浙江大学

科研和工程中的C++编程实验报告

小组成员：

高涛 李逸婷 毛晨炀 徐超颖 徐可添

目录

[摘要 4](#_Toc393069376)

[题目设计 4](#_Toc393069377)

[设计内容 4](#_Toc393069378)

[应用平台 4](#_Toc393069379)

[详细设计 5](#_Toc393069380)

[程序结构 5](#_Toc393069381)

[主要功能 6](#_Toc393069382)

[函数实现 7](#_Toc393069383)

[Algorithm 7](#_Toc393069384)

[Common 7](#_Toc393069385)

[Model 7](#_Toc393069386)

[Utility 7](#_Toc393069387)

[VideoDate 7](#_Toc393069388)

[ViewModel 7](#_Toc393069389)

[View 7](#_Toc393069390)

[开发日志 7](#_Toc393069391)

[程序调试及运行 7](#_Toc393069392)

[程序运行结果 7](#_Toc393069393)

[程序使用说明 12](#_Toc393069394)

[程序开发总结 13](#_Toc393069395)

[附件（源程序） 13](#_Toc393069396)

[CmakeLists.txt 13](#_Toc393069397)

[mian.cpp 16](#_Toc393069398)

[View 16](#_Toc393069399)

[ui\_traffic\_signmianwindows.h 16](#_Toc393069400)

[traffic\_signmianwindows.h 19](#_Toc393069401)

[traffic\_signmianwindows.cpp 20](#_Toc393069402)

[MainWindowSink.hpp 23](#_Toc393069403)

[MainWindowSink.cpp 23](#_Toc393069404)

[ViewModel 24](#_Toc393069405)

[TrafficSignViewModel.hpp 24](#_Toc393069406)

[TrafficSignViewModel.cpp 25](#_Toc393069407)

[RunCommand.hpp 29](#_Toc393069408)

[ReadCommand.hpp 30](#_Toc393069409)

[OpenCommand.hpp 31](#_Toc393069410)

[Utility 31](#_Toc393069411)

[VideoData.hpp 31](#_Toc393069412)

[VideoData.cpp 32](#_Toc393069413)

[MarkData.hpp 34](#_Toc393069414)

[ImageData.hpp 34](#_Toc393069415)

[ImageData.cpp 34](#_Toc393069416)

[Model 35](#_Toc393069417)

[TrafficSignModel.hpp 35](#_Toc393069418)

[TrafficSignModel.cpp 36](#_Toc393069419)

[Common 38](#_Toc393069420)

[OpenCommandParam.hpp 38](#_Toc393069421)

[INotification.hpp 38](#_Toc393069422)

[ICommand.hpp 39](#_Toc393069423)

# 摘要

## 题目设计

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

## 设计内容

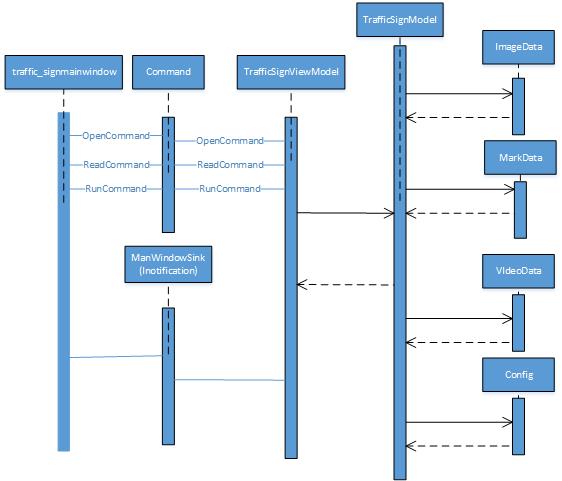
操作系统：Ubuntu Desktop 14.04  
集成开发环境：Qt Creator 3.0.1 (GCC 4.8.2)  
依赖库：Qt 4, Boost 1.54, OpenCV 2.4, Eigen 2  
工程管理：CMake 2.8  
版本控制：GitHub

## 应用平台

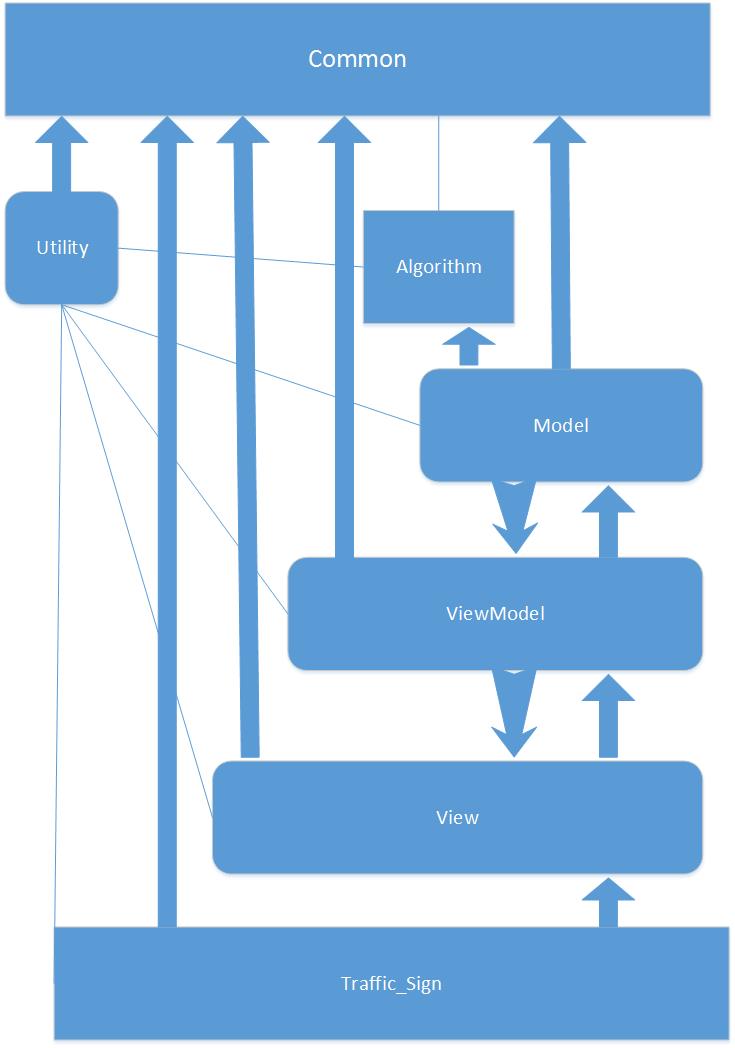
Ubuntu Desktop 14.04

# 详细设计

## 程序结构



图片 2类关系图



图片 1层次结构图

## 主要功能

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。

## 函数实现

### Algorithm

**1.Tracker.h**

1.1Tracker类

执行根据所给配置文件分析跟踪目标标标志，记录该标志所在的长方形区域功能。

1.1.1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

1.1.2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。

1.1.1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

1.2.1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）

2.1.1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**

3.1OpenCommandParam类，继承自ICommandParam 类

负责记录Open这个命令的参数信息

3.1.1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

3.1.2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。

1.1.1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

1.1.2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

1.1.1 成员变量：

cv::Mat image

存放图像信息

1.1.2 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

### VideoDate

主要功能是视频的读取存储和视频帧画面的提取输出。

1.1.1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。

1.1.2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

1.1 TraficSignViewModel

主要功能是接收View层的指2.1.1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

void fireEvent(const std::string &property);

1.1.2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。

1.2 OpenCommand （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。

1.2.1 成员变量

TrafficSignViewModel \*m\_ViewModel，ViewModel层对象。

1.2.2 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

1.1 Traffic\_signMainWindow

主要用于对界面鼠标点击事件的捕获和图像的显示。

1.1.1 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

1.1.2 成员函数

void on\_VideoButton\_clicked()，on\_ExitButton\_clicked()，on\_RunButton\_clicked()，用于捕获鼠标单击事件

QTimer \*running\_timer，QT的计时器

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)，SetReadCommand(const boost::shared\_ptr<ICommand> &ptr)，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)，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）。

