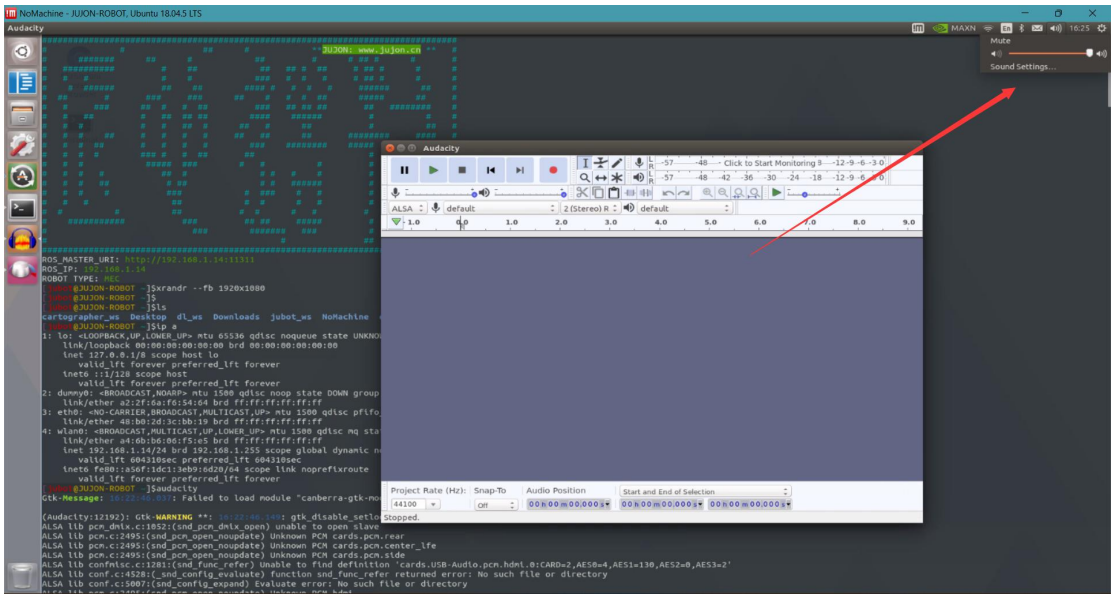



## 26.4.2 语音数据采集测试

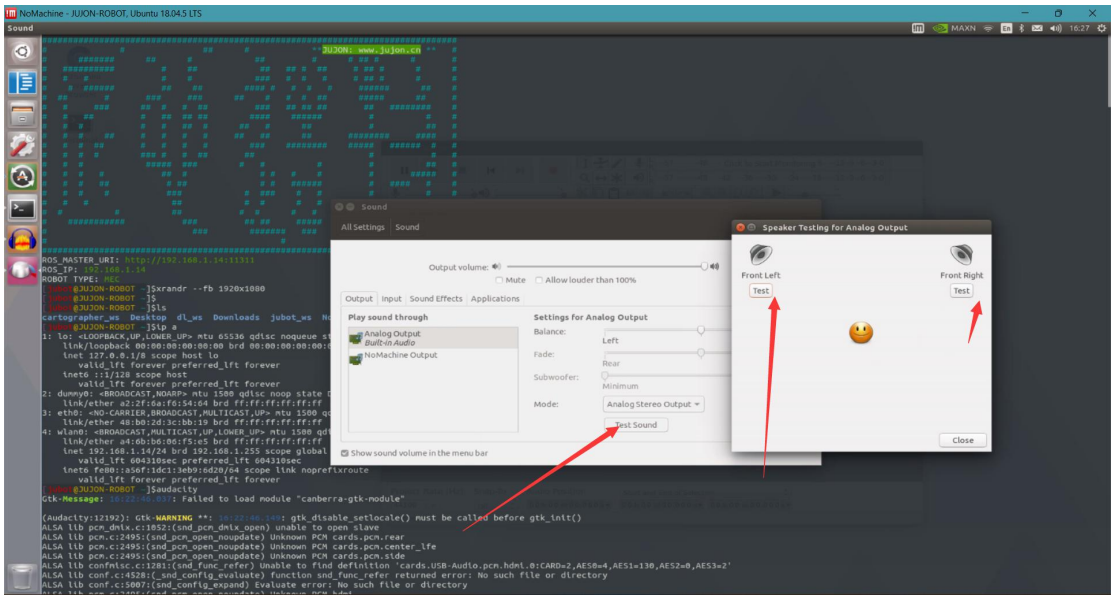
在路威套件中，我们安装了录音软件 audacity,打开终端输入 audacity。



