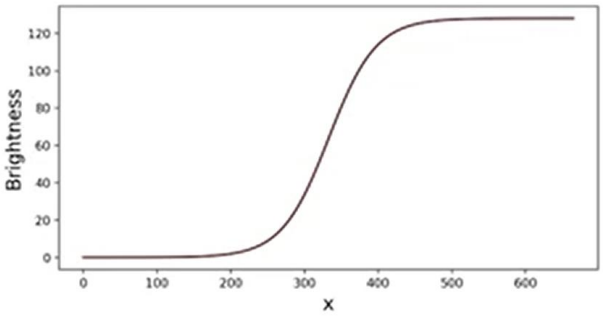
Laplacian filter:

Using first order derivatives (as in Prewitt and Sobel)

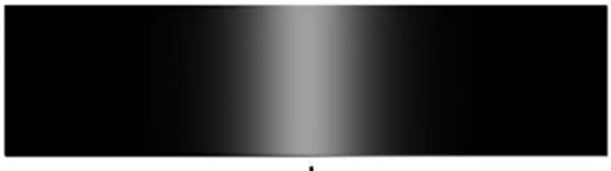
to detect edges has a drawback

Consider this image which contains
a smoothed edge

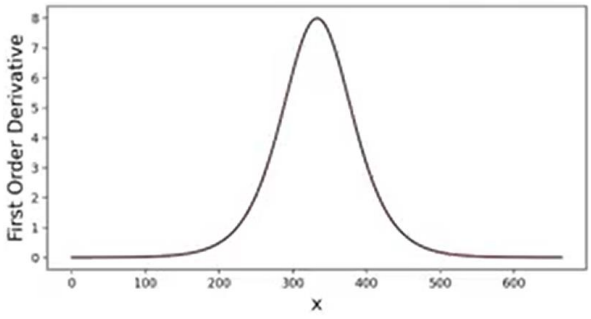
One row is plotted below



First order derivative with Sobel
(Image is scaled for illustration)



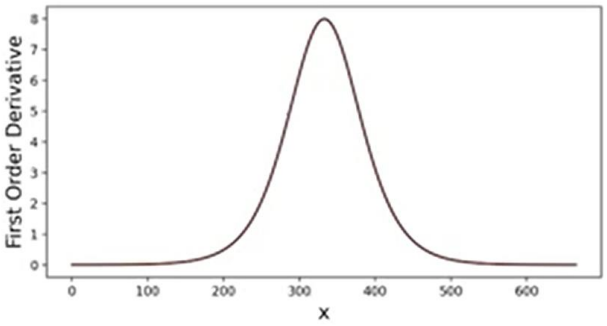
One row is plotted below



First order derivative with Sobel
(Image is scaled for illustration)

If gradient threshold is set to 2

One row is plotted below

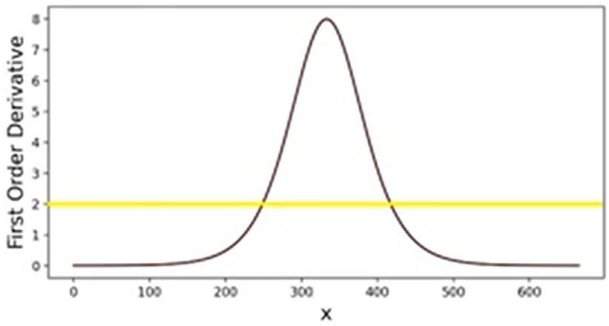


First order derivative with Sobel
(Image is scaled for illustration)

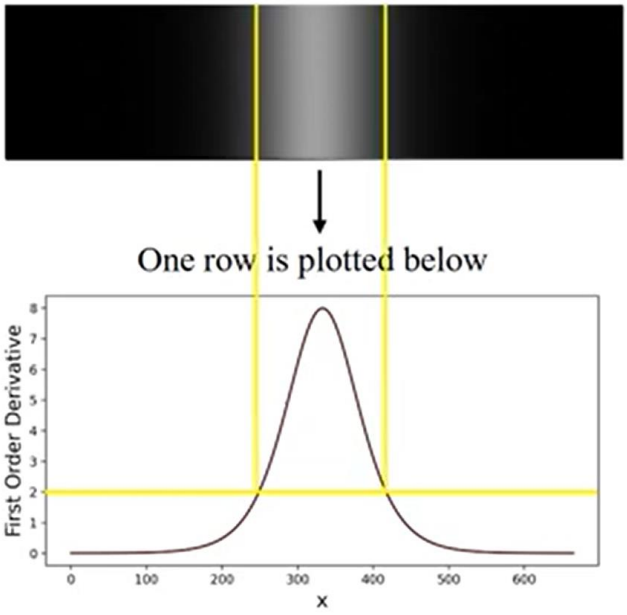
If gradient threshold is set to 2

Thickness of detected edge
is 168 pixels

One row is plotted below



First order derivative with Sobel
(Image is scaled for illustration)