1.2 MainWindowSink(INotification的子类)

主要用于当前显示属性更改的通知。

1.2.1成员变量

Traffic\_signMainWindow \*p\_Window，主窗口对象指针。

1.2.2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

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。

# 程序调试及运行

## 程序运行结果

运行结果：

编译

cd traffic-sign

mkdir build

cd build

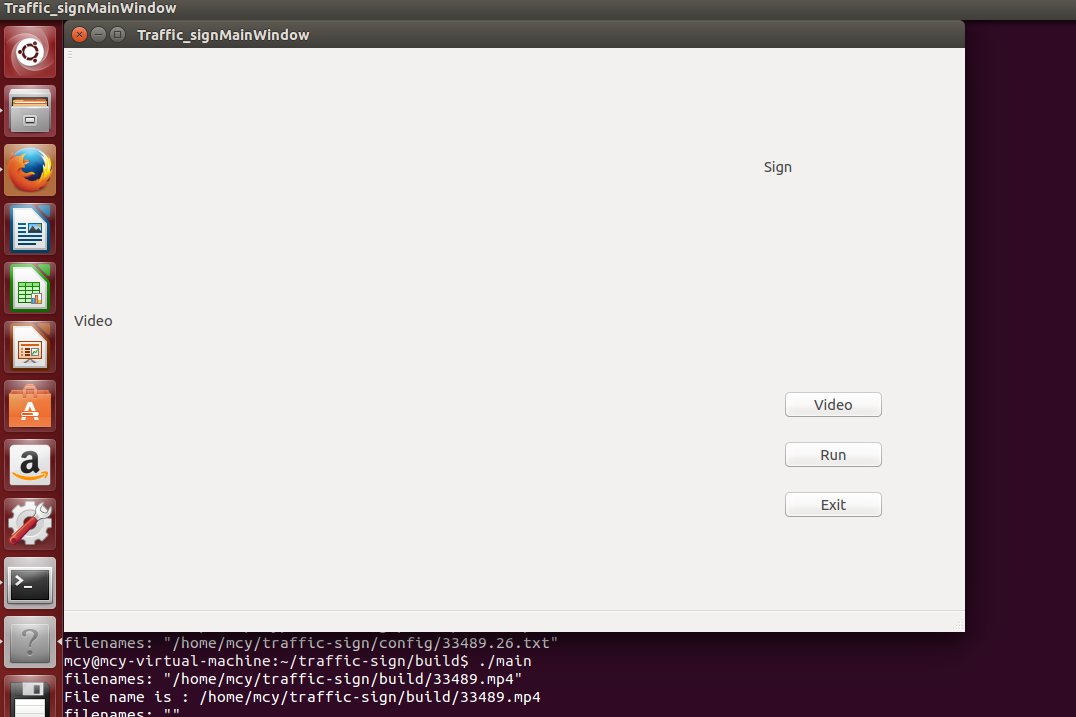
cmake ..&& make

运行

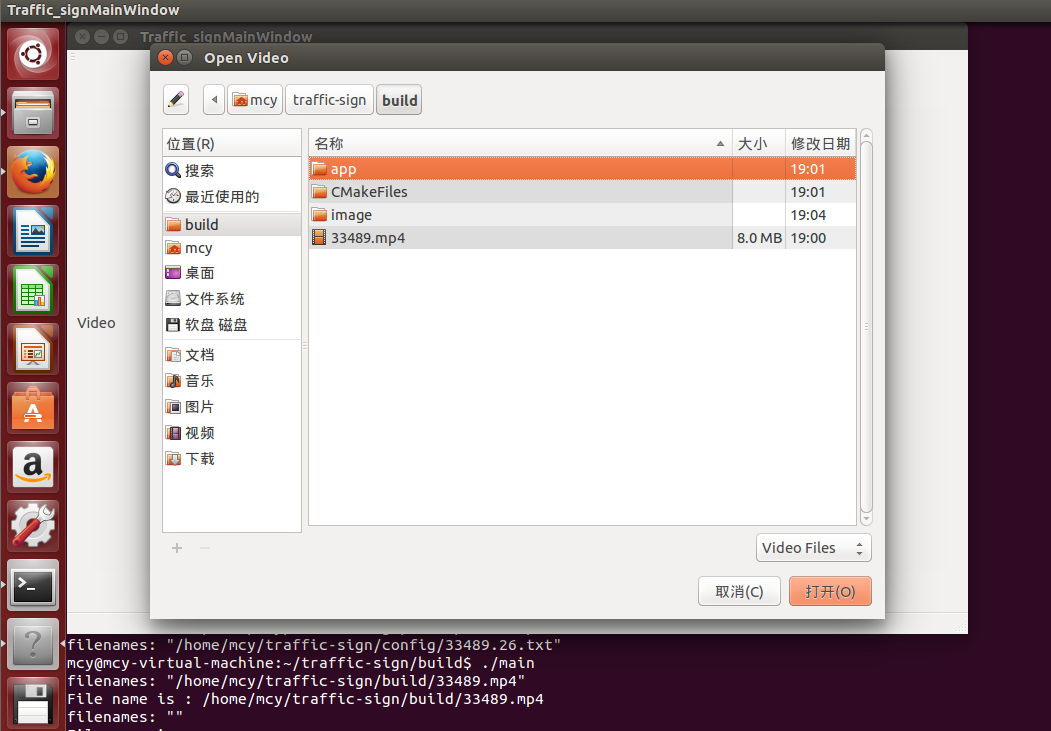
./main

以33489.mp4为例，展示运行结果：

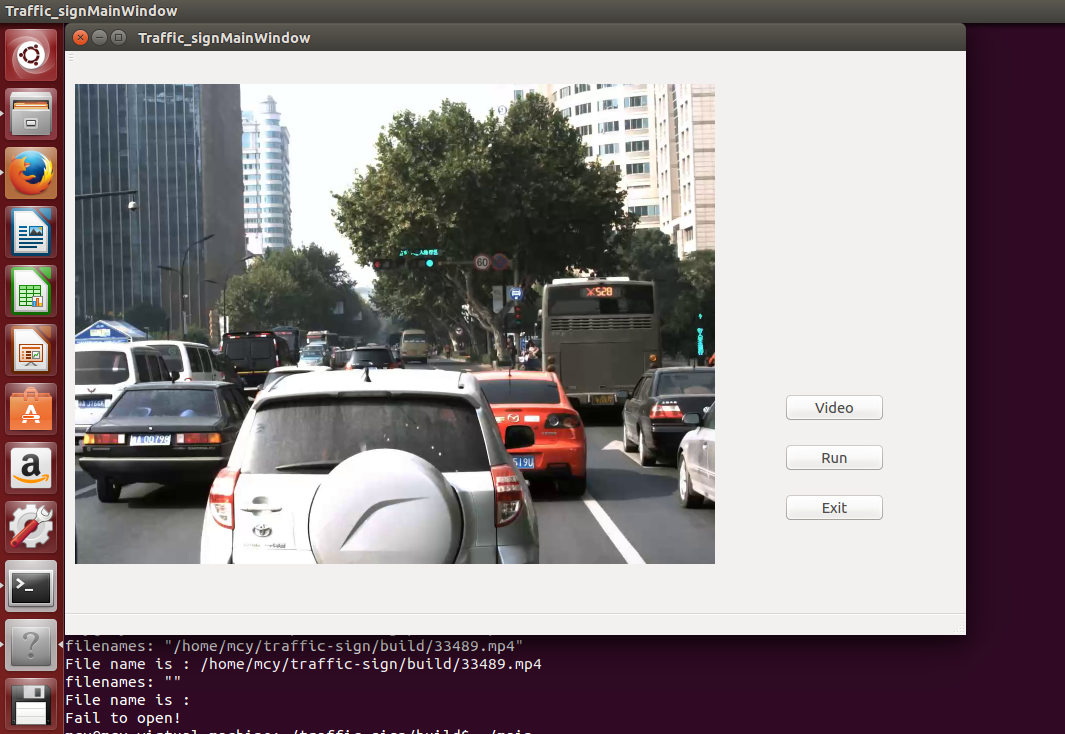
主界面



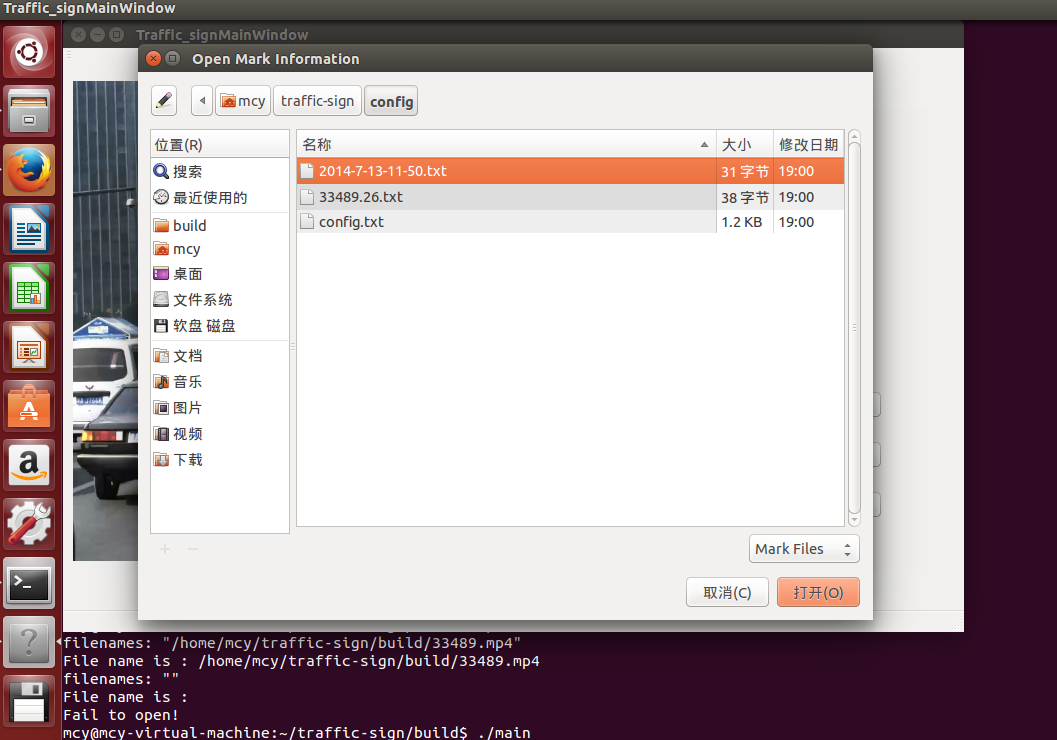
载入视频，按video按钮

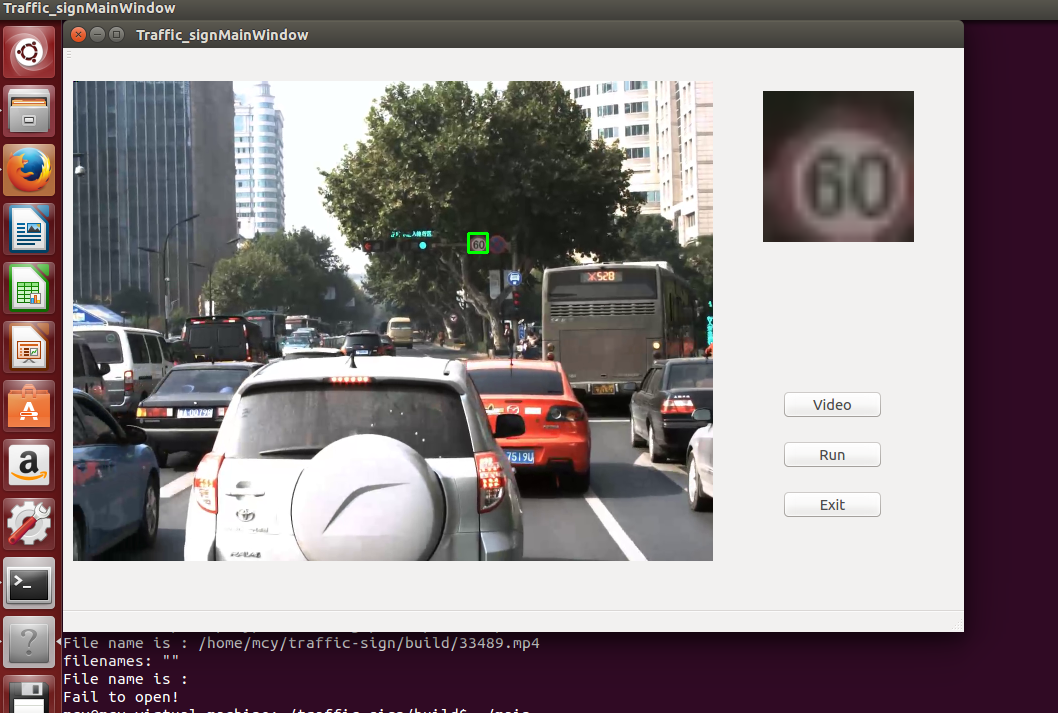


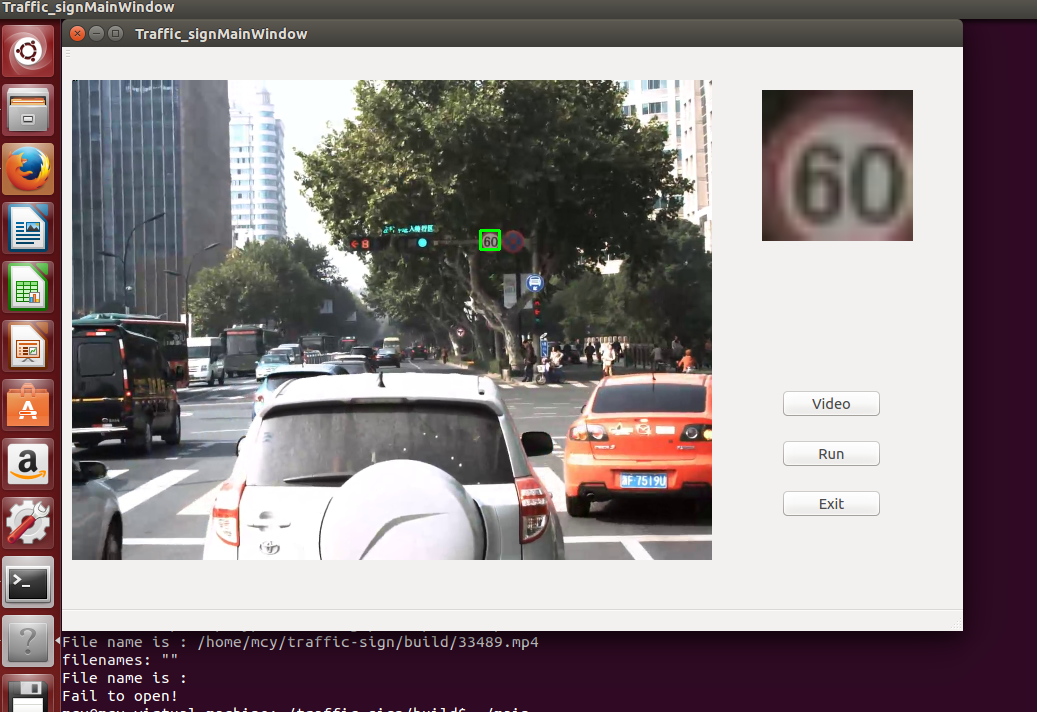
载入33489.mp4视频文件

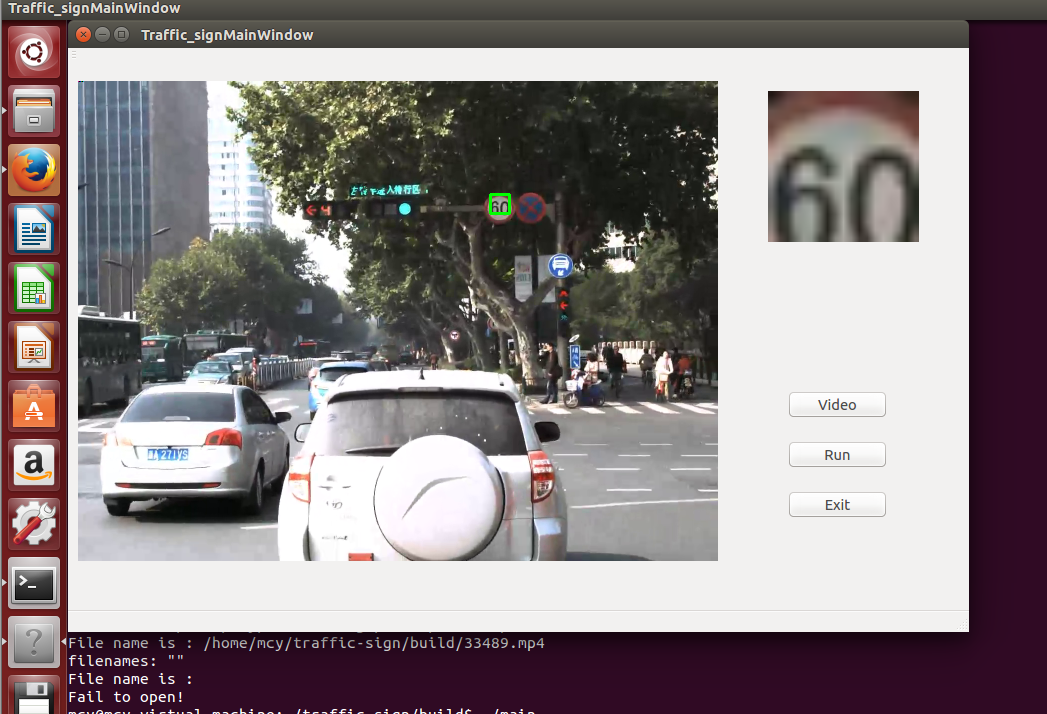


载入配置文件，按run按钮

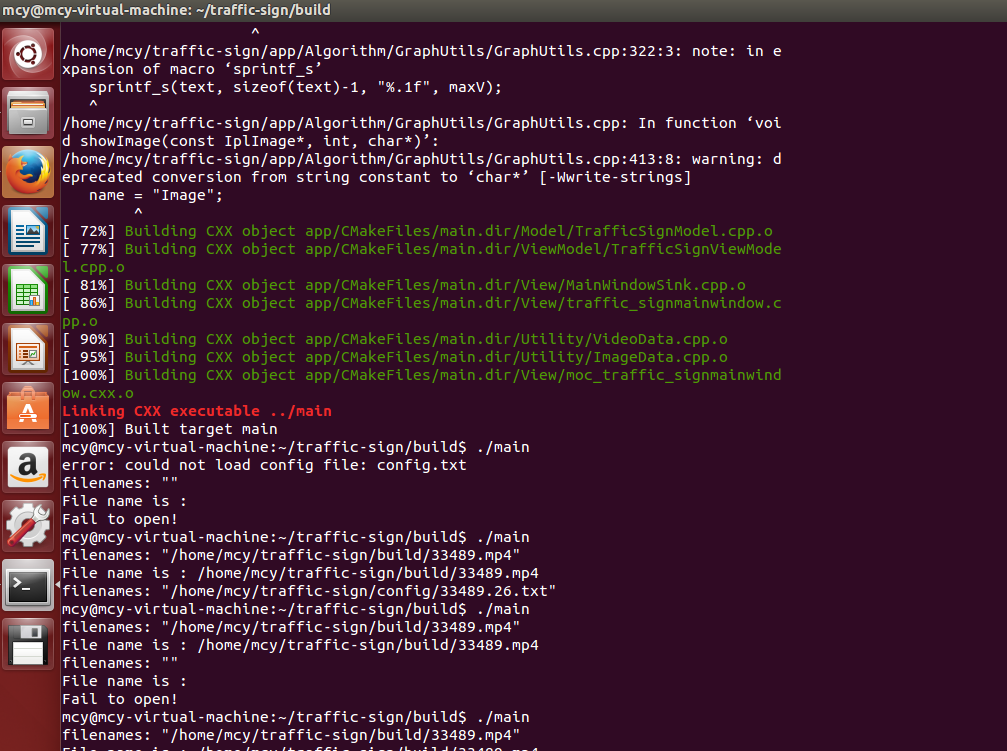
载入333489.txt后运行



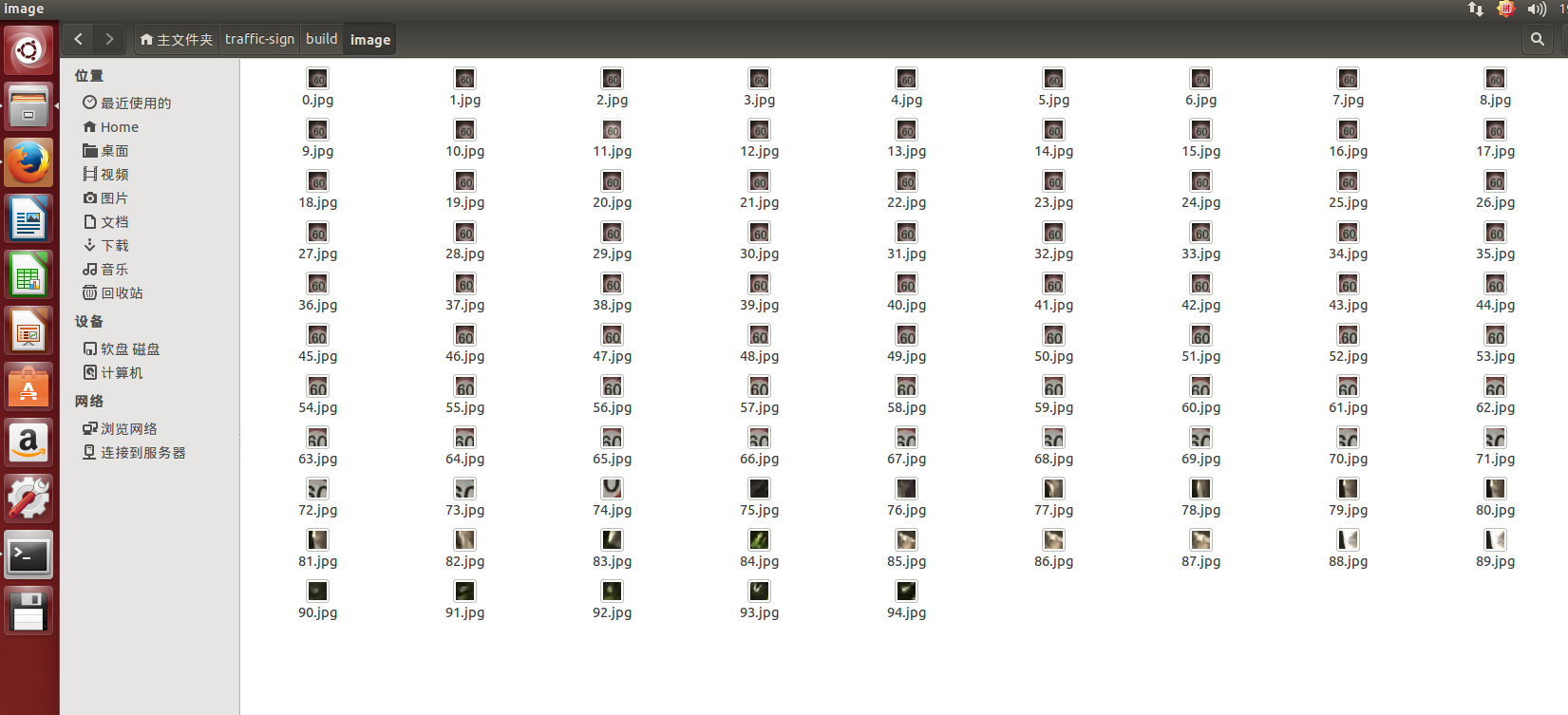


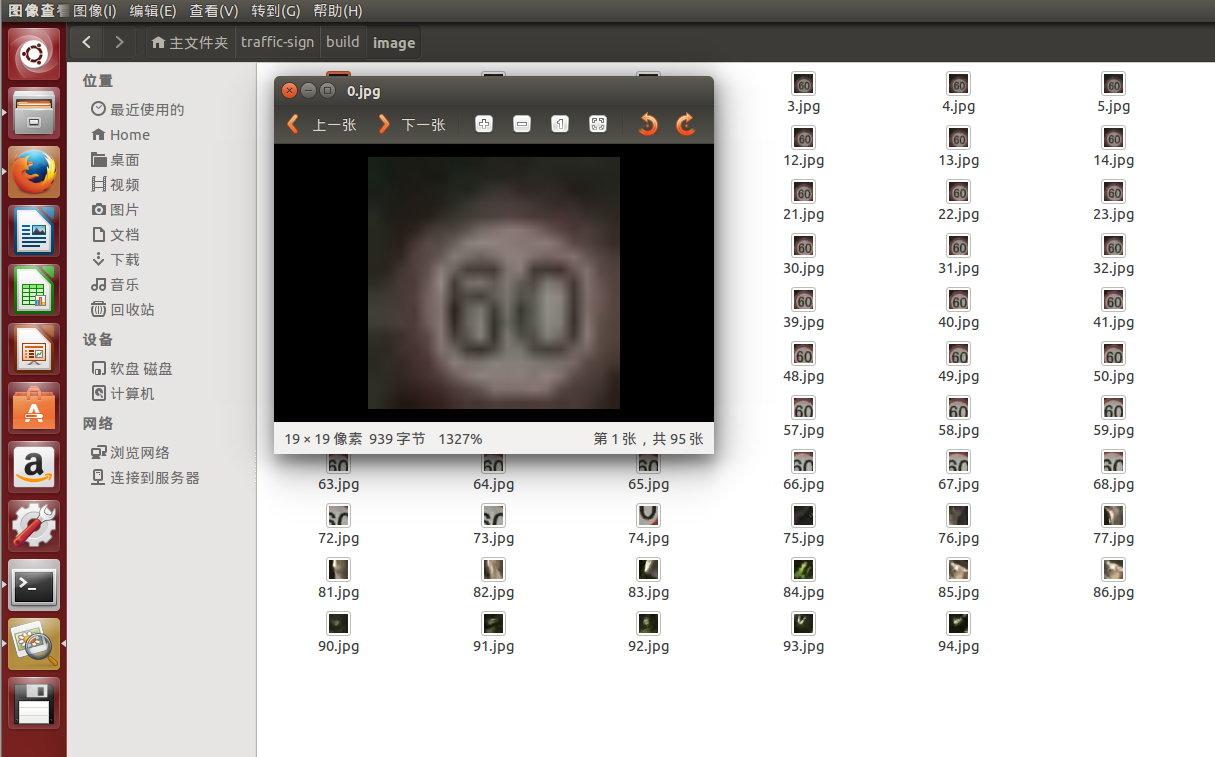


按Exit按钮退出



在image文件中记录下了交通标志的图片文件





## 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：

# 附件（源程序）

## CmakeLists.txt

# This is a CMake build file, for more information consult:

# http://en.wikipedia.org/wiki/CMake

# and

# http://www.cmake.org/Wiki/CMake

# http://www.cmake.org/cmake/help/syntax.html

# http://www.cmake.org/Wiki/CMake\_Useful\_Variables

# http://www.cmake.org/cmake/help/cmake-2-8-docs.html

# to compile the local code you can use: cmake ./ && make -j2

# ----------------------------------------------------------------------

# Base CMake setup

CMAKE\_MINIMUM\_REQUIRED (VERSION 2.8)

FIND\_PACKAGE (OpenCV 2.4 REQUIRED)

#----------------- common setting -------------------

SET(ROOT\_DIR ../)

SET(EXECUTABLE\_OUTPUT\_PATH ${ROOT\_DIR})

INCLUDE\_DIRECTORIES(${CMAKE\_CURRENT\_BINARY\_DIR})

INCLUDE\_DIRECTORIES(

./

./Common

./Model

./Utility

