AI VIETNAM
All-in-One Course

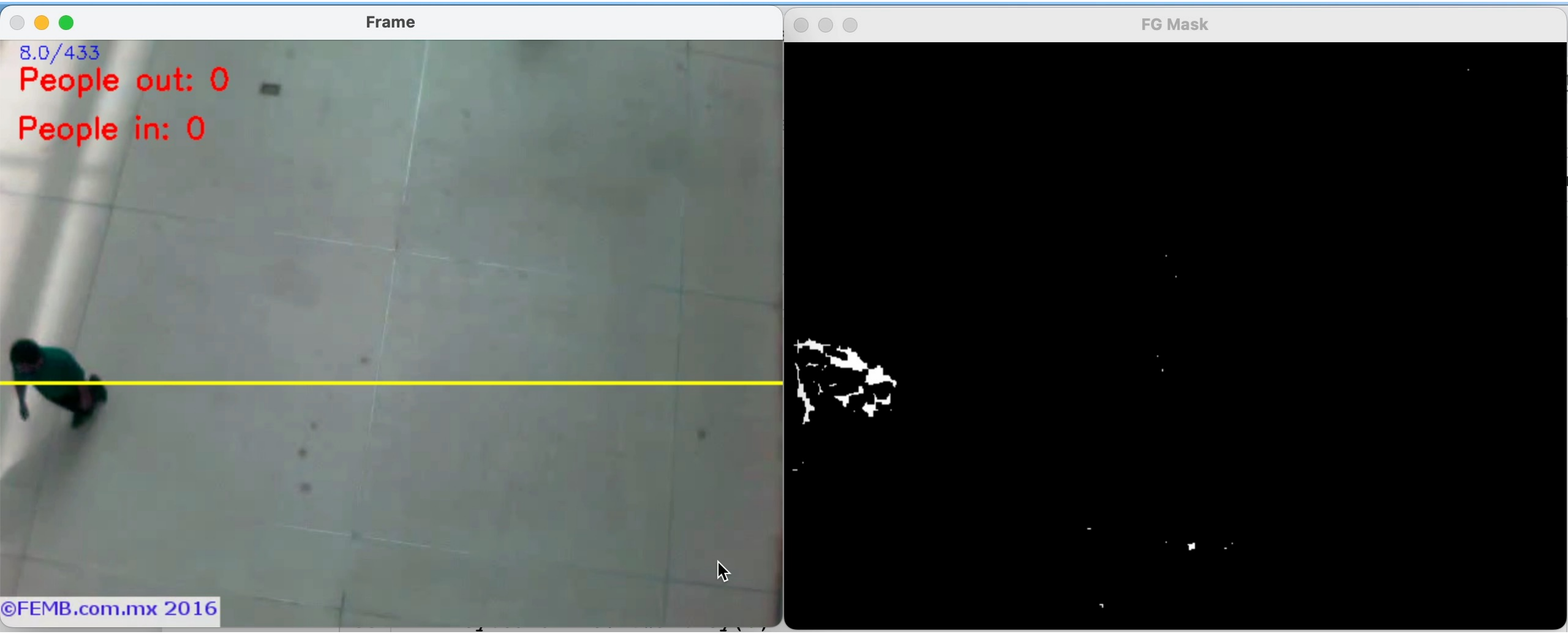
Background Subtraction

(Computer Vision Foundation)

Vinh Dinh Nguyen
PhD in Computer Science

Year 2023

People Counter at Suppermarket



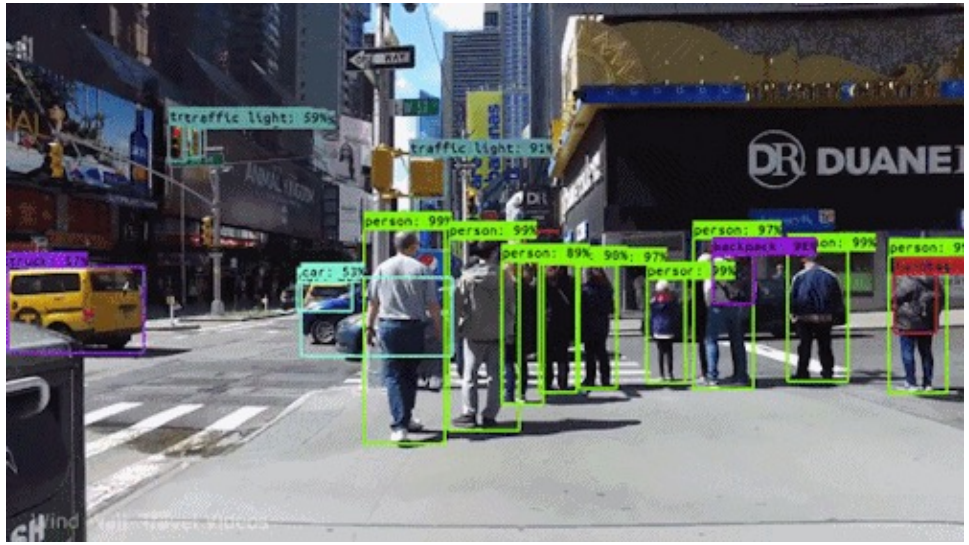
Outline

- Introduction to Computer Vision
- Edge Detection Algorithm
- Line Detection Algorithm
- Contour Detection Algorithm
- Shape Detection Algorithm
- Background Subtraction Algorithm

Outline

- Introduction to Computer Vision
- Edge Detection Algorithm
- Line Detection Algorithm
- Contour Detection Algorithm
- Shape Detection Algorithm
- Background Subtraction Algorithm

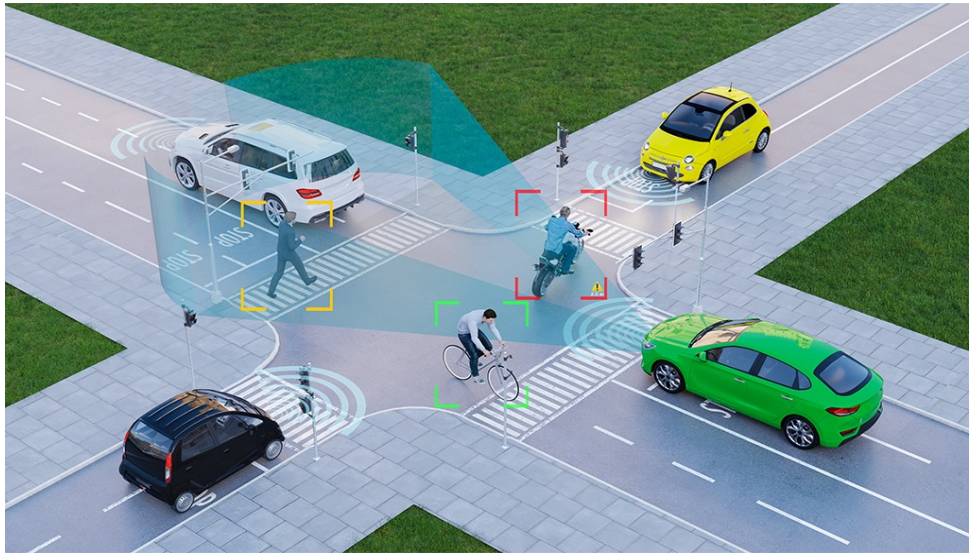
Computer Vision



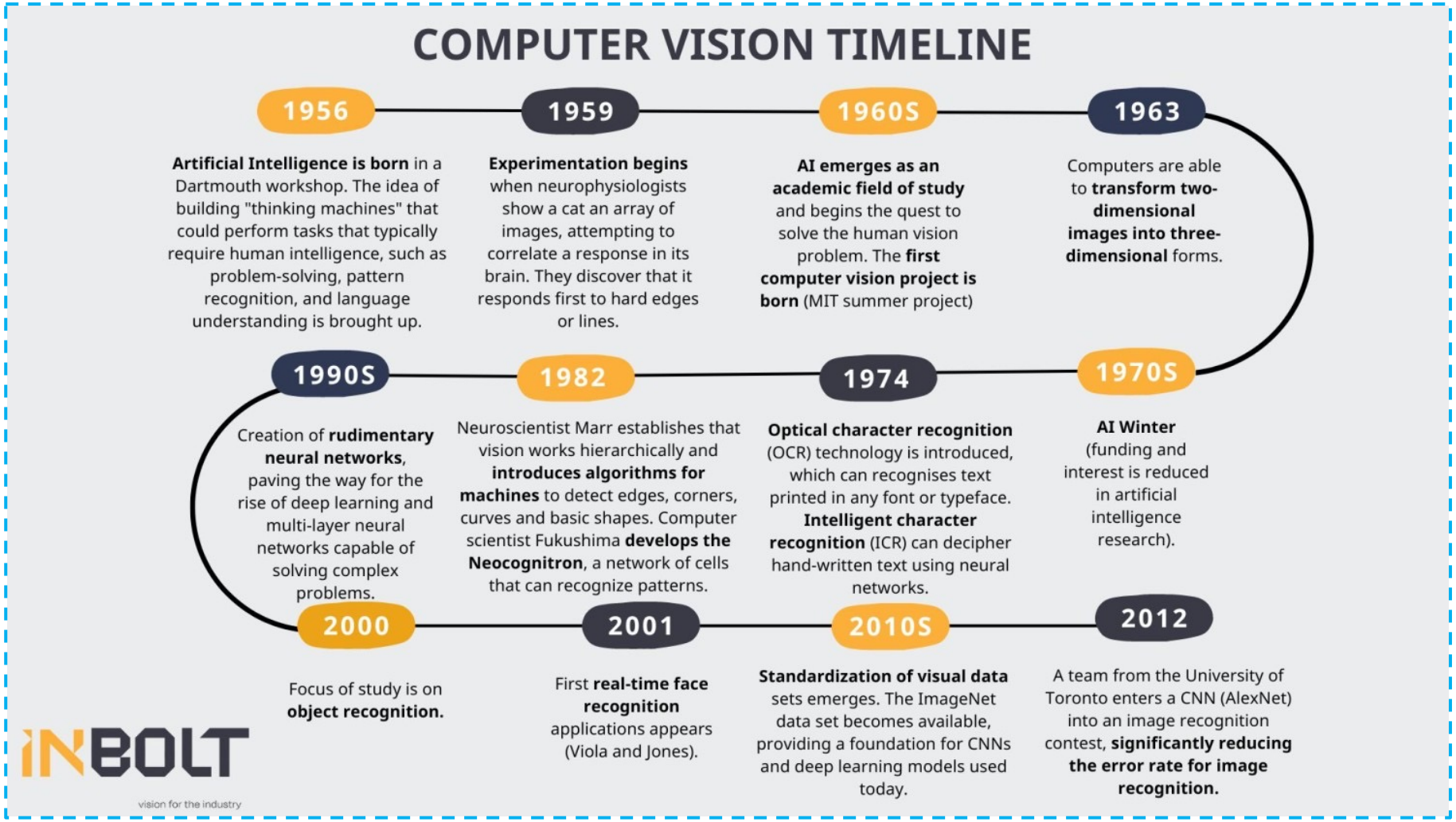
Help computers "*see*" and "*understand*" the content of digital images such as photographs and videos

Computer Vision Market to Hit $58.29 Billion by 2030: Grand View Research, Inc. (Bloomberg)

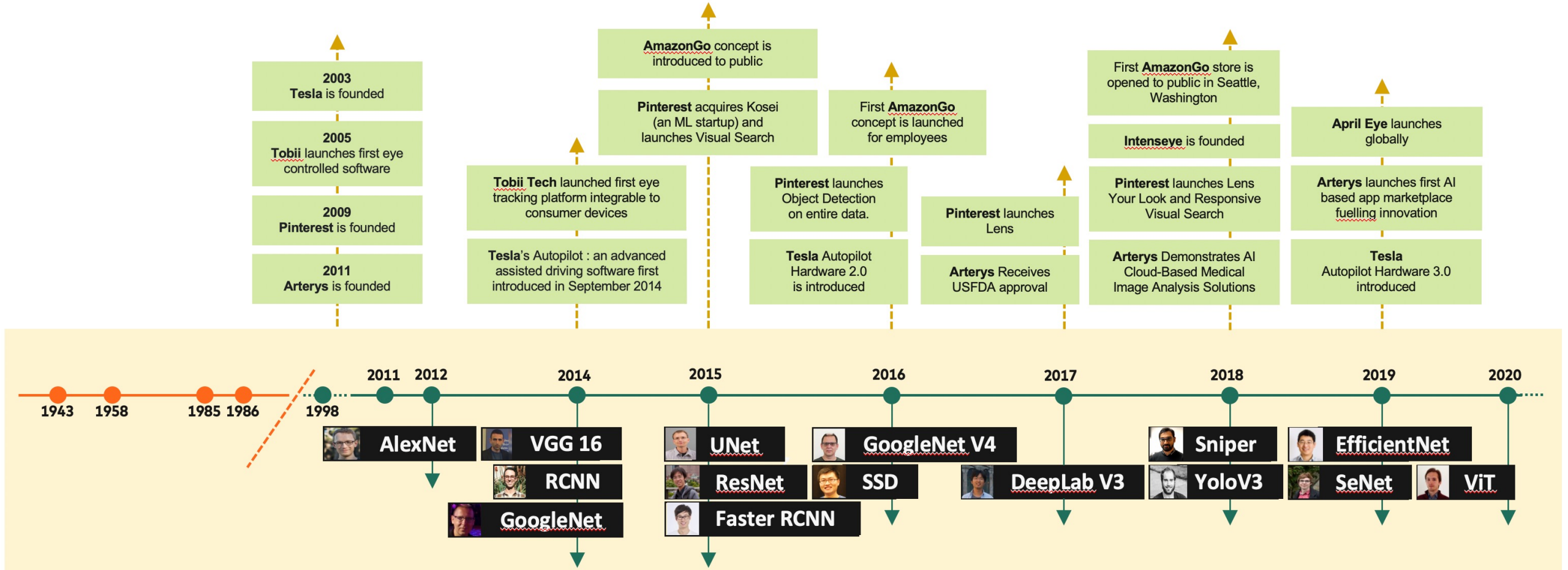
What is Computer Vision? How does computer vision work?



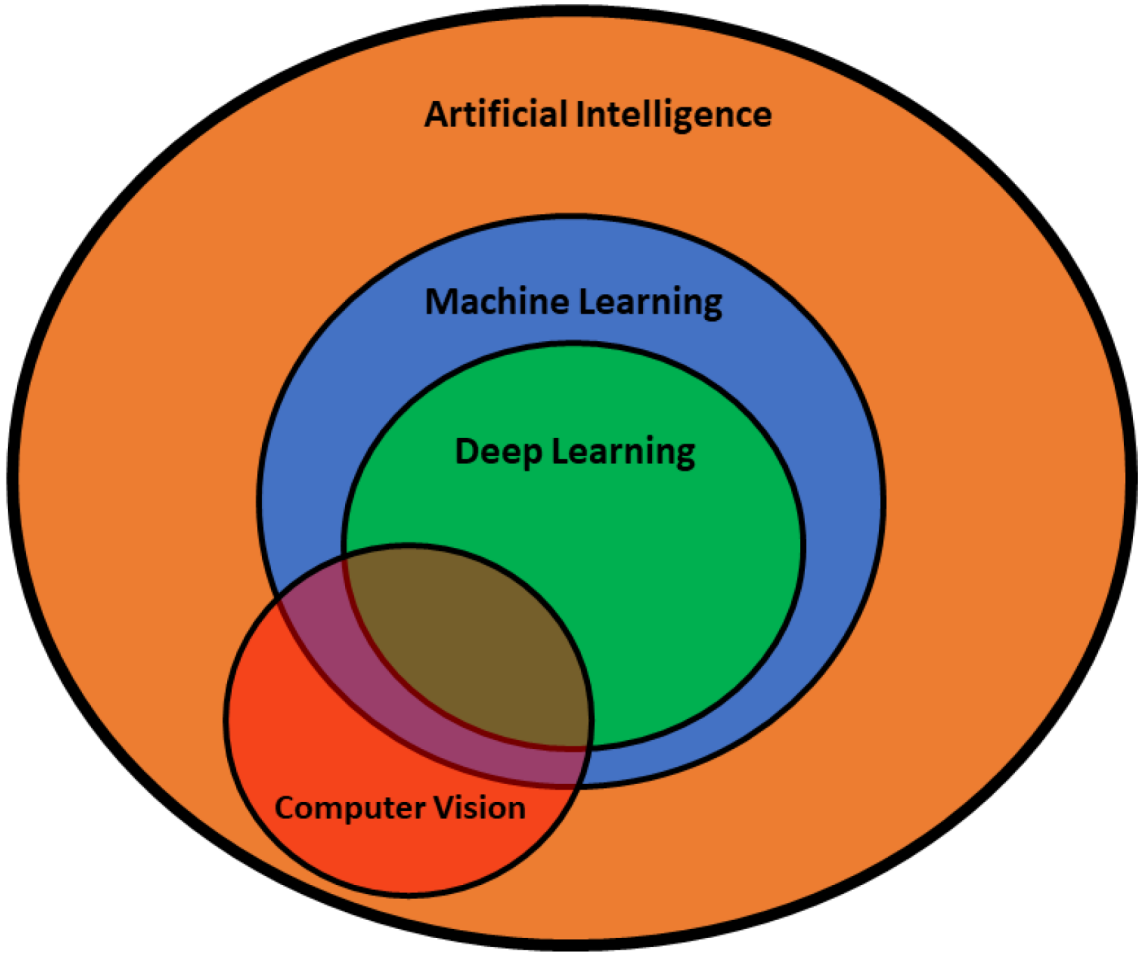
Computer Vision History

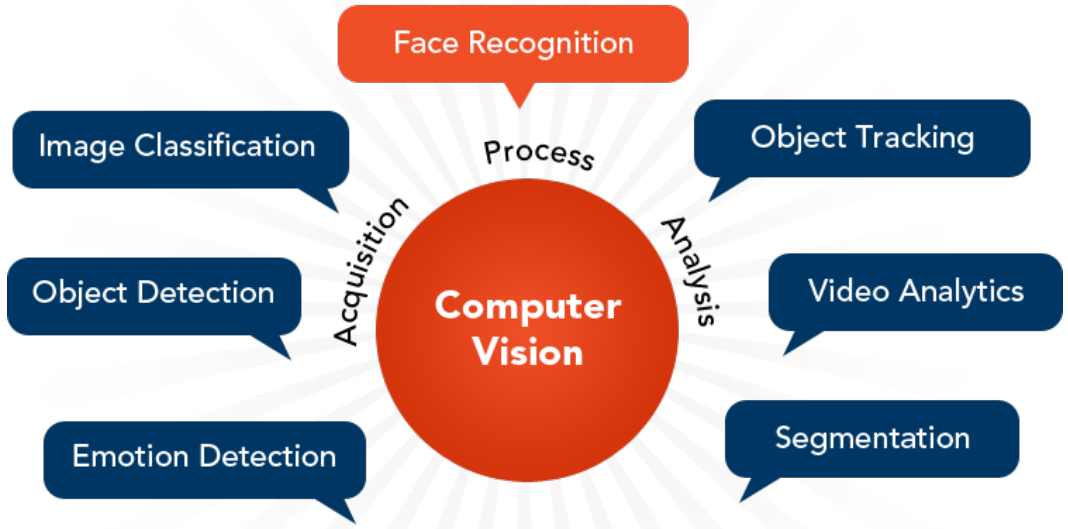


Computer Vision History

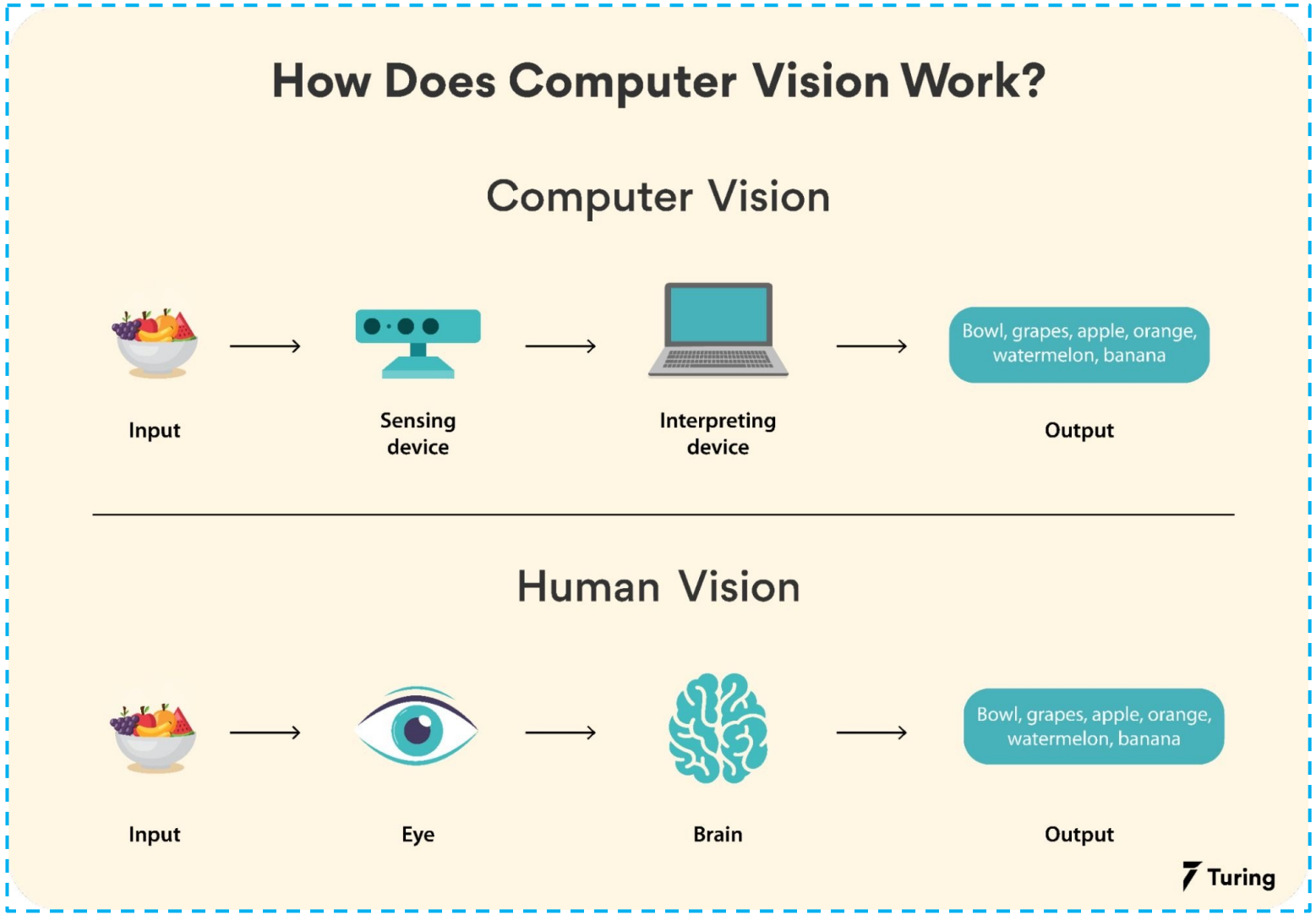


Computer Vision Applications



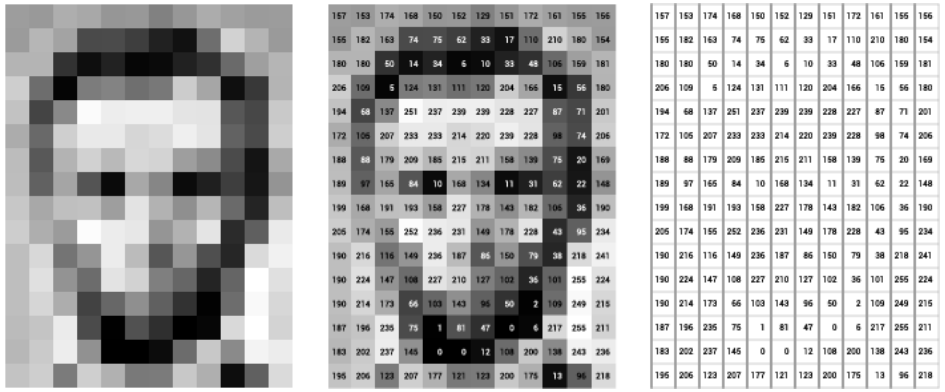


How Does Computer Vision Work

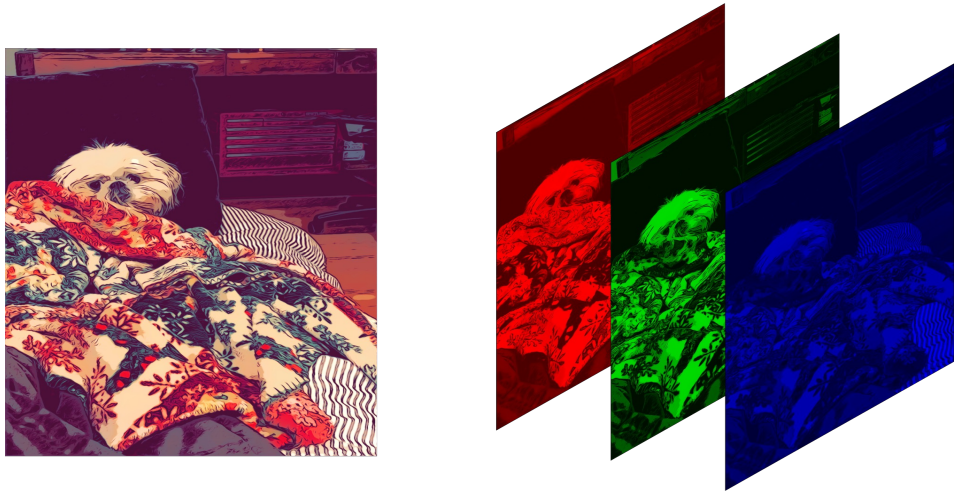


https://www.turing.com/kb/all-you-need-to-know-about-computer-vision

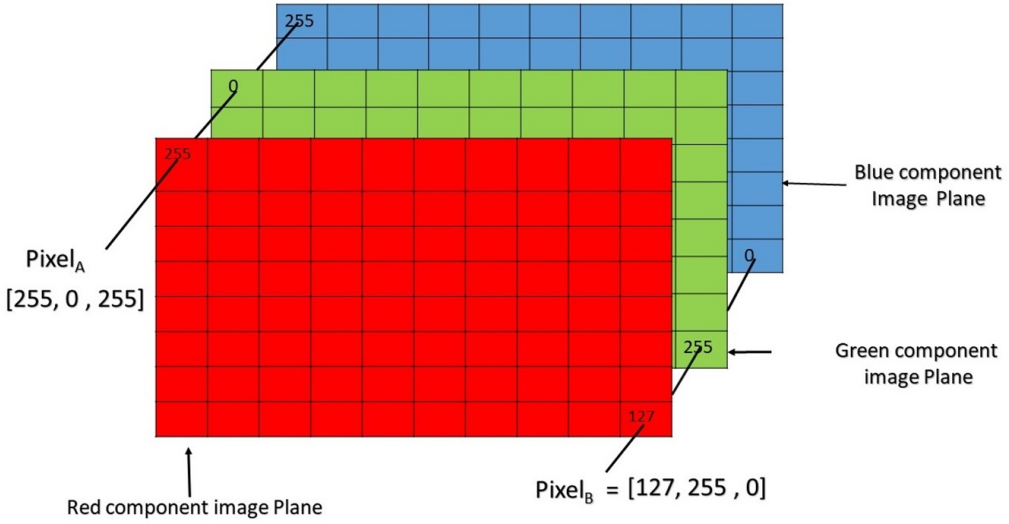
Image in Computer Vision



Grayscale image



Color image



Read and Display Image Using OpenCV

Color image

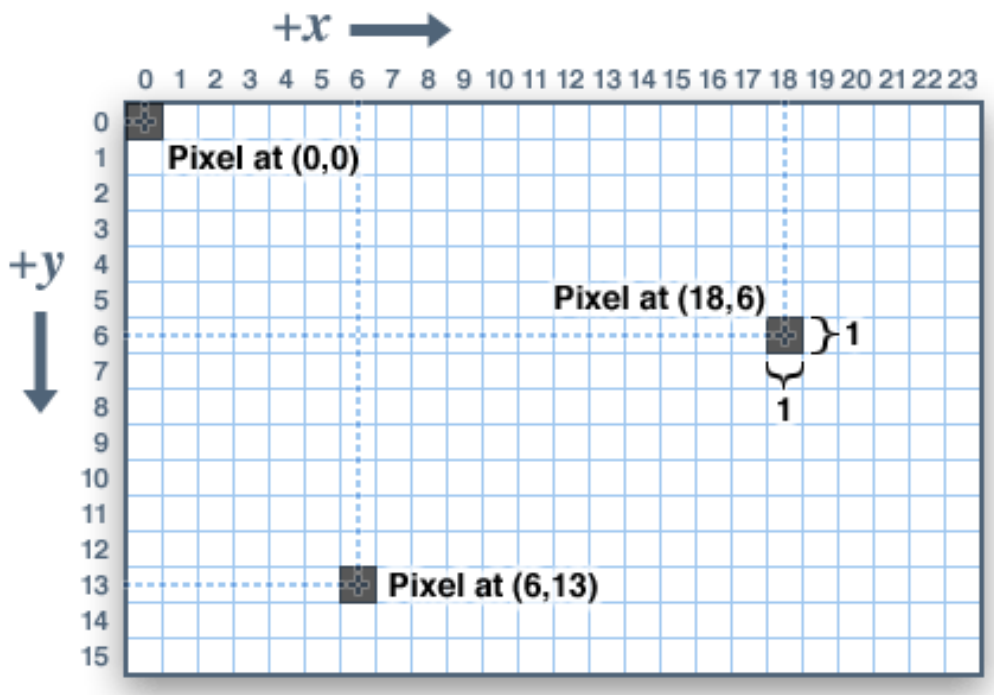
Read and display image

```
import cv2
from google.colab.patches import cv2_imshow

 # Opencv code to read and display image
color_img = cv2.imread("/content/cat_image.jpeg", cv2.IMREAD_COLOR)
cv2_imshow(color_img)


# Opencv to convert color image to grayscale image
gray_image = cv2.cvtColor(color_img, cv2.COLOR_BGR2GRAY)
cv2_imshow(gray_image)
```

