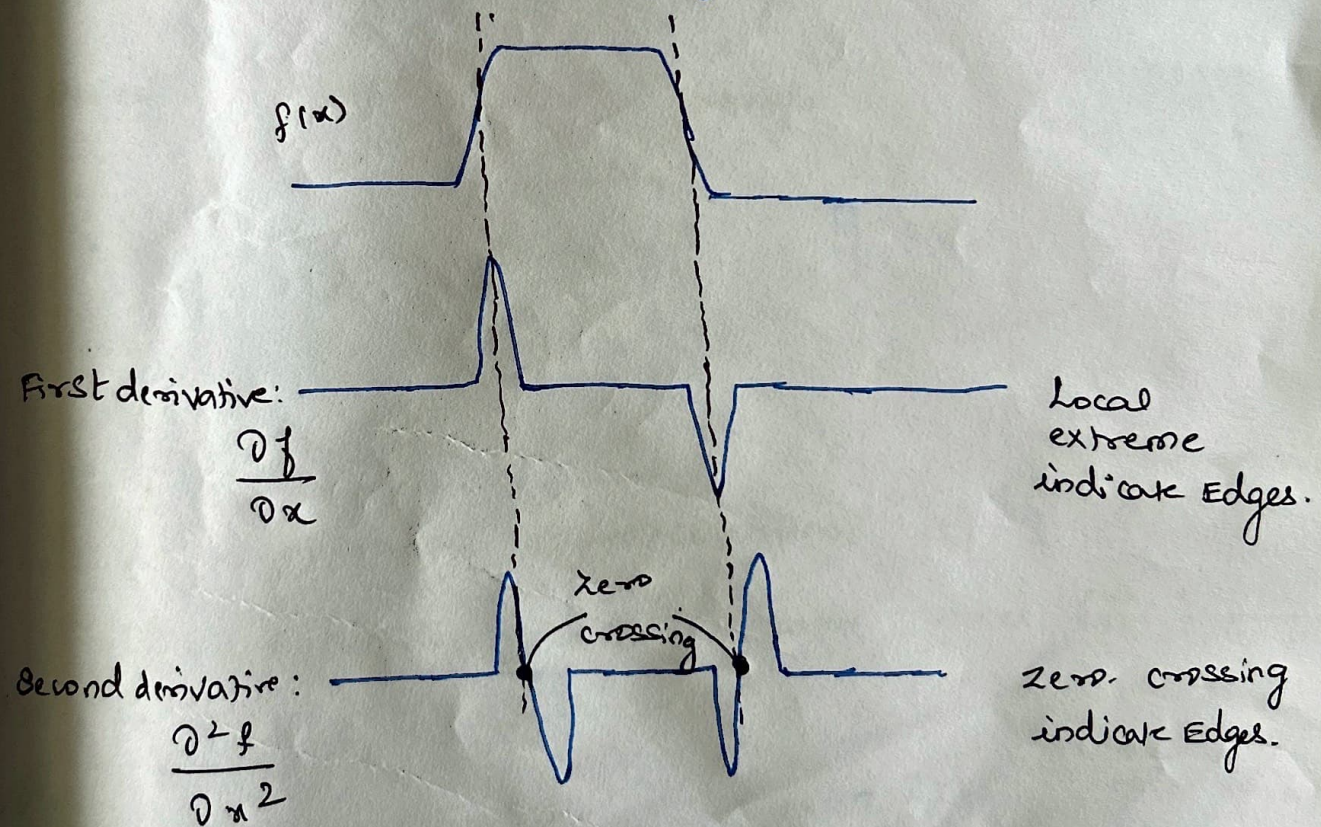
Unit III - Feature Extraction:

Laplacian of Gradient (LoG) Edge detection. [Edge Detection using 2nd Derivative:-

* Laplacian of Gradient is also known as Laplacian of Gaussian (LoG) (or) Mexican hat.
It is a second-order derivative operator used for edge detection and feature extraction.

* LoG identifies edge locations by finding zero-crossings in the second derivative.

Let us assume, the 1D sig $f(x)$



* In second derivative => At the edges, we do not get peaks, but we get zeros <> very strong zero-crossings.

Laplacian: Sum of pure second derivatives.

$$\nabla^2 I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2} \rightarrow (1)$$

Gradient = ∇ operator

Laplacian = ∇^2 operator

Laplacian is the sum of the 2nd derivative of Image w.r.to x and the 2nd derivative of Image w.r.to y.

* When we apply laplacian operator over an image, we end-up with zero crossings at the edges.

* Laplacian operator dosnot provide the direction or the orientation of the edge.

