

Opencv入门第一课

Opencv简介

OpenCV（开源计算机视觉库：<http://opencv.org>）是一个包含数百种计算机视觉算法的开源库。

OpenCV 具有模块化结构，这意味着该功能包包含多个共享或静态库。以下模块可用：

- [核心功能 \(core\)](#) - 定义基本数据结构的紧凑模块，包括稠密的多维数组 `Mat` 和所有其他模块使用的基本功能。
- [图像处理 \(imgproc\)](#) - 一个图像处理模块，包括线性和非线性图像滤波、几何图像变换（调整大小、仿射和透视变换、通用的基于表的重映射）、色彩空间转换、直方图等。
- [视频分析 \(video\)](#) - 一个视频分析模块，包括运动估计、背景去除和对象跟踪算法。
- [相机校准和 3D 重建 \(calib3d\)](#) - 基本的多视图几何算法、单目和双目相机标定、对象位姿估计、双目关联算法和 3D重建相关的元素。
- [2D 特征框架 \(features2d\)](#) - 特征检测器、描述子和描述子匹配。
- [对象检测 \(objdetect\)](#) - 检测对象和预定义类的实例（例如，人脸、眼睛、杯子、人、汽车等）。
- [高层级GUI界面 \(highgui\)](#) - 易于使用的界面，具有简单的 UI 功能。
- [视频 I/O \(videoio\)](#) - 一个易于使用的视频捕获和视频编解码的接口。
- ... 其他一些辅助模块，例如 FLANN 和 Google 测试封装、Python封装等。

Windows 安装和配置(opencv 4.5.2)

准备工作

- opencv和opencv_contrib

[opencv](#)

[opencv_contrib](#)

这里由于安装第三方库时候需要源码编译 `opencv` 和 `opencv_contrib`，所以对于opencv我们直接下载源代码，不要选择预编译文件，除了占空间没有什么好处。

 opencv-4.5.2-android-sdk.zip	232 MB
 opencv-4.5.2-dldt-2021.3-vc16-avx2-debug.7z	192 MB
 opencv-4.5.2-dldt-2021.3-vc16-avx2.7z	161 MB
 opencv-4.5.2-dldt-2021.3-vc16-avx2.zip	246 MB
 opencv-4.5.2-docs.zip	85.4 MB
 opencv-4.5.2-ios-framework.zip	176 MB
 opencv-4.5.2-vc14_vc15.exe	213 MB
 Source code (zip)	我们需要的
 Source code (tar.gz)	

注意opencv版本要和对应的opencv_contrib版本匹配

- [MinGW64](#)
















首先简单介绍一下 `MinGW`

MinGW 的全称是：Minimalist GNU on Windows 。它实际上是将经典的开源C/C++编译器 GCC/G++ 移植到了 Windows 平台下，并且包含了 Win32API，因此可以将源代码编译为可在 Windows 中运行的可执行程序。而且还可以使用一些 Windows 不具备的，Linux平台下的开发工具。一句话来概括：MinGW 就是 GCC/G++ 的 Windows 版本。

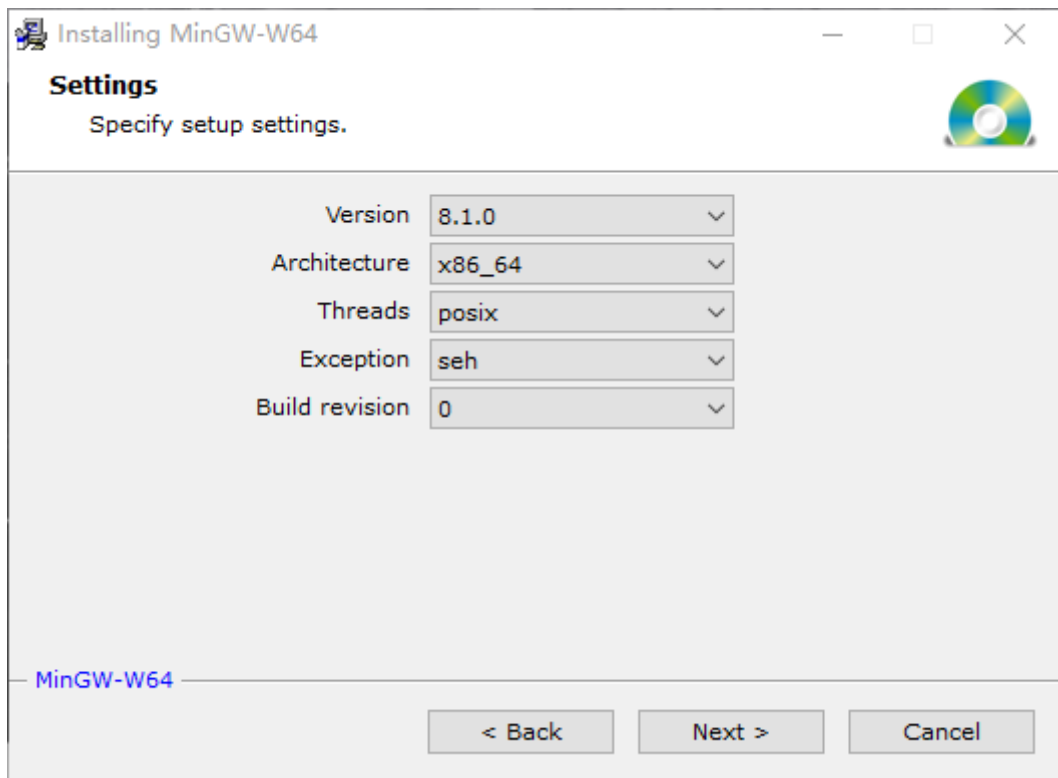
以上是 MinGW 的介绍，MinGW-w64 与 MinGW 的区别在于 MinGW 只能编译生成32位可执行程序，而 MinGW-w64 则可以编译生成 64位 或 32位 可执行程序。正因为如此，MinGW 现已被 MinGW-w64 所取代

