使用 OpenCV 必备知识点 (C++)

Dezeming Family

2022 年 5 月 23 日

DezemingFamily 系列文章和电子书**全部都有免费公开的电子版**，可以很方便地进行修改和重新发布。如果您获得了 DezemingFamily 的系列电子书，可以从我们的网站 [https://dezeming.top/] 找到最新的版本。对文章的内容建议和出现的错误也欢迎在网站留言。

目录

一　参考资料

本节是学习 OpenCV 的参考资料介绍。

对 C++ 用户来说，掌握《C++ primer》是必需的，之后可以通过 [1] 来入门 OpenCV3。

对于 Python 用户来说，《OpenCV 轻松入门》是一个很好的选择，入门较为简单。不过可以看出该书是仿照 [2] 来写的，[2] 更加全面。

以及 OpenCV 的官方文档网站：[3]，该网站的内容非常全面，适合有一定经验的人去查找和使用。

有了前面这些参考资料的基础，接下来如何使用 OpenCV 就很容易去自己探索了。我在知乎和 B 站看到很多科研工作人员和 CV 爱好者给出了一些有趣的课程和学习资料，比如三维定位、手势识别，不但包含离线处理系统，还包括实时系统（目标跟踪、实时三维重建），都挺有意思，以后有精力我也会计划安排一些小项目。

二　基本模块

在这个示例中，我编译的是完整的 OpenCV3.4.2 版本，OpenCV、contrib 以及 CUDA。本节来讲解一下 OpenCV 的模块（相当于功能集合）。

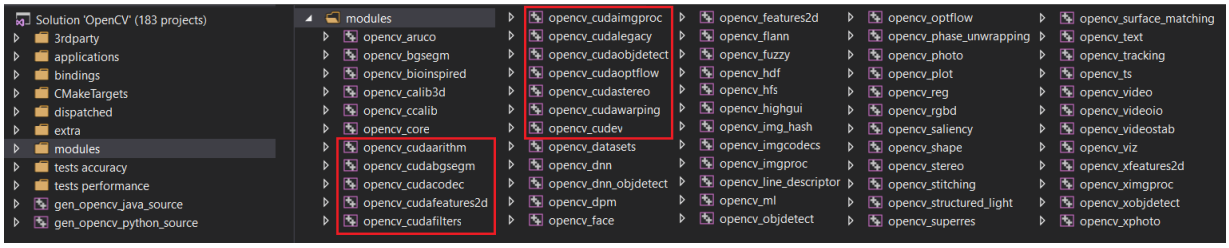

图 1: OpenCV 的 modules

打开 cmake 生成的工程，可以看到图1的工程。modules 下是我们需要用到的模块。除了红框里与 cuda 加速相关的模块，我们把重点放在一般的模块上。

最基本的模块是 core，里面包含了 OpenCV 的基本实现，比如一些针对 cv::Mat 的基本操作。我们先把除了 contrib 模块之外的基本模块进行介绍（我会在模块后加上“主模块”记号），然后按照字典顺序介绍 contrib 模块的内容（我会在模块后面加上“contrib”记号），模块名等同于工程名。

