

第一节课习题

崔华坤

二 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：

- `sudo apt-get install SoftwareName`，如 `sudo apt-get install kdevelop`

- 软件通常安装位置：

可执行文件：/usr/bin

库文件：/usr/lib

配置文件：/etc

文档：/usr/share

2. linux 的环境变量是什么？我如何定义新的环境变量？

答：

- 环境变量是指操作系统和软件运行时，当需要依赖第三方库，或者需要查找某个软件时，会通过环境变量来寻找。Linux的环境变量有多个文件可以设置，如：

`etc/profile`：用户登录时，执行一次，用于设置所有用户使用的全局变量

`etc/bashrc`：每次新打开终端shell时执行一次；

`~/.profile`：Ubuntu当用户登录时，执行一次，只对单个用户生效；

`~/.bashrc`：当登录以及每次打开新的shell时，执行一次，只对单个用户生效；

`etc/environment`：设置整个系统的环境

- 定义新的环境变量方式：

打开上述相应的环境配置文件，添加：`PATH=$PATH:/home/stevencui/`

或者命令行输入：`export PATH=$PATH:/home/stevencui/`

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：

```
stevencui@ubuntu:/$ ll
total 136
drwxr-xr-x 24 root root 4096 Jun 16 2017 ./
drwxr-xr-x 24 root root 4096 Jun 16 2017 ../
drwxr-xr-x 2 root root 4096 Jun 16 2017 bin/
drwxr-xr-x 3 root root 4096 Jun 16 2017 boot/
drwxrwxr-x 2 root root 4096 Jun 16 2017 cdrom/
drwxr-xr-x 17 root root 4260 Feb 26 08:58 dev/
drwxr-xr-x 134 root root 12288 Feb 26 18:35 etc/
drwxr-xr-x 3 root root 4096 Jun 16 2017 home/
lrwxrwxrwx 1 root root 32 Jun 16 2017 initrd.img -> boot/initrd.img-4.4.0-31-generic
drwxr-xr-x 23 root root 4096 Jun 16 2017 lib/
drwxr-xr-x 2 root root 4096 Jun 16 2017 lib32/
drwxr-xr-x 2 root root 4096 Jun 16 2017 lib64/
drwx----- 2 root root 16384 Jun 16 2017 lost+found/
drwxr-xr-x 4 root root 4096 Jun 28 2017 media/
drwxr-xr-x 2 root root 4096 Apr 11 2014 mnt/
drwxr-xr-x 3 root root 4096 Jan 3 14:41 opt/
dr-xr-xr-x 312 root root 0 Feb 26 08:58 proc/
drwx----- 2 root root 4096 Aug 3 2016 root/
drwxr-xr-x 23 root root 780 Feb 26 18:35 run/
drwxr-xr-x 2 root root 12288 Jun 16 2017/sbin/
drwxr-xr-x 2 root root 4096 Aug 3 2016 srv/
dr-xr-xr-x 13 root root 0 Feb 26 08:58 sys/
drwxrwxrwt 4 root root 28672 Feb 27 08:17 tmp/
drwxr-xr-x 11 root root 4096 Jun 16 2017 usr/
drwxr-xr-x 13 root root 4096 Aug 3 2016 var/
lrwxrwxrwx 1 root root 29 Jun 16 2017 vmlinuz -> boot/vmlinuz-4.4.0-31-generic
```

/bin: 二进制可执行命令

/dev: 设备特殊文件

/etc: 系统管理和配置文件

/home: 用户主目录

/lib: 动态链接共享库

/sbin: 系统管理员使用的管理命令

/tmp: 公用临时文件存储点

/root: 系统管理员的主目录

/mnt: 让用户临时挂载其他的文件系统

/lost+found: 系统非正常关机留下的文件

/proc: 系统内存的映射，可获取系统信息

/var: 大文件的溢出区

/usr: 几乎所有的应用程序和文件目录，包括：

/usr/bin: 应用程序

/usr/doc: linux文档

/usr/include: 开发和编译应用程序所需要的头文件

/usr/lib: 动态链接库和软件包的配置文件

/usr/man: 帮助文档

/usr/src: 源代码

/usr/local/bin: 本地增加的命令

/usr/local/lib: 本地增加的库根文件

4. 假设我要给a.sh 加上可执行权限，该输入什么命令？

答：\$ chmod +x 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 useradd -d /home/xiang -m xiang

\$ sudo chown xiang:xiang a.sh

三、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：AR、移动机器人、无人机、自动驾驶。

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：定位和建图是相互关联的，因为要定位出在陌生环境中的位置，首先需要已知周围环境的准确地图，而构建陌生环境的地图，则需要已知当前时刻的位姿。

