



讲师：贾志刚

OpenCV4 特征提取实战教程



课程介绍

- OpenCV4开发环境
- 学习内容与方法
- 课程部分案例演示

OpenCV开发环境

- OpenCV4. x
- VS2017
- Windows10 64位
- 开发环境配置+代码测试

学习内容与方法

- OpenCV4 特征提取与对象检测模块
- OpenCV基础知识-图像处理与二值分析
- C++版本
- 原理解释+函数代码演示

课程部分案例演示

- 对象检测
- 文档对齐
- 瑕疵检测
- 无缝拼接
- 标签与条码定位

资料与代码

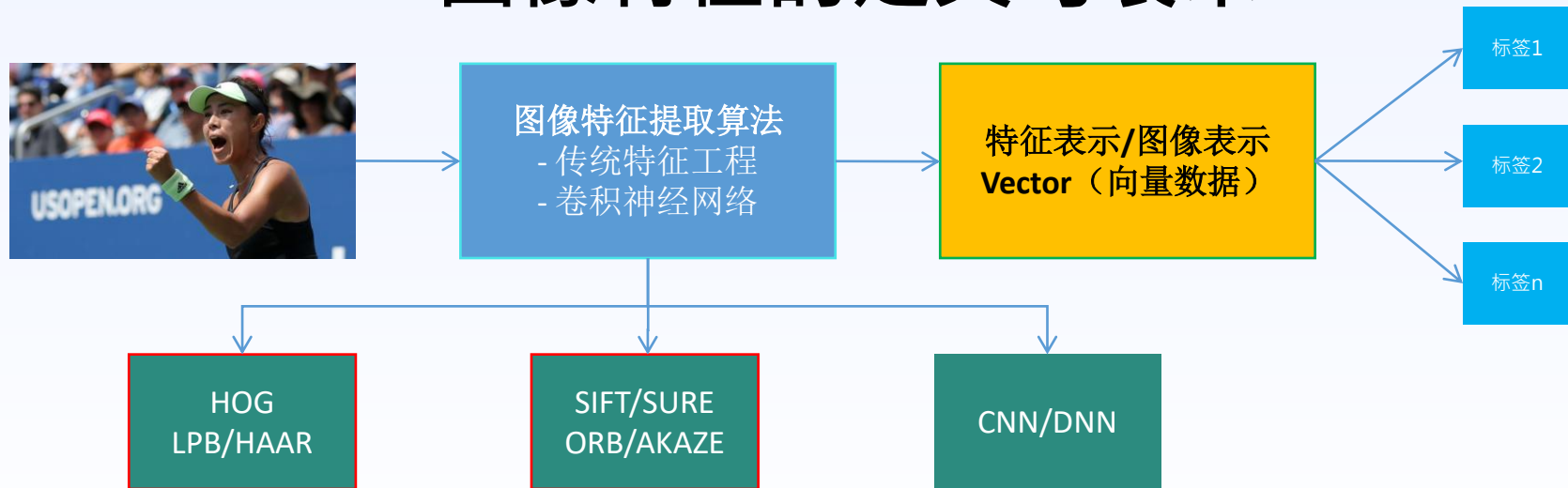
- 课程提供全部源码
- 提供演示代码相关的图像测试文件



什么是图像特征

- 图像特征的定义与表示
- 图像特征提取概述
- 图像特征的应用

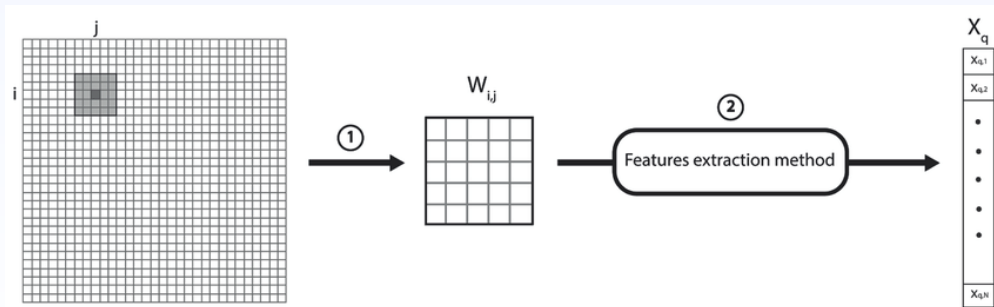
图像特征的定义与表示



图像特征表示是该图像唯一的表述，是图像的DNA

图像特征提取概述

- 传统图像特征提取 - 主要是基于纹理、角点、颜色分布、梯度、边缘等。
- 深度卷积神经网络特征提取 - 基于监督学习、自动提取特征
- 特征数据/特征属性
 - - 尺度空间不变性
 - - 像素迁移不变性
 - - 光照一致性原则
 - - 旋转不变性原则



