

# 第 1 章作业

聂欣雨

2019 年 11 月 10 日

## 1 熟悉 Linux

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?

答: 使用 `sudo apt-get install xxx` 命令或 `sudo apt install xxx` 命令安装软件。若是下载了安装包 (deb 文件), 可以使用 `sudo dpkg -i install xxx.deb` 安装软件。通常一个软件的不同部分被放置在不同的位置:

- 程序的文档->/usr/share/doc;/usr/local/share/doc
- 程序->/usr/share;/usr/local/share
- 程序的启动项->/usr/share/apps;/usr/local/share
- 程序的语言包->/usr/share/locale;/usr/local/share/locale
- 可执行文件->/usr/bin;/usr/local/bin

有的软件会选择放在 /opt 目录中, 但目录结构和在 /usr 或 /usr/local 中相同。图 1 是使用 dpkg 安装的百度云软件的部分存放位置, 可以看出, 与 windows 不同, 在 Ubuntu 上安装的软件文件较为分散。

```
/usr/share/icons/hicolor/48x48/apps
/usr/share/icons/hicolor/48x48/apps/baidunetdisk.png
/usr/share/icons/hicolor/32x32
/usr/share/icons/hicolor/32x32/apps
/usr/share/icons/hicolor/32x32/apps/baidunetdisk.png
/usr/share/icons/hicolor/24x24
/usr/share/icons/hicolor/24x24/apps
/usr/share/icons/hicolor/24x24/apps/baidunetdisk.png
/usr/share/icons/hicolor/16x16
/usr/share/icons/hicolor/16x16/apps
/usr/share/icons/hicolor/16x16/apps/baidunetdisk.png
/opt
/opt/baidunetdisk
/opt/baidunetdisk/version
/opt/baidunetdisk/v8_context_snapshot.bin
/opt/baidunetdisk/swiftshader
/opt/baidunetdisk/swiftshader/libGLESv2.so
/opt/baidunetdisk/swiftshader/libEGL.so
/opt/baidunetdisk/snapshot_blob.bin
/opt/baidunetdisk/resources.pak
/opt/baidunetdisk/resources
/opt/baidunetdisk/resources/electron.asar
```

图 1: 百度云文件分布截图

2. linux 的环境变量是什么? 我如何定义新的环境变量?

答: 环境变量是一组保存了系统或程序运行时所需参数信息的变量, 常见的环境变量有 PATH 和 HOME 等。对于不同的作用域, 有不同的方法定义环境变量:

- 当前终端: `export VAR_NAME=value`

- 当前用户：将 `export VAR_NAME=value` 添加到 `~/.bashrc` 文件中，并执行 `source ~/.bashrc` 命令或重启
  - 所有用户：将 `export VAR_NAME=value` 添加到 `/etc/profile` 文件中，并执行 `source /etc/profile` 命令或重启
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。
- 答：一般根目录下有 `bin`、`boot`、`dev`、`etc`、`home`、`lib`、`media`、`mnt`、`opt`、`root`、`sbin`、`tmp` 等目录。
- `bin` 目录，存放能被 `root` 账号和一般账号执行的可执行文件，如 `cat`、`chmod`、`cp` 和 `mv` 等命令。
  - `home` 目录，一般用户的家目录都存放在 `/home` 目录之中。
  - `media` 目录，一般将硬盘、光盘等设备挂载在这里。
  - `root` 目录，`root` 用户的家目录，除 `root` 用户外，其他用户均没有 `rw` 权限。
  - `tmp` 目录，一般用户或程序放置临时文件的地方。
4. 假设我要给 `a.sh` 加上可执行权限，该输入什么命令？
- 答：
- 对文件所属用户加上可执行权限：`sudo chmod u+x a.sh`
  - 对文件所属组群的用户加上可执行权限：`sudo chmod g+x a.sh`
  - 对其他用户加上可执行权限：`sudo chmod o+x a.sh`
  - 简单粗暴的方法：`sudo chmod 777 a.sh`
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？
- 答：`sudo chown xiang:xiang a.sh`

## 2 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。
- 答：
- 增强现实（Augmented Reality, AR）。AR 技术需要实施定位设备在环境中的方位。
  - 智能移动机器人。在没有足够外部信息（已有的地图、足够精确的 GPS 定位信息等）的情况下，移动机器人往往需要对未知的环境进行建模并同时对自身进行定位。

- 无人驾驶。SLAM 能够帮助无人车感知周围环境，更好地完成导航、避障、路径规划等任务。

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？

答：定位和建图是这两个任务是协同合作、互相依赖的关系。只有在正确的地图中，机器人才能够被准确地定位；而要搭建一个正确的地图，则需要对新加入地图的元素进行精确地定位。

