PCV

- **Image**: Defined as a 2D-function $f(x,y)$
- **Digital Image**: Image when x, y and f intensity values are finite and discrete.
- **Pixel**: Digital image is composed of finite no. of elements called pixels, each having a particular location and value
- Intensity $f(x,y) = I$
- Computer Vision involves processing of digital image for autonomous machine perception and human interpretation

— **Steps**

① **Image Acquisition**: Capturing an image using a digital camera or scanner or importing an existing image into a computer.

② **Image Enhancement**: Improving visual quality of an image (increasing contrast, reducing noise, removing artifacts)

③ **Image Restoration**: Removing degradation from an image (blurring, noise, distortion)

④ **Image Processing**: CBIR is used as basis for extracting features of interest in an image.

(5) Wavelets: Foundation for representing images in various degrees of resolution. Used for image data compression and pyramidal representation (images subdivided into smaller regions)

(6) Compression: Techniques for reducing storage required to save / store an image, or bandwidth required to transmit it.

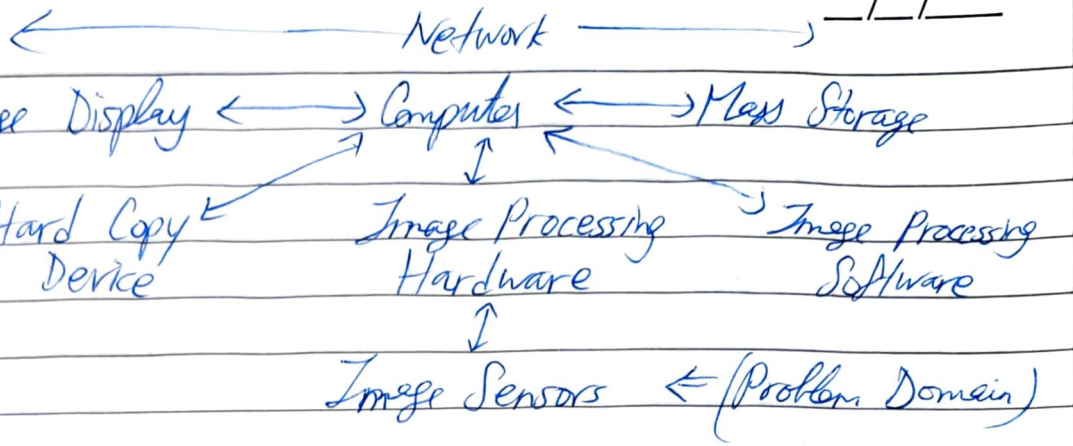
Exp JPEG (Joint Photographic Experts Group) image compression standard.

(7) Morphological parsing: It deals with tools for extracting image components that are useful in representation and description of shape

(8) Image Segmentation: Dividing image into regions or segments, each corresponding to a certain object / feature in the image.

(9) Feature Extraction: Follows output of segmentation, which is raw pixel data, either boundary of region or all points in the region. Consists of feature detection (finding features in image, region or boundary) and feature description (assigns quantitative attributes to detected features.)

(10) Image Pattern Classification: Assigns a label to an object based on its feature descriptors



- $f(x,y) = i(x,y) * r(x,y)$ $\quad (0 < i(x,y) < \infty,\ 0 < r(x,y) < 1)$
  - $f(x,y)$ ↓ image
  - $i(x,y)$ ↓ illumination
  - $r(x,y)$ ↓ reflectance

- Digitalization of an analog signal involves:

(a) Sampling: Discretization of the space

Note: Smallest element resulting from discretization of the space is called pixel (picture element)
For 3D images, it is called voxel (volumetric pixel)

(b) Quantization: Discretization of intensity values

- Intensity Range: $[0, L-1]$
  where $L = 2^k$ (k = no. of bits used to represent intensity)
- Dynamic range: $\frac{max}{min}$ ~~(3060)~~
- Image Contrast = max − min
- No. of storage bits: $b = M * N * k$

- Resolution :

Spatial Resolution : No. of pixels per unit distance (dpi → dots per inch)

Intensity Resolution : No. of bits used to quantize intensity. (k)

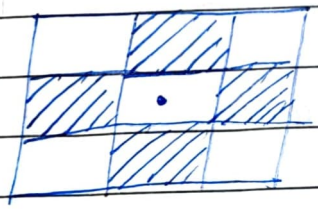
☆ Basic Relationship between pixels

- $f(x,y)$: Digital image
Pixels : q and p
Subset of pixels of $f(x,y)$: S

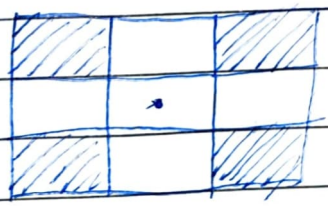
$$\begin{bmatrix} (x-1,y-1) & (x-1,y) & (x-1,y+1) \\ (x,y-1) & (x,y) & (x,y+1) \\ (x+1,y-1) & (x+1,y) & (x+1,y+1) \end{bmatrix}$$

— Neighbours

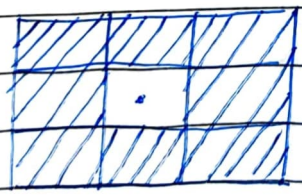
$N_4(P)$:



$N_D(P)$:



$N_8(P)$



— Adjacency

- V : set of gray-level values used to define adjacency

- 4-adjacency : q is in the set $N_4(P)$ (p,q)
- 8-adjacency : q is in the set $N_8(P)$

- m-adjacency: (i) q is in set $N_4(p)$ (or)
  (ii) q is in the set $N_D(p)$ and set $N_4(p) \cap N_4(q)$ has no pixels whose values are from V.

— Digital Path

- A path from pixel p with the coordinates (x,y) to corresponding ~~(right)~~ pixel q with coordinates (s,t) is a sequence.

- ~~Region (suggest fits an hours)~~
- Region: Subset of pixels in image if connected set
- Foreground: If an image contains 'K' disjoint regions

$$R_u = \bigcup_{K=1}^{N} R_K$$

- Background: $(R_u)^c$
- Border: Set of points that are adjacent to points that are in complement of R.
- An edge is a place in an image where intensity changes sharply (intensity discontinuity)

— Distance Measures

- For pixels p, q, z with coordinates $(x,y), (s,t), (v,w)$

$D(p,q) \geq 0$ $(D(p,q) = 0$ if $p=q)$
$D(p,q) = D(q,p)$
$D(p,z) \leq D(p,q) + D(q,z)$

- Types: For pixels p and q with coordinates (x,y) (s,t)

(a) Euclidean: $D_e(p,q) = \sqrt{(x-s)^2 + (y-t)^2}$

(b) City-Block: $D_4(p,q) = |x-s| + |y-t|$

(c) Chessboard Distance: $D_8(p,q) = \max(|x-s|, |y-t|)$

- $D_4 <= 2$

$$\begin{bmatrix} & & 2 & & \\ & 2 & 1 & 2 & \\ 2 & 1 & 0 & 1 & 2 \\ & 2 & 1 & 2 & \\ & & 2 & & \end{bmatrix}$$

$D_8 <= 2$

$$\begin{bmatrix} 2 & 2 & 2 & 2 & 2 \\ 2 & 1 & 1 & 1 & 2 \\ 2 & 1 & 0 & 1 & 2 \\ 2 & 1 & 1 & 1 & 2 \\ 2 & 2 & 2 & 2 & 2 \end{bmatrix}$$

Ex- Consider the image:

$$\begin{bmatrix} 100 & 110 & 90 & 95 \\ 98 & 140 & 145 & 135 \\ 89 & 90 & 88 & 85 \\ 102 & 105 & 99 & 115 \end{bmatrix}$$

(a) Spatial Resolution:
Matrix size: $4 \times 4 = 16$ pixels

(b) Min intensity = 85
Max intensity = 145 (Intensity resolution)
$\therefore k = 8 \quad 2^k = 256$ [0, 255]

(c) Contrast = $145 - 85 = 60$

(d) Dynamic range = $\frac{145}{85} = 1.705$

(e) $b = M \times N \times k = 4 \times 4 \times 8 = 128$ bits (16 bytes)

Ex- Find the shortest path (4, 8, m) between p and q for $V = \{0, 1\}$ and $V = \{1, 2\}$

(q)

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

(p)

4-path : $V = \{0,1\}$ ✗
$V = \{1,2\}$ (Cost = 6) ✓

8-path : $V = \{0,1\}$ (Cost = 4) ✓
$V = \{1,2\}$ (Cost = 4) ✓

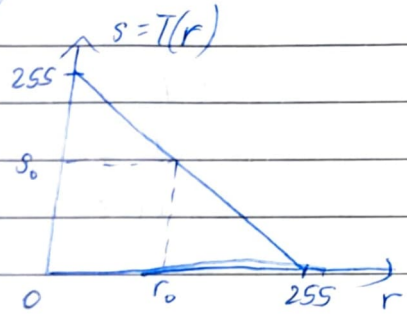
m-path : $V = \{0,1\}$ (Cost = 5)
$V = \{1,2\}$ (Cost = 6)

Intensity Transformation ($S = T(r)$)

- While intensity transformation involves single pixel operation, spatial filtering involves neighbourhood operation.

— Negation

- Image negative, $S = L - 1 - r$

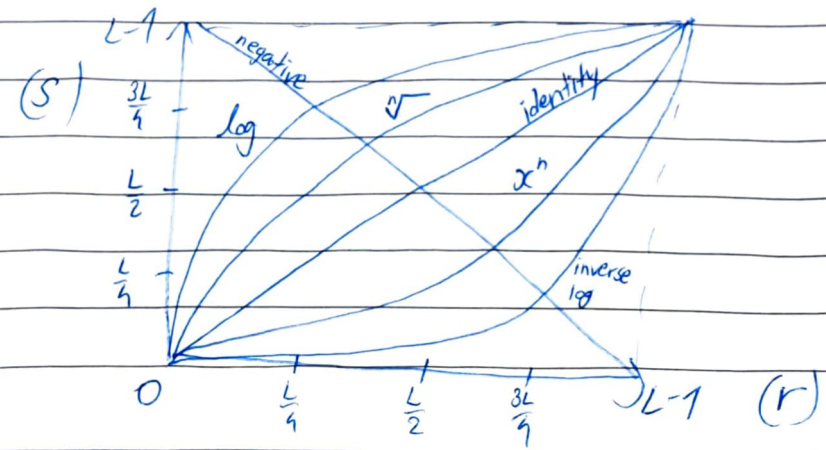


max intensity — input pixel intensity

— Log Transformation

$S = c \log(1+r)$

- Maps narrow range of lower intensity values into wider range of output values (spreading). Opposite is true for higher values of input levels (compression)



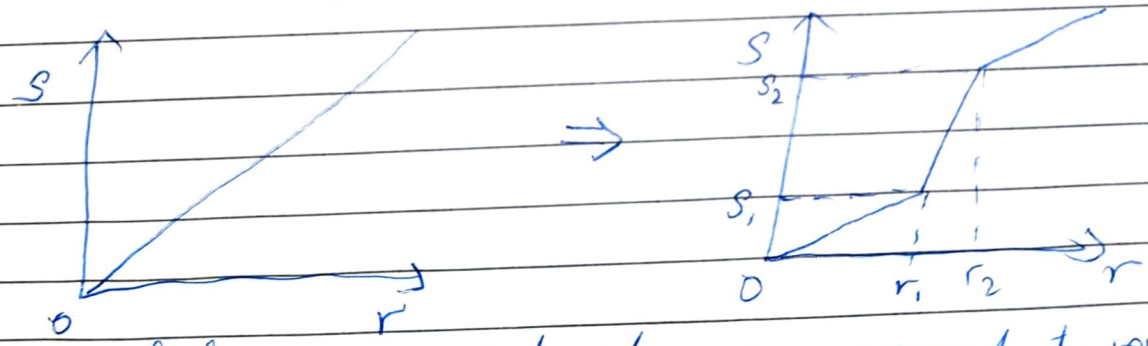
— Power-law Transformation

$s = c r^{\gamma}$

$\gamma = 1$ (ideal case)
$\gamma < 1$ (log transform)
$\gamma > 1$ (opposite)

— Contrast Stretching

- Expands range of intensity levels.
- Sometimes, most pixels fall into a narrow range of intensities, making it look dull / low contrast.
- Hence, contrast stretching redistributes intensities to utilize the full range $[0, 255]$ making the image clearer and higher contrast.