If gradient threshold is set to 2

Thickness of detected edge
is 168 pixels

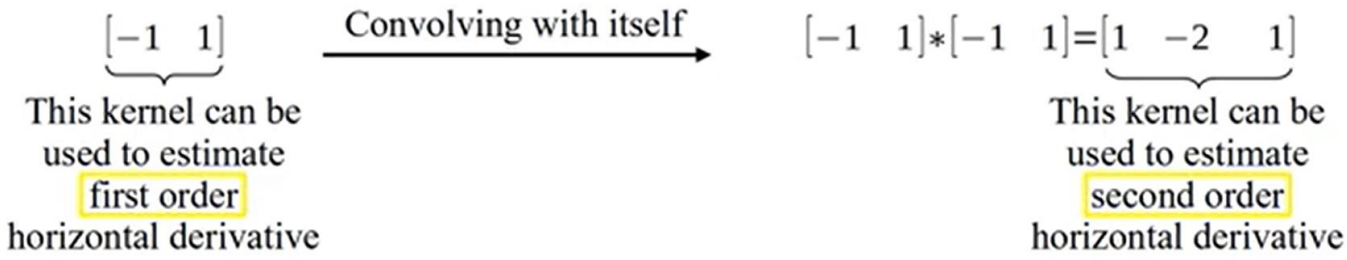
Second order derivative estimate is used
in Laplacian Edge Detection

Brightness image is a 2D matrix and Laplacian is defined as

$$\nabla^2 I(x,y) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

Second order derivative in horizontal direction

Second order derivative in vertical direction



backward derivative filter [-1 1] can be written as the odd filter [-1 1 0]

[-1 1 0] * [0 -1 1 0 0] = > [1 -2 1]

0 is padding to get the same size (1 x3) filter.

To compute Laplacian, two kernels are summed

$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

These are two versions of Laplacian kernel

Diagonal kernels can also be included

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 1 \\ 0 & -2 & 0 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

To compute Laplacian, two kernels are summed

$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

These are two versions of Laplacian kernel

Sum of coefficients is zero for each kernel

Diagonal kernels can also be included

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 1 \\ 0 & -2 & 0 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

To compute Laplacian, two kernels are summed

$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

These are two versions of Laplacian kernel

Sum of coefficients is zero for each kernel

So, convolution outputs are zero on flat regions

Diagonal kernels can also be included

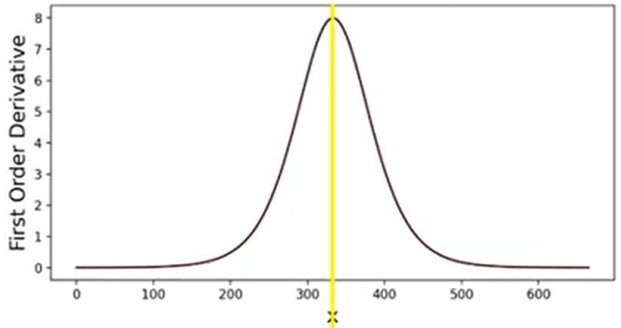
$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 1 \\ 0 & -2 & 0 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Consider this image which contains a smoothed edge

First order derivative with Sobel (Image is scaled for illustration)

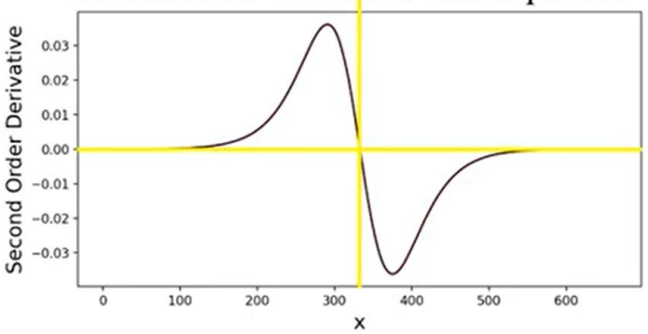
Consider same image

First order derivative with Sobel



Zero-crossing of the second order derivative indicates edge location which is 1 pixel thick

Second order derivative with Laplacian



Steps for Laplacian Edge Detection

Second order derivative is very sensitive to noise, smoothing is needed before edge detection

Gaussian and Laplacian kernels are convolved and a single kernel is obtained

Method is named as Laplacian of Gaussian (LoG)

