第 1 章 OpenCV 快速入门

1.1 OpenCV 源码结构

OpenCV 是计算机视觉中的经典库，具有跨平台和多语言支持特性，功能非常强大。

官网地址：https://opencv.org

参考文档：https://docs.opencv.org/master

论坛：http://answers.opencv.org

Github：https://github.com/opencv/opencv

提 Issue：https://github.com/opencv/opencv/issues

OpenCV 的代码结构如下图所示：

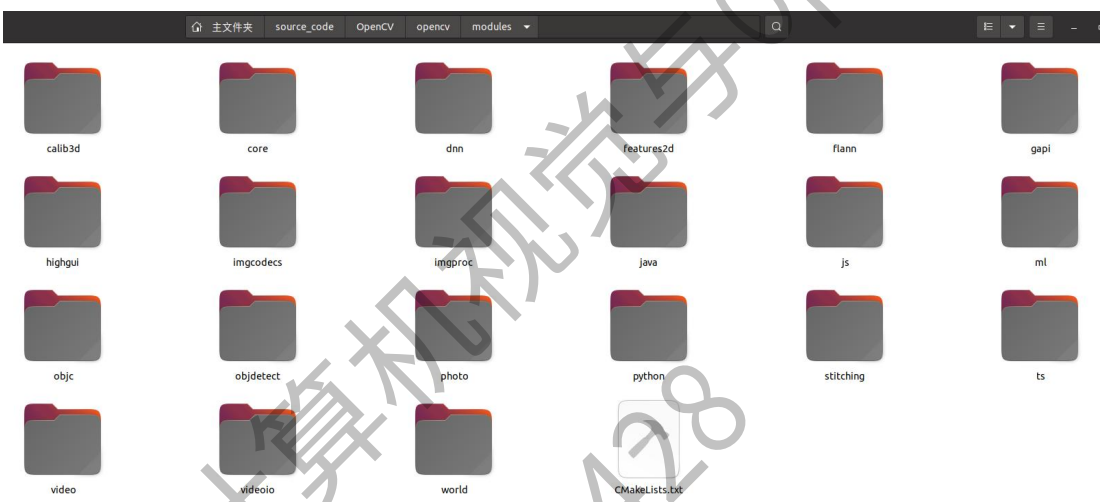
整个代码包括 9 个文件夹和若干文件，各文件夹以及文件的作用如下：

- 3rdparty：第三方库存放路径
- apps：工具存放文件夹
- cmake：存放 cmake 编译生成项目工程时的依赖文件；
- data：存放 OpenCV samples 中用到的资源文件；
- doc：存放文档相关文件；
- include：存放引用的头文件
- modules：源码模块，源码存放位置；
- platforms：存放交叉编译工具

- samples：存放官方提供的 sample；
- CMakeLists.txt：cmake 编译脚本。

1.2 OpenCV 常用模块

OpenCV 代码中包含 21 个文件夹，如下图所示：



各模块的功能如下。

- calib3d：该模块由相机校准（calibration）和三维重建（3d）两个部分组成，主要用于相机标定与三维视觉等；
- core：OpenCV 的内核模块，定义了基础数据结构与基础计算；
- dnn：该模块主要用于深度学习推理部署，不支持模型训练；
- features2d：该模块主要用于特征点处理，例如特征点检测与匹配等；
- flann：FLANN 为快速最近邻算法（Fast Library for Approximate Nearest Neighbors）的缩写，该模块包含快速近似最近邻搜索和聚类等功能；
- gapi：该模块对图像处理算法做了加速处理，不属于 OpenCV 的功能模块；
- highgui：该模块用于创建图像化界面操作，例如创建和操作图像显示窗口、鼠标与键盘事件处理，进度条等图像化交互操作；
- imgcodecs：该模块负责图像文件读写，如图像读取与保存；

- imgproc：该模块是 OpenCV 图像处理最重要的模块，主要功能如图像滤波、图像几何变换、直方图操作等；
- ml：该模块为机器学习模块，包含常见的机器学习算法，如支持向量机和随机森林等；
- objdetect：该模块主要用于图像目标检测，例如 Haar 特征检测等；
- photo：该模块主要负责照片处理，如照片修复和去噪等；
- stitching：该模块负责图像拼接，功能包括图像特征点寻找与匹配等图像拼接技术；
- video：该模块用于视频分析，如运动估计、背景分离等；
- videoio：该模块负责视频读写，主要视频文件的读取和写入。