在定位的同时需要建图，主要有两点原因：首先，许多其他任务都需要地图，比如路径规划或提供可视化地图给用户以进行决策。其次，通过地图，可以进行回环检测（loop closure）减小或消除漂移带来的累计误差。

3. SLAM 发展历史如何？我们可以将它划分成哪几个阶段？

答：概率 SLAM 起源于 1986 年召开的 IEEE Robotics and Automation Conference。在这之后的一段时间内，人们将定位和建图作为两个问题分开处理。在 1995 年，SLAM 问题在理论上有了新的突破，人们开始把定位和建图结合在一起，并且提出了及时定位与建图（SLAM）的概念。在这之前，SLAM 又被称为 Concurrent Mapping and Localization (CML)。进入 2000 年之后，SLAM 问题不再局限于激光测距仪，视觉传感器（单目、双目、深度摄像头）开始受到欢迎。并且，人们不再局限于使用基于马克洛夫假设的贝叶斯滤波框架对 SLAM 进行建模，优化的方法逐渐占据了主流。

4. 列举三篇在 SLAM 领域的经典文献。

答：

- LSD-SLAM: Large-Scale Direct Monocular SLAM
- ORB-SLAM: A Versatile and Accurate Monocular SLAM System
- VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator

### 3 CMake 练习

更改两个 c 文件 hello.c、useHello.c 的扩展名为 cpp，按照图 2 的方式组织工程文件。

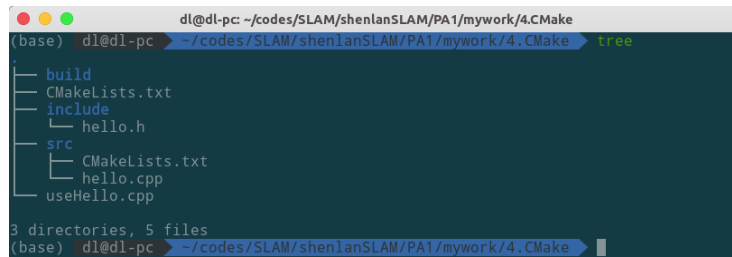


图 2: 项目文件组织图

其中, src/CMakeLists.txt 文件内容如下:

```
# 编译 hello.cpp 为 libhello.so
add_library(hello SHARED hello.cpp)
# 将 libhello.so 安装至安装路径的 lib 目录下
install(TARGETS hello DESTINATION lib)
```

CMakeLists.txt 文件内容如下:

```
project(hello)
cmake_minimum_required(VERSION 2.8)
# 设置默认编译类型为 Release
if(NOT CMAKE_BUILD_TYPE)
set(CMAKE_BUILD_TYPE Release)
endif()
# 设置默认编译路径为 /usr/local
if(CMAKE_INSTALL_PREFIX_INITIALIZED_TO_DEFAULT)
set(CMAKE_INSTALL_PREFIX /usr/local CACHE PATH
    ↪ "default install path" FORCE)
endif()

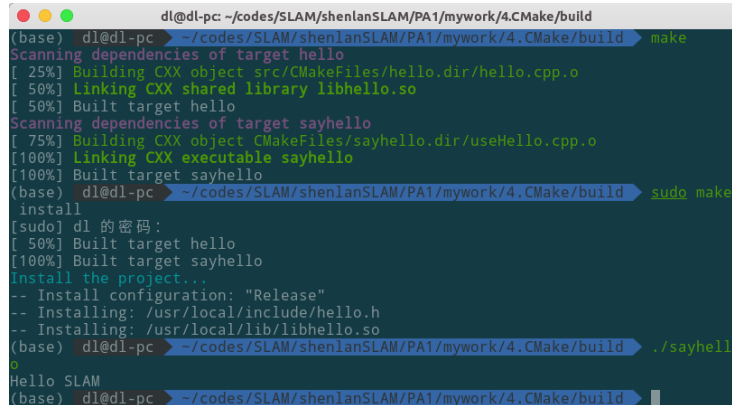
include_directories(include)

add_subdirectory(src)
# 编译 useHello.cpp 为 sayhello
add_executable(sayhello useHello.cpp)

target_link_libraries(sayhello hello)
# 将 hello.h 安装至安装路径的 include 文件夹下
```

```
install(FILES ${PROJECT_SOURCE_DIR}/include/hello.h
        DESTINATION include)
```