./View

./ViewModel

${local\_QT\_PATH}/include

${local\_INCLUDE\_DIRS\_ALL}

/usr/include/eigen2

)

#MESSAGE(STATUS "CMAKE\_CURRENT\_BINARY\_DIR " ${CMAKE\_CURRENT\_BINARY\_DIR})

#-------------------------- add common files -------------------------

FILE(GLOB src\_main

"./\*.cpp"

"./Algorithm/\*.cpp"

"./Algorithm/GraphUtils/\*.cpp"

"./Model/\*.cpp"

"./ViewModel/\*.cpp"

"./View/\*.cpp"

"./Utility/\*.cpp"

)

FILE(GLOB h\_main

"./\*.hpp"

"./Algorithm/\*.h"

"./Model/\*.hpp"

"./ViewModel/\*.hpp"

"./View/\*.hpp"

"./Utility/\*.hpp"

)

SET(h\_qt

#traffic\_signmainwindow.h

"./View/traffic\_signmainwindow.h"

)

#MESSAGE(STATUS "CPP: " ${src\_main})

#MESSAGE(STATUS "H: " ${h\_qt})

#----------------- set QT4 files ---------------

FIND\_PACKAGE(Qt REQUIRED)

#MESSAGE(STATUS "QT\_USE\_FILE" ${QT\_USE\_FILE}) # include qt cmake files

INCLUDE(${QT\_USE\_FILE})

SET(src\_forms

#traffic\_signmainwindow.ui

./View/traffic\_signmainwindow.ui

)

QT4\_WRAP\_UI(headers\_ui ${src\_forms})

QT4\_WRAP\_CPP(h\_moc ${h\_qt})

#----------------- UNIX and WINDOWS difference -------------

SET(QT\_ROOT "/usr/share/qt4")

SET(opencv\_LIBRARIES

opencv\_core opencv\_imgproc opencv\_highgui opencv\_ml

opencv\_video opencv\_features2d

opencv\_calib3d

opencv\_objdetect opencv\_contrib

opencv\_legacy opencv\_flann

) # quick hack for opencv2.4 support

#------------------ build application -----------

SET(appname main)

ADD\_EXECUTABLE(${appname} ${src\_main} ${h\_moc} ${headers\_ui} ${h\_main}

)

#MESSAGE(STATUS "ui: " ${headers\_ui})

#MESSAGE(STATUS "qt: " ${QT\_LIBRARIES})

TARGET\_LINK\_LIBRARIES(

${appname}

${QT\_QTMAIN\_LIBRARY}

${QT\_LIBRARIES}

${boost\_LIBRARIES}

${opencv\_LIBRARIES}

)

