对OpenCV中的策略模式分析——以FeatureDetector为例

目录

[策略模式简介 2](#_Toc148716548)

[1.1 策略（Strategy）模式 2](#_Toc148716549)

[1.2 策略模式的结构 2](#_Toc148716550)

[OpenCV简介 3](#_Toc148716551)

[2.1 OpenCV介绍 3](#_Toc148716552)

[FeatureDetector简介 3](#_Toc148716553)

[3.1 FeatureDetector基类 3](#_Toc148716554)

[派生于FeatureDetector的具体的特征检测器 4](#_Toc148716555)

[4.1 Fast 4](#_Toc148716556)

[4.2 STAR 5](#_Toc148716557)

[4.3 SIFT 6](#_Toc148716558)

[4.4 SURF 7](#_Toc148716559)

[4.5 ORB 8](#_Toc148716560)

[4.6 MSER 9](#_Toc148716561)

[4.7 GFTT 10](#_Toc148716562)

[4.8 HARRIS 11](#_Toc148716563)

[4.9 Dense 12](#_Toc148716564)

[4.10 SimpleBlob 13](#_Toc148716565)

[类图 14](#_Toc148716566)

[5.1 FeatureDetector基类与其子类类图 14](#_Toc148716567)

# 策略模式简介

1.1 策略（Strategy）模式

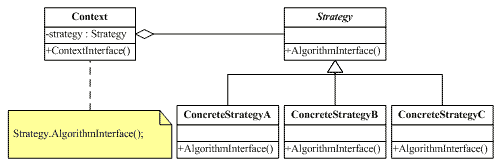
策略模式的用意是针对一组算法，将每一个算法封装到具有共同接口的独立的类中，从而使得它们可以相互替换。策略模式使得算法可以在不影响到客户端的情况下发生变化。

使用策略模式可以把行为和环境分割开来。环境类负责维持和查询行为类，各种算法则在具体策略类（ConcreteStrategy）中提供。由于算法和环境独立开来，算法的增减、修改都不会影响环境和客户端。当出现新的促销折扣或现有的折扣政策出现变化时，只需要实现新的策略类，并在客户端登记即可。策略模式相当于"可插入式（Pluggable）的算法"。

1.2 策略模式的结构

策略模式是对算法的包装，是把使用算法的责任和算法本身分割开，委派给不同的对象管理。策略模式通常把一个系列的算法包装到一系列的策略类里面，作为一个抽象策略类的子类。用一句话来说，就是："准备一组算法，并将每一个算法封装起来，使得它们可以互换。"

下面是一个示意性的策略模式结构图：



这个模式涉及到三个角色：

环境（Context）角色：持有一个Strategy类的引用。

抽象策略（Strategy）角色：这是一个抽象角色，通常由一个接口或抽象类实现。此角色给出所有的具体策略类所需的接口。

具体策略（ConcreteStrategy）角色：包装了相关的算法或行为。

# OpenCV简介

2.1 OpenCV介绍

OpenCV（Open Source Computer Vision Library）是一个开源计算机视觉库，它提供了丰富的图像处理和计算机视觉算法，可用于开发各种视觉应用和项目。OpenCV最初由英特尔（Intel）发起，后来成为一个开源项目，得到了广泛的支持和贡献。

# FeatureDetector简介

3.1 FeatureDetector基类

FeatureDetector是OpenCV中的一个2D 图像特征检测器的抽象基类，定义了特征检测器的通用接口和方法。具体的特征检测器算法会派生自这个基类，实现特定的特征检测算法，并提供其具体实现。这种设计使得 OpenCV 可以灵活支持多种特征检测算法，并让开发者可以根据需要选择适当的算法来处理图像。

class CV\_EXPORTS\_W FeatureDetector : public virtual Algorithm

{

public:

virtual ~FeatureDetector();

CV\_WRAP void detect( const Mat& image, CV\_OUT vector<KeyPoint>& keypoints, const Mat& mask=Mat() ) const;

void detect( const vector<Mat>& images, vector<vector<KeyPoint> >& keypoints, const vector<Mat>& masks=vector<Mat>() ) const;

CV\_WRAP virtual bool empty() const;

CV\_WRAP static Ptr<FeatureDetector> create( const string& detectorType );

protected:

virtual void detectImpl( const Mat& image, vector<KeyPoint>& keypoints, const Mat& mask=Mat() ) const = 0;

static void removeInvalidPoints( const Mat& mask, vector<KeyPoint>& keypoints );

};

