

2018 年第 1 次课习题答案

——Vance 吴方熠

2. 熟悉 Linux

2.1. 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？

用 apt 工具安装软件命令：`sudo apt-get install <packagename>`

.deb 格式软件包安装命令：`sudo dpkg -i <package_file.deb>`

系统默认软件的可执行文件一般安装在 `/usr/share` 目录下。用户自行下载安装的软件一般安装在 `/usr/local/share` 目录下，也有安装在 `/opt` 中的。

2.2. linux 的环境变量是什么？我如何定义新的环境变量？

环境变量是在操作系统中一个具有特定名字的对象，它包含了一个或多个应用程序将使用到的信息。Linux 是一个多用户的操作系统，每个用户登录系统时都会有一个专用的运行环境，通常情况下每个用户的默认的环境都是相同的。这个默认环境就是一组环境变量的定义。每个用户都可以通过修改环境变量的方式对自己的运行环境进行配置。

环境变量通常以大写字符来表示。要定义新的环境变量，可用指令“`export <variate_name>=<value>`”，例如“`export SLAM=hello\ SLAM`”

定义（环境）变量要遵循几个规则：

1. 变量与变量内容以一个等号 `=` 来连结，如下所示：

`SLAM=http://www.slamcn.org`

2. 等号两边不能直接接空格符，如下所示为错误：

`SLAM = http://www.slamcn.org`（错误） 或 `SLAM=hello SLAM`（错误）

3. 变量名称只能是英文字母与数字，但是开头字符不能是数字，如下为错误：

`2SLAM=http://www.slamcn.org`（错误）

4. 变量内容若有空格符可使用双引号 `"` 或单引号 `'` 将变量内容结合起来，但双引号内的特殊字符如 `$` 等，可以保有原本的特性，单引号内的特殊字符则仅为一般字符（纯文本）。

5. 可用跳脱字符 `\` 将特殊符号（如 `[Enter]`，`$`，`\`，空格符，`'` 等）发成一般字符；

6. 若该变量为扩增变量内容时，则可用“\$变量名称”或 \${变量} 累加内容，如下所示：

```
PATH="$PATH":/home/bin
```

2.3. 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。

位于根目录(/)下的常见目录列表如下：

/bin - 重要的二进制 (binary) 应用程序

/boot - 启动 (boot) 配置文件

/dev - 设备 (device) 文件

/etc - 配置文件、启动脚本等 (etc)

/home - 本地用户主 (home) 目录

/lib - 系统库 (libraries) 文件

/media - 挂载可移动介质 (media)，诸如 CD、数码相机等

/mnt - 挂载 (mounted) 文件系统

/opt - 提供一个供可选的 (optional) 应用程序安装目录

/proc - 特殊的动态目录，用以维护系统信息和状态，包括当前运行中进程 (processes) 信息。

/sbin - 重要的系统二进制 (system binaries) 文件

/sys - 系统 (system) 文件

/tmp - 临时 (temporary) 文件

/usr - 包含绝大部分所有用户 (users) 都能访问的应用程序和文件

/var - 经常变化的 (variable) 文件，诸如日志或数据库等

2.4. 假设我要给 a.sh 加上可执行权限，该输入什么命令？

```
chmod a+x a.sh
```

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

```
chown xiang:xiang a.sh
```

3. SLAM 综述文献阅读

3.1. SLAM 会在哪些场合中用到？至少列举三个方向。

机器人导航，无人驾驶，排危搜救，虚拟现实/增强现实，三维场景重建，环境测绘等。

3.2. 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？

定位确定了机器人在一个环境中的具体位置，侧重对自身的了解；建图将环境的观察结果整合到一个连续的模型中，侧重对外在的了解。

定位和建图是相互依赖的，准确的定位离不开一个正确的地图；要构建精度更高的地图，则离不开准确的定位。只有定位没有建图，定位误差会随着时间和移动距离的增加而逐渐积累加大，如果有地图的存在，可以通过回环检测帮助机器人及时消除累计误差，提高定位精度，同时建立的地图也会更加准确。

