

一、熟悉linux

计算机领域的绝大多数科研人员都在 Linux 下工作,不掌握 Linux 会使你在研究道路上寸步难行。Linux 系统的基本知识亦是学习本课程的先决条件。如果你还未接触过 Linux,请阅读本次习题中提供的材料(见 books/目录下)。建议阅读《鸟哥的 Linux 私房菜》第 1、2 章了解 Linux 历史,第 5-8 章了解基础操作。如果你在用 Ubuntu,也可以参考 Ubuntu 维基上自带的指南:<http://wiki.ubuntu.org.cn/>

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?

`sudo apt-get install XXX` 例如: `sudo apt-get install gcc`

安装位置

软件分布在系统的各个文件夹中

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?

系统与软件在运行过程中,通过环境变量寻找参数,例如:第三方库,某些软件位置等

ubuntu存储环境变量的文件如下:

- `etc/environment`: 设置整个系统的环境
- `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`: 每次打开shell执行一次
- `~/.profile`: 用户登录执行一次,仅对当前用户有效
- `~/.bashrc`: 每次打开shell执行一次,仅对当前用户有效

定义新的环境变量 确认需求,在上述对应的文件后添加: `$ export PATH=my_path:$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。

根目录结构:

```
allen@big:/$ tree -L 1
.
├── bin
├── boot
├── cdrom
├── dev
├── etc
├── home
├── initrd.img -> boot/i
├── initrd.img.old -> bo
├── lib
├── lib64
├── lost+found
├── media
├── mnt
├── opt
├── proc
├── root
├── run
├── sbin
├── snap
├── srv
├── sys
├── tmp
├── usr
├── var
├── vmlinuz -> boot/vmli
└── vmlinuz.old -> boot/
```

目录用途:

- /bin: 二进制可执行文件的目录
- /media: 可移除装置, 包括软盘、光盘、硬盘、U盘、各种盘
- /etc: 系统主要配置文件, 常用的有/etc/init.d/:所有服务的自启动脚本
- /var: 存放系统运作过程生成的各种文件 (log, run, cache等)
- /usr: 几乎所有的应用程序和文件目录, 类似windows下的C:\Windows\ + C:\Program files\的综合体

4. 假设我要给 a.sh 加上可执行权限,该输入什么命令?

```
sudo 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?

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

二、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。

AR, 移动机器人, 自动驾驶, 无人机

2.SLAM定位与建图是什么关系?为什么在定位的同时需要建图?

定位和建图二者相辅相成, 高精地图确保准确的定位, 同时, 定位的准确性极大影响建图的效果。

3.SLAM 发展历史如何?我们可以将它划分成哪几个阶段?

大致有如下里程碑:

1. EKF-SLAM: 滤波器实用非常重要的一步
2. FASTSLAM: 加入粒子滤波重定位
3. Graph-SIAM: 目前90%的学者都在做这一块吧

激光slam部分:

GMapping

LOAM: Ji Zhang

V-LOAM: 视觉做补偿

Cartographer: google开源

4.列举三篇在 SLAM 领域的经典文献。

2008年: EKF monocular SLAM 将slam推向单目相机, 低成本slam成为可能

Civera J, Davison A J, Montiel J M M. Inverse depth parametrization for monocular SLAM[J]. IEEE transactions on robotics, 2008, 24(5): 932-945.

ORB-SLAM

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.

Vins-mono

Qin T, Li P, Shen S. Vins-mono: A robust and versatile monocular visual-inertial state estimator[J]. IEEE Transactions on Robotics, 2018, 34(4): 1004-1020.

三、CMake 练习

cmake 是一种常用、方便的,用于组织 Linux 下 C++ 程序的工具。有许多库,例如 OpenCV、g2o、Ceres 等,都用 cmake 组织它们的工程。所以,不管是使用别人的库,还是编写自己的库,都需要掌握一些 cmake 的基本知识。

完成以下工作: 书写一个由 cmake 组织的 C++ 工程,要求如下:

1. include/hello.h 和 src/hello.c 构成了 libhello.so 库。hello.c 中提供一个函数 sayHello(),调用此函数时往屏幕输出一行“Hello SLAM”。我们已经为你准备了 hello.h 和 hello.c 这两个文件,见“code/”目录下。
2. 文件 useHello.c 中含有一个 main 函数,它可以编译成一个可执行文件,名为“sayhello”。
3. 默认用 Release 模式编译这个工程。
4. 如果用户使用 sudo make install,那么将 hello.h 放至/usr/local/include/下,将 libhello.so 放至/usr/local/lib/下。请按照上述要求组织源代码文件,并书写 CMakeLists.txt。

问题

提供给我们的代码是.c后缀,但是里面却写着 `#include <iostream>` 导致编译不通过

我把c文件都改成cpp后缀了

回答:

CMakeLists.txt文件内容