Discrete laplacian (∇^2) operator:-

$I_{i-1,j-1}$	$I_{i,j-1}$	$I_{i+1,j-1}$
$I_{i-1,j}$	$I_{i,j}$	$I_{i+1,j}$
$I_{i-1,j+1}$	$I_{i,j+1}$	$I_{i+1,j+1}$

ε

Finite difference approximations:

To find laplacian for the center pixel ($I_{i,j}$)

$$\frac{\partial^2 I}{\partial x^2} = \frac{\left(\frac{I_{i-1,j} - I_{i,j}}{\varepsilon}\right) - \left(\frac{I_{i,j} - I_{i+1,j}}{\varepsilon}\right)}{\varepsilon}$$

$$\frac{\partial^2 I}{\partial x^2} = \frac{I_{i-1,j} - I_{i,j} - I_{i,j} + I_{i+1,j}}{\varepsilon^2}$$

$$\frac{\partial^2 I}{\partial x^2} = \frac{1}{\varepsilon^2}\left(I_{i-1,j} - 2I_{i,j} + I_{i+1,j}\right) \rightarrow (4)$$

w.r.t y

$$\frac{\partial^2 I}{\partial y^2} = \frac{1}{\varepsilon^2}\left(I_{i,j-1} - 2I_{i,j} + I_{i,j+1}\right) \rightarrow (5)$$

Sub (4) & (5) in (1),

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

$$= \frac{1}{\varepsilon^2}\left(I_{i-1,j} - 2I_{i,j} + I_{i+1,j}\right) + \frac{1}{\varepsilon^2}\left(I_{i,j-1} - 2I_{i,j} + I_{i,j+1}\right)$$

$$= \frac{1}{\varepsilon^2}\left[I_{i-1,j} - 4I_{i,j} + I_{i+1,j} + I_{i,j-1} + I_{i,j+1}\right]$$

Convolution Mask:

$$\nabla^2 = \frac{1}{\varepsilon^2}$$

0	1	0
1	-4	1
0	1	0

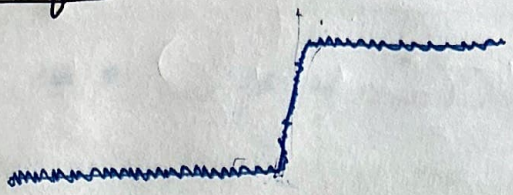
(or)

$$\nabla^2 = \frac{1}{6\varepsilon^2}$$

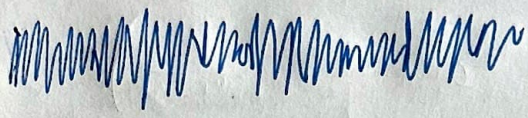
1	4	1
4	-20	4
1	4	1

⇒ The disadv of this operator is, we are in a square grid but edges can appear in any orientation. Here diagonal pixel about the center pixel is zero, so modify the fifth mask.

Effects of Noise:-



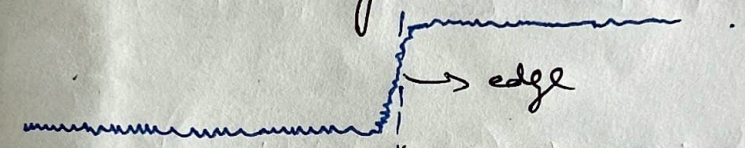
$\nabla f(x)$ (Gradient)



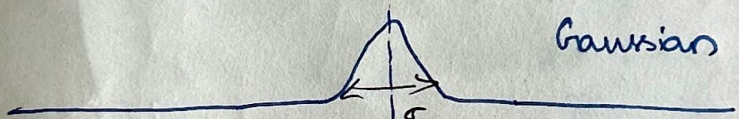
⇒ The first derivative of $f(x)$ ⇒ lost the edge.

* So we need to suppress the noise, before edge detection (ie) before the application of gradient operator. (or) Laplacian operator.

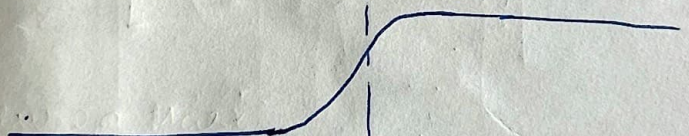
Gaussian Smoothing:- (Blur)



noisy signal.



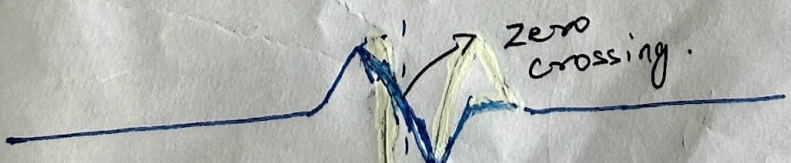
⇒ Smooth it with Gaussian σ → width.



⇒ After applying Gaussian.

⇒ gradient operator (1st derivative)

$\nabla(n_\sigma * f) = \nabla(n_\sigma) * f$



Laplacian operator (2nd derivative)

$\nabla^2(n_\sigma * f) = \nabla^2(n_\sigma) * f$

Gradient	Laplacian
i) Provides location, magnitude and direction of the edge	i) Provides only location of the edge.
ii) Detection using maxima Thresholding	ii) Detection based on zero crossing.
iii) Non-linear operation. Requires two convolutions	iii) Linear operation. Requires only one Convolution

— x —

Hough Transform:

* Paul Hough introduced the concept of Hough Transform in 1962.
* Hough Transform is a feature extraction method used to find basic shapes in a picture like circles, lines and ellipses.
* It transfers these shapes representation from the spatial domain to the parameter allowing for effective detection even incase of distortions like noise or occlusion.
* Hough Transform is used to detect the boundaries of an image.