这里选择 MinGW64 配合其他 IDE 而不是直接上 Visual Studio 的原因主要是 Visual Studio 本身过于庞大，而且其中绝大部分功能我们基本用不上，并且 Visual Studio 会默认被安装到 C 盘，即使你选择了其他盘还是会有很大一部分文件强制安装到 C 盘，十分不友好。卸载的时候更是天大的难题，不重装系统的情况下是基本不可能卸载干净的。当然如果你的项目中要是使用到了第三方库，而他是由 Visual Studio 编译出来的，那么就只能安装 VS 了。后面也会简单介绍一下 Visual Studio 的安装方法。

使用提供的下载地址下载如下的 MinGW-w64-builds

Pre-built toolchains and packages					
	Version	Host	GCC / Mingw-w64 Version	Languages	Additional Software in Package Manager
 Arch Linux	Arch Linux		8.2.0/5.0.4	Ada, C, C++, Fortran, Obj-C, Obj-C++	305, full list: Show
 Cygwin	Rolling	Windows 	5.4.0/5.0.2	Ada, C, C++, Fortran, Obj-C	5 (bzip2, libgcrypt, libgpg-error, minizip, xz, zlib)
 Debian	Debian 7 (Wheezy)		4.6.3/2.0.3	Ada, C, C++, Fortran, Obj-C, Obj-C++, OCaml	2 (gdb, nsis)
	Debian 8 (Jessie)		4.9.1/3.2.0		9 (gdb, libassuan, libgcrypt, libgpg-error, libksba, libnptl, nsis, win-iconv, zlib)
	Debian 9 (Stretch)		6.3.0/5.0.0		
	Debian 10 (Buster)		8.3.0/6.0.0		
 Fedora	Fedora 19		4.8.1/?	Ada, C, C++, Fortran, Obj-C, Obj-C++	149, full list: Show
 MacPorts	Rolling	macOS 	8.2.0/5.0.4	C, C++, Fortran, Obj-C, Obj-C++	1 (nsis)
 MingW-W64-builds	Rolling	Windows 	7.2.0/5.0.3	C, C++, Fortran	4 (gdb, libiconv, python, zlib)
 Msys2	Rolling	Windows 	9.2.0/trunk	Ada, C, C++, Fortran, Obj-C, Obj-C++, OCaml	many
 Ubuntu	12.04 Precise Pangolin		4.6.3/2.0.1	Ada, C, C++, Fortran, Obj-C, Obj-C++, OCaml	2 (nsis, gdb)
	14.04 Trusty Tahr		4.8.2/3.1.0		
	14.10 Utopic Unicorn		4.9.1/3.1.0		
	15.04 Vivid Vervet		4.9.2/3.2.0		
	15.10 Willy Werewolf		4.9.2/4.0.2		3 (nsis, gdb, zlib)
	16.04 Xenial Xerus		5.3.1/4.0.4		
 Win-Builds	1.5	Windows  Linux 	4.8.3/3.3.0	C, C++	91, full list: Show

