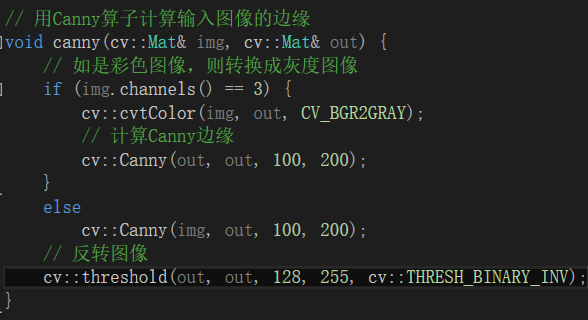
本节的目标，是对视频序列中的每一帧进行处理。我们拟创建一个自定义类，封装OpenCV的视频捕获和处理框架。为了使这个类更加通用，我们不把处理函数封装进类中，而是在该类中定义一个指向视频帧处理函数的指针，这样具体的函数由用户自己定义，并设置类的函数指针指向该函数就好了，当然这样要求所有处理函数都有统一的原型，我们采用的处理函数原型如下：

void processFrame(cv::Mat& img, cv::Mat& out);

在这里我们以应用Canny算法提取图像的边缘为例，对视频帧进行处理。

让我们开始编码吧！

1. 首先定义处理函数：

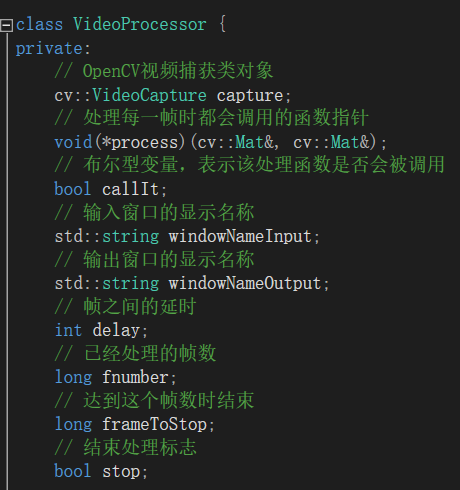


说明：

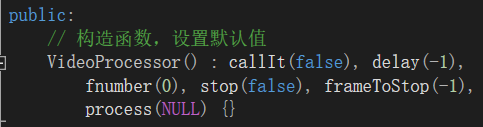
* 1. 由于Canny算子处理的是灰度图，因此如果输入图像是彩色图像，得先转为灰度图
  2. 100和200两个阈值的设定是需要根据具体任务来设定的，此处其值的选择有些随意
  3. Canny算子的输出图中，通常是轮廓显示为白色，背景显示为黑色的，这里通过threshold阈值化将黑白颠倒过来，因此最后一个参数用的是cv::THRESH\_BINARY\_INV(小于阈值变白色，大于阈值变黑色)，阈值128其实并不唯一，设置任何位于区间(0,255)中的数值作为阈值均可

