Pi-aSLAM 全记录

1 起步工作

1.1 SD卡烧录

如果树莓派出现红灯常亮，绿灯闪烁几下后不亮，拔下SD卡后绿灯有规律闪烁，则说明SD卡出现问题，需要重新烧录。因此建议对SD卡进行全盘备份。

烧写参考 https://blog.csdn.net/ourkix/article/details/113412367

进行烧写需要SD卡格式化工具（系统自带工具即可）和映像写入工具（可用win32 image writer）

1.2 系统初始化

该过程需要从树莓派外接键鼠、显示屏。

1.2.1 显示屏设置

在VNC连接前，**显示屏需要从mini-HDMI口连接**

为了正常显示，需要修改SD卡固件部分的config.txt，确保以下几项没有被注释掉。

```
hdmi_force_hotplug=1
config_hdmi_boost=4
hdmi_group=2
hdmi_mode=16
hdmi_drive=2
```

参考链接：https://blog.csdn.net/weixin_30045751/article/details/114020270

注意：固件中设置的hdmi_mode会覆盖系统里的设置。如果想在系统中改变分辨率，需要把固件的这行注释掉。

1.2.2 系统初始化

用户名是pi（密码自己定，我的123456）

Ctrl+Shift+T 可以打开terminal

1.2.3 打开SSH

```
sudo raspi-config
```

3 Interface config 找到SSH，打开

1.2.4 打开VNC server

同上

```
sudo raspi-config
```

3 Interface config 找到VNC设置，开启

显示分辨率的设置在2 Display Settings

1.2.5 树莓派重启

```
sudo shutdown -r now
```

1.3 VNC连接

1.3.1 找到树莓派IP地址

确保以下设置无误：

1. wifi的适配器设置，共享页面，允许其他设备连接和控制
2. 进一步设置里允许1703服务
3. 重新设置后拔掉网线重新插入

1.3.2 VNC连接

电脑需要先安装VNC Viewer。打开，填写用户名密码，注意IP地址。

1.4 网络问题

1.4.1 网络重连

如果电脑的wifi连接切换，树莓派就会出现连不到网的情况。尝试从树莓派ping境内网站

```
ping www.baidu.com
```

如果不通，将适配器设置里的共享关闭再重新打开，然后拔掉网线重新插上，再次连接后就可以了。

1.4.2 github连接

电脑不开梯子，访问ip-lookup，输入github.com，得到一条ip地址（如192.30.255.112）。打开系统DNS文件

```
sudo vi /etc/hosts
```

加入条目

```
151.101.72.249 github.global.ssl.fastly.net
192.30.255.112 github.com
```

修改完**需要重启**，否则无法生效。

2 软硬件配置调试

2.1 系统换源

```
sudo nano /etc/apt/sources.list
```

加这两行

```
deb http://mirrors.tuna.tsinghua.edu.cn/raspbian/raspbian/ buster main non-free
contrib
deb-src http://mirrors.tuna.tsinghua.edu.cn/raspbian/raspbian/ buster main non-
free contrib
```

然后

```
sudo apt-get uodate
```

换源完可以安装一个vim试一下

```
sudo apt-get install vim
```

2.2 安装opencv v3

2.2.1 准备工作

清理不需要的大型软件

```
sudo apt-get purge wolfram-engine
sudo apt-get purge libreoffice*
sudo apt-get clean
sudo apt-get autoremove
```

系统更新

```
sudo apt-get upgrade
```

2.2.2 安装必要工具

安装build-essential、cmake、git和pkg-config

```
sudo apt-get install build-essential cmake git pkg-config
```

安装图像工具包

```
sudo apt-get install libjpeg8-dev
sudo apt-get install libtiff5-dev
sudo apt-get install libjasper-dev
sudo apt-get install libpng12-dev
```

安装视频工具包

```
sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev
```

安装GTK

```
sudo apt-get install libgtk2.0-dev
```

安装数值优化工具包

```
sudo apt-get install libatlas-base-dev gfortran
```

2.2.3 下载源码和解压

下载Opencv 3.4.3

```
wget -O opencv-3.4.3.zip https://github.com/Itseez/opencv/archive/3.4.3.zip
```

解压OpenCV

```
unzip opencv-3.4.3.zip
```

下载OpenCV_contrib库：

```
wget -O opencv_contrib-3.4.3.zip
https://github.com/Itseez/opencv_contrib/archive/3.4.3.zip
```

解压OpenCV_contrib库：

```
unzip opencv_contrib-3.4.3.zip
```

2.2.4 编译前配置

建立目录

```
cd ~/opencv-3.4.3
mkdir build
cd build
```

cmake设置

```
/** CMAKE_BUILD_TYPE是编译方式
* CMAKE_INSTALL_PREFIX是安装目录
* OPENCV_EXTRA_MODULES_PATH是加载额外模块
* INSTALL_PYTHON_EXAMPLES是安装官方python例程
* BUILD_EXAMPLES是编译例程（这两个可以不加，不加编译稍微快一点点，想要C语言的例程的话，
在最后一行前加参数INSTALL_C_EXAMPLES=ON，要C++例程的话在最后一行前加参数
INSTALL_C_EXAMPLES=ONINSTALL_CXX_EXAMPLES=ON）
**/

sudo cmake -D CMAKE_BUILD_TYPE=RELEASE \
    -D CMAKE_INSTALL_PREFIX=/usr/local \
    -D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib-3.4.3/modules \
    -D INSTALL_PYTHON_EXAMPLES=ON \
    -D INSTALL_CXX_EXAMPLES=ON \
    -D BUILD_EXAMPLES=ON ..
```

