XZD-P100AI 边缘计算盒子

一、产品概述

随着 5G+AI 成为数字化经济发展引擎，AI 赋能渗透也越来越广，AI 边缘计算，因低时延、稳定可靠、灵活拓展等优势，结合云边融合应用体系，成为新的数据赋能趋势；XZD-NV-P100 系列 AI 边缘算力智盒，是基于 NVIDIA Jetson Orin Nano 4/8G 嵌入式 ARM 架构、超强算力 SOC 芯片开发的 AI 边缘算力产品；拥有成熟完善、稳定可靠的 AI 软硬件开发环境和配套服务，让算法移植简单，产品工程化开发容易，从而大大缩短 AI 产品的落地周期；产品工业化设计，稳定可靠。

二、产品核心优势

●算法移植周期短：Nvidia jeston jetpack 一站式深度学习开发工具包：Ubuntu Linux OS、CUDA、cuDNN、OpenCV、tensorRT 等，支持 Caffe/TensorFlow/Pytorch/Mxnet/Paddle Lite 等主流深度学习框架；

●超高性能 AI 计算与编解码能力：GPU 综合算力强，最新的 NVIDIA AI 加速引擎 TensorRT；

●丰富的接口与灵活部署能力，AIoT 边缘联动：USB、HDMI、RS-485、RS232、SATA、RJ45、自定义 I/O 等；

●千兆网卡，支持网络 IP 设定内外网隔离等应用，适用于多种组网环境；支持 wiFi、4G/5G LTE 无线传输；

●支持 M.2 NVMe SSD 系统盘扩展，预留支持 SATA 2.5 寸硬盘；

●工业级产品，稳定可靠：紫铜片+冷凝硅脂散热性能优，防浪涌、防雷击，支持宽温工作环境；

三、产品应用场景

●智慧安防：社区/园区、校园、银行、城市综合体等；

●安全生产：智慧工地、工厂、仓储、加油站、消防应急等

●工业机器视觉：工业质检等；

●智慧交通：道路养护、全息路口，车路协同等；

●智慧金融：智慧网点；

●智慧巡检：电力巡检、无人值守等

●新零售：营业厅，商超，连锁等;

第一章 快速上手

一、学习路线

1..Jetson Orin nano 主板的学习路线

1. 本产品提供出厂镜像，可使用套餐自带的**固态硬盘**插入主板开机，不需要按照本教程重复烧录镜像，详细的镜像环境可以参考附录镜像里面的文档，如果烧录的是 jetson orin nano 官方的镜像则需要自己搭建相关的 AI 环境，因软件版本差异可能会出现报错，需自行想办法解决。烧录镜像的方法请看**第 2 章 jetson orin nano 基础教程**

2. 当你烧录其他第三方镜像，并顺利开机后。你如果想要对 jetson orin nano 进行一些简单的配置，比如：

- SD 卡扩容
- ssh\VNC 登录
- windowns 和 jetson orin nano 相互传输文件
- 增大 jetson orin nano 的空间
- 系统备份等操作 可以观看**第 3 章 系统基础设置教程**

3.当你想要了解 jetson orin nano 的 GPIO 口的用法，并具有一定的 python 基础可以观看**第 4 章 GPIO 硬件控制教程**

4.当想要学习人工智能视觉可以观看**第 5 章 AI 视觉进阶教程**

每一个不同颜色的框为一个部分，每一个部分都是依赖上一个部分的，很少是单独的，如果前面的环境没搭建好，就会报错，而使用的是我们的搭建好的镜像是可以直接运行里面 AI 的案例的。

5.当想要学习 ROS 的系统课程，可以观看**第 6 章 ROS 进阶教程**，掌握了 ROS 的知识后，可以购买我们的 ROS 配件进行进阶学习，我们搭建好的镜像也是有对应 ROS 配件的源码的。

二、 快速上手教程

1. 如使用配套的 nvme（固态硬盘），默认包含出厂镜像，开机即可使用，无需重复烧录镜像，如使用第三方的 nvme（固态硬盘），建议烧录我们提供的出厂镜像。如何烧录镜像，请观看**第二章 基础教程**

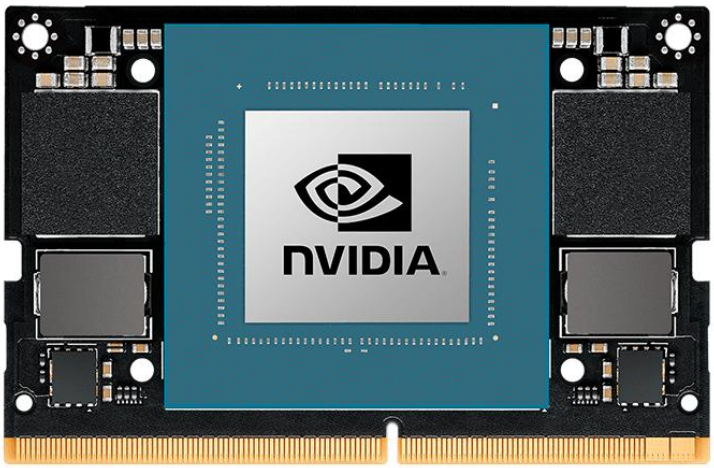
2. 当你镜像烧录完成并且 jetson Orin Nano 能正常开机，启动的具体过程可以看第二和第三章 的 **jetson Orin Nano 启动**，就可以进行以下的玩法了。

- 第 4 章 GPIO 硬件控制教程 主要是：**使用板载的 GPIO 驱动 LED、OLED 等 硬件**
- 第 5 章 AI 视觉进阶教程 主要是：**讲的是如何利用 GPU、CPU 来实现 AI 智能的操作**
- 其它的章节都是 jetson Orin Nano 和 ROS 系统的基础，如果你掌握了可以忽略。
- 本系统还搭载了我司其它的 ROS 配件的源码，如果你从我司购买到 ROS 配件， 可以根据对应的 ROS 配件教程进行多种玩法。

第二章 基础教程

一、Jetson Orin Nano 介绍

jetson Orin Nano 是 jetson 系列的一个新成员，核心板如下图所示



概述

Jetson Orin Nano 开发套件搭载了 8GB/4GB 版本内存的 Orin Nano 模组。

- Orin SoC 的 CPU 基于 Cortex-A78AE 架构。
- Orin Nano 8GB/4GB 中搭载了 6 个核心，最大频率 1.5 GHz
- GPU 方面为 Ampere 架构的 GA10B，搭载了了 1024 个 CUDA 核心和 32 个 Tensor Cores，最大频率为 625MHz。
- Orin Nano 只有硬件解码器，没有硬件编码器，编码能力只有 1080p30，由 1-2 个 CPU 核心提供支持。 从计算资源数量来看，Orin Nano 8GB 的规格与 Orin nano 8GB 版基本一致，为 AGX Orin 64GB 版的一半，不过频率更低，因此功率也更低。Nano 模组的功率只有 15W。在极低

的功率下，Orin Nano 仍提供了极强的性能，单精度浮点性能为 1.28 TFLOPs。**Orin Nano 8GB 和 Orin nano 8GB 规格非常接近，但除了频率和功率还有以下区别。**

orin nano 和 nano 之间的性能对比

Models	Jetson Nano (FPS)	Jetson Orin Nano 8GB (FPS)
PeopleNet (v2.5 unpruned)	2	116
Action Recognition 2D	32	368
Action Recognition 3D	1	26
LPR	47	979
Dashcam Net	11	398
Bodypose Net	3	136
Resnet50	36	1144

由图可以发现：Orin Nano 的性能是数量级上的提升。根据官方给出的数据，Orin Nano 达到了 每秒 40 万亿次运算(TOPS)的 AI 性能，是之前 nano 的 80 倍。

orin nano 4GB 和 8GB 的对比图

	Jetson Orin Nano 4GB	Jetson Orin Nano 8GB
AI Performance	20 TOPs	40 TOPs
GPU	512-core NVIDIA Ampere architecture GPU with 16 Tensor Cores	1024-core NVIDIA Ampere architecture GPU with 32 Tensor Cores
GPU Max Frequency	625 MHz	
CPU	6-core Arm® Cortex®-A78AE v8.2 64-bit CPU 1.5MB L2 + 4MB L3	
CPU Max Frequency	1.5 GHz	
Memory	4GB 64-bit LPDDR5 34 GB/s	8GB 128-bit LPDDR5 68 GB/s
Storage	- (Supports external NVMe)	
Video Encode	1080p30 supported by 1-2 CPU cores	
Video Decode	1x 4K60 (H.265) 2x 4K30 (H.265) 5x 1080p60 (H.265) 11x 1080p30 (H.265)	
Camera	Up to 4 cameras (8 via virtual channels***) 8 lanes MIPI CSI-2 D-PHY 2.1 (up to 20Gbps)	
PCIe*	1 x4 + 3 x1 (PCIe Gen3, Root Port, & Endpoint)	
USB*	3x USB 3.2 Gen2 (10 Gbps) 3x USB 2.0	
Networking*	1x GbE	
Display	1x 4K30 multi-mode DP 1.2 (+MST)/eDP 1.4/HDMI 1.4**	
Power	5W - 10W	7W - 15W
Mechanical	69.6mm x 45mm 260-pin SO-DIMM connector	

二、烧录系统和 SDK

1.打开 NVIDIA 的 jetpack 下载网址：

https://developer.nvidia.com/zh-cn/embedded/jetpack

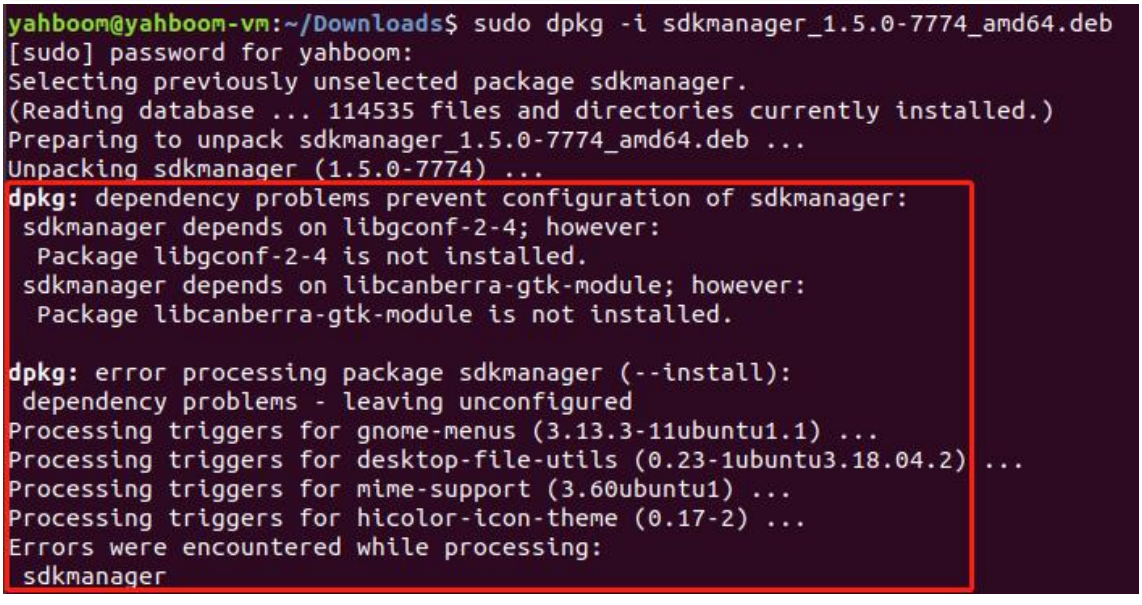
用虚拟机 Ubuntu18.04（20.04 也可以）系统，点击下载 SDK Manager，使用前请先注册/登录 NVIDIA 账号。

2.安装 SDK Manager。

先进入刚刚下载的.deb 文件的路径，例如这里下载到 Downloads 目录。

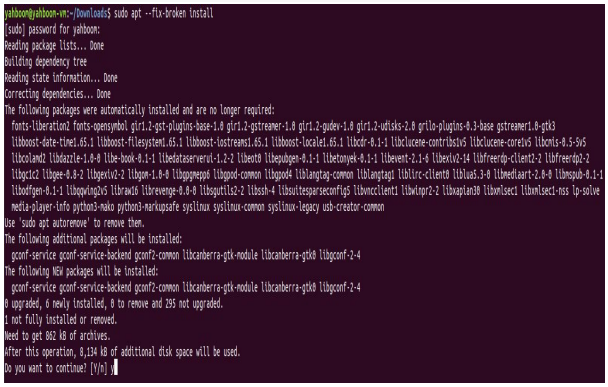
```
cd Downloads/
```


```
sudo dpkg -i sdkmanager_1.9.2-10889_amd64.deb
```

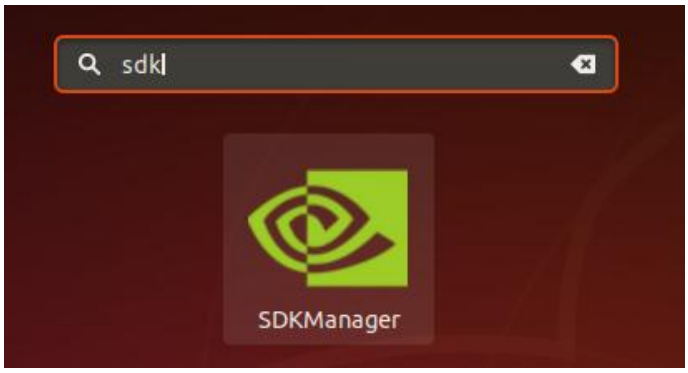


此时系统可能会报错找不到依赖文件，输入以下命令解决此问题。

```
sudo apt \--fix-broken install
```



3.打开 Ubuntu18.04 系统的程序，搜索 SDK，可以找到 SDKManager，打开文件。



登录 NVIDIA 账号，会在浏览器弹出链接，输入用户名和密码登录进去。

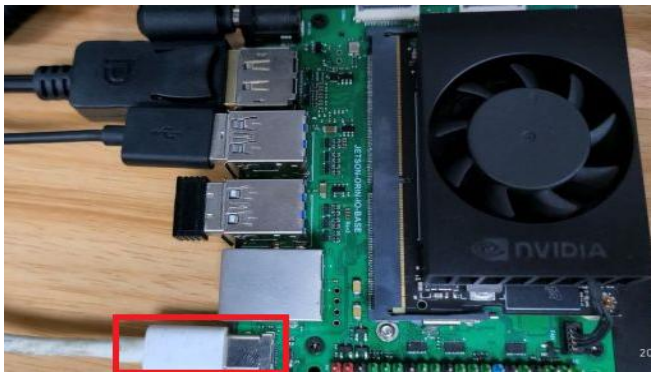
4.虚拟机 Ubuntu18.04 连接 jetson Orin nano

此时需要让 jetson Orin nano 进入系统 REC 刷机模式。

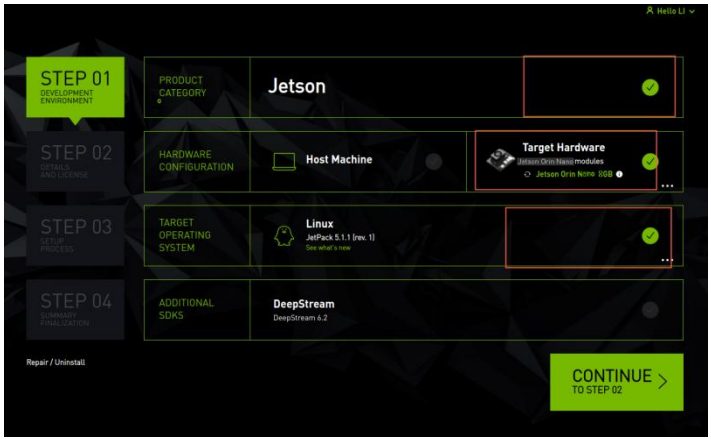
将跳线帽连接到 FC REC 和 GND 引脚，也就是连接到核心板下方载板的第二和第三个引脚，如下图所示：

连接线路，将 HDMI 显示屏、鼠标、键盘和 microUSB 数据线连接到 Jetson Orin nano 上，最后再接入电源。由于上一步已经将跳线帽连接 FC REC 和 GND 引脚，所以上电开机后会自

动进入 REC 刷机模式。



5.在虚拟机 Ubuntu18.04 的 SDKManager 软件选择 Target Hardware 为 Jetson Orin nano modules，JetPack 版本，这里以 5.1.1 版本为例。

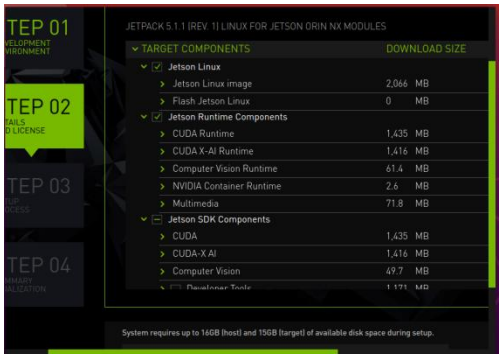


如果在 Target Hardware 显示未连接状态，请确认 nano 是否进入 REC 刷机模式并连接上虚拟机，然后点击 refresh 刷新。这里注意一下使用虚拟机需要把设备设置为连接到虚拟机上。

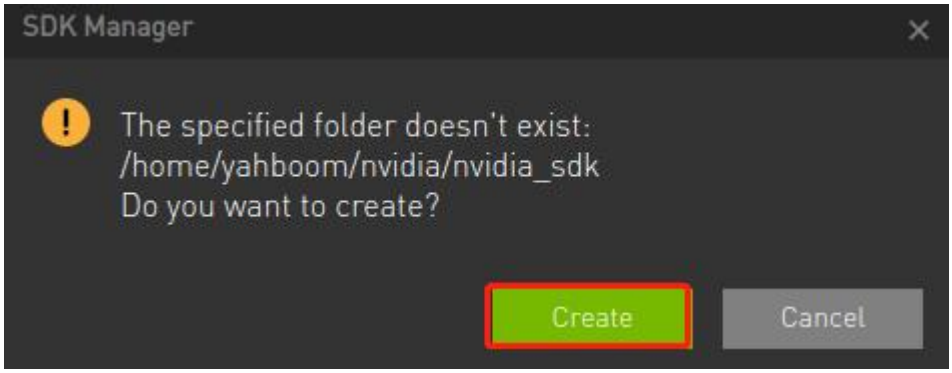
Jetson Orin nano 有两个版本，根据实际选择，8GB 的选 8GB module(不选有 developer)，4G 的选 4G module,一般插上后会自动识别出。

确认无误后点击 CONTINUE

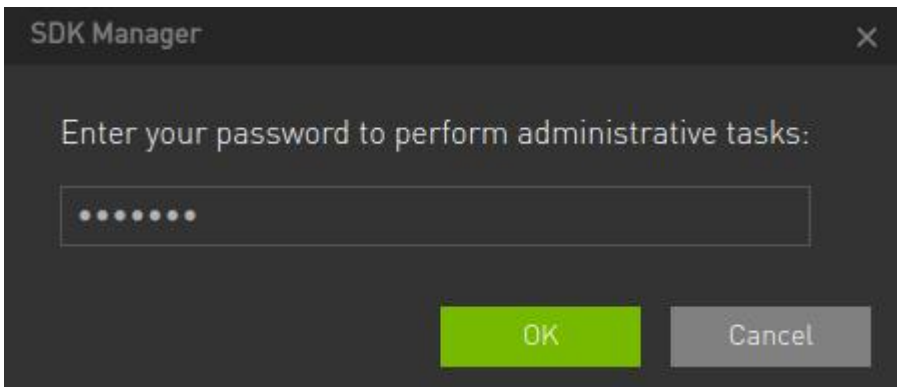
6.默认会勾选Jetson OS和Jetson SDK Components，表示刷入系统和SDK，可以单独选系统 OS 或者软件 SDK，但是单独刷入软件 SDK 前需要保证已经刷入系统 OS。



文件下载路径保持默认就好，勾选协议，点击 CONTINUE 进入下一步。

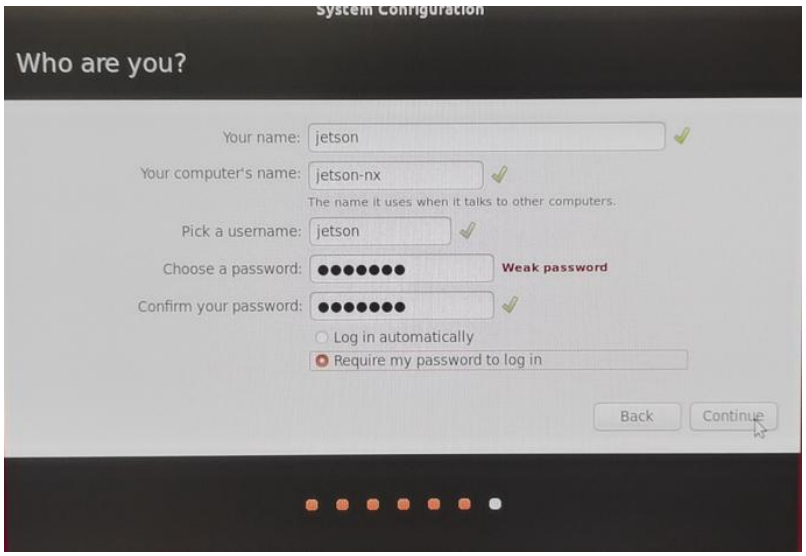


输入虚拟机的密码。



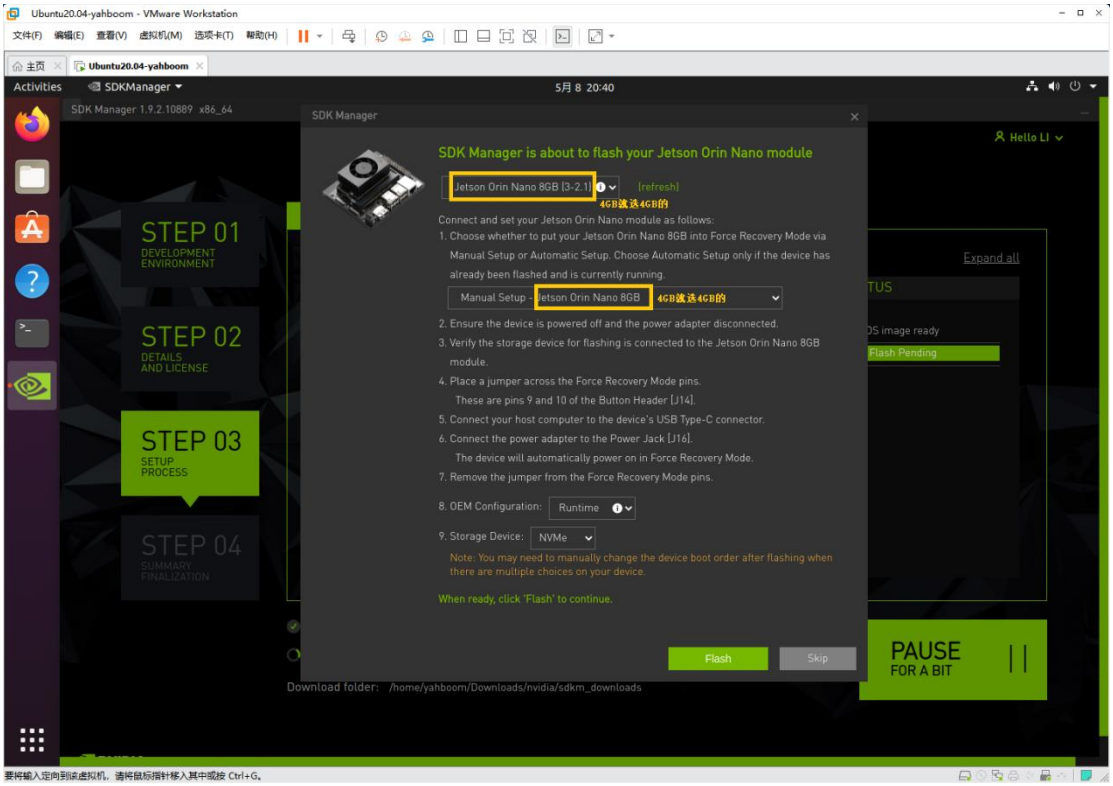
此时 SDKManager 会先下载需要烧录的文件，等待烧录文件下载完成即可开始烧录系统和 SDK。

7.等待系统 OS 烧录完成后，Jetson Orin nano 会自动重启进入系统，此时需要根据系统提示给设置系统的基本功能，必须执行的包括设置用户名与密码、连接与虚拟机同一个局域网的网络等。切换到 Jetson Orin nano 系统进行设置，这里的设置都比较简单，就不一一截图说明了，设置用户名和密码这里一定要牢记，否则会出现登录不进系统的问题。



8.系统设置完成后会再次重启 Jetson Orin nano，此时会与虚拟机断开连接，可以重新插拔一下 USB 数据线连接到虚拟机上。再输入刚刚设置的 Jetson Orin nano 的用户名和密码。点击安装即可进行安装软件 SDK。

下图是个 8GB 的



注意：由于刷入 SDK 需要用到局域网内的网络传输数据，为了稳定传输请插入网线。

完成后提示全部安装成功，点击 FINISH。如果在安装过程中出现过某个软件安装失败的情况，请点击重新安装即可。

9.注意：烧录完系统和 SDK 后，请将 FC REC 和 GND 之间的跳线帽拔掉。

3、固态盒子烧写系统

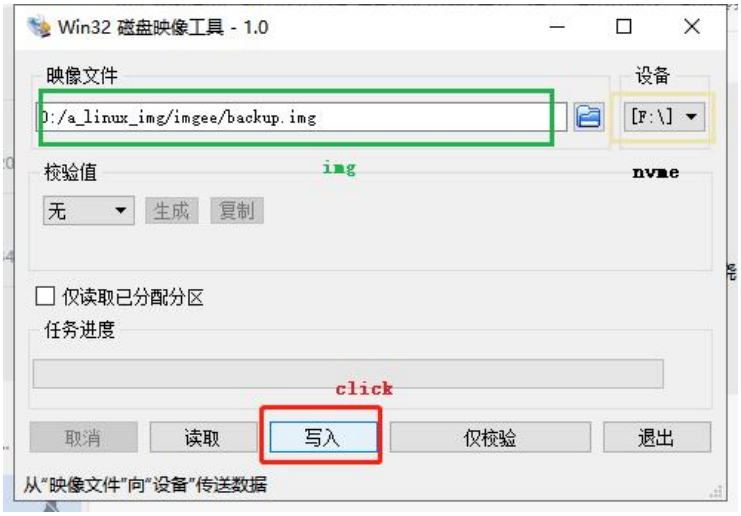
1. 首先从 jetson orin Nano 上拆下 nvme（固态硬盘）插进固态硬盘盒子，需要拧螺丝。

2. 固态硬盘盒子通过 usb 连接到电脑，如果 nvme 里面有东西，**请先进行格式化，再进行下面的操作**如何格式化看**已烧写固态重写 nvme 系统**的教程

3. 打开 Win32DiskImager 烧录工具

4. 如果固态硬盘格式化没进行分区，烧录工具是无法读出盘符，则需要进行分区工具进行分区，如何分区看**已烧写固态重写 nvme 系统**的教程

5. 如果已经分区了，选择镜像，选好盘符，点击 write/写即可，如图



6. 等待镜像烧录完成，把 nvme 插回 jetson orin Nano，上电一段时间后就可以成功开机了。

4、备份 nvme 系统

1.备份准备

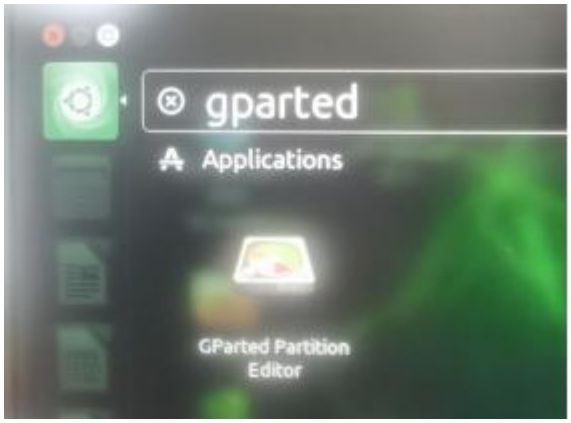
打开 Ubuntu 电脑（虚拟机）系统的终端，输入以下命令安装分区管理工具 gparted

sudo apt install gparted

```
yahboom@YAB:~$ sudo apt install gparted
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

将固态硬盘从主板上拔下来，插入连接到电脑（虚拟机），此时电脑会自动识别到固态硬盘（需固态硬盘盒子）。

打开电脑（虚拟机）的应用仓库，搜索并打开 gparted



注意：错误操作可能导致系统或文件无法异常，此步骤一定要选择正确的磁盘设备号。

2、记录分区信息

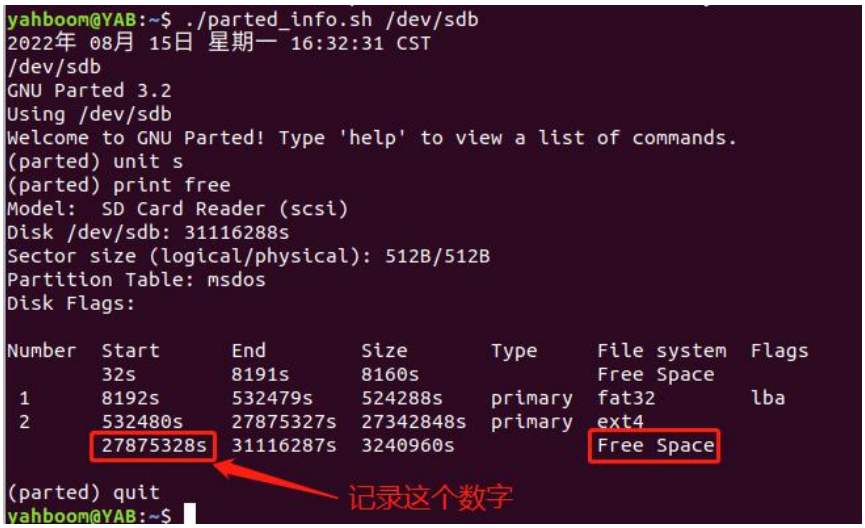
记录磁盘设备号：/dev/sdb

输入以下命令查看分区信息，记录 Free Space 那一行，Start 那一列的数字，**注意去掉最后的单位 s。**

```
./parted_info.sh /dev/sdb
```

其中/dev/sdb 参数必须指定，与上一步查看到的磁盘设备号一致。

如果 parted_info.sh 文件没运行权限，输入 chmod +x parted_info.sh 增加权限。



parted_info.sh 文件内容

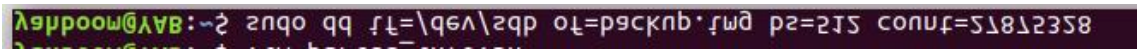
```
#!/bin/bash

date

echo $1

sudo parted $1    <<EOF

unit s

print free

quit

EOF
```

3、备份镜像 IMG 文件

打开系统终端，电脑（虚拟机）的储存空间必须大于镜像。

```
sudo dd if=/dev/sdb of=backup.img bs=512 count=27875328
```

其中 if=/dev/sdb 为第一步查询到的磁盘设备号，of=backup.img 为备份的名称，bs=512 表示块大小，count=27875328 表示备份大小，此数据从第二步中获取。



重新开启一个终端运行以下命令查看进度。

```
sudo watch -n 3 pkill -USR1 ^dd$
```

```
9897508864 bytes (9.9 GB, 9.2 GiB) copied, 609.736 s, 16.2 MB/s
9486+0 records in
9485+0 records out
9945743360 bytes (9.9 GB, 9.3 GiB) copied, 612.742 s, 16.2 MB/s
9524+1 records in
9524+1 records out
9987665920 bytes (10 GB, 9.3 GiB) copied, 615.451 s, 16.2 MB/s
yahboom@YAB:~$
```

等待备份完成即可。

4、拷贝镜像到 Windows

由于备份完的镜像在 Linux 电脑（虚拟机）中，此时需要打开虚拟机中的共享文件夹功能把文件传输到 Windows。

点击虚拟机的设置

依次点击‘选项’-‘共享文件夹’-‘总是启用’-‘添加’，然后添加 Windows 电脑的位置作为共享文件夹，下图中以添加到 D:\Virtual Machines\share 路径为例。点击确定保存配置。

将镜像文件复制到共享文件夹

```
sudo cp backup.img /mnt/hgfs/share/
```

到此为止，镜像备份完毕。

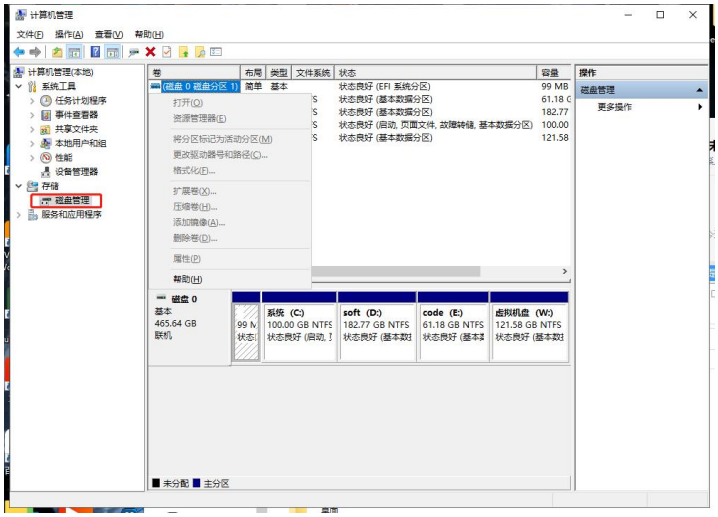
5、已烧写固态重写 nvme 系统

本教程用于：如果你想烧录其它版本的系统、jetson 官方系统，或者是想重新烧写 yahboom 版的系统到 nvme（固态硬盘）

1.因为 nvme 里面原本就有了系统，重新烧写之前需要对其格式化

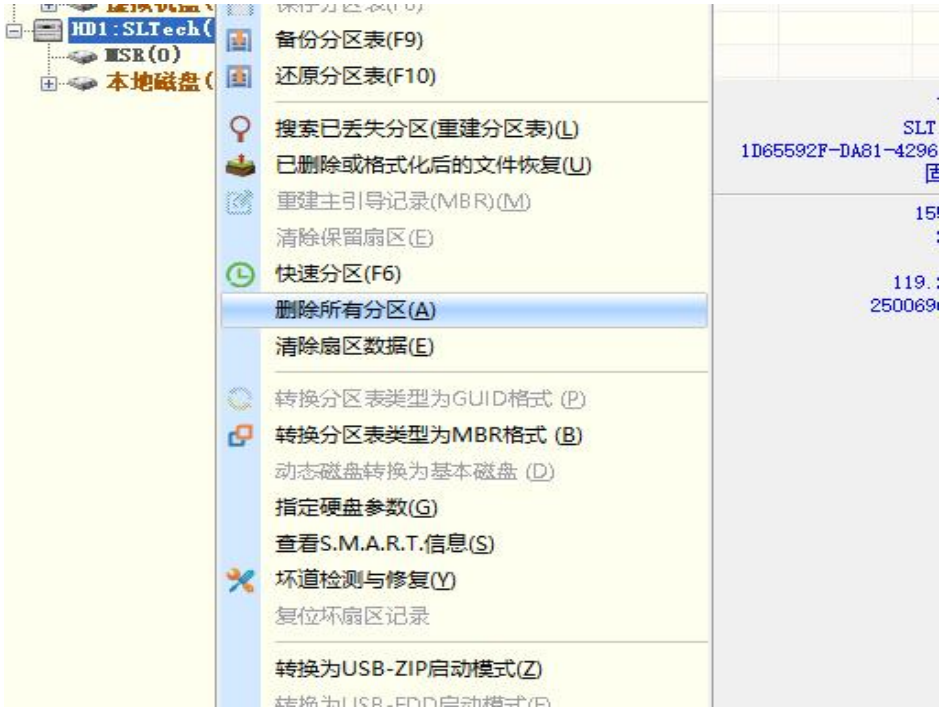
1. 先把固态硬盘从 jetson orin orin 中拆下来，然后插进固态硬盘的盒子，通过 usb 线连接上电脑。


2.在资料的附件中找到 DiskGenius.exe 该工具对 nvme 系统进行格式化的工具，或者用电脑里面的磁盘管理工具进行格式化。 **注意要选对 nvme 的盘，不要选错了盘符把自己电脑给格式化了**

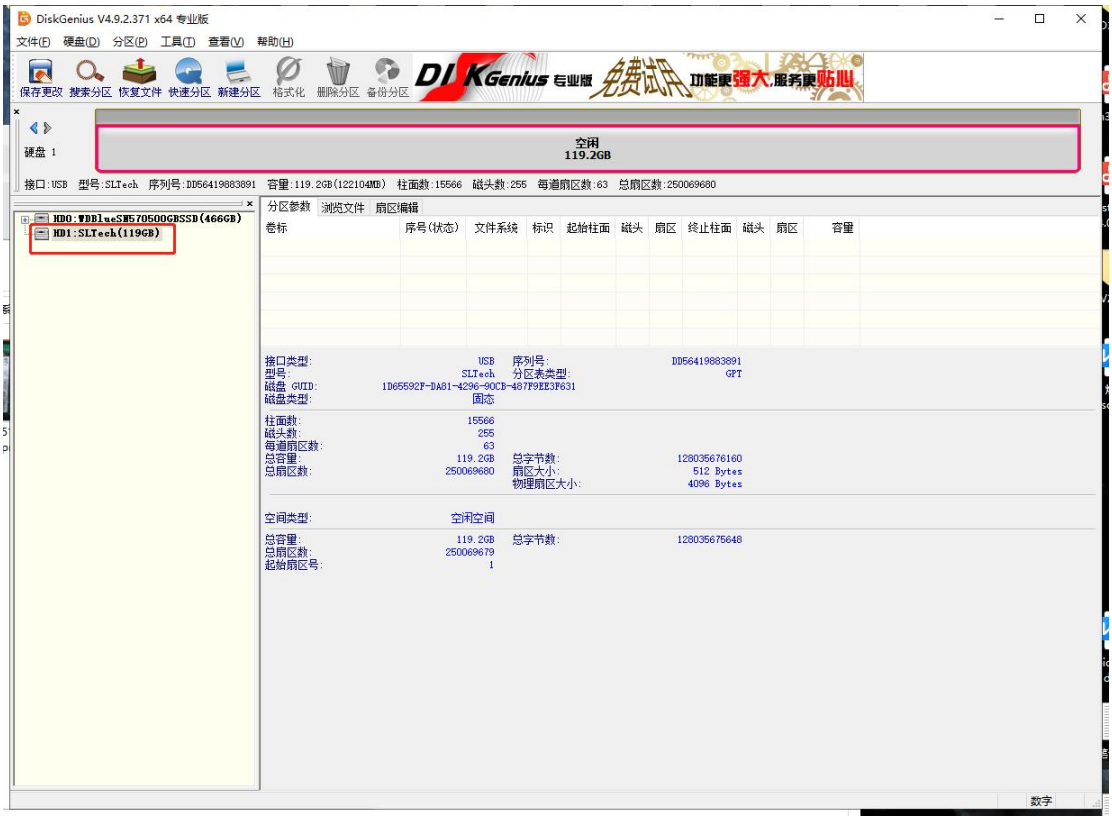


或者使用工具

2.使用 DiskGenius.exe 该工具对 nvme 删除所有多余的分区

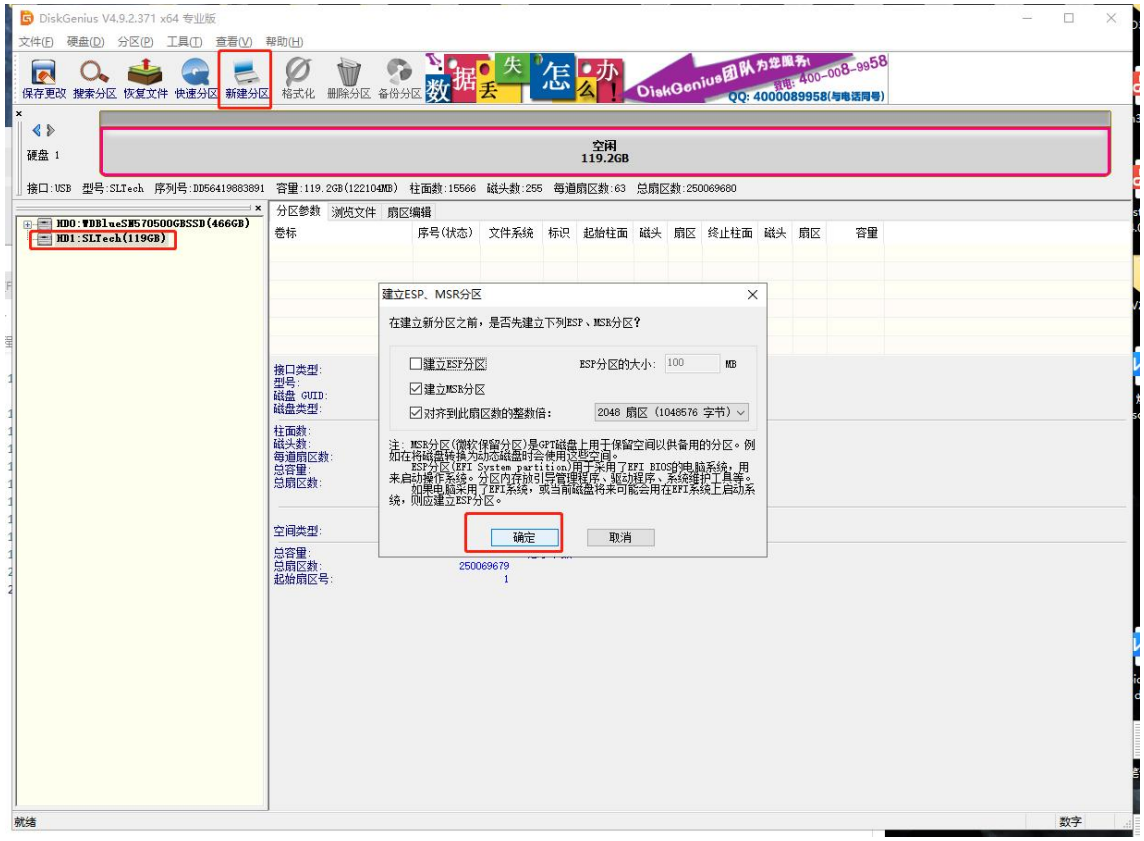


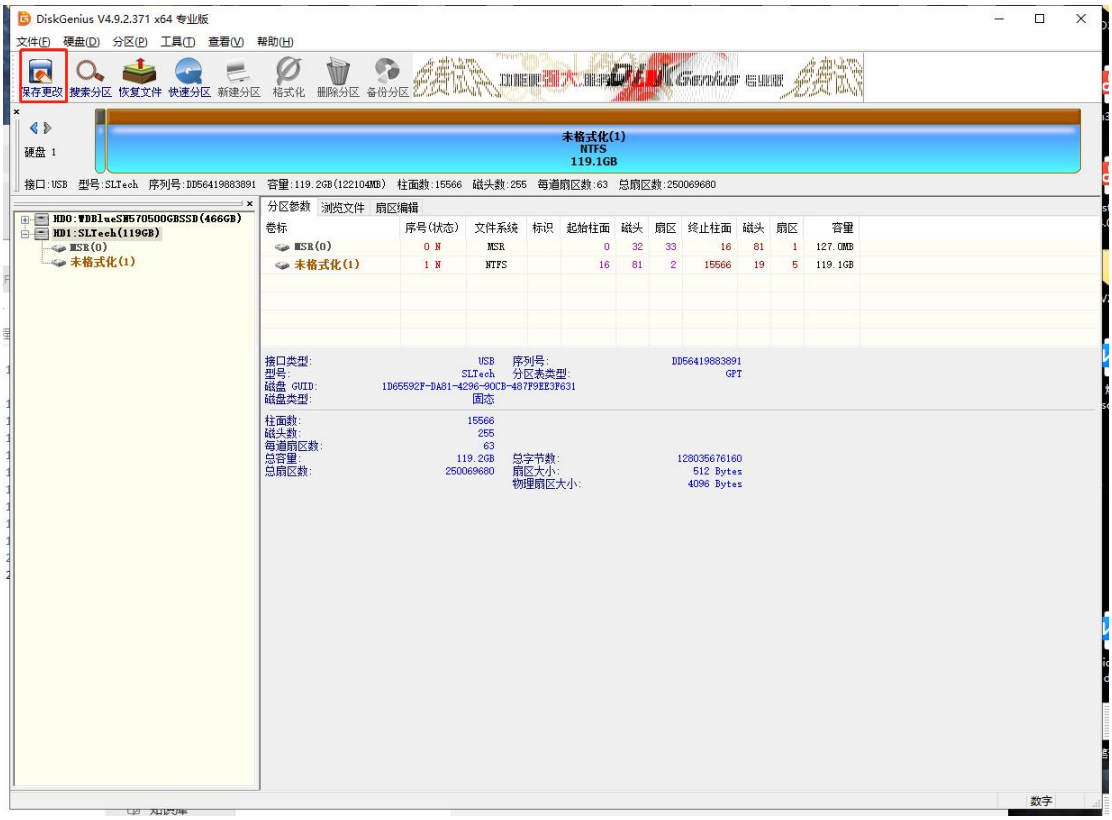
成功删掉后



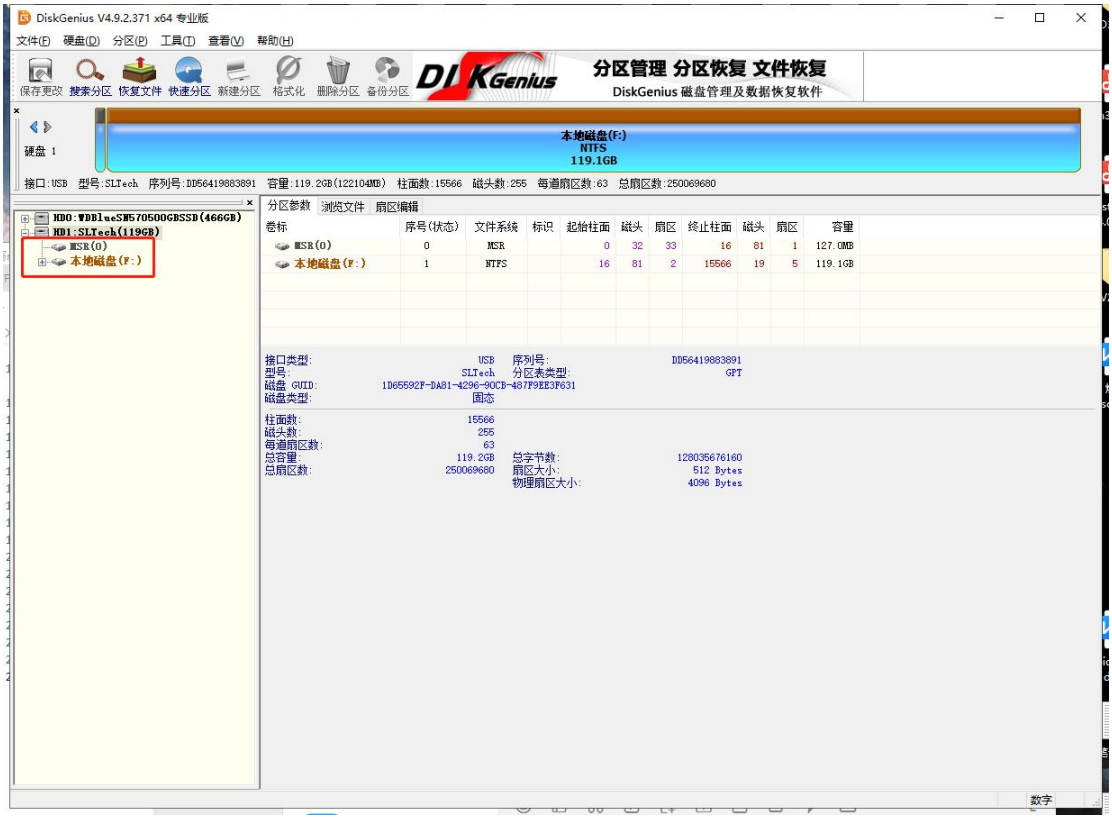
3.新建一个分区让电脑能正常识别的

有弹出信息，点确定就可以了，然后最好点击保存更改，如下图





成功后即可看到如下信息。



4.烧写系统

1. 如果你想直接下载 jetson 最原生的官方系统，可以看 **SDKmanager 烧写系统**这 个教程进行操作

2. 如果你想下载其它的第三方镜像或者 yahboom 版的镜像，可以看**固态盒子烧写 系统**这个教程进行操作

3. 其它烧写系统方法的参考链接

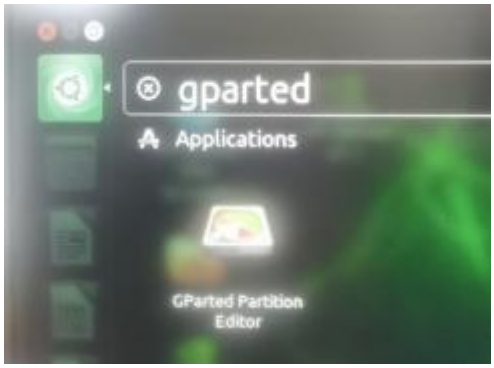
https://blog.csdn.net/weixin_48208348/article/details/129442788

6、 nvme 系统扩容

由于我们无法直接扩充正在运行的系统的空间，所以需要先将固态硬盘拆下来，放入固态硬盘盒，然后连接到电脑（虚拟机）再进行操作。

本教程使用的是 128G 固态硬盘，套餐固态现已升级为 256GB

1. 虚拟机打开 gparted

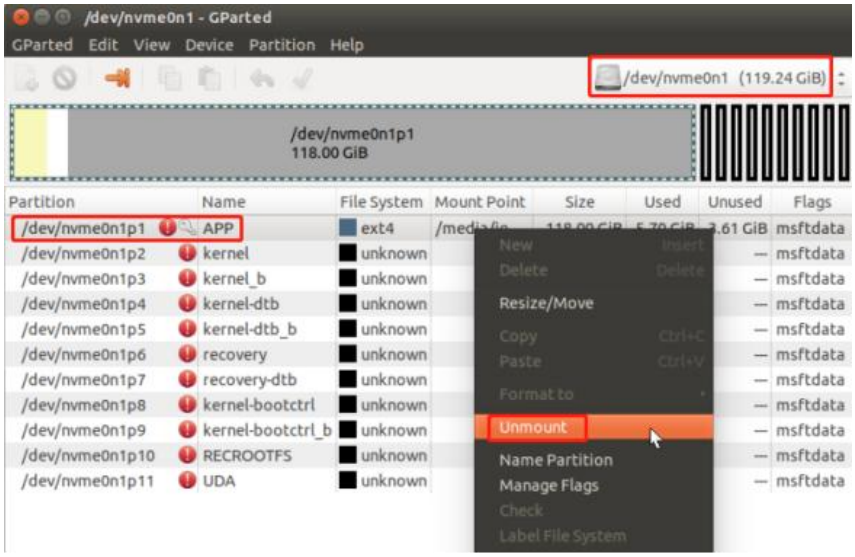


如果没有这个软件，先进行下载

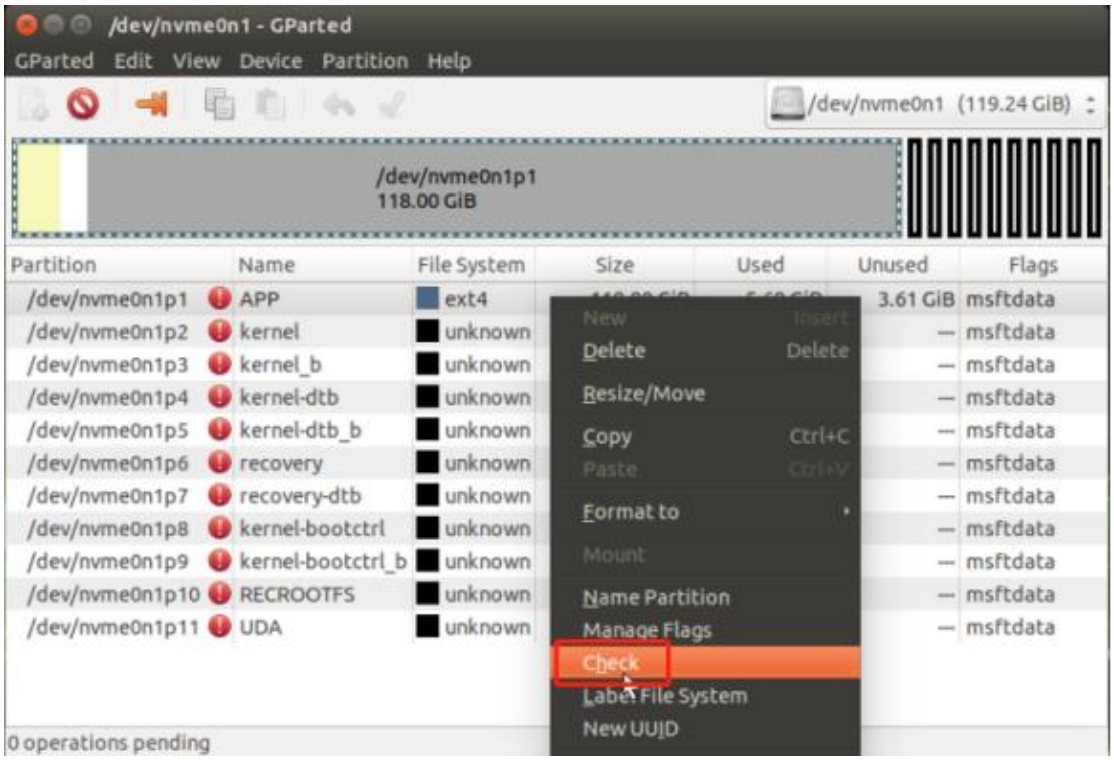
```
sudo apt install gparted
```

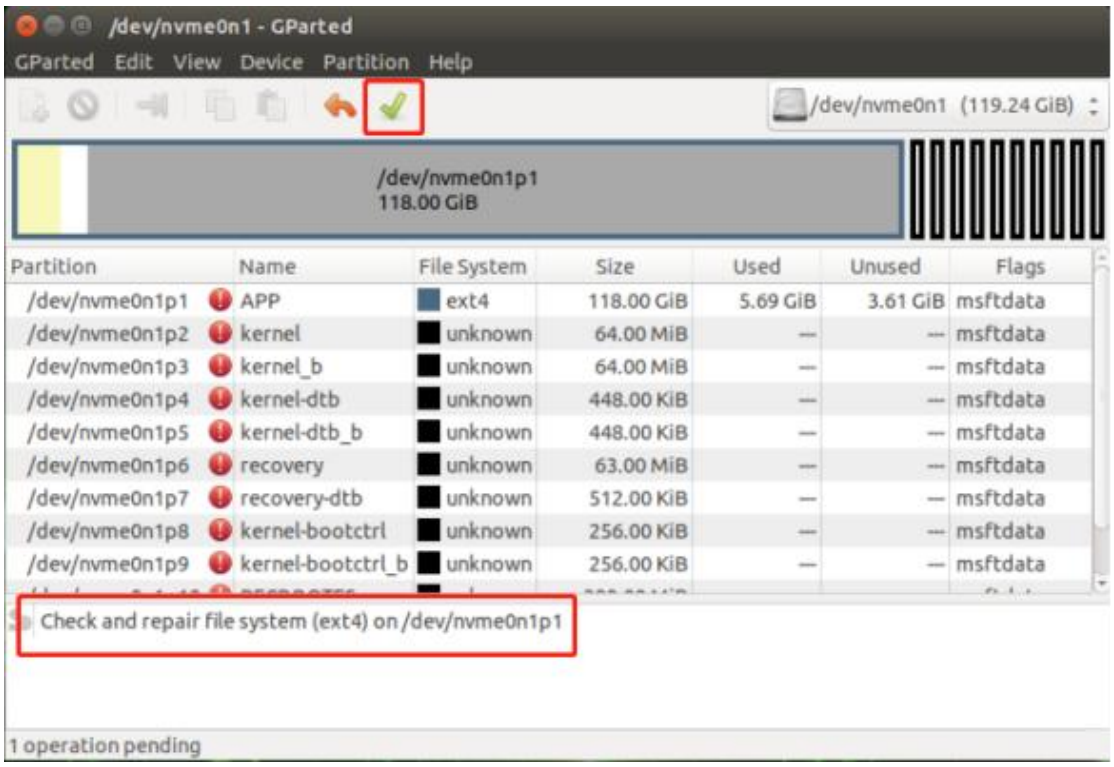
2. 选择对应的 NVME 硬盘/dev/nvme0n1（以实际为准），查看信息，APP 分区对应 Partition 为/dev/nvme0n1p1。注意：此步骤一定要选择正确的

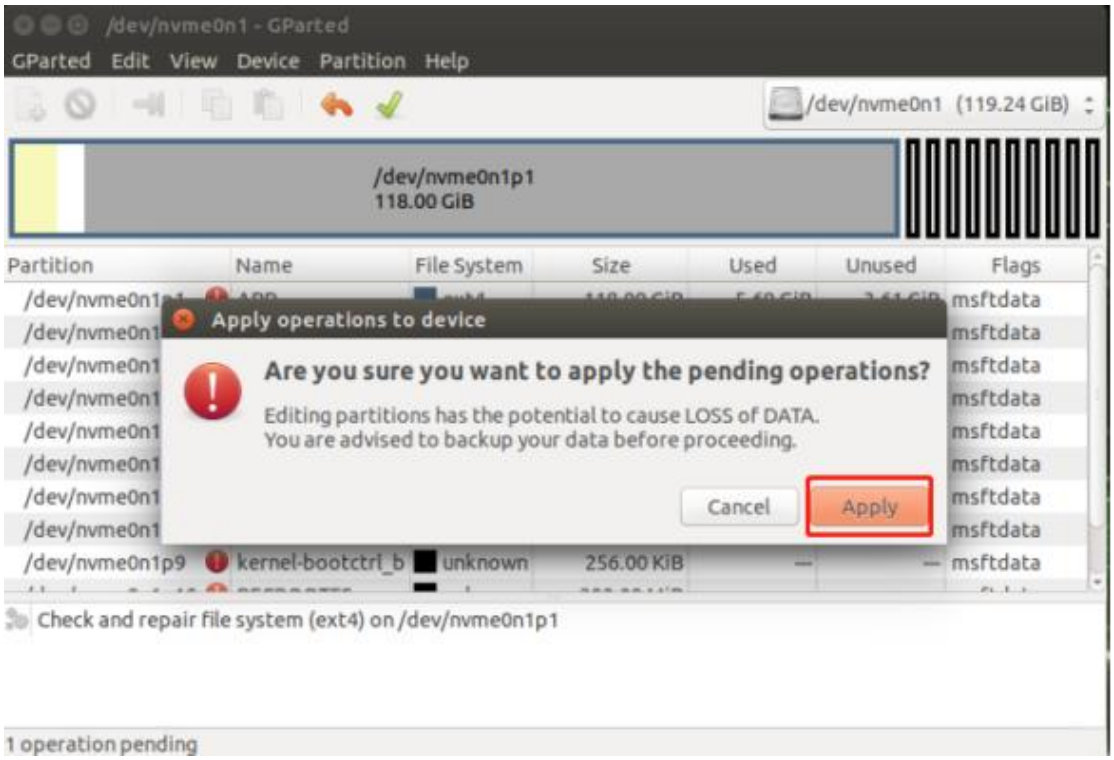
硬盘号。 可以看到 APP 分区有一部分是灰色，需要将灰色部分变为白色则为正常。颜色代表：黄色表示已使用空间，白色表示未使用空间，灰色表示不可用空间。这是由于恢复的系统是经过压缩得到的，所以内部空间需要重新检测一遍才能扩展到整个分区容量。 右键 APP 分区，点击 Unmount 卸载分区。



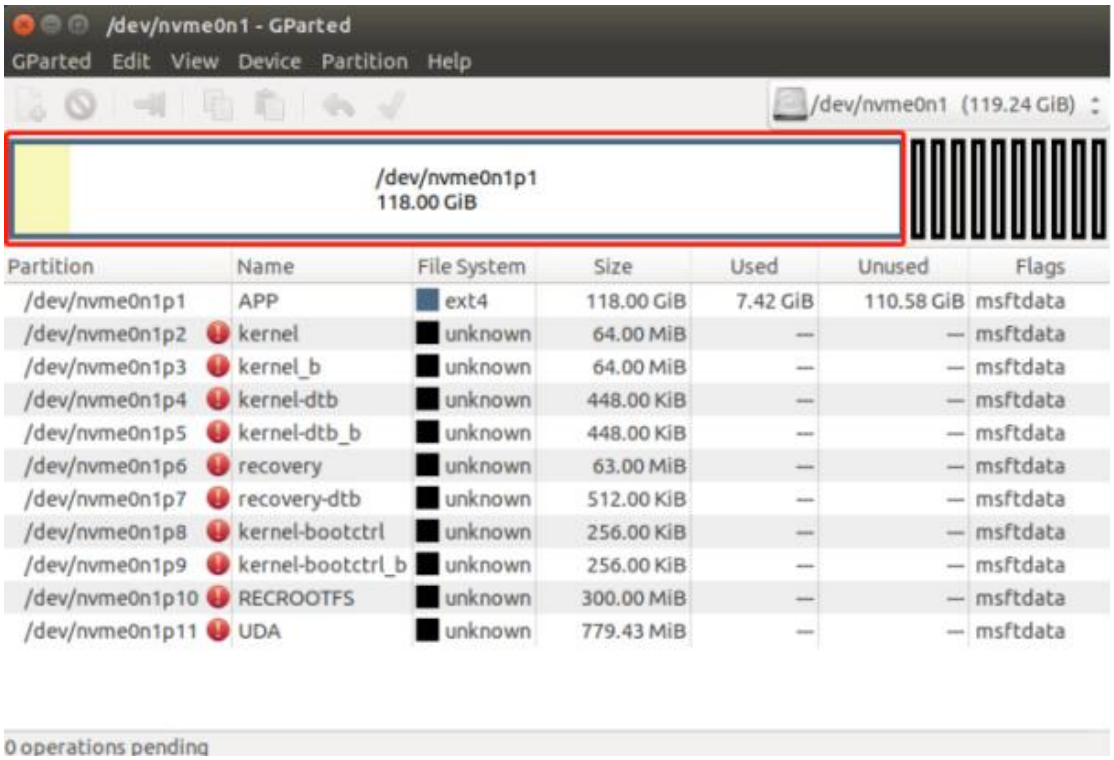
3. 选中 APP 分区，再次点击右键，选择 Check，然后根据提示完成操作就可以。


/dev/nvme0n1 - GParted
GParted Edit View Device Partition Help
/dev/nvme0n1 (119.24 GiB)
/dev/nvme0n1p1
118.00 GiB
Partition Name File System Size Used Unused Flags
/dev/nvme0n1p1 APP ext4 3.61 GiB msftdata
/dev/nvme0n1p2 kernel unknown — msftdata
/dev/nvme0n1p3 kernel_b unknown — msftdata
/dev/nvme0n1p4 kernel-dtb unknown — msftdata
/dev/nvme0n1p5 kernel-dtb_b unknown — msftdata
/dev/nvme0n1p6 recovery unknown — msftdata
/dev/nvme0n1p7 recovery-dtb unknown — msftdata
/dev/nvme0n1p8 kernel-bootctrl unknown — msftdata
/dev/nvme0n1p9 kernel-bootctrl_b unknown — msftdata
/dev/nvme0n1p10 RECROOTFS unknown — msftdata
/dev/nvme0n1p11 UDA unknown — msftdata
New Insert
Delete Delete
Resize/Move
Copy Ctrl+C
Paste Ctrl+V
Format to
Mount
Name Partition
Manage Flags
Check
Label File System
New UUID
0 operations pending


/dev/nvme0n1 - GParted
GParted Edit View Device Partition Help
/dev/nvme0n1 (119.24 GiB)
/dev/nvme0n1p1
118.00 GiB
Partition Name File System Size Used Unused Flags
/dev/nvme0n1p1 APP ext4 118.00 GiB 5.69 GiB 3.61 GiB msftdata
/dev/nvme0n1p2 kernel unknown 64.00 MiB — — msftdata
/dev/nvme0n1p3 kernel_b unknown 64.00 MiB — — msftdata
/dev/nvme0n1p4 kernel-dtb unknown 448.00 KiB — — msftdata
/dev/nvme0n1p5 kernel-dtb_b unknown 448.00 KiB — — msftdata
/dev/nvme0n1p6 recovery unknown 63.00 MiB — — msftdata
/dev/nvme0n1p7 recovery-dtb unknown 512.00 KiB — — msftdata
/dev/nvme0n1p8 kernel-bootctrl unknown 256.00 KiB — — msftdata
/dev/nvme0n1p9 kernel-bootctrl_b unknown 256.00 KiB — — msftdata
Check and repair file system (ext4) on /dev/nvme0n1p1
1 operation pending



4. 完成后就可以看到灰色部分变成白色了。



5.完成将固态硬盘从固态硬盘盒拆下来安装回去。

第三章 系统设置

1、jetson Orin Nano 的启动

1.将烧录好镜像的固态硬盘插进主板的固态硬盘的卡槽中，并将卡槽的螺丝拧紧

2.接入电源和 HDMI 线连接到显示器

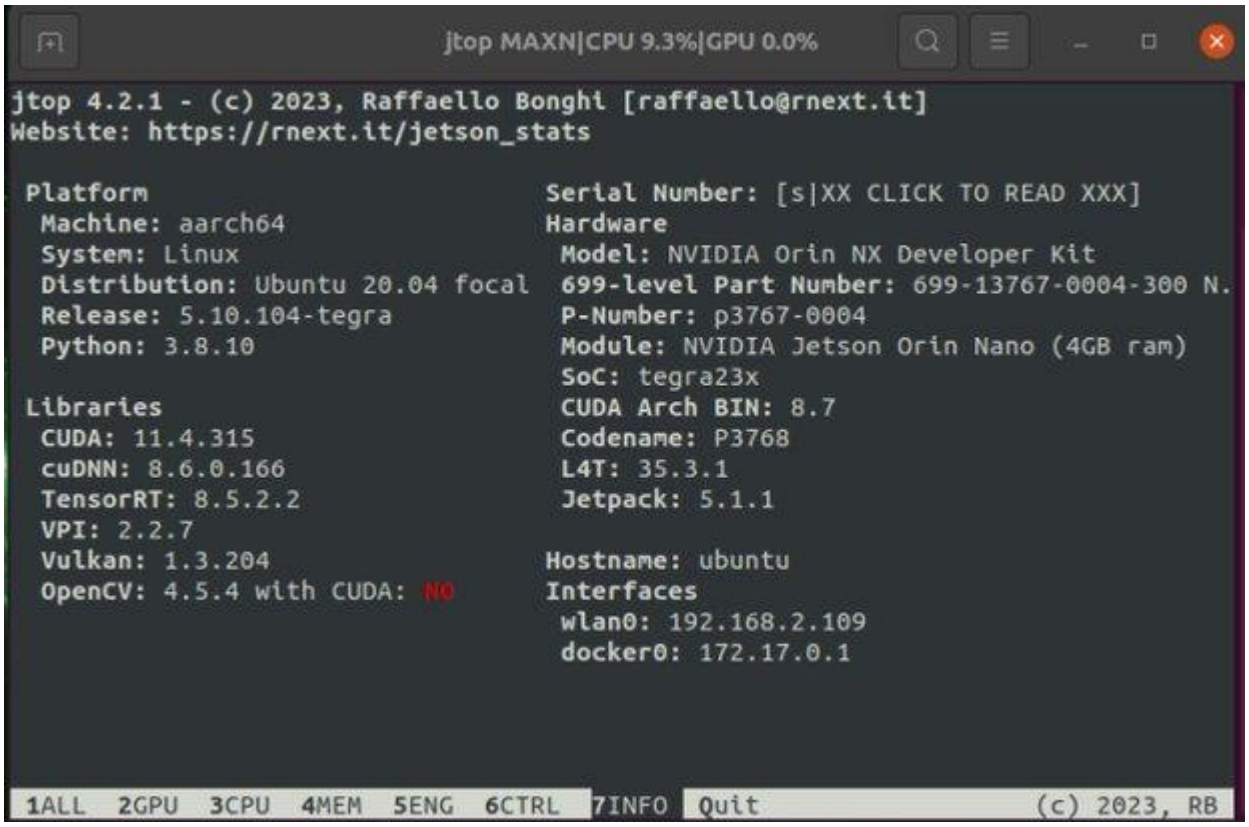
3.需要用官方配套的19V2.37A的圆头DC接口电源给Orin Nano板子供电。

4.官方电源是没有开关的上电就启动的，可以看到旁边的 DS1 指示灯也亮起，等待系统进入到桌面此过程接屏幕会显示在屏幕上。

2、Jetson Orin Nano 系统及桌面介绍

Jetson Orin Nano 系统

yahboom 搭载的系统是基于 ubuntu20.04 的 jetson 官方系统所搭建出的 yahbooom 版本的系统。系统的配置如下图所示：



可得到：

linux 64 位系统

ubtntu 20.04

python 3.8

CUDA 11.4

cuDNN 8.6

TensorRT 8.5

opencv 4.5.4

jetpack 5.1.1

jetson Orin Nano 密码：yahboom

桌面介绍

Linux 的桌面就很简单易懂了，只要正常开机，就能直接进入了桌面，通过连接鼠标、键盘、显示器，就能像操控电脑一样操作 jetson orin nano 了。下附一些详细的桌面介绍链接，本教程就不再阐述了 链接：https://blog.csdn.net/qq_33662195/article/details/127121722

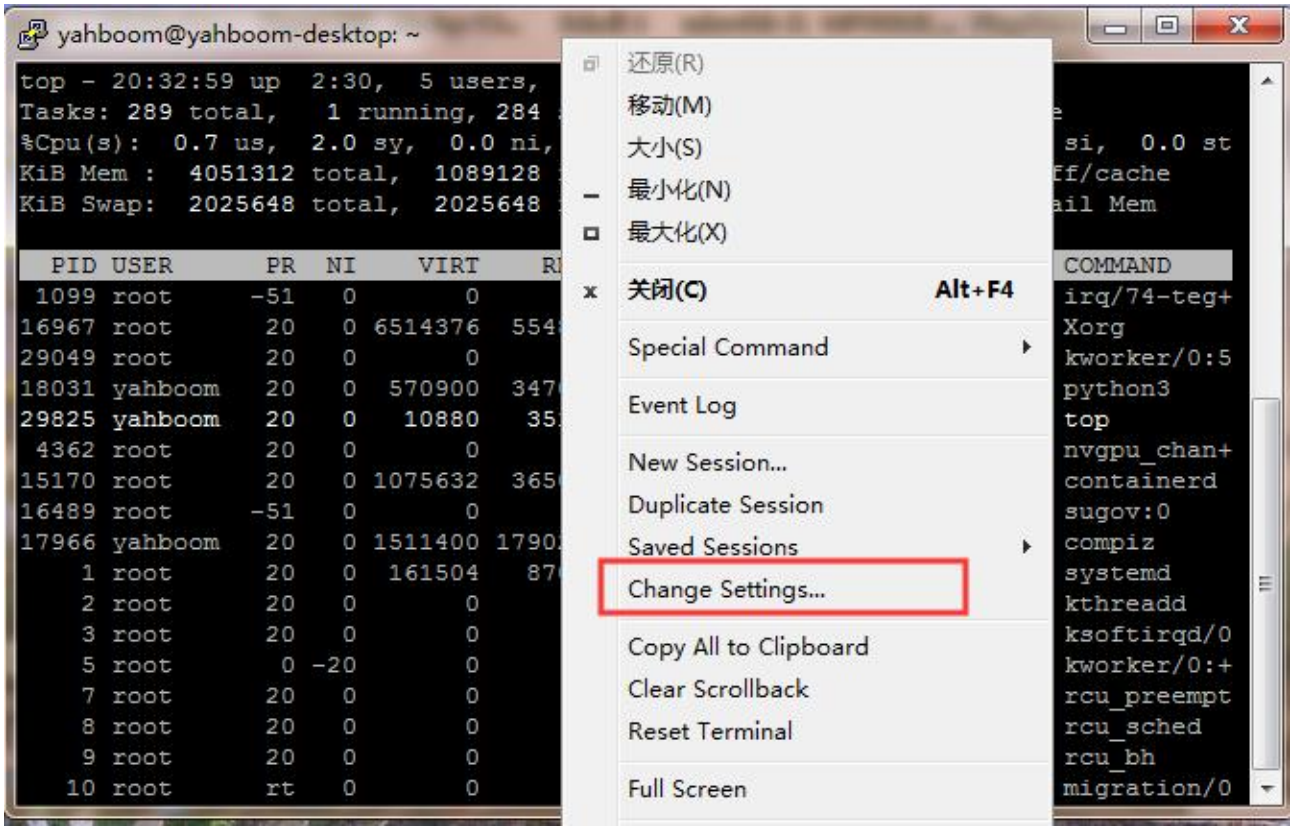
3、网络配置

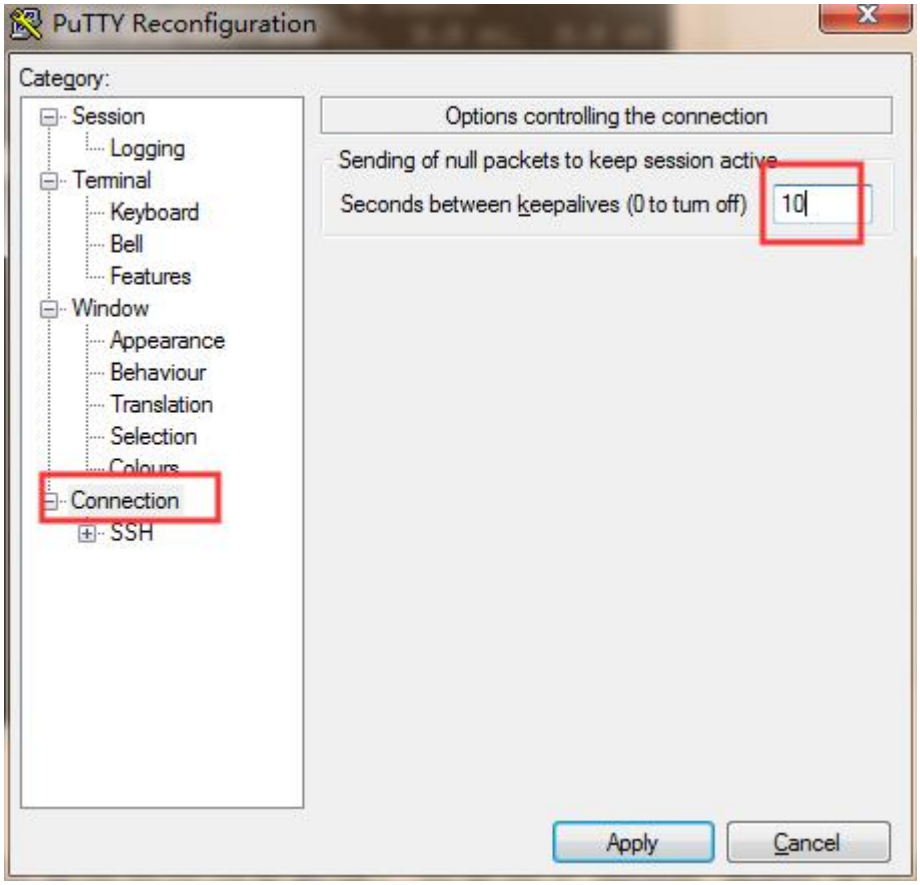
1、远程登录。

根据自己喜好选择 PuTTY、SSH、Xshell 等工具远程登录。以下以 PuTTY 工具举例。注意：如果发现电脑无法远程，可以尝试双方互相 ping 一下，nano 上查看 ip 地址命令：ifconfig。

Windows 下查看本地 ip 地址 cmd 命令：ipconfig。知道对方 ip 地址后，ping 192.168.1.xx 后面 ip 地址根据实际命令得出来的修改。

如果发现 putty 经常会自动掉线，可以尝试以下方法：


yahboom@yahboom-desktop: ~
top - 20:32:59 up 2:30, 5 users,
Tasks: 289 total, 1 running, 284
%Cpu(s): 0.7 us, 2.0 sy, 0.0 ni,
KiB Mem : 4051312 total, 1089128
KiB Swap: 2025648 total, 2025648
PID USER PR NI VIRT
1099 root -51 0 0
16967 root 20 0 6514376
29049 root 20 0 0
18031 yahboom 20 0 570900
29825 yahboom 20 0 10880
4362 root 20 0 0
15170 root 20 0 1075632
16489 root -51 0 0
17966 yahboom 20 0 1511400
1 root 20 0 161504
2 root 20 0 0
3 root 20 0 0
5 root 0 -20 0
7 root 20 0 0
8 root 20 0 0
9 root 20 0 0
10 root rt 0 0
si, 0.0 st
ff/cache
ail Mem
COMMAND
irq/74-teg+
Xorg
kworker/0:5
python3
top
nvgpu_chan+
containerd
sugov:0
compiz
systemd
kthreadd
ksoftirqd/0
kworker/0:+
rcu_preempt
rcu_sched
rcu_bh
migration/0
还原(R)
移动(M)
大小(S)
最小化(N)
最大化(X)
关闭(C) Alt+F4
Special Command
Event Log
New Session...
Duplicate Session
Saved Sessions
Change Settings...
Copy All to Clipboard
Clear Scrollback
Reset Terminal
Full Screen


PuTTY Reconfiguration
Category:
Session
Logging
Terminal
Keyboard
Bell
Features
Window
Appearance
Behaviour
Translation
Selection
Colours
Connection
SSH
Options controlling the connection
Sending of null packets to keep session active
Seconds between keepalives (0 to turn off)
10
Apply
Cancel

A.进入 putty，选择左侧的 Connection

B.在右侧有 Sending of null packets to keep session active

设为 10 即可

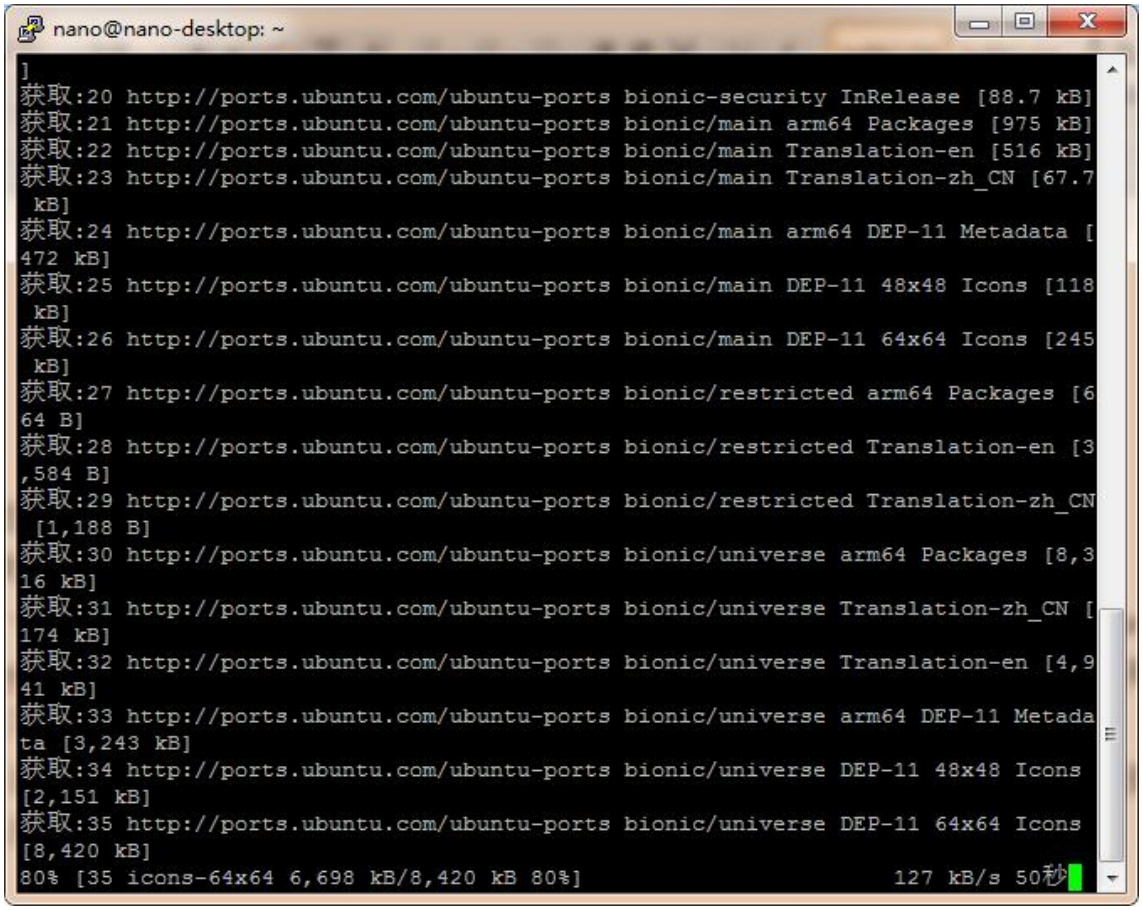
（意思每十秒发送一次空包用来保持连接）

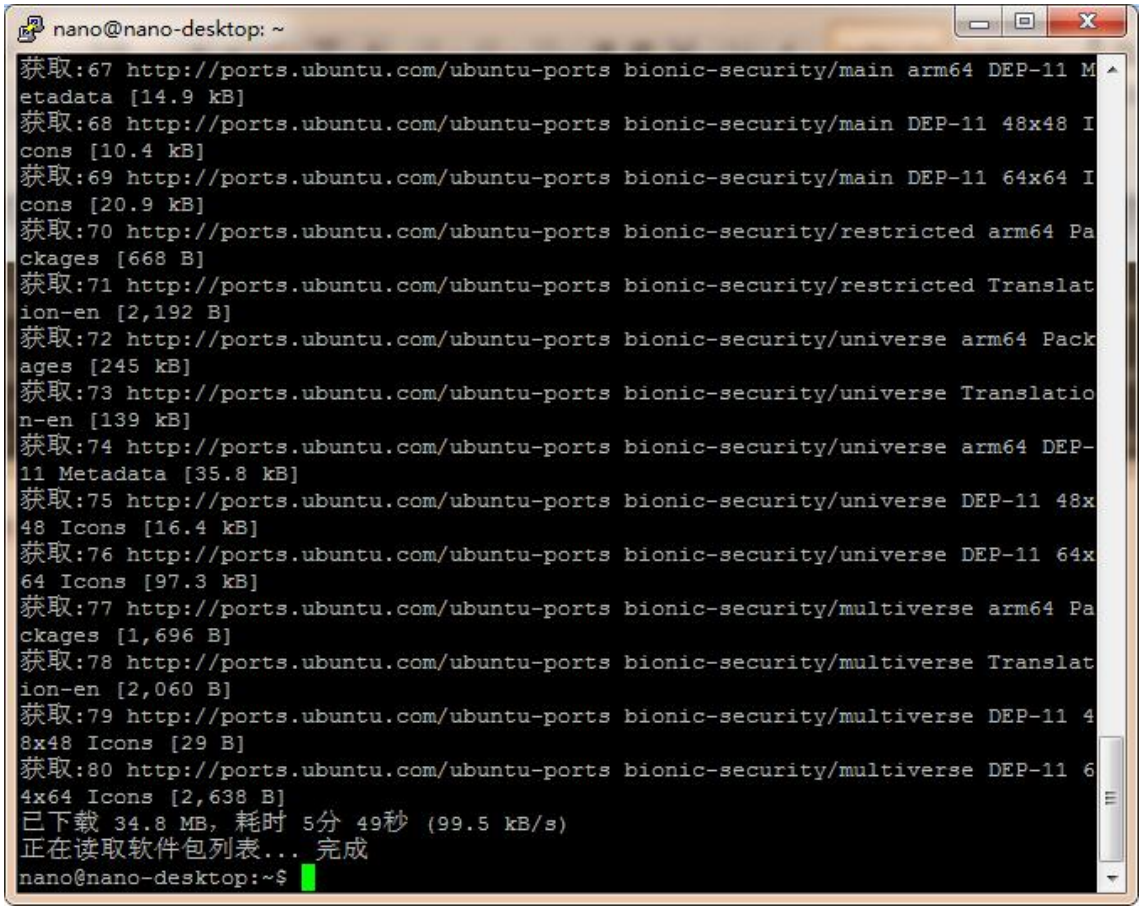
2、关于更新源。

一般来说，安装完系统后应当更新源，但是由于 Jetson Orin nano 采用的是 aarch64 架构的 Ubuntu 20.04.2 LTS 系统，与 AMD 架构的 Ubuntu 系统不同，而我没有找到完美的国内源，所以不推荐大家换源。

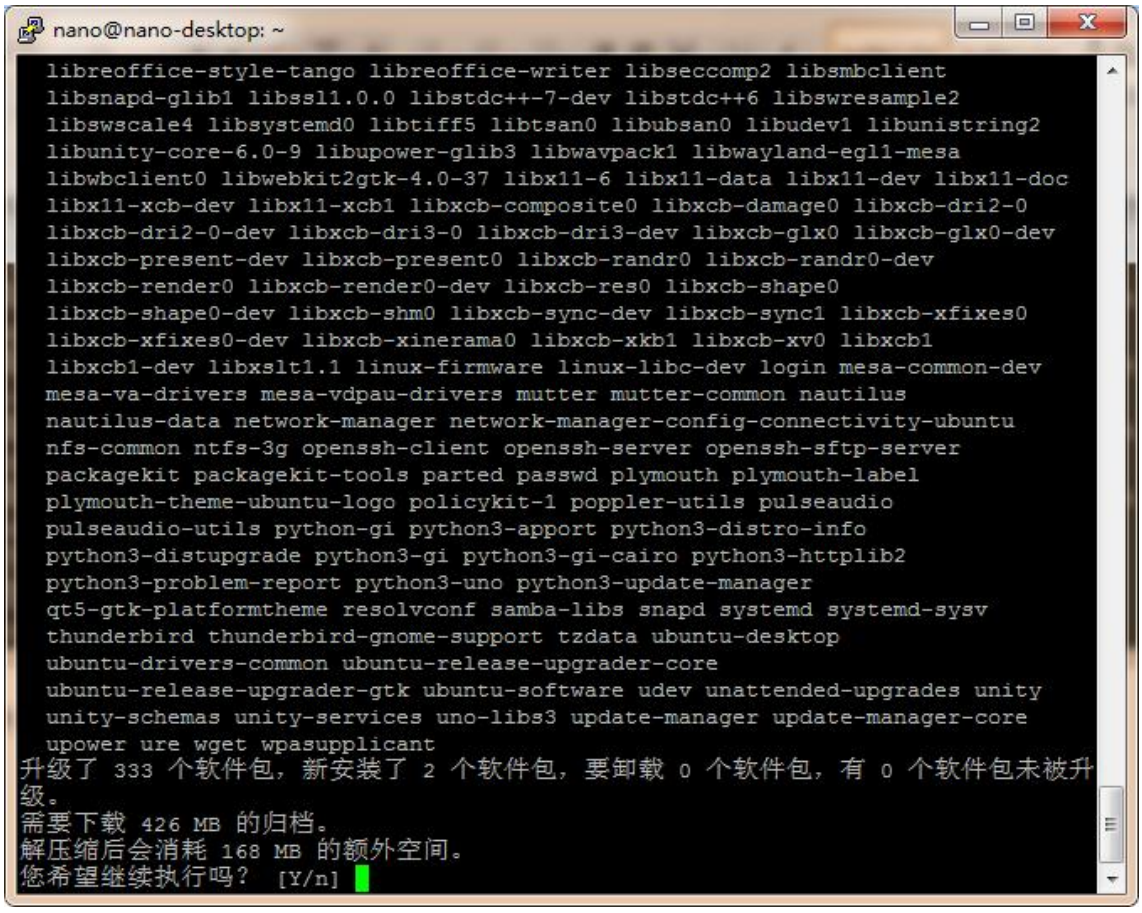
这里并没有换源，还是使用 Jetson Orin nano 的默认源进行更新。更新过程很漫长，大家可以执行完命令，做其他事吧。以下两个操作建议一定在做 AI 的项目前执行，否则安装一些库会找不到安装地址，导致后面频繁出错。

sudo apt-get update

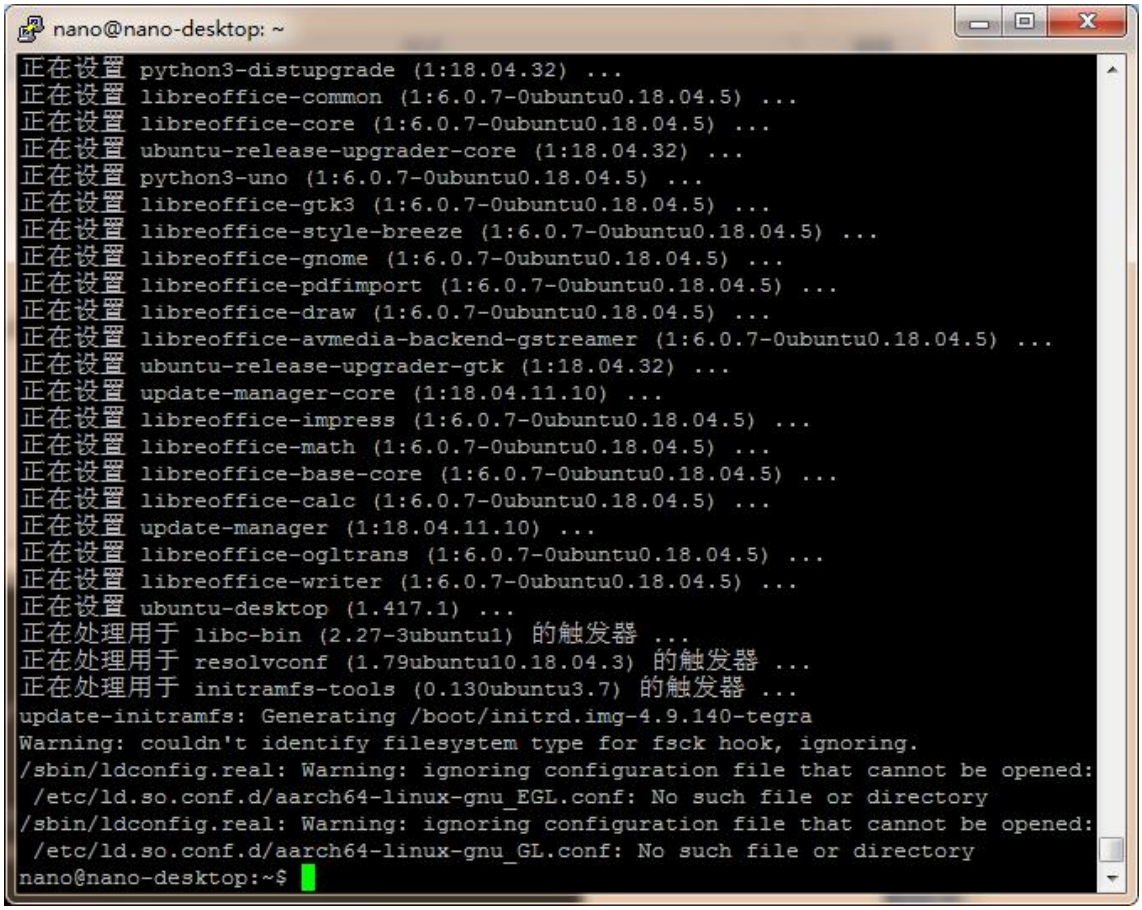

nano@nano-desktop: ~
]
获取:20 http://ports.ubuntu.com/ubuntu-ports bionic-security InRelease [88.7 kB]
获取:21 http://ports.ubuntu.com/ubuntu-ports bionic/main arm64 Packages [975 kB]
获取:22 http://ports.ubuntu.com/ubuntu-ports bionic/main Translation-en [516 kB]
获取:23 http://ports.ubuntu.com/ubuntu-ports bionic/main Translation-zh_CN [67.7
kB]
获取:24 http://ports.ubuntu.com/ubuntu-ports bionic/main arm64 DEP-11 Metadata [
472 kB]
获取:25 http://ports.ubuntu.com/ubuntu-ports bionic/main DEP-11 48x48 Icons [118
kB]
获取:26 http://ports.ubuntu.com/ubuntu-ports bionic/main DEP-11 64x64 Icons [245
kB]
获取:27 http://ports.ubuntu.com/ubuntu-ports bionic/restricted arm64 Packages [6
64 B]
获取:28 http://ports.ubuntu.com/ubuntu-ports bionic/restricted Translation-en [3
,584 B]
获取:29 http://ports.ubuntu.com/ubuntu-ports bionic/restricted Translation-zh_CN
[1,188 B]
获取:30 http://ports.ubuntu.com/ubuntu-ports bionic/universe arm64 Packages [8,3
16 kB]
获取:31 http://ports.ubuntu.com/ubuntu-ports bionic/universe Translation-zh_CN [
174 kB]
获取:32 http://ports.ubuntu.com/ubuntu-ports bionic/universe Translation-en [4,9
41 kB]
获取:33 http://ports.ubuntu.com/ubuntu-ports bionic/universe arm64 DEP-11 Metada
ta [3,243 kB]
获取:34 http://ports.ubuntu.com/ubuntu-ports bionic/universe DEP-11 48x48 Icons
[2,151 kB]
获取:35 http://ports.ubuntu.com/ubuntu-ports bionic/universe DEP-11 64x64 Icons
[8,420 kB]
80% [35 icons-64x64 6,698 kB/8,420 kB 80%] 127 kB/s 50秒


nano@nano-desktop: ~
获取:67 http://ports.ubuntu.com/ubuntu-ports bionic-security/main arm64 DEP-11 M
etadata [14.9 kB]
获取:68 http://ports.ubuntu.com/ubuntu-ports bionic-security/main DEP-11 48x48 I
cons [10.4 kB]
获取:69 http://ports.ubuntu.com/ubuntu-ports bionic-security/main DEP-11 64x64 I
cons [20.9 kB]
获取:70 http://ports.ubuntu.com/ubuntu-ports bionic-security/restricted arm64 Pa
ckages [668 B]
获取:71 http://ports.ubuntu.com/ubuntu-ports bionic-security/restricted Translat
ion-en [2,192 B]
获取:72 http://ports.ubuntu.com/ubuntu-ports bionic-security/universe arm64 Pack
ages [245 kB]
获取:73 http://ports.ubuntu.com/ubuntu-ports bionic-security/universe Translatio
n-en [139 kB]
获取:74 http://ports.ubuntu.com/ubuntu-ports bionic-security/universe arm64 DEP-
11 Metadata [35.8 kB]
获取:75 http://ports.ubuntu.com/ubuntu-ports bionic-security/universe DEP-11 48x
48 Icons [16.4 kB]
获取:76 http://ports.ubuntu.com/ubuntu-ports bionic-security/universe DEP-11 64x
64 Icons [97.3 kB]
获取:77 http://ports.ubuntu.com/ubuntu-ports bionic-security/multiverse arm64 Pa
ckages [1,696 B]
获取:78 http://ports.ubuntu.com/ubuntu-ports bionic-security/multiverse Translat
ion-en [2,060 B]
获取:79 http://ports.ubuntu.com/ubuntu-ports bionic-security/multiverse DEP-11 4
8x48 Icons [29 B]
获取:80 http://ports.ubuntu.com/ubuntu-ports bionic-security/multiverse DEP-11 6
4x64 Icons [2,638 B]
已下载 34.8 MB，耗时 5分 49秒 (99.5 kB/s)
正在读取软件包列表... 完成
nano@nano-desktop:~$

sudo apt-get full-upgrade

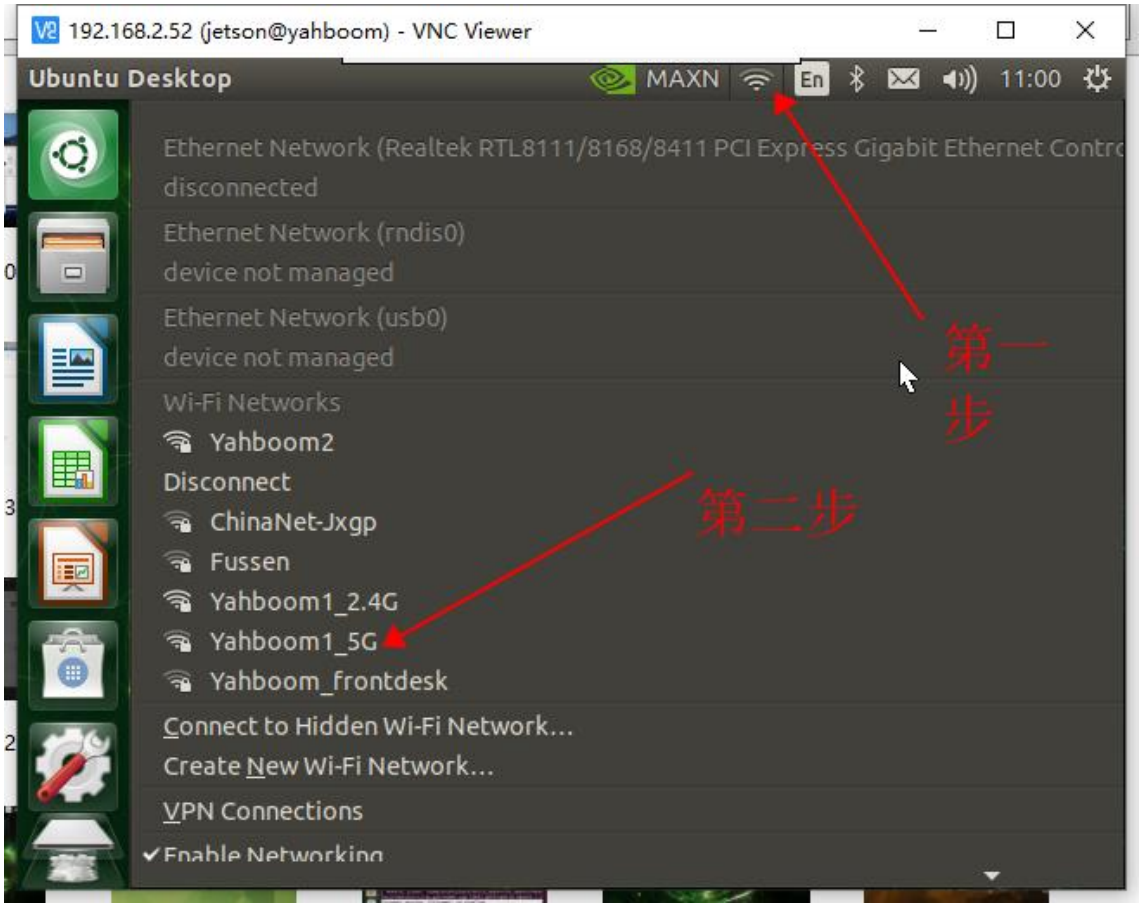


在过程中输入 Y 确认更新。第二个过程时间根据网络情况可能会花 2 个小时左右，请耐心等待。完成后如下图。



至此网络配置完成

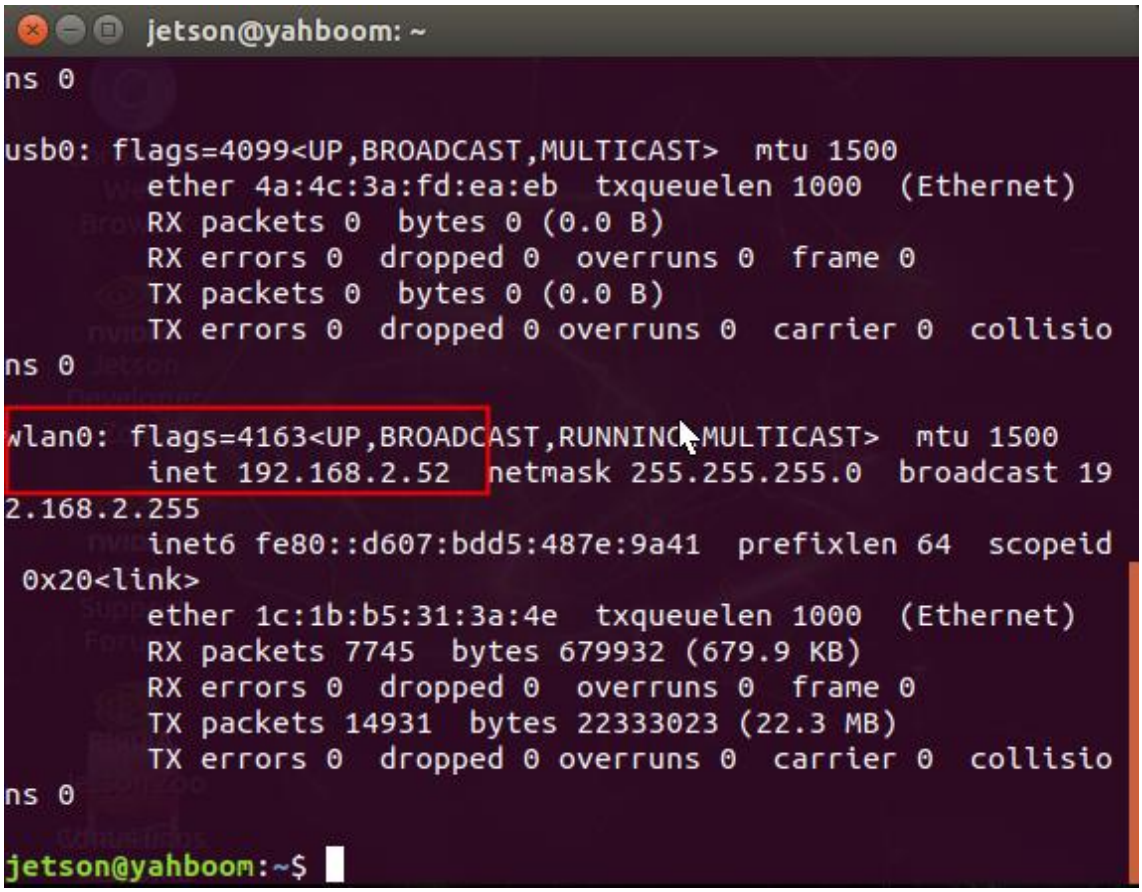
3.Jetson Orin nano 连接 wifi



第一步是点击上面的网络符号，第二步选择我们需要连接的 网络，输入密码即可，我这里是已经连接好 yahboom2 的网络了的

获取主板的 ip（连网的情况下）

```
ifconfig
```

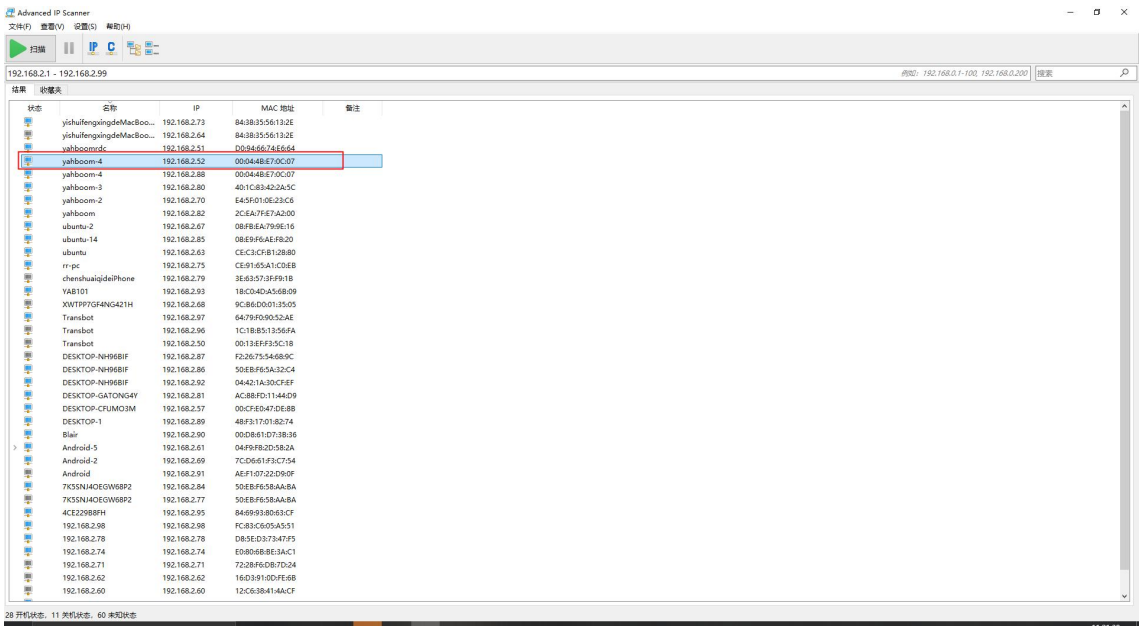


因为用的是 wifi 所以看 wlan0 这一行的 IP，可以发现我这里的 ip 是 192.168.2.52。

4.Jetson Orin nano 连接网线

要是我们没有显示屏的情况下想知道 ip，可以采取网线直插的方法，之后电脑也连同一个路由器的网落下，下载一个 ip 扫描软件来进行 ip 扫描就可以了 Advanced IP Scanner。

扫描的 ip



4、Jetson Orin Nano 远程登录教程&&远程传输文件教程

提示:配置好的镜像，用户名为 jetson 原始密码为:yahboom,

1.远程登录教程

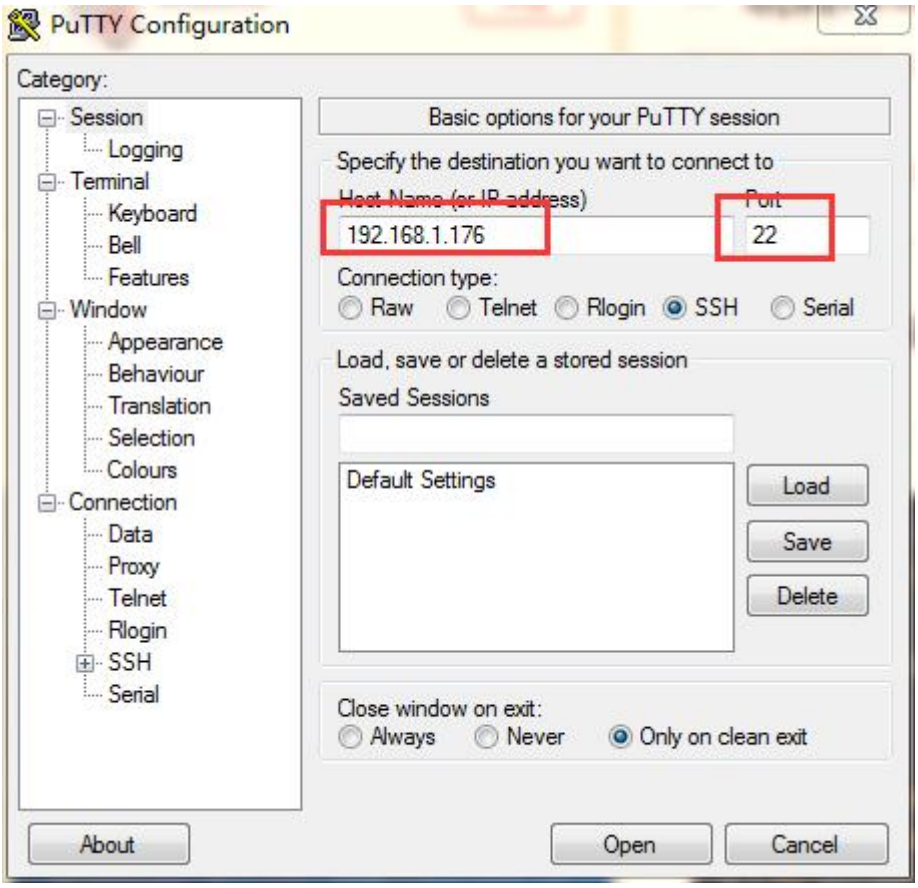
1、先确定自己板子的 ip 地址。

方法 1：直接在安装系统过程中，在界面通过 ctr+Alt+T 打开命令提示符，输入：ifconfig

找到对应的有线网卡 eth0 的 IP 地址，如果购买了无线网卡，请看 wlan 下地址。

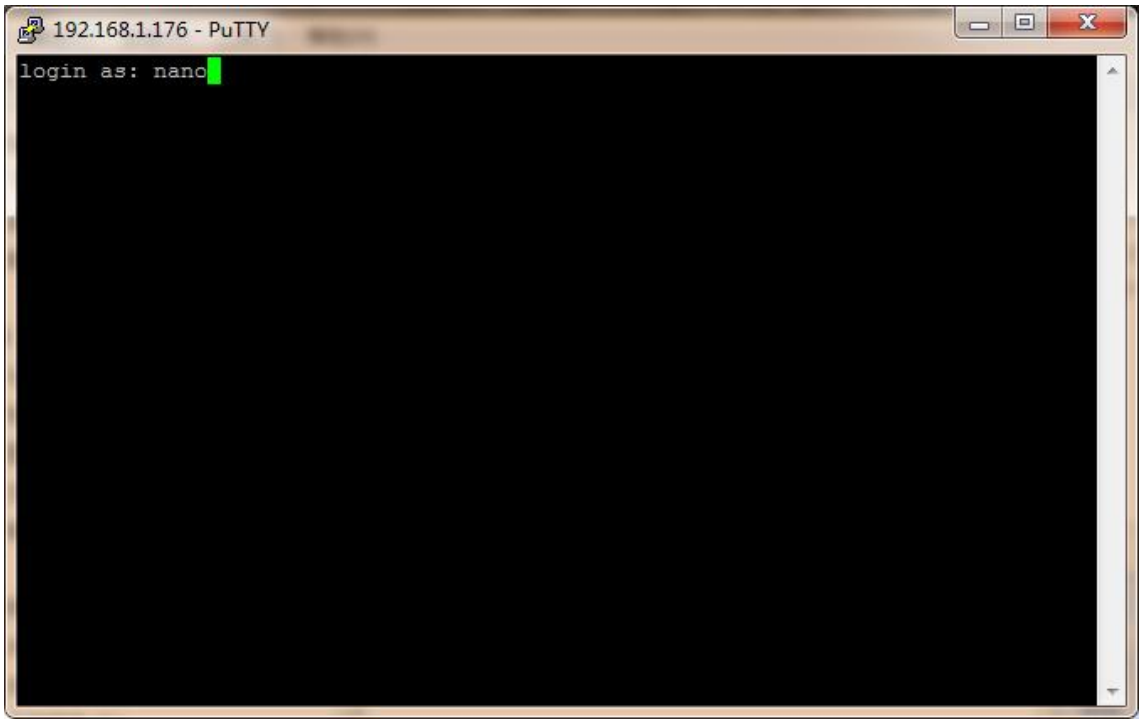
方法 2：可以登录无线路由器管理系统，找到板子的 ip 地址。

2、打开 puttY 在下面输入 ip 地址和端口号，默认系统已经开启了 ssh 服务。



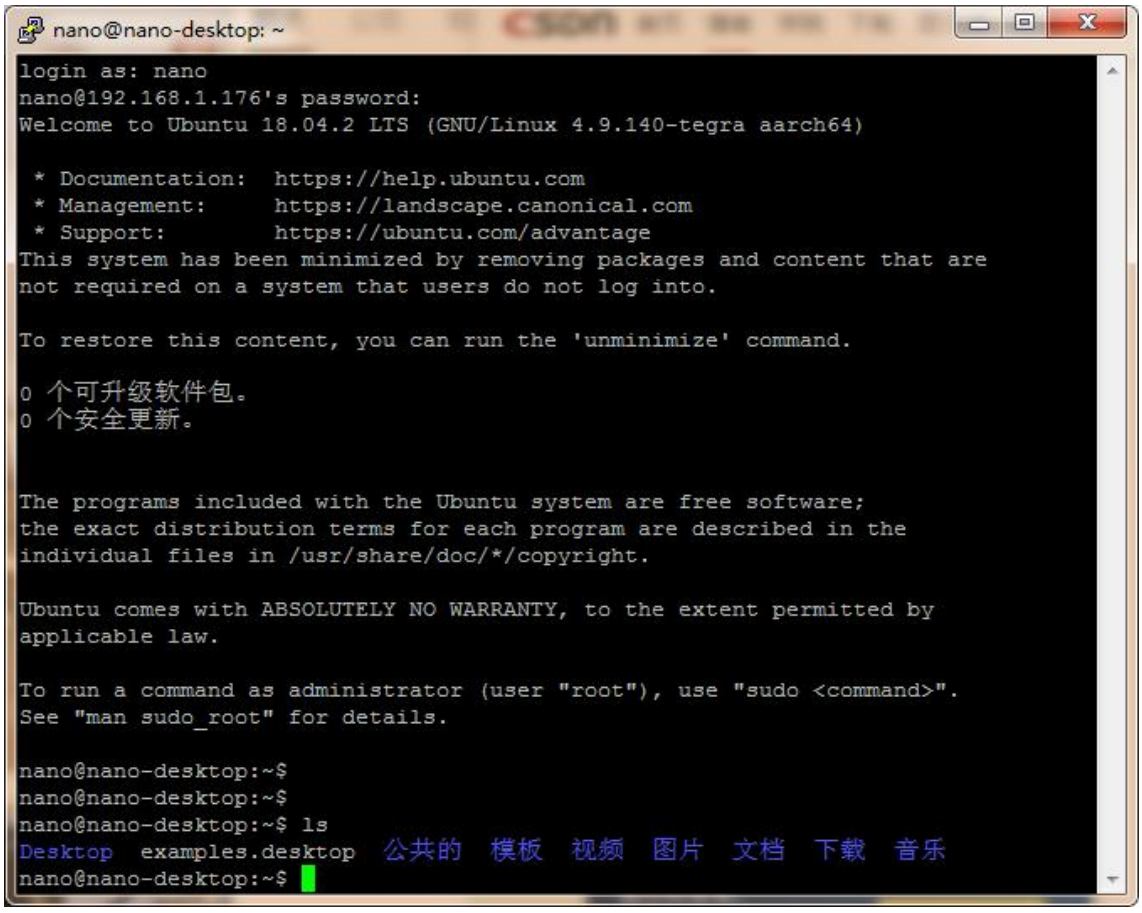
最后打开 Open，如下提示点击是。

3、输入我们在安装系统输入的登录名，假如这里是 Orin Nano。



然后输入密码后进入终端模式。

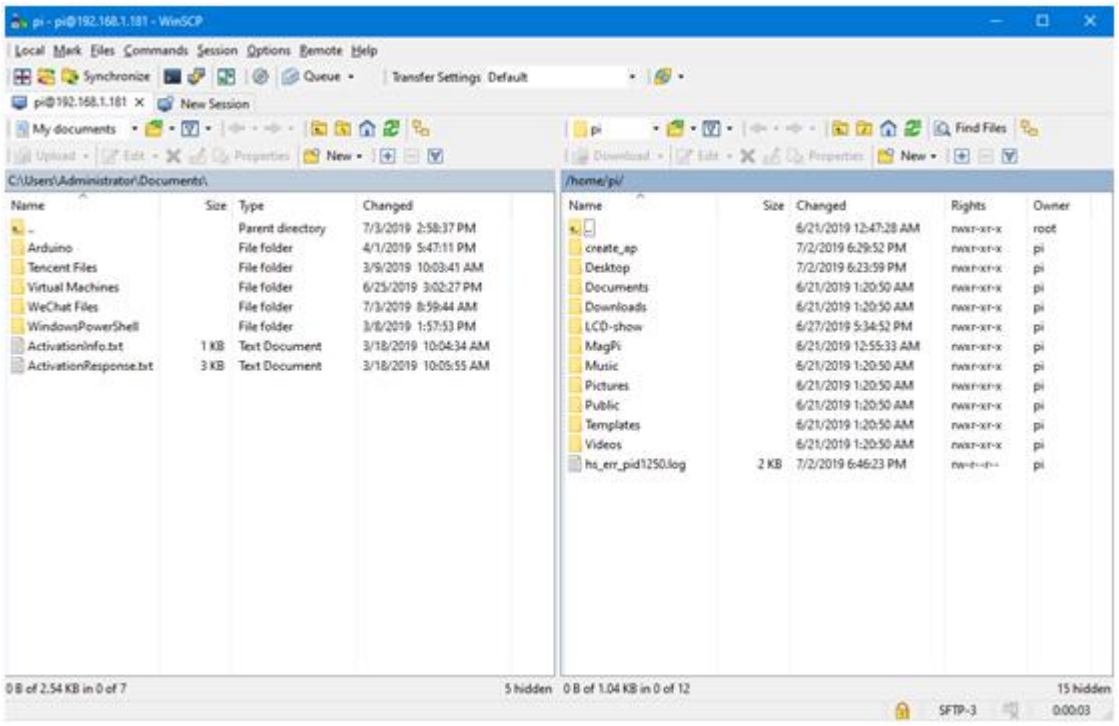
注意：输入密码这里，你输入的密码是隐藏的，输完之后直接回车即可



2.远程传输文件教程

有时候我们需要在 windows 和 linux 两个不同的系统之间传输文件，由于这是两个不同的文件系统，就需要用到了所谓的 ssh 服务来跨系统的传输文件。这里我用 winSCP 软件来传输，简单又好用。

新建站点输入 IP 地址，用户名，密码点击登录，如果以上信息都不怎么修改，可以直接保存，下次就可以直接进入到对应的用户系统。



出现以下界面即为登录成功

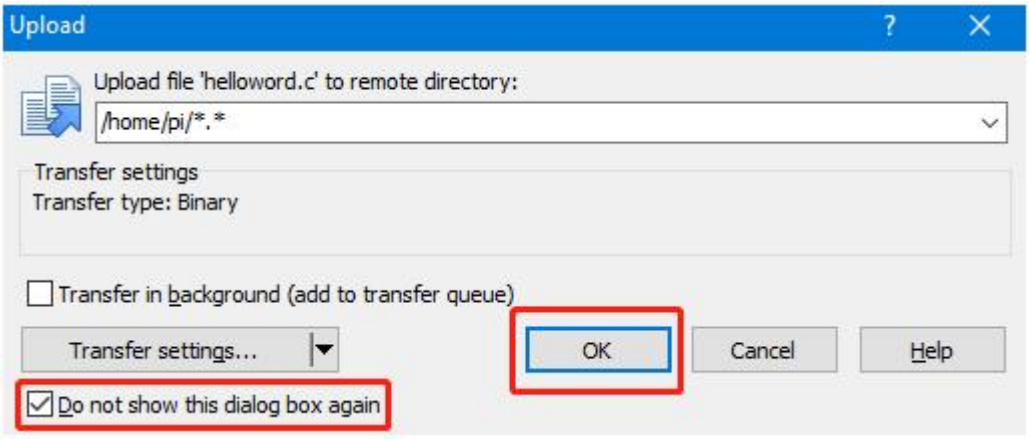
左边为 window 端，右边为 linux 系统端，可以直接拖拽文件到另一端进行文件传输，也可以在文件上右击鼠标选择对应的操作，如果移动，删除等。

点击 Login 登录成功后会显示以下界面，左边的是 win 电脑的文件夹，右边的是 Orin Nano 的文件夹。

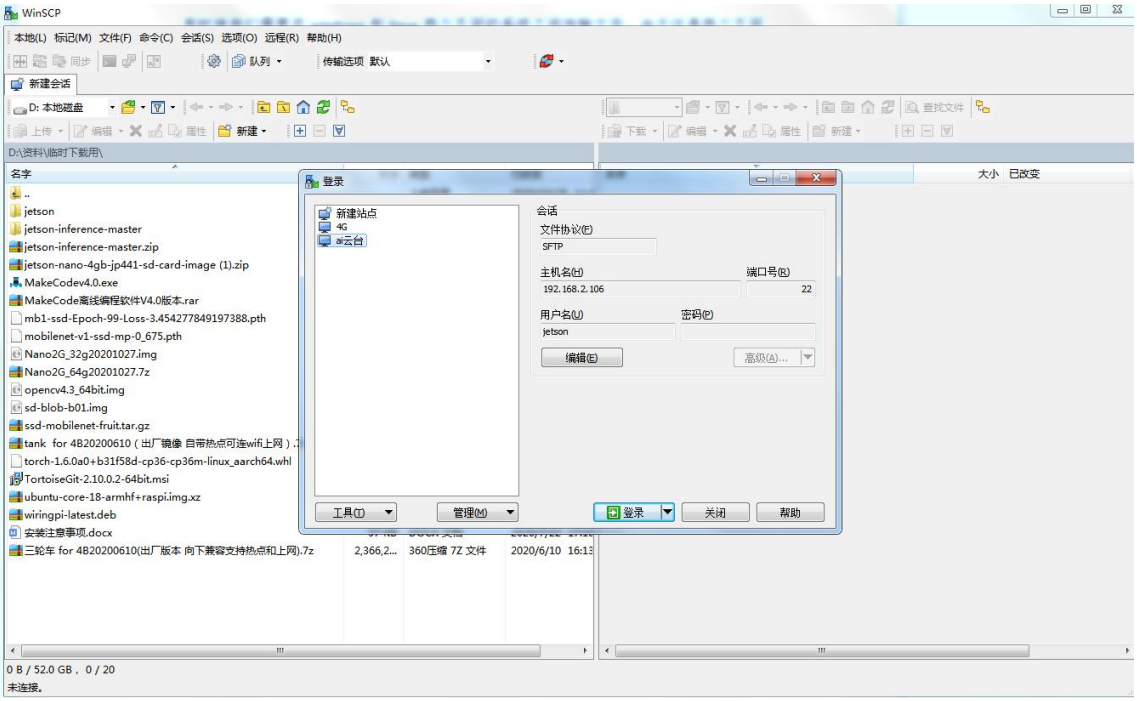
文件传输有三种操作方式，第一种是直接把文件从左边拉到右边，或者从右边拉到左边，系统会自动复制一份文件传输过去。

第二种是鼠标选中文件，然后按一下 F5 键，则被选中的文件会复制一份到另一边。

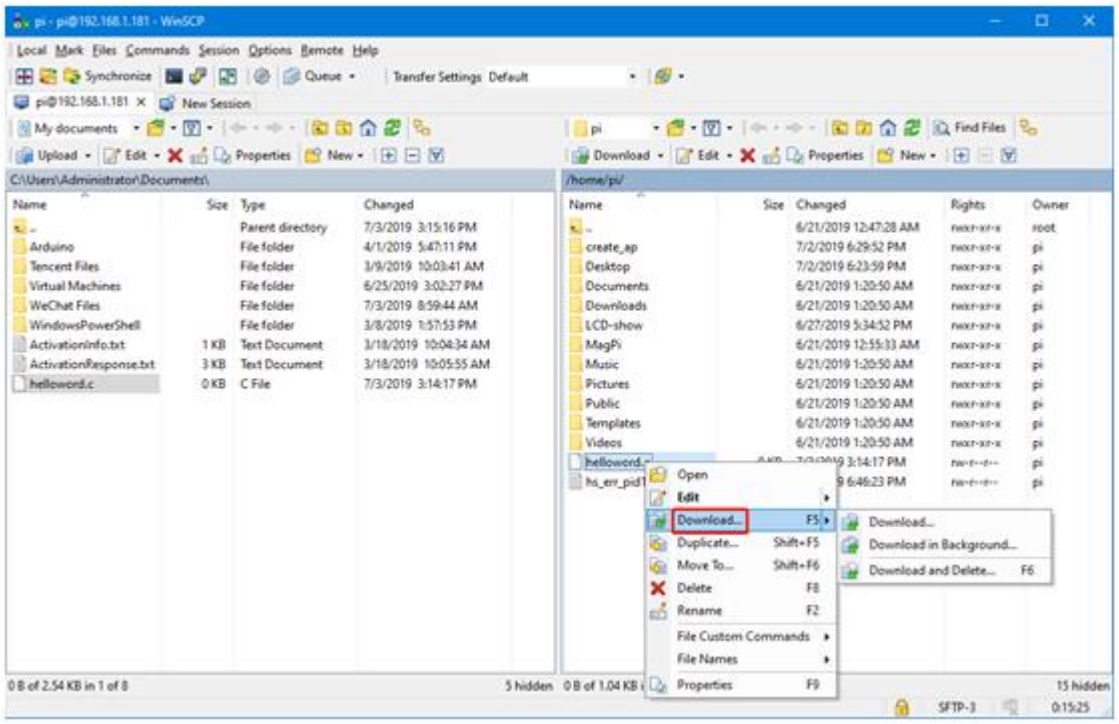
第三种是选中文件点击鼠标右键，如果是从 win 电脑传到 Orin Nano 则点击 upload，



会弹出一个提示，可以选择不再提示，并且点击 OK，则文件自动传输过去。



如果从 Orin Nano 传文件到 win 电脑上，则按鼠标右键选中文件，选择 Download



5、Jetson Orin Nano 通过 vnc 远程桌面控制

提示:配置好的镜像,用户名为 jetson 原始密码为:yahboom,如果使用的是配置好的镜像，VNC 已经配置好了，可直接跳到第 6 步，依据当前 ip 地址登录 VNC

1、安装 vino

```
sudo apt update
```

```
jetson@jetson-desktop:~$ 安装vino
-bash: 安装vino: command not found
jetson@jetson-desktop:~$ sudo apt update
Get:1 file:/var/cuda-repo-10-2-local-10.2.89  InRelease
Ign:1 file:/var/cuda-repo-10-2-local-10.2.89  InRelease
Get:2 file:/var/visionworks-repo  InRelease
Ign:2 file:/var/visionworks-repo  InRelease
Get:3 file:/var/visionworks-sfm-repo  InRelease
Ign:3 file:/var/visionworks-sfm-repo  InRelease
Get:4 file:/var/visionworks-tracking-repo  InRelease
Ign:4 file:/var/visionworks-tracking-repo  InRelease
Get:5 file:/var/cuda-repo-10-2-local-10.2.89  Release [574 B]
Get:5 file:/var/cuda-repo-10-2-local-10.2.89  Release [574 B]
Get:6 file:/var/visionworks-repo  Release [2,001 B]
Get:6 file:/var/visionworks-repo  Release [2,001 B]
Get:7 file:/var/visionworks-sfm-repo  Release [2,005 B]
Get:7 file:/var/visionworks-sfm-repo  Release [2,005 B]
Get:8 file:/var/visionworks-tracking-repo  Release [2,010 B]
Get:8 file:/var/visionworks-tracking-repo  Release [2,010 B]
Hit:15 http://ports.ubuntu.com/ubuntu-ports bionic InRelease
Hit:13 https://repo.download.nvidia.cn/jetson/common r32.4 InRe
Hit:14 https://repo.download.nvidia.cn/jetson/t210 r32.4 InRele
Get:16 http://ports.ubuntu.com/ubuntu-ports bionic-updates InRe
```

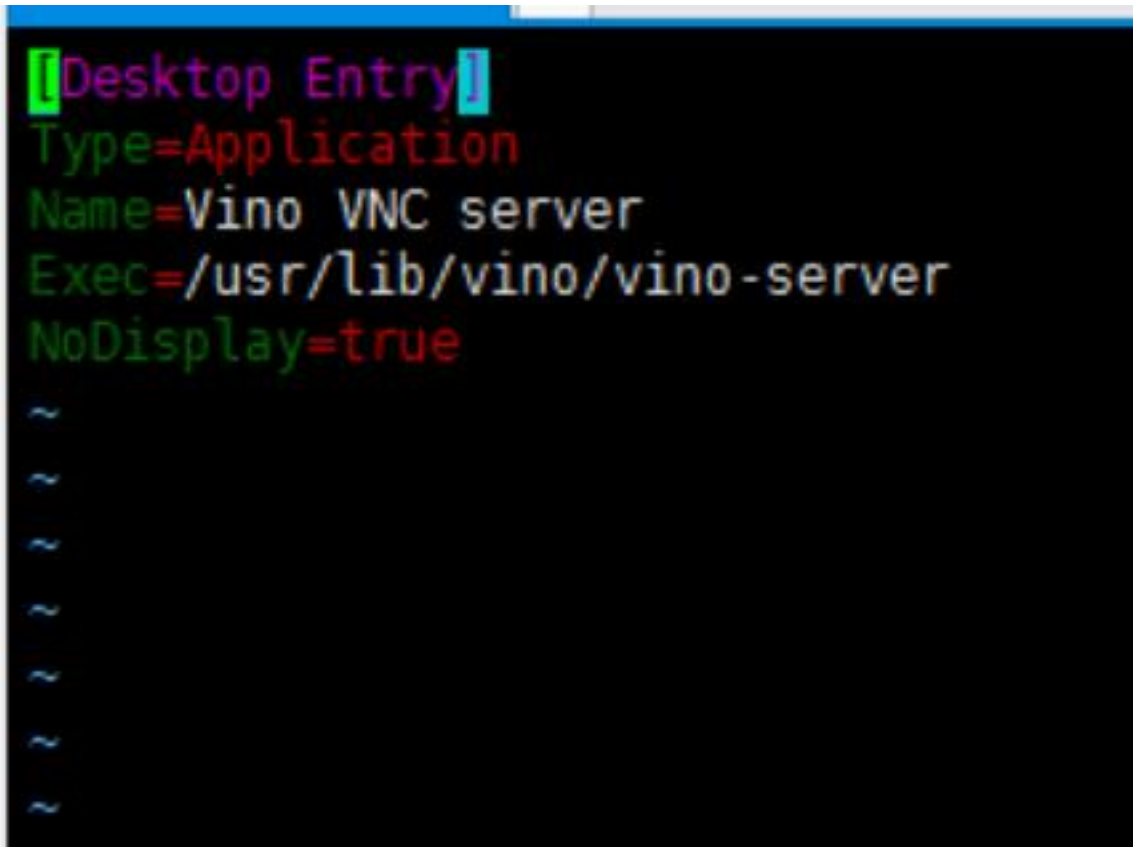
sudo apt install vino

```
jetson@jetson-desktop:~$ sudo apt install vino
Reading package lists... Done
Building dependency tree
Reading state information... Done
vino is already the newest version (3.22.0-3ubuntu1.1).
vino set to manually installed.
The following packages were automatically installed and are no long
  apt-clone archdetect-deb bogl-bterm busybox-static cryptsetup-bi
  kwayland-data kwin-common kwin-data kwin-x11 libdebian-installer4
  libkf5completion5 libkf5declarative-data libkf5declarative5 libkf
  libkf5jobwidgets-data libkf5jobwidgets5 libkf5kcmutils-data libkf
  libkf5package-data libkf5package5 libkf5plasma5 libkf5quickaddons
```

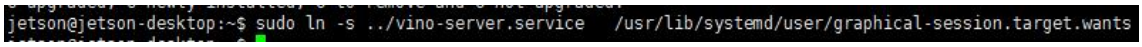
2、设 Enable VNC 服务 （此时手动可打开 vnc server）

sudo ln

-s ../vino-server.service



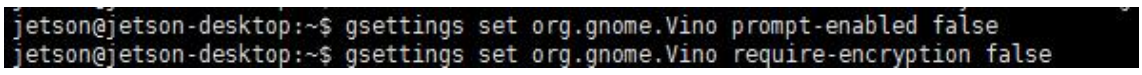
/usr/lib/systemd/user/graphical-session.target.wants



配置 VNC server:

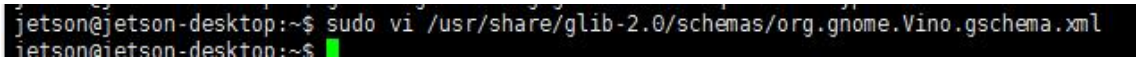
gsettings set org.gnome.Vino prompt-enabled false

gsettings set org.gnome.Vino require-encryption false



编辑 org.gnome,恢复丢失的"enabled"参数,输入一下命令进入文件，将下方 key 内容添加到文件的最后面。保存并退出。

sudo vi /usr/share/glib-2.0/schemas/org.gnome.Vino.gschema.xml



<key name='enabled' type='b'>

<summary>Enable remote access to the desktop<summary>

<description>

If true, allows remote access to the desktop via the RFB protocol.

Users on remote machines may then connect to the desktop using a

VNC viewer.

<description>

<default>false<default>

<key>

