

2.熟悉Linux

1.描述apt-get安装软件的整体步骤，说明Ubuntu是如何管理软件依赖关系和软件版本的。

使用apt-get安装软件大致分为4步：step1.扫描本地存放的软件包更新列表(由"apt-get update"命令刷新更新列表，也就是/var/lib/apt/lists/)，找到最新版本的软件包；step2.进行软件包依赖关系检查，找到支持该软件正常运行的所有软件包；step3.从软件源所指定的镜像站点中，下载相关软件；step4.解压软件包，并自动完成应用程序的安装和配置

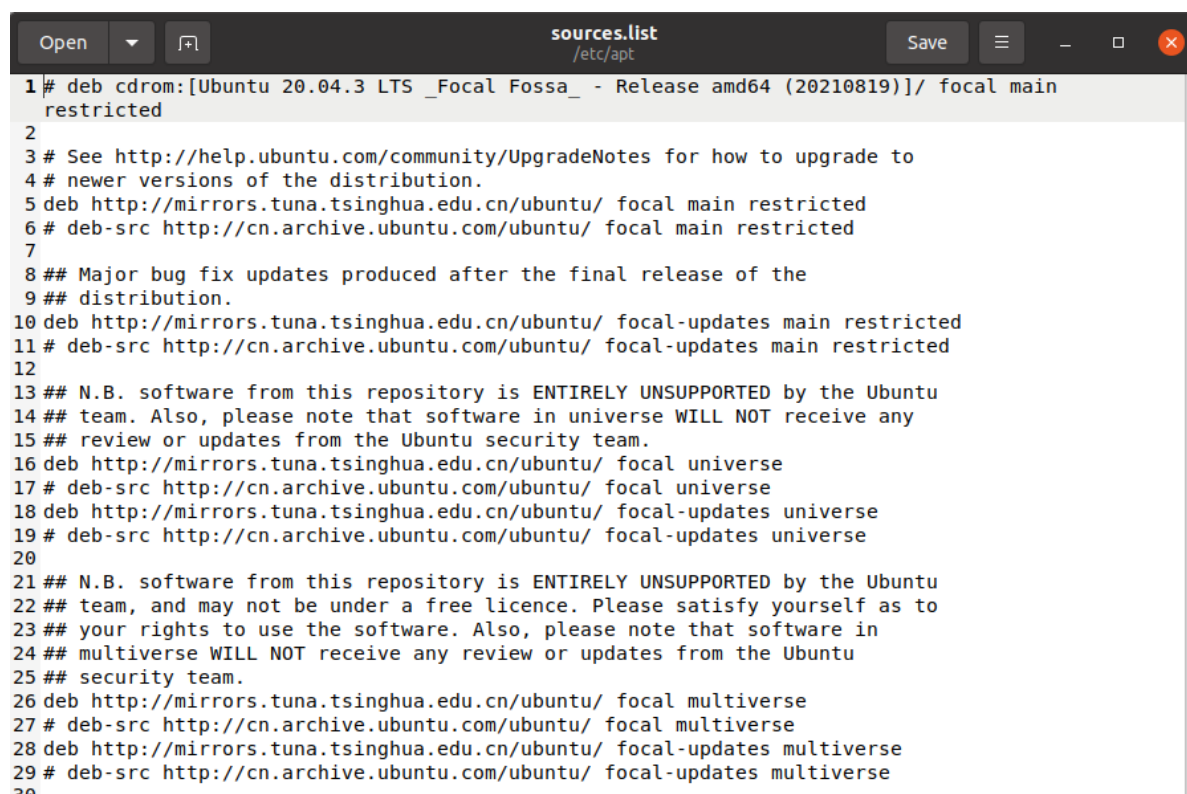
Ubuntu上用APT(Advanced Package Tool)软件包管理工具，这个工具会从Ubuntu的软件源里调用安装所需要安装的包，而且可以自动分析和解决依赖关系，并且将所依赖的软件都装上。Ubuntu如果想安装指定版本的软件，可以在安装时在安装包名后指定要安装的版本即可。

2.什么是软件源？如何更新系统自带的软件源？如何安装来自第三方软件源中的软件？

软件源(软件仓库)一般指debian系操作系统的应用程序安装包仓库，其中存放大量的软件包，apt会从软件源中下载软件，在/etc/apt/sources.list中可以为apt配置软件源

实际上，Ubuntu中软件源还细分为两种：Ubuntu官方软件和PPA软件源

Ubuntu官方软件源中包含了Ubuntu系统中所用的绝大部分软件，它对应的源列表是/etc/apt/sources.list。在这个文件中，记录了Ubuntu官方源的地址，但如果在国内想要加快apt安装和更新软件源的速度，也可以替换为常用的清华源、阿里源等镜像地址，如下图所示



```
1 # deb cdrom:[Ubuntu 20.04.3 LTS _Focal Fossa_ - Release amd64 (20210819)]/ focal main
2 restricted
3 # See http://help.ubuntu.com/community/UpgradeNotes for how to upgrade to
4 # newer versions of the distribution.
5 deb http://mirrors.tuna.tsinghua.edu.cn/ubuntu/ focal main restricted
6 # deb-src http://cn.archive.ubuntu.com/ubuntu/ focal main restricted
7
8 ## Major bug fix updates produced after the final release of the
9 ## distribution.
10 deb http://mirrors.tuna.tsinghua.edu.cn/ubuntu/ focal-updates main restricted
11 # deb-src http://cn.archive.ubuntu.com/ubuntu/ focal-updates main restricted
12
13 ## N.B. software from this repository is ENTIRELY UNSUPPORTED by the Ubuntu
14 ## team. Also, please note that software in universe WILL NOT receive any
15 ## review or updates from the Ubuntu security team.
16 deb http://mirrors.tuna.tsinghua.edu.cn/ubuntu/ focal universe
17 # deb-src http://cn.archive.ubuntu.com/ubuntu/ focal universe
18 deb http://mirrors.tuna.tsinghua.edu.cn/ubuntu/ focal-updates universe
19 # deb-src http://cn.archive.ubuntu.com/ubuntu/ focal-updates universe
20
21 ## N.B. software from this repository is ENTIRELY UNSUPPORTED by the Ubuntu
22 ## team, and may not be under a free licence. Please satisfy yourself as to
23 ## your rights to use the software. Also, please note that software in
24 ## multiverse WILL NOT receive any review or updates from the Ubuntu
25 ## security team.
26 deb http://mirrors.tuna.tsinghua.edu.cn/ubuntu/ focal multiverse
27 # deb-src http://cn.archive.ubuntu.com/ubuntu/ focal multiverse
28 deb http://mirrors.tuna.tsinghua.edu.cn/ubuntu/ focal-updates multiverse
29 # deb-src http://cn.archive.ubuntu.com/ubuntu/ focal-updates multiverse
30
```

PPA源出现的背景是因为系统自带的源是有限的，我们肯定需要一些其他的软件包，然后如果是直接下载deb格式的文件的话，又不能获取到更新和维护，所以就得到PPA源了

更新系统自带的软件源

#使用gedit编译修改软件源，国内开源镜像站点汇总：

<https://www.asfor.cn/server/mirror/index.html>

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

```
#更新软件源
```

```
sudo apt-get update
```

安装来自第三方软件源中的软件

多数软件如vscode、typora等都有deb安装包，可以使用如下命令进行安装

```
sudo dpkg -i *.deb
```

3.除了apt-get以外，还有什么方式在系统中安装所需软件？除了Ubuntu以外，其他发行版使用什么软件管理工具？

其他软件安装方式

- 下载deb格式文件，使用dpkg命令离线安装
- 源码编译安装，通常分为四步：配置->编译->安装->清理临时文件，一般步骤如下

```
mkdir build && cd build
```

```
cmake..
```

```
make -j
```

```
sudo make install
```

