

Part1

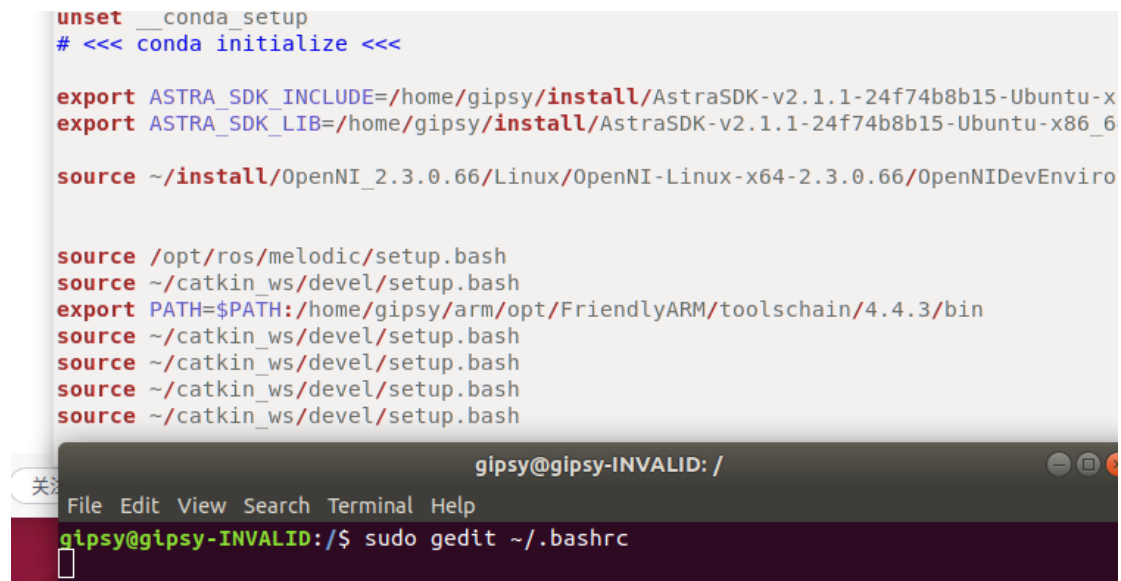
2.熟悉 Linux

(a) 如何在 Ubuntu 中安装软件（命令行界面）？它们通常被安装在什么地方？

Ubuntu 下用命令行界面安装软件使用的命令为：sudo apt-get install + 安装包名称，这里 sudo 是安装软件时添加的 root 用户权限。

安装位置：下载的软件的存放位置：/var/cache/apt/archives，安装后软件的默认位置：/usr/share。

(b) linux 的环境变量是什么？我如何定义新的环境变量？



```
unset __conda_setup
# <<< conda initialize <<<

export ASTRA_SDK_INCLUDE=/home/gipsy/install/AstraSDK-v2.1.1-24f74b8b15-Ubuntu-x
export ASTRA_SDK_LIB=/home/gipsy/install/AstraSDK-v2.1.1-24f74b8b15-Ubuntu-x86_6
source ~/install/OpenNI_2.3.0.66/Linux/OpenNI-Linux-x64-2.3.0.66/OpenNIDevEnviro

source /opt/ros/melodic/setup.bash
source ~/catkin_ws/devel/setup.bash
export PATH=$PATH:/home/gipsy/arm/opt/FriendlyARM/toolschain/4.4.3/bin
source ~/catkin_ws/devel/setup.bash
source ~/catkin_ws/devel/setup.bash
source ~/catkin_ws/devel/setup.bash
source ~/catkin_ws/devel/setup.bash
```

The image shows a terminal window with a menu bar (File, Edit, View, Search, Terminal, Help) and a title bar (gipsy@gipsy-INVALID: /). The terminal content displays a series of commands to set up environment variables, including unsetting __conda_setup, initializing conda, exporting ASTRA_SDK_INCLUDE and ASTRA_SDK_LIB, sourcing OpenNI environment variables, and sourcing ROS and catkin workspace setup files. The prompt at the bottom is gipsy@gipsy-INVALID:/\$ and the command being entered is sudo gedit ~/.bashrc.

