EDU

CSDN学院　IT实战派

# 计算机视觉

## 角点检测代码讲解

讲师：屈老师

# 本节目标

1. 学习Harris角点检测示例
2. 掌握使用角点检测和匹配的一般流程
3. 掌握SURF和ORB角点检测的实现

# Harris角点检测示例



## 相关函数

void cornerHarris(InputArray *src*, OutputArray *dst*,
int *blockSize*, int *ksize*, double *k*, int *borderType*=BORDER_DEFAULT );

- src，输入图像，即源图像，填Mat类的对象即可，且需为单通道8位或者浮点型图像。
- dst，即这个参数用于存放Harris角点检测的响应强度，和源图片有一样的尺寸和类型。
- blockSize，表示邻域的大小，更多的详细信息在cornerEigenValsAndVecs中有讲到。
- ksize，表示Sobel()算子的孔径大小。
- k，Harris参数。
- borderType，图像像素的边界模式，注意它有默认值BORDER_DEFAULT。更详细的解释，参考borderInterpolate函数。

# Harris角点检测示例

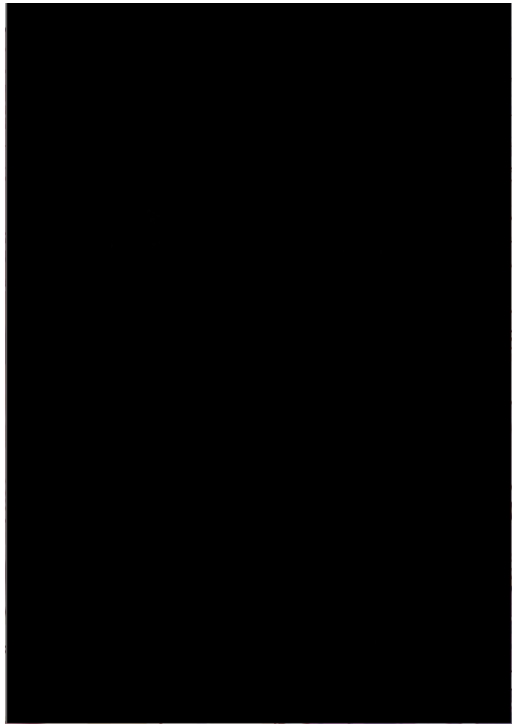
## cornerHarris 原始结果

OpenCV3中提供的函数并不能直接应用到你自己的代码之中
要想在一幅图之中框选出角点，我们还需要做一些其他的工作

首先，我们来看一下这个函数中产生的 OutputArray 是一个什么样的存在，我们以下面这张图片作为样例

由此可见，此函数我们需要进一步进行调整，才能达到想要的效果

原图



OutputArray

# 程序示例

为了使我们可以很好的使用这个函数，我们首先要对得到的结果进行下一步处理

此步处理中，我们消除掉了可能出现的角点聚集在一起的情况

```
//Harris 计算
cv::cornerHarris(image, cornerStrength,
    neighbourhood,// neighborhood size
    aperture,     // 孔径大小
    k);           // Harris 参数
cv::imshow("ini", cornerStrength);
//内部阈值计算
//配合函数minMaxLoc设置的变量，但后续并没有用处，只为配合函数使用
double minStrength;
cv::minMaxLoc(cornerStrength, &minStrength, &maxStrength);
// 局部最大值检测，使得减少出现角点聚集的情况出现
cv::Mat dilated;
cv::dilate(cornerStrength, dilated, cv::Mat());
cv::compare(cornerStrength, dilated, localMax, cv::CMP_EQ);
```