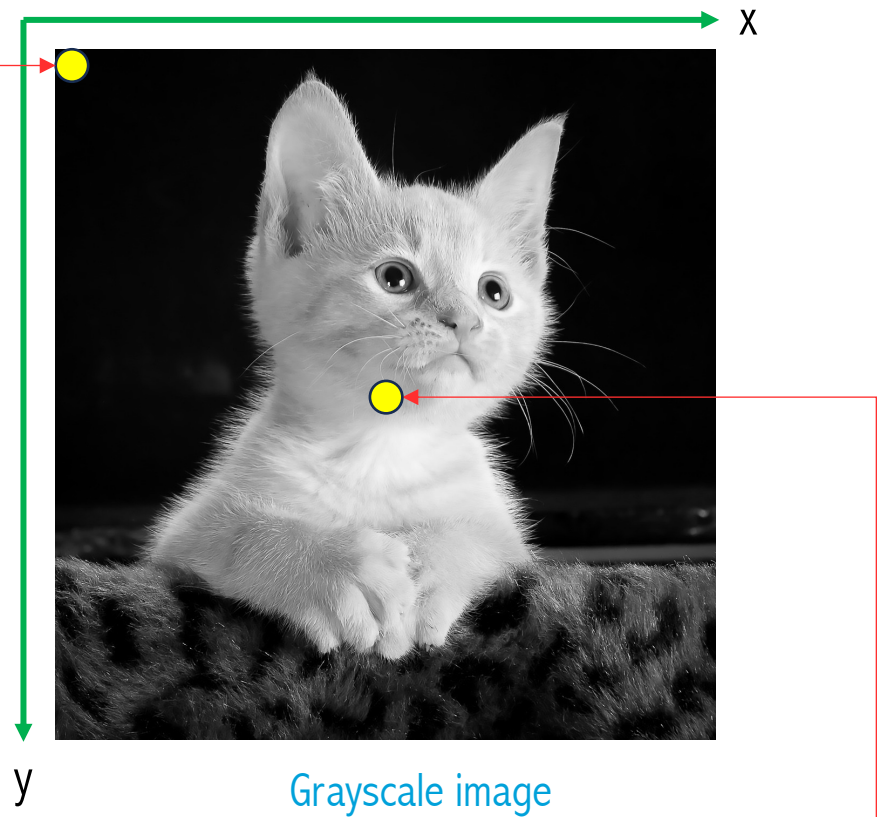
Convert Color Image to GrayScale Image

Grayscale image

Image Coordinate System: Grayscale Image



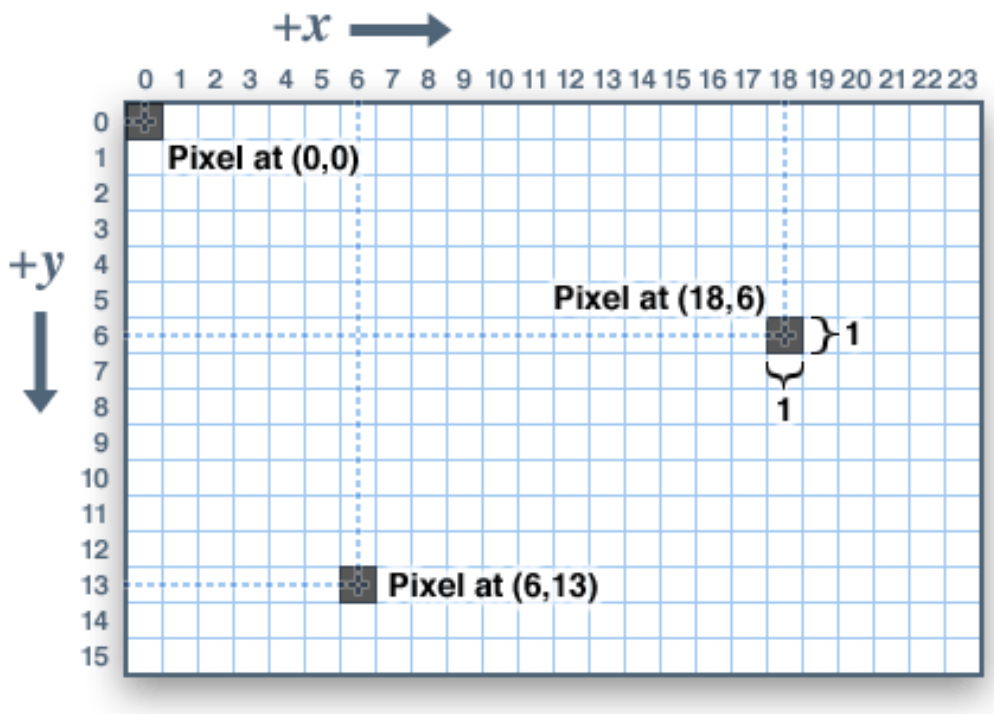

```
#Access the pixel value at position (x,y)
x = 0; y = 0
pixelValue = gray_image[y,x]
print("Pixel Value", pixelValue)

Pixel Value 0
```



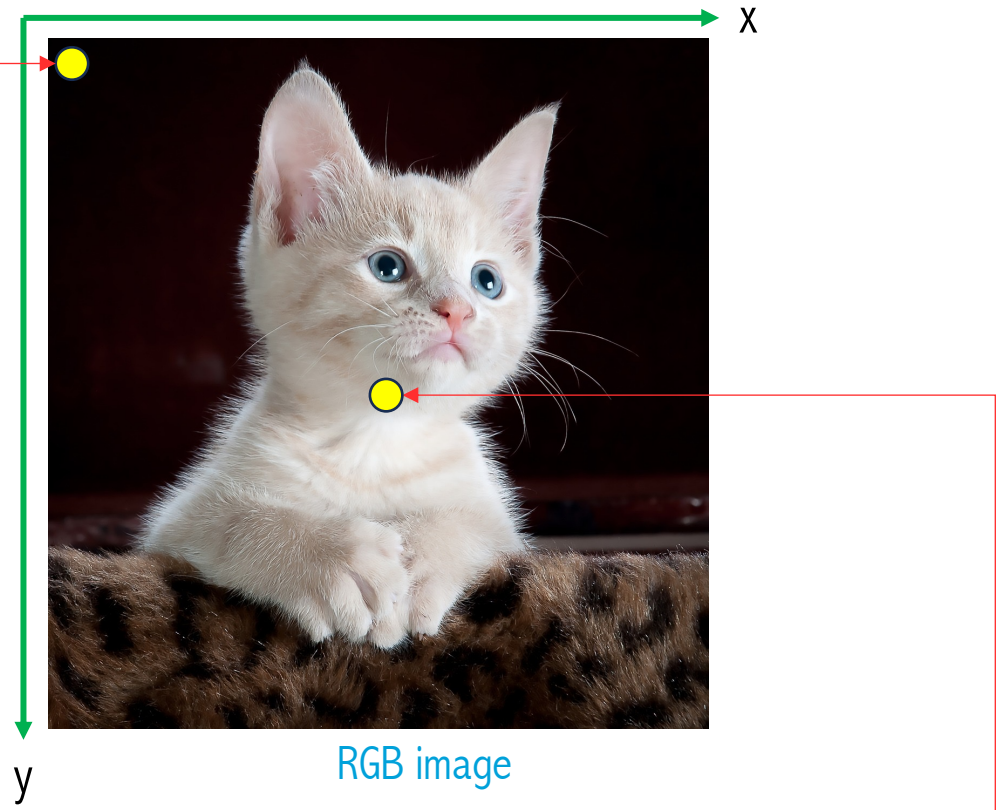
Grayscale image

```
#Access the pixel value at the image center (Grayscale)
width, height = gray_image.shape
x = int(width/2)
y = int(height/2)
pixelValue = gray_image[y,x]
print("Pixel Value", pixelValue)

Pixel Value 167
```

Image Coordinate System: RGB Image




```
#Access the pixel value at the image center (Color)
width, height, channels = color_img.shape
x = 0; y = 0
pixelValue = color_img[y,x]
print("Pixel Value", pixelValue)

Pixel Value [0 0 0]
```



RGB image

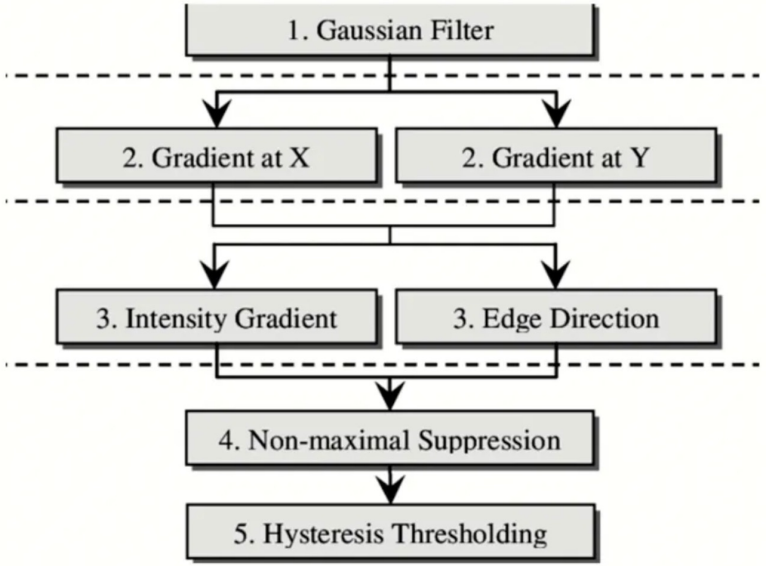
```
#Access the pixel value at the image center (Color)
width, height, channels = color_img.shape
x = int(width/2)
y = int(height/2)
pixelValue = color_img[y,x]
print("Pixel Value", pixelValue)

Pixel Value [156 167 171]
```

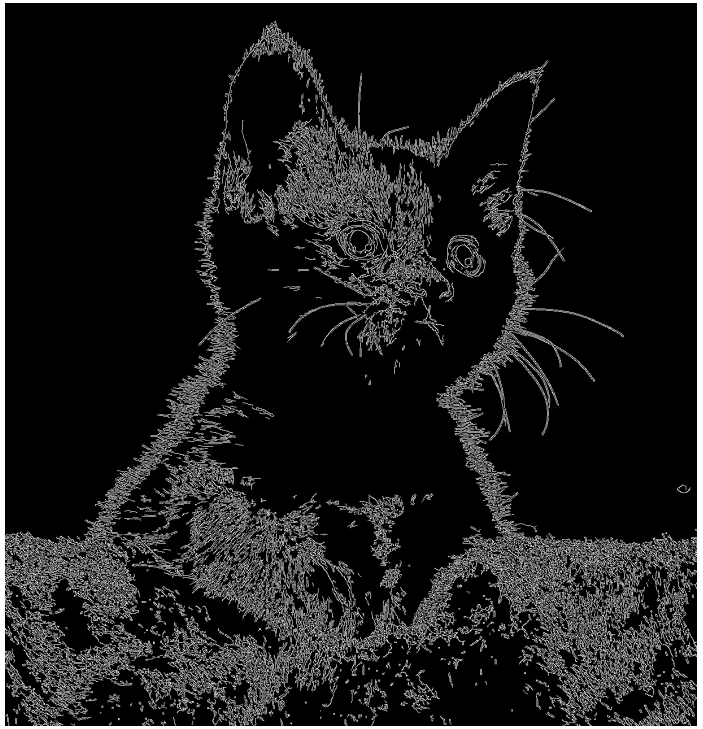
Outline

- Introduction to Computer Vision
- Edge Detection Algorithm
- Line Detection Algorithm
- Contour Detection Algorithm
- Shape Detection Algorithm
- Background Subtraction Algorithm

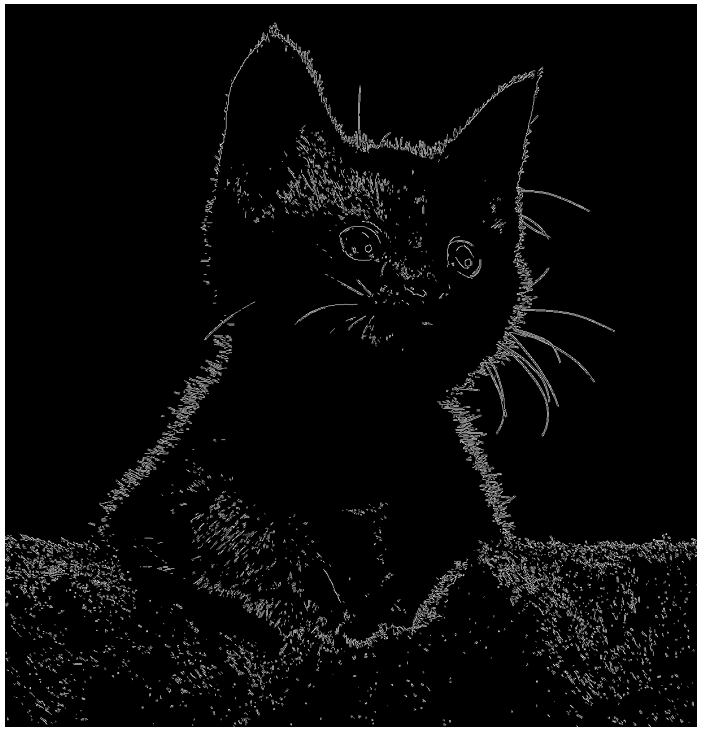
Edge Detection: Canny Algorithm

With edge detection, we can determine the objects on the image without other details





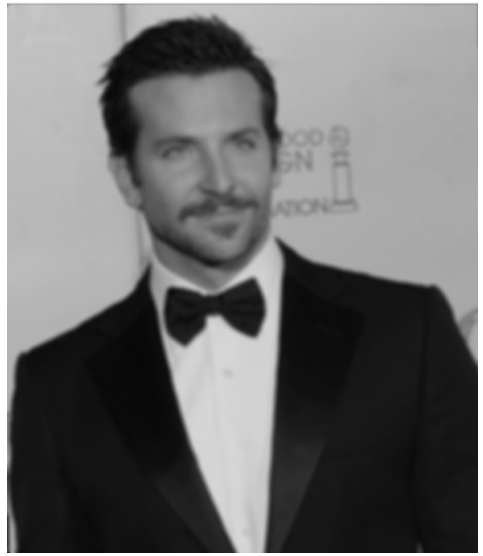
```
lower = 50 # Lower Threshold
upper = 150 # Upper threshold
```



```
lower = 250 # Lower Threshold
upper = 300 # Upper threshold
```

```
#Ege Detection Canny Algorithm
# Setting parameter values
t_lower = 250  # Lower Threshold
t_upper = 300  # Upper threshold

# Applying the Canny Edge filter
edge_canny = cv2.Canny(gray_image, t_lower, t_upper)

cv2_imshow(edge_canny)
```

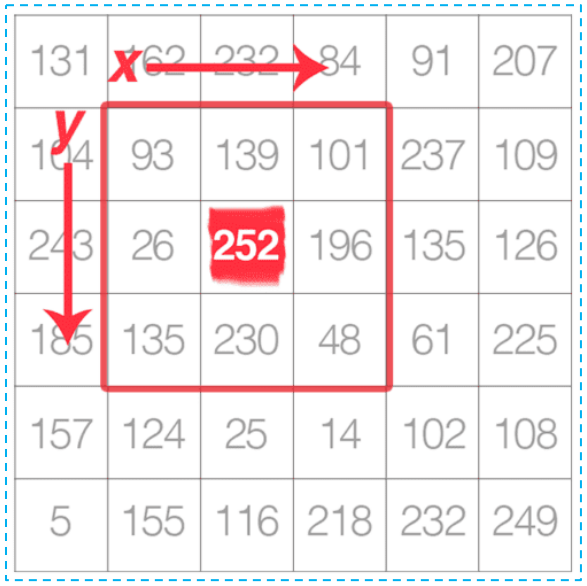
Gaussian Filter



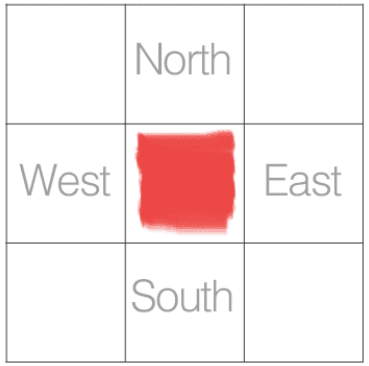
$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

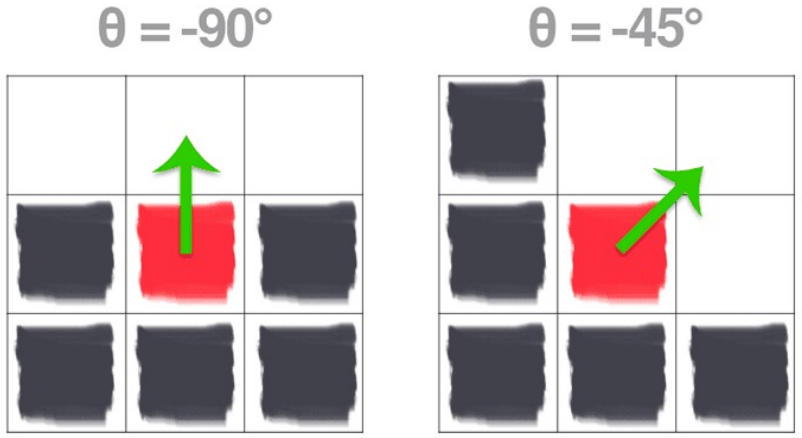
```python
def gaussian_kernel(size, sigma):
    if size % 2 == 0:
        size = size + 1

    max_point = size // 2  # both directions (x,y) maximum cell start point
    min_point = -max_point  # both directions (x,y) minimum cell start point

    K = np.zeros((size, size))  # kernel matrix
    for x in range(min_point, max_point + 1):
        for y in range(min_point, max_point + 1):
            value = (1 / (2 * np.pi * (sigma ** 2)) * np.exp((-(x ** 2 + y ** 2))
                                                             / (2 * (sigma ** 2))))
            K[x - min_point, y - min_point] = value

    return K

kernel = gaussian_kernel(5, 1.4)
img = cv2.imread("/Users/nguyendinhvinh2004gmail.com/Downloads/canny_edge_detector-m
img_gaussian = cv2.filter2D(img, -1, kernel)

cv2.imshow("img", img)
cv2.imshow("img_gaussian", img_gaussian)

cv2.waitKey(0)

close_window_os()
```

Image Gradient



An image gradient is defined as a directional change in image intensity





Vertical change $G_y = I(x, y + 1) - I(x, y - 1)$

Horizontal change $G_x = I(x + 1, y) - I(x - 1, y)$

The gradient magnitude is used to measure how strong the change in image intensity is

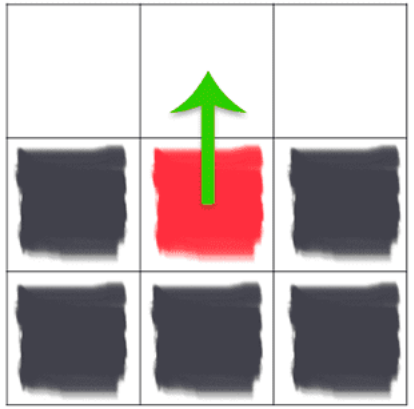
$$G = \sqrt{G_x^2 + G_y^2}$$

The gradient orientation is used to determine in which direction the change in intensity is pointing

$$\theta = \arctan2(G_y, G_x) \times \left(\frac{180}{\pi}\right)$$

Image Gradient

255	255	255
0		0
0	0	0



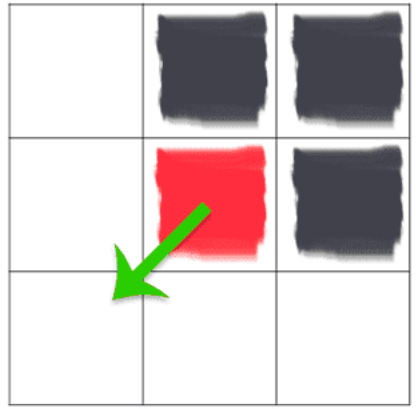
$$G = \sqrt{0^2 + (-255)^2} = 255$$

$G_x = 0 - 0 = 0$

$G_y = 0 - 255 = -255$

$$\theta = \arctan2(-255, 0) \times \left(\frac{180}{\pi}\right) = -90^\circ$$

255	0	0
255		0
255	255	255



$$G = \sqrt{(-255)^2 + 255^2} = 360.62$$

$$\theta = \arctan2(255, -255) \times \left(\frac{180}{\pi}\right) = 135^\circ$$

Image Gradient

X-gradient

Y-gradient

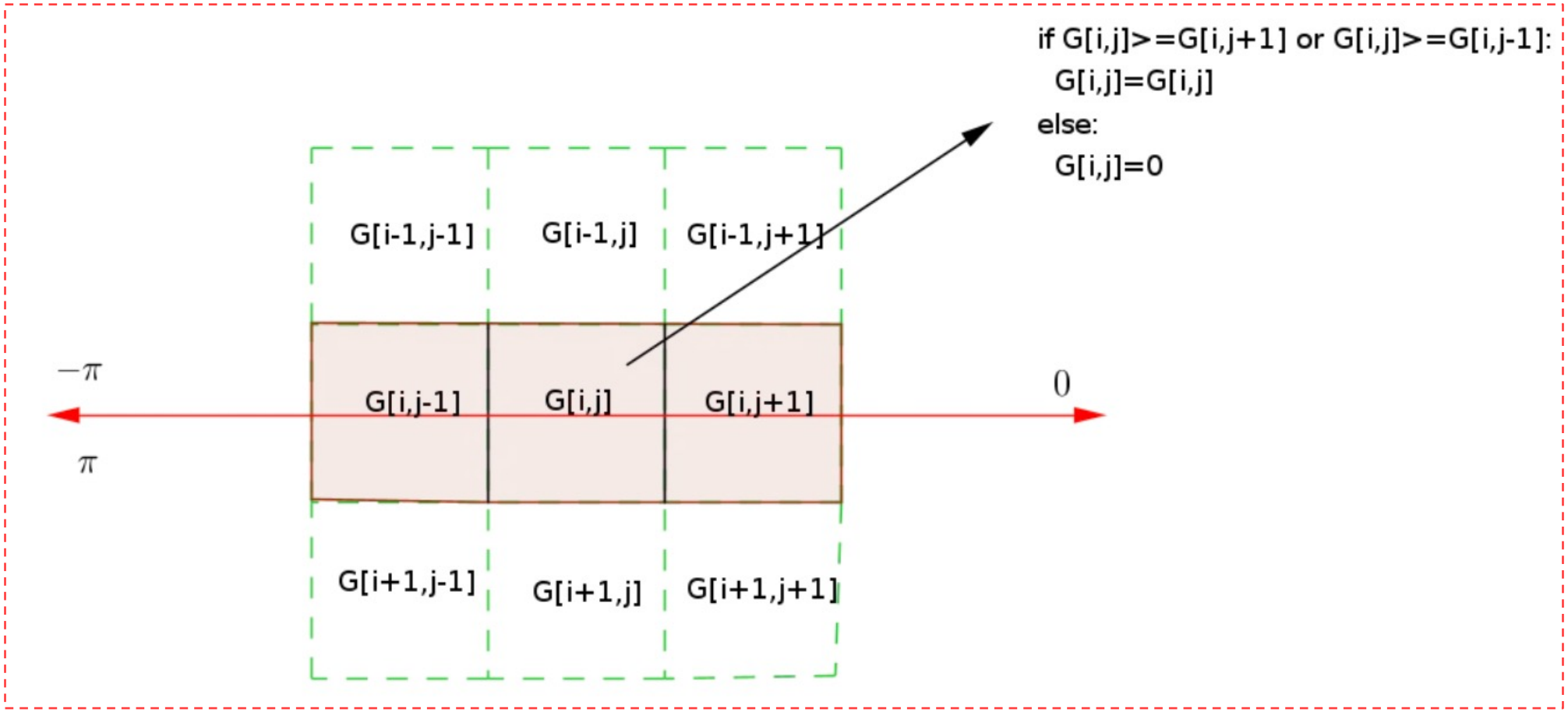
Gradient magnitude

Gradient Direction

```python
img_gaussian = np.float64(img_gaussian)
mask_x = np.zeros((2, 1))
mask_x[0] = -1
mask_x[1] = 1

I_x = cv2.filter2D(img_gaussian, -1, mask_x)
mask_y = mask_x.T
I_y = cv2.filter2D(img_gaussian, -1, mask_y)

Gm = (I_x ** 2 + I_y ** 2) ** 0.5
Gd = np.rad2deg(np.arctan2(I_y, I_x))

cv2.imshow("I_x", I_x)
cv2.imshow("I_y", I_y)
cv2.imshow("Gm", Gm)
cv2.imshow("Gd", Gd)

cv2.waitKey(0)
close_window_os()
```