1.3 扩展模块 opencv_contrib

OpenCV 在视觉算法中的功能非常强大，其中一个原因就是该算法库一直在与时俱进的更新最新的算法，对于具有专利的算法（如 SURF）以及一些还没有稳定的算法，OpenCV 会将其置于扩展模块中，这些扩展模块包含在 opencv_contrib 代码库中。对于稳定的算法，会被移到 OpenCV 主仓库代码中，因此需要谨慎的使用 opencv_contrib，因为不同版本的函数可能存在差异。

在 OpenCV 3.x 版本之后，opencv_contrib 就不再包含于 OpenCV 源码中，opencv_contrib 的源码可以在 Github 上下载，然后参与 OpenCV 源码编译，编译方法将在案例 5 中讲解。

opencv_contrib 的模块及其功能说明如下。

- alphamat：Alpha Matting 信息流算法；
- aruco：增强现实标记算法；
- barcode：条形码检测与解码方法；
- bgsegm：增强背景-前景分割算法；
- bioinspired：仿生学视觉模型和衍生工具；
- ccalib：用于三维重建的自定义校准模式；
- cudaarithm：CUDA 矩阵运算；
- cudabgsegm：CUDA 背景分割；

- cudacodec：CUDA 视频编解码；
- cudafeatures2d：CUDA 特征检测与描述；
- cudafilters：CUDA 图像滤波；
- cudaimgproc：CUDA 图像处理；
- cudalegacy：CUDA 传统支持；
- cudaobjdetect：CUDA 目标检测；
- cudaoptflow：CUDA 光流算法；
- cudastereo：CUDA 立体匹配；
- cudawarping：CUDA 图像扭曲；
- cudev：CUDA 设备层；
- cvv：计算机视觉程序交互式可视化调试的 GUI；
- datasets：用于处理不同数据集的框架；
- dnn_objdetect：基于 DNN 的目标检测；
- dnn_superres：基于 DNN 的超分；
- dpm：基于可变形零件的模型；
- face：人脸分析；
- freetype：使用 freetype/harfbuzz 绘制 UTF-8 字符串；
- fuzzy：基于模糊数学的图像处理；
- hdf：分层数据格式 I/O 例程；
- hfs：基于层次特征选择的图像分割方法；
- img_hash：该模块提供了不同的图像哈希算法的实现；
- intensity_transform：该模块提供了用于调整图像对比度的强度变换算法的实现；
- julia：OpenCV Julia 绑定；
- line_descriptor：用于从图像中提取线条的二进制描述符；
- mcc：Macbeth 图表模块；
- optflow：光流算法；

- ovis：OGRE 三维可视化器；
- phase_unwrapping：相位展开 API；
- plot：Mat 数据绘制函数；
- quality：图像质量分析 API；
- rapid：基于轮廓的三维目标跟踪；
- reg：图像配准；
- rgbd：RGB 深度处理；
- saliency：显著性 API；
- sfm：运动结构分析；
- shape：形状距离与匹配；
- stereo：立体匹配算法；
- structured_light：结构光 API；
- superres：超分模块；
- surface_matching：曲面匹配；
- text：场景文字检测与识别；
- tracking：追踪 API；
- videostab：视频稳定；
- viz：三维可视化器；
- wechat_qrcode：微信二维码检测器，用于检测和解析二维码；
- xfeatures2d：features2d 扩展模块；
- ximgproc：imgproc 扩展模块；
- xobjdetect：objdetect 扩展模块；
- xphoto：photo 扩展模块。

1.4 Windows 端 C++开发环境搭建

Windows 上进行 C++开发，常用的 IDE（集成开发工具）为 Visual Studio，该软件由

微软发布，最新版本为 Visual Studio 2019（Visual Studio 2022 正式版待发布）。本案例基于 Visual Studio 2019 开发，Visual Studio 2019 的安装方法如下。