1. 接下来开始定义VideoProcessor类。

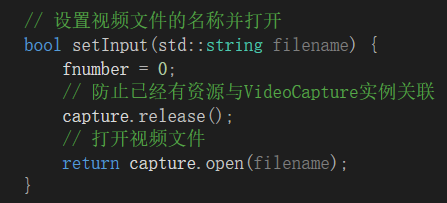
首先是私有成员属性：



定义构造函数为这些属性赋初值：



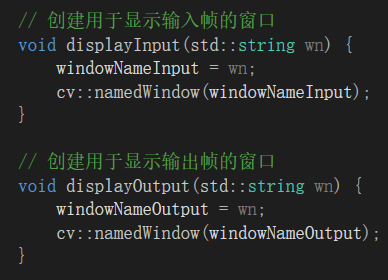
定义函数设置视频文件的名称并打开：



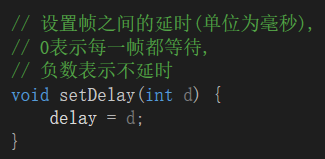
为了能打开摄像头，我们再定义一个重载版本：



定义两成员方法分别创建用于显示输入帧和输出帧的窗口：

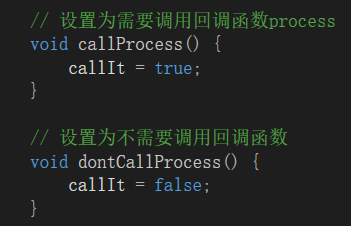


定义方法用于设置帧间延时：

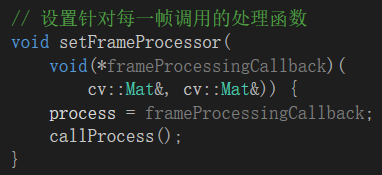


说明：注释里所说的功能由后面run函数来实现。

为了增强灵活性，我们允许用户选择不对帧进行处理（这样输出帧直接等于输入帧）：

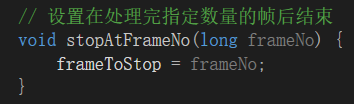


定义设置帧处理函数的成员方法：

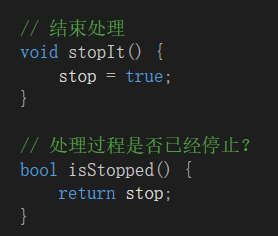


说明：在设置处理函数的同时，通过调用callProcess置callIt为true

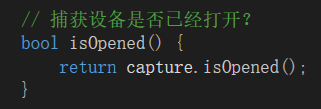
设置在处理完指定数量的帧后结束：



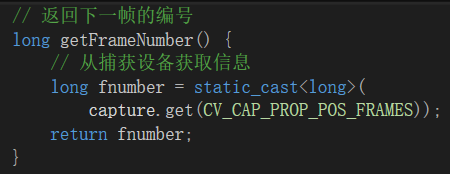
定义成员方法用于终止处理过程以及返回终止状态：



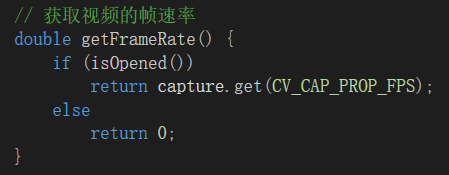
定义用于检测视频文件或设备是否已打开的成员方法：



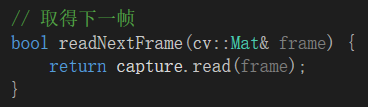
定义成员方法返回下一帧的编号：



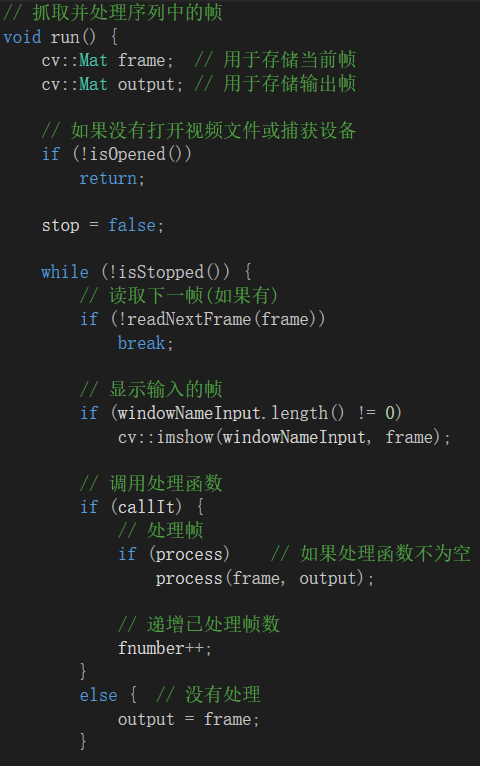
定义成员方法获取视频的帧速率：

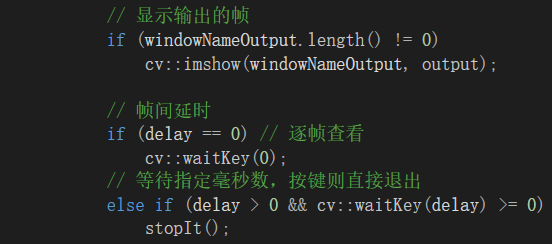


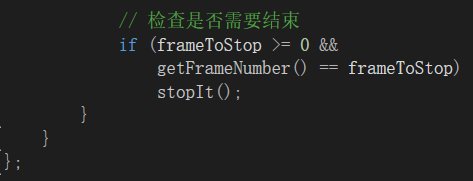
定义私有成员方法读取下一帧：



有了以上的铺垫，可以编写最主要的run方法了：

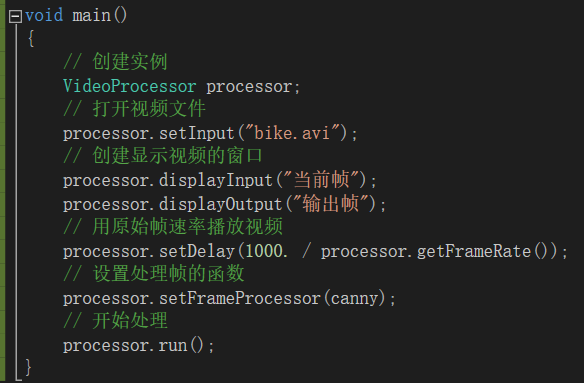






run方法首先调用OpenCV类cv::VideoCapture的read方法，然后执行一系列操作。但是在执行之前，要先检查该操作是否需要执行。只有指定了输入窗口的名称（用displayInput方法），才会显示输入窗口；只有指定了回调函数（用setFrameProcessor方法），才会运行帧处理函数；只有定义了输出窗口的名称（用displayOutput），才会显示输出窗口；只有指定了延时（用setDelay），才会执行延时（注意此段代码和书中不一致，以支持逐帧查看）。最后，如果定义了需处理的最大帧数（用stopAtFrameNo），就需要检查当前的帧数。