```
    <key name='disable-xdamage' type='b'>
      <summary>Whether we should disable the XDamage extension of X.org</summary>
      <description>
        If true, do not use the XDamage extension of X.org. This extension does
        not work properly on some video drivers when using 3D effects.
        Disabling it will make Vino work in these environments, with slower
        rendering as a side effect.
      </description>
      <default>false</default>
    </key>

    <key name='notify-on-connect' type='b'>
      <summary>Notify on connect</summary>
      <description>
        If true, show a notification when a user connects to the system.
      </description>
      <default>true</default>
    </key>
    <key name='enabled' type='b'>
      <summary>Enable remote access to the desktop</summary>
      <description>
         If true, allows remote access to the desktop via the RFB
         protocol. Users on remote machines may then connect to the
         desktop using a VNC viewer.
      </description>
      <default>false</default>
    </key>
  </schema>
</schemalist>
```

设置为 Gnome 编译模式

sudo glib-compile-schemas /usr/share/glib-2.0/schemas

现在屏幕共享面板在单位控制中心工作...但这并不足以让 vino 运行!所以您需要在会话启动时添加程序:Vino-server，使用以下命令行:

```
/usr/lib/vino/vino-server
```

```
jetson@jetson-desktop:~$ /usr/lib/vino/vino-server
```

这种是属于手动启动，如果每次都需要手动启动会比较麻烦下面会设置开机自启动的形式。

3、设置 VNC 登陆密码('thepassword' 修改为自己的密码)

```
gsettings set org.gnome.Vino authentication-methods "['vnc']"
gsettings set org.gnome.Vino vnc-password $(echo -n 'thepassword' |base64)
```

```
jetson@jetson-desktop:~$ gsettings set org.gnome.Vino vnc-password $(echo -n 'yahboom'|base64)
```

4、重启机器，验证是否设置 vnc 成功

```
sudo reboot
```

5.设置开机自启动 VNC Server

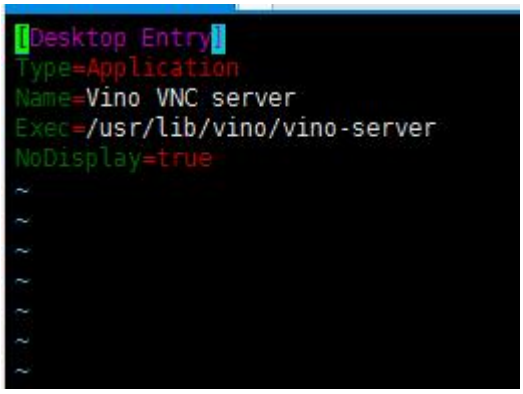
VNC 服务器只有在您本地登录到 Jetson 之后才可用。如果您希望 VNC 自动可用，请使用系统设置应用程序来启用自动登录。

```
gsettings set org.gnome.Vino enabled true
mkdir -p ~/.config/autostart
vi ~/.config/autostart/vino-server.desktop
```

将下面的内容添加到该文件中，保存并退出。

```
[Desktop Entry]
Type=Application
Name=Vino VNC server
Exec=/usr/lib/vino/vino-server
```

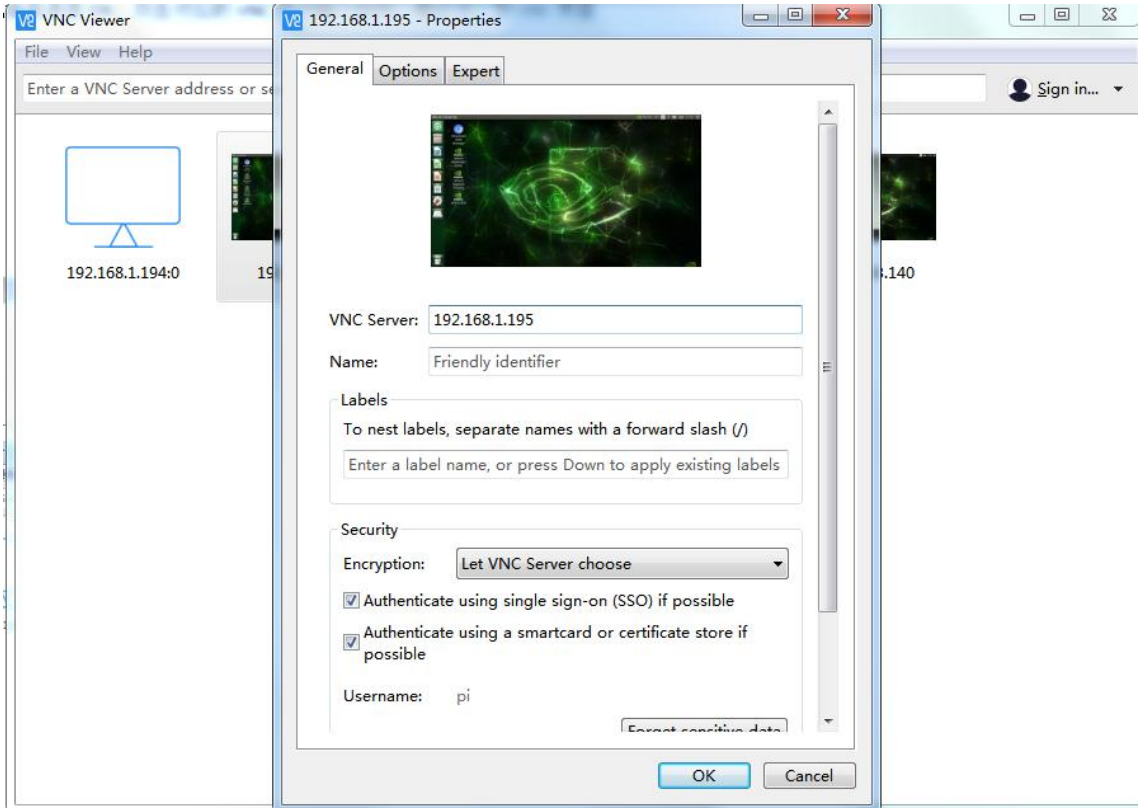
NoDisplay=true

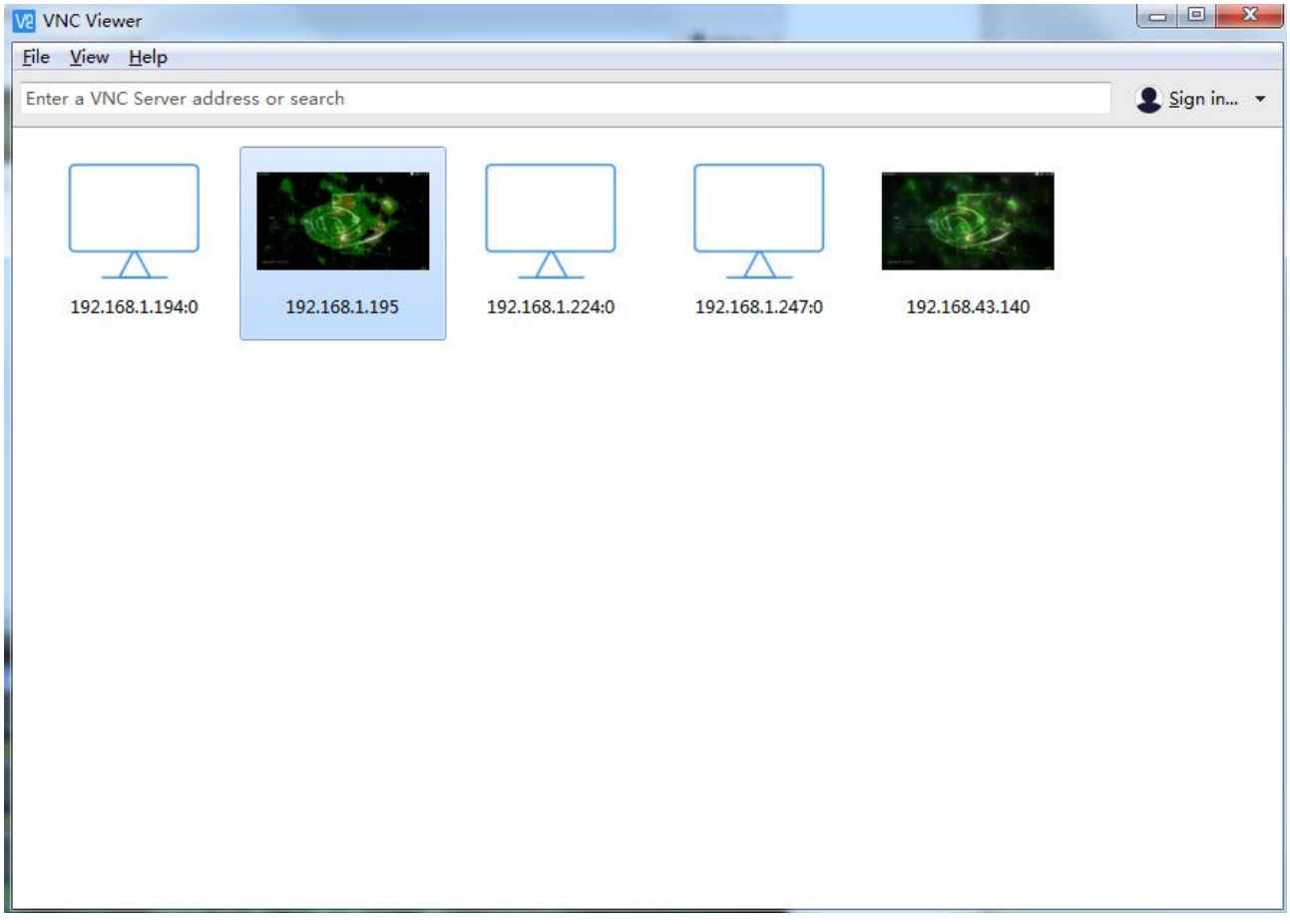


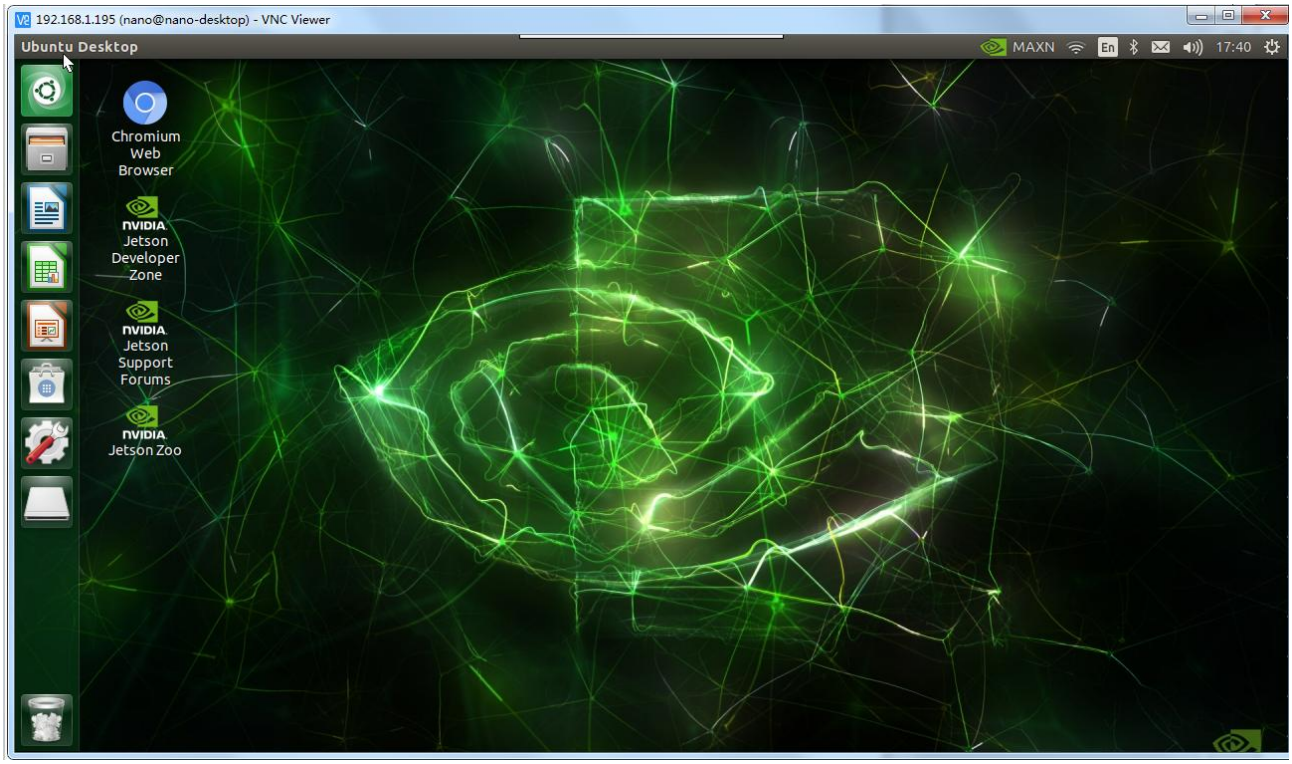
如果系统设置为需要输入用户密码才可以进入桌面，以上的改脚本需要等进入桌面后才会启动，建议将系统设置为用户自动登录到桌面。

6.连接 VNC Server

使用 vnc viewer 软件进行 VNC 连接，首先需要查询 ip 地址，我这里查到的是 192.168.1.195，输入 IP 地址后点击 OK，双击对应的 VNC 用户输入密码，最后进入到 VNC 界面

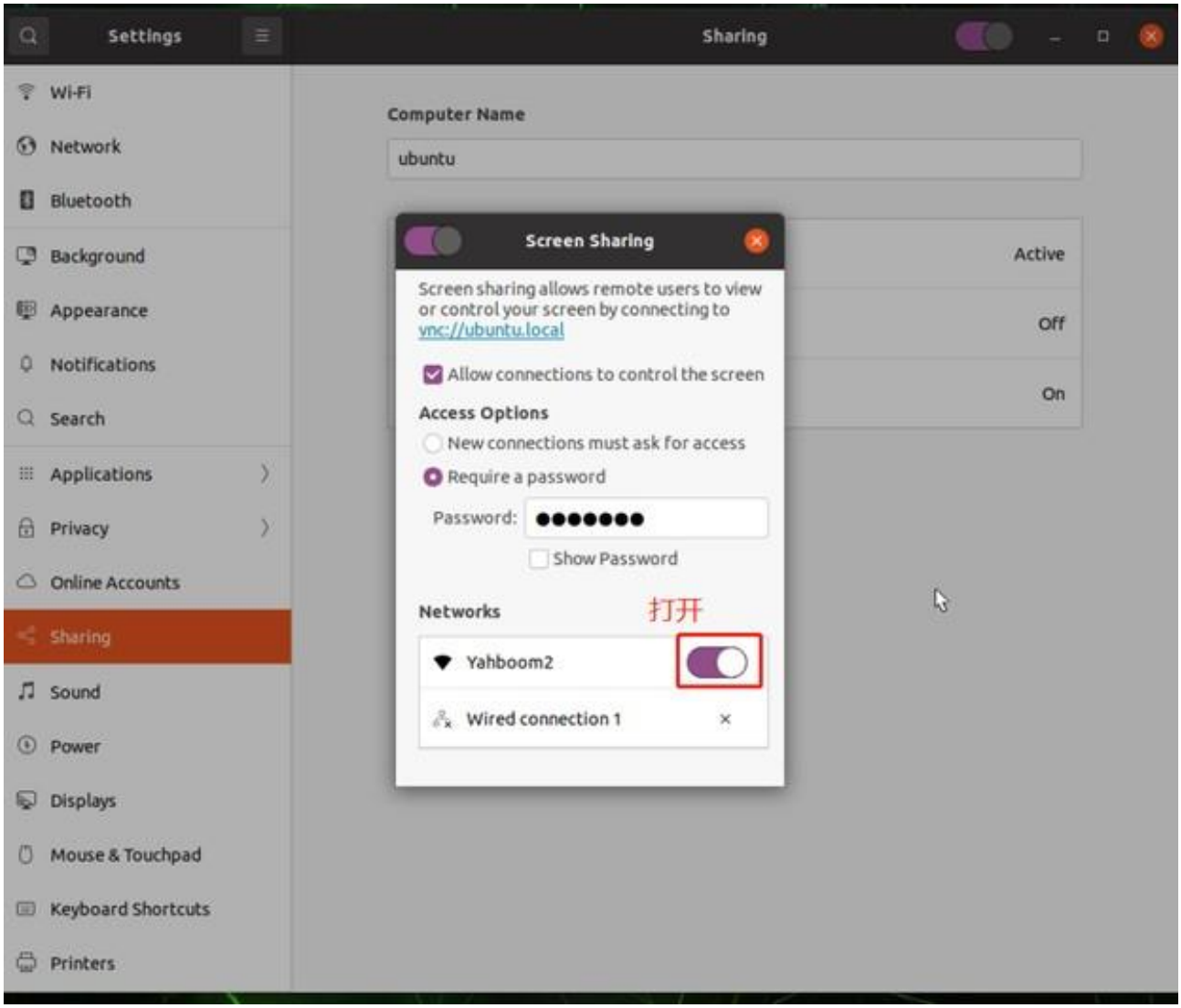

VNC Viewer
File View Help
Enter a VNC Server address or se
Sign in...
192.168.1.194:0
192.168.1.195 - Properties
General
Options
Expert
VNC Server: 192.168.1.195
Name: Friendly identifier
Labels
To nest labels, separate names with a forward slash (/)
Enter a label name, or press Down to apply existing labels
Security
Encryption: Let VNC Server choose
Authenticate using single sign-on (SSO) if possible
Authenticate using a smartcard or certificate store if possible
Username: pi
OK
Cancel


VNC Viewer
File View Help
Enter a VNC Server address or search
Sign in...
192.168.1.194:0
192.168.1.195
192.168.1.224:0
192.168.1.247:0
192.168.43.140



7.jetson orin Nano 需要是用 VNC 注意的事项

每连接到一个新的 wifi ,都需要在设置的页面把共享的 wifi 打开，如图所示



使用 VNC 桌面，需要接好 dp 的显示屏线，不然无法进入到 jetson orin 的桌面 Nano 的桌面

6、Jtop 的安装和使用

Jtop 的安装

(1) 安装 JTOP 查看可查 CPU 等的占用情况

```
sudo apt-get update
sudo apt-get full-upgrade
sudo apt install curl
sudo apt install nano
```

```
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py #下载安装脚本
sudo python3 get-pip.py #  运行安装脚本
sudo pip3 install jetson-stats
#sudo systemctl restart jetson_stats.service
reboot
jtop
```

检查已经安装的系统组件

(1)Jetson 的 OS 镜像已经自带了 JetPack，cuda，cudnn，opencv 等都已经安装好，并有例子，这些例子安装路径如下所示

```
TensorRT /usr/src/tensorrt/samples/
CUDA /usr/local/cuda-11.4/samples/
cuDNN /usr/src/cudnn_samples_v8/
VisionWorks /usr/share/visionworks/sources/samples/
/usr/share/visionworks-tracking/sources/samples/
/usr/share/visionworks-sfm/sources/samples/

OpenCV /usr/share/opencv4/samples/
```

(2) 检查 CUDA Jetson orin nano 中已经安装了 CUDA11.4 版本，但是此时你如果运行 nvcc -V 是不会成功的，需要你把 CUDA 的路径写入环境变量中。OS 中自带 Vim 工具，所以运行下面的命令编辑环境变量

首先，查看 cuda 的 bin 目录下是否有 nvcc：

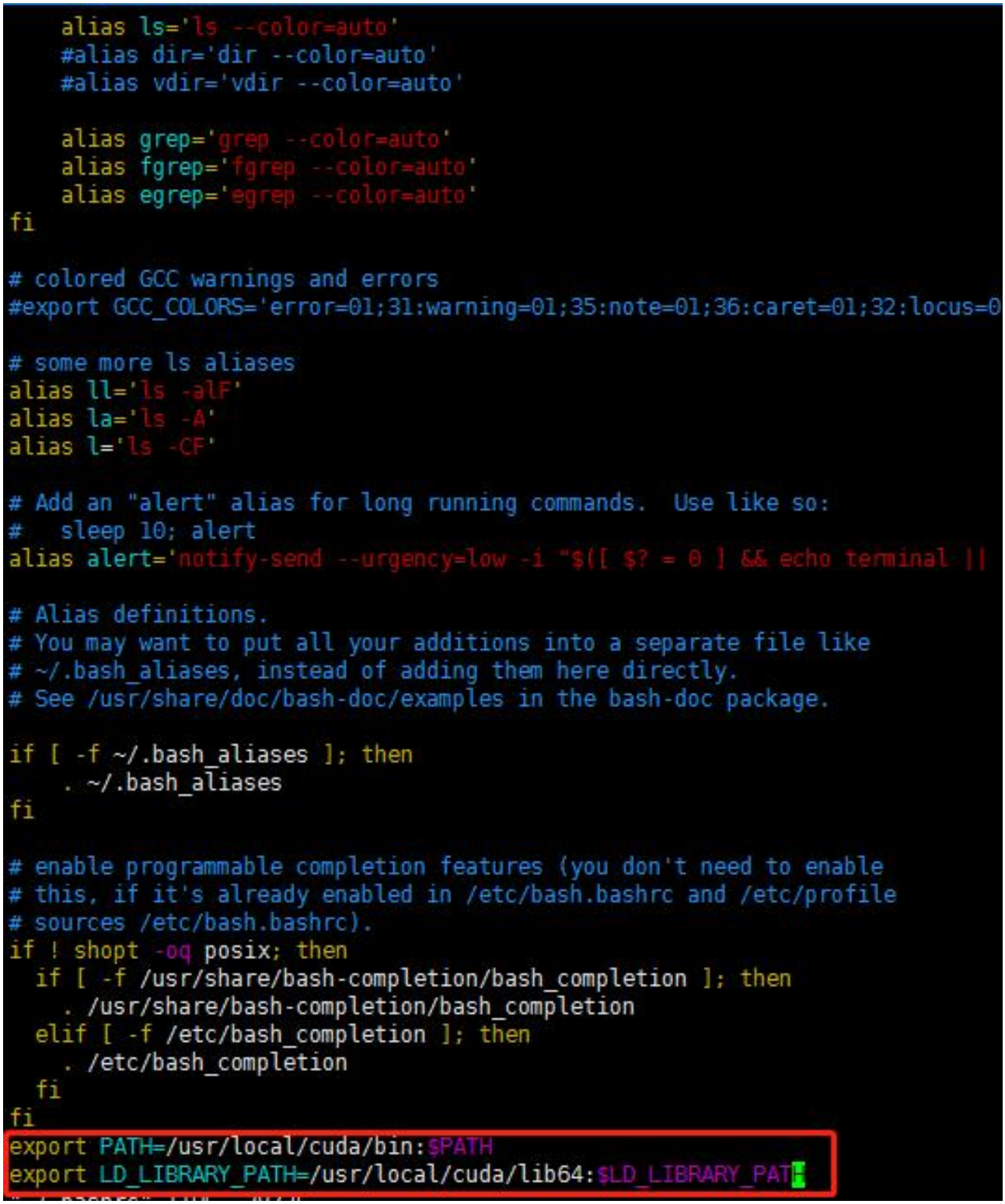
```
ls /usr/local/cuda/bin
```

如果存在，

```
sudo vim ~/.bashrc 进入配置文件；  在最后面添加以下两行：
```

注意:在 vim 中通过 Esc 退回命令模式，通过 I 切换到输入模块，在输入模式下才可以输入文本

```
export PATH=/usr/local/cuda/bin:$PATH

export LD_LIBRARY_PATH=/usr/local/cuda/lib64:$LD_LIBRARY_PATH
```



注意:通过 Esc 退出到命令模式后，通过按下:开始输入命令，wq 为保存并退出，q 为退出，q!为强制退出

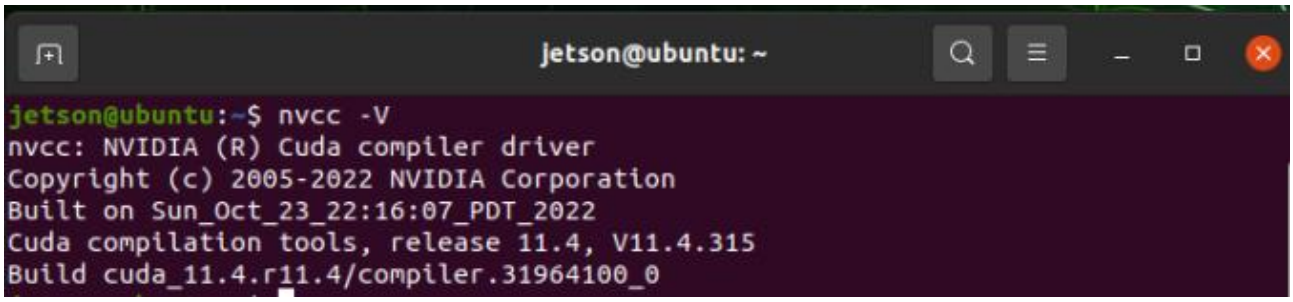
保存退出。

然后需要 source 下生效下。

```
source ~/.bashrc
```

source 后，此时再执行 nvcc -V 执行结果如下

beckhans@Jetson:~$ nvcc -V



（3）检查 OpenCV

Jetson-nano 中已经安装了 OpenCV4.5.4 版本，可以使用命令检查 OpenCV 是否安装就绪

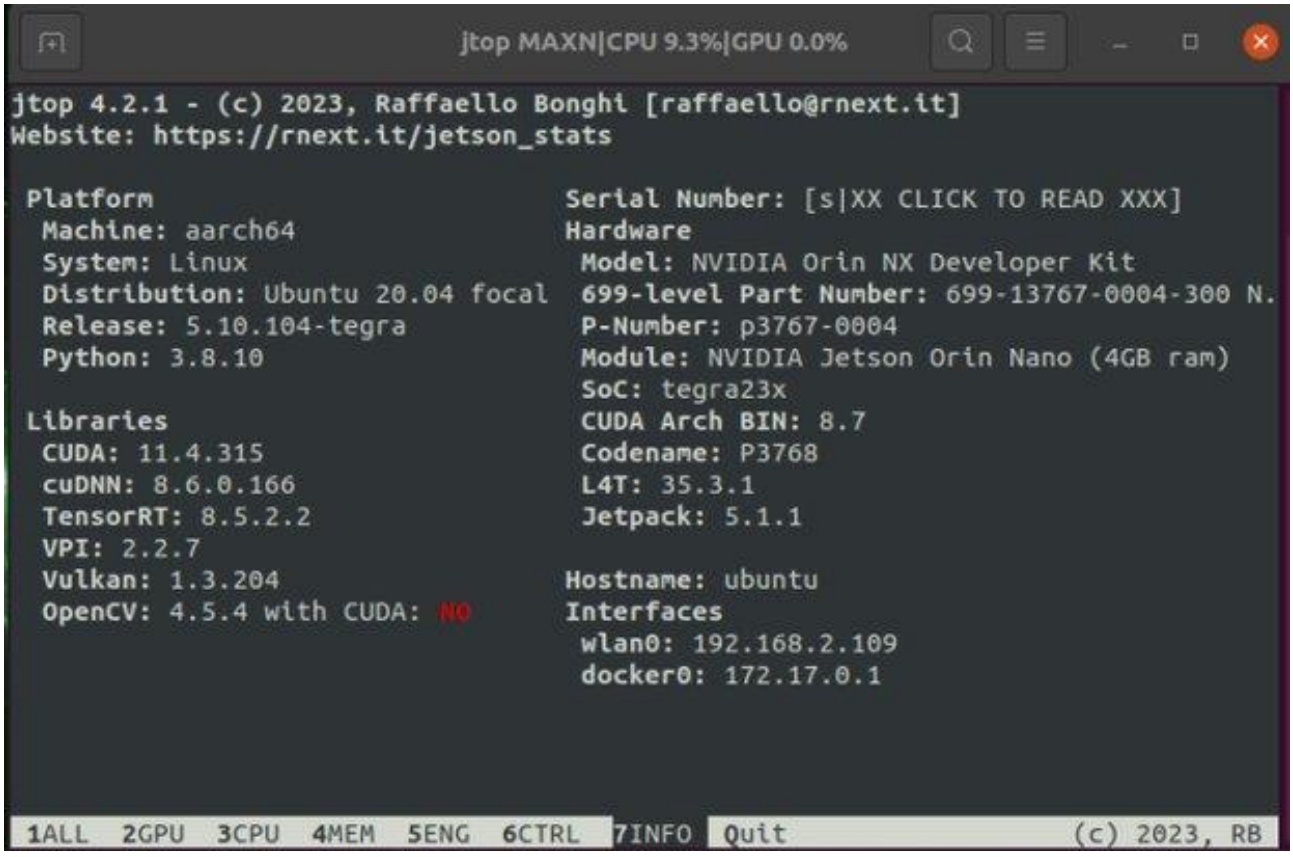
pkg-config opencv4 --modversion

如果 OpenCv 安装就绪，会显示版本号，我的版本是 4.5.4

（4）检查 cuDNN

Jetson-nano 中已经安装好了 cuDNN，并有例子可供运行，我们运行一下例子，也正好验证上面的 CUDA

终端输入 jtop,按键盘的右方向键选到 **7info**,可以看到 cuDNN 的版本，如下图所示：



第四章 GPIO 硬件控制

1、Jetson Orin Nano 与外部设备串口通讯教程

1.准备工作

本节测试 Jetson Orin Nano 串口自己收发的教程。由下图可以看到 Jetson Orin Nano 的串口的 TXD 和 RXD 引脚分别对应物理引脚 8,10。

BCM编码	功能名	物理引脚		功能名	BCM编码
	3V3	1	2	5V	
2	SDA	3	4	5V	
3	SCL	5	6	GND	
4	D4	7	8	D14(TXD)	14
	GND	9	10	D15(RXD)	15
17	D17	11	12	D18	18
27	D27	13	14	GND	
22	D22	15	16	D23	23
	3V3	17	18	D24	24
10	D10	19	20	GND	
9	D9	21	22	D25	25
11	D11	23	24	D8	8
	GND	25	26	D7	7
0	DO(ID_SD)	27	28	D1(ID_SC)	1
5	D5	29	30	GND	
6	D6	31	32	D12	12
13	D13	33	34	GND	
19	D19	35	36	D16	16
26	D26	37	38	D20	20
	GND	39	40	D21	21

接线: Jetson Orin Nano 引脚 8 (TXD) →Jetson Orin Nano 引脚 10 (RXD)

开启串口权限，注意这个权限关机后就也被关闭，下次需要重新开启

```
sudo chmod 777 /dev/ttyTHS1
```

2.使用例程测试串口的功能

从 github 下载一个串口通讯案例，源代码和解释可以到这个 github 连接查看

```
git clone <https://github.com/JetsonHacksNano/UARTDemo>
```

进入文件夹

```
cd UARTDemo
```

如果您不打算在 UART 上使用串行控制台，则应禁用串行控制台(可不做看个人选择)：

```
systemctl stop nvgetty
systemctl disable nvgetty
udevadm trigger
```

安装 serial 模块

```
 sudo apt-get install python3-serial
```

终端输入运行程序:

```
sudo python3 uart_example.py
```

```
nano@nano-desktop:~/UARTDemo$ python3 uart_example.py
UART Demonstration Program
NVIDIA Jetson Nano Developer Kit
```

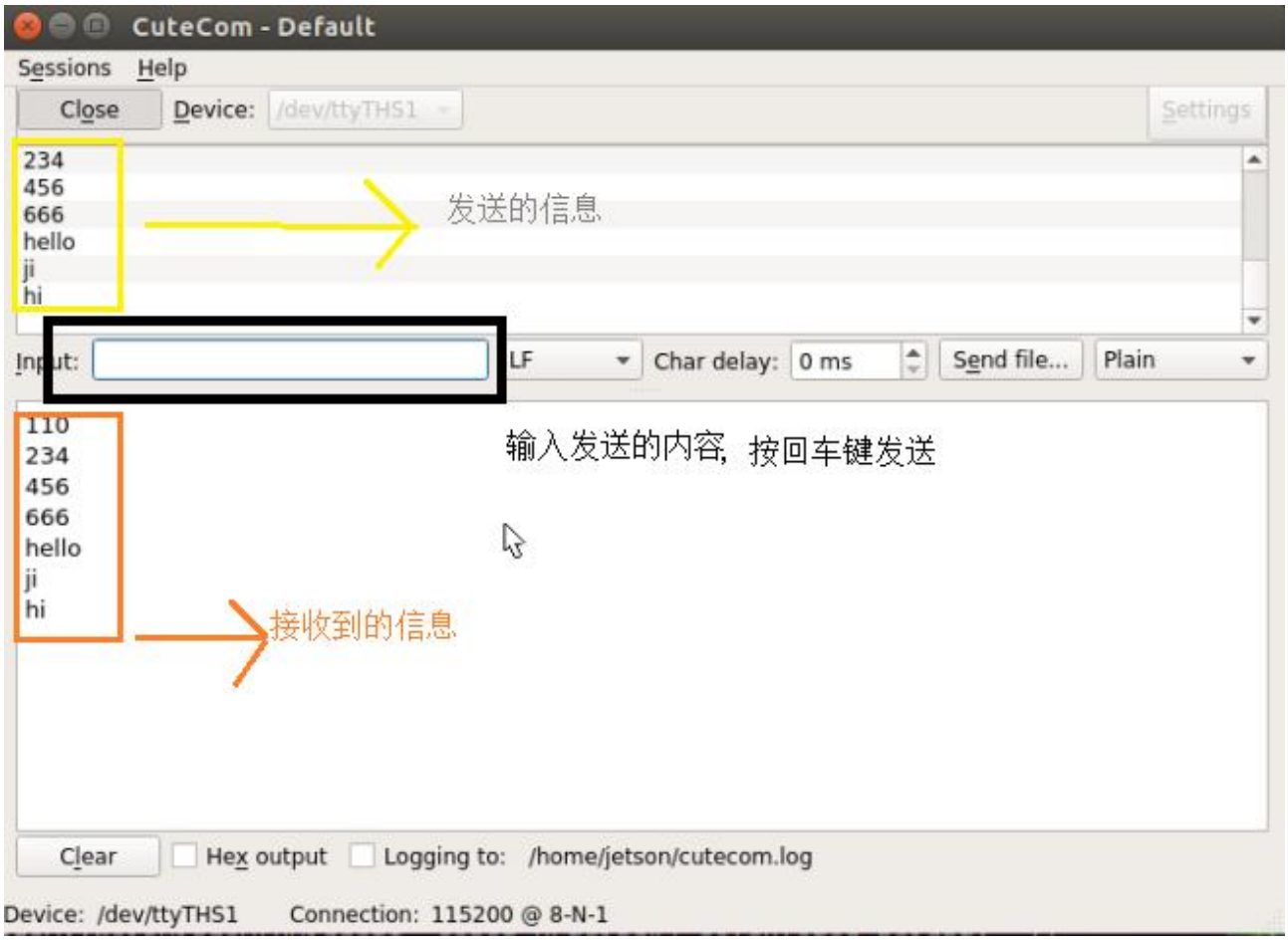
运行后可以看到 Jetson Orin Nano 循环的发送"NVIDIA Jetson Orin Nano Developer Kit\r\n"里面的字符，并显示终端上。

3.使用 linux 的串口助手测试

1.运行以下命令

```
sudo apt install cutecom
sudo cutecom
```

就可以看见 cutecom 打开了 一般不需要设置，直接点击 **open** 就能使用，然后通过 InPUT 输入文本，按回车键就能发送内容了。 效果如图：



4.使用 Yahboom 例程测试

1.运行以下命令

```
cd ~/GPIO_test
python3 test_serial_810.py #如果没有这个程序，从资料的附件中传输到orin Nano 上
```

现象既可以看到，发送什么信息就回复什么信息

5.注意事项

如果使用 USB 转 TTL 模块电脑和 Orin Nano 进行通信注意一下几点 1.杜邦线不可太长，太长会乱码 2.出现只能收不能发的情况是电压不足导致的，把 usb 转 ttl 模块的 5V 口和 nano 的 5V 进行连接 3.如果线合理，但出现乱码，波特率、奇偶校验、停止位检查是否一致 4.其它情况：

https://blog.csdn.net/lxj362343/article/details/89646731

7、Jetson Orin Nano I2C 通讯教程

Jetson Orin Nano 的 I2C 引脚如图所示，要使用前需要开启 I2C 服务。

GPIO和BCM对照表

BCM编码	功能名	物理引脚		功能名	BCM编码
	3V3	1	2	5V	
2	SDA	3	4	5V	
3	SCL	5	6	GND	
4	D4	7	8	D14(TXD)	14
	GND	9	10	D15(RXD)	15
17	D17	11	12	D18	18
27	D27	13	14	GND	
22	D22	15	16	D23	23
	3V3	17	18	D24	24
10	D10	19	20	GND	
9	D9	21	22	D25	25
11	D11	23	24	D8	8
	GND	25	26	D7	7
0	D0(ID_SD)	27	28	D1(ID_SC)	1
5	D5	29	30	GND	
6	D6	31	32	D12	12
13	D13	33	34	GND	
19	D19	35	36	D16	16
26	D26	37	38	D20	20
	GND	39	40	D21	21

首先安装 I2Ctool，终端输入：

```
sudo apt-get update
```

```
sudo apt-get install -y i2c-tools
```

检查安装情况，终端输入：

```

```

```
apt-cache policy i2c-tools
```

输出如下即为安装成功

```

```

```
i2c-tools:
```

```
已安装：4.0-2
```

```
候选：  4.0-2
```

```
版本列表：
```

```
*** 4.0-2 500
```

```
500 http://ports.ubuntu.com/ubuntu-ports bionic/universe arm64
```

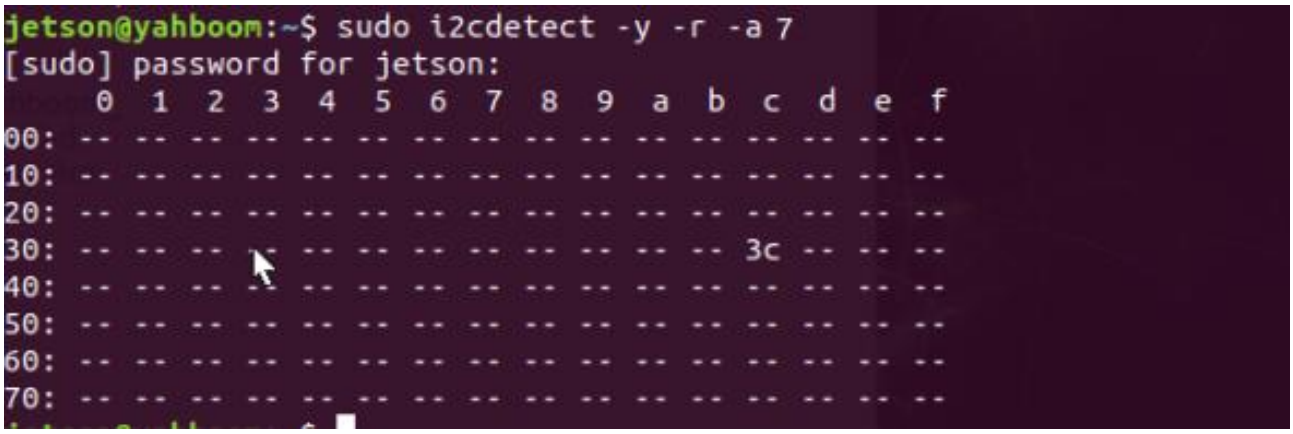
```
Packages
```

```
100 /var/lib/dpkg/status
```

扫描某一总线 bus 上所有 i2c 设备，并且打印出设备 i2c 总线地 址，例如这里 I2C 引脚上挂载了一个地址为 0x0f 的设备，则会 显示起对应的设备 I2C 地址

```

```

```
sudo i2cdetect -y -r -a 7
```



smbus 为 python 库，如果未安装 smbus，终端输入:

```
sudo apt-get update
sudo apt-get install -y python3-smbus
```

Smbus 协议有许多相关的库函数可以用于 I2C 通讯

function	description	parameters	return value
SMBus Access			
write_quick (addr)	Quick transaction.	int addr	long
read_byte (addr)	Read Byte transaction.	int addr	long
write_byte (addr, val)	Write Byte transaction.	int addr, char val	long
read_byte_data (addr, cmd)	Read Byte Data transaction.	int addr, char cmd	long
write_byte_data (addr, cmd, val)	Write Byte Data transaction.	int addr, char cmd, char val	long
read_word_data (addr, cmd)	Read Word Data transaction.	int addr, char cmd	long
write_word_data (addr, cmd, val)	Write Word Data transaction.	int addr, char cmd, int val	long
process_call (addr, cmd, val)	Process Call transaction.	int addr, char cmd, int val	long
read_block_data (addr, cmd)	Read Block Data transaction.	int addr, char cmd	long [ ]
write_block_data (addr, cmd, vals)	Write Block Data transaction.	int addr, char cmd, long [ ]	None
block_process_call (addr, cmd, vals)	Block Process Call transaction.	int addr, char cmd, long [ ]	long [ ]
I2C Access			
read_i2c_block_data (addr, cmd)	Block Read transaction.	int addr, char cmd	long [ ]
write_i2c_block_data (addr, cmd, vals)	Block Write transaction.	int addr, char cmd, long [ ]	None

下面提供了本店 oled 模块的程序案例，如果需要测试要使用相应的 oled 模块（可以在我们店铺选购改 oled 模块）

接线:

Jetson Orin Nano 引脚 3（SDA）→oled 模块 SDA

Jetson Orin Nano 引脚 5（SCL）→oled 模块 SCL

Jetson Orin Nano 引脚 2（5V）→oled 模块 VCC

Jetson Orin Nano 引脚 6（GND）→oled 模块 GND

导入 Adafruit_SSD1306 库这是 oled 库，使用自己的镜像情况下得下载这个库

```
pip install Adafruit_SSD1306
```

```python
#!/usr/bin/env python3
# coding=utf-8
import time
import os

import Adafruit_SSD1306 as SSD

from PIL import Image
from PIL import ImageDraw
from PIL import ImageFont

import subprocess

# V1.0.1
class Yahboom_OLED:
    def __init__(self, i2c_bus=1, debug=False):
        self.__debug = debug
        self.__i2c_bus = i2c_bus
        self.__top = -2
        self.__x = 0

        self.__total_last = 0
        self.__idle_last = 0
        self.__str_CPU = "CPU:0%"

    def __del__(self):
        if self.__debug:
            print("---OLED-DEL---")

    # 初始化OLED，成功返回:True，失败返回:False
    # Initialize OLED, return True on success, False on failure
```

初始化 oled：

```
# 初始化OLED，成功返回:True，失败返回:False
# Initialize OLED, return True on success, False on failure
def begin(self):
    try:
        self.__oled = SSD.SSD1306_128_32(
            rst=None, i2c_bus=self.__i2c_bus, gpio=1)
        self.__oled.begin()
        self.__oled.clear()
        self.__oled.display()
        self.__width = self.__oled.width
        self.__height = self.__oled.height
        self.__image = Image.new('1', (self.__width, self.__height))
        self.__draw = ImageDraw.Draw(self.__image)
        self.__font = ImageFont.load_default()
        if self.__debug:
            print("---OLED begin ok!---")
        return True
    except:
        if self.__debug:
            print("---OLED no found!---")
        return False
```

之后就是读取 nano 的一些基础信息函数感兴趣的可以去这个 py 文件里面去详细了解，分别是获取本机 ip、tf 卡空间占用率、内存占用率、系统时间等信息。

终端输入：

```
cd ~/GPIO_test
sudo python3 test_i2c_oled.py
```

如果是自己搭建的镜像，可以从资料的附件中找到 test_i2c_oled.py 后传输到 jetson orin Nano 上 实验现象：


NVIDIA
CPU:15 10:40:38
RAM:10% -> 15.0GB
SDC:30% -> 119.9GB
IPA:192.168.2.112

第五章 AI 视觉进阶教程

1、摄像头测试

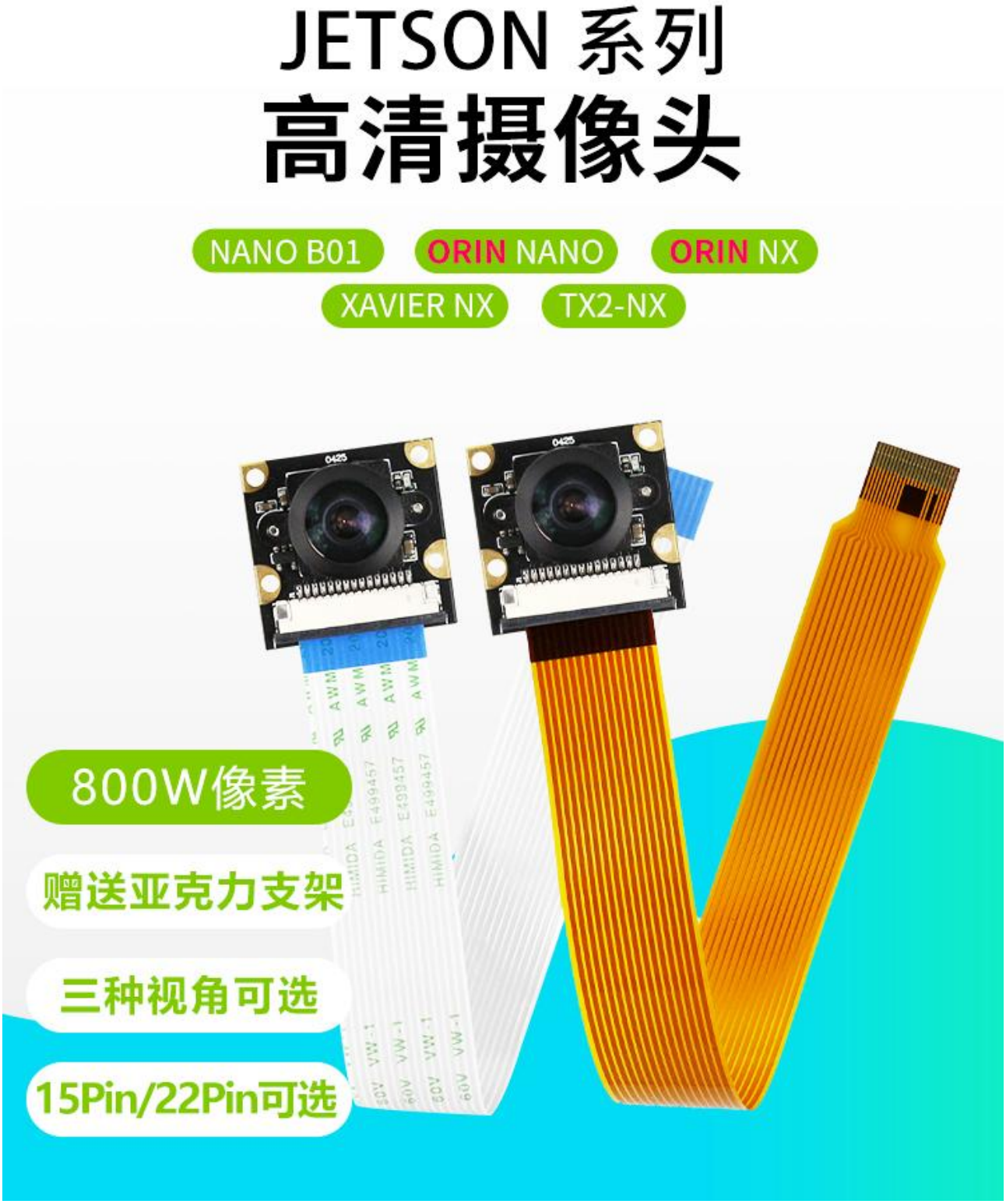
摄像头测试

1.板载摄像头教程

2.USB 摄像头测试 **USB 摄像头测试教程**

1.板载摄像头教程

注意：暂时只支持 IMX219 传感器的摄像头：



如何测试摄像头，打开 Jetson orin Nano 的终端

写入命令：nvgstcapture-1.0 ， 摄像头就会起来了

笔者简单的使用了手册中的几个命令

1. --prev_res 预览视屏的分辨率，高度和宽度，用的是 CSI 摄像头的话

范围是 2 to 12 (5632x4224)

e.g.， nvgstcapture-1.0 --prev-res=3

1. --cus-prev-res 自定义预览分辨率，宽度和高度，仅支持 CSI 摄像头

e.g., nvgstcapture-1.0 --cus-prev-res=1920x1080

多个命令同时使用的话用！隔开

想关掉摄像头的额话，直接在终端输入 q 再按回车

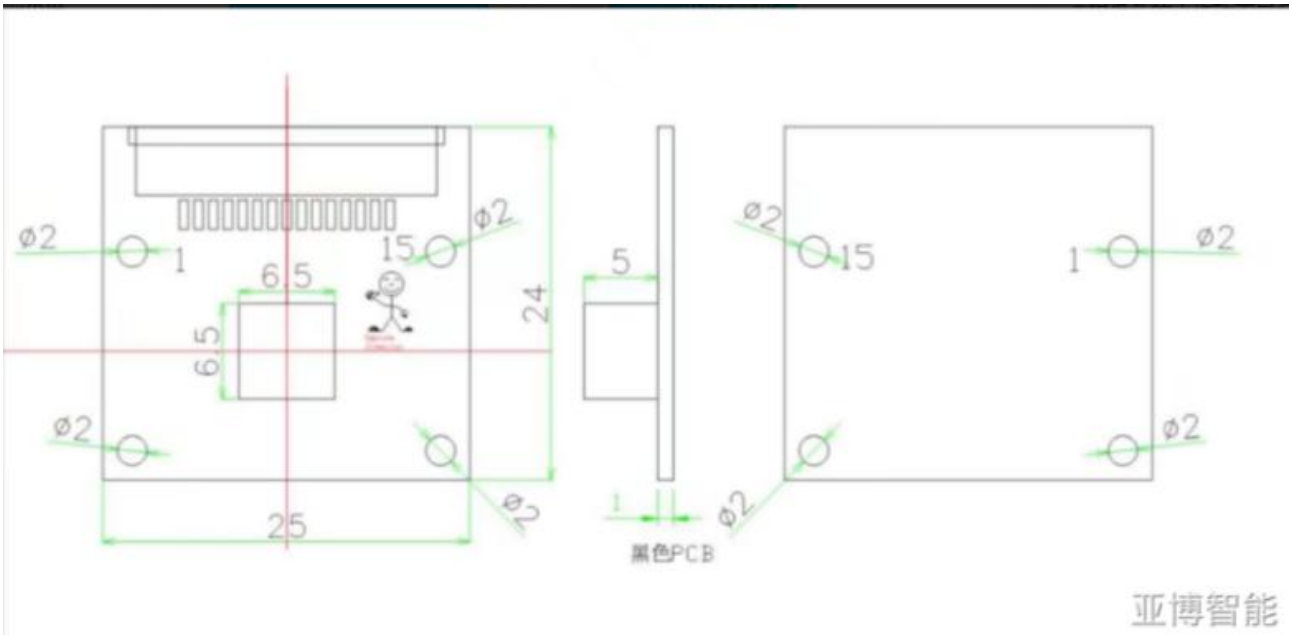
想捕获图片的话，在终端输入 j 再按回车，图片将保存当前目录下

1. 也可以（YAHBOOM 版的镜像）

```
cd /home/jetson/GPIO_TEST
python3 test_camera_csi.py #默认打开/dev/video0
```

常见问题：

摄像头的孔距和尺寸参数是怎样的



- csi160 摄像头畸变系数能给吗

答： 没有这个畸变系数

2.USB 摄像头测试

USB 摄像头测试教程

我使用的亚博智能科技的树莓派高清USB免驱摄像头，在Nano下的ubuntu系统中也不用安装驱动，使用方便配置简单。

输入：

```
ls /dev/video*
```

将检测到两个摄像头，/dev/video0 是 Orin Nano 开发板上自带的 CSI 摄像头，/dev/video1 就是刚连接上的 USB 摄像头了。如果没有插 CSI 摄像头则应该是 video0

测试摄像头：

1）使用应用程序 camorama

输入命令：

```
sudo apt-get install camorama
```

安装完成后，在终端中输入命令：（实际应用中为了方便，我将板载的 CSI 摄像头拿掉了，所以默认的仅有刚安装的 USB 摄像头）

```
camorama
```

即可显示出视频信息。这个需要让 USB 摄像头的变化为 video0，所以如果同时接了 csi 和 usb 摄像头，usb 摄像头一般是被分配到 video1 那么启动会报错。

2）使用应用程序茄子（cheese）

输入命令：

```
sudo apt-get install cheese
```

装好后，用命令：

```
cheese
```

即可打开。

1. 也可以（YAHBOOM 版的镜像）

```
cd /home/jetson/GPIO_test
python3 test_camera_usb.py #默认打开/dev/video0
```

1. Jupter lab 和 Jetcham 安装

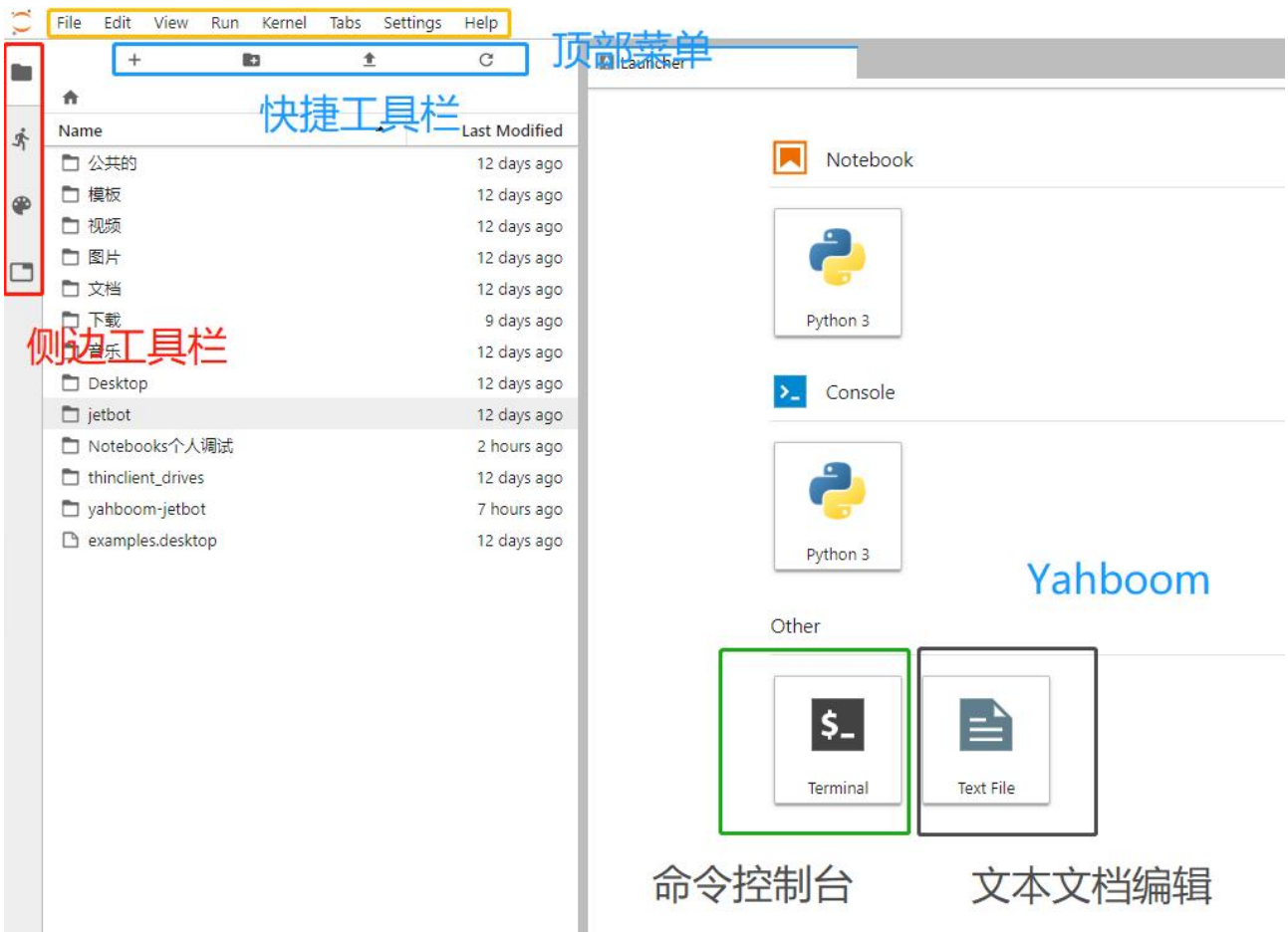
2、Jupyter lab 和 Jetcham 的安装

使用前说明

本教程适用于自行搭建的镜像，如果使用的使 YAHBOOM 版的镜像，这部分已经安装，可以忽略不看

JupyterLab 介绍

JupyterLab 界面是一个仪表板，可以访问交互式 iPython 笔记本，以及 Jetbot 的文件夹结构和进入 Ubuntu 操作系统的终端窗口。您将看到的第一个视图包括顶部的**菜单栏**，**左侧边栏中**的目录树以及最初向"启动器"页面打开的**主工作区**。



有关所有功能和菜单操作的完整详细信息，请参阅 JupyterLab 界面：

https://jupyterlab.readthedocs.io/en/stable/user/interface.html

文档。以下是一些在本课程中特别有用的关键功能：

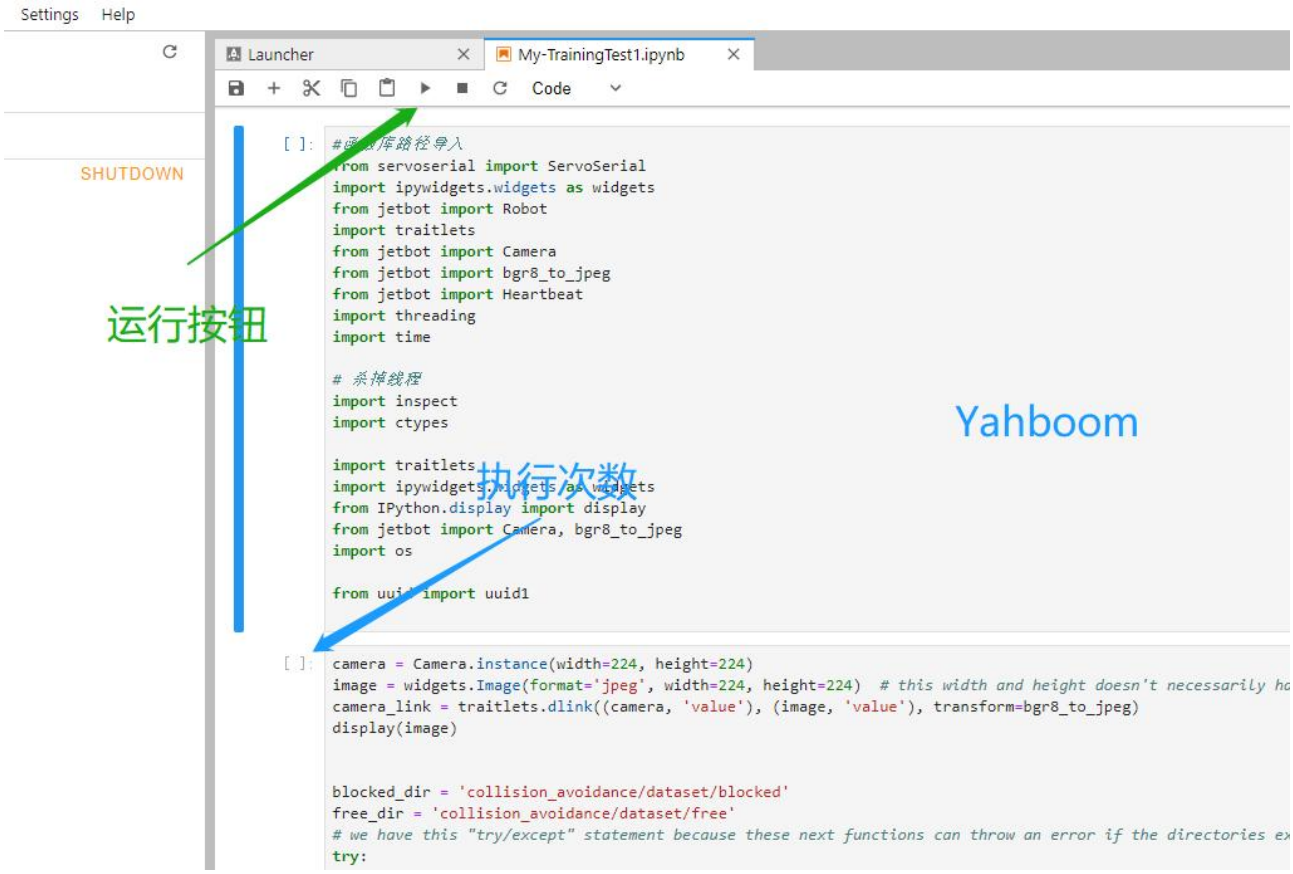
文件浏览器：

左侧栏中的文件浏览器允许导航 Jetson Orin Nano 文件结构。双击笔记本或文件会在主工作区中打开它。

- iPython 笔记本：

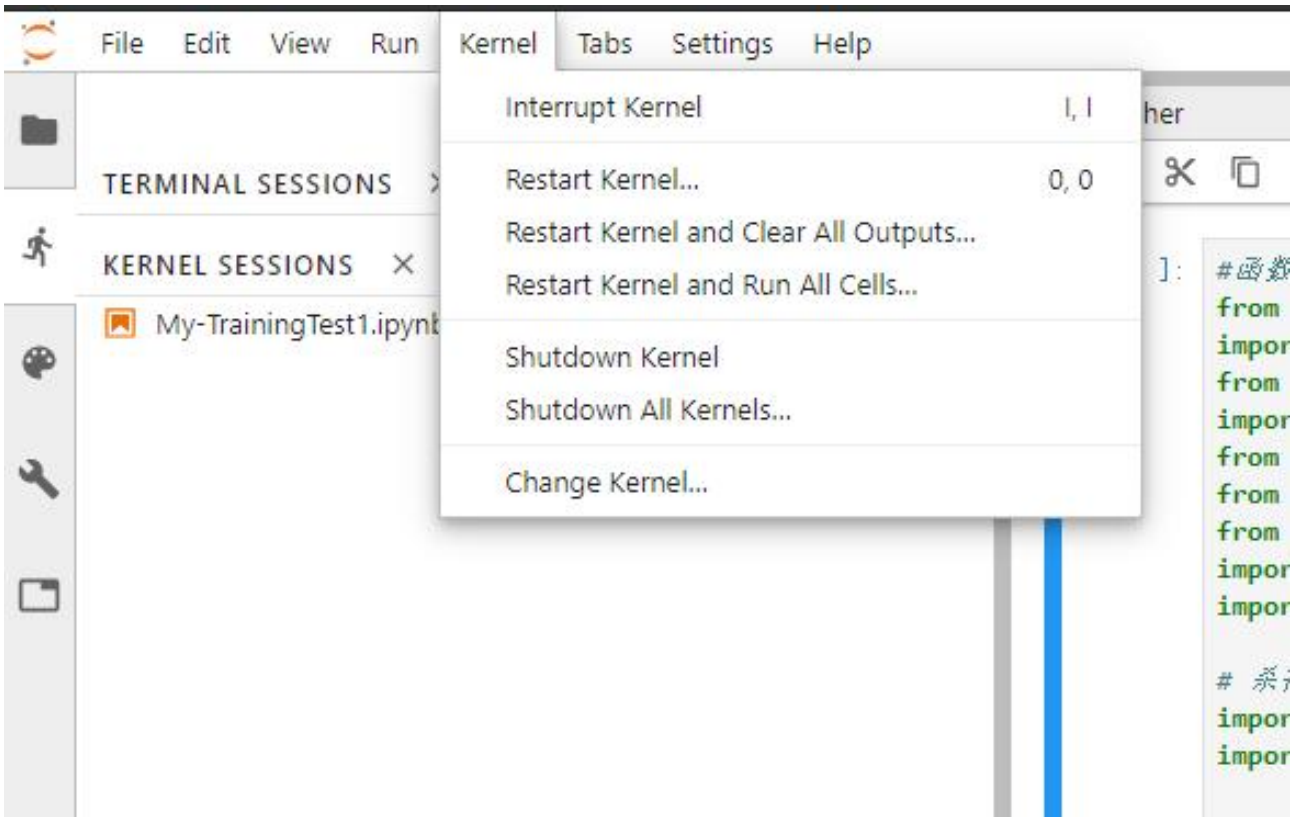
本课程中使用的交互式笔记本具有".ipynb"文件扩展名。从文件浏览器双击笔记本时，它将在主工作区中打开，其过程将开始。笔记本包括文本和代码"

单元格"。当代码单元"运行"时，通过单击笔记本顶部的运行按钮或键盘快捷键[CTRL] +[ENTER]，将执行单元格中的代码块，并显示输出（如果有）在笔记本电脑的下方。在每个可执行单元格的左侧，括号中有"执行计数"或"提示编号"。如果单元格运行时间超过几秒钟，您会在那里看到一个星号标记，表示单元格尚未完成执行。完成该单元格的处理后，括号中将显示一个数字。



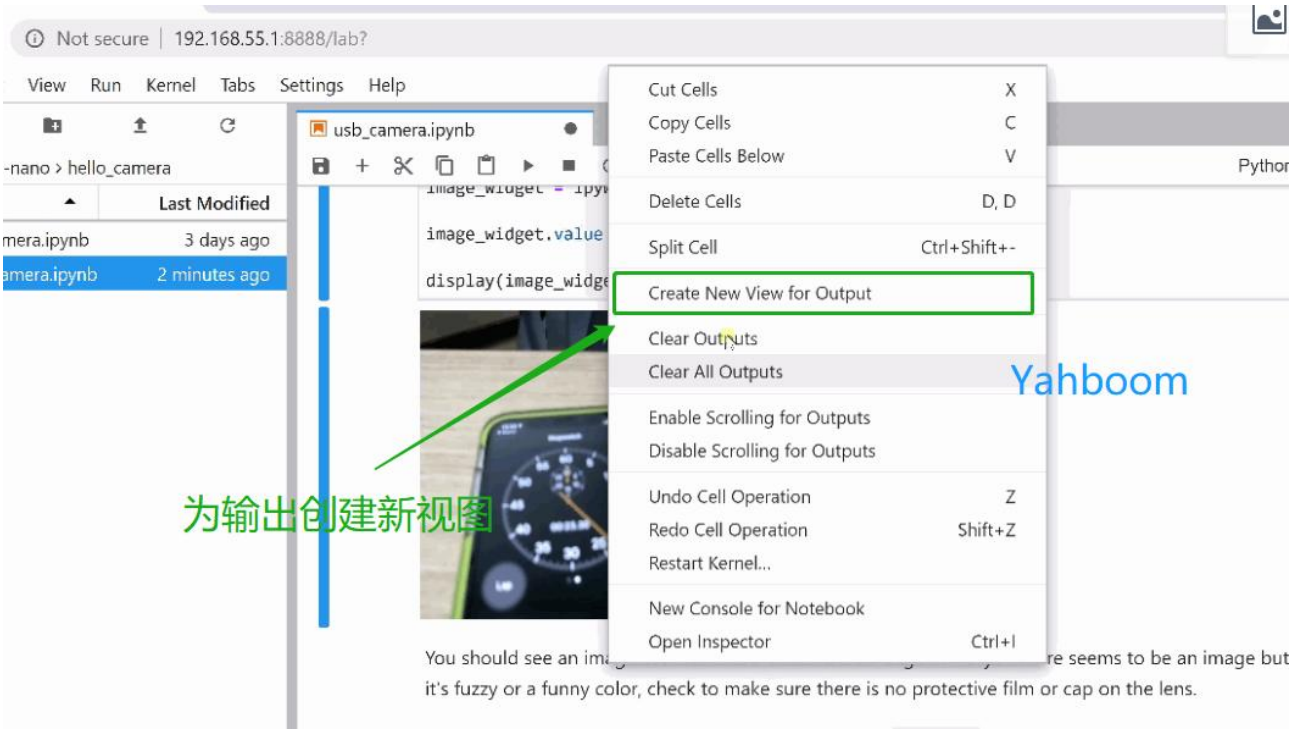
- 内核操作：

每个正在运行的笔记本的内核是一个运行用户代码的独立进程。从文件浏览器打开笔记本时，内核会自动启动。主菜单栏上的内核菜单包含关闭或重新启动内核的命令，您需要定期使用它们。内核关闭后，不能执行任何代码单元。重新启动内核时，所有内存都会因导入的包，变量赋值等而丢失。



单元格标签：

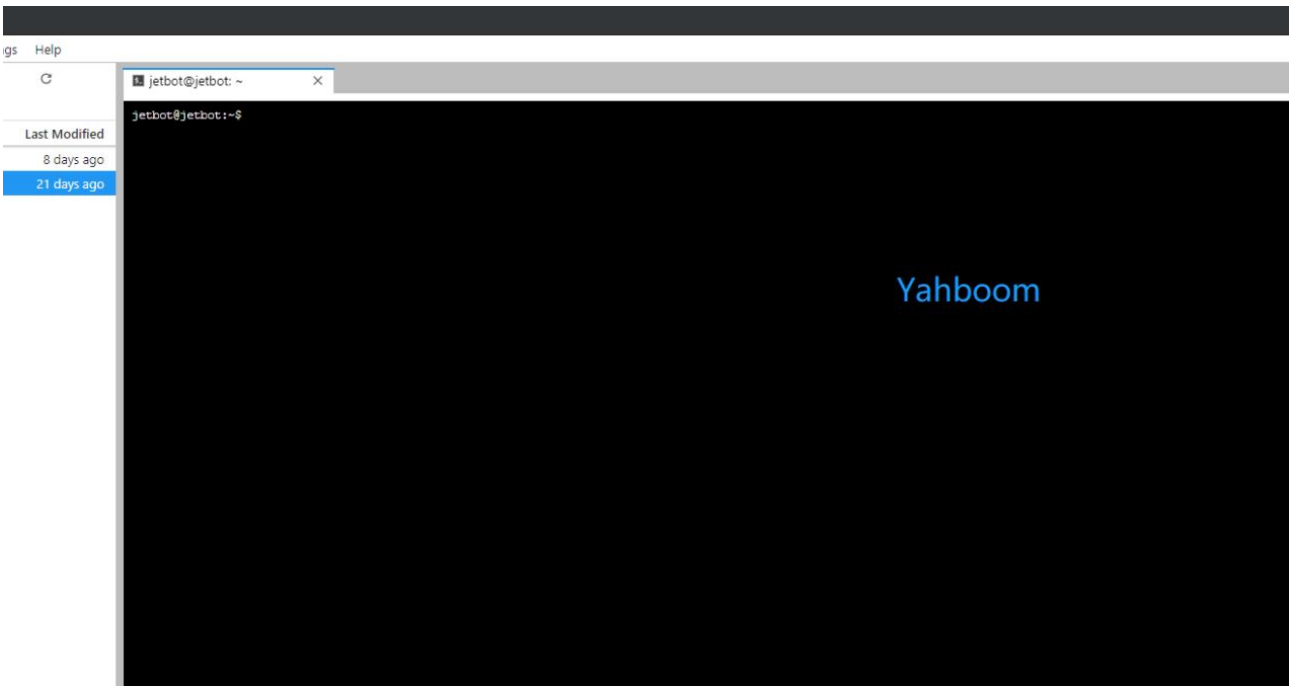
通过右键单击单元格并选择"为输出创建新视图"，可以将任何单元格移动到主工作区中的新窗口选项卡。这样，您可以在仍然观看特定单元格的同时继续向下滚动 JupyterLab 笔记本。这在包含相机视图的单元格中特别有用！



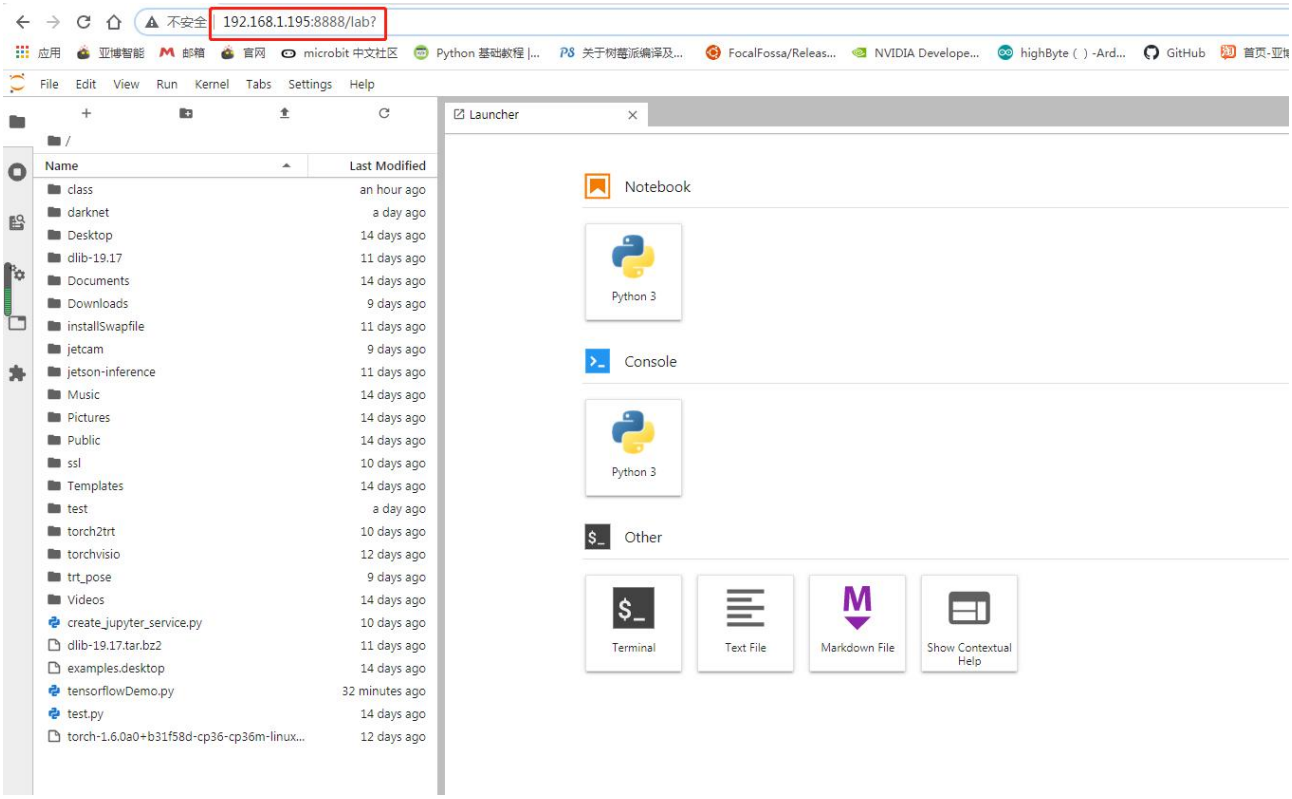
- 终端窗口：

您可以直接通过 Jupyter 远程登录在 JetbotUbuntu OS 的终端窗口中工作。

在 Launcher 页面中，单击"其他"下的终端图标。要调出 Launcher 页面，如果它不再可见，请单击左侧栏顶部的"+"图标。



JupyterLab 登录方式为在浏览器网页输入端输入 IP:8888



简单介绍下，接下来赶紧来安装吧(配置好的镜像无需安装，开机已经启动好 jupterlab 直接可以到网页上根据 IP 和密码登录，密码为 yahboom)

安装 nodejs 和 npm

```
sudo apt-get update
```

```
sudo apt install nodejs npm
```

但是用直接用上面这个命令安装后的版本是比较低的后续要安装 jupyterlab 插件可能会报错，用一下版本可以查看,至少要 12.3.0 版本的 node

```
node -v
```

```
npm -v
```

安装 n 模块，用这个模块来更新或者指定安装 node 的版本

```

```

```
npm install -g n
```

先说明下这个模块的功能，一下命令了解下先，不用操作

清除 npm 缓存：npm cache clean -f

安装 n 模块：npm install -g n

安装官方稳定版本：n stable

安装最新官方版本：n latest

安装某个指定版本：n 11.6.0

查看已安装的 node 版本: n

查看当前 node 版本：node -v

删除指定版本：n rm 7.5.0

好的，了解完 n 模块的功能后来安装对应版本的 node,也可以安装最新版的例如以下，

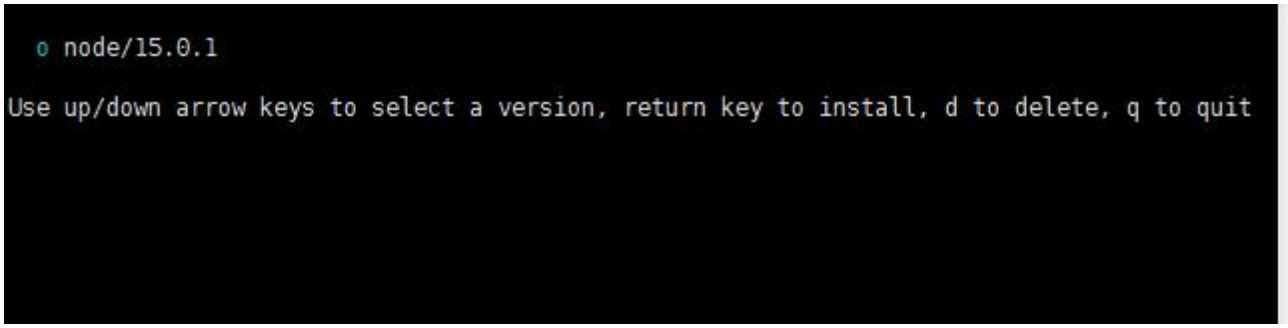
```

```

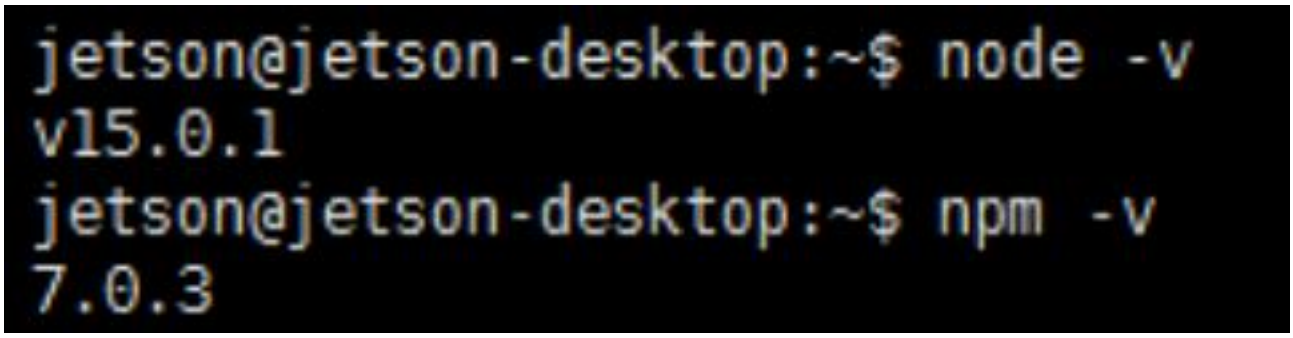
```
sudo n latest
```

安装好后，node -v 查看下版本，如果版本没有变，那么可以尝试重启下，如果还是没有变，执行

```
sudo n
```

会出现一个画面，可以看到我们安装过的 node 版本名，例如我们这里是 v15.0.1，通过上下方向按键控制光标选择这个版本，然后回车镜像安装，然后查看下版本，如果还是没有变，一般再重启下就可以了。





如果上面的命令报错，可参考以下链接

- 解决报错链接：https://blog.csdn.net/jabony/article/details/105288314?spm=1001.2101.3001.6650.3&utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-3-105288314-blog-101249978.2

35%5Ev27%5Epc_relevant_t0_download&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-3-105288314-blog-101249978.235%5Ev27%5Epc_relevant_t0_download&utm_relevant_index=4

- 换个方法安装的链接（需重烧镜像再操作）：

  https://www.cnblogs.com/mo3408/p/16828382.html

  安装 jupyterlab：（警告忽略，失败多次执行即可）

  ```
  sudo pip3 install jupyter jupyterlab

  sudo jupyter labextension install @jupyter-widgets/jupyterlab-manager

  sudo jupyter labextension install @jupyterlab/statusbar
  ```

  生成相应配置文件：（如果某个文件报权限问题，可以尝试用 sudo chmod 777 赋予权限）

  ```
  jupyter lab --generate-config
  ```

  设置进入 notebook 的密码(这里会要设置两次,第二次为确认输入的密码):

  ```
  jupyter notebook password
  ```

当第一次登录使用 notebook 时需要输入你在这里设置的密码才能进入，请务必记住的当前设置的密码！

设置开机自启动 jupterlab，create_jupyter_service.py 文件在环境配置的附件文件里面

运行 create_jupyter_service.py 文件产生 jupyter_service.service 文件

```
python3 create_jupyter_service.py
```

然后将产生的该服务文件移动至系统服务

```
sudo mv nano_jupyter.service
/etc/systemd/system/nano_jupyter.service
#使能该服务
sudo systemctl enable nano_jupyter.service
#  手动开启该服务
sudo systemctl start nano_jupyter.service
```

安装 jetcam

JetCam 是用于 NVIDIA Jetson 的易于使用的 Python 相机界面。使用 Jetson 的 Accelerated GStreamer 插件可与各种 USB 和 CSI 摄像机配合使用。轻松读取图像作为 numpy 数组 image = camera.read()。设置相机以 running

= True 将回调附加到新框架。JetCam 使在 Python 中创建 AI 项目的原型变得容易，尤其是在 JetCard 中安装的 Jupyter Lab 编程环境中。

接下来开始安装:

```
git clone <https://github.com/NVIDIA-AI-IOT/jetcam>

cd jetcam

sudo python3 setup.py install
```

详细的使用即函数可以到 https://github.com/NVIDIA-AI-IOT/jetcam 查看

3、安装 TensorFlow(选看)

4、Jetson Orin Nano 安装 TensorFlow GPU 教程

今天的目标是安装 TensorFlow GPU 版本，安装 TensorFlow GPU 版本需要成功配置好 CUDA。不过在安装 TensorFlow GPU 之前，有一些机器学习必须用到的安装包也需要来安装一下。注意配置好的镜像已安装好 tensorflow，无需安装。

1. 安装 pip

因为 Jetson Orin Nano 中已经安装了 Python3.8 版本，所以安装 pip 还是比较简单的

```
sudo apt-get install python3-pip python3-dev
```

安装后 pip 是比较旧的版本，需要把它升级到最新版 python3 -m pip install --upgrade pip #升级 pip

运行 pip3 -V 成功后显示

2. 安装那些机器学习领域非常重要的包

sudo apt-get install python3-numpy

(是 Python 语言的一个扩展程序库，支持大量的维度数组与矩阵运算，此外也针对数组运算提供大量的数学函数库。)

sudo apt-get install python3-scipy

(Scipy 是一个用于数学、科学、工程领域的常用软件包，可以处理插值、积分、优化、图像处理、常微分方程数值解的求解、信号处理等问题。)

sudo apt-get install python3-pandas

(pandas 是基于 NumPy 的一种工具，该工具是为了解决数据分析任务而创建的。Pandas 纳入了大量库和一些标准的数据模型，提供了高效地操作大型数据集所需的工具。pandas 提供了大量能使我们快速便捷地处理数据的函数

和方法。你很快就会发现，它是使 Python 成为强大而高效的数据分析环境的重要因素之一。）

sudo apt-get install python3-matplotlib

(Matplotlib 是一个 Python 的 2D 绘图库，它以各种硬拷贝格式和跨平台的交互式环境生成出版质量级别的图形)

sudo apt-get install python3-sklearn

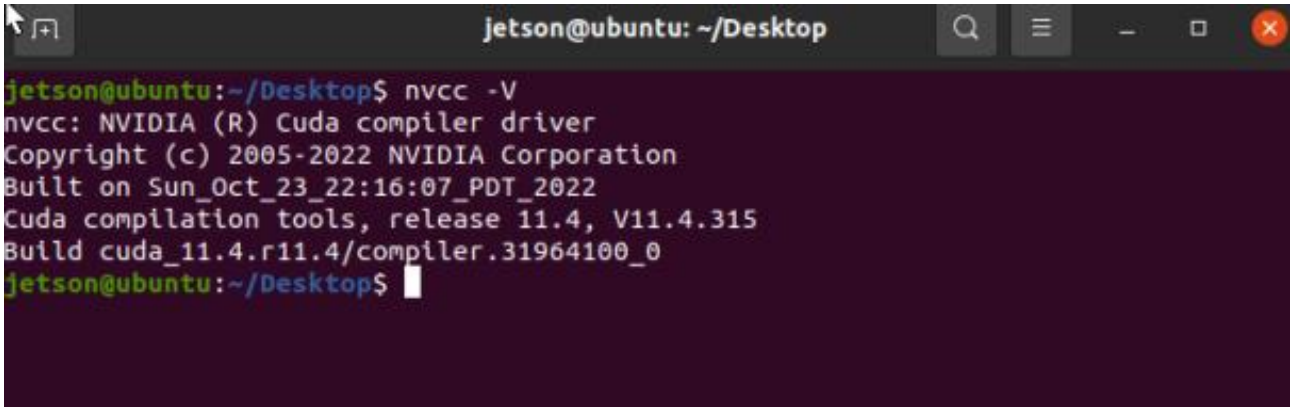
(简单高效的数据挖掘和数据分析工具)

3. 安装 TensorFlow GPU 版

（1）确认 CUDA 已经被正常安装

nvcc -V

如果能看到 CUDA 版本号，即为正确安装



如果报错参考这链接的解决方法

https://zhuanlan.zhihu.com/p/513220749

安装所需要的包

```
sudo apt-get install python3-pip
sudo python3 -m pip install --upgrade pip
sudo pip3 install -U testresources setuptools==65.5.0
```

安装 python 的依赖项

```
$ sudo pip3 install -U numpy==1.22 future==0.18.2 mock==3.0.5
keras_preprocessing==1.1.2 keras_applications==1.0.8 gast==0.4.0
protobuf pybind11 cython pkgconfig packaging h5py==3.6.0
```

1.

安装 TensorFlow GPU 版本（在线安装经常中断，建议使用离线安装）

2.

（3.1）在线安装

3.

```
$ sudo pip3 install --extra-index-url
https://developer.download.nvidia.com/compute/redist/jp/v51
tensorflow
```

4.

以下是官网的 tensorflow 安装说明。

https://docs.nvidia.com/deeplearning/frameworks/install-tf-jetson-platform/index.html#install

（3.2）离线安装

因为在线安装下载太慢了，所以我们可以选择离线包安装，安装包需要去网上下载。需要根据当前系统的 JetPack 版本安装对应的 TensorFlow。在我们的环境搭建附件里面也有存放一个离线包，但是需要看是否与您当前的系统的 jetpack 版本匹配。

https://docs.nvidia.com/deeplearning/frameworks/install-tf-jetson-platform-release-notes/tf-jetson-rel.html#tf-jetson-rel

TensorFlow Version	NVIDIA TensorFlow Container	JetPack Version
2.12.0	23.04	5.1.x
2.11.0	23.03, 23.02, 23.01	
2.10.1	22.12	5.0.2
2.10.0	22.11, 22.10	
2.9.1	22.09, 22.07	
	22.06	5.0.1
2.8.0	22.05, 22.04, 22.03	5.0

1）将 WHL 文件直接通过 winSCP 软件上传到 jeston Orin Nano 上的 /home/jetson 文件夹。

2）上传完成之后，输入指令(pip3 install +您对应的版本安装包)

pip3 install xxx.whl

下载途中可能也会需要在线安装一些软件包 直接 Y(YES)通过。

3）完成安装,输入以下指令检测 tensorflow 是否成功安装。

python3

import tensorflow as tf

没有报错这说明安装成功。

附录

其它参考教程：

https://forums.developer.nvidia.com/t/official-tensorflow-for-jetson-nano/71770

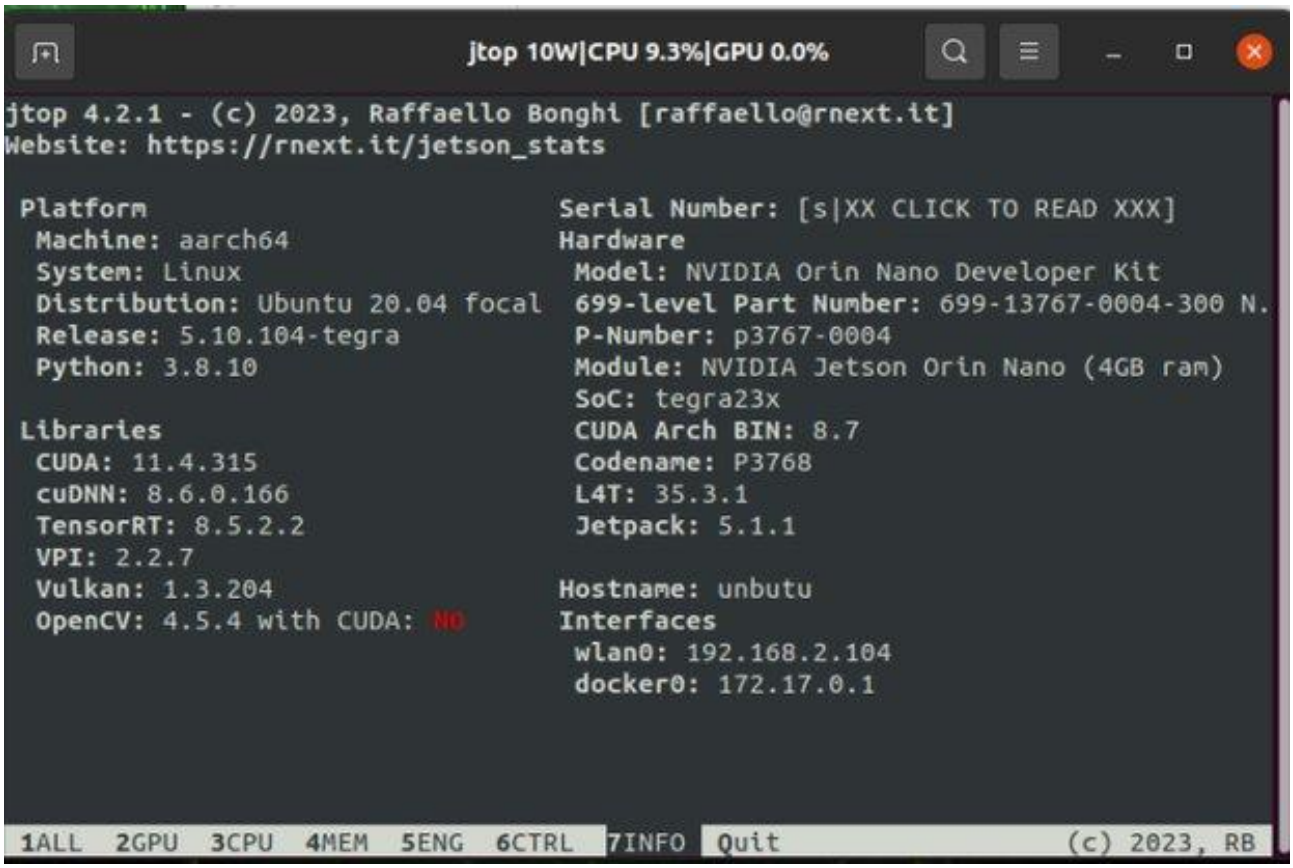
https://docs.nvidia.com/deeplearning/frameworks/install-tf-jetson-platform/index.html#install

4、安装 Torch&&Torchvision(选看)

安装 Torch&&Torchvision(选看)

使用前说明本教程针对自行搭建环境的用户，如果使用的是 yahboom 版的镜像的用户，可以忽略不看

本教程的系统配置如图所示：



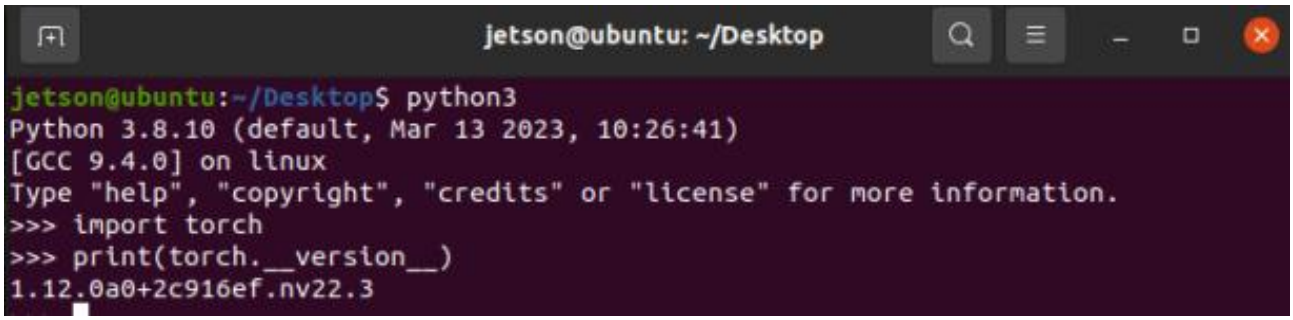
1.安装 torch

找到附件下的 torch 文件夹下的 whl 文件上传到 jieson orin Nano 上

```
pip3 install torch-xxx.whl
```

注意：如果直接 pip3 install torch 是没有带 GPU 版本的，后续训练模型可能会报错要找到带 GPU 版本的，必须要到 jetson 官网找 等待安装完成即

可



2.安装 torchvision

通过这个网址找到与 torch 相对应的版本

https://github.com/pytorch/vision **发现 torch 是 1.12，那么 torchvision 即为 0.13.0** 输入以下命令

```
sudo apt-get install libjpeg-dev zlib1g-dev libpython3-dev
libavcodec-dev libavformat-dev libswscale-dev
git clone --branch v0.13.0 https://github.com/pytorch/vision
torchvision
cd torchvision
export BUILD_VERSION=0.13.0
python3 setup.py install --user
```

等待安装完成即可

附录

其它参考链接：

https://blog.csdn.net/weixin_45463952/article/details/125857709?utm_medium=distribute.pc_relevant.none-task-blog-2~default~baidujs_baidulandingword~default-1-125857709-blog-128227652.235^v32^pc_relevant_default_base3&spm=1001.2101.3001.4242.2&utm_relevant_index=4

5、jetson-inference 环境搭建（选看）

jetson-inference 环境搭建

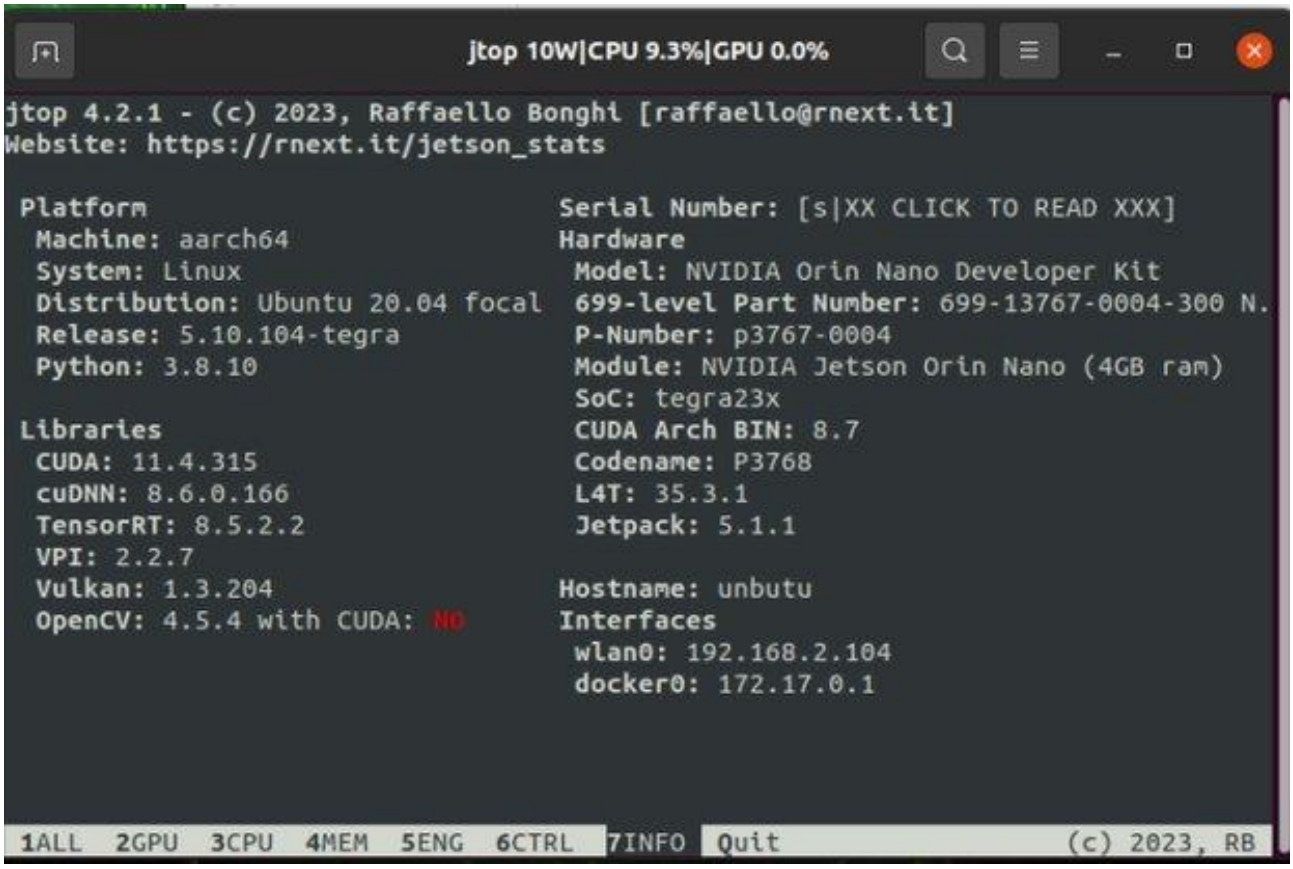
jetson-inference 环境搭建 1.使用前说明 2.本教程的环境版本配置如图所示：3. 开始搭建 3.1 下载需要的依赖 3.2 下载相关源码 3.3 下载相关的

python 模块 3.4 对文件进行修改 4.安装模型 5.开始编译 6.进行验证是否安装成功附录

1.使用前说明

本教程适用于自主搭建 jetson nano 的镜像，直接使用 YAHBOOM 版的镜像可忽略给教程。

2.本教程的环境版本配置如图所示：



如果不想自主完全搭建，可以使用我们提供好的 jetson-inference 压缩包，把压缩包传入到 jetson orin Nano 中，解压，直接从"安装模块"开始看

3. 开始搭建

3.1 下载需要的依赖

```
sudo apt-get update
sudo apt-get install git cmake
```

3.2 下载相关源码

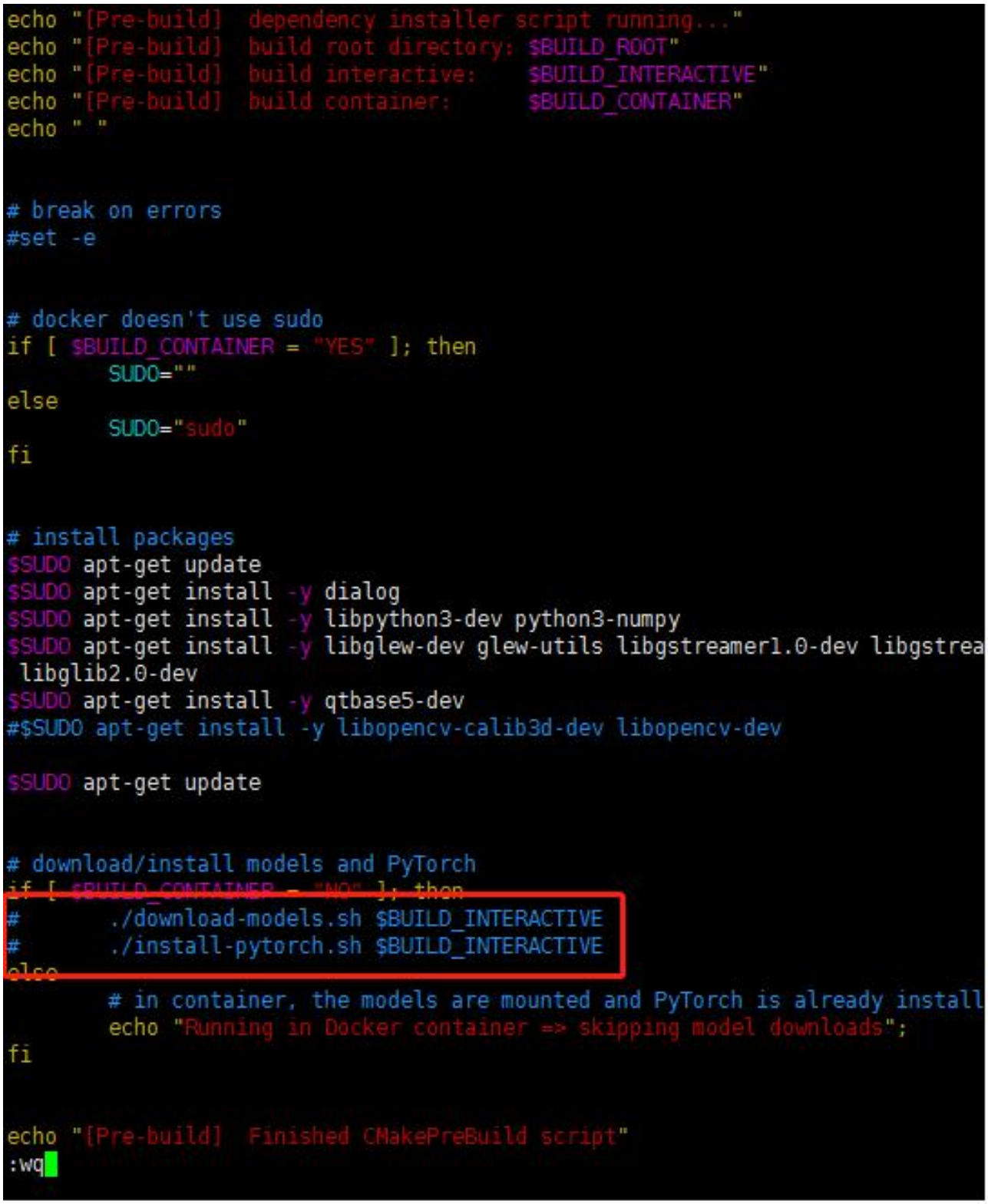
```
git clone https://github.com/dusty-nv/jetson-inference
cd jetson-inference
git submodule update --init ##如果中途报网络错误，需要做到科学上网，多运行几次，不然会下载不完整
```

3.3 下载相关的 python 模块

```
sudo apt-get install libpython3-dev python3-numpy
sudo apt-get install python3-scipy
sudo apt-get install python3-pandas
sudo apt-get install python3-matplotlib
sudo apt-get install python3-sklearn
```

3.4 对文件进行修改

编辑 jetson-inference/CMakePrebuild.sh。把./download-models.sh 注释掉,（前面加个#注释）如图）



```
echo "[Pre-build]  dependency installer script running..."
echo "[Pre-build]  build root directory: $BUILD_ROOT"
echo "[Pre-build]  build interactive:    $BUILD_INTERACTIVE"
echo "[Pre-build]  build container:      $BUILD_CONTAINER"
echo " "


# break on errors
#set -e


# docker doesn't use sudo
if [ $BUILD_CONTAINER = "YES" ]; then
        SUDO=""
else
        SUDO="sudo"
fi


# install packages
$SUDO apt-get update
$SUDO apt-get install -y dialog
$SUDO apt-get install -y libpython3-dev python3-numpy
$SUDO apt-get install -y libglew-dev glew-utils libgstreamer1.0-dev libgstrea
 libglib2.0-dev
$SUDO apt-get install -y qtbase5-dev
#$SUDO apt-get install -y libopencv-calib3d-dev libopencv-dev

$SUDO apt-get update


# download/install models and PyTorch
if [ $BUILD_CONTAINER = "NO" ]; then
#       ./download-models.sh $BUILD_INTERACTIVE
#       ./install-pytorch.sh $BUILD_INTERACTIVE
else
        # in container, the models are mounted and PyTorch is already install
        echo "Running in Docker container => skipping model downloads";
fi


echo "[Pre-build]  Finished CMakePreBuild script"
:wq
```

4.安装模型

方法 1：可以执行以下的步骤

```
cd jetson-inference/tools
./download-models.sh
```

进行选择后，会自动下载模型到 data/network 的文件路径当中，需要科学上网才能正常下载

方法 2：可以把我们提供在环境搭建的附件中找到 jetson-inference 需要的包，把里面的压缩包传到 jetso nano 的 jetson-inference/data/network 里面，然后进行解压 解压命令

```
for tar in *.tar.gz; do tar xvf $tar; done
```

注释：

1. 对于解压多个.gz 文件的，用此命令： for gz in *.gz; do gunzip $gz; done
2. 对于解压多个.tar.gz 文件的，用下面命令： for tar in *.tar.gz; do tar xvf $tar; done

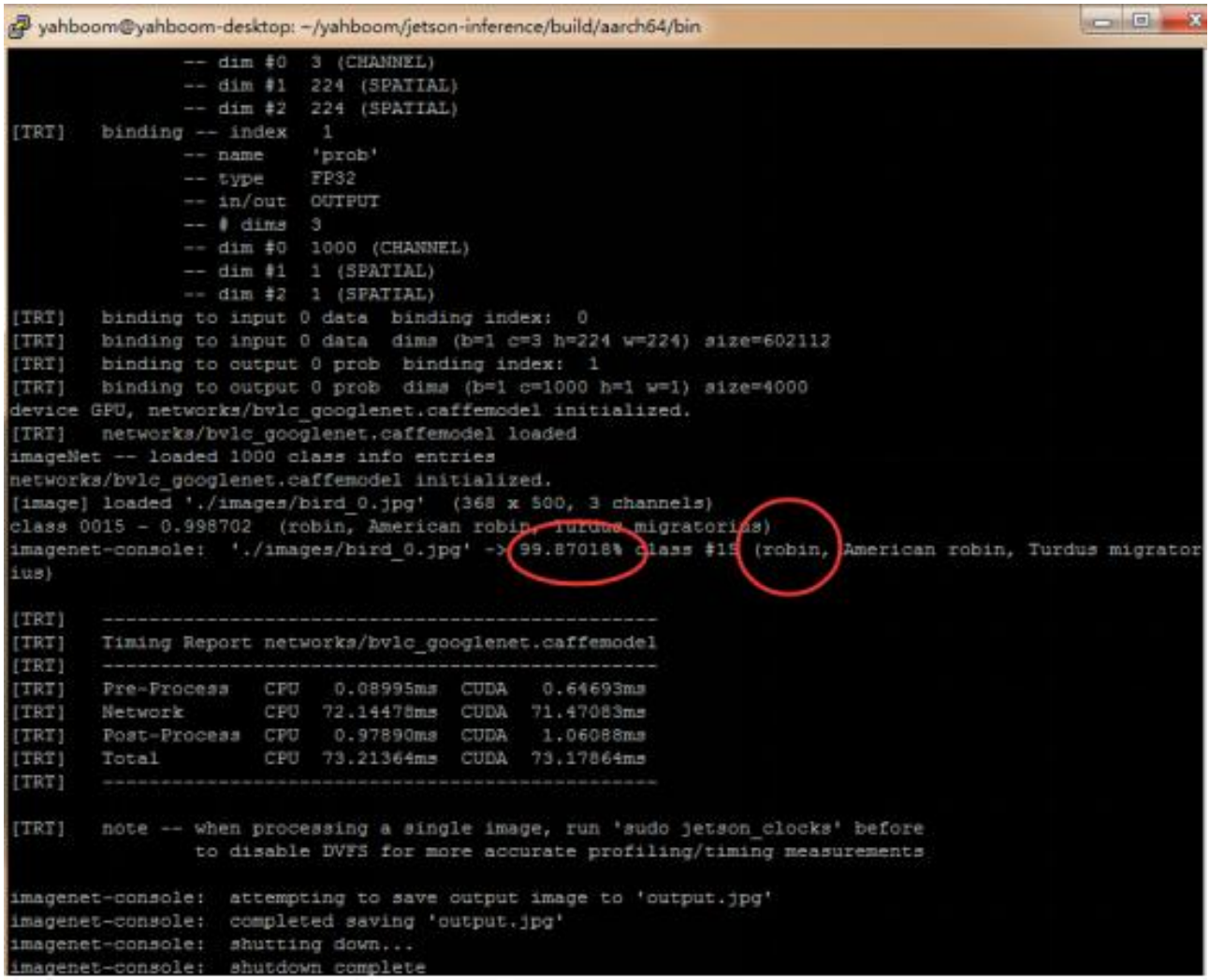
5.开始编译

```
cd jetson-inference
mkdir build   #用我们提供给的包，这句可省略
cd build
cmake ../
make (或者 make -j8)     #（在 build 目录下）
sudo make install        #（在 build 目录下）
```

如果中途有报错，说明源码下载不全,请回到 3.2 步骤多执行 git submodule update --init 这行命令，或者到浏览器下载，方法可百度

6.进行验证是否安装成功

```
cd jetson-inference/build/aarch64/bin

./imagenet-console ./images/bird_0.jpg output.jpg
# 执行等待许久后出现如下（第一次需要很长时间，后面执行就会很快）
```



找到对应目录下查看 output.jpg 如下，会在图片上端显示识别结果。

附录

其它参考教程：

1.https://blog.csdn.net/aal779/article/details/122055432

2.https://github.com/dusty-nv/jetson-inference/blob/master/docs/building-repo-2.md

6、Hello AI World

0.简介

NVIDIA TensorRT™是一个高性能深度学习推理平台。它包括深度学习推理优化器和运行时，可为深度学习推理应用程序提供低延迟和高吞吐量。在推理期间，基于 TensorRT 的应用程序比仅 CPU 平台的执行速度快 40 倍。使用 TensorRT，您可以优化在所有主要框架中培训的神经网络模型，以高精度校准低精度，最后部署到超大规模数据中心，嵌入式或汽车产品平台。

TensorRT 构建于 NVIDIA 的并行编程模型 CUDA 之上，使您能够利用 CUDA-X AI 中的库，开发工具和技术，为人工智能，自动机器，高性能计算和图形优化所有深度学习框架的推理。

ensorRT 为深度学习推理应用的生产部署提供 INT8 和 FP16 优化，例如视频流，语音识别，推荐和自然语言处理。降低精度推断可显着减少应用程序延迟，这是许多实时服务，自动和嵌入式应用程序的要求。

Hello AI World 可以完全在 Jetson 上运行，包括使用 TensorRT 进行推理和使用 PyTorch 进行迁移学习。Hello AI World 的推理部分——包括为 Python 或 C++编写自己的图像分类和对象检测应用程序，以及现场摄像机演示——可以在大约两个小时或更短的时间内在 Jetson 上运行，而迁移学习最好让它在一夜之间运行。

2. 图像分类推理

7、图像分类推理

1.ImageNet 对图像进行分类

有多种类型的深度学习网络可用，包括识别、检测/定位和语义分割。我们在本教程中强调的第一个深度学习功能是图像识别，它使用在大型数据集上训练的分类网络来识别场景和对象。

imageNet 对象接受输入图像并输出每个类别的概率。在 1000 个对象的 ImageNet ILSVRC 数据集上进行了训练后，GoogleNet 和 ResNet-18 模型在构建步骤中自动下载。有关可以下载和使用的其他分类模型，我们提供了 C++和 Python 的示例程序

在 Jetson 上使用 ImageNet 程序：

首先，让我们尝试使用 imagenet 程序在一些示例图像上测试 imagenet 识别。它加载一个或多个图像，使用 TensorRT 和 imageNet 类进行推理，然

后覆盖分类结果并保存输出图像。该项目在 images/目录下提供了供您使用的示例图像。

构建项目后，请确保您的终端位于 aarch64/bin 目录中：

```
cd jetson-inference/build/aarch64/bin
```

接下来，让我们使用 imagenet 程序对示例图像进行分类，使用 C++或 Python 变体。如果您使用的是 Docker 容器，建议将已分类的输出映像保存到 images/test 挂载的目录中。然后，这些图像将很容易从主机设备的 jetson 推理/data/images/test 目录中查看

```
# C++
$ ./imagenet images/orange_0.jpg images/test/output_0.jpg      # (default network is googlenet)

# Python
$ ./imagenet.py images/orange_0.jpg images/test/output_0.jpg   # (default network is googlenet)
```

注意：第一次运行每个模型时，TensorRT 将花费几分钟时间来优化网络。 这个经过优化的网络文件随后被缓存到磁盘上，因此将来使用该模型运行会更快地加载

除了加载单个图像外，您还可以加载其他图像的目录或视频文件。

2.下载其他分类模型

以下预先训练的图像分类模型可供使用，并将自动下载：

Network	CLI argument	NetworkType enum
AlexNet	`alexnet`	`ALEXNET`
GoogleNet	`googlenet`	`GOOGLENET`
GoogleNet-12	`googlenet-12`	`GOOGLENET_12`
ResNet-18	`resnet-18`	`RESNET_18`
ResNet-50	`resnet-50`	`RESNET_50`
ResNet-101	`resnet-101`	`RESNET_101`
ResNet-152	`resnet-152`	`RESNET_152`
VGG-16	`vgg-16`	`VGG-16`
VGG-19	`vgg-19`	`VGG-19`
Inception-v4	`inception-v4`	`INCEPTION_V4`

通常，更复杂的网络可以具有更高的分类精度，并增加了运行时间。

使用不同的分类模型：

您可以通过将命令行上的--network 标志设置为上表中相应的 CLI 参数之一来指定要加载的模型。默认情况下，如果未指定可选的--network 标志，则会加载 GoogleNet。

构建项目后，请确保您的终端位于 aarch64/bin 目录中：

```
cd jetson-inference/build/aarch64/bin
```

下面是使用 ResNet-18 模型的一些示例：

```
# C++
$ ./imagenet --network=resnet-18 images/dog_2.jpg
images/test/output_dog.jpg

# Python
$ ./imagenet.py --network=resnet-18 images/dog_2.jpg
images/test/output_dog.jpg
```

注意：要是自己搭建环境的情况下得自己下载 resnet-18.tar.gz 模型文件到 network 文件夹下并解压，才能运行上面的程序。使用我们提供的镜像就可以直接输入上面的程序

3.运行实时摄像头识别演示

我们之前使用的 imagenet.cpp/imagenet.py 样本也可以用于实时相机流。

支持的摄像机类型包括：

- MIPI CSI cameras (`csi://0`)
- V4L2 cameras (`/dev/video0`)
- RTP/RTSP streams (`rtsp://username:password@ip:port`)

以下是在相机订阅源上启动程序的一些典型场景。

C++

```
$ ./imagenet csi://0                    # MIPI CSI camera
$ ./imagenet /dev/video0                # V4L2 camera
$ ./imagenet /dev/video0 output.mp4     # save to video file
```

python

```
$ ./imagenet.py csi://0                 # MIPI CSI camera
$ ./imagenet.py /dev/video0             # V4L2 camera
$ ./imagenet.py /dev/video0 output.mp4  # save to video file
```

OpenGL 窗口中显示的是实时摄像机流、分类对象名称、分类对象的置信度以及网络的帧速率。在 Jetson Nano 上，GoogleNet 和 ResNet-18 的帧速率应该高达约 75 FPS（比其他 Jetson 的更快）。

该应用程序可以识别多达 1000 种不同类型的对象，因为分类模型是在包含 1000 类对象的 ILSVRC ImageNet 数据集上训练的。1000 种类型对象的名称映射，可以在 data/networks/ilsvrc12_synset_words.txt 下的 repo 中找到

Hello AI World 关于图像分类的教程的这一部分到此结束。接下来，我们将开始使用对象检测网络，它为我们提供每帧多个对象的边界框坐标。

8、训练图像分类模型

训练图像分类模型

训练图像分类模型 1.训练前的准备 2.开始对猫狗数据集重新训练 3.执行训练的命令 4.训练开始的信息解读 5.模型精度讲解 6.训练完毕后，把模型转成一个 onNano 的模型文件 7.用 TensorRT 处理图像 8. 1 个训练周期、35 个训练周期、100 个训练周期的结果 9.附录

1.训练前的准备

在这个教程开始前，python 必须安装好 **torchCPU+GPU 版本**，Yahboom 的镜像是已经安装的了。如果没安装，请自行安装，需要科学上网，也可以看本系列教程的 torch 的安装教程。

```
cd jetson-inference/build
```

./install-pytorch.sh

2.开始对猫狗数据集重新训练

```
cd jetson-inference/python/training/classification/data
wget https://nvidia.box.com/shared/static/o577zd8yp3lmxf5zhm38svrbrv45am3y.gz -O cat_dog.tar.gz
tar xvzf cat_dog.tar.gz
```

如果图片因为网络问题下载不成功，可以到附件中的"模型/cat_dog"中找到 cat_dog.tar.gz 传输到 Nano 上，yahboom 可以跳过此步骤

3.执行训练的命令

```
cd jetson-inference/python/training/classification
python3 train.py --model-dir=models/cat_dog data/cat_dog
```

注意:如果内存不足导致训连过程中被“终止”,请尝试交换虚拟内存和禁用桌面 GUI.

```
#禁用桌面 GUI
sudo init 3   # stop the desktop
sudo init 5   # restart the desktop
```

交换虚拟内存（参考链接）：

https://github.com/dusty-nv/jetson-inference/blob/master/docs/pytorch-transfer-learning.md#mounting-swap

4.训练开始的信息解读

```
Use GPU: 0 for training
=> dataset classes:  2 ['cat', 'dog']
=> using pre-trained model 'resnet18'
=> reshaped ResNet fully-connected layer with:
Linear(in_features=512, out_features=2, bias=True)
Epoch: [0][  0/625] Time  0.932 ( 0.932)   Data  0.148 ( 0.148)
Loss 6.8126e-01 (6.8126e-01)   Acc@1  50.00 ( 50.00)   Acc@5
100.00 (100.00)
Epoch: [0][ 10/625] Time  0.085 ( 0.163)   Data  0.000 ( 0.019)
Loss 2.3263e+01 (2.1190e+01)   Acc@1  25.00 ( 55.68)   Acc@5
100.00 (100.00)
Epoch: [0][ 20/625] Time  0.079 ( 0.126)   Data  0.000 ( 0.013)
Loss 1.5674e+00 (1.8448e+01)   Acc@1  62.50 ( 52.38)   Acc@5
100.00 (100.00)
```

```
Epoch: [0][ 30/625] Time  0.127 ( 0.114)   Data  0.000 ( 0.011)
Loss 1.7583e+00 (1.5975e+01)   Acc@1  25.00 ( 52.02)   Acc@5
100.00 (100.00)
Epoch: [0][ 40/625] Time  0.118 ( 0.116)   Data  0.000 ( 0.010)
Loss 5.4494e+00 (1.2934e+01)
```

Epoch: [0]:代表训练到第几个周期 0：第一个 如果没指定参数，默认是只训练 35 个周期 **可以使用--epochs=N 指定训练多少周期 N=100,即训练 100 个周期**

[N/625]:是您所在时代的当前图像批次小批量处理训练图像以提高性能,默认的批处理大小是 8 个图像，可以用--batch=N 将括号中的数字乘以批次大小(例如批次[100/625]->图像[800/5000]) -Time:当前图像批次的处理时间(秒) -Data:当前图像批次的磁盘加载时间(秒) -Loss:模型产生的累积误差(预期误差和预测误差) **如果你想训练 10 次，批处理的图像为 6，即运行一下命令**

```
python3 train.py --model-dir=models/cat_dog data/cat_dog
--epochs=10 --batch=6
```

5.模型精度讲解

在这个由 5000 幅图像组成的数据集上，在 Jetson orin Nano 上训练 ResNet-18 每个时期大约需要 2 分钟，或者大约 55 分钟来训练模型达到 35 个时期和 80%的分类精度。下图用于分析历元的训练进度与模型准确性的关

系:　　在大约第 30 个时期，ResNet-18 模型达到 80%的精确度，在第 65 个时期，它收敛于 82.5%的精确度。有了额外的训练时间，您可以通过增加数据集的大小来进一步提高准确性或者尝试更复杂的模型。 默认情况下，训练脚本设置为运行 35 个时期，但是如果您不希望等待那么长时间来测试您的模型，您可以提前退出训练并继续下一步(可以选择稍后从您停止的地方重新开始训练)。**通过--resume 这个参数进行恢复** 我们的**附录/模型文件/cat_dog** 中提供了训练 35 个周期的模型和 100 个周期的模型，如果你不想训练，可以自己把模型传到 Nano 下的**你的路径/jetson-inference/python/training/classification/models/cat_dog** 解压就好，yahboom 的镜像则不需要操作，里面已经存在 100 次的训练后的模型

6.训练完毕后，把模型转成一个 onNano 的模型文件

onNano 的模型文件方便 TensorRT 进行处理

```
python3 onNano_export.py --model-dir=models/cat_dog
```

这将创建一个名为 resnet18.onNano 下面的 jetson-inference/python/training/classification/models/cat_dog/

7.用 TensorRT 处理图像

以 yahboom 镜像的命令为例

```
```

```
cd /home/jetson/jetson-inference/python/training/classification
export NET=models/cat_dog
export DATASET=data/cat_dog
imagenet.py --model=$NET/resnet18.onNano --input_blob=input_0
--output_blob=output_0 --labels=$DATASET/labels.txt
$DATASET/test/cat/01.jpg cat.jpg
```

以上的命令执行完毕，可以在下面的路径观察的到结果，该命令是处理一张图片 路径:

/home/jetson/jetson-inference/python/training/classification

如果你想要测试所有的图片可以输入一下命令

```
cd /home/jetson/jetson-inference/python/training/classification
export NET=models/cat_dog
export DATASET=data/cat_dog
imagenet --model=$NET/resnet18.onNano --input_blob=input_0
--output_blob=output_0 --labels=$DATASET/../labels.txt \
        $DATASET/test/dog ./data/cat_dog/test_output_dog
```

输出的结果在

/home/jetson/jetson-inference/python/training/classification/data/cat_dog/test_output_dog 路径下

如果你想要打开摄像头进行检测

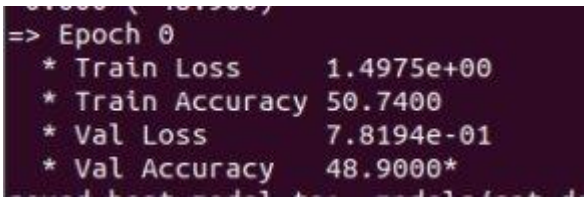
```
imagenet.py --model=$NET/resnet18.onNano --input_blob=input_0
--output_blob=output_0 --labels=$DATASET/labels.txt csi://0      #对
CSI 摄像头

imagenet.py --model=$NET/resnet18.onNano --input_blob=input_0
--output_blob=output_0 --labels=$DATASET/labels.txt
v4l2:///dev/video0      #对 v4l2 摄像头
```

其它视频流文件的打开方式，可参考：

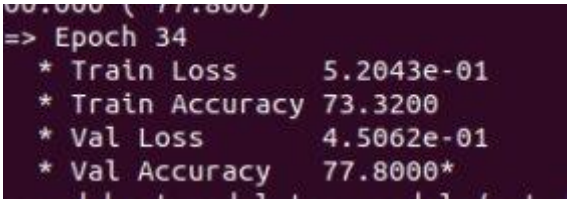
https://github.com/dusty-nv/jetson-inference/blob/master/docs/aux-streaming.md

8. 1 个训练周期、35 个训练周期、100 个训练周期的结果

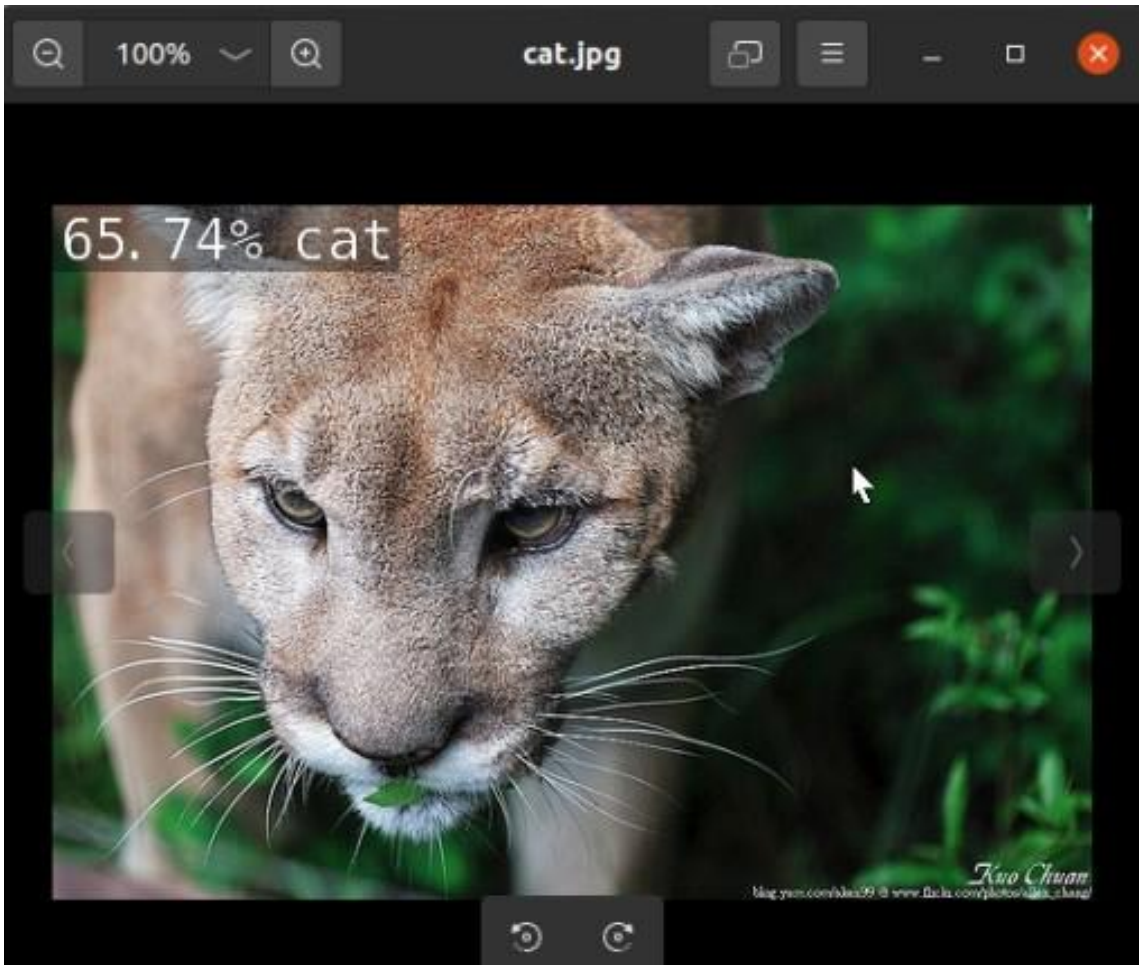
- 1 个训练周期



- 35 个训练周期



- 35 个训练周期识别结果



- 100 个训练周期识别结果

可以看到 100 个训练周期的结果比 35 个训练周期的结果的精度大大提升。

9.附录

如果你想对自己的各种类别不同的图片进行训练 可以参考以下的链接：

https://github.com/dusty-nv/jetson-inference/blob/master/docs/pytorch-collect.md

9、目标检测推理

目标检测推理

1.使用 DetectNet 定位对象

前面的识别示例输出表示整个输入图像的类概率。接下来，我们将重点关注对象检测，并通过提取边界框来查找帧中各种对象的位置。与图像分类不同，对象检测网络能够每帧检测许多不同的对象。

detectNet 对象接受图像作为输入，并输出检测到的边界框的坐标列表及其类别和置信度值。detectNet 可以从 Python 和 C++中使用。有关可供下载的各种预训练检测模型，请参见下文。使用的默认模型是在 MS COCO 数据集上训练的 91 类 SSD-Mobilenet-v2 模型，该模型在 Jetson 上使用 TensorRT 实现了实时推理性能。

2.从图像中检测对象

首先，让我们尝试使用 detectnet 程序来定位静态图像中的对象。除了输入/输出路径之外，还有一些额外的命令行选项：

- 更改正在使用的检测模型的网络标志（默认为 SSD-Mobilenet-v2）（可选）
- optional--overlay 标志，可以是以逗号分隔的 box、line、labels、conf 和 none 的组合。

  默认值为--overlay=box、labels，conf，它显示框、标签和置信度值。

  长方体选项绘制填充的边界框，而直线仅绘制未填充的轮廓
- optional--alpha 值，它设置叠加过程中使用的 alpha 混合值（默认值为 120）。
- 阈值，用于设置检测的最小阈值（默认值为 0.5）。

如果您使用的是 Docker 容器，建议将输出图像保存到 images/test 挂载的目录中。然后，在 jetson 推理/data/images/test 下，可以很容易地从主机设备上查看这些图像.

注：运行案例之前，要是自己搭建环境的情况下得自己下载 resnet-18.tar.gz 模型文件到 network 文件夹下才能运行上面的程序。使用我们提供的镜像就可以直接输入上面的程序。

请确保您的终端位于 aarch64/bin 目录中：

```
cd jetson-inference/build/aarch64/bin
```

以下是使用默认的 SSD-Mobilenet-v2 模型在图像中检测行人的一些示例：

```
# C++
$ ./detectnet --network=ssd-mobilenet-v2 images/xingren.png images/test/output_xinren.png

# Python
$ ./detectnet.py --network=ssd-mobilenet-v2 images/xingren.png images/test/output_xinren.png
```

注意：第一次运行每个模型时，TensorRT 将花费几分钟时间来优化网络。 这个经过优化的网络文件随后被缓存到磁盘上，因此将来使用该模型的运行将更快地加载。

3.处理视频文件

可以在 images 文件夹里存放视频，路径为

```
cd jetson-inference/build/aarch64/bin
```

运行程序：

```
# C++
./detectnet pedestrians.mp4 images/test/pedestrians_ssd.mp4

# Python
./detectnet.py pedestrians.mp4 images/test/pedestrians_ssd.mp4
```

效果：

可以使用--threshold 设置来向上或向下更改检测灵敏度（默认值为 0.5）。

4.运行实时摄像头识别演示

我们之前使用的 detectnet.cpp/detectnet.py 样本也可以用于实时相机流。支持的摄像机类型包括：

- MIPI CSI cameras (csi://0)
- V4L2 cameras (/dev/video0)
- RTP/RTSP streams (rtsp://username:password@ip:port)

以下是在相机订阅源上启动程序的一些典型场景。

C++

```
$ ./detectnet csi://0                          # MIPI CSI camera
$ ./detectnet /dev/video0                     # V4L2 camera
$ ./detectnet /dev/video0 output.mp4       # save to video file
```

python

```
$ ./detectnet.py csi://0                          # MIPI CSI camera
$ ./detectnet.py /dev/video0                     # V4L2 camera
$ ./detectnet.py /dev/video0 output.mp4  # save to video file
```

OpenGL 窗口中显示的是覆盖有检测到的对象的边界框的实时摄像机流。请注意，基于 SSD 的机型目前具有最高的性能。这是一个使用模型的例子：

如果在视频馈送中没有检测到所需的对象，或者您得到了虚假的检测，请尝试使用--threshold 参数降低或增加检测阈值（默认值为 0.5）。

训练图像分类模型

训练图像分类模型 1.训练前的准备 2.开始对猫狗数据集重新训练 3.执行训练的命令 4.训练开始的信息解读 5.模型精度讲解 6.训练完毕后，把模型转成一个 onNano 的模型文件 7.用 TensorRT 处理图像 8. 1 个训练周期、35 个训练周期、100 个训练周期的结果 9.附录

1.训练前的准备

在这个教程开始前，python 必须安装好 **torchCPU+GPU 版本**，Yahboom 的镜像是已经安装的了。如果没安装，请自行安装，需要科学上网，也可以看本系列教程的 torch 的安装教程。

```
cd jetson-inference/build
./install-pytorch.sh
```

2.开始对猫狗数据集重新训练

```
cd jetson-inference/python/training/classification/data
wget https://nvidia.box.com/shared/static/o577zd8yp3lmxf5zhm38svrbrv45am3y.gz -O cat_dog.tar.gz
tar xvzf cat_dog.tar.gz
```

如果图片因为网络问题下载不成功，可以到附件中的"模型/cat_dog"中找到 cat_dog.tar.gz 传输到 Nano 上，yahboom 可以跳过此步骤

3.执行训练的命令

```
cd jetson-inference/python/training/classification
python3 train.py --model-dir=models/cat_dog data/cat_dog
```

注意:如果内存不足导致训连过程中被“终止”,请尝试交换虚拟内存和禁用桌面 GUI.