安装前需要去官网下载安装文件，对于个人开发者，可以选择下载社区版 Community 2019，如图 1.6 所示。

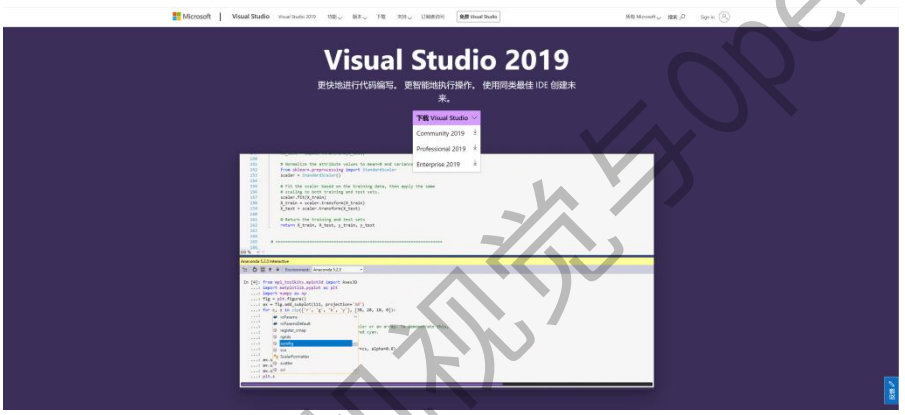


图 1.6

Community 2019 下载的文件名称为 vs_Community.exe，双击该文件进行安装，安装过程中可以选择下载安装工作负载（即不同开发环境）与单个组件，如图 1.7 所示。

图 1.7

Visual Studio 2019 支持 C++桌面开发，Python 开发，Node.js 开发等，可以选择自己需要的环境进行安装，本案例需要 C++桌面开发，勾选安装即可，安装完成重启即完成

了 Visual Studio 2019 的安装。

OpenCV 常用的两种开发语言是 C++和 Python，本节讲解在 Windows 下搭建 OpenCV C++语言开发环境，环境搭建方法有两种：安装官方发布的库文件和源码编译。

源码编译的方法参见 1.4 节，本案例介绍使用库文件安装的方法。使用库文件安装的方法进行环境搭建过程如下：

首先进入官网选择 Library->Releases，进入 Release 库文件包下载页面根据相应的环境下载库文件，如图 1.8 所示。

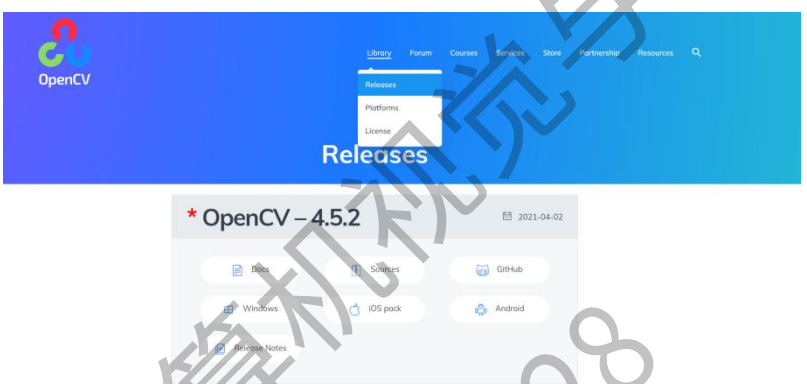


图 1.8

知识点：库文件分为 Release 和 Debug 两种形式，Release 为发行版本，其中不含调试信息，生成文件包时编译器会做优化，文件包较小；Debug 为调试版本，含调试信息，用于开发人员开发调试，正式发布产品中使用 Release 版本库文件。

本案例选择 OpenCV-4.5.2 版本，下载的文件名为 opencv-4.5.2-vc14_vc15.exe，双击该文件解压，如图 1.9 所示。

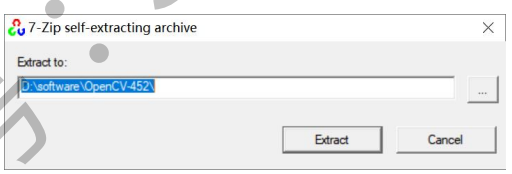


图 1.9

也可以右键选择文件，“解压到当前文件夹”，会在当前路径下解压得到文件夹 opencv，其中包含两个子文件夹：

- build，存放编译的二进制库文件；

- sources，存放当前版本的源码。

进入 build 文件夹，路径下有六个文件夹：

- bin，存放第三方库依赖文件；
- etc，存放资源文件；
- include，存放头文件；
- java，存放 Windows 系统 Java 库文件；
- python，存放 Windows 系统 Python 语言安装文件；
- x64 存放 Windows C++库文件，包含 vc14 和 vc15 两个版本。

