# 简介

## 概述

OpenCV全程Open Source Computer Vision Library，即开源计算机视觉库，由Intel建立，如今由Willow Garage支持。

## 基本架构

OpenCV由多个模块组成，分别是：

1、calib3d：Calibration（校准）和3D组合缩写，该模块主要是相机校准和三维重建相关的内容，包括基本的多视角几何算法、单个立体摄像头标定、物体姿态估计、立体相似度算法、3D信息重建等。

2、contrib：Contributed/Experimental Stuf的缩写，该模块包含了一些最近添加的不太稳定的可选功能，如新增了新型人脸识别、立体匹配、人工视网膜模型等技术。

# 环境搭建

## 下载

OpenCV3.1：

<https://opencv.org/opencv-3-1/>

<https://sourceforge.net/projects/opencvlibrary/>

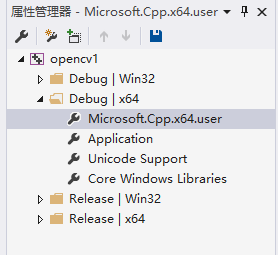
## 环境变量

设置系统环境变量Path:

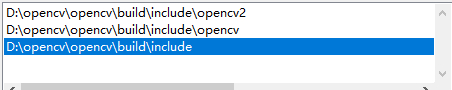
D:\opencv\opencv\build\x64\vc14\bin

## VS设置

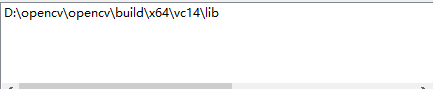
选择属性管理器🡪DebugX64🡪MicrosoftCppX64.user



选择属性🡪VC++目录🡪包含目录：



设置库目录：



设置链接器🡪输入🡪附加依赖项：

opencv\_world310d.lib

## 新建工程

源代码🡪添加🡪添加新项🡪Win32控制台🡪C++文件🡪空项目

代码：

#include <opencv2/opencv.hpp>

#include <iostream>

using namespace cv;

int main(int argc, char\*\* argv)

{

Mat src = imread("D:/opencv/projects/test.jpg");

if (src.empty())

{

printf("could not load image!");

}

namedWindow("src image", CV\_WINDOW\_AUTOSIZE);

imshow("src image", src);

waitKey(0);

return 0;

}

生成🡪生成解决方案🡪设置Debug X64🡪调试

注意：这里需要设置为Debug X64，因为前面设置的是Debug的库信息等，如果是Release或者Debug X86都是生成解决方案失败的。

# HighGUI图形用户界面

## 图像

### 加载图像

API：cv::imread

imread功能是加载图像文件成为一个Mat对象，其中：

imread("img\_path", IMREAD\_UNCHANGED);

第一个参数表示图像文件名；

第二个参数，表示加载的图像是什么类型，支持常见的三个参数值：

IMREAD\_UNCHANGED(<0)：表示加载原图，不做任何改变

IMREAD\_GRAYSCALE(0)：表示把原图作为灰度图像加载进来

IMREAD\_COLOR(>0)：表示把原图作为RGB图像加载进来

注意：OpenCV支持JPG、PNG、TIFF等常见格式图像文件加载

代码：

#include <opencv2\opencv.hpp>

#include <iostream>

using namespace cv;

int main(int argc, char\*\* argv)

{

Mat img = imread("D:/opencv/projects/2.jpg",1);//1等价于IMREAD\_ COLOR

if (img.empty())

{

printf("can not load img!");

return -1;

}

namedWindow("Windows", CV\_WINDOW\_AUTOSIZE);

imshow("Windows", img);

waitKey(0);

return 0;

}

### 显示图像

API：cv::namedWindows与cv::imshow

namedWindows功能是创建一个OpenCV窗口，它是由OpenCV自动创建与释放的，无需手动销毁。

常见用法：

namedWindows(" Windows Title ",WINDOW\_AUTOSIZE);