图像特征应用



- 图像处理
- 从图像到图像
- 特征提取
- 从图像到向量（数据）

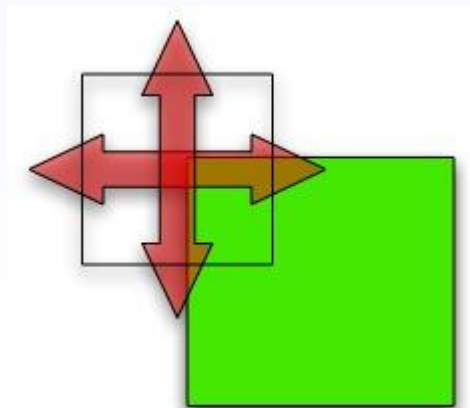
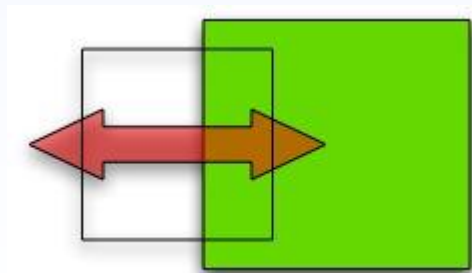


角点检测

- 角点检测算法介绍
- OpenCV函数支持
- 代码演示

角点检测算法

- 什么是图像的角点
- Harris角点检测算法
- Shi-tomas角点检测算法



角点检测算法

- 各个方向的梯度变化

$$E(u, v) = \sum_{x,y} w(x, y) [I(x + u, y + v) - I(x, y)]^2$$

$$\sum_{x,y} [I(x + u, y + v) - I(x, y)]^2$$

$$E(u, v) \approx \sum_{x,y} [I(x, y) + uI_x + vI_y - I(x, y)]^2$$

∴

$$E(u, v) \approx \sum_{x,y} u^2 I_x^2 + 2uv I_x I_y + v^2 I_y^2$$

$$E(u, v) \approx \begin{bmatrix} u & v \end{bmatrix} \left(\sum_{x,y} w(x, y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \right) \begin{bmatrix} u \\ v \end{bmatrix}$$

$$E(u, v) \approx \begin{bmatrix} u & v \end{bmatrix} \left(\sum_{x,y} w(x, y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \right) \begin{bmatrix} u \\ v \end{bmatrix}$$

$$M = \sum_{x,y} w(x, y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

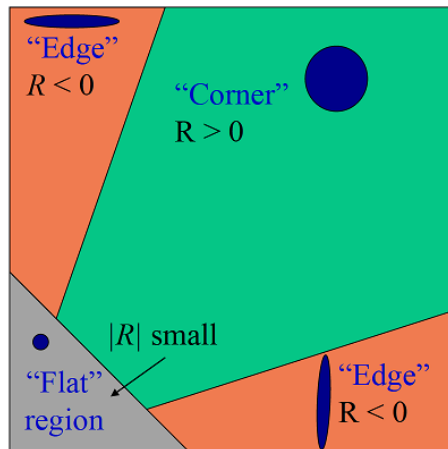
$$E(u, v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix}$$

角点检测算法

- Harris角点检测算法

$$R = \det(M) - k * \text{trace}(M)^2 = \lambda_1 \lambda_2 - k * (\lambda_1 + \lambda_2)^2$$

k: constant (0.04 to 0.06)

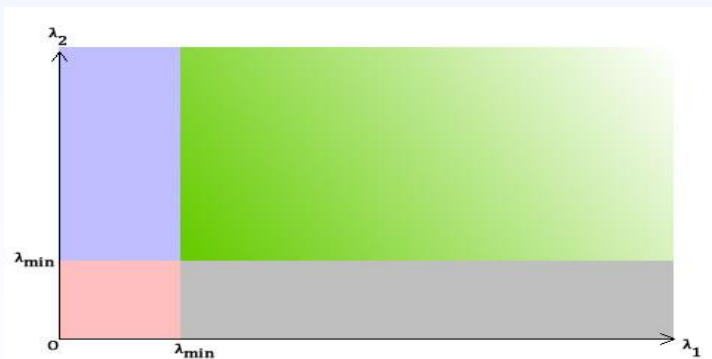


函数说明:

```
void cv::cornerHarris(  
    InputArray src, // 输入  
    OutputArray dst, // 输出  
    int blockSize, // 块大小  
    int ksize, // Sobel  
    double k, // 常量系数  
    int borderType = BORDER_DEFAULT  
)
```

角点检测算法

- Shi-Tomas角点检测算法



$$R = \min(\lambda_1, \lambda_2)$$

函数说明:

```
void cv::goodFeaturesToTrack(  
    InputArray image, // 输入图像  
    OutputArray corners, // 输出的角点坐标  
    int maxCorners, // 最大角点数目  
    double qualityLevel, // 质量控制  
    double minDistance, // 重叠控制  
    InputArray mask = noArray(),  
    int blockSize = 3,  
    bool useHarrisDetector = false,  
    double k = 0.04
```

)