按照 Windows 安装软件的通用方法安装，注意要记住安装路径



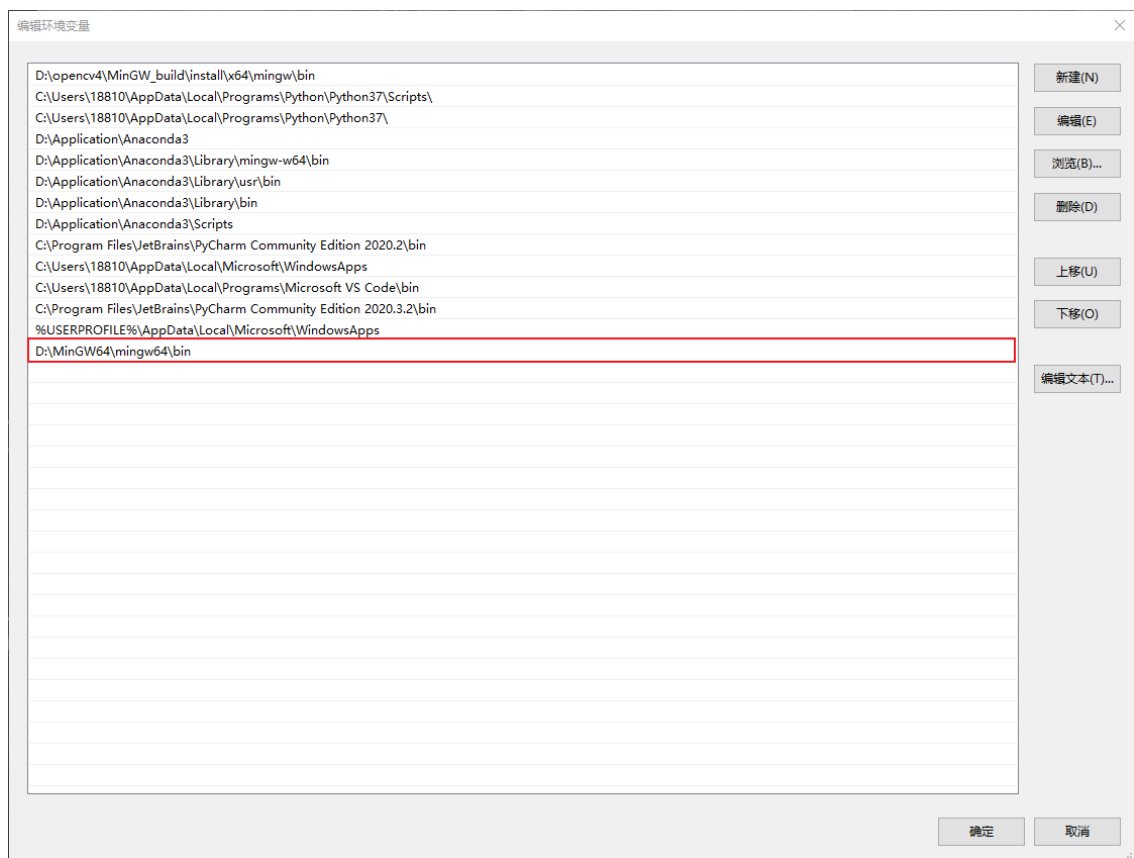
这里有一些选项，这里简单介绍一下各个选项的含义，不想了解的可以直接选择我的配置

- Architecture 是指电脑系统是 32位 还是 64位，根据你的电脑系统做出对应选择。
- Thread 指程序运行时使用的线程模型，分为posix和Windows两种。如果只开发在Windows下的程序，就选择Win32，性能会高一点，但是对于C++11的多线程标准库是不支持的，只能使用WinAPI的多线程。如果需要跨平台编译就选择posix，支持C++线程库。这只是一个不是很准确的区分，具体的区别有兴趣的可以详细研究一下。
- Exception是异常处理模型，异常处理在开发中非常重要，在开发的过程中，大部分的时间会耗在处理各种异常情况上。如果你之前选择了 64位，则这里有两个异常处理模型供你选择，seh 是新发明的，而 sjlj 则是古老的。seh 性能比较好，但不支持 32位。sjlj 稳定性好，支持 32位。
- Build revision 不可选择的选项，不用管。

之后就是标准的Windows应用安装流程。

安装之后需要配置环境变量，也就是告诉系统你的 MinGW 在哪。环境变量选择的方式为：在我的电脑右键选择属性->选择高级系统设置->环境变量。

这里环境变量分为两种：用户变量和系统变量。如果你的电脑只有当前用户需要用到 MinGW，那就选择用户变量进行配置，如果多用户同时使用则选择系统变量进行配置。直接在 Path 这个路径上添加 MinGW 安装路径中的 bin 路径即可。



- CMake

注意版本号要高于opencv要求的最低版本，对于opencv来说最低要求的Cmake版本定义在源代码目录下的 `cmake/OpenCVMinDepVersions.cmake` 中，我这里是3.5.1

CMake配置

打开 cmake-gui 软件，选择源代码目录和构建文件生成目录



之后点击 Configure 进行第一次配置，我这里已经配置过，所以中间的选项框内有东西，如果是第一次配置的话，里面应该是空白的。点击之后弹出一个构建文件格式选择框，我们这里选 `MinGW Makefiles`，点击 Finish。

第一次配置完成之后，我们需要更改一些选项

- **BUILD_opencv_world**
这个选项的目的是将所有模块的库文件合并成一个大的库文件，在链接时候会方便一点。对于cmake工程可能这么做的好处不明显，但是对于visual studio工程用过的人都知道添加附加依赖项时候一个一个复制名称的痛苦
- **OPENCV_ENABLE_NONFREE**
这个选项的目的是让你可以使用带专利的比如SURF角点之类的第三方模块中的功能
- **OPENCV_EXTRA_MODULES_PATH** (D:\opencv4\opencv_contrib-4.5.2\modules)
指定你的第三方模块的路径，具体的路径为 `opencv_contrib/modules` 目录

我这里只提了对于安装第三方模块有用的选项，至于其他的有兴趣的可以自己摸索。之后再次点击 **Configure**，这里可能会出现由于下载失败而配置失败的情况，这个问题在Ubuntu系统配置时也会碰到，我们在之后的Ubuntu配置中再讲。