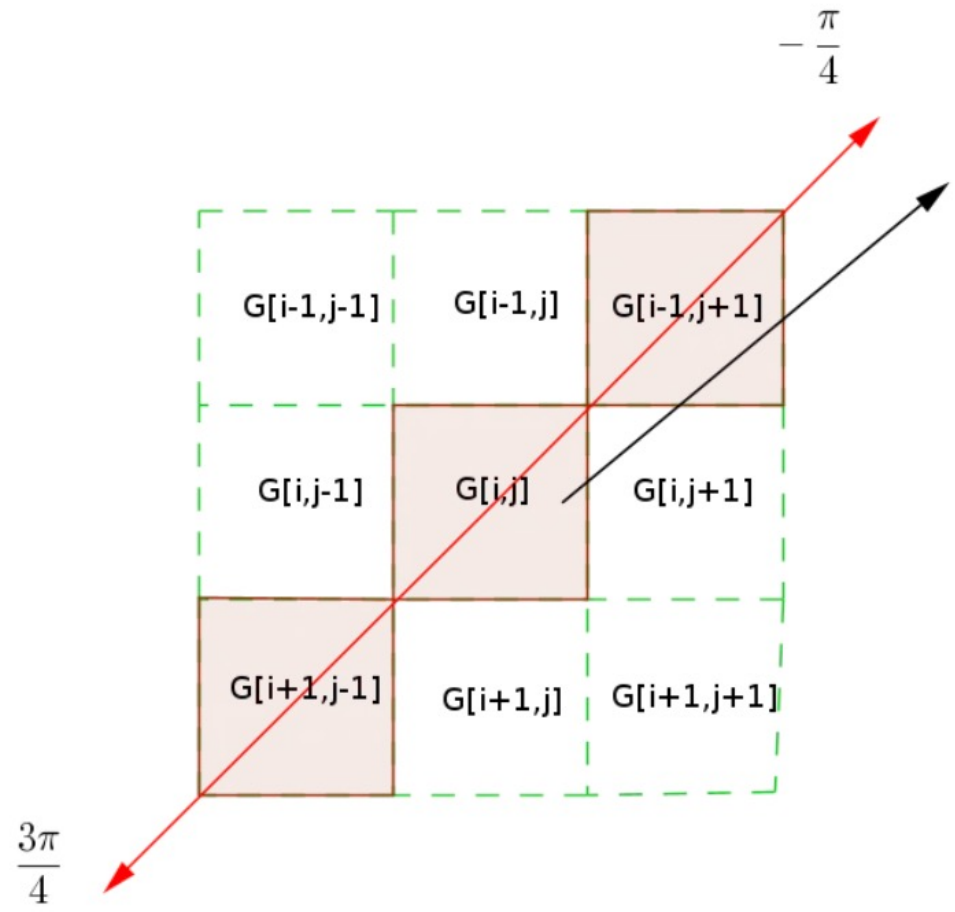
Gradient magnitude edges thicker than many edge detection. Next step is the make edges one pixel thick.

Non-Maximal Suppression



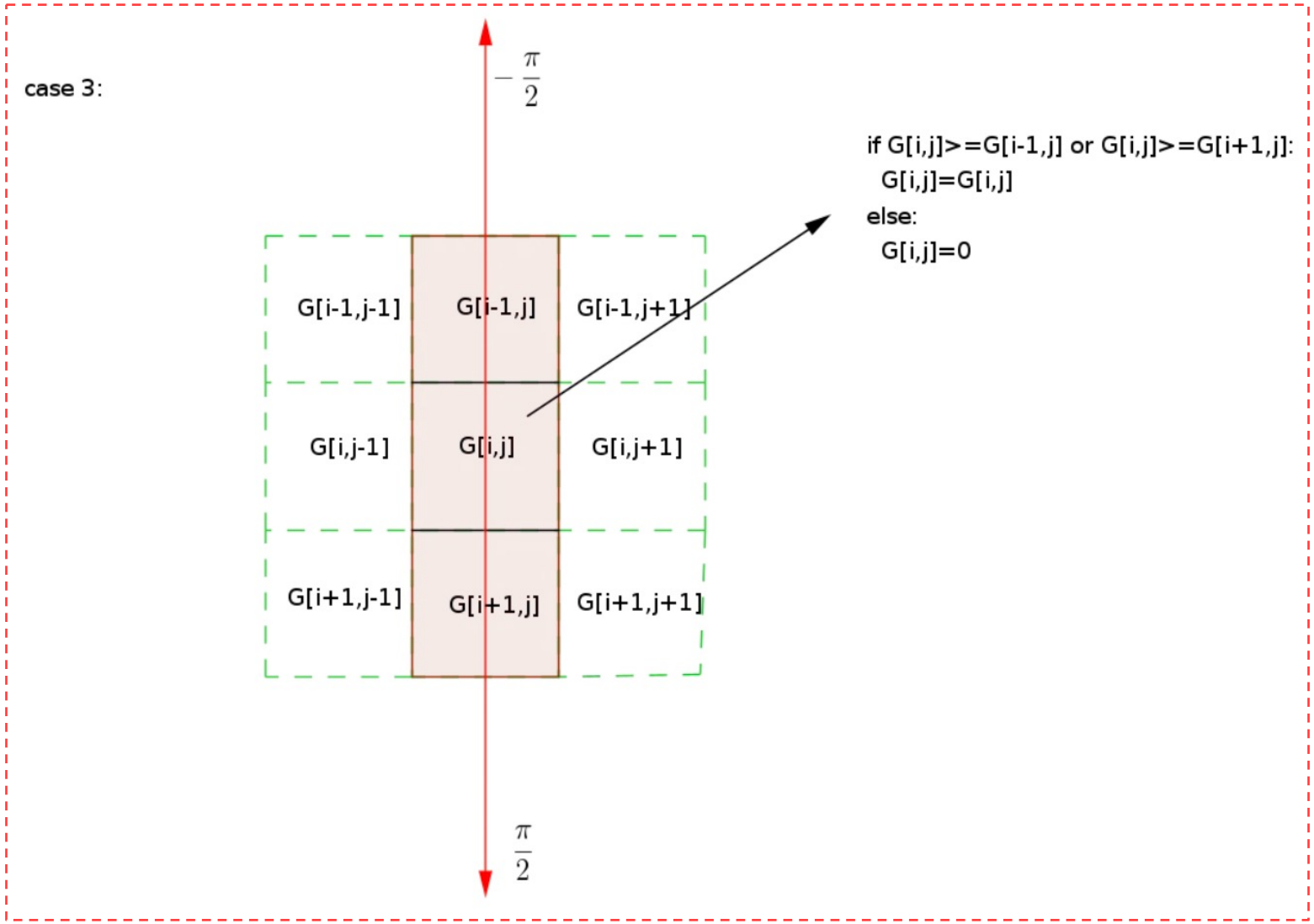
Non-Maximal Suppression

case 2:



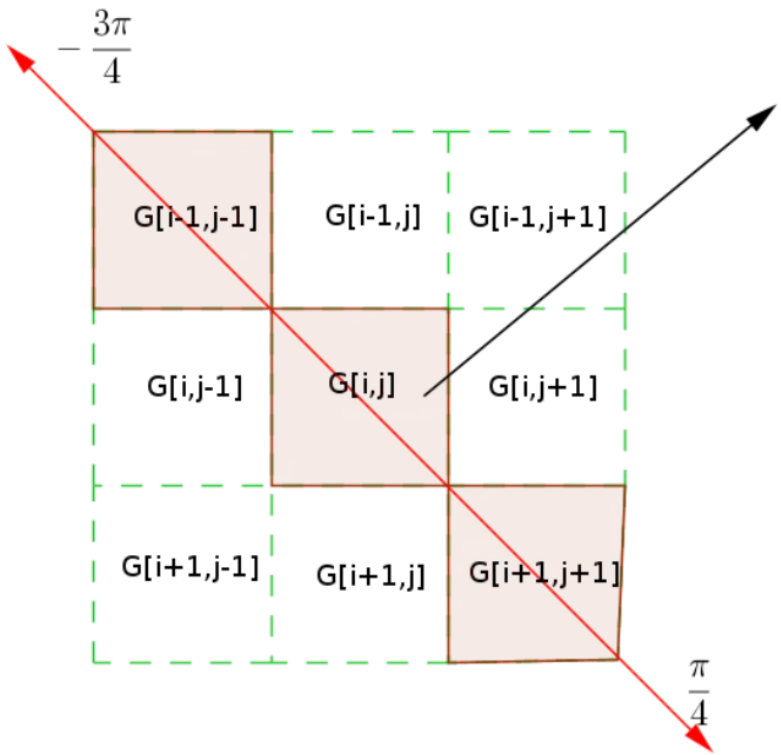

```
if G[i,j]>=G[i-1,j+1]or G[i,j]>=G[i+1,j-1]:
  G[i,j]=G[i,j]
else:
  G[i,j]=0
```

Non-Maximal Suppression



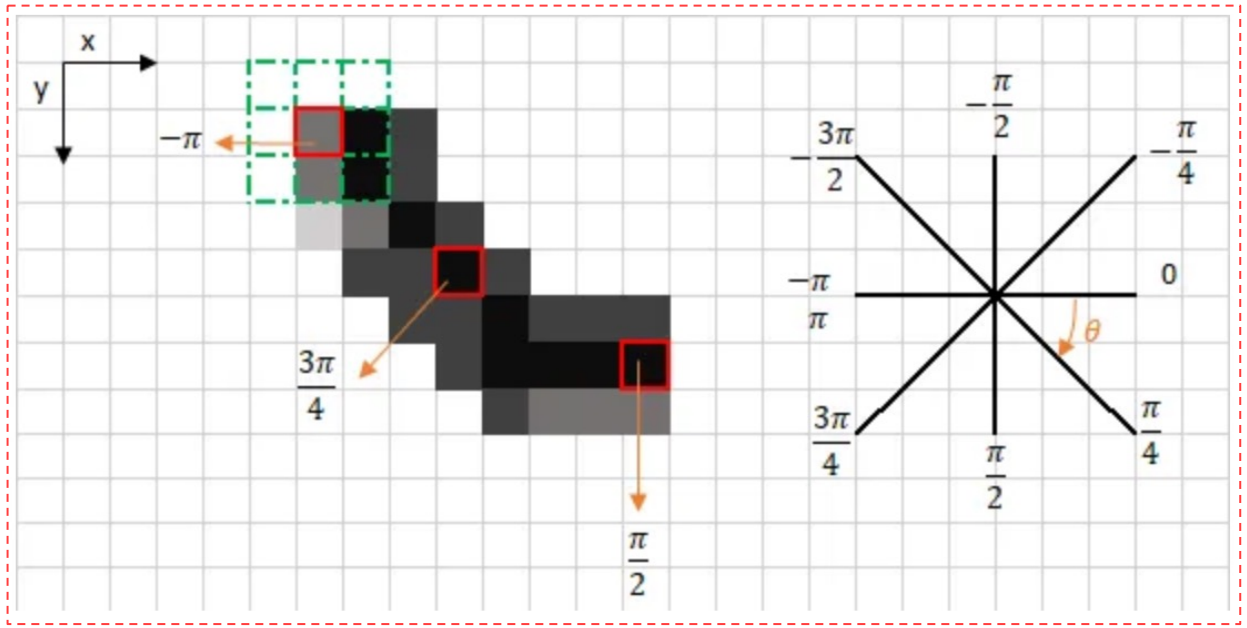
Non-Maximal Suppression

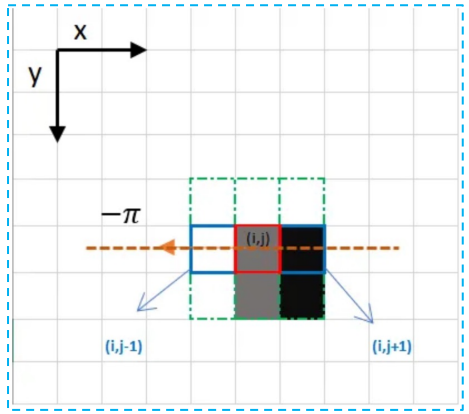
case 4:

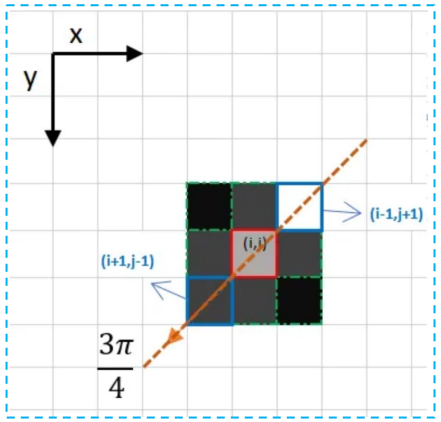



```
if G[i,j]>=G[i-1,j-1]or G[i,j]>=G[i+1,j+1]:
  G[i,j]=G[i,j]
else:
  G[i,j]=0
```

Non-Maximal Suppression







Hãy cho biết kết quả khi thực hiện Non-Maximal Suppression trên 2 ví dụ trên?

Check every pixel on the Gradient magnitude image and choose 2 neighbor of the pixel according to Gradient Direction. If center pixel is larger than the both neighbors then keep it, otherwise set the pixel to 0.

Non-Maximal Suppression

```python
def f_NMS(Gm, Gd):
    num_rows, num_cols = Gm.shape[0], Gm.shape[1]
    Gd_bins = 45 * (np.round(Gd / 45))

    G_NMS = np.zeros(Gm.shape)

    neighbor_a, neighbor_b = 0., 0.

    for r in range(1, num_rows - 1):
        for c in range(1, num_cols - 1):
            angle = Gd_bins[r, c]
            if angle == 180. or angle == -180. or angle == 0.:
                neighbor_a, neighbor_b = Gm[r + 1, c], Gm[r - 1, c]
            elif angle == 90. or angle == -90.:
                neighbor_a, neighbor_b = Gm[r, c - 1], Gm[r, c + 1]
            elif angle == 45. or angle == -135.:
                neighbor_a, neighbor_b = Gm[r + 1, c + 1], Gm[r - 1, c - 1]
            elif angle == -45. or angle == 135.:
                neighbor_a, neighbor_b = Gm[r - 1, c + 1], Gm[r + 1, c - 1]
            else:
                print("error")
                return

            if Gm[r, c] > neighbor_a and Gm[r, c] > neighbor_b:
                G_NMS[r, c] = Gm[r, c]

    return G_NMS
```

Gradient magnitude

Gradient NMS

Double threshold

Gradient NMS

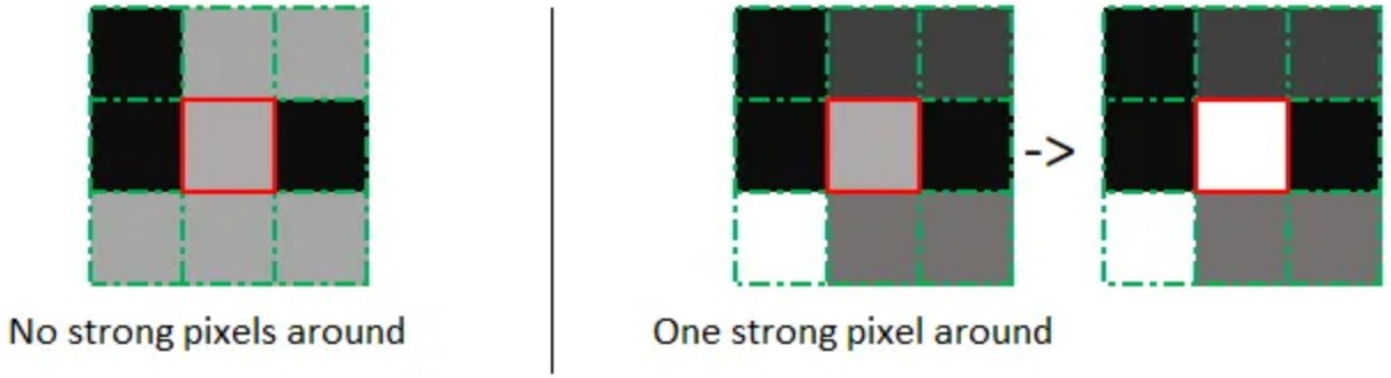
The double threshold step aims at identifying 3 kinds of pixels: strong, weak, and non-relevant

Double Threshold

```python
def threshold_edge(img, weak_pixel = 75, strong_pixel = 255):
    highT = 0.15
    lowT = 0.05

    highThreshold = img.max() * highT;
    lowThreshold = highThreshold * lowT;

    M, N = img.shape
    res = np.zeros((M,N), dtype=np.uint8)

    strong_i, strong_j = np.where(img >= highThreshold)
    zeros_i, zeros_j = np.where(img < lowThreshold)

    weak_i, weak_j = np.where((img <= highThreshold) & (img >= lowThreshold))

    res[strong_i, strong_j] = strong_pixel
    res[weak_i, weak_j] = weak_pixel

    return res
```

Edge Tracking by Hysteresis

Based on the threshold results, the hysteresis consists of transforming weak pixels into strong ones



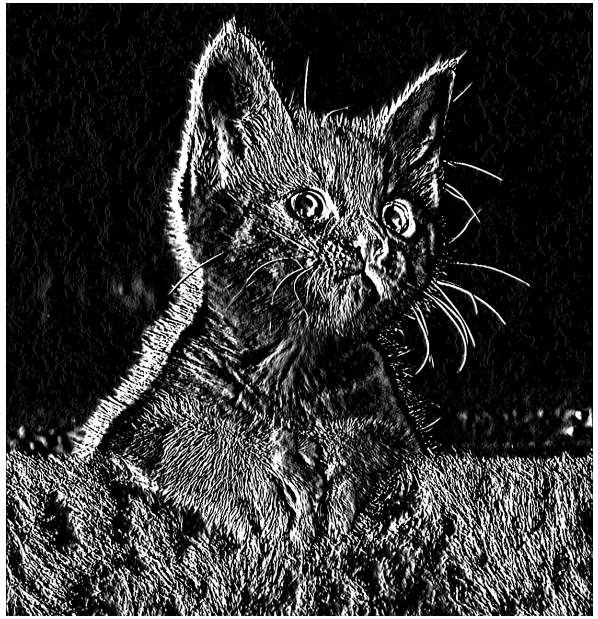

```python
def hysteresis(img, weak, strong=255):
    M, N = img.shape
    for i in range(1, M-1):
        for j in range(1, N-1):
            if (img[i,j] == weak):
                try:
                    if ((img[i+1, j-1] == strong) or (img[i+1, j] == strong) or (img[i+1,
                        or (img[i, j-1] == strong) or (img[i, j+1] == strong)
                        or (img[i-1, j-1] == strong) or (img[i-1, j] == strong) or (img[i-
                        img[i, j] = strong
                    else:
                        img[i, j] = 0
                except IndexError as e:
                    pass
    return img
```

Double Threshold

Final Result

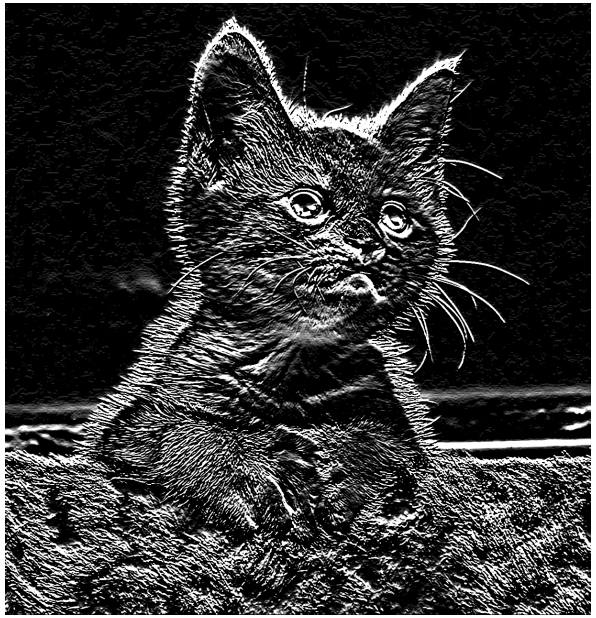
AI VIETNAM
All-in-One Course

Edge Detection: Sobel Algorithm



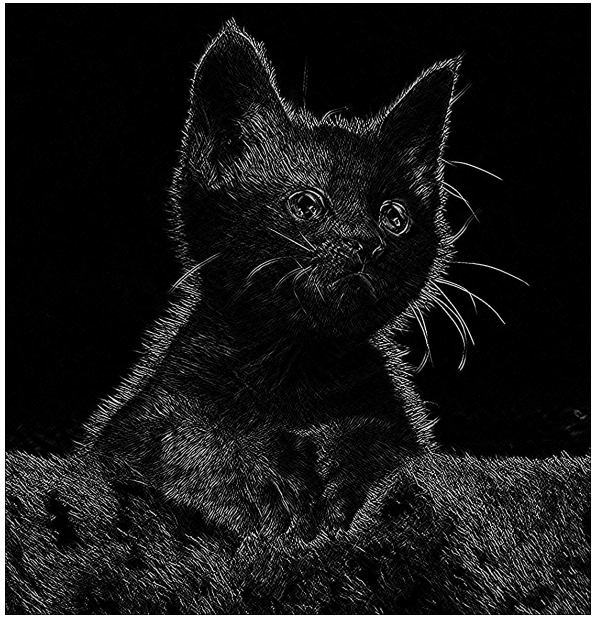
Sobel in X-direction

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



Sobel in Y-direction

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



Sobel in XY-direction

$$\mathbf{G} = \sqrt{\mathbf{G}_x^{\,2} + \mathbf{G}_y^{\,2}}$$

```
#Ege Detection by Sobel Algorithm

img_blur = cv2.GaussianBlur(gray_image, (3,3), 0)

# Sobel Edge Detection
sobelx = cv2.Sobel(src=img_blur, ddepth=-1, dx=1, dy=0, ksize=5) # Sobel Edge Detection on the X axis
sobely = cv2.Sobel(src=img_blur, ddepth=-1, dx=0, dy=1, ksize=5) # Sobel Edge Detection on the Y axis
sobelxy = cv2.Sobel(src=img_blur, ddepth=-1, dx=1, dy=1, ksize=5) # obel Edge Detection on the X, Y axis

cv2_imshow(sobelx)
cv2_imshow(sobely)
cv2_imshow(sobelxy)
```

Canny Vs Sobel Edge Detection

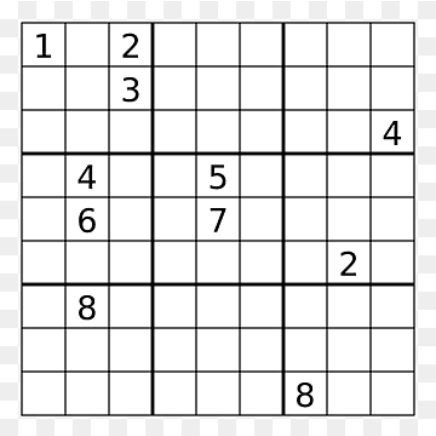
(a) (b)

Which one is the result of Canny (or Sobel) algorithm?