Image is convolved with kernel

Zero-crossings are detected as edge locations

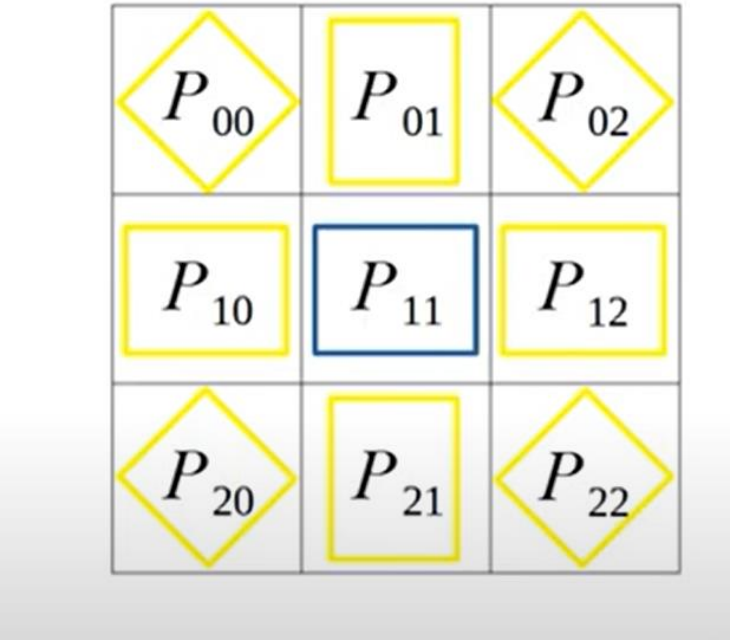
LoG needs a single pass over the image

LoG outputs thinner (1 pixel thick) edges compared to Sobel and Prewitt

Laplacian output does not provide edge direction

On Laplacian results, opposite neighbour pairs are inspected for zero corssings:

Apply the Laplacian kernel to the image.
Find the zero crossings in the Laplacian image.
Display the zero crossings image.
The zero crossings image will show the edges of the objects in the image.



- Four cases of zero-crossings:
  - {+,-}
  - {+,0,-}
  - {-,+}
  - {-,0,+}

Canny Edge detection:

Canny Edge Detector

- Smooth Image with Gaussian filter
- Compute Derivative of filtered image
- Find Magnitude and Orientation of gradient
- Apply Non-max suppression
- Apply Thresholding (Hysteresis)

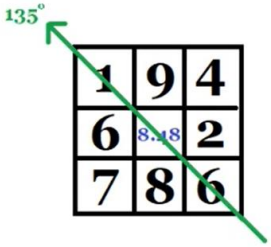
Sobel operator - example

$$I = \begin{bmatrix} 1 & 9 & 4 \\ 6 & 3 & 2 \\ 7 & 8 & 6 \end{bmatrix}, \quad G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \otimes I, \quad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \otimes I$$

$G_x = 6, G_y = -6$

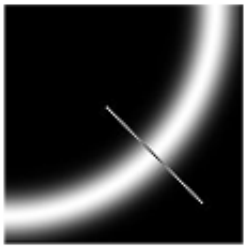
$G = \text{sqrt}(36+36) = 8.48\ldots$

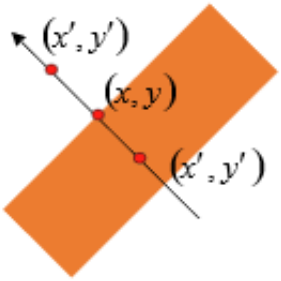
$\Theta = \arctan(-6/6) = 135^\circ$



1

Non-maximum suppression





$$M(x,y) = \begin{cases} |\nabla S|(x,y) & \text{if } |\nabla S|(x,y) > |\Delta S|(x',y') \\ & \& |\Delta S|(x,y) > |\Delta S|(x',y'') \\ 0 & \text{otherwise} \end{cases}$$

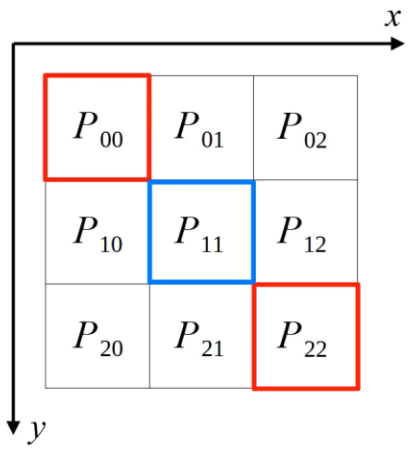
x' and x'' are the neighbors of x along normal direction to an edge