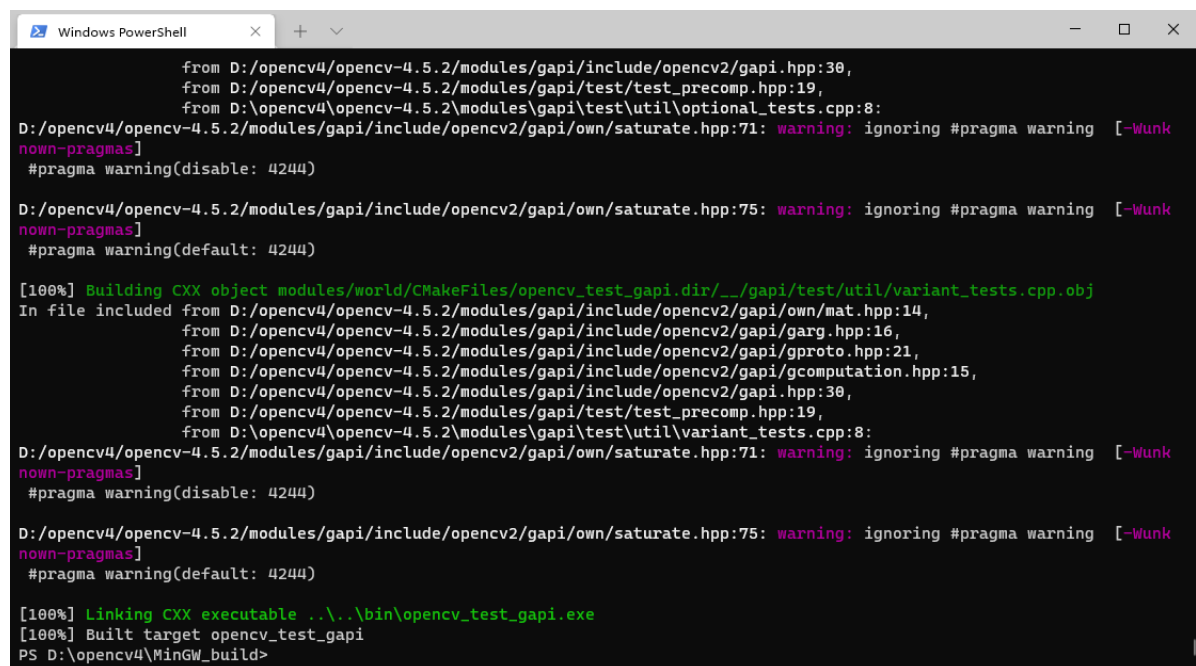
此处假设你已经第二次配置成功，那么接下来点击生成即可

MinGW编译

打开 **windows** 下的命令行工具，进入到你的构建目录中，输入以下命令

```
mingw32-make -j8
```

接下来就是漫长的等待，编译成功后你会看到如下显示



```
from D:/opencv4/opencv-4.5.2/modules/gapi/include/opencv2/gapi.hpp:30,  
from D:/opencv4/opencv-4.5.2/modules/gapi/test/test_precomp.hpp:19,  
from D:/opencv4/opencv-4.5.2/modules/gapi/test/util/optional_tests.cpp:8:  
D:/opencv4/opencv-4.5.2/modules/gapi/include/opencv2/gapi/own/saturate.hpp:71: warning: ignoring #pragma warning [-Wunk  
nown-pragmas]  
#pragma warning(disable: 4244)  
  
D:/opencv4/opencv-4.5.2/modules/gapi/include/opencv2/gapi/own/saturate.hpp:75: warning: ignoring #pragma warning [-Wunk  
nown-pragmas]  
#pragma warning(default: 4244)  
  
[100%] Building CXX object modules/world/CMakeFiles/opencv_test_gapi.dir/__/gapi/test/util/variant_tests.cpp.obj  
In file included from D:/opencv4/opencv-4.5.2/modules/gapi/include/opencv2/gapi/own/mat.hpp:14,  
from D:/opencv4/opencv-4.5.2/modules/gapi/include/opencv2/gapi/garg.hpp:16,  
from D:/opencv4/opencv-4.5.2/modules/gapi/include/opencv2/gapi/gproto.hpp:21,  
from D:/opencv4/opencv-4.5.2/modules/gapi/include/opencv2/gapi/gcomputation.hpp:15,  
from D:/opencv4/opencv-4.5.2/modules/gapi/include/opencv2/gapi.hpp:30,  
from D:/opencv4/opencv-4.5.2/modules/gapi/test/test_precomp.hpp:19,  
from D:/opencv4/opencv-4.5.2/modules/gapi/test/util/variant_tests.cpp:8:  
D:/opencv4/opencv-4.5.2/modules/gapi/include/opencv2/gapi/own/saturate.hpp:71: warning: ignoring #pragma warning [-Wunk  
nown-pragmas]  
#pragma warning(disable: 4244)  
  
D:/opencv4/opencv-4.5.2/modules/gapi/include/opencv2/gapi/own/saturate.hpp:75: warning: ignoring #pragma warning [-Wunk  
nown-pragmas]  
#pragma warning(default: 4244)  
  
[100%] Linking CXX executable ..\..\bin\opencv_test_gapi.exe  
[100%] Built target opencv_test_gapi  
PS D:\opencv4\MinGW_build>
```

之后我们执行

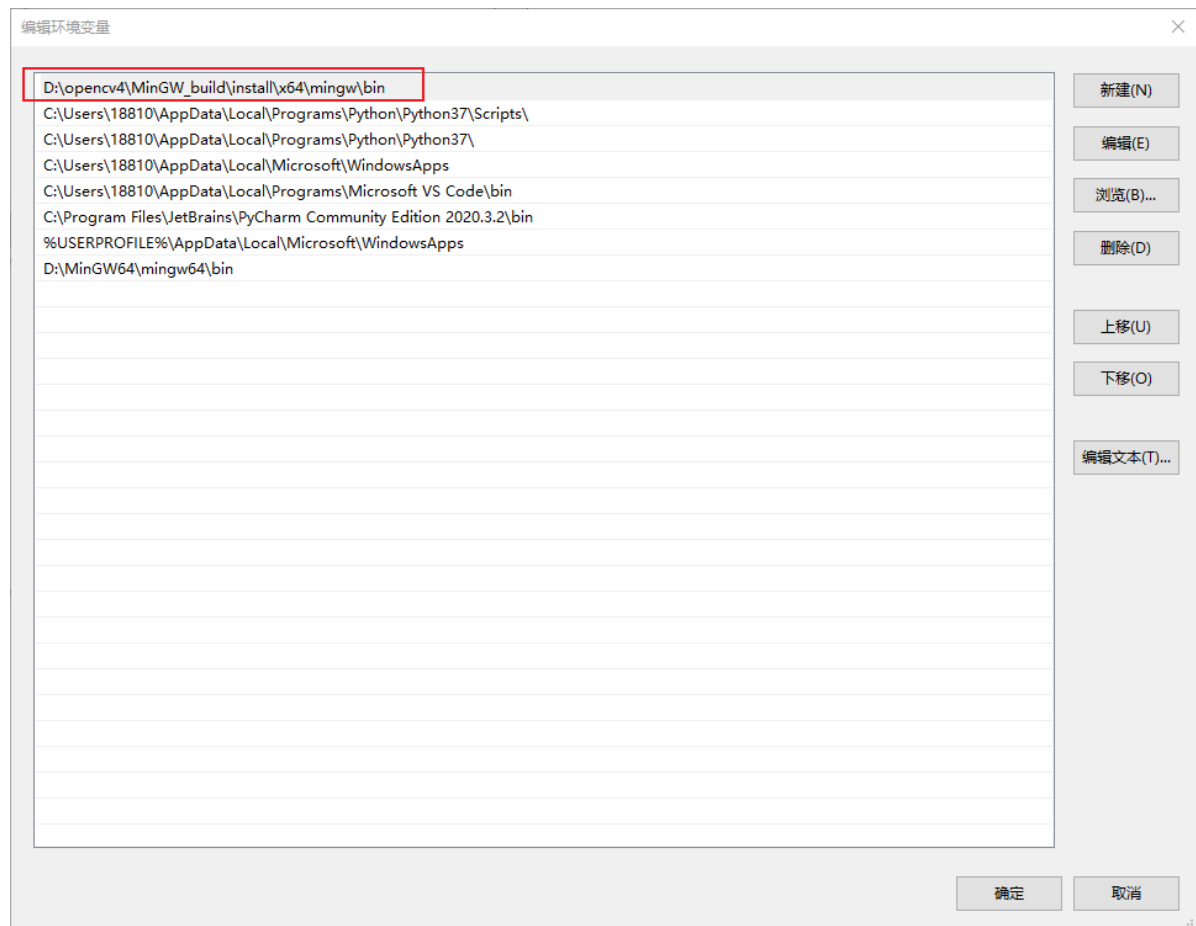
```
mingw32-make install
```

完成后，你会在构建目录下发现 `install` 文件夹，其中包括所有我们需要的文件

此电脑 > 新加卷 (D:) > opencv4 > MinGW_build > install				
名称	修改日期	类型	大小	
bin	2021/7/13 23:17	文件夹		
etc	2021/7/13 15:56	文件夹		
include	2021/7/13 15:56	文件夹		
x64	2021/7/13 15:56	文件夹		
LICENSE	2021/4/2 14:23	文件	12 KB	
OpenCVConfig.cmake	2021/7/13 14:51	CMAKE 文件	7 KB	
OpenCVConfig-version.cmake	2021/7/13 14:51	CMAKE 文件	1 KB	
setup_vars_opencv4.cmd	2021/7/13 22:28	Windows 命令脚本	1 KB	