- pip或者conda安装，这种并不是ubuntu自带的安装工具，但在特定的情况下需要使用这两种安装方式

其他发行版的软件管理工具

distribution 代表	软件管理机制	使用指令	在线升级机制(指令)
Red Hat/Fedora	RPM	rpm, rpmbuild	YUM (yum)
Debian/Ubuntu	DPKG	dpkg	APT (apt-get)

4.环境变量PATH是什么？有什么用途？LD_LIBRARY_PATH是什么？指令ldconfig有什么用途？

环境变量PATH：PATH变量用于保存可以搜索的目录路径，如果待运行的程序不在当前目录，系统便可以去依次搜索PATH变量中记录的目录，如果在这些目录中找到待运行的程序，系统便可以运行。

LD_LIBRARY_PATH：用于在程序加载运行期间查找动态链接库时指定除了系统默认路径之外的其他路径，LD_LIBRARY_PATH中指定的路径会在系统默认路径之前进行查找。设置方法如下(其中，LIBDIR1和LIBDIR2为两个库目录)

```
export LD_LIBRARY_PATH=LIBDIR1:LIBDIR2:$LD_LIBRARY_PATH
```

ldconfig的用途：ldconfig的用途主要是在默认搜寻目录/lib和/usr/lib以及动态库配置文件/etc/ld.so.conf内所列的目录下，搜索出可共享的动态链接库(格式如lib.so)，进而创建出动态装入程序(ld.so)所需的连接和缓存文件。缓存文件默认为/etc/ld.so.cache，此文件保存已排好序的动态链接库名字列表，为了让动态链接库为系统所共享，需运行动态链接库的管理命令ldconfig，此执行程序存放在/sbin目录下。

ldconfig通常在系统启动时运行，而当用户安装了一个新的动态链接库时，就需要手动运行这个命令。

5.Linux文件权限有哪几种？如何修改一个文件的权限？

Linux文件的基本权限有9个，分别是owner/group/others(文件所有者，称为属主/文件属组的同组用户/可以访问文件的其他用户)三种身份各有自己的read/write/execute权限。访问权限的表示方法有三种，即三组九位字母表示法、三组九位二进制表示法和三位八进制表示法。其中用r表示读，w表示写，x表示可执行可查找，用-表示无权限。

修改文件权限的命令是chmod，执行该命令要求必须为文件属主或者是root用户才能使用。有两种修改方式，字母形式修改权限和数字形式修改权限。

- 字母形式修改权限，即“用户对象 操作符号 操作权限”

chmod [选项] 模式 文件名

- 数字形式修改权限

chmod 八进制模式 文件名

6.Linux用户和用户组是什么概念？用户组的权限是什么意思？有哪些常见的用户组？

Linux操作系统是多用户的分时操作系统，具有功能强大的用户管理机制，它将用户分为组，每个用户都属于某个组，每个用户都需要进行身份验证，同时用户只能在所属组所拥有的权限内工作，这样不仅方便管理，而且增加了系统的安全性。

用户：分为普通用户、管理员用户(root)和系统用户。普通用户在系统上的任务是进行普通的工作，root用户对系统具有绝对的控制权，但操作不当会对系统造成损毁。所以在进行简单任务时使用普通用户即可。

用户组：用户组是用户的容器，通过组，我们可以更加方便的归类、管理用户。用户能从用户组继承权限，一般分为普通用户组，系统用户组，私有用户组。当创建一个新用户时，若没有指定所属的组，系统就建立一个与该用户同名的私有组，当然此时该私有组中只包含这个用户自己。标准组可以容纳多个用户，若使用标准组，在创建一个新的用户时就应该指定他所属于的组。