Process is performed on a 3x3 neighborhood

Image coordinate system is as shown

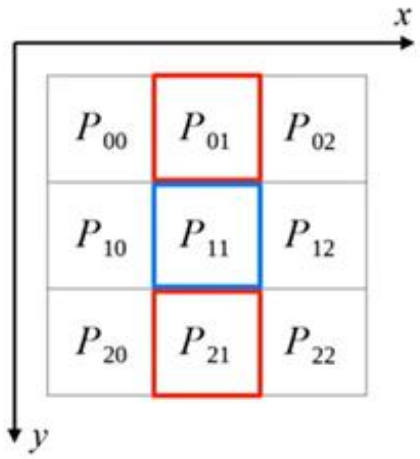
At Sobel output, gradient directions are quantized to one of the angles: 0º, 45º, 90º, 135º

Consider gradient direction of center pixel P_{11}



As a second example, let gradient direction angle of P_{11} be 135º, then magnitudes of P_{00} and P_{22} are compared with magnitude of P_{11}

If magnitude of P_{11} is less than either or both of P_{00} and P_{22}, then it is suppressed to 0, otherwise P_{11} is kept as an edge pixel



Consider gradient direction of center pixel P_{11}

As an example, let gradient direction angle of P_{11} be 90°, then magnitudes of P_{01} and P_{21} are compared with magnitude of P_{11}

If magnitude of P_{11} is less than either or both of P_{01} and P_{21}, then it is suppressed to 0, otherwise P_{11} is kept as an edge pixel

UCF Center for Research in Computer Vision

Non-maximum suppression

Before Non-Max Suppression

After Non-Max Suppression

Hysteresis Thresholding

After edge detection (with or without edge thinning), result is thresholded to remove faint details and noise

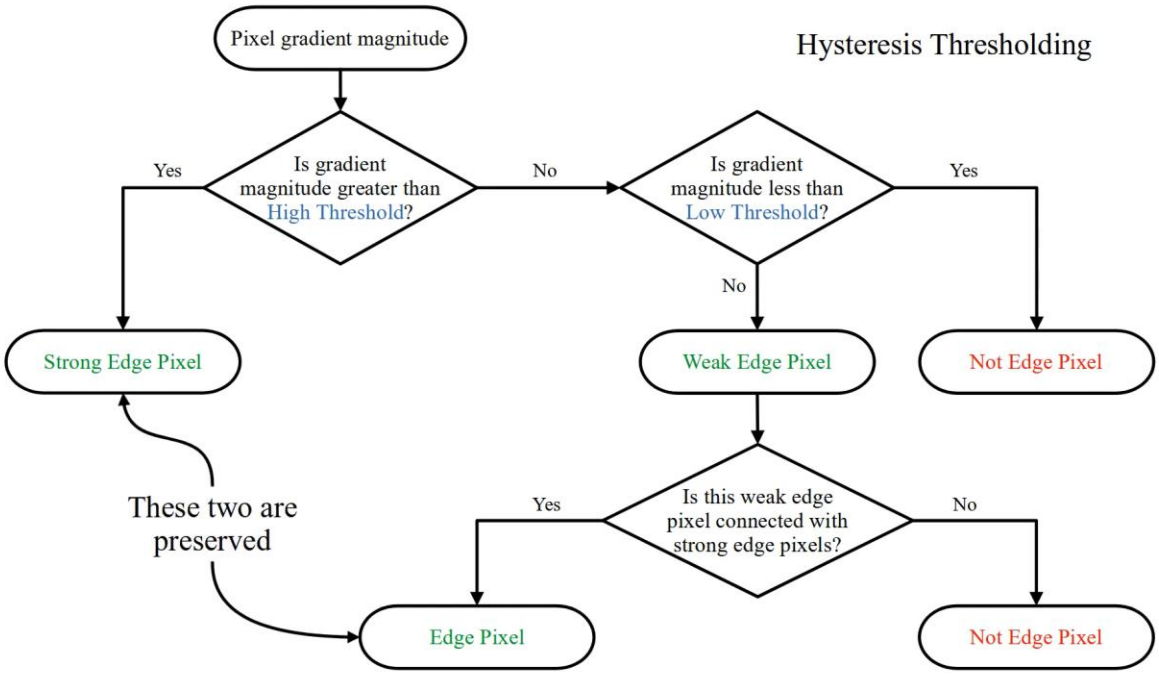
No matter how carefully the threshold is chosen, some edge pixels stay below threshold and are lost