备份build文件

因为下一步的编译会使用build文件中的东西，假如编译失败后还要重新进行cmake，比较耽误时间，这里可以直接备份一下cmake好的build文件夹，命名为build1，重新make的时候可以拿来用。

```
cd ..
cp -r build ./build1
```

2.2.5 xfeatures2d相关文件

有11个文件需要在上一步下载，但一般会下载失败，需要手动导入opencv_contrib/modules/xfeatures2d/src/这个路径

2.2.6 为树莓派增加SWAP

在开始编译之前，建议你增加交换空间。这将使你使用树莓派的所有四个内核来编译OpenCV，而不会由于内存耗尽导致编译挂起。

```
sudo nano /etc/dphys-swapfile
```

然后编辑 CONF_SWAPSIZE 变量：从100增加到2048

重新启动交换服务:

```
sudo /etc/init.d/dphys-swapfile stop
sudo /etc/init.d/dphys-swapfile start
```

2.2.7 开始编译

```
sudo make -j4 2>&1 | tee make.log
```

编译需要至少30分钟，可能会遇到各种bug，需要从头再来

[BUG.1] 编译opencv-找不到cuda文件问题

报错

```
/home/pi/Downloads/opencv-
3.4.3/modules/stitching/include/opencv2/stitching/detail/matchers.hpp:52:12: fatal
error: opencv2/xfeatures2d/cuda.hpp: No such file or directory
#  include "opencv2/xfeatures2d/cuda.hpp"
```

解决方法：在stitching/CMakeList.txt里加入include path，或将该文件中的这个include改为绝对路径。

注意：如果改成绝对路径，编译完就不要急着按2.2.8删掉中间文件，否则会造成orb-slam2编译失败

编译完成后，不要忘记**install**

```
sudo make install
```

以及更新动态链接库

```
sudo ldconfig
```

2.2.8 收尾工作

首先可以用python测试一下是否安装正确

```
pi@raspberrypi:~ $ python3
Python 3.7.3 (default, Jan 22 2021, 20:04:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import cv2
>>> cv2.__version__
'3.4.3'
```

要记着把交换区设置改回来（见2.1.6）

```
sudo nano /etc/dphys-swapfile
```

编译文件如果之后不需要就可以删掉了

```
cd ../..
sudo rm -rf opencv*
```

2.3 安装orb-slam2

2.3.1 安装依赖库

```
sudo apt-get install libboost-all-dev libblas-dev liblapack-dev
```

2.3.2 安装eigen

首先官网下载eigen 3.2.10

```
https://gitlab.com/libeigen/eigen/-/releases/3.2.10
```

下载解压后，进行编译安装

```
mkdir build
cd build
cmake ..
make
sudo make install
```

2.3.3 安装Pangolin

Pangolin是一个绘图库，orb-slam2用它来画轨迹图。

安装Pangolin前需要使用如下命令安装libglew-dev，不然编译不过。

```
sudo apt-get install libglew-dev
```

从github上下载Pangolin源码。

```
https://github.com/stevenlovegrove/Pangolin/releases/tag/v0.5
```

注意此处**不要下载master分支**，要使用**v0.5分支**！master是开发分支，不稳定，0.6版本也有bug，必须用0.5版本。

在终端中进入源码主目录，并输入以下命令完成Pangoline的编译，安装。

```
mkdir build
cd build
cmake ..
make
sudo make install
```

[BUG.2] CODEC_FLAG_GLOBAL_HEADER 未声明问题：

```
/home/pi/Downloads/Pangolin-0.5/src/video/drivers/ffmpeg.cpp:501:33: error:
‘CODEC_FLAG_GLOBAL_HEADER’ was not declared in this scope
        stream->codec->flags |= CODEC_FLAG_GLOBAL_HEADER;
```

解决方法：注释掉Pagolin/src/CMakeList.txt中FFMPEG相关的一段

```
#find_package(FFMPEG QUIET)
#if(BUILD_PANGOLIN_VIDEO AND FFMPEG_FOUND)
#  set(HAVE_FFMPEG 1)
#  list(APPEND INTERNAL_INC  ${FFMPEG_INCLUDE_DIRS} )
#  list(APPEND LINK_LIBS ${FFMPEG_LIBRARIES} )
#  list(APPEND HEADERS ${INCDIR}/video/drivers/ffmpeg.h)
#  list(APPEND SOURCES video/drivers/ffmpeg.cpp)
#  message(STATUS "ffmpeg Found and Enabled")
#endif()
```

2.3.4 为树莓派增加SWAP

ORB-SLAM2的编译也必须像2.2.6一样增加虚拟内存，否则会因为内存耗尽导致卡死。

```
sudo nano /etc/dphys-swapfile
```

编辑 CONF_SWAPSIZE 变量：从100增加到2048

重新启动交换服务:

```
sudo /etc/init.d/dphys-swapfile stop
sudo /etc/init.d/dphys-swapfile start
```

2.3.5 下载ORB_SALM2源码

可以用git clone

```
git clone https://github.com/raulmur/ORB_SLAM2.git
```

为了防止编译时卡死，可以限制一下编译使用的核数。具体来说，打开build.sh文件夹

```
cd ORB_SLAM2
vim ./build.sh
```

把所有的 'make -j' 改为 'make -j4'即可。

接下来，进行编译

```
chmod +x build.sh
./build.sh 2>&1 | tee ../build.log
```

编译耗时较长，至少30分钟，需要耐心等待。

[BUG.3] 静态断言问题

```
/home/pi/Pi-aSLAM/ORB_SLAM2-master/src/LoopClosing.cc:438:21:   required from here
/usr/include/c++/8/bits/stl_map.h:122:21: error: static assertion failed: std::map
must have the same value_type as its allocator
    static_assert(is_same<typename _Alloc::value_type, value_type>::value,
```