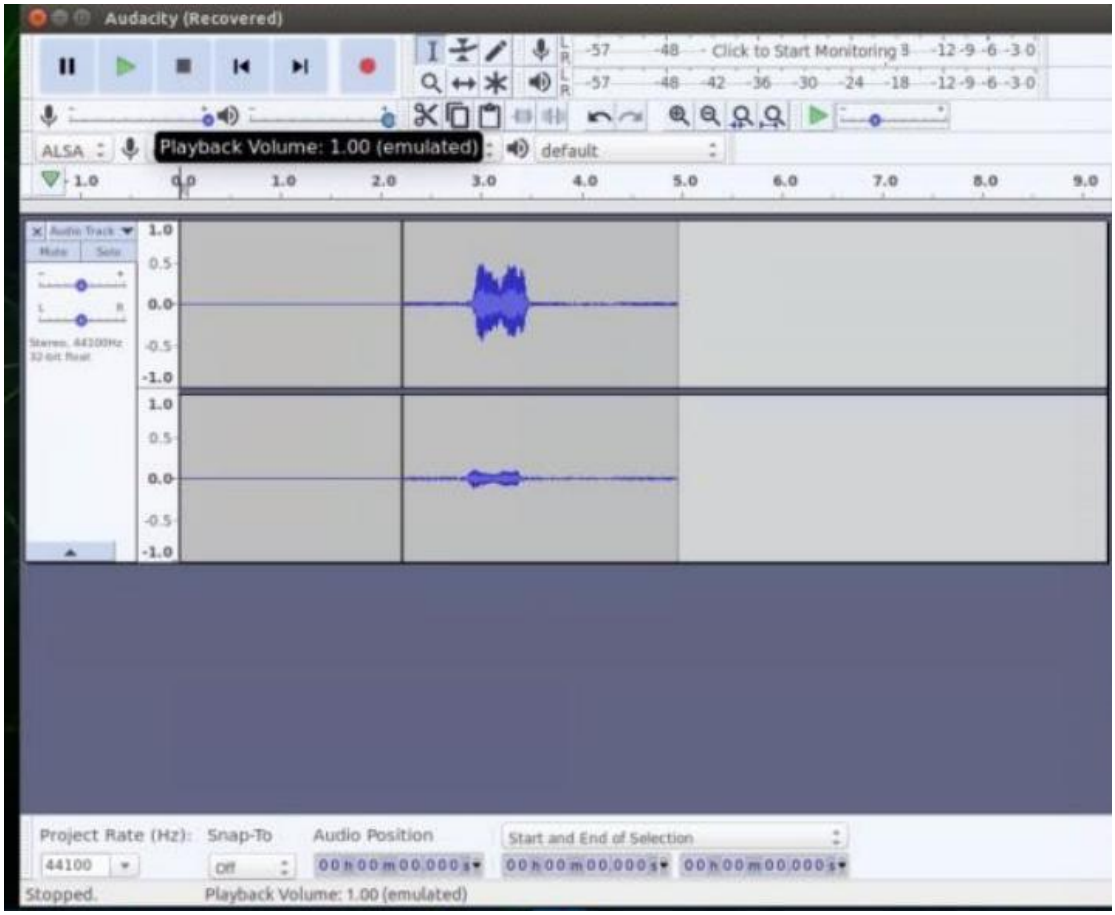
点击系统右上角音量按键，打开 sound settings。



选择输出声音设备，点击测试，注意：麦克风不是音响，这里需要外接一个音响，如果想单独测试麦克风及麦克风录音情况的话直接跳过本小节参看下一节的demo。




连接音响，打开上面打开的软件点击 record 按钮录音。

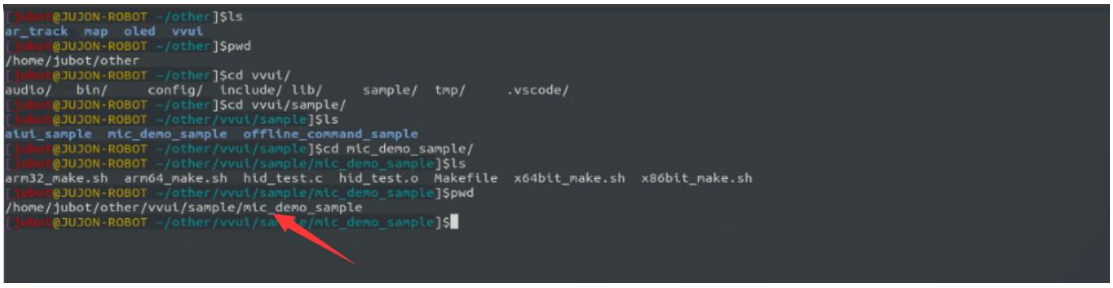



然后可以点击play 播放录音，则证明麦克风与音箱驱动一切正常。

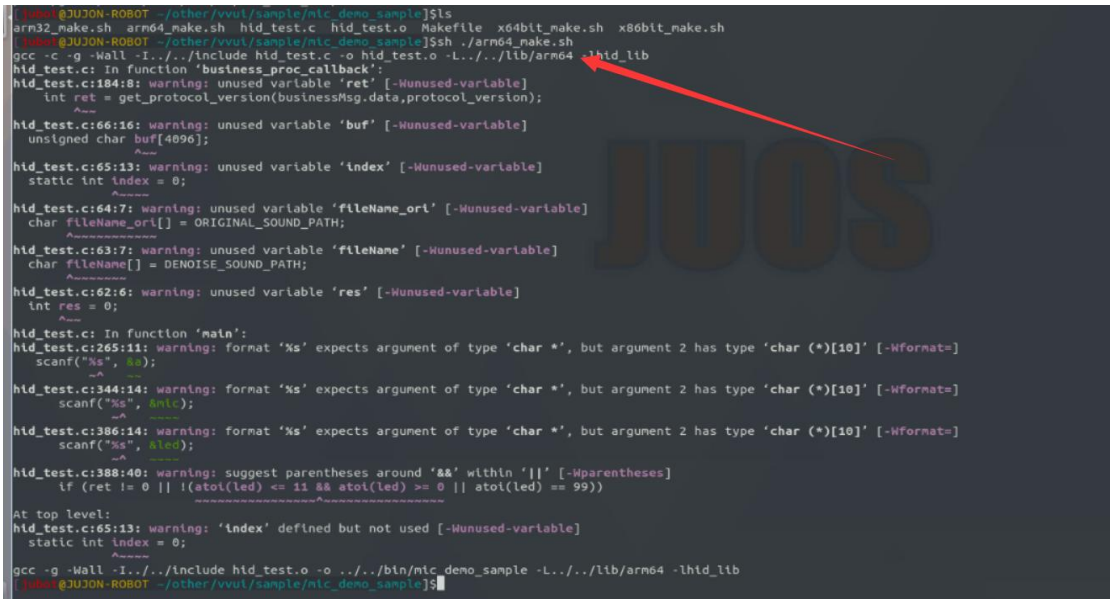
### 26.4.3 语音阵列数据采集测试

我们不仅可以通过语音板录音，还可以提取声源，控制 LED，接下来我们在路威套件中再次测试麦克风阵列是否正常。

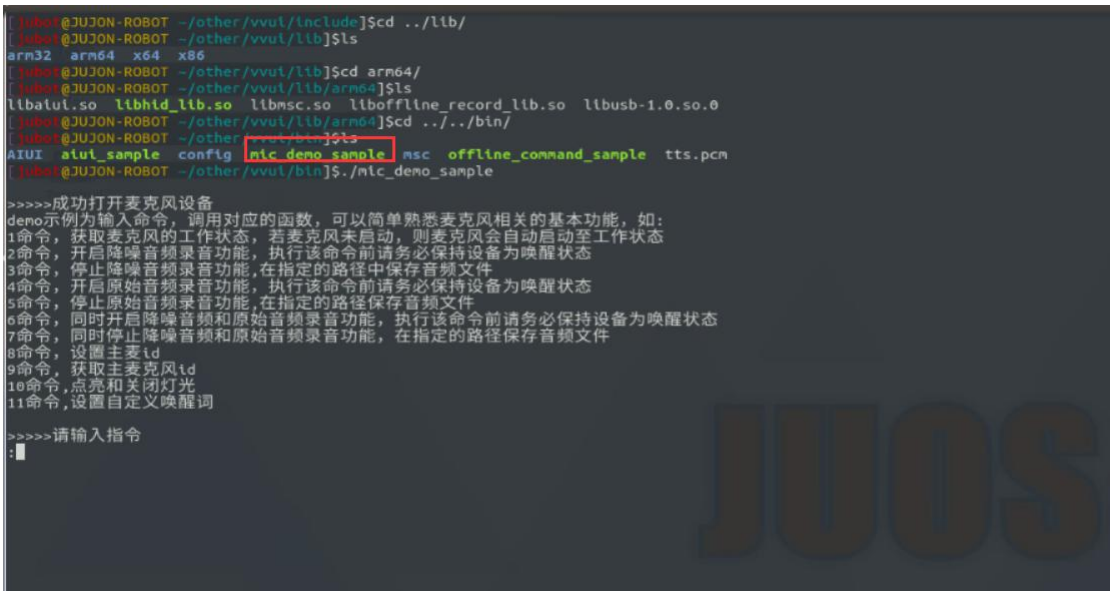
首先进入到下图中的目录下：




使用sh ./arm64_make.sh编译官方测试包组。




进入到该目录，运行我们上一步编译好的程序：




测试如下：

```
[jubot@JUJON-ROBOT ~/other/vvui/bin]$./mic_demo_sample

>>>>>成功打开麦克风设备
demo示例为输入命令，调用对应的函数，可以简单熟悉麦克风相关的基本功能，如:
1命令，获取麦克风的工作状态，若麦克风未启动，则麦克风会自动启动至工作状态
2命令，开启降噪音频录音功能，执行该命令前请务必保持设备为唤醒状态
3命令，停止降噪音频录音功能,在指定的路径中保存音频文件
4命令，开启原始音频录音功能，执行该命令前请务必保持设备为唤醒状态
5命令，停止原始音频录音功能,在指定的路径保存音频文件
6命令，同时开启降噪音频和原始音频录音功能，执行该命令前请务必保持设备为唤醒状态
7命令，同时停止降噪音频和原始音频录音功能，在指定的路径保存音频文件
8命令，设置主麦id
9命令，获取主麦克风id
10命令,点亮和关闭灯光
11命令,设置自定义唤醒词

>>>>>请输入指令
:1
>>>>>麦克风正在启动,软件版本为:R-2.2,协议版本为:

>>>>>请输入指令
:10

>>>>>请输入要点亮的灯id,特别地99表示关闭灯光:1

>>>>>设置灯光成功

>>>>>请输入指令
:
```

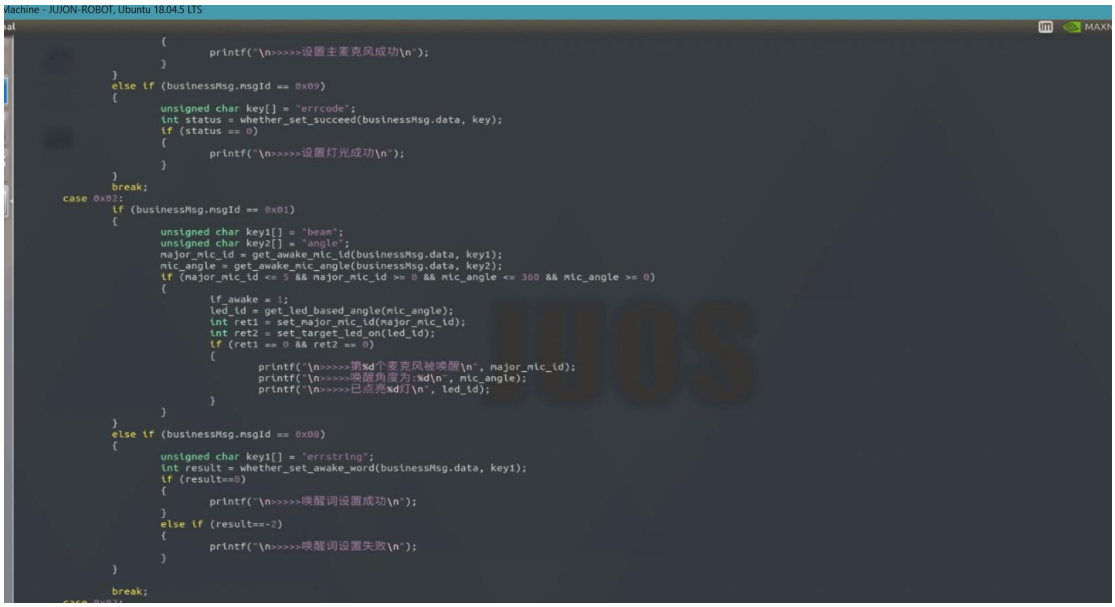
可以观察到语音板灯光变化。

测试代码如下所示：

```
[jubot@JUJON-ROBOT ~/other/vvui/bin]$ls
AIUI  aiui_sample  config  mic_demo_sample  msc  offline_command_sample  tts.pcm
[jubot@JUJON-ROBOT ~/other/vvui/bin]$cd ../sample/mic_demo_sample/
[jubot@JUJON-ROBOT ~/other/vvui/sample/mic_demo_sample]$ls
arm32_make.sh  arm64_make.sh  hid_test.c  hid_test.o  Makefile  x64bit_make.sh  x86bit_make.sh
[jubot@JUJON-ROBOT ~/other/vvui/sample/mic_demo_sample]$vim hid_test.c
```

麦克风录音主要代码如下：

```
        {
                char fileName[] = DENOISE_SOUND_PATH;
                get_denoised_sound(fileName, businessMsg.data);
        }
        else if (businessMsg.msgId == 0x03)
        {
                unsigned char key[] = "errcode";
                int status = whether_set_succeed(businessMsg.data, key);
                if (status == 0)
                {
                        //printf("\n>>>>>您已停止录音\n");
                }
        }
        else if (businessMsg.msgId == 0x04)
        {
                unsigned char key[] = "errcode";
                int status = whether_set_succeed(businessMsg.data, key);
                if (status == 0)
                {
                        //printf("\n>>>>>开/关原始音频成功\n");
                }
        }
        else if (businessMsg.msgId == 0x05)
        {
                unsigned char key[] = "errcode";
                int status = whether_set_succeed(businessMsg.data, key);
                if (status == 0)
                {
                        printf("\n>>>>>设置主麦克风和灯光成功,升级版本不推荐使用\n");
                }
        }
        else if (businessMsg.msgId == 0x06)
        {
                char fileName_ori[] = ORIGINAL_SOUND_PATH;
                get_original_sound(fileName_ori, businessMsg.data);
        }
        else if (businessMsg.msgId == 0x07)
        {
                unsigned char key2[] = "beam";
```




