Solution of 第一节课习题 (macOS 平台)

张吉祥

2018 年 3 月 3 日

1 习题说明

2 熟悉 Linux

1. Ubuntu 中常用 apt-get 来安装软件，以安装 build-essential 为例：

```
$ sudo apt-get install build-essential
```

   它们通常被安装在/usr/local 或/usr/bin 或/usr/local/bin 等,具体安装路径可以用 **$ locate xxx** 命令查询。

2. Linux 的环境变量是操作系统中具有特定名字的对象，它包含了应用程序将使用到的信息。常用的环境变量包括 PATH, HOME 等。我们可以通过以下三种方式定义新的环境变量：

   - 在/etc/profile 文件中添加变量
   - 在用户目录下的.bash_profile 文件中增加变量
   - 直接运行 export 命令定义变量

3. Linux 根目录下面的目录结构如图 1 所示。其中/dev 为设备目录，/etc 为配置文件目录，/home 为用户目录，/usr/bin 为绝大部分的用户可使用指令所在目录，/usr/include 为头文件目录，/usr/lib 为库文件目录。

4. 给 a.sh 加上可执行权限的命令：

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

5. 修改 a.sh 文件所有者

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

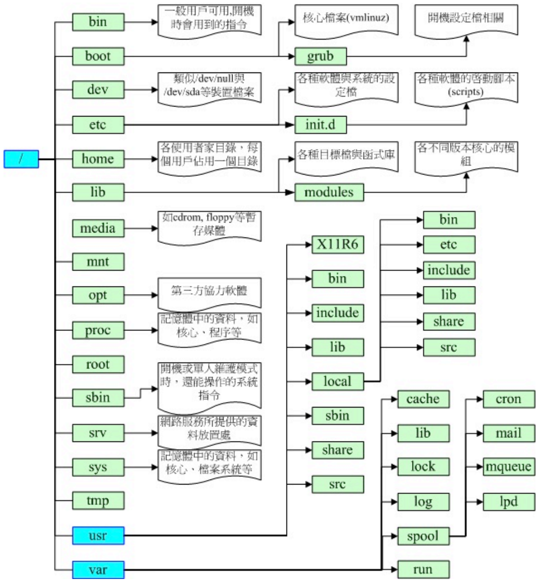


图 1: 目录结构

3 SLAM 综述文献阅读

1. SLAM 应用场合：增强现实 (AR)；移动机器人；自动驾驶定位；手持设备定位。

2. SLAM 中定位与建图的关系：定位侧重对自身的了解，建图侧重对外在的了解。因为定位与建图相互关联，准确的定位需要精确的地图，精确的地图来自准确的定位。

3. SLAM 发展历史分为三个阶段：提出问题 (1986 - 2004)、寻找算法 (2004 - 2015)、完善算法 (2015 -)。

4. SLAM 领域三篇经典文献：

   - Davison A J, Reid I D, Molton N D, et al. **MonoSLAM: Real-time single camera SLAM**[J]. IEEE transactions on pattern analysis and machine intelligence, 2007, 29(6): 1052-1067.

   - Klein G, Murray D. **Parallel tracking and mapping for small AR workspaces**[C]//Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on. IEEE, 2007: 225-234.

   - Mur-Artal R, Montiel J M M, Tardos J D. **ORB-SLAM: a versatile and accurate monocular SLAM system**[J]. IEEE Transactions on Robotics, 2015, 31(5): 1147-1163.

4 CMake 练习

工程文件目录结构:

