



从硬件到内核空间是由系统自动完成的， 在app中按照要求的格式读取触摸屏设备文件中的数据

格式：

struct input\_event {

struct timeval time;

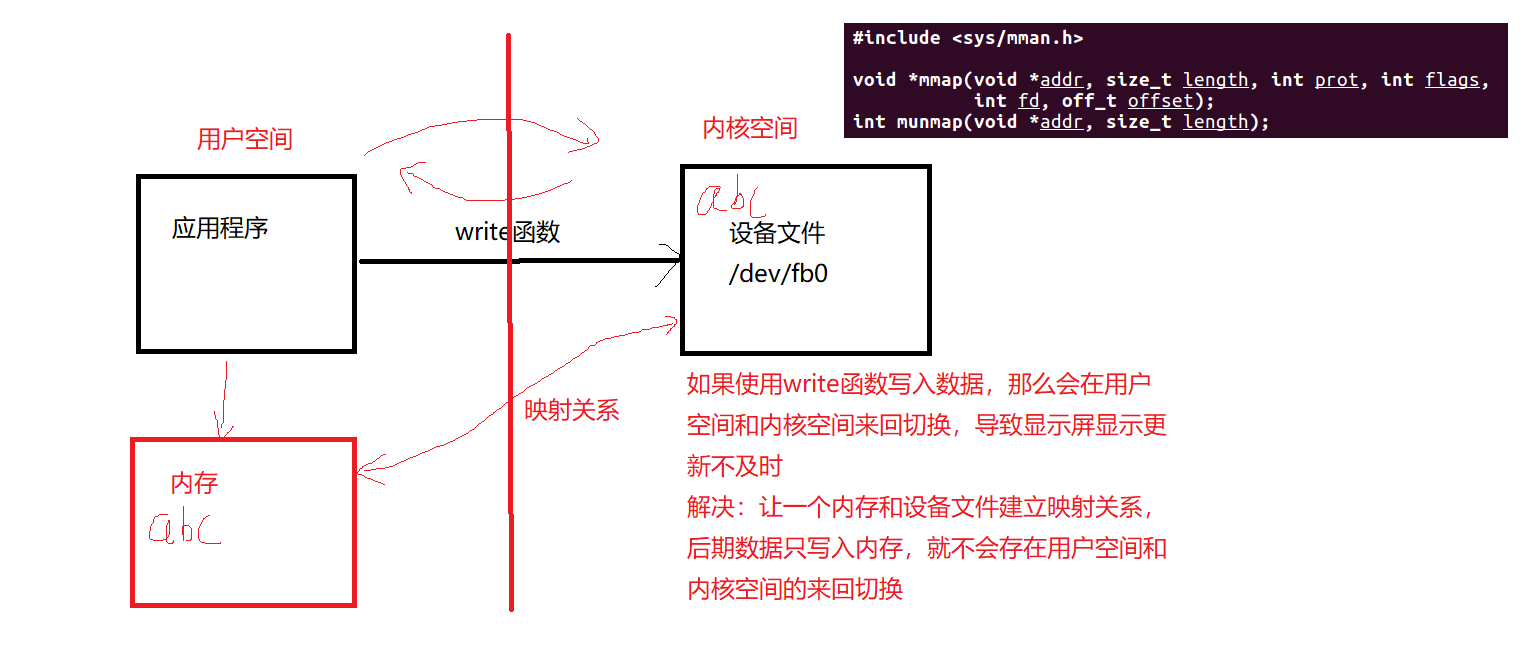
\_\_u16 type;

\_\_u16 code;

\_\_s32 value;

};