代码演示



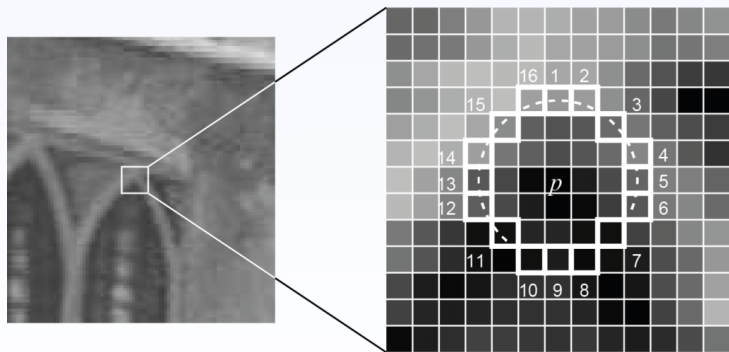


关键点检测

- 图像特征点/关键点
- 关键点检测函数
- 代码演示

ORB关键点检测

- 2011论文，比SIFT与SURF速度更快
- ORB算法可以看出两个部分组成：快速关键点定位+BRIEF描述子生成



Fast关键点检测：

选择当前像素点 P ，阈值 T ，周围16个像素点，超过连续 $N=12$ 个像素点大于或者小于 P ，

Fast1：优先检测1、5、9、13

循环所有像素点

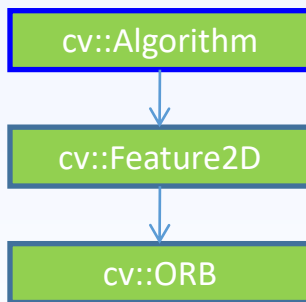
关键点检测函数

ORB对象创建,
`Orb = cv::ORB::create(500)`

```
virtual void cv::Feature2D::detect(  
    InputArray image, // 输入图像  
    std::vector< KeyPoint > & keypoints, // 关键点  
    InputArray mask = noArray() // 支持mask  
)
```

KeyPoint数据结构-四个最重要属性:

- pt
- angle
- response
- size



图像特征点/关键点

```
1 cv::Ptr<cv::ORB> orb = cv::ORB::create(500);  
2 cv::Mat img2Gray;  
3 cv::cvtColor(image, img2Gray, cv::COLOR_BGR2GRAY);  
4 std::vector<cv::KeyPoint> img_kps;  
5 cv::Mat img_descriptors;  
6 orb->detectAndCompute(img2Gray, cv::Mat(), img_kps, img_descriptors);
```





特征描述子

- 特征描述子
- 详解SIFT特征描述子
- 代码演示

图像特征描述子

- 基于关键点周围区域
- 浮点数表示与二值编码
- 描述子长度

Algorithm	OpenCV Object	Descriptor Size
SIFT	cv.SIFT('ConstrastThreshold',0.04,'Sigma',1.6)	128 Bytes
SURF(128D)	cv.SURF('Extended',true,'HessianThreshold',100)	128 Floats
SURF(64D)	cv.SURF('HessianThreshold',100)	64 Floats
KAZE	cv.KAZE('NOctaveLayers',3,'Extended',true)	128 Floats
AKAZE	cv.AKAZE('NOctaveLayers',3)	61 Bytes
ORB	cv.ORB('MaxFeatures',100000)	32 Bytes
ORB(1000)	cv.ORB('MaxFeatures',1000)	32 Bytes
BRISK	cv.BRISK()	64 Bytes
BRISK(1000)	cv.BRISK(); cv.KeyPointsFilter.retainBest(features,1000)	64 Bytes

ORB特征描述子

$$m_{pq} = \sum_{x,y} x^p y^q I(x, y),$$

几何矩公式

$$C = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}} \right)$$

计算中心位置

计算角度

$$\theta = \text{atan2}(m_{01}, m_{10}),$$

变换矩阵M

$$R_\theta = \begin{pmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{pmatrix}$$

像素块坐标矩阵 Q:

$$Q = \begin{bmatrix} x_1, x_2, \dots, x_N \\ y_1, y_2, \dots, y_N \end{bmatrix}$$

旋转矩阵 Q_θ :

$$Q_\theta = R_\theta Q \quad (10)$$

ORB特征描述子生成步骤

- 提取特征关键点
- 描述子方向指派
- 特征描述子编码（二值编码32位）

$$\tau(\mathbf{p}; \mathbf{x}, \mathbf{y}) := \begin{cases} 1 & : \mathbf{p}(\mathbf{x}) < \mathbf{p}(\mathbf{y}) \\ 0 & : \mathbf{p}(\mathbf{x}) \geq \mathbf{p}(\mathbf{y}) \end{cases}$$

$$f_n(\mathbf{p}) := \sum_{1 \leq i \leq n} 2^{i-1} \tau(\mathbf{p}; \mathbf{x}_i, \mathbf{y}_i)$$