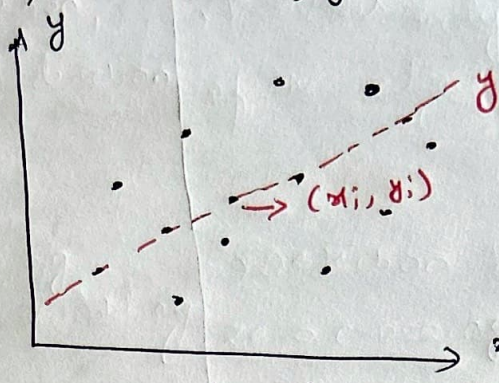
Problem in Edge map:

1. Extraneous Data: which points to fit to?
2. Incomplete Data: only part of the model is visible
3. Noise.

Solution: Hough Transform → It deals with these 3 Problems.

Hough Transform for Line Detection:

Given: Edge points (x_i, y_i)

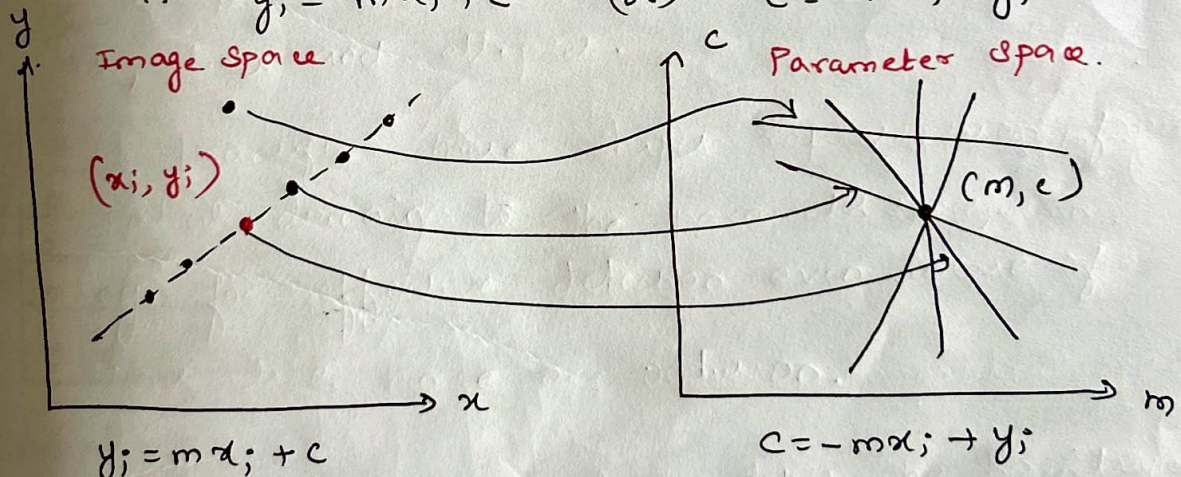


* The above graph shows the bunch of edges in an image and embedded straight line is found
* The goal is to find the straight line, where the edges actually lie exactly, on the st. line.
* The equation of the st. line is

$$y = mx + c \quad \text{where } m \rightarrow \text{slope}, \; c \rightarrow \text{Intercept}.$$

* Let us consider one point on the straight line (x_i, y_i)

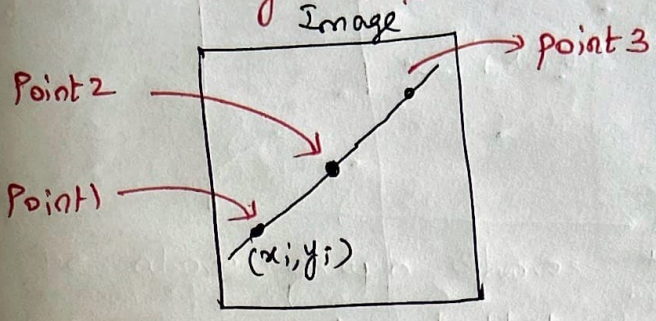
$$\therefore \quad y_i = mx_i + c \quad (\text{or}) \quad c = -mx_i + y_i$$



* A point which does not lie on the line ⇒ forms a line in parameter space but does not pass thro' the point (m, c).
* All points line on the st. line in Image Space forms a line on the Parameter space, all intersect at the point (m, c).

* A point in Image space → maps to line in Parameter Space.
* A line in Image space → maps to point in Parameter Space.

