Université Constantine 2
جامعة قسنطينة 2

Artificial Vision

– Course 2 –

Chapter 2: IMAGE ANALYSIS (Low-Level Processes)

Dr. Benaliouche Houda
Faculté des nouvelles technologies

Houda.benaliouche@univ-constantine2.dz

Université Constantine 2
جامعة قسنطينة 2

Artificial Vision

– Course 2 –

Chapter 2: IMAGE ANALYSIS (Low-Level Processes)

Dr. Benaliouche Houda

Faculté des nouvelles technologies

Houda.benaliouche@univ-constantine2.dz

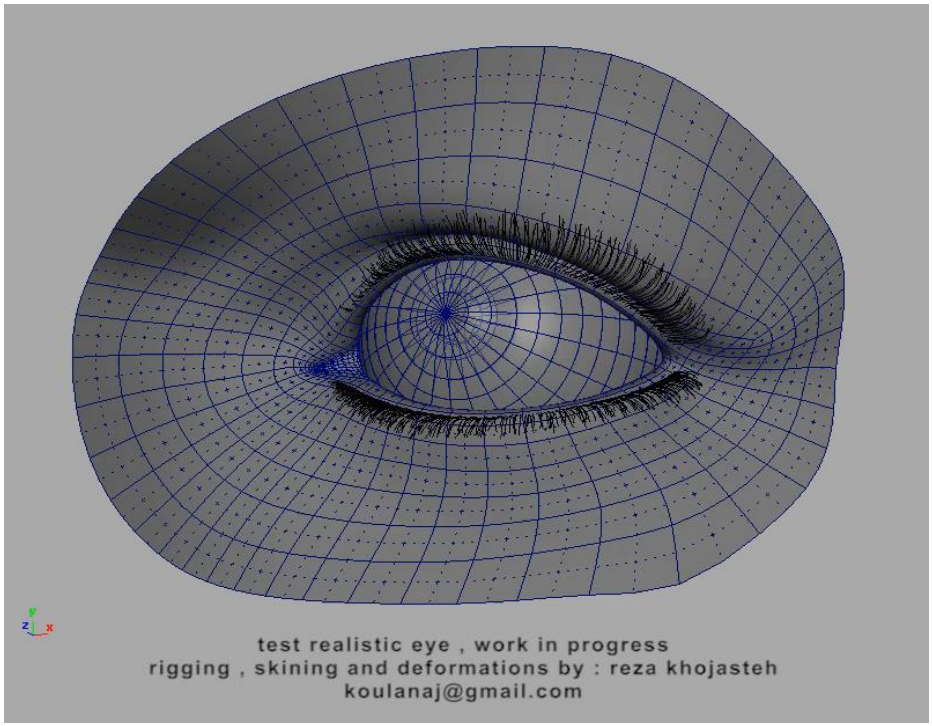
Etudiants concernés

Faculté/Institut	Département	Niveau	Spécialité
Nouvelles technologies	/	Master 2	Sciences de Données et Intelligence Artificielle (SDIA)

Summary

Prerequisites

- Mathematical Notions
- Algorithmic Notions



Course Objective

- Focus on Image Formation, Primitives Detection, and Image Segmentation

OUTLINE

✓ Overview of image analysis

- Briefly introduce **Image Analysis**.
- Discuss the difference between **low-level**, **mid-level**, and **high-level** image processing.
- Focus on low-level processes such as **image formation**, **primitive detection**, and **segmentation**.

✓ Image formation

- Explain the **image formation process** in digital cameras.
- Discuss the **camera model**, including pinhole camera approximation.
- **Optics and Illumination**: Explain how light interacts with objects and how cameras capture this light.
- Mention **radiometric properties** (intensity, brightness).

✓ Primitives detection

- Define **primitives** as basic image elements like **edges**, **lines**, and **corners**.
- **Edge Detection**: Explain the importance of edges and methods such as the **Sobel** and **Canny** edge detectors.
- **Corner Detection**: Introduce the **Harris corner detector** and its importance in tracking image features.

OUTLINE

✓ Image segmentation

"Introduction to Image Segmentation"

Define **image segmentation** as the process of dividing an image into meaningful regions.

Explain why segmentation is crucial for **object recognition** and **scene understanding**.

Introduce types of segmentation:

Thresholding.

Clustering (K-means).

Edge-based methods

Region-based methods.

Graph-based methods

Deep-learning based method

✓ Challenges in low-level Image Analysis

- Discuss challenges such as:
  - **Noise** and its impact on edge detection.
  - **Illumination changes** affecting segmentation.
  - **Complex scenes** leading to segmentation failures.

Section 1:
Overview of image analysis

Overview of image analysis

What is Image Analysis?

Image analysis refers to the process of extracting meaningful information from images, often used in applications such as medical imaging, autonomous driving, surveillance, and more. It involves processing and interpreting visual data to achieve specific goals, such as object recognition, classification, or segmentation.

Overview of image analysis

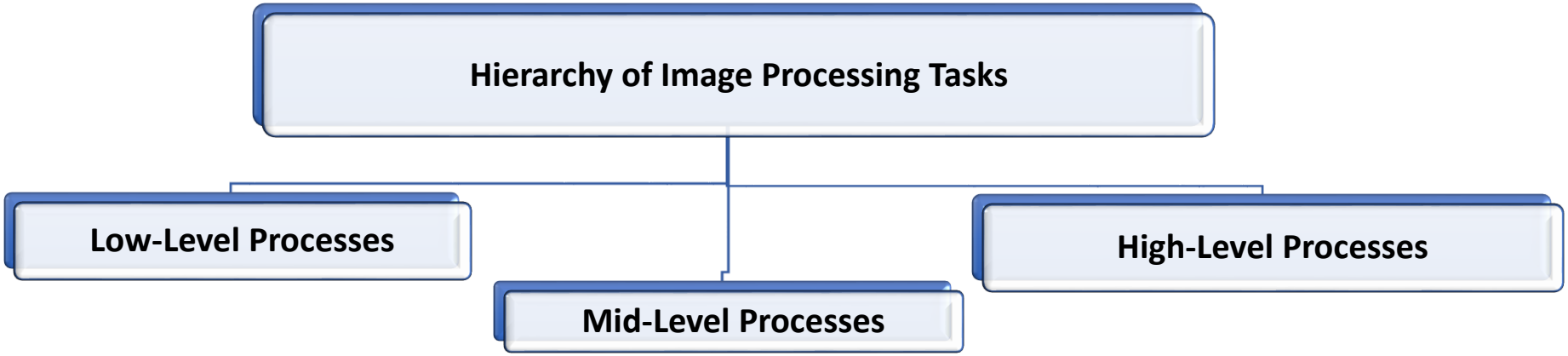
Key Objective:
The goal of image analysis is to transform visual data into a form that is more useful for human interpretation or further automated processing by computers.

Overview of image analysis

Hierarchy of Image Processing Tasks:

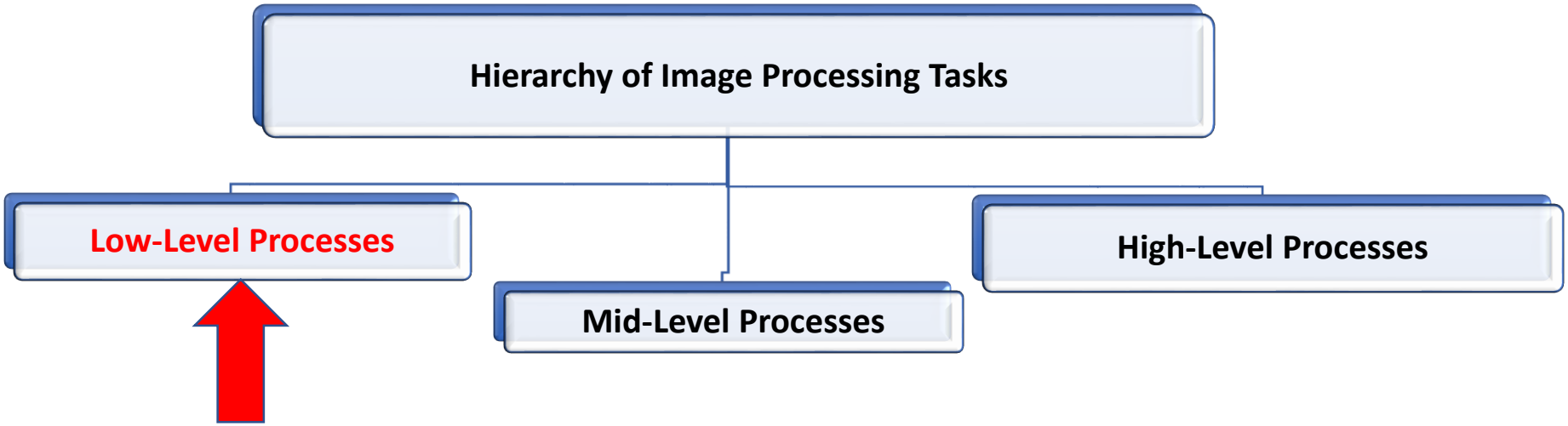
•Image analysis tasks are typically categorized into **three levels** based on their complexity:

- **Low-Level Processes**: These operations are focused on basic transformations and enhancements of raw pixel data.
  - Examples include **image formation**, **noise reduction**, **edge detection**, and **segmentation**.
- **Mid-Level Processes**: These involve extracting meaningful information from images, such as detecting **objects**, **regions**, or **features** like edges and textures.
- **High-Level Processes**: These processes involve understanding and interpreting the content of an image, such as **scene understanding**, **object recognition**, or **image classification**.



Overview of image analysis

Key Note: Today's focus will be on **low-level processes**, which are fundamental for higher-level tasks. Understanding the low-level operations enables more effective mid- and high-level processing.



Overview of image analysis

Low-Level Image Processing:

•**Definition**: Low-level image processing refers to pixel-level operations that are performed directly on the image to improve its quality, extract key features, or prepare it for further analysis.

•**Examples of Low-Level Tasks**:

- **Image Formation**: This involves understanding how images are captured and represented digitally. It includes aspects such as the camera model, optics, lighting, and sensor characteristics.
- **Primitives Detection**: Primitives such as edges, corners, and textures are detected in an image. These form the building blocks for more advanced analysis.
- **Image Segmentation**: This task involves dividing the image into meaningful regions or segments, usually based on properties like pixel intensity or color.

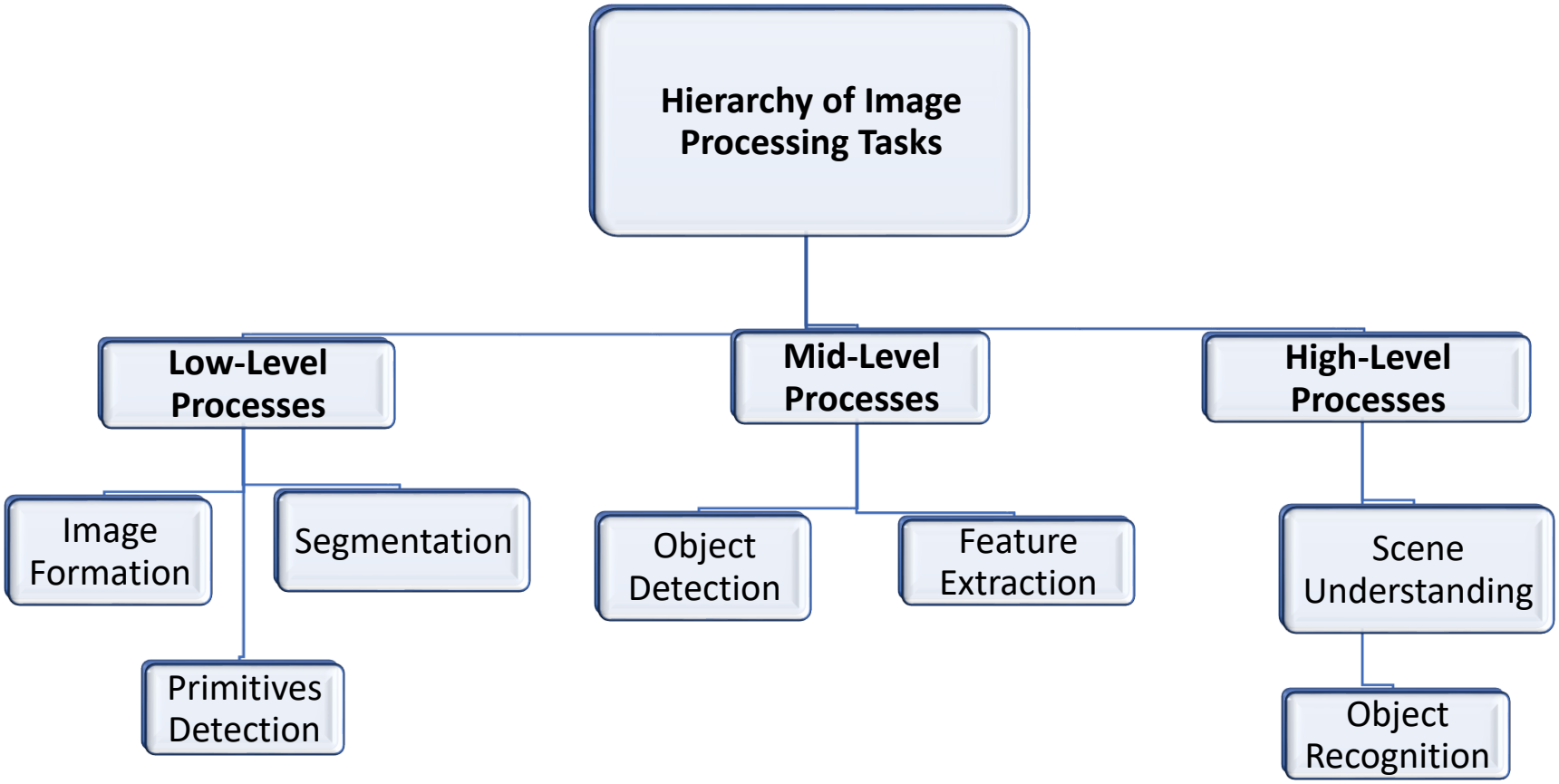
Overview of image analysis

Importance of Low-Level Processing:

•**Foundation for Higher-Level Vision**: Low-level processes are essential for generating features that mid- and high-level processes rely on. For example, accurate **edge detection** is critical for object recognition, and proper **segmentation** helps in separating objects within a scene.

•**Pre-processing for Noise Reduction**: Low-level operations often deal with noise and imperfections in images caused by poor lighting, camera artifacts, or environmental factors. Techniques like **smoothing**, **filtering**, and **enhancement** are used to improve image quality.

Overview of image analysis



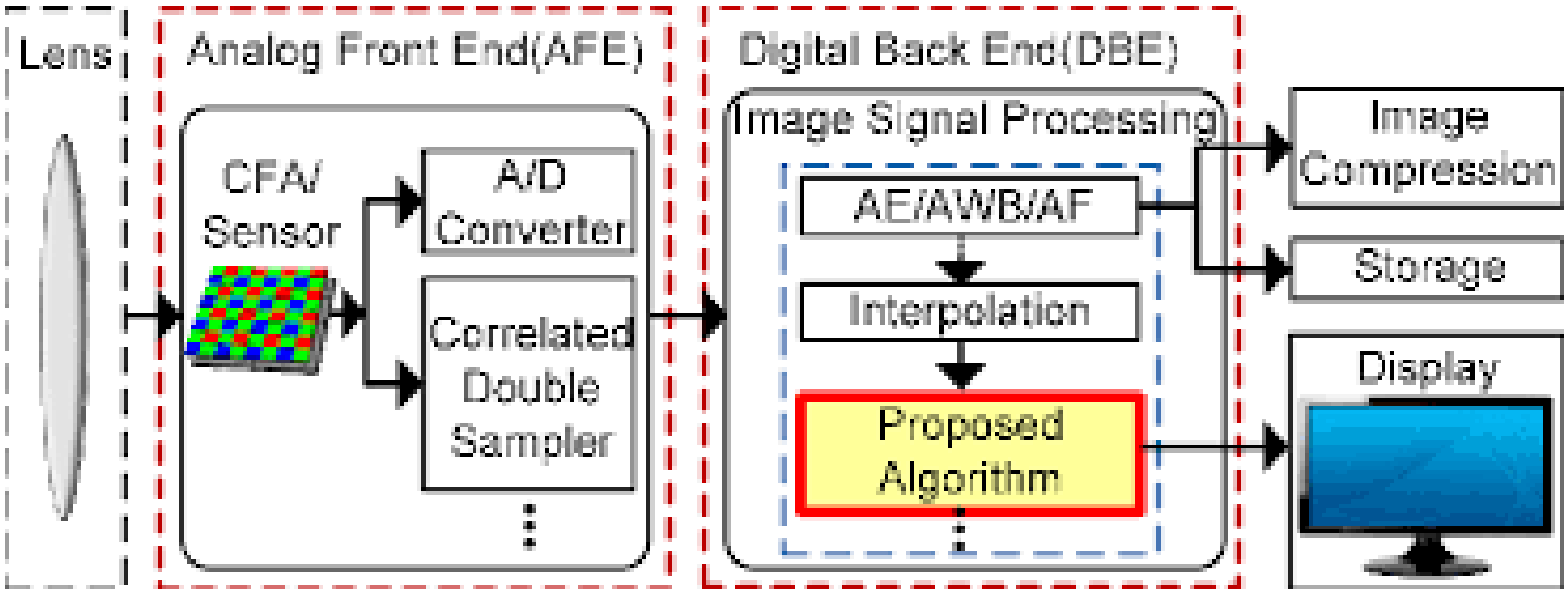
Section 2:
image formation

Understanding Image Formation

- Explain the **image formation process** in digital cameras.
- Discuss the **camera model**, including pinhole camera approximation.
- **Optics and Illumination**: Explain how light interacts with objects and how cameras capture this light.
- Mention **radiometric properties** (intensity, brightness).

Understanding Image Formation

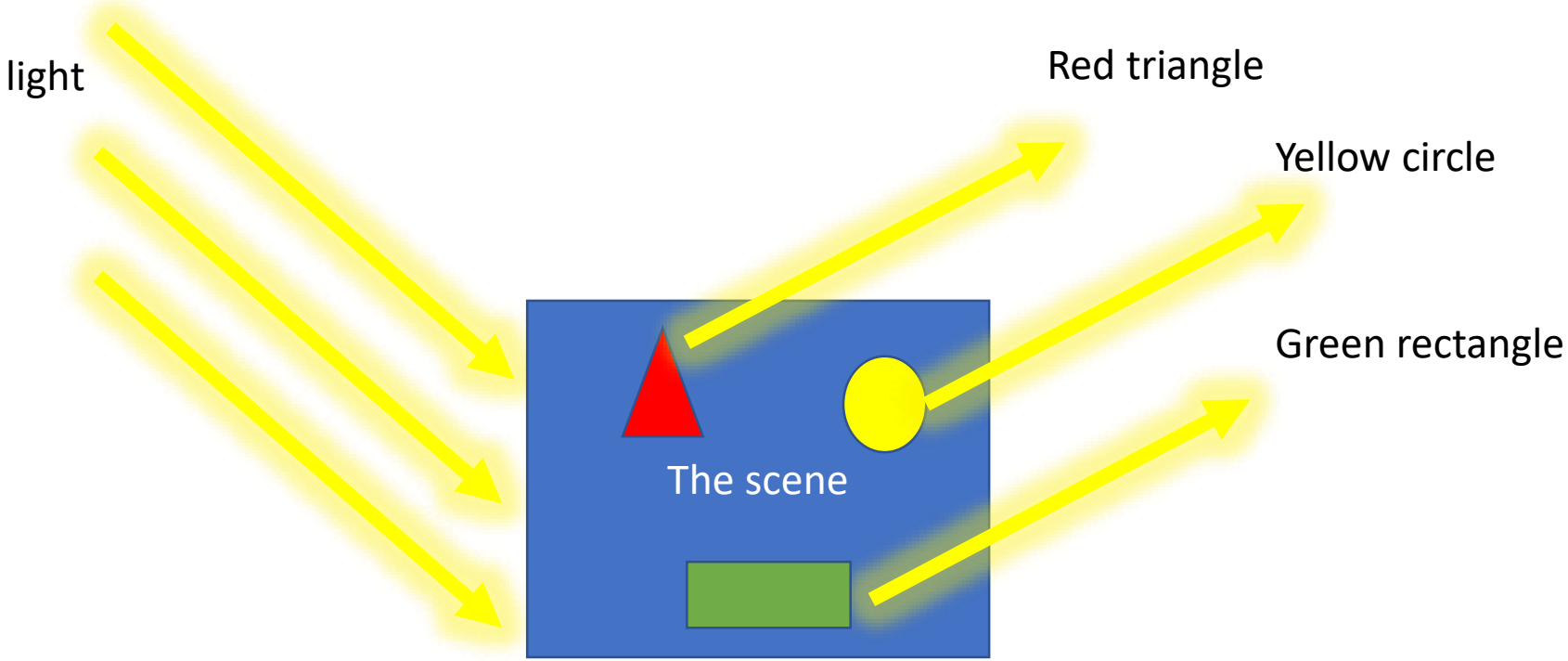
The **image formation process** in digital cameras involves several key steps that transform light from a scene into a digital image that can be stored and processed. This process is largely based on the principles of optics, sensors, and digital conversion. Here's a breakdown of how it works:



Understanding Image Formation

1. Light Interaction with the Scene

•The process begins when light from an external light source (such as the sun or a lightbulb) illuminates a scene. Objects in the scene reflect or emit light, and this light carries information about the color, texture, and intensity of the scene.

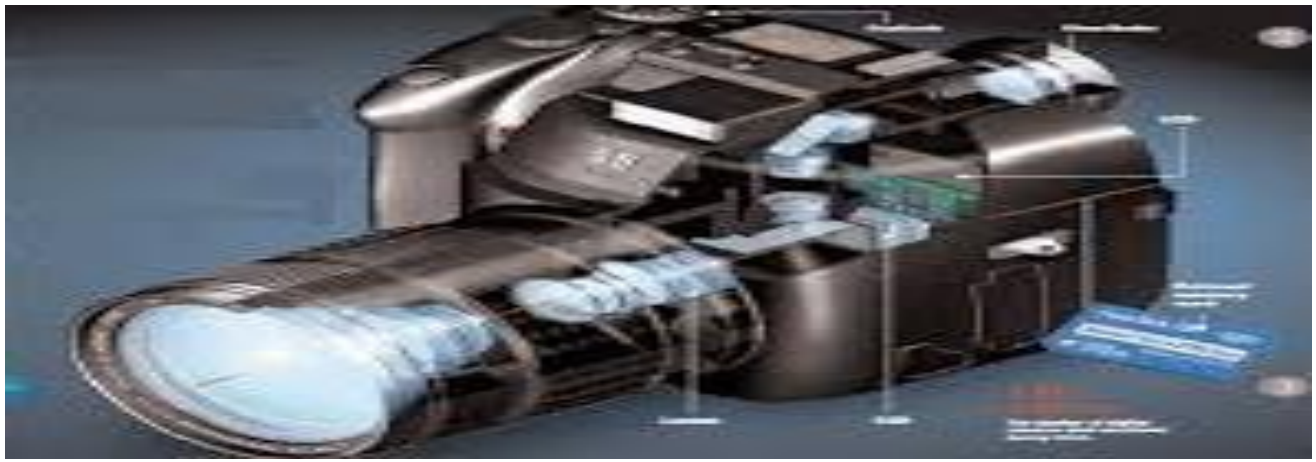


Understanding Image Formation

2. Light Passing through the Lens

•The **lens** of the camera is responsible for focusing the light from the scene onto the camera's sensor.

- **Focusing**: The lens bends (refracts) incoming light rays so they converge to form a clear image on the sensor.
- **Aperture**: The size of the aperture (an adjustable opening in the lens) controls the amount of light entering the camera. A wider aperture allows more light, while a smaller one reduces the light.
- **Focal Length**: The lens also determines the **field of view** and the **level of magnification** based on its focal length. A short focal length provides a wide-angle view, while a long focal length provides a narrow, zoomed-in view.

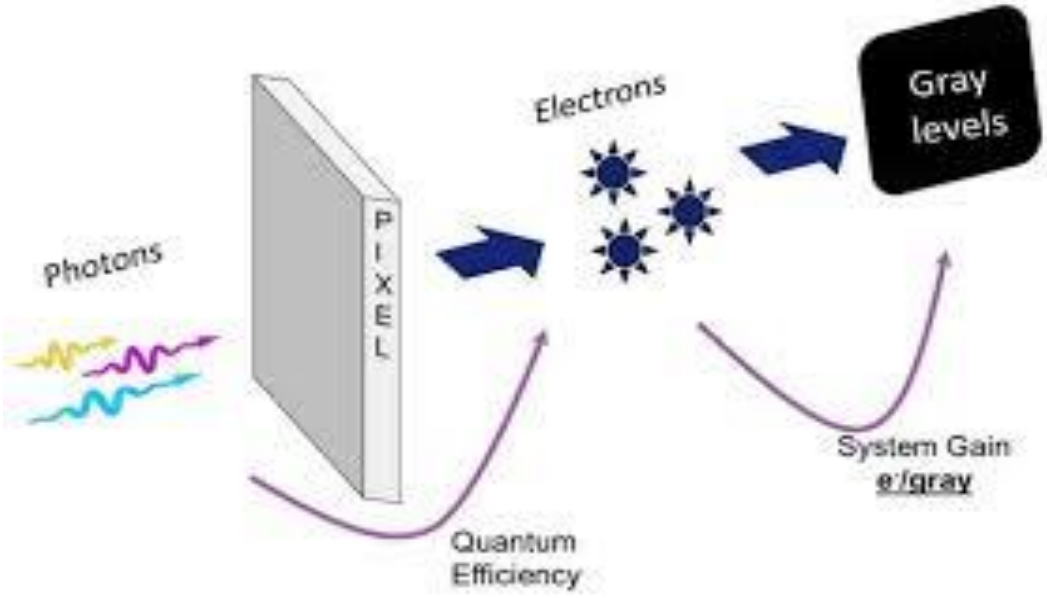


Understanding Image Formation

4. Converting Light into Electrical Signals (Sampling)

•When light hits the sensor, each photosite converts the light energy (photons) into an **electrical charge** proportional to the light's intensity. Brighter light creates a stronger charge, while dim light creates a weaker charge. This process is known as **sampling**.

- The sensor samples the continuous light field at discrete points (pixels), capturing the intensity of light at each location.
- The result is a grid of electrical charges, one for each pixel, representing the intensity of the light at that point in the image.



Understanding Image Formation

5. Color Filter and Color Image Formation

•**Color filters** are applied to the sensor's photosites because individual photosites can only measure the intensity of light, not its color.

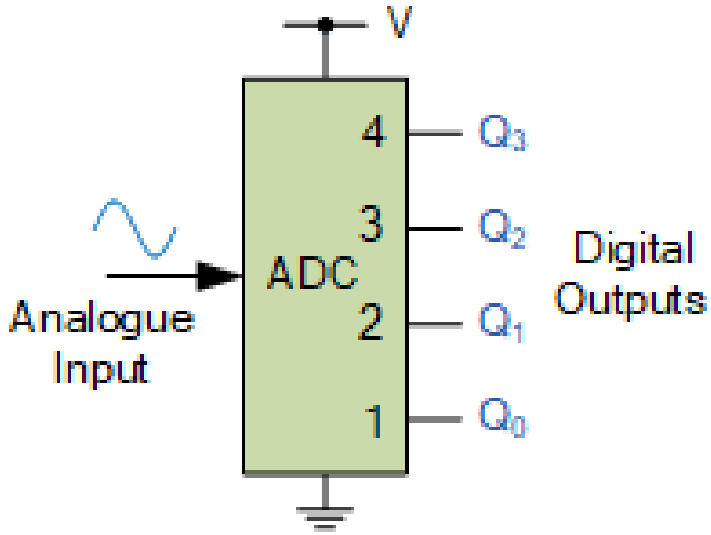
- A **Bayer filter** (a grid of red, green, and blue filters) is placed over the sensor, allowing each pixel to capture only one of the three primary colors (red, green, or blue).
- Typically, there are twice as many green filters as red or blue because the human eye is more sensitive to green light.
- The camera's processor later reconstructs the full-color image by combining the information from neighboring pixels in a process called **demosaicing**.

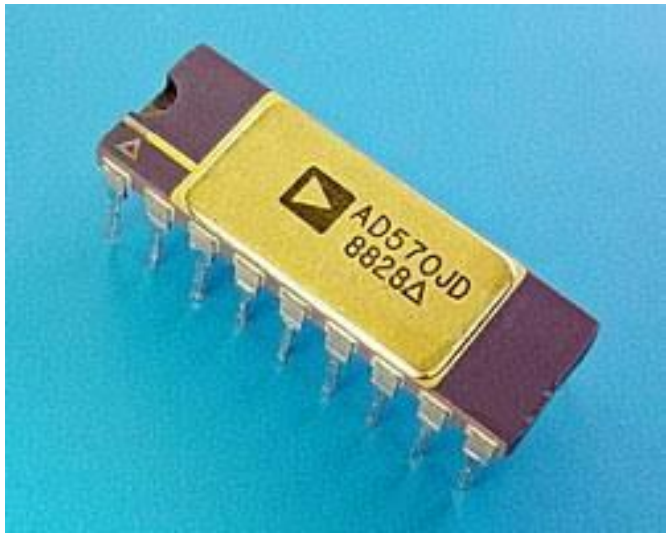
Understanding Image Formation

6. Quantization and Analog-to-Digital Conversion

•Once the photosites have measured the light intensity and the color filters have filtered the light, the electrical charges are converted into **digital values** through **Analog-to-Digital Conversion (ADC)**.

- The continuous range of light intensities is mapped to a finite set of values (e.g., in an 8-bit image, each pixel is represented by a value between 0 and 255, where 0 is black and 255 is white).
- This process is known as **quantization**, where the continuous intensity values are rounded to the nearest discrete digital level.



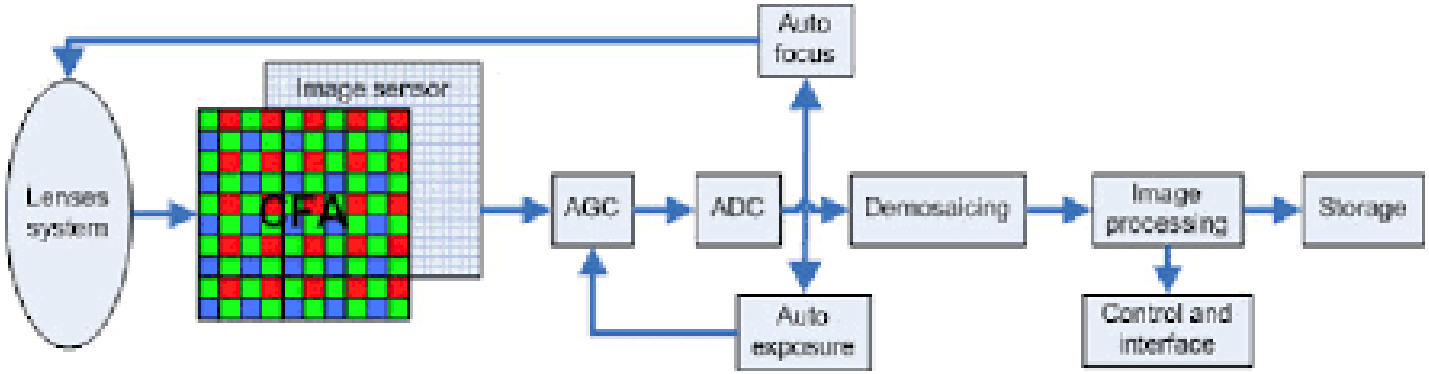


Understanding Image Formation

7. Image Processing in the Camera

•Once the light data is digitized, the camera's internal processor (or image signal processor) applies several **post-processing steps** to enhance the image:

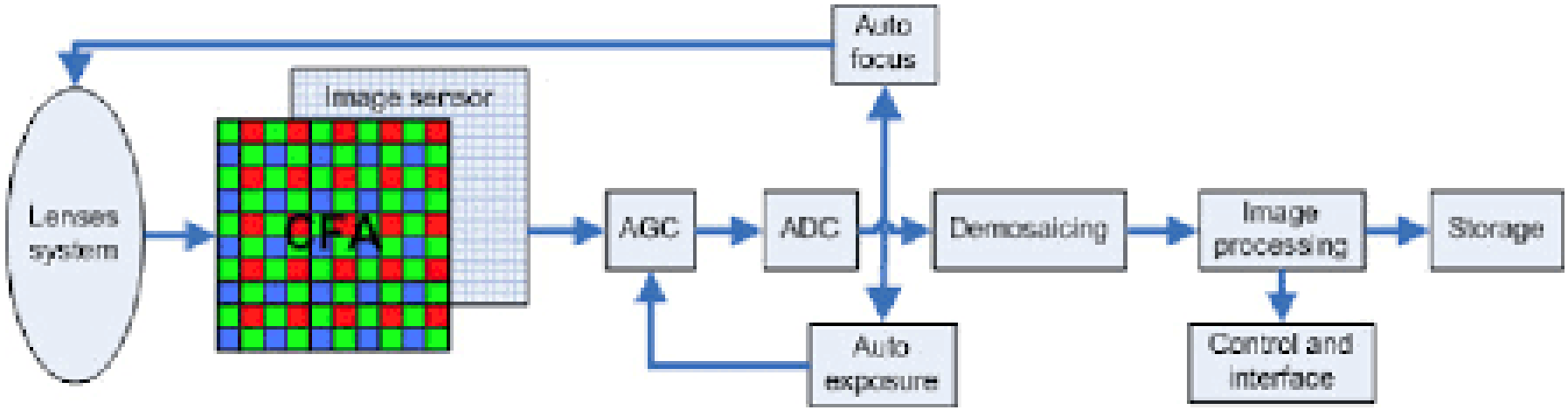
- **White balance**: Adjusts the colors in the image to compensate for the color temperature of the light source (e.g., warm indoor lighting vs. cool outdoor lighting).
- **Noise reduction**: Removes random variations in pixel values caused by sensor imperfections or low light conditions.
- **Sharpening**: Enhances edges in the image to make it appear clearer.
- **Compression**: The raw image data is often compressed into formats such as JPEG to reduce file size.



Understanding Image Formation

8. Storing the Image

•After processing, the digital image is stored in the camera's memory card in a chosen format (JPEG, RAW, etc.). The image is now ready for viewing, editing, or sharing.



Understanding Image Formation

Summary:
The image formation process in digital cameras involves:

1.Light interacting with a scene.
2.Passing through the camera lens and being focused onto a sensor.
3.The sensor sampling the light and converting it into electrical signals.
4.Color filters separating light into primary colors.
5.Analog-to-digital conversion to represent the intensity of light as digital values.
6.Post-processing to enhance and store the image.

These steps together allow a digital camera to convert real-world scenes into high-quality digital images.

Understanding Image Formation

Sampling:

•**Definition**: Sampling is the process of converting a continuous signal (like the light intensity that forms an image) into a discrete set of values by measuring it at regular intervals.

•**How it works**:

- When light enters a camera sensor, the sensor measures the intensity of light at each point (usually at pixel locations). This measurement happens across a 2D grid of small sensing elements called **pixels**.
- Each pixel collects the incoming light over a small area. The sensor "samples" the scene by measuring the light intensity at each pixel location at regular intervals, creating a grid of intensity values.
- The **spatial resolution** of the image depends on the number of pixels in this grid. More pixels mean higher resolution and better detail.

Understanding Image Formation

Key point: Sampling reduces the continuous (analog) world into discrete pixels. The finer the sampling, the closer the digital image resembles the original continuous image.

Example: Consider a 5x5 pixel grid capturing an image. Sampling occurs at each point in this grid, and each pixel samples the light intensity at its position.

Understanding Image Formation

5x5 Simple pixel grid

Understanding Image Formation

Quantization:

•**Definition**: Quantization is the process of mapping the continuous range of intensity values (the amount of light at each pixel) into a limited set of discrete levels.

•**How it works**:

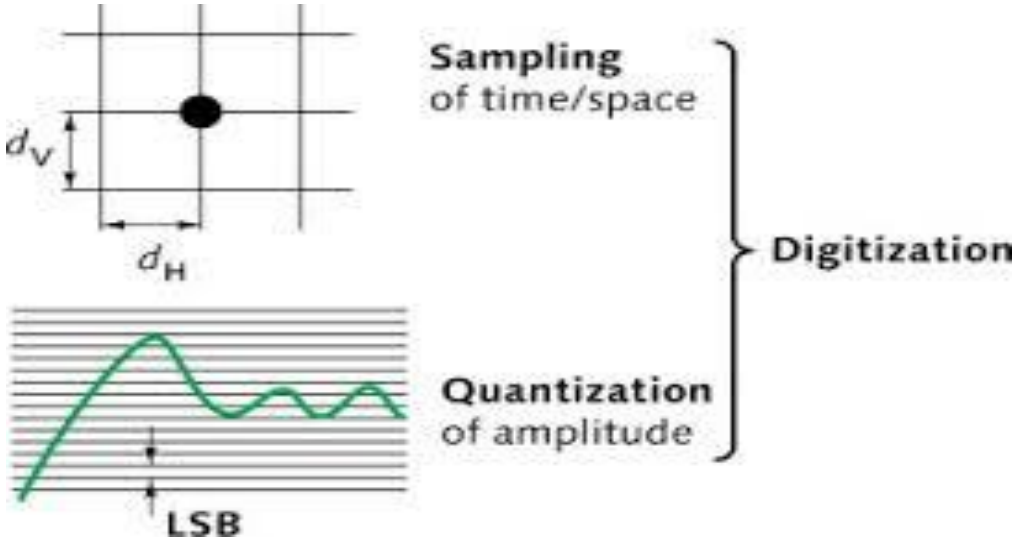
- Once light intensity has been sampled at each pixel, it needs to be represented digitally. This means the continuous intensity values measured by the sensor must be mapped to a finite number of intensity levels (often expressed in bits).
- For example, in an 8-bit image, each pixel's intensity is represented by a value between **0 and 255**, where 0 is black and 255 is white, with shades of gray in between.
- Quantization introduces some loss of information because the continuous range of intensities must be approximated to fit within a fixed number of discrete levels.
- The **bit depth** of the image determines how many levels are available for quantization. A higher bit depth (e.g., 16-bit) allows more precision in representing intensity levels, reducing the loss of information.

Understanding Image Formation

Summary:

•**Sampling** converts continuous spatial information into discrete points (pixels) in a grid, while **quantization** assigns each pixel a discrete value corresponding to the intensity of the light captured by the sensor.

•Both processes are critical in converting an analog scene (continuous light) into a digital image that can be stored, processed, and displayed by digital systems.



Understanding Image Formation

Key point: Quantization reduces the continuous range of intensity values into discrete steps. The more levels of quantization, the better the digital approximation of the image.
Example: If the intensity of light at a pixel is measured as 87.6, it will be rounded to the nearest available level, say 88, based on the bit depth (e.g., 8-bit).

Section 3:
Primitives detection

Primitives Detection

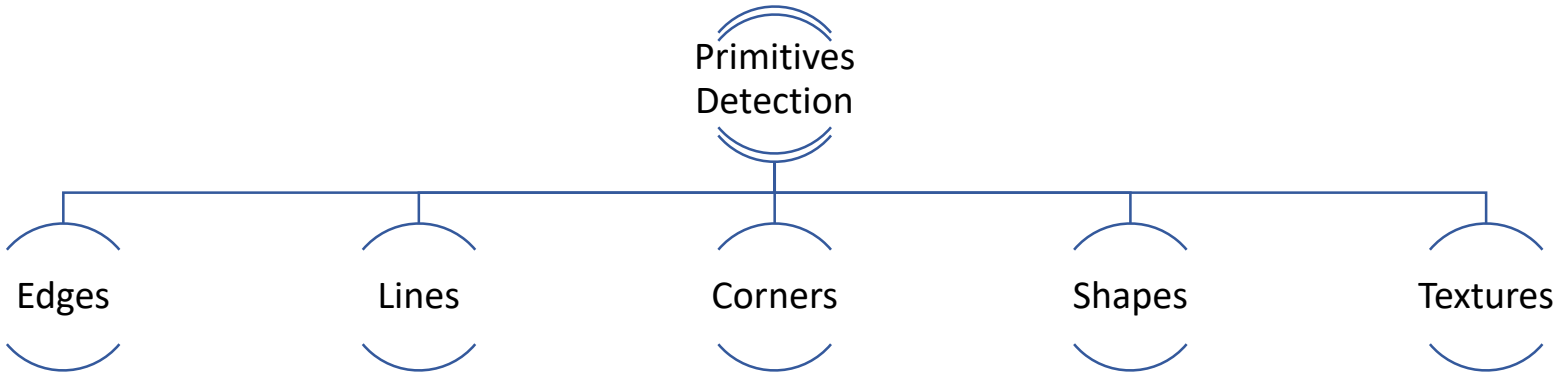
•Define **primitives** as basic image elements like **edges**, **corners**, and **regions**.
•**Edge Detection**: Explain the importance of edges and methods such as the **Sobel** and **Canny** edge detectors.
•**Corner Detection**: Introduce the **Harris corner detector** and its importance in tracking image features.

Example of an image with edge detection applied.