内存映射



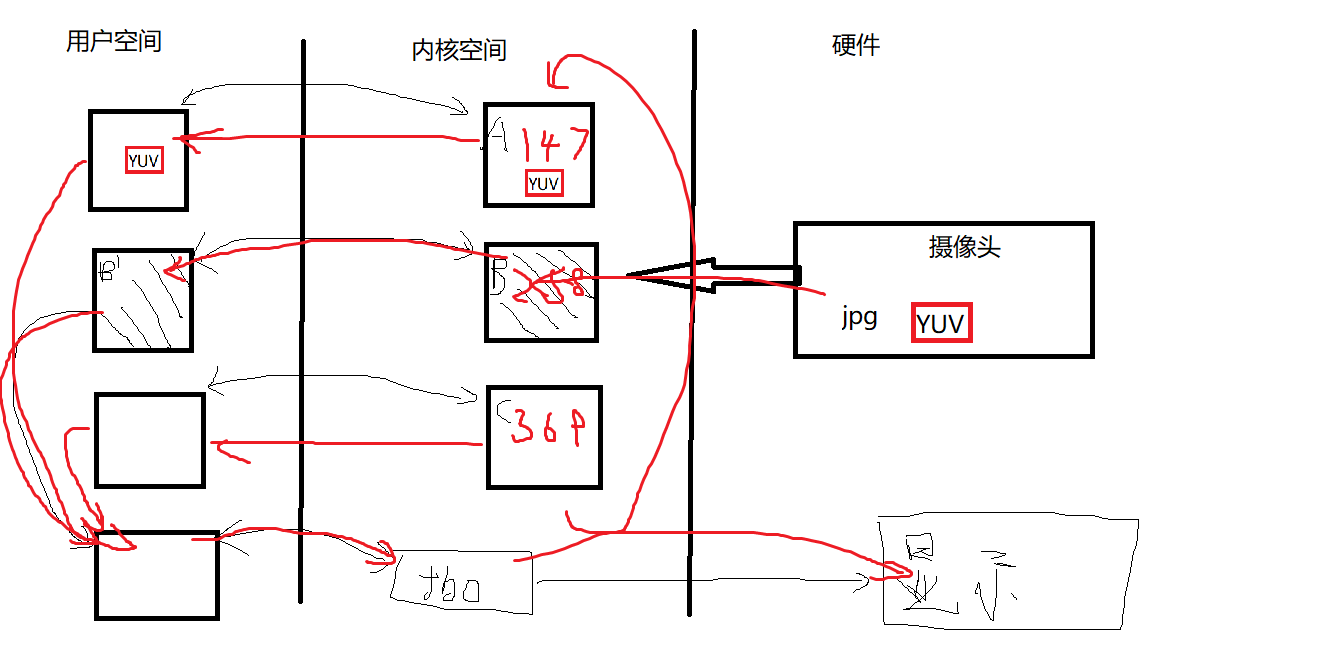
一、V4L2

video for linux 2

摄像头作为一个硬件连接到开发板后，在系统中也会有一个设备文件进行代表。

设备文件是 /dev/video7

任务： 参照 【camerainfo】，实现摄像头的实时监控显示



1. 显示v4f2

重点：截取图象的两种方法  
1,用mmap（内存映射）方式截取视频

<https://blog.csdn.net/jiang_dlut/article/details/5904252>

2.

//接受车牌识别完成信号

    // 1，打开摄像头

    // 2，查看摄像头信息

  // 3，设置摄像头信息

     //准备映射表

    // 打开LCD设备

     // 获取LCD显示器的设备参数

    //lcd设备文件 mmap映射到本进程地址空间

// 清屏

  // 显存双缓冲优化

    // bzero(fb, 2 \* lcdinfo.xres \* lcdinfo.yres \* 4);

    // 获取RGB偏移量

    // 设置即将要申请的摄像头缓存的参数

// 使用该参数reqbuf来申请缓存

  // 根据刚刚设置的reqbuf.count的值，来定义相应数量的struct v4l2\_buffer

    // 每一个struct v4l2\_buffer对应内核摄像头驱动中的一个缓存

    // 启动摄像头数据采集

    //创建一个新的线程，不断的获取触摸屏坐标

    //创建一个新的线程，完成数据库操作

    //创建一个进程，让该进程去执行车牌识别的程序

    pid = fork();