环境变量配置

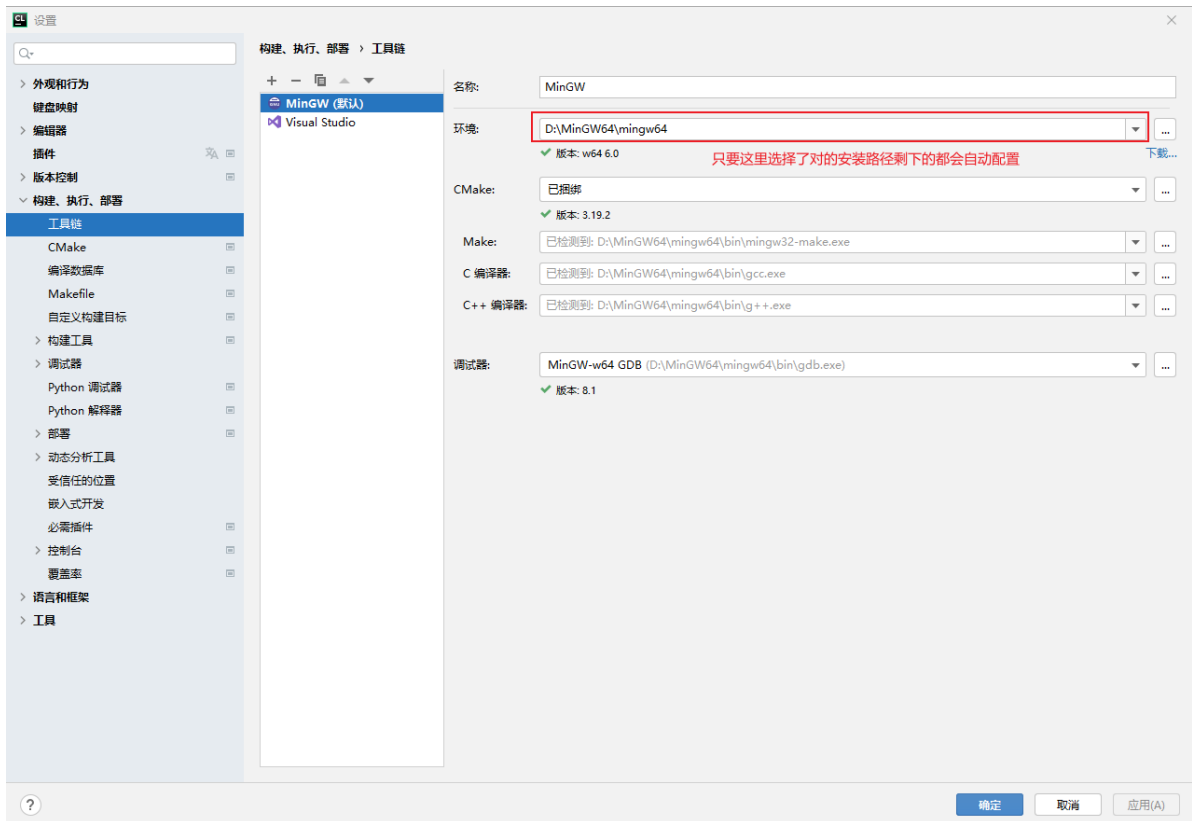
接下来我们需要将 opencv 配置到我们的环境变量中，如下图所示



到此，我们在Windows下的安装就完成了。

Clion开发配置

接下来，让我们来测试一下我们的 opencv 库是否可以正常运行，我这里选择使用 clion，当然你可以选择任何你喜欢的IDE。首先需要配置 clion 的构建工具链



Windows下程序演示

这里我们选择示例程序中的 `kmeans.cpp`

```
#include "opencv2/highgui.hpp"
#include "opencv2/core.hpp"
#include "opencv2/imgproc.hpp"
#include <iostream>

using namespace cv;
using namespace std;

// static void help()
// {
//     cout << "\nThis program demonstrates kmeans clustering.\n"
//           "It generates an image with random points, then assigns a random
//           number of cluster\n"
//           "centers and uses kmeans to move those cluster centers to their
//           representative location\n"
//           "Call\n"
//           "./kmeans\n" << endl;
// }

int main( int /*argc*/, char** /*argv*/ )
{
    const int MAX_CLUSTERS = 5;
    Scalar colorTab[] =
    {
        scalar(0, 0, 255),
        scalar(0,255,0),
        scalar(255,100,100),
        scalar(255,0,255),
        scalar(0,255,255)
    }
```

```

};

Mat img(500, 500, CV_8UC3);
RNG rng(12345);

for(;;)
{
    int k, clusterCount = rng.uniform(2, MAX_CLUSTERS+1);
    int i, sampleCount = rng.uniform(1, 1001);
    Mat points(sampleCount, 1, CV_32FC2), labels;

    clusterCount = MIN(clusterCount, sampleCount);
    std::vector<Point2f> centers;

    /* generate random sample from multigaussian distribution */
    for( k = 0; k < clusterCount; k++ )
    {
        Point center;
        center.x = rng.uniform(0, img.cols);
        center.y = rng.uniform(0, img.rows);
        Mat pointChunk = points.rowRange(k*sampleCount/clusterCount,
                                           k == clusterCount - 1 ? sampleCount
                                           :
                                           (k+1)*sampleCount/clusterCount);
        rng.fill(pointChunk, RNG::NORMAL, Scalar(center.x, center.y),
                  Scalar(img.cols*0.05, img.rows*0.05));
    }

    randShuffle(points, 1, &rng);

    double compactness = kmeans(points, clusterCount, labels,
                                TermCriteria( TermCriteria::EPS+TermCriteria::COUNT, 10, 1.0),
                                3, KMEANS_PP_CENTERS, centers);

    img = Scalar::all(0);

    for( i = 0; i < sampleCount; i++ )
    {
        int clusterIdx = labels.at<int>(i);
        Point ipt = points.at<Point2f>(i);
        circle( img, ipt, 2, colorTab[clusterIdx], FILLED, LINE_AA );
    }
    for( i = 0; i < (int)centers.size(); ++i)
    {
        Point2f c = centers[i];
        circle( img, c, 40, colorTab[i], 1, LINE_AA );
    }
    cout << "Compactness: " << compactness << endl;

    imshow("clusters", img);

    char key = (char)waitkey();
    if( key == 27 || key == 'q' || key == 'Q' ) // 'ESC'
        break;
}

return 0;
}

