ECE 478-578
Intelligent Robotics I

PhD. Husnu Melih Erdogan – Electrical & Computer Engineering

herdogan@pdx.edu

Teaching Assistant

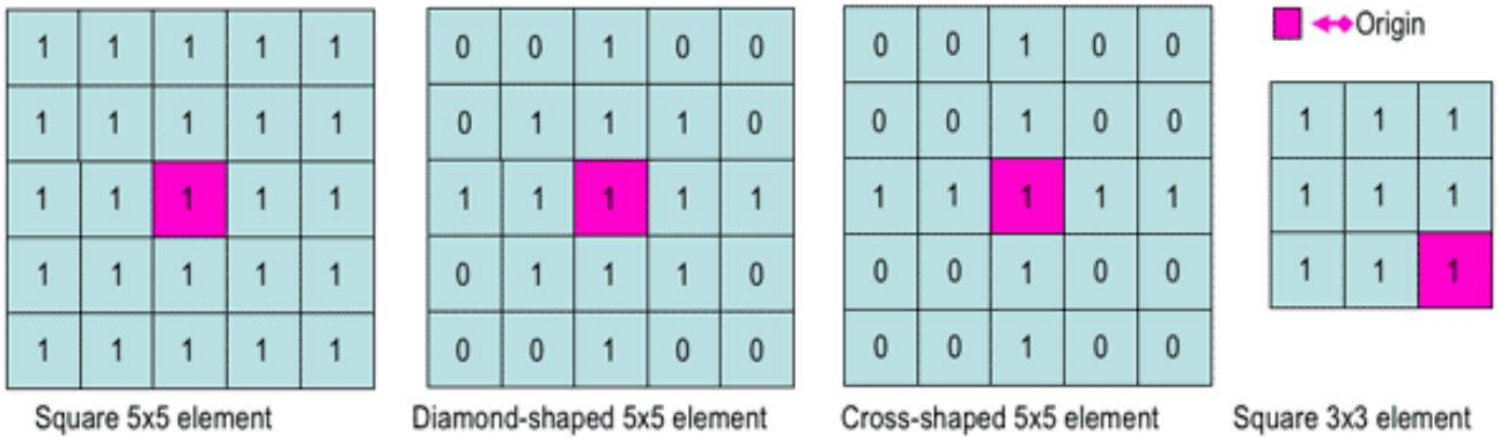
Portland State
UNIVERSITY

Introduction to OpenCV 3 – Part 2

Portland State
UNIVERSITY

Basic Structuring Element

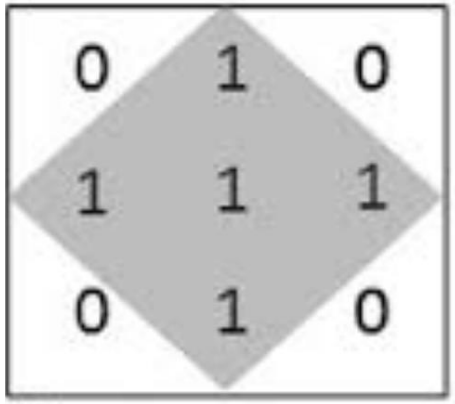
- Simply a binary image
- The matrix dimensions specify the *size* of the structuring element.
- The pattern of ones and zeros specifies the *shape* of the structuring element.
- An *origin* of the structuring element is usually one of its pixels, although generally the origin can be outside the structuring element.



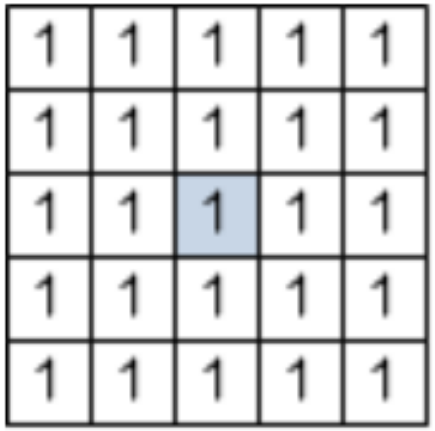
Examples of simple structuring elements.

https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm

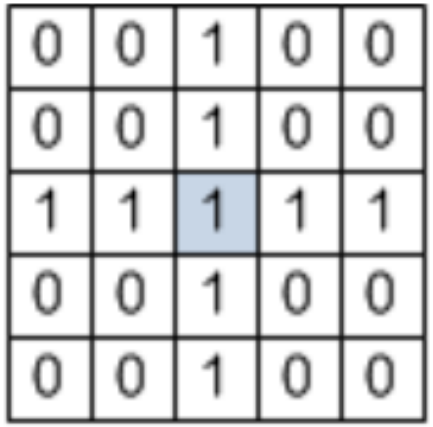
Different Shape Structuring Elements



Diamond



Square



Cross

```
[1 0 0 0 1]
[0 1 0 1 0]
[0 0 1 0 0]
[0 1 0 1 0]
[1 0 0 0 1]
```

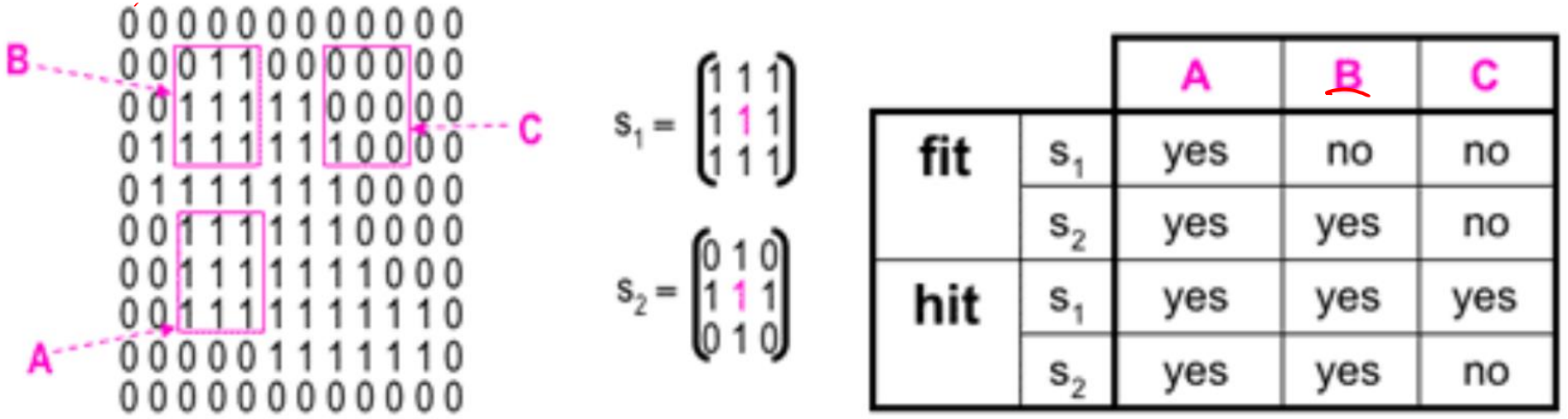
X

Portland State UNIVERSITY

Hit and Fit

When a structuring element is placed in a binary image, each of its pixels is associated with the corresponding pixel of the neighborhood under the structuring element.

- The structuring element is said to **fit** the image if, for each of its pixels set to 1, the corresponding image pixel is also 1.
- Similarly, a structuring element is said to **hit**, or intersect, an image if, at least for one of its pixels set to 1 the corresponding image pixel is also 1.



		A	B	C
fit	s_1	yes	no	no
	s_2	yes	yes	no
hit	s_1	yes	yes	yes
	s_2	yes	yes	no

Fitting and hitting of a binary image with structuring elements s_1 and s_2.

https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm

Grayscale Morphology

Dilation:

$$(I \oplus H)(u,v) = \max_{(i,j)\in H} \{I(u+i, v+j) + H(i,j)\}$$

Erosion:

$$(I \ominus H)(u,v) = \min_{(i,j)\in H} \{I(u+i, v+j) + H(i,j)\}$$

http://www.coe.utah.edu/~cs4640/slides/Lecture11.pdf

Morphological Operations - Dilation

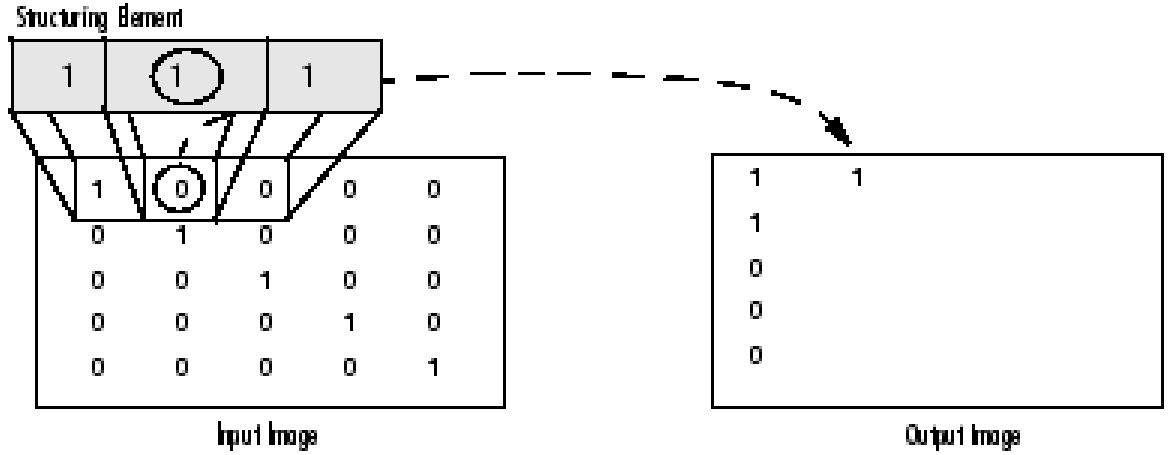
- Enlarges object by adding boundary pixels to object
- Fill in small holes in object
- If structuring element hits result is 1

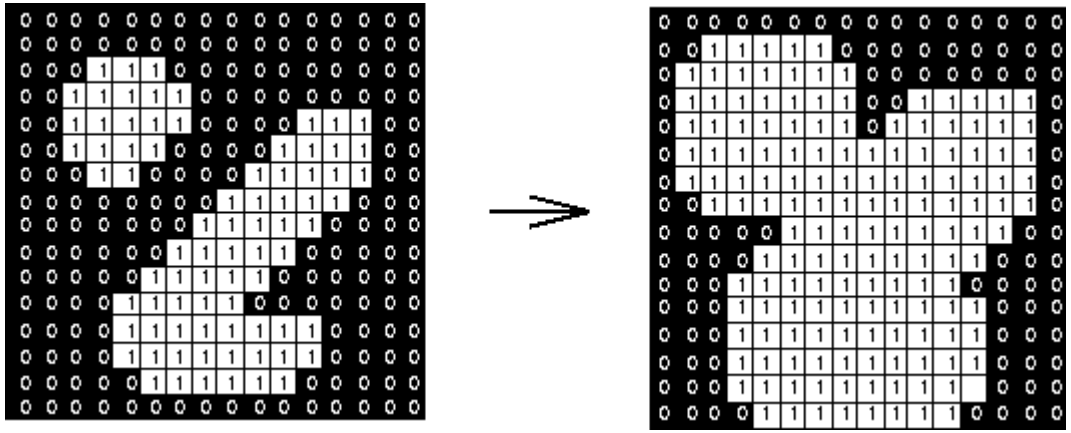
denoted by f ⊕ s

$$g(x,y) = \begin{cases} 1, & \text{if } s \text{ hits } f \\ 0, & \text{otherwise} \end{cases}$$

http://www.csie.nuk.edu.tw/~ayen/teach/cv/cv-note04.pdf

Morphological Operations - Dilation





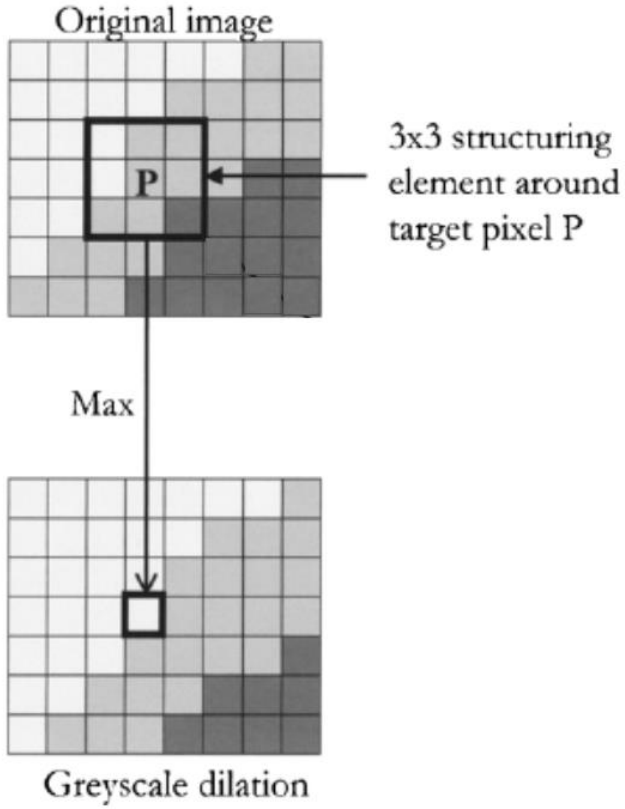


Image Source

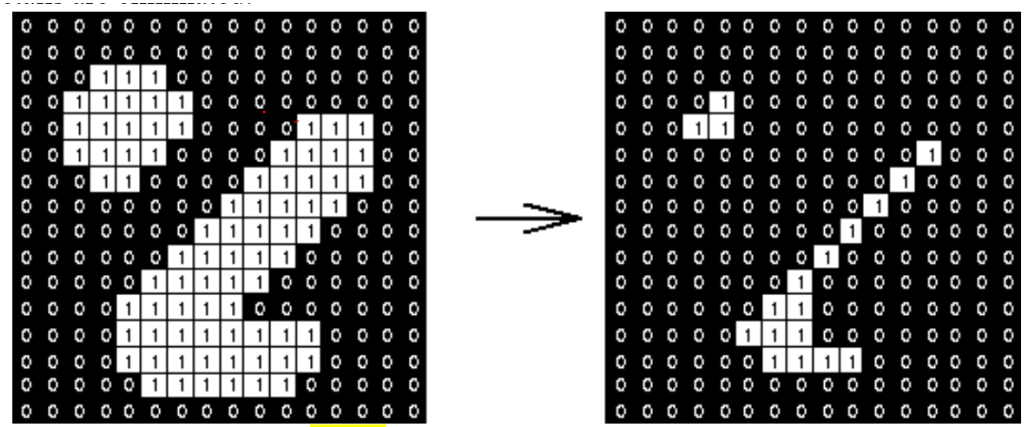
Morphological Operations - Erosion

- It has the effect of stripping away boundary pixels
- It enlarges holes in object
- It removes unwanted small-scale features
- It reduces size of other features
- If structuring element fits put 1

