

# 第一章作业

Student name: Francisk

Due date: January 23th, 2022

## 1 第 1 题

已阅。

## 2 第 2 题

### 2.1 要求 1

“apt-get install” 安装软件包分为 4 步：

1. 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
2. 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
3. 从软件源所指的镜像站点中，下载相关软件包
4. 解压软件包，并自动完成应用程序的安装和配置。

依赖的意思就是你调用某个别人写好的函数之前需要使用某个包或者调用某个包。Ubuntu 上用与 Debian 一样的 Deb 软件管理工具，apt-get 就是 Ubuntu 的 Deb 软件管理工具，即 APT 包管理工具。这个软件会从 Ubuntu 的软件源库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安装完成。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边指定所要安装的版本即可。

## 2.2 要求 2

软件源一般指 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。更换系统自带的软件源：

```
1 #STEP1: 备份源列表
2 sudo cp /etc/apt/sources.list /etc/apt/sources.list.bak
3 #STEP2: 打开sources.list文件
4 sudo gedit /etc/apt/sources.list
5 #STEP3: 将文件中的http://mirrors.aliyun.com/ubuntu/ 统一替换为（以阿里云为
   例）http://mirrors.aliyun.com/ubuntu/ 保存退出
6 #STEP4: 更新并升级（更新过程询问是否下载包，输入“y”）
7 sudo apt-get update && sudo apt-get upgrade
```

Listing 1: 更换软件源

## 2.3 要求 3

其他安装方式：

1. 下载 deb 包来进行安装：

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

Listing 2: 使用 deb 包进行安装

2. make 方式安装：对于某些软件仓库没有的库或者安装包，或者下载过慢的这种情况，我们可以选择直接下载其源码，然后进行源码安装。通常分为四步：配置-> 编译-> 安装-> 清除临时文件：

```
1 git clone https: //....
2 cd XXX
3 make
4 sudo make install
5 make clean
```

Listing 3: 使用 make 进行安装

3. pip 安装，这种方式不是 Ubuntu 自带的安装方式，需要在特定环境下安装

```
1 pip install XXX
```

Listing 4: 使用 pip 进行安装

其他发行版本的软件管理工具如图 1.1 所示

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)

图 1.1 其他发行版本的软件管理工具

2.4 要求 4

环境变量 PATH：指在操作系统中用来给系统和应用程序设置运行环境的一些参数，如临时文件夹位置和系统文件夹位置等，它包含了一个或多个应用程序将使用到的信息。

LD\_LIBRARY\_PATH：程序已经成功编译并且链接成功后，使用 LD\_LIBRARY\_PATH 来搜索目录，该变量中只有动态库有意义。

ldconfig 的作用：ldconfig 是一个动态链接库管理命令，为了让动态链接库为系统所共享，还需运行动态链接库的管理命令ldconfig。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（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。

2.5 要求 5

文件权限是指文件的访问控制，即那些用户和组群可以访问文件以及可以执行什么样的操作。默认情况下文件或目录的创建者即为该对象的属主。属主对文件或者目录有特别的操作权限。

访问权限规定三种不同类型的用户：

文件属主（Owner）：文件的所有者，称为属主。

同组用户（Group）：文件属组的同组用户。

其它用户（Others）：可以访问文件的其他用户。

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，用 w 表示写，用 x 表示可执行可查找，用-表示无权限。

文件权限的修改方法：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。有两种修改方法，字母形式修改权限；数字形式修改权限。

```
1 #给a.sh添加可执行权限
2 sudo chmod +x a.sh
3 #给a.sh的所有者赋予读写执行权限，所属群组和其他人均赋予读和执行权限
4 sudo chmod 755 a.sh
```

Listing 5: 更改文件权限的两种方式

## 2.6 要求 6

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 用户）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是进行使用普通用户。

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，若没有指定他所属于的组，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。

## 2.7 要求 7

gcc, g++, cmake, 我的 Ubuntu 上默认是 g++, 它支持 c++11 标准。

# 3 第 3 题

## 3.1 要求 1

SLAM 会在以下场景中用到：

1. 增强现实 (AR)：使用收集或者平板电脑观察家中的家具，在房间里摆放虚拟装饰品等。
2. 自动驾驶定位：在室内及遮挡严重的室外环境中，GPS 定位精度低；高精度管道系统成本过高；基于无线信号的定位方案需要实现布置使用场景。而 SLAM 则可以直接使用相机或者激光雷达，在小场景范围内，无需预先布置场景的情况下对车辆进行定位。
3. 无人机建图和定位：无人机在飞行的过程中需要知道哪里有障碍物，该怎么规避，怎么重新规划路线。
4. 智能家居定位和建图：如扫地机需要使用二维的激光 SLAM 定位并构建家里的地图，规划扫地路线。