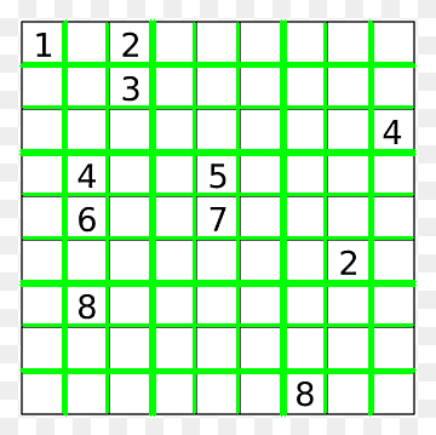
Outline

- Introduction to Computer Vision
- Edge Detection Algorithm
- Line Detection Algorithm
- Contour Detection Algorithm
- Shape Detection Algorithm
- Background Subtraction Algorithm

Line Detection: Hough Transform



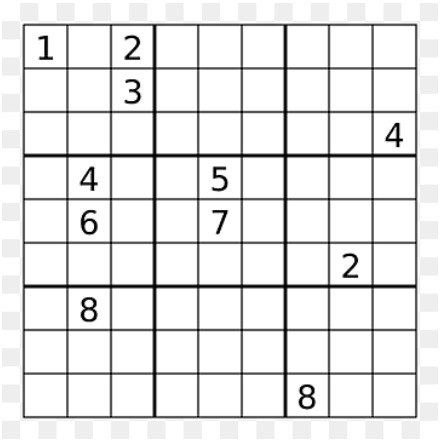
Input Image



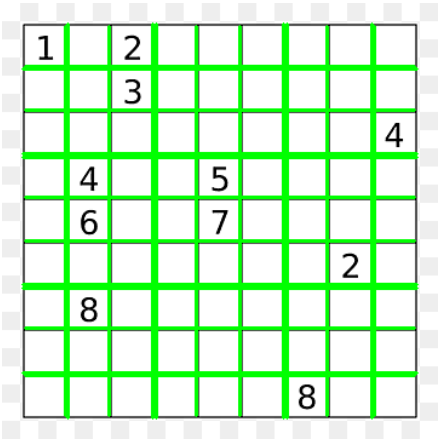
Line Detection Result

```
# Read image
image = cv2.imread('/content/sudoku.png')

# Convert image to grayscale
gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)

# Use canny edge detection
edges = cv2.Canny(gray,50,200)

# Apply HoughLinesP method to
# to directly obtain line end points
lines_list =[]
lines = cv2.HoughLinesP(
            edges, # Input edge image
            1, # Distance resolution in pixels
            np.pi/180, # Angle resolution in radians
            threshold=200, # Min number of votes for valid line
            minLineLength=5, # Min allowed length of line
            maxLineGap=10 # Max allowed gap between line for joining them
            )
```

Line Detection: Hough Transform



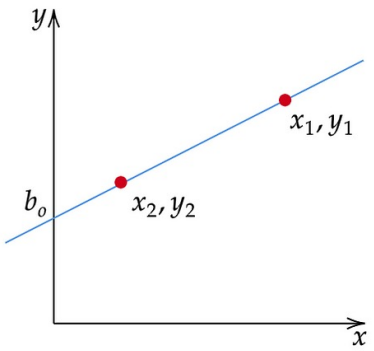
Input Image

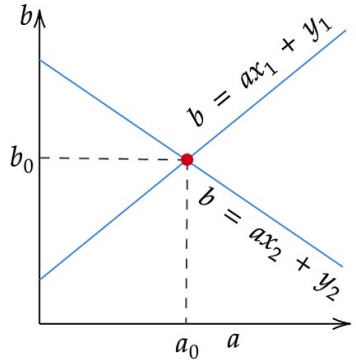


Line Detection Result

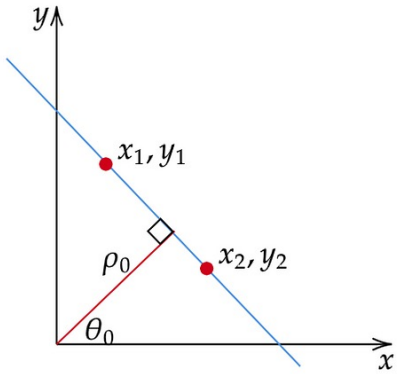
Line in the image space can be expressed with two variables

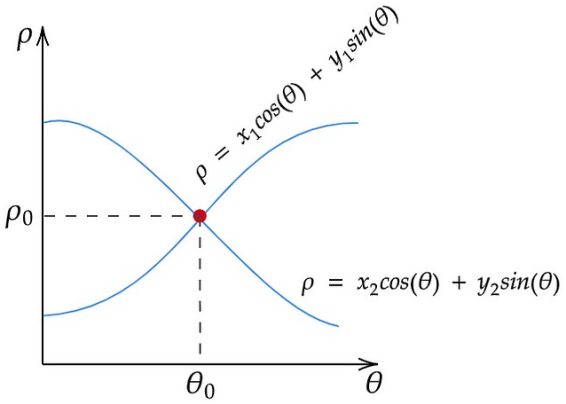
$y = ax + b$



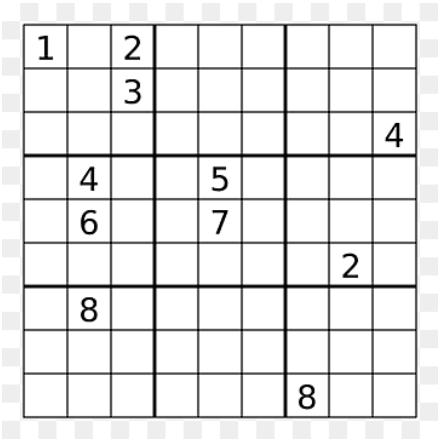


$\rho = x\cos(\theta) + y\sin(\theta)$

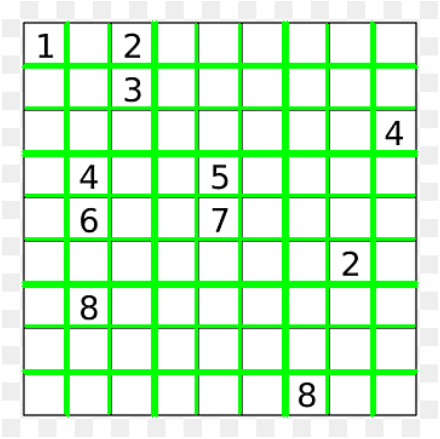




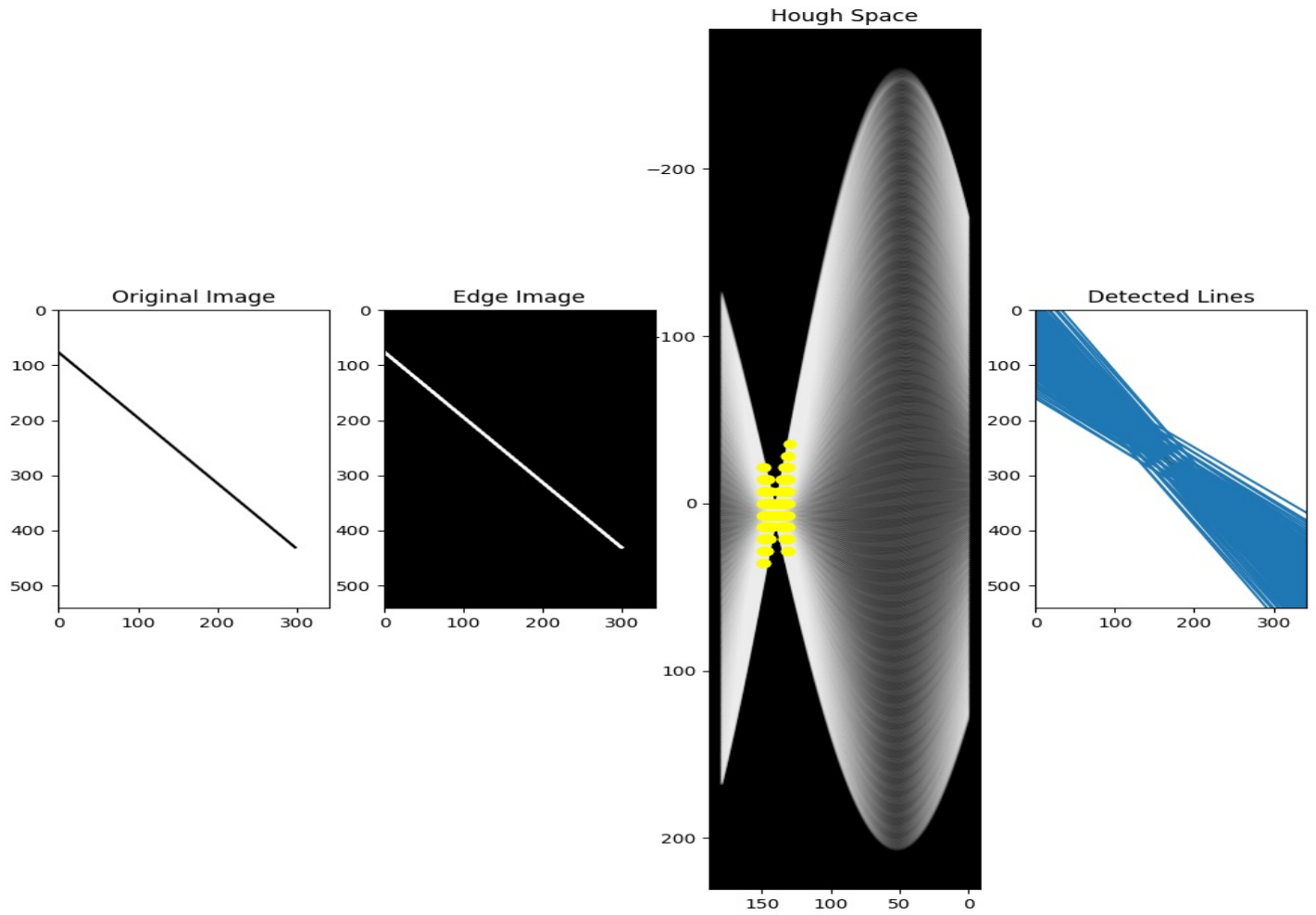
Line Detection: Hough Transform



Input Image



Line Detection Result



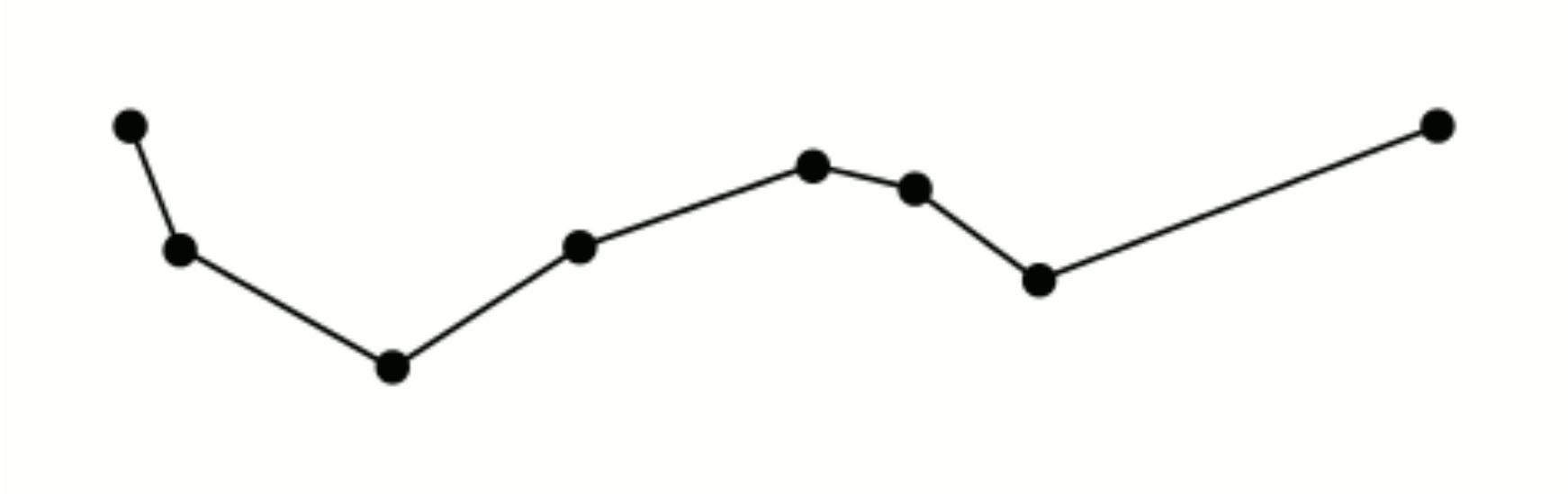
The yellow dots in the Hough Space indicate that lines exist and are represented by the θ and ρ pairs.

Outline

- Introduction to Computer Vision
- Edge Detection Algorithm
- Line Detection Algorithm
- Contour Detection Algorithm
- Shape Detection Algorithm
- Background Subtraction Algorithm

Contour Detection Algorithm

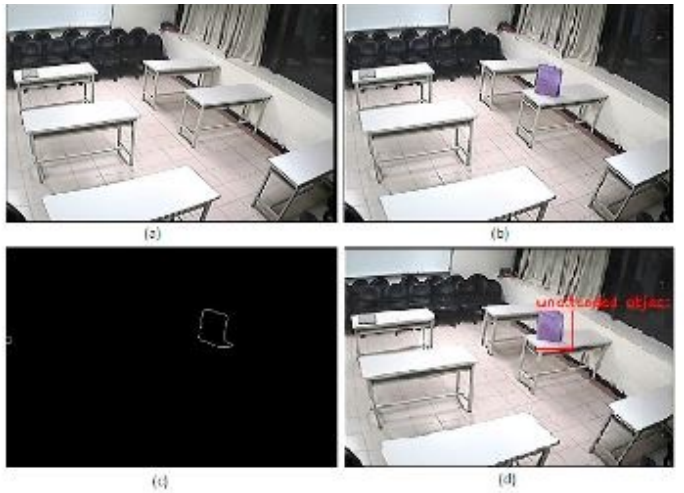
Contours detection is a process can be explained simply as a curve joining all the continuous points (along with the boundary), having same colour or intensity



Contour Approximation

Application of Contours in Computer Vision

Motion Detection



Unattended object detection

Background / Foreground Segmentation

Contour Tracing: Square Method

Square Tracing Algorithm

Demonstration

- Every time you find yourself standing on a black pixel, turn left
- Every time you find yourself standing on a white pixel, turn right, until you encounter the **start** pixel again.
- The black pixels you walked over will be the contour of the pattern.

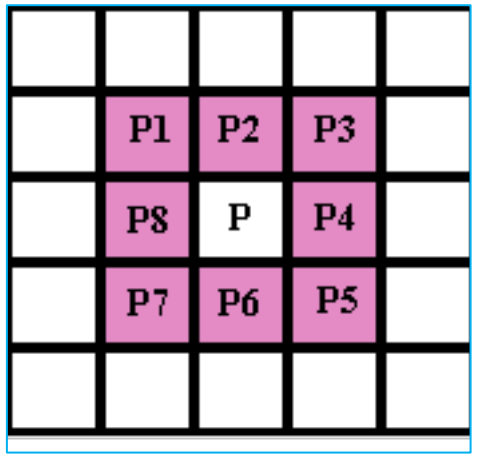
Demonstration:

A reason to change the stopping criterion

Demonstration:

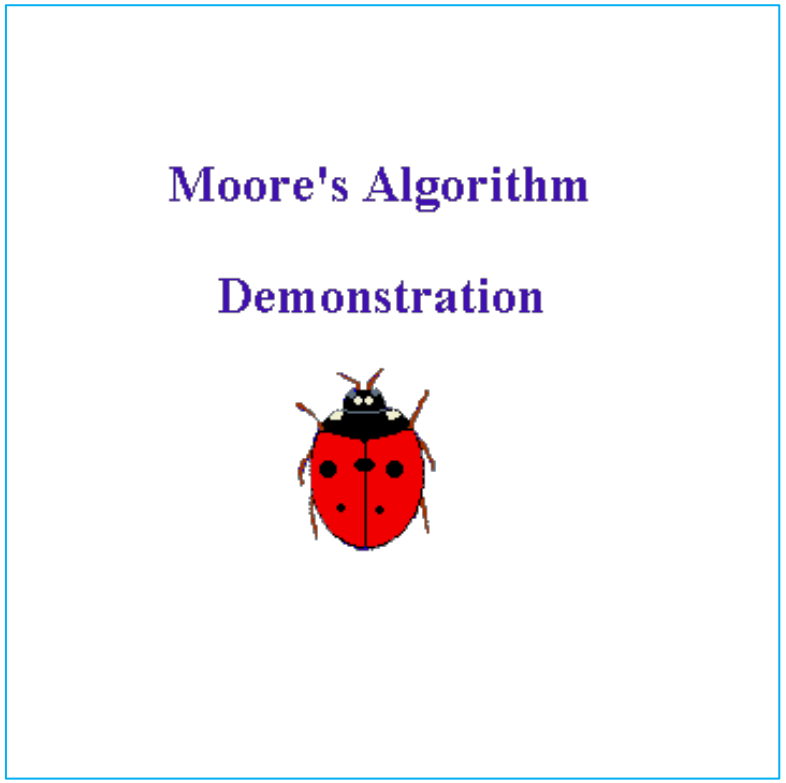
Algorithm fails to contour trace an 8-connected pattern

Contour Tracing: Moore-Neighbor



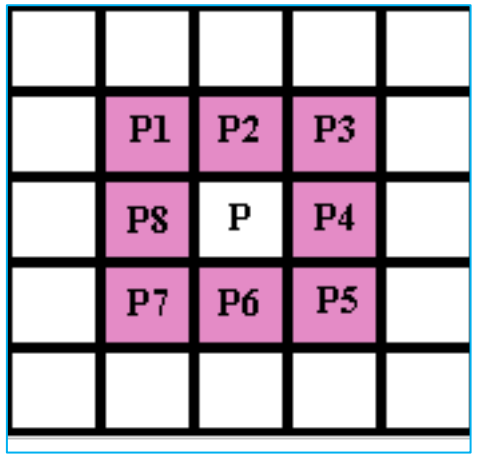
Moore neighborhood

8-neighbors or ***indirect neighbors***



Every time you hit a black pixel, **P**, backtrack i.e. go back to the white pixel you were previously standing on, then, go **around** pixel **P** in a clockwise direction, visiting each pixel in its Moore neighborhood, until you hit a black pixel.
The algorithm terminates when the start pixel is visited for a second time. The black pixels you walked over will be the contour of the pattern.

Contour Tracing: Radial Sweep



Moore neighborhood

8-neighbors or ***indirect neighbors***

Radial Sweep

Demonstration

Every time you locate a new boundary pixel, make it your current pixel, **P**, and draw an **imaginary line segment** joining **P** to the **previous** boundary pixel. Then, **rotate** the segment about **P** in a clockwise direction until it hits a black pixel in **P**'s Moore neighborhood. Rotating the segment is identical to checking each pixel in the Moore neighborhood of **P**.

Contour Tracing: Theo Pavlidis

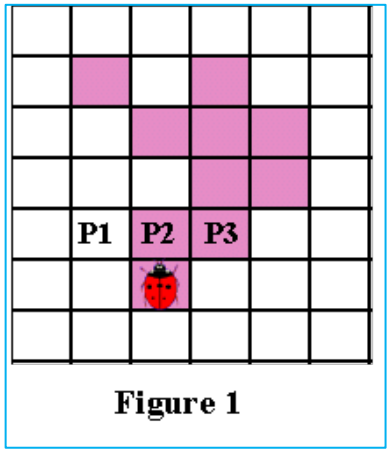


Figure 1

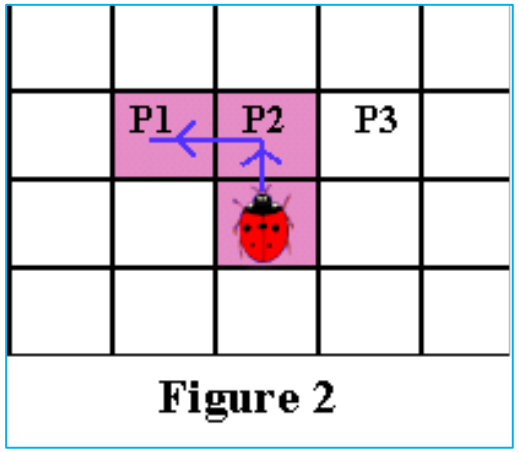


Figure 2

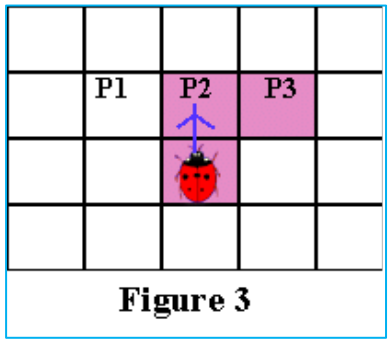


Figure 3

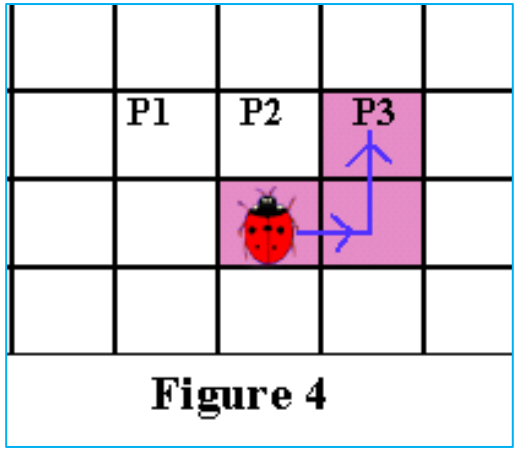
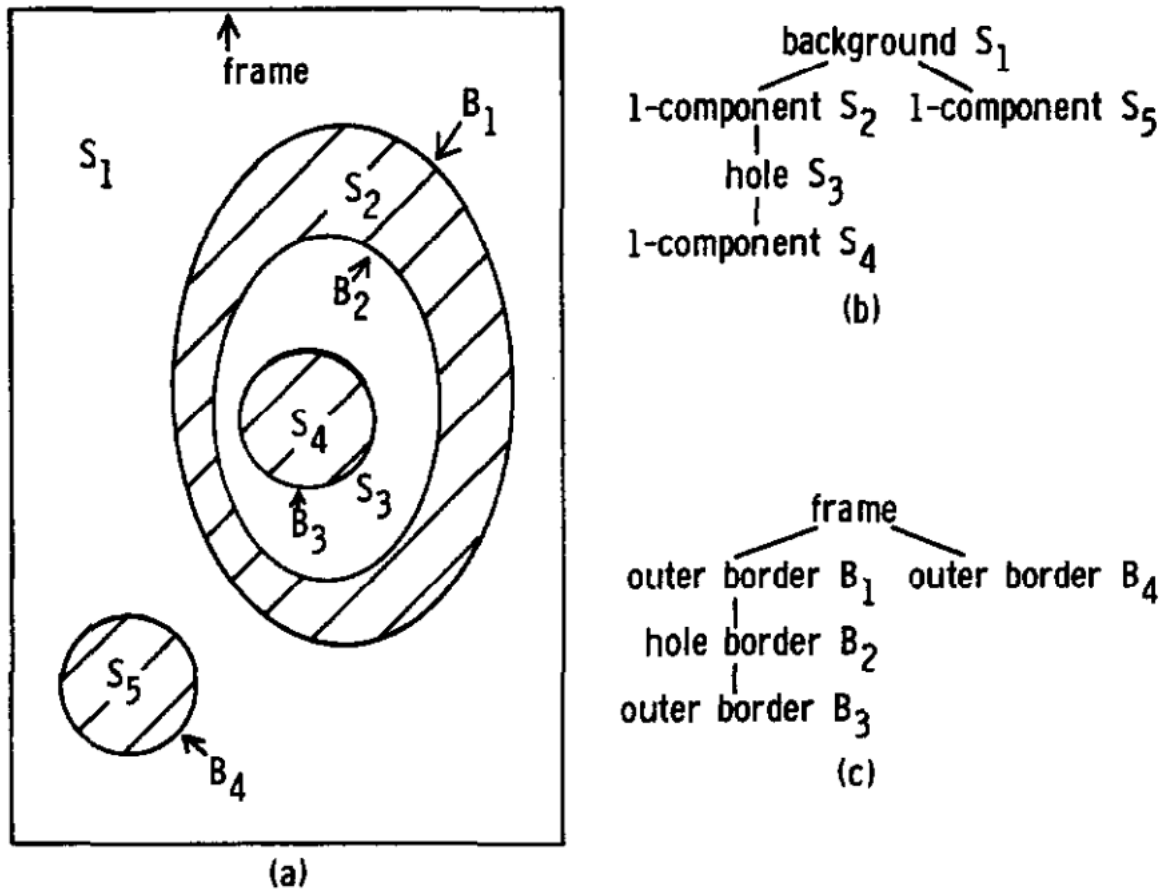


Figure 4

Theo Pavlidis' Algorithm

Demonstration

Contour Tracing: Suzuki's Contour



Surroundness among connected components (b) and among borders (c).

COMPUTER VISION, GRAPHICS, AND IMAGE PROCESSING 30, 32–46 (1985)

Topological Structural Analysis of Digitized Binary Images by Border Following

SATOSHI SUZUKI*

Graduate School of Electronic Science and Technology, Shizuoka University, Hamamatsu 432, Japan

AND

KEIICHI ABE†

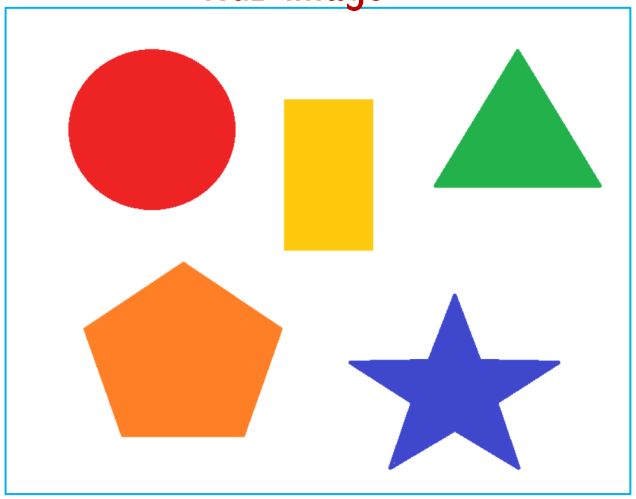
Department of Computer Science, Shizuoka University, Hamamatsu 432, Japan

Received December 16, 1983

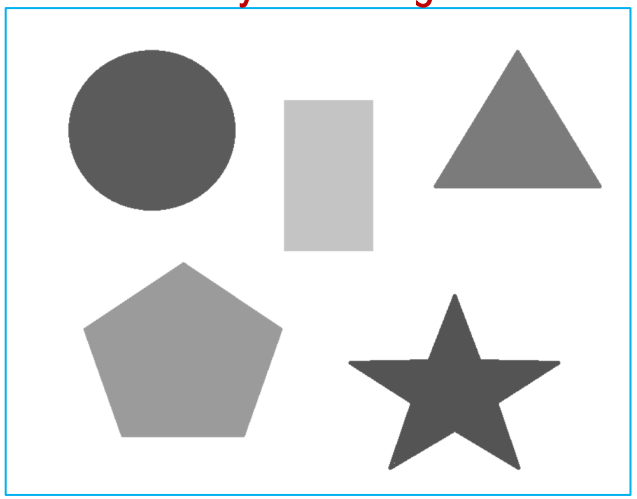
Contour Detection using OpenCV: https://learnopencv.com/contour-detection-using-opencv-python-c/

Contour Detection Algorithm

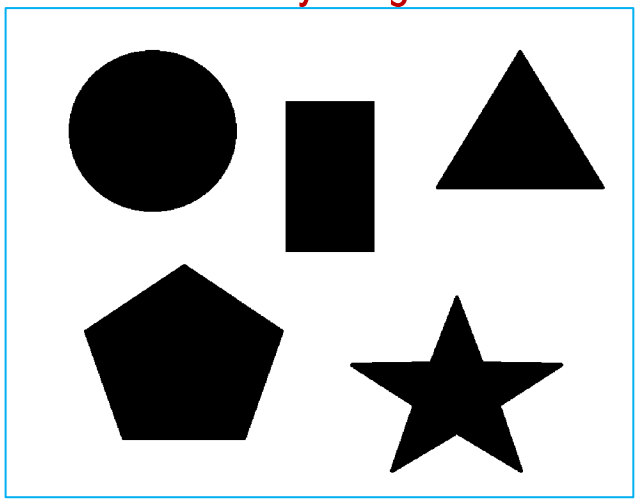
RGB image



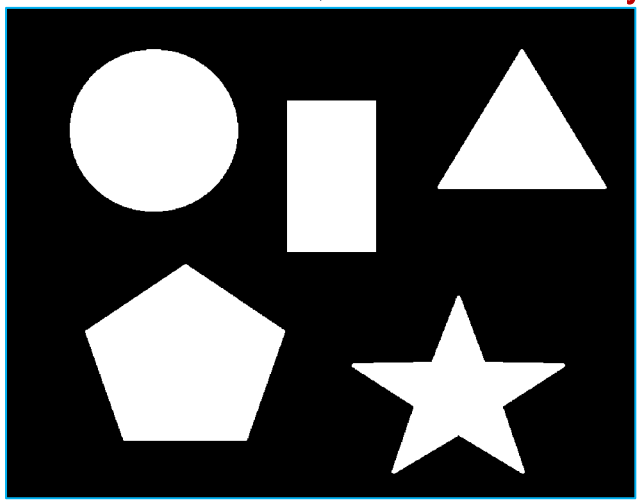
Grayscale image



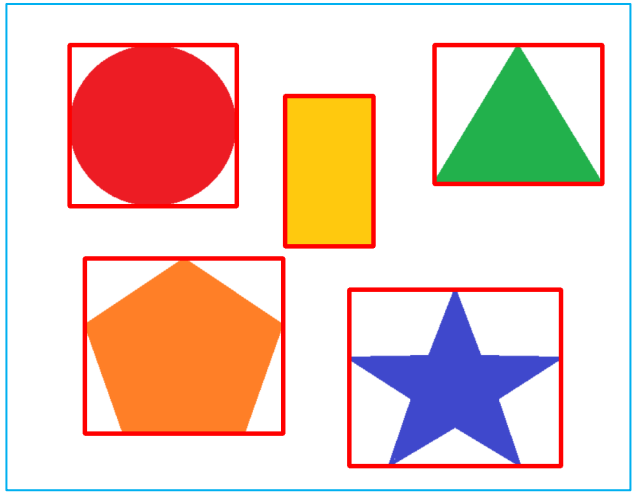
Binary Image



Inverse Binary Image



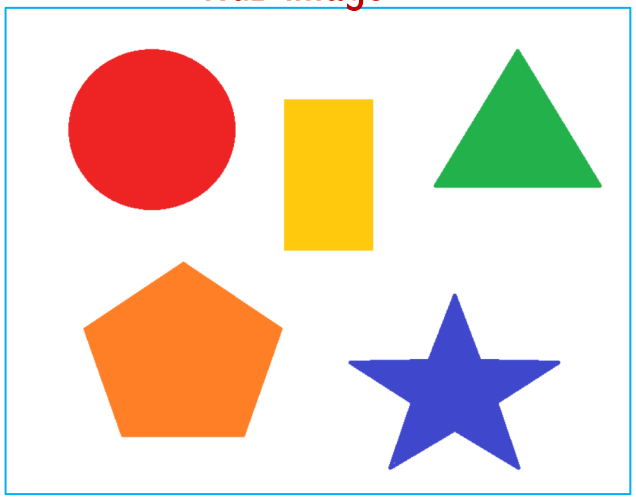
Final Contour Result



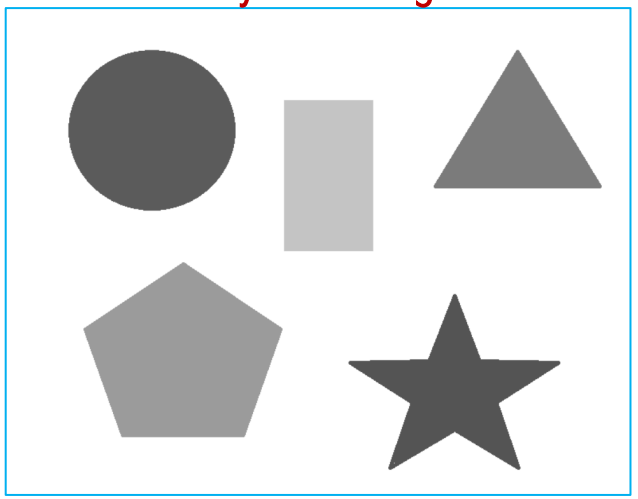
Interesting applications, such as image-foreground extraction, simple-image segmentation, detection and recognition.