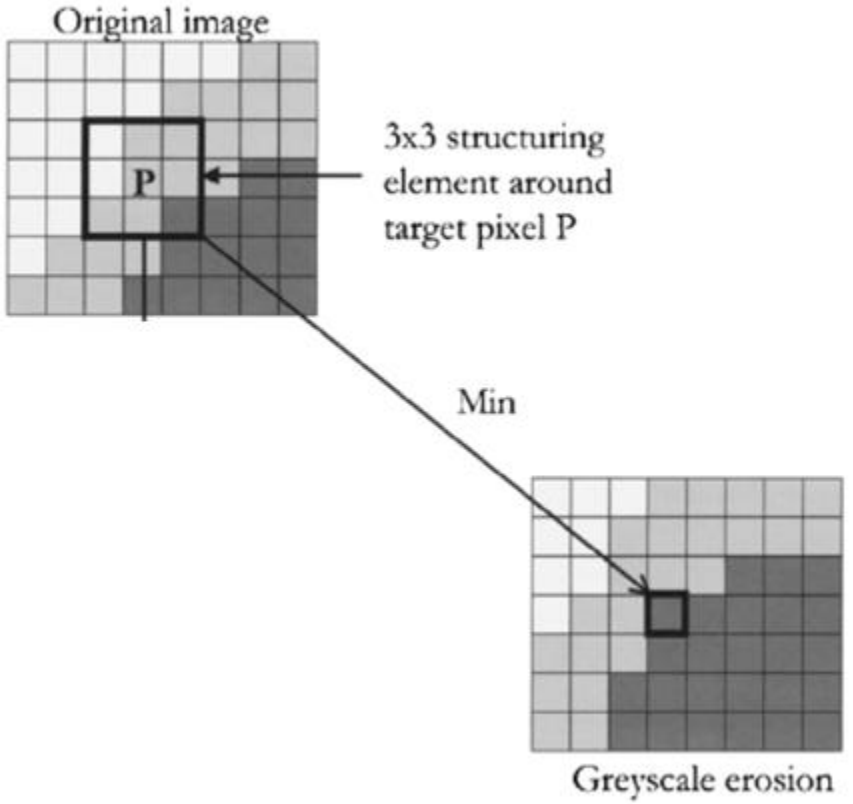
$$g(x,y) = \begin{cases} 1, & \text{if } s \text{ fits } f \\ 0, & \text{otherwise} \end{cases}$$

https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm

Morphological Operations - Erosion



Erosion: a 3×3 square structuring element
(www.cs.princeton.edu/~pshilane/class/mosaic/).



https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm

Morphological Operations - Opening

- An opening is defined as an **erosion followed by a dilation** using the same structuring element for both operations.
- Grayscale opening consists simply of a Grayscale erosion followed by a Grayscale dilation.
- Small bright regions are removed
- And white regions are more isolated

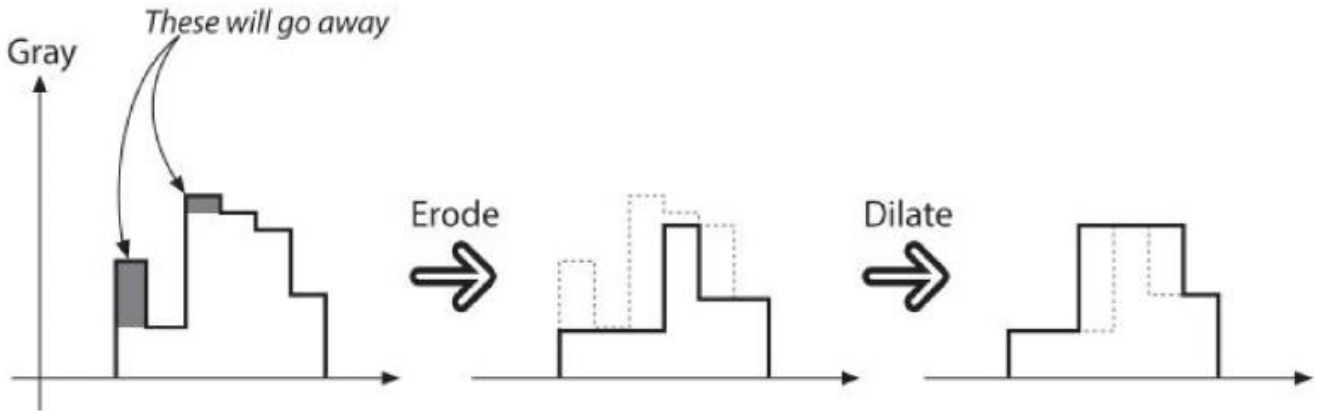


Figure 10-25. Morphological opening operation applied to a (one-dimensional) non-Boolean image: the upward outliers are eliminated

https://homepages.inf.ed.ac.uk/rbf/HIPR2/open.htm

Portland State UNIVERSITY

Morphological Operations - Closing

- An closing is defined as an dilation followed by a erosion using the same structuring element for both operations.
- Grayscale opening consists simply of a grayscale dilation followed by a grayscale erosion.
- Removes unwanted noisy segments
- Bright regions are joined but retain basic size

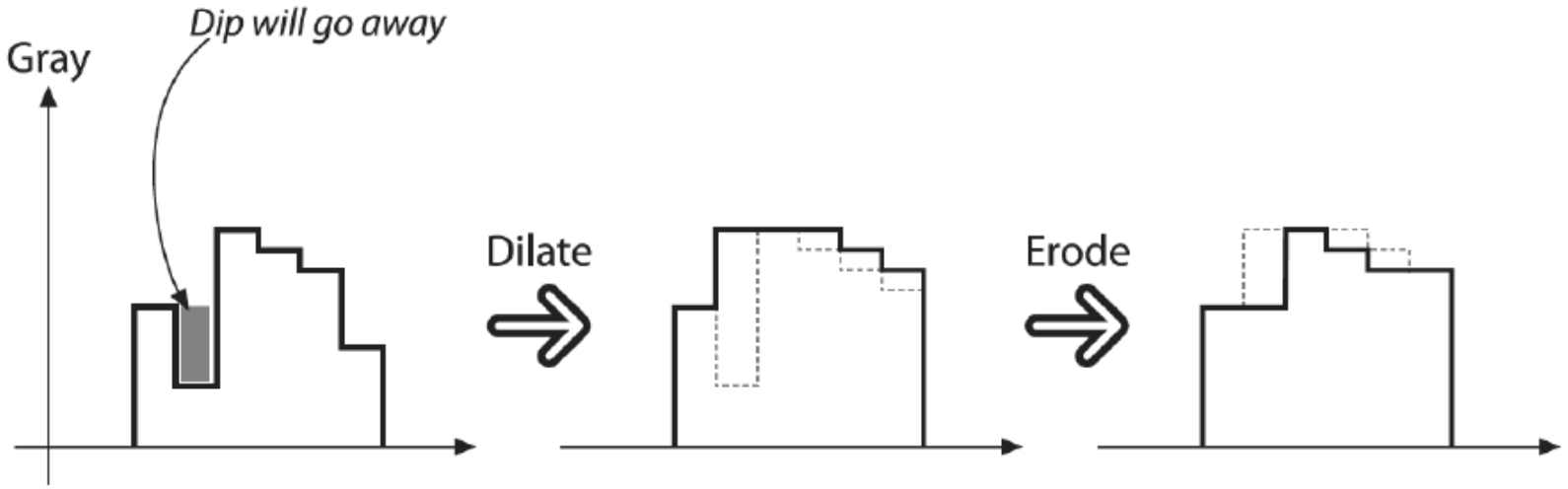


Figure 5-12. Morphological closing operation: the downward outliers are eliminated as a result

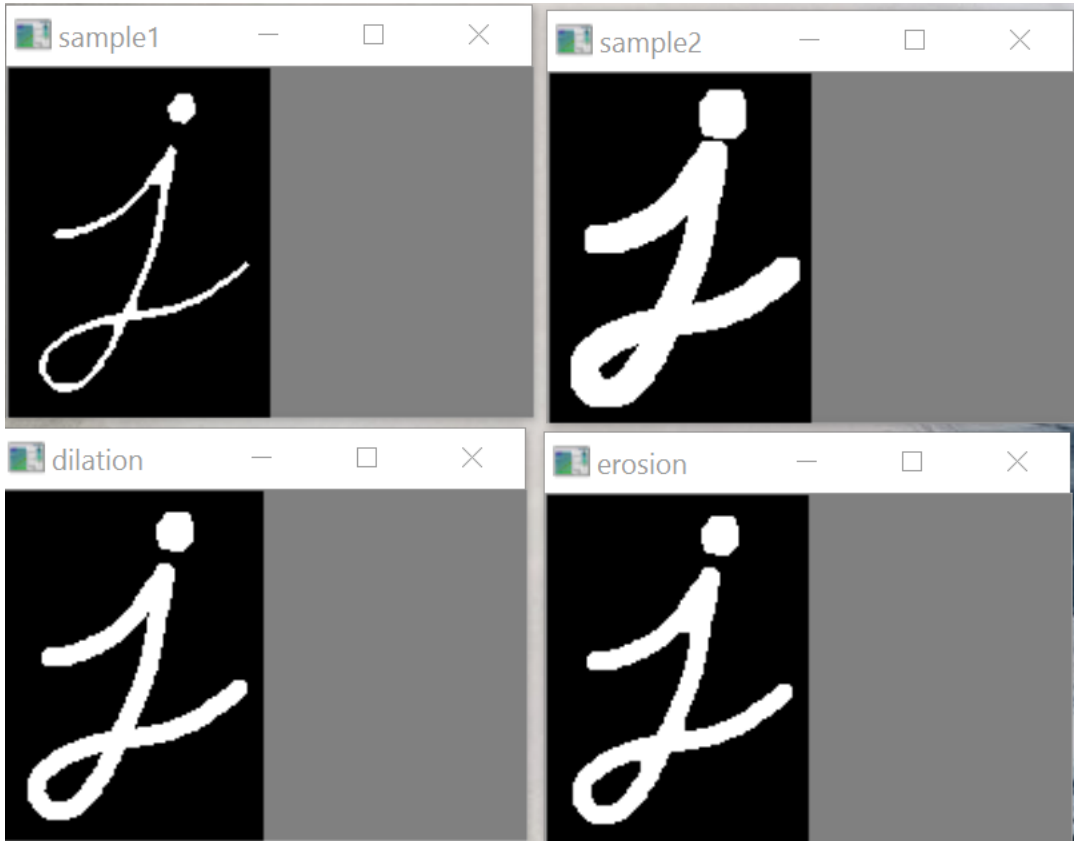
Portland State UNIVERSITY

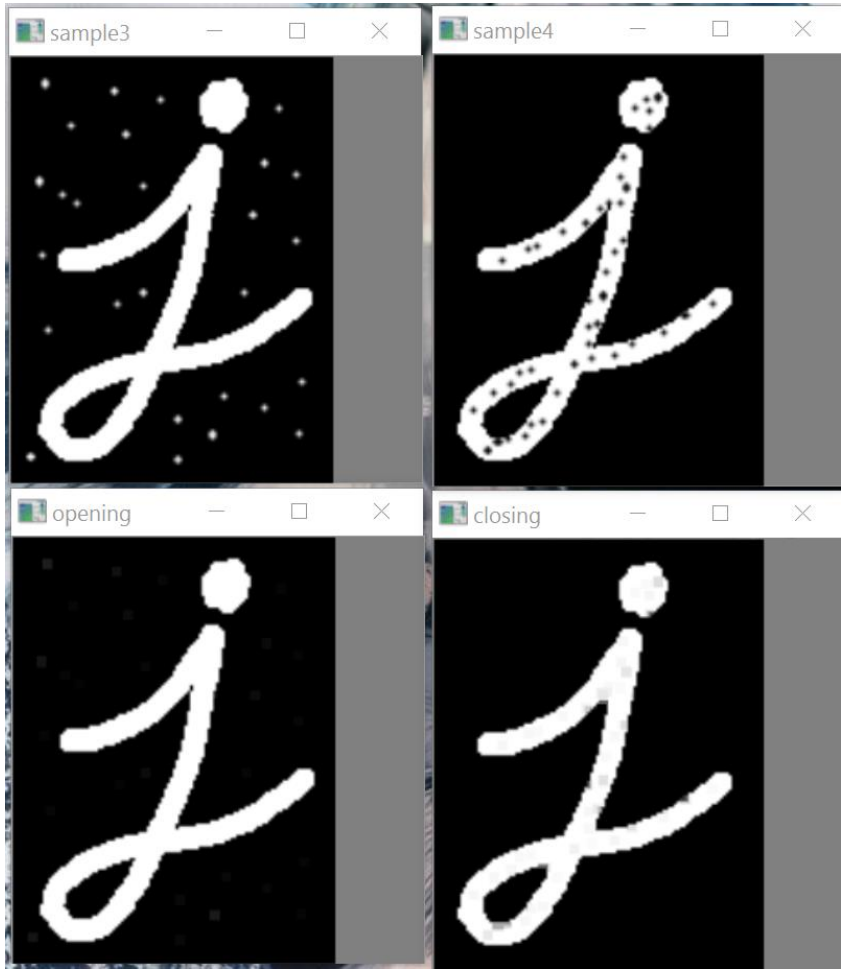
Morphological Operations

```
import cv2
import numpy as np

img1 = cv2.imread('sample1.png',0)
cv2.imshow("sample1", img1)

img2 = cv2.imread('sample2.png',0)
cv2.imshow("sample2", img2)

img3 = cv2.imread('sample3.png',0)
cv2.imshow("sample3", img3)

img4 = cv2.imread('sample4.png',0)
cv2.imshow("sample4", img4)

#5x5 structuring elemnent all ones
kernel = np.ones((5,5),np.uint8)

#dilation
dilation = cv2.dilate(img1,kernel,iterations = 1)
cv2.imshow("dilation", dilation)

#erosion
erosion = cv2.erode(img2,kernel,iterations = 1)
cv2.imshow("erosion", erosion)
```

```
#opening
opening = cv2.morphologyEx(img3, cv2.MORPH_OPEN, kernel)
cv2.imshow("opening", opening)

#closing
closing = cv2.morphologyEx(img4, cv2.MORPH_CLOSE, kernel)
cv2.imshow("closing", closing)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Morphological Operations

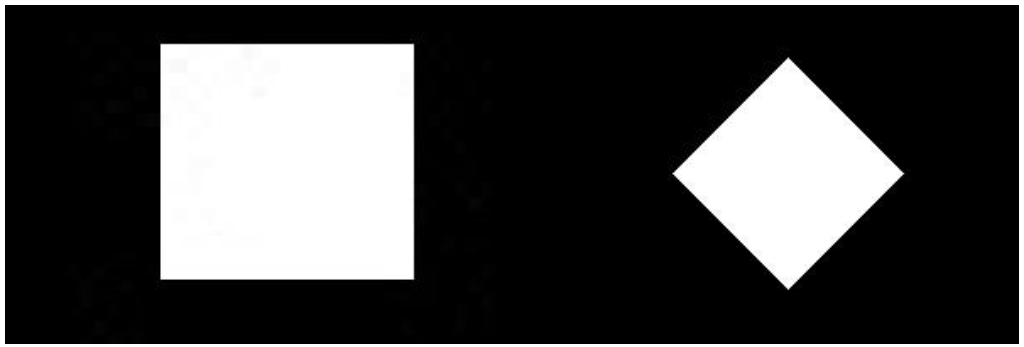




Portland State UNIVERSITY

Finding Corners