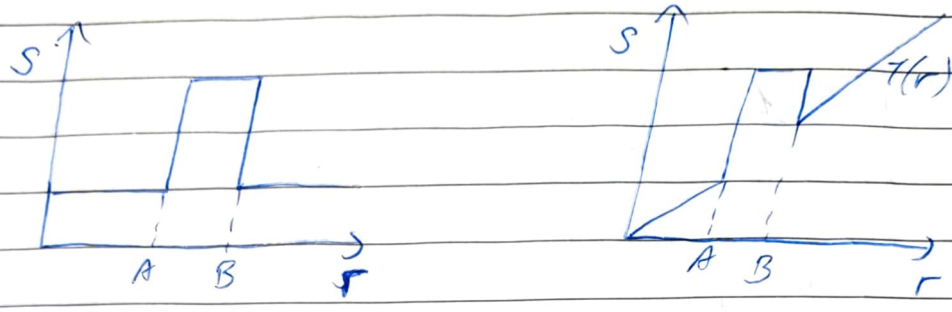
- Here below r_1, intensities are mapped to very low values (darker)
Between r_1 and r_2, intensities are stretched linearly (contrast enhanced)

$$\text{Eq: } s = \frac{(r - r_1)}{(r_2 - r_1)} \times (s_2 - s_1)$$

Above r_2, intensities flatten near maximum (brighter)

- ## Intensity - level slicing

- Highlights specific range of intensities in an image.



- ## Bit-Plane Slicing

- Decomposing an image into it's bit planes for analyzing the relative importance of each bit in the image.

$$\begin{bmatrix} 6 & 7 \\ 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 110 & 111 \\ 000 & 000 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$

MSB plane Centre bit plane LSB plane.

- ## ★ Spatial Filtering

- Mechanism: Moving filter mask from point to point in an image

- Operation: Weighted sum of input pixels

- A filtered image is generated as the centre of the mask moves to every pixel in the input image.

$$g(x,y) = T[f(x,y)]$$

$$R = w_1 z_1 + w_2 z_2 + \dots = \sum w_i z_i$$

(w_i: weight on mask; z_i: pixel value (input))

Ex: Use the following 3×3 mask to filter the shaded pixels in the given 5×5 image.

$$\begin{bmatrix} 0 & 1/6 & 0 \\ 1/6 & 1/3 & 1/6 \\ 0 & 1/6 & 0 \end{bmatrix}$$

(Mask)

80	40	50	70	90
40	50	80	60	100
35	255	70	0	120
30	45	80	100	130
40	50	90	125	140

(Image)

50: $0\times 80 + 40\times\frac{1}{6} + 50\times 0 + 40\times\frac{1}{6} + 50\times\frac{1}{3} + 80\times\frac{1}{6} +$

$0\times 35 + 255\times\frac{1}{6} + 70\times 0 = 85$

⋮

70: $0\times 50 + \frac{1}{6}\times 80 + 0\times 60 + \frac{1}{6}\times 255 + \frac{1}{3}\times 70 + 0\times\frac{1}{6} +$

$0\times 45 + \frac{1}{6}\times 80 + 0\times 100 = 92$

⋮

Final image:

80	40	50	70	90
40	85	65	61	100
35	118	92	58	120
30	84	77	89	130
40	50	90	125	140

- Types

- Method: ① Correlation (moving mask over image)
  ② Convolution (mask rotated 180°)

- Filter: ① Smoothing (low-pass)
  ② Sharpening (high-pass)

- 1D Correlation & Convolution

- Input: 0 0 0 1 0 0 0 0 (origin ↙)
  Filter/Mask: 1 2 3 2 8

- Align origin of input with LSB of filter ~~and add zero padding~~
  Then apply zero-padding on input.
  Then multiply corresponding bits and sum → output bit in each shift.

padding | origin | padding

(I/P) 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0

(Mask) 1 2 3 2 8 0 0 0 0 0 0 0 0 0 0 0 ⇒ 0

0 1 2 3 2 8 0 0 0 0 0 0 0 0 0 0 ⇒ 0

0 0 1 2 3 2 8 0 0 0 0 0 0 0 0 0 ⇒ 0

0 0 0 1 2 3 2 8 0 0 0 0 0 0 0 0 ⇒ 8

0 0 0 0 1 2 3 2 8 0 0 0 0 0 0 0 ⇒ 2

0 0 0 0 0 1 2 3 2 8 0 0 0 0 0 0 ⇒ 3

0 0 0 0 0 0 1 2 3 2 8 0 0 0 0 0 ⇒ 2

0 0 0 0 0 0 0 1 2 3 2 8 0 0 0 0 ⇒ 1

0 0 0 0 0 0 0 0 1 2 3 2 8 0 0 0 ⇒ 0

0 0 0 0 0 0 0 0 0 1 2 3 2 8 0 0 ⇒ 0

0 0 0 0 0 0 0 0 0 0 1 2 3 2 8 0 ⇒ 0

0 0 0 0 0 0 0 0 0 0 0 1 2 3 2 8 ⇒ 0

- In convolution, the filter is just reversed and same procedure is followed

— Smoothing Spatial Filters (Low-pass)

- Used for blurring and noise reduction.

(a) Smoothing Linear Filters (Mean)

$$\frac{1}{9} \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad \frac{1}{16} \times \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

(standard average) (weighted average)

$$g(x,y) = \frac{\sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t)\, f(x+s, y+t)}{\sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t)}$$

(General implementation of weighted average filtering)

~~(b) Order~~

Ex:- (I/P)

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 10 & 20 & 30 & 0 \\ 0 & 40 & 50 & 60 & 0 \\ 0 & 70 & 80 & 90 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$ (with padding)

$$\text{Filter} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad \sum\sum w(s,t) = 9$$

10: $g(1,1) = \frac{\text{Sum of surrounding intensities + self} \times w(1,1)}{9}$

$= \frac{(10+20+40+50)\times 1}{9} = 13.33 \simeq 13$

__/__/__

20: $g(1,2) = \frac{10+20+30+40+50+60}{9} = 23.33 \simeq 23$

30: $g(1,3) = \frac{20+50+60+30}{9} = 17.78 \simeq 18$

40: $g(2,1) = \frac{10+20+40+50+70+80}{9} = 30$

50: $g(2,2) = \frac{10+20+30+40+50+60+70+80+90}{9} = 50$

60: $g(2,3) = \frac{20+30+50+60+80+90}{9} = 36.67 \simeq 37$

70: $g(3,1) = \frac{40+50+70+80}{9} = 26.67 \simeq 27$

80: $g(3,2) = \frac{40+50+60+70+80+90}{9} = 43.34 \simeq 43$

90: $g(3,3) = \frac{50+60+80+90}{9} = 31.12 \simeq 31$

Filtered image:

$$\begin{bmatrix} 13 & 23 & 18 \\ 30 & 50 & 37 \\ 27 & 43 & 31 \end{bmatrix}$$

- As a result, the final image looks smoother, less noisy and also blurrier, thus losing detail. Salt-and-pepper noise and sharp edges (high-frequency variations) get reduced.

(b) Order-Statistics Filter (Median)

- Response is based on ordering (ranking) the pixels encompassed by the filter and replacing centre pixel with median value.

- Removes salt-and-pepper noise.

Ex:

10	15	20
20	100	20
20	20	25

→ 20

10, 15, 20, 20, (20), 20, 20, 25, 100

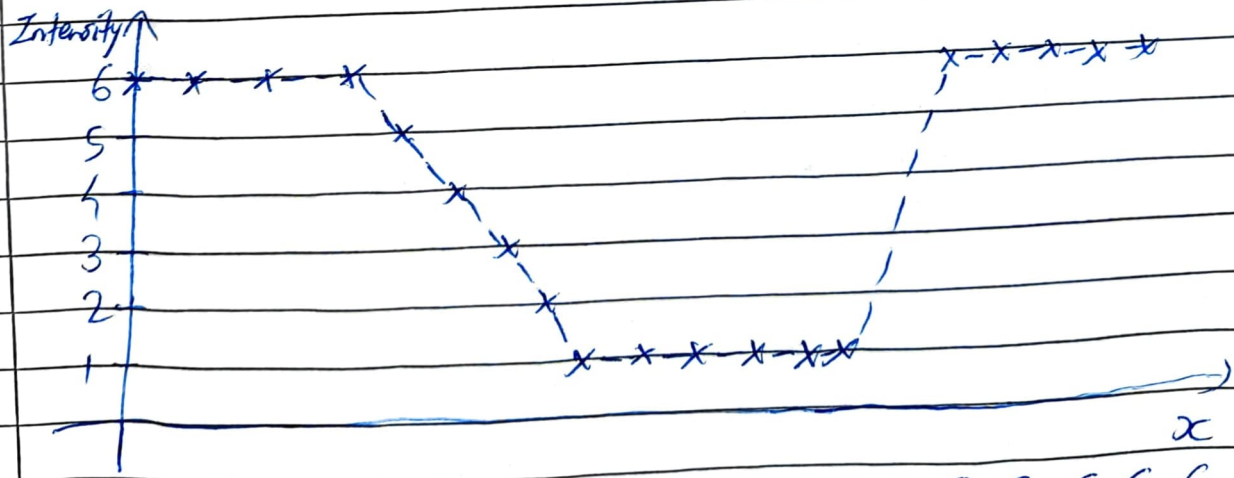
— Sharpening Filters (High-pass)

- For enhancing intensity transitions (borders, edges)
- Intensity transitions between adjacent pixels are related to derivatives of the image
- Since an image is a discrete function, traditional definition of derivation cannot be applied

$$\frac{\partial f}{\partial x} = f(x+1) - f(x)$$

$$\frac{\partial^2 f}{\partial x^2} = f(x+1) - 2f(x) + f(x-1)$$

Ex:



Scan:	6	6	6	6	5	4	3	2	1	1	1	1	1	6	6	6	6
First:	0	0	0	-1	-1	-1	-1	-1	0	0	0	0	5	0	0	0	0
Second:		0	0	-1	0	0	0	0	1	0	0	0	5	-5	0	0	0

