# Edge detection

## I Introduction

### 1 Edge detection

- In digital image processing, edge detection is a technique used in computer vision detection and image processing to find the boundaries of objects within images.
- Edge detection works by detecting discontinuities in brightness. The points in an image where the brightness changes sharply are sets of curved line segments that are called edges.
- Edge detection algorithms include Sobel, Laplacian of Gaussian (LoG), Canny, ...

### 2 Uses of edge detection

- The edges of an image allow us to see the boundaries of objects in an image. This has many different applications in different fields like:
+ Object recognition
+ Image matching
+ Document analysis
+ Horizon detection
+ Line following robots
+ ....

### 3 Identifying edges

- Edges in image are caused by different variety of factors:
+ Surface normal discontinuity
+ Depth discontinuity
+ Surface color discontinuity
+ Illumination discontinuity

# II Edge detection methods

## 1 Sobel operator

- Sobel operator is used to detect two kinds of edges in an image:
+ Vertical direction (y-direction)
+ Horizontal direction (x-direction)

## 1.1 Implementation

- In Sobel operator, the image is processed in the X and Y directions separately first, and then combined together to form a new image which represents the sum of the X and Y edges of the image.
- First, convert the image from an color image to a Grayscale image.
- Then, we will use a kernel 3x3 matrix, one for x-direction and one for y-direction. The gradient for x-direction has minus numbers on the left hand side and positive numbers on the right hand side and zeros in the center column. Similarly, the gradient for y-direction has minus numbers on the bottom and positive numbers on top and zeros in the middle row.

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

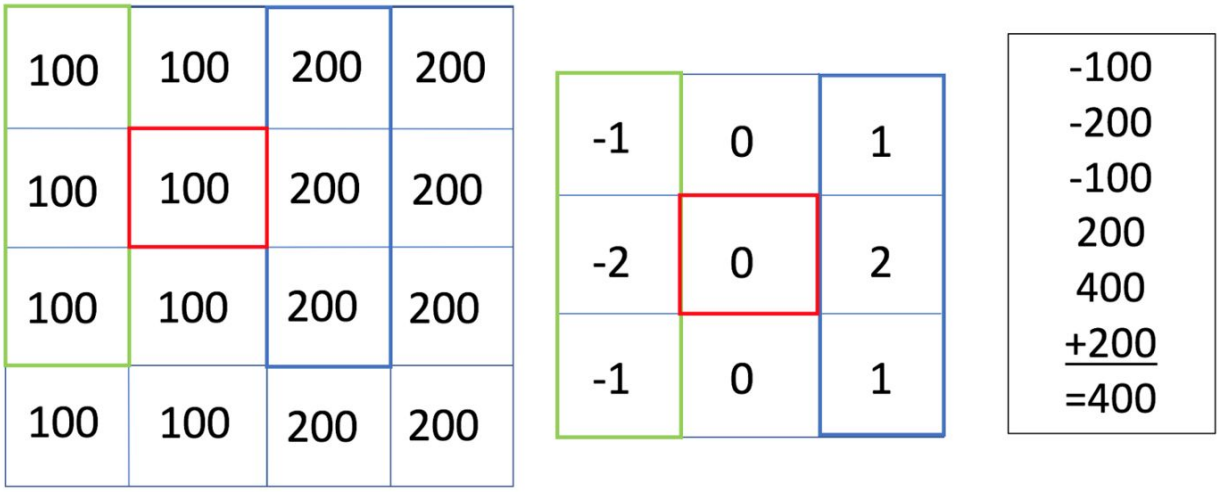
Gx

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

Gy

- The next step in Sobel Operator is scanning across X and Y direction of image to find out the amount of the difference by placing the gradient

matrix over each pixel of the image. We get two images as output, one for X-Direction and other for Y-Direction.
- By using Kernel Convolution, we can see in the image below there is an edge between the column of 100 and 200 values.




Kernel Convolution: The bigger the value at the end, the more noticeable the edge will be.

The Kernel Convolution image above is an example of X Direction Kernel usage. If an image was scanning from left to right and the filter was set at (2,2) in the image above, it would have the value of 400, which is non-zero and therefore would have a highly chance that it has an edge at that point. In case someone wants to exaggerate the edge, then the filter value of (-2,2) is needed to change to a higher magnitude (I.E (-5,5)). This would make the gradient of the edge larger and more noticeable.

If all the pixels of images were the same value, then the convolution would be zero. From there, the gradient matrix will provide a big response when one side is brighter. Another thing is that the side of the output convolution doesn't really matter.

## 2 Laplacian of Gaussian (LoG)

- Edge detection algorithms like the Sobel Operator work on the first derivative of an image. One limitation of Sobel Operator is affection of noise. Shadow or tiny color changes can be detected as edges. An alternative solution is using the second derivative, or Laplacian of Gaussian method.

### 2.1 Implementation

- Since derivative filters are very sensitive to noise, it is common to smooth the image (e.g., using a Gaussian filter) before applying the Laplacian.

Applying Gaussian filter:

Gaussian

$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$

Applying Laplacian function:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Laplacian and Gaussian functions in a single equation:

$$\mathrm{LoG}(x,y) = -\frac{1}{\pi\sigma^4}\left[1-\frac{x^2+y^2}{2\sigma^2}\right]e^{-\frac{x^2+y^2}{2\sigma^2}}$$

- There are different ways to find an approximate discrete convolution kernel that approximates the effect of the Laplacian. A possible kernel is:

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

## 3 Canny Edge detector

- Canny method includes 5 main steps:

+ Apply Gaussian filter to smooth the image in order to remove the noise
+ Find the intensity gradients of the image
+ Apply non-maximum suppression to get rid of spurious response to edge detection
+ Apply double threshold to determine potential edges
+ Track edge by hysteresis: Finalize the detection of edges by suppressing all the other edges that are weak and not connected to strong edges.

# III Implementation

## 1 Code

Above is the code of Canny Edge detector, Laplacian of Gaussian (LoG) and Sobel operator

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt

scale = 1
delta = 0
ddepth = cv.CV_16S

image = cv.imread("cadastre2.png")
image = cv.cvtColor(image, cv.COLOR_BGR2GRAY)

image = cv.GaussianBlur(image, (3, 3), 0)

# Sobel operator
# Grandient X
grad_x = cv.Sobel(image, 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(image, 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)

# Laplacian of Gaussian
laplacian = cv.Laplacian(image, ddepth, ksize=3)
laplacian = cv.convertScaleAbs(laplacian)

# Canny Edge detector
low_threshold = 25
ratio = 3
kernel_size = 3

img = cv.imread("house8.png")
img_gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

img_blur = cv.blur(img_gray, (3, 3))
detected_edges = cv.Canny(img_blur, low_threshold, low_threshold * ratio,
kernel_size)
mask = detected_edges != 0
canny = img * (mask[:, :, None].astype(img.dtype))

plt.subplot(2, 2, 1), plt.imshow(image, cmap='gray')
plt.title('Original'), plt.xticks([]), plt.yticks([])
plt.subplot(2, 2, 2), plt.imshow(laplacian, cmap='gray')
```

```
plt.title('Laplacian'), plt.xticks([]), plt.yticks([])
plt.subplot(2, 2, 3), plt.imshow(sobel, cmap='gray')
plt.title('Sobel'), plt.xticks([]), plt.yticks([])
plt.subplot(2, 2, 4), plt.imshow(canny, cmap='gray')
plt.title('Canny'), plt.xticks([]), plt.yticks([])

plt.show()
```

# 2 Result