# COMPUTER VISION

Acquired images → Image Processing → Feature Extraction → classification → output : Classical computer vision

Acquired image → Deep Neural Network → output : Modern computer vision

## Image Formation

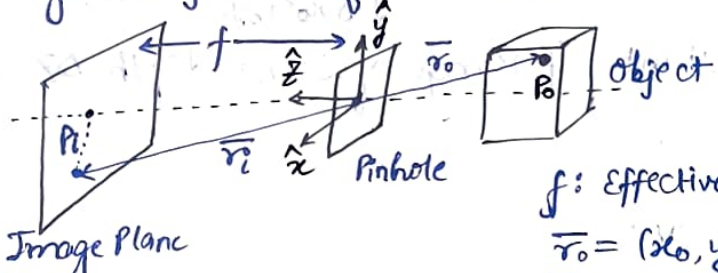
### Geometry of Image Formation

↳ Mapping b/w image and world coordinates.

1. Pinhole camera Model
2. Projective geometry
3. Projective Matrix.

### Image Formation in camera:

↳ Image: Projection of 3D scene onto 2D plane.




f: Effective focal length

$\bar{r}_o = (x_o, y_o, z_o)$

$\bar{r}_i = (x_i, y_i, f)$

Using similar Δs,

$$\frac{\bar{r}_i}{f} = \frac{\bar{r}_o}{z_o} \Rightarrow \frac{x_i}{f} = \frac{x_o}{z_o}, \frac{y_i}{f} = \frac{y_o}{z_o}$$

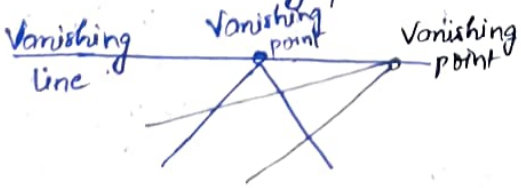
Pinhole: barrier
→ Reduce blurring
→ Opening known as aperture.

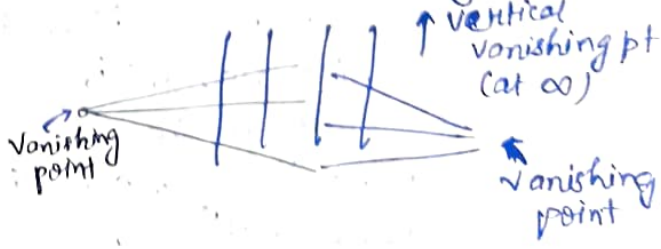
↳ Lost: length, angles

Preserved: straight lines are still straight.

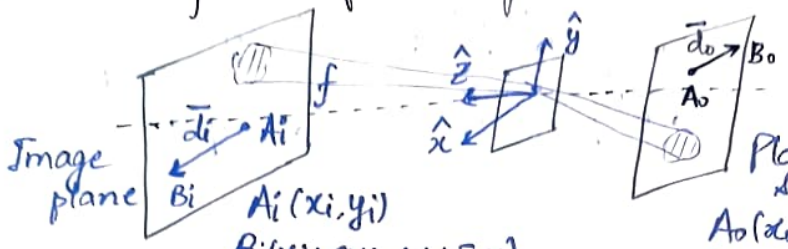
### Vanishing Points and Lines

↳ Parallel lines in the world intersect in the image at a 'vanishing point'.



## Perspective Projection of a Straight Line




$A_i(x_i, y_i)$

$B_i(x_i+\delta x_i, y_i+\delta y_i)$

$A_o(x_o, y_o, z_o)$

$B_o(x_o+\delta x_o, y_o+\delta y_o, z_o)$

Magnification, $|m| = \frac{\|\bar{d}_i\|}{\|\bar{d}_o\|} = \left|\frac{f}{z_o}\right| \Rightarrow \boxed{m = \frac{f}{z_o}}$ ($<0$ when image is inverted)

From perspective projection,

$$\frac{x_i}{f} = \frac{x_o}{z_o}, \quad \frac{y_i}{f} = \frac{y_o}{z_o}$$

$$\Rightarrow \frac{\delta x_i}{f} = \frac{\delta x_o}{z_o}, \quad \frac{\delta y_i}{f} = \frac{\delta y_o}{z_o}$$

$$\therefore |m| = \frac{\|\bar{d}_i\|}{\|\bar{d}_o\|} = \frac{\sqrt{\delta x_i^2 + \delta y_i^2}}{\sqrt{\delta x_o^2 + \delta y_o^2}} = \left|\frac{f}{z_o}\right|$$

→ $m \approx$ constant, if range of scene depth $\Delta Z \ll$ average scene depth $\bar{Z}$.

→ $\frac{\text{Area}_i}{\text{Area}_o} = m^2$

### Challenges in CV:

→ Perspective illusion
→ viewpoint variation
→ Illumination
→ Motion
→ Perception vs. Measurement.
→ Intraclass variability

### Applications:

→ Optical character recognition (OCR)
→ Face detection, smile detection
→ vision-based biometrics, facial login
→ Object recognition
→ 3D from images
→ Special effects: Motion capture
→ Interactive games: Kinect
→ Sports
→ Medical imaging
→ Auto cars
→ Industrial robots
→ vision in space
→ Mobile robots
→ Augmented reality & Virtual reality.

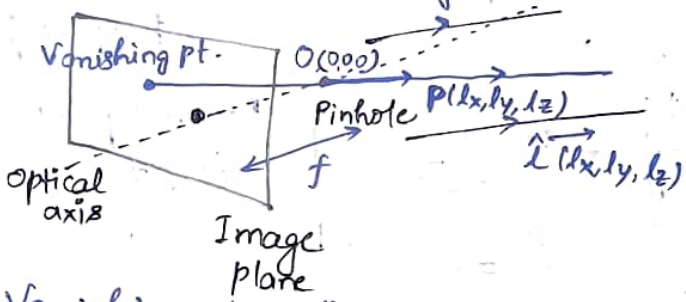
# Pinhole Camera and Perspective Projection

Perspective projection: Mapping from 3D onto 2D.

Pinhole camera: Imaging device which captures accurately the geometry of perspective projection.

Camera & world coordinate system
↳ In general, not aligned.

Vanishing Point: location depends on the orientation of parallel straight lines.




Vanishing pt. of line: Projection of point P

$$(x_{vp}, y_{vp}) = \left(f\frac{l_x}{l_z}, f\frac{l_y}{l_z}\right).$$

Pinhole → tiny
↳ cause diffraction, if it's too tiny.

Ideal pinhole diameter: $\boxed{d \approx 2\sqrt{f\lambda}}$

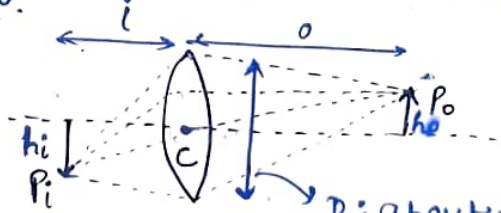
f: effective focal length
λ: wavelength

## Lenses

↳ Same projection as pinhole, but gather more light.

Gaussian lens (thin lens) law:

$$\boxed{\frac{1}{i} + \frac{1}{o} = \frac{1}{f}}$$




f: focal length
i/o: image/object distance.
If $o = \infty$, $f = i$.

D: aperture / diameter of lens
↳ reduce/increase to control image brightness.

Magnification: $\boxed{m = \frac{h_i}{h_o} = \frac{i}{o}}$

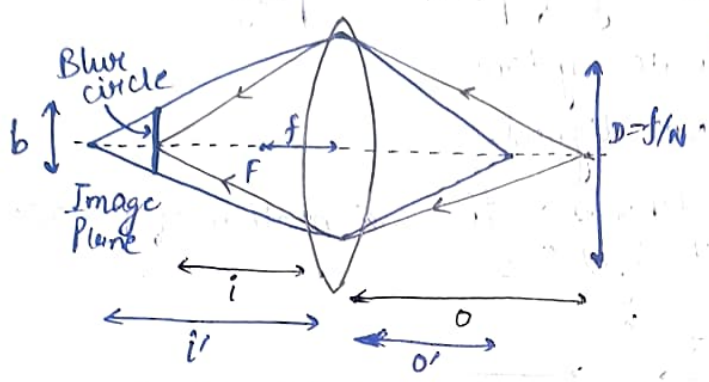
For two lens system, $m = \frac{i_2}{o_2}\cdot\frac{i_1}{o_1}$

Aperture, $D = f/N$

f-Number, $N = f/D$

- Lens distortion (vignetting)
- Chromatic aberration
- Geometric distortion
  - Radial, tangential

## Lens Defocus




From similar $\Delta$s,

$$\frac{b}{D} = \frac{|i'-i|}{i'}$$

Blur circle diameter,

$$b = \frac{D}{i'}|i'-i|$$

$b \propto D \propto \frac{1}{N}$

Defocused point:

$$\frac{1}{i'} + \frac{1}{o'} = \frac{1}{f} \Rightarrow i' = \frac{o'f}{o'-f}$$

$$\Rightarrow i'-i = \frac{f}{o'-f} \cdot \frac{f}{o-f} \cdot (o-o')$$

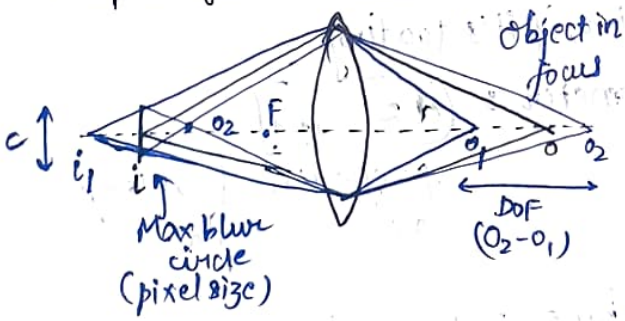
$$\therefore b = Df\left|\frac{(o-o')}{o'(o-f)}\right| \Rightarrow \boxed{b = \frac{f^2}{N}\left|\frac{(o-o')}{o'(o-f)}\right|}$$

Focused point

$$i = \frac{of}{o-f}$$

→ Move image plane or lens or both for focusing.

## Depth of Field (DoF)

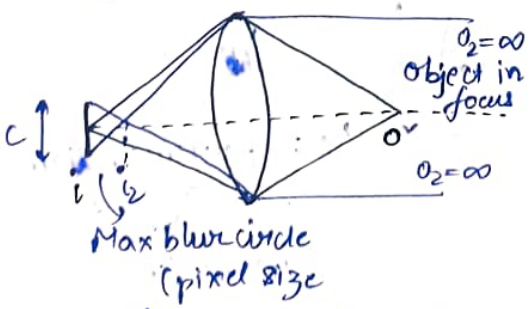



$o_1, o_2$: nearest & farthest distances for blur circle to be max$^m$ c.

$$c = \frac{f^2(o-o_1)}{No_1(o-f)} = \frac{f^2(o_2-o)}{No_2(o-f)}$$

DoF: $$\boxed{o_2-o_1 = \frac{2of^2cN(o-f)}{f^4 - c^2N^2(o-f)^2}}$$

## Hyperfocal distance




The closest distance $o=h$, the lens must be focused to keep objects at $\infty$ ($o_2$) acceptably sharp (blur circle $\le c$).

hyperfocal distance, $$\boxed{h = \frac{f^2}{Nc} + f}$$

→ Large aperture (small-f no.)
- Bright image or short exposure time
- Shallow depth of field

→ Small aperture (large f-No.)
- Dark image or long exposure time
- Large DoF.

Allow camera rotation:

$$x = K[R\ t]X \Rightarrow w\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha & s & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix}\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Degree of freedoms : (5) (6)

## Projective geometry:

1. A circle may appear as an ellipse.
2. Parallel lines may meet at a finite point.
3. A rectangle may appear as a parallelogram.

→ Angles, distances, ratio of distances are not preserved.

→ $(kx, ky, k)$ → corresponds to → $\left(\frac{kx}{k}, \frac{ky}{k}\right) = (x, y)$
in homo. coord. in cartesian coord.

→ $(kx, ky, k)$ are equivalent for all $k$'s.

→ Point at infinity : $(x, y, k)$ where $k=0$
i.e., $\left(\frac{x}{0}, \frac{y}{0}\right)$.

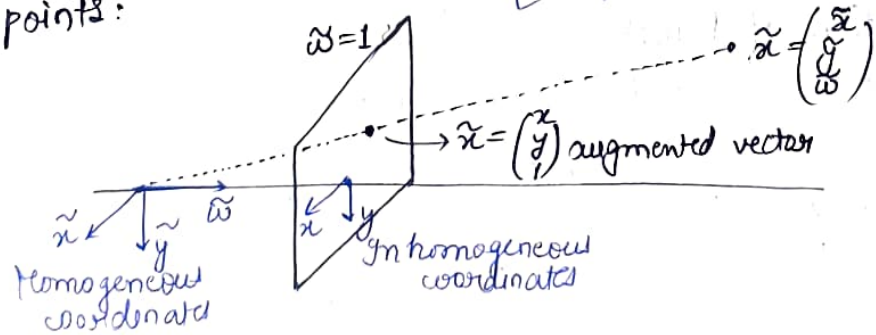
## Homogeneous Point

$$X = \begin{pmatrix} x \\ y \end{pmatrix} \in \mathbb{R}^2$$

$$\tilde{X} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \in \mathbb{P}^2$$

where $\mathbb{P}^2 = \mathbb{R}^3$ is called projective space
[Except the element (0,0,0)]

2D-points:



16-01-2025

# CAMERA

↳ CCD & CMOS camera : Convert light into electrical signals to produce images.

CCD sensor : Have passive pixels that transfer charge to a single amplifier

~~CMOS sensor~~

↳ Have better image quality, especially in low-light settings.

CCD camera :

↳ Have Charged Coupling Device

↳ Photons → electrons → no.s → bits

↳ Have alternative RGB pixels that capture the respective colors.

SAURABH KUMAR | SC22B146

| CCD Camera Sensor | CMOS Camera Sensor |
|---|---|
| • Have passive filters that transfer the charge to a single amplifier<br>• Charge Coupled Device.<br>• Each charge in each pixels are transferred using horizontal and vertical shift registers, then converted to voltage and amplified.<br>• Not possible to integrate peripherals like timers and ADC, in the main sensor, required additional chip and thus large size.<br>• Requires different power supplies, typically 7–10 V, thus consuming more power.<br>• Low processing speed as each pixel is processed one-by-one.<br>• Large fill factor, noise and sensitivity.<br>• Better camera image quality.<br>• Use global shutter.<br>• Suffer from smear.<br>• Image distortion due to blooming. | • Have active pixels that amplify the charge in each pixel.<br>• Complementary Metal Oxide Semiconductor.<br>• Charge to voltage conversion and amplification are carried out in the pixels itself.<br>• Fabrication procedure is very similar to that of IC, it is possible to integrate peripheral components into single chip, thus, possible to have camera on chip or SoC.<br>• Requires single power supply, typically 3.3–5 V, thus consuming less power.<br>• High speed as processing is carried out in the pixel itself.<br>• Low fill factor, noise and sensitivity.<br>• Cheaper, energy efficient<br>• Use rolling shutter (R.S. artifacts).<br>• Affected by skew, wobble & partial exposure. |

# Homogeneous Coordinate System

## Homogeneous coordinates:

$(x, y) \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$ Homo. image coord.

$(x, y, z) \Rightarrow \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$ Homo. scene coord.

[Converting to homogeneous coordinates]

$\begin{bmatrix} x \\ y \\ w \end{bmatrix} \Rightarrow (x/w, y/w)$ $\quad \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \Rightarrow (x/w, y/w, z/w)$ [Converting from homogeneous coordinates]

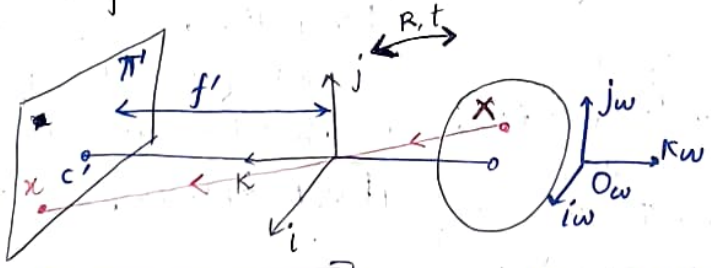
Projective Space $\rightarrow \mathbb{P}^2$

$\begin{pmatrix} kx \\ ky \\ k \end{pmatrix} \Rightarrow \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$ [Invariant to scaling]

$\left(\frac{kx}{k}, \frac{ky}{k}\right) \rightarrow (x, y) \Rightarrow \mathbb{R}^2$ [Cartesian]

→ Point in Cartesian is ray in Homogeneous.

## Camera Projection Matrix:




$$x = K[R \; t]X$$

x : image coordinates : (u, v, 1)
K : Intrinsic Matrix (3×3)
R : Rotation (3×3)
t : Translation (3×1)
X : world coordinates : (X, Y, Z, 1)

Intrinsic assumption:
- ↳ Unit skew ratio
- ↳ Optical center at (0,0)
- ↳ No skew

Extrinsic assumptions:
- ↳ No rotation
- ↳ Camera at (0,0,0).

$$x = K[I \; 0]X \Rightarrow w\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

(K)

# 2-D Line :

21-01-2025

## Parametric form of line :

$ax + by + c = 0$

$$\begin{bmatrix} a & b & c \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = 0 \Rightarrow l^T x = 0 \quad ; l = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

$(x_1, x_2, x_3) \Rightarrow \left(\frac{x_1}{x_3}, \frac{x_2}{x_3}\right)$

Homogeneous space — Non-homo.

For Two lines $l$ and $l'$ intersect at $x$.