SIFT特征描述子

- SIFT特征提取算法(**S**cale **I**nvariant **F**eature **T**ransform-**SIFT**)

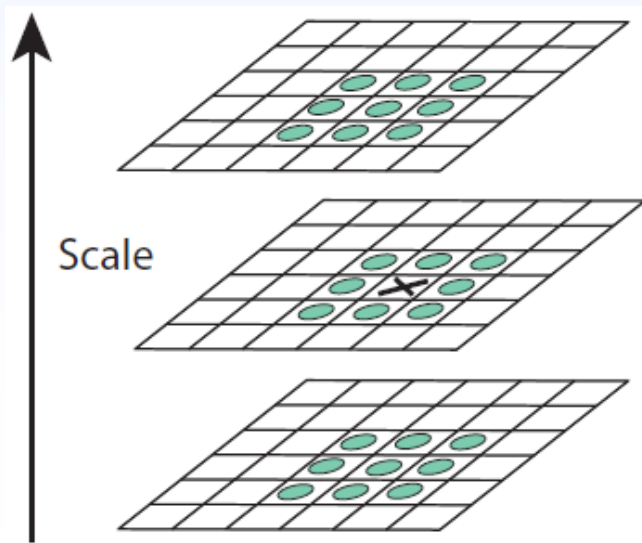
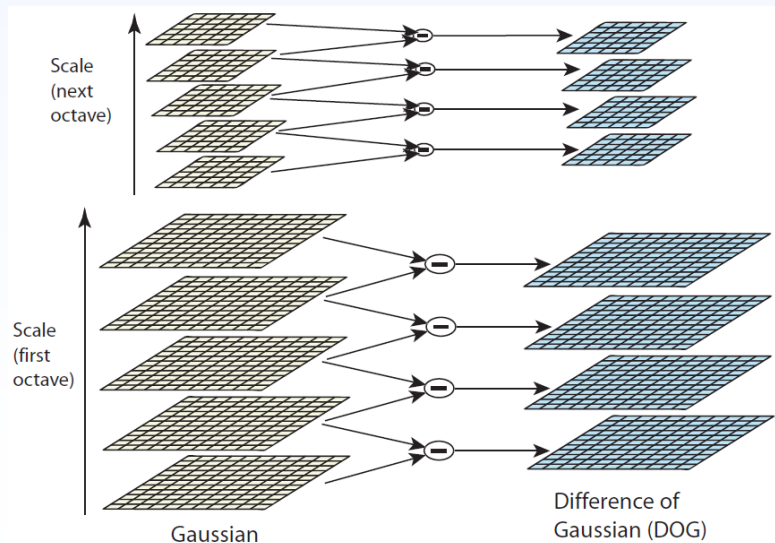
- 尺度空间不变性
- 像素迁移不变性
- 角度旋转不变性

SIFT特征提取步骤

- 尺度空间极值检测
- 关键点定位
- 方向指派
- 特征描述子

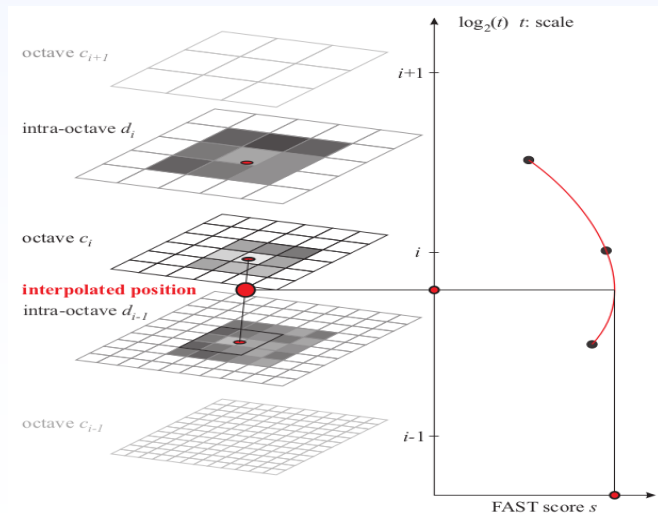
尺度空间极值检测

- 构建尺度空间 - 图像金字塔+高斯尺度空间
- 三层空间中的极值查找



关键点定位

- 极值点定位-求导拟合
- 删除低对比度与低响应候选点



$$\hat{x} = -\frac{\partial^2 H^{-1}}{\partial \mathbf{x}^2} \frac{\partial H}{\partial \mathbf{x}}$$

$$H(\mathbf{x}) = H + \frac{\partial H^T}{\partial \mathbf{x}} \mathbf{x} + \frac{1}{2} \mathbf{x}^T \frac{\partial^2 H}{\partial \mathbf{x}^2} \mathbf{x}$$

$$\frac{\partial^2 H}{\partial \mathbf{x}^2} = \begin{bmatrix} d_{xx} & d_{yx} & d_{sx} \\ d_{xy} & d_{yy} & d_{sy} \\ d_{xs} & d_{ys} & d_{ss} \end{bmatrix}$$

