浙江大学

科研和工程中的 C++编程开发报告

小组成员：高涛 李逸婷 毛晨炀 徐超颖 徐可添

目录

摘要

课题设计

目的：主要用于为后续使用 Deep Learning 训练交通标志检测器提供更多的训练样本；也可用于其他视频中物体（如人脸）的追踪。
输入：一段含有交通标志的视频，交通标志出现时所在的帧与位置信息
输出：该交通标识所出现的所有帧和位置信息、该交通标志在这些帧的截图

开发环境

操作系统：Ubuntu Desktop 14.04
集成开发环境：Qt Creator 3.0.1 (GCC 4.8.2)
依赖库：Qt 4, Boost 1.54, OpenCV 2.4, Eigen 2
工程管理：CMake 2.8
版本控制：GitHub

编译运行

1. 复制项目到本地：
   git clone
   https://github.com/Zhejiang-University-GKC/traffic-sign.git
2. cd traffic-sign
3. 可以在外部编译，先新建一个文件夹：mkdir build
4. 拷贝配置文件：cp config/* build/
5. cd build
6. 如果需要保存追踪结果的截图，需要手动建立 image 文件夹：mkdir image
7. 编译：cmake .. && make
8. 运行：./main
9. 点击 Video 选择视频文件
10. 点击 Run 选择需要追踪物体的初始信息并运行