知识点：vc 为 Visual Studio 的编译器，vc14 为 Visual Studio 2015 的编译器，vc15 为 Visual Studio 2017 的编译器，而 Visual Studio 2019 的编译器版本为 vc16。

对于 Visual Studio 2019，可以选用最新版本的 vc15 编译器对应的库文件，vc15 路径下包含两个文件夹：bin（存放动态库文件）和 lib（存放静态库文件），OpenCV 工具库调用最重要的三个文件为：头文件(存放于 include 文件夹)、动态库(opencv_world452.dll 或者 opencv_world452d.dll）以及静态库（opencv_world452.lib 或者 opencv_world452d.lib）。

下载解压完成后，需要配置 OpenCV 开发环境，并编写测试代码进行测试。

选择打开 Visual Studio 2019，创建新项目。创建项目可以直接选择创建“控制台应用”，创建的项目中有主程序，并默认输出“Hello World”，如图 1.10 所示。

图 1.10

配置新项目，设置项目名称和项目位置，解决方案名称可以使用默认的，和项目名称相同，如图 1.11 所示。

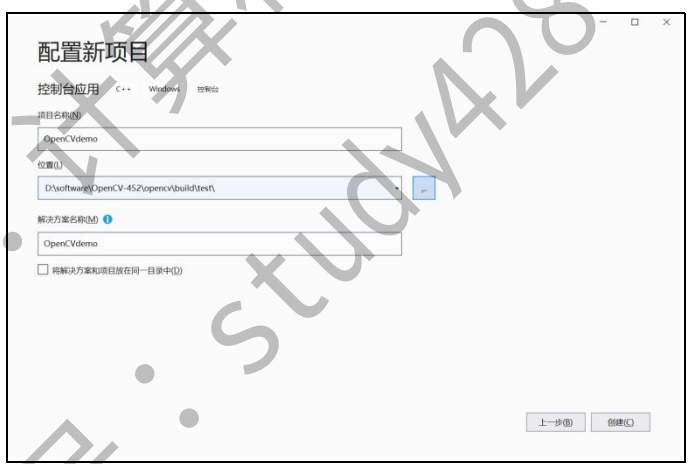


图 1.11

在项目中需要配置 OpenCV 库引用才能使用 OpenCV，环境配置需要配置三个方面内容：包含目录，库目录和链接器依赖项。包含目录与库目录的配置如图 1.12 所示。