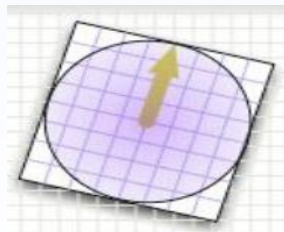
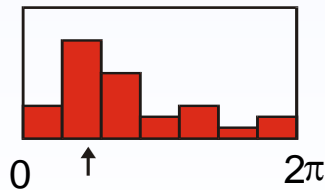
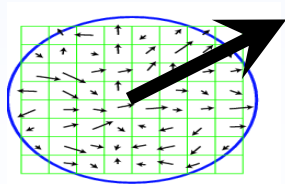
$$\frac{\partial H}{\partial \mathbf{x}} = \begin{bmatrix} d_x \\ d_y \\ d_s \end{bmatrix}.$$

方向指派

- 关键点方向指派
- Scale尺度最近的图像，1.5倍大小的高斯窗口

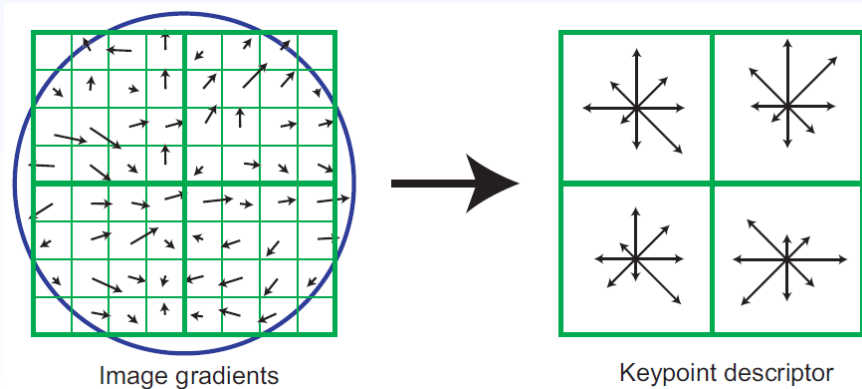
$$m(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2}$$

