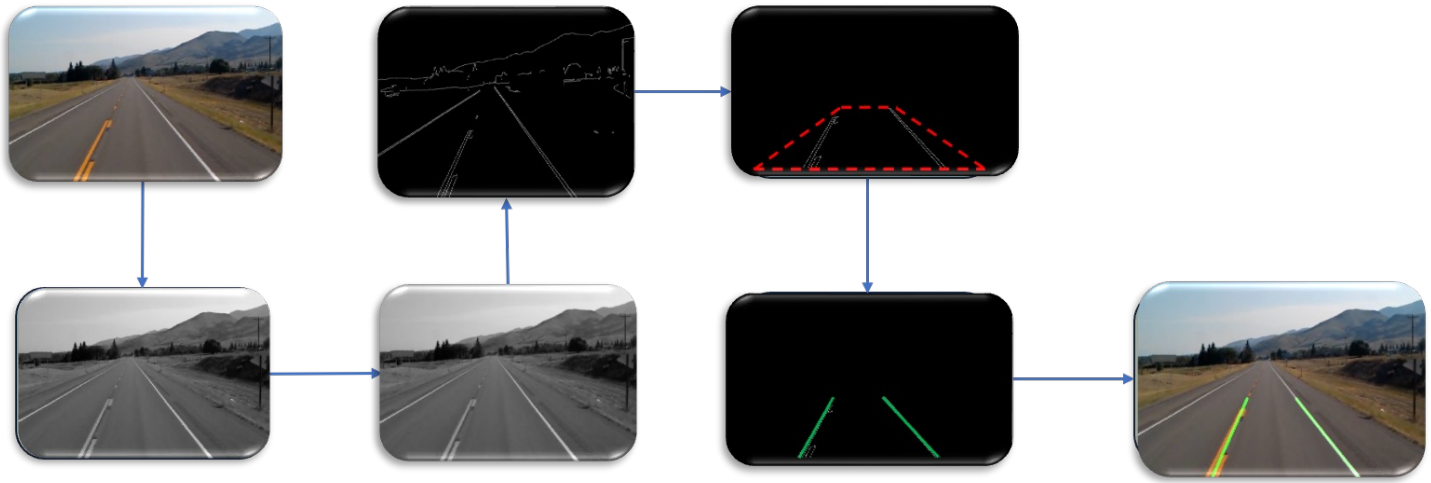
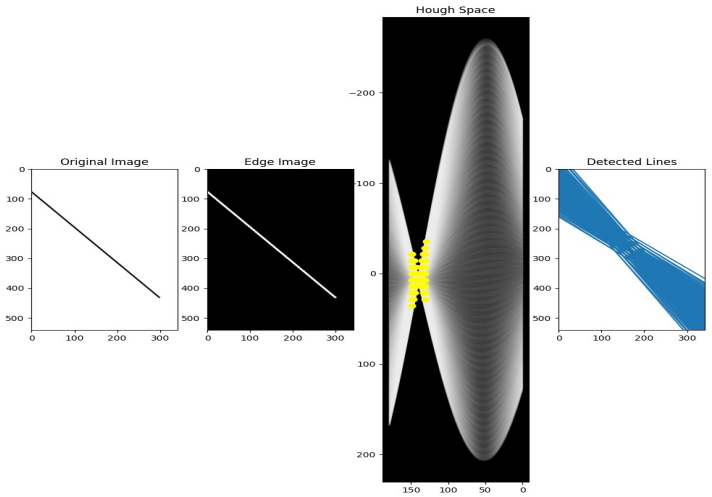
AI VIETNAM
All-in-One Course

Lane Detection

(Computer Vision Foundation)





Vinh Dinh Nguyen
PhD in Computer Science

Year 2023

Outline

- Background Subtraction Assignment: Review
- Grayscale Conversion
- Image Blurring Techniques
- Region of Interest (ROI)
- Line Detection based on Hough Transform
- Lane Detection Demo

Outline

- Background Subtraction Assignment: Review
- Grayscale Conversion
- Image Blurring Techniques
- Region of Interest (ROI)
- Line Detection based on Hough Transform
- Lane Detection Demo

Assignment 1 Review

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) :

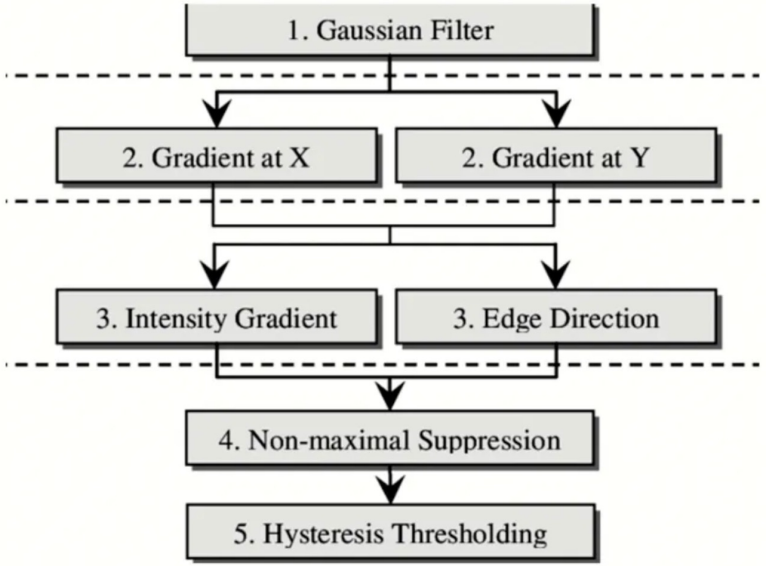
Cho trước thông tin hình ảnh lưu trữ khách hàng đến siêu thị được lưu trữ dưới dạng video file (dataset.mp4), hãy phát triển chương trình đếm tổng số người ra vào siêu thị bằng cách sử dụng các kỹ thuật computer visions cơ bản như background subtraction, contour detection, erosion và dilation. Hình 1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

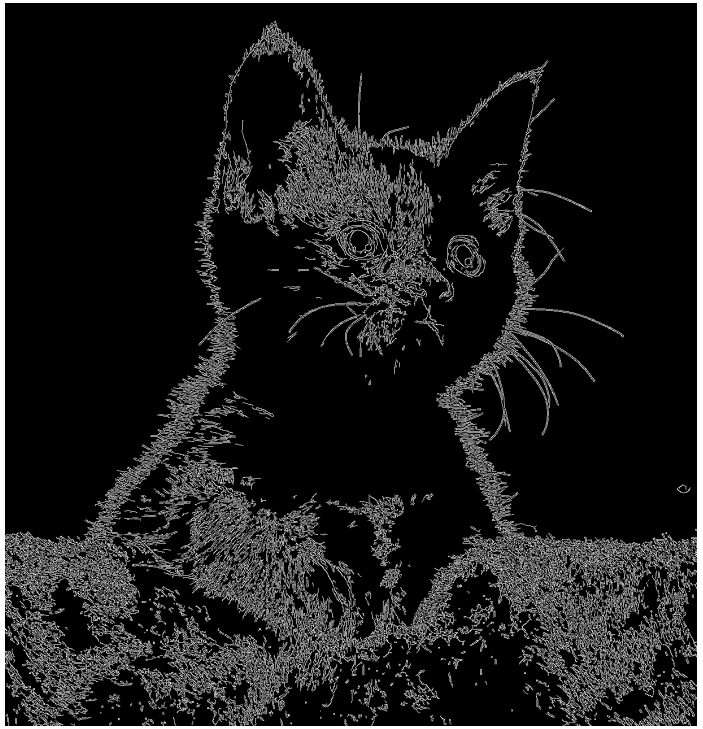
Kết quả từ giải thuật Background Substraction

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

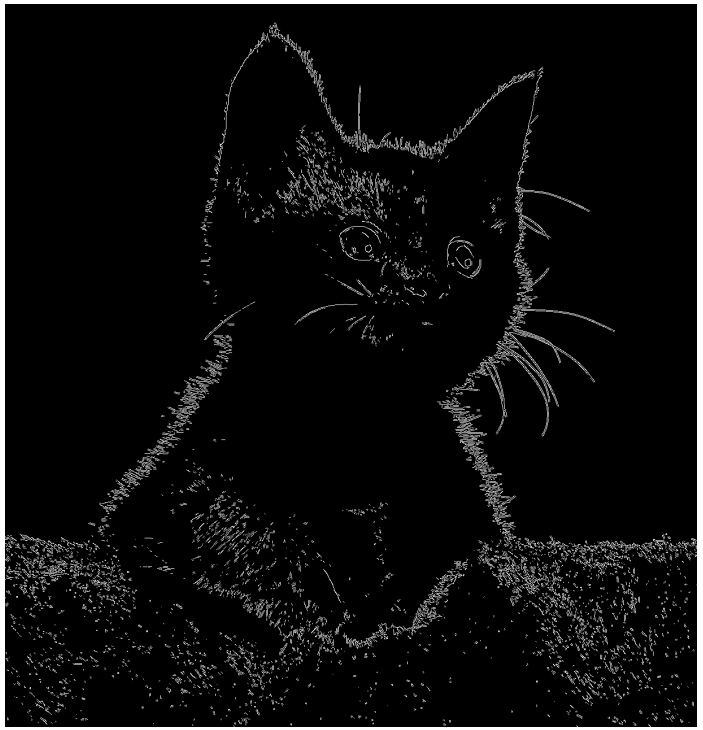
Edge Detection: Canny Algorithm

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





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

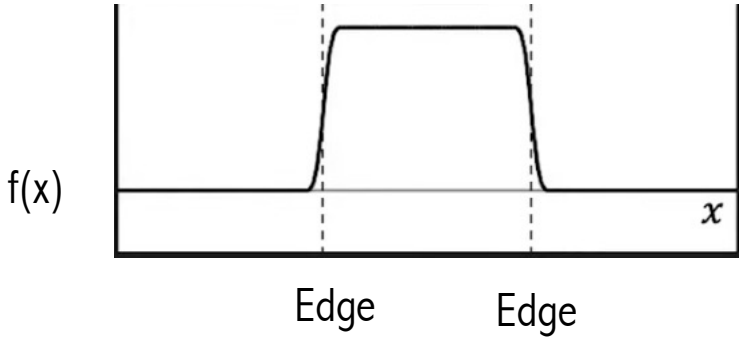


```
lower = 250 # Lower Threshold
upper = 300 # Upper threshold
```

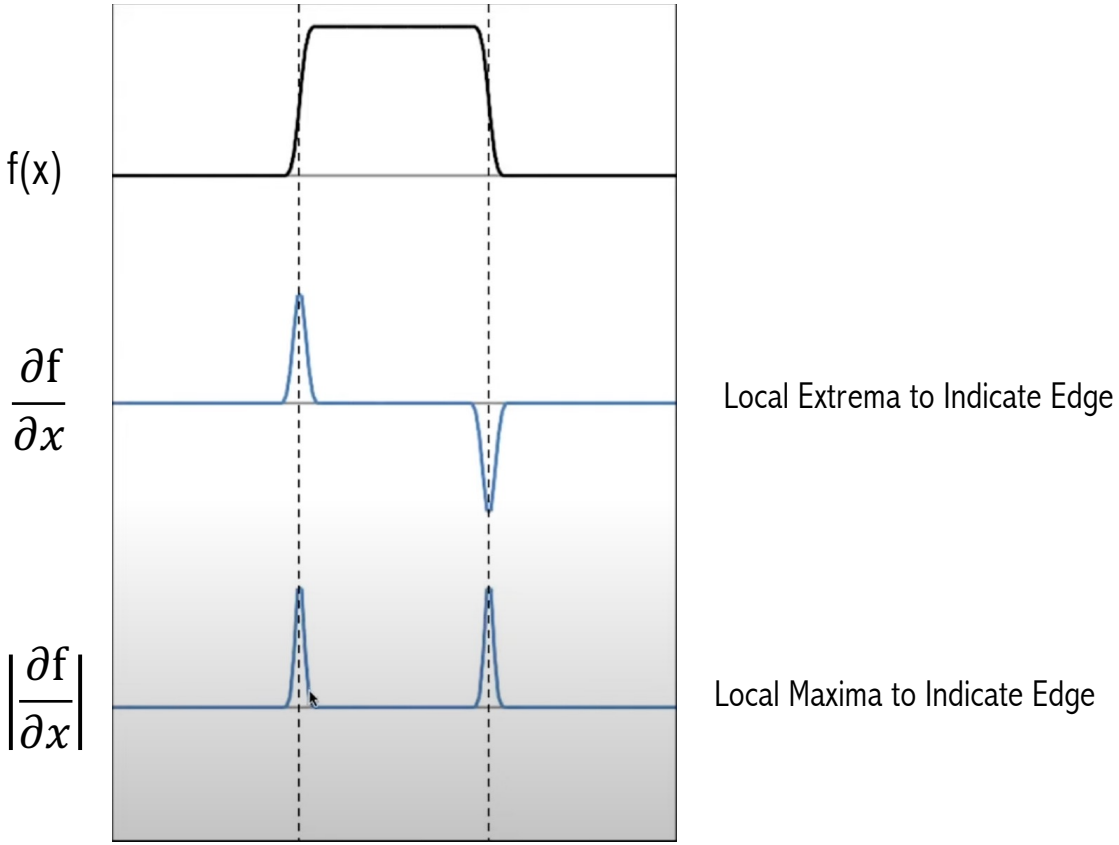
```
#Ege Detection Canny Algorithm
# Setting parameter values
t_lower = 250  # Lower Threshold
t_upper = 300  # Upper threshold

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

cv2_imshow(edge_canny)
```

1D Edge Detection

Edge is a rapid change in image intensity in a small region

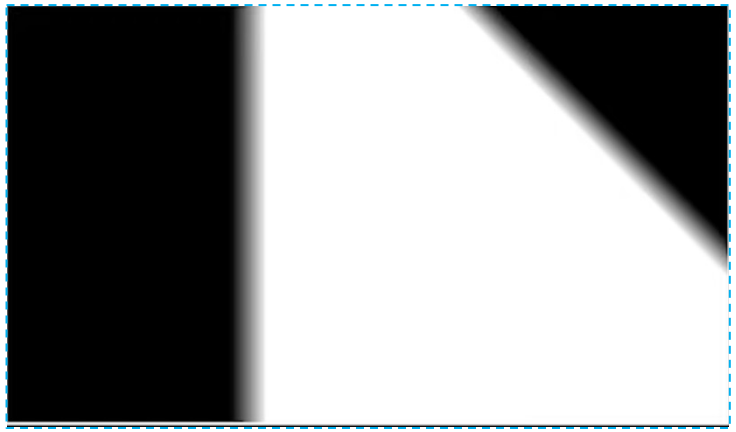


Basic calculus: Derivative of a continous functions represent the amount of changes in the function



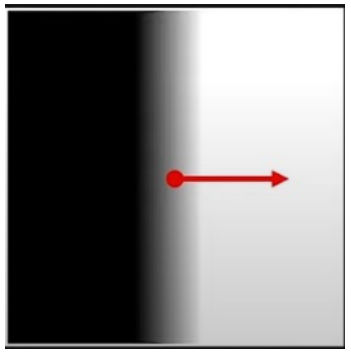
2D Edge Detection

I(x,y)

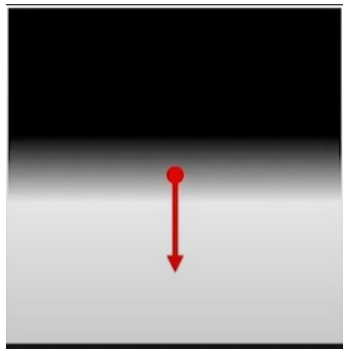


Edge

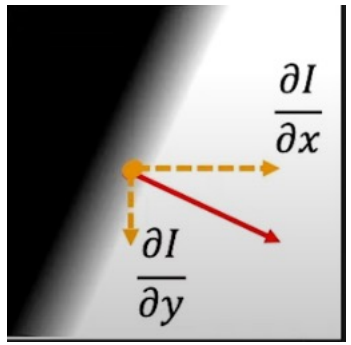
Edge



$$\nabla I = \left(\frac{\partial I}{\partial x}, 0\right)$$



$$\nabla I = \left(0, \frac{\partial I}{\partial y}\right)$$



$$\nabla I = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right)$$

Basic calculus: Partial derivatives of 2D continuous function represents the amount of changes along each dimension.

$$\nabla I = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right)$$

Continuous case

$$\frac{\partial I(x,y)}{\partial x} = \lim_{\Delta x \to 0} \frac{I(x+\Delta x, y) - I(x,y)}{\Delta x}$$

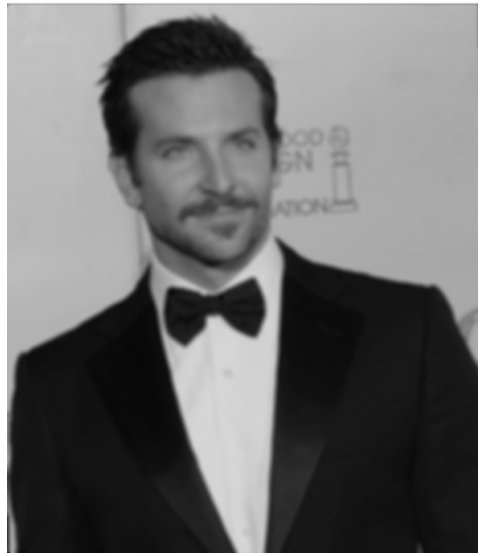
Discrete case

$$\frac{\partial I(x,y)}{\partial x} \approx \frac{I(x+1,y) - I(x-1,y)}{2}$$

$$\frac{\partial I(x,y)}{\partial y} \approx \frac{I(x,y+1) - I(x,y+1)}{2}$$

Can be implemented as Convolution

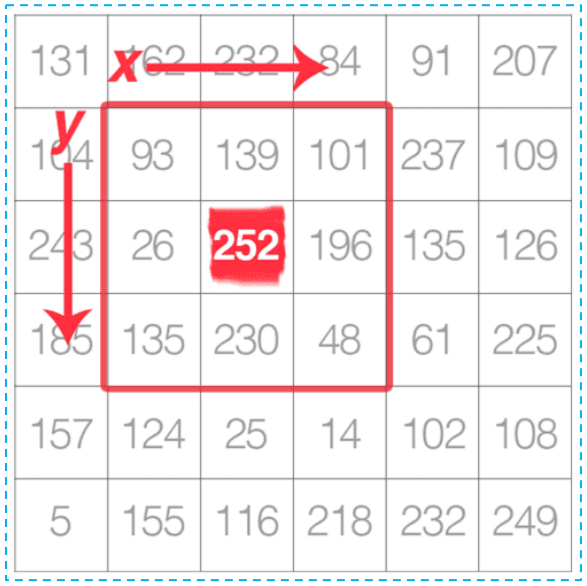
Gaussian Filter



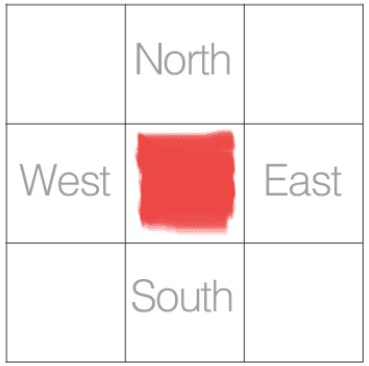
$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

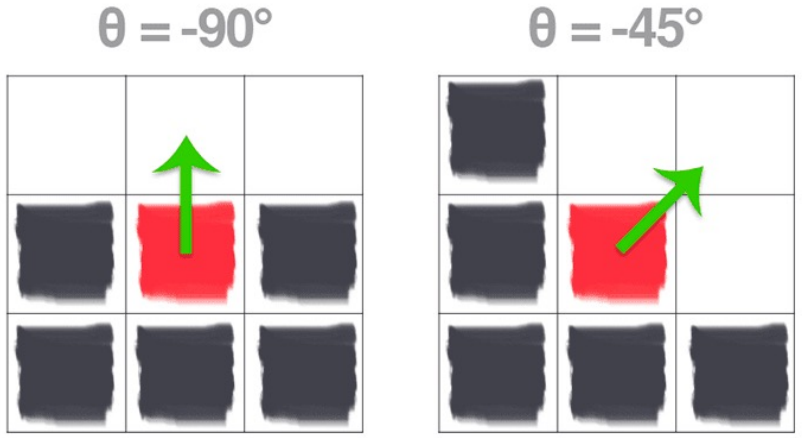
```python
def gaussian_kernel(size, sigma):
    if size % 2 == 0:
        size = size + 1

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

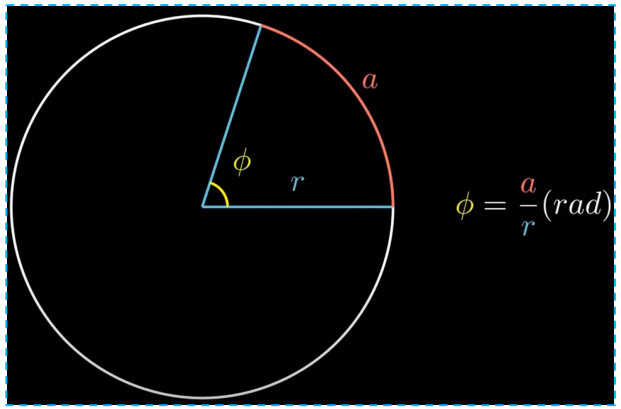
The gradient magnitude is used to measure how strong the change in image intensity is

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

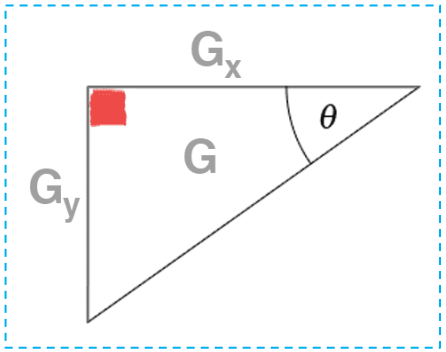
The gradient orientation is used to determine in which direction the change in intensity is pointing

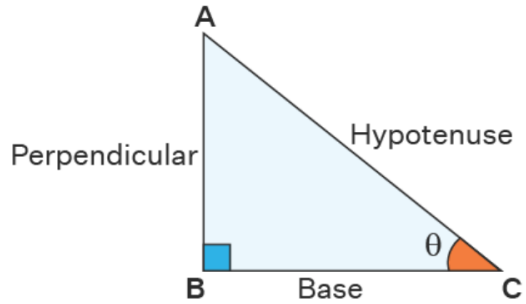
$$\theta = \arctan2(G_y, G_x) \times \left(\frac{180}{\pi}\right)$$

Image Gradient



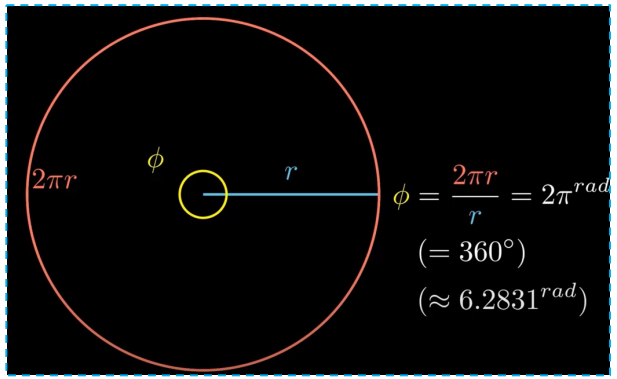
$$\theta = \arctan2(G_y, G_x) \times \left(\frac{180}{\pi}\right)$$

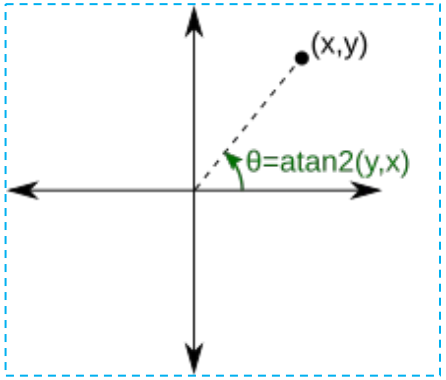




$$\tan\theta = \frac{\text{Perpendicular}}{\text{Base}}$$

$$\Rightarrow \theta = \tan^{-1}\left(\frac{\text{Perpendicular}}{\text{Base}}\right)$$

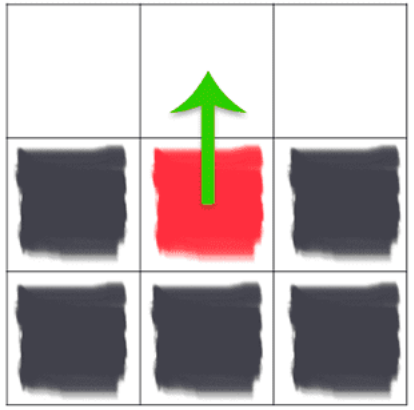




$$\text{atan2}(y,x) = \begin{cases} \arctan\left(\frac{y}{x}\right) & \text{if } x > 0, \\ \arctan\left(\frac{y}{x}\right) + \pi & \text{if } x < 0 \text{ and } y \geq 0, \\ \arctan\left(\frac{y}{x}\right) - \pi & \text{if } x < 0 \text{ and } y < 0, \\ +\frac{\pi}{2} & \text{if } x = 0 \text{ and } y > 0, \\ -\frac{\pi}{2} & \text{if } x = 0 \text{ and } y < 0, \\ \text{undefined} & \text{if } x = 0 \text{ and } y = 0. \end{cases}$$

Image Gradient

255	255	255
0		0
0	0	0



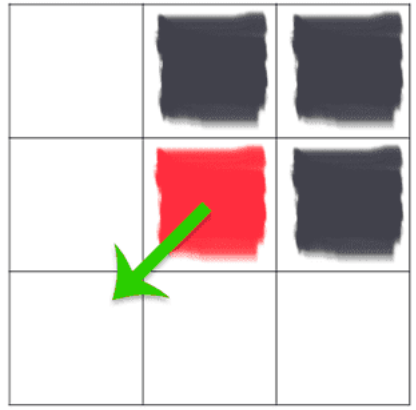
$$G = \sqrt{0^2 + (-255)^2} = 255$$

$G_x = 0 - 0 = 0$

$G_y = 0 - 255 = -255$

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

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

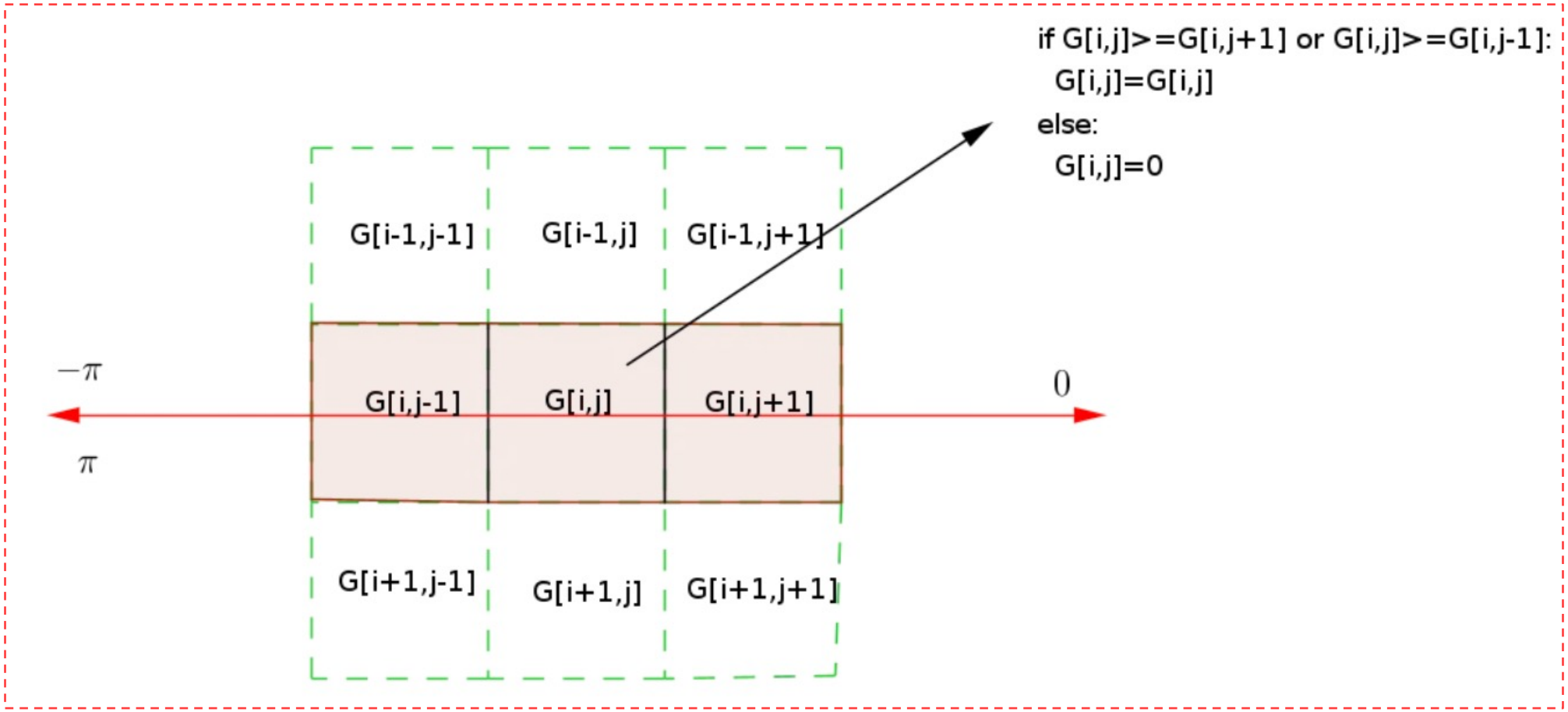
Gradient magnitude

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()
```