解决方法：将/home/pi/Pi-aSLAM/ORB_SLAM2-master/include/LoopClosing.h中第49-50行

```
typedef map<KeyFrame*,g2o::Sim3,std::less<KeyFrame*>,
        Eigen::aligned_allocator<std::pair<const KeyFrame*, g2o::Sim3> > >
KeyFrameAndPose;
```

修改为

```
typedef map<KeyFrame* const,g2o::Sim3,std::less<KeyFrame*>,
        Eigen::aligned_allocator<std::pair<KeyFrame* const, g2o::Sim3> > >
KeyFrameAndPose;
```

2.3.6 收尾工作

要把交换区大小及时改回来，否则flash闪存卡寿命很容易迅速到期。

```
sudo nano /etc/dphys-swapfile
```

编辑 CONF_SWAPSIZE 变量：从2048恢复到100

重新启动交换服务:

```
sudo /etc/init.d/dphys-swapfile stop
sudo /etc/init.d/dphys-swapfile start
```

2.4 摄像头调试

2.4.1 系统配置

```
sudo raspi-config
```

3 Interface config 找到Camera，打开

设置完成需要重启生效。

2.4.2 摄像头测试

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

如果出现video0设备就是正常的。

```
vcgencmd get_camera
```

如果support和detected都是1就说明正常。

```
raspistill -o test_camera.jpg -t 500
```

如果照片保存成功就说明摄像头正常。-t参数是拍照延时，单位毫秒，默认值好像是5000.

[BUG.4] 找不到摄像头

```
mmal: Cannot read camera info, keeping the defaults for OV5647
mmal: mmal_vc_component_create: failed to create component 'vc.ril.camera'
(1:ENOMEM)
mmal: mmal_component_create_core: could not create component 'vc.ril.camera' (1)
mmal: Failed to create camera component
mmal: main: Failed to create camera component
mmal: Camera is not detected. Please check carefully the camera module is
installed correctly
```

解决方法：检查一下接线是否松动，如果松动就重新接线，然后重启树莓派，再次尝试就可以了。

2.5 麦克纳姆轮测试

2.5.1 更新wiringPi

由于官方系统自带的wiringPi目前不支持树莓派4B，需要手动更新。

先查看树莓派 gpio 版本

```
gpio -v
```

如果是2.50就需要更新

```
wget https://project-downloads.drogon.net/wiringpi-latest.deb
sudo dpkg -i wiringpi-latest.deb
```

再次检查gpio版本，如果是2.52就可以了。

2.5.2 工作台准备

为准备车轮测试，需要**将小车抬高，使四轮悬空**。否则小车可能在通电一瞬间运动起来，冲出桌面，产生结构损坏。

2.5.3 电源切换

接下来的测试需要将电源从外接电源**切换至车载电源**。这是因为需要运动时，树莓派和小车驱动电源部分需要共地，外接电源无法保证共地（我自己搭的车载电源电路是共地的）。

安装车载电池时，一定要确保**总电源开关处于关闭状态**，否则可能产生电火花或意料之外的情况。

电池安装完毕后，先**用万用表测试电压**是否在标准范围内（10.5-12V）。若电压过高或过低有可能损坏树莓派。如果电压不足，需要充电。

测试无误后，先**不接树莓派电源**，尝试开启总开关，观察各电路板状态以及车轮是否会异常转动。正常情况下，上面板的降压和下面板的两个PWM驱动都应该亮红灯，表示电源正常。此外，如果车轮高速转动或有异响，可能说明电路有问题。

2.5.4 代码解释

利用wiringPi的API，我们可以通过代码操作树莓派的GPIO端口。

首先，需要用pinMode函数指定GPIO是以PWM方式输出还是数字输出（即高低电平）

```
pinMode(GPIO_move_direction_BR_b, OUTPUT);
pinMode(GPIO_pwm_front_left, PWM_OUTPUT);
```

如果使用PWM，还需要使用softPWMCreate函数指定PWM参数范围

```
softPwmCreate(GPIO_pwm_front_left, 0, 100);
```

对于高低电平的写入，使用 digitalWrite 函数

```
digitalWrite(GPIO_move_direction_FL_a, LOW);
```

而PWM使用 softPWMWrite 函数

```
softPwmWrite(GPIO_pwm_back_left, 0);
```

2.5.5 测试

使用./src/test/move中的代码。

```
make
sudo ./test_wheels
```

此处执行必须要使用superuser身份，因为GPIO驱动需要系统调用，需要较高权限。

正常情况下，每个轮子都应该朝前转动。如果方向有问题，那么有以下几种解决方法：

- 更改驱动代码
- 重新连接驱动板的方向控制线
- 重新连接电机到驱动板的线

2.6 麦克纳姆轮驱动

下面简单介绍一下车轮驱动的原理。

2.6.1 L298N芯片

小车的车轮连着电机。我们需要让电机按照我们希望的速度和方向旋转，就需要通过PWM信号控制电机。电机不能直接接收PWM信号，需要一个芯片来进行翻译，这个芯片就是L298N。