  // 开始抓取摄像头数据并在屏幕播放视频

        //wait4touch(); //如果没有点击触摸屏会阻塞，导致后面的代码无法执行

        // 从队列中取出填满数据的缓存，让摄像头不再写入该空间

        //获取摄像头的yuv数据，转换后显示到开发板屏幕

      // 将已经读取过数据的缓存块重新置入队列中，让摄像头将数据写入该空间

    //设置收到SIGUSR1信号之后去调用函数func，该函数实现车牌识别

    signal(SIGUSR1, func);

    fifoIn = open("/tmp/LPR2SQLitIn", O\_RDWR);

    fifoOut = open("/tmp/LPR2SQLitOut", O\_RDWR);

    //创建一个新的线程，用户的模式切换

    pthread\_t tid;

    pthread\_create(&tid, NULL, fun, NULL);

**1. 实验提纲指导**

**1.1 验收效果**

实现对指定照片进行车牌识别，输出车牌字符串和确信率。

**1.2 实验目的**

* 了解openCV使用流程
* 了解HyperLPR使用流程

**2. 知识准备**

**2.1 openCV**

OpenCV是一个开源的计算机视觉和机器学习软件库，主要为开发计算机视觉程序提供一组公共的底层架构。



OpenCV库有超过2500个已优化的算法，包含了最先进的计算机视觉和机器学习算法，这些算法可以被用来检测和识别人脸，跟踪移动的物体，提取物体的三维模型等，openCV已经有超过5万用户群体，估计下载数量超过1500万。

本实训案例中所使用的的车牌识别开源项目HyperLPR就是基于openCV库开发的，因此在移植部署HyperLPR之前，需要先准备好openCV库。

**2.2 HyperLPR**

[HyperLPR](https://gitee.com/zeusees/HyperLPR.git) 是一款高性能开源中文车牌识别框架，可以识别国内常见的各种车牌：

* 单行蓝牌
* 单行黄牌



HyperLPR 在码云（gitee）可直接[下载](https://gitee.com/zeusees/HyperLPR.git)，使用cmake管理工程，源码提供了丰富的测试用例。另外，由于6818没有专用于深度学习的神经网络芯片（NPU），因此运行该识别程序速度性能稍差。

**2.3 cmake**

cmake 是一个跨平台的编译工具，它并不直接构建出最终的软件，而是生成标准的Makefile文件（在类Unix平台），从而可以非常方便地直接使用命令 make 来进行源码编译。

使用 cmake，主要注意以下几点：

**一，编译目录与源码目录要分开**  
一般而言，camke 会将配置和编译过程所产生的文件保存在其执行目录中，这一过程会产生大量的文件，因此 cmake 命令语法规定不能在源码顶层目录中运行，一般的做法是，单独创建一个空目录（例如叫build），然后在空目录中执行 cmake。

