**新型冠状病毒药物活性预测程序**

（使用说明书）

目录

[一、引言 3](#_Toc29305)

[1.1编写目的 3](#_Toc4192)

[1.2背景 3](#_Toc10085)

[1.3参考资料 3](#_Toc14042)

[二、用途 4](#_Toc22244)

[2.1功能 4](#_Toc26554)

[2.2性能 4](#_Toc3532)

[三、运行环境 5](#_Toc20362)

[3.1硬件设备 5](#_Toc30048)

[3.2支持软件 5](#_Toc28125)

[3.3支持库 5](#_Toc1247)

[四、平台操作说明 6](#_Toc17180)

[4.1海康威视设备相关数据的获取 6](#_Toc12425)

[4.2登录界面 8](#_Toc6660)

[4.3视频监控功能及操作 8](#_Toc2495)

[4.4录像回放功能及操作 9](#_Toc12733)

[4.5视频回放功能及操作 11](#_Toc8486)

[4.6智能计数功能及操作 12](#_Toc30190)

[4.7电子地图功能及操作 13](#_Toc17151)

[4.8报表打印功能及操作 15](#_Toc7234)

[五、源码附录 18](#_Toc9993)

# 一、引言

## 1.1编写目的

本说明书为指导生物信息学专业相关人员使用“新型冠状病毒药物活性预测程序”而编写，希望该手册使他们在使用该系统过程中能起到无师自通的作用。本手册介绍了“新型冠状病毒药物有活性预测程序”的基本使用方法。

## 1.2背景

本程序由湖南省衡阳市南华大学计算机学院相关人员编写,使用者是生物信息学相关领域研究人员。本程序将有助于减少专业领域人员对SARS-COV-2病毒药物活性检测时的工作量。

## 1.3参考资料

计算机软件用户手册国家标准

数据设计与实现 王能斌等编 华中理工出版社

海康威视SDK

OpenCV官方文档

《Deep Learning》 Ian GoodfellowYoshua Bengio中英文版

Scikit-learn官方文档

# 二、用途

## 2.1功能

长江崩岸治理施工过程智能监控平台，主要由视频监控、录像回放、视频回放、智能计数、电子地图、报表打印等六大模块组成。平台界面如图1所示。

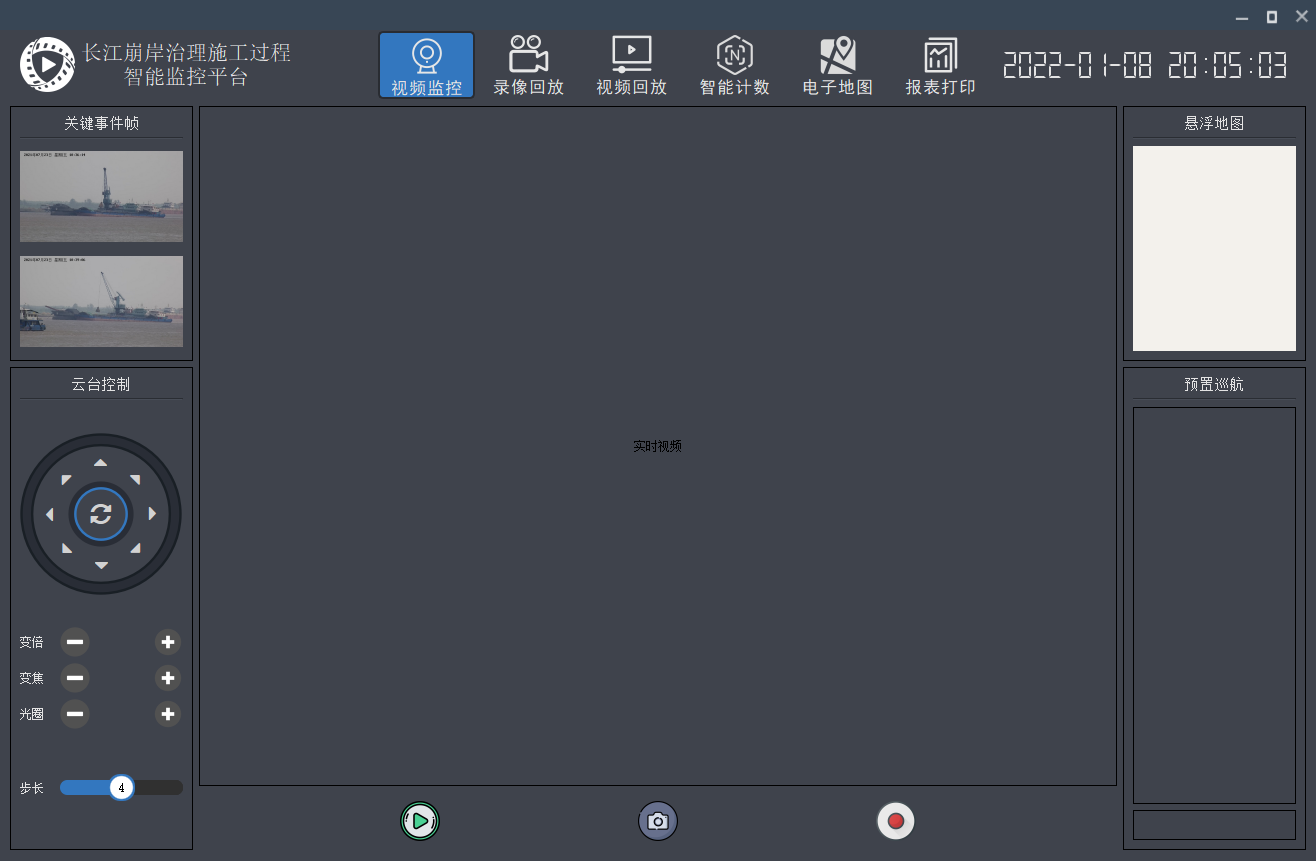


图1

## 2.2性能

本平台利用先进的机器视觉技术对施工过程进行全流程的智能分析，对施工过程的重要事件自动识别并以图像日志形式记录存档，进一步利用机器视觉技术对抛投石料、构件等进行自动识别与统计计数、保证施工过程管控评价的公平与客观，做到重点堤段崩岸治理过程管控有据可依、有源可溯，从而促进施工过程的科学规范。

# 三、运行环境

## 3.1硬件设备

系统 windows7及以上的64位操作系统

打印机 windows支持的打印机

## 3.2支持软件

开发工具 Pycharm

开发工具 jupyter-notebook

开发工具 python

开发工具 streamlit

## 3.3支持库

Python 3.8.13

streamlit

java 8

# 四、平台操作说明

## 4.1海康威视设备相关数据的获取

由于本平台利用海康威视智能摄像机搭建智能数据采集模块，因此需要获取海康威视智能摄像机的IP地址，且设置摄像机的账号密码等，故需要先行下载海康威视官方软件iVMS-4200获取摄像机数据。

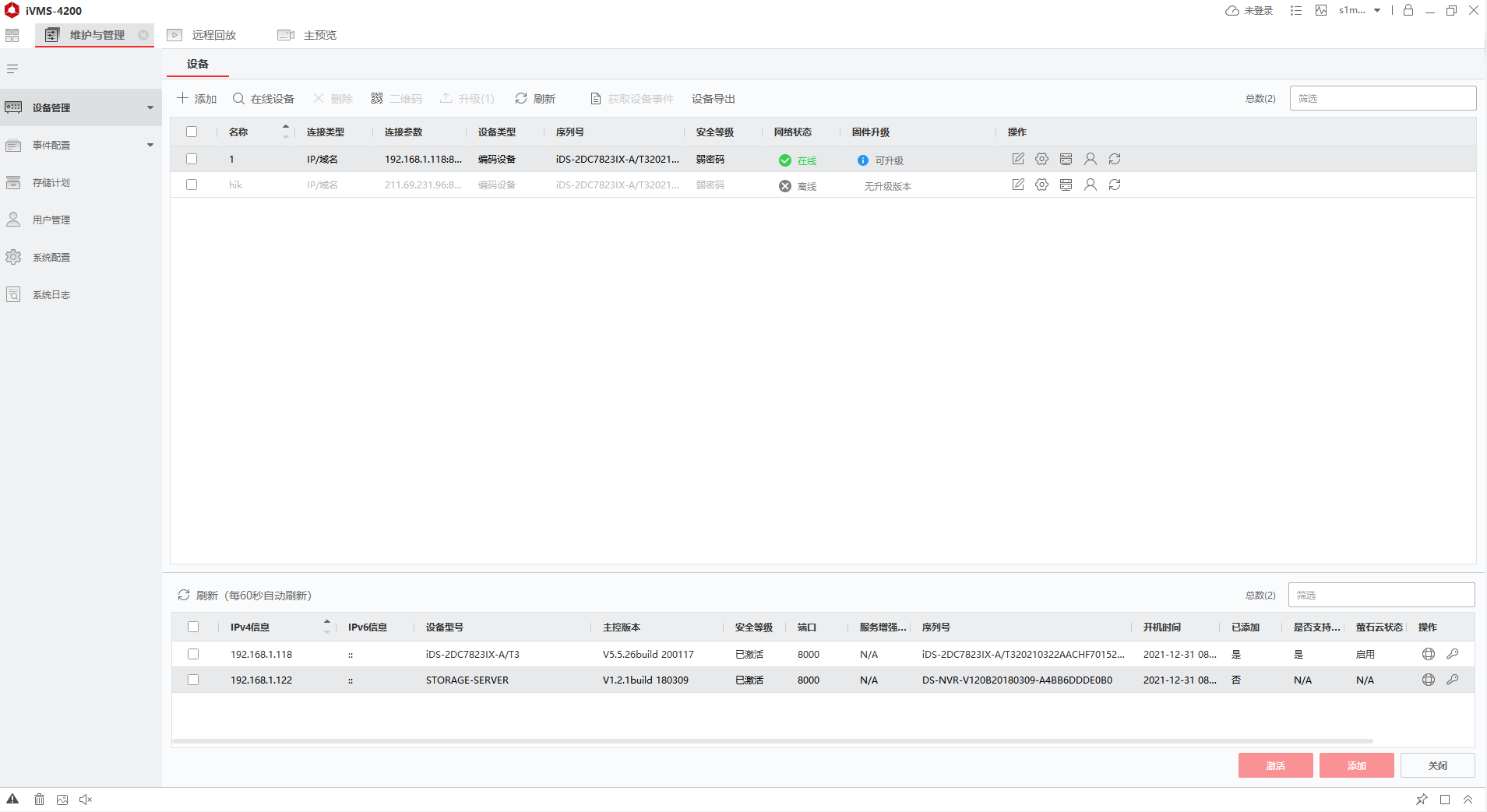
以下以iVMS-4200 3.5.0.9版本为例进行演示操作，获取摄像机的IP地址及账号密码。当摄像头连接网络后，点击iVMS-4200软件，如图2所示，为该软件主界面。

