Fundamentals of Computer Vision

LINEAR ALJEBRA

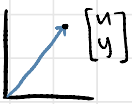
Vector

↳ column vector → $v \in R^{n \times 1}$ → $v = \begin{bmatrix} v_1 \\ v_2 \\ v_n \end{bmatrix}$

↳ row vector → v^T (transpose) $\in R^{1 \times n}$ → $v^T = [v_1 \; v_2 \; \dots \; v_n]$

Vector uses

↳ can represent offset in 2D/3D space



↳ data can be treated as a vector

↳ these don't have geometric interpretations

Matrix

↳ an array of numbers → $A \in R^{m \times n}$

↳ size is m (rows) by n (columns)

↳ if $m=n$, then the matrix is a square

Flattening it

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ a_{31} & a_{32} & \dots & a_{3n} \end{bmatrix} \rightarrow [a_{11}, a_{12}, a_{1n}, a_{21}, a_{22}, a_{2n}, a_{31}, a_{32}, a_{3n}]$$

Images

↳ in Python

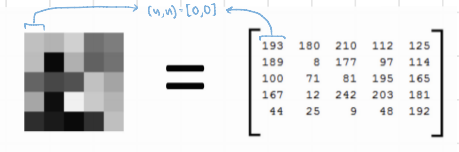
an image → matrix of pixel brightness

Color images

↳ Grayscale images

↳ 1 number per pixel

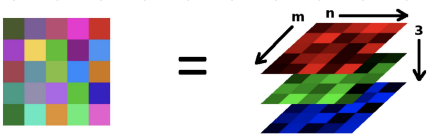
↳ stored as an $m \times n$ matrix



↳ color images

↳ 3 numbers per pixel → Red, Green, Blue

↳ stored as $m \times n \times 3$ matrix



VECTOR OPERATIONS

↳ **Norm** → $\|n\|_2 = \sqrt{\sum_{i=1}^{n} n_i^2}$

↳ any function that satisfies 4 properties $f: \mathbb{R}^n \to \mathbb{R}$

↳ Non Negatives → for all $n \in \mathbb{R}^n$, $f(n) \geq 0$

↳ Definiteness → $f(n) = 0$ only if $n = 0$

↳ Homogenity → for all $n \mathbb{R}^n$, $t \in \mathbb{R}$, $f(tn) = |t| f(n)$

↳ Triangle inequality → for all $n, y \in \mathbb{R}^n$, $f(n+y) \leq f(n) + f(y)$

- Example norms

$$\|x\|_1 = \sum_{i=1}^{n} |x_i| \qquad \|x\|_\infty = \max_i |x_i|$$

- General ℓ_p norms:

$$\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}$$

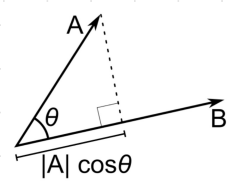
↳ **Dot Product** → aka Inner Product

$$\begin{bmatrix} a & b & c \end{bmatrix} \begin{bmatrix} d \\ e \\ f \end{bmatrix} = \begin{bmatrix} ad + be + cf \end{bmatrix}$$

↳ if B is a unit vector

↳ me A.B = length of A

↳ which lies in the direction of B



SPECIAL MATRICES

↳ **Identity Matrix I**

$$I_{3\times3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

↳ **Diagonal matrix**

$$\begin{bmatrix} 3 & 0 & 0 \\ 0 & 7 & 0 \\ 0 & 0 & 25 \end{bmatrix}$$

↳ **Symmetric matrix**

$A^T = A$

$$\begin{bmatrix} 1 & 2 & 5 \\ 2 & 1 & 7 \\ 5 & 7 & 1 \end{bmatrix}$$

↳ **Skew symmetric matrix**

$$\begin{bmatrix} 1 & -2 & -5 \\ 2 & 1 & -7 \\ 5 & 7 & 1 \end{bmatrix}$$

MATRIX OPERATIONS

Addition

- can only add a matrix with
  - matching dimensions

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} a+1 & b+2 \\ c+3 & d+4 \end{bmatrix}$$

- a scalar

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} + 7 = \begin{bmatrix} a+7 & b+7 \\ c+7 & d+7 \end{bmatrix}$$

Scaling

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times 3 = \begin{bmatrix} 3a & 3b \\ 3c & 3d \end{bmatrix}$$

Powers

- $A \cdot A = A^2$
- $A \cdot A \cdot A = A^3$

Product

→ aka Matrix Multiplication

$A \times B$ → order matters

$$\begin{bmatrix} 0 & 2 \\ 4 & 6 \end{bmatrix} \times \begin{bmatrix} 1 & 3 \\ 5 & 7 \end{bmatrix} = \begin{bmatrix} 0\times1+2\times5 & 0\times3+2\times7 \\ 4\times1+6\times5 & 4\times3+6\times7 \end{bmatrix}$$

- The product of 2 matrices is
  - Associative $(AB)C = A(BC)$
  - Distributive $A(B+C) = AB + AC$
  - Not Communitative $AB \neq BA$

Transpose

- flipped matrix
  - so row 1 becomes column 1
  - $(ABC)^T = C^T B^T A^T$ → useful identity

$$\begin{bmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \end{bmatrix}^T \cdot \begin{bmatrix} 0 & 2 & 4 \\ 1 & 3 & 5 \end{bmatrix}$$

Determinant

- $\det(A)$ returns a scalar
- represents area

$$\det\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right) = ad - bc$$

- $\det(AB) = \det(A) \cdot \det(B)$
- $\det(AB) = \det(BA)$
- $\det(A^{-1}) = 1/\det(A)$
- $\det(A^T) = \det(A)$
- $\det(A) = 0$ → A is singular

Trace

- trace (A) = sum of diagonal

$$tr\left(\begin{bmatrix} 1 & 3 \\ 5 & 7 \end{bmatrix}\right) = 1 + 7 = 8$$

- $tr(AB) = tr(BA)$
- $tr(A + B) = tr(A) + tr(B)$

TRANSFORMATION

done using multiplication

$n' = An$

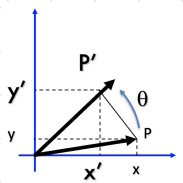
Scaling

$$\begin{matrix} S & \times & n & = & n' \end{matrix}$$

$$\begin{bmatrix} S_n & 0 \\ 0 & S_y \end{bmatrix} \times \begin{bmatrix} n \\ y \end{bmatrix} = \begin{bmatrix} S_n n \\ S_y y \end{bmatrix}$$

Rotation (2D)

↳ counter clockwise rotation by an angle θ



$$\begin{matrix} P' & = & R & P \end{matrix}$$

$$\begin{bmatrix} n' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} n \\ y \end{bmatrix} = \begin{bmatrix} \cos\theta n - \sin\theta y \\ \cos\theta y + \sin\theta n \end{bmatrix}$$

↳ Multiple Transformations

$P' = R_2 R_1 S_P$

↳ right to left → $R_2(R_1(S_P))$

Order matters

$P'' = T \cdot S \cdot P$ (Translation, Scale, by matrix)

$$= \begin{bmatrix} 1 & 0 & t_n \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} S_n & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} n \\ y \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} S_n & 0 & t_n \\ 0 & S_y & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} n \\ y \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} S_n n + t_n \\ S_y y + t_y \\ 1 \end{bmatrix}$$

→ Different ≠ ←

$P''' = S \cdot T \cdot P$

$$= \begin{bmatrix} S_n & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & t_n \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} n \\ y \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} S_n & 0 & S_n t_n \\ 0 & S_y & S_y t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} n \\ y \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} S_n n + S_n t_n \\ S_y y + S_y t_y \\ 1 \end{bmatrix}$$

Generalized Matrix

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} u \\ y \end{bmatrix} = \begin{bmatrix} au+by \\ cu+dy \end{bmatrix}$$

we can
- ↳ rotate
- ↳ scale
- ↳ skew
- ↳ can not Translate

SOLUTION →

Homogenous Matrix

Translation

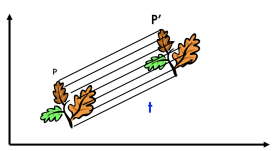
$$\begin{bmatrix} a & b & c \\ a & e & f \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} u \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} au+by+c \\ du+ey+f \\ 0+0+1 \end{bmatrix}$$

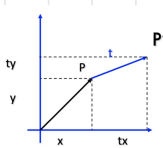
we can
- ↳ rotate
- ↳ scale
- ↳ skew
- ↳ Translate
- ↳ Divide → will divide the result by it's last coordinate after doing matrix multiplication

$$\begin{bmatrix} u \\ y \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} u/1 \\ y/1 \\ 1 \end{bmatrix}$$

2D Transformation using Homogenous Coordinates

↳ Translation





$$\mathbf{P} = (x, y) \rightarrow (x, y, 1)$$

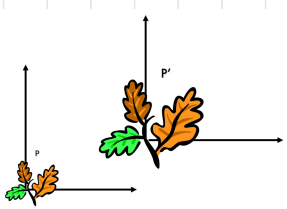
$$\mathbf{t} = (t_x, t_y) \rightarrow (t_x, t_y, 1)$$

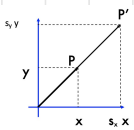
$$\mathbf{P'} \rightarrow \begin{bmatrix} x+t_x \\ y+t_y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{t} \\ 0 & 1 \end{bmatrix} \cdot \mathbf{P} = \mathbf{T} \cdot \mathbf{P}$$

(t → T, P → P)

$$T = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

↳ Scaling





$$\mathbf{P} = (x, y) \rightarrow \mathbf{P'} = (s_x x, s_y y)$$

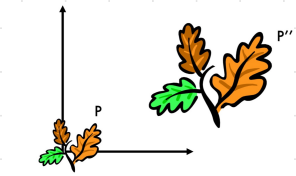
$$\mathbf{P} = (x, y) \rightarrow (x, y, 1)$$

$$\mathbf{P'} = (s_x x, s_y y) \rightarrow (s_x x, s_y y, 1)$$

$$\mathbf{P'} \rightarrow \begin{bmatrix} s_x x \\ s_y y \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{S}} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{S'} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \cdot \mathbf{P} = \mathbf{S} \cdot \mathbf{P}$$

$$S = \begin{bmatrix} Sn & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

↳ Scaling and Translating

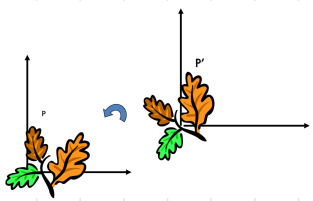


$P'' = T \cdot P' = T \cdot (S \cdot P) = T \cdot S \cdot P$

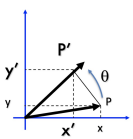
$$P'' = T \cdot S \cdot P = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} =$$

$$= \begin{bmatrix} s_x & 0 & t_x \\ 0 & s_y & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x x + t_x \\ s_y y + t_y \\ 1 \end{bmatrix} = \begin{bmatrix} S & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

↳ Rotation



Counter-clockwise rotation by an angle θ



$$x' = \cos\theta x - \sin\theta y$$
$$y' = \cos\theta y + \sin\theta x$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$P' = RP$$

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

↳ $R \cdot R^T = R^T \cdot R = 1$ → R^T produces rotation in the opposite direction

↳ $\det(R) = 1$

e.g. rotating $(1,0)$ by $90°$

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

↳ Scaling + Rotation + Translation

$$P' = (T\ R\ S)\ P$$

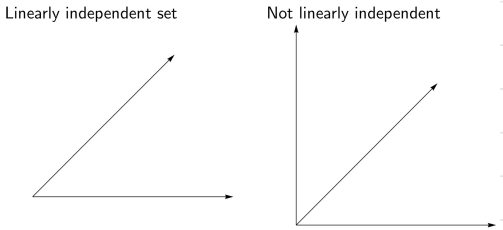
$$P' = T \cdot R \cdot S \cdot P = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} =$$

$$= \begin{bmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} =$$

$$= \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} S & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} RS & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Linear Independence

- Suppose we have a set of vectors $v_1, \ldots, v_n$
- If we can express v_1 as a linear combination of the other vectors $v_2, \ldots, v_n$, then v_1 is linearly *dependent* on the other vectors
  - The direction v_1 can be expressed as a combination of the directions $v_2, \ldots, v_n$ (e.g., $v_1 = 0.7v_2 - 0.7v_4$)
- If no vector is linearly dependent on the rest of the set, the set is linearly *independent*.
  - Common case: a set of vectors $v_1, \ldots, v_n$ is always linearly independent if each vector is perpendicular to every other vector (and non-zero).



Matrix Inverse

↳ A^{-1}

$$\begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}^{-1} = \begin{bmatrix} 1/2 & 0 \\ 0 & 1/3 \end{bmatrix}$$

↳ $A \cdot A^{-1} = I$ → if matrix has an inverse you can undo a transformation

↳ $(A^{-1})^{-1} = A$

↳ $(AB)^{-1} = B^{-1} \cdot A^{-1}$

↳ $A^{-T} = (A^{T})^{-1} = (A^{-1})^{T}$

Matrix Rank

↳ col rank (A) = max no. of linearly independent col vectors of A

↳ row rank (A) = max no. of linearly independent row vectors of A

↳ col rank = row rank

↳ for transformation matrices

it tells you dimmentions of output

eg if rank (A) = 1, then transformation $p' = Ap$

↳ matrix with rank 1

$$\begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x+y \\ 2x+2y \end{bmatrix}$$

↳ full rank → no information lost

↳ m x m matrix

↳ has inverse matrix

- Maps an $m \times 1$ vector uniquely to another $m \times 1$ vector

↳ singular

↳ rank < m

↳ inverse does not exist

- At least one dimension is getting collapsed. No way to look at the result and tell what the input was

↳ for non-square matrice

↳ inverse does not exist

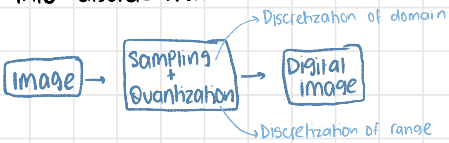
eg rank 1 matrix transformation collapses a 2D image into a line → useful for detecting edges

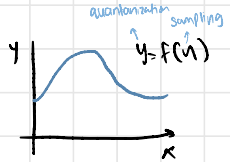
Image Filtering

→ to extract features

Digitalization

↳ process of transforming continous space into discrete space





Sampling

↳ Discretization of domain

↳ specifies resolution of image

$y = f(n)$

Quantization

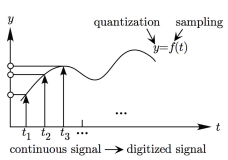
↳ Discretization of range

↳ specifies color space

$y = f(n)$

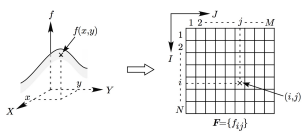
Digitization of

↳ 1D function



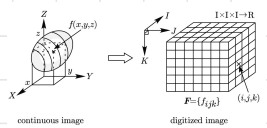
↳ 2D function

↳ discritization performed in 2 axis



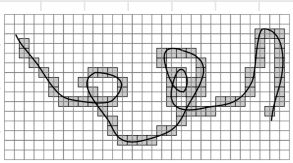
↳ 3D function

↳ discritization performed in 3 axis



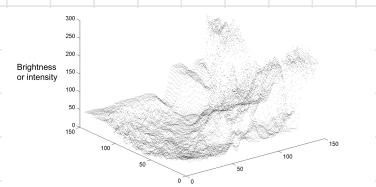
↳ an arc

↳ discritization performed in



↳ Gray scale digital image

↳ discritization performed in 2 axis and intensity

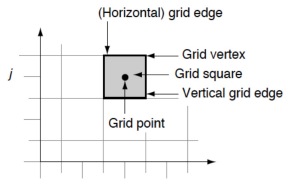


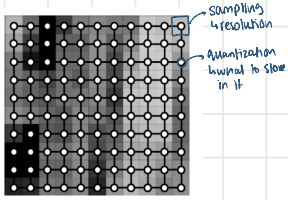
Definition

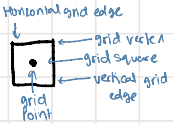
↳ an image P is a function defined on a finite rectangular subset G of a regular planar orthogonal array ← 2D Grid

an element of G is called a pixel

↳ P assigns a value of P(p) to each $p \in G$



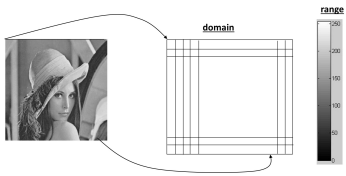




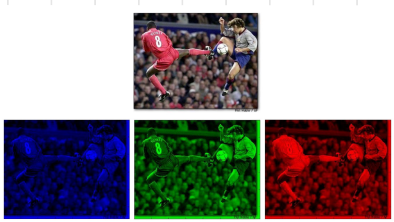
↳ pictures are sampled and quantized

- they may have only a finite number of possible values
- i.e., 0 to 255, 0-1, ...