As a result, edges are divided into edge pieces

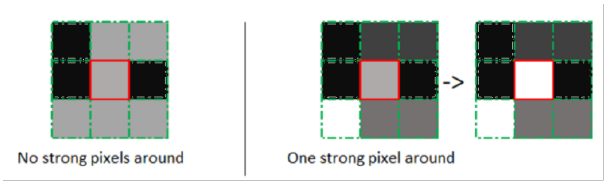
If threshold is lowered as a remedy, more noise pixels are marked as edge

Hysteresis thresholding incorporates a second threshold

This means more decision rules and higher performance on eliminating noise while keeping edges

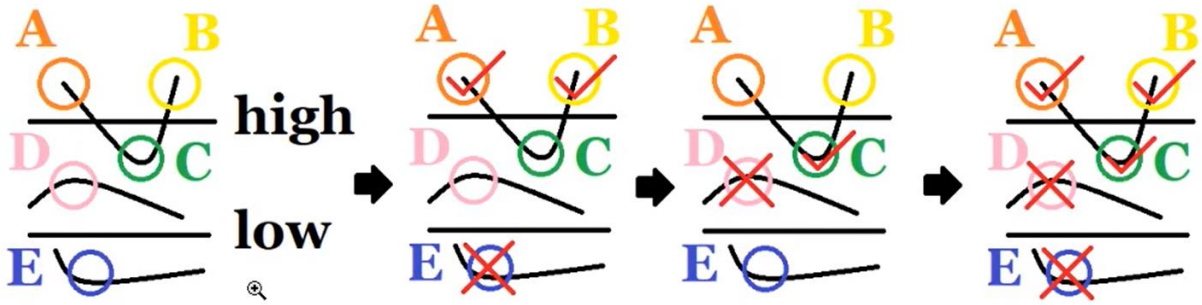


Hysteresis Thresholding [L, H]