7.常见的Linux下C++编译器有哪几种？在你的机器上默认是那一种？它能够支持C++的哪个标准？

常见的C++编译器：GCC，Clang，Visual Studio

在我的机器上默认为g++，其版本为9.4.0，支持C++11、C++14、C++17这些标准

```
xwl@xwl-System-Product-Name:~$ g++ --version
g++ (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0
Copyright (C) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

GCC 版本	C++常用标准							
	C++98/03	C++11	C++14	C++17	GNU++98	GNU++11	GNU++14	GNU++17
10.1 ~ 8.4	c++98/c++03	c++11	c++14	c++17	gnu++98/gnu++03	gnu++11	gnu++14	gnu++17
7.5 ~ 5.5	c++98/c++03	c++11	c++14	c++1z (部分支持)	gnu++98/gnu++03	gnu++11	gnu++14	gnu++1z (部分支持)
4.9.4 ~ 4.8.5	c++98/c++03	c++11	c++1y (部分支持)		gnu++98/gnu++03	gnu++11	gnu++1y (部分支持)	
4.7.4	c++98	c++11 (部分支持)			gnu++98	gnu++11 (部分支持)		
4.6.4	c++98	c++0x (部分支持)			gnu++98	gnu++0x (部分支持)		
4.5.4	c++98	c++0x (部分支持)			gnu++98	gnu++0x (部分支持)		

3.SLAM综述文献阅读

1.SLAM会在哪些场合中用到？至少列举三个方向

- AR/VR
- 自动驾驶
- 无人机
- 仓储物流机器人

2.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？

定位和建图两个部分是相互依赖的。传感器感知周围环境并建立地图时需要有自身的位姿信息，才能确定landmark的位置信息；同时传感器数据和环境地图进行匹配，才能完成自身的定位过程。建图的准确性依赖于定位的精度，而定位的实现又离不开精确的建图。

3.SLAM发展历史如何？我们可以将它划分成哪几个阶段？

A seminal work in SLAM is the research of R.C. Smith and P. Cheeseman on the representation and estimation of spatial uncertainty in 1986. Other pioneering work in this field was conducted by the research group of [Hugh F. Durrant-Whyte](#) in the early 1990s, which showed that solutions to SLAM exist in the infinite data limit. This finding motivates the search for algorithms which are computationally tractable and approximate the solution. The acronym SLAM was coined within the paper, "Localization of Autonomous Guided Vehicles" which first appeared in [ISR](#) in 1995.

The self-driving STANLEY and JUNIOR cars, led by [Sebastian Thrun](#), won the DARPA Grand Challenge and came second in the DARPA Urban Challenge in the 2000s, and included SLAM systems, bringing SLAM to worldwide attention. Mass-market SLAM implementations can now be found in consumer robot vacuum cleaners. (摘自Wikipedia)

SLAM问题的提出到现在已有30多年，大致划分为以下几个阶段：

- 基于距离传感器的传统SLAM技术：SLAM技术最早是基于概率统计的扩展卡尔曼滤波的方法(EKF-SLAM)，后来经过改进学者们提出了Fast SLAM方法，该方法大大减小计算量，提升运算速度。
- 基于视觉传感器的SLAM技术：2010年后，越来越多的SLAM系统采用视觉传感器作为主要的输入信号，提出了基于特征点法的SLAM、基于直接法的SLAM
- SLAM技术未来研究热点将是多传感器融合SLAM、语义SLAM、SLAM与深度学习的结合

4.从什么时候开始SLAM区分为前端和后端？为什么我们要把SLAM区分为前端和后端？

大约是在基于非线性优化的SLAM方法兴起时，SLAM开始区分前后端。

前端将传感器数据抽象成适用于估计的模型，估计相邻数据帧间传感器的运动，以及局部地图的样子；后端在这些经由前端处理的抽象数据上执行推理，后端接受不同时刻里程计测量的传感器位姿，以及回环检测的信息，对它们进行优化，得到全局一致的轨迹和地图。前后端所实现的功能不同，目的不同。

5.列举三篇在SLAM领域的经典文献

Mur-Artal, R., Montiel, J.M.M. and Tardos, J.D., 2015. ORB-SLAM: a versatile and accurate monocular SLAM system. IEEE transactions on robotics, 31(5), pp.1147-1163.

Forster, C., Zhang, Z., Gassner, M., Werlberger, M. and Scaramuzza, D., 2016. SVO: Semidirect visual odometry for monocular and multicamera systems. *IEEE Transactions on Robotics*, 33(2), pp.249-265.

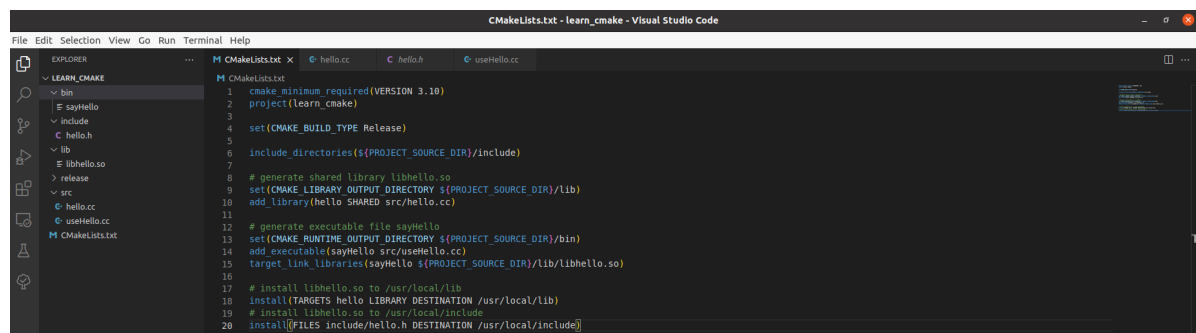
Engel, J., Koltun, V. and Cremers, D., 2017. Direct sparse odometry. *IEEE transactions on pattern analysis and machine intelligence*, 40(3), pp.611-625.

Qin, T., Li, P. and Shen, S., 2018. Vins-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics*, 34(4), pp.1004-1020.

Mourikis, A.I. and Roumeliotis, S.I., 2007, April. A Multi-State Constraint Kalman Filter for Vision-aided Inertial Navigation. In *ICRA* (Vol. 2, p. 6).

4.CMake练习

工程目录及编写的CMakeLists.txt如下



程序编译与运行如下

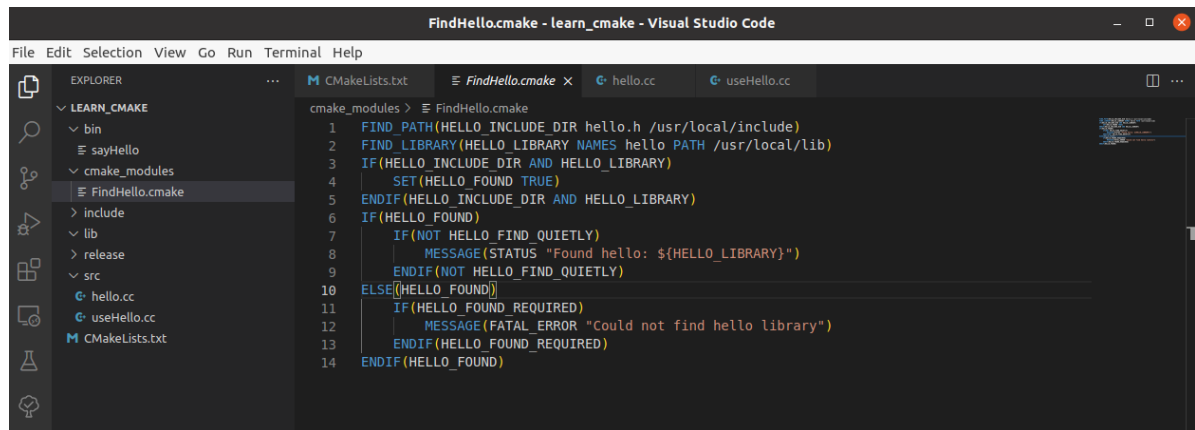
```
xwl@xwl-System-Product-Name:~/Documents/视觉SLAM基础与VIO进阶/L1/code/learn_cmake$ ls
bin CMakeLists.txt include lib src
xwl@xwl-System-Product-Name:~/Documents/视觉SLAM基础与VIO进阶/L1/code/learn_cmake$ mkdir release
xwl@xwl-System-Product-Name:~/Documents/视觉SLAM基础与VIO进阶/L1/code/learn_cmake$ cd release/
xwl@xwl-System-Product-Name:~/Documents/视觉SLAM基础与VIO进阶/L1/code/learn_cmake/release$ cmake ..
-- The C compiler identification is GNU 9.4.0
-- The CXX compiler identification is GNU 9.4.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/xwl/Documents/视觉SLAM基础与VIO进阶/L1/code/learn_cmake/release
xwl@xwl-System-Product-Name:~/Documents/视觉SLAM基础与VIO进阶/L1/code/learn_cmake/release$ make
Scanning dependencies of target sayHello
[ 25%] Building CXX object CMakeFiles/sayHello.dir/src/useHello.cc.o
[ 50%] Linking CXX executable ../bin/sayHello
[ 50%] Built target sayHello
Scanning dependencies of target hello
[ 75%] Building CXX object CMakeFiles/hello.dir/src/hello.cc.o
[100%] Linking CXX shared library ../lib/libhello.so
[100%] Built target hello

xwl@xwl-System-Product-Name:~/Documents/视觉SLAM基础与VIO进阶/L1/code/learn_cmake/release$ cd ../bin
xwl@xwl-System-Product-Name:~/Documents/视觉SLAM基础与VIO进阶/L1/code/learn_cmake/bin$ ./sayHello
Hello SLAM
```

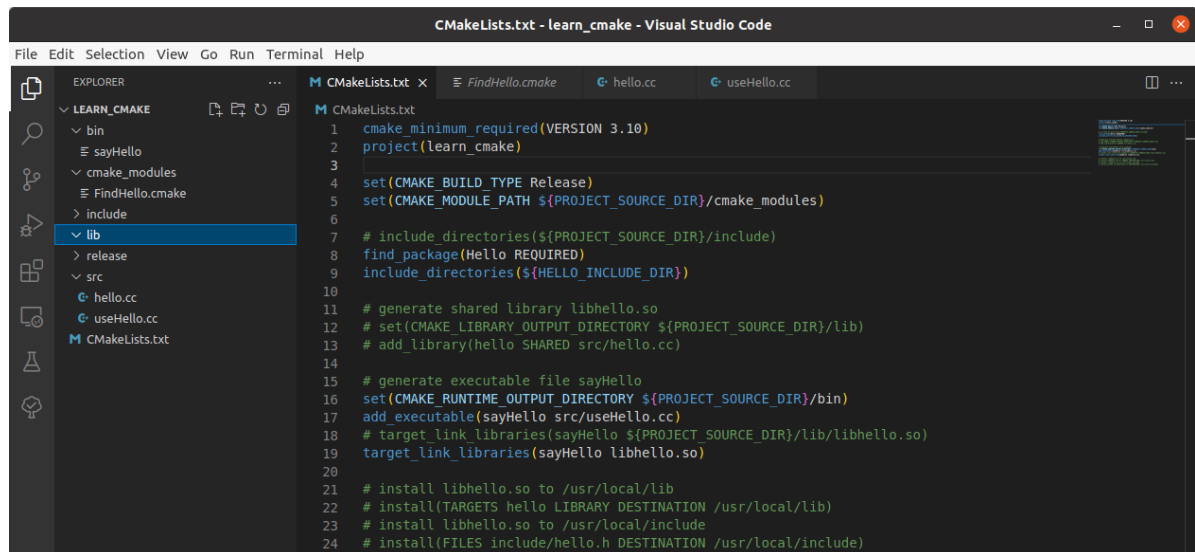
安装头文件和库文件到系统目录如下