Primitives Detection

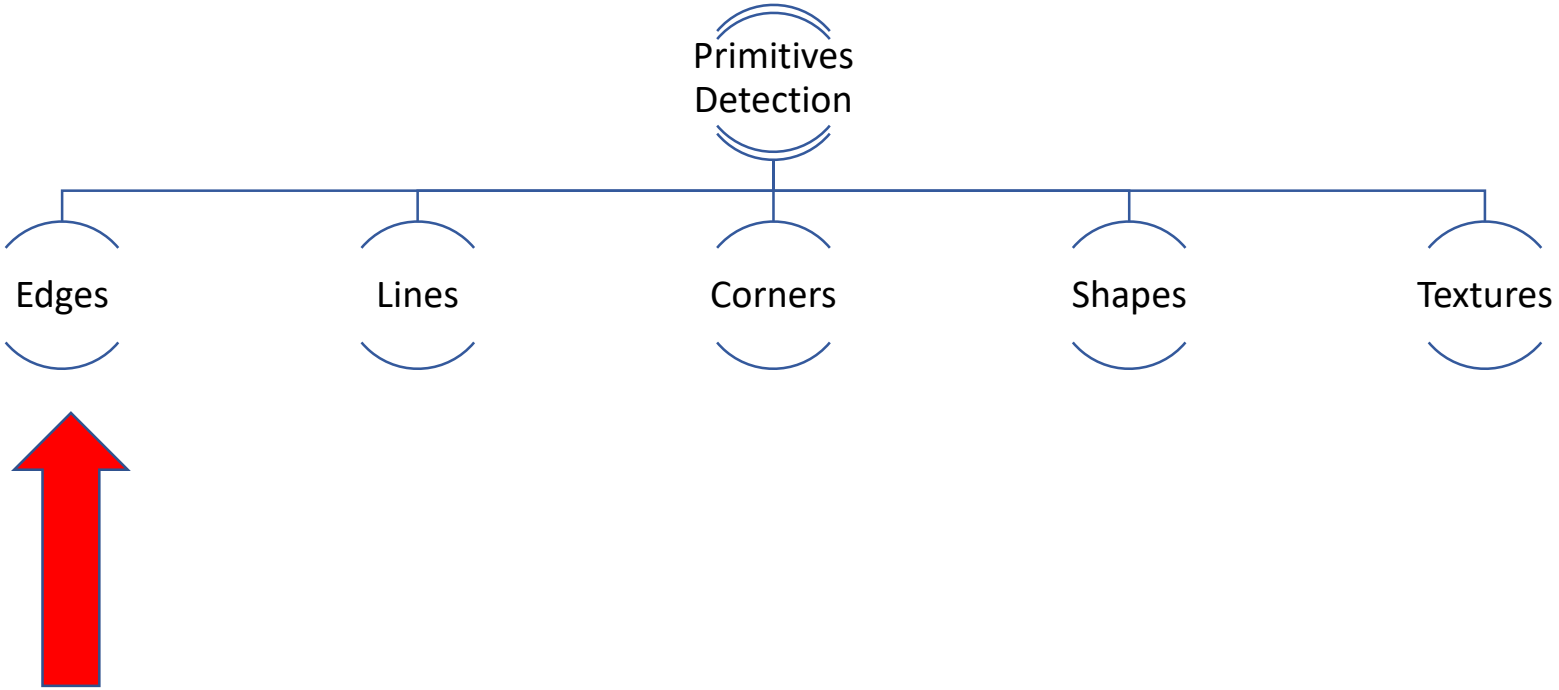
Definition

In computer vision, **primitives detection** refers to the process of identifying basic geometric features or structures in an image that serve as foundational elements for further analysis. These primitives include simple elements like **edges**, **corners, lines, shapes and textures,** which are crucial for interpreting more complex objects or patterns in the image.



Primitives Detection

1/ Edges



Primitives Detection

Key Primitives Detected in Computer Vision:

1/ Edges:

•**Description**: Edges represent areas in an image where the intensity of pixel values changes abruptly. These changes often signify the boundaries of objects or regions.

•**Detection Methods**: Techniques like Sobel, Prewitt, or Canny edge detection algorithms are commonly used.

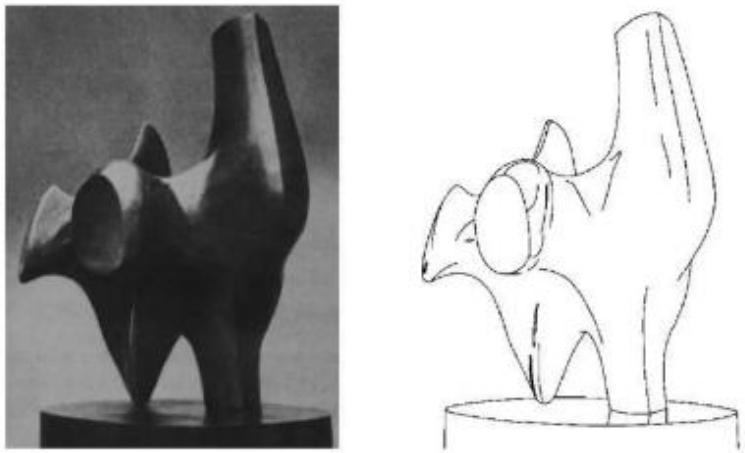
•**Application**: Edge detection is critical for object detection, image segmentation, and contour recognition.

Primitives Detection

Key Primitives Detected in Computer Vision:

1/ Edges:

•Rapid change in image intensity within a small region



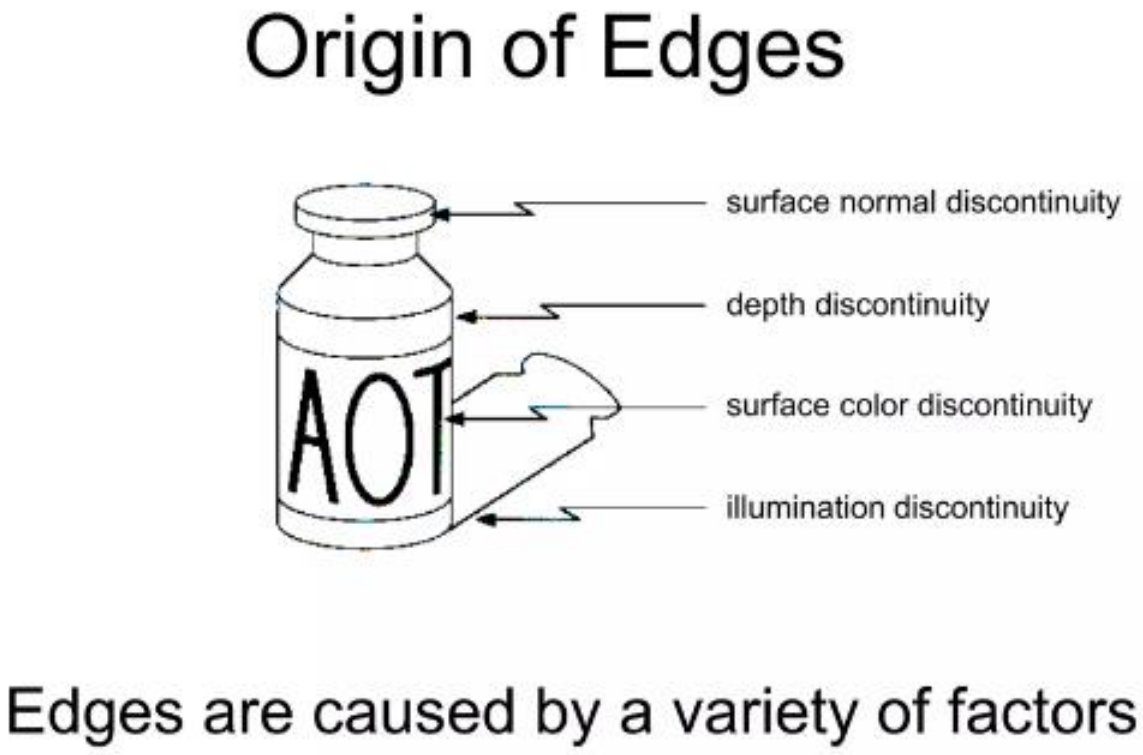
Convert a 2D image into a set of curves

– Extracts salient features of the scene

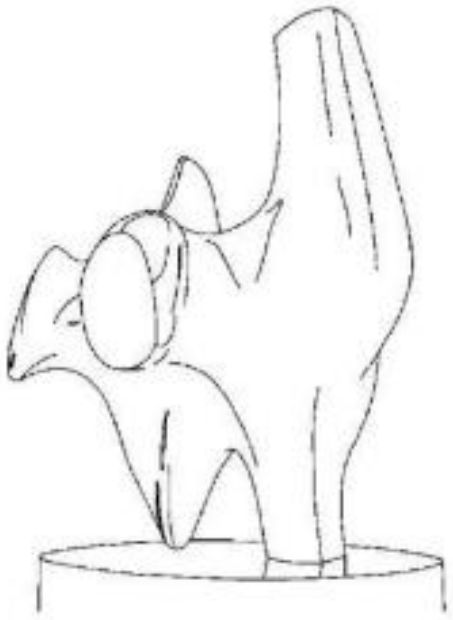
– More compact than pixels

Primitives Detection

•**Origin of Edges:**



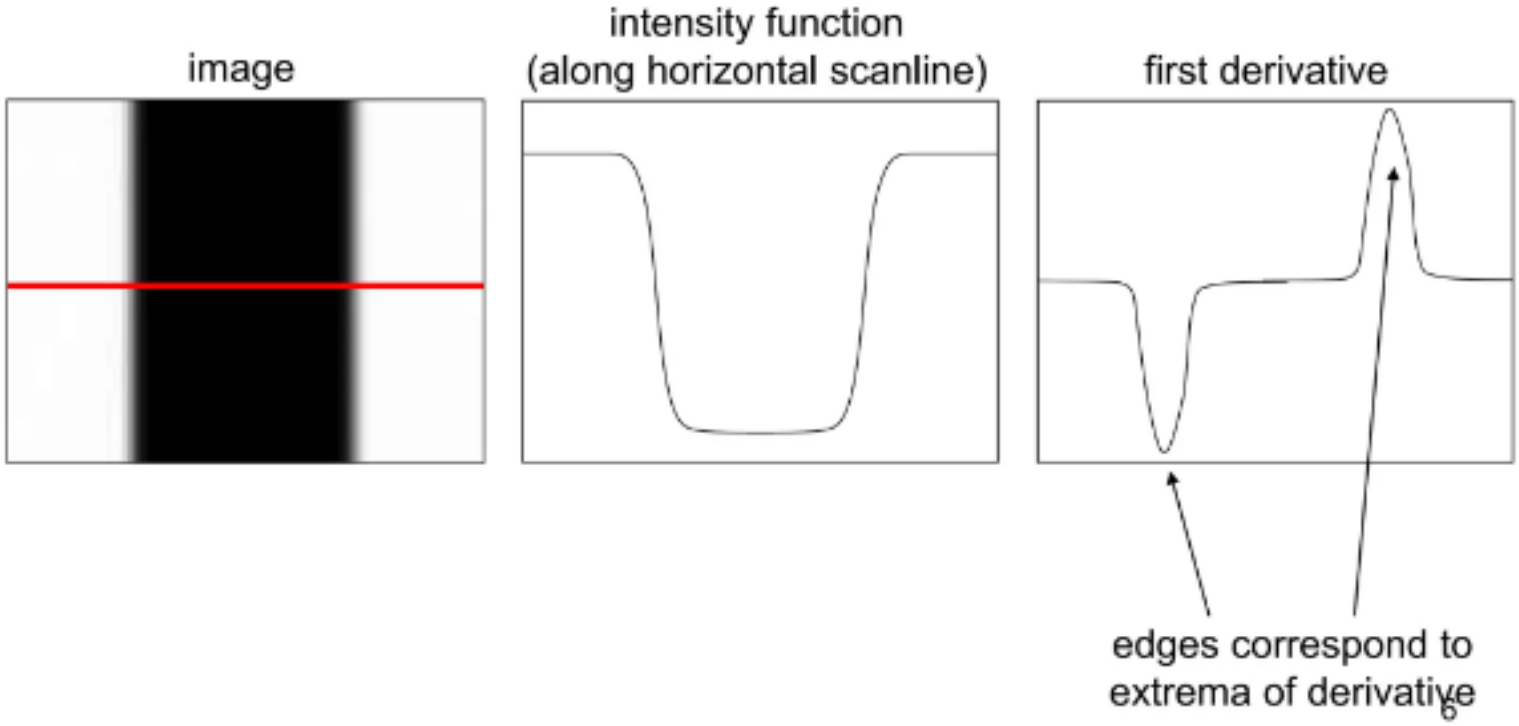
Primitives Detection



How can you tell that a pixel is on an edge?

Characterizing edges

- An edge is a place of rapid change in the image intensity function



Primitives Detection

The Edge Operator produces:

- **Edge position**
- **Edge Magnitude**
- **Edge Orientation**

Performance requirement:

Low false positives and false negatives

Primitives Detection

Edge Detection Algorithms

- Explain the steps of **Sobel operator** (gradient approximation).
- Overview of **Canny Edge Detection**:
  - Gradient calculation.
  - Non-maximum suppression.
  - Hysteresis thresholding.

Primitives Detection

Edge Detection Algorithms

- Explain the steps of **Sobel operator** (gradient approximation).
- Overview of **Canny Edge Detection**:
  - Gradient calculation.
  - Non-maximum suppression.
  - Hysteresis thresholding.

Primitives Detection

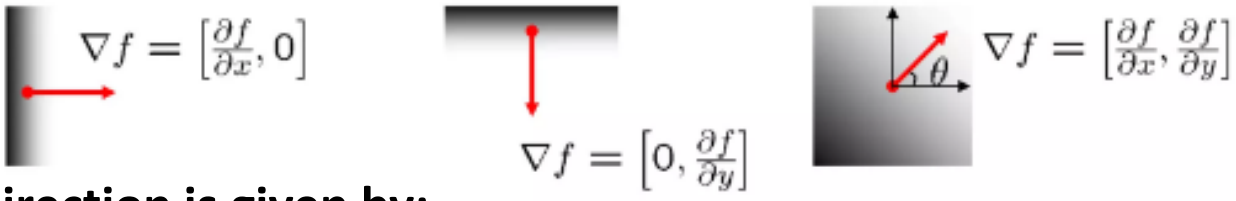
Key Primitives Detected in Computer Vision: corners

Image gradient:

The gradient of an image:

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

The gradient points in the direction of most rapid change in intensity:

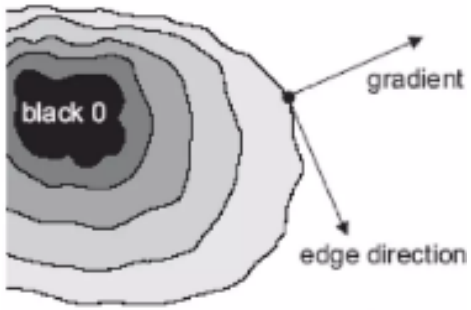


The gradient direction is given by:

$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

The edge strength is given by the gradient magnitude:

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$



Sobel operator:

In practice, it is common to use:

$$g_x = \begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array} \qquad g_y = \begin{array}{|c|c|c|}\hline -1 & -2 & -1 \\ \hline 0 & 0 & 0 \\ \hline 1 & 2 & 1 \\ \hline \end{array}$$

Magnitude: $g = \sqrt{g_x^2 + g_y^2}$

Orientation: ...

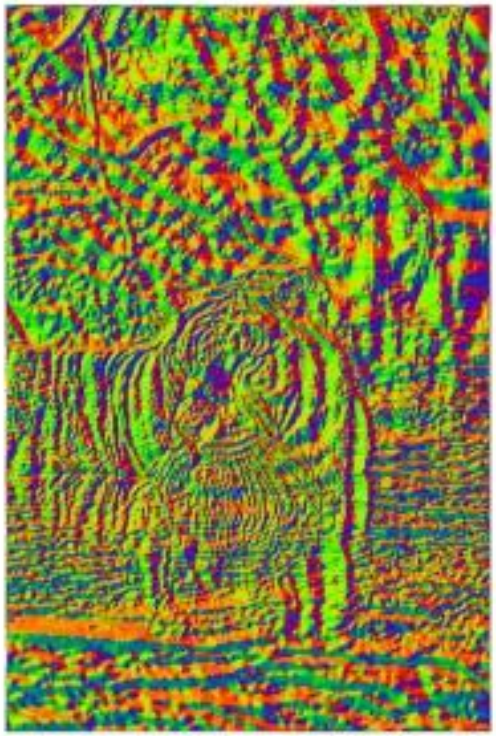
$$\Theta = \tan^{-1}\left(\frac{g_y}{g_x}\right)$$

Primitives Detection

Sobel operator

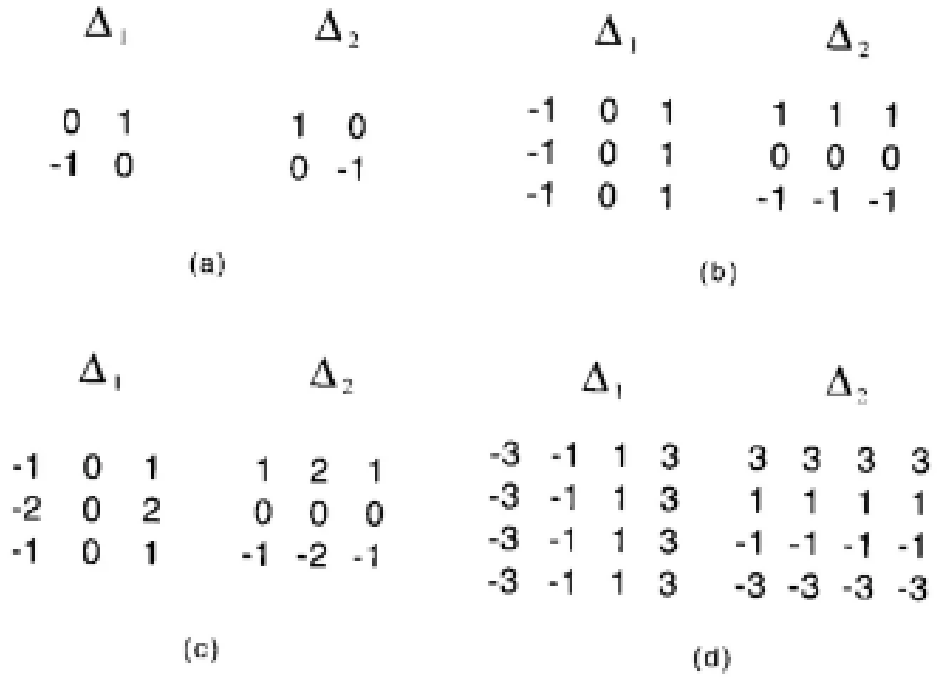
Original

Magnitude



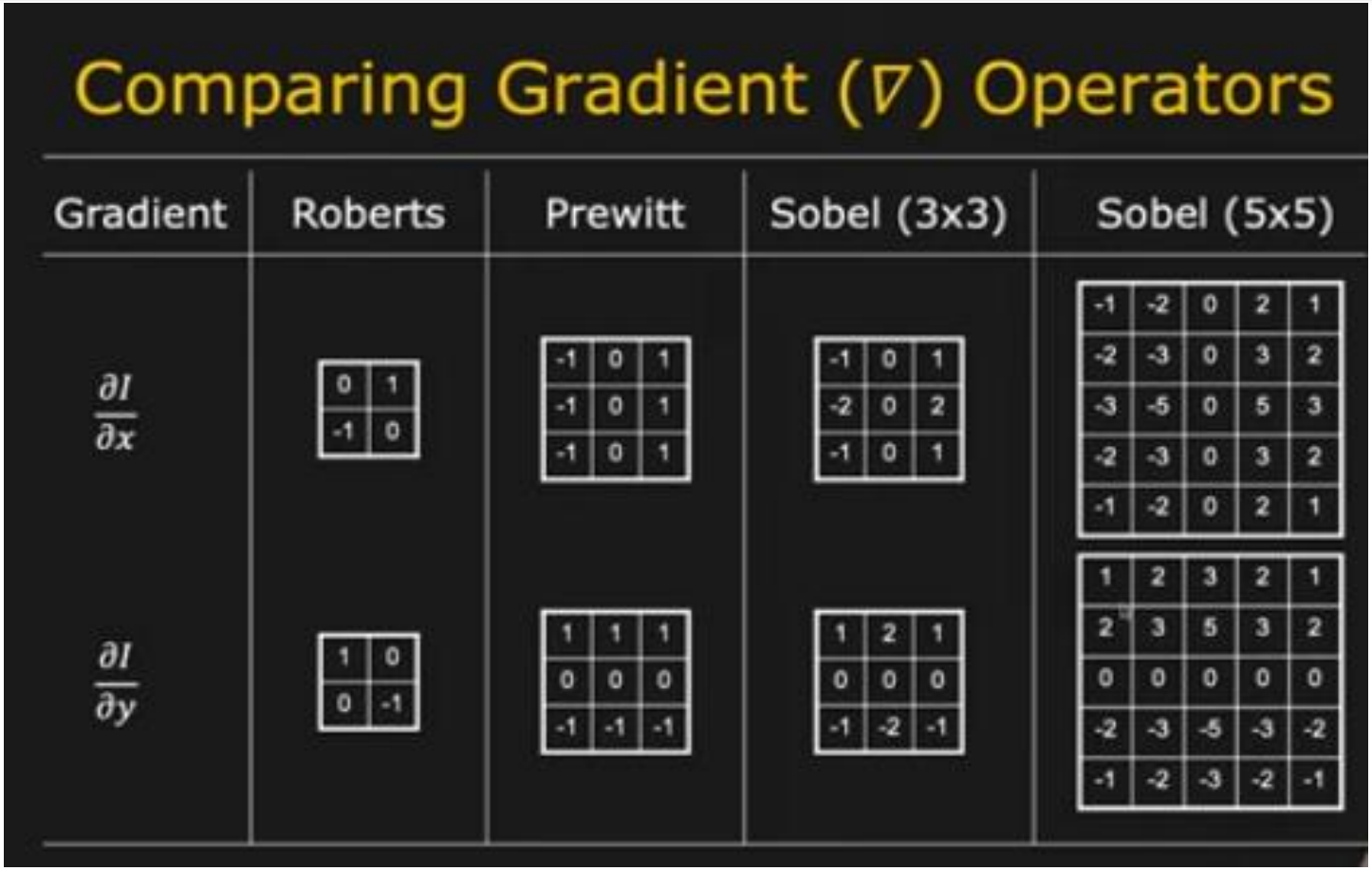
Orientation

Gradient operators



(a): Roberts' cross operator (b): 3x3 Prewitt operator
(c): Sobel operator (d) 4x4 Prewitt operator

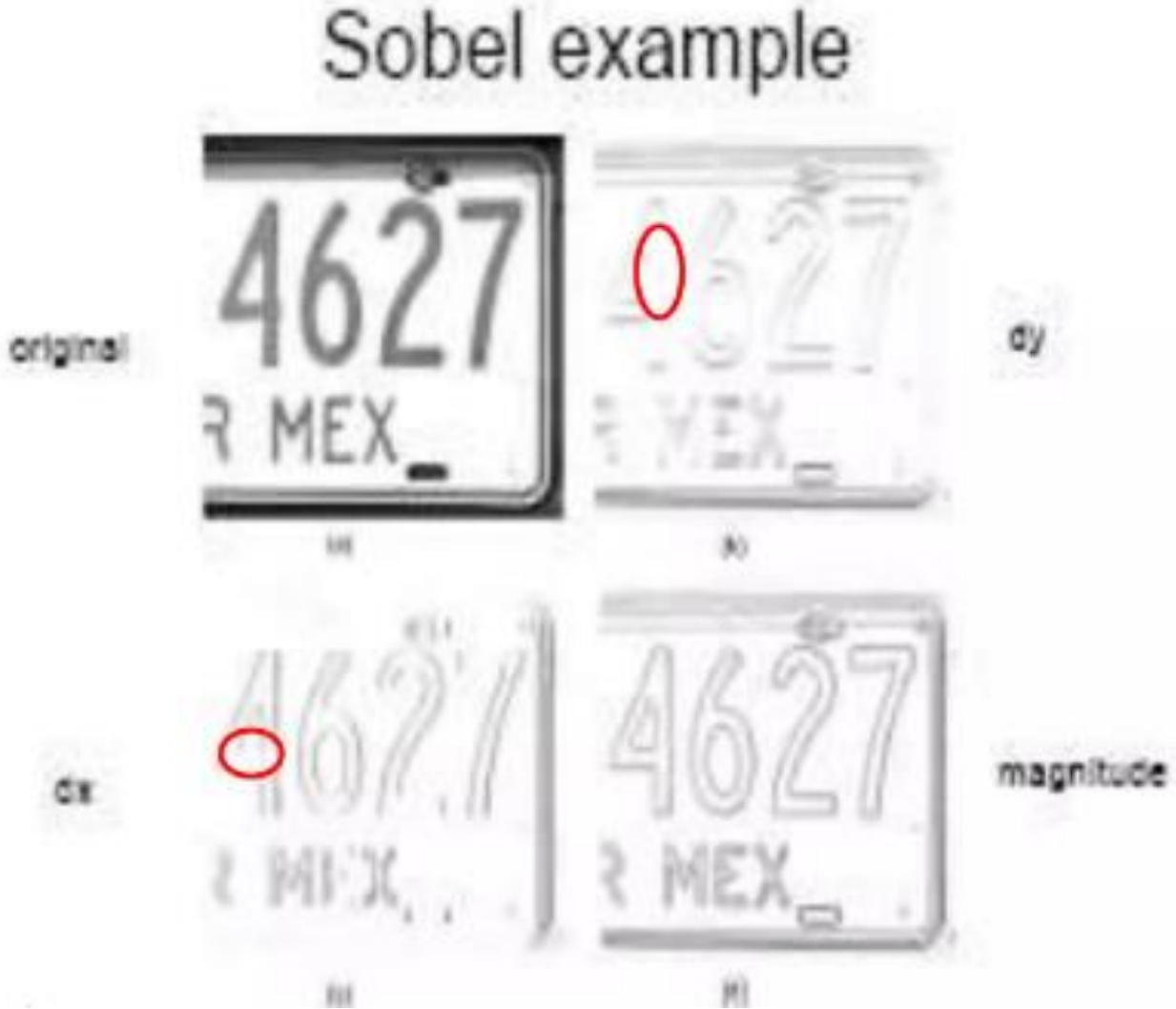
Primitives Detection



Good localization
Noise sensitive
Poor detection

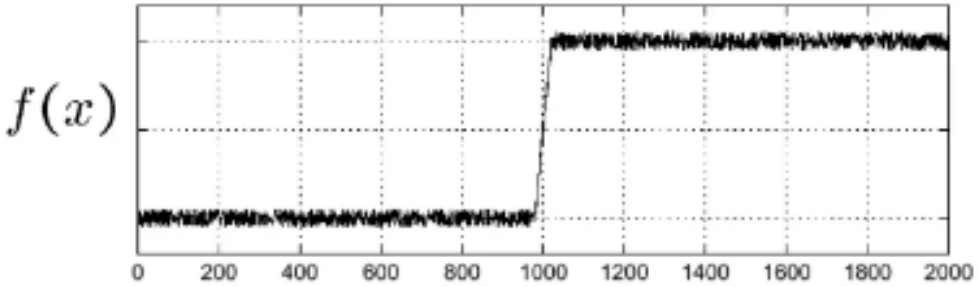
Poor localization
Less Noise sensitive
Good detection

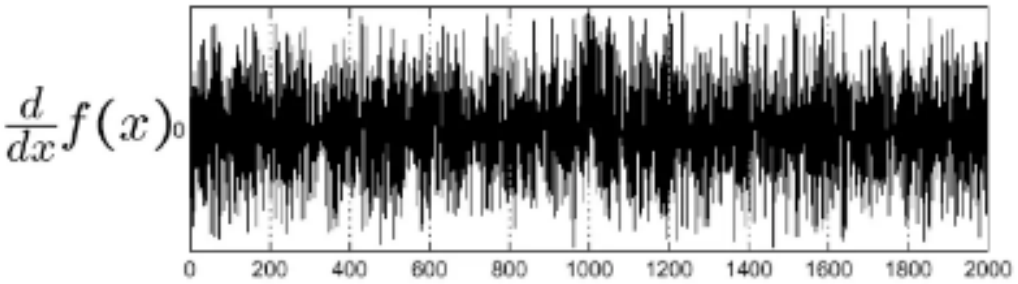
Primitives Detection



Effects of noise

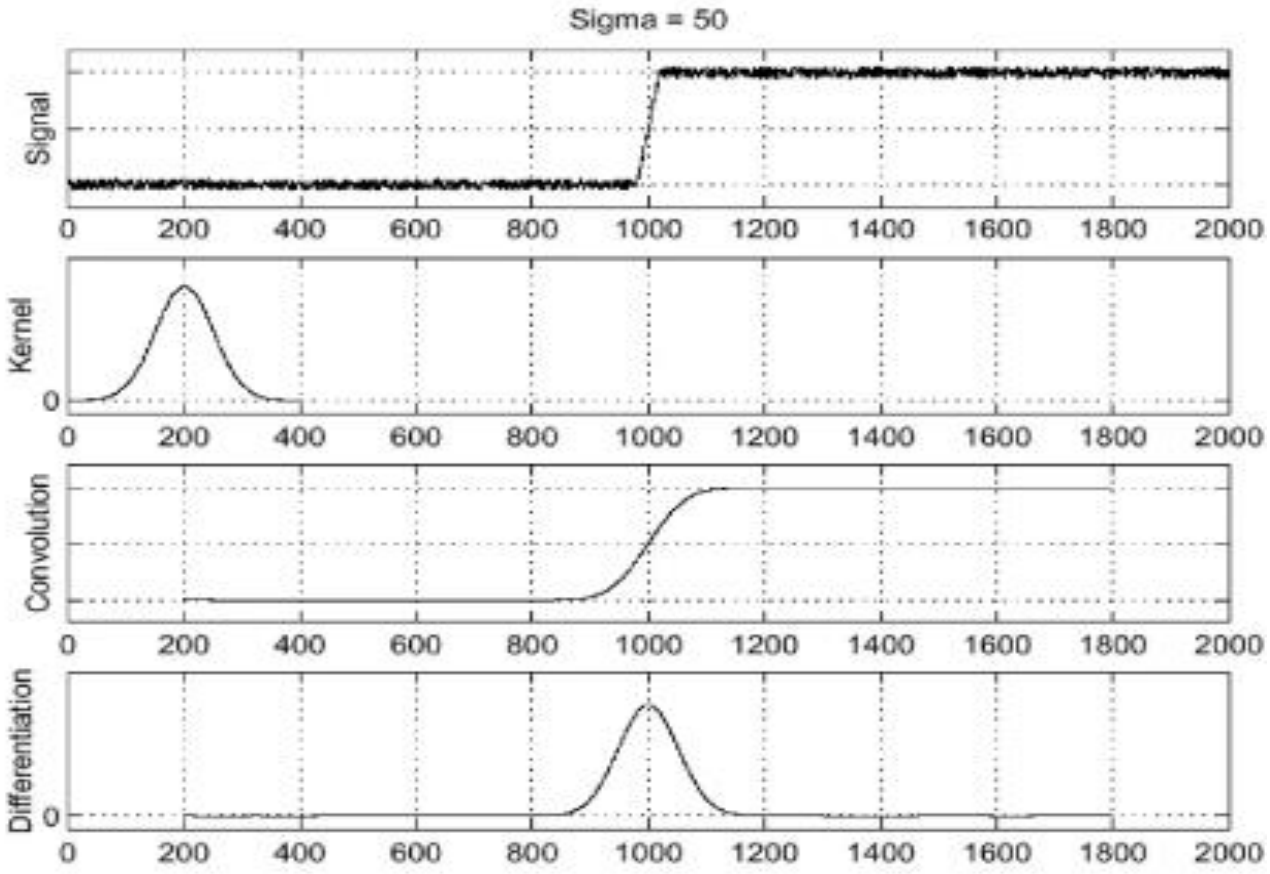
- Consider a single row or column of the image
  - Plotting intensity as a function of position gives a signal





Where is the edge?

Solution: smooth first



f

h

$h \star f$

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

Where is the edge?

Look for peaks in $\frac{\partial}{\partial x}(h \star f)$

Derivative theorem of convolution

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

- This saves us one operation:

f

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

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

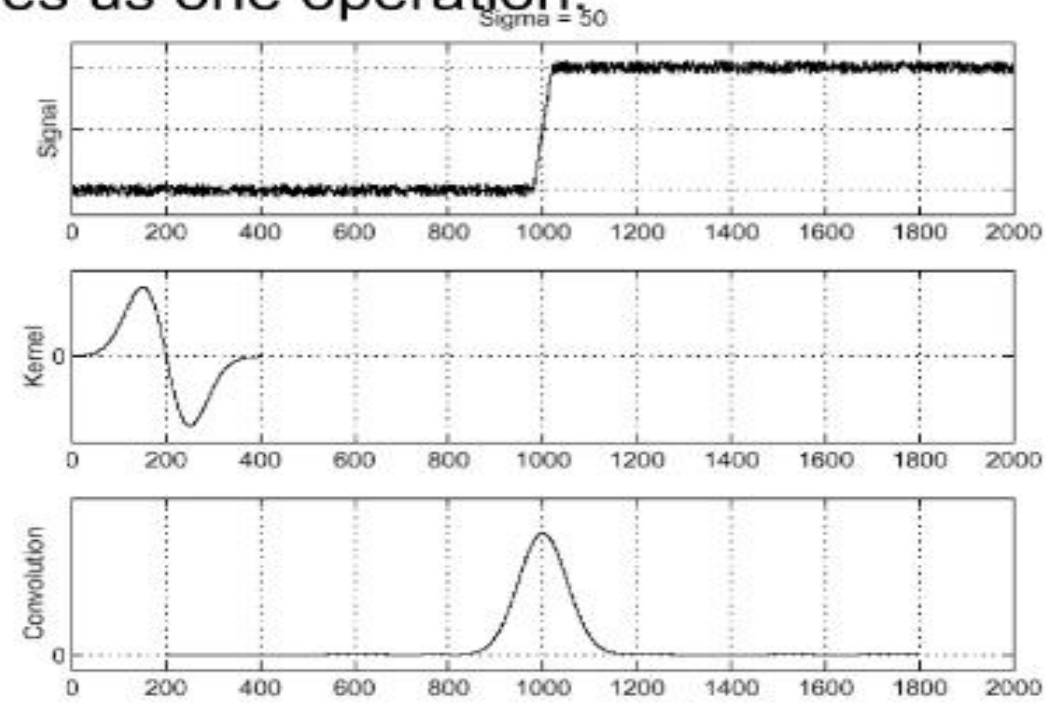


Image Features I

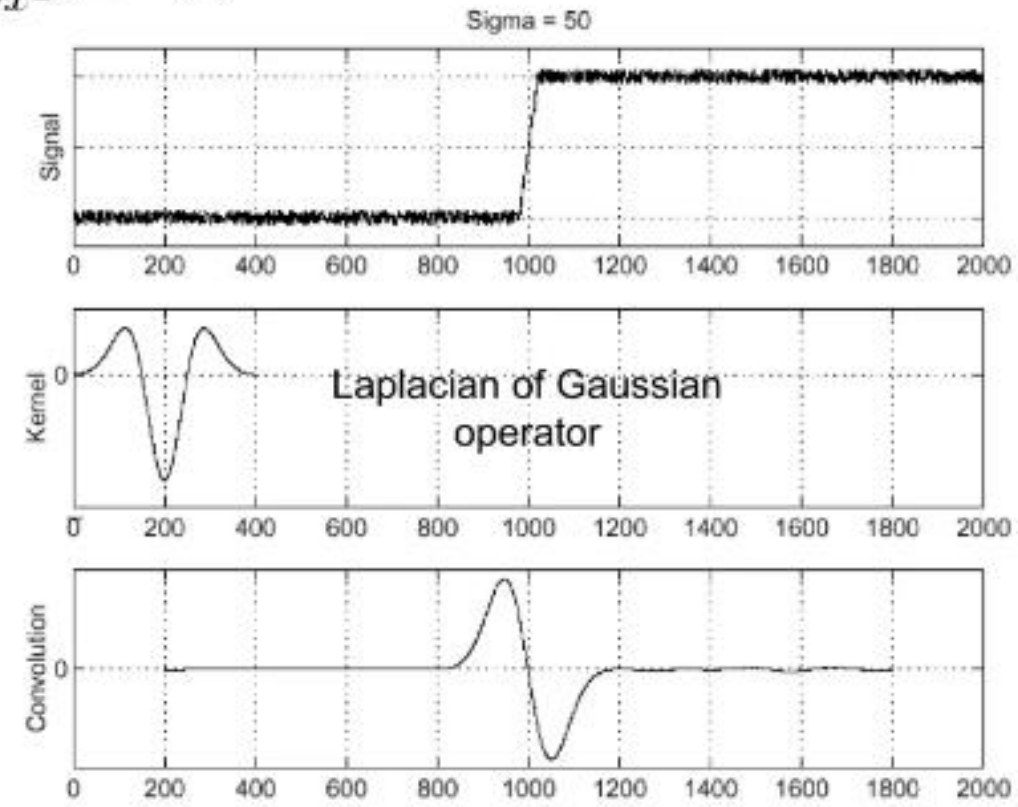
Laplacian of Gaussian (LoG)

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

f

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

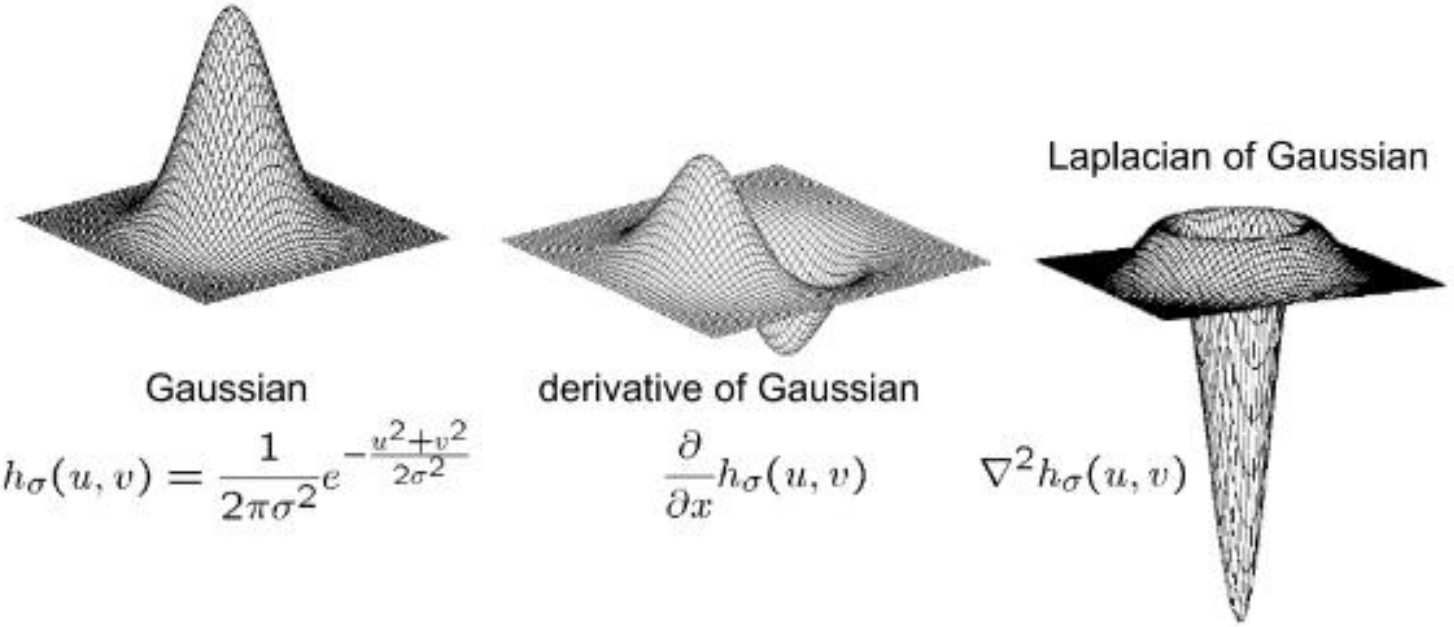
$(\frac{\partial^2}{\partial x^2}h) \star f$



Where is the edge?

Zero-crossings of bottom graph

2D edge detection filters:



Gaussian

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

derivative of Gaussian

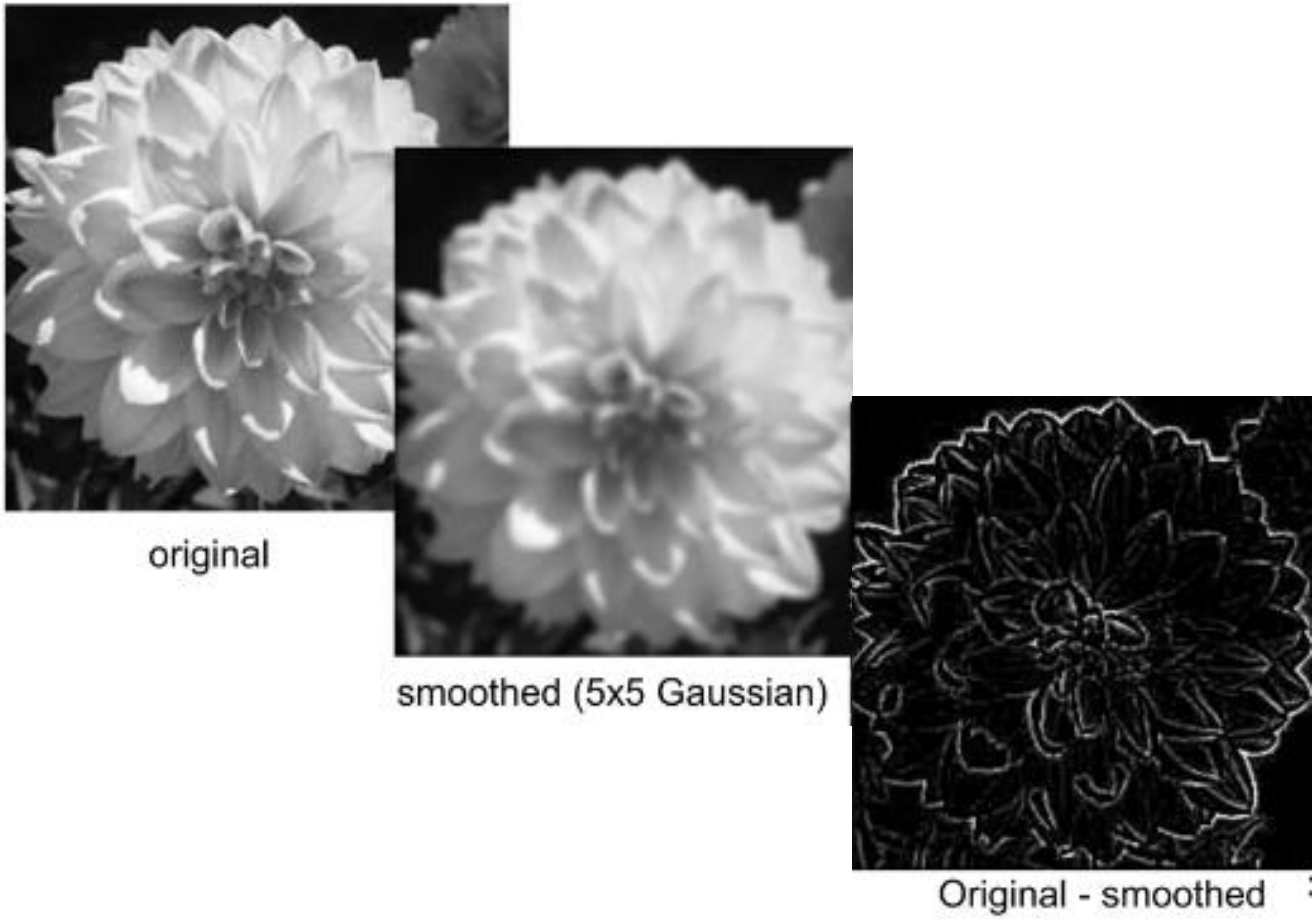
$$\frac{\partial}{\partial x} h_\sigma(u,v)$$

$$\nabla^2 h_\sigma(u,v)$$

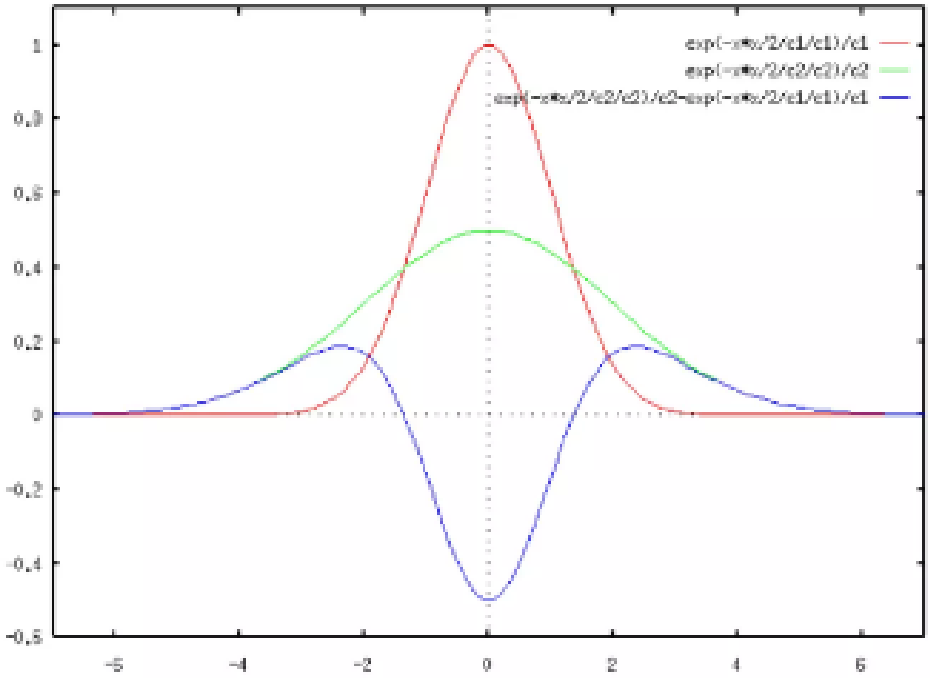
∇^2 is the **Laplacian** operator:

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

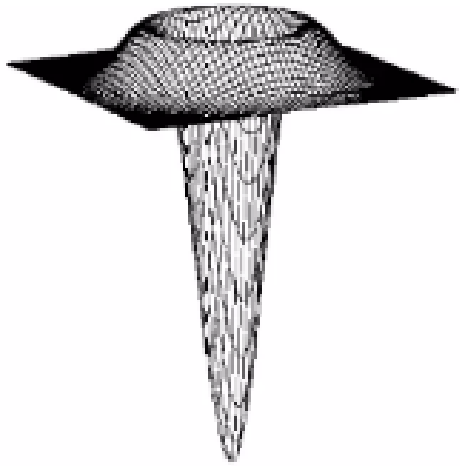
Edge detection by substraction:



Difference of Gaussians (DoG)



DoG



LoG

Primitives Detection

Edge Detection Algorithms

- Explain the steps of **Sobel operator** (gradient approximation).
- Overview of **Canny Edge Detection**:
  - Gradient calculation.
  - Non-maximum suppression.
  - Hysteresis thresholding.