```
#禁用桌面 GUI
sudo init 3   # stop the desktop
sudo init 5   # restart the desktop
```

交换虚拟内存（参考链接）：

https://github.com/dusty-nv/jetson-inference/blob/master/docs/pytorch-transfer-learning.md#mounting-swap

4.训练开始的信息解读

```
Use GPU: 0 for training
=> dataset classes:  2 ['cat', 'dog']
=> using pre-trained model 'resnet18'
=> reshaped ResNet fully-connected layer with:
Linear(in_features=512, out_features=2, bias=True)
Epoch: [0][  0/625] Time  0.932 ( 0.932)   Data  0.148 ( 0.148)
Loss 6.8126e-01 (6.8126e-01)   Acc@1  50.00 ( 50.00)   Acc@5
100.00 (100.00)
Epoch: [0][ 10/625] Time  0.085 ( 0.163)   Data  0.000 ( 0.019)
Loss 2.3263e+01 (2.1190e+01)   Acc@1  25.00 ( 55.68)   Acc@5
100.00 (100.00)
Epoch: [0][ 20/625] Time  0.079 ( 0.126)   Data  0.000 ( 0.013)
Loss 1.5674e+00 (1.8448e+01)   Acc@1  62.50 ( 52.38)   Acc@5
100.00 (100.00)
```

```
Epoch: [0][ 30/625] Time  0.127 ( 0.114)   Data  0.000 ( 0.011)
Loss 1.7583e+00 (1.5975e+01)   Acc@1  25.00 ( 52.02)   Acc@5
100.00 (100.00)
Epoch: [0][ 40/625] Time  0.118 ( 0.116)   Data  0.000 ( 0.010)
Loss 5.4494e+00 (1.2934e+01)
```

Epoch: [0]:代表训练到第几个周期 0：第一个 如果没指定参数，默认是只训练 35 个周期 **可以使用--epochs=N 指定训练多少周期 N=100,即训练 100 个周期**

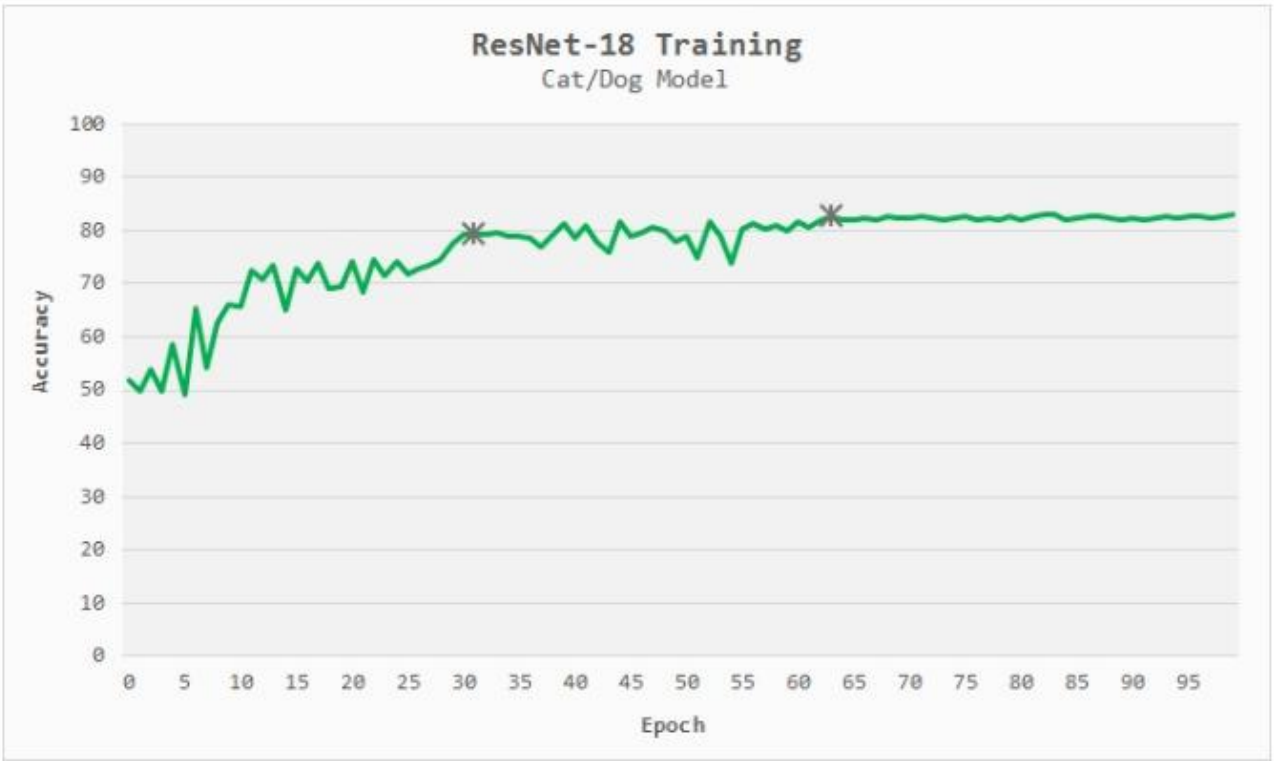
[N/625]:是您所在时代的当前图像批次小批量处理训练图像以提高性能,默认的批处理大小是 8 个图像，可以用--batch=N 将括号中的数字乘以批次大小(例如批次[100/625]->图像[800/5000]) -Time:当前图像批次的处理时间(秒) -Data:当前图像批次的磁盘加载时间(秒) -Loss:模型产生的累积误差(预期误差和预测误差) **如果你想训练 10 次，批处理的图像为 6，即运行一下命令**

```
python3 train.py --model-dir=models/cat_dog data/cat_dog
--epochs=10 --batch=6
```

5.模型精度讲解

在这个由 5000 幅图像组成的数据集上，在 Jetson orin Nano 上训练 ResNet-18 每个时期大约需要 2 分钟，或者大约 55 分钟来训练模型达到 35 个时期和 80%的分类精度。下图用于分析历元的训练进度与模型准确性的关

系:



在大约第 30 个时期，ResNet-18 模型达到 80%的精确度，在第 65 个时期，它收敛于 82.5%的精确度。有了额外的训练时间，您可以通过增加数据集的大小来进一步提高准确性或者尝试更复杂的模型。 默认情况下，训练脚本设置为运行 35 个时期，但是如果您不希望等待那么长时间来测试您的模型，您可以提前退出训练并继续下一步(可以选择稍后从您停止的地方重新开始训练)。**通过--resume 这个参数进行恢复** 我们的**附录/模型文件/cat_dog** 中提供了训练 35 个周期的模型和 100 个周期的模型，如果你不想训练，可以自己把模型传到 Nano 下的**你的路径 /jetson-inference/python/training/classification/models/cat_dog** 解压就好，yahboom 的镜像则不需要操作，里面已经存在 100 次的训练后的模型

6.训练完毕后，把模型转成一个 onNano 的模型文件

onNano 的模型文件方便 TensorRT 进行处理

```
python3 onNano_export.py --model-dir=models/cat_dog
```

这将创建一个名为 resnet18.onNano 下面的

jetson-inference/python/training/classification/models/cat_dog/

7.用 TensorRT 处理图像

以 yahboom 镜像的命令为例

```
cd /home/jetson/jetson-inference/python/training/classification
export NET=models/cat_dog
export DATASET=data/cat_dog
imagenet.py --model=$NET/resnet18.onNano --input_blob=input_0
--output_blob=output_0 --labels=$DATASET/labels.txt
$DATASET/test/cat/01.jpg cat.jpg
```

以上的命令执行完毕，可以在下面的路径观察的到结果，该命令是处理一张图片 路径:

/home/jetson/jetson-inference/python/training/classification

如果你想要测试所有的图片可以输入一下命令

```
cd /home/jetson/jetson-inference/python/training/classification
export NET=models/cat_dog
export DATASET=data/cat_dog
imagenet --model=$NET/resnet18.onNano --input_blob=input_0
--output_blob=output_0 --labels=$DATASET/../labels.txt \
              $DATASET/test/dog ./data/cat_dog/test_output_dog
```

输出的结果在
/home/jetson/jetson-inference/python/training/classification/data/cat_dog/test_output_dog 路径下

如果你想要打开摄像头进行检测

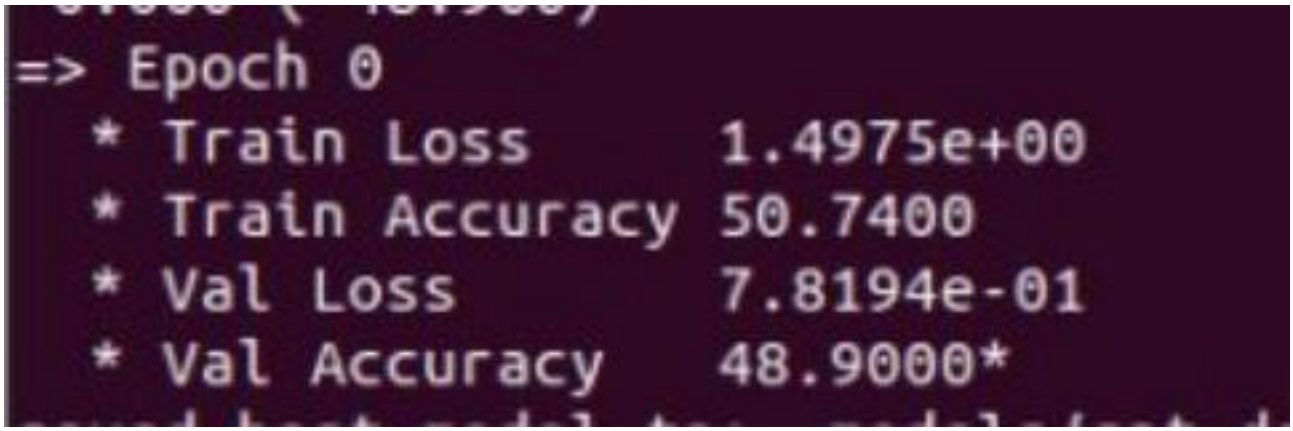
```
imagenet.py --model=$NET/resnet18.onNano --input_blob=input_0
--output_blob=output_0 --labels=$DATASET/labels.txt csi://0      #对
CSI 摄像头

imagenet.py --model=$NET/resnet18.onNano --input_blob=input_0
--output_blob=output_0 --labels=$DATASET/labels.txt
v4l2:///dev/video0      #对 v4l2 摄像头
```

其它视频流文件的打开方式，可参考：

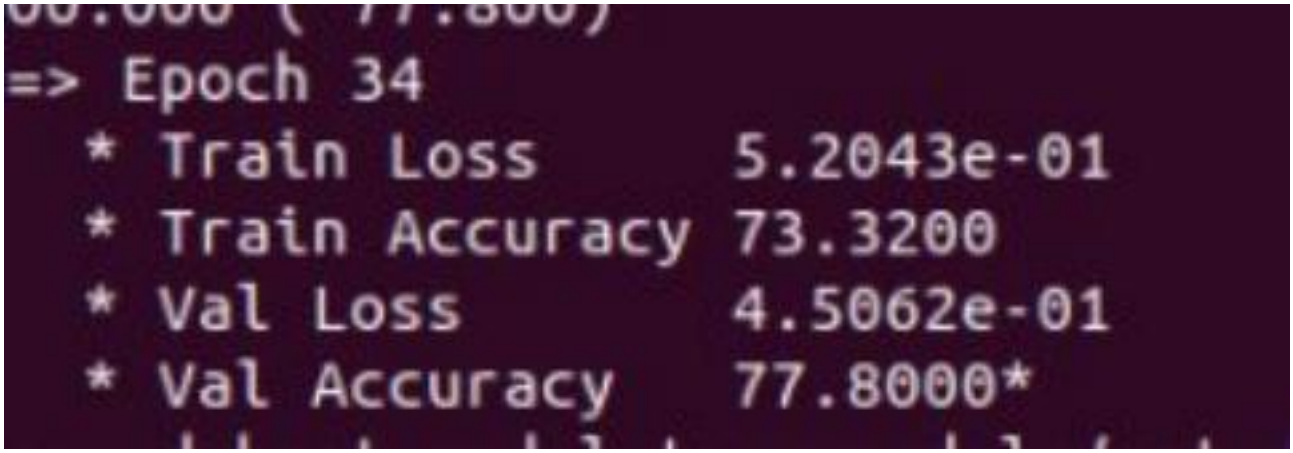
https://github.com/dusty-nv/jetson-inference/blob/master/docs/aux-streaming.md

8. 1 个训练周期、35 个训练周期、100 个训练周期的结果

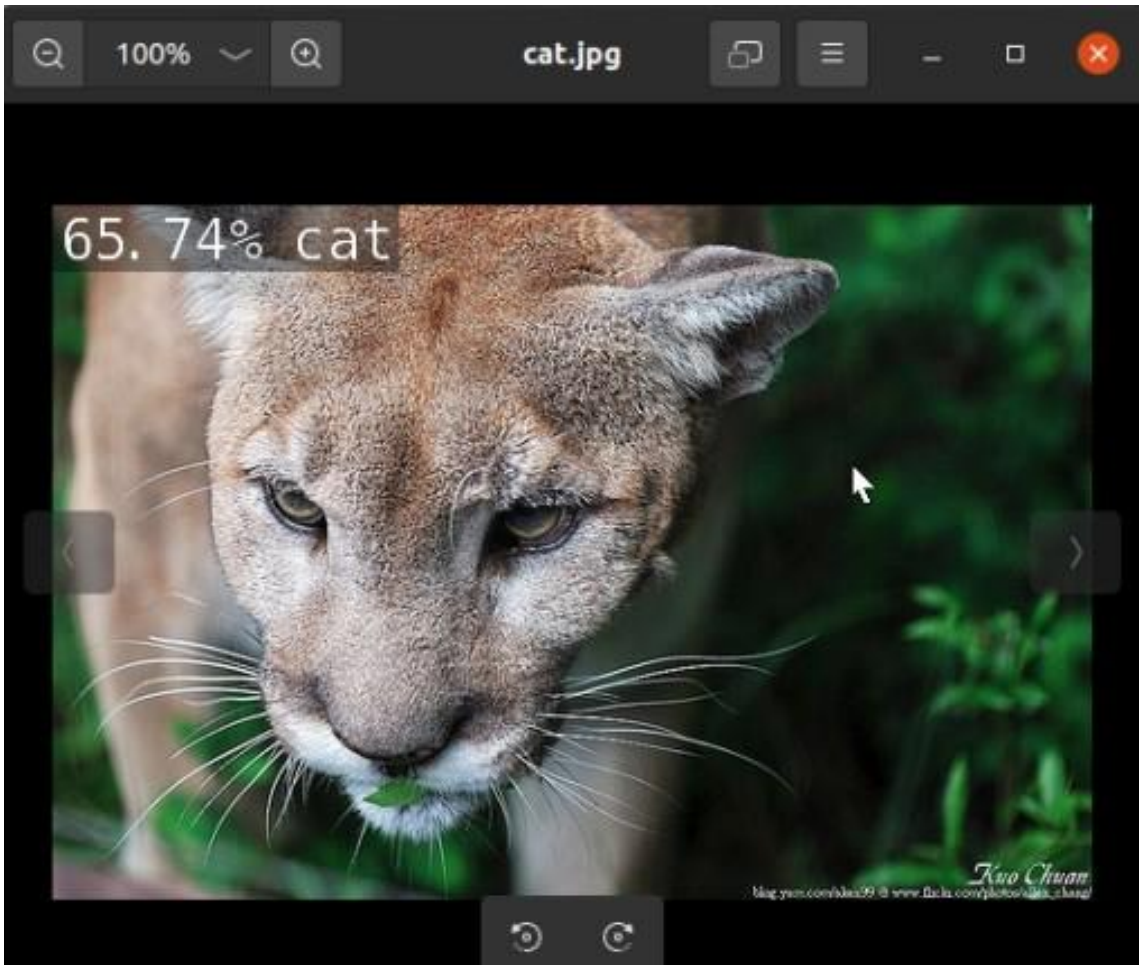
- 1 个训练周期



- 35 个训练周期



- 35 个训练周期识别结果



- 100 个训练周期识别结果

可以看到 100 个训练周期的结果比 35 个训练周期的结果的精度大大提升。

9.附录

如果你想对自己的各种类别不同的图片进行训练 可以参考以下的链接：

https://github.com/dusty-nv/jetson-inference/blob/master/docs/pytorch-collect.md

3. 目标检测推理

目标检测推理

1.使用 DetectNet 定位对象

前面的识别示例输出表示整个输入图像的类概率。接下来，我们将重点关注对象检测，并通过提取边界框来查找帧中各种对象的位置。与图像分类不同，对象检测网络能够每帧检测许多不同的对象。

detectNet 对象接受图像作为输入，并输出检测到的边界框的坐标列表及其类别和置信度值。detectNet 可以从 Python 和 C++中使用。有关可供下载的各种预训练检测模型，请参见下文。使用的默认模型是在 MS COCO 数据集上训练的 91 类 SSD-Mobilenet-v2 模型，该模型在 Jetson 上使用 TensorRT 实现了实时推理性能。

2.从图像中检测对象

首先，让我们尝试使用 detectnet 程序来定位静态图像中的对象。除了输入/输出路径之外，还有一些额外的命令行选项：

更改正在使用的检测模型的网络标志（默认为 SSD-Mobilenet-v2）（可选）

optional--overlay 标志，可以是以逗号分隔的 box、line、labels、conf 和 none 的组合。

默认值为--overlay=box、labels，conf，它显示框、标签和置信度值。

长方体选项绘制填充的边界框，而直线仅绘制未填充的轮廓

- optional--alpha 值，它设置叠加过程中使用的 alpha 混合值（默认值为 120）。
- 阈值，用于设置检测的最小阈值（默认值为 0.5）。

如果您使用的是 Docker 容器，建议将输出图像保存到 images/test 挂载的目录中。然后，在 jetson 推理/data/images/test 下，可以很容易地从主机设备上查看这些图像.

注:运行案例之前,要是自己搭建环境的情况下得自己下载 resnet-18.tar.gz 模型文件到 network 文件夹下才能运行上面的程序。使用我们提供的镜像就可以直接输入上面的程序。

请确保您的终端位于 aarch64/bin 目录中：

```
cd jetson-inference/build/aarch64/bin
```

以下是使用默认的 SSD-Mobilenet-v2 模型在图像中检测行人的一些示例：

```
# C++
$ ./detectnet --network=ssd-mobilenet-v2 images/xingren.png
images/test/output_xinren.png

# Python
$ ./detectnet.py --network=ssd-mobilenet-v2 images/xingren.png
images/test/output_xinren.png
```

注意：第一次运行每个模型时，TensorRT 将花费几分钟时间来优化网络。 这个经过优化的网络文件随后被缓存到磁盘上，因此将来使用该模型的运行将更快地加载。

3.处理视频文件

可以在 images 文件夹里存放视频，路径为

```
cd jetson-inference/build/aarch64/bin
```

运行程序：

```
# C++
./detectnet pedestrians.mp4 images/test/pedestrians_ssd.mp4

# Python
./detectnet.py pedestrians.mp4 images/test/pedestrians_ssd.mp4
```

效果：

可以使用--threshold 设置来向上或向下更改检测灵敏度（默认值为 0.5）。

4.运行实时摄像头识别演示

我们之前使用的 detectnet.cpp/detectnet.py 样本也可以用于实时相机流。

支持的摄像机类型包括：

- MIPI CSI cameras (csi://0)
- V4L2 cameras (/dev/video0)
- RTP/RTSP streams (rtsp://username:password@ip:port)

以下是在相机订阅源上启动程序的一些典型场景。

C++

```
$ ./detectnet csi://0                   # MIPI CSI camera
$ ./detectnet /dev/video0               # V4L2 camera
$ ./detectnet /dev/video0 output.mp4    # save to video file
```

python

```
$ ./detectnet.py csi://0                # MIPI CSI camera
$ ./detectnet.py /dev/video0            # V4L2 camera
$ ./detectnet.py /dev/video0 output.mp4 # save to video file
```

OpenGL 窗口中显示的是覆盖有检测到的对象的边界框的实时摄像机流。请注意，基于 SSD 的机型目前具有最高的性能。这是一个使用模型的例子：

如果在视频馈送中没有检测到所需的对象，或者您得到了虚假的检测，请尝试使用--threshold 参数降低或增加检测阈值（默认值为 0.5）。

4. 训练对象检测模型

训练对象检测模型

训练对象检测模型 1.训练前的准备 2.对基础模型进行下载 3.进行图片素材的下载 4.开始训练训练参数的配置说明 5.将模型转换为 ONNX6.导入要识别的图片如果想要打开摄像头进行识别可以输入以下命令 7.识别结果

1.训练前的准备

在这个教程开始前，python 必须安装好 **torchCPU+GPU 版本，Yahboom 的镜像是已经安装的了。**如果没安装，请自行安装，需要科学上网，也可以看本系列教程的 torch 的安装教程。

```
cd jetson-inference/build
./install-pytorch.sh
```

因为基础模型已经有一些基础的信息，如果从头训练将会大大增加训练的时间，所以本教程只讲解对基础模型重新训练，如果想从头训练请看上一章的训练教程

2.对基础模型进行下载

```
cd /home/jetson/jetson-inference/python/training/detection/ssd
wget https://nvidia.box.com/shared/static/djf5w54rjvpqocsiztzaandq1m3avr7c.pth -O models/mobilenet-v1-ssd-mp-0_675.pth
pip3 install -v -r requirements.txt
```

如果因为无法科学上网，导致模型下载不了，可以到/附录/模型文件下/fruit 文件夹下找到，并把模型传到

/home/jetson/jetson-inference/python/training/detection/ssd/models 文件下 Yahboom 的镜像则无需操作

3.进行图片素材的下载

```
cd /home/jetson/jetson-inference/python/training/detection/ssd
python3 open_images_downloader.py --class-names "Apple,Orange,Banana,Strawberry,Grape,Pear,Pineapple,Watermelon" --data=data/fruit
```

这条命令是把所有相关的图片都下载下来。 **如果你的系统空间不足或者不想下这么多的图片可以使用一下的方法进行改进** YAHBOOM 的镜像是把全部的相关的图片都下载了

```
python3 open_images_downloader.py --stats-only --class-names "Apple,Orange,Banana,Strawberry,Grape,Pear,Pineapple,Watermelon" --data=data/fruit
```

在实践中，为了减少训练时间(和磁盘空间)，您可能希望保持图像总数< 10K。虽然你用的图像越多，你的模型就越精确。您可以使用限制下载的数据量--max-images 选项或--max-annotations-per-class 选项:

1. --stats-only：只下载对应类的图片，不会下载不必要的图片

2. --max-images 将总数据集限制为指定数量的图像，同时保持每个类的图像分布与原始数据集大致相同。如果一个类比另一个类有更多的图像，比例将大致保持不变。
3. --max-annotations-per-class 将每个类限制在指定数量的边界框内，如果一个类的可用边界框少于该数量，将使用它的所有数据——如果数据在类间的分布不平衡，这将非常有用。

例如，如果您只想对水果数据集使用 2500 张图像，您可以像这样启动下载程序:

```
python3 open_images_downloader.py --max-images=2500 --class-names "Apple,Orange,Banana,Strawberry,Grape,Pear,Pineapple,Watermelon" --data=data/fruit
```

如果--max-boxes 选项或--max-annotations-per-class 未设置，默认情况下，所有可用的数据都将被下载——因此，在此之前，一定要先用 --stats-only。

4.开始训练

```
cd /home/jetson/jetson-inference/python/training/detection/ssd
```

```
python3 train_ssd.py --data=data/fruit --model-dir=models/fruit
--batch-size=4 --epochs=30
```

注意:如果内存不足或者您的流程在培训过程中被“终止”,请尝试安装交换和禁用桌面 GUI.

```
#禁用桌面 GUI
sudo init 3 # stop the desktop
sudo init 5 # restart the desktop
```

交换虚拟内存（参考链接）：

https://github.com/dusty-nv/jetson-inference/blob/master/docs/pytorch-transfer-learning.md#mounting-swap

训练参数的配置说明

配置选项	默认值	描 述
--data	data/	数据集的位置
--model-dir	models/	输出已训练模型检查点的目录
--resume	none	要从中恢复训练的现有检查

配置选项	默认值	描 述
		点的路径
--batch-size	4	根据可用内存尝试增加内存
--epochs	30	达到 100 是可取的，但会增加培训时间
--workers	2	数据加载器线程数(0 =禁用多线程)

如果您想在全部纪元完成训练之前测试您的模型，您可以按 Ctrl+C 来终止训练脚本，并在以后使用--resume=论点 我们的附录/模型文件/fruit 中提供了训练 30 个周期的模型和 100 个周期的模型，如果你不想训练，可 以自己把模型传到 NX 下的你的路径 /jetson-inference/python/training/detection/ssd/models/fruit 解压就好，yahboom 的镜像则不需要操作，里面已经存在 100 次的训练后的模型

5.将模型转换为 ONNX

接下来，我们需要将训练好的模型从 PyTorch 转换为 ONNX，这样我们就可以用 TensorRT 加载它:

```
python3 onnx_export.py --model-dir=models/fruit
```

这将保存一个名为 ssd-mobilenet.onnx 下面的 jetson-inference/python/training/detection/ssd/models/fruit/

6.导入要识别的图片

如果自己有相关的图片，可以把相关的图片上传到 NX 进行识别。没有可以到附录/模型文件/fruit 下上传 960 张水果的图片到 nx 的 /home/jetson/jetson-inference/python/training/detection/ssd/data 上，即（images.zip）然后解压。

```
export IMAGES=/home/jetson/jetson-inference/python/training/detection/ssd/data/images
cd $IMAGES && mkdir test && cd ../..
detectnet --model=models/fruit/ssd-mobilenet.onnx --labels=models/fruit/labels.txt \
          --input-blob=input_0 --output-cvg=scores --output-bbox=boxes \
            "$IMAGES/fruit_*.jpg" $IMAGES/test/fruit_%i.jpg
```

输出的结果在

/home/jetson/jetson-inference/python/training/detection/ssd/data/test 文件路径下 **如果使用的是自己的图片，想要运行上述命令，要把图片名改成 fruit_*.jpg，图片必须是 jpg 格式的，否则自行更改运行命令，本教程不在阐述了**

如果想要打开摄像头进行识别可以输入以下命令

```
detectnet --model=models/fruit/ssd-mobilenet.onnx
--labels=models/fruit/labels.txt \
            --input-blob=input_0 --output-cvg=scores
--output-bbox=boxes \
                csi://0       #csi 摄像头

detectnet --model=models/fruit/ssd-mobilenet.onnx
--labels=models/fruit/labels.txt \
            --input-blob=input_0 --output-cvg=scores
--output-bbox=boxes \
                v4l2:///dev/video0      #对 v4l2 摄像头
```

其它视频流文件的打开方式，可参考：

https://github.com/dusty-nv/jetson-inference/blob/master/docs/aux-streaming.md

7.识别结果

YAHBOOM 的镜像中存在了用训练 30 次的模型和训练 100 次的模型的识别结果 30 次的结果：/home/jetson/jetson-inference/python/training/detection/ssd/data/test_30 100 次的结果:/home/jetson/jetson-inference/python/training/detection/ssd/data/test_100 本教程随机选取图片进行对比 结果的如图所示：

可见，训练次数越小，精度不高，很可能导致图像无法识别的原因出现。

5. 语义分割

语义分割

1.语义分割介绍

我们将在本教程中介绍的下一个深度学习功能是语义分割。语义分割是基于图像识别的，除了分类发生在像素级别，而不是整个图像。这是通过对预先训练的图像识别主干进行卷积来实现的，该主干将模型转换为能够按像素标记的全卷积网络（FCN）。分割对于环境感知特别有用，它为每个场景产生了许多不同潜在对象的密集的每像素分类，包括场景前景和背景。

segNet 接受 2D 图像作为输入，并输出具有每像素分类掩模覆盖的第二图像。遮罩的每个像素对应于已分类的对象类别。segNet 可以从 Python 和 C++ 中使用。

下载其他得模型

在 Jetson 上具有实时性能的 FCN-ResNet18 网络的各种预训练分割模型。下面是可供使用的预训练语义分割模型的表，以及用于加载它们的 segnet 的相关--network 参数。它们基于 21 类 FCN-ResNet18 网络，使用 PyTorch 在各种数据集和分辨率上进行了训练，并导出为 ONNano 格式以加载 TensorRT。

Dataset	Resolution	CLI Argument	Accuracy	Jetson Nano	Jetson Xavier
Cityscapes	512x256	`fcn-resnet18-cityscapes-512x256`	83.3%	48 FPS	480 FPS
Cityscapes	1024x512	`fcn-resnet18-cityscapes-1024x512`	87.3%	12 FPS	175 FPS
Cityscapes	2048x1024	`fcn-resnet18-cityscapes-2048x1024`	89.6%	3 FPS	47 FPS
DeepScene	576x320	`fcn-resnet18-deepscene-576x320`	96.4%	26 FPS	360 FPS
DeepScene	864x480	`fcn-resnet18-deepscene-864x480`	96.9%	14 FPS	190 FPS
Multi-Human	512x320	`fcn-resnet18-mhp-512x320`	86.5%	34 FPS	370 FPS
Multi-Human	640x360	`fcn-resnet18-mhp-640x360`	87.1%	23 FPS	325 FPS
Pascal VOC	320x320	`fcn-resnet18-voc-320x320`	85.9%	45 FPS	508 FPS
Pascal VOC	512x320	`fcn-resnet18-voc-512x320`	88.5%	34 FPS	375 FPS
SUN RGB-D	512x400	`fcn-resnet18-sun-512x400`	64.3%	28 FPS	340 FPS
SUN RGB-D	640x512	`fcn-resnet18-sun-640x512`	65.1%	17 FPS	224 FPS

这里大家可以根据自己想要得模型来下载网址是

https://github.com/dusty-nv/jetson-inference

2.图片语义分割

以下是一个使用城市景观模型分割城市街道场景的示例：

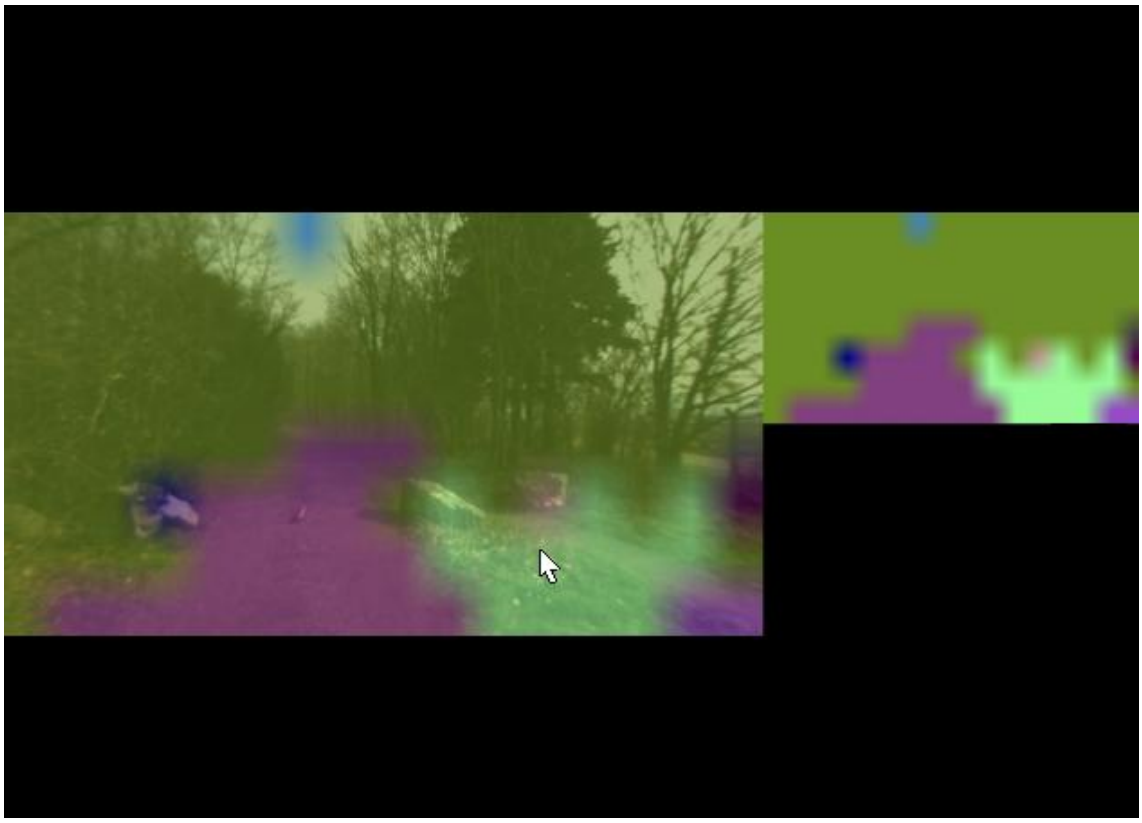
构建项目后，请确保您的终端位于 aarch64/bin 目录中：

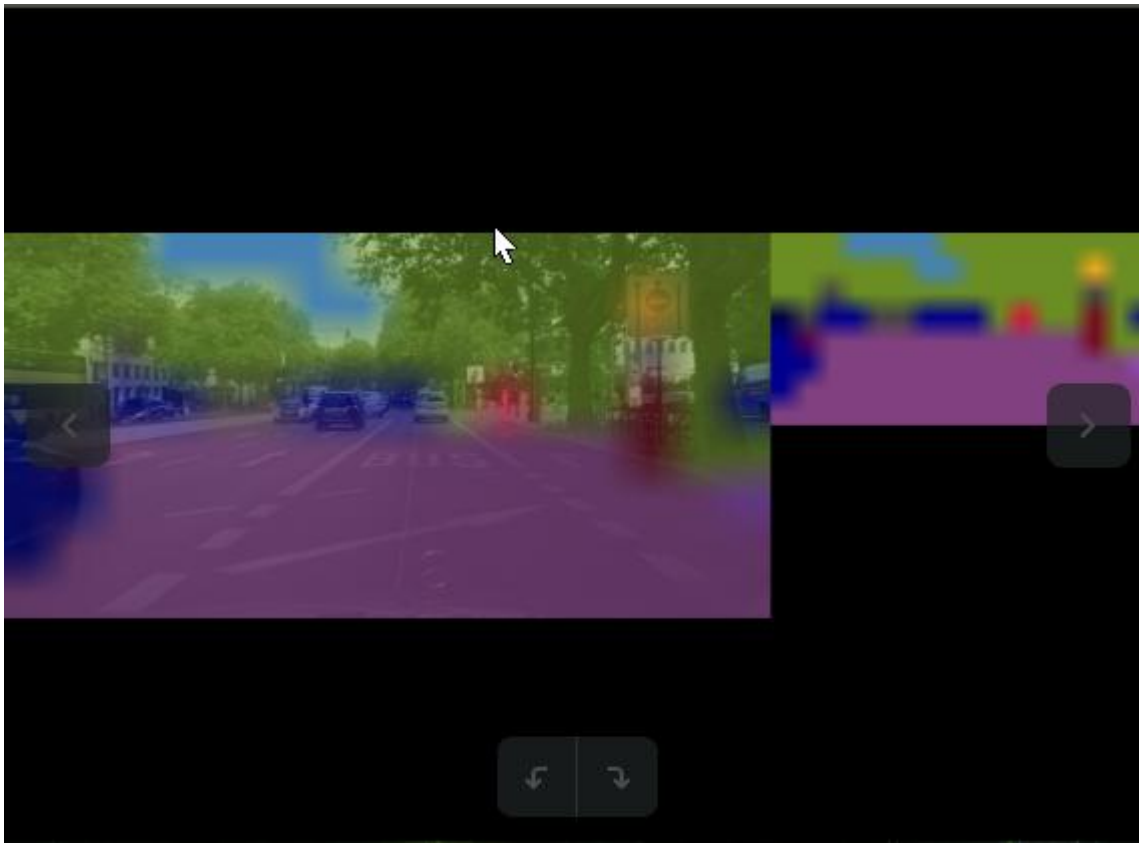
```
cd jetson-inference/build/aarch64/bin
```

下面是使用 fcn-resnet18-cityscapes 模型的一些示例：

--network= 这里面可以放自己下载的模型文件进去。例如我这里的是 fcn-resnet18-cityscapes

```
# C++
$ ./segnet --network=fcn-resnet18-cityscapes images/city_0.jpg images/test/output.jpg

# Python
$ ./segnet.py --network=fcn-resnet18-cityscapes images/city_0.jpg images/test/output.jpg
```





下面例子是 DeepScene 数据集由越野森林小径和植被组成，有助于户外机器人的路径跟踪。

C++

```
$ ./segnet --network=fcn-resnet18-deepscene images/trail_0.jpg images/test/output_overlay.jpg                    # overlay
$ ./segnet --network=fcn-resnet18-deepscene --visualize=mask images/trail_0.jpg images/test/output_mask.jpg    # mask
```

python

```
$ ./segnet.py --network=fcn-resnet18-deepscene images/trail_0.jpg images/test/output_overlay.jpg                    # overlay
$ ./segnet.py --network=fcn-resnet18-deepscene --visualize=mask images/trail_0.jpg images/test/output_mask.jpg # mask
```

注意：要是自己搭建环境的情况下得自己下载模型文件到 network 文件夹下才能运行上面的程序。使用我们提供的镜像就可以直接输入上面的程序

3.运行实时摄像机分割演示

我们之前使用的 segnet.cpp / segnet.py 样本也可以用于实时相机流。支持的摄像机类型包括：

MIPI CSI cameras (csi://0)

V4L2 cameras (/dev/video0)

RTP/RTSP streams (rtsp://username:password@ip:port)

以下是启动该程序的一些典型场景-有关可用的模型

C++

```
$ ./segnet --network=<model> csi://0                    # MIPI CSI
camera
$ ./segnet --network=<model> /dev/video0                # V4L2
camera
$ ./segnet --network=<model> /dev/video0 output.mp4     # save
to video file
```

python

```
$ ./segnet.py --network=<model> csi://0                 # MIPI CSI
camera
$ ./segnet.py --network=<model> /dev/video0             # V4L2
camera
$ ./segnet.py --network=<model> /dev/video0 output.mp4  # save
to video file
```

其中 model 是我们可以选择的我这里使用的模型是 fcn-resnet18-mhp。

OpenGL 窗口中显示的是叠加了分割输出的实时摄像机流，以及为了清晰起见的实心分割遮罩。以下是一些与不同型号一起使用的示例，可供尝试：

```
# C++
$ ./segnet --network=fcn-resnet18-deepscene csi://0

# Python
$ ./segnet.py --network=fcn-resnet18-deepscene csi://0
```

6. 动作识别

动作识别

1.介绍

动作识别对视频帧序列中发生的活动、行为或手势进行分类。DNN 通常使用具有添加的时间维度的图像分类主干。例如，基于 ResNet18 的预训练模型使用 16 帧的窗口。也可以跳过帧来延长模型对动作进行分类的时间窗口。

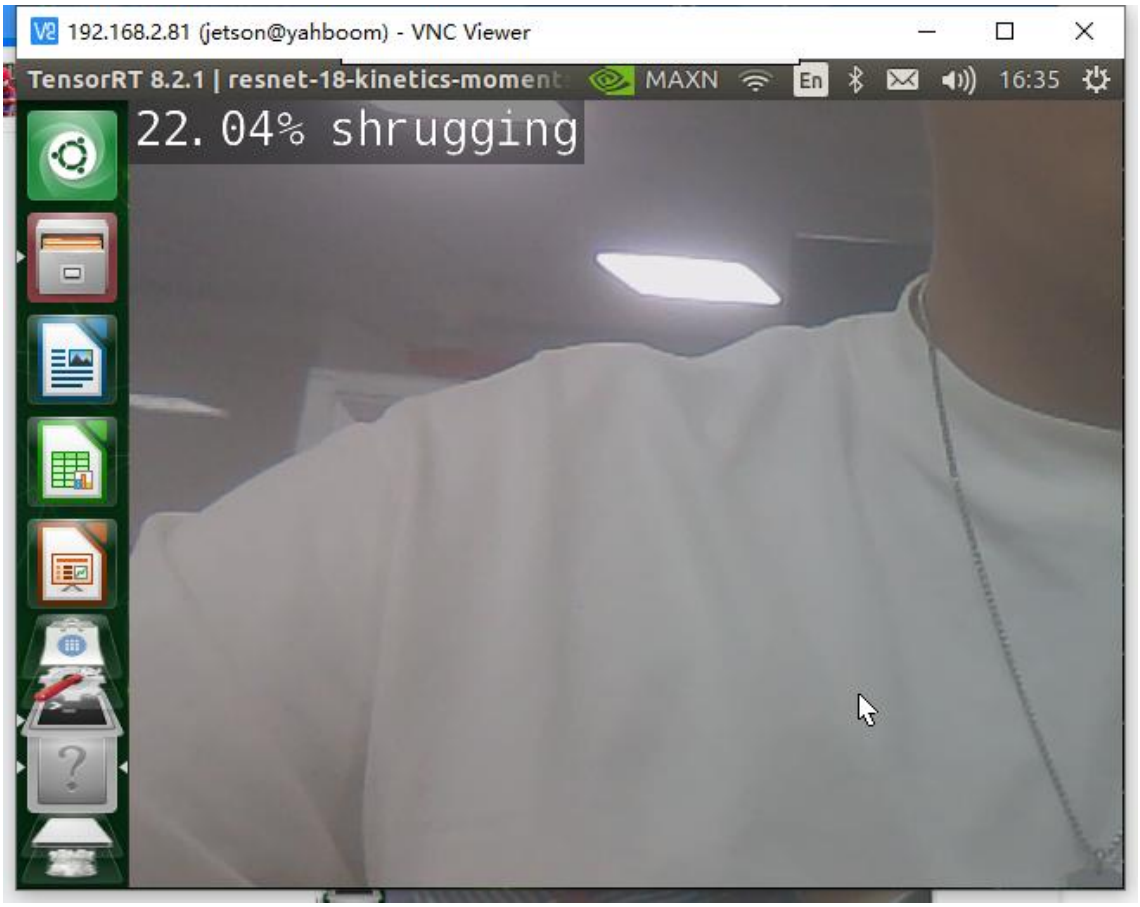
actionNet 模型对象一次接收一个视频帧，将它们作为模型的输入进行缓冲，并以最高置信度输出类。actionNet 可以从 Python 和 C++中使用。作为使用 actionNet 类的示例，镜像有 C++和 Python 的示例程序。

2.运行示例

要在实时摄像机流或视频上运行动作识别，请从输入设备或文件路径。

```
# C++
$ ./actionnet /dev/video0              # V4L2 camera input, display output (default)
```

```
$ ./actionnet input.mp4 output.mp4   # video file input/output (mp4, mkv, avi, flv)

# Python
$ ./actionnet.py /dev/video0              # V4L2 camera input, display output (default)
$ ./actionnet.py input.mp4 output.mp4   # video file input/output (mp4, mkv, avi, flv)
```



默认情况下，模型将每隔一帧处理一次，以延长对操作进行分类的时间窗口。

注意：要是自己搭建环境的情况下得自己下载模型文件到 network 文件夹下才能运行上面的程序。使用我们提供的镜像就可以直接输入上面的程序

以下是可用的预先训练的动作识别模型，以及用于加载它们的 actionnet 的相关--network 参数

Model	CLI argument	Classes
Action-ResNet18-Kinetics	`resnet18`	1040
Action-ResNet34-Kinetics	`resnet34`	1040

默认值为 resnet18。这些模型是在 Kinetics 700 和 Moments in Time 数据集上训练的（类别标签列表请参见此处）。

7. 姿态估计

姿态估计

1.简介

姿势估计包括定位形成骨架拓扑（又名链接）的各种身体部位（又名关键点）。姿势估计有多种应用，包括手势、AR/VR、HMI（人机界面）和姿势/步态校正。为人体和手部姿势估计提供了预先训练的模型，该模型能够在每帧中检测多个人。

poseNet 对象接受图像作为输入，并输出对象姿势列表。每个对象姿势都包含一个检测到的关键点列表，以及它们的位置和关键点之间的链接。您可以查询这些以查找特定功能。poseNet 可以从 Python 和 C++中使用。

作为使用 poseNet 类的示例，有 C++和 Python 的示例程序。这些样本能够检测图像、视频和摄像头中多个人的姿势。有关支持的各种类型的输入/输出流的更多信息。

2.图像的姿态估计

首先，让我们尝试在一些示例图像上运行 posenet 示例。建议将输出图像保存到 images/test 挂载的目录中。然后，在 jetson 推理/data/images/test 下，可以很容易地从主机设备上查看这些图像

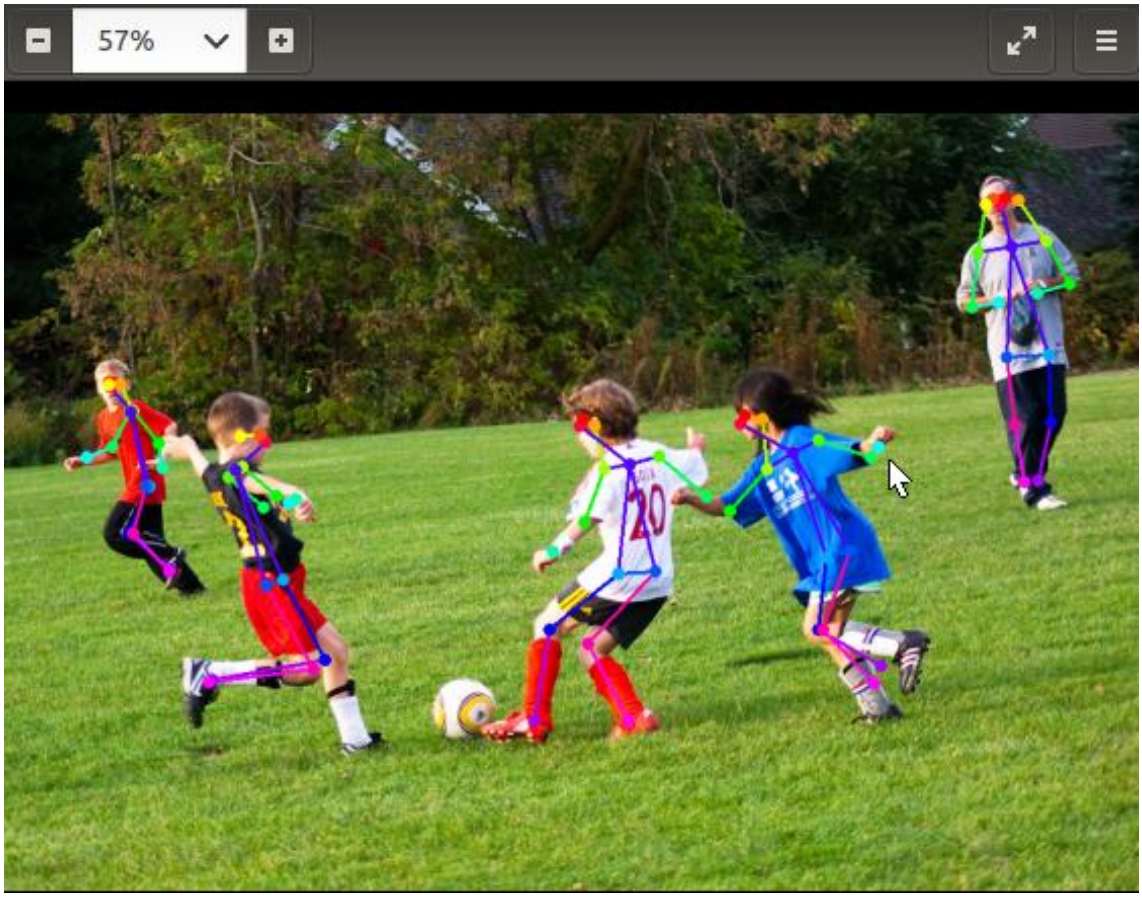
构建项目后，请确保您的终端位于 aarch64/bin 目录中：

```
cd jetson-inference/build/aarch64/bin
```

以下是使用默认 pose-ResNet18-Body 模型进行人体姿势估计的一些示例：

```
# C++
$ ./posenet images/humans_4.jpg images/test/pose_humans_%i.jpg

# Python
```

```
$ ./posenet.py images/humans_4.jpg
images/test/pose_humans_%i.jpg
```

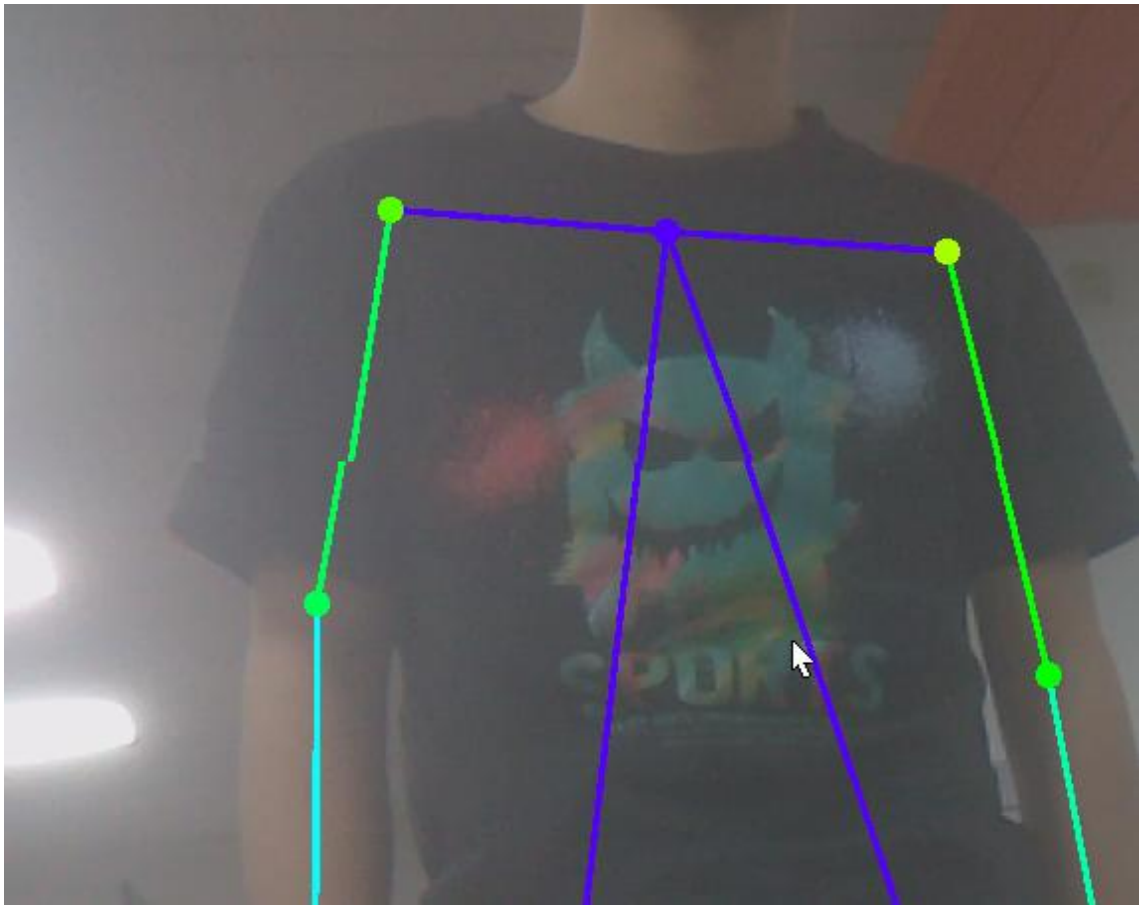


注意：第一次运行每个模型时，TensorRT 将花费几分钟时间来优化网络。 这个经过优化的网络文件随后被缓存到磁盘上，因此将来使用该模型的运行将更快地加载。

3.视频姿态估计

要在实时摄像机流或视频上运行姿势估计，请从“摄像机流和多媒体”页面输入设备或文件路径。

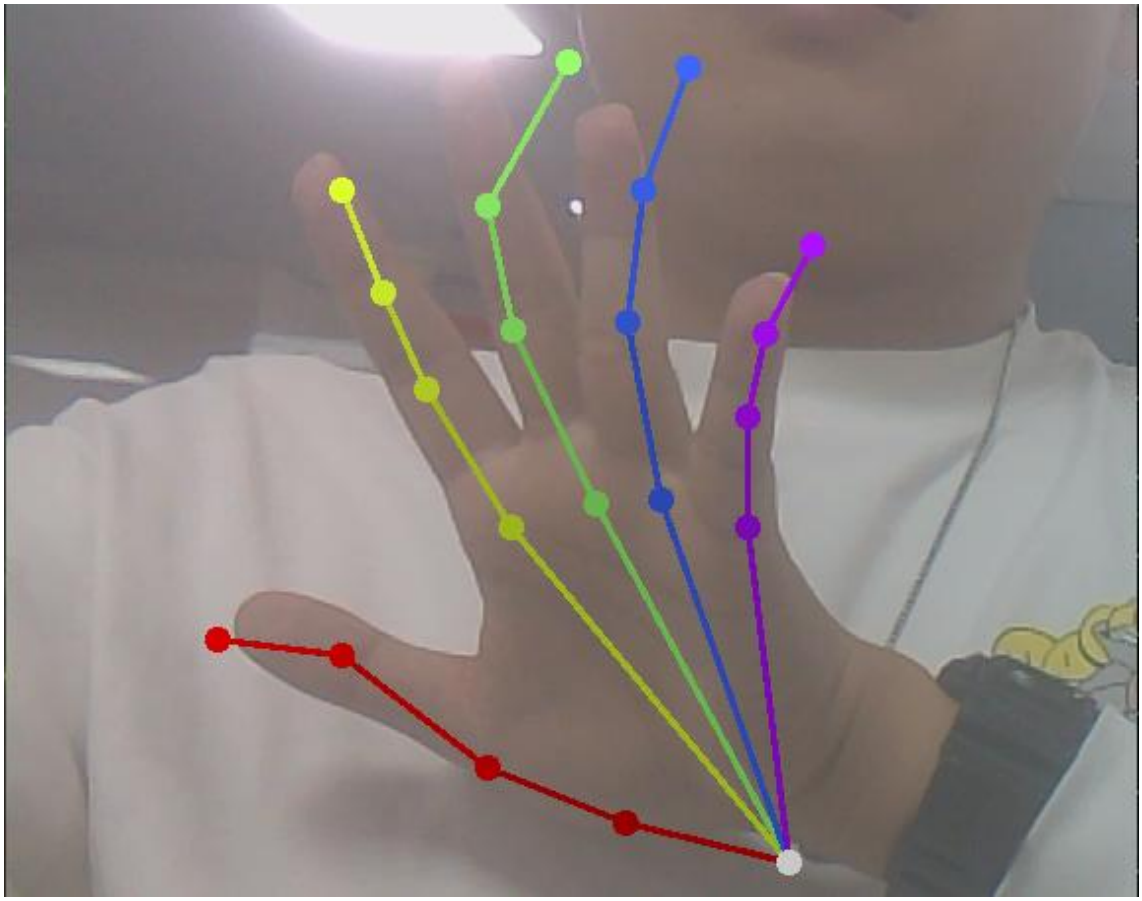
```
# C++
$ ./posenet /dev/video0       # csi://0 if using MIPI CSI camera

# Python
$ ./posenet.py /dev/video0    # csi://0 if using MIPI CSI camera
```



还有个手部识别：

```
# C++
$ ./posenet --network=resnet18-hand /dev/video0
```

```
# Python
$ ./posenet.py --network=resnet18-hand /dev/video0
```



注意：要是自己搭建环境的情况下得自己下载模型文件到 network 文件夹下才能运行上 面的程序。使用我们提供的镜像就可以直接输入上面的程序

8. 背景去除

背景去除

1.简介

背景去除（又称背景减法，或显著对象检测）生成一个遮罩，将图像的前景与背景分割开来。您可以使用它来替换或模糊背景（类似于视频会议应用程序），也可以帮助预处理其他视觉 DNN，如对象检测/跟踪或运动检测。所使用的模型是一个完全卷积网络 U²-Net。

backgroundNet 对象获取图像，并输出前景遮罩。backgroundNet 可以从 Python 和 C++中使用。 作为使用 backgroundNet 类的示例，有 C++和 Python 的示例程序：

2.运行示例

以下是删除和替换图像背景的示例：

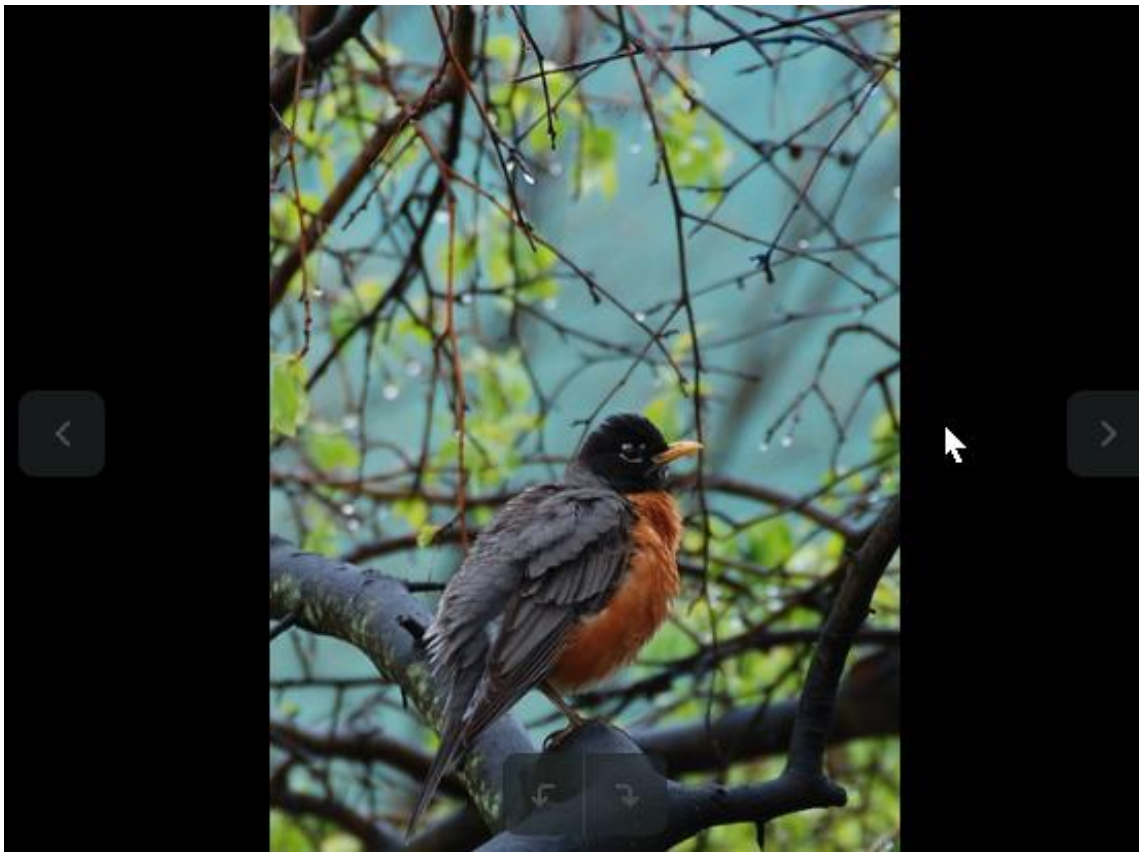
构建项目后，请确保您的终端位于 aarch64/bin 目录中：

```
cd jetson-inference/build/aarch64/bin
```

```
# C++
```

```
$ ./backgroundnet images/bird_0.jpg
images/test/bird_mask.png                    # remove the
background (with alpha)
$ ./backgroundnet --replace=images/snow.jpg images/bird_0.jpg
images/test/bird_replace.jpg      # replace the background

# Python
$ ./backgroundnet.py images/bird_0.jpg
images/test/bird_mask.png                    # remove the
background (with alpha)
$ ./backgroundnet.py --replace=images/snow.jpg images/bird_0.jpg
images/test/bird_replace.jpg # replace the background
```

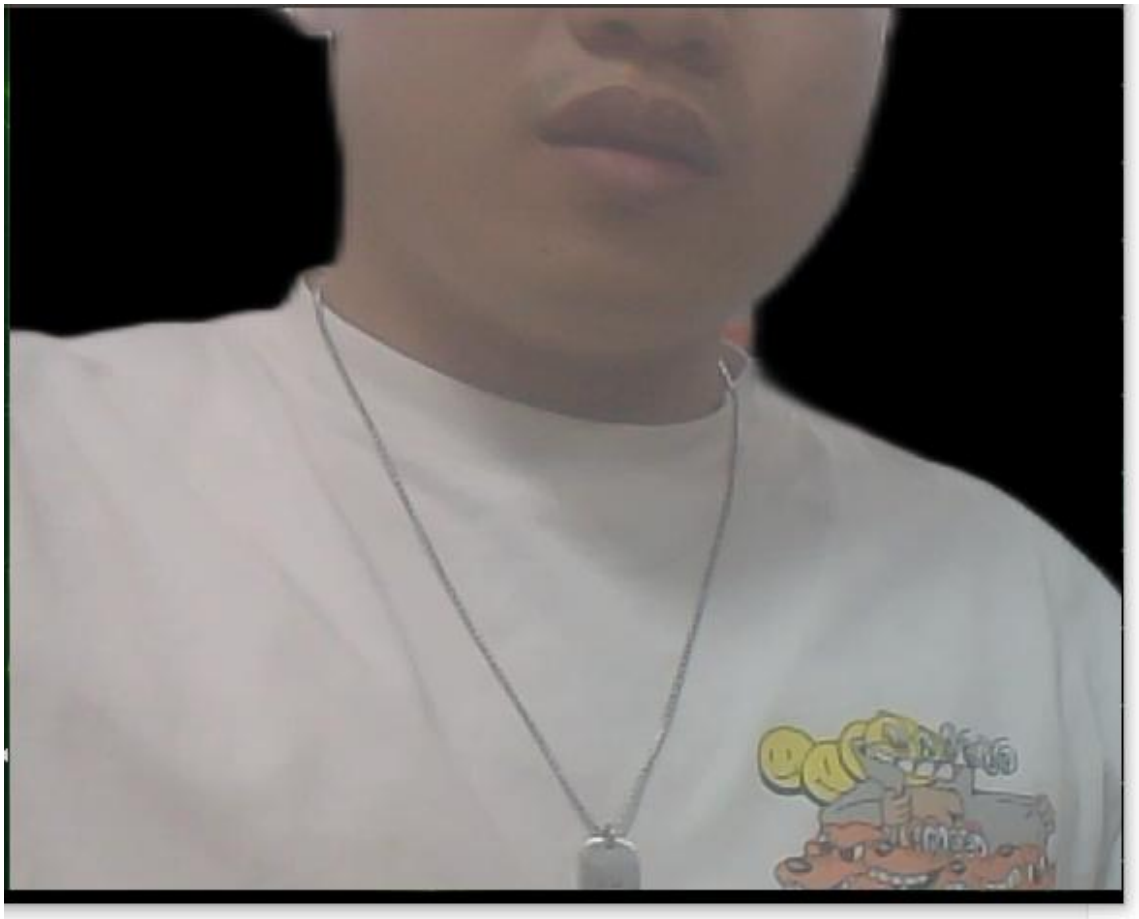
--replace 命令行参数接受用于替换背景的图像的文件名。它将被重新缩放到与输入相同的分辨率



3.实时视频

要在实时摄像机流上运行背景删除或替换，请从“摄像机流和多媒体”页面传入设备：

```
# C++
$ ./backgroundnet /dev/video0                                    # remove the background
$ ./backgroundnet --replace=images/coral.jpg /dev/video0   # replace the background

# Python
$ ./backgroundnet /dev/video0                                    # remove the background
$ ./backgroundnet --replace=images/coral.jpg /dev/video0   # replace the background
```



通过指定输出流，您可以在显示器上（默认设置）、通过网络（与 WebRTC 类似）查看，或将其保存到视频文件中

9. 单眼深度估计

单眼深度估计

1.介绍

深度传感对于地图绘制、导航和障碍物检测等任务很有用，但历史上它需要立体相机或 RGB-D 相机。现在有 DNN 能够从单个单眼图像推断相对深度（又

称单眼深度）。请参阅麻省理工学院的 FastDepth 论文，了解使用全卷积网络（FCN）实现这一目标的一种方法。

depthNet 对象接受单色图像作为输入，并输出深度图。深度图被着色以进行可视化，但原始深度场也可用于直接访问深度。depthNet 可以从 Python 和 C++中使用。作为使用 depthNet 类的示例，我们提供了 C++和 Python 的示例程序：

2.图像上单声道深度

首先，让我们尝试在一些示例图像上运行 depthnet 示例。除了输入/输出路径之外，还有一些额外的命令行选项是可选的：

--network 更改正在使用的深度模型的网络标志

建议将输出图像保存到 images/test 挂载的目录中。然后，在 jetson 推理 /data/images/test 下，可以很容易地从主机设备上查看这些图像

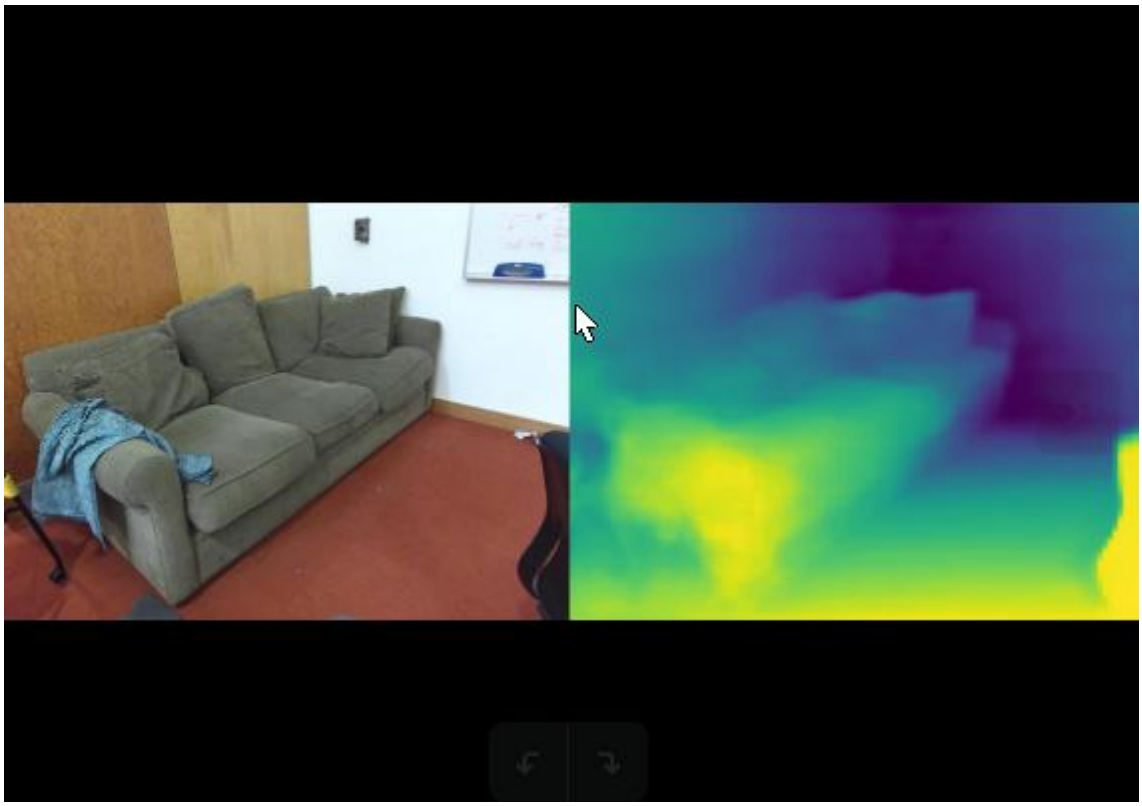
构建项目后，请确保您的终端位于 aarch64/bin 目录中：

```
cd jetson-inference/build/aarch64/bin
```

以下是室内场景单声道深度估计的一些示例：

```
# C++
$ ./depthnet images/room_1.jpg images/test/depth_room_%i.jpg
```

```
# Python
$ ./depthnet.py images/room_1.jpg images/test/depth_room_%i.jpg
```



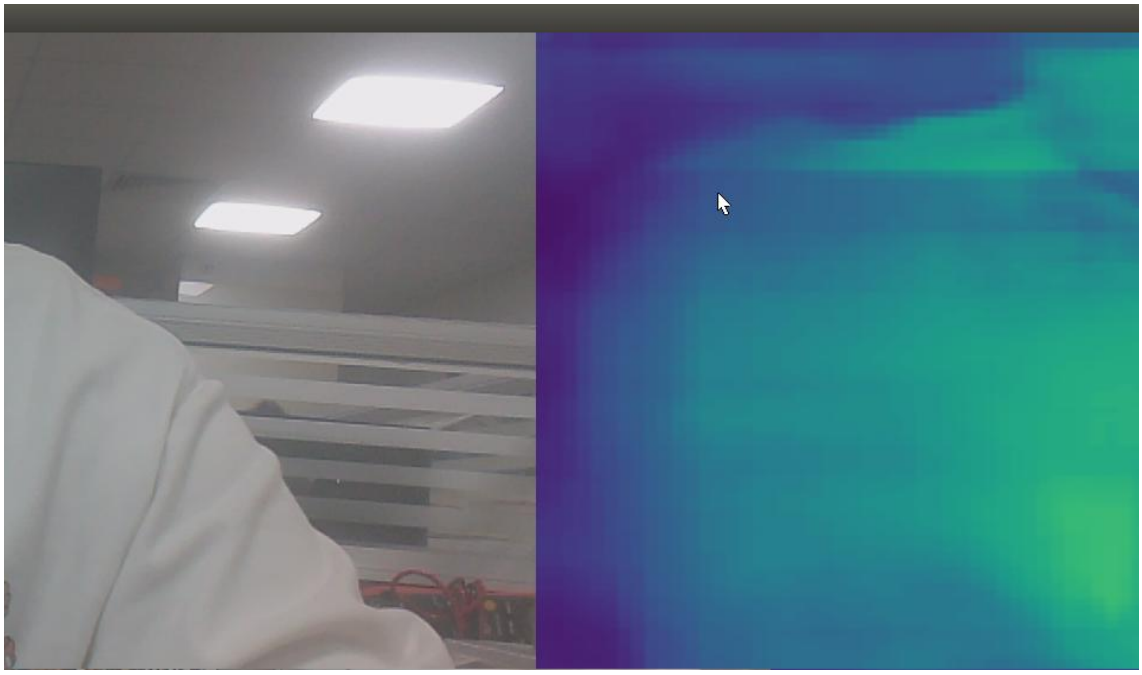
注意：第一次运行每个模型时，TensorRT 将花费几分钟时间来优化网络。 这个经过优化的网络文件随后被缓存到磁盘上，因此将来使用该模型的运行将更快地加载。

3.视频单声道深度

要在实时摄像机流或视频上运行单声道深度估计，请从“摄像机流和多媒体”页面输入设备或文件路径。

```
# C++
```

```
$ ./depthnet /dev/video0       # csi://0 if using MIPI CSI camera

# Python
$ ./depthnet.py /dev/video0    # csi://0 if using MIPI CSI camera
```



注意：如果屏幕太小，无法容纳输出，可以使用--depth scale=0.5 来缩小大小深度图像的大小，或者减小相机的大小，其中--输入宽度=X--输入高度=Y

10. DeepStream 环境搭建(选看)

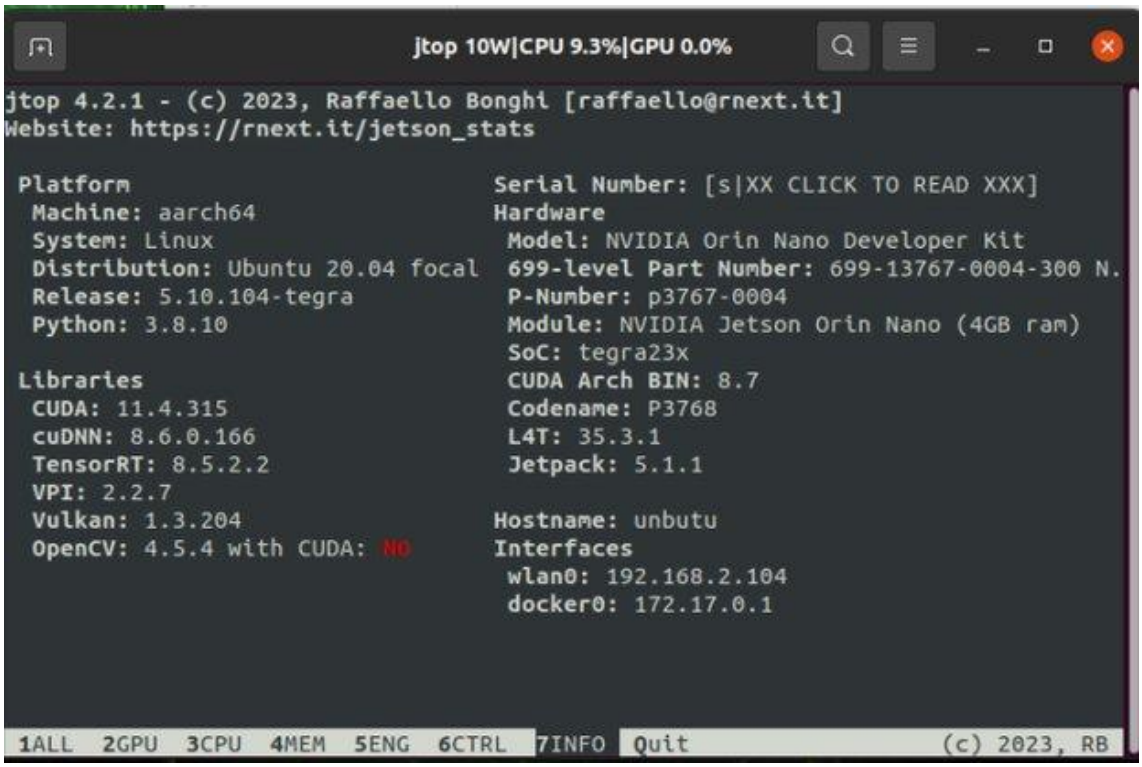
DeepStream 环境搭建

DeepStream 环境搭建 1.搭建前说明 2.本教程的 jetson orin Nano 配置 3. 开始搭建 3.1 下载相关的依赖 3.2 下载 librdkafka 并安装 3.3 安装 Deepsteam4.验证附录

1.搭建前说明

本教程适用于自主搭建的镜像，如果使用的是 YAHBOOM 版本的镜像可以忽略本教程

2.本教程的 jetson orin Nano 配置



经过官网的查询，这个配置能下载 Deepstream6.2 版本的 网址：

https://docs.nvidia.com/metropolis/deepstream/dev-guide/text/DS_Quickstart.html#update-bsp-library

3.开始搭建

3.1 下载相关的依赖

```
sudo apt install \
libssl1.1 \
libgstreamer1.0-0 \
gstreamer1.0-tools \
gstreamer1.0-plugins-good \
gstreamer1.0-plugins-bad \
gstreamer1.0-plugins-ugly \
gstreamer1.0-libav \
libgstreamer-plugins-base1.0-dev \
libgstrtspserver-1.0-0 \
libjansson4 \
libyaml-cpp-dev
```

3.2 下载 librdkafka 并安装

```
git clone https://github.com/edenhill/librdkafka.git
cd librdkafka
git reset --hard 7101c2310341ab3f4675fc565f64f0967e135a6a
```

```
./configure
make -j2
sudo make install
sudo mkdir -p /opt/nvidia/deepstream/deepstream-6.2/lib
sudo cp /usr/local/lib/librdkafka*
/opt/nvidia/deepstream/deepstream-6.2/lib
```

3.3 安装 Deepsteam

通过登录到这里面找到

https://developer.nvidia.com/embedded/deepstream-on-jetson-downloads-archived deepstream_sdk_v6.2.0_jetson.tbz2，并进行下载。 或者是在我们提供的环境搭建的附件找到

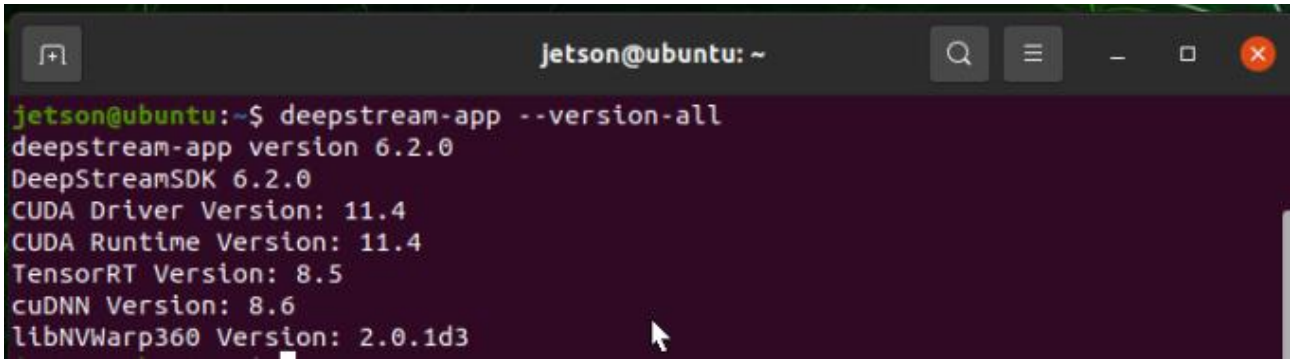
deepstream_sdk_v6.2.0_jetson.tbz2，并传输给 jetson orin Nano.

运行一下命令进行安装

```
sudo tar -xvf deepstream_sdk_v6.2.0_jetson.tbz2 -C /
cd /opt/nvidia/deepstream/deepstream-6.2
sudo ./install.sh
sudo ldconfig
```

4.验证

1. deepstream-app --version-all 来查看安装版本



2. 进入

/opt/nvidia/deepstream/deepstream/samples/configs/deepstream-app 里面跑一下案例

```
cd /opt/nvidia/deepstream/deepstream/samples/configs/deepstream-app
sudo deepstream-app -c source4_1080p_dec_infer-resnet_tracker_sgie_tiled_display_int8.txt
```

等待一段时间，出现视频，即表示 deepstream 安装成功。

附录

其它参考链接： https://zhuanlan.zhihu.com/p/460637017

https://docs.nvidia.com/metropolis/deepstream/dev-guide/text/DS_Quickstart.html#update-bsp-library

11. 汽车检测

汽车检测

1.介绍

该项目包含背靠背检测器应用程序，以展示 Deepstream SDK 的功能。请按照 apps/sample_apps/deepstream-app/README 中的说明安装 deepstream SDK、deepstream SDK 本身和应用程序的必备组件。要是使用我们镜像可以跳过这一步。

2.下载模型

要下载第二个 nvinfer 的模型，请访问

https://github.com/NVIDIA-AI-IOT/redaction_with_deepstream

或者从我们提供的资料附录上获取汽车检测模型传输到 jetson orin nano 上使用以下命令：

```
$ cd /opt/nvidia/deepstream/deepstream/samples/models
$ sudo mkdir Secondary_FaceDetect
$ cd Secondary_FaceDetect
$ sudo wget
https://github.com/NVIDIA-AI-IOT/redaction_with_deepstream/raw/
master/fd_lpd_model/fd_lpd.caffemodel
```

$ sudo wget https://raw.githubusercontent.com/NVIDIA-AI-IOT/redaction_with_deepstream/master/fd_lpd_model/fd_lpd.prototxt

$ sudo wget https://raw.githubusercontent.com/NVIDIA-AI-IOT/redaction_with_deepstream/master/fd_lpd_model/labels.txt

这里要是下载不了，可以到电脑上下载了再传到 jetson orin nano 相对应的路径上。

背靠背检测器应用程序管道：

3.编译步骤和执行

首先是设置 Makefile 的文件里面的 cuda_ver 成我们自己的版本，之后就是运行程序

首先是到路径：

```
$ cd /opt/nvidia/deepstream/deepstream/sources/apps/sample_apps/deepstream_reference_apps/back-to-back-detectors
```

之后运行

```
$ Set CUDA_VER in the MakeFile as per platform.
    For Jetson, CUDA_VER=11.4
    For x86, CUDA_VER=11.8
$ sudo make

$ ./back-to-back-detectors <h264_elementary_stream>
  Ex.: ./back-to-back-detectors ../../../../../samples/streams/sample_720p.h264
```

结果应该如下所示：

注意： 使用 sudo 运行上述命令。 将 secondary_detector_config.txt 中的路径编辑到从上述站点下载的模型的位置。

本文件应描述样品背靠背检测器的应用。此示例建立在 deeptstream-test1 示例的基础上，以演示如何在管道中添加多个背靠背检测器。“nvinfer”元素的两个实例依次添加到 nvstreammux 之后和显示组件之前的管道中。这两个“nvinfer”实例都有自己的配置文件。第一个“nvinfer”实例（人/车/自行车/路标）将始终充当主要检测器。

12. 姿态检测

姿态检测

介绍

该项目包含使用 Deepstream SDK 构建的 3D 身体姿势应用程序。

前提条件

已安装 DeepStream SDK 6.2，可从以下网址获得 http://developer.nvidia.com/deepstream-sdk 请遵循中的说明 /opt/nvidia/deepstream/deepstream/sources/apps/sample_apps/deepstream-app/README 关于如何安装构建 Deepstream SDK 应用程序的先决条件。

搭建

1. 克隆应用程序 /opt/nvidia/deepstream/deepstream/sources/apps/sample_apps/并将项目主页定义为 export BODYPOSE3D_HOME=/deepstream-bodypose-3d.

```
echo "export BODYPOSE3D_HOME=/opt/nvidia/deepstream/deepstream/sources/apps/sample_apps/deepstream_reference_apps/deepstream-bodypose-3d" << ~/.bashrc
source ~/.bashrc
```

1. 安装 NGC CLI 的下载器（yahboom 版是不用安装，如果是自己搭建的镜像可参考下面连接） https://ngc.nvidia.com/setup/installers/cli **如果网络不好，可以从附件里面找到并传输文件到 orin Nano 上，下面需要用到的模型也是可以这样干**

```
sudo mkdir -p $BODYPOSE3D_HOME/models
cd $BODYPOSE3D_HOME/models
# Download PeopleNet
ngc registry model download-version "nvidia/tao/peoplenet:deployable_quantized_v2.5"
# Download BodyPose3DNet
```

```
ngc registry model download-version
"nvidia/tao/bodypose3dnet:deployable_accuracy_v1.0"
sudo apt-get install tree #yahboom 的镜像不需要这步
```

如果模型下载不了，可以通过电脑传输到 home 目录下，再通过 mv 的命令移动到对应的文件夹里 到现在为止，目录树应该看起来像这样

```
tree $BODYPOSE3D_HOME -d
$BODYPOSE3D_HOME
├── configs
├── models
│   ├── bodypose3dnet_vdeployable_accuracy_v1.0
│   └── peoplenet_vdeployable_quantized_v2.5
├── sources
│   ├── deepstream-sdk
│   └── nvdsinfer_custom_impl_BodyPose3DNet
└── streams
```

1. 下载并提取特征值 3.4.0 在项目下。

```
cd $BODYPOSE3D_HOME
sudo wget
https://gitlab.com/libeigen/eigen/-/archive/3.4.0/eigen-3.4.0.tar.gz
```

```
sudo tar xvzf eigen-3.4.0.tar.gz
sudo ln eigen-3.4.0 eigen -s
```

如果下载不了，可以通过电脑传输到 home 目录下，再通过 mv 的命令移动到对应的文件夹里

1. Deepstream SDK 版本 6.2 或更高版本，此步骤不是必需的。

yahboom 的镜像可以不用操作

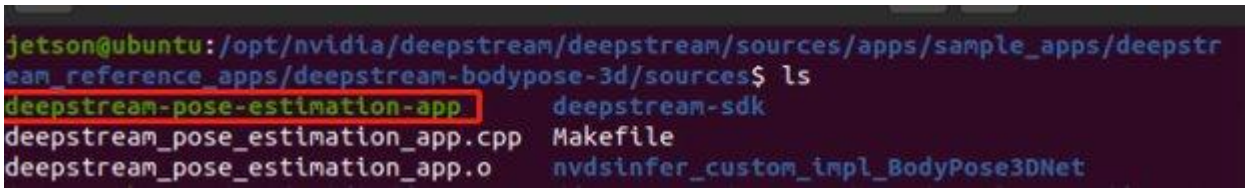
```
# Copy deepstream sources
cp $BODYPOSE3D_HOME/sources/deepstream-sdk/eventmsg_payload.cpp /opt/nvidia/deepstream/deepstream/sources/libs/nvmsgconv/deepstream_schema
# Build new nvmsgconv library for custom Product metadata
cd /opt/nvidia/deepstream/deepstream/sources/libs/nvmsgconv
sudo CUDA_VER=11.4 make && make install
```

5.编译

```
# Build custom nvinfer parser of BodyPose3DNet
```

```
cd $BODYPOSE3D_HOME/sources/nvdsinfer_custom_impl_BodyPose3DNet
sudo CUDA_VER=11.4 make
# Build deepstream-pose-estimation-app
cd $BODYPOSE3D_HOME/sources
sudo CUDA_VER=11.4 make
```

运行成功后可以看到这两个目录下有以下的文件

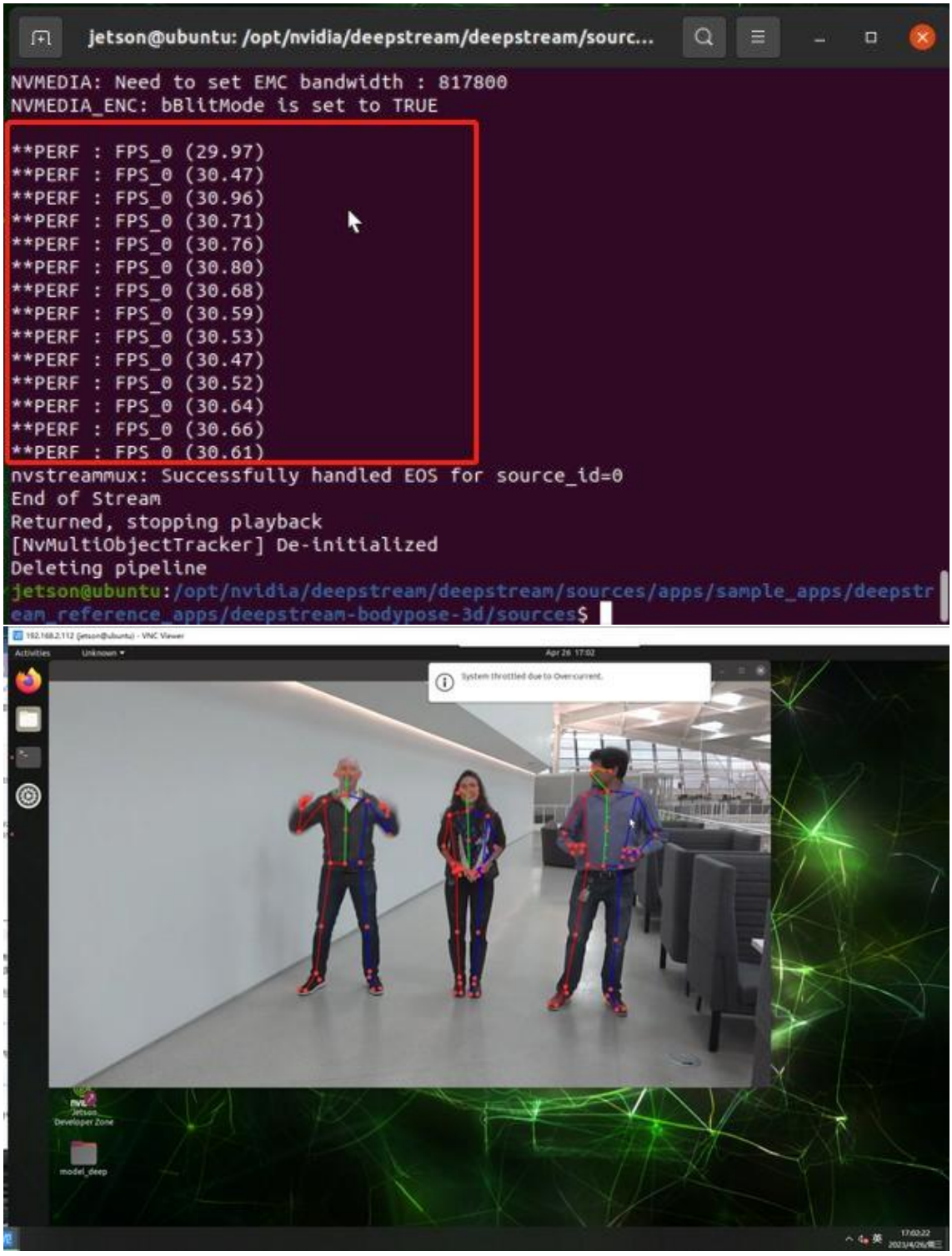


运行程序

1. 以下命令处理 URI 格式的输入视频，并在窗口中呈现叠加的姿态估计。

```
./deepstream-pose-estimation-app --input file://$BODYPOSE3D_HOME/streams/bodypose.mp4
```

注意：请提供源视频文件的绝对路径。 可以看到视频播放的结果，帧率可到达 30 帧



1. 当数据源是视频文件时，下面的命令将带有骨架覆盖的输出视频保存到$BODYPOSE3D_HOME/streams/bodypose_3dbp.mp4 并将骨架的关键点保存到$BODYPOSE3D_HOME/streams/bodypose_3dbp.json

./deepstream-pose-estimation-app --input file://$BODYPOSE3D_HOME/streams/bodypose.mp4 --output $BODYPOSE3D_HOME/streams/bodypose_3dbp.mp4 --focal 800.0 --width 1280 --height 720 --fps --save-pose $BODYPOSE3D_HOME/streams/bodypose_3dbp.json

json 解析：

```
[{
  "num_frames_in_batch": 1,
  "batches": [{
    "batch_id": 0,
    "frame_num": 3,
    "ntp_timestamp": 1639431716322229000,
    "num_obj_meta": 6,
    "objects": [{
      "object_id": 3,
      "pose25d": [707.645203, 338.592499, -0.000448, 0.867188, ...],
      "pose3d": [297.649933, -94.196518, 3520.129883, 0.867188, ...]
    },{
    ...
    }]
```

```
  }]
}, {
```

pose25d 包含 34x4 彩车。四项一组代表一个关键点的[x, y, zRel, conf]价值观。x 和 y 是关键点在图像坐标中的位置；zRel 是骨骼的根关键点(即骨盆)的相对深度值。x, y, zRel 数值以毫米为单位。conf 是预测的置信度值。

pose3d 还包含 34x4 彩车。四项一组代表一个关键点的[x, y, z, conf]价值观。x, y, z 是关键点在以相机为原点的世界坐标中的 3D 位置。x, y, z 数值以毫米为单位。conf 是预测的置信度值。

1. 当数据源是 RTSP 流并且结果被发布到 RTSP 流时

rtsp://localhost:8554/ds-test

```
./deepstream-pose-estimation-app --input rtsp://<ipa_address>:<port>/<topic> --output rtsp://
```

1. 为了将 pose3D 和 pose25D 元数据发布到消息代理，请执行以下操作

```
./deepstream-pose-estimation-app --input file://$BODYPOSE3D_HOME/streams/bodypose.mp4 --conn-str "localhost;9092;test"
```

"localhost;9092;test"是消息代理的连接字符串 localhost，端口号 9092 和主题名称 test。 请在连接字符串两边加上双引号，因为 shell 中的保留字符。

13. yolo5 简介

yolov5 简介

yolov5 简介 1 什么是 YOLO?2.什么是 YOLO V5 附录:

1 什么是 YOLO?

YOLO 是'You only look once'的首字母缩写，是一种将图像划分为网格系统的对象检测的算法。 网格中的每个单元格负责检测物理中心点落在网络自身内部的对象或物体。 由于其速度和准确性，YOLO 是最著名的物体检测算法之一。

2.什么是 YOLO V5

YOLO V5 是由 Ultralytics 公司开源 YOLO 版本，且 完全基于 PyTorch 实现 ，给大量 AI 人员带来了福音。 在我们还对 YOLOv4 的各种骚操作、丰富的实验对比惊叹不已时，YOLOv5 又带来了更强、实时、更精确的目标检测技术。

1. 更快、更实时 按照官方给出的数目，现版本的 YOLOv5 每个图像的 推理时间最快 0.007 秒 ，即每秒 140 帧（FPS） 在 CPU 上，每个图像的推理时间快至 7ms，意味着每秒 140 帧（FPS)！远远大于人眼对 20 帧的要求。相比之下，YOLOv4 在相同的条件下只能到到 50 帧 。而在 GPU 上，FPS 更高，可高达 400.

2.更小巧(内存少）YOLOv5 的权重文件大小只有 YOLOv4 的　1/9　。

3.更短的训练时间 YOLOv5 在单一 V-100 GPU 的情况下，COCO 2017 数据集上的训练时间分别为：

YOLOv5 s	YOLOv5 m	YOLOv5 l	YOLOv
2天	4天	6天	8天

附录：

其它参考教程

https://zhuanlan.zhihu.com/p/172121380

https://blog.csdn.net/qq_26420885/article/details/128658851

14. yolo5 的环境搭建（选看）

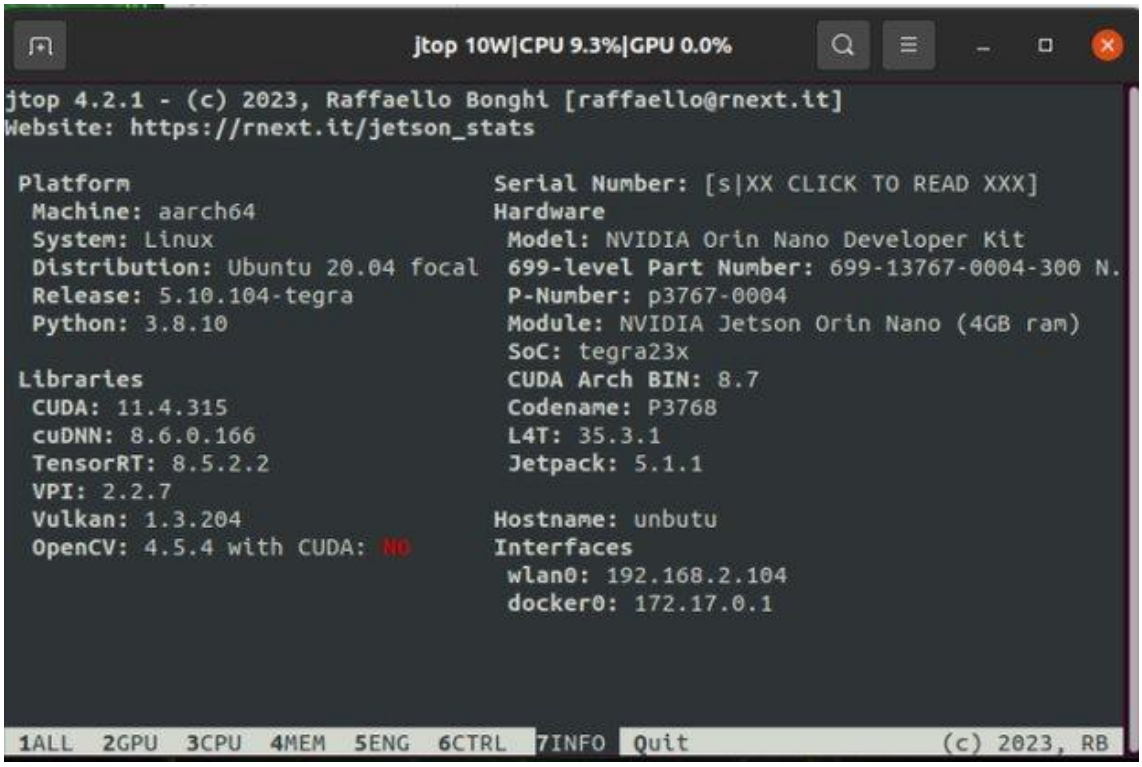
yolo5 的环境搭建

yolo5 的环境搭建本教程适用于 jetson orin Nano 官方镜像自己动手搭建，如果使用的是 YAHBOOM 版的镜像，本教程可以忽略。1.准备工作 2.yolo5 的环境搭建（yolo5 v5.0）2.1 下载 yolo5 的需要的模块-torch2.2 安装对应版本的 torchvision2.3 下载 yolo5 的源码 3.验证 yolo5 是否搭建成功

本教程适用于 jetson orin Nano 官方镜像自己动手搭建，如果使用的是 YAHBOOM 版的镜像，本教程可以忽略。

1.准备工作

jetson orin Nano 一台 jetson orin Nano 的配置如下：



如果看了前面一章的 torch 和 torchvision 的安装教程，可以直接从 2.3 小节开始

2.yolo5 的环境搭建（yolo5 v5.0）

2.1 下载 yolo5 的需要的模块-torch

（如果跟着教程安装了 jetson-inference 的环境这部部分可忽略）

```
sudo apt-get install python3-pip libopenblas-base libopenmpi-dev
pip3 install Cython
pip3 install numpy
torch-1.12.0a0+2c916ef.nv22.3-cp38-cp38-linux_aarch64    # 注意你自己.whl 包路径
sudo apt-get install libjpeg-dev zlib1g-dev libpython3-dev
libavcodec-dev libavformat-dev libswscale-dev
```

torch-1.12.0a0+2c916ef.nv22.3-cp38-cp38-linux_aarch64 这文件从环境搭建的附件获取，通过 winSCP 传输到 jetson 上

2.2 安装对应版本的 torchvision

```
git clone --branch v0.13.0 https://github.com/pytorch/vision
torchvision
cd torchvision
export BUILD_VERSION=0.13.0
python3 setup.py install --user
```

如果 git clone 报错，请检查网络重新运行

2.3 下载 yolo5 的源码

```
cd ~
```

```
git clone https://github.com/marcoslucianops/DeepStream-Yolo
```

```
python3 -m pip install --upgrade pip
```

```
cd yolov5
```

因为 jetson orin Nano 已经自带 opencv4.5.4 了所以不需要在装 python 的 opencv 的，可以通过 import cv2 来进行验证 因此我们需要打开 yolo5 目录下的 requirements 文件，在这行前面添加个#号

```
# pip install -r requirements.txt

# base ----------------------------------------
matplotlib>=3.2.2
numpy>=1.18.5
#opencv-python>=4.1.2
Pillow
PyYAML>=5.3.1
scipy>=1.4.1
torch>=1.7.0
torchvision>=0.8.1
tqdm>=4.41.0

# logging -------------------------------------
tensorboard>=2.4.1
# wandb

# plotting ------------------------------------
seaborn>=0.11.0
pandas

# export --------------------------------------
# coremltools>=4.1
# onnx>=1.8.1
# scikit-learn==0.19.2  # for coreml quantization

# extras --------------------------------------
thop  # FLOPS computation
pycocotools>=2.0  # COCO mAP
```

修改完后运行

```
pip3 install -r requirements.txt -i
```

```
https://mirror.baidu.com/pypi/sample
```

等待下载完即可

3.验证 yolo5 是否搭建成功

```
cd ~/yolov5
python3 detect.py
```

等待他自动下载权重文件，如果网络不行，请从我们提供的**环境搭建的附件**里面获取 yolov5s.pt 文件放到 yolo5 的文件夹下面 如果没有报错，说明 yolo5 搭建成功,并会在 yolov5/runs/detect/exp 路径下存放识别出来的结果

下面是运行成功的图片

exp5:是因为第 5 次运行了 python3 detect.py 这个命令，所以结果存放到了 exp5 的目录下

如果运行第三步可能会出现报错，是因为 torch 的版本太高导致的，因此我们需要进行一些修改

```
sudo gedit /usr/local/lib/python3.8/dist-packages/torch/nn/modules/upsampling.py
```

对 153 行进行一个修改，recompute_scale_factor=self.recompute_scale_factor 这行，在前面添加‘#’即可。 修改后的结果如下图所示：

```
def forward(self, input: Tensor) -> Tensor:
    return F.interpolate(input, self.size, self.scale_factor, self.mode, self.align_corners,
                         #recompute_scale_factor=self.recompute_scale_factor
                         )
```

再运行第 3 步的操作就能正常工作了。

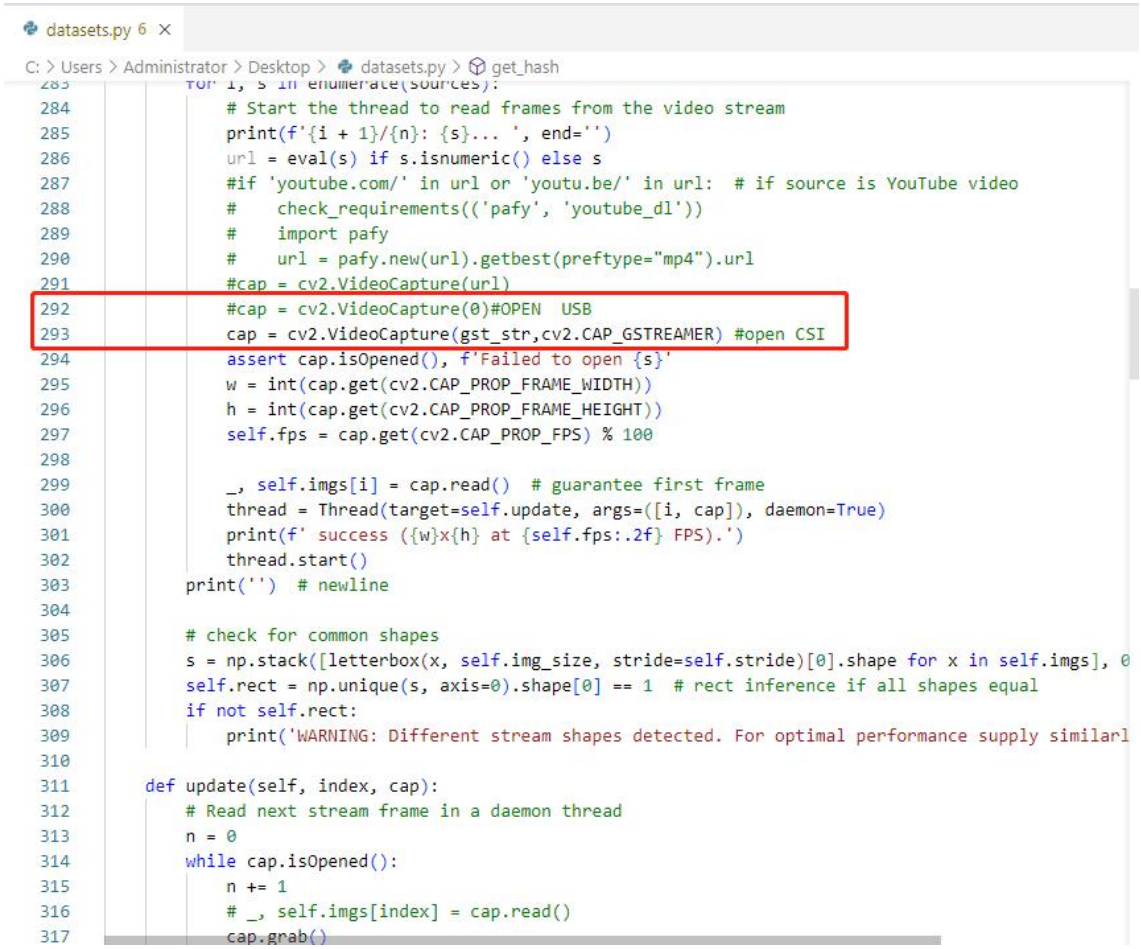
15. yolo5 的实时检测

利用 yolo5 实现摄像头的实时检测

利用 yolo5 实现摄像头的实时检测 1.使用方法 2.注意事项

1.使用方法

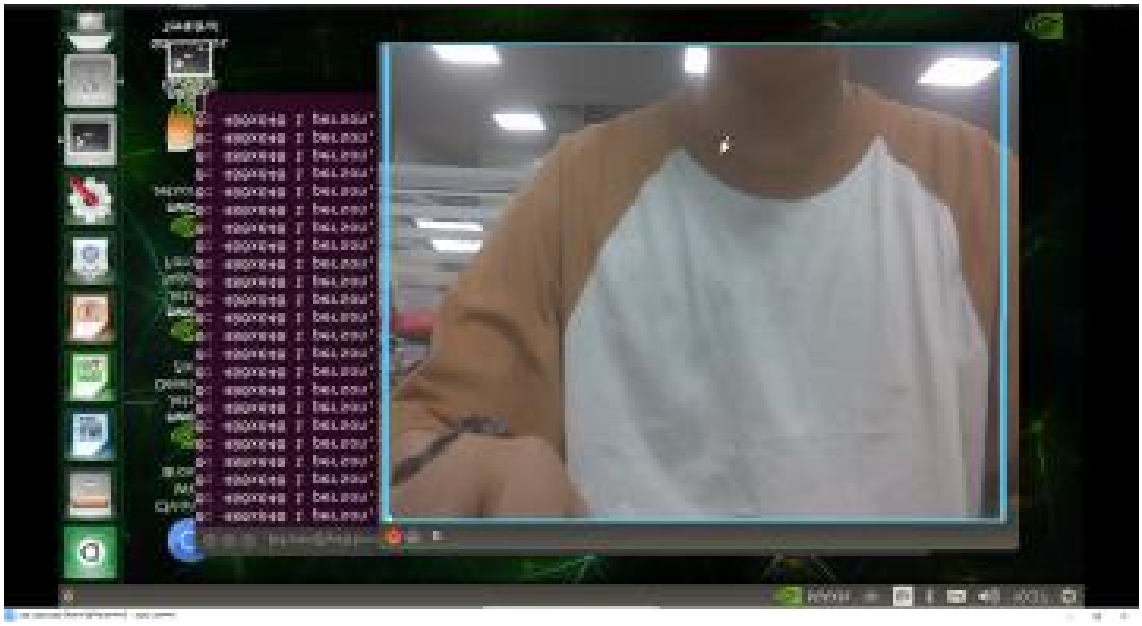
如果是直接使用 YAHBOOM 版的镜像,并且使用的是 USB 摄像头 需要在 ~/yolov5/utils 的 datasets.py 文件中进行一个简单的修改 对 292 行做取消注释'#'。293 行增加'#'。



然后运行以下的命令

```
cd ~/yolov5 && python3 detect.py --source 0
```

等待一段时间后，USB 摄像头就打开了 可以发现画面会显示出识别的物体



按 Ctrl+c 并且关掉摄像头的画面，即可结束程序 并且在 yolov5/runs/detect/exp 路径下存放识别出来的结果（一个视频）

2.注意事项

1. 如果中途因为网络的问题报错，可以从环境搭建的附件中的 yolov5s.pt 放到 yolov5 的文件夹下
2. CSI 摄像头跑本教程只能在 orin NX 16G 的板子和对应的系统才能正常使用，其它板子因为 jatpack 的版本和自身的功率、性能的不兼容 yoloV5 的原因是打不开的
3. 如果是自己搭建的镜像，不是使用 YAHBOOM 版配置好的镜像，需要自行改写 datasets.py 的文件，可参考下面的链接 https://blog.csdn.net/AlwaysNoError/article/details/123298884 如果

自己搭建的镜像还出现 SPPF 报错，可参考一下教程

https://blog.csdn.net/m0_50004939/article/details/126739291

16. yolo5+tensorrt 加速

yolo5+tensorrt 加速

yolo5+tensorrt 加速 1.使用前注意 2.开始使用 3.测试是否可以实现加速

1.使用前注意

如果使用的是 YAHBOOM 版的镜像，无需搭建环境，如果是自行搭建环境需要下载 tensorrt 的包 yoloV5 的版本和 tensorrt 的包要对应 **本教程使用的是 yolo5 V5.0 的版本，所以 tensorrt 加速的包也要使用 V5.0 的** 这是 tensorrt 下载链接：

https://github.com/wang-xinyu/tensorrtx/tree/master/yolov5

2.开始使用

1. 把 tensorrt/yolov5 下的 gen_wts.py 复制到 yolov5 的文件夹下
2. 执行 gen_wts.py 生成.wts 文件。

```
python3 gen_wts.py yolov5s.pt
```

1. 到目录 tensorrtx 下的 yolov5 文件夹老规矩，创建一个 build 文件，并进入

```
mkdir build
cd build
cmake ..
```

1. 将 yololayer.h 里的 CLASS_NUM 修改成你的。官方用的是 coco 数据集，所以默认是 80。(使用的是官方的可以忽略这步)
2. 执行 makeFile。（每次修改为 CLASS_NUM 都要 make 一次）

```
make -j2
```

1. 从 yoloV5 的文件路径下的 wts 文件复制到 tensorrtx/yolov5 里。
2. 生成.engine 文件

```
sudo ./yolov5 -s ../yolov5s.wts yolov5s.engine s
```

3.测试是否可以实现加速

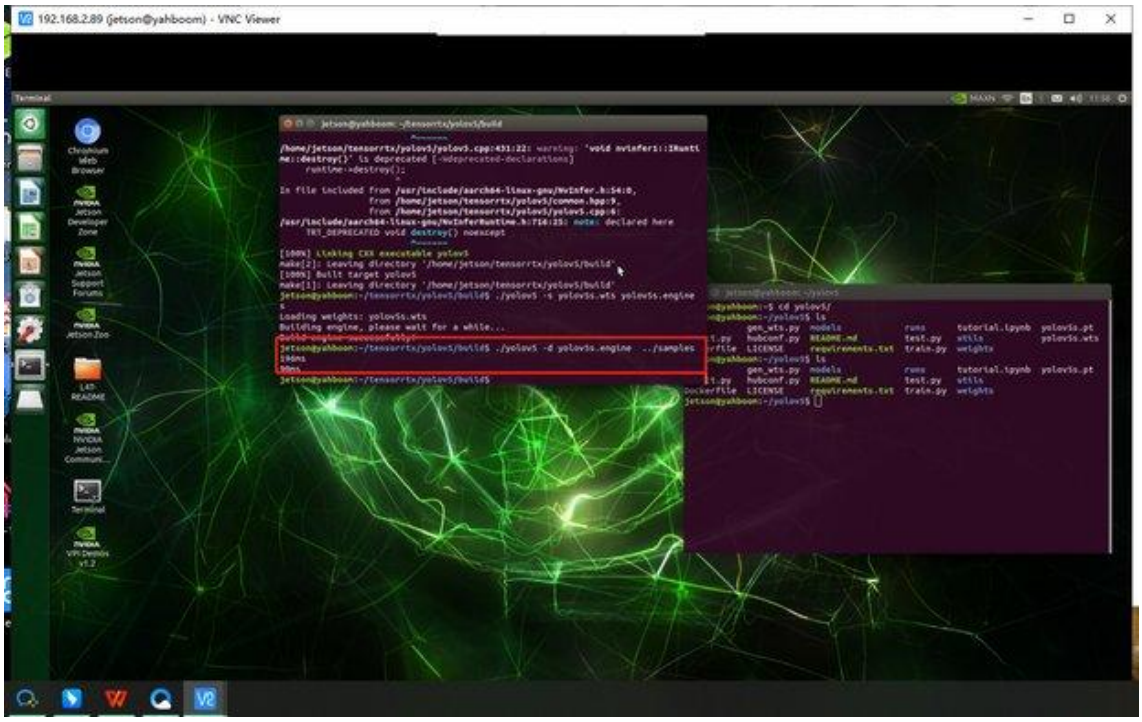
方法 1：进入 tensorrtx/yolo5/build 的文件夹下，运行命令

```
sudo ./yolov5 -d yolov5s.engine ../samples
```

可以和在 yolov5 文件夹下运行

```
python3 detect.py
```

进行一个对比，可以很清楚的看到加入了 tensorrtx 后，识别的图片时间大大减小



方法 2：进入 tensorrtx/yolo5 的文件夹下，运行命令

```
python3 yolov5_trt.py
```

同样和可以和在 yolov5 文件夹下运行

```
python3 detect.py
```

进行一个对比，可以很清楚的看到加入了 tensorrtx 后，识别的图片时间大大减小

17. yolo5+tensorrt 加速+DeepStream(打开摄像头)

yolo5+tensorrt 加速+DeepStream(打开摄像头)

yolo5+tensorrt 加速+DeepStream(打开摄像头)1.使用前注意事项 2.使用说明 2.1 模型转换 2.2 部署模型 2.3 修改 deepstream 配置文件（YAHBOOM 版的镜像可省略此步）3.编译运行

1.使用前注意事项

如果采用的是 YAHBOOM 版的镜像，不需要搭建 DeepStream 这部分的环境了。如果是自己搭建的镜像，这需要搭建 DeepStream 这部分的环境，可参考我们提供的 DeepStream 搭建教程，也可自行百度

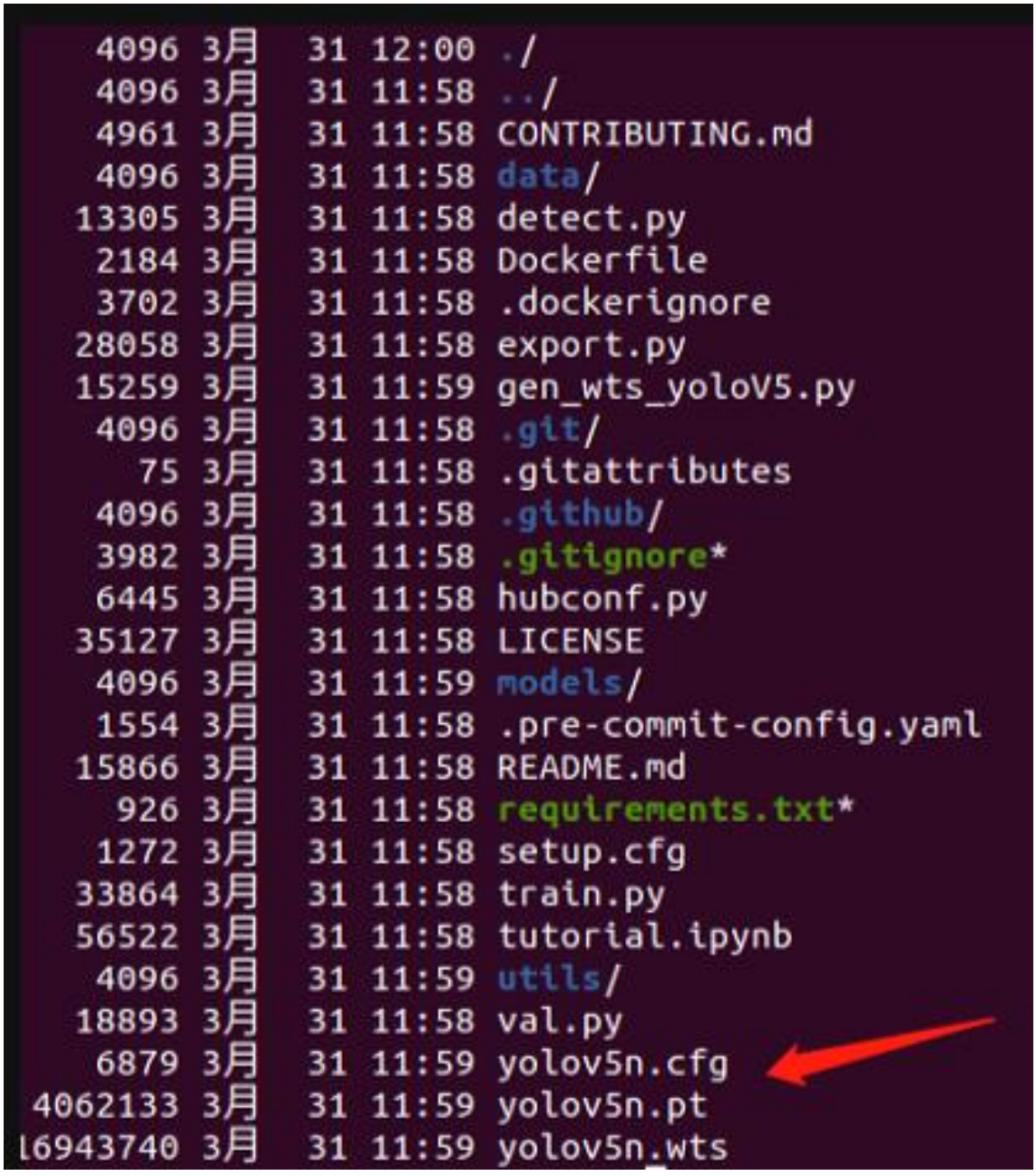
2.使用说明

2.1 模型转换

```
git clone https://github.com/marcoslucianops/DeepStream-Yolo.git   #yahboom 的镜像是不需要这步的
cd DeepStream-Yolo/utils    #这里的 DeepStream-Yolo 是在~/目录下的
#主要拷贝转换脚本到自己的 yolov5 项目下即可
cp gen_wts_yoloV5.py ../../yolov5
cd ../../yolov5
#根据自己的权重文件修改
python3 gen_wts_yoloV5.py -w  ./yolov5s.pt
```

2.2 部署模型

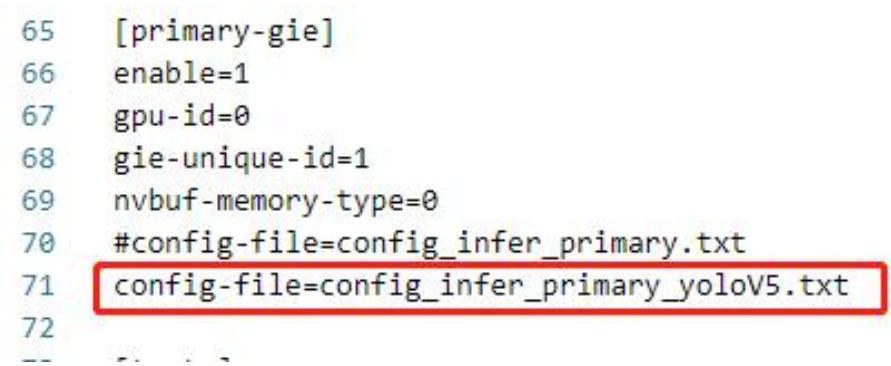
1. 成功运行上一步后，yolov5 的目录中会出现两个文件，yolov5n.cfg 以及 yolov5n.wts



2. 把 yolov5n.cfg 和 yolov5n.wts 放到 jetson nano 的 DeepStream-Yolo 文件夹中

2.3 修改 deepstream 配置文件（YAHBOOM 版的镜像可省略此步）

1. 修改 deepstream_app_config.txt 文件 修改的内容如下： 注释 70 行，在后面添加一行： config-file=config_infer_primary_yoloV5.txt 如图所示：



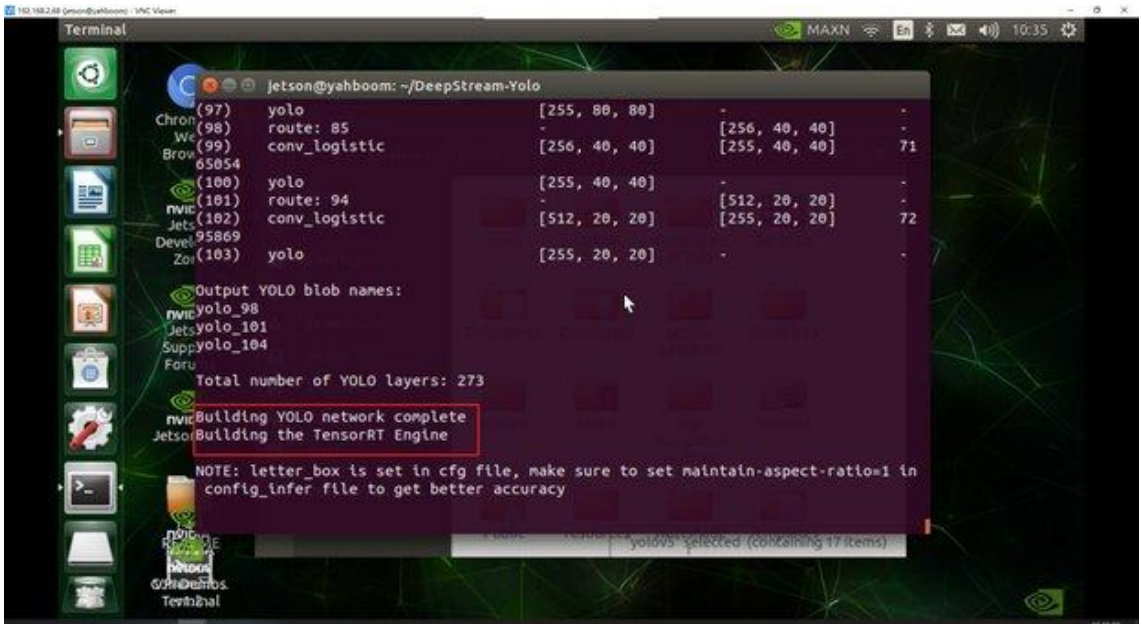
2. 修改第二个配置文件 config_infer_primary_yoloV5.txt

```
[property]
# 省略 ...
**model-engine-file=model_b2_gpu0_fp16.engine** # 修改 fp32->fp16
batch-size=2 # batch-size 改为 2，速度会快一些
# 省略...
**network-mode=2 #** 修改为 2，强制使用 fp16 推理
# 省略 ...
```

注意：FPS 与输入图像大小，batch-size，interval 等参数有关，需要根据实际应用优化，这里我们直接把 infer 的 batch-size 改为 2，模型的推理速度就会有明显提升

3.编译运行

```
cd nvdsinfer_custom_impl_Yolo/

CUDA_VER=11.4 make -j4 #根据你的 CUDA 版本修改 11.4 数字部分

cd ..

deepstream-app -c deepstream_app_config.txt
```



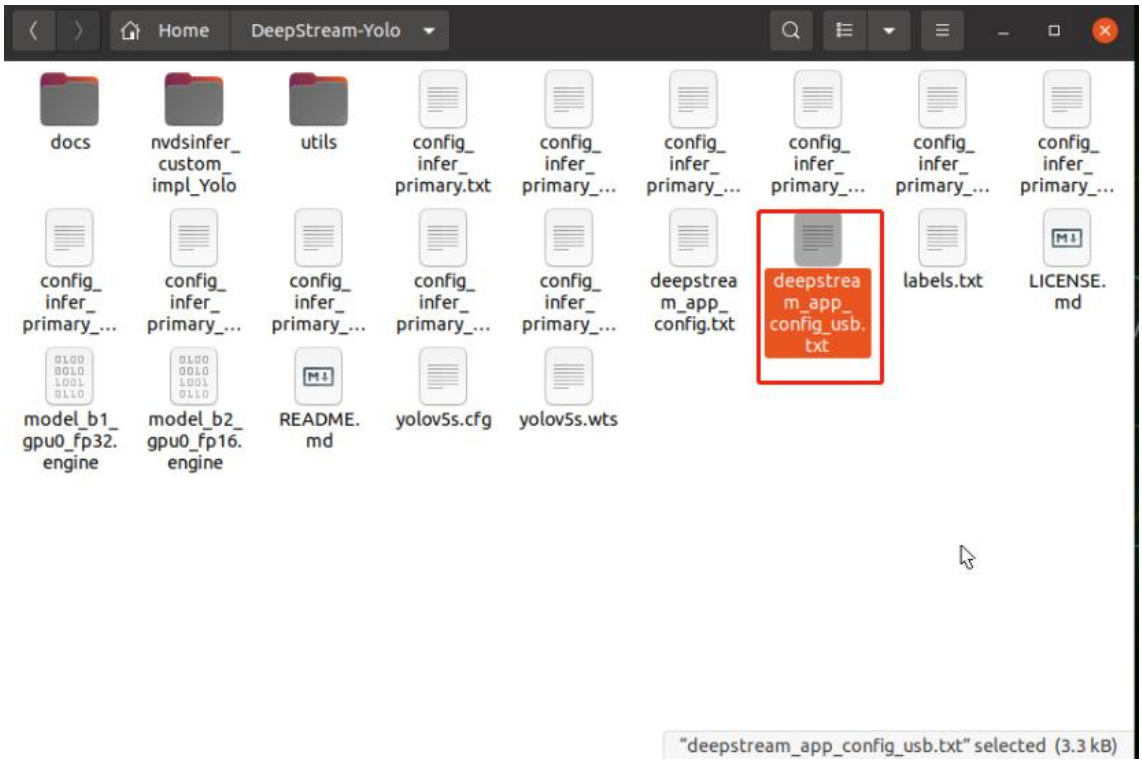
等待一段时间，可以看到 CSI 摄像头的画面打开了

注意

1. 如果是使用自己的搭建的镜像，需要在 deepstream_app_config.txt 进行修改 修改如图所示：

```
#[source0]
#enable=1
#type=3
#uri=file:///opt/nvidia/deepstream/deepstream/samples/streams/sample_1080p_h264.m
#num-sources=1
#gpu-id=0
#cudadec-memtype=0

[source1]
enable=1
#Type - 1=CameraV4L2 2=URI 3=MultiURI 4=RTSP 5=CSI
type=5
camera-csi-sensor-id=0
camera-width=1280
camera-height=720
camera-fps-n=30
camera-fps-d=1
```

2. 如果使用的是 USB 摄像头，需要把资料里面的附件里面的 **deepstream_app_config_usb.txt** 这个文件上传到 jetson，和 **deepstream_app_config.txt** 这文件是同一个地方的,如下图所示：



然后运行

```
cd ~/DeepStream-Yolo
deepstream-app -c deepstream_app_config_usb.txt
```

稍等一会，就能实现检测了

deepsteram 文件配置的参数说明，相关的参考连接如下：

https://blog.csdn.net/weixin_38369492/article/details/104859567

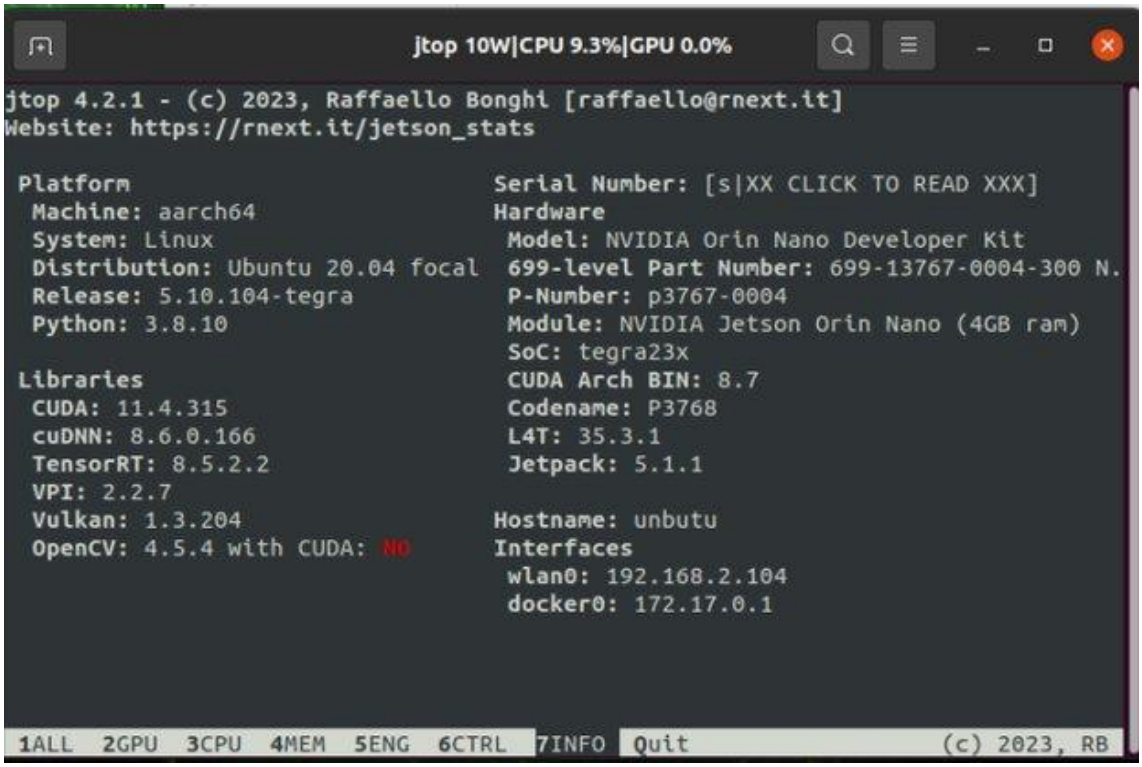
18. Mediapipe 环境搭建（选看）

Mediapipe 环境搭建

Mediapipe 环境搭建 1.使用前说明 2. 环境搭建附录

1.使用前说明

本教程适用于自己搭建的镜像，如果使用的是 YAHBOOM 版的镜像，本教程可以忽略 本教程的环境配置如下图所示



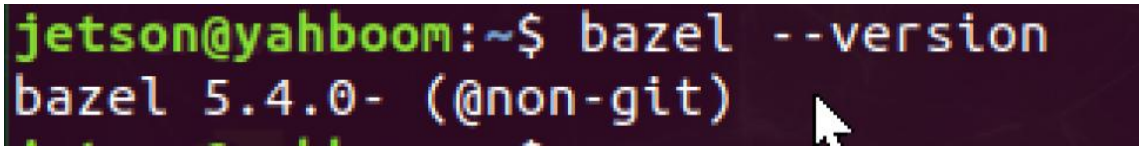
2. 环境搭建

1. 直接把环境搭建的附件中的 bazel 和 mediapipe-0.8.4-cp38-cp38-linux_aarch64.whl 两个文件传输到 jetson orin Nano 上

2. 在 jetson orin Nano 运行以下命令

```
sudo chmod +x bazel

mv bazel /usr/local/bin
```

输入以下的命令:bazel --version 结果如图所示:



1. 安装 mediapipe

```
pip3 install opencv-contrib-python==3.4.17.63

pip3 install mediapipe-0.8.4-cp38-cp38-linux_aarch64.whl

pip3 uninstall opencv-contrib-python  #因为 jetson orin Nano 已经自带的了
```

4.验证是否安装成功

```
python3

import mediapipe as mp
```

如果没有报错，则说明安装成功

附录

其它参考教程

https://blog.csdn.net/weixin_43659725/article/details/120211312

19. **Mediapipe 开发**

MediaPipe 开发

mediapipe github：https://github.com/google/mediapipe mediapipe 官网：https://google.github.io/mediapipe/ dlib 官网：http://dlib.net/ dlib github：https://github.com/davisking/dlib

1.简介

MediaPipe 是一款由 Google 开发并开源的数据流处理机器学习应用开发框架。它是一个基于图的数据处理管线，用于构建使用了多种形式的数据源，如视频、音频、传感器数据以及任何时间序列数据。 MediaPipe 是跨平台的，可以运行在嵌入式平台(树莓派等)，移动设备(iOS 和 Android)，工作站和服务器上，并支持移动端 GPU 加速。 MediaPipe 为实时和流媒体提供跨平台、可定制的 ML 解决方案。

MediaPipe 的核心框架由 C++ 实现，并提供 Java 以及 Objective C 等语言的支持。MediaPipe 的主要概念包括数据包(Packet)、数据流(Stream)、计算单元（Calculator）、图（Graph）以及子图（Subgraph）。

MediaPipe 的特点：

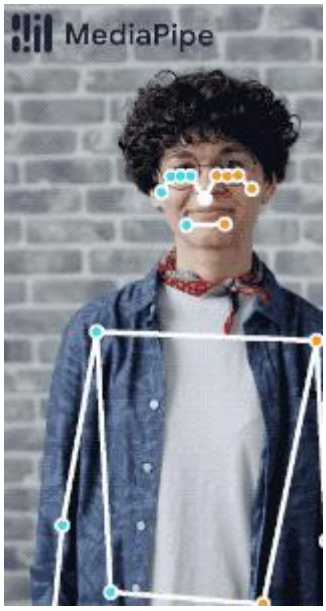
端到端加速：内置的快速 ML 推理和处理即使在普通硬件上也能加速。

一次构建，随时随地部署：统一解决方案适用于 Android、iOS、桌面/云、web 和物联网。

即用解决方案：展示框架全部功能的尖端 ML 解决方案。

免费开源：Apache2.0 下的框架和解决方案，完全可扩展和定制。

MediaPipe 中的深度学习解决方案

Face Detection	Face Mesh	Iris	Hands	Pose	Holis
					
Hair Segmentation	Object Detection	Box Tracking	Instant Motion Tracking	Objectron	KNI

Android	iOS	C++	Python	JS	Coral	
Face Detection	✅	✅	✅	✅	✅	✅
Face Mesh	✅	✅	✅	✅	✅	
Iris	✅	✅	✅			
Hands	✅	✅	✅	✅	✅	
Pose	✅	✅	✅	✅	✅	
Holistic	✅	✅	✅	✅	✅	
Selfie Segmentation	✅	✅	✅	✅	✅	
Hair Segmentation	✅		✅			
Object Detection	✅	✅	✅			✅
Box Tracking	✅	✅	✅			
Instant Motion Tracking	✅					

Objectron	✅		✅	✅	✅	
KNIFT	✅					
AutoFlip			✅			
MediaSequence			✅			
YouTube 8M			✅			

2.使用

这里只示范 py 文件的案例。