## 3.2 要求 2

定位：系统要明白自身的状态（即位置）；建图：系统也要了解外在的环境（即地图）定位和见图是紧密耦合的，根据 [1]：

Being precisely localized in an environment, a correct map is necessary, but in order to construct a good map it is necessary to be properly localized when elements are added to the map. 在一个环境中精确定位之后，一个正确的地图是必要的，但为了构建一个好的地图，当新的元素加到地图中时需要精准地定位。

### 3.3 要求 3

根据 [1], SLAM 可分为两阶段: 古典阶段和算法分析阶段:

1. 1986-2004 年, 古典年代, 这个阶段从 SLAM 的提出到建立完整的系统, 引入了 SLAM 概率论推导方法, 包括基于扩展卡尔曼滤波、粒子滤波和最大似然估计, 同时, 面临效率和数据关联鲁棒性问题的挑战;
2. 2004-2015 年, 算法分析年代, 这一阶段有许多包括可观测性, 收敛性和一致性的 SLAM 基本特性研究, 理解了稀疏特征在高效 SLAM 解决方案中的重要角色, 并且开发了许多开源 SLAM 库。

### 3.4 要求 4

在 2010 年出现基于视觉 SLAM 的方法之后将 SLAM 分为前端和后端。

总的来说, SLAM 本质上是对运动主体自身和周围环境空间不确定性的估计。前端根据状态估计理论, 把定位和建图的不确定性表达出来, 送给后端, 后端采用滤波器或非线性优化, 估计状态的均值何不确定性。

具体来说, 前端全称叫做“前端视觉里程计”(VO), 其任务是估算相邻图像间相机的运动, 以及局部地图的样子, 理论上来说, 一方面, 将相邻时刻的运动串起来, 就构成了机器人的运动轨迹, 从而解决了定位问题; 另一方面, 根据每个时刻的相机位置, 计算出各像素对应的空间点的位置, 就得到了地图。但是实际中, 仅通过视觉里程计来估计轨迹会出现累计漂移, 而为了解决漂移问题, 需要引入后端优化和回环检测, 其中, 后端全称叫后端非线性优化, 其接受不同是可视化里程计的相机位姿以及回环检测的信息, 对他们进行优化, 得到全局一致的轨迹和地图。

### 3.5 要求 5

1. MonoSLAM, 首个单目 V-SLAM 系统 [2].
2. PTAM, 首个基于关键帧 BA 的单目 V-SLAM 系统 [3].
3. LSD-SLAM, 单目直接法的代表系统 [4].

## 4 第 4 题

### 4.1 要求 1, 2, 3

工程组织如图 4.1 所示

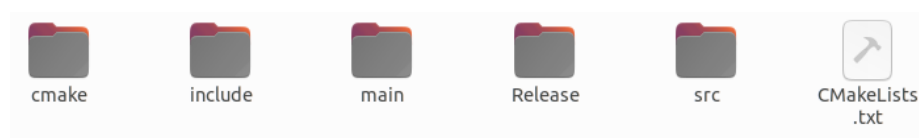


图 4.1 工程组织图

编译出来 sayhello 置于 main 文件夹下, 执行 sayhello 并输出, 如图 4.2 所示

```
wrk@MyPC:~/CLionProjects/CMakeLearning/cmake/CMake_practice/Release/main$ ls
CMakeFiles  cmake_install.cmake  Makefile  sayhello
wrk@MyPC:~/CLionProjects/CMakeLearning/cmake/CMake_practice/Release/main$ ./sayhello
Hello SLAM
wrk@MyPC:~/CLionProjects/CMakeLearning/cmake/CMake_practice/Release/main$
```

图 4.2 sayhello 输出

各部分代码如下:

```
1 mkdir Release && cd Release
2 cmake -DCMAKE_BUILD_TYPE=Release ..
```

Listing 6: 按照 Release 方式编译工程

```
1 PROJECT(CMAKE_)
2 ADD_SUBDIRECTORY(include)
3 ADD_SUBDIRECTORY(src)
4 ADD_SUBDIRECTORY(main)
5 ADD_SUBDIRECTORY(cmake)
```

Listing 7: 工程/CMakeLists.txt

```
1 INSTALL(FILES hello.h DESTINATION include/hello)
```

Listing 8: include/CMakeLists.txt

```
1 FIND_PATH(myHeader hello.h)
2 INCLUDE_DIRECTORIES(..include) #添加一个头文件搜索路径
3 SET(LIBHELLO_SRC hello.cpp)
4 ADD_LIBRARY(hello SHARED ${LIBHELLO_SRC})
5 ADD_LIBRARY(hello_static STATIC ${LIBHELLO_SRC})
6 SET_TARGET_PROPERTIES(hello_static PROPERTIES OUTPUT_NAME "hello")
7 GET_TARGET_PROPERTY(OUTPUT_VALUE hello_static OUTPUT_NAME)
8 MESSAGE(STATUS "this is the hello_static OUTPUT_NAME:${OUTPUT_VALUE}")
9 SET_TARGET_PROPERTIES(hello PROPERTIES VERSION 1.2 SOVERSION 1)
10 SET_TARGET_PROPERTIES(hello PROPERTIES CLEAN_DIRECT_OUTPUT 1)
11 SET_TARGET_PROPERTIES(hello_static PROPERTIES CLEAN_DIRECT_OUTPUT 1)
12 INSTALL(TARGETS hello hello_static LIBRARY DESTINATION lib ARCHIVE
    DESTINATION lib)
```