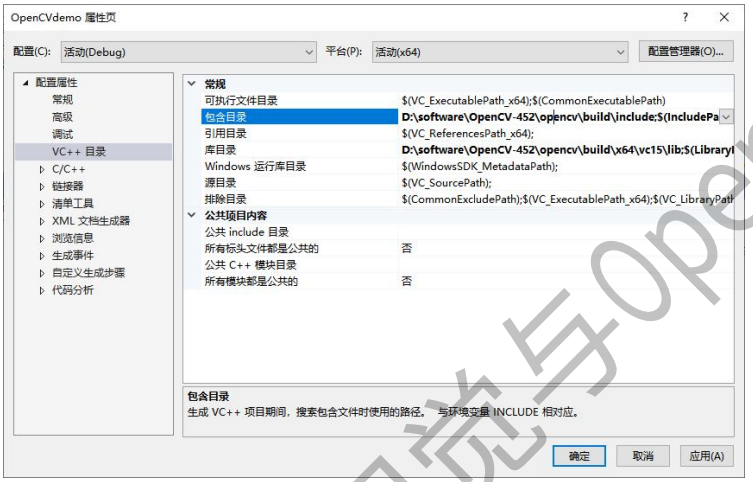


图 1.12

在“链接器-输入-附加依赖项”中配置附加依赖项，即静态库文件名称，如图 1.13 所示，配置时注意选择 Debug 或 Release 对应的静态库文件。

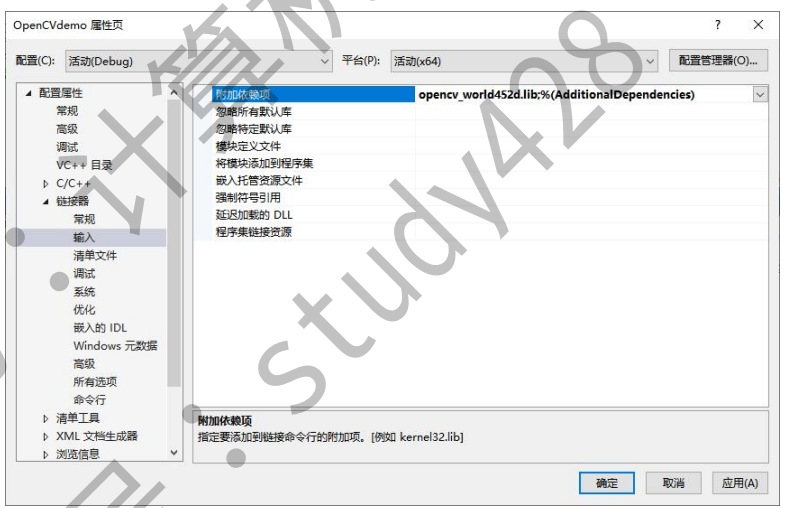


图 1.13

选择“应用”和“确定”后即完成配置，然后编写测试代码测试 OpenCV 库的使用，本案例的测试方法为读取一张图像，并将图像显示出来。

```
// OpenCVdemo.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。
```

```
#include <iostream>
#include "opencv2/opencv.hpp"

int main()
{
    cv::Mat img = cv::imread("src.jpg", 1);          //图像读取
    cv::imshow("测试图像显示", img);                  //图像显示
    cv::waitKey(0);                                  //等待操作
    cv::destroyWindow("测试图像显示");                //窗口对象销毁
}
```

注意：开发中需要根据是 Debug 模式还是 Release 模式选择不同的 OpenCV 库，Debug 模式配置 opencv_world452d.lib 库，而 release 则配置 opencv_world452.lib 库。

选择项目，右击“生成”，编译项目，编译完成结果如图 1.14 所示。

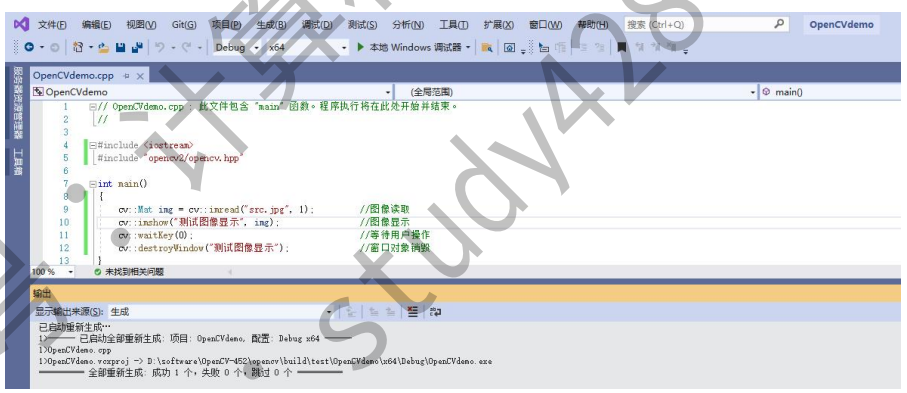


图 1.14

编译没有出错，此时可以点击图 1.14 中的“本地 Windows 调试器”或者按下 F5 键运行项目。如果没有做额外的配置，此时运行项目会弹出图 1.15 所示的错误框。