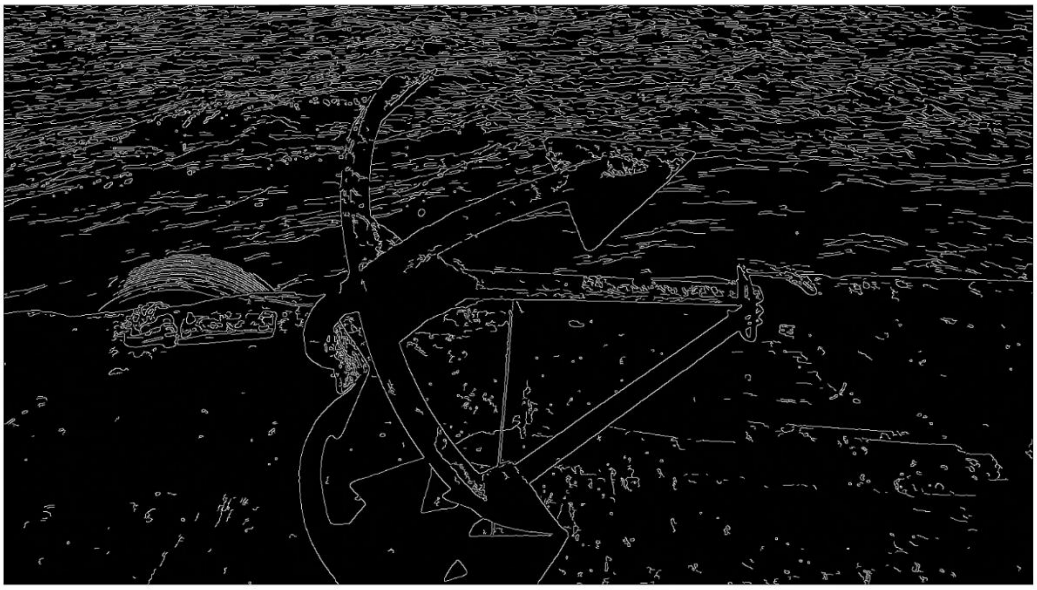
Double threshold

- apply two thresholds (high & low)
- detect potential edges, connect weak edges

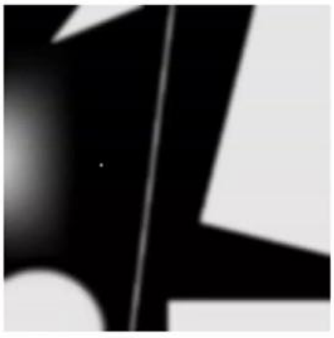


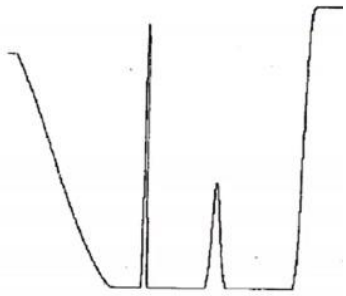
Final Canny Edges

This is the result of Canny Edge Detection



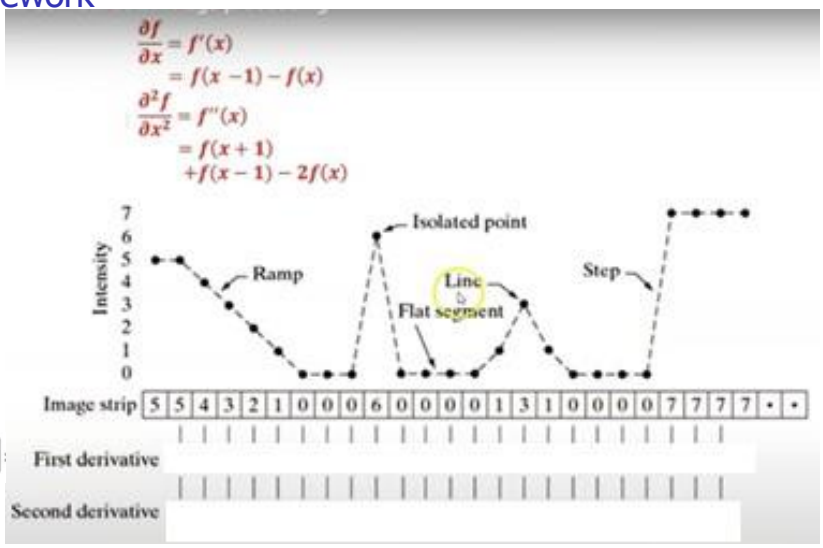
Homework





a) Image b) Horizontal Intensity profile through center of the image

Homework



$$[-1 \quad 1]$$

$$[-1 \quad 1] * [-1 \quad 1] = [1 \quad -2 \quad 1]$$

Homework

Homework:

1. Compute the firstorder and second-order derivative

2. What is the difference between first and second derivatives? Briefly explain the conclusion concerning edge detection.

Source:

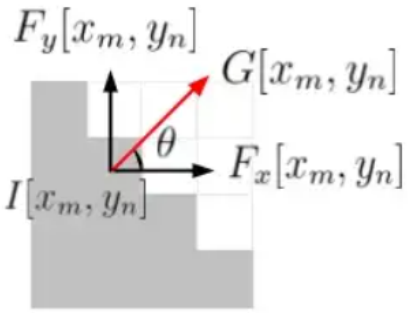
Computer Vision with Hüseyin Özdemir

https://www.youtube.com/watch?v=aCbfvxgYy8s

Extra resource/ work:

https://www.youtube.com/watch?v=kewse-JsjH0

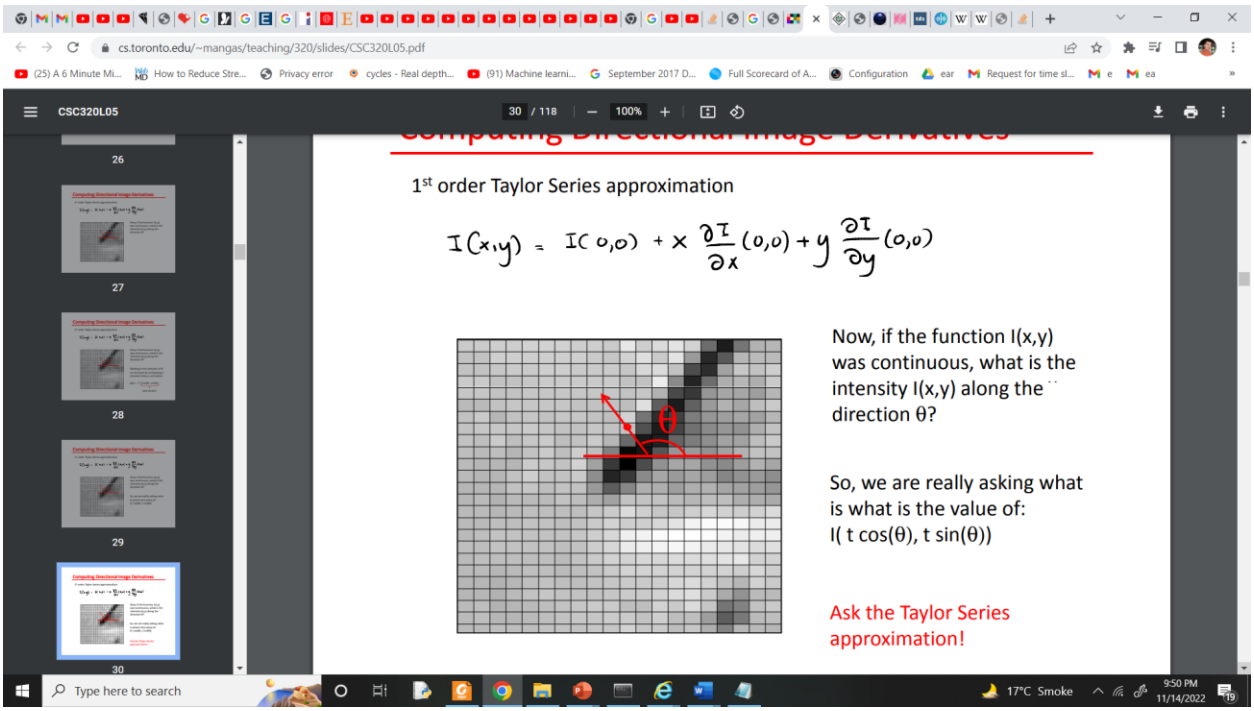
The same analysis can be carried out by considering Ky (*derivation kernel with respect to y*).



