**版本信息**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **版本信息记录表** | | | | |
| 版本号 | 修订内容 | 修订人 | 修订日期 | 审核人 |
| 1.0 | 初版 | 党慧杰 | 2018.08.20 |  |
| 2.0 | 修订 | 党慧杰 | 2018.10.18 |  |
|  |  |  |  |  |

# 边缘提取算法

## 【实验目的】

三种不同的算子识别出图片边缘。

## 【实验原理】

1. **OpenCV简介**

OpenCV是一个开放源代码的计算机视觉应用平台，由英特尔公司下属研发中心俄罗斯团队发起该项目，开源BSD证书，OpenCV的目标是实现实时计算机视觉，是一个跨平台的计算机视觉库。其应用领域包括：二维和三维特征工具箱、运动估算、人脸系统识别、姿势识别、人机交互、移动机器人、运动理解、对象鉴别、分割与识别、立体视觉、运动跟踪、增强实现(AR技术)。

1. **边缘识别算子**

边缘识别又称边缘检测，是模仿人类视觉的一个过程。在检测物体边缘时, 先对其轮廓点进行粗略检测, 然后通过链接规则把原来检测到的轮廓点连接起来, 同时也检测和连接遗漏的边界点及去除虚假的边界。边缘检测的目的是去发现图像中关于形状和反射或透射比的信息, 是图像处理、图像分析、模式识别、计算机视觉以及人类视觉的基本步骤之一。

1. Sobel算子

Sobel算子可以直接计算Gx 、Gy可以检测到边的存在，以及从暗到亮，从亮到暗的变化。仅计算| Gx |，产生最强的响应是正交 于x轴的边； | Gy |则是正交于y轴的边。

函数原型如下：

void Sobel(InputArray src, OutputArray dst, int ddepth, int xorder, int yorder, int ksize=3, double scale=1, double delta=0, int borderType=BORDER\_DEFAULT )

参数说明：

* src: 原始图像
* dst: 结果输出图像
* ddepth: 输出图像的深度
* xorder: x方向上的差分阶数，1或0
* yorder: y方向上的差分阶数，1或0
* ksize: 模板大小，取值为1，3，5，7。默认为3
* scale: 可选的计算导数时的放大因子。默认为1
* delta: 在存储目标图像前可选的添加到图像的值，默认为0。
* borderType: 像素向外逼近的方法。默认为BORDER\_DEFAULT，即对全部边界进行计算。

1. Canny算子

Canny的目标是找到一个最优的边缘检测算法。其原始原型如下：

void Canny(InputArray image, OutputArray edges, double threshold1, double threshold2, int apertureSize=3, bool L2gradient=false )

参数说明：

* image: 输入图像，8位单通道
* edges: 输出边缘图像
* threshold1: 阈值1
* threshold2: 阈值2
* apertureSize: Sober算子大小，默认为3
* L2gradient: 是否采用更精确的方式计算图像梯度，默认false

1. Laplacian算子

Laplacian算子是n维欧几里德空间中的一个二阶微分算子，定义为梯度grad的散度div。可使用运算模板来运算这定理定律。函数原型如下：

void Laplacian(InputArray src, OutputArray dst, int ddepth, int ksize=1, double scale=1, double delta=0, int borderType=BORDER\_DEFAULT )

参数说明：

* src: 原始图像
* dst: 结果输出图像
* ddepth: 输出图像的深度
* ksize: 计算二阶导数滤波器尺寸大小
* scale: 可选的计算Laplace值时的放大因子。默认为1
* delta: 在存储目标图像前可选的添加到图像的值，默认为0。
* borderType: 像素向外逼近的方法。默认为BORDER\_DEFAULT，即对全部边界进行计算。

