第 1 章

数字图像视觉概述

计算机视觉处理是当今信息科学中发展最快的热点研究方向，涉及光学、电子和计算机科学等多个学科。计算机科学中的数字图像处理是其重要基础。本章将阐述数字图像处理的重要基础以及计算机视觉的基本概念。这些理论概念虽然重要，但是如果全面展开来讲，不但内容繁多（至少一本书），而且读者会在有限的时间内由于对抽象理论感到枯燥而逐渐失去学习兴趣。因此，本书将这些理论知识浓缩为一章，为后面章节做一个理论铺垫。在后续学习过程中，一旦看到理论术语，只需翻阅第 1 章即可，以此方便读者阅读本书。如果是从来没有接触过 OpenCV 开发的读者，更要学习一下本章内容，从而对图像处理有感性认识。理论的东西，不能一开始就贪多，贪多肯定会感到枯燥，从而放弃；但也不能完全没有，否则就是沙滩筑高楼。

1.1 图像的基本概念

1.1.1 图像和图形

图像是对客观世界的反映。“图”是指物体透射光或反射光的分布，“像”是人的视觉对“图”的认识。“图像”是两者的结合。图像既是一种光的分布，又包含人的视觉心理因素。图像的最初取得是通过对物体和背景的“摄取”。这里的“摄取”意味着一种“记录”的过程，如照相、摄影、扫描等，这是图像与图形的主要区别。

图形是用数学规则产生的或具有一定规则的图案。图形往往用一组符号或线条来表示性质，例如房屋设计图，我们用线条来表现房屋的结构。图像和图形在一定条件下可向另一方转化。

1.1.2 什么是数字图像

数字图像又称数码图像或数位图像，是一种以二维数组（矩阵）形式表示的图像。数字图像

是由模拟图像数字化得到、以像素为基本元素、可以用数字计算机或数字电路存储和处理的图像。

数字图像可以从许多不同的输入设备和技术生成，例如数码相机、扫描仪、坐标测量机等，也可以从任意的非图像数据合成得到，例如数学函数或者三维几何模型。三维几何模型是计算机图形学的一个主要分支，数字图像处理领域主要研究它们的变换算法。

1.1.3 数字图像的特点

数字图像有如下特点：

1）信息量大

以数目较少的电视图像为例，一幅电视图像一般由 512×512 像素、8bit 组成，其总数据量为 512×512×8bit=2097152bit=262144B=256KB。这么大的数据量必须由计算机处理，且计算机内存容量要大。

2）占用频率的带宽大

一般语言信息（如电话、传真、电传、电报等）的带宽仅 4kHz 左右，而图像信息所占用频率的带宽要大 3 个数量级。例如普通电视的标准带宽是 6.5MHz，等于语言带宽的 14 倍。因此，在摄影、传输、存储、处理、显示等各环节的实现上技术难度大，这使得对频带的压缩技术的要求变得迫切。

3）相关性大

每幅图像中相邻像素之间不独立，具有很大的相关性，有时大片大片的像素间具有相同或接近的灰度。例如电视画面，前后两幅图像的相关系数往往在 0.95 以上。因此，压缩图像信息的潜力很大。

4）非客观性

图像信息最终的接收器是人的视觉系统。图像信息和视觉系统都十分复杂，与环境条件、视觉特性、情绪、精神状态、知识水平都有关，例如航空照片判读。因此，要求图像系统与视觉系统有良好的“匹配”，必须研究图像的统计规律和视觉特征。

1.1.4 图像单位（像素）

任意一幅数字图像粗看起来似乎是连续的，实际上是不连续的，它由许多密集的色点组成。就像任意物质一样，肉眼看上去是连续的，但实质上都是由分子组成的。这些色点是构成一幅图像的基本单元，被称为像素（或像素点、像元，Pixel）。例如，一幅图片由 30 万个色点组成，那这幅图片的像素就是 30 万。像素是数字图像的基本元素。显然，像素越多，画面就越清晰。

像素是感光元件记录光信号的基本单位，通常来说 1 个像素对应 1 个光电二极管。我们常说相机是多少像素，这个像素就是说这款相机的感光元件有多少个，有 100 万个感光元件的相机就是 100 万像素的相机，有 4000 万个感光元件的相机就是 4000 万像素的相机，以此类推。一台 100 万像素的相机拍摄的照片洗成 5 寸比洗成 6 寸清晰一点。像素高并不意味着画质一定好。一款相机的画质是由感光元件尺寸、像素数量、图像处理算法、镜头等共同决定的。同时，画质本身就是分辨率、动态范围、色彩深度、色彩准确性、噪点等诸多指标的集合。像素高低只对分辨率这

个单一指标影响较大。

像素是在模拟图像数字化时对连续空间进行离散化得到的。每个像素具有整数行（高）和列（宽）位置坐标，同时每个像素都具有整数灰度值或颜色值。

示例图像及其信息如图 1-1 和图 1-2 所示。

图 1-1

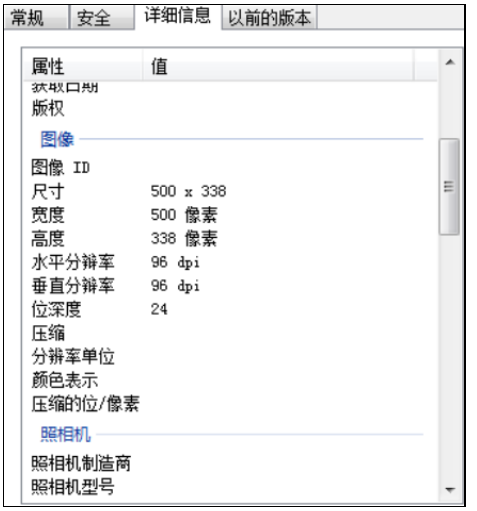

图 1-2

可以看到图 1-1 的尺寸是 500×338，表示图片由一个 500×338 的像素点矩阵构成，这幅图片的宽度是 500 像素，高度是 338 像素，共有 500×338 像素 = 149000 像素。

又如，屏幕分辨率是 1024×768，也就是说设备屏幕的水平方向上有 1024 个像素点，垂直方向上有 768 个像素点。像素的大小是没有固定长度的，不同设备上每个单位像素色块的大小是不一样的。例如，尺寸面积大小相同的两块屏幕，分辨率可以是不一样的，分辨率高的屏幕上的像素点（色块）就多，单个色块面积也更小，所以屏幕内显示的画面就更细致。而分辨率低的屏幕上的像素点（色块）较少，单个像素面积更大，显示的画面就没那么细致。

1.1.5　图像分辨率

图像分辨率是指每英寸图像内的像素点数。图像分辨率是有单位的，叫作像素每英寸。分辨率越高，像素点密度越高，图像就越逼真（做大幅喷绘时，要求图片分辨率要高，就是为了保证每英寸的画面上拥有更多的像素点）。

1.1.6　屏幕分辨率

屏幕分辨率是指屏幕在纵横方向上的像素点数，单位为像素（px）。每个屏幕都有其特定的分辨率。屏幕分辨率越高，所呈现的色彩数量和清晰度也就越高。分辨率决定了计算机屏幕上能够显示的信息量，通常用水平和垂直像素的数量来衡量。

在相同大小的屏幕上，较低的分辨率（例如 640×480）意味着屏幕上显示的像素较少，单个像素的尺寸较大。而较高的分辨率（例如 1600×1200）则意味着屏幕上显示的像素较多，单个像

素的尺寸相对较小。这种差异影响了图像的清晰度和细节表现。

1.1.7 图像的灰度

灰度是指将白色与黑色之间按对数关系划分为若干等级的一个概念。通常，灰度分为 256 个等级，其中 0 表示黑色。灰度实际上是没有色彩的表现形式，RGB 色彩分量在这种情况下全部相等。

例如，在一个二值灰度图像中，像素值只能为 0 或 1，因此其灰度级为 2。而对于一个 256 级灰度的图像，当 RGB 三个分量相等时，例如 RGB(100, 100, 100)，则表示灰度值为 100；同理，RGB(50, 50, 50)则表示灰度值为 50。

一幅图像中不同位置的亮度是不一样的，可用 $f(x,y)$来表示点(x,y)上的亮度。由于光是一种能量形式，故亮度是非负有限的（$0 \leqslant f(x,y) < \infty$）。在图像处理中常用的灰度，其含义是：单色图像中，坐标点(x,y)的亮度称为该点的灰度。设灰度为 L，则 $L_{min} \leqslant L \leqslant L_{max}$，区间$[L_{min}, L_{max}]$称为灰度范围。

在室内处理图像时，一般 $L_{min} \approx 0.005$Lux，$L_{max} \approx 100$Lux，其中 Lux 是勒克斯（照明单位）。实际使用时，把这个区间规格化为$[0, L_{max}]$，其中 $L_{min}=0$ 为黑色，L_{max} 为白色，而所有在白色和黑色之间的值代表连续变化的灰度。图像灰度化处理可以作为图像处理的预处理步骤，为之后的图像分割、图像识别和图像分析等上层操作做准备。

1.1.8 灰度级

灰度级表明图像中不同灰度值的最大数量。灰度级越大，图像的亮度范围就越大。灰度级有时候会与灰度混淆。首先应该明确，灰度（值）是针对单个像素而言的，表示灰度图像单个像素点的亮度值，值越大，像素点就越亮，反之越暗。灰度级表示灰度图像的亮度层次，比如第 1 级、第 2 级……第 255 级等，如图 1-3 所示。

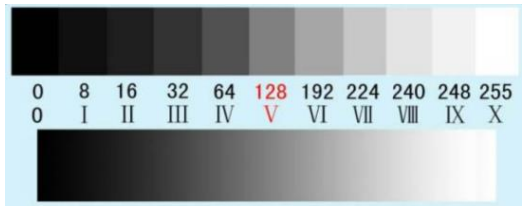

图 1-3

在图 1-3 中，第 0 级的灰度是 0，第 1 级的灰度是 8，第 2 级的灰度是 16，等等。每个等级对应着一个灰度值。级数越多，图像的亮度范围越大，层次就越丰富。有时，把最大级数称为一幅图像的灰度级数。

1.1.9 图像深度

图像深度是指存储每个像素所用的位数，也用于度量图像的色彩分辨率。图像深度确定彩色

图像每个像素可能有的颜色数，或者确定灰度图像每个像素可能有的灰度级数。比如一幅单色图像，若每个像素有 8 位，则最大灰度数目为 2 的 8 次方，即 256。一幅彩色图像 RGB 三个分量的像素位数分别为 4、4、2，则最大颜色数目为 2 的 4+4+2 次方，即 1024，也就是说图像深度为 10 位，每个像素可以是 1024 种颜色中的一种。

例如，一幅画的尺寸为 1024×768 像素，深度为 16，则它的数据量为 1.5MB。计算如下：

1024×768×16bit = (1024×768×16)/8B = [(1024×768×16)/8]/1024KB = {[(1024×768×16)/8]/1024}/1024MB=1.5MB

1.1.10　二值图像

二值图像（Binary Image）是指图像上的每一个像素只有两种可能的取值或灰度等级状态，常用黑白、B&W、单色图像表示二值图像。二值图像一般用来描述字符图像，其优点是占用空间少；缺点是当表示人物、风景的图像时，只能展示其边缘信息，图像内部的纹理特征表现不明显。这时候要使用纹理特征更为丰富的灰度图像。

二值图像中，每个像素只用 1bit 表示，只有黑白两种颜色。二值图像按名字来理解只有两个值，即 0 和 1，0 代表黑，1 代表白，或者说 0 表示背景，而 1 表示前景。其保存也相对简单，每个像素只需要 1bit 就可以完整存储信息。如果把每个像素看成随机变量，一共有 N 个像素，那么二值图像有 2 的 N 次方种变化，8 位灰度图有 255 的 N 次方种变化，8 位三通道 RGB 图像有 16777216^N 种变化，注意：16777216=256（红）×256（绿）×256（蓝）。也就是说，同样尺寸的图像，二值图像保存的信息更少。

1.1.11　灰度图

灰度图又称灰阶图，是用灰度表示的图像。除了常见的卫星图像、航空照片外，许多地球物理观测数据也以灰度图表示。我们平时看到的灰度图由 0~255 个像素组成。

灰度图是二值图像的进化版本，是彩色图像的退化版，也就是灰度图保存的信息比彩色图像少，但比二值图像多。灰度图只包含一个通道的信息，而彩色图通常包含三个通道的信息。单一通道可以理解为单一波长的电磁波，所以红外遥感、X 断层成像等单一通道电磁波产生的图像都为灰度图。另外，灰度图在实际应用中具有易于采集和传输等特性，使得基于灰度图开发的算法非常丰富。

灰度图的每个像素只有一个采样颜色，这类图像通常显示为从最暗的黑色到最亮的白色的灰度。灰度图与黑白图像不同，在计算机图像领域，黑白图像只有黑色与白色两种颜色，但是灰度图在黑色与白色之间还有许多级的颜色深度。灰度图经常是在单个电磁波频谱（如可见光）内测量每个像素的亮度得到的，用于显示的灰度图通常用每个采样像素 8 位的非线性尺度来保存，这样可以有 256 级灰度（如果用 16 位，就有 65536 级）。

1.1.12　彩色图像

彩色图像也就是 RGB 图像，每个像素通常由红（R）、绿（G）、蓝（B）三个分量来表示，

分量值为 0~255。我们日常获得的通常是彩色图像，很多时候我们需要将彩色图像转换成灰度图像，也就是将 3 个通道（RGB）转换成 1 个通道。

1.1.13 通道

通道把图像分解成一个或多个颜色成分，通常可以分为单通道、三通道和四通道。

- 单通道：一个像素点只需一个数值表示。单通道只能表示灰度，0 为黑色。单通道图像就是图像中每个像素点只需用一个数值表示。
- 三通道：把图像分为红、绿、蓝三个通道。三通道可以表示彩色，全 0 表示黑色。
- 四通道：在 RGB 的基础上增加了 Alpha 通道，Alpha 通道表示透明度，Alpha=0 表示全透明。

1.1.14 图像存储

在计算机中，用 $M\times N$ 的矩阵表示一幅尺寸大小为 $M\times N$ 的数字图像，矩阵元素的值就是该图像对应位置上的像素值。三通道图像数据在内存中的存储是连续的，每个通道的元素按照矩阵行列顺序进行排列。通常计算机按照 RGB 方式存储三通道图像格式，而图像采集设备输出图像格式一般是 BGR 方式。

1.2 图像噪声

1.2.1 图像噪声的定义

图像噪声可以理解为妨碍人的视觉器官或系统传感器对所接收的图像源信息进行理解或分析的各种因素。一般图像噪声是不可预测的随机信号，只能用概率统计的方法去认识。噪声作用于图像处理的输入、采集、处理以及输出的全过程，特别是图像在输入、采集的过程中引入的噪声，会影响图像处理的全过程，以至于影响输出结果。噪声对图像的影响无法避免，因此一个良好的图像处理系统，无论是模拟处理还是计算机处理，无一不将最前一级的噪声减小到最低作为主攻目标。因此，滤除图像中的噪声就成为图像处理中极为重要的步骤，对图像处理有着重要的意义。

数字图像的噪声主要来源于图像获取的数字化过程。图像传感器的工作状态受各种因素的影响，如环境条件、传感器元件质量等。在图像传输的过程中，所用的传输信道受到干扰，也会产生噪声污染。例如，通过无线网络传输的图像可能会因为光或其他大气因素的干扰而受到噪声污染。图像噪声的种类有多种，包括高斯噪声、瑞利噪声、伽马噪声、指数噪声、均匀噪声以及脉冲噪声（又称为椒盐噪声或双极性噪声）等。其中，脉冲噪声在图像噪声中最为常见。在图像生成和传输的过程中，经常会产生脉冲噪声，主要表现在成像的短暂停留中。脉冲噪声对图像质量有较大的影响，需要采用图像滤波方法给予滤除。

1.2.2　图像噪声的来源

外部噪声是指系统外部干扰以电磁波的方式或经电源串进系统内部而引起的噪声，例如电气设备、天体放电现象等引起的噪声。

内部噪声一般可分为以下 4 种：

（1）由光和电的基本性质引起的噪声。例如，电流是由电子或空穴粒子的集合定向运动形成的，而这些粒子运动的随机性会形成散粒噪声；导体中自由电子的无规则热运动会形成热噪声；根据光的粒子性，图像是由光量子传输的，而光量子密度随时间和空间变化会形成光量子噪声等。

（2）电器的机械运动产生的噪声。例如，各种接头因抖动而引起电流变化所产生的噪声，如磁头、磁带等抖动或一起的抖动等。

（3）器材材料本身引起的噪声。例如，正片和负片的表面颗粒性以及磁带、磁盘表面缺陷所产生的噪声。随着材料科学的发展，这些噪声在不断减少，但目前来讲，还是不可避免。

（4）系统内部设备电路所引起的噪声。例如，电源引入的交流噪声，偏转系统和箝位电路所引起的噪声等。

1.2.3　图像噪声的滤除

通过平滑图像，可以有效减少和消除图像中的噪声，从而改善图像质量，这对于提取对象特征以进行分析非常有利。经典的平滑技术通常使用局部算子对噪声图像进行处理。在对某个像素进行平滑处理时，仅对其局部小邻域内的其他像素进行操作。这种方法的优点在于计算效率高，并且能够实现多个像素的并行处理。近年来，出现了一些新的图像平滑处理技术，这些技术结合了人眼的视觉特性，运用了模糊数学理论、小波分析、数学形态学和粗糙集理论等新方法，取得了良好的效果。

灰度图像常用的滤波方法主要分为线性和非线性两大类。线性滤波方法一般通过取模板做离散卷积来实现。这种方法在平滑脉冲噪声点的同时，也会导致图像模糊，从而损失图像细节信息。非线性滤波方法中应用最多的是中值滤波，中值滤波可以有效地滤除脉冲噪声，具有相对好的边缘保持特性，并易于实现，因此被公认为一种有效的方法。然而，中值滤波同时也会改变未受噪声污染的像素的灰度值，使图像变得模糊。随着滤波窗口长度的增加和噪声污染的加重，中值滤波效果明显下降。

针对中值滤波方法的缺陷，目前科学家已经提出了一些改进方法，但这些方法都是无条件地对所有的输入样本进行滤波处理。然而，对于一幅噪声图像来说，只有一部分像素受到了噪声的干扰，其余的像素仍保持原值。无条件地对每个像素进行滤波处理必然会损失图像的某些原始信息。因此，人们提出在滤波处理中加入判断的过程，即首先检测图像的每个像素是否为噪声，然后根据噪声检测结果进行切换，输出结果在原像素灰度和中值滤波或其他的滤波器计算结果之间切换。由于是有选择地进行滤波处理，避免了不必要的滤波操作和图像的模糊，因此滤波效果得到了进一步的提高。但这些方法在判断和滤除脉冲噪声的过程中还存在一定的缺陷，比如对于较亮或较暗的图像会产生较多的噪声误判和漏判，甚至无法进行噪声检测，同时算法的计算量也明显增加，从而影响了滤波效果和速度。

既然图像有时不可避免地会产生噪声，那就需要对图像进行处理。

1.3 图像处理

信息是自然界物质运动的一个重要方面，人们认识和改造世界需要各种信息图像。这些信息是人类获取外界知识的主要来源，其中约 80%的信息通过人眼获得。在现代科学研究、生产活动等各个领域，越来越多地使用图像信息来认识和判断事物，以解决实际问题。获取图像信息固然重要，但我们的主要目的是对这些信息进行处理，以便从大量复杂的图像数据中提取出感兴趣的信息。图像处理是指对图像信息进行加工和处理，以满足视觉或应用上的需求。因此，从某种意义上来说，对图像信息的处理比图像本身更为重要。

21 世纪是一个充满信息的时代，图像作为人类感知世界的视觉基础，是获取、表达和传递信息的重要手段。计算机时代所说的图像处理通常指的是数字图像处理，即利用计算机对图像进行处理。这一技术的发展历史并不悠久，数字图像处理技术的起源可以追溯到 20 世纪 20 年代，当时通过海底电缆对从英国伦敦传输到美国纽约的一幅照片采用了数字压缩技术。

数字图像处理技术首先帮助人们以更客观、准确的方式认识世界。人的视觉系统能够帮助人类获取超过 3/4 的信息，而图像和图形则是所有视觉信息的载体。尽管人眼的辨识能力很高，可以识别上千种颜色，但在很多情况下，图像对于人眼而言是模糊的，甚至是不可见的。通过图像增强技术，可以使这些模糊或不可见的图像变得清晰明亮。

在计算机视觉这一领域诞生的初期，一种普遍的研究范式是将图像看作二维的数字信号，然后借用数字信号处理中的方法进行处理，这就是图像处理。

图像处理又称为影像处理，是用计算机对图像进行分析，以获得所需结果的技术，一般包括图像压缩、增强和复原、匹配和识别 3 个部分。目前所说的图像处理一般指数字图像处理（Digital Image Processing）。计算机视觉与图像处理的区别主要在于：计算机视觉的侧重点在于使用计算机来模拟人的视觉，对客观事物进行“感知”；而图像处理的侧重点则在于“处理”，提取所需要的有效信息，两者相辅相成。

1.3.1 图像处理的分类

图像处理通常可以分为 3 类：光学模拟处理、电学模拟处理和计算机数字处理。

1）光学模拟处理

光学模拟处理也称光信息处理。它建立在傅里叶光学的基础上，进行光学滤波、相关运算、频谱分析等，可以实现图像像质的改善、图像识别、图像的几何畸变和光度的校正、光信息的编码和存储、图像的伪彩色化、三维图像显示、对非光学信号进行光学处理等。

2）电学模拟处理

电学模拟处理是指把光强度信号转换成电信号，然后用电子学的方法对信号进行加、减、乘、除，进行浓度分割、反差放大、彩色合成、光谱对比等。在电视视频信号处理中常应用它。近期发展较快的 CCD（电荷耦合器件）模拟处理方法有 3 种处理功能：

- 模拟延迟，改变时钟脉冲频率就能实现模拟延迟。
- 多路调制，把并列输入的信号转换成串行的时序信号，或者建立它们的反变换，可实现数

据信息的重新排列。

- 做成各种相应的滤波器，而滤波器就是一个信号处理装置。

CCD 在设备和成本方面有优点，滤波较计算机易实现。

3）计算机数字处理

图像的计算机数字处理是在以计算机为中心的、包括各种输入/输出及显示设备在内的数字图像处理系统上进行的。它将连续的模拟图像变换成离散的数字图像后，使用由特定的物理模型和数学模型编制而成的程序进行控制，并实现各种要求的处理。

1.3.2　数字图像处理

数字图像处理技术，通俗地讲是指应用计算机以及数字设备对图像进行加工处理的技术。通常包括如下几个过程。

1）图像信息的获取

为了在计算机上进行图像处理，必须把作为处理对象的模拟图像转换成数字图像信息。图像信息的获取一般包括图像的摄取、转换及数字化等几个步骤。这部分主要由处理系统硬件实现。

一般情况下，由于图像处理的设备比较大，不易在室外使用，因此通常输入图像分两步进行：首先在室外通过摄影机、照相机、数码相机等设备将图像记录下来，然后在室内利用输入设备进行输入。一般用磁带记录的是视频信号，通过 AN 口、1394 口输入视频采集卡；用胶片记录的是照片，可通过扫描仪扫描输入；电子照片可直接通过串口、并口或 USB 口输入。

2）图像信息的存储与交换

由于数字图像中的信息量庞大，在处理过程中必须对数据进行存储和交换。为了有效解决大数据量与交换和传输时间之间的矛盾，通常除了采用大容量内存存储器进行并行传输和直接存储访问外，还需要利用外部磁盘、光盘和磁带等存储方式，以提高处理效率。这部分主要功能通常由硬件完成。

3）具体的图像处理

数字图像处理是指将空间上离散的、在幅度上量化分层的数字图像经过特定的数学模式进行加工处理，以获得人眼视觉或某种接收系统所需的图像。自 20 世纪 80 年代以来，计算机技术和超大规模集成电路技术的迅猛发展，极大地推动了通信技术（包括语言数据、图像）的飞速发展。因为图像通信具有形象直观、可靠、高效率等一系列优点，尤其是数字图像通信比模拟图像通信更具抗干扰性，便于进行压缩编码处理且易于加密，因此数字处理技术在图像通信工程中获得了广泛应用。

