OpenCV

Wang Ruchen

May 29, 2015

目录

1. 安装

1. 进入解压后的 opencv3.0 目录

```
mkdir build
```

2. 进入 build 目录

```
cd build
cmake -D CMAKE_BUILD_TYPE=Release -D CMAKE_INSTALL_PREFIX=/usr/local ..
make
sudo make install
```

2. core

核心功能模块，包含如下内容：

- OpenCV 基本数据结构
- 动态数据结构
- 绘图函数

- 数组操作相关函数
- 辅助功能与系统函数和宏
- 与 OpenGL 的互操作

2.1 基本数据类型

2.1.1 Point_

- Point_：定义二维坐标 x,y
  - Point Point2d(Point_<double>) Point2f(Point_<float>) Point2i(Point_<int>)
    e.g.：Point2i a(x,y);

2.1.2 Point3_

- Point3_：定义三维坐标 x,y,z

2.1.3 Size_

- Size_：定义图片或矩阵的大小，包含 width 和 height 两个元素。

2.1.4 Rect_

- Rect_：定义二维图片或矩阵，包含元素 x,y,width,height。x,y 为起始坐标，width,height 为矩形的长宽。

2.1.5 RotatedRect

- RotatedRect：旋转矩形，包含元素 center,size,angle,box。

```
1 RotatedRect(const Point2f& center, const Size2f& size, float angle)
```

  - center：矩形的中心坐标
  - size：矩形的长宽
  - angle：顺时针旋转的角度

```
RotatedRect(const Point2f& point1, const Point2f& point2, const Point2f& point3)
```

– point1,2,3　旋转后三点的坐标，按照顺时针或者逆时针的方向给出。

```
points(Point2f pts[ ])
```

– 点数组来存储矩形顶点

```
 boundingRect()
```

计算点集的最外面矩形边界

2.1.6 Matx

定义大小、数据类型已知的矩阵

- Matx12f，Matx12d···

2.1.7 Vec

短向量，基于 Matx。

- Vec2b　Vec<uchar,2>
- Vec2s　Vec<short,2> ···
- Vec2f，Vec2i，Vec2b

2.1.8 Range

范围

- 包括两个元素 start，end

2.1.9 Mat

矩阵结构

- M.data 数据区域的指针
- M.dims 矩阵维度
- M.sizes 维度
- M.elemSize() 每个元素占的字节空间大小，与元素类型相关，如 CV_8U
- M.step[] 用来计算元素地址，M.step[i] 表示所有比 i 大的维度所占空间大小

创建特殊矩阵

- diag 提取指定矩阵的对角线

```
1    Mat::diag(const cv::Mat &d)
```

- ones

```
1    Mat::ones(int rows, int cols, int type)
```

- zeros

```
1    Mat::zeros(int rows, int cols, int type)
```

- eye

```
1    Mat::eye(int rows, int cols, int type)
```

相关属性

- rows 行数　row(i)

- cols 列数　col(i)
- begin 返回矩阵迭代器，并将其设置为第一矩阵元
- end 返回矩阵迭代器，并将其设置为最后元素之后的矩阵元
- at 返回对指定数组元素的引用
- size 替代矩阵大小规格 Size(cols, rows) 的方法

```
1    Mat::size() const
```

- depth 返回一个矩阵元素的深度
- type 矩阵的类型

```
1    Mat::type() const
```

- elemSize 返回矩阵元素大小 (以字节为单位)

```
1    Mat::elemSize() const
```

- total 返回数组的总数

```
1    Mat::total() const
```

矩阵操作

- t 转置矩阵

```
1    Mat::t() const
```

- inv 反转矩阵

```
1    Mat::inv(int method=DECOMP_LU) const
```

- mul 执行两个矩阵按元素相乘或这两个矩阵的除法

```
1    Mat::mul(InputArray m, double scale=1) const
```

- cross 计算 3 元素向量的一个叉乘积
- dot 计算两向量的点乘

```
1    Mat::dot(InputArray m) const
```