L298N芯片自带PCB板，板上接的线分为电源和信号。电源的输入要求5-46V直流电，一般使用12V。输出是两组，可接两个马达。由于是PWM，不分正负，只有正转和反转的区别。

信号部分，两边的两组是使能端。使能端就是PWM信号，通过高电平占比来控制转速。使能端默认有跳帽连5V信号，即占比100%。我们会使用使能端来控制转速。

信号余下的四个脚是逻辑输入，IN1/IN2控制A输出，IN3/IN4控制B输出。控制一组输出的两个脚，1/0和0/1分别是两个方向的转动（需要自己试一下正反）。1/1是刹车制动，0/0是不控制。

因此，每个轮子的控制需要两根方向+一根PWM，总共4*3=12根线。

2.6.2 异或门升压

树莓派GPIO端口输出的电平是3.3V，而驱动L298N需要5V。因此我们需要用5V的逻辑门来进行升压。实际中我使用了SN74HC86N(CMOS异或门)接地实现同门来进行升压。由于有12根输入线，一共需要12个非门，即3个

74HC86N芯片。

[BUG.5] 同或门不工作问题：有时候会遇到输入高电平输出为0V，这是因为电源电压太高（5.10V就会出问题），使得3.3V左右的输入高电平被误认为是低电平。此时调整电源电压即可。

2.6.3 麦克纳姆轮

麦克纳姆轮是一种可以让四轮车实现原地转弯和平移的轮子，但使用中需要改变轮子的旋转方向。

具体来说，前进时所有轮子均向前，后退时所有轮子均向后，和普通小车一样。在平移时，平移方向一侧轮子俯视向内对转，另一侧两轮俯视向外对转。原地旋转时 同侧轮方向相同，异侧轮方向相反。

2.6.4 GPIO端口

可以在树莓派中用以下命令查看GPIO信息：

```
gpio readall
```

端口信息：

```
+-----+-----+---------+------+---+---Pi 4B--+---+------+---------+-----+-----+
| BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
+-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
|     |     |    3.3v |      |   |  1 || 2  |   |      | 5v      |     |     |
|   2 |   8 |   SDA.1 |   IN | 1 |  3 || 4  |   |      | 5v      |     |     |
|   3 |   9 |   SCL.1 |   IN | 1 |  5 || 6  |   |      | 0v      |     |     |
|   4 |   7 | GPIO. 7 |   IN | 1 |  7 || 8  | 1 | IN   | TxD     | 15  | 14  |
|     |     |      0v |      |   |  9 || 10 | 1 | IN   | RxD     | 16  | 15  |
|  17 |   0 | GPIO. 0 |   IN | 0 | 11 || 12 | 0 | IN   | GPIO. 1 | 1   | 18  |
|  27 |   2 | GPIO. 2 |   IN | 0 | 13 || 14 |   |      | 0v      |     |     |
|  22 |   3 | GPIO. 3 |   IN | 0 | 15 || 16 | 0 | IN   | GPIO. 4 | 4   | 23  |
|     |     |    3.3v |      |   | 17 || 18 | 0 | IN   | GPIO. 5 | 5   | 24  |
|  10 |  12 |    MOSI |   IN | 0 | 19 || 20 |   |      | 0v      |     |     |
|   9 |  13 |    MISO |   IN | 0 | 21 || 22 | 0 | IN   | GPIO. 6 | 6   | 25  |
|  11 |  14 |    SCLK |   IN | 0 | 23 || 24 | 1 | IN   | CE0     | 10  | 8   |
|     |     |      0v |      |   | 25 || 26 | 1 | IN   | CE1     | 11  | 7   |
|   0 |  30 |   SDA.0 |   IN | 1 | 27 || 28 | 1 | IN   | SCL.0   | 31  | 1   |
|   5 |  21 | GPIO.21 |   IN | 1 | 29 || 30 |   |      | 0v      |     |     |
|   6 |  22 | GPIO.22 |   IN | 1 | 31 || 32 | 0 | IN   | GPIO.26 | 26  | 12  |
|  13 |  23 | GPIO.23 |   IN | 0 | 33 || 34 |   |      | 0v      |     |     |
|  19 |  24 | GPIO.24 |   IN | 0 | 35 || 36 | 0 | IN   | GPIO.27 | 27  | 16  |
|  26 |  25 | GPIO.25 |   IN | 0 | 37 || 38 | 0 | IN   | GPIO.28 | 28  | 20  |
|     |     |      0v |      |   | 39 || 40 | 0 | IN   | GPIO.29 | 29  | 21  |
+-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
| BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
+-----+-----+---------+------+---+---Pi 4B--+---+------+---------+-----+-----+
```

实际分配：

颜色	GPIO号	物理端口号	轮子	信号
orange	24	35	FL	PWM
yellow	25	37	FR	PWM
blue	27	36	BL	PWM
green	28	38	BR	PWM
purple	22	31	FL	D_f
gray	26	32	FL	D_b
purple	0	11	FR	D_f
gray	1	12	FR	D_b
white	2	13	BL	D_f
black	3	15	BL	D_b
brown	4	16	BR	D_f
red	5	18	BR	D_b

2.6.5 测试

使用./src/test/move中的代码。

```
make
sudo ./test_move
```

[BUG.6] 全车接通电源后，进行测试，后左轮和后右轮有时会在停止指令下转动，转动时有时无，后轮的L298N甚至一度发烫，判断为电路短路。经检查，是L298N和不锈钢车板之间绝缘未做好，导致触电短路。充分进行绝缘隔离后，问题解决。

3 ROS配置调试

3.1 安装ROS

ORB-SLAM2支持在线和非在线运行，如果需要在线运行必须接入ROS.