Canny edge detector

- The Canny operator gives single-pixel-wide images with good continuation between adjacent pixels
- It is the most widely used edge operator today; no one has done better since it came out in the late 80s. Many implementations are available.
- It is very sensitive to its parameters, which need to be adjusted for different application domains.

J. Canny, ***A Computational Approach To Edge Detection***, IEEE Trans. Pattern Analysis and Machine Intelligence, 8:679-714, 1986.

Canny edge detector

Steps:

1. **Convolve image** *f* with a Gaussian *G* of scale σ
2. Filter with Sobel kernel in both horizontal (g_x) and vertical direction (g_y)
3. Estimate the **local edge gradient** for each pixel.

$$\Theta = \tan^{-1}\left(\frac{g_y}{g_x}\right)$$

4. Find the location of the edge using zero crossing

$$\frac{\partial^2}{\partial n^2} G * f = 0$$

5. Compute the magnitude of the edge at that location using

$$g = \sqrt{g_x^2 + g_y^2}$$

Canny edge detector: streaking problem

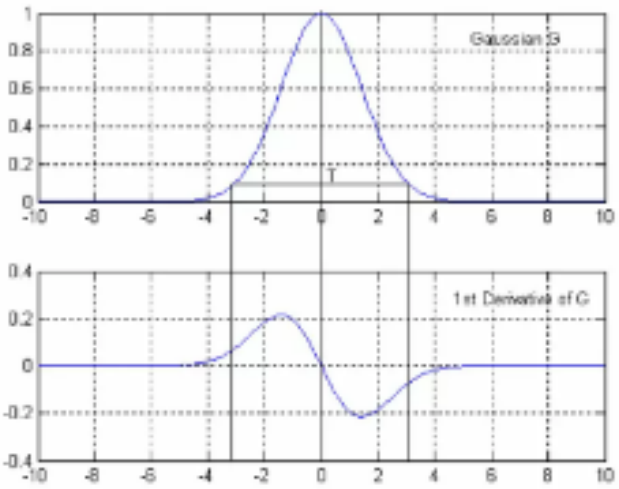
Threshold edges with hysteresis to avoid "streaking problem" (breaking up of the edge contour caused by operator fluctuating above and below the threshold).

Thresholding with hysteresis:

1. Two thresholds are set, T1 and T2, T1<T2
2. Responses above T2 are actual edges
3. Responses below T1 are not edges
4. Responses between T1 and T2 are edges if they are connected to any edges of the strong response.

Guidelines

Sigma	Size of Mask
0.5	3x3
1	5x5
2	9x9
3	13x13
4	19x19



x-mask

15	69	114	69	15
35	155	255	155	35
0	0	0	0	0
-35	-155	-255	-155	-35
-15	-69	-114	-69	-15

y-mask

15	35	0	-35	-15
69	155	0	155	69
114	255	0	-255	-114
69	155	0	155	69
15	35	0	35	15

Sigma=1

Canny Edge Detector

- Guidelines for Use:

  The effect of the Canny operator is determined by three parameters:

  – the width of the Gaussian kernel used in the smoothing phase (σ)

  – the upper threshold of hysterisis

  – and the lower threshold used by the tracker.

Canny Edge Detector

- Usually, the upper tracking threshold can be set quite high, and the lower threshold quite low for good results.
- Setting the lower threshold too high will cause noisy edges to break up.
- Setting the upper threshold too low increases the number of spurious and undesirable edge fragments appearing in the output.

Effect of σ (Gaussian kernel size)

original | Canny with $\sigma = 1$ | Canny with $\sigma = 2$

The choice of σ depends on desired behavior

- large σ detects large scale edges
- small σ detects fine features

30

Canny Edge Detector

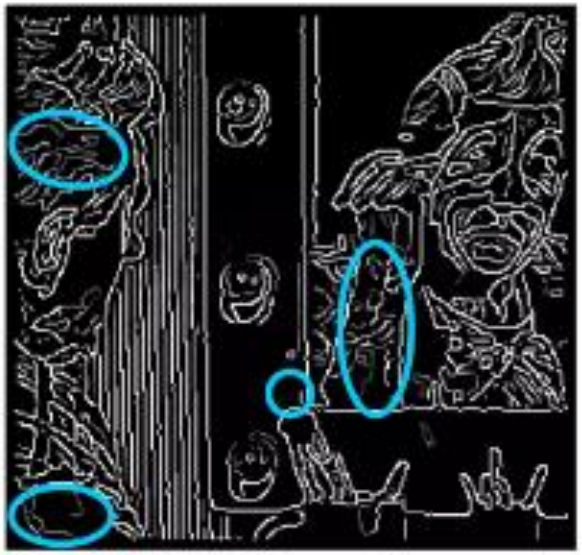
Original image

St. dev. 1.0
Upper, lower thr. of 255 and 1

Notice:

- Most of the major edges are detected
- lots of details have been picked out well
- note that this may be too much detail for subsequent processing.

Canny Edge Detector



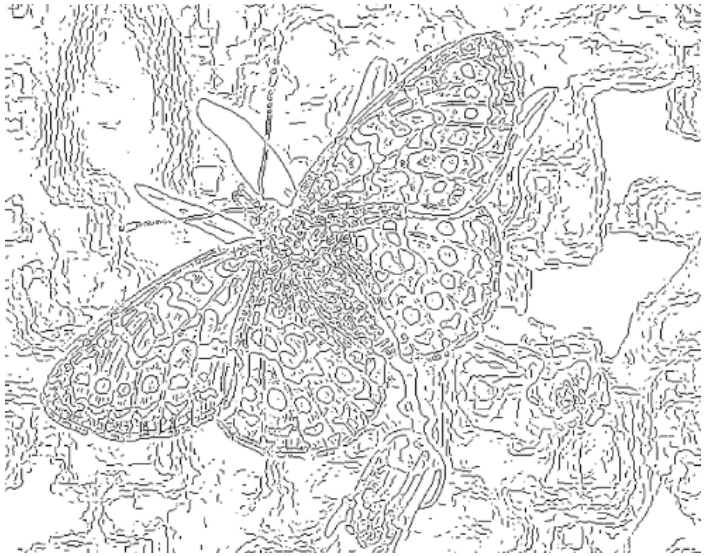
lowering the upper thr. to 128.
The lower threshold and the
Gaussian st. dev. remains the same

Gaussian used has a st. dev. 2.0.
Same thresholds

- faint edges are detected along with some short `noisy' fragments

- Much of the detail on the wall are no longer detected,
- but most of the strong edges remain.
- Edges are also smoother and less noisy.

Primitives Detection

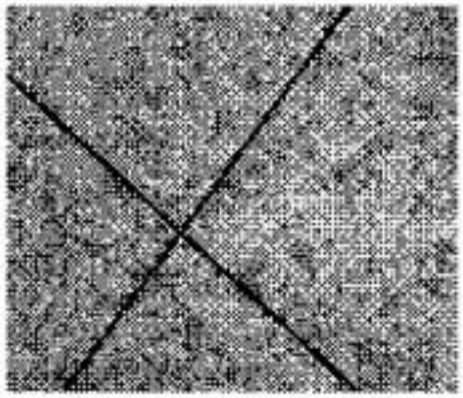


Fine scale high threshold

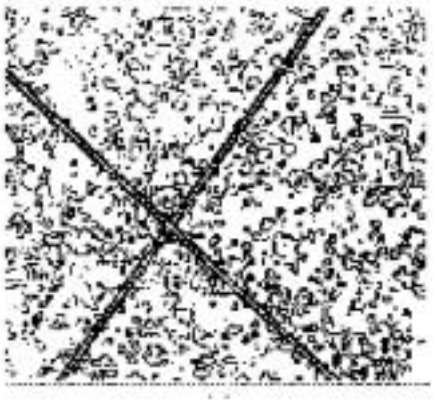
Coarse scale high threshold

Coarse scale low threshold

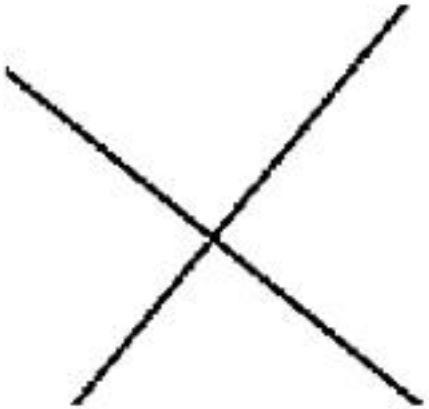
Finding Curves With Specific Shape



Original Image



Edge Image



Target Lines

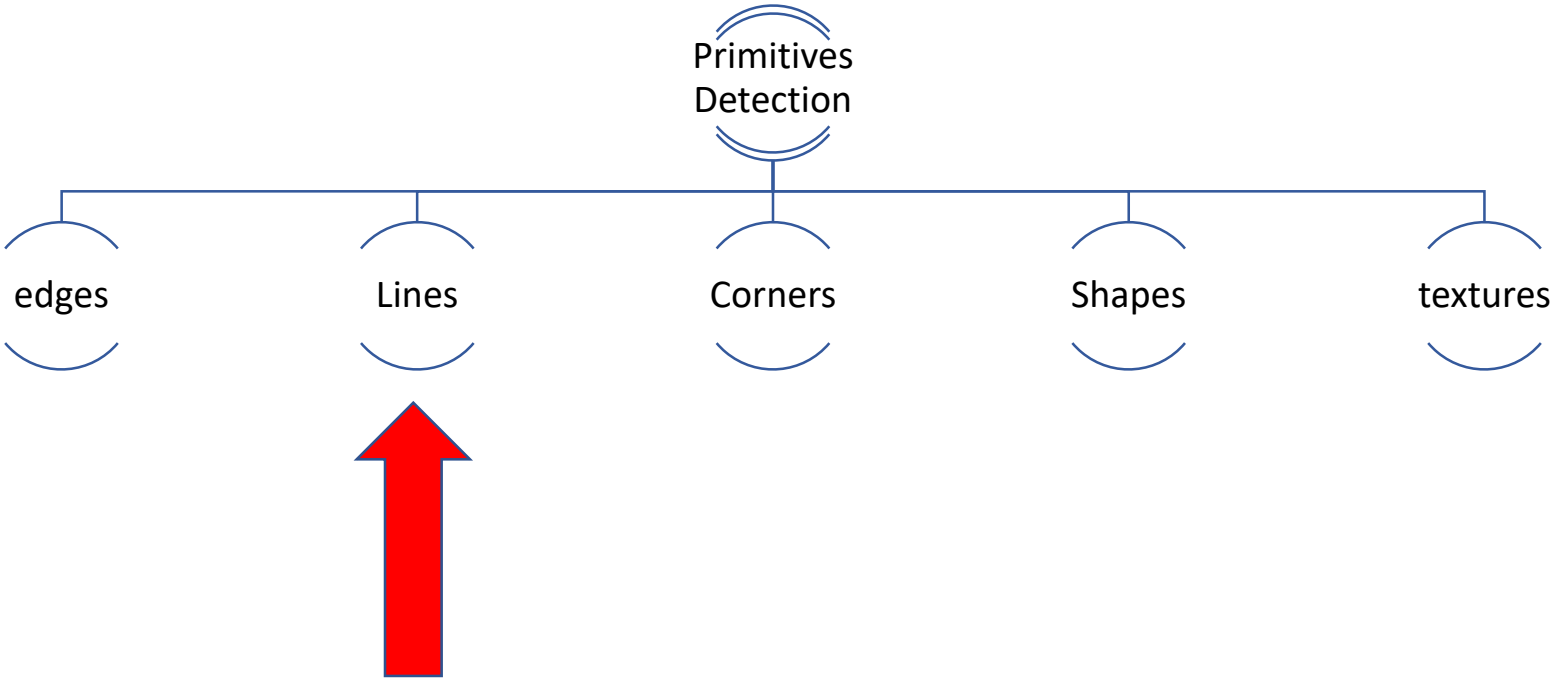
Primitives Detection

An edge is not a line...

How can we detect ***lines*** ?

Primitives Detection

2/ Lines



Primitives Detection

Key Primitives Detected in Computer Vision:

3/ Lines:

- **Description**: Lines are continuous edges and are important for identifying linear structures in an image, such as roads, borders, or text.
- **Detection Methods**: Hough transform is often used to detect straight lines.
- **Application**: Useful in applications like road lane detection, document analysis, and architectural vision systems.

Finding lines in an image

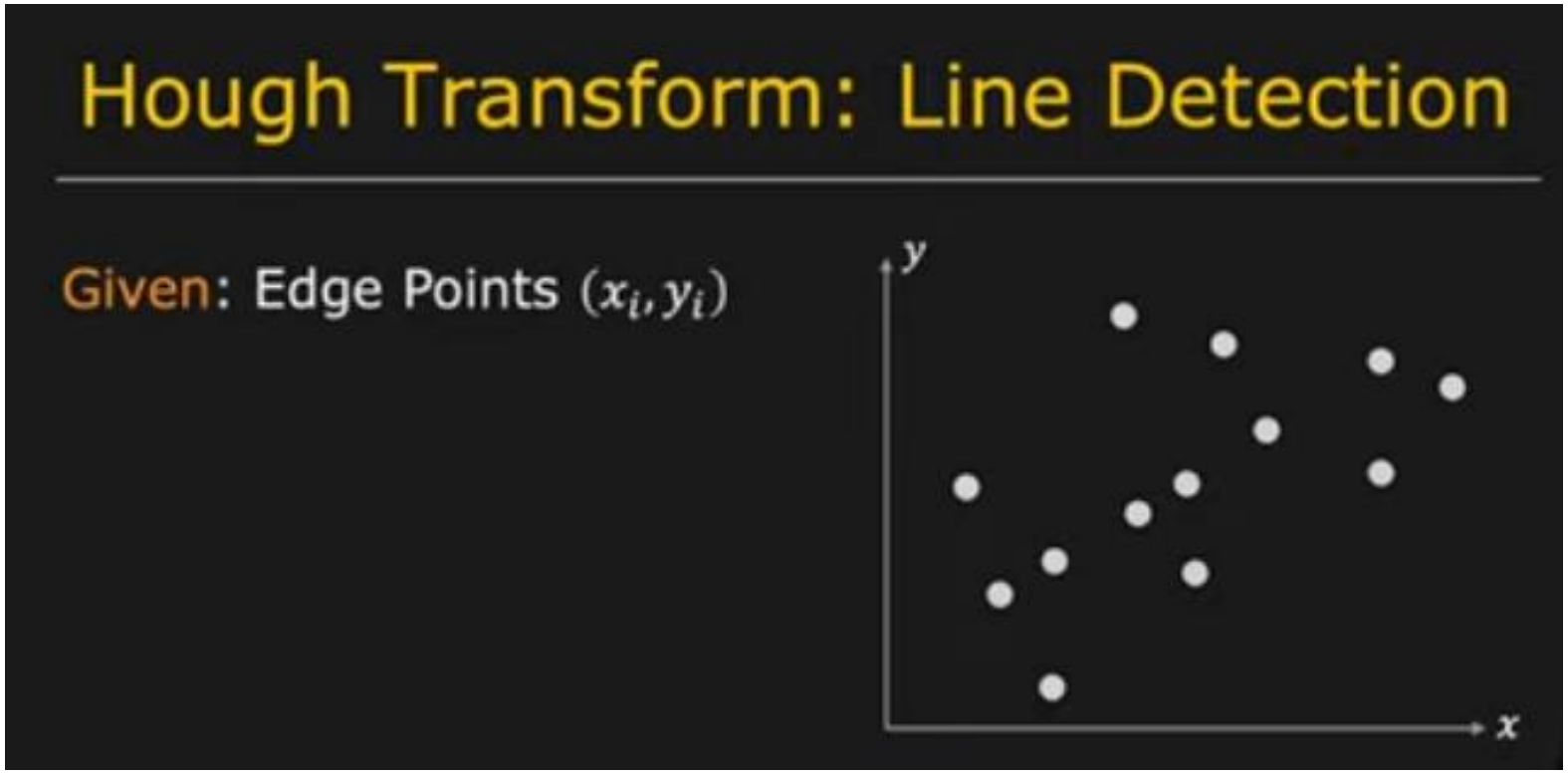
- Option 1:
  - Search for the line at every possible position/orientation
  - What is the cost of this operation?
- Option 2:
  - Use a voting scheme: Hough transform

Primitives Detection

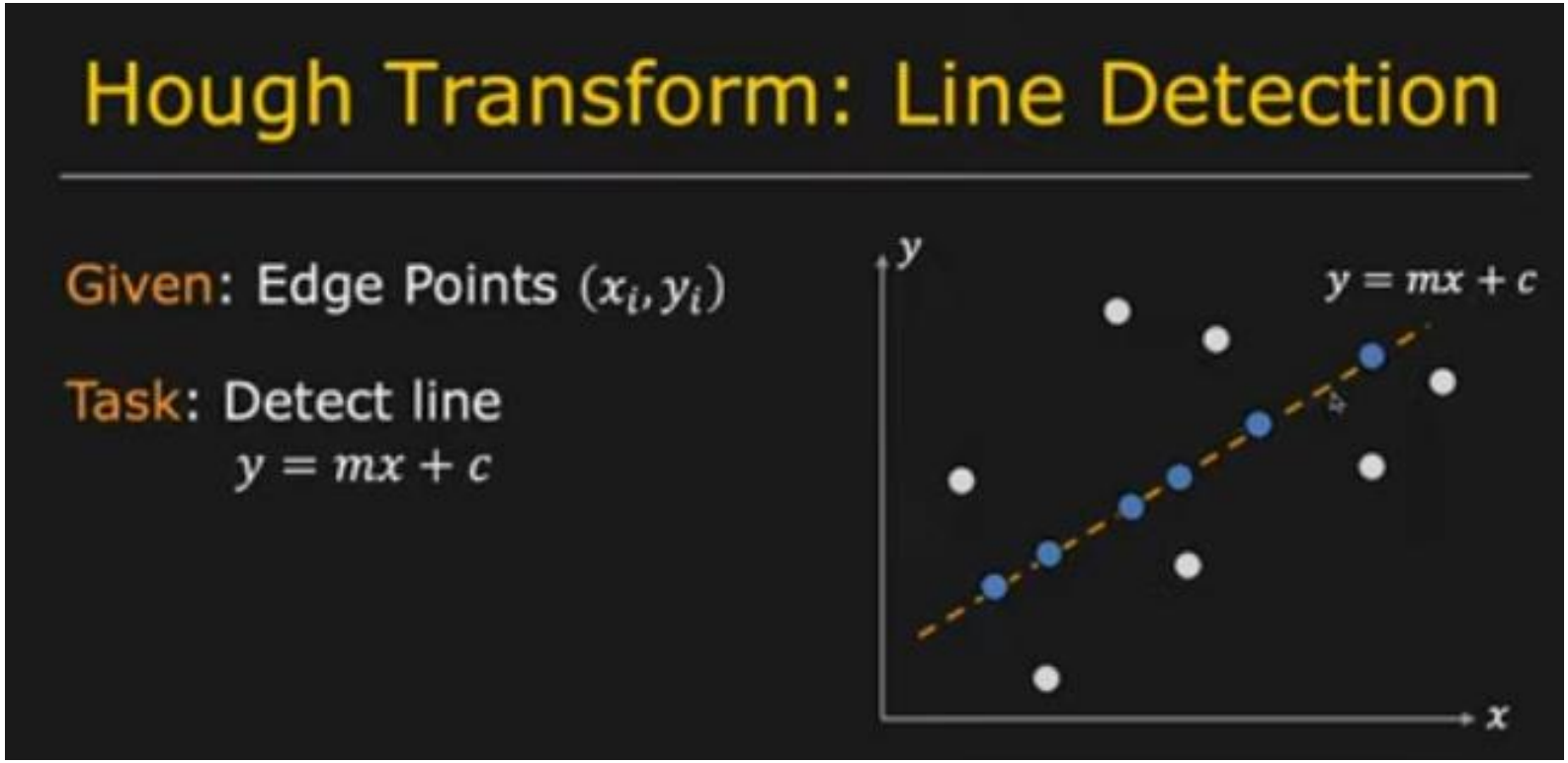
Hough transform ?

the key idea is to map points from the image space to a parameter space, where the target shapes form clusters, making them easier to detect.

Primitives Detection

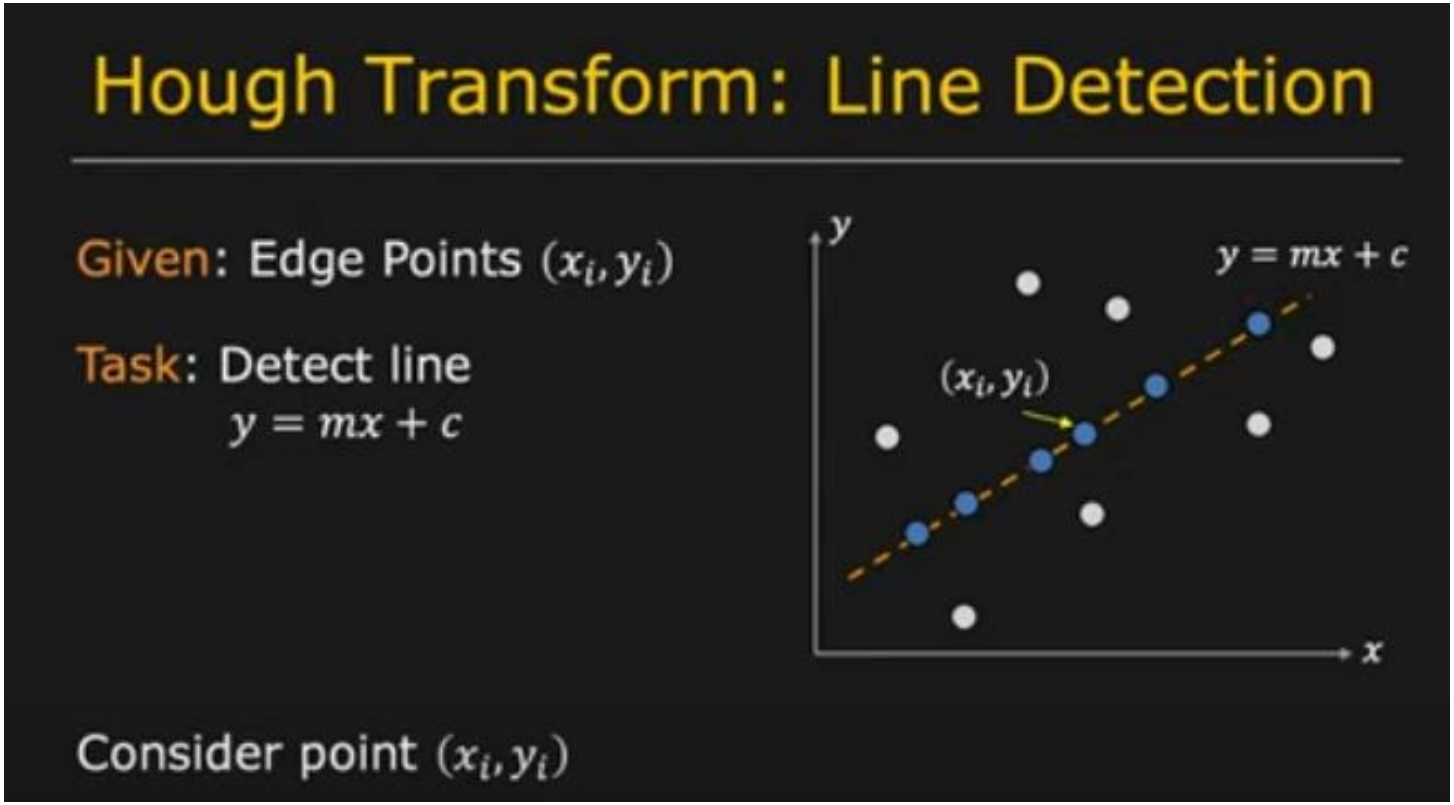


Primitives Detection



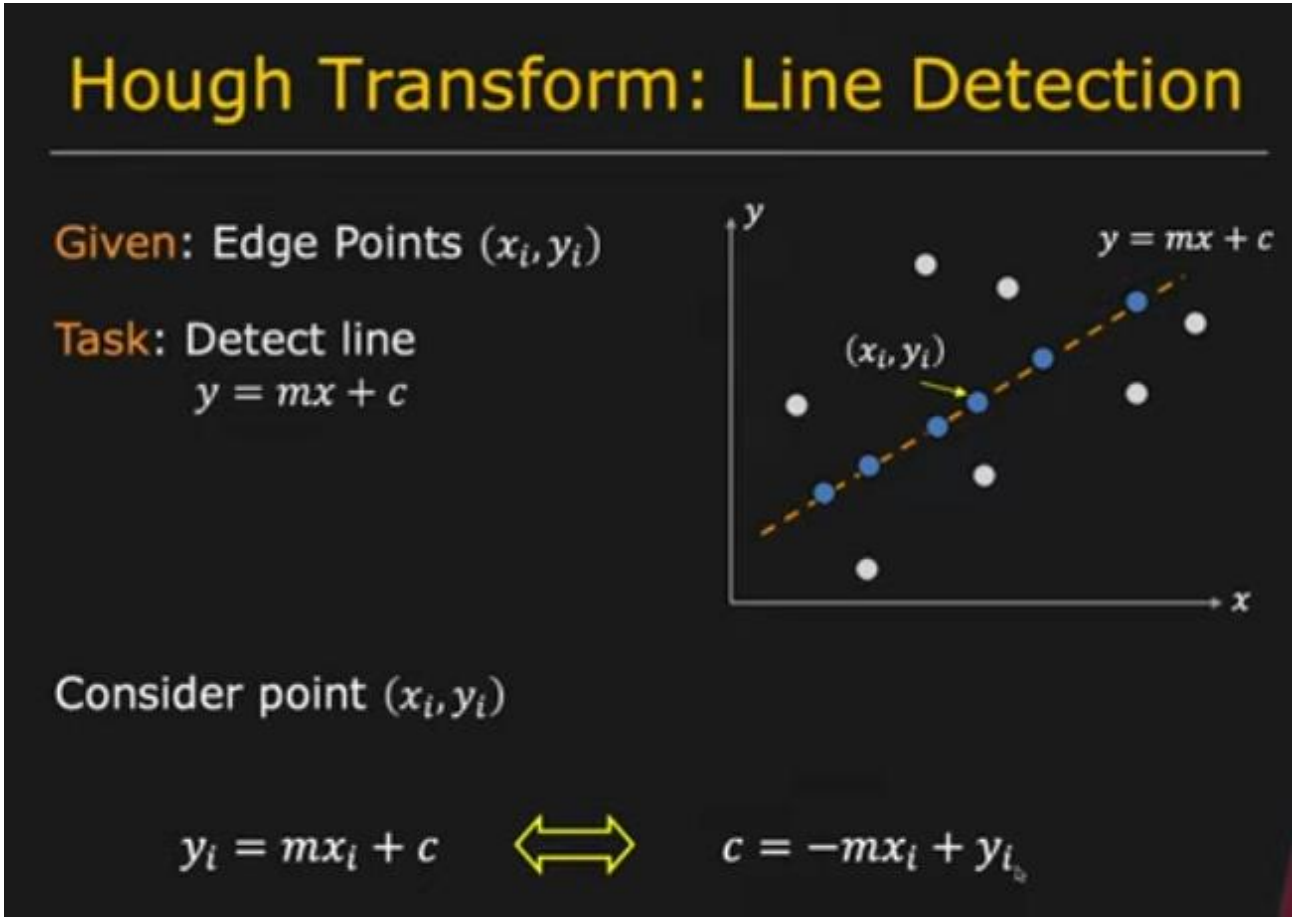
m: the slope
c : the intercept

Primitives Detection



Let consider one point on this traight line

Primitives Detection

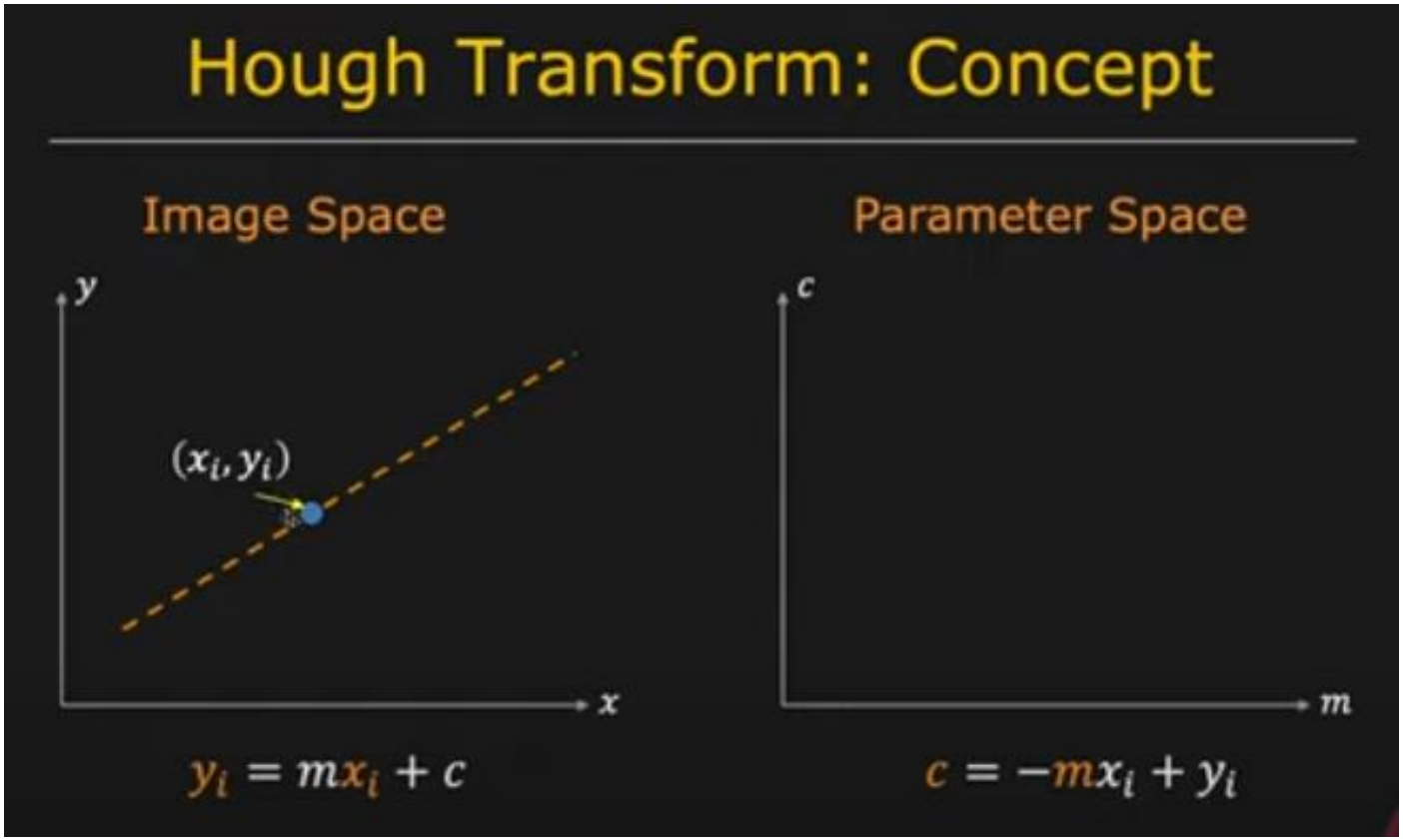


We can Rewrite the point equation

Primitives Detection

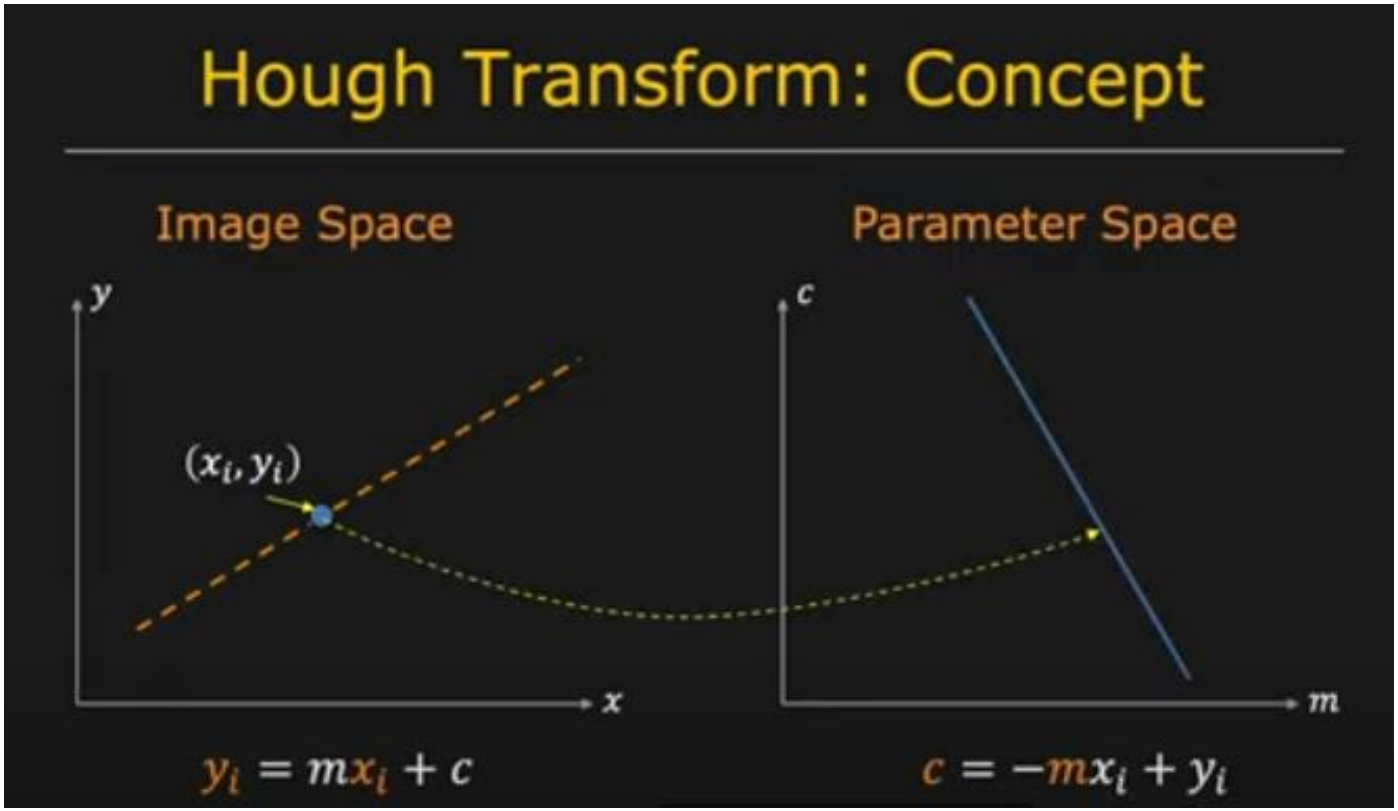
Rewrite the point equation→Xy is the image space and m c is the parameter space

Primitives Detection



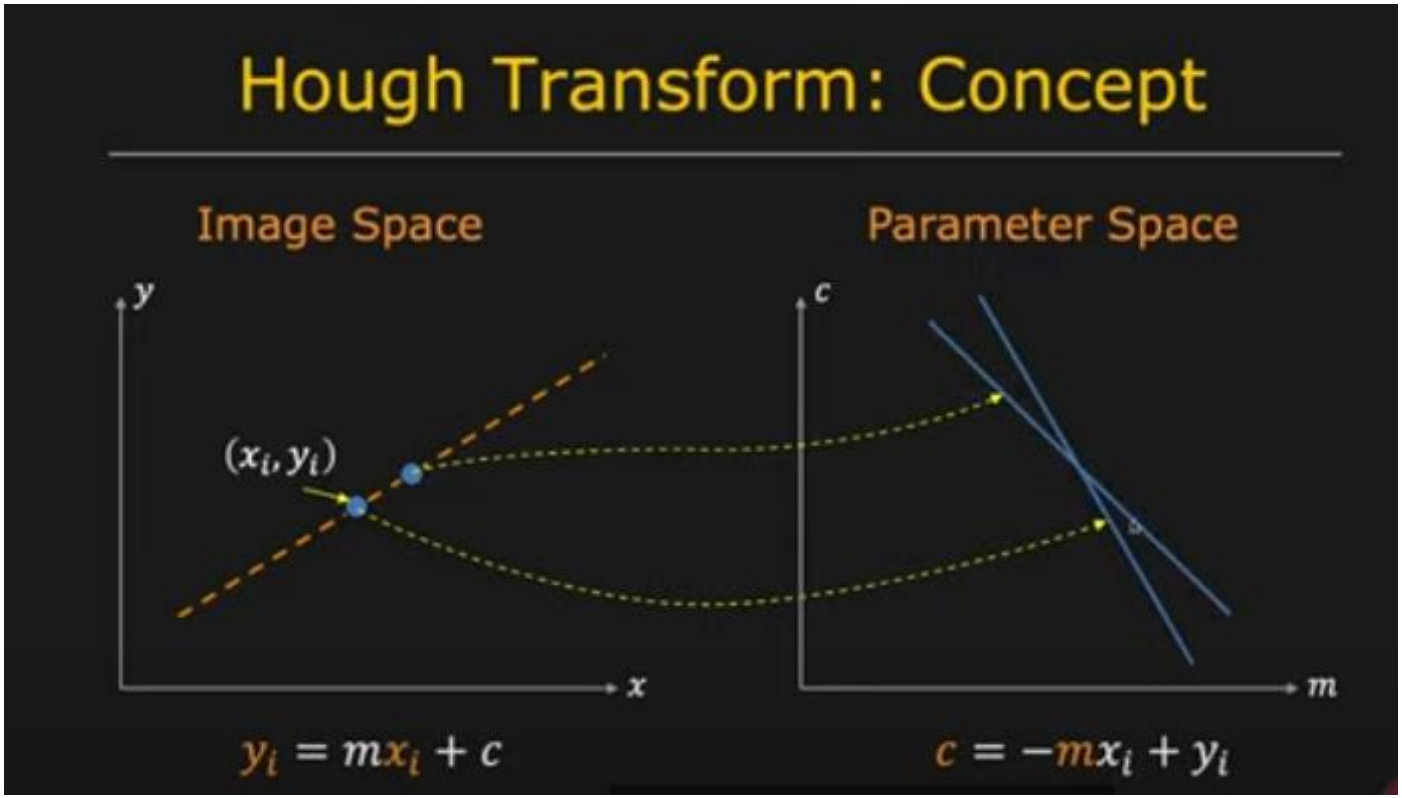
Lets consider a line in the image space, and a particular point that lie on this line