# Harris角点检测示例



## 程序示例

这一步处理中，我们将图像进一步处理，再一次削减角点数量，以便得到最正确的角点

并将角点强度所形成的图像进行返回，我们可以得到以下的效果

```
cv::Mat cornerMap;
// 对角点图像进行阈值化
threshold = qualityLevel*maxStrength;
cv::threshold(cornerStrength, cornerTh, threshold, 255, cv::THRESH_BINARY);
// 转换为8位图像
cv::Mat result;
cornerTh.convertTo(cornerMap, CV_8U);
cornerTh.convertTo(result, CV_8U);
// 非极大值抑制
cv::bitwise_and(cornerMap, localMax, cornerMap);

//cv::imshow("bitandbef", result);
cv::bitwise_and(result, cornerMap, result);
//cv::imshow("bitand", result);

/*std::cout << cornerMap.type() << "cornerMap_type" << std::endl;
std::cout << cornerTh.type() << "cornerTh_type" << std::endl;*/

return cornerMap;
```

# 程序示例

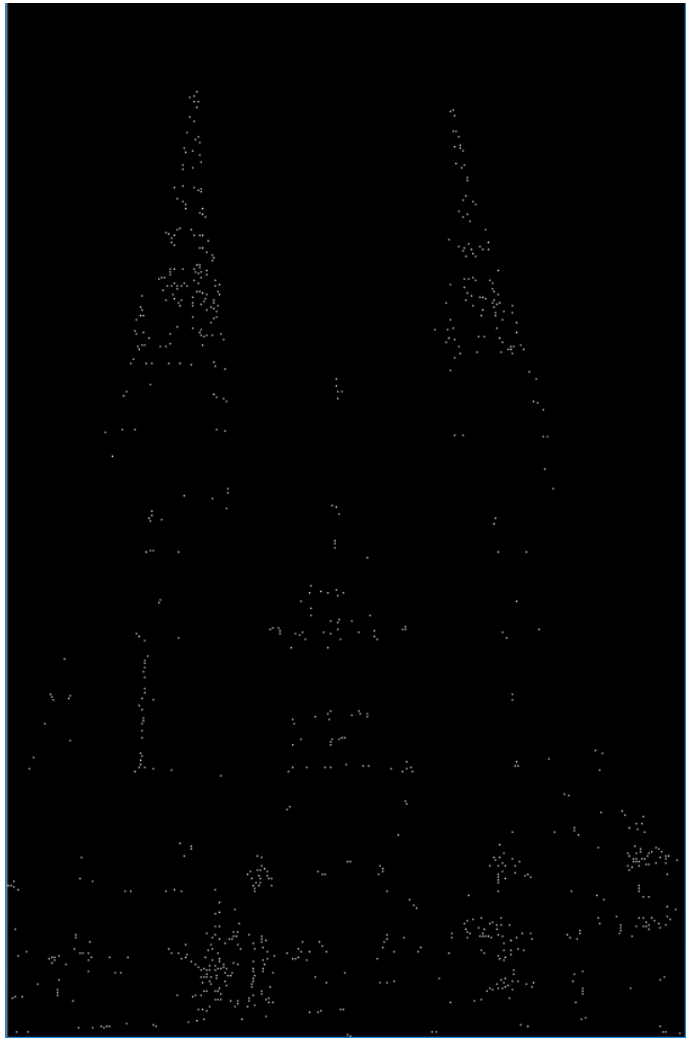
最后，我们将要把角点用□的方式画出，以便进行效果的测试

```
// 遍历像素得到所有特征
for (int y = 0; y < cornerMap.rows; y++) {
    const uchar* rowPtr = cornerMap.ptr<uchar>(y);
    for (int x = 0; x < cornerMap.cols; x++) {
        // 如果是特征点
        if (rowPtr[x])
        {
            points.push_back(cv::Point(x, y));
        }
    }
}
```

```
std::vector<cv::Point>::const_iterator it = points.begin();
// 对于所有角点
while (it != points.end())
{
    cv::circle(image, *it, radius, color, thickness);
    ++it;
}
```

# 运行效果

原图



处理后的图

最后效果图

# 角点检测一般流程

OpenCV3支持SIFT/SURF/ORB/KAZE/FAST/BRISK/AKAZE等角点检测，均为Feature2D的子类，
一般使用流程如下： **检测算子创建→检测→提取描述子→匹配**