## 26.5 实验结果

能够测试语音阵列是否正常，尝试在上述demo中修改一些其他功能，比如跑马灯等功能。

## 26.6 实验报告

实验目的

实验要求

实验内容

实验总结

# 第二十七章 TTS语音播报

## 27.1 实验目的

在路威套件中使用科大讯飞SDK的TTS语音播报。

## 27.2 实验要求

掌握如何申请科大讯飞SDK，能够使用TTS语音功能包。

## 27.3 实验工具

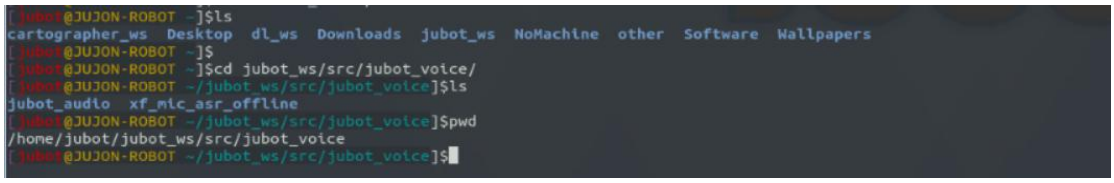
个人电脑一台，路威套件。

## 27.4 实验内容

获取SDK；配置TTS功能包；使用TTS功能包。

### 27.4.1 TTS语音功能包

本章介绍路威套件语音合成 TTS 功能（Text To Speech）应用。

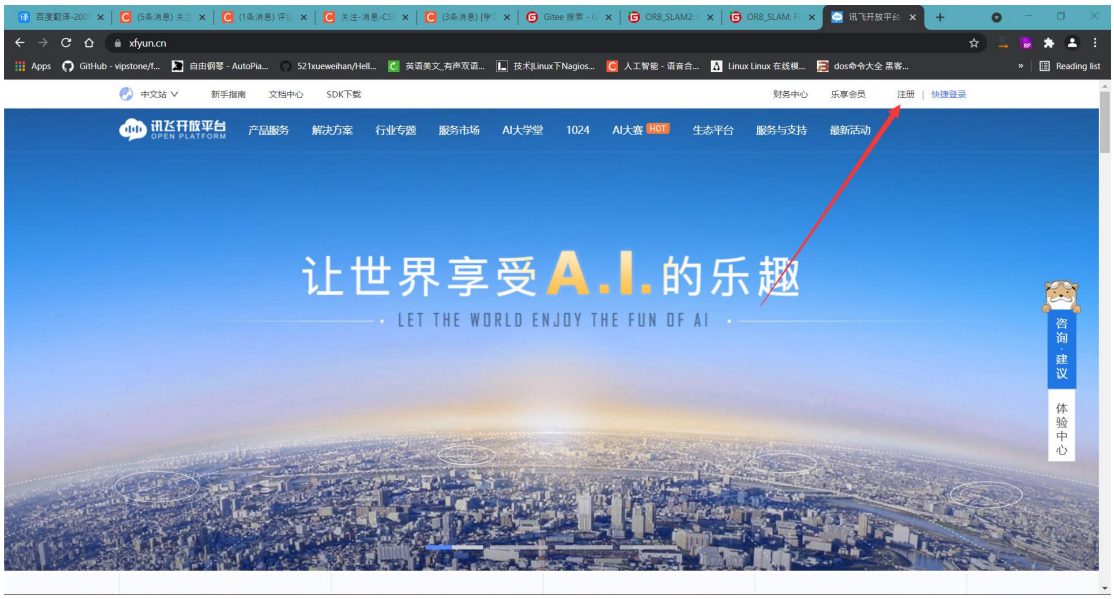



语音合成功能（TTS）基于讯飞开放平台语音技术，实现了将文字信息转化为声音信息，利用科大讯飞语音 api，赋予路威套件语音播报的功能。使用步骤如下。

功能包源码位于机器人端~/jubot_ws/src/jubot_voice/jubot_audio/路径下。

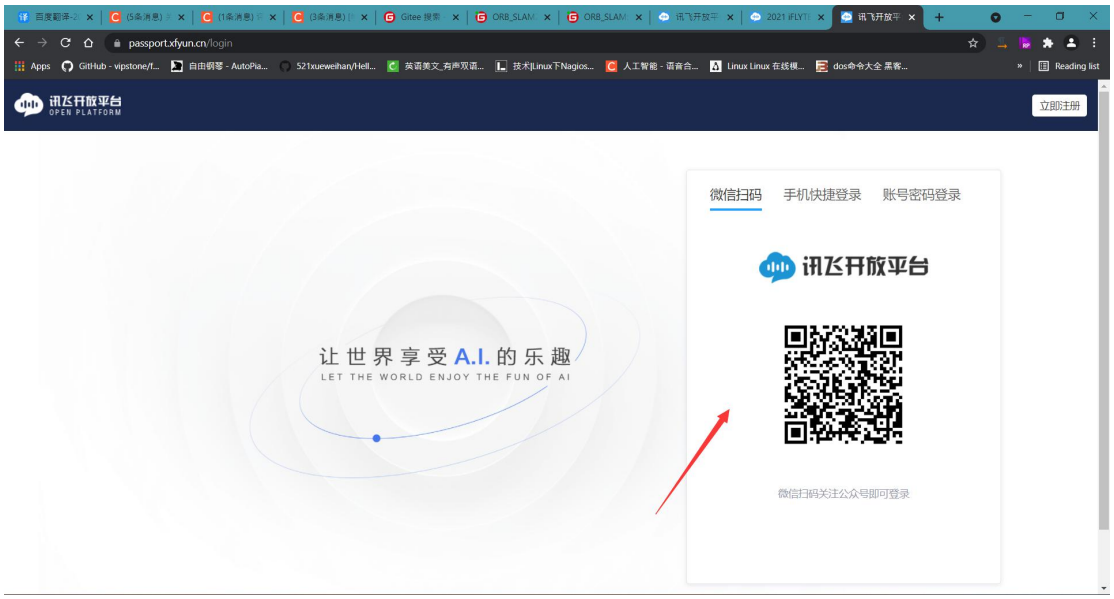
### 27.4.2 获取科大讯飞SDK

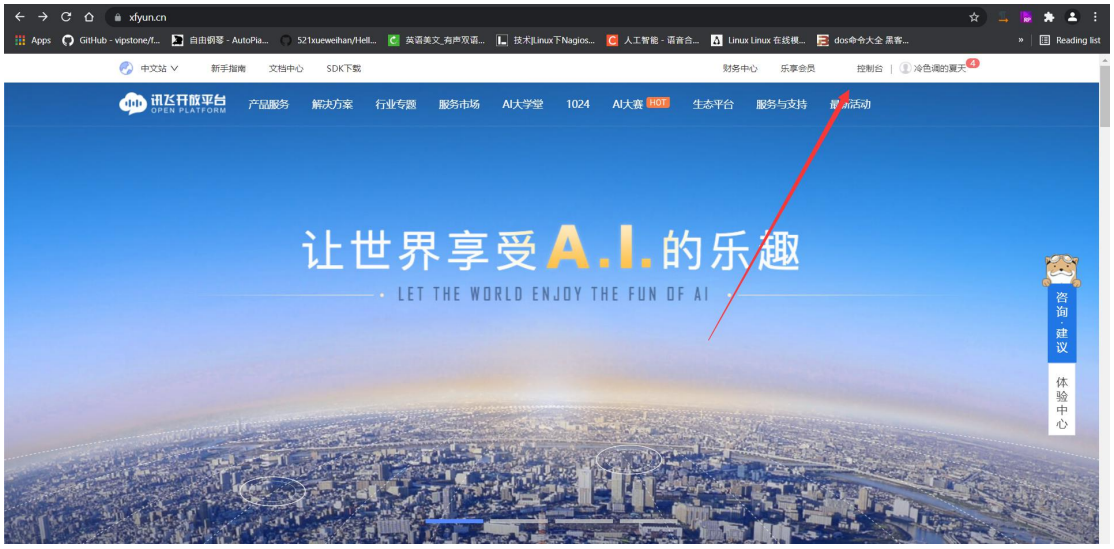
打开讯飞开放平台官网（https://www.xfyun.cn），注册讯飞开放平台账号。

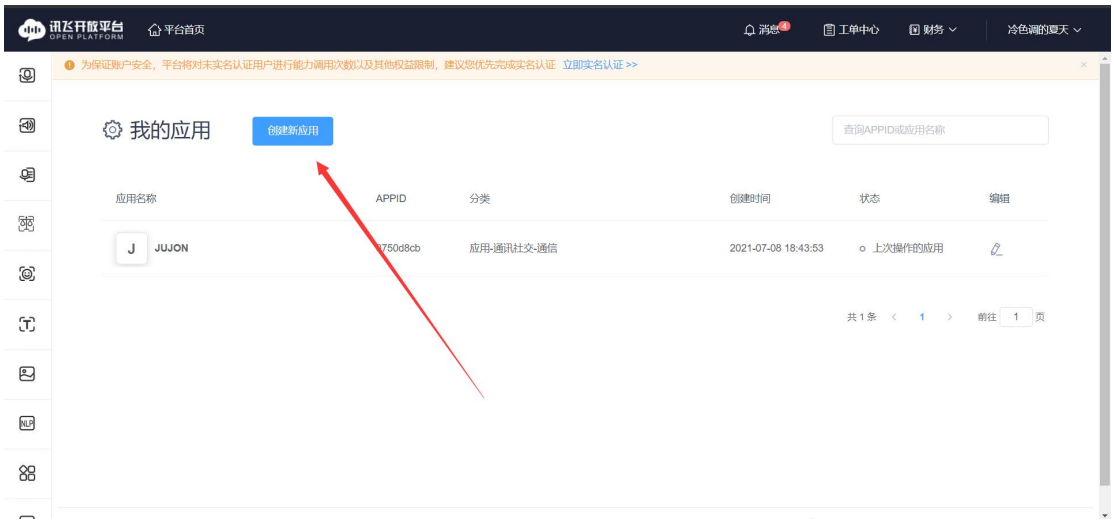



注册完成后，登录进入控制台，创建语音应用。


扫码快速登录控制台。




输入自定义应用信息，完成应用创建。


创建完成后，进入所创建的应用，点击语音合成界面：


记录下所创建应用的 APPID, APISecret 以及 APIKey 三个信息，这三个信息即为功能包所需的授权信息。此时，即完成了语音应用的创建。

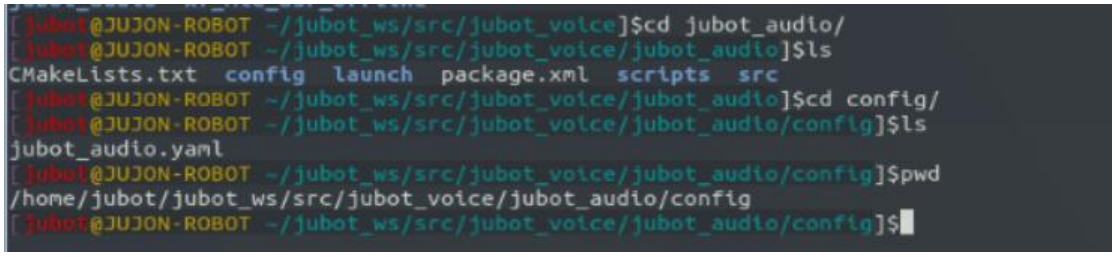
个人账号可调用应用的数量有限，如需更多调用量，请进行实名认证或者联系科大讯飞进行购买扩容。

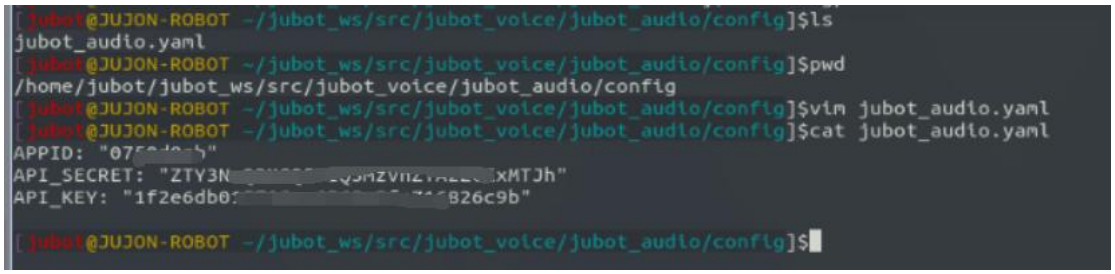
### 27.4.3 配置jubot_audio 功能包

首先，需要配置调用语音合成 API 所需要的 APPID,APIKey 与APISecret。通过 SSH 登录到机器人，切换到 jubot_audio/config/路径下。

```
cd ~/jubot_ws/src/jubot_voice/jubot_audio/config
```

利用 VIM 编辑器打开此路径下的 jubot_audio.yaml 文件。



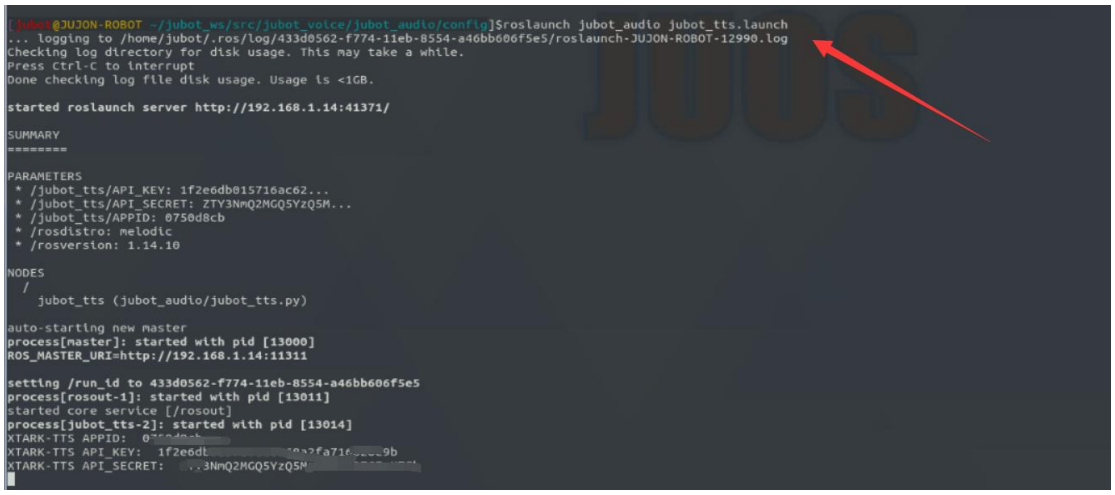

将 APPID,APIKey 与 APISecret 参数引号中的值替换为用户所创建的语音合成应用的 APPID,APIKey 与APISecret 值。配置完成后，即可运行 jubot_audio 功能包。

### 27.4.4 运行jubot_audio 功能包

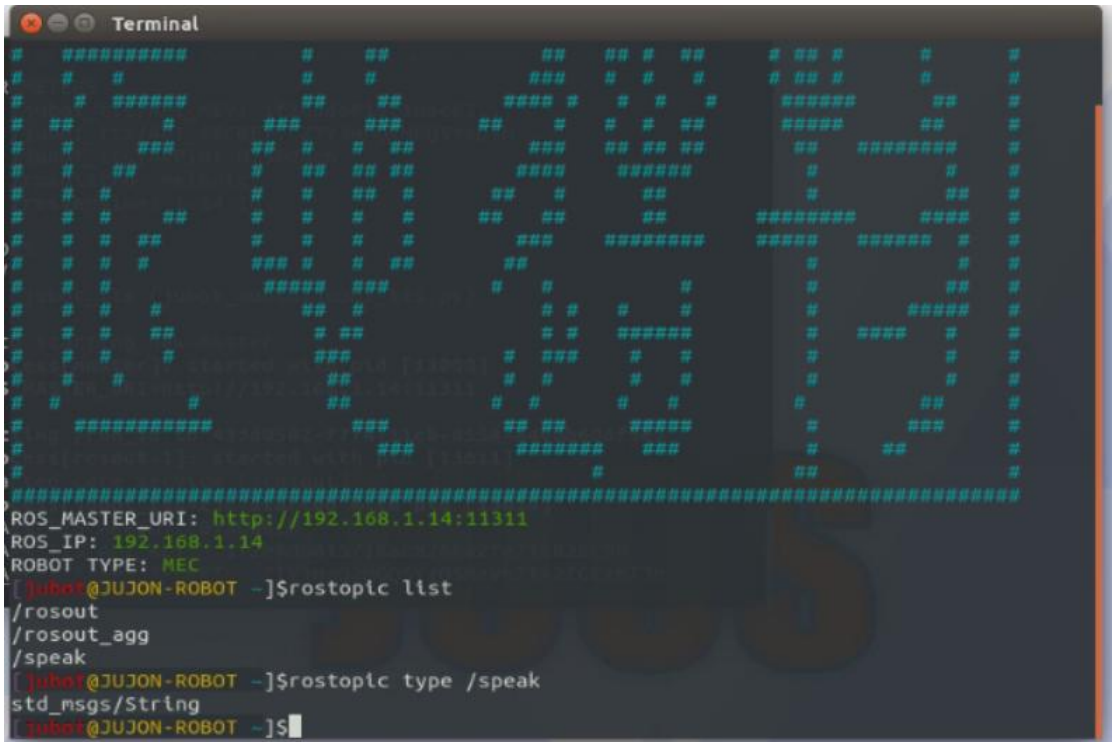
通过 SSH 连接到机器人，启动语音合成 launch 启动文件。

```
roslaunch jubot_audio jubot_tts.launch
```

当显示如下信息时，表示语音合成节点启动成功。




新建一个终端，查看当前话题列表与话题消息类型。




可以看到，/speak 话题即为语音合成话题，话题消息类型为 std_msgs/String 类型，此时，向/speak 话题上发布字符串类型数据，机器人即可合成相应语音并播放。