环境变量如上所示，添加新的环境变量使用 export 命令；

(c) linux 根目录下面的目录结构是什么样的？至少说出 3 个目录的用途。

这里在根目录下使用 ls 查看结构如下图所示：

```
gipsy@gipsy-INVALID: /
File Edit View Search Terminal Help
gipsy@gipsy-INVALID:/$ tree -L 1
.
├── bin
├── boot
├── cdrom
├── dev
├── etc
├── home
├── initrd.img -> boot/initrd.img-5.4.0-73-generic
├── initrd.img.old -> boot/initrd.img-5.4.0-72-generic
├── lib
├── lib32
├── lib64
├── lost+found
├── media
├── mnt
├── opt
├── proc
├── root
├── run
├── sbin
├── snap
├── srv
├── sys
├── tmp
├── usr
├── var
├── vmlinuz -> boot/vmlinuz-5.4.0-73-generic
└── vmlinuz.old -> boot/vmlinuz-5.4.0-72-generic

23 directories, 4 files
gipsy@gipsy-INVALID:/$
```

其中，部分目录用途如下：

- /home：用户主目录
- /var：系统日志文件
- /dev：设备文件
- /bin：系统二进制可执行文件
- /media：媒体文件

(d) 假设我要给 a.sh 加上可执行权限，该输入什么命令？

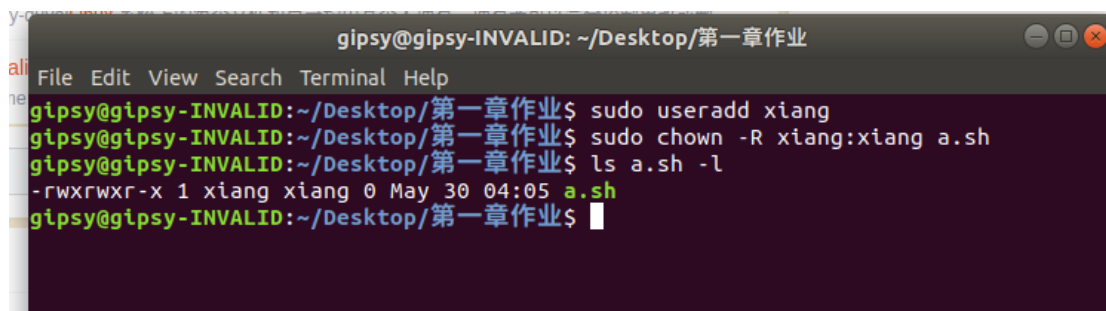
这里其实有两种方式，具体命令如下：

sudo chmod a+x a.sh 或者 chmod +x a.sh #因为题目没说明给哪个用

户增加可执行权限，这个给所有用户（a 代表 all）添加可执行权限。

`sudo chmod 777 a.sh` #这里直接给 a.sh 访问方式全加上权限。

(e) 假设我要将 a.sh 文件的所有者改成 xiang:xiang, 该输入什么命令?



```
gipsy@gipsy-INVALID: ~/Desktop/第一章作业
File Edit View Search Terminal Help
gipsy@gipsy-INVALID:~/Desktop/第一章作业$ sudo useradd xiang
gipsy@gipsy-INVALID:~/Desktop/第一章作业$ sudo chown -R xiang:xiang a.sh
gipsy@gipsy-INVALID:~/Desktop/第一章作业$ ls a.sh -l
-rwxrwxr-x 1 xiang xiang 0 May 30 04:05 a.sh
gipsy@gipsy-INVALID:~/Desktop/第一章作业$
```

3.SLAM 综述文献阅读

(a) SLAM 会在哪些场合中用到? 至少列举三个方向。

SLAM 在很多场合都有用到, 这里列举几个常见的方向:

VR、自动驾驶、室内定位、手持设备定位、三维重建。

(b) SLAM 中定位与建图是什么关系? 为什么在定位的同时需要建图?

在 SLAM 问题中, 定位和建图是互相关联的关系, 在 SLAM 框架中, 定位即相机的 pose 需要通过环境中的点计算得到, 我们想要让相机的 pose 更精确, 就要让环境中计算的 3D 点更精确, 这些 3D 点正是地图信息。这也是 SLAM 问题的核心处理方式 (同时定位与地图构建)。引用文献的回答就是: 为了在环境中精确地本地化, 正确的地图是必要的, 但为了构建一个好的地图, 当元素被添加到地图中时, 就必须正确地本地化。

(c) SLAM 发展历史如何? 我们可以将它划分成哪几个阶段?

