2 熟悉 Linux (2 分,约 2 小时)

1. 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?

sudo apt-get install 软件名
或者 sudo dpkg -i 软件名.deb，还有编译安装二进制文件安装啊之类的

一般被安装在/usr 和/usr/local 下，也有在/opt 下的

2. linux 的环境变量是什么？我如何定义新的环境变量?

当我们使用指令的时候，比如说 ls，系统会依照环境变量 PATH 的设定去每一个 PATH 定义的目录下面寻找名为 ls 的可执行文件。

可以直接在 terminal 中使用 export 命令增加新的环境变量，比如 export PATH=$PATH:/myexecutable/。
PATH 中的路径通过：相隔，即这条命令的意思是在当前 PATH 的末尾加上 myexecutable 这个目录。这样通过 terminal 的修改立即生效但仅对本次本账户有效果，关闭 terminal 之后会失效。

另外可以修改当前用户目录下的~/.bashrc 文件，在文件末尾加上 export PATH=$PATH:/myexecutable/可以使得这次修改永久对当前用户有效。这种修改需要当前用户再打开 terminal，或者手动执行 source ~/.bashrc 使之生效。
同样可以修改 etc/bashrc etc/profile 等文件，这样的修改是系统级别的。

3. 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。

/为根目录，根目录下面包括
/bin,/sbin,/etc,/dev,/proc,/var,/tmp,/usr,/home,/boot,/lib,/opt,/mnt,/media,/src 这些目录。

其中 bin 是用户们使用的二进制可执行文件目录，比如 ls 就在这里面。Sbin 是系统管理员用的二进制可执行文件所在目录。Boot 是开机相关的引导文件目录。Etc 是配置文件目录。Dev 是设备文件目录。Proc 是进程相关。Var 是变量文件。Tmp 是临时文件。Usr 是用户的二进制啊库文件等的所在，它有自己的/usr/bin,/usr/sbin/user/lib,/usr/local。Home 目录是默认是放每个用户个人文件的地方，也就是~/。Lib 是系统库。Opt 是可选的附加程序目录。Mnt 是挂在目录。Media 是光碟啊软盘啊之类的设备目录。Src 是服务目录。

4. 假设我要给 a.sh 加上可执行权限，该输入什么命令?

以“给用户加可执行权限，而组和其他人只保留 read 权限”为例
sudo chmod u+x,g=r,o=r a.sh
另外也可以用数值 421 的组合来修改权限

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

只有 root 权限可以更改文件所有者和所在组，sudo chown xiang:xiang a.sh

3 SLAM 综述文献阅读 (3 分,约 3 小时)

1. *SLAM 会在哪些场合中用到？至少列举三个方向。*

   无人驾驶，无人车，无人机，无人船，AR，机器人，三维重建啊，总之需要理解环境并做出交互的场景都需要 SLAM。

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？*

   定位和建图是相辅相成的，毕竟定位是需要参考系的，而地图就是我们建的参考系。很多情况下，机器人或者人进入到一个陌生空间，是没有办法提前获得地图的。比如“好奇号”到了火星，我们现在没有其他技术可以提前测量好整个火星的地图，只能让火星车一边探索一边建图。

3. *SLAM 发展历史如何？我们可以将它划分成哪几个阶段？*

   最早的是 80 年代到 00 年代的基于概率论的 SLAM，比 EKF 啊粒子滤波啊这些基于贝叶斯滤波的 SLAM。

   然后 00 年代后期 SLAM 的架构因为一些经典算法的涌现而确定下来了。 可以把 SLAM 分为前后端，前端主要负责处理传感器数据，比如里程计相机激光雷达 IMU 乃至磁力计声纳等等；后端要负责建图，回环检测，重定位和优化等等。比如基于 graph 的 SLAM 就具备回环检测和全局优化的能力。

4. *列举三篇在 SLAM 领域的经典文献*

   Gmapping – Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective sampling.

   Cartographer – real-time loop closure in 2d lidar slam

   PATM – parallel tracking and mapping for small AR workspace

   ORB-SLAM – ORB-SLAM: a versatile and accurate monocular SLAM system

4 CMake 练习 (2 分,约 1.5 小时)

1. *请按照上述要求组织源代码文件,并书写 CMakeLists.txt。*

   文件结构

   ```
   cmake_pratice/build
                 /CMakeLists.txt
                 /include
                       /hello.h
                 /src
                       /CMakeLists.txt
                       /hello.cpp
                       /useHello.cpp
   ```

   其中 cmake_pratice/CMakeLists.txt 内容:

   ```
   cmake_minimum_required(VERSION 2.7)
   PROJECT(HOMEWORK)
   ADD_SUBDIRECTORY(src)
   ```

   而 cmake_pratice/src/CMakeLists.txt 内容:

```
INCLUDE_DIRECTORIES(inlcude)

SET(LIB_HELLO_SRC hello.cpp)
SET(LIBRARY_OUT_PATH ${PROJECT_BINARY_DIR}/lib)

ADD_LIBRARY(hello SHARED ${LIB_HELLO_SRC})

INSTALL(TARGETS hello LIBRARY DESTINATION lib)
INSTALL(FILES ${PROJECT_SOURCE_DIR}/include/hello.h DESTINATION include)

SET(EXE_HELLO_SRC useHello.cpp)
set(CMAKE_BUILD_TYPE "Release")
ADD_EXECUTABLE(main ${EXE_HELLO_SRC})
TARGET_LINK_LIBRARIES(main hello)
```