4）图像的输出和显示

数字图像处理的最终目的是提供便于人眼或接收系统解释和识别的图像，因此图像的输出和显示十分重要。一般图像输出的方式可分为硬拷贝（如照相、打印、扫描等）和软拷贝（如 CRT 监视器及各种新型的平板监视器等）。

1.3.3　数字图像处理常用的方法

数字图像处理常用的方法有图像变换、图像增强、图像分割、图像描述、图像分类（识别）

和图像重建等。

1. 图像变换

由于图像阵列很大，直接在空间域中进行处理涉及的计算量很大，因此往往采用各种图像变换的方法，如傅里叶变换、沃尔什变换、离散余弦变换等间接处理技术，将空间域的处理转换为变换域的处理。这样不仅可以减少计算量，而且能获得更有效的处理（如傅里叶变换可在频域中进行数字滤波处理）。目前新兴研究的小波变换在时域和频域中都具有良好的局部化特性，在图像处理中也有着广泛且有效的应用。

图像编码压缩技术可减少描述图像的数据量（比特数），以便节省图像传输和处理的时间，以及减少所占用的存储器容量。压缩可以在不失真的前提下获得，也可以在允许失真的条件下进行。编码是压缩技术中重要的方法，它在图像处理技术中是发展最早且比较成熟的技术。

2. 图像增强

对于一个数字图像处理系统来说，一般可以将处理流程分为 3 个阶段：首先是图像预处理阶段，其次是特征抽取阶段，最后是识别分析阶段。图像预处理阶段尤为重要，如果这个阶段处理不好，会直接导致后面的工作无法展开。图像增强是图像预处理阶段的重要步骤。

在采集图像时，由于光照的稳定性与均匀性等噪声的影响，灰尘对 CCD 摄影机镜头的影响，以及图像传输过程中由于硬件设备而获得的噪声，使得获取的图像不够理想，往往存在噪声、对比度不够、目标不清晰、有其他物体干扰等缺点。这就需要用图像增强技术来改善图像效果。

图像增强就是增强图像中用户感兴趣的信息，其主要目的有两个：一是改善图像的视觉效果，提高图像成分的清晰度；二是使图像变得更有利于计算机处理。

图像增强不是以图像保真原则为基点来处理图像的，而是根据图像质量变坏的一般情况提出一些改善方法。例如，在图像处理中，可以采用图像均衡的方法来缩小图像灰度差别，采用平滑滤波的方法去除图像存在的噪声，采用边缘增强的方法改善图像轮廓的不明显。

图像增强主要应用在图像特别暗时，或者因为曝光太亮而无法让目标突出时，这个时候就需要把目标的亮度提高一点，然后把不必要的障碍（俗称噪声）调暗，以利于目标清晰度最大化。

图像增强的方法通过一定手段对原图像附加一些信息或变换数据，有选择地突出图像中感兴趣的特征或者抑制（掩盖）图像中某些不需要的特征，使图像与视觉响应特性相匹配。

在图像增强过程中，不分析图像降质的原因，处理后的图像不一定逼近原始图像。

通过各种手段来获得清晰图像的方法就是图像增强。根据增强的信息不同，图像增强可以分为边缘增强、灰度增强、色彩饱和度增强等。其中，灰度增强又可以根据增强处理过程所在的空间不同，分为空间域增强和频率域增强两大类，分别简称空域法和频域法。

1）空域法

空域法主要是直接在空间域内对图像进行运算处理，分为点运算算法和邻域去噪算法（也称邻域增强算法）。

点运算通常包括灰度变换和直方图修正等，目的或使图像成像均匀，或扩大图像动态范围，扩展对比度。

邻域去噪算法分为图像平滑和锐化两种。平滑一般用于消除图像噪声，但是也容易引起图像边缘的模糊，常用算法有均值滤波、中值滤波。锐化的目的在于突出物体的边缘轮廓，便于目标识别，常用算法有梯度法、算子、高通滤波、掩码匹配法、统计差值法等。

2）频域法

频域法是利用图像变换方法将原来的图像空间中的图像以某种形式转换到其他空间中，然后利用该空间的特有性质进行图像处理，最后转换回原来的图像空间中，从而得到处理后的图像。

频域法增强技术的基础是卷积理论。其中，频域变换可以是傅里叶变换、小波变换、DCT 变换、Walsh 变换等。

我们可以用一幅图来表示图像增强所用的具体方法分类，如图 1-4 所示。

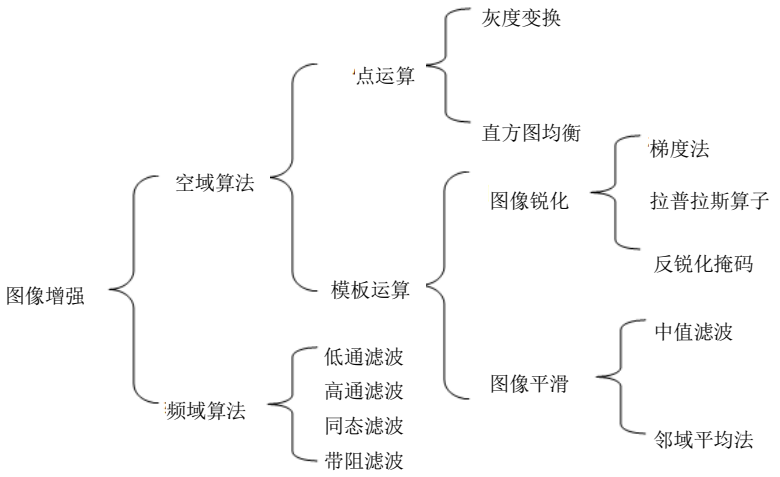

图 1-4

当然，作为初学者，我们不需要面面俱到、全部掌握，可以先选择掌握重点的几项。

3. 图像分割

图像分割是数字图像处理中的关键技术之一，它将图像中有意义的特征部分提取出来。有意义的特征有图像中的边缘、区域等，这是进一步进行图像识别、分析和理解的基础。虽然目前已研究出不少边缘提取、区域分割的方法，但还没有一种普遍适用于各种图像的有效方法。因此，对图像分割的研究还在不断深入之中，是目前图像处理研究的热点之一。

4. 图像描述

图像描述是图像识别和理解的必要前提。作为简单的二值图像，可采用其几何特性描述物体的特性。一般图像的描述方法采用二维形状描述，可分为有边界描述和区域描述两类。对于特殊的纹理图像，可采用二维形状描述。随着图像处理研究的深入发展，已经开始进行三维物体描述的研究，提出了体积描述、表面描述、广义圆柱体描述等方法。

5. 图像分类（识别）

图像分类（识别）属于模式识别的范畴，其主要内容是对经过某些预处理（增强、复原、压缩）的图像进行分割和特征提取，从而进行判决分类。图像分类常采用经典的模式识别方法，有统计模式分类和句法（结构）模式分类。近年来新发展起来的模糊模式识别和人工神经网络模式分类在图像识别中越来越受重视。

6. 图像重建

图像重建是指对一些三维物体，应用 X 射线、超声波等物理方法取得物体内部结构数据，再将这些数据进行运算处理而构成物体内部某些部位的图像。目前图像重建成功的例子是 CT 技术（计算机断层扫描成像技术）、彩色超声波等。这是图像处理的另一个发展方向。

1.3.4 图像处理的应用

图像处理的应用十分广泛，大大促进了现代社会的发展。例如，日常生活中的人脸支付就用到了图像处理，出入停车场时的车牌识别也用到了图像处理。下面用一张表格来简明扼要地说明图像处理的常见应用，如表 1-1 所示。

表 1-1 图像处理的常见应用

领 域	应用内容
物理化学	结晶分析、谱分析
生物医学	细胞分析、染色体分类、血球分类、X 光、CT
环境保护	水质及大气污染调查
地质	资源勘探、地图绘制
农林	植被分布调查、农作物估产
海洋	鱼群探察
水利	河流分布、水利及水害调查
气象	云图分析、灾害性检测等
通信	传真、电视、可视电话图像通信
工业	工业探伤、计算机视觉、自动控制、机器人
法律	公安指纹识别、人像鉴定
交通	铁路选定、交通指挥、汽车识别
军事	侦察、成像融合、成像制导
宇航	星际探险照片处理
文化	多媒体、动画特技

1.4 图像信号处理层次

图像信号的处理分为以下 3 个层次：

（1）图像处理：图像采集、存储，图像重建，图像变换、增强、恢复、校正，图像（视频）压缩编码。

（2）图像分析：边缘检测，图像分割，目标表达、描述，目标颜色、形状、纹理、空间和运动分析，目标检测、识别。

（3）图像理解：图像配准（Image registration）、融合，3-D 表示、建模、场景恢复，图像感知、解释、推理，基于内容的图像和视频检索。图像配准就是将不同时间、不同传感器（成像设备）或不同条件（天气、温度、摄像位置和角度等）下获取的两幅或多幅图像进行匹配、叠加的

过程，它已经被广泛地应用于遥感数据分析、计算机视觉、图像处理等领域。

1.5　计算机视觉

1.5.1　计算机视觉的概念

计算机视觉是人工智能的一个重要分支，旨在让计算机能够“看见”并理解图像和视频内容。随着计算能力的提升和深度学习技术的发展，计算机视觉在近年来取得了飞速的进展，并在多个领域实现了广泛应用。

计算机视觉就是使用光学非接触式感应设备自动接收并解释真实场景的图像，以获得信息并控制机器或流程。计算机视觉技术是一门涉及人工智能、神经生物学、心理物理学、计算机科学、图像处理、模式识别等诸多领域的交叉学科。它主要通过计算机来模拟人的视觉功能，从客观事物的图像中提取信息，进行处理并加以理解，最终用于实际检测、测量和控制。计算机视觉技术最大的特点是速度快、信息量大和功能多。

人类在生产实践的过程中，针对自身能力的局限性，发明和创造了许多智能机器来辅助或代替人类完成任务。智能机器能模拟人类的功能，感知外部世界并有效地解决人所不能解决的问题。人类感知外部世界主要通过视觉、触觉、听觉和嗅觉等感觉器官，其中约 80%的信息是由视觉获取的。因此，对智能机器来说，赋予机器以人类视觉功能是极其重要的。

在现代工业自动化生产中，涉及各种各样的检查、测量和零件识别应用，例如汽车零配件尺寸检查和自动装配的完整性检查、电子装配线的元件自动定位、饮料瓶盖的印刷质量检查、产品包装上的条码和字符识别等。这类应用的共同特点是连续大批量生产，对外观质量的要求非常高。通常这种带有高度重复性和智能性的工作只能靠人工检测来完成，我们经常在一些工厂的现代化流水线后面看到数以百计甚至逾千的检测工人来执行这道工序，但这在给工厂增加巨大的人工成本和管理成本的同时，仍然不能保证 100%的检验合格率（“零缺陷”），而当今企业之间的竞争已经不允许哪怕是 0.1%的缺陷存在。有些时候，如微小尺寸的精确快速测量、形状匹配、颜色辨识等，用人眼根本无法连续稳定地进行，其他物理量传感器也难有用武之地。这时，人们开始考虑把计算机的快速性和可靠性、结果的可重复性与人类视觉的高度智能化和抽象能力相结合，由此逐渐形成了一门新学科——计算机视觉。

计算机视觉是研究如何使用计算机模拟生物宏观视觉功能的科学与技术。通俗地说，它是用计算机代替人眼进行测量和判断。首先，通过 CCD 照相机将被摄取的目标转换为图像信号，并将其传送至专用的图像处理系统。这些信号根据像素分布、亮度和颜色等信息转变为数字化信号。接着，图像系统对这些信号进行各种运算，以提取目标的特征，如面积、长度、数量和位置等。最后，根据预设的容许度和其他条件输出结果，如尺寸、角度、偏移量、个数、合格/不合格、有/无等。计算机视觉的特点包括自动化、客观性、非接触性和高精度。与一般的图像处理系统相比，计算机视觉更加强调精度和速度，并且在工业现场环境下具有更高的可靠性。

计算机视觉是一个相当新且发展十分迅速的研究领域。人们从 20 世纪 50 年代开始研究二维图像的统计模式识别；60 年代 Roberts 开始进行三维计算机视觉的研究；70 年代中期，MIT 人工智能实验室正式开设“计算机视觉”课程；80 年代，开始了全球性的研究热潮，计算机视觉获得

了蓬勃发展，新概念、新理论不断涌现。现在，计算机视觉仍然是一个非常活跃的研究领域，与之相关的学科涉及图像处理、计算机图形学、模式识别、人工智能、人工神经元网络等。

1.5.2 计算机视觉系统的构成和分类

典型的视觉系统一般由以下 3 部分构成：

（1）图像获取：光源、镜头、相机、采集卡、机械平台。
（2）图像处理与分析：工控主机、图像处理分析软件、图形交互界面。
（3）判决执行：电传单元、机械单元。

视觉系统输出的并非图像视频信号，而是经过运算处理之后的检测结果，如尺寸数据。上位机如 PC 和 PLC 实时获得检测结果后，指挥运动系统或 I/O 系统执行相应的控制动作，如定位和分选。

机器视觉系统通常可以分为三大类：基于智能相机、基于嵌入式和基于 PC。

1.5.3 机器视觉的优势

虽然人类视觉擅长对复杂、非结构化的场景进行定性解释，但机器视觉凭借速度、精度和可重复性等优势，擅长对结构化场景进行定量测量。举例来说，在生产线上，机器视觉系统每分钟能够对数百个甚至数千个元件进行检测。配备适当分辨率的相机和光学元件后，机器视觉系统能够轻松检验小到人眼无法看到的物品的细节特征。

另外，由于消除了检验系统与被检验元件之间的直接接触，机器视觉还能够防止元件损坏，也避免了机械部件磨损的维护时间和成本投入。通过减少制造过程中的人工参与，机器视觉还带来了额外的安全性和操作优势。此外，机器视觉还能够防止洁净室受到人为污染，也能让工人免受危险环境的威胁。

1.5.4 机器视觉系统的应用

目前，国际上视觉系统的应用方兴未艾，而在中国，工业视觉系统尚处于概念导入期，各行业的领先企业在解决了生产自动化的问题以后，已开始将目光转向测量自动化方面。机器视觉极适用于大批量生产过程中的测量、检查和辨识，如零件装配完整性、装配尺寸精度、零件加工精度、位置/角度测量、零件识别、特性/字符识别等。其最大的应用行业为汽车、制药、电子与电气、制造、包装/食品/饮料、医学。例如对汽车仪表盘加工精度的检查，高速贴片机上对电子元件的快速定位，对管脚数目的检查，对 IC（集成电路芯片）表面印的字符的辨识，胶囊生产中对胶囊壁厚和外观缺陷的检查，轴承生产中对滚珠数量和破损情况的检查，食品包装上面对生产日期的辨识，对标签贴放位置的检查，等等。

1.5.5　计算机视觉与相关学科的关系

计算机视觉与相关学科的关系如下：

- 计算机图形学（Computer Graphics）：通过几何基元（如线、圆和自由曲面等）来生成图像，属于图像综合，它在可视化（Visualization）和虚拟现实（Virtual Reality）中起着很重要的作用。计算机视觉正好用于解决相反的问题，即从图像中估计几何基元和其他特征，属于图像分析。
- 模式识别（Pattern Recognition）：研究分类问题，确定符号、图画、物体等输入对象的类别，强调一类事物区别于其他事物所具有的共同特征，一般不关心三维世界的恢复问题。
- 人工智能（Artificial Intelligence）：涉及智能系统的设计和智能计算的研究，在经过图像处理和图像特征提取过程后，要用人工智能方法对场景特征进行表示，并分析和理解场景。
- 媒体计算（Multimedia Computing）：文字、图形、图像、动画、视频、音频等各类感觉媒体的共性的基础计算理论、计算方法，以及媒体系统实现技术，以实现下一代计算机能听、能看、会说、会学习为目标。
- 认知科学与神经科学（Cognitive science and Neuroscience）：将人类视觉作为主要的研究对象。计算机视觉中已有的许多方法与人类视觉极为相似，许多计算机视觉研究者对研究人类视觉计算模型比研究计算机视觉系统更感兴趣，希望计算机视觉更加自然化，更加接近生物视觉。

1.6　OpenCV 概述

OpenCV 是一个基于 BSD 许可（开源）发行的跨平台计算机视觉和机器学习软件库，可以运行在 Linux、Windows、Android 和 Mac OS 操作系统上。

OpenCV 用 C++语言编写，它具有 C++、Python、Java 和 MATLAB 接口，并支持 Windows、Linux、Android 和 Mac OS。OpenCV 主要倾向于实时视觉应用，并在可用时使用 MMX 和 SSE 指令，如今也提供对于 C#、Ch、Ruby、GO 的支持。

OpenCV 提供的视觉处理算法非常丰富，并且其部分以 C 语言编写，加上开源特性，如果处理得当，不需要添加新的外部支持也可以完整地编译链接生成执行程序，所以很多人用它来做算法的移植。OpenCV 的代码经过适当改写，可以正常地运行在 DSP 系统和 ARM 嵌入式系统中。

OpenCV 是一个开放源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，也就是用摄影机和计算机代替人眼对目标进行识别、跟踪以及测量等，并进一步进行图像处理。

OpenCV 由于其开源特性以及强大的社区支持，发展极其迅速。OpenCV 1.0 正式版本于 2006 年发布，可以运行在 Mac OS 以及 Linux 平台上，但是主要提供 C 的接口；2009 年发布了 OpenCV 2.0 版本，其代码已显著优化，带来了全新的 C++函数的接口，将 OpenCV 的能级无限放大，使开发者使用起来更加方便。同时，增加了新的平台支持，包括 iOS 和 Android，通过 CUDA 和 OpenGL 实现了 GPU 加速；在编程语言方面，为 Python 和 Java 用户提供了接口。2014 年 8 月，OpenCV 3.0

Alpha 发布，其最重大的革新之处在于 OpenCV 3.0 改变了项目架构的方式。之前的 OpenCV 是一个相对整体的项目，各个模块都以整体的形式构建，然后组合在一起，而 OpenCV 3 抛弃了整体架构，使用内核+插件的架构形式，使 OpenCV 更加轻量化。当前，OpenCV 随着工业 4.0 与机器人无人机的发展，已经在应用领域得到了广泛的应用，越来越多从事机器视觉与图像处理的开发者选择 OpenCV 作为开发工具实现应用开发。

计算机视觉市场巨大且持续增长，但这方面没有标准 API，如今的计算机视觉软件大概有以下 3 种：

（1）研究派代码（慢，不稳定，独立，与其他库不兼容）。

（2）耗费很高的商业化工具（比如 Halcon、MATLAB+Simulink）。

（3）依赖硬件的一些特别的解决方案（比如视频监控、制造控制系统、医疗设备）。

这是如今的现状，而标准的 API 将简化计算机视觉程序和解决方案的开发，OpenCV 致力于成为这样的标准 API。

OpenCV 致力于真实世界的实时应用，通过优化 C 代码的编写对其执行速度带来了可观的提升，并且可以通过购买 Intel 的 IPP（Integrated Performance Primitive，集成性能原语）高性能多媒体函数库得到更快的处理速度。

OpenCV 的应用领域非常广泛，比如人机互动、物体识别、图像分割、人脸识别、动作识别、运动跟踪、机器人、运动分析、机器视觉、结构分析、汽车安全驾驶、军工、卫星导航等。可以说学好了 OpenCV，就业和职业发展前景广阔！

编写本书时，OpenCV 的新版本为 4.10，本书采用该版本，它发布于 2024 年 6 月，如图 1-5 所示。

图 1-5

OpenCV 最显著的优化是从 4.10 开始对 JPEG 图像的读取和解码有了 77%的速度提升，超过了 scikit-image、imageio、pillow。其他改进点如下：

（1）dnn 模块的改进，包括：

- 改善内存消耗。
- 增加了将模型转储为与 Netron 工具兼容的 pbtxt 格式的功能。
- 支持多个新的 TFlite、ONNX 和 OpenVINO 层。
- 改进了现代 Yolo 探测器支持。
- 添加了 CuDNN 9+和 OpenVINO 2024 支持。

（2）core 模块的改进，包括：

- 为 cv::Mat 添加了 CV_FP16 数据类型。
- 扩展了 HAL API，用于 minMaxIdx、LUT、meanStdDev 和其他函数。

（3）imgproc 模块的改进，包括：

- 为 cv::remap 添加了相对位移场选项。
- 重构了 findContours 和 EMD。
- 扩展了 HAL API，用于 projectPoints、equalizeHist、Otsu 阈值和其他功能。
- 添加了针对现代 ARMv8 和 ARMv9 平台优化的新底层 HAL 库（KleidiCV）。

（4）支持 CUDA 12.4+。

（5）添加了 zlib-ng 作为经典 zlib 的替代品。

（6）对 Wayland、Apple VisionOS 和 Windows ARM64 的实验性支持。

（7）OpenCV Model Zoo 提供跨平台的预训练深度学习模型。其新增功能包括：

- 支持更多的模型结构，例如新的卷积架构或者神经网络架构。
- 提升模型的性能，可能通过模型优化或者使用更高效的实现方式。
- 提供更多的预处理和后处理的选项，以便用户可以更灵活地使用这些模型。
- 增加对新硬件或者新框架的支持，例如新版的 TensorRT 或是 ONNX Runtime。

1.7　Qt 简介

Qt 是 1991 年由 Haavard Nord 和 Eirik Chambe-Eng 开发的跨平台 C++图形用户界面应用程序开发框架。发展至今，它既可以开发 GUI 程序，也可以开发非 GUI 程序，比如控制台工具和服务器。Qt 同 Linux 上的 Motif、Openwin、GTK 等图形界面库和 Windows 平台上的 MFC、OWL、VCL、ATL 是同类型的。与其他用户开发界面的软件相比，Qt 更容易使用和学习。

Qt 是一个跨平台的 C++应用程序框架，支持 Windows、Linux、Mac OS X、Android、iOS、Windows Phone、嵌入式系统等。也就是说，Qt 可以同时支持桌面应用程序开发、嵌入式开发和移动开发，覆盖了现有的所有主流平台。开发者只需要编写一次代码，而后在发布到不同平台之前重新编译即可。

Qt 不仅仅是一个 GUI 库，它除了可以创建漂亮的界面之外，还有很多其他组件。例如，开发者不再需要研究 STL（Standard Template Library，标准模板库），不再需要 C++的头文件，也不再需要去找解析 XML、连接数据库和 Socket 的各种第三方库，因为这些组件已经内置在 Qt 中了。

Qt 是应用程序开发的一站式解决方案！Qt 虽然庞大，封装层次较深，但其速度并不慢。它虽不及 MFC，但比 Java、C#要快。Qt 程序在运行前最终会编译成本地计算机的可执行代码，而不是依托虚拟机来运行。Qt 的工具家族丰富，目前包括 Qt Creator、QtEmbedded、Qt Designer 快速开发工具、Qt Linguist 国际化工具等。

Qt 非常适合跨平台开发领域，是国内 C++程序员要掌握的第二主流开发工具（第一要掌握的主流开发工具是 Visual C++）。Qt 的最新版本可以从其官网（https://www.qt.io/）上下载，笔者在

编写本书时的最新版本是 Qt 6，但使用 Qt6 还是 Qt5，笔者考虑了很久。

由于 Qt 6 不支持 Windows 7，而国内很多人还在使用着 Windows 7，因此笔者决定使用 Qt5 来开发 OpenCV。毕竟，本书不是主要介绍 Qt 本身，所以即使使用 Qt5 来开发 OpenCV，使用 Qt6 的读者也完全可以学习本书，并且本书源码也是能够在 Qt6 上运行的。

另外，Qt6 还更改了授权方式，对于开发商业应用比较不友好。如果还在使用 QtWidgets，可以说完全没有必要升级到 Qt6，因为 Qt6 已经基本放弃了 QtWidgets。而如果是使用 QML 开发，那就必须升级到 Qt6 了，不然很多功能都是缺失的。当然，这些功能对于我们学习 OpenCV 来说根本没影响。

随着我国对软件自主可控要求的不断提高，使用 Qt 开发 C++应用变得越来越流行，尤其是在高新技术领域。而 OpenCV 作为常用的图形图像处理库，广泛应用于软件开发的高尖技术领域。因此，学习使用 Qt C++来开发 OpenCV 应用显得尤为重要。使用 Qt C++开发 OpenCV 应用还有一个显著的优势，即具备跨平台特性，这已成为当今商业软件的标准配置。