```
cd ~/med_test/scripts                                    #如果是自己搭建的镜像，在我们提供的附件/源代码里面可以找到
python3 06_FaceLandmarks.py                              # 人脸特效
python3 07_FaceDetection.py                              # 人脸检测
python3 08_Objectron.py                                  # 三维物体识别
python3 09_VirtualPaint.py                               # 画笔
python3 10_HandCtrl.py                                   # 手指控制
python3 11_GestureRecognition.py                         # 手势识别
```

在使用过程中，需注意的有

- 手部检测、姿态检测、整体检测、面部检测均具备点云查看功能，以面部检测为例。
- 所有功能【q 键】为退出。
- 整体检测：包括手部、脸部、身体姿态检测。
- 三维物体识别：可识别的物体有：【'Shoe', 'Chair', 'Cup', 'Camera'】，一共 4 类；点击【f 键】切换识别物体；jetson 系列不可用键盘按键，切换识别物体需改源码中的【self.index】参数。
- 画笔：右手食指和中指合并时是选择状态，同时弹出颜色选框，两指尖移动到对应颜色位置时，选中该颜色（黑色为橡皮擦）；食指和中指分开始是绘画状态，可在画板上任意绘制。
- 手指控制：点击【f 键】切换识别效果。
- 手指识别：以右手为准设计的手势识别，满足特定条件时，均可以准确识别。可识别的手势有：【Zero、One、Two、Three、Four、Five、Six、Seven、Eight、Ok、Rock、Thumb_up（点赞）、Thumb_down（拇指向下）、Heart_single（单手比心）】，一共 14 类。

程序运行：

```
python3 06_FaceLandmarks.py                    # 人脸特效
```

效果图：

```
python3 07_FaceDetection.py                    # 人脸检测
```

效果图：

```
python3 08_Objectron.py                    # 三维物体识别
```

效果图：

```
python3 09_VirtualPaint.py                    # 画笔
```

效果图：

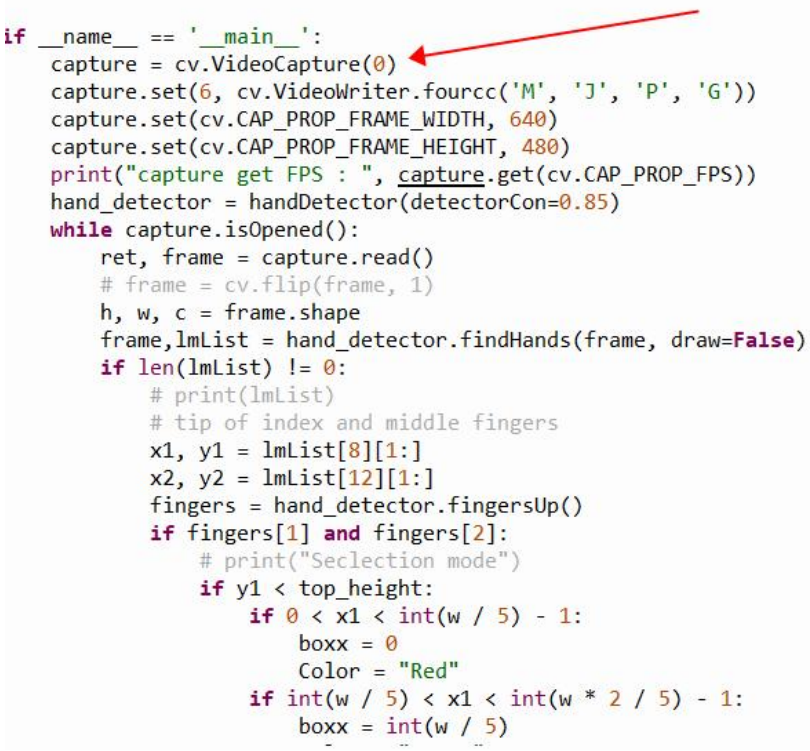
```
python3 10_HandCtrl.py                    # 手指控制
```

效果图：

```
python3 11_GestureRecognition.py                    # 手势识别
```

效果图：

注意：这几个案例都是在 CSI 板载摄像头上使用的，要是想用 usb 摄像头可以修改程序里面的 capture 改成下图这样。



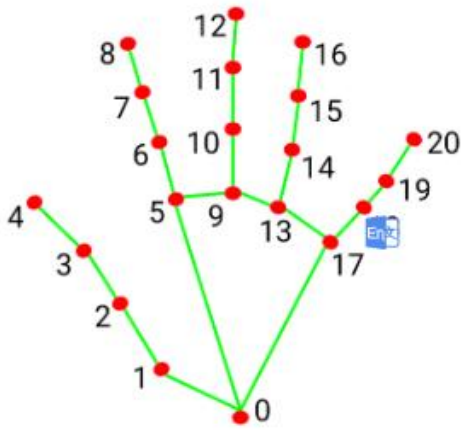
```
if __name__ == '__main__':
    capture = cv.VideoCapture(0)
    capture.set(6, cv.VideoWriter.fourcc('M', 'J', 'P', 'G'))
    capture.set(cv.CAP_PROP_FRAME_WIDTH, 640)
    capture.set(cv.CAP_PROP_FRAME_HEIGHT, 480)
    print("capture get FPS : ", capture.get(cv.CAP_PROP_FPS))
    hand_detector = handDetector(detectorCon=0.85)
    while capture.isOpened():
        ret, frame = capture.read()
        # frame = cv.flip(frame, 1)
        h, w, c = frame.shape
        frame,lmList = hand_detector.findHands(frame, draw=False)
        if len(lmList) != 0:
            # print(lmList)
            # tip of index and middle fingers
            x1, y1 = lmList[8][1:]
            x2, y2 = lmList[12][1:]
            fingers = hand_detector.fingersUp()
            if fingers[1] and fingers[2]:
                # print("Seclection mode")
                if y1 < top_height:
                    if 0 < x1 < int(w / 5) - 1:
                        boxx = 0
                        Color = "Red"
                    if int(w / 5) < x1 < int(w * 2 / 5) - 1:
                        boxx = int(w / 5)
```

3.MediaPipe Hands

MediaPipe Hands 是一款高保真的手和手指跟踪解决方案。它利用机器学习（ML）从一帧中推断出 21 个手的 3D 坐标。

在对整个图像进行手掌检测后，根据手部标记模型通过回归对检测到的手区域内的 21 个 3D 手关节坐标进行精确的关键点定位，即直接坐标预测。该模型学习一致的内部手姿势表示，甚至对部分可见的手和自我遮挡也具有鲁棒性。

为了获得地面真实数据，用了 21 个 3D 坐标手动注释了约 30K 幅真实世界的图像，如下所示 (从图像深度图中获取 Z 值，如果每个对应坐标都有 Z 值)。为了更好地覆盖可能的手部姿势，并对手部几何体的性质提供额外的监督，还绘制了各种背景下的高质量合成手部模型，并将其映射到相应的 3D 坐标。

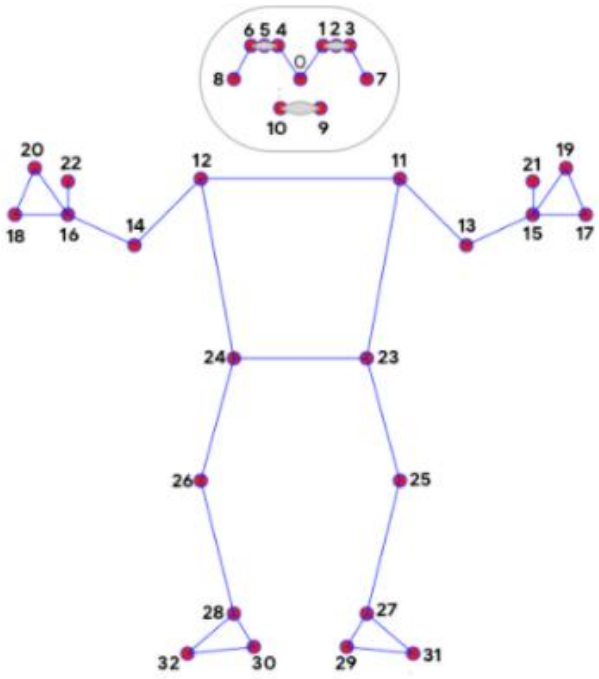


0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

4、MediaPipe Pose

MediaPipe Pose 是一个用于高保真身体姿势跟踪的 ML 解决方案，利用 BlazePose 研究，从 RGB 视频帧推断出 33 个 3D 坐标和全身背景分割遮罩，该研究也为 ML Kit 姿势检测 API 提供了动力。

MediaPipe 姿势中的地标模型预测了 33 个姿势坐标的位置（参见下图）。



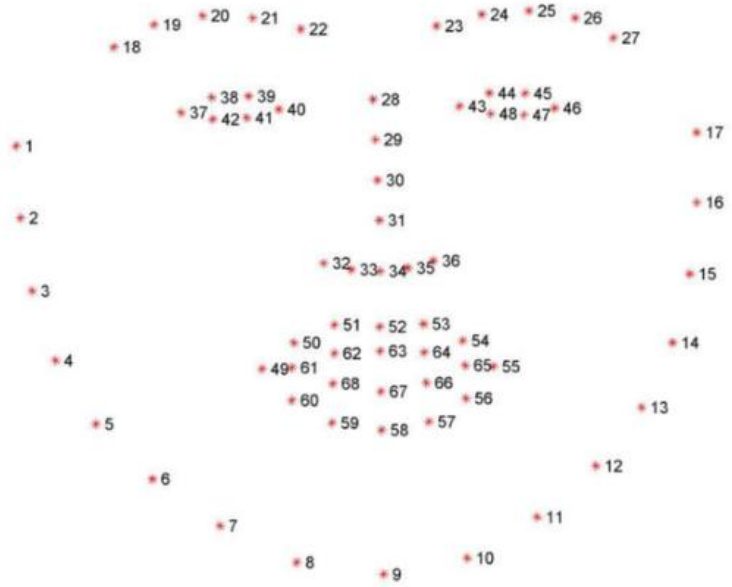
0. nose
1. left_eye_inner
2. left_eye
3. left_eye_outer
4. right_eye_inner
5. right_eye
6. right_eye_outer
7. left_ear
8. right_ear
9. mouth_left
10. mouth_right
11. left_shoulder
12. right_shoulder
13. left_elbow
14. right_elbow
15. left_wrist
16. right_wrist
17. left_pinky
18. right_pinky
19. left_index
20. right_index
21. left_thumb
22. right_thumb
23. left_hip
24. right_hip
25. left_knee
26. right_knee
27. left_ankle
28. right_ankle
29. left_heel
30. right_heel
31. left_foot_index
32. right_foot_index

5.dlib

对应的案例是人脸特效。

DLIB 是一个现代 C++工具包，包含机器学习算法和工具，用于在 C++中创建复杂的软件来解决现实世界问题。它被工业界和学术界广泛应用于机器人、嵌入式设备、移动电话和大型高性能计算环境等领域。

dlib 库采用 68 点位置标志人脸重要部位，比如 18-22 点标志右眉毛，51-68 标志嘴巴。使用 dlib 库的 get_frontal_face_detector 模块探测出人脸，使用 shape_predictor_68_face_landmarks.dat 特征数据预测人脸特征数值



第六章 ROS1 进阶教程

1. ROS1 基础教程

1、ros 简介

1、ROS 简介

1、ROS 简介 1.1、ROS 的主要特点 1.2、ROS 的整体架构 1.2.1、计算图级 1.2.2、文件系统级 1.2.3、开源社区级 1.3、通讯机制 1.3.1、Topic1.3.2、Service1.3.3、Action1.4、常用组件 1.5、发行版本

ROS wiki : http://wiki.ros.org/

ROS 教学：http://wiki.ros.org/ROS/Tutorials

ROS 安装：http://wiki.ros.org/melodic/Installation/Ubuntu

ROS（Robot Operating System，简称“ROS”）是一个适用于机器人的开源的操作系统。它提供了操作系统应有的服务，包括硬件抽象，底层设备控制，常用函数的实现，进程间消息传递，以及包管理。它也提供用于获取、编译、编写、和跨计算机运行代码所需的工具和库函数。

ROS 的主要目标是为机器人研究和开发提供代码复用的支持。ROS 是一个分布式的进程（也就是“节点”）框架，这些进程被封装在易于被分享和发布的程序包和功能包中。ROS 也支持一种类似于代码储存库的联合系统，这个系统也可以实现工程的协作及发布。这个设计可以使一个工程的开发和实现从文件系统到用户接口完全独立决策（不受 ROS 限制）。同时，所有的工程都可以被 ROS 的基础工具整合在一起。

1.1、ROS 的主要特点

（1）分布式架构(每一个工作进程都看作一个节点，使用节点管理器统一管理)，

（2）多语言支持(如 C++、Python 等)，

（3）良好的伸缩性(既可以写一个节点，也可以通过 roslaunch 将很多节点组织成一个更大的工程)，

（4）源码开放(ROS 遵循 BSD 协议，对个体和商业应用及修改完全免费)。

1.2、ROS 的整体架构

开源社区级 ： 主要包括开发人员知识、代码、算法共享。

文件系统级 ： 用于描述可以在硬盘上查到的代码及可执行程序，

计算图级 ： 体现进程与进程、进程与系统之间的通讯。

1.2.1、计算图级

节点

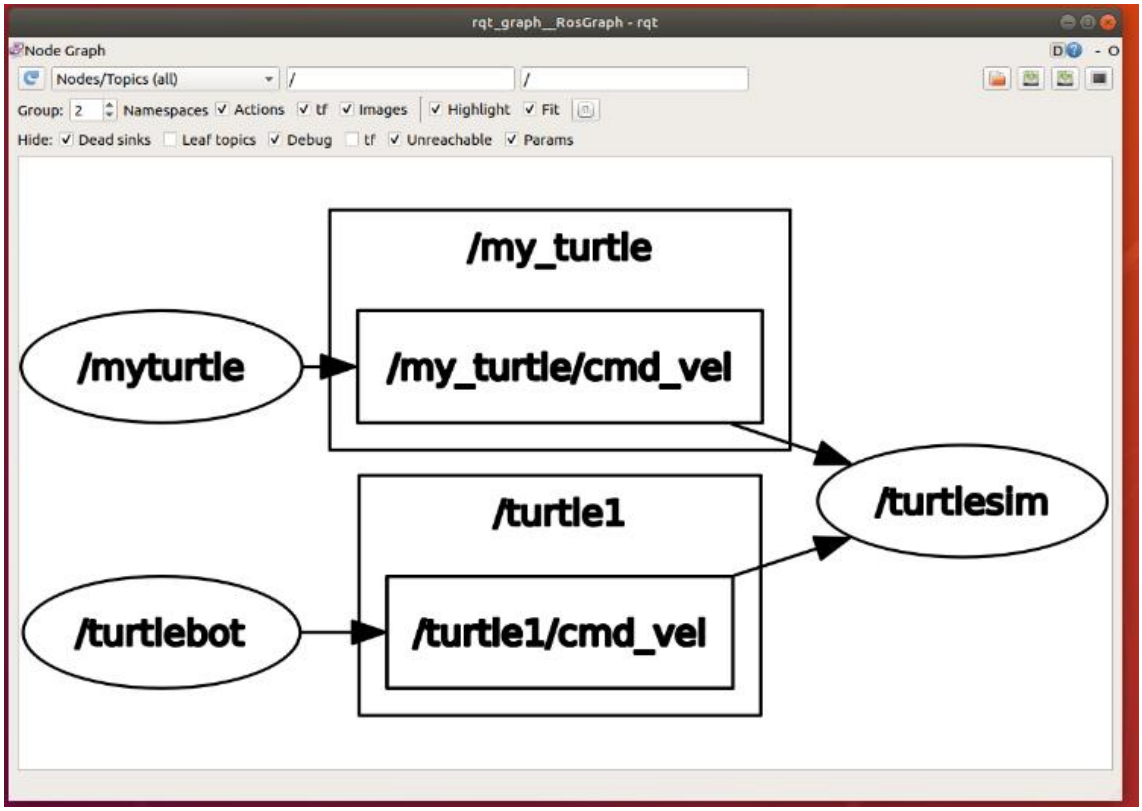
```
roscore
rosrun turtlesim turtlesim_node
```

下列命令是整条，输入一部分后用【Tab】键补齐即可，然后修改内容

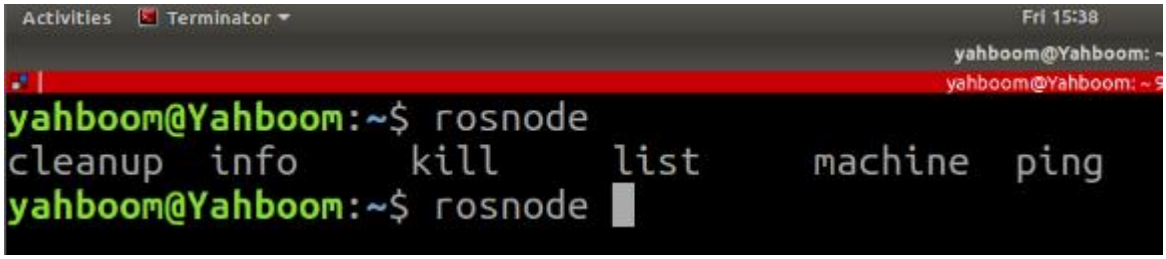
```
rosservice call /spawn "x: 3.0
y: 3.0
theta: 90.0
name: 'my_turtle'"
```

节点是主要的计算执行进程。ROS 是有很多节点组成的。

```
rqt_graph
```



当我们在命令行里输入【rosnode 】，然后双击 Tab 键,会发现命令行下方出现这样几个单词



ROS 命令行工具 rosnode：

rosnode 命令	作用
rosnode list	查询当前运行的所有节点

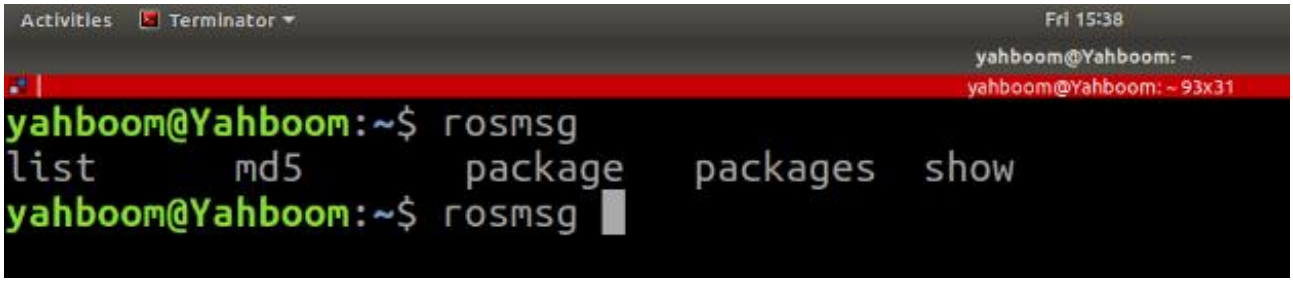
rosnode 命令	作用
rosnode info node_name	显示该节点的详细信息
rosnode kill node_name	结束某个节点
rosnode ping	测试该节点是否存活
rosnode machine	列出在特定机器或列表机器上运行的节点
rosnode cleanup	清除不可运行节点的注册信息

在开发调试时经常会需要查看当前 node 以及 node 信息，所以请记住这些常用命令。如果想不起来，也可以通过 `rosnode help` 来查看 `rosnode` 命令的用法。

- 消息

节点之间通过消息实现彼此的逻辑联系与数据交换。

当我们在命令行里输入【rosmsg】，然后双击 `Tab` 键,会发现命令行下方出现这样几个单词



ROS 命令行工具 rosmsg：

rosmsg 命令	作用
rosmsg show	显示一条消息字段
rosmsg list	列出所有消息
rosmsg package	列出所有功能包消息
rosmsg packages	列出所有具有该消息的功能包
rosmsg md5	显示一条消息的 MD5 检验值

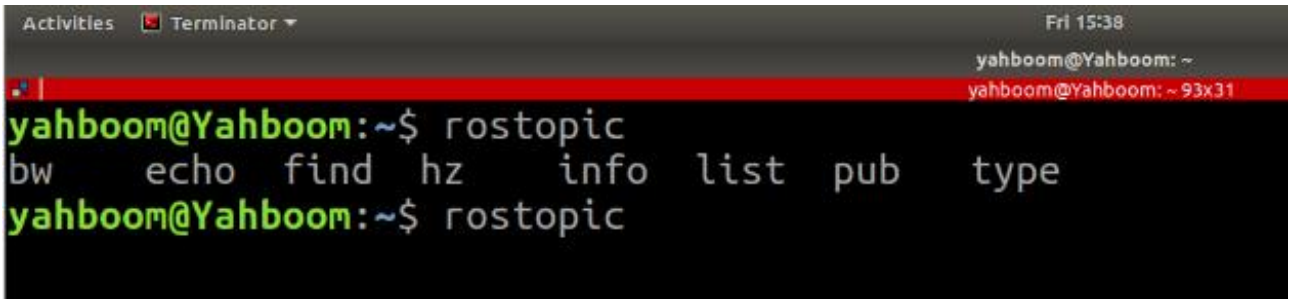
- 话题（主题）

话题是一种传递消息（发布/订阅）的方式。每一条消息都要发布到相应的主题上，每一个话题都是强类型的。

ROS 的话题消息可以使用 TCP/IP 或 UDP 传输，ROS 默认使用的传输方式是 TCP/IP。基于 TCP 传输成为 TCPROS，是一种长连接方式；基于 UDP 传

输的成为 UDPROS，是一种低延迟、高效率的传输方式，但容易丢失数据，适合于远程操作。

当我们在命令行里输入【rostopic】，然后双击 Tab 键,会发现命令行下方出现这样几个单词



ROS 命令行工具 rostopic：

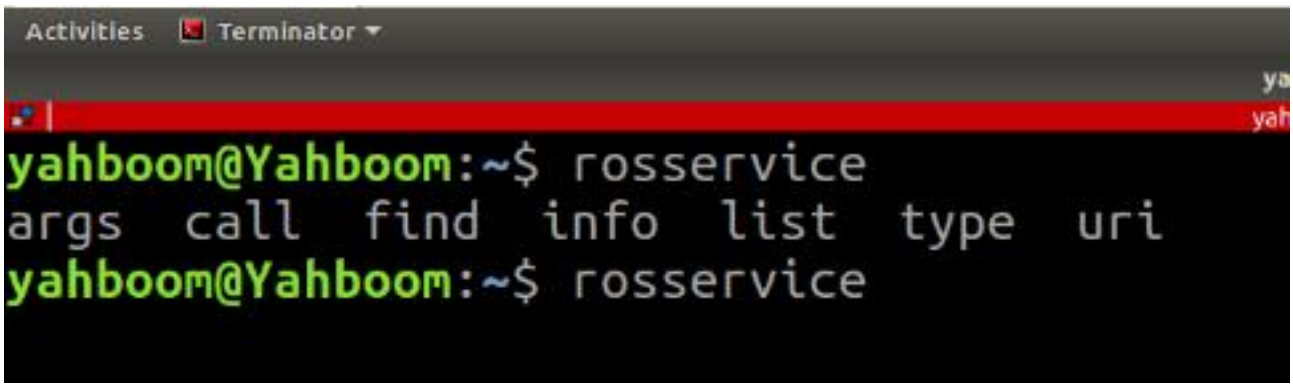
rostopic 命令	作用
rostopic bw /topic	显示主题所使用的带宽
rostopic echo /topic	将主题对应的消息输出到屏幕
rostopic find message_type	按照类型查找主题
rostopic hz /topic	显示主题的发布频率
rostopic info /topic	输出关于该主题

rostopic 命令	作用
	的信息
rostopic list	输出活动主题列表
rostopic pub /topic type args	将数据发布到主题
rostopic type /topic	输出主题类型

- 服务

服务用于请求应答模型，也必须有一个唯一的名称。当一个节点提供某个服务时，所有的节点都可以通过使用 ROS 客户端所编写的代码与之通讯。

当我们在命令行里输入【rosservice】，然后双击 Tab 键,会发现命令行下方出现这样几个单词



ROS 命令行工具 rosservice：

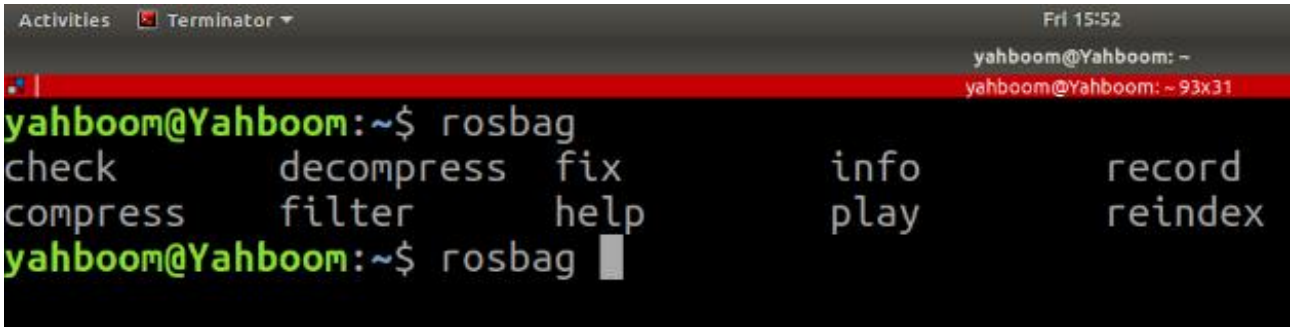
rosservice 命令	作用

rosservice 命令	作用
rosservice args /service	显示服务参数
rosservice call /service	用输入的参数请求服务
rosservice find /service	按照类型查找主题
rosservice info /service	显示指定服务的信息
rosservice list	显示活动的服务信息
rosservice uri /service	显示 ROSRPC URI 服务
rosservice type /service	显示服务类型

- 消息记录包

消息记录包是一种用于保存和回放 ROS 消息数据的文件格式，保存在.bag 文件中。是一种用于存储数据的重要机制。

当我们在命令行里输入【rosbag】，然后双击 Tab 键,会发现命令行下方出现这样几个单词



ROS 命令行工具 rosbag:

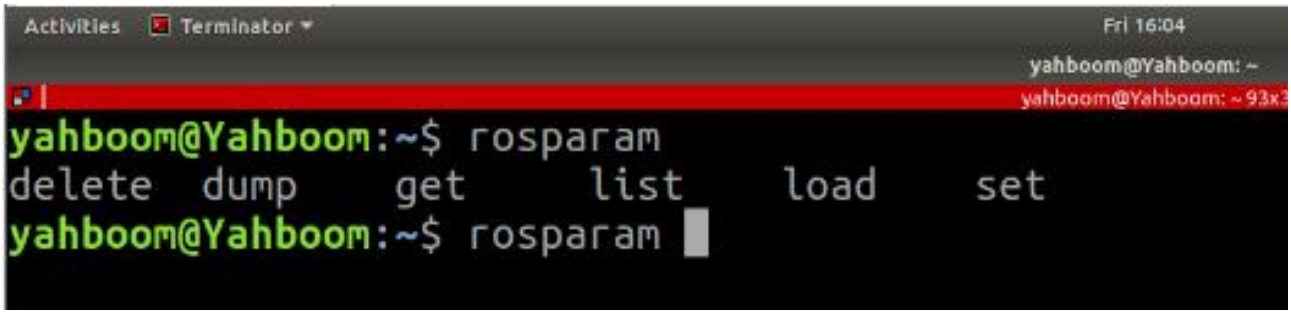
rosbag 命令	作用
check	确定一个包是否可以在当前系统中进行，或者是否可以迁移。
decompress	压缩一个或多个包文件。
filter	解压一个或多个包文件。
fix	在包文件中修复消息，以便在当前系统中播放。
help	获取相关命令指引帮助信息。
info	总结一个或多个包文件的内容。

rosbag 命令	作用
play	以一种时间同步的方式回放一个或多个包文件的内容。
record	用指定话题的内容记录一个包文件。
reindex	重新索引一个或多个包文件。

- 参数服务器

参数服务器是可通过网络访问的共享的多变量字典，通过关键字存储在节点管理器上。

当我们在命令行里输入【rosparam】，然后双击 Tab 键,会发现命令行下方出现这样几个单词



ROS 命令行工具 rosparam：

rosparam 命令	作用
rosparam delete	删除参数

rosparam 命令	作用
parameter	
rosparam dump file	将参数服务器中的参数写入到文件
rosparam get parameter	获得参数值
rosparam list	列出参数服务器中的参数
rosparam load file	从文件中加载参数到参数服务器
rosparam set parameter value	设置参数

- 节点管理器（Master）

节点管理器用于主题、服务名称的注册和查找等。在整个 ROS 系统中如果没有节点管理器，就不会有节点之间的通讯。

1.2.2、文件系统级

文件系统级

综合功能包

功能包

功能包清单

消息

服务

代码

其他

功能包之间可以配置依赖关系。如果功能包 A 依赖功能包 B，那么在 ROS 构建系统时，B 一定要早于 A 的构建，并且 A 可以使用 B 中的头文件和库文件。

文件系统级的概念如下：

- 功能包清单：

这个清单是指明功能包的依赖关系、源文件编译标志信息等。功能包中的 package.xml 文件就是一个功能包清单。

- 功能包：

功能包是 ROS 系统中软件组织的基本形式，包含运行的节点以及配置文件等。

ROS package 相关命令

rospack 命令	**作用**
rospack help	显示rospack的用法
rospack list	列出本机所有

rospack 命令	作用
	package
rospack depends [package]	显示 package 的依赖包
rospack find [package]	定位某个 package
rospack profile	刷新所有 package 的位置记录

- 综合功能包

  将几个功能包组织在一起，即可形成综合功能包。

- 消息类型

  ROS 中节点之间发送消息时需要事先进行消息说明。ROS 中提供了标准类型消息，也可以自行定义。消息类型的说明存储在功能包下的 msg 文件内。

- 服务类型

  定义了在 ROS 系统中由每个进程提供的关于服务请求和响应的数据结构。

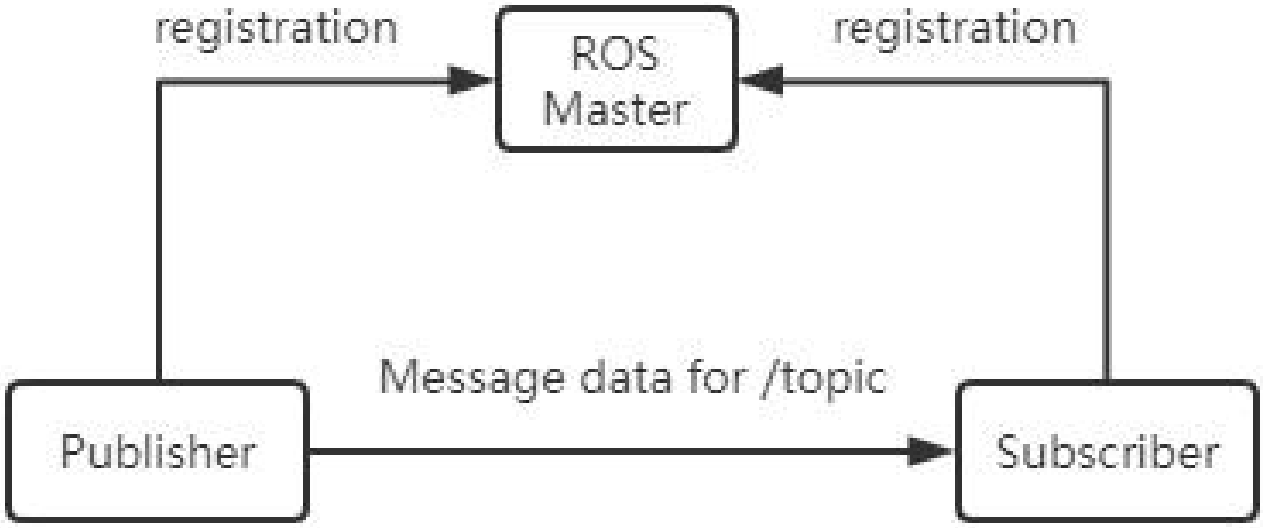
1.2.3、开源社区级

- 发行版（Distribution）：ROS 发行版是可以独立安装、带有版本号的一系列综合功能包。ROS 发行版像 Linux 发行版一样发挥类似的作用。这使得 ROS 软件安装更加容易，而且能够通过一个软件集合维持一致的版本。
- 软件库（Repository）：ROS 依赖于共享开源代码与软件库的网站或主机服务，在这里不同的机构能够发布和分享各自的机器人软件与程序。
- ROS 维基（ROS Wiki）：ROS Wiki 是用于记录有关 ROS 系统信息的主要论坛。任何人都可以注册账户、贡献自己的文件、提供更正或更新、编写教程以及其他行为。
- Bug 提交系统（Bug Ticket System）：如果你发现问题或者想提出一个新功能，ROS 提供这个资源去做这些。
- 邮件列表（Mailing list）：ROS 用户邮件列表是关于 ROS 的主要交流渠道，能够像论坛一样交流从 ROS 软件更新到 ROS 软件使用中的各种疑问或信息。
- ROS 问答（ROS Answer）：用户可以使用这个资源去提问题

1.3、通讯机制

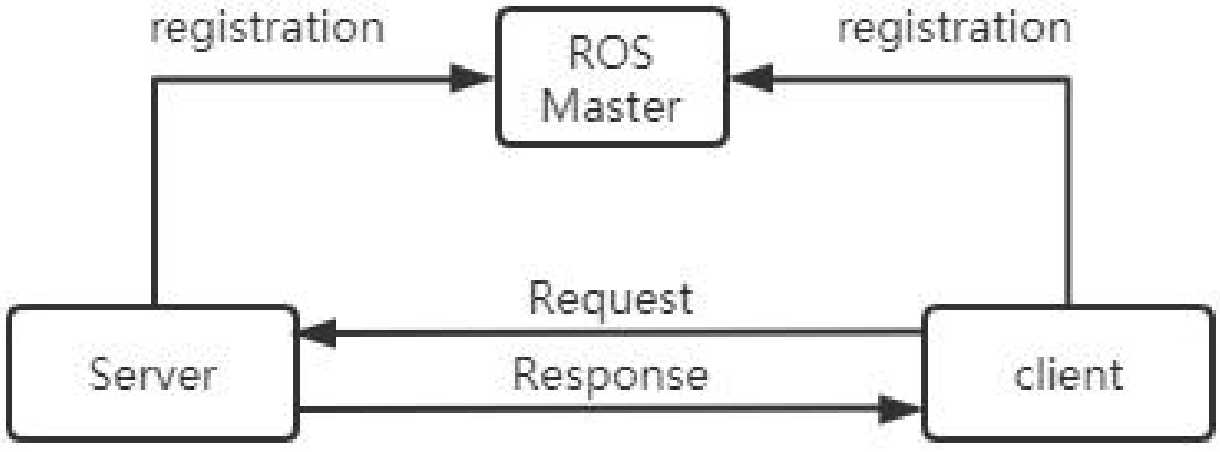
1.3.1、Topic

ros 中广为使用的是异步的 publish-subscribe 通讯模式。Topic 一般 用于单向，消息流通讯。Topic 一般拥有很强的类型定义：一种类型的 topic 只能接受/ 发送特定数据类型（message type）的 message。Publisher 没有被要求类型一致性，但是接受时 subscriber 会检查类型 的 md5，进而报错。



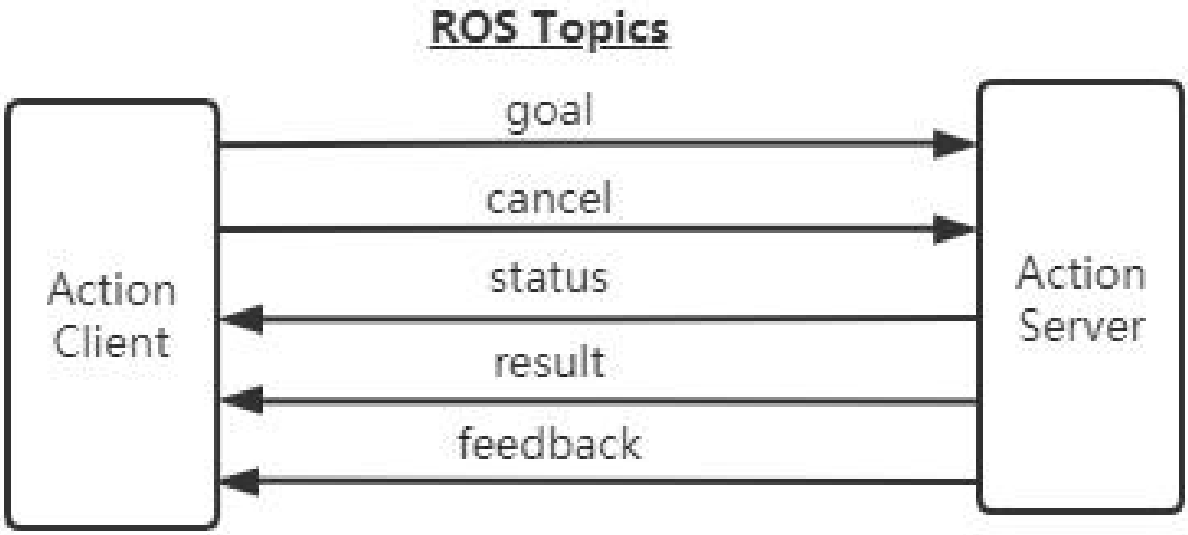
1.3.2、Service

service 用于处理 ros 通讯中的同步通讯，采用 server/client 语义。每个 service type 拥 有 request 与 response 两部分，对于 service 中的 server，ros 不会检查重名（name conflict），只有最后注册的 server 会生效，与 client 建立连接。



1.3.3、Action

action 使用多个 topic 组成，用于定义任务，任务定义包括目标（Goal）、任务执行过程状态反馈（Feedback）和结果（Result）等。编译 action 将会自动产生 7 个结构体分别为：Action、ActionGoal、ActionFeedback、ActionResult、Goal、Feedback、Result 结构体。



Action 的特点：

- 一种问答通讯机制
- 带有连续反馈
- 可以在任务中终止进行
- 基于 ROS 的消息机制实现

Action 的接口：

- goal：发布任务目标
- cancel：请求取消任务
- status：通知客户端当前状态
- feedback：周期反馈任务运行的监控数据

- result：向客户端发送任务的执行结果，只发布一次。

通讯模式特点对比

特点	Topic	Service	Action
响应机制	无	结果响应	进度响应，结果响应
同步性	异步	同步	异步
通讯模型	`Publisher`，`Subscriber`	`Client`，`Server`	`Client`，`Server`
Node对应关系	多对多	多（Client）对一（Server）	多（Client）对一（Server）

1.4、常用组件

launch 启动文件；TF 坐标变换；Rviz；Gazebo；QT 工具箱；Navigation；MoveIt!

launch：启动文件（Launch File）是 ROS 中一种同时启动多个节点的途径，它还可以自动启动 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便利。

TF 坐标变换：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，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。

QT 工具箱：为了方便可视化调试和显示，ROS 提供了一个 Qt 架构的后台图形工具套件——rqt_common_plugins，其中包含不少实用工具：日志输出工具(rqt_console)、计算图可视化工具（rqt_graph）、数据绘图工具（rqt_plot）、参数动态配置工具（rqt_reconfigure）

Rviz：rviz 是一款三维可视化工具，很好地兼容了各种基于 ROS 软件框架的机器人平台。在 rviz 中，可以使用 XML 对机器人、周围物体等任何实物进行尺寸、质量、位置、材质、关节等属性的描述，并且在界面中呈现出来。同时，rviz 还可以通过图形化方式，实时显示机器人传感器的信息、机器人的运动状态、周围环境的变化等。

Gazebo：Gazebo 是一个功能强大的三维物理仿真平台，具备强大的物理引擎、高质量的图形渲染、方便的编程与图形接口，最重要的还有其具备开源免费的特性。虽然 Gazebo 中的机器人模型与 rviz 使用的模型相同，但是需要在模型中加入机器人和周围环境的物理属性，例如质量、摩擦系数、弹性系数等。机器人的传感器信息也可以通过插件的形式加入仿真环境、以可视化的方式显示。

Navigation：navigation 是 ROS 的二维导航功能包，简单来说，就是根据输入的里程计等传感器的信息流和机器人的全局位置，通过导航算法，计算得出安全可靠的机器人速度控制指令。

Moveit：Moveit！功能包是最常用的工具包，主要用来进行轨迹规划。Moveit！配置助手用来配置一些在规划中需要用到的文件，非常关键。

1.5、发行版本

参考链接：http://wiki.ros.org/Distributions

Distro	Release date	Poster	*Tuturtle*, turtle in tutorial	EOL date
ROS Noetic Ninjemys (**Recommended**)	May 23rd, 2020			May, 2025 (Focal EOL)
ROS Melodic Morenia	May 23rd, 2018			May, 2023 (Bionic EOL)
ROS Lunar Loggerhead	May 23rd, 2017			May, 2019
ROS Kinetic Kame	May 23rd, 2016			April, 2021 (Xenial EOL)
ROS Jade Turtle	May 23rd, 2015			May, 2017
ROS Indigo Igloo	July 22nd, 2014			April, 2019 (Trusty EOL)
ROS Hydro Medusa	September 4th, 2013			May, 2015
ROS Groovy Galapagos	December 31, 2012			July, 2014
ROS Fuerte Turtle	April 23, 2012			--
ROS Electric Emys	August 30, 2011			--
ROS Diamondback	March 2, 2011			--

ROS 的发行版本（ROS distribution）指 ROS 软件包的版本，其与 Linux 的发行版本（如 Ubuntu）的概念类似。推出 ROS 发行版本的目的在于使开发人员可以使用相对稳定的代码库，直到其准备好将所有内容进行版本升级为止。因此，每个发行版本推出后，ROS 开发者通常仅对这一版本的 bug 进行修复，同时提供少量针对核心软件包的改进。截至 2019 年 10 月，ROS 的主要发行版本的版本名称、发布时间与版本生命周期如下表所示 ：

版本名称	发布日期	版本生命周期	操作系统平台
ROS Noetic Ninjemys	2020 年 5 月	2025 年 5 月	Ubuntu 20.04
ROS Melodic Morenia	2018 年 5 月 23 日	2023 年 5 月	Ubuntu 17.10, Ubuntu 18.04, Debian 9, Windows 10
ROS Lunar Loggerhead	2017 年 5	2019 年 5	Ubuntu 16.04,

版本名称	发布日期	版本生命周期	操作系统平台
	月 23 日	月	Ubuntu 16.10, Ubuntu 17.04,Debian 9
ROS Kinetic Kame	2016 年 5 月 23 日	2021 年 4 月	Ubuntu 15.10, Ubuntu 16.04, Debian 8
ROS Jade Turtle	2015 年 5 月 23 日	2017 年 5 月	Ubuntu 14.04, Ubuntu 14.10, Ubuntu 15.04
ROS Indigo	2014	2019	Ubuntu

版本名称	发布日期	版本生命周期	操作系统平台
Igloo	年 7 月 22 日	年 4 月	13.04, Ubuntu 14.04
ROS Hydro Medusa	2013 年 9 月 4 日	2015 年 5 月	Ubuntu 12.04, Ubuntu 12.10, Ubuntu 13.04
ROS Groovy Galapagos	2012 年 12 月 31 日	2014 年 7 月	Ubuntu 11.10, Ubuntu 12.04, Ubuntu 12.10
ROS Fuerte Turtle	2012 年 4	--	Ubuntu 10.04,

版本名称	发布日期	版本生命周期	操作系统平台
	月 23 日		Ubuntu 11.10, Ubuntu 12.04
ROS Electric Emys	2011 年 8 月 30 日	--	Ubuntu 10.04, Ubuntu 10.10, Ubuntu 11.04, Ubuntu 11.10
ROS Diamondback	2011 年 3 月 2 日	--	Ubuntu 10.04, Ubuntu 10.10, Ubuntu 11.04

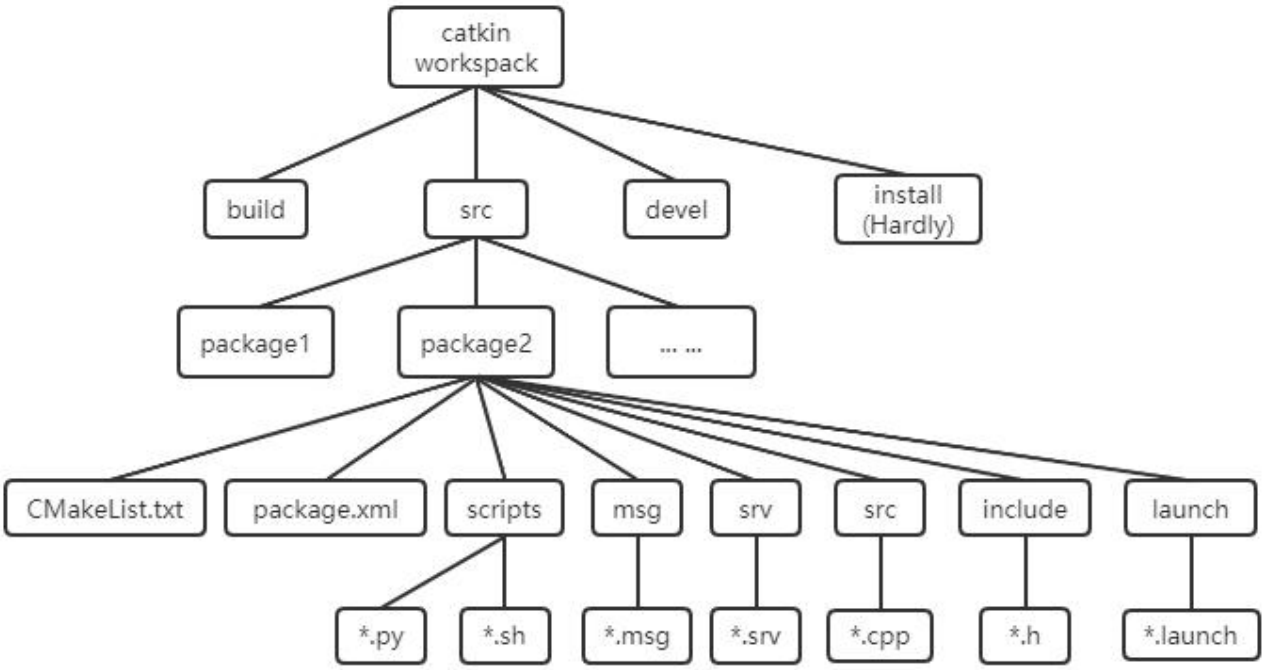
版本名称	发布日期	版本生命周期	操作系统平台
ROS C Turtle	2010年8月2日	--	Ubuntu 9.04, Ubuntu 9.10, Ubuntu 10.04, Ubuntu 10.10
ROS Box Turtle	2010年3月2日	--	Ubuntu 8.04, Ubuntu 9.04, Ubuntu 9.10, Ubuntu 10.04

2、项目文件结构

2、项目文件结构

2.1、项目文件结构

ROS 的文件结构，并非每个文件夹都是必须的，根据业务需求设计。



2.2、工作空间

工作空间就是管理和组织 ROS 工程项目文件的地方，直观的形容就是一个仓库，里面装载着 ROS 的各种项目工程，便于系统组织管理调用。在可视化图形界面里是一个文件夹。我们自己写的 ROS 代码通常就放在工作空间中。其下主要的一级目录有四个：

src：源空间；ROS 的 catkin 软件包（源代码包）

build：编译空间；catkin（CMake）的缓存信息和中间文件

devel：开发空间；生成的目标文件（包括头文件，动态链接库，静态链接库，可执行文件等）、环境变量

install：安装空间

最顶层的工作空间（可以任意命名）和 src （必须为 src）文件夹是需要自己创建；

- build 和 devel 文件夹由 catkin_make 命令自动创建；
- install 文件夹由 catkin_make install 命令自动创建，几乎不怎么使用，一般不创建。

注意：使用 catkin_make 编译之前一定要回到最顶层的工作空间。同一个工作空间下，不允许存在同名功能包；不同工作空间下，允许存在同名功能包。

```
mkdir -p ~/catkin_ws/src  # 创建
cd catkin_ws/             # 进入工作空间
catkin_make               # 编译
source devel/setup.bash   # 更新工作空间环境
```

2.3、package 功能包

package 是一种特定的文件结构和文件夹组合。通常将实现同一个具体功能的程序代码放到一个 package 中。只有 CMakeLists.txt 和 package.xml 是【**必须**】的，其余路径根据软件包是否需要来决定。

创建功能包

```
cd ~/catkin_ws/src
catkin_create_pkg my_pkg rospy rosmsg roscpp
```

【rospy】、【rosmsg】、【roscpp】是依赖库，可根据业务需求添加，也可加其他的，创建时加上就不需要再配置，忘记添加需要自己配置。

文件结构

```
|-- CMakeLists.txt  # （必须）当前 package 的编译规则。通常需要为 c++ 代码添加编译时的依赖，执行等操作。
|—— package.xml     # （必须）package 的描述信息。通常添加一些 ros 库的支持。
|—— include 文件夹   # 存放 c++ 头文件的
|—— config 文件夹    # 存放参数配置文件
|—— launch 文件夹    # 存放 launch 文件(.launch 或.xml)
|—— meshes 文件夹    # 存放机器人或仿真场景的 3D 模型 (.sda, .stl, .dae 等）；
|—— urdf 文件夹      # 存放机器人的模型描述(.urdf 或.xacro）；
```

```
|—— rviz 文件夹          # rviz 文件
|—— src 文件夹           # c++源代码
|—— scripts 文件夹       # 可执行脚本；例如 shell 脚本(.sh)、Python 脚本(.py)；
|—— srv 文件夹           # 自定义 service
|—— msg 文件夹           # 自定义 topic
|—— action 文件夹        # 自定义 action
```

2.4、CMakeLists.txt 介绍

2.4.1、概述

CMakeLists.txt 原本是 Cmake 编译系统的规则文件，而 Catkin 编译系统基本沿用了 CMake 的编译风格，只是针对 ROS 工程添加了一些宏定义。所以在写法上，catkin 的 CMakeLists.txt 与 CMake 的基本一致。

这个文件直接规定了这个 package 要依赖哪些 package，要编译生成哪些目标，如何编译等等流程。所以 CMakeLists.txt 非常重要，它指定了由源码到目标文件的规则，catkin 编译系统在工作时首先会找到每个 package 下的 CMakeLists.txt，然后按照规则来编译构建。

2.4.2、格式

CMakeLists.txt 的基本语法都还是按照 CMake，而 Catkin 在其中加入了少量的宏，总体的结构如下：

```
cmake_minimum_required() # 所需 CMake 版本
project()                # 软件包名称
find_package()           # 找到编译需要的其他 CMake/Catkin package
catkin_python_setup()    # 启用 Python 模块支持
add_message_files()      # message 生成器
add_service_files()      # service 生成器
add_action_files()       # action 生成器
generate_message()       # 生成不同语言版本的 msg/srv/action 接口
catkin_package()         # 生成当前 package 的 cmake 配置，供依赖本包的其他软件包调用
add_library()            # 用来指定编译产生的库。默认的 catkin 编译产生共享库。
add_executable()         # 生成可执行二进制文件
add_dependencies()       # 定义目标文件依赖于其他目标文件，确保其他目标已被构建
target_link_libraries()  # 指定可执行文件链接的库。这个要用在add_executable()后面。
catkin_add_gtest()       # 测试建立
install()                # 安装至本机
```

2.4.3、Boost

如果使用 C ++和 Boost，您需要在 Boost 上调用 find_package（），并指定用作组件的 Boost 的哪些方面。例如，如果你想使用 Boost 线程，你会说：

```
find_package（Boost REQUIRED COMPONENTS thread）
```

2.4.4、catkin_package（）

catkin_package（）是一个 **catkin** 提供的 CMake 宏。这是为构建系统指定 catkin 特定信息所必需的，后者又用于生成 pkg-config 和 CMake 文件。

在使用 add_library（）或 add_executable（）声明任何目标之前，**必须调用**此函数。该函数有 5 个**可选**参数：

- INCLUDE_DIRS - 包的导出包含路径
- LIBRARIES - 从项目导出的库
- CATKIN_DEPENDS - 该项目依赖的其他 catkin 项目
- DEPENDS - 该项目所依赖的非 catkin CMake 项目。为了更好的理解，请看这个解释。
- CFG_EXTRAS - 其他配置选项

完整的宏文档可以在这里找到。

举个例子：

```
catkin_package(
```

```
  INCLUDE_DIRS include
  LIBRARIES ${PROJECT_NAME}
  CATKIN_DEPENDS roscpp nodelet
  DEPENDS eigen opencv)
```

这表示包文件夹中的文件夹“include”是导出头文件的地方。CMake 环境变量$ {PROJECT_NAME}评估到之前传给 project（）函数的任何东西，在这种情况下它将是“robot_brain”。“roscpp”+“nodelet”是需要存在来构建/运行此程序包的软件包，“eigen”+“opencv”是需要存在的用于构建/运行此程序包的系统依赖项。

2.4.5、包括路径和库路径

在指定目标之前，您需要指定可以为所述目标找到资源的位置，特别是头文件和库：

```
- 包括路径 - 在哪里可以找到代码（最常见于 C / C ++）的头文件
- 库路径 - 哪些库位于该可执行目标建立对象？
- include_directories（<dir1>，<dir2>，...，<dirN>）
- link_directories（<dir1>，<dir2>，...，<dirN>）
```

- include_directories（）

include_directories 的参数应该是 find_package 调用和需要包含的任何其他目录生成的* _INCLUDE_DIRS 变量。如果您使用 catkin 和 Boost，您的 include_directories（）调用应该如下所示：

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

第一个参数“include”表示包中的 include /目录也是路径的一部分。

- link_directories（）

例：

```
link_directories(~/my_libs)
```

CMake link_directories（）函数可用于添加额外的库路径，但是不推荐这样做。所有 catkin 和 CMake 软件包在 find_packaged 时都会自动添加链接信息，只需链接到 target_link_libraries（）中的库。

请参阅本 cmake 的线程可以看到使用的详细例子 target_link_libraries（）在 link_directories（）。

2.4.6、可执行目标

要指定必须构建的可执行目标，我们必须使用 add_executable（） CMake 函数。

```
add_executable(myProgram src/main.cpp src/some_file.cpp
src/another_file.cpp)
```

这将构建一个名为 myProgram 的目标可执行文件，它由 3 个源文件构建：src / main.cpp，src / some_file.cpp 和 src / another_file.cpp。

2.4.7、库文件

该 add_library () CMake 的功能是用来指定库来构建。默认情况下，catkin 构建共享库。

```
add_library（$ {PROJECT_NAME} $ {$ {PROJECT_NAME} _SRCS})
```

2.4.8、target_link_libraries

使用 target_link_libraries（）函数来指定可执行目标链接的库。这通常在 add_executable（）调用之后完成。如果找不到 ros，则添加 $ {catkin_LIBRARIES}。

句法：

```
target_link_libraries（<executableTargetName>，<lib1>，<lib2>，...
<libN>）
```

例：

```
add_executable(foo src/foo.cpp)
add_library(moo src/moo.cpp)
target_link_libraries(foo moo)   -- This links foo against libmoo.so
```

请注意，在大多数用例中不需要使用 link_directories（），因为信息通过 find_package（）自动引入。

2.4.8、消息、服务和动作

消息（.msg），服务（.srv）和动作（.action）文件在 ROS 包构建和使用之前需要一个特殊的预处理器构建步骤。这些宏的要点是生成编程语言特定的文件，以便可以利用其选择的编程语言中的消息，服务和动作。构建系统将使用所有可用的生成器（例如 gencpp，genpy，genlisp 等）生成绑定。

提供了三个宏来分别处理消息，服务和动作：

- add_message_files
- add_service_files
- add_action_files

这些宏之后必须跟随调用生成的宏：

```
generate_messages（）
```

如果你从未接触过 CMake 的语法，请阅读《CMake 实践》：https://github.com/Akagi201/learning-cmake/blob/master/docs/cmake-practice.pdf 。掌握 CMake 语法对于理解 ROS 工程很有帮助。

2.5、package.xml 介绍

- 概述

该软件包清单是一个 XML 文件名为 package.xml 中必须包括与任何兼容包的根文件夹。package.xml 也是一个 catkin 的 package 必备文件，它是这个软件包的描述文件，在较早的 ROS 版本(rosbuild 编译系统)中，这个文件叫做 manifest.xml，用于描述 pacakge 的基本信息。如果你在网上看到一些 ROS 项目里包含着 manifest.xml，那么它多半是 hydro 版本之前的项目了。pacakge.xml 包含了 package 的名称、版本号、内容描述、维护人员、软件许可、编译构建工具、编译依赖、运行依赖等信息。

package.xml 文件 build_depend 必须包含 message_generation，run_depend 必须包含 message_runtime。

- 格式

```
<pacakge>                # 根标记文件
<name>                   # 包名
<version>                # 版本号
<description>            # 内容描述
```

```
<maintainer>            # 维护者
<license>               # 软件许可证
<buildtool_depend>      # 编译构建工具，通常为 catkin
<depend>                # 指定依赖项为编译、导出、运行需要的依赖，最常用
<build_depend>          # 编译依赖项
<build_export_depend>   # 导出依赖项
<exec_depend>           # 运行依赖项
<test_depend>           # 测试用例依赖项
<doc_depend>            # 文档依赖项
```

- 依赖关系

具有最小标签的包清单不指定对其他包的任何依赖关系。软件包可以有六种依赖关系：

构建依赖关系<build_depend>指定构建此包所需的包。在构建时需要这些软件包中的任何文件时才是这种情况。这可以包括在编译时的头文件，链接到这些包的库文件或在构建时需要任何其他资源（特别是当这些包在 CMake 中是 find_package（）时）。在交叉编译场景中，构建依赖关系针对目标体系结构。

构建导出依赖关系<build_export_depend>指定根据此包构建库所需的包。当您将此头文件包含在此包中的公用头文件中时（特别是当 CMake 中的 catkin_package（）中声明为（CATKIN_DEPENDS 时），就是这种情况。

执行依赖关系<exec_depend>指定在此程序包中运行代码所需的软件包。当您依赖此程序包中的共享库（尤其是当 CMake 中的 catkin_package（）中声明为（CATKIN_DEPENDS ）时），就是这种情况。

测试依赖关系<test_depend>仅指定单元测试的附加依赖项。他们不应该将已经提到的任何依赖关系重复为构建或运行依赖关系。

构建工具依赖关系<buildtool_depend>指定此软件包需要构建自身的构建系统工具。通常唯一的构建工具是 catkin。在交叉编译场景中，构建工具依赖关系用于执行编译的架构。

文档工具依赖关系<doc_depend>指定此软件包需要生成文档的文档工具。

- 附加标签

`<url>` - 有关该软件包信息的 URL，通常是 ros.org 上的 wiki 页面。

`<author>` - 包的作者

```
<url type="website">http://www.ros.org/wiki/turtlesim</url>
<author>Yahboom</author>
```

3、常用命令与工具

3、常用命令与工具

3.1、启动节点方式

3.1.1、launch 文件

用 roslaunch 命令启动 launch 文件至少有两种方式：

1）、借助 ros package 路径启动

格式如下：

```
roslaunch package 名称 launch 文件名称
roslaunch pkg_name launchfile_name.launch
```

2）、直接给出 launch 文件的绝对路径

格式如下：

```
roslaunch path_to_launchfile
```

不论用上述哪种方式启动 launch 文件，都可以在后边添加参数，比较常见的参数有

--screen: 令 ros node 的信息（如果有的话）输出到屏幕上，而不是保存在某个 log 文件中，这样比较方便调试

arg:=value: 如果 launch 文件中有待赋值的变量，可以通过这种方式赋值，例如：

```
roslaunch pkg_name launchfile_name model:=urdf/myfile.urdf # launch file 中有参数 “model” 需要赋值
```

或

```
roslaunch pkg_name launchfile_name model:='$(find urdf_pkg)/urdf/myfile.urdf' # 用 find 命令提供路径
```

roslaunch 命令运行时首先会检测系统的 rosmaster 是否运行，如果已经启动，就用现有的 rosmaster；如果没有启动，会先启动 rosmaster,然后再执行 launch 文件中的设置，一次性把多个节点按照我们预先的配置启动起来。

需要注意的是， launch 文件不需要编译，设置好之后可以直接用上述方式运行。

3.1.2、rosrun

必须先启动节点管理器（master），master 是用来管理系统中的很多进程的，每个 node 启动时都要向 master 注册管理 node 之间的通信。master 启动后再去通过 master 注册每一个 node 节点。在 Ubuntu 终端中输入命令：

```
roscore
```

node 的启动，rosrun+包名+节点名；rosrun 方法每次只能运行一个节点。

```
rosrun [--prefix cmd] [--debug] pkg_name node_name [ARGS]
```

rosrun 将会寻找 package 的名为 executable 的可执行程序，将可选参数 ARGS 传入。

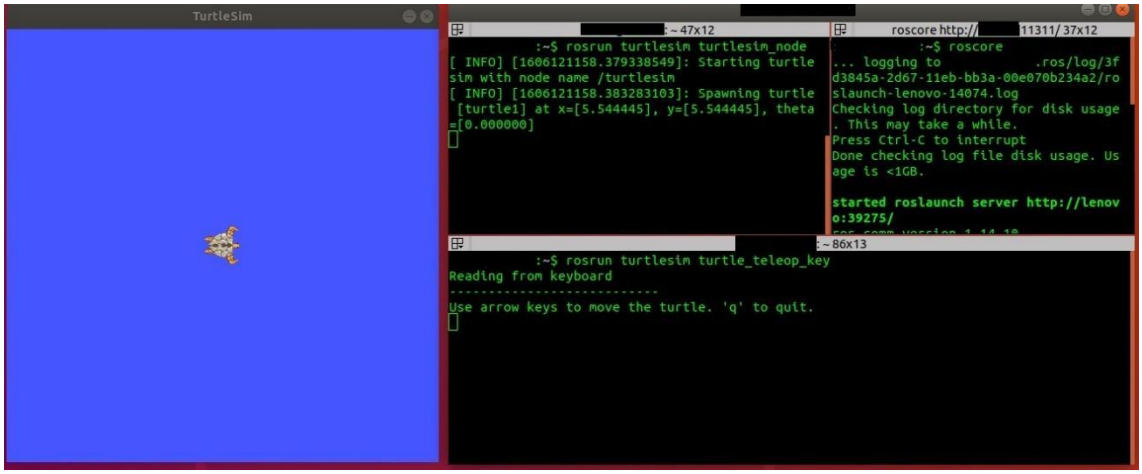
3.1.3、python

如果是 python 代码，可以直接在 py 文件所在目录下直接启动，注意区分 python2 和 python3。

3.1.4、启动一个小乌龟

```
roscore
rosrun turtlesim turtlesim_node      # 启动小海龟仿真器节点
rosrun turtlesim turtle_teleop_key  # 启动小海龟键盘控制器节点
```

启动完成后，可以通过键盘输入操控小海龟移动,键盘操控时，光标一定要在【rosrun turtlesim turtle_teleop_key】这个命令行下，点击键盘【上】、【下】、【左】、【右】控制小海龟移动。



并且在 rosrun turtlesim turtlesim_node 终端会打印一些小海龟的日志信息

```
[ INFO] [1607648666.226328691]: Starting turtlesim with node name /turtlesim
[ INFO] [1607648666.229275030]: Spawning turtle [turtle1] at x=[5.544445], y=[5.544445], theta=[0.000000]
```

3.1.5、启动两个小乌龟

安装功能包

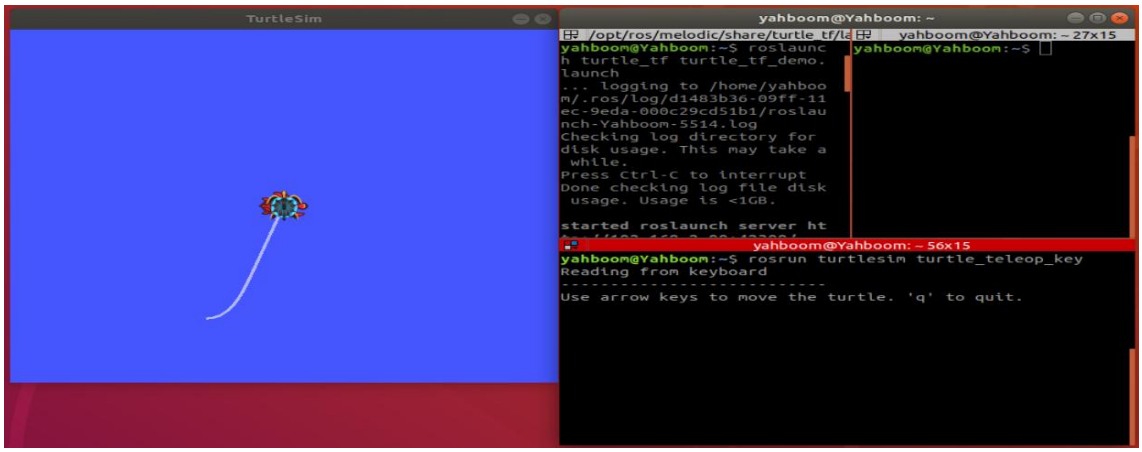
```
sudo apt install ros-melodic-turtle-tf
```

启动

```
roslaunch turtle_tf turtle_tf_demo.launch
```

键盘控制节点

```
rosrun turtlesim turtle_teleop_key
```



此时，按下键盘【上】、【下】、【左】、【右】驱动小乌龟运动；可以观察到一只小乌龟跟随另一只运动。

3.2、launch 文件

3.2.1、概述

在 ROS 中一个节点程序一般只能完成功能单一的任务，但是一个完整的 ROS 机器人一般由很多个节点程序同时运行、相互协作才能完成复杂的任务，因此这就要求在启动机器人时就必须要启动很多个节点程序，如果一个 node 一个 node 的启动，比较麻烦。通过 launch 文件以及 roslaunch 命令可以一次性启动多个 node，方便“一键启动”，并且可以设置丰富的参数。

3.2.1、文件的格式

launch 文件本质上是一种 xml 文件，在某些编辑器中可以高亮显示关键字，方便阅读，可以在头部添加，也可不添加

```
<?xml version="1.0"?>
```

与其他 xml 格式的文件类似，launch 文件也是通过**标签（tag）**的方式书写， 主要的 **tag** 如下：

```
<launch>                <!--根标签-->
<node>                  <!--需要启动的 node 及其参数-->
<include>               <!--包含其他 launch-->
<machine>                <!--指定运行的机器-->
<env-loader>             <!--设置环境变量-->
<param>                 <!--定义参数到参数服务器-->
```

```
<rosparam>                    <!--加载 yaml 文件中的参数到参数服务器-->
<arg>                         <!--定义变量-->
<remap>                       <!--设定 topic 映射-->
<group>                       <!--设定分组-->
</launch>                     <!--根标签-->
```

1.标签【node】

标签【node】是 launch 文件的核心部分。

```
<launch>
    <node pkg="package_name" type="executable_file"
name="node_name"/>
    <node pkg="another_package" type="another_executable"
name="another_node"></node>
    ...
</launch>
```

其中

- pkg 是节点所在的 package 名称
- type 是 package 中的可执行文件，如果是 python 编写的，就可能是 .py 文件，如果是 c++ 编写的，就是源文件编译之后的可执行文件的名字。

- name 是节点启动之后的名字，每一个节点都要有自己独一无二的名字。

  注意： roslaunch 不能保证 node 的启动顺序，因此 launch 文件中所有的 node 都应该对启动顺序有鲁棒性。

  还可以设置更多参数，如下：

```
<launch>
    <node
        pkg=""
        type=""
        name=""
        respawn="true"
        required="true"
        launch-prefix="xterm -e"
        output="screen"
        ns="namespace"
    />
</launch>
```

  上述命令中，

-

`respawn`：若该节点关闭，是否自动重新启动

-
-

`required`：若该节点关闭，是否关闭其他所有节点

-
-

`launch-prefix`： 是否新开一个窗口执行。例如，需要通过窗口进行机器人移动控制的时候，应该为控制 node 新开一个窗口；或者当 node 有些信息输出，不希望与其他 node 信息混杂在一起的时候。

-
-

`output`：默认情况下，launch 启动 node 的信息会存入下面的 log 文件中，可以通过此处参数设置，令信息显示在屏幕上

-
-
-
-
-

- /.ros/log/
-
-
-

  ns：将 node 归入不同的 namespace，即在 node name 前边加 ns 指定的前缀。为了实现这类操作，在 node 源文件中定义 node name 和 topic name 时要采用 relative name， 即不加符号 /。

-

  计算图源的名称分为：

  1）基础名称，例如：topic

  2）全局名称，例如：/A/topic

  3）相对名称，例如：A/topic

  4）私有名称，例如：~topic

  在发布或者定阅时，有这么一行代码

```
ros::init(argc, argv, "publish_node");
ros::NodeHandle nh;
ros::Publisher pub = nh.advertise<std_msgs::string>("topic",1000);
```

2.标签【remap】

经常作为 node 标签的子标签出现，可以用来修改 topic。在很多 rosnode 源文件中，可能并没有指定接收的或者发送的 topic，而仅仅是用 `input_topic` 和 `output_topic` 代替，这样在使用中需要将抽象的 topic 名字替换成具体场景中的 topic 名字。

简单地说，remap 的作用就是方便同一个 node 文件被应用到不同的环境中，用 remap 从外部修改一下 topic 即可，不需要改变源文件。

remap 常见的使用格式如下：

```
<node pkg="some" type="some" name="some">
    <remap from="origin" to="new" />
</node>
```

3.标签【include】

这个标签的作用是将另外一个 launch 文件添加到本 launch 文件中，类似 launch 文件的嵌套。基本格式：

```
<include file="path-to-launch-file" />
```

上边的文件路径可以给具体路径，但是一般来说为了程序的可移植性，最好借助 `find` 命令给出文件路径：

```
<include file="$(find package-name)/launch-file-name" />
```

上述命令中，`$(find package-name)` 等价于本机中相应 package 的路径。这样即使换了其他主控，只要安装了同样的 package，就可以找到对应的路径。

有时，另一个 launch 引入的 node 可能需要统一命名，或者具有类似特征的 node 名字，比如 /my/gps, /my/lidar, /my/imu，即 node 具有统一的前缀，方便查找。这可以通过设置 `ns` （namespace）属性来实现，命令如下：

```
<include file="$(find package-name)/launch-file-name " ns="my" />
```

4.标签【arg】

通过【arg】可以使参数重复使用，也便于多处同时修改。 【arg】三种常用方法:

- `<arg name="foo">`：声明一个 【arg】，但不赋值。稍后可以通过命令行赋值，或者通过【include】标签赋值。
- `<arg name="foo" default="1">`：赋默认值。
- `<arg name="foo" value="1">`：赋固定值。

通过命令行赋值

```
```

```
roslaunch package_name file_name.launch arg1:=value1
arg2:=value2
```

5.变量替换

在 launch 文件中常用的变量替换形式有两个

- `$(find pkg)`： 例如`$(find rospy)/manifest.xml`. 如果可能，强烈推荐这种基于 package 的路径设置
- `$(arg arg_name)`： 先设置默认值，如果没有额外的赋值，就用这个默认值了

例如:

```
<arg name="gui" default="true" />
<!-- 先设置默认值，如果没有额外的赋值，就用这个默认值了 -->
<param name="use_gui" value="$(arg gui)"/>
```

另一个例子:

```
<node pkg="package_name" type="executable_file"
name="node_name" args="$(arg a) $(arg b)" />
```

这样设置之后，在启动 roslaunch 时，可以为 args 参数赋值

```
roslaunch package_name file_name.launch a:=1 b:=5
```

6.标签【param】

与【arg 】不同，【param】是共享的，并且它的取值不仅限于 value，还可以是文件，甚至是一行命令。

- 格式

-
-
-
-
-
-
```
<param name="param_name" type="type1"
value="val"/>                    # type 可以省略，系统自动判断
```
-
-
```
<param name="param_name" textfile="$(find
pkg)/path/file"/>                # 读取 file 存成 string
```
-

- <param name="param_name" command="$(find pkg)/exe '$(find pkg)/arg.txt'"/>
-
- 实例：
-
- <param name="param" type="yaml" command="cat '$(find pkg)/*.yaml'"/> # command 的结果存在 param 中
-
- 【param】可以是在全局范围中，它的 name 就是原本的 name，也可以在某个更小的范围中，比如 node，那么它在全局的名字就是 node/param 形式.

  例如在全局范围中定义如下 param

  <param name="publish_frequency" type="double" value="10.0" />

  再在 node 范围中定义如下 param

  <node name="node1" pkg="pkg1" type="exe1">

```
    <param name="param1" value="False"/>
</node>
```

如果用 rosparam list 列出 server 中的 【param】，则有

```

/publish_frequency
/node1/param1    # 自动加上了 namespace 前缀
```

注意：虽然 【param】名字前加了 namespace，但是依然全局范围

7.标签【rosparam】

【param】只能对单个 【param】操作，而且只有三种：value、textfile、command 形式，返回的是单个 【param】的内容。 【rosparam】则可以批量操作，还包括一些对参数设置的命令，如 dump、delete 等

- load：从 YAML 文件中加载一批 param，格式如下：
-
-
-
-
-

- <rosparam command="load" file="$(find rosparam)/example.yaml" />
-
-
-
  delete: 删除某个 param
-
-
-
-
-
  <rosparam command="delete" param="my_param" />
-
-
-
  类似【param】的赋值操作
-

-
-
-
-
- `<rosparam param="my_param">[1,2,3,4]</rosparam>`
-
-

  或者
-
-
-
-
-
-
- `<rosparam>`
-
-

  `a: 1`

-
- b: 2
-
- </rosparam>
-
-

【rosparam】也可以放在【node】中，此时【param】名字前边都加上 node namespace.

8.标签【group】

如果要对多个 node 进行同样的设置，比如都在同一个特定的 namespace，remap 相同的 topic 等，可以用【group】。在【group】中可以使用所有常见的标签进行设置，例如

```
<group ns="rosbot">
    <remap from="chatter" to="talker"/>        # 对该 group 中后续所有 node 都有效
    <node ... />
    <node ... >
```

```
        <remap from="chatter" to="talker1"/>   # 各个 node 中可以重新设置 remap
    </node>
</group>
```

3.3、TF 坐标变换

tf 是一个允许用户随时跟踪多个坐标系的功能包。tf 维护实时缓冲的树结构中的坐标帧之间的关系，并允许用户在任意两个坐标帧之间任意时间点上转换点、向量等。

Tf 包就是把某个点在某一个坐标系的坐标转换为另外一个坐标系的坐标，传感器可以看做一个坐标系，机器可以看做一个坐标系，障碍物可以看做一个点。

在【3.1.5】启动两个小乌龟后，执行下面操作。

3.3.1、tf 常用工具

1.view_frames 工具

能够监听当前时刻所有通过 ROS 广播的 tf 坐标系，并绘制出树状图表示坐标系之间的连接关系，生成名为 frame.pdf 文件，保存到本地当前位置。

```
rosrun tf view_frames
```

2.rqt_tf_tree 工具

虽然 view_frames 能够将当前坐标系关系保存在离线文件中，但是无法实时反映坐标关系，所以可以用 rqt_tf_tree 实时刷新显示坐标系关系

```
rosrun rqt_tf_tree rqt_tf_tree
```

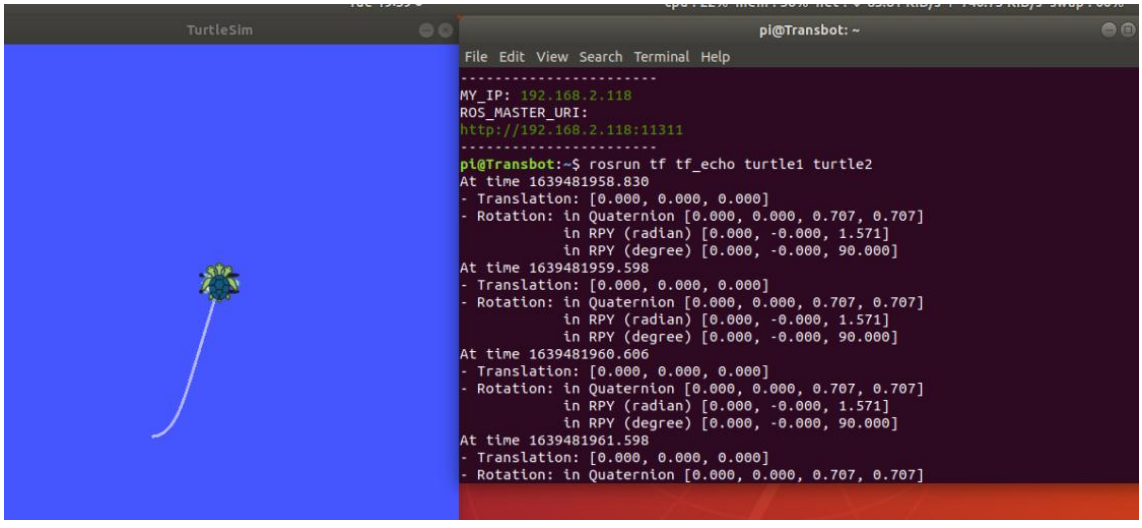
3.tf_echo 工具

使用 tf_echo 工具可以查看两个广播参考系之间的关系。

```
rosrun tf tf_echo <source_frame> <target_frame>
```

打印从 source_frame 到 target_frame 的旋转平移变换；例如：

```
rosrun tf tf_echo turtle1 turtle2
```



4.static_transform_publisher

发布两个坐标系之间的静态坐标变换，这两个坐标系不发生相对位置变化。

命令格式：

```
static_transform_publisher x y z yaw pitch roll frame_id child_frame_id
period_in_ms
static_transform_publisher x y z qx qy qz qw frame_id child_frame_id
period_in_ms
```

在 launch 中使用：

```
<launch>
<node pkg="tf" type="static_transform_publisher"
name="link1_broadcaster" args="1 0 0 0 0 0 1 link1_parent link1 100"
/>
</launch>
```

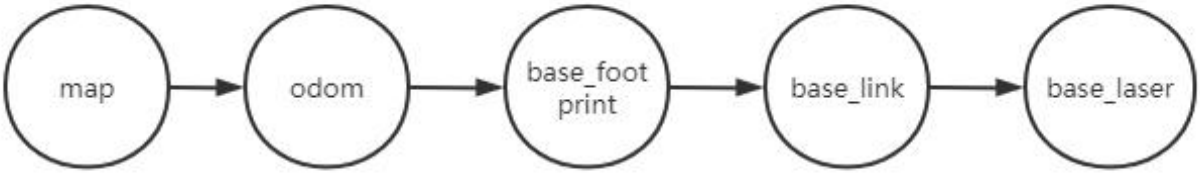
5.roswtf plugin

一个插件，分析你当前的 tf 配置并试图找出常见问题。

```
roswtf
```

3.3.2、常用的坐标系

常用的坐标系也就是frame_id，有map，odom，base_link，base_footprint，base_laser 等。



- 世界坐标(map)

该 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。

- 里程计坐标系(odom)

odom 是一个全局坐标系，通过里程计记录机器人当前的运动姿态。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，使得 odom 坐标系不能作为长期的全局参考。这里要区分开 odom topic，这是两个概念，一个是坐标系，一个是根据编码器（或者视觉等）计算的里程计。但是两者也有关系，odom topic 转化得位姿矩阵是 odom–>base_link 的 tf 关系。

odom 和 map 坐标系在机器人运动开始是重合的。但是，随着时间的推移

是不重合的，而出现的偏差就是里程计的累积误差。在一些校正传感器合作校正的 package 比如 amcl 会给出一个位置估计（localization），这可以得到 map−>base_link 的 tf，所以估计位置和里程计位置的偏差也就是 odom 与 map 的坐标系偏差。如果你的 odom 计算没有错误，那么 map−>odom 的 tf 就是 0。odom 坐标系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。

- 基座标(base_link)

机器人本体（基座）坐标系与机器人中心重合，坐标系原点一般为机器人的旋转中心。

base_footprint：原点为 base_link 原点在地面的投影，有些许区别（z 值不同）。

- 坐标之间的关系

在机器人系统中，我们使用一棵树来来关联所有坐标系，因此每个坐标系都有一个父坐标系和任意子坐标系，如下：map --> odom --> base_link 世界坐标系是 odom 坐标系的父，odom 坐标系是 base_link 的父。虽然直观来说，map 和 odom 应连接到 base_link，这是不允许的，因为每坐标系只能有一个父类。

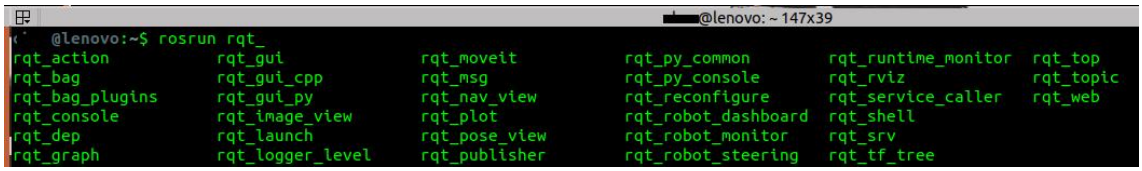
- 坐标系权限

odom 到 base_link 的转换是由里程计源计算和发布的。然而，定位模块不发布 map 到 base_link 的转换(transform)。相反，定位模块先接收 odom

到 base_link 的 transform，并使用这个信息发布 map 到 odom 的 transform。

3.4、rqt(QT 工具)

打开命令行窗口输入 `rosrun rqt` 然后双击 `Tab` 键，便可查看 ROS 中 QT 工具包含的内容,如下图所示：

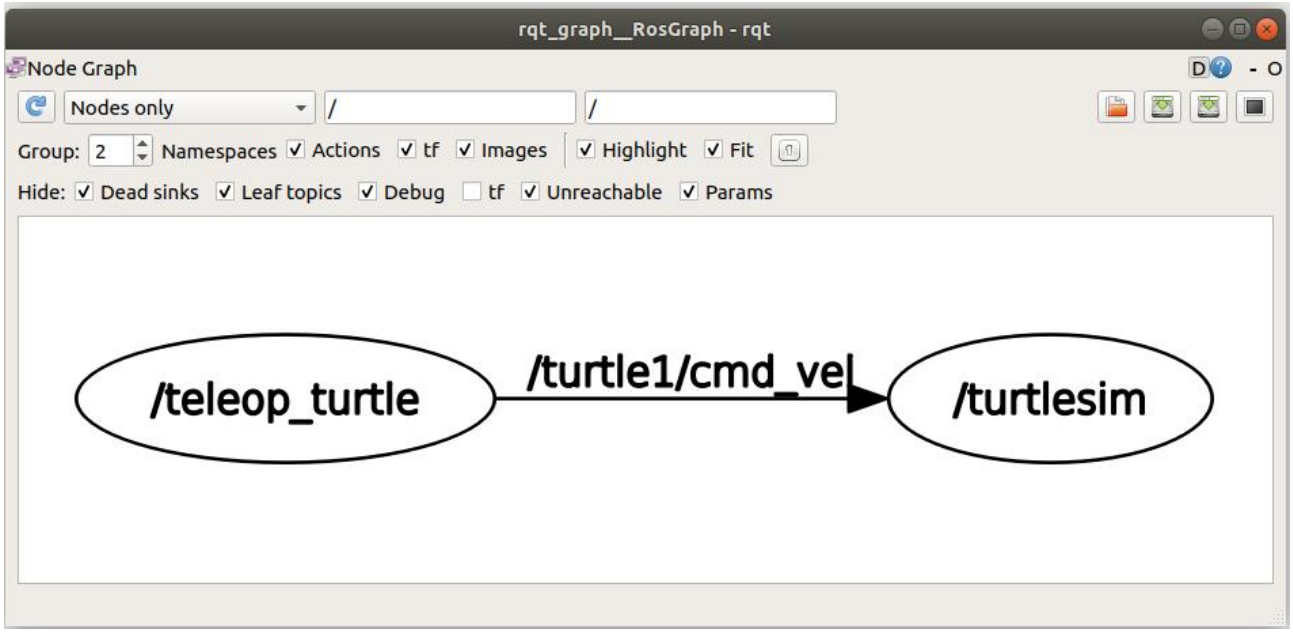


接下来我们以小海龟为例简单的介绍一下常用的几种 QT 工具：

1.rqt_graph 计算图可视化

打开命令行窗口输入以下命令，弹出一个对话窗。

```
rosrun rqt_graph rqt_graph
```



rqt_topic__TopicPlugin - rqt

Topic Monitor

Topic	Type	Bandwidth	Hz	Value
/attached_collision_object	moveit_msgs/AttachedCollisionObject			not monitored
/execute_trajectory/cancel	actionlib_msgs/GoalID			not monitored
/execute_trajectory/goal	moveit_msgs/ExecuteTrajectoryActionGoal			not monitored
/joint_states	sensor_msgs/JointState			not monitored
/move_group/cancel	actionlib_msgs/GoalID			not monitored
/move_group/goal	moveit_msgs/MoveGroupActionGoal			not monitored
/pickup/cancel	actionlib_msgs/GoalID			not monitored
/pickup/goal	moveit_msgs/PickupActionGoal			not monitored
/place/cancel	actionlib_msgs/GoalID			not monitored
/place/goal	moveit_msgs/PlaceActionGoal			not monitored
/planning_scene	moveit_msgs/PlanningScene			not monitored
/rosout	rosgraph_msgs/Log			not monitored
/rosout_agg	rosgraph_msgs/Log			not monitored
/trajectory_execution_event	std_msgs/String			not monitored
/turtle1/cmd_vel	geometry_msgs/Twist	987.06B/s	25.36	
angular	geometry_msgs/Vector3			
x	float64			0.0
y	float64			0.0
z	float64			0.0
linear	geometry_msgs/Vector3			
x	float64			2.0
y	float64			0.0
z	float64			0.0
/turtle1/color_sensor	turtlesim/Color	188.16B/s	62.50	
b	uint8			255
g	uint8			184
r	uint8			179
/turtle1/pose	turtlesim/Pose	1.25KB/s	62.50	
angular_velocity	float32			0.0
linear_velocity	float32			2.0
theta	float32			2.431999921798706
x	float32			3.320207118988037
y	float32			8.266461372375488

从图像中我们可以很清晰的看得出/teleop_turtle 节点通过 /turtle1/cmd_vel 话题给/turtlesim 节点进行数据传递。

/teleop_turtle 为具备 Publisher(发布)功能的节点。

/turtlesim 为具备 Subscriber(订阅)功能的节点。

/turtle1/cmd_vel 为 publisher 和 subscriber 通讯的话题。

2.rqt_topic 查看话题

```
rosrun rqt_topic rqt_topic
```

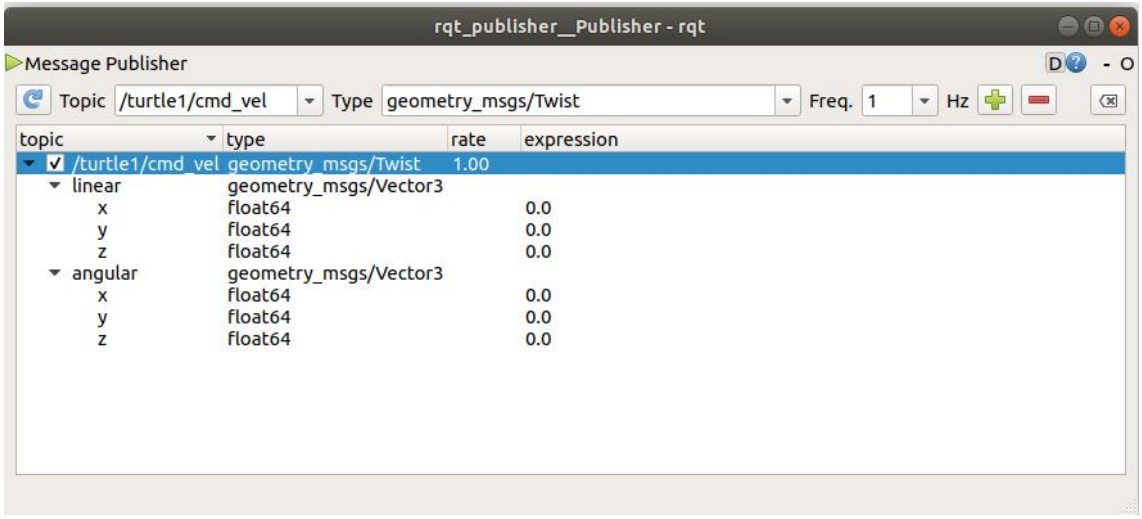
通过该工具，我们可以清楚地看到小海龟的一些实时变化的信息。

3.rqt_publisher

rqt_publisher 提供了一个 GUI 插件，用于发布具有固定或计算字段值的任意消息。打开命令行窗口输入以下命令,弹出一个对话窗。

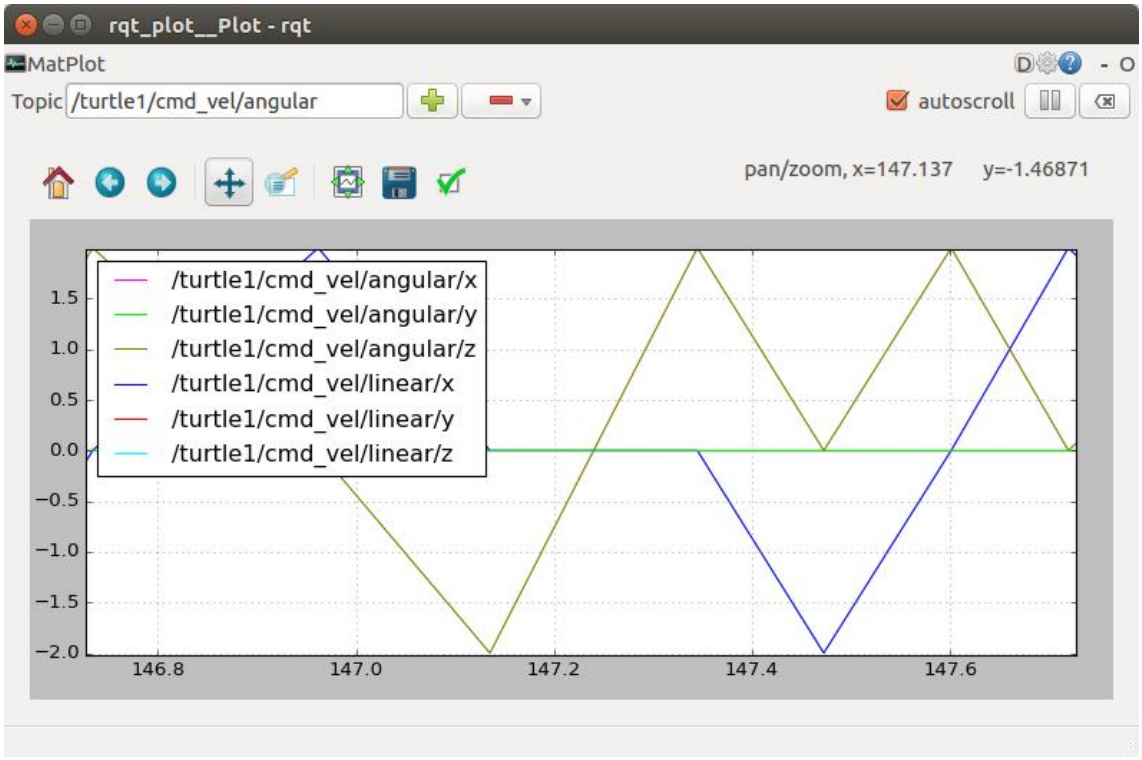
```
rosrun rqt_publisher rqt_publisher
```

点击 Topic 右边的选择框找到我们需要的/turtle1/cmd_vel 话题,点击右侧加号添加，显示如下：



4.rqt_plot 数据绘图

rosrun rqt_plot rqt_plot



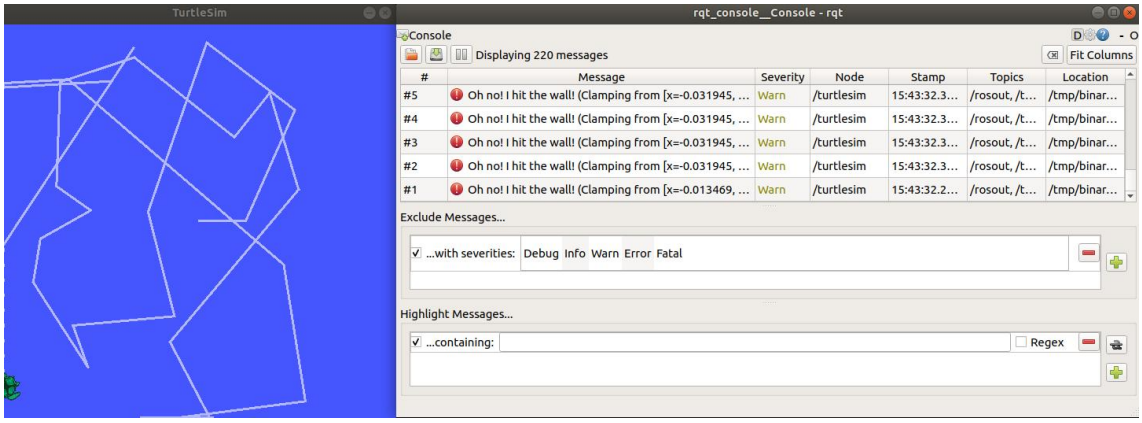
6.rqt_console 日志输出

ROS 日志（log）系统的功能是让程序生成一些日志消息，显示在屏幕上、发送到特定 topic 或者储存在特定 log 文件中，以方便调试、记录、报警等。

序号	等级	解析
1	DEBUG	调试日志，供开发测试使用
2	INFO	常规日志，用户可见级别的信息
3	WARN	警告信息。
4	ERROR	错误信息。程序出错后打印的信息
5	FATAL	致命错误。出现宕机的日志记录

ROS 中的日志消息按照严重性由低到高可以分为 5 级：DEBUG、INFO、WARN、ERROR、FATAL。只要程序可以运行就不需要注意，但 ERROR 和 FATAL 出现就表示程序存在着严重问题导致无法运行。

```
rosrun rqt_console rqt_console
```



日志输出工具属于 ROS 日志框架(logging framework)的一部分，用来显示节点的输出信息，从图上我们可以看出它再提示小海龟已经撞墙啦。

- 常用 API

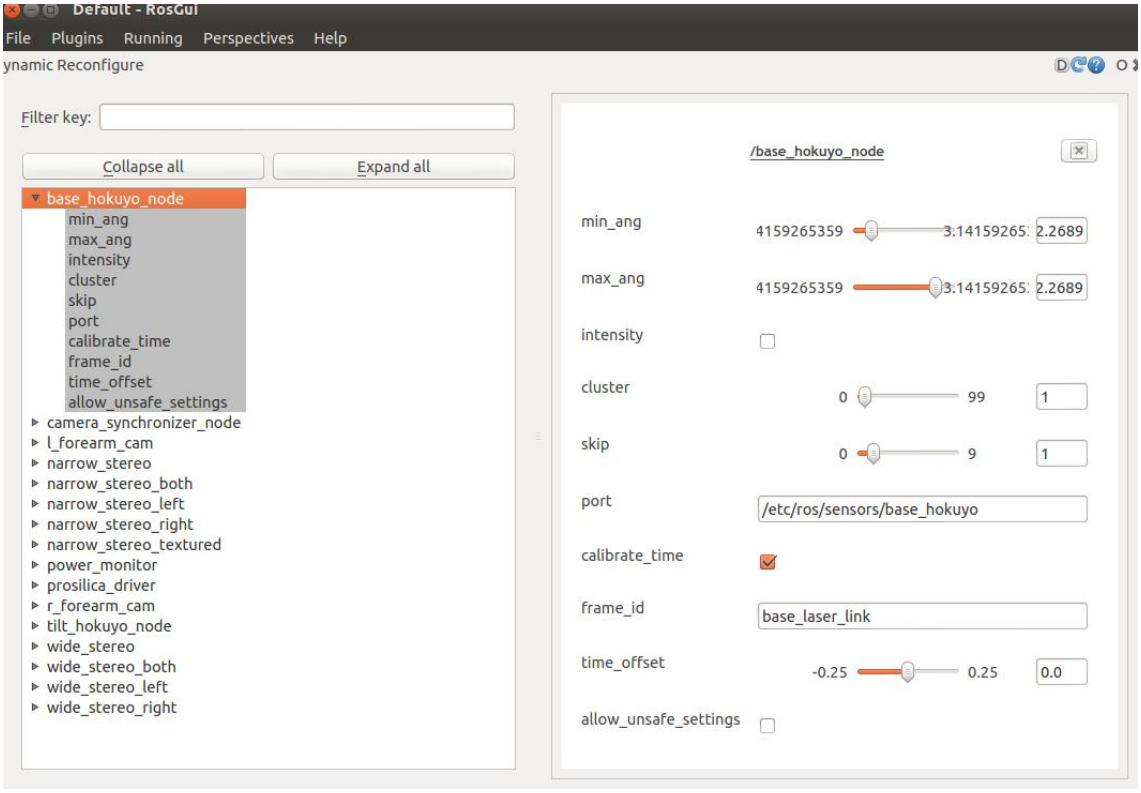
C++基础 API 格式	C++stream API 格式	Python 日志 API
ROS_DEBUG("打印的内容");	ROS_DEBUG_STREAM("打印的内容" << "hello");	rospy.logdebug("打印的内容")
ROS_INFO("打印的内容");	ROS_INFO_STREAM("打印的内容" << "hello");	rospy.loginfo("打印的内容")
ROS_WARN("打印的内容");	ROS_WARN_STREAM("打印的内容" << "hello");	rospy.logwarn("打印的内容")
ROS_ERROR("打印的内容");	ROS_ERROR_STREAM("打印的内容" << "hello");	rospy.logerror("打印的内容")
ROS_FATAL("	ROS_FATAL_STREAM("	rospy.logfatal("打

C++基础 API 格式	C++stream API 格式	Python 日志 API
打印的内容");	打印的内容" << "hello");	印的内容")

7.rqt_reconfigure 动态参数配置

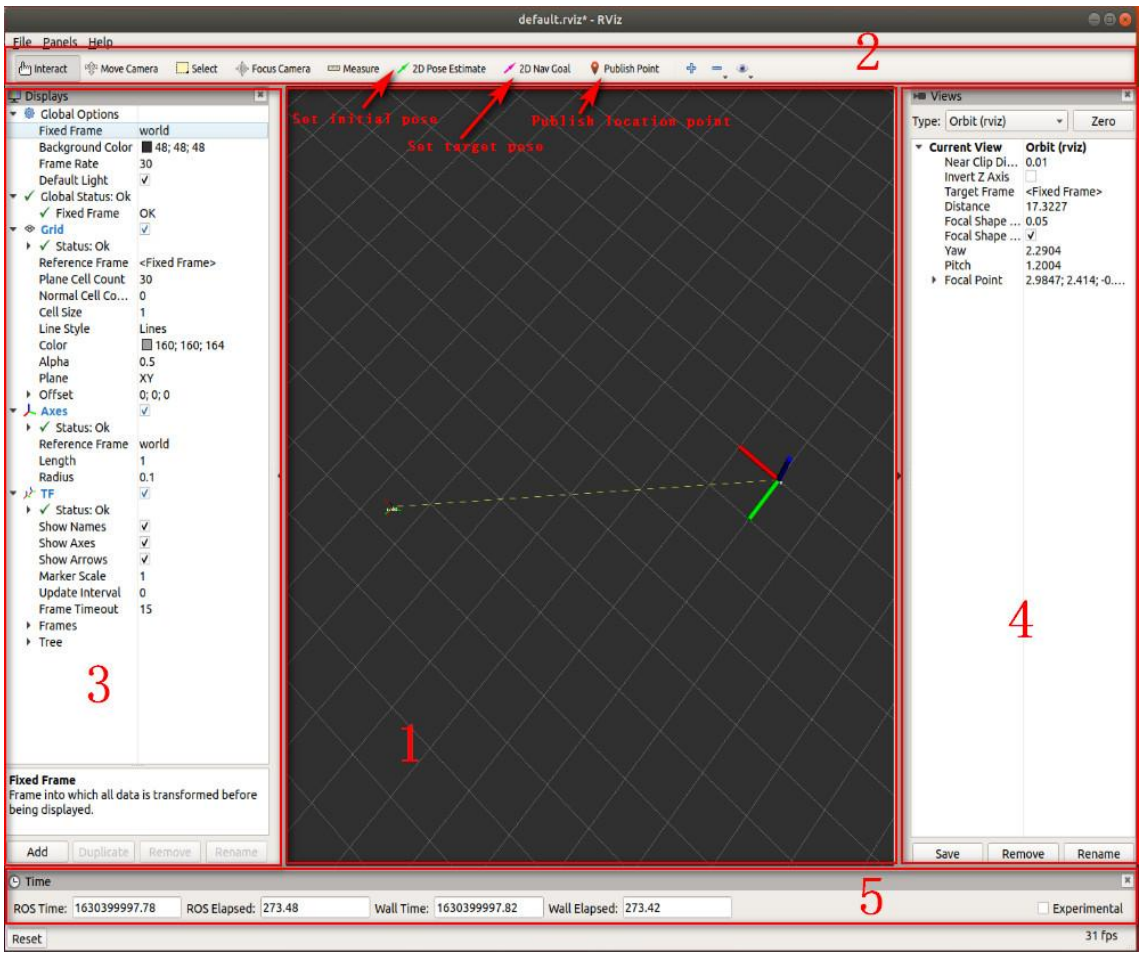
```
rosrun rqt_reconfigure rqt_reconfigure
```

图片来源 ROS wiki：



3.5、Rviz

rviz 是 ros 自带的一个图形化工具，可以方便的对 ros 的程序进行图形化操作。其使用也是比较简单。



【Set initial pose】、【Set target pose】、【Publish location point】：一般在建图导航时使用。

rviz 界面主要包含以下几个部分：

1：3D 视图区，用于可视化显示数据，目前没有任何数据，所以显示黑色。

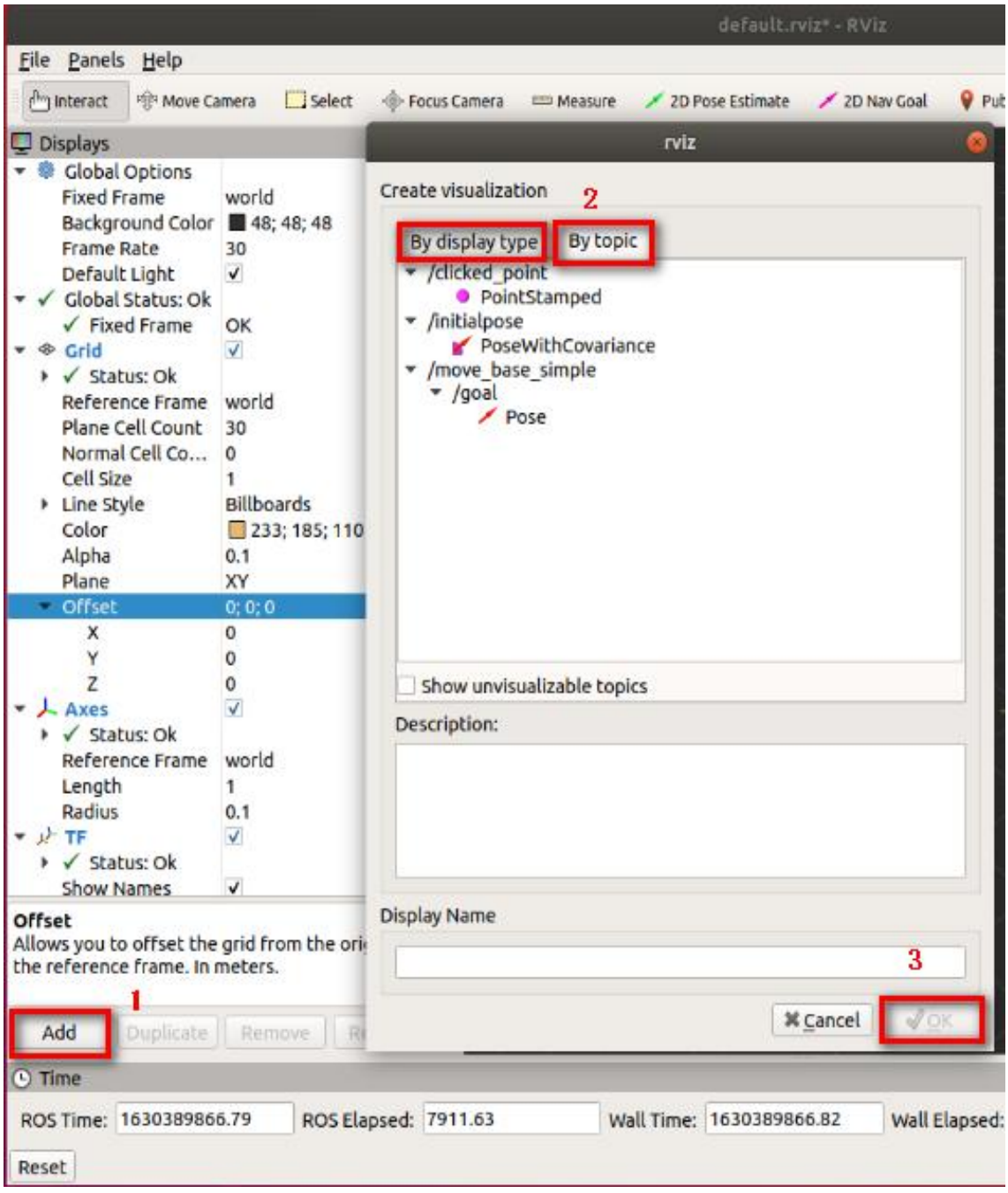
2：工具栏，提供视角控制、目标设置、发布地点等工具。

3：显示项列表，用于显示当前选择的显示插件，可以配置每个插件的属性。

4：视角设置区，可以选择多种观测视角。

5：时间显示区，显示当前的系统时间和 ROS 时间。

- 添加显示



第一步：点击【Add】按钮。会弹出一个选框。

第二步：可以通过显示类型【By display type】选择添加，不过需要自己修改对应话题，坐标系才可以显示出来；也可以通过选择话题【By topic】的方式，直接添加就可以正常显示。

第三步：点击【OK】即可。

3.6、ROS 常用命令

命令	作用
catin_create_pkg	创建功能包的信息
rospack	获取功能包的信息
catkin_make	编译工作空间中的功能包
rosdep	自动安装功能包依赖的其他包
roscd	功能包目录跳转
roscp	拷贝功能包中的文件
rosed	编辑功能包中的文件
rosrun	运行功能包中的可执行文件
roslaunch	运行启动文件

4、发布者

4、发布者 Publisher

4.1、发布者

发布者，顾名思义就是起到发布消息的作用。这里的消息，可以是下位机传给上位机的传感器信息，然后通过上位机打包封装好发送到订阅了该话题的订阅者；也可以是上位机的数据做了运算后打包封装好发送给给订阅该话题的订阅者。

4.2、 创建工作空间和话题功能包

4.2.1、创建工作空间

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/src
catkin_init_workspace
```

4.2.2、编译工作空间

```
cd ~/catkin_ws/
catkin_make
```

4.2.3、更新环境变量

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

4.2.4、检查环境变量

```
echo $ROS_PACKAGE_PATH
```

4.2.5、创建功能包

```
cd ~/catkin_ws/src
catkin_create_pkg learning_topic std_msgs rospy roscpp
geometry_msgs turtlesim
```

解释说明：learning_topic 为功能包的名字

4.2.6、编译功能包

```
cd ~/catkin_ws
catkin_make
source ~/catkin_ws/devel/setup.bash
```

4.3、创建一个发布者

4.3.1、创建步骤

1）、初始化 ROS 节点

2）、创建句柄

3）、向 ROS Master 注册节点信息，包括发布的话题名和话题中的消息类型以及队列长度

4）、创建并且初始化消息数据

5）、按照一定频率循环发送消息

4.3.2、C++语言实现

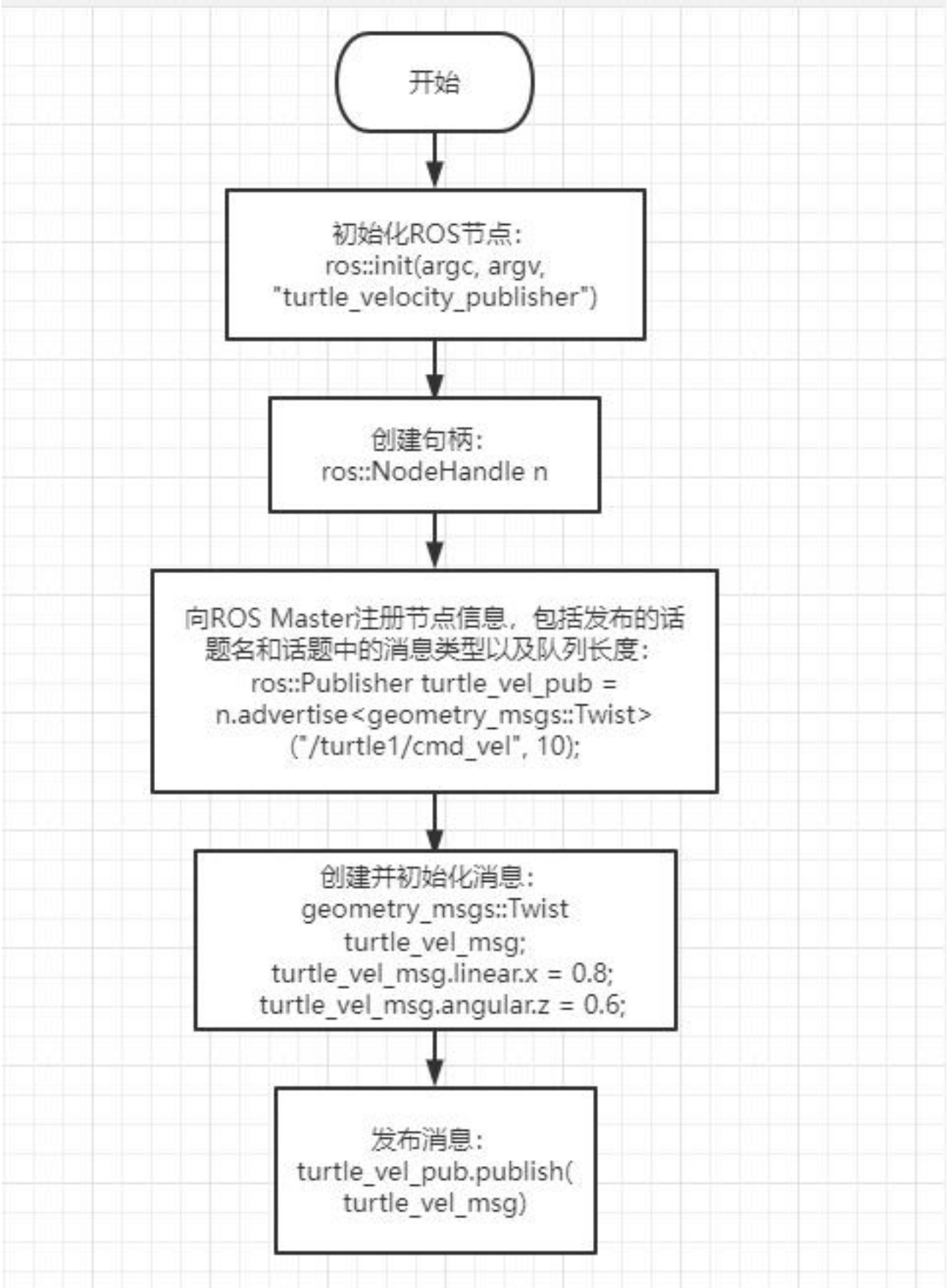
1）、在功能包的 src 文件夹,创建一个 c++文件（文件后缀为.cpp），命名为 turtle_velocity_publisher.cpp

2）、把下边的程序代码复制粘贴到 turtle_velocity_publisher.cpp 文件中

```
/*创建一个小海龟的速度发布者*/
#include <ros/ros.h>
#include <geometry_msgs/Twist.h>
int main(int argc, char **argv){

    ros::init(argc, argv, "turtle_velocity_publisher");//ROS 节点初始化

    ros::NodeHandle n;//这里是创建句柄

```

```
    //创建一个 Publisher，发布名为/turtle1/cmd_vel 的 topic，消息类型为 geometry_msgs::Twist，队列长度 10
    ros::Publisher turtle_vel_pub = n.advertise<geometry_msgs::Twist>("/turtle1/cmd_vel", 10);

    ros::Rate loop_rate(10);//设置循环的频率

    while (ros::ok()){
            //初始化需要发布的消息，类型需与 Publisher 一致
        geometry_msgs::Twist turtle_vel_msg;
        turtle_vel_msg.linear.x = 0.8;
        turtle_vel_msg.angular.z = 0.6;

        turtle_vel_pub.publish(turtle_vel_msg);// 发布速度消息

        //打印发布的速度内容
        ROS_INFO("Publsh turtle velocity command[%0.2f m/s, %0.2f rad/s]", turtle_vel_msg.linear.x, turtle_vel_msg.angular.z);

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

3）、程序流程图，可对应 1.3.1 内容查看


开始
初始化ROS节点：
ros::init(argc, argv,
"turtle_velocity_publisher")
创建句柄：
ros::NodeHandle n
向ROS Master注册节点信息，包括发布的话题名和话题中的消息类型以及队列长度：
ros::Publisher turtle_vel_pub =
n.advertise<geometry_msgs::Twist>
("/turtle1/cmd_vel", 10);
创建并初始化消息：
geometry_msgs::Twist
turtle_vel_msg;
turtle_vel_msg.linear.x = 0.8;
turtle_vel_msg.angular.z = 0.6;
发布消息：
turtle_vel_pub.publish(
turtle_vel_msg)

4）、在 CMakelist.txt 中配置，build 区域下，添加如下内容

```
add_executable(turtle_velocity_publisher
src/turtle_velocity_publisher.cpp)
target_link_libraries(turtle_velocity_publisher ${catkin_LIBRARIES})
```

5）、工作空间目录下编译代码

```
cd ~/catkin_ws
catkin_make
source devel/setup.bash        #需要配置环境变量，否则系统无法找到运行程序
```

6）、运行程序

运行 roscore

```
roscore
```

运行小海龟节点

```

```

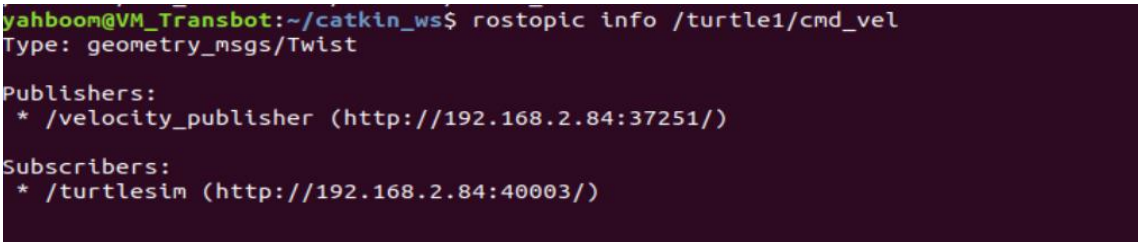
```
rosrun turtlesim turtlesim_node
```

运行发布者，持续给小海龟发送速度

```
rosrun learning_topic turtle_velocity_publisher
```

7）、运行效果截图

8）、程序运行说明

- 在终端输入 rostopic list 查看话题列表会发现/turtle1/cmd_vel 这个话题
- 我们用 rostopic info /turtle1/cmd_vel 查看会发现
- 

这说明小海龟是一个订阅/turtle1/cmd_vel 这个速度话题的，所以，发布者不断发送速度数据，小海龟接收到后，就开始按照速度做运动。

4.3.3、Python 语言实现

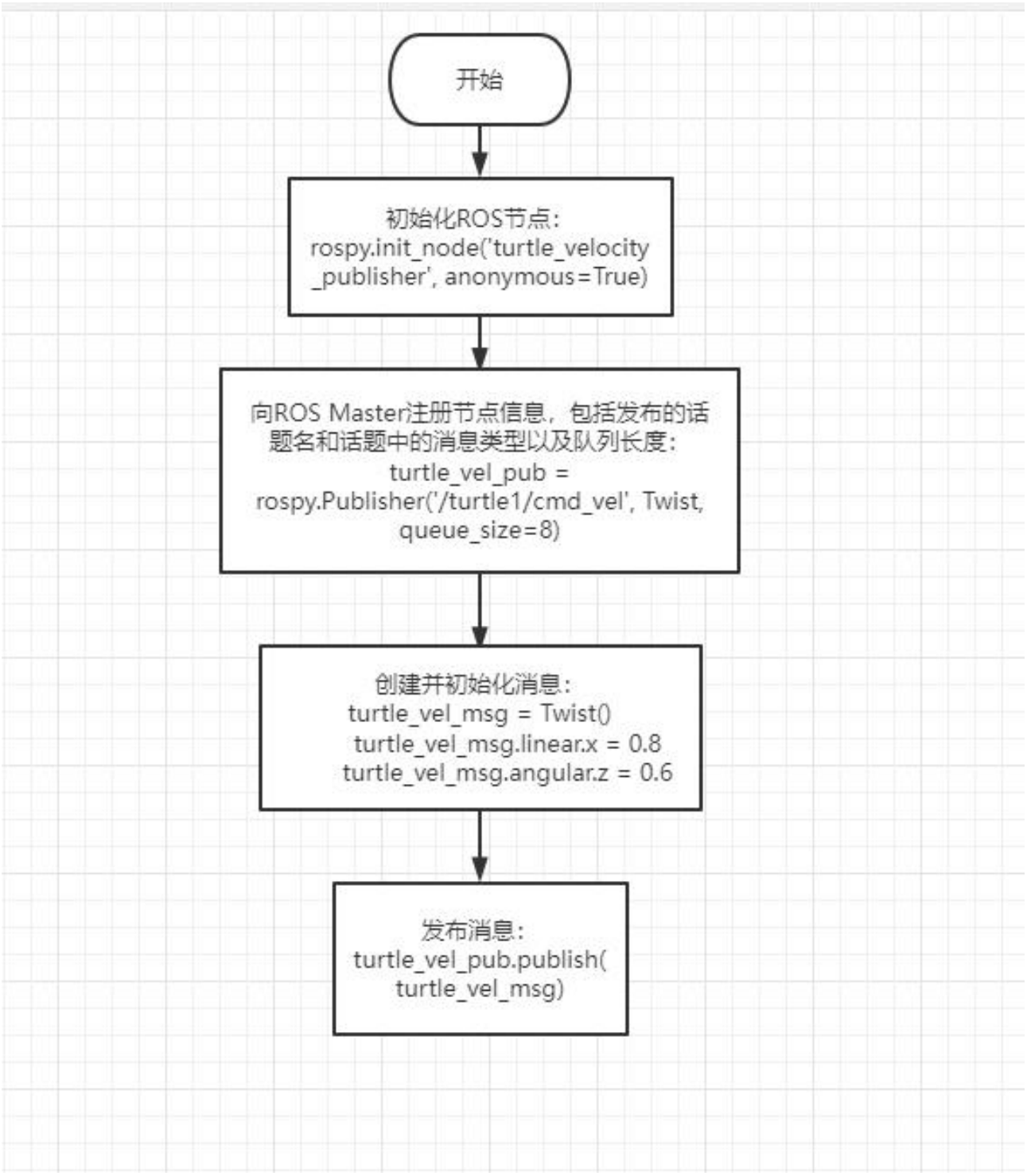
1）、在功能包目录下，新建以一个文件夹 scripts，然后在 scripts 文件夹下新建一个 python 文件（文件后缀.py），命名为 turtle_velocity_publisher.py

2）、把下边的程序代码复制粘贴到 turtle_velocity_publisher.py 文件中

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#  该例程将发布 turtle1/cmd_vel 话题，消息类型 geometry_msgs::Twist

import rospy
from geometry_msgs.msg import Twist

def turtle_velocity_publisher():

    rospy.init_node('turtle_velocity_publisher', anonymous=True) # ROS 节点初始化

    #  创建一个小海龟速度发布者，发布名为/turtle1/cmd_vel 的 topic，消息类型为 geometry_msgs::Twist，8 代表消息队列长度
```

```
    turtle_vel_pub = rospy.Publisher('/turtle1/cmd_vel', Twist,
queue_size=8)


    rate = rospy.Rate(10) #设置循环的频率

    while not rospy.is_shutdown():
            # 初始化 geometry_msgs::Twist 类型的消息
        turtle_vel_msg = Twist()
        turtle_vel_msg.linear.x = 0.8
        turtle_vel_msg.angular.z = 0.6

            # 发布消息
        turtle_vel_pub.publish(turtle_vel_msg)
            rospy.loginfo("linear is :%0.2f m/s, angular is :%0.2f rad/s",
                          turtle_vel_msg.linear.x, turtle_vel_msg.angular.z)


        rate.sleep()# 按照循环频率延时

if __name__ == '__main__':
    try:
        turtle_velocity_publisher()
    except rospy.ROSInterruptException:
        pass
```

3）、程序流程图



4）、运行程序

-

运行 roscore

-
-
-
-
-
- roscore
-
-
-
-

运行小海龟节点

-
-
-
-
-
- rosrun turtlesim turtlesim_node

-
-
-

  运行发布者，持续给小海龟发送速度

-
-

  ```
  
  ```

-
-
-
-

  ```
  rosrun learning_topic turtle_velocity_publisher.py
  ```

-
-

  注意：在运行前，需要给 turtle_velocity_publisher.py 添加可执行的权限，在 turtle_velocity_publisher.py 文件夹内打开终端，

  ```
  
  ```

  ```
  sudo chmod a+x turtle_velocity_publisher.py
  ```

  所有的 python 都需要添加执行文件权限，否则就会报错！

  5）、运行效果和程序说明参考 1.3.2。

5、订阅者

5、订阅者

5.1、订阅者

订阅者，接收发布者发布的数据，然后进入其回调函数，在回调函数里边处理接收到的数据。其核心内容是回调函数，每个订阅者订阅的话题都有回调函数。

5.2 创建一个订阅者

5.2.1、创建步骤

1）、初始化 ROS 节点

2）、创建句柄

3）、订阅需要的话题

4）、循环等待话题消息，接收到消息后进入回调函数

5）、在回调函数中完成消息处理。

5.2.2、C++语言实现

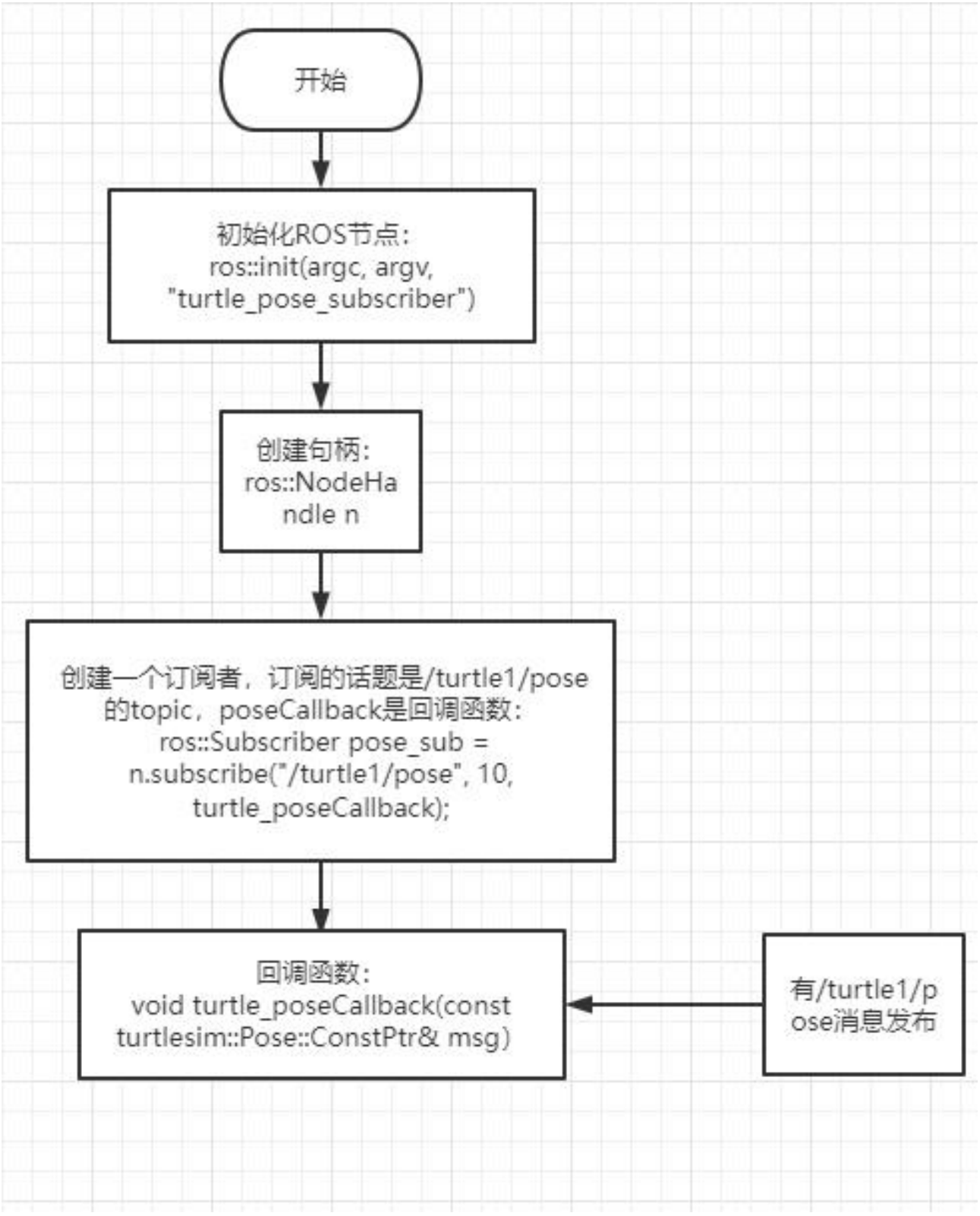
1）、在“发布者”教程中，创建的功能包的 src 文件夹下，新建一个 c++ 文件，命名为 turtle_pose_subscriber.cpp

2）、把下边的程序代码复制粘贴到 turtle_pose_subscriber.cpp 文件中

```
/*创建一个小海龟的当前位姿信息接收*/
#include <ros/ros.h>
#include "turtlesim/Pose.h"
// 接收消息后，会进入消息回调函数，回调函数里边会对接收到的数据进行处理
void turtle_poseCallback(const turtlesim::Pose::ConstPtr& msg){
    // 打印接收到的消息
    ROS_INFO("Turtle pose: x:%0.3f, y:%0.3f", msg->x, msg->y);
}

int main(int argc, char **argv){

    ros::init(argc, argv, "turtle_pose_subscriber");// 初始化 ROS 节点
```

```
    ros::NodeHandle n;//这里是创建句柄

    // 创建一个订阅者，订阅的话题是/turtle1/pose 的 topic，
poseCallback 是回调函数
    ros::Subscriber pose_sub = n.subscribe("/turtle1/pose", 10,
turtle_poseCallback);

    ros::spin(); // 循环等待回调函数

    return 0;
}
```

3）、程序流程图，可对应 5.2.1 内容查看



4）、在 CMakelist.txt 中配置，build 区域下，添加如下内容

```
add_executable(turtle_pose_subscriber
src/turtle_pose_subscriber.cpp)
target_link_libraries(turtle_pose_subscriber ${catkin_LIBRARIES})
```

5）、工作空间目录下编译代码

```
cd ~/catkin_ws
catkin_make
source devel/setup.bash        #需要配置环境变量，否则系统无法找到运行程序
```

6）、运行程序

运行 roscore

```
roscore
```

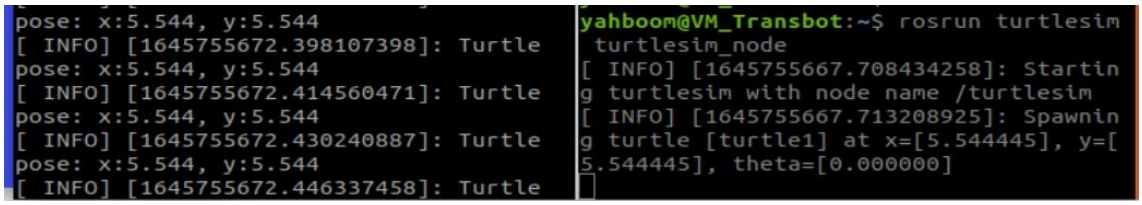
- 运行小海龟节点

```
rosrun turtlesim turtlesim_node
```

- 运行订阅，持续接收小海龟发送位姿数据

```
rosrun learning_topic turtle_pose_subscriber
```

7）、运行截图



8）、程序运行说明

- 在运行小海龟的节点后，小海龟会不断的发送自身的位姿信息，发送的话题名字是

  /turtle1/pose

- 而 turtle_pose_subscriber 运行后，它会接收小海龟发送过来的数据消息，然后在回调函数里边把这些信息打印出来。

5.2.3、python 语言实现

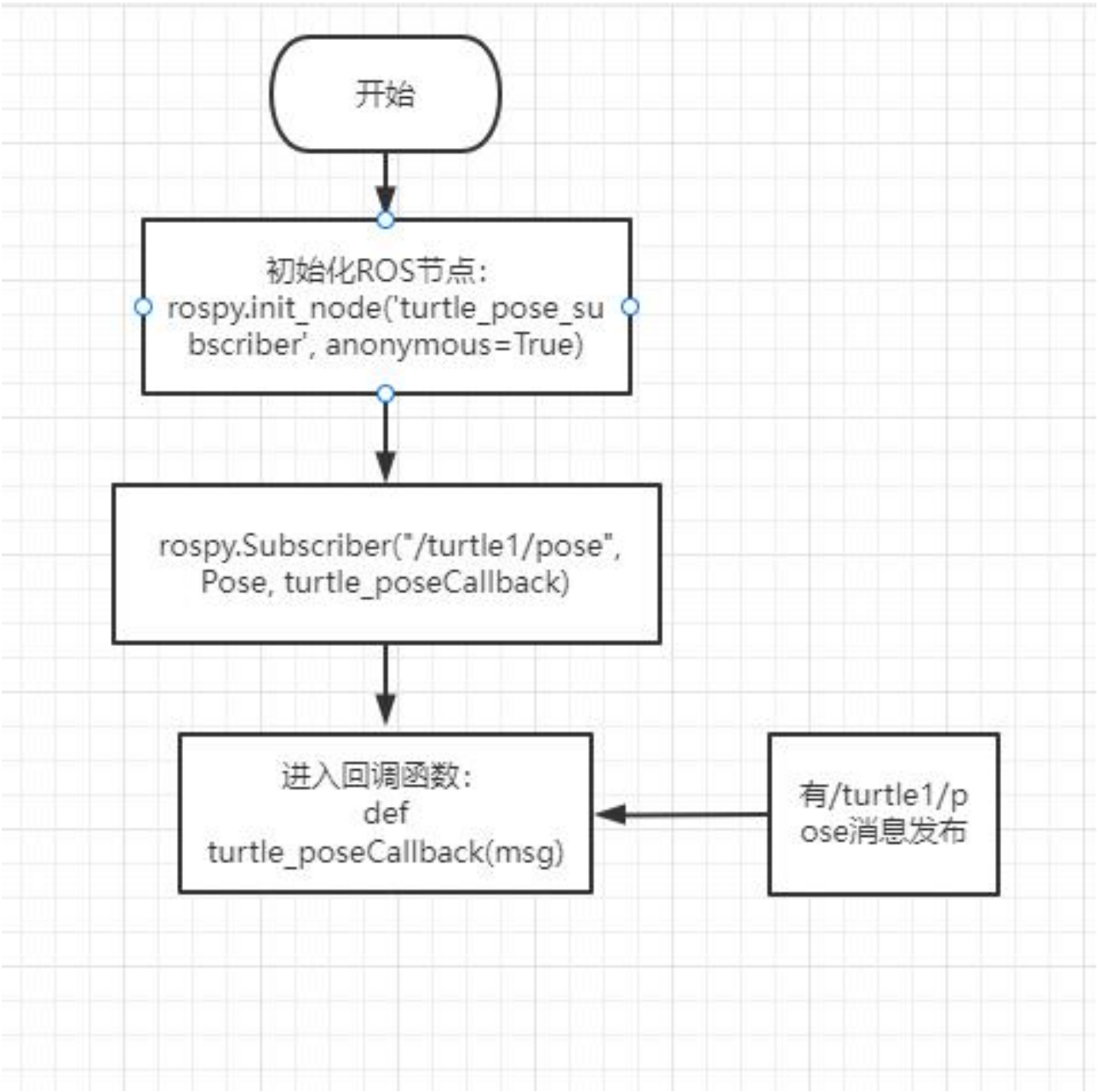
1）、在功能包目录下，新建以一个文件夹 scripts，然后在 scripts 文件夹下新建一个 python 文件（文件后缀.py），命名为 turtle_pose_subscriber.py