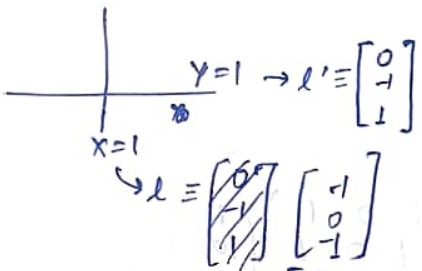
$\Rightarrow \boxed{x = l \times l'}$ (Cross Product)

$\Rightarrow l^T x = 0$

$\Rightarrow l^T (l \times l') = 0$

- $ax + by + c = 0$ is same as $(ka)x + (kb)y + (kc) = 0 \quad \forall k \neq 0$
- $(0,0,0)^T \rightarrow$ not a line
- $(a,b,c)^T$, $k(a,b,c)^T \rightarrow$ same line
- Point $x = (x,y)^T$ lies on line $l = (a,b,c)^T$ iff $ax + by + c = 0$ i.e., $(x, y, 1)(a,b,c)^T = 0$

Eg




$\rightarrow l' \equiv \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix}$

$\rightarrow l \equiv \begin{bmatrix} -1 \\ 0 \\ -1 \end{bmatrix}$

$$l \times l' = \begin{vmatrix} i & j & k \\ -1 & 0 & 1 \\ 0 & -1 & 1 \end{vmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \equiv \left(\frac{1}{1}, \frac{1}{1}\right)$$

→ Point at infinity : Ideal point

↳ $(x, y, 0) \equiv \left(\frac{x}{0}, \frac{y}{0}\right)$

## Parallel Lines

Eg. $x = 1 \quad (l)$

$x = 2 \quad (l')$

$l = [1 \;\; 0 \;\; -1]^T$

$l' = [1 \;\; 0 \;\; -2]^T$

$$l \times l' = \begin{pmatrix} i & j & k \\ 1 & 0 & -1 \\ 1 & 0 & -2 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

## Line at infinity : $\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = l_\infty$

$\boxed{l^T \cdot l_\infty = 0} \Rightarrow (0 \;\; 0 \;\; 1)\begin{pmatrix} x \\ y \\ 0 \end{pmatrix} = 0.$

↳ ideal point

Line joining two points $x_1$ and $x_2$: $l = x_1 \times x_2$

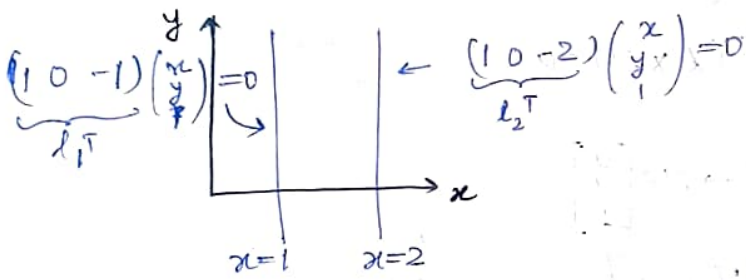
Cross Product:

Convert one vector to skew: $[x]_\times = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix}$

$l_1^T = (0 \;\; 1 \;\; -1)$ ; $l_2^T = (1 \;\; 0 \;\; -2)$

$$l_1 \times l_2 = [l_1]_\times l = \begin{pmatrix} 0 & 1 & 1 \\ -1 & 0 & 0 \\ -1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ -2 \end{pmatrix} = \begin{pmatrix} -2 \\ -1 \\ -1 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix}$$

Parallel lines:




$\underbrace{(1 \;\; 0 \;\; -1)}_{l_1^T} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = 0$ ← $\underbrace{(1 \;\; 0 \;\; -2)}_{l_2^T} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = 0$

$$l_1 \times l_2 = [l_1]_\times l_2 = \begin{pmatrix} 1 & 0 & 1 \\ -1 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ -2 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

$$\underbrace{l_\infty}_{(0 \;\; 0 \;\; 1)^T} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = 0$$

Conics

Circle: $x^2 + y^2 - 1 = 0$ $\longrightarrow$ $(x/z)^2 + (y/z)^2 - 1 = 0$ $\xrightarrow{xz^2}$ $x^2 + y^2 - z^2 = 0$

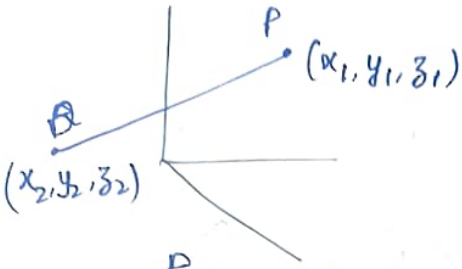
Parabola: $x^2 - y = 0$ $\longrightarrow$ $(x/z)^2 - (y/z) = 0$ $\xrightarrow{xz^2}$ $x^2 - yz = 0$

Hyperbola: $x^2 - y^2 - 1 = 0$ $\longrightarrow$ $(x/z)^2 - (y/z)^2 - 1 = 0$ $\xrightarrow{}$ $x^2 - y^2 - z^2 = 0$

In general:

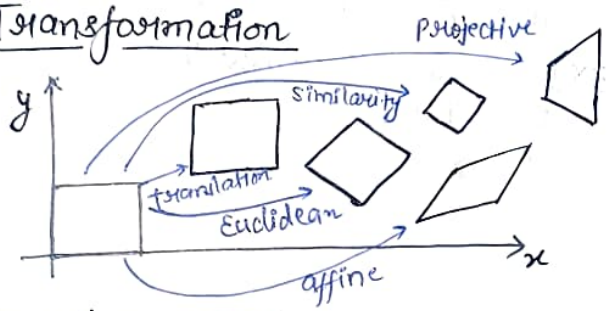
$ax^2 + by^2 + cxy + dx + ey + f = 0 \longrightarrow ax^2 + by^2 + cxy + dxz + eyz + fz^2 = 0$

## 3D Line




Parametric form: $(1-\lambda)P + \lambda Q$

## 2D Transformation




**Translation:** 2 DoF

$$x' = x + t \Leftrightarrow \bar{x}' = \begin{pmatrix} I & t \\ 0^T & 1 \end{pmatrix} \bar{x}$$

**Euclidean:** 2D Translation + 2D Rotation ; 3 DoF

$$x' = Rx + t \Leftrightarrow \bar{x}' = \begin{pmatrix} R & t \\ 0^T & 1 \end{pmatrix} \bar{x}$$

**Similarity:** 2D Translation + Scaled 2D Rotation ; 4 DoF

$$x' = sRx + t \Leftrightarrow \bar{x}' = \begin{bmatrix} sR & t \\ 0^T & 1 \end{bmatrix} \bar{x}$$

**Affine:** 2D Linear Transformation ; 6 DoF

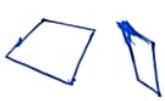
$$x' = Ax + t \Leftrightarrow \bar{x}' = \begin{bmatrix} A & t \\ 0^T & 1 \end{bmatrix} \bar{x}$$

**Projective:** Homography ; 8 DoF

$$\tilde{x}' = \tilde{H}\tilde{x} \quad \left(\bar{x} = \frac{1}{\tilde{w}}\tilde{x}\right)$$

### 2D Transformation Hierarchy:

Projective (8 DoF) $\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$



Affine (6 DoF) $\begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$

Similarity (4 DoF) $\begin{bmatrix} s r_{11} & s r_{12} & t_x \\ s r_{21} & s r_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$

Euclidean (3 DoF) $\begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$

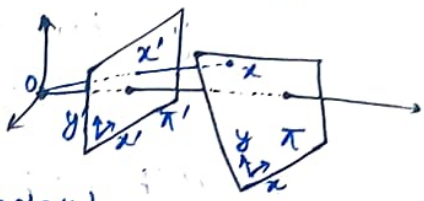
(3D) **2D Transformation:**

| Transformation | Matrix | # DoF | Preserves | Icon |
|---|---|---|---|---|
| Translation | $[I \; t]_{2\times3}$ $(3\times4)$ | 2 (3) | Orientation |  |
| Rigid (Euclidean) | $[R \; t]_{2\times3}$ $(3\times4)$ | 3 (6) | Length |  |
| Similarity | $[sR \; t]_{2\times3}$ $(3\times4)$ | 4 (7) | Angles |  |
| ~~Affine~~ | $[A]_{2\times3}$ $(3\times4)$ | 6 (12) | Parallelism |  |
| Projective | $[\tilde{H}]_{3\times3}$ $(4\times4)$ for 2D, for 3D | 8 (15) | Straight lines |  |

# ESTIMATING HOMOGRAPHY

## Homography (Plane-to-plane Mapping)

↳ Central projection maps points on one plane to points on another plane, and represented by a linear mapping of homogeneous coordinates $x' = Hx$.




$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$$ (upto a scalar factor)

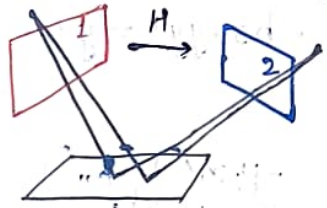
↳ 9 parameters

↳ 8 DoFs

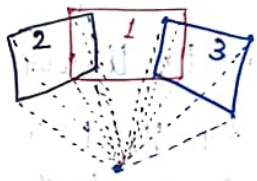
→ There are 2 kinds of homography expressed by the same form:

(i) Two-view homography

(ii) Homography induced by central projection.




Two-view homography




Homography induced by central projection

**Projectivity:** An invertible mapping $h$ from $P^2$ to itself such that three points $x_1, x_2, x_3$ lie on the same line if and only if $h(x_1), h(x_2), h(x_3)$ do.

↳ = collinearity

= projective transformation

= homography

**Setup homography equations:**

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \text{OR} \quad \begin{pmatrix} x_1' \\ x_2' \\ x_3' \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

$$\Rightarrow x' = h_{11}x + h_{12}y + h_{13}$$
$$y' = h_{21}x + h_{22}y + h_{23}$$
$$w' = h_{31}x + h_{32}y + h_{33}$$

[• Select 4 points in a plane with known coordinates.
• Form equations.]

Converting back to cartesian coordinates.

$$x' = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + h_{33}} \quad \text{(Divide by } w')$$

$$y' = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + h_{33}}$$

[OR, $x' = x_1'/x_3'$, $y' = x_2'/x_3'$]

$$\Rightarrow x'(h_{31}x + h_{32}y + h_{33}) = h_{11}x + h_{12}y + h_{13}$$
$$y'(h_{31}x + h_{32}y + h_{33}) = h_{21}x + h_{22}y + h_{23}$$

Set $h_{33} = 1$ to solve them.

[Remark: No calibration at all necessary;
Does not work if $h_{33} = 0$ in H]

$$\Rightarrow h_{11}x + h_{12}y + h_{13} - h_{31}xx' - h_{32}yx' = x'$$
$$h_{21}x + h_{22}y + h_{23} - h_{31}xy' - h_{32}yy' = y'$$

Convert to linear system.

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x_1' & -y_1x_1' \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y_1' & -y_1y_1' \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x_2' & -y_2x_2' \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y_2' & -y_2y_2' \\ \vdots & & & & & & & \\ x_n & y_n & 1 & 0 & 0 & 0 & -x_nx_n' & -y_ny_n' \\ 0 & 0 & 0 & x_n & y_n & 1 & -x_ny_n' & -y_ny_n' \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix} = \begin{bmatrix} x_1' \\ y_1' \\ x_2' \\ y_2' \\ \vdots \\ x_n' \\ y_n' \end{bmatrix}$$

$\Rightarrow \boxed{Ah = b}$, $A \rightarrow 2N \times 8$ matrix of source & dest. points.
$h \rightarrow$ 8 element homography vector
$b \rightarrow 2N$-dim. vector of transformed coordinates.

[One point pair provides 2 constraints,
8 DOF $\Rightarrow$ 4 point correspondences needed.]

## Solving H:

Assume $N(\geq 4)$ point correspondences are given:

$$\{\{(x_i, y_i), (x_i', y_i')\} \mid i = 1, \ldots, N\},$$

$\lambda_i$ is a point-dependent scalar factor.

$$\therefore \lambda_i \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix}$$

$$\Rightarrow \lambda_i (h_1 x_i + h_2 y_i + h_3) = x_i' \quad -(1)$$
$$\lambda_i (h_4 x_i + h_5 y_i + h_6) = y_i' \quad -(2)$$
$$\lambda_i (h_7 x_i + h_8 y_i + h_9) = 1 \quad -(3)$$

$$\Rightarrow \lambda_i = 1/(h_7 x_i + h_8 y_i + h_9) \quad [\text{from } (3)] \quad -(4)$$

Substitute $\lambda_i$ in (1), (2),

$$\Rightarrow h_1 x_i + h_2 y_i + h_3 = x_i' (h_7 x_i + h_8 y_i + h_9)$$
$$h_4 (x_i) + h_5 y_i + h_6 = y_i' (h_7 x_i + h_8 y_i + h_9)$$

In matrix form,

$$\begin{bmatrix} x_i & y_i & 1 & 0 & 0 & 0 & -x_i' x_i & -y_i' y_i & -x_i' \\ 0 & 0 & 0 & x_i & y_i & 1 & -y_i' x_i & -y_i' y_i & -y_i' \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \\ h_9 \end{bmatrix} = 0$$

[N points $\equiv$ 2N equations]

$$\Rightarrow \boxed{Ah = 0} \quad -(5)$$

Because $h$ is homogeneous, the solutions $h$ and $\alpha h$ make no difference for any $\alpha \neq 0$.

(5) have 2N eq.ns with 9 unknowns, exact soln cannot always exist.

Thus, solve the least-squared-error solution:

$$\min_h \|Ah\|^2, \text{ such that } \|h\|^2 = 1.$$

$$\text{or, } \arg\min_h \frac{\|Ah\|^2}{\|h\|^2} = \frac{h^T A^T A h}{\|h\|^2} \quad -(6)$$

According to the principle of Rayleigh quotient, the soln of (6) is the eigenvector corresponding to the smallest eigenvalue of matrix $A^TA$.

Thus, the soln can be obtained by performing the eigen-decomposition, $A^TA = PDP^T$, where $D \in R^{N \times N}$ is a diagonal matrix consisting of eigenvalues (w.l.o.g., from large to small) of matrix $A^TA$.

The soln is the last column of P (i.e., the eigenvector corresponding to the smallest eigenvalue of matrix $A^TA$).

↳ can be computed by performing singular value decomposition (SVD), $A = U\Sigma V^T$
(if we arrange the singular values are from large to small (descending order) in the matrix $\Sigma$).

↳ The soln is then the last column of V.

## Singular Value Decomposition

↳ Factorization of a real or complex matrix into a rotation, followed by a scaling followed by another rotation.

↳ It generalizes the eigendecomposition of a square normal matrix with an orthonormal eigenbasis to any mxn matrix.

Applications:

(i) Image compression: Reduces interpixel redudancy.
(ii) Reduce data from 3D to 2D – dimensionality reduction.
(iii) Projecting data to different planes – linear transformations

Factorization:

$$A_{m \times n} = U_{m \times m} \Sigma_{m \times n} V^*_{n \times n},$$

U: mxm orthogonal matrix
V: nxn orthogonal matrix
$\Sigma$: mxn with non-zero element along the diagonal.

- The singular values appear along the main diagonal of $\Sigma$.
- They appear in descending order: $\sigma_1 > \dots \sigma_n > 0$
  where, $\sigma_1^2 > \dots > \sigma_n^2 > 0$ are eigenvalues of $AA^T$ & $A^TA$.

Assuming $m \geq n$.

To factorize A, consider $A^TA$ and $AA^T$.

**Properties:** $A^TA$ and $AA^T$ are symmetric matrices.

$$A^TA = (A^TA)^T = A^T(A^T)^T = A^TA$$

**SVD:**

- Calculate $AA^T$, $A^TA$.
- Find eigenvalues and $\Sigma$.
- Calculate $U$ and $V$.
- Find the SVD factorization form $- U\Sigma V^T$.