第一个参数是窗口的名称；

第二个参数是大小设置，WINDOW\_AUTOSIZE会自动根据图像大小，显示窗口大小，不能人为改变窗口大小；WINDOW\_NORMAL跟QT集成的时候会使用，允许修改窗口大小。

isShow根据窗口名称显示图像到指定的窗口上去。

第一个参数是窗口名称；

第二个参数是Mat对象。

### 修改图像

API：cv::cvtColor

cvtColor的功能是把图像从一个彩色空间转换到另外一个色彩空间。

cvtColor(image, gray\_imge, COLOR\_BGR2GRAY);

参数：

第一个参数表示源图像；

第二个参数表示色彩空间转换之后的图像；

第三个参数表示源和目标色彩空间，如：COLOR\_BGR2HLS（亮度/饱和度）、COLOR\_BGR2GRAY（灰度）等。

代码：

#include <opencv2\opencv.hpp>

#include <iostream>

using namespace cv;

int main(int argc, char\*\* argv)

{

Mat src = imread("D:/opencv/projects/2.jpg",1);

if (src.empty())

{

printf("can not load img!");

return -1;

}

namedWindow("src img", CV\_WINDOW\_AUTOSIZE);

imshow("src img", src);

namedWindow("out img", CV\_WINDOW\_AUTOSIZE);

Mat out;

cvtColor(src, out, CV\_BGR2GRAY);

imshow("out img", out);

waitKey(0);

return 0;

}

### 保存图像

API：cv::imwrite

保存图像文件到指定目录路径。

只有8位、16位的PNG、JPG、TIFF文件格式而且是单通道或者三通道的BGR的图像才可以通过这种方式保存。

保存PNG格式的时候可以保存透明通道的图片。

可以指定压缩参数。

代码：

#include <opencv2\opencv.hpp>

#include <iostream>

using namespace cv;

int main(int argc, char\*\* argv)

{

Mat src = imread("D:/opencv/projects/2.jpg",1);

if (src.empty())

{

printf("can not load img!");

return -1;

}

namedWindow("src img", CV\_WINDOW\_AUTOSIZE);

imshow("src img", src);

namedWindow("out img", CV\_WINDOW\_AUTOSIZE);

Mat out;

cvtColor(src, out, CV\_BGR2GRAY);

imshow("out img", out);

imwrite("D:/opencv/projects/6.jpg", out);

waitKey(0);

return 0;

}

## 视频

## 数据处理

## 滑动条

### 创建滑动条

createTrackbar函数用于创建一个可以调整整数值的滑动条（常常也称为轨迹条），并将滑动条附加到指定的窗口上。

### 获取当前轨迹条位置

getTrackbarPos用于获取当前轨迹条的位置。

## 鼠标操作

# core组件

## 数据结构

### 基础图像容器Mat

### 常用数据结构

#### Point类

#### Scalar类

#### Size类

#### Rect类

## 基本操作

## 像素

### 像素值存储

存储像素值需要指定颜色空间和数据类型。其中，颜色空间是针对一个给定的颜色，如何组合颜色元素以对其编码。最简单的颜色空间要属灰度级空间，只处理黑色和白色，对它们进行组合便可以产生不同程度的灰色。

对于彩色方式则有更多种类的颜色空间，但不论哪种方式都是把颜色分成三者或四个基元素，通过组合基元素可以产生所有的颜色。RGB颜色空间是最常用的一种颜色空间，这归功于它也是人眼内部构成颜色的方式。

### 访问像素值

### 获取图像像素指针

CV\_Assert(myImage.depth()==CV\_8U);

Mat.ptr<uchar>(int i=0)获取像素矩阵的指针，索引i表示第几行，从0开始计行数。

获得当前行指针const uchar\* current = myImage.ptr<uchar>(row);

获取当前像素点P(row,col)的像素值p(row,col) = current[col]