SLAM 是近 30 年新提出来的研究方向, 发展历史如下几个阶段:

根据文献描述, 分为以上几个阶段:

- 1985-1990：有学者提出了同时完成定位和建图两个部分，进而引出 SLAM 的概念；
- 21 世纪之后，SLAM 研究者开始借鉴 SFM 的方法，引入 BA 进行优化；
- 近来，随着 ML 和 DL 的发展，研究者都在使用这些技术融入 SLAM 系统，提高 SLAM 系统的效率；

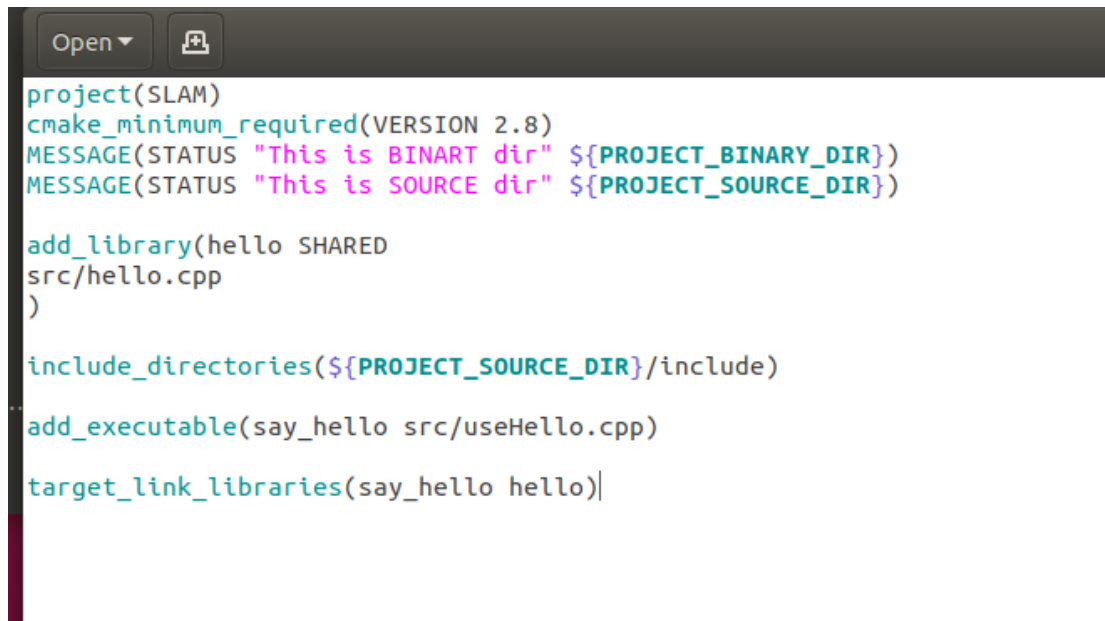
(d) 列举三篇在 SLAM 领域的经典文献。

SLAM 领域经典文献比较多，这里列举几个直接提出框架的文献：

- ORBSLAM 系列 (1,2,3)：a versatile and accurate monocular SLAM system
- VINS-Mono：VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator
- DSO：Direct Sparse Odometry
- MSCKF：A Multi-State Constraint Kalman Filter for Vision-aided Inertial Navigation
- MonoSLAM：MonoSLAM: Real-Time Single Camera SLAM
- PTAM：PTAM-Parallel Tracking and Mapping for Small AR Workspace
- LSD-SLAM：LSD-SLAM: Large-Scale Direct Monocular SLAM

4.CMake 练习

Cmakelists.txt 文件如下所示：



```
project(SLAM)
cmake_minimum_required(VERSION 2.8)
MESSAGE(STATUS "This is BINART dir" ${PROJECT_BINARY_DIR})
MESSAGE(STATUS "This is SOURCE dir" ${PROJECT_SOURCE_DIR})

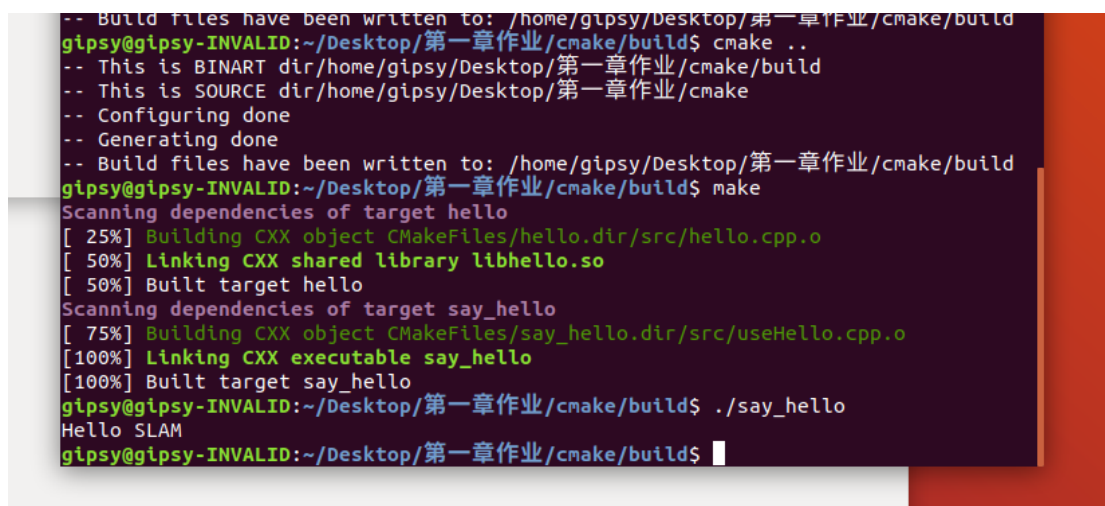
add_library(hello SHARED
src/hello.cpp
)

include_directories(${PROJECT_SOURCE_DIR}/include)

add_executable(say_hello src/useHello.cpp)

target_link_libraries(say_hello hello)|
```