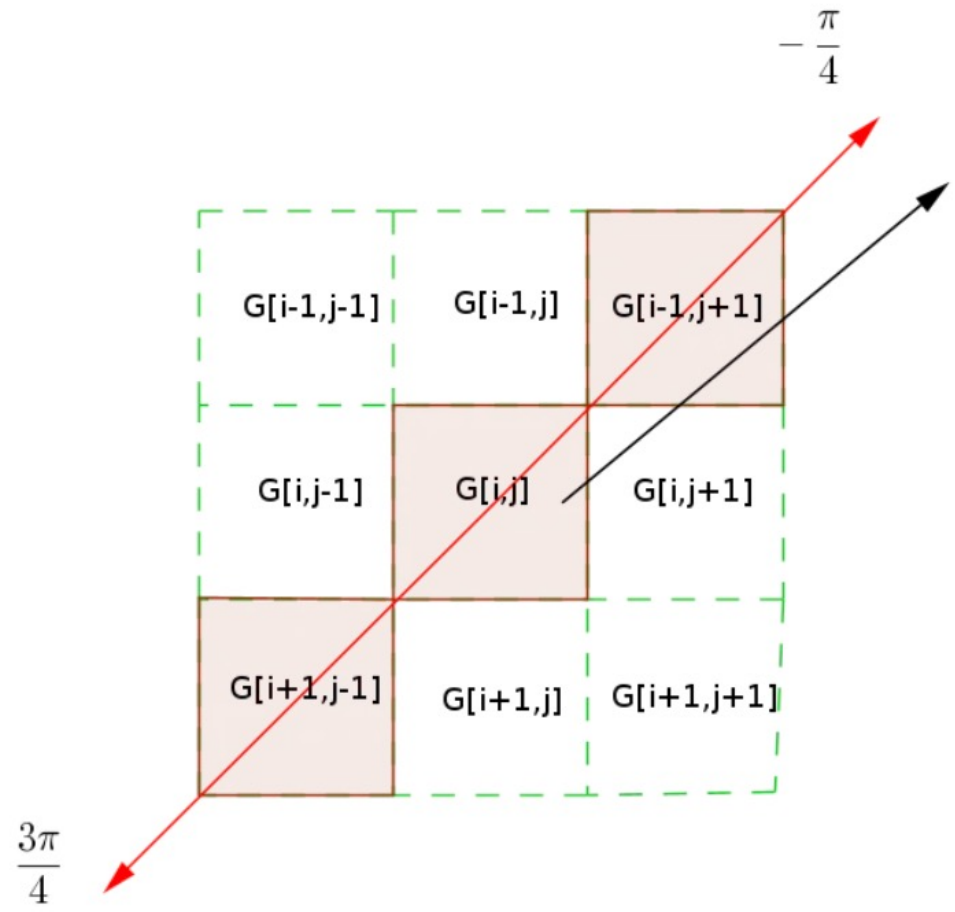
Gradient magnitude edges thicker than many edge detection. Next step is the make edges one pixel thick.

Non-Maximal Suppression



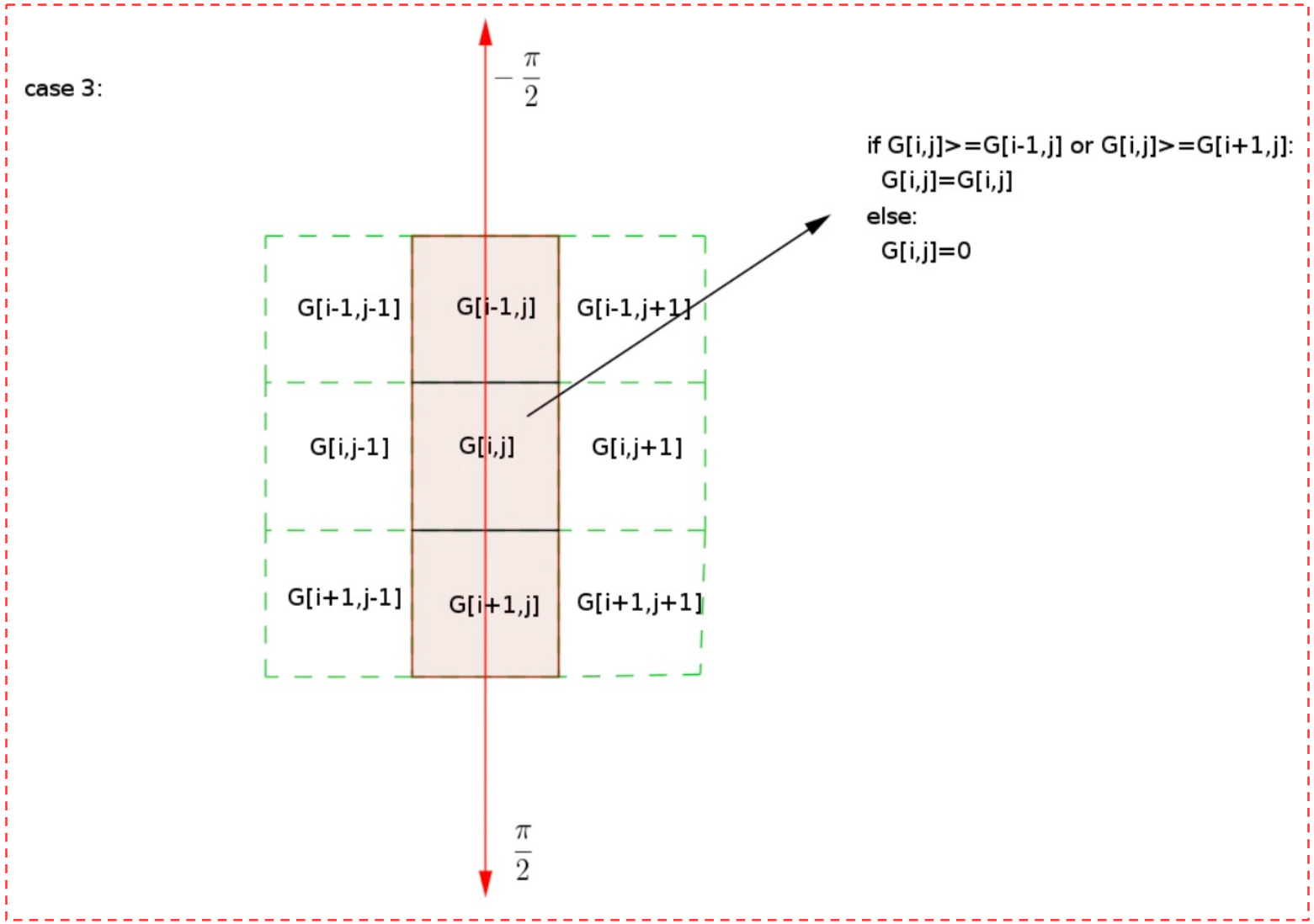
Non-Maximal Suppression

case 2:



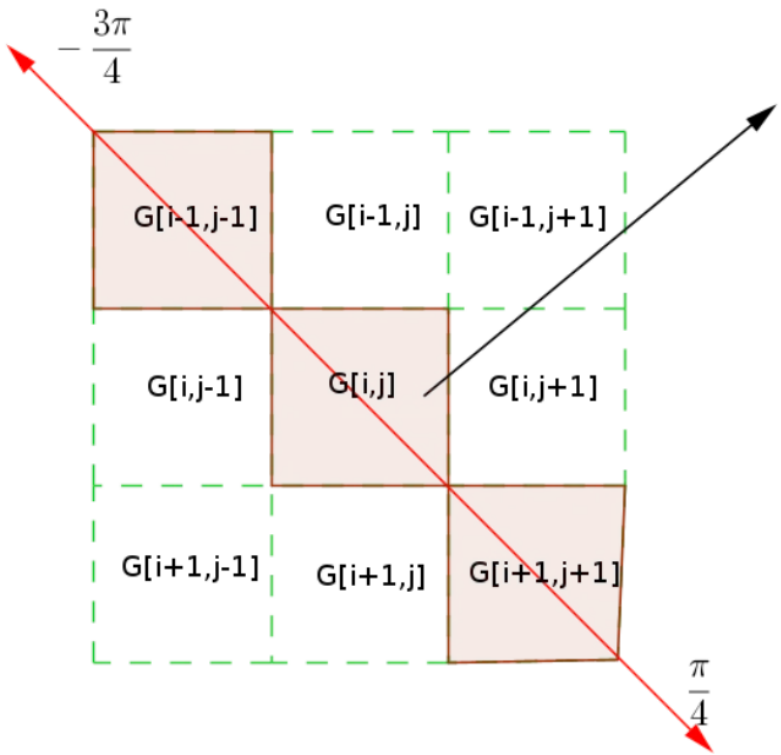

```
if G[i,j]>=G[i-1,j+1]or G[i,j]>=G[i+1,j-1]:
  G[i,j]=G[i,j]
else:
  G[i,j]=0
```

Non-Maximal Suppression



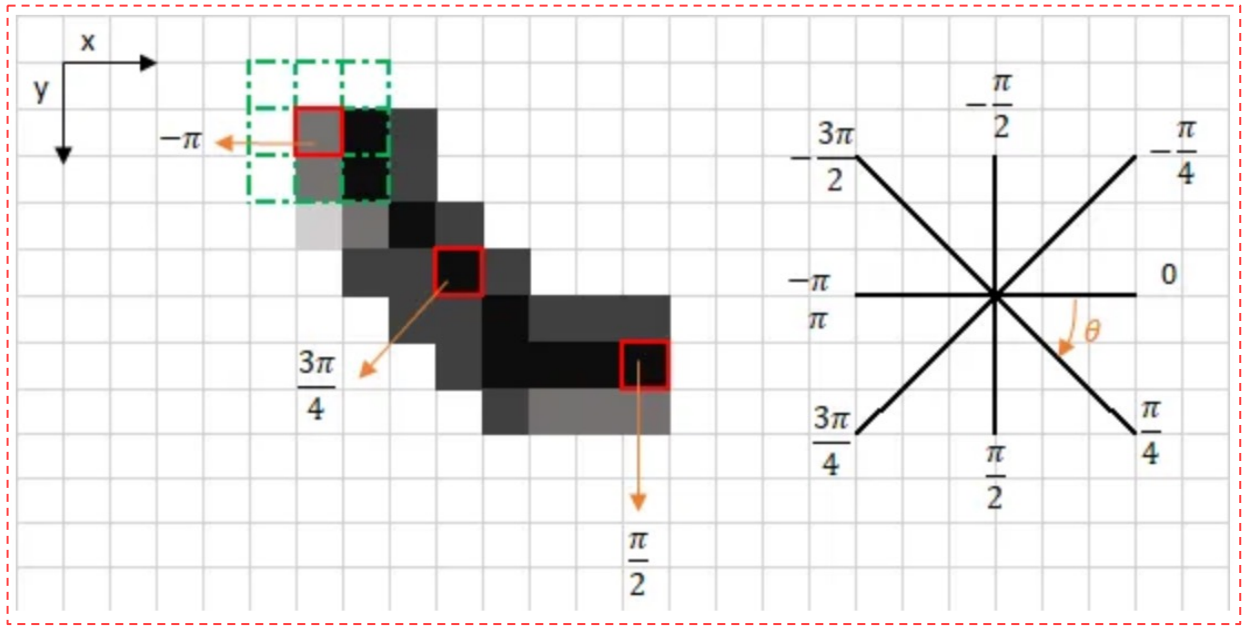
Non-Maximal Suppression

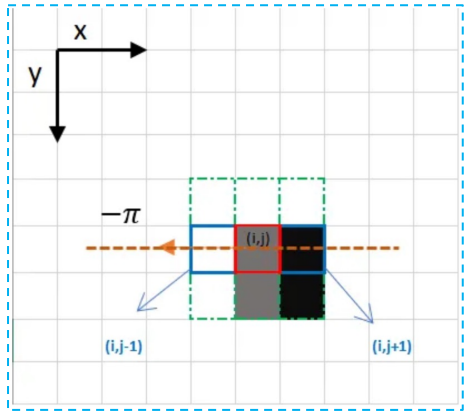
case 4:

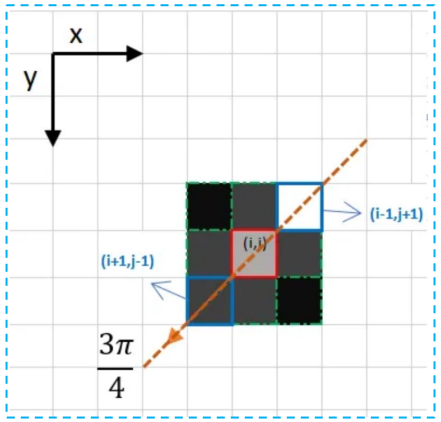



```
if G[i,j]>=G[i-1,j-1]or G[i,j]>=G[i+1,j+1]:
  G[i,j]=G[i,j]
else:
  G[i,j]=0
```

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

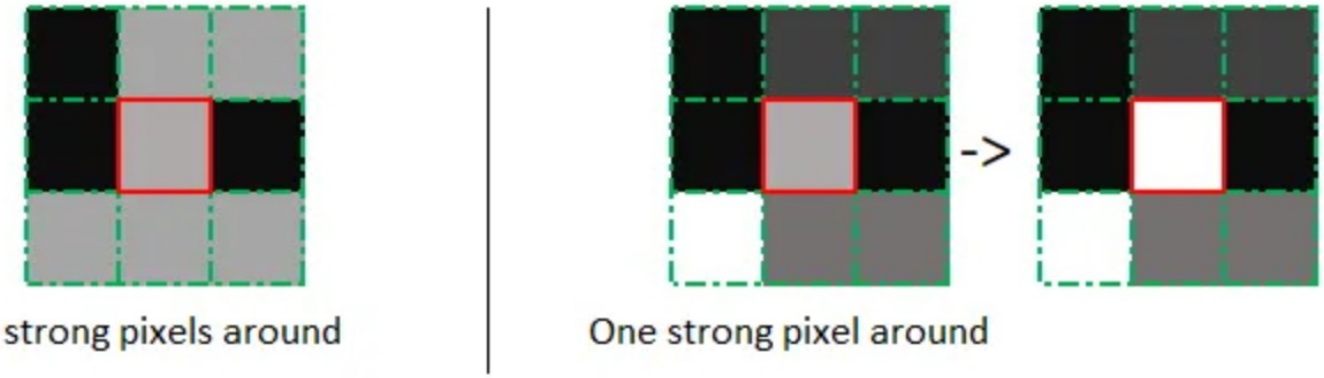
The double threshold step aims at identifying 3 kinds of pixels: strong, weak, and non-relevant