# 派生于FeatureDetector的具体的特征检测器

4.1 Fast

FAST（Features from Accelerated Segment Test）是一种快速的特征点检测算法，用于在图像中寻找关键特征点。

class CV\_EXPORTS\_W FastFeatureDetector : public FeatureDetector

{

public:

enum

{ // Define it in old class to simplify migration to 2.5

TYPE\_5\_8 = 0, TYPE\_7\_12 = 1, TYPE\_9\_16 = 2

};

CV\_WRAP FastFeatureDetector( int threshold=10, bool nonmaxSuppression=true );

AlgorithmInfo\* info() const;

protected:

virtual void detectImpl( const Mat& image, vector<KeyPoint>& keypoints, const Mat& mask=Mat() ) const;

int threshold;

bool nonmaxSuppression;

};

void detect( const vector<Mat>& images, vector<vector<KeyPoint> >& keypoints, const vector<Mat>& masks=vector<Mat>() ) const;

CV\_WRAP virtual bool empty() const;

CV\_WRAP static Ptr<FeatureDetector> create( const string& detectorType );

protected:

virtual void detectImpl( const Mat& image, vector<KeyPoint>& keypoints, const Mat& mask=Mat() ) const = 0;

static void removeInvalidPoints( const Mat& mask, vector<KeyPoint>& keypoints );

};

enum{...}：定义了一个匿名枚举。这些枚举值代表FAST检测器的不同类型，基于它们使用的邻居数量来定义。例如，`TYPE\_5\_8`使用5个连续的像素（从8个周围像素中选取）来确定角点。

Virtual void detectImpl()，这是一个虚函数，用于检测图像中的关键点，其具体功能在派生类中实现，以提供具体的检测功能。

threshold表示用于角点检测的阈值。

nonmaxSuppression表示是否应用非最大值抑制。

void FastFeatureDetector::detectImpl( const Mat& image, vector<KeyPoint>& keypoints, const Mat& mask ) const

{

Mat grayImage = image;

if( image.type() != CV\_8U ) cvtColor( image, grayImage, CV\_BGR2GRAY );

FAST( grayImage, keypoints, threshold, nonmaxSuppression );

KeyPointsFilter::runByPixelsMask( keypoints, mask );

}

这段代码中实现了FAST角点检测的逻辑，以及对FAST检测器的扩展，使用FAST算法在grayImage中检测关键点（或角点）。其中grayImage表示源图像，keypoints是检测到的关键点的输出向量。Threshold是该FAST算法使用的阈值。强度变化大于此阈值的特征被视为角点。nonmaxSuppression是布尔标志。如果为真，该算法将使用非最大抑制来过滤那些彼此接近且在其局部不是最大值的关键点。

4.2 STAR

StarDetector（也称为CenSurE：Center Surround Extremas）是一种特征检测方法，专门用于在图像中检测角点或兴趣点。这些特征点通常在计算机视觉和图像处理任务中很有用，例如图像匹配、物体识别、全景图合成等。它同样继承于FeatureDetector。

void StarDetector::detectImpl( const Mat& image, vector<KeyPoint>& keypoints, const Mat& mask ) const

{

Mat grayImage = image;

if( image.type() != CV\_8U ) cvtColor( image, grayImage, CV\_BGR2GRAY );

(\*this)(grayImage, keypoints);

KeyPointsFilter::runByPixelsMask( keypoints, mask );

}

该函数接受三个参数：image（输入图像），keypoints（用于存储检测到的关键点），和mask（用于过滤关键点的遮罩）。首先初始化grayImage，并将图像转换为灰度:然后使用StarDetector对象（即\*this）来检测输入图像grayImage中的关键点，并将检测到的关键点保存在keypoints向量中。最后，调用KeyPointsFilter类中的runByPixelsMask函数，该函数使用提供的遮罩mask来过滤之前检测到的关键点。根据遮罩，某些关键点可能会保留，而其他关键点可能会被丢弃。

4.3 SIFT

是一个用于图像处理中的特征提取算法，全称是"Scale-Invariant Feature Transform"。它可以从图像中检测和描述局部特征，这些特征对于尺度、旋转、亮度变化等都是不变的。SIFT继承自Feature2D类，Feature2D类继承自FeatureDetector类和DescriptorExtractor类。组合了特征检测和特征描述的功能。

void buildGaussianPyramid( const Mat& base, vector<Mat>& pyr, int nOctaves ) const;

void buildDoGPyramid( const vector<Mat>& pyr, vector<Mat>& dogpyr ) const;

void findScaleSpaceExtrema( const vector<Mat>& gauss\_pyr, const vector<Mat>& dog\_pyr,

vector<KeyPoint>& keypoints ) const;