Line detection Algorithm:-



Step1: Quantize parameter space (m, c)

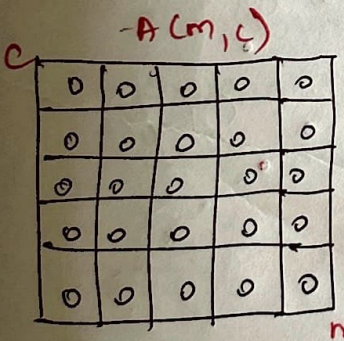
Step2: Create accumulator array A(m, c)

Step3: Set A(m, c) = 0 for all (m, c)

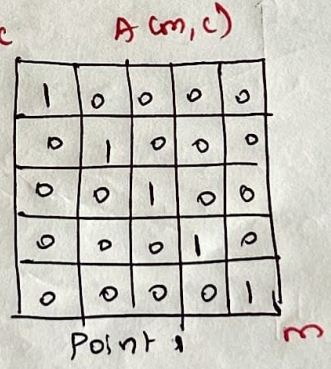
Step4: For each edge point (xi, yi), update accumulator array by 1.

$$A(m,c) = A(m,c) + 1 \text{ , if } (m,c) \text{ lies on the line: } c = -mx_i + y_i$$

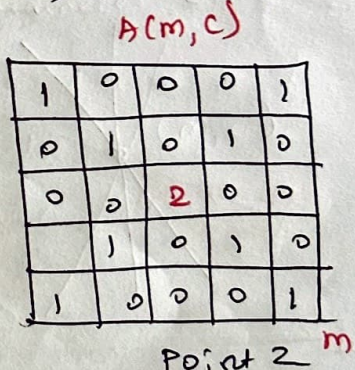
Step 5: Find local maxima in A(m, c)



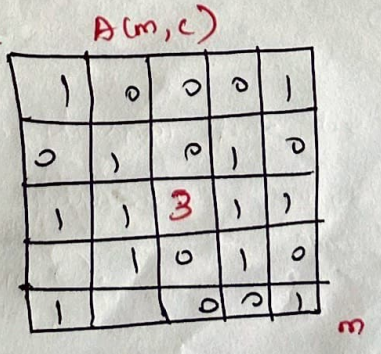
Initial



Point 1



Point 2

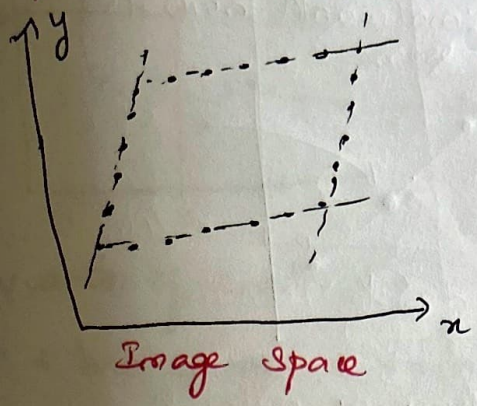


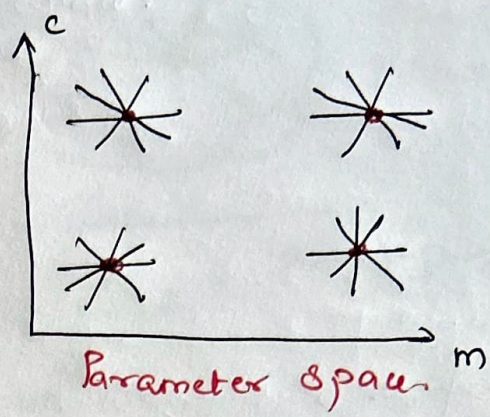
Point 3.

* The point of Intersection, which corresponds to the line that passes through all of these 3 points. gets the max vote of three.

* The mc values at those maxima correspond to the lines in the image.

Multiple Line Detection:-





* The 4 intersection points in the parameter space corresponds to the four lines that pass through these points in image space.

Better parameterization:-

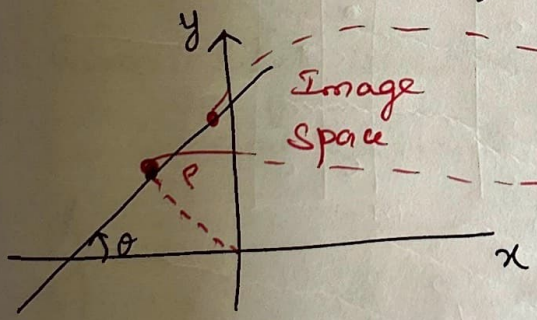
Issue: Slope of the line $-\infty \leq m \leq \infty$.

- Large Accumulator.
- More memory and computation.

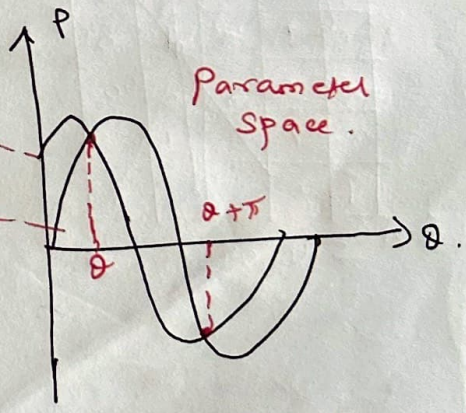
Solution: use $x\sin\theta - y\cos\theta + P = 0$.

Orientation θ is finite: $0 \leq \theta < \pi$

Distance P is finite.