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

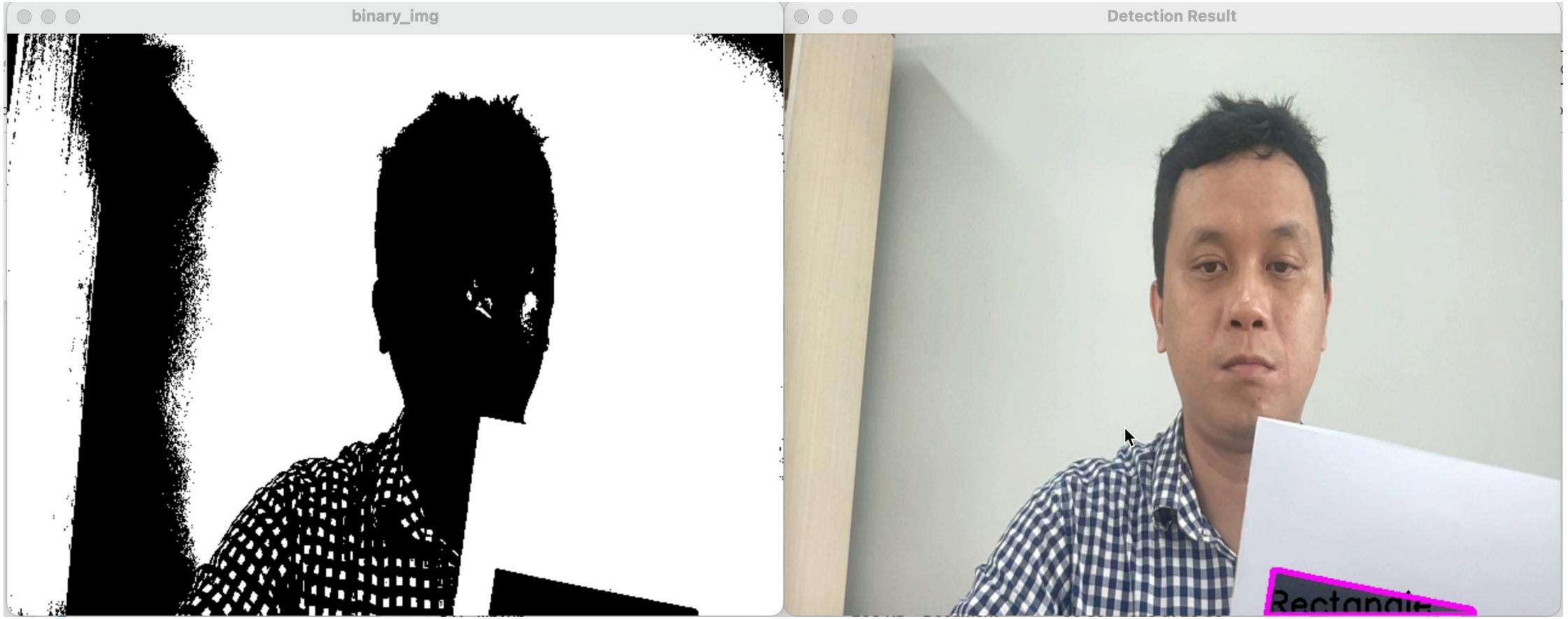



```python
def hysteresis(img, weak, strong=255):
    M, N = img.shape
    for i in range(1, M-1):
        for j in range(1, N-1):
            if (img[i,j] == weak):
                try:
                    if ((img[i+1, j-1] == strong) or (img[i+1, j] == strong) or (img[i+1,
                        or (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

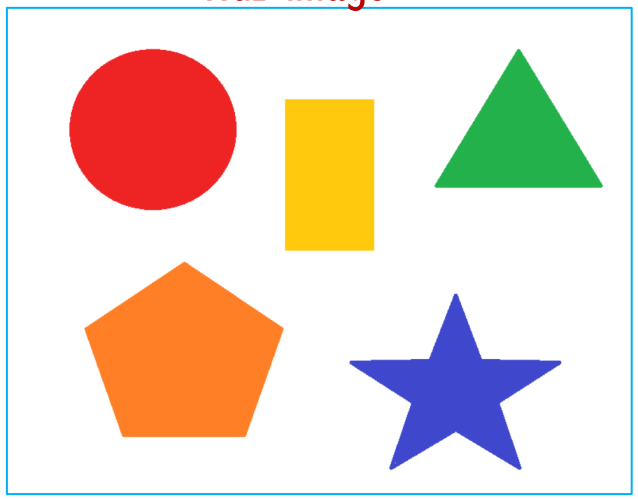
Final Result

Review Online Shape Detection

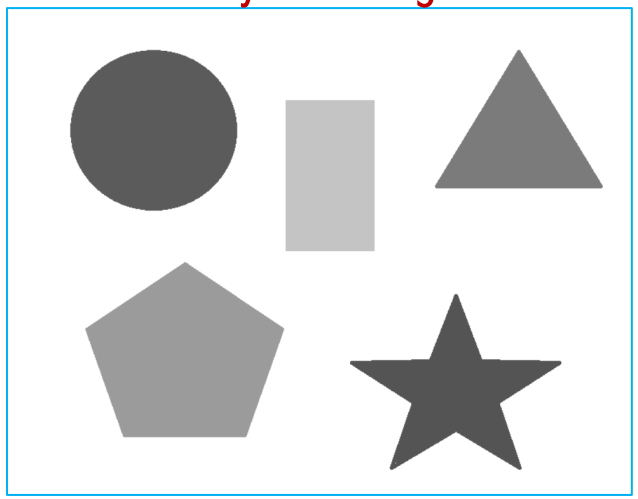


Online Shape Detection

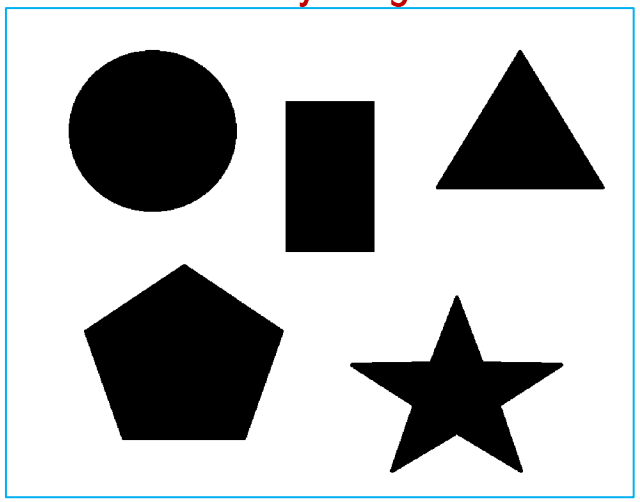
RGB image



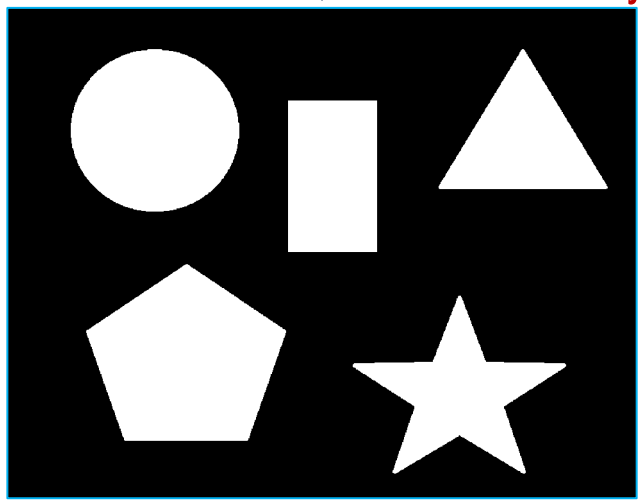
Grayscale image



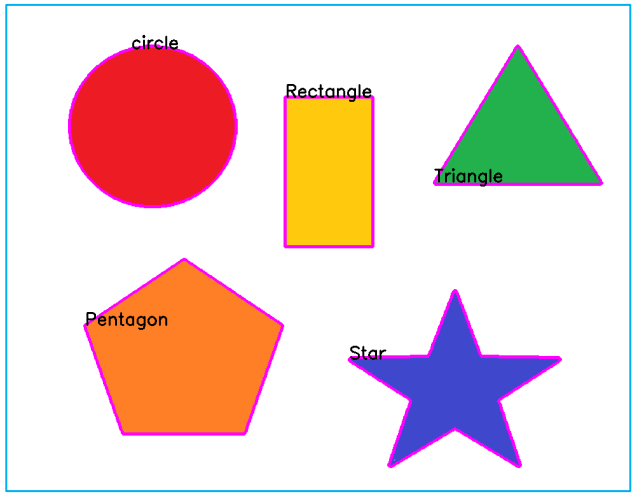
Binary Image



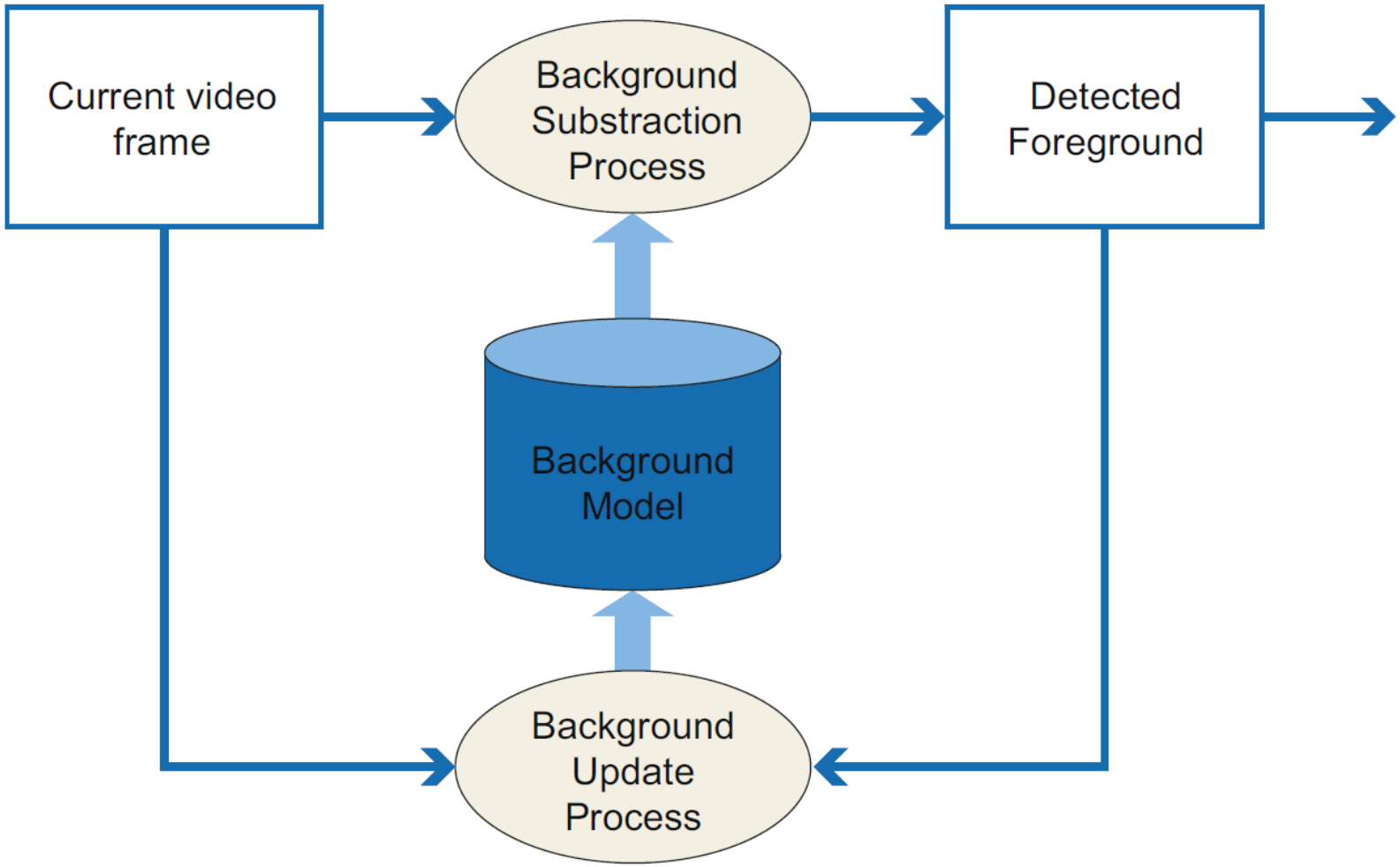
Inverse Binary Image



Result



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

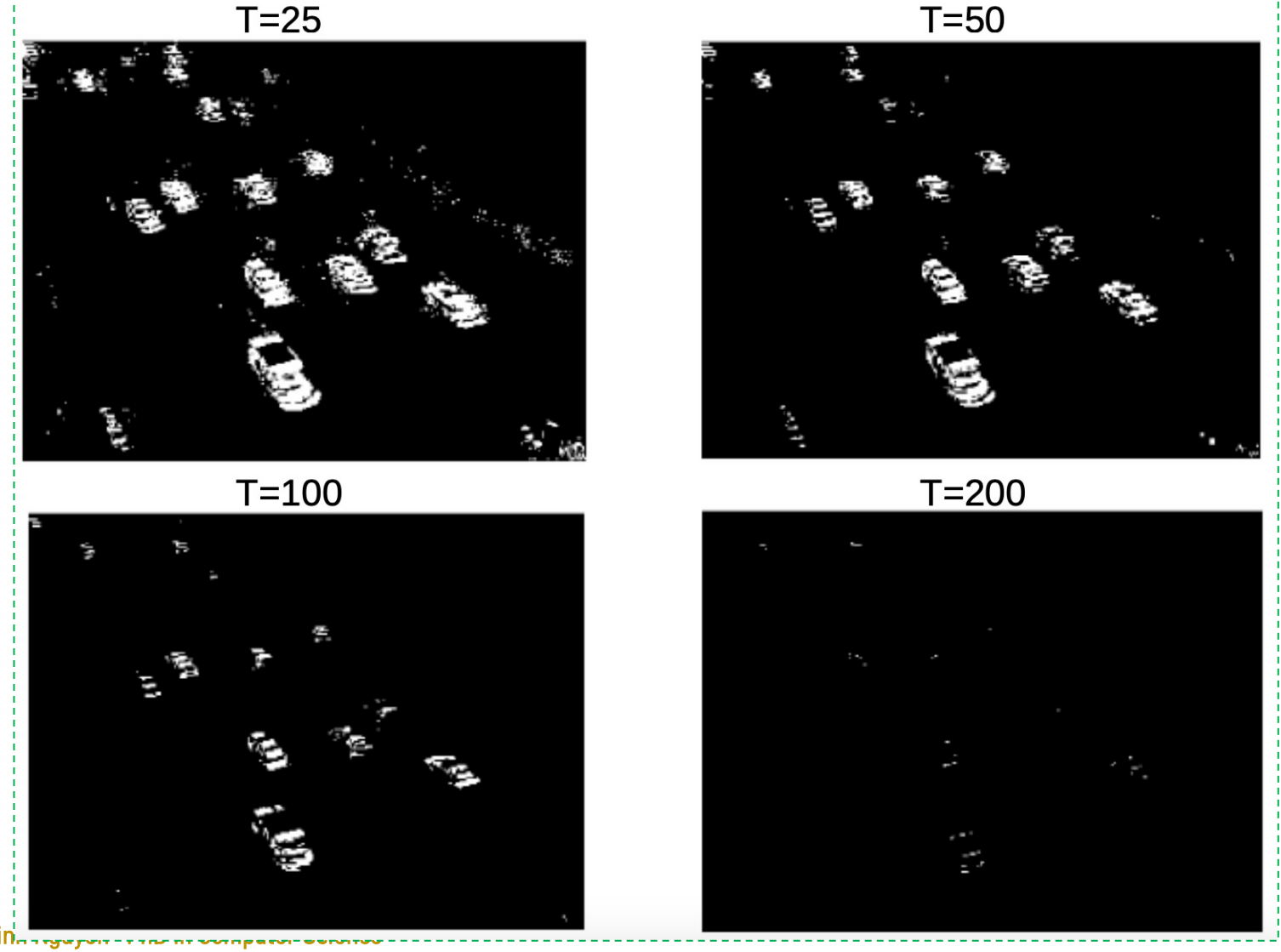
$$B(x, y, t) = I(x, y, t-1)$$

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

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

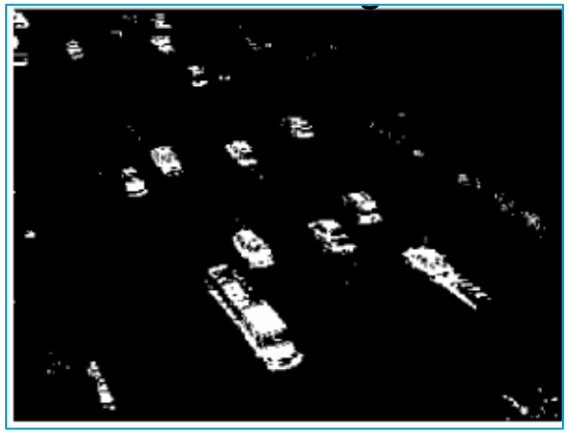
Mean Filter

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

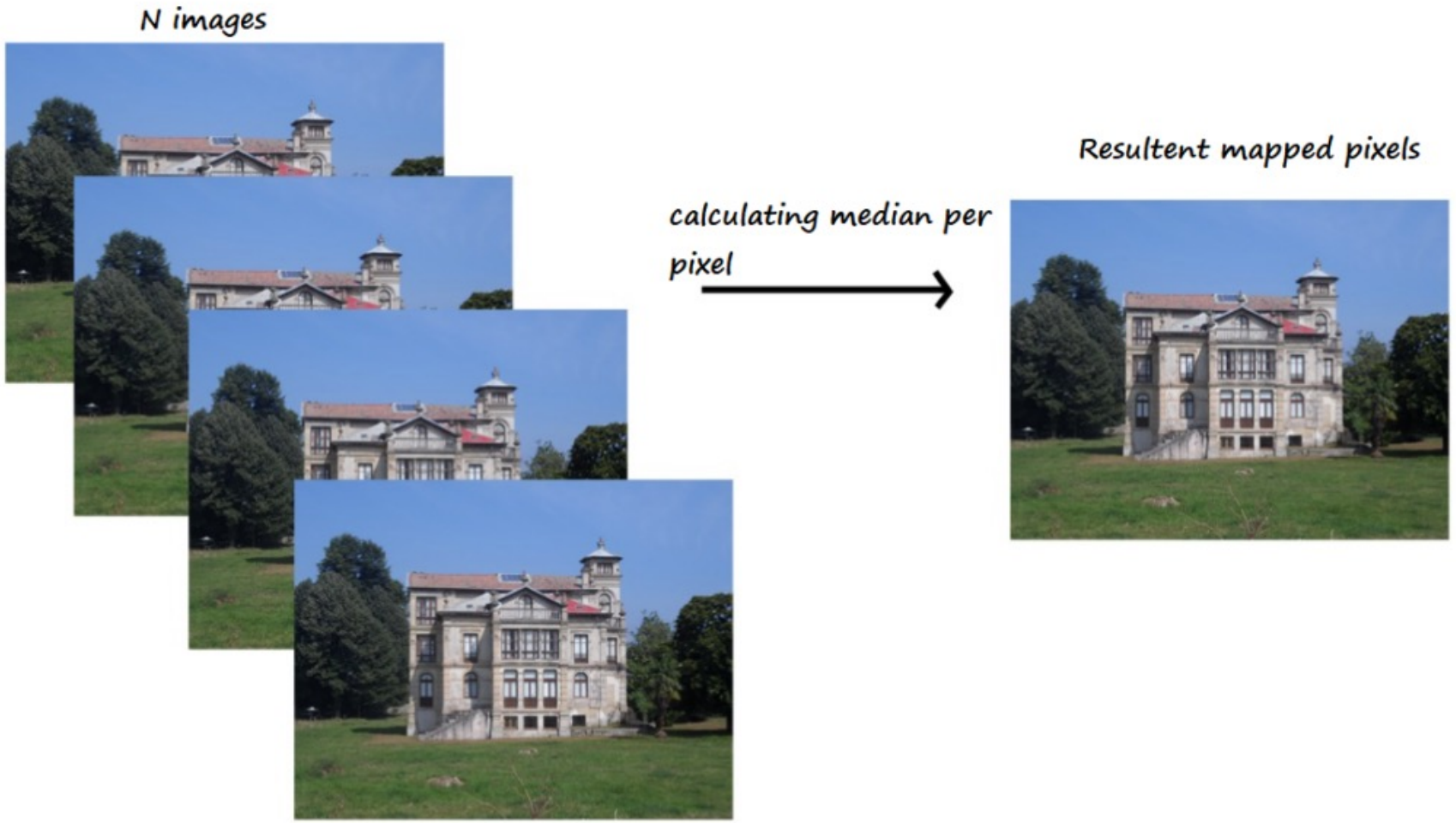
Assuming that the background is more likely to appear in a scene, we can use the median of the previous n frames as the background model

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

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

	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

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

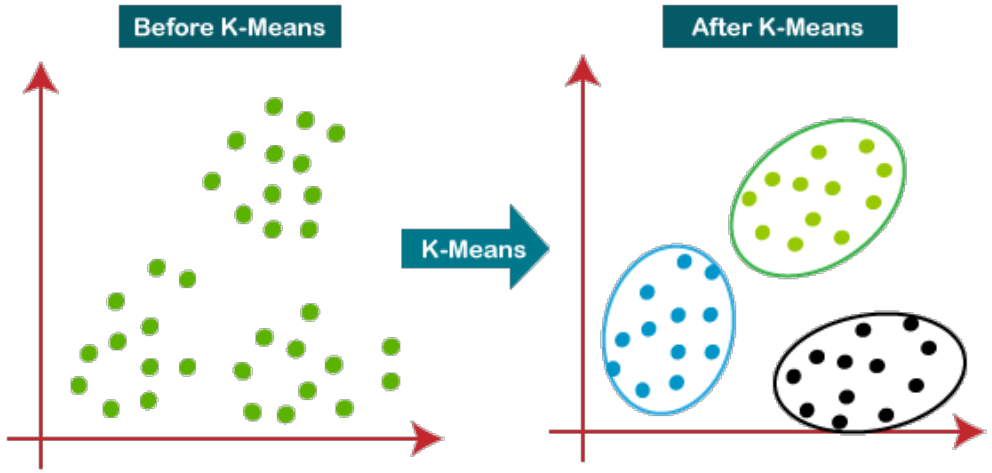
4. Minimize the cost function based with respective to:

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

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

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

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



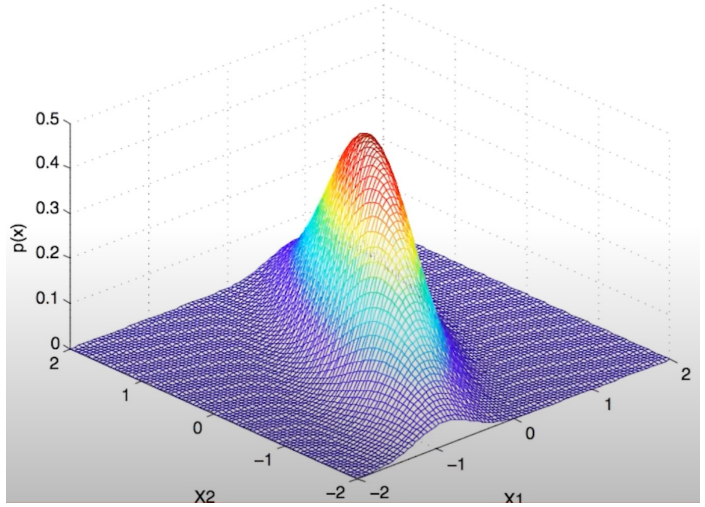
Quick look at Gaussian Mixture Model

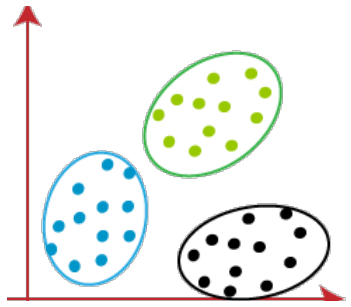
Multivariate Gaussian distribution for $\mathbf{x} \in \mathrm{R}^d$:

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

- 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,

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

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

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

- 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

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

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?

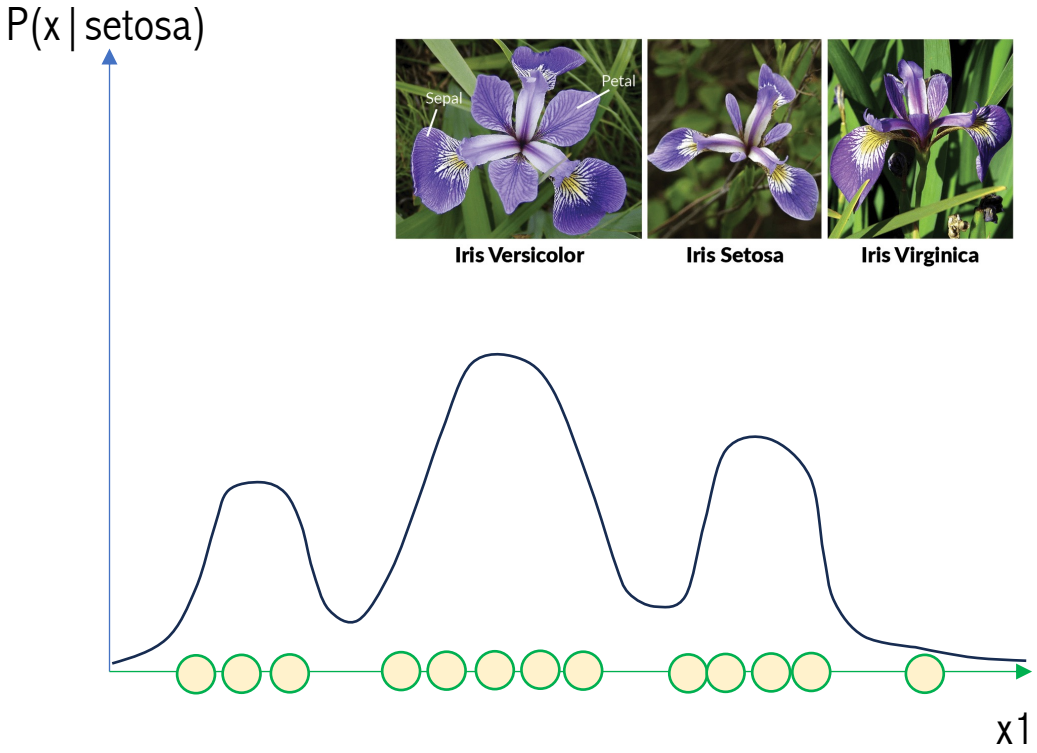
- use same algorithm
- update μ_k and σ_k^2 with maximum likelihood estimator,

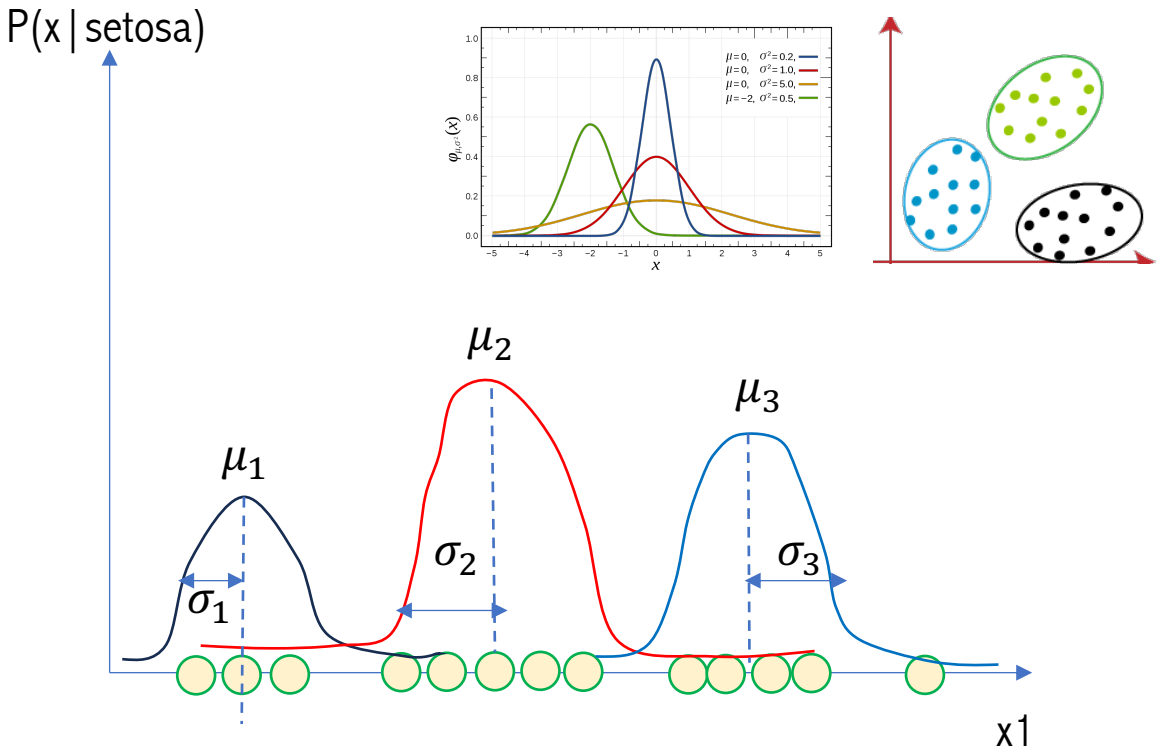
$$\mu_{k,j} = \frac{1}{n_k} \sum_{c_i = k} x_{i,j}$$

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

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

Quick look at Gaussian Mixture Model





Covariance Matrix

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

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

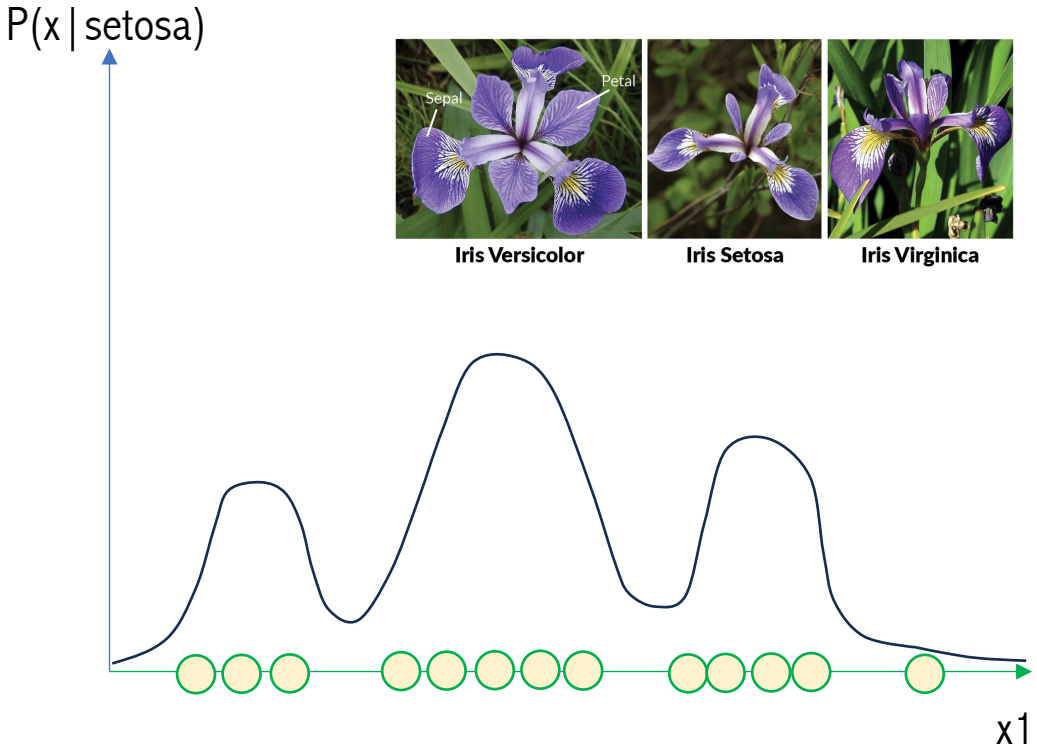
Latent variable (binary): z_k

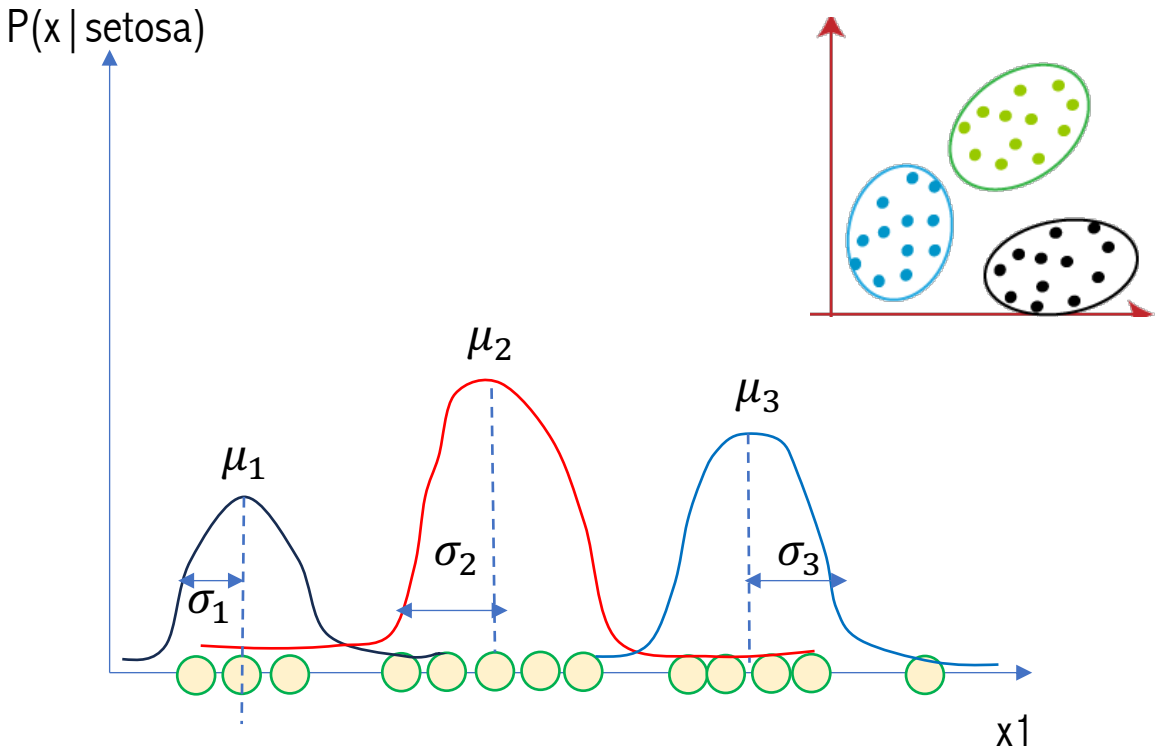
$$P(z_k=1) = \pi_k \qquad P(z_1=1) = \pi_1$$

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

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

Quick look at Gaussian Mixture Model





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

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