Listing 9: src/CMakeLists.txt

```
1 ADD_EXECUTABLE(sayhello useHello.cpp)
2 TARGET_LINK_LIBRARIES(sayhello hello)
3 INCLUDE_DIRECTORIES(..include) #添加一个头文件搜索路径
```

Listing 10: main/CMakeLists.txt

## 4.2 要求 4

库的安装结果如图 4.3 所示：



```

wrk@MyPC:~/CLionProjects/CMakeLearning/cmake/CMake_practice/Release$ sudo make install
[sudo] wrk 的密码:
CMake Warning (dev) at src/CMakeLists.txt:13:
  Syntax Warning in cmake code at column 55

  Argument not separated from preceding token by whitespace.
This warning is for project developers.  Use -Wno-dev to suppress it.

-- this is the hello_static OUTPUT_NAME:hello
CMake Warning (dev) in CMakeLists.txt:
  No cmake_minimum_required command is present.  A line of code such as

    cmake_minimum_required(VERSION 3.16)

  should be added at the top of the file.  The version specified may be lower
  if you wish to support older CMake versions for this project.  For more
  information run "cmake --help-policy CMP0000".
This warning is for project developers.  Use -Wno-dev to suppress it.

-- Configuring done
-- Generating done
-- Build files have been written to: /home/wrk/CLionProjects/CMakeLearning/cmake/CMake_practice/Release
[ 33%] Built target hello_static
[ 66%] Built target hello
[100%] Built target sayhello
Install the project...
-- Install configuration: ""
-- Up-to-date: /usr/local/include/hello/hello.h
-- Up-to-date: /usr/local/lib/libhello.so.1.2
-- Up-to-date: /usr/local/lib/libhello.so.1
-- Up-to-date: /usr/local/lib/libhello.so
-- Up-to-date: /usr/local/lib/libhello.a
-- Up-to-date: /usr/local/lib/cmake/hello/FindHELLO.cmake
wrk@MyPC:~/CLionProjects/CMakeLearning/cmake/CMake_practice/Release$

```

图 4.3 hello 库的安装结果

### 4.3 要求 5

hello 库的 FindHello.cmake 文件用于寻找库，

```

1 FIND_PATH(HELLO_INCLUDE_DIR hello.h /usr/include/hello /usr/local/include/
    hello)
2 FIND_LIBRARY(HELLO_LIBRARY NAMES hello PATH /usr/lib /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_FIND_REQUIRED)
12 MESSAGE(FATAL_ERROR "Could not find hello library")
13 ENDIF (HELLO_FIND_REQUIRED)
14 ENDIF (HELLO_FOUND)

```

Listing 11: hello 工程/cmake/FindHELLO.cmake

```

1 INSTALL(FILES FindHELLO.cmake DESTINATION lib/cmake/hello) #把FindHELLO.

```

cmake 安装到对应位置

Listing 12: hello 工程/cmake/CMakeLists.txt

建立工程 Find 如图 4.4 所示

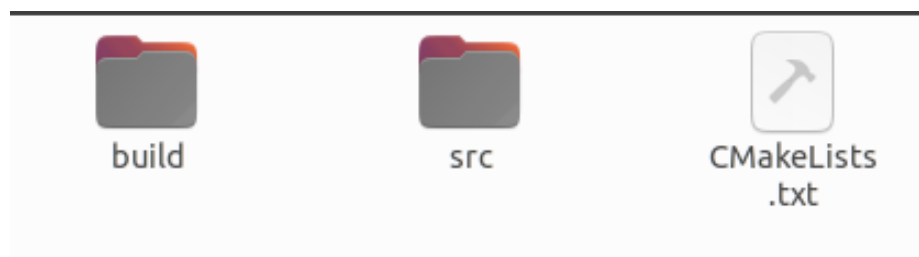


图 4.4 测试工程 Find 的工程组织

```
1 PROJECT(Find)
2 cmake_minimum_required(VERSION 3.16)
3 SET(HELLO_DIR /usr/local/lib) #库文件位置
4 SET(CMAKE_MODULE_PATH /usr/local/lib/cmake/hello) #FindHELLO.cmake 位置
5 ADD_SUBDIRECTORY(src)
```