Digitization



RGB channels



↳ 2 steps

1. Sampling

↳ discretize width and height of image

↳ domain

2. Quantization

↳ what values to store in each domain

Original (256 colors) | 8 colors | 4 colors

Resolution

↳ a display parameter

↳ defined by dots per inch (DPI)

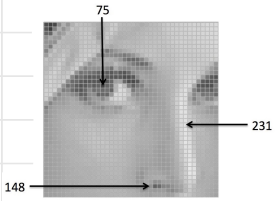
72 dpi → standard value for devices

Gray scale image

↳ the image contains discrete no. of Pixels

↳ Pixel value: grayscale

↳ [0, 255]



Color image

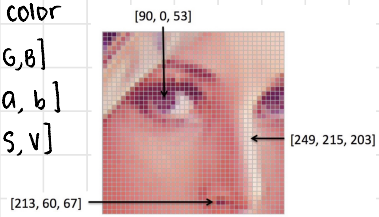
↳ the image contains discrete no. of Pixels

↳ Pixel value: color

↳ RGB: [R, G, B]

↳ Lab: [L, a, b]

↳ HSV: [H, S, V]



Other images

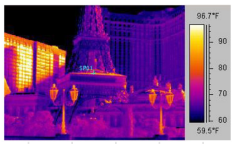
↳ Infrared image

↳ Ultrasound image

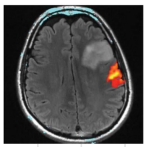
↳ CT scan image

Image – other examples

→ ultrasound image



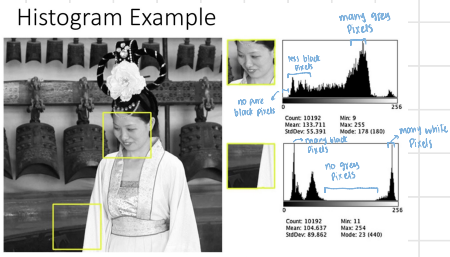
infrared image

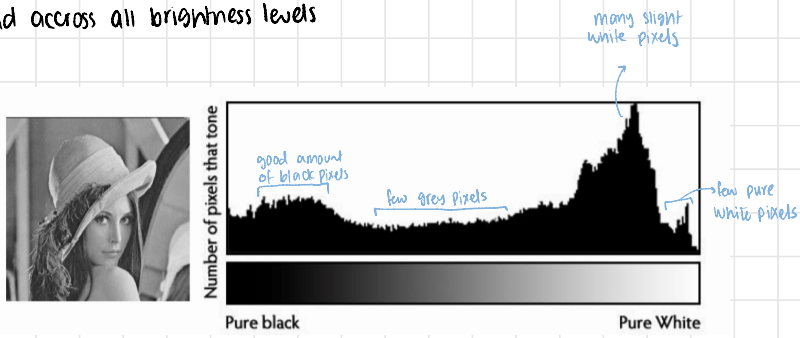


→ CT scan image

Image Histogram

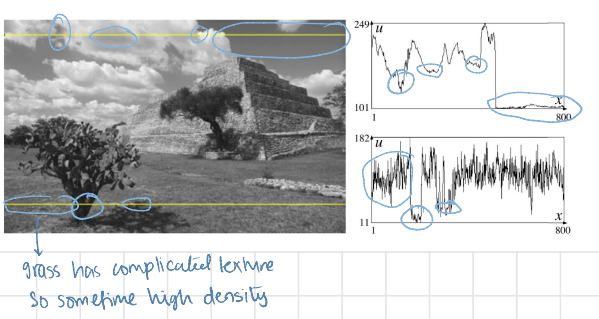
- Contrast
  - Bright images have histograms to the right
  - Dark images have histograms to the left
- Peak Distributions
  - High contrast images have seperated peaks
  - low contrast images have peaks closer
- Identify Uniformity
  - a well exposed image will have a balanced histogram spread across all brightness levels





intensity profiles for selected 2 rows

- dark → down
- bright → up



NOISE

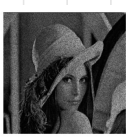
image noise
- Light variations
- Camera Electronics
- Surface Reflectance
- Lens → have something in your lens

noise
- It is random
- It occurs with some probability
- has a distribution

TYPES OF NOISE

1. Additive noise
- noise is added to original Pixel intensity independently

$I_{new} = I_{og} \quad N$ (image ← I_{new}, I_{og}; N → noise)

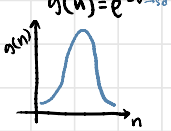
Solution: Gaussian Blur

2. Multiplicative noise
- gives grainy texture
- noise scales with Pixel intensity

$I_{new} = I_{og} \times N$ (image ← I_{new}, I_{og}; N → noise)


3. Gaussian noise

$g(n) = e^{\frac{-n^2}{2\sigma^2}}$



mean = 0 (centered at true value)
variance - controls intensity spread

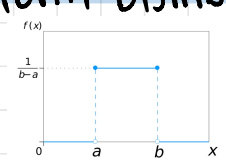
Solution: Gaussian Filter

4. Salt and Pepper noise
- random black and white dots
- each Pixel is randomly made black or white with a uniform probability distribution

Salt-pepper


Solution: Median Filter

uniform Distribution



FILTERING → Fixes noise

Image Filtering

↳ compute function of local neighbourhood at each position

↳ It is used to

↳ enhance images

↳ denoise, resize, increase contrast, etc

↳ extract information from images

↳ texture, edges, distinctive points, etc

↳ Detect patterns

↳ template matching

Output: F (filter) × I (intensity)

Filtering

↳ modify pixels based on function of neighborhood

10	30	10
20	11	20
11	9	1

$f(p)$ →

	5.7	

↳ Output → Linear combination of neighbourhood pixels

1	3	0
2	10	2
4	1	1

Image

⊗

1	0	-1
1	0.1	-1
1	0	-1

Kernel

↳ aka Filter

=

1		
	5	

Filter Output

$1\times1 + 3\times0 + 0\times-1$

$2\times1 + 10\times0.1 + 2\times(-1)$

$4\times1 + 1\times0 + 1\times(-1) = 5$

$$I = \begin{bmatrix} 1 & 3 & 0 \\ 2 & 10 & 2 \\ 4 & 1 & 1 \end{bmatrix} \quad K = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0.1 & -1 \\ 1 & 0 & -1 \end{bmatrix} \quad I \otimes K = \begin{bmatrix} 5 \end{bmatrix}$$

convolution

True filtering operation

A similar alternate

correlation

Linear relationship

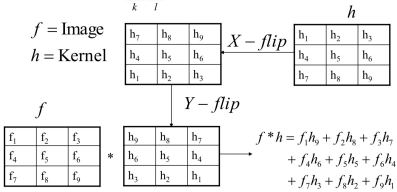
↳ The filter is flipped before applying

↳ is not flipped

$$f * h = \sum_k \sum_l f(k,l)h(-k,-l)$$

f = Image

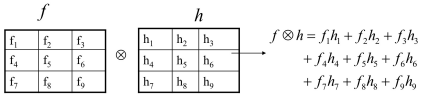
h = Kernel



$$f \otimes h = \sum_k \sum_l f(k,l)h(k,l)$$

f = Image

h = Kernel



↳ It is a filtering operation

↳ expresses the amount of overlap of one function as it is shifted over another function

↳ is associative

F.(G.I) = (F.G).I

↳ used in

↳ edge detection

↳ sharpening

↳ smoothing

e.g $h = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$ this filter highlights edges by subtracting local averages

↳ compares similarity of 2 sets of data

↳ relatedness of the signals

↳ used in template matching

pros

↳ Associative

↳ multiple filters can be combined into one

↳ used in CNN as efficient

Averages

↳ Mean

$$I = \frac{I_1 + I_2 + \dots I_n}{n} = \frac{\sum_{i=1}^{n} I_i}{n}$$

↳ weighted mean

$$I = \frac{w_1 I_1 + w_2 I_2 + \dots w_n I_n}{n} = \frac{\sum_{i=1}^{n} w_i I_i}{n}$$

FILTERING EXAMPLES

A. Smoothing Filters

1. Gaussian Filter
2. Mean (Box) Filter

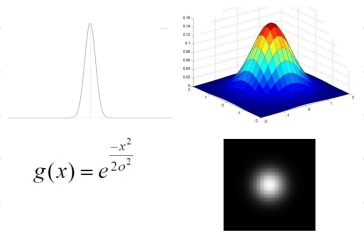
1. Gaussian Filter → KING OF SMOOTHING

↳ Provides natural blurring

PRO ↳ Preserves edge structure → better than mean filter

1D

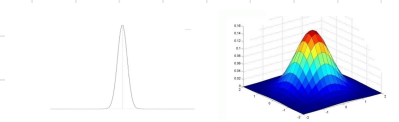
$$g(u) = e^{\frac{-u^2}{2\sigma^2}}$$



$$g(x) = e^{\frac{-x^2}{2\sigma^2}}$$

always symmetric

2D



$$g(x) = e^{\frac{-x^2}{2\sigma^2}} \qquad g(x,y) = e^{\frac{-(x^2+y^2)}{2\sigma^2}}$$

$$g(u) = e^{\frac{-(u^2+v^2)}{2\sigma^2}}$$

$$g(x) = [.011 \quad .13 \quad .6 \quad 1 \quad .6 \quad .13 \quad .011]$$

.6 · .6

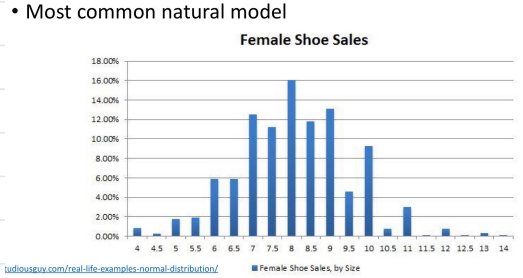
.13 · .13

.011

Gaussian Filter Properties

1. Most common natural model
2. Smooth function
   ↳ It has infinite no. of derivatives
3. It is symmetric
4. Fourier Transform of Gaussian is Gaussian
5. Convolution of a Gaussian with itself is Gaussian
6. Gaussian is seperable
   ↳ 2D convolution can be performed by two 1D convolutions
7. There are cells in the eye that perform Gaussian Filtering

• Most common natural model



Singular value decomposition (SVD) → for compression

$$A = U \Sigma V^T$$

(m×n) (m×n) (n×n)

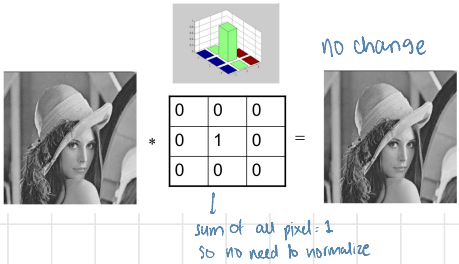
U and V are Unitary matrices

top 20 imp

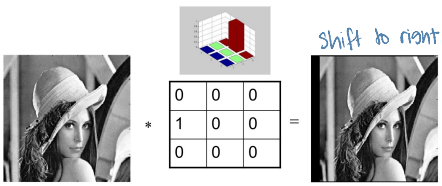
neechay walay values useless

Sum of all Pixel should be 1

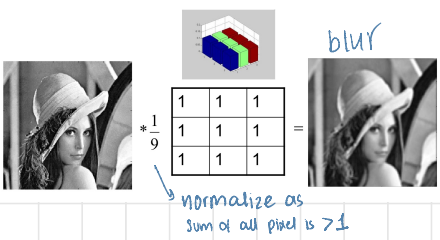
Filtering Examples - 1



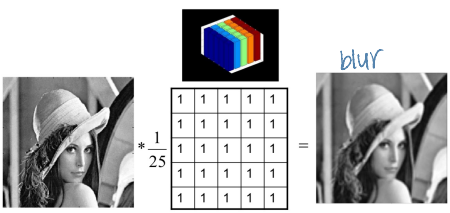
Filtering Examples - 2



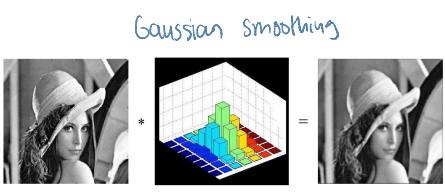
Filtering Examples - 3



Filtering Examples - 4



Filtering Examples - 5

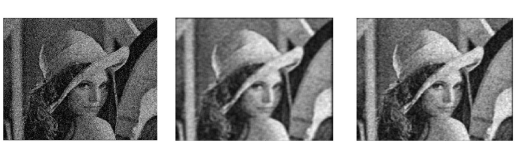


Gaussian Smoothing

Filtering Examples - 6

Gaussian Smoothing | Smoothing by Averaging

Filtering Examples - 7



After additive Gaussian Noise | After Averaging | After Gaussian Smoothing

2. Box filter

↳ replaces each pixel with an avg of it's neighborhood

↳ It achives smoothness effect

↳ removes sharp features

low ↳ reduces noise but blurs edges

$g[\cdot,\cdot]$

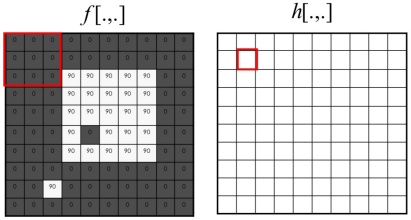
$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

normalize

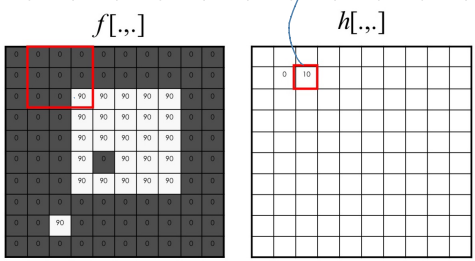
Image filtering

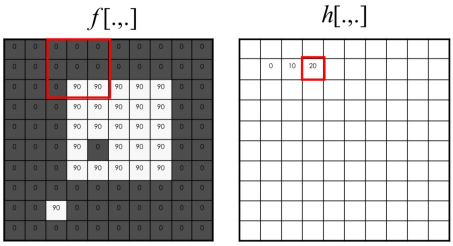
$g[\cdot,\cdot]$ $\frac{1}{9}$ [1 1 1; 1 1 1; 1 1 1]