3.3. SLAM 发展历史如何？我们可以将它划分成哪几个阶段？

同时定位与地图构建(simultaneous localization and mapping, SLAM)最早源于机器人领域，其目标是在一个未知的环境中实时重建环境的三维结构并同时为机器人自身进行定位。SLAM 在过去 30 年中取得了惊人的进步，实现了大规模的实际应用，并正向工业级应用稳步转变中。

我们可以将 SLAM 的发展分为以下两个阶段：

- 古典时代 *Classical age* (1986-2004)，古典时代引入了 SLAM 的主要概率公式，包括基于扩展卡尔曼滤波器，粒子滤波器和最大似然估计的方法。此外，它还描述了与效率和大数据关联相关的基本挑战
- 算法分析时代 *Algorithmic-analysis age* (2004-2015 年)，算法分析期间研究了 SLAM 的基本属性，包括可观察性，收敛性和一致性。在这个时期，对稀疏性对高效 SLAM 解算器的关键作用也被理解了，诞生了主要的开源 SLAM 库。

近几年 SLAM 的研究热点与发展趋势有以下几种：缓解特征依赖，稠密三维重建，多传感器融合，结合深度学习等等。

3. 4. 列举三篇在 SLAM 领域的经典文献

【1】A. J. Davison, I. D. Reid, N. D. Molton and O. Stasse, "MonoSLAM: Real-Time Single Camera SLAM," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 29, no. 6, pp. 1052-1067, June 2007.

【2】G. Klein and D. Murray, "Parallel Tracking and Mapping for Small AR Workspaces," 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality, Nara, 2007, pp. 225-234.

【3】R. Mur-Artal, J. M. M. Montiel and J. D. Tardós, "ORB-SLAM: A Versatile and Accurate Monocular SLAM System," in IEEE Transactions on Robotics, vol. 31, no. 5, pp. 1147-1163, Oct. 2015.

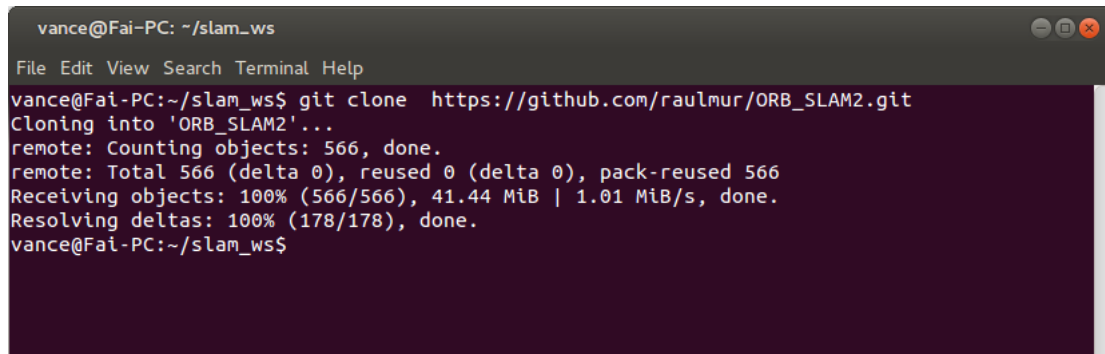
4. CMake 练习

4. 1. 书写一个由 CMake 组织的 C++工程，请按照要求组织源代码文件，并书写 CMakeLists.txt。

```
1  CMAKE_MINIMUM_REQUIRED(VERSION 2.8)
2
3  PROJECT(hello)
4
5  IF(NOT CMAKE_BUILD_TYPE)
6      SET(CMAKE_BUILD_TYPE Release)
7  ENDIF()
8
9  INCLUDE_DIRECTORIES(${PROJECT_SOURCE_DIR}/include)
10
11 SET(CMAKE_LIBRARY_OUTPUT_DIRECTORY
12     ${PROJECT_SOURCE_DIR}/lib)
13 ADD_LIBRARY(hello SHARED
14     include/hello.h
15     src/hello.cc
16 )
17
18 SET(CMAKE_RUNTIME_OUTPUT_DIRECTORY
19     ${PROJECT_SOURCE_DIR}/bin)
20 ADD_EXECUTABLE(sayhello src/useHello.cc)
21 TARGET_LINK_LIBRARIES(sayhello hello)
22
23 INSTALL(FILES include/hello.h DESTINATION include)
24 INSTALL(TARGETS hello LIBRARY DESTINATION lib)
```