图2

在主界面中点击IMG_256搜索在线设备按钮，主界面下方会出现局域网内所存在的所有的海康威视相关设备，如图3所示，左边表格第一行第一列，IPv4信息下192.168.1.118即为该摄像机的IP地址。

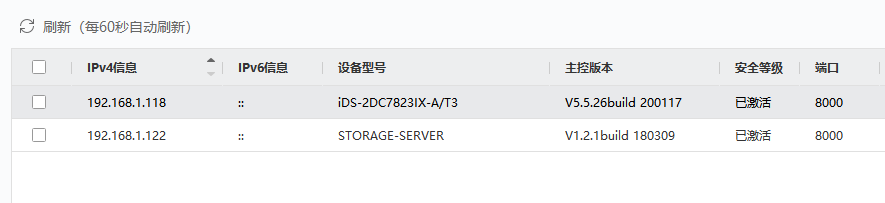


图3

之后，勾选左边的方框，如下图4所示。

图4

再点击主界面左上角IMG_256添加按钮，如图5所示，依次在名称、地址、用户名、密码中填入用户设置的账号密码。（注：地址必须为摄像机的IP地址）



图5

## 4.2登录界面



图6

登录界面如图6所示，在IP地址栏内输入之前查找到的摄像机IP地址（例如：192.168.1.118），在用户名、密码栏填入之前用户设置的用户名和密码，点击登录按钮，即可登录长江崩岸治理施工过程智能监控平台。

## 4.3视频监控功能及操作

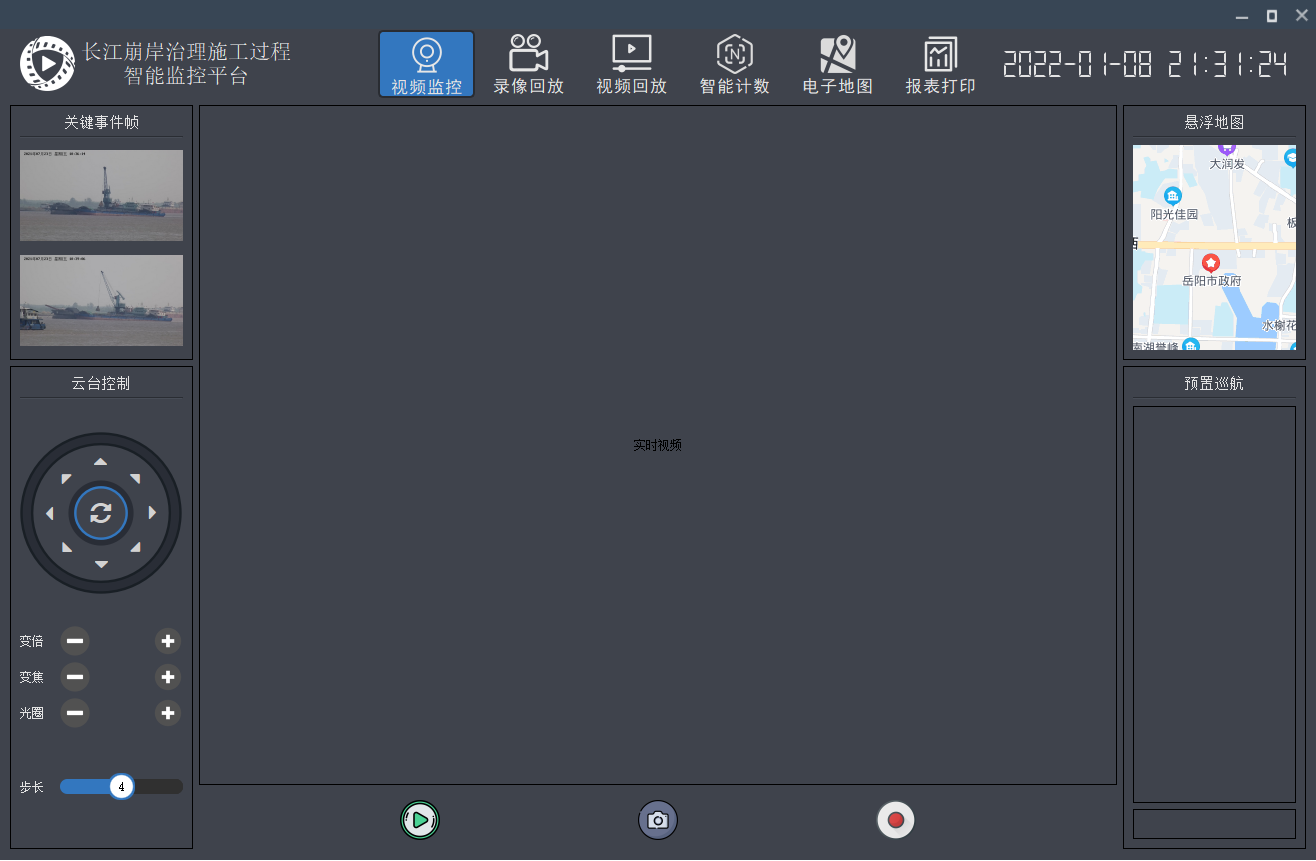


图7

视频监控界面如图7所示，点击IMG_256预览按钮，平台将远程连接海康威视智能摄像机，并在界面上实时显示监控画面；点击IMG_256截图按钮，能够将当前这一时刻的监控画面保存至软件根目录下的Capture文件夹中，点击IMG_256录像按钮，能够从当前时刻开始对监控画面进行录像，再次点击录像按钮即可停止录像，录像文件自动以MP4格式保存在根目录下的Capture文件夹中。

在云台控制栏下，如图8所示，8个方向键为摄像机的控制键，可以控制摄像机的转向；通过设置变倍、变焦、光圈等可以调节摄像机的清晰度，焦距等；通过设置步长可以调节摄像机单次转动的幅度。



图8

## 4.4录像回放功能及操作

该界面功能主要是回放及下载智能摄像机内存中保留的录像文件,点击左上角的录像回放按钮，切换至录像回放界面，如图9所示。

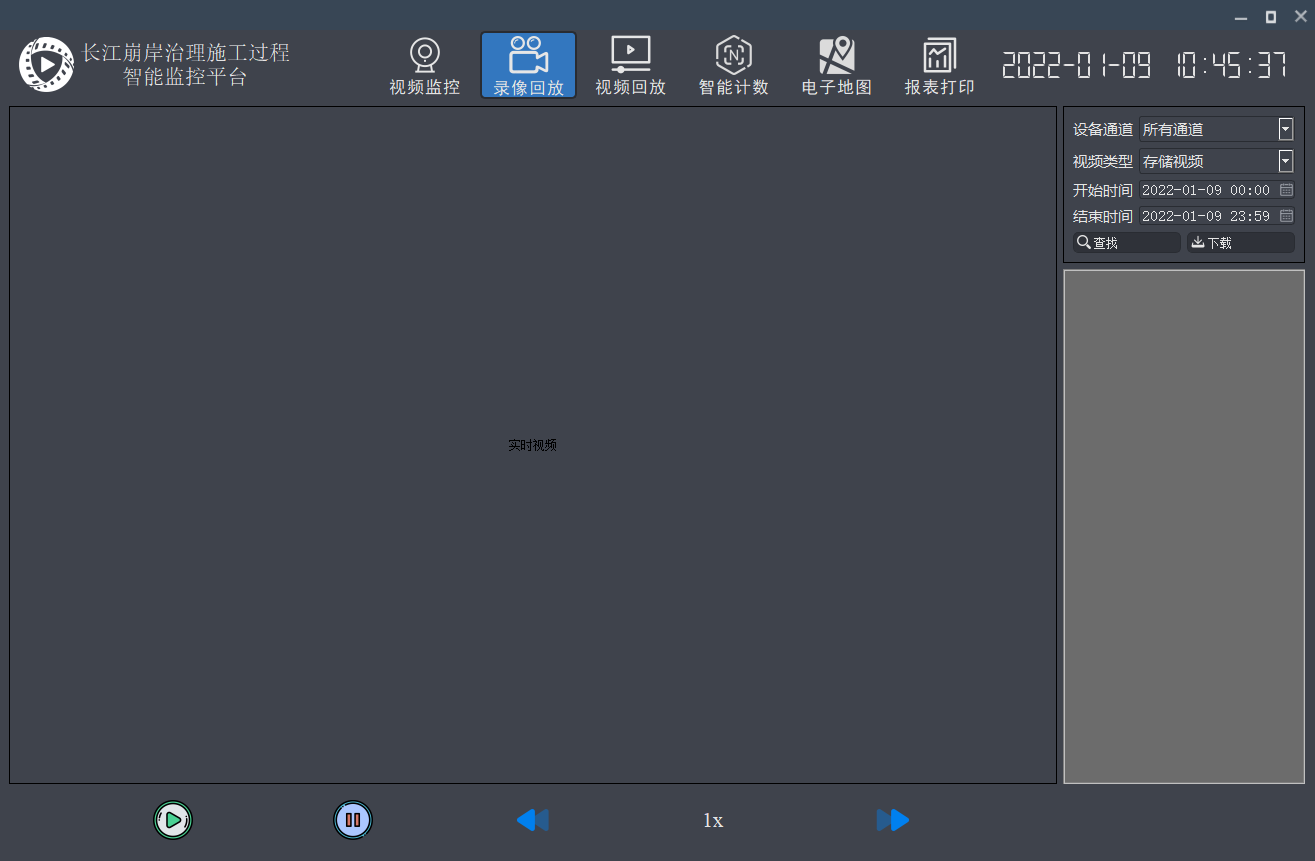


图9

在录像回放界面右上角的开始时间栏和结束时间栏内填入指定时间，如图10所示，图为指定2021年10月20日到2021年10月25日时间段内的视频。

IMG_256

图10

点击IMG_256查找按钮，即可查找指定时间内的视频，如图11所示；点击IMG_256下载按钮，即可下载指定时间内的视频；点击IMG_256回放按钮即可回放指定时间内的视频；点击IMG_256暂停按钮，即可暂停回放视频；点击IMG_256IMG_256倍速按钮，可以调节回放速度，快速查看回放视频。

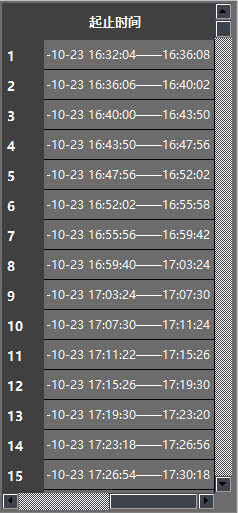


图11

## 4.5视频回放功能及操作

该界面功能主要是回放本地文件中的视频文件。点击视频回放按钮，切换至视频回放界面，如图12所示。点击IMG_256打开按钮，选择指定视频进行回放，下面光轴IMG_256为回放视频时长，滑动光标即可选中时间点回放，点击IMG_256暂停按钮，即可暂停回放视频。