3.1.1 准备工作

首先，为ROS仓库创建仓库列表文件

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

添加密钥

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

显示imported:1，即说明添加成功。

接下来，进行包管理器和系统更新

```
sudo apt-get update
sudo apt-get upgrade
```

3.1.2 安装rosdep

首先安装依赖

```
sudo apt install -y python-rosdep python-rosinstall-generator python-wstool
python-rosinstall build-essential cmake
```

进行rosdep初始化之前，因为github服务器无法直连，需要将所有github的url修改为代理访问。 参考链接

```
sudo vim /usr/lib/python2.7/dist-packages/rosdep2/sources_list.py
第72行添加https://ghproxy.com/
第311行添加url="https://ghproxy.com/"+url

sudo vim /usr/lib/python2.7/dist-packages/rosdistro/__init__.py
第68行添加https://ghproxy.com/

sudo vim /usr/lib/python2.7/dist-packages/rosdep2/gbpdistro_support.py
第36行添加https://ghproxy.com/
第204行添加gbpdistro_url = "https://ghproxy.com/" + gbpdistro_url

sudo vim /usr/lib/python2.7/dist-packages/rosdep2/rep3.py
第39行添加https://ghproxy.com/

sudo vim /usr/lib/python2.7/dist-packages/rosdistro/manifest_provider/github.py
第68行添加https://ghproxy.com/
第119行添加https://ghproxy.com/
```

rosdep初始化

```
rosdep update # 注意此命令不要使用superuser权限
```

若显示"updated cache in ..."则说明初始化成功。如果一次不成功可以多试几次。

3.1.3 ROS包下载

创建catkin工作空间

```
mkdir -p ~/ros_catkin_ws
cd ~/ros_catkin_ws
```

这里我们安装Desktop版。除了ROS核心库以外，还包含可视化工具rqt，rviz，以及robot-generic库

```
rosinstall_generator desktop --rosdistro melodic --deps --wet-only --tar >
melodic-desktop-wet.rosinstall
```

如果没有输出说明正常. 此时的melodic-desktop-wet.rosinstall就是包含所有软件包源码地址的目录。为了方便下载，我们需要手动将所有地址更改为使用代理，即

```
https://github.com/xxx/xxx
```

改为

```
https://ghproxy.com/https://github.com/xxx/xxx
```

这一步可以自动替换完成。完成后，从该目录进行init

```
wstool init src melodic-desktop-wet.rosinstall
```

如果出现update complete.说明下载完成。配置正确的话该过程耗时不会超过5分钟。

接下来，安装依赖包

```
cd ~/ros_catkin_ws
rosdep install -y --from-paths src --ignore-src --rosdistro melodic -r --
os=debian:buster
```

如果正确配置镜像源，该过程耗时约30分钟，需要耐心等待。如果出现

```
#All required rosdeps installed successfully
```

说明安装完成。

3.1.4 ROS编译

在编译之前，可以扩展一下交换区，避免内存不足。

```
sudo nano /etc/dphys-swapfile
```

然后编辑 CONF_SWAPSIZE 变量：从100增加到2048

重新启动交换服务:

```
sudo /etc/init.d/dphys-swapfile stop
sudo /etc/init.d/dphys-swapfile start
```

开始编译！

```
sudo ./src/catkin/bin/catkin_make_isolated --install -DCMAKE_BUILD_TYPE=Release --
install-space /opt/ros/melodic
```

该过程需要1-2h左右，需要耐心等待。

编译成功后，添加ROS环境变量至bash。

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

一定要记着把交换区设置改回来

```
sudo nano /etc/dphys-swapfile
```

然后编辑 CONF_SWAPSIZE 变量：从2048减小到100

重新启动交换服务:

```
sudo /etc/init.d/dphys-swapfile stop
sudo /etc/init.d/dphys-swapfile start
```

最后，尝试运行ros_core和rviz，确认安装成功。

```
roscore
rosrun rviz rviz
```

编译完成后，ros_catkin_ws下的build_isolated和devel_isolated都可以删掉，占用空间太多。

3.2 ROS基本知识

3.2.1 catkin工作空间

Catkin工作空间是创建、修改、编译catkin软件包的目录。catkin的工作空间，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。

catkin的结构十分清晰，它包括了src、build、devel三个路径，在有些编译选项下也可能包括其他。但这三个文件夹是catkin编译系统默认的。它们的具体作用如下：

- src/: ROS的catkin软件包（源代码包）
- build/: catkin（CMake）的缓存信息和中间文件
- devel/: 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

3.2.2 软件包

Package不仅是Linux上的软件包，也是catkin编译得基本单元，我们使用 catkin_make 编译的对象就是每 个ROS的package。

```
+-- PACKAGE
    +-- CMakeLists.txt # package的编译规则（必须）
    +-- package.xml # package的描述信息（必须）
    +-- src/ # 源代码
    +-- include/ # 头文件
    +-- scripts/ # 可执行脚本
    +-- msg/ # 自定义消息
    +-- srv/ # 自定义服务
    +-- models/ # 3D模型文件
    +-- urdf/
    +-- launch/
```