### 图像运算

#### 像素范围操作

API：saturate\_cast<uchar>

saturate\_cast的功能是确保RGB值的范围在0~255之间（大于255按照255处理，小于0按照0处理）

saturate\_cast<uchar>(-100)，返回0

saturate\_cast<uchar>(288)，返回255

saturate\_cast<uchar>(100)，返回100

#### 掩膜操作

矩阵的掩膜（mask，也称为kernel，核函数）操作十分简单，通过掩膜操作实现图像对比度提高。

I(I,j) = 5\*I(i,j) – [I(I,j) – [I(i+1,j) + I(i,j-1) + I(I,j+1)]]

代码：

#include <opencv2\opencv.hpp>

#include <iostream>

#include <math.h>

using namespace cv;

int main(int argc, char\*\* argv)

{

Mat src = imread("D:/opencv/projects/2.jpg",1);

if (src.empty())

{

printf("can not load img!");

return -1;

}

namedWindow("src img", CV\_WINDOW\_AUTOSIZE);

imshow("src img", src);

int cols = (src.cols-1) \* src.channels();//宽度

int offset = src.channels();

int rows = src.rows;//高度

Mat dst;

dst = Mat::zeros(src.size(),src.type());

for (int row = 1; row < (row - 1); row++) {

const uchar\* previous = src.ptr<uchar>(row - 1);

const uchar\* current = src.ptr<uchar>(row);

const uchar\* next = src.ptr<uchar>(row + 1);

uchar\* output = dst.ptr<uchar>(row);

for (int col = offset; col < cols; col++) {

output[col] = saturate\_cast<uchar>(5\*current[col] - (current[col-offset] + current[col+offset] + previous[col] + next[col]));

}

}

namedWindow("dst img", CV\_WINDOW\_AUTOSIZE);

imshow("dst img", dst);

waitKey(0);

return 0;

}

调用函数filter2D功能：

1. 定义掩膜：Mat kernel = (Mat\_<char>(3,3)<<0,-1,0,-1,5,-1,0,-1,0);
2. Filter2D(src,dst,src.depth(),kernel);其中，src与dst是Mat类型变量，src.depth()表示位图深度，有32、24、8等。

代码：

#include <opencv2\opencv.hpp>

#include <iostream>

#include <math.h>

using namespace cv;

int main(int argc, char\*\* argv)

{

Mat src = imread("D:/opencv/projects/2.jpg",1);

Mat dst;

if (src.empty())

{

printf("can not load img!");

return -1;

}

namedWindow("src img", CV\_WINDOW\_AUTOSIZE);

imshow("src img", src);

/\*

int cols = (src.cols-1) \* src.channels();//宽度

int offset = src.channels();

int rows = src.rows;//高度

dst = Mat::zeros(src.size(),src.type());

for (int row = 1; row < (row - 1); row++) {

const uchar\* previous = src.ptr<uchar>(row - 1);

const uchar\* current = src.ptr<uchar>(row);

const uchar\* next = src.ptr<uchar>(row + 1);

uchar\* output = dst.ptr<uchar>(row);

for (int col = offset; col < cols; col++) {

output[col] = saturate\_cast<uchar>(5\*current[col] - (current[col-offset] + current[col+offset] + previous[col] + next[col]));

}

}

\*/

Mat kernel = (Mat\_<char>(3,3)<<0,-1,0,-1,5,-1,0,-1,0);

filter2D(src,dst,src.depth(),kernel);

namedWindow("dst img", CV\_WINDOW\_AUTOSIZE);

imshow("dst img", dst);

waitKey(0);

return 0;