**创建**

cv::Ptr<cv::算子> 给算子起的名字 = cv::算子::create(阈值);
e.g. cv::Ptr<cv::ORB> orb = cv::ORB::create(nkeypoint);

**检测**

上一步给算子起的名字-> detect(图片, 内容为KeyPoints的vector向量);
e.g. orb->detect(img_1, keypoints_1);

**提取描述子**

上一步给算子起的名字->compute(图片, 上一步中得到的向量, Mat类矩阵);
e.g. orb->compute(img_1, keypoints_1, descriptors_1);

**匹配**

BFMatcher类对象.match(描述子1,描述子2,内容为DMatch的vector向量);
e.g. matcher.match(descriptors_1, descriptors_2, mathces);

# Demo核心函数介绍

FAST角点检测算法在OpenCV3中以虚类的方式呈现，因此在使用上与SURF算法有一定的区别
我们在调用时一般使用如下方法

```
vector<cv::KeyPoint>keypoints;
cv::Ptr<cv::FeatureDetector> fast = cv::FastFeatureDetector::create(40);
```

之后，再使用OpenCV3中提供的角点检测函数，即可找到图片中的角点

```
fast->detect(image, keypoints);
cv::drawKeypoints(image, keypoints, image, cv::Scalar(255, 255, 255),
    cv::DrawMatchesFlags::DRAW_OVER_OUTIMG);
```

# Demo核心函数介绍

SURF算法是SIFT算法的高效变种，其计算速度远快于SIFT，因此我们在这里介绍SURF

CV_WRAP static Ptr<SURF> create(double *hessianThreshold*=100,
int *nOctaves* = 4, int *nOctaveLayers* = 3,
bool *extended* = false, bool *upright* = false);

在这个函数中，我们唯一需要了解的即为 hessianThreshold 这个参数
这个参数代表着 Hessisan 矩阵行列式所计算出的曲率强度
此数值越高，代表着区分匹配点的要求越高

当然，在OpenCV3中其给出了默认值100，不过一般推荐在1000~2500之间，如下面这样

```
Ptr<xfeatures2d::SURF> detector = xfeatures2d::SURF::create(minHessian);
Ptr<DescriptorExtractor> descriptor = xfeatures2d::SURF::create();
Ptr<DescriptorMatcher> matcher1 = DescriptorMatcher::create("BruteForce");
```

# 实战应用

在SURF算法中，为了执行匹配，我们首先检测待匹配的两幅图中的特征点

```
// 检测特征点
detector->detect(img1, keyPoint1);
detector->detect(img2, keyPoint2);
```

之后，我们提取描述这些特征点的描述子

```
// 提取特征点描述子
descriptor->compute(img1, keyPoint1, descriptors1);
descriptor->compute(img2, keyPoint2, descriptors2);
```

最后，我们将两幅图像中的描述子进行匹配

```
// 匹配图像中的描述子
matcher1->match(descriptors1, descriptors2, matches);
```

# 实战应用

在将我们的结果展示在电脑屏幕前，我们还需要进行最后一步
即为将对应的匹配点之间通过连线的方式画在图像上

为了使我们的结果看起来更加清晰，我们将匹配到的点的数量进行削减

```
std::nth_element(matches.begin(), matches.begin()+24, matches.end());
matches.erase(matches.begin()+25, matches.end());
```