Contour Detection Algorithm

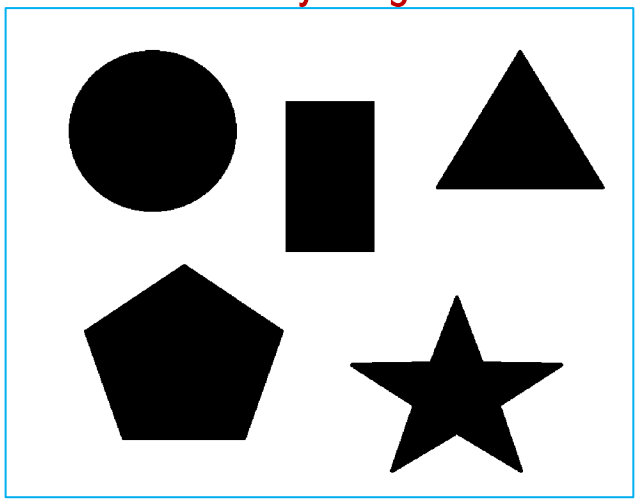
RGB image



Grayscale image



Binary Image



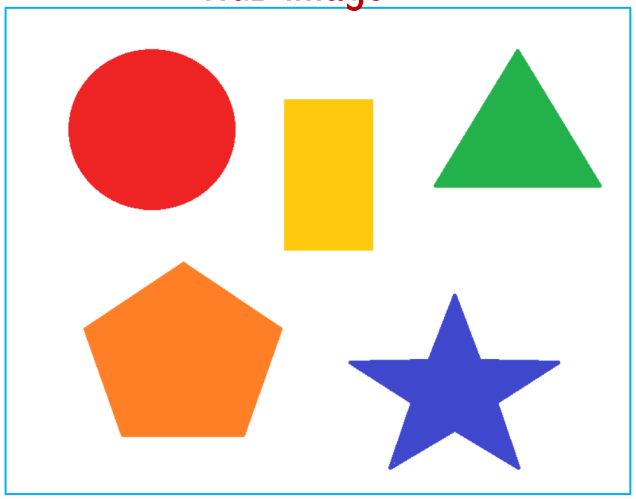
Contours are defined as the line joining all the points along the boundary of an image that are having the same intensity

```python
# Read image for contour detection
input_image = cv2.imread("/content/shapes.png")

# Make a copy to draw bounding box
input_image_cpy = input_image.copy()

# Convert input image to grayscale
gray_img = cv2.cvtColor(input_image, cv2.COLOR_BGR2GRAY)
```

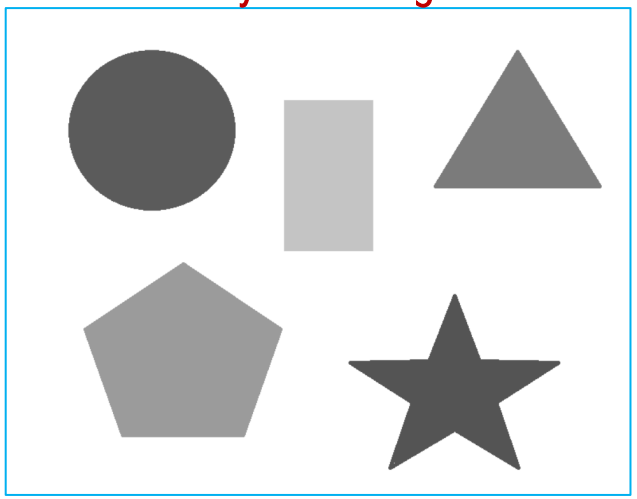
```python
# Convert the grayscale image to binary
ret, binary_img = cv2.threshold(
    gray_img, 200, 255, cv2.THRESH_BINARY)
```

Contour Detection Algorithm

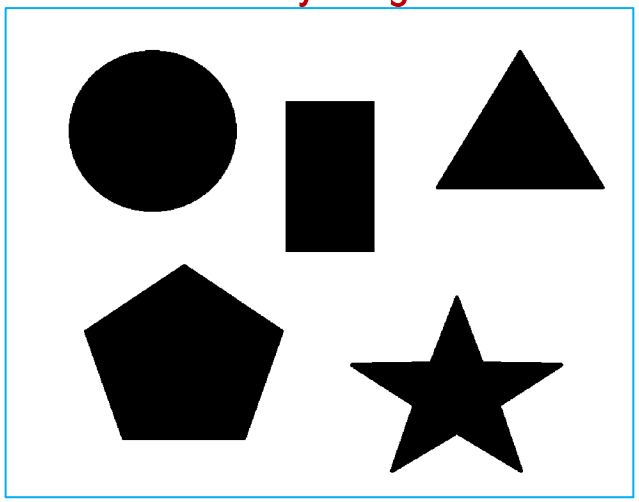
RGB image



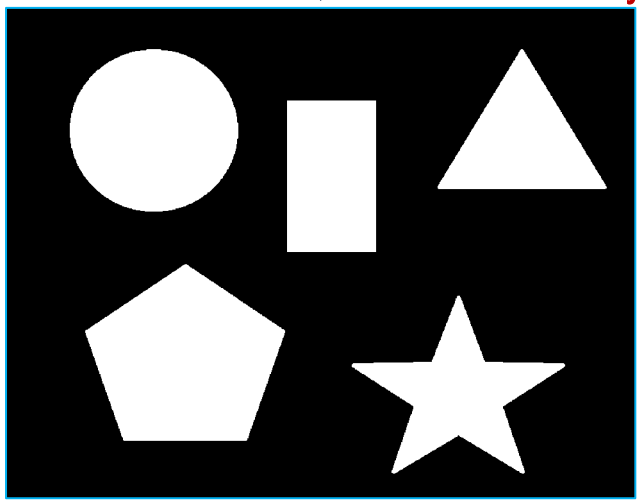
Grayscale image



Binary Image



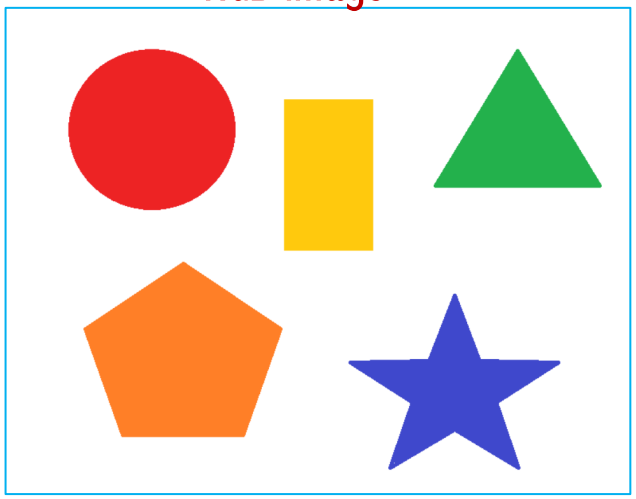
Inverse Binary Image



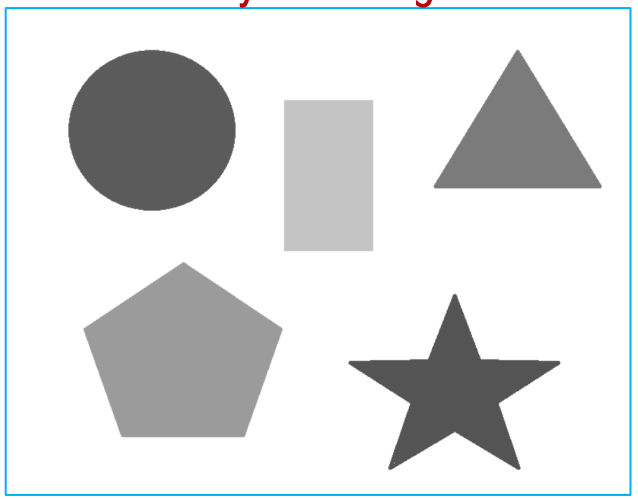
```
# Invert image
inverted_binary_img = ~ binary_img
```

Contour Detection Algorithm

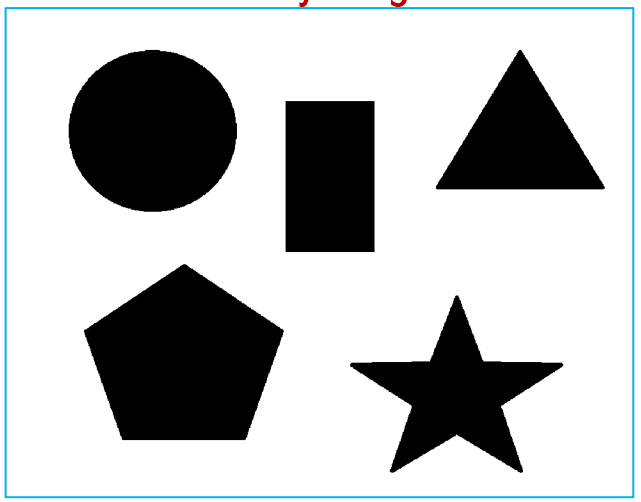
RGB image



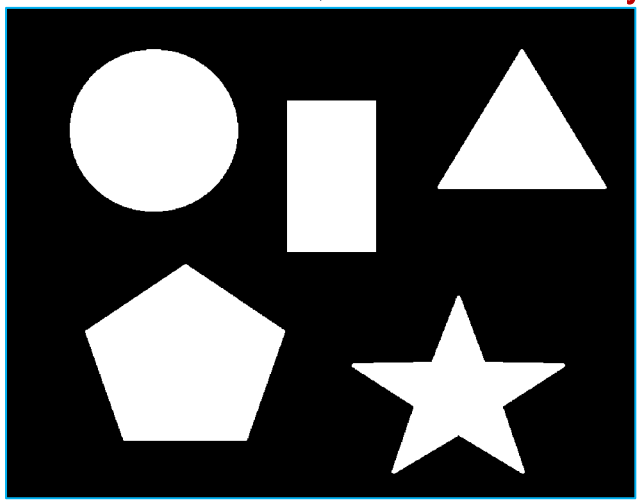
Grayscale image



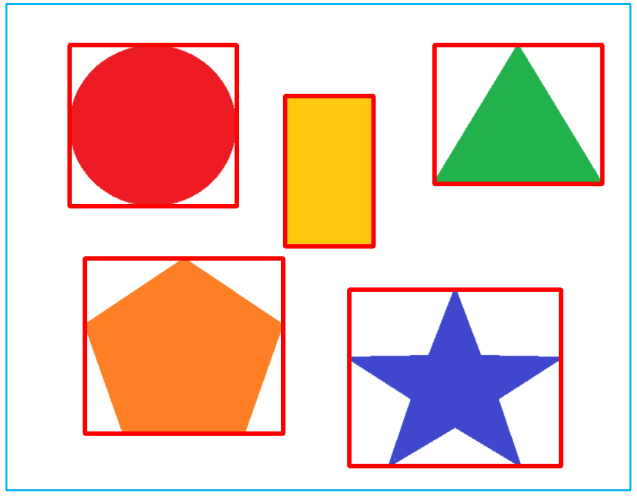
Binary Image



Inverse Binary Image



Final Contour Result



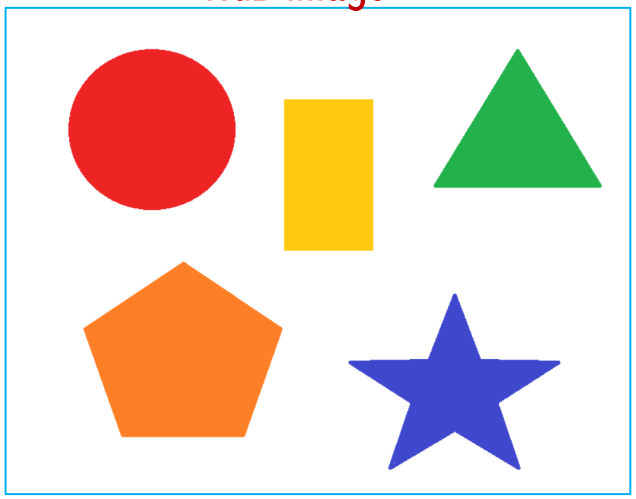
```
# Detect contours
# hierarchy variable contains information about the relationship between each contours
contours_list, hierarchy = cv2.findContours(inverted_binary_img,
                                            cv2.RETR_TREE,
                                            cv2.CHAIN_APPROX_SIMPLE) # Find contours

# Draw a bounding box around all detected contours
for c in contours_list:
    x, y, w, h = cv2.boundingRect(c)

    # Make sure contour area is large enough
    if (cv2.contourArea(c)) > 10000:
        cv2.rectangle(input_image_cpy, (x, y), (x + w, y + h), (0, 0, 255), 5)
```

Outline

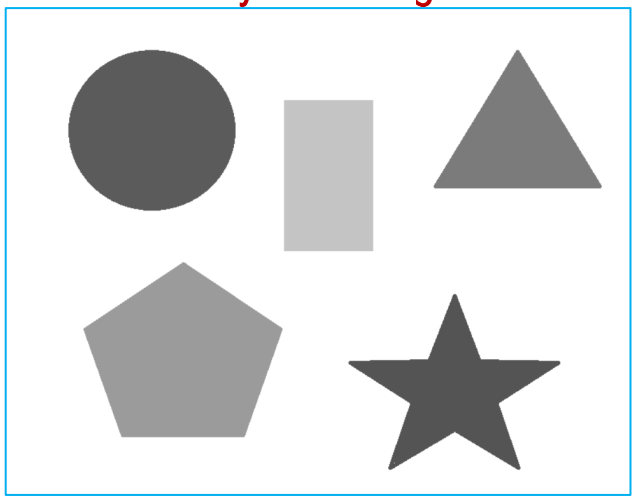
- Introduction to Computer Vision
- Edge Detection Algorithm
- Line Detection Algorithm
- Contour Detection Algorithm
- Shape Detection Algorithm
- Background Subtraction Algorithm

Shape Detection Algorithm

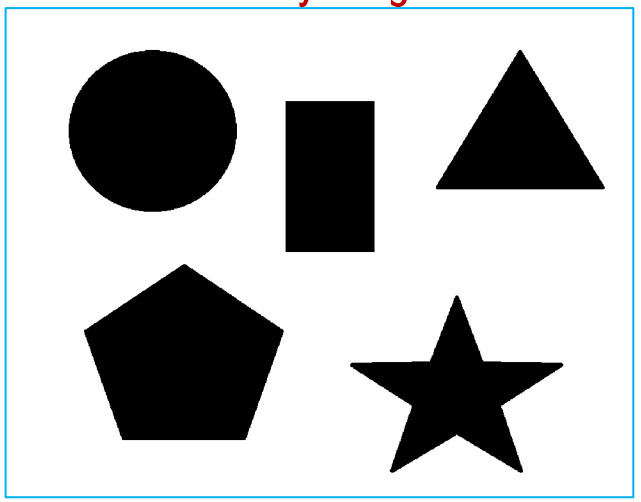
RGB image



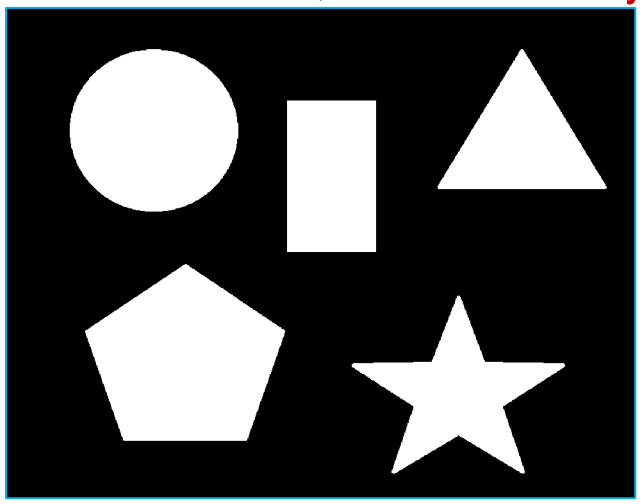
Grayscale image



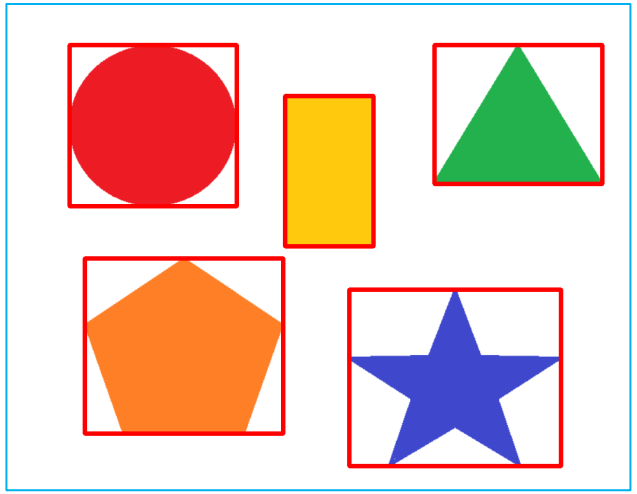
Binary Image



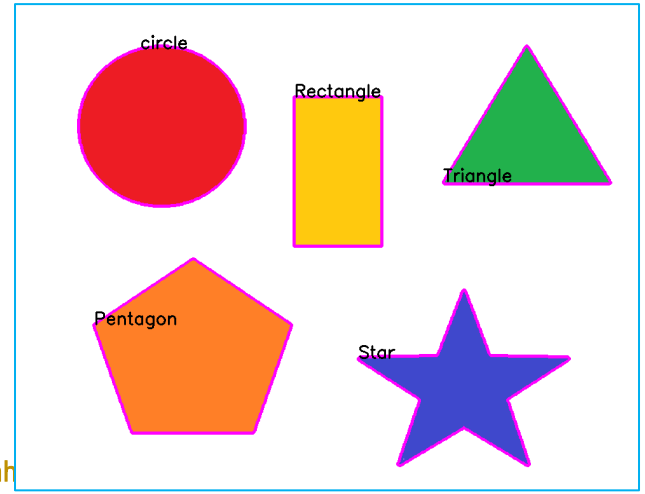
Inverse Binary Image



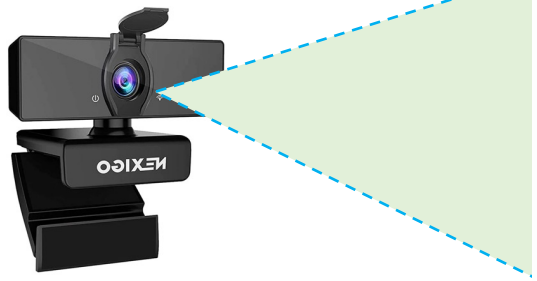
Contour Results



Shape Detection



Read and Display Image From Webcam



```
#Read and display image using camera
import cv2
def show_webcam():
    cam = cv2.VideoCapture(0)
    while True:
        ret_val, img = cam.read()
        cv2.imshow('my webcam', img)
        if cv2.waitKey(1) == ord('q'):
            break
    cam.release()
    cv2.destroyAllWindows()

show_webcam()
```