Eg $A = \begin{bmatrix} 2 & 2 \\ -1 & 1 \end{bmatrix} \Rightarrow A^T = \begin{bmatrix} 2 & -1 \\ 2 & 1 \end{bmatrix}$

step-①: calculate $AA^T$ and $A^TA$.

$$AA^T = \begin{pmatrix} 2 & 2 \\ -1 & 1 \end{pmatrix}\begin{pmatrix} 2 & -1 \\ 2 & 1 \end{pmatrix} = \begin{pmatrix} 8 & 0 \\ 2 & 0 \end{pmatrix}$$

$$A^TA = \begin{pmatrix} 2 & -1 \\ 2 & 1 \end{pmatrix}\begin{pmatrix} 2 & 2 \\ -1 & 1 \end{pmatrix} = \begin{pmatrix} 5 & 3 \\ 3 & 5 \end{pmatrix}$$

step-②: Eigenvalues and $\Sigma$.

$$|AA^T - \lambda I| = 0, \quad |A^TA - \lambda I| = 0$$

$$\hookrightarrow \left|\begin{pmatrix} 8 & 0 \\ 0 & 2 \end{pmatrix} - \lambda\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\right| = 0$$

$$\Rightarrow \begin{vmatrix} 8-\lambda & 0 \\ 0 & 2-\lambda \end{vmatrix} = 0$$

$$\Rightarrow (8-\lambda)(2-\lambda) = 0$$

$$\Rightarrow \lambda_1 = 8, \lambda_2 = 2.$$

$\therefore$ Singular values: $\sigma_1 = \sqrt{\lambda_1} = \sqrt{8}$

$$\sigma_2 = \sqrt{\lambda_2} = \sqrt{2}$$

Therefore, $\Sigma = \begin{bmatrix} \sqrt{8} & 0 \\ 0 & \sqrt{2} \end{bmatrix}$

step-③: Finding $U$.

$$(AA^T - \lambda I)x = 0$$

$$\Rightarrow \begin{pmatrix} 8 & 0 \\ 0 & 2 \end{pmatrix} - \lambda_1\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}x_1 = 0, \quad \begin{pmatrix} 8 & 0 \\ 0 & 2 \end{pmatrix} - \lambda_2\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}x_2 = 0$$

$$\Rightarrow \begin{pmatrix} 8-\lambda_1 & 0 \\ 0 & 2-\lambda_1 \end{pmatrix}x_1 = 0 \qquad \Rightarrow \begin{pmatrix} 8-\lambda_2 & 0 \\ 0 & 2-\lambda_2 \end{pmatrix}x_2 = 0$$

$$\Rightarrow \begin{pmatrix} 8-8 & 0 \\ 0 & 2-8 \end{pmatrix}x_1 = 0 \qquad \Rightarrow \begin{pmatrix} 8-2 & 0 \\ 0 & 2-2 \end{pmatrix}x_2 = 0$$

$$\Rightarrow \begin{pmatrix} 0 & 0 \\ 0 & -6 \end{pmatrix}x_1 = 0 \qquad \Rightarrow \begin{pmatrix} 6 & 0 \\ 0 & 0 \end{pmatrix}x_2 = 0$$

$$\Rightarrow x_1 = \begin{pmatrix} -1 \\ 0 \end{pmatrix}. \qquad \Rightarrow x_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

$\Rightarrow x_1 = \begin{bmatrix} -1 \\ 0 \end{bmatrix}, \quad x_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

$\Rightarrow u_1 = \frac{x_1}{\|x_1\|}$, & $u_2 = \frac{x_2}{\|x_2\|}$

$\rightarrow u_1 = \begin{bmatrix} -1 \\ 0 \end{bmatrix}, \quad u_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

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

Step-④: Finding V.

Similarly, $(A^TA - \lambda I)x = 0$

$\Rightarrow v_1 = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}, \quad v_2 = \begin{bmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$

$\therefore V = \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} \\ 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}$

Step-⑤: Complete SVD

$A = U \Sigma V^T$

$$\begin{bmatrix} 2 & 2 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \sqrt{8} & 0 \\ 0 & \sqrt{2} \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}$$

SVD compression:

→ SVD can compress any form of data.

→ SVD takes a matrix (square or non-square) and divides it into 2 orthogonal and a diagonal matrix as a sum of much simpler rank one matrices.

→ This allows to approximate the input matrix.

→ This new factorized matrices can represent the original matrix with lesser space.

$A = U \Sigma V'$

$= [u_1 \dots u_n] \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_n \end{bmatrix} \begin{bmatrix} v_1^T \\ \vdots \\ v_n^T \end{bmatrix}$

$= u_1 \sigma_1 v_1^T + \dots + u_n \sigma_n v_n^T$

$= \sum_{i=1}^{n} \sigma_i u_i v_i^T$

$\sigma_1 \geq \dots \geq \sigma_n$ (first few terms have largest impact on total sum).

$\Rightarrow$ We can approximate A by adding only the first few terms.

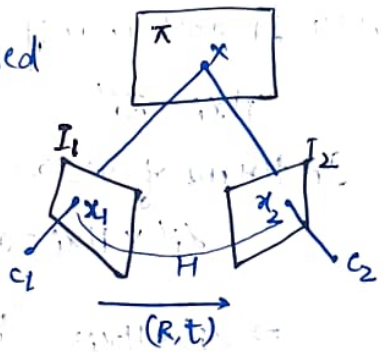
## Homography Estimation:

1. Use the known points to get matrix A. [$Ah=0$]
2. Use SVD in numpy.linalg to get matrix V.
3. The last column of V will be matrix h.
4. Normalize $h_9$ to 1 and reshape to 3×3.
5. Get matrix H.

**Applications:** Image Mosaicing → stitching multiple overlapping images to get a single, larger image with a wider field of view.

## Existence of Projective Homography

### ① Planar scene:




$x_1$ and $x_2$ is the 3D point X expressed in $C_1$ and $C_2$, respectively.

$$x_2 = Rx_1 + t.$$

$N = [n_1, n_2, n_3]^T$ is the unit normal vector representing the plane $\pi$ w.r.t. $C_1$, and d is the $\perp^{r}$ distance from plane to $C_1$:

$$N^T x_1 = n_1 X + n_2 Y + n_3 Z = d;$$

$$\Rightarrow \frac{N^T x_1}{d} = 1 \quad \forall\, x_1 \in \pi.$$

Combining two eqns, we get

$$x_2 = \left(R + \frac{tN^T}{d}\right) x_1,$$

since $\lambda_1 x_1 = x_1$ and $\lambda_2 x_2 = x_2$, we get

$$\lambda x_2 = \underbrace{\left(R + \frac{tN^T}{d}\right)}_{H} x_1$$

### ② Plane at infinity:

Scene is very far away from the camera (eg. aerial images).

$$H = \left(R + \frac{tN^T}{d}\right) \Rightarrow H_\infty = \lim_{d\to\infty}\left(R + \frac{tN^T}{d}\right) = R.$$

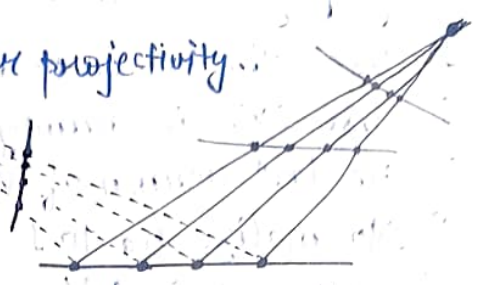
This is same as pure rotation, i.e., $t = (0,0,0)^T$.

$$\Rightarrow H = R.$$

## Projective transformation b/w lines:

→ Cross ratio is an invariant under projectivity..

→ Cross ratio will have the same value for all the 4 sets of collinear points.



Cross ratio,

$$\text{Cross}(\bar{x}_1, \bar{x}_2, \bar{x}_3, \bar{x}_4) = \frac{|\bar{x}_1 \bar{x}_2||\bar{x}_3 \bar{x}_4|}{|\bar{x}_1 \bar{x}_3||\bar{x}_2 \bar{x}_4|}$$

↓ Invariant under any projective transformation.

where $|\bar{x}_i \bar{x}_j| = \det \begin{bmatrix} x_{i1} & x_{j1} \\ x_{i2} & x_{j2} \end{bmatrix}$

## Direct Linear Transform (DLT):

$Hx_i = x_i'$ (Given set of 4 point correspondences: $x_i \leftrightarrow x_i'$).

In terms of vector cross product,

$x_i' \times Hx_i = 0$, where $Hx_i = \begin{pmatrix} h^{1T} x_i \\ h^{2T} x_i \\ h^{3T} x_i \end{pmatrix}$, $x_i' = (x_i', y_i', w_i')^T$

$$\Rightarrow x_i' \times Hx_i = \begin{pmatrix} y_i' h^{3T} x_i - w_i' h^{2T} x_i \\ w_i' h^{1T} x_i - x_i' h^{3T} x_i \\ x_i' h^{2T} x_i - y_i' h^{1T} x_i \end{pmatrix}$$

As $h^{jT} x_i = x_i^T h^j$, $j = 1, \ldots, 3$,

$$\begin{pmatrix} 0^T & -w_i' x_i^T & y_i' x_i^T \\ w_i' x_i^T & 0^T & -x_i' x_i^T \\ -y_i' x_i^T & x_i' x_i^T & 0^T \end{pmatrix} \begin{pmatrix} h^1 \\ h^2 \\ h^3 \end{pmatrix} = 0$$

(Only 1st 2 rows are independent)

$$\Rightarrow \begin{pmatrix} 0^T & -w_i' x_i^T & y_i' x_i^T \\ w_i' x_i^T & 0^T & -x_i' x_i^T \end{pmatrix} \begin{pmatrix} h^1 \\ h^2 \\ h^3 \end{pmatrix} = 0$$

$$\Rightarrow A_i h = 0$$

[The 3rd row is obtained, upto scale, from sum of $x_i'$ times 1st row and $y_i'$ times the second.

$A_i \rightarrow 2 \times 9$ matrix

$h \rightarrow$ a 9-vector made up of all elements in H, i.e., $h = \begin{pmatrix} h^1 \\ h^2 \\ h^3 \end{pmatrix}$, $H = \begin{pmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{pmatrix}$. with $h_i$ the $i^{th}$ element of $h_i$ $w_i$ chosen 1.

$$a \times b = \begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix}$$

$$= (a_2 b_3 - a_3 b_2)\hat{i} - (a_1 b_3 - a_3 b_1)\hat{j} + (a_1 b_2 - a_2 b_1)\hat{k}$$

## Least square solutions:

↳ Point correspondences are corrupted with noise.

↳ An exact soln for $Ah=0$ does not exist!

$\underset{h}{\text{minimize}} \|Ah\|$ subject to $\|h\|=1$.

↳ Obtained by taking the 9-vector right-null-space of A.

**Right null-space:** Right singular value that corresponds to the smallest singular value ($\sigma_9$) in the SVD of A ($v_9$),

$$SVD(A) = [u_1, u_2, \ldots, u_{2i}] \begin{bmatrix} \sigma_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_9 \\ \vdots & & \vdots \\ 0 & 0 & 0 \end{bmatrix} [v_1, v_2, \ldots, v_9]^T$$

left singular vector ($2i \times 2i$); singular values ($2i \times 9$); right singular vectors ($9 \times 9$)

$m > n$, $\text{rank}(A) = r$.

$\sigma_{n-r}, \ldots, \sigma_n = 0$, i.e., $\text{rank}(A) = r$, if A is not corrupted by noise and an exact soln for $Ah=0$ exists!

If A is noise corrupted, $\sigma_{n-r}, \ldots, \sigma_n \neq 0$.

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

$Av_i = u_i \sigma_i$

$\|Av_i\|$ is minimized when $u_i \sigma_i$ is at its min$^m$, i.e., smallest singular value, i.e., $\sigma_n$.

The soln of $\underset{h}{\text{argmin}} \|Ah\|$ s.t. $\|h\|=1$ is given by setting $h = v_n$. (last column of V).

**Degeneracy in Homography Estimation:** $[Ah=0]$.

↳ Rank of A drops below 8 if 3 of the min$^m$ 4 points correspondences are collinear.

↳ cannot solve for h ⇒ critical configuration or degeneracy.

→ A projective transformation must preserve collinearity.

**Data Normalization:** Carried out by a transformation of points:

(i) Points are translated so that their centroid is at origin.

(ii) Points are then scaled so that the average distance from origin is equal to $\sqrt{2}$.

(iii) Transformation is then applied to each of 2 images independently.

↳ Average point is equal to $(1,1,1)^T$ after normalization.

↳ No magnitude difference in linear eqn $Ah=0$.

→ Data normalization is an essential step in DLT algorithm.
↳ Must not be considered optional.

## Distances in DLT:

### Algebraic distance:

DLT minimizes $\|Ah\|$.
$e = Ah$ → residual error.
$e_i$ → partial error for each $x_i \leftrightarrow x_i'$

algebraic distance:

$$d_{alg}(x_i', Hx_i)^2 = \|e_i\|^2 = \left\| \begin{bmatrix} 0^T & -w_i' x_i^T & -y_i' x_i^T \\ -w_i' x_i^T & 0^T & -x_i' x_i^T \end{bmatrix} h \right\|^2$$

$d_{alg}(x_1, x_2)^2 = a_1^2 + a_2^2$, where $a = (a_1, a_2, a_3)^T = x_1 \times x_2$

$$\sum_i d_{alg}(x_i', Hx_i)^2 = \sum_i \|e_i\|^2 = \|Ah\|^2 = \|e\|^2$$

### Geometric distance:

Error in one image,

$$\hat{H} = \underset{H}{\operatorname{argmin}} \sum_i d(x_i', H\bar{x}_i)^2$$

$x$ : measured coordinates
$\hat{x}$ : estimated coordinates
$\bar{x}$ : true coordinates.
$d(\cdot,\cdot)$ → euclidean distance (in image)

Symmetric transfer error,

$$\hat{H} = \underset{H}{\operatorname{argmin}} \sum_i d(x_i, H^{-1}x_i')^2 + d(x_i', Hx_i)^2$$

Reprojection error,

$$(\hat{H}, \hat{x}_i, \hat{x}_i') = \underset{H, \hat{x}, \hat{x}_i'}{\operatorname{argmin}} \sum_i d(x_i, \hat{x}_i)^2 + d(x_i', \hat{x}_i')^2,$$

subject to $\hat{x}_i' = \hat{H}\hat{x}_i$.

Transfer error,

$$\sum_i d(x_i', Hx_i)^2$$

Sampson error:

$C_H(x) = 0$ : cost fn $Ah = 0$ satisfied by pt. $x = (x, y, x', y')^T$
$\hat{x}$ : desired pt. such that $C_H(\hat{x}) = 0$, where $\delta_x = \hat{x} - x$.
By Taylor expansion,

$$C_H(x + \delta_x) = C_H(x) + \left(\frac{\partial C_H}{\partial x}\right)\delta_x = 0.$$

Approximated cost fn: $J\delta_x = -\epsilon$,
$J$: ~~do~~ partial-derivative matrix
$\epsilon$ : cost $C_H(x)$ associated with $x$.

# Camera Calibration and Pose Estimation (Camera Localization)

## Camera Calibration

↳ Process to determine
- ↳ extrinsic parameters $(R, T)$ of a camera.
- ↳ intrinsic parameters ($K$ plus lens distortion).

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \underbrace{[R|T]}_{\text{extrinsic matrix}} \overbrace{\begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}}^{\text{point in camera frame}} = \underbrace{\begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix}}_{K: \text{intrinsic/ calibration matrix}} \underbrace{\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix}}_{\text{pt. in the camera frame}}$$

→ Estimating of extrinsic parameters with known intrinsic parameters: camera localization.

→ Neglect lens distortion (eg, barrel distortion) for now.

Mapping a 3-D world point to image point:

$$x = PX$$

($x$: pixel coordinate; $P$: transformation; $X$: world coordinate)

Wanted: Extrinsic & intrinsic parameters of a camera.

Given: Coordinates of object points (control points).

Observed: Coordinates of those known 3D object points in the image.

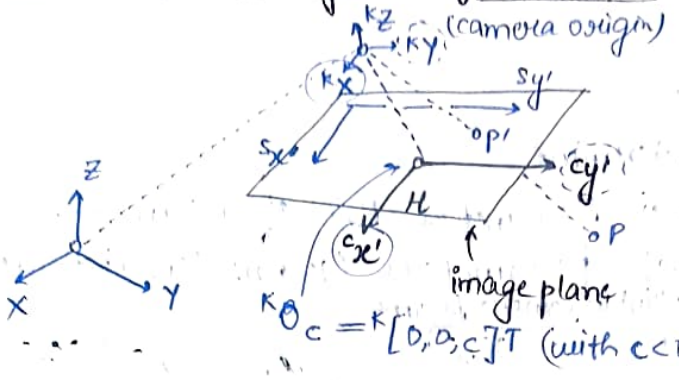
### Coordinate systems:

1. world/object coordinate system $S_o$ : $[X, Y, Z]^T$ (no index)
2. camera coordinate system $S_K$ : $[{}^K X, {}^K Y, {}^K Z]^T$
3. Image plane coordinate system $S_C$ : $[{}^C x, {}^C y]^T$
4. Sensor coordinate system $S_S$ : $[{}^S x, {}^S y]^T$.

$$\begin{bmatrix} {}^S x \\ {}^S y \\ 1 \end{bmatrix} = {}^S H_C \cdot {}^C P_K \cdot {}^K H_O \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

(in sensor system; ${}^S H_C$: image plane to sensor; ${}^C P_K$: camera to image; ${}^K H_O$: object to camera; in the object system)

## x and y axis coordinate systems of K and C:

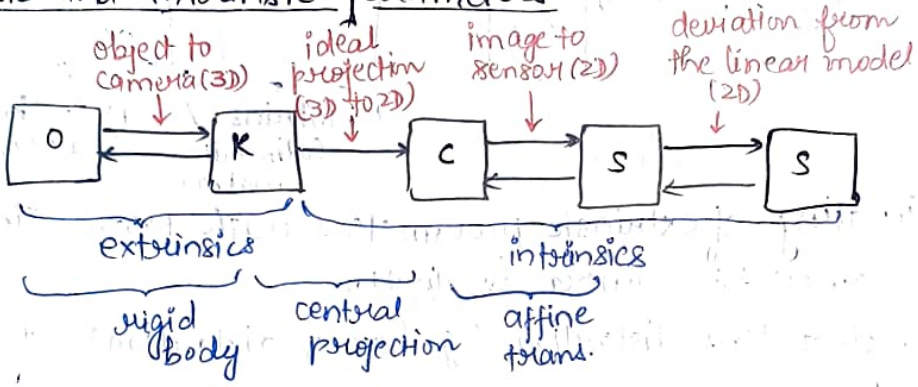



$^{K}O_C = {}^{K}[0, 0, c]^T$ (with $c<0$)

→ Dim's of x- and y-axes in c.s. K and C are identical.

→ The origin of c.s. C expressed in K is (0, 0, c).

## Extrinsic and intrinsic parameters:




**Extrinsic parameters:** Describe the pose (= position and heading) of the camera w.r.t. world.

↳ Invertible transformations.

↳ 6 parameters: 3 for the position + 3 for heading.