图12

## 4.6智能计数功能及操作

该界面的功能主要是基于机器视觉技术对施工过程进行全流程的智能分析，对施工过程的重要事件自动识别并以图像日志形式记录保存至MySQL数据库中，对抛投石料、构件等进行自动识别与统计计数，如图13所示。

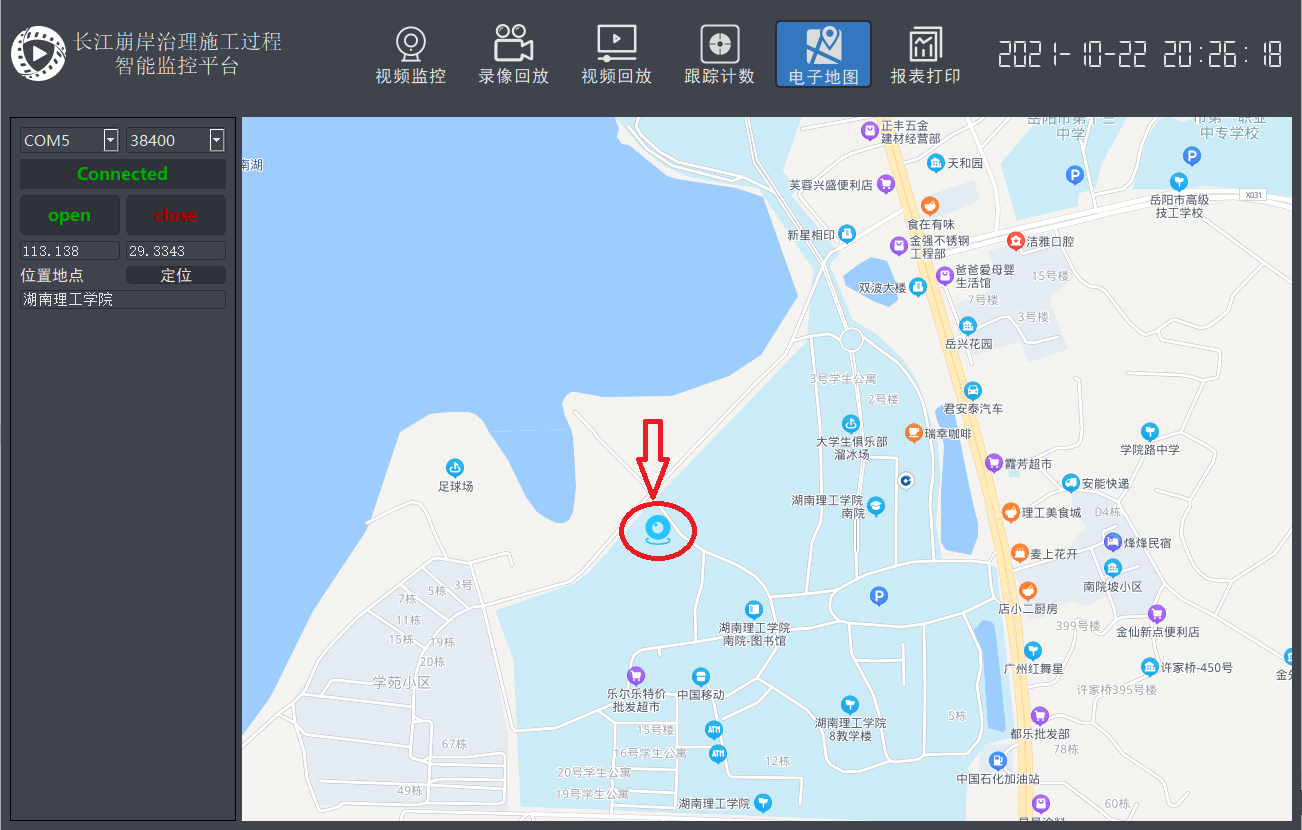
点击IMG_256识别按钮，选择要识别的施工视频，平台能够自动跟踪抛投过程，并将计数结果显示在右侧计数栏中，且物料类别会显示在物料框上，该例子显示为gabion（石笼）。



图13

## 4.7电子地图功能及操作

该界面功能主要是定位当前施工地点，同时将施工地点位置保存至数据库中。点击电子地图界面左上角IMG_256open按钮，之后点击IMG_256定位按钮即可开启定位功能，如图14所示。

图14

点击电子地图右上角的IMG_256混合按钮，即可显示带有街道的卫星定位影像，如图15所示。



图15

## 4.8报表打印功能及操作

该界面的功能主要是读取数据库中保存的施工信息，如施工时间，施工地点，施工关键帧，并打印文字报表或图像报表，点击查询所有IMG_256数据按钮，如图16所示，显示MySQL数据库中全部数据。

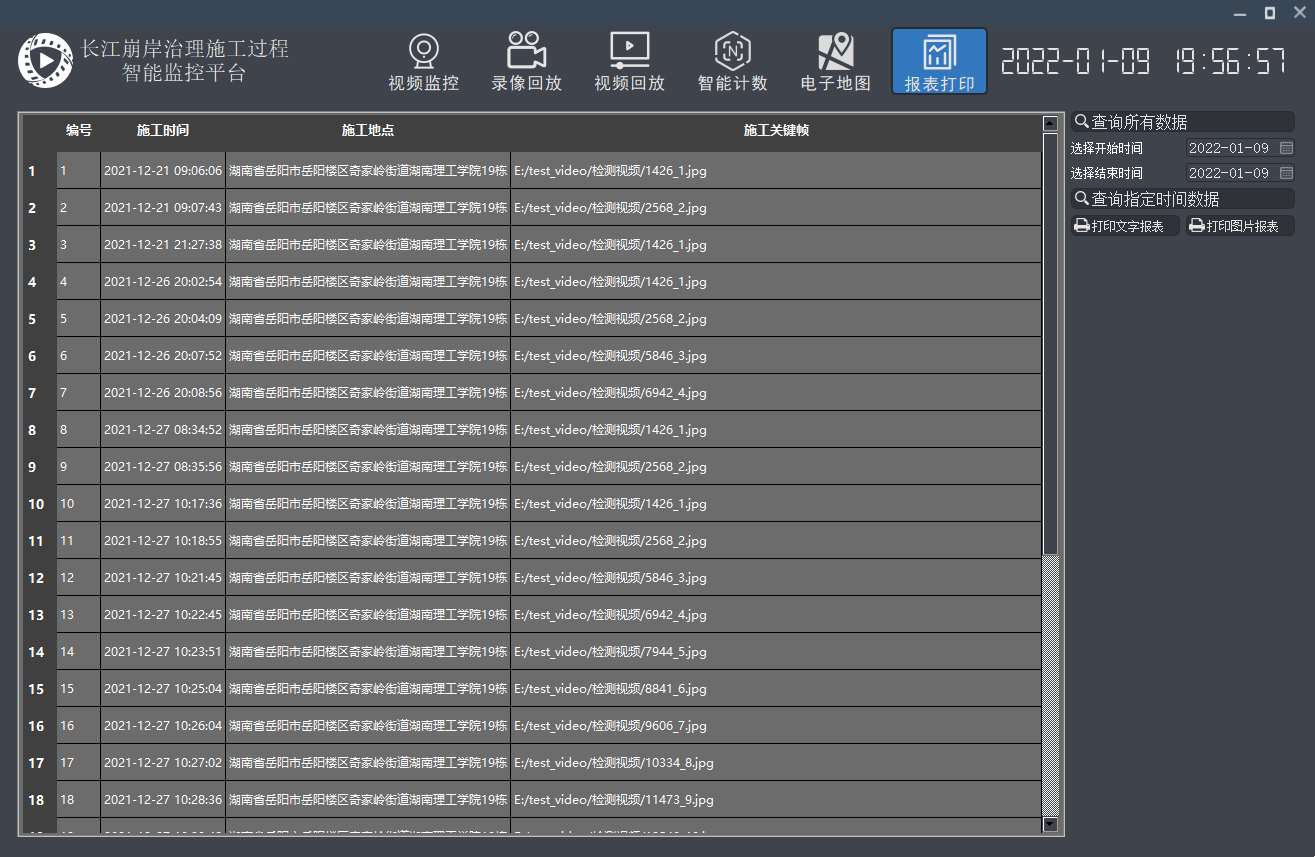


图16

点击选择开始时间和结束时间，可以查询指定时间（例：查找2021年12月20日至2021年12月22日）数据，如图17所示，查询结果如图18所示。

IMG_256

图17



图18

点击IMG_256打印文字报表按钮，即可预览要打印的内容，如图19所示，点击IMG_256打印按钮即可打印报表；点击IMG_256编辑按钮，即可编辑报表；点击IMG_256保存按钮，即可将报表保存至指定位置；点击IMG_256PDF按钮，即可将报表输出为PDF文件。



图19

点击IMG_256打印图片报表按钮，即可将指定时间数据打印成报表，结果如图20所示。

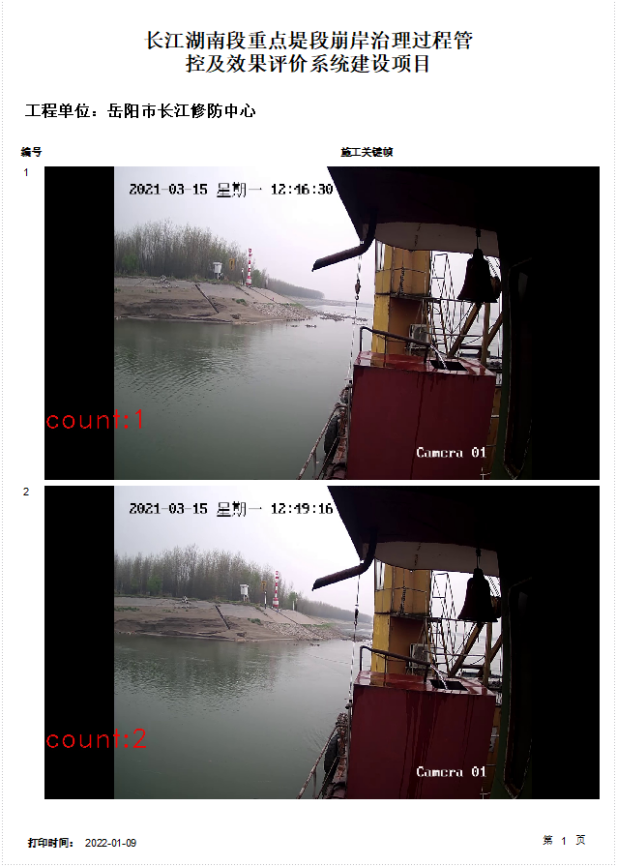


图20

# 五、源码附录

#include "widget.h"

#include "login.h"

#include <QIcon>

#include <QApplication>

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

{

QApplication a(argc, argv);

Login w;

w.show();

return a.exec();

}

#ifndef BASETITLEBAR\_H

#define BASETITLEBAR\_H

#include <QWidget>

#include <QLabel>

#include <QPushButton>

#include <QTimer>

enum ButtonType

{

MIN\_BUTTON = 0, // 最小化和关闭按钮;