第 2 章

搭建 OpenCV 开发环境

工欲善其事，必先利其器。本章将搭建基于 Qt 的 OpenCV 开发环境。由于 Qt 和 OpenCV 的跨平台特性，因此将分别在 Windows 和 Linux 下进行环境搭建。建议读者在两种环境下都尝试搭建，因为本书后面有一些实例是在 Linux 下运行的，这也是笔者有意为之。毕竟视觉处理软件很大一部分是在嵌入式设备中运行是，比如无人机系统，因此我们也要重视 Linux 下的 OpenCV 开发。

2.1 Windows 下搭建 OpenCV 开发环境

2.1.1 下载和安装 Qt

对于图像视觉编程来说，如果所有事情都要从头开始写，那结果将是灾难性的。幸运的是国际开源界已经为我们提供了一个强大的视觉函数库——OpenCV。

本书使用 Qt 5.14.2，这是最后一个由官方提供安装包文件的版本了，从 Qt 5.15 开始要自己进行源码编译才能使用。下载 Qt 5.14 的官方网址如下：

```
    https:// download.qt.io/archive/qt/5.14/5.14.2/qt-opensource-windows-
x86-5.14.2.exe
```

如果直接把这个网址输入浏览器中，会得到这样的提示：

```
    Download from your IP address is not allowed
```

因为 Qt 官方禁止我们下载了，可以把这个网址放到迅雷中下载。如果这个下载地址失效了，也可以向笔者求助。

下载下来的文件名是 qt-opensource-windows-x86-5.14.2，直接双击就可以开始安装，但安装过程需要连接互联网，因为要在线验证账号。具体安装步骤如下：

步骤 01 直接双击 qt-opensource- windows-x86-5.14.2.exe 文件，显示欢迎安装界面，如图 2-1 所示。

步骤 02 单击 Next 按钮，出现 Qt Account 对话框，此时提示输入 Qt 账号（可以到官网上去注册）。输入账号后，一直单击 Next 按钮，直到出现“安装文件夹”对话框，在该对话框中设置 Qt 安装路径，这里安装在 C:\Qt\Qt5.14.2，如图 2-2 所示。需要注意的是，安装路径中不要有空格或中文。

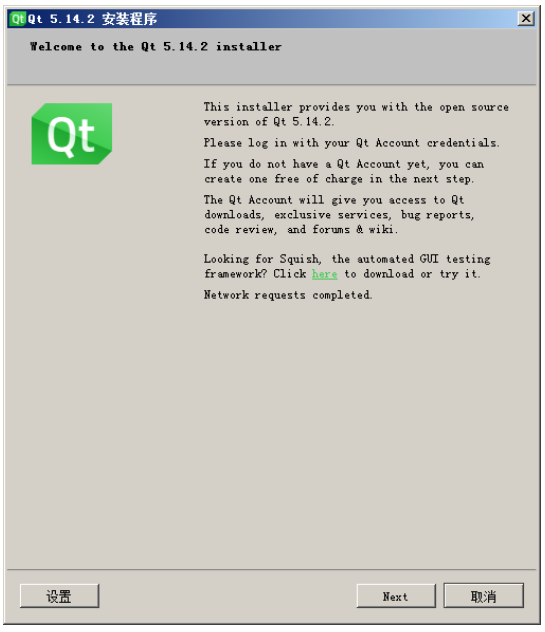

图 2-1

图 2-2

在安装 Qt 5.14 时会自动安装 Qt Creator（开发 Qt 程序的 IDE），如果要将 Qt 开发对应的常见文件类型关联到 Qt Creator（即用 Qt Creator 打开文件），那么在图 2-2 所示的窗口中就要勾选左下角的复选框。

步骤 03 继续单击“下一步”按钮，此时出现“选择组件”对话框，主要是选择开发时所需要的编译器，包含调试器的集成开发环境，如图 2-3 所示。我们需要勾选 Qt 5.14.2 下的 Sources 和 MinGW7.3.0 64-bit，以及 Developer and Designer Tools 下的 Qt Creator4.11.1 CDB Debug....。

（1）Sources 表示 Qt 的源码，因为我们要编译源码，所以要勾选该选项。

（2）MinGW 即 Minimalist GNU For Windows，是一些头文件和库的集合，该集合允许人们在没有第三方动态链接库的情况下使用 gcc 编译器产生 Win32 程序。通俗地讲，它就是开源的 C 语言编译器 gcc 的 Windows 版本，可以将我们的应用程序源码编译为可在 Windows 中运行的可执行程序，即在 Windows 平台上使用的 gcc 编译器。7.3.0 是 MinGW 的版本号，64-bit 表示这个 MinGW 可以编译生成 64 位或 32 位可执行程序。而 32-bit 只能编译生成 32 位可执行程序。

> **注意**　必须勾选 Qt 5.14.2 下的 MinGW 7.3.0 64-bit，虽然 Developer and Designer Tools 下也有 MinGW 7.3.0 64-bit，但两者含义是不同的。勾选 Qt 5.14.2 下的 MinGW 7.3.0 64-bit 后，会在安装目录（这里是 C:\Qt\Qt 5.14.2）下生成子文件夹“5.14.2\mingw73_64”和“Tools\mingw730_64”，前者存放开发 Qt 所需的且是 MinGW 编译的库和头文件，以及一些和 Qt 相关的工具程序；后者存放 gnu/gcc 工具链和有限的一些库和头文件，且这些库和头文件不是 Qt 库和 Qt 头文件，和 Qt 无关。因此，“5.14.2\mingw73_64”文件夹中的 mingw73_64 的含义是该文件夹下的 Qt 相关的库文件和工具程序都是 MinGW 编译生成的；而“Tools\mingw730_64”文件夹中的 mingw730_64 的含义是这个文件夹包含的是 MinGW 本身这个 GNU 工具链。

如果我们只勾选 Developer and Designer Tools 下的 MinGW 7.3.0 64-bit，那么只会在安装目录下生成子文件夹“Tools\mingw730_64”，里面仅包含 gnu/gcc 等工具链和有限的一些库和头文件。显然，仅凭这些东西无法直接开发 Qt 应用程序，因为没有 Qt 库和头文件给我们使用。

因此，要开发 Qt 应用程序，就必须勾选 Qt 5.14.2 下的 MinGW 7.3.0 64-bit，否则无法使用 Qt 中的函数和头文件。通常也把子文件夹所包含的内容叫作构建套件（kit），也就是一个开发包。

另外，如果读者不喜欢用 MinGW，那可以勾选 MSVC，这也是微软出品的编译器，一般配合 VC++这个 IDE 时才用这款编译器。这里我们用 Qt Creator 这个 IDE。

步骤 04　Qt Creator4.11.1 CDB Debug 是带有 CDB 调试器的 Qt 集成开发环境。

步骤 05　接着一直单击“下一步”按钮，然后就开始正式安装，稍等片刻即可安装完成，如图 2-4 所示。如果在图 2-4 中勾选 Launch Qt Creator 复选框，那么该对话框关闭后会启动 Qt Creator。保持默认选中状态，然后单击“完成”按钮，此时将启动 Qt Creator，如图 2-5 所示。

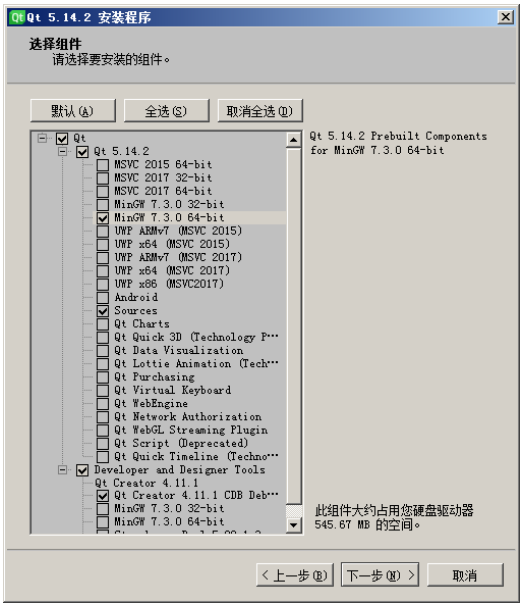

图 2-3

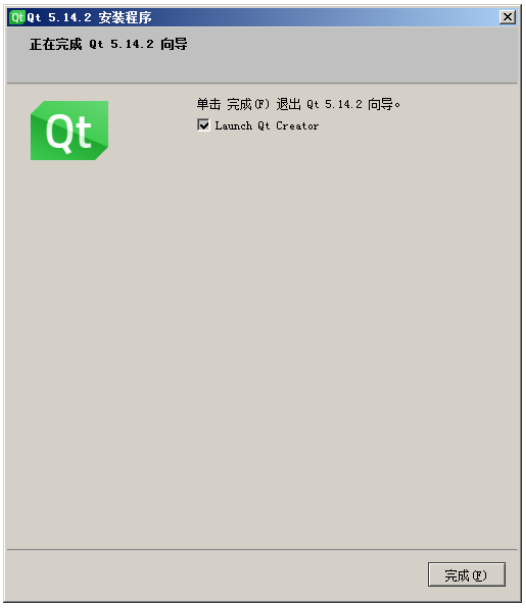

图 2-4

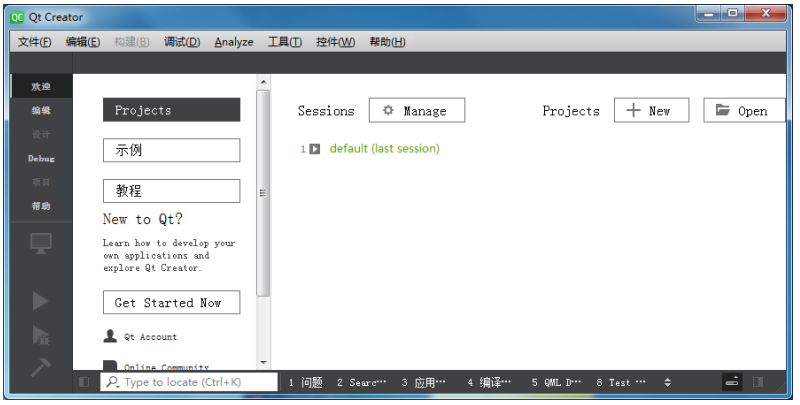

图 2-5

至此，Qt 5.14 安装成功。以前用过 Qt 的读者应该知道，Qt Creator 是老牌的 Qt 开发环境了，功能也日趋强大。下面我们用 Qt Creator 新建一个 Qt 图形界面程序，以此来测试 Qt 是否工作正常。

【例 2.1】第一个 Qt 程序

1）*启动* Qt Creator

步骤 01 依次单击主菜单栏中的“文件→新建文件或项目”或直接按 Ctrl+N 快捷键，打开“新建项目”对话框，然后在左侧选择 Application，在右侧选择 Qt Widgets Application，如图 2-6 所示。Widgets 是小部件的意思，或者说是控件，因此 Qt Widgets Application 称为 Qt 控件程序。它提供了一组 UI 元素，用于创建经典的桌面风格的用户界面。

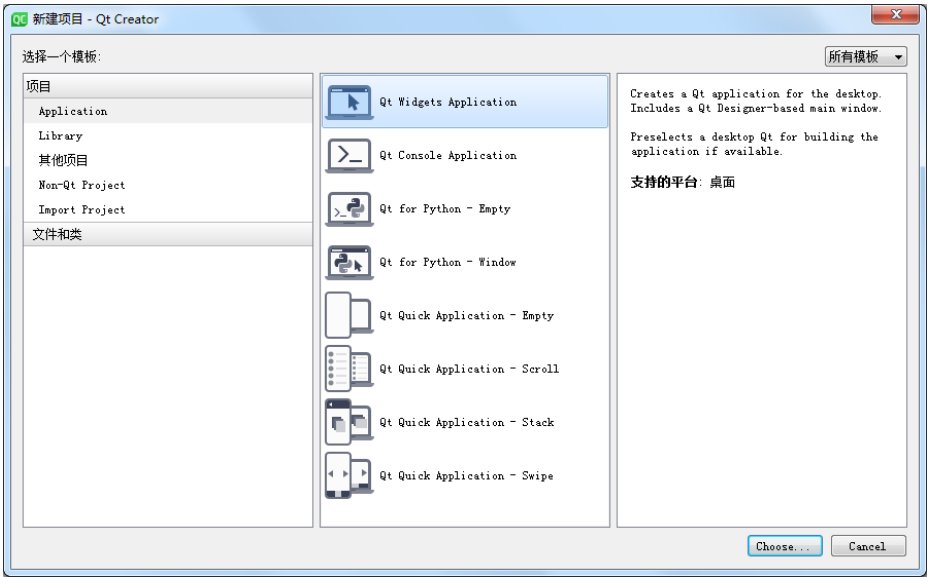

图 2-6

步骤 02 单击 Choose…按钮，在新出现的 Location 对话框上设置项目名称和路径，如图 2-7 所示。

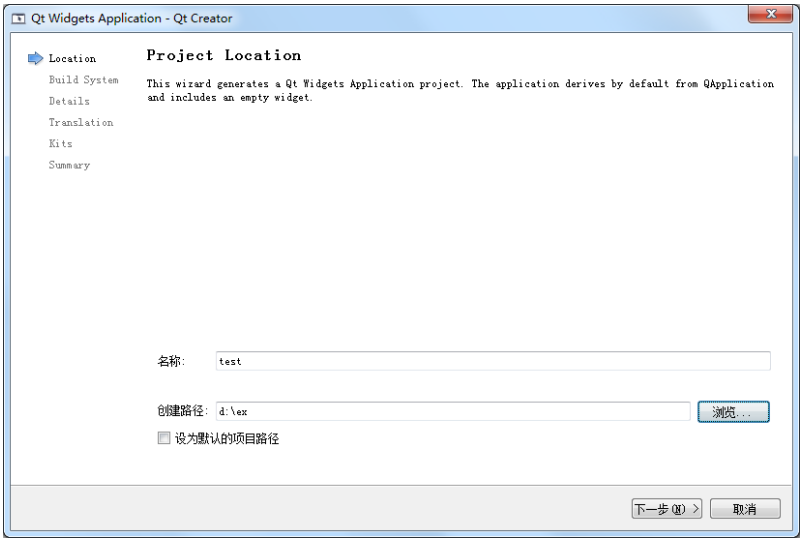

图 2-7

这个路径和目录必须预先创建好，否则不能进行下一步操作，这也是 Qt Creator 不够智能的地方，不会自动帮我们创建目录。

步骤 03　单击“下一步”按钮，在新出现的 Build System 对话框中显示构建工具（qmake），且已经自动探测到了，我们不需要去选择，保持默认即可。

步骤 04　继续单击“下一步”按钮，在新出现的 Details 对话框中显示类信息（比如类名、基类等），如图 2-8 所示。

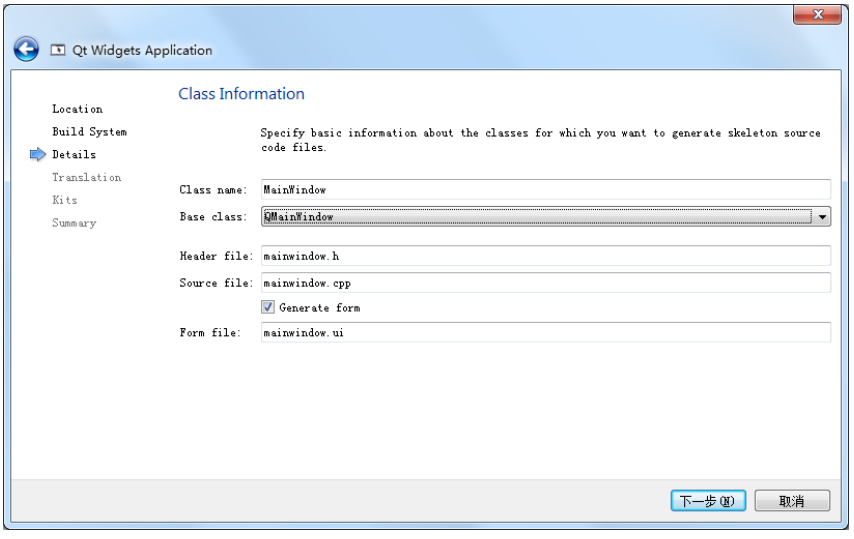

图 2-8

Qt 程序一般由头文件、cpp 源文件和.ui 界面文件组成，前两者就是存放代码的文件，.ui 界面文件则是用于可视化界面设计的，比如拖放控件等。

步骤 05 继续单击“下一步”按钮，出现 Translation 对话框，在该对话框上可以选择应用程序的语言，比如英文，这里保持默认。

步骤 06 再单击“下一步”按钮，出现 Kits 对话框，该对话框显示了当前可用的开发包，因为我们安装 Qt 的时候选择了基于 MinGW 的 64 位开发包，所以现在自动探测出来了，如图 2-9 所示。

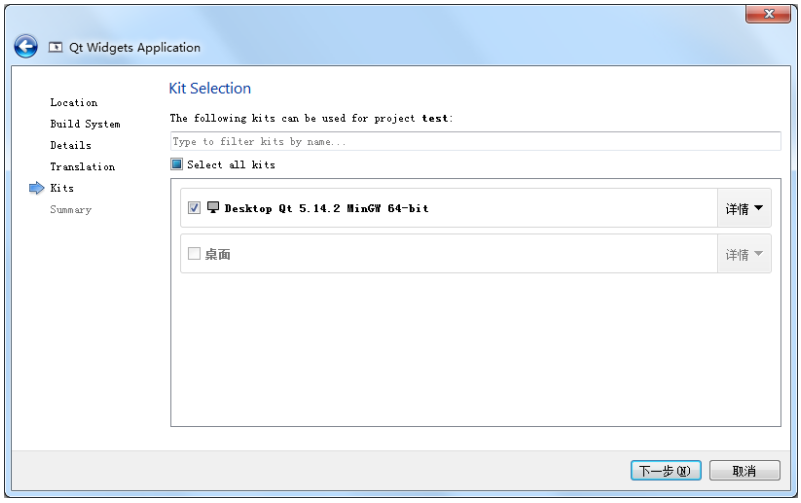

图 2-9

步骤 07 再单击“下一步”按钮，出现 Summary 对话框，如图 2-10 所示。

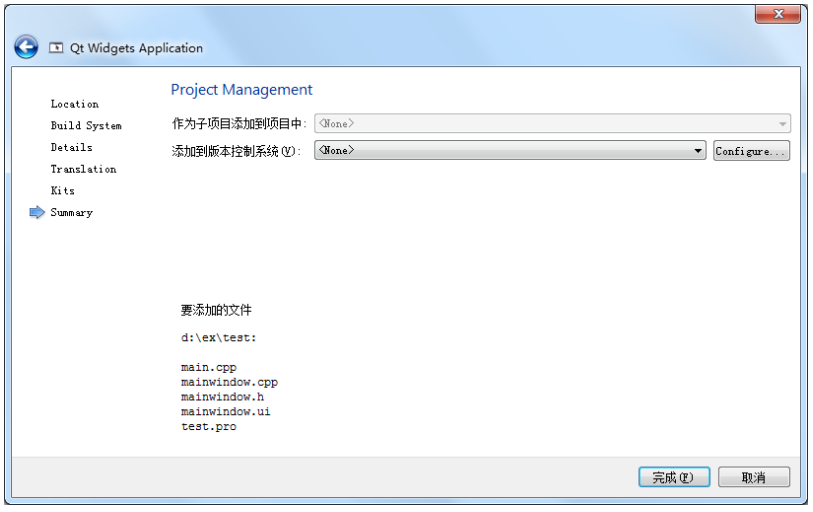

图 2-10

注意该对话框下方显示的要向 d:\ex\test 这个目录添加的文件。其中：

- main.cpp 是包含 main 函数的 C++源文件。
- mainwindow.cpp 是绘制主窗口的 C++源文件。
- mainwindow.h 是对应的头文件。

- mainwindow.ui 是主窗口的描述文件。Qt 中的 UI 文件是一种特殊的 XML 格式文件，包含了界面上各种控件的信息，如按钮、文本框、下拉菜单等，以及它们的布局和属性设置，用于描述应用程序的用户界面。这些文件可以使用 Qt 的可视化设计工具 Qt Designer 来创建和编辑。Qt Designer 提供了直观的界面，允许用户通过拖放和配置界面元素的方式来设计应用程序的图形用户界面（GUI）。UI 文件可以在 Qt 应用程序中通过 Qt User Interface Compiler（uic）工具转换为 C++代码，供应用程序使用。
- test.pro 是项目（描述）文件。在 Qt 中，.pro 文件（也称为项目文件）是 Qt 项目管理系统（qmake）所使用的配置文件。这个文件定义了如何构建 Qt 应用程序或库，它使用简单的键值对语法，允许我们指定源文件、头文件、库依赖、配置选项等。qmake 是 Qt 工具包中自带的一个非常方便的工具，用于读取.pro 文件，并生成一个标准的 Makefile（在 Unix-like 系统上），还可以生成 Microsoft Visual Studio 可以使用的项目文件等，以便可以使用相应的构建工具（如 make 或 IDE）来构建项目。最关键的是它可以自动解决依赖关系，不用手写 Makefile，而且是跨平台的。

步骤 08　最后单击“完成”按钮，完成 Qt 的安装，编辑代码窗口会自动打开 main.cpp 文件，并显示了 main.cpp 中自动生成了代码。我们来看一下 main 函数：

```
01#include "mainwindow.h"
02#include <QApplication>
03
04int main(int argc, char *argv[])
05{
06    QApplication a(argc, argv);
07    MainWindow w;
08    w.show();
09    return a.exec();
10}
```

代码解释如下：

- 第 02 行代码引入所有 Qt 程序都要包含的头文件 QApplication。
- 第 06 行代码创建一个 QApplication 对象，用于管理应用程序的资源，这对于运行任何使用 Qt Widgets（控件）的 Qt 程序都是必需的；对于不使用 Qt 控件的 GUI 应用程序，可以改用 QGuiApplication。
- 第 07 行代码创建主窗口对象。类 MainWindow 是安装向导为我们生成的，该类封装一个主窗口，在主窗口上可以添加各类控件。常见的控件包括文本编辑、滚动条、标签和按钮等。
- 第 08 行代码调用 show 这个成员函数在屏幕上显示主窗口。
- 第 09 行代码使 QApplication 对象进入其事件循环。当 Qt 应用程序运行时，会生成事件并将事件发送到应用程序的控件。常见的事件有鼠标按下和键盘按键。

其他源文件中的内容就不叙述了，毕竟我们的主要目标不是 Qt 本身。

2）运行项目

步骤 01　按快捷键 Ctrl+R 或依次单击主菜单栏中的“构建→运行”选项，也可以单击 Qt Creator

左下角的三角形按钮，该按钮是运行的快捷键，功能和 Ctrl+R 一样，如图 2-11 所示。

步骤 02 如果修改过代码，那么第一次运行时，会弹出对话框提示保存修改，如图 2-12 所示。

图 2-11

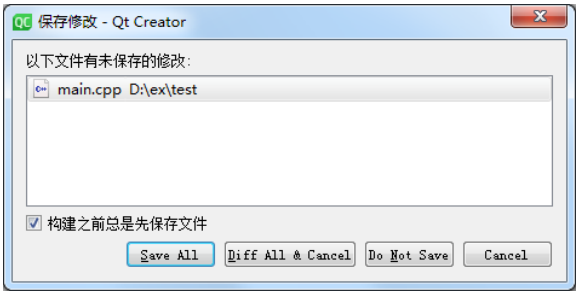

图 2-12

勾选“构建之前总是先保存文件”复选框，这样以后如果修改了代码，运行前就可以自动保存文件，而不需要我们手动去保存了。

步骤 03 最后单击 Save All 按钮。

运行结果如图 2-13 所示。

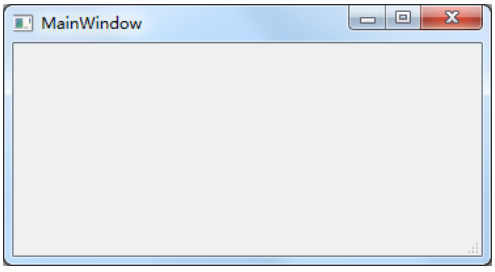

图 2-13

此时，在 d:\ex 下新生成了两个文件夹，test 和 build-test-Desktop_Qt_5_14_2_MinGW_64_bit-Debug。

test 文件夹下存放的是工程源码，比如 main.cpp、mainwindow.cpp，还存放有工程文件 test.pro。双击 test.pro 就可以直接通过 Qt Creator 来打开整个工程；也可以先打开 Qt Creator，然后在 Qt Creator 中打开 test.pro。

build-test-Desktop_Qt_5_14_2_MinGW_64_bit-Debug 文件夹中存放的是编译生成的可执行文件和其他临时文件，这个文件夹在 Qt Creator 重新编译运行工程的时候会被清空。我们可以在 build-test-Desktop_Qt_5_14_2_MinGW_64_bit-Debug\debug\下找到 test.exe 这个可执行文件。