```


具体代码的含义你不需要关心。此外我们需要编写对应的 `CMakeLists.txt` 文件

```
cmake_minimum_required(VERSION 3.19)
project(opencv_test)
set(OpenCV_DIR "D:/opencv4/MingW_build/install/x64/mingw/lib")
set(CMAKE_CXX_STANDARD 14)
set(CMAKE_BUILD_TYPE Debug)

find_package(OpenCV 4.5.2 REQUIRED)

include_directories(${OpenCV_INCLUDE_DIRS})

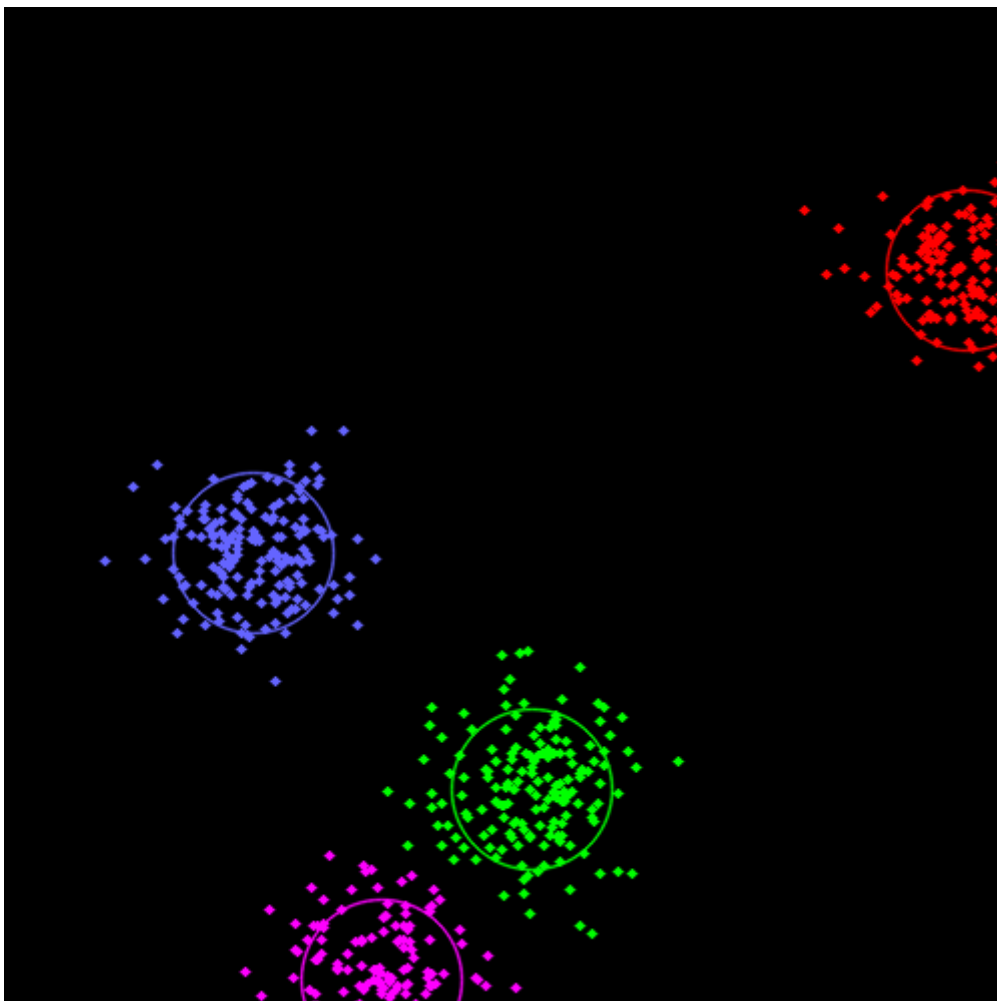
add_executable(${PROJECT_NAME} kmeans.cpp)

target_link_libraries(${PROJECT_NAME} ${OpenCV_LIBS})
```

最重要的是这一句

```
set(OpenCV_DIR "D:/opencv4/MingW_build/install/x64/mingw/lib")
```

在这个目录下有 `opencvConfig.cmake` 文件，他告诉了 `CMake` 去哪里寻找相关的头文件和库文件。如果你的 `opencv` 安装成功了，那么 `build` 并执行之后你会看到如下效果



补充Visual Stiduo安装方法


- CMake配置

同上述配置方法，注意要勾选 64 位

?