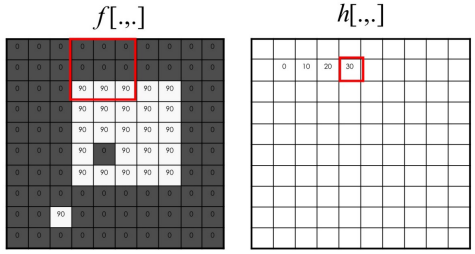


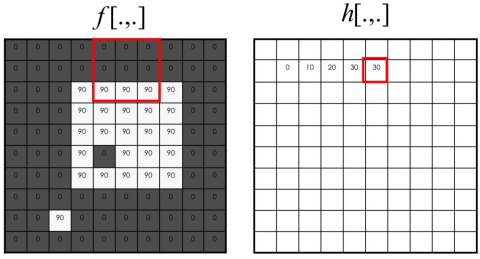
$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

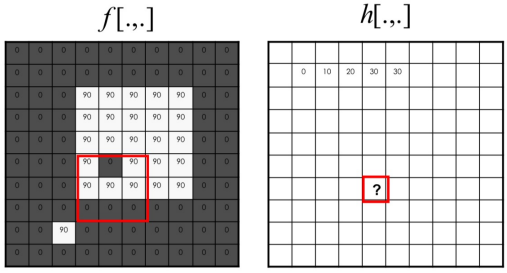
Credit: S

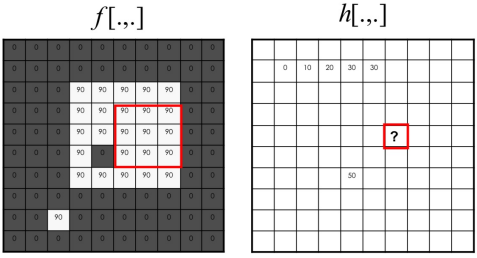


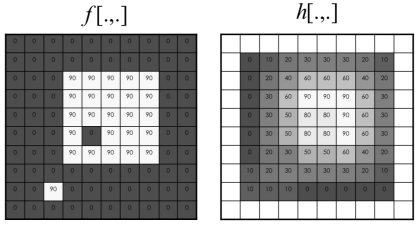












Smoothing with Box filter

Practice with kernels

Original

0	0	0
0	1	0
0	0	0

?

Practice with kernels

Original

0	0	0
0	1	0
0	0	0

Filtered
(no change)

Practice with kernels

Original

0	0	0
0	0	1
0	0	0

?

Practice with kernels

Original

0	0	0
0	0	1
0	0	0

Shifted left
By 1 pixel

Practice with kernels

using 2 filters

Original

increase intensity

0	0	0
0	2	0
0	0	0

$-\frac{1}{9}$

smooth image

1	1	1
1	1	1
1	1	1

(Note that filter sums to 1)

?

Practice with kernels

Original

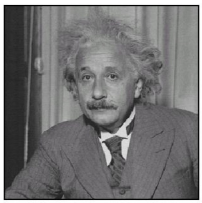
0	0	0
0	2	0
0	0	0

$-\frac{1}{9}$

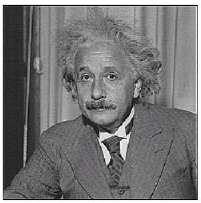
1	1	1
1	1	1
1	1	1

Sharpening filter
- Accentuates differences with local average

B. Sharpening Filters



before



after

A. Laplacian Filter: Second-Order Edge Detection

$$h = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

- Highlights areas of rapid intensity change.
- Used in **medical imaging and document enhancement**.

B. Unsharp Masking

$$I_{\text{sharpened}} = I + \lambda(I - I_{\text{blurred}})$$

- Enhances edges without amplifying noise.

Example: Used in **photo editing software**.

C. Edge Detection Filters

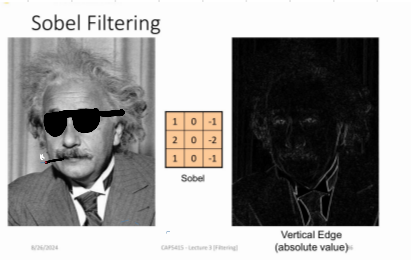
↳ Sobel Filtering

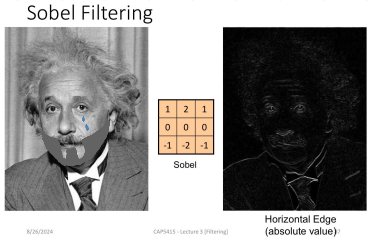
Sobel Filtering

↳ detects Gradients

↳ detects horizontal and vertical edges

↳ used in
- ↳ Self driving cars
- ↳ object detection





Linear filter Properties

1. Linearity: filter($f_1 + f_2$) = filter(f_1) + filter(f_2)
2. Shift invariance: filter(shift(f)) = shift(filter(f))
   ↳ same behaviour regardless of pixel location
3. More properties:
   - Commutative: $a * b = b * a$
     - Conceptually no difference between filter and signal
     - particular filtering implementations might break this equality
   - Associative: $a * (b * c) = (a * b) * c$
     - Often apply several filters one after another: $(((a * b_1) * b_2) * b_3)$
     - This is equivalent to applying one filter: $a * (b_1 * b_2 * b_3)$
   - Distributes over addition: $a * (b + c) = (a * b) + (a * c)$
   - Scalars factor out: $ka * b = a * kb = k(a * b)$
   - Identity: unit impulse $e = [0, 0, 1, 0, 0]$, $a * e = a$

Any linear, shift-invariant operator can be represented as a convolution

D. Non Linear Filters → Handling Impulse Noise

↳ Median Filter

Median Filter → Saviour for Salt and Pepper noise

↳ replaces each pixel with median of it's neighbourhood

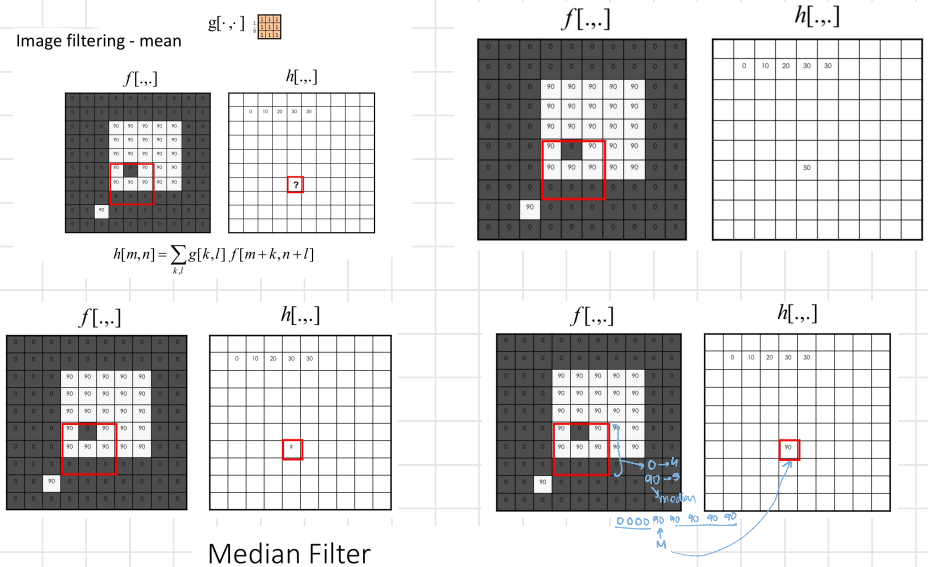
↳ operates over a window by selecting the median intensity in the window

↳ it is NOT the same as convolution

PRO

↳ retains edges better than mean box filtering

↳ used in fingerprint enhancement



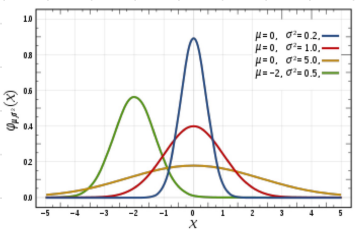
PRACTICAL MATTERS

How big should the filter be?

↳ values at edges should be near 0

↳ Gaussians have infinite extent...

rule of thumb for Gaussian ↳ set filter half width to about 3σ



mean

μ = center point

$\sigma^2\uparrow$: short curve

$\sigma^2\downarrow$: tall curve

sd

Computational Efficiency

↳ convolutions are slow for large images

↓Solution

↳ Fourier Transform

↳ converts convolution into multiplication

↳ used in MRI and astronomy

Filtering Conclusion

1. Noise Removal
   - ↳ Gaussian
   - ↳ Mean
   - ↳ Median
2. Sharpning
   - ↳ Laplacian
   - ↳ Unsharp Masking
3. Edge Detection
   - ↳ Sobel
   - ↳ Canny
4. Efficiency hrlfs
   - ↳ Fourier Transform

Boundary effects

What about near the edge?

↳ Pixels near the edges do not have full neighbours

↓

Solutions

↳ Padding (replicate, zero, reflection)

↳ Cropping (ignore edges)

↳ clip filter (black)

↳ wrap around

↳ copy edge

↳ reflect across edge

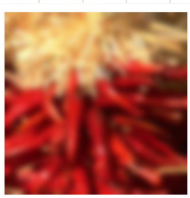


IMAGE DERIVATIVES

Derivative

↳ rate of change

- Speed is a rate of change of a distance, X=V.t
- Acceleration is a rate of change of speed, V=a.t

First order derivative

↳ diff b/w pixel intensity of neighbouring pixels

Discrete Derivative

$$\frac{df}{dx} = \lim_{\Delta x \to 0} \frac{f(x) - f(x - \Delta x)}{\Delta x} = f'(x)$$

$$\frac{df}{dx} = \frac{f(x) - f(x-1)}{1} = f'(x)$$

$$\frac{df}{dx} = f(x) - f(x-1) = f'(x)$$

Derivative

$$\frac{df}{dx} = \lim_{\Delta x \to 0} \frac{f(x) - f(x - \Delta x)}{\Delta x} = f'(x) = f_x$$

$$y = x^2 + x^4 \qquad \frac{dy}{dx} = 2x + 4x^3$$

$$y = \sin x + e^{-x} \qquad \frac{dy}{dx} = \cos x + (-1)e^{-x}$$

Backward diff

I at current pixel − I at previous pixel (intensity)

Forward diff

I at current pixel − I at next pixel

Central diff

I at next pixel − I at previous pixel

$$\frac{df}{dx} = f(x) - f(x-1) = f'(x) \quad \text{Backward difference}$$

$$\frac{df}{dx} = f(x) - f(x+1) = f'(x) \quad \text{Forward difference}$$

$$\frac{df}{dx} = f(x+1) - f(x-1) = f'(x) \quad \text{Central difference}$$

Example: Finite Difference

$f(x) = 10 \quad 15 \quad 10 \quad 10 \quad 25 \quad 20 \quad 20 \quad 20$

$f'(x) = 0 \quad 5 \quad -5 \quad 0 \quad 15 \quad -5 \quad 0 \quad 0$ → backward difference

$f''(x) = 0 \quad 5 \quad -10 \quad 5 \quad 15 \quad -20 \quad 5 \quad 0$

Derivative Masks → use masks instead of calculating all

Backward difference	[-1 1]
Forward difference	[1 -1]
Central difference	[-1 0 1]

2nd Order Derivative

Derivative in 2-D

Given function $f(x,y)$

Gradient vector $\nabla f(x,y) = \begin{bmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{bmatrix} = \begin{bmatrix} f_x \\ f_y \end{bmatrix}$

Gradient magnitude $|\nabla f(x,y)| = \sqrt{f_x^2 + f_y^2}$

Gradient direction $\theta = \tan^{-1}\frac{f_x}{f_y}$

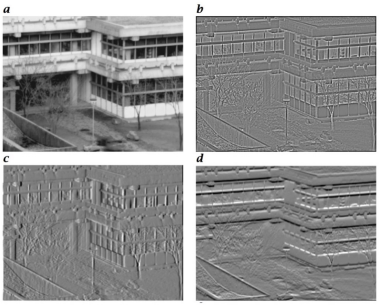
Derivative of Images

Derivative masks $f_x \Rightarrow \frac{1}{3}\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$ $f_y \Rightarrow \frac{1}{3}\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$

$$I = \begin{bmatrix} 10 & 10 & 20 & 20 & 20 \\ 10 & 10 & 20 & 20 & 20 \\ 10 & 10 & 20 & 20 & 20 \\ 10 & 10 & 20 & 20 & 20 \\ 10 & 10 & 20 & 20 & 20 \end{bmatrix} \quad I_x = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 10 & 10 & 0 & 0 \\ 0 & 10 & 10 & 0 & 0 \\ 0 & 10 & 10 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$I = \begin{bmatrix} 10 & 10 & 20 & 20 & 20 \\ 10 & 10 & 20 & 20 & 20 \\ 10 & 10 & 20 & 20 & 20 \\ 10 & 10 & 20 & 20 & 20 \\ 10 & 10 & 20 & 20 & 20 \end{bmatrix} \quad I_y = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

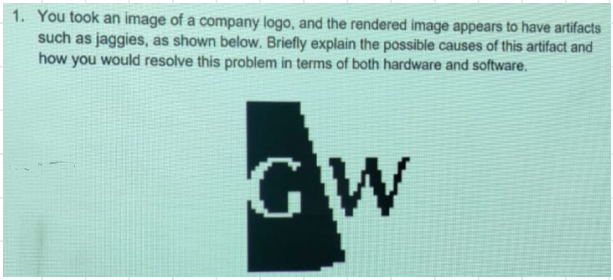
Example



a. Original image
b. Laplacian operator
c. Horizontal derivative
d. Vertical derivative

- **a**: Original image (input).
- **b**: Laplacian operator applied (edge detection).
- **c**: Horizontal derivative (detects vertical edges).
- **d**: Vertical derivative (detects horizontal edges).

1. You took an image of a company logo, and the rendered image appears to have artifacts such as jaggies, as shown below. Briefly explain the possible causes of this artifact and how you would resolve this problem in terms of both hardware and software.



Q1)
The artifacts in the image, such as jagged edges (aliasing), are likely caused by one or more of the following factors:

Possible Causes:

1. Low Resolution Rendering:
   - The image might have been rendered at a low resolution, leading to visible pixelation and jagged edges.
2. Poor Anti-Aliasing:
   - If the rendering system lacks proper anti-aliasing, edges will appear rough instead of smooth.
3. Incorrect Scaling:
   - Improper resizing of the image (e.g., nearest-neighbor scaling instead of bilinear or bicubic interpolation) can introduce artifacts.
4. Display or Hardware Issues:
   - If displayed on a low-quality screen with poor pixel density or color depth, the image may appear jagged.
5. Compression Artifacts:
   - If the image was heavily compressed, loss of detail could result in pixelation.