}

### 图像重映射

### 直方图统计像素

#### 计算图像直方图

#### 利用查找表修改图像外观

#### 直方图均值化

#### 反向投影直方图检测特定图像内容

#### 均值平移算法查找目标

#### 比较直方图搜索相似图像

#### 积分图像统计像素

## 颜色

### 策略设计模式比较颜色

### GrabCut算法分割图像

### 颜色空间转换

cvtColor函数是OpenCV里的颜色空间转换函数，可以实现RGB颜色向HSV、HIS等颜色空间的转换，也可以转换为灰度图像。

### 色调/饱和度/亮度表示颜色

## ROI区域图像叠加&图像混合

### 感兴趣区域ROI

在图像处理领域，我们常常需要设置感兴趣区域（ROI：region of interest），来专注或者简化工作过程。

定义ROI区域有两种方法：第一种是使用表示矩形区域的Rect。它指定矩形的左上角坐标（构造函数的前两个参数）和矩阵的长度（构造函数的后两个参数）以定义一个矩形区域。

另一种定义ROI的方式是指定感兴趣行或列的范围（Range）。Range是指从起始索引到终止索引（不包括终止索引）的一连段连续序列。cRange可以用来定义Range。

### 线性混合操作

API：addWeight

## 多通道

### 通道分离：split()

split函数用于将一个多通道数组分离成几个单通道数组。

### 通道合并：merge()

Merge()函数是split函数的逆向操作—将多个数组合并成一个多通道的数组。它通过组合一些给定的单通道数组，将这些孤立的单通道数组合成一个多通道的数组，从而创建出一个由多个单通道阵列组成的多通道阵列。

## 对比度/亮度值

算子

一般的图像处理算子都是一个函数，它接受一个或多个输入图像，并产生输出图像。算子的一般形式：

g(x)=h(f(s))或者g(x)=h(f0(x)……fn(x))

点操作

点操作有一个特点：仅仅根据输入像素值（有时可加上某些全局信息或参数），来计算响应的输出像素值。这些算子包括亮度（brightess）和对比度（contrast）调整、颜色校正（colorcorrection）和变换（transformations）。

两种常见的点操作（或者说是算子）是乘上一个常数（对应对比度的调节）以及加上一个常数（对应亮度值的调节），公式如下：

g(x)=a\*f(x)+b

参数a称为增益（gain），常常被用来控制图像的对比度，参数b称为偏差（bias），常常被用来控制图像的亮度。

## 离散傅里叶变换

## 输入输出XML/YAML

# imgproc组件

## 图像处理

### 卷积

#### 用cv::filter2D进行卷积

#### 通过cv::sepFilter2D使用可分核

#### 生成卷积核

### 线性滤波

smooth和blur是图像处理中最简单和常用的操作之一。使用该操作的原因之一就是为了给图像预处理时候减低噪声，使用smooth、blur操作背后的是数学的卷积计算，通常这些卷积算子计算都是线性操作，所以又叫线性滤波。

线性滤波器：线性滤波器经常用于剔除输入信号中不想要的频率或者从许多频率中选择一个想要的频率。

几种常见的线性滤波器：

低通滤波器：允许低频率通过

高通滤波器：允许高频率通过

带通滤波器：允许一定范围的频率通过

带阻滤波器：阻止一定范围通过并且允许其他频率通过

全通滤波器：允许所有频率通过，仅仅改变相位关系

陷波滤波器：阻止一个狭窄频率范围通过，是一种特殊的带阻滤波器

#### 邻域算子与线性邻域滤波

#### 方框滤波

#### 均值滤波



假设有6\*6的图像像素点矩阵。

卷积过程：6\*6上面是个3\*3的窗口（即3\*3算子），从左向右，从上向下移动，黄色的每个像素点值之和取平均值赋给中心红色像素作为它卷积处理之后新的像素值。每次移动一个像素格。

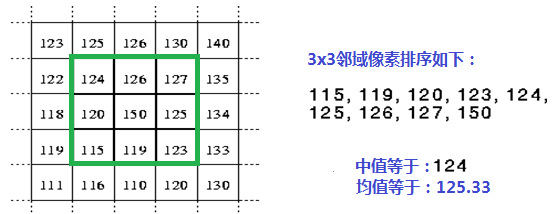
注：最边缘的1个像素被忽略了，没有被处理，如果7\*7的核则有5个边缘像素没有被处理，处理边缘的方法可以是向外扩散或者插值，这样就可以处理边缘像素了。