- reshape 改变矩阵的通道数量，比如把二维矩阵改变成一维矩阵

```
1    Mat::reshape(int cn)
```

- resize 更改矩阵的行数

```
1    Mat::resize(size_t sz, const Scalar& s)
```

  - sz　新的行数
  - s　分配给新添加的元素的值

- reserve 保留一定行的数量

```
1    Mat::reserve(size_t sz)
```

  - sz　行数

- push_back 将元素添加到矩阵底部

```
Mat::push_back(const Mat& elem)
```

– elem 增加的一个或多个元素

- pop_back 从底部的列表删除元素

```
Mat::pop_back(size_t nelems=1)
```

– nelems 删除的行的数目

赋值关系

- clone

```
cv::Mat F = M.clone();
```

- copyTo

```
cv::Mat G;
F.copyTo(G);
```

- convertTo

```
convertTo ( OutputArray m , int rtype , double alpha =1, double beta =0 )
```

生成一个新矩阵，矩阵的中值为原矩阵中的值乘以 alpha，然后再加上 beta

- assignTo
- setTo

3. class

3.1 KeyPoint

Keypoint 是 opencv 中的一个类，就是一个关键点，这个关键点存放了很多信息，同时也说明：想获得一个关键点，需要很多信息来初始化它。

- angle：角度，表示关键点的方向，通过 Lowe 大神的论文可以知道，为了保证方向不变形，SIFT 算法通过对关键点周围邻域进行梯度运算，求得该点方向。-1 为初值。
- class_id：当要对图片进行分类时，我们可以用 class_id 对每个特征点进行区分，未设定时为 -1，需要靠自己设定
- octave：代表是从金字塔哪一层提取的得到的数据。
- pt：关键点点的坐标
- response：响应程度，代表该点强壮大小。response 代表着该关键点 how good，更确切的说，是该点角点的程度。
- size：该点直径的大小

注意一个问题：keypoint 只是保存了 opencv 的 sift 库检测到的特征点的一些基本信息，也就上面所说的这些，但 sift 所提取出来的特征向量其实不是在这个里面，特征向量通过 SiftDescriptorExtractor 提取，结果放在一个 Mat 的数据结构中。这个数据结构才真正保存了该特征点所对应的特征向量。具体见后文对 SiftDescriptorExtractor 所生成的对象的详解。

4. imgproc

包含的内容如下：

- 图像滤波
- 直方图
- 结构分析和形状描述
- 图像变换

4.1 图像滤波

4.1.1 bilateralFilter 双边滤波

```
void bilateralFilter(InputArray src, OutputArray dst, int d, double sigmaColor,
    double sigmaSpace, int borderType=BORDER_DEFAULT )
```

4.1.2 blur 均值滤波

```
void blur(InputArray src, OutputArray dst, Size ksize, Point anchor=Point(-1,-1), int
     borderType=BORDER_DEFAULT )
```

- ksize size(3,3) ···

4.1.3 boxFilter 方框滤波

```
void boxFilter(InputArray src, OutputArray dst, int ddepth, Size ksize, Point anchor=
    Point(-1,-1), bool normalize=true, int borderType=BORDER_DEFAULT )
```

- ddepth 输出图像的深度，-1 代表使用原图深度
- ksize 内核的大小

4.1.4 GaussianBlur 高斯滤波

```
GaussianBlur(InputArray src, OutputArray dst, Size ksize, double sigmaX, double
    sigmaY=0, int borderType=BORDER_DEFAULT )
```

- sigmaX 表示高斯核函数在 X 方向的的标准偏差
- sigmaY 表示高斯核函数在 Y 方向的的标准偏差

4.1.5 medianBlur 中值滤波

```
1 void medianBlur(InputArray src, OutputArray dst, int ksize)
```

4.1.6 filter2D 滤波器掩码

```
1 void filter2D(InputArray src, OutputArray dst, int ddepth, InputArray kernel, Point
      anchor=Point(-1,-1), double ∆=0, int borderType=BORDER_DEFAULT )
```

- kernel 给定的掩膜

4.2 数学形态学

4.2.1 getStructuringElement 获得不同结构元素

```
1 Mat getStructuringElement(int shape, Size ksize, Point anchor=Point(-1,-1))
```

- shape 结构元素的形状：
  - 矩形（包括线形）MORPH_RECT
  - 椭圆（包括圆形）MORPH_ELLIPSE
  - 十字形 MORPH_CROSS
- ksize 结构元素的大小