- **CMakeLists.txt**: 定义package的包名、依赖、源文件、目标文件等编译规则，是package不可少的成 分
- **package.xml**: 描述package的包名、版本号、作者、依赖等信息，是package不可少的成分
- src/: 存放ROS的源代码，包括C++的源码和(.cpp)以及Python的module(.py)
- include/: 存放C++源码对应的头文件
- scripts/: 存放可执行脚本，例如shell脚本(.sh)、Python脚本(.py)
- msg/: 存放自定义格式的消息(.msg)
- srv/: 存放自定义格式的服务(.srv)

- models/: 存放机器人或仿真场景的3D模型(.sda, .stl, .dae等)
- urdf/: 存放机器人的模型描述(.urdf或.xacro)
- launch/: 存放launch文件(.launch或.xml)

开发自己的软件包：

```
catkin_create_pkg "packageName" "depends"
```

自动生成的结构：

```
+-- test_pkg
    --- CMakeLists.txt
    +-- include
    +-- test_pkg
    --- package.xml
    +-- src
```

常用的depends软件包：

```
std_msgs 标准message（见3.2.3）
rospy python语言接口
roscpp C++语言接口
```

cmake常用命令

```
cmake_minimum_required() #CMake的版本号
project() #项目名称
find_package() #找到编译需要的其他CMake/Catkin package
catkin_python_setup() #catkin新加宏，打开catkin的Python Module的支持
add_message_files() #catkin新加宏，添加自定义Message/Service/Action文件
add_service_files()
add_action_files()
generate_message() #catkin新加宏，生成不同语言版本的msg/srv/action接口
catkin_package() #catkin新加宏，生成当前package的cmake配置，供依赖本包的其他软件包调用
add_library() #生成库
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() #链接
catkin_add_gtest() #catkin新加宏，生成测试
install() #安装至本机
```

package.xml 常用标签

```
<pacakge> 根标记文件
<name> 包名
<version> 版本号
<description> 内容描述
<maintainer> 维护者信息
<license> 软件许可证
<buildtool_depend> 编译构建工具，通常为catkin
<build_depend> 编译依赖项，与Catkin中的
<run_depend> 运行依赖项
```

参考链接：ros_wiki/package.xml

3.2.3 ROS通信架构

node

在ROS中，一个进程就是一个节点（Node），节点由Master统一管理。命令roscore就是启动Master的命令。

启动node的命令是

```
rosrun "pkg_name" "node_name"
```

如果node太多，就需要用.launch文件（本质是xml）来启动。

```
roslaunch "pkg_name" "file_name.launch"
```

launch文件的常用标签是

```
<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> <!--启动yaml文件参数到参数服务器-->
<arg> <!--定义变量-->
<remap> <!--设定参数映射-->
<group> <!--设定命名空间-->
</launch> <!--根标签-->
```

使用命令rqt_graph可以查看节点间的通信结构

```
rqt_graph
```

使用命令rosnode可以对node进行操作

```
rosnode list # 查看所有node
rosnode kill [node_name] 杀掉一个node
```

topic

ROS中的topic是一套node间通信的机制。使用topic通信必须遵照指定的格式。topic是单向通信机制，发出方称为publisher，接受方称为subscriber。常用的topic相关命令有：

```
rostopic list 列出当前所有的topic
rostopic info "topic_name" 显示某个topic的属性信息，可查看publisher和subscriber等
rostopic echo "topic_name" 显示某个topic的内容
rostopic pub "topic_name" "value" ... 向某个topic发布内容
```

topic通信需要遵守预定义好的格式，这种格式被称为message(msg)。message的定义常放在.msg文件中。

service

service是一种临时双向通信机制，它不仅可以发送消息，同时还会有反馈。所以 service 包括两部分，一部分是请求方(Clinet)，另一部分是应答方/服务提供方(Server)。当需要某个数据时，Client就会发送一个request，等待server处理反馈回一个reply，这样通过类似"request-reply"的机制完成整个服务通信。

常用的service命令：

```
rosservice list 显示服务列表
rosservice info service信息，可以查看server
rosservice type 打印服务类别
rosservice find 按服务类别查找服务
rosservice call 使用所提供的args调用服务
rosservice args 打印服务参数
```

action

action 是一种异步响应的通信方式，主要补充了 service 同步通信的不 足。因为service 中采用同步通信机制，客户端的程序需要等到服务器响应后才能运行其他程序。一个action也有client和server，client是动作发起者，server是执行者。

利用action进行请求响应，action的内容格式应包含三个部分，目标、反馈、结果。

- 目标
  - client要求server做的事情
- 反馈
  - server给client的实时反馈
- 结果
  - server给client的最终反馈

action的规范文件格式如下：

```
uint32 dishwasher_id # Specify which dishwasher we want to use
---
uint32 total_dishes_cleaned
---
float32 percent_complete # Define feedback message
```

要使用action，在CMakeLists.txt 里应该有如下的内容：

```
File: CMakeLists.txt
...
find_package(catkin REQUIRED genmsg actionlib_msgs actionlib)
...
add_action_files(DIRECTORY action FILES DoDishes.action)
generate_message(DEPENDENCIES actionlib_msgs)
add_action_files(DIRECTORY action FILES Handling.action)
generate_message(DEPENDENCIES actionlib_msgs)
```

3.2.4 topic通信例子

tutorials/src/talker.cpp

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

int main(int argc, char **argv)
{

    //必须在最开始调用ros::init()。ros::init() 函数需要接受argc和argv。
    ros::init(argc, argv, "talker");

    // NodeHandle是node和ros通信的入口，第一个NodeHandle的构造会完整初始化node，
    // 最后一个NodeHandle的析构会关闭node。
    ros::NodeHandle n;