#### 高斯滤波

### 非线性滤波

#### 中值滤波

中值滤波对椒盐噪声具有很好的抑制作用。



#### 双边滤波

均值模糊无法克服边缘像素信息丢失缺陷，原因是均值滤波是基于平均权重。

高斯模糊部分克服了该缺陷，但是无法完全避免，因为没有考虑像素值的不同。

高斯双边模糊是边缘保留的滤波方法，避免了边缘信息丢失，保留了图像轮廓不变。

### 形态学滤波

#### 腐蚀与膨胀

#### 开运算/闭运算/形态学梯度/顶帽/黑帽

### 漫水填充

漫水填充法是一种用哪个特定的颜色填充连通区域，通过设置可连通像素的上下限以及连通方式来达到不同的填充效果的方法。漫水填充经常被用来标记或分离图像的一部分，以便对其进行进一步处理或分析，也可以用来从输入图像获取掩码区域，掩码会加速处理过程，或只处理掩码指定的像素点，操作的结果总是某个连续区域。

### 图像金字塔与图片尺寸缩放

图像金字塔最初用于机器视觉和图像压缩，一幅图像的金字塔是一系列以金字塔形状排列的，分辨率逐步降低且来源于同一张原始图的图像集合。其通过梯次向下采样获得，直到达到某个终止条件才停止采样。

#### 高斯金字塔

#### 拉普拉斯金字塔

#### 尺寸调整

#### 尺寸缩放

### 阈值化

阈值可以被视作最简单的图像分割方法。

阈值的选取依赖于具体的问题，即物体在不同的图像中有可能会有不同的灰度值。

#### Ostu算法

#### 自适应算法

## 图像变换

图像变换（image transform），即将一幅图像转变成图像数据的另一种表现形式。变换最常见的例子就是傅里叶变换，即将图像转换为源图像数据的另一种表现形式。这类操作的结果仍然保存为OpenCV图像结构的形式，但是新图像的每个单独像素表示原始输出图像的频谱分量，而不是通过所考虑的空间分量。

### 边缘检测

边缘检测的一般步骤：

1. 滤波

边缘检测的算法主要是基于图像强度的一阶和二阶导数，但导数通常对噪声很敏感，因此必须采用滤波器来改善与噪声有关的边缘检测器的性能。常见的滤波方法主要有高斯滤波，即采用离散化的高斯函数产生一组归一化的高斯核，然后基于高斯核函数对图像灰度矩阵的每一点进行加权求和。

1. 增强

增强边缘的基础是确定图像各点邻域强度的变化值。增强算法可以将图像灰度点邻域强度值有显著变化的点凸显出来。在具体编程中，可以通过计算梯度幅值来确定。

1. 检测

经过增强的图像，往往邻域中有很多点的梯度值比较大，而在特定的应用中，这些点并不是要找的边缘点，所以应该采用某种方法来对这些点进行取舍。实际工程中，常用的方法是通过阈值化方法来检测。

#### canny算子

边缘勾画了图像的元素，含有重要的视觉信息。正因此如此，边缘可应用于目标识别等领域。但是简单的二至边缘分布图存在两个主要缺点：第一，检测到的边缘过厚，这加大了识别物体边界的难度；第二，也是最重要的，通常不可能找到既低到足以检测到图像中所有重要边缘，又高到足以避免产生太多无关紧要边缘的阈值。Canny算法试图解决这个问题。

#### sobel算子

#### Laplacian算子

#### scharr滤波器

### 霍夫变换

在许多应用场合中需要快速准确地检测出直线或者圆。其中一种非常有效的解决方法是霍夫（Hough）变换，其为图像处理中从图像中识别几何图形的基本方法之一。最基本的霍夫变换是从黑白图像中检测直线（线段）。