$$\theta(x, y) = \tan^{-1}((L(x, y+1) - L(x, y-1)) / (L(x+1, y) - L(x-1, y)))$$



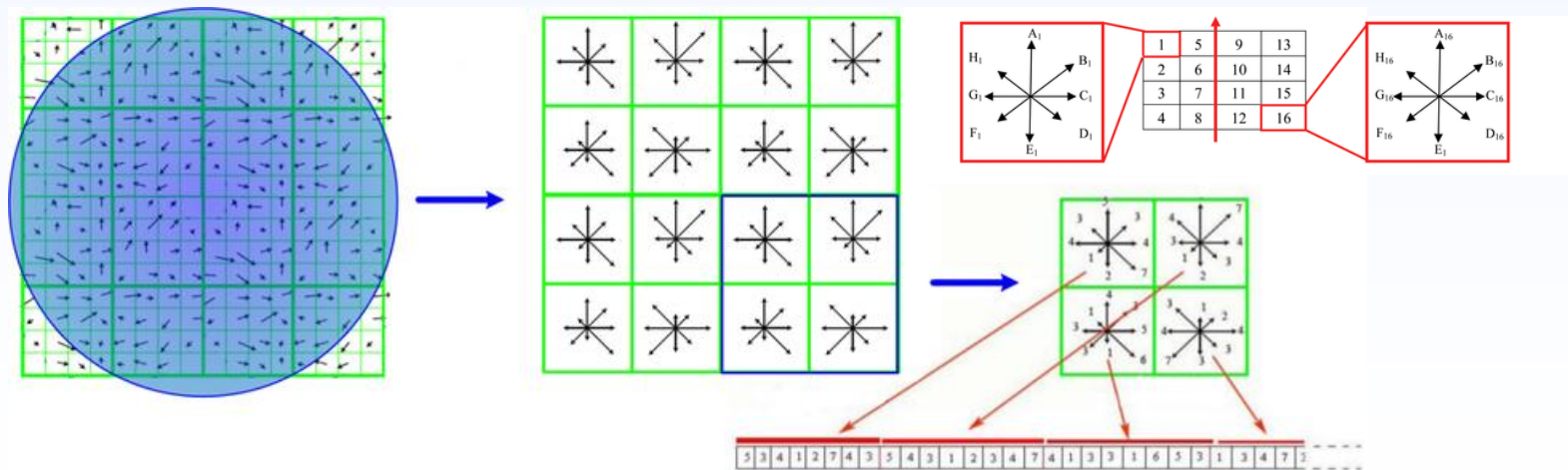
特征描述子

- 128维度向量/特征描述子
- 描述子编码方式



特征描述子

- 128维度向量/特征描述子
- 描述子编码方式





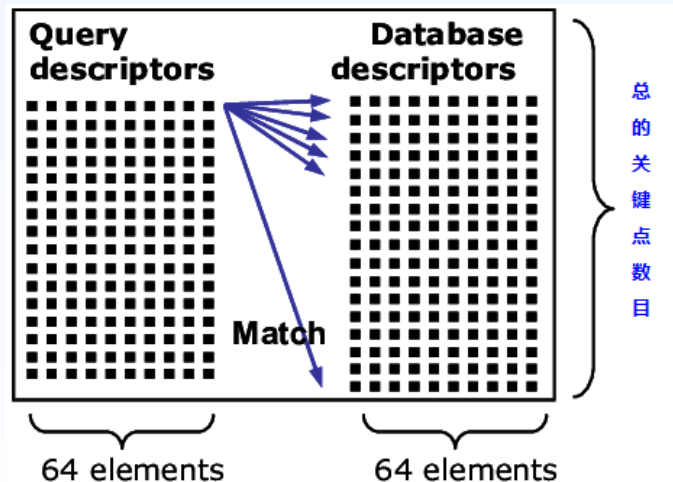
特征匹配

- 特征匹配算法
- 特征匹配函数
- 特征匹配方法对比

特征匹配算法

- 暴力匹配，全局搜索，计算最小距离，返回相似描述子合集
- FLANN匹配，2009发布的开源高维数据匹配算法库
- 全称为Fast Library for Approximate Nearest Neighbors
- 支持KMeans、KDTree、KNN、多探针LSH等搜索与匹配算法

描述子	匹配方法
SIFT, SURF, and KAZE	LI Norm
AKAZE, ORB, and BRISK.	Hamming distance

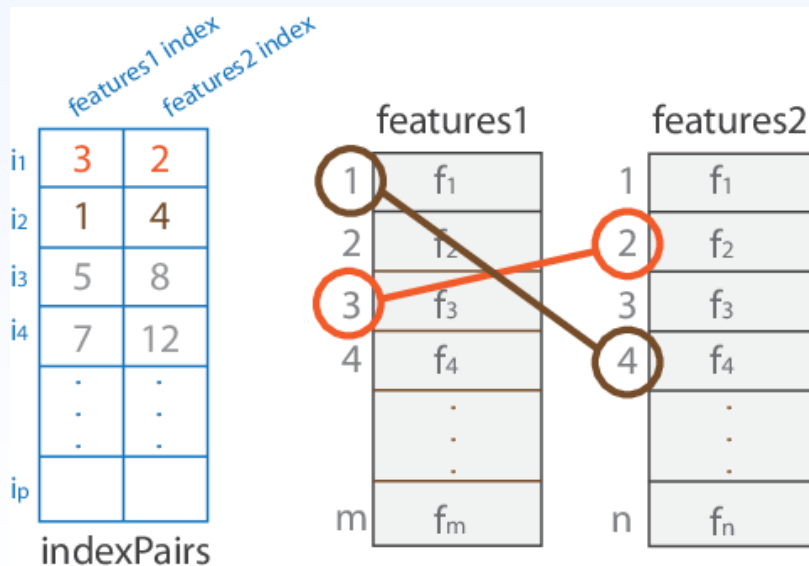


特征匹配DMatch数据结构

DMatch数据结构:

- queryIdx
- trainIdx
- distance

distance表示距离，值越小表示匹配程度越高



特征匹配函数

- 暴力匹配，全局搜索，计算最小距离，返回相似描述子合集
- FLANN匹配，2009发布的开源高维数据匹配算法库
- 全称为Fast Library for Approximate Nearest Neighbors
- 支持KMeans、KDTree、KNN、多探针LSH等搜索与匹配算法

```
// 暴力匹配
```

```
auto bfMatcher = BFMatcher::create(NORM_HAMMING, false);  
std::vector<DMatch> matches;  
bfMatcher->match(box_descriptors, scene_descriptors, matches);  
Mat img_orb_matches;  
drawMatches(box, box_kpts, box_in_scene, scene_kpts, matches, img_orb_matches);  
imshow("ORB暴力匹配演示", img_orb_matches);
```

```
// FLANN匹配
```

```
auto flannMatcher = FlannBasedMatcher(new flann::LshIndexParams(6, 12, 2));  
flannMatcher.match(box_descriptors, scene_descriptors, matches);  
Mat img_flann_matches;  
drawMatches(box, box_kpts, box_in_scene, scene_kpts, matches, img_flann_matches);  
namedWindow("FLANN匹配演示", WINDOW_FREERATIO);  
imshow("FLANN匹配演示", img_flann_matches);
```

OpenCV特征匹配方法对比



(a) SIFT Features (1907, 2209)



(b) Image Matching with SIFT



(d) SURF Features (3988, 4570)



(e) Image Matching with SURF



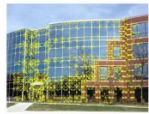
(g) KAZE Features (1291, 1359)