通过构建高斯金字塔（Gaussian pyramid）和差分金字塔（Difference of Gaussians，DoG pyramid）来实现在不同尺度下检测极值点，高斯金字塔通过在不同尺度下对图像进行高斯模糊来生成，DOG金字塔是相邻尺度下高斯图像相减得到的。对于每个尺度下的极值点，SIFT进行精确的关键点定位，保持稳定性。为实现旋转不变性，为每一个关键点分配一个反向。SIFT还会滤除质量低的关键点。SIFT的实现相对复杂，这里只讲这么多。

4.4 SURF

SURF（Speeded-Up Robust Features）是一种基于尺度空间的特征检测算法，它对图像的尺度和旋转变化具有不变性。

class CV\_EXPORTS\_W SURF : public Feature2D

{

public:

//! the default constructor

CV\_WRAP SURF();

//! the full constructor taking all the necessary parameters

explicit CV\_WRAP SURF(double hessianThreshold,

int nOctaves=4, int nOctaveLayers=2,

bool extended=true, bool upright=false);

//! returns the descriptor size in float's (64 or 128)

CV\_WRAP int descriptorSize() const;

//! returns the descriptor type

CV\_WRAP int descriptorType() const;

//! finds the keypoints using fast hessian detector used in SURF

void operator()(InputArray img, InputArray mask,

CV\_OUT vector<KeyPoint>& keypoints) const;

//! finds the keypoints and computes their descriptors. Optionally it can compute descriptors for the user-provided keypoints

void operator()(InputArray img, InputArray mask,

CV\_OUT vector<KeyPoint>& keypoints,

OutputArray descriptors,

bool useProvidedKeypoints=false) const;

AlgorithmInfo\* info() const;

、、、

protected:

void detectImpl( const Mat& image, vector<KeyPoint>& keypoints, const Mat& mask=Mat() ) const;

void computeImpl( const Mat& image, vector<KeyPoint>& keypoints, Mat& descriptors ) const;

};

SURF公开继承自Feature2D，首先构造函数创建了一个SURF特征检测器对象，指定了算法的各种参数。方法CV\_WRAP int descriptorSize() const和CV\_WRAP int descriptorType() const分别返回SURF描述符的大小和类型。之后的两个函数分别用于在图像上检测关键点和计算关键点的描述符。第一个函数仅检测关键点，而第二个函数还可以计算描述符。CV\_PROP\_RW 宏用于定义参数的可读写属性。protected下的两个函数computeImpl和detectImpl实现了SURF算法关键点检测和描述符计算的具体逻辑，供公共接口方法调用。

**4.5** ORB

ORB（Oriented FAST and Rotated BRIEF）是一种基于FAST和BRIEF的特征检测算法，它具有旋转不变性。

class CV\_EXPORTS\_W ORB : public Feature2D

{

public:

// the size of the signature in bytes

enum { kBytes = 32, HARRIS\_SCORE=0, FAST\_SCORE=1 };

CV\_WRAP explicit ORB(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 );

// returns the descriptor size in bytes

int descriptorSize() const;

// returns the descriptor type

int descriptorType() const;

// Compute the ORB features and descriptors on an image

void operator()(InputArray image, InputArray mask, vector<KeyPoint>& keypoints) const;

// Compute the ORB features and descriptors on an image

void operator()( InputArray image, InputArray mask, vector<KeyPoint>& keypoints,

OutputArray descriptors, bool useProvidedKeypoints=false ) const;

AlgorithmInfo\* info() const;

protected:

void computeImpl( const Mat& image, vector<KeyPoint>& keypoints, Mat& descriptors ) const;

void detectImpl( const Mat& image, vector<KeyPoint>& keypoints, const Mat& mask=Mat() ) const;

、、、

};

ORB公开继承自Feature2D，首先定义了一个枚举，其中kBytes表示ORB特征描述子的大小为32字节。然后是构造ORB对象，其中包含了ORB算法的各种参数。函数int descriptorSize() const 和 int descriptorType() const返回ORB描述子的大小和类型。之后的两个函数用于计算输入图像中的ORB特征点。int descriptorSize() const返回该算法信息。protected 下的两个函数 computeImpl 和 detectImpl 实现了ORB特征的计算和检测，供公共接口方法调用。CV\_PROP\_RW 宏用于定义可读写的属性。

4.6 MSER

MSER（Maximally Stable Extremal Regions）是一种特征检测算法，用于检测图像中的稳定极值区域，这些区域通常对应于图像中的目标或物体的区域。

class CV\_EXPORTS\_W MSER : public FeatureDetector

{

public:

//! the full constructor

CV\_WRAP explicit MSER( int \_delta=5, int \_min\_area=60, int \_max\_area=14400,

double \_max\_variation=0.25, double \_min\_diversity=.2,

int \_max\_evolution=200, double \_area\_threshold=1.01,

double \_min\_margin=0.003, int \_edge\_blur\_size=5 );

//! the operator that extracts the MSERs from the image or the specific part of it

CV\_WRAP\_AS(detect) void operator()( const Mat& image, CV\_OUT vector<vector<Point> >& msers,

const Mat& mask=Mat() ) const;

AlgorithmInfo\* info() const;

protected:

void detectImpl( const Mat& image, vector<KeyPoint>& keypoints, const Mat& mask=Mat() ) const;

int delta;

int minArea;

int maxArea;

double maxVariation;

double minDiversity;

int maxEvolution;

double areaThreshold;

double minMargin;

int edgeBlurSize;

};

构造函数初始化MSER各项参数进而控制检测过程的行为。operator() 函数是用于从图像中提取 MSER 区域的操作符。它接受输入图像、MSER 区域的容器 msers 和可选的掩码图像 mask，并将检测到的 MSER 区域存储在 msers 中。通过detectImpl实现实际的MSER检测，检测稳定的极值区域。

4.7 GFTT

GFTT（Good Features to Track）是一种特征检测算法，用于在图像中寻找适合跟踪的良好特征点。它旨在检测那些在不同图像帧之间具有良好可跟踪性的特征点。

class CV\_EXPORTS\_W GFTTDetector : public FeatureDetector

{

public:

CV\_WRAP GFTTDetector( int maxCorners=1000, double qualityLevel=0.01, double minDistance=1,

int blockSize=3, bool useHarrisDetector=false, double k=0.04 );

AlgorithmInfo\* info() const;

protected:

virtual void detectImpl( const Mat& image, vector<KeyPoint>& keypoints, const Mat& mask=Mat() ) const;

int nfeatures;

double qualityLevel;

double minDistance;

int blockSize;

bool useHarrisDetector;

double k;

};

通过构造函数可以设置GFTT检测的一些初始参数，如：maxCorners用于限定检测到的点数的最大值、qualityLevel表示检测到的角点的质量水平（通常是0.10到0.01之间的数值，不能大于1.0）等等。

GFTT 通常用于检测图像中的角点。它通过计算每个像素处的角点响应值来确定角点的位置。在角点检测后，可以对检测到的角点应用非极大值抑制，以剔除冗余的角点。

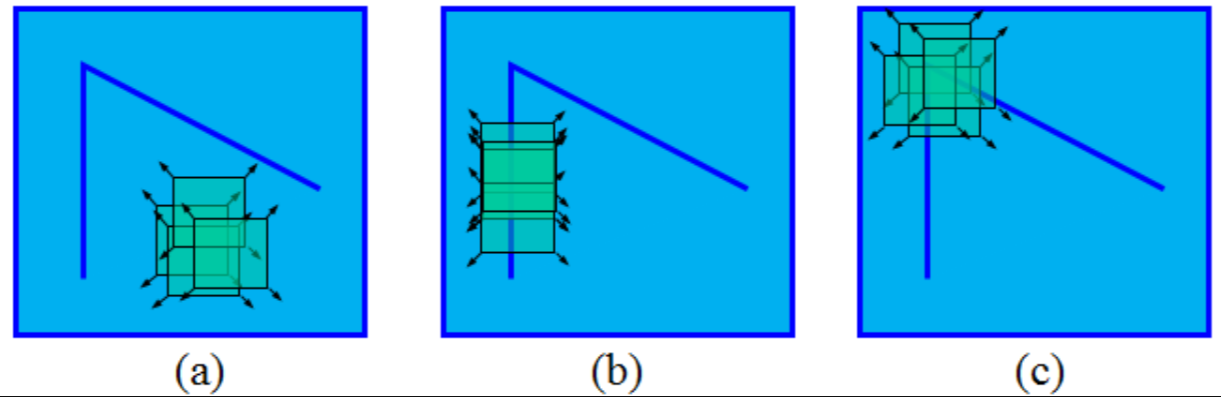
detectImpl()是一个虚拟方法，用于执行GFTT角点检测算法，将检测到的角点转化为关键点（KeyPoint），并将这些关键点存储在 keypoints 向量中。

4.8 HARRIS

Harris 角点检测方法是一种经典的角点检测技术，它通过计算结构矩阵中的特征值来确定角点位置。Harris 角点检测通常用于检测具有局部强度变化的角点，适合用于目标跟踪和图像拼接等任务。