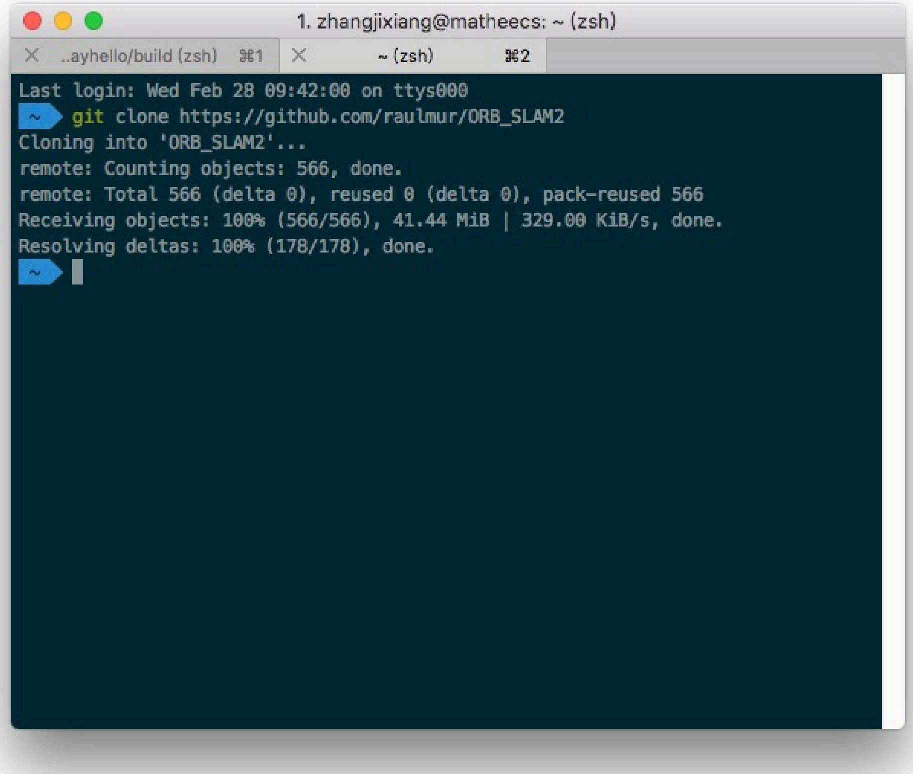
- **sayhello/**
  - CMakeLists.txt
  - include/
    - * hello.h
  - src/
    - * hello.cpp
  - test/
    - * useHello.cpp

sayhello/CMakeLists.txt 源码:

```
cmake_minimum_required(VERSION 2.8)
project(sayhello)

set(CMAKE_BUILD_TYPE "Release")

include_directories(${PROJECT_SOURCE_DIR}/include)
install(FILES ${PROJECT_SOURCE_DIR}/include/hello.h DESTINATION /usr/local/include)

add_library(hello SHARED ${PROJECT_SOURCE_DIR}/src/hello.cpp)
install(TARGETS hello LIBRARY DESTINATION /usr/local/lib)

set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${PROJECT_SOURCE_DIR}/test)
add_executable(sayhello ${PROJECT_SOURCE_DIR}/test/useHello.cpp)
target_link_libraries(sayhello hello)
```

