Lab-7

BLOB Detection

- Blob detection is a basic method in computer vision used to locate **areas of interest** in a picture.
- Blob detection is the process of **finding related areas** in an image that share features.
- It assembles linked **pixels into clusters**.
- **Applications** for blob detection include image segmentation, feature extraction, object tracking, etc.
- **Note:-** you can apply various **morphological**, **smoothing** operations in combination before apply blob detector function for better results.
  - To improve blob detection performance, image preprocessing, including **noise reduction**, **contrast enhancement**, and **color space conversions.**

Techniques

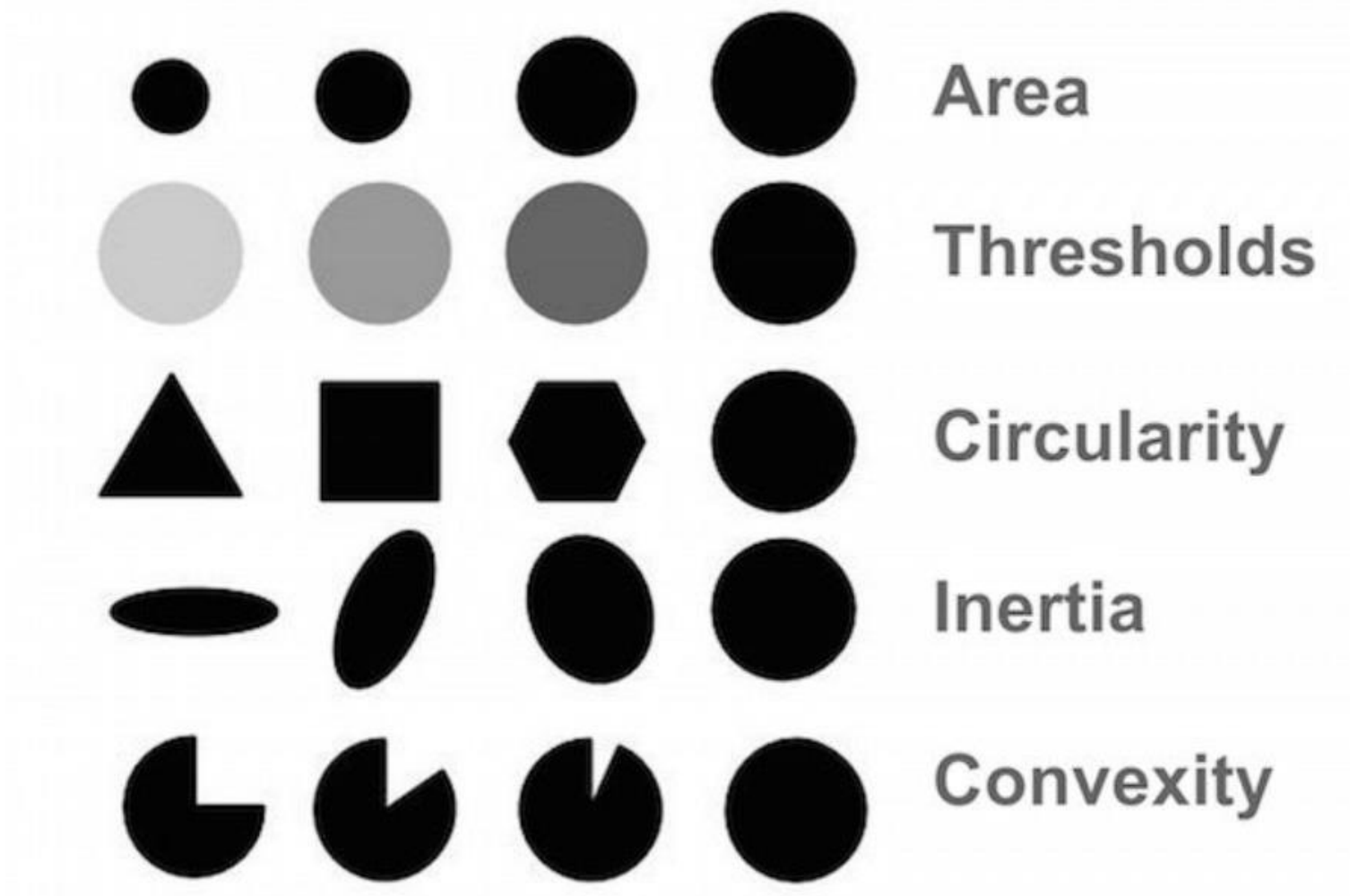
- **Laplacian of Gaussian (LoG):** Gaussian-smoothed image's Laplacian is the method used here.
  - **Most accurate** but slowest
- **Difference of Gaussian (DoG):**DoG algorithm computes the difference between two Gaussian-smoothed pictures.
  - Faster approximation of LoG
- **Determinant of Hessian (DoH):**Uses the determinant of the Hessian matrix.
  - Here, local curvature of an image is represented by the Hessian matrix.
  - **Fastest Method**