说明：由于依赖库的安装目录不同，可能需要在编译前修改 cmakelists.txt

详细设计

程序结构

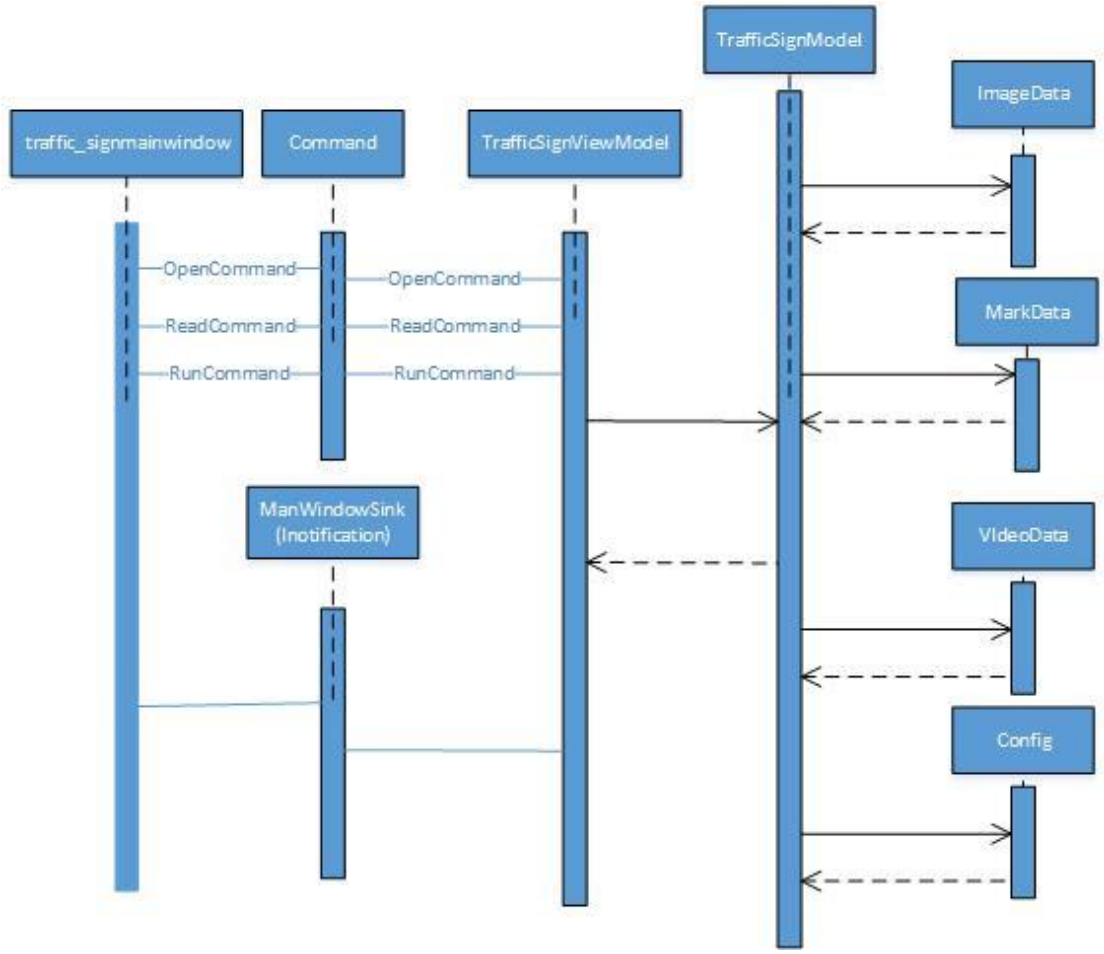

图 1 类关系图

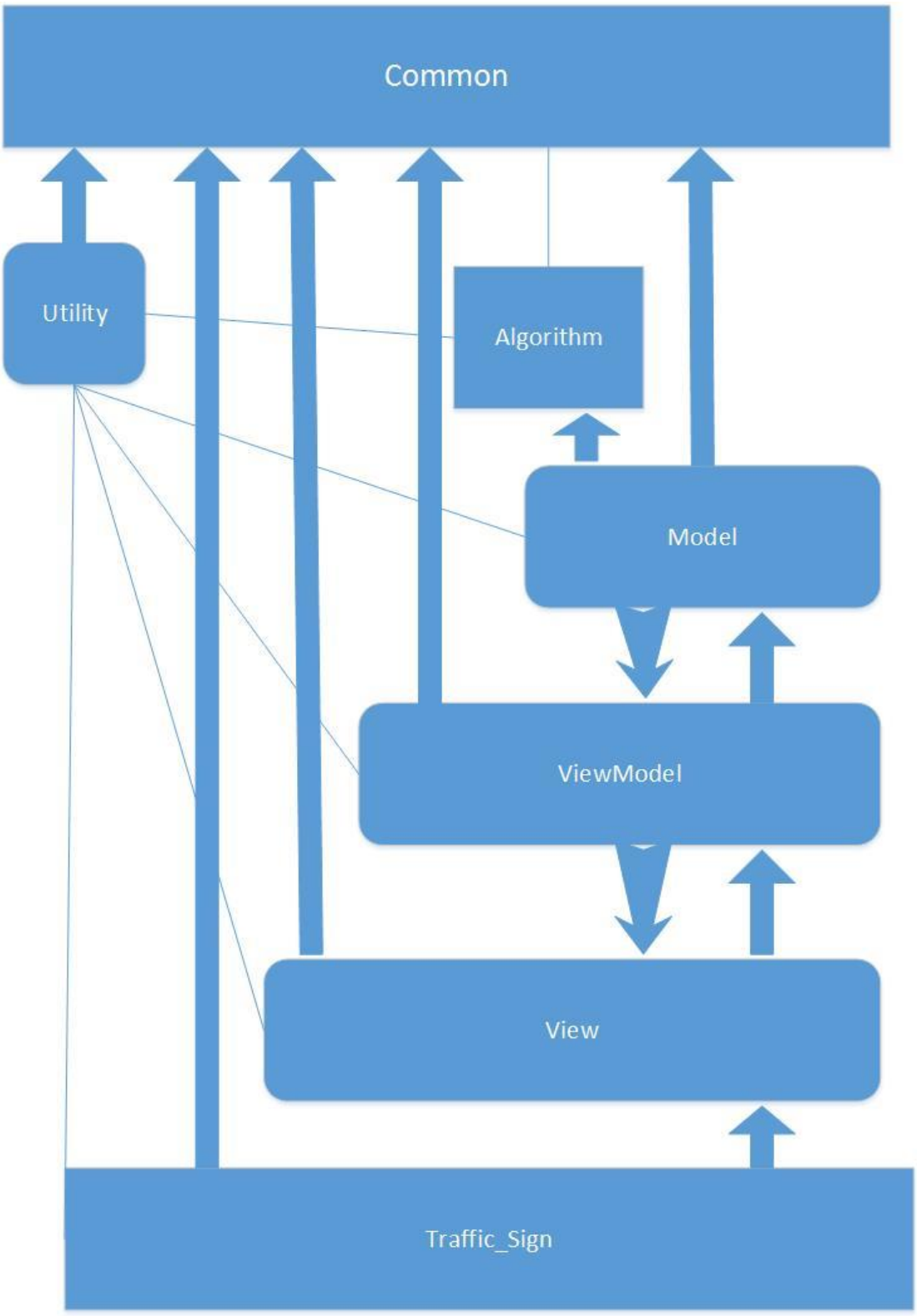
Common
Utility
Algorithm
Model
ViewModel
View
Traffic_Sign

图 2 层次结构图

主要实现

Algorithm

1. Tracker.h

1.1 Tracker 类

执行根据所给配置文件分析跟踪目标标志，记录该标志所在的长方形区域功能。

成员变量：

Tracker::const Config& m_config

保存配置文件信息的变量 m_config

Tracker::bool m_initialised

记录是否被初始化过的标志变量 m_initialised，1 表示已初始化，0 表示未初始化过

Tracker::std::vector<Features*> m_features

Vector 中保存了特征信息 Feature，容器名是 m_feature

Tracker::std::vector<Kernel*> m_kernels;

算法中所使用的核的记录容器 m_kernels

Tracker::LaRank* m_pLearner;

采样信息，以供给下次分析提供样本信息，让机器学习

Tracker::FloatRect m_bb;

记录跟踪分析出的目标标志所在的矩形区域信息

成员函数：

Tracker::Tracker(const Config& conf);

以配置文件作为构造函数的参数，并进行构造初始化

Tracker::~Tracker();

析构函数

void Tracker::Initialise(const cv::Mat& frame, FloatRect bb);

初始化每一帧中标志所在矩形区域信息

void Tracker::Reset();

复位 Tracker，清空信息

void Tracker::Track(const cv::Mat& frame);

跟踪每一帧中的标志

inline const FloatRect& Tracker::GetBB() const { return m_bb; }

返回标志所在矩形区域的信息

inline bool Tracker::IsInitialised() const { return m_initialised; }

返回 Tracker 是否被初始化

void Tracker::UpdateLearner(const ImageRep& image)

升级 Tracker 中的信息，为下次更高效追踪提供数据和样本

Common

1. ICommand.hpp

1.1 ICommandParam 类（纯虚类）

这个类主要负责 ICommand 中的参数的处理。

成员函数：

ICommandParam() {}

构造函数

virtual ~ICommandParam() {}

析构函数

virtual int GetParamNum()=0;

获取参数个数

virtual void* GetParam(int index)=0;

参数 index 表示第几个参数，这个函数负责获取第 index 个参数的内容

1.2 ICommand 类（纯虚类）

主要负责根据 view 层中触发的事件（比如按了某个按钮），进行对应的命令操作

成员函数：

ICommand() {}

构造函数

virtual ~ICommand() {}

析构函数

virtual void Execute(const boost::shared_ptr<ICommandParam> &param)=0;

执行命令函数，参数传入由 ICommandParam 处理后过的对应的参数数据

2. INotification.hpp

2.1 INotification 类（纯虚类）

成员函数：

INotification(){}

构造函数

virtual ~INotification(){}

析构函数

virtual void OnPropertyChanged(const std::string &property)=0;

这个函数负责当 Model 层数据取出处理好之后给予 View 层一个显示的信号

3.OpenCommandParam.hpp

3.1 OpenCommandParam 类，继承自 ICommandParam 类

负责记录 Open 这个命令的参数信息

成员变量：

std::string video_filename

video_filename 变量记录录像名

std::string mark_filename;

mark_filename 变量记录标记好的信息

成员函数：

OpenCommandParam() {}

构造函数

~OpenCommandParam() {}

析构函数

int GetParamNum() { return 1; }

获取参数数量

void SetParam(const std::string &s) { video_filename=s; }

获取参数（即外部获取的录像路径名）传入类内部交给 video_filename 保存

void* GetParam(int index)

获取第 index 个参数的内容

Model

1.TrafficSignModel .hpp

1.1 TrafficSignModel 类

主要用于 Model 层中负责与数据交换，包括保存、提取和修改数据。

成员变量：

boost::shared_ptr<ImageData> sp_image

智能指针 sp_image 来记录从保存的数据中取出来的图片信息

boost::shared_ptr<VideoData> sp_video

智能指针 sp_video 来记录录像信息

boost::shared_ptr<MarkData> sp_sMark, sp_tMark

智能指针 sp_Mark 来记录图像中的标记信息

boost::shared_ptr<Config> sp_config

智能指针 sp_config 来记录配置文件的信息

std::string videoName

记录录像名的变量 videoName

成员函数：

TrafficSignModel()

构造函数

~TrafficSignModel()

析构函数

void OpenVideo(const std::string &filename)

打开指定的录像

void SetImage(const cv::Mat &image)

将传入的图像信息复制保存到数据中

void ReadMark(const std::string &filename)

读取录像中的标记信息

boost::shared_ptr<ImageData> GetImage()

读取保存数据中的图像信息，返回的是指向 Mat 图像的智能指针

boost::shared_ptr<Config> GetConfig()

读取配置文件信息，返回的是指向配置文件信息的智能指针

boost::shared_ptr<MarkData> GetMark(int)