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

$\frac{\partial^2 f}{\partial x^2} = f(x+1,y) - 2f(x,y) + f(x-1,y)$

$\frac{\partial^2 f}{\partial y^2} = f(y+1,x) - 2f(y,x) + f(y-1,x)$

~~$\nabla^2 f = f(x+1) + f(y+1) + f(x-1)$~~

$$\nabla^2 f = f(x+1,y) + f(x-1,y) + f(x,y-1) + f(x,y+1) - 4f(x,y)$$

(4-neighbors Laplacian)

- Including diagonals, 8-neighbors Laplacian is:

$$\nabla^2 f = f(x+1,y) + f(x-1,y) + f(x,y-1) + f(x,y+1) + f(x+1,y+1) + f(x+1,y-1) + f(x-1,y+1) + f(x-1,y-1) - 8f(x,y)$$

- (N_4 Laplacian)

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

(8-neighbors N_8 Laplacian)

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

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

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

- After applying $\nabla^2 f$ formula on each pixel in the image that matrix is subtracted from the original image to obtain the sharpened image:

$$f_{sharp}(x,y) = f(x,y) - \alpha \nabla^2 f(x,y)$$

where α controls strength $[0.2, 0.8]$

If $\nabla^2 f$ is (−)ve, subtracting boosts the pixel intensity
(+)ve, subtracting reduces the pixel intensity

- **Unsharp Masking:**

$$g_{mask}(x,y) = f(x,y) - \bar{f}(x,y) \quad \bar{f}: \text{blurred image}$$

- **High-boost Filtering:**

$$g(x,y) = f(x,y) + k^* g_{mask}(x,y) \qquad (k > 1)$$

- **Gradient:**

$$\nabla f = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

$$\nabla f \approx |G_x| + |G_y|$$
$$= \sqrt{(G_x)^2 + (G_y)^2}$$

- **Robert's operator:**

(G_x): $\begin{bmatrix} +1 & 0 \\ 0 & -1 \end{bmatrix}$ (G_y): $\begin{bmatrix} 0 & +1 \\ -1 & 0 \end{bmatrix}$

- **Sobel operator:**

$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$ $\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$

$\begin{bmatrix} z_1 & z_2 & z_3 \\ z_4 & z_5 & z_6 \\ z_7 & z_8 & z_9 \end{bmatrix}$

$$R: \sqrt{(z_5 - z_9)^2 + (z_6 - z_8)^2}$$
$$\approx |z_5 - z_9| + |z_6 - z_8|$$

(Detects diagonal edges)

$$S: \nabla f = \sqrt{(z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)|^2 + |(z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)|^2}$$

Note: In 1D, derivative ~~gives~~ tells how fast intensity changes.
In 2D, gradient vector gives edge strength (magnitude) and edge orientation: $\theta = \tan^{-1}\left(\frac{G_y}{G_x}\right)$

While Robert's detects diagonal edges, Sobel detects horizontal and vertical edges while reducing noise. Canny is a more advanced operator for better results.

☆ Histogram Processing

- Histogram is a plot that shows how frequently each intensity value occurs in an image.
- $h(r_k) = n_k$ → k^{th} gray level $\quad k \in [0, L-1]$ for image of size $M \times N$
- Normalized histogram, $p(r_k) = \frac{n_k}{MN}$
- Shape of histogram provides useful information for contrast enhancement.

– Histogram Equalization

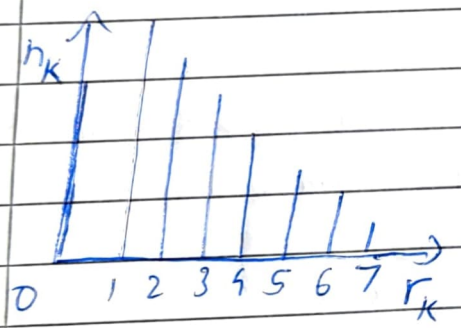
- Spreads histogram of input image so that intensity levels span a wider range of intensity scale.

$$s_k = T(r_k) = (L-1)\sum_{j=0}^{k} p_r(r_j)$$

$$\left(s_k = \frac{L-1}{MN}\sum_{j=0}^{k} n_j\right)$$

Exa: Suppose 3-bit image ($L=8$) of size $64 \times 64 = 4096$ px

r_k	n_k	$p_r(r_k) = n_k/MN$	s_k
$r_0 = 0$	790	0.19	1
$r_1 = 1$	1023	0.25	3
$r_2 = 2$	850	0.21	5
$r_3 = 3$	656	0.16	6
$r_4 = 4$	329	0.08	6
$r_5 = 5$	245	0.06	7
$r_6 = 6$	122	0.03	7
$r_7 = 7$	81	0.02	7



$$s_0 = 7 \times \sum_{j=0}^{0} p(r_j)$$

$$= 7 \times 0.19 = 1.33 \simeq 1$$

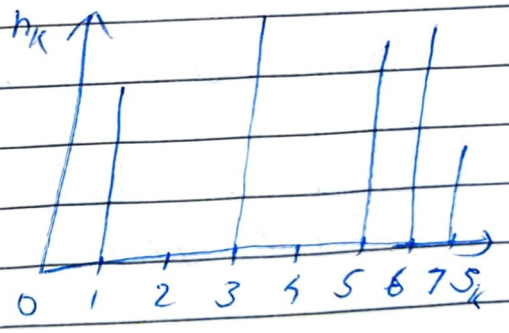
$$s_1 = 7 \times (0.19 + 0.25) = 3.08 \simeq 3$$

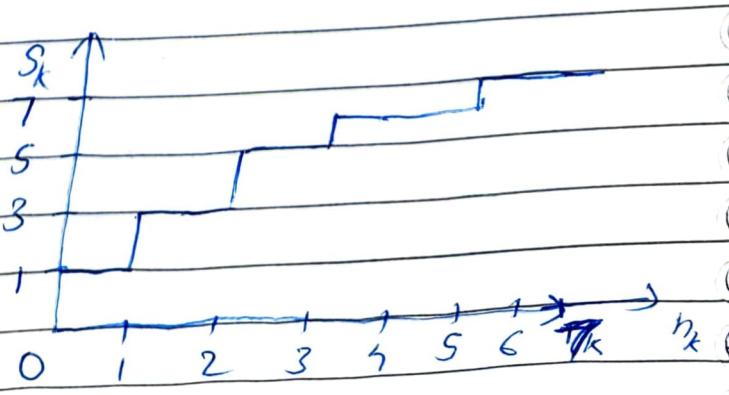
$$s_2 = 7 \times (0.19 + 0.25 + 0.21)$$

$$= 4.55 \simeq 5$$

Hence,

s_k	n_k
$s_0 = 0$	0
$s_1 = 1$	790
$s_2 = 2$	0
$s_3 = 3$	1023
$s_4 = 4$	0
$s_5 = 5$	850
$s_6 = 6$	985
$s_7 = 7$	448





Histogram Specification / Matching

① Obtain $p_r(r_j)$ and s_k from input image.

② Use specified PDF and obtain transformation function $G(z_q)$

$$G(z_q) = (L-1)\sum_{i=0}^{q} p_z(z_i) = s_k$$

③ Mapping from s_k to z_q : $z_q = G^{-1}(s_k)$

Ex: From previous example,

	z_q	$p_z(z_q)$	s_k	$G(z_q)$	CDF(z)
(Specified)	$z_0 = 0$	0.00	$s_0 = 1$	0	0
	$z_1 = 1$	0.00	$s_1 = 3$	0	0
	$z_2 = 2$	0.00	$s_2 = 5$	0	0
	$z_3 = 3$	0.15	$s_3 = 6$	1	0.15
	$z_4 = 4$	0.20	$s_4 = 6$	2	0.35
	$z_5 = 5$	0.30	$s_5 = 7$	5	0.65
	$z_6 = 6$	0.20	$s_6 = 7$	6	0.85
	$z_7 = 7$	0.15	$s_7 = 7$	7	1

$$G(z_0) = 7\sum_{i=0}^{0} p_z(z_i) = 7 \times 0.00 = 0$$

$$G(z_1) = 7 \times (0.00 + 0.00) = 0$$

⋮

r_k	s_k	z_q	$p_r(r_k)$	(CDF) $p_z(z_q)$
0	1	3	0.19	0.15
1	3	4	0.25	0.35
2	5	5	0.21	0.65
3	6	6	0.16	0.85
4	6	6	0.08	0.95
5	7	7	0.06	1
6	7	7	0.04	1
7	7	7	0.02	1

FEATURE DETECTION & MATCHING

★ Detection of Discontinuities

- Most common way to look for discontinuities is to scan a mask across the image.
- For an image having pixel intensities $z_1, z_2, \ldots, z_i$ and mask $= \begin{bmatrix} w_1 & w_2 & w_3 \\ w_4 & w_5 & w_6 \\ w_7 & w_8 & w_9 \end{bmatrix}$

Result, $R = w_1 z_1 + w_2 z_2 + \ldots = \sum w_i z_i$

– Point Detection

- Finding pixels that are very different from surrounding pixels (isolated)

$|R| \geq T$ (threshold, what counts as a discontinuity)

$$\text{Mask} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

– Line Detection (one-pixel-wide line)

$$\begin{bmatrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{bmatrix} \quad \begin{bmatrix} -1 & -1 & 2 \\ -1 & 2 & -1 \\ 2 & -1 & -1 \end{bmatrix} \quad \begin{bmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{bmatrix} \quad \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix}$$

(Horizontal) (+45°) (Vertical) (−45°)

— ⟋ | ⟍

- These are directional filters designed to detect specific orientation of lines.

— Edge Detection (2nd)

- Used the 2nd-derivative to detect regions of rapid intensity changes (edges) (boundaries between two diff regions)

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \quad \text{(Laplacian)}$$

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \qquad \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \text{(including diagonals)}$$

— Roberts Cross-Gradient Operators (detects diagonal edges) (using 1st derivative)

$$G_x = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}, \quad G_y = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \qquad \left(G = \sqrt{G_x^2 + G_y^2} \right)$$

- Detects fine edges only, not smooth / thick ones and due to small size, highly sensitive to noise.

— Prewitt Operator (1st)

- Detects edges in horizontal & vertical directions.

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \text{(V)} \qquad G_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \text{(H)}$$

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

(Smoother, less sensitive to noise, equal weights)

– Sobel Operator (1st)

- Gives more weight to center pixels, better noise suppression

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

(vertical edges, change in horizontal) (horizontal edge, change in vertical edges)

☆ Thresholding

- Seperating objects from the background in an image

$$g(x,y) = \begin{cases} 1, & \text{if } f(x,y) > T \text{ (object pixel)} \\ 0, & \text{if } f(x,y) \leq T \text{ (background pixel)} \end{cases}$$

Ex: Apply thresholding with $T = 4$

$$\begin{bmatrix} 1 & 1 & 2 & 2 & 3 \\ 1 & 1 & 2 & 2 & 8 \\ 1 & 1 & 2 & 7 & 8 \\ 1 & 1 & 6 & 7 & 8 \\ 1 & 5 & 6 & 7 & 8 \end{bmatrix} \Rightarrow \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

– Basic Global Thresholding

- To find the right threshold without finalizing beforehand.