## 【实验环境】

操作系统

Windows 系列 36位/64位

开发环境

Visual Studio 2012、OpenCV

开发语言

C++

设备

PC（配备USB3.0）、Kinect

## 【实验步骤】

### 一、新建Win32控制台应用程序

1. 点击菜单栏【文件】->【新建】->【项目】或快捷件Ctrl + Shift + N 新建项目，如下图所示：



图 1

2. 在弹出的【新建项目】框中选择模板Visual C++，在中间的栏目中选择【Win32 控制台应用程序】。输入工程名称和保存路径，点击确定。

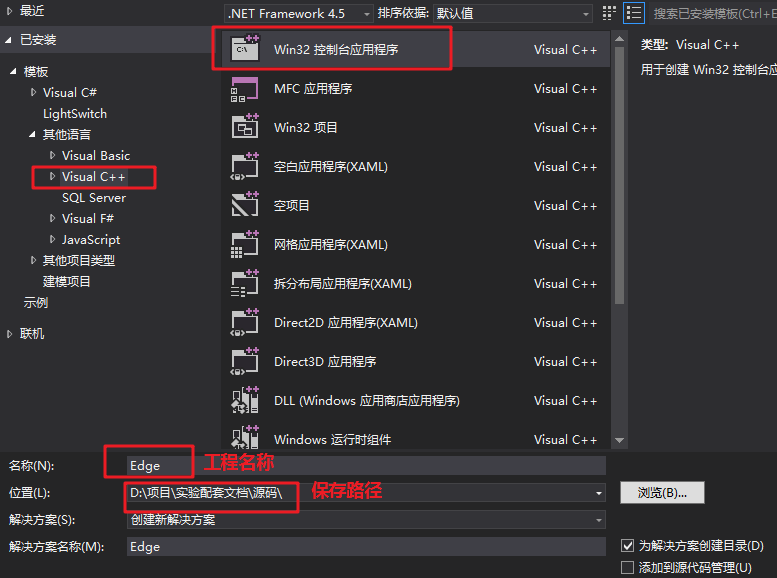


图2

3. 在弹出的【Win32 应用程序向导】对话框中，保持默认配置，直至完成

### 二、OpenCV配置

在Visual Studio 2012项目中配置OpenCV，参见文档《Opencv的配置》。Kinect的配置，参见文档《Kinect使用说明》。

### 三、Sobel算子

**1. 新建ImageHandler类**

1）右击项目，选择【添加】->【类】，如下图所示：

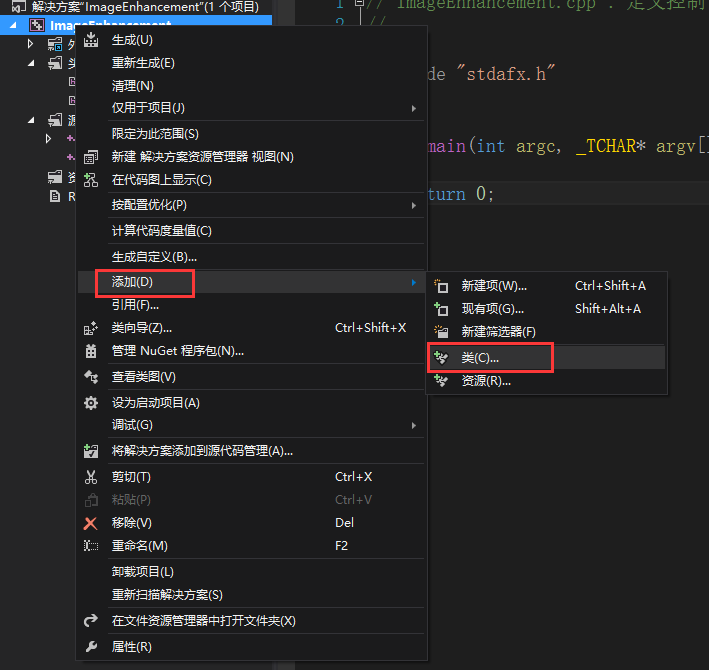


图3

2）选择C++类，点击【添加】

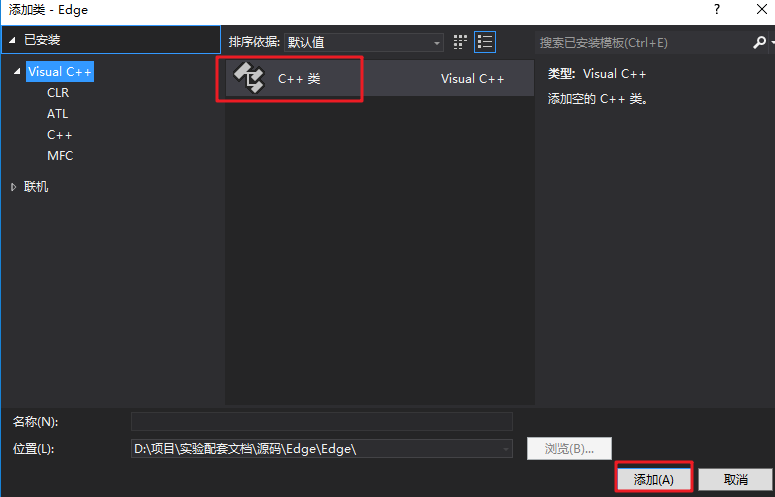


图 4

3）输入类名，点击确定，新类就创建完成。

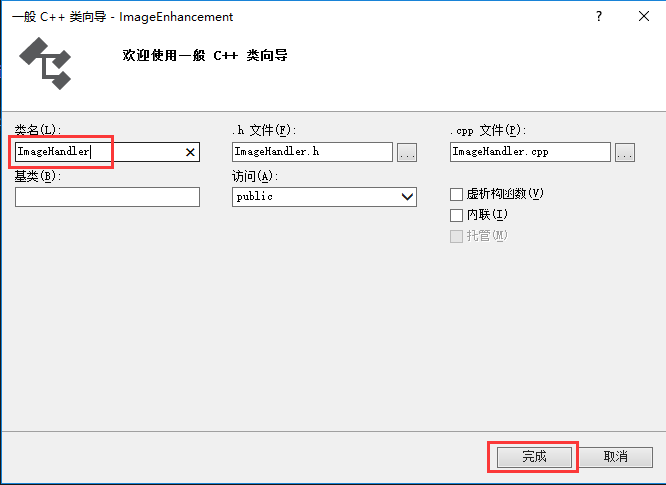


图5

**2. 引入必需的头文件和命名空间**

该方法主要是利用OpenCV提供的函数来完成，因而首先必需添加相关的头文件。打开‘ImageHandler.h’文件。

添加如下头文件：

#include <opencv2/core/core.hpp>

#include <opencv2/imgproc/imgproc.hpp>

#include <opencv2/highgui/highgui.hpp>