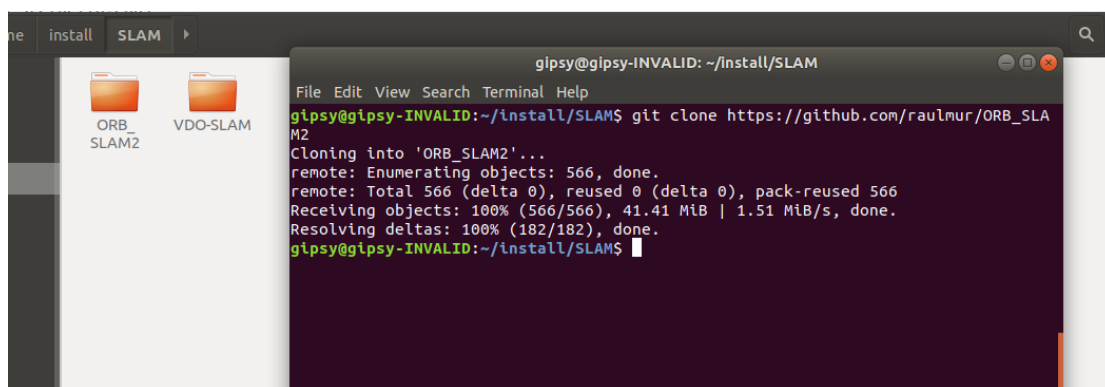
执行情况：



```
-- Build files have been written to: /home/gipsy/Desktop/第一章作业/cmake/build
gipsy@gipsy-INVALID:~/Desktop/第一章作业/cmake/build$ cmake ..
-- This is BINART dir/home/gipsy/Desktop/第一章作业/cmake/build
-- This is SOURCE dir/home/gipsy/Desktop/第一章作业/cmake
-- Configuring done
-- Generating done
-- Build files have been written to: /home/gipsy/Desktop/第一章作业/cmake/build
gipsy@gipsy-INVALID:~/Desktop/第一章作业/cmake/build$ make
Scanning dependencies of target hello
[ 25%] Building CXX object CMakeFiles/hello.dir/src/hello.cpp.o
[ 50%] Linking CXX shared library libhello.so
[ 50%] Built target hello
Scanning dependencies of target say_hello
[ 75%] Building CXX object CMakeFiles/say_hello.dir/src/useHello.cpp.o
[100%] Linking CXX executable say_hello
[100%] Built target say_hello
gipsy@gipsy-INVALID:~/Desktop/第一章作业/cmake/build$ ./say_hello
Hello SLAM
gipsy@gipsy-INVALID:~/Desktop/第一章作业/cmake/build$
```

5.理解 ORB-SLAM2 框架

下载 ORB-SLAM2:

