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？

1) 使用命令行工具 apt 安装。

例如安装 vim 编辑器：sudo apt install vim

2) 使用 Deb 包安装

例如：sudo dpkg -i <package.deb>

通常安装在/usr 目录下，但也有部分文件在/var、/lib 目录下。

如 vim(部分)：

```
jc@jc-mint:~$ locate vim
/etc/vim
/etc/vim/vimrc
/etc/vim/vimrc.tiny
/lib/modules/4.15.0-20-generic/kernel/drivers/media/platform/vim2m.ko
/usr/bin/vim.tiny
/usr/lib/debug/usr/lib/xplayer/plugins/vimeo
/usr/lib/debug/usr/lib/xplayer/plugins/vimeo/libvimeo.so
/usr/lib/libreoffice/share/config/soffice.cfg/svx/ui/formnavimenu.ui
/usr/lib/mime/packages/vim-common
/usr/lib/x86_64-linux-gnu/gstreamer-1.0/libgstxvimagesink.so
/usr/lib/xplayer/plugins/vimeo
/usr/lib/xplayer/plugins/vimeo/libvimeo.so
/usr/lib/xplayer/plugins/vimeo/vimeo.plugin
/usr/share/vim
/usr/share/app-install/desktop/vim-common:vim.desktop
/usr/share/app-install/desktop/vim-gui-common:gvim.desktop
/usr/share/app-install/desktop/yi:yi-vim.desktop
```

Cmake(部分)：

```
jc@jc-mint:~$ locate cmake
/lib/firmware/carl9170fw/extra/FindGPERF.cmake
/lib/firmware/carl9170fw/extra/FindPackageHandleStandardArgs.cmake
/lib/firmware/carl9170fw/extra/FindUSB-1.0.cmake
/lib/firmware/carl9170fw/extra/GCCVersion.cmake
/lib/firmware/carl9170fw/extra/sh-elf-linux.cmake
/usr/lib/ruby/2.5.0/rubygems/ext/cmake_builder.rb
/usr/share/cmake
/usr/share/cmake/bash-completion
/usr/share/cmake/bash-completion/bash-completion-config-version.cmake
/usr/share/cmake/bash-completion/bash-completion-config.cmake
/usr/share/gtksourceview-3.0/language-specs/cmake.lang
/usr/share/icons/Mint-Y/apps/16/cmake.png
/usr/share/icons/Mint-Y/apps/16@2x/cmake.png
/usr/share/icons/Mint-Y/apps/22/cmake.png
/usr/share/icons/Mint-Y/apps/22@2x/cmake.png
/usr/share/icons/Mint-Y/apps/24/cmake.png
```

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$PATH 变量的值（一系列目录），决定了 shell 到哪个目录中寻找命令或者程序。

1) 查看环境变量值：echo $PATH

```
jc@jc-mint:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games
```

2) 添加 PATH 环境变量(临时)，可用：
export PATH=/xxx/xxx/xxx:$PATH **但是这种方法在关闭终端后失效。**
3) 永久添加环境变量(影响当前用户):
编辑 ~/.bashrc 文件，gedit ~/.bashrc
export PATH="/xxx/xxx/xxx:$PATH"
4) 永久添加环境变量(影响所有用户):
编辑 /etc/profile 文件，gedit /etc/profile
export PATH="/xxx/xxx/xxx:$PATH"

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。

目录结构：

部分目录作用：

目录	说明	备注
bin	存放普通用户可执行的指令	即使在单用户模式下也能够执行处理
boot	开机引导目录	包括 Linux 内核文件与开机所需要的文件
dev	设备目录	所有的硬件设备及周边均放置在这个设备目录中
etc	各种配置文件目录	大部分配置属性均存放在这里
opt	第三方软件安装目录	现在习惯性的放置在/usr/local 中
root	系统管理员主目录	除 root 之外,其他用户均放置在/home 目录下
tmp	存放临时文件目录	所有用户对该目录均可读写
usr	应用程序放置目录	
var	存放系统执行过程经常改变的文件	

4. 假设我要给 a.sh 加上可执行权限，该输入什么命令？

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？

xang:xiang 的意思是 xiang 组下 xiang 用户吗？如果是，那么：
改变所属组群：sudo chgrp xiang a.sh
改变拥有者：sudo chown xiang a.sh

三、SLAM 综述文献阅

1. SLAM 会在哪些场合中用到？至少列举三个方向。

1) 增强现实
2) 自主机器人
3) 计算机视觉与信号处理
4) 无人驾驶

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？

定位与建图的关系是相互关联相互制约的，准确的定位需要精确的地图，精确的地图来自准确的定位。定位的同时需要建图，因为如果没有地图，主体的定位会很快发生漂移（draft），从而产生很大的累计误差。如果存在地图，可以通过回环检测及时纠正（reset）定位误差，很大程度上限制（limit）了估计错误。而地图构建的信息来自每一次定位，所以在定位的同时需要建图。