## mian.cpp

#include "traffic\_signmainwindow.h"

#include <QApplication>

#include "Model/TrafficSignModel.hpp"

#include "ViewModel/TrafficSignViewModel.hpp"

#include <boost/shared\_ptr.hpp>

#include <boost/make\_shared.hpp>

using namespace std;

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

{

QApplication a(argc, argv);

Traffic\_signMainWindow w;

boost::shared\_ptr<TrafficSignModel> sp\_TrafficSignModel =

boost::make\_shared<TrafficSignModel>();

boost::shared\_ptr<TrafficSignViewModel> sp\_TrafficSignViewModel =

boost::make\_shared<TrafficSignViewModel>();

sp\_TrafficSignViewModel->SetModel(sp\_TrafficSignModel);

sp\_TrafficSignViewModel->SetEvent(w.GetEvent());

w.SetImage(sp\_TrafficSignViewModel->GetImagePtr(0));

w.SetMark(sp\_TrafficSignViewModel->GetImagePtr(1));

w.SetOpenCommand(sp\_TrafficSignViewModel->GetOpenCommand());

w.SetReadCommand(sp\_TrafficSignViewModel->GetReadCommand());

w.SetRunCommand(sp\_TrafficSignViewModel->GetRunCommand());

w.show();

return a.exec();

## View

### ui\_traffic\_signmianwindows.h

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Form generated from reading UI file 'traffic\_signmainwindow.ui'

\*\*

\*\* Created by: Qt User Interface Compiler version 4.8.6

\*\*

\*\* WARNING! All changes made in this file will be lost when recompiling UI file!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef UI\_TRAFFIC\_SIGNMAINWINDOW\_H

#define UI\_TRAFFIC\_SIGNMAINWINDOW\_H

#include <QtCore/QVariant>

#include <QtGui/QAction>

#include <QtGui/QApplication>

#include <QtGui/QButtonGroup>

#include <QtGui/QHeaderView>

#include <QtGui/QLabel>

#include <QtGui/QMainWindow>

#include <QtGui/QMenuBar>

#include <QtGui/QPushButton>

#include <QtGui/QStatusBar>