Solutions:

Hardware Solutions:

- Use a Higher Resolution Display:

A higher pixel density (DPI) display can help smooth out jagged edges.

- Upgrade Graphics Hardware:

A better GPU with proper anti-aliasing support can render smoother edges.

Software Solutions:

- Enable Anti-Aliasing:

Implement anti-aliasing techniques (e.g., MSAA, FXAA, or SSAA) to smooth edges.

- Use Vector Graphics:

If possible, use scalable vector graphics (SVG) instead of raster images to avoid pixelation.

- Render at a Higher Resolution:

Generating the image at a higher resolution and then downscaling with proper interpolation (e.g., bicubic or Lanczos resampling) can improve smoothness.

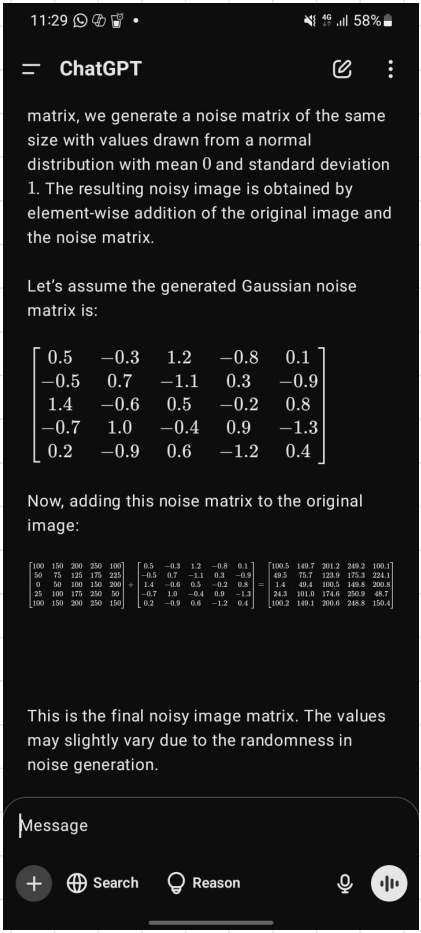
- Choose Proper Image Formats:

Use lossless formats (e.g., PNG, SVG) instead of lossy ones (e.g., JPEG) to avoid compression artifacts.

1. How would you add Gaussian noise to an image? Please use the 5x5 image shown below as an example and add Gaussian noise with a mean of 0 and a standard deviation (std) of 1.

```
image = np.array([[100, 150, 200, 250, 100],
                  [50, 75, 125, 175, 225],
                  [0, 50, 100, 150, 200],
                  [25, 100, 175, 250, 50],
                  [100, 150, 200, 250, 150]], dtype=np.float32)
```

2. What would you do to increase the impact of noise so that the final noisy image is highly distorted?



To increase the impact of noise and make the final image highly distorted, you can:

- Increase the standard deviation (std) of the Gaussian noise, making the noise values larger and more disruptive.
- Scale the noise by multiplying it with a higher factor before adding it to the image.
- Apply the noise multiple times iteratively to amplify its effect.
- Reduce the image's original intensity range, making the noise more dominant.

These methods will result in a more distorted and less recognizable image.

EDGE DETECTION

Edge Detection

↳ identify points in an image where intensity changes sharply

Edge Detectors

↳ Prewitt

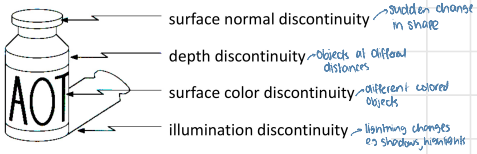
↳ Sobel

↳ Marr-Hildreth

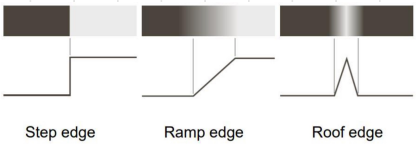
↳ Canny

Origion of Edges

↳ edges are caused by variety of factors



Types of Edges



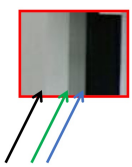
Why edge detection

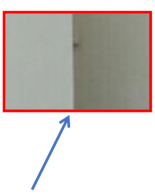
↳ to extract useful info from images

recognising objects

↳ Recover geometry

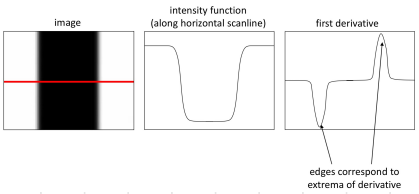
Closeup of edges





Characterizing Edges

↳an edge is a point of rapid intensity change in an image



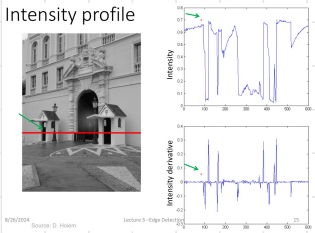
↳First derivative

↳measures the rate of intensity change

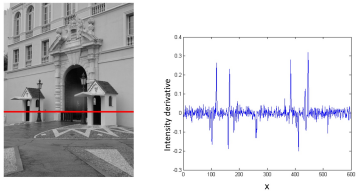
↳Second derivative

↳measures the rate of change of gradient

Intensity Profile



↳with a little Gaussian Noise



Effects of Noise in Edge Detection

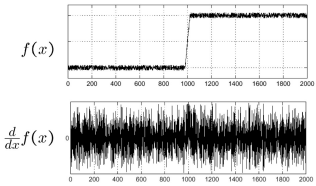
↳noise causes false edges

as high frequency noise appears as sudden intensity variations

↓ solution

apply smoothing before differentiation

- Plotting intensity as a function of position gives a signal



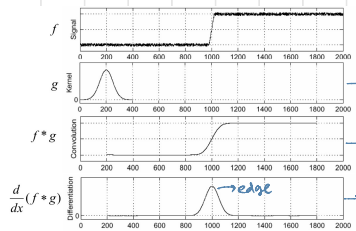
Where is the edge?

Conclusion:
Noise in the graphs would be visible as random, irregular fluctuations in the intensity plot, making it harder to identify the true edges in the image

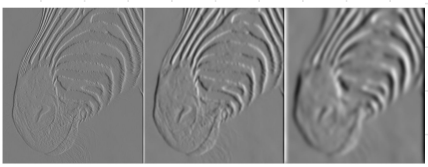
Smoothing

↳ removes noise

con ↳ but blurs edges



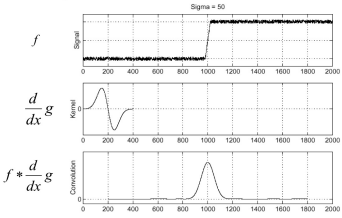
To find edges, look for peaks in $\frac{d}{dx}(f * g)$



1 pixel 3 pixels 7 pixels

Derivative theorem of Convolution

- Convolution is differentiable: $\frac{d}{dx}(f * g) = f * \frac{d}{dx} g$
- This saves us one operation:

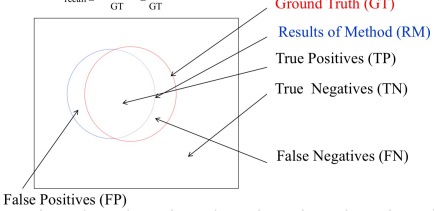


Sourc

Evaluate Edge Detection

$$\text{precision} = \frac{GT \cap RM}{RM} = \frac{TP}{RM}$$

$$\text{recall} = \frac{GT \cap RM}{GT} = \frac{TP}{GT}$$



Design Criteria for Edge Detection

↳ Good detection

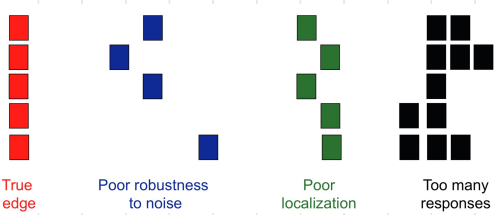
↳ find all real edges

↳ ignoring noise or other artifacts

↳ Good localization

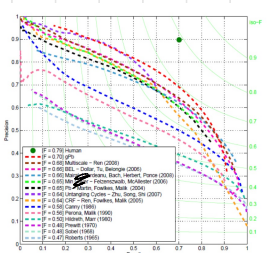
↳ as close as possible to the true edges

↳ one point only for each true edge point



45 years of Boundary Detection

[Pre deep learning]

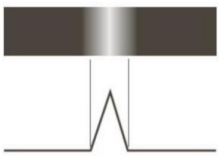


45 year boundary

1. Canny
2. Prewitt
3. Sob

Pop Quiz

- We have a signal with roof edge, with width of 5 pixels.
- Let's say we have a smoothing kernel
  - Size 11
  - Median filter instead of Gaussian filter
- With such filter smoothing we will not be able to detect any edges
  - True/False



Roof edge

Question:
We have a signal with a roof edge (width of 5 pixels). If we use a smoothing kernel of size 11 with a **median filter** instead of a Gaussian filter, will we **not be able to detect any edges**? (True/False)

Step-by-Step Reasoning:

1. **Roof Edge Definition**:
   - A **roof edge** is a gradual transition in intensity (not a sharp step edge). It has a width of 5 pixels in this case.
2. **Median Filter**:
   - A median filter is a **non-linear smoothing filter** that replaces each pixel's value with the median of the neighboring pixels (within the kernel size).
   - It is effective at removing **noise** while preserving edges, especially in cases of salt-and-pepper noise.
3. **Kernel Size (11)**:
   - The kernel size is **11 pixels**, which is significantly larger than the width of the roof edge (5 pixels).
4. **Effect of Median Filter on Roof Edge**:
   - Since the kernel size (11) is larger than the roof edge width (5), the median filter will **blur the roof edge**.
   - The gradual transition of the roof edge will be smoothed out, making it harder to detect.
5. **Conclusion**:
   - With a median filter of size 11, the roof edge will be **blurred to the point where it may become undetectable**.

Answer:

True – With such a large median filter, we will not be able to detect the roof edge.

Edge Detectors

1. Prewitt
2. Sobel
3. Marr-Hildreth
4. Canny

Prewitt and Sobel Edge Detector

- Compute Derivatives
  - in x and y direction
- Find gradient magnitude
- Threshold gradient magnitude

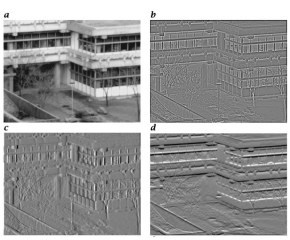
Discrete derivatives

		Masks
$\frac{df}{dx} = f(x) - f(x-1) = f'(x)$	Backward difference	[-1 1]
$\frac{df}{dx} = f(x) - f(x+1) = f'(x)$	Forward difference	[1 -1]
$\frac{df}{dx} = f(x+1) - f(x-1) = f'(x)$	Central difference	[-1 0 1]

Image Derivatives

Given function	$f(x, y)$
Gradient vector	$\nabla f(x, y) = \begin{bmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{bmatrix} = \begin{bmatrix} f_x \\ f_y \end{bmatrix}$
Gradient magnitude	$\lvert\nabla f(x, y)\rvert = \sqrt{f_x^2 + f_y^2}$
Gradient direction	$\theta = \tan^{-1} \frac{f_x}{f_y}$

Example



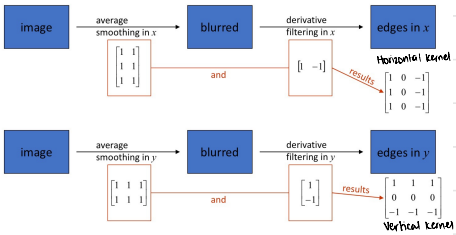
a. Original image
b. Laplacian operator
c. Horizontal derivative
d. Vertical derivative

- **a**: Original image (input).
- **b**: Laplacian operator applied (edge detection).
- **c**: Horizontal derivative (detects vertical edges).
- **d**: Vertical derivative (detects horizontal edges).

1. **a. Original Image**:
   - This is the starting image before any operations are applied.
   - It shows the raw input image with no modifications.
2. **b. Laplacian Operator**:
   - The Laplacian operator is used for **edge detection**.
   - It highlights regions of rapid intensity change in the image, which typically correspond to edges.
   - The result is an image where edges are emphasized, and smooth regions are suppressed.
3. **c. Horizontal Derivative**:
   - This represents the result of applying a **horizontal derivative filter** (e.g., Sobel or Prewitt filter in the horizontal direction).
   - It detects edges that are oriented **vertically** (i.e., horizontal intensity changes).
4. **d. Vertical Derivative**:
   - This represents the result of applying a **vertical derivative filter** (e.g., Sobel or Prewitt filter in the vertical direction).
   - It detects edges that are oriented **horizontally** (i.e., vertical intensity changes).

1. Prewitt Edge Detector

↳ calculates gradient of image intensity which highlights regions where there are sharp changes in intensity → edges



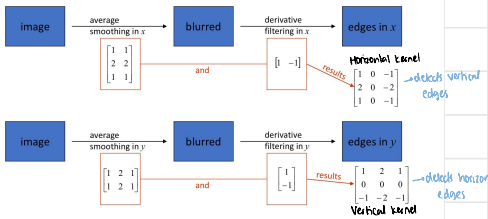
Steps

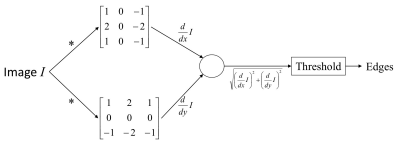
1. Apply kernels $\begin{bmatrix}1&0&-1\\1&0&-1\\1&0&-1\end{bmatrix}$ $\begin{bmatrix}1&1&1\\0&0&0\\-1&-1&-1\end{bmatrix}$
2. Calculate gradient magnitude (edge strength) at each pixel $\Delta = \sqrt{(\partial/\partial x\, I)^2 + (\partial/\partial y\, I)^2}$
   ↳ pixels with high gradient magnitudes are edges

CON
↳ sensitive to noise

2. Sobel Edge Detector

↳ similar to Prewitt
↳ adds higher weight to central pixel

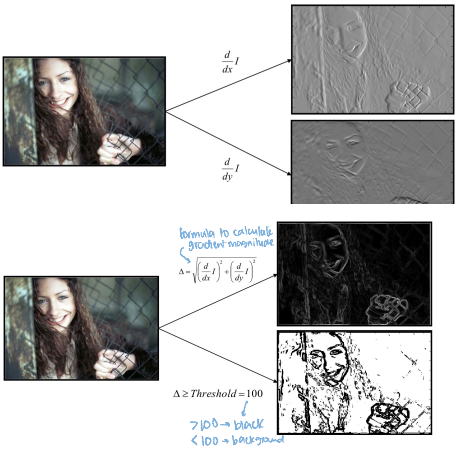




Steps

1. Apply kernels $\begin{bmatrix}1&0&-1\\2&0&-2\\1&0&-1\end{bmatrix}$ $\begin{bmatrix}1&2&1\\0&0&0\\-1&-2&-1\end{bmatrix}$
2. Calculate gradient magnitude (edge strength) at each pixel $\Delta = \sqrt{(\partial/\partial x\, I)^2 + (\partial/\partial y\, I)^2}$
   ↳ pixels with high gradient magnitudes are edges