Listing 13: 工程/CMakeLists.txt

```
1 FIND_PACKAGE(HELLO)
2 IF(HELLO_FOUND)
3     ADD_EXECUTABLE(helloexe main.cpp)
4     INCLUDE_DIRECTORIES(${HELLO_INCLUDE_DIR}) #添加头文件搜索路径
5     TARGET_LINK_LIBRARIES(helloexe ${HELLO_LIBRARY}) #链接
6
7     MESSAGE(STATUS "The HELLO_INCLUDE_DIR is:${HELLO_INCLUDE_DIR}")
8     MESSAGE(STATUS "The HELLO_LIBRARY is:${HELLO_LIBRARY}")
9 ENDIF(HELLO_FOUND)
```

Listing 14: src/CMakeLists.txt

```
1 #include "hello.h"
2 int main()
3 {
4     sayHello();
5     return 0;
6 }
```

Listing 15: src/CMakeLists.txt

测试过程和结果如图 4.5 所示

```

wrk@MyPC:~/CLionProjects/CMakeLearning/cmake/t8_FindHello$ mkdir build && cd build
wrk@MyPC:~/CLionProjects/CMakeLearning/cmake/t8_FindHello/build$ ls
wrk@MyPC:~/CLionProjects/CMakeLearning/cmake/t8_FindHello/build$ make ..
make: 对“.”无需做任何事。
wrk@MyPC:~/CLionProjects/CMakeLearning/cmake/t8_FindHello/build$ cmake ..
-- The C compiler identification is GNU 9.3.0
-- The CXX compiler identification is GNU 9.3.0
-- 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
CMake Warning (dev) at src/CMakeLists.txt:7:
  Syntax Warning in cmake code at column 44

  Argument not separated from preceding token by whitespace.
  This warning is for project developers.  Use -Wno-dev to suppress it.

CMake Warning (dev) at src/CMakeLists.txt:8:
  Syntax Warning in cmake code at column 40

  Argument not separated from preceding token by whitespace.
  This warning is for project developers.  Use -Wno-dev to suppress it.

-- Found Hello: /usr/local/lib/libhello.so
-- The HELLO_INCLUDE_DIR is:/usr/local/include/hello
-- The HELLO_LIBRARY is:/usr/local/lib/libhello.so
-- Configuring done
-- Generating done
-- Build files have been written to: /home/wrk/CLionProjects/CMakeLearning/cmake/t8_FindHello/build
wrk@MyPC:~/CLionProjects/CMakeLearning/cmake/t8_FindHello/build$ make
Scanning dependencies of target helloexe
[ 50%] Building CXX object src/CMakeFiles/helloexe.dir/main.cpp.o
[100%] Linking CXX executable helloexe
[100%] Built target helloexe
wrk@MyPC:~/CLionProjects/CMakeLearning/cmake/t8_FindHello/build$ ls
CMakeCache.txt  CMakeFiles  cmake_install.cmake  Makefile  src
wrk@MyPC:~/CLionProjects/CMakeLearning/cmake/t8_FindHello/build$ cd src/
wrk@MyPC:~/CLionProjects/CMakeLearning/cmake/t8_FindHello/build/src$ ls
CMakeFiles  cmake_install.cmake  helloexe  Makefile
wrk@MyPC:~/CLionProjects/CMakeLearning/cmake/t8_FindHello/build/src$ ./helloexe
Hello SLAM
wrk@MyPC:~/CLionProjects/CMakeLearning/cmake/t8_FindHello/build/src$

```

图 4.5 测试过程和结果

## 5 第 5 题

### 5.1 要求 1

gflags 安装

```

1 git clone https://github.com/gflags/gflags.git
2 cd gflags
3 mkdir build && cd build
4 cmake -DCMAKE_INSTALL_PREFIX=/usr/local -DBUILD_SHARED_LIBS=ON -
  DGFLAGS_NAMESPACE=gflags ../

```

```
5 make -j4
6 sudo make install
```

Listing 16: gflags 安装

## glog 安装

```
1 git clone https://github.com/google/glog
2 cd glog/
3 mkdir build && cd build
4 cmake ..
5 make -j4
6 sudo make install
```

Listing 17: glog 安装

gtest 安装: <https://github.com/google/googletest/releases> 下载 release-1.8.1.tar.gz

```
1 tar -xvzf googletest-release-1.8.1.tar.gz
2 cd googletest-release-1.8.1/
3 mkdir build && cd build
4 cmake ..
5 make
6 sudo make install
```