×

←



Specify the generator for this project

Visual Studio 15 2017

Optional platform for generator(if empty, generator uses: Win32)

x64

Optional toolset to use (argument to -T)

☒ Use default native compilers

☐ Specify native compilers

☐ Specify toolchain file for cross-compiling

☐ Specify options for cross-compiling

Finish

Cancel

- 生成 visual stiduo解决方案

- 编译

选定要生成的项目配置(K):

项目	配置	平台	解决方案配置	生成
ade	Debug	x64	Debug x64	<input type="checkbox"/>
ade	Release	x64	Release x64	<input type="checkbox"/>
ALL_BUILD	Debug	x64	Debug x64	<input checked="" type="checkbox"/>
ALL_BUILD	Release	x64	Release x64	<input checked="" type="checkbox"/>
gen_opencv_js...	Debug	x64	Debug x64	<input type="checkbox"/>
gen_opencv_js...	Release	x64	Release x64	<input type="checkbox"/>
gen_opencv_o...	Debug	x64	Debug x64	<input type="checkbox"/>
gen_opencv_o...	Release	x64	Release x64	<input type="checkbox"/>
gen_opencv_o...	Debug	x64	Debug x64	<input type="checkbox"/>
gen_opencv_o...	Release	x64	Release x64	<input type="checkbox"/>
gen_opencv_o...	Debug	x64	Debug x64	<input type="checkbox"/>
gen_opencv_o...	Release	x64	Release x64	<input type="checkbox"/>

生成(B)

重新生成(R)

清理(C)

全选(S)

撤消全选(D)

关闭

- 项目配置

Linux安装和配置

准备工作

- opencv和opencv_contrib

[opencv](#)

[opencv_contrib](#)

如上

- CMake

首先查询下你当前的 cmake 版本是否符合编译 OpenCV 的要求，如果不满足则需要对 cmake 进行升级，具体的教程可以参考[cmake升级](#)

- gcc(g++)

只需要执行以下命令就可以安装包括cmake和g++/gcc在内的一系列ubuntu系统下编译所需要的工具

```
sudo apt-get install build-essential
```

一般来说如果你的Ubuntu版本符合主流，比如在这个时间点版本大于等于16.04，那么默认的gcc/g++版本对于绝大多数的库来说是足够的，到目前为止需要更换gcc/g++版本的情况我只在安装 cuda 的时候遇到过，所以这里不介绍gcc/g++版本升级的问题，有兴趣的可以自行了解。

编译安装

- cmake-gui 配置

具体的方法与 windows 下一致，这里我们需要注意两点

1. 模块文件下载失败如何解决，对于 opencv 来说，当你遇到了下载失败的情况，你需要首先找到构建目录下的 CMakeDownloadLog.txt。当存在下载失败的情况时你的文件应该长得是这样的



```
Open CMakeDownloadLog.txt
~/opencv4/build

#use_cache "/home/teamo/opencv4/opencv-4.5.2/.cache"
#match_hash_in_cmake_cache "OCV_DOWNLOAD_IPP_ICV_HASH_3rdparty_ippicv_ippicv_2020_lnx_intel64_20191018_general_tgz"
#match_hash_in_cmake_cache "OCV_DOWNLOAD_ADE_HASH_3rdparty_ade_v0_1_if_zip"
#do_copy "detect.caffemodel" "238e2b2d6f3c18d6c3a30de0c31e23cf" "https://raw.githubusercontent.com/WeChatCV/opencv_3rdparty/a8b69ccc738421293254aec5ddb38bd523503252/detect.caffemodel" "/home/teamo/opencv4/build/downloads/wechat_qrcode"
#missing "/home/teamo/opencv4/build/downloads/wechat_qrcode/detect.caffemodel"
#check_md5 "/home/teamo/opencv4/opencv-4.5.2/.cache/wechat_qrcode/238e2b2d6f3c18d6c3a30de0c31e23cf-detect.caffemodel"
#mismatch_md5 "/home/teamo/opencv4/opencv-4.5.2/.cache/wechat_qrcode/238e2b2d6f3c18d6c3a30de0c31e23cf-detect.caffemodel"
"d41d8cd98f00b204e9800998ecf8427e"
#delete "/home/teamo/opencv4/opencv-4.5.2/.cache/wechat_qrcode/238e2b2d6f3c18d6c3a30de0c31e23cf-detect.caffemodel"
#cmake_download "/home/teamo/opencv4/opencv-4.5.2/.cache/wechat_qrcode/238e2b2d6f3c18d6c3a30de0c31e23cf-detect.caffemodel" "https://raw.githubusercontent.com/WeChatCV/opencv_3rdparty/a8b69ccc738421293254aec5ddb38bd523503252/detect.caffemodel"
#try 1
# getaddrinfo(3) failed for raw.githubusercontent.com:443
# Couldn't resolve host 'raw.githubusercontent.com'
# Closing connection 0
#

#do_copy "detect.prototxt" "6fb4976b32695f9f5c6305c19f12537d" "https://raw.githubusercontent.com/WeChatCV/opencv_3rdparty/a8b69ccc738421293254aec5ddb38bd523503252/detect.prototxt" "/home/teamo/opencv4/build/downloads/wechat_qrcode"
#missing "/home/teamo/opencv4/build/downloads/wechat_qrcode/detect.prototxt"
#check_md5 "/home/teamo/opencv4/opencv-4.5.2/.cache/wechat_qrcode/6fb4976b32695f9f5c6305c19f12537d-detect.prototxt"
#mismatch_md5 "/home/teamo/opencv4/opencv-4.5.2/.cache/wechat_qrcode/6fb4976b32695f9f5c6305c19f12537d-detect.prototxt"
"d41d8cd98f00b204e9800998ecf8427e"
#delete "/home/teamo/opencv4/opencv-4.5.2/.cache/wechat_qrcode/6fb4976b32695f9f5c6305c19f12537d-detect.prototxt"
#cmake_download "/home/teamo/opencv4/opencv-4.5.2/.cache/wechat_qrcode/6fb4976b32695f9f5c6305c19f12537d-detect.prototxt" "https://raw.githubusercontent.com/WeChatCV/opencv_3rdparty/a8b69ccc738421293254aec5ddb38bd523503252/detect.prototxt"
#try 1
# getaddrinfo(3) failed for raw.githubusercontent.com:443
# Couldn't resolve host 'raw.githubusercontent.com'
# Closing connection 0
#

#do_copy "sr.caffemodel" "cbfcd60361a73beb8c583eea7e8e6664" "https://raw.githubusercontent.com/WeChatCV/opencv_3rdparty/a8b69ccc738421293254aec5ddb38bd523503252/sr.caffemodel" "/home/teamo/opencv4/build/downloads/wechat_qrcode"
#missing "/home/teamo/opencv4/build/downloads/wechat_qrcode/sr.caffemodel"
#check_md5 "/home/teamo/opencv4/opencv-4.5.2/.cache/wechat_qrcode/cbfcd60361a73beb8c583eea7e8e6664-sr.caffemodel"
#mismatch_md5 "/home/teamo/opencv4/opencv-4.5.2/.cache/wechat_qrcode/cbfcd60361a73beb8c583eea7e8e6664-sr.caffemodel"
"d41d8cd98f00b204e9800998ecf8427e"
#delete "/home/teamo/opencv4/opencv-4.5.2/.cache/wechat_qrcode/cbfcd60361a73beb8c583eea7e8e6664-sr.caffemodel"
#cmake_download "/home/teamo/opencv4/opencv-4.5.2/.cache/wechat_qrcode/cbfcd60361a73beb8c583eea7e8e6664-sr.caffemodel" "https://raw.githubusercontent.com/WeChatCV/opencv_3rdparty/a8b69ccc738421293254aec5ddb38bd523503252/sr.caffemodel"
#try 1
# getaddrinfo(3) failed for raw.githubusercontent.com:443
# Couldn't resolve host 'raw.githubusercontent.com'
# Closing connection 0
#
```