图 1.15

这个错误称为“运行时错误”，运行时错误的常见原因是动态库文件无法找到或者不匹配。这种问题的解决办法有如下两种：

- 第一种方法是将动态库路径配置到环境变量中。可以选择“计算机—>属性—>高级系统设置—>环境变量”，找到 Path 变量，将 OpenCV 文件包中的 opencv/build/x64/vc15/bin 路径配置到环境变量中，这样运行时就能找到动态库的路径。
- 第二种方法就是将动态库拷贝到生成的可执行文件所在路径中，如图 1.16 所示。

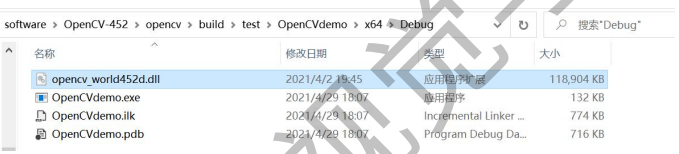


图 1.16

这两种办法是为了让可执行文件 OpenCVdemo.exe 能在搜索路径中找到 OpenCV 的动态库文件，当前路径和环境变量都是 Windows 系统中可执行文件链接时的搜索路径。

再次执行，程序正常运行，如图 1.17 所示，表明 Windows 系统中配置 OpenCV C++语言开发环境完成。

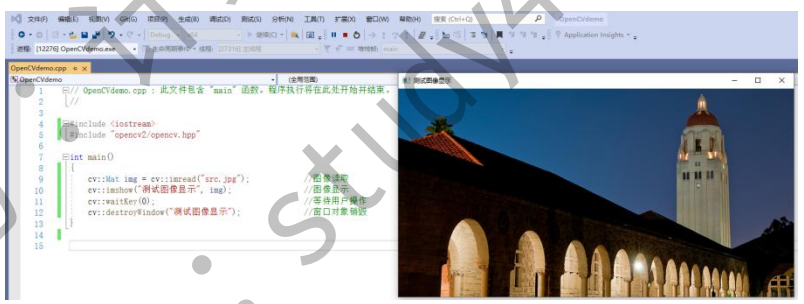


图 1.17

在 C++语言开发时，需要包含"opencv2/opencv.hpp"头文件，该头文件对所有模块的头文件做了引用：

```
#ifndef OPENCV_ALL_HPP
#define OPENCV_ALL_HPP
```

```
// 该头文件仅用于定义哪些模块参与了编译，定义了 HAVE_OPENCV_modulename 的值
#include "opencv2/opencv_modules.hpp"

// Core 模块为必需的模块
#include "opencv2/core.hpp"

// 通过 HAVE_OPENCV_modulename 检查可选模块的引入
#ifdef HAVE_OPENCV_CALIB3D
#include "opencv2/calib3d.hpp"
#endif
#ifdef HAVE_OPENCV_FEATURES2D
#include "opencv2/features2d.hpp"
#endif
#ifdef HAVE_OPENCV_DNN
#include "opencv2/dnn.hpp"
#endif
#ifdef HAVE_OPENCV_FLANN
#include "opencv2/flann.hpp"
#endif
#ifdef HAVE_OPENCV_HIGHGUI
#include "opencv2/highgui.hpp"
#endif
#ifdef HAVE_OPENCV_IMGCODECS
#include "opencv2/imgcodecs.hpp"
#endif
#ifdef HAVE_OPENCV_IMGPROC
#include "opencv2/imgproc.hpp"
#endif
#ifdef HAVE_OPENCV_ML
```

```
#include "opencv2/ml.hpp"
#endif
#ifdef HAVE_OPENCV_OBJDETECT
#include "opencv2/objdetect.hpp"
#endif
#ifdef HAVE_OPENCV_PHOTO
#include "opencv2/photo.hpp"
#endif
#ifdef HAVE_OPENCV_STITCHING
#include "opencv2/stitching.hpp"
#endif
#ifdef HAVE_OPENCV_VIDEO
#include "opencv2/video.hpp"
#endif
#ifdef HAVE_OPENCV_VIDEOIO
#include "opencv2/videoio.hpp"
#endif

#endif
```

如果对 OpenCV 结构比较熟悉，在使用时可以只包含对应的某个模块，如图像滤波功能调用可以只包含"opencv2/imgproc.hpp"头文件，在后续章节讲解每个模块时，都会讲解包含该模块对应的头文件，导读内容可以查看当前模块暴露的算法函数。