2）、把下边的程序代码复制粘贴到 turtle_pose_subscriber.py 文件中

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

import rospy
from turtlesim.msg import Pose

def poseCallback(msg):
```

```
    rospy.loginfo("Turtle pose: x:%0.3f, y:%0.3f", msg.x, msg.y)

def turtle_pose_subscriber():

    rospy.init_node('turtle_pose_subscriber', anonymous=True)# ROS
节点初始化

     # 创建一个 Subscriber，订阅名为/turtle1/pose 的 topic，注册回调函
数 poseCallback
    rospy.Subscriber("/turtle1/pose", Pose, poseCallback)


    rospy.spin()# 循环等待回调函数

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

3）、程序流程图



4）、运行程序

-

  运行 roscore

-

-

-
-
-
- roscore
-
-
- 运行小海龟节点
-
-
-
-
- rosrun turtlesim turtlesim_node
-
-
-

运行订阅者，持续接收小海龟发送位姿数据

-
-
-
-
-
- rosrun learning_topic turtle_pose_subscriber.py
-
-

5）、运行效果和程序说明参考 5.2.2。

6、自定义话题消息与使用

6、自定义话题消息与使用

6.1、自定义话题消息

切换至~/catkin_ws/src/learning_topic 功能包目录下，然后新建一个文件夹，命名为 msg，用来存放自定义的话题消息。

6.1.1、定义 msg 文件

切换至 msg 目录下，新建一个空白的 msg 文件，以 msg 为后缀表示代表是 msg 文件。这里我们以 Information.msg 为例子说明下，把以下代码复制到刚刚创建好的 msg 文件里边。

```
string company
string city
```

6.1.2、在 package.xml 中添加功能包依赖

```
<build_depend>message_generation</build_depend>
<exec_depend>message_runtime</exec_depend>
```

6.1.3、在 CMakeLists.txt 添加编译选项

```
在 find_package 里边加上 message_generation
add_message_files(FILES Information.msg)
generate_messages(DEPENDENCIES std_msgs)
```

6.1.4、编译生成语言相关文件

```
cd ~/catkin_ws
catkin_make
```

6.1.5、C++语言实现

1）、切换至~/catkin_ws/src/learning_topic/src 下，新建两个 cpp 文件，命名为 Information_publisher.cpp 和 Information_subscriber.cpp，把以下代码分别复制到里边，

Information_publisher.cpp

```
/**
 * 该例程将发布/company_info 话题，消息类型是自定义的 learning_topic::Information
 */

#include <ros/ros.h>
#include "learning_topic/Information.h"

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

```
    // 创建节点句柄
    ros::NodeHandle n;

    // 创建一个 Publisher，发布名为/company_info 的 topic，消息类型为
learning_topic::Person，队列长度 10
    ros::Publisher Information_pub =
n.advertise<learning_topic::Information>("/company_info", 10);

    // 设置循环的频率
    ros::Rate loop_rate(1);

    int count = 0;
    while (ros::ok())
    {
        // 初始化 learning_topic::Information 类型的消息
          learning_topic::Information info_msg;
          info_msg.company = "Yahboom";
          info_msg.city = "Shenzhen";

        // 发布消息
          Information_pub.publish(info_msg);

         ROS_INFO("Information: company:%s    city:%s ",
                          info_msg.company.c_str(), info_msg.city.c_str());
```

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

    return 0;
}
```

Information_subscriber.cpp

```
/**
 * 该例程将订阅/company_info 话题，自定义消息类型
learning_topic::Information
 */

#include <ros/ros.h>
#include "learning_topic/Information.h"

// 接收到订阅的消息后，会进入消息回调函数处理数据
void CompanyInfoCallback(const
learning_topic::Information::ConstPtr& msg)
{
    // 打印接收到的消息
```

```
    ROS_INFO("This is: %s    in %s", msg->company.c_str(),
msg->city.c_str());
}

int main(int argc, char **argv)
{

    ros::init(argc, argv, "company_Information_subscriber")
```

7、客户端

7、客户端

在 ROS 通讯中，除了话题通讯，还有一种就是服务通讯。服务包括客户端和服务端，客户端就是请求服务，服务端就是提供服务的。本节就以客户端为主要内容，说 c++和 python 如何实现客户端。

7.1、准备工作

7.1.1、建立功能包

1）、切换至~/catkin_ws/src 目录下，

```
catkin_create_pkg learning_server std_msgs rospy roscpp
geometry_msgs turtlesim
```

2）、切换至~/catkin_ws 目录下，

```
catkin_make
```

7.2、C++语言实现

7.2.1、实现步骤

1）、初始化 ROS 节点

2）、创建句柄

3）、创建一个 Client 实例

4）、初始化并发布服务请求数据

5）、等待 Server 处理之后的应答结果

7.2.2、切换至~/catkin_ws/src/learning_server/src 目录下，新建一个.cpp 文件，命名为 a_new_turtle,把下边代码粘贴在里边

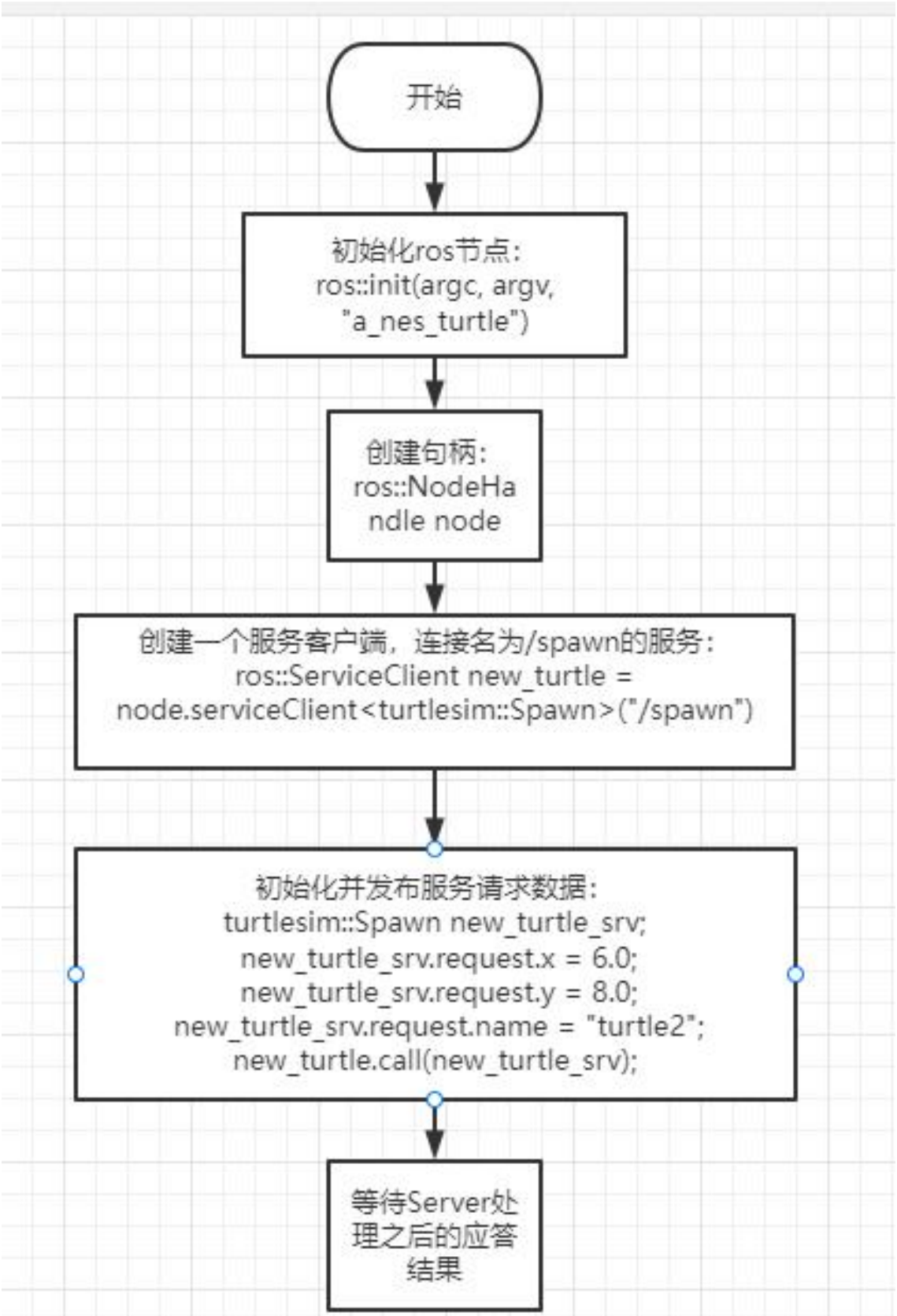
a_new_turtle.cpp

```
/**
 * 该例程将请求小海龟节点里的/spawn 服务，会在规定的位置出现一只新的小海龟
 */

```

```
#include <ros/ros.h>
#include <turtlesim/Spawn.h>

int main(int argc, char** argv)
{

    ros::init(argc, argv, "a_nes_turtle");// 初始化 ROS 节点

    ros::NodeHandle node;

    ros::service::waitForService("/spawn"); // 等待/spawn 服务

    ros::ServiceClient new_turtle =
node.serviceClient<turtlesim::Spawn>("/spawn");//创建一个服务客户
端，连接名为/spawn 的服务

   // 初始化 turtlesim::Spawn 的请求数据
    turtlesim::Spawn new_turtle_srv;
    new_turtle_srv.request.x = 6.0;
    new_turtle_srv.request.y = 8.0;
    new_turtle_srv.request.name = "turtle2";

   // 请求服务传入 xy 位置参数以及名字参数
    ROS_INFO("Call service to create a new turtle name is %s,at the
x:%.1f,y:%.1f", new_turtle_srv.request.name.c_str(),
```

```
            new_turtle_srv.request.x,
            new_turtle_srv.request.y);

    new_turtle.call(new_turtle_srv);


    ROS_INFO("Spwan turtle successfully [name:%s]",
new_turtle_srv.response.name.c_str());// 显示服务调用结果

    return 0;
};
```

1）、程序流程图


开始
初始化ros节点：
ros::init(argc, argv,
"a_nes_turtle")
创建句柄：
ros::NodeHa
ndle node
创建一个服务客户端，连接名为/spawn的服务：
ros::ServiceClient new_turtle =
node.serviceClient<turtlesim::Spawn>("/spawn")
初始化并发布服务请求数据：
turtlesim::Spawn new_turtle_srv;
new_turtle_srv.request.x = 6.0;
new_turtle_srv.request.y = 8.0;
new_turtle_srv.request.name = "turtle2";
new_turtle.call(new_turtle_srv);
等待Server处
理之后的应答
结果

2）、在 CMakelist.txt 中配置，build 区域下，添加如下内容

```
add_executable(a_new_turtle src/a_new_turtle.cpp)
target_link_libraries(a_new_turtle ${catkin_LIBRARIES})
```

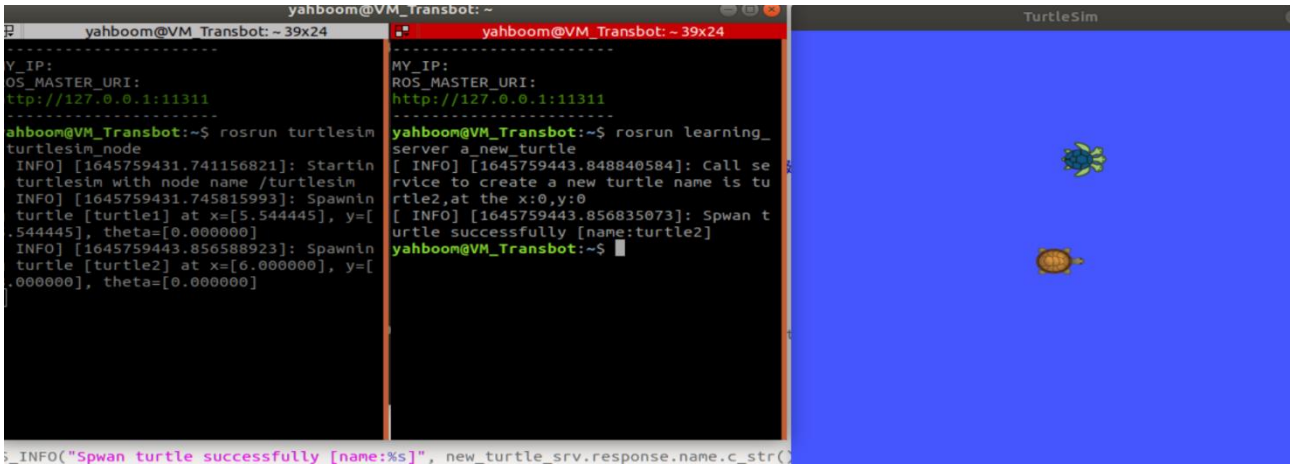
3）、工作空间目录下编译代码

```
cd ~/catkin_ws
catkin_make
source devel/setup.bash        #需要配置环境变量，否则系统无法找到运行程序
```

4）、运行程序

```
roscore
rosrun turtlesim turtlesim_node
rosrun learning_server a_new_turtle
```

5）、运行效果截图



6）、程序说明

在启动小海龟的节点后，再运行 a_new_turtle 这个程序会发现，画面中会出现另外一直小海龟，这是因为小海龟的节点提供了服务/spawn，该服务会产生另外一直小海龟 turtle2，查看小海龟提供的服务可以通过 rosservice list 命令来查看，如下图所示

```
yahboom@VM_Transbot:~/catkin_ws/src/learning_server/src$ rosservice list
/clear
/kill
/reset
/rosout/get_loggers
/rosout/set_logger_level
/spawn
/turtle1/set_pen
/turtle1/teleport_absolute
/turtle1/teleport_relative
/turtle2/set_pen
/turtle2/teleport_absolute
/turtle2/teleport_relative
/turtle6/set_pen
/turtle6/teleport_absolute
/turtle6/teleport_relative
/turtlesim/get_loggers
/turtlesim/set_logger_level
```

可以通过 rosservice info /spawn，查看这个服务需要的参数，如下图所示

```
yahboom@VM_Transbot:~/catkin_ws/src/learning_server/src$ rosservice info /spawn
Node: /turtlesim
URI: rosrpc://192.168.2.85:57303
Type: turtlesim/Spawn
Args: x y theta name
```

可以看出需要有 4 个参数：x，y，theta，name，这四个参数在

a_new_turtle.cpp 里边有初始化

```
srv.request.x = 6.0;
srv.request.y = 8.0;
srv.request.name = "turtle2";
```

注意：theta 没有赋值，默认为 0

7.3、python 语言实现

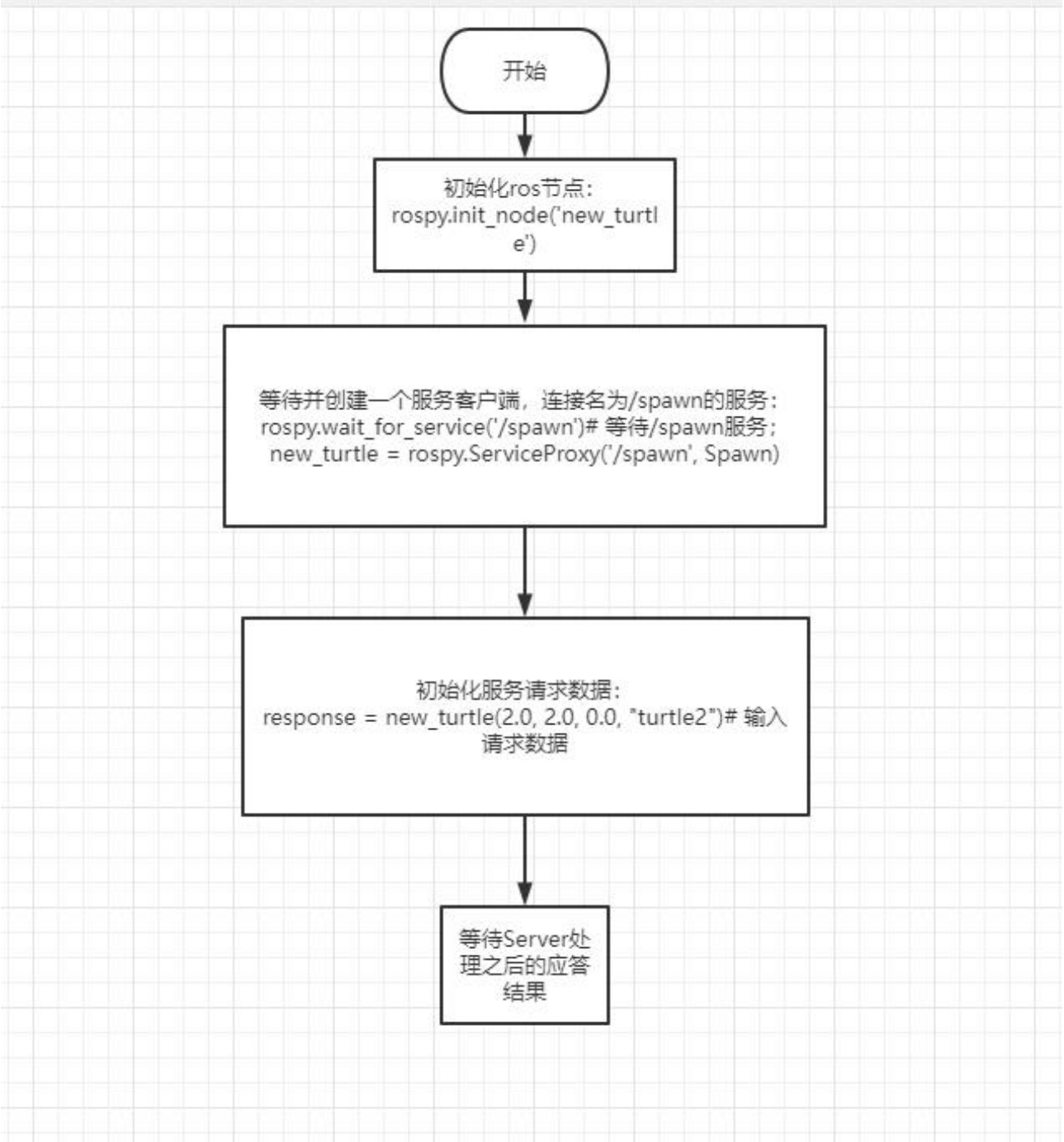
7.3.1、切换至~/catkin_ws/src/learning_server 目录下，新建一个 script 文件夹，切进去，新建一个 py 文件，命名为 a_new_turtle,把下边代码粘贴在里边

a_new_turtle.py

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

import sys
import rospy
from turtlesim.srv import Spawn

def turtle_spawn():
```

```
    rospy.init_node('new_turtle')# ROS 节点初始化

    rospy.wait_for_service('/spawn')# 等待/spawn 服务

    try:
        new_turtle = rospy.ServiceProxy('/spawn', Spawn)


        response = new_turtle(2.0, 2.0, 0.0, "turtle2")# 输入请求数据
        return response.name
    except rospy.ServiceException, e:
        print "failed to call service : %s"%e

if __name__ == "__main__":
    #服务调用并显示调用结果
    print "a new turtle named %s." %(turtle_spawn())
```

1）、程序流程图



2）、运行程序

```
roscore
rosrun turtlesim turtlesim_node
```

```
rosrun learning_server a_new_turtle.py
```

3)、程序运行效果和程序说明与 C++实现的效果一致，这里主要说下 python 如何提供服务需要的参数，

```

 response = add_turtle(2.0, 2.0, 0.0, "turtle2")
```

对应的参数，分别是 x，y，theta，name。

服务端

8、服务端

上节课我们说到了客户端请求服务，然后服务端提供服务，这节课我们就来说说服务端是如何实现提供服务的。

8.1、C++语言实现

8.1.1、实现步骤

1）、初始化 ROS 节点

2）、创建 Server 实例

3）、循环等待服务请求，进入回调函数

4）、在回调函数中完成服务的功能处理，并且反馈应答数据

8.1.2、切换至~/catkin_ws/src/learning_server/src 目录下，新建一个.cpp 文件，命名为 turtle_vel_command_server,把下边代码粘贴在里边

turtle_vel_command_server.cpp

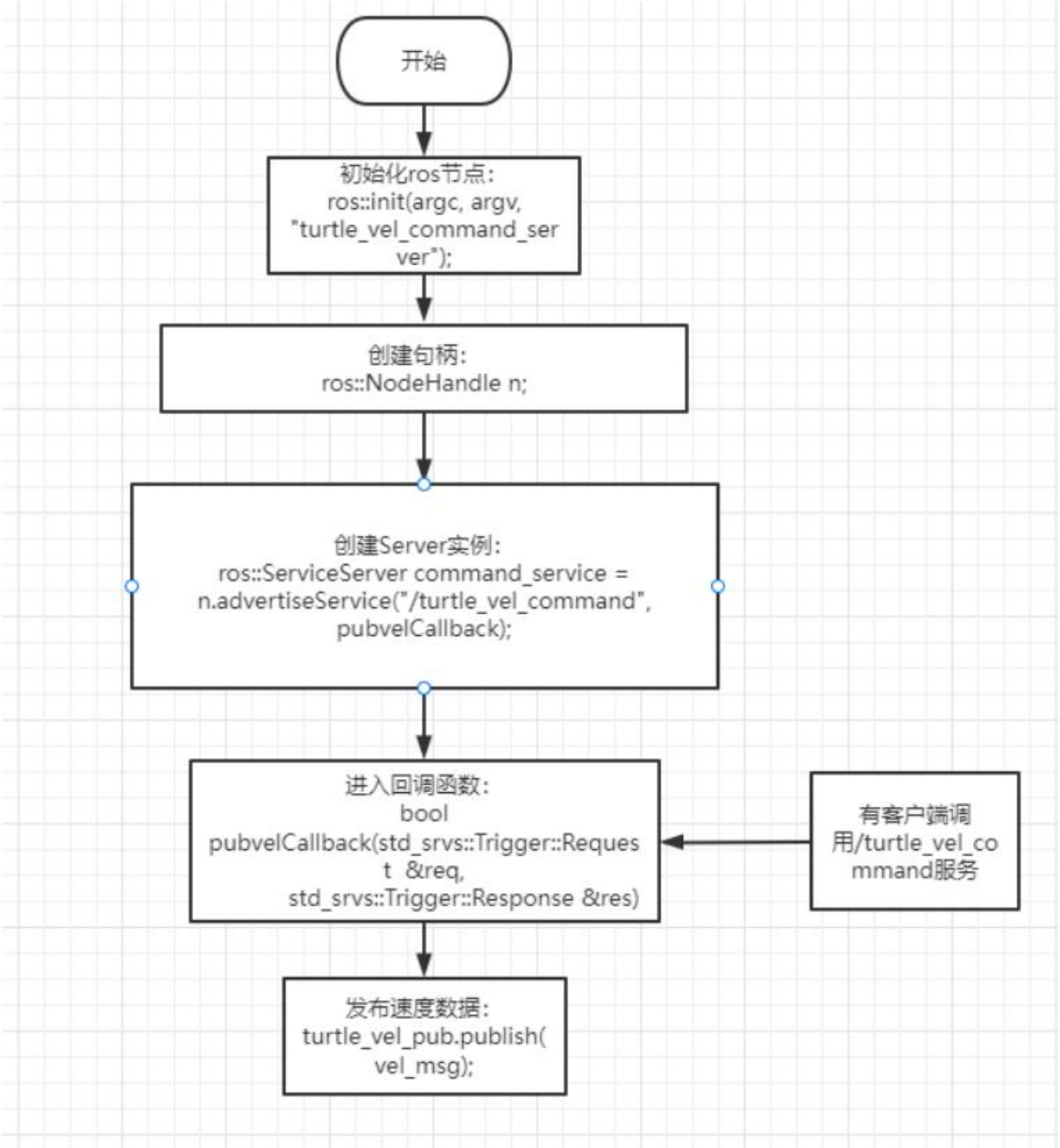
```
/**
 * 该例程将执行/turtle_vel_command 服务，服务数据类型 std_srvs/Trigger
 */

#include <ros/ros.h>
#include <geometry_msgs/Twist.h>
#include <std_srvs/Trigger.h>

ros::Publisher turtle_vel_pub;
bool pubvel = false;

// service 回调函数，输入参数 req，输出参数 res
bool pubvelCallback(std_srvs::Trigger::Request  &req,
                    std_srvs::Trigger::Response &res)
{
```

```
    pubvel = !pubvel;

        ROS_INFO("Do you want to publish the vel?: [%s]",
pubvel==true?"Yes":"No");// 打印客户端请求数据

    // 设置反馈数据
    res.success = true;
    res.message = "The status is changed!";

    return true;
}

int main(int argc, char **argv)
{

    ros::init(argc, argv, "turtle_vel_command_server");


    ros::NodeHandle n;

    // 创建一个名为/turtle_vel_command 的 server，注册回调函数
pubvelCallback
```

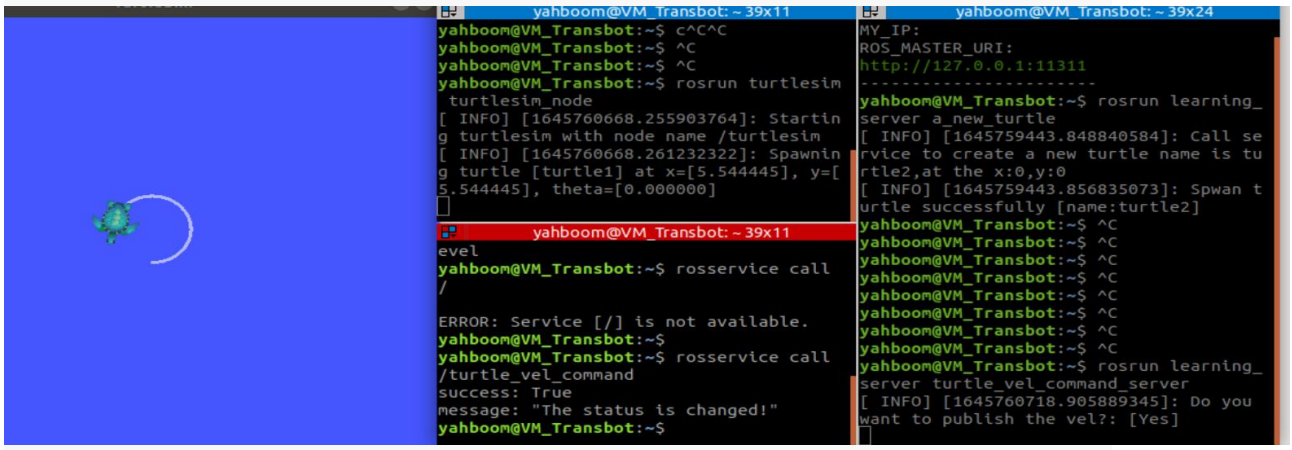
```
    ros::ServiceServer command_service =
n.advertiseService("/turtle_vel_command", pubvelCallback);

    // 创建一个 Publisher，发布名为/turtle1/cmd_vel 的 topic，消息类
型为 geometry_msgs::Twist，队列长度 8
    turtle_vel_pub =
n.advertise<geometry_msgs::Twist>("/turtle1/cmd_vel", 8);

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

    while(ros::ok())
    {

        ros::spinOnce();// 查看一次回调函数队列

        // 判断 pubvel 为 True，则发布小海龟速度指令
        if(pubvel)
        {
            geometry_msgs::Twist vel_msg;
            vel_msg.linear.x = 0.6;
            vel_msg.angular.z = 0.8;
            turtle_vel_pub.publish(vel_msg);
```

```
        }

        loop_rate.sleep();//按照循环频率延时
    }

    return 0;
}
```

1）、程序流程图



2）、在 CMakelist.txt 中配置，build 区域下，添加如下内容




```
add_executable(turtle_vel_command_server
src/turtle_vel_command_server.cpp)
target_link_libraries(turtle_vel_command_server ${catkin_LIBRARIES})
```

3）、工作空间目录下编译代码

```
cd ~/catkin_ws
catkin_make
source devel/setup.bash        #需要配置环境变量，否则系统无法找到运行程序
```

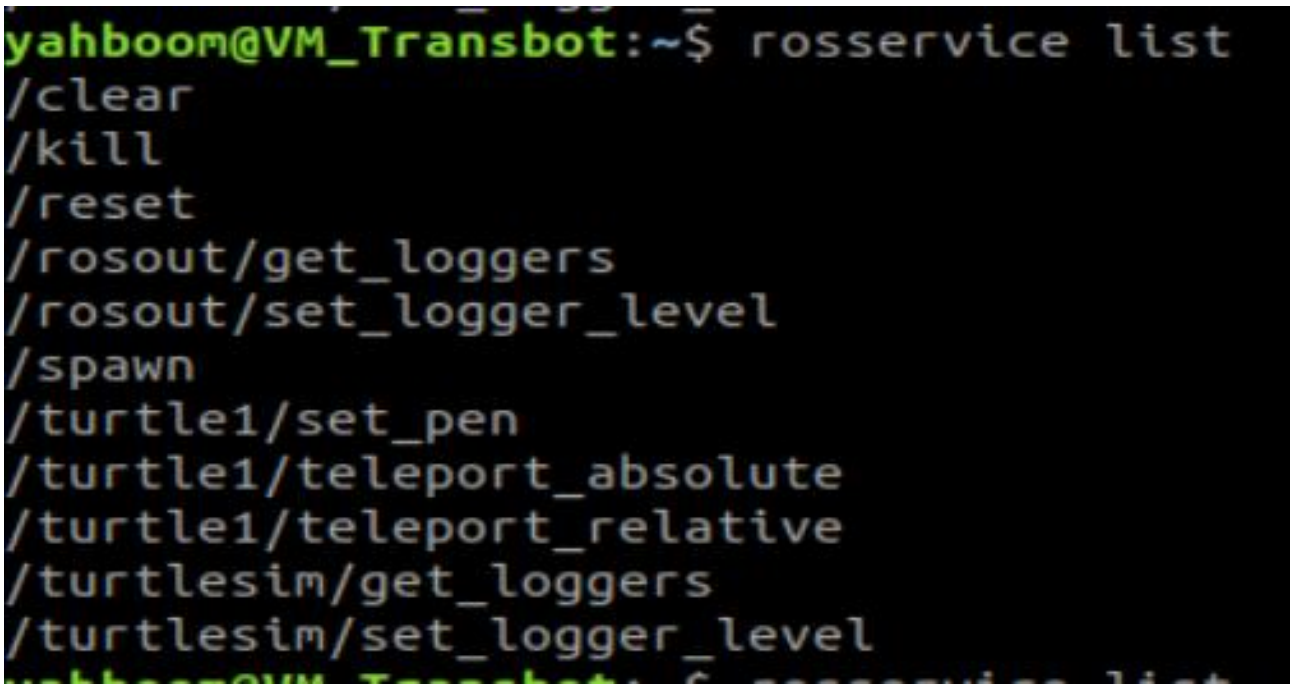
4）、运行程序

```
roscore
rosrun turtlesim turtlesim_node
rosrun learning_server turtle_vel_command_server
```

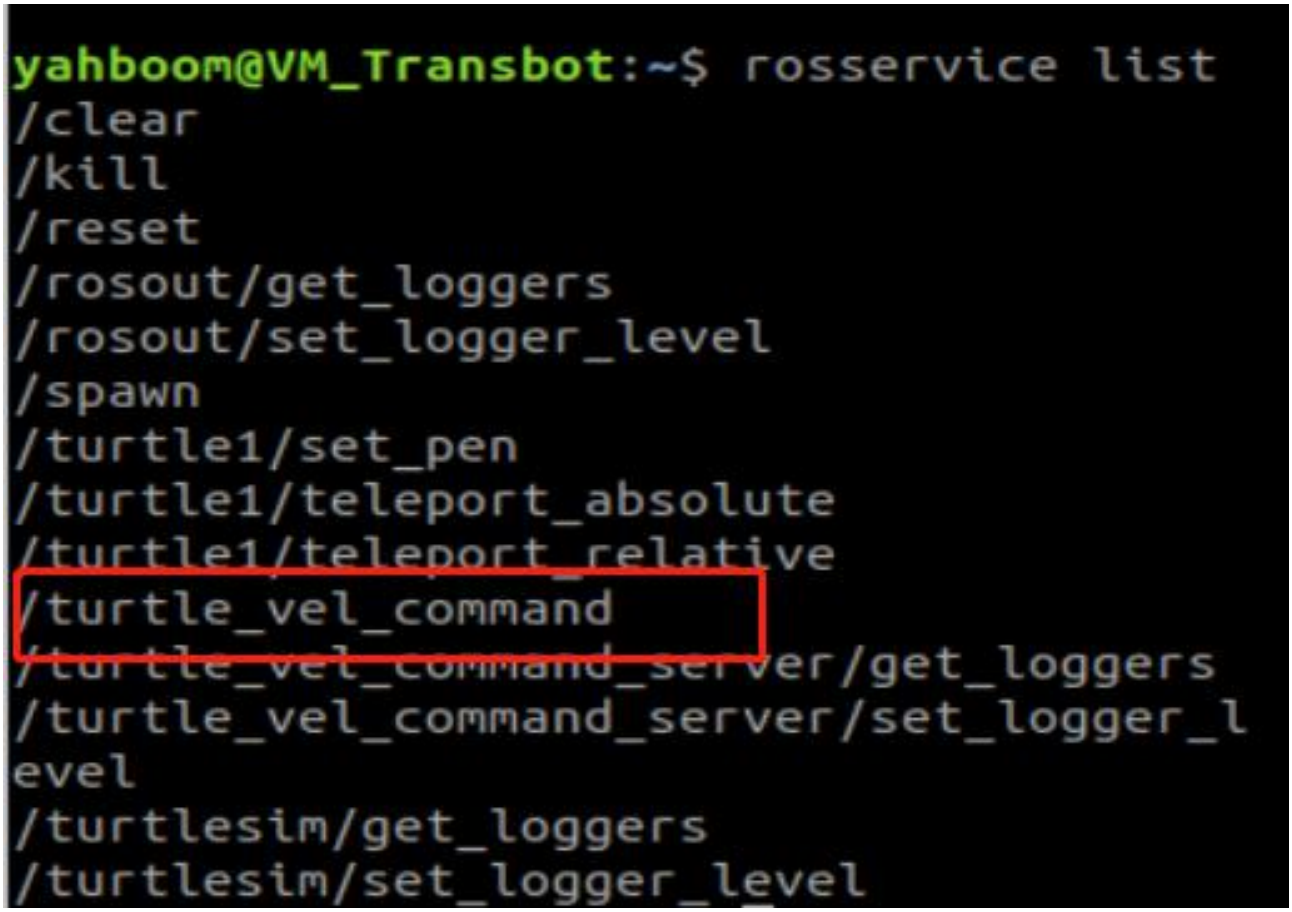
5）、运行效果截图

6）、程序说明

首先，运行小海龟这个节点后，可以在终端输入 rosservice list，查看当前的服务有哪些，结果如下



然后，我们再运行 turtle_vel_command_server 程序，再输入 rosservice list，会发现多了个 turtle_vel_command_server，如下图所示



然后，我们通过在终端输入 rosservice call /turtle_vel_command_server 调用这个服务，会发现小海龟做圆周运动，再次调用服务的话，小海龟停止了运动。这是因为在服务回调函数里边，我们把 pubvel 的值做了反转，然后反馈回去，主函数就会判断 pubvel 的值，如果是 True 就发布速度指令，为 False 则不发布指令。

8.2、python 语言实现

8.2.1、切换至~/catkin_ws/src/learning_server 目录下，新建一个 script 文件夹，切进去，新建一个 py 文件，命名为 turtle_vel_command_server，把下边代码粘贴在里边

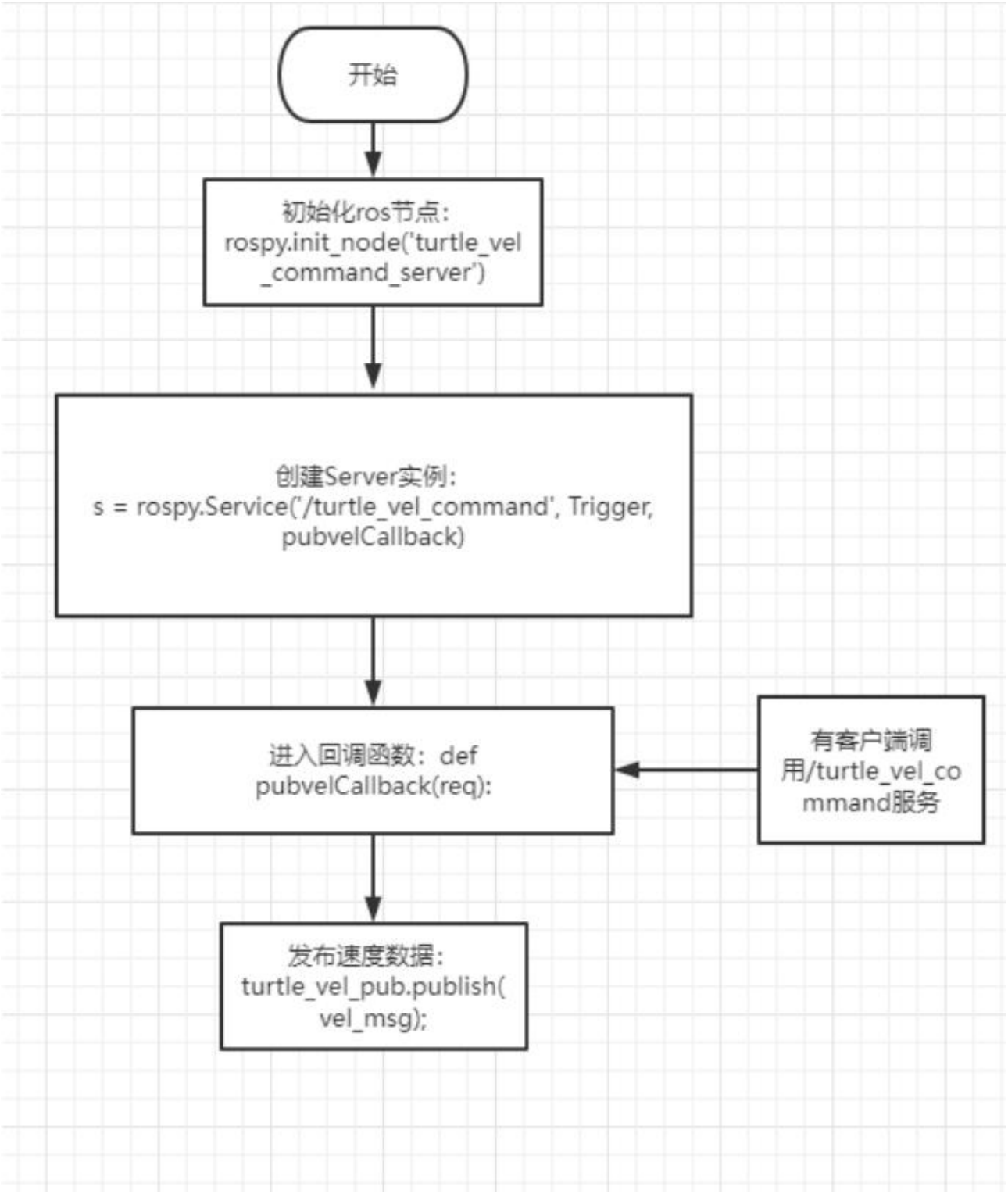
turtle_vel_command_server.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#  该例程将执行/turtle_command 服务，服务数据类型 std_srvs/Trigger

import rospy
import thread,time
from geometry_msgs.msg import Twist
from std_srvs.srv import Trigger, TriggerResponse

pubvel = False;
turtle_vel_pub = rospy.Publisher('/turtle1/cmd_vel', Twist,
queue_size=8)

def pubvel_thread():
    while True:
        if pubvel:
            vel_msg = Twist()
            vel_msg.linear.x = 0.6
            vel_msg.angular.z = 0.8
            turtle_vel_pub.publish(vel_msg)
```

```
            time.sleep(0.1)

def pubvelCallback(req):
    global pubvel

    pubvel = bool(1-pubvel)

    rospy.loginfo("Do you want to publish the vel?[%s]", pubvel)# 显示请求数据

    return TriggerResponse(1, "Change  state!")# 反馈数据

def turtle_pubvel_command_server():

    rospy.init_node('turtle_vel_command_server')# ROS 节点初始化

    # 创建一个名为/turtle_command 的 server，注册回调函数 pubvelCallback
    s = rospy.Service('/turtle_vel_command', Trigger, pubvelCallback)

    # 循环等待回调函数
```

```
    print "Ready to receive turtle_pub_vel_command."

    thread.start_new_thread(pubvel_thread, ())
    rospy.spin()

if __name__ == "__main__":
    turtle_pubvel_command_server()
```

1）、程序流程图



2）、运行程序

```
roscore
```

```
rosrun turtlesim turtlesim_node
```

```
rosrun learning_server turtle_vel_command_server.py
```

3）、程序运行效果和程序说明与 C++实现的效果一致。

8、自定义服务消息与使用

9、自定义服务消息与使用

9.1、自定义服务消息

切换至~/catkin_ws/src/learning_server 功能包目录下，然后新建一个文件夹，命名为 srv，用来存放自定义的服务消息。

9.1.1、定义 srv 文件

切换至 srv 目录下，新建一个空白的 srv 文件，以 srv 为后缀表示代表是 srv 文件。这里我们以 IntPlus.srv 为例子说明下，把以下代码复制到刚刚创建好的 srv 文件里边。

```
uint8 a
uint8 b

---
```

```
uint8 result
```

这里说明下 srv 文件的构成，由符号---分为上下两个部分，上边表示 request 下边是 response。

9.1.2、在 package.xml 中添加功能包依赖

```
<build_depend>message_generation</build_depend>
<exec_depend>message_runtime</exec_depend>
```

9.1.3、在 CMakeLists.txt 添加编译选项

```
在 find_package 里边加上 message_generation
add_service_files(FILES IntPlus.srv)
generate_messages(DEPENDENCIES std_msgs)
```

9.1.4、编译生成语言相关文件

```
cd ~/catkin_ws
catkin_make
```

9.1.5、C++语言实现

1）、切换至~/catkin_ws/src/learning_server/src 下，新建两个 cpp 文件，命名为 IntPlus_server.cpp 和 IntPlus_client.cpp，把以下代码分别复制到里边，

IntPlus_server.cpp

```
/**
 * 该例程将执行/Two_Int_Plus 服务，服务数据类型 learning_service::IntPlus
 */

#include <ros/ros.h>
#include "learning_server/IntPlus.h"

// service 回调函数，输入参数 req，输出参数 res
bool IntPlusCallback(learning_server::IntPlus::Request  &req,
                     learning_server::IntPlus::Response &res)
{

    ROS_INFO("number 1 is:%d   ,number 2 is:%d   ", req.a, req.b);// 显示请求数据

    res.result = req.a + req.b ;// 反馈结果为两数之和

```

```
    return res.result;
}

int main(int argc, char **argv)
{

    ros::init(argc, argv, "IntPlus_server"); // ROS 节点初始化

    ros::NodeHandle n;// 创建节点句柄

    // 创建一个 server，注册回调函数 IntPlusCallback
    ros::ServiceServer Int_Plus_service =
n.advertiseService("/Two_Int_Plus", IntPlusCallback);

    // 循环等待回调函数
    ROS_INFO("Ready to caculate.");
    ros::spin();

    return 0;
}
```

IntPlus_client.cpp

```
/*
```

```
 该例程将请求/Two_Int_Plus服务，服务数据类型learning_service::IntPlus
两个整型数相加求和
 */

#include <ros/ros.h>
#include "learning_server/IntPlus.h"
#include <iostream>
using namespace std;
int main(int argc, char** argv)
{
    int i,k;
    cin>>i;
    cin>>k;

    ros::init(argc, argv, "IntPlus_client");// 初始化 ROS 节点


    ros::NodeHandle node;// 创建节点句柄

   // 发现/Two_Int_Plus 服务后，创建一个服务客户端
    ros::service::waitForService("/Two_Int_Plus");
    ros::ServiceClient IntPlus_client =
node.serviceClient<learning_server::IntPlus>("/Two_Int_Plus");

   // 初始化 learning_service::IntPlus 的请求数据
```

```
    learning_server::IntPlus srv;
    srv.request.a = i;
    srv.request.b   = k;

    ROS_INFO("Call service to plus %d and %d", srv.request.a,
srv.request.b);// 请求服务调用

    IntPlus_client.call(srv);

    // 显示服务调用结果
    ROS_INFO("Show the result : %d", srv.response.result);// 显示服务
调用结果

    return 0;
}
```

2）、修改 CMakeLists.txt 文件

```

add_executable(IntPlus_server src/IntPlus_server.cpp)
target_link_libraries(IntPlus_server ${catkin_LIBRARIES})
add_dependencies(IntPlus_server
${PROJECT_NAME}_generate_messages_cpp)

add_executable(IntPlus_client src/IntPlus_client.cpp)
target_link_libraries(IntPlus_client ${catkin_LIBRARIES})
```

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

3）、核心部分

这里的实现流程和之前的一样，主要不一样的就是引入头文件和使用自定义服务文件：

引入头文件是

```
#include "learning_server/IntPlus.h"
```

前边 learning_server 是功能包名字，后边的 IntPlus.h 是刚才创建的 srv 文件产生的头文件名字

使用自定义服务文件是

```
client：
learning_server::IntPlus srv;
srv.request.a = i;
srv.request.b  = k;
#i，k 为终端输入的加数
ros::ServiceClient IntPlus_client =
node.serviceClient<learning_server::IntPlus>("/Two_Int_Plus");
IntPlus_client.call(srv);
```

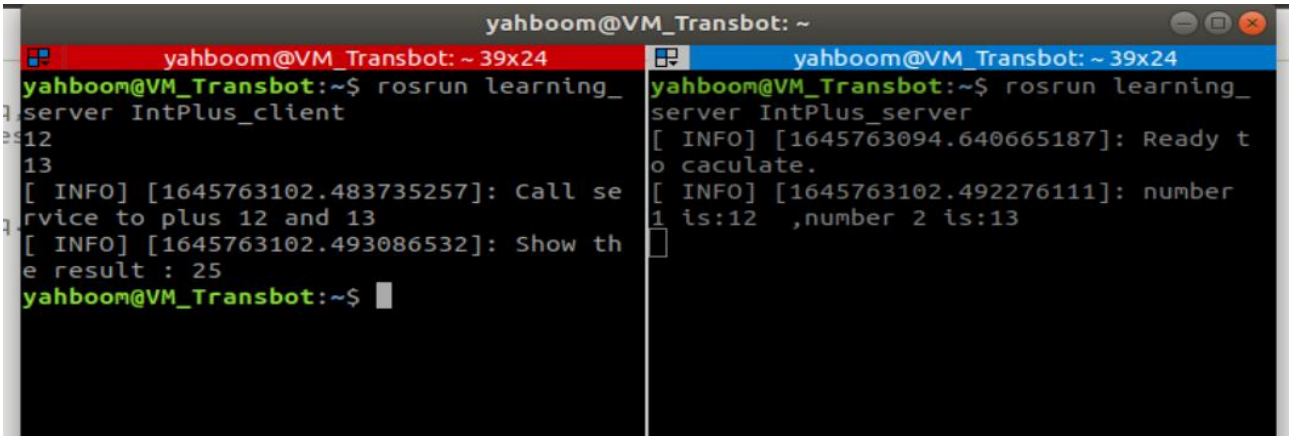
server：

```
ros::ServiceServer Int_Plus_service =
n.advertiseService("/Two_Int_Plus", IntPlusCallback);
bool IntPlusCallback(learning_server::IntPlus::Request  &req,
                     learning_server::IntPlus::Response &res)
```

4）、运行程序

```
roscore
rosrun learning_server IntPlus_client
rosrun learning_server IntPlus_server
```

5）、运行截图



6）、程序说明

运行 IntPlus_server 后，会提示准备计算；运行 IntPlus_client 后，终端输入两个整型数字，接着 IntPlus_server 会计算出结果，并且返回回去给 IntPlus_client，随后打印出结果。

9.1.6、Pytho 语言实现

1)、切换至~/catkin_ws/src/learning_server/script 下，新建两个 py 文件，命名为 IntPlus_server.py 和 IntPlus_client.py，把以下代码分别复制到里边，

IntPlus_server.py

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

from learning_server.srv import IntPlus, IntPlusResponse

def IntPlusCallback(req):

    rospy.loginfo("Ints: a:%d    b:%d", req.a, req.b)# 显示请求数据

    return IntPlusResponse(req.a+req.b)# 反馈数据

def IntPlus_server():

    rospy.init_node('IntPlus_server')# ROS 节点初始化

     # 创建一个 server，注册回调函数 IntPlusCallback
    s = rospy.Service('/Two_Int_Plus', IntPlus, IntPlusCallback)
```

```
    print "Ready to caculate two ints."# 循环等待回调函数
    rospy.spin()
if __name__ == "__main__":
    IntPlus_server()
```

IntPlus_client.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import sys
import rospy
from learning_server.srv import IntPlus, IntPlusRequest
def Plus_client():
    # ROS 节点初始化
    rospy.init_node('IntPlus_client')
    rospy.wait_for_service('/Two_Int_Plus')
    try:
        Plus_client = rospy.ServiceProxy('/Two_Int_Plus', IntPlus)
```

```
        response = Plus_client(22, 20)# 请求服务调用，输入请求数据

        return response.result
    except rospy.ServiceException, e:
        print "failed to call service : %s"%e

if __name__ == "__main__":
    #服务调用并显示调用结果
    print "Show two_int_plus result : %s" %(Plus_client())
```

2）、核心部分

这里主要是说明下如何导入自定义的服务消息模块和使用：

导入

```
server:
from learning_server.srv import IntPlus, IntPlusResponse
client:
from learning_server.srv import IntPlus, IntPlusRequest
```

使用

```
server：
s = rospy.Service('/Two_Int_Plus', IntPlus, IntPlusCallback)
```

```
return IntPlusResponse(req.a+req.b)# 反馈数据
client:
response = Plus_client(12, 20)# 请求服务调用，输入请求数据
return response.result
```

3）、运行程序

运行程序前，先给 py 文件增加可执行权限

```
sudo chmod a+x IntPlus_server.py
sudo chmod a+x IntPlus_client.py
```

运行程序

```
roscore
rosrun learning_server IntPlus_client.py
rosrun learning_server IntPlus_server.py
```

4）、程序运行说明

这里与 C++版本不一致的是，这里的加数是程序里边设定（12 和 20）的，因此开启服务后，就立即能够返回结果。

9、TF 发布与监听

10、tf 发布与监听

10.1、tf 功能包

10.1.1、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、坐标系间完成点、向量等坐标变换。

10.1.2、使用步骤

1）、监听 tf 变换

接收并缓存系统中发布的所有坐标系变换数据，并从中查询所需要的坐标变换关系。

2）、广播 tf 变换

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

10.2、tf 坐标系广播与监听的编程实现

10.2.1、创建并且编译功能包

```
cd ~/catkin_ws/src
catkin_create_pkg learning_tf rospy roscpp turtlesim tf
cd ..
catkin_make
```

10.2.2、如何实现一个 tf 广播器

1）、定义 tf 广播器（TransformBroadcaster）；

2）、初始化 tf 数据，创建坐标变换值；

3）、发布坐标变换（sendTransform）；

10.2.3、如何实现一个 tf 监听器

1）、定义 TF 监听器（TransformListener）；

2）、查找坐标变换（waitForTransform、lookupTransform）

10.2.4、C++语言实现 tf 广播器

1)、在功能包 learning_tf 的 src 文件夹,创建一个 c++文件(文件后缀为.cpp)，命名为 turtle_tf_broadcaster.cpp

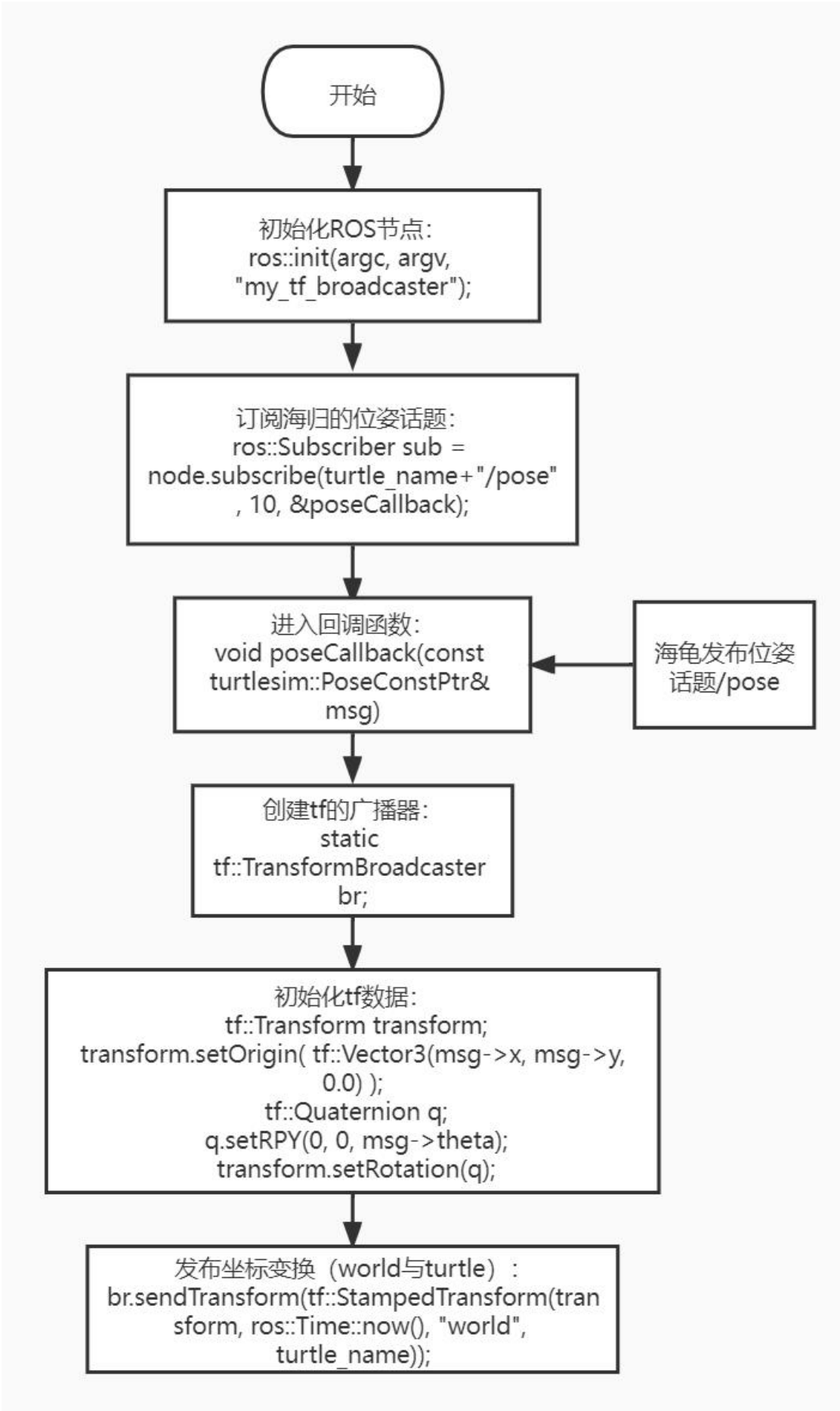
2）、把下边的程序代码复制粘贴到 turtle_tf_broadcaster.cpp 文件中

```
#include <ros/ros.h>
```

```
#include <tf/transform_broadcaster.h>
#include <turtlesim/Pose.h>

std::string turtle_name;

void poseCallback(const turtlesim::PoseConstPtr& msg)
{

    static tf::TransformBroadcaster br;// 创建 tf 的广播器

    // 初始化 tf 数据
    tf::Transform transform;
    transform.setOrigin( tf::Vector3(msg->x, msg->y, 0.0) );//设置 xyz
坐标
    tf::Quaternion q;
    q.setRPY(0, 0, msg->theta);//设置欧拉角：以 x 轴，y 轴，z 轴旋转
    transform.setRotation(q);

    br.sendTransform(tf::StampedTransform(transform,
ros::Time::now(), "world", turtle_name));// 广播 world 与 turtle 坐标系之
间的 tf 数据
}

int main(int argc, char** argv)
{
```

```
    ros::init(argc, argv, "turtle_world_tf_broadcaster");// 初始化 ROS 节
点

    if (argc != 2)
    {
        ROS_ERROR("Missing a parameter as the name of the
turtle!");
        return -1;
    }

    turtle_name = argv[1];// 输入参数作为海龟的名字

    // 订阅海龟的位姿话题/pose
    ros::NodeHandle node;
    ros::Subscriber sub = node.subscribe(turtle_name+"/pose", 10,
&poseCallback);

        // 循环等待回调函数
    ros::spin();

    return 0;
};
```

3）、程序流程图


开始
初始化ROS节点:
ros::init(argc, argv,
"my_tf_broadcaster");
订阅海归的位姿话题:
ros::Subscriber sub =
node.subscribe(turtle_name+"/pose"
, 10, &poseCallback);
进入回调函数:
void poseCallback(const
turtlesim::PoseConstPtr&
msg)
海龟发布位姿
话题/pose
创建tf的广播器:
static
tf::TransformBroadcaster
br;
初始化tf数据:
tf::Transform transform;
transform.setOrigin(tf::Vector3(msg->x, msg->y,
0.0));
tf::Quaternion q;
q.setRPY(0, 0, msg->theta);
transform.setRotation(q);
发布坐标变换（world与turtle）:
br.sendTransform(tf::StampedTransform(tran
sform, ros::Time::now(), "world",
turtle_name));

4）、代码解析

首先，先订阅小海龟的/pose 位姿话题，倘若有该话题发布，那么就进入回调函数。再回调函数里边，首先先创建了 tf 的广播器，然后初始化 tf 数据，数据的值就是订阅/pose 话题传过来的数据。最后，通过 br.sendTransform 把小海龟对于世界坐标的变换发布出去，这里说下 sendTransform 这个函数。有 4 个参数，第一个参数表示 tf::Transform 类型的坐标转换（也就是之前初始化的 tf 数据），第二个参数是时间戳，第三个和第四个是变换的源坐标系和目标坐标系。

10.2.4、C++语言实现 tf 监听器

1)、在功能包 learning_tf 的 src 文件夹,创建一个 c++文件(文件后缀为.cpp)，命名为 turtle_tf_listener.cpp

2）、把下边的程序代码复制粘贴到 turtle_tf_listener.cpp 文件中

```
/**
 * 该例程监听 tf 数据，并计算、发布 turtle2 的速度指令
   turtle2->turtle1 = world->turtle*world->turtle2
 */

#include <ros/ros.h>
#include <tf/transform_listener.h>
#include <geometry_msgs/Twist.h>
```

```
#include <turtlesim/Spawn.h>

int main(int argc, char** argv)
{

    ros::init(argc, argv, "turtle1_turtle2_listener");// 初始化 ROS 节点


    ros::NodeHandle node; // 创建节点句柄

    // 请求服务产生 turtle2
    ros::service::waitForService("/spawn");
    ros::ServiceClient add_turtle =
node.serviceClient<turtlesim::Spawn>("/spawn");
    turtlesim::Spawn srv;
    add_turtle.call(srv);

    // 创建发布 turtle2 速度控制指令的发布者
    ros::Publisher vel =
node.advertise<geometry_msgs::Twist>("/turtle2/cmd_vel", 10);

    tf::TransformListener listener;// 创建 tf 的监听器

    ros::Rate rate(10.0);

    while (node.ok())
```

```
    {
        // 获取 turtle1 与 turtle2 坐标系之间的 tf 数据
        tf::StampedTransform transform;
        try
        {
            listener.waitForTransform("/turtle2", "/turtle1",
ros::Time(0), ros::Duration(3.0));
            listener.lookupTransform("/turtle2", "/turtle1",
ros::Time(0), transform);
        }
        catch (tf::TransformException &ex)
        {
            ROS_ERROR("%s",ex.what());
            ros::Duration(1.0).sleep();
            continue;
        }

        // 根据 turtle1 与 turtle2 坐标系之间的位置关系，通过数学计算公
式，得出角速度和线速度，发布 turtle2 的速度控制指令
        geometry_msgs::Twist turtle2_vel_msg;

        turtle2_vel_msg.angular.z = 6.0 *
atan2(transform.getOrigin().y(),
```

```
                                                transform.getOrigin().x());
            turtle2_vel_msg.linear.x = 0.8 *
sqrt(pow(transform.getOrigin().x(), 2) +
                                              pow(transform.getOrigin().y(), 2));
            vel.publish(turtle2_vel_msg);

            rate.sleep();
        }
        return 0;
};
```

3）、程序流程图

4）、代码解析

首先，通过服务调用产生另外一只小乌龟 turtle2，然后创建 turtle2 速度控制发布者；接着创建一个监听器，监听和查找 turtle1 和 tuetle2 的左边变换，这里涉及到两个函数 waitForTransform 和 lookupTransform

waitForTransform(target_frame,source_frame,time,timeout):两个 frame 分别表示目标坐标系和源坐标系，time 表示等待两个坐标系之间变换的时间，因为坐标变换是阻塞程序，所以需要设置 timeout，表示超时时间。

lookupTransform(target_frame,source_frame,time,transform):给定源坐标系(source_frame)和目标坐标系(target_frame)，得到两个坐标系之间指定时间(time)的坐标变换(transform)。

经过 lookupTransform 我们得到了坐标变换的结果 transform，然后通过 transform.getOrigin().y(),transform.getOrigin().x()得到 x,y 的值，接着通过数学运算得到角速度 angular.z 和线速度 linear.x，最后发布出去，让 turtle2 做运动。

10.2.5、修改 CMakelist.txt 和编译

1）、修改修改 CMakelist.txt

修改功能包下的 CMakelist.txt，添加下边内容

```
add_executable(turtle_tf_listener src/turtle_tf_listener.cpp)
target_link_libraries(turtle_tf_listener ${catkin_LIBRARIES})

add_executable(turtle_tf_broadcaster src/turtle_tf_broadcaster.cpp)
target_link_libraries(turtle_tf_broadcaster ${catkin_LIBRARIES})
```

2）、编译

```
cd ~/catkin_ws
catkin_make
```

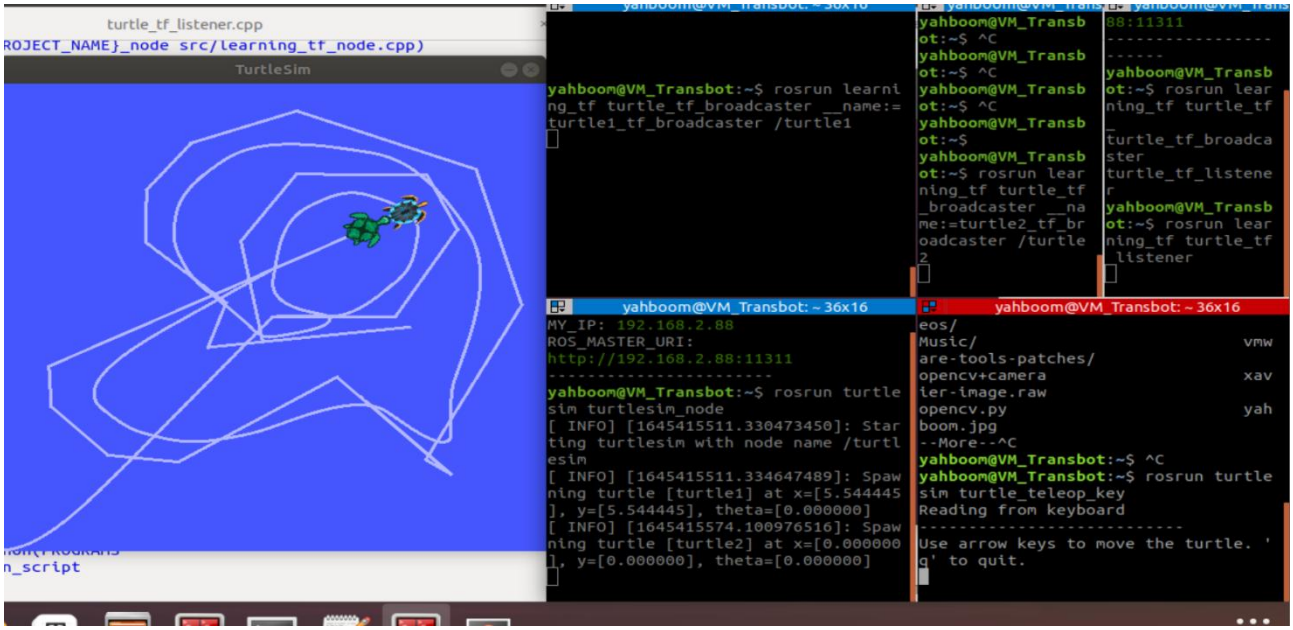
source devel/setup.bash　　#需要配置环境变量，否则系统无法找到运行程序

10.2.6、启动与运行效果演示

1）、启动

```
roscore
rosrun turtlesim turtlesim_node
rosrun learning_tf turtle_tf_broadcaster __name:=turtle1_tf_broadcaster /turtle1
rosrun learning_tf turtle_tf_broadcaster __name:=turtle2_tf_broadcaster /turtle2
rosrun learning_tf turtle_tf_listener
rosrun turtlesim turtle_teleop_key  #打开海龟键盘控制程序，控制海龟运动
```

2）、效果展示



3）、程序说明

启动 roscore 后，开启小海龟节点，这时候终端会出现一只小海龟；然后我们发布两个 tf 变换，turtle1->world，turtle2->world，因为要想知道 turtle2 与 turtle1 之间的变化，则需要知道他们跟 world 之间的变换；然后，开启 tf 监听程序，这时候会发现终端产生了另外一只小海龟 turtle2，并且 turtle2 会向 turtle1 移动；然后，我们打开键盘控制，通过按下方向键来控制 turtle1 移动，然后 turtle2 会追着 turtle1 移动。

10.2.7、Python 语言实现 tf 广播器

1）、在功能包 learning_tf,创建一个文件夹 script，切换到该目录下，新建一个.py 文件，命名为 turtle_tf_broadcaster.py

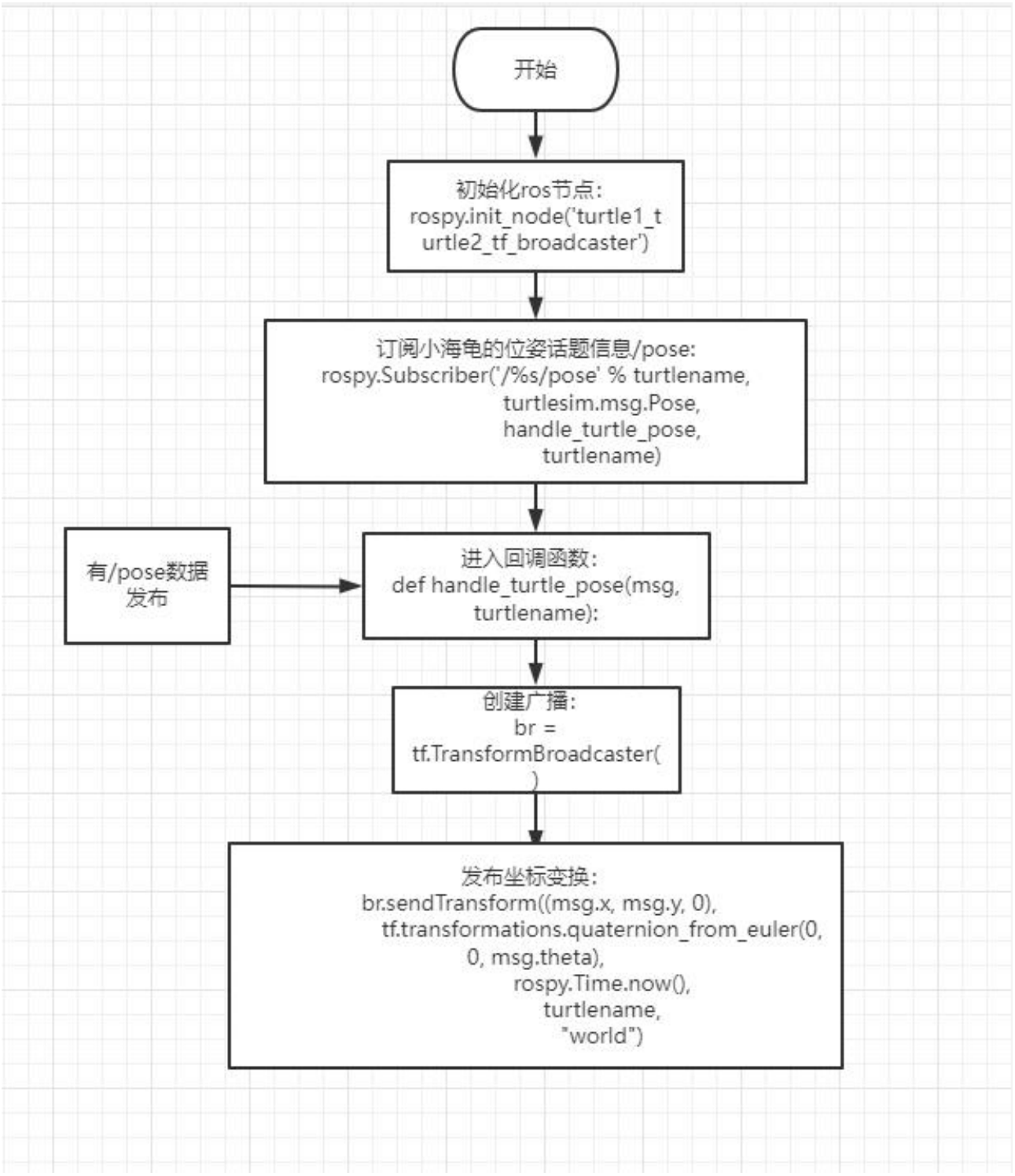
2）、把下边的程序代码复制粘贴到 turtle_tf_broadcaster.py 文件中

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

```
# -*- coding: utf-8 -*-

import roslib
roslib.load_manifest('learning_tf')
import rospy

import tf
import turtlesim.msg

def handle_turtle_pose(msg, turtlename):
    br = tf.TransformBroadcaster()#定义一个 tf 广播
    #广播 world 与输入命名的 turtle 之间的 tf 变换
    br.sendTransform((msg.x, msg.y, 0),
                     tf.transformations.quaternion_from_euler(0, 0,
msg.theta),
                     rospy.Time.now(),
                     turtlename,
                     "world")

if __name__ == '__main__':

    rospy.init_node('turtle1_turtle2_tf_broadcaster')#初始化 ros 节点

```

```
    turtlename = rospy.get_param('~turtle') #从参数服务器中获取 turtle
的名字
    #订阅/pose 话题数据，也就是 turtle 的位姿信息
    rospy.Subscriber('/%s/pose' % turtlename,
                     turtlesim.msg.Pose,
                     handle_turtle_pose,
                     turtlename)
    rospy.spin()
```

3）、程序流程图



10.2.8、Python 语言实现 tf 监听器

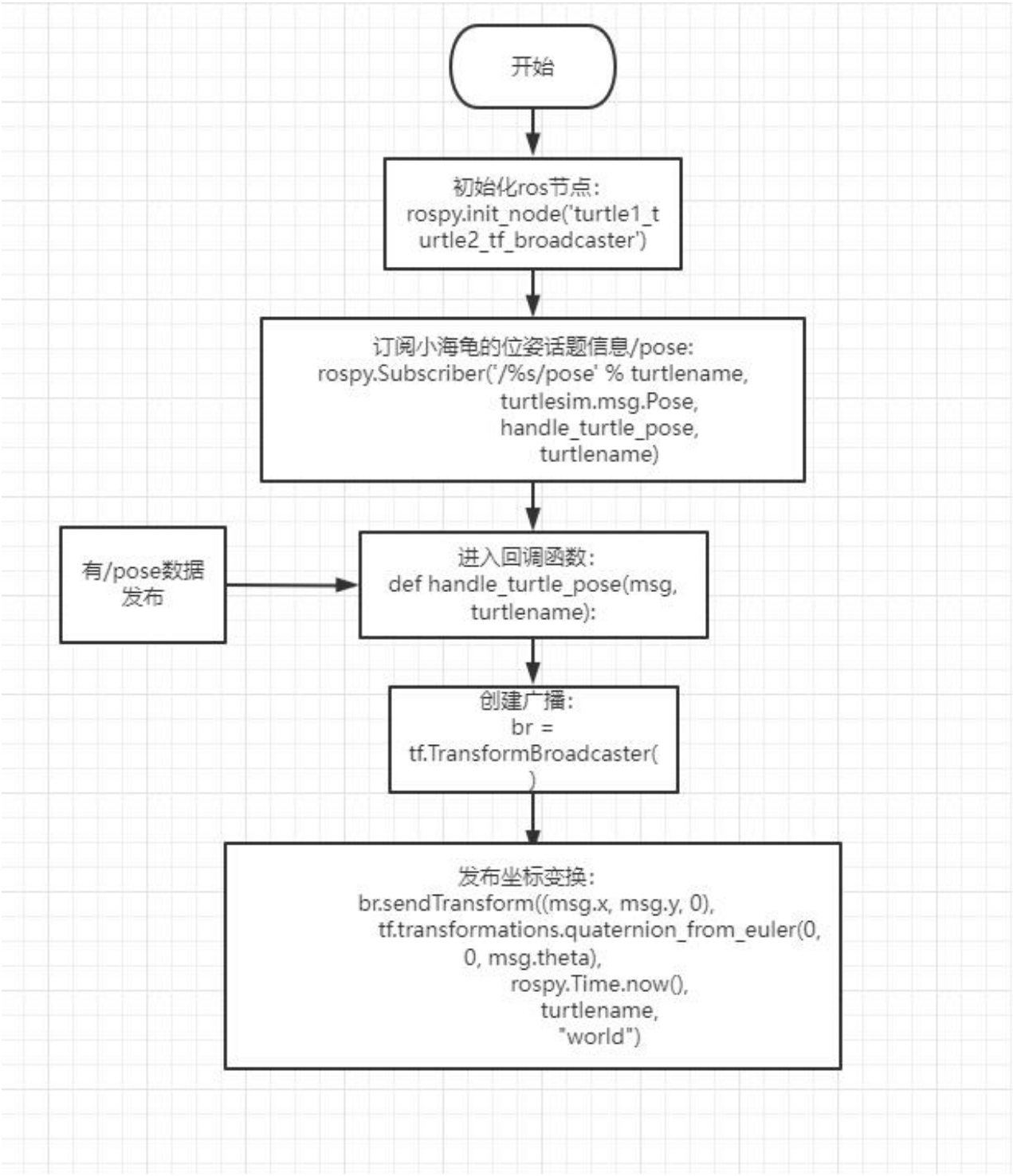
1）、在功能包 learning_tf 的 script 文件夹,创建一个 python 文件（文件后缀为.py），命名为 turtle_tf_listener.py

2）、把下边的程序代码复制粘贴到 turtle_tf_listener.py 文件中

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

import rospy
import math
import tf
import geometry_msgs.msg
import turtlesim.srv

if __name__ == '__main__':
    rospy.init_node('turtle_tf_listener')#初始化 ros 节点

    listener = tf.TransformListener()#初始化一个监听者

    rospy.wait_for_service('spawn')
    #调用服务产生另一只海龟 turtle2
    spawner = rospy.ServiceProxy('spawn', turtlesim.srv.Spawn)
    spawner(8, 6, 0, 'turtle2')
    #声明一个发布者，用来发布 turtle2 的速度
    turtle_vel = rospy.Publisher('turtle2/cmd_vel',
geometry_msgs.msg.Twist,queue_size=1)

```

```
    rate = rospy.Rate(10.0)
    while not rospy.is_shutdown():
        try:
            #查找 turtle2 和 turtle1 之间的 tf 变化
            (trans,rot) = listener.lookupTransform('/turtle2', '/turtle1',
rospy.Time(0))
        except (tf.LookupException, tf.ConnectivityException,
tf.ExtrapolationException):
            continue
        #通过数学计算，算出线速度和角速度，然后发布出去
        angular = 6.0 * math.atan2(trans[1], trans[0])
        linear = 0.8 * math.sqrt(trans[0] ** 2 + trans[1] ** 2)
        cmd = geometry_msgs.msg.Twist()
        cmd.linear.x = linear
        cmd.angular.z = angular
        turtle_vel.publish(cmd)
        rate.sleep()
```

3）、程序流程图



10.2.9、启动与运行效果展示

1）、编写一个 launch 文件

在功能包目录下，新建一个文件夹 launch，切换至 launch 里边，新建一个 launch 文件，命名为 start_tf_demo_py.launch，把以下内容复制到里边，

```
<launch>

    <!-- 海龟节点-->
    <node pkg="turtlesim" type="turtlesim_node" name="sim"/>
   <!--广播 turtle1->world-->
    <node name="turtle1_tf_broadcaster" pkg="learning_tf"
type="turtle_tf_broadcaster.py" respawn="false" output="screen" >
      <param name="turtle" type="string" value="turtle1" />
    </node>
   <!--广播 turtle2->world-->
    <node name="turtle2_tf_broadcaster" pkg="learning_tf"
type="turtle_tf_broadcaster.py" respawn="false" output="screen" >
      <param name="turtle" type="string" value="turtle2" />
    </node>
   <!--监听-->
   <node pkg="learning_tf" type="turtle_tf_listener.py"
name="listener" />
   <!--海龟键盘控制节点-->
```

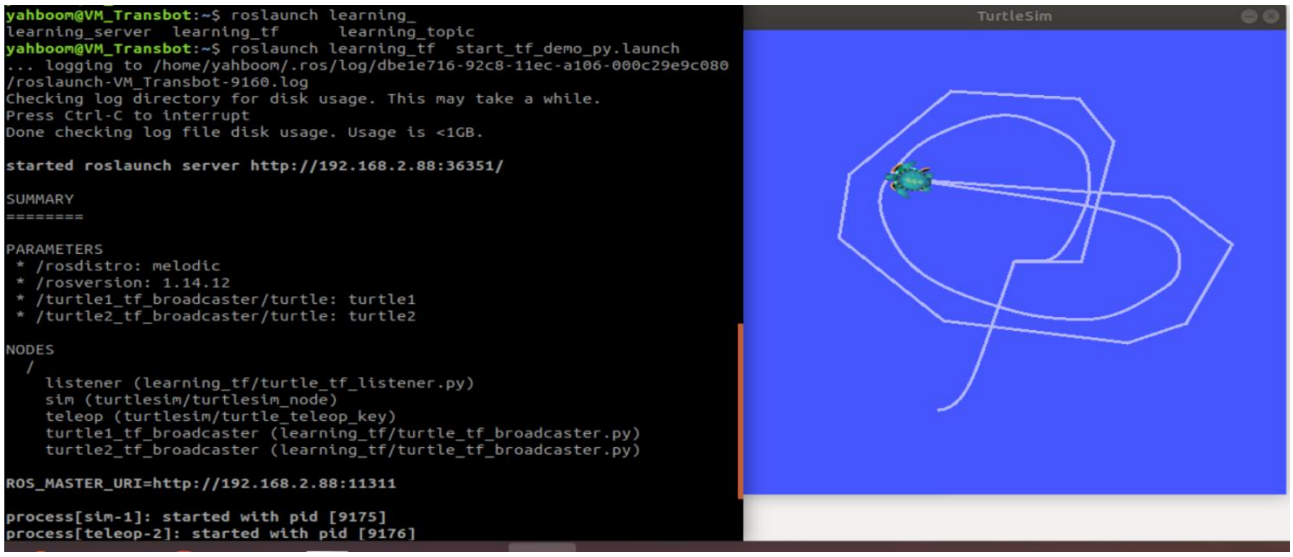
```
    <node pkg="turtlesim" type="turtle_teleop_key" name="teleop"
output="screen"/>
</launch>
```

2）、启动

```
roslaunch learning_tf start_tf_demo_py.launch
```

程序运行后，鼠标点击运行 launch 的窗口，按下方向键，turtle2 会跟着 turtle1 移动。

3）、运行结果如下图



2. ROS1 视觉图像处理

1、AR 视觉

1、AR 视觉

功能包：~/yahboomcar_ws/src/yahboomcar_visual

本节效果可在安装了我们相对应镜像的主板上演示。

1.1、概述

增强现实(Augmented Reality)，简称“AR”，技术是一种将虚拟信息与真实世界巧妙融合的技术，广泛运用了多媒体、三维建模、实时跟踪及注册、智能交互、传感等多种技术手段，将计算机生成的文字、图像、三维模型、音乐、视频等虚拟信息模拟仿真后，应用到真实世界中，两种信息互为补充，从而实现对真实世界的“增强”。

AR 系统具有三个突出的特点：①真实世界和虚拟世界的信息集成；②具有实时交互性；③是在三维尺度空间中增添定位虚拟物体。

增强现实技术包含了多媒体、三维建模、实时视频显示及控制、多传感器融合、实时跟踪及注册、场景融合等新技术与新手段。

1.2、使用方法

在使用 AR 案例时，必须得有相机的内参，不然无法运行。内参文件与代码同目录（功能包的 AR 文件夹下）；不同相机对应不同内参。

（**我们对应主板镜像里已存放好标定的相机内参文件，无需二次标定**）

启动单目相机/树莓派 CSI 相机

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

启动 Jetson CSI 相机

```
roslaunch yahboomcar_visual yahboom_csi.launch
```

启动标定节点 (单目相机/树莓派 CSI 相机)

```
rosrun camera_calibration cameracalibrator.py image:=/usb_cam/image_raw camera:=/usb_cam --size 9x6 --square 0.02
```

启动标定节点 (jetson CSI 相机)

```
rosrun camera_calibration cameracalibrator.py image:=/csi_cam_0/image_raw camera:=/csi_cam_0 --size 9x6 --square 0.02
```

标定完后，将【calibrationdata.tar.gz】文件移动到【home】目录下。

```
sudo mv /tmp/calibrationdata.tar.gz ~
```

解压后，将该文件夹里面的【ost.yaml】打开，找到相机内参矩阵和畸变系数修改到【USB_camera.yaml】或者【csi_camera.yaml】文件对应的位置，只需修改两处【data】的内容即可。例如：以下内容。

```
camera_matrix: !!opencv-matrix
  rows: 3
  cols: 3
  dt: d
  data: [615.50506,    0.       , 365.84388,
            0.       , 623.69024, 238.778   ,
            0.       ,    0.       ,    1.       ]
distortion_model: plumb_bob
distortion_coefficients: !!opencv-matrix
  rows: 1
  cols: 5
  dt: d
  data: [0.166417, -0.160106, -0.008776, 0.025459, 0.000000]
```

一共 12 种效果。

```
["Triangle", "Rectangle",
"Parallelogram","WindMill","TableTennisTable", "Ball", "Arrow",
"Knife", "Desk",
"Bench", "Stickman", "ParallelBars"]
```

启动命令

```
roslaunch yahboomcar_visual simple_AR.launch display:=true
flip:=false
```

display 参数：True；本地显示图像窗口；Fasle，则不显示。

flip 参数：是否水平切换画面，默认即可。

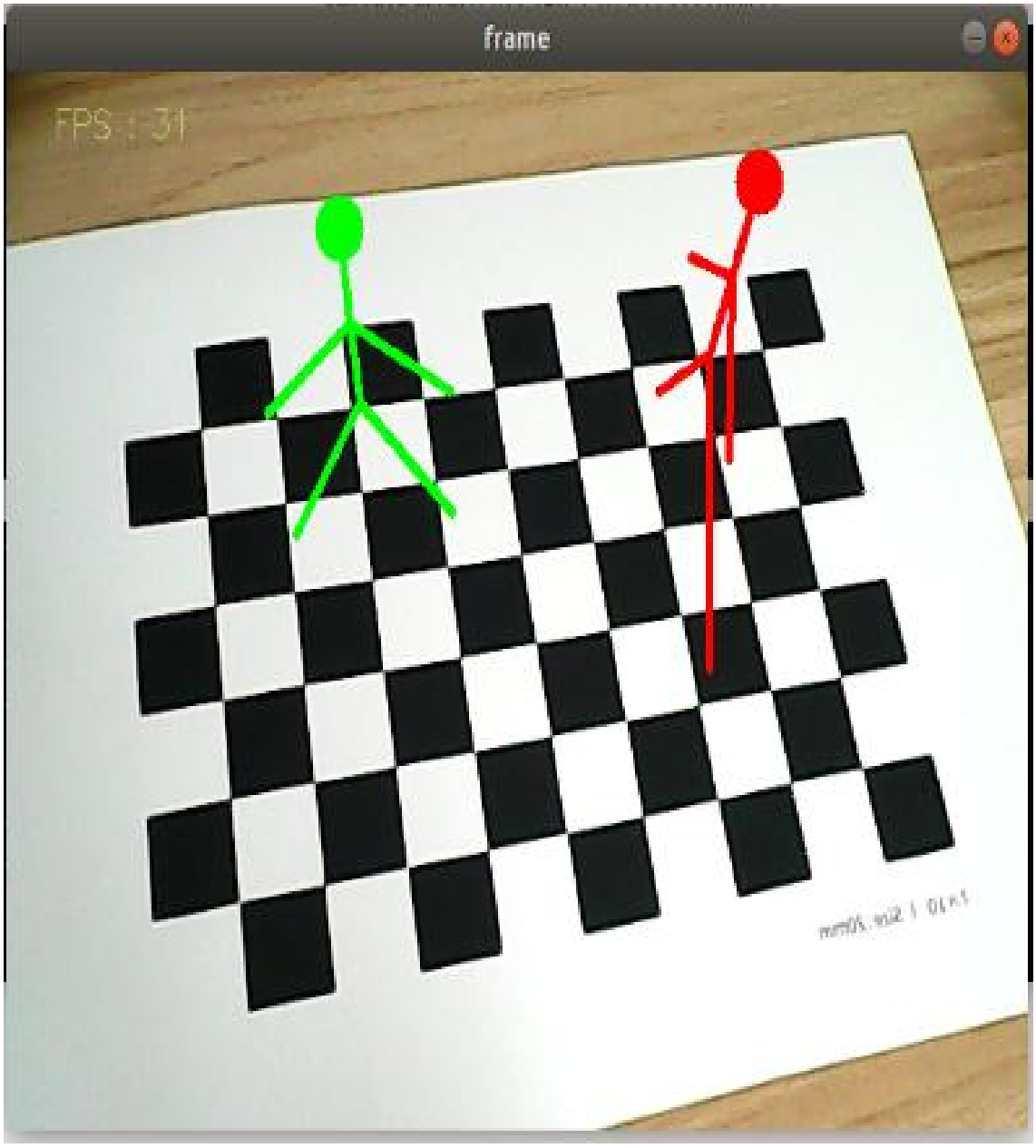
如果是单目相机则需要将 simple_AR_usb.py 修改成 simple_AR.py;

如果是 CSI 相机，则需要将 simple_AR_csi.py 修改成 simple_AR.py

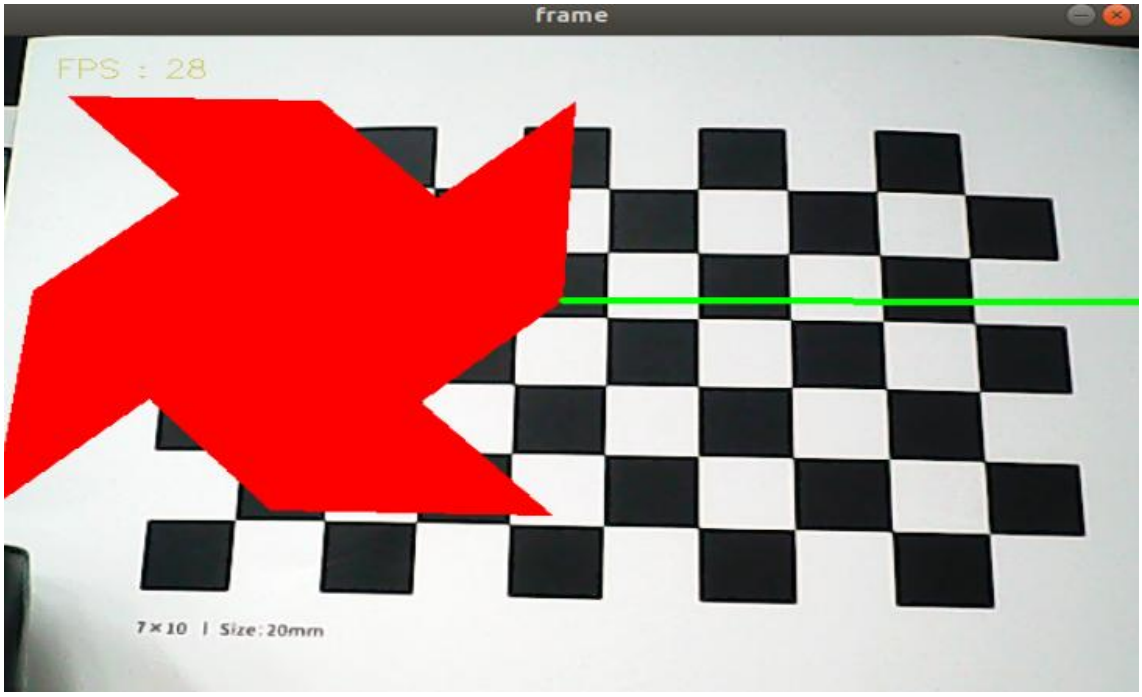
根据需求设置参数，也可直接修改 launch 文件，启动时便无需附带参数。不开启画面时，可使用网络监控方式查看

开启设备的 IP:8080

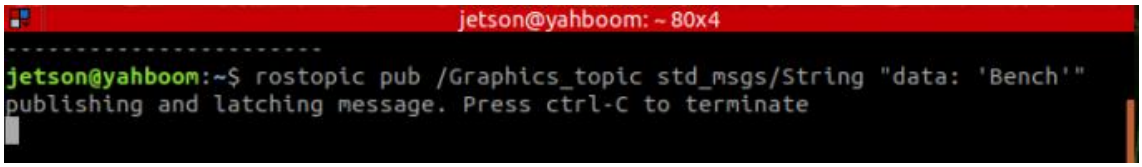
1）在显示画面的情况下（即 display 为 true），【q】键退出，【f】键切换不同效果，也可以使用命令行切换。

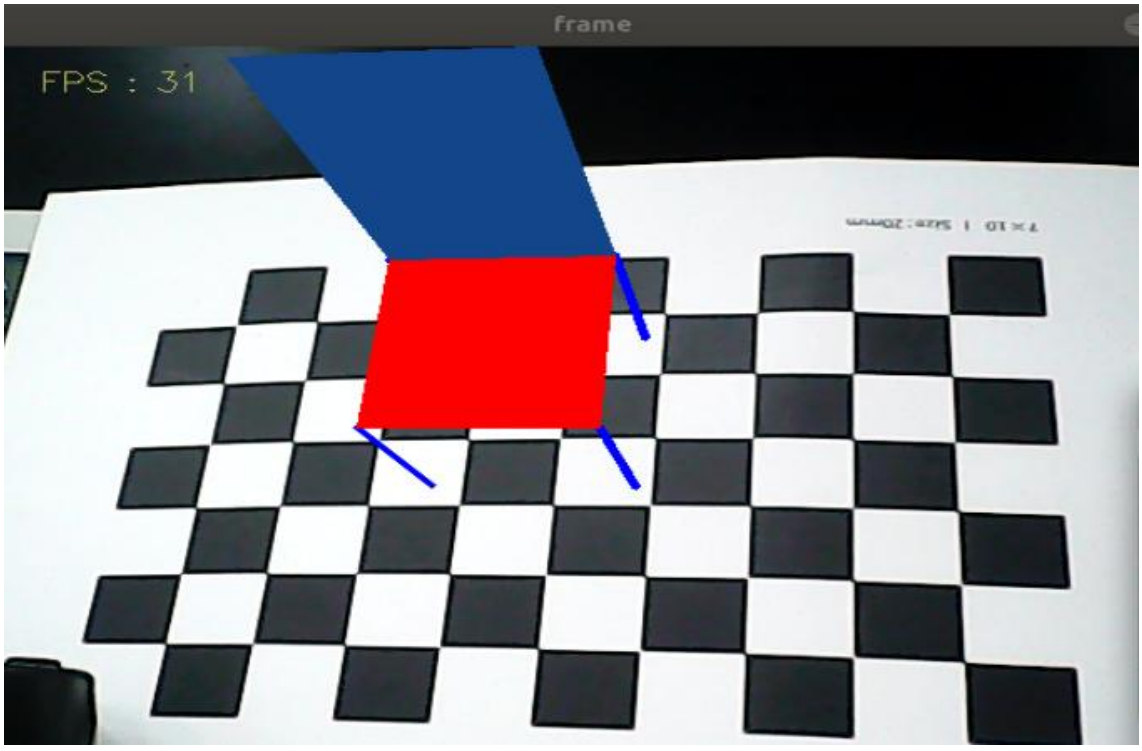


使用【f】或【F】键切换不同效果。



2）在不显示画面的情况下（即 display 为 false），只能通过命令行切换效果



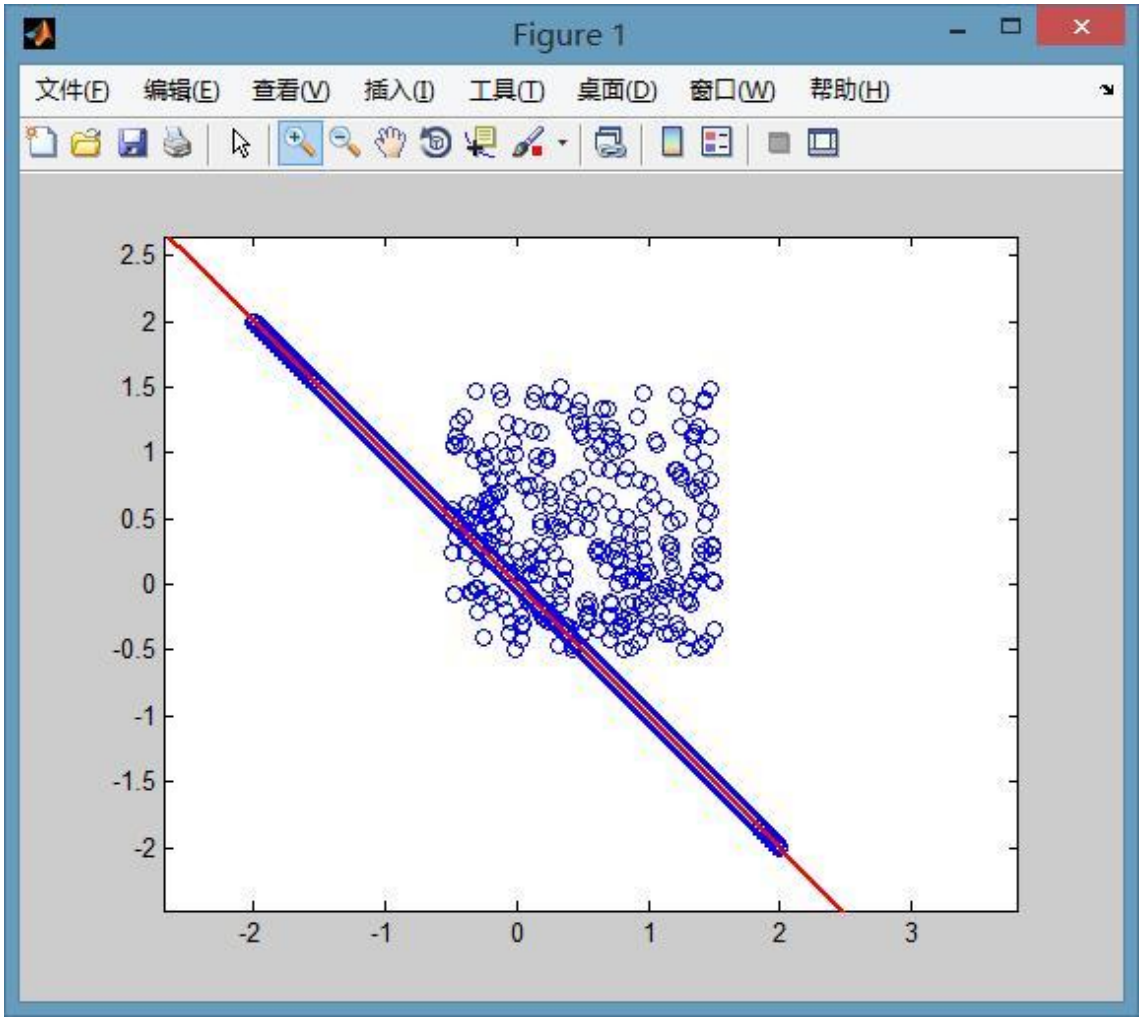


1.3、源码解析

1.3.1、算法原理

使用 RANSAC 方案从 3D-2D 点对应中查找对象姿势。

RanSaC 算法（随机采样一致）原本是用于数据处理的一种经典算法，其作用是在大量噪声情况下，提取物体中特定的成分。下图是对 RanSaC 算法效果的说明。图中有一些点显然是满足某条直线的，另外有一团点是纯噪声。目的是在大量噪声的情况下找到直线方程，此时噪声数据量是直线的 3 倍。



如果用最小二乘法是无法得到这样的效果的，直线大约会在图中直线偏上一点。

RANSAC 的基本假设是： （1）数据由“局内点”组成，例如：数据的分布可以用一些模型参数来解释；（2）“局外点”是不能适应该模型的数据；（3）除此之外的数据属于噪声。 局外点产生的原因有：噪声的极值；错误的测量方法；对数据的错误假设。 RANSAC 也做了以下假设：给定一组（通常很小的）局内点，存在一个可以估计模型参数的过程；而该模型能够解释或者适用于局内点。

1.3.2、核心代码

设计流程：

是

否

输入图像

是否每个图片的角点

查找角点亚像素

输出图像

计算对象姿态

输出图像点和雅可比矩阵

绘制并输出图像

launch 文件

```
<launch>
    <arg name="flip" default="False"/>
    <arg name="display" default="False"/>
    <node name="simple_AR" pkg="yahboomcar_visual"
type="simple_AR.py" output="screen" args="$(arg display)">
        <param name="flip" type="bool" value="$(arg flip)"/>
        <remap from="/simpleAR/camera" to="/simpleAR/camera"/>
    </node>
    <!-- web_video_server -->
    <node pkg="web_video_server" type="web_video_server"
name="web_video_server" output="screen"/>
</launch>
```

python 主函数

```
    def process(self, img):
        if self.flip == 'True': img = cv.flip(img, 1)
        gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
        # 查找每个图片的角点
        retval, corners = cv.findChessboardCorners(
```

```
            gray, self.patternSize, None,
            flags=cv.CALIB_CB_ADAPTIVE_THRESH +
cv.CALIB_CB_NORMALIZE_IMAGE + cv.CALIB_CB_FAST_CHECK)
        # 查找角点亚像素
        if retval:
            corners = cv.cornerSubPix(
                gray, corners, (11, 11), (-1, -1),
                (cv.TERM_CRITERIA_EPS +
cv.TERM_CRITERIA_MAX_ITER, 30, 0.001))
            # 计算对象姿态 solvePnPRansac
            retval, rvec, tvec, inliers = cv.solvePnPRansac(
                self.objectPoints, corners, self.cameraMatrix,
self.distCoeffs)
            # 输出图像点和雅可比矩阵
            image_Points, jacobian = cv.projectPoints(
                self.__axis, rvec, tvec, self.cameraMatrix, self.distCoeffs, )
            # 绘制图像
            img = self.draw(img, corners, image_Points)
        return img
```

关键函数

https://docs.opencv.org/3.0-alpha/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

- findChessboardCorners()

```
def findChessboardCorners(image, patternSize, corners=None,
flags=None):
    '''
    查找图片角点
    :param image: 输入原始的棋盘板图像。该图像必须是一张 8 位的灰度图或色彩图。
    :param patternSize: (w,h)，棋盘上每一排和每一列的内角数。w=棋盘板一行上黑白块的数量-1，h=棋盘板一列上黑白块的数量-1。
    例如：10x6 的棋盘板，则(w,h)=(9,5)
    :param corners: array，检测到的角点的输出数组。
    :param flags: int,不同的操作标记，能够为 0 或者下述值的组合：
    CALIB_CB_ADAPTIVE_THRESH 使用自适应阈值法把图像转换为黑白图，而不是使用一个固定的阈值。
    CALIB_CB_NORMALIZE_IMAGE 在利用固定阈值或自适应阈值法二值化图像之前，利用直方图均衡化图像。
    CALIB_CB_FILTER_QUADS 使用额外的标准(如轮廓面积，周长，正方形形状)来过滤掉在轮廓检索阶段提取的假四边形。
```

```
    CALIB_CB_FAST_CHECK 对图像运行一个快速检查机制以查找棋盘板的角点，如果没有找到角点则返回一个快捷提醒。
    当没有观察到棋盘时，可以极大地加快在退化条件下的调用。
    :return: retval, corners
    '''
    pass
```

- cornerSubPix()

我们需要使用 cornerSubPix()对检测到的角点作进一步的优化计算，可使角点的精度达到亚像素级别。

```
def cornerSubPix(image, corners, winSize, zeroZone, criteria):
    '''
    亚像素角点检测函数
    :param image: 输入图像
    :param corners: 像素角点（既作为输入也作为输出）
    :param winSize: 区域大小为 NXN; N=(winSize*2+1)
    :param zeroZone: 类似于 winSize，但是总具有较小的范围，Size(-1,-1)表示忽略
    :param criteria: 停止优化的标准
    :return: 亚像素角点
    '''
```

```
    pass
```

- solvePnPRansac()

```
def solvePnPRansac(objectPoints, imagePoints, cameraMatrix,
distCoeffs,
                   rvec=None, tvec=None, useExtrinsicGuess=None,
iterationsCount=None,
                   reprojectionError=None, confidence=None,
inliers=None, flags=None):
    '''
    计算对象姿态
    :param objectPoints:   对象点列表
    :param imagePoints:    角点列表
    :param cameraMatrix:   相机矩阵
    :param distCoeffs:     畸变系数
    :param rvec:
    :param tvec:
    :param useExtrinsicGuess:
    :param iterationsCount:
    :param reprojectionError:
    :param confidence:
    :param inliers:
```

```
    :param flags:
    :return: retval, rvec, tvec, inliers
    '''
    pass
```

使用 RANSAC 方案从 3D-2D 点对应关系中查找对象姿态。该函数在给定一组对象点、它们对应的图像投影以及相机矩阵和失真系数的情况下，估计对象姿势。此函数找到一个使重新投影误差最小的姿势，即重新观察误差，即观察到的像素点投影 imagePoints 与物体投影（projectPoints（））objectPoints 之间的平方距离之和。 RANSAC 的使用可以避免异常值对结果的影响。

- projectPoints()

```
def projectPoints(objectPoints, rvec, tvec, cameraMatrix, distCoeffs, imagePoints=None, jacobian=None, aspectRatio=None):
    '''
    输出图像点和雅可比矩阵
    :param objectPoints:
    :param rvec:    旋转向量
    :param tvec:    平移向量
    :param cameraMatrix:    摄影机矩阵
    :param distCoeffs:     失真系数
```

```
    :param imagePoints:
    :param jacobian:
    :param aspectRatio:
    :return: imagePoints, jacobian
    '''
    pass
```

2、AR 二维码

2、AR 二维码

2、AR 二维码 2.1、概述 2.2、创建 ARTag2.2.1、安装软件包 2.2.2、创建 AR 二维码 2.3、ARTag 识别 2.3.1、启动识别实例 2.3.2、launch 文件解析（以单目相机/树莓派 CSI 相机为例）2.3.3、ar_track_alvar 节点 2.3.4、查看节点图 2.3.5、查看 tf 树 2.3.6、查看输出信息

2.1、概述

wiki：http://wiki.ros.org/ar_track_alvar/

源码：https://github.com/ros-perception/ar_track_alvar.git

功能包位置：~/yahboomcar_ws/src/yahboomcar_visual

ARTag（AR 标签，AR 是“增强现实”的意思）是一种基准标记系统，可以理解为其他物体一种参照物，看起来类似二维码，但其编码系统和二维码还

是有很大区别，多用在相机标定，机器人定位，增强现实（AR）等应用场合，其中一个很重要的作用就是识别物体与相机的位姿关系。物体上可以贴上 ARTag，或者在平面上贴上 ARTag 标签，用来标定相机。摄像头识别到 ARTag 后，可以算出标签在相机坐标中的位置与姿态。

ar_track_alvar 有 4 个主要功能：

生成不同大小、分辨率和数据/ID 编码的 AR 标签。

识别和跟踪单个 AR 标签的姿势，可选择集成 kinect 深度数据（当 kinect 可用时）以获得更好的姿势估计。

识别和跟踪由多个标签组成的“束”的姿势。这允许更稳定的姿态估计、对遮挡的鲁棒性以及对多边对象的跟踪。

使用相机图像自动计算捆绑中标签之间的空间关系，这样用户就不必手动测量并在 XML 文件中输入标签位置来使用捆绑功能。

Alvar 比 ARToolkit 更新、更先进，ARToolkit 一直是其他几个 ROS AR 标签包的基础。Alvar 具有自适应阈值处理以处理各种光照条件、基于光流的跟踪以实现更稳定的姿态估计，以及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减慢。

2.2、创建 ARTag

2.2.1、安装软件包

melodic:

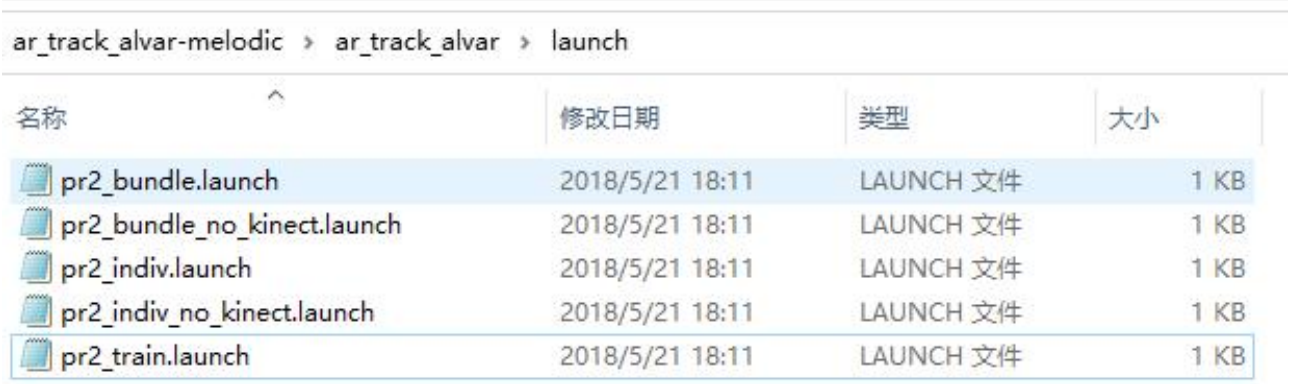
```
```

```
sudo apt install ros-melodic-ar-track-alvar
```

noetic 系统没有命令来安装 ar-track-alvar，只能通过源码安装:

```
sudo apt install python3-colcon-common-extensions
```

```
#  在工作空间的 src 文件夹下
git clone https://github.com/machinekoder/ar_track_alvar.git -b
noetic-devel
#然后重新编译工作空间
catkin build
```



ar_track_alvar 是一个开源的 marker 库，它提供了 pr2+kinect 的示例。该软件包的第一个用例是识别和跟踪（可能的）多个 AR 标签的姿势，每个 AR 标签都是单独考虑的。

2.2.2、创建 AR 二维码

- 连续生成多个标签在一张图片上

```
rosrun ar_track_alvar createMarker
```

```
Description:
  This is an example of how to use the 'MarkerData' and 'MarkerArtoolkit'
  classes to generate marker images. This application can be used to
  generate markers and multimarker setups that can be used with
  SampleMarkerDetector and SampleMultiMarker.

Usage:
  /opt/ros/melodic/lib/ar_track_alvar/createMarker [options] argument

    65535             marker with number 65535
    -f 65535          force hamming(8,4) encoding
    -1 "hello world"  marker with string
    -2 catalog.xml    marker with file reference
    -3 www.vtt.fi     marker with URL
    -u 96             use units corresponding to 1.0 unit per 96 pixels
    -uin              use inches as units (assuming 96 dpi)
    -ucm              use cm's as units (assuming 96 dpi) <default>
    -s 5.0            use marker size 5.0x5.0 units (default 9.0x9.0)
    -r 5              marker content resolution -- 0 uses default
    -m 2.0            marker margin resolution -- 0 uses default
    -a                use ArToolkit style matrix markers
    -p                prompt marker placements interactively from the user


Prompt marker placements interactively
  units: 1 cm 0.393701 inches
  marker side: 9 units
  marker id (use -1 to end) [0]:
```

可以在这里输入【ID】和位置信息，输入【-1】结束。可生成一个或多个，

布局自己设计。

```
Prompt marker placements interactively
  units: 1 cm 0.393701 inches
  marker side: 9 units
  marker id (use -1 to end) [0]: 0
  x position (in current units) [0]: 0
  y position (in current units) [0]: 0
ADDING MARKER 0
  marker id (use -1 to end) [1]: 1
  x position (in current units) [18]: 0
  y position (in current units) [0]: 10
ADDING MARKER 1
  marker id (use -1 to end) [2]: 2
  x position (in current units) [18]: 10
  y position (in current units) [0]: 0
ADDING MARKER 2
  marker id (use -1 to end) [3]: 3
  x position (in current units) [10]: 10
  y position (in current units) [18]: 10
ADDING MARKER 3
  marker id (use -1 to end) [4]: -1
Saving: MarkerData_0_1_2_3.png
Saving: MarkerData_0_1_2_3.xml
```

- 生成单个数字

命令+参数直接生成数字图片；例如

```
rosrun ar_track_alvar createMarker 11
rosrun ar_track_alvar createMarker -s 5 33
```

11：数字是 11 的二维码。 -s：指定图像大小。 5：5x5 的图片。33：数字是 33 的二维码。

2.3、ARTag 识别

注意：启动摄像头时，需要加载相机标定文件，否则无法识别。

2.3.1、启动识别实例

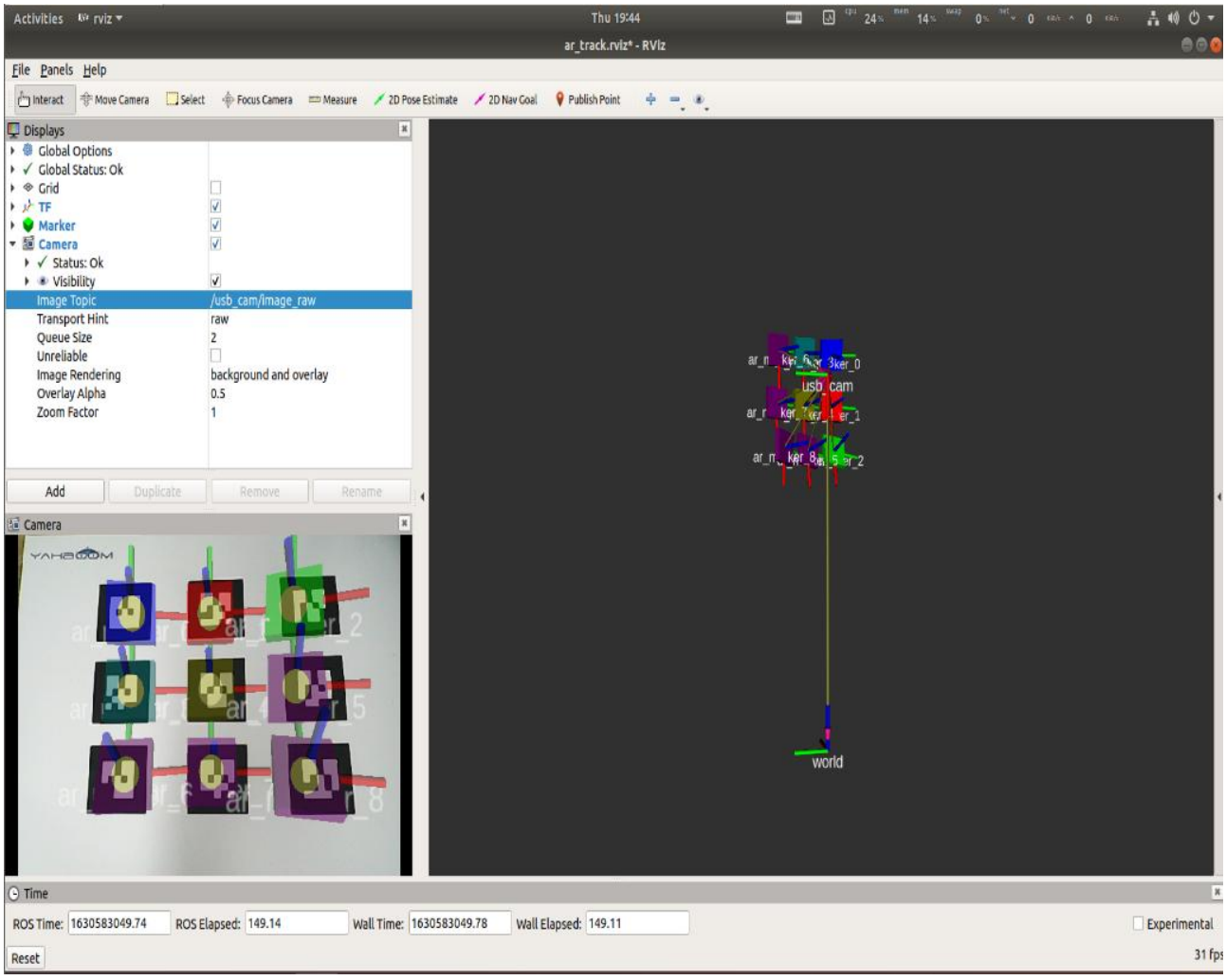
```
roslaunch yahboomcar_visual ar_track.launch open_rviz:=true
```

- open_rviz 参数默认打开就好。
-
-

如果是单目相机或者树莓派 CSI 相机则需要将 ar_track_usb.launch 修改成 ar_track.launch

-

如果是 Jetson CSI 相机，则需要将 ar_track_csi.launch 修改成 ar_track.launch

- 

在 rviz 中，需要设置对应的相机话题名。

- Image_Topic：如果是单目相机或者树莓派相机，则需要勾选【/usb_cam/image_raw】 如果是 Jetson CSI 相机，则需要勾选【/csi_cam_0/image_raw】。
- Marker：rviz 的显示组件，不同的方块显示 AR 二维码所在位置。

- TF：rviz 的显示组件，用来显示 AR 二维码的坐标系。
- Camera：rviz 的显示组件，显示相机画面。
- world：世界坐标系。
- usb_cam：相机坐标系。

2.3.2、launch 文件解析（以单目相机/树莓派 CSI 相机为例）

```
<launch>
    <!-- 设置 camDevice 参数，默认是 USBCam -->
    <arg name="open_rviz" default="true"/>
    <arg name="marker_size" default="5.0"/>
    <arg name="max_new_marker_error" default="0.08"/>
    <arg name="max_track_error" default="0.2"/>
    <!-- 设置相机图像话题、相机内参话题、相机 frame -->
    <arg name="cam_image_topic"
default="/usb_cam/image_raw"/>
    <arg name="cam_info_topic" default="/usb_cam/camera_info"/>
    <arg name="output_frame" default="/usb_cam"/>
    <!-- 启动相机节点 -->
    <include file="$(find usb_cam)/launch/usb_cam-test.launch"/>
    <!-- 设置相机坐标系和世界坐标系对应关系 -->
```

```
    <node pkg="tf" type="static_transform_publisher"
name="world_to_cam" args="0 0 0.5 0 1.57 0 world usb_cam 10"/>
    <!-- 启动 AR 识别节点 -->
    <node name="ar_track_alvar" pkg="ar_track_alvar"
type="individualMarkersNoKinect" respawn="false"
output="screen">
        <param name="marker_size" type="double" value="$(arg
marker_size)"/>
        <param name="max_new_marker_error" type="double"
value="$(arg max_new_marker_error)"/>
        <param name="max_track_error" type="double" value="$(arg
max_track_error)"/>
        <param name="output_frame" type="string" value="$(arg
output_frame)"/>
        <remap from="camera_image" to="$(arg
cam_image_topic)"/>
        <remap from="camera_info" to="$(arg cam_info_topic)"/>
    </node>
    <!-- 启动 rviz -->
    <node pkg="rviz" type="rviz" name="rviz" args="-d $(find
yahboomcar_visual)/rviz/ar_track.rviz" if="$(arg open_rviz)"/>
</launch>
```

节点参数：

- marker_size (double) ： 黑色方形标记边框一侧的宽度（以厘米为单位）。
- max_new_marker_error (double)：在不确定条件下确定何时可以检测到新标记的阈值。
- max_track_error (double) ： 一个阈值，用于确定在标记消失之前可以观察到多少跟踪错误。
- camera_image (string) ： 提供用于检测 AR 标签的图像话题名。这可以是单色的，也可以是彩色的，但应该是未经校正的图像，因为校正是在这个包中进行的。
- camera_info (string) ： 提供摄像机校准参数以便校正图像的主题名称。
- output_frame (string) ： 发布 AR 标签在相机坐标系下的坐标位置。

2.3.3、ar_track_alvar 节点

- **Subscribed topic**

话题名	数据类型
/camera_info	(sensor_msgs/CameraInfo)
/image_raw	(sensor_msgs/Image)

- **Published Topics**

话题名	数据类型
/visualization_marker	(visualization_msgs/Marker)
/ar_pose_marker	(ar_track_alvar/AlvarMarkers)

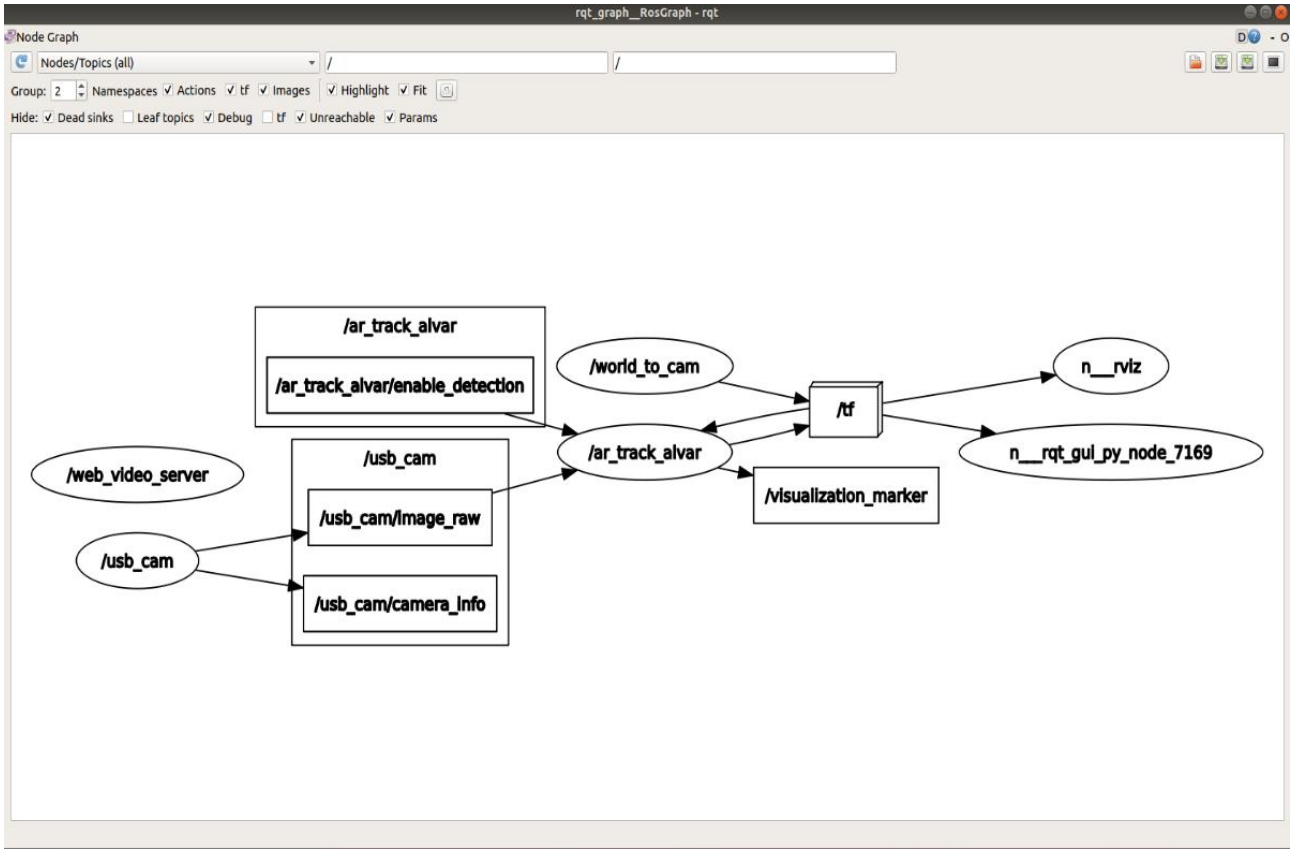
- **Provided tf Transforms**

- 单个二维码：相机坐标系 → AR 标签坐标系

  多个二维码:提供从相机坐标系到每个 AR 标签坐标系(命名为 ar_marker_x)的变换，其中 x 是标记的 ID 号。

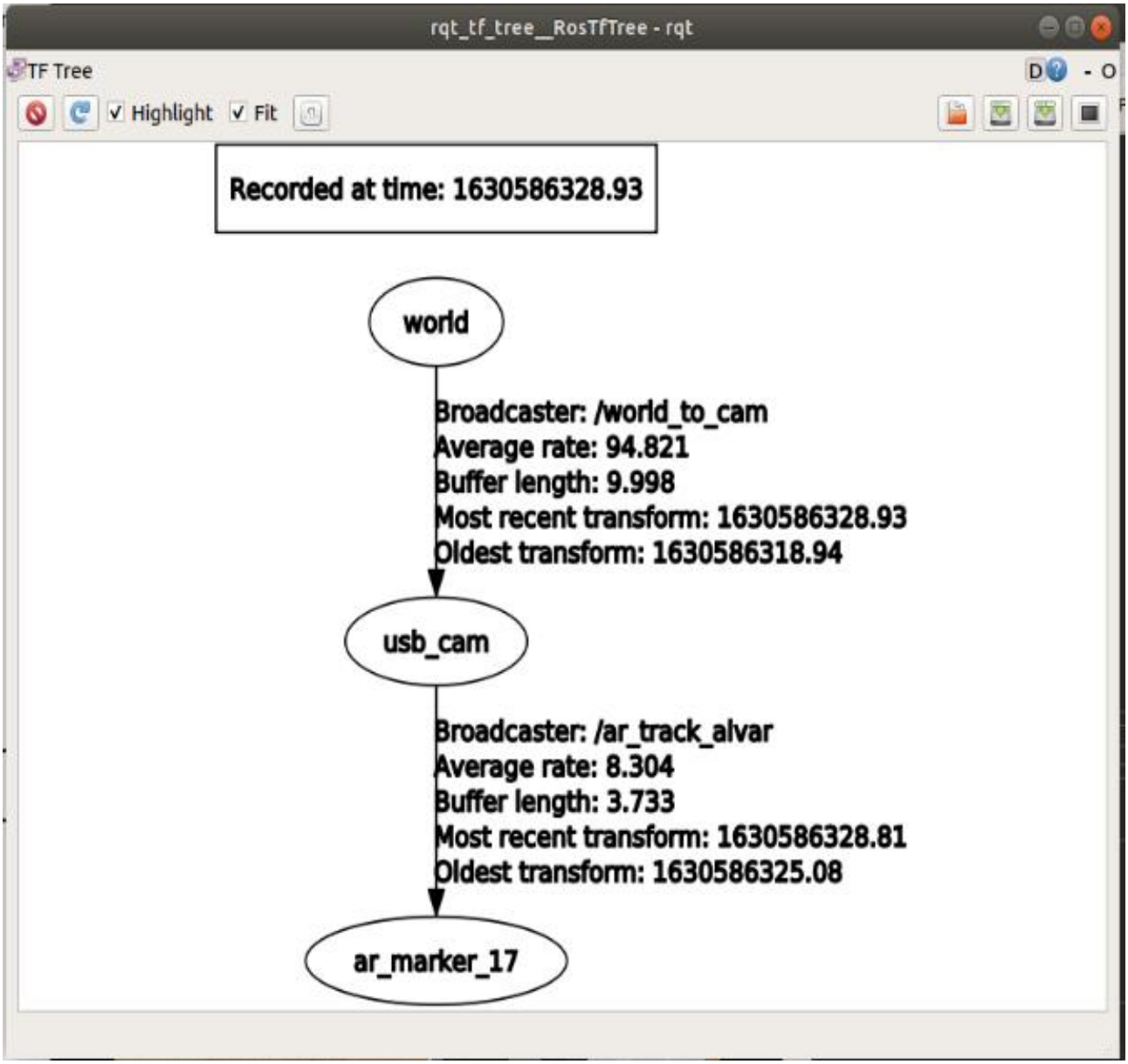
  2.3.4、查看节点图

  ```
  rqt_graph
  ```



2.3.5、查看 tf 树

```
rosrun rqt_tf_tree rqt_tf_tree
```



通过 rviz，我们可以很直观的看到二维码和相机的相对位置。相机和世界坐标系是自己设定的。

2.3.6、查看输出信息

```
rostopic echo /ar_pose_marker
```

显示如下：

```
  header:
    seq: 0
    stamp:
      secs: 1630584915
      nsecs: 196221070
    frame_id: "/usb_cam"
  id: 3
  confidence: 0
  pose:
    header:
      seq: 0
      stamp:
        secs: 0
        nsecs:         0
      frame_id: ''
    pose:
      position:
        x: 0.0249847882514
        y: 0.0290736736336
        z: 0.218054183012
      orientation:
```

```
            x: 0.682039034537
            y: 0.681265739969
            z: -0.156112715404
            w: 0.215240718735
```

- frame_id：相机的坐标系名称
- id：识别的数字是 3
- pose：二维码的位姿
- position：二维码坐标系相对相机坐标系的位置
- orientation：二维码坐标系相对相机坐标系的姿态

3、ROS+Opencv 基础

3、 ROS+Opencv 基础

3、 ROS+Opencv 基础 3.1、概述 3.2、单目相机/树莓派 CSI 相机 3.2.1、启动单目相机/树莓派 CSI 相机 3.2.2、启动彩色图像反转 3.3、Jetson CSI 相机 3.3.1、启动 Jetson CSI 相机 3.2.2、启动彩色图像反转

3.1、概述

Wiki：http://wiki.ros.org/cv_bridge/

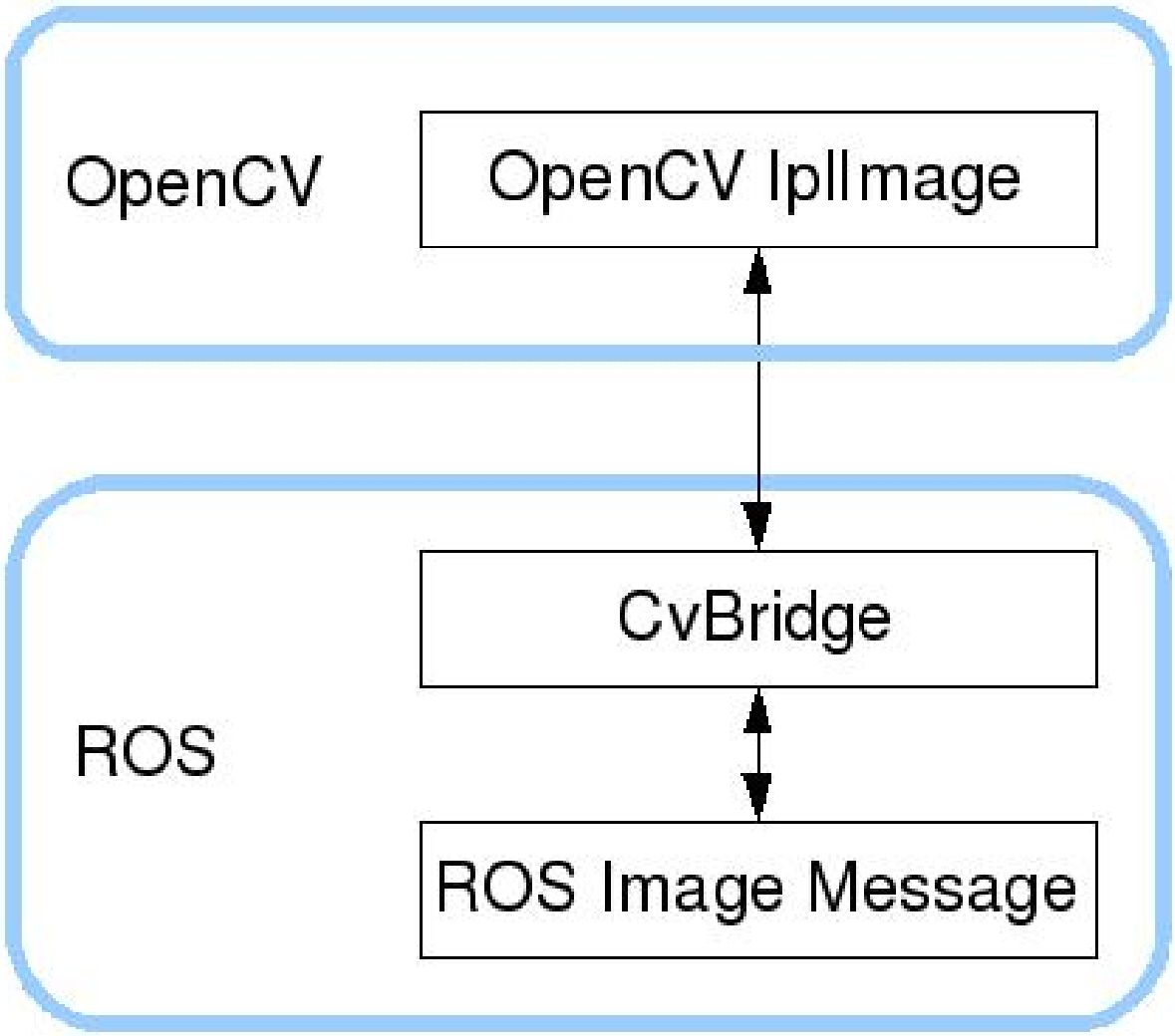
教学：http://wiki.ros.org/cv_bridge/Tutorials

源码：https://github.com/ros-perception/vision_opencv.git

功能包位置：~/yahboomcar_ws/src/yahboomcar_visual

ROS 在安装的过程中就已经集成了 Opencv3.0 以上的版本，所以安装配置几乎不需要过多考虑，ROS 以自己的 sensor_msgs/Image 消息格式传递图像，无法直接进行图像处理，但是提供的【CvBridge】可以完美转换和被转换图像数据格式。【CvBridge】是一个 ROS 库，相当于 ROS 和 Opencv 之间的桥梁。

Opencv 和 ROS 图像数据转换如下图所示：



虽然安装配置不需要过多考虑，但是使用环境还是需要配置的，主要是【package.xml】和【CMakeLists.txt】这两个文件。该功能包不仅仅是使用【CvBridge】，还需要【Opencv】和【PCL】，所以一起配置完毕。

package.xml

添加如下内容