$x\sin\theta - y\cos\theta + P = 0$

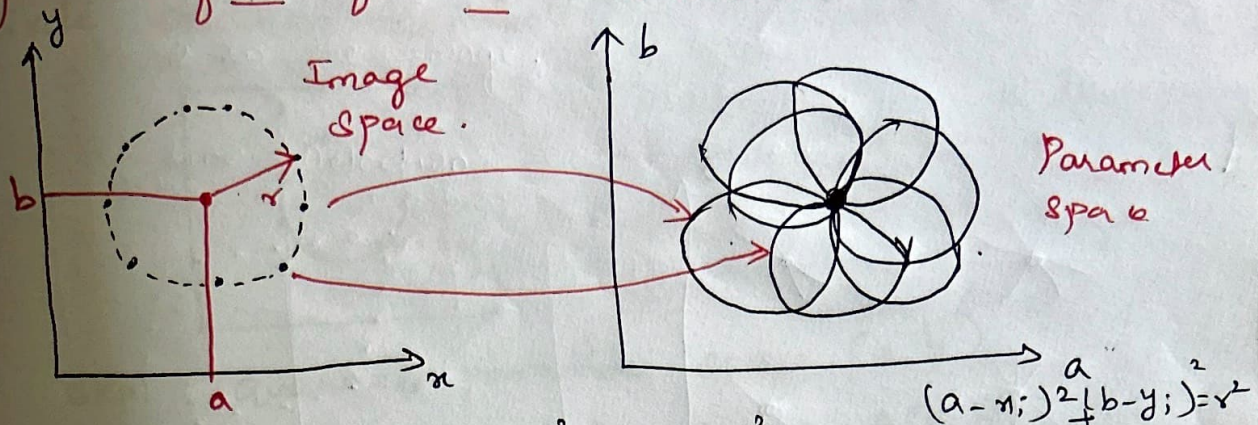


$x\sin\theta - y\cos\theta + P = 0.$

$\theta \rightarrow$ Angle between st. line and the horizontal axis

$P \rightarrow$ distance of the line from the origin.

Hough Transform for circle Detection:-



Equation of circle: $(x_i-a)^2+(y_i-b)^2=r^2$

a & b → corresponds to the center

r → radius of the circle.

* A point in an Image space → corresponds to a circle in the parameter space.

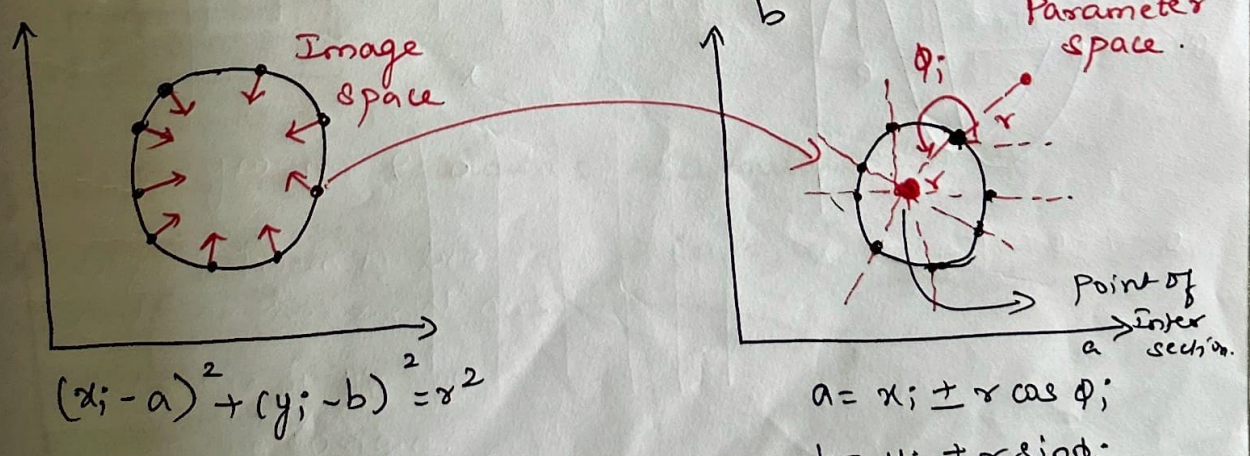
* If radius r is known: Accumulator Array is A(a,b).

* The array has parameters a and b.

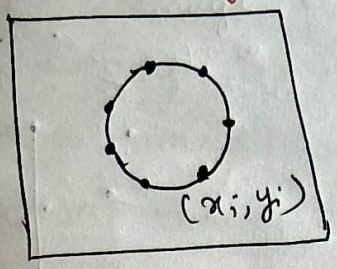
* All circles in the parameter space intersects at one point. This point corresponds to the a & b values.

Gradient Information:-

Given: Edge location (x_i,y_i), Edge Direction ϕ_i and Radius r.



Circle Detection Algorithm:-



Step1 : Quantize parameter space (a, b)

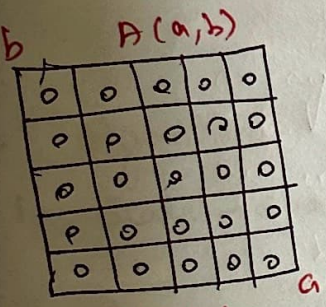
Step2: Create accumulator Array A(a, b)

Step3: Set A(a,b) = 0 for all (a, b).