```
import cv2
import numpy as np
# global variables
img = None
ret = None
threshold = None
se_square = None
se_cross = None
se_diamond = None
se_x = None
# load the image and conver it to a binary image
def load_images():
    global img, ret, threshold
    img = cv2.imread('corner_binary.jpg')
    cv2.imshow('original image',img)
    ret, threshold = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)
```



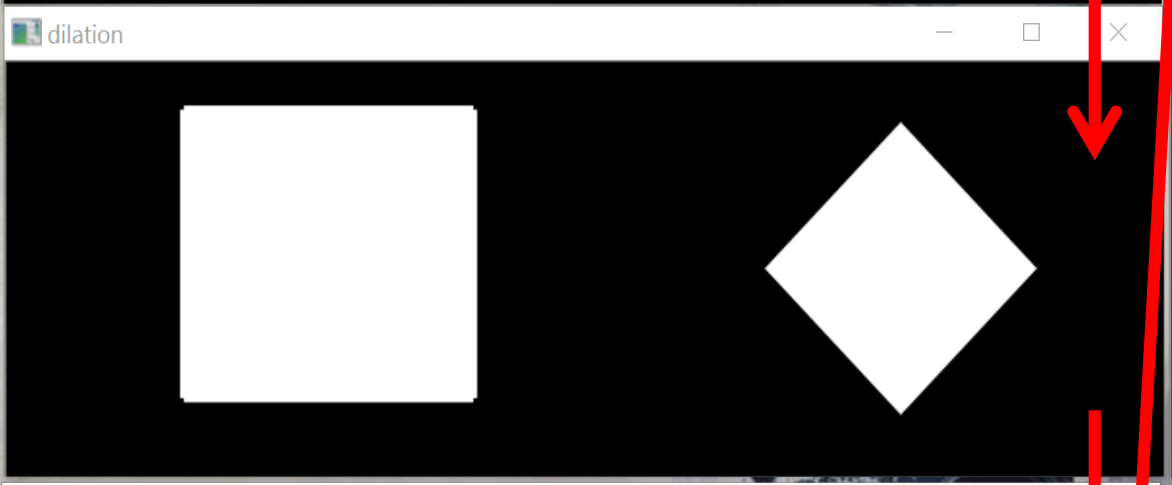
Finding Corners – Cont.

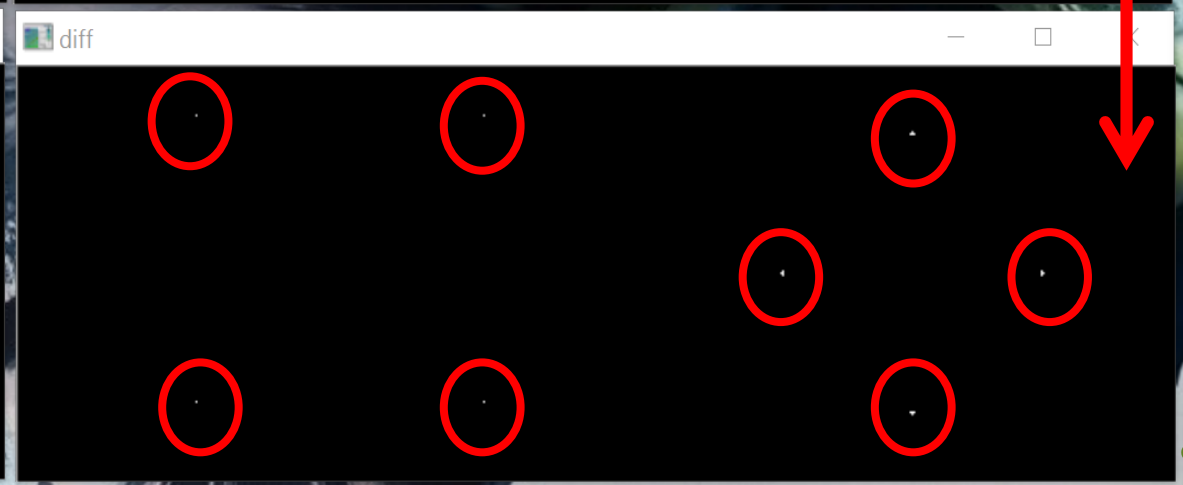
```
# create structing elements for the morphological operations
def create_structuring_elements():
    global se_square, se_cross, se_diamond, se_x

    se_square= np.matrix([[1, 1, 1, 1, 1], [1, 1, 1, 1, 1], [1, 1, 1, 1, 1], [1, 1, 1, 1, 1], [1, 1, 1, 1, 1]])
    print "se_square"
    print se_square
    se_cross= np.matrix([[0, 0, 1, 0, 0], [0, 0, 1, 0, 0], [1, 1, 1, 1, 1], [0, 0, 1, 0, 0], [0, 0, 1, 0, 0]])
    print "se_cross"
    print se_cross
    se_diamond= np.matrix([[0, 0, 1, 0, 0], [0, 1, 1, 1, 0], [1, 1, 1, 1, 1], [0, 1, 1, 1, 0], [0, 0, 1, 0, 0]])
    print "se_diamond"
    print se_diamond
    se_x= np.matrix([[1, 0, 0, 0 ,1], [0, 1, 0, 1, 0], [0, 0, 1, 0, 0], [0, 1, 0, 1, 0], [1, 0, 0, 0, 1]])
    print "se_x"
    print se_x
```

Portland State UNIVERSITY

Finding Corners – Cont.

```
# detect corners by morphological operations
def corner_detection():
    #global ret, threshold
    result1 = cv2.dilate(threshold,se_cross,iterations = 1)
    cv2.imshow('dilation',result1)

    result1 = cv2.erode(result1,se_diamond,iterations = 1)
    cv2.imshow('erode',result1)

    result2 = cv2.dilate(threshold,se_x,iterations = 1)
    cv2.imshow('dilation2', result2)

    result2 = cv2.erode(result2,se_square,iterations = 1)
    cv2.imshow('erode2',result2)

    diff = cv2.absdiff(result1,result2);
    cv2.imshow('diff',diff)

load_images()
create_structuring_elements()
corner_detection()

cv2.waitKey(0)
cv2.destroyAllWindows()
```





*Python 2.7.13 S

File Edit Shell