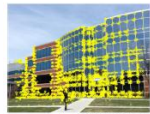
(h) Image Matching with KAZE



(j) AKAZE Features (1458, 1554)



(k) Image Matching with AKAZE



(m) ORB Features (5095, 5345)



(n) Image Matching with ORB

ORB>BRISK>SURF>SIFT>AKAZE>KAZE

The sequence of algorithms for computational efficiency of feature-detection-description per feature-point is:

**ORB>ORB(1000)>BRISK>BRISK(1000)>SURF(64D)
>SURF(128D)>AKAZE>SIFT>KAZE**

The order of efficient feature-matching per feature-point is:

**ORB(1000)>BRISK(1000)>AKAZE>KAZE>SURF(64D)
>ORB>BRISK>SIFT>SURF(128D)**

The feature-detector-descriptors can be rated for the speed of total image matching as:

**ORB(1000)>BRISK(1000)>AKAZE>KAZE>SURF(64D)
>SIFT>ORB>BRISK>SURF(128D)**

单应性变换/透视变换

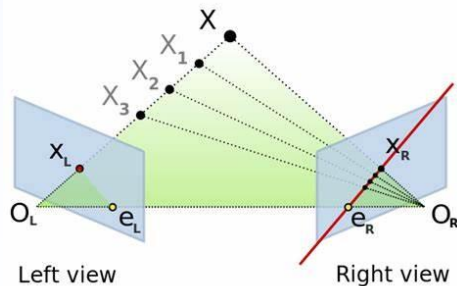


单应性变换/透视变换

- `Mat cv::findHomography(`
- `InputArray` `srcPoints, // 输入`
- `InputArray` `dstPoints, // 输出`
- `int` `method = 0,`
- `double` `ransacReprojThreshold = 3,`
- `OutputArray` `mask = noArray(),`
- `const int` `maxIters = 2000,`
- `const double` `confidence = 0.995`
- `)`

$$\sum_i \left(x'_i - \frac{h_{11}x_i + h_{12}y_i + h_{13}}{h_{31}x_i + h_{32}y_i + h_{33}} \right)^2 + \left(y'_i - \frac{h_{21}x_i + h_{22}y_i + h_{23}}{h_{31}x_i + h_{32}y_i + h_{33}} \right)^2$$

$$\begin{pmatrix} x'_1 \\ x'_2 \\ x'_3 \end{pmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$



拟合方法:

- 最小二乘 (O)
- 随机采样一致性 (RANSC)
- 渐进采样一致性 (RHO)



特征匹配案例学习

- 对象检测/发现
- 文档对齐
- 无缝拼接
- 标签与条码检测与匹配

对象检测与发现

Feature Detection & Description



Feature Matching



Outlier Rejection

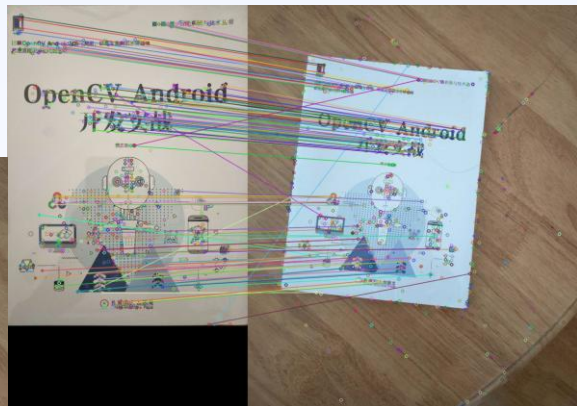
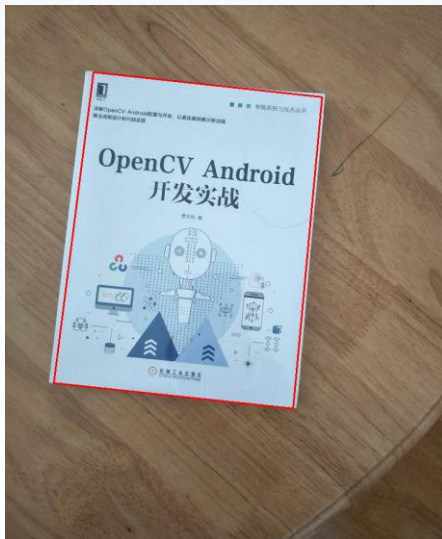