```
<build_depend>sensor_msgs</build_depend>
<build_export_depend>sensor_msgs</build_export_depend>
<exec_depend>sensor_msgs</exec_depend>

<build_depend>std_msgs</build_depend>
<build_export_depend>std_msgs</build_export_depend>
<exec_depend>std_msgs</exec_depend>
<build_depend>cv_bridge</build_depend>
<build_export_depend>cv_bridge</build_export_depend>
<exec_depend>cv_bridge</exec_depend>
<exec_depend>image_transport</exec_depend>
```

【cv_bridge】：图像转换依赖包。

- CMakeLists.txt

该文件配置内容较多，具体内容请查看源文件。

3.2、单目相机/树莓派 CSI 相机

3.2.1、启动单目相机/树莓派 CSI 相机

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

查看话题

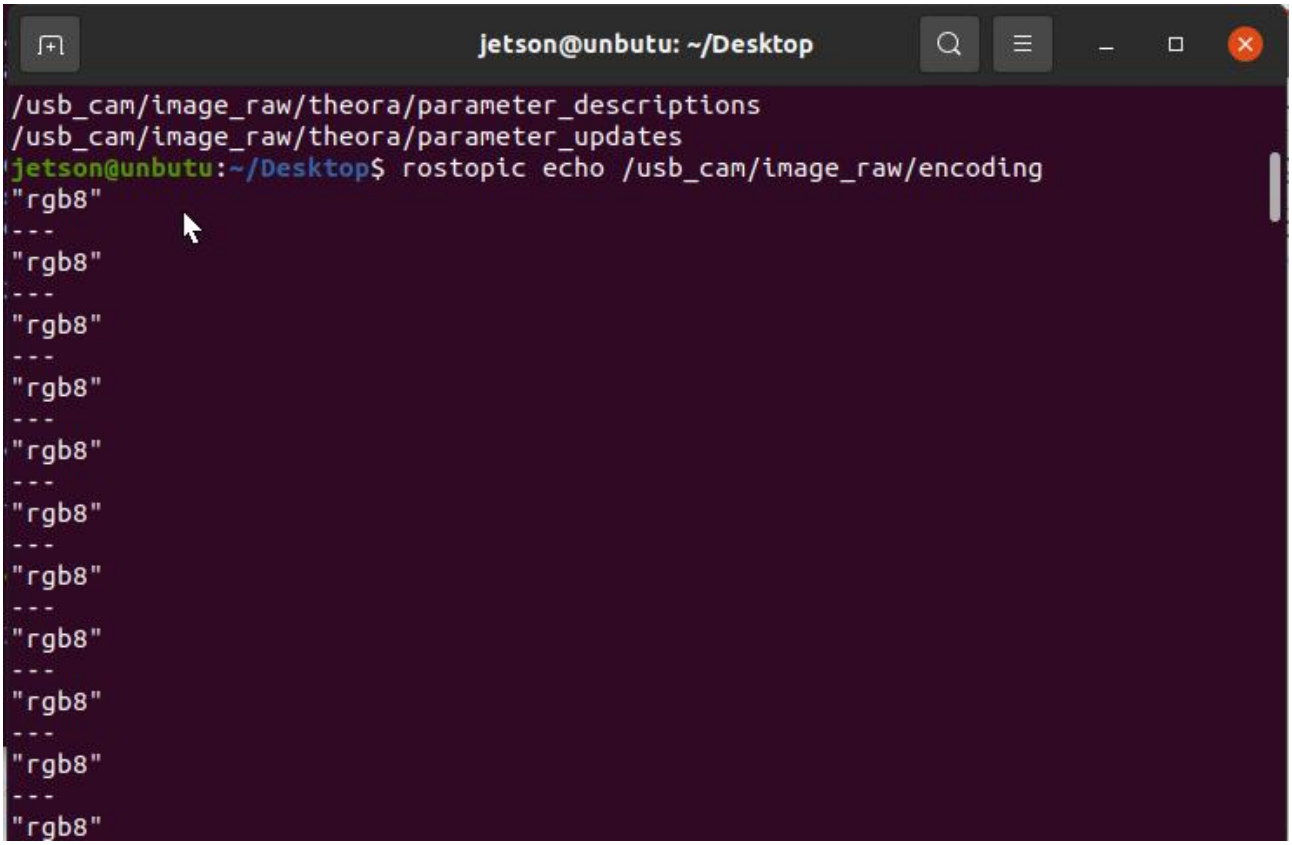
```
rostopic list
```

可以看到话题很多，本节常用的就几个

话题名	数据类型
/usb_cam/image_raw	sensor_msgs/Image
/usb_cam/image_raw/compressedDepth	sensor_msgs/CompressedImage
/usb_cam/image_raw/compressed	sensor_msgs/CompressedImage

查看话题的编码格式：rostopic echo +【topic】+encoding，例如

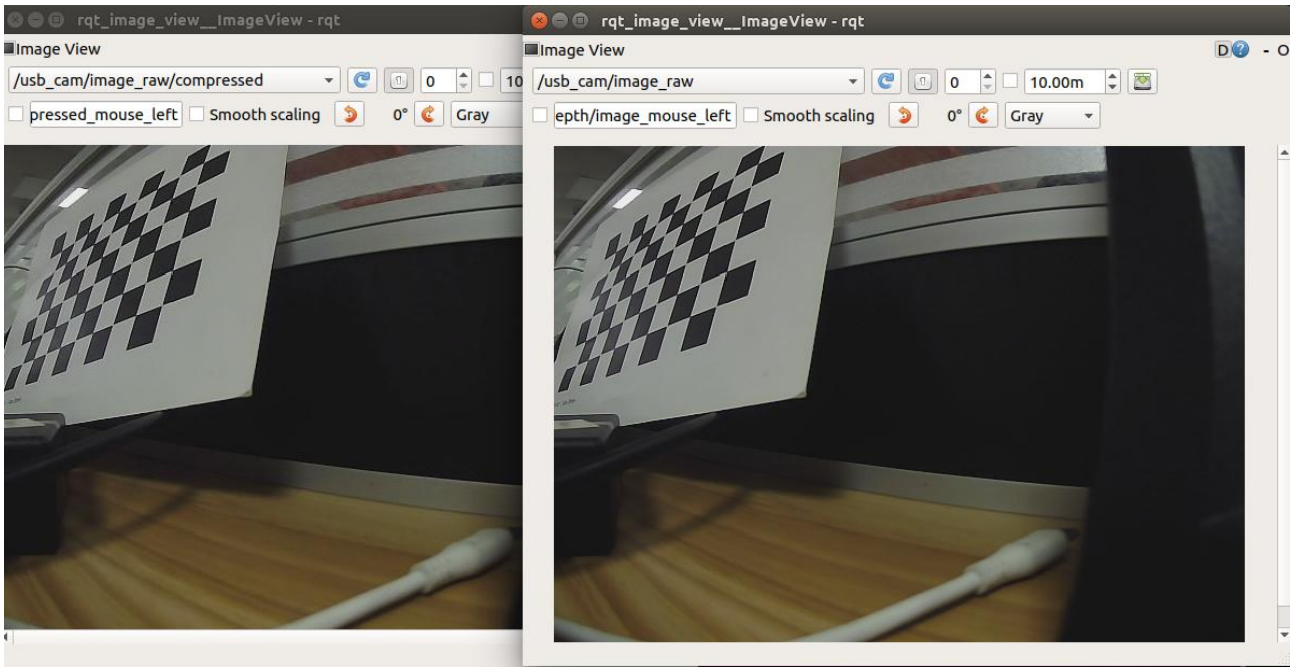
```
rostopic echo /usb_cam/image_raw/encoding
```



话题后面带有【compressed】或【compressedDepth】属于压缩过的话题，ROS 在图像传输时，由于网络、主控运行速度、主控运行内存、视频流数据庞大等因素可能会导致数据丢包，获取不到话题。所以没办法，我只能订阅压缩过的话题。同时打开两个图像订阅不同话题测试，如果设备性能很好，网络也很好的话，看不出什么变化，否则，会发现图像压缩后的话题会流畅很多。

查看图片

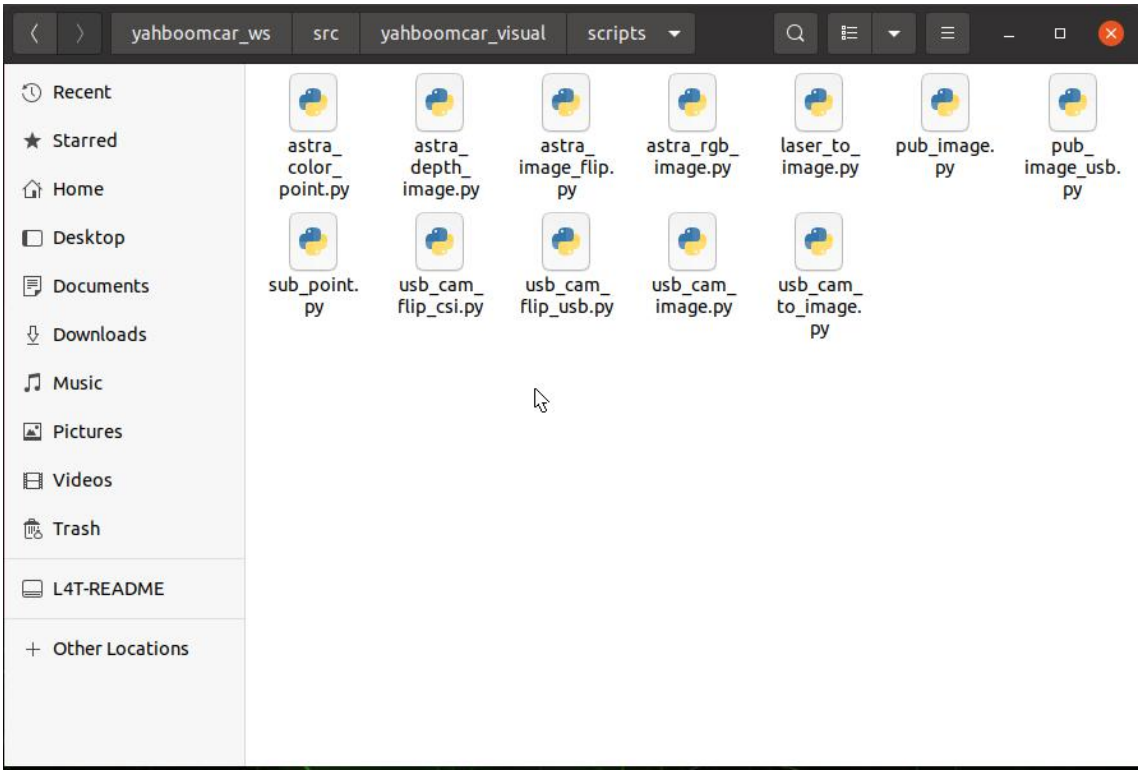
```
rqt_image_view
```



3.2.2、启动彩色图像反转

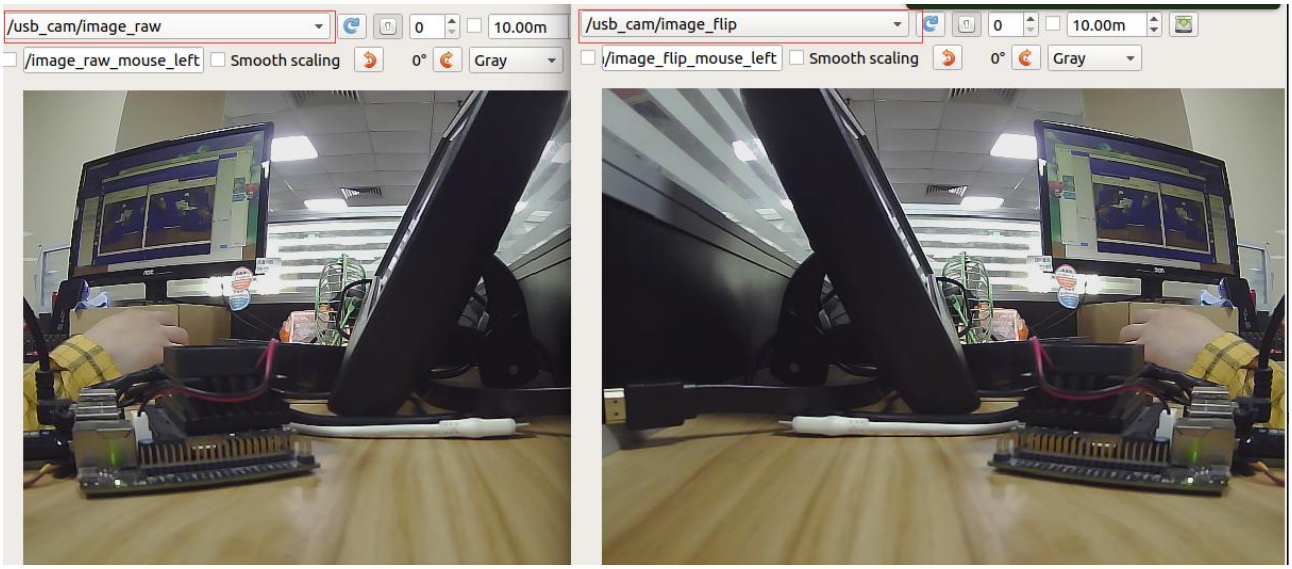
```
roslaunch yahboomcar_visual usb_cam_flip.launch
```

- 此时需要将 usb_cam_flip_usb.py 改成 usb_cam_flip.py



图像查看

```
rqt_image_view
```



- py 代码分析

这里创建了两个订阅者和两个发布者，一个处理一般图像数据，一个处理压缩图像数据。

1）创建订阅者

订阅的话题是【"/usb_cam/image_raw"】，数据类型【Image】，回调函数【topic()】。

订阅的话题是【"/usb_cam/image_raw/compressed"】，数据类型【CompressedImage】，回调函数【compressed_topic()】。

2）创建发布者

发布的话题是【"/usb_cam/image_flip"】，数据类型【Image】，队列大小【10】。

发布的话题是【"/usb_cam/image_flip/compressed"】，数据类型【CompressedImage】，队列大小【10】。

```
sub_img = rospy.Subscriber("/usb_cam/image_raw", Image, topic)
pub_img = rospy.Publisher("/usb_cam/image_flip", Image,
queue_size=10)
sub_comimg = rospy.Subscriber("/usb_cam/image_raw/compressed",
CompressedImage, compressed_topic)
pub_comimg = rospy.Publisher("/usb_cam/image_flip/compressed",
CompressedImage, queue_size=10)
```

3）回调函数

```
# 正常图像传输处理
def topic(msg):
    if not isinstance(msg, Image):
        return
    bridge = CvBridge()
    frame = bridge.imgmsg_to_cv2(msg, "bgr8")
    # Opencv 处理图像
```

```
    frame = cv.resize(frame, (640, 480))
    frame = cv.flip(frame, 1)
    # opencv mat ->    ros msg
    msg = bridge.cv2_to_imgmsg(frame, "bgr8")
    #  图像处理完毕，直接发布
    pub_img.publish(msg)


#  压缩图像传输处理
def compressed_topic(msg):
    if not isinstance(msg, CompressedImage): return
    bridge = CvBridge()
    frame = bridge.compressed_imgmsg_to_cv2(msg, "bgr8")
    # Opencv 处理图像
    frame = cv.resize(frame, (640, 480))
    frame = cv.flip(frame, 1)
    # Create CompressedIamge
    msg = CompressedImage()
    msg.header.stamp = rospy.Time.now()
    #  图像数据转换
    msg.data = np.array(cv.imencode('.jpg', frame)[1]).tostring()
    #  图像处理完毕，直接发布
    pub_comimg.publish(msg)
```

3.3、Jetson CSI 相机

3.3.1、启动 Jetson CSI 相机

```
roslaunch yahboomcar_visual yahboom_csi.launch
```

查看话题

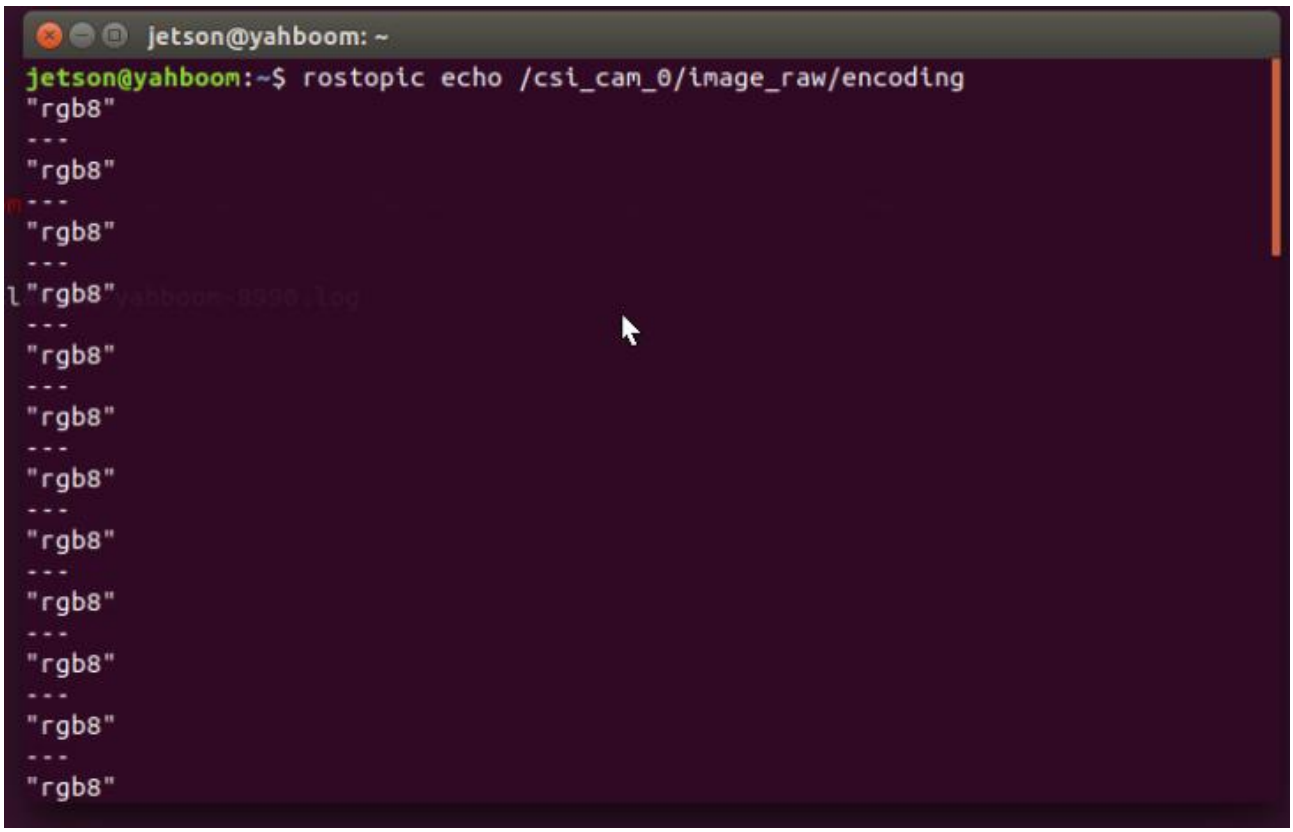
```
rostopic list
```

可以看到话题很多，本节常用的就几个

话题名	数据类型
/csi_cam_0/image_raw	sensor_msgs/Image
/csi_cam_0/image_raw/compressed	sensor_msgs/CompressedImage

查看话题的编码格式：rostopic echo +【topic】+encoding，例如

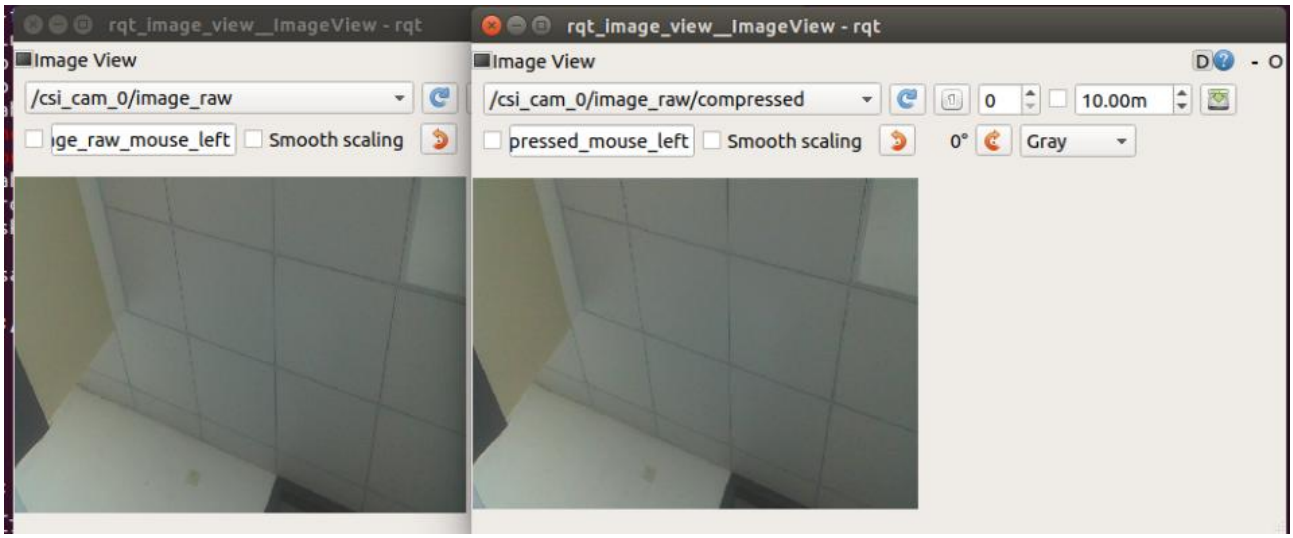
```
rostopic echo /csi_cam_0/image_raw/encoding
```



话题后面带有【compressed】或【compressedDepth】属于压缩过的话题，ROS 在图像传输时，由于网络、主控运行速度、主控运行内存、视频流数据庞大等因素可能会导致数据丢包，获取不到话题。所以没办法，我只能订阅压缩过的话题。同时打开两个图像订阅不同话题测试，如果设备性能很好，网络也很好的话，看不出什么变化，否则，会发现图像压缩后的话题会流畅很多。

查看图片

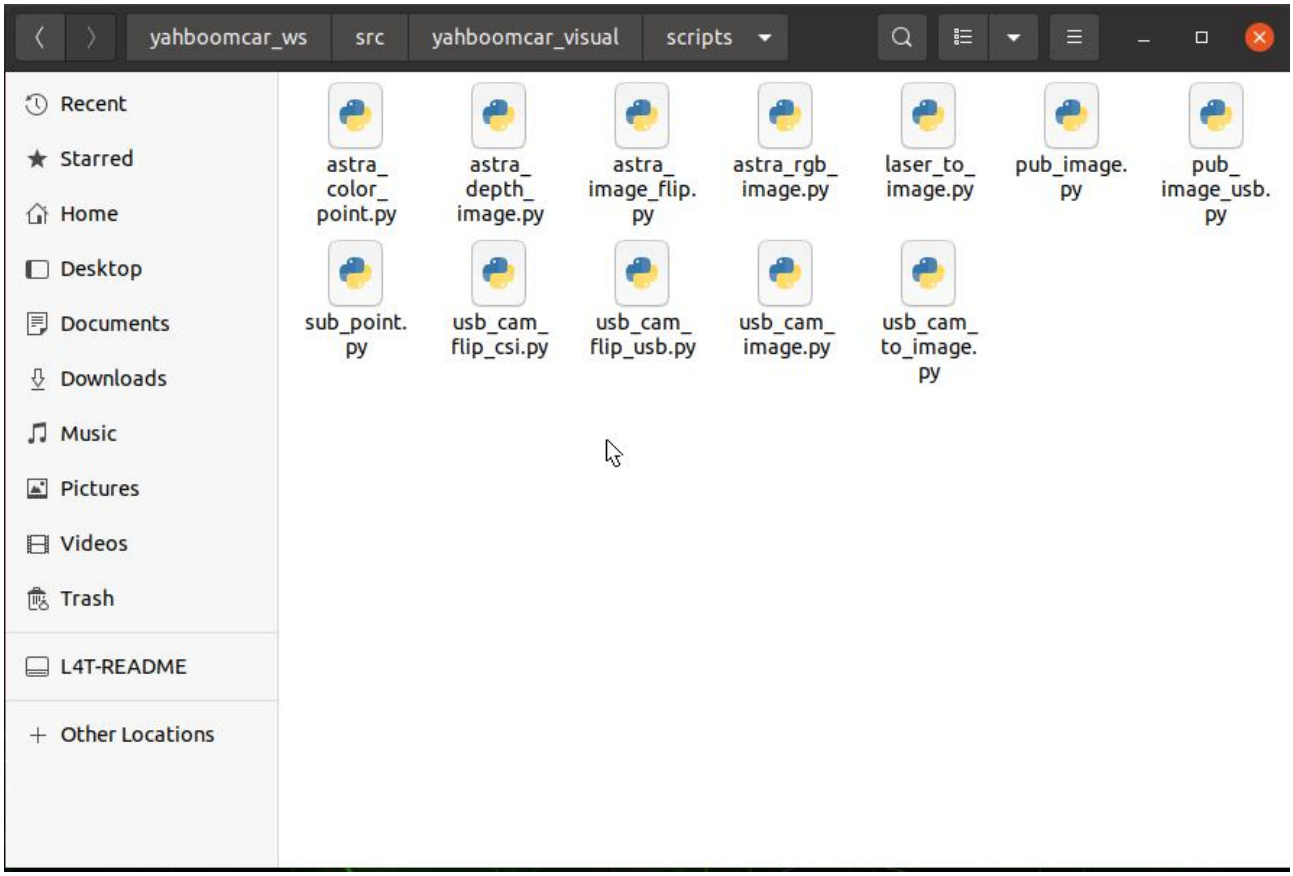
```
rqt_image_view
```



3.2.2、启动彩色图像反转

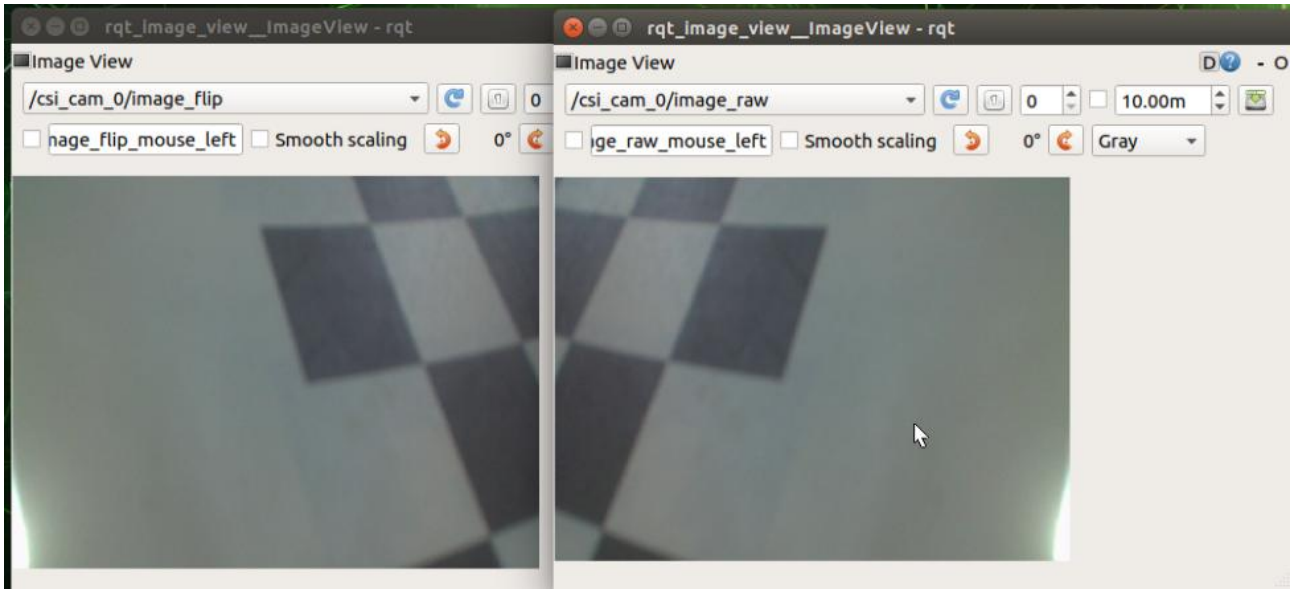
```
roslaunch yahboomcar_visual usb_cam_flip.launch
```

- 此时需要将 usb_cam_flip_csi.py 改成 usb_cam_flip.py



图像查看

```
rqt_image_view
```



- py 代码分析

这里创建了两个订阅者和两个发布者，一个处理一般图像数据，一个处理压缩图像数据。

1）创建订阅者

订阅的话题是【"/csi_cam_0/image_raw"】，数据类型【Image】，回调函数【topic()】。

订阅的话题是【"/csi_cam_0/image_raw/compressed"】，数据类型【CompressedImage】，回调函数【compressed_topic()】。

2）创建发布者

发布的话题是【"/csi_cam_0/image_flip"】，数据类型【Image】，队列大小【10】。

发布的话题是【"/csi_cam_0/image_flip/compressed"】，数据类型【CompressedImage】，队列大小【10】。

```
sub_img = rospy.Subscriber("/csi_cam_0/image_raw", Image, topic)
pub_img = rospy.Publisher("/csi_cam_0/image_flip", Image,
queue_size=10)
sub_comimg =
rospy.Subscriber("/csi_cam_0/image_raw/compressed",
CompressedImage, compressed_topic)
pub_comimg = rospy.Publisher("/csi_cam_0/image_flip/compressed",
CompressedImage, queue_size=10)
```

3）回调函数

```
# 正常图像传输处理
def topic(msg):
    if not isinstance(msg, Image):
        return
    bridge = CvBridge()
    frame = bridge.imgmsg_to_cv2(msg, "bgr8")
```

```
    # Opencv 处理图像
    frame = cv.resize(frame, (640, 480))
    frame = cv.flip(frame, 1)
    # opencv mat ->    ros msg
    msg = bridge.cv2_to_imgmsg(frame, "bgr8")
    #  图像处理完毕，直接发布
    pub_img.publish(msg)


#  压缩图像传输处理
def compressed_topic(msg):
    if not isinstance(msg, CompressedImage): return
    bridge = CvBridge()
    frame = bridge.compressed_imgmsg_to_cv2(msg, "bgr8")
    # Opencv 处理图像
    frame = cv.resize(frame, (640, 480))
    frame = cv.flip(frame, 1)
    # Create CompressedIamge
    msg = CompressedImage()
    msg.header.stamp = rospy.Time.now()
    #  图像数据转换
    msg.data = np.array(cv.imencode('.jpg', frame)[1]).tostring()
    #  图像处理完毕，直接发布
```

```
    pub_comimg.publish(msg)
```

ROS+Opencv 应用

4、ROS+Opencv 应用

4.1、概述

wiki：http://wiki.ros.org/opencv_apps

源码：https://github.com/ros-perception/opencv_apps.git

大部分代码最初取自

https://github.com/Itseez/opencv/tree/master/samples/cpp

功能包：~/software/library_ws/src/opencv_apps

该功能包订阅的话题是【/image】，我们要做的是打开相机节点，写一个将相机的话题转成【/image】节点将【/image】话题发布出来。

开启相机并发布【/image】话题的节点的路径：

```
~/yahboomcar_ws/src/yahboomcar_visual/scripts/pub_image.py
```

opencv_apps 程序提供各种节点，这些节点在内部运行 opencv 的功能，并将结果发布到 ROS 话题中。使用 opencv_apps 程序时，按照自身的业务需求，只需运行一个 launch 文件即可，这样就可以不必再编写这些功能的程序代码。

ROS Wiki 中有相关的节点解析，相应节点的话题订阅与话题发布、相关参数介绍等。详情请查看 ROS WiKi。

4.2、使用

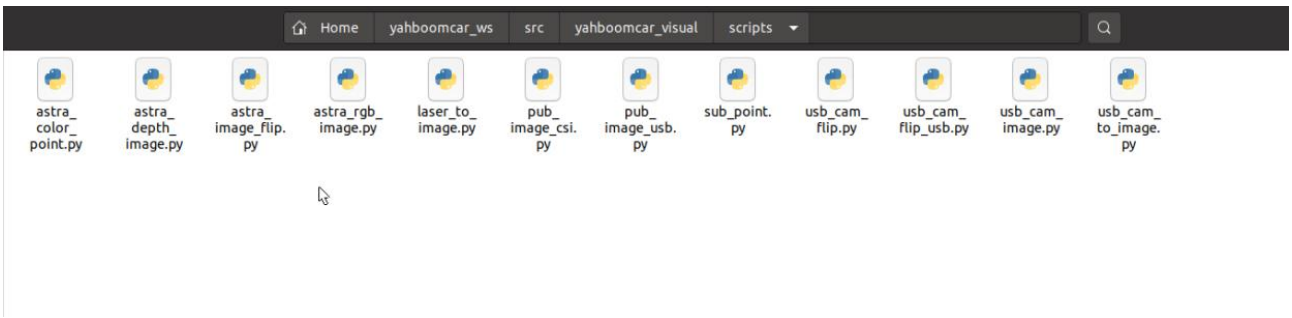
4.2.1、启动

第一步：启动相机

```
roslaunch yahboomcar_visual opencv_apps.launch img_flip:=false
```

如果是单目相机或者树莓派 CSI 相机，需要将 pub_image_usb.py 改成 pub_image.py

如果是 Jetson 相机，则需要将 pub_image_csi.py 改成 pub_image.py



- img_flip 参数：图像是否需要水平翻转，默认 false 就好。

【usb_cam-test.launch】文件默认开启【web_video_server】节点，可以直接使用【IP:8080】网页实时查看图像。

第二步：启动 Opencv_apps 的功能

```
roslaunch opencv_apps face_recognition.launch          # 人脸识别
```

```
roslaunch opencv_apps corner_harris.launch                    # harris 角点检测
roslaunch opencv_apps camshift.launch                         # 目标追踪算法
roslaunch opencv_apps convex_hull.launch                      # 多边形轮廓
roslaunch opencv_apps discrete_fourier_transform.launch       # 离散傅里叶变换算法
roslaunch opencv_apps edge_detection.launch                   # 边缘检测算法
roslaunch opencv_apps face_detection.launch                   # 人脸检测算法
roslaunch opencv_apps fback_flow.launch                       # 光流检测算法
roslaunch opencv_apps find_contours.launch                    # 轮廓检测
roslaunch opencv_apps general_contours.launch                 # 一般轮廓检测
roslaunch opencv_apps goodfeature_track.launch                # 特征点追踪
roslaunch opencv_apps hls_color_filter.launch                 # HLS 颜色过滤
```

```
roslaunch opencv_apps hough_circles.launch                    # 霍夫圆检测
roslaunch opencv_apps hough_lines.launch                      # 霍夫直线检测
roslaunch opencv_apps hsv_color_filter.launch                 # HSV 颜色过滤
roslaunch opencv_apps lk_flow.launch                          # LK 光流算法
roslaunch opencv_apps people_detect.launch                    # 人体检测算法
roslaunch opencv_apps phase_corr.launch                       # 相位相关位移检测
roslaunch opencv_apps pyramids.launch                         # 图像金字塔采样算法
roslaunch opencv_apps rgb_color_filter.launch                 # RGB 颜色过滤
roslaunch opencv_apps segment_objects.launch                  # 清除背景检测算法
roslaunch opencv_apps smoothing.launch                        # 简单过滤器
roslaunch opencv_apps threshold.launch                        # 阈值图像处理
```

```
roslaunch opencv_apps watershed_segmentation.launch        # 分水岭分割算法
```

几乎每个功能案例都会有一个参数【debug_view】，布尔类型，是否使用Opencv 显示图片，默认显示。

如果不需要显示则设置为【False】，例如

```
roslaunch opencv_apps contour_moments.launch debug_view:=False
```

但是这样子启动后，有些案例肯不能通过其他方式显示出来，因为在源码中，有些【debug_view】设置为【False】，就会把图像处理给关闭掉。

4.2.2、显示方法

- rqt_image_view

输入以下命令，选择对应话题

```
rqt_image_view
```

- opencv

系统默认显示，无需做什么处理。

- 网页观看

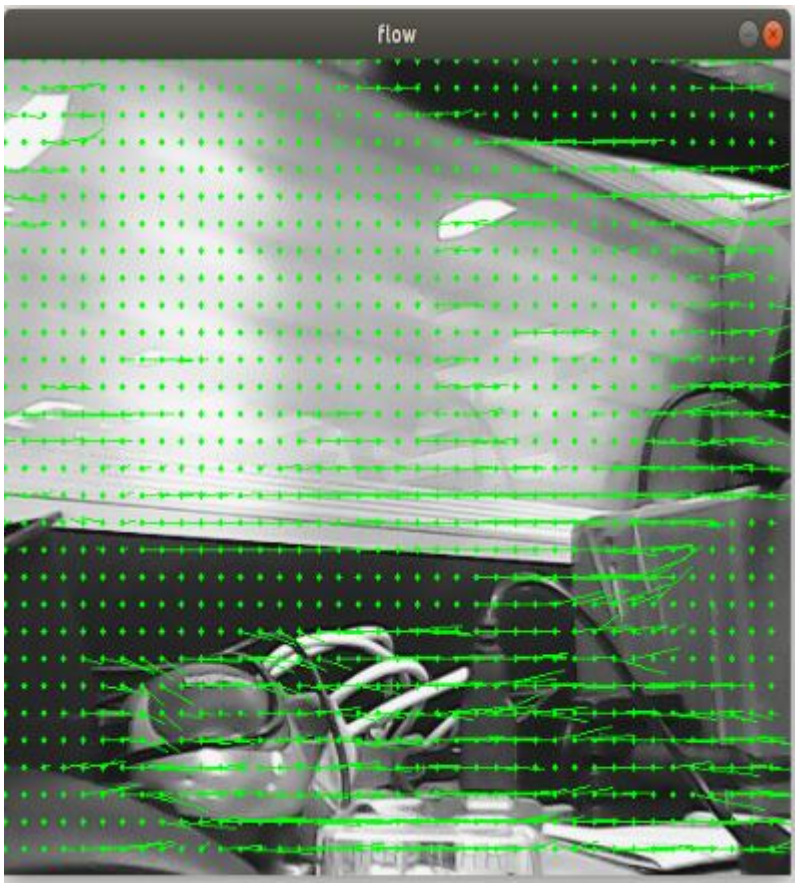
（同局域网下）再浏览器中输入 IP+port，例如：

192.168.2.79:8080

4.2.3、效果显示

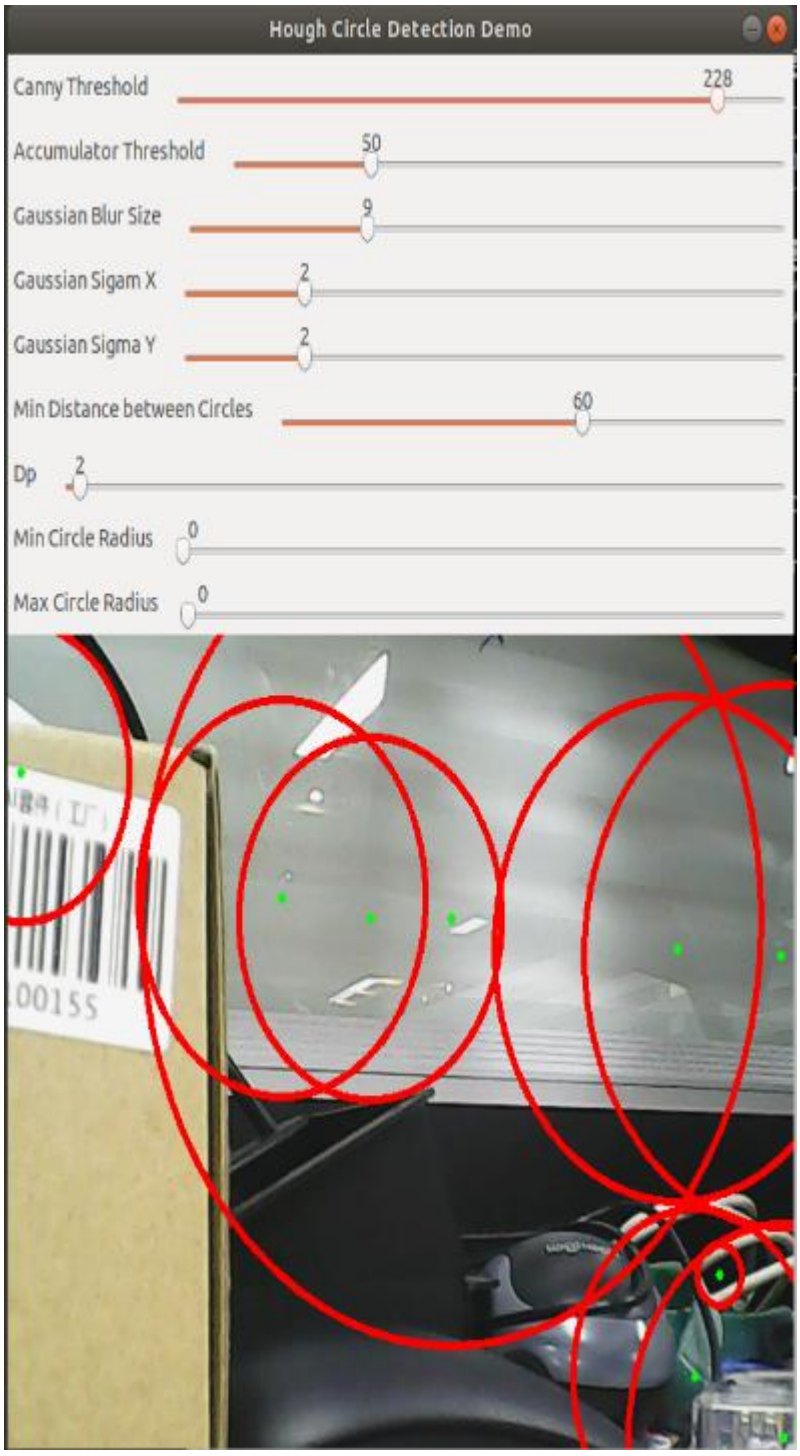
- 光流检测算法

移动画面，观察现象。



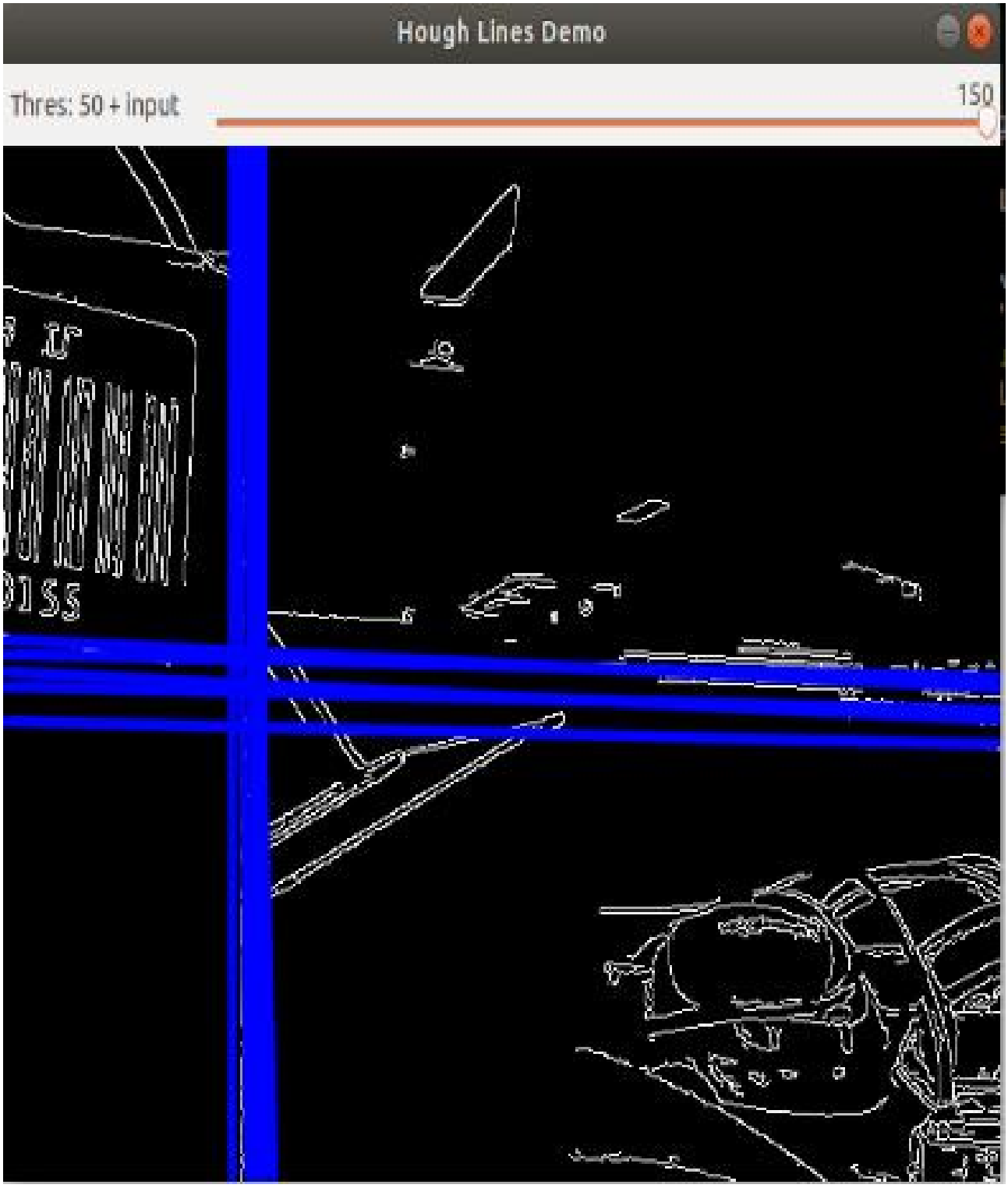
- 特征点追踪

- 霍夫圆检测



- 霍夫直线检测

阈值越低，线条越多，画面越容易卡。

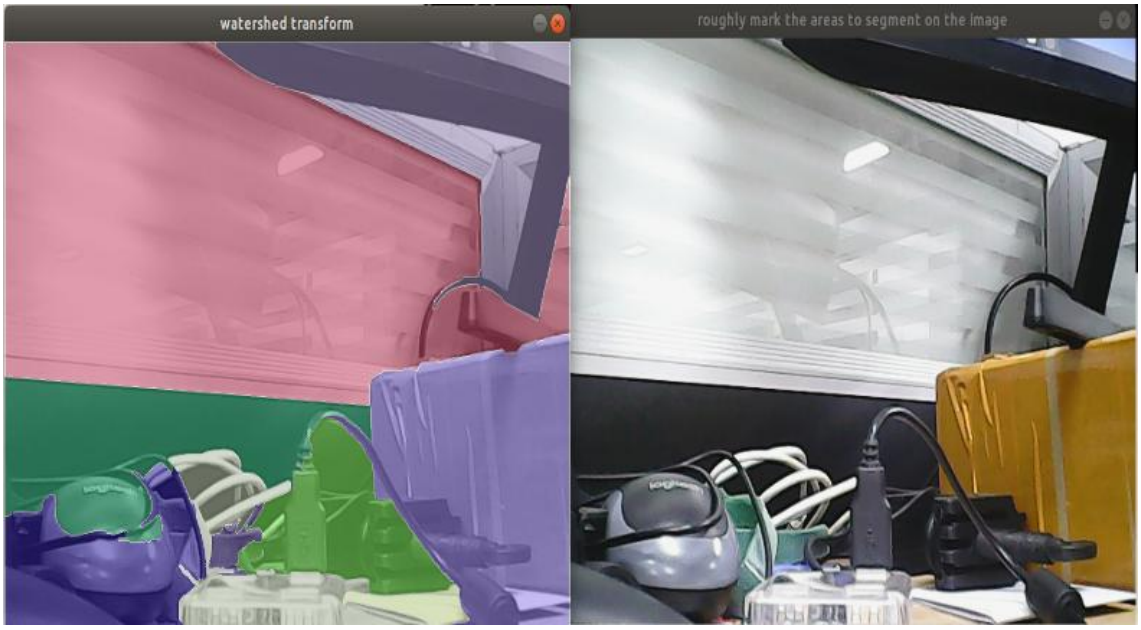


- 相位相关位移检测

相机移动越快，圆的半径越大。

- 分水岭分割算法

使用鼠标选中不同物体，系统自动区分出来。



4.3、节点

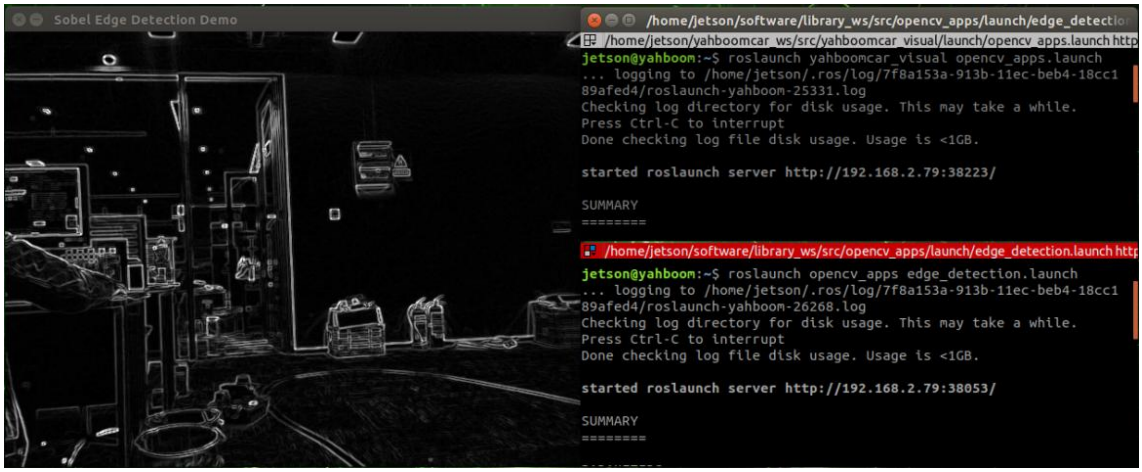
本节每个案例都会有一个订阅图像和发布图像的话题。

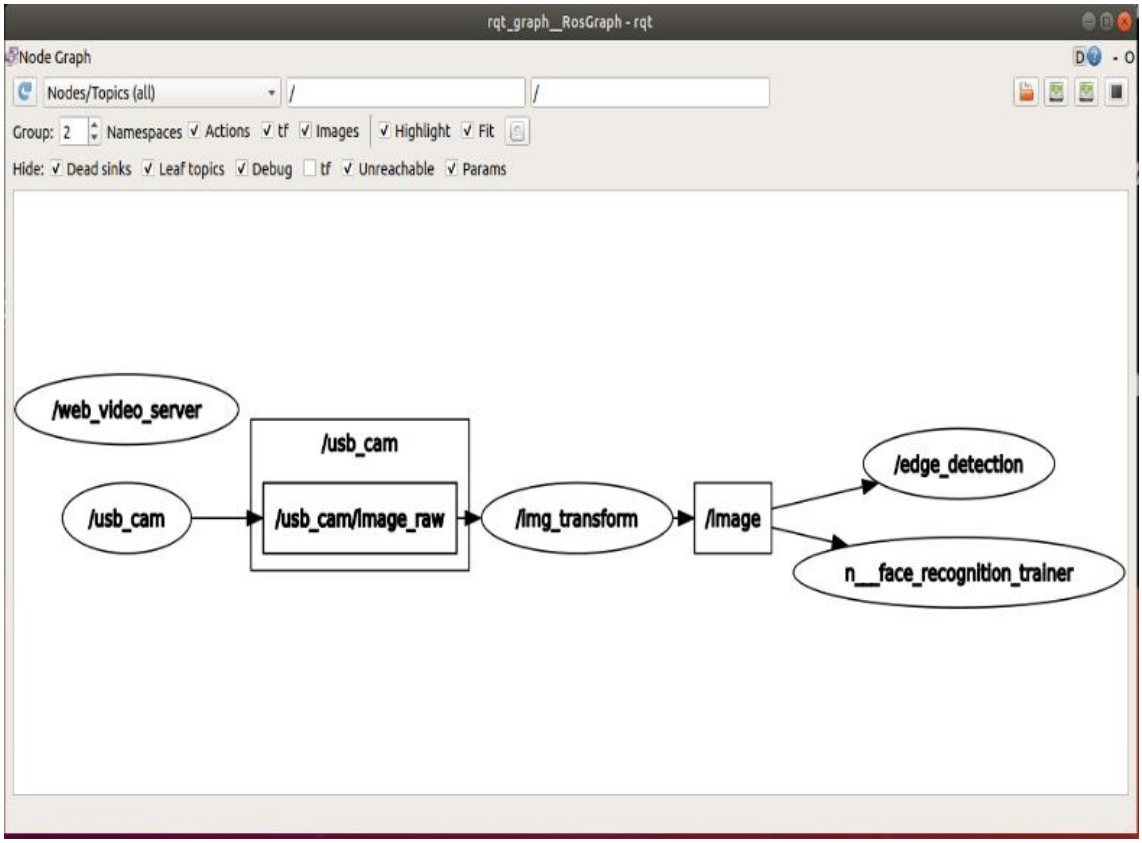
4.3.1、边缘检测算法

参数	类型	默认	解析
~use_camera_info	bool	true	订阅【camera_info】这个话题，获得默认坐标系 ID，否则直接使用图像信息。
~debug_view	bool	false	是否创建窗口显示该节点图像

参数	类型	默认	解析
~edge_type	int	0	指定边缘检测方法：0：索贝尔算子，1：拉普拉斯算子，2：Canny 边缘检测
~canny_threshold1	int	100	指定第二个 canny 阈值
~canny_threshold2	int	200	指定第一个 canny 阈值
~apertureSize	int	3	索贝尔算子的孔径大小。
~apply_blur_pre	bool	True	是否将 blur()应用于输入图像
~postBlurSize	double	3.2	输入图像孔径大小
~apply_blur_post	bool	False	是否将 GaussianBlur() 应用于输入图像

参数	类型	默认	解析
~L2gradient	bool	False	canny 的参数
~queue_size	int	3	队列大小





4.3.2、轮廓矩

参数	类型	默认	解析
~use_camera_info	bool	true	订阅【camera_info】这个话题，获得默认坐标系 ID，否则直接使用图像信息。
~debug_view	bool	false	是否创建窗口显示该节点图像
~canny_low_threshold	int	0	Canny 边缘检测低阈值
~queue_size	int	3	队列大小

4.3.3、人脸识别

此案例是通过即时收集人的图像自主训练，实时识别，步骤略有复杂。

参数	类型	默认	解析
~approximate_sync	bool	false	订阅【camera_info 这个话题，获得 认坐标系 ID,否 直接使用图像 息。
~queue_size	int	100	订阅话题的队列 大小
~model_method	string	"eigen"	人脸识别的方法 "eigen", "fisher or "LBPH"
~use_saved_data	bool	true	从~data_dir 路 下加载训练数据
~save_train_data	bool	true	将训练数据保存 到~data_dir 路 下，以便重新训
~data_dir	string	"~/opencv_apps/face_data"	保存训练数据路

参数	类型	默认	解析
			径
~face_model_width	int	190	训练人脸图像的 宽度
~face_model_height	int	90	训练人脸图像的 高度
~face_padding	double	0.1	每个人脸的填充 比
~model_num_components	int	0	人脸识别器模型 的组件数量（0 视为无限制）
~model_threshold	double	8000.0	人脸识别模型阈 值
~lbph_radius	int	1	半径参数（仅适 于 LBPH 方法）
~lbph_neighbors	int	8	邻域参数（仅适 于 LBPH 方法）
~lbph_grid_x	int	8	网格 x 参数（仅 用于 LBPH 方法

参数	类型	默认	解析
~lbph_grid_y	int	8	网格 y 参数 (仅用于 LBPH 方
~queue_size	int	100	图像订阅者队列大小

操作步骤：

1、首先在下图冒号后面，输入人物名字：Yahboom

2、确认名字：y

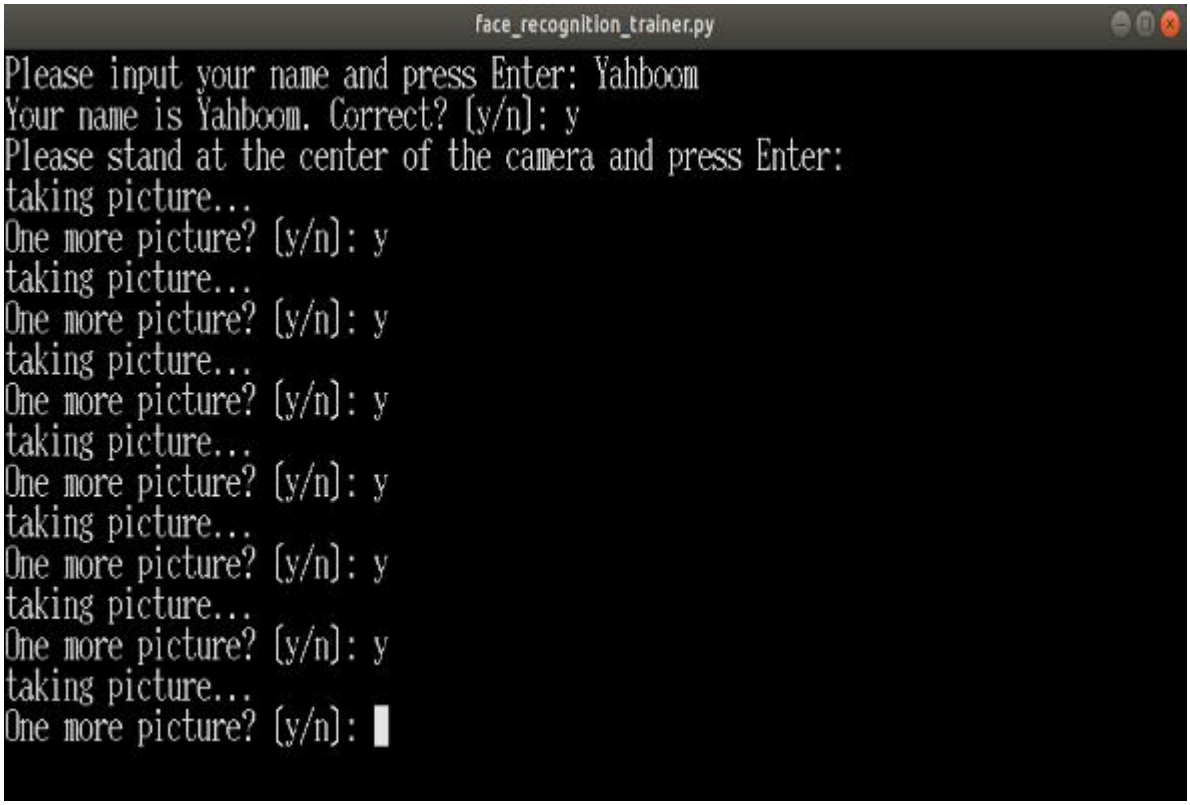
3、然后将人脸放在图像中央，点击确认。

4、循环增加一张照片：y，点击确认。

5、结束图片收集，输入：n，点击确认。

6、关闭这个 launch 文件，重启。

如果需要录入所认识别，依次循环 1~5，直到所有识别人员录入完毕，再执行第六步。



第 3 步：要保证能够识别到人脸

最终识别效果

4、Mediapipe 开发

5、MediaPipe 开发

5、MediaPipe 开发 5.1、简介 5.2、使用 5.3、MediaPipe Hands5.4、MediaPipe Pose5.5、dlib

mediapipe github：https://github.com/google/mediapipe

mediapipe 官网：https://google.github.io/mediapipe/

dlib 官网：http://dlib.net/

dlib github：https://github.com/davisking/dlib

5.1、简介

MediaPipe 是一款由 Google 开发并开源的数据流处理机器学习应用开发框架。它是一个基于图的数据处理管线，用于构建使用了多种形式的数据源，如视频、音频、传感器数据以及任何时间序列数据。 MediaPipe 是跨平台的，可以运行在嵌入式平台(树莓派等)，移动设备(iOS 和 Android)，工作站和服务器上，并支持移动端 GPU 加速。 MediaPipe 为实时和流媒体提供跨平台、可定制的 ML 解决方案。

MediaPipe 的核心框架由 C++ 实现，并提供 Java 以及 Objective C 等语言的支持。MediaPipe 的主要概念包括数据包(Packet)、数据流(Stream)、计算单元（Calculator）、图（Graph）以及子图（Subgraph）。

MediaPipe 的特点：

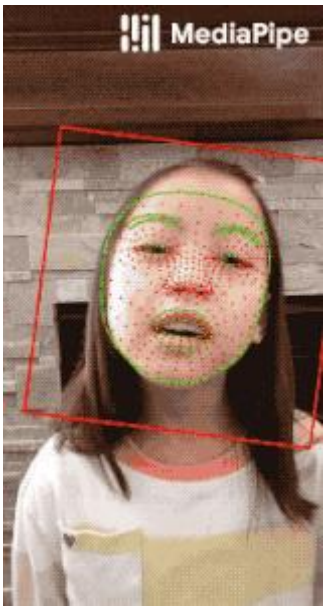
端到端加速：内置的快速 ML 推理和处理即使在普通硬件上也能加速。

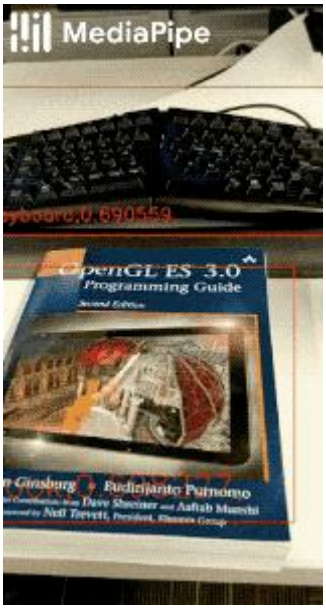
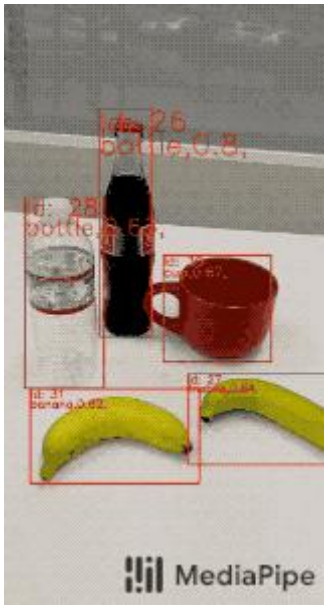
一次构建，随时随地部署：统一解决方案适用于 Android、iOS、桌面/云、web 和物联网。

即用解决方案：展示框架全部功能的尖端 ML 解决方案。

免费开源：Apache2.0 下的框架和解决方案，完全可扩展和定制。

MediaPipe 中的深度学习解决方案

Face Detection	Face Mesh	Iris	Hands	Pose
				

Face Detection	Face Mesh	Iris	Hands	Pose
Hair Segmentation	Object Detection	Box Tracking	Instant Motion Tracking	Objectr
				

	Android	iOS	C++	Python	JS	Coral
Face Detection	✅	✅	✅	✅	✅	✅
Face Mesh	✅	✅	✅	✅	✅	
Iris	✅	✅	✅			
Hands	✅	✅	✅	✅	✅	
Pose	✅	✅	✅	✅	✅	

	Android	iOS	C++	Python	JS	Coral
Holistic	✅	✅	✅	✅	✅	
Selfie Segmentation	✅	✅	✅	✅	✅	
Hair Segmentation	✅		✅			
Object Detection	✅	✅	✅			✅
Box Tracking	✅	✅	✅			
Instant Motion Tracking	✅					
Objectron	✅		✅	✅	✅	
KNIFT	✅					
AutoFlip			✅			
MediaSequence			✅			
YouTube 8M			✅			

5.2、使用

```
---------------------------------- ROS
----------------------------------------
roslaunch yahboomcar_mediapipe cloud_Viewer.launch          # 点云查看：支持 01~04
roslaunch yahboomcar_mediapipe 01_HandDetector.launch       # 手部检测
roslaunch yahboomcar_mediapipe 02_PoseDetector.launch       # 姿态检测
roslaunch yahboomcar_mediapipe 03_Holistic.launch           # 整体检测
roslaunch yahboomcar_mediapipe 04_FaceMesh.launch           # 面部检测
roslaunch yahboomcar_mediapipe 05_FaceEyeDetection.launch   # 人脸识别
```

如果使用的是单目相机或者树莓派 CSI 相机则分别需要将

01_HandDetector_usb.py、02_PoseDetector_usb.py、

03_Holistic_usb.py、04_FaceMesh_usb.py、

05_FaceEyeDetection_usb.py 名称改成 01_HandDetector.py、

02_PoseDetector.py、03_Holistic.py、04_FaceMesh.py、

05_FaceEyeDetection.py

如果使用的是 jetson CSI 相机则分别需要将

01_HandDetector_csi.py、02_PoseDetector_csi.py、03_Holistic_csi.py、04_FaceMesh_csi.py、05_FaceEyeDetection_csi.py 名称改成 01_HandDetector.py、02_PoseDetector.py、03_Holistic.py、04_FaceMesh.py、05_FaceEyeDetection.py

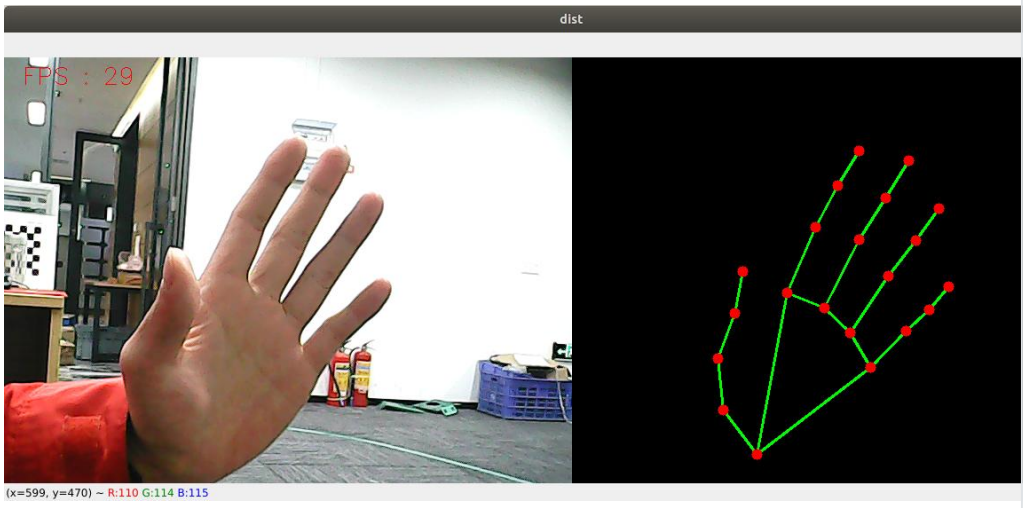
```
-------------------------------- not ROS ----------------------------------------
cd ~/yahboomcar_ws/src/yahboomcar_mediapipe/scripts     # 进入源码所在目录
python3 06_FaceLandmarks.py                              # 人脸特效
python3 07_FaceDetection.py                              # 人脸检测
python3 08_Objectron.py                                  # 三维物体识别
python3 09_VirtualPaint.py                               # 画笔
python3 10_HandCtrl.py                                   # 手指控制
python3 11_GestureRecognition.py                         # 手势识别
```

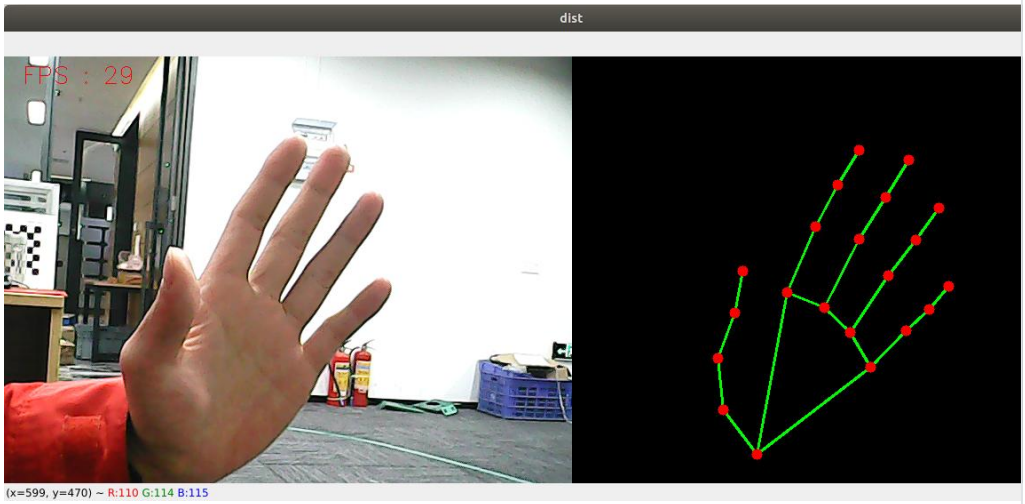
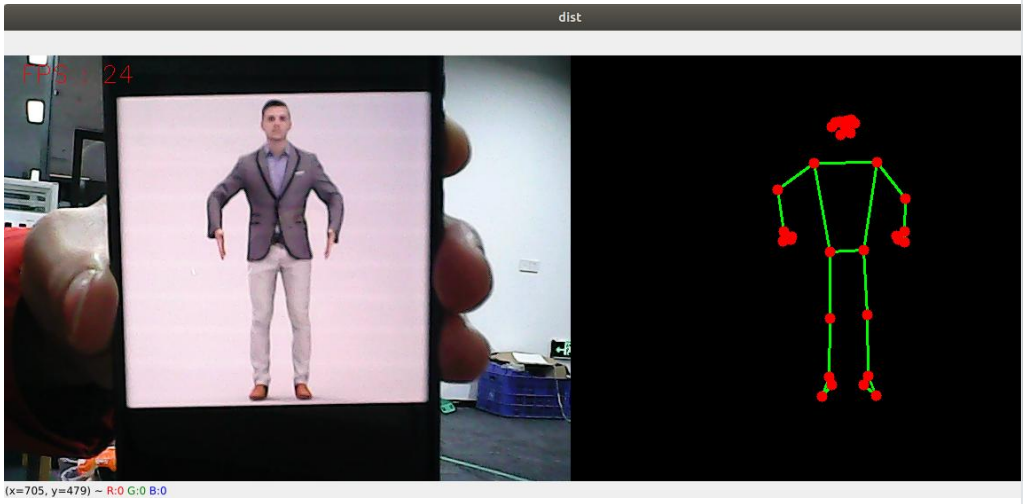
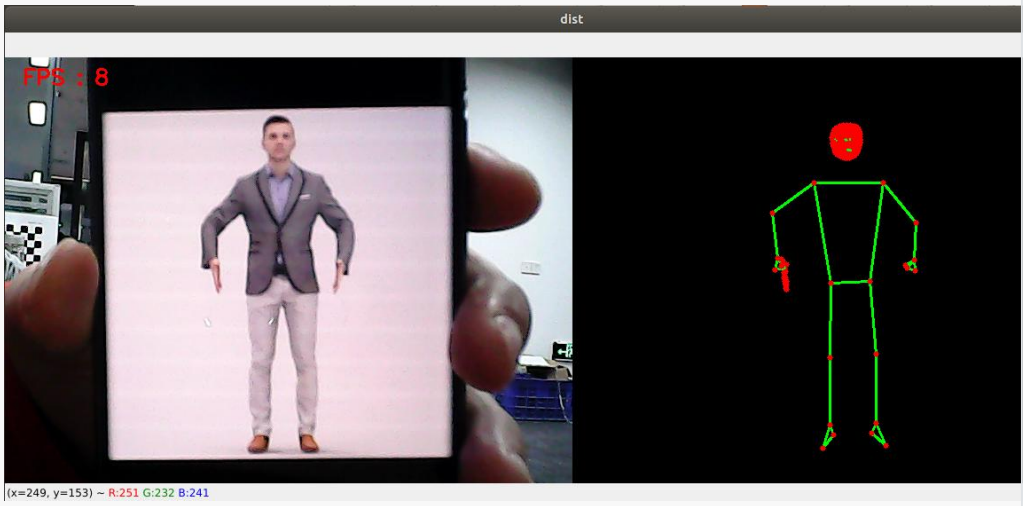
在使用过程中，需注意的有

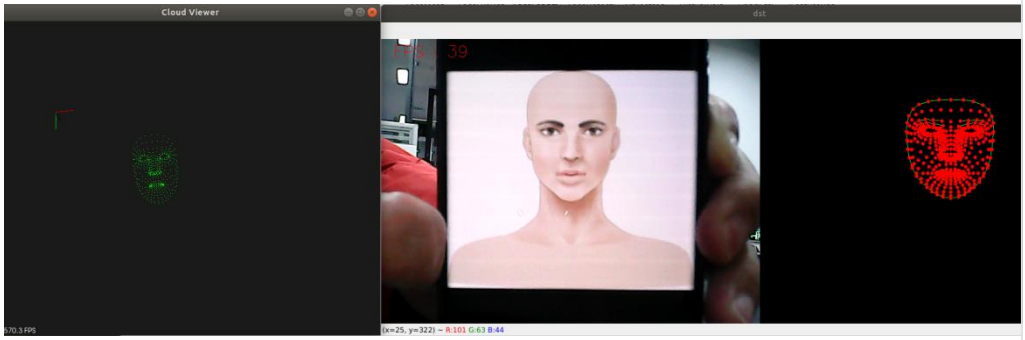
- 手部检测、姿态检测、整体检测、面部检测均具备点云查看功能，以面部检测为例。
- 所有功能【q 键】为退出。
- 整体检测：包括手部、脸部、身体姿态检测。

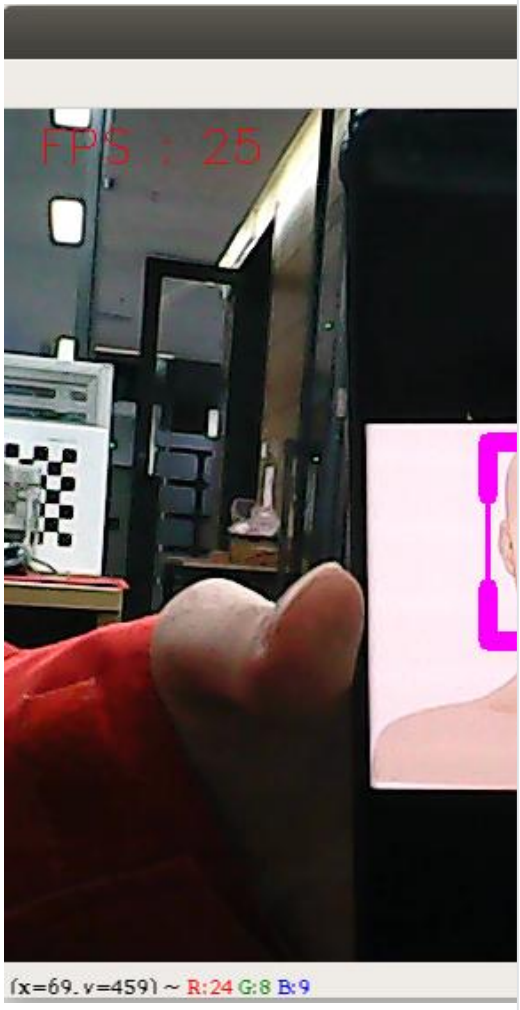
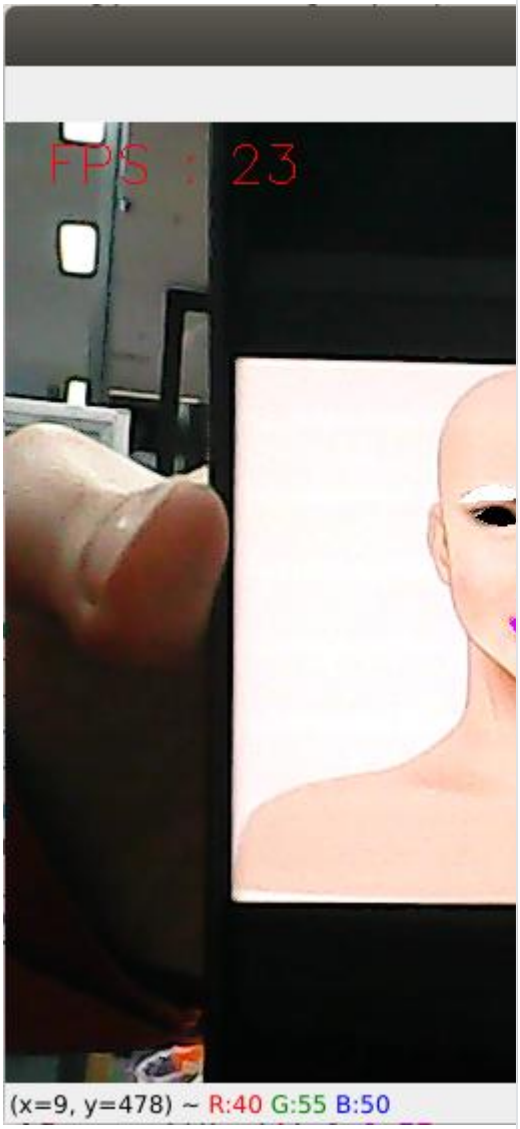
- 三维物体识别：可识别的物体有：【'Shoe', 'Chair', 'Cup', 'Camera'】，一共 4 类；点击【f 键】切换识别物体；jetson 系列不可用键盘按键，切换识别物体需改源码中的【self.index】参数。
- 画笔：右手食指和中指合并时是选择状态，同时弹出颜色选框，两指尖移动到对应颜色位置时，选中该颜色（黑色为橡皮擦）；食指和中指分开始是绘画状态，可在画板上任意绘制。
- 手指控制：点击【f 键】切换识别效果。
- 手指识别：以右手为准设计的手势识别，满足特定条件时，均可以准确识别。可识别的手势有：【Zero、One、Two、Three、Four、Five、Six、Seven、Eight、Ok、Rock、Thumb_up（点赞）、Thumb_down（拇指向下）、Heart_single（单手比心）】，一共 14 类。

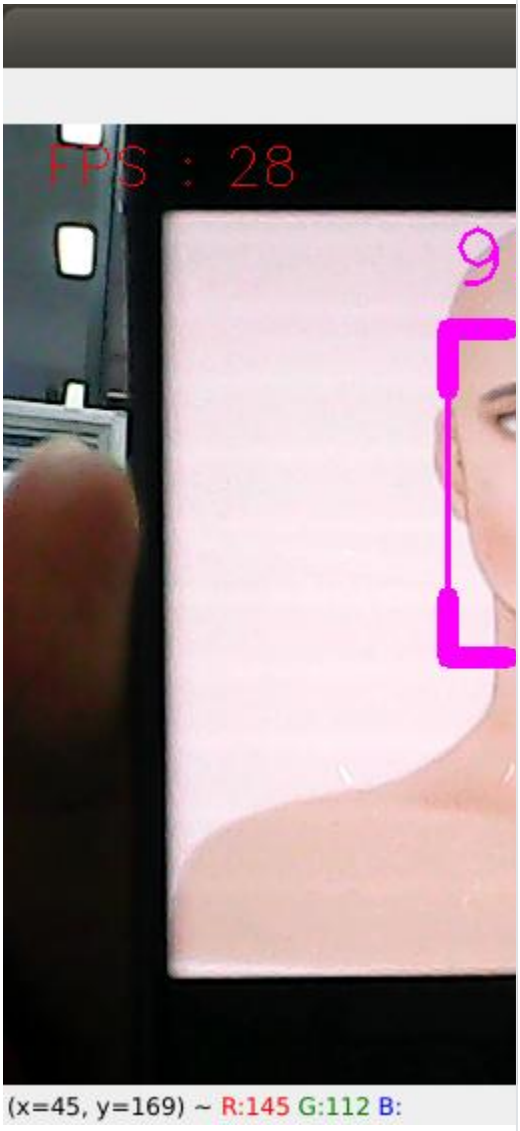
01、手部检测



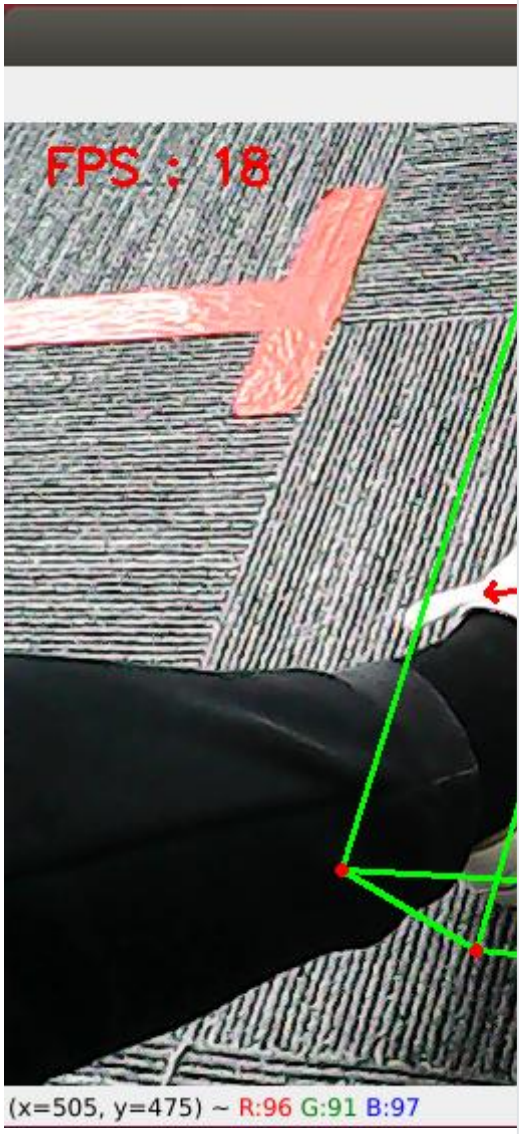
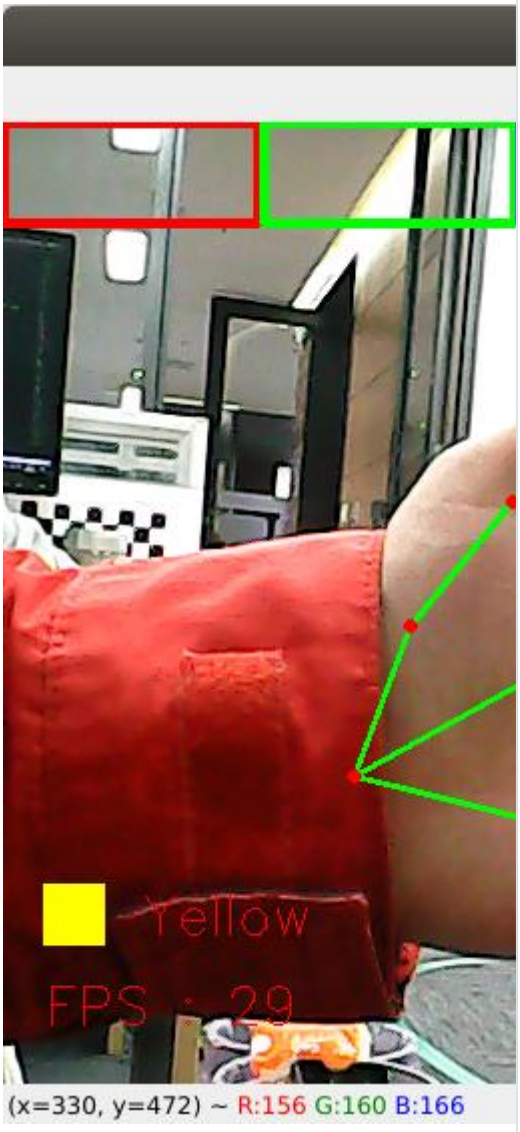
01、手部检测	
02、姿态检测	
03、整体检测	

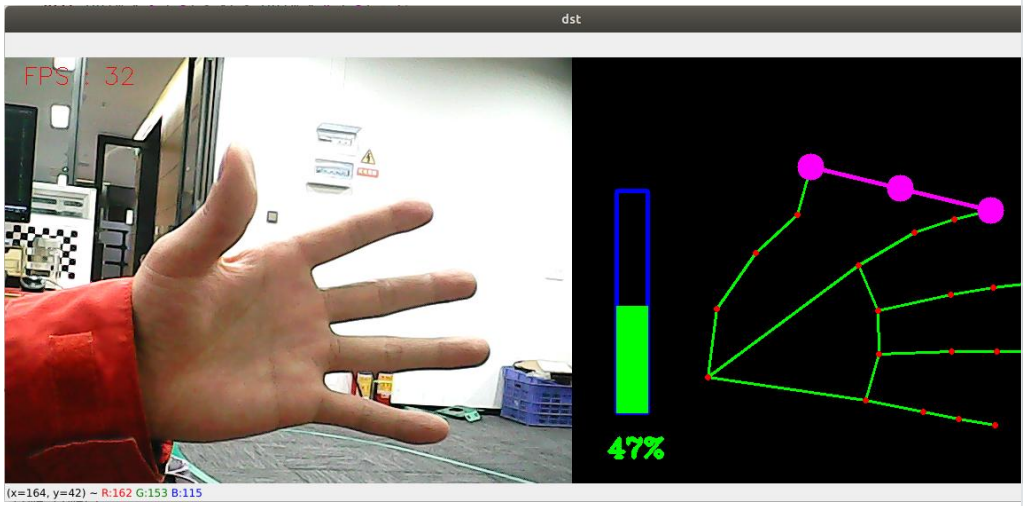
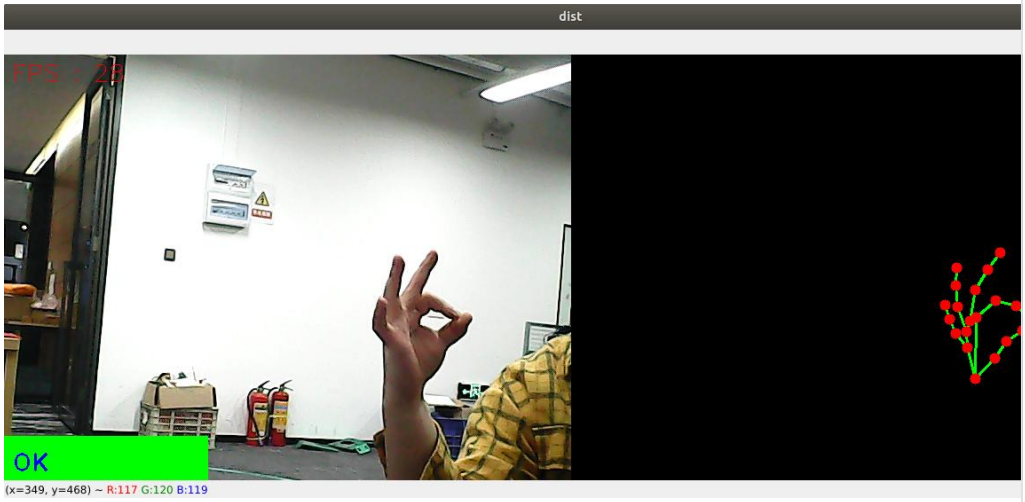
04、面部检测	
05、人脸识别	06、人脸特效
07、人脸检测	

05、人脸识别	06、人脸特效	07、人脸检测
		

05、人脸识别	06、人脸特效	07、人脸检测
		

05、人脸识别	06、人脸特效	07、人脸检测
08、三维物体识别		09、画笔

05、人脸识别	06、人脸特效	07、人脸检测
		

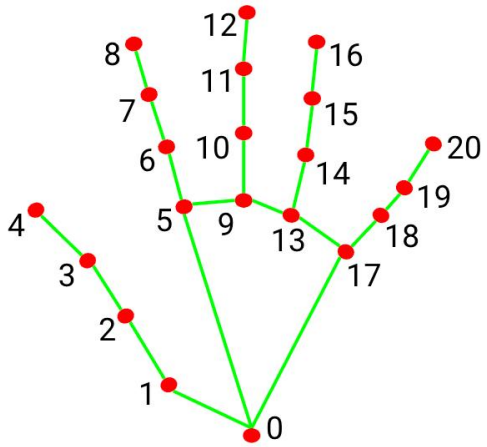
10、手指控制	
11、手势识别	

5.3、MediaPipe Hands

MediaPipe Hands 是一款高保真的手和手指跟踪解决方案。它利用机器学习（ML）从一帧中推断出 21 个手的 3D 坐标。

在对整个图像进行手掌检测后，根据手部标记模型通过回归对检测到的手区域内的 21 个 3D 手关节坐标进行精确的关键点定位，即直接坐标预测。该模型学习一致的内部手姿势表示，甚至对部分可见的手和自我遮挡也具有鲁棒性。

为了获得地面真实数据，用了 21 个 3D 坐标手动注释了约 30K 幅真实世界的图像，如下所示 (从图像深度图中获取 Z 值，如果每个对应坐标都有 Z 值)。为了更好地覆盖可能的手部姿势，并对手部几何体的性质提供额外的监督，还绘制了各种背景下的高质量合成手部模型，并将其映射到相应的 3D 坐标。

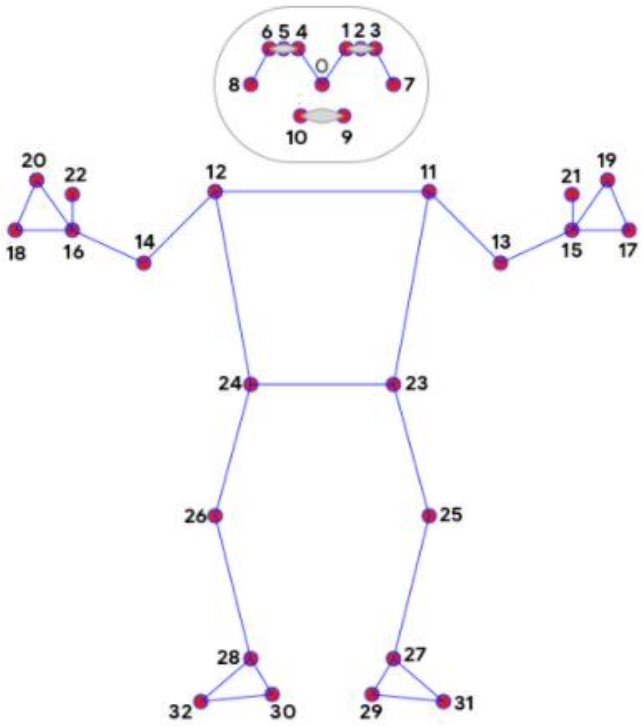


0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

5.4、MediaPipe Pose

MediaPipe Pose 是一个用于高保真身体姿势跟踪的 ML 解决方案，利用 BlazePose 研究，从 RGB 视频帧推断出 33 个 3D 坐标和全身背景分割遮罩，该研究也为 ML Kit 姿势检测 API 提供了动力。

MediaPipe 姿势中的地标模型预测了 33 个姿势坐标的位置（参见下图）。



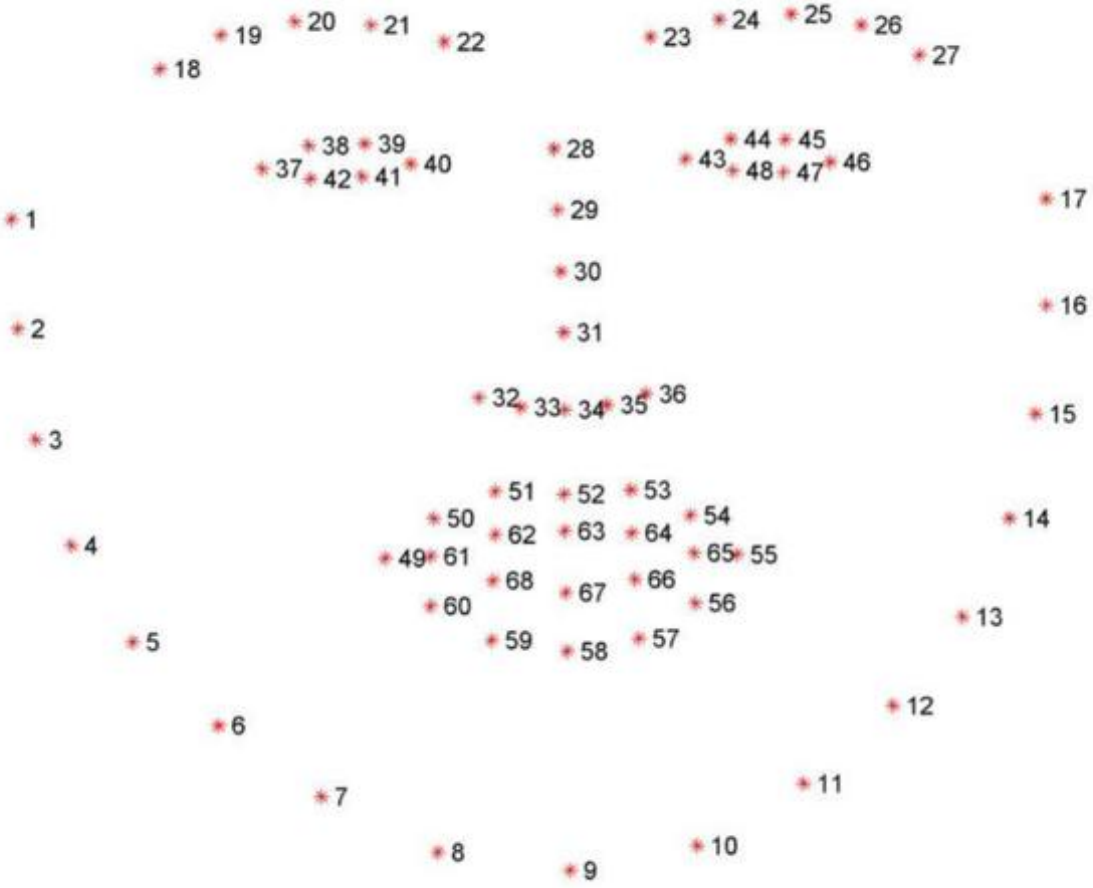
0. nose
1. left_eye_inner
2. left_eye
3. left_eye_outer
4. right_eye_inner
5. right_eye
6. right_eye_outer
7. left_ear
8. right_ear
9. mouth_left
10. mouth_right
11. left_shoulder
12. right_shoulder
13. left_elbow
14. right_elbow
15. left_wrist
16. right_wrist
17. left_pinky
18. right_pinky
19. left_index
20. right_index
21. left_thumb
22. right_thumb
23. left_hip
24. right_hip
25. left_knee
26. right_knee
27. left_ankle
28. right_ankle
29. left_heel
30. right_heel
31. left_foot_index
32. right_foot_index

5.5、dlib

对应的案例是人脸特效。

DLIB 是一个现代 C++工具包，包含机器学习算法和工具，用于在 C++中创建复杂的软件来解决现实世界问题。它被工业界和学术界广泛应用于机器人、嵌入式设备、移动电话和大型高性能计算环境等领域。

dlib 库采用 68 点位置标志人脸重要部位，比如 18-22 点标志右眉毛，51-68 标志嘴巴。使用 dlib 库的 get_frontal_face_detector 模块探测出人脸，使用 shape_predictor_68_face_landmarks.dat 特征数据预测人脸特征数值



第七章 ROS2 进阶教程

1. Docker 使用教程

2. docker 概述和 docker 安装

1、docker 概述和 docker 安装

Docker 官网：http://www.docker.com

Docker 中文网站：https://www.docker-cn.com

Docker Hub（仓库）官网：https://hub.docker.com

1.1、docker 概述

docker 是一个应用容器引擎项目，基于 go 语言开发，开源。

目前 ros2 的课程全部置于 docker 容器内，客户可以体验学习使用容器化的开发方法。

1.1.1、docker 为什么会出现

先提几个场景：

1. 运维帮你开发的项目部署到服务器上，告诉你有问题启动不起来。你在本地跑了一下发现没问题...
2. 要上线的项目因为一些软件的版本的更新，导致不可用了...

3. 有项目涉及到的环境内容非常多，各种中间件，各种配置，还要部署好多台服务器...

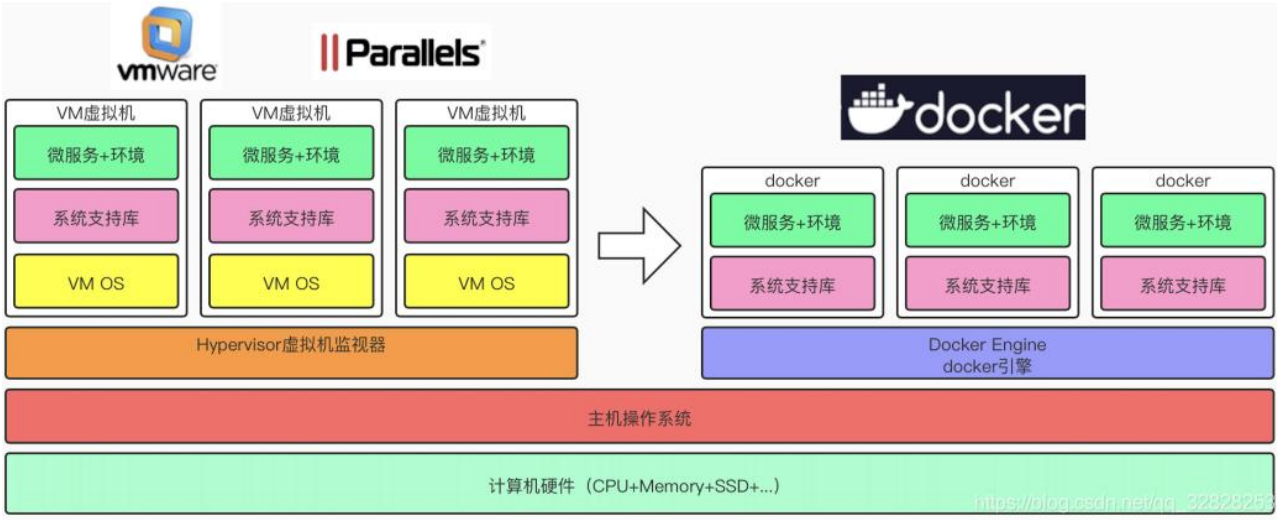
这些问题其实总结起来就是跟环境有关。 要避开各种因环境不同导致的问题，那么最好是在部署项目的时候，连同项目所需要的各种环境一起部署了最好。比如，项目中涉及到 redis、mysql、jdk、es 等环境，在部署 jar 包的时候把整个环境都带上。那么问题来了，怎么样能让项目带上环境一起呢?

Docker 就是来解决这个问题的!

1.1.2、Docker 的核心思想

这个就是 docker 的 logo，一条装满集装箱的鲸鱼，在鲸鱼背上，集装箱相互之间是隔离的，这也就是 docker 的核心思想了。 比如之前有多个应用在同一台服务器上运行，可能会有软件的端口占用冲突，现在隔离后就可以独自运行了。另外，docker 可以最大化的利用服务器的能力。

1.1.3、对比虚拟机与 Docker



docker 守护进程可以直接与主操作系统进行通信，为各个 docker 容器分配资源；它还可以将容器与主操作系统隔离，并将各个容器互相隔离。虚拟机启动需要数分钟，而 docker 容器可以在数毫秒内启动。由于没有臃肿的从操作系统，docker 可以节省大量的磁盘空间以及其他系统资源。

虚拟机更擅长于彻底隔离整个运行环境。例如，云服务提供商通常采用虚拟机技术隔离不同的用户。而 docker 通常用于隔离不同的应用，例如前端，后端以及数据库。

docker 容器比虚拟机更加节省资源，速度更加快（启动、关闭、新建、删除）

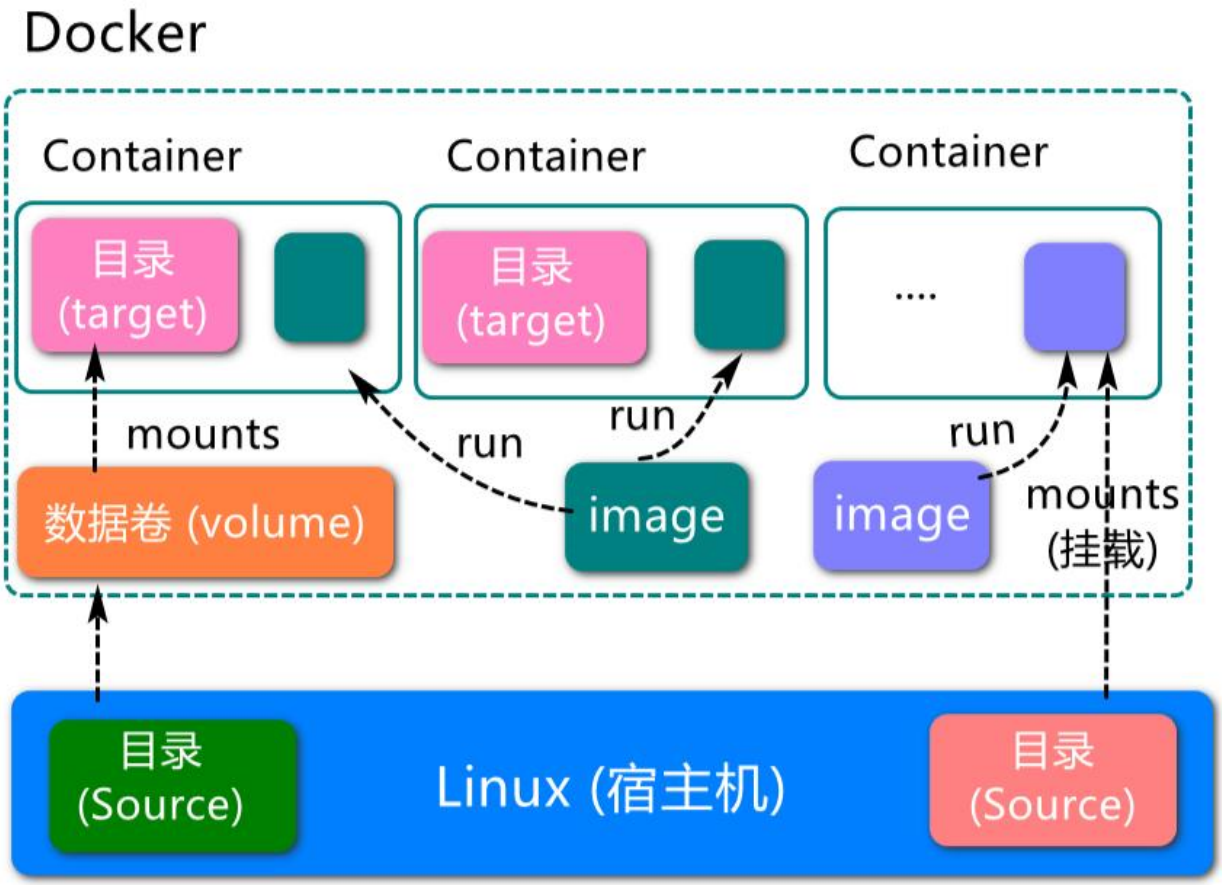
1.1.4、docker 架构

docker 使用客户端-服务器（client-server）架构。docker 客户端（client）与 docker 守护进程（daemon）通信，后者负责构建、运行和分发 docker 容器的重担。docker 客户端和守护进程可以在同一系统上运行，也可以将

docker 客户端连接到远程 docker 守护进程。docker 客户端和守护进程通过 UNIX 套接字或网络接口使用 REST API 进行通信。另一个 docker 客户端是 docker compose，它让你可以处理由一组容器组成的应用程序。

- docker client 是安装完 docker 之后，直接使用的 docker 命令。
- docker host 是我们的 docker 宿主机 (就是安装了 docker 的操作系统)
- docker daemon 是 docker 的后台守护进程，侦听并处理 docker 客户端命令，管理 docker 对象，例如镜像，容器，网络和卷。
- registry 是 docker 拉取镜像的远程仓库,提供大量的镜像供下载，下载完成之后保存在 images(本地镜像仓库)中.
- images 是 docker 本地的镜像仓库，可以通过 docker images 查看镜像文件。

1.1.5、Docker 核心对象



1.1.6、镜像、容器、仓库

镜像（image）：

docker 镜像（Image）就是一个只读的模板。镜像可以用来创建 docker 容器，一个镜像可以创建很多容器。 就好似 Java 中的类和对象，类就是镜像，容器就是对象。

容器（container）：

docker 利用容器（container）独立运行的一个或一组应用。容器是用镜像创建的运行实例。它可以被启动、开始、停止、删除。每个容器都是相互隔离的，保证安全的平台。可以把容器看做是一个简易版的 linux 环境（包括 root 用户权限、进程空间、用户空间和网络空间等）和运行在其中的应用程序。容器的定义和镜像几乎一模一样，也是一堆层的统一视角，唯一区别在于容器的最上面那一层是可读可写的。

仓库（repository）：

仓库 (repository) 是集中存放镜像文件的场所。仓库分为公开仓库 (public) 和私有仓库（private）两种形式。

最大的公开仓库是 docker hub(https://hub.docker.com/)，存放了数量庞大的镜像供用户下载。国内的公开仓库包括阿里云 、网易云等。

需要正确的理解仓储/镜像/容器这几个概念 ：

- docker 本身是一个容器运行载体或称之为管理引擎。我们把应用程序和配置依赖打包好形成一个可交付的运行环境，这个打包好的运行环境就似乎 image 镜像文件。只有通过这个镜像文件才能生成 docker 容器。image 文件可以看作是容器的模板。docker 根据 image 文件生成容器的实例。同一个 image 文件，可以生成多个同时运行的容器实例。
- image 文件生成的容器实例，本身也是一个文件，称为镜像文件。
- 一个容器运行一种服务，当我们需要的时候，就可以通过 docker 客户端创建一个对应的运行实例，也就是我们的容器。

- 至于仓库，就是放了一堆镜像的地方，我们可以把镜像发布到仓库中，需要的时候从仓库中拉下来就可以了。

1.1.7、Docker 运行机制

docker pull 执行过程：

1. 客户端将指令发送给 docker daemon
2. docker daemon 先检查本地 images 中有没有相关的镜像
3. 如果本地没有相关的镜像，则向镜像服务器请求，将远程镜像下载到本地

docker run 执行过程：

1. 检查本地是否存在指定的镜像，不存在就从公有仓库下载
2. 利用镜像创建并启动一个容器
3. 分配一个文件系(简版 linux 系统)，并在只读的镜像层外面挂载一层可读写层
4. 从宿主机配置的网桥接口中桥接一个虚拟接口到容器中去
5. 从地址池配置一个 ip 地址给容器
6. 执行用户指定的应用程序

1.2、docker 安装

1、官网安装参考手册：https://docs.docker.com/engine/install/ubuntu/

2、目前默认出厂镜像已经安装好，没有安装的可以使用下面命令一键安装：

```
curl -fsSL https://get.docker.com | bash -s docker --mirror Aliyun
```

3、查看 docker 版本

```
sudo docker version
```

```
jetson@ubuntu:~$ docker version
Client:
 Version:           20.10.21
 API version:       1.41
 Go version:        go1.18.1
 Git commit:        20.10.21-0ubuntu1~20.04.1
 Built:             Thu Jan 26 21:15:21 2023
 OS/Arch:           linux/arm64
 Context:           default
 Experimental:      true

Server:
 Engine:
  Version:          20.10.21
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.18.1
  Git commit:       20.10.21-0ubuntu1~20.04.1
  Built:            Thu Nov 17 20:19:30 2022
  OS/Arch:          linux/arm64
  Experimental:     false
 containerd:
  Version:          1.6.12-0ubuntu1~20.04.1
  GitCommit:
 runc:
  Version:          1.1.4-0ubuntu1~20.04.1
  GitCommit:
 docker-init:
  Version:          0.19.0
  GitCommit:
```

4、测试命令

```
sudo docker run hello-world
```

输出如下内容则表示 Docker 安装成功

```
jetson@ubuntu:~$ sudo docker run hello-world
[sudo] password for jetson:
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
7050e35b49f5: Pull complete
Digest: sha256:4e83453afed1b4fa1a3500525091dbfca6ce1e66903fd4c01ff015dbcb1ba33e
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (arm64v8)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

1、docker 镜像容器常用命令

2、docker 镜像容器常用命令

2、docker 镜像容器常用命令 2.1、不使用 sudo 命令 2.2、帮助命令 2.3、镜像命令 2.4、容器命令 2.5、常用其他命令 2.6、命令小结

2.1、不使用 sudo 命令

通常情况下，操作 docker 命令，需要加上前缀 sudo，例如：

```
sudo docker version
```

但是添加 docker 用户组之后，就可以不用加 sudo 前缀了。docker 用户组的添加方法（在运行 docker 的宿主机内运行命令）：

```
sudo groupadd docker          # 添加 docker 用户组
sudo gpasswd -a $USER docker  # 将当前用户添加至 docker 用户组，其中$USER 可以自动解析到当前登陆的用户
newgrp docker                 # 更新 docker 用户组
```

添加上述命令之后，使用【docker images】命令测试，如果没报错，说明已经可以不使用 sudo 命令了。如果报如下错误：

```
pi@ubuntu:~$ docker images
WARNING: Error loading config file: /home/pi/.docker/config.json: open /home/pi/.docker/config.json: permission denied
```

则在宿主机中执行如下命令解决：

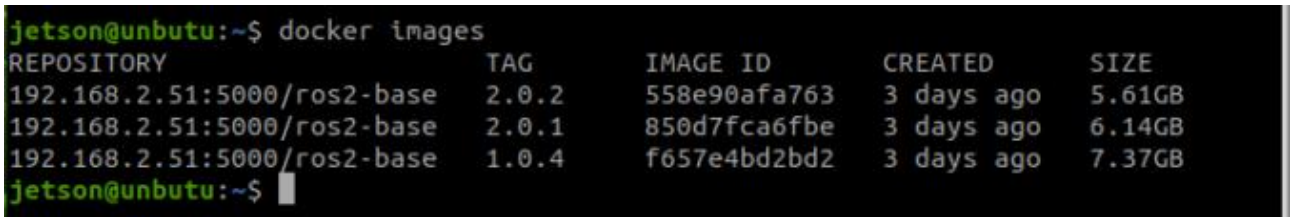
```
sudo chown "$USER":"$USER" /home/"$USER"/.docker -R
sudo chmod g+rwx "/home/$USER/.docker" -R
```

2.2、帮助命令

```
docker info     # 显示 Docker 系统信息，包括镜像和容器数。。
docker --help   # 帮助
```

2.3、镜像命令

1、docker images 列出本地主机上的镜像




```
# 解释
REPOSITORY 镜像的仓库源
TAG 镜像的标签
IMAGE ID 镜像的 ID
CREATED 镜像创建时间
SIZE 镜像大小
```

同一个仓库源可以有多个 TAG，代表这个仓库源的不同版本，我们使用 REPOSITORY：TAG 定义不同的镜像，如果你不定义镜像的标签版本，docker 将默认使用 lastest 镜像！

可选项

-a： 列出本地所有镜像

-q： 只显示镜像 id

--digests： 显示镜像的摘要信息

2、docker pull 从选定的仓库下载镜像

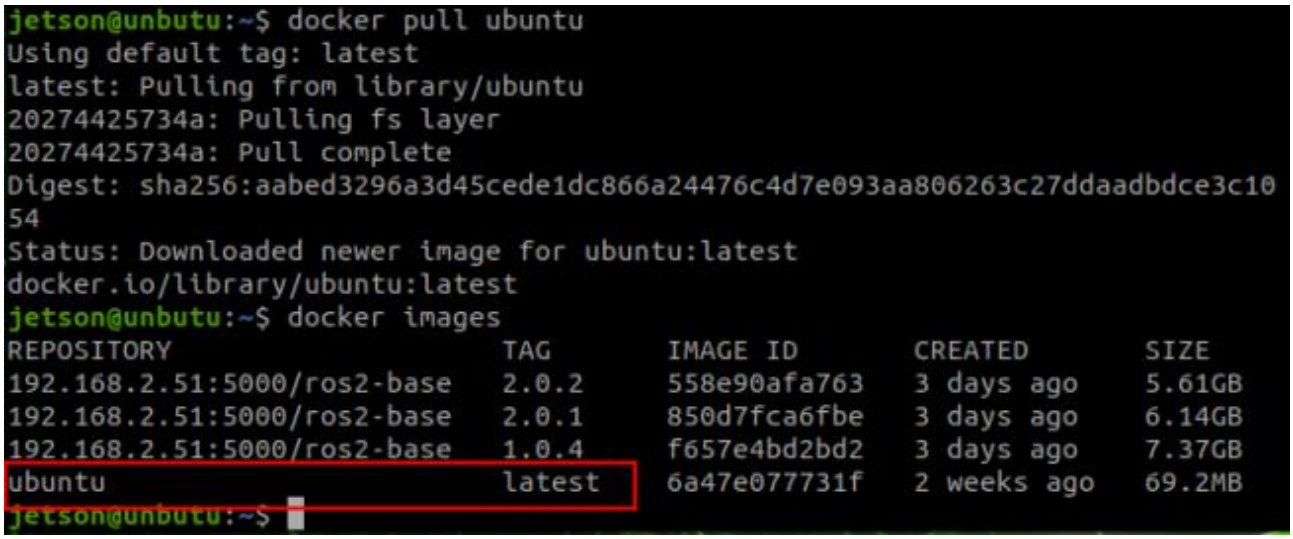
```
jetson@unbutu:~$ docker pull --help

Usage:  docker pull [OPTIONS] NAME[:TAG|@DIGEST]

Pull an image or a repository from a registry

Options:
  -a, --all-tags                Download all tagged images in the repository
      --disable-content-trust   Skip image verification (default true)
      --platform string         Set platform if server is multi-platform
                                capable
  -q, --quiet                   Suppress verbose output
jetson@unbutu:~$
```

使用方法

docker pull 镜像仓库地址/镜像名：版本

示例（无法下载，仅做参考）： docker pull yahboomtechnology/ros-foxy:1.0.0

```
# 例如下载 ubuntu 镜像
jetson@ubuntu:~$ docker pull ubuntu          # 其中 ubuntu 是镜像的名字
Using default tag: latest                    # tag 是版本号，不写 tag，默认是 latest
latest: Pulling from library/ubuntu
cd741b12a7ea: Pull complete                  # 分层下载
Digest: sha256:67211c14fa74f070d27cc59d69a7fa9aeff8e28ea118ef3babc295a0428a6d21
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest              # 真实位置
```



可以看到此时比之前多了个 ubuntu 的镜像

3、docker search 搜索对应 DockerHub 仓库中的镜像

```
# 搜索 ros2 镜像测试
jetson@ubuntu:~$ docker search ros2
NAME                                DESCRIPTION                                     STARS               OFFICIAL
            AUTOMATED
osrf/ros2                           **Experimental** Docker Images for ROS2
deve…     60                    [OK]
tiryoh/ros2-desktop-vnc             A Docker image to provide HTML5 VNC
interfac…     11
althack/ros2                        An assortment of development containers
for …     7
tiryoh/ros2                                         unofficial ROS2
image
uobflightlabstarling/starling-mavros2               ROS2 version of
MAVROS
# docker search 某个镜像的名称 对应 DockerHub 仓库中的镜像
# 可选项
--filter=stars=50 ： 列出收藏数不小于指定值的镜像。
```

4、docker rmi 删除镜像

```
# 删除镜像
docker rmi -f [镜像 id]                    # 删除单个镜像
docker rmi -f [镜像名:tag] [镜像名:tag]    # 删除多个镜像
docker rmi -f $(docker images -qa)         # 删除全部镜像
```

2.4、容器命令

有镜像才能创建容器，我们这里使用 ubuntu 的镜像来测试，下载镜像：

```
# ubuntu 是镜像的名称
docker pull ubuntu
```

1、docker run 运行镜像启动容器

```
# 命令参数说明
docker run [OPTIONS] IMAGE [COMMAND][ARG...]
# 常用参数
--name="Name" # 给容器指定一个名字
-d # 后台方式运行容器，并返回容器的 id!
```

-i # 以交互模式运行容器，通过和 -t 一起使用

-t # 给容器重新分配一个终端，通常和 -i 一起使用

-P # 随机端口映射（大写）

-p # 指定端口映射（小结），一般可以有四种写法

ip:hostPort:containerPort

ip::containerPort

hostPort:containerPort (常用)

#测试，查询是否下载镜像

jetson@ubuntu:~$ docker images

REPOSITORY TAG IMAGE

ID CREATED SIZE

ubuntu latest bab8ce5c00ca 6 weeks

ago 69.2MB

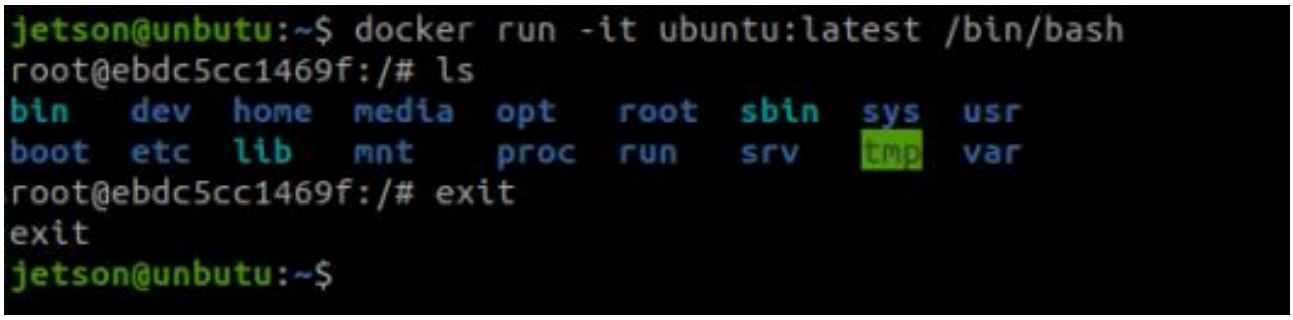
#使用 ubuntu 进行用交互模式启动容器，在容器内执行/bin/bash 命令！

jetson@ubuntu:~$ docker run -it ubuntu:latest /bin/bash

#输入 exit 退出容器

#可以看到 jetson@ubuntu：是主机系统界面，root@ebdc5cc1469f：则

是在 docker 容器内界面



2、docker ps 列出所有运行的容器

```
# 命令
docker ps [OPTIONS]
# 常用参数说明
-a # 列出当前所有正在运行的容器 + 历史运行过的容器
-l # 显示最近创建的容器
-n=? # 显示最近 n 个创建的容器
-q # 静默模式，只显示容器编号。
```

其中 container id 是容器的 ID，image 是该容器使用的镜像名称，created 是容器创建时间，status 是容器当前状态

3、退出容器

```
exit            # 容器停止退出
ctrl+P+Q        # 容器不停止退出
```

4、多终端进入正在运行的容器

```
# 命令 1
docker exec -it [容器 id] [bashShell]

# 测试
jetson@ubuntu:~$ docker ps -a
CONTAINER
ID    IMAGE              COMMAND         CREATED         STATUS
PORTS       NAMES
c54bf9efae47    ubuntu:latest    "/bin/bash"    2 hours ago    Up 4
seconds                          funny_hugle
3b9c01839579    hello-world      "/hello"       3 hours ago    Exited (0)
3 hours ago                 jovial_brown

jetson@ubuntu:~$ docker exec -it c54b /bin/bash       #容器的 id 可以
简写，只要是该容器唯一标识就行
```

```
root@c54bf9efae47:/#
```

```
# 命令 2
docker attach [容器 id]
```

```
# 测试
jetson@ubuntu:~$ docker ps -a
CONTAINER
ID    IMAGE              COMMAND         CREATED         STATUS
PORTS       NAMES
c54bf9efae47    ubuntu:latest    "/bin/bash"    2 hours ago    Up 35
seconds                         funny_hugle
3b9c01839579    hello-world      "/hello"       3 hours ago    Exited (0)
3 hours ago                jovial_brown

jetson@ubuntu:~$ docker attach c54b                    # 容器的 id 可以
简写，只要是该容器唯一标识就行
root@c54bf9efae47:/#
```

区别：

exec 是在容器中打开新的终端，并且可以启动新的进程，一般使用该方法进入容器

attach 直接进入容器启动命令的终端，不会启动新的进程

5、启动停止容器

```
docker start     [ 容器 id or 容器名 ]              # 启动容器
docker restart [ 容器 id or 容器名 ]              # 重启容器
docker stop      [ 容器 id or 容器名 ]              # 停止容器
docker kill      [ 容器 id or 容器名 ]              # 强制停止容器
```

6、删除容器

```
docker rm [容器 id]                                 # 删除指定容器
docker rm -f $(docker ps -a -q)                  # 删除所有容器
docker ps -a -q|xargs docker rm                  # 删除所有容器
```

2.5、常用其他命令

1、查看容器中运行的进程信息，支持 ps 命令参数。

```
# 命令
docker top [容器 id]
```

```
# 测试
jetson@ubuntu:~$ docker ps -a
CONTAINER
ID    IMAGE              COMMAND         CREATED         STATUS
PORTS       NAMES
c54bf9efae47    ubuntu:latest    "/bin/bash"    2 hours ago    Up 2
minutes                          funny_hugle
3b9c01839579    hello-world      "/hello"       3 hours ago    Exited (0)
3 hours ago                jovial_brown
jetson@ubuntu:~$ docker top c54b                # 容器的 id 可以简写，
只要是该容器唯一标识就行
UID         PID         PPID        C         STIME        TTY         TIME
 CMD
root        9667        9647        0         14:20        pts/0       00:00:00
/bin/bash
```

2、查看容器/镜像的元数据

```
# 命令
docker inspect [容器 id]
```

```
# 测试查看容器元数据
jetson@ubuntu:~$ docker ps -a
CONTAINER
ID    IMAGE              COMMAND         CREATED         STATUS
PORTS       NAMES
c54bf9efae47    ubuntu:latest    "/bin/bash"    2 hours ago    Up 4
minutes                          funny_hugle
3b9c01839579    hello-world      "/hello"       3 hours ago    Exited (0)
3 hours ago                 jovial_brown
jetson@ubuntu:~$ docker inspect c54bf9efae47
[
    {
        # 完整的 id，这里上面的容器 id，就是截取的这个 id 前几位
        "Id":
"c54bf9efae471071391202a8718b346d9af76cb1ff17741e206280603d
6f0056",
        "Created": "2023-04-24T04:19:46.232822024Z",
        "Path": "/bin/bash",
        "Args": [],
        "State": {
            "Status": "running",
```

```
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 9667,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2023-04-24T06:20:58.508213216Z",
            "FinishedAt": "2023-04-24T06:19:45.096483592Z"
        },
    。。。。
```


测试查看镜像元数据

```
jetson@ubuntu:~$ docker images
REPOSITORY                    TAG         IMAGE
ID          CREATED             SIZE
ubuntu                        latest      bab8ce5c00ca     6 weeks
ago         69.2MB
hello-world                   latest      46331d942d63     13 months
ago     9.14kB
```

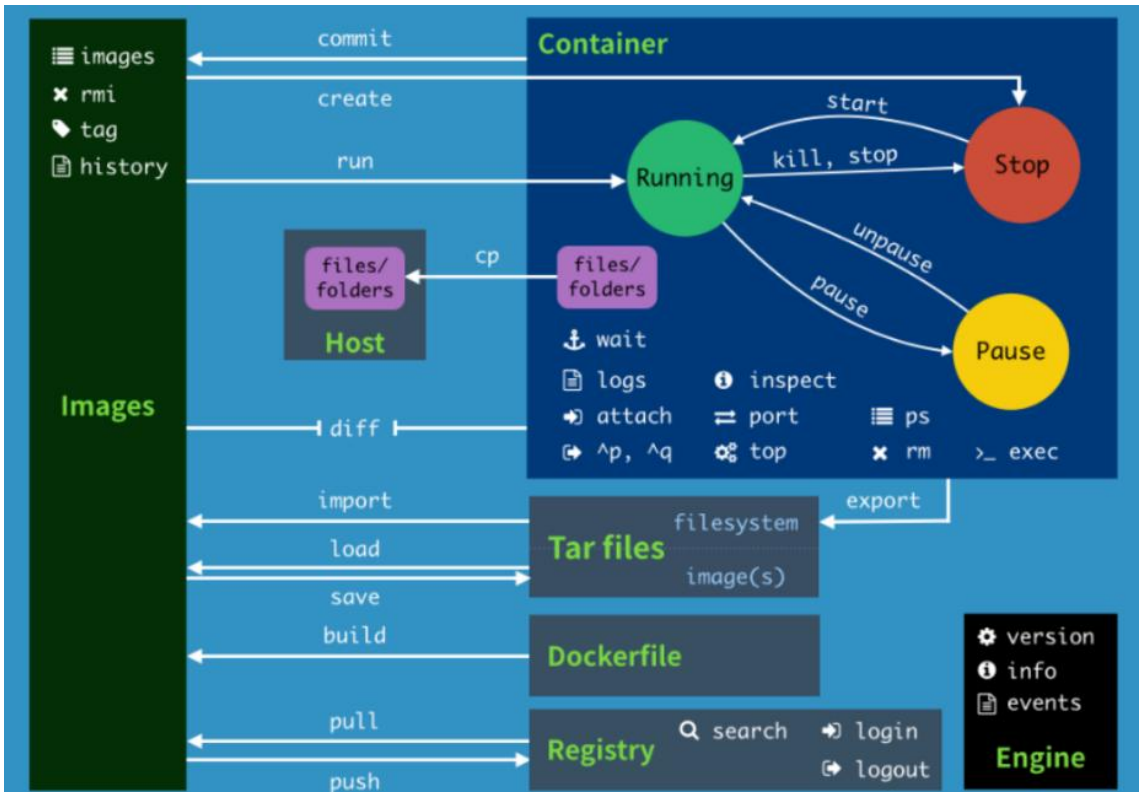
```
jetson@ubuntu:~$ docker inspect bab8ce5c00ca
[
    {
        "Id":
"sha256:bab8ce5c00ca3ef91e0d3eb4c6e6d6ec7cffa9574c447fd8d54a
8d96e7c1c80e",
        "RepoTags": [
            "ubuntu:latest"
        ],
        "RepoDigests": [
            "ubuntu@sha256:67211c14fa74f070d27cc59d69a7fa9aeff8
e28ea118ef3babc295a0428a6d21"
        ],
        "Parent": "",
        "Comment": "",
        "Created": "2023-03-08T04:32:41.063980445Z",
        "Container":
"094fd0c521be8c84d81524e4a5e814e88a2839899c56f654484d32d1
71c7195b",
        "ContainerConfig": {
            "Hostname": "094fd0c521be",
            ..............
```

```
            "Labels": {
                "org.opencontainers.image.ref.name": "ubuntu",
                "org.opencontainers.image.version": "22.04"
            }
        },
        "DockerVersion": "20.10.12",
        "Author": "",
        "Config": {
            "Hostname": "",
            .........
            "Labels": {
                "org.opencontainers.image.ref.name": "ubuntu",
                "org.opencontainers.image.version": "22.04"
            }
        },
        "Architecture": "arm64",
        "Variant": "v8",
        "Os": "linux",
        "Size": 69212233,
        "VirtualSize": 69212233,
        "GraphDriver": {
            "Data": {
```

```
                "MergedDir":
"/var/lib/docker/overlay2/8418b919a02d38a64ab86060969b37b4359
77e9bbdeb6b0840d4eb698280e796/merged",
                "UpperDir":
"/var/lib/docker/overlay2/8418b919a02d38a64ab86060969b37b4359
77e9bbdeb6b0840d4eb698280e796/diff",
                "WorkDir":
"/var/lib/docker/overlay2/8418b919a02d38a64ab86060969b37b4359
77e9bbdeb6b0840d4eb698280e796/work"
            },
            "Name": "overlay2"
        },
        "RootFS": {
            "Type": "layers",
            "Layers": [
                "sha256:874b048c963ab55b06939c39d59303fb975d32
3822a4ea48a02ac8dc635ea371"
            ]
        },
        "Metadata": {
            "LastTagTime": "0001-01-01T00:00:00Z"
        }
```

```
    }
]
```

2.6、命令小结



2、docker 镜像深入理解和发布镜像

3、docker 镜像深入理解和发布镜像

3、docker 镜像深入理解和发布镜像 3.1、镜像的理解 3.2、UnionFS（联合文件系统）3.3、镜像分层 3.3.1、分层理解 3.3.2、docker 镜像要采用分层

3.1、镜像的理解

1、镜像是一种轻量级、可执行的独立软件包，它包含运行某个软件所需要的所有内容。我们将应用程序、配置打包成一个成型的、可交付、可部署的运行环境，包括代码、运行时所需要的库、环境变量和配置文件等，这个大包好的运行环境就是 image 镜像文件。

2、只有通过镜像文件才能生成 docker 容器实例。

3.2、UnionFS（联合文件系统）

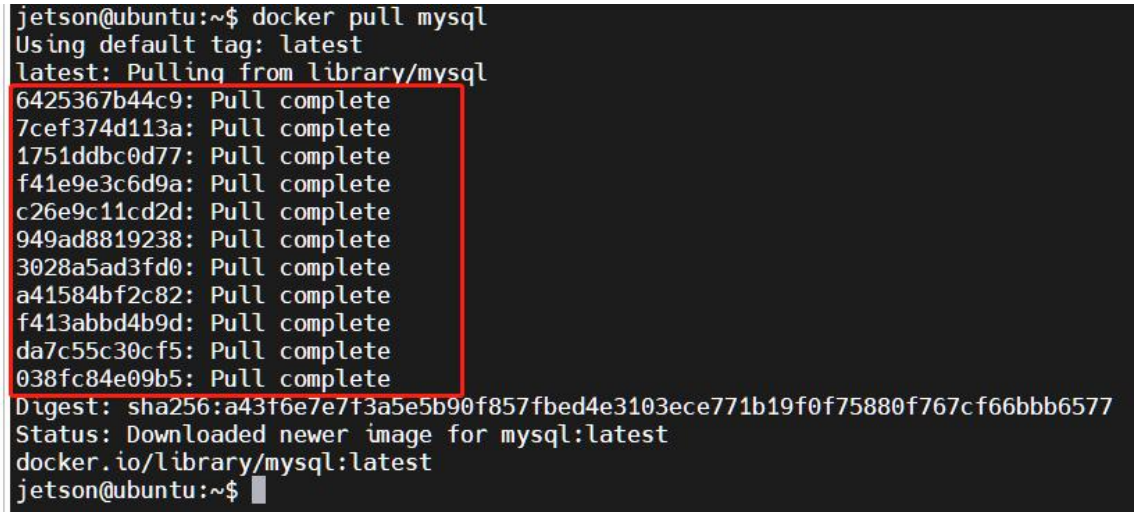
1、Union 文件系统（UnionFS）是一种分层的、轻量级的、高性能的文件系统，它是 docker 镜像的基础，并且支持对文件系统的修改作为一次提交来一层层的叠加，同时可以将不同目录挂在到同一个虚拟文件系统下。