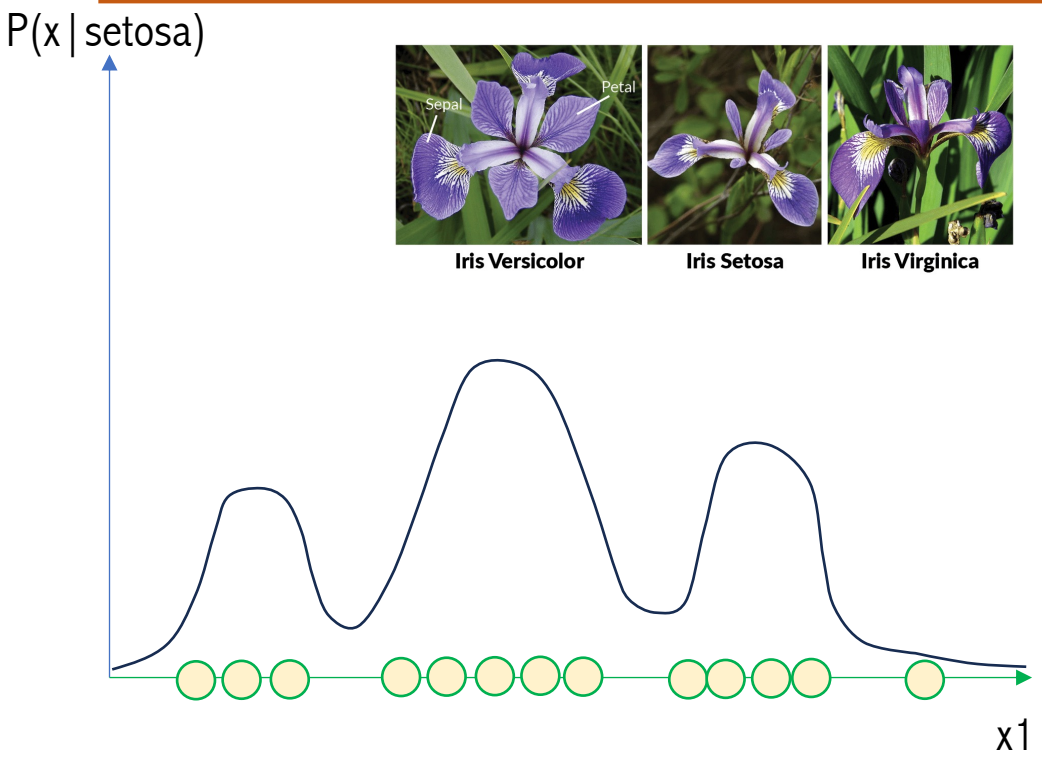
$$P(x) = \sum_{k=1}^{k} P(z_k=1)P(x|\ z_k=1)$$

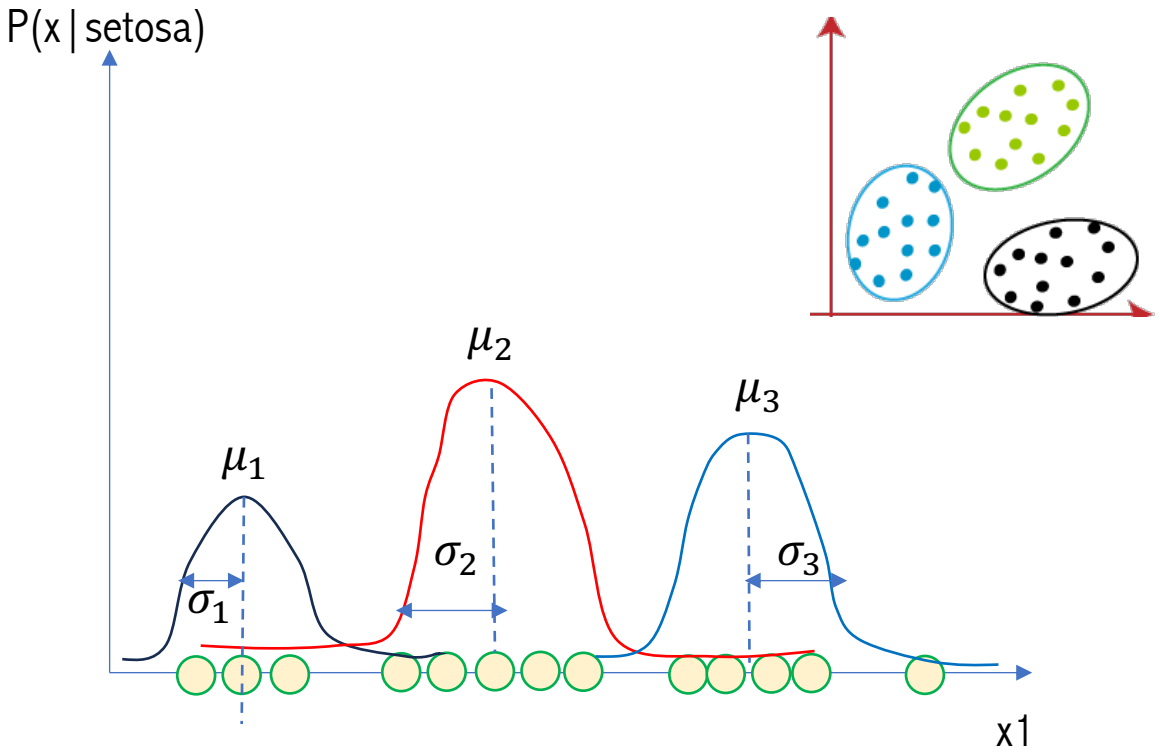
$$P(z_k=1|x) = \frac{P(x|\ z_k=1)P(z_k=1)}{P(X)}$$

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





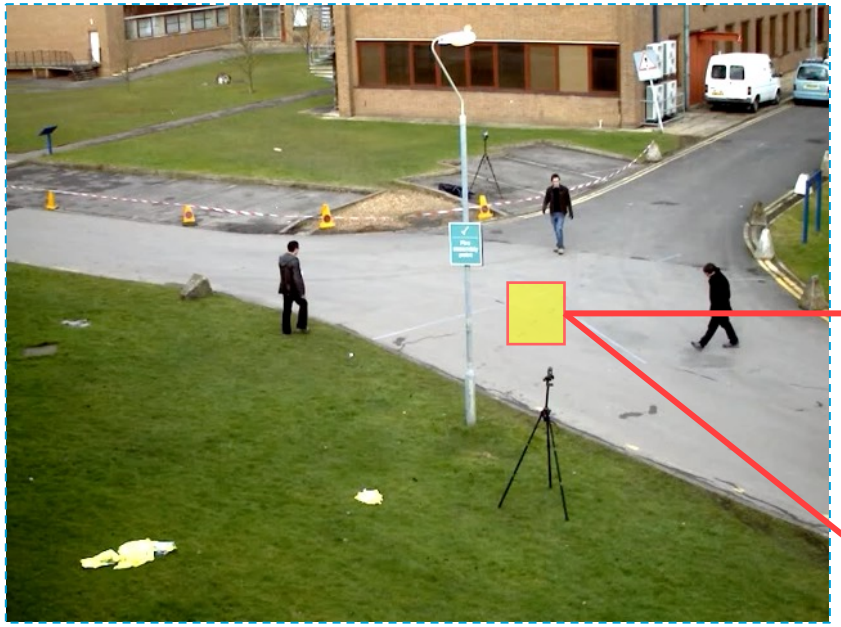
$$\ln(P(X)) = \prod_{i=1}^{N} \ln(\sum_{k=1}^{K} \pi_k N(X|\mu_k, \Sigma_k))$$

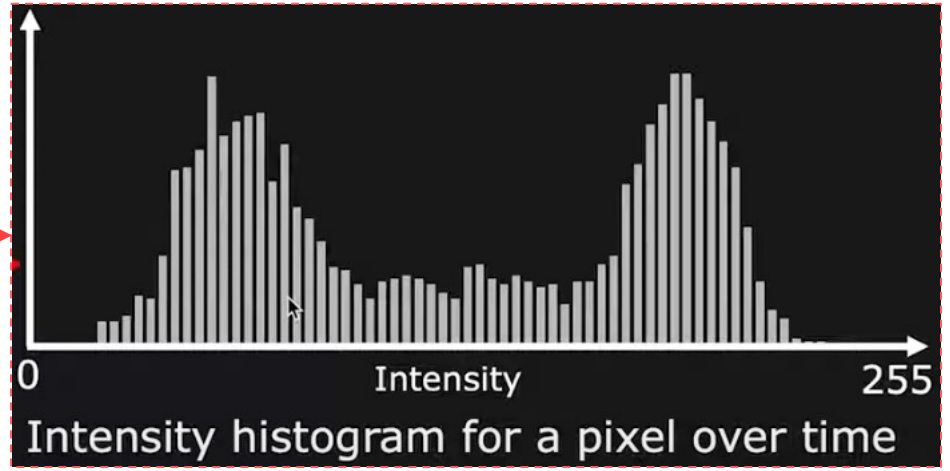
$$\ln(P(X|\ \pi, \mu, \Sigma\ )) = \prod_{i=1}^{N} \ln(\sum_{k=1}^{K} \pi_k N(X|\mu_k, \Sigma_k))$$

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

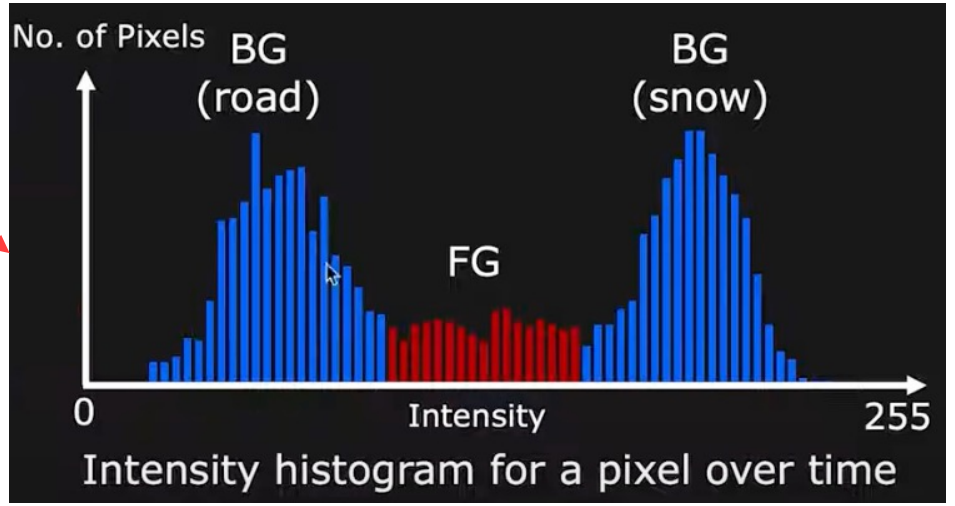
$$\frac{\partial \ln(P(X|\ \pi, \mu, \Sigma\ )}{\partial \Sigma_k} = 0$$

Mixture Model



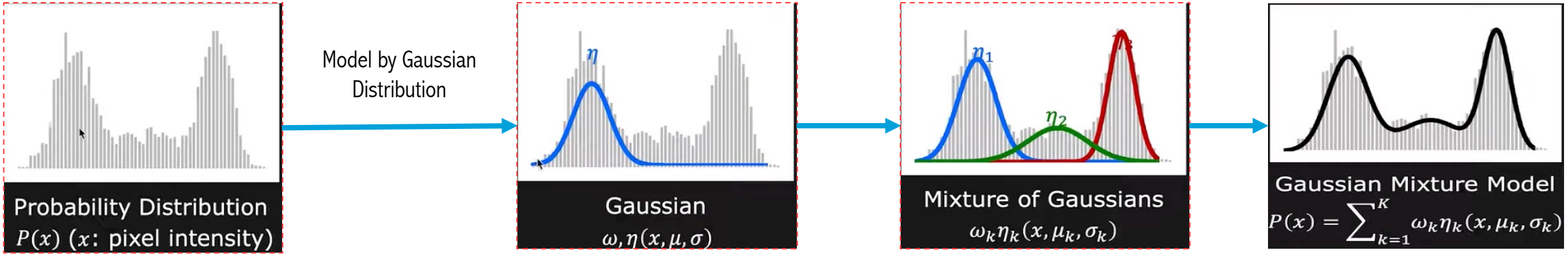


Intensity variation due to static scene (road), and occasional moving objects (vehicles)



Intuition: Pixels are background most of the time. We try to model the distribution and then classify a pixel as a foreground or a background based on this distribution.

Gaussian Mixture Model



1- Dimensional Gaussian

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

Assump P(x) is made with K different Gaussians.

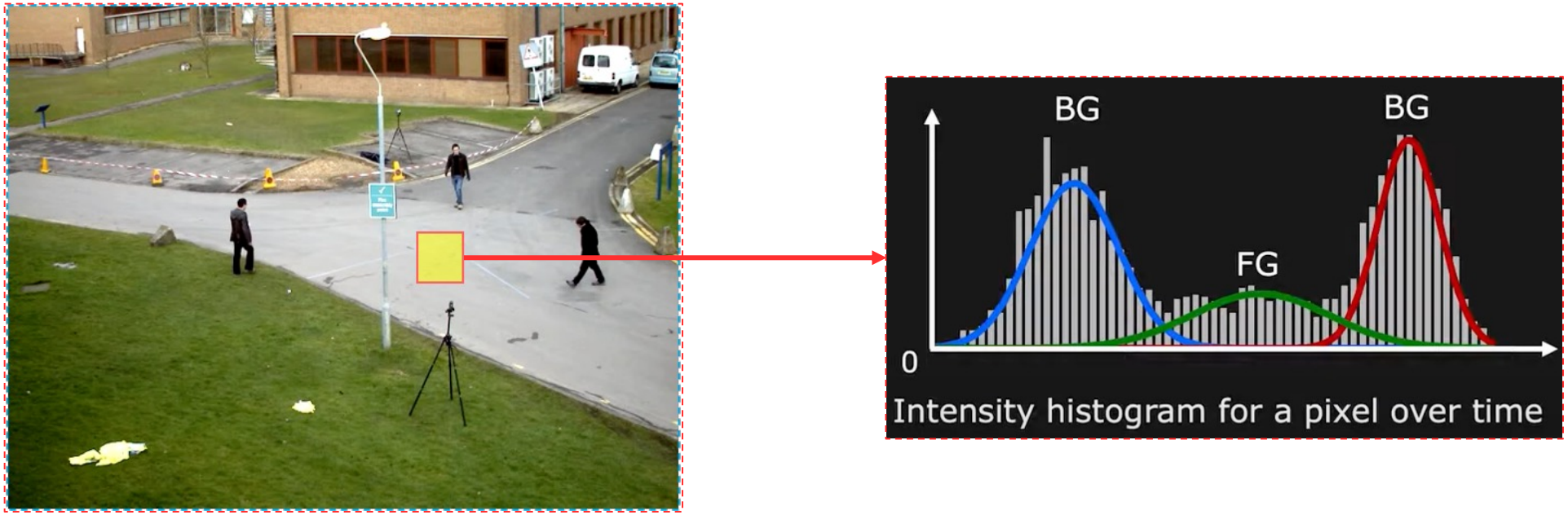
GMM Distribution: Weighted sum of K Gaussians

$$P(x) \cong \sum_{k=1}^{K} \omega_k \eta_k(x, \mu_k, \sigma_k) \quad \text{such that} \quad \sum_{k=1}^{K} \omega_k = 1$$

How about high dimensional GMM?

Background Modeling with GMM

- Given: A GMM for intensity/color variation at a pixel over time
- Classify: Individual Gaussian as foreground or background



Intuition: Pixels are background most of the time. That is, Gaussians with large supporting evidence (scale) ω and small σ

Large: $\frac{\omega}{\sigma} = Background$

Small: $\frac{\omega}{\sigma} = Foreground$

Change Detection Using GMM

For each pixel:

1. Compute pixel color histogram H using first N frames
2. Normalize histogram: $\widehat{H} = \frac{H}{\|H\|}$
3. Model as $\widehat{H}$ mixture of K (3 to 5) Gaussians
4. For each subsequent frame:
   a. The pixel value X belongs to Gaussian k in GMM for which $\|X - \mu_k\|$ is minimum and $\|X - \mu_k\| < 2.5\ \sigma_k$
   b. if $\frac{\omega}{\sigma}$ is large then classify pixel as background. Else classify as foreground
   c. Update histogram H using new pixel intensity
   d. if $\widehat{H}$ - $\frac{H}{\|H\|}$ is large, we update $\widehat{H} = \frac{H}{\|H\|}$, and then refit GMM

Background Subtraction With Mixture of Gaussian (MOG)

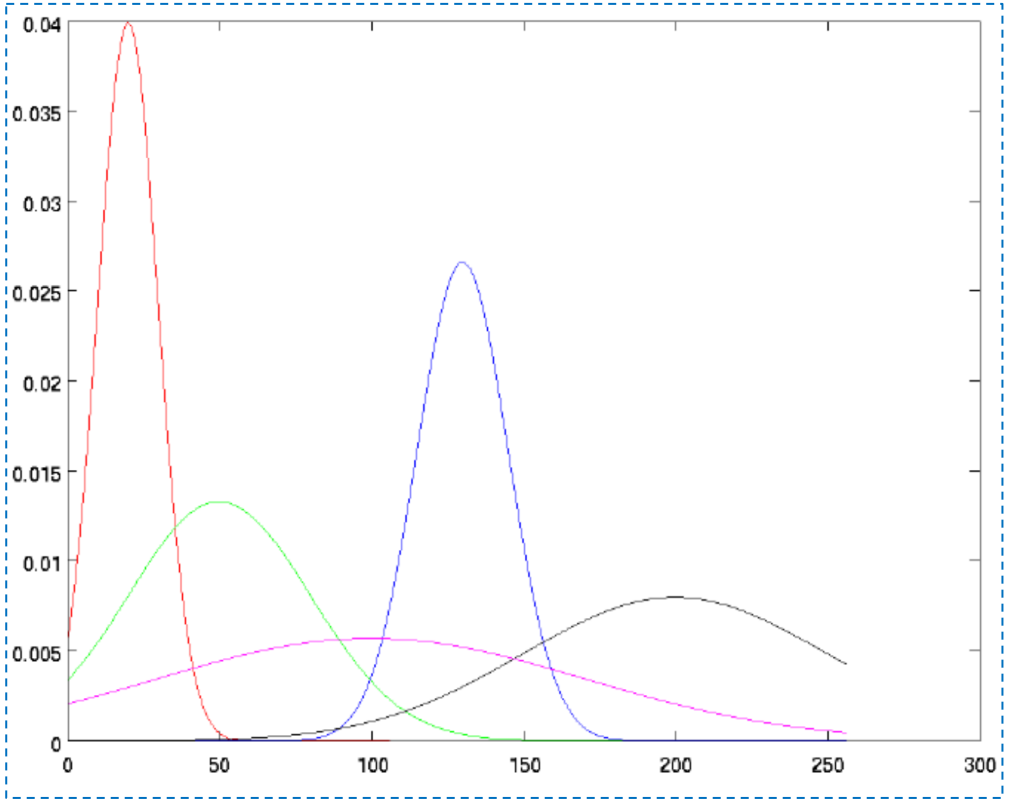
At any time t, what is known about a particular pixel (x0 , y0) is its history:

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

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

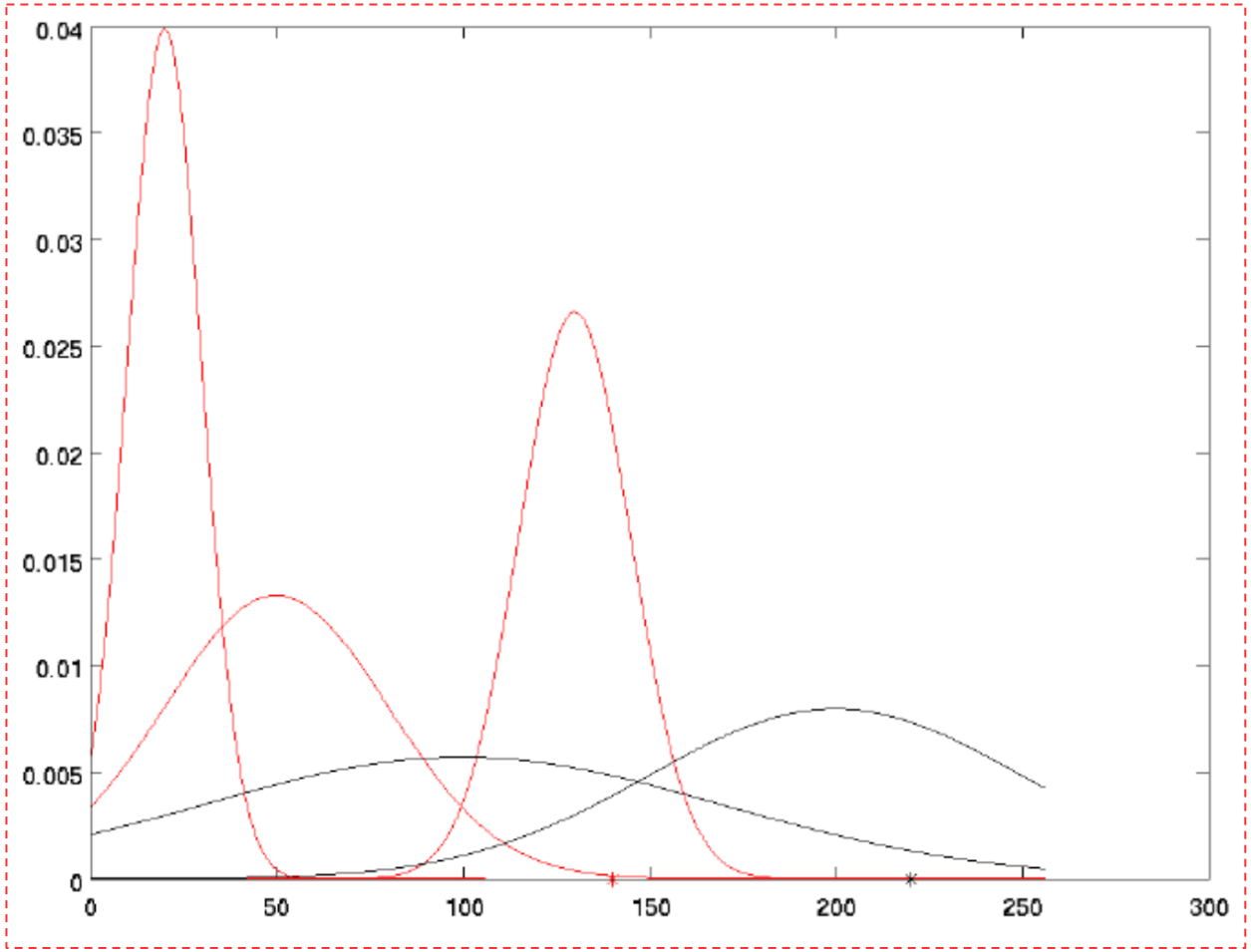
$$P(X_t) = \sum_{i=1}^{K} \omega_{it} \mathcal{N}(X_t | \mu_{it}, \Sigma_{it})$$

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

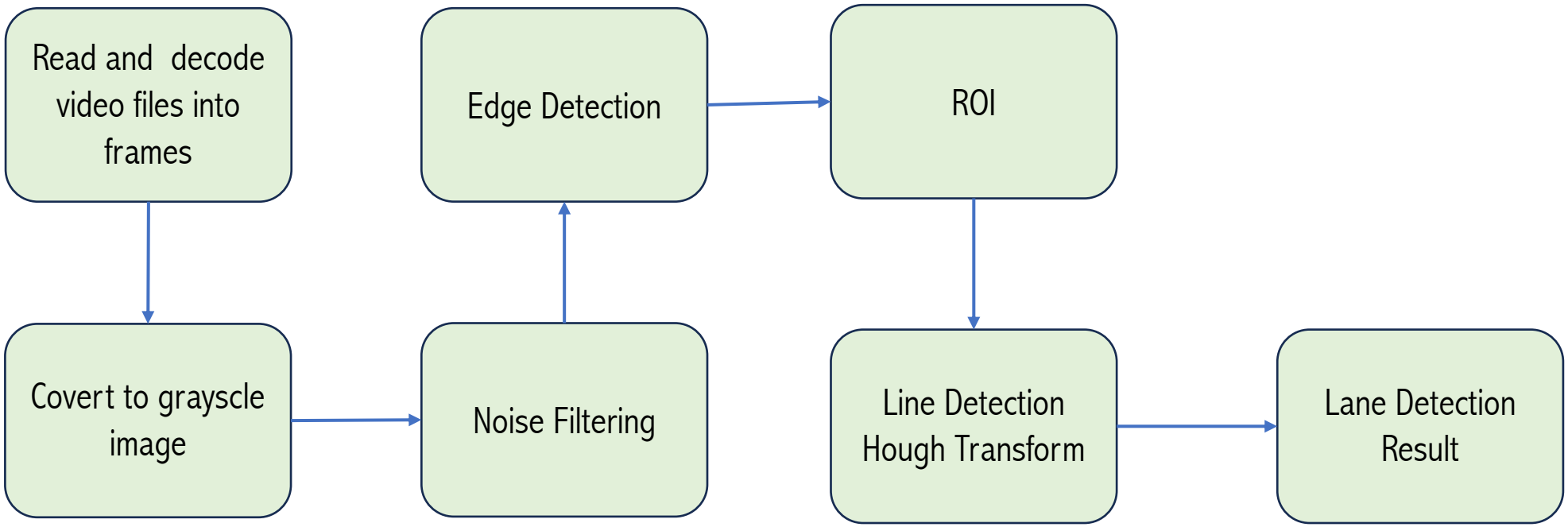


Mixture of Gaussian (MOG) Background Subtraction

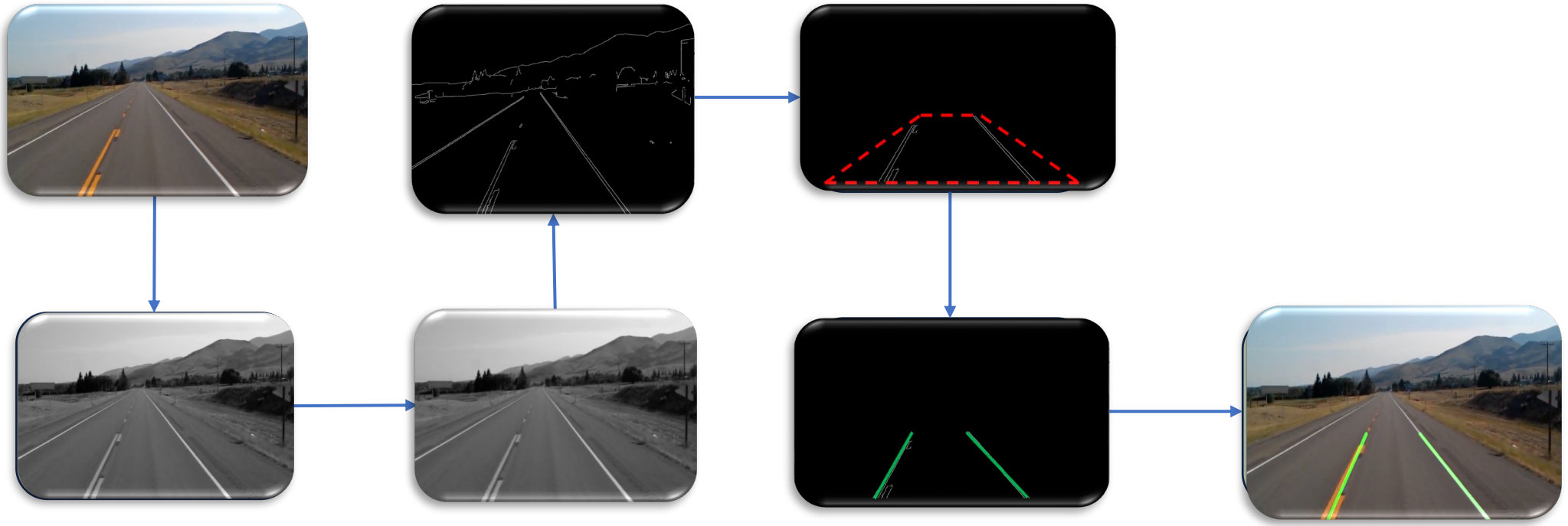
After background model estimation red distributions become the background model and black distributions are considered to be foreground



Lane Detection Work Flow



Lane Detection Work Flow



Outline

- Background Subtraction Assignment: Review
- Grayscale Conversion
- Image Blurring Techniques
- Region of Interest (ROI)
- Line Detection based on Hough Transform
- Lane Detection Demo

Color to Grayscale

Original Image

Grayscale Image