PRO
↳ better noise supression
↳ more robust for edge detection



3. Marr Hildreth Edge Detector

→ aka Laplacian of Gaussian (LOG)

↳ combines Gaussian smoothing + Laplacian operator to detect edges in an image

Steps

1. smooth image by Gaussian Filter → image is smoothed to remove noise
2. Apply Laplacian → highlights regions of rapid intensity change → edges
3. Find zero crossings → detects edges by finding points where Laplacian output crosses 0
   - ↳ scan along each row
   - ↳ record an edge point at the location of 0 crossing
   - ↳ Repeat above step along each column

Pros

↳ detects both bright and dark edges

↳ can detect edges at different scales

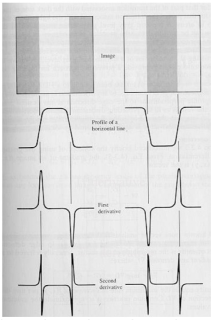
Con

↳ sensitive to noise, may detect false edges

Second Derivative

↳ maxima minima of first derivative

↳ zero-crossings of 2^{nd} derivative



(smoothed image, gaussian filter, og image)

1. $S I = G \times I$

(laplacian)

2. $\Delta^2 S = \frac{\delta^2}{\delta x^2} S + \frac{\delta^2}{\delta y^2} S$

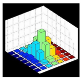
3. Zero crossing
   $[+,-]$ $[+,0,-]$
   $[-,+]$ $[-,0,+]$

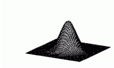
↳ slope $= |a+b|$

↳ apply threshold to slope

1 • Gaussian smoothing

$$\overbrace{S}^{\text{smoothed image}} = \overbrace{g}^{\text{Gaussian filter}} * \overbrace{I}^{\text{image}}$$





2 • Find Laplacian

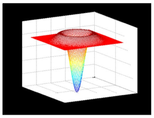
$$g(x,y) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

$$\Delta^2 S = \underbrace{\frac{\partial^2}{\partial x^2} S}_{\text{second order derivative in } x} + \underbrace{\frac{\partial^2}{\partial y^2} S}_{\text{second order derivative in } y}$$

- ∇ is used for gradient (derivative)
- Δ is used for Laplacian

• Deriving the Laplacian of Gaussian (LoG)

$$\Delta^2 S = \Delta^2(g * I) = (\Delta^2 g) * I$$

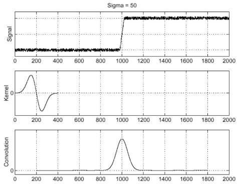


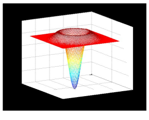
$$g(x,y) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

$$\Delta^2 g(x,y) = -\frac{1}{\sqrt{2\pi}\sigma^3}\left(2 - \frac{x^2+y^2}{\sigma^2}\right) e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Finding Zero Crossings

- Four cases of zero-crossings :
  - {+,-}
  - {+,0,-}
  - {-,+}
  - {-,0,+}
- Slope of zero-crossing {a, -b} is |a+b|.
- To mark an edge
  - Compute slope of zero-crossing
  - Apply a threshold to slope





LOG Filter

Laplacian of Gaussian (LoG):

- The combination of Gaussian smoothing and the Laplacian operator is called the Laplacian of Gaussian (LoG).
- LoG is defined as the second derivative of the Gaussian function

$$\Delta^2 G_\sigma = -\frac{1}{\sqrt{2\pi}\sigma^3}\left(2-\frac{x^2+y^2}{\sigma^2}\right)e^{-\frac{x^2+y^2}{2\sigma^2}}$$

			Y			
0.0008	0.0066	0.0215	0.031	0.0215	0.0066	0.0008
0.0066	0.0438	0.0982	0.108	0.0982	0.0438	0.0066
0.0215	0.0982	0	-0.242	0	0.0982	0.0215
0.031	0.108	-0.242	-0.7979	-0.242	0.108	0.031 X
0.0215	0.0982	0	-0.242	0	0.0982	0.0215
0.0066	0.0438	0.0982	0.108	0.0982	0.0438	0.0066
0.0008	0.0066	0.0215	0.031	0.0215	0.0066	0.0008

On the Separability of LoG

- Similar to separability of Gaussian filter
  - 2D Gaussian can be separated into 2 one-dimensional Gaussians

$$g(x,y) = \frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{x^2+y^2}{2\sigma^2}} = \frac{1}{\sqrt{2\pi}\sigma}\left(e^{-\frac{x^2}{2\sigma^2}}\right)\left(e^{-\frac{y^2}{2\sigma^2}}\right)$$

$g_1(x)$ $g_2(y)$

$$h(x,y) = I(x,y) * g(x,y) \quad n^2 \text{ multiplications}$$

$$h(x,y) = (I(x,y) * g_1(x)) * g_2(y) \quad 2n \text{ multiplications}$$

$$g_1 = [.011 \quad .13 \quad .6 \quad 1 \quad .6 \quad .13 \quad .011] \qquad g_2 = \begin{bmatrix} .011 \\ .13 \\ .6 \\ 1 \\ .6 \\ .13 \\ .011 \end{bmatrix}$$

$$\Delta^2 S = \Delta^2(g * I) = (\Delta^2 g) * I = I * (\Delta^2 g)$$

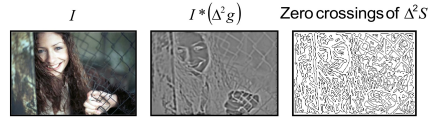
Requires n^2 multiplications

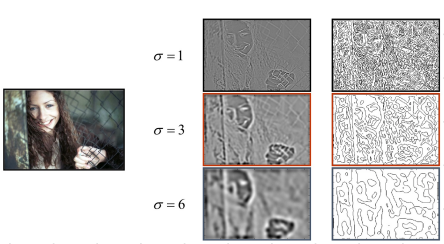
$$\Delta^2 S = (I * g_{yy}(y)) * g(x) + (I * g_{xx}(x)) * g(y)$$

Requires $4n$ multiplications

Example

I $\quad I * (\Delta^2 g)$ $\quad$ Zero crossings of $\Delta^2 S$





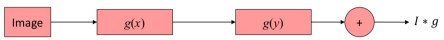
Seperatability

- A filter is **separable** if it can be broken down into multiple, simpler filters that are applied sequentially.
- For example, a 2D filter can often be separated into two 1D filters (one for the horizontal direction and one for the vertical direction).

The 2D Gaussian function is separable, meaning it can be expressed as the product of two 1D Gaussian functions:

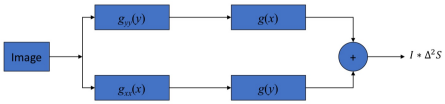
$G(x,y) = G(x) \cdot G(y)$

Gaussian Filtering



The Laplacian operator is not inherently separable, but when combined with the Gaussian, the LoG can be approximated in a separable way. by applying a 1D Gaussian filter followed by a 1D second derivative filter in both the horizontal and vertical directions.

Laplacian of Gaussian Filtering



Algorithm

- Compute LoG
  - Use 2D filter $\Delta^2 g(x,y)$
  - Use 4 1D filters $g(x),\ g_{xx}(x),\ g(y),\ g_{yy}(y)$
- Find zero-crossings from each row
- Find slope of zero-crossings
- Apply threshold to slope and mark edges

4. Canny Edge Detector

pros
↳ detects edges accurately while minimizing noise and false edges

Steps

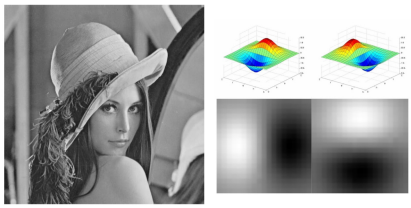
1. Smooth image with Gaussian Filter → to reduce noise that cause false edge detection
2. Compute derivative of filtered image
3. Find magnitude and orientation of gradient
4. Apply Non max Supression → removes weak edges by only keeping local maxima
5. Apply Thresholding (Hysteresis)

pros
↳ finds optimal edges
↳ reduces false edges
↳ provides good localization → how close detected edges to true edges

cons
↳ more expensive

1. **Noise Reduction (Gaussian Smoothing)**:
   - The first step is to smooth the image using a **Gaussian filter** to reduce noise.
   - This helps prevent false edge detection caused by noise.

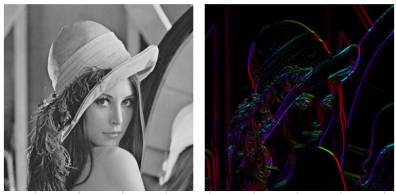
Canny



2. **Gradient Calculation**:
   - Compute the gradient of the image intensity using a gradient operator (e.g., Sobel or Prewitt).
   - This gives two gradient images:
     - **Gx**: Gradient in the horizontal direction (detects vertical edges).
     - **Gy**: Gradient in the vertical direction (detects horizontal edges).

Canny-Gradients

3. Gradient Orientation

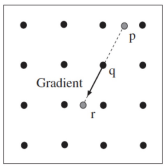


→ apply diff colors to edges in same direction

4. Non-Maximum Suppression:

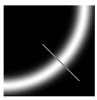
- This step thins the edges by keeping only the local maxima in the gradient magnitude image.
- For each pixel, the algorithm checks if it is the maximum along the gradient direction. If not, it is suppressed (set to zero).

Non-maximum suppression

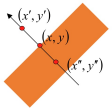


If gradient responses at r and p are smaller than q, q is an edge

Non-maximum suppression



$$M(x,y)=\begin{cases}|\nabla S|(x,y) & \text{if } |\nabla S|(x,y)>|\Delta S|(x',y') \\ & \&\ |\Delta S|(x,y)>|\Delta S|(x'',y'') \\ 0 & \text{otherwise}\end{cases}$$



x' and x'' are the neighbors of x along normal direction to an edge

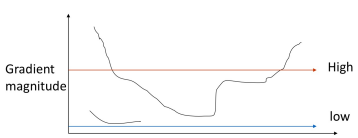
Before Non-Max Suppression

After Non-Max Suppression

5. Double Thresholding:

- Two thresholds are applied to the gradient magnitude image:
  - **High threshold**: Strong edges are kept.
  - **Low threshold**: Weak edges are kept only if they are connected to strong edges.
- This helps distinguish between strong edges, weak edges, and noise.

Hysteresis Thresholding [L, H]

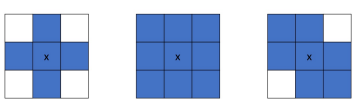


- If the gradient at a pixel is
  - above "**High**", declare it as an '**edge pixel**'
  - below "**Low**", declare it as a "**non-edge-pixel**"
  - **between** "low" and "high"
    - Consider its neighbors iteratively then declare it an "edge pixel" if it is **connected** to an 'edge pixel' **directly** or via pixels **between** "low" and "high".

Edge Tracking by Hysteresis:

- Weak edges that are connected to strong edges are retained as part of the final edge map.
- Weak edges that are not connected to strong edges are discarded (considered noise).

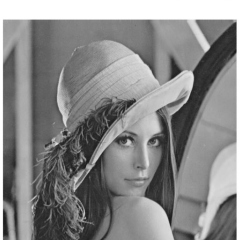
Hysteresis Thresholding [L, H]

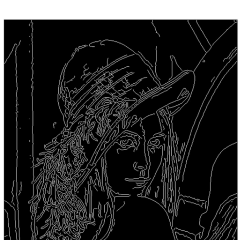


4 connected 8 connected 6 connected

1. Threshold at low/high levels to get weak/strong edge pixels
2. Do connected components, starting from strong edge pixels

Final Canny Edges





Effect of Gaussian kernel (smoothing)

Smoothing with a Gaussian kernel **reduces noise** but also **blurs** the image.

The amount of smoothing depends on the **σ** value.

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

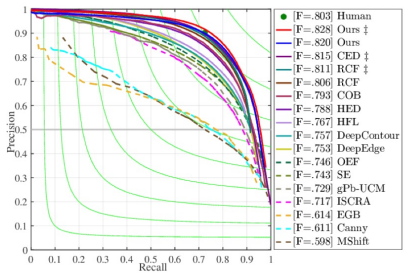
A smaller σ (standard deviation) value results in
-less smoothing.
-detects fine features and
-smaller edges

A larger σ value results in
-more smoothing
- detects large-scale edges and
-ignores smaller details.

- **Effect of σ (Standard Deviation):**
  - **Small σ**: Better for detecting fine details but more sensitive to noise.
  - **Large σ**: Better for detecting large-scale edges but may miss small details.
- **Application**:
  - The choice of σ depends on the desired behavior:
    - Use a **small σ** if you want to detect fine features.
    - Use a **large σ** if you want to detect large-scale edges and reduce noise.

Edge Detection with Deep Learning

- We will revisit edge detection
  - After Deep Learning tutorial lectures
  - If time permits

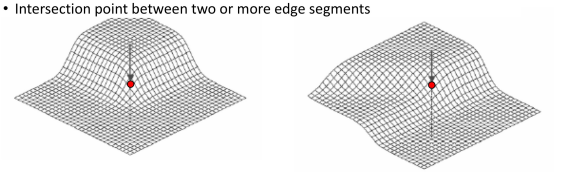


INTEREST POINT DETECTION

Interest Point

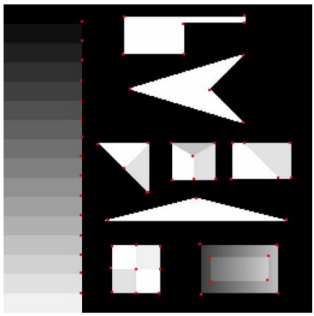
is a location in an image that is:
Highly distinctive: It stands out from its neighbors.
Stable under transformations: It remains recognizable despite changes in scale, rotation, or lighting.
Useful for matching across images.

- Expressive texture
  - The point at which the direction of the boundary of object changes abruptly
  - Intersection point between two or more edge segments



Synthetic & Real Interest Points

4 corners are indicated in red



Types of Interest Points

Corners: Where two edges meet.
Blobs: Distinct, uniform regions (e.g., circles in images).
Edges: Sudden intensity changes.

How to Detect Interest Points

There are two main approaches:

1. **Brightness-Based Approach**
   Uses image derivatives (rate of intensity change).
   Works well for high contrast regions.
2. **Boundary-Based Approach**
   Detects points at edge intersections.
   Requires edge detection + curvature analysis.

Properties of Interest Point Detectors

A good interest point detector should:

- ✅ Detect **most** true interest points.
- ✅ Avoid **false** detections.
- ✅ Be **precise** (well localized).
- ✅ Be **robust** to noise, lighting, and transformations.
- ✅ Be **efficient** (fast enough for real-time applications).

Interest Point Dechon Uses

- Automate object tracking
- Point matching for computing disparity
- Stereo calibration
  - Estimation of fundamental matrix
- Motion based segmentation
- Recognition
- 3D object reconstruction
- Robot navigation
- Image retrieval and indexing

Real-World Examples:

✅ **Facial Recognition**: Detects keypoints like eyes, nose, and mouth corners.