Primitives Detection



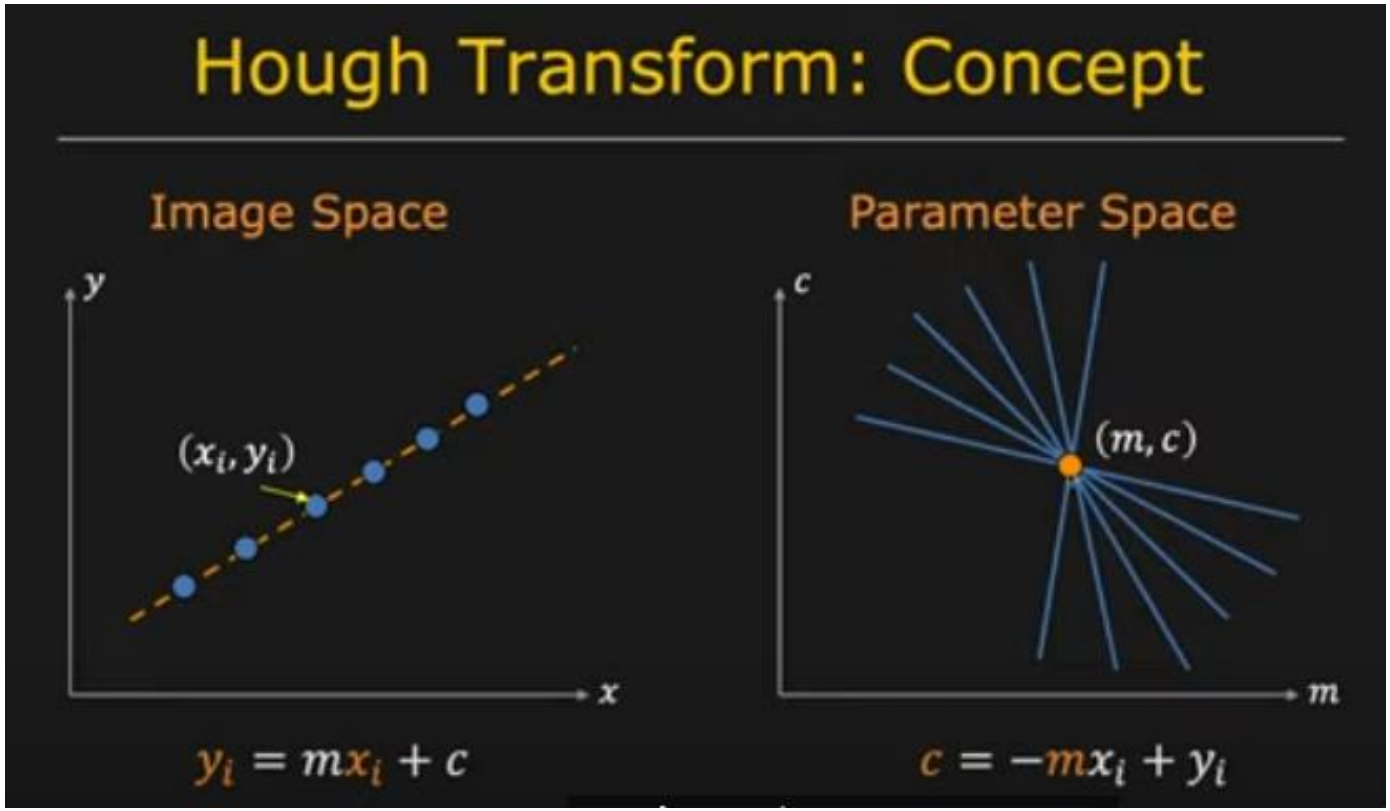
A point in the image space is a line in the parameter space

Primitives Detection



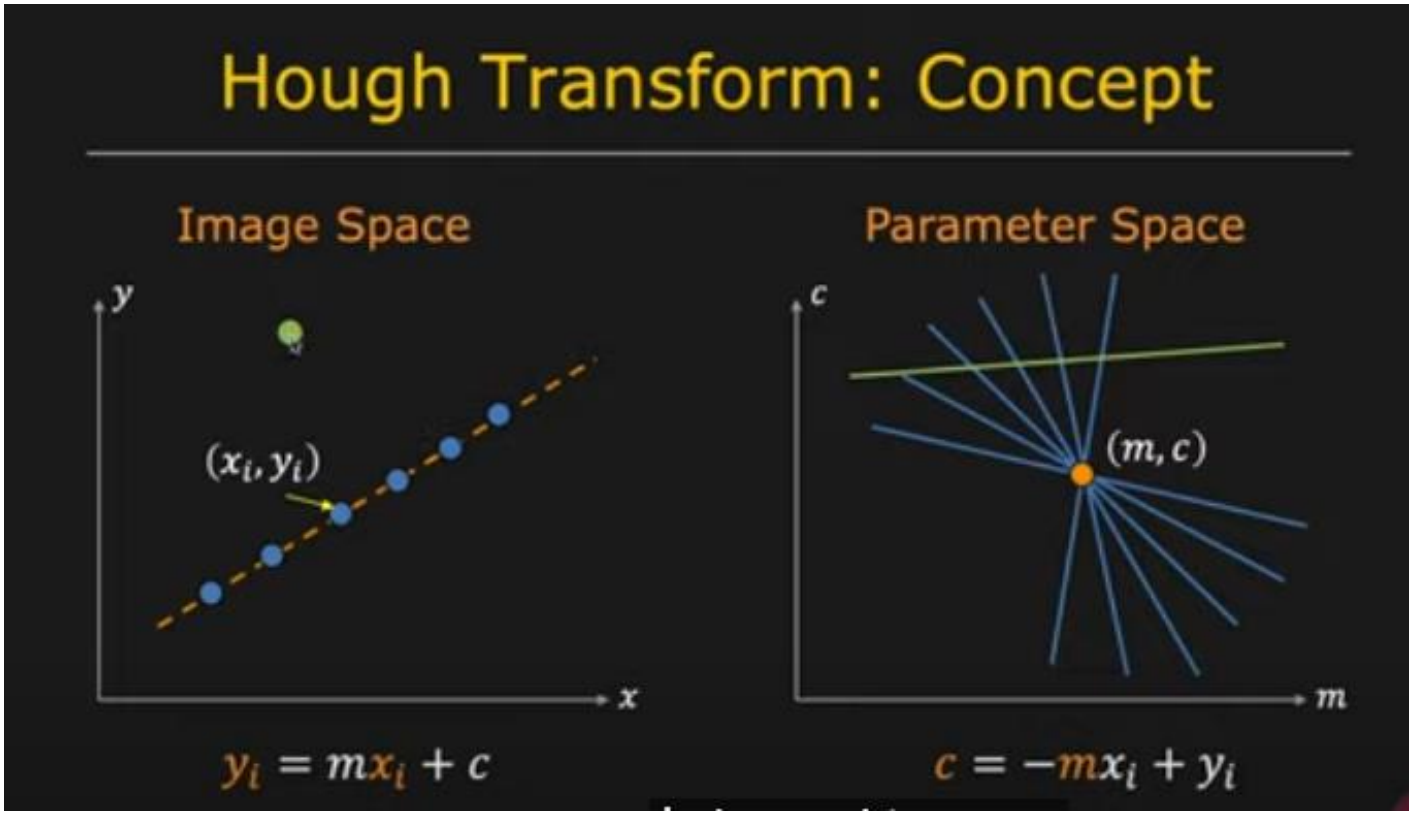
Another point in the image space is another line in parameter space

Primitives Detection



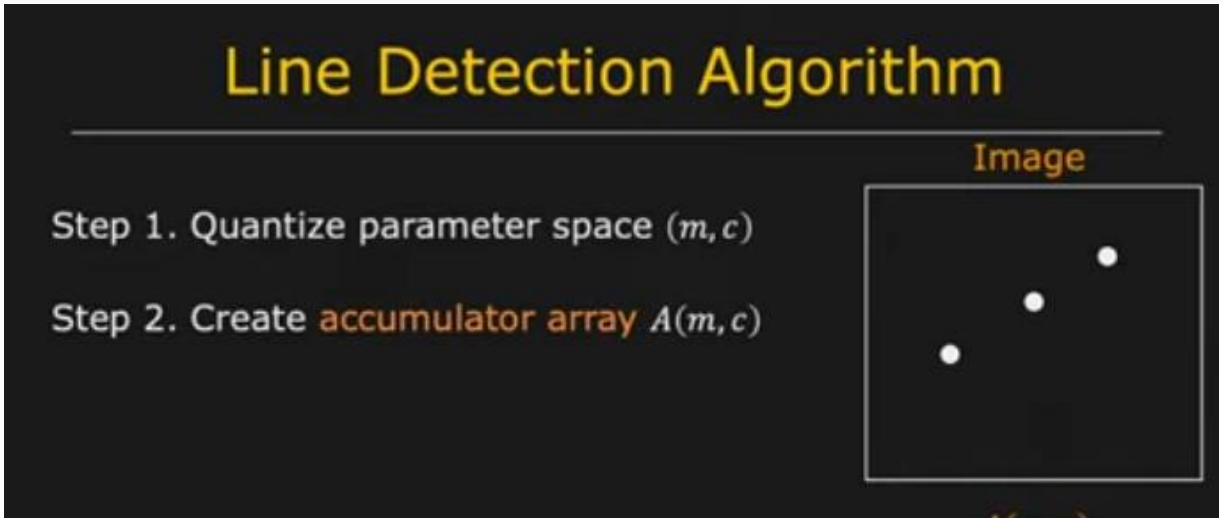
more points belonging to the same line in the image space are more lines in parameter space having one intersection point, wich is (m,c)

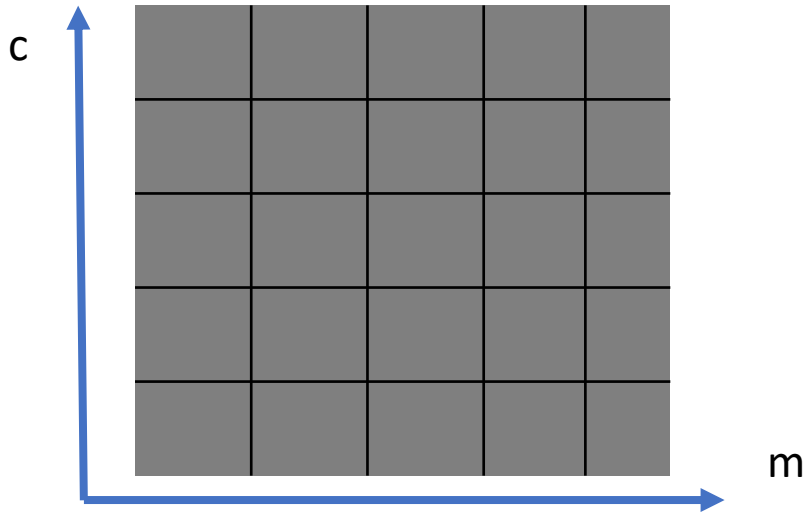
Primitives Detection



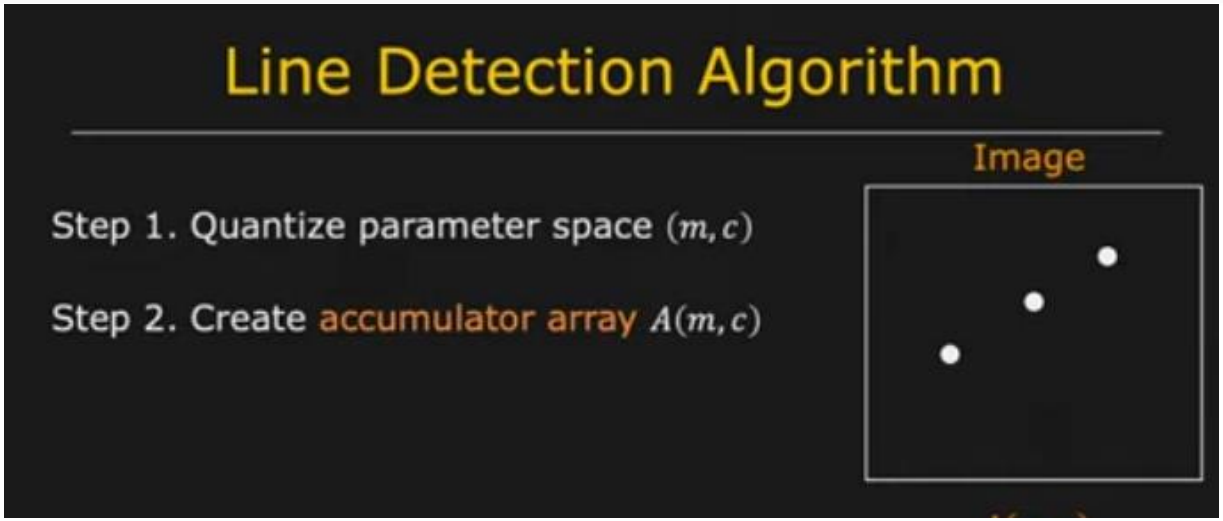
If we take another point (the green point over here) in the image space, that will be a line (the green one) in the parameter space not passing through (m,c)

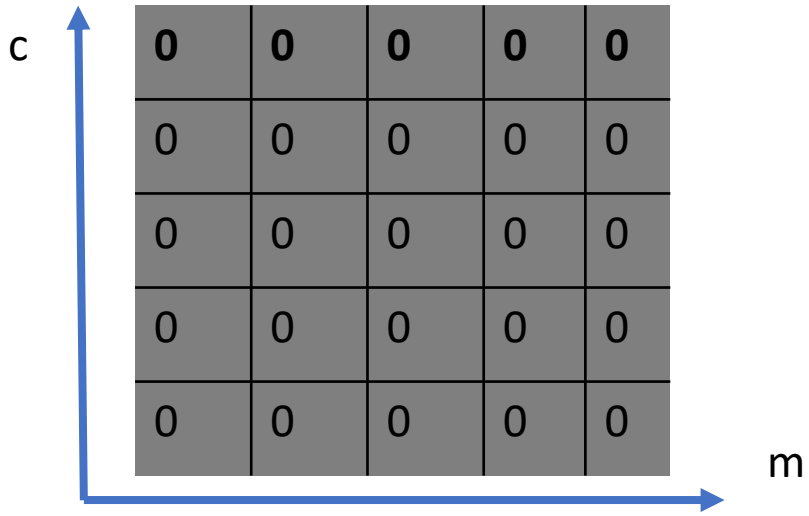
Primitives Detection



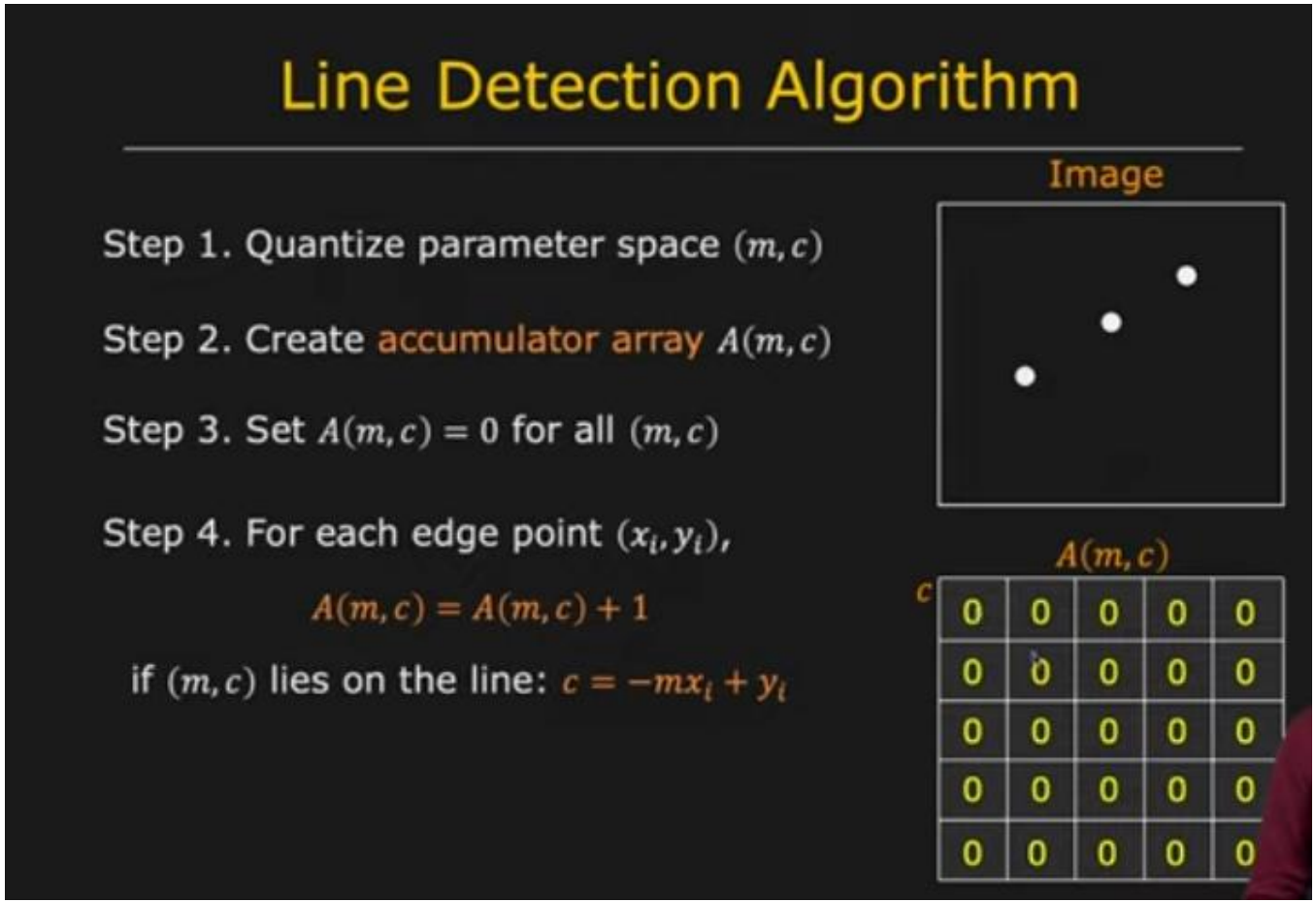


The line detection algorithm use the following steps



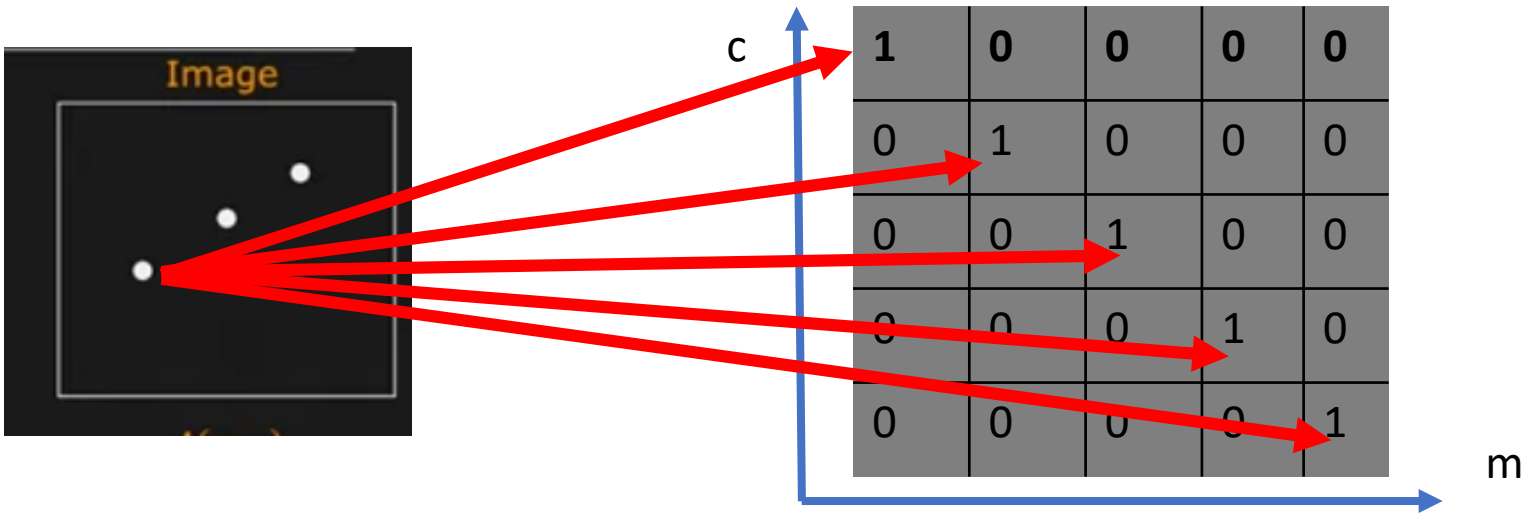


Initialize A(m,c) to 0



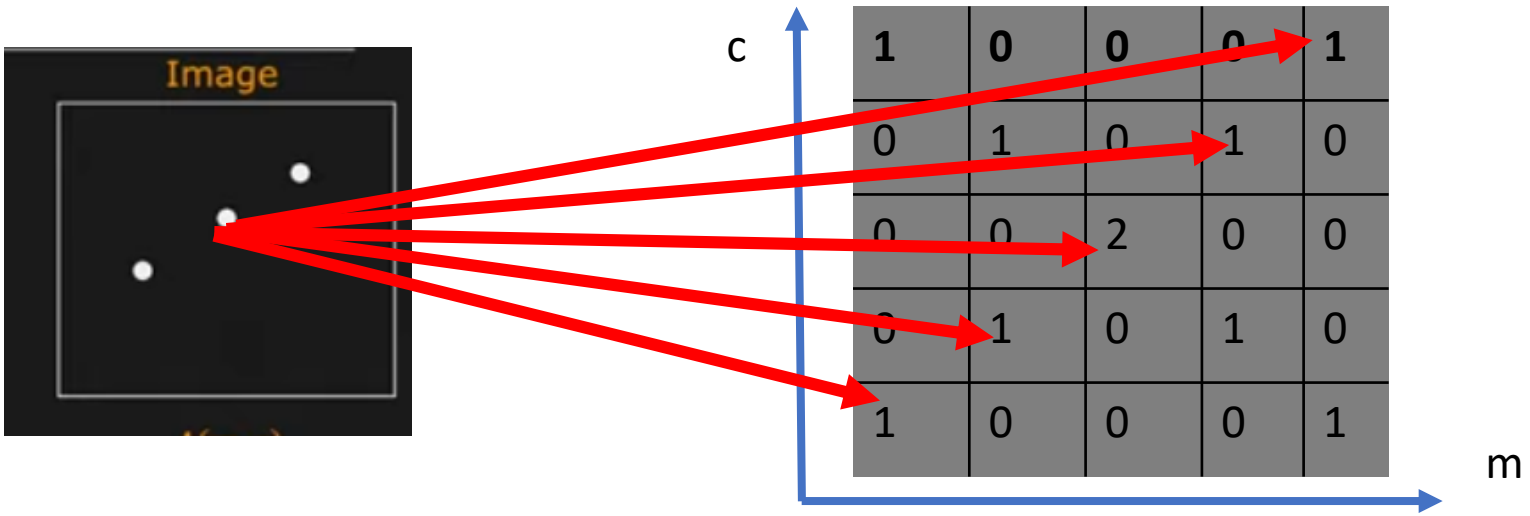
For each edge point, We do incrementation if (m,c) lies on the line

Primitives Detection



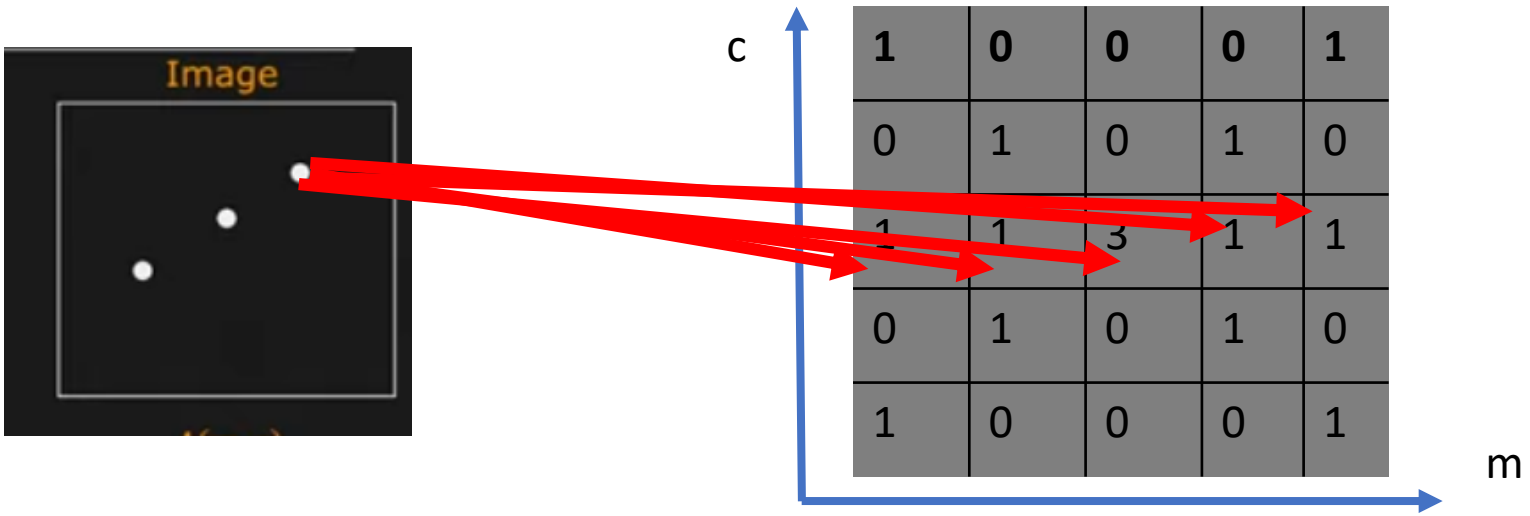
Incrementation for the first edge point

Primitives Detection

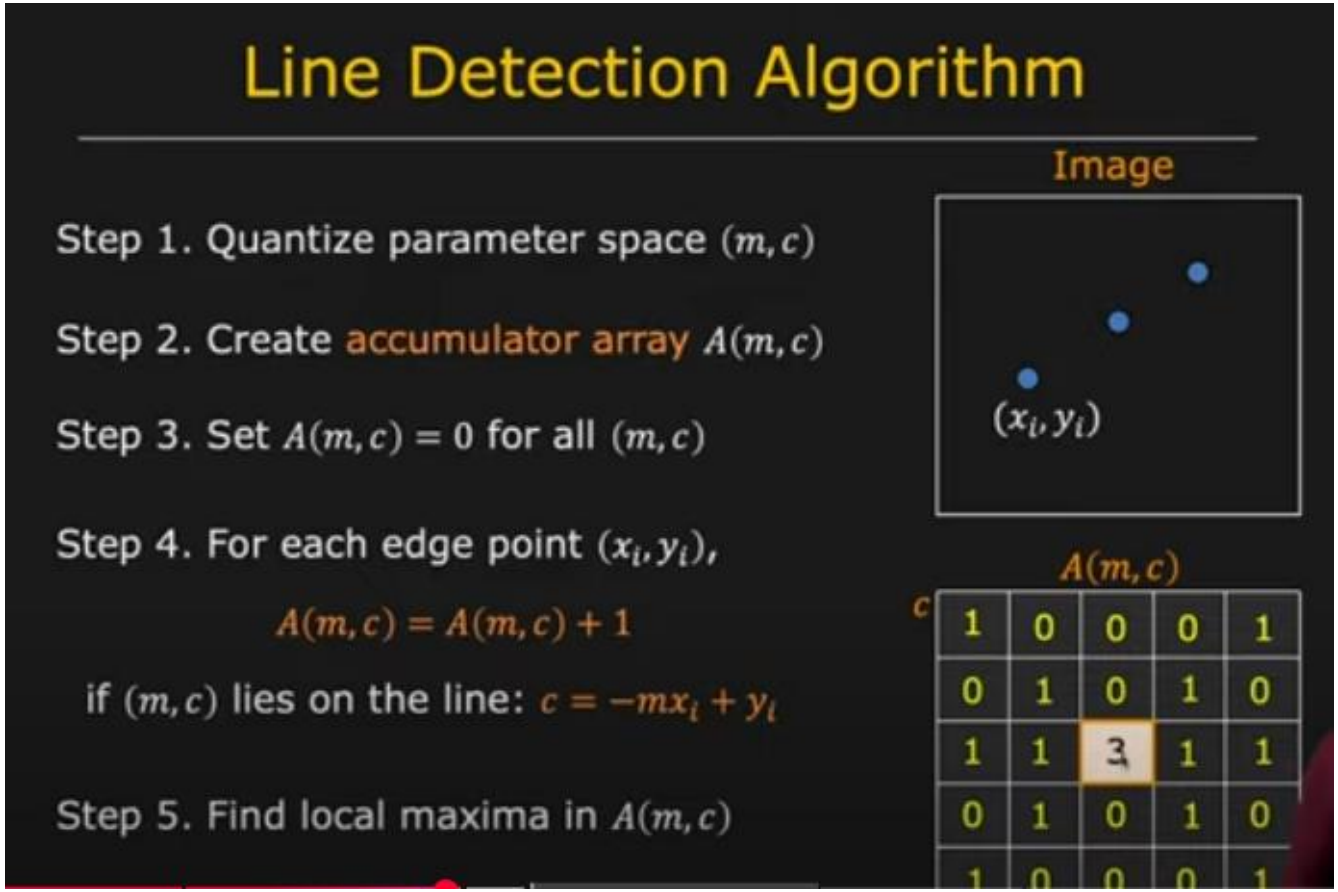


Incrementation for the second edge point
The intersection point has a vote of 2

Primitives Detection

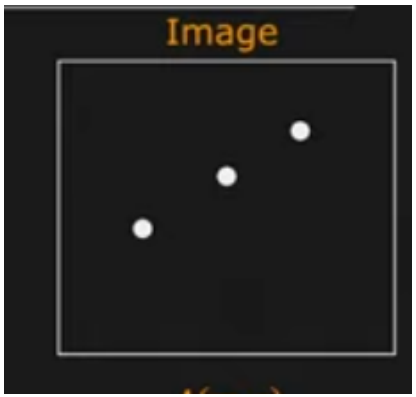


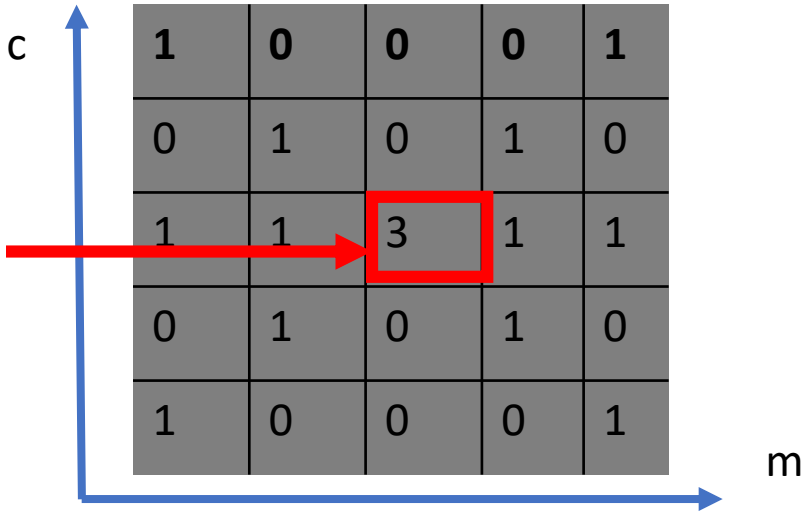
Incrementation for the third edge point
The intersection point has a vote of 3



Step 5: Find local maxima in A(m,c)

Primitives Detection

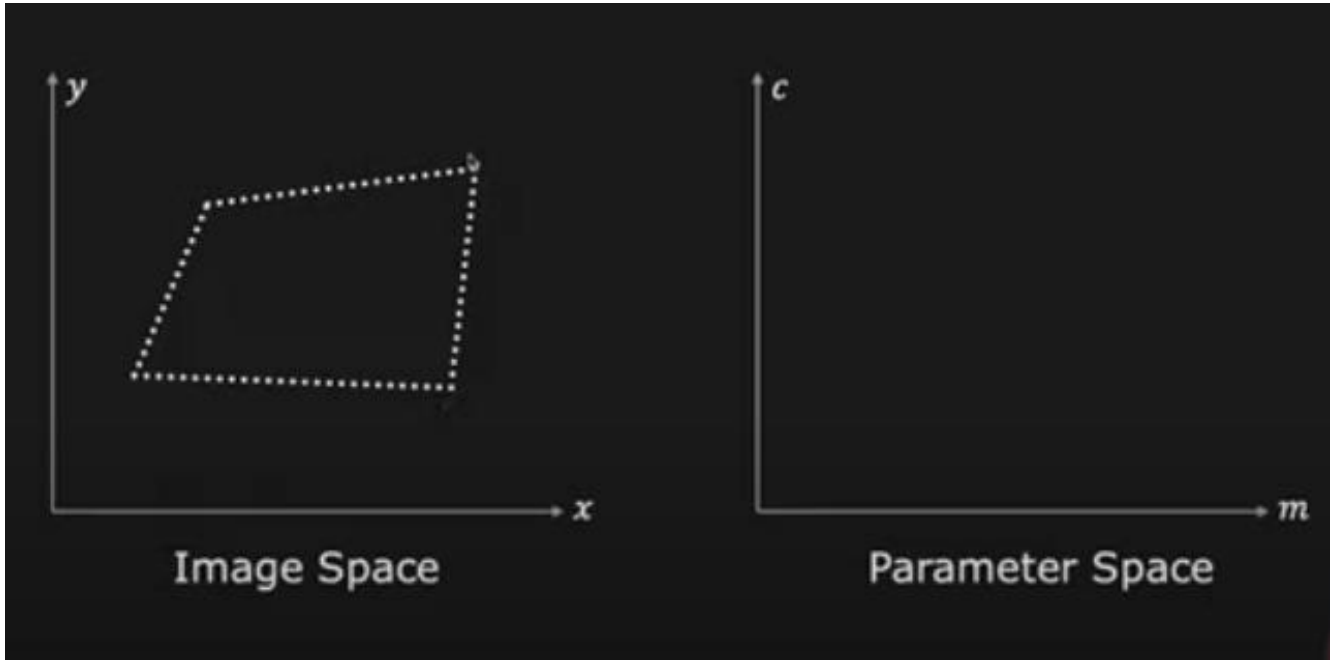




Step 5: Find local maxima in A(m,c)

Primitives Detection

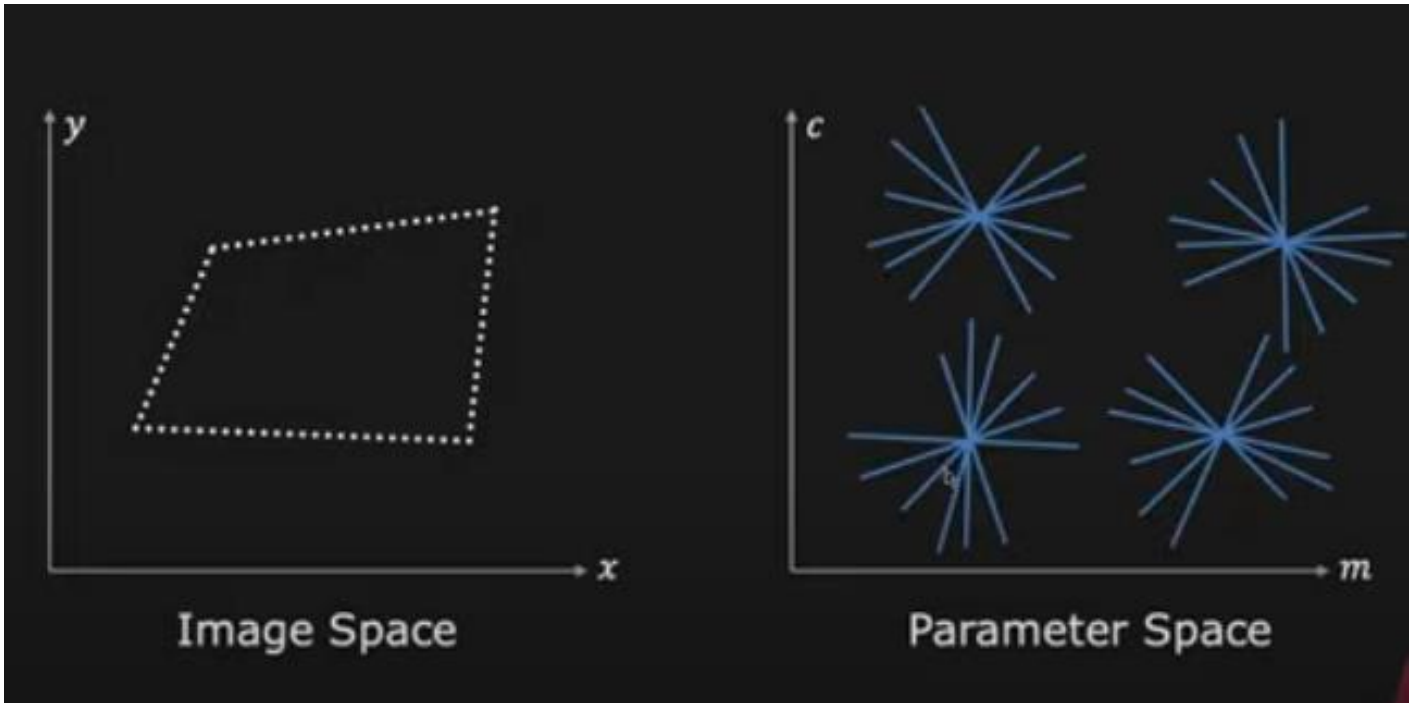
Example of multiple line detection



Four segments in image space

Primitives Detection

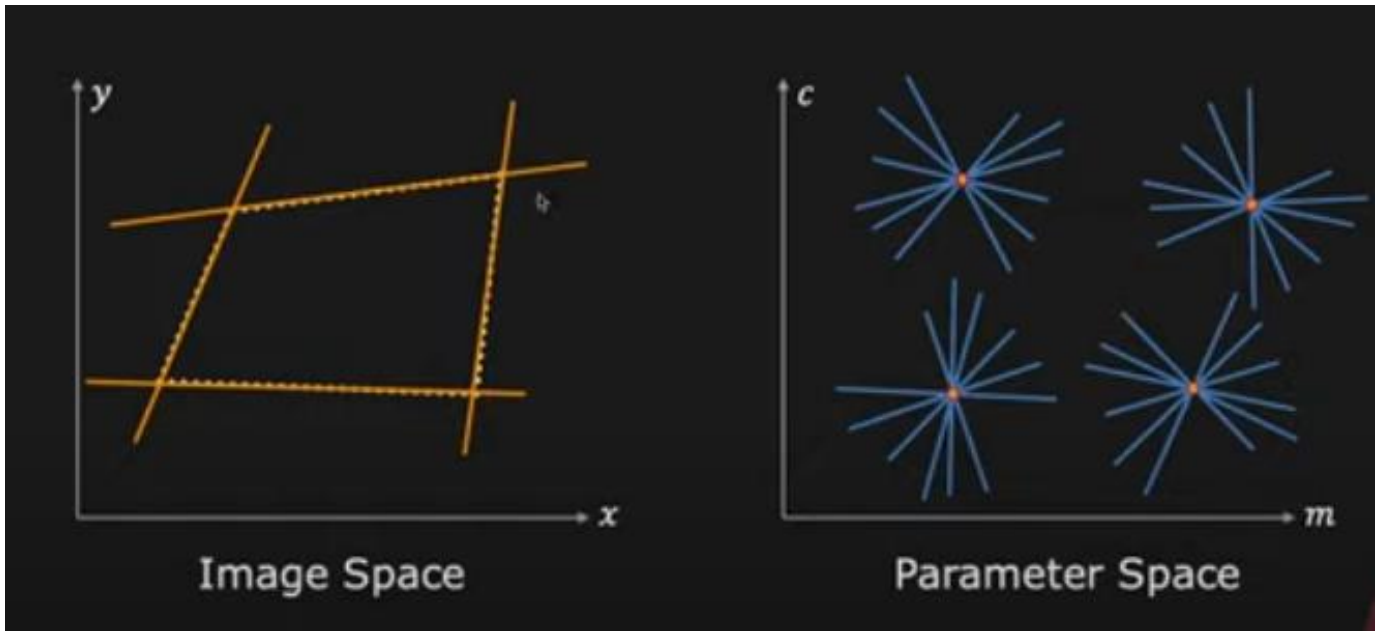
Example of multiple line detection



Four segments in image space→ four intersections in parameter space

Primitives Detection

Example of multiple line detection



Four segments in image space→ four intersections in parameter space→these intersection points correspond to the four lines that pass through these points in image

Hough Transform Mechanics

How big should the accumulator cells be ?

Too big

Different lines may be merged

Too small

Noise causes lines to be missed

Hough Transform Mechanics: solution

How many lines?

- ✓ Count the peaks in the accumulator array
- ✓ Treat adjascent peaks as a single peak

Handling inaccurate edge locations?

Increment patche in accumulator rather than single point

Hough Transform complexity

- **Goal:**
  To find a curve of specific shape (line, circle, … etc) in an edge image.
  - Edges need not be connected
  - Key Idea: Edges VOTE for the possible curve

- Computational Complexity:
  With n points to find the line connecting each two:

  Complexity = n x (n-1)

  $= n^2$

  Hough transform is computationally attractive!

Image and Hough Spaces

Equation of Line: $y = mx + c$

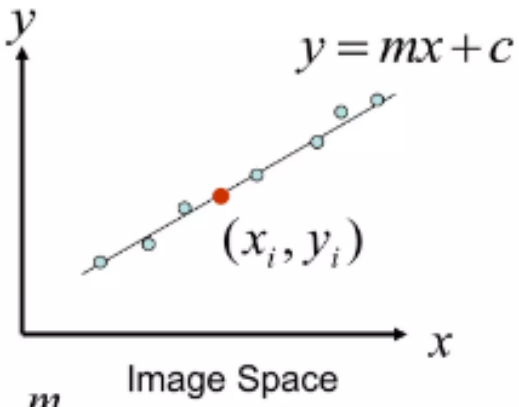
Find: (m, c)

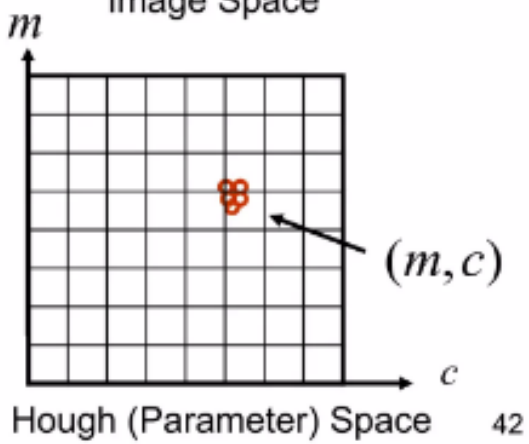
Consider point: (x_i, y_i)

$y_i = mx_i + c \quad or \quad c = -x_i m + y_i$

- Every pair of points will form a line *(m, c)* and add a vote in the cell indexed by *(m, c)*

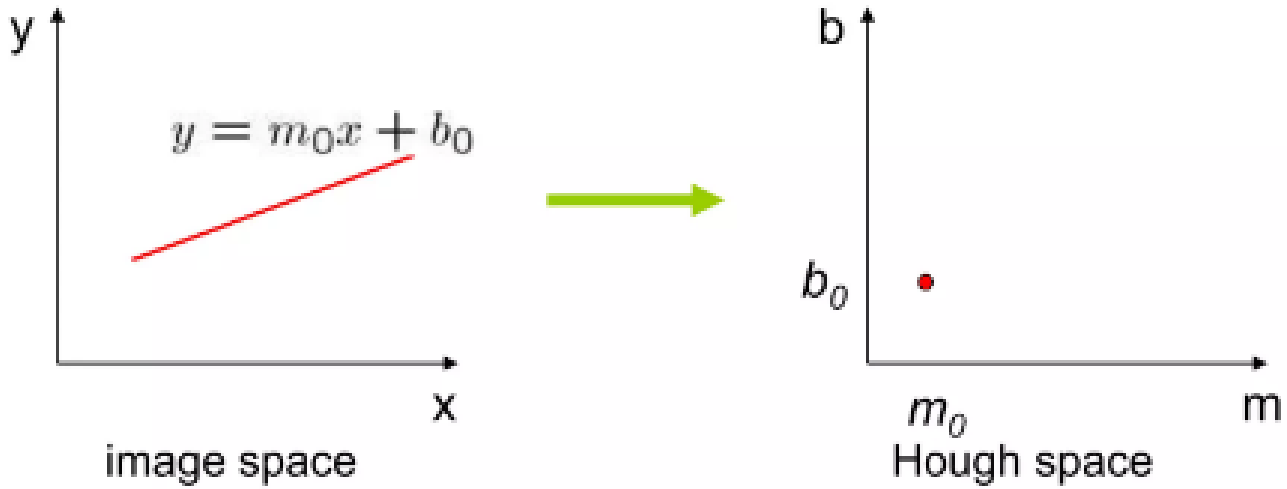
- The number of votes at *(m, c)* Corresponds to the number of points that lie on the line *(m, c)*.