Listing 18: gtest 安装

## 5.2 要求 2,3

改成 glog 打印, 运用 gflags 传入打印次数 `print_times` 为 10 次并输出, 结果如图 5.1 所示

```

[100%] Built target gmock_main
wrk@MyPC:~/CLionProjects/CMakeLearning/cmake/t9_CMake_practice/build$ cd main/
wrk@MyPC:~/CLionProjects/CMakeLearning/cmake/t9_CMake_practice/build/main$ ls
CMakeFiles  cmake_install.cmake  Makefile  sayhello
wrk@MyPC:~/CLionProjects/CMakeLearning/cmake/t9_CMake_practice/build/main$ ./sayhello
I20220121 23:12:45.619333 76189 hello.cpp:13] Hello SLAM
wrk@MyPC:~/CLionProjects/CMakeLearning/cmake/t9_CMake_practice/build/main$ ./sayhello -print_times 10
I20220121 23:12:55.956351 76195 hello.cpp:13] Hello SLAM
I20220121 23:12:55.956480 76195 hello.cpp:13] Hello SLAM
I20220121 23:12:55.956493 76195 hello.cpp:13] Hello SLAM
I20220121 23:12:55.956506 76195 hello.cpp:13] Hello SLAM
I20220121 23:12:55.956524 76195 hello.cpp:13] Hello SLAM
I20220121 23:12:55.956537 76195 hello.cpp:13] Hello SLAM
I20220121 23:12:55.956549 76195 hello.cpp:13] Hello SLAM
I20220121 23:12:55.956560 76195 hello.cpp:13] Hello SLAM
I20220121 23:12:55.956576 76195 hello.cpp:13] Hello SLAM
I20220121 23:12:55.956588 76195 hello.cpp:13] Hello SLAM
wrk@MyPC:~/CLionProjects/CMakeLearning/cmake/t9_CMake_practice/build/mainwrk@Hwrwrk@Mywrk@Mywrk@MyPC:~/
wrk@MyPC:~/CLionProjects/CMakeLearning/cmake/t9_CMake_practice/build/main$ 

```

图 5.1 glog 打印，传入 gflags 参数测试结果

```

1 PROJECT(CMAKE_)
2 ADD_SUBDIRECTORY(include)
3 ADD_SUBDIRECTORY(src)
4 ADD_SUBDIRECTORY(main)
5 ADD_SUBDIRECTORY(cmake)
6 ADD_SUBDIRECTORY(test)

```

Listing 19: 工程/CMakeLists.txt

```

1 FIND_PACKAGE(gflags REQUIRED)
2 INSTALL(FILES hello.h DESTINATION include/hello)

```

Listing 20: include/CMakeLists.txt

```

1 #pragma once
2 #include <gflags/gflags.h>
3
4 DECLARE_int32(tmp);
5 void sayHello();
6 long my_fac(long n);

```

Listing 21: include/hello.h

```

1 INCLUDE_DIRECTORIES(..include) #添加一个头文件搜索路径
2
3 SET(LIBHELLO_SRC hello.cpp)
4 ADD_LIBRARY(hello SHARED ${LIBHELLO_SRC})
5 ADD_LIBRARY(hello_static STATIC ${LIBHELLO_SRC})
6 SET_TARGET_PROPERTIES(hello_static PROPERTIES OUTPUT_NAME "hello")
7 GET_TARGET_PROPERTY(OUTPUT_VALUE hello_static OUTPUT_NAME)
8 MESSAGE(STATUS "this is the hello_static OUTPUT_NAME:${OUTPUT_VALUE}")
9 SET_TARGET_PROPERTIES(hello PROPERTIES VERSION 1.2 SOVERSION 1)
10 SET_TARGET_PROPERTIES(hello PROPERTIES CLEAN_DIRECT_OUTPUT 1)

```

```

11 SET_TARGET_PROPERTIES(hello_static PROPERTIES CLEAN_DIRECT_OUTPUT 1)
12
13 FIND_PACKAGE(gflags REQUIRED) #找到gflags包
14 FIND_PACKAGE(glog REQUIRED)  #找到glog包
15
16 INSTALL(TARGETS hello hello_static LIBRARY DESTINATION lib ARCHIVE
    DESTINATION lib)

```

Listing 22: src/CMakeLists.txt

```

1 #include "hello.h"
2 #include <iostream>
3
4 #include <gflags/gflags.h>
5 #include <glog/logging.h>
6
7 DEFINE_int32(tmp, 100, "This is temp test value!");
8
9
10 void sayHello()
11 {
12     FLAGS_logtostderr = 1; //输出到控制台
13     LOG(INFO) <<"Hello SLAM";
14 }
15
16 long my_fac(long n)
17 {
18     if(n<1) return 1;
19     else    return n*my_fac(n-1);
20 }

```

Listing 23: src/hello.cpp

```

1 INCLUDE_DIRECTORIES(..include) #添加一个头文件搜索路径
2 FIND_PACKAGE(gflags REQUIRED) #找到gflags包
3 FIND_PACKAGE(glog REQUIRED)  #找到glog包
4 ADD_EXECUTABLE(sayhello useHello.cpp)
5 TARGET_LINK_LIBRARIES(sayhello hello gflags glog)

```

Listing 24: main/CMakeLists.txt

```

1 #include <iostream>
2 #include "hello.h"
3 #include <gflags/gflags.h>
4 #include <glog/logging.h>
5
6 DEFINE_int32(print_times, 1, "The print times"); #定义打印次数gflags