✅ **Self-Driving Cars**: Identifies road markings, traffic signs, and objects.

✅ **3D Reconstruction**: Matches interest points across multiple views to infer depth.

1. Object Tracking
Track consistent features across video frames.
Example: Tracking a player in a sports video.

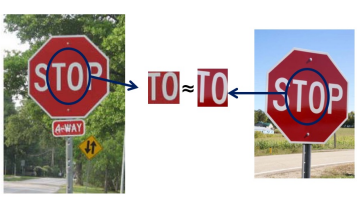
2. Stereo Vision (Depth Estimation)
Match points across left and right images to estimate depth.
Used in 3D reconstruction.

3. Image Stitching (Panoramas)
Detect interest points and match them across overlapping images.
Example: Creating a wide-angle panorama from multiple images.

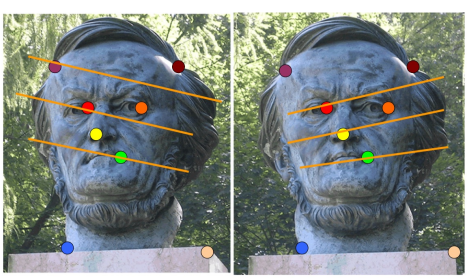
4. Feature-Based Object Recognition
Detect key points and describe them with feature descriptors.
Example: Recognizing landmarks in images.

Correspondence across views

- Correspondence: matching points, patches, edges, or regions across images



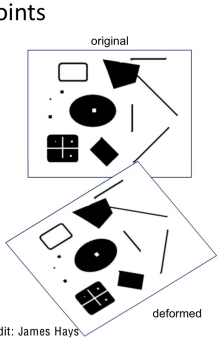
Example: estimating "fundamental matrix" that corresponds two views



Example: structure from motion

This class: interest points

- Suppose you have to click on some point, go away and come back after I deform the image, and click on the same points again.
  - Which points would you choose?



Slide Credit: James Hays

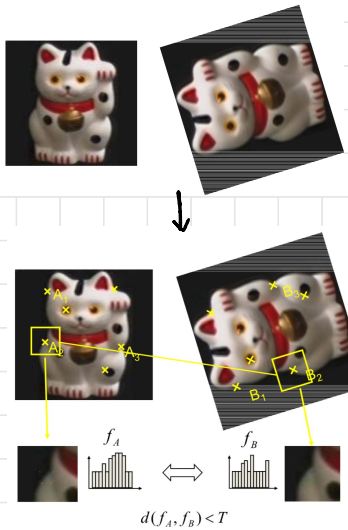
Keypoint Matching

enable matching across images.

1. Detect Keypoints – Find corners or blobs.
2. Define a Region – Extract a neighborhood around the keypoint.
3. Normalize – Adjust for scale and rotation.
4. Compute Descriptor – Convert the patch into a unique vector.
5. Match Descriptors – Compare descriptors from different images.

Keypoint Goal

↳ Detect Points that are
↳ repeatable
↳ distinctive



Trade Offs

Detection of interest points

More Repeatable
Robust detection
Precise localization

More Points
Robust to occlusion
Works with less texture

Description of patches

More Distinctive
Minimize wrong matches

More Flexible
Robust to expected variations
Maximize correct matches

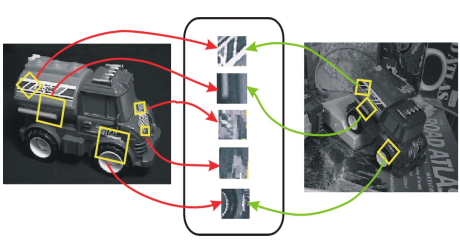
More Repeatable	More Distinctive
Detecting many points	Detecting fewer, but unique points
Robust to noise	Robust to transformations
Works with fewer textures	Works with occlusion

never changing

Invariant local Features

A good feature should be invariant to transformations:

Translation – Interest points should move consistently.
Rotation – Descriptors should stay the same even if the image rotates.
Scaling – Features should be detectable at multiple scales.
Lighting Changes – Interest points should be robust to brightness changes.



Features Descriptors

Choosing Interest Points

Where would you tell your friend to meet you?

Feature extraction: Corners

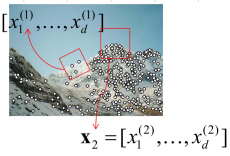
Local Features

↳ the main components are

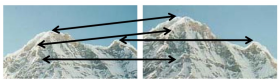
1) **Detection**: Identify the interest points
2) **Description** :Extract feature vector descriptor surrounding each interest point.
3) **Matching**: Determine correspondence between descriptors in two views

1

2



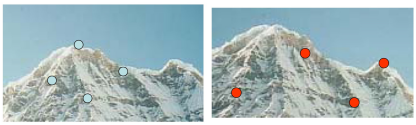
3



Goal

1. Interest Operator Repetability

↳ detecting the same points in both the images

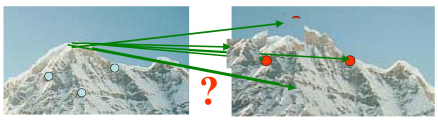


No chance to find true matches!

- Yet we have to be able to run the detection procedure *independently* per image.

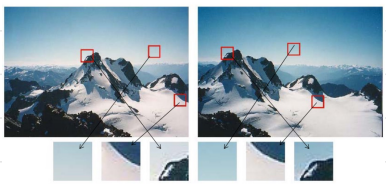
2. Descriptor Distinctiveness

↳ determine reliably which point goes to which



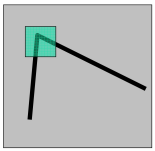
- Must provide some invariance to geometric and photometric differences between the two views.

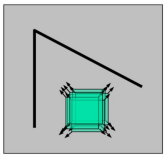
Some patches can be localized or matched with higher accuracy than others.



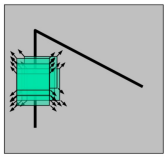
Harris Corner Detector

-Corner point can be recognized in a window
-If we shift a small window in any direction, a corner should produce a large intensity change in all directions.

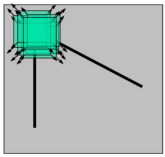




"flat" region: no change in all directions



"edge": no change along the edge direction



"corner": significant change in all directions

Maths of Harris Detector

uses eigenvalues to find corners.

1. Auto-correlation function

$$E(u,v) = \sum_{x,y} \underbrace{[I(x+u, y+v)}_{\text{shifted intensity}} - \underbrace{I(x,y)}_{\text{intensity}}]^2$$

Measures the change in intensity when shifting the image.

2. Taylor series Expansion

Approximate E(u,v) using the Taylor Series Expansion:

2 Approximate $E(u,v)$ using the **Taylor Series Expansion**:

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix}$$

- Where M is the **second-moment matrix**:

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

  - I_x, I_y are image derivatives.
  - $w(x,y)$ is a Gaussian window function.

3. Compute Eigenvalues λ_1, λ_2

Corner: λ1 λ 2 are large.
Edge: One eigenvalue is large, the other is small.
Flat Region: Both eigenvalues are small.

4. Corner Response Function

$R = \det(M) - k \cdot (\text{trace}(M))^2$

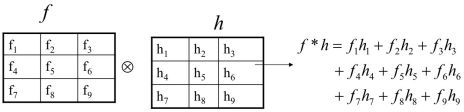
R > 0: Corner detected! 🎯
R < 0: Edge detected.
R ≈ 0: Flat region.

Correlation → fxh

$$f \otimes h = \sum_k \sum_l f(k,l)h(k,l)$$

f = Image

h = Kernel

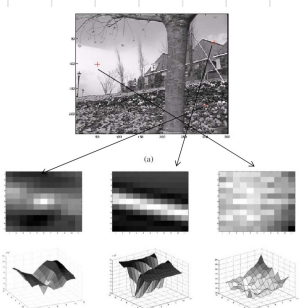


$$f \otimes h = \sum_k \sum_l f(k,l)h(k,l)$$ Cross correlation

$$f \otimes f = \sum_k \sum_l f(k,l)f(k,l)$$ Auto correlation

Auto correlation → f x f

measures how similar a signal (or image) is to a shifted version of itself.



Cross correlation

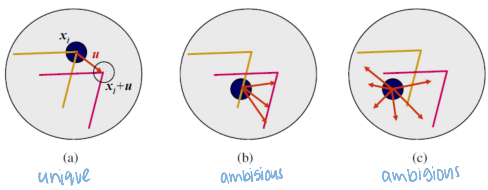
measures the similarity between two signals (or images) as one is shifted over the other.

Measures how similar the template is to each part of the image.
Used in template matching to find objects in an image.

Aperture Problem → Intrest Point's Problem

arises when we observe motion through a small window (aperture).
If we can only see a small part of an image, we may misinterpret the true direction of motion.

The problem exists because edges have ambiguous motion perception.



Sum of Squared Differences (SSD)

→ measures how similar 2 patches are

- **SSD** measures **how similar two patches** are:

$$SSD = \sum_{x,y} (I(x,y) - J(x,y))^2$$

- **Cross-Correlation** compares patterns:

$$R = \sum_{x,y} I(x,y) \cdot J(x,y)$$

  - Used in **template matching**.

Correlation Vs SSD

$$SSD = \sum_k \sum_l (f(k,l) - h(k,l))^2$$ Sum of Squares Difference

minimize

$$SSD = \sum_k \sum_l \left(f(k,l)^2 - 2h(k,l)f(k,l) + h(k,l)^2\right)$$

minimize

$$SSD = \sum_k \sum_l (-2h(k,l)f(k,l))$$ These terms do not depend on correlation

$$SSD = \sum_k \sum_l (2h(k,l)f(k,l))$$

maximize

$$Correlation = \sum_k \sum_l (h(k,l)f(k,l))$$

maximize

$$f \otimes f = \sum_k \sum_l f(k,l)f(k,l)$$

Tradeoffs:
SSD is faster, but sensitive to intensity changes.
Correlation is more robust, but slower.

Taylor Series

$f(x)$ Can be represented at point a in terms of its derivatives

$$f(x) = f(a) + (x-a)f_x + \frac{(x-a)^2}{2!}f_{xx} + \frac{(x-a)^3}{3!}f_{xxx} + \dots$$

Express $I(x+u, y+v)$ at (x, y):

$$I(x+u, y+v) = I(x,y) + I_x(x+u-x) + I_y(y+v-y)$$

$$I(x+u, y+v) = I(x,y) + I_x u + I_y v$$

$$f(x,y,t) = f(x+dx, y+dy, t+dt)$$

Taylor Series of right side

$$f(x,y,t) = f(x,y,t) + \frac{\partial f}{\partial x}(x+dx-x) + \frac{\partial f}{\partial y}(y+dy-y) + \frac{\partial f}{\partial t}(t+dt-t)$$

$$0 = f_x dx + f_y dy + f_t dt$$

$$0 = f_x \frac{dx}{dt} + f_y \frac{dy}{dt} + f_t \frac{dt}{dt}$$

$$0 = f_x u + f_y v + f_t \qquad f_x u + f_y v + f_t = 0$$

Optical Flow Constrain Equation

Brook Taylor (1685-1731)

His marriage in 1721 with Miss Brydges of Wallington, Surrey, led to an estrangement from his father, which ended in 1723 after her death in giving birth to a son, who also died.

1725 he married—this time with his father's approval—Sabetta Sawbridge of Olantigh, Kent, who also died in childbirth in 1730 ; in this case, however, his daughter, Elizabeth, survived.

Taylor was elected a fellow of the Royal Society early in 1712, and in the same year sat on the committee for adjudicating the claims of Sir Isaac Newton and Gottfried Leibniz about Calculus.

Maths of Harris Detector

- Change of intensity for the shift (u,v)

$$E(u,v) = \sum_{x,y} \underbrace{[I(x+u, y+v)}_{\text{shifted intensity}} - \underbrace{I(x,y)]^2}_{\text{intensity}}$$

Auto-correlation

$$E(u,v) = \sum_{x,y} [\underbrace{I(x+u, y+v)}_{\text{shifted intensity}} - \underbrace{I(x,y)}_{\text{intensity}}]^2$$

$$E(u,v) = \sum_{x,y} [\underbrace{I(x,y) + uI_x + vI_y}_{\text{shifted intensity}} - \underbrace{I(x,y)}_{\text{intensity}}]^2$$

Taylor Series

$$E(u,v) = \sum_{x,y} [uI_x + vI_y]^2$$

$$E(u,v) = \sum_{x,y} \left[\begin{pmatrix} u & v \end{pmatrix} \begin{pmatrix} I_x \\ I_y \end{pmatrix} \right]^2$$

$$E(u,v) = \sum_{x,y} \begin{pmatrix} u & v \end{pmatrix} \begin{pmatrix} I_x \\ I_y \end{pmatrix} \begin{pmatrix} I_x & I_y \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}$$

$$M = \sum_{x,y} \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix}$$

$$E(u,v) = \begin{pmatrix} u & v \end{pmatrix} \left[\sum_{x,y} \begin{pmatrix} I_x \\ I_y \end{pmatrix} \begin{pmatrix} I_x & I_y \end{pmatrix} \right] \begin{pmatrix} u \\ v \end{pmatrix}$$

$$E(u,v) = \begin{pmatrix} u & v \end{pmatrix} M \begin{pmatrix} u \\ v \end{pmatrix}$$

a) Given two points x=(2,4) and y=(3,7), find the line passing through the two points using homogenous coordinates and cross product

↳ Convert to homogenous

$x = (2,4,1)$

$y = (3,7,1)$

↳ cross product

$L = x \cdot y$

↳ find determinant

$$L = \begin{vmatrix} i & j & k \\ 2 & 4 & 1 \\ 3 & 7 & 1 \end{vmatrix}$$

$d = i((4\times1)-(7\times1)) - j(2-3) + k(14-12)$

$= -3i + j + 2k$

↳ form line eq

$L = (-3, 1, 2)$

$-3x + y + 2 = 0$

$3x - y - 2 = 0$

b) Derive the rotation matrix R such that a=Rb, where image a is the rotated version of image b in non-homogenous or conventional cartesian coordinates.

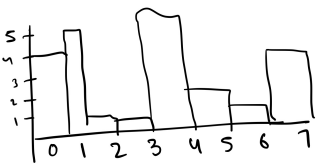
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

↳ Generalise as $a = Rb$

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Given the following grayscale image with each pixel represented by three bits.

1	4	5	6	7
7	7	0	0	4
7	7	0	0	2
1	1	4	4	3
1	1	4	4	5



Question 3: (CLO2/C5) **[10+5=15]**

Given the following grayscale image with each pixel represented by three bits and a 3x3 filter.

Image:

1	4	5	6	7
7	7	0	0	4
7	7	0	0	2
1	1	4	4	3
1	1	4	4	5

kernel:

1	2	1
0	0	0
-1	-2	-1

a) Construct the output filtered image using the above kernel and mirror padding.
b) Comment on the comparison of the given and output images: how they are different.