例如直接用命令行发布‘JUJON’字符串，机器人即可播放。语音合成支持中文与英文，可以直接发布中文语句，机器人即可朗读中文语句。

## 27.5 实验结果

可使用科大讯飞TTS模块实现文字转换语音。

## 27.6 实验报告

实验目的

实验要求

实验内容

实验总结

# 第二十八章 麦克风阵列语音识别

## 28.1 实验目的

在 ROS 系统中测试语音功能包。

## 28.2 实验要求

掌握远场语音阵列驱动操作，能够使用语音功能包。

## 28.3 实验工具

个人电脑一台，路威套件。

## 28.4 实验内容

语音唤醒；语音识别；语音控制路威套件运动；寻找声源；语音导航；主动休眠

本语音功能包利用科大讯飞 6 麦克风语音识别板，主要提供了以下功能：

- 语音唤醒：自动定位声源并设定主麦；
- 语音识别：识别已设定的有效指令；
- 语音控制路威套件运动：前进、后退、左转、右转、停；
- 寻找声源：路威套件移动到唤醒声源处；
- 语音导航：通过语音命令路威套件前往已标记的地图中的目标点；
- 主动休眠：等待下一次唤醒；

下一条语音指令可以打断正在执行的语音动作，例如在路威套件正在左转的时候成功识别到"路威过来"指令，路威套件左转动停止，转为去寻找声源；导航除外，因为控制节点不同，需要到达目标点后才能进行其他语音动作，但导航可以打断导航本 身，例如还没到达 1 点可以发布 2 点来更新目标点。

**雷达检测障碍：**语音控制路威套件的同时雷达检测路威套件周围的障碍物，当路威套件距离最近障碍物小于 0.5m，并且路威套件运动是逐渐靠近障碍物的时候，路威套件自动停下；这时使路威套件靠近障碍物的语音指令无效，使路威套件远离障碍物的语音指令有效。例如：路威套件前进时遇到左前方障碍物被动停下，此时前进和左转指令无效，后退和右转指令仍有效。

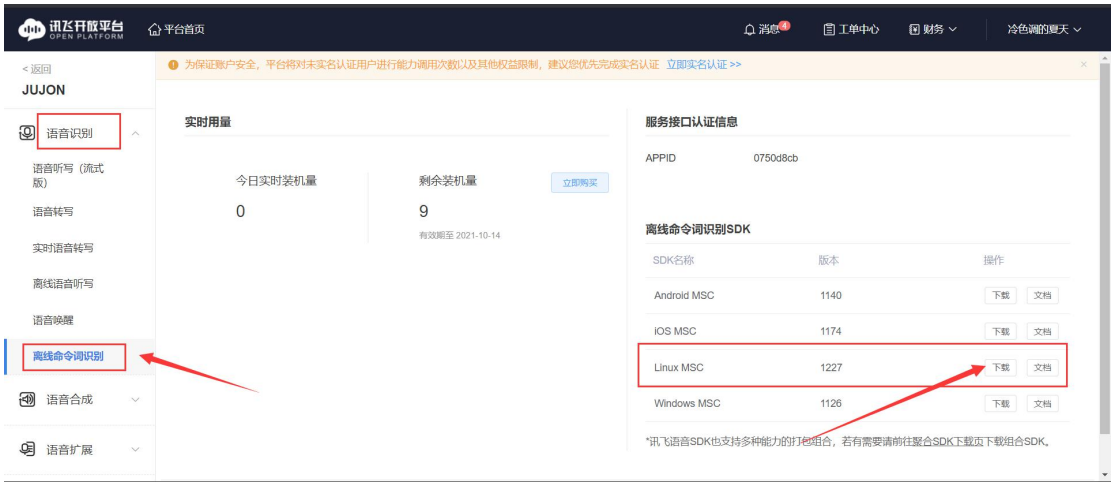
**主麦方向刷新：**结合里程计记录路威套件偏航值，以 5Hz 速度检测路威套件相对唤醒声源的方向，自动刷新主麦方向。

**被动休眠：**连续 15 次语音指令识别失败或者输入空指令后，自动进入休眠状态，当累计 5 次和 10 次语音指令识别失败或者输入空指令时，用户界面会分别发出提醒。

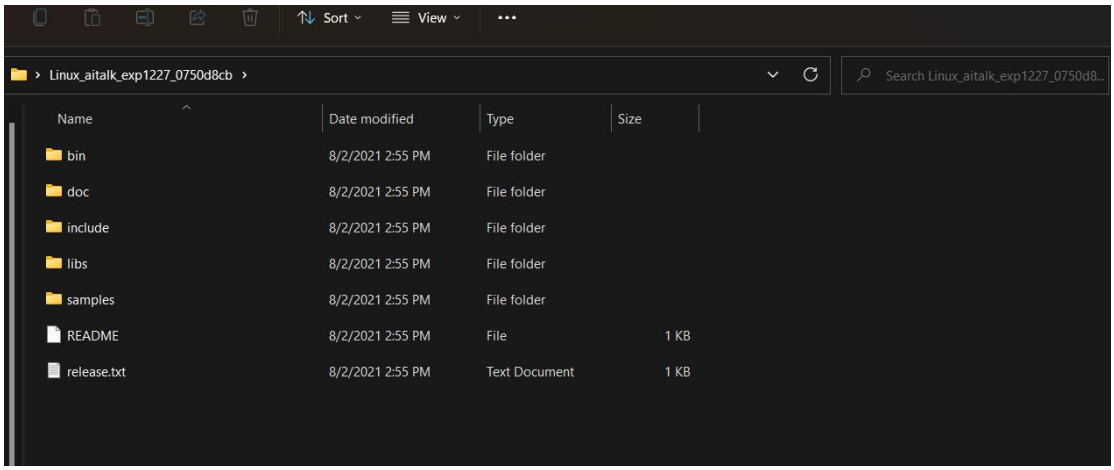
**LED 灯：**通过 LED 灯指示主麦方向以及录音/识别状态，灯亮代表在当前方向的主麦被设定，并且正在录音，此时可以大声说出指令；循环识别状态下，灯灭代表正在处理音频识别中，时间约为 0.3 秒。

### 28.4.1 配置语音控制功能包

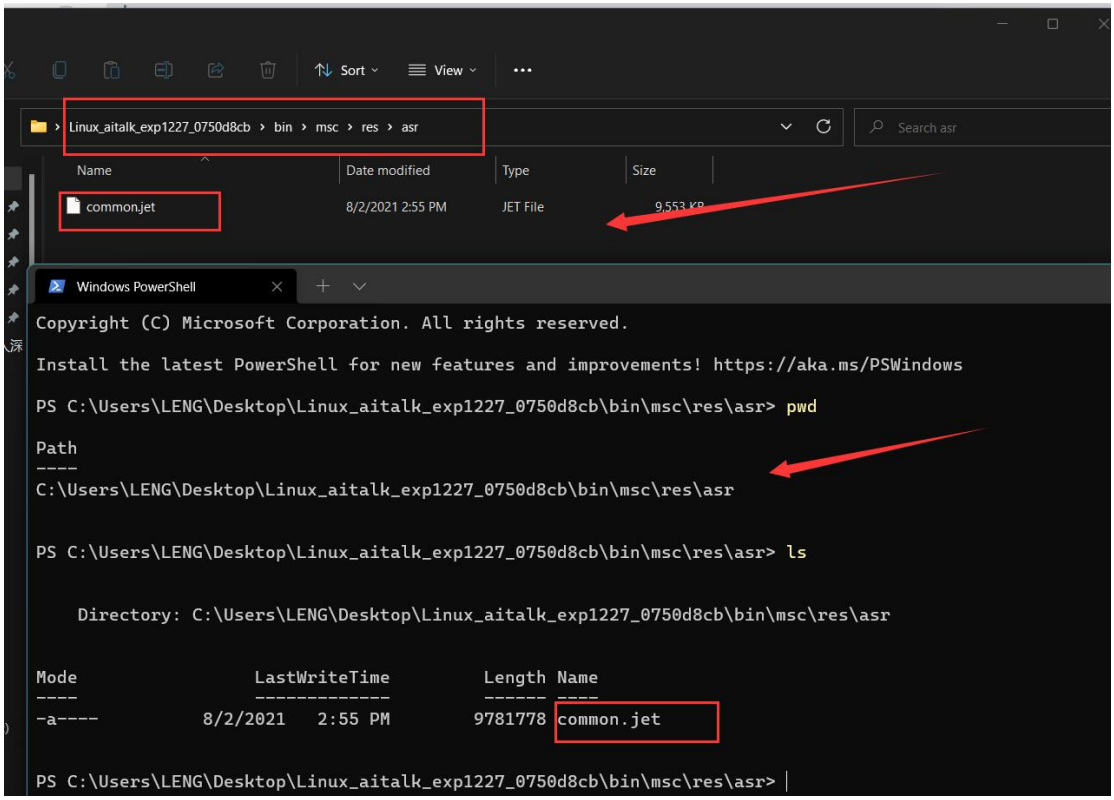
在上一节中我们已经注册并创建了语音识别应用。我们还需要下载离线命令词识别SDK。




下载完成解压如下：

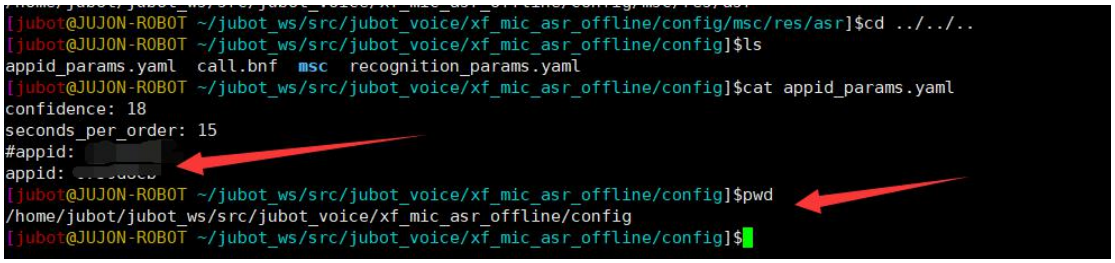



进入到解压后SDK的下面目录中，看到common.jet这个文件。




用它替换路威套件中该目录中的common.jet：

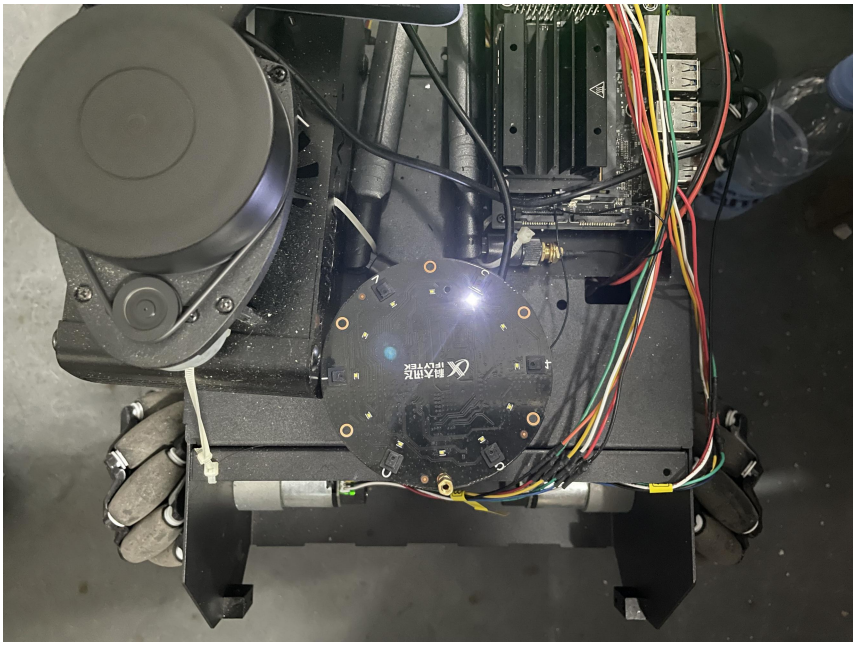

并且再替换下面目录中的appid，这里的appid即为创建语音时候替换的id。




## 28.4.2 语音唤醒

唤醒词为"你好路威"。

在使用过程中，随时可以唤醒或再次唤醒路威套件，唤醒路威套件后自动完成唤醒声源定位，后续可以依赖该定位寻找声源；同时会根据唤醒方向设定主麦并点亮对应方向的LED 灯，如下图。若在语音唤醒的状态下再次唤醒，自动中断上次未完成的录音，转而重新开启一次录音。