```

```

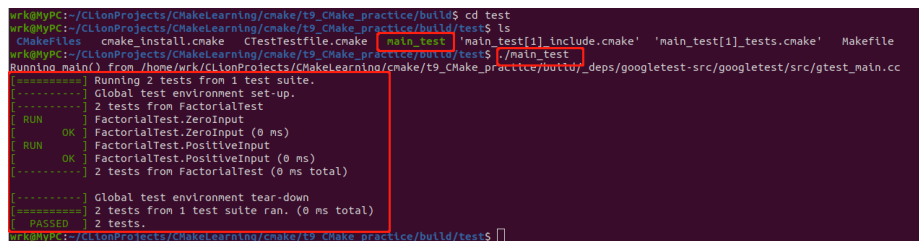
7
8 int main( int argc, char** argv )
9 {
10     gflags::ParseCommandLineFlags(&argc, &argv, true); //用于接受命令行的
        flag参数并更新默认参数
11     google::InitGoogleLogging("daqing"); //初始化一个log
12     FLAGS_logtostderr = 1; //输出到控制台
13     for(int i=FLAGS_print_times; i>0;--i)
14         sayHello();
15     google::ShutdownGoogleLogging(); //不用log时进行释放
16     return 0;
17 }

```

Listing 25: main/main.cpp

### 5.3 要求 4

在 hello.cpp 中写了一个阶乘的函数 my\_fac(long n), 放在 test/main\_test 中进行测试, 结果如图 5.2 所示



```

w@MyPC:~/CLionProjects/CHakeLearning/cmake/t9_CMake_practice/build$ cd test
w@MyPC:~/CLionProjects/CHakeLearning/cmake/t9_CMake_practice/build/test$ ls
CMakeFiles  cmake_install.cmake  CTestTestfile.cmake  main_test  'main_test[i].include.cmake'  'main_test[i].tests.cmake'  Makefile
w@MyPC:~/CLionProjects/CHakeLearning/cmake/t9_CMake_practice/build/test$ ./main_test
Running main() from /home/wrk/CLionProjects/CHakeLearning/cmake/t9_CMake_practice/build/_deps/googletest-src/googletest/src/gtest_main.cc
*****
Running 2 tests from 1 test suite.
*****
Global test environment set-up.
*****
2 tests from FactorialTest
RUN      OK      FactorialTest.ZeroInput (0 ms)
RUN      OK      FactorialTest.PositiveInput (0 ms)
*****
2 tests from FactorialTest (0 ms total)
*****
Global test environment tear-down
*****
2 tests from 1 test suite ran. (0 ms total)
*****
2 PASSED
w@MyPC:~/CLionProjects/CHakeLearning/cmake/t9_CMake_practice/build/test$

```

图 5.2 gtest 测试结果

```

1 INCLUDE_DIRECTORIES(..include) #hello.h 搜索路径
2 FIND_PACKAGE(gflags REQUIRED) #找到gflags包
3 FIND_PACKAGE(glog REQUIRED) #找到glog包
4
5 set(CMAKE_CXX_STANDARD 11)
6 include(FetchContent) #载入模块
7 FetchContent_Declare(
8     googletest
9     URL https://github.com/google/googletest/archive/609281088
        cfefc76f9d0ce82e1ff6c30cc3591e5.zip)
10

```

```
11 set(gtest_force_shared_crt ON CACHE BOOL "" FORCE)
12 FetchContent_MakeAvailable(googletest)
13
14 enable_testing()
15 add_executable( main_test main_test.cpp)
16 target_link_libraries(main_test gtest_main hello gflags glog)
17
18 #使CMake runner能够在二进制中找到测试，使用GoogleTest模块
19 include(GoogleTest)
20 gtest_discover_tests(main_test)
```

Listing 26: test/CMakeLists.txt

```
1 #include <gtest/gtest.h>
2 #include <gflags/gflags.h>
3 #include <glog/logging.h>
4 #include <hello.h>
5
6 TEST(FactorialTest, ZeroInput){
7     EXPECT_EQ(my_fac(0), 1);}
8
9 TEST(FactorialTest, PositiveInput){
10     EXPECT_EQ(my_fac(1), 1);
11     EXPECT_EQ(my_fac(2), 2);
12     EXPECT_EQ(my_fac(3), 6);}
```

Listing 27: test/main\_test.txt

整个工程树如图 5.3 所示



```
wrk@MyPC:~/CLionProjects/CMakeLearning/cmake/t9_CMake_practice$ tree -L 2
.
├── build
│   ├── bin
│   ├── cl.sh
│   ├── cmake
│   ├── CMakeCache.txt
│   ├── CMakeFiles
│   ├── cmake_install.cmake
│   ├── CTestTestfile.cmake
│   ├── _deps
│   ├── include
│   ├── lib
│   ├── main
│   ├── main_test[1]_include.cmake
│   ├── Makefile
│   ├── src
│   ├── test
│   └── Testing
├── cmake
│   ├── CMakeLists.txt
│   └── FindHELLO.cmake
├── CMakeLists.txt
├── include
│   ├── CMakeLists.txt
│   └── hello.h
├── main
│   ├── CMakeLists.txt
│   └── useHello.cpp
├── src
│   ├── CMakeLists.txt
│   └── hello.cpp
├── test
│   ├── CMakeLists.txt
│   └── main_test.cpp
└── 16 directories, 17 files
```