2、镜像可以通过分层来进行继承，基于基础镜像，可以制作各种具体的应用镜像。

Union 文件系统的特性：一次性同时加载多个文件系统，但是从外面来看，只能看到一个文件系统；联合加载会把各层文件系统叠加起来，这样最终的文件系统会包含所有分层的文件和目录。

3.3、镜像分层

下载一个镜像时，注意观察下载的日志输出，可以看到是一层一层的在下载：



查看镜像分层的方式可以通过命令：docker image inspect 镜像名称

jetson@ubuntu:~$ docker image inspect mysql:latest

```
[
    {
        "Id":
"sha256:5371f8c3b63eec64a33b35530be5212d6148e0940111b57b68
9b5ba1ffe808c8",
        .....
        "RootFS": {
            "Type": "layers",
```

```
            "Layers": [
                "sha256:d6d4fc6aef875958d6186f85f03d88e6bb6484a
b2dd56b30a79163baceff2f6d",
                "sha256:05c3b0b311a02bc56ca23105a76d16bc9b8c1d
3e6eac808f4efb1a2e8350224b",
                "sha256:7b80f7f05642477ebc7d93de9539af27caab7c4
1a768db250fe3fe2b5506ca2c",
                "sha256:50e037faefab22cb1c75e60abb388b823e96a84
5650f3abd6d0a27e07a5a1d5e",
                "sha256:66040abb3f7201d2cc64531349a8225412db10
29447a9431d59d999c941d56f6",
                "sha256:857162425652837a362aa5f1c3d4974cc837027
28793de52ba48176d5367a89b",
                "sha256:7eebed3016f6b6ab68aa8e6be35f0689a3c18d3
31b7b542984a0050b859eaf26",
                "sha256:2fc4c142633d57d795edc0f3fd457f99a35fa611
eab8b8c5d75c66e6eb729bc2",
                "sha256:7fde2d12d484f0c14dabd9ca845da0bcdaf60bd
773a58ca2d73687473950e7fe",
                "sha256:9319848a00d38e15b754fa9dcd3b6e77ac8506
850d32d8af493283131b9745a3",
```

```
                "sha256:5ff94d41f068ea5b52244393771471edb6a9a10f
7a4ebafda9ef6629874a899b"
            ]
        },
        "Metadata": {
            "LastTagTime": "0001-01-01T00:00:00Z"
        }
    }
]
```


3.3.1、分层理解

所有的 docker 镜像都起始于一个基础镜像层，当进行修改或增加新的内容时，就会在当前镜像层之上，创建新的镜像层。

举一个简单的例子，假如基于 ubuntu 20.04 创建一个新的镜像，这就是新镜像的第一层；如果在该镜像中添加 python 包，就会在基础镜像层之上创建第二个镜像层；如果继续添加一个安全补丁，就会创建第三个镜像层，每一层一层的叠加。

docker 镜像都是只读的，当容器启动时，一个新的可写层被加载到镜像的顶部！这一层就是我们通常说的容器层，容器之下的都叫镜像层！

3.3.2、docker 镜像要采用分层的好处

资源共享，比如有多个镜像都从相同的 Base 镜像构建而来，那么宿主机只需在磁盘上保留一份 base 镜像，同时内存中也只需要加载一份 base 镜像，这样就可以为所有的容器服务了，而且镜像的每一层都可以被共享。

3.4、制作和发布镜像

3.4.1、制作镜像

方式 1、从容器中提交一个镜像：

```
# 命令
docker commit -m="提交的描述信息" -a="作者" 容器 id 要创建的目标镜像名:[标签名] 【也可省略 -m -a 参数】

# 测试
jetson@ubuntu:~$ docker ps -a
```

```
CONTAINER
ID     IMAGE               COMMAND         CREATED         STATUS
PORTS       NAMES
c54bf9efae47     ubuntu:latest     "/bin/bash"     3 hours ago     Up 24
minutes     funny_hugle

jetson@ubuntu:~$ docker commit c54bf9efae47 ubuntu:1.0
sha256:78ca7be949b6412f74ba12e8d16bd548aaa7c3fa25134326db3
a67784f848f8f
jetson@ubuntu:~$ docker images       #  此时生成了 ubuntu:1.0 的镜像
REPOSITORY                    TAG         IMAGE
ID          CREATED             SIZE
ubuntu                        1.0         78ca7be949b6     5 seconds
ago     69.2MB
ubuntu                        latest      bab8ce5c00ca     6 weeks
ago       69.2MB
hello-world                   latest      46331d942d63     13 months
ago     9.14kB
```

```
jetson@unbutu:~$ docker ps -a
CONTAINER ID   IMAGE          COMMAND       CREATED       STATUS                     PORTS
ebdc5cc1469f   ubuntu:latest  "/bin/bash"   2 hours ago   Exited (0) 2 hours ago
a28343337ad3   ubuntu:latest  "/bin/bash"   2 hours ago   Exited (0) 2 hours ago
jetson@unbutu:~$ docker commit ebdc5cc1469f ubuntu:1.0
sha256:43aeb49d903b2ab361486b223ad32d4b62ffc3bd4dffbe72bcb30555bf7ace9a
jetson@unbutu:~$ docker images
REPOSITORY                    TAG       IMAGE ID       CREATED         SIZE
ubuntu                        1.0       43aeb49d903b   7 seconds ago   69.2MB
192.168.2.51:5000/ros2-base   2.0.2     558e90afa763   3 days ago      5.61GB
192.168.2.51:5000/ros2-base   2.0.1     850d7fca6fbe   3 days ago      6.14GB
ubuntu                        latest    6a47e077731f   2 weeks ago     69.2MB
jetson@unbutu:~$
```

方式 2、dockerfile 制作镜像：

```
# 命令
docker build -f dockerfile 文件路径 -t 新镜像名字:TAG .  # docker build 命令最后有一个 . 表示当前目录
# 测试
docker build -f dockerfile-ros2 -t yahboomtechnology/ros-foxy:1.2 .
```

关于 dockerfile 的编写请参考：

https://docs.docker.com/develop/develop-images/dockerfile_best-practices/

3.4.2、发布镜像

docker 仓库（repository）是集中存放镜像文件的场所。最大的公开仓库是 docker hub(https://hub.docker.com/)，存放了数量庞大的镜像供用户下载。国内的公开仓库包括阿里云 、网易云等。

3.4.2.1、发布镜像到 docker hub 步骤：

1、地址：https://hub.docker.com/ ，先注册账号 2、保证账号可以正常登录

3、使用 tag 命令修改镜像名称

发布镜像到 docker hub 的规范是：

```
docker push 注册用户名/镜像名
```

比如我这里的注册用户名是：pengan88，那就要先修改镜像名称

```
# 命令：
docker tag [镜像 ID] [修改后的镜像名称]

# 测试
jetson@ubuntu:~$ docker images
```

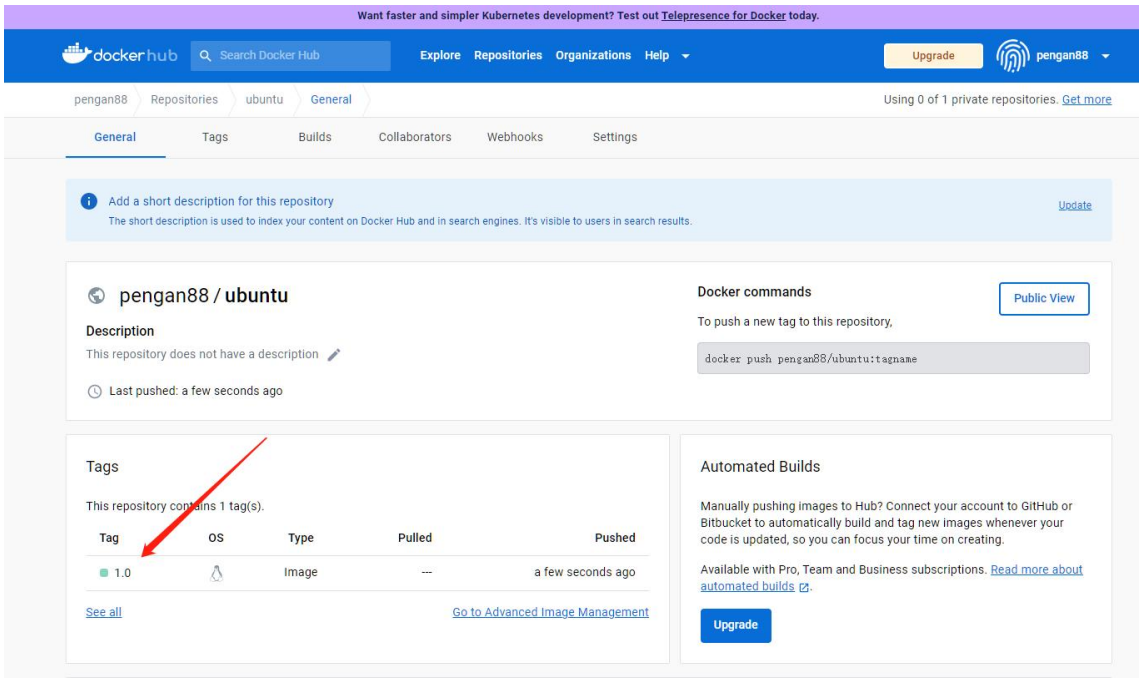
```
REPOSITORY                    TAG         IMAGE
ID          CREATED             SIZE
ubuntu                        1.0         78ca7be949b6     5 seconds
ago     69.2MB
ubuntu                        latest      bab8ce5c00ca     6 weeks
ago       69.2MB
hello-world                   latest      46331d942d63     13 months
ago     9.14kB
jetson@ubuntu:~$ docker tag 78ca7be949b6 pengan88/ubuntu:1.0
jetson@ubuntu:~$ docker images
REPOSITORY                    TAG         IMAGE
ID          CREATED             SIZE
pengan88/ubuntu               1.0         78ca7be949b6     23 minutes
ago     69.2MB
ubuntu                        1.0         78ca7be949b6     23 minutes
ago     69.2MB
ubuntu                        latest      bab8ce5c00ca     6 weeks
ago       69.2MB
hello-world                   latest      46331d942d63     13 months
ago      9.14kB
```

4、登录 docker hub 发布镜像：

```
jetson@ubuntu:~$ docker login -u pengan88
Password:          # 这里输入 docker hub 注册的账号密码
WARNING! Your password will be stored unencrypted in
/home/jetson/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#cred
entials-store
Login Succeeded
```

```
jetson@ubuntu:~$ docker push pengan88/ubuntu:1.0
The push refers to repository [docker.io/pengan88/ubuntu]
ca774712d11b: Pushed
874b048c963a: Mounted from library/ubuntu
1.0: digest:
sha256:6767d7949e1c2c2adffbc5d3c232499435b95080a25884657fae
366ccb71394d size: 736
```

5、访问 docker hub 可查看到已经发布成功



3.4.2.2、发布镜像生成 tar 压缩包步骤：

以下方法都是在宿主机中操作，而非 docker 里面操作

1、方式一：docker export 和 docker import（从容器中导出压缩包）

通过 docker ps -a 查询到容器的 ID，使用以下命令将该容器生成压缩包

```
docker export -o xxx.tar [容器 ID]
#压缩包名字可以自己取，例如 docker export -o ros2-base:1.0.1.tar [容器 ID]
```

生成压缩包后使用 docker import 导入到镜像库

```
docker import xxx.tar [镜像名]:[tag]
#例如 docker import ros2-base:1.0.1.tar
yahboomtechnology/ros2-base:1.0.1
#镜像名和 tag 可以自己定，tag 相当于版本号
```

2、方式二：docker save 和 docker load（从镜像中导出压缩包）

通过 docker images 查询镜像的名字和标签号，使用以下命令将该镜像生成压缩包

```
docker save -o xxx.tar [镜像名]:[tag]
#压缩包名字可以自己定
#例如 docker save -o ros2-base1.0.1.tar
yahboomtechnology/ros2-base:1.0.1
```

生成压缩包后使用 docker load 导入到镜像库

```
docker load -i xxx.tar
#这里是自己的压缩包名字
```

3、两种方式的区别

docker export 是从容器(container)中生成，docker save 是镜像(image)中生成。docker export 比 docker save 保存的包要小，原因是 save 保存的是一整个分层的文件系统，export 导出的只是容器一层文件系统。

docker import 和 docker load 导入都会生成镜像。docker import 可以对镜像指定新名称及版本号，docker load 无法对镜像重命名。

3、docker 硬件交互和数据处理

4、docker 硬件交互和数据处理

4.1、硬件挂载（端口绑定）

1、将自己需要挂载的设备接到主板上，在宿主机中建立 udev 规则（/etc/udev/rules.d/）

以下是以 ros 控制板和雷达设备挂载为例，实际情况根据自己的设备挂载

2、然后在开启容器时，将设置了规则的 devices 通过 --device=/dev/myserial --device=/dev/rplidar 等参数挂载到 docker 容器中

```
docker run -it --device=/dev/myserial --device=/dev/rplidar
ubuntu:latest /bin/bash
```

3、docker 容器中就能发现该设备了

```
jetson@ubuntu:~$ docker images
REPOSITORY                    TAG         IMAGE
ID          CREATED                 SIZE
ubuntu                        1.0         78ca7be949b6     About an hour
ago     69.2MB
pengan88/ubuntu               1.0         78ca7be949b6     About an
hour ago     69.2MB
hello-world                   latest      46331d942d63     13 months
ago         9.14kB

jetson@ubuntu:~$ ll /dev | grep ttyUSB*
lrwxrwxrwx     1 root     root             7 Apr 23 18:07 myserial ->
ttyUSB0
lrwxrwxrwx     1 root     root             7 Apr 23 18:07 rplidar -> ttyUSB1
crwxrwxrwx     1 root     dialout 188,     0 Apr 23 18:07 ttyUSB0
crwxrwxrwx     1 root     dialout 188,     1 Apr 23 18:07 ttyUSB1
```

```
jetson@ubuntu:~$ docker run -it    --device=/dev/myserial
--device=/dev/rplidar ubuntu:latest    /bin/bash
```

```
root@03522257ba30:/# ls /dev    # docker 中已经有 myserial 和 rplidar
console    fd    full    mqueue    myserial    null    ptmx    pts    random
rplidar    shm    stderr    stdin    stdout    tty    urandom    zero
```

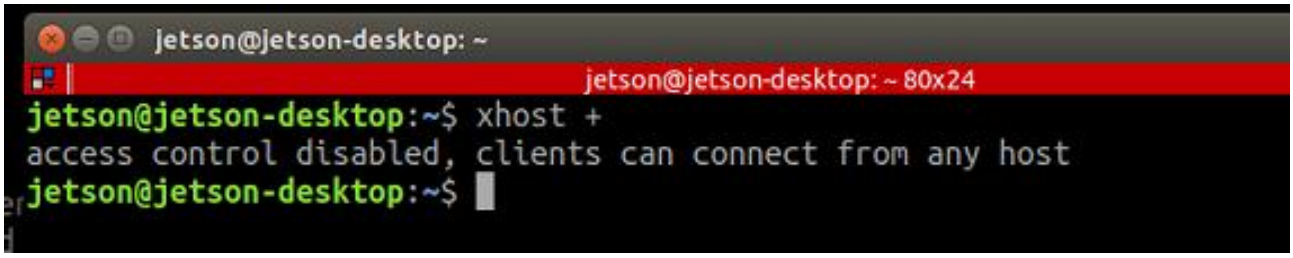
4.2、docker 中 GUI 的显示

1、在宿主机中安装（出厂镜像默认已经安装）：

```
sudo apt-get install tigervnc-standalone-server tigervnc-viewer
sudo apt-get install x11-xserver-utils
```

2、在宿主机中执行：xhost +

显示下图正常后，执行 3 步骤：



3、在宿主机中执行命令进入容器：

```
docker run -it \                                    # 交互式运行 docker 镜像
--env="DISPLAY" \                                   # 开启显示 GUI 界面
--env="QT_X11_NO_MITSHM=1" \                        # 采用 X11 的端口 1 进行显示
-v /tmp/.X11-unix:/tmp/.X11-unix \                  # 映射显示服务节点目录
yahboomtechnology/ros-foxy:1.0.0                    # 要启动的镜像名称
/bin/bash                                           # 在容器内执行/bin/bash 命令
```

4、测试

```
在容器中执行：rviz2
```

4.3、docker 容器和宿主机互传文件

4.3.1、使用 cp 指令拷贝文件

4.3.1.1、从容器内拷贝文件到主机上

```
# 命令
docker cp [容器 id:容器内路径] [目的主机路径]


# 测试
# 容器内执行，创建一个文件测试
jetson@ubuntu:~$ docker ps -a
CONTAINER
ID    IMAGE              COMMAND         CREATED         STATUS
              PORTS       NAMES
c54bf9efae47    ubuntu:latest    "/bin/bash"    2 hours ago    Up 9
minutes                            funny_hugle
3b9c01839579    hello-world      "/hello"       3 hours ago    Exited (0)
3 hours ago                 jovial_brown
jetson@ubuntu:~$ docker attach c5
root@c54bf9efae47:/# cd
root@c54bf9efae47:~# ls
root@c54bf9efae47:~# touch test.txt
root@c54bf9efae47:~# ls
```

test.txt

root@c54bf9efae47:~# pwd

/root

root@c54bf9efae47:/# （read escape sequence） #直接按 ctrl+P+Q

容器不停止退出

jetson@ubuntu:~$ docker cp c54bf9efae47:/root/test.txt ~/

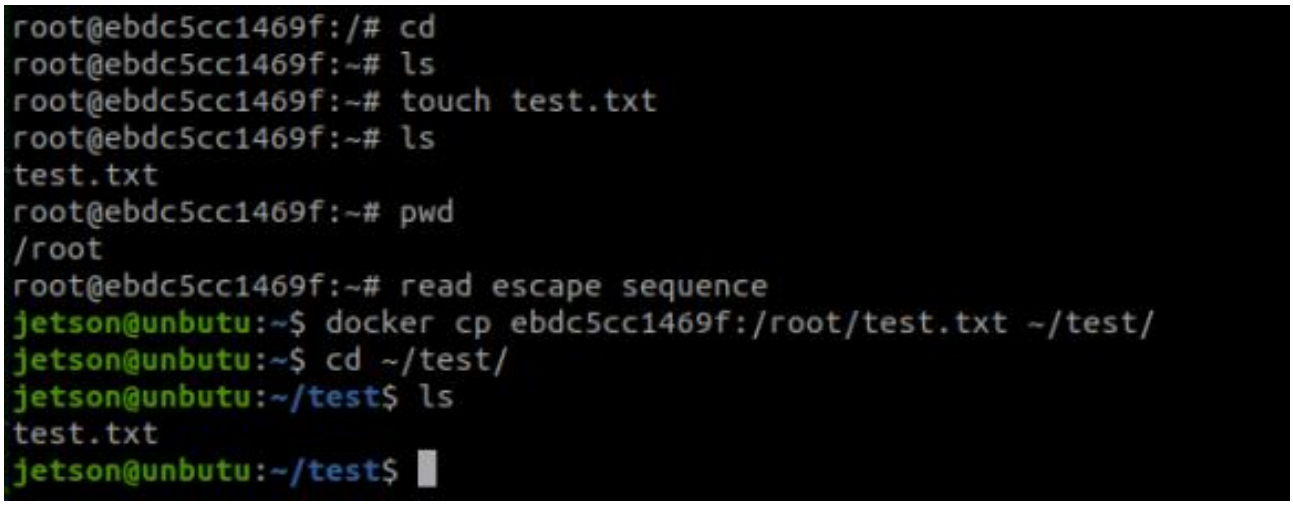
jetson@ubuntu:~$ ls # 可以看到 test.txt 文件已经拷贝进来了

Desktop Documents Downloads fishros Music openvino

Pictures Public rootOnNVMe run_docker.sh sensors snap

temp Templates test.txt Videos

示例图：



4.3.1.2、从宿主机拷贝文件到容器上

命令

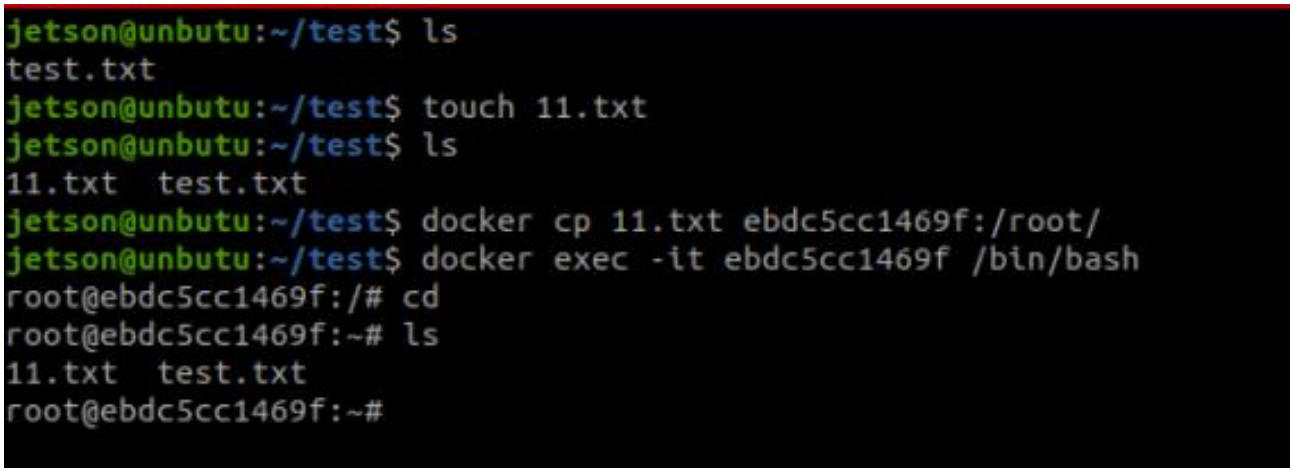
docker cp [宿主机文件路径] [容器 id:容器内路径]

#测试

```
jetson@ubuntu:~$ docker ps -a
CONTAINER
ID    IMAGE              COMMAND         CREATED         STATUS
              PORTS       NAMES
c54bf9efae47    ubuntu:latest    "/bin/bash"    2 hours ago    Up 5
minutes                          funny_hugle
3b9c01839579    hello-world      "/hello"       3 hours ago    Exited (0)
3 hours ago                jovial_brown
jetson@ubuntu:~$ ls
Desktop  Documents  Downloads  fishros  Music  openvino
Pictures  Public  rootOnNVMe  run_docker.sh  sensors  snap
temp  Templates  test.txt  Videos
jetson@ubuntu:~$ touch 11.txt
jetson@ubuntu:~$ ls
11.txt  Desktop  Documents  Downloads  fishros  Music
openvino  Pictures  Public  rootOnNVMe  run_docker.sh  sensors
snap  temp  Templates  test.txt  Videos
jetson@ubuntu:~$ docker cp 11.txt c54bf9efae47:/root/
jetson@ubuntu:~$ docker exec -it c54b /bin/bash
root@c54bf9efae47:/# ls
```

```
bin   boot   dev   etc   home   lib   media   mnt   opt   proc   root
run   sbin   srv   sys   tmp   usr   var
root@c54bf9efae47:/# cd /root/
root@c54bf9efae47:~# ls        # 11.txt 文件已经拷贝进来了
11.txt   test.txt
```

示例图：



4.3.2、使用数据卷

4.3.2.1、数据卷概述

将应用和运行的环境打包形成容器运行，运行可以伴随着容器，但是我们对于数据的要求，是希望能够持久化的！就好比，你安装一个 mysql，结果你把容器删了，就相当于删库跑路了，这肯定不行吧！所以我们希望容器之间有可能可以共享数据，docker 容器产生的数据，如果不通过 docker commit

生成新的镜像，使得数据作为镜像的一部分保存下来，那么当容器删除后，数据自然也就没有了！这样是行不通的！

为了能保存数据在 docker 中我们就可以使用卷！让数据挂载到我们本地！这样数据就不会因为容器删除而丢失了！

特点：

1、数据卷可在容器之间共享或重用数据

2、卷中的更改可以直接生效

3、数据卷中的更改不会包含在镜像的更新中

4、数据卷的生命周期一直持续到没有容器使用它为止

4.3.2.2、数据卷使用

```
# 命令
docker run -it -v [宿主机绝对路径目录:容器内目录] [镜像名]

# 测试
docker run -it -v /home/jetson/temp:/root/temp
yahboomtechnology/ros-foxy:3.4.0 /bin/bash
```

宿主机中的/home/jetson/temp 目录和容器内的/root/temp 目录就可以共享数据了

4、进入 docker 容器

5、进入 docker 容器

5.1、相关概念

1、什么是 docker 的宿主机

宿主机就是我们调用命令使用镜像创建容器的服务器。这里是指我们系统主板（jetson 或树莓派等），以下提到的宿主机都是指这个。

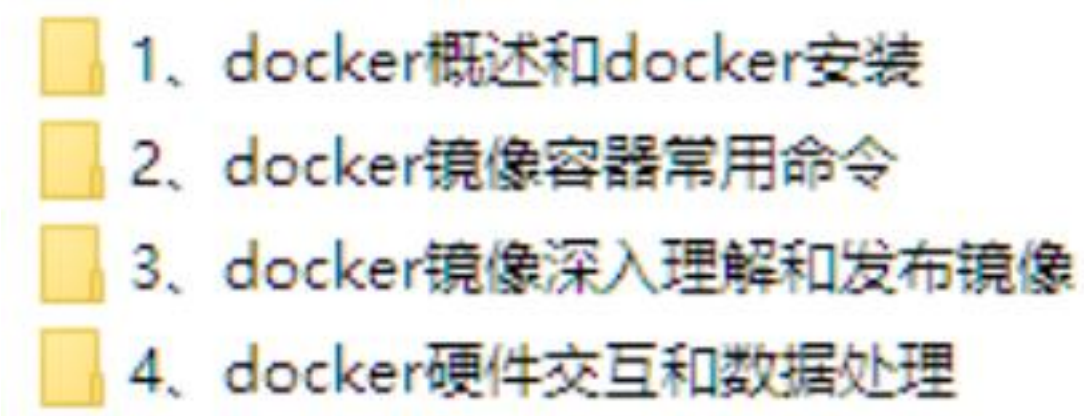
2、什么是 GUI

GUI 即图形用户界面，这里主要是指：opencv 显示的图像窗口、rviz 界面、rqt 界面等。

3、什么是 docker 容器

打开 docker 镜像运行后的就是容器

4、在操作本章节教程前，请确保自己已经掌握了下面这些章节的知识，否则可能会感到学起来比较吃力。出现这种情况，请反复查看下面这些预先知识的内容，熟练掌握后会觉得很轻松，Come on，you are the best!



5.2、如何查询使用的 docker 镜像版本

1、用户在烧录了出厂系统镜像开机启动后，查看镜像：

```
jetson@jetson-desktop:~$ docker images
REPOSITORY                          TAG         IMAGE
ID          CREATED                 SIZE
192.168.2.51:5000/ros2-base     2.0.1       fce9745648c0     3 days
ago             6.5GB
192.168.2.51:5000/ros2-base     1.0.3       cs41dfas5fx0     3 days
ago             7.2GB
192.168.2.51:5000/ros2-base     1.0.2       31e97028c1c0     3 days
ago             7.2GB
```

会看到有多个 docker 镜像版本，实际情况根据自己查询到的镜像显示

2、为什么不能只放一个 docker 镜像呢?

如果你看过【Docker ------ 3、docker 镜像深入理解和发布镜像】这章的教程，应该知道 docker 镜像是分层机制，也就是后面一个 tag 的镜像依赖前面一个 tag 的镜像。所以宿主机中可能会存在多个版本的 docker 镜像，这些镜像的 tag 会以递增的方式更新。

后续我们更新了新的课程也会用发布新的 docker 镜像的方式来更新功能

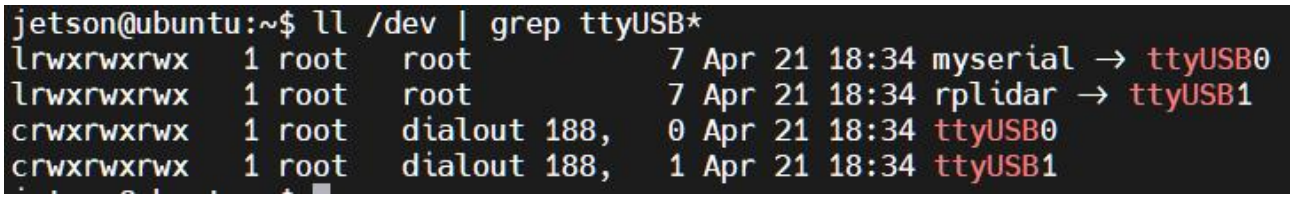
5.3、查看外设连接情况

以下用激光雷达和串口设备绑定为例

此步骤在宿主机上操作：

1、这里是查看除摄像头之外的外设

```
ll /dev | grep ttyUSB*
```



2、查看摄像头设备

```
ls /dev/video*
```

5.4、编辑启动脚本

此时根据上一步查到的设备编辑启动脚本

以下用激光雷达和串口设备为例

编辑运行 docker 的脚本，此步骤在宿主机上操作：

1、创建一个 docker 运行脚本【run_usb_docker.sh】一般会放在 home 目录下

```
chmod +x run_usb_docker.sh    #给脚本赋予可执行权限
```

【run_usb_docker.sh】脚本的内容如下：

不带注释的可以直接复制并根据下面的注释按需修改

注意：以下添加主机设备给容器时，如果宿主机没有连接该设备，需要去掉相应的添加操作，才能开启容器

```
#!/bin/bash
xhost +

docker run -it \
--net=host \
--env="DISPLAY" \
--env="QT_X11_NO_MITSHM=1" \
-v /tmp/.X11-unix:/tmp/.X11-unix \
-v /home/jetson/temp:/root/yahboomcar_ros2_ws/temp \
--device=/dev/video0 \
--device=/dev/myserial \
--device=/dev/rplidar \
```

```
-p 9090:9090 \
-p 8888:8888 \
yahboomtechnology/ros2-base:2.0.2 /bin/bash
```

带注释的脚本说明：

注意：以下添加主机设备给容器时，如果宿主机没有连接该设备，需要去掉相应的添加操作，才能开启容器

```
#!/bin/bash
xhost +                                          # xhost 是用来支持 docker 内显示 GUI 的

docker run -it \                                 # 交互式运行 docker 镜像
--net=host \                                     # 容器网络设置为 host 模式
--env="DISPLAY" \                                # 开启显示 GUI 界面
--env="QT_X11_NO_MITSHM=1" \                     # 采用 X11 的端口 1 进行显示
-v /tmp/.X11-unix:/tmp/.X11-unix \               # 映射显示服务节点目录
```

```
-v /home/jetson/temp:/root/yahboomcar_ros2_ws/temp \      # 作为宿主机和容器临时传输文件的目录，有需要传输文件，可以使用这个目录
--device=/dev/video0 \                                    # 添加主机设备给容器，没有连接摄像头，请去掉这行
--device=/dev/myserial \                                  # 这里是串口设备端口,没有连接串口设备，请去掉这行
--device=/dev/rplidar \                                   # 这里是雷达设备端口,没有连接雷达设备，请去掉这行
-p 9090:9090 \                                            # 开放的端口
-p 8888:8888 \
yahboomtechnology/ros2-base:2.0.2 /bin/bash          # 要启动的镜像名称，根据 5.2 步骤中查询到的修改；在容器内执行/bin/bash 命令

#注意：以上添加主机设备给容器时，如果宿主机没有连接该设备，需要去掉相应的添加操作，才能开启容器
```

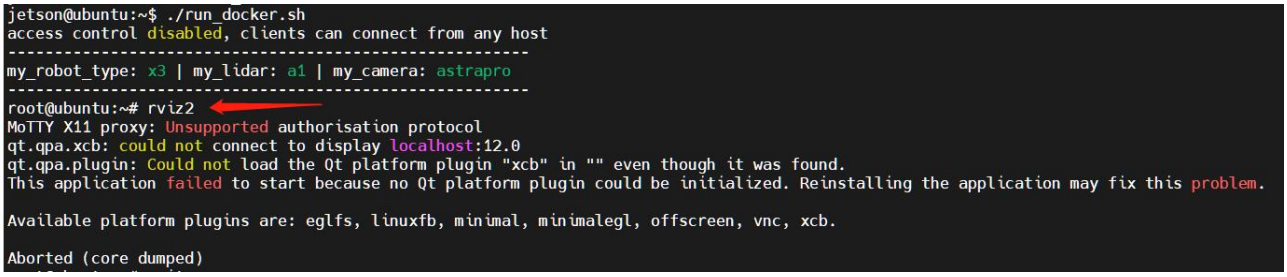
5.5、执行脚本

docker 启动脚本编辑完后，在 docker 的宿主机上（可在 VNC 上面或者在主板显示屏上）打开终端

注意：这里必须是在 VNC 上面或者在主板显示屏上执行，不可在通过 ssh 远程进入的终端（如通过 MobaXterm 进入的终端）中执行，否则可能容器

中无法显示 GUI 图像，如下在 MobaXterm 中进入终端执行

run_usb_docker.sh 进入容器后，无法显示 rviz

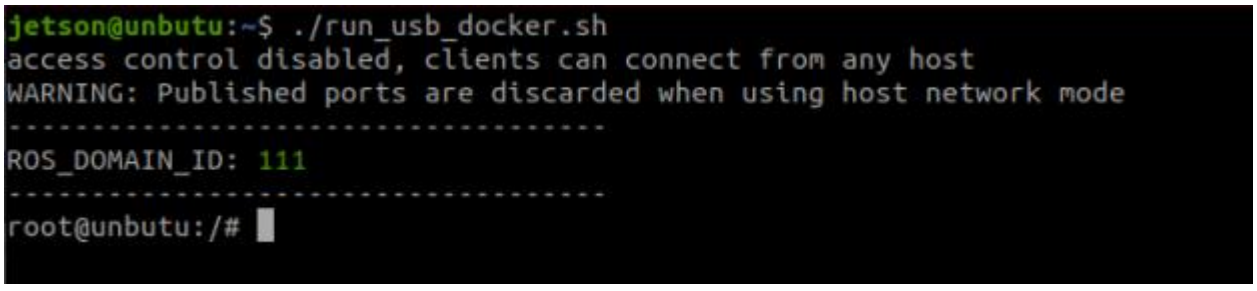


在 VNC 界面或者在屏幕上运行之前创建的启动脚本（注意：每次执行该脚本都是从镜像中创建一个新的容器）

```
./run_usb_docker.sh
```

即可正确进入容器，并能显示 GUI 画面，可以再次执行 rviz2 命令测试。

```
rviz2
```



如果执行 rviz2 命令后无法显示 GUI，显示如下错误：（一般在树莓派主控中有可能出现）

```
root@ubuntu:~# rviz2
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-root'
dbus[97]: The last reference on a connection was dropped without closing the con
nection. This is a bug in an application. See dbus_connection_unref() documentat
ion for details.
Most likely, the application was supposed to call dbus_connection_close(), since
 this is a private connection.
  D-Bus not built with -rdynamic so unable to print a backtrace
Aborted (core dumped)
```

需要再启动脚本中再加一个参数：

```
--security-opt apparmor:unconfined
```

即：

```
#!/bin/bash
xhost +

docker run -it \
--net=host \
--env="DISPLAY" \
--env="QT_X11_NO_MITSHM=1" \
-v /tmp/.X11-unix:/tmp/.X11-unix \
--security-opt apparmor:unconfined \                    # 添加了这句参数
-v /home/jetson/temp:/root/yahboomcar_ros2_ws/temp \
--device=/dev/video0 \
```

```
--device=/dev/myserial \
--device=/dev/rplidar \
-p 9090:9090 \
-p 8888:8888 \
yahboomtechnology/ros2-base:2.0.2 /bin/bash
```

然后再次运行脚本即可进入新的容器，并能显示 GUI 画面。

5.6、多终端进入同一个 docker 容器

1、在上面的步骤中已经开启了一个 docker 容器，可以在宿主机上打开另一个终端查看：

```
docker ps -a
```

2、现在在这个新打开的终端中进入该 docker 容器：

```
docker exec -it [容器 id] /bin/bash
```

图片示例：

成功进入容器，还可以通过同样的方法再打开无数个终端进入该容器。

3、注意：

（1）执行 2 步骤中的命令时要确保容器处于开启【UP】状态

（2）如果容器处于【Exited】关闭状态，请参见下面 5.7 步骤操作

5.7、如何开启已经处于【Exited】关闭状态的容器

这里分两种情况： **摄像头等相关外部设备有变动** 和 **外部设备没有变动的**（参考 5.3 查看外设连接情况）

1、摄像头等相关外部设备有变动的

如果该容器中有一些修改需要保留，可以参考以下命令生成一个新的镜像（详细发布镜像的步骤请看之前的教程）

```
```

从容器中提交一个镜像：

docker commit [容器 id] [要创建的目标镜像名:[标签名]]

例如：docker commit 66c40ede8c68 yahboomtechnology/ros-foxy:1.1

标签名根据自己的情况递增

然后再运行这个新的镜像进入容器：参见本章节【5.2 到 5.5】步骤执行

2、摄像头等相关外部设备没有变动的，那直接参见【5.7.1、再次进入【Exited】关闭状态的容器】步骤执行。

5.7.1、再次进入【Exited】关闭状态的容器

在 docker 的宿主机上【可在 VNC 上面或者在主板屏幕上】打开终端

注意：这里必须是在 VNC 上面或者在主板屏幕上执行，不可在通过 ssh 远程进入的终端（如通过 MobaXterm 进入的终端）中执行，否则可能容器中无法显示 GUI 图像，当然如果你不需要显示 GUI 图像，那就可以。

1、首先查看容器的状态

```
docker ps -a
```

2、开启 GUI 访问权限

```
xhost +
```

3、开启容器【这里容器的 ID 是可以简写的，只要能够唯一识别当前已经存在的容器即可】

```
docker start [容器的 ID]
```

4、再次进入该容器

```
docker exec -it [容器的 ID] /bin/bash
```

5、在容器内打开 rviz 查看是否已经能够开启 GUI 画面

```
rviz2
```

5、未来更新 docker 镜像的方法

6、未来更新 docker 镜像的方法

6、未来更新 docker 镜像的方法 6.1、方式一(重新刷主板的出厂镜像)6.2、方式二(从 docker hub 上下载最新 docker 镜像)6.3、方式三(通过压缩包更新 docker 镜像)

目前 ros2 的课程全部置于 docker 容器内，客户可以体验学习使用容器化的开发方法。

在更新 docker 镜像之前请查询自己的镜像版本号，再根据以下的方法看看自己的镜像是否为最新的镜像

```
docker images
```

未来会持续在 docker 中添加新的功能模块，这些新的功能模块会放到新的 docker 镜像中，用户需要体验这些新的功能，可以有以下三种方式来更新 docker 镜像：

6.1、方式一(重新刷主板的出厂镜像)

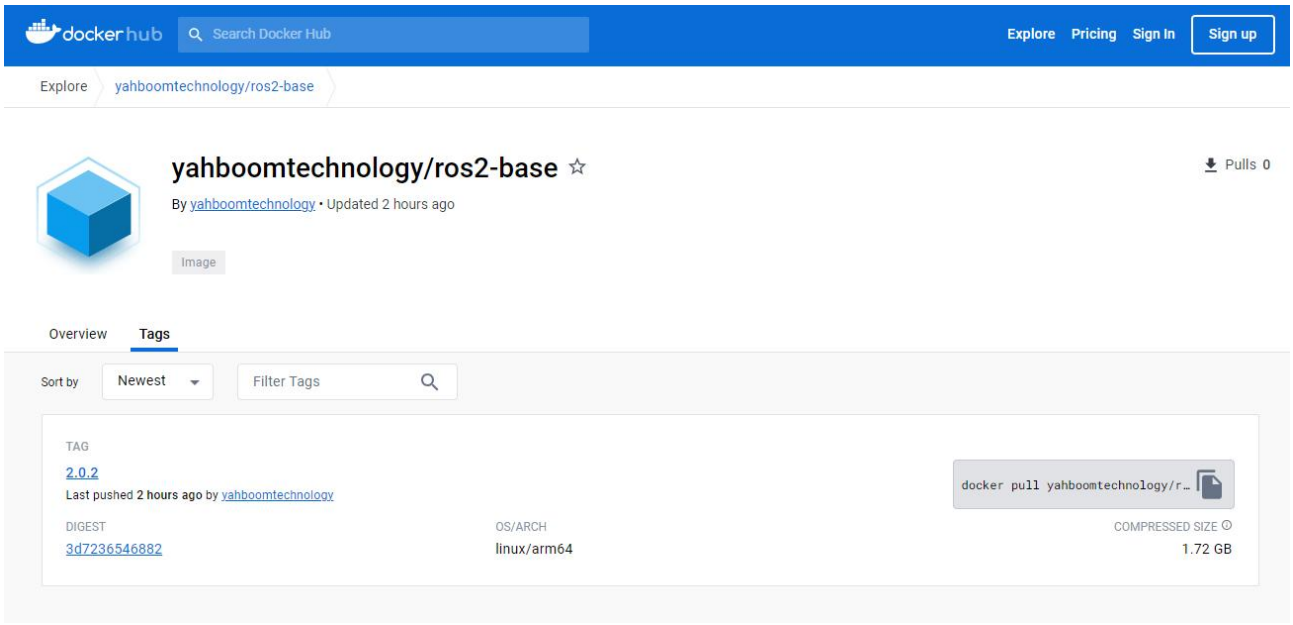
更新了新的 docker 镜像时，会发布带宿主机的 img 镜像，宿主机内已经下载好了新的 docker 镜像，客户直接使用这个 img 刷机即可。如果觉得麻烦可以用下面两种方式。

6.2、方式二(从 docker hub 上下载最新 docker 镜像)

更新了新的 docker 镜像时，用户可以在不刷机的情况下，手动更新镜像

docker hub 镜像网站地址（可能需要科学上网）：

https://hub.docker.com/r/yahboomtechnology/ros2-base/tags



宿主机中使用命令：

```
docker pull [最新的镜像版本号]
```

例如：

```
docker pull yahboomtechnology/ros2-base:2.0.2  # 这里最新的镜像版本号根据实际查看到的修改
```

这个新的镜像版本号 请查看本节目录:【最新 docker 镜像版本号和压缩包】，打开【最新 docker 镜像版本号.txt】文件，如果查看到的版本号比现在宿主机中的版本高，说明有更新，可以更新镜像。这种方式需要从外网下载 docker 镜像，耗时较长，可能会出现超时下载不下来，也有可能网络原因下载失败，如果发生这种情况，请使用另外两种方式。

pull 执行完成后，可以查看已下载的镜像

```
docker images
```

6.3、方式三(通过压缩包更新 docker 镜像)

更新了新的 docker 镜像时，用户可以在不刷机的情况下，手动更新镜像：

新的 docker 镜像会提供一个【xxx.tar】的文件，里面存储的就是新的 docker 镜像，该文件放在本节目录【最新 docker 镜像版本号和 tar 文件】中，如果该文件的的版本号比现在宿主机中的高，说明有更新，可以更新镜像。将该文件下载到宿主机内。

这里的 tar 文件是实时更新的，看到的时候有可能已经大于 2.0.2 版本了，根据实际看到的为准。

在宿主机的【xxx.tar】文件所在目录使用命令，该操作需要一些时间，但是基本不会失败

```
```

`docker load -i xxx.tar`

docker load 执行完成后，可以查看已下载的镜像

```
docker images
```

3. ROS2 基础教程

4. ROS2 简介

1、ROS2 简介

1、ROS2 概述

ROS2 是第二代的 Robot Operating System，ROS1 的升级版本，解决了 ROS1 存在的一些问题。ROS2 最早出现的版本 Arden 是在 2017 年，随着版本的迭代，不断地更新玉优化，现如今已经有了稳定的版本。与 ROS1 相通过，Linux 版本与 ROS2 版本的选择也是有关系的，两者对应的版本如下，

ROS2 版本	Ubuntu 版本
Foxy	Ubuntu20.04
Galactic	Ubuntu20.04
Humble	Ubuntu22.04

根据自己的 Linux 版本，下载对应的 ROS2 版本，本课程以 Foxy 版本为基础。

2、ROS2 特性

2.1、**ROS2 全面支持三种平台**

Ubuntu

Mac OS X

Windows 10

2.2、实现了分布式架构

取消 Master 中央节点，实现节点的分布式发现，发布/订阅，请求/响应通讯。

2.3、**支持实时**

2.4、使用新版本的编程语言

- C++11
- Python3.5+

2.5、**使用了新的编译系统 Ament（ROS 为 Catkin）**

2.6、**ROS1 可以通过 rosbridge 和 ROS 2 通信**

3、ROS2 与 ROS1 的区别

3.1、平台

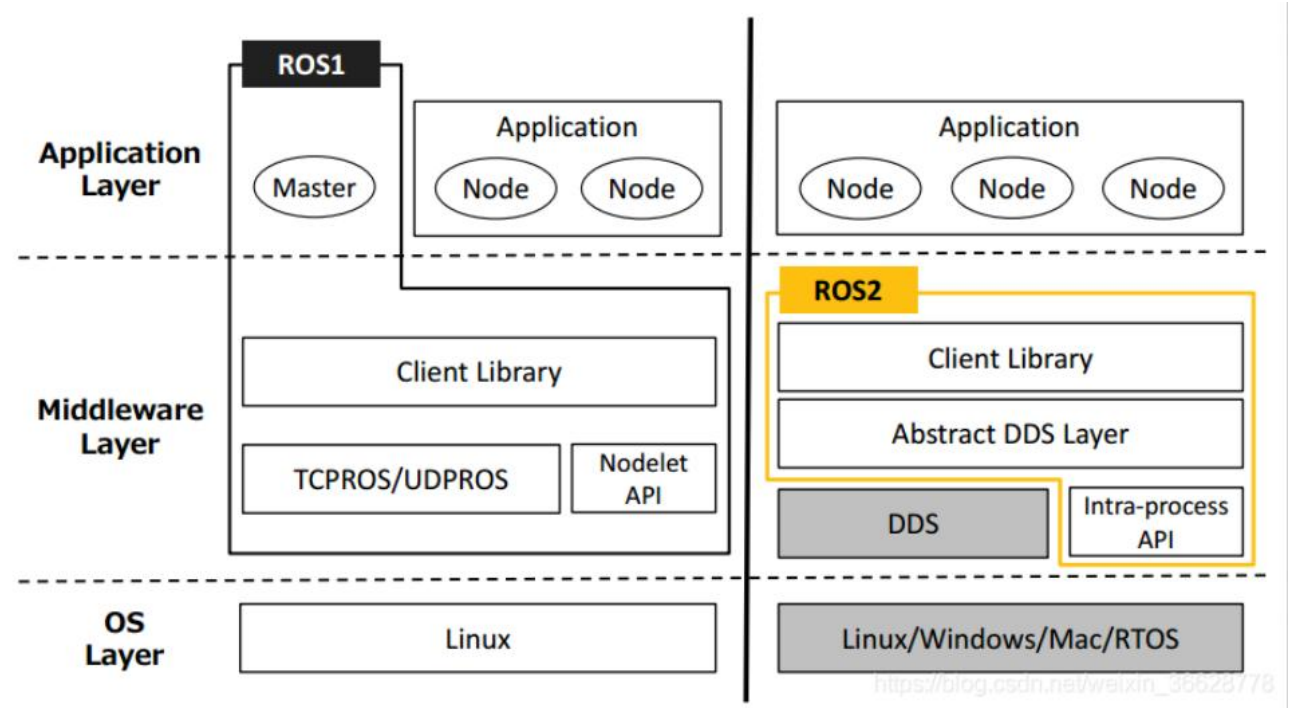
ROS1 目前来说仅支持在 Linux 系统中运行使用，常见的是在 Ubuntu 中搭建使用。而 ROS2 目前在 Ubuntu、Windows 甚至嵌入式开发板上都可以搭建使用，平台更加广泛。

3.2、语言

- C++
- ROS1 的核心是 C++03，而 ROS2 广泛使用 C++11。
-
-
- Python
- ROS1 的 Python 使用版本是 Python2，而 ROS2 使用的 Python 版本至少是 3.5 以上，Foxy 使用的 Python 版本是 3.8。
-

3.3、中间件

ROS1 启动前需要开启 roscore，这个 master 掌握所有的节点之间的通讯，而 ROS2 则没有，只有一个抽象的中间件接口，通过该接口进行传输数据。。目前，此接口的所有实现都基于 DDS 标准。这使得 ROS 2 能够提供各种优质的 Qos 服务策略，从而改善不同网络的通信。



3.4、编译命令

ROS1 的编译命令是 catkin_make，而 ROS2 的编译命令使用 colcon build 命令。

1、ROS2 常用命令与工具

2、ROS2 常用命令与工具

1、包管理工具 ros2 pkg

1.1、ros2 pkg create

功能：创建功能包，创建时候需要指定包名、编译方式、依赖项等。

命令格式：ros2 pkg create --build-type ament_python pkg_name rclpy std_msgs sensor_msgs

ros2 pkg create：创建包的指令

--build-type：新创建的功能包如果使用 C++或者 C，那这里就写 ament_cmake，如果使用 Python，就写 ament_python

pkg_name：创建功能包的名字

rclpy std_msgs sensor_msgs：这些都是一些编译依赖

1.2、ros2 pkg list

功能：查看系统中功能包列表

命令格式：ros2 pkg list

```
yahboom@yahboom-virtual-machine:~$ ros2 pkg list
action_msgs
action_tutorials_cpp
action_tutorials_interfaces
action_tutorials_py
actionlib_msgs
ament_cmake
ament_cmake_auto
ament_cmake_copyright
ament_cmake_core
ament_cmake_cppcheck
ament_cmake_cpplint
ament_cmake_export_definitions
ament_cmake_export_dependencies
ament_cmake_export_include_directories
ament_cmake_export_interfaces
ament_cmake_export_libraries
ament_cmake_export_link_flags
ament_cmake_export_targets
ament_cmake_flake8
ament_cmake_gmock
ament_cmake_gtest
ament_cmake_include_directories
ament_cmake_libraries
ament_cmake_lint_cmake
ament_cmake_pep257
ament_cmake_pytest
ament_cmake_python
ament_cmake_ros
ament_cmake_target_dependencies
```

1.3、ros2 pkg executeables

命令功能：查看包内可执行文件列表

命令格式： ros2 pkg executables pkg_name

```
yahboom@yahboom-virtual-machine:~$ ros2 pkg executables turtlesim
turtlesim draw_square
turtlesim mimic
turtlesim turtle_teleop_key
turtlesim turtlesim_node
```

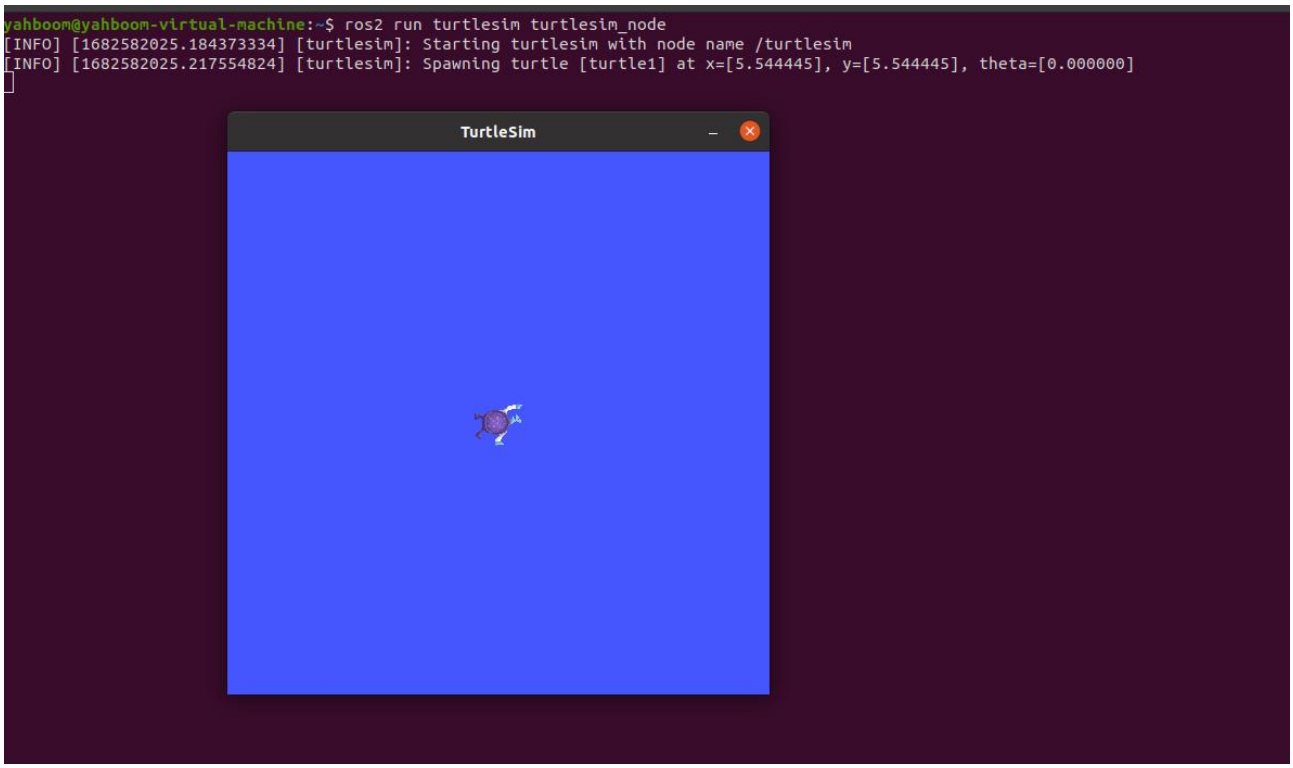
2、节点运行 ros2 run

命令功能：运行功能包节点程序

命令格式：ros2 run pkg_name node_name

pkg_name：功能包名字

node_name：可执行程序的名字



3、节点相关工具 ros2 node

3.1、ros2 node list

命令功能： 罗列出所有在当前域内节点名称

命令格式： ros2 node list

3.2、ros2 node info

命令功能： 查看节点详细信息，包括订阅、发布的消息，开启的服务和动作等

命令格式：ros2 node info node_name

- node_name：需要查看的节点名称

```
yahboom@yahboom-virtual-machine:~$ ros2 node info /turtlesim
/turtlesim
  Subscribers:
    /parameter_events: rcl_interfaces/msg/ParameterEvent
    /turtle1/cmd_vel: geometry_msgs/msg/Twist
  Publishers:
    /parameter_events: rcl_interfaces/msg/ParameterEvent
    /rosout: rcl_interfaces/msg/Log
    /turtle1/color_sensor: turtlesim/msg/Color
    /turtle1/pose: turtlesim/msg/Pose
  Service Servers:
    /clear: std_srvs/srv/Empty
    /kill: turtlesim/srv/Kill
    /reset: std_srvs/srv/Empty
    /spawn: turtlesim/srv/Spawn
    /turtle1/set_pen: turtlesim/srv/SetPen
    /turtle1/teleport_absolute: turtlesim/srv/TeleportAbsolute
    /turtle1/teleport_relative: turtlesim/srv/TeleportRelative
    /turtlesim/describe_parameters: rcl_interfaces/srv/DescribeParameters
    /turtlesim/get_parameter_types: rcl_interfaces/srv/GetParameterTypes
    /turtlesim/get_parameters: rcl_interfaces/srv/GetParameters
    /turtlesim/list_parameters: rcl_interfaces/srv/ListParameters
    /turtlesim/set_parameters: rcl_interfaces/srv/SetParameters
    /turtlesim/set_parameters_atomically: rcl_interfaces/srv/SetParametersAtomically
  Service Clients:

  Action Servers:
    /turtle1/rotate_absolute: turtlesim/action/RotateAbsolute
  Action Clients:
```

4、主题相关工具 ros2 topic

4.1、ros2 topic list

命令功能：罗列出当前域内的所有主题

命令格式：ros2 topic list

```
yahboom@yahboom-virtual-machine:~$ ros2 topic list
/parameter_events
/rosout
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
```

4.2、ros2 topic info

命令功能：显示主题消息类型，订阅者/发布者数量

命令格式：ros2 topic info topic_name

- topic_name：需要查询的话题的名字

```
yahboom@yahboom-virtual-machine:~$ ros2 topic info /turtle1/cmd_vel
Type: geometry_msgs/msg/Twist
Publisher count: 0
Subscription count: 1
```

4.3、ros2 topic type

命令功能：查看话题的消息类型

命令格式：ros2 topic type topic_name

- topic_name：需要查询话题类型的名字

```
yahboom@yahboom-virtual-machine:~$ ros2 topic type /turtle1/cmd_vel
geometry_msgs/msg/Twist
```

4.4、ros2 topic hz

命令功能：显示主题平均发布频率

命令格式：ros2 topic hz topic_name

- topic_name：需要查询话题频率的名字

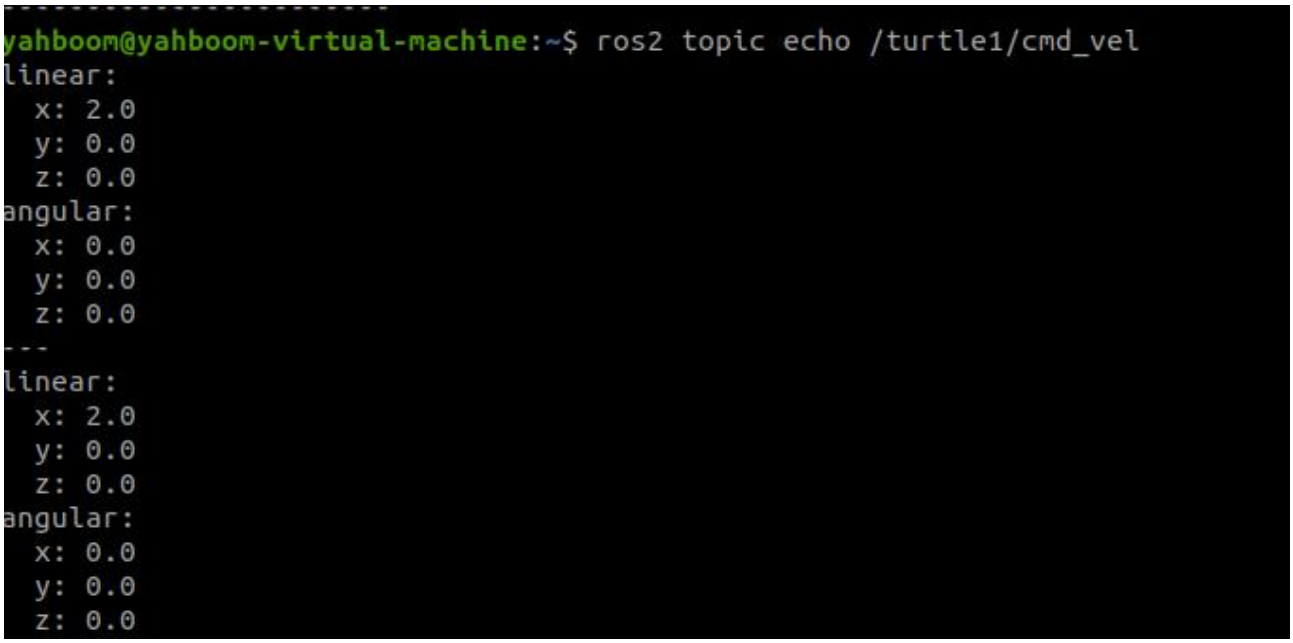
```
yahboom@yahboom-virtual-machine:~$ ros2 topic hz /turtle1/cmd_vel
average rate: 2.532
        min: 0.002s max: 6.513s std dev: 1.44588s window: 19
average rate: 4.026
        min: 0.002s max: 6.513s std dev: 1.06690s window: 36
average rate: 4.613
        min: 0.002s max: 6.513s std dev: 0.93960s window: 47
average rate: 5.803
        min: 0.002s max: 6.513s std dev: 0.80420s window: 65
average rate: 5.961
        min: 0.002s max: 6.513s std dev: 0.75605s window: 74
average rate: 5.991
        min: 0.002s max: 6.513s std dev: 0.72046s window: 82
average rate: 5.755
        min: 0.002s max: 6.513s std dev: 0.70435s window: 86
average rate: 5.568
        min: 0.002s max: 6.513s std dev: 0.68547s window: 91
average rate: 5.419
        min: 0.002s max: 6.513s std dev: 0.67609s window: 94
```

4.5、ros2 topic echo

命令功能：在终端打印主题消息，类似于一个订阅者

命令格式：ros2 topic echo topic_name

- topic_name：需要打印消息的话题的名字



4.5、ros2 topic pub

命令功能：在终端发布指定话题消息

命令格式：ros2 topic pub topic_name message_type message_content

- topic_name：需要发布话题消息的话题的名字
- message_type：话题的数据类型
- message_content：消息内容

默认是以 1Hz 的频率循环发布，可以设置以下参数，

- 参数-1 只发布一次，ros2 topic pub -1 topic_name message_type message_content
- 参数-t count 循环发布 count 次结束，ros2 topic pub -t count topic_name message_type message_content
- 参数-r count 以 count Hz 的频率循环发布，ros2 topic pub -r count topic_name message_type message_content

```
ros2 topic pub turtle1/cmd_vel geometry_msgs/msg/Twist "{linear: {x: 0.5, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.2}}"
```

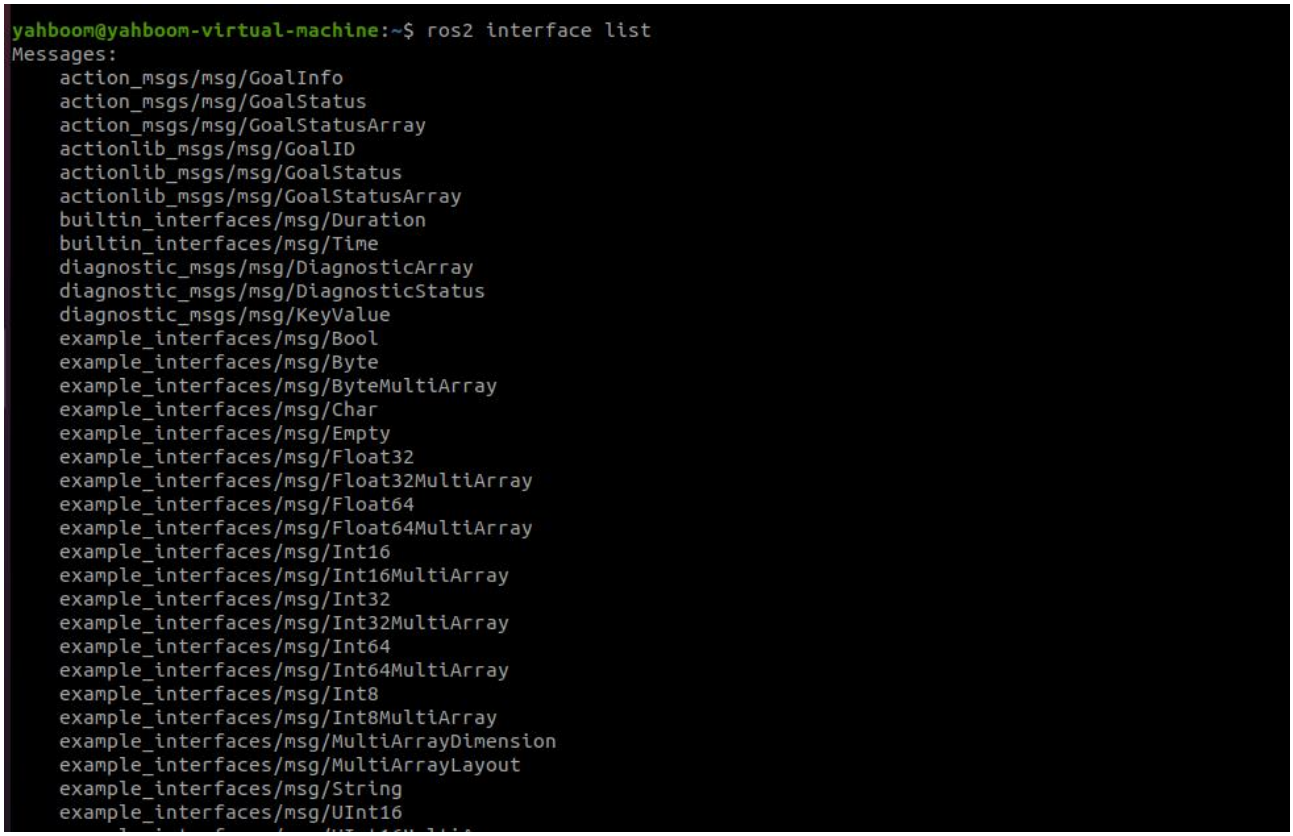
这里需要注意的是冒号后是有个空格。

5、接口相关工具 ros2 interface

5.1、ros2 interface list

命令功能：罗列当前系统的所有接口，包括话题、服务、动作。

命令格式：ros2 interface list



5.2、ros2 interface show

命令功能：显示指定接口的详细内容

命令格式：ros2 interface show interface_name

- interface_name：需要显示的接口内容的名字

```
yahboom@yahboom-virtual-machine:~$ ros2 interface show sensor_msgs/msg/LaserScan
# Single scan from a planar laser range-finder
#
# If you have another ranging device with different behavior (e.g. a sonar
# array), please find or create a different message, since applications
# will make fairly laser-specific assumptions about this data

std_msgs/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
```

6、服务相关工具 ros2 service

6.1、ros2 service list

命令功能：罗列出当前域内所有的服务

命令格式：ros2 interface show interface_name

```
yahboom@yahboom-virtual-machine:~$ ros2 service list
/clear
/kill
/reset
/spawn
/teleop_turtle/describe_parameters
/teleop_turtle/get_parameter_types
/teleop_turtle/get_parameters
/teleop_turtle/list_parameters
/teleop_turtle/set_parameters
/teleop_turtle/set_parameters_atomically
/turtle1/set_pen
/turtle1/teleport_absolute
/turtle1/teleport_relative
/turtlesim/describe_parameters
/turtlesim/get_parameter_types
/turtlesim/get_parameters
/turtlesim/list_parameters
/turtlesim/set_parameters
/turtlesim/set_parameters_atomically
yahboom@yahboom-virtual-machine:~$
```

6.2、ros2 service call

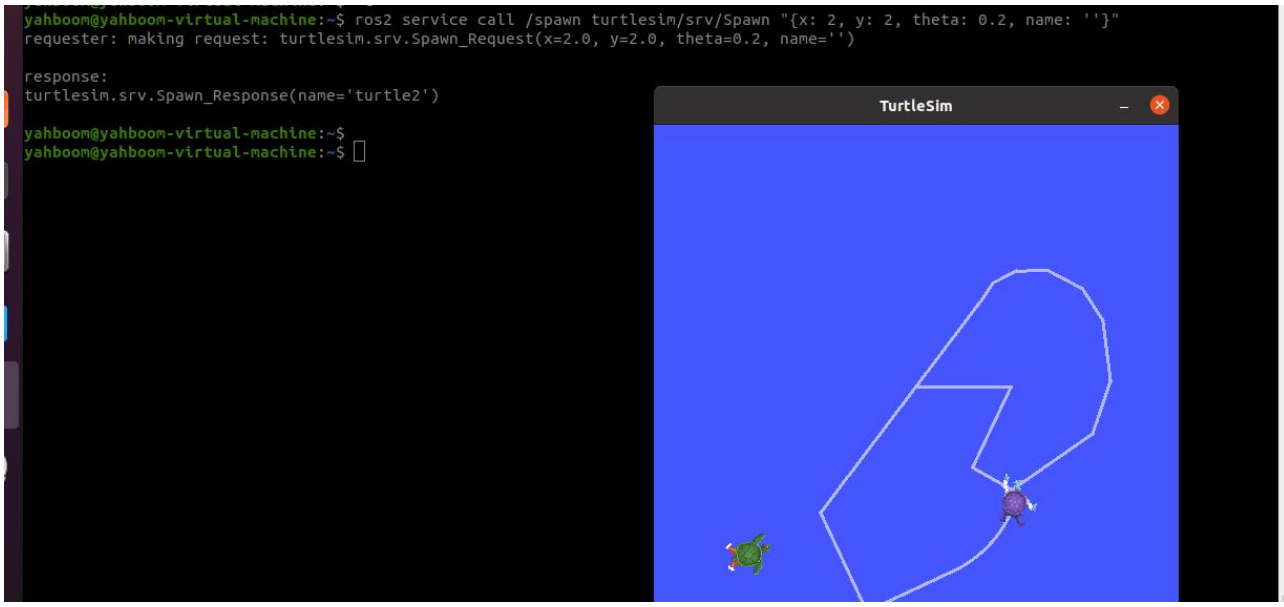
命令功能：调用指定服务

命令格式：ros2 interface call service_name service_Type arguments

- service_name：需要调用的服务
- service_Type：服务数据类型
- arguments：提供服务需要的参数

例如，调用生成海龟服务

```
ros2 service call /spawn turtlesim/srv/Spawn "{x: 2, y: 2, theta: 0.2, name: ''}"
requester: making request: turtlesim.srv.Spawn_Request(x=2.0, y=2.0, theta=0.2, name='turtle2')
```

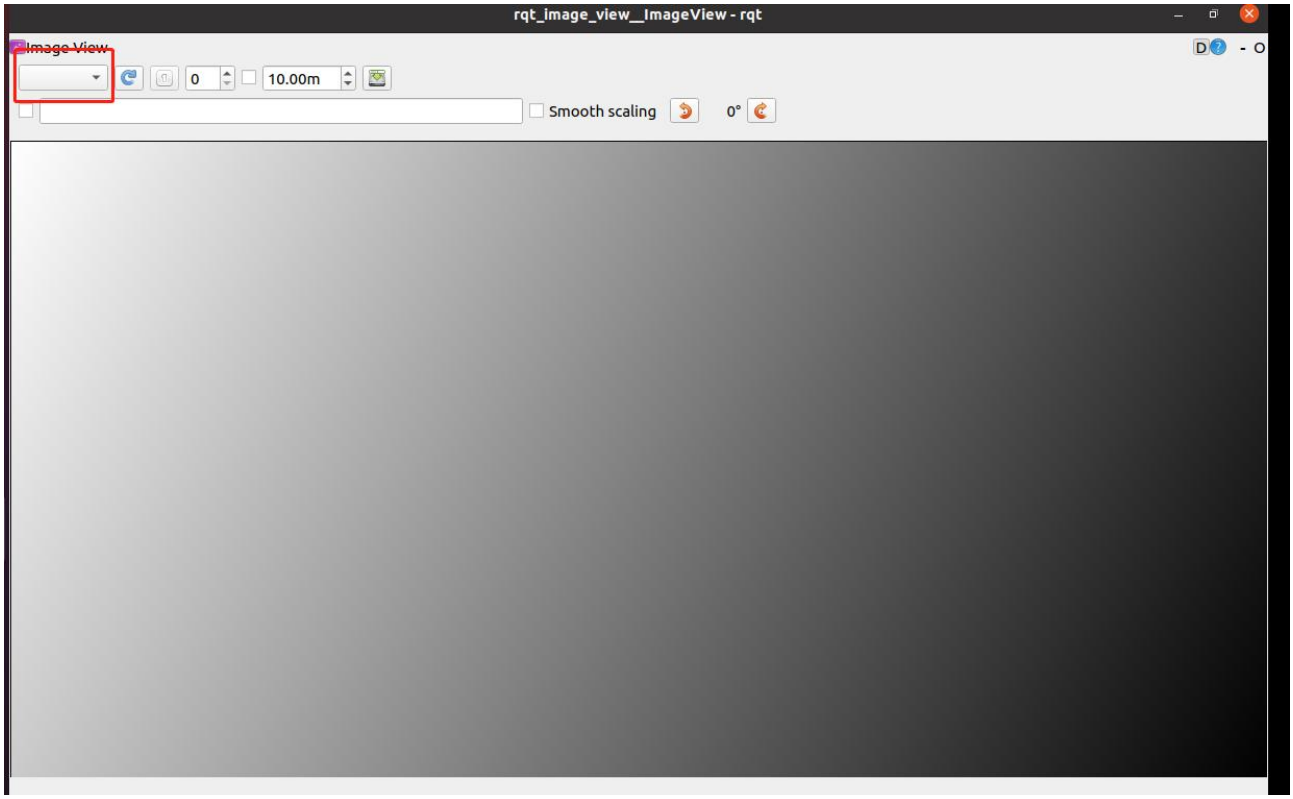


7、rqt_image_view

rosrun rqt_image_view rqt_image_view

rqt_image_view 可以用了查看图像，当前域内有发布了图像话题数据的话，可以使用这个工具查看图像，

```
ros2 run rqt_image_view rqt_image_view
```

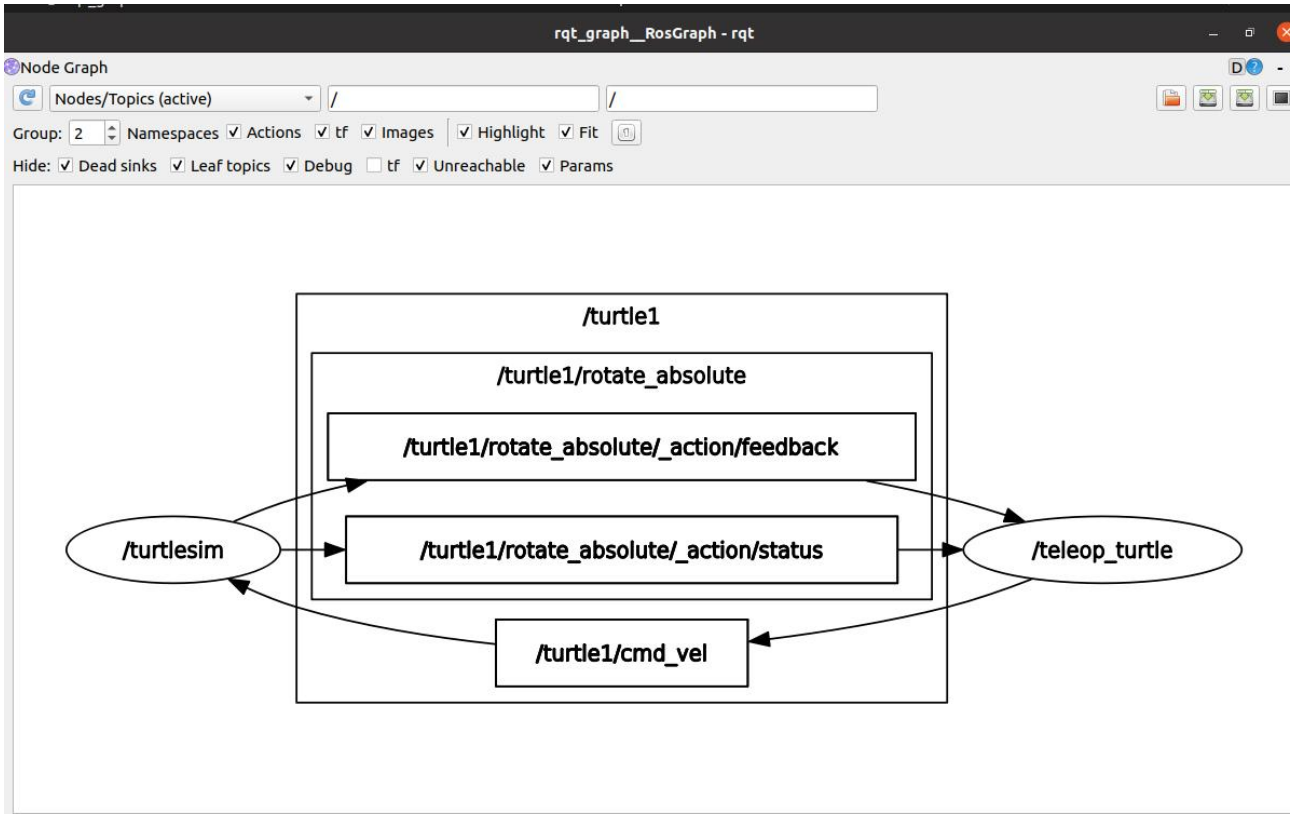


通过左上角选择的图像话题显示图像数据。

8、rqt_graph

rqt_graph 可以用了查看当前域内运行了哪些节点以及节点之间的话题通讯，使用以下命令开启，

```
ros2 run run rqt_graph rqt_graph
```



9、rviz2

Rviz 的核心框架是基于 Qt 可视化工具打造的一个开放式平台，按照 ROS 中的消息发布对应的话题，就可以看到图形化的效果了。在 ROS2 中，使用 rviz2 启动 rviz 工具。

```
rviz2
```

通过以上步骤可以通过插件或者通过话题添加可视化数据，一般选择的是通过话题添加。

10、tf2_tools

tf2_tools 可以查看当前的 TF 树，会在输入命令的终端下生成 frame.pdf 文件。

```
ros2 run tf2_tools view_frames.py
```

2、ROS2 话题通讯

3、ROS2 话题通讯

话题通讯是 ROS2 使用频率最高的一种通信方式，话题通信是基于发布订阅模式，有发布者发布指定话题的数据，订阅者只要订阅了该话题的数据，就可以接收到数据。接下来就说明下如何使用 Python 语言实现节点之间的话题通讯。

1、新建工作空间

终端输入，

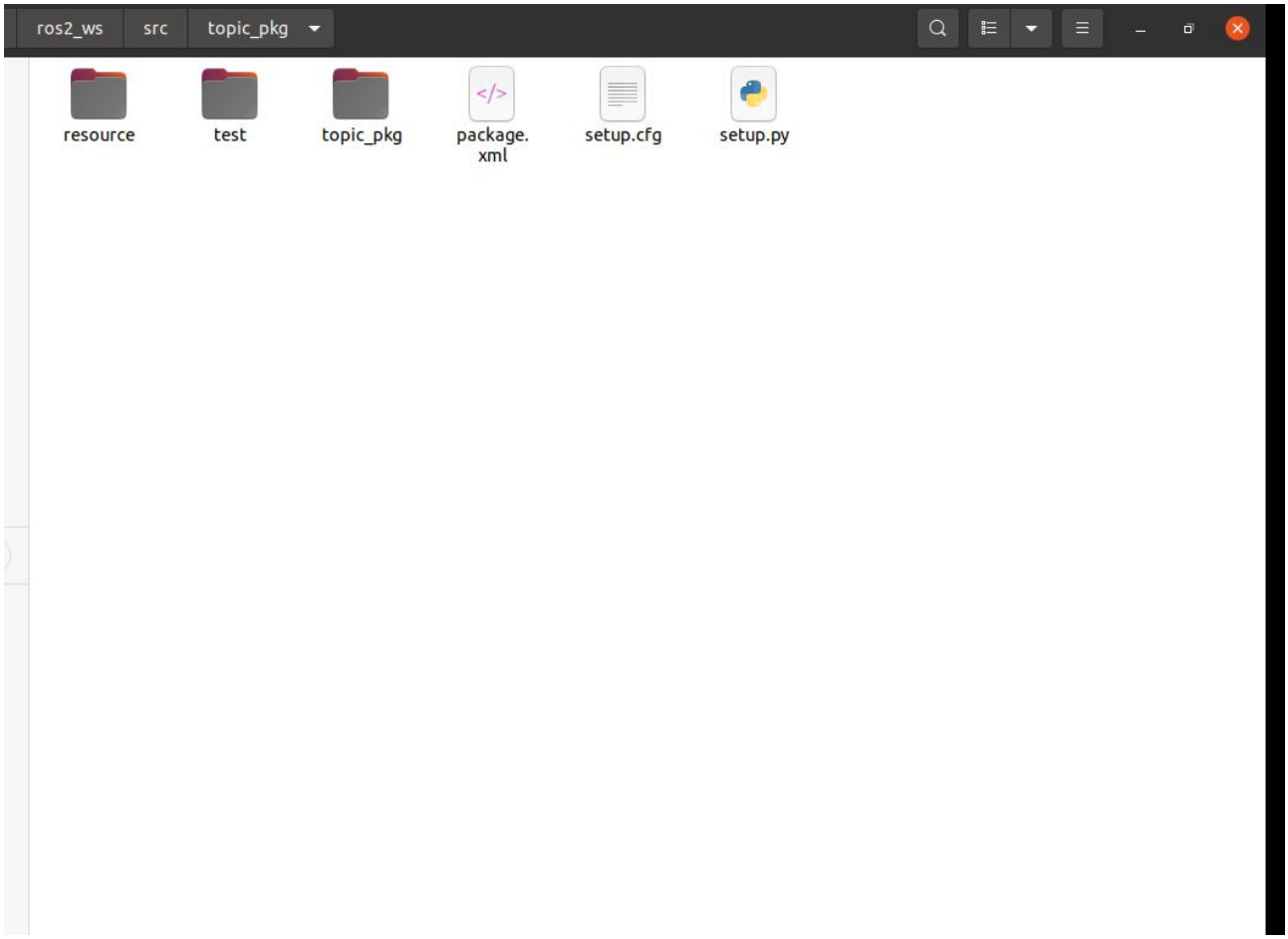
```
mkdir -p ros2_ws/src
cd ros2_ws/src
```

ros2_ws 是工作空间的名字，src 是存放功能包的目录。

2、新建功能包

终端输入，

```
cd ~/ros2_ws/src
ros2 pkg create --build-type ament_python topic_pkg
```



切换到 topic_pkg 文件夹中，有以下几个文件和文件夹，我们编写的 python 程序放在该目录下的 topic_pkg 文件夹下，也就是，

~/ros2_ws/src/topic_pkg/topic_pkg

3、编写发布者 python 文件

3.1、程序源码

新建一个文件，命名为 publisher_demo.py，

```
cd ~/ros2_ws/src/topic_pkg/topic_pkg
sudo gedit publisher_demo.py
```

把以下部分复制到该文件中，

```
import rclpy
from rclpy.node import Node
from std_msgs.msg import String
class Topic_Pub(Node):
    def __init__(self,name):
        super().__init__(name)
        self.pub = self.create_publisher(String,"/topic_demo",1)
        self.timer = self.create_timer(1,self.pub_msg)

    def pub_msg(self):
        msg = String()
```

```
        msg.data = "Hi,I send a message."
        self.pub.publish(msg)

def main():
    rclpy.init()
    pub_demo = Topic_Pub("publisher_node")
    rclpy.spin(pub_demo)
```

这里是用模块化的思想来实现，这种方法有利于我们移植、修改代码。首先定义了一个类，类名字是 Topic_Pub，里边有两个部分，一个是 init，另一个是 pub_msg。init 是初始化这个类本身，pub_msg 是这个类的方法，也就是说，只要我们创建了这个类的对象，就可以使用里边的变量和方法。

```
#导入 rclpy 库
import rclpy
from rclpy.node import Node
#导入 String 字符串消息
from std_msgs.msg import String
#创建一个继承于 Node 基类的 Topic_Pub 节点子类  传入一个参数 name
def __init__(self,name):
        super().__init__(name)
```

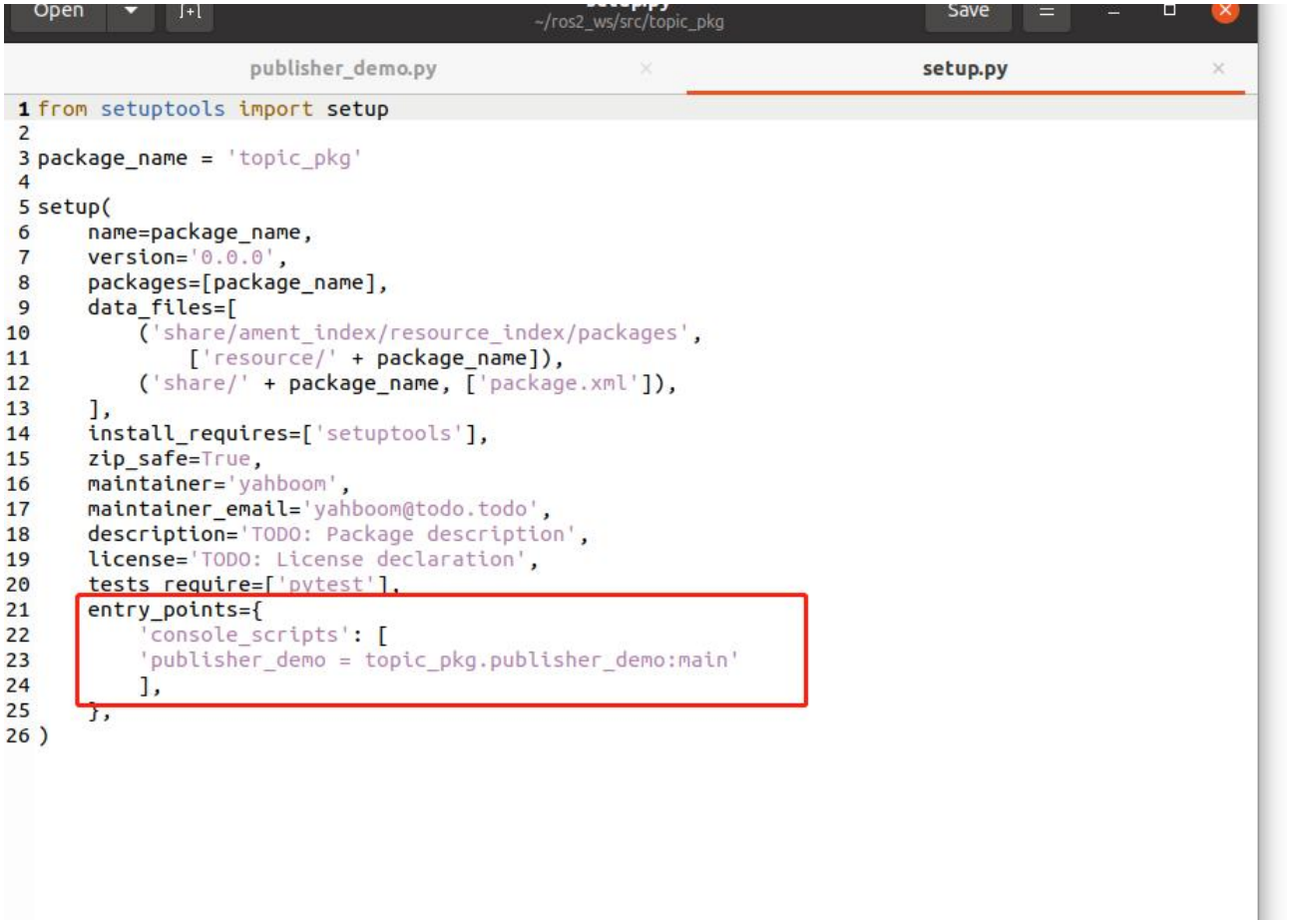
```
        #创建一个发布者，使用 create_publisher 的函数，传入的参数分别是：
        #话题数据类型、话题名称、保存消息的队列长度
          self.pub = self.create_publisher(String,"/topic_demo",1)
        #创建一个定时器，间隔 1s 进入中断处理函数，传入的参数分别是：
        #中断函数执行的间隔时间，中断处理函数
          self.timer = self.create_timer(1,self.pub_msg)
#定义中断处理函数
def pub_msg(self):
          msg = String()   #创建一个 String 类型的变量 msg
          msg.data = "Hi,I send a message." #给 msg 里边的 data 赋值
          self.pub.publish(msg) #发布话题数据

#主函数
def main():
    rclpy.init() #初始化
    pub_demo = Topic_Pub("publisher_node") #创建 Topic_Pub 类对象，传入的参数就是节点的名字
    rclpy.spin(pub_demo) #执行 rclpy.spin 函数，里边传入一个参数，参数是刚才创建好的 Topic_Pub 类对象
```

3.2、修改 setup.py 文件

终端输入，

```
cd ~/ros2_ws/src/topic_pkg
sudo gedit setup.py
```

找到如下图所示的位置，



在'console_scripts': []里边添加以下内容，

```
'publisher_demo = topic_pkg.publisher_demo:main'
```

代码解释：

可执行文件的名字是我们自定义的，也就是我们 ros2 run 时候需要执行哪个程序，功能包名字是你的 python 文件所在的哪个功能包，后边跟的是 python 文件的名字，最后的 main 是程序执行入口。只要是 python 格式写的程序，编译生成可执行文件，就都需要在这里按照以上格式添加。

3.3、编译工作空间

```
cd ~/ros2_ws
colcon build
```

编译成功，输入以下指令，把工作空间添加到系统环境变量，

```
echo "source ~/ros2_ws/install/setup.bash" >> ~/.bashrc
```

然后，重新打开终端或者刷新环境变量生效，

```
source ~/.bashrc
```

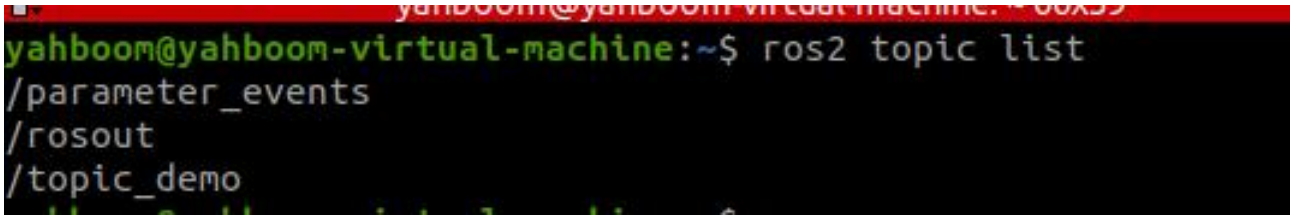
3.4、运行程序

终端输入，

```
ros2 run topic_pkg publisher_demo
```

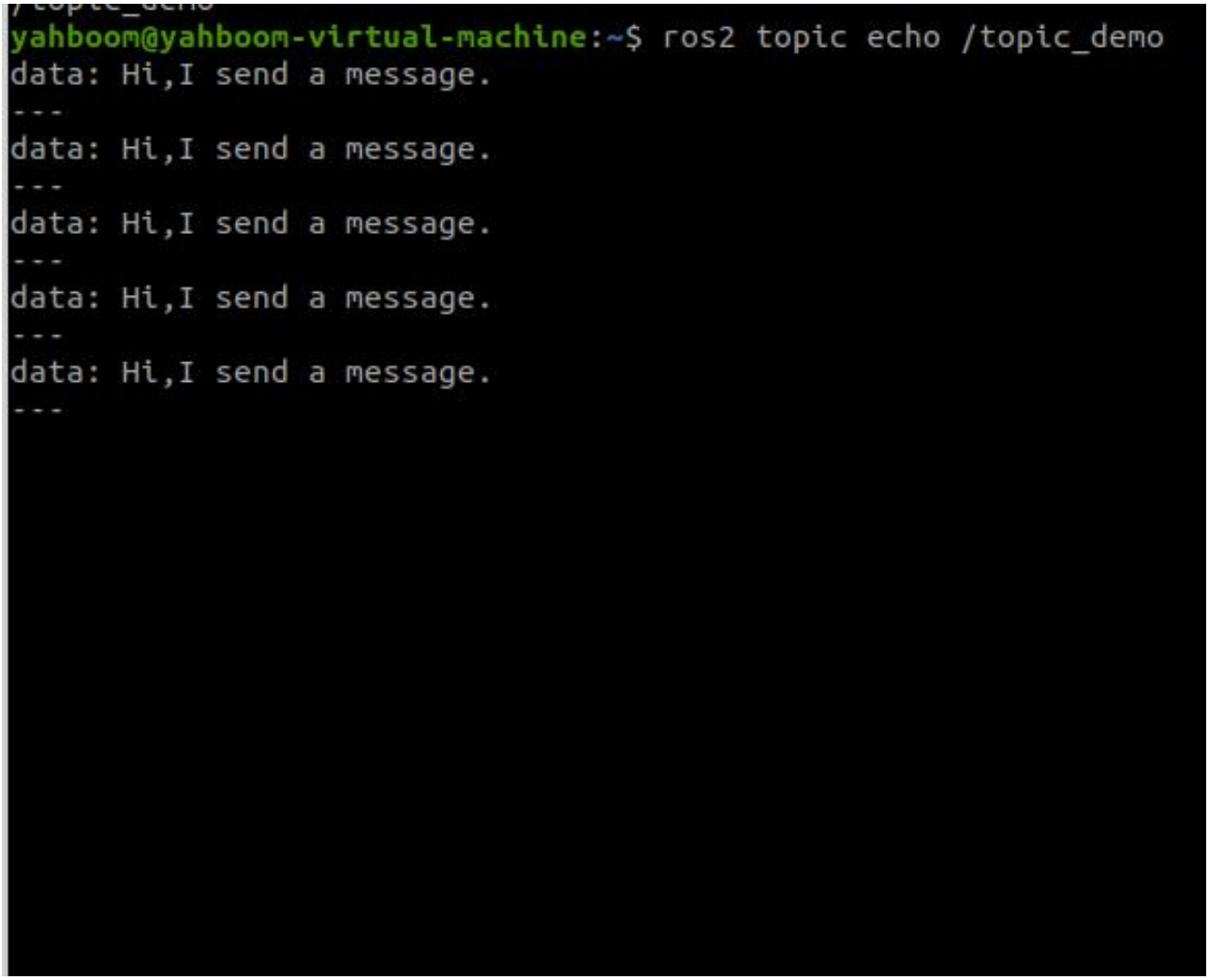
程序成功运行后是没有打印任何东西的，我们可以通过 ros2 topic 工具来查看数据，首先，先查看这个是否有话题发布，终端输入，

```
ros2 topic list
```



这个 topic_demo 就是程序里定义的话题数据了，接下来，我们用 ros2 topic echo 来打印下这个数据，终端输入，

```
ros2 topic echo /topic_demo
```



可以看出，终端打印的"Hi,I send a message."与我们代码里边的 msg.data = "Hi,I send a message."一致。

4、编写订阅者 python 文件

4.1、程序源码

新建一个文件，命名为 subscriber_demo.py，

```
cd ~/ros2_ws/src/topic_pkg/topic_pkg
sudo gedit subscriber_demo.py
```

把以下部分复制到该文件中，

```
#导入相关的库
import rclpy
from rclpy.node import Node
from std_msgs.msg import String

class Topic_Sub(Node):
    def __init__(self,name):
        super().__init__(name)
        #创建订阅者使用的是 create_subscription，传入的参数分别是：话题数据类型，话题名称，回调函数名称，队列长度
        self.sub = self.create_subscription(String,"/topic_demo",self.sub_callback,1)
    #回调函数执行程序：打印接收的到信息
    def sub_callback(self,msg):
        print(msg.data)
def main():
    rclpy.init() #ROS2 Python 接口初始化
    sub_demo = Topic_Sub("subscriber_node") #  创建对象并进行初始化
    rclpy.spin(sub_demo)
```

4.2、修改 setup.py 文件

终端输入，

```
cd ~/ros2_ws/src/topic_pkg
sudo gedit setup.py
```

找到如下图所示的位置，

```
from setuptools import setup

package_name = 'topic_pkg'

setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name],
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='yahboom',
    maintainer_email='yahboom@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
        'publisher_demo = topic_pkg.publisher_demo:main',
        'subscriber_demo = topic_pkg.subscriber_demo:main'
        ],
    },
)
```

在'console_scripts': []里边按照格式添加以下内容，**注意新加的一句需要和上一句用逗号隔开，否则会报错**

'subscriber_demo = topic_pkg.subscriber_demo:main'

'subscriber_demo = topic_pkg.subscriber_demo:main'

4.3、编译工作空间

```
cd ~/ros2_ws
colcon build
```

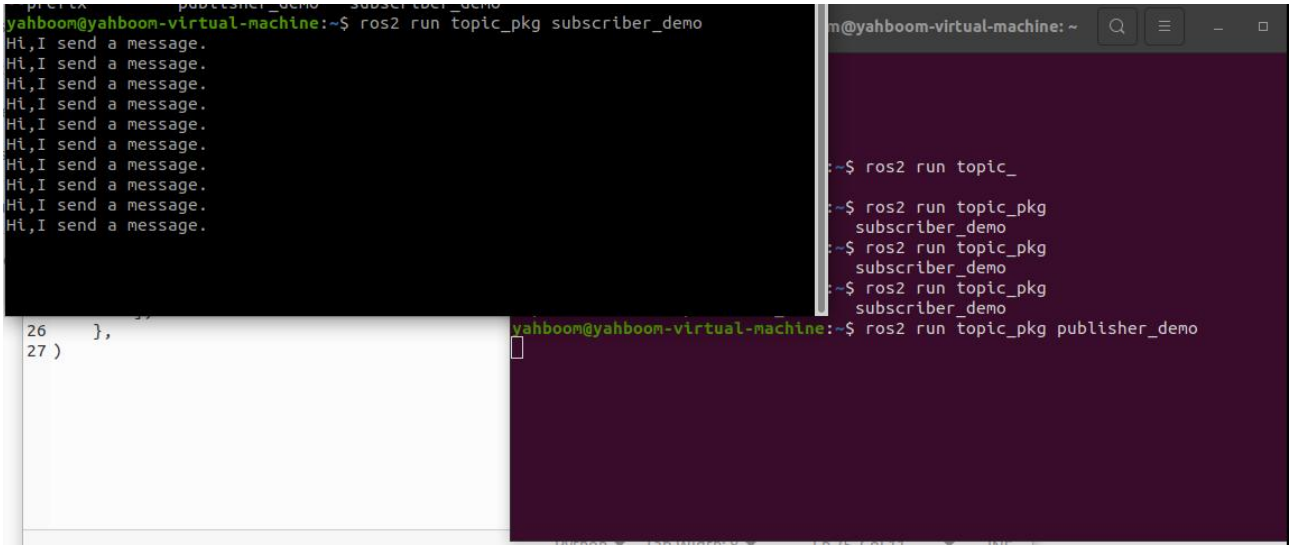
编译完成后，刷新下工作空间的环境变量，

```
source ~/ros2_ws/install/setup.bash
```

4.4、运行程序

终端输入，

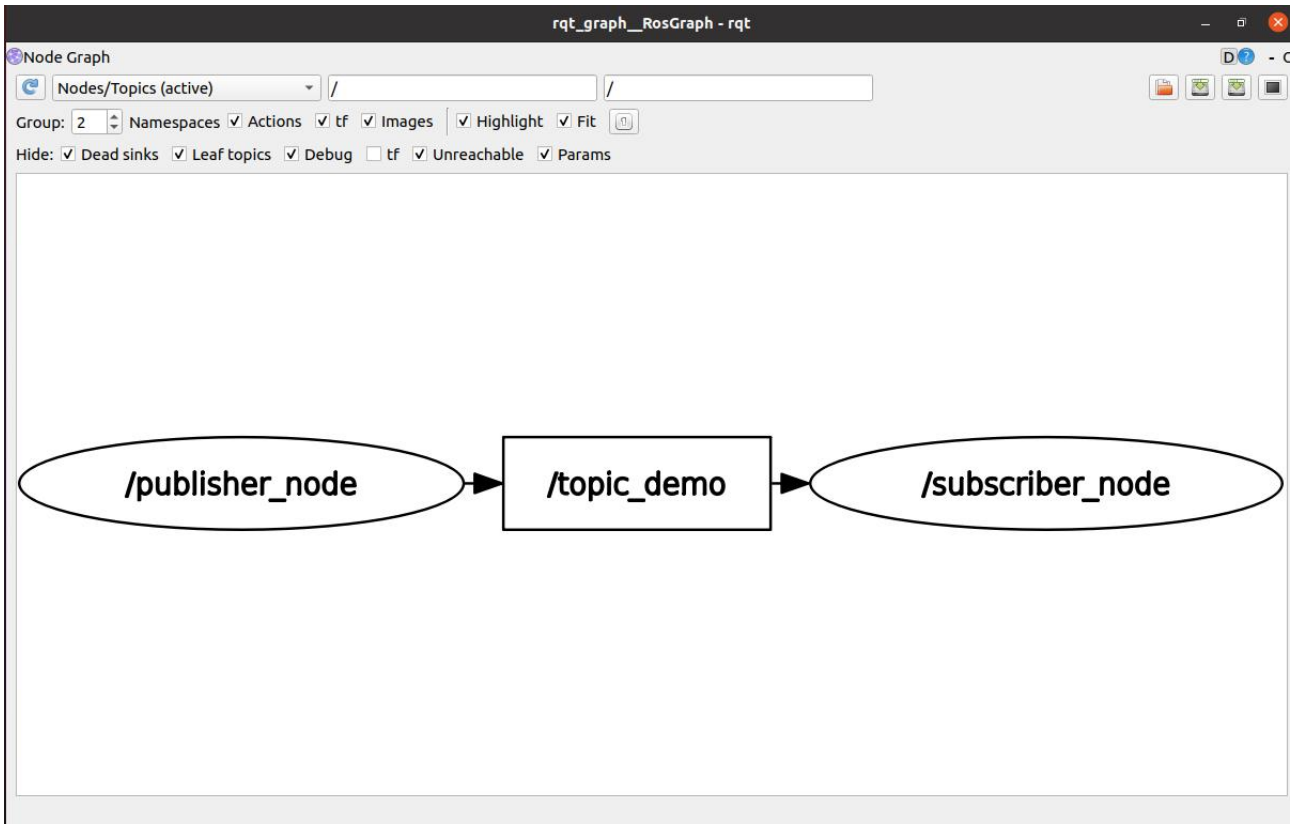
```
#启动发布者节点
ros2 run topic_pkg publisher_demo
#启动订阅者节点
ros2 run topic_pkg subscriber_demo
```



如上图所示，运行订阅者这点的终端会打印发布者发布的/topic_demo 的信息。

可以用以下命令查看两个节点之间的话题通讯，

```
ros2 run rqt_graph rqt_graph
```



5、总结

5.1、python 编写话题通讯程序框架

```
1、导入库文件
import rclpy
from rclpy.node import Node
2、创建类
class ClassName(Node):
    def __init__(self,name):
        super().__init__(name)
        #create subcriber
        self.sub =
self.create_subscription(Msg_Type,Topic_Name,self.CallbackFunction,Msg_Line_Size
)

        #create publisher
        selt.pub = self.create_publisher(Msg_Type,Topic_Name,Msg_Line_Size)

        #create timer
        self.timer = self.create_timer(Timer,self.TimerExcuteFunction)

    def CallbackFunction(self):
        ...
    def TimerExcuteFunction(self):
        ...
3、主函数main:主要是初始化节点，创建对象
def main():
    rclpy.init()
    class_object = ClassName("node_name")
    rclpy.spin(class_object)
```

5.2、修改 setup.py 文件内容

使用 python 编写节点程序，需要修改 setup.py 文件，参考上边内容，添加生成自己的节点程序。

3、ROS2 服务通讯

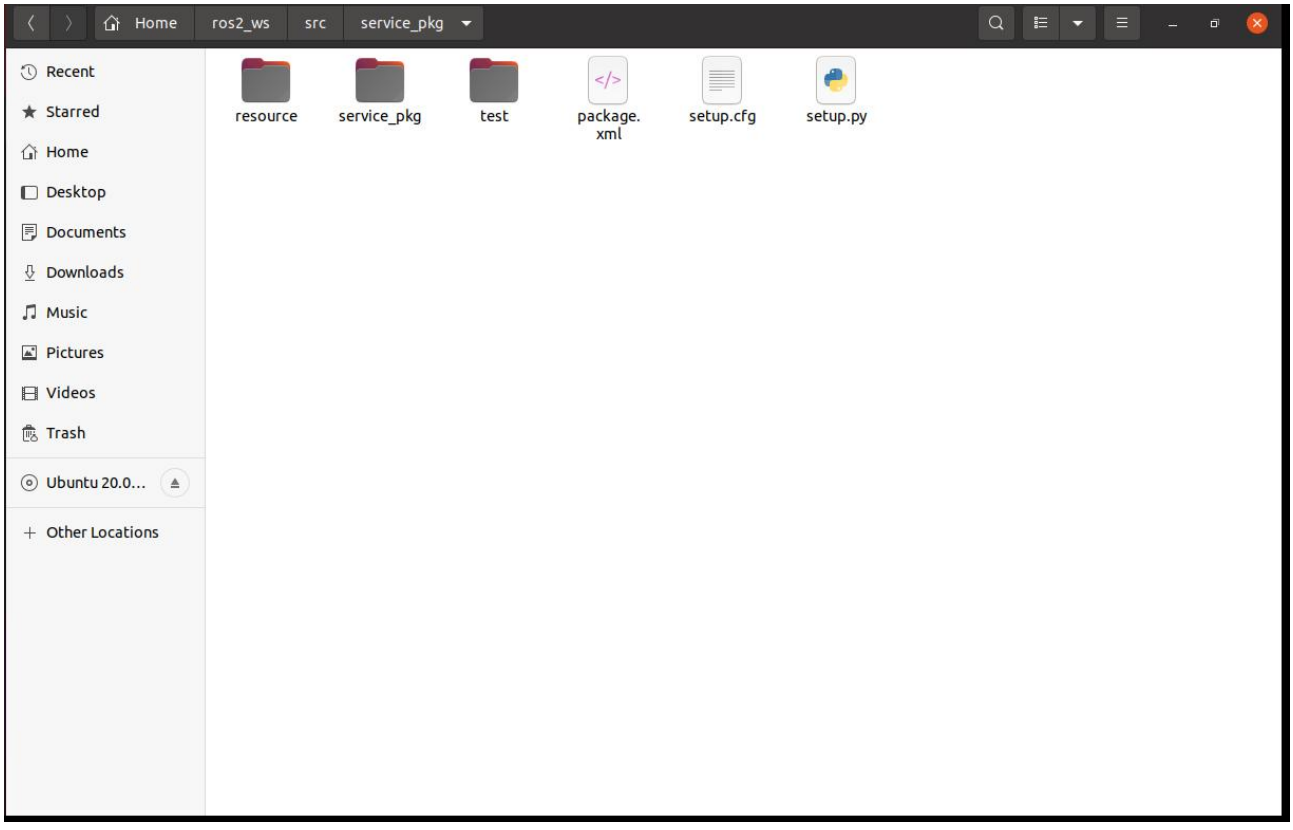
ROS2 服务通讯

服务通讯是 ROS2 节点通讯方式的其中一种，它与话题通讯不一样的是，服务通讯有反馈的机制，它会反馈服务的结果。因此，服务通讯大多数运用在需要反馈结果的程序中，比如说小海龟的例程，调用 Spawn 服务生成一个海龟，服务完成后（生成了海龟后），会打印出海龟的名字。接下来就说明下如何使用 Python 语言实现节点之间的服务通讯。

1、新建功能包

在之前建立的工作空间目录的 src 新建功能包，终端输入，

```
cd ~/ros2_ws/src
ros2 pkg create --build-type ament_python service_pkg
```



2、编写服务端 python 文件

2.1、程序源码

新建一个文件，命名为 server_demo.py，

```
cd ~/ros2_ws/src/service_pkg/service_pkg
sudo gedit server_demo.py
```

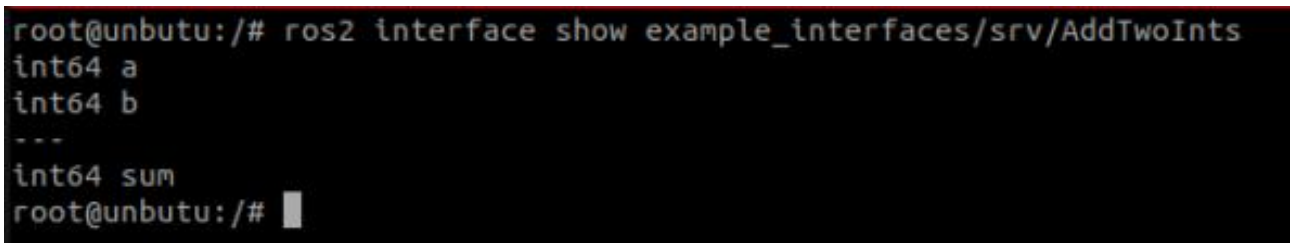
把以下部分复制到该文件中，

```
#导入相关的库文件
import rclpy
from rclpy.node import Node
from example_interfaces.srv import AddTwoInts

class Service_Server(Node):
    def __init__(self,name):
        super().__init__(name)
        #创建一个服务端，使用的是 create_service 函数，传入的参数分别是：
        #服务数据的数据类型、服务的名称，服务回调函数（也就是服务的内容）
```

```
        self.srv = self.create_service(AddTwoInts, '/add_two_ints',
self.Add2Ints_callback)
    #这里的服务回调函数的内容是把两个整型数相加，然后返回相加的结果
    def Add2Ints_callback(self,request,response):
        response.sum = request.a + request.b
        print("response.sum = ",response.sum)
        return response
def main():
    rclpy.init()
    server_demo = Service_Server("publisher_node")
    rclpy.spin(server_demo)
```

重点看下服务回调函数，Add2Ints_callback，这里需要传入的参数除了 self，还有就是 request 和 response，request 是服务需要的参数，response 是服务的反馈结果。request.a 和 request.b 是 request 部分的内容，response.sum 是 response 部分的内容，这里首先看看下 AddTwoInts 这个类型的数据是怎么样的，可以使用以下命令查看，

```
ros2 interface show example_interfaces/srv/AddTwoInts
```



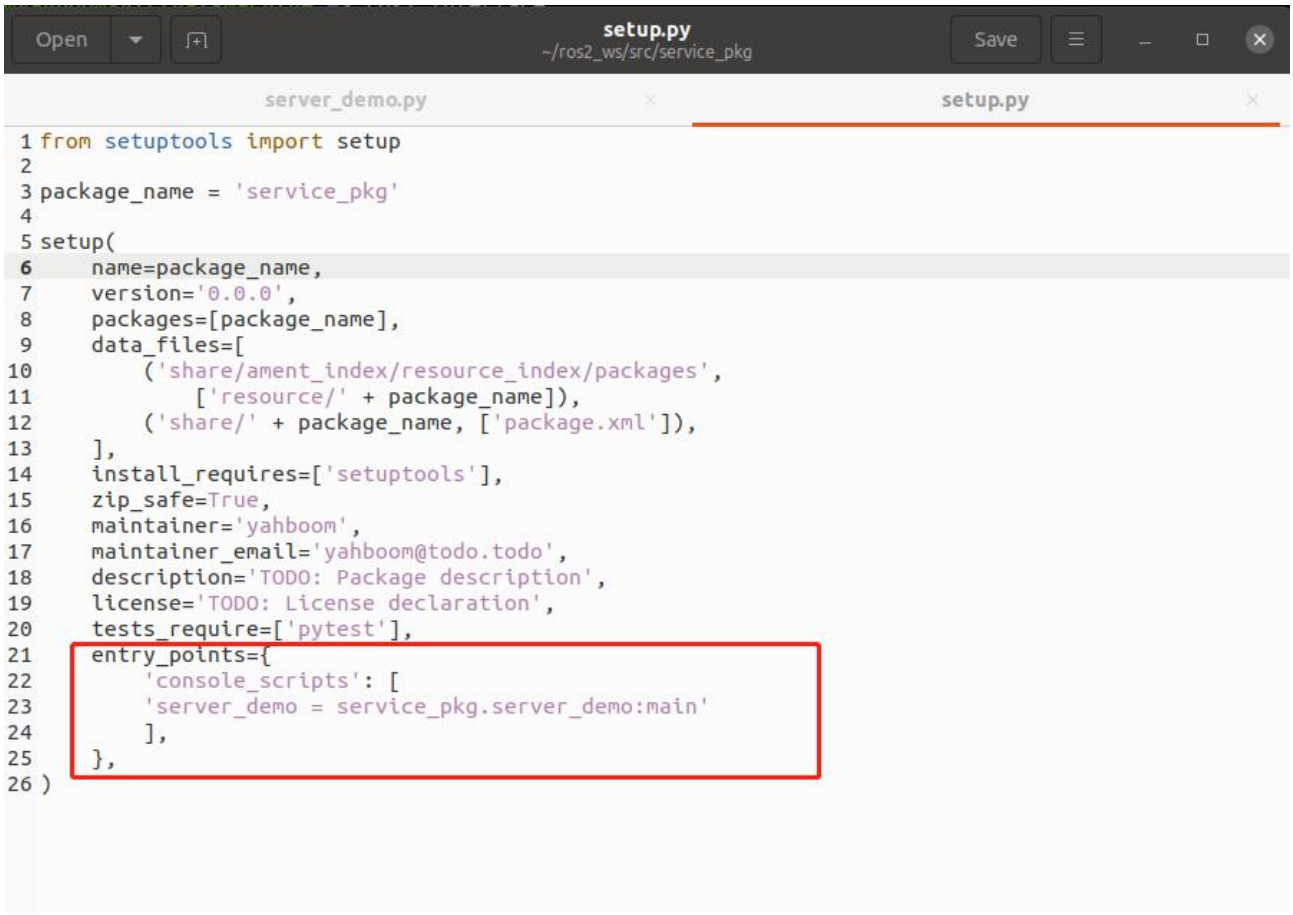
---部分把该类型的数据划分成了两个部分，上边代表的是 request，下边代表的是 response。然后各自的领域中又各自的变量，比如 int64 a、int64 b，所有在再传入参数的是，需要指定 a、b 的值是是多少。同样，反馈的结果也需要指定 sum 的值是多少。

2.2、修改 setup.py 文件

终端输入，

```
cd ~/ros2_ws/src/service_pkg
sudo gedit setup.py
```

找到如下图所示的位置，



在'console_scripts': []里边添加以下内容。

```
'server_demo = service_pkg.server_demo:main'
```

2.3、编译工作空间

```
cd ~/ros2_ws
colcon build
```

编译完成后，刷新下工作空间的环境变量，

```
source ~/ros2_ws/install/setup.bash
```

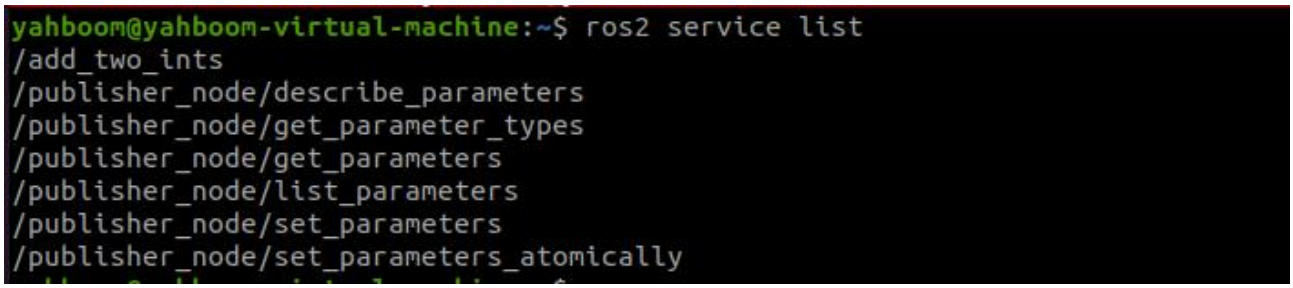
2.4、运行程序

终端输入，

```
ros2 run service_pkg server_demo
```

运行后，由于没有调用该服务，所以没有反馈数据，可以通过命令行方式调用该服务，首先查询当前有哪些服务，终端输入，

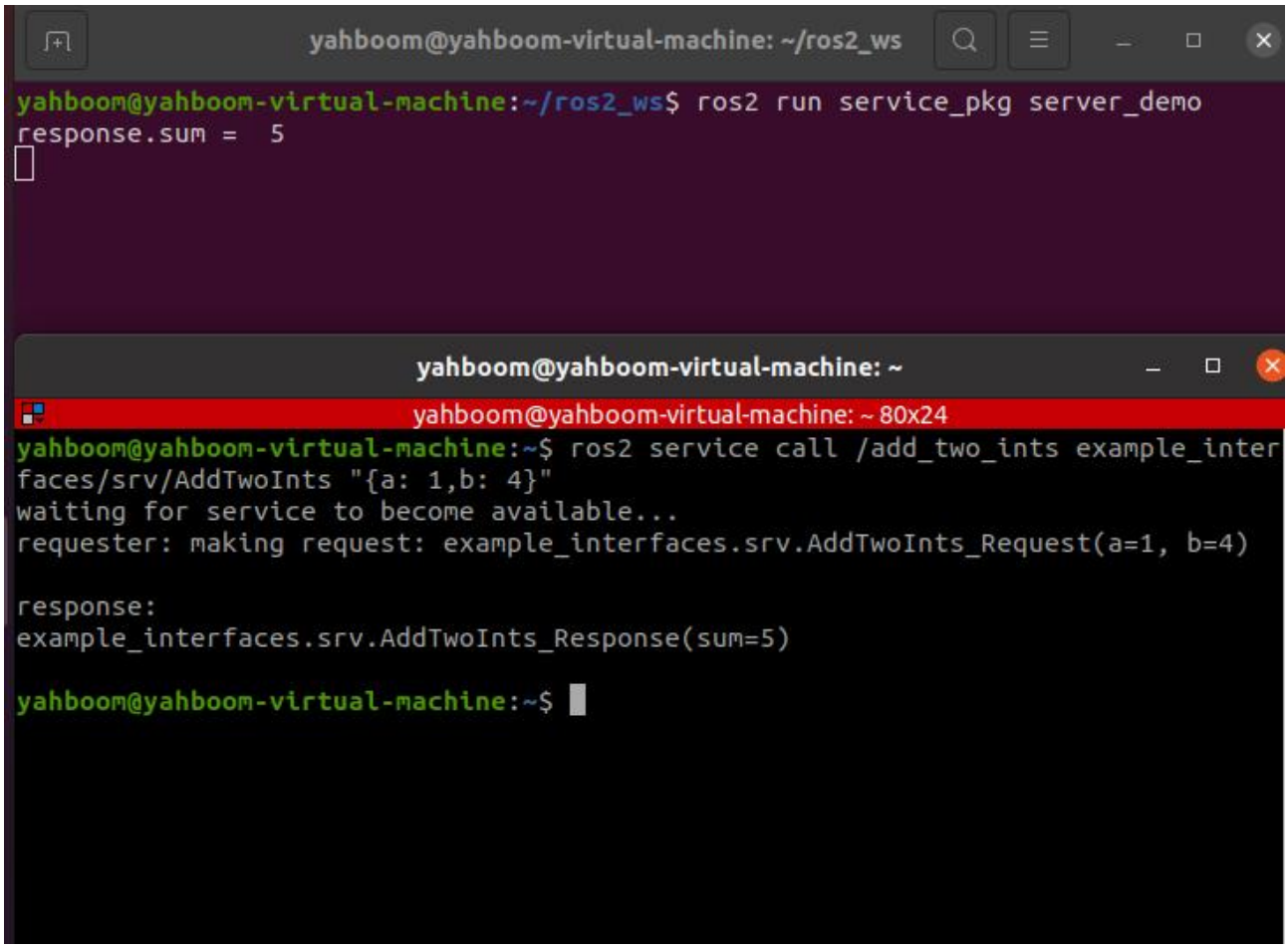
```
ros2 service list
```



/add_two_ints 就是我们需要调用的服务，通过以下命令进行调用，终端输入，

```
ros2 service call /add_two_ints example_interfaces/srv/AddTwoInts "{a: 1,b: 4}"
```

这里我们把 a 的值赋值成 1，b 的值赋值成 4，也就是调用服务计算 1 和 4 的和，



由上图可以看出，调用了服务后，反馈回来的结果是 5，运行服务端的终端也打印了反馈的值。

3、编写客户端 python 文件

3.1、程序源码

新建一个文件，命名为 client_demo.py，

```
cd ~/ros2_ws/srcservice_pkg/service_pkg
sudo gedit client_demo.py
```

把以下内容复制到里边，

```
#导入相关的库
import rclpy
from rclpy.node import Node
from example_interfaces.srv import AddTwoInts

class Service_Client(Node):
    def __init__(self,name):
        super().__init__(name)
        #创建客户端，使用的是 create_client 函数，传入的参数是服务数据的数据类型、服务的话题名称
        self.client = self.create_client(AddTwoInts,'/add_two_ints')
        # 循环等待服务器端成功启动
        while not self.client.wait_for_service(timeout_sec=1.0):
            print("service not available, waiting again...")
        # 创建服务请求的数据对象
        self.request = AddTwoInts.Request()

    def send_request(self):
        self.request.a = 10
        self.request.b = 90
```

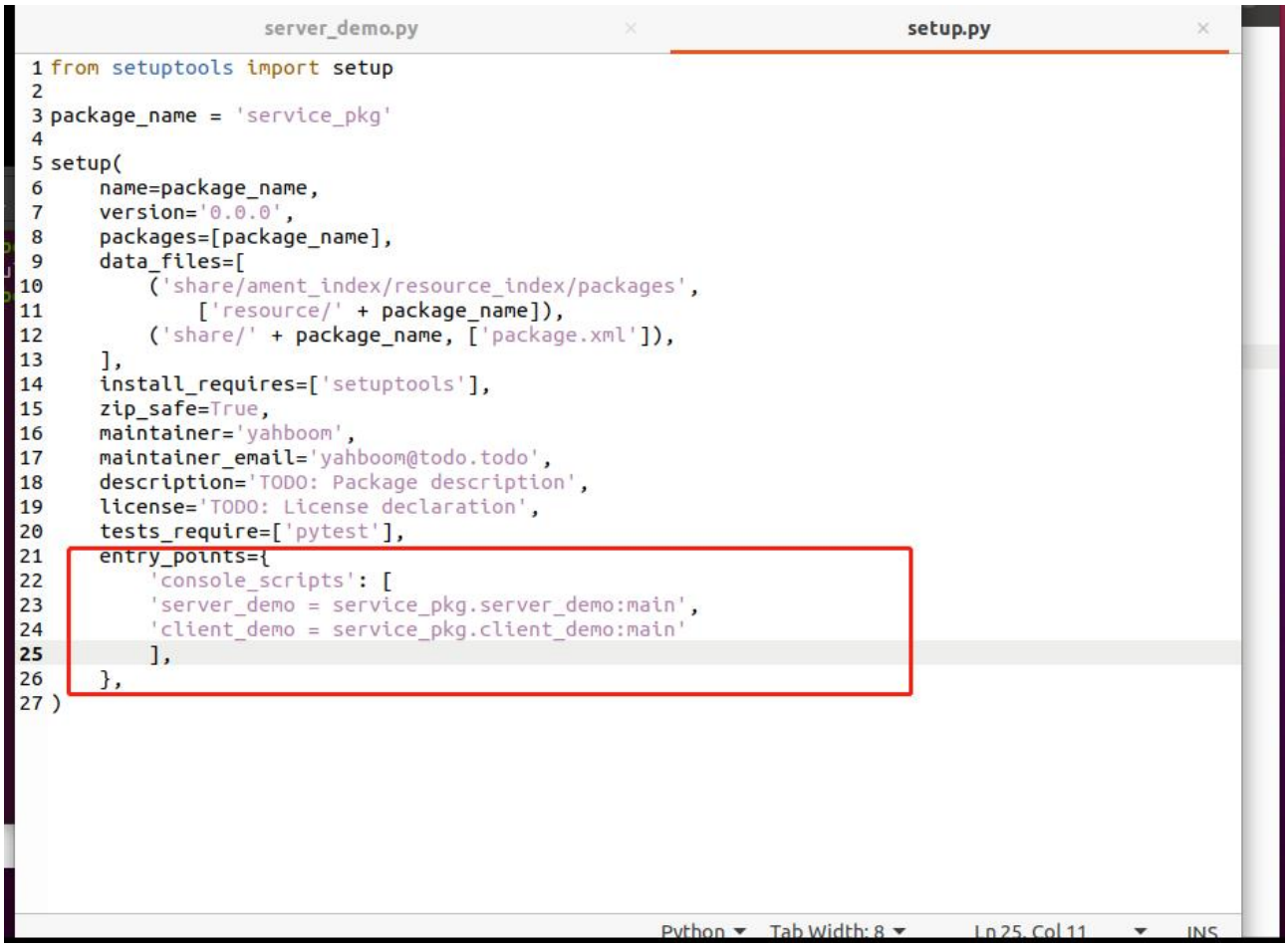
```
        #发送服务请求
            self.future = self.client.call_async(self.request)

def main():
    rclpy.init() #节点初始化
    service_client = Service_Client("client_node") #创建对象
    service_client.send_request() #发送服务请求
    while rclpy.ok():
            rclpy.spin_once(service_client)
        #判断数据是否处理完成
            if service_client.future.done():
                try:
                 #获得服务反馈的信息并且打印
                    response = service_client.future.result()
                    print("Result = ",response.sum)
                except Exception as e:
                    service_client.get_logger().info('Service call
failed %r' % (e,))

            break
```

3.2、修改 setup.py 文件

终端输入，

```
cd ~/ros2_ws/src/topic_pkg
sudo gedit setup.py
```

找到如下图所示的位置，



在'console_scripts': []里边添加以下内容，**注意新加的一句需要和上一句用逗号隔开，否则会报错**

```
'client_demo = service_pkg.client_demo:main'
```

```
'client_demo = service_pkg.client_demo:main'
```

3.3、编译工作空间

```
cd ~/ros2_ws
colcon build
```

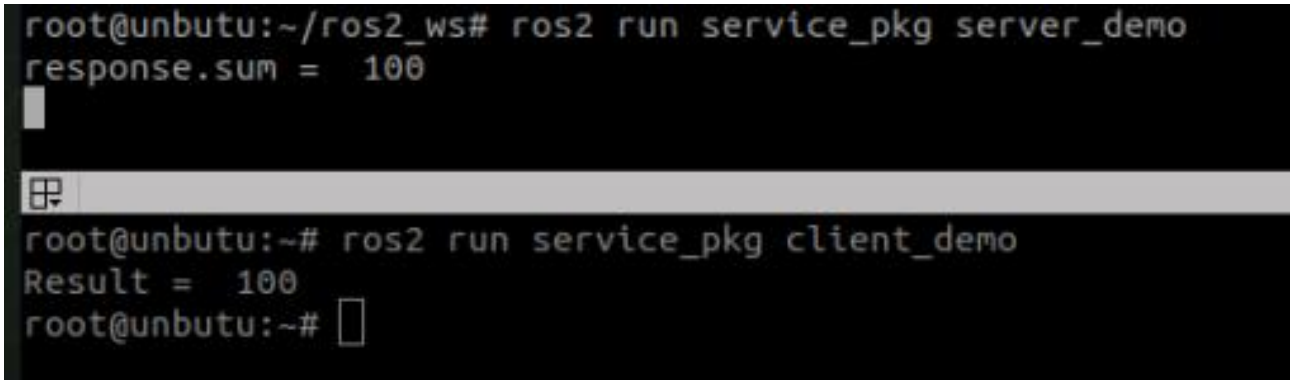
编译完成后，刷新下工作空间的环境变量，

```
source ~/ros2_ws/install/setup.bash
```

3.4、运行程序

终端输入，

```
#启动服务端
ros2 run service_pkg server_demo
#启动客户端
ros2 run service_pkg client_demo
```



先运行服务端，然后运行客户端，客户端提供 a=10，b=90，服务端进行求和，得到结果是 100，结果在两者终端打印。

4、ROS2launch 文件启动

5、ROS2launch 文件启动

ROS2 中，launch 用于多节点启动和配置程序运行参数等功能，ROS2 的 launch 文件格式有 xml、yaml 和 python 格式。本节课程以 python 格式的 launch 文件为例，相对于另外两种格式，python 格式的更加灵活：

python 拥有众多的函数库，可以在启动文件中使用；

ROS2 通用启动特性和特定启动特性是用 Python 编写的，因此可以访问 XML 和 YAML 可能没有公开的启动特性；

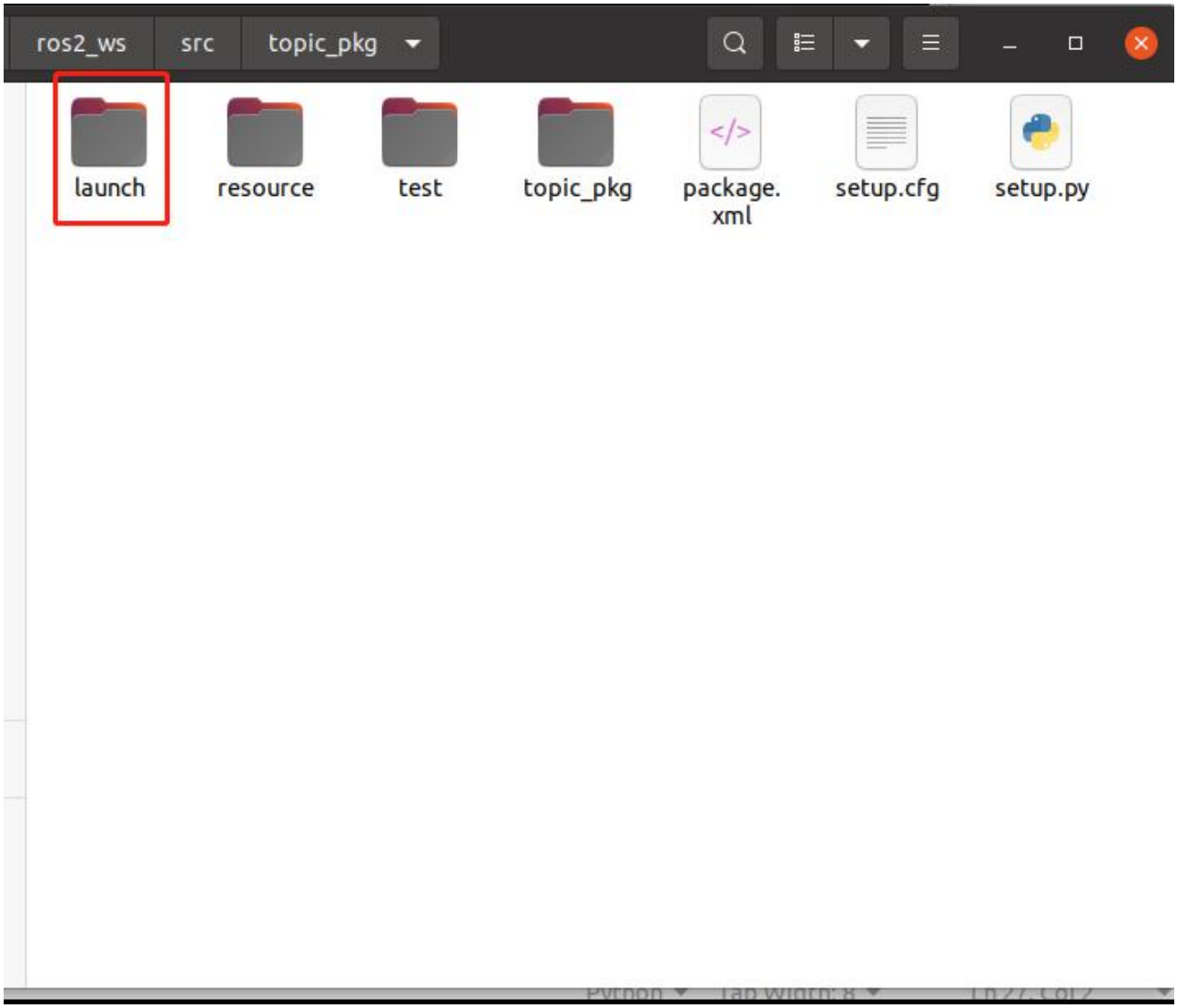
使用 python 语言编写 ROS2 launch 文件，最主要的是把每个节点、文件、脚本等抽象成一个 action，用统一的接口来启动，主要结构是，

```
def generate_launch_description():
    return LaunchDescription([
        action_1,
        action_2,
        ...
        action_n
    ])
```

1、建立文件夹存放 launch 文件

我们在之前创建的功能包的路径下，新建一个文件夹，存放 launch 文件，终端输入，

```
cd ~/ros2_ws/src/topic_pkg
mkdir launch
```