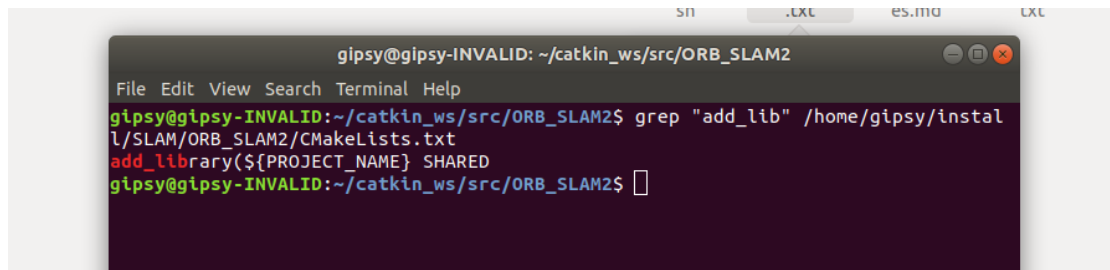


```
File Edit View Search Terminal Help
gipsy@gipsy-INVALID: ~/install/SLAM
gipsy@gipsy-INVALID:~/install/SLAM$ git clone https://github.com/raulmur/ORB_SLAM2
Cloning into 'ORB_SLAM2'...
remote: Enumerating objects: 566, done.
remote: Total 566 (delta 0), reused 0 (delta 0), pack-reused 566
Receiving objects: 100% (566/566), 41.41 MiB | 1.51 MiB/s, done.
Resolving deltas: 100% (182/182), done.
gipsy@gipsy-INVALID:~/install/SLAM$
```

关于 ORB-SLAM2 的 CmakeLists.txt:

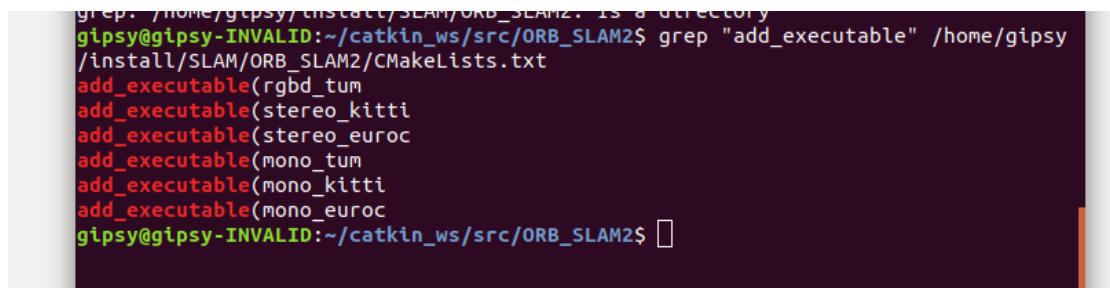
(a) ORB-SLAM2 将编译出什么结果？有几个库文件和可执行文件？

库文件：一个



```
gipsy@gipsy-INVALID: ~/catkin_ws/src/ORB_SLAM2
File Edit View Search Terminal Help
gipsy@gipsy-INVALID:~/catkin_ws/src/ORB_SLAM2$ grep "add_lib" /home/gipsy/install/SLAM/ORB_SLAM2/CMakeLists.txt
add_library(${PROJECT_NAME} SHARED
gipsy@gipsy-INVALID:~/catkin_ws/src/ORB_SLAM2$
```