Erosion and Dilation Technique

Frame 1

Grayscale Image

Frame 2

Grayscale Image

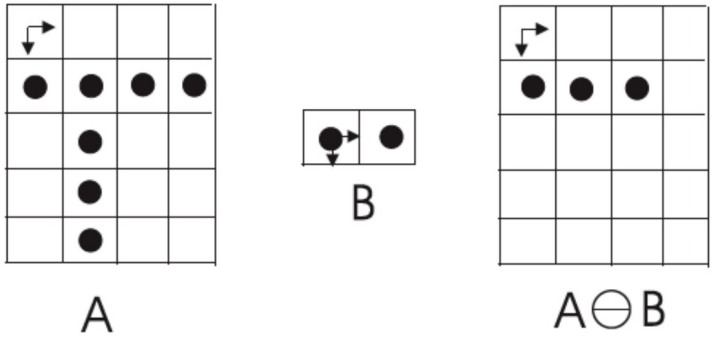
Image Difference

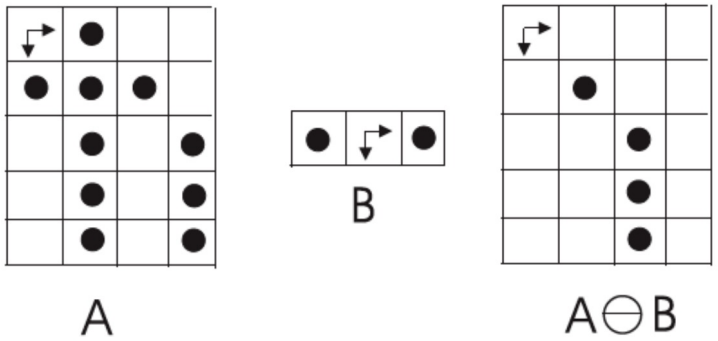
Binary image

Erosion Image

Dilation Image

Erosion Technique





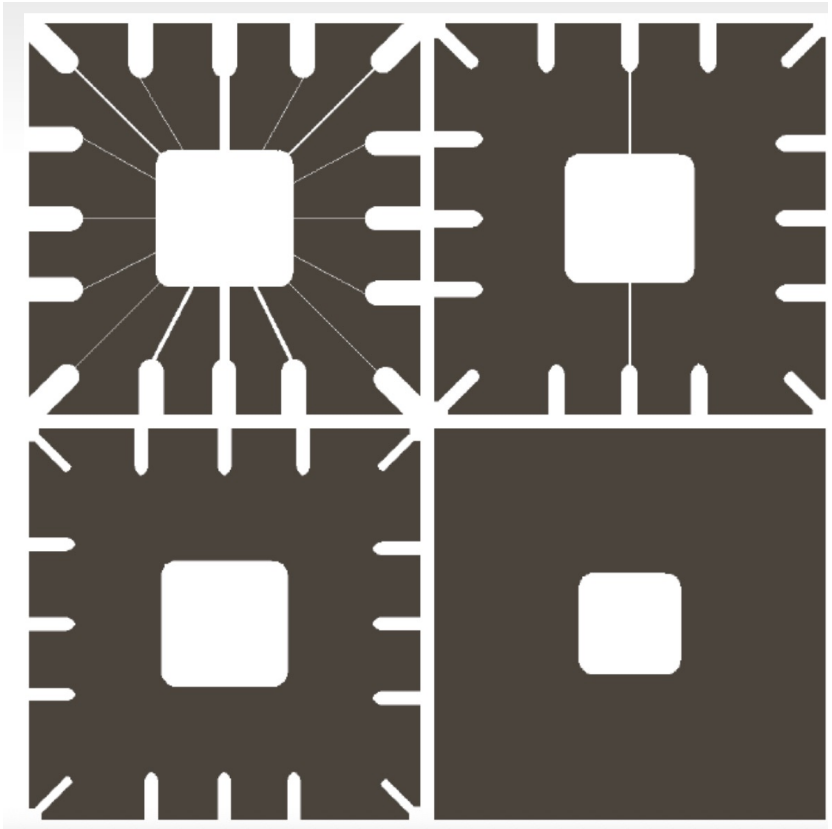
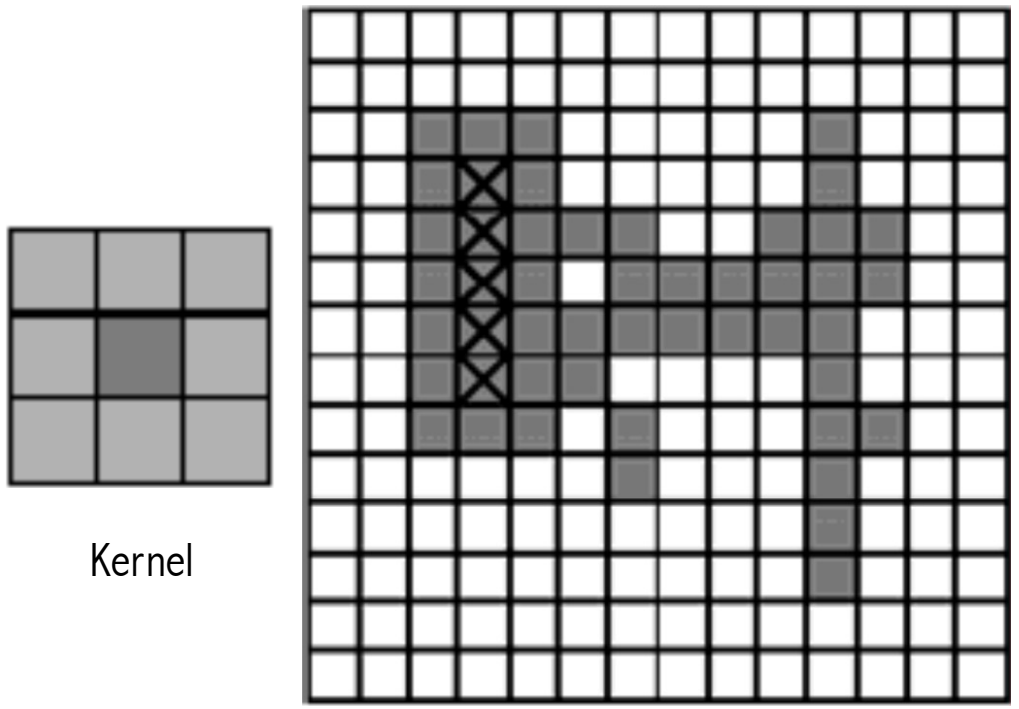
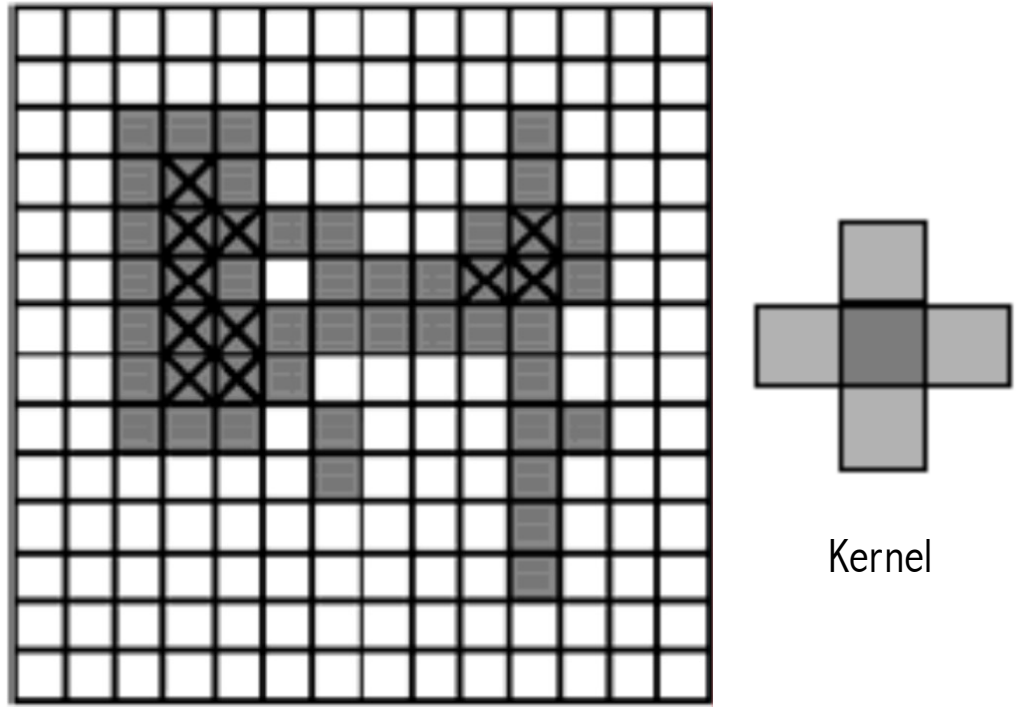


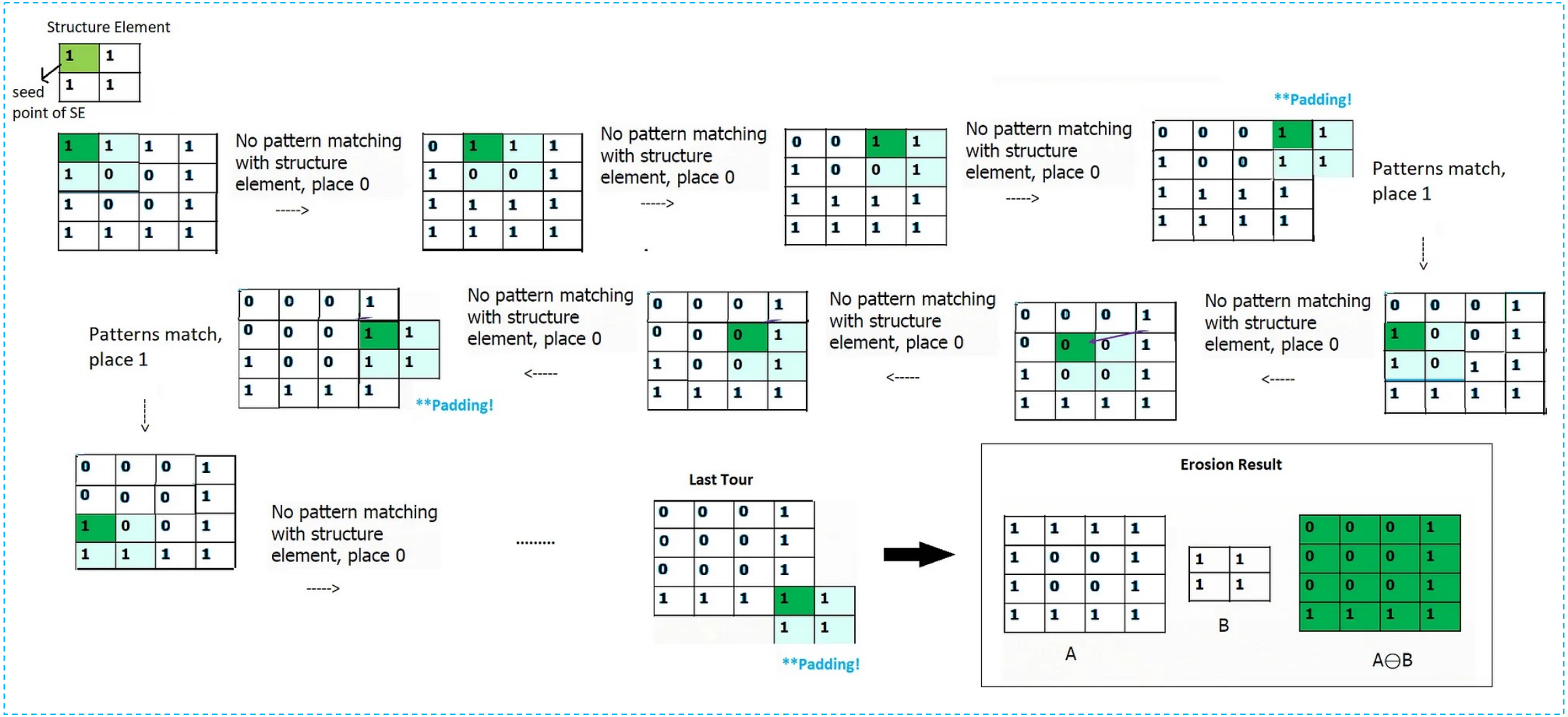
Image erode using square structuring elemements sides of 11x11, 15x15, and 45x45

Erosion Technique





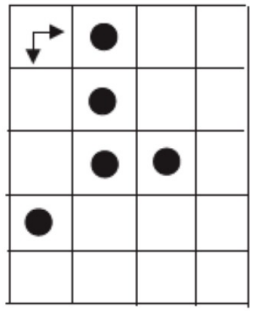
Erosion Technique



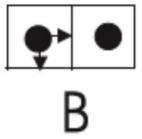
https://towardsdatascience.com/image-processing-part-3-dbf103622909

52

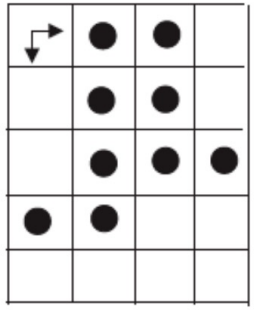
Dilation Technique



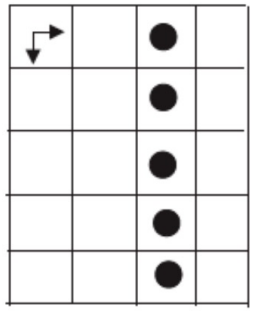
A



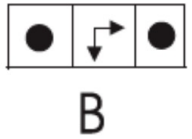
B



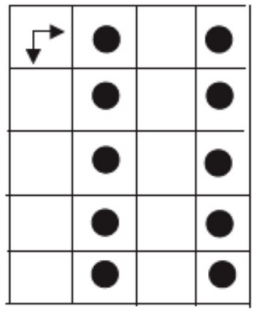
A⊕B



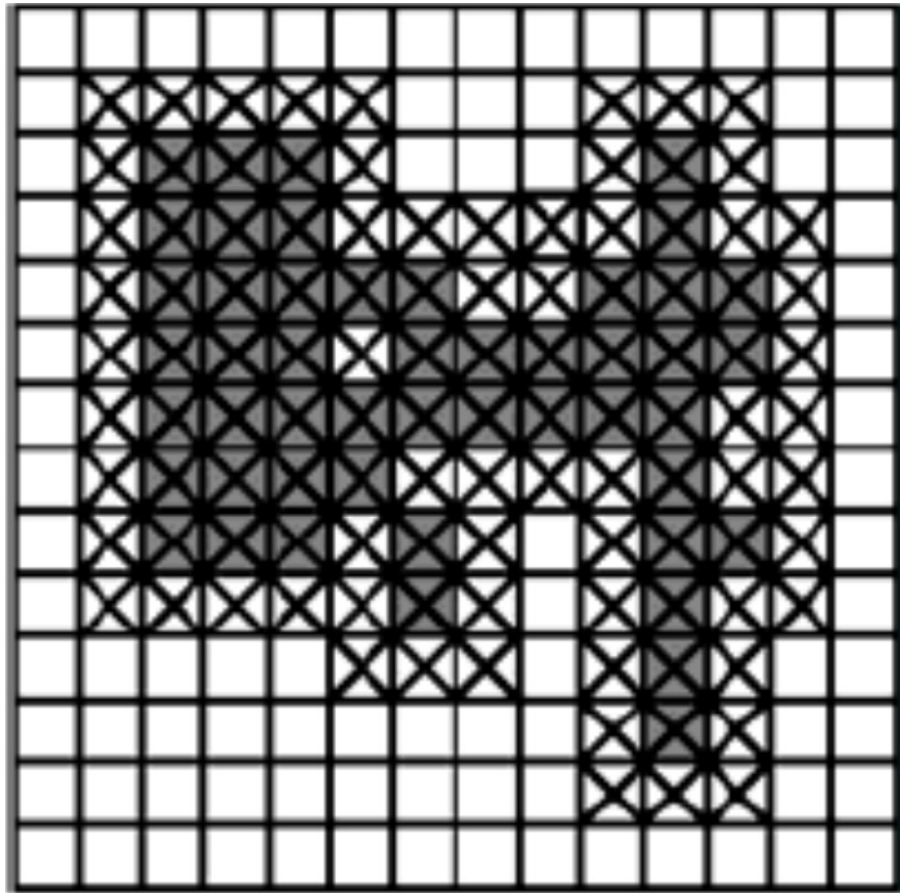
A

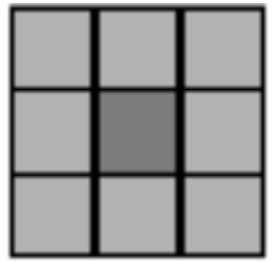


B



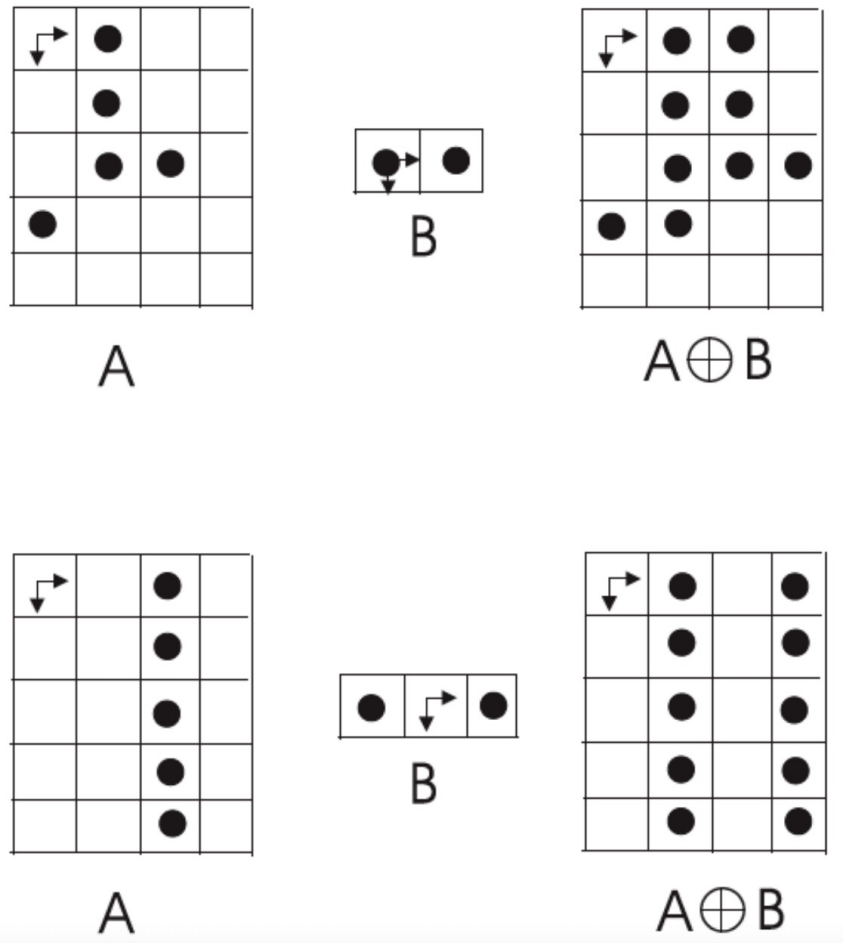
A⊕B

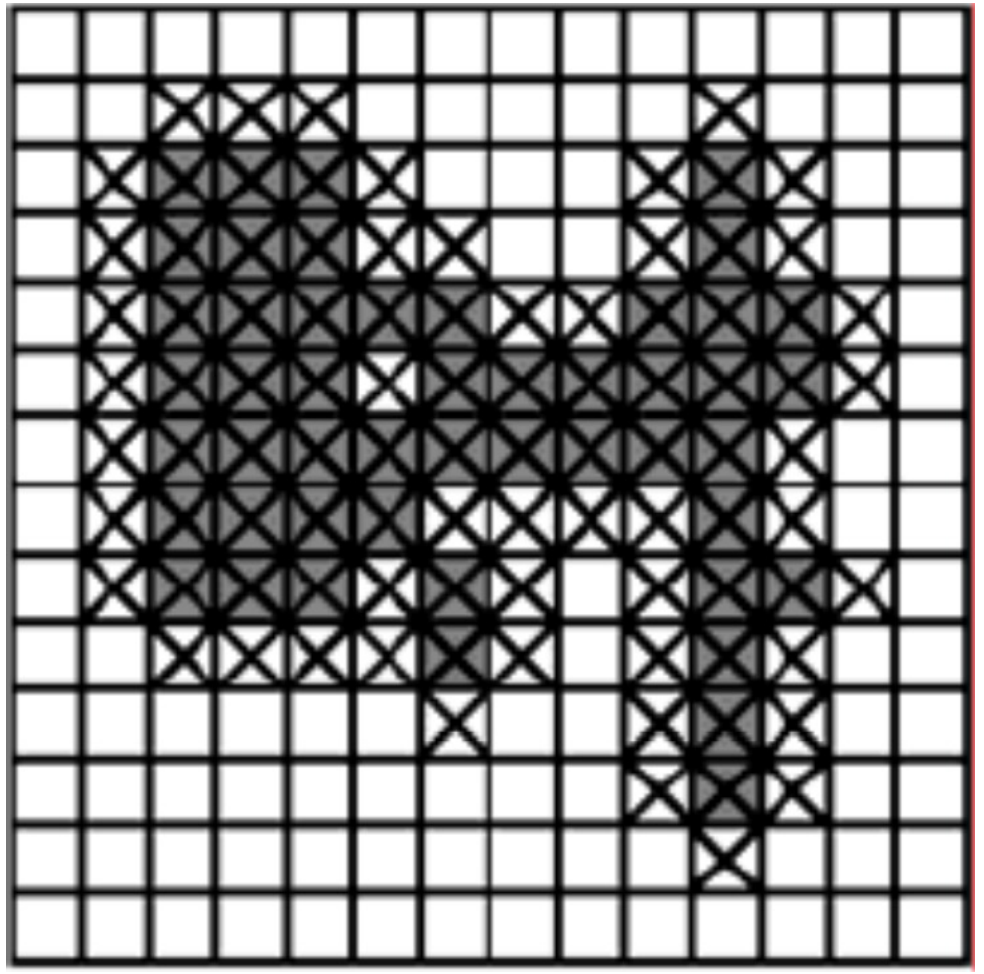




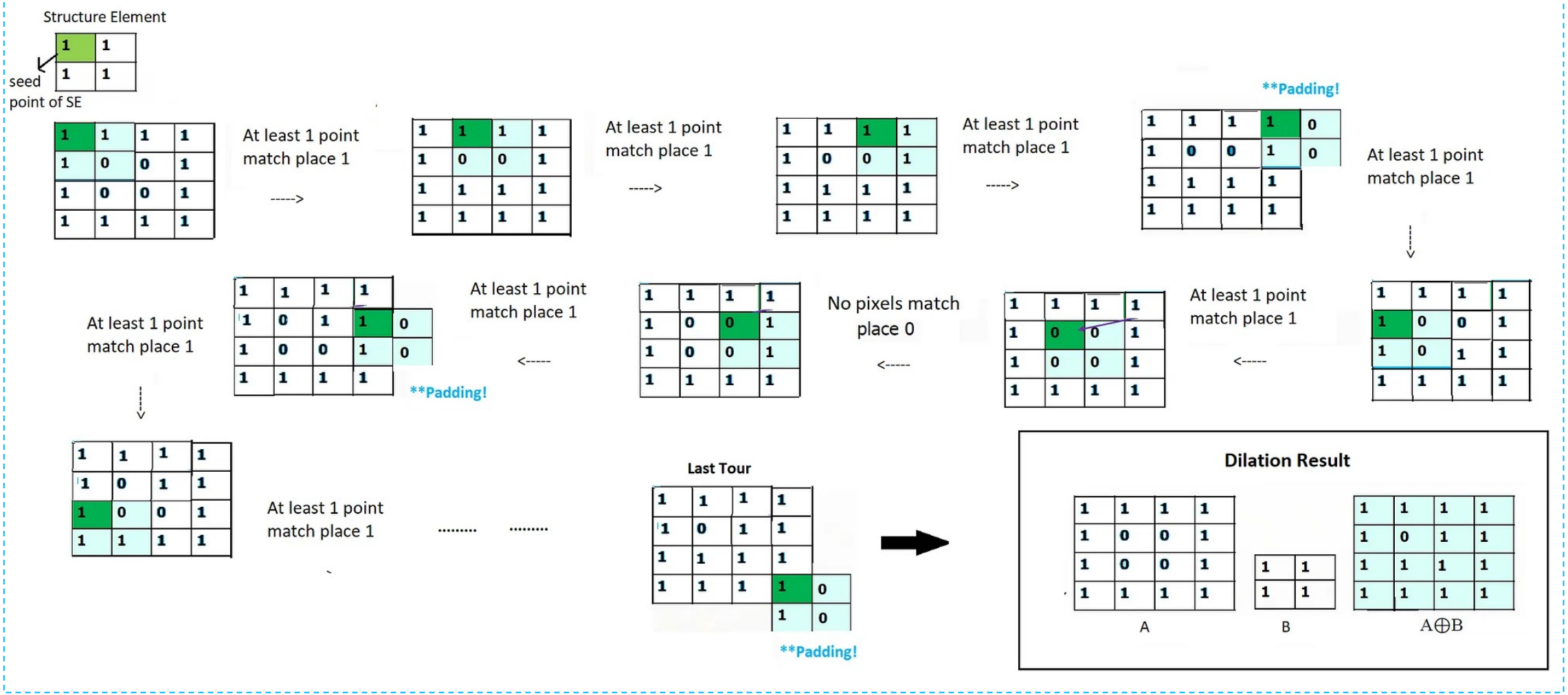
Kernel

Dilation Technique





Dilation Technique



Outline

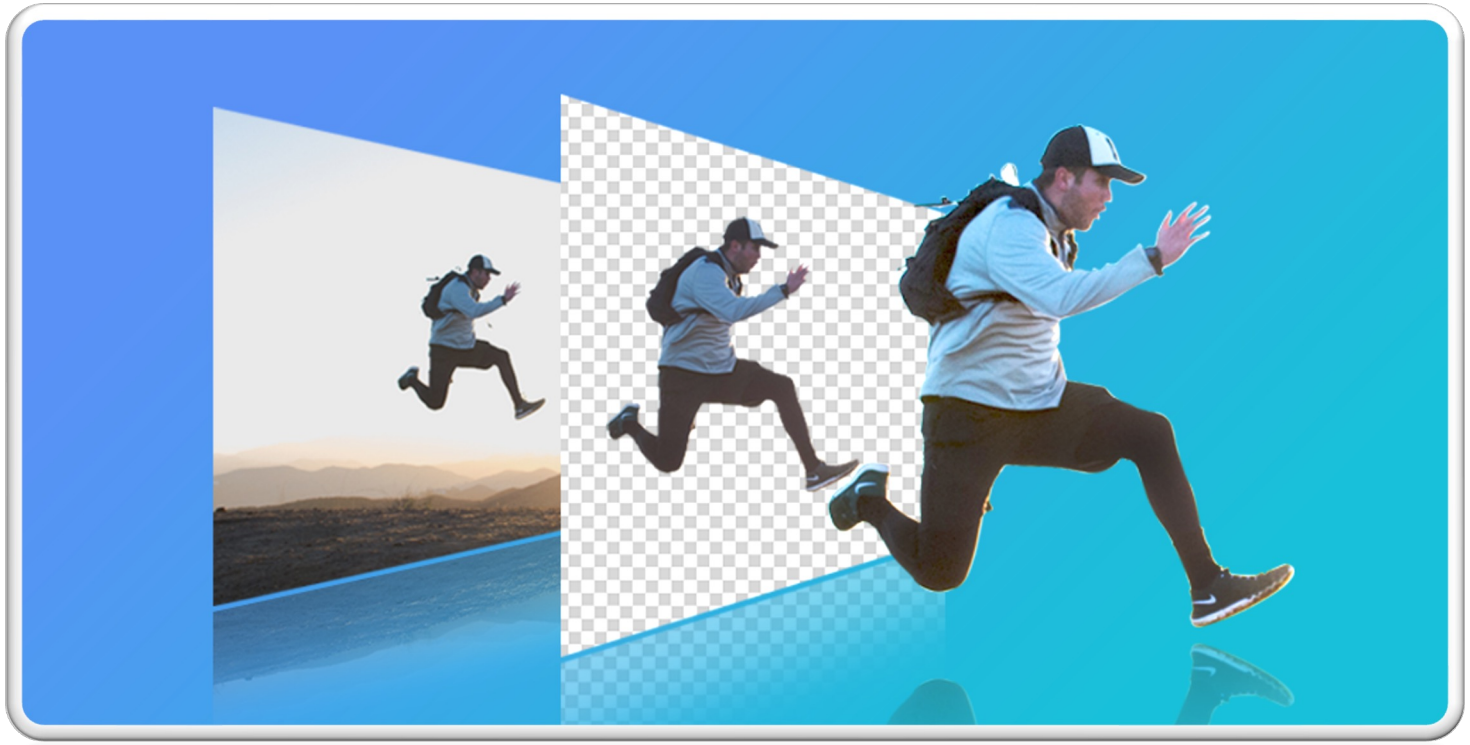
- Introduction to Computer Vision
- Edge Detection Algorithm
- Line Detection Algorithm
- Contour Detection Algorithm
- Shape Detection Algorithm
- Background Subtraction Algorithm

Background Substraction

Background subtraction is a widely used approach to detect moving objects in a sequence of frames from static cameras

Given an image (mostly likely to be a video frame), we want to identify the foreground objects in that image!