MIN\_MAX\_BUTTON , // 最小化、最大化和关闭按钮;

ONLY\_CLOSE\_BUTTON // 只有关闭按钮;

};

class BaseTitleBar : public QWidget

{

Q\_OBJECT

public:

BaseTitleBar(QWidget \*parent = NULL);

~BaseTitleBar();

// 设置标题栏背景色;

void setBackgroundColor(int r, int g, int b);

// 设置标题栏图标;

void setTitleIcon(QString filePath);

// 设置标题内容;

void setTitleContent(QString titleContent);

// 设置标题栏长度;

void setTitleWidth(int width);

// 设置标题栏上按钮类型;

void setButtonType(ButtonType buttonType);

// 设置标题栏中的标题是否会滚动;具体可以看效果;

void setTitleRoll();

// 保存/获取 最大化前窗口的位置及大小;

void saveRestoreInfo(const QPoint point, const QSize size);

void getRestoreInfo(QPoint& point, QSize& size);

private:

void paintEvent(QPaintEvent \*event);

void mouseDoubleClickEvent(QMouseEvent \*event);

void mousePressEvent(QMouseEvent \*event);

void mouseMoveEvent(QMouseEvent \*event);

void mouseReleaseEvent(QMouseEvent \*event);

// 初始化控件;

void initControl();

// 信号槽的绑定;

void initConnections();

// 加载样式文件;

void loadStyleSheet(const QString &sheetName);

signals:

// 按钮触发的信号;

void signalButtonMinClicked();

void signalButtonRestoreClicked();

void signalButtonMaxClicked();

void signalButtonCloseClicked();

private slots:

// 按钮触发的槽;

void onButtonMinClicked();

void onButtonRestoreClicked();

void onButtonMaxClicked();

void onButtonCloseClicked();

void onRollTitle();

private:

QLabel\* m\_pIcon; // 标题栏图标;

QLabel\* m\_pTitleContent; // 标题栏内容;

QPushButton\* m\_pButtonMin; // 最小化按钮;

QPushButton\* m\_pButtonRestore; // 最大化还原按钮;

QPushButton\* m\_pButtonMax; // 最大化按钮;

QPushButton\* m\_pButtonClose; // 关闭按钮;

// 标题栏背景色;

int m\_colorR;

int m\_colorG;

int m\_colorB;

// 最大化，最小化变量;

QPoint m\_restorePos;

QSize m\_restoreSize;

// 移动窗口的变量;

bool m\_isPressed;

QPoint m\_startMovePos;

// 标题栏跑马灯效果时钟;

QTimer m\_titleRollTimer;

// 标题栏内容;

QString m\_titleContent;

// 按钮类型;

ButtonType m\_buttonType;

};

#endif // BASETITLEBAR\_H

#include "BaseTitleBar.h"

#include <QHBoxLayout>

#include <QPainter>

#include <QFile>

#include <QMouseEvent>

#include <QDebug>

#define BUTTON\_HEIGHT 30 // 按钮高度;

#define BUTTON\_WIDTH 30 // 按钮宽度;

#define TITLE\_HEIGHT 30 // 标题栏高度;

BaseTitleBar::BaseTitleBar(QWidget \*parent)

: QWidget(parent)

, m\_colorR(153)

, m\_colorG(153)

, m\_colorB(153)

, m\_isPressed(false)

, m\_buttonType(MIN\_MAX\_BUTTON)

{

// 初始化;

initControl();

initConnections();

loadStyleSheet("MyTitle");

}

BaseTitleBar::~BaseTitleBar()

{

}

// 初始化控件;

void BaseTitleBar::initControl()

{

m\_pIcon = new QLabel;

m\_pTitleContent = new QLabel;

m\_pButtonMin = new QPushButton;

m\_pButtonRestore = new QPushButton;

m\_pButtonMax = new QPushButton;

m\_pButtonClose = new QPushButton;

m\_pButtonMin->setFixedSize(QSize(BUTTON\_WIDTH, BUTTON\_HEIGHT));

m\_pButtonRestore->setFixedSize(QSize(BUTTON\_WIDTH, BUTTON\_HEIGHT));

m\_pButtonMax->setFixedSize(QSize(BUTTON\_WIDTH, BUTTON\_HEIGHT));

m\_pButtonClose->setFixedSize(QSize(BUTTON\_WIDTH, BUTTON\_HEIGHT));

m\_pTitleContent->setObjectName("TitleContent");

m\_pButtonMin->setObjectName("ButtonMin");

m\_pButtonRestore->setObjectName("ButtonRestore");

m\_pButtonMax->setObjectName("ButtonMax");

m\_pButtonClose->setObjectName("ButtonClose");

QHBoxLayout\* mylayout = new QHBoxLayout(this);

mylayout->addWidget(m\_pIcon);

mylayout->addWidget(m\_pTitleContent);

mylayout->addWidget(m\_pButtonMin);

mylayout->addWidget(m\_pButtonRestore);

mylayout->addWidget(m\_pButtonMax);

mylayout->addWidget(m\_pButtonClose);

mylayout->setContentsMargins(5, 0, 0, 0);

mylayout->setSpacing(0);

m\_pTitleContent->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Fixed);

this->setFixedHeight(TITLE\_HEIGHT);

this->setWindowFlags(Qt::FramelessWindowHint);

}

// 信号槽的绑定;

void BaseTitleBar::initConnections()

{

connect(m\_pButtonMin, SIGNAL(clicked()), this, SLOT(onButtonMinClicked()));

connect(m\_pButtonRestore, SIGNAL(clicked()), this, SLOT(onButtonRestoreClicked()));

connect(m\_pButtonMax, SIGNAL(clicked()), this, SLOT(onButtonMaxClicked()));

connect(m\_pButtonClose, SIGNAL(clicked()), this, SLOT(onButtonCloseClicked()));

}

// 设置标题栏背景色,在paintEvent事件中进行绘制标题栏背景色;

//在构造函数中给了默认值，可以外部设置颜色值改变标题栏背景色;

void BaseTitleBar::setBackgroundColor(int r, int g, int b)

{

m\_colorR = r;

m\_colorG = g;

m\_colorB = b;

// 重新绘制（调用paintEvent事件）;

update();

}

// 设置标题栏图标;

void BaseTitleBar::setTitleIcon(QString filePath)

{

QPixmap titleIcon(filePath);

m\_pIcon->setPixmap(titleIcon.scaled(25 , 25));

}

// 设置标题内容;

void BaseTitleBar::setTitleContent(QString titleContent)

{

m\_pTitleContent->setText(titleContent);

m\_titleContent = titleContent;

}

// 设置标题栏长度;

void BaseTitleBar::setTitleWidth(int width)

{

this->setFixedWidth(width);

}

// 设置标题栏上按钮类型;

// 由于不同窗口标题栏上的按钮都不一样，所以可以自定义标题栏中的按钮;

// 这里提供了四个按钮，分别为最小化、还原、最大化、关闭按钮，如果需要其他按钮可自行添加设置;

void BaseTitleBar::setButtonType(ButtonType buttonType)

{

m\_buttonType = buttonType;

switch (buttonType)

{

case MIN\_BUTTON:

{

m\_pButtonRestore->setVisible(false);

m\_pButtonMax->setVisible(false);

}

break;

case MIN\_MAX\_BUTTON:

{

m\_pButtonRestore->setVisible(false);

}

break;

case ONLY\_CLOSE\_BUTTON:

{

m\_pButtonMin->setVisible(false);

m\_pButtonRestore->setVisible(false);

m\_pButtonMax->setVisible(false);

}

break;

default:

break;

}

}

// 设置标题栏中的标题是否会自动滚动，跑马灯的效果;

// 一般情况下标题栏中的标题内容是不滚动的，但是既然自定义就看自己需要嘛，想怎么设计就怎么搞O(∩\_∩)O！

void BaseTitleBar::setTitleRoll()

{

connect(&m\_titleRollTimer, SIGNAL(timeout()), this, SLOT(onRollTitle()));

m\_titleRollTimer.start(200);

}

// 保存窗口最大化前窗口的位置以及大小;

void BaseTitleBar::saveRestoreInfo(const QPoint point, const QSize size)

{

m\_restorePos = point;

m\_restoreSize = size;

}

// 获取窗口最大化前窗口的位置以及大小;

void BaseTitleBar::getRestoreInfo(QPoint& point, QSize& size)

{

point = m\_restorePos;

size = m\_restoreSize;

}

// 绘制标题栏背景色;

void BaseTitleBar::paintEvent(QPaintEvent \*event)

{

//设置背景色;

QPainter painter(this);

QPainterPath pathBack;

pathBack.setFillRule(Qt::WindingFill);

pathBack.addRoundedRect(QRect(0, 0, this->width(), this->height()), 3, 3);

QRect temp\_rect(0, 0+this->height()/2, this->width(), this->height()/2);

pathBack.addRect(temp\_rect);

painter.setRenderHint(QPainter::SmoothPixmapTransform, true);

painter.fillPath(pathBack, QBrush(QColor(m\_colorR, m\_colorG, m\_colorB)));

// 当窗口最大化或者还原后，窗口长度变了，标题栏的长度应当一起改变;

if (this->width() != this->parentWidget()->width())

{

this->setFixedWidth(this->parentWidget()->width());

}

QWidget::paintEvent(event);

}

// 双击响应事件，主要是实现双击标题栏进行最大化和最小化操作;

void BaseTitleBar::mouseDoubleClickEvent(QMouseEvent \*event)

{

// 只有存在最大化、还原按钮时双击才有效;

if (m\_buttonType == MIN\_MAX\_BUTTON)

{

// 通过最大化按钮的状态判断当前窗口是处于最大化还是原始大小状态;

// 或者通过单独设置变量来表示当前窗口状态;

if (m\_pButtonMax->isVisible())

{

onButtonMaxClicked();

}

else

{

onButtonRestoreClicked();

}

}

return QWidget::mouseDoubleClickEvent(event);

}

// 以下通过mousePressEvent、mouseMoveEvent、mouseReleaseEvent三个事件实现了鼠标拖动标题栏移动窗口的效果;

void BaseTitleBar::mousePressEvent(QMouseEvent \*event)

{

if (m\_buttonType == MIN\_MAX\_BUTTON)

{

// 在窗口最大化时禁止拖动窗口;

if (m\_pButtonMax->isVisible())

{

m\_isPressed = true;

m\_startMovePos = event->globalPos();

}

}

else

{

m\_isPressed = true;

m\_startMovePos = event->globalPos();

}

return QWidget::mousePressEvent(event);

}

void BaseTitleBar::mouseMoveEvent(QMouseEvent \*event)

{

if (m\_isPressed)

{

QPoint movePoint = event->globalPos() - m\_startMovePos;

QPoint widgetPos = this->parentWidget()->pos();

m\_startMovePos = event->globalPos();

this->parentWidget()->move(widgetPos.x() + movePoint.x(), widgetPos.y() + movePoint.y());