例如：

# 创建并进入目录 build/

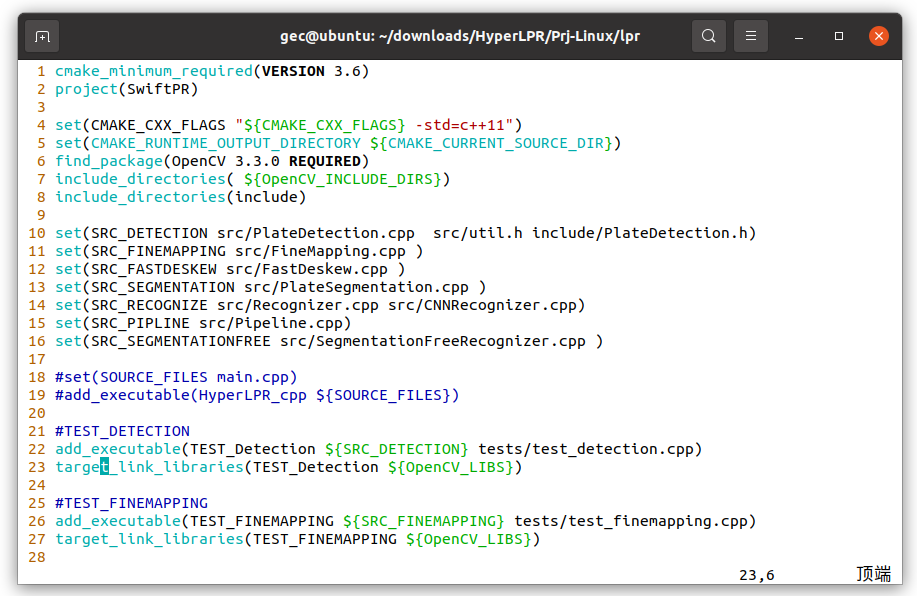
gec@ubuntu:~$ mkdir build/

gec@ubuntu:~$ cd build/

# 在空目录 build/ 中执行cmake，并指定源码路径

gec@ubuntu:~/build$ cmake /home/gec/opencv-3.4.10

**二，CMakeList.txt**  
在执行 cmake 的时候，默认会读取所指定路径下的 CMakeList.txt 文件，所以如果要修改编译的选项，比如编译器、生成的文件名、编译所依赖的库等信息一般都是通过修改 CMakeList.txt 来达到，比如：

  
CMakeList.txt 的一般语法结构

从上面可以看到，在 CMakeList.txt 中可以：

*#定义变量及其值：*

set(SRC\_DETECTION xxx yyy)

*#添加编译文件及其依赖：*

add\_executable(app xxx yyy)

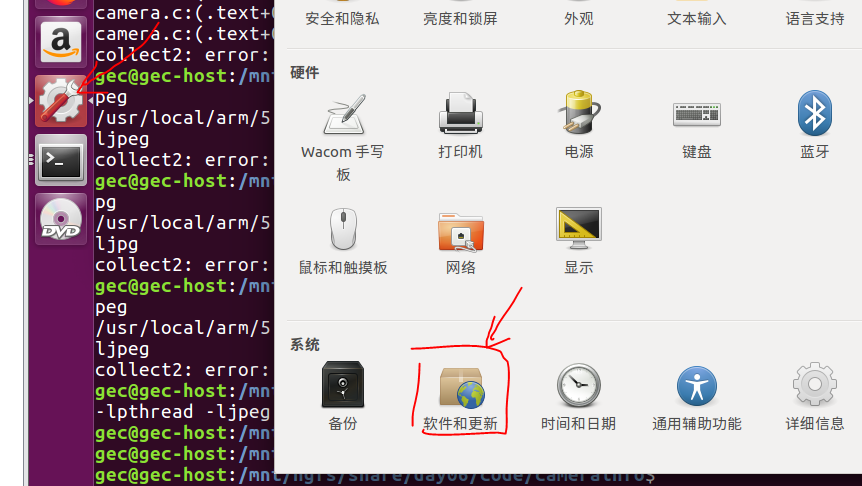
*#指定编译某执行文件所需库：*

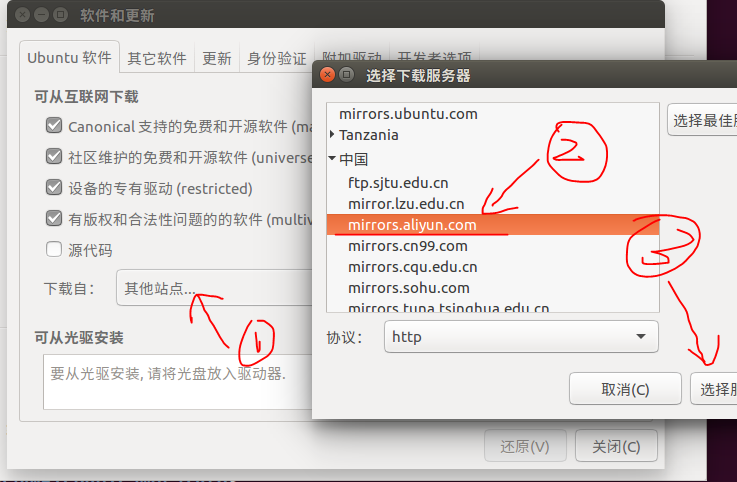
target\_link\_libraries(app xxx yyy)