上面的例子比较简单，我们没有写代码，也没有拖放控件到窗口上。该实例的主要目的是测试刚安装好的 Qt 的开发功能是否工作正常。

2.1.2 下载和解压 OpenCV

Qt Creator 这个 IDE 准备好了后，就可以准备 OpenCV 了。OpenCV 可以到其官网（https://opencv.org/releases/）下载，打开首页后就可以找到 OpenCV – 4.10.0，如图 2-14 所示。

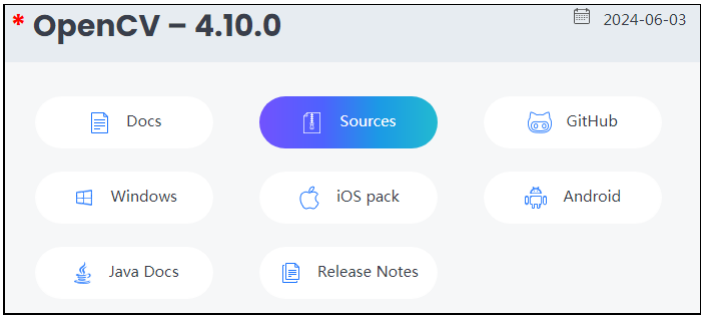

图 2-14

其中：

- Docs：一些英文文档，比较晦涩难懂。
- Sources：是当前版本的 OpenCV 源码。
- Windows：Windows 环境下编译好的文件（build）+源码。
- iOS pack：iOS 环境下编译好的文件（build）+源码。
- Android：Android 环境下编译好的文件（build）+源码。

在 OpenCV– 4.10.0 下，单击 Sources，等 3 秒后开始下载。下载下来的文件是 4.10.0.zip，这是个压缩包文件。如果读者不想下载，可以到随书源码的 somesofts 文件夹中找到 4.10.0.zip 文件。

笔者的计算机上已经安装了 WinRAR 软件，准备用它来解压 opencv-4.10.0.zip 文件。把 4.10.0.zip 复制到 D:\，然后对其右击，在弹出的快捷菜单中选择“解压到当前文件夹”，解压后的目录为 D:\opencv-4.10.0。进入这个文件夹，就可以看到各个不同的文件夹和文件了。

我们需要通过源码编译出 MinGW 能识别的 OpenCV 库，从而让程序可以调用这些库，因为程序也将由 MinGW 来编译。

2.1.3　了解构建工具

相信读者都学过 C/C++语言了，对编译链接过程已经很熟悉了，也就是“编译预处理→编译→链接”。然而，编译 OpenCV 可不是这么简单，因为它是源码文件很多，依赖关系复杂。因此我们需要一个能有效管理编译过程的工具。这个工具就叫作构建工具，比较著名的构建工具是 make 程序。

与构建相关联的是编译，编译是将一个源文件转换成一个二进制文件，而构建就是对于编译的安排。在一个大的工程（比如 OpenCV 源码）中，通常包含很多源文件，其中可能还包含复杂的依赖关系，构建就是对于多个源文件进行编译的合理安排。

为什么需要构建工具呢？大多数工程师在学习编程时都使用非常简单的例子，而这个例子可能就只有一个源文件，因此大多数工程师开始都是直接调用 gcc 或 javac 等工具，或者使用 IDE 中提供的便捷的编译工具，例如以下例子就是将同一目录的源码转换为二进制文件：

```
gcc *.c -o test
```

gcc 非常智能，可以在当前目录的子目录中查找要导入的代码，但是它找不到文件系统的其他

部分中存储的代码（或由多个项目共享的库）。它还只知道如何构建 C 代码。大型系统通常使用各种编程语言编写不同部分，并且这些部分之间具有网络，这意味着单一编译器无法构建整个系统。当工程日趋复杂，一个简单的编译命令无法满足要求而需要多个编译命令的组合时，可能会想到使用 Shell 脚本来组织编译命令，这些脚本会按正确的顺序构建应用。然而，随着工程的进一步膨胀，Shell 脚本也显得力不从心，会遇到很多问题：

（1）构建变得很烦琐。随着系统变得越来越复杂，在构建脚本上花费的时间几乎与编写代码一样。调试 Shell 脚本非常痛苦，并且越来越多的技巧层层叠加。

（2）速度很慢。为了确保应用不会意外依赖过时的库，需要让构建脚本在每次运行时按顺序构建每个依赖项。因此需要考虑添加一些逻辑来检测哪些部分需要重新构建。这听起来非常复杂，并且对于脚本来说也很容易出错，脚本会变得越来越复杂，导致难以管理。

笔者的理解是从最初的 gcc、shell 脚本到现在完善的构建工具，都是对于编译过程的进一步抽象，以使编译与构建更加易于理解和维护。底层做的事情都是一样的，是为了让人更好理解和维护。

Make 是一个命令行工具和构建自动化工具，用于管理和构建软件项目。它通过一个 Makefile 脚本文件来定义构建任务和依赖关系，然后根据这些规则自动执行任务，以生成最终的目标文件。Make 可以自动化构建、任务并行执行、控制环境变量，并可以在多种操作系统上运行。它具有依赖管理功能，确保任务的正确构建顺序。

Make 的作用就是一个自动化的构建工具，它告知编译器正确的处理顺序，保证每个依赖都能得到处理，最后生成相应的工程。而 Makefile 的作用相当于一张图纸，它用来告诉 Make 具体怎样处理编译过程。

如果只是一个小型的工程，比如就几个源码文件，那么不使用 Make 只依靠编译器并添加几个指令也完全可以处理。但是，如果是一个较大的项目，里面有复杂的依赖关系，此时只依靠靠编译器并添加几个处理指令的方式已经不太现实了，使用 Make 和 Makefile 来进行自动化构建才是正确的选择。

那么 MinGW 提供构建工具了吗？答案当然是肯定的，在路径 C:\Qt\Qt5.14.2\Tools\mingw730_64\bin\下可以看到 mingw32-make.exe 程序，它就是 MinGW 提供的构建工具。

2.1.4 下载和安装 CMake

2.1.3 节提到了 Make 是用来处理编译顺序和依赖关系并构建最终项目的工具，但是问题来了，不同的平台会有不同的构建方式，例如，Linux 生成最终的二进制程序和.so 动态链接，而 Windows 生成.exe 程序和 dll 动态链接；同时两者的依赖也可能不同。如果在不同的平台上写不同的 Makefile，会相当麻烦；而且 Makefile 文件本身的写法也很复杂，在大型工程中较难阅读和理解。此时 CMake 就派上用场了。

CMake 是“cross platform make”的缩写。虽然名字中含有“make”，但是 CMake 和 UNIX 上常见的 make 系统是分开的，而且更为高阶。CMake 本身不执行 Make 过程，而是根据不同平台的特性生成对应的 Makefile，这样每个工程只要写一个 CMake 文件即可，其余的交给不同平台的处理器来产生不同的 Makefile 文件；而且 CMake 的语法也更加简洁，适合阅读。

Make 和 CMake 的关系如图 2-15 所示。

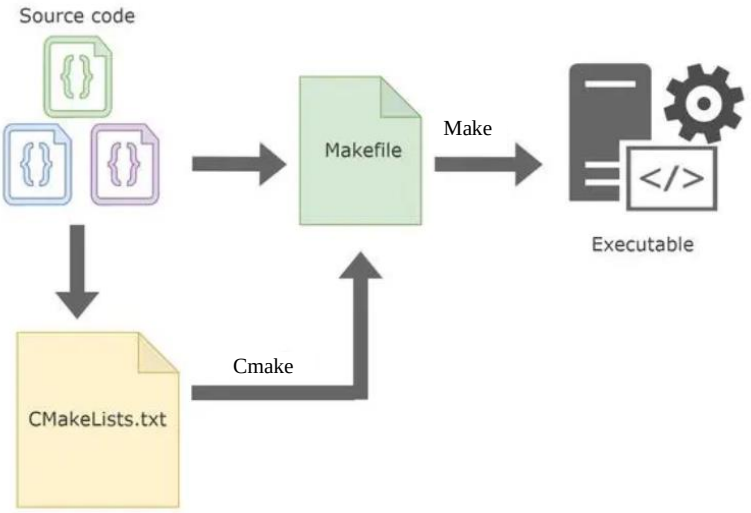

图 2-15

总之，Cmake 是生成 Makefile 文件的工具，Make 则是根据 Makefile 执行构建的工具。在使用 CMake 生成构建文件后，可以使用相应的构建工具（如 Make、Visual Studio 等）执行实际的编译过程，将源码编译为可执行文件或库。

在 Window 上使用 CMake 比较方便，因为有图形界面版本的 CMake，一般称为 CMake GUI。我们可以到 CMake 官网（https://cmake.org/download/）下载 CMake GUI。下载下来的是一个安装包文件 cmake-3.30.2-windows-x86_64.msi。如果读者不想下载，也可以在随时源码的 somesofts 文件夹下找到该安装包。安装过程很简单，直接双击安装包即可开始安装。需要注意的是，要在 Install Options 对话框上勾选 Add CMake shortcut to the Desktop 复选框，如图 2-16 所示。这样，安装后可以在桌面上生成一个快捷方式。

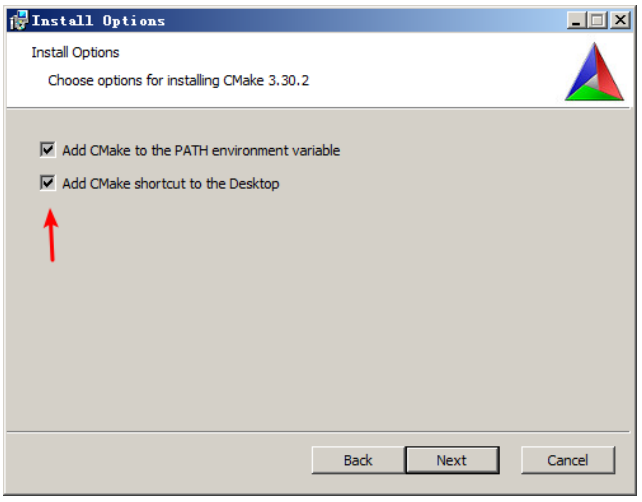

图 2-16

安装完成后，要配置环境变量，将 QT 的 mingw32 编译器的路径添加到 Path 环境变量中（注意：QT 有两个路径），如下所示：

```
C:\Qt\Qt5.14.2\5.14.2\mingw73_64\bin
C:\Qt\Qt5.14.2\Tools\mingw730_64\bin
```

添加完毕后，重启计算机使之生效。

2.1.5 生成 Makefile 文件

在用 CMake 生成 Makefile 文件之前，我们要做一项重要的工作。CMake 在配置阶段会到国外一些网站下载一些文件放到一个名为“.cache”的文件夹中，这个文件夹是 CMake 在 OpenCV 源码根目录下自动新建的。但由于国内下载文件比较慢，所以经常会出现 CMake 卡死的情况。解决的方法是通过下载日志文件 CMakeDownloadLog.txt 把下载失败的文件下载地址记录下来，然后复制到迅雷中下载；下载完毕后，将文件名改为“md5-filename”的形式，然后放到“.cache”文件夹中的相应位置。再次启动 CMake 配置时，CMake 看到相应位置已经有文件了，就不会再去下载了，从而让 CMake 能继续工作下去。

我们可以在 somesofts 文件夹中找到.cache 文件夹，然后把它复制到 D:\opencv-4.10.0 下。

现在可以使用 CMake 生成 Makefile 文件了。

步骤 01 在桌面上双击 CMake (cmake-gui)，打开 cmake-gui 界面，单击 Where is the source code 右端的 Browse Source...按钮，选择 opencv 源码的文件夹，这里是 D:\opencv-4.10.0。再单击 Where to build the binaries 右端的 Browse Build...按钮，选择一个新建的文件夹，用于保存编译后的文件和编译过程中产生的中间文件。笔者在 D 盘下新建了一个文件夹 opencvBuild。最终选择后的界面如图 2-17 所示。

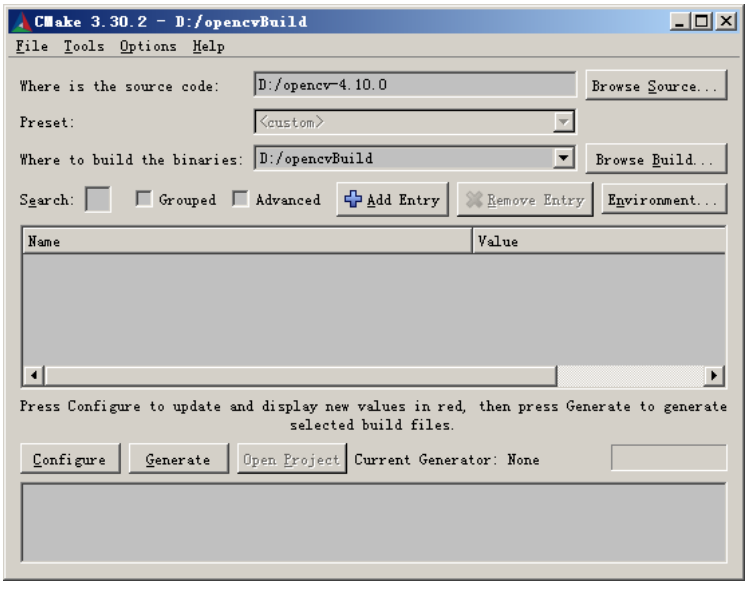

图 2-17

步骤 02　单击左下方的 Configure 按钮，在出现的 CMakeSetup 对话框上做如图 2-18 所示的选择。

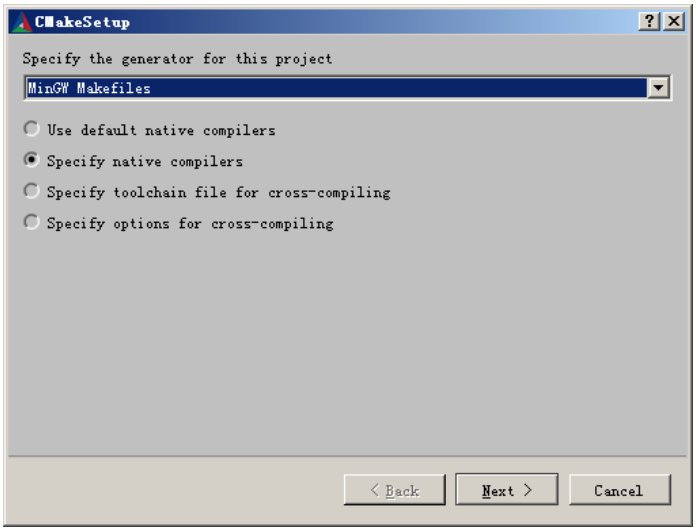

图 2-18

步骤 03　单击 Next 按钮，在新的对话框中选择 gcc 和 g++的路径，分别是：

```
C:/Qt/Qt5.14.2/Tools/mingw730_64/bin/gcc.exe
C:/Qt/Qt5.14.2/Tools/mingw730_64/bin/g++.exe
```

如图 2-19 所示。

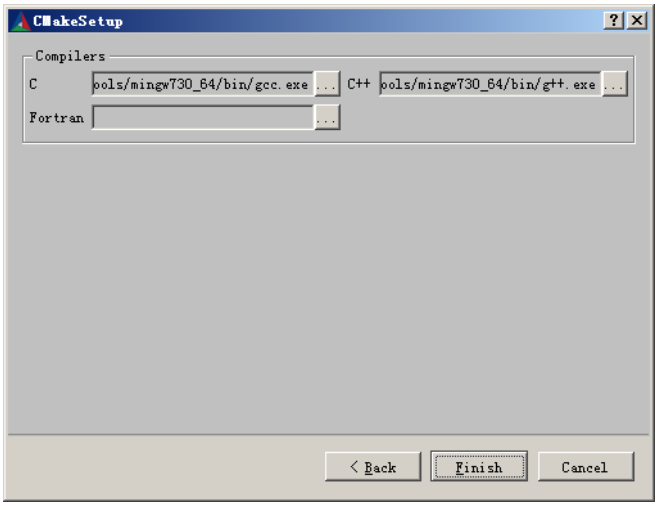

图 2-19

步骤 04　最后单击 Finish 按钮，就可以 CMake 在工作了，如图 2-20 所示。

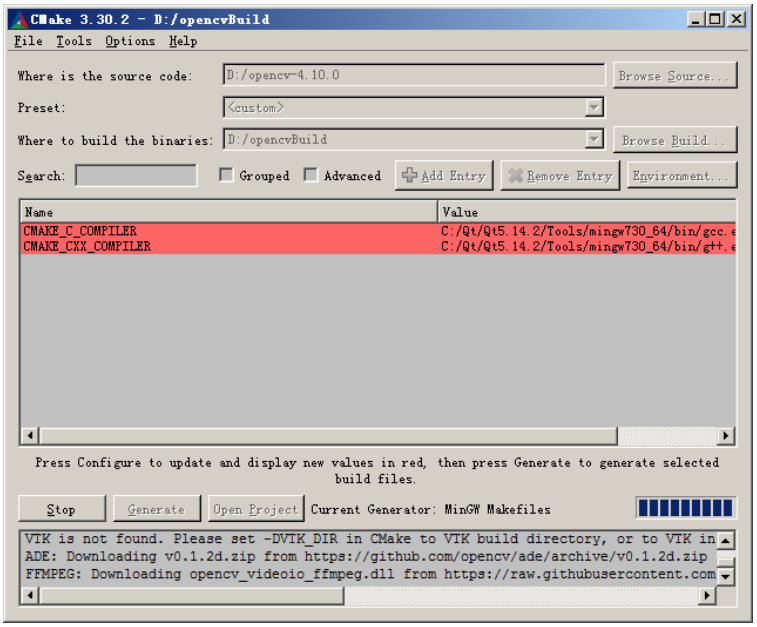

图 2-20

经过 100 多秒后，配置结束，如图 2-21 所示。

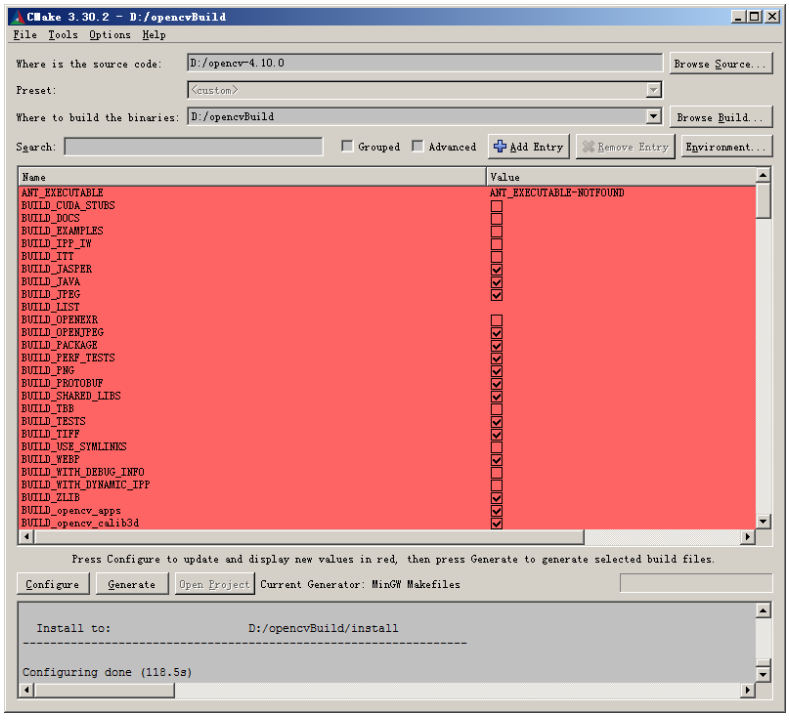

图 2-21

步骤 05 下面修改一些配置，以便将来编译时对不需要的组件就不用去编译了。具体修改如表 2-1 所示。

表 2-1　修改配置

配置项	值
JAVA	BUILD JAVA（不选） BUILD_opencv_java_bindings_generator（不选）
WITH	WITH_OPENGL（选中）
WITH	WITH_QT（选中）
WITH	WITH_IPP（不选）
BUILD_PROTOBUF	不选
WITH_PROTOBUF	不选
OPENCV_GENERATE_SETUPVARS	不选

步骤 06　再次单击 Configure 按钮进行配置，配置完毕后，单击 Generate 按钮，当出现 Generating done 时说明 Makefile 生成成功了，该文件可以在 D:\opencvBuild 下找到。关闭 CMake。

2.1.6　开始编译 OpenCV

下面准备编译 OpenCV。

步骤 01　打开命令行窗口，进入 D:\opencvBuild，开始执行编译命令：

```
mingw32-make -j 8
```

-j 这个选项用来指定线程数，这里指定 8 个线程进行编译，这样可以快一点。这个命令耗时有点长，稍等片刻，编译完成，如图 2-22 所示。

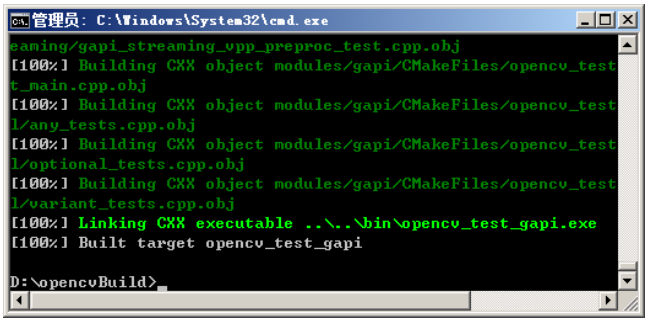

图 2-22

步骤 02　编译完成之后，输入如下命令进行安装：

```
mingw32-make install
```

稍等片刻，安装完成，我们可以在 D:\opencvBuild\install\x64\mingw 下看到 bin 和 lib 两个文件夹，这两个文件夹下的内容是我们稍后编程需要用到的。其中 bin 下的主要是.dll 文件，这些文件是程序运行时所要链接的动态链接库；lib 下的主要是.a 静态库文件，这些文件是程序编译时需要用到的文件。

2.1.7 Qt 开发的第一个 OpenCV 程序

准备工作终于完成了，现在可以开始用 Qt 编写 OpenCV 程序了。

【例 2.2】Qt 开发的第一个 OpenCV 程序

1）运行 Qt 项目

步骤 01 在桌面上找到 Qt Creator 4.11.1 (Community)，双击打开 Qt Creator，新建一个控制台项目，项目名是 hello。在 Qt Creator 主界面的 Projects 旁单击 New 按钮，如图 2-23 所示。

图 2-23

步骤 02 在出现的“新建项目”对话框中选中 Qt Console Application，如图 2-24 所示。

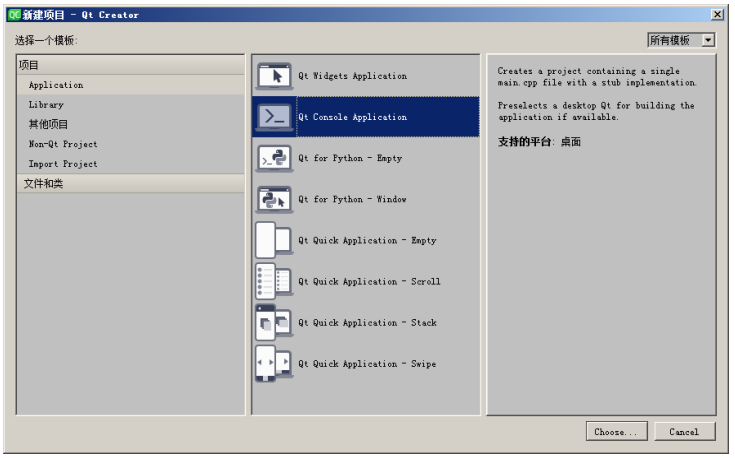

图 2-24

步骤 03 单击 Choose 按钮，在新的对话框中输入项目名称和路径，建议用全英文路径，如图 2-25 所示。

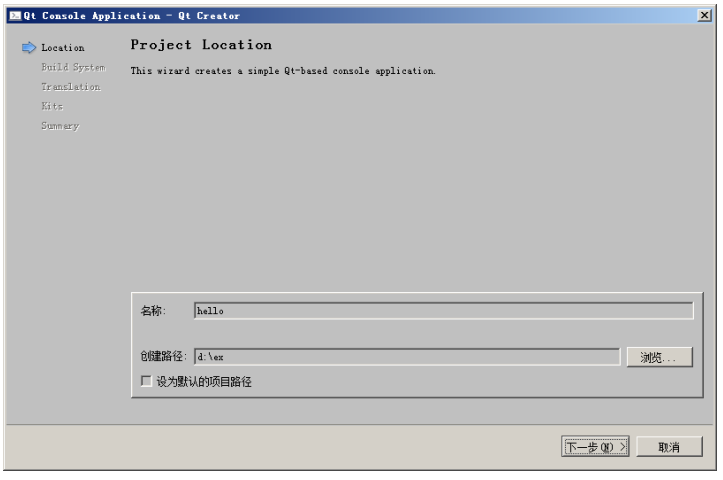

图 2-25

步骤 04　随后连续单击“下一步”按钮，直到向导结束出现编辑窗口，此时可以看到自动生成了一个 main 函数。我们来简单修改一下程序，使之既能输出图形对话框，又能在命令窗口中输出字符串，如下代码：

```
#include <QApplication>
#include <QMessageBox>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);          // 定义应用程序对象
    QString text = "helloworld";         // 定义一个字符串并赋值
    // 创建并显示消息对话框
    QMessageBox::information(nullptr, "hi", text);
    puts("helloworld");                  // 在控制台窗口输出字符串
    return a.exec();                     // 进入事件循环
}
```