5. 理解 ORB-SLAM2 框架

5.1. 从 github.com 下载 ORB-SLAM2 的代码，下载完成后，请给出终端截图



```
vance@Fai-PC: ~/slam_ws
File Edit View Search Terminal Help
vance@Fai-PC:~/slam_ws$ git clone https://github.com/raulmur/ORB_SLAM2.git
Cloning into 'ORB_SLAM2'...
remote: Counting objects: 566, done.
remote: Total 566 (delta 0), reused 0 (delta 0), pack-reused 566
Receiving objects: 100% (566/566), 41.44 MiB | 1.01 MiB/s, done.
Resolving deltas: 100% (178/178), done.
vance@Fai-PC:~/slam_ws$
```

5.2. (a) ORB-SLAM2 将编译出什么结果？有几个库文件和可执行文件？

由 *CMakeLists.txt* 文件可知，ORB-SLAM2 采用 Release 编译模式，包含了 C++ 11 的特性，依赖于 OpenCV (2.4.3 以上版本)，Eigen (3.1.0 以上版本) 和 Pangolin

由以下代码片段可知该工程生成了一个名为“ORB_SLAM2”的分享库文件：

```
cmake_minimum_required(VERSION 2.8)
project(ORB_SLAM2)
...
add_library(${PROJECT_NAME} SHARED
src/System.cc
src/Tracking.cc
...)
```

由以下代码片段可知工程分别在 3 个子文件夹 Examples/RGB-D, Examples/Stereo, Examples/Monocular 内生成了 1/2/3 个可执行文件，分别用于生成深度相机、双目相机和单目相机的实例。

```
set(CMAKE_RUNTIME_OUTPUT_DIRECTORY
${PROJECT_SOURCE_DIR}/Examples/RGB-D)

add_executable(rgbd_tum Examples/RGB-D/rgbd_tum.cc)
target_link_libraries(rgbd_tum ${PROJECT_NAME})

set(CMAKE_RUNTIME_OUTPUT_DIRECTORY
${PROJECT_SOURCE_DIR}/Examples/Stereo)
...
add_executable(mono_euroc Examples/Monocular/mono_euroc.cc)
target_link_libraries(mono_euroc ${PROJECT_NAME})
```

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

include 文件夹包含了工程的所有头文件。

src 文件夹包含了工程头文件的实现。

Examples 文件夹包含了三种传感器（单目、双目、深度相机）的可执行程序源文件，一些和数据集（TUM, KITTI）以及和工程运行相关的参数文件，以及用 ROS 定义的 ORB-SLAM2 包的相关文件。