① Select an initial T_0 (average gray level / middle $\frac{L}{2}$)

② Segment image into two groups like in $g(x,y)$
$G_1 (>T)$, $G_2 (\leq T)$

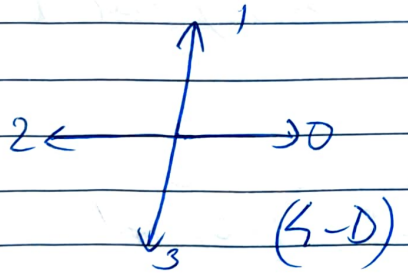
③ Compute average intensity m_1 and m_2 for the pixels in G_1 and G_2

④ Compute a new threshold, $T_{new} = \frac{m_1 + m_2}{2}$

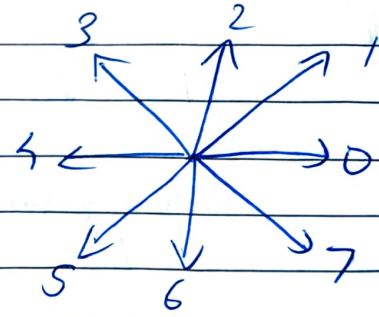
⑤ Check for convergence, $|T_{new} - T_{old}| <$ some small value than quit (after certain iterations)

★ Chain Codes

- Way to represent the shape of an object's boundary using numbers that describe direction between connected pixels (to describe the outline / contour in digital image)
- These are generated by following a boundary in CLOCKWISE direction



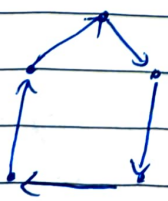
(4-D)



(8-D)

- Limitations: • Chain code tends to be long
  • Noise, tiny bumps, irregularities can cause changes in the code.

Ex:



Chain Codes = 76421, 64217, 42176, 21764, 17642 ← smallest value (normalized)

(since it is hard to choose the right start point)

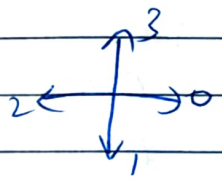
- To normalize rotation, since if shape rotates the direction numbers change, we take the first difference of the chain code. (how directions change)

$$d(i) = [c(i+1) - c(i)] \bmod N \quad \text{(4/8 direction)}$$

This way the code becomes rotation-invariant where only turning pattern matters.

~~Ex: Original = 0 1 0 3 3 2 2 2 3 0~~

~~Final~~



Ex: Original = 0 1 2 3
Rotated = 1 2 3 0 (90°)

$d = (1-0,\ 2-1,\ 3-2,\ 0-3) \bmod 4$
$= (1, 1, 1, 1)$

Now, rotate again → 2 3 0 1
$d = (2-1,\ 3-2,\ 0-3,\ 1-0) \bmod 4$
$= (1, 1, 1, 1)$ (Same)

Note: $-3 \bmod 4 \Rightarrow (-3+4) \bmod 4,\ 1 \bmod 4 = 1$

★ Texture

- Describes smoothness, coarseness and regularity of region, made of sub-elements arranged in patterns
- Goal of texture analysis is to ~~find~~ compare textures to see if they are made of the same stuff (repeated patterns of brightness / color)

☆ Co-occurrence Matrix

- Describes texture by relationships between neighbouring pixels, by counting how often two unique gray levels i, j occur next to each other in a given distance / direction.

- For an image having N gray-level values, the co-occurrence matrix has size N×N which is computed based on (a) relative distance (number of pixel's away/gap) and (b) relative orientation/direction.

Ex: Find no. of co-occurrence of pixel i to neighbouring pixel j ($\theta = 0°$, $N=4$, $d=1$)

$$\begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 2 & 2 & 2 \\ 2 & 2 & 3 & 3 \end{bmatrix} \Rightarrow \begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ 0 & 2 & 2 & 1 & 0 \\ 1 & 0 & 2 & 0 & 0 \\ 2 & 0 & 0 & 3 & 1 \\ 3 & 0 & 0 & 0 & 1 \end{array}$$

☆ Canny Edge Detection

- Compared to Sobel and Prewitt, this edge detection tool is able to achieve:

(a) Good Detection: Minimize the probability of FP (wrong edges) and FN (missing real edges)

(b) Good Localization: Detect edges as close as possible to real edges.

(c) Minimal Response: Only detect an edge once, no multiple edge responses near each other

by combining gradient-based edge detection with noise reduction and edge refinement techniques.

① Image Smoothing using Gaussian Filter (remove noise)

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

R: smoothed image, I: input image, Gaussian kernel

- For a 5×5 Gaussian kernel with $\sigma = 1.5$ (S.D)

$$g(x,y) = \frac{1}{159} \begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

- Here, center pixel gets the highest weight, neighbour's weight lowers depending on distance (smooth averaging)
  $\sigma \rightarrow$ controls degree of smoothing

② Gradient Calculation (find edges)

- Smoothened image is then filtered with a Sobel kernel in both X and Y direction to get derivatives G_x and G_y

$$\text{Edge Gradient, } G = \sqrt{G_x^2 + G_y^2} \quad \text{(edge strength)}$$

$$\text{Gradient Angle, } \theta = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

- Gradient → rate of change of intensity in an image

③ Non-maximum suppression (thinning)

- After calculating gradient, we get an image of edge strength (blurry and thick edges) and an image showing direction of edge

- Now, we look along the gradients direction and compare the current pixel's gradient strength with its neighbours along the same direction. If it is not the biggest, it is not the center of the edge, suppress → 0. Else, keep it (local maxima)

- for each pixel (i,j):

  if $M(i,j) < M(i_1,j_1)$ or $M(i,j) < M(i_2,j_2)$

  then $I_{NEW}(i,j) = 0$

  else

  $I_{NEW}(i,j) = M(i,j)$

④ Double Thresholding

- Some edges are ~~weak~~ strong (real), while others are weak (noise/shadows). Hence, we need to filter them

- However, instead of using one fixed threshold, Canny uses two; T_H (high) and T_L (low) such that

  if $M_G \geq T_H$:

  then Strong edge (real) ✓ (M_G = gradient magnitude)

  elif $T_L \leq M_G < T_H$

  then Weak edge (depends on connection to strong edges)

  ~~dif~~ $M_G < T_L$

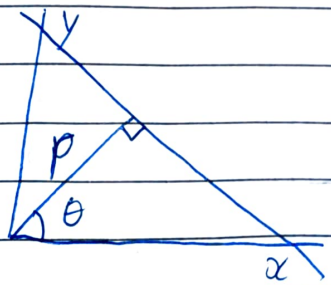
  then definitely not an edge ✗

(5) Edge Tracking by Hysteresis

- Weak edges might be either part of a real edge but faint due to lighting/texture, or just random noise
- If any weak edge pixel is connected (8-neighbours) to a strong edge pixel, then keep it, else suppress.

— Hough Transform

- Detects straight lines in image using polar coordinate system.



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

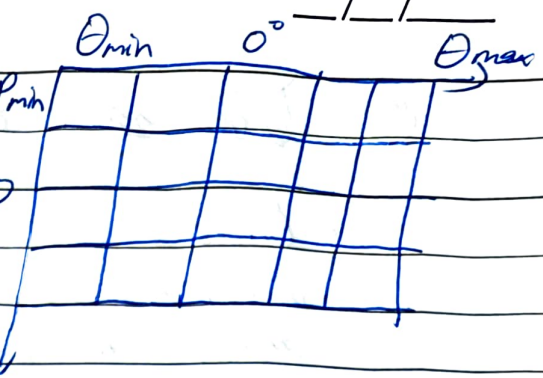
↳ ⊥ distance of line from origin

θ → angle between x-axis and the perpendicular

- After running Canny edge detection, we get only edge pixels scattered around, hence Hough transform is used to group these points together to form a line/curve.
- For each edge pixel (x, y) we try all values of θ and compute ρ values from origin ⇒ (ρ, θ) pairs
- For a given set of ~~points~~ pixels through majority voting, we can obtain a common line that passes through all points and conforms the right (ρ, θ) value.

— Accumulator array/matrix

- Each element of the matrix tells no. of points/pixels that are positioned on the line represented by (ρ_i, θ_i) (collinear points)



- Range of θ is $\pm 90^\circ$

Range of ρ is $\pm \frac{N}{2}$

if image is of size $(N \times N)$

★ Feature Extraction (SIFT)

- SIFT → Scale Invariant Feature Transform
Goal is to find keypoints (important features) in the image that are detectable under different scales, lighting conditions and rotations.

① Scale-Space Peak Selection

- To find potential locations of features (keypoints) by building a scale-space (collection of progressively blurred images) invariant to scale, suppressing noise
- Blurring is done using Gaussian filter with increasing σ value (blur factor)
- Scale-space is built in octaves (groups of progressively smaller images)

$[N \times N, \sigma \uparrow] \rightarrow [N/2 \times N/2, \sigma \uparrow]$ each generating multiple blurred versions of an image

Octave 1 → Octave 2

Small details → Large features

② Difference of Gaussian (DoG)

- Edges, corners, etc. are points where intensity changes sharply. If you take highly blurred version of the image and subtract with a less-blurred one, in each octave, you get high values only at places that changed.

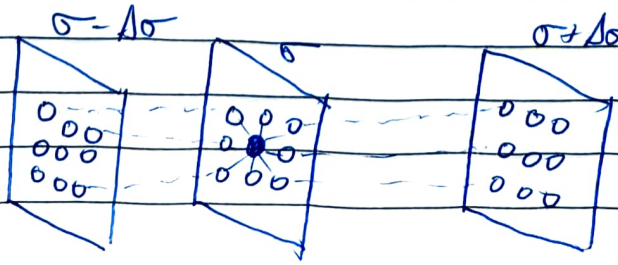
$$DoG = I(x, y, k\sigma) - I(x, y, \sigma) \quad \text{(same octave)}$$

where $k \rightarrow$ multiplies between successive sigma changes

- Hence, we can compute DoG images within each octave to highlight regions of rapid intensity change (potential keypoints)

③ Keypoint Localization

- Consider the DoG image as 3D data: two spatial dimensions (x, y) and one scale dimension (σ) in each octave
- For each point in that DoG scale space, we compare it's intensity with 8-neighbours in the same image (σ), 9-neighbours in image of scale $(\sigma + \Delta\sigma)$ and 9-neighbours in image of scale $(\sigma - \Delta\sigma)$ (26 neighbours)
- If $D(x, y, \sigma) >$ all 26 neighbours $\rightarrow$ local maxima (brighter)
  Else local minima



Here $\Delta\sigma \rightarrow$ change in sigma in same octave

- These minima / maxima → legit keypoints left represented in that scale (scale invariant)
  (size)
- However, not all local extremum are good features, some can be noise, low-contrast, ~~not edges~~
  To filter them, apply thresholds to remove low-contrast points, eliminate keypoints lying on edges (not corners)

③ Orientation Assignment

- If image rotates, the gradient directions will shift around that keypoint
- SIFT assigns each keypoint a dominant orientation to become (orientation invariant)
- Take the neighbourhood pixels of the keypoint on the image blurred with the σ value with which we found that keypoint, and Gaussian blur it
- Compute gradient ~~about~~ across that keypoint in that patch (how much intensity changes across x-direction and y-direction) (and orientation in which edge is pointing)

Eg:

$$\begin{bmatrix} 121 & 10 & 78 & 96 \\ 48 & 152 & 64 & 125 \\ 145 & 78 & \boxed{85} & 89 \\ 154 & 214 & 86 & 200 \end{bmatrix}$$

For pixel [85],

$G_x = 89 - 78 = 11$

$G_y = 84 - 86 = 8$

$G = \sqrt{G_x^2 + G_y^2} = 13.6$

$\theta = \tan^{-1}\left(\frac{G_y}{G_x}\right) = 36°$

- Build a histogram of gradient orientations (36 bins 360/10)

X-axis : Orientation, Y-axis : magnitude sum

We plot the histogram for all neighbouring pixels of that keypoint. The x-axis has ranges of orientation (θ) and y-axis tells sum of magnitudes of pixels that fall under that bin.