```
Python 2.7.13
AMD64)] on wi
Type "copyrig
>>>
 RESTART: C:/
ind_corners.p
se_square
[[1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]]
se_cross
[[0 0 1 0 0]
 [0 0 1 0 0]
 [1 1 1 1 1]
 [0 0 1 0 0]
 [0 0 1 0 0]]
se_diamond
[[0 0 1 0 0]
 [0 1 1 1 0]
 [1 1 1 1 1]
 [0 1 1 1 0]
 [0 0 1 0 0]]
se_x
[[1 0 0 0 1]
 [0 1 0 1 0]
 [0 0 1 0 0]
 [0 1 0 1 0]
 [1 0 0 0 1]]
```

Portland State UNIVERSITY

Filters

- Filtering is a technique for modifying or enhancing an image.
- You can filter an image to emphasize certain features or remove other features.
- Image processing operations implemented with filtering include smoothing, sharpening, and edge enhancement.
- Filtering is a *neighborhood operation,* in which the value of any given pixel in the output image is determined by applying some algorithm to the values of the pixels in the neighborhood of the corresponding input pixel. A pixel's neighborhood is some set of pixels, defined by their locations relative to that pixel.

Correlation and Convolution

Correlation and Convolution

- Correlation: $G = H \otimes F$

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i+u, j+v]$$

- Convolution: $G = H * F$

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i-u, j-v]$$

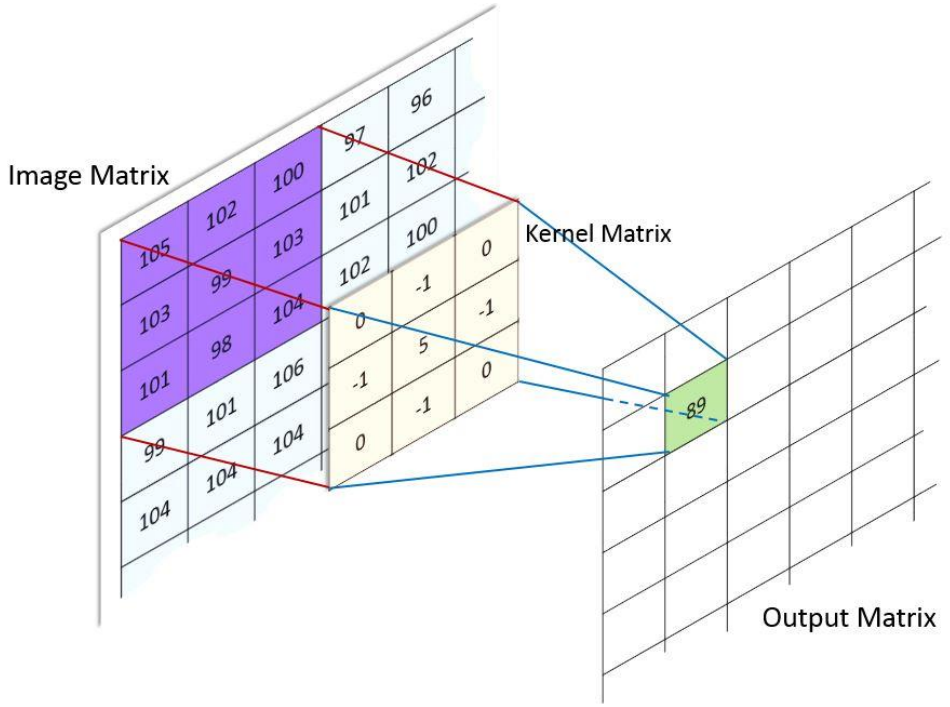
- If you want to learn more about correlation and convolution
  - http://www.cs.umd.edu/~djacobs/CMSC426/Convolution.pdf

https://web.eecs.umich.edu/~jjcorso/t/598F14/files/lecture_0924_filtering.pdf

Convolution

- Flip the kernel both **horizontally** and **vertically**. As our selected kernel is symmetric, the flipped kernel is equal to the original.
- Multiply each element of the kernel with its corresponding element of the image matrix (the one which is overlapped with it)
- Sum up all product outputs and put the result at the same position in the output matrix as the center of kernel in image matrix.

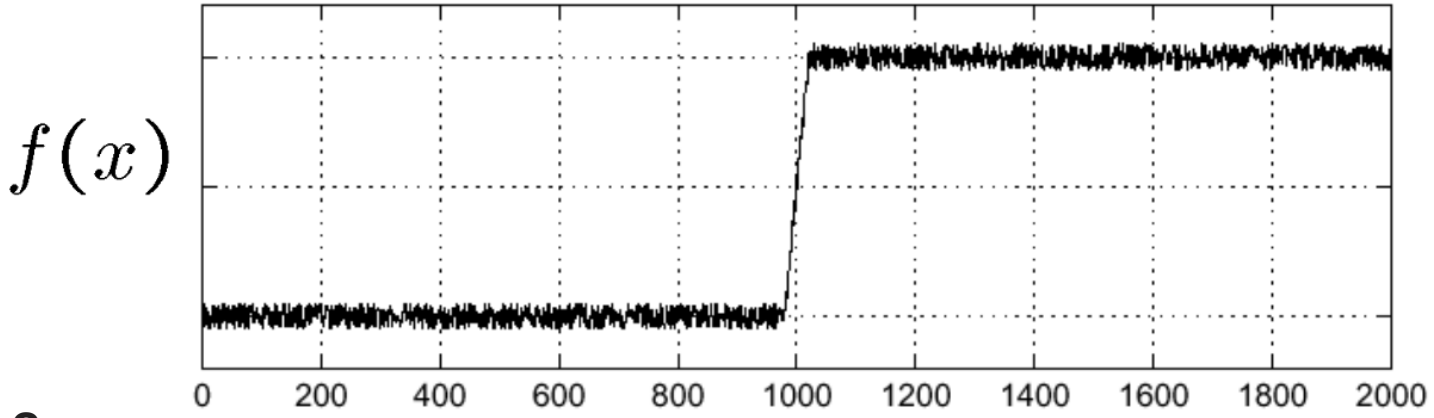
$$Kernel = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$



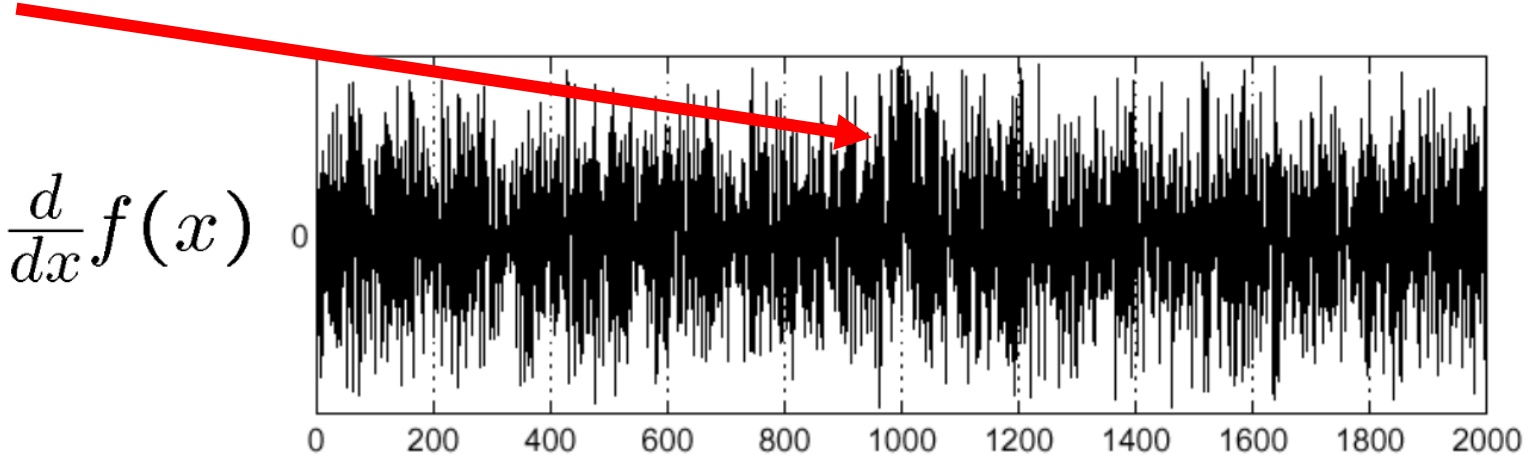
$$105 * 0 + 102 * -1 + 100 * 0 + 103 * -1 + 99 * 5 + 103 * -1 + 101 * 0 + 98 * -1 + 104 * 0 = 89$$

Portland State UNIVERSITY

Effects of Noise



Where is the edge?



High Pass and Low Pass Filters

- A low-pass filter is a filter that passes low-frequency signals and attenuates signals with frequencies higher than the cut-off frequency. The actual amount of attenuation for each frequency varies depending on specific filter design.

- A high-pass filter is a filter that passes high frequencies well, but attenuates frequencies lower than the cut-off frequency. Sharpening is fundamentally a high pass operation in the frequency domain.

http://research.ijcaonline.org/volume109/number14/pxc3900999.pdf

Mean and Median Filter

Mean Filter

Median Filter

http://www.bogotobogo.com/Matlab/Matlab_Tutorial_Digital_Image_Processing_6_Filter_Smoothing_Low_Pass_fspecial_filter2.php

Portland State UNIVERSITY

Blurring

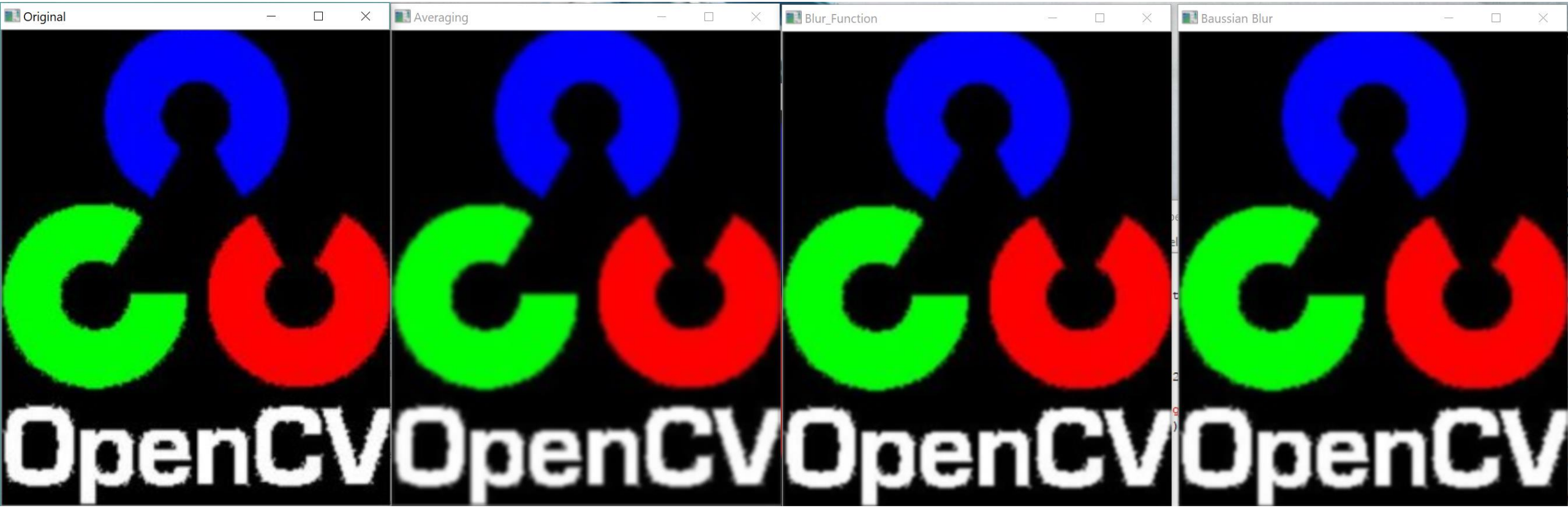
$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

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

img = cv2.imread('input.png')

#Create a kernel
kernel = np.ones((5,5),np.float32)/25

#Apply custom-made filter to theimage
custom = cv2.filter2D(img,-1,kernel)

# Use blur function
blur = cv2.blur(img,(3,3))

# Use Gaussian Blur function
gaussianblur = cv2.GaussianBlur(img,(5,5),0)

cv2.imshow('Original', img)
cv2.imshow('Averaging', custom)
cv2.imshow('Blur_Function', blur)
cv2.imshow('Baussian Blur', gaussianblur)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Portland State UNIVERSITY

Blurring



$$K = \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

blur = cv2.blur(img,(3,3))

gaussianblur = cv2.GaussianBlur(img,(5,5),0)

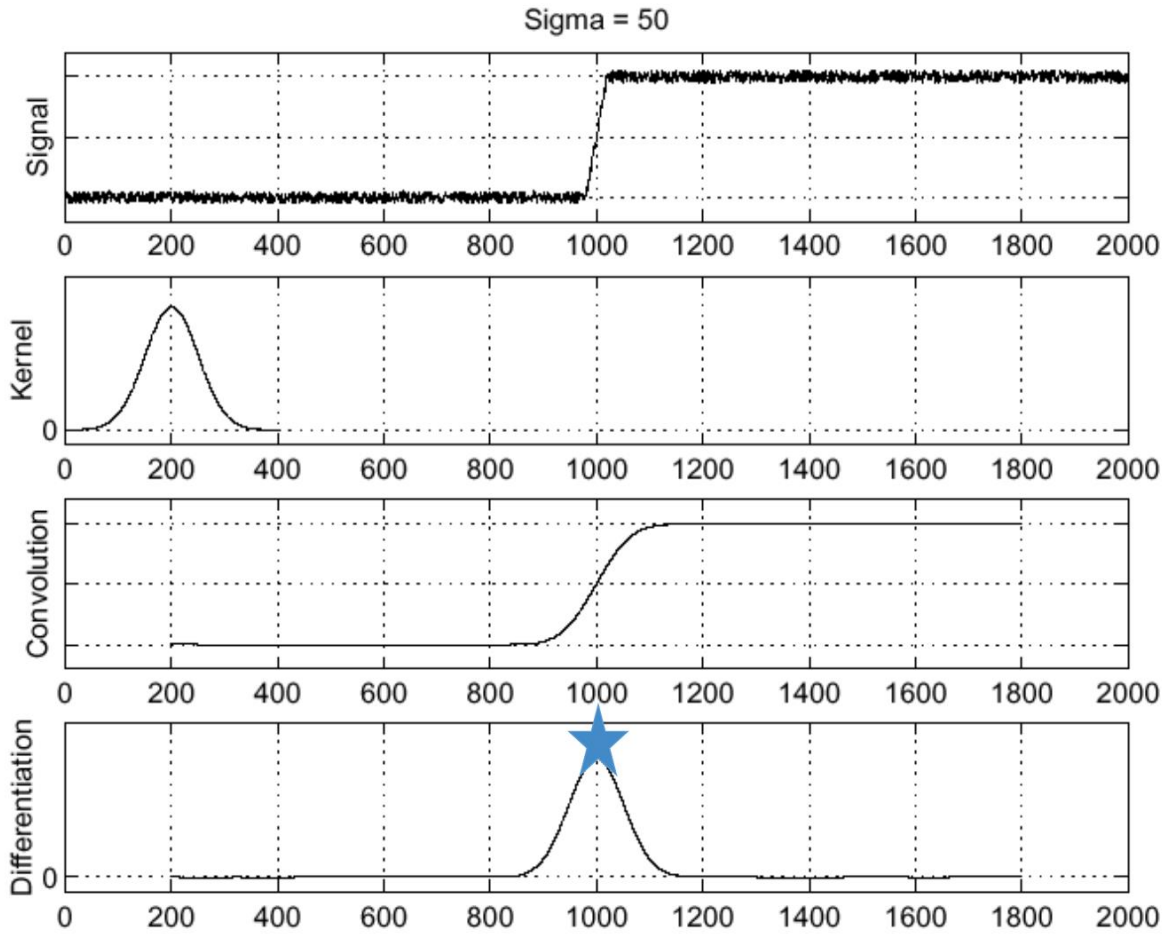
Low Pass Filters - Gaussian

f

h

$h \star f$

$\frac{\partial}{\partial x}(h \star f)$



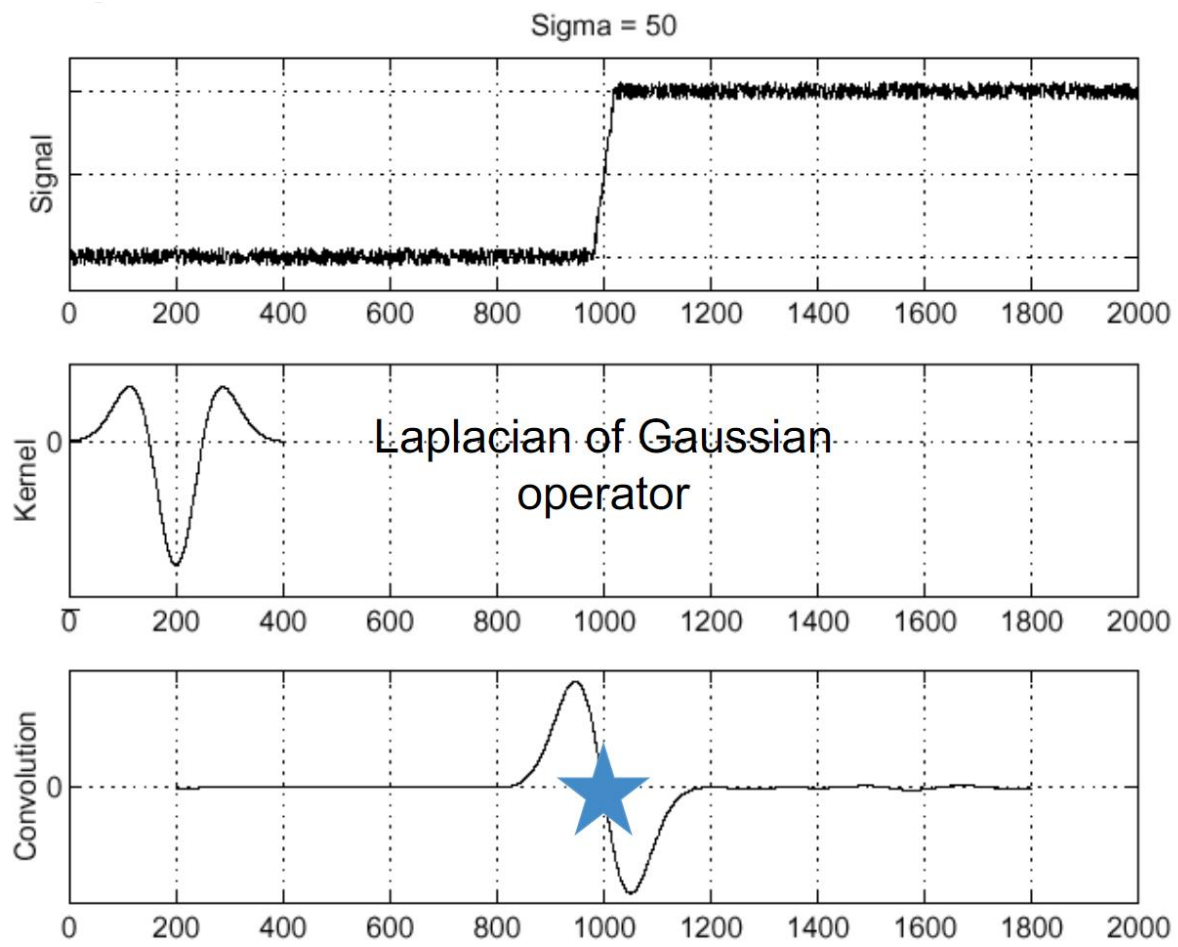
Where is the edge? Look for peaks in $\frac{\partial}{\partial x}(h \star f)$

Low Pass Filters – Laplacian of Gaussian

f

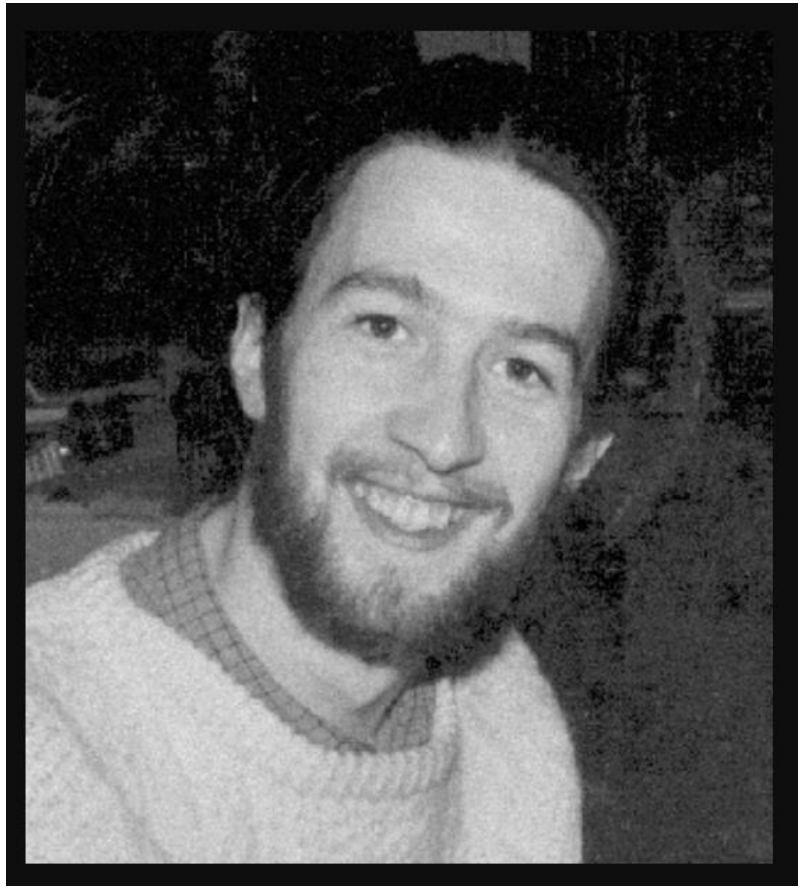
$\frac{\partial^2}{\partial x^2} h$

$(\frac{\partial^2}{\partial x^2} h) \star f$



Portland State
UNIVERSITY

Low Pass Filters - Gaussian



https://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm

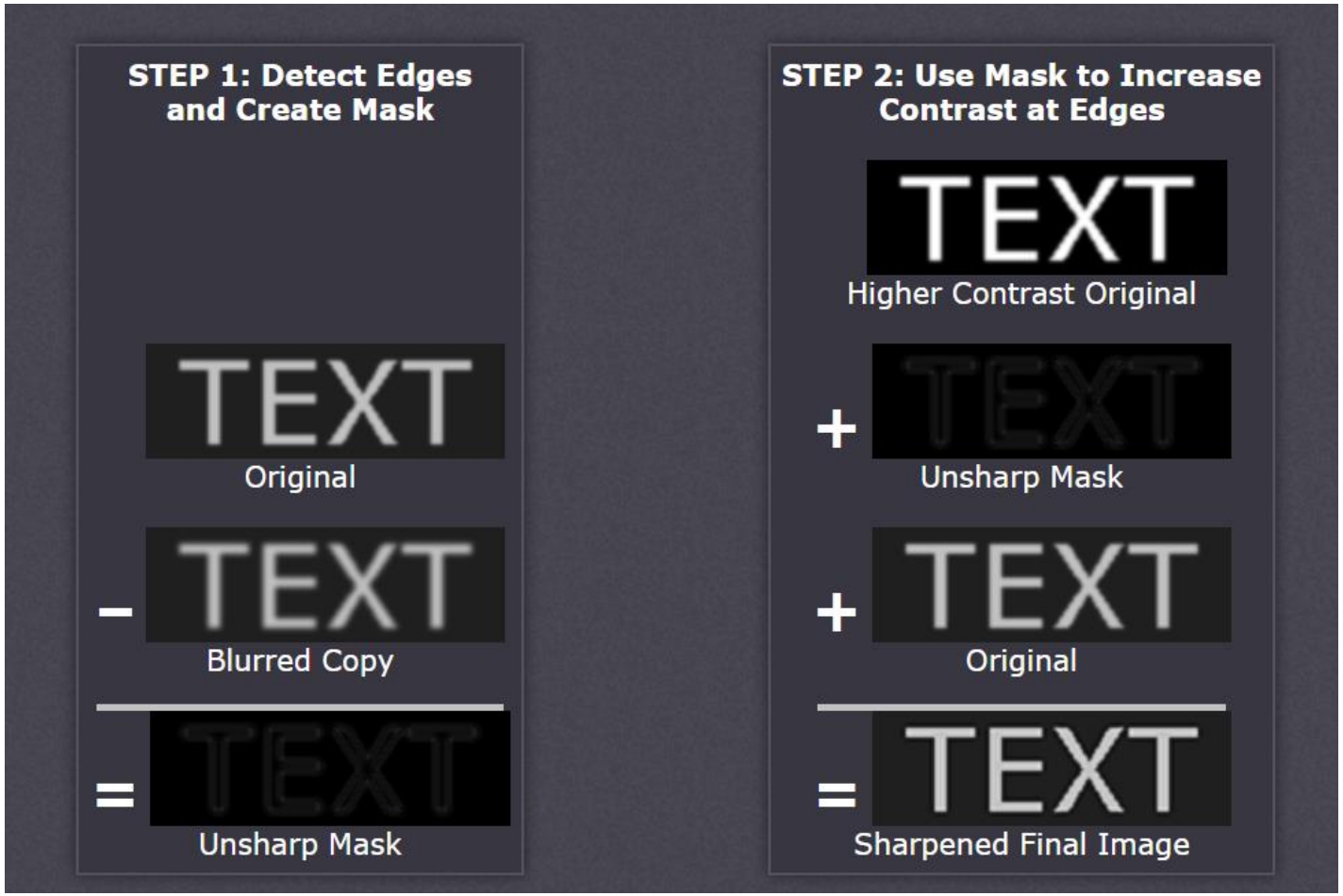
Sharpening (Edge Enhencement)

- To enhance line structures or other details in an image
- A high-pass filter can be used to make an image appear sharper.
- These filters emphasize fine details in the image
- While low-pass filtering smooths out noise, high-pass filtering does just the opposite: it *amplifies noise*.
- You can get away with this if the original image is not too noisy; otherwise the noise will overwhelm the image.

$$\begin{bmatrix} -1/9 & -1/9 & -1/9 \\ -1/9 & 1 & -1/9 \\ -1/9 & -1/9 & -1/9 \end{bmatrix}$$

https://pdfs.semanticscholar.org/fcbf/0322f336f5d2470b623fe4115dd4efbea9b7.pdf
https://diffractionlimited.com/help/maximdl/High-Pass_Filtering.htm

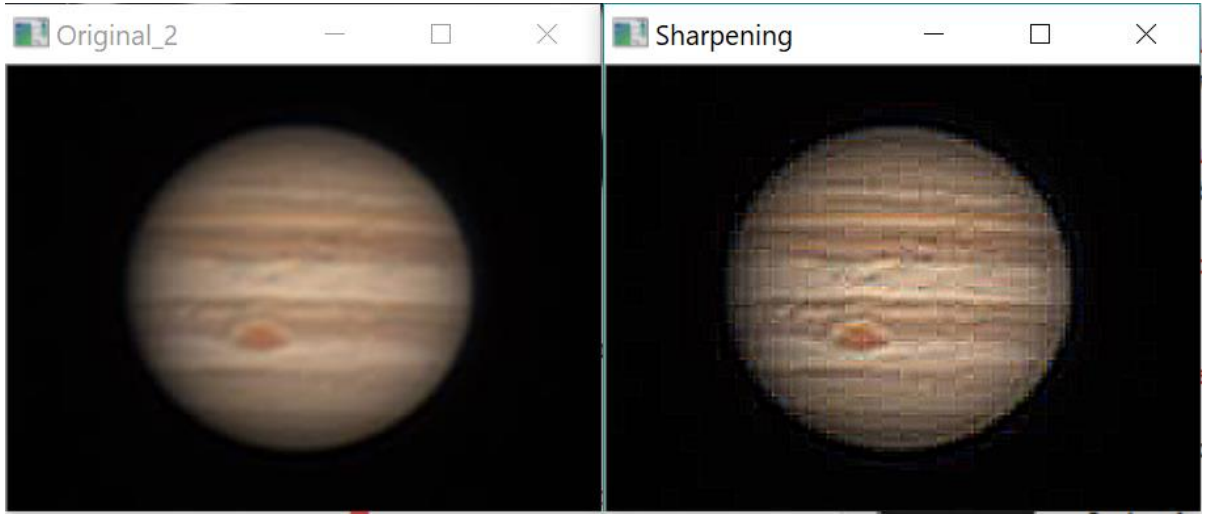
Sharpening - Unsharp Mask

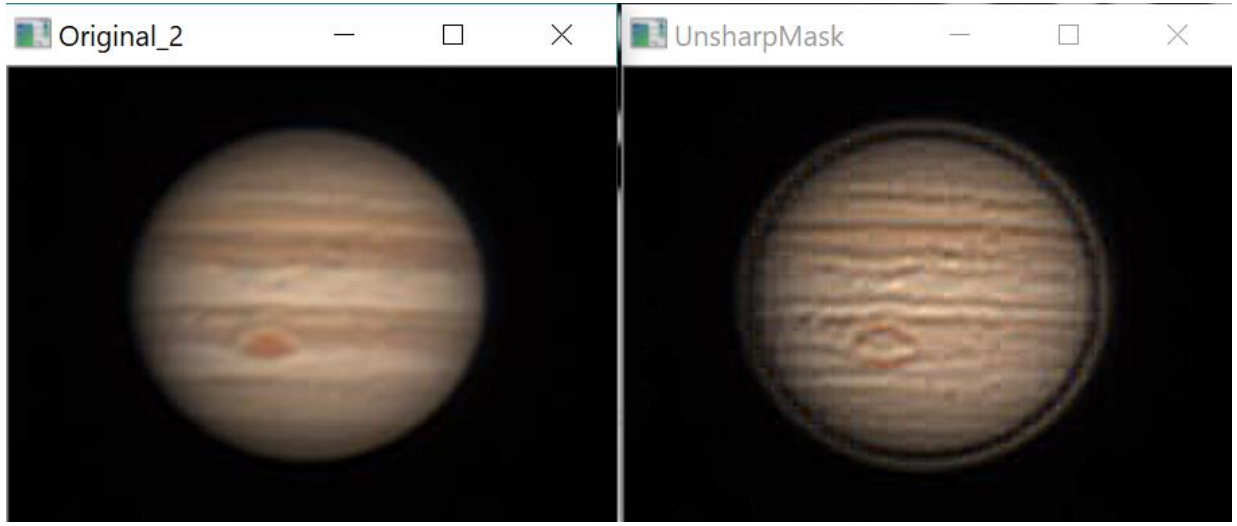


Portland State UNIVERSITY

Sharpening

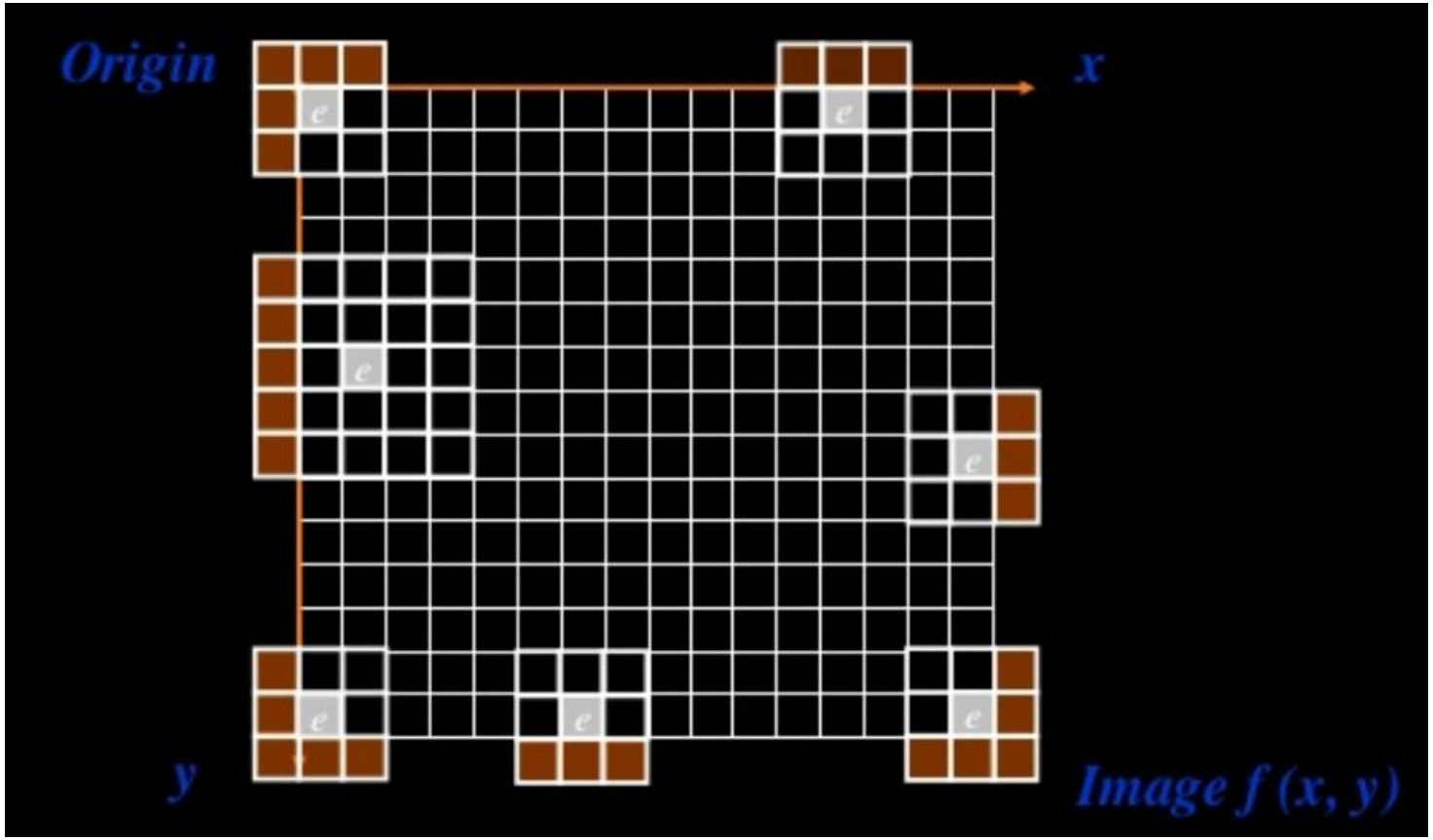
```
# Use Sharpening
kernel = np.array([[-1,-1,-1], [-1,9,-1], [-1,-1,-1]])
sharp = cv2.filter2D(img2, -1, kernel)