之后，我们将相互之间能构成匹配的点画在图像上

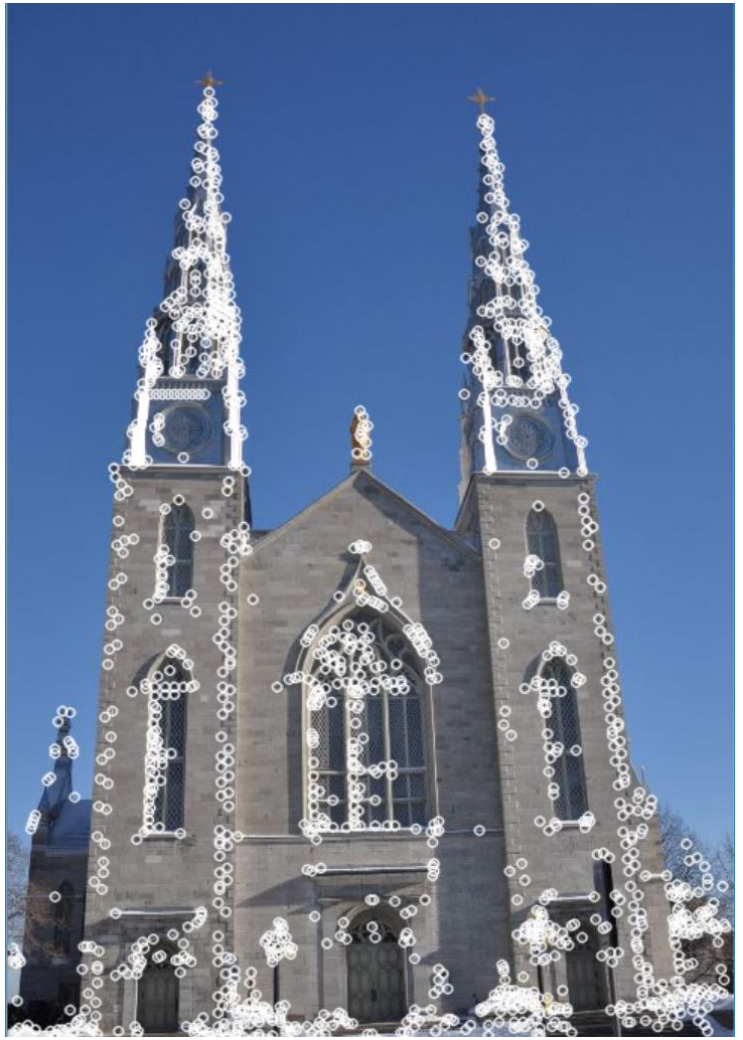
```
//画匹配点
Mat img_matches;
drawMatches(img1, keyPoint1, img2, keyPoint2, matches, img_matches);
imshow("img_matches", img_matches);
```