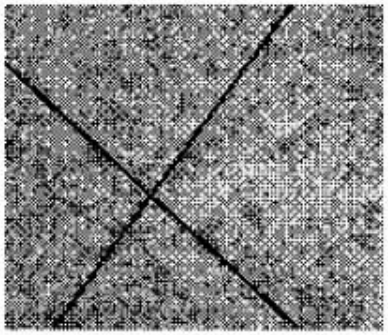
42

Finding lines in an image

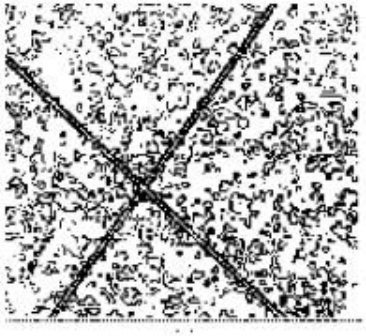


- Connection between image (x,y) and Hough (m,b) spaces
  - A line in the image corresponds to a point in Hough space
  - To go from image space to Hough space:
    - given a set of points (x,y), find all (m,b) such that y = mx + b

Hough Transform Line Detection Example

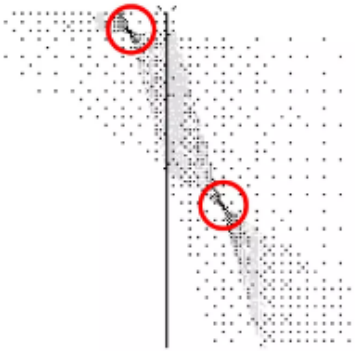


Original Image

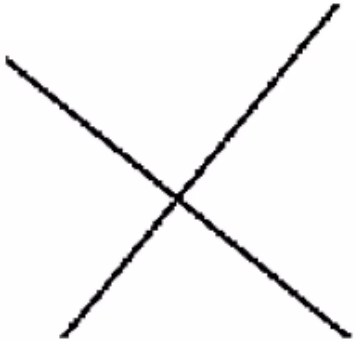


Edge Image

Notice: many non-belonging edges



Parameter Space

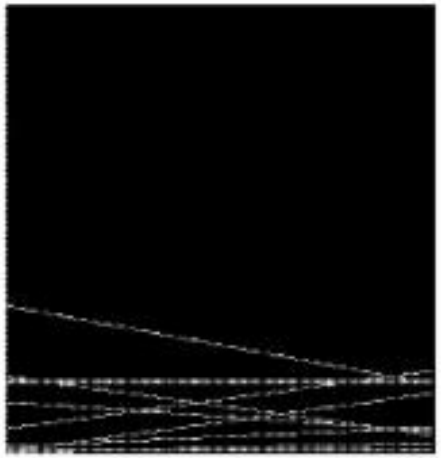


Detected Lines

Hough Transform Line Detection Example

Example:

Original Image



Detected Lines

Original Image with superimposed Lines

Better Parameterization
(Hough Space Sinusoid)

NOTE: $-\infty \le m \le \infty$

- Large Accumulator
- More memory and computations

Improvement: (Finite Accumulator Array Size)

Line equation: $R = x\cos\theta + y\sin\theta$

Here $0 \le \theta \le \pi$

$0 \le R \le R_{\max}$

Given points (x_i, y_i) find (R, θ)

R is the perpendicular distance from the line to the origin
θ is the angle this perpendicular makes with the x axis

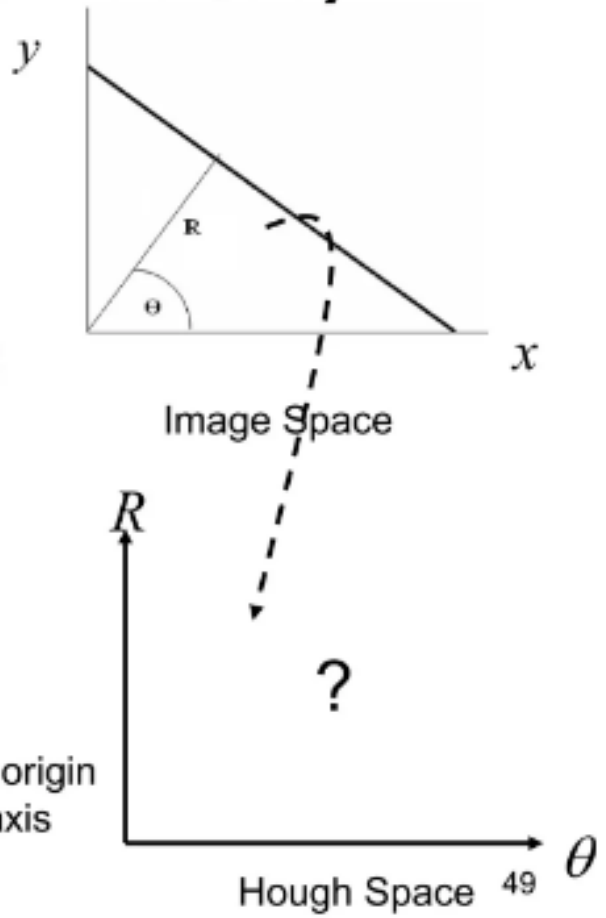


Image Features I

Hough transform algorithm

1. Initialize H[R, θ]=0
2. for each edge point I[x,y] in the image

   for θ = 0 to 180

   $$R = x\cos\theta + y\sin\theta$$

   H[R, θ] += 1
3. Find the value(s) of (R, θ) where H[R, θ] is maximum
4. The detected line in the image is given by

$$R = x\cos\theta + y\sin\theta$$

Hough Space Sinusoid Example

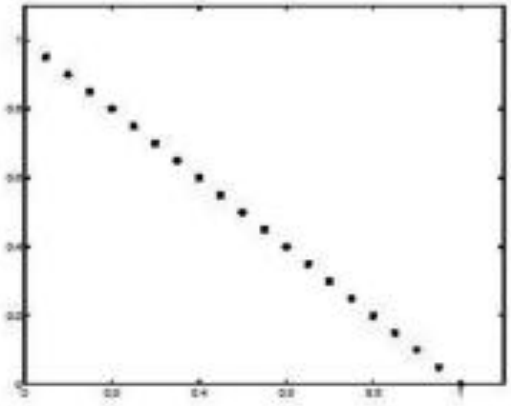


Image space

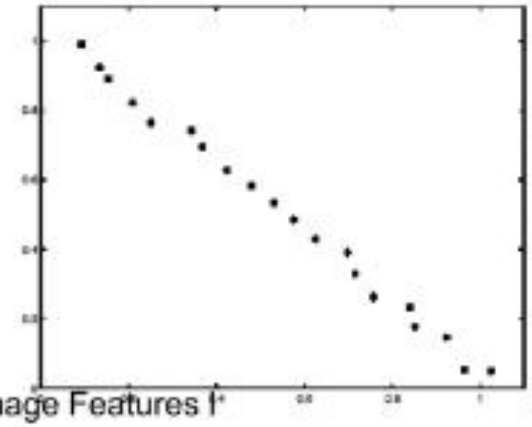
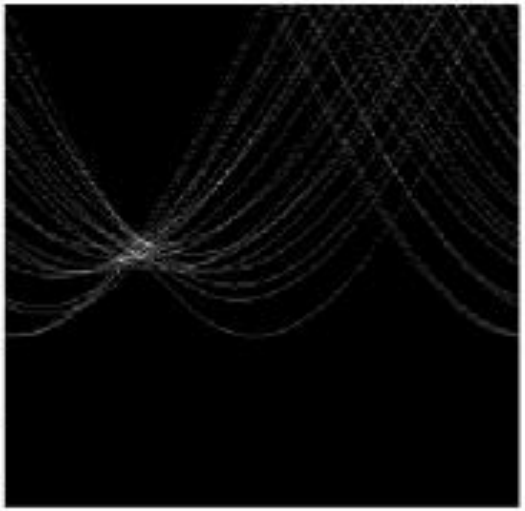


Image Features I

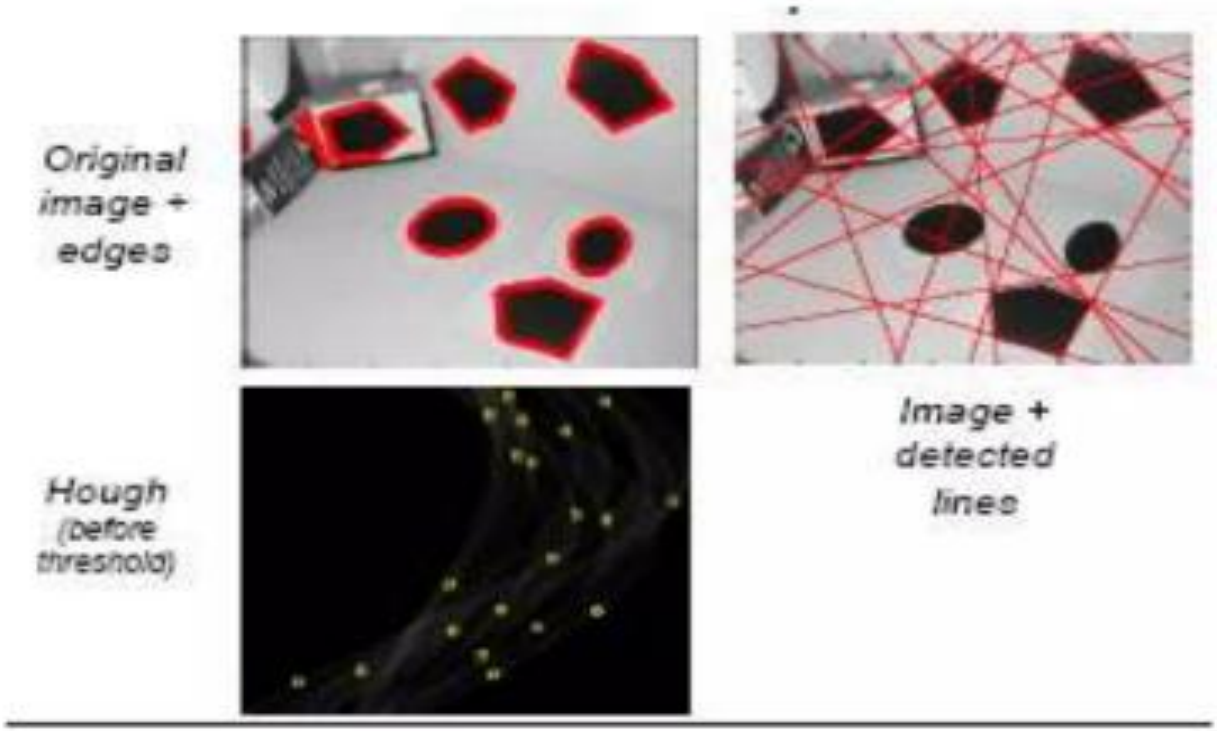
Votes



Horizontal axis is θ, vertical is *R*.

51

Hough Transform Line Detection Example



Note: Axis of R-θ parameter space is rotated 90° !

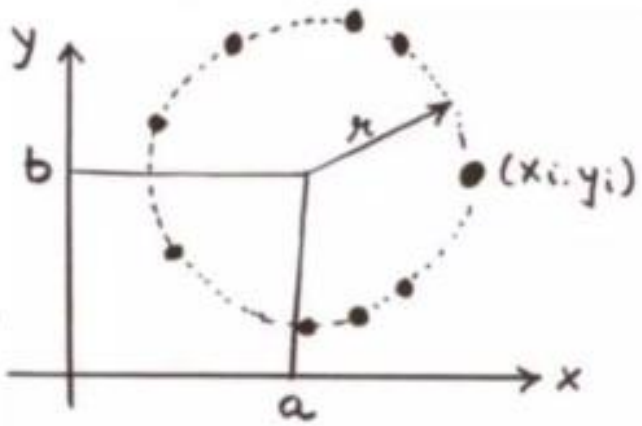
Finding Circles by Hough Transform

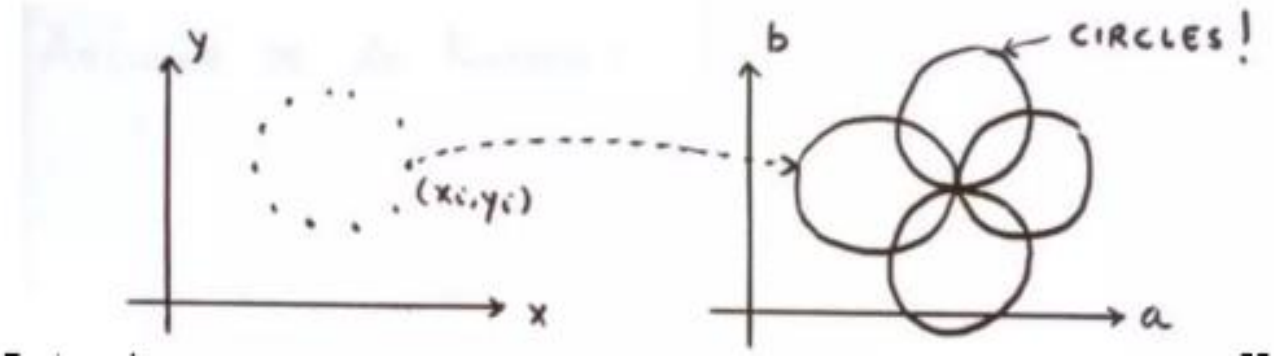
Equation of Circle:

$$(x_i - a)^2 + (y_i - b)^2 = r^2$$

If radius is known: (2D Hough Space)

Accumulator Array $A(a,b)$

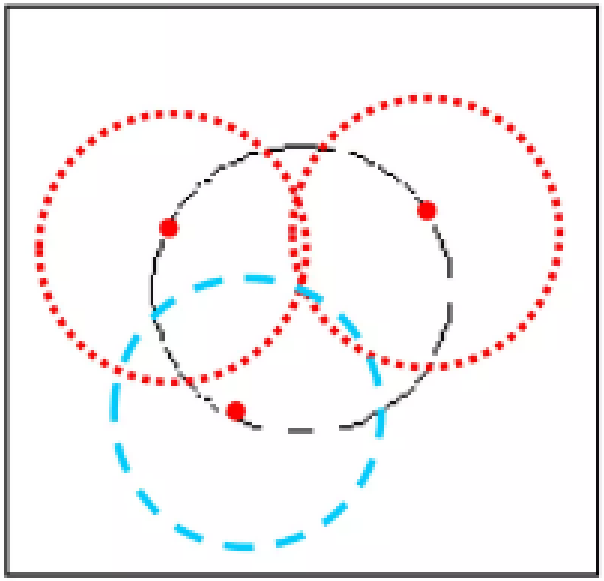


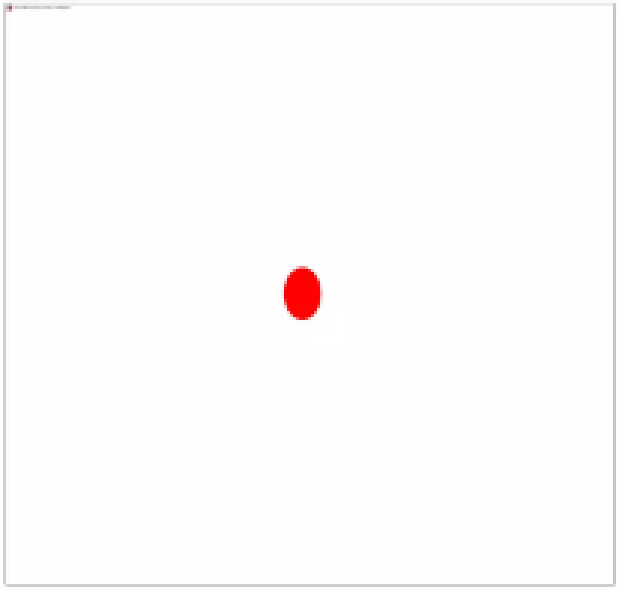


Algorithm: Hough Transform for Circle Detection

- Again the Hough transform is applied to each point whose edge magnitude exceeds a certain threshold.
- The processing results correspond to points of local maxima of accumulator cells in the parameter space ***a,b*** of the center point of the circle and the radius ***r***.
- If the length of the radius of the circle searched is known then we can deal with a 2-dimensional parameter space (for center point coordinates ***a,b***).

Demo:Hough Transform





Original Circle

point on circle

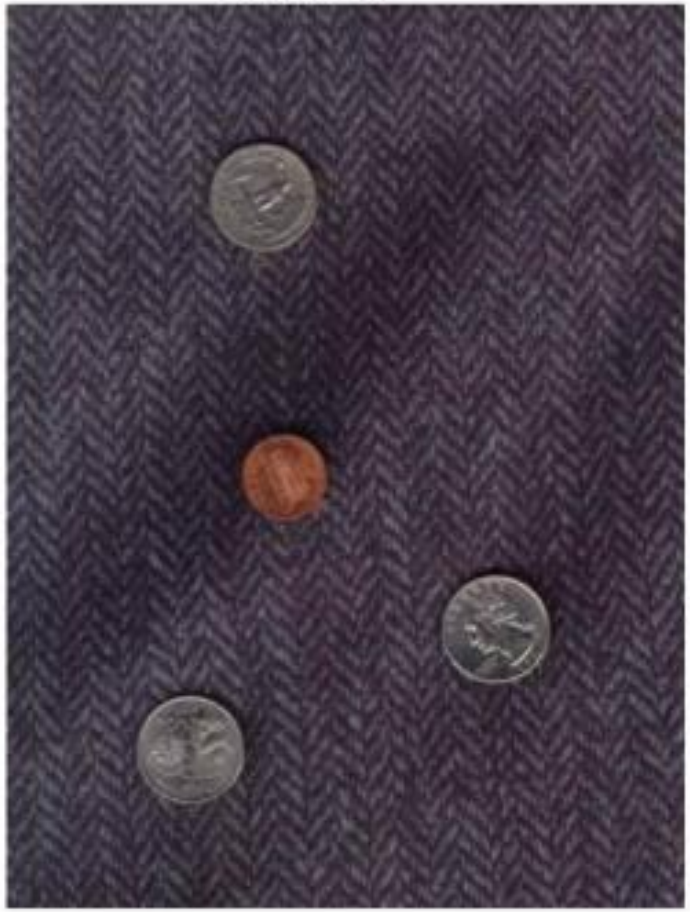
circles passing through point

loci of centers of circles passing through point

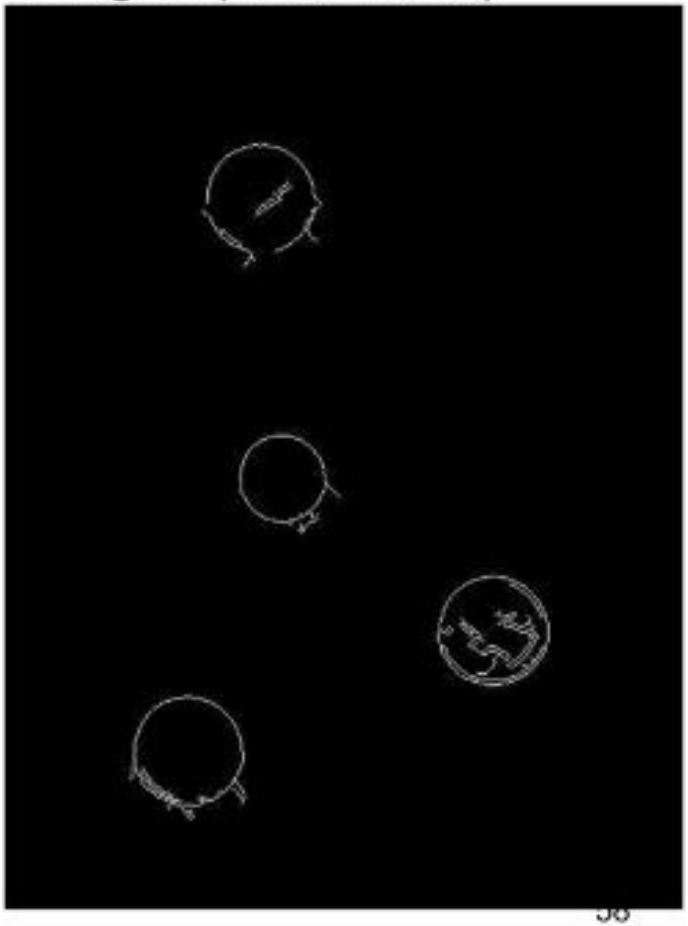
Intersection of circles specifies center of original circle

Finding Coins

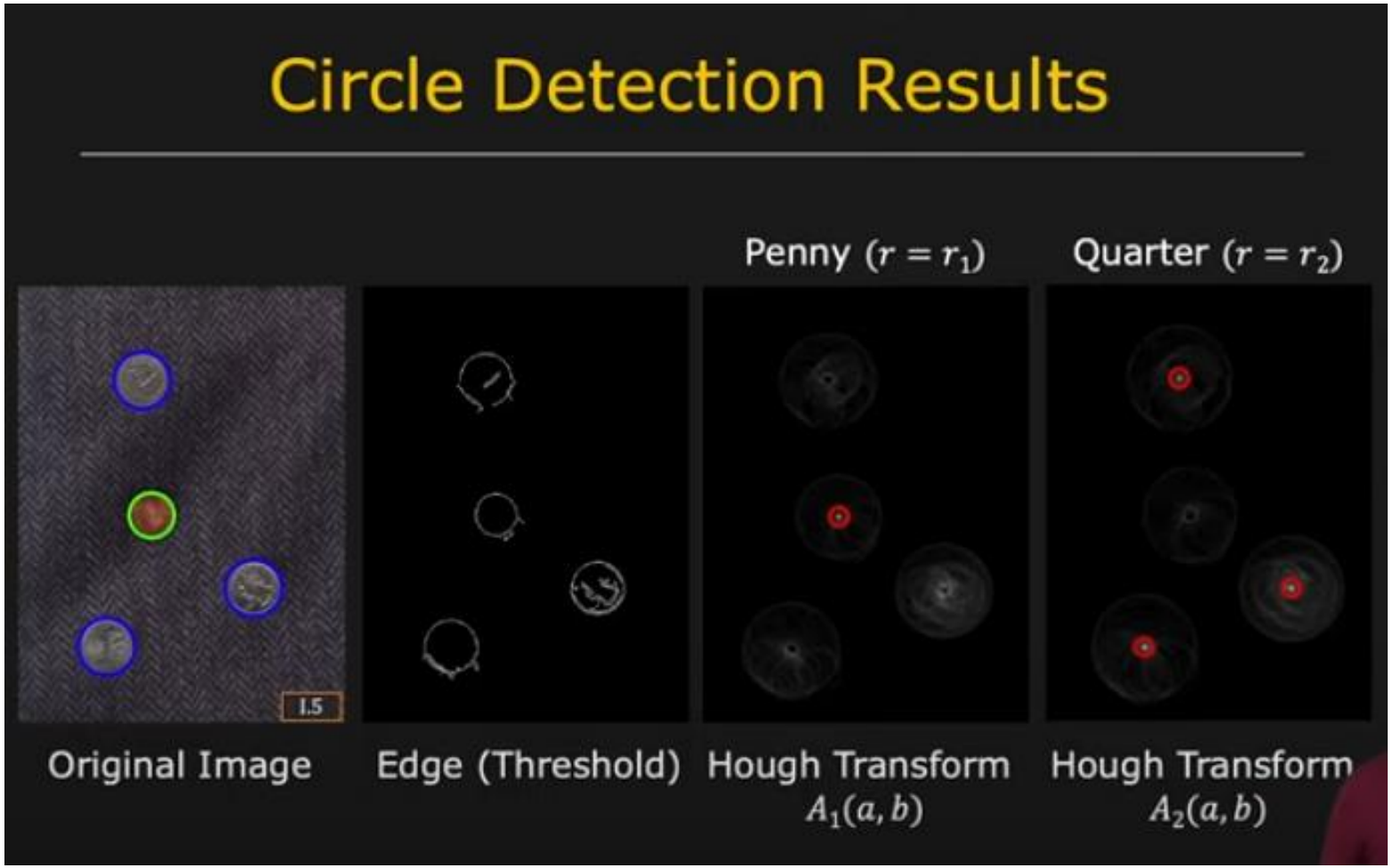
Original



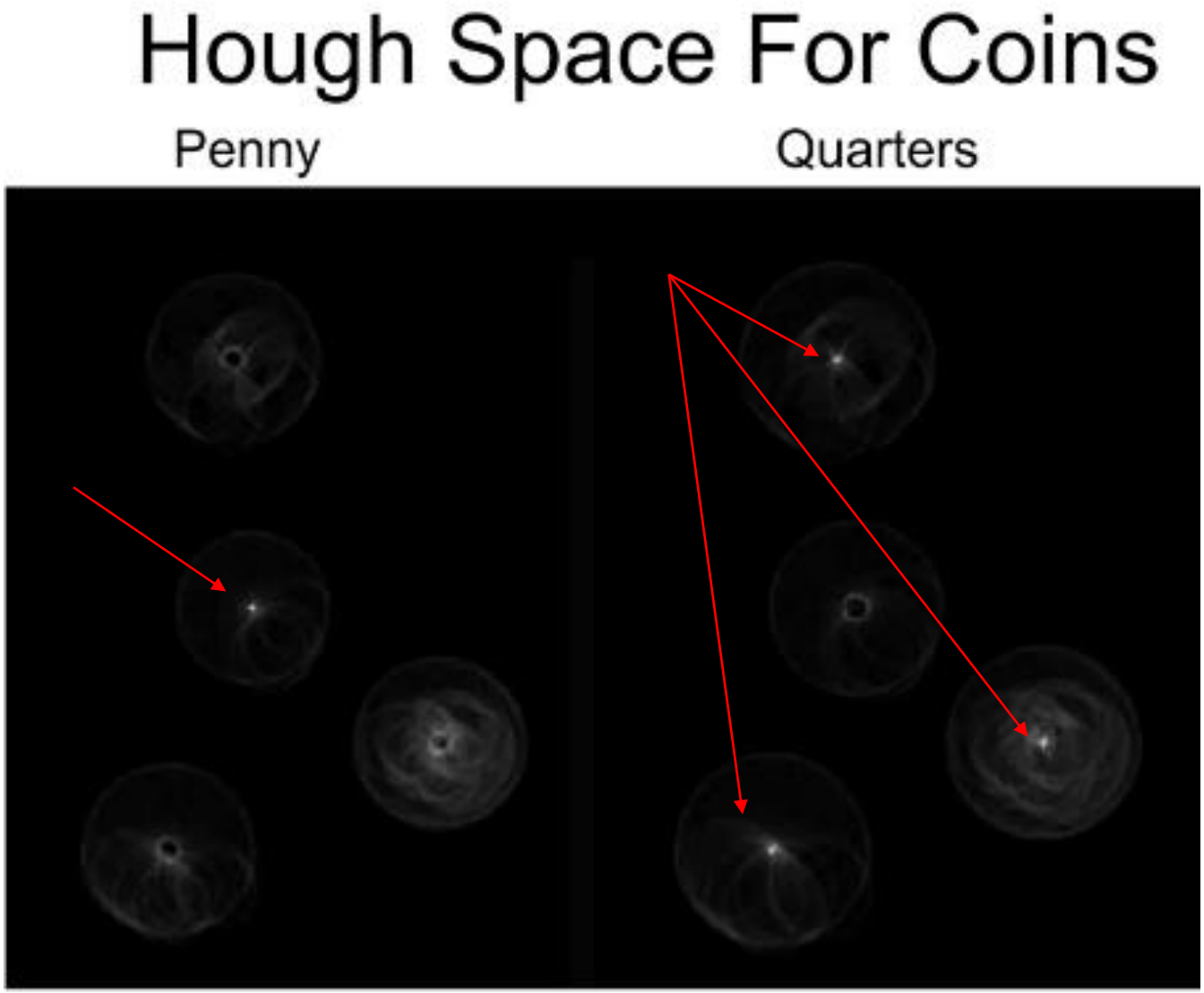
Edges (note noise)



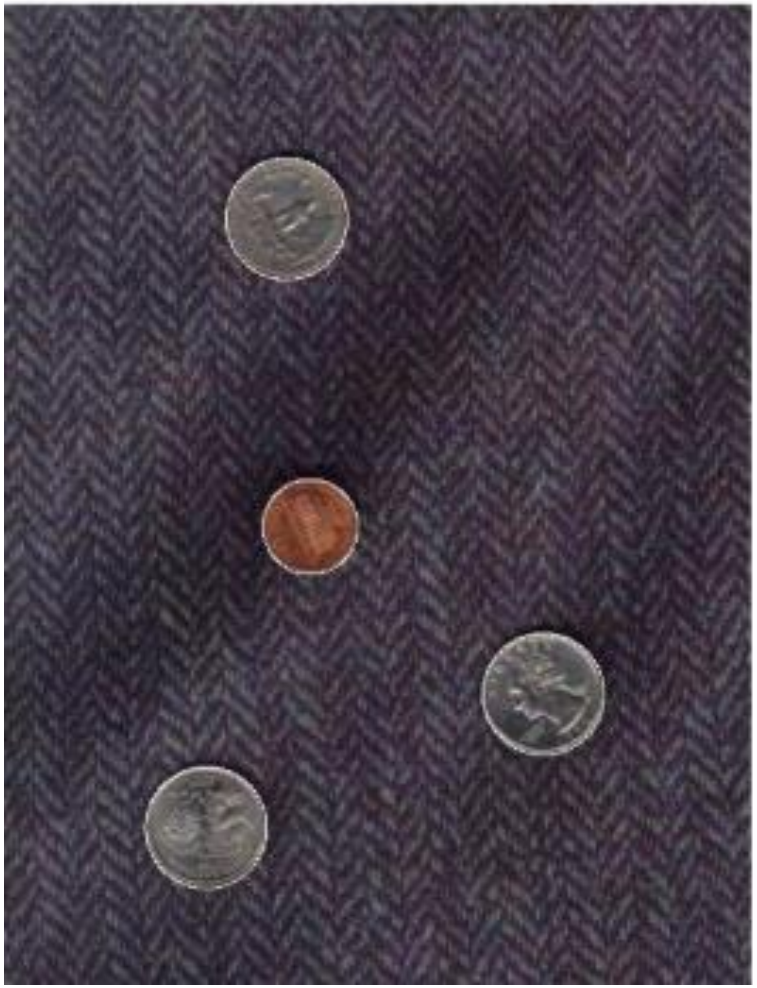
Primitives Detection



Primitives Detection



Detected Coins



Note that because the quarters and penny are different sizes, a different Hough transform (with separate accumulators) was used for each circle size.

For Hough Applet see
http://www.markschulze.net/java/hough/

Hough Transform: Comments

- Works on Disconnected Edges
- Relatively insensitive to occlusion
- Effective for simple shapes (lines, circles, etc)
- Handling inaccurate edge locations

Hough Transform for Circle Detection

- If looking for circles, instead of lines the analytic expression is:

$$(x-a)^2+(y-b)^2=r^2$$

- As there are 3 parameters then the accumulator data structure must be three dimensional, i.e. the accumulator cell is ***A(a,b,r).***

Primitives Detection

$$(x-a)^2+(y-b)^2=r^2$$

The Hough Transform for circle detection uses a 3D accumulator to collect votes for each possible (a,b,r) combination. Peaks in this 3D space indicate the presence of circles with specific centers and radii.

Primitives Detection

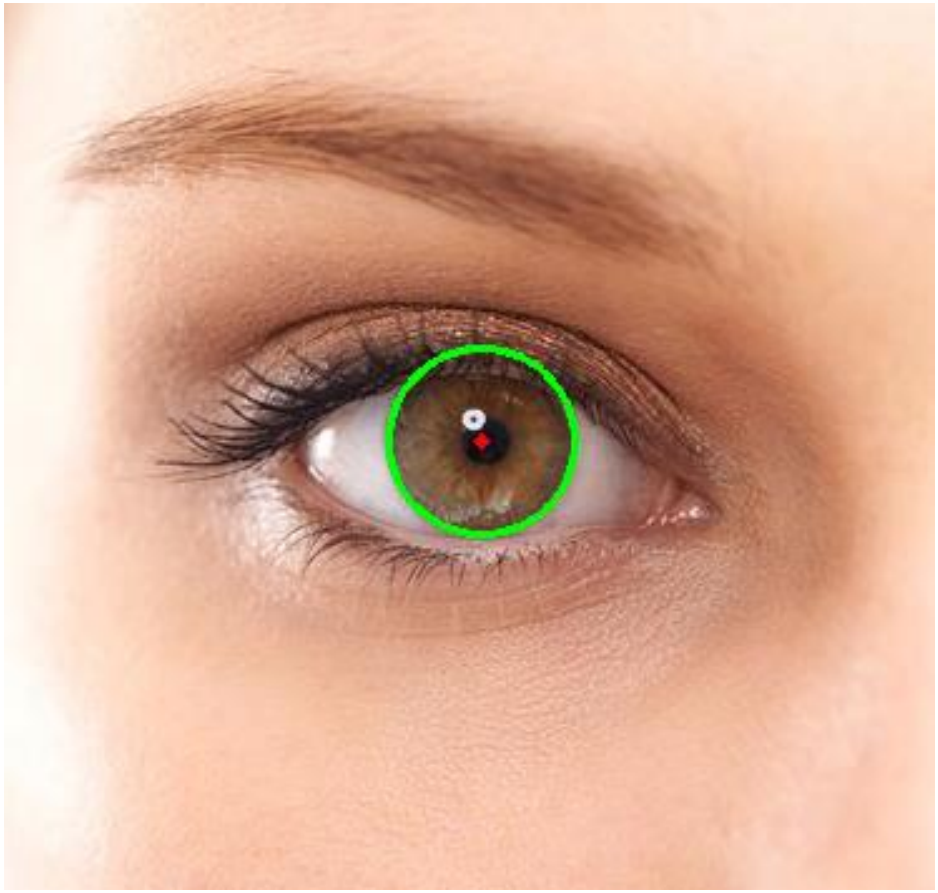
Hough transform Advantages and Limitations

•**Advantages**: Effective in detecting shapes even if they are partially obscured or noisy.

•**Limitations**: Computationally intensive, especially for circles due to the 3D accumulator array.

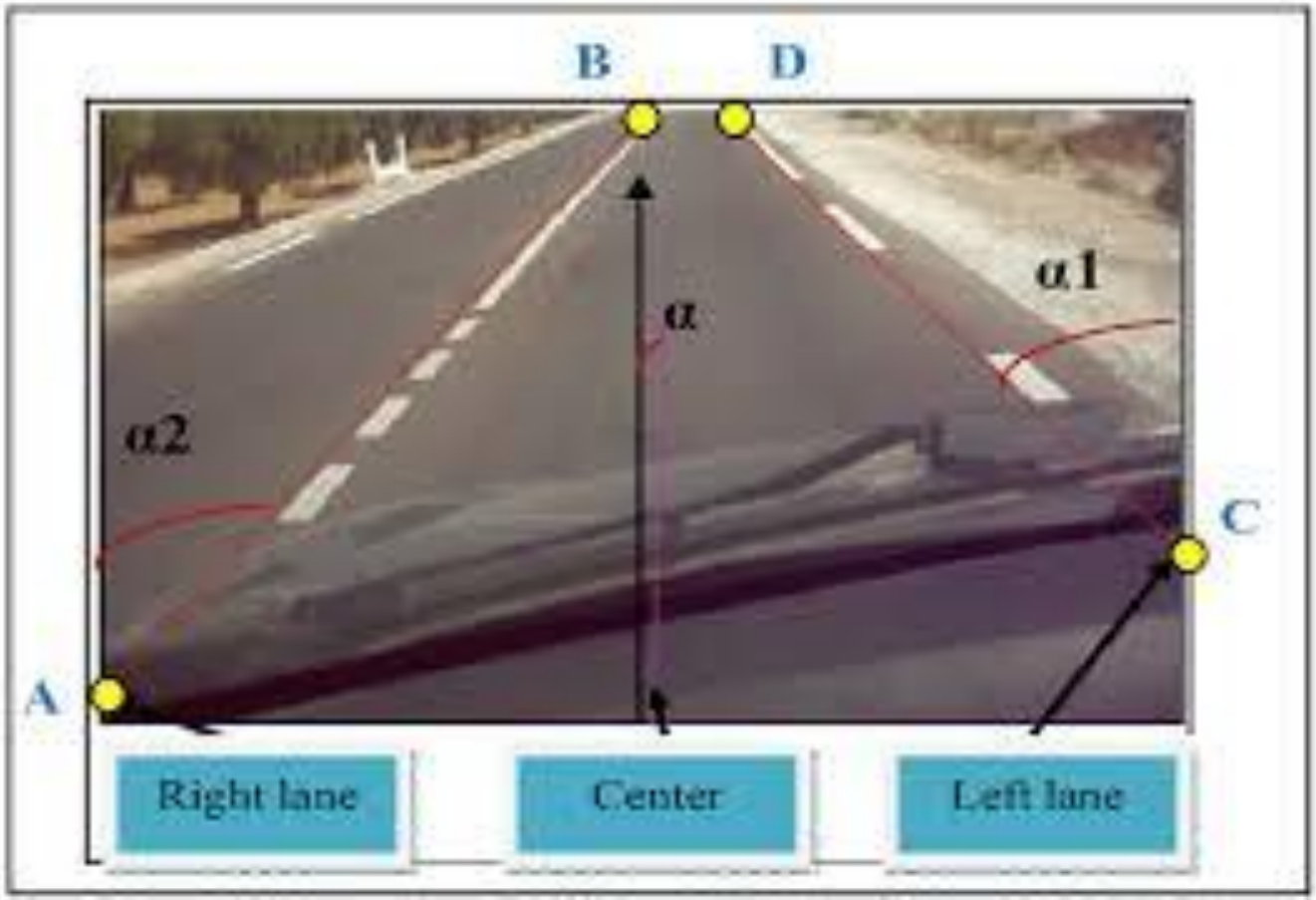
Primitives Detection

Example of detection by hough transform (circle detection)



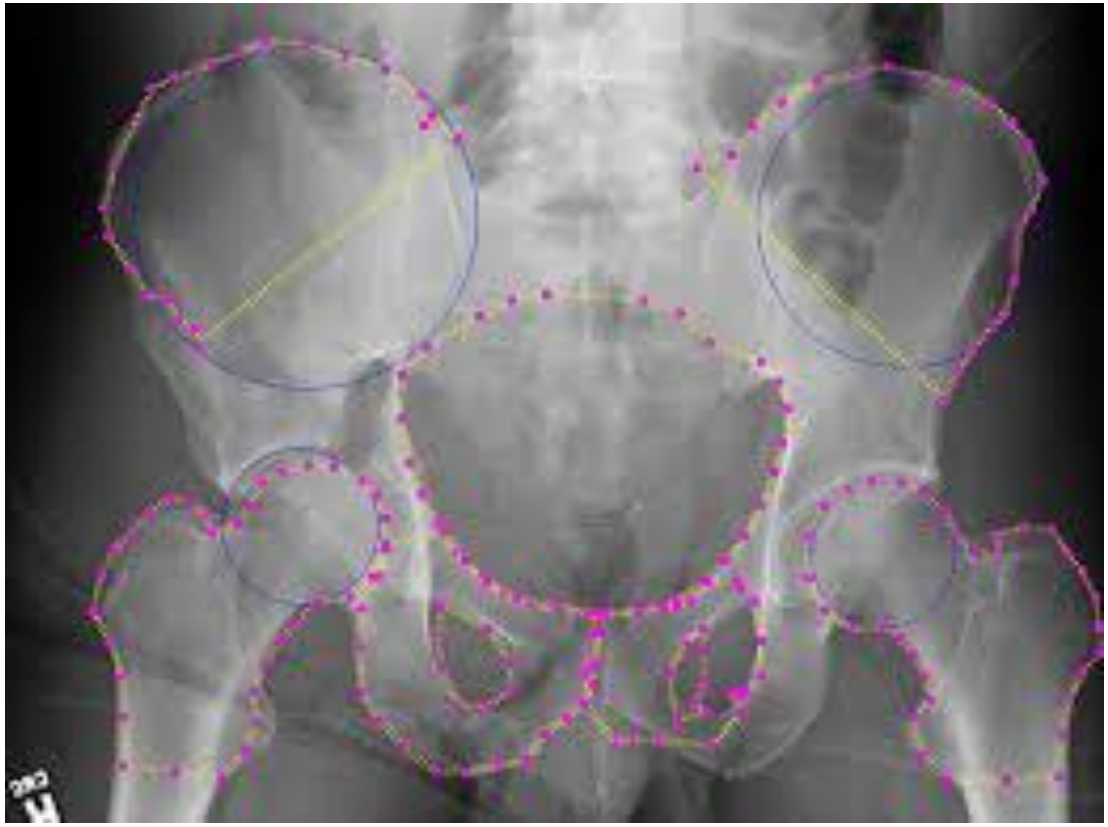
Primitives Detection

Example of detection by hough transform (road lane detection)



Primitives Detection

Example of detection by hough transform (line detection)



Primitives Detection

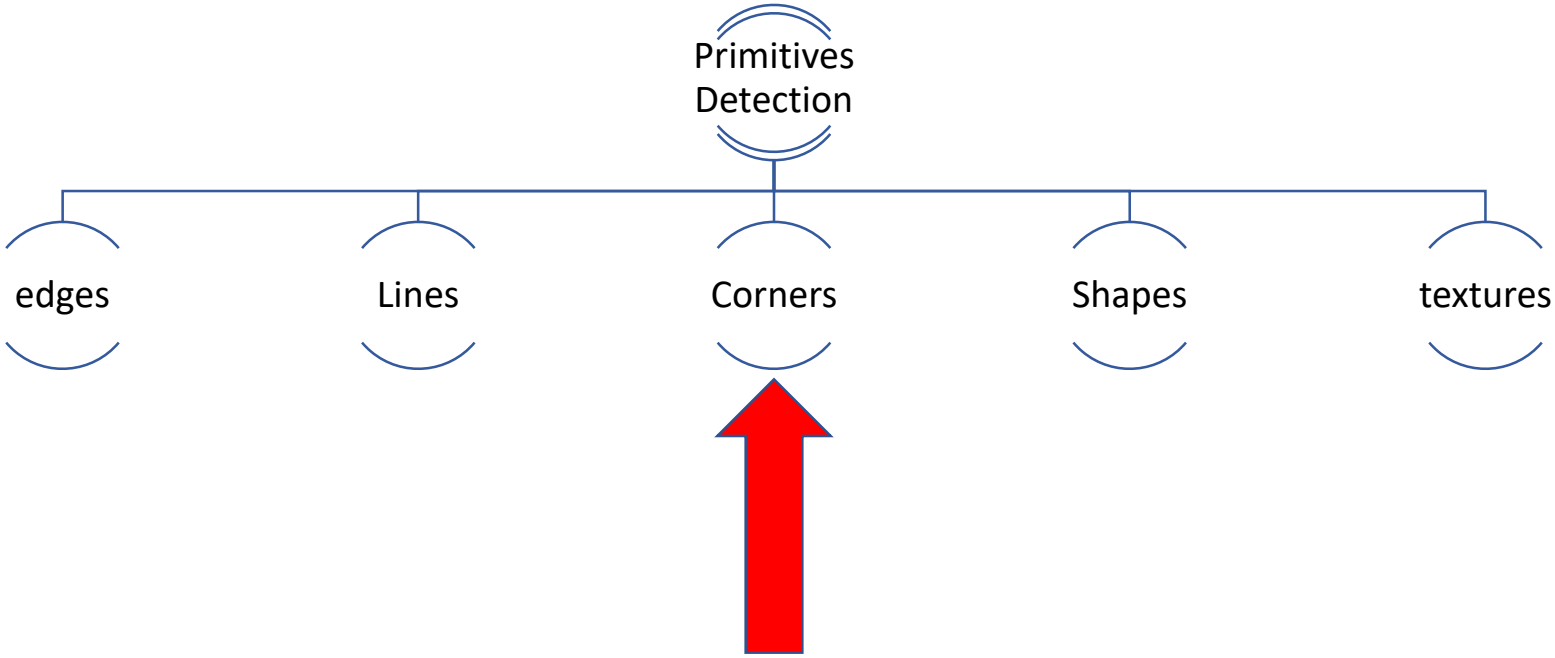
Applications

•**Line Detection**: Useful in road lane detection, document analysis, and various image processing tasks requiring straight edge recognition.