# Unsharp Masking
mask = cv2.GaussianBlur(img2,(0, 0), 3);
unsharp = cv2.addWeighted(mask, 4, img2, -3, 0);
```





Portland State UNIVERSITY

Border Extrapolation and Boundary Conditions

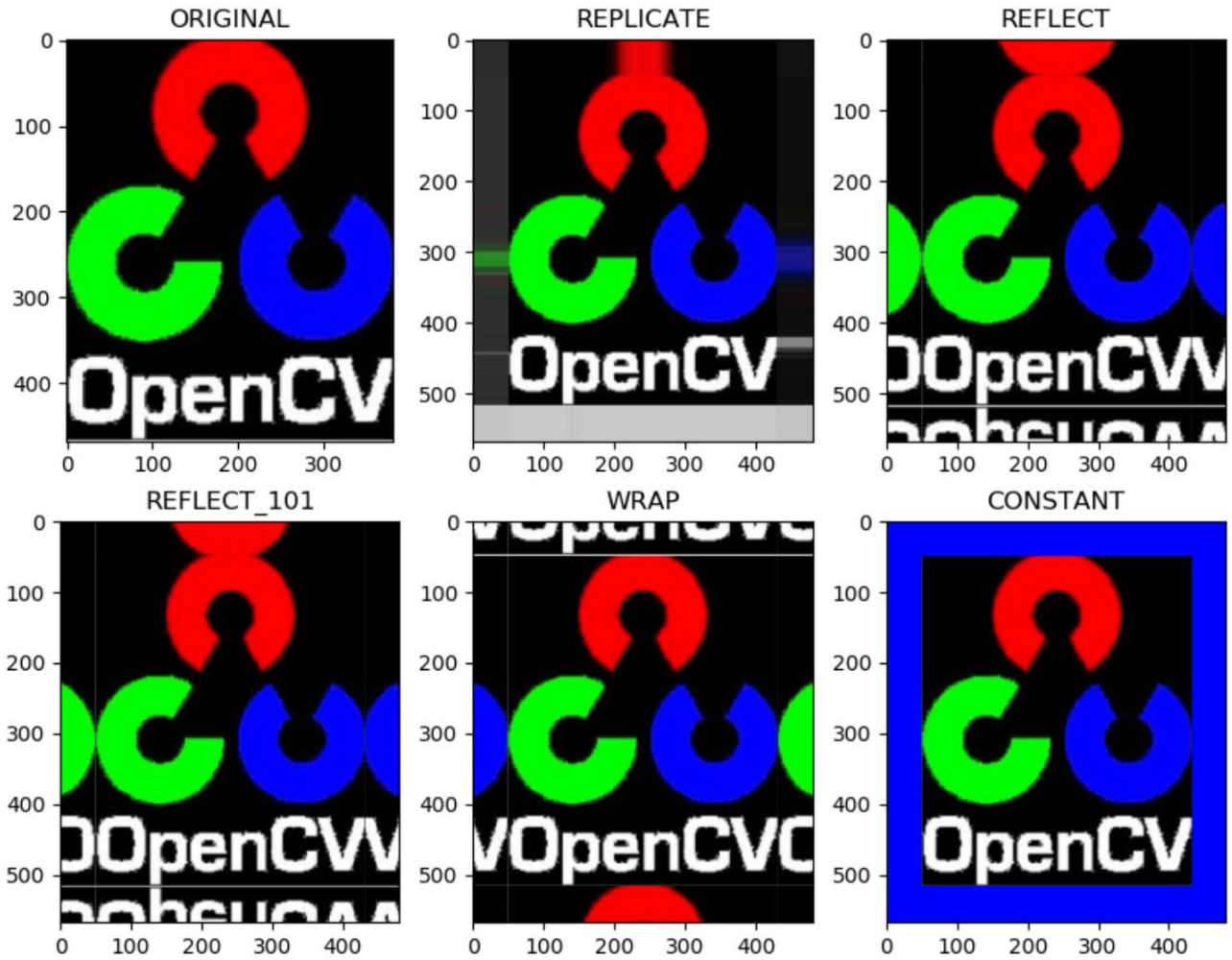


https://www.slideshare.net/anujarora3304/spatial-filtering-using-image-processing-32890033

Border Extrapolation and Boundary Conditions

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

red = [0,0,255]

img1 = cv2.imread('opencv.png')

#create boarders
replicate = cv2.copyMakeBorder(img1,50,50,50,50,cv2.BORDER_REPLICATE)
reflect = cv2.copyMakeBorder(img1,50,50,50,50,cv2.BORDER_REFLECT)
reflect101 = cv2.copyMakeBorder(img1,50,50,50,50,cv2.BORDER_REFLECT_101)
wrap = cv2.copyMakeBorder(img1,50,50,50,50,cv2.BORDER_WRAP)
constant= cv2.copyMakeBorder(img1,50,50,50,50,cv2.BORDER_CONSTANT,value=red)

# print results
# subplot (rows - columns - plot number)
plt.subplot(231),plt.imshow(img1,'gray'),plt.title('ORIGINAL')
plt.subplot(232),plt.imshow(replicate,'gray'),plt.title('REPLICATE')
plt.subplot(233),plt.imshow(reflect,'gray'),plt.title('REFLECT')
plt.subplot(234),plt.imshow(reflect101,'gray'),plt.title('REFLECT_101')
plt.subplot(235),plt.imshow(wrap,'gray'),plt.title('WRAP')
plt.subplot(236),plt.imshow(constant,'gray'),plt.title('CONSTANT')

plt.show()
```

Border Extrapolation and Boundary Conditions



Thresholding

- The simplest segmentation method

- Application example: Separate out regions of an image corresponding to objects which we want to analyze. This separation is based on the variation of intensity between the object pixels and the background pixels.

- **To differentiate the pixels we are interested in from the rest** (which will eventually be rejected), we perform a comparison of each pixel intensity value with respect to a *threshold* (determined according to the problem to solve).