这个检测器在 OpenCV 中同样由 GoodFeaturesToTrackDetector 类实现，但在参数设置上会启用 Harris 角点检测。

当一个窗口在图像上移动，在平滑区域如图(a)，窗口在各个方向上没有变化。在边缘上如图(b)，窗口在边缘的方向上没有变化。在角点处如图(c)，窗口在各个方向上具有变化。Harris角点检测正是利用了这个直观的物理现象，通过窗口在各个方向上的变化程度，决定是否为角点。



4.9 Dense

是 OpenCV 中的一个特征检测器类，用于检测密集的特征点，通常用于密集光流和密集匹配等计算机视觉任务。把输入图像分割成大小相等的网格，每一个网格提取一个像素作为特征点。类似于图像尺度金字塔，该方法也可以生成不同层图像的特征点，每一层图像所分割的网格大小是不同的，即表示各层的尺度不同。

class CV\_EXPORTS DenseFeatureDetector : public FeatureDetector

{

public:

explicit DenseFeatureDetector( float initFeatureScale=1.f, int featureScaleLevels=1,

float featureScaleMul=0.1f,

int initXyStep=6, int initImgBound=0,

bool varyXyStepWithScale=true,

bool varyImgBoundWithScale=false );

AlgorithmInfo\* info() const;

protected:

virtual void detectImpl( const Mat& image, vector<KeyPoint>& keypoints, const Mat& mask=Mat() ) const;

double initFeatureScale;

int featureScaleLevels;

double featureScaleMul;

int initXyStep;

int initImgBound;

bool varyXyStepWithScale;

bool varyImgBoundWithScale;

};

通过构造函数设置参数，如：initFeatureScale表示初始图像层特征点的尺度，默认为1、featureScaleLevels表示需要构建多少层图像，默认为1等等。

通过detectImpl实际实现检测，遍历当前层所有网格、把网格的左上角坐标处的像素作为该网格的特征点，并保存。调整下一层图像特征点的参数，再进行掩码矩阵的特征点处理。

4.10 SimpleBlob

SimpleBlobDetector 用于检测明亮的小斑点，这些斑点通常具有与周围区域不同的亮度或颜色。它可以检测圆形斑点，如二值图像中的白色圆圈或黑色斑点。

class CV\_EXPORTS\_W SimpleBlobDetector : public FeatureDetector

{

public:

struct CV\_EXPORTS\_W\_SIMPLE Params

{

CV\_WRAP Params();

CV\_PROP\_RW float thresholdStep;

CV\_PROP\_RW float minThreshold;

CV\_PROP\_RW float maxThreshold;

CV\_PROP\_RW size\_t minRepeatability;

CV\_PROP\_RW float minDistBetweenBlobs;

CV\_PROP\_RW bool filterByColor;

CV\_PROP\_RW uchar blobColor;

CV\_PROP\_RW bool filterByArea;

CV\_PROP\_RW float minArea, maxArea;

CV\_PROP\_RW bool filterByCircularity;

CV\_PROP\_RW float minCircularity, maxCircularity;

CV\_PROP\_RW bool filterByInertia;

CV\_PROP\_RW float minInertiaRatio, maxInertiaRatio;

CV\_PROP\_RW bool filterByConvexity;

CV\_PROP\_RW float minConvexity, maxConvexity;

void read( const FileNode& fn );

void write( FileStorage& fs ) const;

};

CV\_WRAP SimpleBlobDetector(const SimpleBlobDetector::Params &parameters = SimpleBlobDetector::Params());

virtual void read( const FileNode& fn );

virtual void write( FileStorage& fs ) const;

protected:

struct CV\_EXPORTS Center

{

Point2d location;

double radius;

double confidence;

};

virtual void detectImpl( const Mat& image, vector<KeyPoint>& keypoints, const Mat& mask=Mat() ) const;

virtual void findBlobs(const Mat &image, const Mat &binaryImage, vector<Center> &centers) const;

Params params;

AlgorithmInfo\* info() const;

};

这个算法的步骤大概如下：先设置一个低阈值（minThreshold）和高阈值（maxThreshold），用每一个阈值对图像进行二值化，得到一系列图像，对于每个二值图片查照这些图像的边，计算每一个轮廓的中心，将每一个二值图片的特征点放在一起，定义最小距离，在这个距离区域内的特征中心点归为一组，对应一个bolb特征，得到点的集合。估计最后的bolb特征和相应半径，返回keypoints。

## 类图

5.1 FeatureDetector基类与其子类类图