•**Circle Detection**: Applied in biomedical imaging (like cell counting), object recognition, and industrial vision tasks requiring circular feature detection.

Primitives Detection

3/ corners



Primitives Detection

Key Primitives Detected in Computer Vision:

3/ Corners:

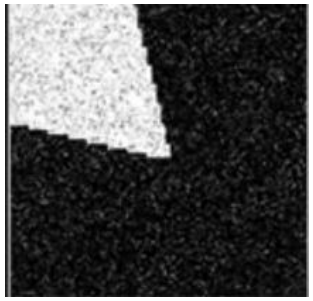
•**Description**: Corners are points where two or more edges meet. They are highly distinct features that can be used for recognizing specific points in an image.

•**Detection Methods**: Harris corner detection, Shi-Tomasi corner detector.

•**Application**: Used in feature matching, tracking, and object recognition tasks

“Flat” region

“Edge” region



“Corner” region

A corner is : a Rapid change of image intensity in two directions within a small region

Primitives Detection

Key Primitives Detected in Computer Vision: corners

In each case we normalize the output:

White: strong positive value

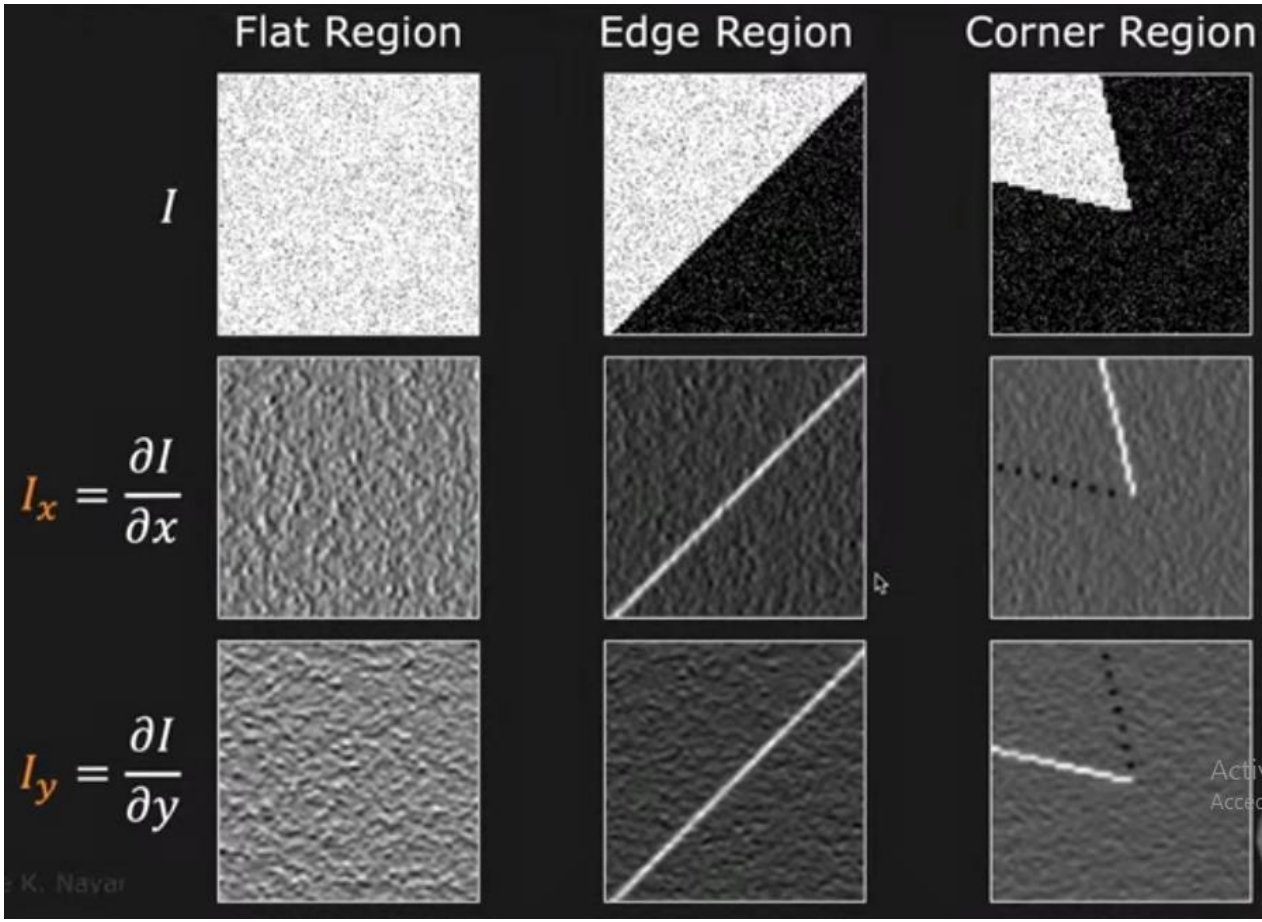
Black: strong negative value

Gray: low value (or 0)

Derivative of the corner region:

- Low values every where except a strong positive value (white edge) and a strong negative value (black edge) for Ix
- Low values every where except a strong negative value (black edge) and a strong positive value (white edge) for Iy

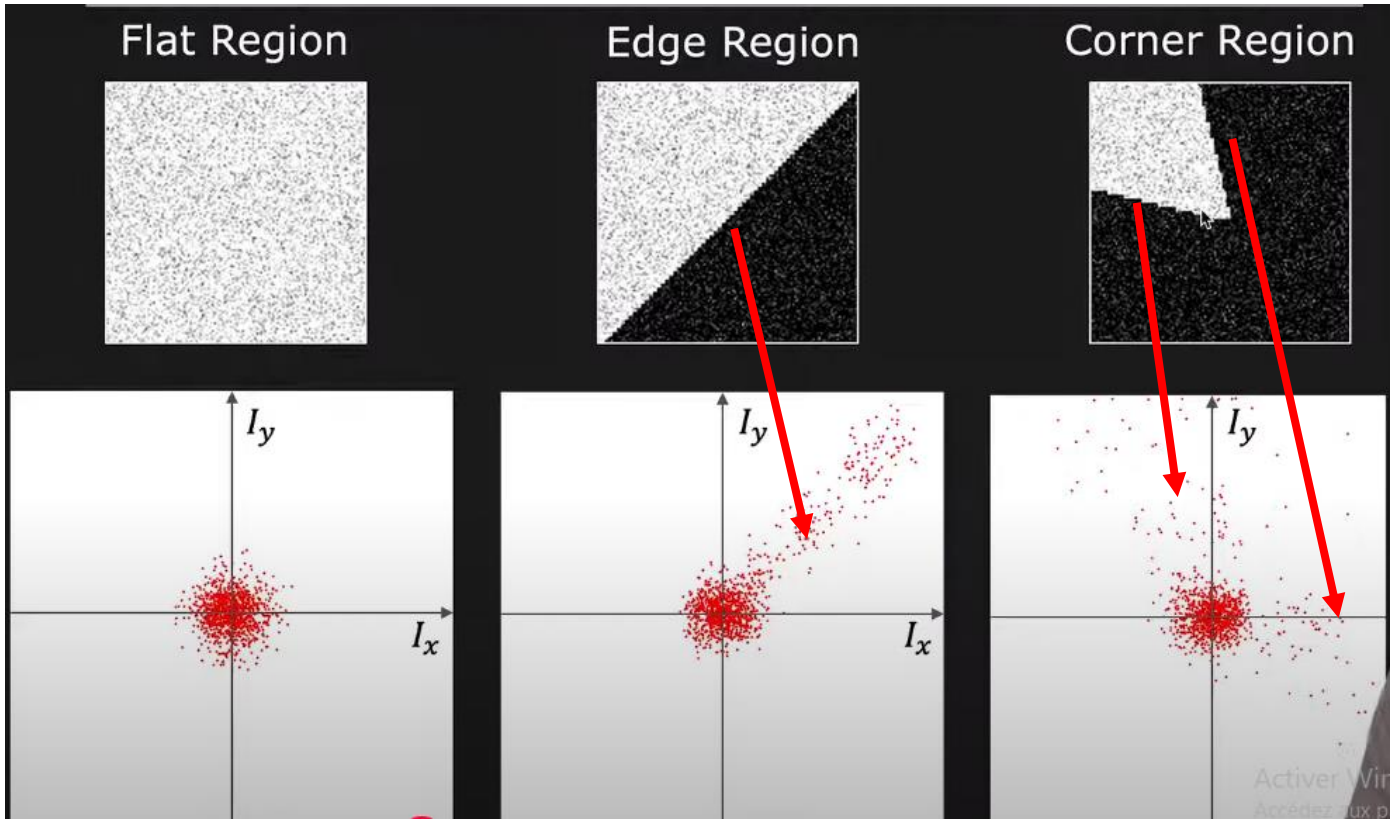
Image Gradients



Primitives Detection

Key Primitives Detected in Computer Vision: corners

Image Gradients space



Very compact cluster close to the origin

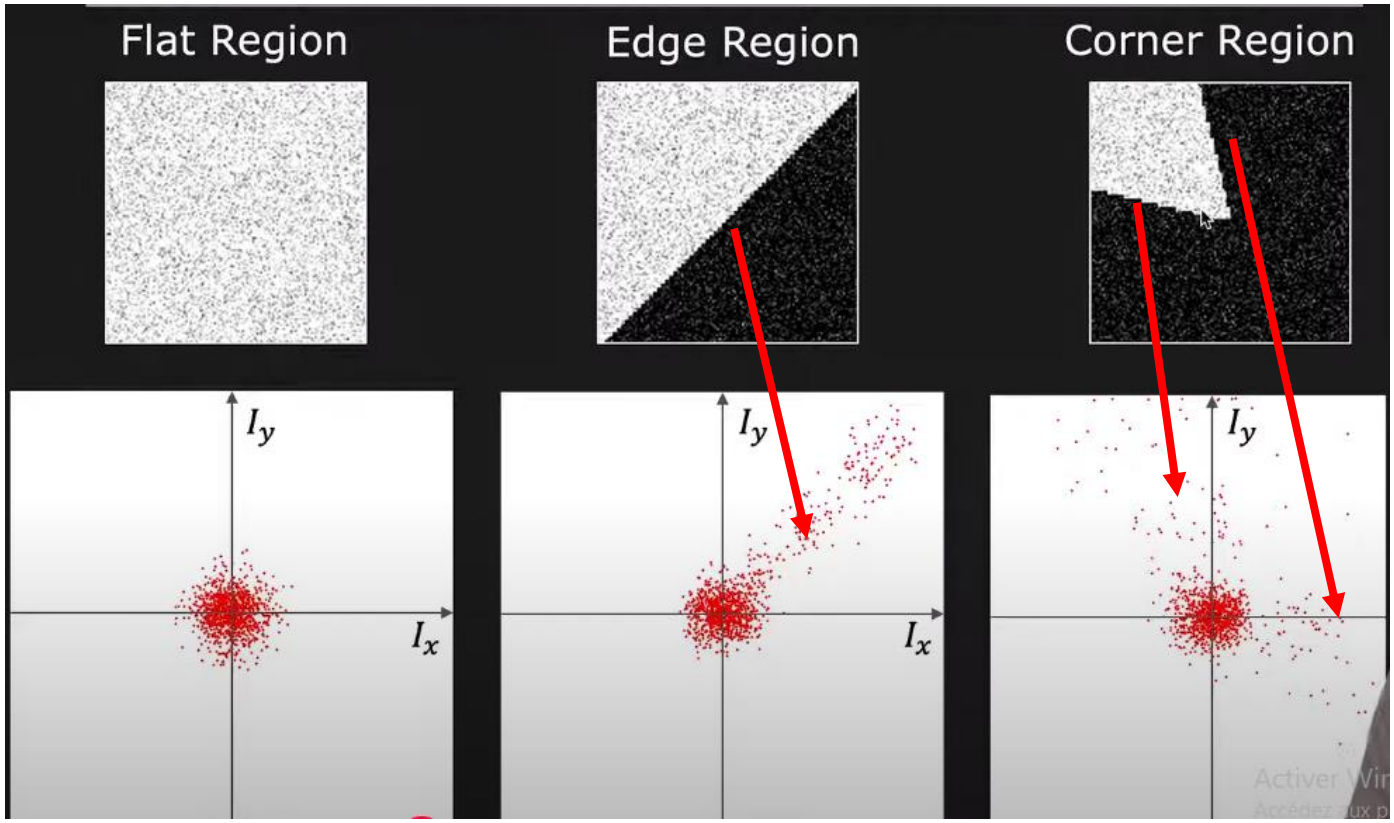
Very compact cluster close to the origin + very long linear cluster

Very compact cluster close to the origin + two long linear cluster

Primitives Detection

Key Primitives Detected in Computer Vision: corners

Image Gradients space

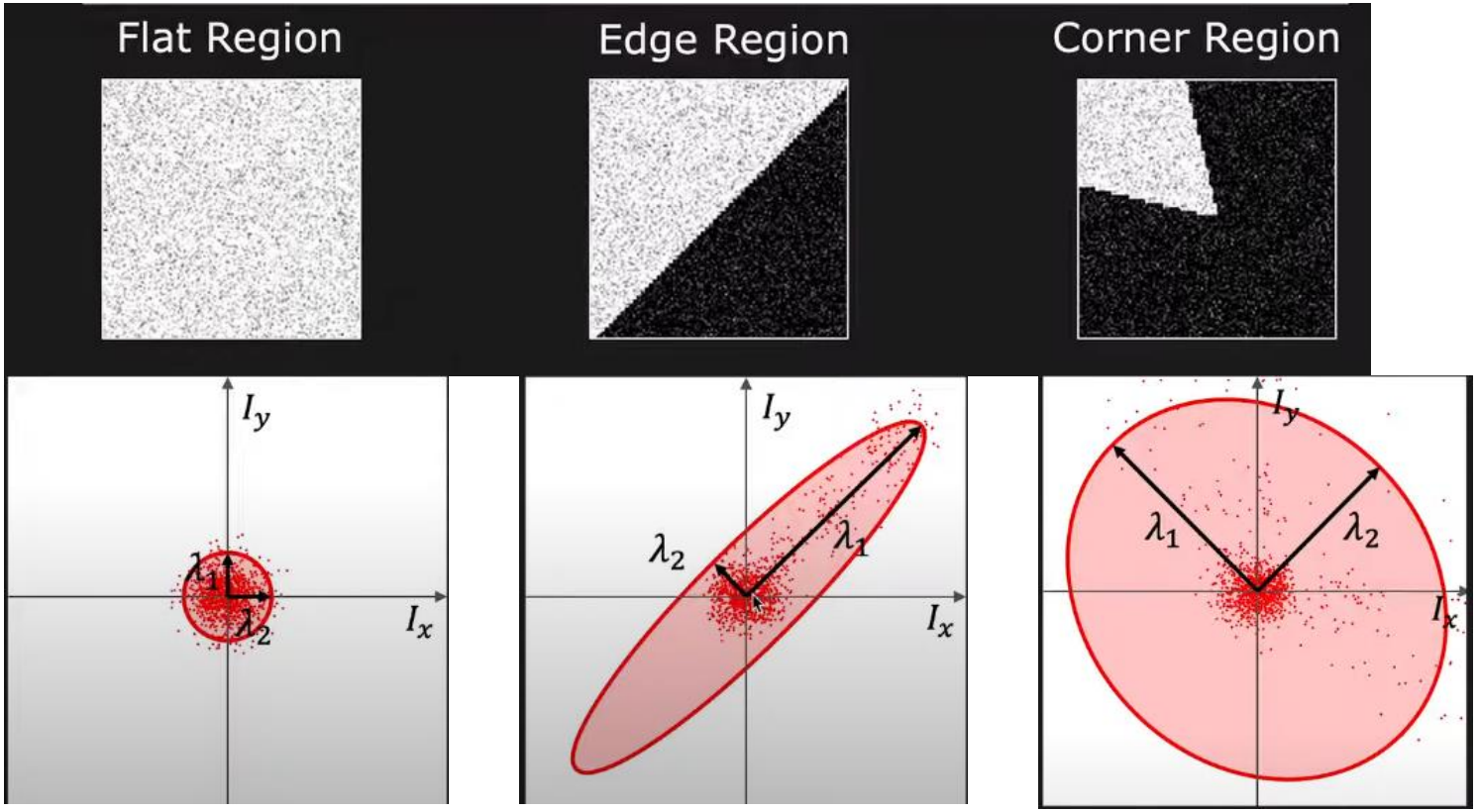


The key idea is to quantify this structure ie describe it with a small number of parameters→use those parameters to categorize or classify the region as being flat, edge or corner

Primitives Detection

Key Primitives Detected in Computer Vision: corners

Fitting Elliptical Disk to distribution

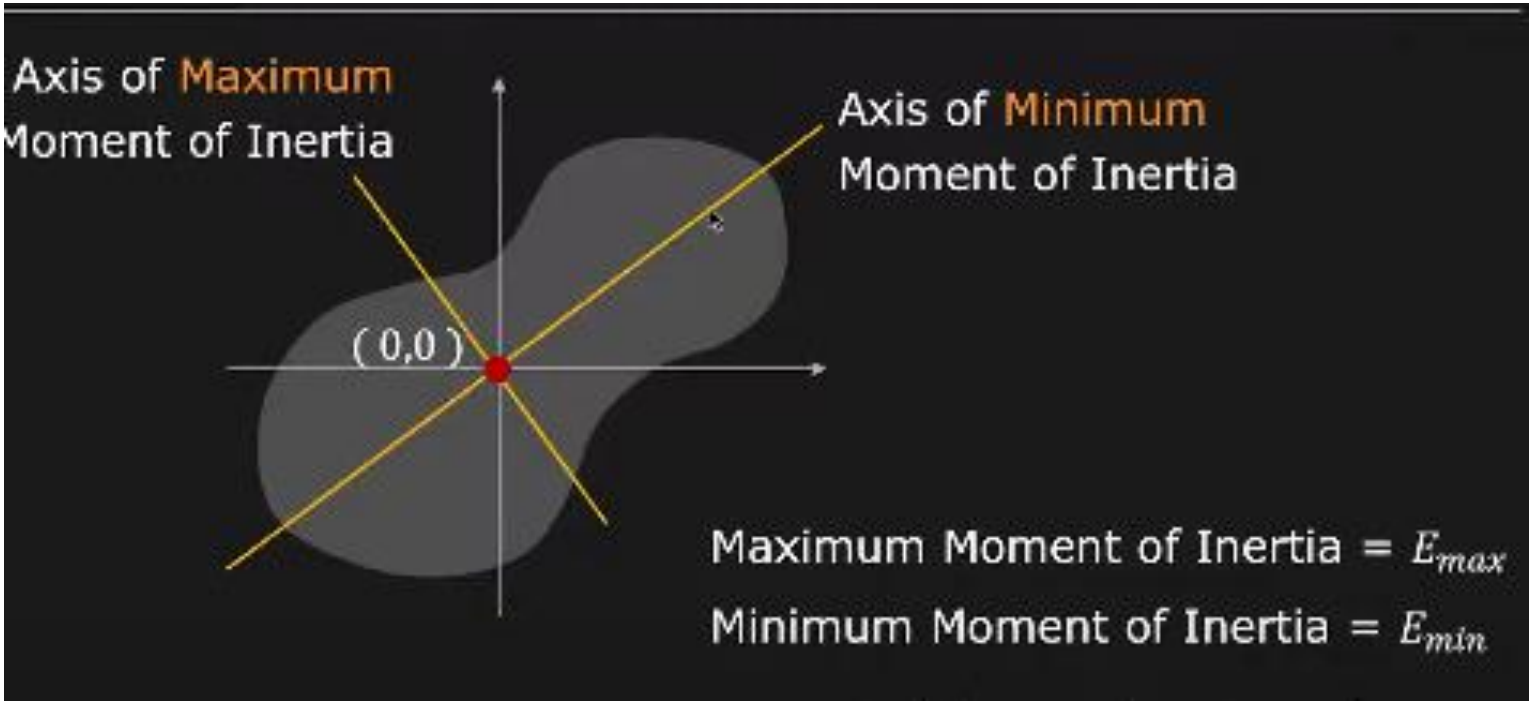


The way we do that is fitting elliptical disk to distribution, and then calculating parameters

Primitives Detection

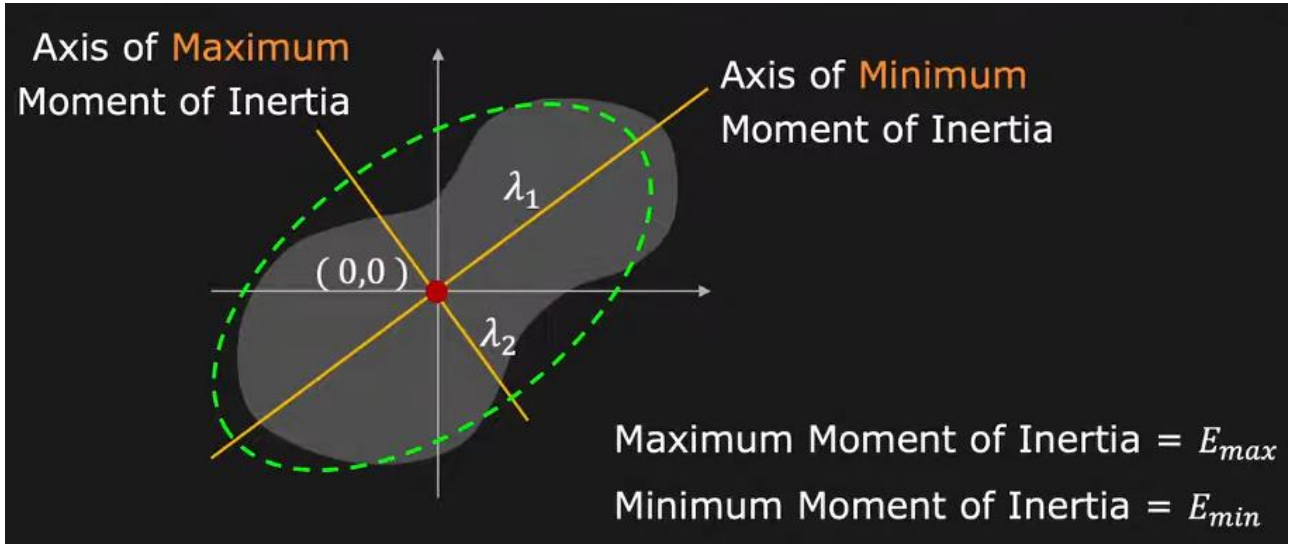
Key Primitives Detected in Computer Vision: corners

Fitting Elliptical Disk to distribution



Key Primitives Detected in Computer Vision: corners

Fitting Elliptical Disk to distribution



Length of Semi-Major Axis = $\lambda_1 = E_{max}$

Length of Semi-Minor Axis = $\lambda_2 = E_{min}$

Key Primitives Detected in Computer Vision: corners

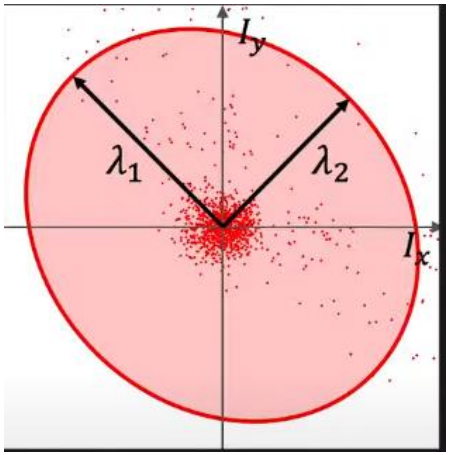
Fitting Elliptical Disk to distribution

Second Moments for a Region:

$$a = \sum_{i \in W} \left(I_{x_i}{}^2\right) \qquad b = 2\sum_{i \in W} \left(I_{x_i} I_{y_i}\right)$$

$$c = \sum_{i \in W} \left(I_{y_i}{}^2\right)$$

W: Window centered at pixel

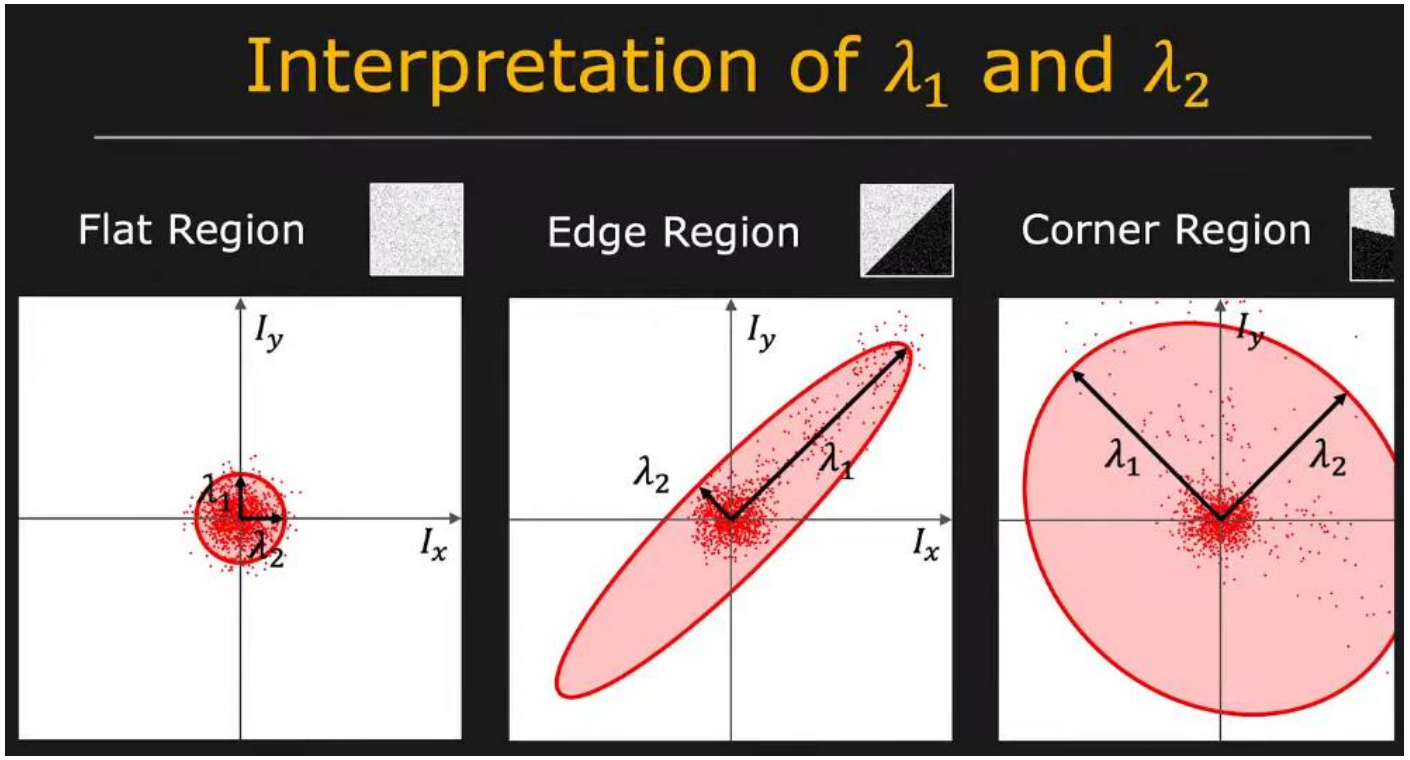


Ellipse Axes Lengths:

$$\lambda_1 = E_{max} = \frac{1}{2}\left[a + c + \sqrt{b^2 + (a-c)^2}\right]$$

$$\lambda_2 = E_{min} = \frac{1}{2}\left[a + c - \sqrt{b^2 + (a-c)^2}\right]$$

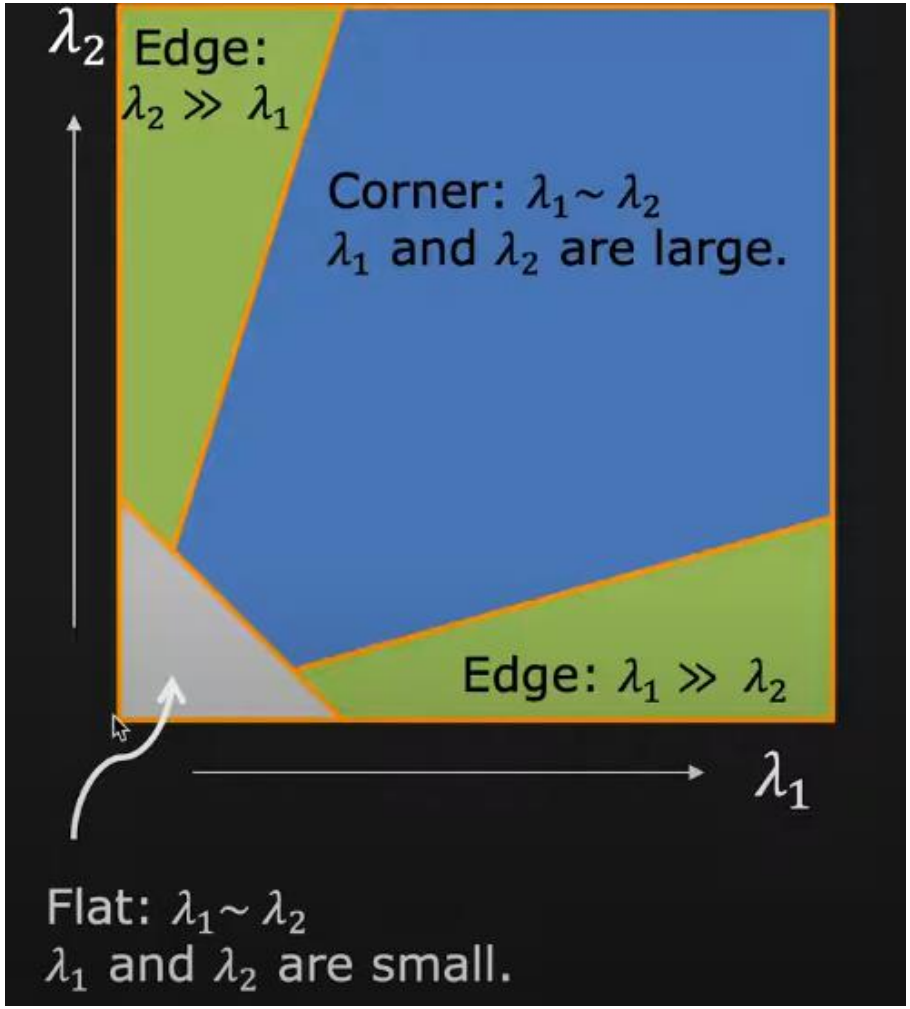
Key Primitives Detected in Computer Vision: corners



Primitives Detection

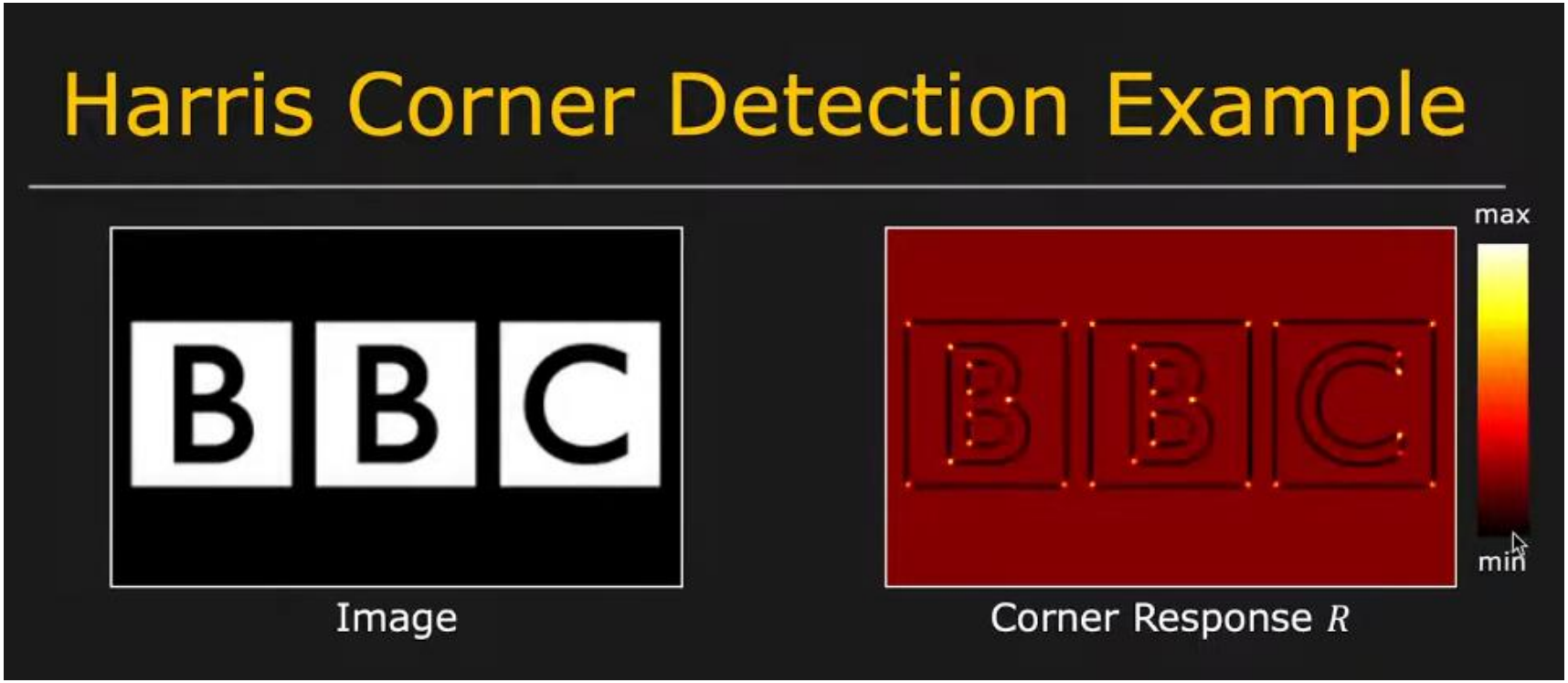
Key Primitives Detected in Computer Vision: corners

Fitting Elliptical Disk to distribution



Primitives Detection

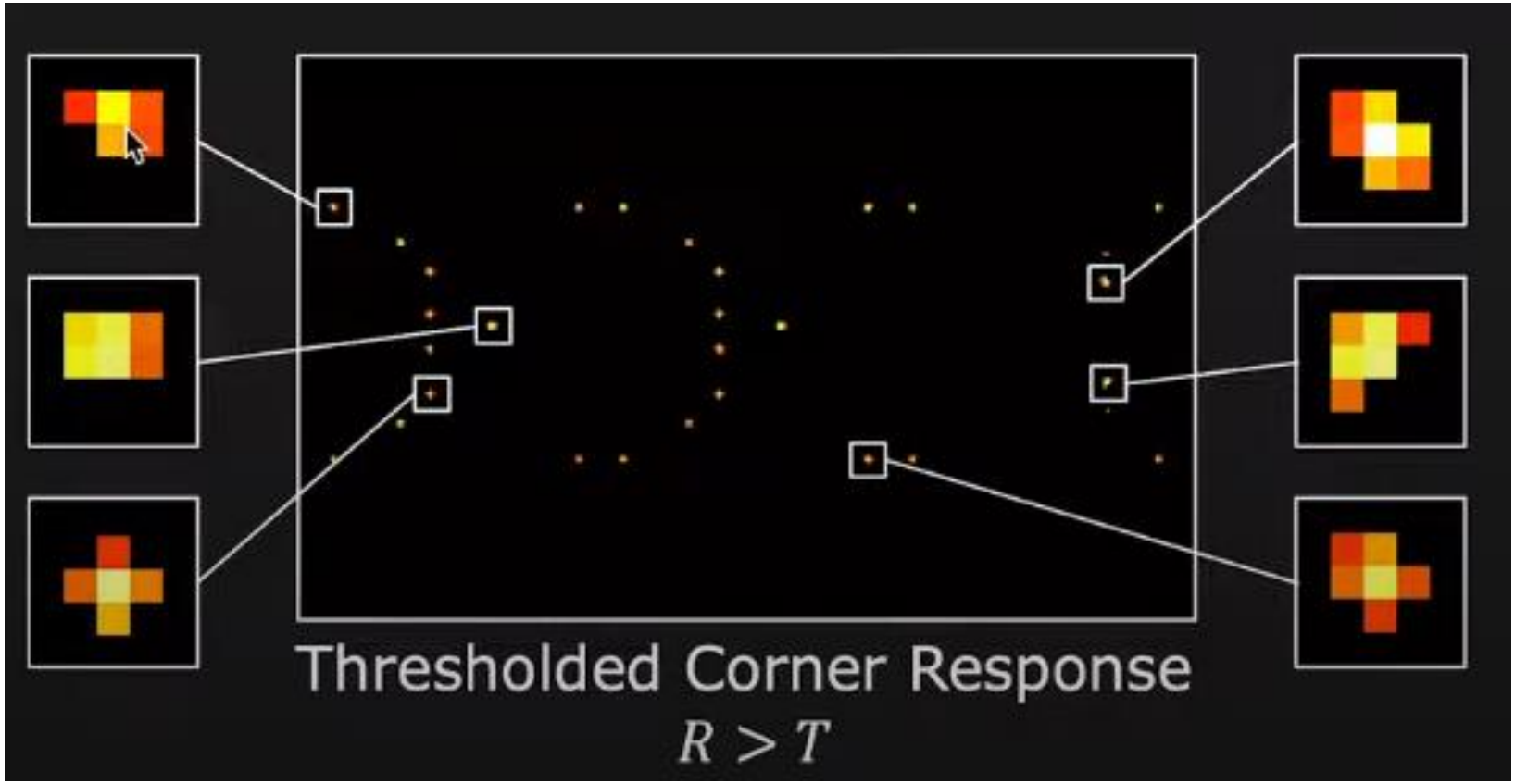
Key Primitives Detected in Computer Vision: corners



Primitives Detection

Key Primitives Detected in Computer Vision: corners

problem



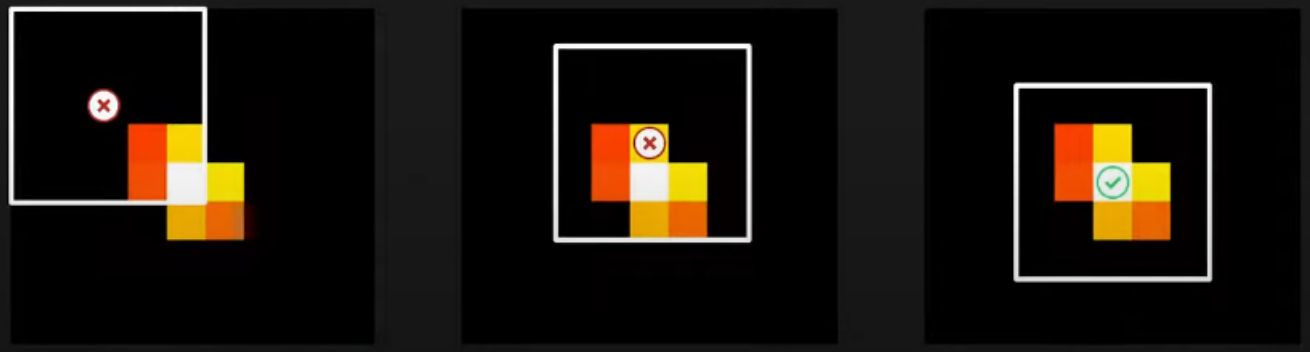
Primitives Detection

Key Primitives Detected in Computer Vision: corners

Non-Maximal Suppression

1. Slide a window of size k over the image.

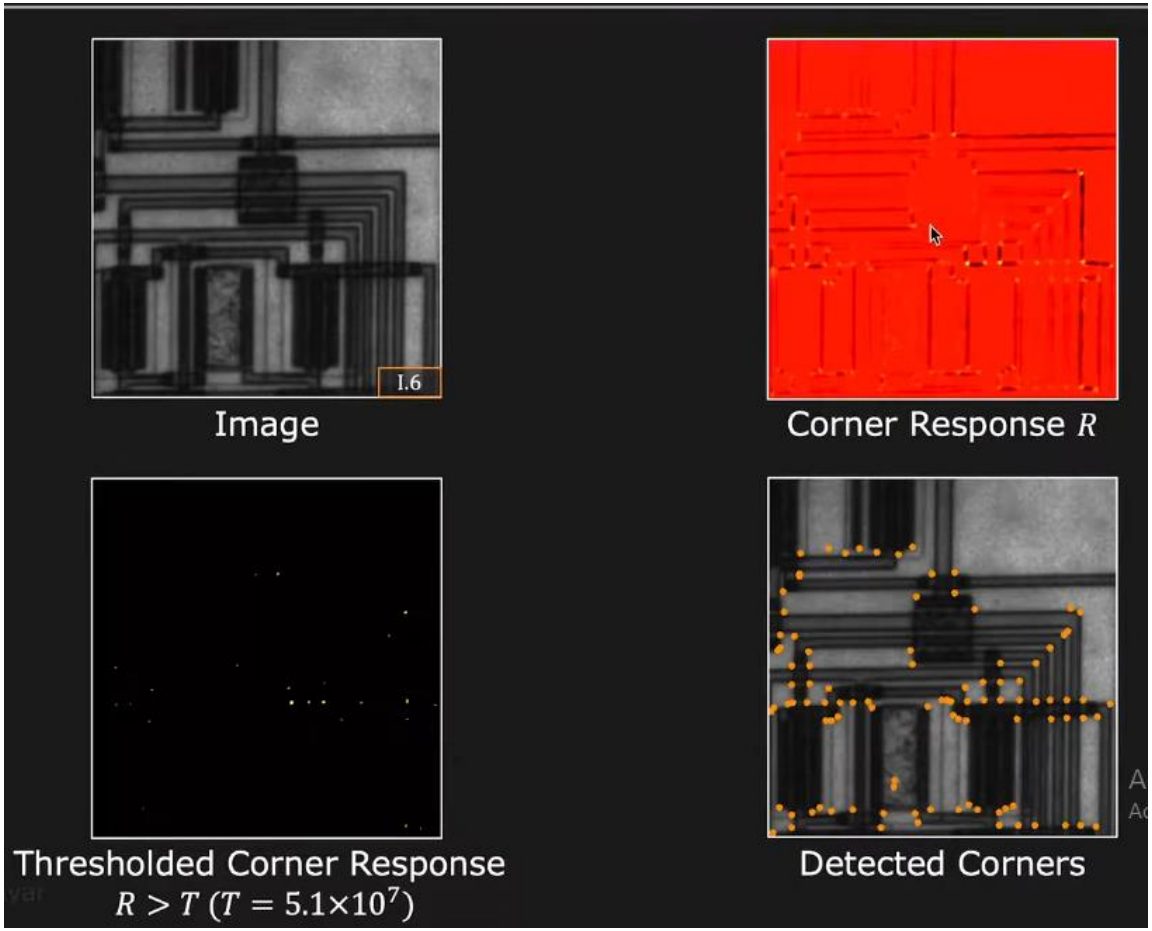
2. At each position, if the pixel at the center is the maximum value within the window, label it as positive (retain it). Else label it as negative (suppress it).