这里面有相当多的技术都来自比较新的论文，有些论文会自成一个 module。详细的文档可以参考 [4]。

2 1　主模块

Calib3d（主模块）

该模块用来校准单个相机以及双目或多个相机，从二维图像中获取三维世界的信息。

Feature2d（主模块）

用来检测、描述和匹配特征点（二维特征）。

Flann（主模块）

快速最近邻库，有很多算法需要用到该库来实现最邻近搜索。

GPU-CUDA（主模块）

就是一大堆 cuda 模块。

Highgui（主模块）

包含了一个轻量级 GUI，以及用户交互函数。

Imgcodecs（主模块）

图像读写相关的函数（编解码）。

Imgproc（主模块）

包含很多图像处理函数，比如边缘检测、卷积模糊、轮廓检测等。

Ml（主模块）

一个很完备的机器学习模块。

Objdectect（主模块）

用来检测特定目标的算法，例如人脸检测或行人检测。也可以来训练检测器检测其他物体。

Photo（主模块）

计算摄影学的相关工具和算法。

Stitching（主模块）

图像拼接相关的算法，包含完整的拼接流程。

Video（主模块）

可用于视频流的算法，如运动提取、特征跟踪和前景提取。

Videoio（主模块）

以前放在 Highgui 里，OpenCV3.0 以后被单独移了出来。该模块用来进行视频读写。

Videostab（主模块）

包含一组可用于解决视频稳定问题的函数和类。

Viz（主模块）

用于进行 3D 可视化的工具，例如 mesh 可视化。

2 2 Contrib 模块

aruco（contrib）

ArUco 和 ChArUco 标记，与位姿估计、棋盘标定、AR 等有关。

bgsegm（contrib）

背景分割，包含了用于实时人体跟踪的算法。

bioinspired（contrib）

生物视觉相关的内容，包括瞬态事件分割、快速 HDR 色调映射等。

ccalib（contrib）

自定义标定相关算法。比如用于三维重建的图案，全向相机标定等。

datasets（contrib）

数据集读取器，读取已经存在于计算机视觉数据库的代码，并基于数据集的数据来训练和测试。

dnn（contrib）

深度神经网络，可以读取 Caffe 神经网络库中训练的图像识别网络。
dnn_objdetect 模块是用于目标检测的深度神经网络。

dpm（contrib）

可变形组件模型。Felzenszwalb 大佬的可变形组件的级联目标识别程序。

face（contrib）

人脸识别算法，例如 Eigen、Fisher 和 LBPH 算法等。

fuzzy（contrib）

模糊逻辑图像变换和逆变换、模糊图像处理方法。

hdf（contrib）

层次型数据存储，包含分层数据格式的 I/O 例程，可以存储大量数据。

hfs（contrib）

包含一种高效的图像分割算法。

line_descriptor（contrib）

线段提取和匹配算法，根据提取、描述和使用二进制描述子来锁定线段。

optflow（contrib）

与光流估计有关，包括运行和评估深流、简单流、稀疏流和运动模板（剪影流）的算法。

plot（contrib）

用于绘制一维和二维数据。

reg（contrib）

图像配准，基于像素的图像配准算法。

rgbd（contrib）

RGB 深度处理模块，包括三维目标识别、快速表面法线计算、三位平面查找、3D 视觉里程计等内容。

saliency（contrib）

显著性检测 API。

stereo（contrib）

双目立体匹配，有很多不同的描述符用于双目匹配。

structured_light（contrib）

结构光的应用，生成和投影格雷码图案，使用它们来计算场景的稠密深度。

surface_matching（contrib）

点对特征，用于 3D 目标检测和定位。

text（contrib）

视觉文本匹配，在场景中检测和识别文本。

tracking（contrib）

基于视觉的目标跟踪。

xfeatures2d（contrib）

是 Features2D 的扩展版本，包含了一些需要付费的 2D 特征检测描述子算法。

ximgproc（contrib）

图像处理方法的扩展，包括结构森林、引导滤波、超像素重建等重要方法。

xobjdetect（contrib）

功能增强的 2D 目标检测。

xphoto（contrib）

扩展的计算摄影方法，还有附加的照片处理算法，比如图像修复、色彩平衡等。

2 3 头文件

最全的头文件定义在了 include/opencv2/opencv.hpp 中，它不但包括了主模块，还包括了 contrib 模块。为了编译更快，我们也可以单独包含需要的文件，比如如果我们只想用 OpenCV 来读取图像，那么完全可以只包含 imgcodecs.hpp 头文件。

我经常能从网上看到很多人说不知道哪个函数要引用哪个头文件，或者哪个函数属于什么功能，这是对 OpenCV 的 modules 种类不熟导致的，在我们使用 OpenCV 之前，一定要尽可能把主模块中的几个重点主模块的功能搞清楚，比如 Imgproc、Highgui 和 Imgproc，这样会对以后的开发奠定良好的基础。

三 OpenCV 的数据类型

除了 int、float 这种从 C++ 继承来的原语类型，OpenCV 的类型主要分为三种：

- 点、向量、矩形、尺寸（size）等基础数据类型。
- 辅助对象类型，例如垃圾收集指针、数据切片范围（range）、抽象的终止条件类以及异常处理类等。
- 大型数组类型，例如 Mat、SparseMat 类型。我们放在下一个章节统一进行介绍。

我们本章节重点介绍前两个类型：基础数据类型和辅助对象类型。辅助对象类型涉及不少 C++ 技巧（其实并不难），我们在这里并不细讲，而是在其他文章中通过实例进行讲解。

3 1 基础数据类型

基础数据类型主要有 Point 类、Scalar 类、Size 类、Rect 类、RotatedRect 类、Matx 类（注意这不是大型数组类型，而是小矩阵，比如 3×3 的矩阵）、Vec 类、Complex 类（不同于 STL 的复数类）。