```
xwl@xwl-System-Product-Name:~/Documents/视觉SLAM基础与VIO进阶/L1/code/learn_cmake/bin$ cd ../release/
xwl@xwl-System-Product-Name:~/Documents/视觉SLAM基础与VIO进阶/L1/code/learn_cmake/release$ sudo make install
[ 25%] Linking CXX executable ../bin/sayHello
[ 50%] Built target sayHello
[100%] Built target hello
Install the project...
-- Install configuration: "Release"
-- Installing: /usr/local/lib/libhello.so
-- Installing: /usr/local/include/hello.h
```


编写FindHello.cmake和修改CMakeLists.txt如下所示

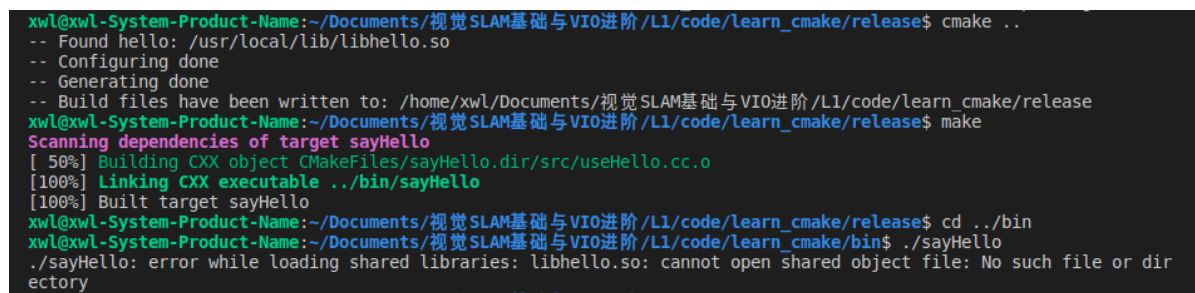


```
FindHello.cmake - learn_cmake - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
  LEARN_CMAKE
    bin
    sayHello
    cmake_modules
      FindHello.cmake
    include
    lib
    release
    src
    hello.cc
    useHello.cc
    CMakeLists.txt
  M CMakeLists.txt
  FindHello.cmake
  hello.cc
  useHello.cc
  CMakeLists.txt
cmake_modules > FindHello.cmake
1 FIND_PATH(HELLO_INCLUDE_DIR hello.h /usr/local/include)
2 FIND_LIBRARY(HELLO_LIBRARY NAMES hello PATH /usr/local/lib)
3 IF(HELLO_INCLUDE_DIR AND HELLO_LIBRARY)
4   SET(HELLO_FOUND TRUE)
5 ENDIF(HELLO_INCLUDE_DIR AND HELLO_LIBRARY)
6 IF(HELLO_FOUND)
7   IF(NOT HELLO_FIND_QUIETLY)
8     MESSAGE(STATUS "Found hello: ${HELLO_LIBRARY}")
9   ENDIF(NOT HELLO_FIND_QUIETLY)
10 ELSE(HELLO_FOUND)
11   IF(HELLO_FOUND_REQUIRED)
12     MESSAGE(FATAL_ERROR "Could not find hello library")
13   ENDIF(HELLO_FOUND_REQUIRED)
14 ENDIF(HELLO_FOUND)
```



```
CMakeLists.txt - learn_cmake - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
  LEARN_CMAKE
    bin
    sayHello
    cmake_modules
      FindHello.cmake
    include
    lib
    release
    src
    hello.cc
    useHello.cc
    CMakeLists.txt
  M CMakeLists.txt
  FindHello.cmake
  hello.cc
  useHello.cc
  CMakeLists.txt
M CMakeLists.txt
1 cmake_minimum_required(VERSION 3.10)
2 project(learn_cmake)
3
4 set(CMAKE_BUILD_TYPE Release)
5 set(CMAKE_MODULE_PATH ${PROJECT_SOURCE_DIR}/cmake_modules)
6
7 # include_directories(${PROJECT_SOURCE_DIR}/include)
8 find_package(Hello REQUIRED)
9 include_directories(${HELLO_INCLUDE_DIR})
10
11 # generate shared library libhello.so
12 # set(CMAKE_LIBRARY_OUTPUT_DIRECTORY ${PROJECT_SOURCE_DIR}/lib)
13 # add_library(hello SHARED src/hello.cc)
14
15 # generate executable file sayHello
16 set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${PROJECT_SOURCE_DIR}/bin)
17 add_executable(sayHello src/useHello.cc)
18 # target_link_libraries(sayHello ${PROJECT_SOURCE_DIR}/lib/libhello.so)
19 target_link_libraries(sayHello libhello.so)
20
21 # install libhello.so to /usr/local/lib
22 # install(TARGETS hello LIBRARY DESTINATION /usr/local/lib)
23 # install libhello.so to /usr/local/include
24 # install(FILES include/hello.h DESTINATION /usr/local/include)
```

编译成功，但是运行可执行文件时提示找不到动态库libhello.so，如下所示



```
xwl@xwl-System-Product-Name:~/Documents/视觉SLAM基础与VIO进阶/L1/code/learn_cmake/release$ cmake ..
-- Found hello: /usr/local/lib/libhello.so
-- Configuring done
-- Generating done
-- Build files have been written to: /home/xwl/Documents/视觉SLAM基础与VIO进阶/L1/code/learn_cmake/release
xwl@xwl-System-Product-Name:~/Documents/视觉SLAM基础与VIO进阶/L1/code/learn_cmake/release$ make
Scanning dependencies of target sayHello
[ 50%] Building CXX object CMakeFiles/sayHello.dir/src/useHello.cc.o
[100%] Linking CXX executable ../bin/sayHello
[100%] Built target sayHello
xwl@xwl-System-Product-Name:~/Documents/视觉SLAM基础与VIO进阶/L1/code/learn_cmake/release$ cd ../bin
xwl@xwl-System-Product-Name:~/Documents/视觉SLAM基础与VIO进阶/L1/code/learn_cmake/bin$ ./sayHello
./sayHello: error while loading shared libraries: libhello.so: cannot open shared object file: No such file or directory
```

解决该问题的方案有三种：

- 用ln将需要的so文件链接到/usr/lib或者/lib这两个默认的目录下边