引入命名空间：

using namespace cv;

如下图所示：

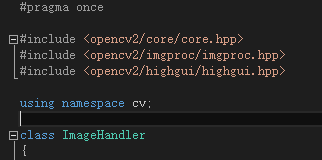


图6

注意：如果不能导入头文件，请检查OpenCV的配置是否正确。

**3. Sobel算子边缘检测的实现**

1）声明函数

打开‘ImageHandler.h’文件，在 ImageHandler 类中添加如下函数声明：

void EdgeSobel(Mat frame);

其中，参数frame为需要处理的一帧图像。

2）函数实现

算法实现的流程如下：

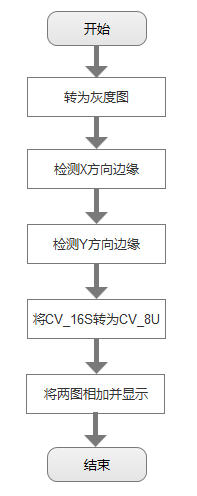


图7

打开‘ImageHandler.cpp’文件，键入如下程序：

void ImageHandler::EdgeSobel(Mat frame)

{

Mat grayImage, resultImage;

//转为灰度图

cvtColor(frame, grayImage, CV\_RGB2GRAY);

Mat ax, ay;

Mat axx, ayy;

// x方向

Sobel(grayImage, ax, CV\_16S, 1, 0, 3);

// y方向

Sobel(grayImage, ay, CV\_16S, 0, 1, 3);

//将CV\_16S转为CV\_8U

convertScaleAbs(ax, axx);

convertScaleAbs(ay, ayy);

//将两图相加

addWeighted(axx, 0.5, ayy, 0.5, 0,resultImage);

// 显示

imshow("Sobel 算子边缘检测", resultImage);

}

### 四、拉普拉斯算子

**1. 拉普拉斯算子边缘检测的实现**

1）声明函数

打开‘ImageHandler.h’文件，在 ImageHandler 类中添加如下函数声明：

void EdgeLaplacian(Mat frame);