4.2.2 dilate 膨胀

```
1 void dilate(InputArray src, OutputArray dst, InputArray kernel, Point anchor=Point
      (-1,-1), int iterations=1, int borderType=BORDER_CONSTANT, const Scalar&
      borderValue=morphologyDefaultBorderValue() )
```

- src 输入图像，图像的通道可以是任意的，但是图像位深应该为以下几种：CV_8U, CV_16U, CV_16S, CV_32F,CV_64F
- dst 输出图像
- kernel 输入结构元素

4.2.3 erode 膨胀

```
1 void erode(InputArray src, OutputArray dst, InputArray kernel, Point anchor=Point
    (-1,-1), int iterations=1, int borderType=BORDER_CONSTANT, const Scalar&
    borderValue=morphologyDefaultBorderValue() )
```

4.2.4 morphologyEx 形态学操作

```
1 void morphologyEx(InputArray src, OutputArray dst, int op, InputArray kernel, Point
    anchor=Point(-1,-1), int iterations=1, int borderType=BORDER_CONSTANT, const
    Scalar& borderValue=morphologyDefaultBorderValue() )
```

- src 输入图像，Mat 类的对象即可。图像位深应该为以下五种之一：CV_8U, CV_16U,CV_16S, CV_32F 或 CV_64F。
- dst 输出图像，函数的输出参数，需要和源图片有一样的尺寸和类型。
- op 表示形态学运算的类型，可以是如下之一的标识符：
  - MORPH_OPEN – 开运算（Opening operation）
  - MORPH_CLOSE – 闭运算（Closing operation）
  - MORPH_GRADIENT -形态学梯度（Morphological gradient）
  - MORPH_TOPHAT - “顶帽”(“Top hat”)
  - MORPH_BLACKHAT - “黑帽”(“Black hat “)
- kernel 形态学运算的内核。若为 NULL 时，表示的是使用参考点位于中心 3x3 的核。我们一般使用函数 getStructuringElement 配合这个参数的使用。

4.3 几何变换

4.3.1 LinearPolar

将图像映射到线性极坐标空间中。

```
1 void cvLinearPolar(const CvArr* src, CvArr* dst, CvPoint2D32f center, double
    maxRadius, int flags=CV_INTER_LINEAR+CV_WARP_FILL_OUTLIERS )
```

- src 输入图像。
- dst 输出图像。
- center 变换的中心，输出图像在这里最精确。
- maxRadius 反幅度的尺度参数。
- flags 插值方法和以下选择标志的结合
  - CV_WARP_FILL_OUTLIERS 填充输出图像所有像素，如果这些点有和外点对应的，则置零。
  - CV_WARP_INVERSE_MAP 表示矩阵由输出图像到输入图像的逆变换，并且因此可以直接用于像素插值。否则，函数从 map_matrix 中寻找逆变换。

4.3.2 LogPolar

将图像映射到对数极坐标空间中。

```
1 void cvLogPolar(const CvArr* src, CvArr* dst, CvPoint2D32f center, double M, int
    flags=CV_INTER_LINEAR+CV_WARP_FILL_OUTLIERS )
```

- src 输入图像。
- dst 输出图像。
- center 变换的中心，输出图像在这里最精确。

- M 幅度的尺度参数。
- flags 插值方法和以下选择标志的结合
  - CV_WARP_FILL_OUTLIERS 填充输出图像所有像素，如果这些点有和外点对应的，则置零。
  - CV_WARP_INVERSE_MAP 表示矩阵由输出图像到输入图像的逆变换，并且因此可以直接用于像素插值。否则，函数从 map_matrix 中寻找逆变换。

4.3.3 Remap

对图像进行普通几何变换

```
1 void remap(InputArray src, OutputArray dst, InputArray map1, InputArray map2, int
    interpolation, int borderMode=BORDER_CONSTANT, const Scalar& borderValue=Scalar()
    )
```

- src 输入图像.
- dst 输出图像.
- mapx 为 x 坐标的映射 (32fC1 image).
- mapy 为 y 坐标的映射 (32fC1 image).
- flags 插值方法和以下开关选项的组合:
  - CV_WARP_FILL_OUTLIERS 填充边界外的像素. 如果输出图像的部分象素落在变换后的边界外，那么它们的值设定为 fillval。