**3. 实验步骤**

**3.1 环境准备**

**1、设置网络站点**





点击【关闭】

不管是 openCV 还是 HyperLPR，都是用 cmake 来管理源码的，因此要确保在 ubuntu 中装好 cmake，命令如下：

# cmake安装：

gec@ubuntu:~$ sudo apt-get update

gec@ubuntu:~$ sudo apt-get install cmake

**3.2 编译 openCV**

第一步，[点击下载 openCV-3.4.10](http://vm.yueqian.com.cn:8886/group1/M00/0C/3F/wKgP3GGdppKAFwUABXhbORpVMfo182.zip?token=null&ts=null&filename=opencv-3.4.10.zip)  
注意，HyperLPR默认基于openCV-3.4.10，不建议使用其他版本。

第二步，将上述源码解压到 ubuntu 家目录下：

gec@ubuntu:~$ cp /mnt/hgfs/share/day07/opencv-3.4.10源码包.zip .

gec@ubuntu:~$ unzip opencv-3.4.10源码包.zip

第三步，将 [arm.cmake](http://vm.yueqian.com.cn:8886/group1/M00/0C/3F/wKgP3GGdu96AW0nNAAACJi_1ffM7.cmake?token=null&ts=null&filename=arm.cmake) 拷贝到源码目录中，该文件用来指定交叉工具链的名称和路径信息，原始内容如下：

gec@ubuntu:~/opencv-3.4.10$ touch arm.cmake

gec@ubuntu:~/opencv-3.4.10$ gedit arm.cmake

gec@ubuntu:~/opencv-3.4.10$ cat -n arm.cmake

1 set(CMAKE\_SYSTEM\_NAME Linux)

2 set(TOOLCHAIN\_DIR /usr/local/arm/5.4.0/usr)

3

4 set(CMAKE\_C\_COMPILER ${TOOLCHAIN\_DIR}/bin/arm-linux-gcc)

5 set(CMAKE\_CXX\_COMPILER ${TOOLCHAIN\_DIR}/bin/arm-linux-g++)

6

7 set(CMAKE\_SYSTEM\_PROCESSOR arm)

8

9 set(CMAKE\_FIND\_ROOT\_PATH ${TOOLCHAIN\_DIR}

10 ${TOOLCHAIN\_DIR}/arm-none-linux-gnueabi/include

11 ${TOOLCHAIN\_DIR}/arm-none-linux-gnueabi/lib)

12

13 SET(CMAKE\_FIND\_ROOT\_PATH\_MODE\_PROGRAM NEVER)

14

15 set(CMAKE\_FIND\_ROOT\_PATH\_MODE\_LIBRARY ONLY)

16 set(CMAKE\_FIND\_ROOT\_PATH\_MODE\_INCLUDE ONLY)

17

18 set(CMAKE\_INSTALL\_PREFIX /home/gec/tools/opencv-3.4)

gec@ubuntu:~/opencv-3.4.10$

**注意**，要根据自己电脑的实际情况适当修改，主要关注上述文档的**第2行**和**第18行**：

2 set(TOOLCHAIN\_DIR /usr/local/arm/5.4.0/usr)

...

...

18 set(CMAKE\_INSTALL\_PREFIX /home/gec/tools/opencv-3.4)

第2行可以使用命令 which arm-linux-gcc 查看工具链的实际位置，不能盲目照抄。

第18行可以根据自己的需求去改最终安装路径。

第四步，在源码中创建 build 并进入目录执行命令：

# 创建并进入 build/

gec@ubuntu:~/opencv-3.4.10$ mkdir build

gec@ubuntu:~/opencv-3.4.10$ cd build/

#创建最终安装路径

gec@ubuntu:~/opencv-3.4.10/build$ mkdir ~/tools

gec@ubuntu:~/opencv-3.4.10/build$ mkdir ~/tools/opencv-3.4

# 执行配置、编译（需要大概30-50分钟）和安装

gec@ubuntu:~/opencv-3.4.10/build$ cmake /home/gec/opencv-3.4.10/ -DCMAKE\_TOOLCHAIN\_FILE=arm.cmake

gec@ubuntu:~/opencv-3.4.10/build$ make -j4

gec@ubuntu:~/opencv-3.4.10/build$ sudo make install

说明：

* 编译指令 make -j4 中的参数代表使用4条线程，可以根据实际情况修改为 j2、j8 等
* 由于 openCV 项目较大，编译过程一般需要30-50分钟左右，要耐心等待
* 安装指令 make install 执行完毕之后，将会在 build 目录下出现 install 目录，内容如下：