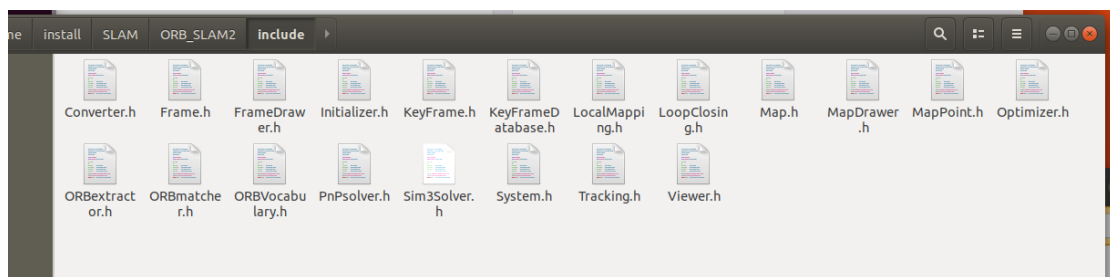
可执行文件：



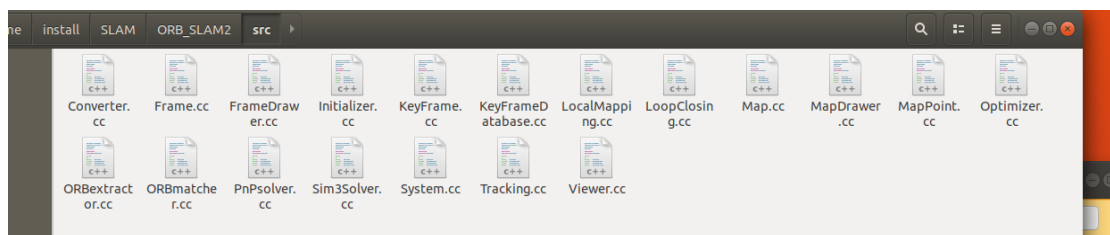
```
gipsy@gipsy-INVALID:~/catkin_ws/src/ORB_SLAM2$ grep "add_executable" /home/gipsy/install/SLAM/ORB_SLAM2/CMakeLists.txt
add_executable(rgbd_tum
add_executable(stereo_kitti
add_executable(stereo_euroc
add_executable(mono_tum
add_executable(mono_kitti
add_executable(mono_euroc
gipsy@gipsy-INVALID:~/catkin_ws/src/ORB_SLAM2$
```