其中，参数frame为需要处理的一帧图像。

2）函数实现

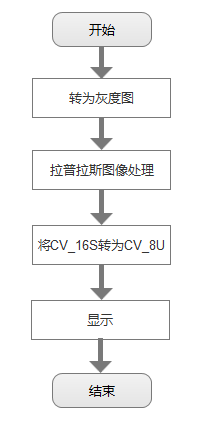


图 8

打开‘ImageHandler.cpp’文件，键入如下程序：

void ImageHandler::EdgeLaplacian(Mat frame)

{

Mat grayImage, resultImage;

//转为灰度图

cvtColor(frame, grayImage, CV\_RGB2GRAY);

// 拉普拉斯图像处理

Laplacian(grayImage, resultImage, CV\_16S, 3);

//将CV\_16S转为CV\_8U

convertScaleAbs(resultImage, resultImage);

// 显示

imshow("Laplacian 算子边缘检测", resultImage);

}

### 五、Canny算子

1）声明函数

打开‘ImageHandler.h’文件，在 ImageHandler 类中添加如下函数声明：

void EdgeCanny(Mat frame);

其中，参数frame为需要处理的一帧图像。

2）函数实现

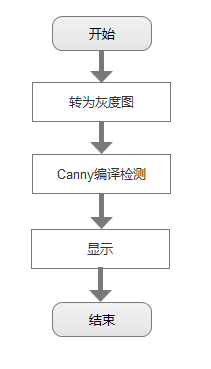


图 9

打开‘ImageHandler.cpp’文件，键入如下程序：

void ImageHandler::EdgeCanny(Mat frame)

{

Mat grayImage, resultImage;

//转为灰度图

cvtColor(frame, grayImage, CV\_RGB2GRAY);

// Canny 编译检测

Canny(grayImage, resultImage, 100, 300, 3);

// 显示

imshow("Canny 算子边缘检测", resultImage);

}

### 六、从Kinect获取彩色图像并进行边缘检测

**1. 引入头文件**

在主程序入口函数之前，添加如下的头部文件：

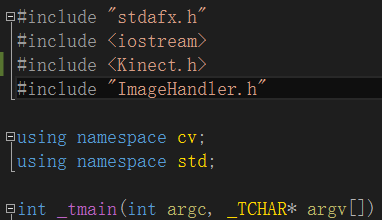


图 10

**2. 获取Kinect彩色图像**

从Kinect获取图像的一般过程为：

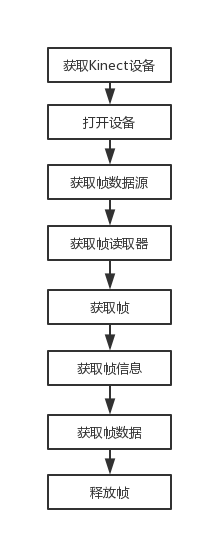


图 11

相关代码如下：

IKinectSensor\* pKinectSensor = NULL;

IColorFrameSource\* pColorFrameSource = NULL;

IColorFrameReader\* pColorFrameReader = NULL;

IFrameDescription\* pFrameDescription = NULL;

ColorImageFormat imgFormat = ColorImageFormat\_None;

int width, height;

uchar\* pBuffer = NULL;

UINT bufferSize = 0;

// 获取设备

HRESULT hr = GetDefaultKinectSensor(&pKinectSensor);

if (FAILED(hr))

{

cout << "获取Kinect设备失败" << endl;

goto exit;

}

// 打开设备

hr = pKinectSensor->Open();

if (FAILED(hr))

{

cout << "打开Kinect设备失败" << endl;

goto exit;

}

// 获取彩色帧数据源

hr = pKinectSensor->get\_ColorFrameSource(&pColorFrameSource);

if (FAILED(hr))

{

cout << "获取Kinect彩色数据源失败" << endl;

goto exit;

}

// 打开颜色帧读取器

hr = pColorFrameSource->OpenReader(&pColorFrameReader);

if (FAILED(hr))

{

cout << "打开Kinect颜色帧读取器失败" << endl;

goto exit;

}

while (1)

{

IColorFrame\* pColorFrame = NULL;

hr = pColorFrameReader->AcquireLatestFrame(&pColorFrame);

if (FAILED(hr)) continue;

// 获取帧信息

pColorFrame->get\_FrameDescription(&pFrameDescription);

pFrameDescription->get\_Width(&width);

pFrameDescription->get\_Height(&height);

// 获取图像格式

pColorFrame->get\_RawColorImageFormat(&imgFormat);

