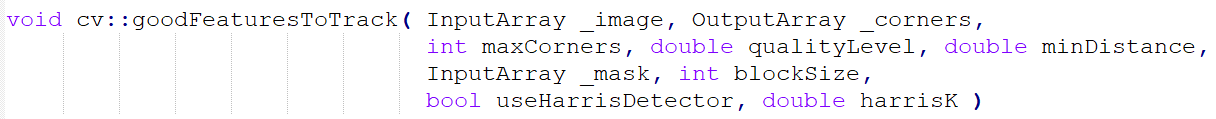
本文主要描述opencv中强角点检测接口goodFeaturesToTrack的用途及源码分析。

**1、接口描述**

在opencv中，该接口的实现是在：

..\sources\modules\imgproc\src\Featureselect.cpp 中。



|  |  |  |
| --- | --- | --- |
| goodFeaturesToTrack | 参数 | 意义 |
| InputArray \_image | 8bits 图像或者32位浮点型数图像，单通道 |
| OutputArray \_corners | 检测到的角点输出向量 |
| int maxCorners | 可检测的角点的最大值，如果检测的角点多于此数目，返回最强的那些角点。 |
| double qualityLevel | 表征图像角点的最小可接受的质量参数。参数值乘以最好的角点的质量(即最小的特征值，见cornerMinEigenVal)，或Harris响应值(见 cornerHarris)。角点质量小于上述乘积就会被拒绝。例如，最好的角点质量为1500，而qualityLevel=0.01，所以所有质量小于15=1500\*0.01的角点就会被拒绝。 |
|  | double minDistance | 决定返回角点的最小的欧式距离 |
|  | InputArray \_mask | 可选参数，region of interest. 如果 mask 为 NULL，则选择整个图像。 |
|  | int blockSize | 是协方差矩阵滤波的窗口大小(见 cornerEigenValsAndVecs) |
|  | bool useHarrisDetector | 使用Harris检测或cornerMinEigenVal |
|  | double harrisK | Harris角点检测特征表达式中的常数k值 |

测试代码如下

#include "opencv2/core/core.hpp"

#include "opencv2/highgui/highgui.hpp"

#include "opencv2/imgproc/imgproc.hpp"

#include <vector>

using namespace cv;

using namespace std;

Mat image;

Mat imageGray;

int thresh = 50; //角点个数控制

int MaxThresh = 255;

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

{

image = imread("1.jpeg",1);

cvtColor(image, imageGray, CV\_RGB2GRAY);

GaussianBlur(imageGray, imageGray, Size(5, 5), 1); // 滤波

Mat dst = Mat::zeros(image.size(), CV\_32FC1);

vector<Point2f> corners;

goodFeaturesToTrack(imageGray, corners, thresh, 0.01, 10, Mat());

//画特征点

for (int i = 0; i<corners.size(); i++)

circle(image, corners[i], 2, Scalar(0, 0, 255), 2);

imshow("Corner Detected", image);

waitKey();

return 0;

}

**在函数内部，大体的处理流程如下：**

1）角点检测。可以通过harris或者shi-tomasi算法检测角点，获得eig特征值集合。

2）非极大值抑制。低于qualityLevel\*特征值最大值的特征值被滤除；并且通过与膨胀后的特征值进行对比，局部最大值也会被滤除。

3）距离检测。挑选出符合距离要求的特征点。

void cv::goodFeaturesToTrack( InputArray \_image, OutputArray \_corners,

int maxCorners, double qualityLevel, double minDistance,

InputArray \_mask, int blockSize,

bool useHarrisDetector, double harrisK )

{

CV\_Assert( qualityLevel > 0 && minDistance >= 0 && maxCorners >= 0 );

CV\_Assert( \_mask.empty() || (\_mask.type() == CV\_8UC1 && \_mask.sameSize(\_image)) );

CV\_OCL\_RUN(\_image.dims() <= 2 && \_image.isUMat(),

ocl\_goodFeaturesToTrack(\_image, \_corners, maxCorners, qualityLevel, minDistance,