    /**
     * advertise() 函数用于建立一个topic。它返回一个publisher，可以调用
```

```
     * 其publish()方法进行具体的通信。如果所有publisher和其copy都被析构，
     * 那么该topic会结束。 参数1是topic的名字，参数2是缓冲区的大小。
     */
    ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);

    // ros::Rate ros自带计时器，10代表10Hz
    ros::Rate loop_rate(10);
    int count = 0;

    // ros::ok ???
    while (ros::ok())
    {
        //msg object
        std_msgs::String msg;
        std::stringstream ss;
        ss << "hello world " << count;
        msg.data = ss.str();
        // ROS 控制台回显
        ROS_INFO("%s", msg.data.c_str());
        // 调用 publish()
        chatter_pub.publish(msg);
        // 调用topic的回调函数
        ros::spinOnce();
        // 计时器沉默一阵子
        loop_rate.sleep();
        ++count;
    }
    return 0;
}
```

tutorials/src/listener.cpp

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

int main(int argc, char **argv)
{

    //必须在最开始调用ros::init()。ros::init() 函数需要接受argc和argv。
    ros::init(argc, argv, "talker");

    // NodeHandle是node和ros通信的入口，第一个NodeHandle的构造会完整初始化node，
    // 最后一个NodeHandle的析构会关闭node。
    ros::NodeHandle n;

    /**
     * 用advertise() 建立一个topic并返回一个publisher。可以调用
     * 其publish()方法进行具体的通信。如果所有publisher和其copy都被析构，
```

```
     * 那么该topic会结束。 参数1是topic的名字，参数2是缓冲区的大小。
     */
    ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);

    // ros::Rate ros自带计时器，10代表10Hz
    ros::Rate loop_rate(10);
    int count = 0;

    // ros::ok ???
    while (ros::ok())
    {
        //msg object
        std_msgs::String msg;
        std::stringstream ss;
        ss << "hello world " << count;
        msg.data = ss.str();
        // ROS 控制台回显
        ROS_INFO("%s", msg.data.c_str());
        // 调用 publish()
        chatter_pub.publish(msg);
        // 调用topic的回调函数
        ros::spinOnce();
        // 计时器沉默一阵子
        loop_rate.sleep();
        ++count;
    }
    return 0;
}
```

tutorials/CMakeLists.txt

```
cmake_minimum_required(VERSION 3.0.2)
project(tutorials)

## Find catkin and any catkin packages
find_package(catkin REQUIRED COMPONENTS
 roscpp
 rospy
 std_msgs
 genmsg
)
## Generate added messages and services
generate_messages(DEPENDENCIES std_msgs)

## Declare a catkin package
catkin_package()

## Build talker and listener
include_directories(include ${catkin_INCLUDE_DIRS})

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

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

add_executable(listener src/listener.cpp)
target_link_libraries(listener ${catkin_LIBRARIES})
add_dependencies(listener tutorials_generate_messages_cpp)
```

tutorials/package.xml

```
<?xml version="1.0"?>
    <package format="2">
    <name>tutorials</name>
    <version>0.0.0</version>
    <description>The tutorials package</description>
    <maintainer email="weixr18@mails.tsinghua.edu.cn">Beiming</maintainer>
    <license>TODO</license>
    <buildtool_depend>catkin</buildtool_depend>

    <build_depend>roscpp</build_depend>
    <build_depend>rospy</build_depend>
    <build_depend>std_msgs</build_depend>

    <build_export_depend>roscpp</build_export_depend>
    <build_export_depend>rospy</build_export_depend>
    <build_export_depend>std_msgs</build_export_depend>

    <exec_depend>roscpp</exec_depend>
    <exec_depend>rospy</exec_depend>
    <exec_depend>std_msgs</exec_depend>
</package>
```

编译

```
cd ~/ros_catkin_ws
catkin_make
```

[BUG.7] Permission denied

如果出现以下bug

```
[Errno 13] Permission denied: '/home/pi/ros_catkin_ws/build/.built_by'
```

就需要更改一下catkin_ws的用户权限，把owner从root改为当前用户。

```
sudo chown $USER: -R ~/ros_catkin_ws
```

启动

在三个终端分别执行

```
roscore
rosrun tutorials listener
rosrun tutorials talker
```

3.2.5 ROS控制小车运动

使用dumbpi文件夹下的代码。

注意：先使用**外接电源**（即不给小车电机供电），测试两个节点可以正常通信后，再切换到车载电源。

首先，将dumbpi文件夹放到ros_catkin_ws/src下。进入上级文件夹，开始编译

```
cd ~/ros_catkin_ws
catkin_make
```

编译完成后，首先在一个窗口中**启动roscore**。

```
roscore
```

打开另一个窗口，**启动节点**

```
bash src/dumbpi/test_move.sh
```

如果出现如下输出，说明正常

```
dumbpi_controller
Starting up ...
dumbpi_keyboard
```

然后，可以通过w/s/d/a/j/k来控制小车。具体见代码dombpi/src/controller.cpp。

代码逻辑：keyboard每100ms接收一个字符并发送给controller，controller根据字符指令改变运动方向。全程全部PWM信号保持恒定可驱动的占空比不变。