由于基础数据类型的运算方式在很多系统中都是一样的（当我们在构建一些计算机算法系统，比如几何处理系统，也会提前定义好一大堆基础数据类型，如果我们在项目中包含了 OpenCV 模块，那么就可以直接使用 OpenCV 的基础数据类型），所以不会细讲每个类的具体功能，而是介绍是干什么用的。

cv::Point{2,3}{i,f,d}

本质上是模板类 Point<>，比如 Point<float,2> 等，但我们一般会使用它的别名，例如 Point2i，Point3f。该类表示一个点，可以做与点有关的各种运算，比如相减，还有可以用来判断是否在一个矩形 Rect 对象的内部（仅限于二维情况。）访问内部元素的方式是.x，.y。

cv::Vec{2,3,4,6}{b,w,s,i,f,d}

固定向量类，也是一个模板类。一般使用别名，例如 Vec3f，Vec4i 等。这是非常常用的基本数据类型。

cv::Scalar

该类直接从 cv::Vec<double,4> 继承而来，因此具有 Vec4d 的特性，同时还包括了四元数的一些相关计算。

cv::Size

一共只有两种形式：cv::Size（等同于 cv::Size2i），以及 cv::Size2f。尺寸类的成员有.width 和.height，我们可以调用.area() 计算面积。

cv::Rect 类

矩形类的成员有.x、.y、（x 和 y 表示矩形的左上角）.width 和.height，我们也可以调用.area() 计算面积。该类并不是继承自 Point 类或者 Rect 类。

cv::RotatedRect 类

该类包含了中心点、尺寸 Size，以及旋转角度。

cv::Matx{1,2,3,4,6}{1,2,3,4,6}{f,d}

固定矩阵类，在编译前就确定了它的维度。别名，比如 Matx33f，Matx24d 等，里面有很多矩阵运算函数，比如矩阵求逆、求解线性系统等。该类并不是我们读取图像时得到的 cv::Mat，注意不要搞混。

cv::Complex 类

复数类，别名有 Complexf 和 Complexd。通过.re 和.im 来访问实部和虚部。

3 2 辅助对象类型

辅助对象类型主要有条件终止类 cv::TermCriteria、范围类 cv::Range、智能指针类 cv::Ptr、异常处理类 cv::Exception、类型检测类 cv::DataType<>、输入输出数组类 cv::InputArray 和 cv::OutputArray。

这些类，除了 cv::Ptr，我们用户平时很少会直接用到。智能指针的使用和 C++ 标准库的智能指针类似，会自动计数引用、自动管理内存和释放。

cv::Range 类相当于构建一个整数序列，和 python 中的 numpy.range 函数非常相似。

对于终止条件类，我们有时候需要判断程序何时退出，该类会使得程序控制更为便捷。

为了在运行中确定数据类型，有 cv::DataType<> 模板类。该类如何进行使用相对有些难懂，要想讲解清楚需要较多的篇幅。

cv::InputArray 和 cv::OutputArray 表示“代理类”，在很多例子中也很常见，它们一般作为参数或返回值。其中 InputArray 表示只读，而 OutputArray 可以进行任意读写。我们也可以使用 cv::noArray 表示该参数不使用输入数组。比如在 SIFT 算法实现中（xfeatures2d 模块），有：

```
void detectAndCompute(InputArray img, InputArray mask,
    std::vector<KeyPoint>& keypoints,
    OutputArray descriptors,
    bool useProvidedKeypoints = false) CV_OVERRIDE;
```

cv::InputArray 和 cv::OutputArray 只用于函数传递参数，所以如果声明全局或局部变量为这种类型是不可取的。很多时候我们并不会直接用到该类，但是由于其很重要，我们会在其他文章中详细解释。

四　矩阵的常用访问方式和操作

我们使用 OpenCV 时最常用的功能就是读取图像，然后访问其中的每个元素，由此，我们总绕不开 cv::Mat 类。但是很多人对该类总是一知半解，所以用起来很别扭。本章节会将基本操作进行详细介绍。

4 1　矩阵的创建

创建一个 3 行 4 列,，4 通道浮点数的矩阵：

```
cv::Mat m;
m.create(3,4,CV_32FC4);
```

我们可以打印 m.rows 和 m.cols 来查看矩阵有多少行多少列。这里和很多程序员的通常用法不同，注意当使用 width 和 height 定义图像长宽时，width 等于 cols，而 height 等于 rows（也就是说，矩阵的长等于一共有多少列，矩阵的宽等于一共有多少行）。