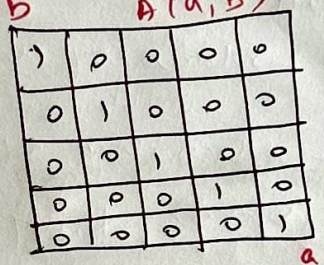
Step4: For each edge point (xi, yi), update accumulator array by 2,

$A(a,b) = A(a,b) + 2$.

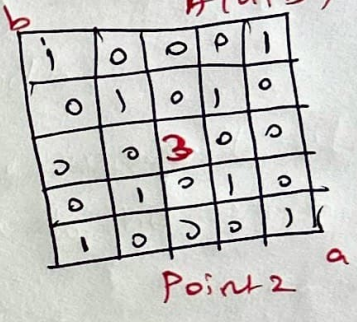
Step5: Find local maxima in A(a, b)



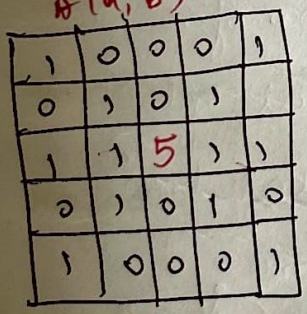
Initial



Point 1



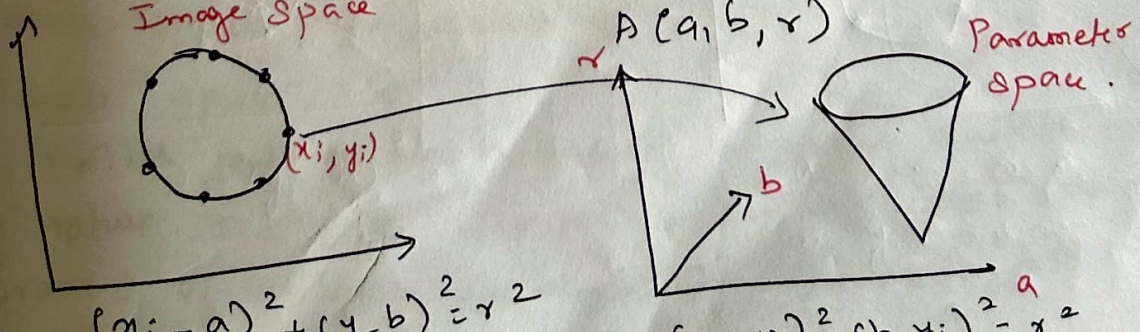
Point 2



Point 3

so on - - -

If Radius r is unknown: Accumulator array is

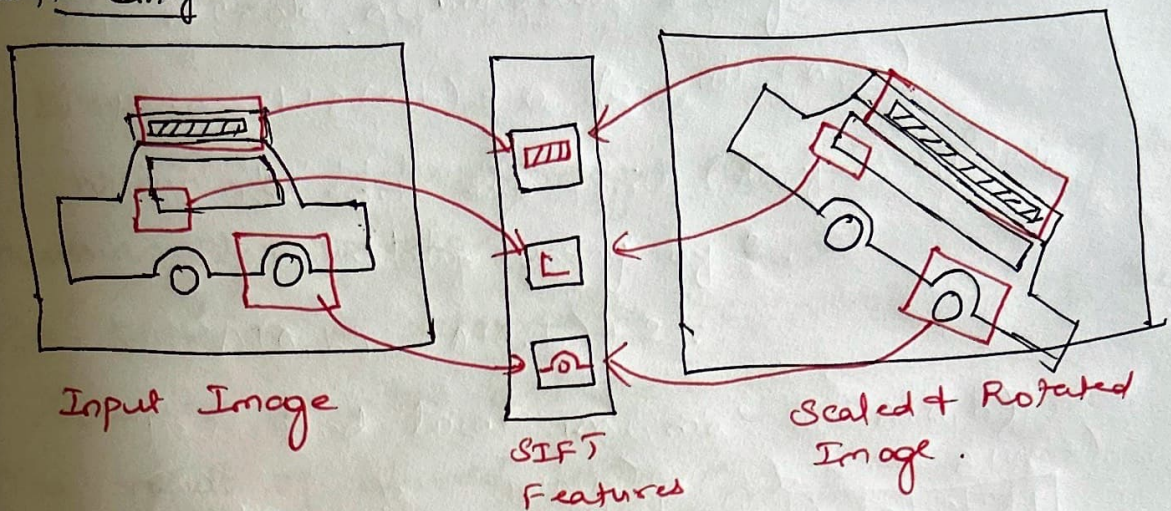


If no of parameters increases in acc. array → high words involved.

Scale - Invariant Feature Transform: (SIFT).

A SIFT detector is a computer vision algorithm used to identify unique "interest points" or key points in an image, even when the Image's scale or rotation changes. (ie) SIFT is invariance to Image scale and rotation.