# OpenCV3实现 FAST

## 效果图

原图

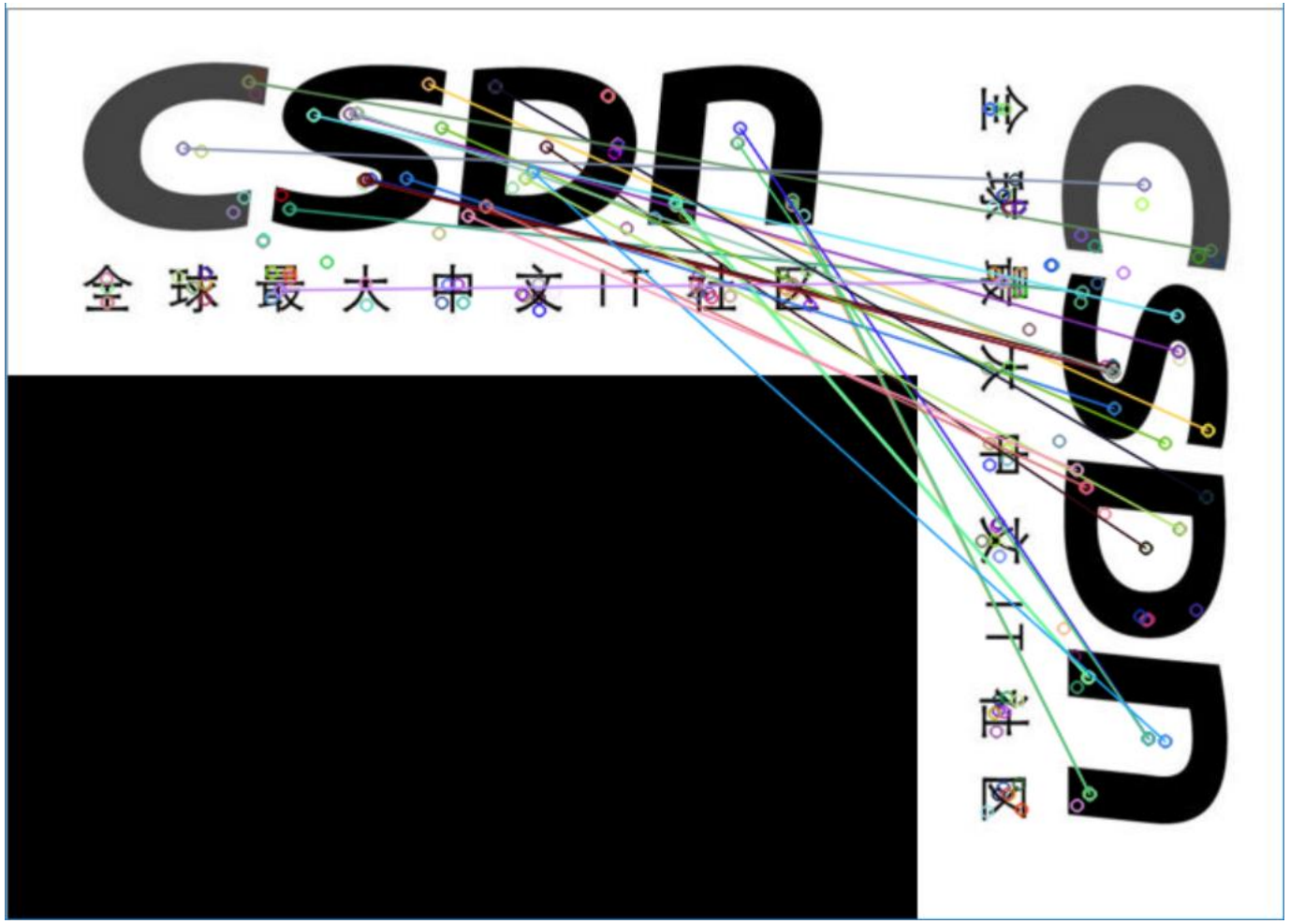
使用SURF



使用FAST

# 效果图


原图




旋转与匹配

# Demo核心函数介绍

与SURF的功能相同，ORB算法也是对两幅图片进行特征点匹配

CV_WRAP static Ptr<ORB> create(int *nfeatures*=500, float *scaleFactor*=1.2f,
int *nlevels*=8, int *edgeThreshold*=31,
int *firstLevel*=0, int *WTA*_K=2, int *scoreType*=ORB::HARRIS_SCORE,
int *patchSize*=31, int *fastThreshold*=20);

就像上述函数所声明的，在绝大多数情况下，我们需要改变的只是第一个参数 nfeatures
其代表了算法将会在图片中找到匹配点的对数

在OpenCV3中，我们可以如下使用这个函数

```
int nkeypoint = 50;
cv::Ptr<cv::ORB> orb = cv::ORB::create(nkeypoint);
```

# 实战应用

在ORB算法中，为了执行匹配，我们首先检测待匹配的两幅图中的特征点

```
//特征点检测
orb->detect(img_1, keypoints_1);
orb->detect(img_2, keypoints_2);
```

之后，我们提取描述这些特征点的描述子

```
//提取特征点描述子
orb->compute(img_1, keypoints_1, descriptors_1);
orb->compute(img_2, keypoints_2, descriptors_2);
```