Primitives Detection

Key Primitives Detected in Computer Vision: corners

Harris Corner Detection Example

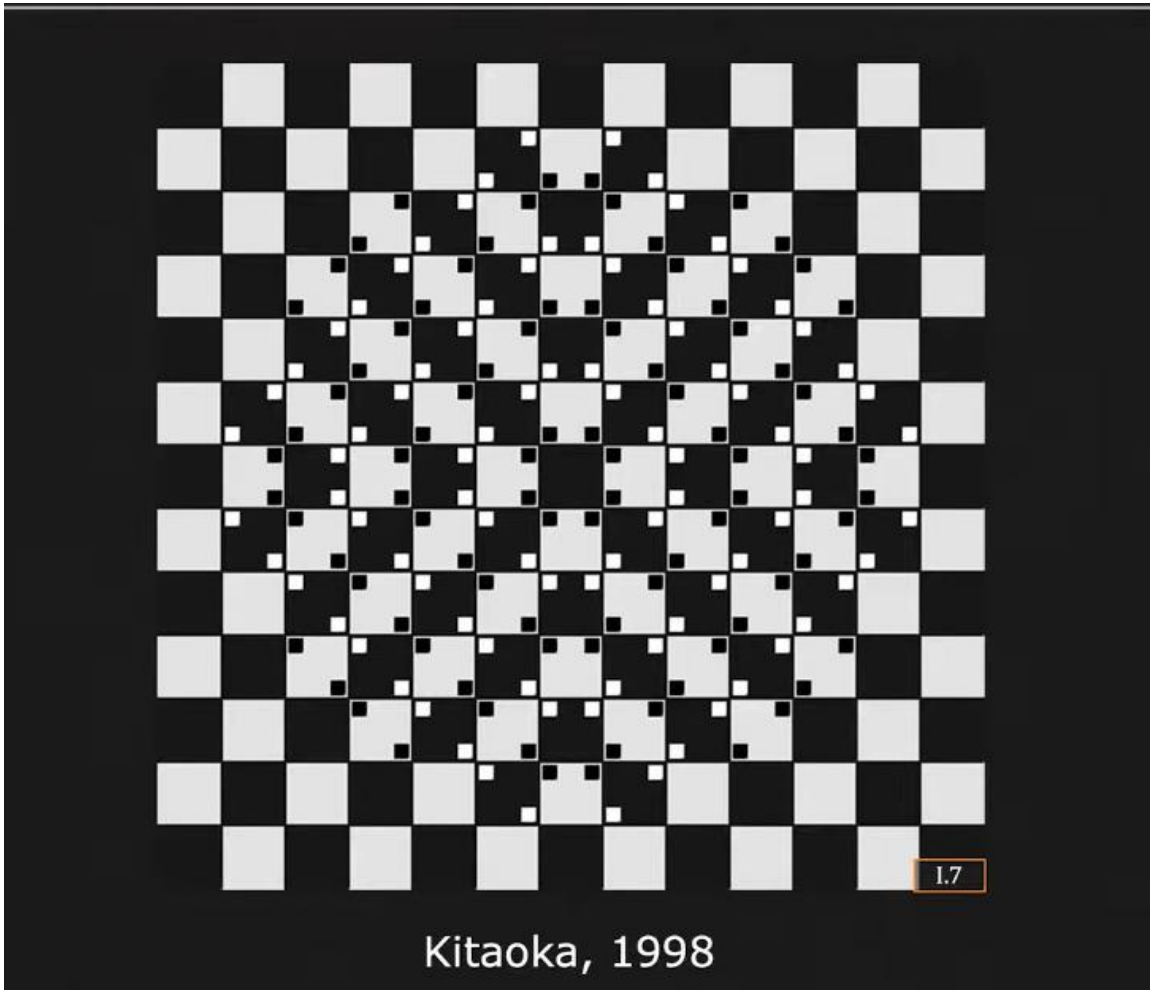


Primitives Detection

Key Primitives Detected in Computer Vision: corners

Corner Illusion: The Bulge

Here you see an image, a checkerboard like pattern, but when you look at it, you'll see that the center of the pattern actually bulges,
You see that the pattern has a lot of these tiny little dots that are placed strategically close to the corners

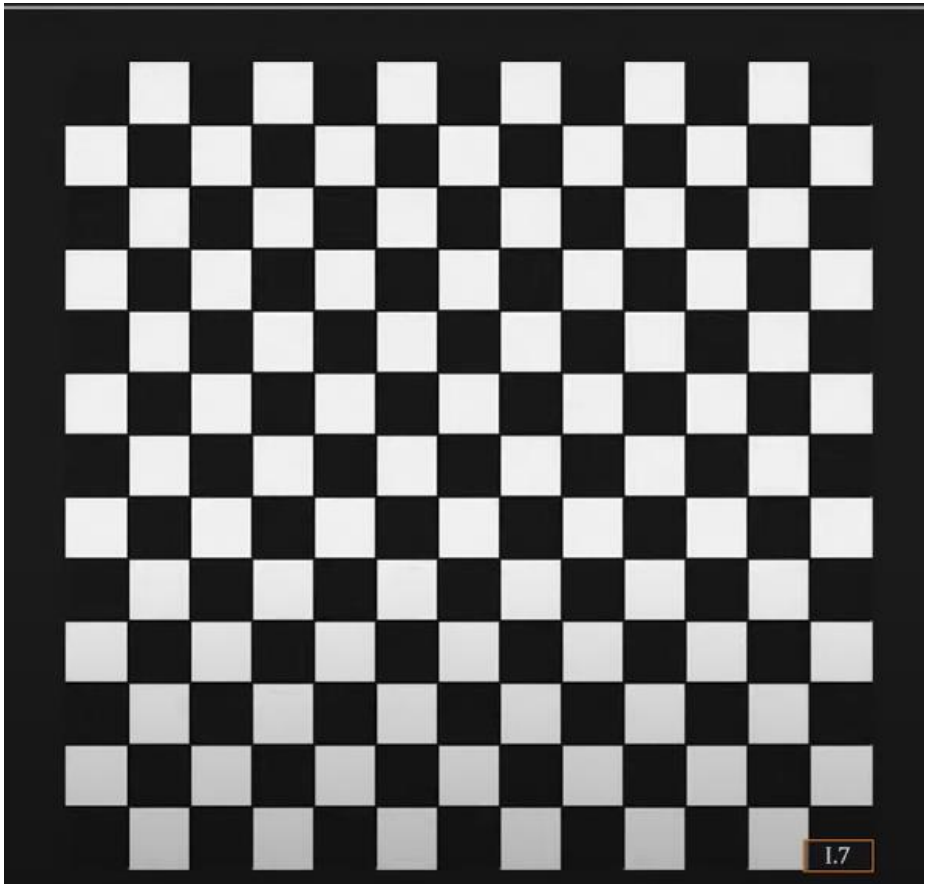


Primitives Detection

Key Primitives Detected in Computer Vision: corners

Corner Illusion: The Bulge

If you remove these tiny littles dots, you see that , indeed, you have a perfect checkerboard pattern (all the lines are either parallel or perpendicular to each other)

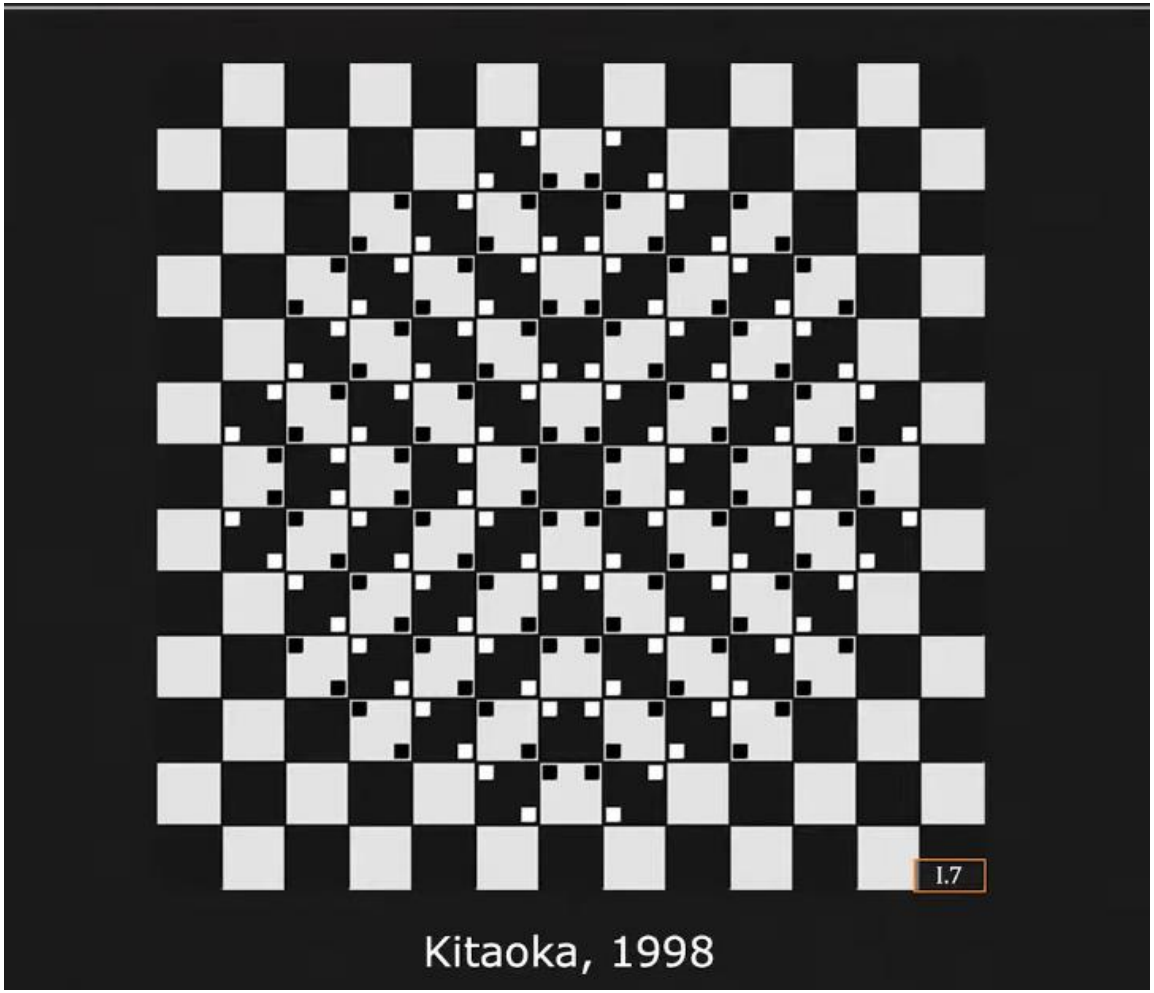


Primitives Detection

Key Primitives Detected in Computer Vision: corners

Corner Illusion: The Bulge

By placing these tiny squares close to the intersections in the main checkerboard pattern, you are actually biasing the human visual system to believe that the corners are at slightly different locations than they actually are,



Primitives Detection

3/ corners

Why the Harris Corner Detector is Important

1.Robust Corner Detection: The Harris Corner Detector is sensitive to corner-like structures and effectively identifies stable interest points, which are crucial for tracking, matching, and recognizing objects across images.

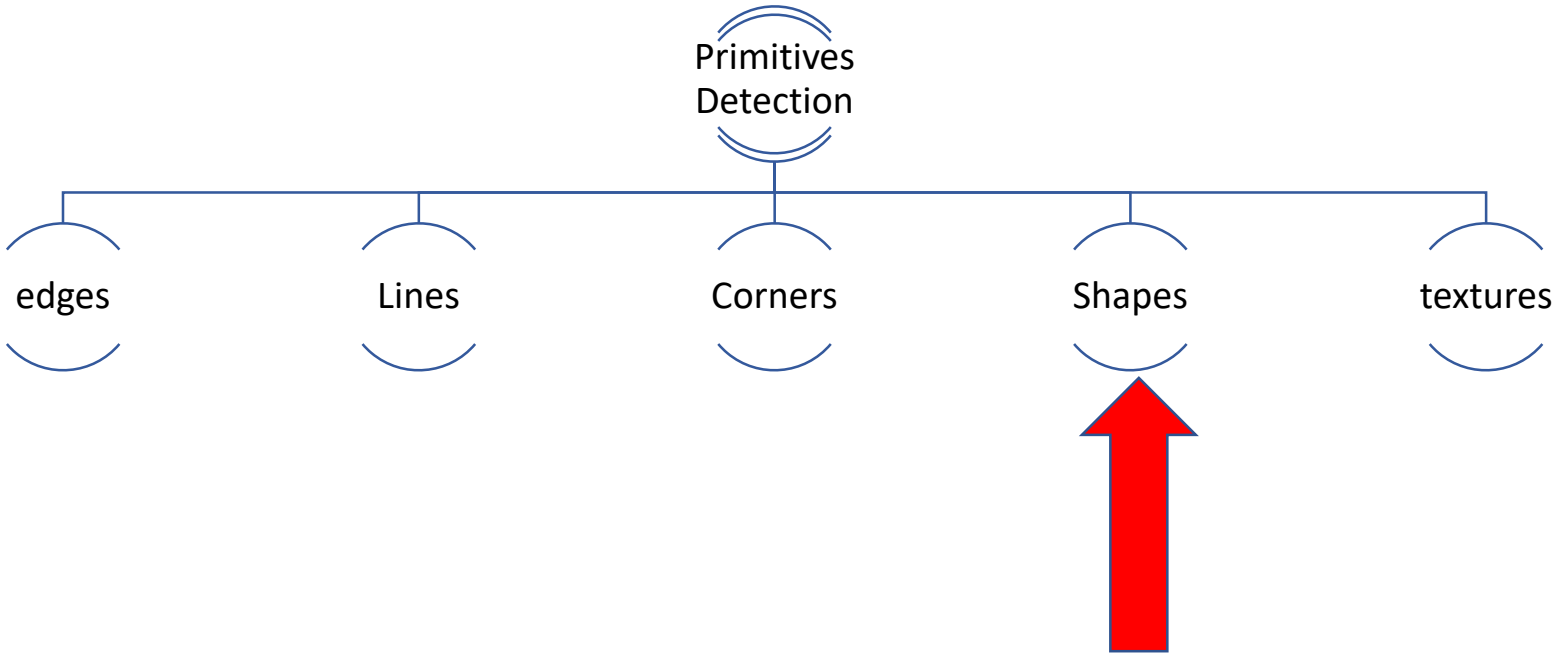
2.Rotation Invariance: The Harris Corner Detector is invariant to image rotation, meaning that it will detect the same corners even if the image is rotated. This property makes it useful in applications where orientation may vary, such as object tracking and image stitching.

3.Applications in Feature Matching: In computer vision tasks like image stitching, 3D reconstruction, and object recognition, reliable feature points are needed. The Harris Corner Detector provides these points, which can be used to match features across images for alignment and transformation estimation.

4.Efficiency: Although more advanced detectors (like SIFT or SURF) exist, the Harris Corner Detector is computationally efficient, making it a good choice for real-time applications and simpler systems where quick processing is important.

Primitives Detection

4/ Shapes



Primitives Detection

Key Primitives Detected in Computer Vision: shapes

4/ Shapes:

•**Description**: Primitives also include basic shapes like circles, squares, triangles, etc., which can be detected using contour analysis or geometric fitting algorithms.

•**Detection Methods**: Hough Circle Transform, RANSAC for fitting shapes.

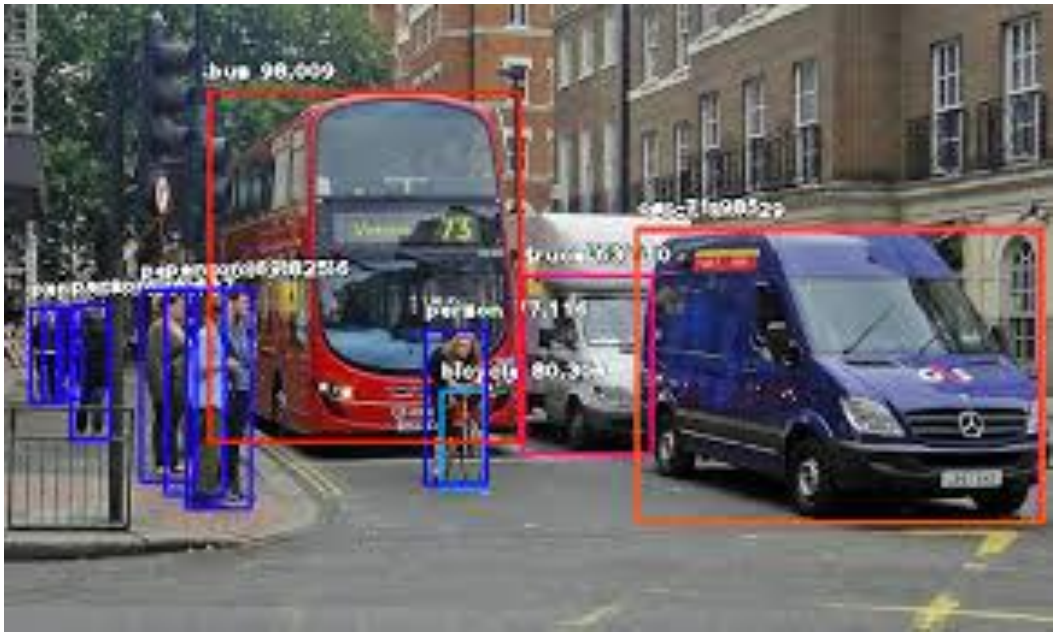
•**Application**: Shape detection is crucial in object recognition, robotics, and medical imaging.

Primitives Detection

Key Primitives Detected in Computer Vision: shapes

4/ Shapes:

•Shape detection is a process in computer vision that identifies and locates different geometric shapes (such as circles, rectangles, triangles, etc.) within an image. This technique is often used to analyze images and extract useful information based on the contours or characteristics of objects.



Primitives Detection

Key Primitives Detected in Computer Vision: shapes

Main Steps in Shape Detection

1.Image Preprocessing: Before detecting shapes, the image usually needs to be preprocessed to improve detection accuracy. This can include converting the image to grayscale, applying filters to reduce noise, and enhancing contrast.

2.Edge Detection: Algorithms like Canny edge detection are used to identify the edges of objects in the image. These edges are essential for identifying the boundaries of shapes.

3.Contour Approximation: Techniques like the Douglas-Peucker contour approximation can simplify detected contours into straight lines, making it easier to determine if they form specific shapes (like triangles or squares).

4.Detection of Specific Shapes: There are specialized algorithms for detecting certain shapes:

1. **Circle Detection**: The Hough Circle Transform is a common method for detecting circles within an image.
2. **Rectangle and Square Detection**: By analyzing contours, it's possible to check if a closed contour has four equal-length sides and right angles.
3. **Polygon Detection**: By counting the number of vertices in a contour and examining their angles, polygons (triangles, hexagons, etc.) can be identified.

5.Classification and Recognition: Once geometric shapes are identified, they can be classified or recognized for use in specific applications.

Primitives Detection

Key Primitives Detected in Computer Vision: shapes

Explanation of Classification

- **Triangles**: Identified by contours with 3 vertices.
- **Squares and Rectangles**: Identified by contours with 4 vertices. The aspect ratio (width/height) is used to differentiate between squares and rectangles.
- **Circles**: Detected using circularity. A contour with a high circularity (close to 1) is classified as a circle.
- **Polygons**: Any shape with more than 4 vertices that doesn't meet the circularity criteria is classified as a general polygon.

Primitives Detection

Key Primitives Detected in Computer Vision: shapes

Applications of Shape Detection

Shape detection has many uses, including:

- **Traffic sign recognition** in assisted driving systems.
- **Document analysis** to detect figures, charts, or checkboxes in forms.
- **Robotics** to allow robots to recognize objects with specific shapes.
- **Medical image processing** to identify particular cells or structures in medical images.

Shape detection is commonly implemented using computer vision libraries like OpenCV, which provides efficient functions for image processing and shape detection.

Primitives Detection

Key Primitives Detected in Computer Vision: shapes

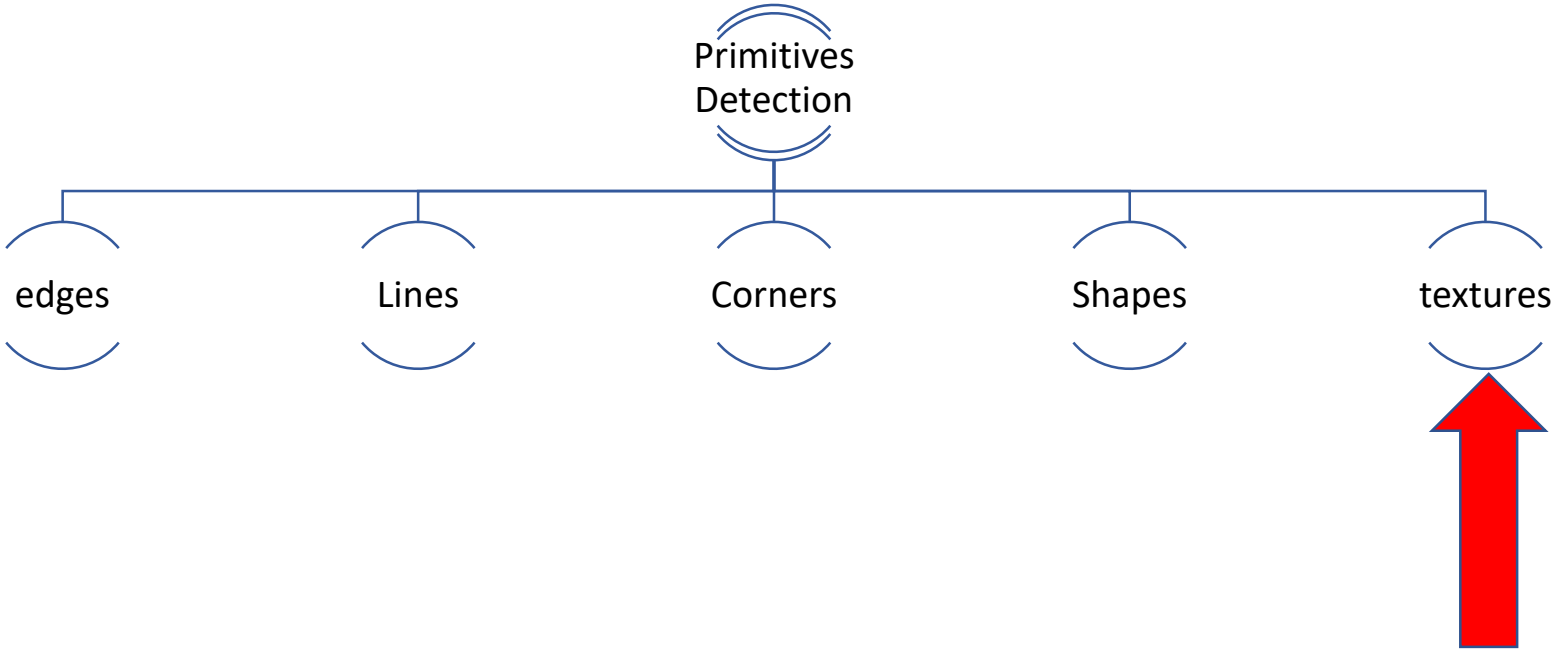
Opencv: the most commonly used functions for shape detection

Example Workflow for Shape Detection

1.Convert the image to grayscale (cv2.cvtColor).
2.Apply Gaussian blur (cv2.GaussianBlur) and threshold or edge detection (cv2.threshold or cv2.Canny).
3.Find contours (cv2.findContours).
4.For each contour, use approximation (cv2.approxPolyDP) and bounding functions to classify shapes.

Primitives Detection

5/ Textures



Primitives Detection

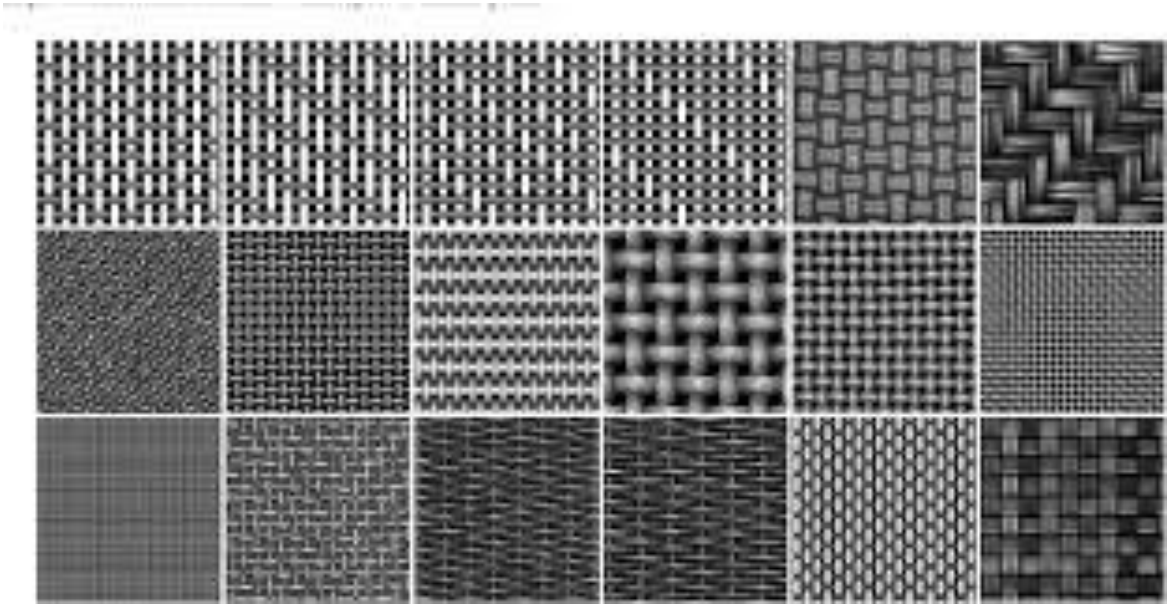
Key Primitives Detected in Computer Vision:

5/ Textures:

•**Description**: Texture is a primitive representing the surface quality or patterns within a region. It describes how pixels are arranged in space.

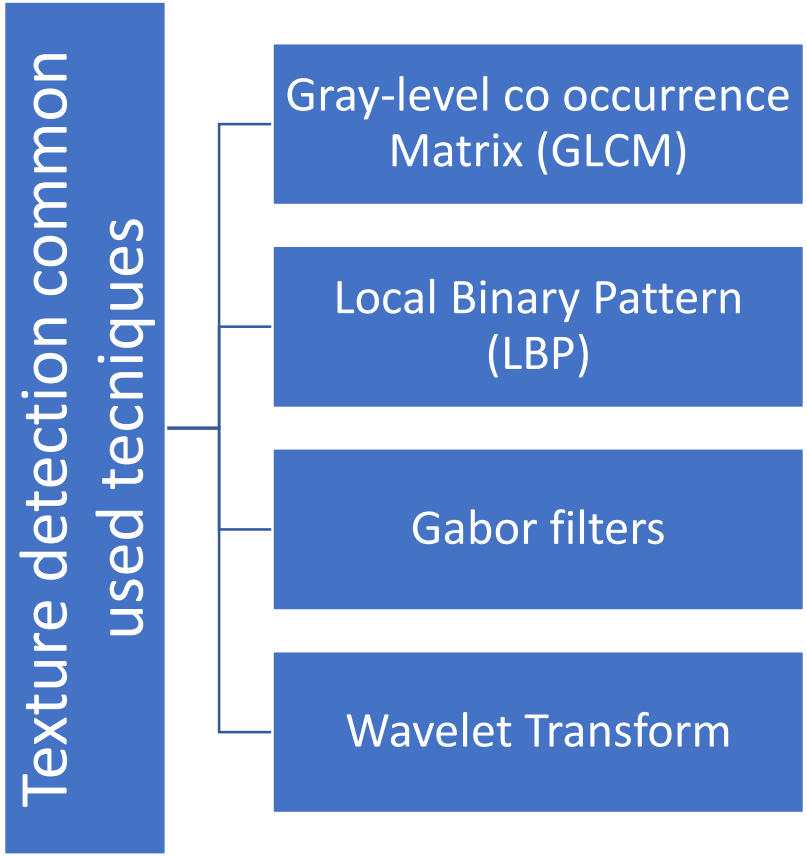
•**Detection Methods**: Local binary patterns (LBP), Gabor filters, and co-occurrence matrices are commonly used.

•**Application**: Important in image classification, material recognition, and surface analysis.



Key Primitives Detected in Computer Vision: textures

Here's an overview of some common techniques for texture detection, with examples.



Section 4:
image segmentation

IMAGE SEGMENTATION

- **image segmentation** is the process of dividing an image into meaningful regions.
- segmentation is crucial for **object recognition** and **scene understanding**.

Types of segmentation:

Image segmentation methods

- Thresholding
- Clustering
- Edge based methods
- Region based methods
- Graph based methods
- Deep Learning methods: (U-net, Mask R-CNN,,)

IMAGE SEGMENTATION

•**Thresholding** is a simple yet powerful method used for segmenting an image by converting it into a binary image. This process involves comparing each pixel's intensity value with a threshold value. If the pixel intensity is greater than or less than the threshold, the pixel is classified into one of two categories (foreground or background).

Image segmentation methods

- Thresholding
- Clustering
- Edge based methods
- Region based methods
- Graph based methods
- Deep Learning methods: (U-net, Mask R-CNN,,)

Thresholding

Binary (global)	• One threshold is applied
multilevel	• More than one threshold is applied
Local(adaptive)	• The threshold value is calculated for smaller regions of the image, making it useful for images with varying lighting conditions.
Otsu's	• One optimal threshold value is calculated by minimizing the intra-class variance between the foreground and background.

Thresholding

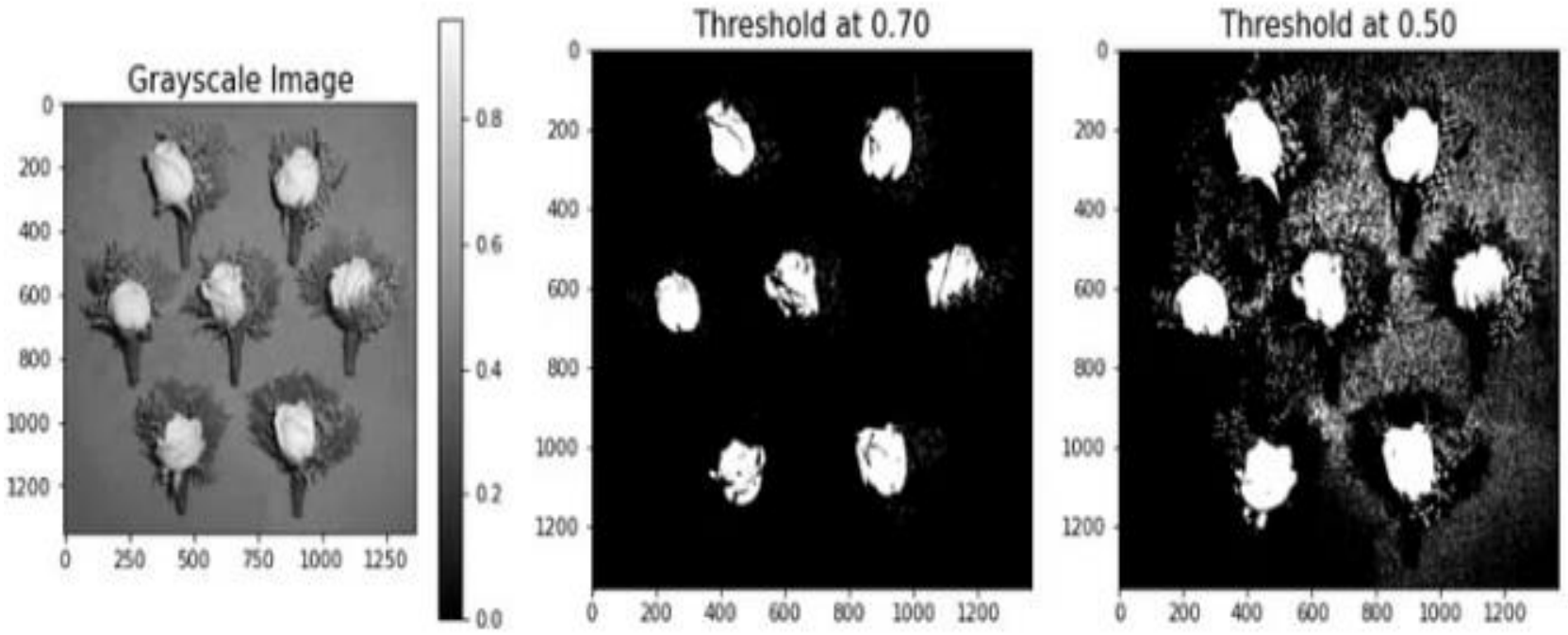
- **Binary (global)**
  - One threshold is applied
- **multilevel**
  - More than one threshold is applied
- **Local(adaptive)**
  - The threshold value is calculated for smaller regions of the image, making it useful for images with varying lighting conditions.
- **Otsu's**
  - One optimal threshold value is calculated by minimizing the intra-class variance between the foreground and background.

Binary Thresholding

Binary Thresholding:

Converts the image into two segments: foreground and background.

Example: Converting all pixels below a certain value to black and above to white.

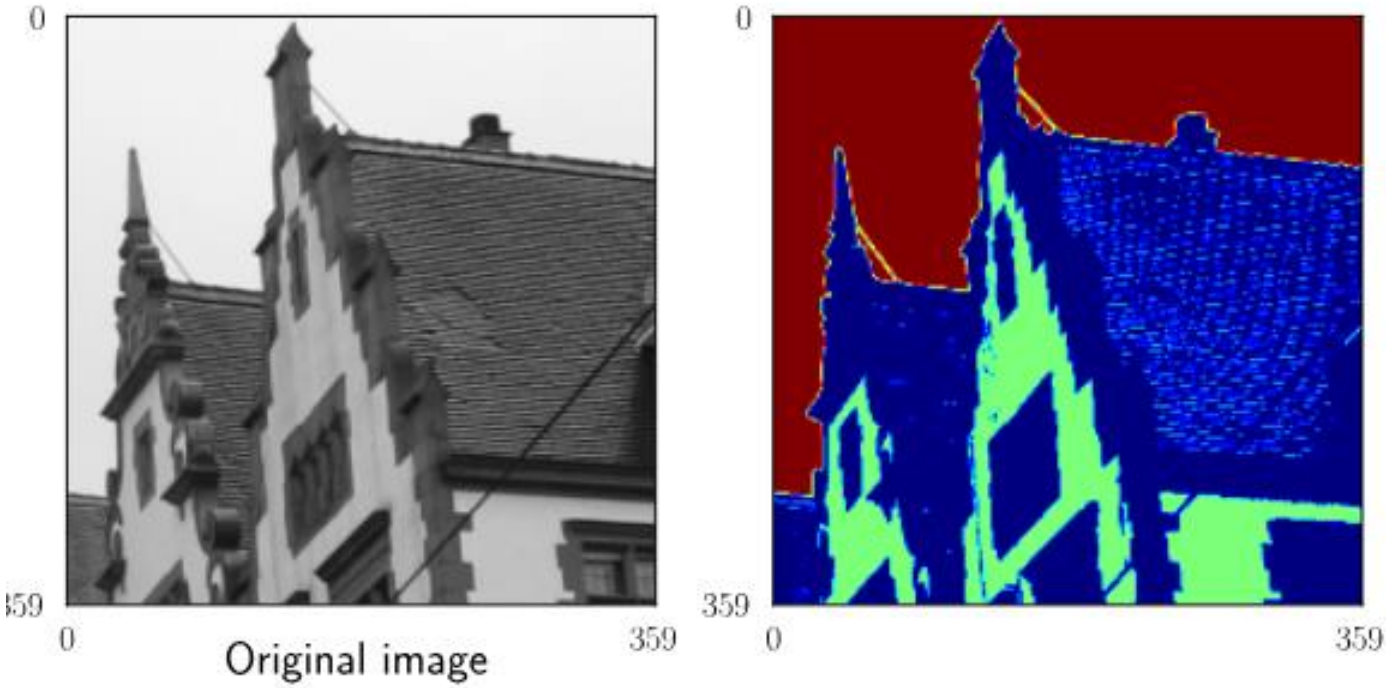


Thresholding

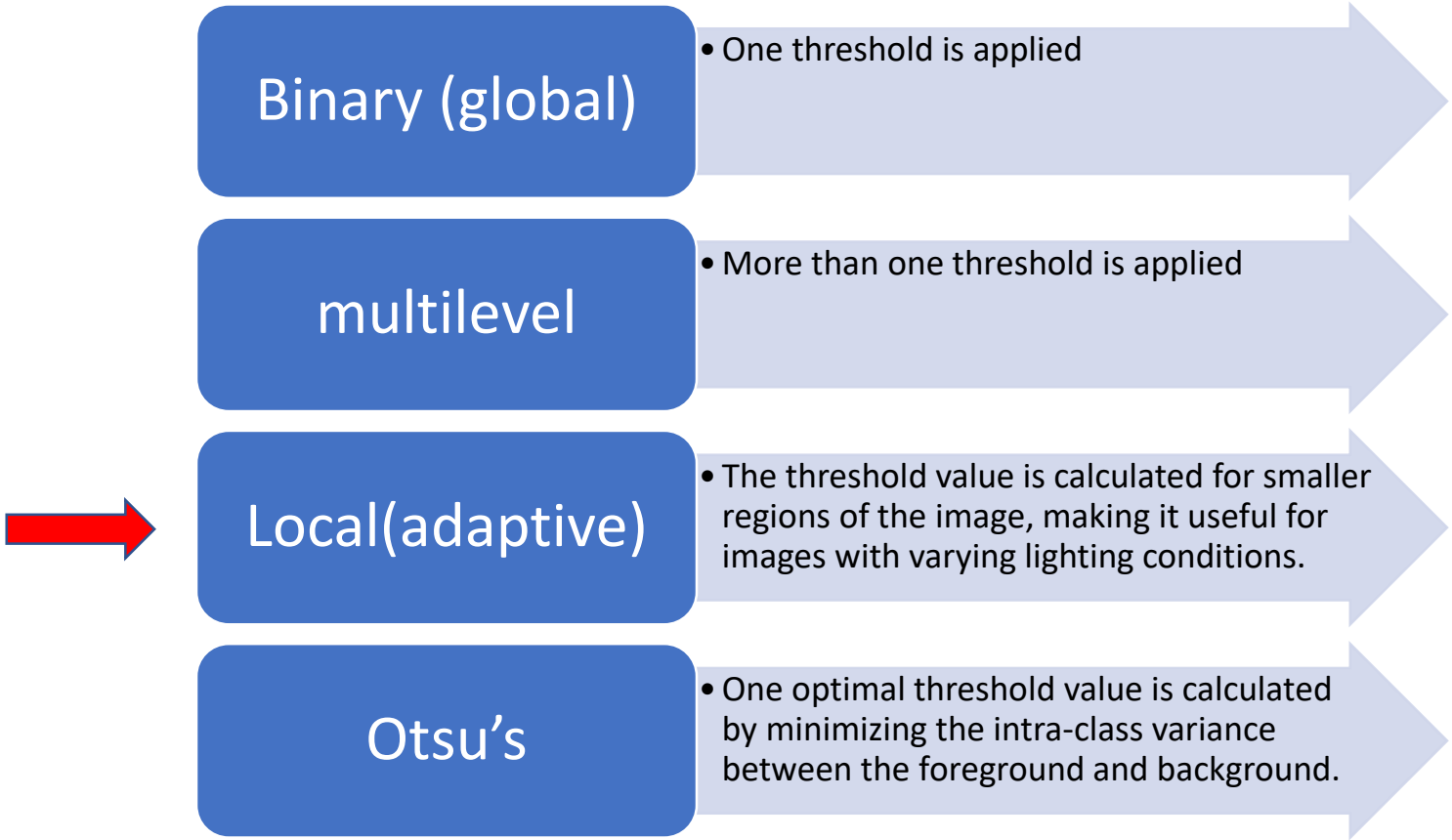
Binary (global)	• One threshold is applied
multilevel	• More than one threshold is applied
Local(adaptive)	• The threshold value is calculated for smaller regions of the image, making it useful for images with varying lighting conditions.
Otsu's	• One optimal threshold value is calculated by minimizing the intra-class variance between the foreground and background.