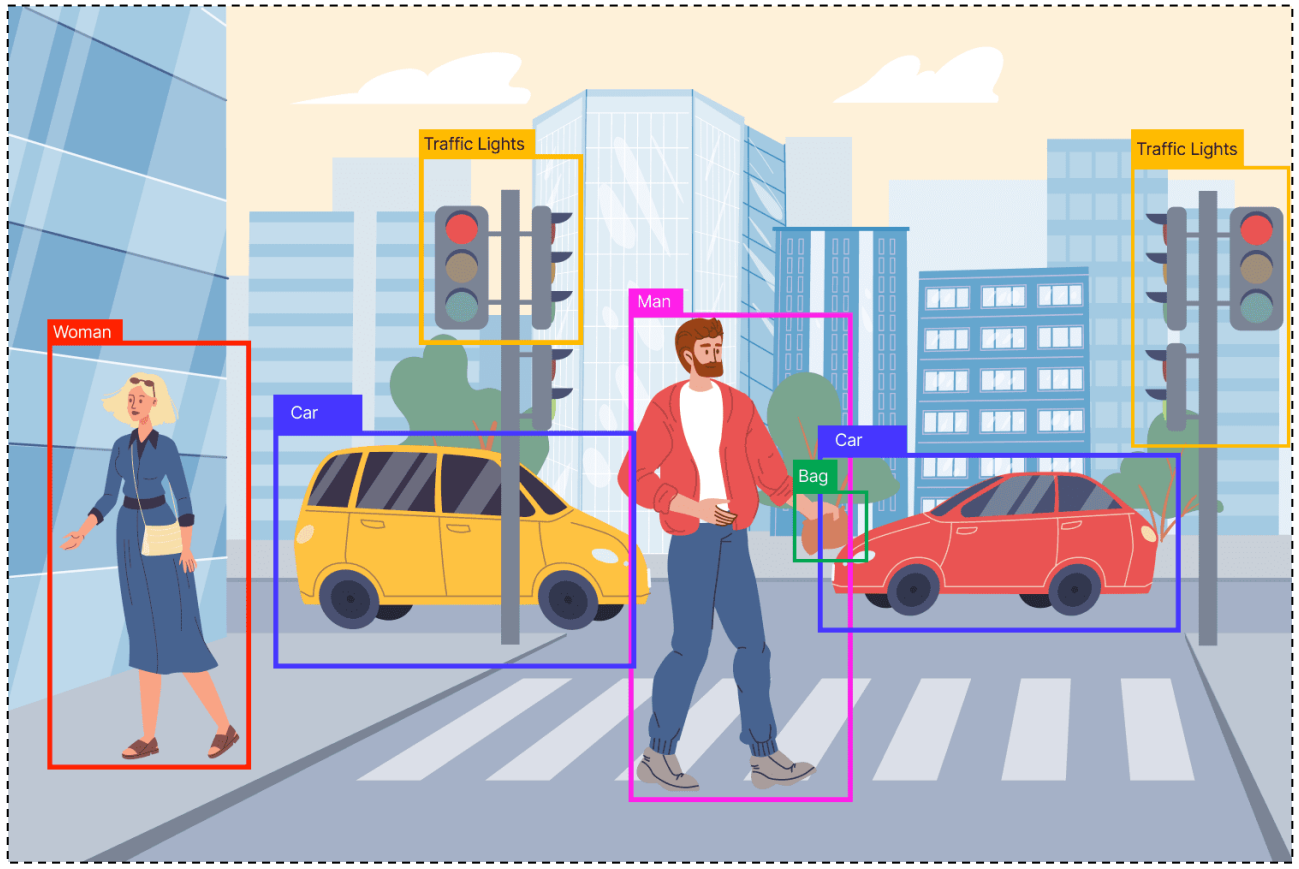
Why do we need to use Background Subtraction?



Background Substraction

Background subtraction is a widely used approach to detect moving objects in a sequence of frames from static cameras

Why do we need to use Background Subtraction?



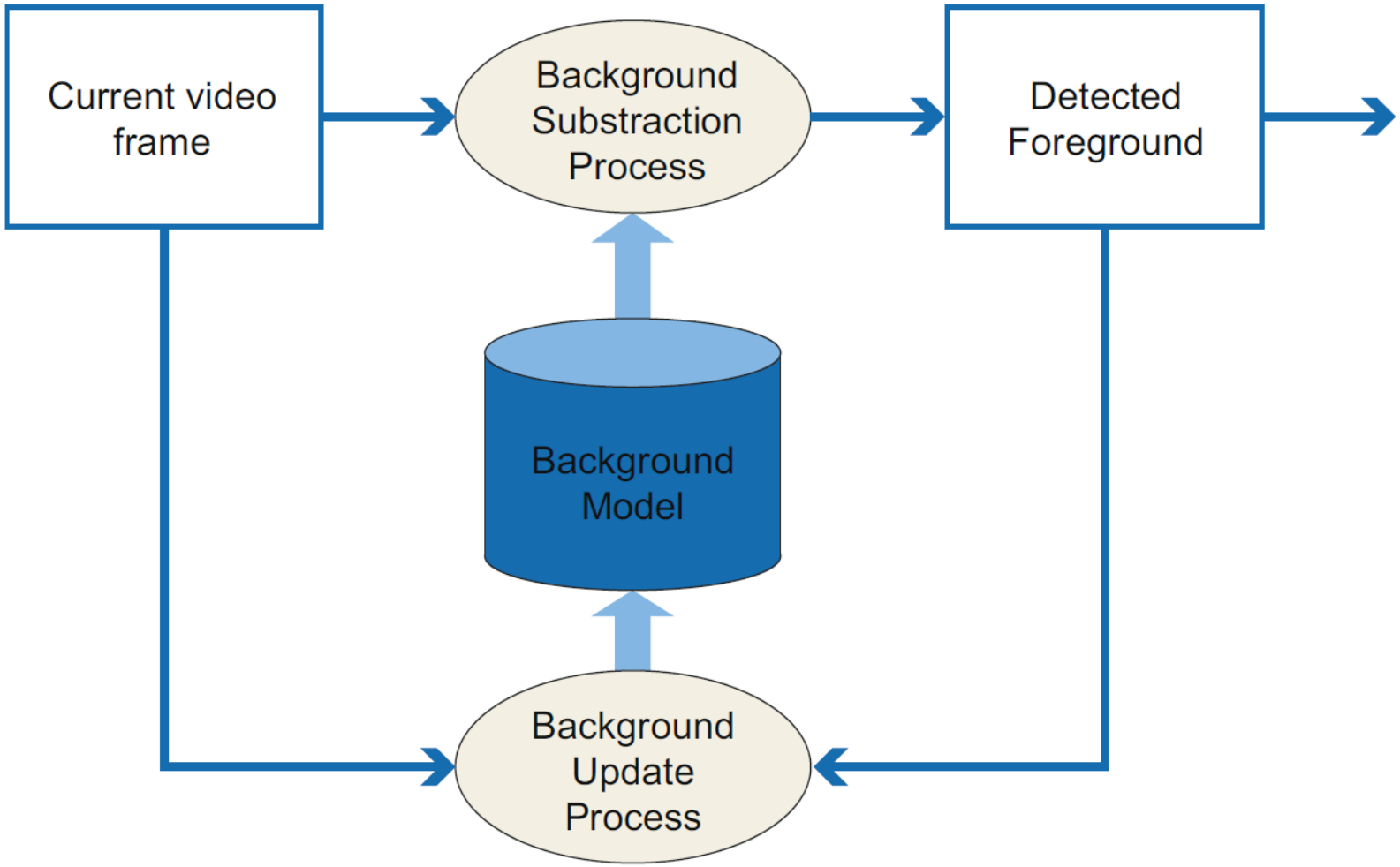
Object detection and classification

Background Substraction

Background subtraction is a widely used approach to detect moving objects in a sequence of frames from static cameras

Why do we need to use Background Subtraction?

Background Substraction



Simple Approach

1. Estimate the background for time t.
2. Subtract the estimated background from the input frame.
3. Apply a threshold T to the absolute difference to get the foreground mask.

Depending on the object structure, speed, frame rate and global threshold, this approach may or may not be useful

Time t

-

Time t + 1

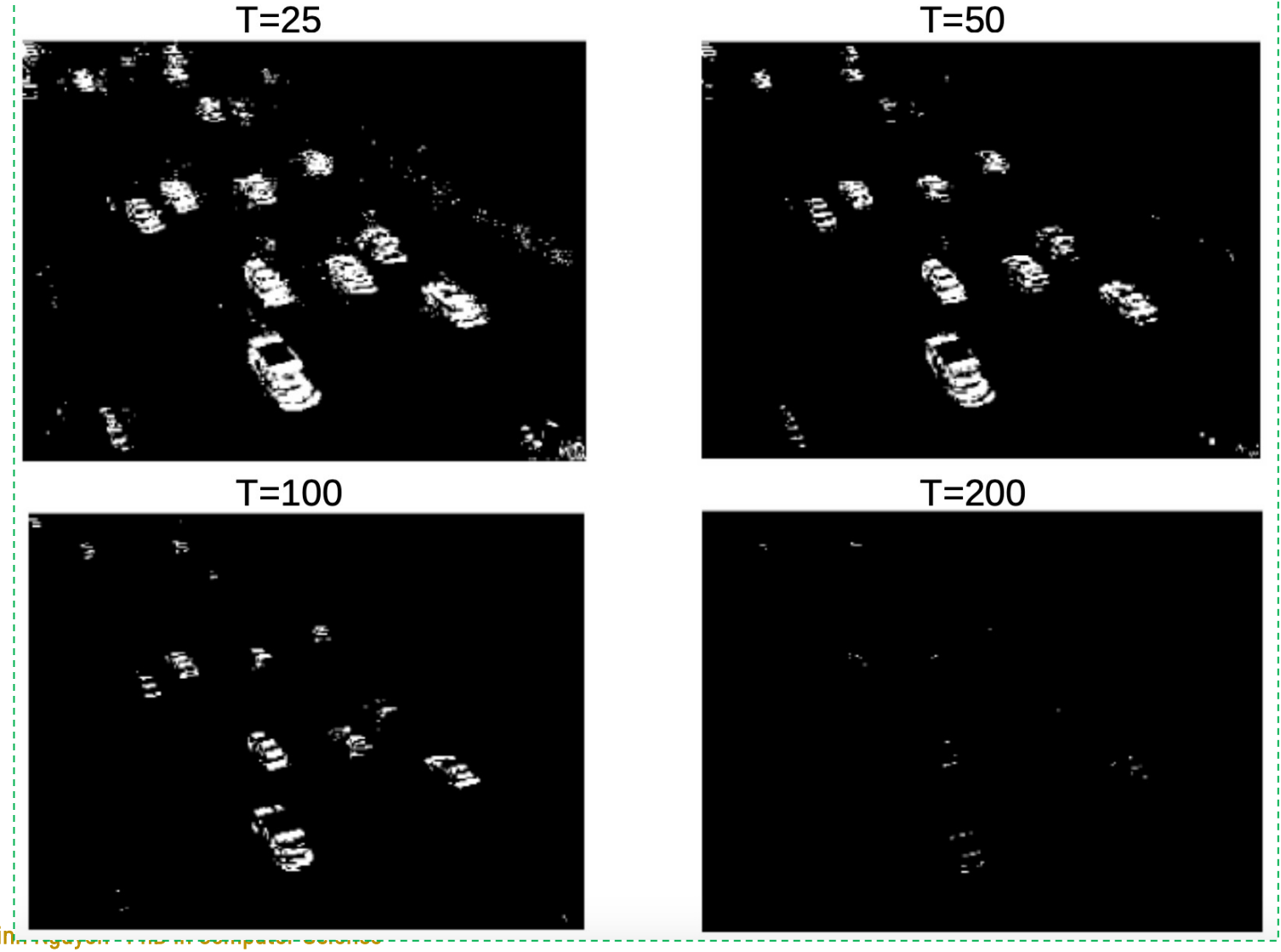
$$B(x, y, t) = I(x, y, t-1)$$

$$|I(x, y, t) - I(x, y, t-1)| > T$$

how can we estimate the background?

Prof. Birgi Tamersoy. The University of Texas at Austin

Simple Approach



$$B(x, y, t) = I(x, y, t - 1)$$

$$|I(x, y, t) - I(x, y, t - 1)| > T$$

Depending on the object structure, speed, frame rate and global threshold, this approach may or may not be useful

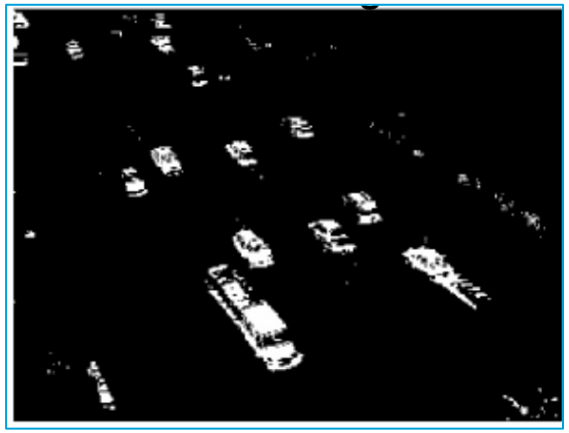
Mean Filter

The background is the mean of the previous n frames

$$B(x,y,t) = \frac{1}{n}\sum_{i=0}^{n-1} I(x,y,t-i)$$

$$\left| I(x,y,t) - \frac{1}{n}\sum_{i=0}^{n-1} I(x,y,t-i) \right| > T$$

n = 10



n = 20

n = 50

Median Filter

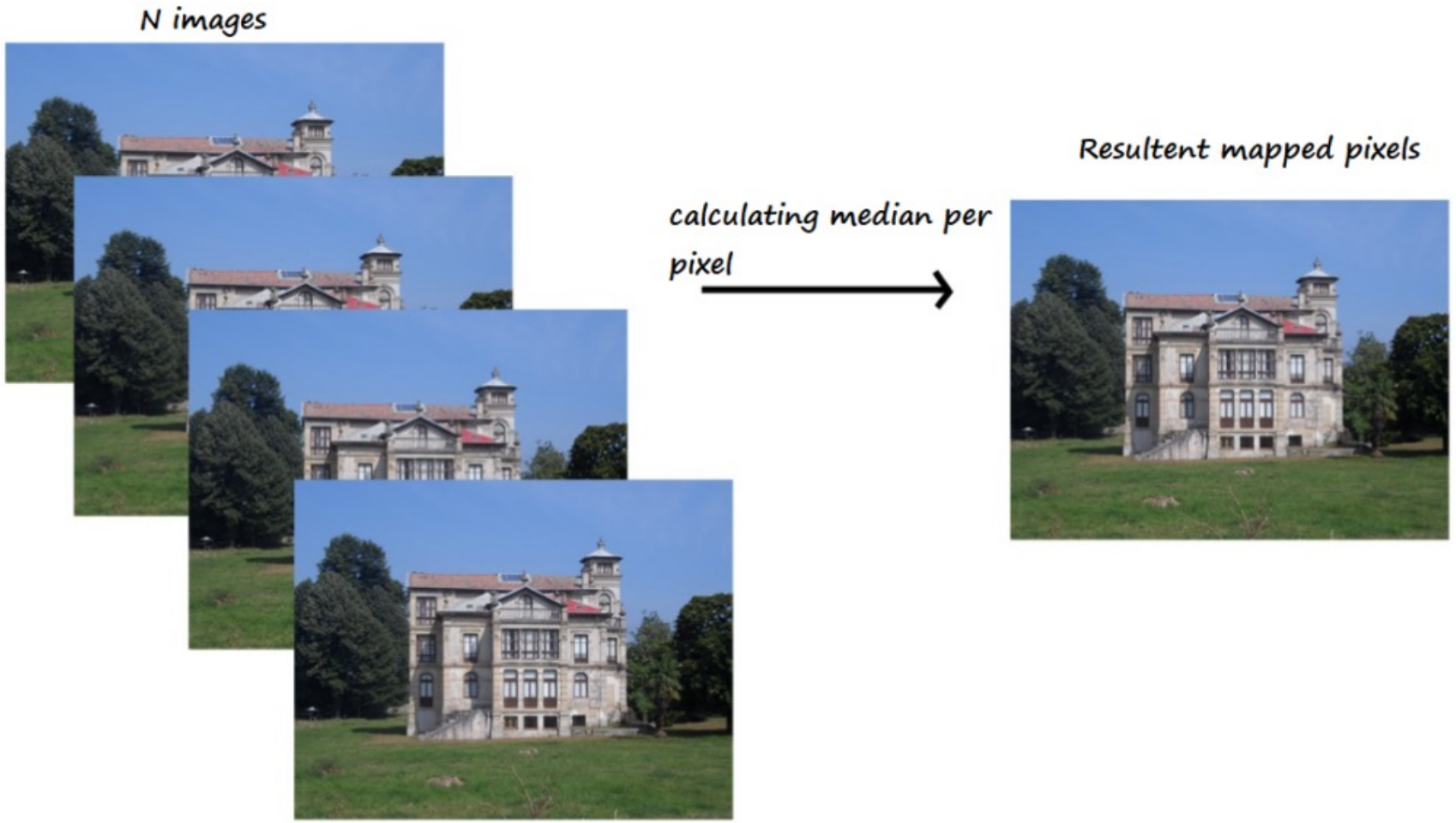
Assuming that the background is more likely to appear in a scene, we can use the median of the previous n frames as the background model

$$B(x,y,t) = \underset{i=0\ldots n-1}{\text{median}} \left(I(x,y,t-i)\right)$$

$$\left|I(x,y,t) - \underset{i=0\ldots n-1}{\text{median}} \left(I(x,y,t-i)\right)\right| > T$$

	n = 10	n = 20	n = 50
Mean Filter			
Median Filter			

Temporal Average Filter



Dicussion

Advantages:

– Extremely easy to implement and use!
– All pretty fast.
– Corresponding background models are not constant, they change over time

Disadvantages:

– Accuracy of frame differencing depends on object speed and frame rate!
– Mean and median background models have relatively high memory requirements.
– There is one global threshold, Th, for all pixels in the image

Quick Look at K-mean

Advantages:

1. Select initial choice of K means
2. Determine which sample belongs to which cluster

$$\Delta_{nk} = 1 \text{ if } k = \text{argmin } \|x_n - \mu_k\|^2$$

$$\Delta_{nk} = 0 \text{ otherwise}$$

3. Define cost function

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} \Delta_{nk} (x_n - \mu_k)^2$$

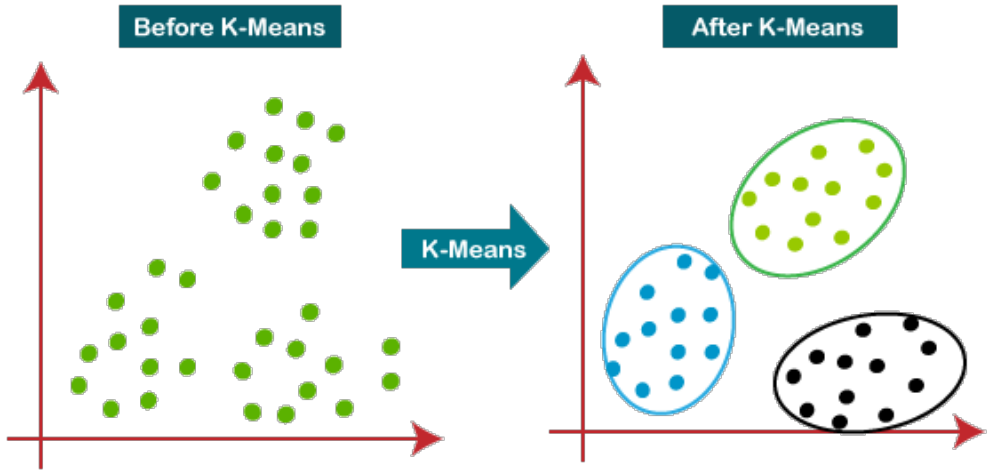
4. Minimize the cost function based with respective to:

$$\frac{\partial j}{\mu_k} = -2 \sum_{n=1}^{N} \Delta_{nk}(x_n - \mu_k) = -2 \sum_{n=1}^{N} \Delta_{nk}(x_n - \mu_k) = 0$$

$$\sum_{n=1}^{N} \Delta_{nk}\ x_n = \sum_{n=1}^{N} \Delta_{nk}\ \mu_k$$

$$\mu_k = \frac{\sum_{n=1}^{N} \Delta_{nk}\ x_n}{\sum_{n=1}^{N} \Delta_{nk}}$$

5. Go to step 2
6. Do it until no further changes in assignment or until reach to maximum number of iterations



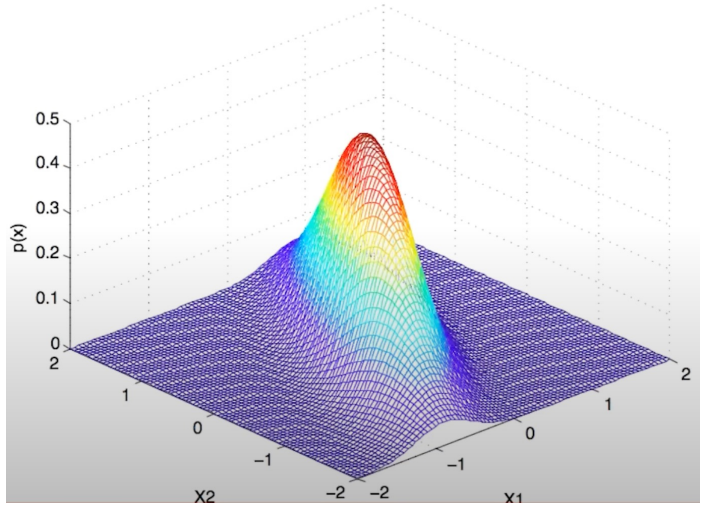
Quick look at Gaussian Mixture Model

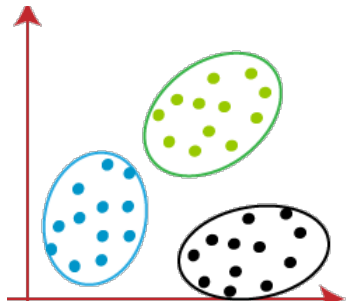
Multivariate Gaussian distribution for $\mathbf{x} \in \mathrm{R}^d$:

$$p(\mathbf{x}|\mu,\Sigma) = (2\pi)^{-\frac{d}{2}} |\Sigma|^{-\frac{1}{2}} e^{-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)}$$

- μ is vector of means
- Σ is covariance matrix

pdf when $\mu = [0,0]$ and $\Sigma = \begin{bmatrix} 0.9 & 0.4 \\ 0.4 & 0.3 \end{bmatrix}$





Normal Distribution Formula

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

$\mu =$ mean of x
$\sigma =$ standard deviation of x
$\pi \approx 3.14159 \ldots$
$e \approx 2.71828 \ldots$

Mixture model:

- observation $\mathbf{x}_i$ in cluster c_i with K clusters
- model each cluster with a Gaussian distribution

$$\mathbf{x}_i \,|\, c_i = k \sim N(\mu_k, \Sigma_k)$$

How do we find $c_1, \ldots, c_n$ (clusters) and $(\mu_1, \Sigma_1), \ldots, (\mu_K, \Sigma_K)$ (cluster centers)?

Quick look at Gaussian Mixture Model

First, let's simplify the model:

- covariance matrices have only diagonal elements,

$$\Sigma = \begin{bmatrix} \sigma_1^2 & 0 & \dots & 0 \\ 0 & \sigma_2^2 & \dots & 0 \\ \dots & \dots & \dots & 0 \\ 0 & 0 & 0 & \sigma_K^2 \end{bmatrix}$$

- set $\sigma_1^2 = \dots = \sigma_K^2$, suppose known

Next, use a method similar to K-means:

- start with random cluster centers
- associate observations to clusters by (log-)likelihood,

$$\ell(\mathbf{x}_i \,|\, c_i = k) = -\frac{d}{2}\log(2\pi) - \frac{1}{2}\log\left(\prod_{j=1}^{d}\sigma_{k,j}^2\right) - \frac{1}{2}\sum_{j=1}^{d}(x_{i,j} - \mu_{k,j})^2/\sigma_{k,j}^2$$

$$\propto -d\log(\sigma_k) - \frac{1}{2\sigma_k^2}\sum_{j=1}^{d}(x_{i,j} - \mu_{k,j})^2$$

$$\propto -\sum_{j=1}^{d}(x_{i,j} - \mu_{k,j})^2$$

- refit centers $\mu_1, \dots, \mu_K$ given clusters by

$$\mu_{k,j} = \frac{1}{n_k}\sum_{c_i = k} x_{i,j}$$

- recluster observations...

Quick look at Gaussian Mixture Model

clustering with K-means

minimize distance

$$d(\mathbf{x}_i, \mu_k) = \sqrt{\sum_{j=1}^{d} (x_{i,j} - \mu_{k,j})^2}$$

clustering with GMM

maximize likelihood

$$\ell(\mathbf{x}_i \mid c_i = k) \propto -\sum_{j=1}^{d} (x_{i,j} - \mu_{k,j})^2$$

update means with K-means

use average

$$\mu_{k,j} = \frac{1}{n_k} \sum_{c_i = k} x_{i,j}$$

update means with GMM

use average

$$\mu_{k,j} = \frac{1}{n_k} \sum_{c_i = k} x_{i,j}$$

Quick look at Gaussian Mixture Model

OK, now what if

$$\Sigma = \begin{bmatrix} \sigma_1^2 & 0 & \dots & 0 \\ 0 & \sigma_2^2 & \dots & 0 \\ \dots & \dots & \dots & 0 \\ 0 & 0 & 0 & \sigma_K^2 \end{bmatrix}$$

and $\sigma_1^2, \dots, \sigma_K^2$ can take different values?