- Point P in world coordinates: $X_P = [X_P, Y_P, Z_P]^T$
- Center O of the projection (camera coord. origin): $X_0 = [X_0, Y_0, Z_0]^T$ (or $Z$ or $Z_0$)

→ In Euclidean coordinate systems:

- Translation b/w origin of world c.s. and camera c.s.:
$$X_0 = [X_0, Y_0, Z_0]^T$$
- Rotation R from $S_0$ to $S_K$
$$^{K}X_P = R(X_P - X_0)$$

→ In homogeneous c.s.,

$$\begin{bmatrix} {}^{K}X_P \\ 1 \end{bmatrix} = \begin{bmatrix} R & 0 \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} I_3 & -X_0 \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} X_P \\ 1 \end{bmatrix}$$

Euclidian H.C.

$$= \begin{bmatrix} R & -RX_0 \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} X_P \\ 1 \end{bmatrix}$$

or, $^{K}X_P = {}^{K}H\, X_P$ with $^{K}H = \begin{bmatrix} R & -RX_0 \\ 0^T & 1 \end{bmatrix}$

**Intrinsic parameters**: Projecting points from camera c.s. to sensor c.s.

↳ Invertible : image plane to sensor, model deviations

↳ Non-invertible : central projection.

→ 3 steps mapping:

① Ideal projection (perspective) to the image plane.

② Mapping to the ~~s.c system~~ sensor c.s. (where the pixels are).

③ Compensation for the fact that the 2 previous mappings are idealized.

→ Assumptions:

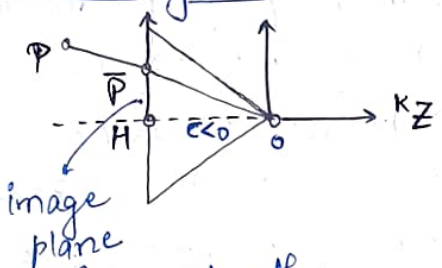
① Distortion-free lens.

② All rays are straight lines and pass through the projection center (origin of the camera c.s. $S_K$).

③ Focal point and principal point lie on the optical axis.

④ Distance from the camera origin to the image plane is the constant 'c'.

Image coordinate systems:




↔ Most popular image c.s. (Rotated 180° from physically motivated c.s.)

Through Intercept theorem,

$$^c x_{\bar{P}} := {}^K x_{\bar{P}} = c \frac{{}^K X_P}{{}^K Z_P}$$

$$^c y_{\bar{P}} := {}^K y_{\bar{P}} = c \frac{{}^K X_P}{{}^K Z_P}$$

$$\left( c = {}^K z_{\bar{P}} = c \frac{{}^K Z_P}{{}^K Z_P} \right)$$

In homogeneous c.s.,

$$\begin{bmatrix} {}^K u_{\bar{P}} \\ {}^K v_{\bar{P}} \\ {}^K w_{\bar{P}} \\ {}^K T_{\bar{P}} \end{bmatrix} = \begin{bmatrix} c & 0 & 0 & 0 \\ 0 & c & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} {}^K X_P \\ {}^K Y_P \\ {}^K Z_P \\ 1 \end{bmatrix}$$

↓ Drop the 3rd row

$$\begin{bmatrix} c x_{\bar{P}} \\ c y_{\bar{P}} \\ 1 \end{bmatrix} = \begin{bmatrix} {}^c u_{\bar{P}} \\ {}^c v_{\bar{P}} \\ {}^c w_{\bar{P}} \end{bmatrix} = \begin{bmatrix} c & 0 & 0 & 0 \\ 0 & c & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} {}^K X_P \\ {}^K Y_P \\ {}^K Z_P \\ 1 \end{bmatrix} = \begin{bmatrix} c {}^K X_P \\ c {}^K Y_P \\ {}^K Z_P \end{bmatrix} = \begin{bmatrix} c \frac{{}^K X_P}{{}^K Z_P} \\ c \frac{{}^K Y_P}{{}^K Z_P} \\ 1 \end{bmatrix}$$

Thus, for any point,

$${}^c x_P = {}^c P_K \, {}^K X_P, \quad \text{with } {}^c P_K = \begin{bmatrix} c & 0 & 0 & 0 \\ 0 & c & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \rightarrow$$

This gives,

$${}^c x = {}^c P X \quad \text{with } {}^c P = {}^c P_K \, {}^K H = \begin{bmatrix} c & 0 & 0 & 0 \\ 0 & c & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & -RX_0 \\ 0^T & 1 \end{bmatrix}$$

Calibration matrix (for ideal camera): $${}^c K = \begin{bmatrix} c & 0 & 0 \\ 0 & c & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} I_3 \mid -X_0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & -X_0 \\ 0 & 1 & 0 & -Y_0 \\ 0 & 0 & 1 & -Z_0 \end{bmatrix}$$

$I_3$ $\quad -X_0$

∴ Overall mapping:

$${}^c P = {}^c K \begin{bmatrix} R \mid -RX_0 \end{bmatrix} = {}^c K R \begin{bmatrix} I_3 \mid -X_0 \end{bmatrix}$$

Coordinates of ${}^c x$:

$$\begin{bmatrix} {}^c w' \\ {}^c v' \\ {}^c w' \end{bmatrix} = \begin{bmatrix} c & 0 & 0 \\ 0 & c & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} X - X_0 \\ Y - Y_0 \\ Z - Z_0 \end{bmatrix}$$

In Euclidean coordinates, this leads to collinearity eqⁿ for the image coordinates.

$${}^c x = c \, \frac{r_{11}(X - X_0) + r_{12}(Y - Y_0) + r_{13}(Z - Z_0)}{r_{31}(X - X_0) + r_{32}(Y - Y_0) + r_{33}(Z - Z_0)},$$

$${}^c y = c \, \frac{r_{21}(X - X_0) + r_{22}(Y - Y_0) + r_{23}(Z - Z_0)}{r_{31}(X - X_0) + r_{32}(Y - Y_0) + r_{33}(Z - Z_0)}.$$

**Mapping to the sensor (without linear errors):** Image → sensor

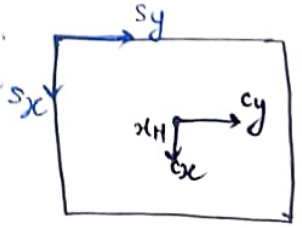
Linear errors:

① Location of the principal point in the image.
② scale difference in $x$ and $y$ based on the chip design.
③ shear compensation.

Location to principal point:

↳ The origin of the sensor system is not at the principal pt.
↳ Compensation through a shift.

$${}^S H_c = \begin{bmatrix} 1 & 0 & x_H \\ 0 & 1 & x_H \\ 0 & 0 & 1 \end{bmatrix}$$



## Shear and scale differences:

↳ scale difference m in x and y.

↳ Shear compensation s (for digital cameras, typically $s \approx 0$)

$$\therefore {}^{s}H_c = \begin{bmatrix} 1 & s & x_H \\ 0 & 1+m & y_H \\ 0 & 0 & 1 \end{bmatrix}$$

Hence, $s_x = {}^{s}H_c \cdot {}^{c}K R [I_3 | -X_0] X$

→ often, ${}^{s}H_c$ is combined with calibration matrix ${}^{c}K$.

$$K = {}^{s}H_c \, {}^{c}K = \begin{bmatrix} 1 & s & x_H \\ 0 & 1+m & y_H \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c & 0 & 0 \\ 0 & c & 0 \\ 0 & 0 & 1 \end{bmatrix} = \boxed{\begin{bmatrix} c & cs & x_H \\ 0 & c(1+m) & y_H \\ 0 & 0 & 1 \end{bmatrix}}$$

↓ affine transformation (K)

↳ 5 parameters:
- camera constant (c)
- principal point $(x_H, y_H)$
- scale difference (m)
- shear (s).

→ Affine camera: camera with an affine mapping to the sensor c.s. (after central projection is applied).

## Direct Linear Transform (DLT):

$x = PX$, with $P = KR[I_3 | -X_0]$ → model of affine camera

maps any object pt. X to the image pt. x.

↳ 11 parameters:
- 6 extrinsic $(R, X_0)$
- 5 intrinsic $(c, x_H, y_H, m, s)$

↳ Homogeneous projection matrix
↳ both in- and extrinsic parameters.

In Euclidean,

$$s_x = \frac{P_{11}X + P_{12}Y + P_{13}Z + P_{14}}{P_{31}X + P_{32}Y + P_{33}Z + P_{34}}$$

$$s_y = \frac{P_{21}X + P_{22}Y + P_{23}Z + P_{24}}{P_{31}X + P_{32}Y + P_{33}Z + P_{34}}$$

DLT maps obj. pt. X to image pt. x,

$$\underset{(3\times1)}{x} = \underset{(3\times3)}{K} \; \underset{(3\times3)}{R} \; \underbrace{\underset{(3\times3)}{[I_3} \; \underset{(3\times1)}{| -X_0]}}_{3\times4} \; \underset{(4\times1)}{X}$$

$$= \underset{(3\times4)}{P} \; \underset{(4\times1)}{X}$$

K → intrinsic
$X_0, R$ → extrinsic

# Computing the Orientation of an Uncalibrated camera:

Calibrated camera
- ↳ 6 unknowns
- ↳ Need atleast 3 points
- ↳ Solved by spatial resection

Uncalibrated camera
- ↳ 11 unknowns
- ↳ Need atleast 6 points
- ↳ Assuming the model of an affine camera.
- ↳ Problem solved by DLT.

**DLT eqn:**

$$x_i = \underset{(3\times4)}{P} X_i = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} X_i$$

∴ Define 3 vectors: $A = \begin{bmatrix} p_{11} \\ p_{12} \\ p_{13} \\ p_{14} \end{bmatrix}$, $B = \begin{bmatrix} p_{21} \\ p_{22} \\ p_{23} \\ p_{24} \end{bmatrix}$, $C = \begin{bmatrix} p_{31} \\ p_{32} \\ p_{33} \\ p_{34} \end{bmatrix}$.

Then, $x_i = PX_i = \begin{bmatrix} A^T \\ B^T \\ C^T \end{bmatrix} X_i = \begin{bmatrix} A^T X_i \\ B^T X_i \\ C^T X_i \end{bmatrix}$

$\Rightarrow x_i = \frac{u_i}{w_i} = \frac{A^T X_i}{C^T X_i}$, $y_i = \frac{v_i}{w_i} = \frac{B^T X_i}{C^T X_i}$

$\Rightarrow x_i C^T X_i - A^T X_i = 0$

$y_i C^T X_i - B^T X_i = 0$

$\Rightarrow -X_i^T A + x_i X_i^T C = 0$
$-X_i^T B + y_i X_i^T C = 0$
} system of linear eqns in parameters A, B, C.

$\Rightarrow a_{x_i}^T p = 0$
$a_{y_i}^T p = 0$
[collect elements of P with a parameter vector p]
↑ rows of P as column-vectors

$\Downarrow$

$Ax = 0$

$\Downarrow$

homogeneous system

$a_{x_i}^T = (-X_i^T, 0^T, x_i X_i^T)$

$a_{y_i}^T = (0^T, -X_i^T, y_i X_i^T)$

↳ solve using SVD (finding null space of A).

↳ No soln if all $X_i$ are on a plane.

- if all $x_i$ and projection center $X_0$ are located on a twisted cubic curve.

## Decomposition of P:

$P = [KR | -KRX_0] = [H | h]$

with $H = KR$, $h = -KRX_0$.

K → triangular matrix
R → rotation matrix

we get projection center through $X_0 = -H^{-1}h$.

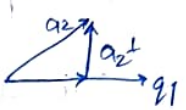
**H = KR structure:** QR-decomposition

Eg. $A = \begin{pmatrix} | & | & | \\ a_1 & a_2 & a_3 \\ | & | & | \end{pmatrix} \Rightarrow Q = \begin{pmatrix} | & | & | \\ q_1 & q_2 & q_3 \\ | & | & | \end{pmatrix}$

$R = \begin{pmatrix} \|a_1\| & \langle a_2, q_1 \rangle & \langle a_3, q_1 \rangle \\ 0 & \|a_2^\perp\| & \langle a_3, q_2 \rangle \\ 0 & 0 & \|a_3^\perp\| \end{pmatrix}$

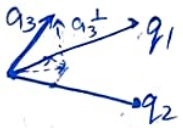
- $q_1 = \frac{a_1}{\|a_1\|}$ $(\because \|q_1\| = 1)$
- $a_2^\perp = a_2 - \langle a_2, q_1 \rangle q_1$

  $q_2 = \frac{a_2^\perp}{\|a_2^\perp\|}$




- $a_3^\perp = a_3 - \langle a_3, q_1 \rangle q_1 - \langle a_3, q_2 \rangle q_2$

  $q_3 = \frac{a_3^\perp}{\|a_3^\perp\|}$




$\therefore H^{-1} = (KR)^{-1} = R^{-1}K^{-1} = R^T K^{-1}$

($R^T$ → Q, $K^{-1}$ → R)

→ $H = KR$ → homogeneous matrix ⇒ calibration matrix

Normalize $K \leftarrow \frac{1}{K_{33}} K$ ↪ +ve diagonal elements

→ To get -ve camera constant,

choose $K \leftarrow KR(z, \pi)$, $R \leftarrow R(z, \pi)R$

using $R(z, \pi) = \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

## Camera calibration by Zhang's Method

$x = KR[I_3 | -X_0]X$ ↪ compute 5 intrinsic parameters.

- $x$: observed image point
- $K$: c, s, m, $x_H$, $y_H$
- $R$: 3 rotations
- $X_0$: 3 translations
- $X$: control point (given)

Using 2D checker board

↳ Observed 2D pattern

↳ Known size and structure

↳ Set the world coordinate system to the corner of the checkerboard.

↳ All points on the checkerboard lie in the XY plane, i.e., $Z = 0$.

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} c & cs & x_H \\ 0 & c(1+m) & y_H \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

→ Z coordinate of each pt. on the checkerboard = 0

↳ Delete 3rd column of extrinsic par. matrix.

$$\Downarrow$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} c & cs & x_H \\ 0 & c(1+m) & y_H \\ 0 & 0 & 1 \end{bmatrix}}_{K} \underbrace{\begin{bmatrix} r_{11} & r_{12} & t_1 \\ r_{21} & r_{22} & t_2 \\ r_{31} & r_{32} & t_3 \end{bmatrix}}_{[r_1, r_2, t]} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

$$H = [h_1, h_2, h_3]$$

For multiple observed pts.: $\begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \underset{3\times3}{H} \begin{bmatrix} X_i \\ Y_i \\ 1 \end{bmatrix}$, $i = 1, \ldots, I$

↳ Estimate 3×3 homography instead of 3×4 projection matrix.

↳ Solve to estimate H. → Need atleast 4 points ($H \rightarrow 8$ DoF) → 2 observations from each point.

↳ No direct decomposition as in DLT.

↳ Exploit constraints of the parameter.

$$[r_1, r_2, t] = K^{-1} [h_1, h_2, h_3]$$

$$\Rightarrow r_1 = K^{-1} h_1, \quad r_2 = K^{-1} h_2.$$

$r_1, r_2, r_3$ form an orthonormal basis

$$\Rightarrow r_1^T r_2 = 0, \quad \|r_1\| = \|r_2\| = 1.$$

$$r_1^T r_2 = 0 \Rightarrow \boxed{h_1^T K^{-T} K^{-1} h_2 = 0}$$

$$\|r_1\| = \|r_2\| = 1 \Rightarrow h_1^T K^{-T} K^{-1} h_1 = h_2^T K^{-T} K^{-1} h_2$$

$$\Rightarrow \boxed{h_1^T K^{-T} K^{-1} h_1 - h_2^T K^{-T} K^{-1} h_2 = 0}$$

Define symmetric and +ve definite matrix: $B = K^{-T} K^{-1}$

$$\Rightarrow h_1^T B h_2 = 0$$

$$h_1^T B h_1 - h_2^T B h_2 = 0$$

From B, the calibration matrix can be recovered through cholesky decomposition.

$$B = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{12} & b_{22} & b_{23} \\ b_{13} & b_{23} & b_{33} \end{pmatrix}$$

$$\text{chol}(B) = AA^T \Rightarrow A = K^{-T}$$

→ If we know B, we can compute K.

↳ B : unknowns

↳ h : known

↳ 2 equations relating B and h.

Cholesky decomposition:

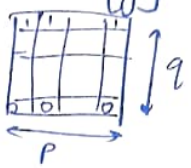
$$l_{ki} = a_{ki} - \sum_{j=1}^{i-1} l_{ij}\, l_{kj}, \quad l_{kk} = \sqrt{a_{kk} - \sum_{j=1}^{k-1} l_{kj}^2}$$

# Image Processing and Feature Extraction

## Image

↳ (grayscale) → 2D function ; domain $x = \begin{bmatrix} x \\ y \end{bmatrix}$

↳ Types : ① Binary → 0 : Black
1 : White




② Grayscale

③ Color

## Image Transformations

↳ Filtering : changes pixel values : $G(x) = h\{F(x)\}$ → changes range (original image)

↳ Warping : changes pixel locations : $G(x) = F(h\{x\})$ → changes domain of image function

Image filtering

↳ Neighborhood operation (filtering)

↳ Point operation (point processing)

- original : $x$
- Darken : $x - 128$
- Lower contrast : $x/2$
- Non-linear lower contrast : $\left(\frac{x}{255}\right)^{1/3} \times 255$
- Invert : $255 - x$
- Lighten : $x + 128$
- Raise contrast : $x \times 2$
- Non-linear raise contrast : $\left(\frac{x}{255}\right)^{2} \times 255$

→ Images are sampled.

**Resolution:** Sampling parameter

↳ defined in dots per inch (DPI) ; standard value : 72 dpi

↳ equivalent measure of spatial pixel density.