其中文件夹 lib/ 里面的文件就是编译好的 openCV 库文件，在接下来的HyperLPR 源码编译和之后开发板运行时都需要用到。



**3.3 编译 HyperLPR**

第一步，[点击下载 zeusees-HyperLPR-master.zip](http://vm.yueqian.com.cn:8886/group1/M00/0C/3F/wKgP3GGdvKGAI8paBmDUUHpPFtQ230.zip?token=null&ts=null&filename=zeusees-HyperLPR-master.zip)

第二步，将上述源码解压到 ubuntu 家目录下：

gec@ubuntu:~$ unzip zeusees-HyperLPR-master.zip

第三步，进入 HyperLPR/Prj-Linux/lpr，修改里面的 [CMakeLists.txt](http://vm.yueqian.com.cn:8886/group1/M00/0C/3F/wKgP3GGd4QSAfz3kAAAFpR5KO0w564.txt?token=null&ts=null&filename=CMakeLists.txt)：

gedit  [CMakeLists.txt](http://vm.yueqian.com.cn:8886/group1/M00/0C/3F/wKgP3GGd4QSAfz3kAAAFpR5KO0w564.txt?token=null&ts=null&filename=CMakeLists.txt)

1. 注释掉版本匹配检查

将版本检查改为3.5.1

cmake\_minimum\_required(VERSION 3.5.1)

#find\_package(OpenCV 3.3.0 REQUIRED)

1. 增加头文件路径：

# 交叉工具链相关的头文件：

include\_directories(/usr/local/arm/5.4.0/usr/include)

include\_directories(/usr/local/arm/5.4.0/usr/arm-none-linux-gnueabi/include)

# openCV的头文件

include\_directories(/home/gec/tools/opencv-3.4/include)

# HyperLP自身头文件

include\_directories(include)

以上头文件的路径不能完全照抄，要根据具体情况来写。

1. 指定交叉工具链

set(CMAKE\_C\_COMPILER /usr/local/arm/5.4.0/usr/bin/arm-linux-gcc)

set(CMAKE\_CXX\_COMPILER /usr/local/arm/5.4.0/usr/bin/arm-linux-g++)

可以使用which命令来查看你的工具链所在地地方，不能盲目照抄。

1. 指定 openCV 库的名称和路径

set(OpenCV\_LIBS

opencv\_core

opencv\_dnn

opencv\_features2d

opencv\_flann

opencv\_highgui

opencv\_imgcodecs

opencv\_imgproc

opencv\_ml

opencv\_objdetect

opencv\_photo

opencv\_shape

opencv\_stitching

opencv\_superres

opencv\_videoio

opencv\_video

opencv\_videostab)

LINK\_DIRECTORIES(/home/gec/tools/opencv-3.4/lib)

以上LINK\_DIRECTORIES(/home/gec/tools/opencv-3.4/lib)是 openCV 实际安装路径（参考[3.2](http://vm.yueqian.com.cn:8886/previewCourse.html?courseId=1383625721834745857" \l "3.2)），不能盲目照抄。

1. 增加编译测试用例的语句：

add\_executable(alpr main.cpp ${SRC\_DETECTION} ${SRC\_FINEMAPPING} ${SRC\_FASTDESKEW} ${SRC\_SEGMENTATION} ${SRC\_RECOGNIZE} ${SRC\_PIPLINE} ${SRC\_SEGMENTATIONFREE})

target\_link\_libraries(alpr ${OpenCV\_LIBS})

说明：

* add\_executable( ) 语句指明了将测试程序 main.cpp 编译生成 alpr，并指明编译时所依赖的各个其他源文件。
* target\_link\_libraries( ) 语句指明了程序 alpr 所依赖的opencv库文件具体内容。

第四步，下载测试用例 [main.cpp](http://vm.yueqian.com.cn:8886/group1/M00/0C/3F/wKgP3GGd4ZGAYL_NAAAEIPm-eMU515.cpp?token=null&ts=null&filename=main.cpp)，并将此测试用例复制到 HyPerLPR/Prj-Linux/lpr 中。