- Once we have separated properly the important pixels, we can set them with a determined value to identify them (i.e. we can assign them a value of (black), white) or any value that suits your needs).

https://docs.opencv.org/2.4/doc/tutorials/imgproc/threshold/threshold.html

Thresholding

- cv2.THRESH_BINARY
- cv2.THRESH_BINARY_INV
- cv2.THRESH_TRUNC
- cv2.THRESH_TOZERO
- cv2.THRESH_TOZERO_INV

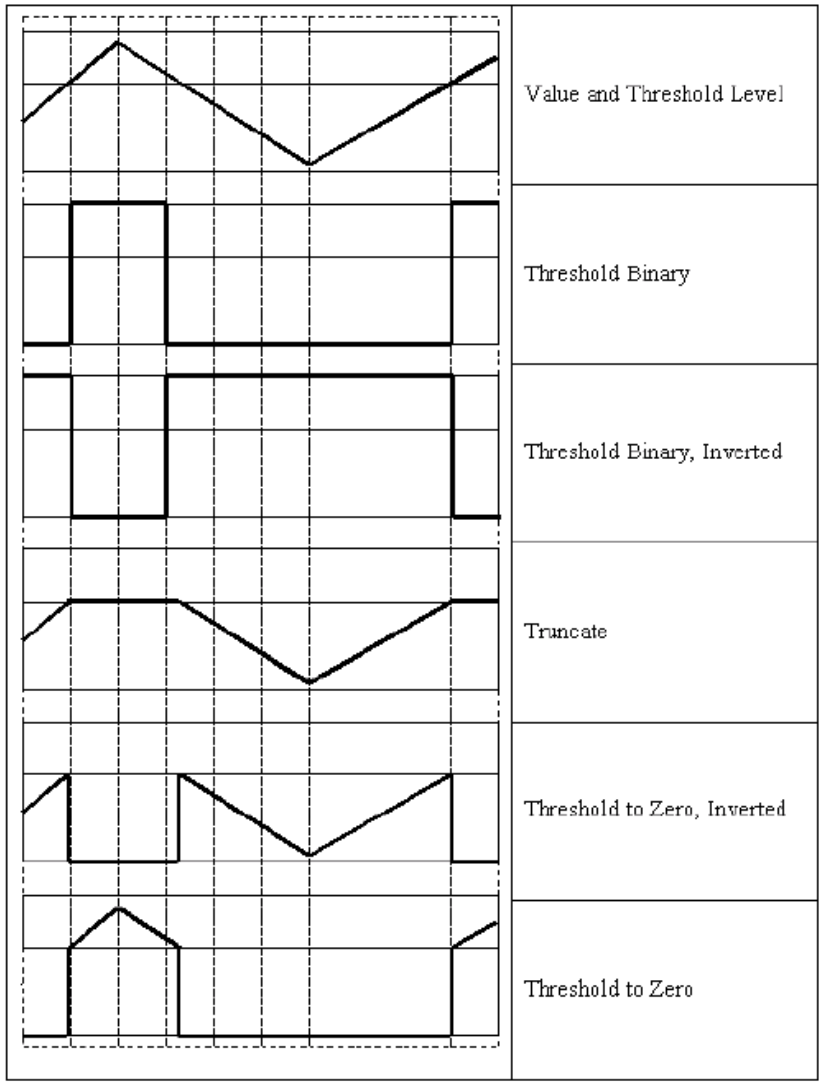


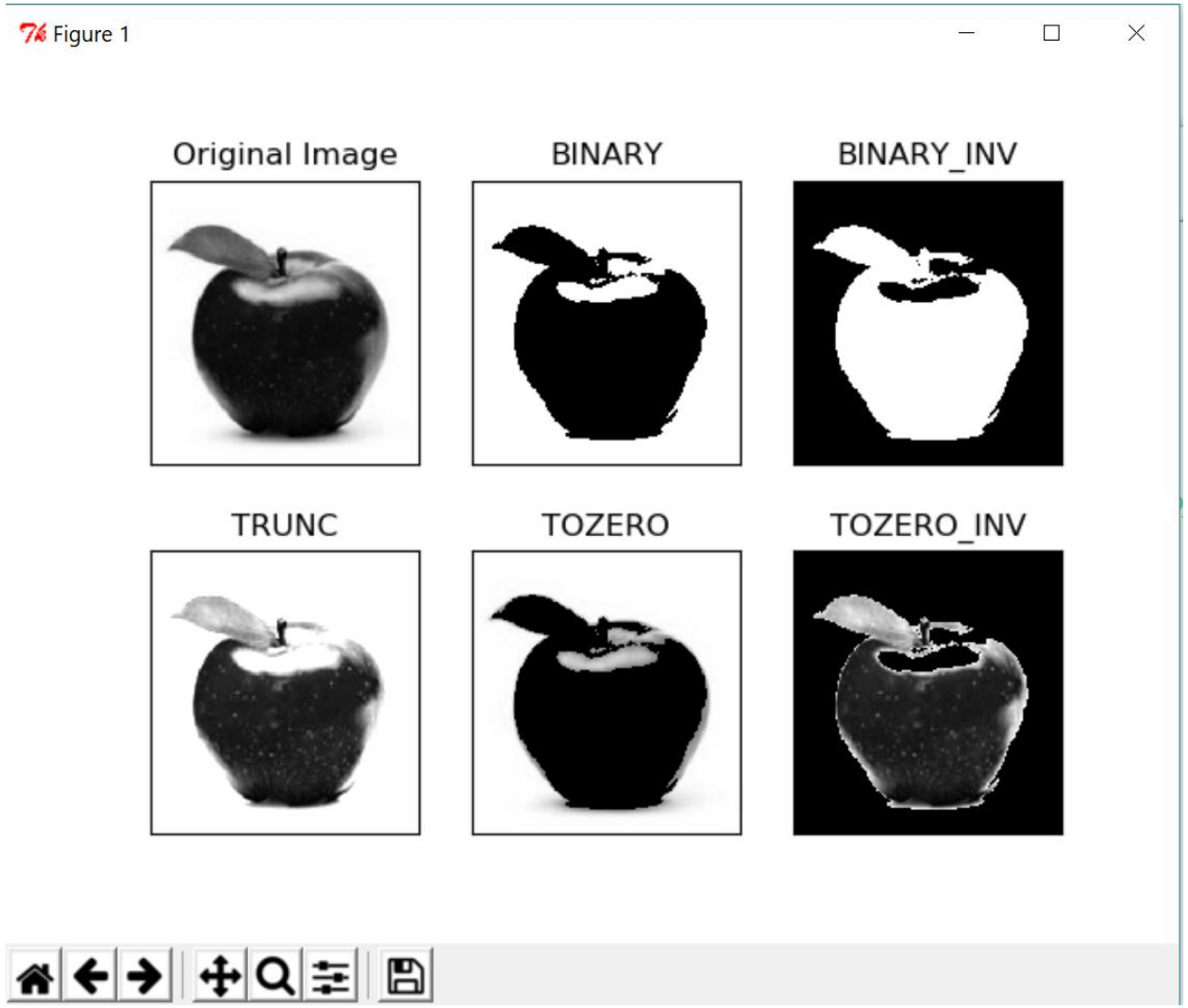
Image Source

Portland State UNIVERSITY

Thresholding

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

# Load input image
img = cv2.imread('input.png',0)

# Use different tresholds
ret,thresh1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
ret,thresh2 = cv2.threshold(img,127,255,cv2.THRESH_BINARY_INV)
ret,thresh3 = cv2.threshold(img,127,255,cv2.THRESH_TRUNC)
ret,thresh4 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO)
ret,thresh5 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO_INV)

# show the results
titles = ['Original Image','BINARY','BINARY_INV','TRUNC','TOZERO','TOZERO_INV']
images = [img, thresh1, thresh2, thresh3, thresh4, thresh5]
for i in xrange(6):
    plt.subplot(2,3,i+1),plt.imshow(images[i],'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

Thresholding



Sobel Filter

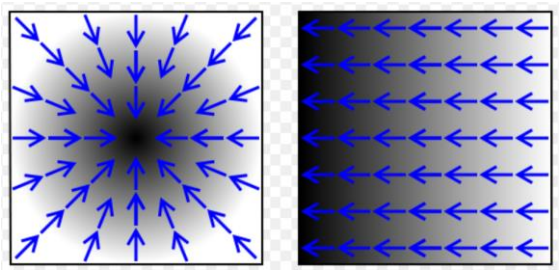
- One of the most important convolutions is the computation of derivatives in an image.
- In an edge, the pixel intensity changes (gradient)
- An image gradient is a directional change in the intensity or color in an image.
- A good way to express changes is by using derivatives. A high change in gradient indicates a major change in the image.

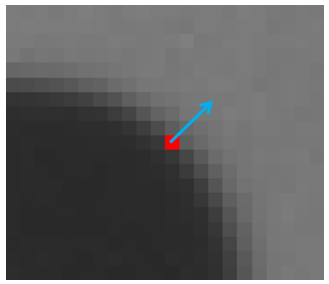
Horizontal Changes:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * I$$

Vertical Changes:

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * I$$





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

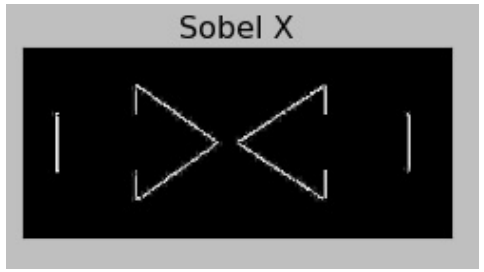
$$G = |G_x| + |G_y|$$

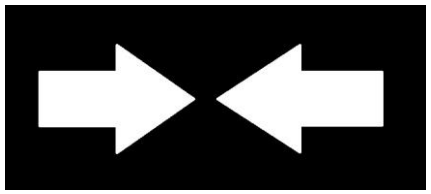
Portland State UNIVERSITY

Sobel Filter

Horizontal Changes:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * I$$



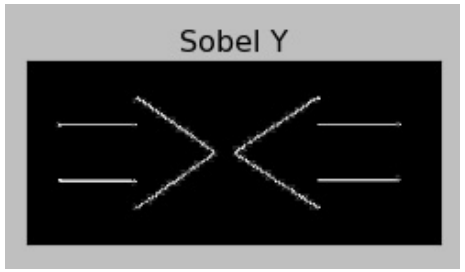


$$G = \sqrt{G_x^2 + G_y^2}$$

$$G = |G_x| + |G_y|$$

Vertical Changes:

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * I$$



$$Angle\ (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

Direction of the edge

OpenCV - Applications

Portland State
UNIVERSITY

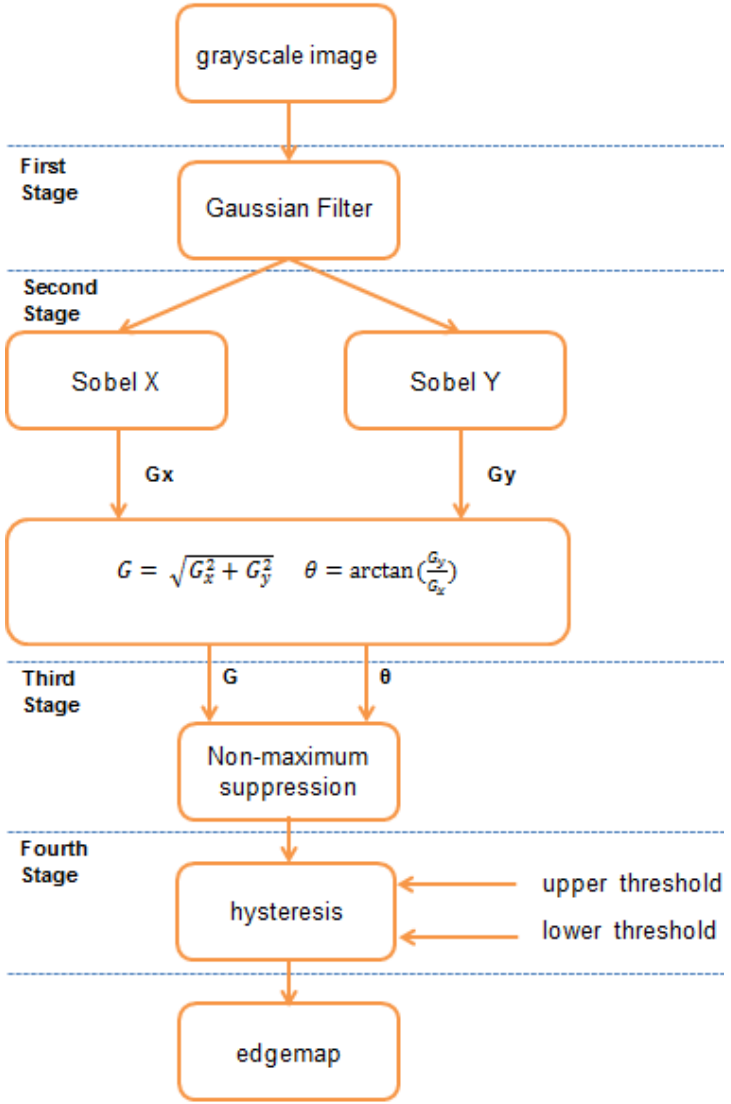
Canny Edge Detection

Portland State
UNIVERSITY

Canny Edge Detection

- Canny Edge Detection is a popular edge detection algorithm.
  - It is a multi-stage algorithm
  - It is a kernel convolution progress
  - It return high response in where there is a sharp change in gradient image
    - First we blur the image to make it smoother
    - Then we use Sobel filter in x and y direction
    - Then we do non maximum suppression
    - We threshold the result to get rid of unreal edges

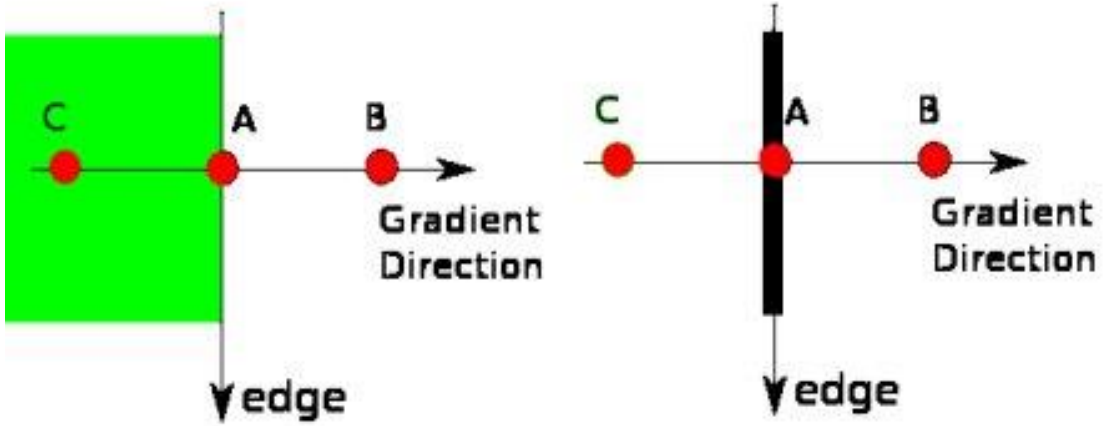
Canny Edge Detection

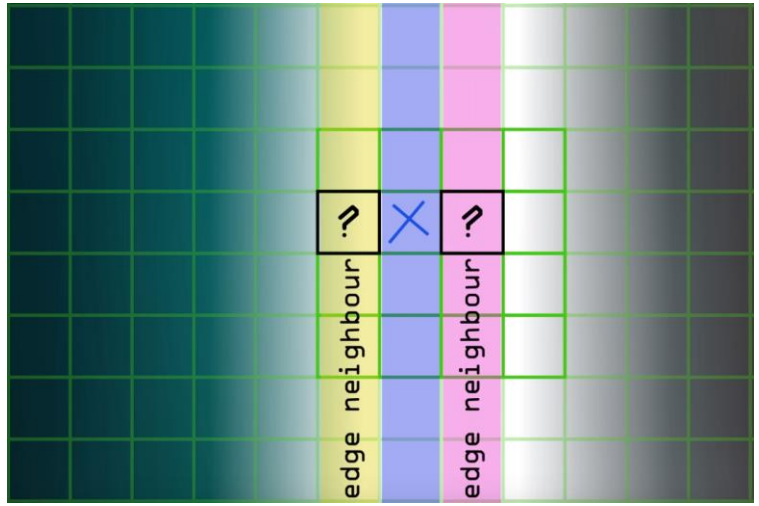


Portland State UNIVERSITY

Canny Edge Detection

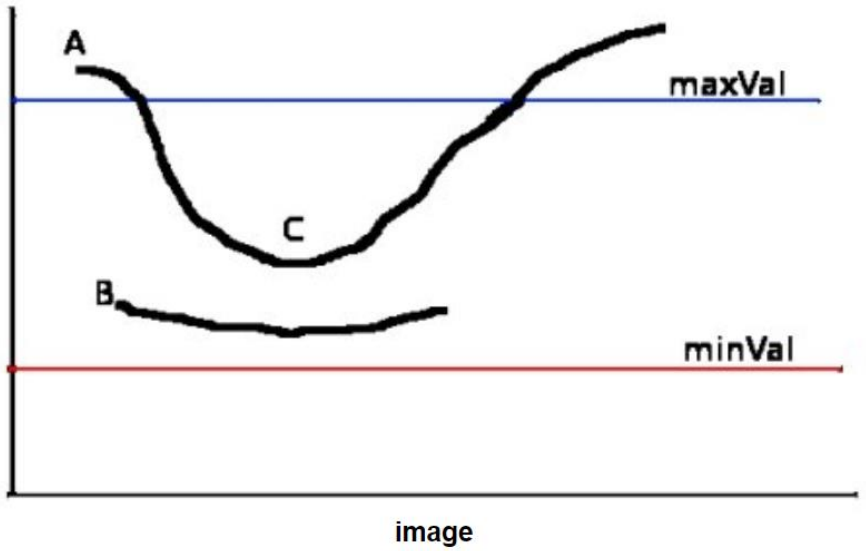
- Try to find the orientation of edge
- At every pixel, pixel is checked if it is a local maximum in its neighborhood in the direction of gradient.