编译安装过程如图 3 所示，hello.h 和 libhello.so 文件被安装到了/usr/local 对应目录下，sayhello 也输出了预期结果。

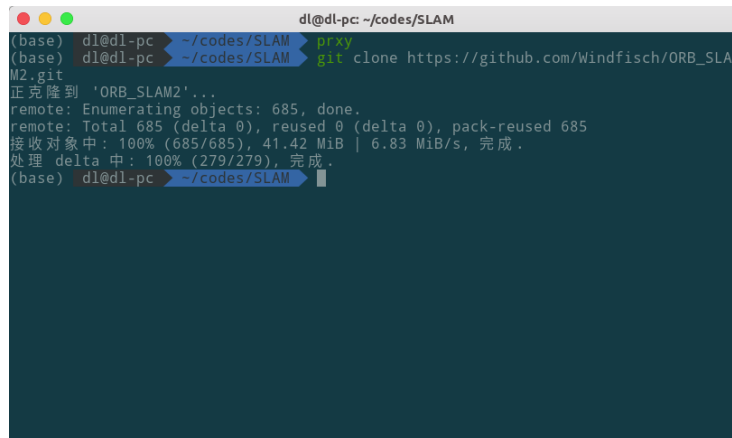


```
dl@dl-pc: ~/codes/SLAM/shenlanSLAM/PA1/mywork/4.CMake/build
(base) dl@dl-pc: ~/codes/SLAM/shenlanSLAM/PA1/mywork/4.CMake/build ➤ make
Scanning dependencies of target hello
[ 25%] Building CXX object src/CMakeFiles/hello.dir/hello.cpp.o
[ 50%] Linking CXX shared library libhello.so
[ 50%] Built target hello
Scanning dependencies of target sayhello
[ 75%] Building CXX object CMakeFiles/sayhello.dir/useHello.cpp.o
[100%] Linking CXX executable sayhello
[100%] Built target sayhello
(base) dl@dl-pc: ~/codes/SLAM/shenlanSLAM/PA1/mywork/4.CMake/build ➤ sudo make
install
[sudo] dl 的密码:
[ 50%] Built target hello
[100%] Built target sayhello
Install the project...
-- Install configuration: "Release"
-- Installing: /usr/local/include/hello.h
-- Installing: /usr/local/lib/libhello.so
(base) dl@dl-pc: ~/codes/SLAM/shenlanSLAM/PA1/mywork/4.CMake/build ➤ ./sayhell
o
Hello SLAM
(base) dl@dl-pc: ~/codes/SLAM/shenlanSLAM/PA1/mywork/4.CMake/build ➤
```

图 3: 编译安装过程图

## 4 理解 ORB-SLAM2 框架

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



```
dl@dl-pc: ~/codes/SLAM
(base) dl@dl-pc: ~/codes/SLAM ➤ proxy
(base) dl@dl-pc: ~/codes/SLAM ➤ git clone https://github.com/Windfish/ORB_SLAM2.git
正在克隆到 'ORB_SLAM2'...
remote: Enumerating objects: 685, done.
remote: Total 685 (delta 0), reused 0 (delta 0), pack-reused 685
接收对象中: 100% (685/685), 41.42 MiB | 6.83 MiB/s, 完成.
处理 delta 中: 100% (279/279), 完成.
(base) dl@dl-pc: ~/codes/SLAM ➤
```

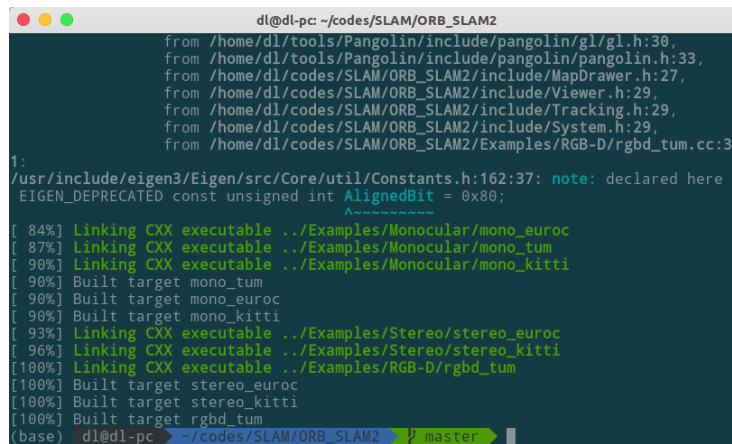
图 4: 下载 ORB\_SLAM2 截图

原版的 ORB\_SLAM2 似乎不支持 OpenCV4, 于是我使用 Windfish 修改过后的版本。

- 阅读 ORB-SLAM2 代码目录下的 CMakeLists.txt, 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?  
答: ORB\_SLAM2 将编译出一个 libORB\_SLAM2.so 共享库, 并编译出 Examples 中的 6 个可执行文件。
  - (b) ORB-SLAM2 中的 include, src, Examples 三个文件夹中都含有什么内容?  
答: src 文件夹中包含了编译 libORB\_SLAM2.so 所需的源文件, include 文件夹中包含了对应的头文件。而 Examples 文件夹中存放了使用 ORB\_SLAM2.so 共享库的示例代码, 包括单目、双目和 RGB-D 摄像机, 编译后会生成 6 个可执行文件。
  - (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?  
答: 首先会链接到 libORB\_SLAM2.so 共享库, 然后将链接到 OpenCV、Eigen3、Pangolin、DBoW2 和 g2o。