QApplication 是 Qt 应用程序中必不可少的一部分，它负责管理应用程序的生命周期，处理事件循环，设置应用的整体样式等。QString 是 Qt 框架中处理文本数据的核心类之一。它提供了强大的字符串处理能力，支持多种文本编码和操作。在这个示例中，我们定义了字符串“text”，并通过 QMessageBox::information 将其在消息对话框上输出；然后，用 C 函数 puts 在控制台上输出字符串“helloworld”。

步骤 05　由于这个项目原本是控制台项目，因此如果要输出图形消息框，还需要在项目配置文件中进行一些修改。打开 hello.pro，在文件开头添加如下代码：

```
QT+=widgets
```

widgets 是在 Qt 中创建用户界面的主要元素，它可以显示数据和状态信息，接收用户输入，并为应该组合在一起的其他小部件提供容器。QT+的意思是加上 widgets 模块。如果此时运行程序，则可以出现消息对话框并可以在控制台上输出字符串“helloworld”。

2）添加 OpenCV 代码

Qt 项目能运行了，但还不是一个 OpenCV 程序，下面加上 OpenCV 代码。

步骤 01　在工程配置文件中添加 OpenCV 库和头文件的路径，在 hello.pro 的末尾添加如下代码：

```
INCLUDEPATH += D:/opencvBuild/install/include/
LIBS += -L D:/opencvBuild/install/x64/mingw/lib/libopencv_*.a
```

宏 INCLUDEPATH 用来指定头文件所在路径，LIBS 用来指定库文件所在路径，这两个宏都要用一个+，并且 LIBS+=右边要用-L 来指定路径，这个写法和 gcc 指定库路径写法类似。

步骤 02　下面添加源码，在 Qt Creator 中打开 main.cpp，输入如下代码：

```
#include <QApplication>
#include <QMessageBox>
#include <opencv2/opencv.hpp>
#include <QDebug>
#include <QDir>
```

```
#include <iostream>

using namespace cv;     // 所有 OpenCV 类都在命名空间 cv 下
using namespace std;

void f()                // 自定义函数，实现两幅图片的混合
{
    double alpha = 0.5; double beta; double input;
    Mat src1, src2, dst; // 创建 Mat 对象，Mat 用于存储图片的矩阵类，dst 用于存放混合后的图片

    // 提示用户输入第一幅图片的权值 alpha
    qDebug() <<"线性混合：";
    qDebug() << "输入第一幅图片的权重 alpha [0.0-1.0]: ";
    cin >> input;       // 用户输入

    // 如果用户输入值介于 0 和 1 之间，则用该输入值作为 alpha 的值
    if (input >= 0 && input <= 1)
        alpha = input;

    QString currentPath = QDir::currentPath();  // 获取当前路径
    qDebug() <<"当前路径是："<< currentPath;

    // 读取两幅大小必须一样的 JPG 图片
    src1 = imread("p1.jpg");
    src2 = imread("sbh.jpg");

    if (src1.empty()) { cout << "Error loading src1" << endl; return; }
    if (src2.empty()) { cout << "Error loading src2" << endl; return; }

    beta = (1.0 - alpha);
    addWeighted(src1, alpha, src2, beta, 0.0, dst);// 将图 1 与图 2 线性混合
    imshow("res",dst);  // 显示混合后的图片
    waitKey(0);         // 等待按键响应后退出，0 改为 5000 就是 5 秒后自动退出
}

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);     // 定义应用程序对象
    QString text = "helloworld";    // 定义一个字符串并赋值

    cv::Mat image = cv::imread("D:/test.jpg");  // 读取 D 盘上的 test.jpg 文件
    if (image.empty()) {            // 判断文件是否为空
       return -1;                   // 为空返回-1，结束程序
    }
   std::string strTitle = text.toStdString(); // 把 QString 字符串转为 std::string 字符串

   namedWindow(strTitle, cv::WINDOW_AUTOSIZE); // 创建一个标题是 strTitle 的窗口
   imshow(strTitle, image);         // 在标题为 strTitle 的窗口中显示图像
```

```
    waitKey(0);                    // 等待用户按键
    destroyWindow(strTitle);       // 销毁标题是 strTitle 的窗口

    // 创建并显示消息对话框
    QMessageBox::information(nullptr, "hi", text);
    f();
    return a.exec();
}
```

在上述代码中，自定义函数 f 的功能是将图片 p1.jpg 和 sbh.jpg 进行混合，它们的大小必须一样。这两幅图片必须复制到 D:\ex\build-hello-Desktop_Qt_5_14_2_MinGW_64_bit-Debug 目录下。其中，OpenCV 库中的 API 函数 imread 用来从文件中读取图片；API 函数 addWeighted 用于将两幅相同大小、相同类型的图片进行融合，第二个参数 alpha 表示第一幅图片所占的权重，第四个参数 beta 表示第二幅图片所占的权重。权重越大的图片显示得越多，比如我们输入 alpha 为 0.9，则主要显示第一幅图片。main 中的代码比较简单，主要是从 D 盘读取一个图片文件 test.jpg，然后调用 f。

3）复制动态链接库到 exe 文件所在文件夹

步骤 01　首先打开以下路径来查看 exe 文件：

```
D:\ex\build-hello-Desktop_Qt_5_14_2_MinGW_64_bit-Debug\debug
```

这个路径是程序生成的 hello.exe 文件所在的路径。hello.exe 是一个 Windows 下的可执行文件，MinGW 能在编译阶段加载 Linux 下的静态库（.a）文件，最终生成 Windows 下的.exe 文件。但 hello.exe 还需要加载 OpenCV 源码编译出来的动态链接库后才能运行。

步骤 02　打开目录 D:\opencvBuild\install\x64\mingw\bin\，把该目录下的 libopencv_core4100.dll、libopencv_highgui4100.dll、libopencv_imgcodecs4100.dll 和 libopencv_imgproc4100.dll 四个文件复制到 hello.exe 文件所在文件夹。

4）运行程序

在 Qt Creator 中单击左下角的三角运行按钮或直接按快捷键 Ctrl+R 来运行项目。运行结果如图 2-26 所示。

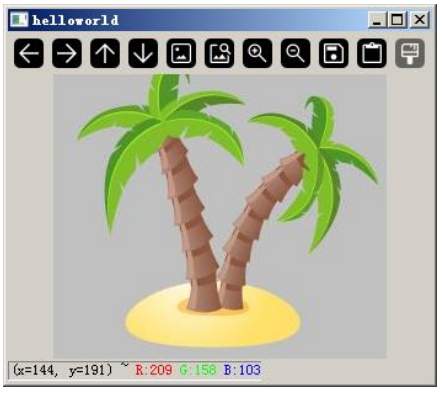

图 2-26

在键盘上随便按一个键，控制台窗口上则会显示 helloworld 信息框。关闭信息框，会让我们在控

制台窗口上输入权值，这里我们输入 0.9，可以看到 sbh.jpg 显示的效果就淡了很多，如图 2-27 所示。

图 2-27

如图图片不能加载，请确认 p1.jgp 和 sbh.jpg 是否复制到以下路径：

```
D:\ex\build-hello-Desktop_Qt_5_14_2_MinGW_64_bit-Debug
```

以及 test.jpg 是否复制到 D 盘。在这个程序中，我们既加载了绝对路径下的图片文件（test.jpg），也加载了当前路径下的图片文件（p1.jpg 和 sbh.jpg）。这 3 个文件可以在工程目录的 res 文件夹下找到。

至此，基于 Qt Creator 的 OpenCV 的开发环境搭建起来了。

2.2 Linux 下搭建 OpenCV 开发环境

本书使用 Qt Creator 开发 OpenCV，其最大的优点就是跨平台。当前，支持跨平台几乎是所有商业应用软件的标配。

在前面的章节中，我们使用 MinGW 在 Windows 环境下实现了 OpenCV 的编译。MinGW 本质上是基于 GCC 编译器，而 GCC 源自 Linux 系统。因此，既然我们能够在 Windows 上成功编译 OpenCV，那么在 Linux 下编译 OpenCV 源码更是小菜一碟了。当然，如果读者急于学习 OpenCV 的相关知识，可以暂时跳过本节，直接学习后面的内容。后续内容主要基于 Windows 下的 Qt Creator 进行介绍。本节的内容可以先保留，以便在未来进行 Linux 开发时（例如在银行、航天等关键领域的应用开发，通常都是在国产 Linux 系统上进行），再回过头来参考。此外，后续章节中的项目源码既可以在 Windows 上编译运行，也可以在 Linux 上编译运行。支持跨平台开发是本书的一大优点，同时也是

为将来在嵌入式开发板上进行 OpenCV 应用开发做好准备。

这里我们在经典的 Ubuntu20.04 上编译 OpenCV。为了节省成本，我们把 Ubuntu 安装在 VMware 虚拟机上，假定读者已经安装好，并以 root 账户登录到 Ubuntu 了，下面开始我们的编译工作。

2.2.1　准备编译 OpenCV 所需依赖

1. 基础依赖

编译 OpenCV 所需的依赖首先是两个编译相关的基础依赖：cmake 和 make，它们都和构建有关。build-essential 是一个软件包，它的主要作用是在系统上安装一组基本的编译工具（比如 gcc）和库文件（比如 C 语言运行时库等）。

```
    sudo apt install -y cmake make
    sudo apt install -y build-essential
```

下面这些依赖就不一一说明了，大部分都是关于图片分析和图片处理的依赖。

```
    sudo apt install -y libgtk2.0-dev
    sudo apt install -y libavcodec-dev
    sudo apt install -y libavformat-dev
    sudo apt install -y libjpeg-dev
    sudo apt install -y libtiff-dev
    sudo apt install -y libswscale-dev
    sudo apt install -y libpng-dev

    sudo add-apt-repository "deb <http:// security.ubuntu.com/ubuntu>
xenial-security main"
```

add-apt-repository 的意思是添加软件源（指向 Ubuntu 的安全更新存储库）。注意：添加这个软件源后才能找到 libjasper-dev 这个包依赖。

```
    sudo apt update
    sudo apt install  -y libjasper-dev
    sudo apt install -y freeglut3-dev
    sudo apt install -y libgl1-mesa-dev
    sudo apt install -y pkg-config
```

2. gstreamer 相关依赖

gstreamer 相关的依赖如下：

```
    sudo apt update
    sudo apt install -y libgstreamer1.0-dev libgstreamer-plugins-base1.0-dev
libgstreamer-plugins-bad1.0-dev libgstreamer-plugins-good1.0-dev
    sudo apt install -y gstreamer1.0-plugins-base gstreamer1.0-plugins-good
gstreamer1.0-plugins-bad gstreamer1.0-plugins-ugly
    sudo apt install -y gstreamer1.0-tools gstreamer1.0-libav
    sudo apt install -y gstreamer1.0-doc gstreamer1.0-x gstreamer1.0-alsa
gstreamer1.0-gl gstreamer1.0-gtk3 gstreamer1.0-qt5 gstreamer1.0-pulseaudio
```

此处依赖是需要用到 gstreamer 相关依赖才进行下载的，如果无须用到，可以不下载此处依赖，

后续 cmake 命令中的参数也需要进行相应的调整。

3. GTK 支持（图形界面库）

安装 gtk2 支持：

```
sudo apt-get install libgtk2.0-dev
```

安装 gtk3 支持：

```
sudo apt-get install libgtk-3-dev
```

2.2.2 编译 OpenCV 源码

把编译 OpenCV 之前需要准备的依赖全部安装完成后，就可以正式开始编译 OpenCV 了。官网下载 OpenCV 源码或者把配套资源 somesofts 目录（后文提到此目录都是指配套资源）下的 OpenCV 源码文件 4.10.0.zip 上传到 Linux 的某个文件夹下，比如/root/soft/，然后进入目录/root/soft 进行解压：

```
unzip 4.10.0.zip
```

那我们的 OpenCV 源码路径就是/root/soft/opencv-4.10.0，这个路径后面要用到。进入/root/soft/opencv-4.10.0，在该路径下新建一个文件夹“.cache”，并把 somesofts\cmake 需要的\Ubuntu\.cache\下的两个子文件夹上传到/root/soft/opencv-4.10.0/.cache，这个.cache 目录会存放 CMake 下载的文件。然而，由于下载太慢，导致无法上传成功，因此笔者采用其他方式把所需的文件全部下载好并放到.cache 下，这样 CMake 一看已经存在需要下载的文件了，就不会再去下载，也就不会卡死了。

另外，我们还需要建立一个空文件夹，用于存放 OpenCV 编译后的文件，这里在/root/soft 下新建一个 opencvBuild 文件夹，命令如下：

```
mkdir -p /root/soft/openvBuild
```

因为编译大型源码需要 Makefile，下面我们准备在 CMake 下生成 Makefile 文件。这里和在 Windows 下一样，也使用 CMake GUI 图形软件，因此我们要在 Ubuntu 图形界面中进行。安装 CMake GUI 软件的命令如下：

```
apt-get install cmake-gui
```

安装完毕后，在终端窗口上输入如下命令启动 CMake GUI：

```
cmake-gui
```

然后在 CMake 窗口上的 Where is the source code 旁输入“/root/soft/opencv-4.10.0”，这个路径就是 OpenCV 源码所在的路径；在 Where to build the binaries 旁输入“/root/soft/openvBuild”，这是编译后存放文件的目录。然后单击 Configure 按钮，在出现的对话框上保持默认选择，如图 2-28 所示。

单击 Finish 按钮后，CMake 开始工作。稍等片刻，工作完成，如果出现很多红色条目，那就再单击 Configure 按钮，第二次结束时就不会有很多红色条目了。我们在 Search 搜索框中输入 IPP，

去掉 WITH_IPP 和 BUILD_IPP_IW 的勾选。如果最终要生成静态库（.a），那么还要去掉 BUILD_SHARED_LIBS 的勾选。这里我们准备生成动态（共享）库（.so），因此保持 BUILD_SHARED_LIBS 处于勾选状态（默认）。另外，如果我们要指定编译后的安装（make install 文件夹，则可以通过选项 CMAKE_INSTALL_PREFIX 来设置指定目录，默认是/usr/local。

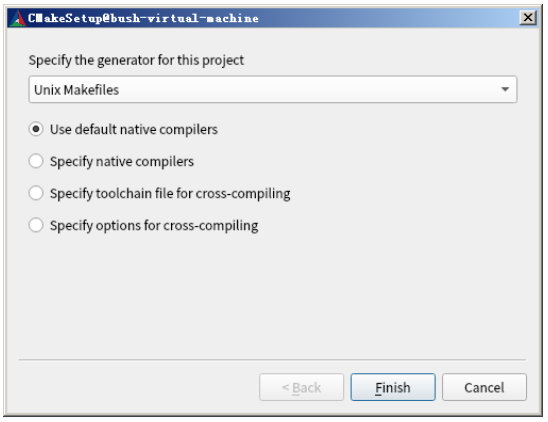

图 2-28

再次单击 Configure 按钮，CMake 配置完成后的界面如图 2-29 所示。

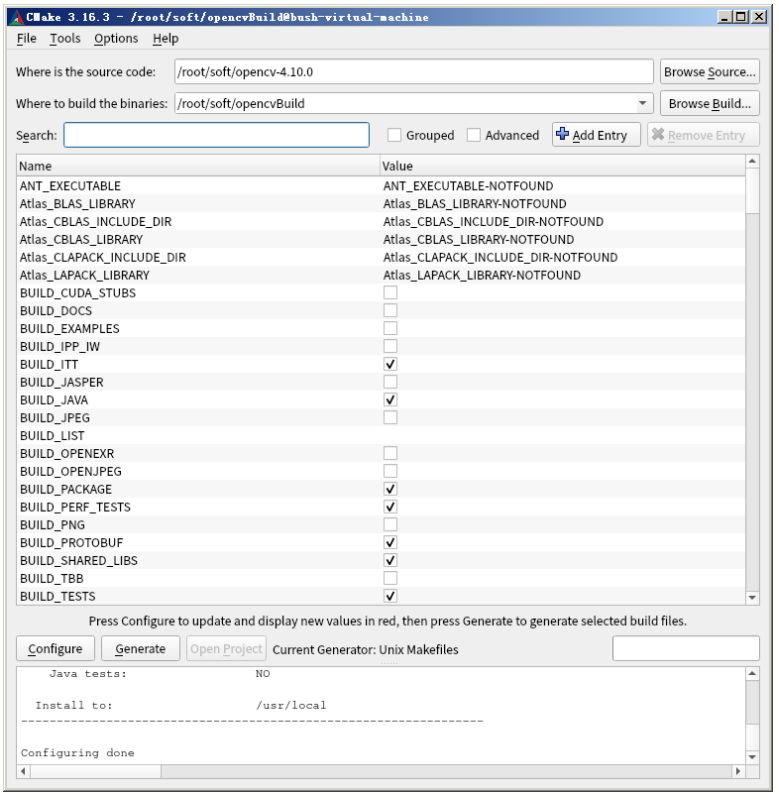

图 2-29

CMake 的配置工作完成后，就可以单击 Generate 按钮来生成 Makefile 文件了，这一步一般不

会出错。成功后，我们可以在 opencvBuild 目录下看到 Makefile 文件，下面就可以开始编译了。

在终端窗口下进入 opencvBuild 目录，然后输入 make 命令开始漫长的编译。因为我们是在虚拟机中编译的，所以编译过程注定是漫长的。

另外细心的读者可能会有疑问，Windows 下用到了“-j”这个选项，也就是使用了多线程编译，为何现在仅用一个 make 了？其实，笔者一开始的确是用“make -j8”命令来进行多线程编译的，但是在编译过程中发现，前期编译速度很快，但是编译到大概 70%的时候，速度就越来越慢，一个上午的时间只编译了 5%，后来就一直卡在 93%。笔者觉得有些奇怪，就打开 htop 查看了进程信息，发现编译的进程根本没有运行，CPU 的占用率也不太正确。进程的状态是 D，表示进程正处于不可中断的等待状态。估计是编译器的堆反应空间不足了，OpenCV 编译被卡住了。

笔者只好重新编译，但是这次没有使用多线程进行编译，只用了 make 命令，果然正常了。因此，这里直接推荐读者用 make 来编译，虽然或许会慢一点，但至少不会卡死。也可以第一次执行 make -j8，然后卡死，再执行 make，第二次的 make 会在第一次卡死（也就是没有编译成功）的地方继续，这样或许总时间反而比一开始就用 make 来得快些。以上都是经验和教训之谈。

最后编译结束，如下所示：

```
    ...
     [ 99%] Building CXX object
modules/gapi/CMakeFiles/opencv_test_gapi.dir/test/util/variant_tests.cpp.o
    [ 99%] Linking CXX executable ../../bin/opencv_test_gapi
    [ 99%] Built target opencv_test_gapi
    [ 99%] Built target opencv_annotation
    [100%] Built target opencv_visualisation
    [100%] Built target opencv_interactive-calibration
    [100%] Built target opencv_version
    [100%] Built target opencv_model_diagnostics
    root@bush-virtual-machine:~/soft/opencvBuild#
```

编译完成之后，输入如下命令进行安装：

```
    make install
```

稍等片刻，安装结束，如下所示：

```
    ...
    -- Installing:
/usr/local/share/opencv4/lbpcascades/lbpcascade_frontalface.xml
    -- Installing:
/usr/local/share/opencv4/lbpcascades/lbpcascade_frontalface_improved.xml
    -- Installing:
/usr/local/share/opencv4/lbpcascades/lbpcascade_profileface.xml
    -- Installing:
/usr/local/share/opencv4/lbpcascades/lbpcascade_silverware.xml
    -- Installing: /usr/local/bin/opencv_annotation
    -- Set runtime path of "/usr/local/bin/opencv_annotation" to "/usr/local/lib"
    -- Installing: /usr/local/bin/opencv_visualisation
    -- Set runtime path of "/usr/local/bin/opencv_visualisation" to
"/usr/local/lib"
    -- Installing: /usr/local/bin/opencv_interactive-calibration
```

```
    -- Set runtime path of "/usr/local/bin/opencv_interactive-calibration" to
"/usr/local/lib"
    -- Installing: /usr/local/bin/opencv_version
    -- Set runtime path of "/usr/local/bin/opencv_version" to "/usr/local/lib"
    -- Installing: /usr/local/bin/opencv_model_diagnostics
    -- Set runtime path of "/usr/local/bin/opencv_model_diagnostics" to
"/usr/local/lib"
    root@bush-virtual-machine:~/soft/opencvBuild#
```

这里采用默认位置安装，因此会安装到如下目录：/usr/local/bin、/usr/local/lib 和/usr/local/share。例如我们可以查看版本信息：

```
    # /usr/local/bin/opencv_version
    4.10.0
```

也可以直接用命令查看版本信息：

```
    # opencv_version
    4.10.0
```

/usr/local/lib 存放的是编程所需的库文件，主要是以 libopencv 开头的 so 文件。编程所需的 OpenCV 头文件则在/usr/local/include/opencv4/opencv2/下。

2.2.3　Linux 下的第一个 OpenCV 程序

准备工作终于完成了，我们可以开始在 Linux 下编写 OpenCV 程序了。

【例 2.3】Linux 下的第一个 OpenCV 程序

步骤 01　在 Linux 下的某个路径（这里是/root/ex/）下新建一个文件夹 opencvTest，然后在该文件夹下新建一个源文件 helloworld.cpp：

```
    gedit helloworld.cpp
```

在 helloworld.cpp 文件中输入如下代码：

```
    #include <iostream>
    #include <opencv2/opencv.hpp>
    using namespace std;
    using namespace cv;
    int main(int argc, char* argv[])
    {
           Mat img;
           string imgpath = "p1.jpg";   // 定义图片文件的路径
           img = imread(imgpath, 1);    // 读取图片文件
           if (img.data == NULL)        // 或 img.empty() // 判断图像数据是否为空
             puts("load failed");       // 提示用户加载图片文件失败
           else imshow("img", img);     // 显示图片
           waitKey(0);                  // 等待用户按键
           return 0;                    // 程序返回
    }
```

代码逻辑很简单，就是去读取当前目录下的 p1.jpg，然后将其显示出来。其中，Mat 即 Matrix（矩阵）的缩写，是 OpenCV 最基本的数据结构。Mat 数据结构主要包含两部分：Header 和 Pointer。Header 中主要包含矩阵的大小、存储方式、存储地址等信息；Pointer 中存储指向像素值的指针。总之，Mat 可以在内存中存储图像文件数据。imread 用于读取图像文件；imshow 用于显示图像。这些函数后续还会详述，现在只需了解即可，因为现在的目的是要验证 Linux 下 OpenCV 开发环境是否正常。

步骤02 保存 helloworld.cpp 文件，用 g++编译并运行。在终端窗口上输入如下命令：

```
    g++ helloworld.cpp -std=c++11  -I /usr/local/include/opencv4/ -L
/usr/local/lib -lopencv_highgui  -lopencv_imgcodecs   -lopencv_imgproc
-lopencv_core -o hello
```