下面我们针对其中一个失败问题来分析一下

```
#do_copy "detect.prototxt" "6fb4976b32695f9f5c6305c19f12537d" "https://raw.githubusercontent.com/WeChatCV/opencv_3rdparty/a8b69ccc738421293254aec5ddb38bd523503252/detect.prototxt" "/home/teamo/opencv4/build/downloads/wechat_qrcode"
#missing "/home/teamo/opencv4/build/downloads/wechat_qrcode/detect.prototxt"
#check_md5 "/home/teamo/opencv4/opencv-4.5.2/.cache/wechat_qrcode/6fb4976b32695f9f5c6305c19f12537d-detect.prototxt"
#mismatch_md5 "/home/teamo/opencv4/opencv-4.5.2/.cache/wechat_qrcode/6fb4976b32695f9f5c6305c19f12537d-detect.prototxt"
"d41d8cd98f00b204e9800998ecf8427e"
#delete "/home/teamo/opencv4/opencv-4.5.2/.cache/wechat_qrcode/6fb4976b32695f9f5c6305c19f12537d-detect.prototxt"
#make_download "/home/teamo/opencv4/opencv-4.5.2/.cache/wechat_qrcode/6fb4976b32695f9f5c6305c19f12537d-detect.prototxt" "https://raw.githubusercontent.com/WeChatCV/opencv_3rdparty/a8b69ccc738421293254aec5ddb38bd523503252/detect.prototxt"
#try 1
# getaddrinfo(3) failed for raw.githubusercontent.com:443
# Couldn't resolve host 'raw.githubusercontent.com'
# Closing connection 0
#
```

其中红色的为这个文件的地址，蓝色的为文件下载后存放的位置。知道了这个问题就很好解决了，我们只需要手动到这个地址下载文件并放到对应的文件夹下即可。

2.  我们需要勾选上这个选项，目的是为了后续的环境变量配置。

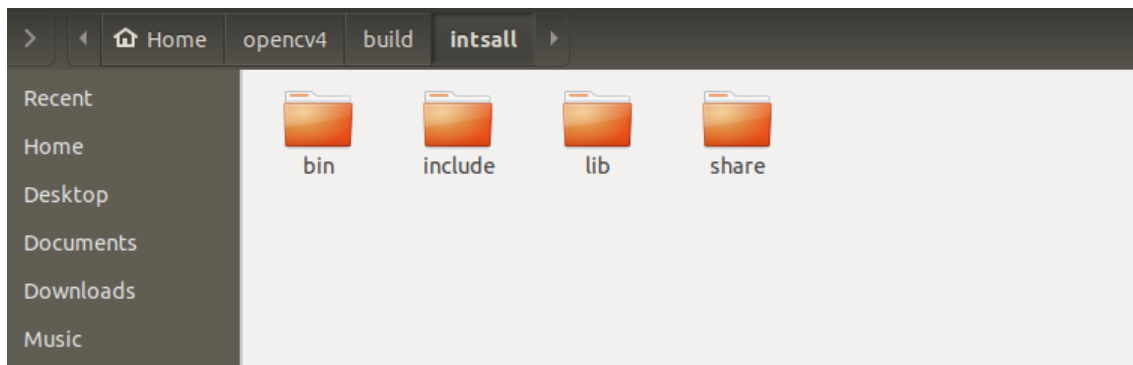
3.  需要手动指定一下安装路径，不然默认会安装到系统路径中覆盖掉之前的版本。

- g++ 编译

在构建目录下执行以下命令

```
make -j8
make install
```

执行结束后，你会在你指定的安装目录下看到以下文件



环境变量配置

Ubuntu 下的环境变量配置和 Windows 下稍有不同，你需要在 `~/.bashrc` 文件末尾添加以下几行

```
export
PKG_CONFIG_PATH=$PKG_CONFIG_PATH:/home/teamo/opencv4/build/install/lib/pkgconfig
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/teamo/opencv4/build/install/lib
```

这里的目录根据你的安装目录而确定，之后执行

```
source ~/.bashrc
sudo ldconfig
```

这时应该配置成功了，你可以使用

```
pkg-config --modversion opencv4
```

这时你应该可以看到正确的版本号 4.5.2，注意，对于 opencv3 系列你应该输入的是

```
pkg-config --modversion opencv
```

这里面更改的是系统默认的 `opencv` 优先版本，也就是当你下载其他依赖 `opencv` 的功能包时的首选依赖版本。

Linux下程序演示

这里与Windows下的一致，暂时空缺。

多版本切换

其实，经过上面的安装和使用步骤我们已经清楚了多版本切换的具体步骤

- 系统默认版本

这个其实就是环境变量配置，我们只需要修改环境变量使他指向正确的 `opencv` 版本即可。

- 工程使用版本

这里我们讨论的只针对 `CMake` 工程，其实需要做的很简单，只需要在编写 `CMakeLists.txt` 文件时修改 `OpenCV_DIR` 目录使其指向正确的 `OpenCV` 安装目录即可。

核心模块和第三方模块简介