}

return QWidget::mouseMoveEvent(event);

}

void BaseTitleBar::mouseReleaseEvent(QMouseEvent \*event)

{

m\_isPressed = false;

return QWidget::mouseReleaseEvent(event);

}

// 加载本地样式文件;

// 可以将样式直接写在文件中，程序运行时直接加载进来;

void BaseTitleBar::loadStyleSheet(const QString &sheetName)

{

QFile file(":/" + sheetName + ".css");

file.open(QFile::ReadOnly);

if (file.isOpen())

{

QString styleSheet = this->styleSheet();

styleSheet += QLatin1String(file.readAll());

this->setStyleSheet(styleSheet);

}

}

// 以下为按钮操作响应的槽;

void BaseTitleBar::onButtonMinClicked()

{

emit signalButtonMinClicked();

}

void BaseTitleBar::onButtonRestoreClicked()

{

m\_pButtonRestore->setVisible(false);

m\_pButtonMax->setVisible(true);

emit signalButtonRestoreClicked();

}

void BaseTitleBar::onButtonMaxClicked()

{

m\_pButtonMax->setVisible(false);

m\_pButtonRestore->setVisible(true);

emit signalButtonMaxClicked();

}

void BaseTitleBar::onButtonCloseClicked()

{

emit signalButtonCloseClicked();

}

// 该方法主要是让标题栏中的标题显示为滚动的效果;

void BaseTitleBar::onRollTitle()

{

static int nPos = 0;

QString titleContent = m\_titleContent;

// 当截取的位置比字符串长时，从头开始;

if (nPos > titleContent.length())

nPos = 0;

m\_pTitleContent->setText(titleContent.mid(nPos));

nPos++;

}

#ifndef BASEWINDOW\_H

#define BASEWINDOW\_H

#include <QWidget>

#include <QMainWindow>

#include "BaseTitleBar.h"

class BaseWindow : public QWidget

{

Q\_OBJECT

public:

BaseWindow(QWidget \*parent = 0);

~BaseWindow();

private:

void initTitleBar();

void paintEvent(QPaintEvent \*event);

void loadStyleSheet(const QString &sheetName);

private slots:

void onButtonMinClicked();

void onButtonRestoreClicked();

void onButtonMaxClicked();

void onButtonCloseClicked();

protected:

BaseTitleBar\* m\_titleBar;

};

#endif // BASEWINDOW\_H

#include "BaseWindow.h"

#include <QDesktopWidget>

#include <QApplication>

#include <QPainter>

#include <QFile>

BaseWindow::BaseWindow(QWidget \*parent)

: QWidget(parent)

{

// FramelessWindowHint属性设置窗口去除边框;

// WindowMinimizeButtonHint 属性设置在窗口最小化时，点击任务栏窗口可以显示出原窗口;

this->setWindowFlags(Qt::FramelessWindowHint | Qt::WindowMinimizeButtonHint);

// 设置窗口背景透明;

// setAttribute(Qt::WA\_TranslucentBackground);

// 初始化标题栏;

initTitleBar();

}

BaseWindow::~BaseWindow()

{

}

void BaseWindow::initTitleBar()

{

m\_titleBar = new BaseTitleBar(this);

m\_titleBar->move(0, 0);

connect(m\_titleBar, SIGNAL(signalButtonMinClicked()), this, SLOT(onButtonMinClicked()));

connect(m\_titleBar, SIGNAL(signalButtonRestoreClicked()), this, SLOT(onButtonRestoreClicked()));

connect(m\_titleBar, SIGNAL(signalButtonMaxClicked()), this, SLOT(onButtonMaxClicked()));

connect(m\_titleBar, SIGNAL(signalButtonCloseClicked()), this, SLOT(onButtonCloseClicked()));

}

void BaseWindow::paintEvent(QPaintEvent\* event)

{

// Q\_UNUSED(event);

// QStyleOption opt;

// opt.init(this);

// QPainter painter(this);

// style()->drawPrimitive(QStyle::PE\_Widget, &opt, &painter, this);

//设置背景色;

QPainter painter(this);

QPainterPath pathBack;

pathBack.setFillRule(Qt::WindingFill);

pathBack.addRoundedRect(QRect(0, 0, this->width(), this->height()), 3, 3);

painter.setRenderHint(QPainter::SmoothPixmapTransform, true);

painter.fillPath(pathBack, QBrush(QColor(63, 67, 76)));

return QWidget::paintEvent(event);

}

void BaseWindow::loadStyleSheet(const QString &sheetName)

{

QFile file(":/Resources/" + sheetName + ".css");

file.open(QFile::ReadOnly);

if (file.isOpen())

{

QString styleSheet = this->styleSheet();

styleSheet += QLatin1String(file.readAll());

this->setStyleSheet(styleSheet);

}

}

void BaseWindow::onButtonMinClicked()

{

if (Qt::Tool == (windowFlags() & Qt::Tool))

{

hide(); //设置了Qt::Tool 如果调用showMinimized()则窗口就销毁了？？？

}

else

{

showMinimized();

}

}

void BaseWindow::onButtonRestoreClicked()

{

QPoint windowPos;

QSize windowSize;

m\_titleBar->getRestoreInfo(windowPos, windowSize);

this->setGeometry(QRect(windowPos, windowSize));

}

void BaseWindow::onButtonMaxClicked()

{

m\_titleBar->saveRestoreInfo(this->pos(), QSize(this->width(), this->height()));

QRect desktopRect = QApplication::desktop()->availableGeometry();

QRect FactRect = QRect(desktopRect.x() - 3, desktopRect.y() - 3, desktopRect.width() + 3, desktopRect.height() + 3);

setGeometry(FactRect);

}

void BaseWindow::onButtonCloseClicked()

{

close();

}

#ifndef CUSTOMSLIDER\_H

#define CUSTOMSLIDER\_H

#include <QSlider>

#include <QMouseEvent>

#include <QCoreApplication>

class CustomSlider : public QSlider

{

Q\_OBJECT

public:

CustomSlider(QWidget \*parent = 0) : QSlider(parent)

{

}

protected:

void mousePressEvent(QMouseEvent \*ev);//重写QSlider的mousePressEvent事件

signals:

void costomSliderClicked();//自定义的鼠标单击信号，用于捕获并处理

};

#endif // CUSTOMSLIDER\_H

#include "customslider.h"

void CustomSlider::mousePressEvent(QMouseEvent \*ev)

{

//注意应先调用父类的鼠标点击处理事件，这样可以不影响拖动的情况

QSlider::mousePressEvent(ev);

//获取鼠标的位置，这里并不能直接从ev中取值（因为如果是拖动的话，鼠标开始点击的位置没有意义了）

double pos = ev->pos().x() / (double)width();

setValue(pos \* (maximum() - minimum()) + minimum());

//发送自定义的鼠标单击信号

emit costomSliderClicked();

}

#undef slots

#include<torch/script.h>

#include <torch/torch.h>

#define slots Q\_SLOTS

#include <opencv2/opencv.hpp>

#include <iostream>

#include <ctime>

/\*\*

\* ImageResizeData 图片处理过后保存图片的数据结构

\*/

class ImageResizeData

{

public:

// 添加处理过后的图片

void setImg(cv::Mat img);

// 获取处理过后的图片

cv::Mat getImg();

// 当原始图片宽高比大于处理过后图片宽高比时此函数返回 true

bool isW();

// 当原始图片高宽比大于处理过后图片高宽比时此函数返回 true

bool isH();

// 添加处理之后图片的宽

void setWidth(int width);

// 获取处理之后图片的宽

int getWidth();

// 添加处理之后图片的高

void setHeight(int height);

// 获取处理之后图片的高

int getHeight();

// 添加原始图片的宽

void setW(int w);

// 获取原始图片的宽

int getW();

// 添加原始图片的高

void setH(int h);

// 获取原始图片的高

int getH();

// 添加从原始图片到处理过后图片所添加黑边大小

void setBorder(int border);

// 获取从原始图片到处理过后图片所添加黑边大小

int getBorder();

private:

// 处理过后图片高

int height;

// 处理过后图片宽

int width;

// 原始图片宽

int w;

// 原始图片高

int h;

// 从原始图片到处理图片所添加的黑边大小

int border;

// 处理过后的图片

cv::Mat img;

};

/\*\*

\* YoloV5 的实现类

\*/

class Detector

{

public:

/\*\*

\* 构造函数

\* @param ptFile yoloV5 pt文件路径

\* @param isCuda 是否使用 cuda 默认不起用

\* @param height yoloV5 训练时图片的高

\* @param width yoloV5 训练时图片的宽

\* @param confThres 非极大值抑制中的 scoreThresh

\* @param iouThres 非极大值抑制中的 iouThresh

\*/

Detector(std::string ptFile, bool isCuda = false, int height = 640, int width = 640, float confThres = 0.7, float iouThres = 0.45);

/\*\*

\* 预测函数

\* @param data 语言预测的数据格式 (batch, rgb, height, width)

\*/

std::vector<torch::Tensor> prediction(torch::Tensor data);

/\*\*

\* 预测函数

\* @param filePath 需要预测的图片路径

\*/

std::vector<torch::Tensor> prediction(std::string filePath);

/\*\*

\* 预测函数

\* @param img 需要预测的图片

\*/

std::vector<torch::Tensor> prediction(cv::Mat img);

/\*\*

\* 预测函数

\* @param imgs 需要预测的图片集合

\*/

std::vector<torch::Tensor> prediction(std::vector <cv::Mat> imgs);

/\*\*

\* 改变图片大小的函数

\* @param img 原始图片

\* @param height 要处理成的图片的高

\* @param width 要处理成的图片的宽

\* @return 封装好的处理过后图片数据结构

\*/

static ImageResizeData resize(cv::Mat img, int height, int width);

/\*\*

\* 改变图片大小的函数

\* @param img 原始图片

\* @return 封装好的处理过后图片数据结构

\*/

ImageResizeData resize(cv::Mat img);

/\*\*

\* 改变图片大小的函数

\* @param imgs 原始图片集合

\* @param height 要处理成的图片的高

\* @param width 要处理成的图片的宽

\* @return 封装好的处理过后图片数据结构

\*/

static std::vector<ImageResizeData> resize(std::vector <cv::Mat> imgs, int height, int width);

/\*\*

\* 改变图片大小的函数

\* @param imgs 原始图片集合

\* @return 封装好的处理过后图片数据结构

\*/

std::vector<ImageResizeData> resize(std::vector <cv::Mat> imgs);

/\*\*

\* 根据输出结果在给定图片中画出框

\* @param imgs 原始图片集合

\* @param rectangles 通过预测函数处理好的结果

\* @param labels 类别标签

\* @param thickness 线宽

\* @return 画好框的图片

\*/

std::vector<cv::Mat> drawRectangle(std::vector<cv::Mat> imgs, std::vector<torch::Tensor> rectangles, std::map<int, std::string> labels, int thickness = 2);