- The largest bin orientation (average within that bin) is assigned as the dominant orientation.

(5) Keypoint Descriptor

- Now that we know the keypoint's location, scale and orientation, now build a descriptor for SIFT to become rotation invariant.

- Take the 16×16 window around the keypoint, divide into 16 4×4 blocks, for each 4×4 block build their own histogram (8-bin) just like (3)

- Now gradient directions of pixels are relative to keypoint orientation (descriptor rotates with keypoint, rotation invariant)

- Combine all histograms, 16 × 8-bins = 128 value SIFT descriptor vector.

- This vector will get used for matching between images

(6) Keypoint Matching

- Keypoints between 2 images are matched by identifying their nearest neighbours.

__/__/__

Mean-Shift Segmentation

- Unsupervised learning algorithm used for clustering where every data point is shifted to their regional mean and the location of the final destination of each point represents the cluster it belongs to.

- Unlike K-means where no. of clusters must be specified in advance, Mean-Shift learns K from data.

- Given: Distribution of N pixels in feature space
  Task: Find modes (clusters) of distribution.

① Set $m_i = f_i$ as initial mean (feature value) for each pixel i

② For each mean m_i
   (a) Place window of size W around m_i
   (b) Compute centroid m within the window, Set $m_i = m$
   (c) Stop if shift is tiny $|m_{i,new} - m_{i,old}| < \varepsilon$

③ After all pixels are shifted and each has their own mode, pixels with same mode are in same cluster.

Note: Centroid → average of all neighbours.

Cons:
- Computationally expensive ($O(n^2)$)
- Sensitive to choice of kernel, radius of kernel

Pros:
- Does not require no. of clusters specified beforehand
- Can handle arbitrary shapes and sizes of clusters

Ques Bandwidth, $h = 2$

Points = $(2,3), (3,4), (4,5), (5,5), (8,8)$

① For point $(2,3)$, → $(2,3)$, $d = 0$
→ $(3,4)$, $d = 1.41$ (euclidean)
→ $(4,5)$, $d = 2.83$
→ $(5,5)$, $d = 3.61$
→ $(8,8)$, $d = 7.81$

Neighbours = $(2,3), (3,4)$ (within h)

New point = $\left(\frac{2+3}{2}, \frac{3+4}{2}\right)$, $(2.5, 3.5) \approx (3,4)$

② For point $(3,4)$

Neighbours within h = $(2,3), (3,4), (4,5)$

New point = $\left(\frac{2+3+4}{3}, \frac{3+4+5}{3}\right) = (3,4)$

③ For point $(4,5)$

Neighbours within h = $(3,4), (4,5), (5,5)$

New point = $(4,5)$

④ For point $(5,5)$, Neighbours = $(4,5), (5,5)$

New point = $(4.5, 5)$

⑤ For point $(8,8)$, Neighbours = $(8,8)$

Newpoint = $(8,8)$

∴ Cluster 1 center = $(3,4)$

Cluster 2 center = $(4.5, 5)$

Cluster 3 center = $(8,8)$

MOTION TRACKING

* Object Tracking

- Locating and following a specific object across a sequence of frames in a video.
- The tracker must keep a consistent identity for the object even as appearance, scale, lighting change.
- Real-world use cases include surveillance and crowd analysis, healthcare (motion of organs, patient monitoring) sports analysis, autonomous vehicle

— Motion Estimation

- Algorithm that estimate how pixels move between consecutive frames. Key components are:

(a) Error metric (to measure match quality) (SSD SAD)
(b) Search technique (to search for best match) (full search, hierachial/pyramids, ~~incremental~~ incremental)
(c) Models : Dense (every pixel) or sparse (features)

Dense	Sparse
produce a motion vector for every pixel. useful for optical flow, video restoration.	track a set of good keypoints (corner)
More computationally expensive and not good for textureless regions	Cheaper and robust for long-term object tracking (KLT)

Translational Alignment

- Here, the entire image moves. We assume that all pixels move by some amount and all pixel values are same in both images.

- Say we have two images I_0 and I_1 (shifted). We are trying to find out the best Displacement vector (u, v) such that if a pixel was at (x, y) in I_0 and at $(x+u, y+v)$ in I_1, the difference at each pixel (error) is minimal.

- $$E_{SSD}(u,v) = \sum [I_1(x+u, y+v) - I_0(x,y)]^2$$

If images match well (when I_1 is shifted back by (u,v)) SSD is small, else errors will be large.
So we try different shifts (u,v) until we get the best alignment (least errors)

- To ignore less important pixels in the image like outside boundary or moving objects in a static background we assign each pixel a weight to give more importance to pixels part of static background / center of image.

$$E_{WSSD}(u,v) = \sum w_0(x_i, y_i)\, w_1(x_i+u, y_i+v) \left[I_1(x_i+u, y_i+v) - I_0(x_i, y_i)\right]^2$$

$w_0 \rightarrow$ weight of pixel in image I_0
$w_1 \rightarrow$ weight for pixel in image I_1

Note: SSD is not very robust, large errors dominate (sum of squared differences)

- A more robust error metric is SAD (absolute difference)

$$E_{SAD}(u,v) = \sum |I_1(x_i+u, y_i+v) - I_0(x_i, y_i)|$$

Here large errors don't dominate as much

- Another way of analyzing how different two images are is cross-correlation (to maximize similarity)

$$E_{cc}(u,v) = \sum I_0(x_i, y_i) I_1(x_i+u, y_i+v)$$

When patterns line up perfectly, high values at same place, product ↑, sum ↑
Otherwise, sum ↓
Maximum correlation occurs when images are best aligned

$$E_{SSD}(u,v) = \sum [I_0(x,y) - I_1(x+u, y+v)]^2$$
$$= \sum I_0^2 + \sum I_1^2 - 2\sum I_0(x,y) I_1(x+u, y+v)$$
$$= \sum I_0^2 + \sum I_1^2 - 2E_{cc}(u,v)$$

— Hierachial Motion Estimation

(for large motion, far-away)

- To minimize error metric, since search through the entire image is simple to implement but slow, to accelerate the process:

① Construct image pyramid by reducing resolution by 2× at each step

512×512 ⇒ 256×256 ⇒ 128×128 ⇒ 64×64
Down sampling is done by taking mean / Gaussian blur on a certain set of pixels

(2) At the thinnest level, search for the best displacement vectors (u, v) using basic motion estimation (SSD)

(3) Upscale (predict motion for next level, clearer image)

Since at the next level, image is 2x larger in width and height, we assume the displacement vectors to be:

$$(u_{new}, v_{new}) = 2 \times (u_{old}, v_{old})$$

(4) Search locally at a finer level around the assumed displacement vectors range instead of the full range

$$(u_{new}, v_{new}) \pm 1$$

(5) Keep repeating till full resolution / original size is reached.

- Learned Motion Models

- Learning the motion parameters wrt the particular application using a set of training videos
- For the new test sequence, a novel parameter is computed using coarse-to-fine algorithm.

☆ Optical Flow (per pixel)

- Visible motion of an object in an image, apparent flow of pixels in an image.
- Used for estimation of object velocity and position of object in the next frame

- However real motion may or may not give rise to optical flow; for a spinning sphere with a stationary light source, motion field exists but no optical flow, on the other hand, for a stationary sphere with a moving light source, no motion exists but optical flow is there.

- Optical flow is intuitively the apparent motion of brightness patterns between consecutive frames $(t, t+\Delta t)$ (time)
(intensity motion)

- We assume a moving point keeps the same brightness

$$I(x+u, y+v, t+\Delta t) = I(x, y, t)$$

- If the motion is small, then the intensity at the new location can be approximated by how intensity changes in space and time: (by Taylor expansion)

$$I(x+u, y+v, t+\Delta t) \approx I(x, y, t) + \frac{\partial I}{\partial x} u + \frac{\partial I}{\partial y} v + \frac{\partial I}{\partial t} \Delta t$$

By cancelling $I(x, y, t)$ on both sides,

$$\frac{\partial I}{\partial x} u + \frac{\partial I}{\partial y} v + \frac{\partial I}{\partial t} \Delta t = 0$$

Dividing by Δt on both sides

$$I_x \left(\frac{u}{\Delta t}\right) + I_y \left(\frac{v}{\Delta t}\right) + I_t = 0$$

$\rightarrow$ velocity per time step.

$$I_x u + I_y v + I_t = 0$$

(Optical flow constraint equation)

(edge) __/__/__
(direction of steepest brightness change)

Taking $\nabla I = (I_x, I_y)$ as gradient vects
$\Rightarrow (u, v)$ → flow vects. (how pixel is moving)

$$\therefore \quad I_t + \nabla I \cdot [u\ v] = 0$$

Here I_t → time derivative (how brightness at that pixel changes between frames)

- Change in brightness over time is caused by the motion of that pixel (u, v) moving through the brightness gradient ∇I.
- $\nabla I \cdot [u\ v]$ is projection of motion vects (u, v) onto direction of image gradient.
- If you move across an edge (dark → bright), $I_t \uparrow$
  If you move along an edge (same brightness), $I_t \downarrow$ or 0.

Note:
- While translational alignment literally tries to find the (u, v) that minimizes brightness difference, through brute force.
  Optical flow linearizes that minimization using derivatives (via Taylor Expansion), thus dropping higher order terms from E_{SSD}

— Lucas-Kanade Solution (patch-wise)

- For each pixel, the motion field (u, v) is constant within a small patch (window) of neighbouring pixels
- Hence, we can use multiple equations from neighbouring pixels to solve for one (u, v)

__/__/__

$Au = B$ where $A = \begin{bmatrix} I_{x_1} & I_{y_1} \\ I_{x_2} & I_{y_2} \\ \vdots & \vdots \\ I_{x_n} & I_{y_n} \end{bmatrix}$ $B = \begin{bmatrix} -I_{t_1} \\ -I_{t_2} \\ \vdots \\ -I_{t_n} \end{bmatrix}$

$u = [u \; v]$

- Goal is to find the minimum of $(Au - B)$
  $\Rightarrow \min |Au - B|^2$ using which we can obtain:
  (least squares solution) $A^T A u = A^T b$

  $u = (A^T A)^{-1} A^T b$ ← best-fit motion vectors for that patch.

Here $I_x, I_y \rightarrow$ spatial brightness gradients at each pixel
$I_t \rightarrow$ temporal brightness difference $(I_2 - I_1)$
$A \rightarrow$ contain spatial gradients for all pixels in patch
$b \rightarrow$ contains negative temporal gradients.