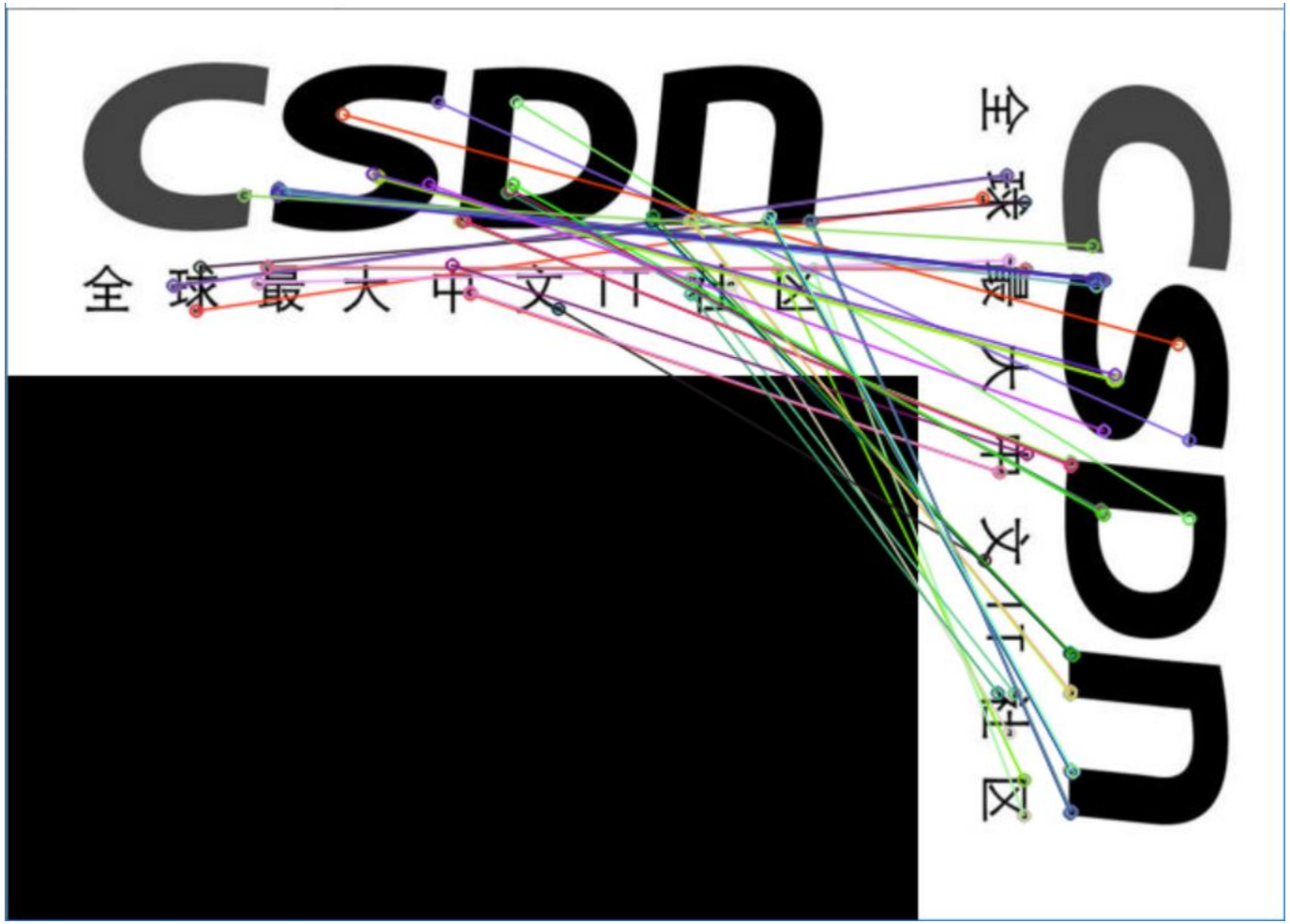
最后，我们将两幅图像中的描述子进行匹配

```
//根据描述子匹配特征点
matcher.match(descriptors_1, descriptors_2, mathces);
```

# 效果图

最后，再将匹配的点绘制在原图的坐标系上


原图




旋转与匹配

# 布置作业

1. 阅读提供示例程序，然后尝试自己编写SURF、ORB角点检测与匹配程序

EDU

CSDN学院　IT实战派