霍夫变换的目的是在二值图像中找出全部直线，并且这些直线必须穿过足够多的像素点。它的处理方法是，检查输入的二值分布图中每个独立的像素点，识别出穿过该像素点的所有可能直线。如果同一条直线穿多很多像素值，就说明这条直线明显到足以被认定。

### 重映射

重映射，就是把一幅图像中某位置的像素放置到另一个图片指定位置的过程。

### 仿射变换

仿射变换（Affine Transformation或Affine Map），又称仿射映射，是指在几何中，一个响亮空间进行一次线性变换并接上一个平移，变换为另一个向量空间的过程。它保持了二维图形的“平直性”（直线经过变换之后依然是直线）和“平行性”（二维图形之间的相对位置关系保持不变，平行线依然是平行线，且直线上点的位置顺序不变）。

一个任意的仿射变换都能表示为乘以一个矩阵（线性变换）接着再加上一个向量（平移）的形式。

我们能够用仿射变换来表示如下三种常见的变换形式：

旋转，rotation（线性变换）

平移，translation（向量加）

缩放，scale（线性变换）

#### 旋转

#### 平移

#### 缩放

### 直方图均值化

## 图像轮廓与分割

虽然Canny之类的边缘检测算法可以根据像素之间的差异，检测出轮廓边界的像素，但是它并没有将轮廓作为一个整体。所以，需要把这些边缘像素组装成轮廓。

### 查找并绘制轮廓

#### 查找轮廓

#### 绘制轮廓

### 寻找物体凸包

凸包就是将最外层的点连接起来构成的凸多边形，它是能包含点集中所有点的。理解物体形状或轮廓的一种比较有用的方法便是计算一个物体的凸包，然后计算其凸缺陷（convexity defect）。很多复杂物体的特性能很好地被这种缺陷表现出来。

### 使用多边形将轮廓包围

### 图像的矩

矩函数在图像分析中有着广泛的应用，如模式识别、目标分类、目标识别与方位估计、图像编码与重构等。

一阶矩与形状有关，二阶矩显示曲线围绕直线平均值的拓展程度，三阶矩则是关于平均值的对称性的测量。由二阶矩和三阶矩可以导出一组共7个不变矩。而不变矩是图像的统计特性，满足平移、伸缩、旋转均不变的不变性，在图像识别领域得到广泛应用。

### 分水岭算法

分水岭算法，是一种基于拓扑理论的数学形态学分割方法，其基本思想是把图像看作是测地学上的拓扑地貌，图像中每一点像素的灰度值表示该点的海拔高度，每一个局部极小值及其影响区域均为集水盆，而集水盆的边界则形成分水岭。

### 图像修补

## 直方图与匹配

直方图可以用来描述各种不同的参数和事物，如物体的色彩分布、物体边缘梯度模板，以及表示目标位置的当前假设的概率分布。

### 直方图计算与绘制

### 直方图对比

### 反向投影

### 模板匹配

# video组件

## 读取视频序列

## 处理视频帧

## 写入视频帧提取视频中的前景物体

# feature2d组件

## 角点检测

在计算机视觉领域，兴趣点（也称关键点或特征点）在目标识别、图像配准、视觉跟踪、三维重建等方面得到广泛应用。其原理是，从图像中选取某些特征点并对图像进行局部分析（即提取局部特征），而非观察整幅图像（即提取全局特征）。只要图像中有足够多的可检测的兴趣点，并且这些兴趣点各不相同且特征稳定、能被精确地定位，上述方法就十分有效。

因为要用于图像内容的分析，所以不管图像拍摄时采用了什么视角、尺度和方位，理想情况下同一场景或目标位置都要检测到特征点。

角点检测（Corner Detection）是计算机视觉系统中用来获取图像特征的一种方法，广泛应用于运动检测、图像匹配、视频追踪、三维建模和目标识别等领域中，也称为特征点检测。