获取标记信息，返回的是指向标记信息的智能指针

cv::Mat *GetFrame(int frameInd)

获取录像的某一帧

const std::string &GetVideoName()

获取录像的名字

Utility

1.ImageData.hpp

1.1 ImageData 类

主要负责存放读取修改图像信息

成员变量：

cv::Mat image

存放图像信息

成员函数：

cv::Mat * GetImage()

读取图像信息，以指针形式返回

void SetImage(const cv::Mat & image)

读入图像信息，并复制信息保存入对象中

2 VideoDate.hpp

2.1 VideoDate 类

主要功能是视频的读取存储和视频帧画面的提取输出。

成员变量

cv::VideoCapture cap

用于获取视频对象，以及对视频处理等功能。

cv::Mat frame

用于储存视频的某一帧的图像。

int currentFrame, totFrame

当前帧数和视频总帧数。

成员函数

void OpenVideo(const std::string &)

利用 VideoCapture 的 open()成员函数打开指定目录的视频文件。

cv::Mat *GetFrame()，cv::Mat *GetFrame(int)

获取指定帧（未指定则为当前帧）画面并输出。

ViewModel

1. TraficSignViewModel 类

成员变量：

boost::shared_ptr<ICommand> sp_OpenCommand

打开视频命令指针

boost::shared_ptr<ICommand> sp_RunCommand

捕获交通标志命令指针

boost::shared_ptr<ICommand> sp_ReadCommand

读取标记好的视频图像命令指针

boost::shared_ptr<TrafficSignModel> sp_Model

Model 层对象指针

boost::shared_ptr<INotification> event

事件标记指针，用来通知 View 层各种数据是否已准备好

boost::shared_ptr<QImage> sp_image_view, sp_mark_view

用于储存原视频图像和标记好的视频图像

成员函数

TrafficSignViewModel()，

构造函数，用于各指针的初始化。

boost::shared_ptr<ICommand> GetOpenCommand()

发出打开视频命令信号。

boost::shared_ptr<ICommand> GetRunCommand()

发出捕获交通标志命令信号。

boost::shared_ptr<ICommand> GetReadCommand()

发出读取标记好的图像命令信号。

boost::shared_ptr<QImage> GetImagePtr(const int &)

将图像指针（原图或标记好的图）发送到 View 层。

void SetEvent(const boost::shared_ptr<INotification> &e)

设定事件完成标记。

void SetModel(boost::shared_ptr<TrafficSignModel> &model)

读取 Model 层对象。

void OpenVideo(const std::string &filename)

向 Model 层发出打开视频命令。

void TrackSign(const std::string &filename)

向 Model 层发出捕获交通标志命令。

void ReadMark(const std::string &filename)

向 Model 层发出读取已标记好的图像命令。

2. OpenCommand 类（ICommand 的子类，ReadCommmad/RunCommand 类似）

主要用于存储打开视频这个命令。令，对 Model 层发出指令请求。

成员变量

TrafficSignViewModel *m_ViewModel

ViewModel 层对象。

成员函数

OpenCommand(TrafficSignViewModel *m)

构造函数。

~OpenCommand()

析构函数。

void Execute(const boost::shared_ptr<ICommandParam> &param)

向 ViewModel 层传入打开视频命令。

View

1. Traffic_signMainWindow 类

主要用于对界面鼠标点击事件的捕获和图像的显示。

成员变量

boost::shared_ptr<ICommand> sp_OpenCommand

打开视频命令指针

boost::shared_ptr<ICommand> sp_RunCommand

捕获交通标志命令指针

boost::shared_ptr<ICommand> sp_ReadCommand

读取标记好的视频图像命令指针

boost::shared_ptr<QImage> sp_image_view, sp_mark_view

用于储存原视频图像和标记好的视频图像

boost::shared_ptr<INotification> sp_Event

事件标记指针，用来通知 View 层各种数据是否已准备好

成员函数

void on_VideoButton_clicked();

void on_ExitButton_clicked();

void on_RunButton_clicked();

用于捕获和触发鼠标单击事件

QTimer *running_timer

QT 中断

boost::shared_ptr<QImage> GetImage(const int &)

获取要显示的图像

boost::shared_ptr<INotification> GetEvent()

从 ViewModel 获取信息准备情况的通知。

explicit Traffic_signMainWindow(QWidget *parent = 0)

构造函数。

~Traffic_signMainWindow()

析构函数。

void SetOpenCommand(const boost::shared_ptr<ICommand> &ptr);

void SetReadCommand(const boost::shared_ptr<ICommand> &ptr);

void SetRunCommand(const boost::shared_ptr<ICommand> &ptr);

向 ViewModel 层发送打开视频/开始捕获/读取结果命令

void SetImage(const boost::shared_ptr<QImage> &ptr);

void SetMark(const boost::shared_ptr<QImage> &ptr);

储存图像信息。

QLabel *GetLabel(const std::string &)

获取当前图像的性质（原图或标志图）。

2. MainWindowSink 类（INotification 的子类）

主要用于当前显示属性更改的通知。

成员变量

Traffic_signMainWindow *p_Window

主窗口对象指针。

成员函数

MainWindowSink(Traffic_signMainWindow *p):p_Window(p)

构造函数初始化指针。

void OnPropertyChanged(const std::string &property)

标记当前播放为视频或标志或已经捕获完毕。

开发日志

7 月 5 日：确定课题，确定开发环境和依赖库，大家一起在自己的计算机上配置开发环境。遇到的主要问题：在虚拟机中安装 Ubuntu 14.04 Desktop 比较慢；Windows 8 系统下的 VMWare WorkStation 不太稳定，安装过程中常常卡住；Mac OS X 系统下的 Parallels Desktop 9 对 Ubuntu 14.04 Desktop 兼容性不好，安装完成后分辨率有问题，只能换用 VMWare Fusion。
李逸婷写了一份 GitHub 使用的简要说明并 push 到我们的 repo 供组员们参考。

7 月 6 日：继续安装和配置开发环境。确定大致分工并且根据自己的任务进行资料的查阅和学习，其中：

高涛负责前端图形界面的绘制和开发，系统架构；
李逸婷负责图像读写、图像信息处理相关工作；
毛晨炀负责标记框的绘制、结果的保存；
徐超颖负责视频处理相关工作；
徐可添负责系统架构和追踪算法相关工作。

遇到的困难有：大家都是第一次接触 MVVM 框架，不能很好理解其设计思想和设计方法，决定由高涛和徐可添重点学习，由高涛负责 View 层、ViewModel 层，徐可添负责 ViewModel 层、Model 层的方式来搭建框架。
将课题信息和分工等 push 到 GitHub 上。