Filter		
-1	0	2
0	2	1
-3	1	2

Image				
0	0	1	3	2
2	3	6	0	3
5	4	2	5	7
1	2	0	4	4

median

0 1 2 2 2 3 4 5 6

$$\begin{bmatrix} X & X & X & X & X \\ X & 2 & 3 & 3 & X \\ X & 2 & 3 & 4 & X \\ X & X & X & X & X \end{bmatrix}$$

correlation

$(2\times-1)+(3\times0)+(6\times2)+$
$(5\times0)+(4\times2)+(2\times1)+$
$(1\times-3)+(2\times1)+(0\times2)=19$

$$\begin{bmatrix} X & X & X & X & X \\ X & 7 & 18 & 19 & X \\ X & 19 & 8 & 29 & X \\ X & X & X & X & X \end{bmatrix}$$

convolution

flip filter horizontally then vertically

$$\begin{bmatrix} 2 & 0 & -1 \\ 1 & 2 & 0 \\ 2 & 1 & -3 \end{bmatrix} \rightarrow \begin{bmatrix} 2 & 1 & -3 \\ 1 & 2 & 0 \\ 2 & 0 & -1 \end{bmatrix}$$ same steps $$\begin{bmatrix} X & X & X & X & X \\ X & 13 & 10 & 2 & X \\ X & 4 & 20 & 11 & X \\ X & X & X & X & X \end{bmatrix}$$

Sobel Filters (for correlation)					
-1	0	1	-1	-2	-1
-2	0	2	0	0	0
-1	0	1	1	2	1
dx			dy		

Image				
0	0	1	3	2
2	3 (A)	6	0	3
5	4	2 (B)	5 (C)	7
1	2	0	4	4

A) For the image I above compute the **gradient** $(dI/dx,\ dI/dy)$, **magnitude of**

For the image I above compute the gradient (dI/dx, dI/dy), magnitude of gradient and the angle of gradient for pixel locations A, B and C marked below. You have to use the Sobel filters (above) to compute the derivatives (no normalization is needed, that is you do not need to divide by 8 after applying the sobel filter). You need to write down how you derived each of these quantities. You may use arctan to represent the angles. The y axis is pointing downwards. (15 points)

at Pixel A

$$\begin{bmatrix} 0 & 0 & 1 \\ 2 & 3 & 6 \\ 5 & 4 & 2 \end{bmatrix}$$

1. Compute Derivatives

$\frac{dI}{dx} = (-1\times0)+(0\times0)+(1\times1)+$
$(-2\times2)+(0\times3)+(2\times6)+$
$(-1\times5)+(0\times4)+(1\times2) = 6$

$\frac{dI}{dy} = (-1\times0)+(-2\times0)+(-1\times1)+$
$(0\times2)+(0\times3)+(0\times6)+$
$(1\times5)+(2\times4)+(1\times2) = 14$

2. Gradient Magnitude

$\|\nabla I\| = \sqrt{\frac{dI}{dx}^2 + \frac{dI}{dy}^2}$
$= \sqrt{6^2+14^2} =$

repeat for B, C

3. Gradient Angle

$\theta = \tan^{-1}\left(\frac{dI}{dy} / \frac{dI}{dx}\right)$
$= \tan^{-1}(14/6) =$

$I_x = dI/dx$				
3	2	1	-1	-1
4	3	2	0	-1
4	3	4	2	1
1	1	3	2	2

$I_y = dI/dy$				
2	3	1	1	-1
2	3	2	-1	1
2	4	4	1	2
-1	0	3	2	3

A) Compute the Harris matrix for the 3 by 3 window highlighted in the above images. The Harris matrix is defined as (10 points)

$$H = \sum_{(x,y)\in W} \begin{bmatrix} I_x(x,y)^2 & I_x(x,y)I_y(x,y) \\ I_x(x,y)I_y(x,y) & I_y(x,y)^2 \end{bmatrix}$$

where W is the window highlighted in the gradient images.

$I_x^2 \rightarrow h_{11} = 3^2+2^2+0^2+3^2+4^2+2^2+1^2+3^2+2^2 = 56$

$I_y^2 \rightarrow h_{22} = 3^2+2^2+(-1)^2+4^2+4^2+1^2+0^2+3^2+2^2 = 60$

$I_{xy} \rightarrow h_{12} = h_{21} = (3\times3)+(2\times2)+(0\times(-1))+(3\times4)+(4\times4)+(2\times1)+(1\times0)+(3\times3)+(2\times2) = 56$

$$H = \begin{bmatrix} 56 & 56 \\ 56 & 60 \end{bmatrix}$$

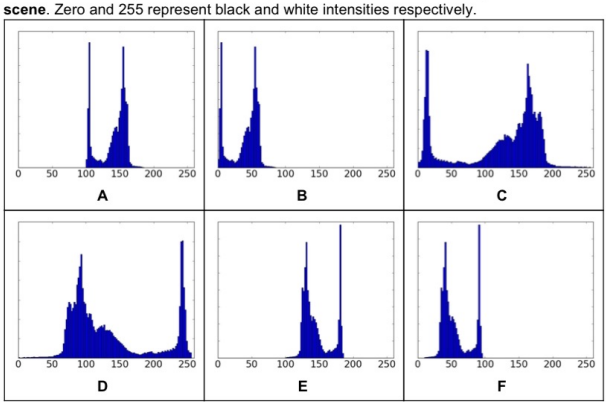
B) Compute the cornerness score C = det(H) – k trace(H)2 for the above window, where det and trace represent the determinant and trace (sum of diagonal elements) of a matrix. Use k = 0.04. (3 points)

det H, k, trace

$C = (56\times60)-(56\times56) - 0.04(56+60)^2 = -314.24$

Question 4 - Histogram (18 points)

Look at the following histograms for images A, B, C, D, E and F, **all images of the same scene**. Zero and 255 represent black and white intensities respectively.



A) Which of the above histograms correspond to the following image? Explain your answer. (6 points)

The histogram must have two peaks. A relatively narrow peak at darker intensities (camera man's coat and tripod's legs), and a wider peak mostly consisting of brighter intensities (the grass, the sky, the building in the background, etc.). This is because there are fewer dark pixels compared to the bright ones. Thus the answer is **C**.

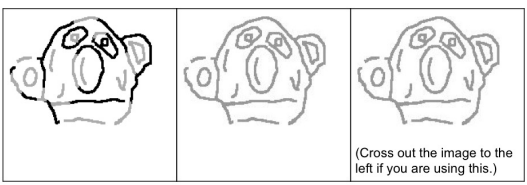
B) Which of the images **D** and **E** has a higher contrast? Why? (4 points)
D has a higher contrast since it has a wider histogram, covering a wider range of intensities.

C) How does image **B** look like compared to image **A**? Explain. (4 points)
B is generally darker than A, as its histogram has been shifted to the left. But, both have about the same contrast.

D) What is the relation between images **C** and **D**? Explain your answer. (4 points)
C is **D** with inverted intensities (negative of **D**). Mathematically **C** = 255 - **D**.

Question 1- Canny Edge detector (10 points)

Consider the images below. The leftmost image represents the result of Canny low- and high- thresholding. The black lines represent the ***strong edges*** and the gray lines indicate the weak edges ***weak edges***. Remember, weak edges are edges whose gradient response (magnitude) is between the low and the high thresholds, and strong edges are those whose gradient magnitude is larger than the high threshold. On the outline image on the right draw the final edges obtained by the Canny edge detector. If you screw up one of the outline images, use the other one, but, only use one of them (10 points).



Answer:

Eigen Vectors and Eigen Values

The eigen vector, x, of a matrix A is a special vector, with the following property

$$Ax=\lambda x$$ Where λ is called eigen value

To find eigen values of a matrix A first find the roots of:

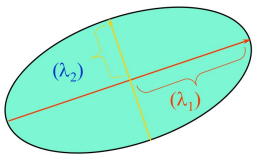
$$\det(A-\lambda I)=0$$

Then solve the following linear system for each eigen value to find corresponding eigen vector

$$(A-\lambda I)x=0$$

$$E(u,v)=\begin{pmatrix}u & v\end{pmatrix}M\begin{pmatrix}u\\ v\end{pmatrix} \qquad M=\sum_{x,y}\begin{bmatrix}I_xI_x & I_xI_y\\ I_xI_y & I_yI_y\end{bmatrix}$$

- E(u,v) is an equation of an ellipse.
- Let λ_1 and λ_2 be eigenvalues of M



Example

$$A=\begin{bmatrix}-1 & 2 & 0\\ 0 & 3 & 4\\ 0 & 0 & 7\end{bmatrix}$$ Eigen Values

$$\lambda_1=7,\ \lambda_2=3,\lambda_3=-1$$

$$\mathbf{x_1}=\begin{bmatrix}1\\ 4\\ 4\end{bmatrix},\ \mathbf{x_2}=\begin{bmatrix}1\\ 2\\ 0\end{bmatrix},\ \mathbf{x_3}=\begin{bmatrix}1\\ 0\\ 0\end{bmatrix}$$ Eigen Vectors

Eigen Values

$$\det(A-\lambda I)=0$$

$$\det\left(\begin{bmatrix}-1 & 2 & 0\\ 0 & 3 & 4\\ 0 & 0 & 7\end{bmatrix}-\lambda\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}\right)=0$$

$$\det\left(\begin{bmatrix}-1-\lambda & 2 & 0\\ 0 & 3-\lambda & 4\\ 0 & 0 & 7-\lambda\end{bmatrix}\right)=0$$

$$(-1-\lambda)((3-\lambda)(7-\lambda)-0)=0$$
$$(-1-\lambda)(3-\lambda)(7-\lambda)=0$$
$$\lambda=-1,\quad \lambda=3,\quad \lambda=7$$

Eigen Vectors

$\lambda=-1$

$$(A-\lambda I)x=0$$

$$\left(\begin{bmatrix}-1 & 2 & 0\\ 0 & 3 & 4\\ 0 & 0 & 7\end{bmatrix}+\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}\right)\begin{bmatrix}x_1\\ x_2\\ x_3\end{bmatrix}=\begin{bmatrix}0\\ 0\\ 0\end{bmatrix}$$

$$\begin{bmatrix}0 & 2 & 0\\ 0 & 4 & 4\\ 0 & 0 & 8\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\end{bmatrix}=\begin{bmatrix}0\\ 0\\ 0\end{bmatrix}$$

$$0+2x_2+0=0$$
$$0+4x_2+4x_3=0$$
$$0+0+8x_3=0$$
$$x_1=1,\quad x_2=0,\quad x_3=0$$

$$\mathbf{x_1}=\begin{bmatrix}1\\ 0\\ 0\end{bmatrix}$$

MATLAB Fuction

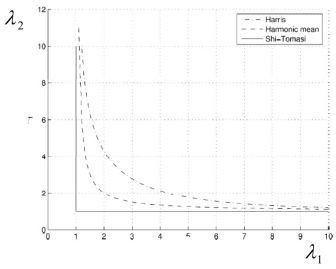
```
[vectorC,valueC]=eig(C);
```

Other Version of Harris Detectors

$$R = \lambda_1 - \alpha\lambda_2$$ **Triggs**

$$R = \frac{\det(D)}{trace(D)} = \frac{\lambda_1\lambda_2}{\lambda_1 + \lambda_2}$$ **Szeliski (Harmonic mean)**

$$R = \lambda_1$$ **Shi-Tomasi**



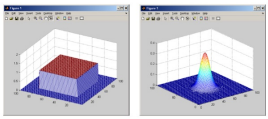
Mathematics of Harris Detector

- Change of intensity for the shift (u,v)

$$E(u,v) = \sum_{x,y} \underbrace{[I(x+u, y+v)}_{\text{shifted intensity}} - \underbrace{I(x,y)]}_{\text{intensity}}^2$$

Auto-correlation

Window functions →



UNIFORM GAUSSIAN

$$E(u,v) = \sum_{x,y} \underbrace{w(x,y)}_{\text{window function}} [\underbrace{I(x+u, y+v)}_{\text{shifted intensity}} - \underbrace{I(x,y)}_{\text{intensity}}]^2$$

$$E(u,v) = \sum_{x,y} \underbrace{w(x,y)}_{\text{window function}} [\underbrace{I(x,y) + uI_x + vI_y}_{\text{shifted intensity}} - \underbrace{I(x,y)}_{\text{intensity}}]^2$$ Taylor Series

$$E(u,v) = \sum_{x,y} w(x,y)[uI_x + vI_y]^2$$

$$E(u,v) = \sum_{x,y} w(x,y)\begin{bmatrix} u & v \end{bmatrix}\begin{pmatrix} I_x \\ I_y \end{pmatrix}^2$$

$$E(u,v) = \sum_{x,y} w(x,y)\begin{bmatrix} u & v \end{bmatrix}\begin{pmatrix} I_x \\ I_y \end{pmatrix}\begin{pmatrix} I_x & I_y \end{pmatrix}\begin{pmatrix} u \\ v \end{pmatrix}$$

$$M = \sum_{x,y}\begin{bmatrix} I_xI_x & I_xI_y \\ I_xI_y & I_yI_y \end{bmatrix}$$

$$E(u,v) = \begin{pmatrix} u & v \end{pmatrix}\left[\sum_{x,y} w(x,y)\begin{pmatrix} I_x \\ I_y \end{pmatrix}\begin{pmatrix} I_x & I_y \end{pmatrix}\right]\begin{pmatrix} u \\ v \end{pmatrix}$$

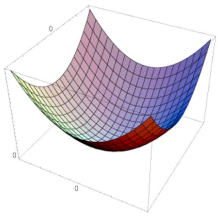
$$E(u,v) = \begin{pmatrix} u & v \end{pmatrix} M \begin{pmatrix} u \\ v \end{pmatrix}$$

Interpreting the second moment matrix

The surface $E(u,v)$ is locally approximated by a quadratic form. Let's try to understand its shape.

$$E(u,v) \approx [u \;\; v] \; M \begin{bmatrix} u \\ v \end{bmatrix}$$

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$



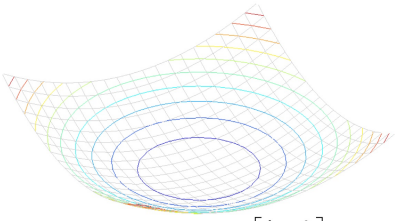
First, consider the axis-aligned case (gradients are either horizontal or vertical)

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

If either λ is close to 0, then this is **not** a corner, so look for locations where both are large.

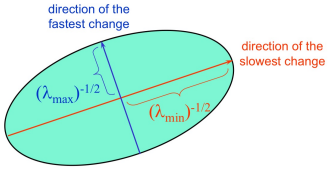
Consider a horizontal "slice" of $E(u, v)$: $[u \;\; v] \; M \begin{bmatrix} u \\ v \end{bmatrix} = \text{const}$

This is the equation of an ellipse.



Diagonalization of M: $M = R^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R$

The axis lengths of the ellipse are determined by the eigenvalues and the orientation is determined by R



Corners as distinctive interest points

$$M = \sum w(x,y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix}$$

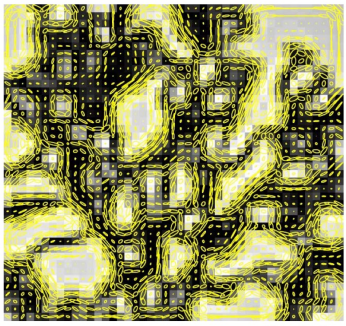
2 x 2 matrix of image derivatives (averaged in neighborhood of a point).