**Images are sampled and Quantized :**

↳ An image contains discrete no. of pixels.

Eg. Pixel value : grayscale (or intensity) : [0, 255]

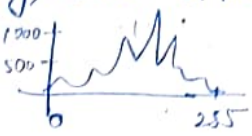
color : RGB : [R, G, B]
Lab : [L, a, b]
HSV : [H, S, V]

↳ Errors due to sampling and quantization.

**Histogram:** A type of image ✓

↳ Provides frequency of brightness (intensity) value in the image.

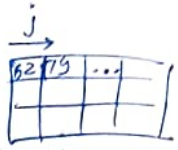
↳ captures the distribution of gray levels in the image (how frequent).




**Images as Discrete Functions:** Digital (discrete)

↳ sample the 2D space on a regular grid

↳ Represented as a matrix of integer values




Cartesian coordinates:

$$f(n,m) = \begin{bmatrix} \cdots & f[-1,-1] & f[0,-1] & f[1,-1] & \cdots \\ \cdots & f[-1,0] & f[0,0] & f[1,0] & \cdots \\ \cdots & f[-1,1] & f[0,1] & f[1,1] & \cdots \end{bmatrix}$$

Image: $f: \mathbb{R}^2 \rightarrow \mathbb{R}^M$

$f(x,y)$ → gives intensity at position $(x,y)$.

↳ Defined over a rectangle with a finite range:

$$f: \underbrace{[a,b] \times [c,d]}_{\text{domain support}} \rightarrow \underbrace{[0,255]}_{\text{range}}$$

↳ color image: $f(x,y) = \begin{bmatrix} r(x,y) \\ g(x,y) \\ b(x,y) \end{bmatrix}$

**Linear shift-invariant image filtering:**

↳ Replace each pixel by a linear combination of its neighbors (and possibly itself).

↳ The combination is determined by filter's kernel.

shifted to all pixel locations

**Systems and Filters**

Filtering → Form a new image whose pixel values are transformed from original pixel values.

↳ To extract useful info from images, or transform images into another domain where we can modify / enhance image properties.

↳ Features (edges, corners, blobs)

↳ Super-resolution, in-painting, de-noising.

$f[n,m]$ → [System S] → $g(n,m)$

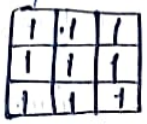
(i/p image) (filter) (o/p image)

$(n,m)$: spatial position in the image

Properties: $\text{filter}(im, f_1 + f_2) = \text{filter}(im, f_1) + \text{filter}(im, f_2)$ ...

$c * \text{filter}(im, f_1) = \text{filter}(im, c * f_1)$.

**Box Filter:** (or 2D rect filter or square mean filter)

Kernel: $g[\cdot,\cdot] = \frac{1}{9}$ 

↳ Replaces pixel with local average

↳ Has smoothing (blurring) effect. (approximate Gaussian blur)

**Convolution:**

1D continuous signals: $(f * g)(x) = \int_{-\infty}^{\infty} f(y)\, g(x-y)\, dy$

(flip; $f$: filtered signal, $g$: filter, $x-y$: i/p signal)

2D discrete signals: $(f * g)(x,y) = \sum_{i,j=-\infty}^{\infty} f(i,j)\, I(x-i, y-j)$

($f$: filtered image, $(i,j=-1)$, $I$: i/p image)

if $f(i,j) \neq 0$ within only $-1 \leq i,j \leq 1$.

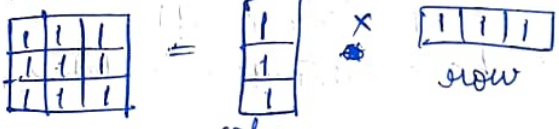
**Correlation:**

$$(f * g)(x,y) = \sum_{i,j=-\infty}^{\infty} f(i,j)\, I(x+i, y+i)$$

(no flip)

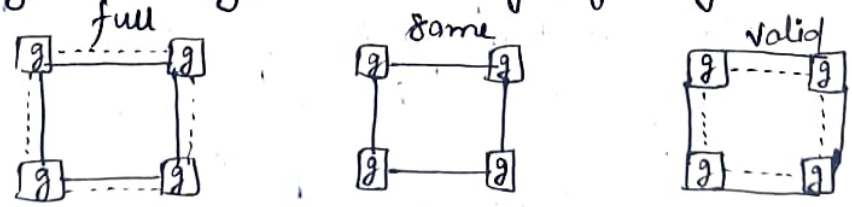
[Most kernels are symmetric.]

**Separable filters:** If it can written as a product of a column and a row.

Eg: box filter: 


→ 2D convolution with a separable filter is equivalent to two 1D convolutions.

**Handling boundary issues in image filtering:**




Border padding ↳ Zero pad
↳ Circular
↳ Replicate
↳ Symmetric.

## Moving Average Filter: → shift-invariant!

$$g[n,m] = \frac{1}{9}\sum_{k=n-1}^{n+1}\sum_{l=m-1}^{m+1} f[k,l]$$

$$= \frac{1}{9}\sum_{k=-1}^{1}\sum_{l=-1}^{1} f[n-k, m-l]$$

$\frac{1}{9}$ × $\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$

↳ Smoothing effect (remove sharp features).

**Image segmentation:** Based on a simple threshold. → Not linear

$$g[n,m] = \begin{cases} 255, & f[n,m] > 100 \\ 0, & \text{otherwise.} \end{cases}$$

## Properties of Systems

Linear Systems:

① Additivity: $S[f_i[n,m] + f_j[n,m]] = S[f_i[n,m]] + S[f_j[n,m]]$

② Homogeneity: $S[\alpha f_i[n,m]] = \alpha S[f_i[n,m]]$

③ Superposition: $S[\alpha f_i[n,m] + \beta f_j[n,m]] = \alpha S[f_i[n,m]] + \beta S[f_j[n,m]]$

④ Stability: $|f[n,m]| \le K \Rightarrow |g[n,m]| \le CK$.

⑤ Invertibility: $S^{-1}[S[f_i[n,m]]] = f[n,m]$.

⑥ Causality: For $n < n_0, m < m_0$, if $f[n,m] = 0 \Rightarrow g[n,m] = 0$

Shift-invariant Systems:

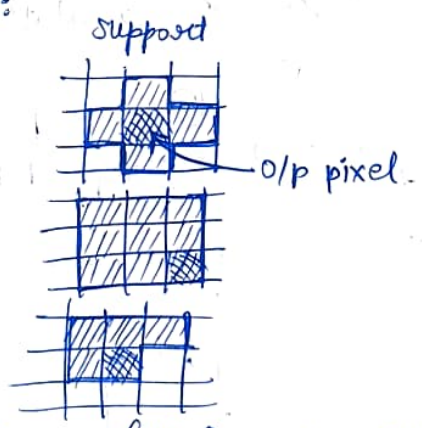
⑦ Shift Invariance: $f[n-n_0, m-m_0] \xrightarrow{S} g[n-n_0, m-m_0]$.

### Causality in images:

Non-causal:

Causal:

Semi-causal:

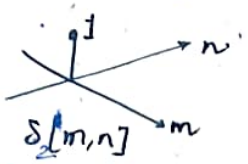



## Characterize a linear shift invariant (LSI) system

2D impulse function:

↳ 1 at [0,1]

↳ 0 everywhere else




Impulse response to moving filter:

$$\delta_2 \xrightarrow{S} h[n,m] = \frac{1}{9}\sum_{k=-1}^{1}\sum_{l=-1}^{1} \delta_2[n-k, m-l]$$

$$= \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

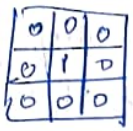
General LSI system: $f[n,m] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k,l] \times \delta_2[n-k, m-l]$

## 2D Convolution:

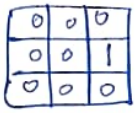
$$f[n,m] * h[n,m] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k,l]\, h[n-k, m-l]$$ → Filtering operation.

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

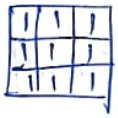
Eg. Kernel:



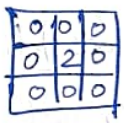
→ Filtered image ↓ no change

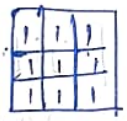


→ shifted right by 1 pixel

$\frac{1}{9}$ 

→ Blur.

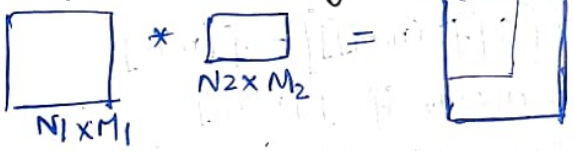


$- \frac{1}{9}$ 

→ Sharpening filter
↳ Accentuates differences with local average

## Image Support and Edge Effect:

Finite support signals convolution




$(N_1+N_2-1) \times (M_1+M_2-1)$

## Convolution as Matrix Multiplication:

① Define i/p and filter

$I = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ $(m_1 \times n_1)$

$F = \begin{bmatrix} 10 & 20 \\ 30 & 40 \end{bmatrix}$ $(m_2 \times n_2)$

② Calculate final o/p size:

$(m_1+m_2-1) \times (n_1+n_2-1)$

③ Zero-pad filter matrix

Zero padded $F = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 10 & 20 & 0 & 0 \\ 30 & 40 & 0 & 0 \end{bmatrix}$

④ Create Toeplitz matrix for each row of zeropadded filter.

$F_0 = \begin{bmatrix} 30 & 0 & 0 \\ 40 & 30 & 0 \\ 0 & 40 & 30 \\ 0 & 0 & 40 \end{bmatrix}$ ; $F_1 = \begin{bmatrix} 10 & 0 & 0 \\ 20 & 10 & 0 \\ 0 & 20 & 10 \\ 0 & 0 & 20 \end{bmatrix}$ , $F_2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

⑤ Create a doubly blocked Toeplitz matrix

$$\text{doubly blocked} = \begin{bmatrix} F_0 & 0 \\ F_1 & F_0 \\ F_2 & F_1 \end{bmatrix}$$

$$= \begin{bmatrix} 30 & 0 & 0 & 0 & 0 & 0 \\ 40 & 30 & 0 & 0 & 0 & 0 \\ 0 & 40 & 30 & 0 & 0 & 0 \\ 0 & 0 & 40 & 30 & 0 & 0 \\ 10 & 0 & 0 & 40 & 30 & 0 \\ 20 & 10 & 0 & 0 & 40 & 30 \\ 0 & 20 & 10 & 0 & 0 & 40 \\ 0 & 0 & 20 & 10 & 0 & 0 \\ 0 & 0 & 0 & 20 & 10 & 0 \\ 0 & 0 & 0 & 0 & 20 & 10 \\ 0 & 0 & 0 & 0 & 0 & 20 \end{bmatrix}$$

doubly blocked =

$$\begin{bmatrix} \begin{bmatrix} 30 & 0 & 0 \\ 40 & 30 & 0 \\ 0 & 40 & 30 \\ 0 & 0 & 40 \end{bmatrix} & \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \\ \begin{bmatrix} 10 & 0 & 0 \\ 20 & 10 & 0 \\ 0 & 20 & 10 \\ 0 & 0 & 20 \end{bmatrix} & \begin{bmatrix} 30 & 0 & 0 \\ 40 & 30 & 0 \\ 0 & 40 & 30 \\ 0 & 0 & 40 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 10 & 0 & 0 \\ 20 & 10 & 0 \\ 0 & 20 & 10 \\ 0 & 0 & 20 \end{bmatrix} \end{bmatrix}$$

⑥ Convert input matrix to a column vector.

$$I = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \Rightarrow \text{Vectorized } I = \begin{bmatrix} 4 \\ 5 \\ 6 \\ 1 \\ 2 \\ 3 \end{bmatrix}$$

⑦ Multiply doubly blocked toeplitz matrix with vectorized i/p signal.
↳ This gives the convolution result.

⑧ Reshape the result to a matrix form.

$$\text{result vector} = \begin{bmatrix} 120 \\ 310 \\ 380 \\ 240 \\ 70 \\ 230 \\ 330 \\ 240 \\ 10 \\ 40 \\ 70 \\ 60 \end{bmatrix} \Rightarrow \text{output} = \begin{bmatrix} 10 & 40 & 70 & 60 \\ 70 & 230 & 330 & 240 \\ 120 & 310 & 380 & 240 \end{bmatrix}$$

## 2D (Cross) correlation:

$$f[n,m] ** h[n,m] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[n,m] * h[n+k, m+l]$$ → Equivalent to convolution without the flip.

↳ Compares the similarity of two sets of data.
↳ Measure of relatedness of two signals.

# Template Matching, Gabor Filter and Edge Detection

## Gaussian Filter

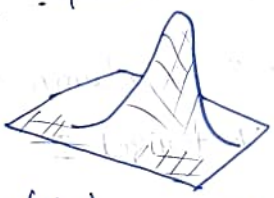
$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\left(\frac{x^2+y^2}{2\sigma^2}\right)}$ , $\sigma$ : std. deviation
↳ determines extent of smoothening.

$= \left(\frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-x^2}{2\sigma^2}\right)\right) \cdot \left(\frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-y^2}{2\sigma^2}\right)\right)$

↳ Infinite support, but discrete filters use finite kernels.
↳ Rule of thumb: set filter half-width to about $3\sigma$.

↓ Product of 2 Gaussians

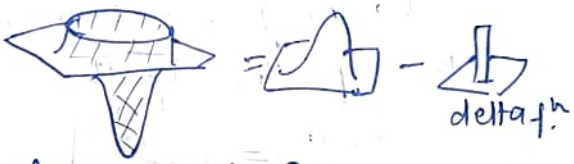
Gaussian: $h_\sigma(x,y)$



Derivative of Gaussian: $\frac{\partial}{\partial x} h_\sigma(x,y)$

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

Laplacian of Gaussian: $\nabla^2 h_\sigma(u,v)$




→ Gaussian smoothing → for removing Gaussian noise.
→ Reduces edge blurring
→ Linear low-pass filters.
→ Computationally efficient (implemented using small 1D filters)
→ Rotationally symmetric.
→ Degree of smoothing controlled by $\sigma$. ( $\sigma \uparrow$ for intensive smoothing. )
→ To create soft shadow effects.

## Gabor Filter

↳ Product of • complex sinusoid, $(c(x,y))$ [carrier]
• Gaussian function, $(g(x,y))$ [envelope].

$c(x,y) = e^{-j(2\pi F(x\cos\theta + y\sin\theta) + P)}$

$g(x,y) = \frac{1}{\sqrt{2\pi}\sigma} e^{-0.5\left(\frac{x_t}{\sigma_x^2} + \frac{y_t}{\sigma_y^2}\right)}$

$\begin{bmatrix} x_t = (x\cos\theta + y\sin\theta)^2 \\ y_t = (-x\sin\theta + y\cos\theta)^2 \end{bmatrix}$

$gabor(x,y) = c(x,y) * g(x,y) = e^{-\left(\frac{x^2+y^2}{2\sigma^2}\right)} \cos(2\pi\mu x) = \psi(x,y)$
(2D)

Carrier :

Gaussian envelope :

Gabor Kernal :

$\theta = 0°$ $\theta = 90°$ $\theta = 45°$

**Median Filter** → For removing salt and pepper noise

↳ Operates over a window by selecting the median intensity in the window.

Eg.

| | | |
|---|---|---|
| 10 | 15 | 20 |
| 23 | 90 | 27 |
| 33 | 31 | 30 |

Sort → 10 15 20 23 **27** 30 31 33 90

↑ Median value

↓ Replace

| | | |
|---|---|---|
| 10 | 15 | 20 |
| 23 | 27 | 27 |
| 33 | 31 | 30 |

**Template Matching**

↳ Find part of one image (instance of pattern) that matches another.

↳ Use pattern as template.

↳ convolution / correlation compares a template (filter) with each local image patch. → Dot product b/w filter & local image patch.

→ Normalized correlation → varies b/w −1 and 1.

↳ Attains 1 when filter & image region are identical.

# Edge Detection

↳ Identify sudden changes (discontinuities in an image).

↳ To extract information, recognize objects, recover geometry and viewpoint.

**Origin of edges:**

1. Surface normal discontinuity
2. depth discontinuity
3. surface color discontinuity
4. illumination discontinuity.

# Image Gradients

**Types of discrete derivative in 1D:**

| | | Filters |
|---|---|---|
| Backward: | $\frac{df}{dx} = f(x) - f(x-1) = f'(x)$ | $[0 \quad 1 \quad -1]$ |
| Forward: | $\frac{df}{dx} = f(x) - f(x+1) = f'(x)$ | $[-1 \quad 1 \quad 0]$ |
| central: | $\frac{df}{dx} = \frac{f(x+1) - f(x-1)}{2} = f'(x)$ | $[1 \quad 0 \quad -1]$ |

**Discrete Derivative in 2D:** Fn: $f(x,y)$

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

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

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

3x3 Image gradient filters: $\frac{1}{3}\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$ · $\frac{1}{3}\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$

**Characterizing Edges:**

↳ Place of rapid change in image intensity function.

**Effects of noise:**

↳ Finite difference filters respond strongly to noise.

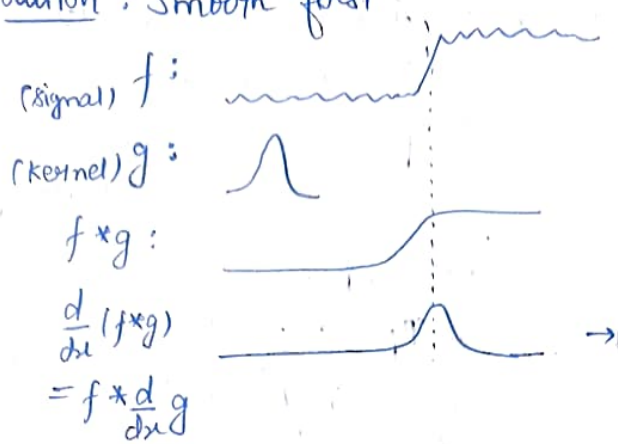
↳ Pixels look very different from their neighbors

↳ Fix: smooth the image → Mean, Gaussian, Median filters.