Multilevel Thresholding

Segments the image into more than two levels by selecting multiple thresholds.
Useful for images with more than two distinct intensity regions.
Example: several thresholds are applied to an image to get several classes (shown in colors).



Thresholding



Local (Adaptative) Thresholding

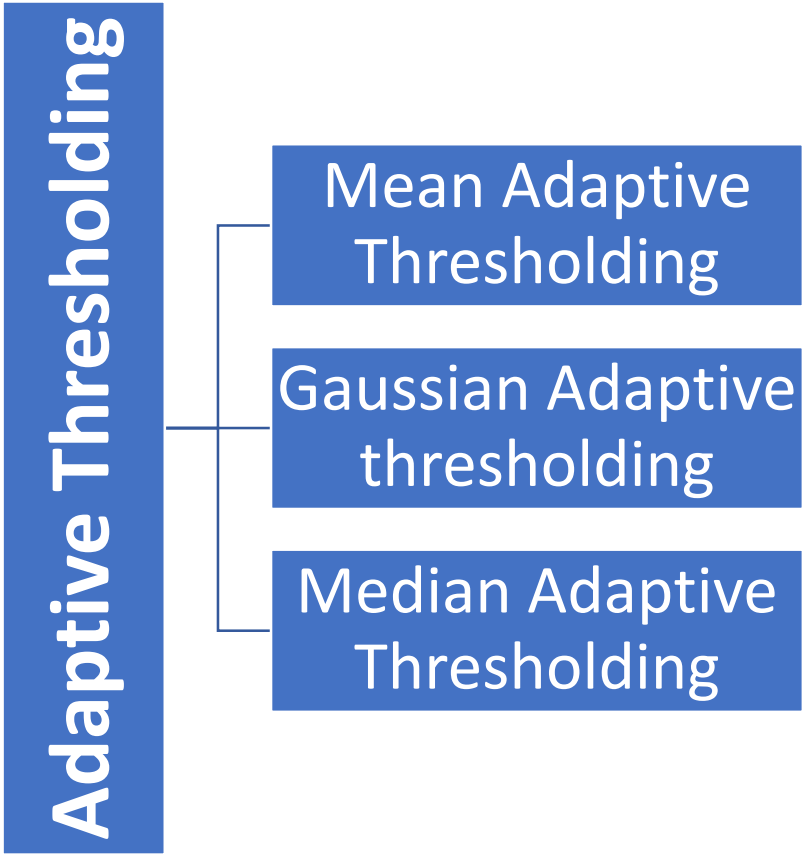
The threshold value is calculated for smaller regions of the image, making it useful for images with varying lighting conditions or non-uniform illumination.

.

Example: An image segmented into low, medium, and high-intensity regions.

Local (Adaptive) Thresholding

Types of Adaptive Thresholding



Local (Adaptative) Thresholding

Applications of Adaptive Thresholding

Document Image Processing:

Binarizing text in scanned documents with uneven lighting or background variations.

Example: Digitizing historical manuscripts.

Medical Imaging:

Segmenting tissues or identifying abnormalities in medical scans where illumination varies across the image.

Example: Retinal blood vessel segmentation.

Industrial Quality Control:

Detecting defects or patterns on surfaces under varying lighting conditions.

Example: Identifying cracks on metal sheets.

License Plate Recognition:

Enhancing plate numbers in images captured under low-light or shadowed conditions.

Traffic Monitoring:

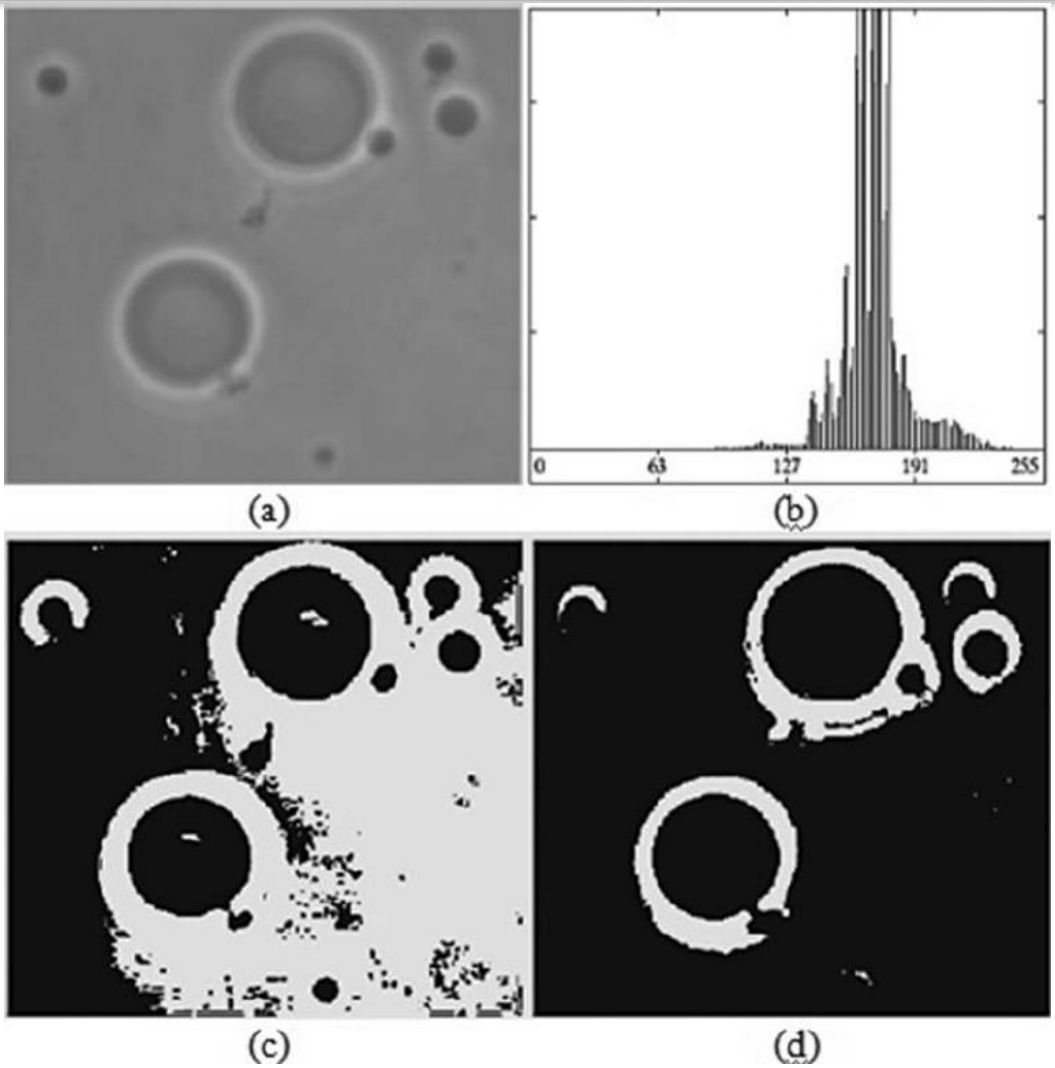
Detecting and segmenting objects like vehicles or pedestrians in surveillance images with non-uniform lighting.

Thresholding

Binary (global)	• One threshold is applied
multilevel	• More than one threshold is applied
Local(adaptative)	• The threshold value is calculated for smaller regions of the image, making it useful for images with varying lighting conditions.
Otsu’s	• One optimal threshold value is calculated by minimizing the intra-class variance between the foreground and background.

Otsu’s Thresholding

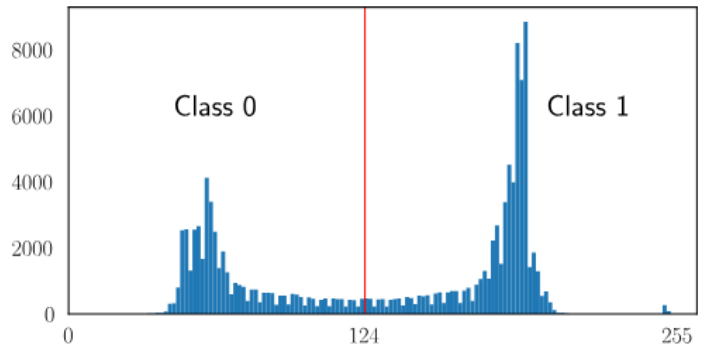
Otsu's Thresholding is a popular method used in image processing for binarizing images (converting them from grayscale to binary). Developed by Nobuyuki Otsu in 1979, this technique helps to separate an image into two classes based on intensity levels, aiming for the best separation that leads to the maximum between-class variance..



(a) Original image. (b) Histogram of an image.
(c) Global thresholding. (d) Otsu’s thresholding.

Otsu's Thresholding

- Compute the histogram and probabilities of each intensity level in the image.
- Initialize variables to track the optimal threshold:
- **Class probabilities** (w_0,w_1): The probability of a pixel belonging to the background or foreground.
- **Class means** (μ0,μ1): The mean intensity of each class.

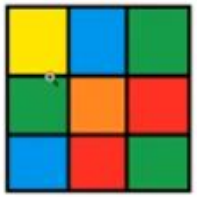


- For each potential threshold t:
  - Separate the pixels into two classes (below and above t).
  - Compute the intra-class variance for the two classes.
  - Track the threshold t that minimizes this intra-class variance.
  - Apply the optimal threshold t to binarize the image.

Otsu's Thresholding

1 Histogram and Probability

image



histogram

probability

total pixel = 9
P(red) = 2/9
P(yellow) = 1/9
P(blue) = 2/9
P(green) = 3/9
P(orange) = 1/9

2 Mean and Variance

histogram

$$\mu = \frac{[(0*2)+(1*1)+(2*2)+(3*3)+(4*1)]}{9} = 2$$

$$\sigma^2 = \frac{\sum_{i=0}^{4}(X_i - \mu)^2}{N}$$

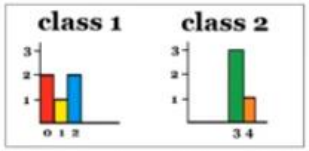
$$= \frac{(0-2)^2 \cdot 2 + (1-2)^2 \cdot 1 + (2-2)^2 \cdot 2 + (3-2)^2 \cdot 3 + (4-2)^2 \cdot 1}{9}$$

$$= 16/9 = 1.77\ldots$$

3 Within class variance

if pixels are classified into N classes (categories),
then the within class variance $(V_w) = \sum_{i=0}^{N}(W_i \cdot \sigma_i^2)$,
where W_i is (# of pixels in class i)/(totoal pixel)

divide into two classes



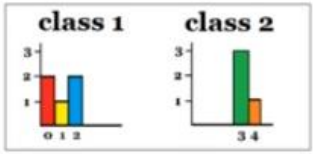
$W_1 = 5/9$ $W_2 = 4/9$
$\sigma_1^2 = 4/5$ $\sigma_2^2 = 3/16$

$V_w = W_1 \cdot \sigma_1^2 + W_2 \cdot \sigma_2^2 = 0.52777$

4 Between class variance

if pixels are classified into N classes (categories),
then the between class variance $(V_b) = V_T - V_w$,
where V_T is the total variance

if pixels are classified into 2 classes,
then the between class variance $(V_b) = W_1 W_2 (\mu_1 - \mu_2)^2$



$W_1 = 5/9$ $W_2 = 4/9$
$\mu_1 = 1$ $\mu_2 = 13/4$

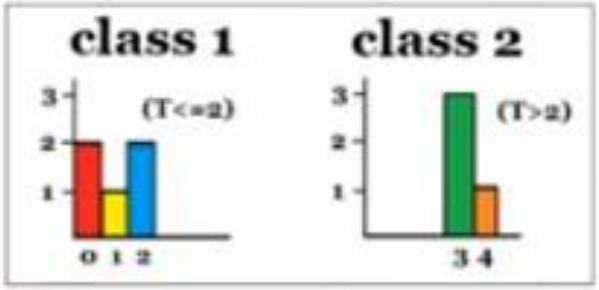
$V_b = \frac{5}{9} * \frac{4}{9} * (1 - \frac{13}{4})^2 = 1.25$

Otsu's Thresholding

Otsu thresholding

The aim is to find the threshold "T"
& classify pixels into 2 classes (class 1 & class 2)
so that the V_w is minimum (V_b is maximum)

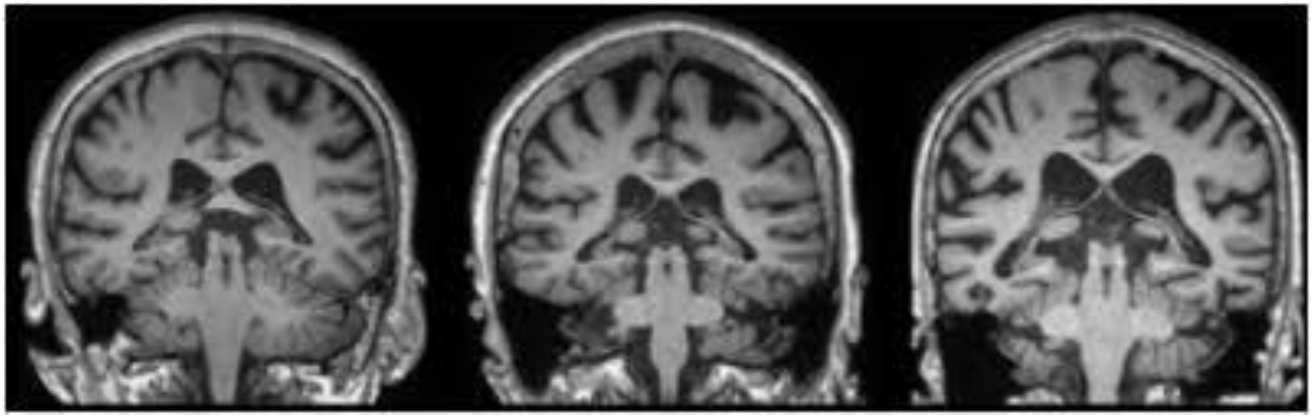
if T = 2

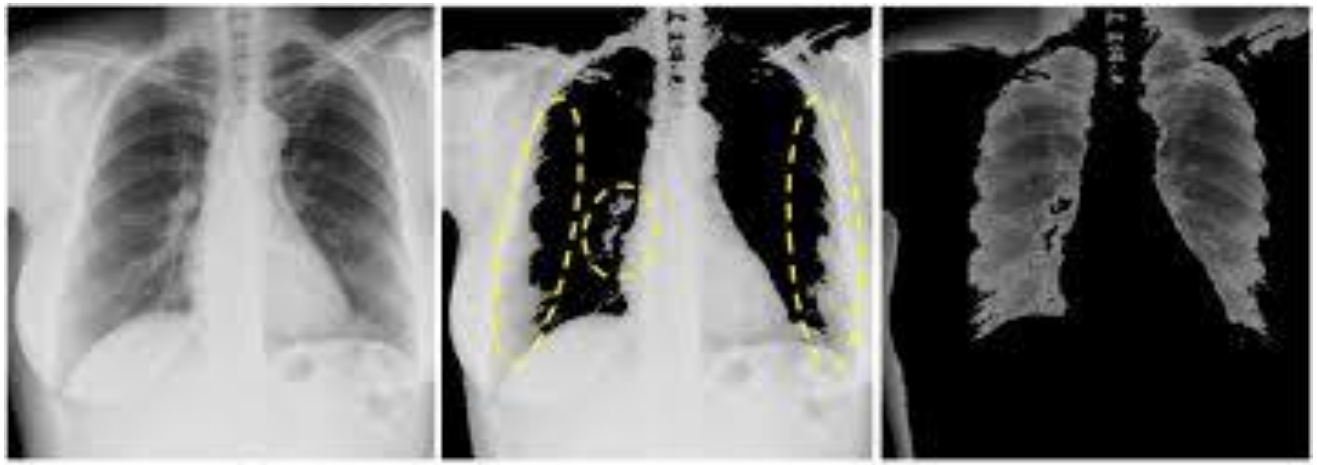


	T=0	T=1	T=2	T=3	T=4
V_b	1.142857	1.38888	1.25	0.5	0
V_w	0.63492	0.38888	0.52777	1.277777	1.777777

Otsu’s Thresholding applications

✓ **Medical Imaging**: Otsu's method can be used to segment structures in medical images, such as identifying tumors in MRI scans by distinguishing between tissue types based on intensity values.



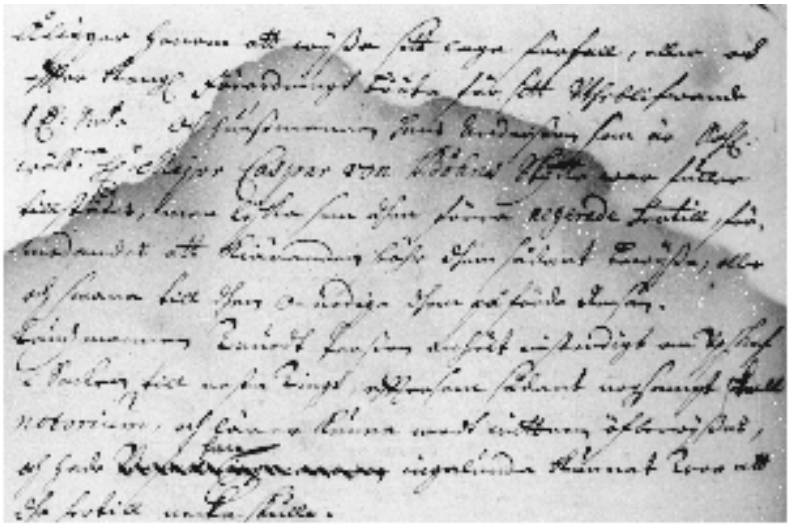


(a) (b)

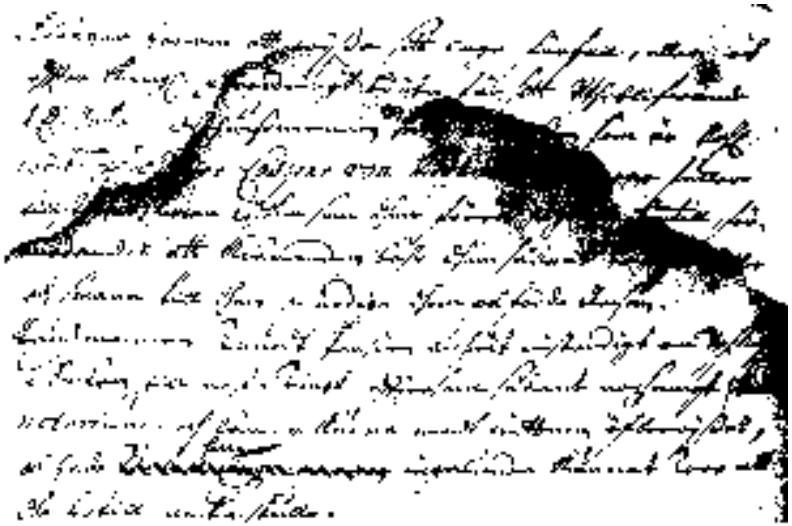
Otsu's Thresholding applications

✓ **Document Image Binarization**:

In document processing, Otsu's thresholding can help in separating text from the background effectively, facilitating OCR (Optical Character Recognition).

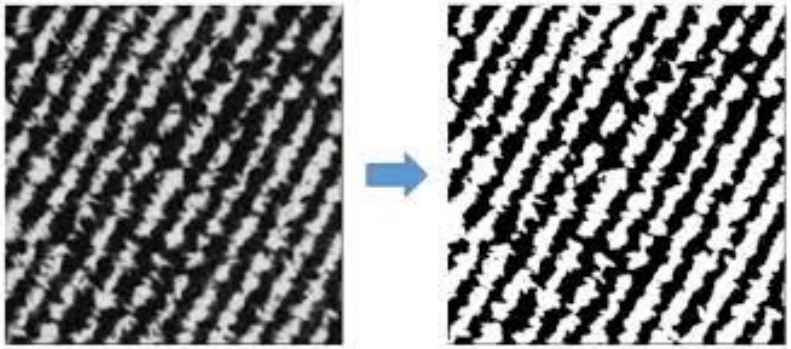


(a) Original Image



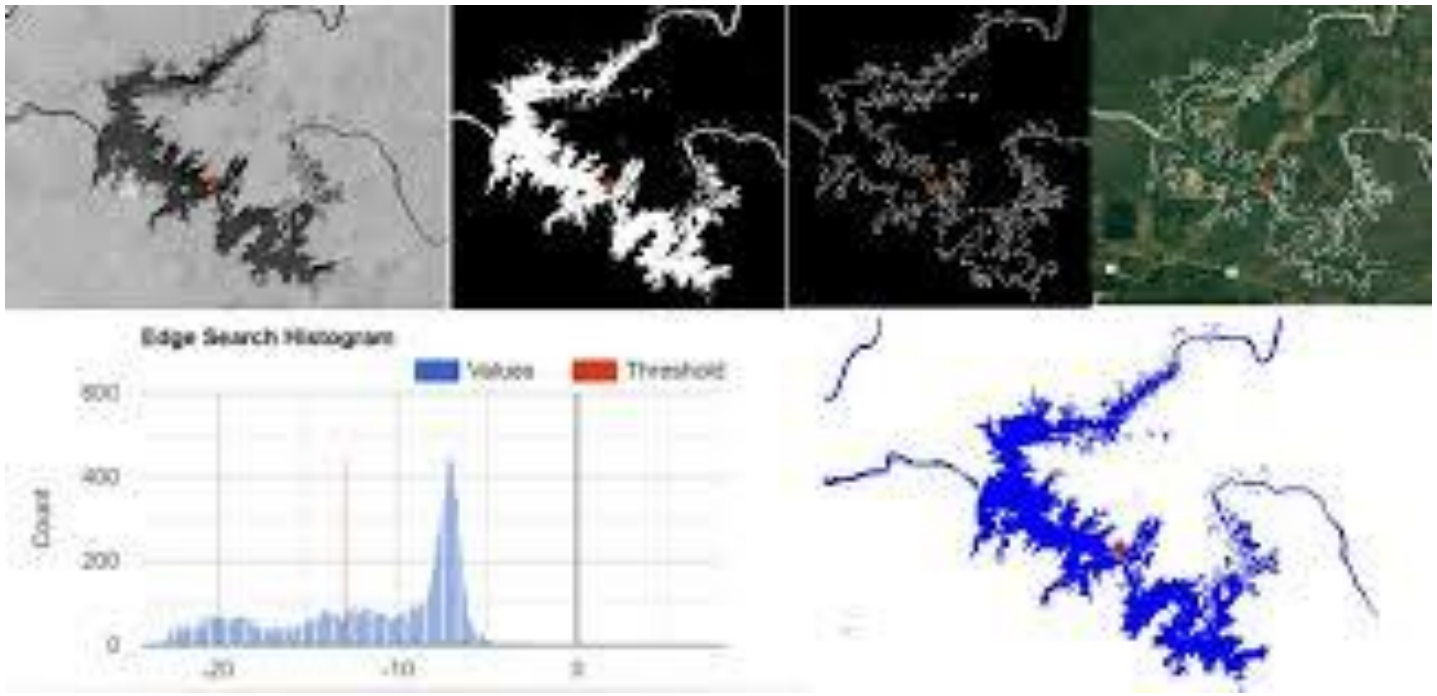
(b) Binarized (Otsu)

Otsu's Thresholding applications



✓ **Image Segmentation in Remote Sensing**:

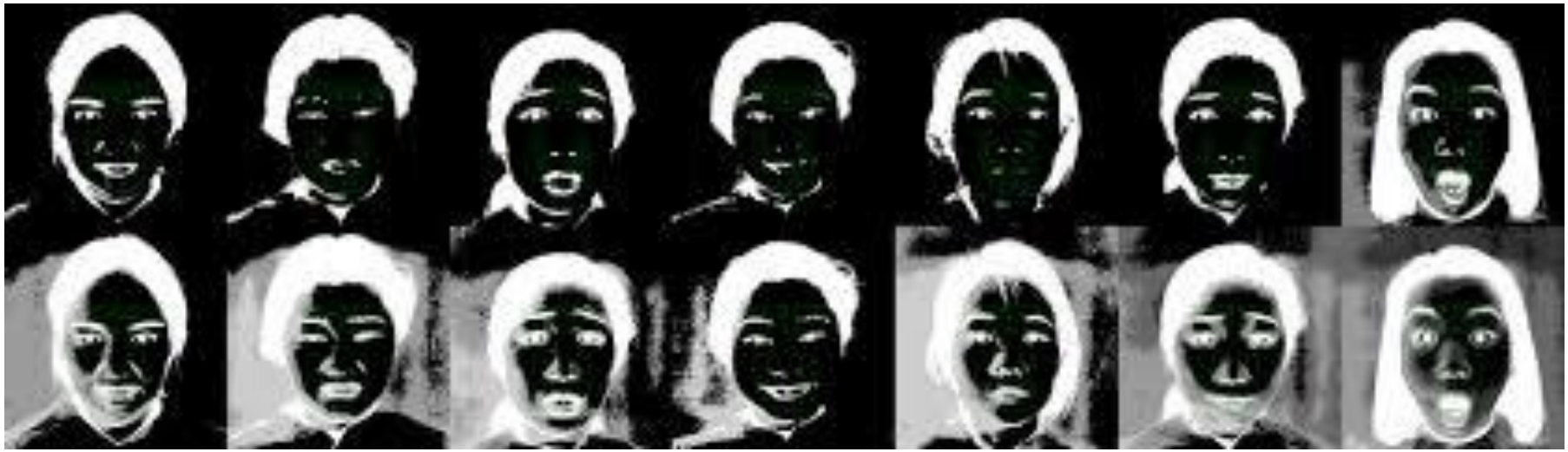
It's applied to segment land cover types in aerial imagery by maximizing the separation of classes represented by different intensity values in grayscale images, example: surface water mapping detection



Otsu's Thresholding applications

✓ **Face Detection**:

This technique can be used for preprocessing images to detect faces by distinguishing facial features from the background



Otsu thresholds. Row 1: three thresholds. Row 2: five thresholds

Global Vs Local Thresholding:

Global vs. Local Thresholding	**Global Thresholding**:	Global vs. Local Thresholding	**Local (Adaptive) Thresholding**:
	A single threshold value is applied across the entire image.		Different threshold values are calculated for smaller regions in the image.
	Suitable for images with uniform lighting and clear contrast between objects and background.		Useful in images with varying lighting conditions.

IMAGE SEGMENTATION

Clustering is an unsupervised machine learning technique used to group a set of objects (data points) into clusters such that objects within the same cluster are more similar to each other than to objects in other clusters. Similarity is typically determined using a distance metric, such as Euclidean distance.
Clustering is widely used in data analysis to discover patterns or structures in datasets without prior labeling of the data.

Image segmentation methods

- Thresholding
- Clustering
- Edge based methods
- Region based methods
- Graph based methods
- Deep Learning methods: (U-net, Mask R-CNN,,)

Clustering algorithms can be broadly classified based on their methodology:

- Partition–based Clustering → K-Means Clustering
- Hiearchical clustering → Agglomerative (Bottom-Up), Divisive (Top-Down)
- Density-based Clustering → DBSCAN (Density-Based Spatial Clustering of Applications with Noise), OPTICS (Ordering Points To Identify the Clustering Str
- Model-based Clustering → Gaussian Mixture Models (GMM)
- Grid- based Clustering → CLIQUE
- Spectral Clustering → Uses eigenvalues of a similarity matrix to perform dimensionality reduction and clustering in a lower-dimensional space.

Clustering Algorithms

Edge based methods:

Key Features of Edge-Based Segmentation

Focus on Boundaries: It identifies significant transitions in pixel intensity (edges) to define regions.

Efficient for Object Boundaries: Useful for applications requiring precise boundary delineation.

Gradient-Based: Many algorithms compute gradients to find regions with high intensity changes.

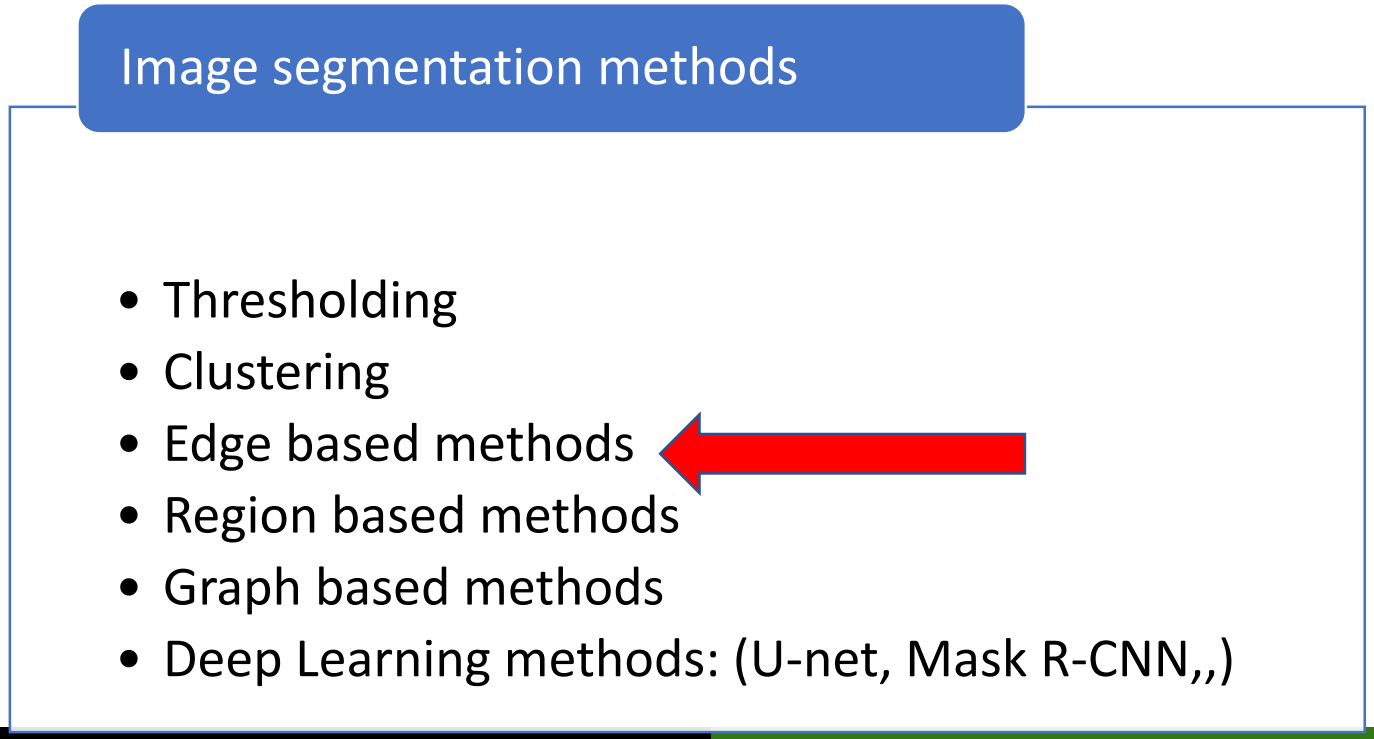


IMAGE SEGMENTATION

Edge based methods:

Types of Edge-Based Segmentation

Gradient-Based Methods:

Detect edges by identifying areas with a high rate of change in intensity.

Example: Sobel, Prewitt, Roberts operators.

Second-Order Derivative Methods:

Use the Laplacian operator to find edges by locating zero-crossings in the second derivative.

Example: Laplacian of Gaussian (LoG), Marr-Hildreth.

Canny Edge Detection:

Combines gradient computation with non-maximum suppression and hysteresis thresholding for robust edge detection.

It is considered one of the best edge detectors.

Edge Linking:

Post-processing step to connect detected edges into continuous contours.

Example: Hough Transform for line or curve detection.

Multi-Scale Edge Detection:

Uses different scales (e.g., Gaussian smoothing) to identify edges at various levels of detail.

Example: Wavelet-based edge detection.

IMAGE SEGMENTATION

Region based methods: is an image segmentation technique that groups pixels into regions based on shared characteristics, such as intensity, color, texture, or spatial proximity. Unlike edge-based methods that focus on boundaries, region-based methods identify entire regions in the image.

Image segmentation methods

- Thresholding
- Clustering
- Edge based methods
- Region based methods
- Graph based methods
- Deep Learning methods: (U-net, Mask R-CNN,,)

IMAGE SEGMENTATION

Key Features of Region-Based Segmentation

Region Homogeneity: Divides the image into regions that are homogeneous in terms of predefined criteria (e.g., intensity, color).

Connectivity: Ensures spatially connected regions.

Completeness: Tends to segment the entire image, leaving no gaps between regions.

IMAGE SEGMENTATION

Region-Based Segmentation algorithms:

	Region Growing	Region Splitting and Merging
Idea	Starts from a set of seed points and expands the region by adding neighboring pixels with similar properties.	Splits the image into smaller regions (usually quadtree structure) and then merges similar regions
Process	Select seed pixels (initial points).Compare neighboring pixels using a similarity criterion (e.g., intensity difference) Add similar pixels to the region and repeat.	Start with the entire image. Split regions that are not homogeneous. Merge adjacent regions that are similar
Strengths	Easy to implement.Produces connected regions	Balances over- and under-segmentation.
Weaknesses	Sensitive to seed selection. Prone to over-segmentation	Computationally expensive.

IMAGE SEGMENTATION

Graph based methods: Graph-based image segmentation represents an image as a graph, where each pixel (or a group of pixels) is a node, and edges between nodes represent the similarity or dissimilarity of their properties, such as intensity, color, or texture. The segmentation process divides the graph into subgraphs (segments) by optimizing a predefined criterion that reflects the similarity within a segment and the difference between segments.

Image segmentation methods

- Thresholding
- Clustering
- Edge based methods
- Region based methods
- Graph based methods
- Deep Learning methods: (U-net, Mask R-CNN,,)

IMAGE SEGMENTATION

Key Features of Graph-based Segmentation

Graph Representation: Models the image as a weighted graph G=(V,E), where:

V: Vertices (nodes), representing pixels or superpixels.

E: Edges connecting vertices.

Weight w(i,j): A measure of similarity (e.g., intensity difference) between two nodes.

Objective: Minimize the dissimilarity within a segment while maximizing the difference between segments.

Graph-based Segmentation

Steps in Graph-Based Segmentation

Graph Construction:

Nodes represent pixels, superpixels, or regions.

Edge weights reflect similarity (e.g., intensity difference, spatial distance).

Edge Weight Calculation:

Common metrics:

Intensity difference: $w(i,j)=|I(i)-I(j)|$

Color difference in RGB/HSV space.

Texture or gradient information.

Graph Partitioning:

Use algorithms like normalized cuts, min-cut/max-flow to partition the graph.

Post-Processing:

Optional refinement (e.g., smoothing, region merging).

IMAGE SEGMENTATION

Deep learning methods: Deep learning-based image segmentation uses neural networks to partition images into regions with semantic or structural significance. It leverages hierarchical feature extraction and end-to-end training to achieve high accuracy and robustness, particularly for complex tasks.

Image segmentation methods

- Thresholding
- Clustering
- Edge based methods
- Region based methods
- Graph based methods
- Deep Learning methods: (U-net, Mask R-CNN,,)

IMAGE SEGMENTATION

Types of Image Segmentation in Deep Learning:

Semantic Segmentation:

Classifies every pixel in an image into a predefined class (e.g., sky, road, car).
Example: Segmentation of different objects in a street scene.

Instance Segmentation:

Identifies individual instances of objects within a class (e.g., separate multiple people in an image). Combines object detection and semantic segmentation.

Panoptic Segmentation:

Merges semantic and instance segmentation to classify all pixels and distinguish object instances simultaneously.

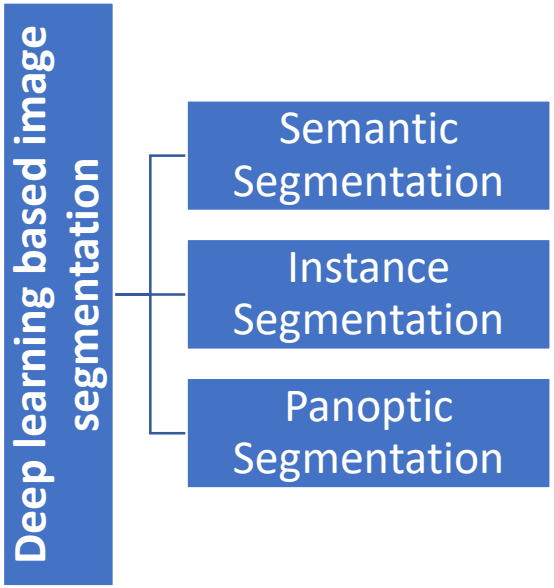


IMAGE SEGMENTATION

Deep Learning Architectures for Image Segmentation

1/ Fully Convolutional Networks (FCNs):

First deep learning architecture designed for semantic segmentation.

Replaces fully connected layers with convolutional layers, allowing the model to handle variable image sizes.

Output: Per-pixel classification map.

2/ U-Net:

A popular architecture with an encoder-decoder structure.

Encoder: Extracts hierarchical features using convolutional and pooling layers.

Decoder: Upsamples the features and refines spatial details using skip connections.

Application: Biomedical image segmentation, such as segmenting cells or organs.

3/ SegNet:

Similar to U-Net but emphasizes efficient memory usage.

Encoder-decoder architecture with pooling indices to improve the upsampling process.

Application: Road and scene segmentation.

4/ DeepLab:

Developed by Google, it incorporates:

- Atrous (dilated) convolutions for multi-scale context.
- Conditional Random Fields (CRFs) for boundary refinement.

Variants: DeepLabv2, DeepLabv3, DeepLabv3+.

Application: Semantic segmentation in autonomous driving.

Deep Learning Architectures for Image Segmentation

5/ Mask R-CNN:

Extends Faster R-CNN for instance segmentation.
Combines object detection and a pixel-level segmentation mask prediction.
Application: Instance segmentation tasks like person segmentation in videos.

6/ Pyramid Scene Parsing Network (PSPNet):

Uses a pyramid pooling module to capture global and local context.
Designed for robust semantic segmentation in complex scenes.
Application: Urban scene understanding.

7/ Attention-Based Models:

Incorporate attention mechanisms to focus on relevant parts of the image.
Examples: Attention U-Net, Transformers for Vision (e.g., SegFormer).

8/ Vision Transformers (ViTs):

Treat images as sequences of patches and process them using self-attention mechanisms.
Application: Advanced semantic segmentation tasks.

✓Section 5:
Challenges in low-level Image Analysis

Challenges in Low-Level Image Analysis

- **Noise and Artifacts**: Sensor noise or transmission errors distort image quality.
- **Illumination Variations**: Changing lighting conditions affect consistency in feature extraction.
- **Scale and Rotation Sensitivity**: Difficulty handling objects at different sizes or orientations.
- **Ambiguous Boundaries**: Weak or blurred edges lead to inaccurate segmentation.
- **High Computational Costs**: Resource-intensive algorithms hinder real-time applications.
- **Lack of Context**: Low-level methods lack semantic understanding of scenes.
- **Blur and Focus Issues**: Motion blur and out-of-focus regions degrade edge and feature quality.
- **Texture and Pattern Variability**: Complex textures and repetitive patterns confuse segmentation.
- **Low Contrast**: Poor distinction between objects and background affects accuracy.
- **Real-Time Processing Needs**: High latency in applications like autonomous driving.
- **Device Variability**: Inconsistent results due to differences in sensors or devices.
- **Occlusions**: Hidden objects complicate feature extraction.
- **Dataset Bias**: Algorithms may fail on data with characteristics different from training datasets.
- **Image Misalignment**: Misregistration of multi-image data affects results.
- **Large Data Volumes**: High-resolution images or datasets overwhelm computational resources

conclusion

Strategies to Address Challenges

1.Preprocessing:

1. Noise reduction, contrast enhancement, and illumination correction.

2.Hybrid Approaches:

1. Combine low-level processing with machine learning or deep learning for robustness.

3.Parallelization:

1. Leverage modern hardware like GPUs or multi-core processors for speed.

4.Adaptive Techniques:

1. Use adaptive thresholding or region-growing methods that adjust to image characteristics.

Low-level image processing remains a critical field, and addressing these challenges is essential for reliable feature extraction and subsequent stages of computer vision.

References

1. Dana H. Ballard & Christopher M. Brown. Computer Vision Prentice Hall, Inc, 1982
2. Robert M. Haralick & Linda G. Shapiro. Computer and Robot Vision, Vol-I, Addison-Wesley Publishing Company, 1992
3. Robert M. Haralick & Linda G. Shapiro. Computer and Robot Vision, Vol-II, Addison-Wesley Publishing Company, Inc, 1993
4. Linda Shapiro & Azriel Rosen eld. Computer Vision and Image Processing, Academic Press, Inc, 1992
5. Berthold Klaus Paul Horn. Robot Vision , MIT Press McGraw-Hill Book Company, 1986
6. Robert J. Schalko. Digital Image Processing and Computer Vision, John Wiley & Sons Inc, 1989
7. George Stockman and Linda Shapiro. Three Dimensional Computer Vision. Prentice Hall 2000.
8. David Marr. Vision, W. H Freeman and Company, NY, 1982
9. Rafael C. Gonzalez and Paul Wintz. Digital Image Processing, Third edition, Addison Wesley, MA. (Now with Prentice Hall, eective 1999).
10. Ernest Hall. Computer Image Processing and Recognition, second edition, Academic press 1982.
11. Azriel Rosenfeld and Avinash C. Kak. Digital Picture Processing, Vol. 1 & Vol. 2, Academic Press, 1982.
12. Robert J. Schalko. Digital Image Processing and Computer Vision: An introduction to theory and implementations, John Wiley & Sons, New York, 1989.
13. William K. Pratt. Digital Image Processing, John Wiley & Sons, 1993.