项目资料 v1.0

1 OpenCV 库基本语法

读取图片 `cv2.imread(path, flag)`

`path`：图片文件路径字符串

`flag`：图片格式

- `cv2.IMREAD_COLOR` 或 1：彩色格式（默认）
- `cv2.IMREAD_GRAYSCALE` 或 0：灰度格式

显示图片 `cv2.imshow(window_name, image)`

`window_name`：窗口名称　　`image`：图片变量名

删除窗口 `cv2.destroyAllWindows()` 、 `cv2.destroyWindow(window_name)`

保存图片文件 `cv2.imwrite(filename, image)`

绘制圆形 `cv2.circle(image, center_coordinates, radius, color, thickness)`

`color`: BGR tuple　　`thickness`：轮廓线条粗细（像素为单位）

高斯模糊算法 `blur = cv.GaussianBlur(image, ksize, 0)`

`ksize`: 高斯核大小

2 相机标定

2.1 Introduction

为确定物体表面点的三维位置与图像中对应点的关系，建立相机成像的几何模型。几何模型参数就是相机参数。求解参数，获取相机内部、外部参数矩阵的过程就称之为相机标定。

内部参数（intrinsic parameter）：真正的相机参数，与使用的光学元件有关。

$$\boldsymbol{A} = \begin{bmatrix} \alpha & 0 & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

其中 α，β 为 x、y 方向的相机焦距，u_0、v_0 为主点坐标。

外部参数（extrinsic parameter）：相机在世界坐标系中的位置和方向（位姿）。旋转矩阵 $\boldsymbol{R}$ 为 3×3 矩阵，位移矩阵为 3×1 矩阵。外参矩阵 $[\boldsymbol{Rt}]$ 为 3×4 矩阵。

张正友标定法：移动相机或移动标定物，从不同角度拍摄单平面的标定图案，推荐四到五个方向。

2.2 Code Implementation

棋盘格内角点（internal corners）坐标

棋盘格图样需要非常清晰，否则无法识别

```
retval, corners = cv2.findChessboardCorners(image, patternSize)
```

`retval`：返回值 True 或 False　　`corners`：棋盘格内角点坐标

`image`：图像源　　`patternSize`：棋盘格内角点行列数

亚像素角点坐标

考虑亚像素，提高精度

```
corners2 = cv2.cornerSubPix(image, corners, winSize, zeroZone=(-1,-1), criteria)
```

corners: 优化前的棋盘格内角点坐标　　winSize: 搜索框大小

criteria: `(cv2.TERM_CRITERIA_MAX_ITER | cv2.TERM_CRITERIA_EPS, 30, 0.001)`

标记角点

```
cv2.drawChessboardCorners(image, patternSize, corners, patternWasFound)
```

patternWasFound: `cv2.findChessboardCorners` 的返回值 True 或 False

内外参数计算

```
retval, matrix, dist, rvec, tvec = cv2.calibrateCamera(obj_points, img_points, size,
    None, None)
```

retval: 返回值 True 或 False　　matrix: 相机参数矩阵　　dist: 畸变系数

rvec: 旋转向量　　tvec: 平移向量

obj_points: 世界坐标系中的点　　img_points: 对应的图像点　　size: 图像尺寸

相机参数优化

```
newMatrix, roi = cv2.getOptimalNewCameraMatrix(matrix, dist, imageSize, scaleParam,
    newImageSize)
```

newMatrix: 优化相机参数矩阵　　roi: 输出图像的感兴趣区域

matrix: 原相机参数矩阵　　dist: 畸变系数　　imageSize: 图像尺寸

scale_param: 自由放缩系数。此处为 1　　newImageSize: 新图像尺寸

3 去除畸变

畸变: 因为透视原因造成的失真。

```
result = cv2.undistort(image, matrix, dist, None, newCameraMatrix)
```

image: 图像源　　matrix: 相机参数矩阵 `cv2.calibrateCamera`　　dist: 畸变系数

newCameraMatrix: 优化相机参数 `cv2.getOptimalNewCameraMatrix`

4 透视变换

透视变换是将图像从一个视平面投影到另外一个视平面的过程。

透视变换矩阵

```
matrix = cv2.getPerspectiveTransform(src, dst)
```

matrix: 透视变换矩阵　　src: 原图四边形顶点坐标　　dst: 输出图像的四边形顶点坐标

```
pts_ = cv2.perspectiveTransform(pts, M)
```

透视变换

```
result = cv2.warpPerspective(src, matrix, dsize)
```

result: 输出图像

src: 图象源　　matrix: 透视变换矩阵　　dsize: 输出图像尺寸

5 边缘识别

分析图像中的不连续之处，边缘像素的梯度明显高于其邻点。

5.1 Canny 算法（已弃用）

```
result = cv2.Canny(image, threshold1 = 100, threshold2 = 200)
```

5.2 findContours（已弃用）

5.3 Holistically-Nested Edge Detection 整体嵌套边缘检测（HED）

定义裁剪类

```
class CropLayer(object):
    def __init__(self, params, blobs):
        self.startX = 0
        self.startY = 0
        self.endX = 0
        self.endY = 0
cv2.dnn_registerLayer("Crop", CropLayer)
```

读取 Caffe 模型 `net = cv2.readNetFromCaffe(protoPath, modelPath)`

`protoPath`：.prototxt 文件路径　　`modelPath`：.caffemodel 文件路径

Preprocessing 图像预处理

Blob 是指图像中的一块连通区域，每一个 Blob 都代表一个前景目标。

```
blob = cv2.dnn.blobFromImage(image, scalefactor, size, mean, swapRB)
```

`image`：图像源　　`scalefactor`：放缩系数，此处为 1

`size`：神经网络在训练的时候要求输入的图片尺寸

`mean`：为减少光照影响，需要将图片整体减去的平均值（RGB tuple）

`swapRB`：是否交换红蓝颜色通道

输入模型

将预处理结果输入神经网络模型，得到 HED 识别结果

```
net.setInput(blob)
hed = net.forward()
```

6 圆形检测

6.1 Circular Hough Transform 霍夫圆变换

霍夫圆变换是将二维图像空间中一个圆转换为该圆半径、圆心横纵坐标所确定的三维参数空间中一个点的过程。圆周上任意三个点为一选举人，而这三个点所确定的圆则为一侯选人。遍历圆周上所有点，任意三个点所确定的候选圆进行投票，以此确定该圆。

霍夫圆变换是检测圆形的传统算法，能够识别不完整的圆。算法复杂度较高，需要大量内存，速度较慢；算法较为敏感，常常忽略因视角产生的椭圆；算法对圆心位置的识别较为准确，半径识别有一定误差。

```
cv2.dnn.HoughCircles(image, method, dp=1, minDist, param1=100, param2=60, minRadius,
    maxRadius)
```

`image`：图像源　　`method`：`cv2.HOUGH_GRADIENT` 唯一选项

`dp`：累加器分辨率　　`minDist`：两圆心间的最小距离

`param1`、`param2`: canny 算法参数　　`minRadius`、`maxRadius`：最小、最大圆半径

6.2 K-means Clustering 聚类算法

目标将样本分为 K 类（cluster）。随机选取 K 个对象作为初始的聚类中心（centroid），然后计算每个对象与各个种子聚类中心之间的距离，把每个对象分配给距离它最近的聚类中心。计算聚类后每个类的质心（即类中样本均值），作为新的类中心。然后重新执行上述步骤，直到聚类结果不再改变。

算法适宜处理大数据量，对异常数据较敏感。

```
from sklearn.cluster import KMeans
km = KMeans(n_clusters=4).fit(circleshed[0, :, 0:2])
```