Derivation of Transformation Function



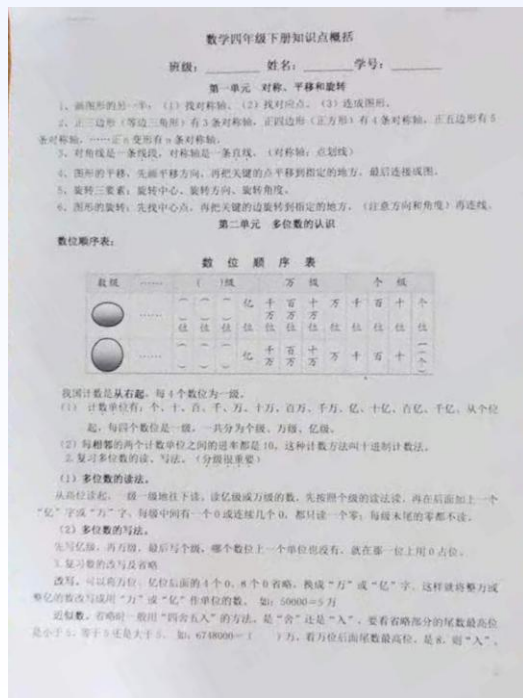
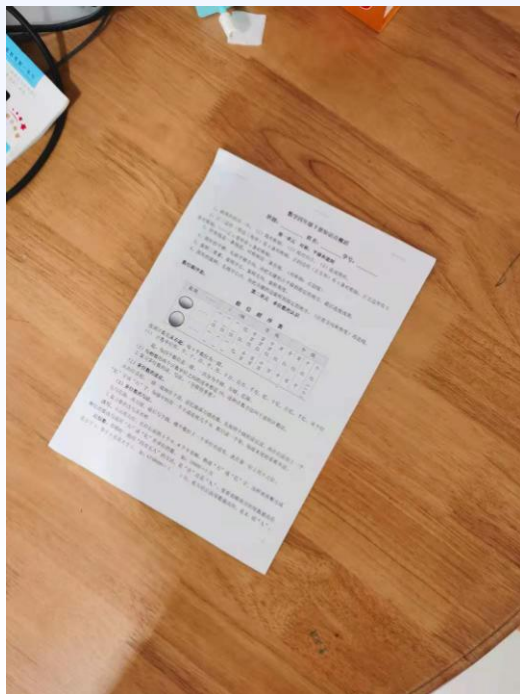
Image Reconstruction & Stitching

- 基于特征的匹配与对象检测
- ORB/AKAZE/SIFT
- 暴力/FLANN
- 透视变换
- 检测框



文档对齐

- 模板表单/文档
- 特征匹配与对齐



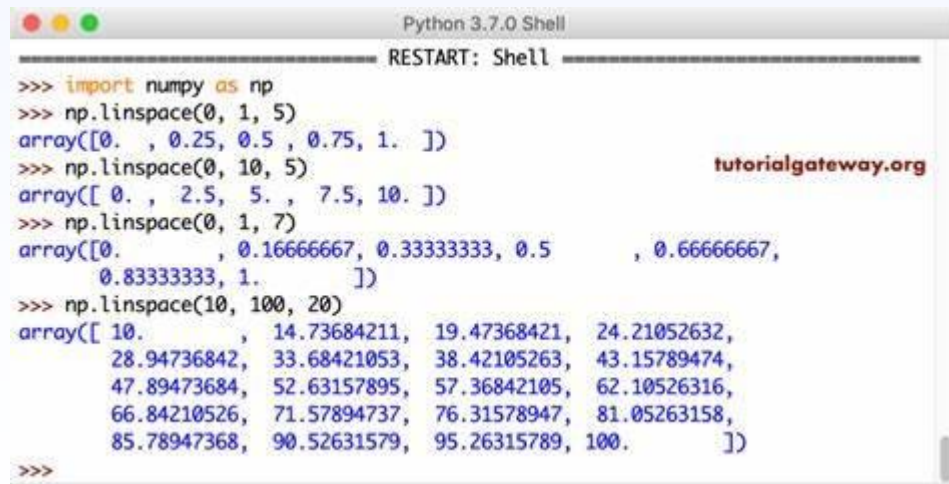
图像拼接

- 特征检测与匹配
- 图像对齐与变换
- 图像边缘融合



图像拼接

- Mask生成 - generateMask
- 梯度边缘融合 - linspace



```
Python 3.7.0 Shell
===== RESTART: Shell =====
>>> import numpy as np
>>> np.linspace(0, 1, 5)
array([0. , 0.25, 0.5 , 0.75, 1.  ])
>>> np.linspace(0, 10, 5)
array([ 0. , 2.5, 5. , 7.5, 10. ])
>>> np.linspace(0, 1, 7)
array([0. , 0.16666667, 0.33333333, 0.5 , 0.66666667,
        0.83333333, 1.  ])
>>> np.linspace(10, 100, 20)
array([ 10. , 14.73684211, 19.47368421, 24.21052632,
        28.94736842, 33.68421053, 38.42105263, 43.15789474,
        47.89473684, 52.63157895, 57.36842105, 62.10526316,
        66.84210526, 71.57894737, 76.31578947, 81.05263158,
        85.78947368, 90.52631579, 95.26315789, 100.  ])
>>>
```

实战：基于匹配的

一个工业级实战案例：

- 特征检测与匹配
- 图像对齐与变换
- 判断检查结果，分割标签





Thank You !