// 拷贝数据

Mat img(height, width, CV\_8UC4);

pBuffer = img.data;

bufferSize = img.rows \* img.step;

pColorFrame->CopyConvertedFrameDataToArray(bufferSize, reinterpret\_cast<BYTE\*>(pBuffer), ColorImageFormat\_Bgra);

// 释放帧

pColorFrame->Release();

imshow("Kinect彩色数据", img);

if (waitKey(30) == 27) // 按ESC退出

{

break;

}

}

**3. 对图像进行边缘检测**

1）在入口程序\_tmain中添加ImageHnadler实例对象。如下所示：

ImageHandler imgHandler;

2）在while（1）中，定位到‘释放帧’代码。在该代码下添加如下处理函数调用：

// 缩小尺寸

resize(img, img, cv::Size(width / 4, height / 4));

// 颜色转换

cvtColor(img, img, CV\_BGRA2BGR);

// 图像边缘检测

imgHandler.EdgeCanny(img);

imgHandler.EdgeLaplacian(img);

imgHandler.EdgeSobel(img);

**4. 运行程序**

点击vs工具栏中的【本地Windows调试器】（如下图）运行该程序。



图 12

程序成功运行后，将弹出如下窗口：

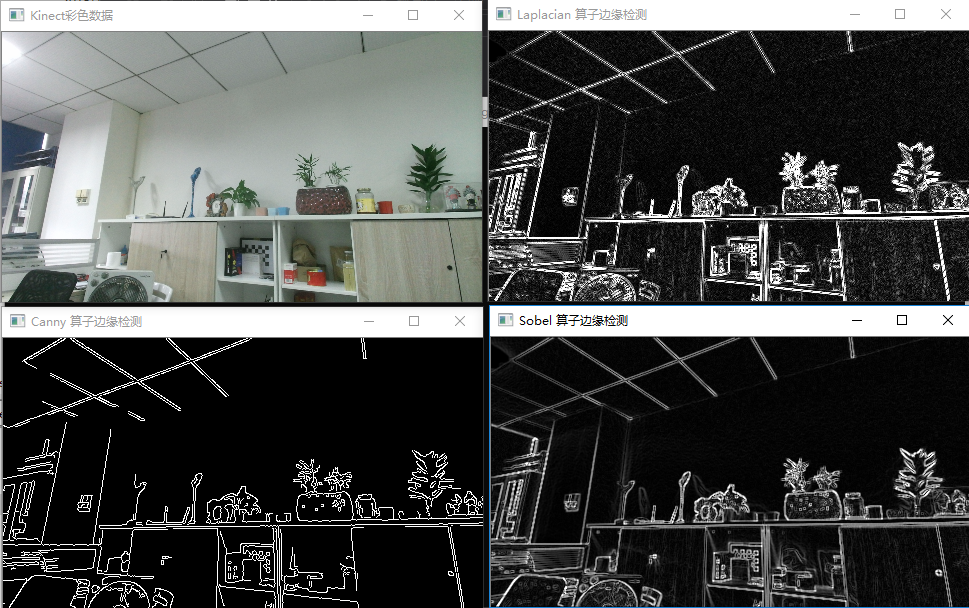


图 13

## 【思考题】

### 选择题

题目1：下面说法错误的是：（C）

A. Sobel 检测算子对噪声具有平滑作用，能很好的消除噪声的影响。

B. Laplace算子对孤立象素的响应要比对边缘或线的响应要更强烈，因此只适用于无噪声图象。

C.存在噪声情况下，使用Laplacian算子检测边缘之前不需要先进行低通滤波。

题目2：下列图象边缘检测算子中抗噪性能最好的是（B）。

A . 梯度算子

B . Prewitt算子

C . Roberts算子

D . Laplacian算子

### 2、简答题

题目1：Sobel的原理是什么？

索贝尔算子（Sobeloperator）是图像处理中的算子之一，主要用作边缘检测。在技术上，它是一离散性差分算子，用来运算图像亮度函数的梯度之近似值。在图像的任何一点使用此算子，将会产生对应的梯度矢量或是其法矢量。

该算子包含两组3x3的矩阵，分别为横向及纵向，将之与图像作平面卷积，即可分别得出横向及纵向的亮度差分近似值。如果以A代表原始图像，Gx及Gy分别代表经横向及纵向边缘检测的图像。