7 月 7 日：环境基本都配置好了。高涛和徐可添参考第一组的程序框架，进行大体框架的搭建，但对于 INotification 类的作用不能很好理解。小组讨论决定，先完成一个具有“按下一个按钮给出一个弹出窗口供用户选择路径，然后读入一个视频并显示其第一帧”这样基本功能的程序，作为第一轮迭代的目标。
大家都在自己的计算机上完成了利用 OpenCV 读入一张图片并显示的小程序，确保开发环境正常。

7 月 8 日：继续前一天的工作。徐超颖、毛晨炀的系统在更新后驱动程序坏了，无法联网，只能重新从头安装。组长学习了 Qt Creator 的使用，并和组员分享。

7 月 9 日：白天组长去社会实践了…
晚上完成了大体框架的搭建。李逸婷、徐超颖学习并实现了几个利用 OpenCV 进行图像、视频的输入输出的类。毛晨炀学习了利用 OpenCV 绘制标记框、对框定区域保存的方法。高涛完成了前端界面，徐可添将这些类和界面整合进之前完成的 MVVM 框架。能够基本达到预期效果，但应用程序第一次读入的视频无法正常显示第一帧，之后读入的视频都能正常显示。

7 月 10 日：我们听取了袁老师对第一个程序的指导意见，明白了 View 和 ViewModel 之间的正确关系，明白了之前贪图方便将 ViewModel 全部暴露给 View 的做法的缺陷，理解了 INotification 的工作机制，并对前一天完成的程序做了相关修改。第一轮迭代完成。
小组成员参考网上《CMake 实践》一文，集体系统学习了 CMake 的工作原理。

7 月 11 日：修正原来程序中的一些 bug。
我们使用的追踪算法基于 ICCV 2011 的“Struck: Structured Output Tracking with Kernels”一文，原作者提供的源代码使用 Makefile 来管理工程，李逸婷学习了 Makefile 并尝试编译链接，但是静态库链接始终有问题，最终还是通过新写一个 cmakelists.txt 解决了问题。我们用作者提供的样例尝试了追踪程序。
组长开始结合论文阅读 Struck 程序的源代码，并尝试将其整合进我们的程序，作为第二轮迭代的目标。

7 月 12 日：因机房有考试停课一天。
为实现对视频的持续处理，组长学习 Qt 中断机制，成功将 Struck 程序整合进我们的程序并完成相关数据传递、消息传递的逻辑。高涛对界面作出相应的修改，完善 View 层逻辑。完成第二轮迭代。其他组员对程序功能进行测试，提交了一些 bug，其中最大的问题是：在配置文件中设置视频缩放可能会引起崩溃。

7 月 13 日：向袁老师展示程序功能。
修正 bug。增加了独立显示标定窗口内容的功能，完善了页面布局。完成第三轮迭代。
小组成员共同撰写实验报告。其中：

徐可添负责摘要和排版
高涛负责设计结构和整合文档
徐超颖、李逸婷负责主要实现
毛晨炀负责程序运行
开发日志为共同完成。

程序运行

程序使用说明

编译程序可以参考摘要中“编译运行”一小节。
在开始运行前，请确保可执行文件所在文件夹中有配置文件 config.txt。该文件的样例可以在项目的 config 文件夹中找到，支持对视频进行缩放、选择追踪算法使用的图像特征等。
运行前，先点击 video 按钮选择一个视频，然后点击 run 按钮导入被追踪对象的信息并开始追踪。追踪对象的信息满足以下格式：

```
start_frame  start_lx  start_ly  start_rx  start_ry
end_frame  end_lx  end_ly  end_rx  end_ry
```

分别表示：初始帧、待追踪区域左上角坐标 x 和 y、待追踪区域右下角坐标 x 和 y、追踪结束帧、结束时待追踪区域左上角坐标 x 和 y、右下角坐标 x 和 y。