```
# convert the RGB image to Gray scale
grayscale = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

(OpenCV) ITU-R 601-2 luma transform: 0.299 R + 0.587 G + 0.114 B

Average-based method: (R + G + B) / 3

The luminosity: 0.2126R+0.7152G+0.0722B

Tại sao phải convert ảnh màu sang ảnh xám?

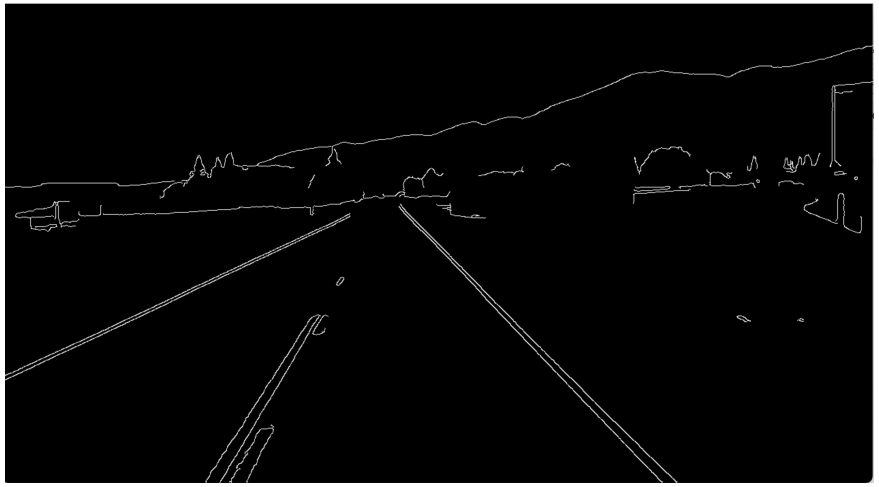
Outline

- Background Subtraction Assignment: Review
- Grayscale Conversion
- Image Blurring Techniques
- Region of Interest (ROI)
- Line Detection based on Hough Transform
- Lane Detection Demo

How Image Blurring Work

Blur Image

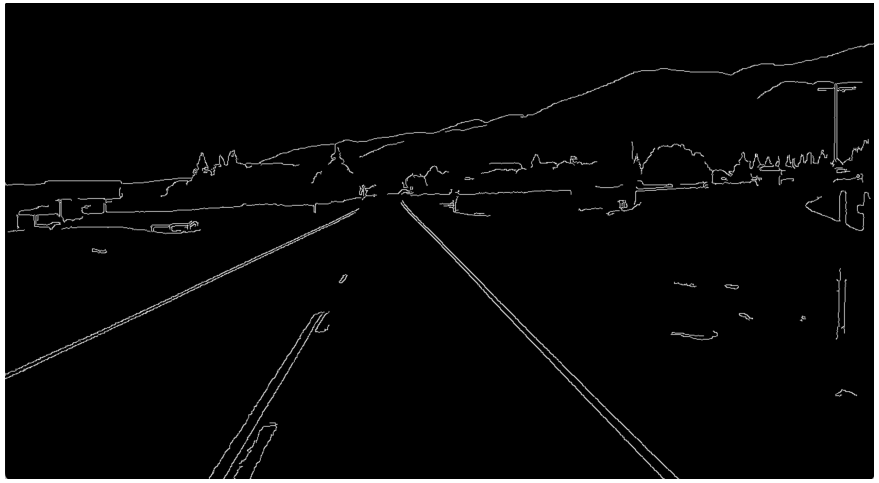
Gaussian Filter (5,5)



Edge Image

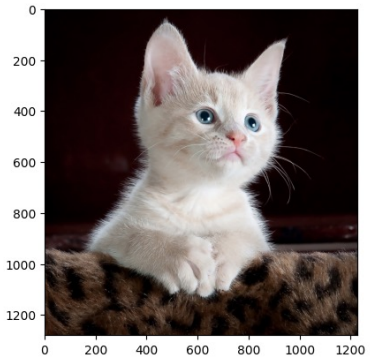
Original Image

No filter

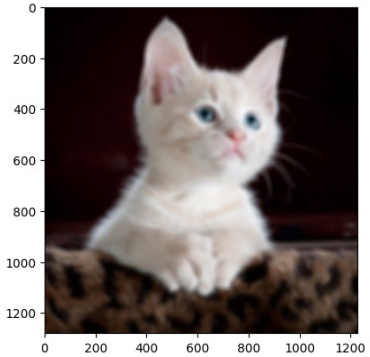


Edge Image

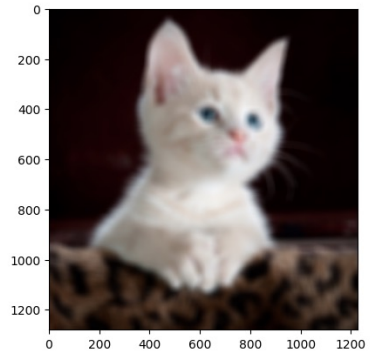
How Image Blurring Work



21x21



31x31



Blurring an image is make the image less sharp. This can be done by smoothing the color transition between the pixels.

Mean Filter (Average Filter/Box Filter)

$$K = \frac{1}{ab}\begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & & 1 \\ \vdots & & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix}$$

a = 3 and b = 3

$$\mathrm{K} = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

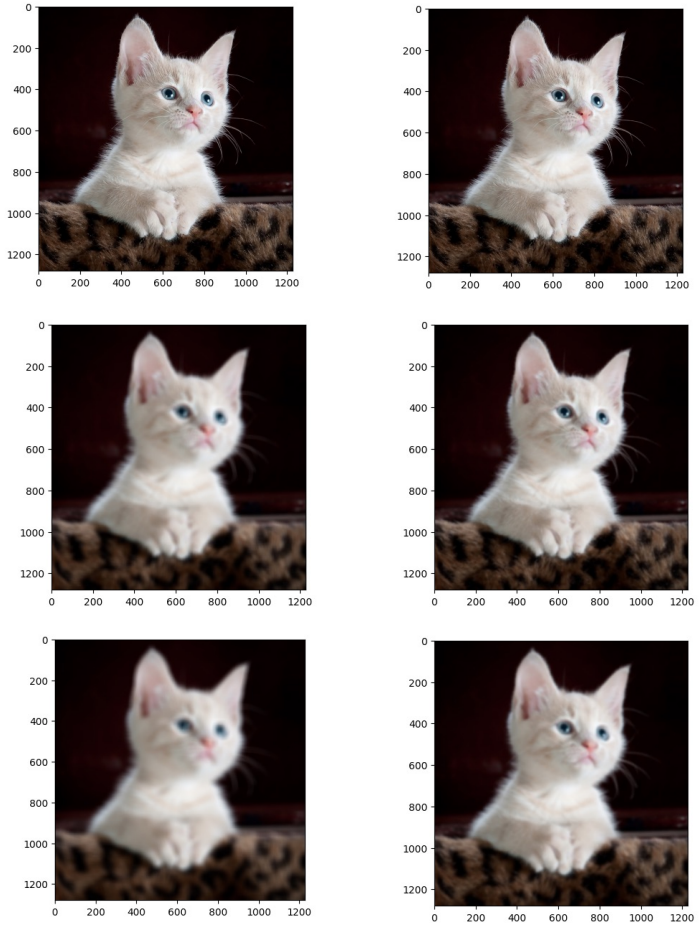
```
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.figure()
plt.imshow(img)

# Show the blurred image with different size of kernel
blurImg = cv2.blur(img,(21,21))
plt.figure()
plt.imshow(blurImg)

blurImg = cv2.blur(img,(31,31))
plt.figure()
plt.imshow(blurImg)

plt.show()
```

The greater value of kernel size, the greater _____ because the number of pixels involved is greater and the transition of colors become smoother.

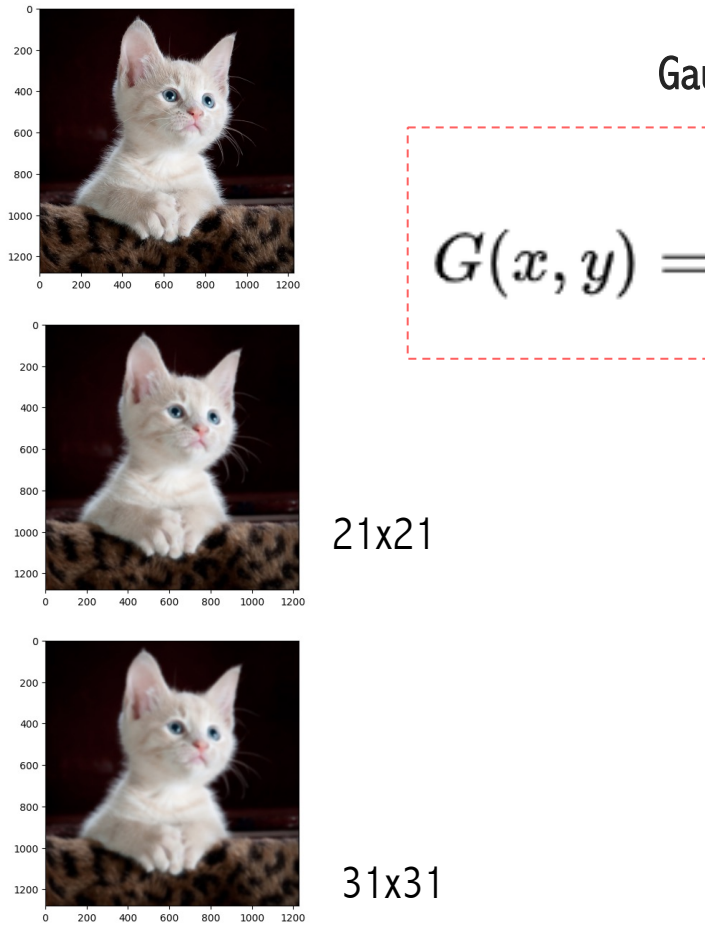
How Image Blurring Work

The Gaussian smoothing operator is a 2-D convolution operator that is used to `blur' images and remove detail and noise.

Mean Filter



Gaussian Filter



21x21

31x31

Gaussian Filter

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

Approximation of 3 x 3 Gaussian kernel, $\sigma = 1$

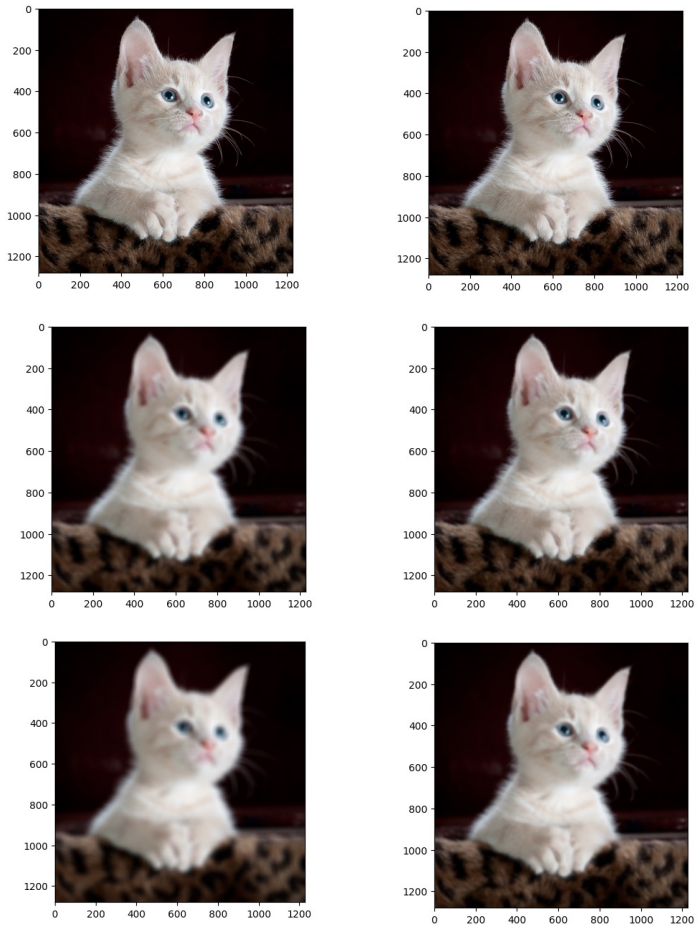
$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

```
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.figure()
plt.imshow(img)
plt.show()
# Show the blurred image with different size of kernel

# Show the blurred image with different size of kernel
blurImg = cv2.GaussianBlur(img,(21,21), 0)
plt.figure()
plt.imshow(blurImg)

blurImg = cv2.GaussianBlur(img,(31,31), 0)
plt.figure()
plt.imshow(blurImg)

plt.show()
```

How Image Blurring Work

The Gaussian smoothing operator is a 2-D convolution operator that is used to `blur' images and remove detail and noise.

Mean Filter | Gaussian Filter



21x21

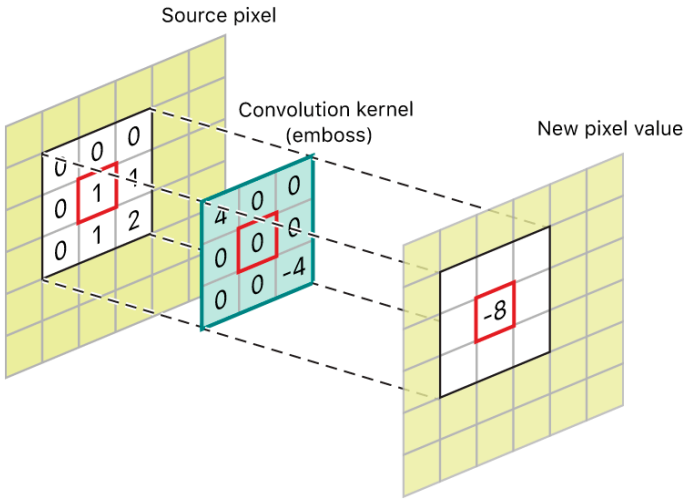
31x31

Gaussian Filter

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

Approximation of 3 x 3 Gaussian kernel, $\sigma = 1$

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

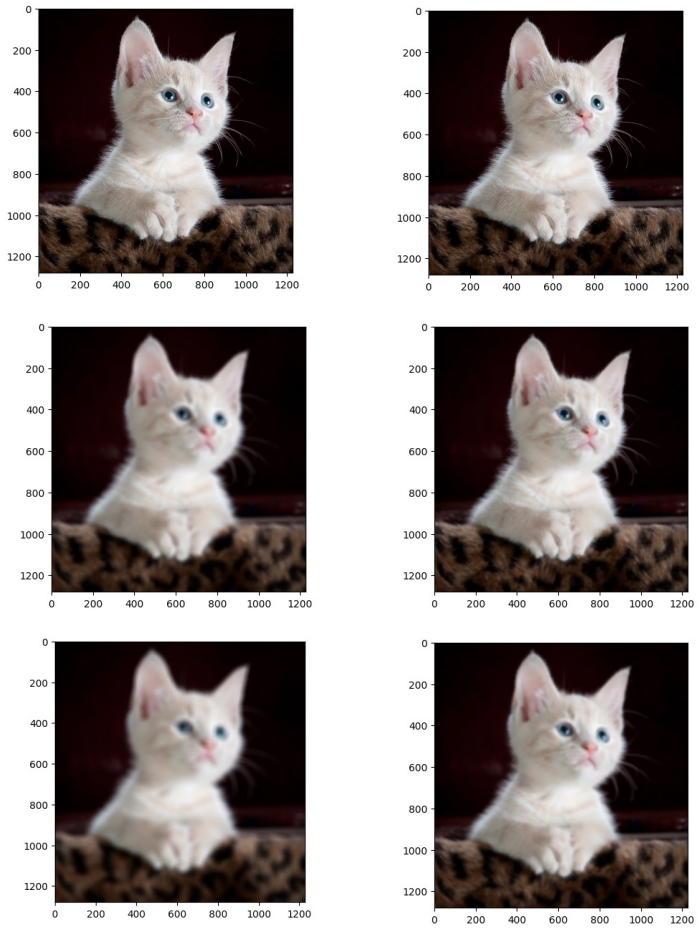


By changing the values in the kernel, we can change the effect on the image — blurring, sharpening, edge detection, noise reduction, etc

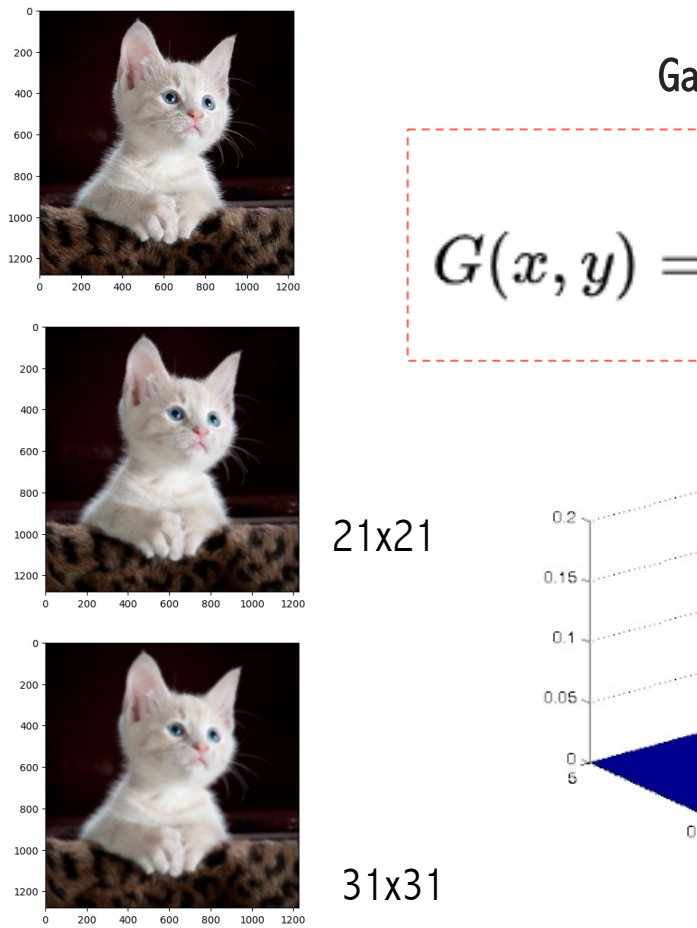
How Image Blurring Work

The Gaussian smoothing operator is a 2-D convolution operator that is used to `blur' images and remove detail and noise.

Mean Filter



Gaussian Filter



21x21

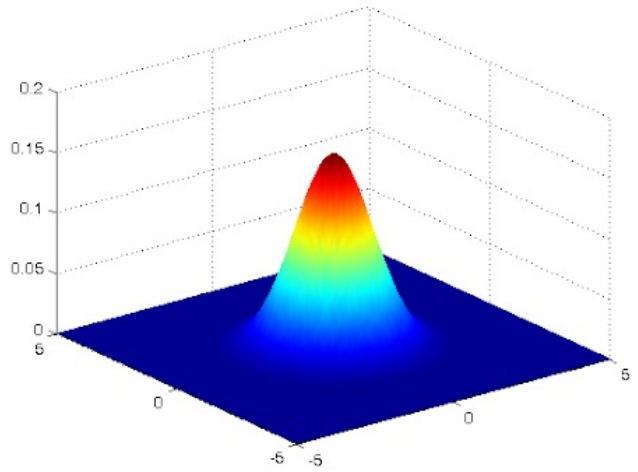
31x31

Gaussian Filter

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

Approximation of 3 x 3 Gaussian kernel, $\sigma = 1$

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



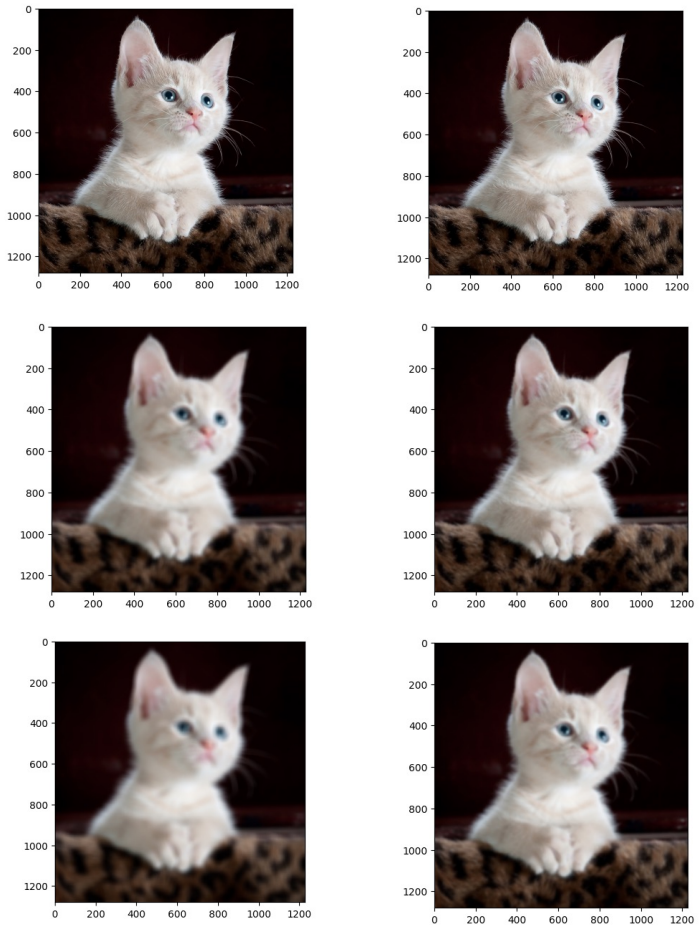
A Gaussian blur is applied by convolving the image with a Gaussian function.

Weighted average of the pixel's values and the height of the curve at that point, the pixels in the center of the group would contribute most significantly to the resulting value

How Image Blurring Work

The Gaussian smoothing operator is a 2-D convolution operator that is used to `blur' images and remove detail and noise.