/\*\*

\* 根据输出结果在给定图片中画出框

\* @param imgs 原始图片集合

\* @param rectangles 通过预测函数处理好的结果

\* @param thickness 线宽

\* @return 画好框的图片

\*/

std::vector<cv::Mat> drawRectangle(std::vector<cv::Mat> imgs, std::vector<torch::Tensor> rectangles, int thickness = 2);

/\*\*

\* 根据输出结果在给定图片中画出框

\* @param imgs 原始图片集合

\* @param rectangles 通过预测函数处理好的结果

\* @param colors 每种类型对应颜色

\* @param labels 类别标签

\* @return 画好框的图片

\*/

std::vector<cv::Mat> drawRectangle(std::vector<cv::Mat> imgs, std::vector<torch::Tensor> rectangles, std::map<int, cv::Scalar> colors, std::map<int, std::string> labels, int thickness = 2);

/\*\*

\* 根据输出结果在给定图片中画出框

\* @param img 原始图片

\* @param rectangle 通过预测函数处理好的结果

\* @param thickness 线宽

\* @return 画好框的图片

\*/

cv::Mat drawRectangle(cv::Mat img, torch::Tensor rectangle, int thickness = 2);

/\*\*

\* 根据输出结果在给定图片中画出框

\* @param img 原始图片

\* @param rectangle 通过预测函数处理好的结果

\* @param labels 类别标签

\* @param thickness 线宽

\* @return 画好框的图片

\*/

cv::Mat drawRectangle(cv::Mat img, torch::Tensor rectangle, std::map<int, std::string> labels, int thickness = 2);

/\*\*

\* 根据输出结果在给定图片中画出框

\* @param img 原始图片

\* @param rectangle 通过预测函数处理好的结果

\* @param colos 每种类型对应颜色

\* @param labels 类别标签

\* @param thickness 线宽

\* @return 画好框的图片

\*/

cv::Mat drawRectangle(cv::Mat img, torch::Tensor rectangle, std::map<int, cv::Scalar> colors, std::map<int, std::string> labels, int thickness = 2);

/\*\*

\* 用于判断给定数据是否存在预测

\* @param clazz 通过预测函数处理好的结果

\* @return 如果图片中存在给定某一种分类返回 true

\*/

bool existencePrediction(torch::Tensor clazz);

/\*\*

\* 用于判断给定数据是否存在预测

\* @param classs 通过预测函数处理好的结果

\* @return 如果图片集合中存在给定某一种分类返回 true

\*/

bool existencePrediction(std::vector<torch::Tensor> classs);

private:

// 是否启用 cuda

bool isCuda;

// 非极大值抑制中的第一步数据清理

float confThres;

// 非极大值抑制中 iou

float iouThres;

// 模型所需要的图片的高

float height;

// 模型所需要的图片的宽

float width;

// 画框颜色 map

std::map<int, cv::Scalar> mainColors;

// 模型

torch::jit::script::Module model;

// 随机获取一种颜色

cv::Scalar getRandScalar();

// 图片通道转换为 rgb

cv::Mat img2RGB(cv::Mat img);

// 图片变为 Tensor

torch::Tensor img2Tensor(cv::Mat img);

// (center\_x center\_y w h) to (left, top, right, bottom)

torch::Tensor xywh2xyxy(torch::Tensor x);

// 非极大值抑制算法

torch::Tensor nms(torch::Tensor bboxes, torch::Tensor scores, float thresh);

// 预测出来的框根据原始图片还原算法

std::vector<torch::Tensor> sizeOriginal(std::vector<torch::Tensor> result, std::vector<ImageResizeData> imgRDs);

// 非极大值抑制算法整体

std::vector<torch::Tensor> non\_max\_suppression(torch::Tensor preds, float confThres = 0.25, float iouThres = 0.45);

};

#include "detector.h"

Detector::Detector(std::string ptFile, bool isCuda, int height, int width, float confThres, float iouThres)

{

model = torch::jit::load(ptFile);

if (isCuda) {

model.to(torch::kCUDA);

}

this->height = height;

this->width = width;

this->isCuda = isCuda;

this->iouThres = iouThres;

this->confThres = confThres;

model.eval();

unsigned seed = time(0);

std::srand(seed);

}

std::vector<torch::Tensor> Detector::non\_max\_suppression(torch::Tensor prediction, float confThres, float iouThres)

{

torch::Tensor xc = prediction.select(2, 4) > confThres;

int maxWh = 4096;

int maxNms = 30000;

std::vector<torch::Tensor> output;

for (int i = 0; i < prediction.size(0); i++)

{

output.push\_back(torch::zeros({ 0, 6 }));

}

for (int i = 0; i < prediction.size(0); i++)

{

torch::Tensor x = prediction[i];

x = x.index\_select(0, torch::nonzero(xc[i]).select(1, 0));

if (x.size(0) == 0) continue;

x.slice(1, 5, x.size(1)).mul\_(x.slice(1, 4, 5));

torch::Tensor box = xywh2xyxy(x.slice(1, 0, 4));

std::tuple<torch::Tensor, torch::Tensor> max\_tuple = torch::max(x.slice(1, 5, x.size(1)), 1, true);

x = torch::cat({ box, std::get<0>(max\_tuple), std::get<1>(max\_tuple) }, 1);

x = x.index\_select(0, torch::nonzero(std::get<0>(max\_tuple) > confThres).select(1, 0));

int n = x.size(0);

if (n == 0)

{

continue;

}

else if (n > maxNms)

{

x = x.index\_select(0, x.select(1, 4).argsort(0, true).slice(0, 0, maxNms));

}

torch::Tensor c = x.slice(1, 5, 6) \* maxWh;

torch::Tensor boxes = x.slice(1, 0, 4) + c, scores = x.select(1, 4);

torch::Tensor ix = nms(boxes, scores, iouThres).to(x.device());

output[i] = x.index\_select(0, ix).cpu();

}

return output;

}

cv::Scalar Detector::getRandScalar()

{

return cv::Scalar(std::rand() % 256, std::rand() % 256, std::rand() % 256);

}

cv::Mat Detector::img2RGB(cv::Mat img)

{

int imgC = img.channels();

if (imgC == 1)

{

cv::cvtColor(img, img, cv::COLOR\_GRAY2RGB);

}

else

{

cv::cvtColor(img, img, cv::COLOR\_BGR2RGB);

}

return img;

}

torch::Tensor Detector::img2Tensor(cv::Mat img)

{

torch::Tensor data = torch::from\_blob(img.data, { (int)height, (int)width, 3 }, torch::kByte);

data = data.permute({ 2, 0, 1 });

data = data.toType(torch::kFloat);

data = data.div(255);

data = data.unsqueeze(0);

return data;

}

torch::Tensor Detector::xywh2xyxy(torch::Tensor x)

{

torch::Tensor y = x.clone();

y.select(1, 0) = x.select(1, 0) - x.select(1, 2) / 2;

y.select(1, 1) = x.select(1, 1) - x.select(1, 3) / 2;

y.select(1, 2) = x.select(1, 0) + x.select(1, 2) / 2;

y.select(1, 3) = x.select(1, 1) + x.select(1, 3) / 2;

return y;

}

torch::Tensor Detector::nms(torch::Tensor bboxes, torch::Tensor scores, float thresh)

{

auto x1 = bboxes.select(1, 0);

auto y1 = bboxes.select(1, 1);

auto x2 = bboxes.select(1, 2);

auto y2 = bboxes.select(1, 3);

auto areas = (x2 - x1) \* (y2 - y1);

auto tuple\_sorted = scores.sort(0, true);

auto order = std::get<1>(tuple\_sorted);

std::vector<int> keep;

while (order.numel() > 0) {

if (order.numel() == 1) {

auto i = order.item();

keep.push\_back(i.toInt());

break;

}

else {

auto i = order[0].item();

keep.push\_back(i.toInt());

}

auto order\_mask = order.narrow(0, 1, order.size(-1) - 1);

auto xx1 = x1.index({ order\_mask }).clamp(x1[keep.back()].item().toFloat(), 1e10);

auto yy1 = y1.index({ order\_mask }).clamp(y1[keep.back()].item().toFloat(), 1e10);

auto xx2 = x2.index({ order\_mask }).clamp(0, x2[keep.back()].item().toFloat());

auto yy2 = y2.index({ order\_mask }).clamp(0, y2[keep.back()].item().toFloat());

auto inter = (xx2 - xx1).clamp(0, 1e10) \* (yy2 - yy1).clamp(0, 1e10);

auto iou = inter / (areas[keep.back()] + areas.index({ order.narrow(0,1,order.size(-1) - 1) }) - inter);

auto idx = (iou <= thresh).nonzero().squeeze();

if (idx.numel() == 0) {

break;

}

order = order.index({ idx + 1 });

}

return torch::tensor(keep);

}

std::vector<torch::Tensor> Detector::sizeOriginal(std::vector<torch::Tensor> result, std::vector<ImageResizeData> imgRDs)

{

std::vector<torch::Tensor> resultOrg;

for (int i = 0; i < result.size(); i++)

{

torch::Tensor data = result[i];

ImageResizeData imgRD = imgRDs[i];

for (int j = 0; j < data.size(0); j++)

{

torch::Tensor tensor = data.select(0, j);

// (left, top, right, bottom)

if (imgRD.isW())

{

tensor[1] -= imgRD.getBorder();

tensor[3] -= imgRD.getBorder();

tensor[0] \*= (float)imgRD.getW() / (float)imgRD.getWidth();

tensor[2] \*= (float)imgRD.getW() / (float)imgRD.getWidth();

tensor[1] \*= (float)imgRD.getH() / (float)(imgRD.getHeight() - 2 \* imgRD.getBorder());

tensor[3] \*= (float)imgRD.getH() / (float)(imgRD.getHeight() - 2 \* imgRD.getBorder());

}

else

{

tensor[0] -= imgRD.getBorder();

tensor[2] -= imgRD.getBorder();

tensor[1] \*= (float)imgRD.getH() / (float)imgRD.getHeight();

tensor[3] \*= (float)imgRD.getH() / (float)imgRD.getHeight();

tensor[0] \*= (float)imgRD.getW() / (float)(imgRD.getWidth() - 2 \* imgRD.getBorder());

tensor[2] \*= (float)imgRD.getW() / (float)(imgRD.getWidth() - 2 \* imgRD.getBorder());

}

// 加了黑边之后预测结果可能在黑边上，就会造成结果为负数

for (int k = 0; k < 4; k++)

{

if (tensor[k].item().toFloat() < 0)

{

tensor[k] = 0;

}

}

}

resultOrg.push\_back(data);

}

return resultOrg;

}

std::vector<torch::Tensor> Detector::prediction(torch::Tensor data)

{

if (!data.is\_cuda() && this->isCuda)

{

data = data.cuda();

}

if (data.is\_cuda() && !this->isCuda)

{

data = data.cpu();

}

torch::Tensor pred = model.forward({ data }).toTuple()->elements()[0].toTensor();

return non\_max\_suppression(pred, confThres, iouThres);