程序运行示例

以 33489.mp4 为例，展示运行结果：

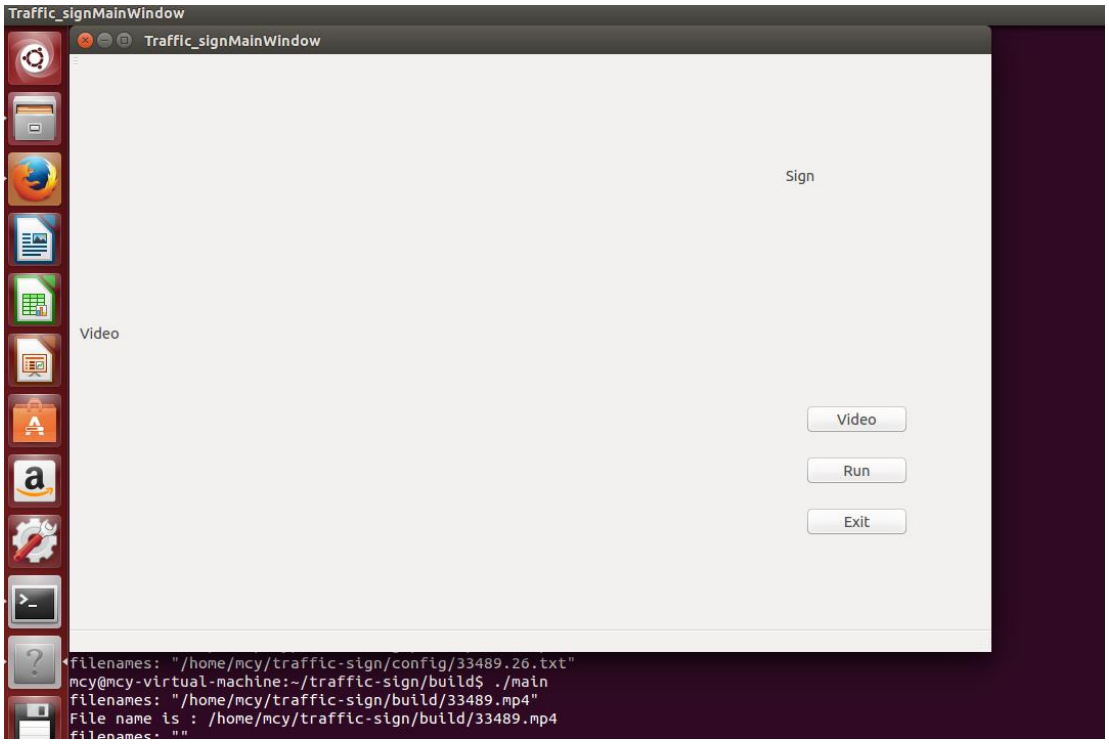

图 3 主界面

载入视频，按 video 按钮：

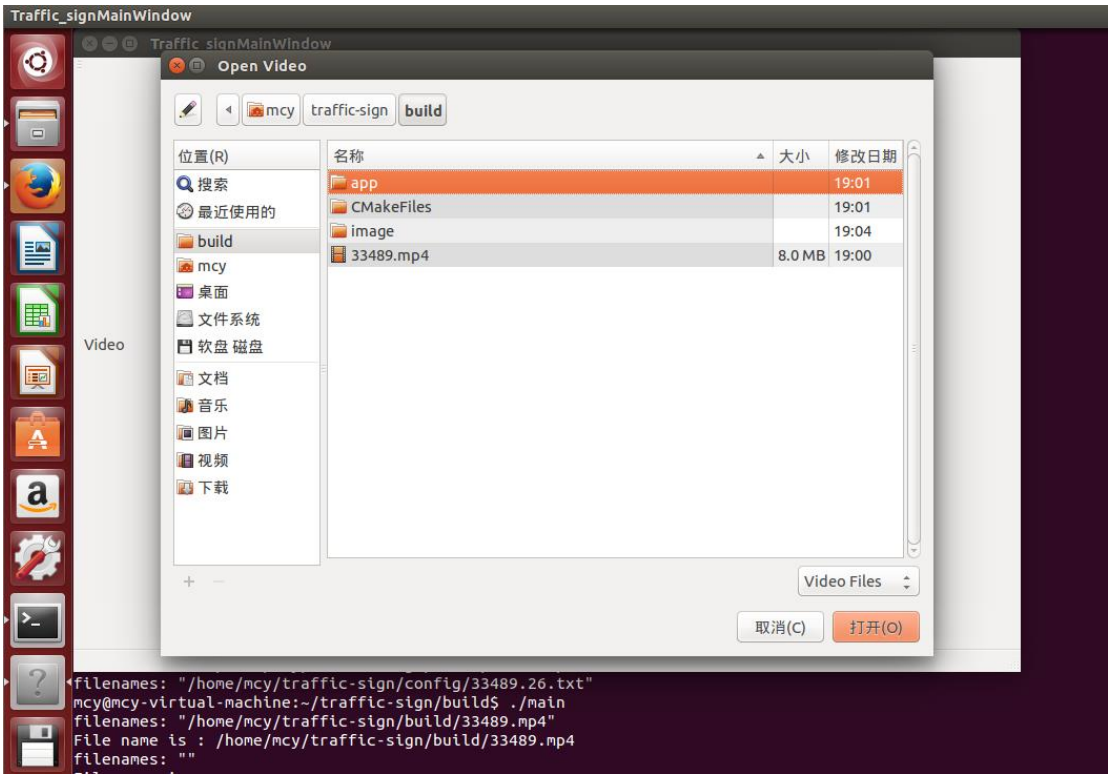

图 4

载入 33489.mp4 视频文件

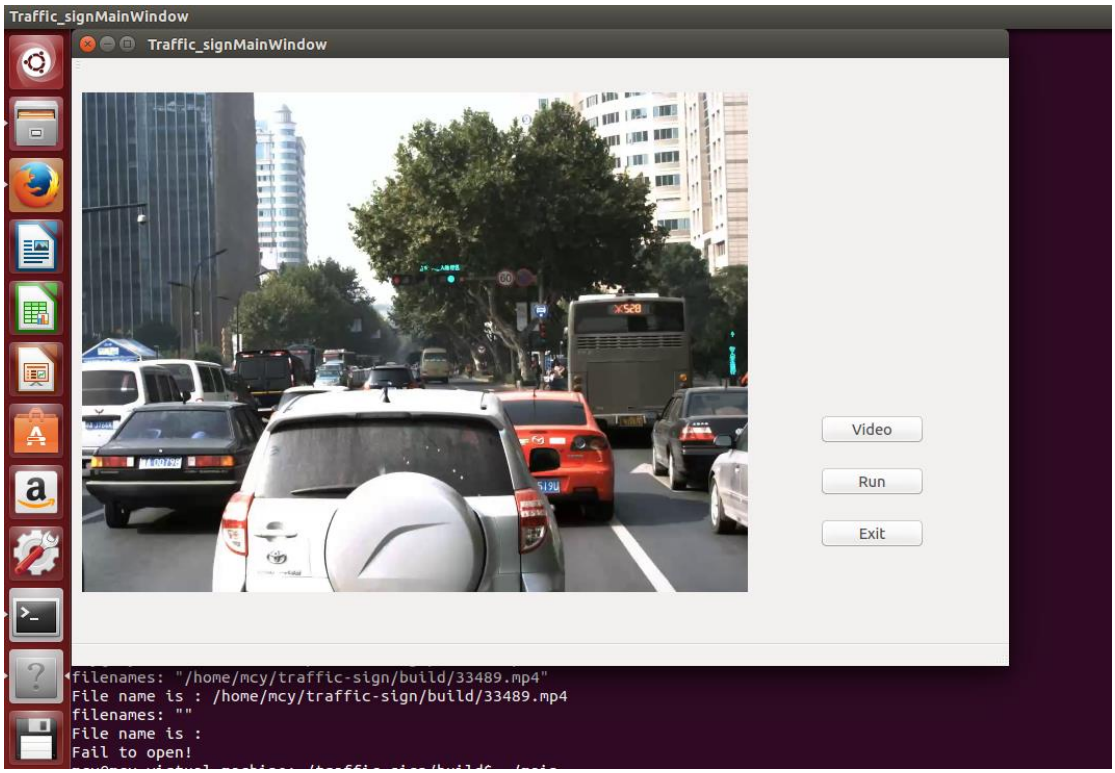

图 5

载入配置文件，按 run 按钮：

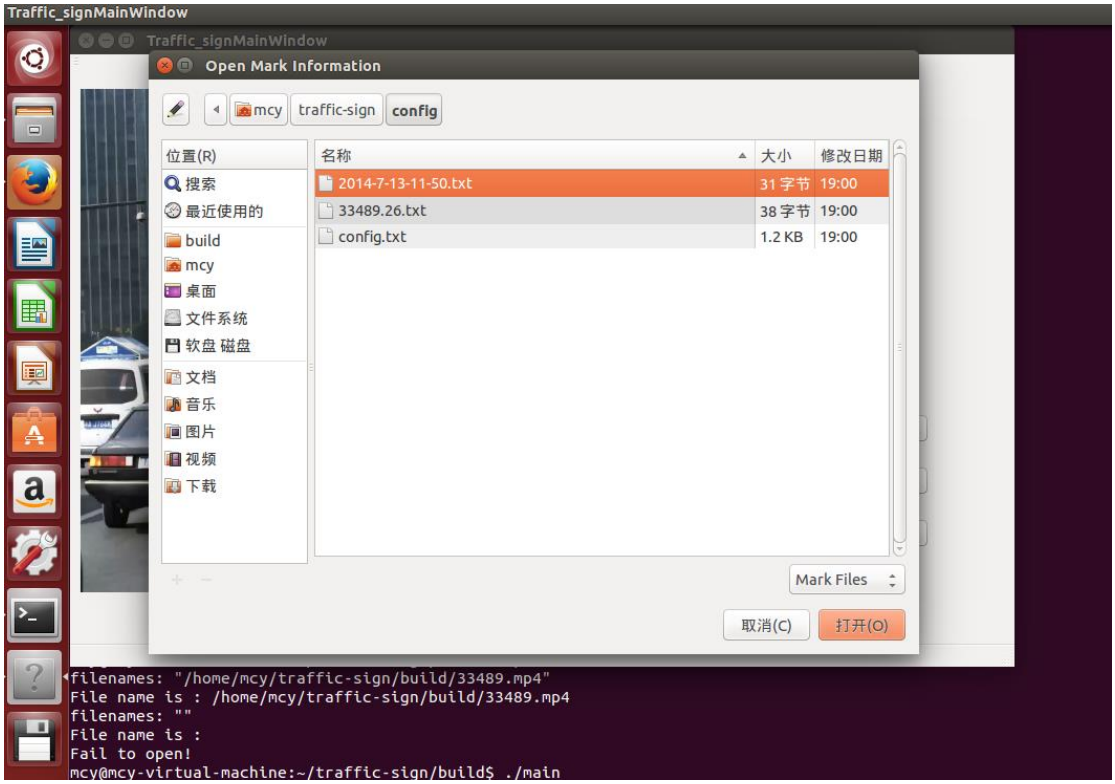

图 6

载入 333489.txt 后运行：

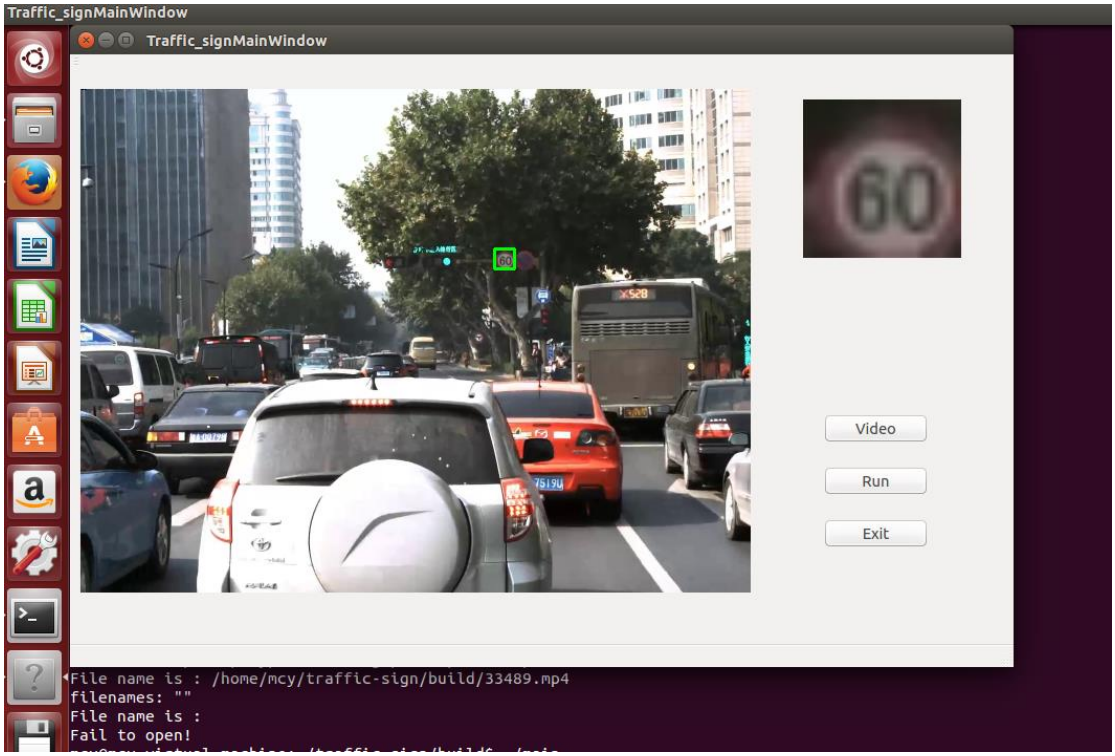

图 7

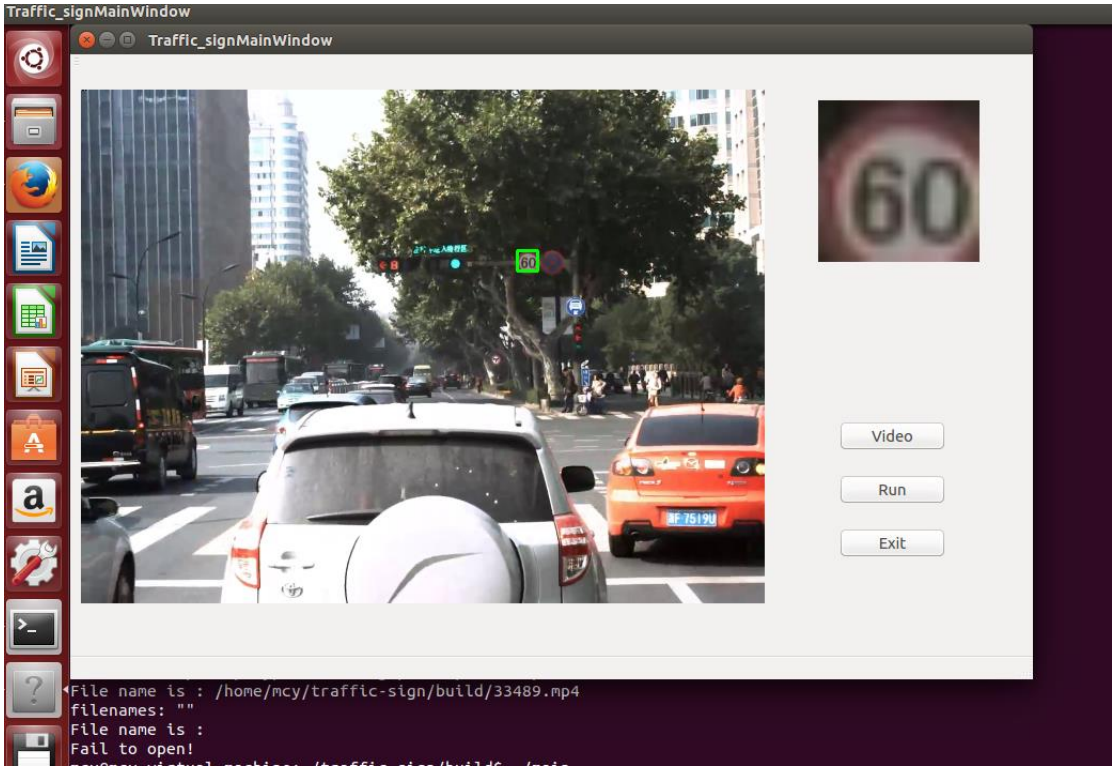

图 8

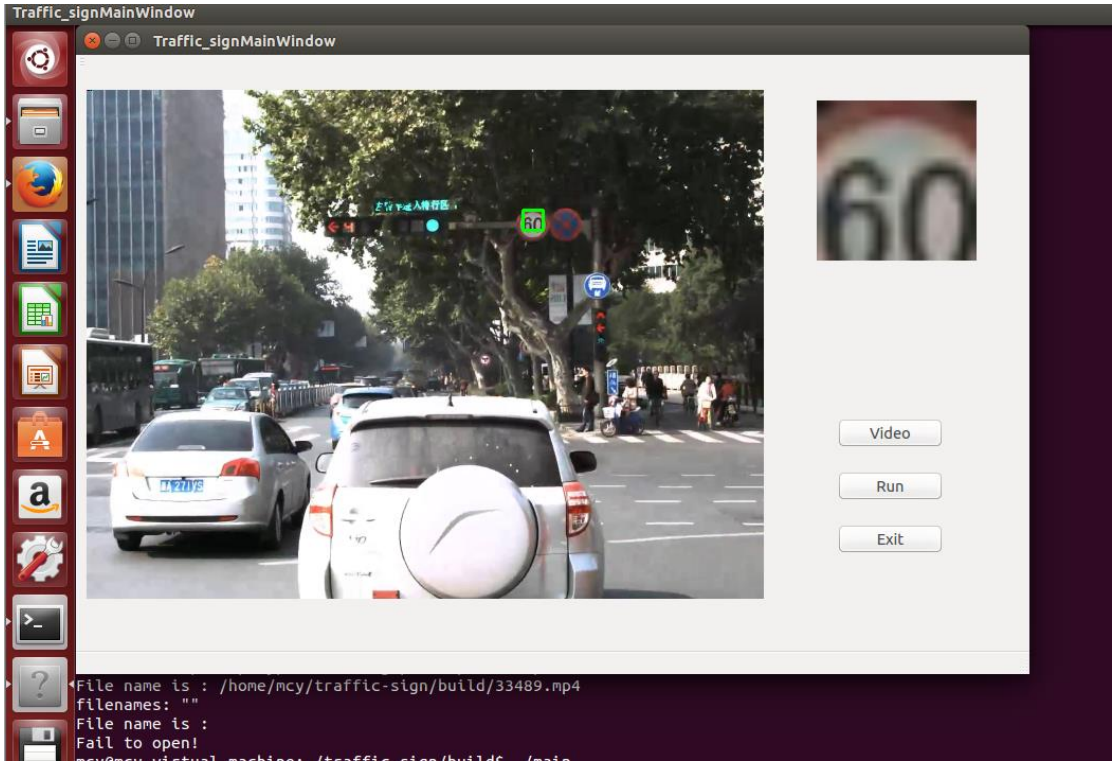

图 9

按 Exit 按钮退出：

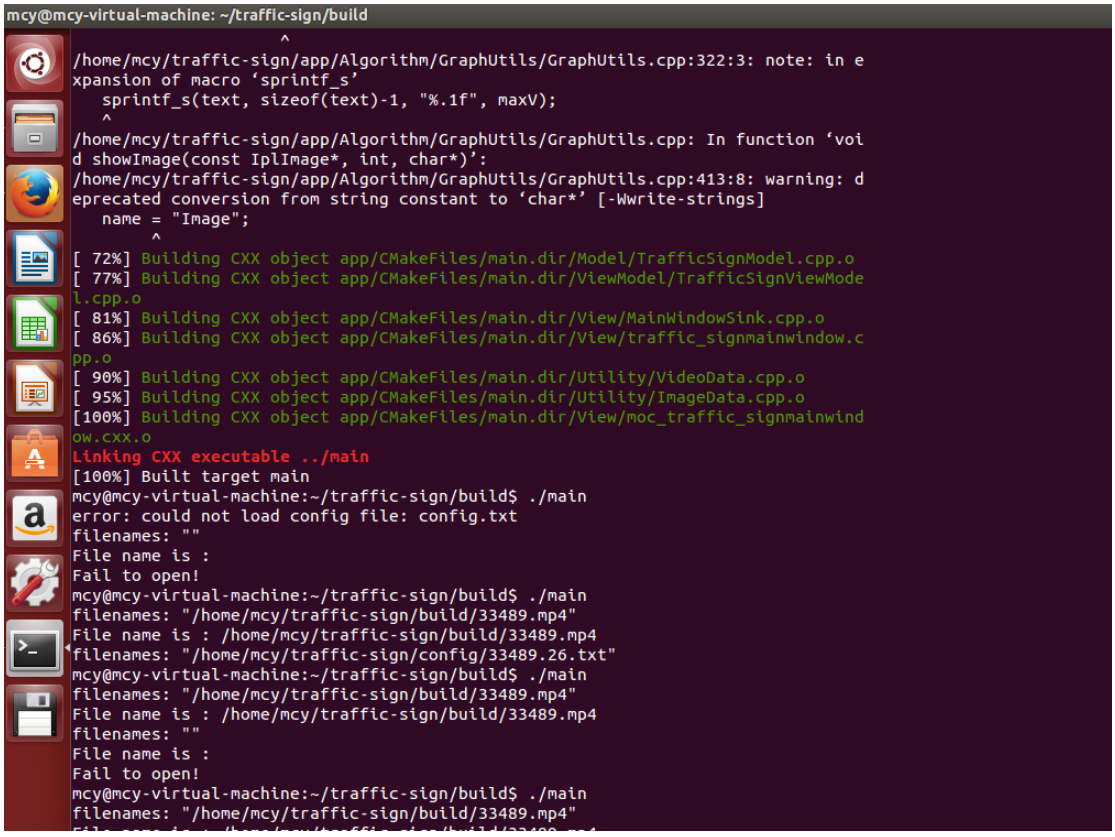