4.3.4 Resize

改变图像大小

```
1 void resize(InputArray src, OutputArray dst, Size dsize, double fx=0, double fy=0,
    int interpolation=INTER_LINEAR )
```

- src 输入图像.
- dst 输出图像.
- dsize 输出图像的尺寸
- interpolation 插值方法:
  - CV_INTER_NN 最近邻插值,
  - CV_INTER_LINEAR 双线性插值 (缺省使用)
  - CV_INTER_AREA 使用象素关系重采样。当图像缩小时候，该方法可以避免波纹出现。当图像放大时，类似于 CV_INTER_NN 方法
  - CV_INTER_CUBIC 立方插值

4.4 各种形式图像转换

4.4.1 adaptiveThreshold 自适应阈值

```
1 void adaptiveThreshold(InputArray src, OutputArray dst, double maxValue, int
    adaptiveMethod, int thresholdType, int blockSize, double C)
```

- src 输入图像
- dst 输出图像
- maxValue 自适应阈值的最大值
- adaptiveMethod 自适应阈值算法使用：CV_ADAPTIVE_THRESH_MEAN_C（均值）或 CV_ADAPTIVE_THRESH_GAUSSIAN_C（加权）
- thresholdType 取阈值类型：CV_THRESH_BINARY 或 CV_THRESH_BINARY_INV
- blockSize 用来计算阈值的象素邻域大小: 3, 5, 7, ...
- C 对方法的有关参数,对方法 CV_ADAPTIVE_THRESH_MEAN_C 和 CV_ADAPTIVE_THRESH_ 它是一个从均值或加权均值提取的常数（见讨论）, 尽管它可以是负数。

4.4.2 cvtColor 色彩转换

```
1 void cvtColor(InputArray src, OutputArray dst, int code, int dstCn=0 )
```

- code 色彩空间转换代码
  有如下转换:
  - RGB↔GRAY:COLOR_BGR2GRAY, COLOR_RGB2GRAY, COLOR_GRAY2BGR, COLOR_
  - RGB↔CIE:COLOR_BGR2XYZ, COLOR_RGB2XYZ, COLOR_XYZ2BGR, COLOR_XYZ2R
  - RGB↔YCrCb JPEG (or YCC):COLOR_BGR2YCrCb, COLOR_RGB2YCrCb, COLOR_YCrC
    COLOR_YCrCb2RGB
  - RGB↔HSV:COLOR_BGR2HSV, COLOR_RGB2HSV, COLOR_HSV2BGR, COLOR_HSV2R
    ...
- dstCn 表示 dst 图像的波段数，这个值默认是 0，它可以从参数 code 中推断。

4.4.3 threshold 固定阈值二值化

```
1 double threshold(InputArray src, OutputArray dst, double thresh, double maxval, int
    type)
```

- thresh 二值化阈值
- maxval 二值化的最大值
- type 运算方法:THRESH_BINARY,THRESH_BINARY_INV,THRESH_TRUNC,THRESH_TOZER

4.4.4 watershed 分水岭算法

```
1 void watershed(InputArray image, InputOutputArray markers)
```

- image 输入图像为 3 路 8 位
- markers 输出

4.5 结构分析和形状描述

4.5.1 findContours

```
1 void findContours(InputOutputArray image, OutputArrayOfArrays contours, OutputArray
    hierarchy, int mode, int method, Point offset=Point())
2 void findContours(InputOutputArray image, OutputArrayOfArrays contours, int mode, int
    method, Point offset=Point())
```