Example Image:



SIFT Algorithm:

1. Scale - space peak selection → Potential location for finding features.
2. Key point Localization → Accurately locating the feature key points.
3. Orientation Assignment → Assigning orientation to Keypoints.
4. Keypoint Descriptor → Describing the keypoints as a high dimensional Vector.
5. Keypoint matching → Keypoints b/w two images are matched.

1. Scale - space peak selection:

i) Scale - space:

The multi-scale nature of objects is quite common in nature. The SIFT detector builds a scale-space pyramid of the image by repeatedly blurring it with increasing amounts of Gaussian blur.

* The scale-space of an image is a function $L(x, y, \sigma)$ that is produced from the convolution of a Gaussian Kernel (Blurring) at diff. scales with the input image.

* Scale-space is separated at into octaves and the no of octaves and scale depends on the size of the Original image.

ii) Blurring:-

* Within an octave, images are blurred using the Gaussian Blur operator.

* Blurring is referred to the convolution of the Gaussian operator and the image.

The result of blurred Image is,

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \rightarrow (1)$$

G → Gaussian blur operator

I → original Image

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

σ → scale parameter.

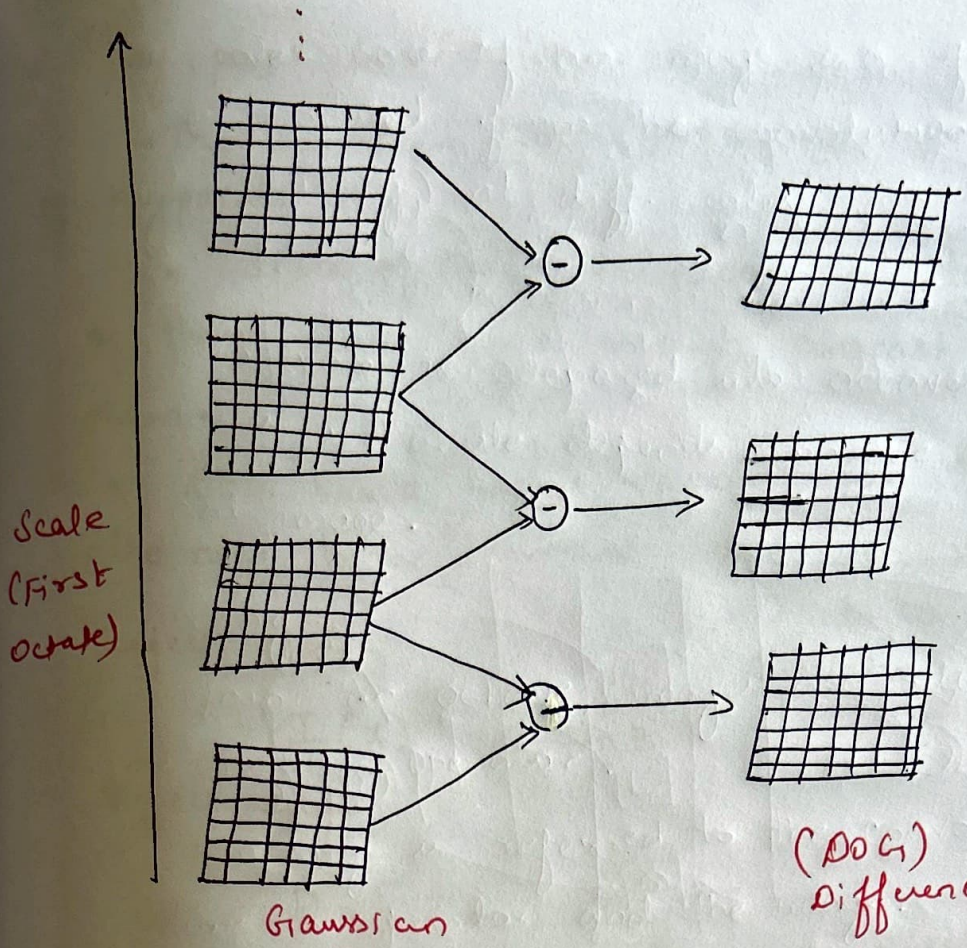
iii) Difference of Gaussian Kernel (DOG).

* Now we use those blurred Images to generate another set of Images called DOG.

* The DOG images are great for finding out interesting keypoints in the image.

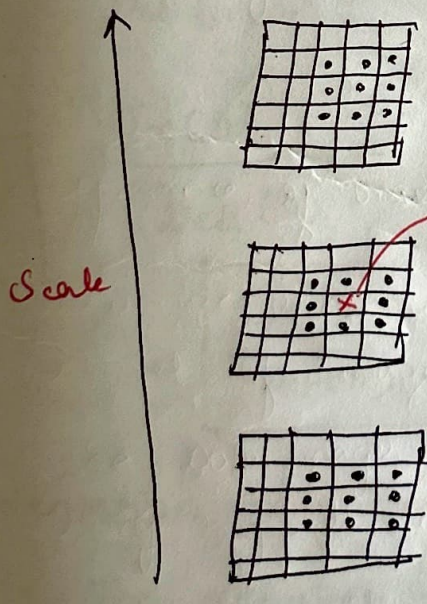
* DOG => Difference of Gaussian blurring of an image with 2 diff σ i.e) σ and $k\sigma$.

$$D(x, y, \sigma) = L(x, y, \sigma) - L(x, y, k\sigma). \rightarrow (3)$$



iv) Finding keypoints:

- So far, we have generated a scale space and used the scale space to calculate DOG. These are then used to calculate Laplacian of Gaussian (LoG) that are scale invariant.