## 5 使用摄像头或视频运行 ORB-SLAM2

### 1. 编译 ORB\_SLAM2 完成截图



```

dl@dl-pc: ~/codes/SLAM/ORB_SLAM2
from /home/dl/tools/Pangolin/include/pangolin/gl/gl.h:30,
from /home/dl/tools/Pangolin/include/pangolin/pangolin.h:33,
from /home/dl/codes/SLAM/ORB_SLAM2/include/MapDrawer.h:27,
from /home/dl/codes/SLAM/ORB_SLAM2/include/Viewer.h:29,
from /home/dl/codes/SLAM/ORB_SLAM2/include/Tracking.h:29,
from /home/dl/codes/SLAM/ORB_SLAM2/include/System.h:29,
from /home/dl/codes/SLAM/ORB_SLAM2/Examples/RGB-D/rgb_d_tum.cc:3
1:
/usr/include/eigen3/Eigen/src/Core/util/Constants.h:162:37: note: declared here
EIGEN_DEPRECATED const unsigned int AlignedBit = 0x80;
~~~~~
[ 84%] Linking CXX executable ../Examples/Monocular/mono_euroc
[ 87%] Linking CXX executable ../Examples/Monocular/mono_tum
[ 90%] Linking CXX executable ../Examples/Monocular/mono_kitti
[ 90%] Built target mono_tum
[ 90%] Built target mono_euroc
[ 90%] Built target mono_kitti
[ 93%] Linking CXX executable ../Examples/Stereo/stereo_euroc
[ 96%] Linking CXX executable ../Examples/Stereo/stereo_kitti
[100%] Linking CXX executable ../Examples/RGB-D/rgb_d_tum
[100%] Built target stereo_euroc
[100%] Built target stereo_kitti
[100%] Built target rgb_d_tum
(base) dl@dl-pc: ~/codes/SLAM/ORB_SLAM2
  
```

图 5: ORB\_SLAM2 编译完成截图

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。  
答: 在 Examples 文件夹下新建 myvideo 文件夹放置 myvideo.cpp、myvideo.mp4 和 myvideo.yaml。在 CMakeLists.txt 尾部加上下列代码

```
# video
set(CMAKE_RUNTIME_OUTPUT_DIRECTORY
↪  ${PROJECT_SOURCE_DIR}/Examples/myvideo)

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

3. 请给出运行截图, 并谈谈你在运行过程中的体会。

答: 我之前安装过 OpenCV、Eigen3 等库, 所以 ORB\_SLAM2 的安装过程也比较顺利。唯一遇到的一个问题是原版的 ORB\_SLAM2 似乎没有提供 OpenCV4 的支持, 仅仅在 CMakeLists.txt 里加入 find\_package (OpenCV 4 REQUIRED) 并不能解决问题, 因此我使用了 Windfish 修改过后的 ORB\_SLAM2。在修改 CMakeLists.txt 加入 myvideo 后, 如图 6 所示成功编译运行。但是多次重复运行中, 偶尔会发生视频部分卡住不动的情况, 并没有找出解决办法。

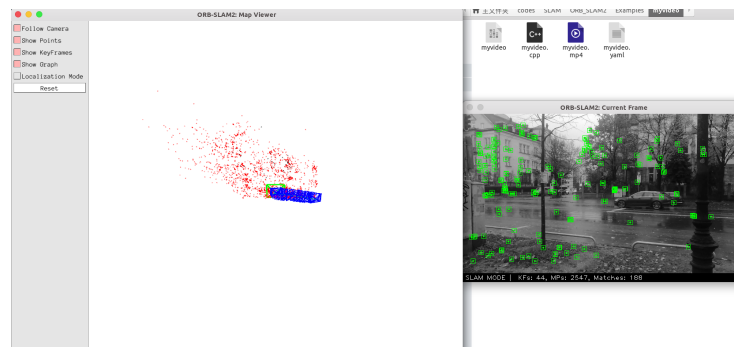


图 6: myvideo 运行截图