3. SLAM 发展历史如何？我们可以将它划分成哪几个阶段？

SLAM 从出现至今有 30 多年的发展历史，可以划分为三个阶段：

最初的 20 年称之为经典阶段（classical age，1986-2004），这个阶段的主要思想是基于概率学的方法（probabilistic formulations），出现了扩展卡尔曼滤波（EKF）、粒子滤波（RBPF）、最大似然估计等方法。

接下来的 10 年称之为算法分析阶段（algorithmic-analysis age，2004-2015），这个阶段重点研究了 SLAM 的基础属性，包括可观测性，收敛性和一致性。同时，对于高效 SLAM 求解很关键的的稀疏性理论也被理解，各种主要的开源 SLAM 库得以发展。

2015 年至今，我们正在进入第三个阶段，称之为鲁棒性理解阶段（robust-perception age）。在这个阶段，SLAM 研究的重点在于 4 个方面：鲁棒性，高级语义理解，资源限制下的调整，具体任务驱动下的 SLAM。

4. 列举三篇在 SLAM 领域的经典文献。

2006 《Probabilistic approaches and data association》 Durrant-Whyte and Bailey
2008 《Visual SLAM》 Neira et al. (special issue)
2016 《Visual place recognition》 Lowry et al.

四、CMake 练习

三个 CMakeLists 文件：

CMakeLists:
```
PROJECT(USECMAKE)
ADD_SUBDIRECTORY(include)
ADD_SUBDIRECTORY(src)
```

include/CMakeLists:
```
INSTALL(FILES hello.h DESTINATION include)
```

src/CMakeLists:

```
ADD_LIBRARY(hello SHARED hello.c)
INSTALL(TARGETS hello
LIBRARY DESTINATION lib)
INCLUDE_DIRECTORIES(/home/jc/SLAM/ch1/cmake/include)
ADD_EXECUTABLE(sayhello useHello.c)
TARGET_LINK_LIBRARIES(sayhello hello
```

生成库文件：

```
jc@jc-mint:~/SLAM/ch1/cmake/build/src$ ls
CMakeFiles  cmake_install.cmake  libhello.so  Makefile  sayhello
```

可执行文件：

```
jc@jc-mint:~/SLAM/ch1/cmake/build/src$ ./sayhello
Hello SLAM
jc@jc-mint:~/SLAM/ch1/cmake/build/src$
```

安装至相应位置：

```
jc@jc-mint:~/SLAM/ch1/cmake/build$ sudo make install
[sudo] password for jc:
[ 50%] Built target hello
[100%] Built target sayhello
Install the project...
-- Install configuration: ""
-- Installing: /usr/local/include/hello.h
-- Installing: /usr/local/lib/libhello.so
jc@jc-mint:~/SLAM/ch1/cmake/build$
```

五、理解 ORB-SLAM2 框架

1. 下载截图

```
jc@jc-mint:~$ git clone https://github.com/raulmur/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), 41.33 MiB | 2.63 MiB/s, done.
Resolving deltas: 100% (198/198), done.
jc@jc-mint:~$ ls
clion-2018.2.3  Music      SLAM        VMwareTools-10.2.0-7259539.tar.gz
Desktop         ORB_SLAM2  temp        vmware-tools-distrib
Documents       Pictures   Templates
Downloads       Public     Videos
jc@jc-mint:~$ ls ORB_SLAM2/
build_ros.sh    cmake_modules    include          README.md   Vocabulary
build.sh        Dependencies.md  License-gpl.txt  src
CMakeLists.txt  Examples         LICENSE.txt      Thirdparty
```

2. 代码结构

(a) 将编译出一个名为 ORB_SLAM2 的工程，包括 1 个库文件和 6 个可执行文件

(b) include 含有用来编译库的头文件，src 含有用来编译库的源文件，example 含有用来编译可执行文件的源文件

(c) 链接到的库有：

${OpenCV_LIBS}
${EIGEN3_LIBS}
${Pangolin_LIBRARIES}
${PROJECT_SOURCE_DIR}/Thirdparty/DBoW2/lib/libDBoW2.so
${PROJECT_SOURCE_DIR}/Thirdparty/g2o/lib/libg2o.so