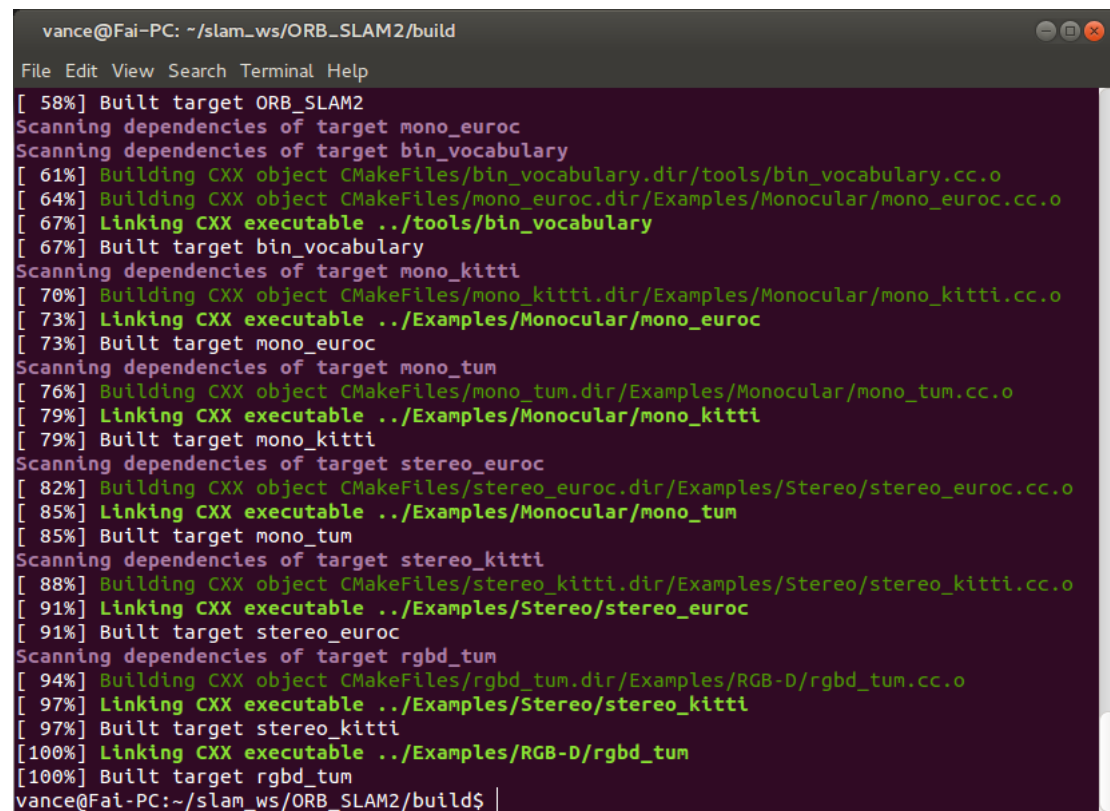
5.2. (c) ORB-SLAM2 中的可执行文件链接到了哪些库？它们的名字是什么？

由以下代码片段可知该工程的可执行文件链接到了由本工程生成的分享库，而由由本工程生成的分享库又链接到了 OpenCV, Eigen3, Pangolin, DBoW2, g2o 这 5 个库。

```
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
)
...
target_link_libraries(rgbd_tum ${PROJECT_NAME})
...
```

6. * 使用摄像头或视频运行 ORB-SLAM2

6.1. 请给出编译完成 ORB-SLAM2 的截图



```
vance@Fai-PC: ~/slam_ws/ORB_SLAM2/build
File Edit View Search Terminal Help
[ 58%] Built target ORB_SLAM2
Scanning dependencies of target mono_euroc
Scanning dependencies of target bin_vocabulary
[ 61%] Building CXX object CMakeFiles/bin_vocabulary.dir/tools/bin_vocabulary.cc.o
[ 64%] Building CXX object CMakeFiles/mono_euroc.dir/Examples/Monocular/mono_euroc.cc.o
[ 67%] Linking CXX executable ../tools/bin_vocabulary
[ 67%] Built target bin_vocabulary
Scanning dependencies of target mono_kitti
[ 70%] Building CXX object CMakeFiles/mono_kitti.dir/Examples/Monocular/mono_kitti.cc.o
[ 73%] Linking CXX executable ../Examples/Monocular/mono_euroc
[ 73%] Built target mono_euroc
Scanning dependencies of target mono_tum
[ 76%] Building CXX object CMakeFiles/mono_tum.dir/Examples/Monocular/mono_tum.cc.o
[ 79%] Linking CXX executable ../Examples/Monocular/mono_kitti
[ 79%] Built target mono_kitti
Scanning dependencies of target stereo_euroc
[ 82%] Building CXX object CMakeFiles/stereo_euroc.dir/Examples/Stereo/stereo_euroc.cc.o
[ 85%] Linking CXX executable ../Examples/Monocular/mono_tum
[ 85%] Built target mono_tum
Scanning dependencies of target stereo_kitti
[ 88%] Building CXX object CMakeFiles/stereo_kitti.dir/Examples/Stereo/stereo_kitti.cc.o
[ 91%] Linking CXX executable ../Examples/Stereo/stereo_euroc
[ 91%] Built target stereo_euroc
Scanning dependencies of target rgb_d_tum
[ 94%] Building CXX object CMakeFiles/rgb_d_tum.dir/Examples/RGB-D/rgb_d_tum.cc.o
[ 97%] Linking CXX executable ../Examples/Stereo/stereo_kitti
[ 97%] Built target stereo_kitti
[100%] Linking CXX executable ../Examples/RGB-D/rgb_d_tum
[100%] Built target rgb_d_tum
vance@Fai-PC:~/slam_ws/ORB_SLAM2/build$ |
```