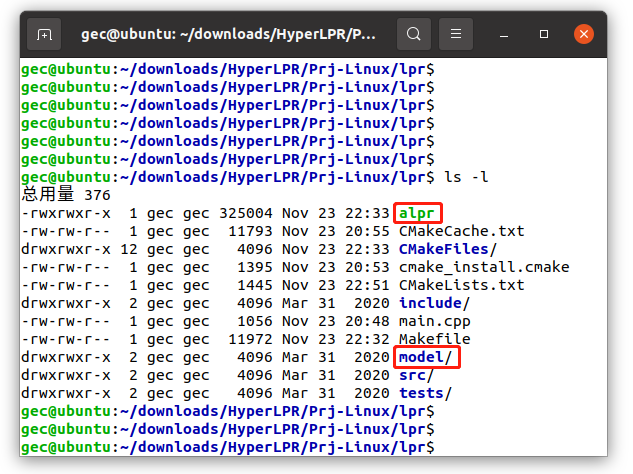
cp /mnt/hgfs/share/day07/main.cpp .

第五步，在 HyPerLPR/Prj-Linux/lpr 下执行：

gec@ubuntu:~/HyperLPR/Prj-Linux/lpr$ cmake .（注意有个.）

gec@ubuntu:~/HyperLPR/Prj-Linux/lpr$ make

第六步，编译完成后，将编译产生的如下文件，放到开发板：

  
HyperLPR编译结果

说明：

* alpr 就是用来进行识别车牌程序
* model/ 是 alpr 正常运行所依赖的数据模型



#将它们全部放到开发板同一目录下，

#将opencv的所有库文件上传到开发板的/lib



如果上传失败，可能是开发板内存已满，可以删除无用的文件

#百度下载一张车牌图片，测试开发板本地车牌识别：

[root@GEC6818 ~]# ./alpr car.jpg

检测到车牌:贵A61000，确信率:0.960277

[root@GEC6818 ~]#



**4. 代码分析**

测试用例的代码如下：

**#include "include/Pipeline.h"**

**#include <sys/types.h>**

**#include <sys/stat.h>**

**#include <unistd.h>**

**#include <fcntl.h>**

**#include <time.h>**

**using** **namespace** std;

**int** **main**(**int** argc,**char** \*\*argv)

{

**if**(argc != 2)

{

printf("请指定一张图片\n");

exit(0);

}

*// 加载数据模型*

pr::PipelinePR **prc**("model/cascade.xml",

"model/HorizonalFinemapping.prototxt",

"model/HorizonalFinemapping.caffemodel",

"model/Segmentation.prototxt",

"model/Segmentation.caffemodel",

"model/CharacterRecognization.prototxt",

"model/CharacterRecognization.caffemodel",

"model/SegmenationFree-Inception.prototxt",

"model/SegmenationFree-Inception.caffemodel");

string pn;

*// 读取一张图片（支持BMP、JPG、PNG等）*

cv::Mat image = cv::imread(argv[1]);

std::vector<pr::PlateInfo> res;

*// 尝试识别图片中的车牌信息*

*// 若图片中不含车牌信息，接口会抛出异常*

**try**{

res = prc.RunPiplineAsImage(image,pr::SEGMENTATION\_FREE\_METHOD);

}

**catch**(...)

{

cout << "未检测到车牌" << endl;

**return** 0;

}

**for**(**auto** st:res)

{

pn = st.getPlateName();

**if**(pn.length() == 9)

{

cout << "检测到车牌: " << pn.data();

cout << "，确信率: " << st.confidence << endl;

}

}

**return** 0;

}

time：输入事件发生的时间

type：数据类型

EV\_SYN 事件分隔

EV\_KEY 描述键盘

EV\_REL 相对位移

EV\_ABS 绝对位移

code：数据类型的进一步描述

value：值

获取触摸屏坐标：

1、打开触摸屏的设备文件

/dev/input/event0

2、按照给定的格式去循环读取数据

3、根据条件进行数据筛选