Canny Edge Detection

- Intensity gradient threshold value is used
- A is an edge
- B is not an edge

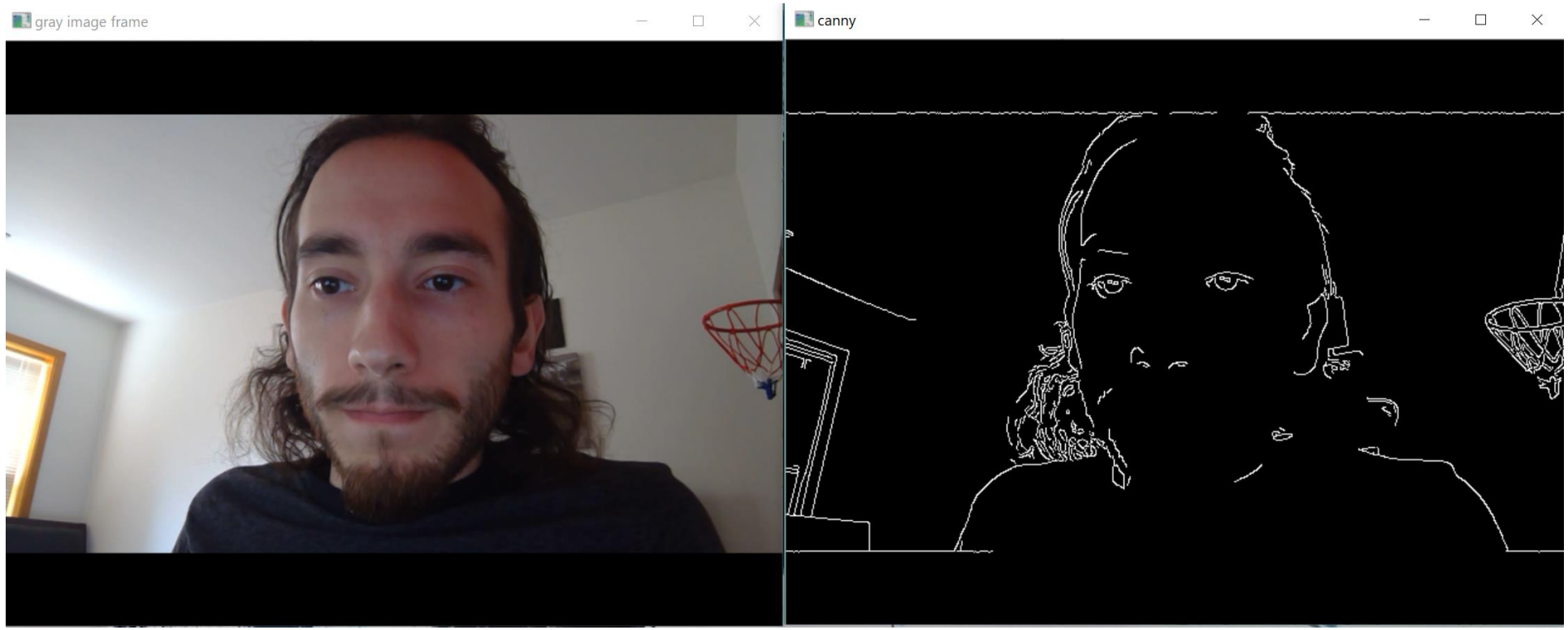


Canny Edge Detection

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

# use camera as a source
# if you have more than 1 camera change the number 0, 1, 2 etc.
cap = cv2.VideoCapture(0)

while(True):
    # Capture frame-by-frame
    ret, frame = cap.read()

    # Our operations on the frame come here
    # Make the frame grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    #perform canny edge detection
    edges = cv2.Canny(gray,100,200)

    #Display the original frame
    cv2.imshow('gray image frame',frame)

    #Display the result of Canny
    cv2.imshow('canny',edges)

    plt.show()
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# When everything done, release the capture
cap.release()
cv2.destroyAllWindows()
```

Portland State UNIVERSITY

Canny Edge Detection



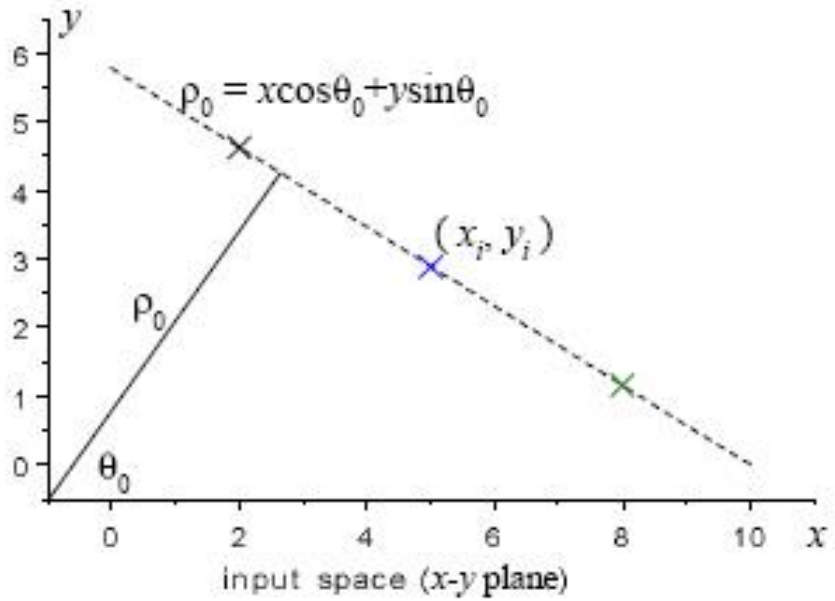
Portland State
UNIVERSITY

Hough Line Transform

Portland State
UNIVERSITY

Hough Line Transform

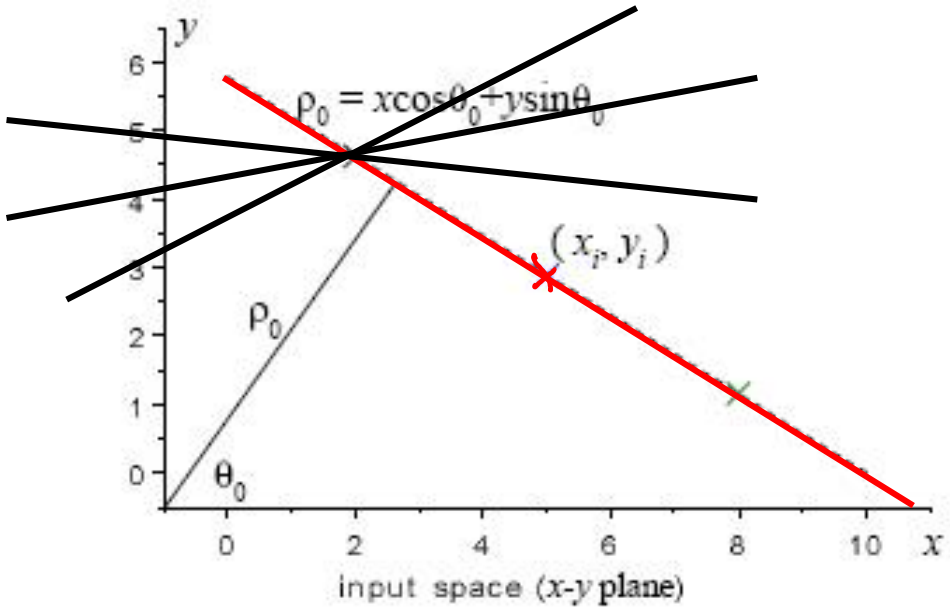
Hough Line Transform

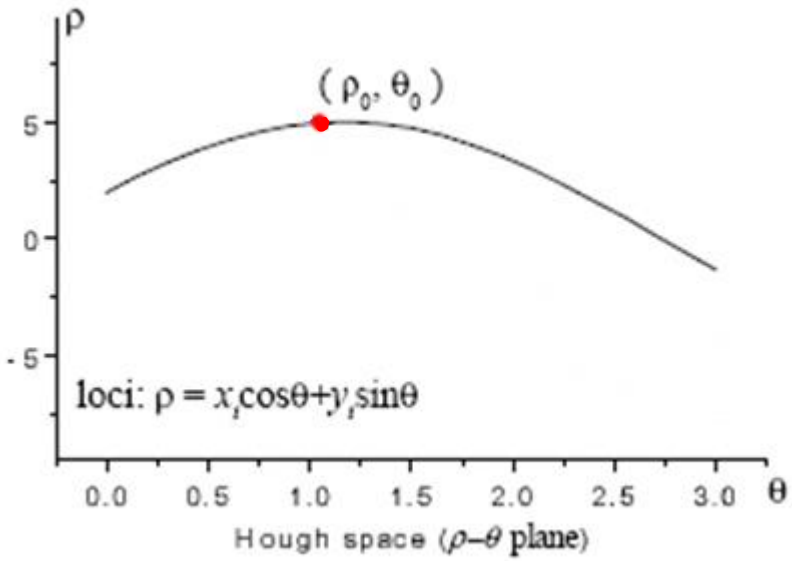


$$\rho_0 = x\cos\theta_0 + y\sin\theta_0$$

Portland State UNIVERSITY

Hough Line Transform



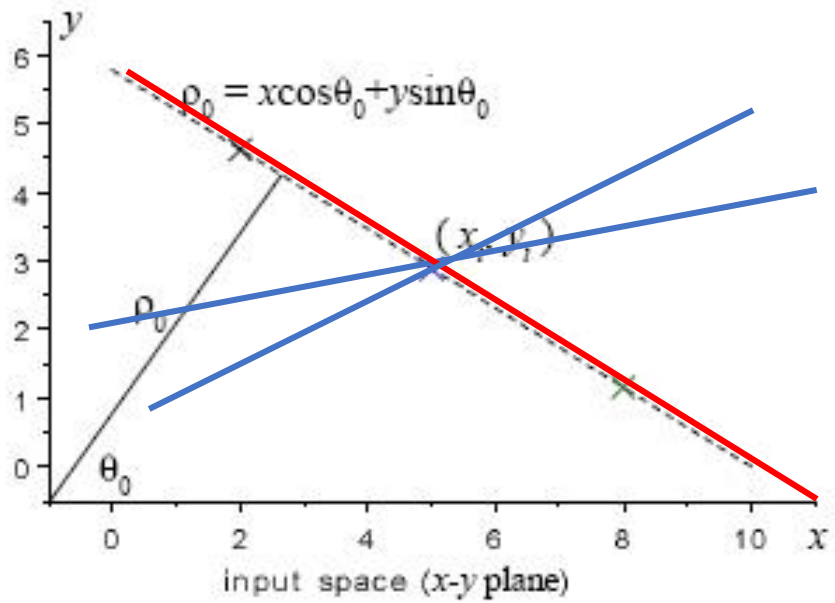


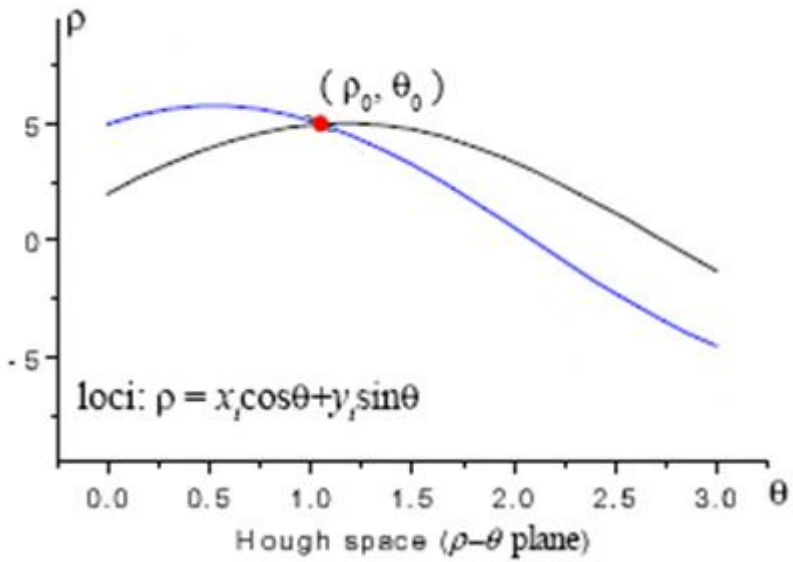
Votes for every (Ƥ, θ)

$$\rho_0 = x\cos\theta_0 + y\sin\theta_0$$

Portland State UNIVERSITY

Hough Line Transform



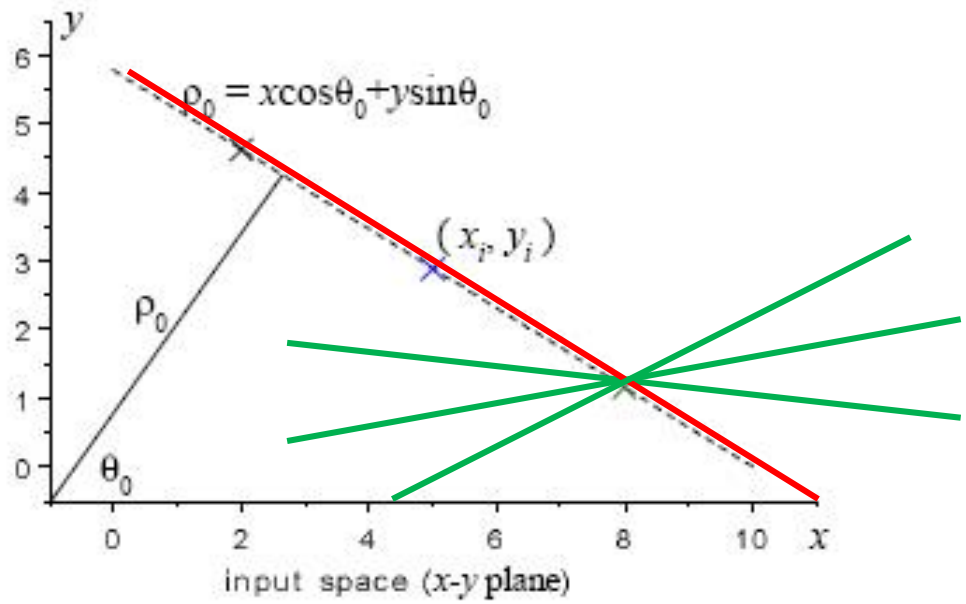


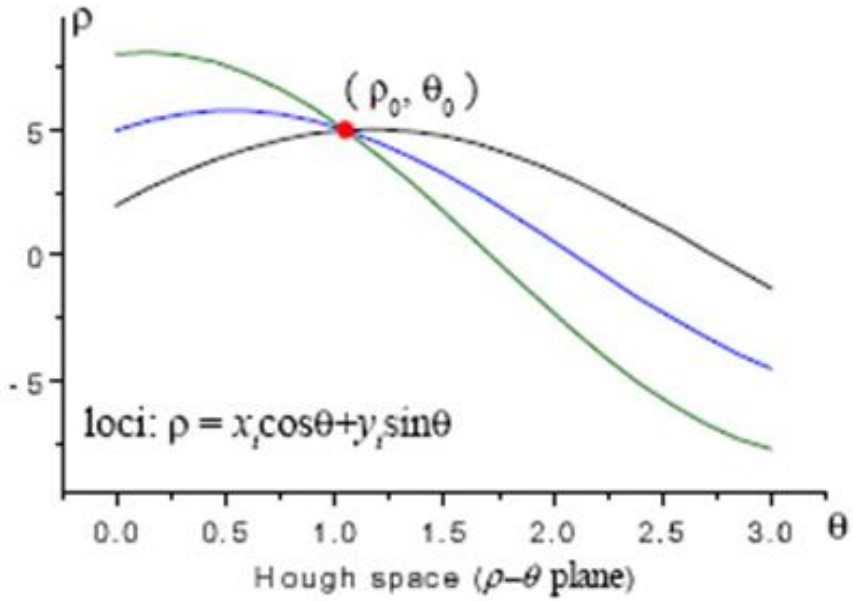
Votes for every (Ᵽ, θ)

$$\rho_0 = x\cos\theta_0 + y\sin\theta_0$$

Portland State UNIVERSITY

Hough Line Transform





Votes for every (Ƥ, θ)

$\rho_0 = x\cos\theta_0 + y\sin\theta_0$

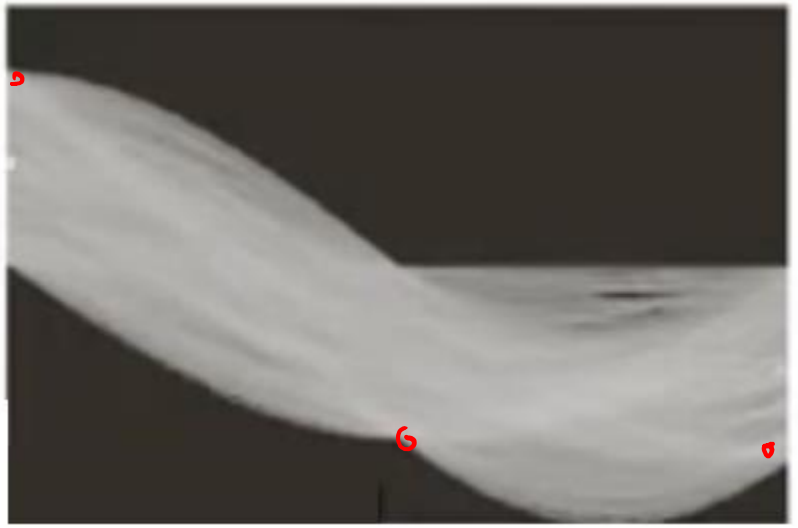
Portland State UNIVERSITY

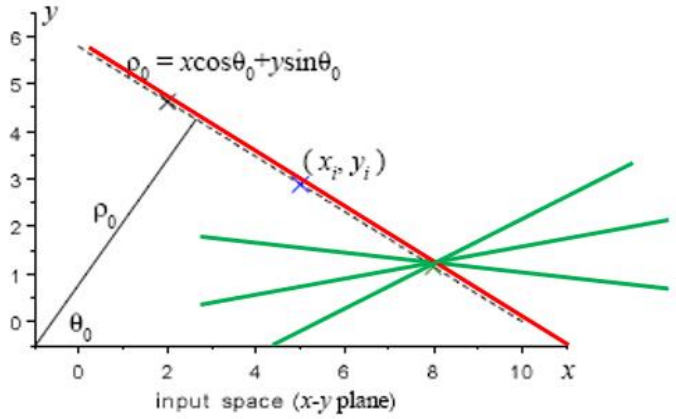
Hough Line Transform

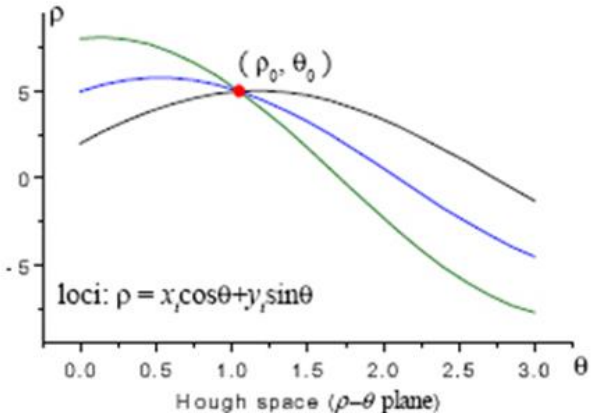
Original Image

Canny Edge Detection

Hough Line Transform







Portland State UNIVERSITY

Hough Line Transform

Hough Space

Draw a line

Hough Line Transform

Draw a line

Original Image with the Line

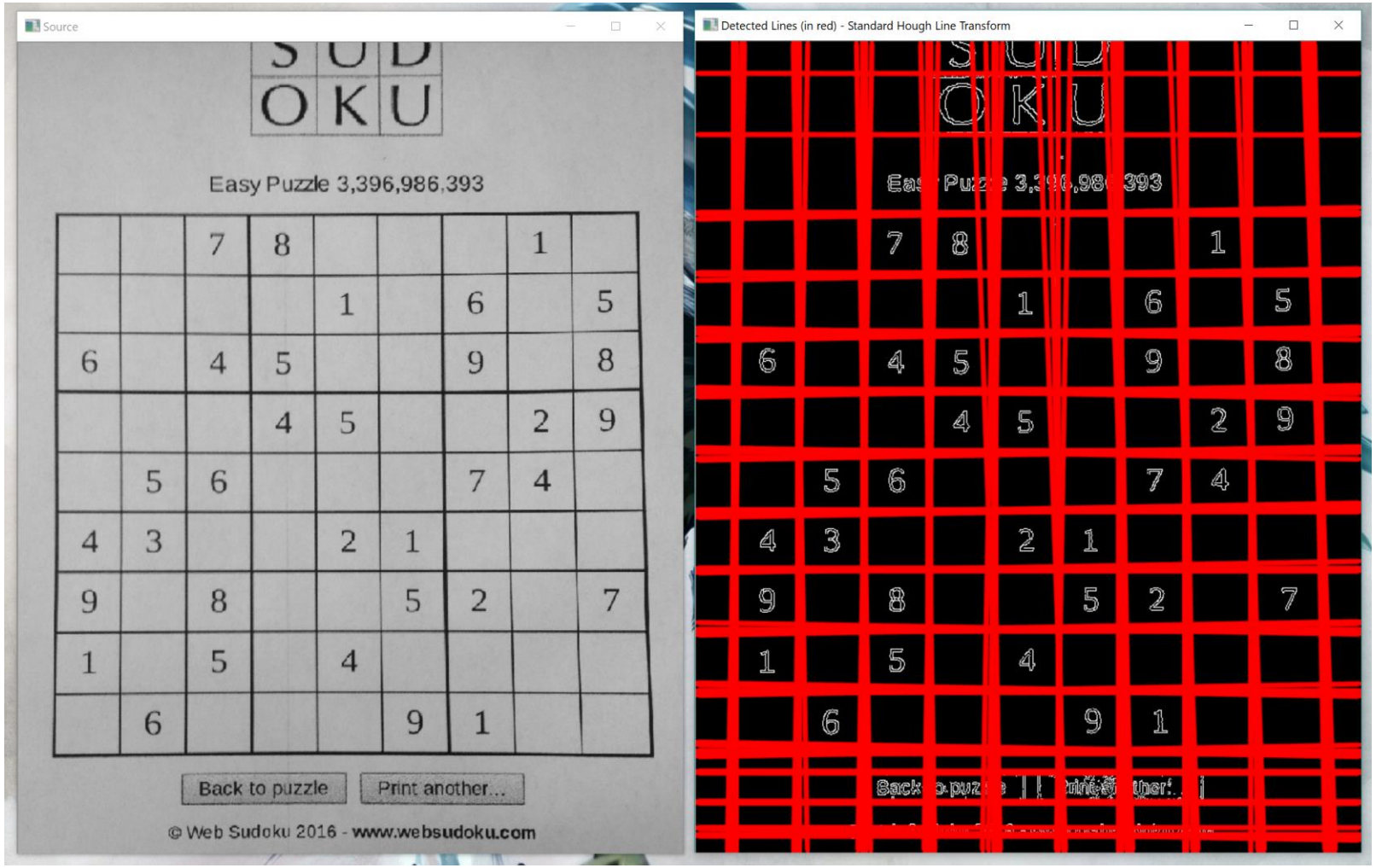
Hough Line Transform