根据麦克风阵列的降噪原理，主麦方向的语音信号为有效信号进行增强，其余 5 个麦克风语音信号作为噪声进行衰减，最后得到降噪音频。

注：唤醒路威套件后通过观察 LED 灯，判断是否正确定位声源并设定主麦，若发现方向有偏差请重新唤醒路威套件，否则后续语音指令的识别率会很低。

注：路威套件靠近墙壁等障碍物时，由于声波反射等作用会导致声源方向识别率较低，尽量远离墙壁唤醒。

### 28.4.3 语音识别

语音识别分两种方式：一种是设定固定的录音时间，把固定时长的音频保存为文件，

然后将整个文件送入识别引擎进行识别；另一种方式是不保存文件，直接送入识别引擎，实现边录音边识别。

路威套件采用的是边录音边识别的方式，实时性相对较好，适用于移动机器人的语音交互功能。

本程序也有音频端点检测功能（VAD），即识别静音 0.8 秒自动判断为音频后端点，后续音频不再有效（本程序设定 0.8 秒，）。用户需要开发有关 vad 的参数介绍等，可参考讯飞官网文档：讯飞 API 文档。若一直有音频输入（即用户一直对着主麦讲话），无法检测到后端点的情况下，语音板会一直录音到最大时长 15 秒后停止录音，反馈识别结果。

最终是否成功准确识别指令也与置信度有关，置信度的含义是降噪音频与识别引擎中的声学模型的匹配程度，取值范围为 0-100，数值越大匹配程度越高识别结果越可信。本程序设定的置信度阈值为 18，大于 18 的认为识别结果可信。对识别结果进行发布，小于 18 的认为识别结果不可信，显示识别失败。不讲话录制空白音频时识别结果置信度为 0。录音最大时长 15 秒和置信度阈值 18 的设置在功能包 config/recognition_params.yaml 文件修改，VAD 音频后端点检测 0.8 秒的设置用户亦可修改，但需要在 liboffline_record_lib 库文件源码中修改，并重新生成so 库文件。

### 28.4.4 语音控制

机器人被唤醒后，在录音时间内说出"路威前进""路威后退"等 5 个语音指令可以控制机器人前进、后退、左转、右转和停止。注意"路威左转"与"路威右转"两个语音指令声学模型相近，差别只在左/右一字，说指令时把重音放在左/右这个字上，识别正确率会提高。

```
process[laser_tracker-1]: started with pid [9448]
/opt/ros/melodic/lib/python2.7/dist-packages/roslib/packages.py:470: UnicodeWarning: Unic
al
  if resource_name in files:
process[call_recognition-2]: started with pid [9449]
process[command_recognition-3]: started with pid [9450]
process[motion_control-4]: started with pid [9456]
process[refresh_mic-5]: started with pid [9462]
<--              塔克机器人语音控制命令           -->
<--               唤醒词:你好路威     -->
<--         路威前进 ---> 机器人前进            -->
<--         路威后退 ---> 机器人后退            -->
<--         路威左转 ---> 机器人左转            -->
<--         路威右转 ---> 机器人右转            -->
<--         路威停止   ---> 机器人停止          -->
<--         路威休眠 ---> 机器人休眠            -->
<--         路威过来 ---> 机器人寻找声源        -->
<--         路威去1点 ---> 机器人导航到1点      -->
<--         路威去2点 ---> 机器人导航到2点      -->
<--         路威去3点 ---> 机器人导航到3点      -->


starting
seems to do something
```

语音指令识别成功后会发布在 voice_control.launch 的终端用户界面。

在启动语音包之前注意要先启动导航节点，来驱动底盘以及雷达，若只启动语音包，小车是可以识别语音的，但是无法实际的去运行左转，右转，前进等动作； 启动雷达可以实现实时避障功能。

启动雷达实时监测周围障碍物，当机器人距离最近障碍物小于 0.5m，并且机器人运动趋势是靠近障碍物的时候，机器人自动停下，用户界面也有相应提示；这时让机器人靠近障碍物的语音指令会无效，让机器人远离障碍物的语音指令仍然有效。

在转向的时候，由于主麦刷新功能检测到机器人朝向相对于唤醒声源方位已经发生了变化，所以会实时主动刷新主麦方向，确保主麦方向与唤醒声源方向相同。

### 28.4.5 寻找声源

在录音时间内说出"路威过来"，机器人执行寻找声源功能，机器人根据唤醒声源方向以及雷达检测来实现寻找声源，注意机器人和声源处不能有障碍物阻挡。

寻找声源功能存在一定的识别角度误差，收到"路威过来"指令后，小车会向识别到的声源方向一直前进，直到扫描到障碍物停下。

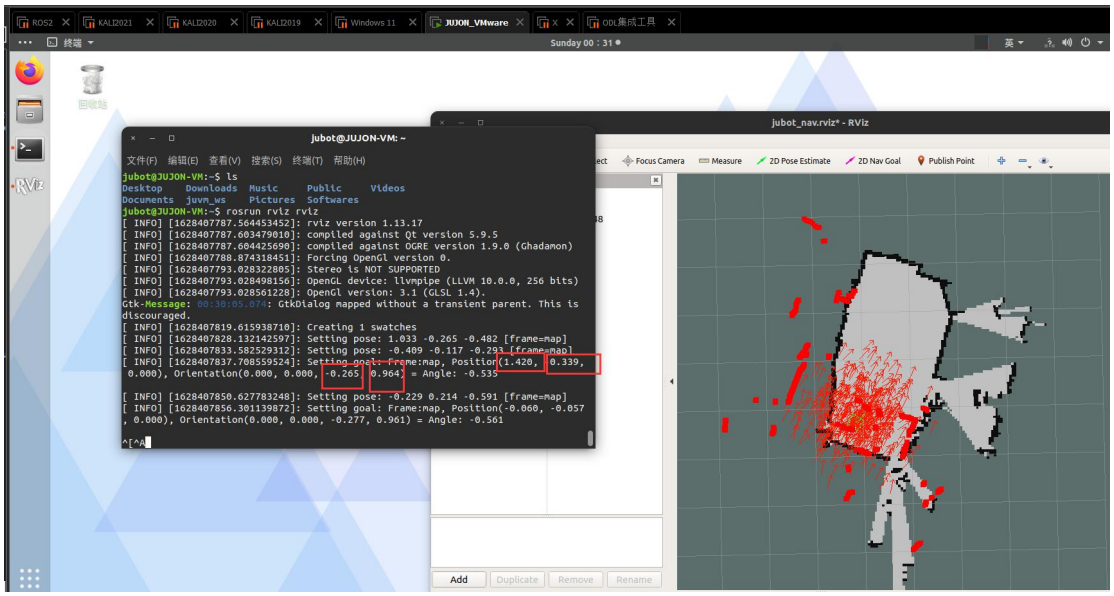
注：阿克曼底盘由于不能原地自转，其寻找声源通过导航实现，方式与其他车型不一致，所以阿克曼小车需要在 config/recognition_params.yaml 文件参数is_ackman 填 true，并且在 jubot_nav 的 jubot_nav.launch 文件上配置好语音导航的地图或者一张比较空旷的地图。

### 28.4.6 语音导航

要进行语音导航相关功能，需先完成路威套件建图导航手册中所述的建图与地图替换步骤，使得 jubot_nav.launch 以当前所需的地图启动。完成建图与导航启动后，可以进行下一步获取目标点坐标的操作。

**获取目标点坐标：**

运行 jubot_nav.launch 后打开 RVIZ，使用 2D 导航工具2D Nav Goal 选取想要的目标点和方向后，RVIZ 终端会显示对应的坐标点参数如下图。




上面四个框依次为x,y,z,w。坐标点取 Position 中的 x 和 y 参数，方向取 Orientation 中的 z 和 w 参数。

设定目标点：得到目标坐标点后在 config/recognition_params.yaml 文件修改 x.y.z.w 参数即可。

```
1_position_x: -2.406        #语音导航1点坐标
1_position_y: 5.532
1_orientation_z: 0.000
1_orientation_w: 1.000

2_position_x: -2.712        #语音导航2点坐标
2_position_y: 0.699
2_orientation_z: 0.000
2_orientation_w: 0.99

3_position_x: -6.234        #语音导航3点坐标
3_position_y: 3.231
3_orientation_z: 0.000
3_orientation_w: 1.000
~
```

完成以上步骤即可进行语音导航功能。

### 28.4.7 主动休眠和被动休眠

语音功能包提供了主动休眠和被动休眠两个休眠功能。

**主动休眠：**在录音时间内说出"路威休眠"，则停止运动进入休眠状态，等待下一次唤醒，不再循环录音识别指令，若正在导航，则结束循环录音并等待到达目标点后停止运动。

**被动休眠：**当连续 15 次语音指令识别失败或者输入空指令后，自动进入休眠状态；当累计 5 次和 10 次语音指令识别失败或者输入空指令时，终端界面会分别发出提醒。若有有效指令输入则计数清零。

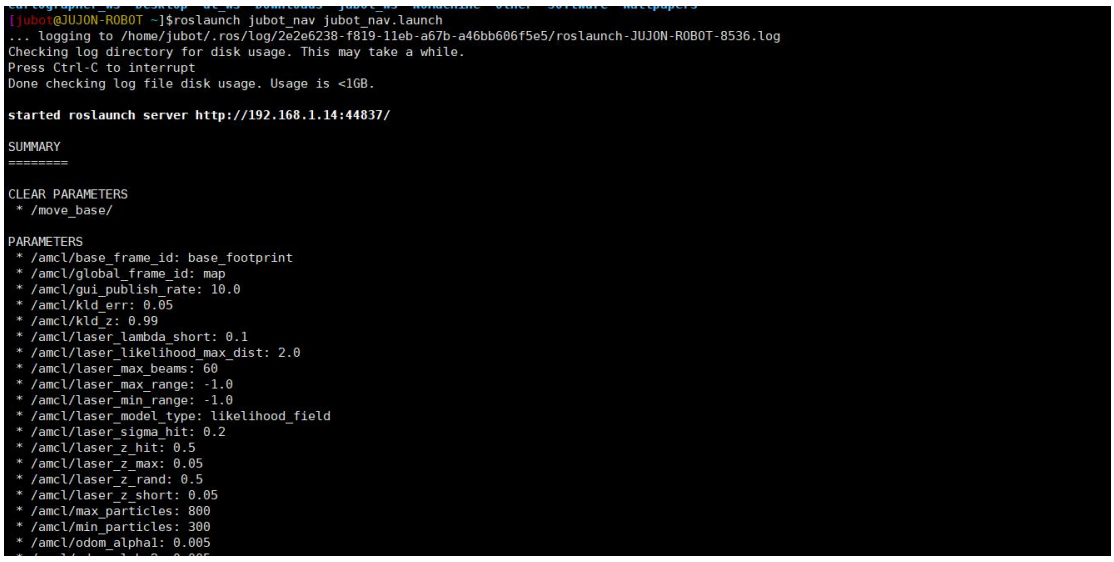
### 28.4.8 运行语音控制软件包

在运行语音控制功能包之前，要确保已经按照路威套件入门教程配置好了机器人的网络。以及已经根据本教程前文所述配置好了语音控制功能包。

本节所有命令均在机器人端运行，终端均需通过 SSH 连接到机器人，后面操作将不再重复描述。

首先，通过以下命令启动机器人底盘导航功能包：

```
roslaunch jubot_nav jubot_nav.launch
```




在启动导航功能包之前，确保已按照建图导航教程完成对环境地图的建立。启动完成之后，如要使用语音导航功能，需在RVIZ 中标定好机器人的位置，并将采集的点位按上文所述输入到语音控制包的配置文件中。

在启动语音控制软件包之前，必须要先启动建图节点，小车才可以运动。

接下来，打开一个新终端，使用如下命令启动麦克风模块驱动节点。

```
roslaunch xf_mic_asr_offline mic_init.launch
```