Mean Filter



Gaussian Filter

21x21

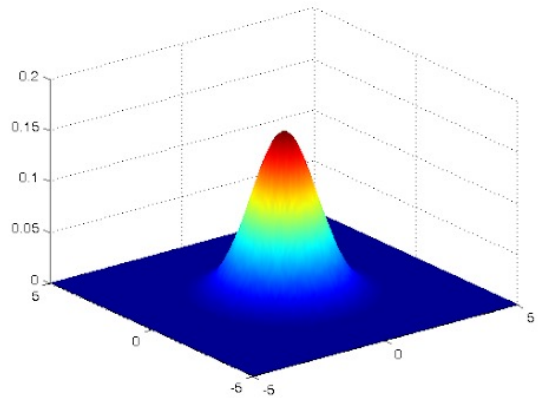
31x31

Gaussian Filter

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

Approximation of 3 x 3 Gaussian kernel, $\sigma = 1$

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



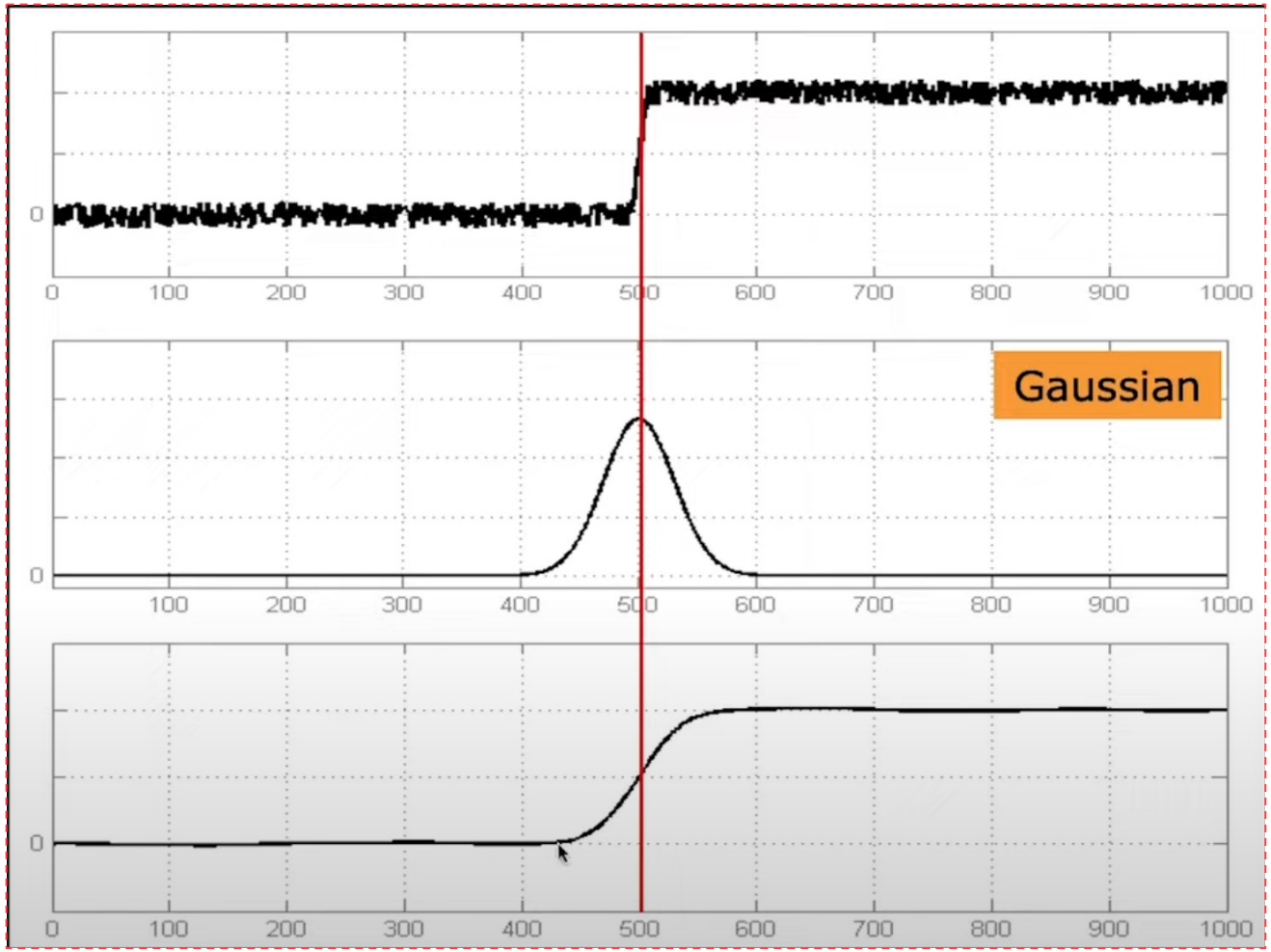
```
for x in range(-radius, radius+1, 1):
    for y in range(-radius, radius+1, 1):
        exponentNumerator = float(-(x * x + y * y))
        exponentDenominator = (2 * sigma * sigma)

        eExpression = np.exp(exponentNumerator / exponentDenominator)
        kernelValue = (eExpression / (2 * pi * sigma * sigma))

        #We add radius to the indices to prevent out of bound issues
        #because x and y can be negative
        kernel[x + radius][y + radius] = kernelValue
        sum += kernelValue

for x in range(-radius, radius+1, 1):
    for y in range(-radius, radius+1, 1):
        kernel[x + radius][y + radius] = kernel[x + radius][y + radius]/sum
```

Gaussian Filter

f

n_σ

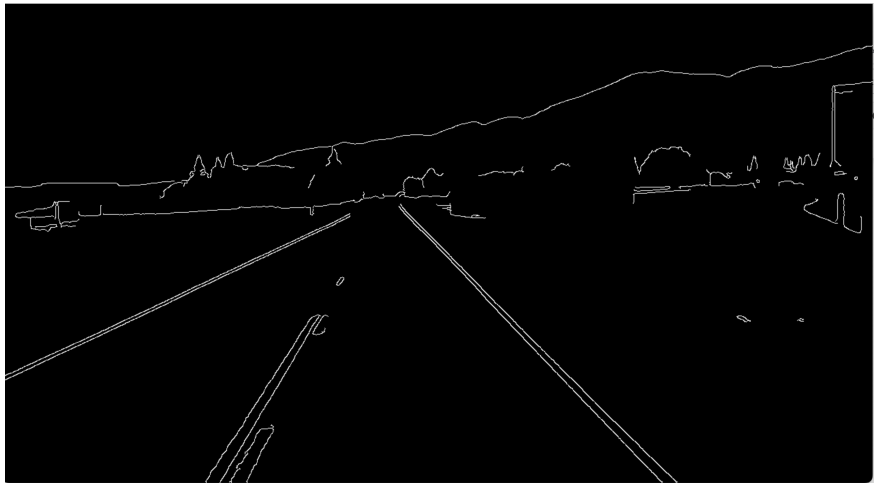
n_σ*f



How Image Blurring Work

Blur Image

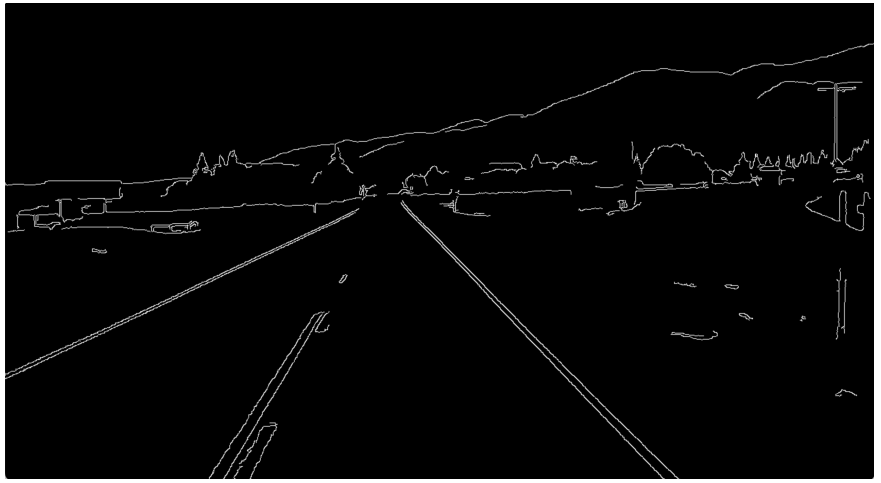
Gaussian Filter (5,5)



Edge Image

Original Image

No filter

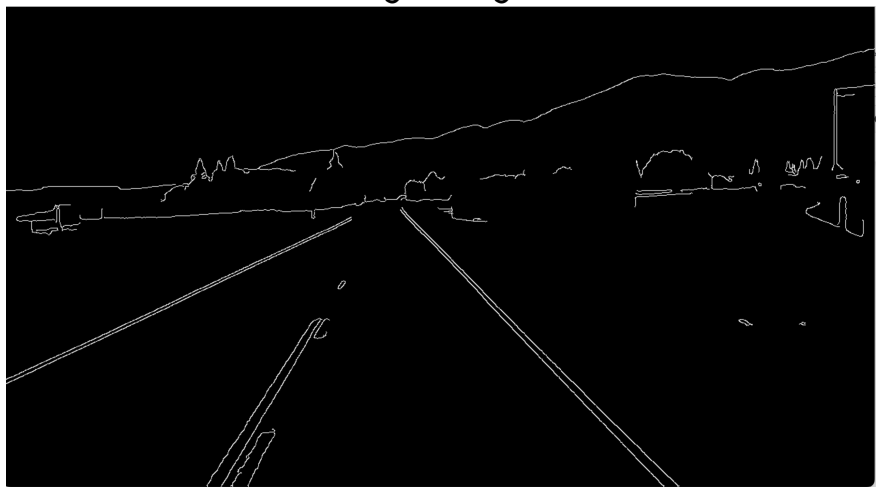


Edge Image

Edge Detection

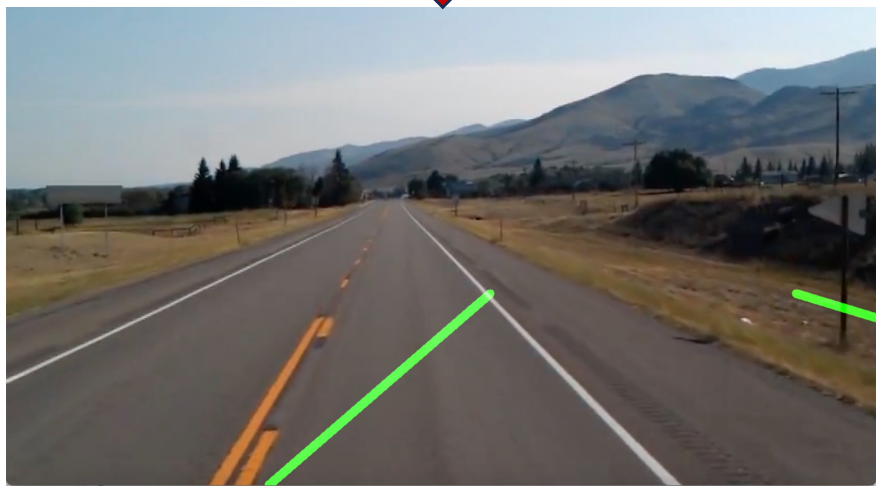
Blur Image

Edge Image



Tại sao kết quả line detection lại không chính xác?

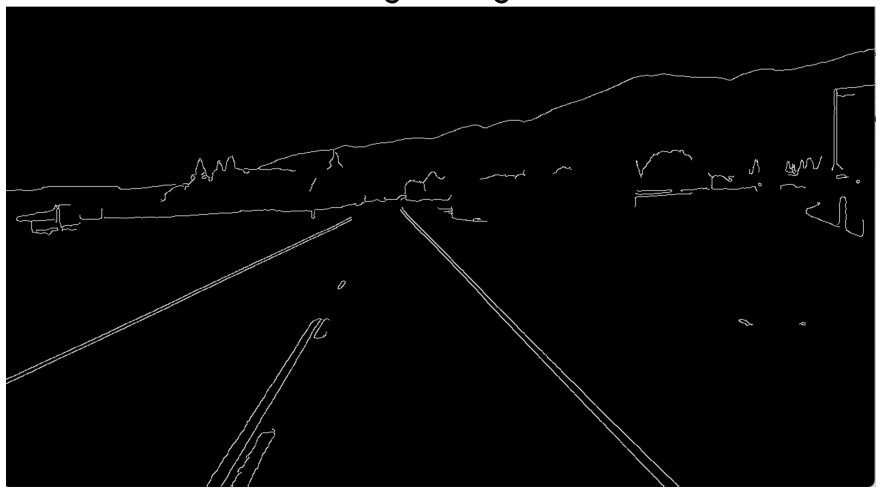
Lane Detection Results

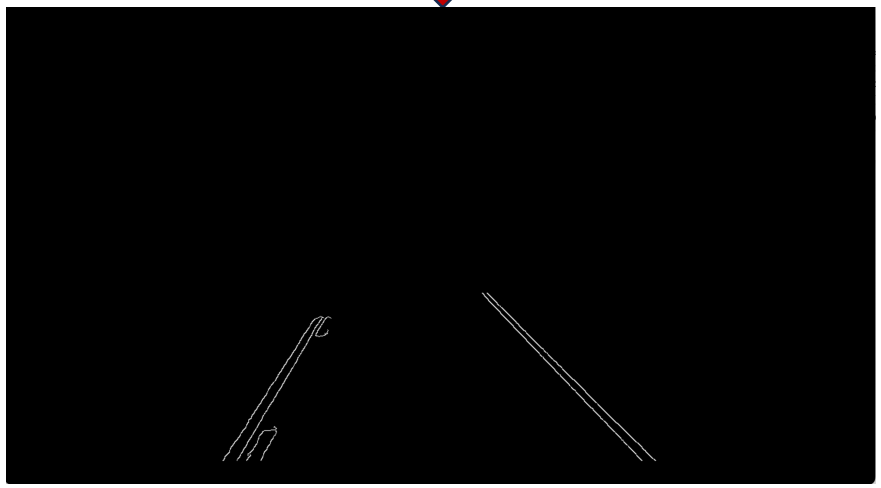


Edge Detection

Blur Image

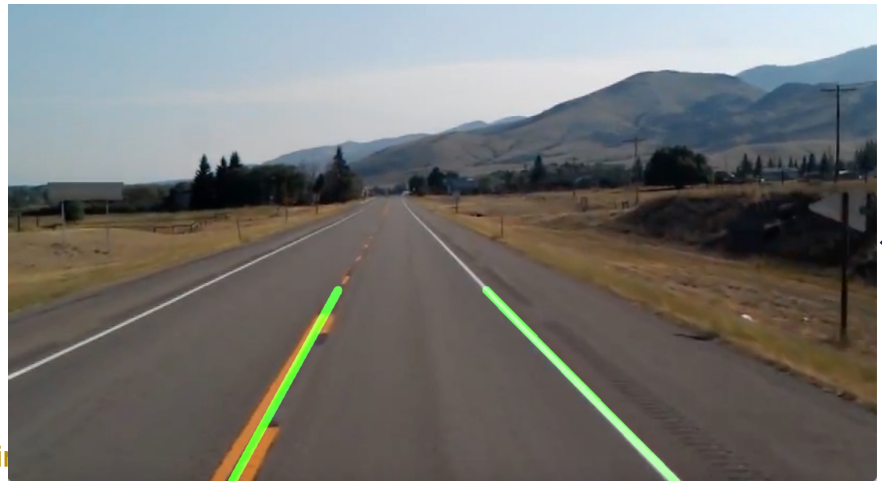
Edge Image





Ego-lane

Lane Detection Results



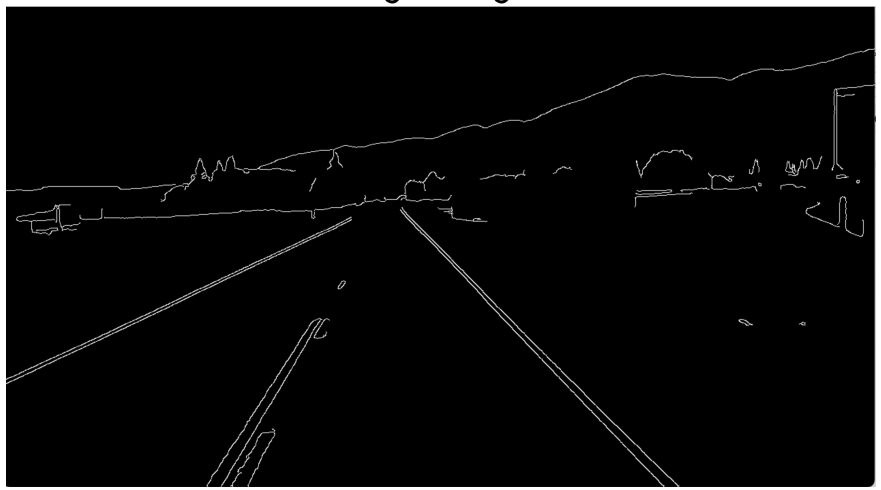
Outline

- Background Subtraction Assignment: Review
- Grayscale Conversion
- Blurring Techniques
- Region of Interest (ROI)
- Line Detection based on Hough Transform
- Lane Detection Demo

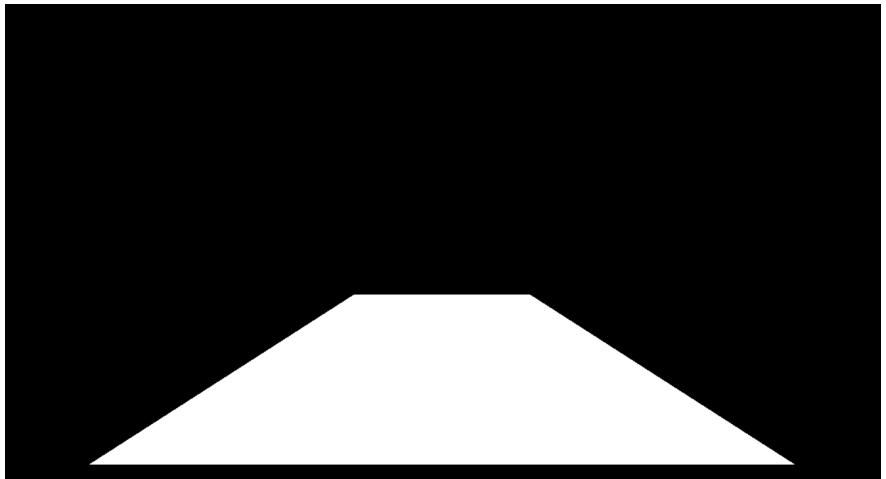
Region of Interest

Blur Image

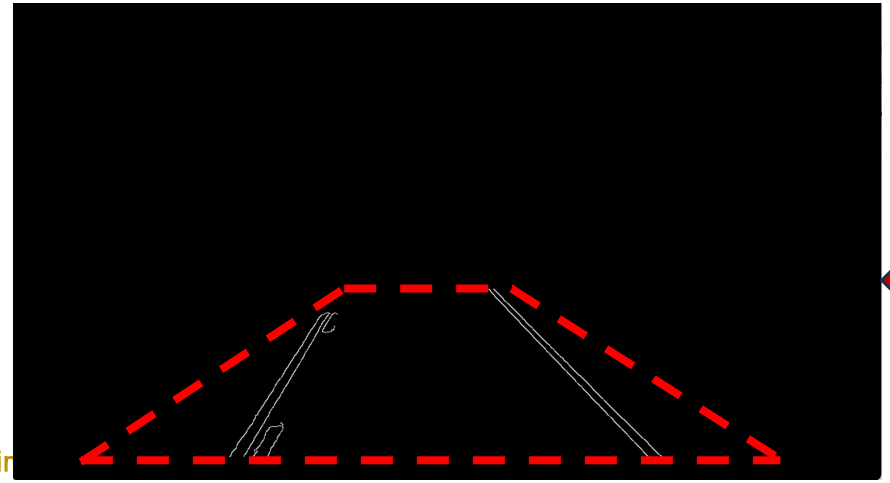
Edge Image



Ego-lane



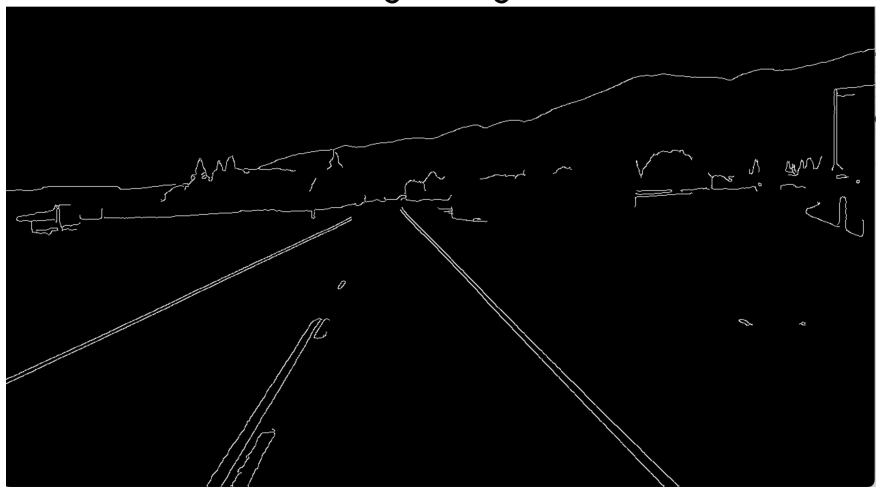
Ego-lane



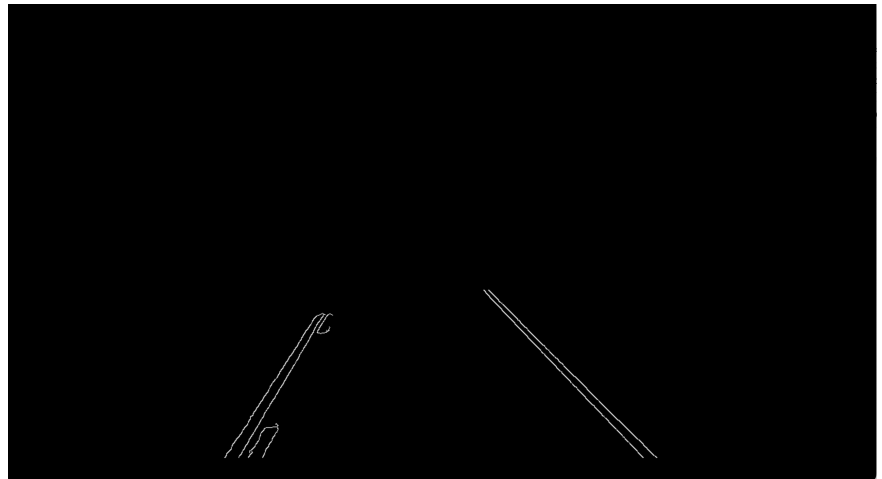
Region of Interest

Blur Image

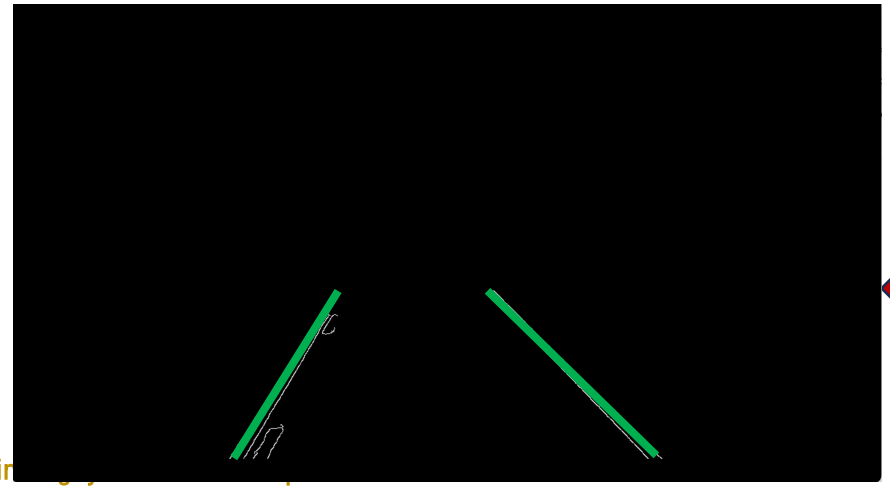
Edge Image



Ego-lane



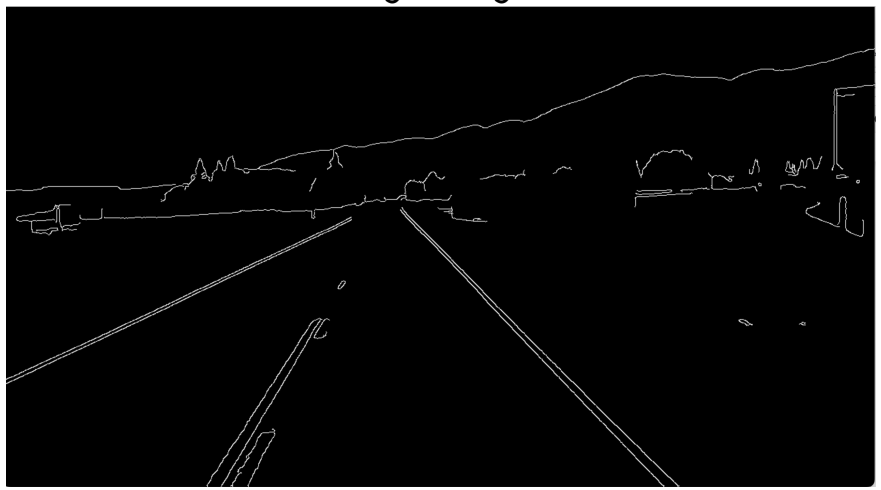
Line Detections



Region of Interest

Blur Image

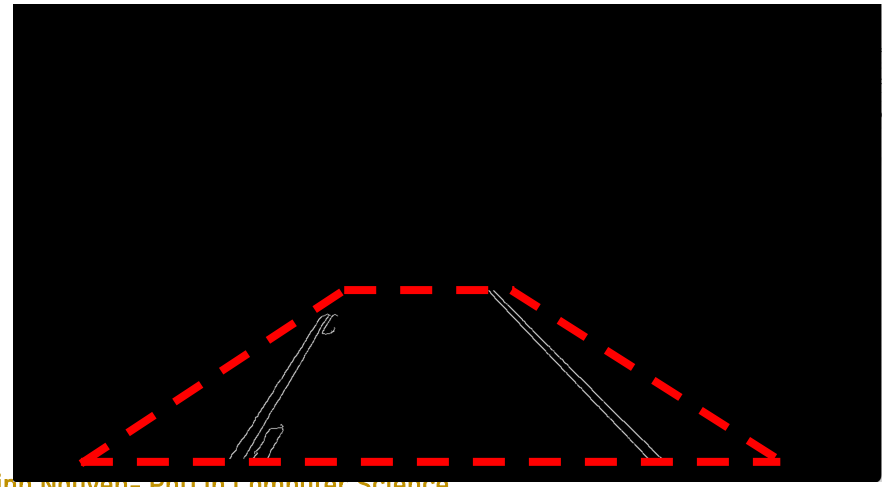
Edge Image



Region of Interest

```python
def region_selection(image):
    mask = np.zeros_like(image)
    ignore_mask_color = 255
    # creating a polygon to focus only on the road in the picture
    # we have created this polygon in accordance to how the camera was pla
    rows, cols = image.shape[:2]
    bottom_left = [cols * 0.1, rows * 0.95]
    top_left    = [cols * 0.4, rows * 0.6]
    bottom_right = [cols * 0.9, rows * 0.95]
    top_right = [cols * 0.6, rows * 0.6]
    vertices = np.array([[bottom_left, top_left, top_right, bottom_right]]
    # filling the polygon with white color and generating the final mask
    cv2.fillPoly(mask, vertices, ignore_mask_color)
    # performing Bitwise AND on the input image and mask to get only the e
    masked_image = cv2.bitwise_and(image, mask)
    return masked_image
```