图 10

在 image 文件中记录下了交通标志的图片文件：

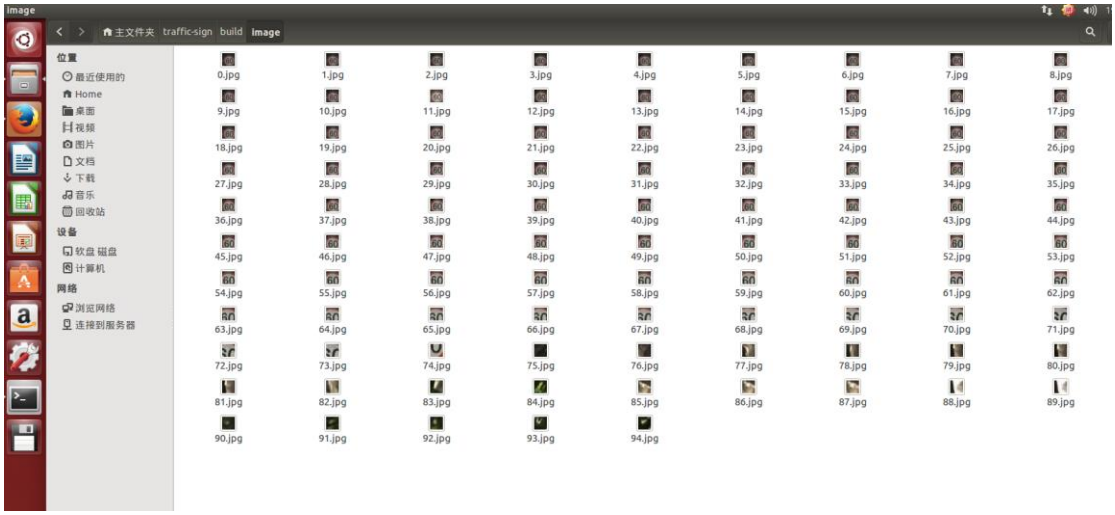

图 11

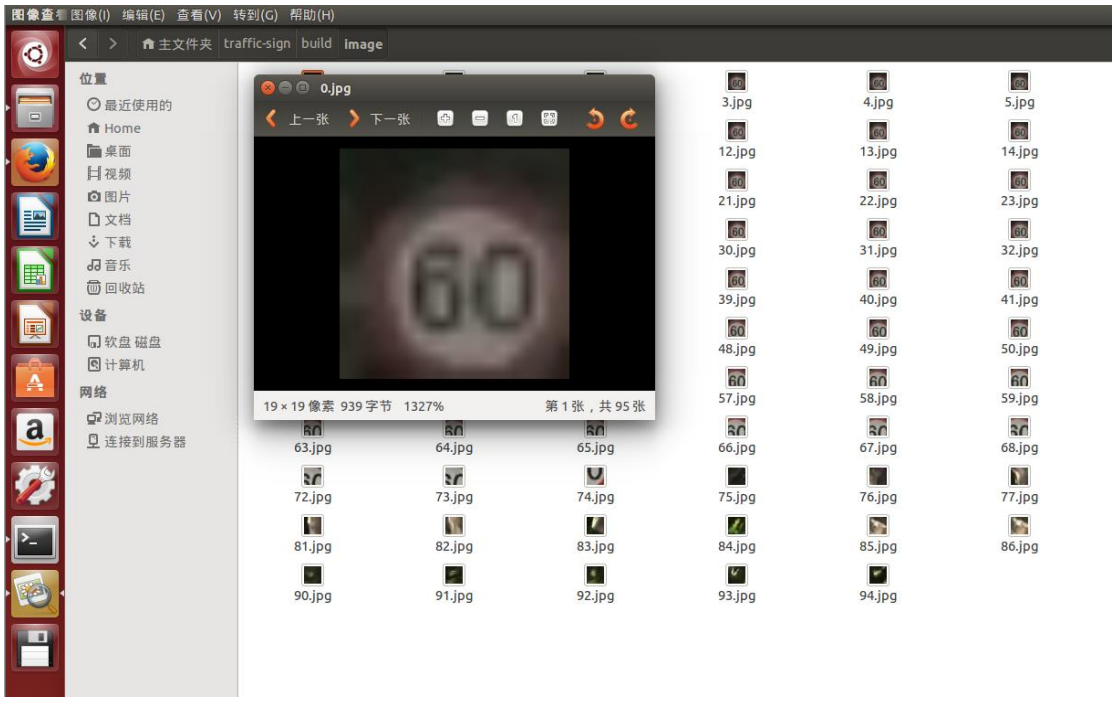

图 12

开发总结

毛晨炀：

这次短学期中学到了许多在科研和工程中 C++的写法，感觉在程序开始写之前的架构设计是整个工程的重点，迭代设计的安排将整个设计耦合度降低，保证了每次得到的迭代成果都是可以运行的，大大保证了整个工程的编写效率，这种设计方法很让人受益。组长在这次短学期的工作中起到了关键性的作用，带领团队将大程完成，我从团队成员中学到很多，感谢每一个成员的付出。

李逸婷（3120101284）：

一开始上来课程与我想象地非常不一样，也没有想到是直接在十天这么短的时间内实现一个项目。果然验证了 DDL 是第一生产力，在短短这么时间内，我们小组一起学习了 QT，学习了 Boost，学习了 OpenCV，都是非常有用的一些东西。