图 5.3 整个工程树

## 6 第 6 题

### 6.1 要求 1

下载 ORB-SLAM2 结果如图 6.1 所示

```

root@ubuntu:~/SLAM/Package$ git clone https://github.com/raulmur/ORB_SLAM2
正在克隆到 'ORB_SLAM2'...
remote: Enumerating objects: 566, done.
remote: Total 566 (delta 0), reused 0 (delta 0), pack-reused 566
接收对象中: 100% (566/566), 41.41 MiB | 3.20 MiB/s, 完成.
处理 delta 中: 100% (182/182), 完成.
root@ubuntu:~/SLAM/Package$ ls
gitlogs  glog  googletest-release-1.8.1  googletest-release-1.8.1.tar.gz  ORB_SLAM2
root@ubuntu:~/SLAM/Package$ cd ORB_SLAM2/
root@ubuntu:~/SLAM/Package/ORB_SLAM2$ ls
build_ros.sh  build.sh  CMakeLists.txt  cmake_modules  Dependencies.md  Examples  include  License-gpl.txt  LICENSE.txt  README.md  src  Thirdparty  Vocabulary
root@ubuntu:~/SLAM/Package/ORB_SLAM2$

```

图 6.1 下载 ORB-SLAM2 完成

## 6.2 要求 2

1. ORB-SLAM 将在 \$PROJECT\_SOURCE\_DIR/lib 文件夹下编译出 19 个动态库文件，在 \$PROJECT\_SOURCE\_DIR/Examples 文件夹下的 3 个子文件夹分别编译出 6 个可执行文件 rgb\_d\_tum, stereo\_kitti, stereo\_euroc, mono\_tum, mono\_kitti, mono\_euroc。
2. include 中包含了 ORB-SLAM2 建立的库的头文件（20 项）；src 中包含了 ORB-SLAM2 建立的库的源文件（19 项）；Examples 包含编译出的 6 个可执行文件，放在 3 个文件夹中，ROS 文件夹下放的推测应该是在 ROS 系统下能运行的 Demo。
3. 可执行文件链接了 5 个库：OpenCV, EIGEN3, Pangolin, libDBow2, libg2o。

## 7 第 7 题

### 7.1 要求 1

编译成功，如图 7.1 所示下载 ORB-SLAM2 结果如图 7.1 所示

```

/home/wrk/SLAM/PACKAGE/ORB_SLAM2/Thirdparty/g2o/g2o/types/./core/base_binary_edge.h:60:80:
ted [-Wdeprecated-declarations]
  60 |         typedef Eigen::Map<Matrix<double, Dj, Di>, Matrix<double, Dj, Di>::Flags & AL
BlockTransposedType;
      |
In file included from /usr/local/include/eigen3/Eigen/Core:366,
                  from /usr/local/include/pangolin/gl/gl.h:40,
                  from /usr/local/include/pangolin/pangolin.h:34,
                  from /home/wrk/SLAM/PACKAGE/ORB_SLAM2/include/MapDrawer.h:27,
                  from /home/wrk/SLAM/PACKAGE/ORB_SLAM2/include/Viewer.h:26,
                  from /home/wrk/SLAM/PACKAGE/ORB_SLAM2/include/Tracking.h:28,
                  from /home/wrk/SLAM/PACKAGE/ORB_SLAM2/include/System.h:29,
                  from /home/wrk/SLAM/PACKAGE/ORB_SLAM2/Examples/Monocular/mono_tum.cc:29:
/usr/local/include/eigen3/Eigen/src/Core/util/Constants.h:162:37: note: declared here
 162 | EIGEN_DEPRECATED const unsigned int AlignedBit = 0x80;
      |
[ 84%] Linking CXX executable ../Examples/Monocular/mono_kitti
[ 87%] Linking CXX executable ../Examples/Monocular/mono_euroc
[ 90%] Linking CXX executable ../Examples/RGB-D/rgbd_tum
[ 90%] Built target mono_euroc
[ 90%] Built target mono_kitti
[ 90%] Built target rgbd_tum
[ 93%] Linking CXX executable ../Examples/Stereo/stereo_kitti
[ 96%] Linking CXX executable ../Examples/Monocular/mono_tum
[100%] Linking CXX executable ../Examples/Stereo/stereo_euroc
[100%] Built target stereo_kitti
[100%] Built target mono_tum
[100%] Built target stereo_euroc
wrk@MyPC:~/SLAM/PACKAGE/ORB_SLAM2$