```
ln -s /where/you/install/lib/*.so /usr/lib
```

```
sudo ldconfig
```

- 修改LD_LIBRARY_PATH

```
export LD_LIBRARY_PATH=/where/you/install/lib:$LD_LIBRARY_PATH
```

```
sudo ldconfig
```

- 修改/etc/ld.so.conf，然后刷新

```
gedit /etc/ld.so.conf
```

```
add /where/you/install/lib
```

```
sudo ldconfig
```

使用第三种方法，修改/etc/ld.so.conf文件如下后，sudo /sbin/ldconfig -v即可



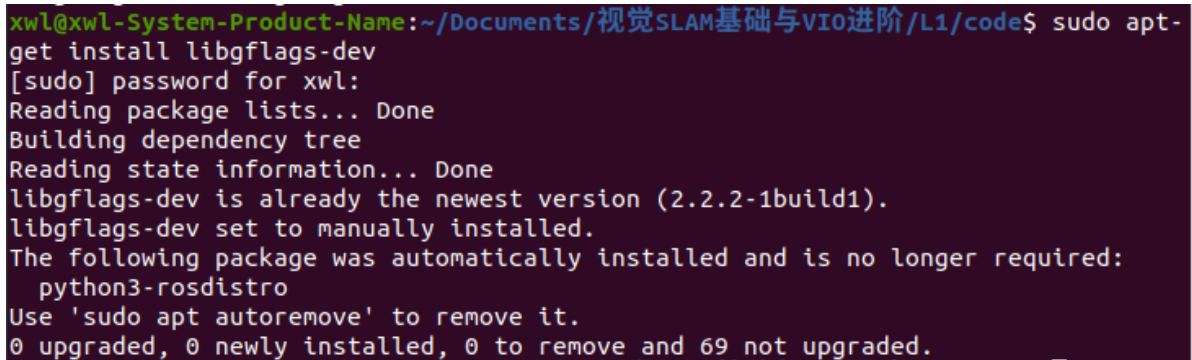
```
Open  ld.so.conf /etc  Save  -  +  x
1 include /etc/ld.so.conf.d/*.conf
2
3 /usr/local/lib
4

xwl@xwl-System-Product-Name:~/Documents/视觉SLAM基础与VIO进阶/L1/code/learn_cmake/bin$ ./sayHello
Hello SLAM
```

5.gflags, glog, gtest的使用

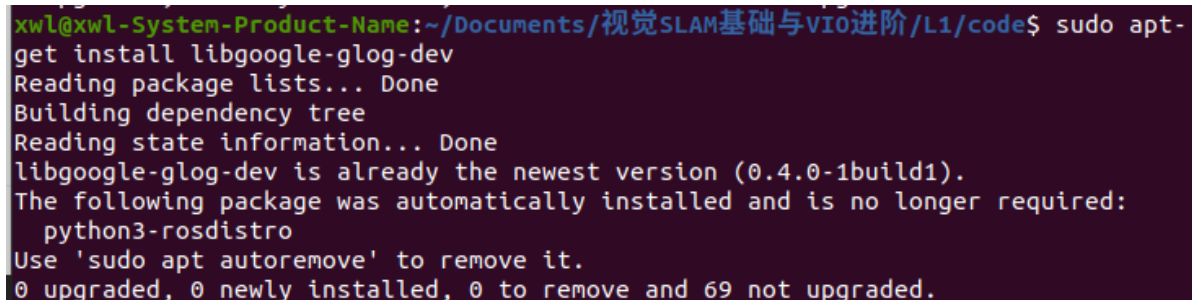
1.安装gflags, glog, gtest

```
sudo apt-get install libgflags-dev
```



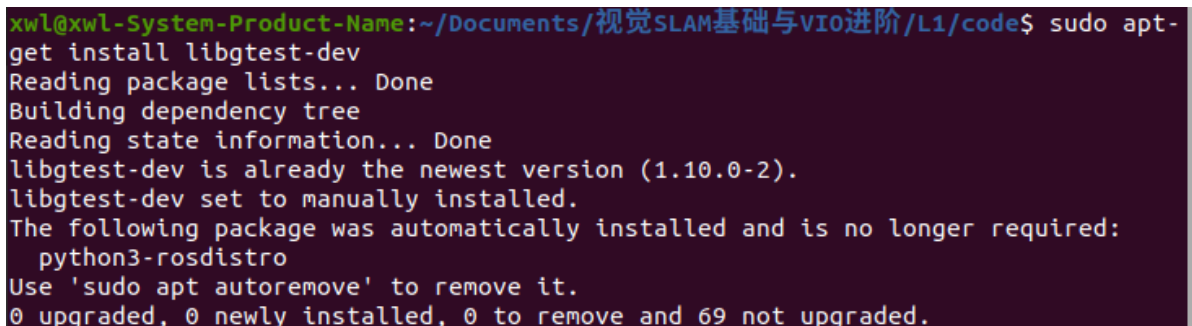
```
xwl@xwl-System-Product-Name:~/Documents/视觉SLAM基础与VIO进阶/L1/code$ sudo apt-get install libgflags-dev
[sudo] password for xwl:
Reading package lists... Done
Building dependency tree
Reading state information... Done
libgflags-dev is already the newest version (2.2.2-1build1).
libgflags-dev set to manually installed.
The following package was automatically installed and is no longer required:
  python3-rosdistro
Use 'sudo apt autoremove' to remove it.
0 upgraded, 0 newly installed, 0 to remove and 69 not upgraded.
```

```
sudo apt-get install libgoogle-glog-dev
```



```
xwl@xwl-System-Product-Name:~/Documents/视觉SLAM基础与VIO进阶/L1/code$ sudo apt-get install libgoogle-glog-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
libgoogle-glog-dev is already the newest version (0.4.0-1build1).
The following package was automatically installed and is no longer required:
  python3-rosdistro
Use 'sudo apt autoremove' to remove it.
0 upgraded, 0 newly installed, 0 to remove and 69 not upgraded.
```

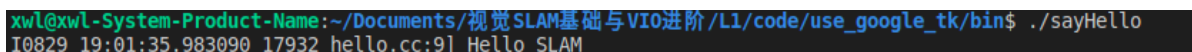
```
sudo apt-get install libgtest-dev
```



```
xwl@xwl-System-Product-Name:~/Documents/视觉SLAM基础与VIO进阶/L1/code$ sudo apt-get install libgtest-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
libgtest-dev is already the newest version (1.10.0-2).
libgtest-dev set to manually installed.
The following package was automatically installed and is no longer required:
  python3-rosdistro
Use 'sudo apt autoremove' to remove it.
0 upgraded, 0 newly installed, 0 to remove and 69 not upgraded.
```

除此之外，均可采用源码编译安装的方式安装上述3个软件

2.使用glog打印



```
xwl@xwl-System-Product-Name:~/Documents/视觉SLAM基础与VIO进阶/L1/code/use_google_tk/bin$ ./sayHello
I0829 19:01:35.983090 17932 hello.cc:9] Hello SLAM
```

3.使用gflags指明打印次数print_times，默认为1，当用户传递该参数时，即打印多少遍Hello SLAM

终端输入 `./sayHello app_containing_foo --print_times=*` , *即打印次数