3.2.6 ROS相关BUG

[BUG.8] 报错 找不到包

```
[rospack] Error: package 'dumbpi' not found
```

解决方法：查看环境变量ROS_PACKAGE_PATH是否有workspace下的src目录

```
echo $ROS_PACKAGE_PATH
```

如果没有，添加

```
export ROS_PACKAGE_PATH=${ROS_PACKAGE_PATH}:~/ros_catkin_ws/src
```

[BUG.9] 报错 找不到可执行文件

解决方法：确保~/.bashrc里有如下语句

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

添加后需要source或者新开一个bash

```
source ~/.bashrc
```

[BUG.10] 多个ROS core

```
killall -9 roscore
killall -9 rosmaster
```

3.3 ROS + ORB-SLAM2

3.3.1 ORB-SLAM2的ROS模块编译

ORB-SLAM2和ROS接口的部分是单独的一个模块，需要单独编译。

首先，将ORB-SLAM2的包目录加到ROS包目录中（建议写入~/.bashrc）

```
export ROS_PACKAGE_PATH=${ROS_PACKAGE_PATH}:ORB_SLAM_PATH/ORB_SLAM2/Examples/ROS
```

在编译之前，**必须扩展交换区**，否则会出现内存不足。

```
sudo nano /etc/dphys-swapfile
```

然后编辑 CONF_SWAPSIZE 变量：从100增加到2048

重新启动交换服务:

```
sudo /etc/init.d/dphys-swapfile stop
sudo /etc/init.d/dphys-swapfile start
```

进行编译！

```
chmod +x build_ros.sh
./build_ros.sh
```

如果没有报错就可以了，这个过程很快，10分钟以内。

设置环境变量

```
source ORB_SLAM2/Examples/ROS/ORB_SLAM2/build/devel/setup.bash
source ~/.bashrc
```

最后，要记得将交换区改回来

```
sudo nano /etc/dphys-swapfile
```

然后编辑 CONF_SWAPSIZE 变量：从2048减小到100

重新启动交换服务:

```
sudo /etc/init.d/dphys-swapfile stop
sudo /etc/init.d/dphys-swapfile start
```

3.3.2 安装usb_cam包

ROS连接树莓派相机需要先安装usb_cam包。在安装这个包之前，需要手动安装依赖camera_info_manager。根据官网提示，需要先去下载源码

```
https://github.com/ros-perception/image_common/tree/hydro-devel
```

注意分支是hydro-devel不要搞错。把项目里camera_info_manager文件夹放到ros_catkin_ws/src下。

然后下载usb_cam的源码

```
https://github.com/ros-drivers/usb_cam/tree/master
```

注意是master而不是默认的develop分支，代码也放到ros_catkin_ws/src下。

进入工作空间，开始编译

```
cd ~/ros_catkin_ws
catkin_make
```

[BUG.11] PIX_FMT_RGB24等符号无定义

只要把这些变量从PIX_FMT_RGBXX变成AV_PIX_FMT_RGBXX即可。

[BUG.12] avcodec_alloc_frame 无定义

改为av_frame_alloc即可。

看到

```
[100%] Built target usb_cam_node
```

就说明编译完成啦！

3.3.3 ROS连接树莓派相机

要先安装一下image-view包来测试。首先，还是找github地址

```
https://github.com/ros-perception/image_pipeline/tree/indigo
```

下载源码后，找到里面的image_view文件夹，放到ros_catkin_ws/src下。

进入工作空间，开始编译

```
cd ~/ros_catkin_ws
catkin_make
```

看到

```
[100%] Built target stereo_view
```

就说明编译成功。

直接运行usb_cam提供的测试

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

如果摄像头是正常挂载的，就应该能从屏幕上看到摄像头的实时图像了！

3.3.4 修改订阅话题名

在调试SLAM之前，需要先修改一个订阅话题名。找到ros_mono.cc

```
vim ~/Pi-aSLAM/ORB_SLAM2-master/Examples/ROS/ORB_SLAM2/src/ros_mono.cc
```

找到第64行

```
ros::Subscriber sub = nodeHandler.subscribe("/camera/image_raw", 1,
&ImageGrabber::GrabImage,&igb);
```

修改为

```
ros::Subscriber sub = nodeHandler.subscribe("/usb_cam/image_raw", 1,
&ImageGrabber::GrabImage,&igb);
```

然后重新编译ORB_SLAM2。在编译之前，**必须扩展交换区**，否则会出现内存不足。

```
sudo nano /etc/dphys-swapfile
```

然后编辑 CONF_SWAPSIZE 变量：从100增加到2048

重新启动交换服务:

```
sudo /etc/init.d/dphys-swapfile stop
sudo /etc/init.d/dphys-swapfile start
```

开始编译：

```
cd ~/Pi-aSLAM/ORB_SLAM2-master/
./build_ros.sh
```

将交换区改回原来大小

```
sudo nano /etc/dphys-swapfile
```

然后编辑 CONF_SWAPSIZE 变量：2048减小到100

重新启动交换服务:

```
sudo /etc/init.d/dphys-swapfile stop
sudo /etc/init.d/dphys-swapfile start
```

3.3.5 ROS+实时单目SLAM调试

终于，来到了见证奇迹的时刻！

首先打开第一个终端，运行usb_cam模块

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

第二个终端，运行ORB-SLAM2

```
cd ~/Pi-aSLAM/ORB_SLAM2-master/
rosrun ORB_SLAM2 Mono Vocabulary/ORBvoc.txt Examples/ROS/ORB_SLAM2/Asus.yaml
```

Vocabulary的加载会持续一段时间（140多MB）..... 经过一段时间的等待，ORB-SLAM2的两个图形界面会出现在你的屏幕上。小心的把摄像头对准一个纹理丰富的区域，然后慢慢开始移动，SLAM就会开始了！