```

图 7.1 编译 ORB-SLAM2 成功

## 7.2 要求 2

在工程的 CMakeLists.txt 下添加这几句话：

```

1 set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${PROJECT_SOURCE_DIR}/Examples/Myvideo)
2 #给的视频
3 add_executable(myvideo
4   Examples/Myvideo/myvideo.cpp)
5 target_link_libraries(myvideo ${PROJECT_NAME})
6 #自己的相机
7 add_executable(myslam
8   Examples/Myvideo/myslam.cpp)
9 target_link_libraries(myslam ${PROJECT_NAME})

```

Listing 28: 修改的工程/CMakeLists.txt

对应的文件路径修改

```

1 string parameterFile = "./myvideo.yaml";
2 string vocFile = "../../Vocabulary/ORBvoc.txt";

```

Listing 29: MyVideo/myslam.cpp 部分修改

### 7.3 要求 3

运行自己电脑的摄像头结果如图 7.2 所示

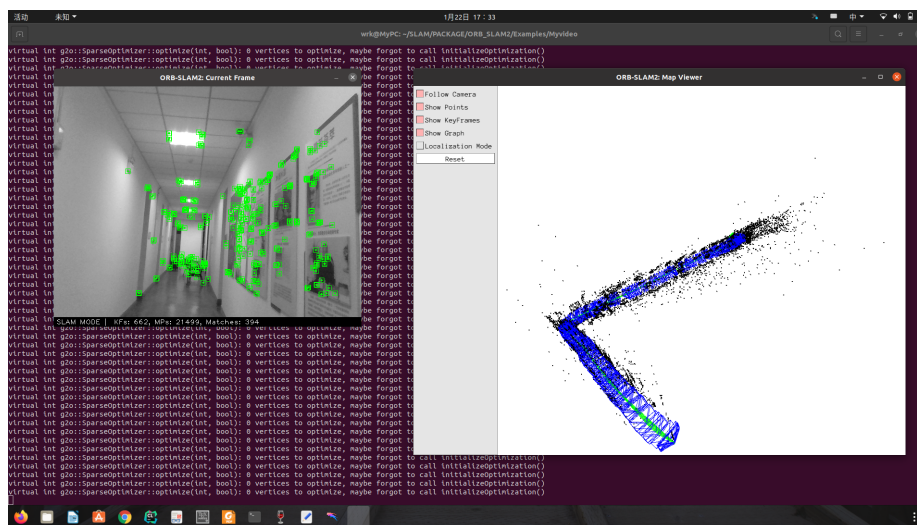


图 7.2 电脑摄像头运行 ORB-SLAM2

运行给的视频结果如图 7.3 所示

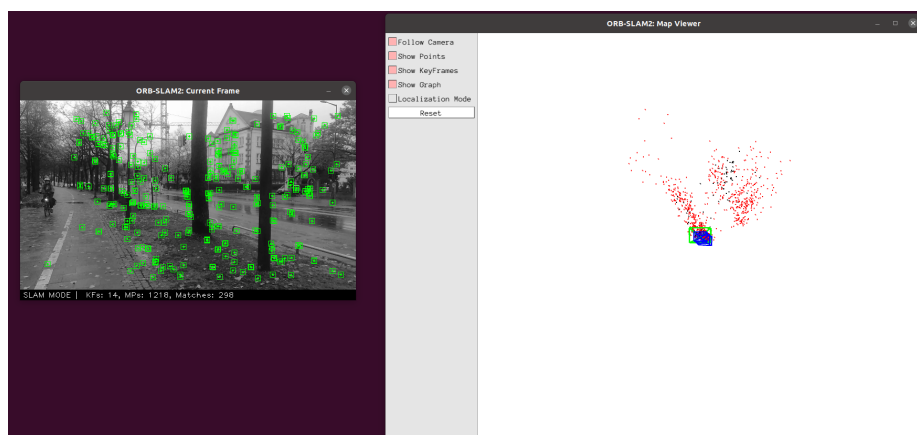


图 7.3 给的视频运行 ORB-SLAM2

运行体会：我尝试在实验楼构建一个带有回环的地图，但是走到白色的

墙比较多，特征点比较少的地方就丢掉了 tracking，怎么都过不去，比如如果进了隧道，什么都看不见，但是还是得能构建的出来地图，比如保持当前的方向，出来之后再进行补偿之类的，我想这应该是 ORB-SLAM2 一个缺陷吧，总之，第一次跑出 SLAM 还是挺开心的。

## 参考文献

- [1] J. Fuentes-Pacheco, J. Ruiz-Ascencio, and J. M. Rendón-Mancha, “Visual simultaneous localization and mapping: a survey,” *Artificial Intelligence Review*, vol. 43, no. 1, pp. 55–81, 2015.
- [2] Davison A J, Reid I D, Molton N D, et al. MonoSLAM: real- time single camera SLAM[J]. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2007, 29(6):1052-1067
- [3] Klein, Georg S. W. and David William Murray. “Parallel Tracking and Mapping for Small AR Workspaces.” 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality (2007): 225-234.
- [4] Engel, Jakob et al. “LSD-SLAM: Large-Scale Direct Monocular SLAM.” *ECCV* (2014).