Edge detection

Edge detection is an image processing technique for finding the boundaries of objects within images. Edge detection is used for image segmentation and data extraction in areas such as image processing, computer vision, and machine vision.

Introduction:

1. Edge.
2. Goal of edge detection.
3. Edge descriptors.
4. The four steps of edge detection.
5. Techniques of Edge Detector.

1. Edge.

Edges are significant local changes of intensity in an image.

2. Goal of edge detection.

a. Produce a line drawing of a scene from an image of that scene.
b. Important features can be extracted from the edges of an image (e.g., corners, lines, curves).
c. These features are used by higher-level computer vision algorithms (e.g., recognition).

3. Edge descriptors.

1. Edge normal: unit vector in the direction of maximum intensity change.
2. Edge direction: unit vector to perpendicular to the edge normal.
3. Edge position or center: the image position at which the edge is located.
4. Edge strength: related to the local image contrast along the normal.

4. The four steps of edge detection.

1. Smoothing: suppress as much noise as possible, without destroying the true edges.
2. Enhancement: apply a filter to enhance the quality of the edges in the image (sharpening).
3. Detection: determine which edge pixels should be discarded as noise and which should be retained (usually, thresholding provides the criterion used for detection).
4. Localization: determine the exact location of an edge (sub-pixel resolution might be required for some applications, that is, estimate the location of an edge to better than the spacing between pixels). Edge thinning and linking are usually required in this step.

5. Techniques of Edge Detector.

Common edge detection algorithms include Sobel, Canny, Prewitt, Roberts, and fuzzy logic methods.

Canny Edge Detector.

The Canny edge detector is the first derivative of a Gaussian and closely approximates the operator that optimizes the product of signal-to-noise ratio and localization. Canny Edge Detection is a popular edge detection algorithm. It was developed by John F. Canny.
The Canny edge detection algorithm: Let *I[i, j]* denote the image.

Smoothing filter: using separable filtering is an array of smoothed data.

$$S[i, j] = G[i, j, \sigma] * I[i, j]$$

Where σ is the spread of the Gaussian and controls the degree of smoothing.

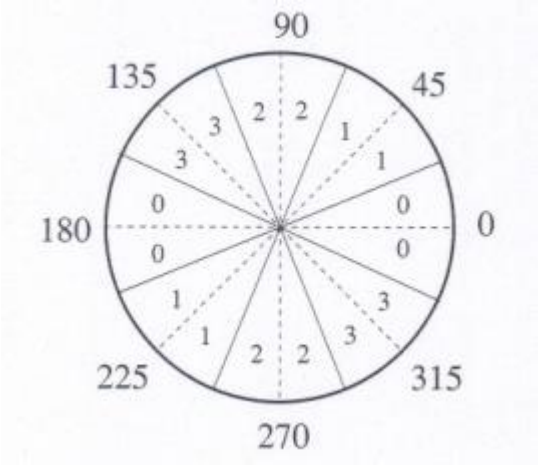
- **Gradient:** The gradient of the smoothed array *S[i, j]* can be computed using the 2 x 2 first-difference approximations to produce two arrays *P[i,j]* and *Q[i,j]* for the x and y partial derivatives. The finite differences are averaged over the 2 x 2 square so that the x and y partial derivatives are computed at the same point in the image. The magnitude and orientation of the gradient can be computed from the standard formulas for rectangular-to-polar conversion:

$$\theta[i, j] = \arctan(Q[i, j], P[i, j])$$

where the arctan function takes two arguments and generates an angle over the entire circle of possible directions.

- **Non-maximum suppression:** Non-maxima suppression thins the ridges of gradient magnitude in M[i, j] by suppressing all values along the line of the gradient that are not peak values of a ridge. The algorithm begins by reducing the angle of the gradient θ[i, j] to one of the four sectors.

$$\varsigma[i, j] = sector(\theta[i, j])$$



- **Threshold:** The typical procedure used to reduce the number of false edge fragments in the non-maxima-suppressed gradient magnitude is to apply a threshold to *N[i, j*]. All values below the threshold are changed to zero. The double threshold step aims at identifying 3 kinds of pixels: strong, weak, and non-relevant:
  1. Strong pixels are pixels that have an intensity so high that we are sure they contribute to the final edge.
  2. Weak pixels are pixels that have an intensity value that is not enough to be considered as strong ones, but yet not small enough to be considered as non-relevant for the edge detection.
  3. Other pixels are considered as non-relevant for the edge. the double thresholds holds for: high threshold, low threshold, all pixels having intensity between both thresholds.
- **Tracing edges:** Finalize the detection of edges by suppressing all the other edges that are weak and not connected to strong edges.

Sobel operator:

A very common operator for doing this is a Sobel Operator, which is an approximation to a derivative of an image. It is separate in the y and x directions. We use a kernel 3 by 3 matrix, one for each x and y direction. The gradient for x direction has minus numbers on the left hand side and positive numbers on the right hand side and we are preserving a little bit of the center pixels.

-1	0	+1
-2	0	+2
-1	0	+1

+1	+2	+1
0	0	0
-1	-2	-1

At each point in the image, the resulting gradient approximations can be combined to give the gradient magnitude:

$$G = \sqrt{G_x^2 + G_y^2}$$

For example: We will use the Canny and Sobel operator.

```
1.  import numpy as np
2.  import cv2 as cv
3.  from matplotlib import pyplot as plt
4.
5.  scale = 1
6.  delta = 0
7.  ddepth = cv.CV_16S
8.
9.  #Load file
10. img = cv.imread('lena.png', cv.IMREAD_COLOR)
11.
12. img = cv.GaussianBlur(img, (3, 3), 0)
13.
14. #Convert image to RGB
15. gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
16.
```

```
#Grandient X
grad_x = cv.Sobel(gray, ddepth, 1, 0, ksize=3, scale=scale, delta=delta, borderType=cv.
BORDER_DEFAULT)
abs_grad_x = cv.convertScaleAbs(grad_x)

#Gradient Y
grad_y = cv.Sobel(gray, ddepth, 0, 1, ksize=3, scale=scale, delta=delta, borderType=cv.
BORDER_DEFAULT)
abs_grad_y = cv.convertScaleAbs(grad_y)

sobel = cv.addWeighted(abs_grad_x, 0.5, abs_grad_y, 0.5, 0)

#Canny edge detection
canny = cv.Canny(gray,50,100)

#Display images
titles = ['Original', 'Sobel', 'Canny' ]
images = [gray, sobel, canny]

for i in range(3):
    plt.subplot(2, 2, i+1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]), plt.yticks([])

plt.show()
```