(b) ORB-SLAM2 中的 include, src, Examples 三个文件夹中都含有什么内容？

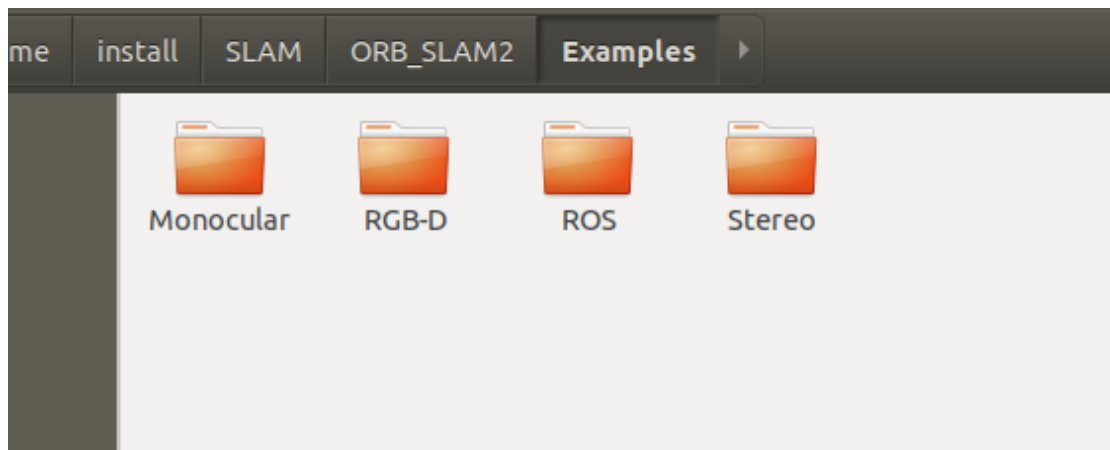
Include：这里里面是一些源文件对应的头文件



Src：这里面是一些源文件



Examples：这里面是单目、双目、rgbd 和 ROS 各个可执行文件以及一些配置文件；



(c) ORB-SLAM2 中的可执行文件链接到了哪些库？它们的名字是什么？

```
target_link_libraries(${PROJECT_NAME}
${OpenCV_LIBS}
${EIGEN3_LIBS}
${Pangolin_LIBRARIES}
${PROJECT_SOURCE_DIR}/Thirdparty/DBoW2/lib/libDBoW2.so
${PROJECT_SOURCE_DIR}/Thirdparty/g2o/lib/libg2o.so
)

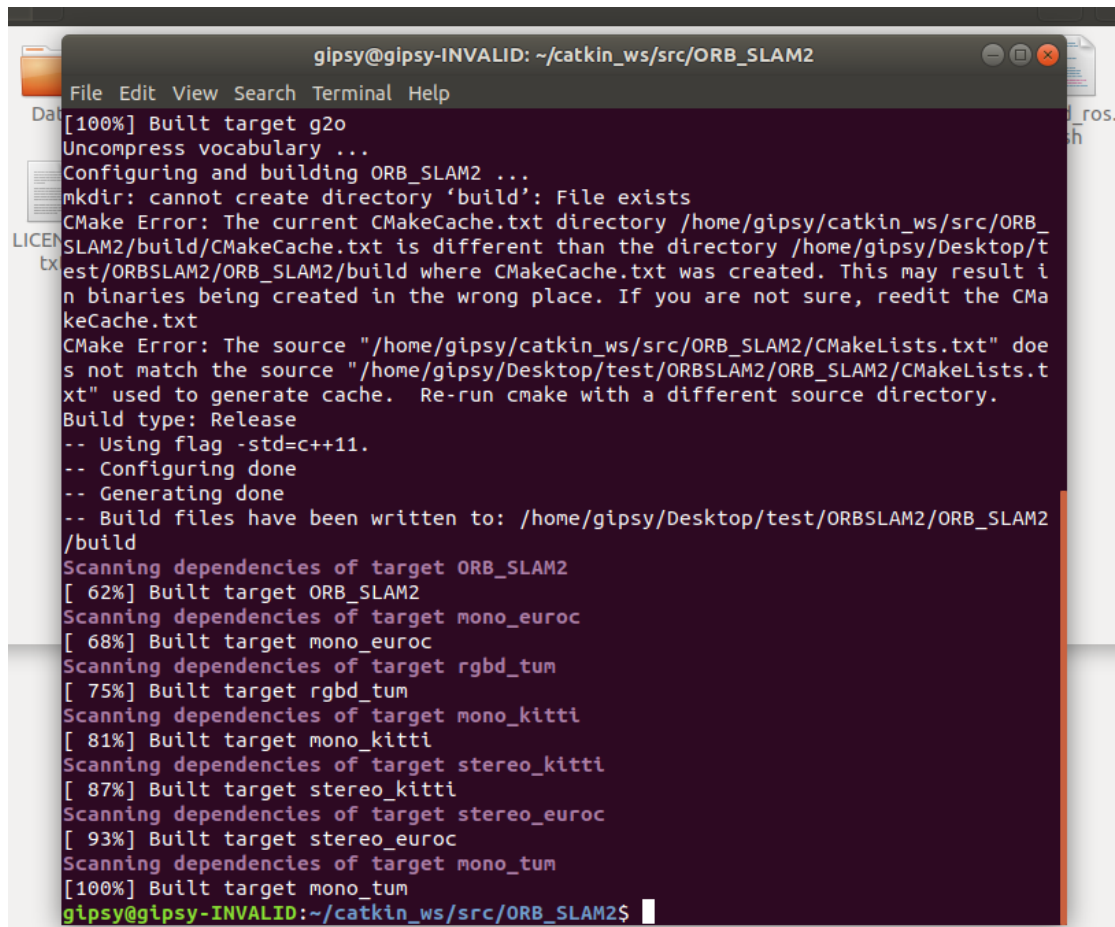
# Build examples
```