- $A^T A$ tells how directional and strong those spatial gradients are (if patch has no texture (flat), $A^T A \approx 0$, if gradient is unidirectional, as the edge, we can detect motion across the edge, not along)
  (L-R) (↕)
- If gradients in both directions (corners/textured regions) brightness varies any direction, motion vectors can be computed accurately (good feature to track)
- $A^T b$ tells which way and how strongly brightness changed ($A^T A$ tells about texture structure)
- Applications include video denoising where optical flow helps in aligning moving objects across frames so that we can average pixel intensities along their true motion paths (trajectories) to cancel noise (motion stays sharp)

KLT

- Tracker that implements an optical flow to track objects in videos. It is composed of:
- Good Features To Track (GFT) feature detection step to detect features at texture-rich structures (corners, edges)
- The feature tracking based on Lucas-Kanade (LK) method for these detected points

① Detect Harris corners in the first frame of the video

② For each detected Harris corner, compute the motion between consecutive frames using optical flow and local affine (to capture more complex motion) (rotation, scale, shear)

③ Now link these motion vectors from frame-to-frame to track the corners

④ Generate new Harris corners after few frames to compensate for new points entering the scene or discard the ones exiting the scene.

⑤ Track new and old Harris points.

Note: To detect Harris corners, $M = \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$ (structure tensor)

$R = \det(M) - k\left(I_x^2 + I_y^2\right)^2$

$R \uparrow \rightarrow$ strong corner

$R \downarrow \rightarrow$ edge / flat region

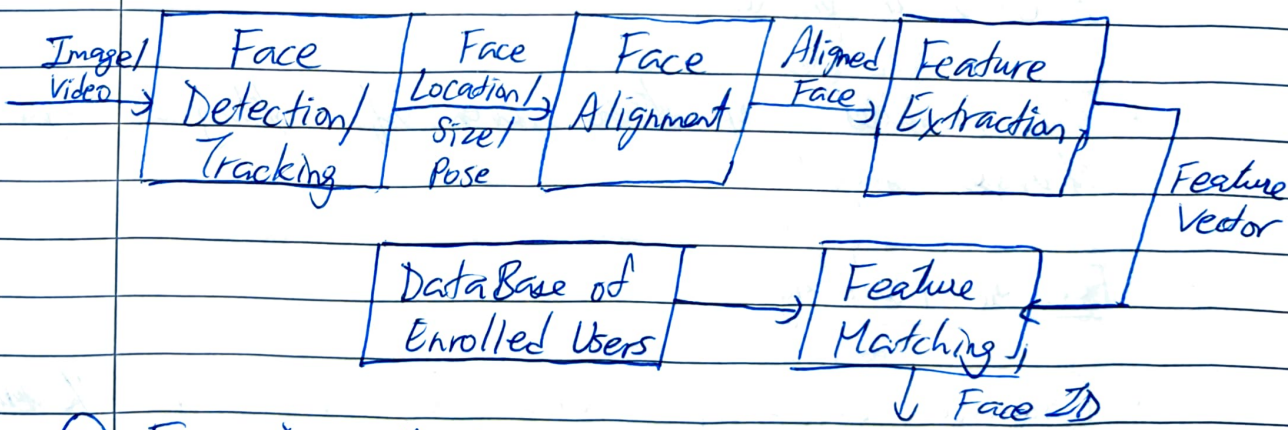
OBJECT DETECTION & RECOGNITION

- Object Detection → Segmentation
  Object Recognition → Machine Learning

☆ Face Recognition

- Maps an individual's facial features mathematically and stores the data as a faceprint.
- The software uses Deep Learning / Machine Learning algorithms to compare a live capture / digital image to the stored faceprint models to verify their identity.
- Applications: Biometric authentication, criminal investigation.

Phases



① Face ~~Detection~~ Detection

- Find where the face is in the image and separate it from the background.
- In videos, since the face moves, the system uses face tracking to follow the face frame-by-frame

- Output gives face location, size, angle/tilt (pose)

② Face Alignment

- Make the detected face look straight, centered and normalized before ~~compared~~ comparing it.
- Done using geometric transforms (rotation, scaling, etc.) to correct for size differences, pose differences, lighting differences to keep eyes at same level, same scale, etc.

③ Feature Extraction

- Analyzes the aligned face and extracts a feature vector which is a set of numerical values that represent distance between eyes, shape of jawline, position of nose, mouth, etc. with the help of 80 nodal points (or 30000 in phones)
- These features stay stable even if lighting or angle changes slightly.

④ Feature Matching

- Compare extracted features from new face with known faces in the database of enrolled users
- The system calculates a similarity score between faces if the score is high enough → face identified.
- ML Algorithms such as SVM, KNN, NN are used.

__/__/__

- Since each face has thousands of pixels (high dimensions) and faces occupy only a small subspace of all images
- PCA (Principal Component Analysis) helps in dimensionality reduction while keeping important face information. (Eigenfaces)
- This makes face recognition computationally efficient, and easy to implement.
- Say we have m training face images, each of size N×N

① • Convert each image into a 1D vector of size N^2

② • Compute the average face, $\phi = \frac{1}{m}\sum_{i=1}^{m} x_i$

and subtract from original image, $a_i = x_i - \phi$
thus centering all data around the mean.

③ • Form the matrix $A = [a_1, a_2, a_3, \ldots, a_m]$ of size $N^2 * m$

④ • Compute the covariance matrix $C = A^T A$ $(m \times m)$
(tells how faces vary together)

⑤ • Compute the eigen values (λ) and eigen vectors (v) of the covariance matrix (C) to compute eigenfaces. (u)

$$A^T A v_i = \lambda v_i \Rightarrow u_i = A v_i$$

Here eigenvectors give us info about direction of variation
eigenvalues tell how strong those directions are.

⑥ • Select K eigenvectors corresponding to the largest K eigenvalues → eigenfaces (main facial pattern directions)

⑦ • Represent each face as a combination of eigenfaces.

For each normalized face: $x_i - \phi = \sum_{j=1}^{k} w_j u_j$

where w_j → weights (coeff of each eigenface)

Now each face is represented as a K-dimensional weight vector:

$$x_i = [w_1, w_2, w_3 \ldots\ldots, w_k]$$

(feature vector of that face)

• Now for a new face test image y:

(i) Preprocess the image (crop, resize, etc.)
(ii) Subtract mean face, $\phi = y - \Psi$ mean
(iii) Project into eigen space:

$$\phi = \sum_{i=1}^{K} w_i u_i$$

Gives the weight vector: $\Omega = [w_1, w_2 \ldots w_k]$

(iv) Compare with training faces:

$$E = \min |\Omega - \Omega_k|$$

if $E < T$ (threshold) → (face recognized)

• Advantages: Simple and fast to compute, no feature marking required (eyes, mouth), works well in constant lighting and poses)

- Limitations: Needs centered, front-view faces, sensitive to lighting, scale, shadows, cannot handle extreme pose/expression variations.

★ Computer Aided Diagnosis

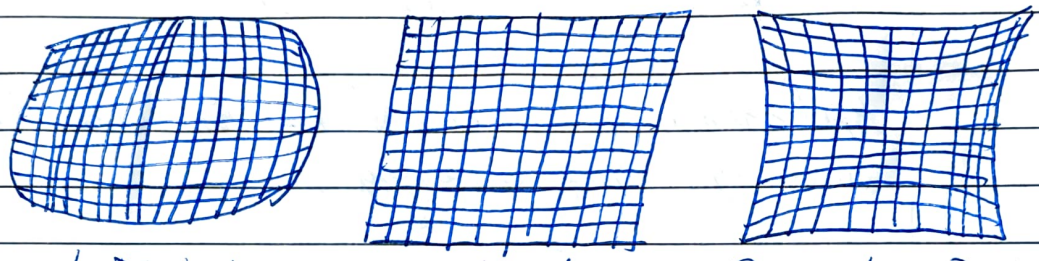
- Software that analyzes medical image to diagnose the disease based on features represented, they assisting radiologists in decision-making.

- Applications are clinical decision support system, robotic surgery, drug discovery, etc.

- Some medical imaging techniques include Ultra-Sonography (US), X-ray, Computed Tomography (CT), Magnetic Resonance Imaging (MRI), Nuclear Imaging (PET, SPECT)

CAMERA CALIBRATION

- Camera converts 3D image → 2D image.
- Camera calibration estimates parameters of camera it
  - correct for lens distortion (not showing true shape)
  - measure size of an object
  - determine location of the camera in the scene.



Barrel Distortion Input Pincushion Distortion

- Intrinsic parameters (describe internal characteristics of camera) include focal length, optical center, etc.
- Extrinsic parameters include rotation, translation (camera's position and orientation relative to real world.)

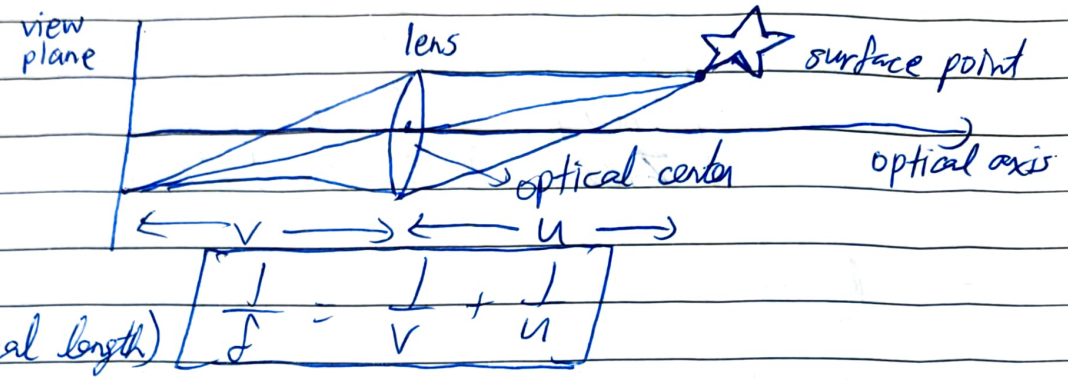
☆ · Camera Models

- Simulates the capture of light from 3D scene onto a 2D image.

– Thin lens Model

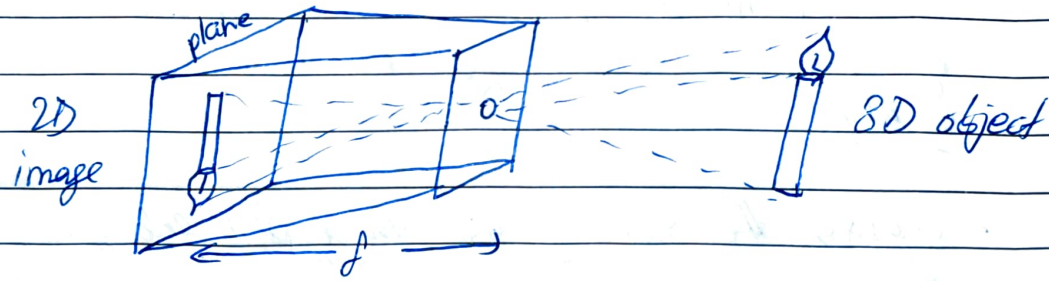
- Modern cameras use a lens to focus light onto a view plane quickly
- In this model, rays of light emitted from a point on an

objed travel along paths through the lens, converging at a point behind the lens (screen)



— Pinhole Camera Model (simples)

- Light from a point travels along a single straight path through a tiny pin-hole onto a view plane (no lens)



✱ Geometric Transformations

- Involves altering the spatial relationships between pixels in an image to manipulate the image's position, size, orientation or shape.