}

std::vector<torch::Tensor> Detector::prediction(std::string filePath)

{

cv::Mat img = cv::imread(filePath);

return prediction(img);

}

std::vector<torch::Tensor> Detector::prediction(cv::Mat img)

{

ImageResizeData imgRD = resize(img);

cv::Mat reImg = img2RGB(imgRD.getImg());

torch::Tensor data = img2Tensor(reImg);

std::vector<torch::Tensor> result = prediction(data);

std::vector<ImageResizeData> imgRDs;

imgRDs.push\_back(imgRD);

return sizeOriginal(result, imgRDs);

}

std::vector<torch::Tensor> Detector::prediction(std::vector<cv::Mat> imgs)

{

std::vector<ImageResizeData> imageRDs;

std::vector<torch::Tensor> datas;

for (int i = 0; i < imgs.size(); i++)

{

ImageResizeData imgRD = resize(imgs[i]);

imageRDs.push\_back(imgRD);

cv::Mat img = img2RGB(imgRD.getImg());

datas.push\_back(img2Tensor(img));

}

torch::Tensor data = torch::cat(datas, 0);

std::vector<torch::Tensor> result = prediction(data);

return sizeOriginal(result, imageRDs);

}

ImageResizeData Detector::resize(cv::Mat img, int height, int width)

{

ImageResizeData imgResizeData;

int w = img.cols, h = img.rows;

imgResizeData.setH(h);

imgResizeData.setW(w);

imgResizeData.setHeight(height);

imgResizeData.setWidth(width);

bool isW = (float)w / (float)h > (float)width / (float)height;

cv::resize(img, img, cv::Size(

isW ? width : (int)((float)height / (float)h \* w),

isW ? (int)((float)width / (float)w \* h) : height));

w = img.cols, h = img.rows;

if (isW)

{

imgResizeData.setBorder((height - h) / 2);

cv::copyMakeBorder(img, img, (height - h) / 2, height - h - (height - h) / 2, 0, 0, cv::BORDER\_CONSTANT);

}

else

{

imgResizeData.setBorder((width - w) / 2);

cv::copyMakeBorder(img, img, 0, 0, (width - w) / 2, width - w - (width - w) / 2, cv::BORDER\_CONSTANT);

}

imgResizeData.setImg(img);

return imgResizeData;

}

ImageResizeData Detector::resize(cv::Mat img)

{

return Detector::resize(img, height, width);

}

std::vector<ImageResizeData> Detector::resize(std::vector<cv::Mat> imgs, int height, int width)

{

std::vector<ImageResizeData> imgRDs;

for (int i = 0; i < imgs.size(); i++)

{

imgRDs.push\_back(Detector::resize(imgs[i], height, width));

}

return imgRDs;

}

std::vector<ImageResizeData> Detector::resize(std::vector<cv::Mat> imgs)

{

return Detector::resize(imgs, height, width);

}

std::vector<cv::Mat> Detector::drawRectangle(std::vector<cv::Mat> imgs, std::vector<torch::Tensor> rectangles, std::map<int, std::string> labels, int thickness)

{

std::map<int, cv::Scalar> colors;

return drawRectangle(imgs, rectangles, colors, labels, thickness);

}

std::vector<cv::Mat> Detector::drawRectangle(std::vector<cv::Mat> imgs, std::vector<torch::Tensor> rectangles, int thickness)

{

std::map<int, cv::Scalar> colors;

std::map<int, std::string> labels;

return drawRectangle(imgs, rectangles, colors, labels, thickness);

}

std::vector<cv::Mat> Detector::drawRectangle(std::vector<cv::Mat> imgs, std::vector<torch::Tensor> rectangles, std::map<int, cv::Scalar> colors, std::map<int, std::string> labels, int thickness)

{

std::vector<cv::Mat> results;

for (int i = 0; i < imgs.size(); i++)

{

results.push\_back(drawRectangle(imgs[i], rectangles[i], colors, labels, thickness));

}

return results;

}

cv::Mat Detector::drawRectangle(cv::Mat img, torch::Tensor rectangle, int thickness)

{

std::map<int, cv::Scalar> colors;

std::map<int, std::string> labels;

return drawRectangle(img, rectangle, colors, labels, thickness);

}

cv::Mat Detector::drawRectangle(cv::Mat img, torch::Tensor rectangle, std::map<int, std::string> labels, int thickness)

{

std::map<int, cv::Scalar> colors;

return drawRectangle(img, rectangle, colors, labels, thickness);

}

cv::Mat Detector::drawRectangle(cv::Mat img, torch::Tensor rectangle, std::map<int, cv::Scalar> colors, std::map<int, std::string> labels, int thickness)

{

std::map<int, cv::Scalar>::iterator it;

std::map<int, std::string>::iterator labelIt;

for (int i = 0; i < rectangle.size(0); i++)

{

int clazz = rectangle[i][5].item().toInt();

it = colors.find(clazz);

cv::Scalar color = NULL;

if (it == colors.end())

{

it = mainColors.find(clazz);

if (it == mainColors.end())

{

color = getRandScalar();

mainColors.insert(std::pair<int, cv::Scalar>(clazz, color));

}

else

{

color = it->second;

}

}

else

{

color = it->second;

}

cv::rectangle(img, cv::Point(rectangle[i][0].item().toInt(), rectangle[i][1].item().toInt()), cv::Point(rectangle[i][2].item().toInt(), rectangle[i][3].item().toInt()), color, thickness);

labelIt = labels.find(clazz);

std::ostringstream oss;

if (labelIt != labels.end())

{

oss << labelIt->second << " ";

}

oss << rectangle[i][4].item().toFloat();

std::string label = oss.str();

cv::putText(img, label, cv::Point(rectangle[i][0].item().toInt(), rectangle[i][1].item().toInt()), cv::FONT\_HERSHEY\_PLAIN, 1, color, thickness);

}

return img;

}

bool Detector::existencePrediction(torch::Tensor clazz)

{

return clazz.size(0) > 0 ? true : false;

}

bool Detector::existencePrediction(std::vector<torch::Tensor> classs)

{

for (int i = 0; i < classs.size(); i++)

{

if (existencePrediction(classs[i]))

{

return true;

}

}

return false;

}

void ImageResizeData::setImg(cv::Mat img)

{

this->img = img;

}

cv::Mat ImageResizeData::getImg()

{

return img;

}

bool ImageResizeData::isW()

{

return (float)w / (float)h > (float)width / (float)height;

}

bool ImageResizeData::isH()

{

return (float)h / (float)w > (float)height / (float)width;

}

void ImageResizeData::setWidth(int width)

{

this->width = width;

}

int ImageResizeData::getWidth()

{

return width;

}

void ImageResizeData::setHeight(int height)

{

this->height = height;

}

int ImageResizeData::getHeight()

{

return height;

}

void ImageResizeData::setW(int w)

{

this->w = w;

}

int ImageResizeData::getW()

{

return w;

}

void ImageResizeData::setH(int h)

{

this->h = h;

}

int ImageResizeData::getH()

{

return h;

}

void ImageResizeData::setBorder(int border)

{

this->border = border;

}

int ImageResizeData::getBorder()

{

return border;

}

#include <iostream>

#include <vector>

using namespace std;

class HungarianAlgorithm

{

public:

HungarianAlgorithm();

~HungarianAlgorithm();

double Solve(vector<vector<double> >& DistMatrix, vector<int>& Assignment);

private:

void assignmentoptimal(int \*assignment, double \*cost, double \*distMatrix, int nOfRows, int nOfColumns);

void buildassignmentvector(int \*assignment, bool \*starMatrix, int nOfRows, int nOfColumns);

void computeassignmentcost(int \*assignment, double \*cost, double \*distMatrix, int nOfRows);

void step2a(int \*assignment, double \*distMatrix, bool \*starMatrix, bool \*newStarMatrix, bool \*primeMatrix, bool \*coveredColumns, bool \*coveredRows, int nOfRows, int nOfColumns, int minDim);

void step2b(int \*assignment, double \*distMatrix, bool \*starMatrix, bool \*newStarMatrix, bool \*primeMatrix, bool \*coveredColumns, bool \*coveredRows, int nOfRows, int nOfColumns, int minDim);

void step3(int \*assignment, double \*distMatrix, bool \*starMatrix, bool \*newStarMatrix, bool \*primeMatrix, bool \*coveredColumns, bool \*coveredRows, int nOfRows, int nOfColumns, int minDim);

void step4(int \*assignment, double \*distMatrix, bool \*starMatrix, bool \*newStarMatrix, bool \*primeMatrix, bool \*coveredColumns, bool \*coveredRows, int nOfRows, int nOfColumns, int minDim, int row, int col);

void step5(int \*assignment, double \*distMatrix, bool \*starMatrix, bool \*newStarMatrix, bool \*primeMatrix, bool \*coveredColumns, bool \*coveredRows, int nOfRows, int nOfColumns, int minDim);

};

#include "hungarian.h"

#include "math.h"

#include "cfloat"

HungarianAlgorithm::HungarianAlgorithm(){}

HungarianAlgorithm::~HungarianAlgorithm(){}

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

// A single function wrapper for solving assignment problem.

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

double HungarianAlgorithm::Solve(vector<vector<double>>& DistMatrix, vector<int>& Assignment)

{

unsigned int nRows = DistMatrix.size();

unsigned int nCols = DistMatrix[0].size();

double \*distMatrixIn = new double[nRows \* nCols];

int \*assignment = new int[nRows];

double cost = 0.0;

// Fill in the distMatrixIn. Mind the index is "i + nRows \* j".

// Here the cost matrix of size MxN is defined as a double precision array of N\*M elements.

// In the solving functions matrices are seen to be saved MATLAB-internally in row-order.

// (i.e. the matrix [1 2; 3 4] will be stored as a vector [1 3 2 4], NOT [1 2 3 4]).

for (unsigned int i = 0; i < nRows; i++)

for (unsigned int j = 0; j < nCols; j++)

distMatrixIn[i + nRows \* j] = DistMatrix[i][j];

// call solving function

assignmentoptimal(assignment, &cost, distMatrixIn, nRows, nCols);

Assignment.clear();

for (unsigned int r = 0; r < nRows; r++)

Assignment.push\_back(assignment[r]);

delete[] distMatrixIn;

delete[] assignment;

return cost;

}

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

// Solve optimal solution for assignment problem using Munkres algorithm, also known as Hungarian Algorithm.