以单目vSLAM为例，首先通过初始化构建初始地图，用初始地图进行相机定位，当相机定位完成后，可以将场景中的其他Landmark加入到地图中，对地图进行更新，进而当相机继续移动时，可根据更新后的地图进行定位。

3. SLAM 发展历史如何？我们可以将它划分成哪几个阶段？

答：

- 史前：~1990，未形成SLAM概念，仅在地图已知的情况下进行定位
- EKF-SLAM：1990~2005，基于滤波（KF、EKF、Particle）求解SLAM
- BA-SLAM：2006~2010，基于优化方法，对定位和建图中的误差进行建模，同时考虑两个问题。
- 前沿：2010~，出现日益成熟的系统和软件，如LSD-SLAM、ORB-SLAM、SVO、DTAM、G2O，出现丰富的传感器，如双目/多目、RGBD、全景，出现其他方向，如语义地图、基于深度学习进行闭环检测。

4. 列举三篇在 SLAM 领域的经典文献。

答：

- R. Mur-Artal, J. M. M. Montiel, and J. D. Tard'os, ORB-SLAM: a versatile and

accurate monocular SLAM system, IEEE Trans. Robot., vol. 31, no. 5, pp. 1147–1163, 2015.

- Davison, A., Reid, I., Molton, N.D., Stasse, O.: MonoSLAM: Real-time single camera SLAM. IEEE Trans. Pattern Analysis and Machine Intelligence 29 (June 2007) 1052–1067.
- P. Li, T. Qin, etc. Monocular Visual-Inertial State Estimation for Mobile Augmented Reality. 2017, ISMAR.

四、CMake 练习

- CMakeLists.txt:

```
CMakeLists.txt x
CMAKE_MINIMUM_REQUIRED(VERSION 2.8)
set(CMAKE_BUILD_TYPE Release)
PROJECT(sayhello)

INCLUDE_DIRECTORIES(${PROJECT_SOURCE_DIR}/include)
ADD_LIBRARY(hello SHARED ${PROJECT_SOURCE_DIR}/src/hello.c)

ADD_EXECUTABLE(sayhello useHello.c)
TARGET_LINK_LIBRARIES(sayhello hello)

INSTALL(TARGETS hello
        LIBRARY DESTINATION /usr/local/lib)

INSTALL(FILES ${PROJECT_SOURCE_DIR}/include/hello.h
        DESTINATION /usr/local/include)
```

- 运行结果:

```

stevencui@ubuntu:~/Project/SLAMCourse/l1/build$ cmake ..
-- Configuring done
-- Generating done
-- Build files have been written to: /home/stevencui/Project/SLAMCourse/l1/build
stevencui@ubuntu:~/Project/SLAMCourse/l1/build$ make
[ 25%] Building C object CMakeFiles/hello.dir/src/hello.c.o
[ 50%] Linking C shared library libhello.so
[ 50%] Built target hello
[ 75%] Building C object CMakeFiles/sayhello.dir/useHello.c.o
[100%] Linking C executable sayhello
[100%] Built target sayhello
stevencui@ubuntu:~/Project/SLAMCourse/l1/build$ ll
total 64
drwxrwxr-x 3 stevencui stevencui 4096 Feb 28 08:29 ./
drwxrwxr-x 5 stevencui stevencui 4096 Feb 28 08:27 ../
-rw-rw-r-- 1 stevencui stevencui 12739 Feb 28 08:27 CMakeCache.txt
drwxrwxr-x 6 stevencui stevencui 4096 Feb 28 08:29 CMakeFiles/
-rw-r--r-- 1 root root 3298 Feb 28 08:28 cmake_install.cmake
-rw-r--r-- 1 root root 53 Feb 28 08:28 install_manifest.txt
-rwxrwxr-x 1 stevencui stevencui 8006 Feb 28 08:29 libhello.so*
-rw-rw-r-- 1 stevencui stevencui 8019 Feb 28 08:29 Makefile
-rwxrwxr-x 1 stevencui stevencui 8563 Feb 28 08:29 sayhello*
stevencui@ubuntu:~/Project/SLAMCourse/l1/build$ ./sayhello
Hello SLAM from sayHello()
Hello world from main()
stevencui@ubuntu:~/Project/SLAMCourse/l1/build$ sudo make install
[ 50%] Built target hello
[100%] Built target sayhello
Install the project...
-- Install configuration: "Release"
-- Installing: /usr/local/lib/libhello.so
-- Up-to-date: /usr/local/include/hello.h