```
SET (PROJECT_NAME cmaketest)

PROJECT(${PROJECT_NAME})
CMAKE_MINIMUM_REQUIRED(VERSION 2.8)

# 设置编译模式
SET(CMAKE_BUILD_TYPE Release) # Release, RelWithDebInfo , Debug

# 设置可执行文件的输出路径和库文件路径
SET(EXECUTABLE_OUTPUT_PATH ${PROJECT_SOURCE_DIR}/bin)
SET(LIBRARY_OUTPUT_PATH ${PROJECT_SOURCE_DIR}/lib)

set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11")

# 头文件
INCLUDE_DIRECTORIES(${PROJECT_SOURCE_DIR}/include)

# 生成共享库
add_library(hello SHARED ${PROJECT_SOURCE_DIR}/src/hello.cpp)

# 生成可执行文件
add_executable(sayHello ${PROJECT_SOURCE_DIR}/src/useHello.cpp)

# 将库文件链接到可执行文件上
target_link_libraries(sayHello hello)

# make install 部分

install(TARGETS hello LIBRARY DESTINATION /usr/local/lib)
install(FILES ${PROJECT_SOURCE_DIR}/include/hello.h DESTINATION /usr/local/include)
```

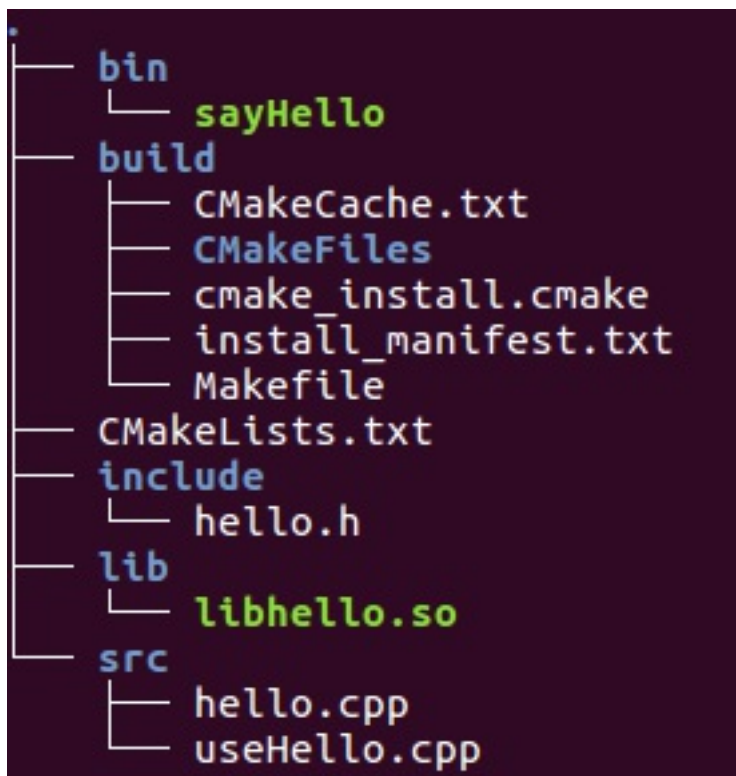
运行结果