在命令中，先用-I 指定了头文件所在路径，由于在程序的 include 中已经写了 opencv2，因此这里只需包含到 opencv4 即可，而 opencv2 就在 opencv4 下；再用-L 包含库文件的路径，注意-I 和-L 后面都有一个空格；接着用-l 指定要链接的动态库，一共 4 个；最后用-o 指定生成的可执行文件的文件名为 hello。

命令运行后，会在当前目录下生成 hello 这个可执行程序。我们把 p1.jpg 放到 hello 同一路径下，然后运行 hello：

```
    ./hello
```

运行结果如图 2-30 所示。

图 2-30

步骤03 除了用 g++编译程序外，经验丰富的开发人员更喜欢用 CMake/Make 方式，也就是先通过 CMake 生成 Makefile，再用 Make 生成可执行文件。下面也来演示一下，在 opencvTest 文件夹下新建文本文件 CMakeLists.txt，然后输入如下代码：

```
    01 cmake_minimum_required(VERSION 2.8)
    02 project( opencvTest )
    03 find_package( OpenCV REQUIRED )
    04 include_directories( ${OpenCV_INCLUDE_DIRS} )
    05 add_executable( opencvTest helloworld.cpp )
    06 target_link_libraries( opencvTest ${OpenCV_LIBS} )
```

这是很简单的 CMakeList 代码，具体解释如下：

第 01 行代码的意思是要求 CMake 最低版本为 2.8。

第 02 行代码指定本项目的工程名为 opencvTest。

第 03 行代码引入外部依赖包，一般格式是 find_package(xxx REQUIRED)。例如，在编译某个工程时，如果需要使用 OpenCV，则在 CMakeLists.txt 中需要添加 find_package(OpenCV REQUIRED)。其原理是寻找 OpenCVConfig.cmake 文件，该文件中指定了 OpenCV 的库路径和头文件路径，从而使得编译时能够找到相应的头文件和库文件。如果不进行任何调整，find_package(OpenCV REQUIRED)会在默认路径中查找 OpenCVConfig.cmake。然而，如果系统默认路径中的 OpenCV 版本不是我们所需的，我们在某个路径下自行编译并安装了更高版本的 OpenCV，然后在编译工程时希望 CMake 能够找到我们自己编译的 OpenCV 版本的 OpenCVConfig.cmake。此时，可以使用 set(CMAKE_PREFIX_PATH xxx) 来指定搜索路径。由于本例我们将 OpenCV 的编译结果安装到了默认目录，在默认路径/usr/local/lib/cmake/opencv4/下可以找到 OpenCVConfig.cmake，因此本例不需要使用 set(CMAKE_PREFIX_PATH xxx)。

使用 find_package 找到.cmake 或.pc 文件，找到后相关的头文件和库文件路径会分别保存在指定变量中，如 XXX_INCLUDE_DIRS、XXX_LIBRARIES，此时再通过 include_directories 或 link_directories 一键引入所有的库和头文件路径。这里，我们通过第 3 行代码把头文件路径存放在 OpenCV_INCLUDE_DIRS 中。

第 04 行代码添加头文件路径到编译器的头文件搜索路径下，多个路径以空格分隔。

第 05 行代码中的第一个参数表示要生成的可执行文件名为 openTest，后面的参数是需要的源码文件。

保存 CMakeLists.txt 文件然后在终端窗口下执行：

```
  # cmake .
-- The C compiler identification is GNU 9.4.0
-- The CXX compiler identification is GNU 9.4.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Found OpenCV: /usr/local (found version "4.10.0")
-- Configuring done
-- Generating done
-- Build files have been written to: /root/ex/opencvTest
```

注意 cmake 后面有一个空格和黑点。文件运行后会在当前目录下生成 Makefile，然后我们就可以使用 make 命令来生成可执行文件 opencvTest 了，命令如下：

```
# make
Scanning dependencies of target opencvTest
[ 50%] Building CXX object CMakeFiles/opencvTest.dir/helloworld.cpp.o
```

```
[100%] Linking CXX executable opencvTest
[100%] Built target opencvTest
```

最后运行可执行文件 opencvTest：

```
# ./opencvTest
```

运行结果也是显示图片 p1.jpg，这里就不截图了。

至此，Linux 下的简陋的 OpenCV 开发环境就建立起来了。为何说简陋呢？因为目前没有用到 IDE。下面就来使用 Qt Creator。

2.2.4 下载 Qt

这里使用 Qt6，在 Linux 下笔者一般喜欢用较新的版本，因为无论操作系统还是应用软件，都是免费的。同时使用 Qt6 也是为了更好地测试 OpenCV。Windows 用 Qt5 来测试 OpenCV，Linux 用 Qt6 来测试 OpenCV，这样我们就可以知道 OpenCV 是否能和不同的 Qt 版本兼容。

我们可以到以下地址下载 Linux 环境下的 Qt 在线安装包：

```
https:// download.qt.io/archive/online_installers/4.3/
```

进入网页后单击 qt-unified-linux-x64-4.3.0-1-online.run，然后开始下载。如果网址失效或不想下载，也可以直接到源码目录的子文件夹 somesofts 下找到 Qt 的安装包。

如果下载下来的文件存放到 Windows 中，下一步还需要把它存放到虚拟机 Ubuntu 中。在虚拟机的桌面上，单击左边工具栏中的第三个按钮来打开文件资源管理器，如图 2-31 所示。

在新出现的文件资源管理器窗口中单击左边的“主目录”，然后单击右上角查询图标旁的排列图标，如图 2-32 所示。这样操作主要是可以在窗口中留出更多空白区域以方便我们新建文件夹。

在空白处新建文件夹，并将这个文件夹命名为 soft（如果 soft 文件夹已经建立过了，则不需要再建立），然后用鼠标双击它进入这个文件夹，再把前面下载到 Windows 下的 qt-unified-linux-x64-4.3.0-1-online.run 文件复制并粘贴到该文件夹中，或者直接拖进去。注意，粘贴进度条即使消失了，也要稍等一会儿才能真正完成粘贴操作。如果操作一切正常，选中该文件后，它的右下角会显示文件的大小（36.6MB），如图 2-33 所示。

图 2-31

图 2-32

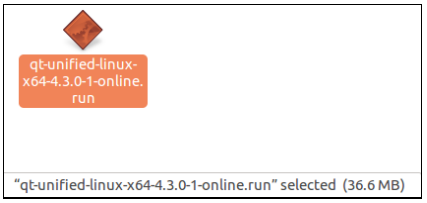

图 2-33

这样，下载下来的 Qt 安装包就复制粘贴到虚拟机 Ubuntu 中了。

2.2.5 安装依赖包

在安装 Qt 前，我们先要联网安装一些依赖性软件包，以免安装过程中提示找不到依赖包。在

终端窗口中依次输入如下命令：

```
apt-get install build-essential
apt-get install g++
apt-get install libx11-dev libxext-dev libxtst-dev
apt-get install libxcb-xinerama0
apt-get install libgl1-mesa-dev
apt-get install libxcb-cursor0
```

每个命令输入后都会自动安装对应的依赖包，如果安装过程中有询问，就采用默认值；如果有些包已经装过了，则会提示该包当前已是最新。倒数第二个命令是安装 OpenGL 核心库，最后一个是安装 xcb-cursor0 库（最重要的），这是 Qt6.5 必须安装的，否则 Qt Creator 启动不了。

2.2.6　安装 Qt

依赖包安装完毕后，就可以开始安装 Qt 了。在 Ubuntu 下打开终端窗口，进入/root/soft 目录，为文件 qt-unified-linux-x64-4.3.0-1-online.run 添加执行权限：

```
chmod +x qt-unified-linux-x64-4.3.0-1-online.run
```

添加执行权限后，就可以开始安装 Qt 6 了。继续运行命令：

```
./qt-unified-linux-x64-4.3.0-1-online.run
```

然后就会出现安装向导窗口，和 Windows 下安装一样，必须输入有效的账号和密码才能进行下一步操作。如果没有账号，可以注册一个。

输入账号和密码后，单击“下一步”按钮，在出现的对话框界面上勾选“我已阅读并同意使用开源 Qt 的条款和条件”和“我是个人用户，我不为任何公司使用 Qt”两行文字前的复选框，然后单击“下一步”按钮。其实这个过程跟 Windows 下的类似。持续单击“下一步”按钮，进入“安装文件夹”界面，保持默认设置，安装到/opt/Qt6 路径下，并在下方选中 Qt6.7 for desktop development，如图 2-34 所示。

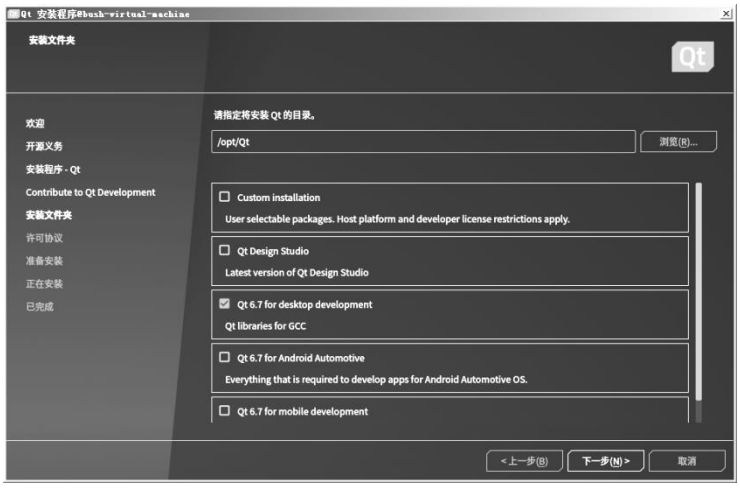

图 2-34

再单击“下一步”按钮，出现“许可协议”对话框，在下方打勾。再单击“下一步”按钮，出现“准备安装”对话框，这个时候就有可能出现状况了，如图 2-35 所示。

图 2-35

这是因为磁盘空间不够了。当然，不是每个人都会出现这个情况，这里是因为笔者的磁盘上原有内容比较多，而且安装虚拟机 Linux 的时候使用了默认的 20GB 硬盘空间，所以现在就出现这个提示了。如果没有这个提示，可以单击“安装”按钮正式开始安装。如果出现这个提示，单击“取消”按钮来关闭安装向导，然后跟着笔者扩充容量。基本思路是为虚拟机 Linux 添加一个磁盘，并把 Qt 安装到这个磁盘上。步骤如下：

步骤 01 添加新硬盘。先关闭 Ubuntu，然后在 VMware 中单击“编辑虚拟机设置”，如图 2-36 所示。

图 2-36

在“虚拟机设置”对话框的“硬件列表”里选中“硬盘（SCSI）”，并单击下方的“添加”按钮，此时出现“添加硬件向导”对话框，我们直接单击“下一步”按钮，随后几个步骤一直保持默认设置，直接单击“下一步”按钮即可。笔者这里添加的新磁盘的容量是 20GB，如图 2-37 所示。

图 2-37

添加完成后，在“虚拟机设置”的硬件列表中就可以看到多了一个新硬盘了，如图 2-38 所示。

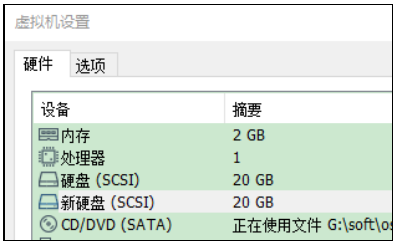

图 2-38

这样我们的虚拟机 Linux 一共有 40GB 的空间了。

步骤 02　开启虚拟机 Ubuntu，打开终端窗口，输入命令 fdisk -l。这个命令可以查看当前硬盘空间的信息。我们可以找到新添加的磁盘 sdb，如图 2-39 所示。

```
Disk /dev/sdb: 20 GiB, 21474836480 字节, 41943040 个扇区
Disk model: VMware Virtual S
单元：扇区 / 1 * 512 = 512 字节
扇区大小(逻辑/物理)：512 字节 / 512 字节
I/O 大小(最小/最佳)：512 字节 / 512 字节
```

图 2-39

然后对其进行格式化。这个概念和 Windows 一样，新加的硬盘肯定要建立起文件系统才能使用，这里我们用 mkfs.ext3 命令把硬盘格式化成 ext3 文件系统，输入命令如下：

```
mkfs.ext3 /dev/sdb
```

稍等片刻，格式化完成，如图 2-40 所示。

```
root@tom-virtual-machine:~/桌面#  mkfs.ext3 /dev/sdb
mke2fs 1.45.5 (07-Jan-2020)
创建含有 5242880 个块（每块 4k）和 1310720 个inode的文件系统
文件系统UUID：b5156368-ba21-4dc1-8e24-57ccc160eb6c
超级块的备份存储于下列块：
        32768, 98304, 163840, 229376, 294912, 819200, 884736,
        4096000

正在分配组表：  完成
正在写入inode表：  完成
创建日志（32768 个块）  完成
写入超级块和文件系统账户统计信息：  已完成

root@tom-virtual-machine:~/桌面#
```

图 2-40

步骤 03　格式化完成后，我们再将这个磁盘挂载到一个文件夹。这个文件夹必须先建立，输入命令如下：

```
mkdir /wwwroot
```

名字可以自己定义。然后我们将磁盘挂载到文件夹/wwwroot 下，输入命令如下：

```
mount /dev/sdb /wwwroot
```

如果没有提示信息，说明挂载成功了，如图 2-41 所示。

```
root@tom-virtual-machine:~/桌面# mount /dev/sdb /wwwroot/
root@tom-virtual-machine:~/桌面#
```

图 2-41

现在往/wwwroot 下存放数据，就相当于存放到/dev/sdb 下了。

步骤 04 我们启动 Qt 安装程序（命令行下执行/root/soft/qt-unified-linux-x64-4.3.0-1-online.run），然后在 Qt 安装向导的“安装文件夹”对话框上，将安装路径指定到/wwwroot/Qt6，如图 2-42 所示。

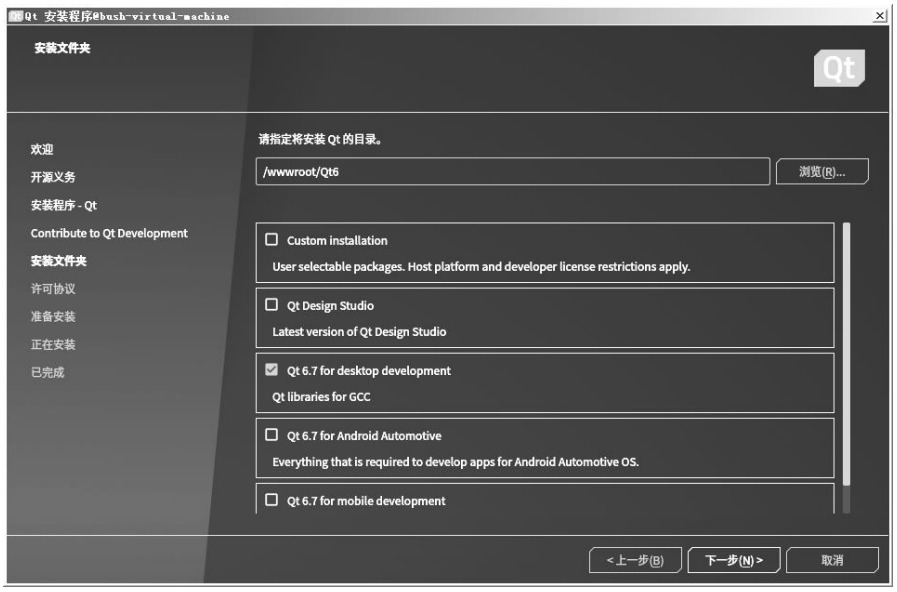

图 2-42

此时单击“下一步”按钮，就没有提示磁盘空间不够的信息了，直接提示已经准备安装了，如图 2-43 所示。

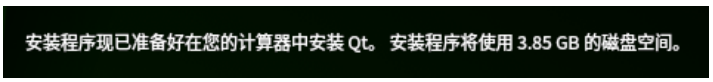

图 2-43

步骤 05 继续单击右下方的“安装”按钮，开始正式安装，这个过程比较漫长，同时要保持网络在线，因为现在都是在线安装了，如图 2-44 所示。

图 2-44

安装完成后，给出提示，如图 2-45 所示。

图 2-45

至此，Linux 下安装 Qt6 完成，我们可以在/wwwroot 文件夹下看到 Qt6 子文件夹了，并且 qtcreator 可执行程序位于./Tools/QtCreator/bin/下。下面启动 qtcreator，输入如下命令：

```
/wwwroot/Qt6/Tools/QtCreator/bin/qtcreator
```

这个命令可以在任意路径下启动 qtcreator，因为我们带上了全路径。如果先进入/wwwroot/Qt6/Tools/QtCreator/bin/下，则只需执行./qtcreator 即可，其中./表示在当前路径下执行。第一次启动比较慢，稍等一会儿，qtcreator 就启动了，如图 2-46 所示。

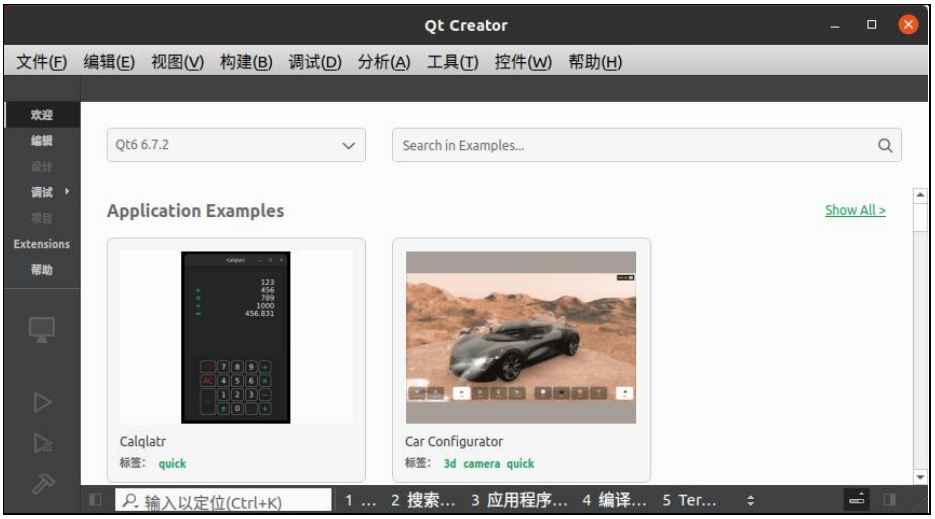

图 2-46

2.2.7　Linux 下用 Qt 开发 OpenCV

相信到这会儿读者已经迫不及待地想要新建项目文件。前面已经介绍过新建项目的过程，这里不再赘述。这就是跨平台软件的好处，一旦学会如何在 Windows 下使用，那么到了 Linux 下，用起来也基本一样。注意：每次重启后，要先把 sdb 加载到/wwwroot 下，命令如下：

```
mount /dev/sdb /wwwroot
```

再启动 qtcreator。下面来看一个例子。

【例 2.4】Linux 下用 Qt 开发的 OpenCV 程序

步骤 01　在虚拟机 Ubuntu 的终端窗口中输入如下命令来启动 Qt Creator：

```
/wwwroot/Qt6/Tools/QtCreator/bin/qtcreator
```

步骤 02　在 Qt Creator 主界面上，依次单击主菜单中的“文件→New Project”选项，随后弹出 New Project—Qt Creator 对话框，在该对话框上选中 Qt Console Application 选项，如图 2-47 所示。

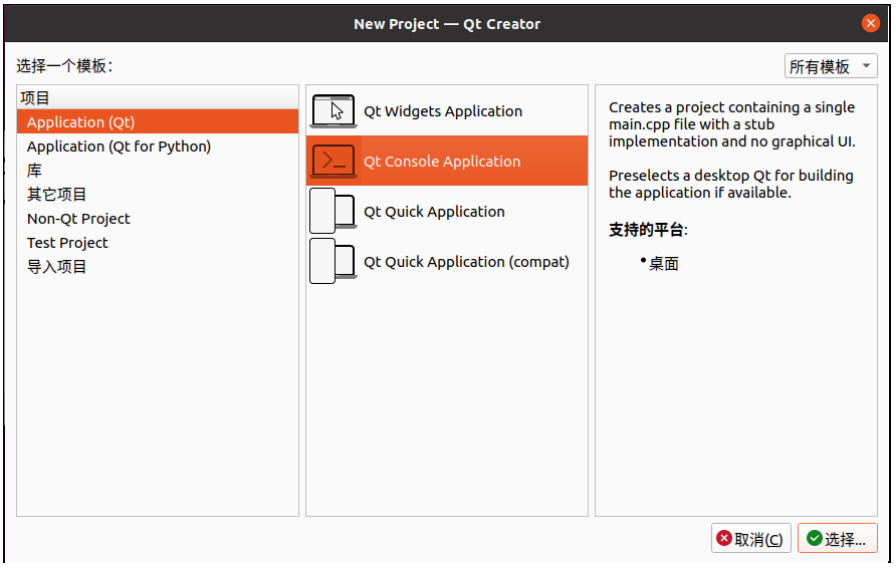

图 2-47

步骤 03 单击右下角的“选择”按钮，出现 Project Location 设置界面，在该界面中输入名称 test，路径随便输，但要确保路径已经存在，这里输入/root/ex，文件夹 ex 已经预先建立好，如图 2-48 所示。

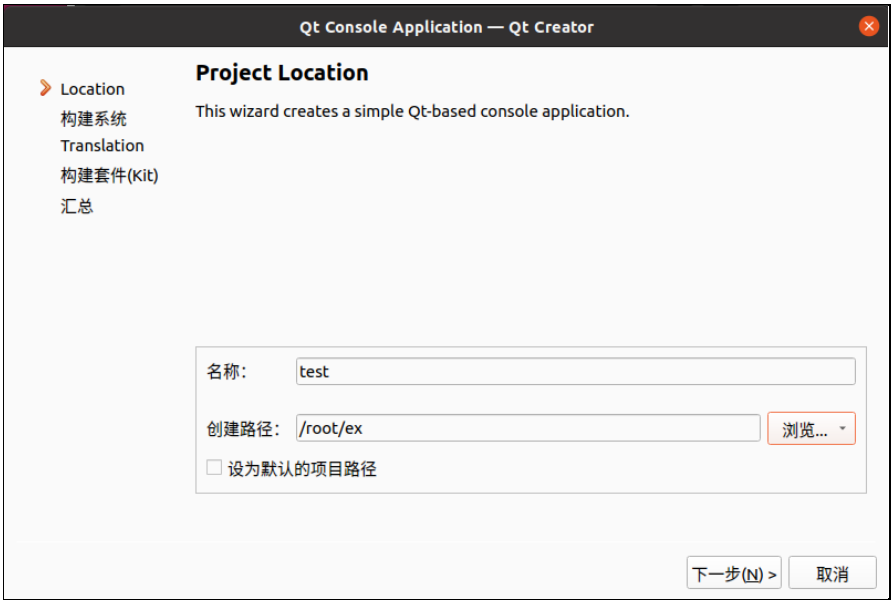

图 2-48

步骤 04 继续单击“下一步”按钮，出现 Define Build System 设置界面，我们选择 Build system（构建系统）为“qmake”，如图 2-49 所示。

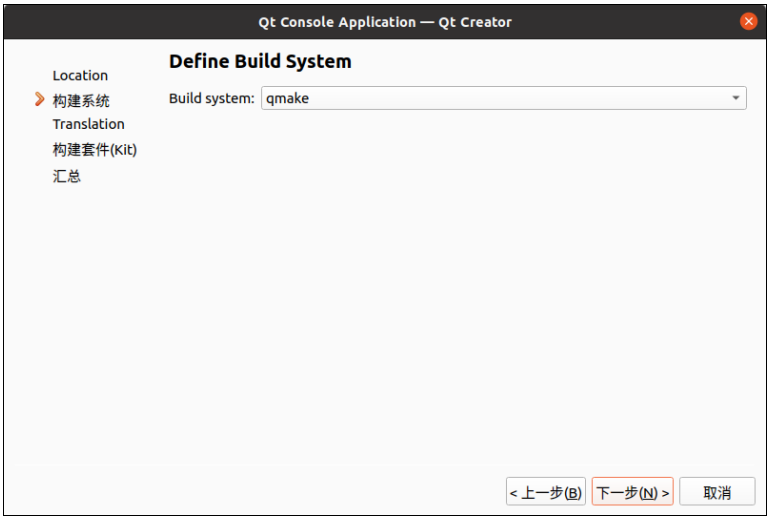

图 2-49

qmake 是 Qt 工具包自带的一个非常方便的工具，可以用于生成 Makefile 以及各种工程文件，还可以生成 Microsoft Visual Studio 可以使用的项目文件等。最关键的是它可以自动解决依赖关系，不用手写 Makefile，而且是跨平台的。