它们的名字分别是 OpenCV、Eigen3、Pangolin、DBoW2、g2o

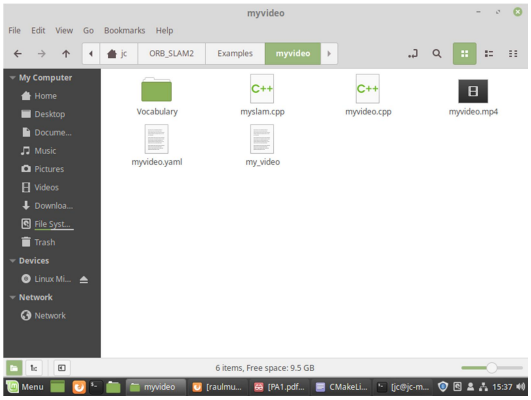
六、使用摄像头或视频运行 ORB-SLAM2

1.编译完成

```
[ 84%] Linking CXX executable ../Examples/RGB-D/rgbd_tum
[ 87%] Linking CXX executable ../Examples/Monocular/mono_tum
[ 90%] Linking CXX executable ../Examples/Monocular/mono_kitti
[ 93%] Linking CXX executable ../Examples/Monocular/mono_euroc
[ 96%] Linking CXX executable ../Examples/Stereo/stereo_kitti
[100%] Linking CXX executable ../Examples/Stereo/stereo_euroc
[100%] Built target rgbd_tum
[100%] Built target mono_tum
[100%] Built target mono_kitti
[100%] Built target mono_euroc
[100%] Built target stereo_kitti
[100%] Built target stereo_euroc
jc@jc-mint:~/ORB_SLAM2$
```

2.使用 myvideo.cpp

1) 在 Example 目录下新建 myvideo 文件夹，放入相关文件如图：

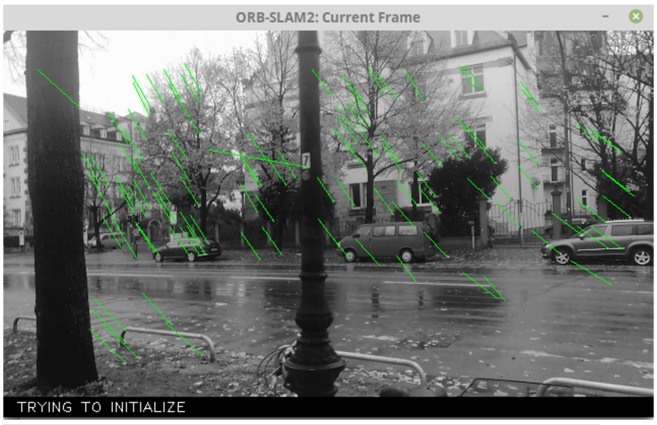


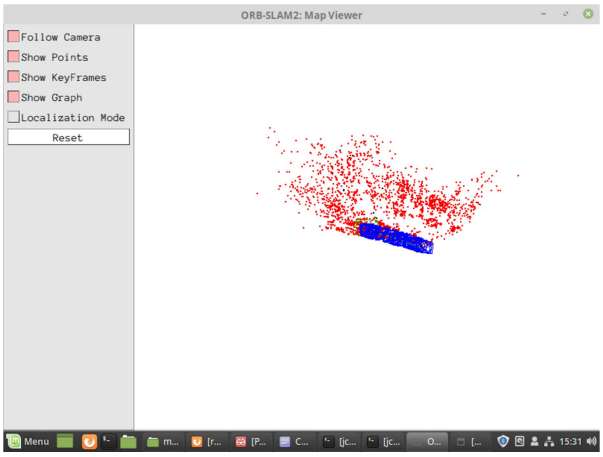
2)在 CMakeLists.txt 末尾添加如下代码：

```
set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${PROJECT_SOURCE_DIR}/Examples/myvideo)
add_executable(my_video
Examples/myvideo/myvideo.cpp)
target_link_libraries(my_video ${PROJECT_NAME})
```

编译完成后在 video 目录下生成 my_video 目标文件

3.运行截图






```
jc@jc-mint:~/ORB_SLAM2/Examples/myvideo$ ./my_video

ORB-SLAM2 Copyright (C) 2014-2016 Raul Mur-Artal, University of Zaragoza.
This program comes with ABSOLUTELY NO WARRANTY;
This is free software, and you are welcome to redistribute it
under certain conditions. See LICENSE.txt.

Input sensor was set to: Monocular

Loading ORB Vocabulary. This could take a while...
Vocabulary loaded!


Camera Parameters:
- fx: 500
- fy: 500
- cx: 320
- cy: 180
- k1: 0
- k2: 0
- p1: 0
- p2: 0
- fps: 30
- color order: BGR (ignored if grayscale)

ORB Extractor Parameters:
- Number of Features: 2000
- Scale Levels: 8
- Scale Factor: 1.2
- Initial Fast Threshold: 20
- Minimum Fast Threshold: 7
Framebuffer with requested attributes not available. Using available framebuffer. You may see visual artifacts.New Map created with 107 points
```

由于环境为虚拟机，所以使用了 myvideo.mp4 来运行 ORB-SLAM2。首先加载 ORB Vocabulary 需要等待几秒钟，这个字典文件是什么还不清楚。运行过程中，帧窗口总是运行几秒后就停止(或者是卡住？尝试了很多次都是在一个位置停止)，停止的同时终端输出最后一行信息，给出了新地图中关键点的数量。但是 Map viewer 窗口仍然在运行，在不断推进建图过程，很奇怪是否视频流也应该继续运行？大概一分钟后停止，所有窗口关闭。一旦开始运行，机器运转的声音就会变大，说明计算资源较多。