```

allen@big:~/Documents/第五期高博视觉slam/我的作业/PA1/code/cmaketest/build$ cd ..
allen@big:~/Documents/第五期高博视觉slam/我的作业/PA1/code/cmaketest$ cd build/
allen@big:~/Documents/第五期高博视觉slam/我的作业/PA1/code/cmaketest/build$ ls
allen@big:~/Documents/第五期高博视觉slam/我的作业/PA1/code/cmaketest/build$ cmake ..
-- The C compiler identification is GNU 5.4.0
-- The CXX compiler identification is GNU 5.4.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/allen/Documents/第五期高博视觉slam/我的作业/PA1/code/cmaketest/build
allen@big:~/Documents/第五期高博视觉slam/我的作业/PA1/code/cmaketest/build$ makeScanning dependencies of target hello
[ 25%] Building CXX object CMakeFiles/hello.dir/src/hello.cpp.o
[ 50%] Linking CXX shared library ../lib/libhello.so
[ 50%] Built target hello
Scanning dependencies of target sayHello
[ 75%] Building CXX object CMakeFiles/sayHello.dir/src/useHello.cpp.o
[100%] Linking CXX executable ../bin/sayHello
[100%] Built target sayHello
allen@big:~/Documents/第五期高博视觉slam/我的作业/PA1/code/cmaketest/build$ sudo make install
[sudo] password for allen:
[ 50%] Built target hello
[100%] Built target sayHello
Install the project...
-- Install configuration: "Release"
-- Installing: /usr/local/lib/libhello.so
-- Installing: /usr/local/include/hello.h
allen@big:~/Documents/第五期高博视觉slam/我的作业/PA1/code/cmaketest/build$ cd ../bin
allen@big:~/Documents/第五期高博视觉slam/我的作业/PA1/code/cmaketest/bin$ ./sayHello
Hello SLAM

```

最终项目的目录如下：



四、理解ORB-SLAM2框架

1. 从 github.com 下载 ORB-SLAM2 的代码

```

allen@big:~$ git clone https://github.com/raulmur/ORB_SLAM2.git
Cloning into 'ORB_SLAM2'...
remote: Enumerating objects: 566, done.
remote: Total 566 (delta 0), reused 0 (delta 0), pack-reused 566
Receiving objects: 100% (566/566), 41.41 MiB | 2.48 MiB/s, done.
Resolving deltas: 100% (177/177), done.
Checking connectivity... done.
  
```

2. 分析ORB-SLAM2代码结构

(a) ORB-SLAM2 将编译出什么结果?有几个库文件和可执行文件?

回答:

一个库文件: ORB_SLAM2.so

6个可执行测试文件:

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?

- include文件夹: 生成库的头文件
- src文件夹: 生成库的源码

- Example: 可执行文件与不同相机对应的相机标定文件

(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?

回答:

链接的库文件包括:

自身编译生成的动态库如下:

```
src/System.cc
src/Tracking.cc
src/LocalMapping.cc
src/LoopClosing.cc
src/ORBextractor.cc
src/ORBmatcher.cc
src/FrameDrawer.cc
src/Converter.cc
src/MapPoint.cc
src/KeyFrame.cc
src/Map.cc
src/MapDrawer.cc
src/Optimizer.cc
src/PnPsolver.cc
src/Frame.cc
src/KeyFrameDatabase.cc
src/Sim3Solver.cc
src/Initializer.cc
src/Viewer.cc
```

外部链接的库包括:

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

五、使用摄像头或视频运行 ORB-SLAM2

1.顺利编译结果