Ego-lane



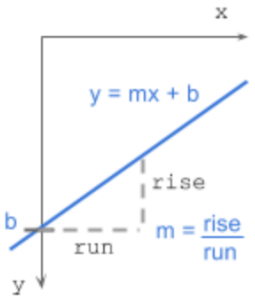
Outline

- Background Subtraction Assignment: Review
- Grayscale Conversion
- Blurring Techniques
- Region of Interest (ROI)
- Line Detection based on Hough Transform
- Lane Detection Demo

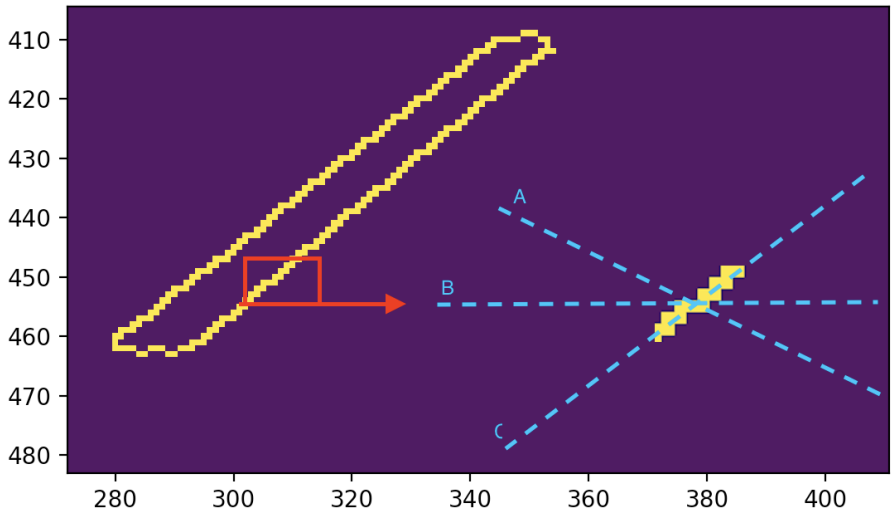
Line Detection Clearly Explain

A line in Cartesian form has the equation:

```
y = mx + b

where:
m = gradient or slope of the line (rise/run)
b = y-intercept
```

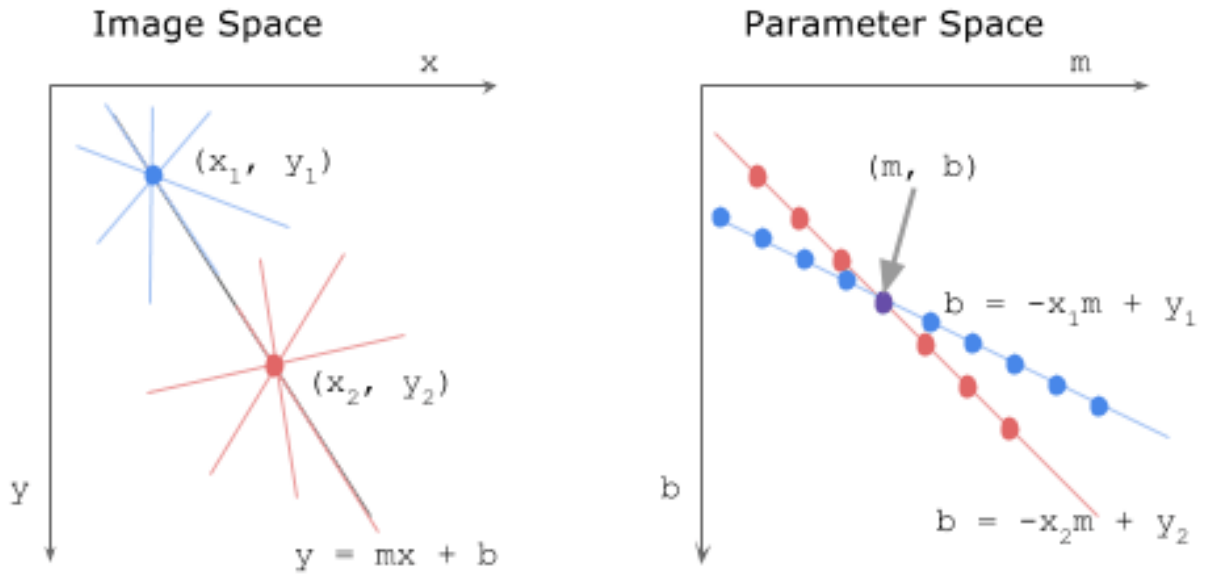


Given a set of edge points or a binary image indicating edges, we want to find as many lines that connect these points in image space.



Line Detection Clearly Explain

Say we have 2 edge points (x1,y1) and (x2,y2). For each edge point at various gradient values (m=-0.5, 1.0, 1.5, etc.), we calculate the corresponding b values. The image below shows the various lines through an edge point in image space and the plot of these lines in parameter space. Points which are collinear in the cartesian image space will intersect at a point in (m,b) parameter space.



Unfortunately, the slope, m, is undefined when the line is vertical (division by 0!). To overcome this, we use another parameter space, the Hough space.

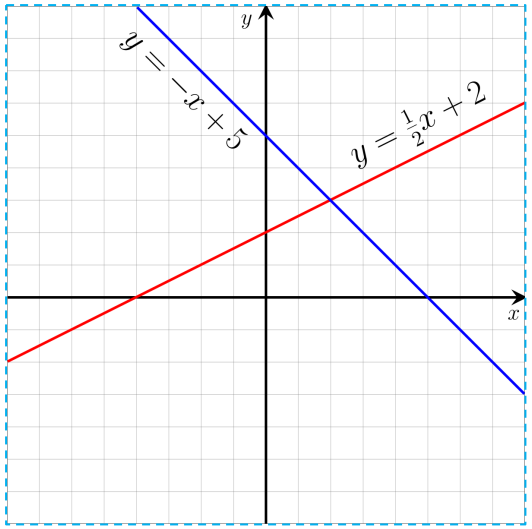
How it works - angle-distance parameter space

All points on a line in image space intersect at a common point in parameter space. This common point (m, b) represents the line in image space.

Line Detection Clearly Explain

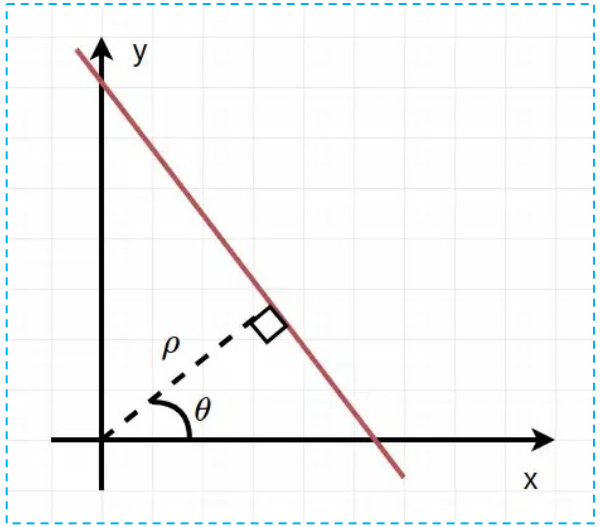
Phương trình đường thẳng cơ bản:

$$y = ax + b$$



Phương trình đường thẳng theo toạ đô cực:

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

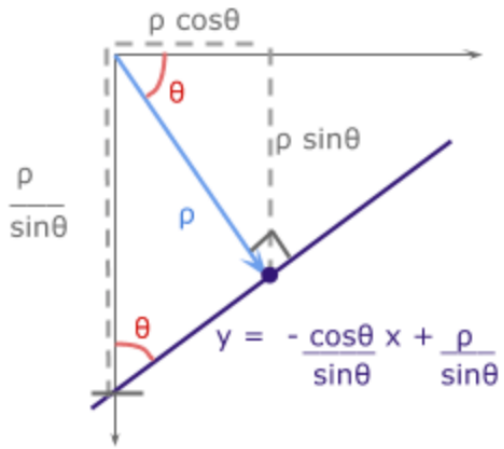


Ý tưởng chung của việc phát hiện đường thẳng trong thuật toán này là tạo mapping từ không gian ảnh (A) sang một không gian mới (B) mà mỗi đường thẳng trong không gian (A) sẽ ứng với một điểm trong không gian (B).

```
ρ = x cos θ + y sin θ

where:
ρ (rho) = distance from origin to the line. [-max_dist to max_dist].
          max_dist is the diagonal length of the image.
θ = angle from origin to the line. [-90° to 90°]
```

Line Detection Clearly Explain

Deriving rho: ρ = x cos θ + y sin θ



With basic trigonometry, we know that for right-angled triangles,

$\sin\theta = \frac{\text{opposite}}{\text{hypotenuse}}$ and $\cos\theta = \frac{\text{adjacent}}{\text{hypotenuse}}$.

We want to convert the cartesian form $y = mx + b$ with parameters (m, b) to polar form with parameters (ϱ, θ).

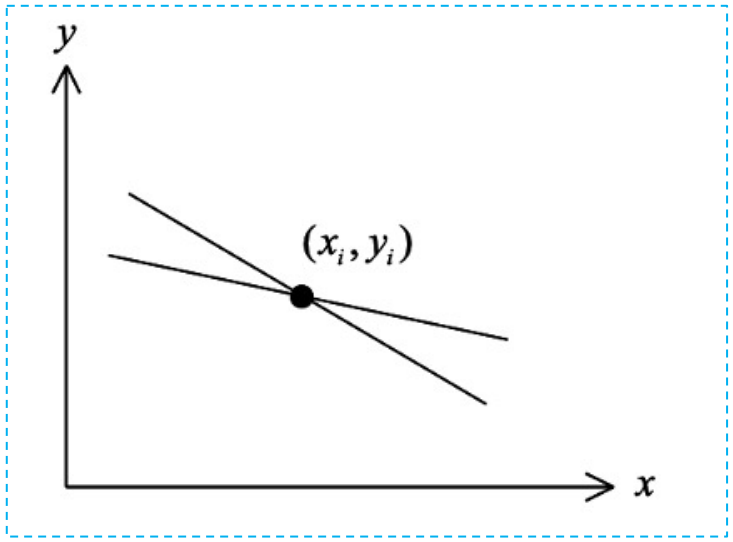
The line from the origin with distance ϱ has a gradient of $\frac{\sin\theta}{\cos\theta}$. The line of interest, which is perpendicular to it, will have a negative reciprocal gradient value of $\frac{-\cos\theta}{\sin\theta}$. For b, the y-intercept of the line of interest, $\sin\theta = \frac{\varrho}{b}$.

With $m = \frac{-\cos\theta}{\sin\theta}$ and $b = \frac{\varrho}{\sin\theta}$, we get $\varrho = x\cos\theta + y\sin\theta$.

Line Detection Clearly Explain

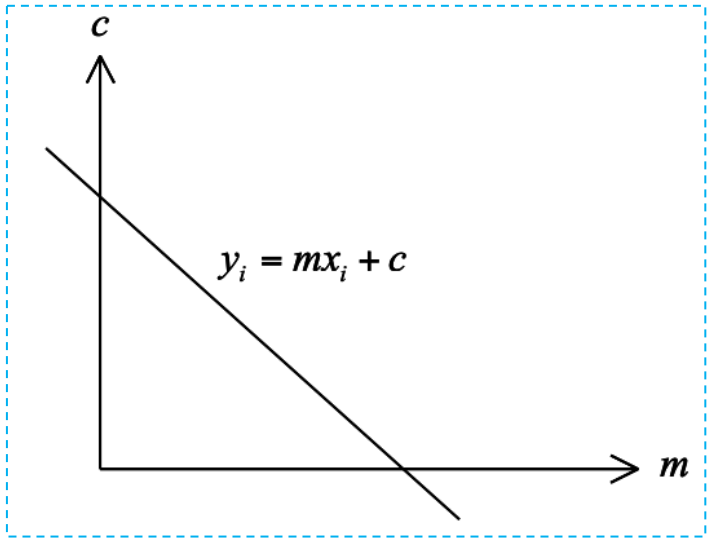
Phương trình đường thẳng cơ bản:

$$y_i = mx_i + c$$



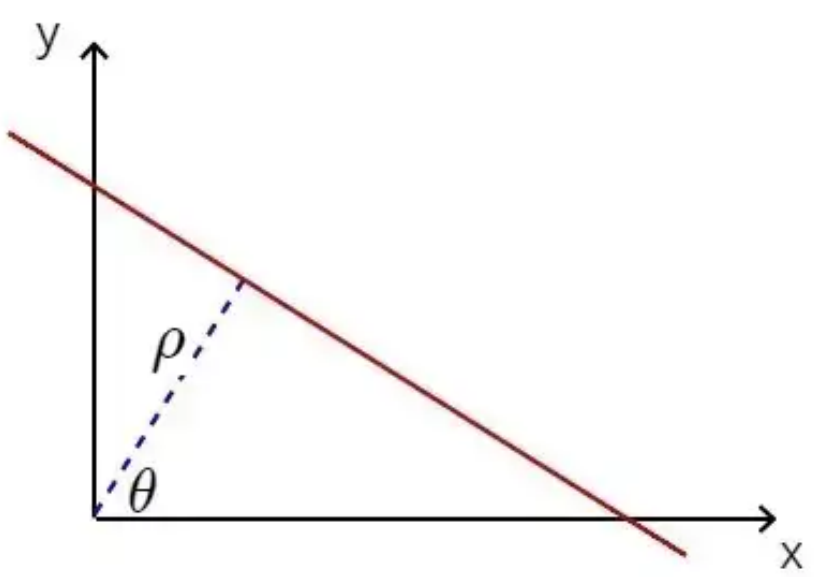
Lines through a point in the Cartesian domain

$$c = y_i - mx_i$$

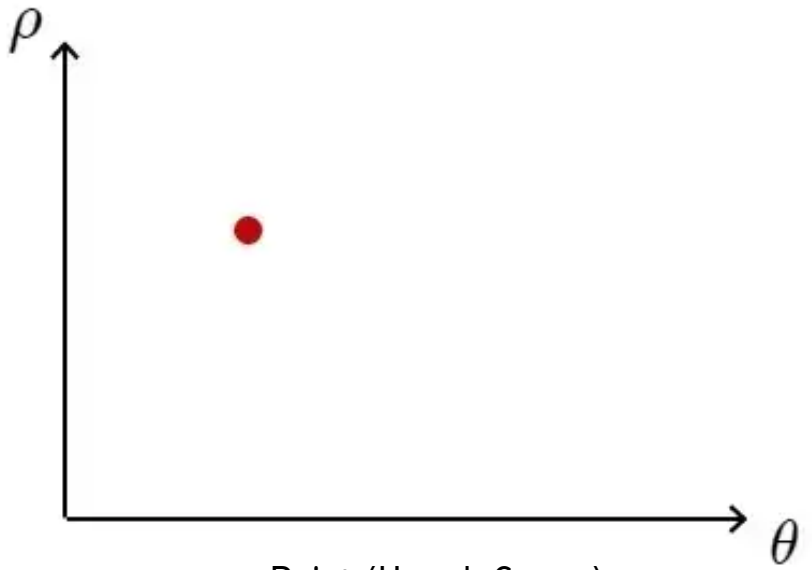


At this point, it is easy to see that each different line through the point (xi,yi) corresponds to one of the points on the line in the (m,c) space.

Line in Image Space to Point in Hough Space

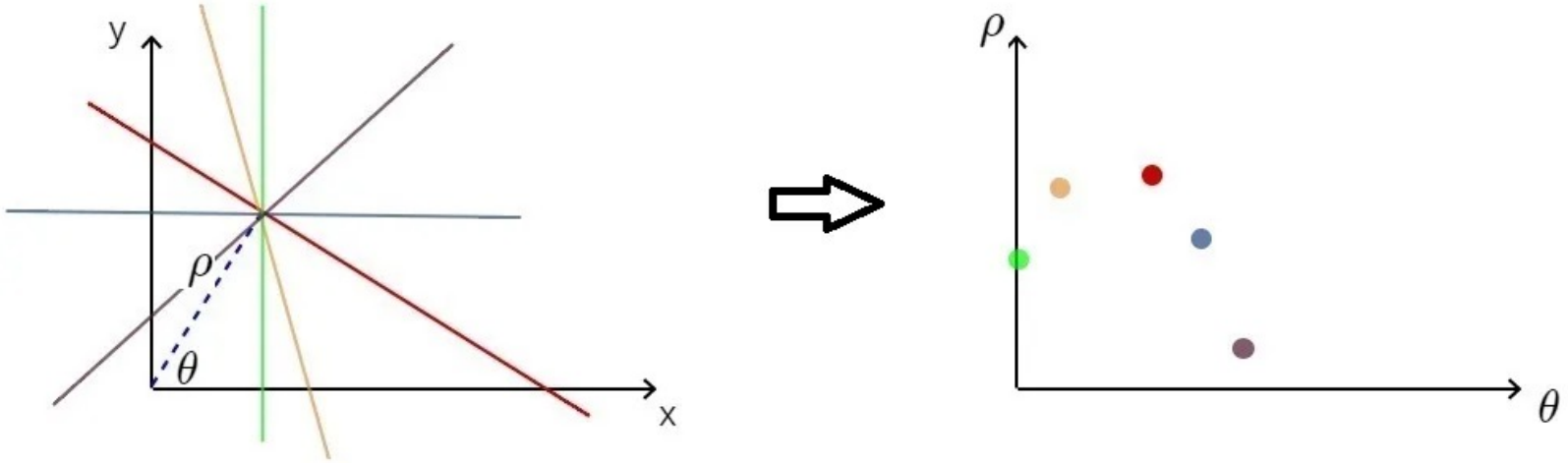


Straight Line: Image Space



Point (Hough Space)

Mapping from Image space to Hough space

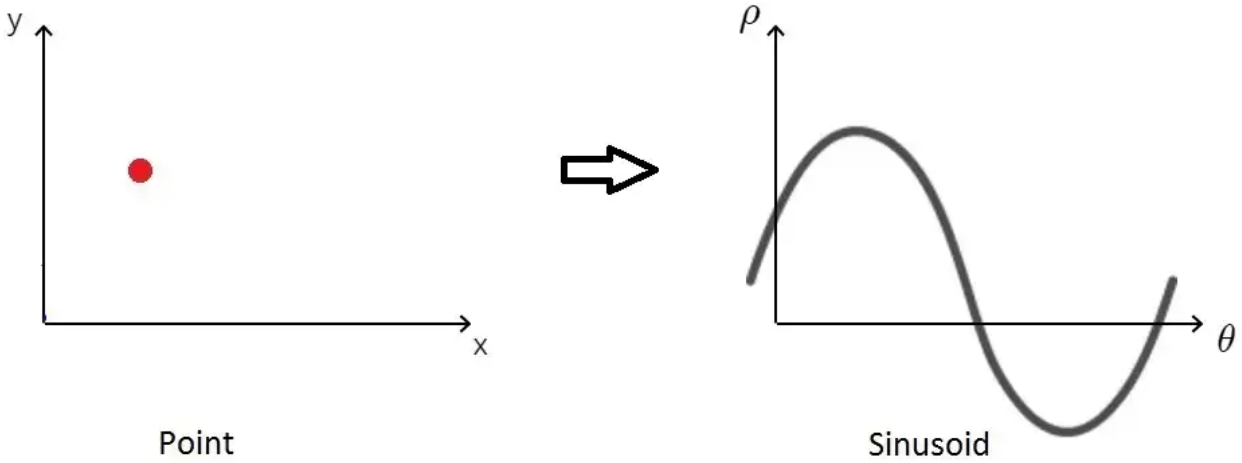


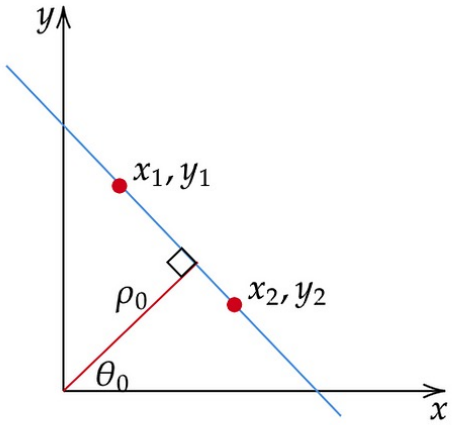
Bunch of lines intersecting at one point

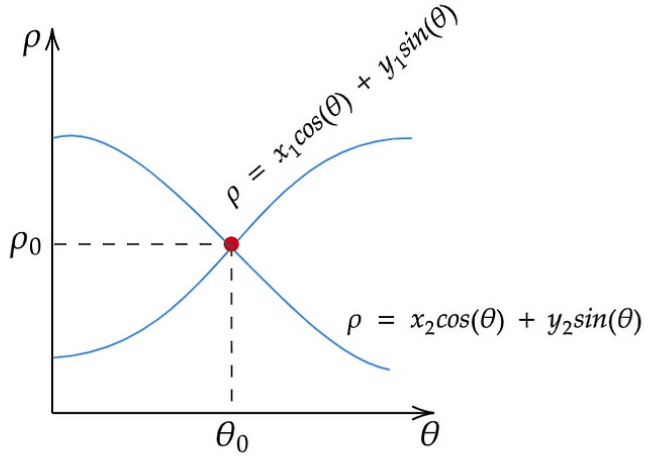
Points which form a sinusoid

It turns out that these points in *(ρ, ϑ)* space are forming a sinusoid

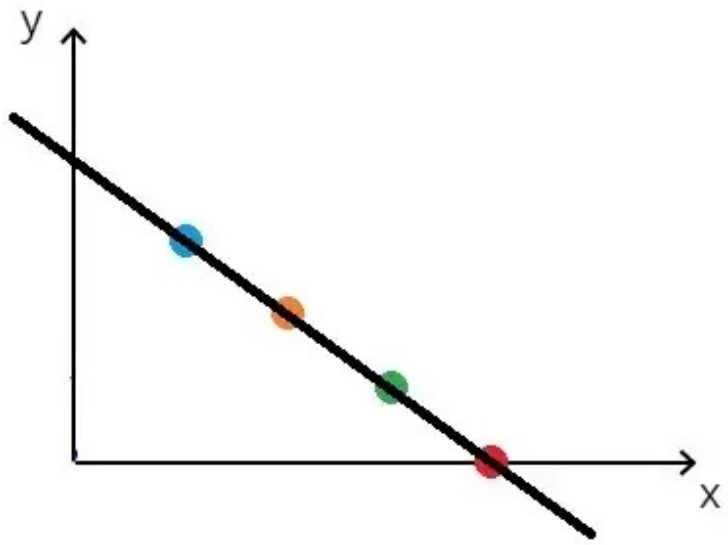
Point in Image Space to Sine in Hough Space