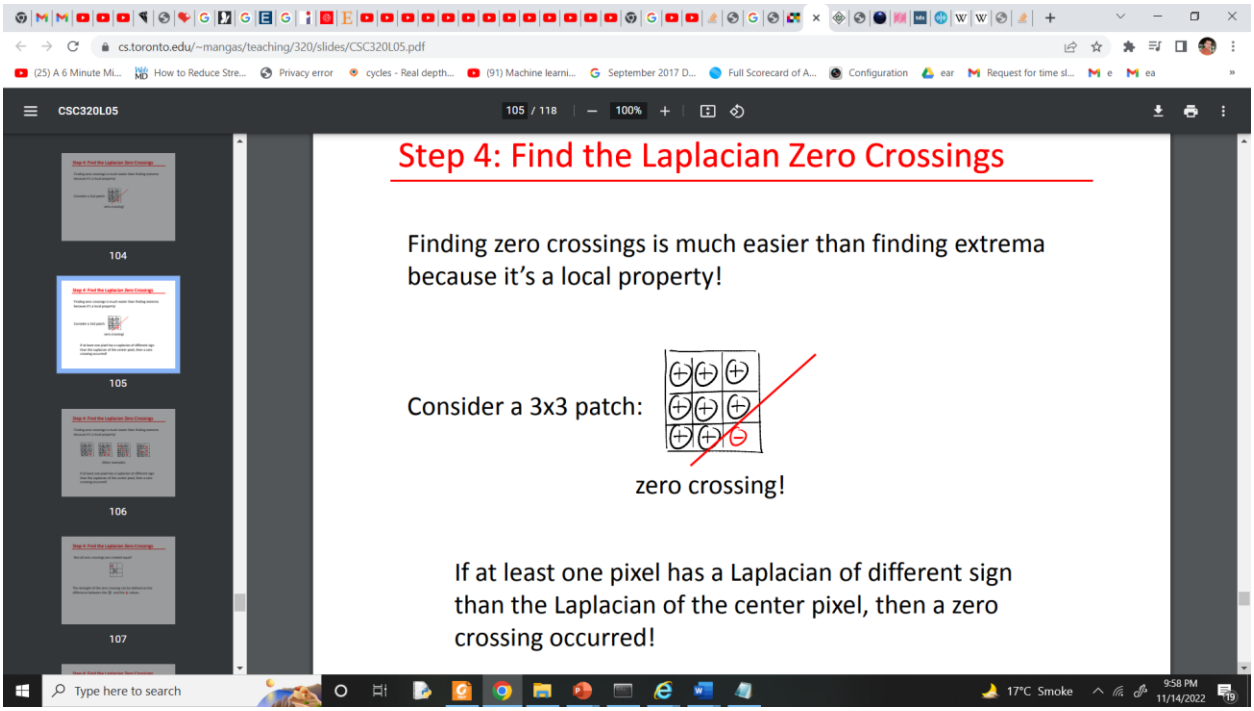
$$|G[x_m, y_n]| = \sqrt{F_x[x_m, y_n]^2 + F_y[x_m, y_n]^2}$$