Notation: $I_x \Leftrightarrow \frac{\partial I}{\partial x}$ $I_y \Leftrightarrow \frac{\partial I}{\partial y}$ $I_x I_y \Leftrightarrow \frac{\partial I}{\partial x}\frac{\partial I}{\partial y}$

Slide Credit: James Hays

Visualization of second moment matrices



Slide Credit: James Hays

Algorithm

- Compute horizontal and vertical derivatives of image I_x and I_y.
- Compute three images corresponding to three terms in matrix *M*.
- Convolve these three images with a larger Gaussian (window).
- Compute scalar cornerness value using one of the *R* measures.
- Find local maxima above some threshold as detected interest points.

Harris Detector [Harris88]

- Second moment matrix

$$\mu(\sigma_I,\sigma_D) = g(\sigma_I) * \begin{bmatrix} I_x^2(\sigma_D) & I_x I_y(\sigma_D) \\ I_x I_y(\sigma_D) & I_y^2(\sigma_D) \end{bmatrix}$$

1. Image derivatives (optionally, blur first)

2. Square of derivatives

$\det M = \lambda_1\lambda_2$
$\operatorname{trace} M = \lambda_1 + \lambda_2$

3. Gaussian filter $g(\sigma_I)$

4. Cornerness function – both eigenvalues are strong

$$har = \det[\mu(\sigma_I,\sigma_D)] - \alpha[\operatorname{trace}(\mu(\sigma_I,\sigma_D))^2] =$$
$$g(I_x^2)g(I_y^2) - [g(I_x I_y)]^2 - \alpha[g(I_x^2) + g(I_y^2)]^2$$

5. Non-maxima suppression

Slide Credit: James Hays

Invariance and covariance

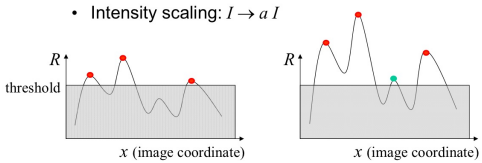
- We want corner locations to be *invariant* to photometric transformations and *covariant* to geometric transformations
  - **Invariance:** image is transformed and corner locations do not change
  - **Covariance:** if we have two transformed versions of the same image, features should be detected in corresponding locations

Slide Credit: James Hays

Affine intensity change

$$I \rightarrow a\,I + b$$

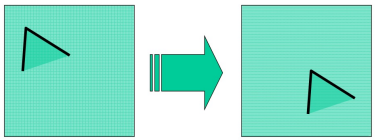
- Only derivatives are used => invariance to intensity shift $I \rightarrow I + b$
- Intensity scaling: $I \rightarrow a\,I$



Partially invariant to affine intensity change

Slide Credit: James Hays

Image translation

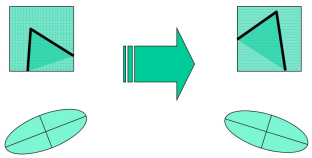


- Derivatives and window function are shift-invariant

Corner location is covariant w.r.t. translation

Slide Credit: James Hays

Image rotation

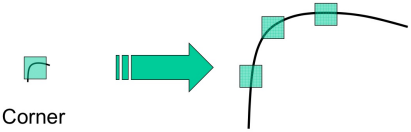


Second moment ellipse rotates but its shape (i.e. eigenvalues) remains the same

Corner location is covariant w.r.t. rotation

Slide Credit: James Hays

Scaling



All points will be classified as edges

Corner location is not covariant to scaling!

Slide Credit: James Hays

BASICS

Features

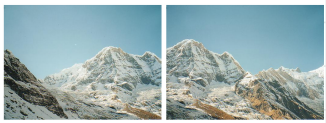
↳ info extracted from image/video
  ↳ handcrafted → feature engineering → HOG, SIFT
  ↳ learned → automatically learned → CNNs

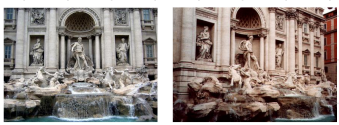
Types of Features

1. Global Features
   ↳ Extracted from the entire image
   eg template, HOG
2. Region based Features
   ↳ Extracted from a smaller window of image
   ↳ apply global method to specific region
3. Local features
   ↳ describe a pixel and the area around the pixel
   ↳ always refer to specific pixel locations

USES OF FEATURES

1. Template Matching
2. Structure from Motion
3. Panorama Stitching

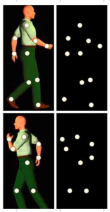


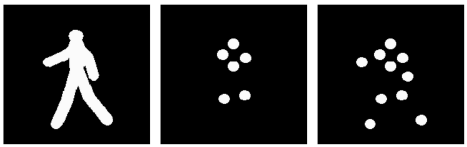


4. Detection
5. Recognition
6. Tracking
7. Stereo Estimation

Finding Features in videos

- Complex actions can be recognized on the basis of 'point-light displays',
  - Facial expressions,
  - Sign Language,
  - Arm movements,
  - Various full-body actions.





Characteristics of Good Features

1. Distinctiveness
   - ↳ each feature can be uniquely identified
2. Repeatability
   - ↳ The same feature can be found in several images
     - ↳ geometrically → translation, rotation, scale
     - ↳ Photometrically → reflectance, illumination
3. Compactness and efficiency
   - ↳ many fewer features than image pixels
   - ↳ run independently per image

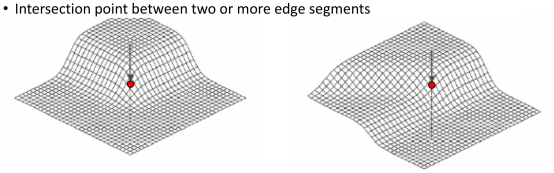
- We want the representation to be as small and as fast as possible
  - Much smaller than a whole image
- We'd like to be able to run the detection procedure *independently* per image
  - Match just the compact descriptors for speed.
  - *Difficult!* We don't get to see 'the other image' at match time, e.g., object detection.

INTEREST POINT DETECTION

Interest point

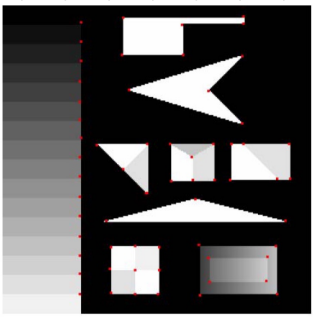
is a location in an image that is:
Highly distinctive: It stands out from its neighbors.
Stable under transformations: It remains recognizable despite changes in scale, rotation, or lighting.
Useful for matching across images.

- Expressive texture
  - The point at which the direction of the boundary of object changes abruptly
  - Intersection point between two or more edge segments



Symmetric & Real Interest Points

4 corners are indicated in red



Types of Interest Points

Corners: Where two edges meet.
Blobs: Distinct, uniform regions (e.g., circles in images).
Edges: Sudden intensity changes.

How to Detect Interest Points

There are two main approaches:

1. **Brightness-Based Approach**
   Uses image derivatives (rate of intensity change).
   Works well for high contrast regions.
2. **Boundary-Based Approach**
   Detects points at edge intersections.
   Requires edge detection + curvature analysis.

Properties of Interest Point Detectors

A good interest point detector should:

- ✅ Detect **most** true interest points.
- ✅ Avoid **false** detections.
- ✅ Be **precise** (well localized).
- ✅ Be **robust** to noise, lighting, and transformations.
- ✅ Be **efficient** (fast enough for real-time applications).

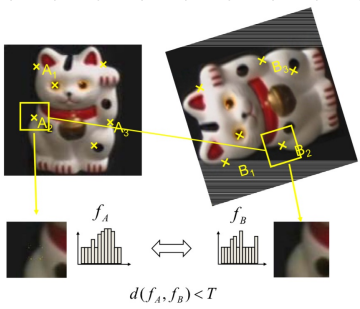
Keypoint Matching

enable matching across images.

1. Detect Keypoints – Find corners or blobs.
2. Define a Region – Extract a neighborhood around the keypoint.
3. Normalize – Adjust for scale and rotation.
4. Compute Descriptor – Convert the patch into a unique vector.
5. Match Descriptors – Compare descriptors from different images.

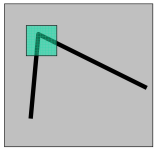
Keypoint Goal

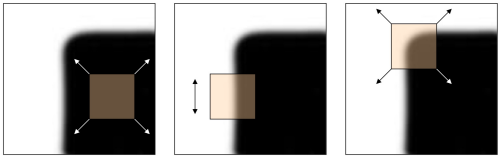
↳ Detect points that are
  ↳ repeatable
  ↳ distinctive



Corner Detection

-Corner point can be recognized in a window
-If we shift a small window in any direction, a corner should produce a large intensity change in all directions.





"Flat" region:
no change in all directions

"Edge":
no change along the edge direction

"Corner":
significant change in all directions

Template Matching

$$\begin{bmatrix} -4 & 5 & 5 \\ -4 & 5 & 5 \\ -4 & -4 & -4 \end{bmatrix} \quad \begin{bmatrix} 5 & 5 & 5 \\ -4 & 5 & -4 \\ -4 & -4 & -4 \end{bmatrix}$$

$$\begin{bmatrix} -4 & 5 & 5 \\ -4 & 5 & 5 \\ -4 & -4 & -4 \end{bmatrix} \quad \begin{bmatrix} 5 & 5 & 5 \\ -4 & 5 & -4 \\ -4 & -4 & -4 \end{bmatrix}$$

Complete set of eight templates can be generated by successive 90 degree of rotations.

Why the summation of filter is 0? → Insensitive to absolute change In intensity!

Correlation → fxh

f → image

f_1	f_2	f_3
f_4	f_5	f_6
f_7	f_8	f_9

$\otimes$

h → kernel

h_1	h_2	h_3
h_4	h_5	h_6
h_7	h_8	h_9

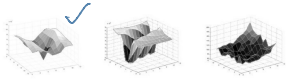
$$f \otimes h = f_1h_1 + f_2h_2 + f_3h_3 + f_4h_4 + f_5h_5 + f_6h_6 + f_7h_7 + f_8h_8 + f_9h_9$$

Corner detection

Three different cases

$E(u,v)$

$E(u,v)$



As a surface

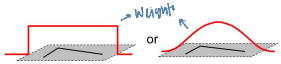
Corner Detection by Auto-correlation → fxf

Change in appearance of window $w(x,y)$ for shift $[u,v]$:

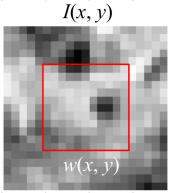
$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u, y+v) - I(x,y)\right]^2$$

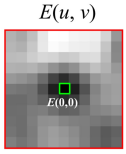
Window function | Shifted intensity | Intensity

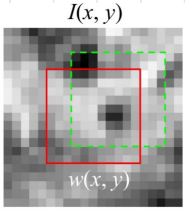
Window function $w(x,y)$ = → weights

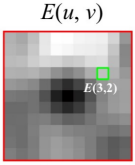


1 in window, 0 outside or Gaussian



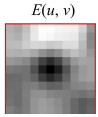






Slide Credit: Jame

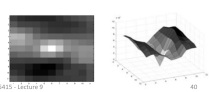
We want to find out how this function behaves for small shifts



But this is very slow to compute naively.
O(window_width^2 * shift_range^2 * image_width^2)

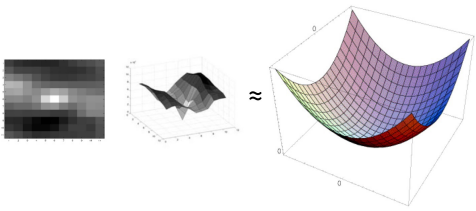
O(11^2 * 11^2 * 600^2) = 5.2 billion of these 14.6k ops per image pixel

But we know the response in E that we are looking for – strong peak.



Corner Detection: strategy

Approximate $E(u,v)$ locally by a quadratic surface



$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u,y+v) - I(x,y)\right]^2$$

The quadratic approximation simplifies to

$$E(u,v) \approx [u \;\; v] \; M \begin{bmatrix} u \\ v \end{bmatrix}$$

where M is a *second moment matrix* computed from image derivatives:

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

$$M = \begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} = \sum \begin{bmatrix} I_x \\ I_y \end{bmatrix} [I_x \; I_y] = \sum \nabla I (\nabla I)^T$$

Recall: Taylor series expansion

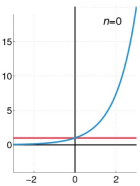
- A function f can be represented by
  - an infinite series of its derivatives at a single point *a*:

$$f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \cdots.$$

Wikipedia

As we care about window centered, we set *a = 0* (MacLaurin series)

Approximation of $f(x) = e^x$ centered at f(0)

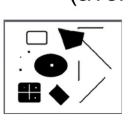


In and Iy are derivatives

Corners as distinctive interest points

$$M = \sum w(x,y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix}$$

2 x 2 matrix of image derivatives
(averaged in neighborhood of a point)



Notation:

$$I_x \Leftrightarrow \frac{\partial I}{\partial x} \qquad I_y \Leftrightarrow \frac{\partial I}{\partial y} \qquad I_x I_y \Leftrightarrow \frac{\partial I}{\partial x}\frac{\partial I}{\partial y}$$

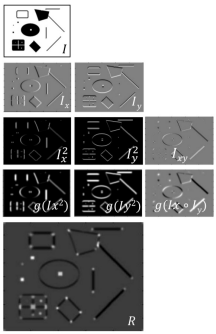
Harris corner detection

1) Compute M matrix for each window to recover a *cornerness* score C.

   Note: We can find M purely from the per-pixel image derivatives!

2) Threshold to find pixels which give large corner response $C >$ threshold.

3) Find the local maxima pixels, i.e., non-maximal suppression.



0. Input image
   We want to compute M at each pixel.

1. Compute image derivatives (optionally, blur first).

2. Compute M components as squares of derivatives.

3. Gaussian filter *g()* with width *s*

   $= g(I_x^2),\ g(I_y^2),\ g(I_x \circ I_y)$

4. Compute cornerness

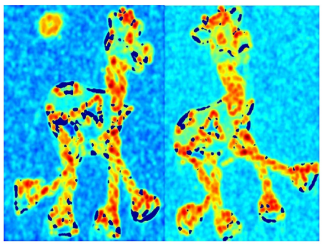
   $C = \det(M) - \alpha\ \mathrm{trace}(M)^2$

   $= g(I_x^2) \circ g(I_y^2) - g(I_x \circ I_y)^2 - \alpha[g(I_x^2) + g(I_y^2)]^2$

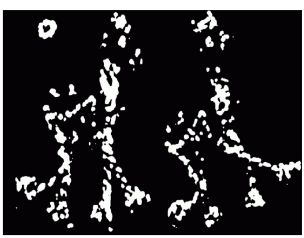
5. Threshold on C to pick high cornerness

6. Non-maximal suppression to pick peaks.

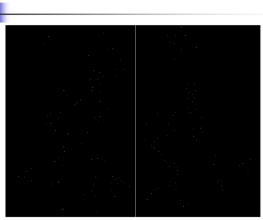
James Hays



Find points with large corner response: $R >$ threshold



Take only the points of local maxima of R



If pixel value is greater than its neighbors then it is a local maxima.