* One pixel in an image is compared with its 8 neighbours as well as 9 pixels in the next scale and 9 pixels in the previous scale.
* If it is a local maxima then it is a potential keypoint.

2. Key point localization:-

- The previous step procedure produces a lot of keypoints.
- Some of them lie along an edge or some of them donot have enough contrast, so remove them.
- DoG has a higher response for edges, so edges also need to be Removed. Use 2×2 Hessian matrix (H) to compute Principal curvature.

Reject flats:-

$$|I(x)| < 0.03$$

Reject Edges:-

$$H = \begin{bmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{bmatrix} \rightarrow \text{Hessian Matrix.}$$

Let $\alpha \rightarrow$ Eigenvalue with larger magnitude.

$\beta \rightarrow$ " " smaller "

$$Tr(H) = I_{xx} + I_{yy} = \alpha + \beta$$

$$Det(H) = I_{xx} I_{yy} - (I_{xy})^2 = \alpha\beta$$

$$\frac{Tr(H)^2}{Det(H)} = \frac{(\alpha+\beta)^2}{\alpha\beta} = \frac{(r\beta+\beta)^2}{r\beta^2}$$

Let $r = \alpha/\beta$, $\alpha = r\beta$

$$\boxed{\frac{Tr(H)^2}{Det(H)} = \frac{(r+1)^2}{r}}$$

$\Rightarrow$ Reject edge if $r < 10$.

3. Orientation Assignment:-

- The next step is to assign an orientation to each keypoint to make it rotation invariance.

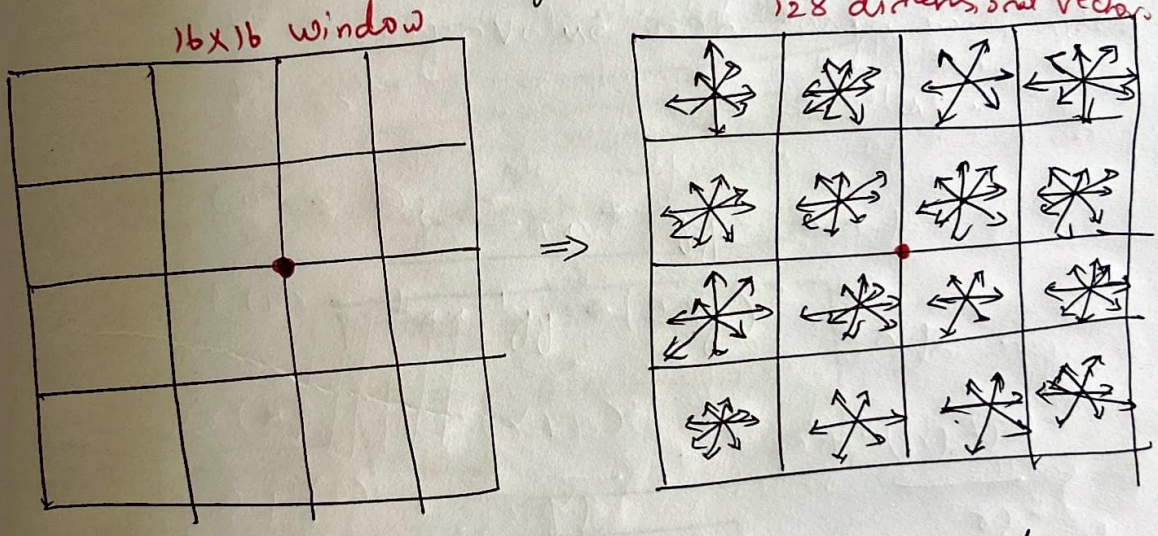
* For each keypoint, a local image patch's gradient orientations are computed and a histogram of these orientations is created.

* The highest peak in the histogram is taken and any peak above 80% of it is also considered to calculate orientation.

4. Keypoint Descriptor:-

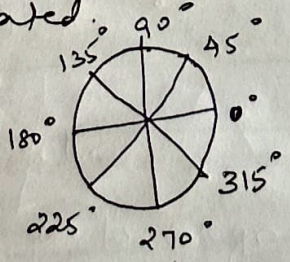
* Each keypoint has location, scale and orientation.

* Next is to compute a descriptor for the local image region about each keypoint that is highly distinctive and invariant as possible to variations such as changes in viewpoint & illumination



• → key point

For each subblock, 8 bin oriented histogram is created.



5. Keypoint Matching:

Keypoints between two images are matched by identifying their nearest neighbour.