当显示以下界面时，即表示麦克风模块驱动启动成功：

```
 * /rosdistro: melodic
 * /rosversion: 1.14.10
 * /seconds_per_order: 15
 * /xf_asr_offline_node/source_path: /home/jubot/jubot...

NODES
  /
    xf_asr_offline_node (xf_mic_asr_offline/voice_control)

ROS_MASTER_URI=http://192.168.1.14:11311

/opt/ros/melodic/lib/python2.7/dist-packages/roslib/packages.py:470: UnicodeWarning: Unicode equal comparison failed to convert both arguments
al
  if resource_name in files:
process[xf_asr_offline_node-1]: started with pid [9303]
-----confidence =18
-----time_per_order =15
source_path=/home/jubot/jubot_ws/src/jubot_voice/xf_mic_asr_offline
appid=0750d8cb
>>>>>成功打开麦克风设备
>>>>>麦克风正在启动,软件版本为:R-2.2,协议版本为:
>>>>>开机中，请稍等!
>>>>>开机成功!
唤醒词是:你好路威;nCM:600
>>>>>未设置主麦，请唤醒或设置主麦
>>>>>第2个麦克风被唤醒
>>>>>唤醒角度为:149
>>>>>已点亮5灯
>>>>>开始录制降噪音频
>>>>>开始一次语音识别!
>>>>>设置主麦成功!

>>>>>停止录制降噪音频
>>>>>是否识别成功： [ 否 ]
>>>>>关键字的置信度: [ 0 ]
>>>>>关键字置信度较低，文本不予显示


>>>>>开始录制降噪音频
```

因离线语音识别需要联网鉴权，在开机后第一次启动麦克风驱动时，可能会出现以下卡在"开机中，请稍等"的情况，出现该情况时，仅需 ctrl+c 终止节点，然后再运行启动功能包命令重新启动即可。

接下来，即可启动语音控制相关节点，重新打开一个新的终端，通过如下命令启动语音控制节点：

```
roslaunch xf_mic_asr_offline voice_control.launch
```

启动成功如下图所示，终端提示了各个语音命令词，用户在语音唤醒机器人后， 即可使用命令词命令机器人运动。

```
ROS_MASTER_URI=http://192.168.1.14:11311

process[laser_tracker-1]: started with pid [9448]
/opt/ros/melodic/lib/python2.7/dist-packages/roslib/packages.py:470: UnicodeWarning: Unicode equal
al
  if resource_name in files:
process[call_recognition-2]: started with pid [9449]
process[command_recognition-3]: started with pid [9450]
process[motion_control-4]: started with pid [9456]
process[refresh_mic-5]: started with pid [9462]
<--             塔克机器人语音控制命令          -->
<--              唤醒词:你好路威     -->
<--         路威前进 ---> 机器人前进           -->
<--         路威后退 ---> 机器人后退           -->
<--         路威左转 ---> 机器人左转           -->
<--         路威右转 ---> 机器人右转           -->
<--         路威停止   ---> 机器人停止         -->
<--         路威休眠 ---> 机器人休眠           -->
<--         路威过来 ---> 机器人寻找声源       -->
<--         路威去1点 ---> 机器人导航到1点     -->
<--         路威去2点 ---> 机器人导航到2点     -->
<--         路威去3点 ---> 机器人导航到3点     -->


starting
seems to do something
```

```
收到指令: 8)
收到指令: 8)
您已经连续【输入空指令or识别失败】5次，累计达15次自动进入休眠，输入有效指令后计数清零
收到指令: 8)
您已经连续【输入空指令or识别失败】10次，累计达15次自动进入休眠，输入有效指令后计数清零
收到指令: 8)
收到指令: 8)
收到指令: 8)
收到指令: 8)
您已经连续【输入空指令or识别失败】5次，累计达15次自动进入休眠，输入有效指令后计数清零
收到指令: 8)
收到指令: 8)
您已经连续【输入空指令or识别失败】10次，累计达15次自动进入休眠，输入有效指令后计数清零
收到指令: 8)
好的，前进
收到指令: 8)
收到指令: 8)
好的，后退
收到指令: 8)
遇到障碍物，已停止运动
收到指令: 8)
收到指令: 8)
```

如要启动语音播报功能（TTS），需按照语音播报功能手册中所述完成语音播报功能的设置，然后通过以下命令启动语音播报功能，启动成功后，当机器人收到语音命令时，将会通过语音合成进行交互提示。

```
roslaunch jubot_audio jubot_tts.launch
```

```
started roslaunch server http://192.168.1.14:40199/

SUMMARY
========

PARAMETERS
 * /jubot_tts/API_KEY: 1f2e6db015716ac62...
 * /jubot_tts/API_SECRET: ZTY3NmQ2MGQ5YzQ5M...
 * /jubot_tts/APPID: 0750d8cb
 * /rosdistro: melodic
 * /rosversion: 1.14.10

NODES
  /
    jubot_tts (jubot_audio/jubot_tts.py)

ROS_MASTER_URI=http://192.168.1.14:11311

process[jubot_tts-1]: started with pid [10493]
XTARK-TTS APPID:  0750d8cb
XTARK-TTS API_KEY:  1f2e6db015716ac6268a2fa716826c9b
XTARK-TTS API_SECRET:  ZTY3NmQ2MGQ5YzQ5MzVhZTA2ZGExMTJh
------>开始发送文本数据
ws is closed

result.mp3:

 File Size: 3.89k     Bit Rate: 50.8k
  Encoding: MPEG audio
  Channels: 1 @ 16-bit
Samplerate: 16000Hz
Replaygain: off
  Duration: 00:00:00.61

In:100%  00:00:00.61 [00:00:00.00] Out:9.79k [  ====|====  ] Hd:4.7 Clip:0
Done.
```

## 28.5 实验结果

能够测试语音阵列是否正常，尝试在上述demo中修改一些其他功能。

## 28.6 实验报告

实验目的

实验要求

实验内容

实验总结

# 第二十九章 多机器人通信

## 29.1 实验目的

实现多个机器人之间通信。

## 29.2 实验要求

掌握多机器人网络配置、多机器人之间软件包应用。

## 29.3 实验工具

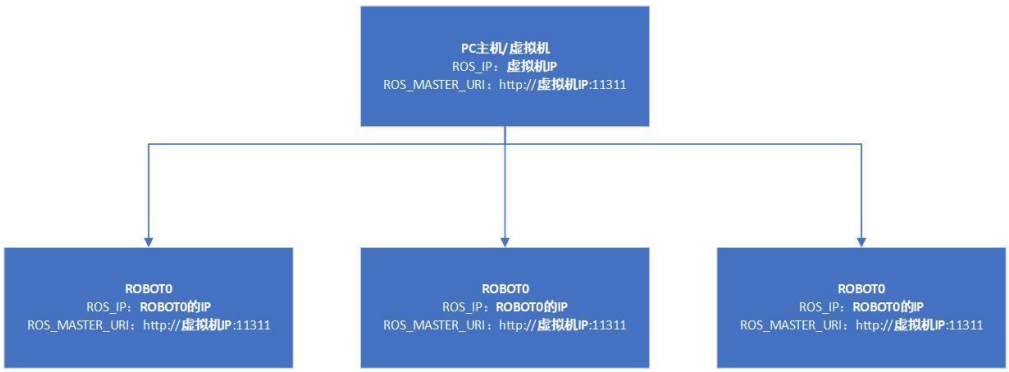
个人电脑一台，路威套件。

## 29.4 实验内容

### 29.4.1 多机器人网络配置

在开始多机器人学习之前，要首先完成多机器人的网络配置。得益于 ROS 灵活的网络通信框架，我们可以通过以下步骤来完成多机器人的网络环境配置。

多机器人通信框架采用一主机多从机的方式，一般为将 PC 主机或虚拟机配置为 ROS 主机，各机器人作为从机挂载到 ROS 网络中，网络拓扑简图如下。




**配置过程：**

打开虚拟机端的~/.bashrc 文件，将 ROS_MASTER_URI 改为如下图所示。

```
interface=ens33
export IPAddress=`ifconfig $interface | grep -o 'inet [^ ]*' | cut -d " " -f2`
source /opt/ros/melodic/setup.bash
source ~/juvm_ws/devel/setup.bash

#将ROBOT_IP修改为获取到的机器人IP
export ROBOT_IP=192.168.1.18

alias sshrobot='ssh jubot@$ROBOT_IP'
export ROS_IP=$IPAddress
export ROS_MASTER_URI=http://$ROBOT_IP:11311
bash ~/.vm.sh $ROS_MASTER_URI $ROS_IP
```

连接到机器人端，修改机器人端的~/.bashrc 文件，如下图所示。

Robot0:

```
export ROS_IP=`ifconfig $interface | grep -o 'inet [^ ]*' | cut -d " " -f2`
#export ROS_MASTER_URI=http://$ROS_IP:11311
#export ROS_MASTER_URI=http://192.168.1.30:11311
export ROS_MASTER_URI=http://192.168.1.18:11311
echo -e "ROS_MASTER_URI: \033[32m$ROS_MASTER_URI\033[0m"
echo -e "ROS_IP: \033[32m$ROS_IP\033[0m"

####修改机器人型号####
# 麦克纳姆轮:    MEC
# 四轮差速:      4WD
export ROBOT_TYPE=4WD
echo -e "ROBOT TYPE: \033[32m$ROBOT_TYPE\033[0m"
```

Robot1:

```
export ROS_IP=`ifconfig $interface | grep -o 'inet [^ ]*' | cut -d " " -f2`
#export ROS_MASTER_URI=http://$ROS_IP:11311
export ROS_MASTER_URI=http://192.168.1.18:11311
#export ROS_MASTER_URI=http://192.168.1.14:11311
echo -e "ROS_MASTER_URI: \033[32m$ROS_MASTER_URI\033[0m"
echo -e "ROS_IP: \033[32m$ROS_IP\033[0m"

####修改机器人型号####
# 麦克纳姆轮:    MEC
# 四轮差速:      4WD
export ROBOT_TYPE=MEC
echo -e "ROBOT TYPE: \033[32m$ROBOT_TYPE\033[0m"
```

多个机器人均参照此配置进行设置。配置完成后，运行source ~/.bashrc 命令，即可生效配置。

至此，多机器人通信网络配置完成，此时，ROS 网络的 Master 节点依赖于虚拟机端，所以，要在机器人端运行任意 ROS 功能时，必须在虚拟机端启动 roscore，否则， 机器人端将会找不到 ROS Master 节点。

### 29.4.1 多机器人功能包介绍

jubot_multirobot 文件夹在机器人的~/jubot_ws/src/文件夹下。

机器人端的软件包，主要运行了软件包中的算法及驱动部分，导航算法以及机器人驱动算法，均运行在机器人端。

在 jubot_multirobot 软件包中，包含了四个子软件包。

```
[jubot@JUJON-ROBOT ~/jubot_ws/src]$ls
CMakeLists.txt  jubot_apps  jubot_ctl  jubot_cv  jubot_driver  jubot_multirobot  jubot_nav  jubot_nav_depthcamera
[jubot@JUJON-ROBOT ~/jubot_ws/src]$cd jubot_multirobot/
[jubot@JUJON-ROBOT ~/jubot_ws/src/jubot_multirobot]$ls
jubot_ctl_multirobot  jubot_driver_multirobot  jubot_nav_multirobot  jubot_talk_multirobot
[jubot@JUJON-ROBOT ~/jubot_ws/src/jubot_multirobot]$
```

软件包说明如下：

| | | |
|---|---|---|
| jubot_ctl_multirobot | jubot_keyboard.la<br><br>unch | 单机器人键盘遥控启动文件（可分别控制单个机器人移动） |

| | | |
|---|---|---|
| | jubot_keyboard_all.launch | 多机器人键盘遥控启动文件（可同时控制多台机器人同时移动） |
| jubot_driver_multirobot | jubot_bringup_multirobot.launch | 多机器人驱动启动文件 |
| | jubot_camera.launch | 多机器人摄像头驱动启动文件 |
| | jubot_lidar.launch | 多机器人雷达驱动启动文件 |
| jubot_nav_multirobot.launch | jubot_map_server.launch | 多机器人导航 map_server 服务启动文件 |
| | jubot_nav.launch | 多机器人导航算法启动文件 |
| jubot_talk_multirobot | listener.py | 多机器人通信订阅者Demo |
| | talker.py | 多机器人通信发布者Demo |