1. 最后可以写main函数了：



测试下吧！注意视频文件bike.avi要放到项目文件夹下面。

1. 通过调用processor.setDelay(0)来测试下逐帧查看功能，通过按键来切换到下一帧。
2. 通过调用类的stopAtFrameNo成员函数来测试下指定在处理完指定数量的帧后结束的功能
3. 通过调用processor.setInput(0);设置从摄像头来获取视频，注意此时getFrameRate获取的帧速率为0，这样processor.setDelay(1000. / processor.getFrameRate());设置的延时将是无穷大，从而出现问题，因此我们改用processor.setDelay(1000.0 / 30); 即人为指定每秒30帧
4. 思考：利用已学知识，是否可以利用OpenCV实现视频播放器的功能？而且还可实现实时视频特效呢。
5. 在面向对象的编程中，最好使用帧处理类而不是帧处理函数。实际上，在定义视频处理算法时，使用类能提供更大的灵活性。我们定义一个接口：

// 处理帧的接口

class FrameProcessor {

public:

// 处理方法

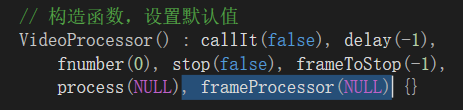
virtual void process(cv::Mat &input, cv::Mat &output) = 0; // 纯虚函数

};

然后在VideoProcessor类中增加一个私有成员属性：

FrameProcessor\* frameProcessor; // 帧处理器对象指针

在构造函数中增加该属性的初始化：



增加一个setFrameProcessor重载函数用设置帧处理类：

// 设置实现FrameProcessor接口的实例

void setFrameProcessor(FrameProcessor\* frameProcessorPtr) {

// 使处理函数失效

process = NULL;

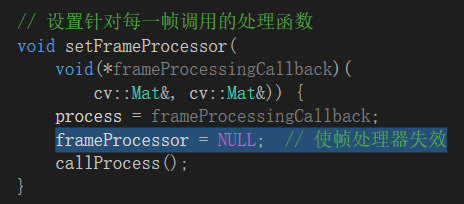
// 这个就是即将被调用的帧处理器接口

frameProcessor = frameProcessorPtr;

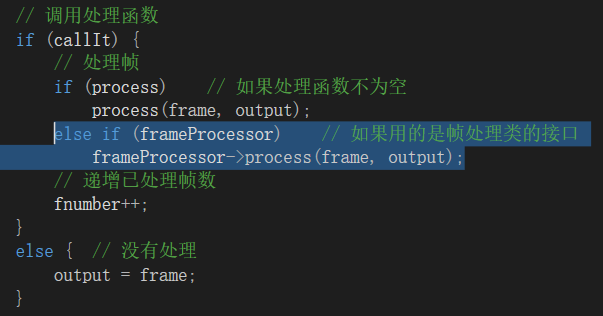
callProcess();

}

同时在之前的setFrameProcessor类中增加一行，即使用处理函数的话就不使用帧处理类：



在run函数中增加2行：



此段修改暂时无法测试，在11.5中会用到此帧处理类。