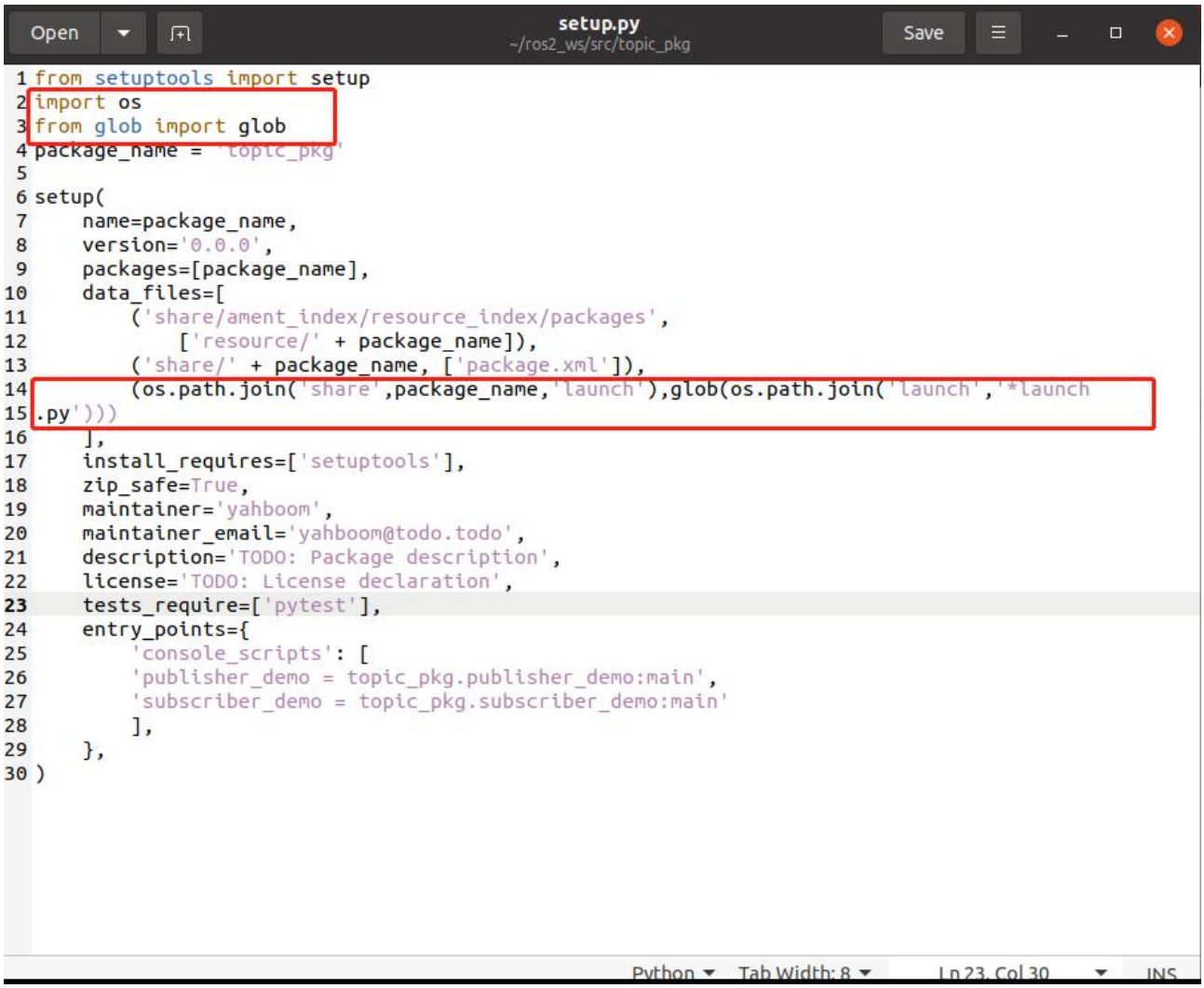


launch 文件命名常以 LaunchName_launch.py，其中，LaunchName 自定义，_launch.py 是常认为固定的。需要修改功能包下的 setup.py 文件，修改内容为添加 launch 路径下的文件，编译才能生成执行的.py 文件，

```
#1、导入相关的头文件
import os
from glob import glob
```

#2、在 data_files 的列表中，加上 launch 路径以及路径下的 launch.py 文件

(os.path.join('share',package_name,'launch'),glob(os.path.join('launch','*launch.py')))



2、编写单个 Node 节点的 launch

终端输入，

```
cd ~/ros2_ws/src/topic_pkg/launch
```

```
gedit single_node_launch.py
```

把以下内容复制到该文件中，

```
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    turtle_node = Node(
                package='turtlesim',
                executable='turtlesim_node',
                    )
    launch_description = LaunchDescription([turtle_node])
    return launch_description
```

2.1、编译工作空间

终端输入，

```
cd ~/ros2_ws
colcon build
```

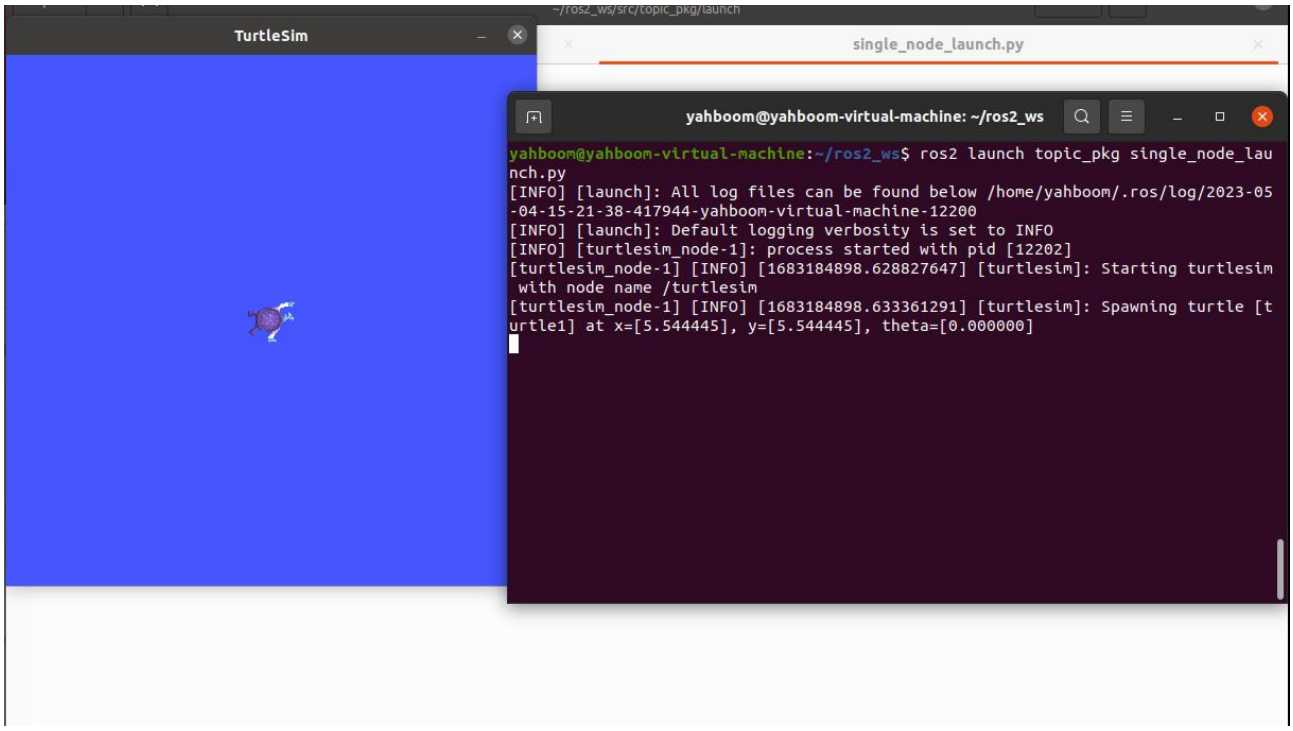
编译完成后，刷新下工作空间的环境变量，

```
source ~/ros2_ws/install/setup.bash
```

2.2、运行程序

终端输入，

```
ros2 launch topic_pkg single_node_launch.py
```



程序运行后，会在运行小海龟的节点。

2.3、源码分析

1）、导入相关库

```
from launch import LaunchDescription
```

```
from launch_ros.actions import Node
```

2）、定义一个函数 generate_launch_description，并且返回一个 launch_description

```

def generate_launch_description():
    turtle_node = Node(
                package='turtlesim',
                executable='turtlesim_node',
                    )
    launch_description = LaunchDescription([turtle_node])
    return launch_description
```

定义了一个变量 turtle_node 作为一个节点启动的返回值，调用 Node 函数，启动重要的两个参数，package 和 executable。

- package：表示功能包，代表功能包的名字。
-
-
- executable：表示执行的程序，可执行程序的名字。
-

然后定义一个变量 launch_description 作为调用 LaunchDescription 函数执行后的返回值，有多个节点启动的话，添加在后边即可。

```
launch_description = LaunchDescription([turtle_node])
```

最后，return launch_description。

3、编写多个 Node 节点的 launch

终端输入，

```
cd ~/ros2_ws/src/topic_pkg/launch
gedit multi_node_launch.py
```

把以下内容复制到该文件中，

```
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    pub_node = Node(
                package='topic_pkg',
                executable='publisher_demo',
                output='screen'
```

```
                )
    sub_node = Node(
                package='topic_pkg',
                executable='subscriber_demo',
                output='screen'
                )
    launch_description = LaunchDescription([pub_node,sub_node])
    return launch_description
```

3.1、编译工作空间

终端输入，

```
cd ~/ros2_ws
colcon build
```

编译完成后，刷新下工作空间的环境变量，

```
source ~/ros2_ws/install/setup.bash
```

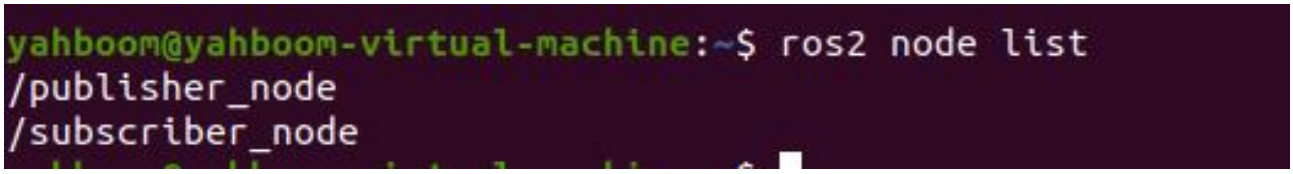
3.2、运行程序

终端输入，

```
ros2 launch topic_pkg multi_node_launch.py
```

终端没有打印内容，我们可以查看哪些节点启动 来验证是否有启动成功，终端输入，

```
ros2 node list
```



由上图可知，启动了两个节点，刚好对应 launch 文件中的两个程序。

3.3、源码解析

大致与 simple_node_launch.py 差不多，不过是多了一个节点，以及在 launch_description = LaunchDescription([pub_node,sub_node])中，多加了一个节点。

4、launch 文件中的话题名称映射

终端输入，

```
cd ~/ros2_ws/src/topic_pkg/launch
gedit remap_name_launch.py
```

把以下内容复制到该文件中，

```
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    turtle_node = Node(
                package='turtlesim',
                executable='turtlesim_node',
                remappings=[("/turtle1/cmd_vel", "/cmd_vel")]
                    )
    launch_description = LaunchDescription([turtle_node])
    return launch_description
```

4.1、编译工作空间

终端输入，

```
cd ~/ros2_ws
colcon build
```

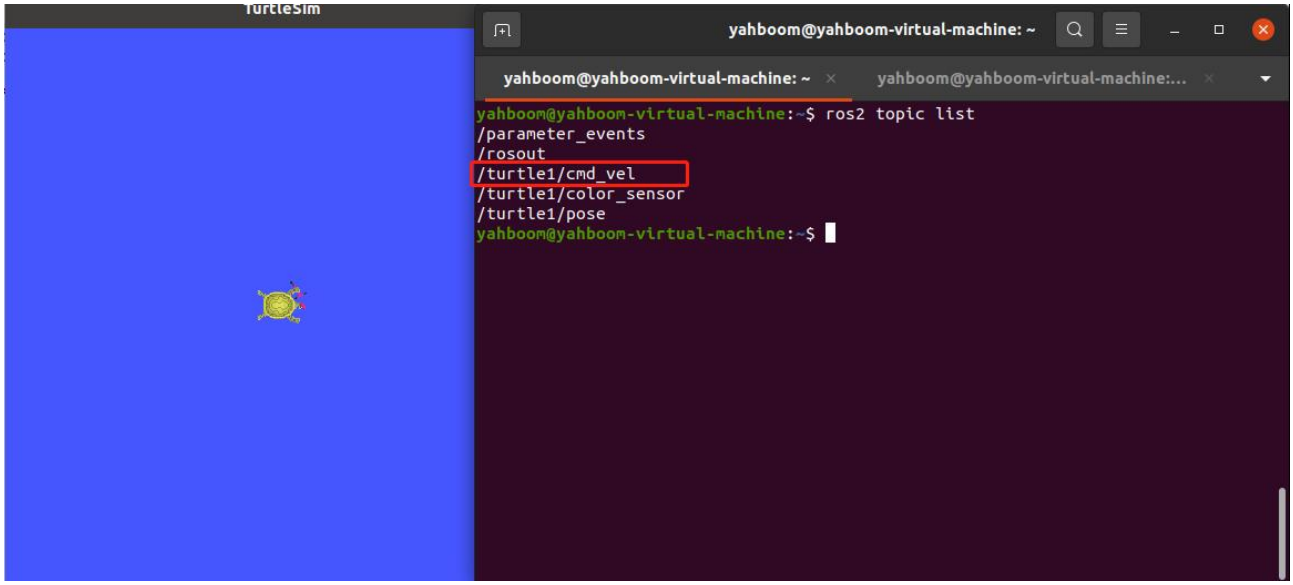
编译完成后，刷新下工作空间的环境变量，

```
source ~/ros2_ws/install/setup.bash
```

4.2、运行程序

我们先看看没有重映射话题前，小海龟的速度话题名称是什么，终端输入，

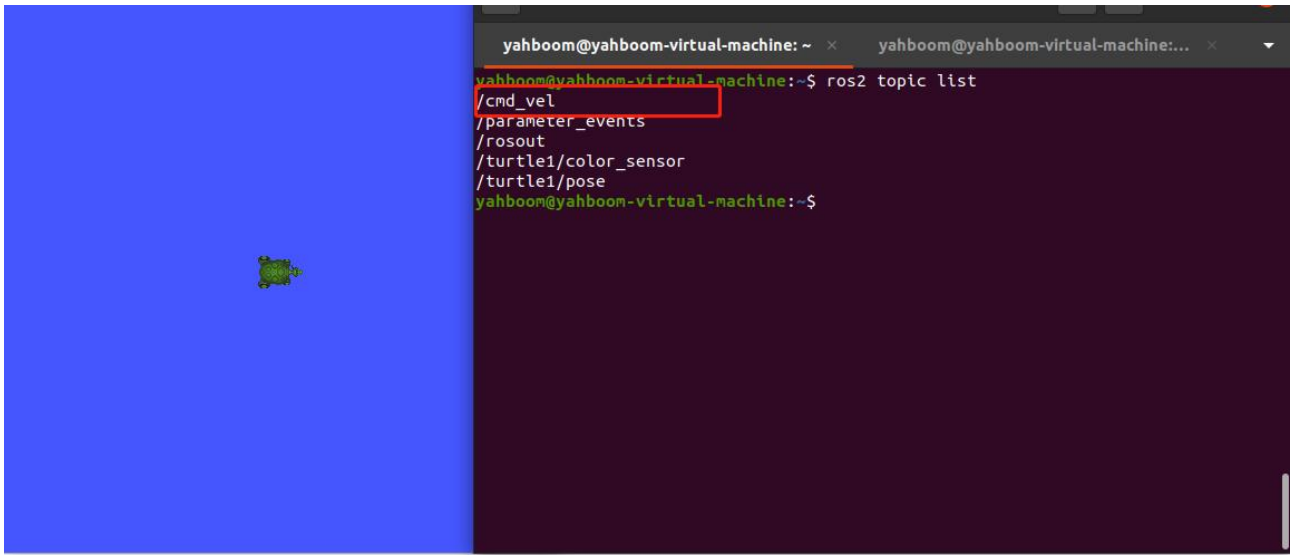
```
ros2 launch topic_pkg single_node_launch.py
ros2 topic list
```



这里的话题是/turtle1/cmd_vel。

再运行重映射话题后的程序，看看小海龟订阅的速度话题名称是什么，终端输入，

```
ros2 launch topic_pkg remap_name_launch.py
ros2 topic list
```



由上图可知，重映射了速度话题名称，映射后的小海龟速度话题名称是/cmd_vel。

4.3、源码分析

在原来的 single_node_launch.py 基础上做了修改，主要是加了以下部分，

```
remappings=[("/turtle1/cmd_vel", "/cmd_vel")]
```

这里就是把原来的/turtle1/cmd_vel 重映射成/cmd_vel。前边的原本的话题名称，后边是我们想要修改成的话题名称。

5、launch 文件启动 launch 文件

终端输入，

```
cd ~/ros2_ws/src/topic_pkg/launch
```

gedit include_launch.py

把以下内容复制到该文件中，

```
import os
from ament_index_python.packages import
get_package_share_directory
from launch import LaunchDescription
from launch.actions import IncludeLaunchDescription
from launch.launch_description_sources import
PythonLaunchDescriptionSource
def generate_launch_description():
    node1 = IncludeLaunchDescription(
    PythonLaunchDescriptionSource([os.path.join(
    get_package_share_directory('topic_pkg'), 'launch'),
    '/multi_node_launch.py'])
    )
    node2 = IncludeLaunchDescription(
    PythonLaunchDescriptionSource([os.path.join(
    get_package_share_directory('topic_pkg'), 'launch'),
    '/single_node_launch.py'])
    )
```

```
    return LaunchDescription([node1,node2])
```

5.1、编译工作空间

终端输入，

```
cd ~/ros2_ws
colcon build
```

编译完成后，刷新下工作空间的环境变量，

```
source ~/ros2_ws/install/setup.bash
```

5.2、运行程序

终端输入，

```
ros2 launch topic_pkg include_launch.py
```

这个 launch 文件会包含启动两个 launch 文件，simple_node_launch.py 和 multi_node_launch.py。可以通过以下命令查看是否启动了这些 launch 文件的节点，终端输入，

```
ros2 node list
```

确实是启动了三个节点，所以是启动成功的。

5.3、源码分析

```
#导入必要的库文件
from ament_index_python.packages import
get_package_share_directory
from launch import LaunchDescription
from launch.actions import IncludeLaunchDescription
from launch.launch_description_sources import
PythonLaunchDescriptionSource
node1 = IncludeLaunchDescription(
    PythonLaunchDescriptionSource([os.path.join(
    get_package_share_directory('topic_pkg'), 'launch'),
    '/multi_node_launch.py'])
    )
```

- os.path.join(get_package_share_directory('topic_pkg')：获取功能包的位置，其中的'topic_pkg'为功能包的名字；

-
-

  launch')：表示存放功能包下存放 launch 文件的文件夹；

-
-

  /multi_node_launch.py'：表示该功能包文件夹下的 launch 文件下的 launch 文件名字也就是示例中/multi_node_launch.py。

-

  6、launch 文件参数配置 rosparam

  终端输入，

  ```
  cd ~/ros2_ws/src/topic_pkg/launch
  gedit param_launch.py
  ```

  把以下内容复制到该文件中，

  ```
  from launch import LaunchDescription
  from launch.actions import DeclareLaunchArgument
  from launch.substitutions import LaunchConfiguration, TextSubstitution
  ```

```
from launch_ros.actions import Node


def generate_launch_description():
    background_r_launch_arg = DeclareLaunchArgument(
     'background_r', default_value=TextSubstitution(text='0'))
    background_g_launch_arg = DeclareLaunchArgument(
     'background_g', default_value=TextSubstitution(text='225'))
    background_b_launch_arg = DeclareLaunchArgument(
     'background_b', default_value=TextSubstitution(text='0'))
    return LaunchDescription([
                                background_r_launch_arg,
                                background_g_launch_arg,
                                background_b_launch_arg,
                                Node(
                                        package='turtlesim',
                                        executable='turtlesim_node',
                                        name='sim',
                                        parameters=[{
                                                'background_r':Launch
Configuration('background_r'),
                                                'background_g':Launch
Configuration('background_g'),
```

```
                                                'background_b':LaunchConfiguration('background_b'),
                                            }]
                                        )
                                    ])
```

6.1、编译工作空间

终端输入，

```
cd ~/ros2_ws
colcon build
```

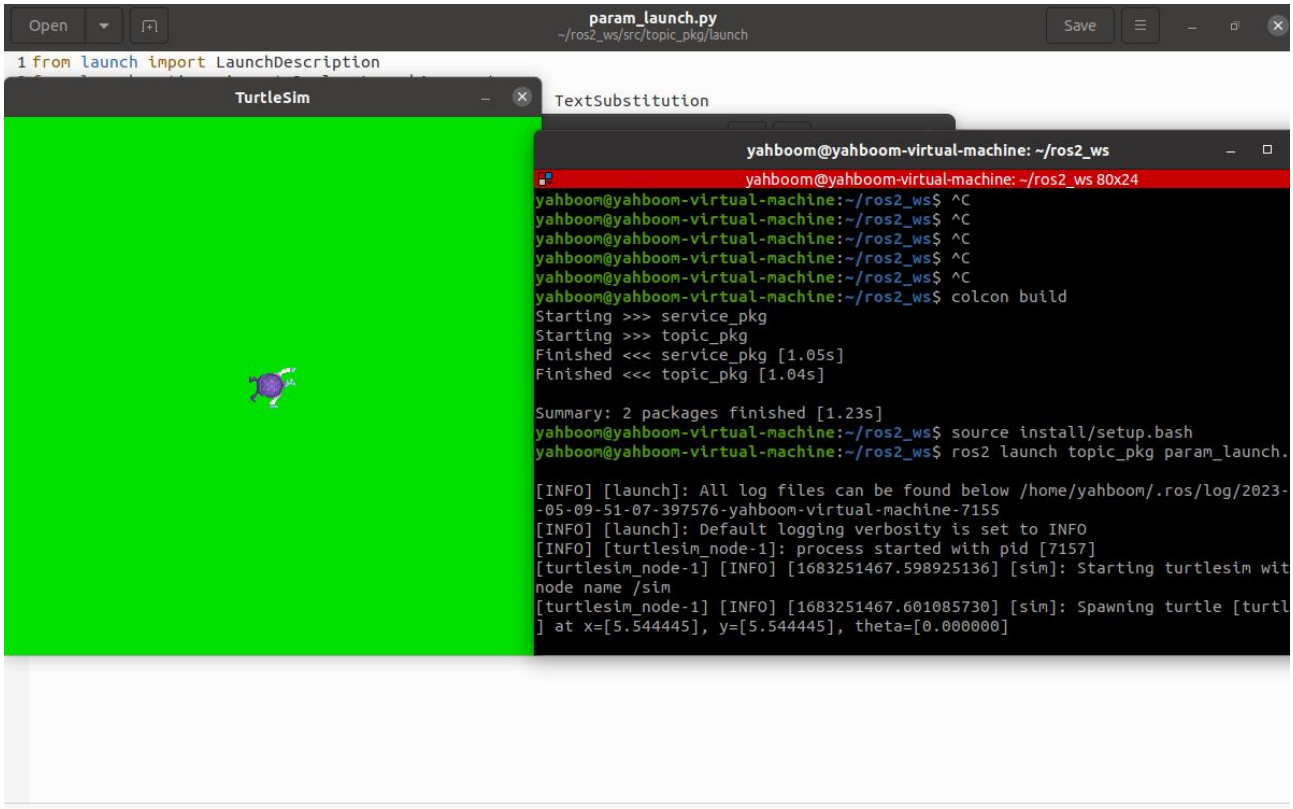
编译完成后，刷新下工作空间的环境变量，

```
source ~/ros2_ws/install/setup.bash
```

6.2、运行程序

终端输入，

```
ros2 launch topic_pkg param_launch.py
```



程序运行后，会加载设置好的参数，修改了默认的 RGB 参数，改变了背景板的颜色。

6.3、源码解析

```
from launch.actions import DeclareLaunchArgument      # 声明 launch 文件内使用的 Argument 类
background_r_launch_arg = DeclareLaunchArgument(
    'background_r', default_value=TextSubstitution(text='0'))  # 创建一个 Launch 文件内参数 background_r
    'background_r', default_value=TextSubstitution(text='0'))  # 创建一个 Launch 文件内参数 background_g
```

```
        'background_r', default_value=TextSubstitution(text='0'))  # 创建一个 Launch 文件内参数 background_b
        background_r_launch_arg,  # 调用以上创建的参数
        background_g_launch_arg,
        background_b_launch_arg,
            parameters=[{                                   # ROS 参数列表
                'background_r': LaunchConfiguration('background_r'),  # 创建参数 background_r
                'background_g': LaunchConfiguration('background_g'),  # 创建参数 background_g
                'background_b': LaunchConfiguration('background_b'),  # 创建参数 background_b
```

argument 与 param 都是参数的意思，但是 argument 是在 launch 文件里边传递的，而 param 是在节点程序内传递的参数。

7、launch 文件加载参数配置表

首先新建一个参数表，终端输入，

```
cd ~/ros2_ws/src/topic_pkg
mkdir config
cd config
```

```
gedit turtle_config.yaml
```

把一下内容复制到 turtle_config.yaml 文件中，

```
sim:
    ros__parameters:
        background_r: 0
        background_g: 0
        background_b: 7
```

保存后退出，然后修改 setup.py 文件，加载参数文件的路径，终端输入，

```
cd ~/ros2_ws/src/topic_pkg
gedit setup.py
```

在以上图片所示位置，添加以下内容，

```
(os.path.join('share', package_name, 'config'),
glob(os.path.join('config', '*.yaml'))),
```

保存后退出，最后编写 launch 文件，终端输入，

```
cd ~/ros2_ws/src/topic_pkg/launch
gedit param_config_launch.py
```

把以下内容复制到该文件中，

```
import os
```

```
from launch import LaunchDescription
from launch_ros.actions import Node
from ament_index_python.packages import
get_package_share_directory
def generate_launch_description():
    config = os.path.join(
     get_package_share_directory('topic_pkg'),
     'config',
     'turtle_config.yaml'
     )
    return LaunchDescription([
                                Node(
                                    package='turtlesim',
                                    executable='turtlesim_node',
                                    name='sim',
                                    parameters=[config]
                                )
                                ])
```

7.1、编译工作空间

终端输入，

```
cd ~/ros2_ws
colcon build
```

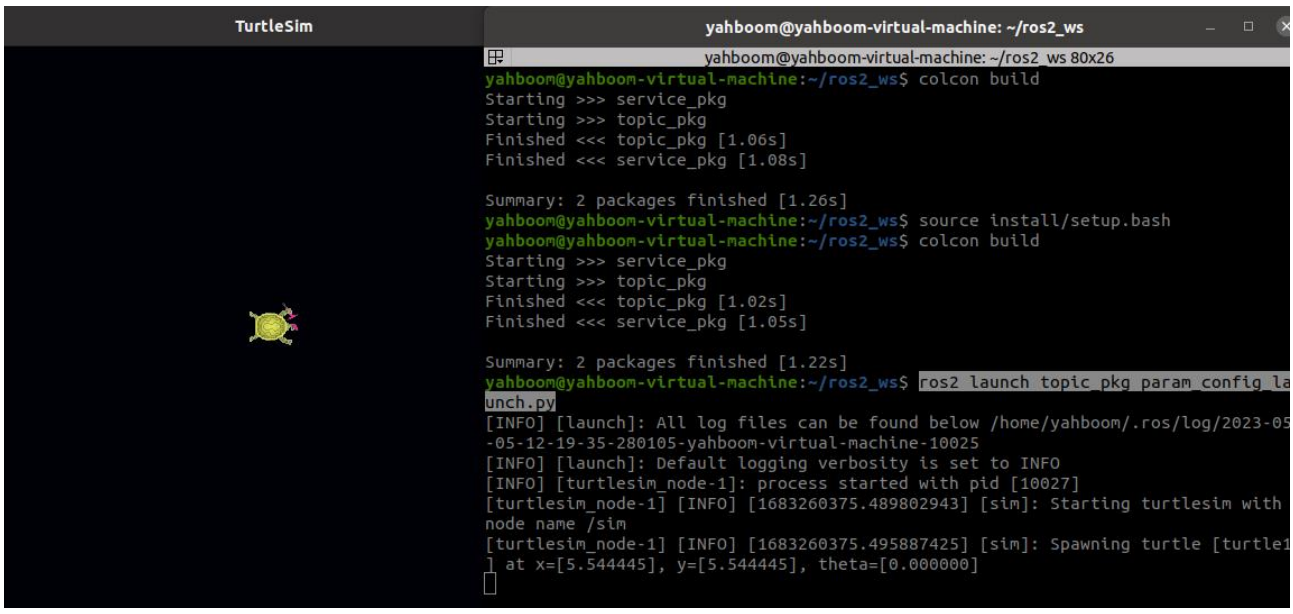
编译完成后，刷新下工作空间的环境变量，

```
source ~/ros2_ws/install/setup.bash
```

7.2、运行程序

终端输入，

```
ros2 launch topic_pkg param_config_launch.py
```



运行程序可以得到小海龟且背景板颜色通过参数设置成黑色。

7.3、源码解析

```
#找到参数文件位置
config = os.path.join(
    get_package_share_directory('topic_pkg'),
    'config',
    'turtle_config.yaml'
    )
#加载参数文件
parameters=[config]
```

我们看下参数文件 turtle_config.yaml，

```
sim:
  ros__parameters:
    background_r: 0
    background_g: 0
    background_b: 7
```

参数文件位置在：~/ros2_ws/src/topic_pkg/config

- sim：表示节点名称
-

- ros__parameters：表示 ros 参数，这里是固定的
-
-
  background_r 参数名称，后边跟的值就是参数设定的值
-

5. ROS2 视觉图像处理

1、opencv 基础课程

1、opencv 基础课程

1.1、读取图片与展示

1.1.1、图像读入

img = cv2.imread('yahboom.jpg', 0) 第一个参数是图片的路径，第二个参数是如何读取这幅图片。

cv2.IMREAD_UNCHANGED：保持原格式不变，-1；

cv2.IMREAD_GRAYSCALE：以灰度模式读入图片，可以用 0 表示；

cv2.IMREAD_COLOR：，读入一副彩色图片，可以用 1 表示；

cv2.IMREAD_UNCHANGED：读入一幅图片，并包括其 alpha 通道，可以用 2 表示。

1.1.2、图像展示

cv.imshow('frame', frame)：打开一个窗口名为 frame，并且显示 frame 帧数据（图像/视频数据）

参数含义：

第一个参数表示创建打开的窗口的名字；

第二个参数表示需要显示的图片。

1.1.3、代码与实际效果展示

运行程序，

```
cd /root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_astra/scripts/opencv/
python3 1_1.py
```

```
import cv2 as cv
if __name__ == '__main__':
    img = cv.imread('yahboom.jpg')
```

```
    while True :
    cv.imshow("frame",img)
    action = cv.waitKey(10) & 0xFF
    if action == ord('q') or action == 113:
        break
    img.release()
    cv.destroyAllWindows()
```

1.2、opencv 图片写入

1.2.1、函数方法：cv2.imwrite('new_img_name', img)

参数含义：

第一个参数是保存的文件名，第二个参数是保存的图像。

1.2.2、代码与实际效果展示

运行程序，

```
cd /root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_astra/scripts/opencv/
python3 1_2.py
```

```
import cv2 as cv
if __name__ == '__main__':
    img = cv.imread('yahboom.jpg')
    cv.imwrite("yahboom_new.jpg",img) #新建文件 yahboom_new.jpg，并且把 yahboom.jpg 写进去
    new_img = cv.imread('yahboom_new.jpg') #读取新写入的图片
    while True :
        cv.imshow("frame",img)
        cv.imshow("new_frame",new_img)
    action = cv.waitKey(10) & 0xFF
    if action == ord('q') or action == 113:
        break
    img.release()
    cv.destroyAllWindows()
```

```python
import cv2 as cv

if __name__ == '__main__':
    img = cv.imread('yahboom.jpg')
    cv.imwrite("yahboom_new.jpg",img)
    new_img = cv.imread('yahboom_new.jpg')
    while True :
        cv.imshow("frame",img)
        cv.imshow("new_frame",new_img)
        action = cv.waitKey(10) & 0xFF
        if action == ord('q') or action == 113:
            break
    img.release()
    cv.destroyAllWindows()
```

1.3、opencv 摄像头读取与显示视频

1.3.1、摄像头读取

capture=cv.VideoCapture(0)

参数含义：

VideoCapture()中参数是 0，表示打开笔记本的内置摄像头，参数是视频文件路径则打开视频，如 cap =

cv2.VideoCapture(“../test.avi”)

1.3.2、显示摄像头视频

ret,img = frame.read()

返回值含义：

ret：ret 是一个 bool 值，判断是否读回正确的帧

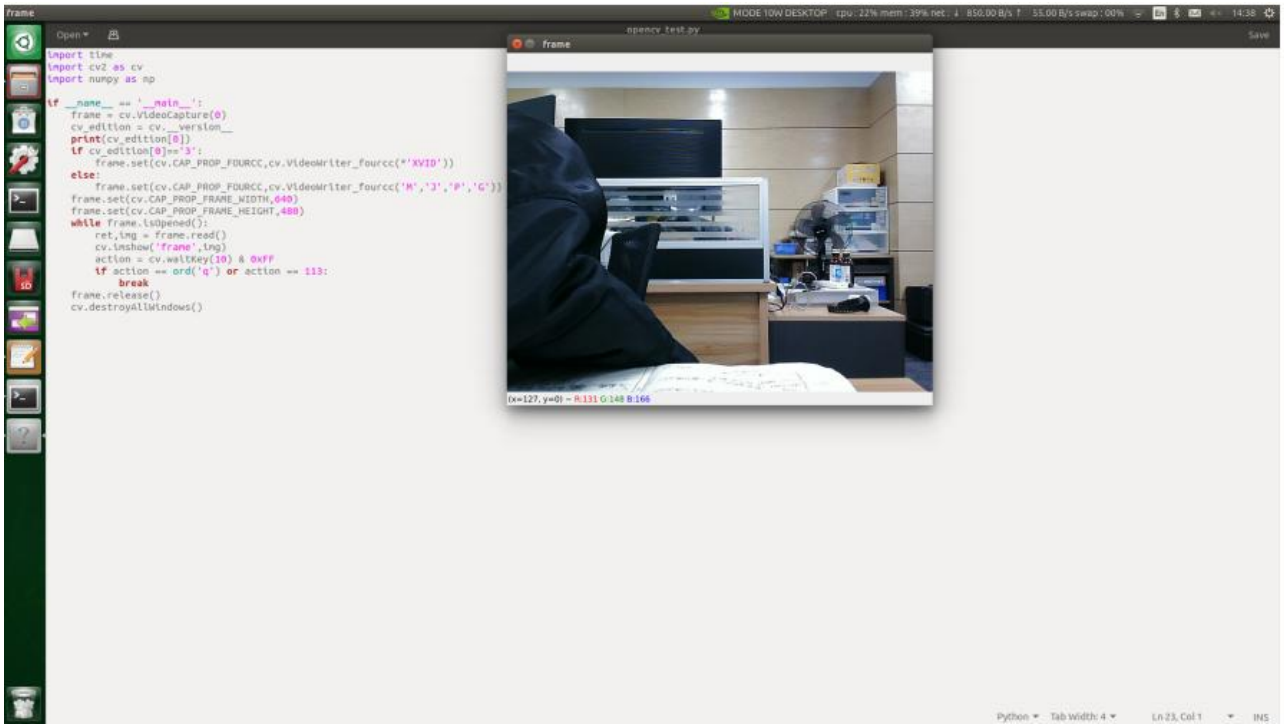
img：每一帧的图像数据

1.3.3、代码与实际效果展示

运行程序，

```
cd /root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_astra/scripts/opencv/
python3 1_3.py
```

```
import cv2 as cv
if __name__ == '__main__':
    frame = cv.VideoCapture(0)
    while frame.isOpened():
        ret,img = frame.read()
        cv.imshow('frame',img)
        action = cv.waitKey(10) & 0xFF
        if action == ord('q') or action == 113:
        break
    frame.release()
    cv.destroyAllWindows()
```



1.4、openc 像素操作

1.4.1、像素操作，我们可以给任意位置改成新的像素颜色。

首先，我们先要读取图像，然后修改 bgr 的数值，赋值一个区域为黑色。

1.4.2、代码与实际效果展示

运行程序

```
cd /root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_astra/scripts/opencv/
python3 1_4.py
```

```
import cv2
if __name__ == '__main__':
    img = cv2.imread('yahboom.jpg')
    (b,g,r) = img[100,100]
    print(b,g,r)
    i=j=0
    for j in range(1,255):
    img[i,j] = (0,0,0)
    for j in range(1,255):
    img[i,j] = (0,0,0)
    while True :
        cv2.imshow("frame",img)
        action = cv2.waitKey(10) & 0xFF
        if action == ord('q') or action == 113:
            break
    img.release()
    cv2.destroyAllWindows()
```

```python
import cv2

if __name__ == '__main__':
    img = cv2.imread('yahboom.jpg')
    (b,g,r) = img[100,100]
    print(b,g,r)
    i=j=0
    for j in range(1,255):
        img[i,j] = (0,0,0)
        for j in range(1,255):
            img[i,j] = (0,0,0)
    while True :
        cv2.imshow("frame",img)
        action = cv2.waitKey(10) & 0xFF
        if action == ord('q') or action == 1
            break
    img.release()
    cv2.destroyAllWindows()
```

红色框部分就是修改的色素值。

2.1、opencv 图片缩放

2.1.1、cv2.resize(InputArray src,OutputArray dst, Size, fx, fy, interpolation)

参数含义：

InputArray src：输入图片

OutputArray ds：输出图片

Size：输出图片尺寸

fx,fy：沿 x 轴，y 轴的缩放系数

interpolation：插入方式，可选择 INTER_NEAREST（最近邻插值），INTER_LINEAR（双线性插值（默

认设置））， INTER_AREA（使用像素区域关系进行重采样）， INTER_CUBIC（4x4 像素邻域的双三次插

值）， INTER_LANCZOS4（8x8 像素邻域的 Lanczos 插值）

需要注意：

输出尺寸格式为（宽，高）

默认的插值方法为：双线性插值

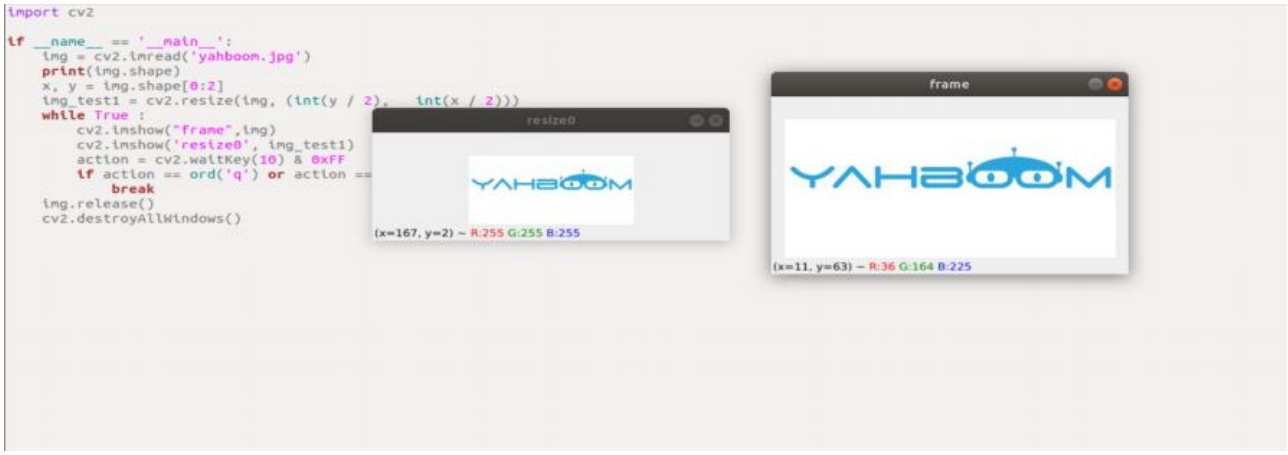
2.1.2、代码与实际效果展示

运行程序

```
cd /root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_astra/scripts/opencv/
python3 2_1.py
```

```
import cv2
if __name__ == '__main__':
    img = cv2.imread('yahboom.jpg')
    print(img.shape)
    x, y = img.shape[0:2]
    img_test1 = cv2.resize(img, (int(y / 2), int(x / 2)))
```

```
    while True :
        cv2.imshow("frame",img)
        cv2.imshow('resize0', img_test1)
        action = cv2.waitKey(10) & 0xFF
        if action == ord('q') or action == 113:
            break
    img.release()
    cv2.destroyAllWindows()
```



2.2、opencv 图片剪裁

2.2.1、图片剪切

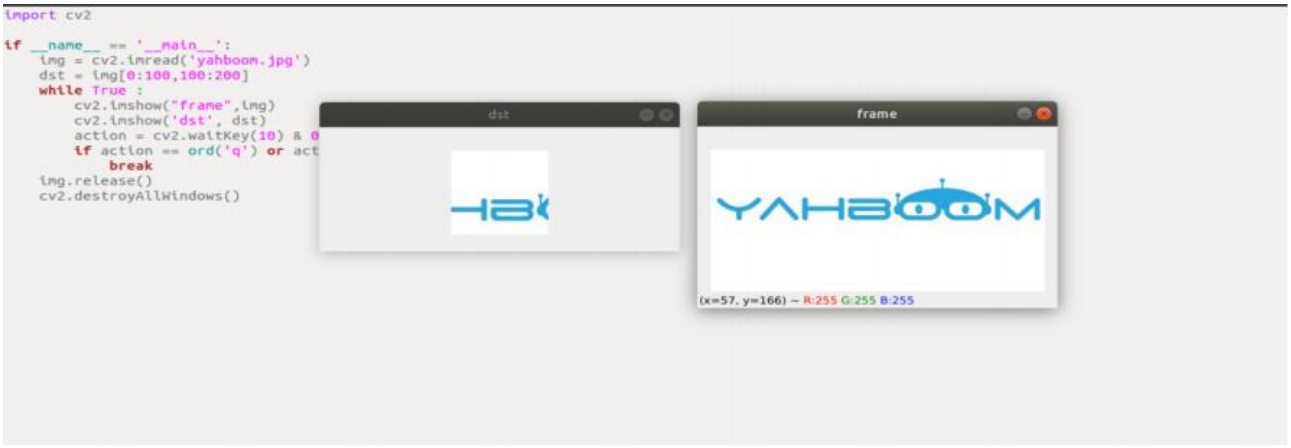
首先读取图像，然后再数组中获取像素点区域。下面代码中选取形区域 X：300-500 Y：500-700，注意图像尺寸是 800*800，所以选择的区域不要超过此分辨率。

2.2.2、代码与实际效果展示

运行程序

```
cd /root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_astra/scripts/opencv/
python3 2_2.py
```

```
import cv2
if __name__ == '__main__':
    img = cv2.imread('yahboom.jpg')
    dst = img[0:100,100:200]
    while True :
        cv2.imshow("frame",img)
        cv2.imshow('dst', dst)
        action = cv2.waitKey(10) & 0xFF
        if action == ord('q') or action == 113:
            break
        img.release()
        cv2.destroyAllWindows()
```



2.3、opencv 图片平移

2.3.1、cv2.warpAffine(src, M, dsize[,dst[, flags[, borderMode[, borderValue]]]])

参数含义：

- src：输入图像
- M：变换矩阵
- dsize：输出图像的大小
- flags：插值方法的组合（int 类型！）
- borderMode：边界像素模式（int 类型！）
- borderValue：（重点！）边界填充值; 默认情况下，它为 0

上述参数中：M 作为仿射变换矩阵，一般反映平移或旋转的关系，为 InputArray 类型的 2×3 的变换矩阵。

在日常进行仿射变换是，只设置前三个参数的情况下，如 cv2.warpAffine(img,M,(rows,cols))就可以实现

基本的仿射变换效果。

2.3.2、如何得到转换矩阵 M？下列举个例子说明，

通过转换矩阵 M 实现将原始图像 src 转换为目标图像 dst：

dst(x, y) = src(M11x + M12y+M13, M21x+M22y+M23)

将原始图像 src 向右侧移动 200、向下移动 100 个像素，则其对应关系为：

dst(x, y) = src(x+200, y+100)

将上述表达式补充完整，即：

dst(x, y) = src(1·x + 0·y + 200, 0·x + 1·y + 100)

根据上述表达式，可以确定对应的转换矩阵 M 中各个元素的值为：

M11=1

M12=0

M13=200

M21=0

M22=1

M23=100

将上述值代入转换矩阵 M，得到：

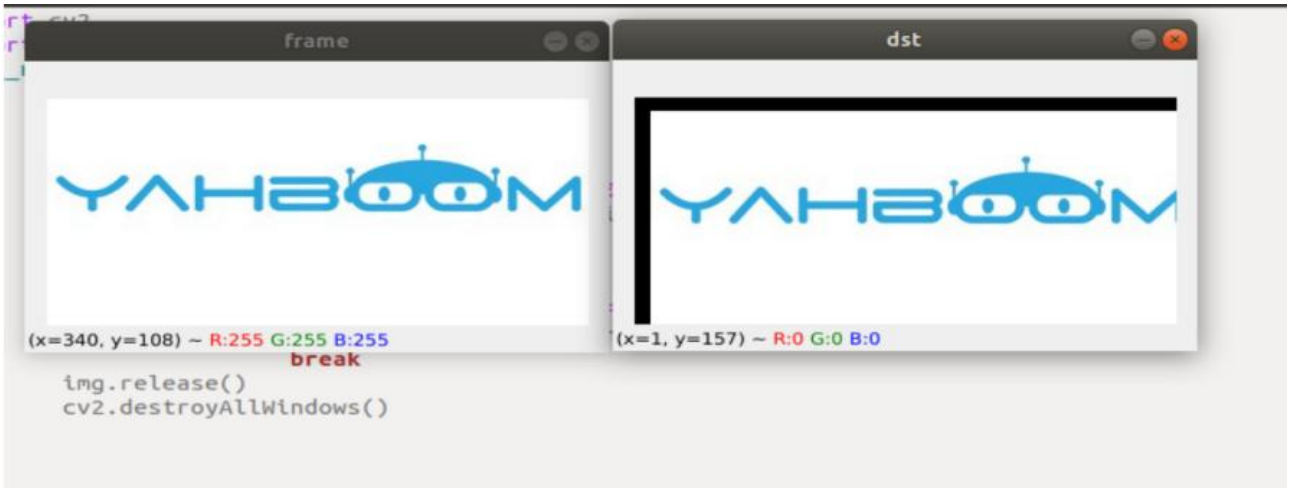
M = [ ]

2.3.3、代码与实际效果展示

```
cd /root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_astra/scripts/opencv/
python3 2_3.py
```

```
import cv2
import numpy as np
if __name__ == '__main__':
    img = cv2.imread('yahboom.jpg')
    imgInfo = img.shape
    height = imgInfo[0]
    width = imgInfo[1]
    matShift = np.float32([[1,0,10],[0,1,10]])# 2*3
    dst = cv2.warpAffine(img, matShift, (width,height))
    while True :
        cv2.imshow("frame",img)
        cv2.imshow('dst', dst)
        action = cv2.waitKey(10) & 0xFF
        if action == ord('q') or action == 113:
            break
```

```
    img.release()
    cv2.destroyAllWindows()
```



2.4、opencv 图片镜像

2.4.1、图片镜像的原理

图像的镜像变换分为两种：水平镜像和垂直镜像。水平镜像以图像垂直中线为轴，将图像的像素进行对换，也就是将图像的左半部和右半部对调。垂直镜像则是以图像的水平中线为轴，将图像的上半部分和下半部分对调。

变换原理：

设图像的宽度为 width，长度为 height。(x,y)为变换后的坐标，(x0,y0)为原图像的坐标，

-

  水平镜像变换

-

向前映射：x=width-x0-1,y=y0

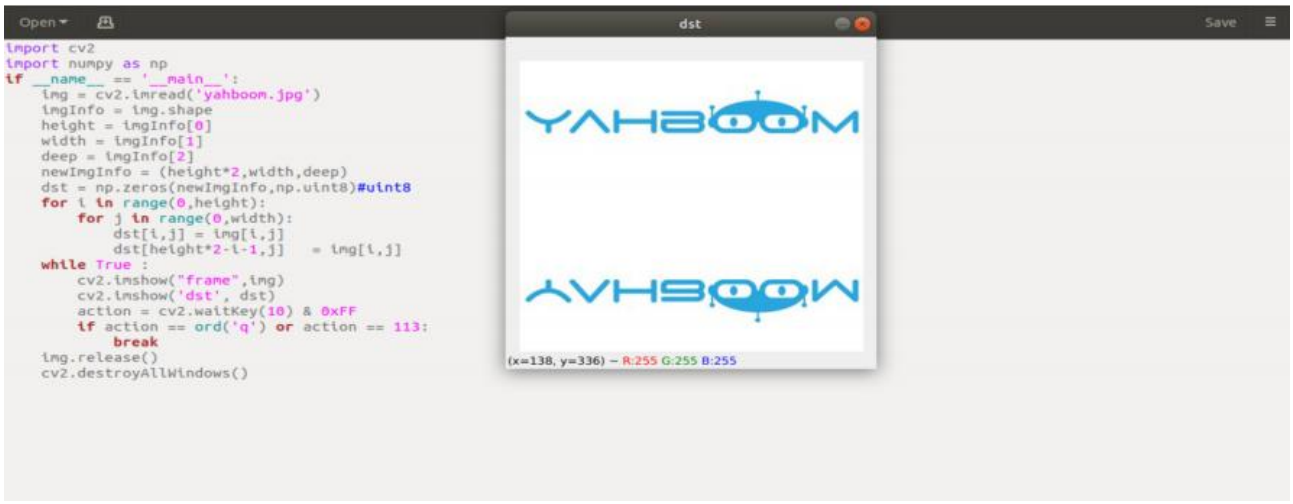
- 向后映射：x0=width-x-1,y0=y
-
- 垂直镜像变换
- 向上映射：x=x0,y=height-y0-1
- 向下映射：x0=x，y0=height-y-1
- 总结：在水平镜像变换时，遍历了整个图像，然后根据映射关系对每个像素都做了处理。实际上，水平镜像变换就是将图像坐标的列换到右边，右边的列换到左边，是可以以列为单位做变换的。同样垂直镜像变换也如此，可以以行为单位进行变换。

2.4.2、以垂直变换为例，看看 Python 如何实现

```
cd
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_astra/scripts/opencv/
python3 2_4.py
```

```
import cv2
import numpy as np
if __name__ == '__main__':
    img = cv2.imread('yahboom.jpg')
    imgInfo = img.shape
    height = imgInfo[0]
    width = imgInfo[1]
    deep = imgInfo[2]
    newImgInfo = (height*2,width,deep)
    dst = np.zeros(newImgInfo,np.uint8)#uint8
    for i in range(0,height):
        for j in range(0,width):
            dst[i,j] = img[i,j]
            dst[height*2-i-1,j] = img[i,j]
    while True :
        cv2.imshow("frame",img)
        cv2.imshow('dst', dst)
```

```
        action = cv2.waitKey(10) & 0xFF
        if action == ord('q') or action == 113:
            break
    img.release()
    cv2.destroyAllWindows()
```



3.1、opencv 图像灰度化处理

3.1.1、图像灰度化

彩色图像转化为灰度图像的过程是图像的灰度化处理。彩色图像中的每个像素的颜色由 R，G，B 三个分量决定，而每个分量中可取值 0-255，这样一个像素点可以有 1600 多万（256256256=1677256）的颜色的变化范围。而灰度图像是 R，G，B 三个分量相同的一种特殊的彩色图像，其中一个像素点的变化范围为 256 种，所以在数字图像处理中一般将各种格式的图像转化为灰度图像以使后续的图像的计算量少一些。灰度图像的描述与彩色图像一样仍然反映了整副图像的整体和局部的色度和高亮等级的分布和特征。

3.1.2、图像灰度化处理

灰度化处理就是将一幅色彩图像转化为灰度图像的过程。彩色图像分为 R，G，B 三个分量，分别显示出红绿蓝等各种颜色，灰度化就是使彩色的 R，G，B 分量相等的过程。灰度值大的像素点比较亮（像素值最大为 255，为白色），反之比较暗（像素最下为 0，为黑色）。图像灰度化核心思想是 R = G = B ，这个值也叫灰度值。

1)最大值法：使转化后的 R，G，B 得值等于转化前 3 个值中最大的一个，即：R=G=B=max（R，G，B）。这种方法转换的灰度图亮度很高。

2)平均值法：是转化后 R，G，B 的值为转化前 R,G,B 的平均值。即：R=G=B=(R+G+B)/3。这种方法产生的灰度图像比较柔和。

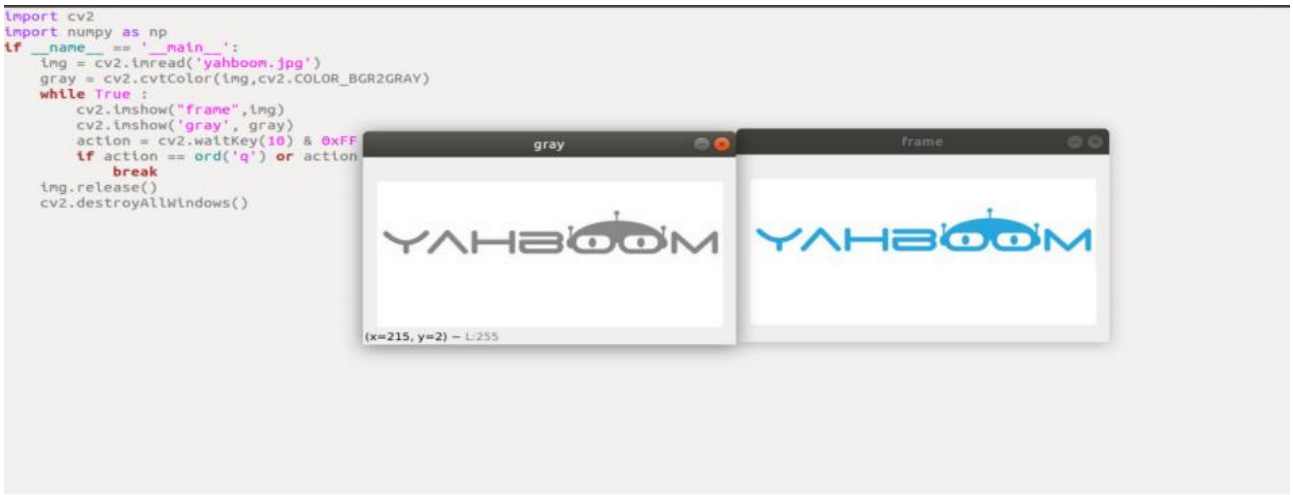
在 OpenCV 中，用 cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)来实现对图像进行灰度化处理

3.1.3、代码与实际效果展示

运行程序，

```
cd /root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_astra/scripts/opencv/
python3 3_1.py
```

```
import cv2
import numpy as np
if __name__ == '__main__':
    img = cv2.imread('yahboom.jpg')
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    while True :
        cv2.imshow("frame",img)
        cv2.imshow('gray', gray)
        action = cv2.waitKey(10) & 0xFF
        if action == ord('q') or action == 113:
            break
        img.release()
        cv2.destroyAllWindows()
```



3.2、opencv 图像二值化处理

3.2.1、二值化核心思想

设阈值，大于阈值的为 0（黑色）或 255（白色），使图像称为黑白图。阈值可固定，也可以自适应阈值。自适应阈值一般为一点像素与这点为中序的区域像素平均值或者高斯分布加权和的比较，其中可以设置一个差值也可以不设置。

3.2.2、cv2.threshold（src, threshold, maxValue,thresholdType）

参数含义：

- src：原图像
-
-
- threshold：当前阈值
-
-
- maxVal：最大阈值，一般为 255
-
-
- thresholdType：阈值类型，一般有下面几个值，

-
  - THRESH_BINARY = 0，#大于阈值的像素点的灰度值设定为 maxValue(如 8 位灰度值最大为 255)，灰度值小于阈值的像素点的灰度值设定为 0。
  - THRESH_BINARY_INV = 1，#大于阈值的像素点的灰度值设定为 0，而小于该阈值的设定为 maxValue。
  - THRESH_TRUNC = 2，#大于阈值的像素点的灰度值设定为 0，而小于该阈值的设定为 maxValue。
  - THRESH_TOZERO = 3，#像素点的灰度值小于该阈值的不进行任何改变，而大于该阈值的部分，其灰度值全部变为 0。
  - THRESH_TOZERO_INV = 4 #像素点的灰度值大于该阈值的不进行任何改变，像素点的灰度值小于该阈值的，其灰度值全部变为 0。
-

返回值：

-
  - retval：与参数 thresh 一致 3
  - dst： 结果图像

注意：在进行二值化之前，我们需要把彩色图进行灰度化处理，得到灰度图。

3.2.3、代码与实际效果展示

运行程序

```
cd /root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_astra/scripts/opencv/
python3 3_2.py
```

```
import cv2
import numpy as np
if __name__ == '__main__':
    img = cv2.imread('yahboom.jpg')
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

ret,thresh1=cv2.threshold(gray,180,255,cv2.THRESH_BINARY_INV)
    while True :
        cv2.imshow("frame",img)
        cv2.imshow('gray', gray)
        cv2.imshow("binary",thresh1)
        action = cv2.waitKey(10) & 0xFF
        if action == ord('q') or action == 113:
            break
    img.release()
    cv2.destroyAllWindows()
```

3.3、opencv 图像边缘检测

3.3.1、图像边缘检测的目的

在保留原有图像属性的情况下，显著减少图像的数据规模。目前有多种算法可以进行边缘检测，虽然 Canny 算法年代久远，但可以说它是边缘检测的一种标准算法，而且仍在研究中广泛使用。

3.3.2、Canny 边缘检测算法

在目前常用的边缘检测方法中，Canny 边缘检测算法是具有严格定义的，可以提供良好可靠检测的方法之一。由于它具有满足边缘检测的三个标准和实现过程简单的优势，成为边缘检测最流行的算法之一。

Canny 边缘检测算法可以分为以下 5 个步骤：

- 使用高斯滤波器，以平滑图像，滤除噪声
- 计算图像中每个像素点的梯度强度和方向

- 应用非极大值（Non-Maximum Suppression）抑制，以消除边缘检测带来的杂散响应
- 应用双阈值（Double-Threshold）检测来确定真实的和潜在的边缘
- 通过抑制孤立的弱边缘最终完成边缘检测

3.3.3、opencv 实现步骤

- 图像灰度化：gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
-
- 高斯过滤（减噪处理）图像：GaussianBlur（src，ksize，sigmaX [，dst [，sigmaY [，borderType]]]）-> dst
-
  - src：输入的图像，一般是灰度后的图片
  - ksize：高斯内核大小
  - sigmaX ：X 方向上的高斯核标准偏差
  - sigmaY ：Y 方向上的高斯核标准差
  - dst：处理后的图像
-

Canny 方法处理得到图像：edges=cv2.Canny(image, threshold1, threshold2[, apertureSize[,L2gradient]])

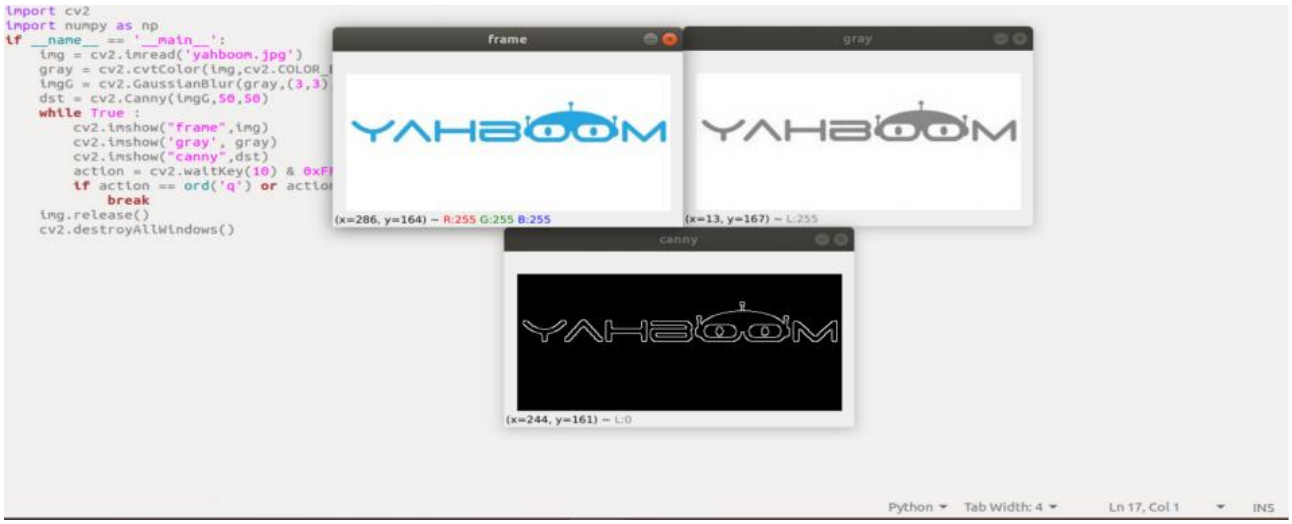
-
  - edges：计算得到的边缘图像
  - image ： 计算得到的边缘图像，一般是高斯处理后得到的图像
  - threshold1 ： 处理过程中的第一个阈值
  - threshold2 ： 处理过程中的第二个阈值
  - apertureSize ： Sobel 算子的孔径大小
  - L2gradient ： 计算图像梯度幅度（gradient magnitude）的标识默认值为 False。如果为 True，则使用更精确的 L2 范数进行计算（即两个方向的导数的平方和再开方），否则使用 L1 范数（直接将两个方向导数的绝对值相加）。

3.3.4、代码和实际效果展示

运行程序，

```
cd /root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_astra/scripts/opencv/
python3 3_3.py
```

```
import cv2
import numpy as np
if __name__ == '__main__':
    img = cv2.imread('yahboom.jpg')
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    imgG = cv2.GaussianBlur(gray,(3,3),0)
    dst = cv2.Canny(imgG,50,50)
    while True :
        cv2.imshow("frame",img)
        cv2.imshow('gray', gray)
        cv2.imshow("canny",dst)
        action = cv2.waitKey(10) & 0xFF
        if action == ord('q') or action == 113:
            break
    img.release()
    cv2.destroyAllWindows()
```



3.4、opencv 线段绘制

3.4.1、cv2.line(dst, pt1, pt2, color, thickness=None, lineType=None, shift=None)

参数含义：

- dst：输出图像
- pt1，pt2：必选参数。线段的坐标点，分别表示起始点和终止点
- color：必选参数。用于设置线段的颜色
- thickness：可选参数。用于设置线段的宽度
- lineType：可选参数。用于设置线段的类型，可选 8（8 邻接连接线-默认）、4（4 邻接连接线）和 cv2.LINE_AA 为抗锯齿

3.4.2、代码与实际效果展示

运行程序，

```
cd
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_astra/scripts/opencv/
python3 3_4.py
```

```
import cv2
import numpy as np
if __name__ == '__main__':
    img = cv2.imread('yahboom.jpg')
    line = cv2.line(img, (50,20), (20,100), (255,0,255), 10)
    while True :
        cv2.imshow("line",line)
        action = cv2.waitKey(10) & 0xFF
        if action == ord('q') or action == 113:
            break
        img.release()
        cv2.destroyAllWindows()
```

3.5、opencv 绘制矩形

3.5.1、cv2.rectangle（img，pt1，pt2，color，thickness=None，lineType=None，shift=None）

参数含义：

- img：画布或者载体图像
- pt1，pt2：必选参数。矩形的顶点，分别表示顶点与对角顶点，即矩形的左上角与右下角（这两个顶点可以确定一个唯一的矩形），可以理解成是对角线。
- color：必选参数。用于设置矩形的颜色
- thickness：可选参数。用于设置矩形边的宽度，当值为负数时，表示对矩形进行填充
- lineType：可选参数。用于设置线段的类型，可选 8（8 邻接连接线-默认）、4（4 邻接连接线）cv2.LINE_AA 为抗锯齿

3.5.2、代码和效果展示

运行程序，

```
cd /root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_astra/scripts/opencv/
python3 3_5.py
```

```
import cv2
import numpy as np
if __name__ == '__main__':
    img = cv2.imread('yahboom.jpg')
    rect = cv2.rectangle(img, (50,20), (100,100), (255,0,255), 10)
    while True :
        cv2.imshow("line",rect)
        action = cv2.waitKey(10) & 0xFF
        if action == ord('q') or action == 113:
            break
    img.release()
    cv2.destroyAllWindows()
```

```python
import cv2
import numpy as np
if __name__ == '__main__':
    img = cv2.imread('yahboom.jpg')
    rect = cv2.rectangle(img, (50,20), (100,100), (255,0,255), 10)
    while True :
        cv2.imshow("line",rect)
        action = cv2.waitKey(10) & 0xFF
        if action == ord('q') or action == 113:
            break
    img.release()
    cv2.destroyAllWindows()
```

3.6、opencv 绘制圆形

3.6.1、cv2.circle(img, center, radius, color[,thickness[,lineType]])

参数含义：

- img:画或者载体图像布
- center：为圆心坐标，格式：　(50,50)
- radius：半径
- thickness：　线条粗细。默认为 1.如果-1 则为填充实心
- lineType：线条类型。默认是 8，连接类型。如下表说明

参数	说明
cv2.FILLED	填充
cv2.LINE_4	4连接类型
cv2.LINE_8	8连接类型
cv2.LINE_AA	抗锯齿，该参数会让线条更平滑

3.6.2、代码和实际效果展示

运行程序，

```
cd /root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_astra/scripts/opencv/
python3 3_6.py
```

```
import cv2
import numpy as np
if __name__ == '__main__':
    img = cv2.imread('yahboom.jpg')
    circle = cv2.circle(img, (80,80), 50, (255,0,255), 10)
    while True :
        cv2.imshow("circle",circle)
```

```
        action = cv2.waitKey(10) & 0xFF
        if action == ord('q') or action == 113:
            break
    img.release()
    cv2.destroyAllWindows()
```

3.7、opencv 绘制椭圆

3.7.1、cv2.ellipse(img, center, axes, angle, StartAngle, endAngle, color[,thickness[,lineType])

参数含义：

- center：椭圆的中心点，（x，y）
- axes：指的是短半径和长半径，（x，y）
- StartAngle：圆弧起始角的角度
- endAngle：圆弧终结角的角度
- img、color、thickness、lineType 可以参考圆的说明

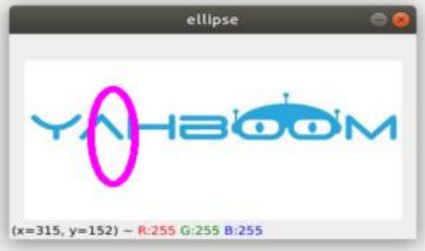
3.7.2、代码和实际效果展示

运行程序，

```
cd /root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_astra/scripts/opencv/
python3 3_7.py
```

```
import cv2
import numpy as np
if __name__ == '__main__':
    img = cv2.imread('yahboom.jpg')
    ellipse = cv2.ellipse(img, (80,80), (20,50),0,0, 360,(255,0,255), 5)
    while True :
        cv2.imshow("ellipse",ellipse)
        action = cv2.waitKey(10) & 0xFF
        if action == ord('q') or action == 113:
            break
    img.release()
    cv2.destroyAllWindows()
```

```
import cv2
import numpy as np
if __name__ == '__main__':
    img = cv2.imread('yahboom.jpg')
    ellipse = cv2.ellipse(img, (80,80), (20,50),0,0, 360,(255,0,255), 5)
    while True :
        cv2.imshow("ellipse",ellipse)
        action = cv2.waitKey(10) & 0xFF
        if action == ord('q') or action == 113:
            break
    img.release()
    cv2.destroyAllWindows()
```



3.8、opencv 绘制多边形

3.8.1、cv2.polylines(img,[pts],isClosed, color[,thickness[,lineType]])

参数含义：

- pts：多边形的顶点
- isClosed：是否闭合。（True/False）
- 其他参数参照圆的绘制参数

3.8.2、代码与实际效果展示

运行程序

```
cd /root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_astra/scripts/opencv/
```

```
python3 3_8.py
```

```
import cv2
import numpy as np
if __name__ == '__main__':
    img = cv2.imread('yahboom.jpg')
    points = np.array([[120,50], [40,140], [120,70], [110,110], [50,50]],np.int32)
    polylines = cv2.polylines(img, [points],True,(255,0,255), 5)
    while True :
        cv2.imshow("polylines",polylines)
        action = cv2.waitKey(10) & 0xFF
        if action == ord('q') or action == 113:
            break
    img.release()
    cv2.destroyAllWindows()
```

3.9、opencv 绘制文字

3.9.1、cv2.putText(img, str, origin, font, size,color,thickness)

参数含义：

-

img：输入图像

-
-

str：绘制的文字

-
-

origin：左上角坐标(整数)，可以理解成文字是从哪里开始的

-
-

font：字体

-
-

size：字体大小

-
-

  color：字体颜色

-
-

  thickness：字体粗细

-
-

  其中，字体可选，

-

FONT_HERSHEY_SIMPLEX Python：cv.FONT_HERSHEY_SIMPLEX	正常大小sans-serif字体
FONT_HERSHEY_PLAIN Python：cv.FONT_HERSHEY_PLAIN	小尺寸sans-serif字体
FONT_HERSHEY_DUPLEX Python：cv.FONT_HERSHEY_DUPLEX	正常大小的sans-serif字体（比FONT_HERSHEY_SIMPLEX更复杂）
FONT_HERSHEY_COMPLEX Python：cv.FONT_HERSHEY_COMPLEX	正常大小的衬线字体
FONT_HERSHEY_TRIPLEX Python：cv.FONT_HERSHEY_TRIPLEX	正常大小的serif字体（比FONT_HERSHEY_COMPLEX更复杂）
FONT_HERSHEY_COMPLEX_SMALL Python：cv.FONT_HERSHEY_COMPLEX_SMALL	较小版本的FONT_HERSHEY_COMPLEX
FONT_HERSHEY_SCRIPT_SIMPLEX Python：cv.FONT_HERSHEY_SCRIPT_SIMPLEX	手写风格的字体
FONT_HERSHEY_SCRIPT_COMPLEX Python：cv.FONT_HERSHEY_SCRIPT_COMPLEX	更复杂的FONT_HERSHEY_SCRIPT_SIMPLEX变体
FONT_ITALIC Python：cv.FONT_ITALIC	标志为斜体字体

-

3.9.2、代码与实际效果展示

运行程序

```
cd /root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_astra/scripts/opencv/
python3 3_9.py
```

```
import cv2
import numpy as np
if __name__ == '__main__':
    img = cv2.imread('yahboom.jpg')
    cv2.putText(img,'This is Yahboom!',(50,50),cv2.FONT_HERSHEY_SIMPLEX,1,(0,200,0),2)
    while True :
        cv2.imshow("img",img)
        action = cv2.waitKey(10) & 0xFF
        if action == ord('q') or action == 113:
            break
    img.release()
```

```
    cv2.destroyAllWindows()
```

```
import cv2
import numpy as np
if __name__ == '__main__':
    img = cv2.imread('yahboom.jpg')
    cv2.putText(img,'This is Yahboom!',(50,50),cv2.FONT_HERSHEY_SIMPLEX,1,(0,200,0),2)
    while True :
        cv2.imshow("img",img)
        action = cv2.waitKey(10) & 0xFF
        if action == ord('q') or action == 113:
            break
    img.release()
    cv2.destroyAllWindows()
```

4.1、opencv 修复图片

图像修复是计算机视觉中的一类算法，其目标是填充图像或视频内的区域。该区域使用二进制掩模进行标识，填充通常根据需要填充的区域边界信息来完成。图像修复的最常见应用是恢复旧的扫描照片。它还用于删除图像中的小的不需要的对象。

4.1.1、dst = cv2.inpaint(src, inpaintMask, inpaintRadius, flags)

参数含义：

- src：源图像，也就是需要修复的图像
- inpaintMask：二进制掩码，指示要修复的像素。
- dst：结果图像
- inpaintRadius：表示修复的半径

- flags：修复算法，主要有 INPAINT_NS (Navier-Stokes based method) or INPAINT_TELEA (Fastmarching based method)

  基于 Navier-Stokes 的修复应该更慢，并且倾向于产生比 fast marching method 的方法更模糊的结果。在实践中，我们没有发现这种情况。INPAINT_NS 在我们的测试中产生了更好的结果，速度也略高于 INPAINT_TELEA。

4.1.2、代码与实际效果展示

（1）、首先，我们先根据完好的图片，对其增加破损，可以理解成是修改其特定部分的像素值，

运行程序，

```
cd /root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_astra/scripts/opencv/
python3 4_1_1.py
```

```
import cv2
import numpy as np
if __name__ == '__main__':
    img = cv2.imread('yahboom.jpg')
```

```
    for i in range(50,100):
        img[i,50] = (0,0,0)
        img[i,50+1] = (0,0,0)
        img[i,50-1] = (0,0,0)
    for i in range(100,150):
        img[150,i] = (0,0,0)
        img[150,i+1] = (0,0,0)
        img[150-1,i] = (0,0,0)
    cv2.imwrite("damaged.jpg",img)
    dam_img = cv2.imread('damaged.jpg')
    while True :
        cv2.imshow("dam_img",dam_img)
        action = cv2.waitKey(10) & 0xFF
        if action == ord('q') or action == 113:
            break
    img.release()
    cv2.destroyAllWindows()
```

运行后会生成一张图片，这张图片看成是原图的破损图片，

（2）、修复刚才创建的照片，首先先读取，然后创建掩码，最后使用函数修复它

运行程序

```
cd /root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_astra/scripts/opencv/
python3 4_1_2.py
```

```
import cv2
import numpy as np
if __name__ == '__main__':
    dam_img = cv2.imread('damaged.jpg')
    imgInfo = dam_img.shape
    height = imgInfo[0]
    width = imgInfo[1]
    paint = np.zeros((height,width,1),np.uint8)
```

```
    for i in range(50,100):
        paint[i,50] = 255
        paint[i,50+1] = 255
        paint[i,50-1] = 255
    for i in range(100,150):
        paint[150,i] = 255
        paint[150+1,i] = 255
        paint[150-1,i] = 255
    dst_img = cv2.inpaint(dam_img,paint,3,cv2.INPAINT_TELEA)
    while True :
        cv2.imshow("dam_img",dam_img)
        cv2.imshow("paint",paint)
        cv2.imshow("dst",dst_img)
        action = cv2.waitKey(10) & 0xFF
        if action == ord('q') or action == 113:
            break
    img.release()
    cv2.destroyAllWindows()
```

如图所示，左边为修复前，中间的是掩码图片，右边是修复后的原图。

4.2、opencv 图片亮度增强

实现过程：对每个像素点的三通道值进行同步放大，同时保持通道值在 0-255 之间，实际上就是遍历每个像素点，给他们加减数值，然后再判断三个通道 rgb 是否在 0-255 区间，大于或者小于则取值 255 或者 0。

4.2.1、代码和实际效果展示

运行程序，

```
cd /root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_astra/scripts/opencv/
python3 4_2.py
```

```
import cv2
```

```
import numpy as np
if __name__ == '__main__':
    img = cv2.imread('yahboom.jpg')
    imgInfo = img.shape
    height = imgInfo[0]
    width = imgInfo[1]
    dst = np.zeros((height,width,3),np.uint8)
    for i in range(0,height):
        for j in range(0,width):
            (b,g,r) = img[i,j]
            bb = int(b) + 100
            gg = int(g) + 100
            rr = int(r) + 100
            if bb > 255:
                bb = 255
            if gg > 255:
                gg = 255
            if rr > 255:
                rr = 255
            dst[i,j] = (bb,gg,rr)
    while True :
        cv2.imshow("dst",dst)
```

```
        action = cv2.waitKey(10) & 0xFF
        if action == ord('q') or action == 113:
            break
    img.release()
    cv2.destroyAllWindows()
```

左边图片是原图，后边图片是增加亮度后的照片。

4.3、opencv 图片磨皮美白

OpenCV 实现对图片磨皮美白的功能，实现的原理和“1.20 OpenCV 图片亮度增强”的原理基本上是一样的，只不过这里我们不需要对 r 值做处理，只需要按照这个公式，p = p(x)*1.4+ y，其中 p(x)表示 b 通道或者 g 通道，y 表示需要增减的数值，同样的，加了数值后，我们需要对数值做判断。

4.3.1、代码与实际效果展示

运行程序

```
cd /root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_astra/scripts/opencv/
python3 4_3.py
```

```
import cv2
```

```
import numpy as np
if __name__ == '__main__':
    img = cv2.imread('yahboom.jpg')
    imgInfo = img.shape
    height = imgInfo[0]
    width = imgInfo[1]
    dst = np.zeros((height,width,3),np.uint8)
    for i in range(0,height):
        for j in range(0,width):
            (b,g,r) = img[i,j]
            bb = int(b*1.4) + 5
            gg = int(g*1.4) + 5
            if bb > 255:
                bb = 255
            if gg > 255:
                gg = 255
            dst[i,j] = (bb,gg,r)
    while True :
        cv2.imshow("origin",img)
        cv2.imshow("dst",dst)
        action = cv2.waitKey(10) & 0xFF
        if action == ord('q') or action == 113:
```

```
            break
    img.release()
    cv2.destroyAllWindows()
```

2、ROS+opencv 应用

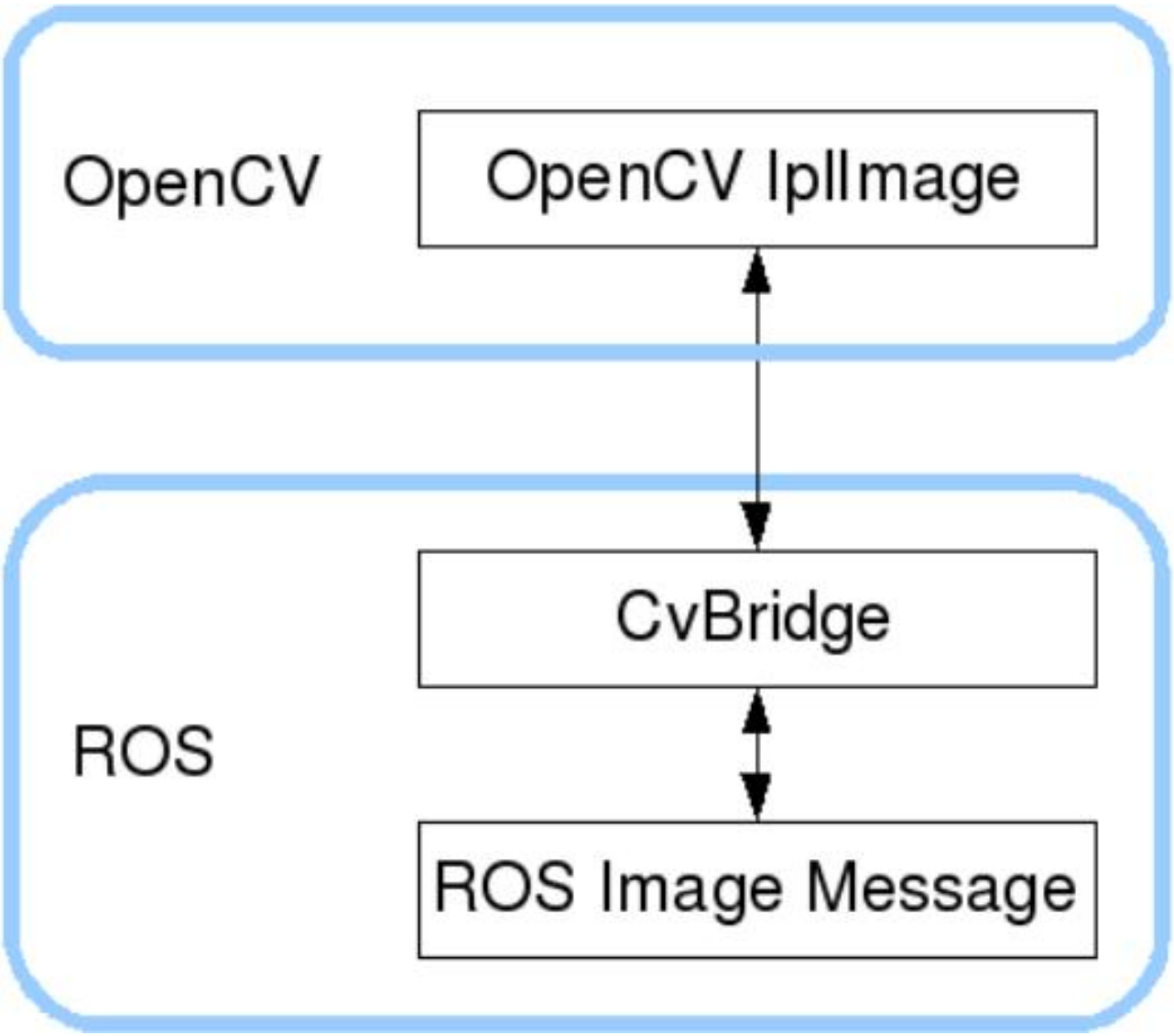
2、ROS+opencv 应用

本节课以 usb 免驱相机为例

ROS 以自己的 sensor_msgs/Image 消息格式传递图像，无法直接进行图像处理，但是提供的【CvBridge】可以完美转换和被转换图像数据格式。

【CvBridge】是一个 ROS 库，相当于 ROS 和 Opencv 之间的桥梁。

Opencv 和 ROS 图像数据转换如下图所示：



2.1、订阅图像数据然后发布转换后的图像数据

运行命令

```
#运行发布图像话题数据节点
ros2 run yahboomcar_visual pub_image

#以下节点二选一
#运行 usb 摄像头话题节点（带图像显示）
ros2 launch usb_cam demo_launch.py

#运行 usb 摄像头话题节点（不带图像显示）
ros2 launch usb_cam Demo_launch.py
```

相关注意事项:

1、在进入 docker 容器前先确定 usb 免驱摄像头设备是否接上，并且在启动脚本里把摄像头设备添加上去，具体教程请看【docker 使用教程——5、进入 docker 容器】

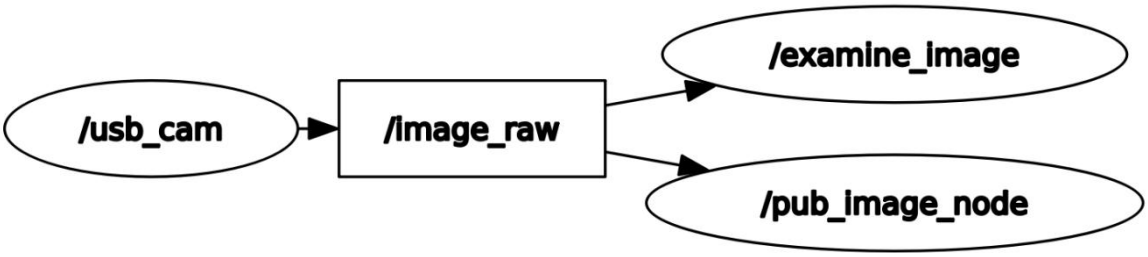
2、由于有些主板的性能不是很好，运行 usb 摄像头话题节点时 GUI 显示会很卡（例如 jetson nano），可以运行不带图像显示的节点，然后用下面教程中的 rqt_image_view 工具查看图像

查看节点通讯图

docker 终端输入

```
```

```
ros2 run rqt_graph rqt_graph
```



运行不带图像显示的节点指令时，在 rqt_graph 中不会显示

/examine_Image 该节点

查看话题数据

先查询有哪些图像话题发布，docker 终端输入，

```
ros2 topic list
```

```
root@jetson-desktop:/# ros2 topic list
/camera_info
/image
/image_raw
/image_raw/compressed
/image_raw/compressedDepth
/image_raw/theora
/parameter_events
/rosout
root@jetson-desktop:/#
```

其中的/image 就是我们发布的话题数据，用以下指令打印看看这个话题的数据内容，

```
ros2 topic echo /image
```

```
---
header:
  stamp:
    sec: 0
    nanosec: 0
  frame_id: ''
height: 480
width: 640
encoding: bgr8
is_bigendian: 0
step: 1920
data:
- 95
- 86
- 122
- 90
- 81
- 117
- 96
- 83
```

可以用 rqt_image_view 工具查看图像，

```
ros2 run rqt_image_view rqt_image_view
```

打开后在左上角选择话题名称/image，即可查看图像（有时候选择完话题后第一次加载不出来，可以结束进程重新输入指令进入界面即可显示图像）

2.2、核心代码解析

代码路径，

```
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_visual/yahboomcar_visual/pub_image.py
```

实现步骤大概上之前两个一样，程序首先订阅了/image_raw 的话题数据，然后转换成图像数据，但是这里还做了格转换，把图像数据转换成话题数据，然后发布出去，也就是图像话题数据->图像数据->图像话题数据。

```
#导入 opecv 库以及 cv_bridge 库
import cv2 as cv
from cv_bridge import CvBridge
#创建 CvBridge 对象
```

```
self.bridge = CvBridge()
#定义一个订阅者订阅 usb 图像话题数据
self.sub_img =
self.create_subscription(Image,'/image_raw',self.handleTopic,500)
#定义了图像话题数据发布者
self.pub_img = self.create_publisher(Image,'/image',500)
#msg 转换成图像数据 imgmsg_to_cv2，这里的 bgr8 是图像编码格式
frame = self.bridge.imgmsg_to_cv2(msg, "bgr8")
#图像数据转换的图像话题数据(cv2_to_imgmsg)然后发布出去
msg = self.bridge.cv2_to_imgmsg(frame, "bgr8")
self.pub_img.publish(msg)
```

3、二维码创建与识别

3、二维码创建与识别

本节教程以 usb 免驱摄像头为例，所有代码均在 docker 中运行，进入 docker 时需要挂载摄像头，具体请参考 docker 教程中的【5、进入 docker 容器】

1、QR 二维码

1.1、QR 二维码简介

QR 码是二维条码的一种，QR 来自英文 “Quick Response” 的缩写，即快速反应的意思，源自发明者希望 QR 码可让其内容快速被解码。QR 码不仅信息容量大、可靠性高、成本低，还可表示汉字及图像等多种文字信息、其保密防伪性强而且使用非常方便。更重要的是 QR 码这项技术是开源的。

1.2、QR 二维码的结构

图片	解析
	**定位标识**（Positioning markings）标明二维码的方向。
	**对齐标记**（Alignment markings）如果二维码很大，这些附加元素帮助定位。
	**计算模式**（Timing pattern）通过这些线，扫描器可以识别矩阵有多大。
	**版本信息**（Version information）这里指定正在使用的QR码的版本号，目前有QR码有40个不同的版本号。 用于销售行业的的版本号通常为1-7。
	**格式信息**（Format information）格式模式包含关于容错和数据掩码模式的信息，并使得扫描代码更加容易。
	**数据和错误校正值**（Data and error correction keys）这些模式保存实际数据。
	**宁静区域**（Quiet zone）这个区域对于扫描器来说非常重要，它的作用就是将自身与周边的进行分离。

1.3、QR 二维码的特点

QR 码中数据值包含重复的信息（冗余值）。因此，即使多达 30%的二维码结构被破坏，而不影响二维码的可读性。QR 码的存储空间多达 7089 位或者是 4296 个字符，包括标点符号和特殊字符，都可以写入 QR 码中。除了数字和字符之外，还可以对单词和短语（例如网址）进行编码。随着更多的数据被添加到 QR 码，代码大小增加，代码结构变得更加复杂。

1.4、QR 二维码创建与识别

1）、源码路径

```
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_visual/simple_qrcode
```

2）、安装包（出厂的 docker 镜像已经安装好）

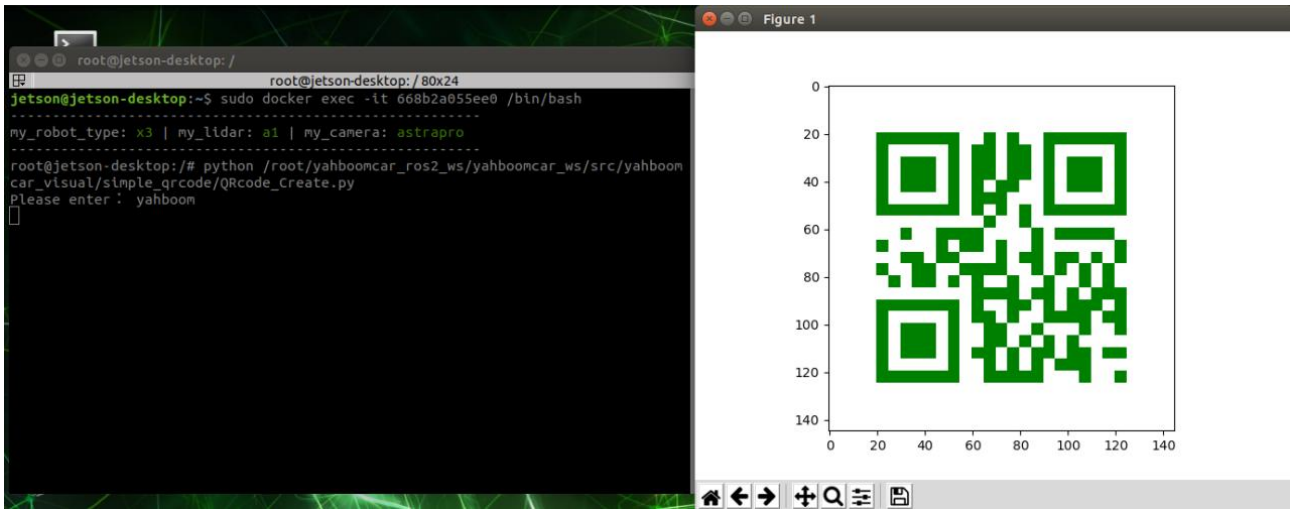
```
python3 -m pip install qrcode pyzbar
sudo apt-get install libzbar-dev
```

3）、创建 QRcode_Create.py

进入 docker，打开终端输入，

```
cd
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_visual/
simple_qrcode/
python QRcode_Create.py
```

程序运行后 会提示输入生成的内容，回车键确认内容。这里以创建"yahboom"字符串为内容作为例子，



右边出现的二维码，拿出手机试着扫描一下，扫描结果会是 yahboom 的字符。

源码解析，

```
#创建 qrcode 对象
qr = qrcode.QRCode(
        version=1,
```

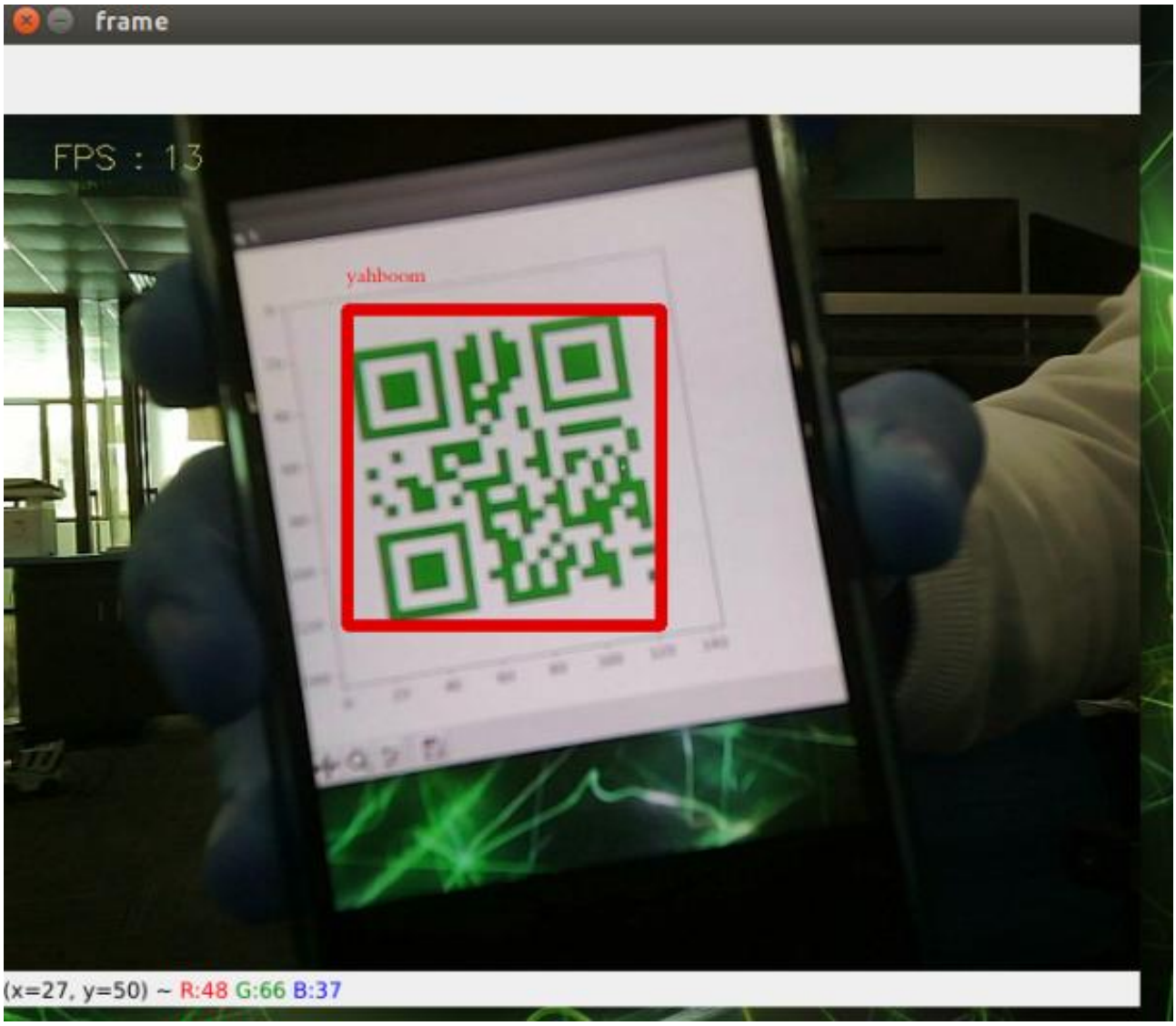
```
        error_correction=qrcode.constants.ERROR_CORRECT_H,
        box_size=5,
        border=4,)
#各参数含义
'''version：值为 1~40 的整数，控制二维码的大小（最小值是 1，是个 12×12 的矩阵）。
            如果想让程序自动确定，将值设置为 None 并使用 fit 参数即可。
error_correction：控制二维码的错误纠正功能。可取值下列 4 个常量。
        ERROR_CORRECT_L：大约 7%或更少的错误能被纠正。
        ERROR_CORRECT_M（默认）：大约 15%或更少的错误能被纠正。
        ROR_CORRECT_H：大约 30%或更少的错误能被纠正。
box_size：控制二维码中每个小格子包含的像素数。
border：控制边框（二维码与图片边界的距离）包含的格子数（默认为 4，是相关标准规定的最小值）'''
#qrcode 二维码添加 logo
my_file = Path(logo_path)
if my_file.is_file(): img = add_logo(img, logo_path)
#添加数据
qr.add_data(data)
#  填充数据
# fill data
```

```
qr.make(fit=True)
# 生成图片
# generate images
img = qr.make_image(fill_color="green", back_color="white")
```

4）、识别 QRcode_Parsing.py

进入 docker，打开终端输入，

```
cd /root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_visual/simple_qrcode/
python QRcode_Parsing.py
```



程序运行后，我们把二维码放置在摄像头前，程序会识别出二维码的内容，在图片上标识出来并且在终端会打印出识别的内容。

源码解析，

```
def decodeDisplay(image, font_path):
    gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
    # 需要先把输出的中文字符转换成 Unicode 编码形式
    barcodes = pyzbar.decode(gray)
```

```
    for barcode in barcodes:
    # 提取二维码的边界框的位置
    (x, y, w, h) = barcode.rect
    # 画出图像中条形码的边界框
    cv.rectangle(image, (x, y), (x + w, y + h), (225, 0, 0), 5)
    encoding = 'UTF-8'
    # 画出来，就需要先将它转换成字符串
    barcodeData = barcode.data.decode(encoding)
    barcodeType = barcode.type
    # 绘出图像上数据和类型
    pilimg = Image.fromarray(image)
    # 创建画笔
    draw = ImageDraw.Draw(pilimg)
    # 参数 1：字体文件路径，参数 2：字体大小
    fontStyle = ImageFont.truetype(font_path, size=12,
encoding=encoding)
    # 参数 1：打印坐标，参数 2：文本，参数 3：字体颜色，参数 4：字体
    draw.text((x, y - 25), str(barcode.data, encoding), fill=(255, 0,
0),font=fontStyle)
    # PIL 图片转 cv2 图片
    image = cv.cvtColor(np.array(pilimg), cv.COLOR_RGB2BGR)
    # 向终端打印条形码数据和条形码类型
```

```
    print("[INFO] Found {} barcode: {}".format(barcodeType,
barcodeData))
    return image
```

4、AR 视觉

4、AR 视觉

本节教程以 usb 免驱摄像头为例，所有代码均在 docker 中运行，进入 docker 时需要挂载摄像头，具体请参考 docker 教程中的【5、进入 docker 容器】

1、概述

增强现实(Augmented Reality)，简称“AR”，技术是一种将虚拟信息与真实世界巧妙融合的技术，广泛运用了多媒体、三维建模、实时跟踪及注册、智能交互、传感等多种技术手段，将计算机生成的文字、图像、三维模型、音乐、视频等虚拟信息模拟仿真后，应用到真实世界中，两种信息互为补充，从而实现对真实世界的“增强”。

AR 系统具有三个突出的特点：①真实世界和虚拟世界的信息集成；②具有实时交互性；③是在三维尺度空间中增添定位虚拟物体。

增强现实技术包含了多媒体、三维建模、实时视频显示及控制、多传感器融合、实时跟踪及注册、场景融合等新技术与新手段。

2、使用方法

在使用 AR 案例的时候，必须要有相机的内参，不然无法运行。内参文件与代码同目录，不同相机对应不同的内参。本节课程以 usb 免驱摄像头为例，内参标定可用棋盘格快速标定**（docker 镜像中已经完成这一步骤）**

本节案例一共有 12 种效果，

```
["Triangle", "Rectangle", "Parallelogram","WindMill","TableTennisTable", "Ball","Arrow", "Knife", "Desk","Bench", "Stickman", "ParallelBars"]
```

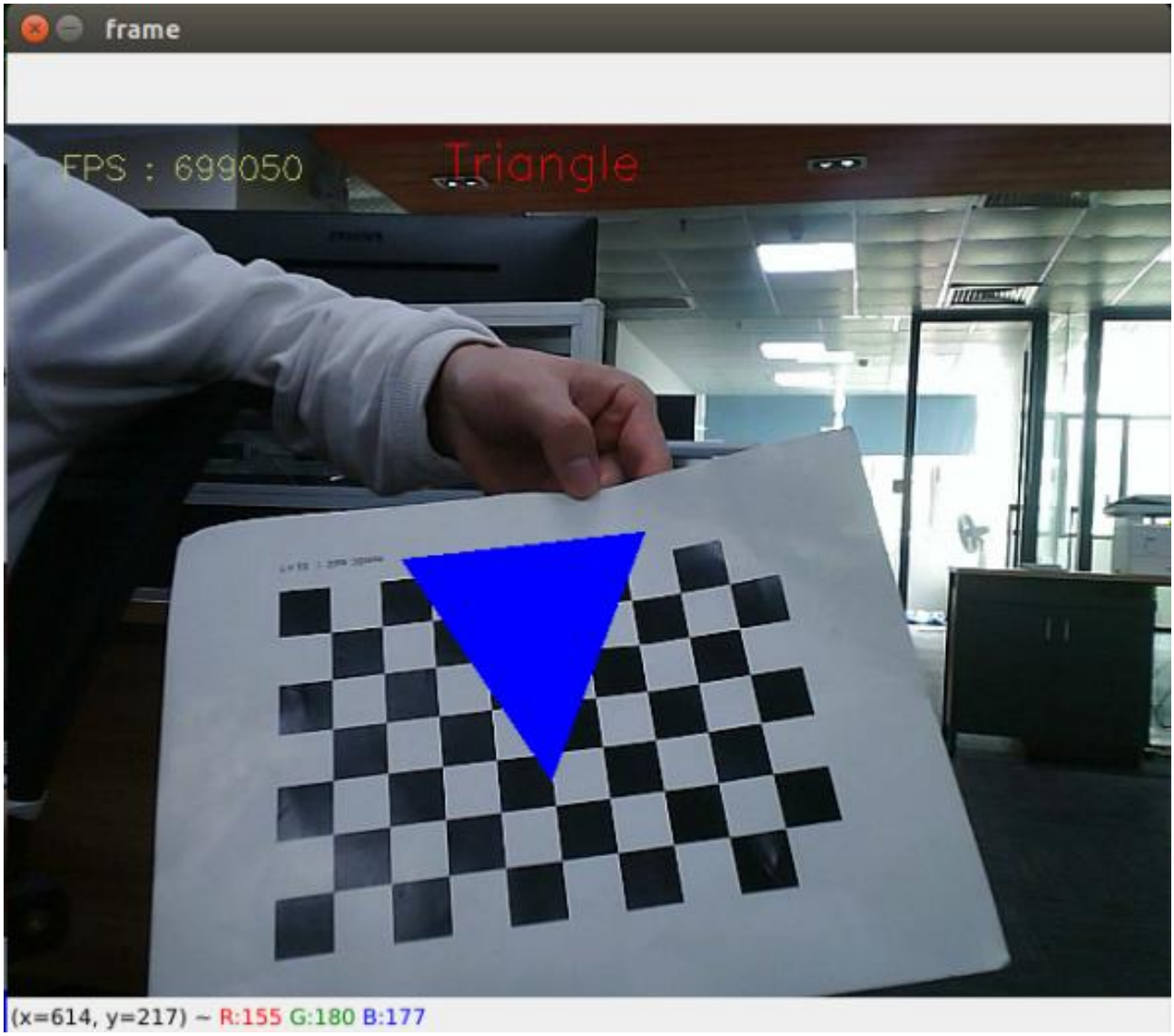
3、启动命令

代码参考路径

```
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_visual/yahboomcar_visual/simple_AR.py
```

进入 docker 容器后，docker 终端输入

```
ros2 run yahboomcar_visual simple_AR
```



【q】键退出，【f】键切换不同效果。

注意事项：有些主板由于性能一般，GUI 显示很卡，导致按键功能失效，有时候能使用按键切换有时候按了没反应，如果需要切换不同效果，可以参考下面的发布不同的话题来达到切换效果

3.1.1、ROS 部署

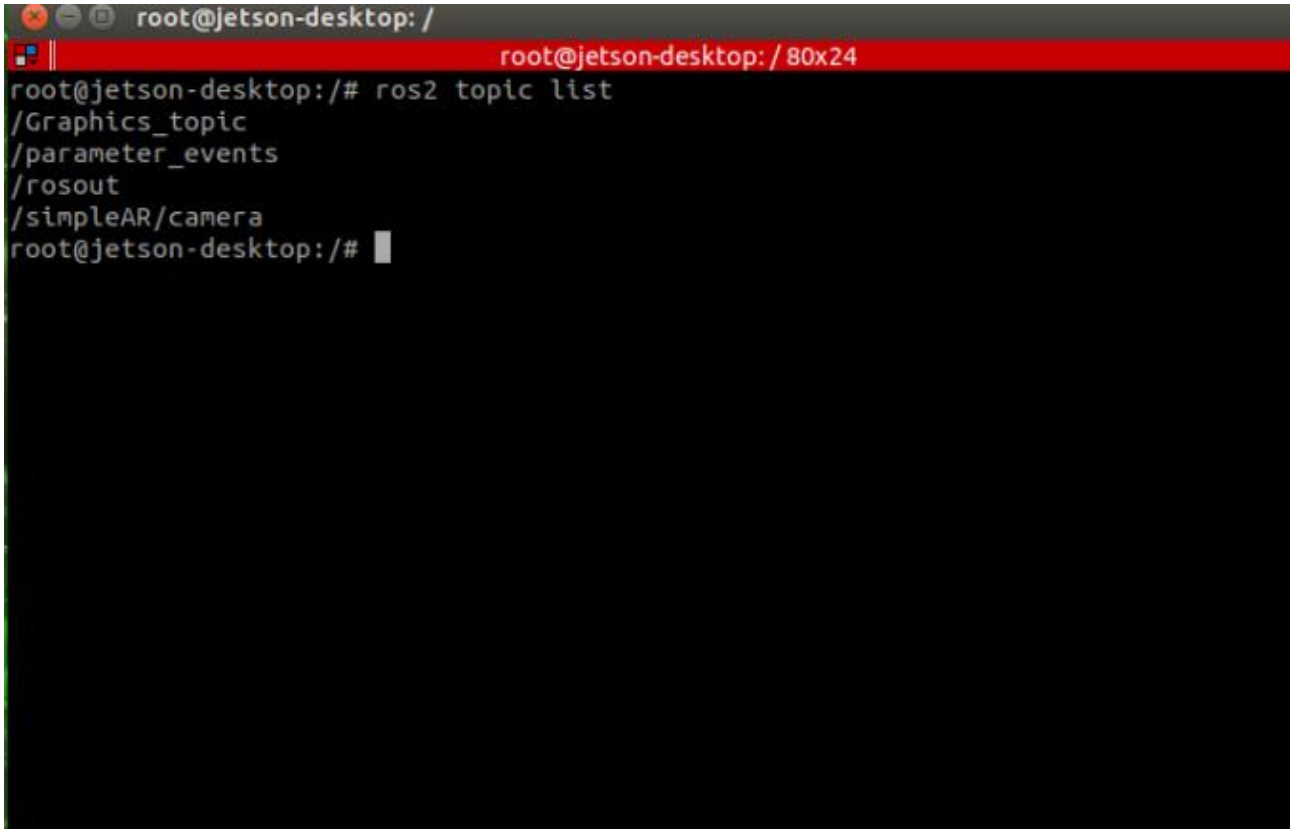
本节课程还部署了 ROS，主要是有以下两个功能：

订阅话题数据，切换不同效果

发布图像

通过以下命令查看 ros 话题，docker 终端输入

```
ros2 topic list
```

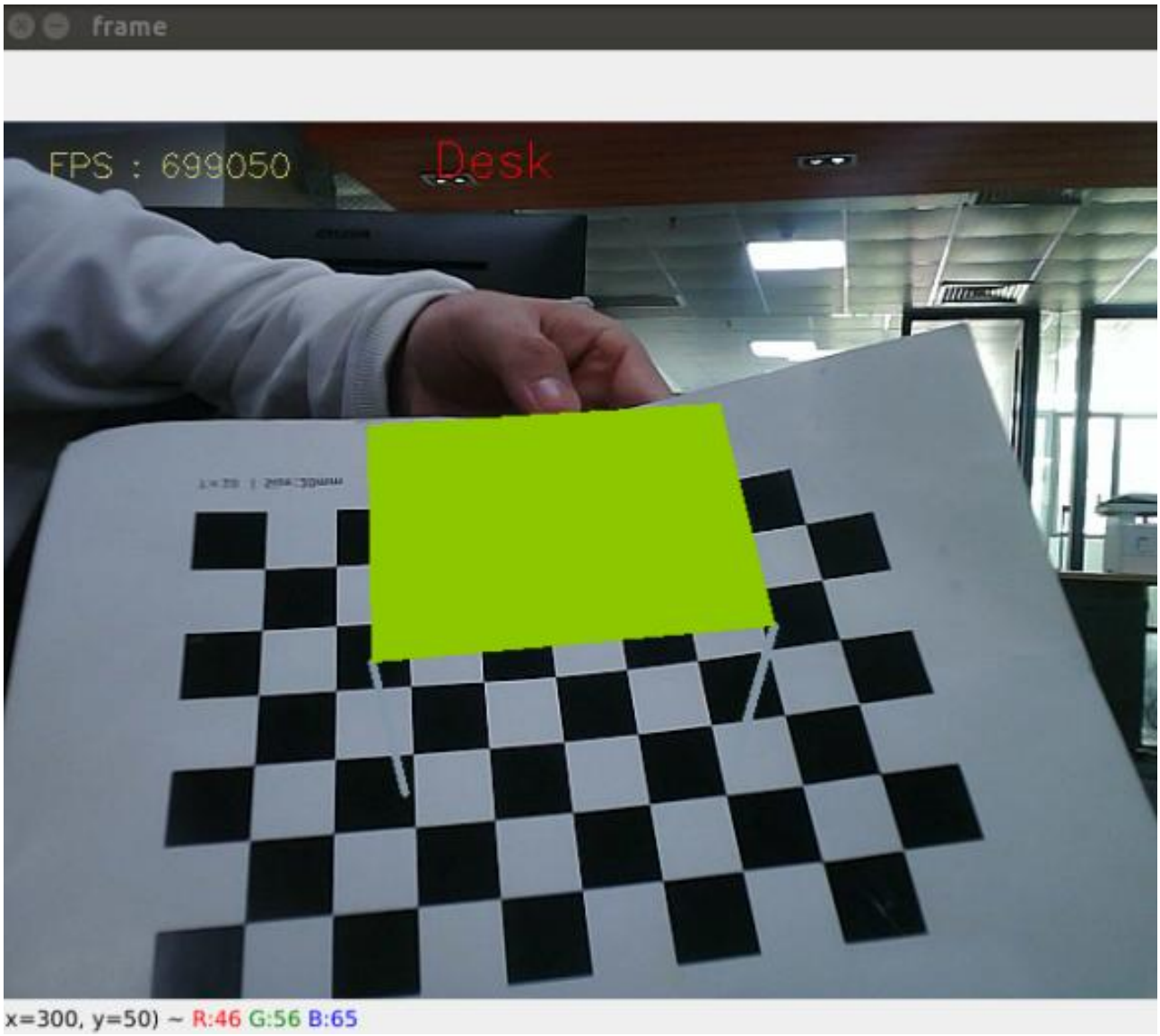


- /Graphics_topic：效果的话题名称，订阅需要识别的效果。
- /simpleAR/camera：图像的话题名称，发布图像。

修改效果，可以通过以下命令修改，例如，我先修改成 Desk（效果名字前文已经给出，可以自行修改）

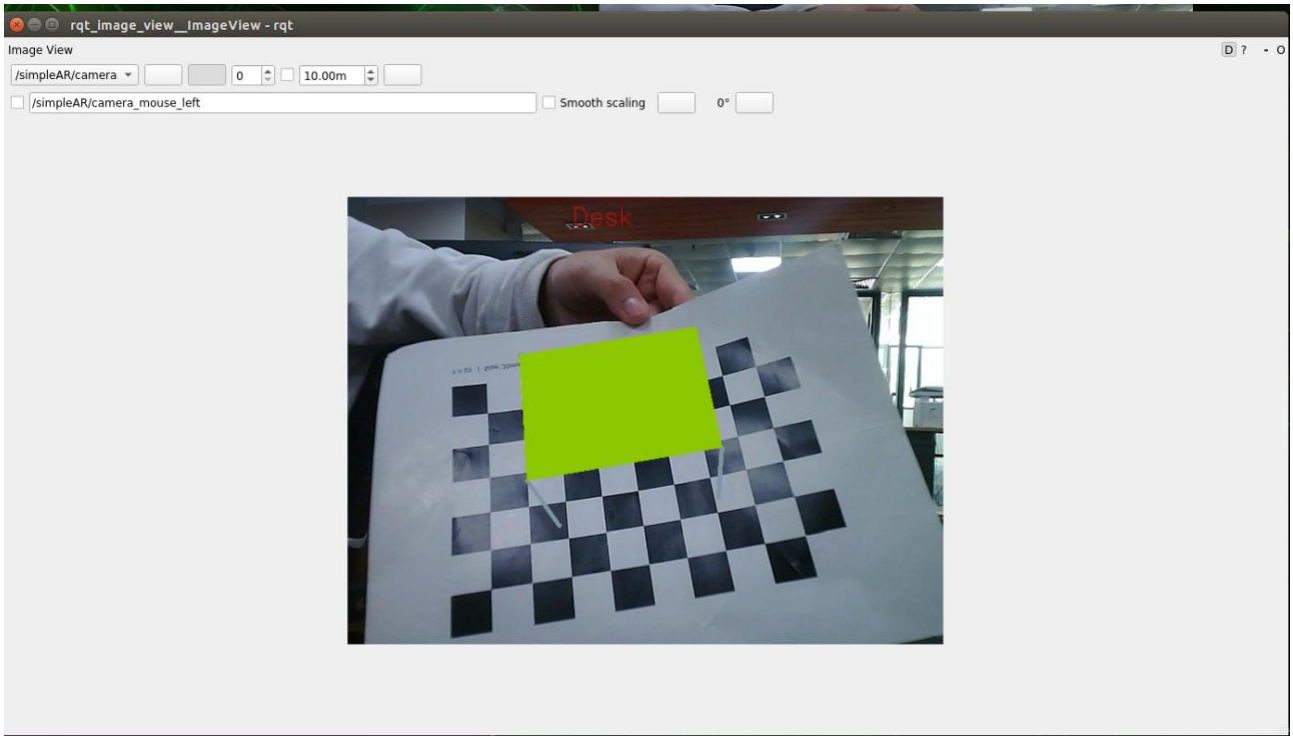
docker 终端输入

```
ros2 topic pub /Graphics_topic std_msgs/msg/String "data: Desk"
```



查看发布的图像可以使用 rqt_image_view 来查看，docker 终端输入，

```
ros2 run rqt_image_view rqt_image_view
```



左上角话题选择/simpleAR/camera 即可查看图像。

5、mediapipe 趣味玩法

5、mediapipe 趣味玩法

本节教程以 usb 免驱摄像头为例，所有代码均在 docker 中运行，进入 docker 时需要挂载摄像头，具体请参考 docker 教程中的【5、进入 docker 容器】

1、mediapipe 简介

MediaPipe 是一款由 Google 开发并开源的数据流处理机器学习应用开发框架。它是一个基于图的数据处理管线，用于构建使用了多种形式的数据源，如视频、音频、传感器数据以及任何时间序列数据。MediaPipe 是跨平台的，

可以运行在嵌入式平台(树莓派等)，移动设备(iOS 和 Android)，工作站和服务器上，并支持移动端 GPU 加速。MediaPipe 为实时和流媒体提供跨平台、可定制的 ML 解决方案。MediaPipe 的核心框架由 C++ 实现，并提供 Java 以及 Objective C 等语言的支持。MediaPipe 的主要概念包括数据包（Packet）、数据流（Stream）、计算单元（Calculator）、图（Graph）以及子图（Subgraph）。

MediaPipe 的特点：

端到端加速：内置的快速 ML 推理和处理即使在普通硬件上也能加速；

一次构建，随时随地部署：统一解决方案适用于 Android、iOS、桌面/云、web 和物联网；

即用解决方案：展示框架全部功能的尖端 ML 解决方案；

免费开源：Apache2.0 下的框架和解决方案，完全可扩展和定制。

2、使用

2.1、程序运行

源码路径参考，

```
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_mediapipe/yahboomcar_mediapipe
```

docker 终端输入，

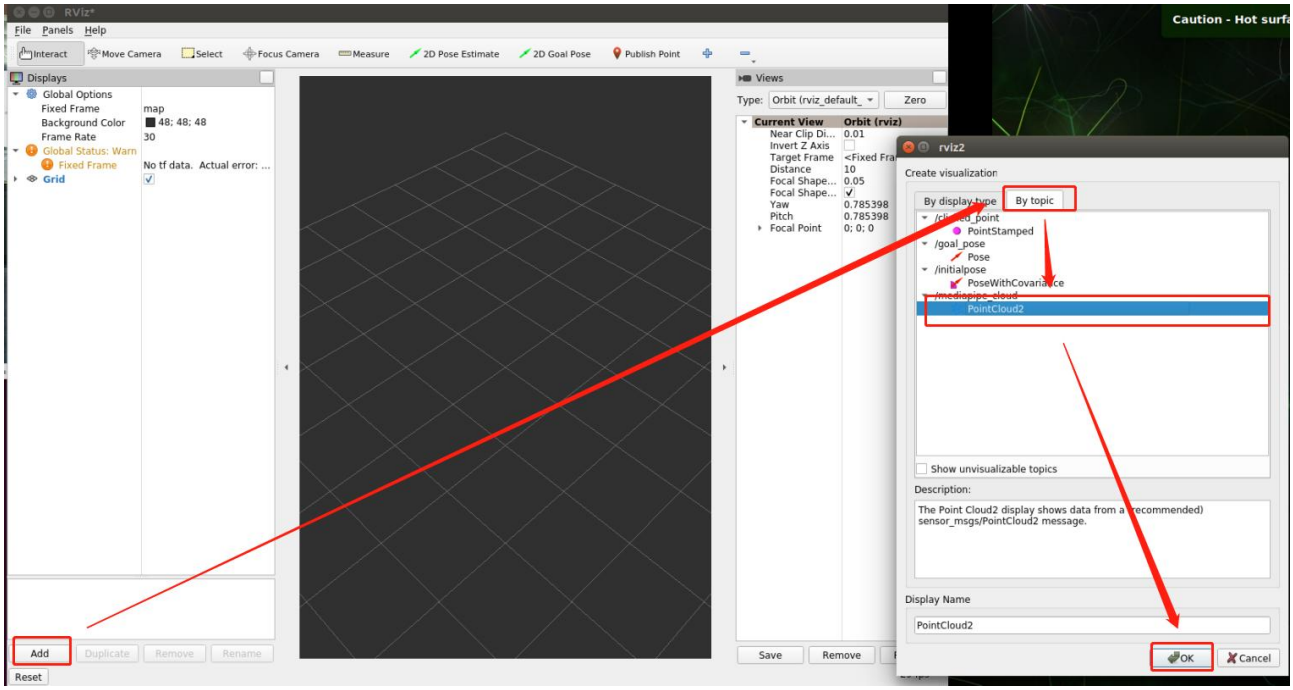
```
# 01_手部检测
ros2 run yahboomcar_mediapipe 01_HandDetector
# 02_姿态检测
ros2 run yahboomcar_mediapipe 02_PoseDetector
# 03_整体检测
ros2 run yahboomcar_mediapipe 03_Holistic
# 04_面部检测
ros2 run yahboomcar_mediapipe 04_FaceMesh

# 05_人脸识别
ros2 run yahboomcar_mediapipe 05_FaceEyeDetection
```

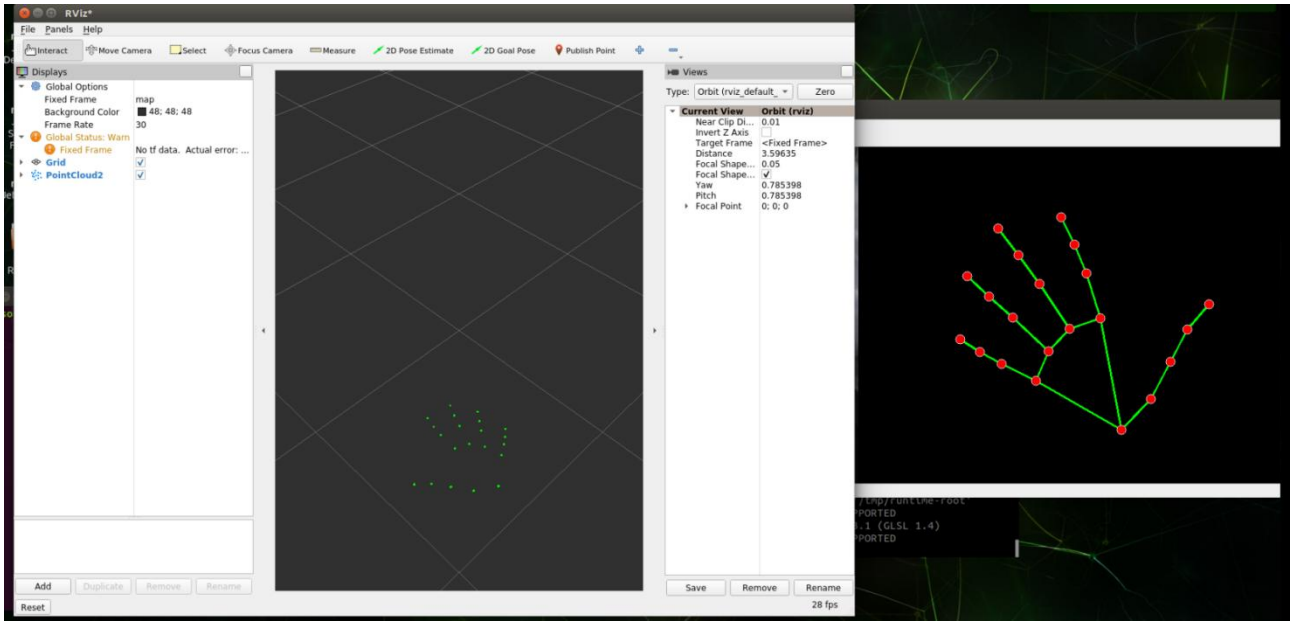
以手部检测为示例，运行截图如下

另外，还可以查看点云数据，docker 终端输入，

```
#运行点云发布程序
ros2 run yahboomcar_point pub_point
#开启 rviz 查看点云
rviz2
```

按照如下步骤，在 rviz 中添加点云话题，

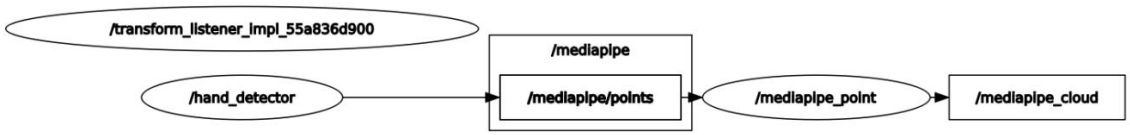




上边是运行手掌检测也就是 01_HandDetector 的程序的点云图。点云查看仅支持 01-04 的 demo。

可以通过 rqt_graph 查看节点话题通讯图，

```
ros2 run   rqt_graph rqt_graph
```



2.2、其他趣味玩法

docker 终端输入，

```
cd
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_media
pipe/yahboomcar_mediapipe

#人脸特效
python3 06_FaceLandmarks.py
#人脸检测
python3 07_FaceDetection.py
#三维物体识别
python3 08_Objectron.py
#画笔
python3 09_VirtualPaint.py
```

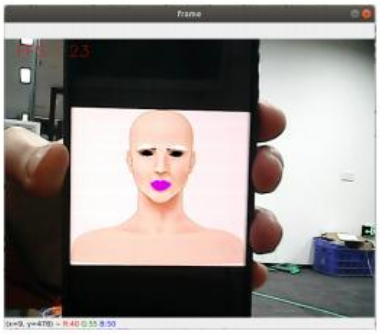
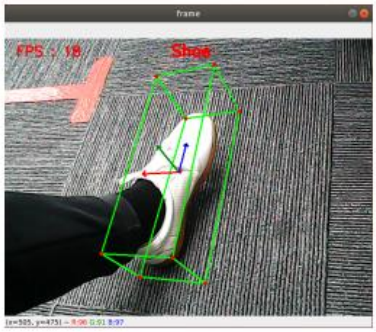
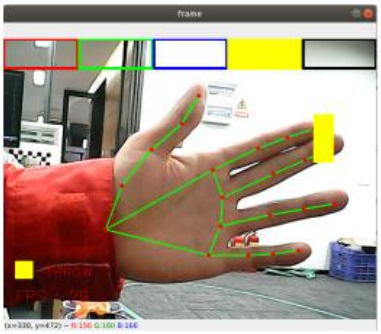
#手指控制

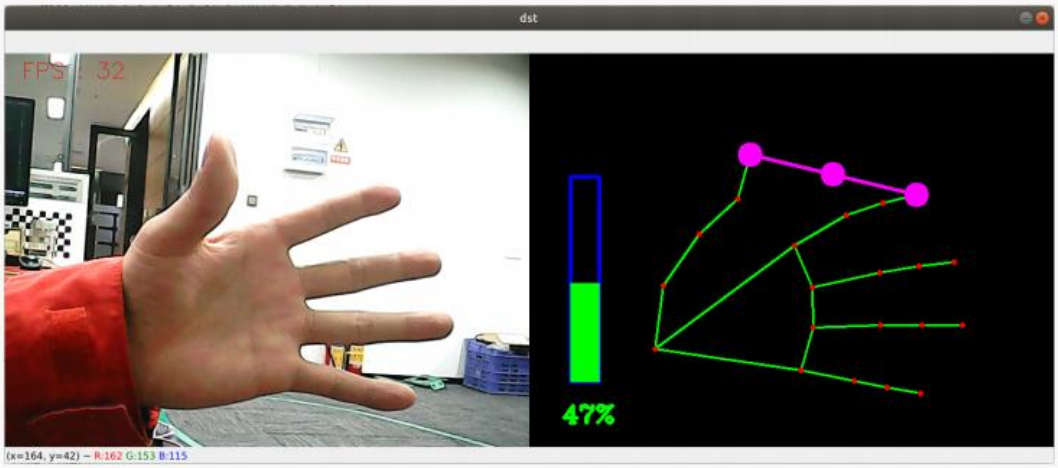
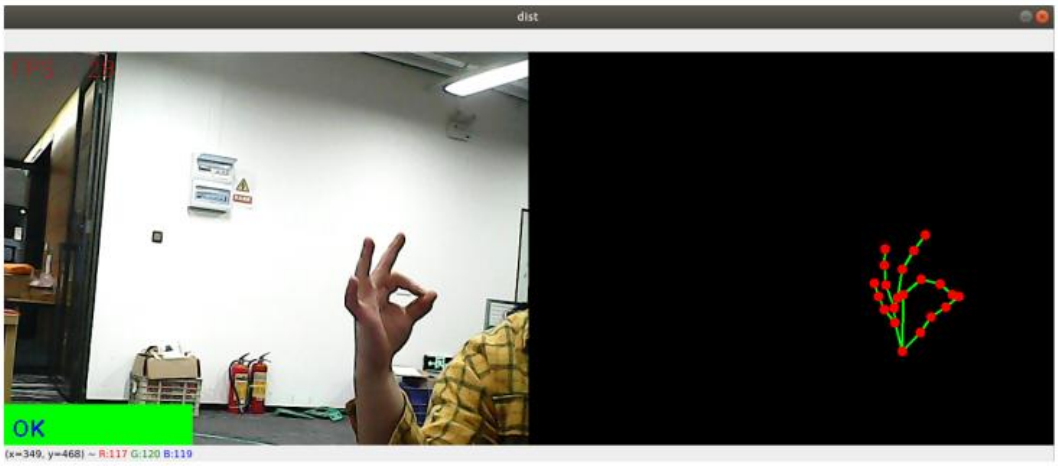
python3 10_HandCtrl.py

#手势识别

python3 11_GestureRecognition.py

3

05、人脸识别	06、人脸特效	07、人脸检测
		
08、三维物体识别		**09、画笔**
		

10、手指控制	
11、手势识别	

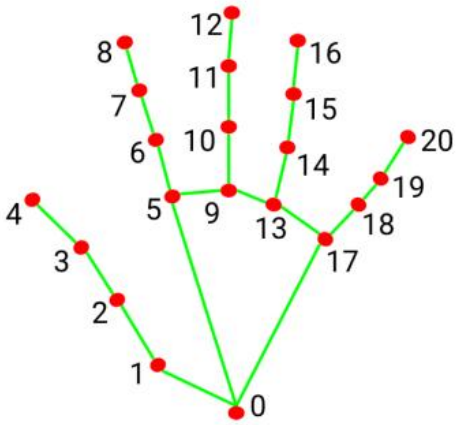
3、Mediapipe Hands

MediaPipe Hands 是一款高保真的手和手指跟踪解决方案。它利用机器学习（ML）从一帧中推断出 21 个手的 3D 坐标。

在对整个图像进行手掌检测后，根据手部标记模型通过回归对检测到的手区域内的 21 个 3D手关节坐标进行精确的关键点定位，即直接坐标预测。该模型学习一致的内部手姿势表示，甚至对部分可见的手和自我遮挡也具有鲁棒性。

为了获得地面真实数据，用了 21 个 3D 坐标手动注释了约 30K 幅真实世界的图像，如下所示 (从图像深度图中获取 Z 值，如果每个对应坐标都有 Z 值)。

为了更好地覆盖可能的手部姿势，并对手部几何体的性质提供额外的监督，还绘制了各种背景下的高质量合成手部模型，并将其映射到相应的 3D 坐标。

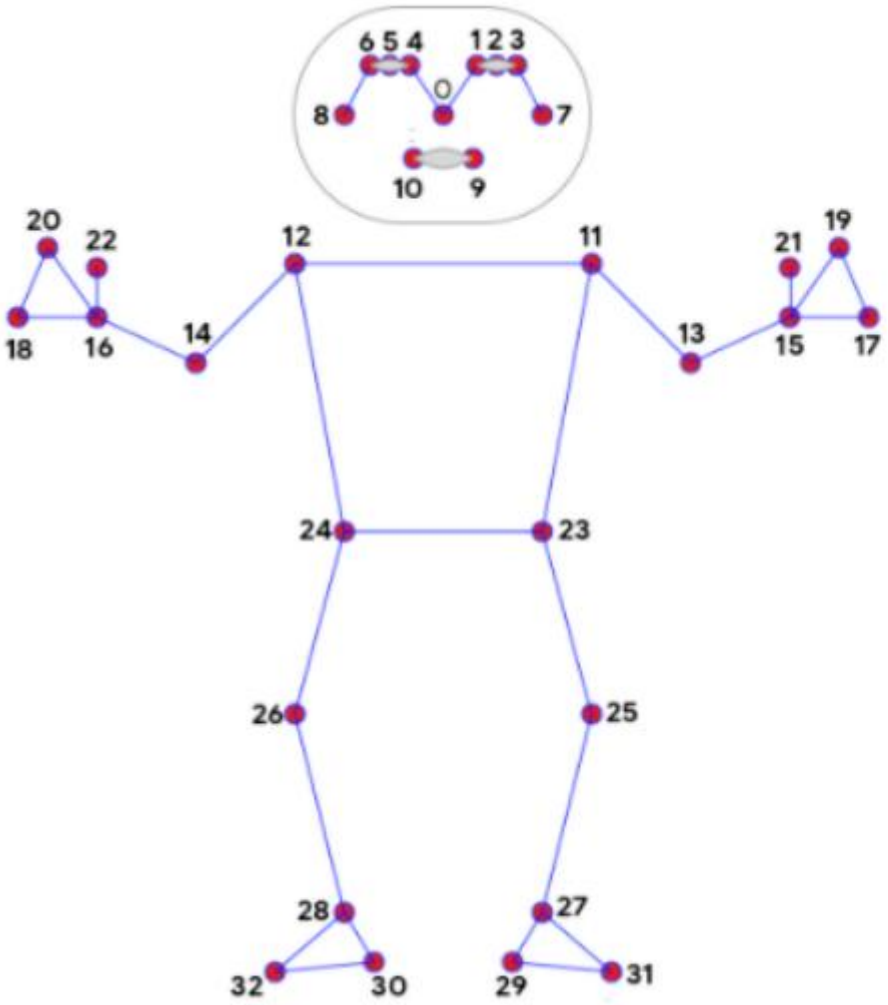


0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

4、Mediapipe Pose

MediaPipe Pose 是一个用于高保真身体姿势跟踪的 ML 解决方案，利用 BlazePose 研究，从 RGB 视频帧推断出 33 个 3D 坐标和全身背景分割遮罩，该研究也为 ML Kit 姿势检测 API 提供了动力。

MediaPipe 姿势中的地标模型预测了 33 个姿势坐标的位置（参见下图）。



0. nose
1. left_eye_inner
2. left_eye
3. left_eye_outer
4. right_eye_inner
5. right_eye
6. right_eye_outer
7. left_ear
8. right_ear
9. mouth_left
10. mouth_right
11. left_shoulder
12. right_shoulder
13. left_elbow
14. right_elbow
15. left_wrist
16. right_wrist

5、dlib

对应的案例是人脸特效。

DLIB 是一个现代 C++工具包，包含机器学习算法和工具，用于在 C++中创建复杂的软件来解决现实世界问题。它被工业界和学术界广泛应用于机器人、嵌入式设备、移动电话和大型高性能计算环境等领域。dlib 库采用68 点位置标志人脸重要部位，比如 18-22 点标志右眉毛，51-68 标志嘴巴。使用dlib 库的 get_frontal_face_detector 模块探测出人脸，使用 shape_predictor_68_face_landmarks.dat 特征数据预测人脸特征数值。