6.2. 如何将 myslam.cpp 或 myvideo.cpp 加入到 ORB-SLAM2 工程中？请给出你的 CMakeLists.txt 修改方案

在 ORB-SLAM2 工程内新建文件夹 homework, 并将 myvideo.cpp, myvideo.mp4, myvideo.yaml 三个文件复制其中, 如 6.3 中截图右下角所示。

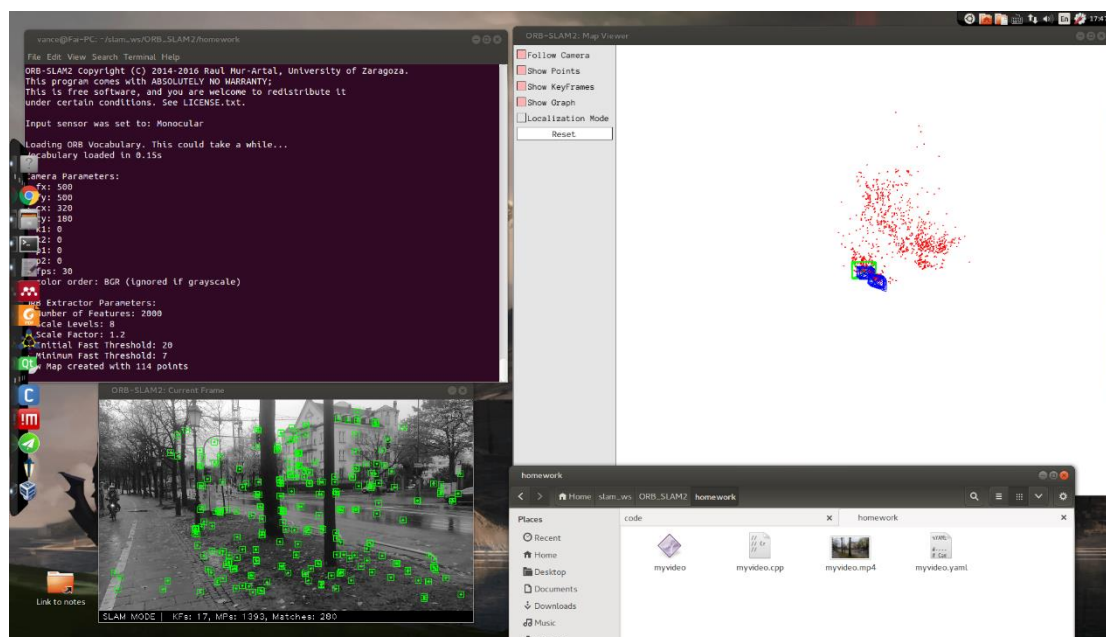
修改 myvideo.cpp 文件代码中相关的路径, 在 CMakeLists.txt 文件末尾添加以下代码片段:

```
# Class homework
set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${PROJECT_SOURCE_DIR}/homework)

add_executable(myvideo homework/myvideo.cpp)
target_link_libraries(myvideo ${PROJECT_NAME})
```

编译后即可运行。

6.3. 用 `myslam.yaml` (`myvideo.yaml`) 这个文件让 ORB-SLAM2 运行起来，看看 ORB-SLAM2 的实际效果。请给出运行截图，并谈谈你在运行过程中的体会。



心得体会：

ORB-SLAM2 效果好，代码风格规范，注释清晰，很适合学习研读；

对新手来说，项目移植会碰到很多问题，要提前学习一些必要的基础知识，比如 Linux 基本操作，CMake 等。上手以后要写出自己的代码，还要学会用一款 IDE 调试代码，学会自己解决 BUG。

一个庞大的项目会有很多的运行参数，将这些参数统一定义到一个参数文件内，通过文件读取的方式来获取参数的值，有助于减少调试的复杂度，这样就不需要每改一次参数就重新编译一次程序。

SLAM 是个庞大的工程，要学好 SLAM 不仅要有扎实的数学基础，还要有过硬编程能力，任重而道远。