```
xwl@xwl-System-Product-Name:~/Documents/视觉SLAM基础与VIO进阶/L1/code/use_google_tk/bin$ ./sayHello
I0829 19:31:59.365237 20547 hello.cc:9] Hello SLAM
xwl@xwl-System-Product-Name:~/Documents/视觉SLAM基础与VIO进阶/L1/code/use_google_tk/bin$ ./sayHello app_containing_foo --print_times=5
I0829 19:32:54.080188 20751 hello.cc:9] Hello SLAM
I0829 19:32:54.080297 20751 hello.cc:9] Hello SLAM
I0829 19:32:54.080307 20751 hello.cc:9] Hello SLAM
I0829 19:32:54.080310 20751 hello.cc:9] Hello SLAM
I0829 19:32:54.080315 20751 hello.cc:9] Hello SLAM
```

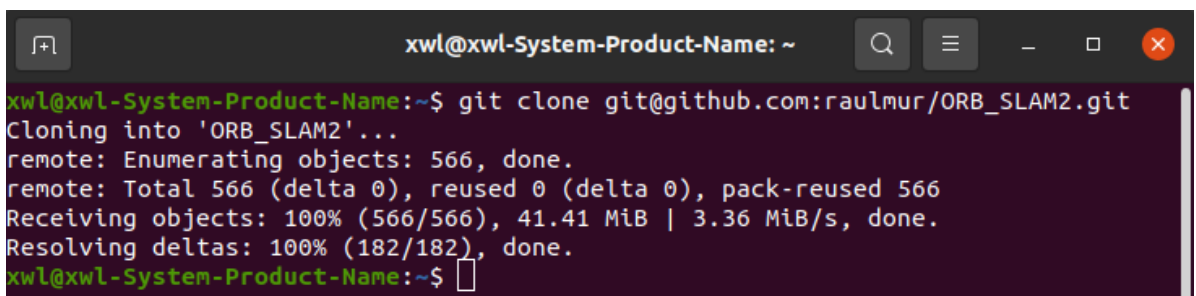
4. 书写gtest单元测试并修改CMakeLists.txt

```
xwl@xwl-System-Product-Name:~/Documents/视觉SLAM基础与VIO进阶/L1/code/use_google_tk/bin$ ./sayHello
[=====] Running 1 test from 1 test suite.
[-----] Global test environment set-up.
[-----] 1 test from sayHelloTest
[ RUN      ] sayHelloTest.NonInput
I0829 20:24:06.940532 25558 hello.cc:9] Hello SLAM
[ OK       ] sayHelloTest.NonInput (0 ms)
[-----] 1 test from sayHelloTest (0 ms total)

[-----] Global test environment tear-down
[=====] 1 test from 1 test suite ran. (0 ms total)
[ PASSED  ] 1 test.
```

6.理解ORB_SLAM2框架

1. 下载ORB_SLAM2终端截图



```
xwl@xwl-System-Product-Name: ~
xwl@xwl-System-Product-Name:~$ git clone git@github.com:raulmur/ORB_SLAM2.git
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 | 3.36 MiB/s, done.
Resolving deltas: 100% (182/182), done.
xwl@xwl-System-Product-Name:~$
```

2. 回答问题

(a) ORB_SLAM2将编译出什么结果？有几个库文件和可执行文件？

编译出1个库文件libORB_SLAM2.so，存放在工程的lib文件夹下

编译出6个可执行文件，分别为

- rgbd_tum，存放在工程的Examples/RGB-D文件夹下
- stereo_kitti和stereo_euroc，存放在工程的Examples/Stereo文件夹下
- mono_tum、mono_kitti和mono_euroc，存放在工程的Examples/Monocular文件夹下

同时，ORB_SLAM2提供了ROS接口，在工程的Examples/ROS/ORB_SLAM2目录下，编译会生成4个节点分别为Mono、MonoAR、Stereo和RGBD

(b) ORB_SLAM2中的include，src，Examples三个文件夹中都含有什么内容？

include和src文件夹下是一些需要编译成动态链接库文件的头文件和源代码。ORB_SLAM2为经典的四线程结构，其中Tracking.cc主要是从图像中提取ORB特征；LocalMapping.cc主要是完成局部地图构建；LoopClosing.cc分为两个过程，分别为闭环探测与闭环矫正；Viewer.cc即调用Pangolin库完成可视化操作。

Examples文件夹下提供了Monocular、RGB-D、ROS和Stereo的接口文件

(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？

链接到了动态库libORB_SLAM2.so，还有OpenCV、Eigen3、Pangolin、DBow2、g2o这些库

7.* 使用摄像头或视频运行ORB_SLAM2

1. 编译完成的截图

我的电脑环境为:Ubuntu20.04+OpenCV-4.2.0+Pangolin-0.5, 通过修改CMakeLists中C++标准为C++14 并且修改部分文件中调用OpenCV库函数的API, 完成编译如下

```
xwl@xwl-System-Product-Name: ~/Documents/视觉SLAM基础与VIO进阶/L1/code/ORB_SLAM2
162 | EIGEN_DEPRECATED const unsigned int AlignedBit = 0x80;
In file included from /home/xwl/Documents/视觉SLAM基础与VIO进阶/L1/code/ORB_SLAM2/Thirdparty/g2o/g2o/types/types_seven_dof_expmap.h:35,
from /home/xwl/Documents/视觉SLAM基础与VIO进阶/L1/code/ORB_SLAM2/include/LoopClosing.h:34,
from /home/xwl/Documents/视觉SLAM基础与VIO进阶/L1/code/ORB_SLAM2/include/LocalMapping.h:26,
from /home/xwl/Documents/视觉SLAM基础与VIO进阶/L1/code/ORB_SLAM2/include/Tracking.h:31,
from /home/xwl/Documents/视觉SLAM基础与VIO进阶/L1/code/ORB_SLAM2/include/System.h:29,
from /home/xwl/Documents/视觉SLAM基础与VIO进阶/L1/code/ORB_SLAM2/Examples/Monocular/mono_euroc.cc:29:
/home/xwl/Documents/视觉SLAM基础与VIO进阶/L1/code/ORB_SLAM2/Thirdparty/g2o/g2o/types/./core/base_binary_edge.h: In instantiation of 'class g2o::BaseBinaryEdge<2, Eigen::Matrix<double, 2, 1>, g2o::Vertex
SBAPOutXYZ, g2o::VertexSBAExpmap>':
/home/xwl/Documents/视觉SLAM基础与VIO进阶/L1/code/ORB_SLAM2/Thirdparty/g2o/g2o/types/./core/base_binary_edge.h:130:36: required from here
/home/xwl/Documents/视觉SLAM基础与VIO进阶/L1/code/ORB_SLAM2/Thirdparty/g2o/g2o/types/./core/base_binary_edge.h:59:80: warning: 'Eigen::AlignedBit' is deprecated [-Wdeprecated-declarations]
59 |         typedef Eigen::Map<Matrix<double, D1, D2>, Matrix<double, D1, D2>::Flags & AlignedBit ? Aligned : Unaligned > HessianBlockType;
In file included from /usr/include/eigen3/Eigen/Core:363,
from /usr/local/include/pangolin/gl/gl.h:40,
from /usr/local/include/pangolin/pangolin.h:34,
from /home/xwl/Documents/视觉SLAM基础与VIO进阶/L1/code/ORB_SLAM2/include/MapDrawer.h:27,
from /home/xwl/Documents/视觉SLAM基础与VIO进阶/L1/code/ORB_SLAM2/include/Viewer.h:26,
from /home/xwl/Documents/视觉SLAM基础与VIO进阶/L1/code/ORB_SLAM2/include/Tracking.h:28,
from /home/xwl/Documents/视觉SLAM基础与VIO进阶/L1/code/ORB_SLAM2/include/System.h:29,
from /home/xwl/Documents/视觉SLAM基础与VIO进阶/L1/code/ORB_SLAM2/Examples/Monocular/mono_euroc.cc:29:
/usr/include/eigen3/Eigen/src/Core/Util/Constants.h:162:37: note: declared here
162 | EIGEN_DEPRECATED const unsigned int AlignedBit = 0x80;
In file included from /home/xwl/Documents/视觉SLAM基础与VIO进阶/L1/code/ORB_SLAM2/Thirdparty/g2o/g2o/types/types_seven_dof_expmap.h:35,
from /home/xwl/Documents/视觉SLAM基础与VIO进阶/L1/code/ORB_SLAM2/include/LoopClosing.h:34,
from /home/xwl/Documents/视觉SLAM基础与VIO进阶/L1/code/ORB_SLAM2/include/LocalMapping.h:26,
from /home/xwl/Documents/视觉SLAM基础与VIO进阶/L1/code/ORB_SLAM2/include/Tracking.h:31,
from /home/xwl/Documents/视觉SLAM基础与VIO进阶/L1/code/ORB_SLAM2/include/System.h:29,
from /home/xwl/Documents/视觉SLAM基础与VIO进阶/L1/code/ORB_SLAM2/Examples/Monocular/mono_euroc.cc:29:
/home/xwl/Documents/视觉SLAM基础与VIO进阶/L1/code/ORB_SLAM2/Thirdparty/g2o/g2o/types/./core/base_binary_edge.h:60:80: warning: 'Eigen::AlignedBit' is deprecated [-Wdeprecated-declarations]
60 |         typedef Eigen::Map<Matrix<double, D1, D2>, Matrix<double, D1, D2>::Flags & AlignedBit ? Aligned : Unaligned > HessianBlockTransposedType;
In file included from /usr/include/eigen3/Eigen/Core:363,
from /usr/local/include/pangolin/gl/gl.h:40,
from /usr/local/include/pangolin/pangolin.h:34,
from /home/xwl/Documents/视觉SLAM基础与VIO进阶/L1/code/ORB_SLAM2/include/MapDrawer.h:27,
from /home/xwl/Documents/视觉SLAM基础与VIO进阶/L1/code/ORB_SLAM2/include/Viewer.h:26,
from /home/xwl/Documents/视觉SLAM基础与VIO进阶/L1/code/ORB_SLAM2/include/Tracking.h:28,
from /home/xwl/Documents/视觉SLAM基础与VIO进阶/L1/code/ORB_SLAM2/include/System.h:29,
from /home/xwl/Documents/视觉SLAM基础与VIO进阶/L1/code/ORB_SLAM2/Examples/Monocular/mono_euroc.cc:29:
/usr/include/eigen3/Eigen/src/Core/Util/Constants.h:162:37: note: declared here
162 | EIGEN_DEPRECATED const unsigned int AlignedBit = 0x80;
[ 90%] Linking CXX executable ../Examples/RGB-D/rgbd_tum
[ 93%] Linking CXX executable ../Examples/Monocular/mono kitti
[ 96%] Linking CXX executable ../Examples/Monocular/mono_euroc
[100%] Linking CXX executable ../Examples/Stereo/stereo_euroc
[100%] Built target rgbd_tum
[100%] Built target mono_kitti
[100%] Built target mono_euroc
[100%] Built target stereo_euroc
```

2. 将myslam.cpp和myvideo.cpp加入到工程

在Examples下新建Test文件夹, 并把myslam.cpp和myvideo.cpp放到该文件夹下, 并修改CMakeLists.txt如下