角点通常被定义为两条边的交点，更严格的说法是，角点的局部邻域应该是具有两个不同区域的不同方向的边界。在实际应用中，大多数所谓的角点检测方法检测的是拥有特定特征的图像点，而不仅仅是“角点”。

在图像中搜索有价值的特征点（兴趣点/关键点）时，使用角点是一种不错的方法。

### Harris角点检测

OpenCV中检测Harris角点的基本函数是cv::cornerHarris。

调用该函数时输入一幅图像，返回的结果是一个浮点数型图像，其中每个像素表示角点强度。然后对输出图像阈值化，以获得检测角点的集合。

### Shi-Tomasi角点检测

Shi-Tomasi算法是Harris算法的改进，此算法最原始的定义是将矩阵M的行列式值与M的迹相减，再将差值同预先给定的阈值进行比较。

goodFeaturesToTrack()函数结合了Shi-Tomasi算子，用于确定图像的强角点。

### 亚像素级角点检测

如果我们进行图像处理的目的不是提取用于识别的特征点而是进行几何测量，这通常需要更高的精度，而函数goodFeatureToTrack()只能提供简单的像素的坐标值也就是说，有时候需要实数坐标值而不是整数坐标值。

亚像素级角点检测的位置在摄像机标定、跟踪并重建摄像机的轨迹，或者重建被跟踪目标的三维结构时，是一个基本的测量值。

## 快速检测特征

Harris算子需要计算图像的导数，而计算导数是非常耗时的。FAST（Feature from Acceleratd Segment Test，加速分割测试获得特征），这种算子用于专门快速检测兴趣点——只需对比几个像素，就可以判断它是否为兴趣点。

用这个算法检测兴趣点的速度非常快，因此十分适合需要优先考虑速度的应用，包括实现视觉跟踪、目标识别等，它们需要在实时视频流中跟踪或匹配多个点。

## 特征检测与匹配

计算机视觉需要图像预处理，比如特征提取，包括特征点，边缘和轮廓之类。以前做跟踪和3D重建，首先就得提取特征。特征点以前成功的是SIFT/SUR/FAST之类算法，现在还可以通过CNN模型形成的特征图来定义。

### 尺度不变特征检测法

尺度不变性的理念是，不仅在任何尺度下拍摄的物体都能检测到一致的关键点，而且每个被检测的特征点都对应一个尺度因子。理想情况下，对于两幅图像中不同尺度的同一个物体点，计算得到的两个尺度因子之间的比率应该等于图像尺度的比率。

#### SIFT尺度不变特征转换

SIFT（Scale-Invariant Feature Transform，尺度不变特征转换）是一种著名的尺度不变特征检测法，SURF算法是SIFT算法的加速版。

SIFT检测特征时采用了图像空间和尺度空间的局部最大值，但它使用拉普拉斯滤波器响应，而不是使用Hessian行列式值。

#### SURF特征点检测

SURF特征（Speeded Up Robust Feature，加速稳健特征）检测属于opencv\_contrib库，在编译OpenCV的时候包含了附加模块才能使用。

#### SURF特征提取

### 多尺度FAST特征检测

#### BRISK特征检测算法

BRISK（Binary Robust Invariant Scalable Keypoints，二元稳健恒定可拓展关键点）检测法。

#### ORB特征检测算法

ORB（Oriented FAST and Rotated BRIEF，定向FAST和旋转BRIEF）。

### 描述和匹配兴趣点

#### FLANN特征点匹配

#### BRISK特征匹配

#### ORB特征匹配

# calib3d组件

## 投影

## 三维重构

### 相机标定

### 相机姿态还原

### 标定相机实现三维重构

### 计算立体图像的深度

## 跟踪

### 跟踪视频中的特征点

### 估算光流

### 跟踪视频中的物体

# 机器学习

# 目标检测