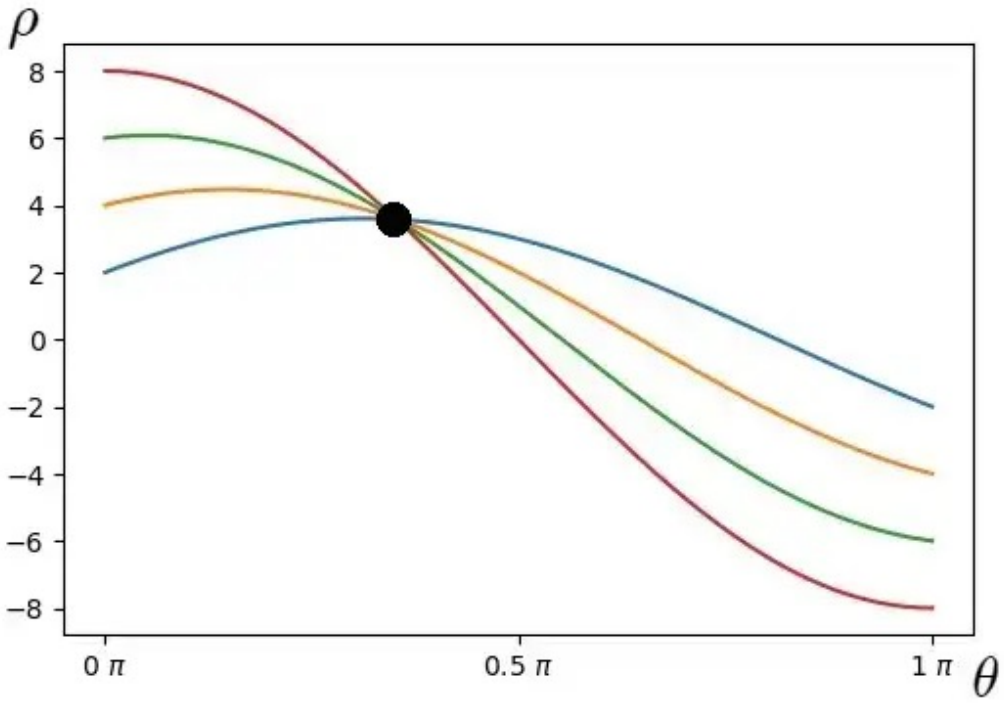




Point in Image Space to Sine in Hough Space

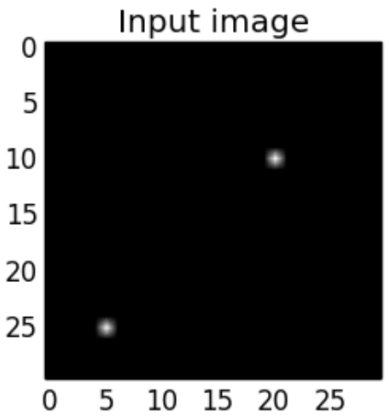


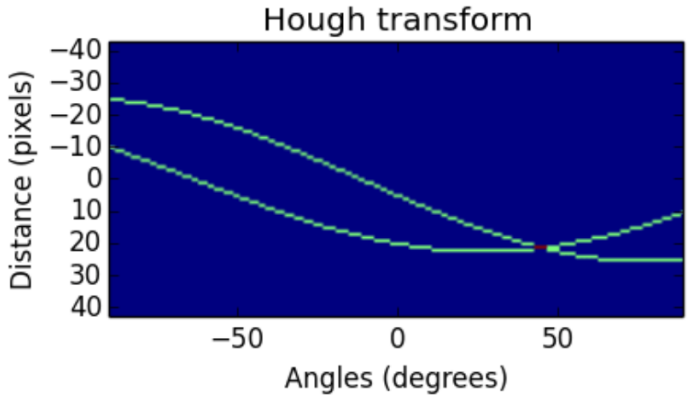
Points which form a line



Bunch of sinusoids intersecting at one point

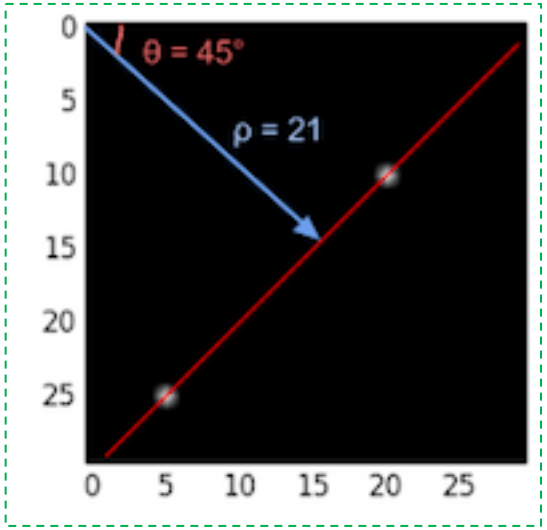
Point in Image Space to Sine in Hough Space





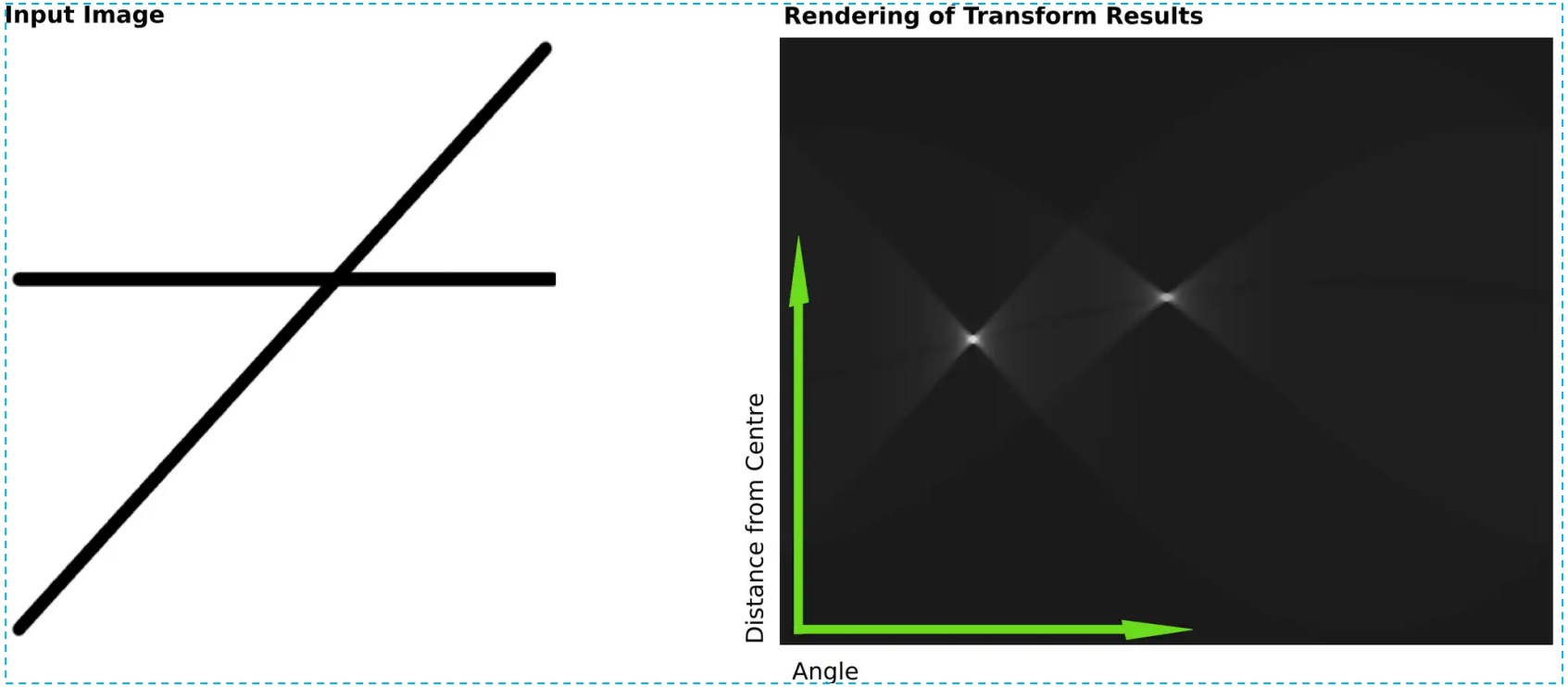
The value of ϱ at various angles for the 2 edge points:

point	-90°	-45°	0°	45°	90°
(5, 25)	-25	-14	5	21	25
(20, 10)	-10	7	20	21	10



To explain the hough transform, I'll use a simple example. I've created an input binary image of size 30 x 30 pixels with points at (5, 25) and (20, 10) shown below. The image is transformed to the hough space by calculating ϱ with a point at each angle from -90° to 90° (negative angles are anti-clockwise starting horizontally from the origin and positive angles are clockwise). The points in hough space make a sinusoidal curve.

Mỗi đường thẳng khác nhau sẽ tạo thành một điểm sáng (nơi giao nhau của nhiều hình sin) trên không gian Hough. Dưới đây là sự biểu diễn 2 đường thẳng trong không gian Hough.



Summary in Line Detection

To sum up, we observed following relations between **Image space** and **Hough space:**

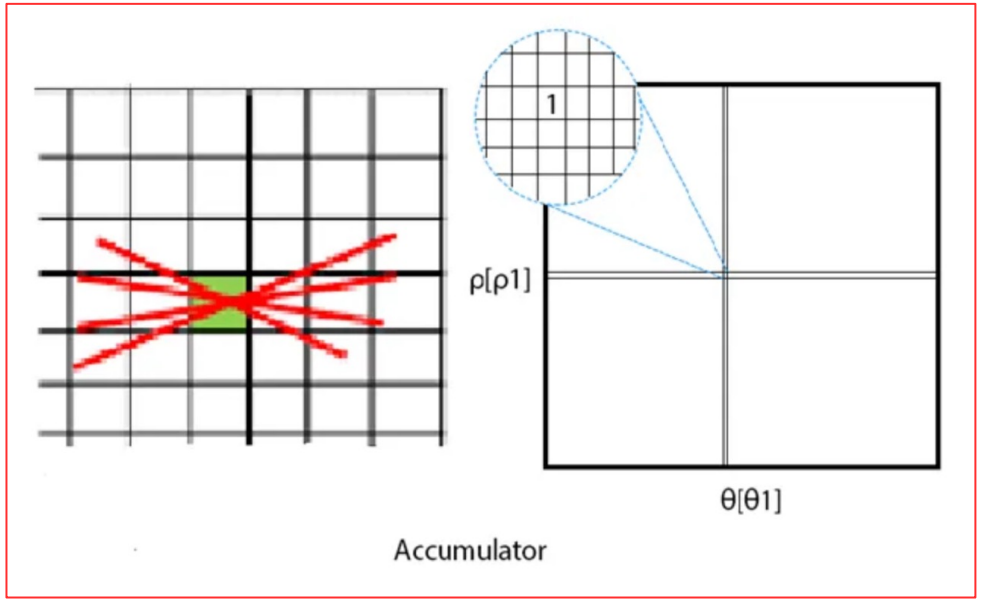
Straight line ⟶ Point

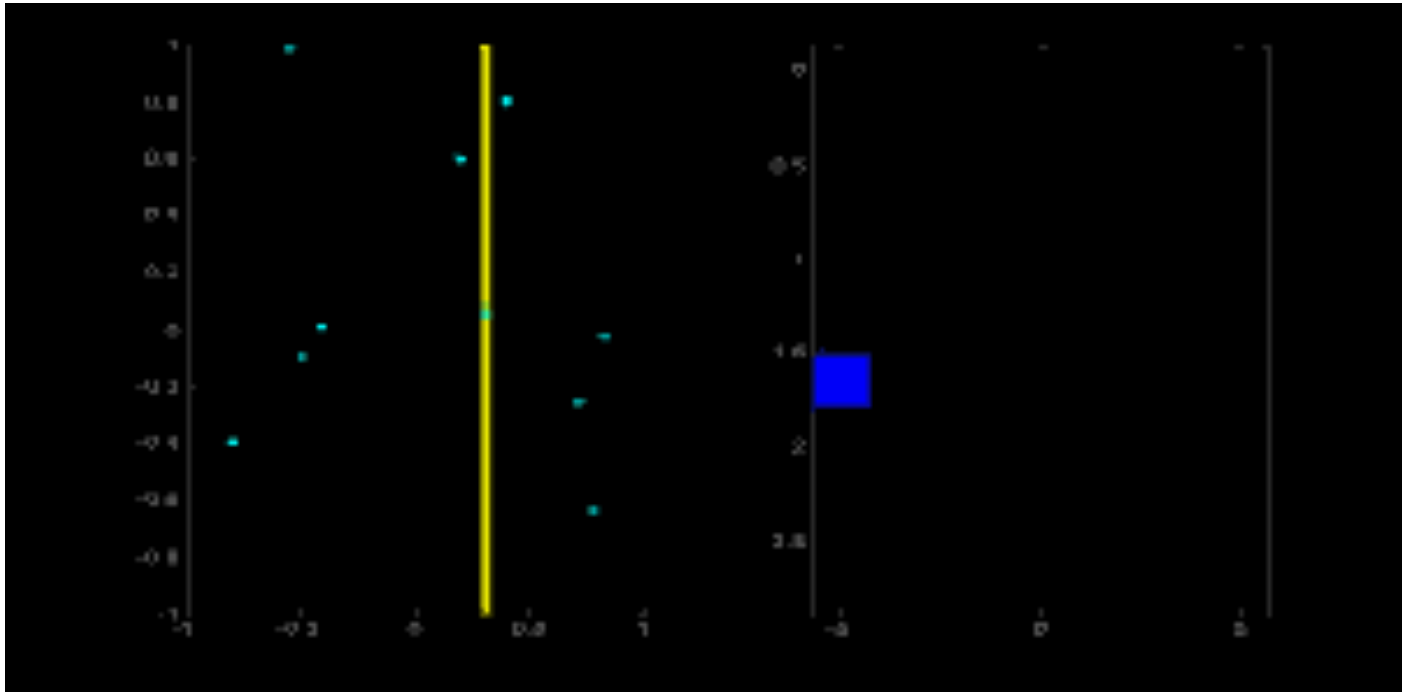
Point ⟶ Sinusoid

Multiple points on Straight line ⟶ Multiple sinusoids intersecting at a point

Multiple lines ⟶ Multiple sinusoid with multiple points of intersection

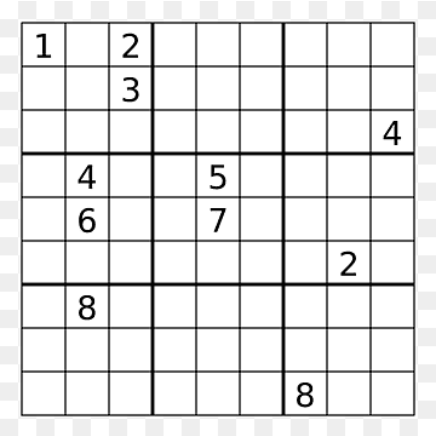
Lane Detection Clearly Explain



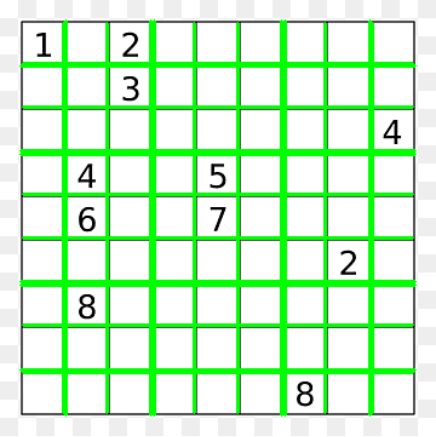


http://homepages.inf.ed.ac.uk/amos/hough.html

Line Detection: Hough Transform



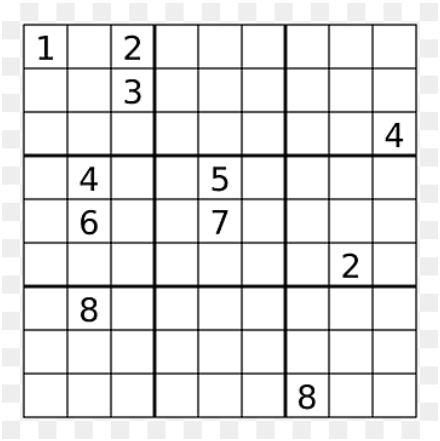
Input Image



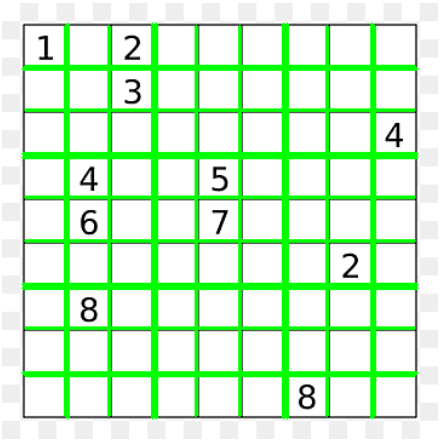
Line Detection Result

```
# 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



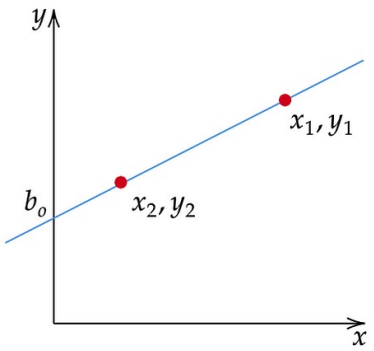
Input Image

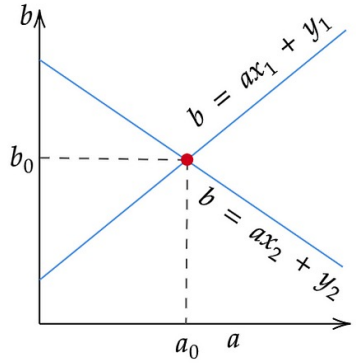


Line Detection Result

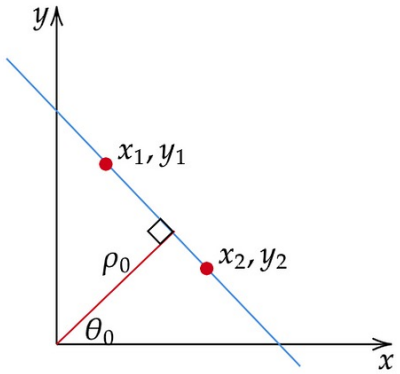
Line in the image space can be expressed with two variables

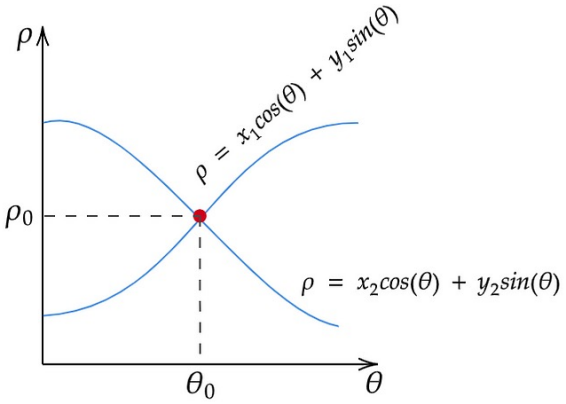
$y = ax + b$



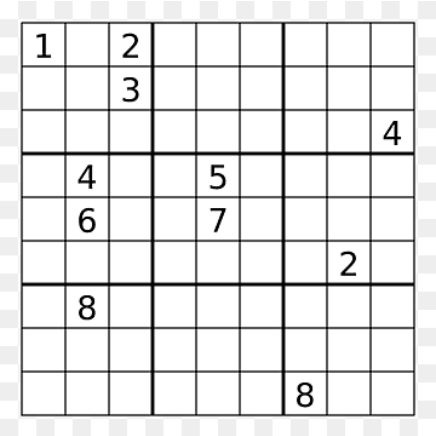


$\rho = x\cos(\theta) + y\sin(\theta)$

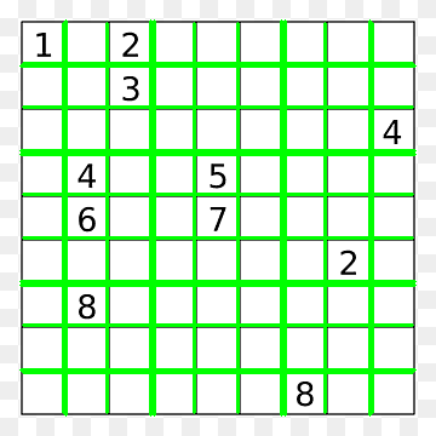




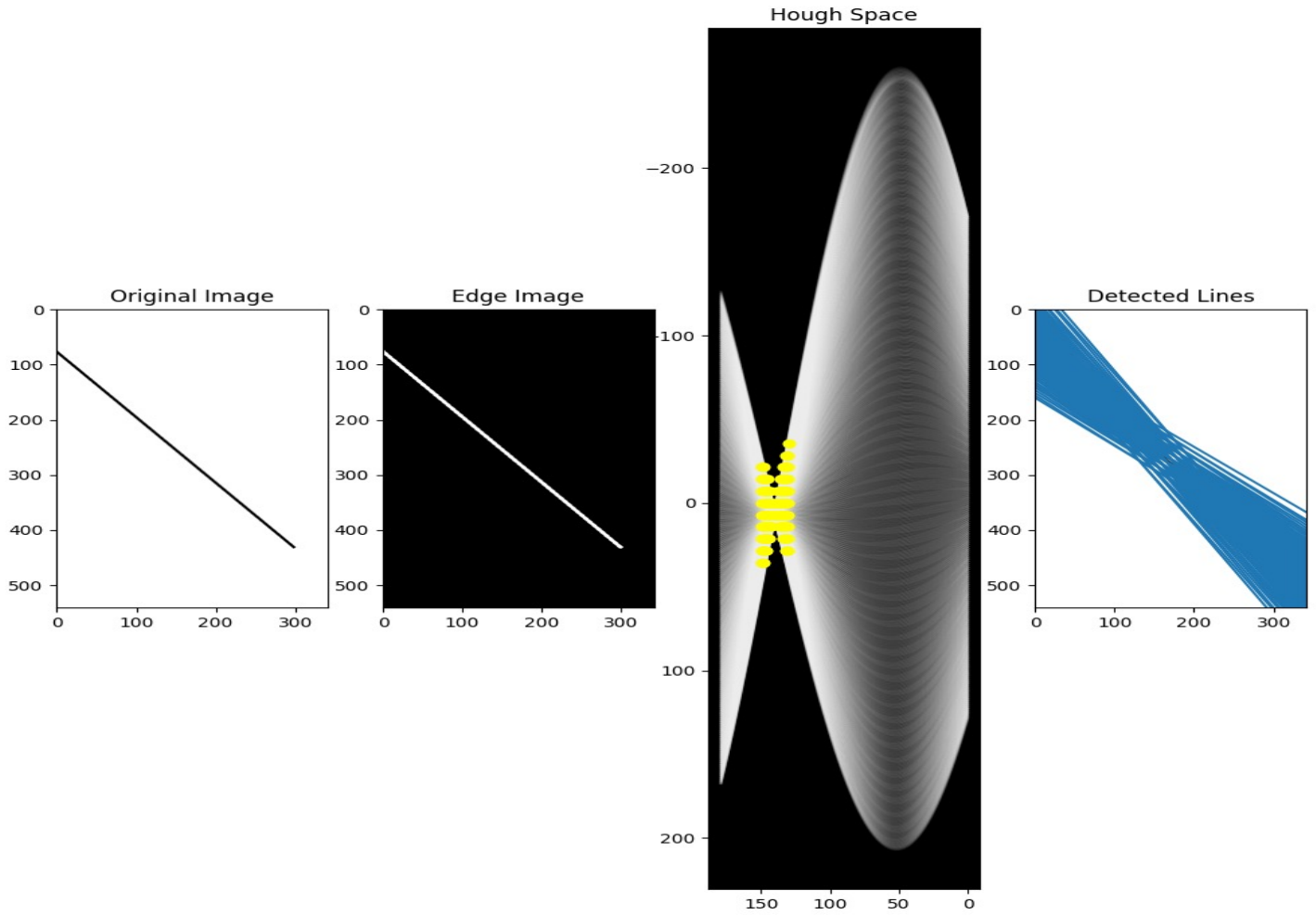
Line Detection: Hough Transform



Input Image



Line Detection Result



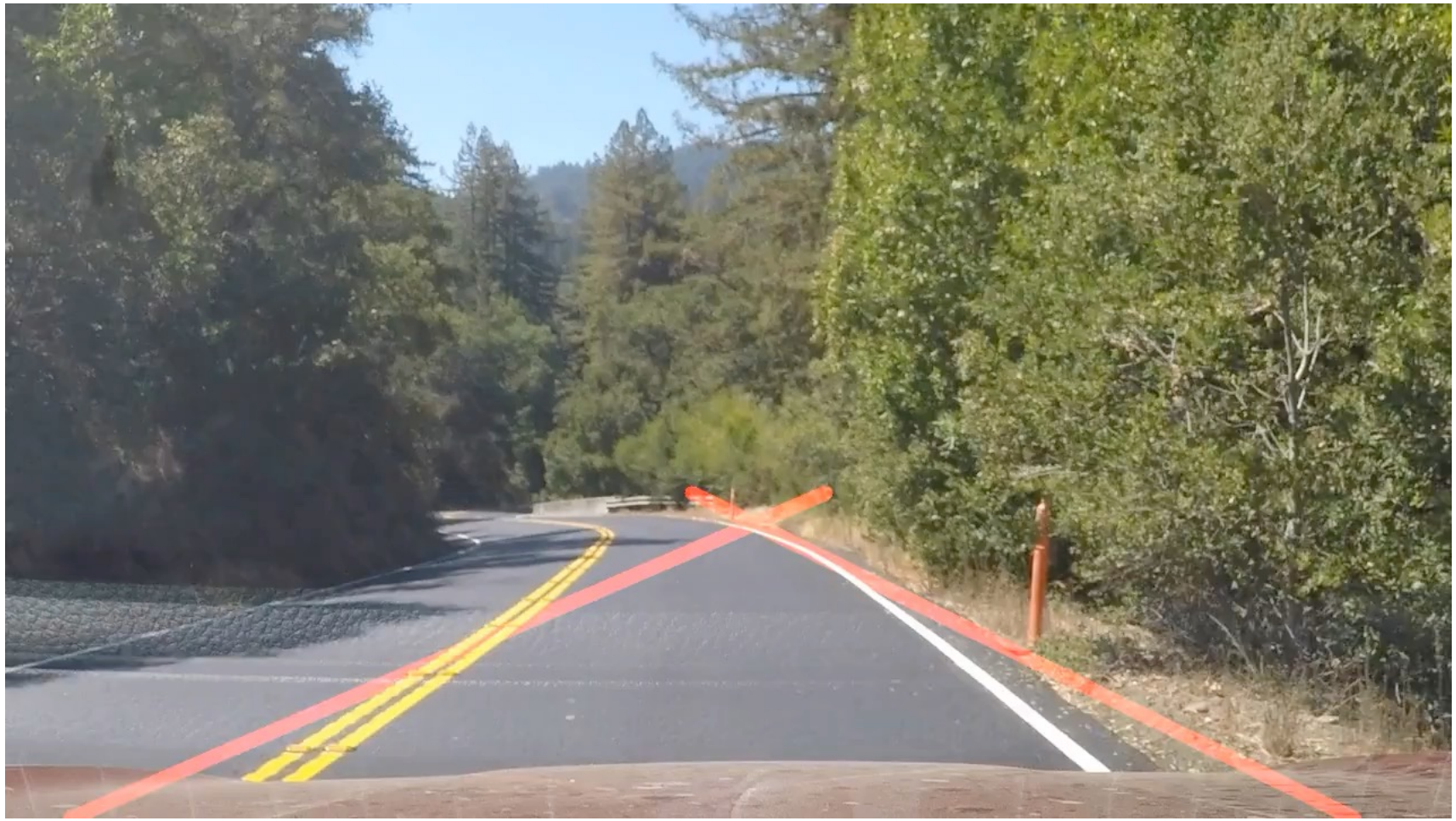
The yellow dots in the Hough Space indicate that lines exist and are represented by the θ and ρ pairs.

Outline

- Background Subtraction Assignment: Review
- Grayscale Conversion
- Blurring Techniques
- Region of Interest (ROI)
- Line Detection based on Hough Transform
- Lane Detection Demo

Lane Detection Demo

Research Challenges




Thank you