```

组织源代码文件，并书写 CMakeLists.txt

五、理解 ORB-SLAM2 框架

1. 代码下载:

```

stevencui@ubuntu:~/Project$ git clone https://github.com/raulmur/ORB_SLAM2.git ORB_SLAM2
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 | 10.00 KiB/s, done.
Resolving deltas: 100% (178/178), 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？

答：编译结果：

1个库文件：libORB_SLAM2.so

6个可执行文件：rgbd_tum、stereo_kitti、stereo_euroc、mono_tum、mono_kitti、mono_euroc

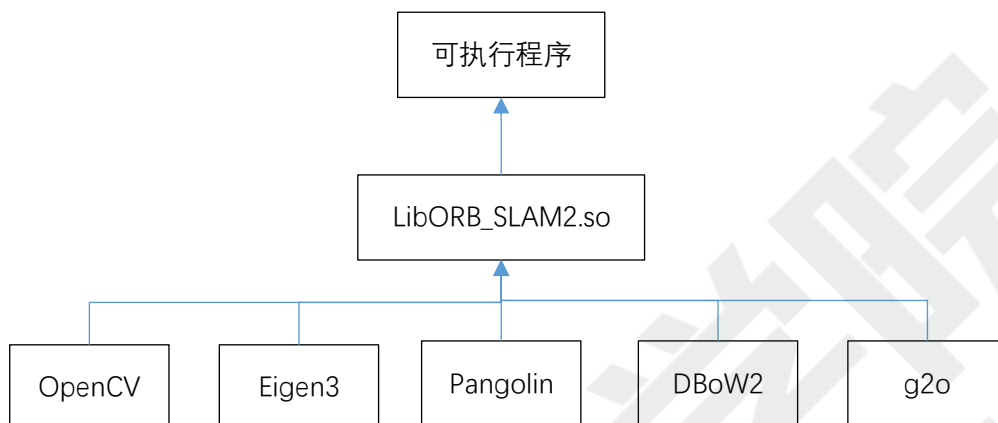
(b) ORB-SLAM2 中的include, src, Examples 三个文件夹中都含有什么内容？

答：include：生成库的头文件，src：生成库的源代码，Examples：不同相机的

采集程序及main函数

(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：



6 * 使用摄像头或视频运行 ORB-SLAM2 (3 分, 约 1 小时)

1. 编译完成的截图

```
[ 56%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/Initializer.cc.o
[ 59%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/Sim3Solver.cc.o
[ 62%] Linking CXX shared library ../lib/libORB_SLAM2.so
[ 62%] Built target ORB_SLAM2
Scanning dependencies of target mono_tum
Scanning dependencies of target mono_kitti
Scanning dependencies of target mono_euroc
Scanning dependencies of target stereo_kitti
Scanning dependencies of target rgbd_tum
Scanning dependencies of target stereo_euroc
[ 65%] Building CXX object CMakeFiles/mono_tum.dir/Examples/Monocular/mono_tum.cc.o
[ 68%] Building CXX object CMakeFiles/stereo_kitti.dir/Examples/Stereo/stereo_kitti.cc.o
[ 71%] Building CXX object CMakeFiles/rgbd_tum.dir/Examples/RGB-D/rgbd_tum.cc.o
[ 75%] Building CXX object CMakeFiles/mono_euroc.dir/Examples/Monocular/mono_euroc.cc.o
[ 78%] Building CXX object CMakeFiles/stereo_euroc.dir/Examples/Stereo/stereo_euroc.cc.o
[ 81%] Building CXX object CMakeFiles/mono_kitti.dir/Examples/Monocular/mono_kitti.cc.o
[ 84%] Linking CXX executable ../Examples/Monocular/mono_tum
[ 87%] Linking CXX executable ../Examples/RGB-D/rgbd_tum
[ 90%] Linking CXX executable ../Examples/Monocular/mono_euroc
[ 93%] Linking CXX executable ../Examples/Stereo/stereo_kitti
[ 96%] Linking CXX executable ../Examples/Monocular/mono_kitti
[100%] Linking CXX executable ../Examples/Stereo/stereo_euroc
[100%] Built target mono_euroc
[100%] Built target mono_tum
[100%] Built target mono_kitti
[100%] Built target stereo_kitti
[100%] Built target rgbd_tum
[100%] Built target stereo_euroc
stevencui@ubuntu:~/Project/ORB_SLAM2$
```

2. CMakeLists.txt 修改方案

答：添加：

```
add_executable(mono_c920 Examples/Monocular/myslam.cpp)
target_link_libraries(mono_c920 ${PROJECT_NAME})
```

3. 运行截图，并谈谈你在运行过程中的体会

答：可能是虚拟机的缘故，图像有撕裂且运行较卡顿，但是缓慢移动时相机姿态能跟踪到，容易丢失，但是回到去过的地方可重定位回来。整体相机轨迹正确。