步骤 05　接着连续单击“下一步”按钮直到结束，这个过程一直保持默认设置即可。

步骤 06　这样一个 MainWindow 程序框架就建立起来了。此时会自动打开 main.cpp 文件，并且该文件中已经有一些自动生成的代码了。我们把很多注释代码删除，然后在 return 语句前添加一行代码：

```
    puts("Hello,welecom to opencv in Linux!");
```

步骤 07　再单击左侧竖条工具栏上的“项目”，然后单击“运行”，再在右边勾选“在终端中运行”复选框，如图 2-50 所示。

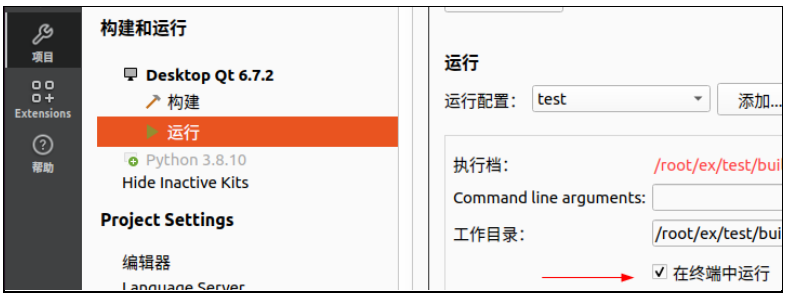

图 2-50

步骤 08　单击左侧竖条工具栏上的“编辑”，切换到编辑窗口界面，然后单击竖条工具栏下方的三角按钮，这个就是运行程序的按钮，或直接按快捷键 Ctrl+R 来运行程序，运行结果如图 2-51 所示。

图 2-51

可以看到，终端（Terminal）窗口中输出字符串了。这就说明 Qt 环境部署成功了。

步骤 09 下面添加 OpenCV 代码。此时可以在/root/ex/test/build/Desktop_Qt_6_7_2-Debug/下看到可执行文件 test。我们在 main.cpp 中输入如下代码：

```
#include <iostream>
#include <opencv2/opencv.hpp>
using namespace std;
using namespace cv;
int main(int argc, char* argv[])
{
    Mat img; // 定义矩阵结果变量 img
    string imgpath = "plane.jpg"; // 文件 plane.jpg 要放到可执行文件同一目录下
    img = imread(imgpath, 1); // 读取图像文件
    if (img.data == NULL) puts("load failed"); // 如果读取失败，则提示加载失败
    else imshow("img", img); // 显示图像
    waitKey(0); // 等待用户按键
    return 0; // 程序返回
}
```

代码很简单，就是从可执行文件同一目录下读取文件 plane.jpg，然后把它显示出来。

步骤 10 配置工程文件，添加头文件和库。打开 test.pro，然后在末尾添加如下代码：

```
INCLUDEPATH += /usr/local/include/opencv4/
LIBS += -L /usr/local/lib -lopencv_highgui  -lopencv_imgcodecs
-lopencv_imgproc  -lopencv_core
```

第一行是添加头文件的所在路径，第二行是设置库路径及其要加载的动态库。

步骤 11 将图片 plane.jpg 存放到/root/ex/test/build/Desktop_Qt_6_7_2-Debug/，然后在 Qt Creator 中运行程序，运行结果如图 2-52 所示。

图 2-52

至此，我们在 Linux 下搭建 OpenCV 开发环境成功！

2.2.8 做个快照

前面我们把 Linux 下的 Qt 环境搭建起来了，为了保存劳动成果，可以用 VMware 软件做个快照。一旦系统出现故障，就可以使用快照恢复到 Qt 安装配置成功的状态。建议养成这样一个良好

的习惯，即做两个快照，第一个是刚刚安装好操作系统的时候的快照，第二个是开发环境部署成功的时候的快照。

2.3　数学函数

OpenCV 定义了一些新数学函数，都在 fast_math.hpp 头文件中。常用的数学函数如下：

```
    CV_INLINE int cvRound( double value )  // 返回跟参数最接近的整数值，四舍五入函数
    CV_INLINE int cvFloor( double x )      // 近似一个浮点数 x 到不大于 x 的最近的整数，
即向下取整
    CV_INLINE int cvCeil( double x)        // 近似一个浮点数 x 到不小于 x 的最近的整数，
即向上取整
    CV_INLINE int cvIsNaN( double value )  // 判断是不是一个数
    CV_INLINE int cvIsInf( double value )  // 判断是否无穷大
    CV_INLINE int cvRound(float value)
    CV_INLINE int cvRound( int value )
    CV_INLINE int cvFloor( float value )
    CV_INLINE int cvFloor( int value )
    CV_INLINE int cvCeil( float value )
    CV_INLINE int cvCeil( int value )
    CV_INLINE int cvIsNaN( float value )   // 判断输入值是否为一个数字
    CV_INLINE int cvIsInf( float value )   // 判断一个数字是否为无穷大
```

虽然标准 C 函数也有类似功能的函数，但在某些场合中 OpenCV 的函数比标准 C 函数操作起来要快。

需要注意的是，OpenCV 中的 cvRound 函数满 5 并不会进位，要满 6 才会进位，所以是四舍六入的，即将 0.5 向下舍，比如 cvRound(2.5)=2，而不是 3，示例如下：

【例 2.5】实验取整的数学函数

步骤 01　打开 Qt Creator，新建一个控制台工程，工程名是 test。

步骤 02　在 IDE 中打开 main.cpp，输入如下代码：

```
#include <opencv2/opencv.hpp>
#include <iostream>
using namespace std;
using namespace cv;  // 所有 OpenCV 类都在命名空间 cv 下
int main(int argc, char *argv[])
{
    cout << "cvRound(2.5) : " << cvRound(2.5) << endl;
    cout << "cvRound(2.6) : " << cvRound(2.6) << endl;
    cout << "cvRound(2.8) : " << cvRound(2.8) << endl;

    cout << "cvFloor(2.5) : " << cvFloor(2.5) << endl;
    cout << "cvFloor(2.6) : " << cvFloor(2.6) << endl;
    cout << "cvCeil(2.5)  : " << cvCeil(2.5) << endl;
    cout << "cvCeil(2.6)  : " << cvCeil(2.6) << endl;
```

```
    return 0;
}
```

在上述代码中，我们分别实验了 cvRound、cvFloor 和 cvCeil 的简单使用。

步骤 03 在工程文件 test.pro 的末尾添加头文件路径和.a 库文件路径：

```
INCLUDEPATH += D:/opencvBuild/install/include/
LIBS += -L
D:/opencvBuild/install/x64/mingw/lib/libopencv_*.a
```

本例用到的 OpenCV 库函数因为都在.a 静态库文件中，因此不需要拷贝动态库文件到 exe 目录。

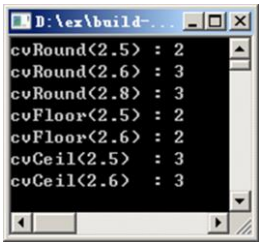

图 2-53

步骤 04 保存工程并运行，结果如图 2-53 所示。果然，cvRound(2.5)的结果依旧是 2，要满 6 才会进位，如 cvRound(2.6)=3。

2.4 OpenCV 架构

OpenCV 现在已经发展得比较庞大了，针对不同的应用，它划分了不同的模块，每个模块专注于不同的功能。一个模块下面可能有类、全局函数、枚举、全局变量等。所有全局函数或变量都在命名空间 cv 下。

我们在 D:\opencvBuild\install\include\opencv2\下可以看到 OpenCV4 不同模块所对应的文件夹，如图 2-54 所示。

图 2-54

这些模块有的经过多个版本的更新已经较为完善，包含较多的功能；有的还在逐渐发展中，包含的功能相对较少。接下来按照文件夹的顺序（字母顺序）介绍各个模块的功能。

- calib3d：这个模块名称由 calibration（校准）和 3D 这两个单词的缩写组合而成，通过名称

可以知道，模块主要包含相机标定与立体视觉等功能，例如物体位姿估计、三维重建、摄像头标定等。

- core：核心功能模块，主要包含 OpenCV 库的基础结构以及基本操作，例如 OpenCV 基本数据结构、绘图函数、数组操作相关函数、动态数据结构等。
- dnn：深度学习模块，这个模块是 OpenCV 4 版本的一个特色，主要包括构建神经网络、加载序列化网络模型等。该模块目前仅适用于正向传递计算（测试网络），原则上不支持反向计算（训练网络）。
- features2d：这个模块名称是由 features（特征）和 2D 这两个单词的缩写组合而成的，其功能主要为处理图像特征点，例如特征检测、描述与匹配等。
- flann：这个模块名称是 Fast Library for Approximate Nearest Neighbors（快速近似最近邻库）的缩写，这个模块是高维的近似近邻快速搜索算法库，主要包含快速近似最近邻搜索与聚类等。
- gapi：这个模块是 OpenCV 4.0 中新增加的模块，旨在加速常规的图像处理，与其他模块相比，这个模块主要充当框架，而不是某些特定的计算机视觉算法。
- highgui：高层 GUI 图形用户界面，包含创建和操作显示图像的窗口，处理鼠标事件及键盘命令，提供图形交互可视化界面等。
- imgcodecs：图像文件读取与保存模块，主要用于图像文件的读取与保存。
- imgproc：这个模块名称由 image（图像）和 process（处理）两个单词的缩写组合而成，是重要的图像处理模块，主要包括图像滤波、几何变换、直方图、特征检测与目标检测等。
- ml：机器学习模块，主要为统计分类、回归和数据聚类等。
- objdetect：目标检测模块，主要用于图像目标检测，例如检测 Haar 特征。
- photo：计算摄影模块，主要包含图像修复和去噪等。
- stitching：图像拼接模块，主要包含特征点寻找与匹配图像、估计旋转、自动校准、接缝估计等图像拼接过程的相关内容。
- video：视频分析模块，主要包含运动估计、背景分离、对象跟踪等视频处理相关内容。
- videoio：视频输入输出模块，主要用于读取与写入视频或者图像序列。

通过对 OpenCV 4.10 模块构架的介绍，相信读者已经对 OpenCV 4.10 整体架构有了一定的了解。其实简单来说，OpenCV 就是将众多图像处理和视觉处理工具集成在一起的软件开发包（Software Development Kit，SDK），其自身并不复杂，只要学习，都可以轻松掌握其使用方式。

这些模块刚开始没必要一下子全部掌握，可以先学习几个常用的模块，其他模块可以在实际工作需要的时候再学习。

2.5　图像输入输出模块 imgcodecs

巧妇难为无米之炊，要处理图像，第一步就是把图像文件从磁盘上读取到内存，处理完毕后再保存到内存。因此，我们先来看图像文件读取与保存模块 imgproc。imgproc 提供了一系列全局函数来读取或保存图像文件。

imgproc 模块中的函数都在 opencv2/imgcodecs.hpp 中声明。由于 opencv2\opencv.hpp 中已经包

含了 opencv2/imgcodecs.hpp，因此程序中也可以只包含 opencv2\opencv.hpp，比如：

```
#include<opencv2\opencv.hpp>
```

因为 opencv.hpp 中有如下代码：

```
#ifdef HAVE_OPENCV_IMGCODECS
#include "opencv2/imgcodecs.hpp"
#endif
```

2.5.1 imread 读取图像文件

函数 imread 用于读取图像文件（或叫作加载图像文件）。该函数声明如下：

```
Mat cv::imread (const String & filename,  int  flags = IMREAD_COLOR );
```

参数说明如下：

- filename: 表示要读取的图像文件名。
- flags: 表示读取模式，可以从枚举 cv::ImreadModes 中取值，默认值是 IMREAD_COLOR，表示始终将图像转换为三通道 BGR 彩色图像。如果从指定文件加载图像成功，就返回 Mat 矩阵；如果无法读取图像（由于缺少文件、权限不正确、格式不受支持或无效），函数就返回空矩阵（Mat::data==NULL）。

Imread 支持常见的图像格式，某些图像格式需要（免费提供的）第三方类库。在 Windows 操作系统下，OpenCV 的 imread 函数支持如下类型的图像载入：

```
JPEG 文件 - *.jpeg, *.jpg, *.jpe
JPEG 2000 文件- *.jp2
PNG 图片 - *.png
便携文件格式- *.pbm, *.pgm, *.ppm
Sun rasters 光栅文件 - *.sr, *.ras
TIFF 文件 - *.tiff, *.tif
Windows 位图- *.bmp,*.dib
```

如果读取失败，通常可以用两种方式来判断：直接判断是否为 NULL，或者调用 Mat::empty() 函数，比如：

```
Mat img;
...
// 开始判断是否加载成功
if (img.data == NULL) puts("load failed");
// if (img.empty()) puts("load failed");
```

值得注意的是，函数 imread 根据内容而不是文件扩展名来确定图像的类型，比如我们把某个 BMP 文件的后缀名改为.jpg，imread 依然能探测到这个文件是 BMP 图像文件。

另外，imread 的第一个参数一般是图像文件的绝对路径或相对路径。对于绝对路径，imread 除了不支持单右斜线（\）形式外，其他斜线形式都支持，比如双右斜线（\\）形式、双左斜线（//）形式、单左斜线（/）形式等。通常相对路径更加方便，只要把图像文件放在工程目录下即可。下

面来看一个例子。

【例 2.6】多种路径读取图像文件

步骤 01　打开 Qt Creator，新建一个控制台工程，工程名是 test。

步骤 02　在 IDE 中打开 main.cpp，输入如下代码：

```
#include <iostream>
#include <opencv2/opencv.hpp>
using namespace std;
using namespace cv;
int main(int argc, char* argv[])
{
    Mat img;
    // string imgpath = "D:\\我的图片\\p1.jpg"; // --1--双右斜线法，路径中含有中文
    // string imgpath = "D:// test// p1.jpg";  // -- 2 --双左斜线法
    // string imgpath = "D:/test/p1.jpg"; // -- 3 --单左斜线法
    // string imgpath = "D:/test// test2\\test3// test4// p1.jpg";// -- 4 --以上 3 种混合法
    string imgpath = "p1.jpg";// // -- 5 --相对路径法，放在工程目录下
    // string imgpath = argv[1];// -- 6 --命令行参数法
    img = imread(imgpath, 1);
    if (img.data == NULL) // 或 img.empty()
      puts("load failed");
    else imshow("img", img);

    waitKey(0);
    return 0;
}
```

我们对上面 6 种路径进行了测试，任何一种 Qt 都是支持的，都可以成功读取并显示图片。较常用的是相对路径方式。如果在工程中运行程序，相对路径就是把图像文件放到构建目录 build-test-Desktop_Qt_5_14_2_MinGW_64_bit-Debug 下；如果是直接双击生成的可执行程序 test.exe，则要把图像文件和 test.exe 放在同一路径下。另外，为了叙述方便，以后只说构建目录，不再把目录名称写出来，因为构建目录是可以设置的，在 Qt Creator 的左边单击“项目”，就可以在右边看到“构建目录”的设置。

注意，第 6 种命令行参数法需要打开工程属性进行设置，在 IDE 上左侧单击“项目”→“Run”，在“运行设置”页上的“Command line arguments:”的右边输入 p1.jpg，如图 2-55 所示。

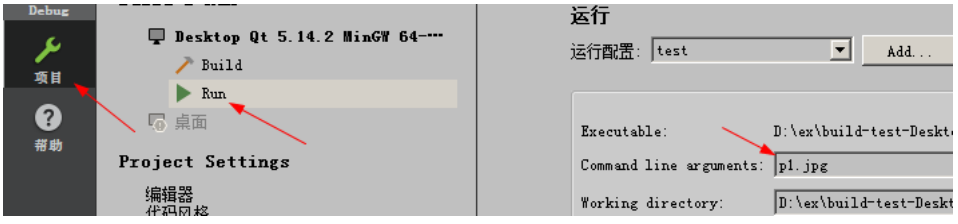

图 2-55

再回到编辑窗口，把下面两行代码中的第一行注释掉，去掉第二行的注释：

```
// string imgpath = "p1.jpg";// // -- 5 --相对路径法，放工程目录下
string imgpath = argv[1];// -- 6 --命令行参数法
```

再运行工程，此时 argv[1]的值是字符串 p1.jpg。另外，调用 imread 后，判断文件是否加载成功，比如调用 img.data == NULL 或 img.empty()。

步骤 03 在工程配置文件 test.pro 的第一行添加 QT+=widgets，末尾添加如下代码：

```
INCLUDEPATH += D:/opencvBuild/install/include/
LIBS += -L D:/opencvBuild/install/x64/mingw/lib/libopencv_*.a
```

再把目录 D:\opencvBuild\install\x64\mingw\bin\ 下的 libopencv_core4100.dll 、libopencv_highgui4100.dll、libopencv_imgcodecs4100.dll 和 libopencv_imgproc4100.dll 四个文件复制到 test.exe 文件所在的文件夹。

另外，为了叙述简洁，以后步骤 03 的这些内容不在每个范例中讲述，默认后面每个范例都会这么做，如果有变化，再单独说明。此外，后续范例都在 test 项目上实现，这样省得每次都去新建项目并做拷贝库的重复工作了。

步骤 04 保存工程并运行，结果如图 2-56 所示。

图 2-56

2.5.2 imwrite 保存图片

函数 imwrite 可以用来输出图像到文件，其声明如下：

```
bool cv::imwrite (const String & filename, InputArray img,
        const std::vector< int > & params = std::vector< int >() );
```

其中参数 filename 表示需要写入的文件名，必须加上后缀，比如 123.png，注意要保存图片为哪种格式，就带什么后缀；img 表示 Mat 类型的图像数据，就是要保存到文件中的原图像数据；params 表示为特定格式保存的参数编码，它有一个默认值 std::vector< int >()，所以一般情况下不用写。

通常，使用 imwrite 函数能保存 8 位单通道或三通道（具有 BGR 通道顺序）图像。16 位无符号（CV_16U）图像可以保存为 PNG、JPEG 2000 和 TIFF 格式。32 位浮点（CV_32F）图像可以保存为 PFM、TIFF、OpenEXR 和 Radiance HDR 格式。三通道（CV_32FC3）TIFF 图像将使用 LogLuv 高动态范围编码（每像素 4 字节）保存。另外，使用此函数可以保存带有 alpha 通道的 PNG 图像。为此，创建 8 位（或 16 位）四通道图像 BGRA，其中 alpha 通道最后到达。完全透明的像素应该

将 alpha 通道设置为 0，完全不透明的像素应该将 alpha 设置为 255/65535。如果格式、深度或通道顺序不同，就在保存之前使用 Mat::convertTo 和 cv::cvtColor 进行转换，或者使用通用文件存储 I/O 函数将图像保存为 XML 或 YAML 格式。

下面示例将演示如何创建 BGRA 图像并将其保存到 PNG 文件中，还将演示如何设置自定义压缩参数。

【例 2.7】创建 BGRA 图像并将其保存到 PNG 文件

步骤 01　打开 Qt Creator，新建一个控制台工程，工程名是 test。

步骤 02　在 IDE 中打开 main.cpp，输入如下代码：

```
    #include "pch.h"
    #include <iostream>
    #include <opencv2/opencv.hpp>
    using namespace cv;
    using namespace std;
    static void createAlphaMat(Mat &mat)
    {
        CV_Assert(mat.channels() == 4);
        for (int i = 0; i < mat.rows; ++i)
        {
            for (int j = 0; j < mat.cols; ++j)
            {
                Vec4b& bgra = mat.at<Vec4b>(i, j);
                bgra[0] = UCHAR_MAX;    // 蓝色
                bgra[1] = saturate_cast<uchar>((float(mat.cols - j)) /
((float)mat.cols) * UCHAR_MAX);         // 绿色
                bgra[2] = saturate_cast<uchar>((float(mat.rows - i)) /
((float)mat.rows) * UCHAR_MAX);         // 红色
                bgra[3] = saturate_cast<uchar>(0.5 * (bgra[1] + bgra[2])); // Alpha
            }
        }
    }
    int main()
    {
        // Create mat with alpha channel
        Mat mat(480, 640, CV_8UC4);
        createAlphaMat(mat);
        vector<int> compression_params;
        compression_params.push_back(IMWRITE_PNG_COMPRESSION);
        compression_params.push_back(9);
        bool result = false;
        try
        {
            result = imwrite("alpha.png", mat, compression_params);
        }
        catch (const cv::Exception& ex)
        {
            fprintf(stderr, "Exception converting image to PNG format: %s\n",
ex.what());
```

```
        }
        if (result)
            printf("Saved PNG file with alpha data.\n");
        else
            printf("ERROR: Can't save PNG file.\n");
        return result ? 0 : 1;
    }
```

在上述代码中，createAlphaMat 是一个自定义函数，用于使用 alpha 通道创建材质；compression_params 用于存放压缩参数；函数 imwrite 把图像矩阵以 compression_params 压缩参数保存到工程目录下的 alpha.png 中，文件 alpha.png 会自动创建。

步骤 03 保存工程并运行，结果如下：

```
Saved PNG file with alpha data.
```

此时可以在构建目录下发现有一个 640×480 的 alpha.png 文件。

2.6 OpenCV 界面编程

OpenCV 支持有限的界面编程，主要针对窗口、控件和鼠标事件等，比如滑块。有了这些窗口和控件，就可以更方便地展现图像和调节图像的参数。这些界面编程主要由 High-level GUI（高级图形用户界面）模块支持。

2.6.1 新建窗口并显示

在 High-Level GUI 模块中，用于新建窗口的函数是 nameWindow，同时可以指定窗口的类型。该函数声明如下：

```
void namedWindow(const string& winname, int flags);
```

参数说明如下：

- winname：表示新建的窗口的名称，自己随便取。
- Flags：表示窗口的标识，一般默认为 WINDOW_AUTOSIZE，表示窗口大小自动适应图片大小，并且不可手动更改；WINDOW_NORMAL 表示用户可以改变这个窗口的大小；WINDOW_OPENGL 表示窗口创建的时候支持 OpenGL。

在 High-Level GUI 模块中，用于显示窗口的函数是 imshow，该函数声明如下：

```
void imshow(const string& winname, InputArray image);
```

参数说明如下：

- winname：表示显示的窗口名，可以使用 cv::namedWindow 函数创建窗口，若不创建，则 imshow 函数将自动创建。

- image：表示窗口中需要显示的图像。

根据图像的深度，imshow 函数会自动对其显示灰度值进行缩放，规则如下：

（1）如果图像数据类型是 8U（8 位无符号整数），就直接显示。

（2）如果图像数据类型是 16U（16 位无符号整数）或 32S（32 位有符号整数），imshow 函数内部就会自动将每个像素值除以 256 并显示，即将原图像素值的范围由[0~255×256]映射到[0~255]。

（3）如果图像数据类型是 32F（32 位浮点数）或 64F（64 位浮点数），imshow 函数内部就会自动将每个像素值乘以 255 并显示，即将原图像素值的范围由[0~1]映射到[0~255]（注意：原图像素值必须归一化）。

需要注意，imshow 之后必须有 waitKey 函数，否则显示窗口将一闪而过，不会驻留屏幕。waitKey 函数的详细说明将在 2.6.6 节中介绍。

【例 2.8】新建窗口并显示 5 秒后退出

步骤 01　打开 Qt Creator，新建一个控制台工程，工程名是 test。

步骤 02　在 IDE 中打开 main.cpp，输入如下代码：

```
#include <opencv2/opencv.hpp>
using namespace cv;
int main()
{
    Mat img;
    img = imread("test.jpg",1);// 参数 1：图片路径。参数 2:显示原图
    namedWindow("窗口 1", WINDOW_AUTOSIZE);
    imshow("窗口 1",img);          // 在“窗口 1”这个窗口输出图片
    waitKey(5000);                 // 等待 5 秒，程序自动退出。改为 0，不自动退出
    return 0;
}
```

在上述代码中，首先利用函数 imread 读取当前目录下的 test.jpg 文件；接着用函数 namedWindow 新建了一个窗口，使用参数 WINDOW_AUTOSIZE，表示窗口大小自动适应图片大小，并且不可手动更改；最后调用 waitKey 函数等待 5 秒后程序自动退出。