同时也了解了一个软件开发的模式和过程，MVVM 模式很好地解除了前端和后端的耦合度，让我们可以更好地分别负责一个部分的开发，并且方便了调试。小组一起配置环境，一起讨论各部分的具体实现方法，不断地推进这个项目，总而言之是很不错的经验~

徐超颖：

这是第一次较为完整地完成一个工程课题，过程中学习到很多之前都没听说或是接触过的东西，比如处理图像和视频的 OpenCV 库，还有 MVVM 这种设计模式。这种新的设计模式极大的降低了耦合度，让每个人的开发都能同步进行。通过几次迭代的方式，保证了每次迭代的正确率。这次工程，整个团队工作让我体会到工程上程序设计的感觉，收获很多。另外要感谢所有组员的付出和组长的指导。

高涛：

这次的程序和之前我编写过的都有很大的不同。主要是体现在对于结构的解耦合和结构层次的概念。在这方面的理解上有了一个比较新的认识，对于如何团队开发程序也有了很大的改观。MVVM 模式很好地解开了耦合也让小组成员可以各自独立地开发自己的对象，不需要一起调试编译，极大地增加了效率。也是第一次用 github 来联合开发程序感受到了 github 的强大。由于是在 linux 上开发，对于操作系统的不太熟悉，造成了一点点小小的困难，但最后都克服了。这次经验收获颇丰。

徐可添：

这次短学期的课程体验和大一相比截然不同。在如此短的时间内要完成这样一个课题起初让我感觉压力巨大，MVVM 设计框架原来更是听都没有听过。不过最终我们还是做到了，我想这本身就是一个胜利了！

在短短十多天里，和组员一起学习、开发、调试、再学习、再开发、再调适的过程给我带来的收获是巨大的。所谓“纸上得来终觉浅，绝知此事要躬行”，在这样一段学习和实践融合的日子里，一方面我对 Qt、对 OpenCV、对 Boost、对 CMake 有了一个接触和体验；另一方面我也领会到了 MVVM 设计框架的精神。当然，更重要的是，这段经历让我意识到了，越是觉得困难的事就越要勇敢去尝试，等到做完了，你才会发现，其实所谓的难事也不过如此。

最后感谢袁老师的悉心指导！感谢组员们辛勤的付出，和相互陪伴的这十几个日日夜夜。

附件（GitHub 地址）

https://github.com/Zhejiang-University-GKC/traffic-sign

2014 年 7 月 14 日