#include <QtGui/QTextBrowser>

#include <QtGui/QToolBar>

#include <QtGui/QWidget>

QT\_BEGIN\_NAMESPACE

class Ui\_Traffic\_signMainWindow

{

public:

QWidget \*centralWidget;

QLabel \*VideoLabel;

QLabel \*SignLabel;

QPushButton \*VideoButton;

QPushButton \*ExitButton;

QTextBrowser \*textBrowser;

QMenuBar \*menuBar;

QToolBar \*mainToolBar;

QStatusBar \*statusBar;

void setupUi(QMainWindow \*Traffic\_signMainWindow)

{

if (Traffic\_signMainWindow->objectName().isEmpty())

Traffic\_signMainWindow->setObjectName(QString::fromUtf8("Traffic\_signMainWindow"));

Traffic\_signMainWindow->resize(538, 354);

centralWidget = new QWidget(Traffic\_signMainWindow);

centralWidget->setObjectName(QString::fromUtf8("centralWidget"));

VideoLabel = new QLabel(centralWidget);

VideoLabel->setObjectName(QString::fromUtf8("VideoLabel"));

VideoLabel->setGeometry(QRect(10, 20, 320, 240));

SignLabel = new QLabel(centralWidget);

SignLabel->setObjectName(QString::fromUtf8("SignLabel"));

SignLabel->setGeometry(QRect(360, 20, 111, 71));

VideoButton = new QPushButton(centralWidget);

VideoButton->setObjectName(QString::fromUtf8("VideoButton"));

VideoButton->setGeometry(QRect(380, 210, 99, 27));

ExitButton = new QPushButton(centralWidget);

ExitButton->setObjectName(QString::fromUtf8("ExitButton"));

ExitButton->setGeometry(QRect(380, 260, 99, 27));

textBrowser = new QTextBrowser(centralWidget);

textBrowser->setObjectName(QString::fromUtf8("textBrowser"));

textBrowser->setGeometry(QRect(350, 100, 161, 81));

Traffic\_signMainWindow->setCentralWidget(centralWidget);

menuBar = new QMenuBar(Traffic\_signMainWindow);

menuBar->setObjectName(QString::fromUtf8("menuBar"));

menuBar->setGeometry(QRect(0, 0, 538, 25));