### 29.4.3 多机器人软件应用

jubot_multirobot 软件包提供了四大功能。

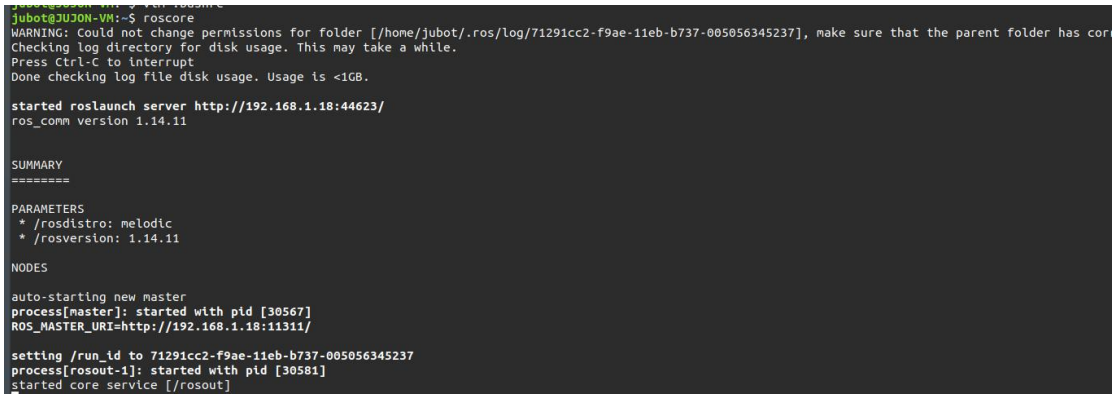
- 多机器人驱动支持
- 多机器人通信示例
- 多机器人独立控制/同时控制
- 多机器人导航框架

在使用各软件包之前，务必确保已经完成多机器人网络配置章节操作。具体操作步骤请参考下文。

jubot_multirobot 扩展包中的 jubot_driver_multirobot 软件包实现了多机器人驱动在同一 ROS 网络中运行，各传感器及控制话题互不影响，并且可互相通信，为多机器人控制奠定了基础。

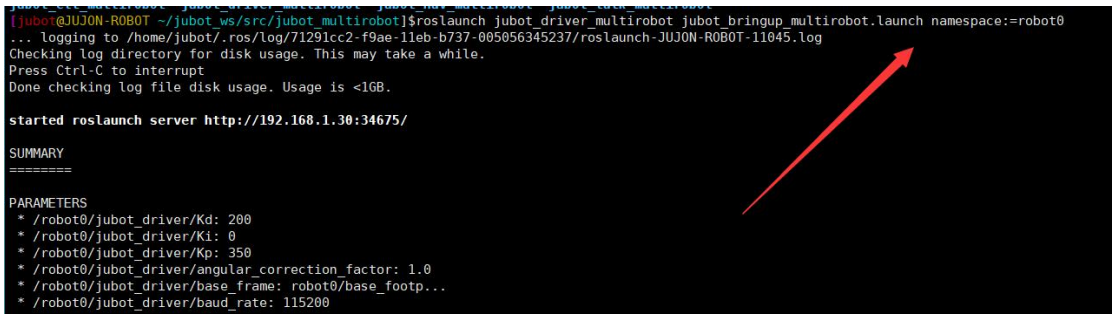
**操作步骤：**

在虚拟机/PC 主机端启动 roscore

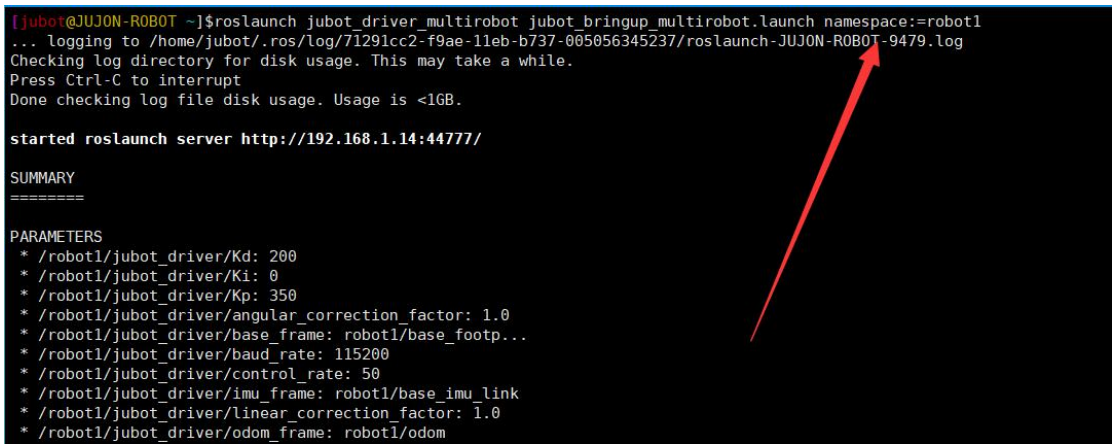



在机器人端启动多机器人启动文件，后面参数 namespace 为当前机器人命名空间，可以自行设置，不同机器人应设置不同的命名空间，机器人即可以当前设置的命名启动。

```
roslaunch jubot_driver_multirobot jubot_bringup_multirobot.launch namespace:=robot0
```



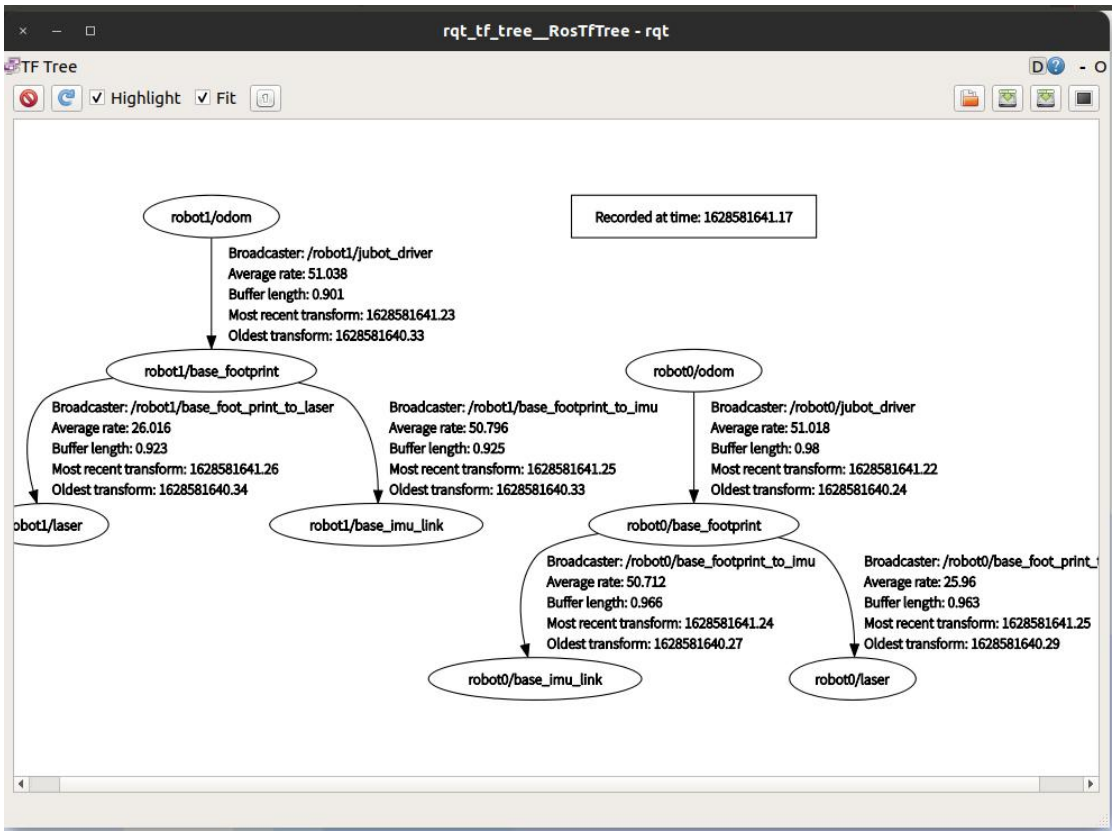

```
roslaunch jubot_driver_multirobot jubot_bringup_multirobot.launch namespace:=robot1
```




机器人话题说明。机器人启动成功后，其发布以及收听的话题，均以设置的命名空间为前缀，如下图。

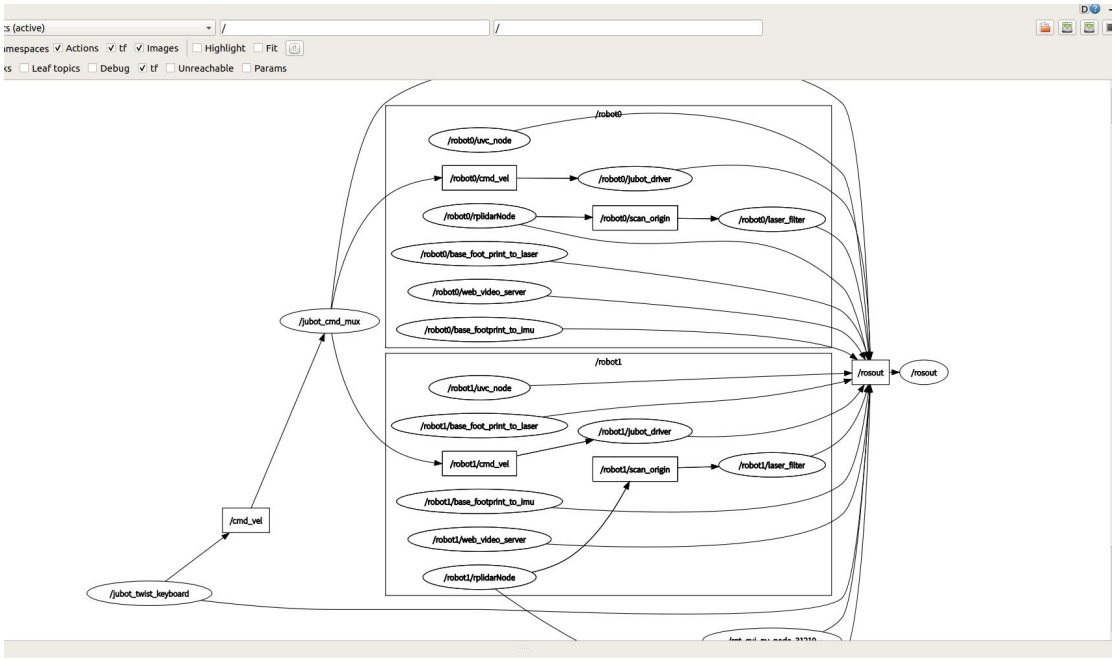
```
jubot@JUJON-VM:~$ rosrun rqt_tf_tree rqt_tf_tree
^Cjubot@JUJON-VM:~$ rostopic list
/robot0/camera_info
/robot0/cmd_vel
/robot0/image_raw/compressed
/robot0/imu
/robot0/jubot/aset
/robot0/jubot/avel
/robot0/jubot/bset
/robot0/jubot/bvel
/robot0/jubot/cset
/robot0/jubot/cvel
/robot0/jubot/dset
/robot0/jubot/dvel
/robot0/jubot_driver/parameter_descriptions
/robot0/jubot_driver/parameter_updates
/robot0/odom
/robot0/scan
/robot0/scan_origin
/robot0/voltage
/robot1/camera_info
/robot1/cmd_vel
/robot1/image_raw/compressed
/robot1/imu
/robot1/jubot/aset
/robot1/jubot/avel
/robot1/jubot/bset
/robot1/jubot/bvel
/robot1/jubot/cset
/robot1/jubot/cvel
/robot1/jubot/dset
/robot1/jubot/dvel
/robot1/jubot_driver/parameter_descriptions
/robot1/jubot_driver/parameter_updates
/robot1/odom
/robot1/scan
/robot1/scan_origin
/robot1/voltage
/rosout
/rosout_agg
/tf
```

其里程计及 IMU 传感器的坐标系名称，也均带有所设置的机器人命名，在虚拟机端运行 `rosrun rqt_tf_tree rqt_tf_tree` 如下图所示。




若启动多个机器人，需注意在运行驱动启动文件时，机器人命名空间（namespace）需设置

为不同，如robot0 与 robot1，同时启动多个机器人驱动时，ROS 节点图如下图所示。




可以看到，多个机器人相关话题均带有命名空间前缀，即可通过编写不同的话题来多机器人通信示例。

jubot_multirobot 扩展包中的 jubot_talk_multirobot 软件包提供了多机器人通信示例，包含了多机器人间通过 ROS 话题通信的示例代码。

在运行此示例前，请确认已完成多机器人网络配置。

**操作步骤：**

首先，在虚拟机端（Master 节点）启动 roscore

```
jubot@JUJON-VM:~$ roscore
WARNING: Could not change permissions for folder [/home/jubot/.ros/log/71291cc2-f9ae-11eb-b737-005056345237], make sure th
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://192.168.1.18:44623/
ros_comm version 1.14.11


SUMMARY
========

PARAMETERS
 * /rosdistro: melodic
 * /rosversion: 1.14.11

NODES

auto-starting new master
process[master]: started with pid [30567]
ROS_MASTER_URI=http://192.168.1.18:11311/

setting /run_id to 71291cc2-f9ae-11eb-b737-005056345237
process[rosout-1]: started with pid [30581]
started core service [/rosout]
^[^A
```