步骤 03　保存工程并运行，结果如图 2-57 所示。

图 2-57

2.6.2 单窗口显示多幅图像

前面新建的窗口中只显示了一幅图，但在某些场景下，比如有多个视频图像，如果一个视频图像显示在一个窗口中，就会因为窗口过多而显得凌乱。此时需要一个窗口能显示多个视频图像。要达到这个效果，原理并不复杂，只需要调整每个视频的尺寸大小为窗口的一部分，多幅图像组合起来正好占满一个窗口。

在 OpenCV 中，我们可以综合利用坐标变换与 Rect 区域提取来实现单窗口显示多幅图像。首先根据输入图像的个数与尺寸确定输入原图像小窗口的构成形态；然后设定每个图像小窗口的具体构成，包括边界、间隙等；最后根据小窗口确定输出图像的尺寸，利用缩放函数 resize 进行图像缩放，完成单窗口下多幅图像的显示。

【例 2.9】单窗口中显示多幅图像

步骤 01 打开 Qt Creator，新建一个控制台工程，工程名是 test。

步骤 02 在 IDE 中打开 main.cpp，输入如下代码：

```
#include <opencv2/opencv.hpp>
using namespace cv;
#include<iostream>
using namespace std;
void showManyImages(const vector<Mat>&srcImages, Size imageSize){
    int nNumImages = srcImages.size();
    Size nSizeWindows;
    if (nNumImages > 12){
        cout << "no more tha 12 images" << endl;
        return;
    }
    // 根据图像序列数量来确定分割小窗口的形态
    switch (nNumImages){
    case 1:nSizeWindows = Size(1, 1); break;
    case 2:nSizeWindows = Size(2, 1); break;
    case 3:
    case 4:nSizeWindows = Size(2, 2); break;
    case 5:
    case 6:nSizeWindows = Size(3, 2); break;
    case 7:
    case 8:nSizeWindows = Size(4, 2); break;
    case 9:nSizeWindows = Size(3, 3); break;
    default:nSizeWindows = Size(4, 3);
    }
    // 设置小图像尺寸、间隙、边界
    int nShowImageSize = 200;
    int nSplitLineSize = 15;
    int nAroundLineSize = 50;
    // 创建输出图像，图像大小根据输入源来确定
    const int imagesHeight = nShowImageSize*
        nSizeWindows.width + nAroundLineSize +
        (nSizeWindows.width - 1)*nSplitLineSize;
```

```
        const int imagesWidth = nShowImageSize*
            nSizeWindows.height + nAroundLineSize +
            (nSizeWindows.height - 1)*nSplitLineSize;
        cout << imagesWidth << "  " << imagesHeight << endl;
        Mat showWindowsImages(imagesWidth, imagesHeight, CV_8UC3, Scalar(0, 0,
0));
        // 提取对应小图像的左上角坐标 x、y
        int posX = (showWindowsImages.cols-(nShowImageSize*nSizeWindows.width +
            (nSizeWindows.width - 1)*nSplitLineSize)) / 2;
        int posY = (showWindowsImages.rows-(nShowImageSize*nSizeWindows.height +
            (nSizeWindows.height - 1)*nSplitLineSize)) / 2;
        cout << posX << "  " << posY << endl;
        int tempPosX = posX;
        int tempPosY = posY;
        // 将多幅小图像整合成一幅大图像
        for (int i = 0; i < nNumImages; i++){
            // 小图像坐标转换
            if ((i%nSizeWindows.width == 0) && (tempPosX != posX)){
                tempPosX = posX;;
                tempPosY += (nSplitLineSize + nShowImageSize);
            }
            // 利用 Rect 区域将小图像置于大图像的相应区域
            Mat tempImage = showWindowsImages
                (Rect(tempPosX, tempPosY, nShowImageSize, nShowImageSize));
            // 利用 resize 函数实现图像缩放
            resize(srcImages[i], tempImage,
                Size(nShowImageSize, nShowImageSize));
            tempPosX += (nSplitLineSize + nShowImageSize);
        }
        imshow("Display multiple images in a single window", showWindowsImages);
    }

    int main(){
        // 图像源输入
        vector<Mat>srcImage(9);
        srcImage[0] = imread("1.jpg");
        srcImage[1] = imread("1.jpg");
        srcImage[2] = imread("1.jpg");
        srcImage[3] = imread("2.jpg");
        srcImage[4] = imread("2.jpg");
        srcImage[5] = imread("2.jpg");
        srcImage[6] = imread("3.jpg");
        srcImage[7] = imread("3.jpg");
        srcImage[8] = imread("3.jpg");
        // 判断当前 vector 读入的正确性
        for (int i = 0; i < srcImage.size(); i++){
            if (srcImage[i].empty()){
                cout << "read error" << endl;
                return -1;
            }
```

```
        }
        // 调用单窗口显示图像
        showManyImages(srcImage, Size(147, 118));
        waitKey(0);
        system("pause");
        return 0;
    }
```

在上述代码中，以 main 开头，首先读取 9 个图片文件，分别存入 srcImage 数组中，其中，srcImage[0]、srcImage[1]和 srcImage[2]都存放的是 1.jpg，srcImage[3]、srcImage[4]和 srcImage[5]都存放的是 2.jpg，srcImage[6]、srcImage[7]和 srcImage[8]都存放的是 3.jpg；然后利用 for 循环判断是否读取成功；最后调用 showManyImages 传入 srcImage 数组和每幅图像的大小，这里的大小是 Size(147,118)。showManyImages 是自定义函数，里面定义了一个大的“底板”图像 showWindowsImages，9 幅图像分别占用 showWindowsImages 中的一块，并且每一块之间都设置了间隙。

步骤 03 保存工程并运行，结果如图 2-58 所示。

图 2-58

2.6.3 销毁窗口

既然有新建窗口，当然也有销毁窗口。在 OpenCV 中，销毁窗口时窗口将自动关闭，这可以

通过函数 destroyWindow 和 destroyAllWindows 来实现，前者是销毁某一个指定名称的窗口，后者是销毁所有新建的窗口。函数 destroyWindow 声明如下：

```
void destroyWindow(const String& winname);
```

其中参数 winname 是要销毁的窗口的名称。

函数 destroyWindow 更加简单，其声明如下：

```
void destroyAllWindows();
```

下面我们来新建 3 个窗口，每个窗口显示 5 秒，再分别销毁。

【例 2.10】销毁 3 个窗口

步骤 01　打开 Qt Creator，新建一个控制台工程，工程名是 test。

步骤 02　在 IDE 中打开 main.cpp，输入如下代码：

```
#include <opencv2/opencv.hpp>
using namespace cv;
#include<iostream>
using namespace std;
int main(){
    // 图像源输入
    vector<Mat>srcImage(3);
    char szName[50] = "";
    for (int i = 0; i < srcImage.size(); i++)
    {
        sprintf_s(szName, "%d.jpg", i+1);
        srcImage[i] = imread(szName); // 读取图像文件
        if (srcImage[i].empty()){ // 判断当前 vector 读入的正确性
            cout << "read "<< szName<<" error" << endl;
            return -1;
        }
        // 调用单窗口显示图像
        namedWindow(szName, WINDOW_AUTOSIZE);
        imshow(szName, srcImage[i]);// 在“窗口 1”这个窗口输出图像
        waitKey(5000);// 等待 5 秒，程序自动退出。改为 0，不自动退出
        destroyWindow(szName);
    }
    // destroyAllWindows();
    cout << "所有的窗口已经销毁了" << endl;
    waitKey(0);
    system("pause");
    return 0;
}
```

在上述代码中，我们在 for 循环中读取图像文件，然后新建窗口，并在窗口中显示图片 5 秒钟后销毁窗口。

如果不想在 for 循环中调用 destroyWindow 函数，也可以在 for 循环外面调用 destroyAllWindows 函数，这样 3 个窗口都显示后，再一起销毁。

步骤 03 保存工程并运行，结果如图 2-59 所示。

图 2-59

2.6.4 调整窗口大小

窗口大小可以通过手动拖拉窗口边框来调整，也可以通过函数方式来调整。调整窗口大小的函数是 resizeWindow，其声明如下：

```
void resizeWindow(const String& winname, int width, int height);
```

其中参数 winname 是要调整尺寸的窗口的名称，width 是调整后的窗口宽度，height 是调整后的窗口高度。

需要注意的是，新建窗口函数 namedWindow 的第二个参数必须为 WINDOW_NORMAL，才可以手动拉动窗口边框来调整大小，并且让图片随着窗口大小而改变。

【例 2.11】调整窗口大小

步骤 01 打开 Qt Creator，新建一个控制台工程，工程名是 test。

步骤 02 在 IDE 中打开 main.cpp，输入如下代码：

```
#include <opencv2\opencv.hpp>
using namespace cv;
#include<iostream>
using namespace std;

int main(){
    vector<Mat>srcImage(1);
    char szName[50] = "";
    int  width = 240, height = 120;

    sprintf_s(szName, "%d.jpg", 1);
    srcImage[0] = imread(szName);
    if (srcImage[0].empty()) { // 判断当前 vector 读入的正确性
        cout << "read " << szName << " error" << endl;
        return -1;
    }
    namedWindow(szName, WINDOW_NORMAL);     // 新建窗口
    imshow(szName, srcImage[0]);            // 在窗口显示图片
    resizeWindow(szName, width, height);    // 调整窗口大小
    waitKey(0);
```

```
    system("pause");
    return 0;
}
```

在上述代码中，首先读入一幅图片，然后新建一个窗口显示该图片，接着调用函数 resizeWindow 调整窗口大小。由于 namedWindow 的第二个参数是 WINDOW_NORMAL，因此图片大小会随着窗口大小的变化而变化。

步骤 03　保存工程并运行，结果如图 2-60 所示。

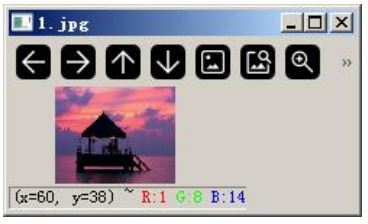

图 2-60

2.6.5　鼠标事件

在 OpenCV 中存在鼠标的操作，比如单击、双击等。对于 OpenCV 来讲，用户的鼠标操作被认为发生了一个鼠标事件，需要对这个鼠标事件进行处理，这就是事件的响应。下面来介绍一下鼠标事件。

鼠标事件包括左键按下、左键松开、左键双击、鼠标移动等。当鼠标事件发生时，OpenCV 会让一个鼠标响应函数自动被调用，相当于一个回调函数，这个回调函数就是鼠标事件处理函数。OpenCV 提供了 setMousecallback 来预先设置回调函数，相当于告诉系统鼠标处理的回调函数已经设置好了，有鼠标事件发生时，系统调用这个回调函数即可。注意是系统调用，而不是开发者调用，因此称为回调函数。函数 setMousecallback 声明如下：

```
    void setMousecallback(const string& winname, MouseCallback onMouse, void*
userdata=0);
```

其中参数 winname 表示窗口的名字，onMouse 是鼠标事件响应的回调函数指针，userdate 是传给回调函数的参数。这个函数名也比较形象，一看就知道是用来设置鼠标回调函数的（set Mouse call back）。

鼠标事件回调函数类型 MouseCallback 定义如下：

```
    typedef void(* cv::MouseCallback)(int event,int x,int y,int flags,void
*useradata);
```

其中参数 event 表示鼠标事件，x 表示鼠标事件的 *x* 坐标，y 表示鼠标事件的 *y* 坐标，flags 表示鼠标事件的标志，userdata 是可选的参数。

鼠标事件 event 主要有以下几种：

```
    enum
    {
```

```
    EVENT_MOUSEMOVE      =0,     // 滑动
    EVENT_LBUTTONDOWN    =1,     // 左键单击
    EVENT_RBUTTONDOWN    =2,     // 右键单击
    EVENT_MBUTTONDOWN    =3,     // 中键单击
    EVENT_LBUTTONUP      =4,     // 左键放开
    EVENT_RBUTTONUP      =5,     // 右键放开
    EVENT_MBUTTONUP      =6,     // 中键放开
    EVENT_LBUTTONDBLCLK   =7,    // 左键双击
    EVENT_RBUTTONDBLCLK   =8,    // 右键双击
    EVENT_MBUTTONDBLCLK   =9     // 中键双击
};
```

鼠标事件标志 flags 主要有以下几种：

```
enum {
    EVENT_FLAG_LBUTTON = 1,     // 左键拖曳
    EVENT_FLAG_RBUTTON = 2,     // 右键拖曳
    EVENT_FLAG_MBUTTON = 4,     // 中键拖曳
    EVENT_FLAG_CTRLKEY = 8,     // 按 Ctrl
    EVENT_FLAG_SHIFTKEY = 16,   // 按 Shift
    EVENT_FLAG_ALTKEY = 32      // 按 Alt
};
```

通过 event 和 flags 就能清楚地了解当前鼠标发生了哪种操作。

【例 2.12】在图片上用鼠标画线

步骤 01 打开 Qt Creator，新建一个控制台工程，工程名是 test。

步骤 02 在 IDE 中打开 main.cpp，输入如下代码：

```
#include <opencv2\opencv.hpp>
using namespace cv;
#include<iostream>
using namespace std;
#define WINDOW "原图"
Mat g_srcImage, g_dstImage;
Point previousPoint;
bool P = false;
void on_mouse(int event, int x, int y, int flags, void*);
int main()
{
    g_srcImage = imread("ter.jpg", 1);
    imshow(WINDOW, g_srcImage);
    setMouseCallback(WINDOW, On_mouse, 0);
    waitKey(0);
    return 0;
}
void on_mouse(int event, int x, int y, int flags, void*)
{
    if (event == EVENT_LBUTTONDOWN)
    {
        previousPoint = Point(x, y);
```

```
        }
        else if (event == EVENT_MOUSEMOVE && (flags&EVENT_FLAG_LBUTTON))
        {
            Point pt(x, y);
            line(g_srcImage, previousPoint, pt, Scalar(0, 0, 255), 2, 5, 0);
            previousPoint = pt;
            imshow(WINDOW, g_srcImage);
        }
    }
```

在上述代码中，on_mouse 就是用来处理鼠标事件的回调函数，当鼠标有动作产生时，on_mouse 会被系统调用，然后在 on_mouse 中判断发生了哪种动作，进而进行相应的处理。本例我们关心的是按下鼠标左键，然后按住左键时的鼠标移动。一旦按下鼠标左键，就记录下当时的位置（也就是画线的开始点），位置记录在 previousPoint 中。接着如果继续有鼠标移动事件发生，我们就开始调用函数 line 画线。最后把画了线的图像数据 g_srcImage 用函数 imshow 显示出来。

步骤 03　保存工程并运行，结果如图 2-61 所示。

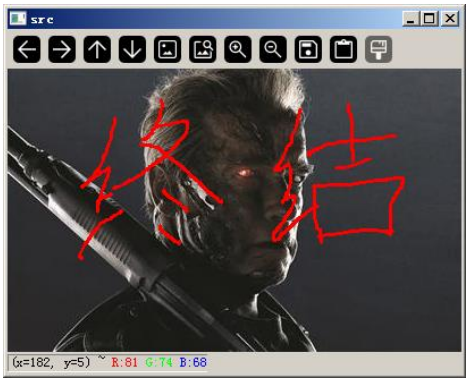

图 2-61

2.6.6　键盘事件

简单、常用的键盘事件是等待按键事件，它由 waitKey 函数来实现，该函数是我们的老朋友了，前面也碰到过多次。无论是刚开始学习 OpenCV，还是使用 OpenCV 进行开发调试，都可以看到 waitKey 函数的身影；然而，基础的东西往往容易被忽略掉，在此可以好好了解一下这个基础又常用的 waitKey 函数。该函数延时一定时间，返回按键的值；当参数为 0 时就永久等待，直到用户按键。该函数声明如下：

```
int cv::waitKey(int delay = 0) ;
```

其中参数 delay 是延时的时间，单位是毫秒，默认是 0，表示永久等待。该函数在至少创建了一个 HighGUI 窗口并且该窗口处于活动状态时才有效。如果有多个 HighGUI 窗口，则其中任何一个都可以处于活动状态。

waitKey 函数是一个等待键盘事件的函数，当参数值 delay≤0 时，等待时间无限长。当 delay 为正整数 *n* 时，至少等待 *n* 毫秒才结束。在等待期间，如果任意按键被按下，则函数结束，返回按

键的键值（ASCII 码）。如果等待时间结束用户仍未按下按键，则函数返回−1。该函数用在处理 HighGUI 窗口的程序中，常与显示图像窗口的 imshow 函数搭配使用。

比如配合图像显示时的常见用法如下：

```
    // 例 1
    cv::imshow("windowname", image);
    cv::waitKey(0);// 任意按键被按下，图片显示结束，返回按键的键值
    // 例 2
    cv::imshow("windowname", image);
    cv::waitKey(10);// 等待至少 10ms 图片显示才结束，期间按下任意按键图片显示结束，返回按键的
键值
```

视频播放时的常见用法如下：

```
    // 例 1
    VideoCapture cap("video.mp4");
    if(!cap.isOpened())
    {
        return -1;
    }
    Mat frame;
    while(true)
    {
        cap>>frame;
        if(frame.empty()) break;
        imshow("windowname",frame);
        if(waitKey(30) >=0) // 延时 30ms，以正常的速率播放视频，播放期间按下任意按
           break;           // 键则退出视频播放，并返回键值
    }

    // 例 2
    VideoCapture cap("video.mp4");
    if(!cap.isOpened())
    {
        return -1;
    }
    Mat frame;
    while(true)
    {
        cap>>frame;
        if(frame.empty()) break;
        imshow("windowname",frame);
        if(waitKey(30) == 27) // 延时 30ms，以正常的速率播放视频，播放期间按下 Esc 按键
            break;            //则退出视频播放，并返回键值
    }
```

总而言之，waitKey 函数是非常简单而且常见的，读者开始入门的时候需要掌握好它，它在开发调试的时候同样是一个好帮手。

2.6.7　滑动条事件

在 OpenCV 中，滑动条设计的主要目的是在视频播放帧中选择特定帧，它在调节图像参数时也会经常用到。在使用滑动条前，需要给滑动条赋予一个名字（通常是一个字符串），接下来将直接通过这个名字进行引用。

创建滑动条的函数是 createTrackbar，该函数声明如下：

```
    int cv::createTrackbar (const String & trackbarname, const String & winname,
int * value,    int count, TrackbarCallback onChange = 0,  void *userdata = 0);
```

参数说明如下：

- trackbarname：表示滑动条的名称。
- Winname：是滑动条将要添加到父窗口的名称，一旦滑动条创建好，它就将被添加到窗口的顶部或底部，滑动条不会挡住任何已经在窗口中的图像，只会让窗口变大，窗口的名称将作为一个窗口的标记，至于滑动条上滑动按钮的确切位置，由操作系统决定，一般都是在最左边。
- value：是一个指向整数的指针，这个整数值会随着滑动按钮的移动而自动变化。
- count：是滑动条可以滑动的最大值。
- onChange：是一个指向回调函数的指针，当滑动按钮移动时，回调函数就会被自动调用。
- userdata：可以是任何类型的指针，一旦回调函数执行，这个参数可以传递给回调函数的 userdata 参数，这样不创建全局变量也可以处理滑动条事件。

回调函数类型 TrackbarCallback 的定义如下：

```
    typedef void(* cv::TrackbarCallback) (int pos, void *userdata);
```

其中参数 pos 表示滚动块的当前位置；userdata 是传给回调函数的可选参数。这个回调函数不是必需的，如果直接赋值为 NULL，就没有回调函数，移动滑动按钮的唯一响应就是 createTrackbar 的参数 value 指向的变量值的变化。

除了创建滑动条的函数外，OpenCV 还提供了函数 getTrackbarPos（用于获取滑动块的位置）和 setTrackbarPos（用于设置滑动条的位置）。函数 getTrackbarPos 声明如下：

```
    int cv::getTrackbarPos (const string& trackName, const string& windowName);
```

其中参数 trackName 是滑动条的名称，windowName 是滑动条将要添加到父窗口的名称。函数返回滑动块的当前位置。

函数 setTrackbarPos 声明如下：

```
    void  cv::setTrackbarPos(const string &trackName, const string& windowName,
            int pos);
```

其中参数 trackName 表示滑动条的名称，windowName 是滑动条将要添加到父窗口的名称，pos 表示要设置的滑动块位置。下面来看一个专业的例子，利用滑动块调节参数。

【例 2.13】利用滑动块控制腐蚀和膨胀

步骤 01　打开 Qt Creator，新建一个控制台工程，工程名是 test。

步骤 02 在 IDE 中打开 main.cpp，输入如下代码：

```
#include <opencv2\opencv.hpp>
using namespace cv;
#include<iostream>
using namespace std;

Mat src, erosion_dst, dilation_dst;
int erosion_elem = 0;
int erosion_size = 0;
int dilation_elem = 0;
int dilation_size = 0;
int const max_elem = 2;
int const max_kernel_size = 21;
void Erosion(int, void*);
void Dilation(int, void*);
int main(int argc, char** argv)
{
    src = imread("ter.jpg", IMREAD_COLOR);
    if (src.empty())
    {
        cout << "Could not open or find the image!\n" << endl;
        return -1;
    }
    namedWindow("Erosion Demo", WINDOW_AUTOSIZE); // 腐蚀演示窗口
    namedWindow("Dilation Demo", WINDOW_AUTOSIZE); // 膨胀演示窗口
    moveWindow("Dilation Demo", src.cols, 0);
    createTrackbar("Element:\n 0: Rect \n 1: Cross \n 2: Ellipse", "Erosion
Demo",
        &erosion_elem, max_elem,Erosion);
    createTrackbar("Kernel size:\n 2n +1", "Erosion Demo",
        &erosion_size, max_kernel_size,Erosion);
    createTrackbar("Element:\n 0: Rect \n 1: Cross \n 2: Ellipse", "Dilation
Demo",
        &dilation_elem, max_elem,      Dilation);
    createTrackbar("Kernel size:\n 2n +1", "Dilation Demo",
        &dilation_size, max_kernel_size,Dilation);
    Erosion(0, 0);
    Dilation(0, 0);
    waitKey(0);
    return 0;
}
void Erosion(int, void*)
{
    int erosion_type = 0;
    if (erosion_elem == 0) { erosion_type = MORPH_RECT; }
    else if (erosion_elem == 1) { erosion_type = MORPH_CROSS; }
    else if (erosion_elem == 2) { erosion_type = MORPH_ELLIPSE; }
    Mat element = getStructuringElement(erosion_type,
        Size(2 * erosion_size + 1, 2 * erosion_size + 1),
```

```
            Point(erosion_size, erosion_size));
        erode(src, erosion_dst, element);
        imshow("Erosion Demo", erosion_dst);
    }
    void Dilation(int, void*)
    {
        int dilation_type = 0;
        if (dilation_elem == 0) { dilation_type = MORPH_RECT; }
        else if (dilation_elem == 1) { dilation_type = MORPH_CROSS; }
        else if (dilation_elem == 2) { dilation_type = MORPH_ELLIPSE; }
        Mat element = getStructuringElement(dilation_type,
            Size(2 * dilation_size + 1, 2 * dilation_size + 1),
            Point(dilation_size, dilation_size));
        dilate(src, dilation_dst, element);
        imshow("Dilation Demo", dilation_dst);
    }
```

在上述代码中，首先读取 ter.jpg，然后利用函数 namedWindow 创建两个窗口，接着利用函数 createTrackbar 创建 4 个滑动条，每个窗口上有两个滑动条。Erosion 和 Dilation 都是滑动条的回调函数，用于响应用户的滑动按钮事件。

函数 erode 和 dilate 分别是 OpenCV 中腐蚀和膨胀图像的函数，图像腐蚀和膨胀是图像学中的两个基本概念，后面章节会讲到。

步骤 03　保存工程并运行，结果如图 2-62 所示。

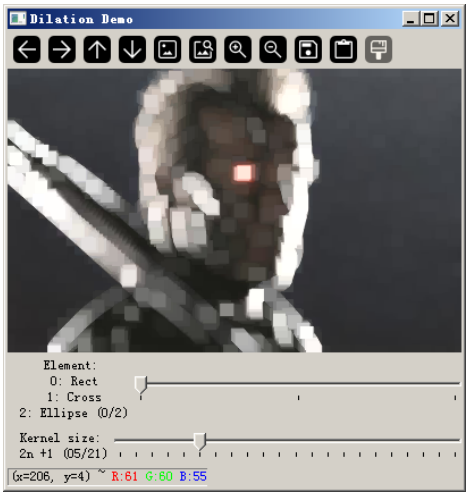

图 2-62