Traffic\_signMainWindow->setMenuBar(menuBar);

mainToolBar = new QToolBar(Traffic\_signMainWindow);

mainToolBar->setObjectName(QString::fromUtf8("mainToolBar"));

Traffic\_signMainWindow->addToolBar(Qt::TopToolBarArea, mainToolBar);

statusBar = new QStatusBar(Traffic\_signMainWindow);

statusBar->setObjectName(QString::fromUtf8("statusBar"));

Traffic\_signMainWindow->setStatusBar(statusBar);

retranslateUi(Traffic\_signMainWindow);

QMetaObject::connectSlotsByName(Traffic\_signMainWindow);

} // setupUi

void retranslateUi(QMainWindow \*Traffic\_signMainWindow)

{

Traffic\_signMainWindow->setWindowTitle(QApplication::translate("Traffic\_signMainWindow", "Traffic\_signMainWindow", 0, QApplication::UnicodeUTF8));

VideoLabel->setText(QApplication::translate("Traffic\_signMainWindow", "Video", 0, QApplication::UnicodeUTF8));

SignLabel->setText(QApplication::translate("Traffic\_signMainWindow", "Sign", 0, QApplication::UnicodeUTF8));

VideoButton->setText(QApplication::translate("Traffic\_signMainWindow", "Video", 0, QApplication::UnicodeUTF8));

ExitButton->setText(QApplication::translate("Traffic\_signMainWindow", "Exit", 0, QApplication::UnicodeUTF8));

} // retranslateUi

};

namespace Ui {

class Traffic\_signMainWindow: public Ui\_Traffic\_signMainWindow {};

} // namespace Ui

QT\_END\_NAMESPACE

#endif // UI\_TRAFFIC\_SIGNMAINWINDOW\_H

### traffic\_signmianwindows.h

#ifndef TRAFFIC\_SIGNMAINWINDOW\_H

#define TRAFFIC\_SIGNMAINWINDOW\_H

#include <QMainWindow>

#include <boost/shared\_ptr.hpp>

#include <boost/make\_shared.hpp>

#include <QtGui>

#include "Common/OpenCommandParam.hpp"

#include "Common/ICommand.hpp"

#include "Common/INotification.hpp"

#include <string>

namespace Ui {

class Traffic\_signMainWindow;

}

class Traffic\_signMainWindow : public QMainWindow

{

Q\_OBJECT

public:

QTimer \*running\_timer;

boost::shared\_ptr<QImage> GetImage(const int &);

boost::shared\_ptr<INotification> GetEvent();

explicit Traffic\_signMainWindow(QWidget \*parent = 0);

~Traffic\_signMainWindow();

void SetOpenCommand(const boost::shared\_ptr<ICommand> &ptr);

void SetReadCommand(const boost::shared\_ptr<ICommand> &ptr);

void SetRunCommand(const boost::shared\_ptr<ICommand> &ptr);

void SetImage(const boost::shared\_ptr<QImage> &ptr);

void SetMark(const boost::shared\_ptr<QImage> &ptr);

QLabel \*GetLabel(const std::string &);

private slots:

void on\_VideoButton\_clicked();

void on\_ExitButton\_clicked();

void on\_RunButton\_clicked();

void run();

private:

Ui::Traffic\_signMainWindow \*ui;

boost::shared\_ptr<ICommand> sp\_OpenCommand;

boost::shared\_ptr<ICommand> sp\_ReadCommand;

boost::shared\_ptr<ICommand> sp\_RunCommand;

boost::shared\_ptr<QImage> sp\_Image, sp\_Mark;

boost::shared\_ptr<INotification> sp\_Event;

bool init\_flag;

};

#endif // TRAFFIC\_SIGNMAINWINDOW\_H

### traffic\_signmianwindows.cpp

#include "traffic\_signmainwindow.h"

#include "ui\_traffic\_signmainwindow.h"

#include "MainWindowSink.hpp"

#include <QFileDialog>

#include <QDebug>

#include <boost/make\_shared.hpp>

#include <signal.h>

Traffic\_signMainWindow::Traffic\_signMainWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::Traffic\_signMainWindow),

running\_timer(new QTimer(this))

{

ui->setupUi(this);

sp\_Event = boost::static\_pointer\_cast<INotification>(

boost::make\_shared<MainWindowSink>(this));

connect(running\_timer, SIGNAL(timeout()), this, SLOT(run()));

}

Traffic\_signMainWindow::~Traffic\_signMainWindow()

{

delete ui;

}

QLabel \* Traffic\_signMainWindow::GetLabel(const std::string &s)

{

if (s=="video") return ui->VideoLabel;

if (s=="mark") return ui->SignLabel;

}

boost::shared\_ptr<QImage> Traffic\_signMainWindow::GetImage(const int &x)

{

if (x==0) return sp\_Image;

else return sp\_Mark;

}

boost::shared\_ptr<INotification> Traffic\_signMainWindow::GetEvent()

{

return sp\_Event;

}

void Traffic\_signMainWindow::SetOpenCommand(const boost::shared\_ptr<ICommand> &ptr) {

sp\_OpenCommand = ptr;

}

void Traffic\_signMainWindow::SetReadCommand(const boost::shared\_ptr<ICommand> &ptr) {

sp\_ReadCommand = ptr;

}

void Traffic\_signMainWindow::SetRunCommand(const boost::shared\_ptr<ICommand> &ptr) {

sp\_RunCommand = ptr;

}

void Traffic\_signMainWindow::SetImage(const boost::shared\_ptr<QImage> &ptr) {

sp\_Image = ptr;

}

void Traffic\_signMainWindow::SetMark(const boost::shared\_ptr<QImage> &ptr) {

sp\_Mark = ptr;

}

void Traffic\_signMainWindow::on\_VideoButton\_clicked()

{

QString fileName = QFileDialog::getOpenFileName(this,

tr("Open Video"), ".",

tr("Video Files (\*.mp4 \*.avi)"));

qDebug()<<"filenames:"<<fileName;

boost::shared\_ptr<OpenCommandParam> param = boost::make\_shared<OpenCommandParam>();

param->SetParam(fileName.toStdString());

sp\_OpenCommand->Execute(boost::static\_pointer\_cast<ICommandParam>(param));

}

void Traffic\_signMainWindow::on\_RunButton\_clicked()

{

init\_flag = true;

QString fileName = QFileDialog::getOpenFileName(this,

tr("Open Mark Information"), ".",

tr("Mark Files (\*.txt)"));

qDebug()<<"filenames:"<<fileName;

boost::shared\_ptr<OpenCommandParam> param = boost::make\_shared<OpenCommandParam>();

param->SetParam(fileName.toStdString());

sp\_ReadCommand->Execute(boost::static\_pointer\_cast<ICommandParam>(param));

running\_timer->start(100);

}

void Traffic\_signMainWindow::on\_ExitButton\_clicked()

{

exit(0);

}

void Traffic\_signMainWindow::run()

{

boost::shared\_ptr<OpenCommandParam> param = boost::make\_shared<OpenCommandParam>();

if (init\_flag)

{

param->SetParam("init");

init\_flag = false;

}

else param->SetParam("run");

sp\_RunCommand->Execute(boost::static\_pointer\_cast<ICommandParam>(param));

}

### MainWindowSink.hpp

#ifndef Traffic\_Sign\_MainWindowSink\_h

#define Traffic\_Sign\_MainWindowSink\_h

#include <string>

#include "Common/INotification.hpp"

#include "View/traffic\_signmainwindow.h"

class MainWindowSink: public INotification {

private:

Traffic\_signMainWindow \*p\_Window;

public:

MainWindowSink(Traffic\_signMainWindow \*p):p\_Window(p)

{

}

void OnPropertyChanged(const std::string &property);

};

#endif

### MainWindowSink.cpp

#include "MainWindowSink.hpp"

void MainWindowSink::OnPropertyChanged(const std::string &property) {

if (property=="ImageData")

{

///DEBUG

//std::cout << "OnPropertyChanged Called" << std::endl;

p\_Window->GetLabel("video")->setPixmap(QPixmap::fromImage(\*(p\_Window->GetImage(0))));

p\_Window->GetLabel("video")->setScaledContents(true);

p\_Window->GetLabel("video")->update();

p\_Window->GetLabel("mark")->setPixmap(QPixmap::fromImage(\*(p\_Window->GetImage(1))));

p\_Window->GetLabel("mark")->setScaledContents(true);

p\_Window->GetLabel("mark")->update();

}

if (property=="TrackDone")

{

p\_Window->running\_timer->stop();