库有：opencv、eigen、pangolin、g2o、DBoW2；

6.使用摄像头或视频运行 ORB-SLAM2

(a) 关于如何将摄像头加入 ORB-SLAM2 中：

编译截图：



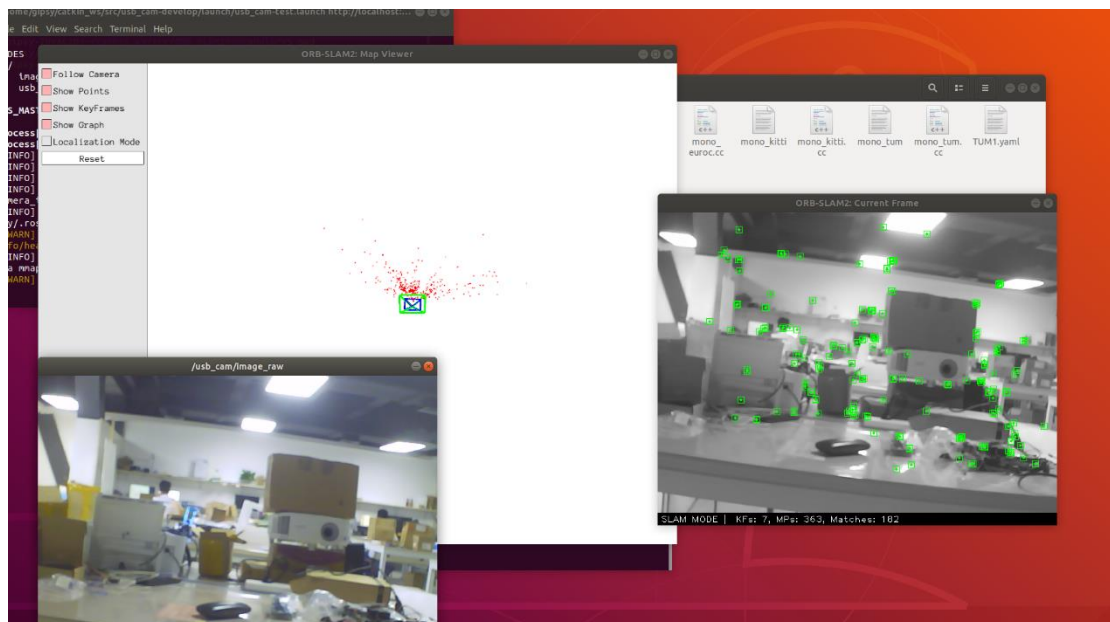
```
gipsy@gipsy-INVALID: ~/catkin_ws/src/ORB_SLAM2
File Edit View Search Terminal Help
[100%] Built target g2o
Uncompress vocabulary ...
Configuring and building ORB_SLAM2 ...
mkdir: cannot create directory 'build': File exists
CMake Error: The current CMakeCache.txt directory /home/gipsy/catkin_ws/src/ORB_SLAM2/build/CMakeCache.txt is different than the directory /home/gipsy/Desktop/test/ORB_SLAM2/ORB_SLAM2/build where CMakeCache.txt was created. This may result in binaries being created in the wrong place. If you are not sure, reedit the CMakeCache.txt
CMake Error: The source "/home/gipsy/catkin_ws/src/ORB_SLAM2/CMakeLists.txt" does not match the source "/home/gipsy/Desktop/test/ORB_SLAM2/ORB_SLAM2/CMakeLists.txt" used to generate cache. Re-run cmake with a different source directory.
Build type: Release
-- Using flag -std=c++11.
-- Configuring done
-- Generating done
-- Build files have been written to: /home/gipsy/Desktop/test/ORB_SLAM2/ORB_SLAM2/build
Scanning dependencies of target ORB_SLAM2
[ 62%] Built target ORB_SLAM2
Scanning dependencies of target mono_euroc
[ 68%] Built target mono_euroc
Scanning dependencies of target rgbd_tum
[ 75%] Built target rgbd_tum
Scanning dependencies of target mono_kitti
[ 81%] Built target mono_kitti
Scanning dependencies of target stereo_kitti
[ 87%] Built target stereo_kitti
Scanning dependencies of target stereo_euroc
[ 93%] Built target stereo_euroc
Scanning dependencies of target mono_tum
[100%] Built target mono_tum
gipsy@gipsy-INVALID:~/catkin_ws/src/ORB_SLAM2$
```

我这里是通过 ROS 的方式运动摄像头 usb_cam 节点和 ORB-SLAM2 节点，然后在 ORB-SLAM2 节点读取 usb_cam 节点输出的 RGB 图像信息，进而完成整个 SLAM 系统的运行。

ROS 下节点如下图所示：


```
gipsy@gipsy-INVALID: ~/catkin_ws/src/ORB_SLAM2/Vocabulary
File Edit View Search Terminal Help
gipsy@gipsy-INVALID:~/catkin_ws/src/ORB_SLAM2/Vocabulary$ PWD
PWD: command not found
gipsy@gipsy-INVALID:~/catkin_ws/src/ORB_SLAM2/Vocabulary$ pwd
/home/gipsy/catkin_ws/src/ORB_SLAM2/Vocabulary
gipsy@gipsy-INVALID:~/catkin_ws/src/ORB_SLAM2/Vocabulary$ roslaunch orb_slam2/mono
Mono
image_view
rosout
usb_cam
gipsy@gipsy-INVALID:~/catkin_ws/src/ORB_SLAM2/Vocabulary$
```

(b) 运行截图：



(c) 体会：

关于 ORB-SLAM2 实际测试效果：

这里我其实跑过 kitti 和 tum 的数据集，在这两个数据集上测试结果都有非常好，用 evo 工具评估，跑出来的轨迹和 groundtruth 非常吻合；但是自己用电脑摄像头跑出来的结果却非常差，而且中间还可能出现了电脑卡住的现象，这里分

析可能有一下几点原因：[1]、我自己测试时，未标定相机，而是直接使用的开源数据集的标定文件；[2]、个人测试时，场景出现动态物体（比如人），因为 ORB-SLAM2 本身没有处理动态物体的能力；[3]、个人测试时，相机移动比较快，导致图像模糊，进行 ORB-SLAM2 在提取特征点时，效果太差。

关于 ORB-SLAM2 现象分析：

因为我个人对 ORB-SLAM2 框架了解的不是很很多，这里通过跑 ORB-SLAM2 分享一下自己推断出来，ORB-SLAM2 框架系统设置的问题；[1]、初始化的时候会初试第一帧保留在 map 上；[2]、map 上的绿色框为关键帧，被保留下来；[3]、map 上的点为特征点。