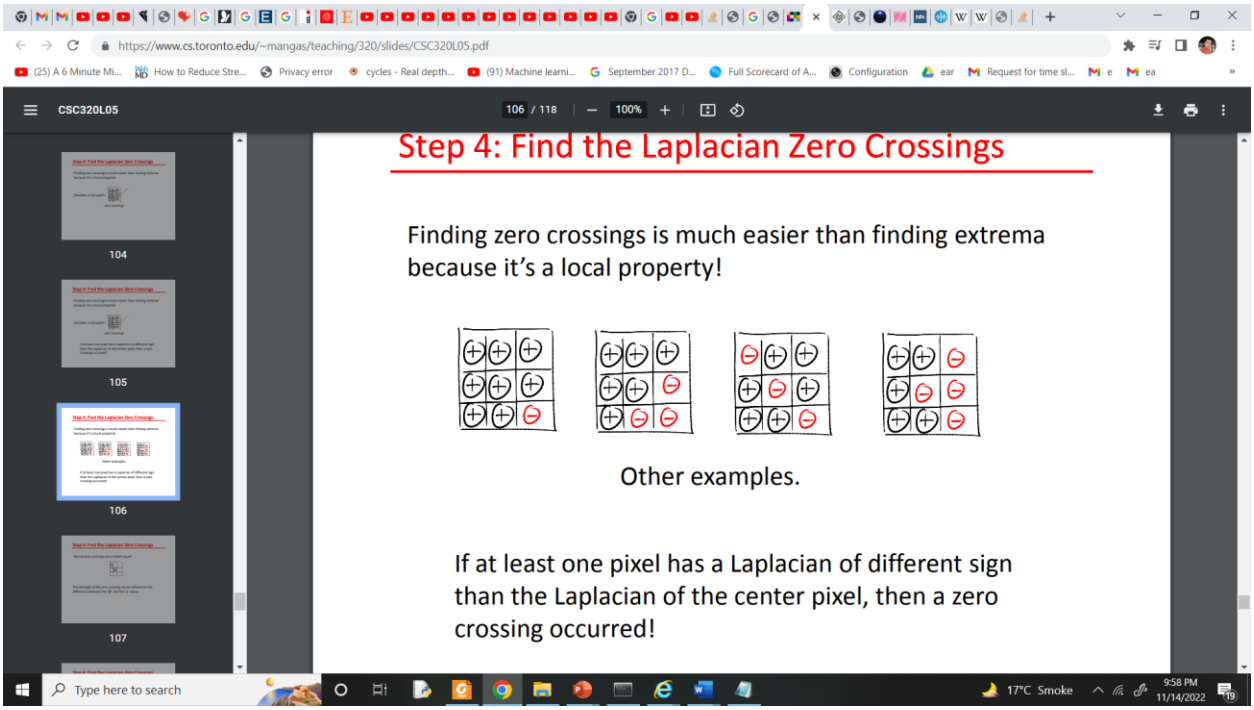
$$\theta = tan^{-1}(\frac{F_y[x_m, y_n]}{F_x[x_m, y_n]})$$

https://towardsdatascience.com/image-derivative-8a07a4118550



https://www.cs.toronto.edu/~mangas/teaching/320/slides/CSC320L05.pdf





https://dsp.stackexchange.com/questions/44928/what-does-derivative-means-in-image-processing:

Look at the numbers in the filter kernel in just **1 dimension for simplicity**.

For a **Sobel and Prewitt matrix** you have something that roughly looks like this**[-1,0,1]**.

[-1 0 1][1 0 -1] or

[-1 1 0] or [1 -1 0]

[0 1 -1] or [0 1 -1]

Convolving this with your image basically computes the difference between the pixel values of the neighboring pixels. You apply **0 to the current pixel**, 1 to the pixel on the right and -1 to the pixel on the left. This gives a first order difference:

next pixel - previous pixel, **i.e. first derivative.**

But now look at a Laplacian operator (second derivative). It looks something like **[1, -2, 1]**. **This computes the difference of differences.** To see how, note that

[1,-2,1] corresponds to => **next - 2 x current + previous**

i.e.

next - current - current + previous

i.e.

(next-current) - (current-previous) // next - previous, **i.e. first derivative.**

Now notice how this is a difference of differences.
(next - current) is like a 1st derivative. (current - previous) is like 1st derivative. Their difference is like a 2nd derivative.

f (x-1, y-1)	*f (x, y-1)*	*f (x+1, y-1)*
f (x-1, y)	*f (x, y)*	*f (x+1, y)*
f (x-1, y+1)	*f (x, y+1)*	*f (x+1, y+1)*

*

0	**1**	**0**
1	**-4**	**1**
0	**1**	**0**

$$f''(x,y) = 1*f(x, y-1) + 1*f(x-1, y) - 4*f(x, y) + 1*f(x+1, y) + 1*f(x, y+1)$$

$$f''(x, y) = f(x, y-1) + f(x-1, y) - 4f(x, y) + f(x+1, y) + f(x,y+1)$$

$$f''(x, y) = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x-1, y) - 4f(x, y)$$