```
CMakeLists.txt - ORB_SLAM2 - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
ORB_SLAM2
  build
  cmake_modules
  Examples
  Monocular
  RGB-D
  ROS
  Stereo
  EuRoC_TimeStamps
  KITTI00-02.yaml
  KITTI03.yaml
  KITTI04-12.yaml
  stereo_euroc
  stereo_euroc.cc
  stereo_kitti
  stereo_kitti.cc
  Test
  myslam.cpp
  myvideo.cpp
  include
  src
  Thirdparty
  Vocabulary
  gitignore
  build_ros.sh
  build.sh
  CMakeLists.txt
  Dependencies.md
  LICENSE.txt
  README.md
CMakeLists.txt
85 add_executable(rgbd_tum
86 Examples/RGB-D/rgbd_tum.cc)
87 target_link_libraries(rgbd_tum ${PROJECT_NAME})
88
89 set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${PROJECT_SOURCE_DIR}/Examples/Stereo)
90
91 add_executable(stereo_kitti
92 Examples/Stereo/stereo_kitti.cc)
93 target_link_libraries(stereo_kitti ${PROJECT_NAME})
94
95 add_executable(stereo_euroc
96 Examples/Stereo/stereo_euroc.cc)
97 target_link_libraries(stereo_euroc ${PROJECT_NAME})
98
99
100 set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${PROJECT_SOURCE_DIR}/Examples/Monocular)
101
102 add_executable(mono_tum
103 Examples/Monocular/mono_tum.cc)
104 target_link_libraries(mono_tum ${PROJECT_NAME})
105
106 add_executable(mono_kitti
107 Examples/Monocular/mono_kitti.cc)
108 target_link_libraries(mono_kitti ${PROJECT_NAME})
109
110 add_executable(mono_euroc
111 Examples/Monocular/mono_euroc.cc)
112 target_link_libraries(mono_euroc ${PROJECT_NAME})
113
114 set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${PROJECT_SOURCE_DIR}/Examples/Test)
115
116 add_executable(my_slam
117 Examples/Test/my_slam.cpp)
118 target_link_libraries(my_slam ${PROJECT_NAME})
119
120 add_executable(my_video
121 Examples/Test/my_video.cpp)
122 target_link_libraries(my_video ${PROJECT_NAME})
```

```

xwl@xwl-System-Product-Name:~/Documents/视觉SLAM基础与VIO进阶/L1/code/ORB_SLAM2$ ./build.sh
Configuring and building Thirdparty/DBoW2 ...
mkdir: cannot create directory 'build': File exists
-- Configuring done
-- Generating done
-- Build files have been written to: /home/xwl/Documents/视觉SLAM基础与VIO进阶/L1/code/ORB_SLAM2/Thirdparty/DBoW2/build
[100%] Built target DBoW2
Configuring and building Thirdparty/g2o ...
mkdir: cannot create directory 'build': File exists
-- BUILD TYPE:Release
-- Compiling on Unix
-- Configuring done
-- Generating done
-- Build files have been written to: /home/xwl/Documents/视觉SLAM基础与VIO进阶/L1/code/ORB_SLAM2/Thirdparty/g2o/build
[100%] Built target g2o
Uncompress vocabulary ...
Configuring and building ORB_SLAM2 ...
mkdir: cannot create directory 'build': File exists
Build type: Release
-- Using flag -std=c++14.
-- Configuring done
-- Generating done
-- Build files have been written to: /home/xwl/Documents/视觉SLAM基础与VIO进阶/L1/code/ORB_SLAM2/build
[ 55%] Built target ORB_SLAM2
[ 61%] Built target mono_euroc
[ 66%] Built target mono_tum
[ 69%] Linking CXX executable ../Examples/Test/myslam
[ 72%] Linking CXX executable ../Examples/Test/myvideo
[ 77%] Built target mono_kitti
[ 83%] Built target rgbd_tum
[ 88%] Built target stereo_kitti
[ 94%] Built target stereo_euroc
[ 97%] Built target myvideo
[100%] Built target mysam

```

3.运行ORB_SLAM2

读视频myvideo.mp4和参数文件myvideo.yaml运行ORB_SLAM2截图如下

```

xwl@xwl-System-Product-Name:~/Documents/视觉SLAM基础与VIO进阶/L1/code/ORB_SLAM2/Examples/Test$ ./myvideo ../Vocabulary/ORBvoc.txt ./myvideo.yaml

ORB-SLAM2 Copyright (C) 2014-2016 Raul Mur-Artal, University of Zaragoza.
This program comes with ABSOLUTELY NO WARRANTY;
This is free software, and you are welcome to redistribute it
under certain conditions. See LICENSE.txt.