5 理解 ORB-SLAM2 框架

1. 下载 ORB-SLAM2 的代码，见图 2 。



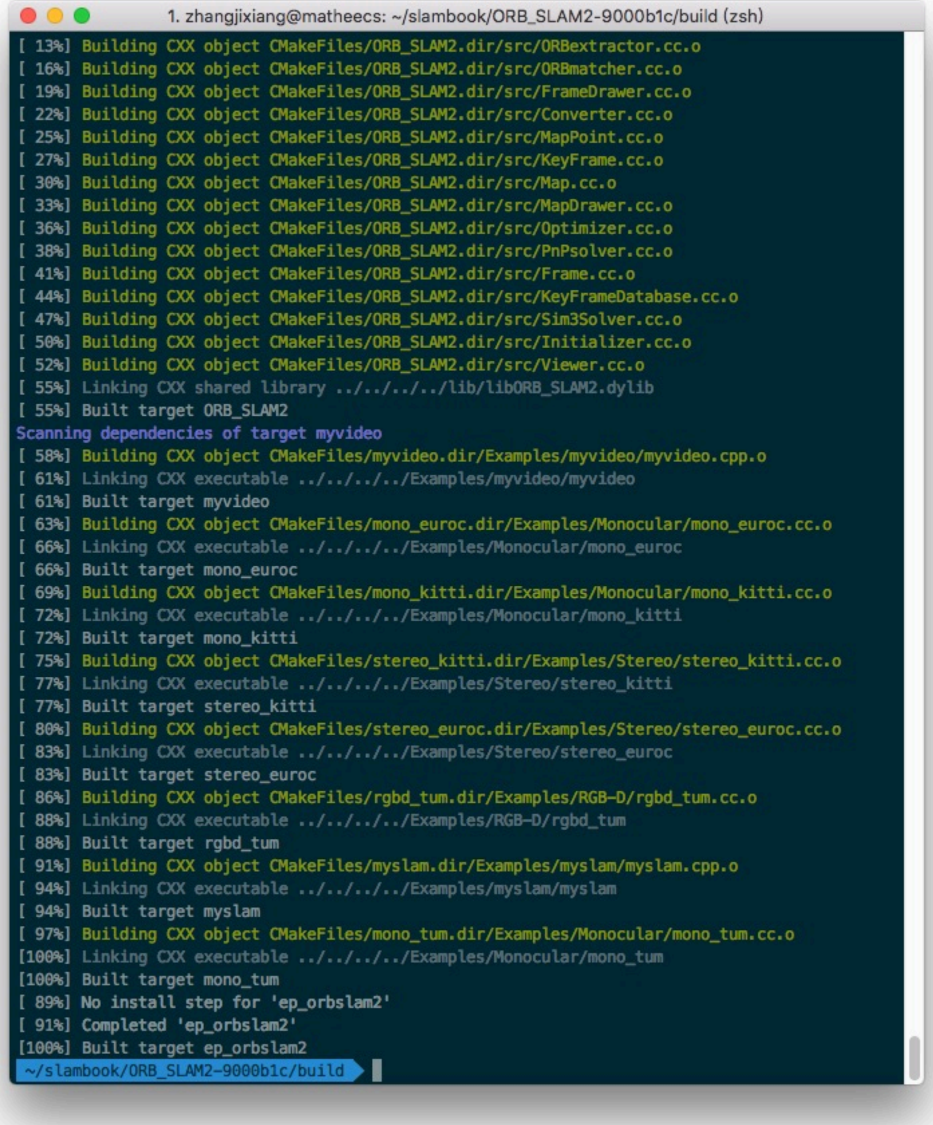
```
Last login: Wed Feb 28 09:42:00 on ttys000
 ~  git clone https://github.com/raulmur/ORB_SLAM2
Cloning into 'ORB_SLAM2'...
remote: Counting objects: 566, done.
remote: Total 566 (delta 0), reused 0 (delta 0), pack-reused 566
Receiving objects: 100% (566/566), 41.44 MiB | 329.00 KiB/s, done.
Resolving deltas: 100% (178/178), done.
 ~ 
```

图 2: 下载 ORB-SLAM2

2. (a) 编译得到**可执行文件**： rgbd_tum, stereo_kitti, stereo_euroc, mono_tum, mono_kitti, mono_euroc, **库文件**:libORB_SLAM2.so(注:macOS 上库文件名变为 libORB_SLAM2.dylib)。一共有 6 个可执行文件和 1 个库文件。
(b)include: 存放工程的头文件
src: 存放工程的源代码文件
Examples: 存放例程测试文件
(c) 可执行文件链接到了 libORB_SLAM2.so(注:macOS 上库文件名变为 libORB_SLAM2.dylib)，而 libORB_SLAM2.so 链接到了 **OpenCV**，**EIGEN3**，**Pangolin**，**DBoW2**，**g2o**。