然后，在其中一机器人端启动talk 节点。

```
rosrun jubot_talk_multirobot talker.py
```

```
^C[jubot@JUJON-ROBOT ~]$rosrun jubot_talk_multirobot talker.py
[INFO] [1628582644.945450]: hello world 1628582644.94
[INFO] [1628582645.046661]: hello world 1628582645.04
[INFO] [1628582645.149032]: hello world 1628582645.15
[INFO] [1628582645.245695]: hello world 1628582645.24
[INFO] [1628582645.349724]: hello world 1628582645.35
[INFO] [1628582645.448978]: hello world 1628582645.45
[INFO] [1628582645.548787]: hello world 1628582645.55
[INFO] [1628582645.649248]: hello world 1628582645.65
[INFO] [1628582645.749350]: hello world 1628582645.75
[INFO] [1628582645.849332]: hello world 1628582645.85
[INFO] [1628582645.949206]: hello world 1628582645.95
[INFO] [1628582646.049206]: hello world 1628582646.05
[INFO] [1628582646.149132]: hello world 1628582646.15
```

此时，机器人将会在/chatter 话题上发布带有时间戳的 hello world 消息。

在任一机器人上启动 listener 节点，收听机器人发出的 hello world 消息。

```
rosrun jubot_talk_multirobot listener.py
```

```
[jubot@JUJON-ROBOT ~/jubot_ws/src/jubot_multirobot]$rosrun jubot_talk_multirobot listener.py
[INFO] [1628582677.967756]: /listener_12708_1628582677431I heard hello world 1628582677.95
[INFO] [1628582678.062395]: /listener_12708_1628582677431I heard hello world 1628582678.05
[INFO] [1628582678.160672]: /listener_12708_1628582677431I heard hello world 1628582678.15
[INFO] [1628582678.262241]: /listener_12708_1628582677431I heard hello world 1628582678.25
[INFO] [1628582678.362999]: /listener_12708_1628582677431I heard hello world 1628582678.35
[INFO] [1628582678.468681]: /listener_12708_1628582677431I heard hello world 1628582678.45
[INFO] [1628582678.561470]: /listener_12708_1628582677431I heard hello world 1628582678.55
[INFO] [1628582678.662756]: /listener_12708_1628582677431I heard hello world 1628582678.65
[INFO] [1628582678.771236]: /listener_12708_1628582677431I heard hello world 1628582678.74
[INFO] [1628582678.871776]: /listener_12708_1628582677431I heard hello world 1628582678.85
[INFO] [1628582678.952989]: /listener_12708_1628582677431I heard hello world 1628582678.94
[INFO] [1628582679.058622]: /listener_12708_1628582677431I heard hello world 1628582679.04
```

## 代码分析：

### talker.py

```python
#!/usr/bin/env python
import rospy
from std_msgs.msg import String

def talker():
    pub = rospy.Publisher('chatter', String, queue_size=10)
    rospy.init_node('talker', anonymous=True)
    rate = rospy.Rate(10) # 10hz
    while not rospy.is_shutdown():
        hello_str = "hello world %s" % rospy.get_time()
        rospy.loginfo(hello_str)
        pub.publish(hello_str)
        rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

### listener.py

```python
#!/usr/bin/env python
import rospy
from std_msgs.msg import String

def callback(data):
    rospy.loginfo(rospy.get_caller_id() + 'I heard %s', data.data)

def listener():

    # In ROS, nodes are uniquely named. If two nodes with the same
    # name are launched, the previous one is kicked off. The
    # anonymous=True flag means that rospy will choose a unique
    # name for our 'listener' node so that multiple listeners can
    # run simultaneously.
    rospy.init_node('listener', anonymous=True)

    rospy.Subscriber('chatter', String, callback)

    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()

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

代码比较简单，在前面章节有详细描述，在这里不再赘述。

## 29.5 实验结果

掌握多机器人网络配置、多机器人之间软件包应用

## 29.6 实验报告

实验目的

实验要求

实验内容

实验总结

# 第三十章 多机器人联动

## 30.1 实验目的

实现多个机器人共同运动，联合运动与导航。

## 30.2 实验要求

掌握多机器人网络配置、软件包应用。

## 30.3 实验工具

个人电脑一台，路威套件。

## 30.4 实验内容

### 30.4.1 多机器人驱动

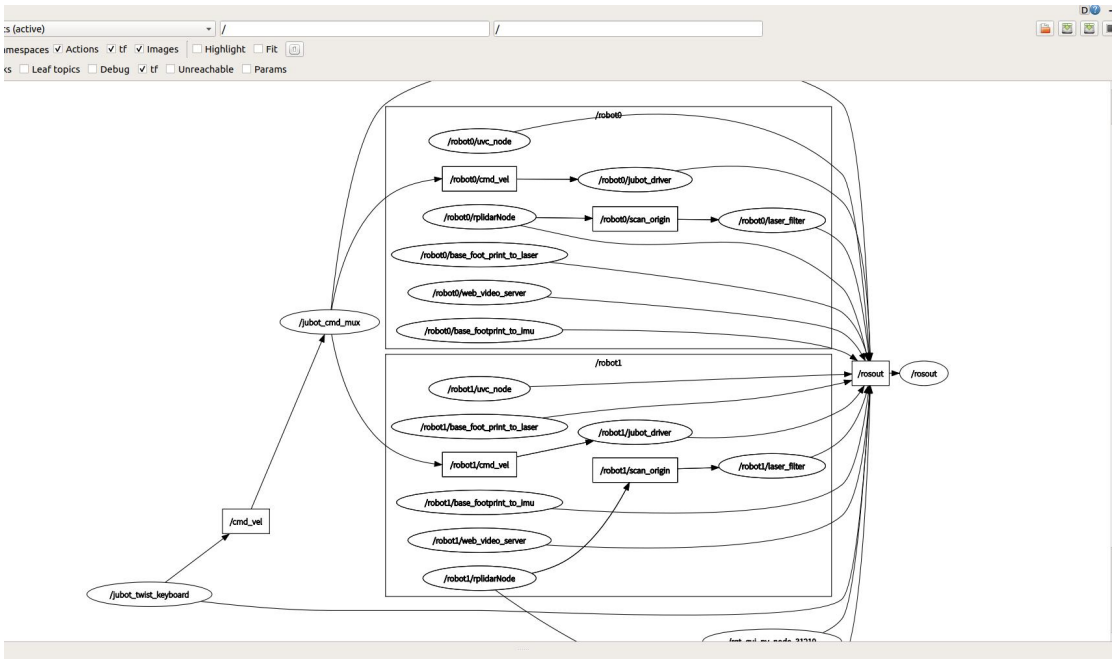
jubot_multirobot 扩展包中的 jubot_ctl_multirobot 软件包可以实现利用键盘或者手柄来完成多台机器人的单独控制或同时控制。

### 30.4.2 单独控制

要进行机器人的控制，首先要根据上一章节（多机器人驱动）启动机器人的底层驱动程序，并设置好机器人的命名空间。

在虚拟机端启动键盘操控节点，在启动时，同样需要设置命名空间（namespace） 参数，以指定将要遥控哪一台机器人。

此时，即可使用键盘遥控 robot0 机器人运动。同时可以开启多个。键盘操控节点以实现通过不同终端控制不同机器人运动。运行节点图如下。




### 30.4.3 同时控制

要同时控制多台机器人运动，同样首先要根据上一章节（多机器人驱动）启动多台机器人的底层驱动程序，并设置好机器人的命名空间 。

在虚拟机端启动键盘同时遥控节点，此时不需要添加命名空间（namespace）参数。

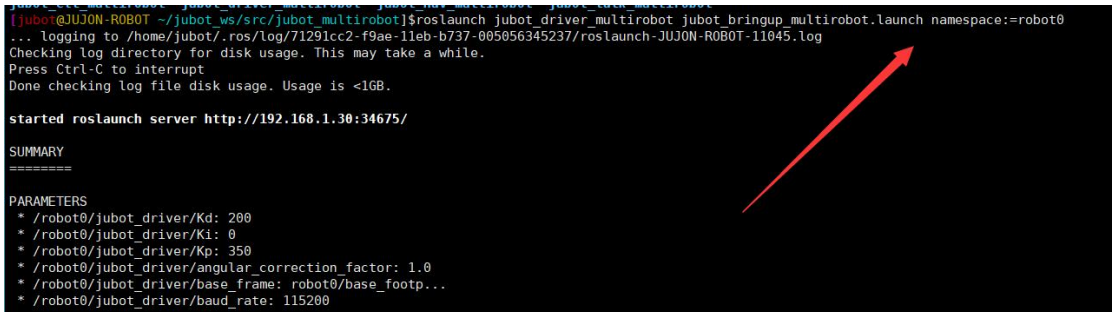
此时节点图如下所示。可以看到，键盘操控节点同时为两台机器人发布了 cmd_vel 控制话题，此时，通过键盘可以同时操控两台机器人同步运动。

### 30.4.4 多机器人导航

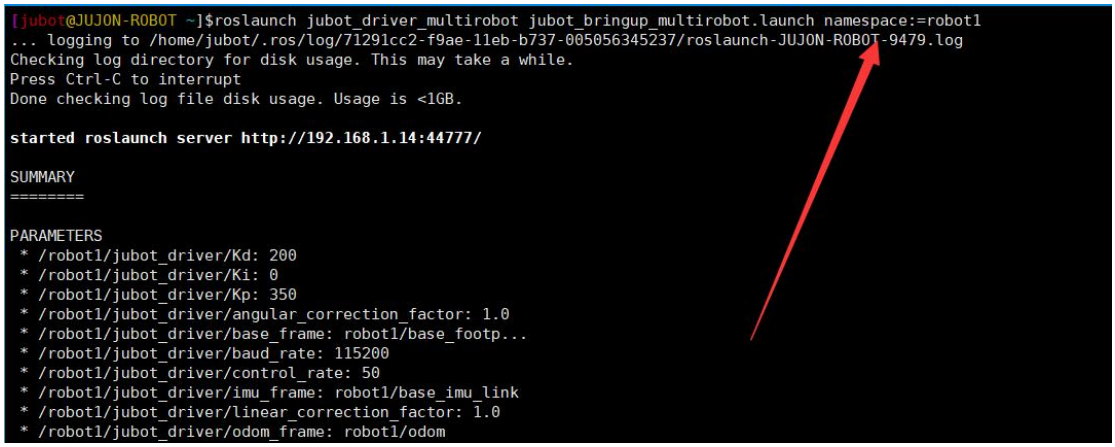
关闭这两个节点，测试多机器人同时控制。

在第一台和第二台机器人上启动多机器人控制节点：

```
roslaunch jubot_driver_multirobot jubot_bringup_multirobot.launch namespace:=robot0
```




```
roslaunch jubot_driver_multirobot jubot_bringup_multirobot.launch namespace:=robot1
```




在虚拟机端使用 `roslaunch jubot_ctl_multirobot jubot_keyboard_all.launch` 启动多机器人联动键盘控制节点，在该终端中点击上下左右等键，发现两台机器人同时运动。

```
jubot@JUJON-VM:~$ roslaunch jubot_ctl_multirobot jubot_keyboard_all.launch
... logging to /home/jubot/.ros/log/71291cc2-f9ae-11eb-b737-005056345237/roslaun
ch-JUJON-VM-31449.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://192.168.1.18:36585/

SUMMARY
========

PARAMETERS
 * /rosdistro: melodic
 * /rosversion: 1.14.11
 * /use_sim_time: False

NODES
  /
    jubot_cmd_mux (jubot_ctl_multirobot/cmd_mux.py)
    jubot_twist_keyboard (jubot_ctl_multirobot/jubot_twist_keyboard.py)

ROS_MASTER_URI=http://192.168.1.18:11311
```

```
/home/jubot/juvm_ws/src/jubot_multirobot/jubot_ctl_multirobot/launch/jubot_keybo
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  帮助(H)

Reading from the keyboard  and Publishing to Twist!
---------------------------
Moving around:
   u    i    o         ^
   j    k    l       < v >
   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%
```

## 30.5 实验结果

掌握多机器人网络配置、多机器人之间软件包应用

## 30.6 实验报告

实验目的

实验要求

实验内容

实验总结