```

-- Build files have been written to: /home/xbot/ORB_SLAM2/build
Scanning dependencies of target ORB_SLAM2
[ 6%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/System.cc.o
[ 6%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/Tracking.cc.o
[ 12%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/LoopClosing.cc.o
[ 21%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/ORBmatcher.cc.o
[ 18%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/ORBextractor.cc.o
[ 21%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/FrameDrawer.cc.o
[ 15%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/LocalMapping.cc.o
[ 25%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/Converter.cc.o
[ 31%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/Frame.cc.o
[ 31%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/Initializer.cc.o
[ 34%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/Optimizer.cc.o
[ 37%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/MapDrawer.cc.o
[ 40%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/Sim3Solver.cc.o
[ 46%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/Viewer.cc.o
[ 46%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/PnPsolver.cc.o
[ 50%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/KeyFrameDatabase.cc.o
[ 53%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/MapPoint.cc.o
[ 56%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/Map.cc.o
[ 59%] Building CXX object CMakeFiles/ORB_SLAM2.dir/src/KeyFrame.cc.o
[ 62%] Linking CXX shared library ../lib/libORB_SLAM2.so
[ 62%] Built target ORB_SLAM2
Scanning dependencies of target rgbd_tum
Scanning dependencies of target mono_tum
Scanning dependencies of target mono_kitti