6 使用摄像头或视频运行 ORB-SLAM2

1. 编译完成如图 3。


1. zhangjixiang@matheecs: ~/slambook/ORB_SLAM2-9000b1c/build (zsh)

```
[ 13%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/ORBextractor.cc.o
[ 16%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/ORBmatcher.cc.o
[ 19%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/FrameDrawer.cc.o
[ 22%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/Converter.cc.o
[ 25%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/MapPoint.cc.o
[ 27%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/KeyFrame.cc.o
[ 30%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/Map.cc.o
[ 33%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/MapDrawer.cc.o
[ 36%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/Optimizer.cc.o
[ 38%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/PnPsolver.cc.o
[ 41%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/Frame.cc.o
[ 44%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/KeyFrameDatabase.cc.o
[ 47%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/Sim3Solver.cc.o
[ 50%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/Initializer.cc.o
[ 52%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/Viewer.cc.o
[ 55%] Linking CXX shared library ../../../../lib/libORB_SLAM2.dylib
[ 55%] Built target ORB_SLAM2
Scanning dependencies of target myvideo
[ 58%] Building CXX object CMakeFiles/myvideo.dir/Examples/myvideo/myvideo.cpp.o
[ 61%] Linking CXX executable ../../../../Examples/myvideo/myvideo
[ 61%] Built target myvideo
[ 63%] Building CXX object CMakeFiles/mono_euroc.dir/Examples/Monocular/mono_euroc.cc.o
[ 66%] Linking CXX executable ../../../../Examples/Monocular/mono_euroc
[ 66%] Built target mono_euroc
[ 69%] Building CXX object CMakeFiles/mono_kitti.dir/Examples/Monocular/mono_kitti.cc.o
[ 72%] Linking CXX executable ../../../../Examples/Monocular/mono_kitti
[ 72%] Built target mono_kitti
[ 75%] Building CXX object CMakeFiles/stereo_kitti.dir/Examples/Stereo/stereo_kitti.cc.o
[ 77%] Linking CXX executable ../../../../Examples/Stereo/stereo_kitti
[ 77%] Built target stereo_kitti
[ 80%] Building CXX object CMakeFiles/stereo_euroc.dir/Examples/Stereo/stereo_euroc.cc.o
[ 83%] Linking CXX executable ../../../../Examples/Stereo/stereo_euroc
[ 83%] Built target stereo_euroc
[ 86%] Building CXX object CMakeFiles/rgbd_tum.dir/Examples/RGB-D/rgbd_tum.cc.o
[ 88%] Linking CXX executable ../../../../Examples/RGB-D/rgbd_tum
[ 88%] Built target rgbd_tum
[ 91%] Building CXX object CMakeFiles/myslam.dir/Examples/myslam/myslam.cpp.o
[ 94%] Linking CXX executable ../../../../Examples/myslam/myslam
[ 94%] Built target myslam
[ 97%] Building CXX object CMakeFiles/mono_tum.dir/Examples/Monocular/mono_tum.cc.o
[100%] Linking CXX executable ../../../../Examples/Monocular/mono_tum
[100%] Built target mono_tum
[ 89%] No install step for 'ep_orbslam2'
[ 91%] Completed 'ep_orbslam2'
[100%] Built target ep_orbslam2
~/slambook/ORB_SLAM2-9000b1c/build
```

图 3: 编译完成

2. 如何把 myvideo.cpp 加入到 ORB-SLAM2 工程中：在 Examples 文件夹里新建 myvideo 子文件夹，再在 myvideo 文件夹中加入 myvideo.cpp、myvideo.mp4、myvideo.yaml、Vocabulary/ORBvoc.txt 等文件。同时在 CMakeLists.txt 中添加以下内容：

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

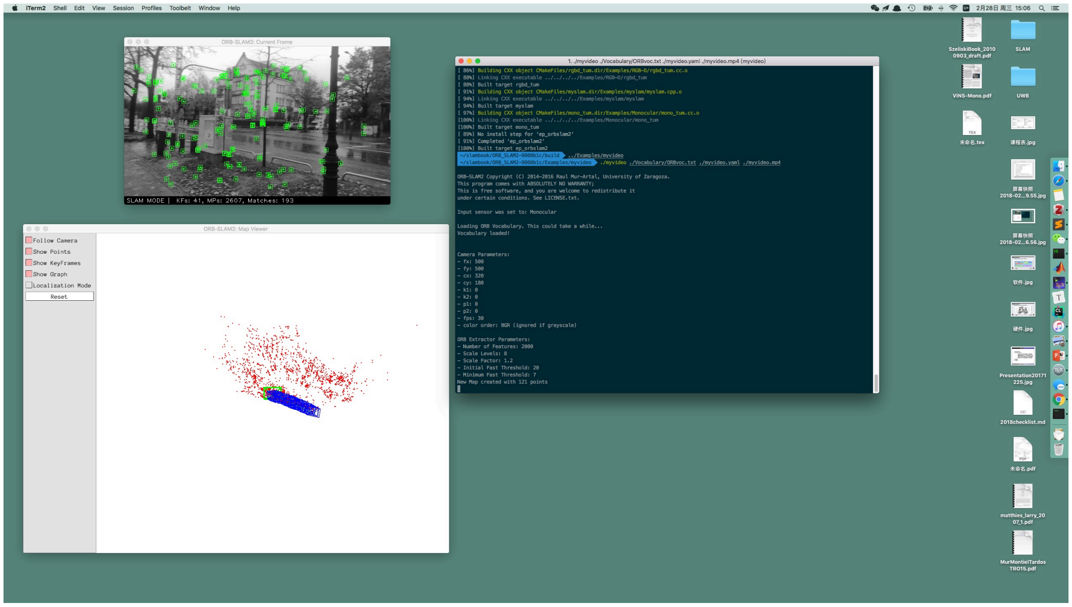
```
target_link_libraries(myvideo ${PROJECT_NAME})
```