Steps and Functions Used

- Load libraries and images.
- Set Up the Simple Blob Detector
  - cv2.SimpleBlobDetector() (old version)
  - cv2.SimpleBlobDetector_Create ()
  - cv2.SimpleBlobDetector_Params()
- Create keypoints detector using .detect(img) function
- Obtain Key Points on the Image cv2.drawKeypoints( )
- Display using cv2.imshow() function
- **DoG, LoG and DoH**
  - from skimage.feature import **blob_dog**, **blob_log**, **blob_doh**

```
# Simple blob detector
import cv2
import numpy as np;
# Read image
im = cv2.imread("blob.jpg", cv2.IMREAD_GRAYSCALE)
# Set up the detector with default parameters.
detector = cv2.SimpleBlobDetector()
# Detect blobs.
keypoints = detector.detect(im)
# Draw detected blobs as red circles.
# cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS ensures the size of the circle corresponds to the size of blob
im_with_keypoints = cv2.drawKeypoints(im, keypoints, np.array([]), (0,0,255), cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
# Show keypoints
cv2.imshow("Keypoints", im_with_keypoints)
cv2.waitKey(0)
```

Parameters



 # Setup SimpleBlobDetector parameters.

```
params = cv2.SimpleBlobDetector_Params()

# Change thresholds

params.minThreshold = 10;

params.maxThreshold = 200;

# Filter by Area.

params.filterByArea = True

params.minArea = 1500

# Filter by Convexity

params.filterByConvexity = True

params.minConvexity = 0.87

# Filter by Inertia

params.filterByInertia = True

params.minInertiaRatio = 0.01
```

```
# Create a detector with the parameters
ver = (cv2.__version__).split('.')
if int(ver[0]) < 3 :
 detector = cv2.SimpleBlobDetector(params)
else :
 detector =
cv2.SimpleBlobDetector_create(params)
```

Comparison

Method	Library	Best For	Speed	Notes
`cv2.SimpleBlobDetector()`	OpenCV	Simple round blobs	Fast	Good for basic blob detection
`blob_dog`	scikit-image	Multi-scale bright blobs	Medium	Uses Difference of Gaussians
`blob_log`	scikit-image	Round blobs	Slow	More accurate but computationally heavy
`blob_doh`	scikit-image	Elongated blobs	Medium	Best for corner-like or filament structures

Refs

- https://towardsdatascience.com/image-processing-blob-detection-204dc6428dd/
- https://github.com/AshishPandey88/Blob-Detection?tab=readme-ov-file
- https://cyrillugod.medium.com/blob-detection-in-action-7425638f5347
- https://mathematica.stackexchange.com/questions/310557/performant-blob-detection-in-images
- https://www.geeksforgeeks.org/find-circles-and-ellipses-in-an-image-using-opencv-python/
- https://www.geeksforgeeks.org/blob-detection-using-opencv/

Image quality enhancement

- Adjusting brightness and contrast
- Sharpening images
- Removing noise from images
- Enhancing color in images
- Image resizing and scaling
- Inverse Transform
- Equalizing histograms
- Super-resolution

Adjusting brightness and contrast

- image2 = cv2.**addWeighted**(image, contrast, np.zeros(image.shape, image.dtype), 0, brightness)

Adjusts the brightness by adding x to each pixel value

  - brightness = x

Adjusts the contrast by scaling the pixel values

  - contrast = y

- **cv2.convertScaleAbs()** function adjust the brightness and contrast using a combination of **scaling** and **shifting** the pixel values.

- **Sharpening images**
  - cv2.filter2D() function **or**
  - cv2.Laplacian() calculates the Laplacian of an image.
  - **The Laplacian is a second-order derivative, meaning it measures the rate of change of the rate of change of pixel intensities.**
- **Removing noise from images**
  - cv2.medianBlur() function
  - cv2.GaussianBlur()
- **Enhancing color in images**
  - cv2.cvtColor() function after converting RGB to HSV and adjusts the hue, saturation, and value (brightness).
- **Image resizing and scaling**
  - cv2.resize() function with cv2.INTER_NEAREST, cv2.INTER_LINEAR, cv2.INTER_CUBIC, and others.

- **Inverse Transform**
  - inverse the color by simply subtracting each value from 255.
- **Equalizing histograms**
  - cv2.equalizeHist() function and reproduce the image back.
- **Super-resolution**
  - Use the pyrUp() and pyrDown() functions
- **Color correction**
  - cvtColor() and inRange() function. **To perform task like** color balance, color grading, and white balancing.