还有一种常见的定义方式，即使用 Size：

```
cv::Size size(3,4);
cv::Mat m2;
m2.create(size, CV_32FC4);
```

注意此时打印 m.rows 和 m.cols 输出 4 和 3。

4 2　矩阵的赋值

矩阵本质上是一个数据实体的头，它和数据实体是分开的，也就是说：

```
cv::Mat m;
m.create(3,4,CV_32FC4);
cv::Mat m2 = m1;
```

此时 m2 和 m1 拥有共同的数据区，因此当你修改了 m2 矩阵中的数据时，也相当于修改了 m1 矩阵中的数据。当两个 cv::Mat 都享用同一个数据区时，内存区的引用指针计数加 1，当所有引用都超过了范围而失效后（例如定义在局部块内的 Mat 运行完以后），智能指针自动释放内存。

想获得两份 cv::Mat 内的数据实体，可以使用 clone() 函数：

```
cv::Mat m2 = m1.clone();
```

4 3 矩阵元素的访问

我们以二维矩阵为例，来介绍矩阵元素的访问。

对于三通道浮点数矩阵，访问第 i 行第 j 列的三个元素：

```
Vec3f v = m.at<Vec3f>(i,j);
```

而对于单通道灰度图，则访问第 i 行第 j 列的元素是：

```
float v = m.at<float>(i,j);
```

还后一些其他访问方式，比如按行访问，用到可以再去查找。

4 4 稀疏矩阵类 cv::SparseMat 简介

如果矩阵中大量元素都是 0，则可以使用稀疏矩阵类 cv::SparseMat。

稀疏矩阵类涉及了不少稀疏矩阵的操作。

4 5 大型数组模板

主要有 cv::Mat_<> 和 cv::SparseMat_<> 这两类。

使用模板定义的矩阵访问会比较方便，比如：

```
cv::Mat_<Vec3f> m(3,4);
m.at(1,2) = cv::Vec3f(0.4f,0.5f,0.6f);
```

即不用再在 at 函数表明类型信息了。

4 6 矩阵函数

矩阵有很多函数，比如类似矩阵相加，求对数等基本操作；还有特征分解等操作。最好在使用前提前浏览一遍（不过有些时候我更倾向于使用 eigen 函数库）。

五 工具函数与小结

OpenCV 有很多专用功能函数，比如判断是否是异常值，比如返回不小于某数的最小的整数（向上取整）：

```
cvCeil()
cvIsNaN()
cv::error()
```

有些函数并不需要 OpenCV 名称空间，有些是需要的，这点需要注意。

在应用 OpenCV 之前，也需要尽量浏览一下这些工具函数，以便于后续使用和开发。关于更多工具函数的信息，可以从 [1] 中查找。

5 1 小结

本文只是一个简单的介绍，但是对于刚接触 OpenCV 的用户来说是足够的。对于一些具体的函数功能并没有做详细的解释，因为官网和索引的书目已经完全足够了。其实在网上找到的关于 OpenCV 的一些使用上的问题（比如怎么访问到某个元素、长宽和 rows），百分之四五十都是用户没有事先了解一下 OpenCV 就开始使用导致的，而事先了解一下 OpenCV 的基本结构就可以对如何使用 OpenCV 更加清晰明确。

对于想对 OpenCV 源码进行修改以及做贡献的人来说，这些内容又是远远不够的，我会在后文里介绍如何写出符合 OpenCV 风格的代码（本人目前并没有对 OpenCV 代码做任何贡献，但因为阅读和分析了大量 OpenCV 以及 OpenCV-Contrib 的源码，所以对如何写出具有 OpenCV 风格的代码的流程还算比较熟悉。关于 OpenCV 源码解读系列，我写过《Harris 角点检测原理与 OpenCV 源码解读》和《SIFT 算法原理与 OpenCV 源码解读》两本电子书）。

参考文献

[1] Kaehler A, Bradski G. Learning OpenCV 3: computer vision in C++ with the OpenCV library[M]. " O'Reilly Media, Inc.", 2016.

[2] Mordvintsev A, Abid K. Opencv-python tutorials documentation[J]. Obtenido de https://media. readthedocs. org/pdf/opencv-python-tutroals/latest/opencv-python-tutroals. pdf, 2014.

[3] https://docs.opencv.org/3.4.2/d1/dfb/intro.html

[4] https://docs.opencv.org/3.4.2/index.html