运行结果:

cmake ..

make

```
xin@ubuntu16:~/VSLAM-course/ch1/cmake_pratice/build$ make
Scanning dependencies of target hello
[ 25%] Building CXX object src/CMakeFiles/hello.dir/hello.cpp.o
[ 50%] Linking CXX shared library libhello.so
[ 50%] Built target hello
Scanning dependencies of target main
[ 75%] Building CXX object src/CMakeFiles/main.dir/useHello.cpp.o
[100%] Linking CXX executable main
[100%] Built target main
```

sudo make install

```
xin@ubuntu16:~/VSLAM-course/ch1/cmake_pratice/build$ sudo make install
[sudo] password for xin:
[ 50%] Built target hello
[100%] Built target main
Install the project...
-- Install configuration: ""
-- Installing: /usr/local/lib/libhello.so
-- Up-to-date: /usr/local/include/hello.h
```

5 理解 ORB-SLAM2 框架

1. *下载完成后,请给出终端截图。*

```
xin@ubuntu16:~/Workspace$ 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 | 5.49 MiB/s, done.
Resolving deltas: 100% (177/177), done.
Checking connectivity... done.
```

2.

a. *ORB-SLAM2 将编译出什么结果?有几个库文件和可执行文件?*

编译结果包含一个 libORB_SLAM2.so 库文件和 6 个可执行文件：mono_tum,rgbd_tum,stereo_kitti,stereo_euroc,mono_euroc,mono_kitti

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

Include 下是各种头文件

src 下是源码，用来编译出 orbslam 库的

example 包含各个例子的源代码，各个例子编译出来的可执行文件也在这里

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?

编译出来的可执行文件都 link 的是 liborb_slam2.so 库

但是 liborb_slam2.so 这个库在被编译的时候又 link 了

opencv_libs

eigen3_libs

libDBow2.so

libg2o.so

6* 使用摄像头或视频运行 ORB-SLAM2(3 分,约 1 小时)

1. 请给出它编译完成的截图。

```
xin@ubuntu16:~/Workspace/ORB_SLAM2/build$ make -j8
Scanning dependencies of target ORB_SLAM2
[  3%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/Tracking.cc.o
[  6%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/LoopClosing.cc.o
[  9%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/ORBextractor.cc.o
[ 12%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/LocalMapping.cc.o
[ 15%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/System.cc.o
[ 18%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/ORBmatcher.cc.o
[ 21%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/FrameDrawer.cc.o
[ 25%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/Converter.cc.o
[ 28%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/MapPoint.cc.o
[ 31%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/KeyFrame.cc.o
[ 34%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/Map.cc.o
[ 37%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/MapDrawer.cc.o
[ 40%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/Optimizer.cc.o
[ 43%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/PnPsolver.cc.o
[ 46%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/Frame.cc.o
[ 50%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/KeyFrameDatabase.cc.o
[ 53%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/Sim3Solver.cc.o
[ 56%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/Initializer.cc.o
[ 59%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/Viewer.cc.o
[ 62%] Linking CXX shared library ../lib/libORB_SLAM2.so
[ 62%] Built target ORB_SLAM2
Scanning dependencies of target stereo_kitti
Scanning dependencies of target rgbd_tum
Scanning dependencies of target stereo_euroc
Scanning dependencies of target mono_tum
Scanning dependencies of target mono_kitti
Scanning dependencies of target mono_euroc
[ 65%] Building CXX object CMakeFiles/stereo_kitti.dir/Examples/Stereo/stereo_kitti.cc.o
[ 68%] Building CXX object CMakeFiles/mono_tum.dir/Examples/Monocular/mono_tum.cc.o
[ 71%] Building CXX object CMakeFiles/mono_euroc.dir/Examples/Monocular/mono_euroc.cc.o
[ 75%] Building CXX object CMakeFiles/rgbd_tum.dir/Examples/RGB-D/rgbd_tum.cc.o
[ 78%] Building CXX object CMakeFiles/mono_kitti.dir/Examples/Monocular/mono_kitti.cc.o
[ 81%] Building CXX object CMakeFiles/stereo_euroc.dir/Examples/Stereo/stereo_euroc.cc.o
[ 84%] Linking CXX executable ../Examples/Monocular/mono_tum
[ 84%] Built target mono_tum
[ 87%] Linking CXX executable ../Examples/RGB-D/rgbd_tum
[ 90%] Linking CXX executable ../Examples/Stereo/stereo_kitti
[ 93%] Linking CXX executable ../Examples/Stereo/stereo_euroc
[ 93%] Built target rgbd_tum
[ 96%] Linking CXX executable ../Examples/Monocular/mono_euroc
[ 96%] Built target stereo_kitti
[ 96%] Built target stereo_euroc
[ 96%] Built target mono_euroc
[100%] Linking CXX executable ../Examples/Monocular/mono_kitti
[100%] Built target mono_kitti
```

2. 请给出你的 CMakeLists.txt 修改方案。

在 example 中创建文件夹“shenlan_vslam”，把 myslam.cpp 进去。

Cmakelist.txt 最下面加入

```
set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${PROJECT_SOURCE_DIR}/Examples/Shenlan)
add_executable(shenlan_vslam
Examples/Shenlan/myslam.cpp)
target_link_libraries(shenlan_vslam ${PROJECT_NAME})
```

3. 请给出运行截图,并谈谈你在运行过程中的体会。

每次都要加载词库不是很方便。