$f(x)$:

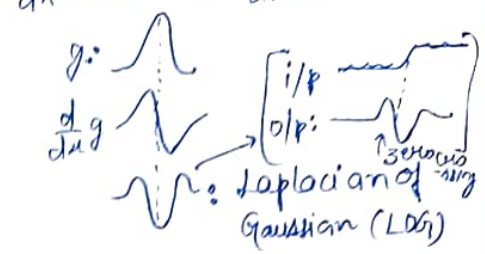
$f'(x)$: → edge info lost

Solution : Smooth first

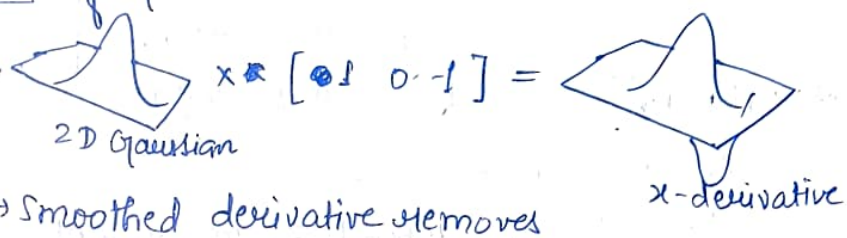



→ To find edges, look for peaks in $\frac{d}{dx}(f*g)$.

$\frac{d}{dx}(f*g) = f * \frac{d}{dx} g$




Derivative of Gaussian Filter :




→ Smoothed derivative removes noise, but blurs edges.

Steps for edge detection :

① select derivative filters.

$S_x =$ | 1 | 0 | -1 |
|---|---|---|
| 2 | 0 | -2 |
| 1 | 0 | -1 |

, $S_y =$ | 1 | 2 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| -1 | -2 | -1 |

② convolve with image to compute derivatives.

$$\frac{\partial f}{\partial x} = S_x \otimes f \,, \quad \frac{\partial f}{\partial y} = S_y \otimes f$$

③ Form image gradient and compute its direction and amplitude.

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right], \quad \theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right), \quad \|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

(gradient) (dirn) (amplitude).

Handling noise: Filter with a Gaussian to smooth, then take gradients.

Edge Adaptive Smoothing → ① Bilateral filter
② Non-local means filter.

$I_p$ : value of image I at position $p = (p_x, p_y)$.
$F[I]$ = o/p of filter F applied to image I.

Strategy for smoothing Images :

Box Average : $BA[I]_p = \sum_{q \in S} B_\sigma(p-q)\, I_q$

result at pixel p; sum over all pixels q; $B_\sigma(p-q)$ normalized box fn; $I_q$ → intensity at pixel q.

→ Square box generates defects → Axis-aligned streaks
↳ Blocky results.
↳ Use an isotropic (circular) window
↳ Use a window with a smooth falloff.
⇓
Gaussian Blur

$$GB[I]_p = \sum_{q \in S} \underbrace{G_\sigma(\|p-q\|)}_{\text{normalized gaussian fn}} I_q$$

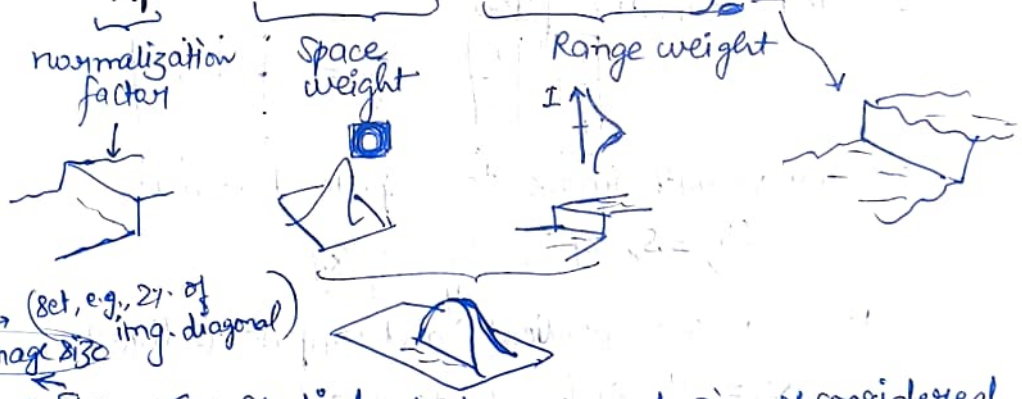
↳ Smoother too much: edges are blurred. (bc of averaging across edges).

$\sigma$ → size of window
↓
common setting: $\propto$ to image size (eg., 2% of image diagonal)
↳ independent of image resolution

## Bilateral Filter

↳ No averaging across edges.
↳ Kernel shape depends on image content.

$$BF[I]_p = \frac{1}{W_p} \sum G_{\sigma_s}(\|p-q\|)\, G_{\sigma_r}(|I_p - I_q|)\, I_q$$




$\propto$ to image size (set, e.g., 2% of img. diagonal)

Space $\sigma_s$: Spatial extent of kernel, size of considered neighborhood

Range $\sigma_r$: "minm" amplitude of an edge.
↳ Independent of resolution and exposure

proportional to edge amplitude (set, eg., mean or median of img. grads)

Bilateral → Averages neighbors with similar intensities.

## NL-Mean Filter

↳ Same goals: smooth within similar regions
↳ Generalize, extend 'similarity'
↳ Averages neighbors with similar neighborhoods.

→ For each and every pixel, p:
↳ define a small, simple fixed size neighborhood.
↳ define vector $V_p$ → a list of neighboring pixel values.

Similar pixels $p, q \Rightarrow$ small vector distance : $\|v_p - v_q\|^2$

Dissimilar pixels $p, q \Rightarrow$ large vector distance.

Filter with this : no spatial term.

$$NLMF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_r}(\|\vec{v}_p - \vec{v}_q\|^2) I_q$$

vector distance to p sets weight for each pixel q.

Noisy source image

↳ Gaussian filter ⇒ low noise, low detail

↳ Anisotropic Diffusion

↳ 'stairsteps' ~ piecewise constant

↳ Bilateral filter → better, but similar stairsteps.

↳ NL Means → sharp, low noise, few artifacts.

Bilateral goals → Local smoothing within similar regions.

↳ Edge preserving smoothing.

↳ Separate large structure & fine detail.

↳ Eliminate outliers.

↳ Filter within edges, not across them.

## Designing an Edge Detector

Criteria for "optimal" edge detector:

① Good detection → Minimize probability of false positives (detecting spurious edges caused by noise), as well as false negatives (missing real edges).

② Good localization : The edges detected must be as close as possible to the true edges.

③ single response : The detector must return one point only for each true edge point, i.e., minimize the no. of local maxima around the true edge.

# Sobel Edge Detector

## Sobel operator:

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

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

↑ Gaussian smoothing

↑ differentiation

## Sobel operation:

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

Dirn of gradient, $\theta = \tan^{-1}\left(\frac{G_y}{G_x}\right)$.

## Sobel Filter problems:

↳ Poor localization (trigger response in multiple adjacent pixels)

↳ Threshold value favors certain dirn's over others.

↳ can miss oblique edges more than horizontal or vertical edges.

↳ False negatives.

**Unsharp Masking:** Smoothing kernels used to sharpen images.

$$g_{sharp} = f + \gamma (f - h_{blur} * f)$$

## Canny Edge Detector → Most widely used

1. Suppress Noise (Gaussian smoothing & sobel grad. operator) [Detection]
2. Compute gradient magnitude and direction (~~Sobel grad operator~~) [sobel operator]
3. Apply Non-Maximum Suppression (localization)
   ↳ Assures minimal response.
4. Use hysteresis and connectivity analysis to detect edges.
   ↳ Single edge point.

Smooth → $n_\sigma * I$

Gradient → $\nabla n_\sigma * I$

Mag. → $\| \nabla n_\sigma * I \|$

Dirn → $\hat{n} = \frac{\nabla n_\sigma * I}{\| \nabla n_\sigma * I \|}$

→ Compute Laplacian along Grad. dirn $\hat{n}$ at each pixel:

$$\frac{\partial^2 (n_\sigma * I)}{\partial \hat{n}^2}$$

## Non-Maximum suppression:

- Edges occurs where gradient reaches a maxima.
- Suppress non-maxima gradient even if it passes threshold.
- Only 8 angle dirns possible.
  - Suppress all pixels in each dirn which are not maxima.
    - Do this in each marked pixel neighborhood.

## Double threshold:

- High threshold to identify the strong pixels (intensity > high thresh.)
- Low threshold to identify non-relevant pixels. (int. < low thresh.).
- All pixels having intensity in b/w are flagged as weak and Hysteresis mechanism helps identify the ones to be considered as strong and ones as non-relevant.

**Hysteresis Threshold:** Transform weak pixels into strong iff atleast one of the pixels around the one being processed is a strong one.

- Declare an edge pixel if it is (weak) connected to an strong edge pixel directly or via pixels b/w low & high.

## Hough Transform

- To find the location of lines in images.
- Can detect lines, circles, and other structures only if their parametric eqn is known.
- Can give robust detection under noise & partial occlusion.

**Prior to H.T.:** Perform edge detection and a thresholding of some edge mag. image ⇒ Have some pixels that may partially describe boundary of some objects.

Find set of pixels that make a straight lines.

Point $(x_i, y_i)$ → Line passing through: $y_i = ax_i + b$

↳ satisfy eqn of some set of params $(a, b)$.

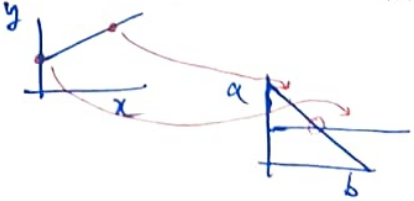
$\Rightarrow b = -ax_i + y_i$

$x, y \rightarrow$ parameters

$a, b \rightarrow$ variables.

↓

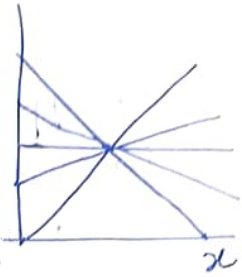
A line in $(a, b)$ space



2 points $(x_1, y_1)$, $(x_2, y_2)$ → define a line in $(x,y)$ plane.

↳ Gives 2 different lines in $(a,b)$ space.

↓ intersect at $(a', b')$.




⇒ All pts. on line defined by these two points in $(x,y)$ space will parameterize lines that intersect in $(a', b')$ in $(a,b)$ space.

Algorithm for H.T.:

① Quantize the parameter space $(a,b)$ by dividing it into cells.
⇒ Accumulator cells.

② Count the no. of times a line intersect a given cell.

③ Cells receiving more than a certain no of counts (called votes) are assumed to correspond to lines in $(x,y)$ space.

## Integral Image (Summed Area Table)

↳ If an image is to be repeatedly convolved with different box filters (and especially of different sizes at diff. locations), we can precompute the summed area table, which is just the running sum of all pixels values from the origin.

$$S(i,j) = \sum_{k=0}^{i} \sum_{l=0}^{j} f(k,l)$$

Recursive (raster-scan) algorithm:

$$s(i,j) = s(i-1,j) + s(i,j-1) - s(i-1,j-1) + f(i,j)$$

Eg.

| | | | |
|---|---|---|---|
| 3 | 2 | 7 | 2 |
| 1 | 5 | 1 | 3 |
| 5 | 1 | 3 | 5 |

I/p Image

| | | | |
|---|---|---|---|
| 3 | 5 | 12 | 14 |
| 4 | 11 | 19 | 24 |
| 9 | 17 | 28 | 38 |

Summed Area Table

$28 = 3 + 19 + 17 - 11$

Summed Area:

$$S(i_0 \ldots i_1, j_0 \ldots j_1) = s(i_1, j_1) - s(i_1, j_0 - 1) - s(i_0 - 1, j_1) + s(i_0 - 1, j_0 - 1)$$

$S = 5 + 1 + 1 + 3 = 28 + 3 - 12 - 9$

$= (10)$.

# Frequency Domain Analysis of Images

Fourier transform: $F(u) = \int_{-\infty}^{\infty} f(x) e^{-j2\pi ux} dx$

Inverse FT: $f(x) = \int_{-\infty}^{\infty} F(u) e^{j2\pi ux} du$

$g(x) = f(x) * h(x) \iff G(u) = F(u) H(u)$
(convolution) (Multiplication)

## 2D FT:

FT: $F(u,v) = \int\int_{-\infty}^{\infty} f(x,y) e^{-j2\pi(ux+vy)} dx dy$

IFT: $f(x,y) = \int\int_{-\infty}^{\infty} F(u,v) e^{j2\pi(xu+yv)} du dv$

Discrete FT: $F[p,q] = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f[m,n] e^{-j2\pi pm/M} e^{-j2\pi qn/N}$

$p, q$: frequencies along $m, n$ $\quad \begin{bmatrix} p=0,\dots,M-1 \\ q=0,\dots,N-1 \end{bmatrix}$

IDFT: $f[m,n] = \frac{1}{MN} \sum_{p=0}^{M-1} \sum_{q=0}^{N-1} F[p,q] e^{j2\pi pm/M} e^{j2\pi qn/N}$

$\begin{bmatrix} n=0,\dots,M-1 \\ n=0,\dots,N-1 \end{bmatrix}$

Image $f(m,n)$ $\xrightarrow{\mathcal{F}}$  $\log|F(p,q)|$

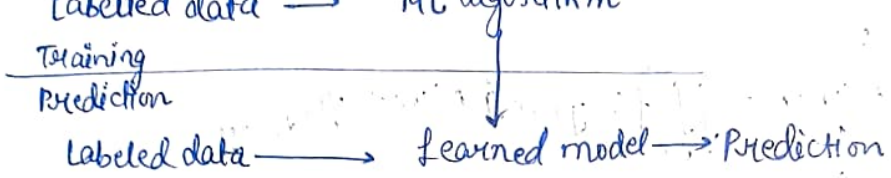


→ FT has peaks at spatial frequencies of repeated structure.

# Deep Learning

**Artificial Intelligence** → scientific field concerned with the development of algorithms that allow computers to learn without being explicitly programmed.

**Machine Learning** → Branch of AI which focuses on methods that learn data and make predictions on unseen data.




**ML Types**

① **Supervised**: Learning with labeled data.
② **Unsupervised**: Discover patterns in unlabeled data.
③ **Reinforcement learning**: Learn to act on feedback / reward.

→ supervised learning categories and techniques:
- ↳ Numerical classifier functions
  - ↳ Linear classifier, perceptron, logistic regression, SVM, NN.
- ↳ Parametric (probabilistic) functions
- ↳ Non-parametric (instance-based) functions → K-nearest neighbors.
- ↳ Symbolic functions.
- ↳ Aggregation (ensemble) learning
  - ↳ Bagging, boosting (Adaboost), random forest.

→ Unsupervised learning categories and techniques:
- ↳ Clustering
  - ↳ K-Means, Mean-shift, spectral.
- ↳ Density estimation.
- ↳ Dimensionality reduction.
  - ↳ Principal component analysis (PCA), factor analysis.

**Nearest Neighbor classifier**: For each data point, assign the class label of the nearest training data point.
- ↳ Distance function to find nearest neighbor.
- ↳ Does not require learning a set of weights.
- ↳ Remember all training data and store.

## K-Nearest Neighbors Classifier:

↳ Considers multiple neighboring data points ($k$) to classify a test data point.
↳ Class of test eg. is obtained by voting.

## Linear Classifier:

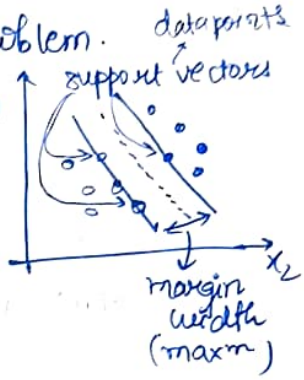
↳ Find a function $f$ of the inputs $x_i$ that separates the classes.

$$f(x_i, W, b) = Wx_i + b.$$

↳ Use pairs of inputs and labels to find weight matrix $W$ and bias vector $b$ (parameters of $f$).
↳ Several methods to find optimal set of parameters: eg., perceptron.
↳ Decision boundary is linear.

## Support vector Machines (SVM): Solves an optimization problem.

↳ Identify a decision boundary that correctly classifies the examples.
↳ Then, increase the geometric margin b/w the boundary and all examples.



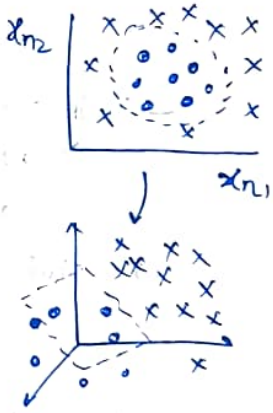

Find $W$ and $b$ by solving:

$$\min \frac{1}{2} \|W\|^2$$

such that $y_i(w.x_i + b) \geq 1 \; \forall x_i$.

## Non-linear Techniques:

Features $z_i$ are obtained as non-linear functions of the i/ps $x_i$.
↳ Non-linear decision boundaries.




## Non-linear SVM:

↳ The original i/p space is mapped to a higher-dimensional space where the training set is linearly separable.
↳ Define a non-linear kernel function to calculate a non-linear decision boundary in the original feature space.