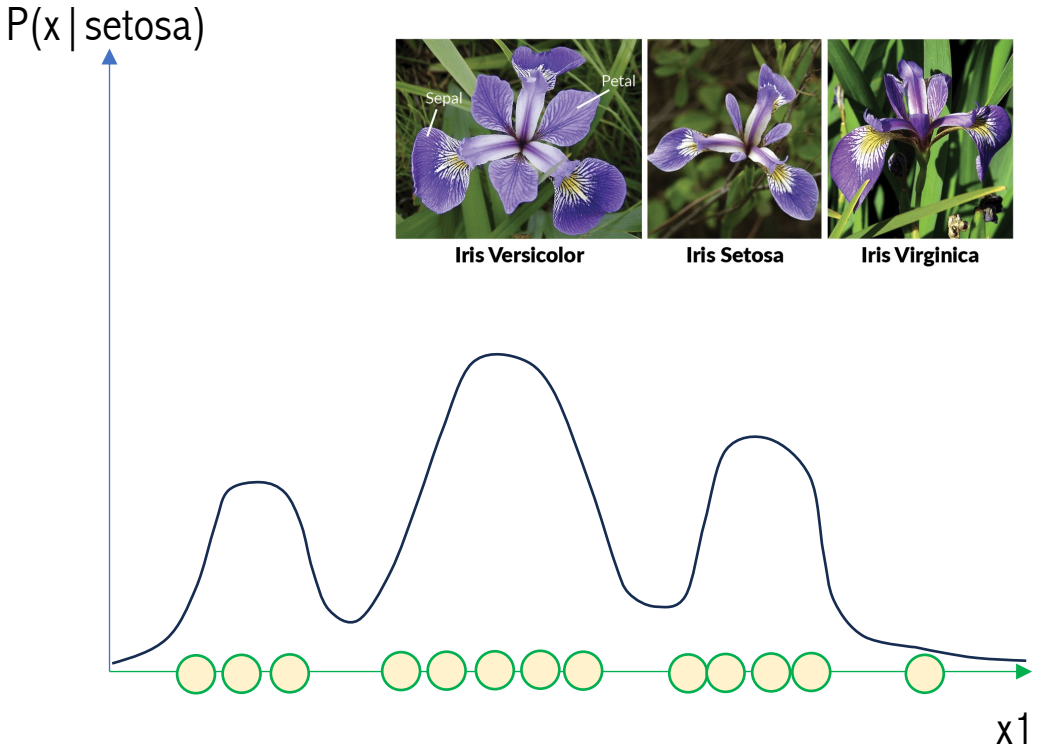
- use same algorithm
- update μ_k and σ_k^2 with maximum likelihood estimator,

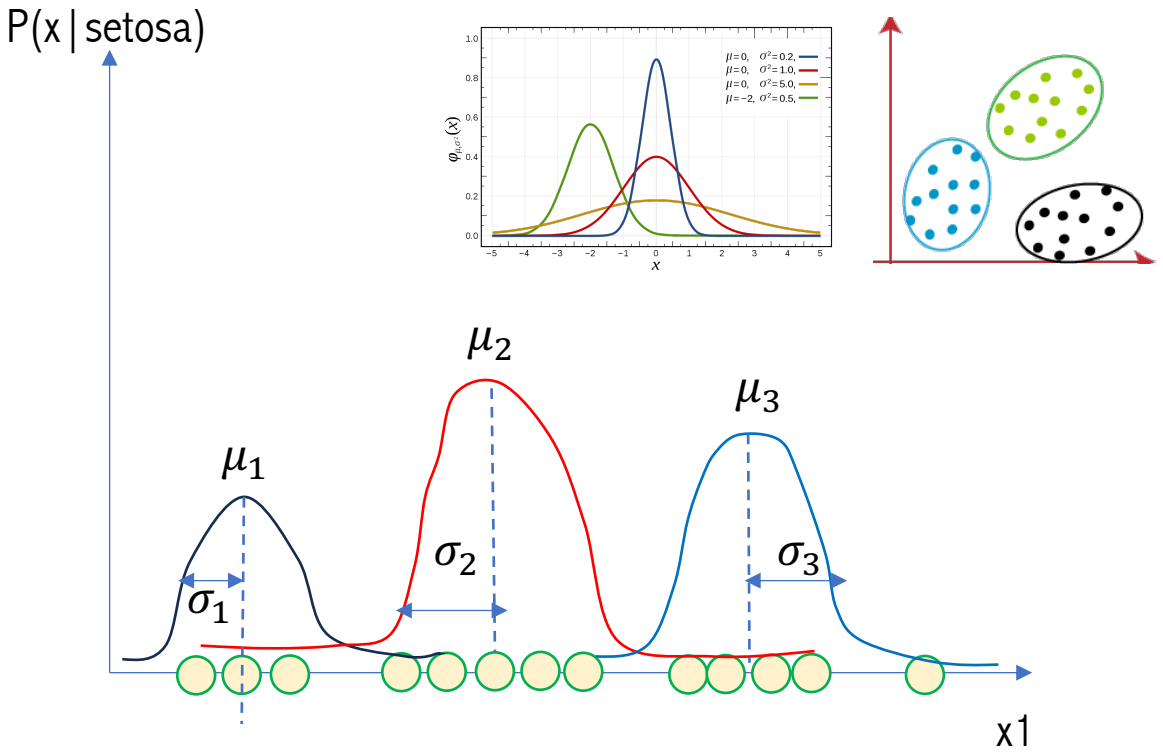
$$\mu_{k,j} = \frac{1}{n_k} \sum_{c_i = k} x_{i,j}$$

$$\sigma_{k,j}^2 = \frac{1}{n_k} \sum_{c_i = k} (x_{i,j} - \mu_{k,j})^2$$

- Clustering helps discover patterns
- *k*-means is a simple approach
- Gaussian mixture models more probabilistic foundation

Quick look at Gaussian Mixture Model





Covariance Matrix

$$P(x \mid \text{setosa}) = \sum_{k=1}^{K} \pi_k N(X \mid \mu_k, \Sigma_k)$$

$$0 < \pi_k < 1 \qquad \sum_{k=1}^{k} \pi_k = 1$$

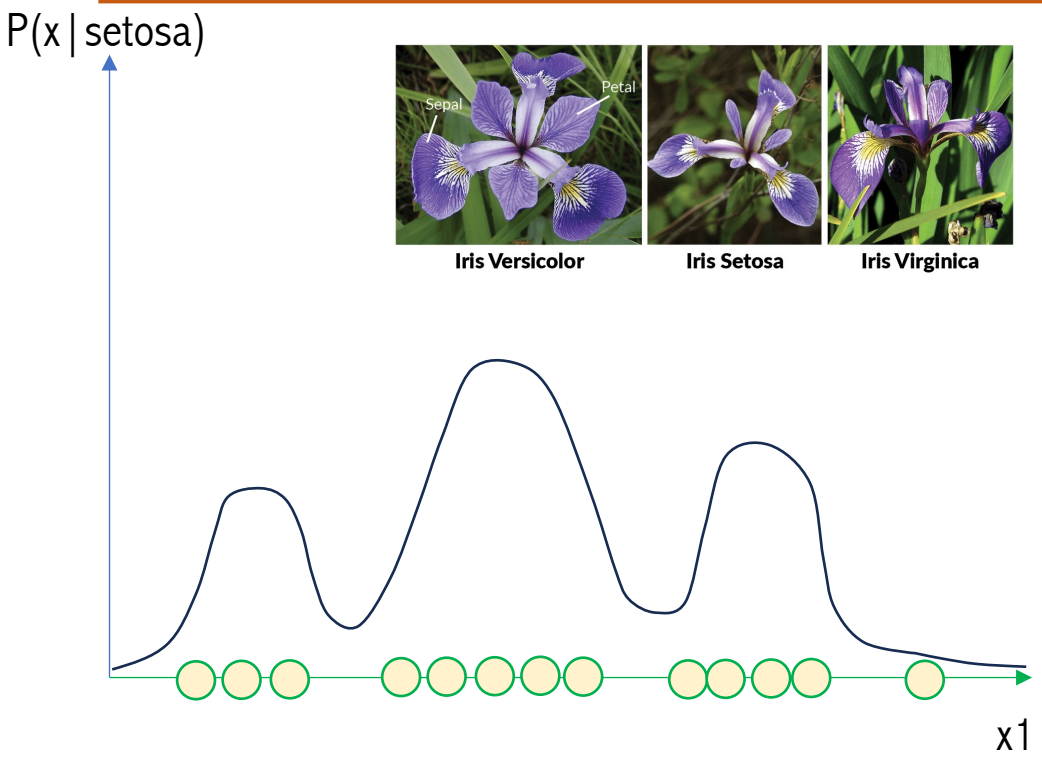
Latent variable (binary): z_k

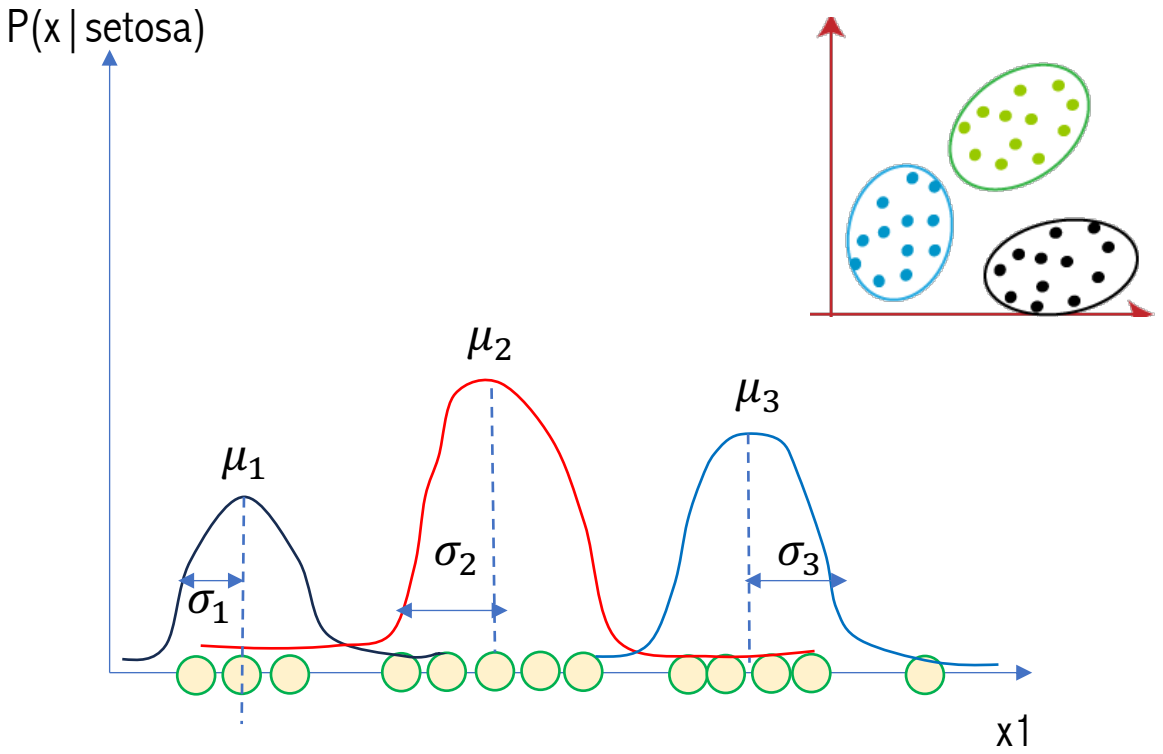
$$P(z_k=1) = \pi_k \qquad P(z_1=1) = \pi_1$$

$$P(z_2=1) = \pi_2$$

$$P(z_3=1) = \pi_3$$

Quick look at Gaussian Mixture Model





$$P(x|\ z_k=1) = N(X|\mu_k, \sum_k\ )$$

$$P(x|\ z_1=1) = N(X|\mu_k, \sum_1\ )$$

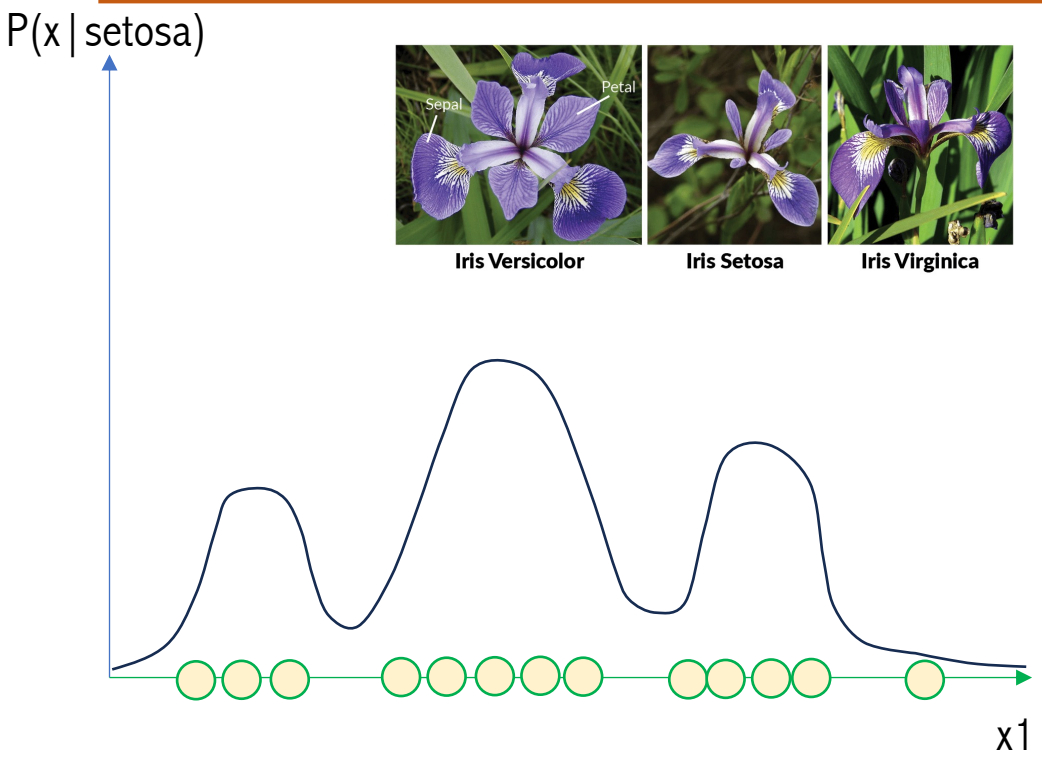
$$P(x) = \sum_{k=1}^{k} P(z_k=1)P(x|\ z_k=1)$$

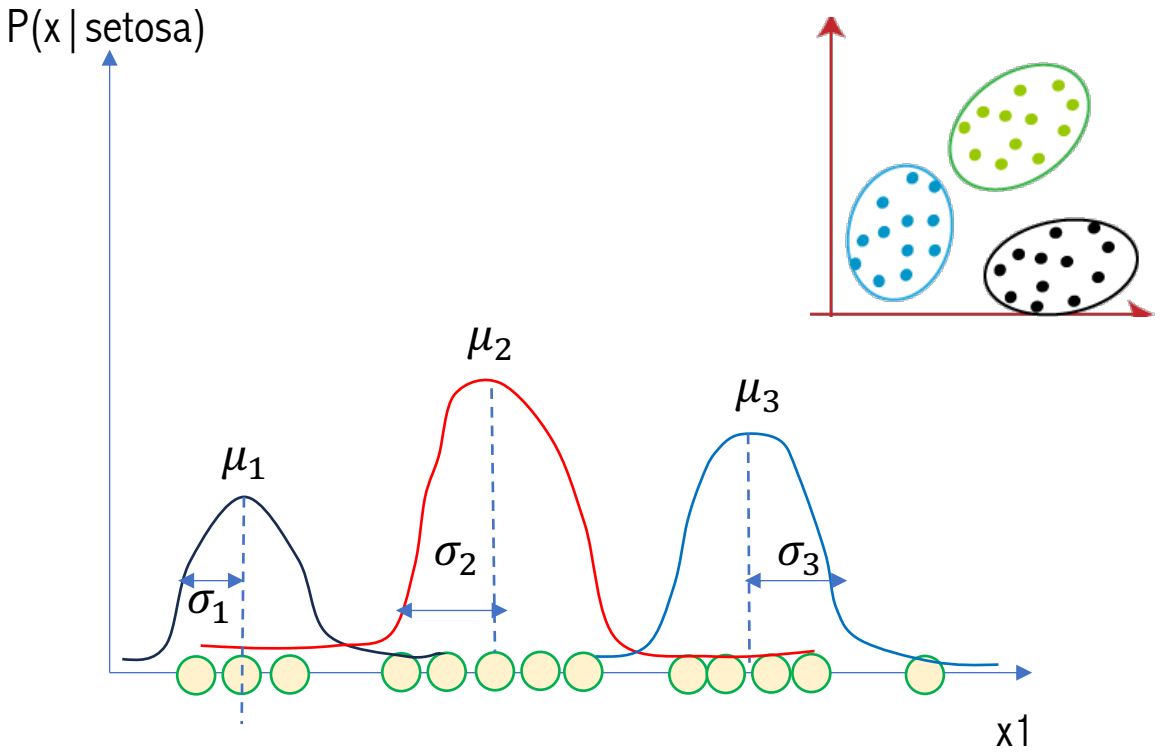
$$P(z_k=1|x) = \frac{P(x|\ z_k=1)P(z_k=1)}{P(X)}$$

$$P(X) = \prod_{i=1}^{N} p(x_i)$$ For all samples

$$\ln(P(X)) = \prod_{i=1}^{N} \ln(p(x_i)) \Rightarrow \ln(P(X)) = \prod_{i=1}^{N} \ln(\sum_{k=1}^{K} \pi_k N(X|\mu_k, \sum_k\ ))$$

Quick look at Gaussian Mixture Model





$$\ln(P(X)) = \prod_{i=1}^{N} \ln(\sum_{k=1}^{K} \pi_k N(X|\mu_k, \Sigma_k))$$

$$\ln(P(X|\ \pi, \mu, \Sigma\ )) = \prod_{i=1}^{N} \ln(\sum_{k=1}^{K} \pi_k N(X|\mu_k, \Sigma_k))$$

$$\frac{\partial \ln(P(X|\ \pi, \mu, \Sigma\ )}{\partial \mu_k} = 0$$

$$\frac{\partial \ln(P(X|\ \pi, \mu, \Sigma\ )}{\partial \Sigma_k} = 0$$

Background Subtraction With Mixture of Gaussian (MOG)

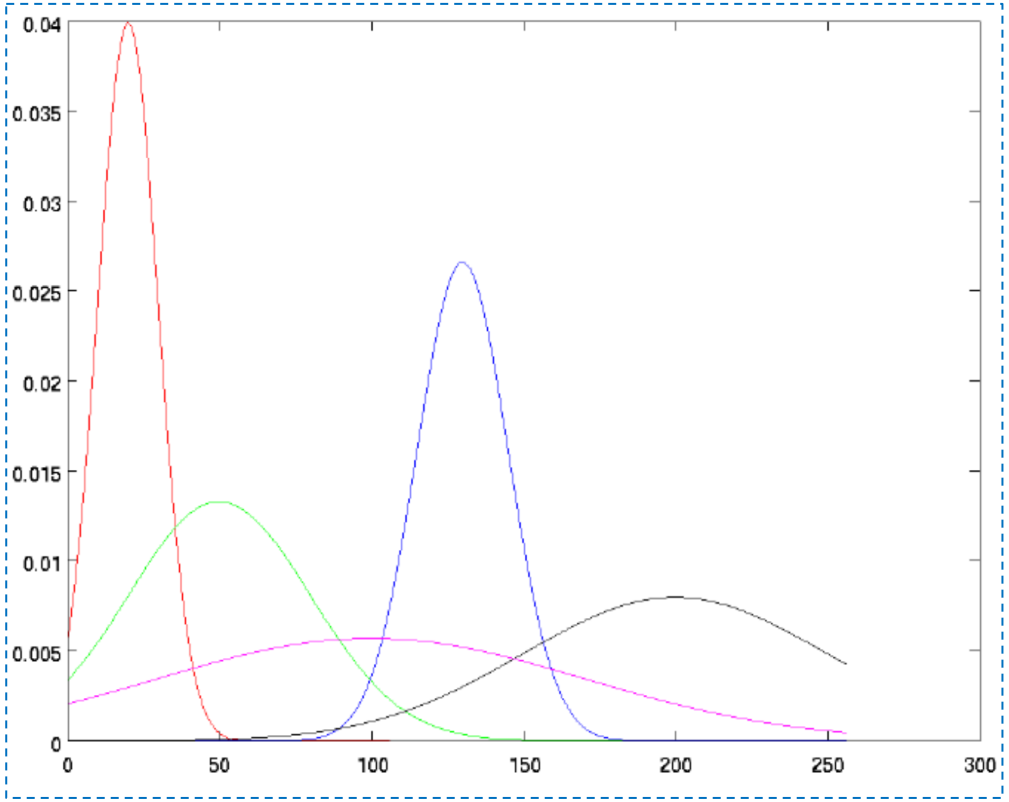
At any time t, what is known about a particular pixel (x0 , y0) is its history:

$$\{X_1, \ldots, X_t\} = \{I(x_0, y_0, i) | 1 \leq i \leq t\}$$

This history is modeled by a mixture of K Gaussian distributions:

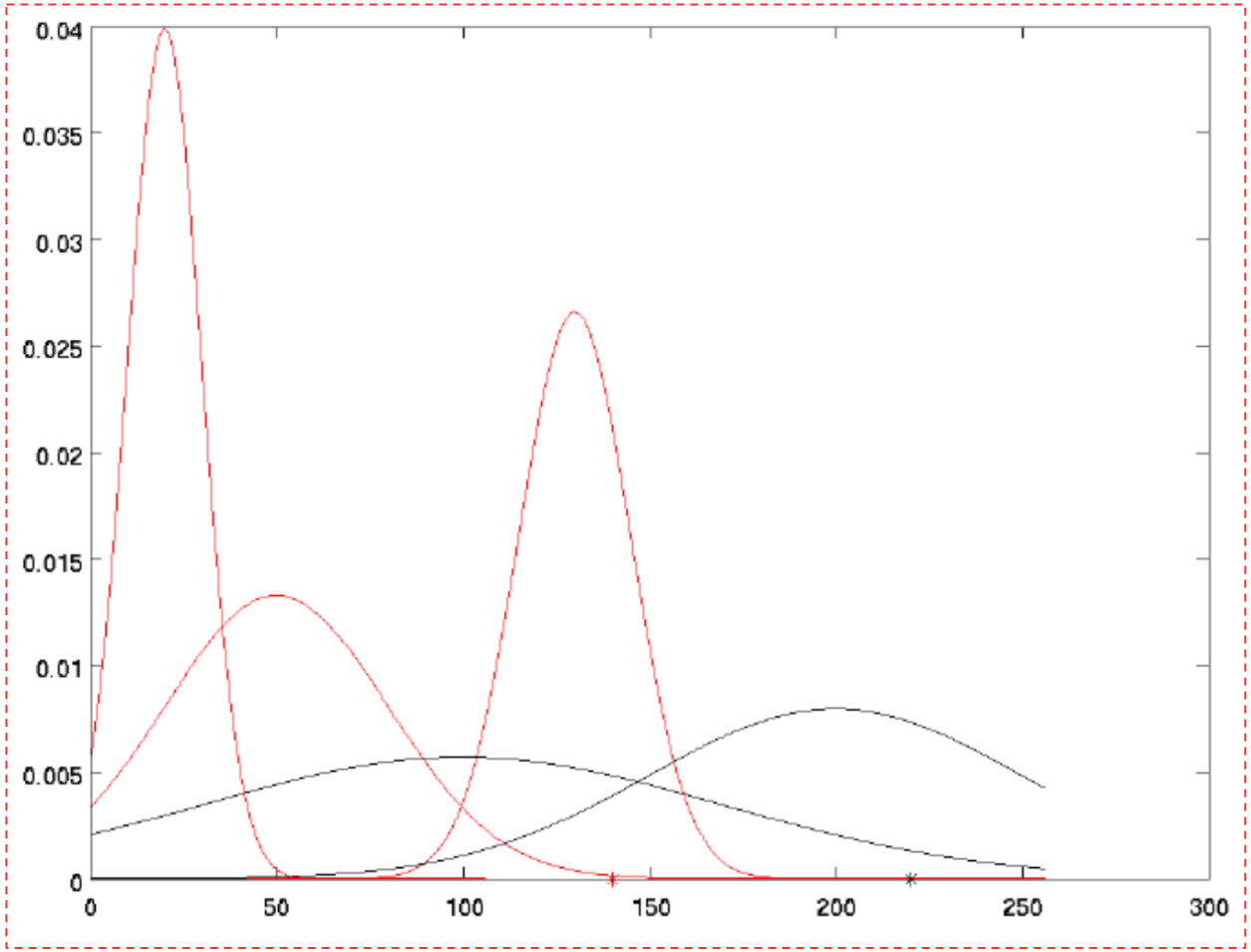
$$P(X_t) = \sum_{i=1}^{K} \omega_{it} \mathcal{N}(X_t | \mu_{it}, \Sigma_{it})$$

$$\mathcal{N}(X_t | \mu_{it}, \Sigma_{it}) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma_{it}|} \exp\left(-\frac{1}{2}(X_t - \mu_{it}^T \Sigma_{it}^{-1}(X_t - \mu_{it})\right)$$

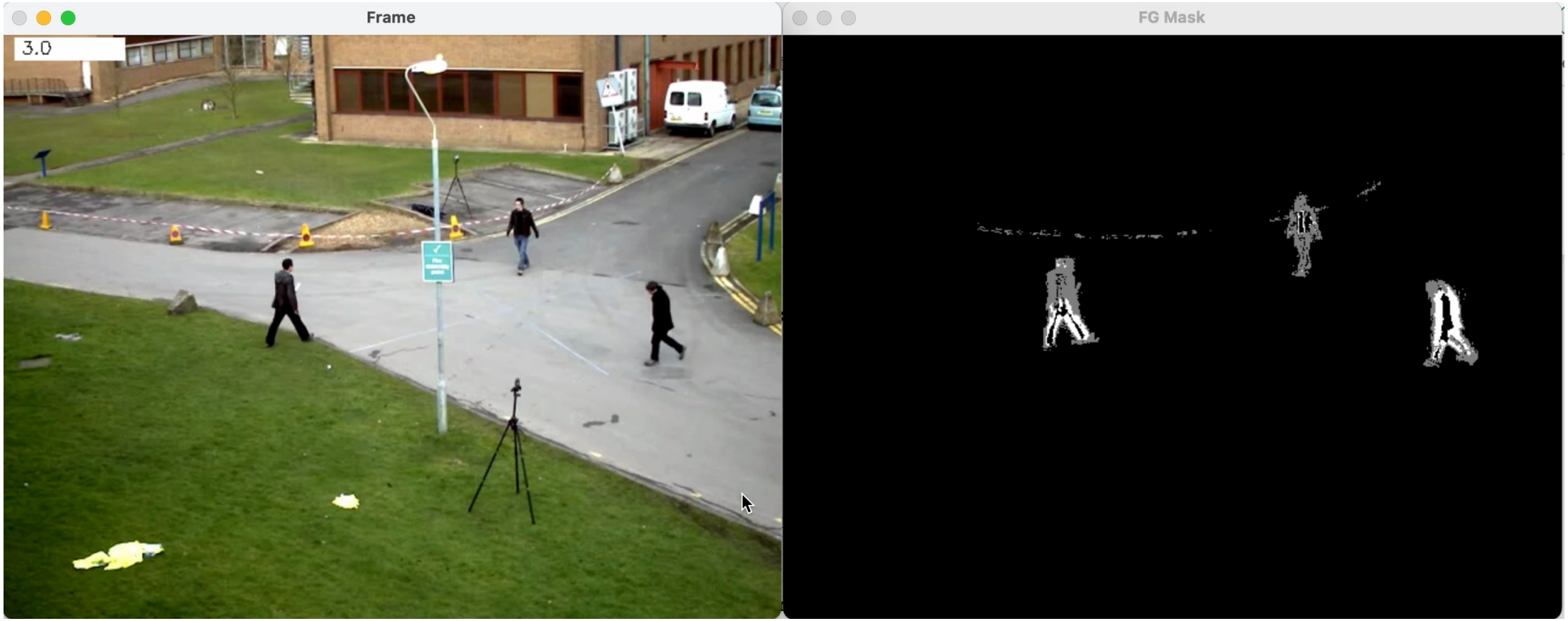


Mixture of Gaussian (MOG) Background Subtraction

After background model estimation red distributions become the background model and black distributions are considered to be foreground



Demo – BGS with MOG



Assignment Dicussion

AI VIET NAM – COURSE 2023

Background Subtraction

(Computer Vision Foundation)

Ngày 30 tháng 9 năm 2023

Giới thiệu về bài tập đếm số lượng người ra vào siêu thị trong ngày sử dụng giải thuật trừ nền (Background Subtraction) :

Cho trước thông tin hình ảnh lưu trữ khách hàng đến siêu thị được lưu trữ dưới dạng video file (dataset.mp4), hãy phát triển chương trình đếm tổng số người ra vào siêu thị bằng cách sử dụng các kỹ thuật computer visions cơ bản như background subtraction, contour detection, erosion và dilation. Hình 1 thể hiện dữ liệu mẫu của bài toán cần giải quyết.

Kết quả từ giải thuật Background Substraction

Kết quả định vị đối tượng và thống kê tổng số lượng người ra vào siệu thị


Thank
you