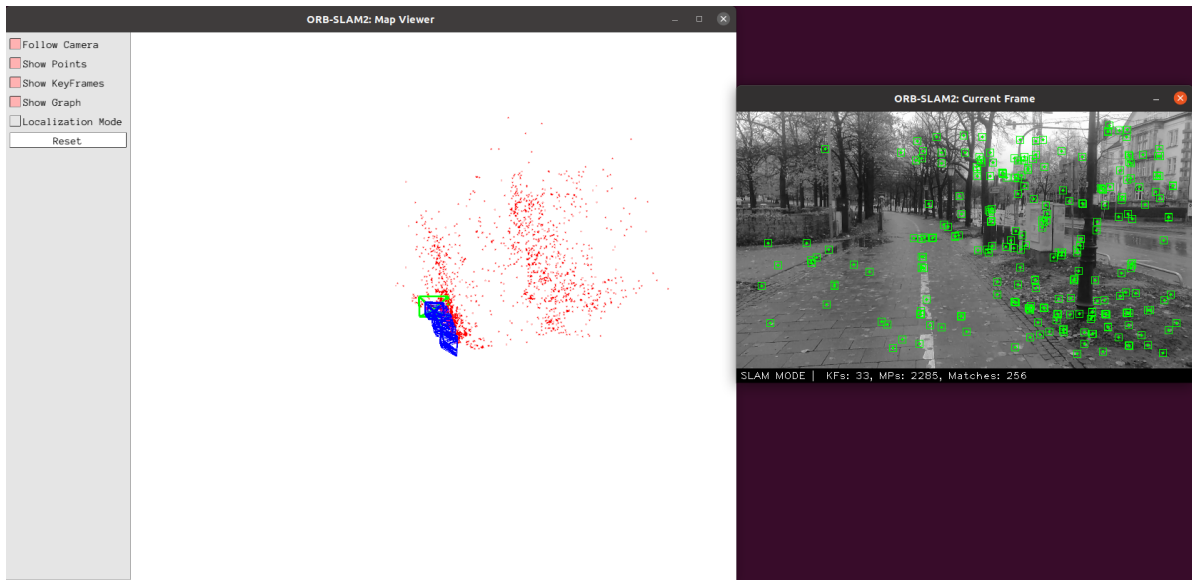
Input sensor was set to: Monocular

Loading ORB Vocabulary. This could take a while...
Vocabulary loaded!

Camera Parameters:
- fx: 500
- fy: 500
- cx: 320
- cy: 180
- k1: 0
- k2: 0
- p1: 0
- p2: 0
- fps: 30
- color order: BGR (ignored if grayscale)

ORB Extractor Parameters:
- Number of Features: 2000
- Scale Levels: 8
- Scale Factor: 1.2
- Initial Fast Threshold: 20
- Minimum Fast Threshold: 7
New Map created with 124 points

```



使用外接USB相机和参数文件myvideo.yaml（相机未标定）运行ORB_SLAM2截图如下

```

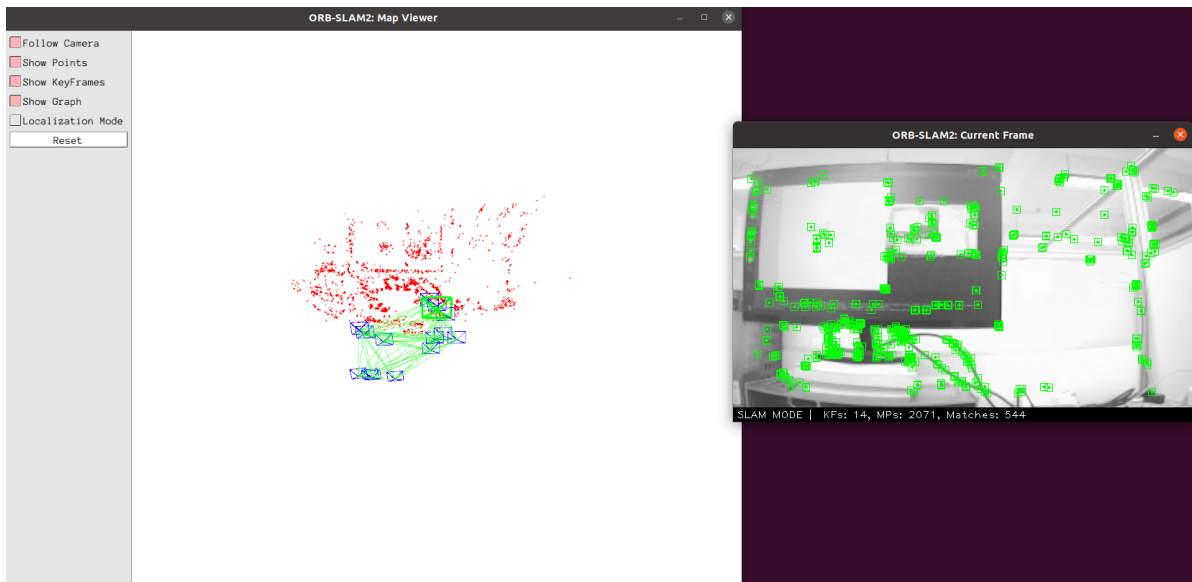
xwlgxwl-System-Product-Name:~/Documents/视觉SLAM基础与VIO进阶/11/code/ORB_SLAM2/Examples/Test$ ./myslam ../Vocabulary/ORBvoc.txt ./myvideo.yaml

ORB-SLAM2 Copyright (C) 2014-2016 Raul Mur-Artal, University of Zaragoza.
This program comes with ABSOLUTELY NO WARRANTY;
This is free software, and you are welcome to redistribute it
under certain conditions. See LICENSE.txt.

Input sensor was set to: Monocular
Loading ORB Vocabulary. This could take a while...
Vocabulary loaded!

Camera Parameters:
- fx: 500
- fy: 500
- cx: 320
- cy: 180
- k1: 0
- k2: 0
- p1: 0
- p2: 0
- fps: 30
- color order: BGR (ignored if grayscale)

ORB Extractor Parameters:
- Number of Features: 2000
- Scale Levels: 8
- Scale Factor: 1.2
- Initial Fast Threshold: 20
- MinNum Fast Threshold: 7
[ WARN:1] global ../modules/videoio/src/cap_gstreamer.cpp (1758) handleMessage OpenCV | GStreamer warning: Embedded video playback halted; module v4l2src0 reported: Internal data stream error.
[ WARN:1] global ../modules/videoio/src/cap_gstreamer.cpp (888) open OpenCV | GStreamer warning: unable to start pipeline
[ WARN:1] global ../modules/videoio/src/cap_gstreamer.cpp (480) isPipelinePlaying OpenCV | GStreamer warning: GStreamer: pipeline have not been created
New Map created with 208 points
  
```



运行给定数据感觉效果还不错，但是单目初始化过程耗费了一定的时间。外接摄像头实时运行时，可能由于环境中纹理信息的缺少或者相机内参没有标定，经常会出现跟丢的情况，然后程序就会创建新的Map

```
New Map created with 94 points
Wrong initialization, resetting...
System Resetting
Resetting Local Mapper... done
Resetting Loop Closing... done
Resetting Database... done
New Map created with 94 points
Wrong initialization, resetting...
System Resetting
Resetting Local Mapper... done
Resetting Loop Closing... done
Resetting Database... done
New Map created with 93 points
Wrong initialization, resetting...
System Resetting
Resetting Local Mapper... done
Resetting Loop Closing... done
Resetting Database... done
New Map created with 86 points
```