- image 为二值单通道图像
- contours 输出轮廓坐标
- hiararchy 参数和轮廓个数相同，每个轮廓 contours[i] 对应 4 个 hierarchy 元素 hierarchy[i][0] hierarchy[i][3]，分别表示后一个轮廓、前一个轮廓、父轮廓、内嵌轮廓的索引编号，如果没有对应项，该值设置为负数
- mode 表示轮廓的检索模式
  - CV_RETR_EXTERNAL 表示只检测外轮廓
  - CV_RETR_LIST 检测的轮廓不建立等级关系
  - CV_RETR_CCOMP 建立两个等级的轮廓，上面的一层为外边界，里面的一层为内孔的边界信息。如果内孔内还有一个连通物体，这个物体的边界也在顶层。
  - CV_RETR_TREE 建立一个等级树结构的轮廓
- method 为轮廓的近似办法
  - CV_CHAIN_APPROX_NONE 存储所有的轮廓点，相邻的两个点的像素位置差不超过 1，即 max（abs（x1-x2），abs（y2-y1））==1
  - CV_CHAIN_APPROX_SIMPLE 压缩水平方向，垂直方向，对角线方向的元素，只保留该方向的终点坐标，例如一个矩形轮廓只需 4 个点来保存轮廓信息
  - CV_CHAIN_APPROX_TC89_L1, CV_CHAIN_APPROX_TC89_KCOS 使用 teh-Chinl chain 近似算法

4.6 绘图相关函数

opencv.hpp

4.6.1 Scalar

- Scalar(a,b,c)
  - a 为蓝色
  - b 为绿色
  - c 为红色

4.6.2 Rectangle

```
1 rectangle(Mat& img, Point pt1, Point pt2, const Scalar&color, int thickness=1,int
      lineType=8, int shift=0)
```

- img 为画矩形的对象
- pt1 所画矩形左上角点坐标
- pt2 所画矩形右下角点坐标

```
1 rectangle(Mat& img,Rect rec, const Scalar&color, int thickness=1, int lineType=8,int
      shift=0 )
```

- rec 为确定矩形的另一种方式，给定左上角的坐标和长宽
- color 指定矩形的颜色
- thickness 线宽
- lingType 边框线型
- shift 坐标点的小数点位数

4.6.3 Line

```
line(Mat& img, Point pt1, Point pt2, const Scalar& color, int thickness=1, int
    lineType=8, int shift=0)
```

- img 为要画的对象
- pt1 线条的起点坐标
- pt2 线条的终点坐标

4.6.4 Circle

```
circle(Mat&img, Point center, int radius, const Scalar&color,int thickness=1, int
    lineType=8, int shift=0)
```

- img 为要画圆的对象
- center 为圆的中心坐标
- radius 为圆的半径

4.6.5 Ellipse

```
ellipse(Mat& img, Point center, Size axes, double angle, double startAngle, double
    endAngle, const Scalar& color, int thickness=1, int lineType=8, int shift=0)
```

- center 椭圆中心的坐标
- axes 为椭圆长短轴的一半
- startAngle 椭圆弧起始的角度
- endAngle 椭圆弧终止的角度

```
ellipse(Mat& img, constRotatedRect& box, const Scalar& color, int thickness=1, int
    lineType=8)
```

- box 指定椭圆中心和旋转角度的信息，通过 RotatedRect 或 CvBox2D. 这表示椭圆画在旋转矩形上（矩形是不可见的，只是指定了一个框而已）

4.6.6 PutText

```
putText(Mat& img, const string& text, Point org, int fontFace, double fontScale,
    Scalar color, int thickness=1, int lineType=8, bool bottomLeftOrigin=false )
```

- text 要显示的文字
- org 文字在图像中左下角的坐标
- fontFace 字体类型，可选择字体：FONT_HERSHEY_SIMPLEX, FONT_HERSHEY_PLAIN··· 所有类型都可以配合 FONT_HERSHEY_ITALIC 使用，产生斜体效果。
- fontScale 字体的大小

4.6.7 drawContours

```
void drawContours(InputOutputArray image, InputArrayOfArrays contours, int contourIdx
    , const Scalar& color, int thickness=1, int lineType=LINE_8, InputArray hierarchy
    =noArray(), int maxLevel=INT_MAX, Point offset=Point() )
```

- image 目标图像
- contours 输入轮廓，轮廓被存放在 vector<vector<Point> > 中
- contourldx 这个参数是描述怎样画轮廓。如果为负数，则画出全部轮廓。
- color 表示化轮廓的颜色。