— Translation

- Shifts an image from one location to another without changing size, shape or orientation.

- Shift by (t_x, t_y), every pixel at (x, y) is moved to $(x + t_x, y + t_y)$

$$\begin{bmatrix} x_{new} \\ y_{new} \\ 1_{new} \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{old} \\ y_{old} \\ 1 \end{bmatrix}$$

— Rotation

- Rotate an image around a specific point (center) by a specific angle θ (CW or ACW)

$$\begin{bmatrix} x_{new} \\ y_{new} \\ z_{new} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{old} \\ y_{old} \\ 1 \end{bmatrix}$$

— Scaling

- Changes the size of the image by enlarging or shrinking it by a scale factor, uniformly ($S_x = S_y$) or non-uniformly (aspect ratio preserved)

$$\begin{bmatrix} x_{new} \\ y_{new} \\ z_{new} \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{old} \\ y_{old} \\ 1 \end{bmatrix}$$

— Skewing (Shearing)

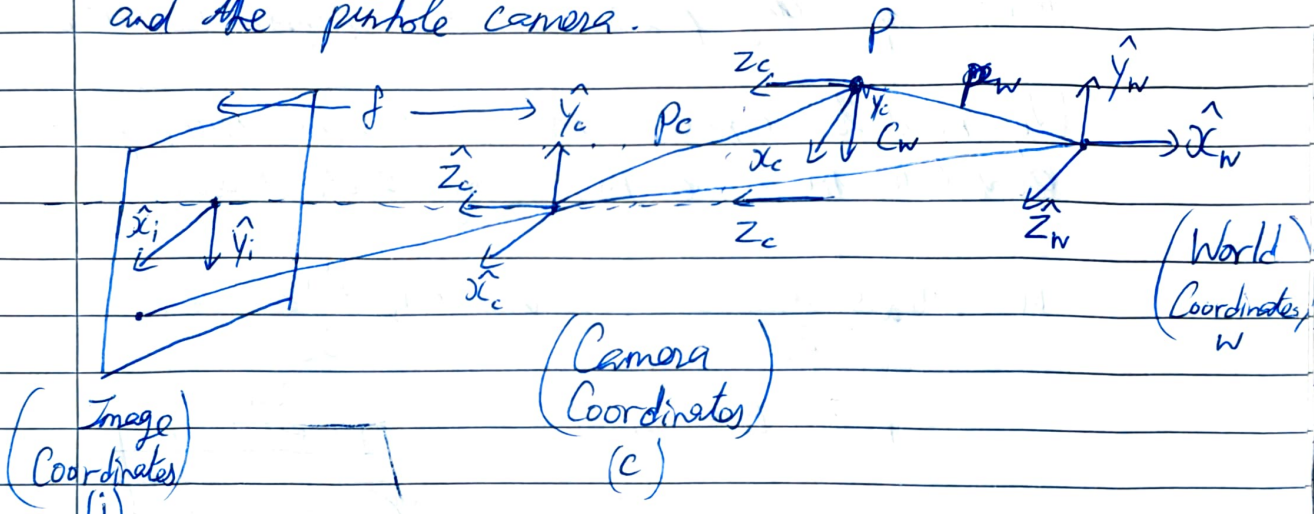
- Distorts the image, shape is slanted, altering angles between lines that were originally perpendicular.
- Image will look pushed in one direction, leaning to one side

$$\begin{bmatrix} 1 & k_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$
(Horizontal skew)

$$\begin{bmatrix} 1 & 0 & 0 \\ k_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$
(Vertical skew)

☆ Forward Imaging Model (3D → 2D)

- Optical axis is the axis perpendicular to the image plane.
- Focal length is the distance between the image plane and the pinhole camera.



Here $(\hat{x}_w, \hat{y}_w, \hat{z}_w)$ → World Coordinate System
$(\hat{x}_c, \hat{y}_c, \hat{z}_c)$ → Camera Coordinate System
$(\hat{x}_i, \hat{y}_i)$ → Image Plane Coordinate System.
P → point of interest
p_c → vector of P with respect to camera origin
p_w → vector of P wrt world coordinate origin (location)
f → focal length of camera

By triangular similarity,

$$\frac{x_i}{f} = \frac{x_c}{z_c}, \quad \frac{y_i}{f} = \frac{y_c}{z_c}$$

Intrinsic Parameters (Image plane / Sensor ⟷ Camera)

Here Z_c → how far point is from camera along the optical axis.

⟷

X_c → horizontal distance of P from optical axis

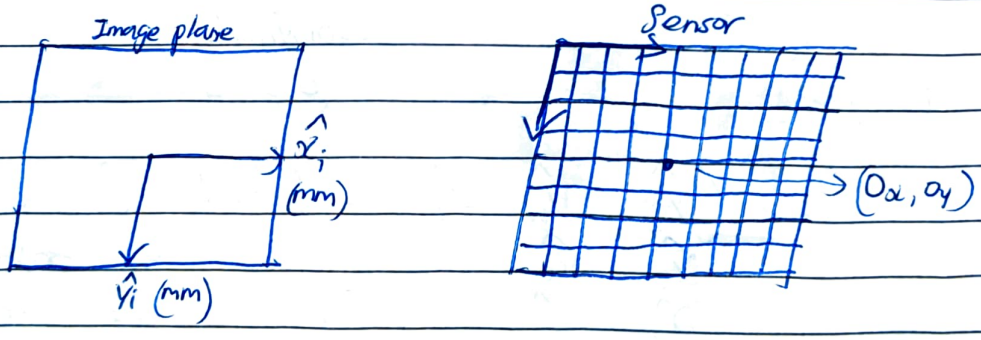
Y_c → vertical distance of P from optical axis

$$x_i = f\frac{X_c}{Z_c}, \quad y_i = f\frac{Y_c}{Z_c}$$

- Since image plane is just a geometric plane where coordinates are in millimetres (mm) or metres (m) describing physical distance, and the image sensor is made of tiny pixels of some physical size,

We need to convert mm → pixels using pixel densities m_x and m_y along each direction (pixels/mm)

$$u = m_x f\frac{X_c}{Z_c}, \quad v = m_y f\frac{Y_c}{Z_c}$$



- Even though the image plane's origin is set at the optical axis intersection, the image sensor's coordinate system sets the origin at the top-left corner of the image, hence we add an offset to the coordinates.

$$u = m_x f\frac{X_c}{Z_c} + O_x, \quad v = m_y f\frac{Y_c}{Z_c} + O_y$$

$u = f_x \frac{x_c}{z_c} + o_x$, $v = f_y \frac{y_c}{z_c} + o_y$

where (f_x, f_y) are focal lengths in pixels.

Here, (f_x, f_y, o_x, o_y) are the intrinsic parameters of the camera (internal geometry)

Note Since computer vision has to perform multiple tasks such as rotating the 3D model, translating, projecting onto 2D image and converting to pixels which will involve a lot of math equations, for simplicity, to represent in one simple matrix for matrix multiplication to be performed.

We insert a homogeneous coordinate in the matrix:

Ex: To translate an image by (t_x, t_y) before,

$$\begin{bmatrix} x \\ y \end{bmatrix} \longrightarrow \begin{bmatrix} x + t_x \\ y + t_y \end{bmatrix}$$

If we want to perform rotation + translation to an image

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = R \begin{bmatrix} x \\ y \end{bmatrix} + t$$

and again,

$$p'' = R_2(R_1 p + t_1) + t_2$$
$$= R_2 R_1 p + R_2 t_1 + t_2$$

This mix of multiplications and addition makes it messy.

On the other hand, if we used homogeneous coordinates,

$$p'' = (BA)p \qquad \begin{matrix} (A \rightarrow \text{translation}) \\ (B \rightarrow \text{rotation}) \end{matrix}$$

For example, by adding a homogeneous coordinate,

$$p = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \quad A = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$p'' = Ap = \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix}$$

Simple translation
For 2D → 3×1, 3×3
For 3D → 4×1, 4×4

- The homogeneous representation of a 2D point $u = (u, v)$ is a 3D point $\tilde{u} = (\tilde{u}, \tilde{v}, \tilde{w})$ such that:

$$u \equiv \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \equiv \begin{bmatrix} \tilde{w}u \\ \tilde{w}v \\ \tilde{w} \end{bmatrix} \equiv \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} = \tilde{u} \quad / \quad u = \frac{\tilde{u}}{\tilde{w}}, \; v = \frac{\tilde{v}}{\tilde{w}}$$

- Hence for perspective projection:

→
$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \simeq \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} \simeq \begin{bmatrix} Z_c u \\ Z_c v \\ Z_c \end{bmatrix} = \begin{bmatrix} f_x x_c + Z_c o_x \\ f_y y_c + Z_c o_y \\ Z_c \end{bmatrix}$$

$$= \begin{bmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix}$$

Here intrinsic matrix

$$M_{int} = [K|0] = \begin{bmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

K: calibration matrix

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = [K|0] \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = M_{int.} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix}$$

Extrinsic Parameters (Camera ⟷ World)

- c_w → location/vector of camera wrt world coordinates.
  R → orientation of camera in world.

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{matrix} \text{(Direction of } \hat{x}_c \text{ in world coordinates)} \\ \text{(Direction of } \hat{y}_c \text{ in world coordinates)} \\ \text{(Direction of } \hat{z}_c \text{ in world coordinates)} \end{matrix}$$

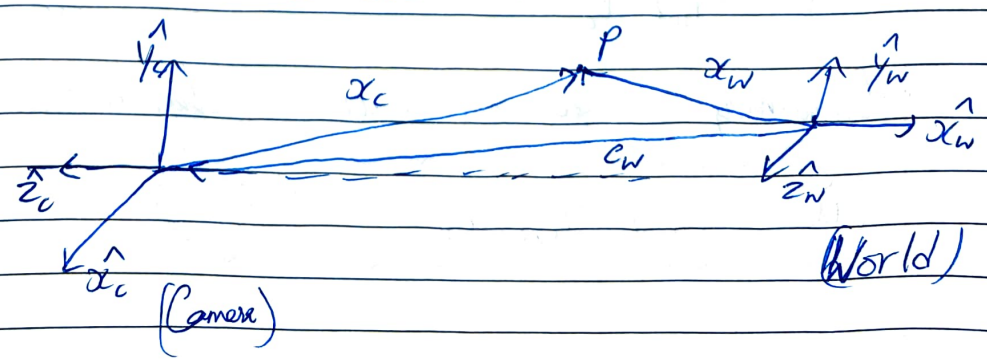
- Here R is an orthonormal matrix whose row/column vectors are orthonormal such that:

$$R^{-1} = R^T, \quad R^T R = R R^T = I$$

- Given (R, c_w), the camera-centric location of point P, in the world coordinate frame:

$$x_c = R(x_w - c_w) = R x_w - R c_w$$

$$x_c = R x_w + t \qquad (t = -R c_w)$$



__/__/__

Here R (rotation matrix) describes how the camera is oriented relative to the world
t (translation vector) describes where the camera is located in world coordinates

$$\tilde{x}_c = \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

Here $M_{ext} = \begin{bmatrix} R_{3\times3} & t \\ 0_{3\times3} & 1 \end{bmatrix}$ $\quad \tilde{x}_c = M_{ext}\, \tilde{x}_w$