\_mask, blockSize, useHarrisDetector, harrisK))

Mat image = \_image.getMat(), eig, tmp;

if (image.empty())

{

\_corners.release();

return;

}

if( useHarrisDetector )

cornerHarris( image, eig, blockSize, 3, harrisK );

else

cornerMinEigenVal( image, eig, blockSize, 3 );

double maxVal = 0;

minMaxLoc( eig, 0, &maxVal, 0, 0, \_mask );

threshold( eig, eig, maxVal\*qualityLevel, 0, THRESH\_TOZERO );

dilate( eig, tmp, Mat());

Size imgsize = image.size();

std::vector<const float\*> tmpCorners;

// collect list of pointers to features - put them into temporary image

Mat mask = \_mask.getMat();

for( int y = 1; y < imgsize.height - 1; y++ )

{

const float\* eig\_data = (const float\*)eig.ptr(y);

const float\* tmp\_data = (const float\*)tmp.ptr(y);

const uchar\* mask\_data = mask.data ? mask.ptr(y) : 0;

for( int x = 1; x < imgsize.width - 1; x++ )

{

float val = eig\_data[x];

if( val != 0 && val == tmp\_data[x] && (!mask\_data || mask\_data[x]) )

tmpCorners.push\_back(eig\_data + x);

}

}

std::sort( tmpCorners.begin(), tmpCorners.end(), greaterThanPtr() );

std::vector<Point2f> corners;

size\_t i, j, total = tmpCorners.size(), ncorners = 0;

if (minDistance >= 1)

{

// Partition the image into larger grids

int w = image.cols;

int h = image.rows;

const int cell\_size = cvRound(minDistance);

const int grid\_width = (w + cell\_size - 1) / cell\_size;

const int grid\_height = (h + cell\_size - 1) / cell\_size;

std::vector<std::vector<Point2f> > grid(grid\_width\*grid\_height);

minDistance \*= minDistance;

for( i = 0; i < total; i++ )

{

int ofs = (int)((const uchar\*)tmpCorners[i] - eig.ptr());

int y = (int)(ofs / eig.step);

int x = (int)((ofs - y\*eig.step)/sizeof(float));

bool good = true;

int x\_cell = x / cell\_size;

int y\_cell = y / cell\_size;

int x1 = x\_cell - 1;

int y1 = y\_cell - 1;

int x2 = x\_cell + 1;

int y2 = y\_cell + 1;

// boundary check

x1 = std::max(0, x1);

y1 = std::max(0, y1);

x2 = std::min(grid\_width-1, x2);

y2 = std::min(grid\_height-1, y2);

for( int yy = y1; yy <= y2; yy++ )

for( int xx = x1; xx <= x2; xx++ )

{

std::vector <Point2f> &m = grid[yy\*grid\_width + xx];

if( m.size() )

{

for(j = 0; j < m.size(); j++)

{

float dx = x - m[j].x;

float dy = y - m[j].y;

if( dx\*dx + dy\*dy < minDistance )

{

good = false;

goto break\_out;

}

}

}

}

break\_out:

if (good)

{

grid[y\_cell\*grid\_width + x\_cell].push\_back(Point2f((float)x, (float)y));

corners.push\_back(Point2f((float)x, (float)y));

++ncorners;

if( maxCorners > 0 && (int)ncorners == maxCorners )

break;

}

}

}

else

{

for( i = 0; i < total; i++ )

{

int ofs = (int)((const uchar\*)tmpCorners[i] - eig.ptr());

int y = (int)(ofs / eig.step);

int x = (int)((ofs - y\*eig.step)/sizeof(float));

corners.push\_back(Point2f((float)x, (float)y));

++ncorners;

if( maxCorners > 0 && (int)ncorners == maxCorners )

break;

}

}

Mat(corners).convertTo(\_corners, \_corners.fixedType() ? \_corners.type() : CV\_32F);

}