}

}

## ViewModel

### TrafficSignViewModel.hpp

#ifndef Traffic\_Sign\_ViewModel\_h

#define Traffic\_Sign\_ViewModel\_h

#include <boost/shared\_ptr.hpp>

#include <boost/make\_shared.hpp>

#include <string>

#include <vector>

#include <opencv2/opencv.hpp>

#include "Common/ICommand.hpp"

#include "Common/INotification.hpp"

#include "Model/TrafficSignModel.hpp"

#include <QMainWindow>

class TrafficSignViewModel {

private:

boost::shared\_ptr<ICommand> sp\_OpenCommand;

boost::shared\_ptr<ICommand> sp\_RunCommand;

boost::shared\_ptr<ICommand> sp\_ReadCommand;

boost::shared\_ptr<TrafficSignModel> sp\_Model;

//std::vector< boost::shared\_ptr<INotification> > events;

boost::shared\_ptr<INotification> event;

boost::shared\_ptr<QImage> sp\_image\_view, sp\_mark\_view;

void fireEvent(const std::string &property);

public:

TrafficSignViewModel();

boost::shared\_ptr<ICommand> GetOpenCommand();

boost::shared\_ptr<ICommand> GetRunCommand();

boost::shared\_ptr<ICommand> GetReadCommand();

boost::shared\_ptr<QImage> GetImagePtr(const int &);

void SetEvent(const boost::shared\_ptr<INotification> &e);

void SetModel(boost::shared\_ptr<TrafficSignModel> &model);

void OpenVideo(const std::string &filename);

void TrackSign(const std::string &filename);

void ReadMark(const std::string &filename);

void RefreshImage();

};

#endif

### TrafficSignViewModel.cpp

#include "TrafficSignViewModel.hpp"

#include "OpenCommand.hpp"

#include "RunCommand.hpp"

#include "ReadCommand.hpp"

#include "Utility/MarkData.hpp"

#include "Algorithm/Tracker.h"

#include "Algorithm/Config.h"

#include <fstream>

#define SPEED 2

using namespace std;

using namespace cv;

TrafficSignViewModel::TrafficSignViewModel() {

sp\_OpenCommand = boost::static\_pointer\_cast<ICommand, OpenCommand>(

boost::make\_shared<OpenCommand>(this));

sp\_RunCommand = boost::static\_pointer\_cast<ICommand, RunCommand>(

boost::make\_shared<RunCommand>(this));

sp\_ReadCommand = boost::static\_pointer\_cast<ICommand, ReadCommand>(

boost::make\_shared<ReadCommand>(this));

sp\_image\_view = boost::make\_shared<QImage>();

sp\_mark\_view = boost::make\_shared<QImage>();

}

void TrafficSignViewModel::SetEvent(const boost::shared\_ptr<INotification> &e) {

event = e;

}

void TrafficSignViewModel::fireEvent(const string &property) {

event->OnPropertyChanged(property);

}

boost::shared\_ptr<QImage> TrafficSignViewModel::GetImagePtr(const int &x) {

if (x==0) return sp\_image\_view;

else return sp\_mark\_view;

}

boost::shared\_ptr<ICommand> TrafficSignViewModel::GetOpenCommand() {

return sp\_OpenCommand;

}

boost::shared\_ptr<ICommand> TrafficSignViewModel::GetReadCommand() {

return sp\_ReadCommand;

}

boost::shared\_ptr<ICommand> TrafficSignViewModel::GetRunCommand() {

return sp\_RunCommand;

}

void TrafficSignViewModel::SetModel(boost::shared\_ptr<TrafficSignModel> &model) {

sp\_Model = model;

}

void TrafficSignViewModel::OpenVideo(const string &filename) {

sp\_Model->OpenVideo(filename);

///DEBUG

//cout<<"ViewModel opened video!"<<endl;

RefreshImage();

}

void TrafficSignViewModel::ReadMark(const string &filename) {

sp\_Model->ReadMark(filename);

}

void rectangle(Mat& rMat, const FloatRect& rRect, const Scalar& rColour)

{

IntRect r(rRect);

rectangle(rMat, Point(r.XMin(), r.YMin()), Point(r.XMax(), r.YMax()), rColour, 2);

}

void TrafficSignViewModel::TrackSign(const string &mode) {

//cout << "TrackSign" << endl;

static Config conf(\*sp\_Model->GetConfig());

static Tracker tracker(conf);

static FloatRect initBB;

static int frameInd;

static MarkData sMark, tMark;

Mat result(conf.frameHeight, conf.frameWidth, CV\_8UC3);

Mat frame;

Mat frameOrig(\*sp\_Model->GetFrame(frameInd));

if (mode=="init")

{

sMark = \*sp\_Model->GetMark(0);

tMark = \*sp\_Model->GetMark(1);

frameInd = sMark.frame;

float scalex, scaley;

scalex = 1.0\*frameOrig.rows/conf.frameHeight;

scaley = 1.0\*frameOrig.cols/conf.frameWidth;

initBB = IntRect(sMark.lx/scalex, sMark.ly/scaley, (sMark.rx-sMark.lx)/scalex, (sMark.ry-sMark.ly)/scaley);

}

if (frameOrig.empty())

{

cout << "error: could not read frame" << endl;

exit(-1);

}

resize(frameOrig, frame, Size(conf.frameWidth, conf.frameHeight));

frame.copyTo(result);

if (frameInd == sMark.frame)

{

tracker.Initialise(frame, initBB);

}

if (tracker.IsInitialised())

{

tracker.Track(frame);

//TODO

if (true) //(outFile)

{

const IntRect& bb = tracker.GetBB();

Mat image(result, cv::Rect(bb.XMin(), bb.YMin(), bb.Width(), bb.Height()));

char outImage[128];

sprintf(outImage, "image/%d.jpg", frameInd/SPEED);

//cout << "Write to: " << outImage << endl;

imwrite(outImage, image);

sp\_Model->SetMark(image);

}

rectangle(result, tracker.GetBB(), CV\_RGB(0, 255, 0));

}

//else sp\_Model->SetMark(result);

sp\_Model->SetImage(result);

RefreshImage();

if (frameInd >= tMark.frame) fireEvent("TrackDone");

frameInd += SPEED;

}

void TrafficSignViewModel::RefreshImage() {

cv::Mat image = \*sp\_Model->GetImage(0)->GetImage();

cv::Mat rgb;

if(image.channels()==3)

{

//cvt Mat BGR 2 QImage RGB

cvtColor(image,rgb,CV\_BGR2RGB);

\*sp\_image\_view = QImage((const unsigned char\*)(rgb.data),

rgb.cols,rgb.rows,

rgb.cols\*rgb.channels(),

QImage::Format\_RGB888);

}

else

{

\*sp\_image\_view = QImage((const unsigned char\*)(image.data),

image.cols,image.rows,

image.cols\*image.channels(),

QImage::Format\_RGB888);

}

fireEvent("ImageData");

image = \*sp\_Model->GetImage(1)->GetImage();

if(image.channels()==3)

{

//cvt Mat BGR 2 QImage RGB

cvtColor(image,rgb,CV\_BGR2RGB);

\*sp\_mark\_view = QImage((const unsigned char\*)(rgb.data),

rgb.cols,rgb.rows,

rgb.cols\*rgb.channels(),

QImage::Format\_RGB888);

}

else

{

\*sp\_mark\_view = QImage((const unsigned char\*)(image.data),

image.cols,image.rows,

image.cols\*image.channels(),

QImage::Format\_RGB888);

}

fireEvent("ImageData");

}

### RunCommand.hpp

#ifndef Traffic\_Sign\_RunCommand\_h

#define Traffic\_Sign\_RunCommand\_h

#include <string>

#include <boost/shared\_ptr.hpp>

#include "Common/ICommand.hpp"

#include "TrafficSignViewModel.hpp"

class RunCommand: public ICommand {

private:

TrafficSignViewModel \*m\_ViewModel;

public:

RunCommand(TrafficSignViewModel \*m) {

m\_ViewModel = m;

};

~RunCommand() {};

void Execute(const boost::shared\_ptr<ICommandParam> &param) {

//m\_ViewModel->flag=true;

m\_ViewModel->TrackSign(\*((std::string\*)param->GetParam(0)));

};

};

#endif

### ReadCommand.hpp

#ifndef Traffic\_Sign\_ReadCommand\_h

#define Traffic\_Sign\_ReadCommand\_h

#include <string>

#include <boost/shared\_ptr.hpp>

#include "Common/ICommand.hpp"

#include "TrafficSignViewModel.hpp"

class ReadCommand: public ICommand {

private:

TrafficSignViewModel \*m\_ViewModel;

public:

ReadCommand(TrafficSignViewModel \*m) {

m\_ViewModel = m;

};