## Multi-class classifications:

↳ 3 or more classes.


## Computer vision Tasks:

Classification, localization, object detection, instance segmentation.

## Non-Free-Lunch Theorem:

No single classifier works the best for all possible problems.
↳ Need to make assumptions to generalize.

**Deep learning**: Uses multiple layers for learning data representation.

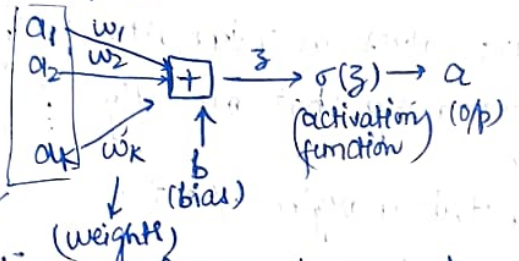
ML: I/p → Feature extraction → classification → output

DL: I/p → Feature extraction + classification → Output.

→ Neural networks with atleast one hidden layer are universal approximators.

↳ Use non-linear mapping of the inputs $x$ to the outputs $f(x)$ to compute complex decision boundaries.

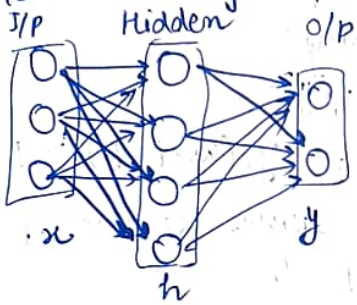
**Neuron**: $f: \mathbb{R}^K \to \mathbb{R}$




$z = a_1 w_1 + a_2 w_2 + \dots + a_K w_K + b$

$a = \sigma(z)$

→ NN with one hidden layer and one o/p layer:




Hidden layer: $h = \sigma(W_1 x + b_1)$

o/p layer: $y = \sigma(W_2 h + b_2)$

4+2=6 neurons (not counting I/ps)

$\begin{cases} (8\times4)+(4\times2) = 20 \text{ weights} \\ 4+2 = 6 \text{ biases} \end{cases}$

↳ 26 learnable parameters.

→ Fully connected (dense) layers

↳ Multilayer perceptron (MLP)

↳ Each neuron is connected to all neurons in succeeding layer.

→ Matrix operation:

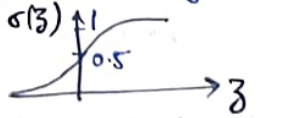
$$\sigma\left(\boxed{W}\boxed{x} + \boxed{b}\right) = \boxed{a}$$

**Softmax layer**: In multi-class classification

↳ Employs softmax activation function → Range [0,1].

↳ Sigmoid activations → for binary classification

↳ $\sigma(z) = \frac{1}{1+e^{-z}}$




↳ $y_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_i}}$, $z$ inputted to softmax layer : logits

[probability: $0 < y_i < 1$]

# Activation Functions:

↳ Non-linear activations are needed to learn complex (non-linear) data representations.

↳ Otherwise, NNs would be just a linear function ($W_1 W_2 x = Wx$).

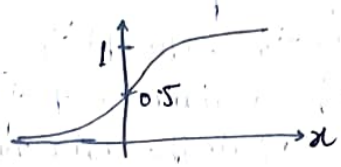
Eg. ① Sigmoid.

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

↓

$[0,1]$

↳ Less common




$\mathbb{R}^n \to [0,1]$

② Tanh.

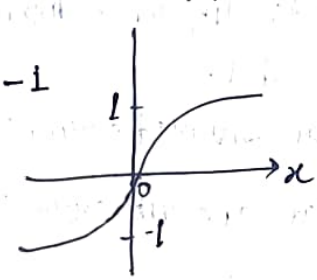
$$\tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

↳ Saturate

↳ Zero-centered

$\tanh(x) = 2 \cdot \sigma(2x) - 1$

↳ sigmoid




$\mathbb{R}^n \to [-1,1]$

③ ReLU (Rectified Linear Unit)

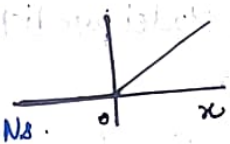
$f(x) = \max(0, x)$

↳ Used in most modern deep NNs.

↳ Fast to compute

↳ Accelerated the convergence of gradient descent

↳ Prevents gradient vanishing problem.



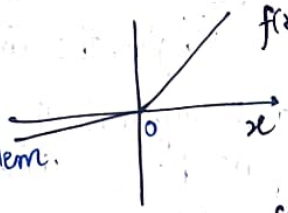

$$f(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$$

$\mathbb{R}^n \to \mathbb{R}_+^n$

④ Leaky ReLU

↳ Has a small -ve slope (e.g., $\alpha = 0.01$)

↳ Resolves dying ReLU problem.




$$f(x) = \begin{cases} ax, & x < 0 \\ x, & x \geq 0 \end{cases}$$

⑤ Linear function

↳ O/p $\propto$ I/p signal

↳ $c = 1$ : identity activation$^n$

↳ Used in regression problems.

$f(x) = cx$

$\mathbb{R}^n \to \mathbb{R}^n$

# Training NNs

Network parameters, $\theta = \{W^1, b^1, W^2, b^2, \ldots, W^L, b^L\}$ → weight matrices, ↳ bias vectors

→ Train the model to learn a set of parameters $\theta$ that are optimal.

① Data preprocessing → Helps convergence during training.
- ↳ Mean subtraction → to obtain zero-centered data.
- ↳ Normalization → divide each feature by its std. deviation.
  - ↳ scale the data within range $[0,1]$ or $[-1,1]$.

② Define a Loss/objective/cost function, $L(\theta)$
- ↳ calculates difference (error) b/w model prediction and true label.

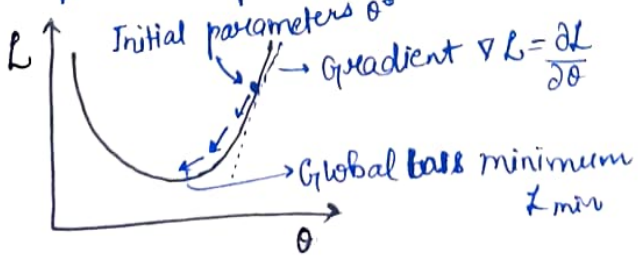
Eg (for regression tasks):
- (i) Mean squared error: $L(\theta) = \frac{1}{n}\sum_{i=1}^{n}[y^{(i)} - \hat{y}^{(i)}]^2$
- (ii) Mean absolute error: $L(\theta) = \frac{1}{n}\sum_{i=1}^{n}|y^{(i)} - \hat{y}^{(i)}|$
- (iii) Cross-entropy (for classification tasks): $L(\theta) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K}\left[y_k^{(i)} \log \hat{y}_k^{(i)} + (1-y_k^{(i)}) \log(1-\hat{y}_k^{(i)})\right]$

Ground-truth class labels → $y_i$
Model predicted class labels → $\hat{y}_i$

## Gradient Descent Algorithm

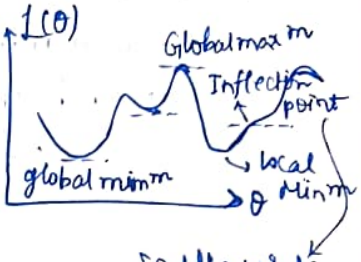
- ↳ To optimize loss function, $L(\theta)$.
- ↳ Applies iterative refinement of the n/w parameters $\theta$.
- ↳ Uses the opposite dirn of the gradient of the loss w.r.t. NN param (i.e. $\nabla L(\theta) = \partial L/\partial \theta_i$) for updating $\theta$.
  - ↳ gives the dirn of fastest increase of $L(\theta)$ when $\theta$ are updated.

Steps:
① Randomly initialize the model parameters, $\theta^0$
② compute the gradient of the loss function at initial parameters $\theta^0$: $\nabla L(\theta^0)$.
③ Update the parameters as: $\theta^{new} = \theta^0 - \alpha \nabla L(\theta^0)$ [$\alpha$: learning rate]
④ Go to step 2 and repeat until a termination criterion is reached.




$$\nabla L(\theta^0) = \begin{bmatrix} \partial L(\theta^0)/\partial w_1 \\ \partial L(\theta^0)/\partial w_2 \\ \vdots \end{bmatrix}$$

→ GD algo. stops when a local minimum of the loss surface is reached.
→ Doesn't guarantee reaching global minimum.
→ However, empirical evidence suggests that GD works well for NNs.
→ Random initialization in NNs results in different initial parameters $\theta^0$ every time the NN is trained.
  ↳ GD may reach different minima at every run.
    ↳ Produce different predicted o/ps.




**Forward propagation**: (forward pass) Passing the i/ps $x$ through the hidden layers to obtain model o/ps (predictions) $y$ → calculate $L$.

**Backpropagation**: (Backward propagation)
↳ Traverses the n/w in reverse order, from o/p $y$ backward towards the i/p $x$ to calculate the gradients of the loss $\nabla L(\theta)$.
↳ Chain rule is used to calculate the partial derivatives of the loss fn w.r.t. $\theta$ in different layers in the n/w.
↳ Each update of model parameter $\theta$ during training takes one forward and one backward pass (e.g., of a batch of i/ps).
↳ Automatic calculation of gradients (automatic differentiation) is available in all current deep learning libraries.

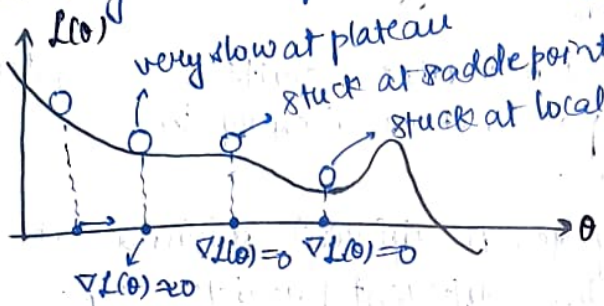
**Mini-Batch G.D.**:
↳ Compute loss $L(\theta)$ on a mini-batch of images, update the parameters $\theta$ and repeat until all images are used.
↳ Much faster training.
↳ Works b/c the gradient from a mini-batch is a good approximation of the gradient from the entire training set.

**Stochastic Gradient Descent**:
↳ Uses mini-batches that consists of a single input example (i.e. 1 image).
↳ Fast, but causes significant fluctuations in the loss function.

## Problems with G.D.

↳ Local minima

↳ can be very slow at plateaus and can get stuck at saddle points.




**G.D with momentum**: Use momentum of gradient for θ optimization

Movement = −ve of gradient + Momentum

weighted average of past gradients

$\theta^t = \theta^{t-1} - v^t$, $v^t$: momentum → accumulated grad. from past several steps.

where $v^t = \beta v^{t-1} + \alpha \nabla L(\theta^{t-1})$

$\therefore \theta^t = \theta^{t-1} - \alpha \nabla L(\theta^{t-1}) - \beta v^{t-1}$

prev. iteration

β: coeff. of momentum
↳ typically = 0.9

**Nesterov Accelerated Momentum**:

$\theta^t = \theta^{t-1} - v^t$

where $v^t = \beta v^{t-1} + \alpha \nabla L(\theta^{t-1} + \beta v^{t-1})$

↳ allows to predict position of param. in the next step.

**Adam (Adaptive Moment Estimation)**

weighted avg. of past and weighted avg. of past squared gradients.

$\theta^t = \theta^{t-1} - \alpha \frac{\hat{v}^t}{\sqrt{\hat{v}^t} + \epsilon}$, where $\hat{v}^t = \frac{v^t}{1-\beta_1}$, $\hat{v}^t = \frac{v^t}{1-\beta_2}$

$v^t = \beta_1 v^{t-1} + (1-\beta_1) \nabla L(\theta^{t-1})$

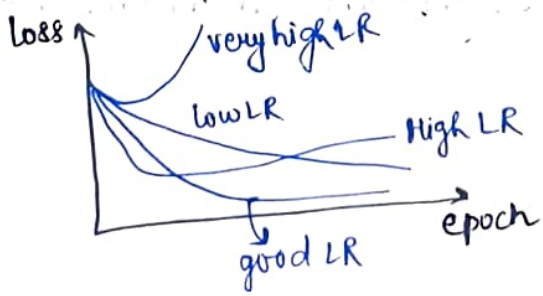
$v^t = \beta_2 v^{t-1} + (1-\beta_2) (\nabla L(\theta^{t-1}))^2$

[Typically, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$]

**Learning Rate**: Step size

High LR ⇒ Loss ↑ or plateaus too quickly

Low LR ⇒ Loss ↓ too slowly (takes many epochs)



**Learning Rate scheduling**: Changes LR during training

↳ **Annealing**: Reduces LR over time (LR ~~decay rate~~ decay).

↳ Approaches: ① ↓ LR by some factor every few epochs.

② Exponential or cosine decay over time.

③ ↓ LR by a constant (eg. by half) whenever validation loss stops improving.

↳ **Warmup**: Gradually ↑ LR initially, afterward let it cool down until the end of training.

**Vanishing Gradient Problem**

↳ During training, gradients can become either very small (vanishing gradient) or very large (exploding gradient).

↳ very small or very large update of parameters.

↳ Solutions: change LR, ReLU activations, regularization, LSTM units in RNNs.

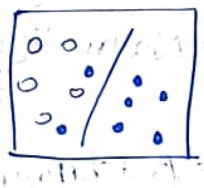
**Generalization**:

① **Underfitting**

↳ Model too simple to represent all relevant class characteristics.

↳ Model with too few parameters.

↳ Produces high error on training set and high error on validation set.




② **Overfitting**:

↳ Model too complex and fits irrelevant characteristics (noise) in the data

↳ Model with too many params.

↳ Produces low error on training error and high error on the validation set


**Regularization: Weight Decay**

L2 weight decay: Penalizing large weights added to the loss fn

$$L_{reg}(\theta) = \underbrace{L(\theta)}_{\text{data loss}} + \lambda \underbrace{\sum_k \theta_k^2}_{\text{regular. loss}}$$

$\lambda$: weight decay coeff.
↳ determines how dominant regular. is during gradient computation.

$l_1$ weight decay: Less common → perform worse than $l_2$ decay.

$$L_{reg}(\theta) = L(\theta) + \lambda \sum_k |\theta_k|$$

→ Combine $l_1$ and $l_2$ regularization (Elastic net regularization).

$$L_{reg}(\theta) = L(\theta) + \lambda_1 \sum_k |\theta_k| + \lambda_2 \sum_k \theta_k^2$$

Regularization: Dropout

↳ Randomly drop units (along with their connections) during training. [Often 20% – 50% units dropped.]

↳ A kind of ensemble learning.

Regularization: Early Stopping

↳ During model training, use of validation set.

↳ Stop when validation accuracy (or loss) has not improved after n epochs. [n → patience ↳ parameter]

Batch Normalization → similar to data preprocessing steps.

↳ Normalize data $x$ to zero mean and unit variance.

$$\hat{x} = \frac{x - \mu}{\sigma}$$

↳ BatchNorm layers alleviate the problems of proper initialization of parameters and hyper-parameters.

↳ Faster convergence training, allow ↑ LR.

↳ Inserted immediately after convolutional layers or fully-connected layers, and before activation layers.

↳ Very common with CNNs.

Hyper-Parameter Tuning: Setting hyper parameters.

↳ No. of layers, no. of neurons per layer

↳ Initial LR, LR decay schedule (eg., decay constant).

↳ Optimizer type

↳ Regularization parameters

↳ Batch size

↳ Activation functions

↳ Loss function.

↳ Grid search: Check all values in a range with step value.

↳ Random search: Randomly sample values for parameters.

↳ Bayesian hyper-parameter optimization.

**K-Fold Cross-Validation** → Hyper-parameter tuning when size of training data is small.

↳ Better and less noisy estimate of model performance by averaging the results across several folds.

Eg. 5-Fold CV.

↳ ① Split train data into 5 equal folds.

② First use 2-5 for training and fold 1 for validation.

③ Repeat by using fold 2 for validation, then fold 3, 4, 5.

④ Average the results over 5 runs.

⑤ Determine best hyper-param and evaluate model on test data.

**Ensemble Learning**

↳ Training multiple classifiers separately and combining their predictions. → outperforms individual classifiers.

↳ **Bagging** (bootstrap aggregating):

↳ Randomly draw subsets from training set (bootstrap samples).

↳ Train separate classifiers on each subset of training set.

↳ Perform classification based on average vote of all classifiers.

↳ **Boosting**:

↳ Train a classifier and apply weights on the training set (apply higher weights on misclassified examples).

↳ Train new classifier, reweight training set a/c to prediction error.

↳ Repeat.

↳ Perform classification based on weighted vote of classifiers.

**Deep vs Shallow Networks**

↳ Deeper n/ws perform better than shallow n/ws, but only upto some limit (after that performance plateaus).

# Convolutional Neural Networks (CNNs)

- Primarily designed for image data.
- Use a convolutional operator for extracting data features.
  - Allow parameter sharing.
  - Efficient to train.
  - Have less parameters than fully-connected NNs.
- Robust to spatial translations of objections in images.
- A convolutional filter slides (convolves) across the image.
  - capture useful features.
- Hidden units in a layer are only connected to a small region of the layer before it (called **receptive field**).
- **Feature map**: Depth of each feature map corresponds to the no. of convolutional filters used at each layer.
- **Max pooling**: Reports "Max" o/p within a rectangular region.
- **Average pooling**: Reports the avg. o/p of a rectangular neighbor-hood.
- Pooling layers reduce the spatial size of feature maps.
  - Reduce no. of params., prevent overfitting.

# RANSAC and Feature Detectors

## RANSAC (RANdom SAmple Consensus)

↳ Robust model fitting algorithm that can estimate a model from data containing many outliers.