5. shape

5.1 Hausdorff

```
int main()
{
    cv::Ptr <cv::HausdorffDistanceExtractor> hausdorff_ptr = cv::
        createHausdorffDistanceExtractor();
    vector<Point> pt1s;
    vector<Point> pt2s;
    pt1s.push_back(Point(0, 0));
    pt1s.push_back(Point(0, 4));
    pt2s.push_back(Point(4, 6));
    pt2s.push_back(Point(4, 2));
    float distance = hausdorff_ptr->computeDistance(pt1s, pt2s);
    cout << distance << endl;
}
```

6. object delection

6.1 matchTemplate

在模板块和输入图像之间寻找匹配，获得匹配结果图像。

```
void matchTemplate(InputArray image, InputArray templ, OutputArray result, int method
    )
```

- image 要搜索的图像对象
- templ 需要匹配的模板
- result 比较结果，必须是单通单 32 位浮点数
- method 比较算法，共有六种方法如下：

- CV_TM_SQDIFF 平方差匹配法，最好的匹配为 0，值越大匹配越差
- CV_TM_SQDIFF_NORMED 归一化平方差匹配法
- CV_TM_CCORR 相关匹配法，采用乘法操作，数值越大表明匹配越好
- CV_TM_CCORR_NORMED 数值越小越好
- CV_TM_CCOEFF 相关系数匹配法，最好的匹配为 1，-1 表示最差的匹配
- CV_TM_CCOEFF_NORMED 归一化相关系数匹配法

例子：

```
int main(int argc, char* argv[])
{
    string path1="/Users/wangruchen/work/program/part-matching/pic/1.jpg";
    string path2="/Users/wangruchen/work/program/part-matching/pic/1-part.png";
    Mat srcIm=imread(path1);
    Mat partIm=imread(path2);
    Mat result;
    matchTemplate(srcIm, partIm, result, TM_SQDIFF);
    normalize(result, result, 0, 1, NORM_MINMAX);//归一化
    double minVal,maxVal;
    Point minPoi,maxPoi;
    minMaxLoc(result, & minVal, &maxVal, &minPoi, &maxPoi);//寻找矩阵result
        中最大值和最小值的位置
    rectangle(srcIm, minPoi, Point2f(minPoi.x+partIm.cols,minPoi.y+partIm.rows),
        Scalar(255,0,0));//在匹配位置画出矩形
    imshow("result", srcIm);
    waitKey(0);
    return 0;
}
```

7. 图像读取和保存

7.1 imread

载入一副图片。

```
Mat imread(const String& filename, int flags=IMREAD_COLOR )
```

- flags
  - CV_LOAD_IMAGE_ANYDEPTH 转换为 16 位 32 位，或者 8 位
  - CV_LOAD_IMAGE_COLOR 转换为彩色
  - CV_LOAD_IMAGE_GRAYSCALE 转换为灰度图
  - 大于 0 返回为 3 通道彩色图像
  - 小于 0 返回为 1 通道灰度图像
  - 小于 0 返回加载图（alpha 通道）
- 支持一下格式：.bmp, .dib, .jpeg, .jpg, .jpe, .jp2, .png, .webp, .pbm, .pgm, .ppm, .sr, .ras, .tiff, .tif

8. 数据类型及位数总结

表示了矩阵中元素的类型以及矩阵的通道个数，它是一系列的预定义的常量，其命名规则为 CV_(位数）+（数据类型）+（通道数)。具体的有以下值：

8.1 Unsigned 8bits

CvMat 数据结构参数：CV_8UC1，CV_8UC2，CV_8UC3，CV_8UC4

8.2 Signed 8bits

CvMat 数据结构参数：CV_8SC1，CV_8SC2，CV_8SC3，CV_8SC4

8.3 Unsigned 16bits

CvMat 数据结构参数：CV_16UC1，CV_16UC2，CV_16UC3，CV_16UC4

8.4 Signed 16bits

CvMat 数据结构参数：CV_16SC1，CV_16SC2，CV_16SC3，CV_16SC4

8.5 Signed 32bits

CvMat 数据结构参数：CV_32SC1，CV_32SC2，CV_32SC3，CV_32SC4

8.6 Float 32bits

CvMat 数据结构参数：CV_32FC1，CV_32FC2，CV_32FC3，CV_32FC4

8.7 Double 64bits

CvMat 数据结构参数：CV_64FC1，CV_64FC2，CV_64FC3，CV_64FC4