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

void HungarianAlgorithm::assignmentoptimal(int \*assignment, double \*cost, double \*distMatrixIn, int nOfRows, int nOfColumns)

{

double \*distMatrix, \*distMatrixTemp, \*distMatrixEnd, \*columnEnd, value, minValue;

bool \*coveredColumns, \*coveredRows, \*starMatrix, \*newStarMatrix, \*primeMatrix;

int nOfElements, minDim, row, col;

/\* initialization \*/

\*cost = 0;

for (row = 0; row<nOfRows; row++)

assignment[row] = -1;

/\* generate working copy of distance Matrix \*/

/\* check if all matrix elements are positive \*/

nOfElements = nOfRows \* nOfColumns;

distMatrix = (double \*)malloc(nOfElements \* sizeof(double));

distMatrixEnd = distMatrix + nOfElements;

for (row = 0; row<nOfElements; row++)

{

value = distMatrixIn[row];

if (value < 0)

cerr << "All matrix elements have to be non-negative." << endl;

distMatrix[row] = value;

}

/\* memory allocation \*/

coveredColumns = (bool \*)calloc(nOfColumns, sizeof(bool));

coveredRows = (bool \*)calloc(nOfRows, sizeof(bool));

starMatrix = (bool \*)calloc(nOfElements, sizeof(bool));

primeMatrix = (bool \*)calloc(nOfElements, sizeof(bool));

newStarMatrix = (bool \*)calloc(nOfElements, sizeof(bool)); /\* used in step4 \*/

/\* preliminary steps \*/

if (nOfRows <= nOfColumns)

{

minDim = nOfRows;

for (row = 0; row<nOfRows; row++)

{

/\* find the smallest element in the row \*/

distMatrixTemp = distMatrix + row;

minValue = \*distMatrixTemp;

distMatrixTemp += nOfRows;

while (distMatrixTemp < distMatrixEnd)

{

value = \*distMatrixTemp;

if (value < minValue)

minValue = value;

distMatrixTemp += nOfRows;

}

/\* subtract the smallest element from each element of the row \*/

distMatrixTemp = distMatrix + row;

while (distMatrixTemp < distMatrixEnd)

{

\*distMatrixTemp -= minValue;

distMatrixTemp += nOfRows;

}

}

/\* Steps 1 and 2a \*/

for (row = 0; row<nOfRows; row++)

for (col = 0; col<nOfColumns; col++)

if (fabs(distMatrix[row + nOfRows\*col]) < DBL\_EPSILON)

if (!coveredColumns[col])

{

starMatrix[row + nOfRows\*col] = true;

coveredColumns[col] = true;

break;

}

}

else /\* if(nOfRows > nOfColumns) \*/

{

minDim = nOfColumns;

for (col = 0; col<nOfColumns; col++)

{

/\* find the smallest element in the column \*/

distMatrixTemp = distMatrix + nOfRows\*col;

columnEnd = distMatrixTemp + nOfRows;

minValue = \*distMatrixTemp++;

while (distMatrixTemp < columnEnd)

{

value = \*distMatrixTemp++;

if (value < minValue)

minValue = value;

}

/\* subtract the smallest element from each element of the column \*/

distMatrixTemp = distMatrix + nOfRows\*col;

while (distMatrixTemp < columnEnd)

\*distMatrixTemp++ -= minValue;

}

/\* Steps 1 and 2a \*/

for (col = 0; col<nOfColumns; col++)

for (row = 0; row<nOfRows; row++)

if (fabs(distMatrix[row + nOfRows\*col]) < DBL\_EPSILON)

if (!coveredRows[row])

{

starMatrix[row + nOfRows\*col] = true;

coveredColumns[col] = true;

coveredRows[row] = true;

break;

}

for (row = 0; row<nOfRows; row++)

coveredRows[row] = false;

}

/\* move to step 2b \*/

step2b(assignment, distMatrix, starMatrix, newStarMatrix, primeMatrix, coveredColumns, coveredRows, nOfRows, nOfColumns, minDim);

/\* compute cost and remove invalid assignments \*/

computeassignmentcost(assignment, cost, distMatrixIn, nOfRows);

/\* free allocated memory \*/

free(distMatrix);

free(coveredColumns);

free(coveredRows);

free(starMatrix);

free(primeMatrix);

free(newStarMatrix);

return;

}

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

void HungarianAlgorithm::buildassignmentvector(int \*assignment, bool \*starMatrix, int nOfRows, int nOfColumns)

{

int row, col;

for (row = 0; row<nOfRows; row++)

for (col = 0; col<nOfColumns; col++)

if (starMatrix[row + nOfRows\*col])

{

#ifdef ONE\_INDEXING

assignment[row] = col + 1; /\* MATLAB-Indexing \*/

#else

assignment[row] = col;

#endif

break;

}

}

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

void HungarianAlgorithm::computeassignmentcost(int \*assignment, double \*cost, double \*distMatrix, int nOfRows)

{

int row, col;

for (row = 0; row<nOfRows; row++)

{

col = assignment[row];

if (col >= 0)

\*cost += distMatrix[row + nOfRows\*col];

}

}

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

void HungarianAlgorithm::step2a(int \*assignment, double \*distMatrix, bool \*starMatrix, bool \*newStarMatrix, bool \*primeMatrix, bool \*coveredColumns, bool \*coveredRows, int nOfRows, int nOfColumns, int minDim)

{

bool \*starMatrixTemp, \*columnEnd;

int col;

/\* cover every column containing a starred zero \*/

for (col = 0; col<nOfColumns; col++)

{

starMatrixTemp = starMatrix + nOfRows\*col;

columnEnd = starMatrixTemp + nOfRows;

while (starMatrixTemp < columnEnd){

if (\*starMatrixTemp++)

{

coveredColumns[col] = true;

break;

}

}

}

/\* move to step 3 \*/

step2b(assignment, distMatrix, starMatrix, newStarMatrix, primeMatrix, coveredColumns, coveredRows, nOfRows, nOfColumns, minDim);

}

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

void HungarianAlgorithm::step2b(int \*assignment, double \*distMatrix, bool \*starMatrix, bool \*newStarMatrix, bool \*primeMatrix, bool \*coveredColumns, bool \*coveredRows, int nOfRows, int nOfColumns, int minDim)

{

int col, nOfCoveredColumns;

/\* count covered columns \*/

nOfCoveredColumns = 0;

for (col = 0; col<nOfColumns; col++)

if (coveredColumns[col])

nOfCoveredColumns++;

if (nOfCoveredColumns == minDim)

{

/\* algorithm finished \*/

buildassignmentvector(assignment, starMatrix, nOfRows, nOfColumns);

}

else

{

/\* move to step 3 \*/

step3(assignment, distMatrix, starMatrix, newStarMatrix, primeMatrix, coveredColumns, coveredRows, nOfRows, nOfColumns, minDim);

}

}

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

void HungarianAlgorithm::step3(int \*assignment, double \*distMatrix, bool \*starMatrix, bool \*newStarMatrix, bool \*primeMatrix, bool \*coveredColumns, bool \*coveredRows, int nOfRows, int nOfColumns, int minDim)

{

bool zerosFound;

int row, col, starCol;

zerosFound = true;

while (zerosFound)

{

zerosFound = false;

for (col = 0; col<nOfColumns; col++)

if (!coveredColumns[col])

for (row = 0; row<nOfRows; row++)

if ((!coveredRows[row]) && (fabs(distMatrix[row + nOfRows\*col]) < DBL\_EPSILON))

{

/\* prime zero \*/

primeMatrix[row + nOfRows\*col] = true;

/\* find starred zero in current row \*/

for (starCol = 0; starCol<nOfColumns; starCol++)

if (starMatrix[row + nOfRows\*starCol])

break;

if (starCol == nOfColumns) /\* no starred zero found \*/

{

/\* move to step 4 \*/

step4(assignment, distMatrix, starMatrix, newStarMatrix, primeMatrix, coveredColumns, coveredRows, nOfRows, nOfColumns, minDim, row, col);

return;

}

else

{

coveredRows[row] = true;

coveredColumns[starCol] = false;

zerosFound = true;

break;

}

}

}

/\* move to step 5 \*/

step5(assignment, distMatrix, starMatrix, newStarMatrix, primeMatrix, coveredColumns, coveredRows, nOfRows, nOfColumns, minDim);

}

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

void HungarianAlgorithm::step4(int \*assignment, double \*distMatrix, bool \*starMatrix, bool \*newStarMatrix, bool \*primeMatrix, bool \*coveredColumns, bool \*coveredRows, int nOfRows, int nOfColumns, int minDim, int row, int col)

{

int n, starRow, starCol, primeRow, primeCol;

int nOfElements = nOfRows\*nOfColumns;

/\* generate temporary copy of starMatrix \*/

for (n = 0; n<nOfElements; n++)

newStarMatrix[n] = starMatrix[n];

/\* star current zero \*/

newStarMatrix[row + nOfRows\*col] = true;

/\* find starred zero in current column \*/

starCol = col;

for (starRow = 0; starRow<nOfRows; starRow++)

if (starMatrix[starRow + nOfRows\*starCol])

break;

while (starRow<nOfRows)

{

/\* unstar the starred zero \*/

newStarMatrix[starRow + nOfRows\*starCol] = false;

/\* find primed zero in current row \*/

primeRow = starRow;

for (primeCol = 0; primeCol<nOfColumns; primeCol++)

if (primeMatrix[primeRow + nOfRows\*primeCol])

break;

/\* star the primed zero \*/

newStarMatrix[primeRow + nOfRows\*primeCol] = true;

/\* find starred zero in current column \*/

starCol = primeCol;

for (starRow = 0; starRow<nOfRows; starRow++)

if (starMatrix[starRow + nOfRows\*starCol])

break;

}

/\* use temporary copy as new starMatrix \*/

/\* delete all primes, uncover all rows \*/

for (n = 0; n<nOfElements; n++)

{

primeMatrix[n] = false;

starMatrix[n] = newStarMatrix[n];

}

for (n = 0; n<nOfRows; n++)

coveredRows[n] = false;

/\* move to step 2a \*/

step2a(assignment, distMatrix, starMatrix, newStarMatrix, primeMatrix, coveredColumns, coveredRows, nOfRows, nOfColumns, minDim);

}

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

void HungarianAlgorithm::step5(int \*assignment, double \*distMatrix, bool \*starMatrix, bool \*newStarMatrix, bool \*primeMatrix, bool \*coveredColumns, bool \*coveredRows, int nOfRows, int nOfColumns, int minDim)

{

double h, value;

int row, col;

/\* find smallest uncovered element h \*/

h = DBL\_MAX;

for (row = 0; row<nOfRows; row++)

if (!coveredRows[row])

for (col = 0; col<nOfColumns; col++)

if (!coveredColumns[col])

{

value = distMatrix[row + nOfRows\*col];

if (value < h)

h = value;

}

/\* add h to each covered row \*/

for (row = 0; row<nOfRows; row++)

if (coveredRows[row])

for (col = 0; col<nOfColumns; col++)

distMatrix[row + nOfRows\*col] += h;

/\* subtract h from each uncovered column \*/

for (col = 0; col<nOfColumns; col++)

if (!coveredColumns[col])

for (row = 0; row<nOfRows; row++)

distMatrix[row + nOfRows\*col] -= h;

/\* move to step 3 \*/

step3(assignment, distMatrix, starMatrix, newStarMatrix, primeMatrix, coveredColumns, coveredRows, nOfRows, nOfColumns, minDim);

}