- Hence Extrinsic Matrix is used to map world coordinates to camera coordinates while
Intrinsic Matrix is used to map camera coordinates to image plane coordinates

- **Camera → Pixel**

$$\begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} = \begin{bmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix}$$

$\tilde{u} = M_{int}\, \tilde{x}_c$

World → Camera

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

$\tilde{x}_c = M_{ext}\, \tilde{x}_w$

- Combining to get the full projection matrix P.

$$\tilde{u} = M_{int}\, M_{ext}\, \tilde{x}_w = P\, \tilde{x}_w$$

$$\begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

- To estimate this projection matrix P,

① Capture an image of object with known geometry. ($\tilde{u}$)

② Identify correspondence between 3D scene points and image points. ($\tilde{x}_w$)

③ For each corresponding point i in scene and image,

$$\underbrace{\begin{bmatrix} u^{(i)} \\ v^{(i)} \\ 1 \end{bmatrix}}_{\text{(Known)}} = \underbrace{\begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix}}_{\text{(Unknown)}} \underbrace{\begin{bmatrix} x_w^{(i)} \\ y_w^{(i)} \\ z_w^{(i)} \\ 1 \end{bmatrix}}_{\text{(Known)}}$$

④ Expand the matrix as linear equations and rearrange.

$$u_i^{(i)} = \frac{p_{11} x_w^{(i)} + p_{12} y_w^{(i)} + p_{13} z_w^{(i)} + p_{14}}{p_{31} x_w^{(i)} + p_{32} y_w^{(i)} + p_{33} z_w^{(i)} + p_{34}}$$

$$v^{(i)} = \frac{p_{21} x_w^{(i)} + p_{22} y_w^{(i)} + p_{23} z_w^{(i)} + p_{24}}{p_{31} x_w^{(i)} + p_{32} y_w^{(i)} + p_{33} z_w^{(i)} + p_{34}}$$

$$\underbrace{\begin{bmatrix} x_w^{(i)} & y_w^{(i)} & z_w^{(i)} & 1 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & x_w^{(i)} & y_w^{(i)} & z_w^{(i)} & 1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots \end{bmatrix}}_{A \text{ (known)}} \underbrace{\begin{bmatrix} p_{11} \\ p_{12} \\ p_{13} \\ \vdots \\ p_{34} \end{bmatrix}}_{p \text{ (unknown)}} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

⑤ Solve for P, $AP = 0$

$A^T A p = \lambda p$, find eigenvectors p with smallest eigen value λ of matrix $A^T A$.

- Calibration pattern (grid / target) is a repeating pattern of known size and spacing.

☆ Vanishing Point

- Point on image plane where mutually parallel lines in 3D space appear to converge / meet (roads, train tracks)
- Each set of parallel lines have a direction vector
  p_1 (x) = $[1, 0, 0]^T$
  p_2 (y) = $[0, 1, 0]^T$
  p_3 (z) = $[0, 0, 1]^T$
- When rays from parallel 3D lines enter the camera and hit the 2D image plane, they appear to converge
- Vanishing points tell us about the direction of lines in 3D, orientation of camera and it's focal length.
- When camera rotates, the direction of world axis in the camera coordinate system is given by Rp_i (parallel lines)
  where p_i → direction of real-world parallel lines,
  R → camera rotation matrix
  Vanishing point corresponds to i^{th} column of R
  $Rp_i = r_i$

$$\hat{x_i} = \begin{bmatrix} x_i - c_x \\ y_i - c_y \\ f \end{bmatrix} \sim R p_i = r_i$$

Vector from camera centre to vanishing point equals direction vector of lines

For two diff sets of parallel lines, we get vanishing points $\hat{x_i}$, $\hat{x_j}$.

Since columns of rotation matrix R are orthogonal (camera's X, Y, Z are ⊥) $\Rightarrow r_i \cdot r_j = 0$

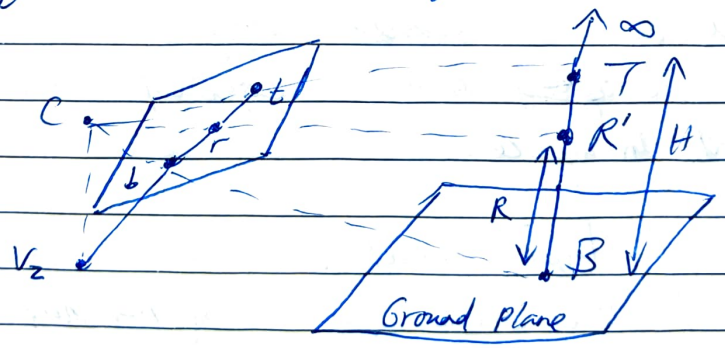
$\hat{x_i} \sim r_i$ and $\hat{x_j} \sim r_j$, hence $\hat{x_i} \cdot \hat{x_j} = 0$

$$(x_i - c_x)(x_j - c_x) + (y_i - c_y)(y_j - c_y) + f^2 = 0$$

(solve for f)

- Application (Single View Metrology)

- To measure the height of an object (H) with the help of a reference object whose height is known.



Here T → top of unknown object
R' → top of reference object
B → bottom point on ground plane.
R → known height of reference object
H → unknown height
∞ → point at infinity (direction of vertical line (convergence)) like vanishing point in 3D

In the image, V_z → projection of ∞ point (vanishing point for all vertical lines)

b → projection of B

r → projection of R

t → projection of T

c → center of projection (camera's optical center)

Scene - cross ratio, $\frac{H}{R} = \frac{\|T-B\| \, \|\infty - R\|}{\|R-B\| \, \|\infty - T\|}$

Image - cross ratio, $\frac{|t-b| \, |V_z - r|}{|r-b| \, |V_z - t|} = \frac{H}{R}$ → pixel distances

$$\left(H = R \cdot \frac{|t-b| \, |V_z - r|}{|r-b| \, |V_z - t|} \right)$$

If t is closer to V_z → object is taller.

If t is only a little higher than r → object is short.

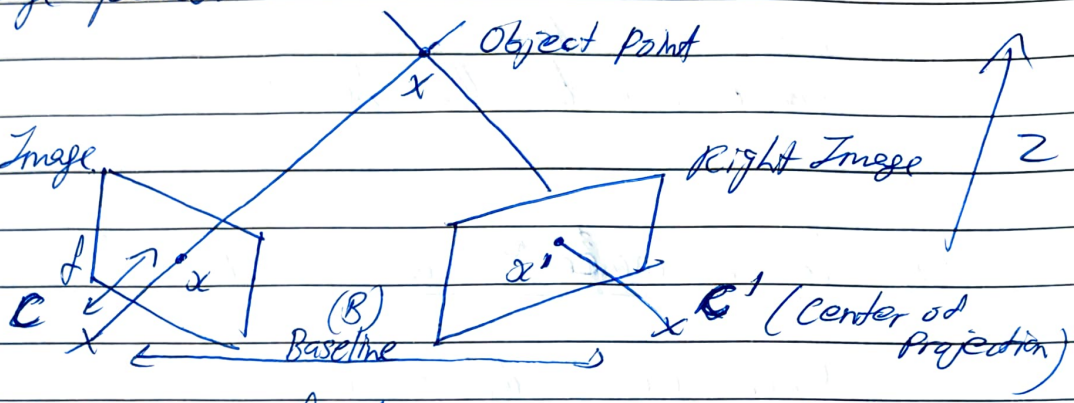
Stereo Correspondence

- Recovery of 3D structure of a scene/object using two images acquired from different viewpoints in space.
- Process of taking multiple images and estimating a 3D model of the scene by finding matching pixels in the images and converting their 2D positions → 3D depths.
- In visual perception, we perceive depth based on differences in appearance between left & right eye.
- Applications include robotic navigation, 3D model building

__/__/__

(depth z)

- Stereo Vision determines position of point in space by finding the intersection of two lines passing through center of projection and projection of object point in on image planes.



- Triangulation → finding 3D point by intersecting 2 rays.
- Here baseline (distance between 2 cameras) and f (focal length) (how strong camera projects into 2D) are all known from calibration.

$$\text{Depth} = \frac{f\,B}{x - x'}$$

(Disparity)
(how far apart x and x' are based on where object is)

Object is close, $(x - x')$ ↑
Object is far, $(x - x')$ ↓

(position of object in each eye's view differs)

☆ RANSAC

- Random Sample Consensus algorithm.
- Outlier detection method that can be used for identifying corresponding points in image plane
- Finding commonality between 2 sets of points for feature based object detection.

__/__/__

① Randomly choose 'S' samples (min samples to fit the model)
② Fit the model to the randomly chosen samples
③ Count the number M of data points (inliers) that fit the model within a measure of error E
④ Repeat N times (① → ③)
⑤ Choose a model that has the largest M of inliers.

☆ Co-occurrence Matrix Formulas (for Texture Analysis)

- Once we build a co-occurrence matrix G for a given image at some distance d and orientation θ, we can summarize the patterns inside G with a few numbers / features: (Haralick) (K → no. of gray levels used to build G)

① Maximum Probability (count of pair / total no. of pairs) (freq)

$\left(\max_{i,j}(pair_{i,j})\right)$, Large → one pair dominates, texture is repetitive
Small → no dominant relationship, texture is varied
Captures how predictable the texture is.

② Correlation (how much value of pixel predicts value of neighbours)

$\left(\sum \frac{(i-m_r)(j-m_c)P_{ij}}{\sigma_r \sigma_c}\right)$ (Do i and j ↑ or ↓ together? (trend))

High correlation → neighbours tend to be similar, vary together (bright → bright, dark → dark)

Low → neighbours are random
Negative → high with low

(how far entries are from diagonal)

③ Contrast (How much pixel pairs differ from one another)

$$\left(\sum\sum (i-j)^2 p_{ij}\right)$$

If neighbouring pixels are similar, $(i-j)^2$ → small contrast → low

If neighbours differ a lot (edges / rough surfaces) contrast high

Used to detect sharp transitions, rough textures, edges.

④ Uniformity (Energy) (Measures smoothness of texture)

$$\left(\sum\sum p_{ij}^2\right)$$

(concentration)

High → few entries dominate, very consistent texture

Low → spread out probabilities, complex texture.

⑤ Homogeneity (gives higher weight to values near the diagonal)

$$\left(\sum\sum \frac{p_{ij}}{1+|i-j|}\right)$$

If $i \approx j$, homogeneity is high (diagonals)

If $|i-j|$ is big, neighbours differ, homogeneity ↓. (noisy texture)

Measure smoothness (how similar intensities are)

⑥ Entropy (measures randomness of texture)

$$\left(-\sum\sum p_{ij} \log_2(p_{ij})\right)$$

Low entropy → one dominant pair → repetitive

High entropy → many possible pairs → random/noisy texture