~ReadCommand() {};

void Execute(const boost::shared\_ptr<ICommandParam> &param) {

//m\_ViewModel->flag=true;

m\_ViewModel->ReadMark(\*((std::string\*)param->GetParam(0)));

};

};

#endif

### OpenCommand.hpp

#ifndef Traffic\_Sign\_OpenCommand\_h

#define Traffic\_Sign\_OpenCommand\_h

#include <string>

#include <boost/shared\_ptr.hpp>

#include "Common/ICommand.hpp"

#include "TrafficSignViewModel.hpp"

class OpenCommand: public ICommand {

private:

TrafficSignViewModel \*m\_ViewModel;

public:

OpenCommand(TrafficSignViewModel \*m) {

m\_ViewModel = m;

};

~OpenCommand() {};

void Execute(const boost::shared\_ptr<ICommandParam> &param) {

m\_ViewModel->OpenVideo(\*((std::string\*)param->GetParam(0)));

};

};

#endif

## Utility

### VideoData.hpp

#ifndef Traffic\_Sign\_VideoData

#define Traffic\_Sign\_VideoData

#include <iostream>

#include "opencv2/opencv.hpp"

class VideoData {

private:

cv::VideoCapture cap;

cv::Mat frame;

int currentFrame, totFrame;

public:

VideoData();

virtual ~VideoData();

void OpenVideo(const std::string &);

cv::Mat \*GetFrame();

cv::Mat \*GetFrame(int);

};

#endif

### VideoData.cpp

#include <iostream>

#include "opencv2/opencv.hpp"

#include "VideoData.hpp"

using namespace std;

using namespace cv;

VideoData::VideoData()

{

currentFrame = 1;

totFrame = 0;

}

VideoData::~VideoData()

{

}

void VideoData::OpenVideo(const string &FileName)

{

cout << "File name is : " << FileName << endl;

cap.open(FileName);

if(!cap.isOpened()) { // check if we succeeded

cout << "Fail to open!" << endl;

exit(0);

}

totFrame = cap.get(CV\_CAP\_PROP\_FRAME\_COUNT);

}

Mat \* VideoData::GetFrame()

{

if(currentFrame > totFrame)

{

exit(0);

}

cap.set(CV\_CAP\_PROP\_POS\_FRAMES, currentFrame++);

if(!cap.read(frame)) // read fail

{

cout << "Fail to read the video!" << endl;

exit(0);

}

///DEBUG

//cout<<"Success to read"<<endl;

//namedWindow("Display Image", CV\_WINDOW\_AUTOSIZE );

//imshow("Display Image", frame);

//waitKey(0);

//int x;

//cin>>x;

return & frame;

}

Mat \* VideoData::GetFrame(int givenFrame)

{

if(givenFrame > totFrame)

{

exit(0);

}

cap.set( CV\_CAP\_PROP\_POS\_FRAMES, givenFrame);

if(!cap.read(frame)) { // read fail

cout << "Fail to read the video!" << endl;

exit(0);

}

return &frame;

}

### MarkData.hpp

#ifndef Traffic\_Sign\_MarkData

#define Traffic\_Sign\_MarkData

struct MarkDataType {

int frame;

int lx, ly, rx, ry;

};

typedef struct MarkDataType MarkData;

#endif

### ImageData.hpp

#ifndef Traffic\_Sign\_ImageData\_h

#define Traffic\_Sign\_ImageData\_h

#include <opencv2/opencv.hpp>

class ImageData {

private:

cv::Mat image;

public:

cv::Mat \* GetImage();

void SetImage(const cv::Mat & image);

};

#endif

### ImageData.cpp

#include "ImageData.hpp"

#include <opencv2/opencv.hpp>

#include <iostream>

cv::Mat \* ImageData::GetImage()

{

return &(this->image);

}

void ImageData::SetImage(const cv::Mat &input\_image)

{

image=input\_image.clone();

}

## Model

### TrafficSignModel.hpp

#ifndef traffic\_sign\_Model\_h

#define traffic\_sign\_Model\_h

#include <boost/shared\_ptr.hpp>

#include <string>

#include <list>

#include "Utility/ImageData.hpp"

#include "Utility/VideoData.hpp"

#include "Utility/MarkData.hpp"

#include "Algorithm/Tracker.h"

#include "Algorithm/Config.h"

typedef std::list<MarkData> MarksData;

class TrafficSignModel {

private:

boost::shared\_ptr<ImageData> sp\_image, sp\_mark;

boost::shared\_ptr<VideoData> sp\_video;

boost::shared\_ptr<MarkData> sp\_sMark, sp\_tMark;

boost::shared\_ptr<Config> sp\_config;

std::string videoName;

public:

TrafficSignModel();

~TrafficSignModel();

void OpenVideo(const std::string &filename);

void SetImage(const cv::Mat &image);

void SetMark(const cv::Mat &image);

void ReadMark(const std::string &filename);

boost::shared\_ptr<ImageData> GetImage(int);

boost::shared\_ptr<Config> GetConfig();

boost::shared\_ptr<MarkData> GetMark(int);

cv::Mat \*GetFrame(int frameInd);

const std::string &GetVideoName();

};

#endif

### TrafficSignModel.cpp

#include "TrafficSignModel.hpp"

#include <string>

#include <boost/make\_shared.hpp>

#include <fstream>

using namespace std;

TrafficSignModel::TrafficSignModel() {

sp\_image = boost::make\_shared<ImageData>();

sp\_mark = boost::make\_shared<ImageData>();

sp\_video = boost::make\_shared<VideoData>();

sp\_sMark = boost::make\_shared<MarkData>();

sp\_tMark = boost::make\_shared<MarkData>();

sp\_config = boost::make\_shared<Config>("config.txt");

srand(sp\_config->seed);

//

}

TrafficSignModel::~TrafficSignModel() {}

void TrafficSignModel::OpenVideo(const string &filename) {

sp\_video->OpenVideo(filename);

sp\_image->SetImage(\*(sp\_video->GetFrame()));

videoName = filename;

}

void TrafficSignModel::ReadMark(const string &filename) {

ifstream fin(filename.c\_str());

if (!fin.is\_open()) {

cout << "File is not opened properly!" << endl;

exit(-1);

}

fin >> sp\_sMark->frame >> sp\_sMark->lx >> sp\_sMark->ly >> sp\_sMark->rx >> sp\_sMark->ry;

fin >> sp\_tMark->frame >> sp\_tMark->lx >> sp\_tMark->ly >> sp\_tMark->rx >> sp\_tMark->ry;

fin.close();

}

boost::shared\_ptr<ImageData> TrafficSignModel::GetImage(int x) {

if (x==0) return sp\_image;

else return sp\_mark;

}

boost::shared\_ptr<Config> TrafficSignModel::GetConfig() {

return sp\_config;

}

boost::shared\_ptr<MarkData> TrafficSignModel::GetMark(int x) {

if (x==0) return sp\_sMark;

else return sp\_tMark;

}

void TrafficSignModel::SetImage(const cv::Mat &image) {

sp\_image->SetImage(image);

}

void TrafficSignModel::SetMark(const cv::Mat &image) {

sp\_mark->SetImage(image);

}

cv::Mat \*TrafficSignModel::GetFrame(int frameInd) {

return sp\_video->GetFrame(frameInd);

}

const string &TrafficSignModel::GetVideoName() {

return videoName;

}

## Common

### OpenCommandParam.hpp

#ifndef Traffic\_Sign\_OpenCommandParam\_h

#define Traffic\_Sign\_OpenCommandParam\_h

#include <string>

#include "ICommand.hpp"

class OpenCommandParam: public ICommandParam {

private:

std::string video\_filename;

std::string mark\_filename;//标记好的信息

public:

OpenCommandParam() {};

~OpenCommandParam() {};

int GetParamNum() { return 1; };

void SetParam(const std::string &s) { video\_filename=s; };

void\* GetParam(int index) {

switch(index)

{

case 0: return &video\_filename; break;

case 1: return &mark\_filename; break;

}

};

};

#endif

### INotification.hpp

#ifndef Traffic\_Sign\_Notification

#define Traffic\_Sign\_Notification

#include <string>

class INotification {

private:

public:

INotification(){};

virtual ~INotification(){};

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

};

#endif

### ICommand.hpp

#ifndef traffic\_sign\_ICommand\_h

#define traffic\_sign\_ICommand\_h

#include <boost/shared\_ptr.hpp>

class ICommandParam {

public:

ICommandParam() {}

virtual ~ICommandParam() {}

virtual int GetParamNum()=0;

virtual void\* GetParam(int index)=0;

};

class ICommand {

public:

ICommand() {}

virtual ~ICommand() {}

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

};

#endif