```
import sys
import math
import cv2 as cv
import numpy as np
def main(argv):


    # Loads an image
    src = cv.imread("sudoku.png", cv.IMREAD_GRAYSCALE)

    # Use Canny Edge Detector
    dst = cv.Canny(src, 50, 200, None, 3)

    # Copy edges to the images that will display the results in BGR
    cdst = cv.cvtColor(dst, cv.COLOR_GRAY2BGR)
    cdstP = np.copy(cdst)

    # dst: grayscale image
    # lines: A vector that will store the parameters (r,θ) of the detected lines
    # rho : The resolution of the parameter r in pixels. We use 1 pixel
    # theta: The resolution of the parameter θ in radians. We use 1 degree (CV_PI/180)
    # threshold: The minimum number of intersections to "*detect*" a line
    # srn and stn: Default parameters to zero (divisor for the distance resolutions rho and theta )
    lines = cv.HoughLines(dst, 1, np.pi / 180, 150, None, 0, 0)

    # draw lines
    if lines is not None:
        for i in range(0, len(lines)):
            rho = lines[i][0][0]
            theta = lines[i][0][1]
            a = math.cos(theta)
            b = math.sin(theta)
            x0 = a * rho
            y0 = b * rho
            pt1 = (int(x0 + 1000*(-b)), int(y0 + 1000*(a)))
            pt2 = (int(x0 - 1000*(-b)), int(y0 - 1000*(a)))
            cv.line(cdst, pt1, pt2, (0,0,255), 3, cv.LINE_AA)
```

Hough Line Transform

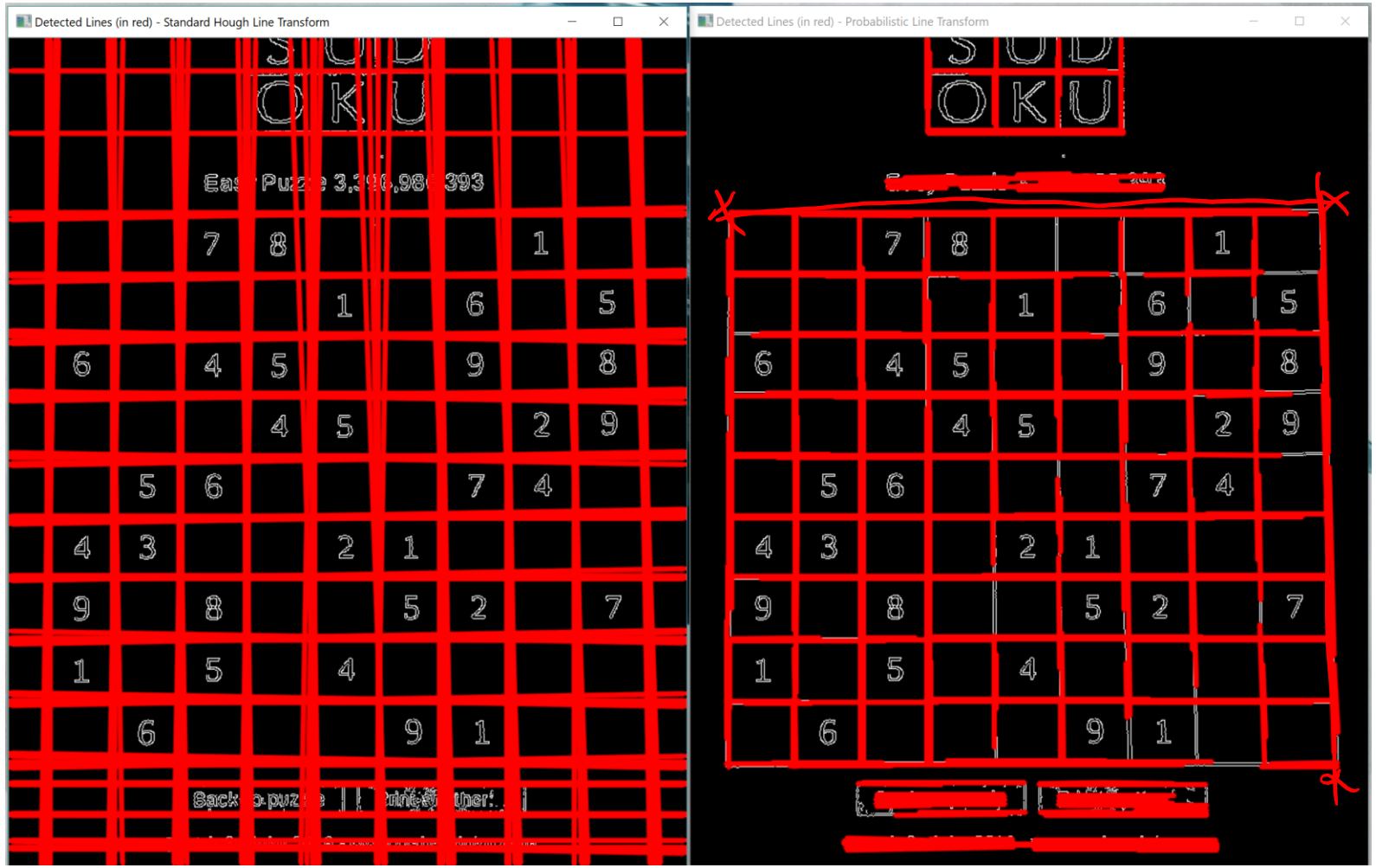
```
    #Perform Probabilistic Hough Transform
    #A more efficient implementation of the Hough Line Transform.
    #It gives as output the extremes of the detected lines
    linesP = cv.HoughLinesP(dst, 1, np.pi / 180, 50, None, 50, 10)

    #draw lines
    if linesP is not None:
        for i in range(0, len(linesP)):
            l = linesP[i][0]
            cv.line(cdstP, (l[0], l[1]), (l[2], l[3]), (0,0,255), 3, cv.LINE_AA)

    cv.imshow("Source", src)
    cv.imshow("Detected Lines (in red) - Standard Hough Line Transform", cdst)
    cv.imshow("Detected Lines (in red) - Probabilistic Line Transform", cdstP)

    cv.waitKey()
    return 0

if __name__ == "__main__":
    main(sys.argv[1:])
```

Hough Line Transform



Portland State UNIVERSITY

Hough Line Transform



Portland State UNIVERSITY

Questions?

Portland State
UNIVERSITY