3. 执行命令

```
$ ./myvideo ./Vocabulary/ORBvoc.txt ./myvideo.yaml ./myvideo.mp4
```

运行截图见图 4。

体会：实际运行效果与光照、纹理、运动速度有关，运动太快会导致特征点跟踪丢失。同时，



```
[ 86%] Building CXX object CMakeFiles/rgbd_tum.dir/Examples/RGB-D/rgbd_tum.cc.o
[ 88%] Linking CXX executable ../../../../Examples/RGB-D/rgbd_tum
[ 88%] Built target rgbd_tum
[ 91%] Building CXX object CMakeFiles/myslam.dir/Examples/myslam/myslam.cpp.o
[ 94%] Linking CXX executable ../../../../Examples/myslam/myslam
[ 94%] Built target myslam
[ 97%] Building CXX object CMakeFiles/mono_tum.dir/Examples/Monocular/mono_tum.cc.o
[100%] Linking CXX executable ../../../../Examples/Monocular/mono_tum
[100%] Built target mono_tum
[ 89%] No install step for 'ep_orbslam2'
[ 91%] Completed 'ep_orbslam2'
[100%] Built target ep_orbslam2
~/slambook/ORB_SLAM2-9000b1c/build  ../Examples/myvideo
~/slambook/ORB_SLAM2-9000b1c/Examples/myvideo  ./myvideo ./Vocabulary/ORBvoc.txt ./myvideo.yaml ./myvideo.mp4

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
New Map created with 121 points
```

图 4: myvideo 运行截图

需要注意 macOS 和 Ubuntu 的差异，**macOS 上 Viewer::Run() 需要在主线程中调用，当从非主线程调用 UI 库时易出错**。故在 macOS 上运行时，需要修改源程序 **myvideo.cpp**，我的修改方案为：

```
#include<iostream>
#include<algorithm>
#include<fstream>
#include<future>
#include<chrono>
#include<thread>
#include <opencv2/opencv.hpp>
#include<System.h>
using namespace std;
int processing(char **argv, ORB_SLAM2::System *slamPtr);

int main(int argc, char **argv)
{
    if(argc != 4)
    {
        cerr << endl << "Usage!" << endl;
        return 1;
    }

    ORB_SLAM2::System SLAM(argv[1],argv[2],ORB_SLAM2::System::MONOCULAR,true);
    auto resultFuture = async(launch::async, processing, argv, &SLAM);
    SLAM.RunViewer();
    return resultFuture.get();
}

int processing(char **argv, ORB_SLAM2::System *slamPtr)
{
    ORB_SLAM2::System& SLAM = *slamPtr;
    cv::VideoCapture cap(argv[3]);
    // cv::VideoCapture cap(0);

    if (!cap.isOpened())
    {
        cerr <<"Could␣not␣open␣camera␣feed."<< endl;
        return -1;
    }

    auto start = chrono::system_clock::now();

    while (1)
    {
        cv::Mat frame;
        cap >> frame;
        if ( frame.data == nullptr )
        {
            cout << "frame.data␣wrong." << endl;
            break;
        }

        cv::Mat frame_resized;
        cv::resize(frame, frame_resized, cv::Size(640,360));

        auto now = chrono::system_clock::now();
        auto timestamp = chrono::duration_cast<chrono::milliseconds>(now -
            start);
        SLAM.TrackMonocular(frame_resized, double(timestamp.count())/1000.0);
        // cv::waitKey(30);
    }
    SLAM.Shutdown();
    return 0;
}
```