Scanning dependencies of target stereo_kitti
Scanning dependencies of target mono_euroc
Scanning dependencies of target stereo_euroc
[ 65%] Building CXX object CMakeFiles/rgb_tum.dir/Examples/RGB-D/rgb_tum.cc.o
[ 68%] Building CXX object CMakeFiles/mono_tum.dir/Examples/Monocular/mono_tum.c
c.o
[ 71%] Building CXX object CMakeFiles/stereo_euroc.dir/Examples/Stereo/stereo_eu
roc.cc.o
[ 75%] Building CXX object CMakeFiles/mono_kitti.dir/Examples/Monocular/mono_kit
ti.cc.o
[ 78%] Building CXX object CMakeFiles/mono_euroc.dir/Examples/Monocular/mono_eur
oc.cc.o
[ 81%] Building CXX object CMakeFiles/stereo_kitti.dir/Examples/Stereo/stereo_ki
tti.cc.o
[ 84%] Linking CXX executable ../Examples/Monocular/mono_euroc
[ 87%] Linking CXX executable ../Examples/Stereo/stereo_kitti
[ 87%] Built target mono_euroc
[ 87%] Built target stereo_kitti
[ 93%] Linking CXX executable ../Examples/Monocular/mono_tum
[ 93%] Linking CXX executable ../Examples/Monocular/mono_kitti
[ 96%] Linking CXX executable ../Examples/RGB-D/rgb_tum
[100%] Linking CXX executable ../Examples/Stereo/stereo_euroc
[100%] Built target mono_tum
[100%] Built target mono_kitti
[100%] Built target rgb_tum
[100%] Built target stereo_euroc
xbot@T470p:~/ORB_SLAM2$

```

注意：这里名字不一样是因为我更换了电脑进行编译。

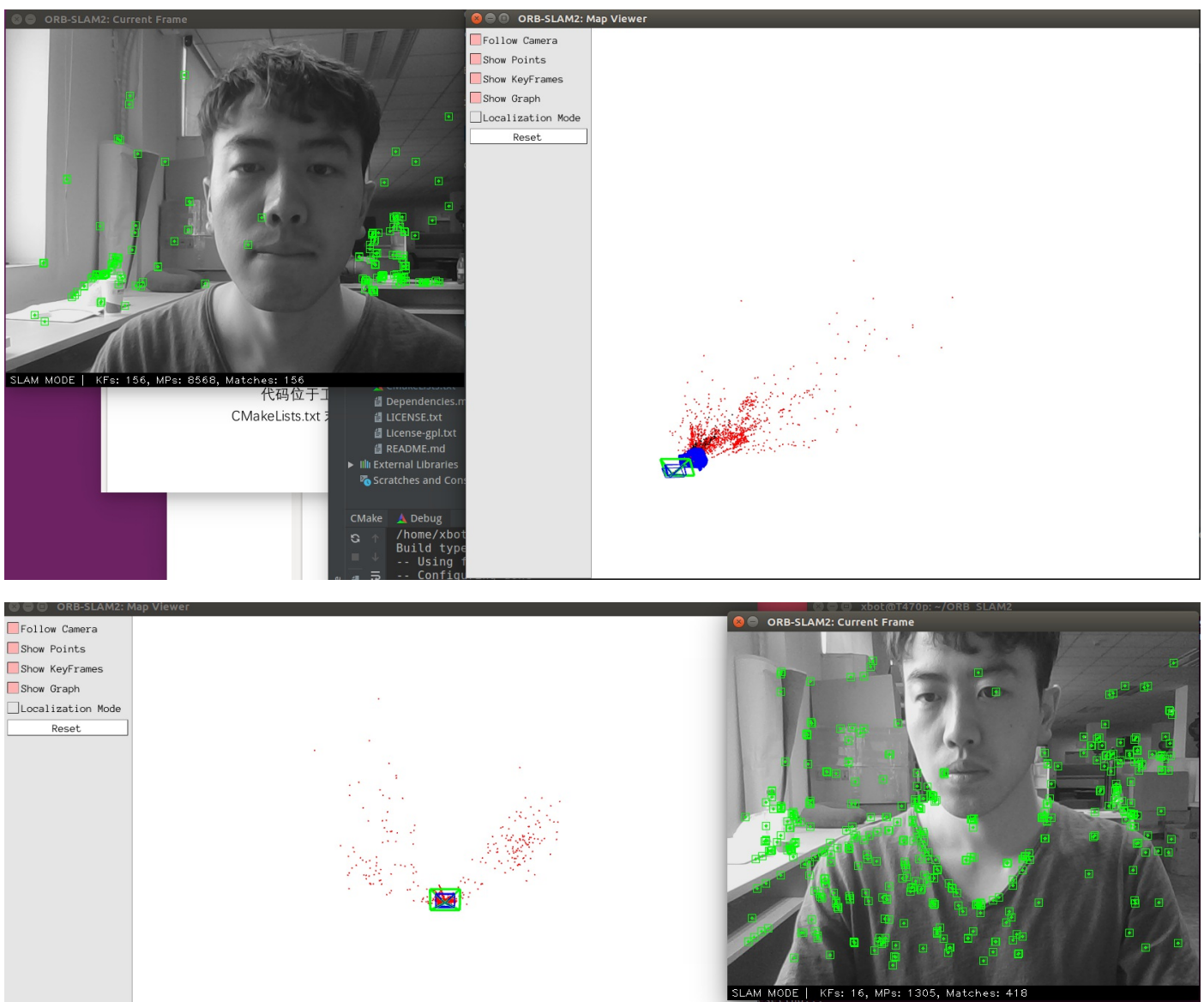
2.我的cmakelist修改方案

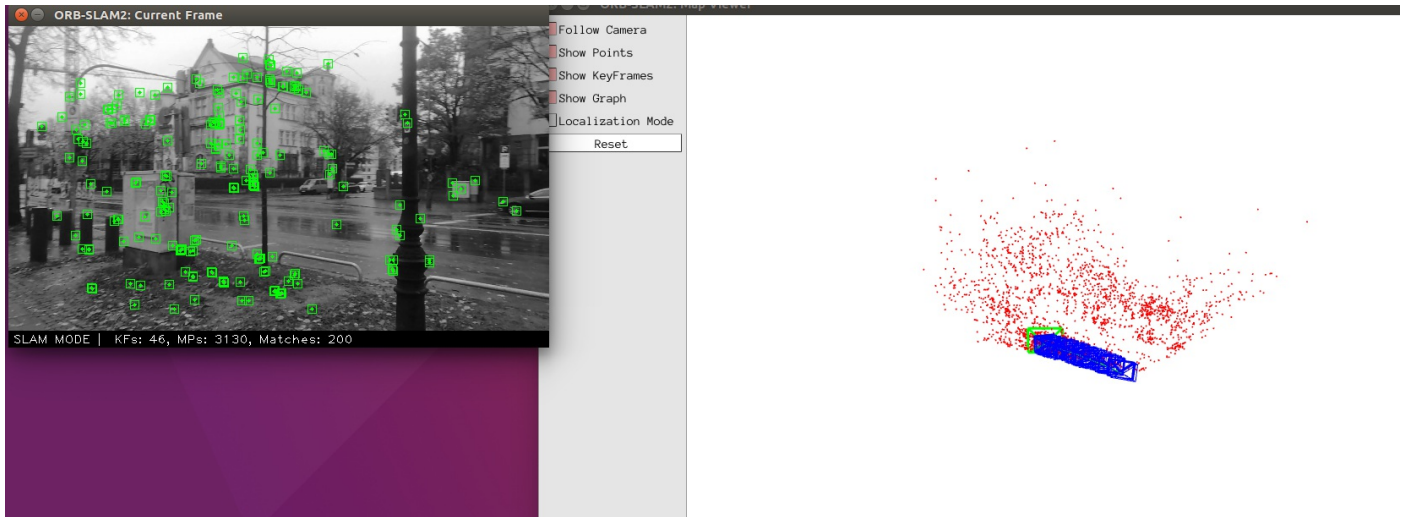
#生成调用myvideo.mp4的可执行文件

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

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

3.实际运行截图





结论：

1. 移动要缓慢，尤其是旋转，否则就会丢失特征。
2. 对光照敏感
3. 从myvideo来看，轨迹大致一致。

六、感悟

我在实验室的电脑跑ORB-SLAM2的时候，出现好多奇怪的问题，opencv各种报错，很心累。后来换了台电脑，一切都很顺利。有时候觉得配环境真的挺考验人的。

本期最大的感触就是：作业量够大，我做得很开心。因为之前有看过高博的书，一直以为深蓝课程作业也不过是高博书上的课后题。

这期报名之后，我认为深蓝在这门课的课后题上是下了功夫的。我觉得很有收获。很开心！！