↳ For edge detection, line fitting, homography estimation, 3D reconstruction.

### RANSAC loop:

① Randomly select a seed group of points on which to base transformation estimate.

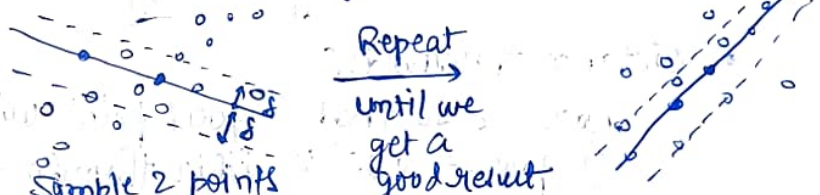
② Compute transformation / Fit a model using only sampled pts.

③ Find inliers.

④ If the no. of inliers is sufficiently large, re-compute least squares estimate of transformations on all of the inliers.

→ Keep the model / transformations with the largest no. of inliers.

Eg Estimate best line (Line Fitting):




Simple 2 points and fit a line to them.

↓

Calculate total no. of points within a threshold.

### Feature-Based Alignment:

① Extract features.

② Compute putative matches.

③ Loop:
- Hypothesize transformation T.
- Verify transformation (search for other matches consistent with T).

## Choosing parameters:

- No. of sample points : $s$ (2 for fitting a line)
- Outlier ratio : $e = \frac{\text{no. of outliers}}{\text{no. of datapoints}}$
- No. of trials : $T$

Choose $T$ so that, with probability $p$, atleast one random sample is set free from outliers.

$p$(fail once) = donot select any inliers

$\Rightarrow 1-p = 1-(1-e)^s$

$(1-e)^s$ : In one trial, prob. that all $s$ sampled pts. are inliers.

$1-(1-e)^s$ : prob. that a sample contains atleast one outlier (ie, is bad)

$p$(fail $T$ times) = select atleast one outlier in all $T$ trials)

$1-p = [1-(1-e)^s]^T$

$\Rightarrow \log(1-p) = T \log(1-(1-e)^s)$

$$\Rightarrow \boxed{T = \frac{\log(1-p)}{\log(1-(1-e)^s)}}$$

- Distance threshold: $\delta$

  Choose $\delta$ so that a good point with noise is likely (eg, prob. = 0.95) is within threshold.

Prob. that all $k$ samples fail : $(1-w^s)^k$ | $k$: samples chosen

$\Rightarrow$ Choose $k$ high enough to keep this below desired failure rate.

## Local Invariant Features

↳ Global representations have major limitations $\Rightarrow$ Match local regions.

↳ Increased robustness to occlusions, articulation, intra-category variations.

Steps: ① Find a set of distinctive key-points

② Define a region around each keypoint.

③ Extract and normalize the region content.

④ Compute a local descriptor from normalized region.

⑤ Match local descriptors.

↳ Region extraction needs to be repeatable and accurate.

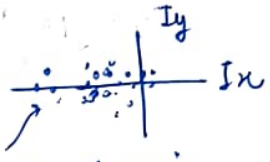
↳ Geometric & photometric invariant.

# Harris Corner Detection

↳ Identifies corner in an image by measuring how the intensity changes in all directions around a pixel.

$\sum I_x^2 \rightarrow$ large $\sum I_x^2 \rightarrow$ small $\sum I_x^2 \rightarrow$ small
$\sum I_y^2 \rightarrow$ large $\sum I_y^2 \rightarrow$ large $\sum I_y^2 \rightarrow$ small
(Corner) (Edge) (Nothing) → flat

↓
Small movement in any dirn causes large change in intesity




**Finding corners:** ① compute image gradients over small region.
$I_x = \frac{\partial I}{\partial x}$, $I_y = \frac{\partial I}{\partial y}$.
② subtract mean from each image gradient: → centered data
③ Compute covariance matrix.
④ compute eigenvectors and eigenvalues.
⑤ Use threshold on eigenvalues to detect corners.

Covariance matrix:

$$\begin{bmatrix} \sum_{p \in P} I_x I_y & \sum_{p \in P} I_x I_y \\ \sum_{p \in P} I_y I_x & \sum_{p \in P} I_y I_y \end{bmatrix}$$

**Intensity change fn:** Change in intensity $E(u,v)$ when window is shifted by $(u,v)$ is

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

window → Gaussian or box

First-order Taylor expansion:

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

$$\therefore E(u,v) \approx \sum_{x,y} w(x,y) \left[ u I_x + v I_y \right]^2$$

$$= [u, v] \sum w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

$M \rightarrow$ autocorrelation matrix
↳ 2×2 Harris matrix

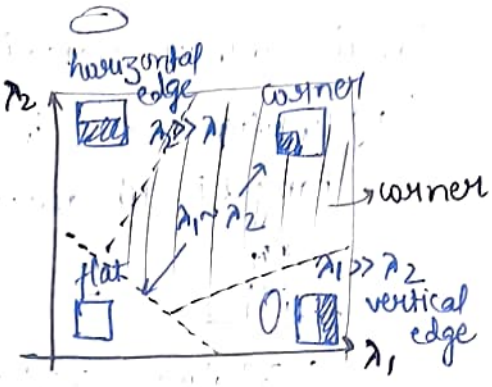
\# Compute eigenvalues ($\lambda$) & eigenvectors ($e$)
$Me = \lambda e$.
$(M - \lambda I)e = 0$

↳ ① $\det(M - \lambda I) = 0 \Rightarrow \lambda_1, \lambda_2$
② For $\lambda_1, \lambda_2$, solve $(M - \lambda I)e = 0$

## Corner Response function

$M \rightarrow$ eigenvalues: $\lambda_1, \lambda_2$

$$R = \det(M) - k(\text{trace}(M))^2$$
$$= (\lambda_1 \lambda_2) - k(\lambda_1 + \lambda_2)^2$$

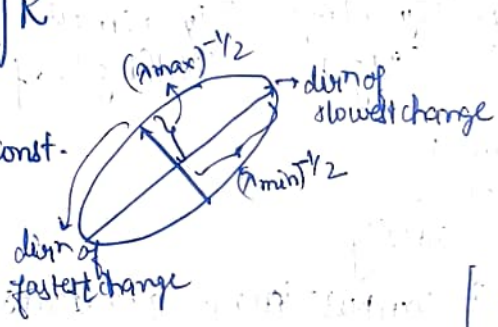


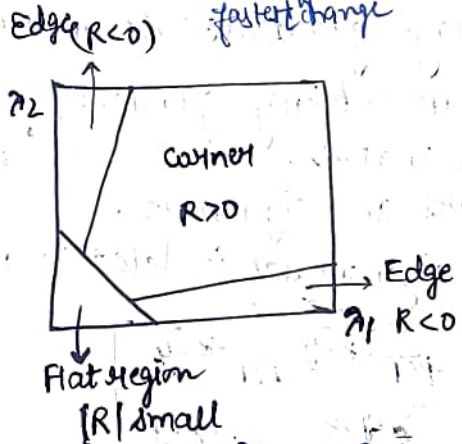

$M \rightarrow$ symmetric

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

Ellipse fn:

$$[u\ v] M \begin{bmatrix} u \\ v \end{bmatrix} = \text{const.}$$



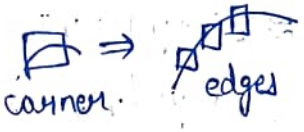




# Kanade & Tomasi:
$R = \min(\lambda_1, \lambda_2)$

# Nobel:
$$R = \frac{\det(M)}{\text{trace}(M) + \epsilon}$$

## Harris Detector: Properties

↳ Translation invariance
↳ Rotation invariance.
↳ Partial invariance to affine intensity change.
↳ Not scale invariant.



# Histograms of Oriented Gradients (HOG) and Bag of Visual Words (BoVW) features

## SIFT (Scale-Invariant Feature Transform) → Advantages:

- Locality: features are local, robust to occlusion and clutter.
- Distinctiveness: individual features can be matched to a large database of objects.
- Quantity: many features can be generated for even small objects.
- Efficiency: close to real-time performance.
- Extensibility: To different feature types.

## SIFT Algorithm:

① scale-space peak selection: potential location of finding features.
② Keypoint localization: Accurately locating feature keypoints.
③ Orientation assignment: Assigning orientation to keypoints.
④ Keypoint descriptor: Describing keypoints as a high-dimensional vector.
⑤ Keypoint matching.

## HoG:

① Compute a HoG.
② (Optional) global image localization.
③ Computing gradient image in x and y.
④ Computing gradient histograms.
⑤ Normalizing across blocks.
⑥ Flattening into a feature vector.

↳ Focuses on structure or shape of an object.
↳ Orientations are calculated in localized ~~patterns~~ portions.

Step: ① Preprocess the data
width to height ratio = 1:2 (preferably 64×128).
↳ For dividing images into 8×8 and 16×16 patches.
② Calculate gradients (dir^n x and y).
③ Calculate magnitude and orientation.
④ Calculate histogram of gradient in 8×8 cells.
⑤ Normalize gradients in 16×16 cell (36×1) ⇝ Reduce lighting variation
⑥ Features for complete image.
↳ Combine 16×16 blocks to get features for the final image.

# Bag of Words

→ Visual words = independent features
→ words from the dictionary.
→ Represent images based on histogram of word occurences.
→ Image comparisons are performed based on such word histograms.

① Feature detection and representation

sliding window
regular grid
interest point detector

→ visual words (keypoints)
K-Means clustered pieces of image

- Image to histogram
- Dictionary → list of words → learned from data
- Extract feature descriptors from a training dataset.
  ↳ points in high-dimensional space.
  ↳ cluster similar descriptors (K-Means)

Task: Find similar looking images
Input: Database of images, dictionary, query image(s).
Output: N most similar database images to query images.
→ words that occur in every image do not help a lot for comparisons.

IF-IDF Reweighting:

TF-IDF = term frequency (of word in a document) − inverse document frequency (of word across several documents)

$$t_{id} = \underbrace{\frac{n_{id}}{n_d}}_{\text{term-frequency}} \underbrace{\log \frac{N}{n_i}}_{\text{IDF}}$$

$t_{id}$: histogram bin of word i for image d
$n_{id}$: ~~term-freq~~: occurences of word i in image d
$n_d$: no. of word occurences in d.
$n_i$: no of images that contain word i
$N$: no of images

→ Relevant words get higher weights, others are weighted down to 0 (those occuring in every image).

Comparing two Histograms:

cosine similarity: $\text{cossim}(x,y) = \cos\theta = \frac{x^T y}{\|x\| \|y\|}$ [0,1]

cosine distance: $d_{\cos}(x,y) = 1 - \text{cossim}(x,y) = 1 - \frac{x^T y}{\|x\| \|y\|}$.
↳ form cost matrix.

Euclidean distance: $\|x - y\|^2 = 2 d_{\cos}(x,y)$ [for $\|x\| = \|y\| = 1$]

# Video Processing

**Optical Flow:** Given two consecutive image frames, estimate the motion of each pixel.
↳ Assumptions: Brightness constancy, small motion.

**Color/Brightness Constancy:** Allows for pixel-to-pixel comparison (not image features)

$$I(x(t), y(t), t) = C$$

$$\boxed{\frac{dI}{dt} = \frac{\partial I}{\partial x}\frac{dx}{dt} + \frac{\partial I}{\partial y}\frac{dy}{dt} + \frac{\partial I}{\partial t} = 0}$$ ... Brightness constancy eqn.

**Derivation:**

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

Assuming small motion, taylor series expansion:

$$I(x,y,t) = I(x,y,t) + \frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial t}\delta t$$
(fixed pt.)

Divide by $\delta t$. Take limit $\delta t \to 0$:

$$\Rightarrow \frac{\partial I}{\partial x}\frac{dx}{dt} + \frac{\partial I}{\partial y}\frac{dy}{dt} + \frac{\partial I}{\partial t} = 0$$

$$\text{or, } I_x u + I_y v + I_t = 0$$

$$\text{or, } \nabla I^T \mathbf{v} + I_t = 0$$

$(1\times2)(2\times1)$

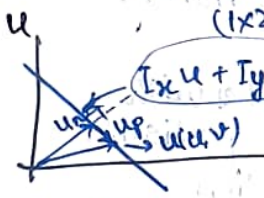
$I_x u + I_y v + I_t = 0$ → 2 unknowns → 2 equations needed

$I_x, I_y$: image gradients (spatial der.) — compute by sobel filter, DOG filter, forward difference

$u, v$: flow velocities

$I_t$: temporal gradient — compute by frame differencing.

optical flow → solve for this (unknown) → soln lies on a line. cannot be found uniquely with a single constraint.




$u = u_n + u_p$
normal flow, parallel flow

$$\hat{u}_n = \frac{(I_x, I_y)}{\sqrt{I_x^2 + I_y^2}}, \quad |u_n| = \frac{|I_t|}{\sqrt{I_x + I_y^2}} \Rightarrow u_n = \frac{|I_t|}{(I_x^2 + I_y^2)}(I_x, I_y)$$

$u_p$ → cannot be determined

**Small Motion:** Linearization of brightness constancy constraint

**Horn-Schrunck Optical Flow**
- → Brightness constancy
- → small motion
- → Smooth flow (flow can vary from pixel to pixel)
- → Global method (dense)

**Lucas-Kanade Optical Flow**
- → Method of differences
- → Constant flow (flow is constant for all pixels)
- → Local method (sparse)

## Constant Flow: Lucas-Kanade Optical Flow

$I_x u + I_y v + I_t = 0$

→ Assume that the surrounding path (say 5×5) has constant flow. Gives 25 equations.

→ Assumptions: Flow is locally smooth
Neighbouring pixels have same displacement.

$$\left.\begin{aligned} I_x(p_1)u + I_y(p_1)v &= -I_t(p_1) \\ &\vdots \\ I_x(p_{25})u + I_y(p_{25})v &= -I_t(p_{25}) \end{aligned}\right\} \Rightarrow \begin{bmatrix} \sum_{p\in P} I_x I_x & \sum_{p\in P} I_x I_y \\ \sum_{p\in P} I_y I_x & \sum_{p\in P} I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = -\begin{bmatrix} \sum_{p\in P} I_x I_t \\ \sum_{p\in P} I_y I_t \end{bmatrix}$$

$A^TA$ $\quad \hat{x}$ $\quad A^Tb$

$\Rightarrow A^TA\,\hat{x} = A^Tb$

$\Rightarrow x = (A^TA)^{-1}A^Tb$

[Summation over each pixel p in patch P.]

↳ solvable when:

$A^TA$ → invertible

$A^TA$ → not too small $\Rightarrow \lambda_1, \lambda_2$ → not too small.

Harris Corner Detector

↳ corners when $\lambda_1, \lambda_2$ are big ⇒ Also when L-K optical flow works best.

→ Barber's pole illusion.

→ Aperture Problem

↳ Small visible image patch

↳ In which diren is line moving?

⇓ Avoid by having patches with different gradients.

## Smoothness: Horn-Schunck Optical Flow.

Enforce Brightness constancy, $I_x u + I_y v + I_t = 0$

⇒ For every pixel, ($I_x(i,j)$)

$$\min_{u,v} [I_x u_{ij} + I_y v_{ij} + I_t]^2$$

Enforce smooth flow field,

$$\min_u (u_{ij} - u_{i+1,j})^2$$ → large for noisy patch

Combining,

$$\min_{u,v} \sum_{i,j} \{E_s(i,j) + \lambda E_d(i,j)\}$$ ← weight

$E_s$: smoothness, $E_d$: brightness constancy

solve → compute partial derivatives, derive update eq's. (Gradient descent).

$$E_s = \frac{1}{4}\left[(u_{ij} - u_{i+1,j})^2 + (u_{i,j} - u_{i,j+1})^2 + (v_{ij} - v_{i+1,j})^2 + (v_{ij} - v_{i,j+1})^2\right]$$

$$E_d = \left[I_x u_{ij} + I_y v_{ij} + I_t\right]^2$$

$$\frac{\partial E}{\partial u_{kl}} = 2(u_{kl} - \bar{u}_{kl}) + 2\lambda(I_x u_{kl} + I_y v_{kl} + I_t) I_x$$

$$\frac{\partial E}{\partial v_{kl}} = 2(v_{kl} - \bar{v}_{kl}) + 2\lambda[I_x v_{kl} + I_y v_{kl} + I_t) I_y$$

u terms depend on k, l → 4 from smoothness, 1 from brightness constancy

Find extrema of E. $\left[\bar{u}_{ij} = \frac{1}{4}\{u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j+1}\}\right]$

$$\Rightarrow (1 + \lambda I_x^2) u_{kl} + \lambda I_x I_y v_{kl} = \bar{u}_{kl} - \lambda I_x I_t$$

$$\lambda I_x I_y u_{kl} + (1 + \lambda) I_y^2) v_{kl} = \bar{v}_{kl} - \lambda I_y I_t$$

or, $Ax = b$

$$\Rightarrow x = A^{-1} b = \frac{\text{adj } A}{\det A} b$$

$$\Rightarrow \{1 + \lambda(I_x^2 + I_y^2)\} u_{kl} = (1 + \lambda I_x^2) \bar{u}_{kl} - \lambda I_x I_y \bar{v}_{kl} - \lambda I_x I_t,$$

$$\{1 + \lambda(I_x^2 + I_y^2)\} v_{kl} = (1 + \lambda I_y^2) \bar{v}_{kl} - \lambda I_x I_y \bar{u}_{kl} - \lambda I_y I_t.$$

(det A)

$$\Rightarrow \boxed{\begin{aligned} \hat{u}_{kl} &= \bar{u}_{kl} - \frac{I_x \bar{u}_{kl} + I_y \bar{v}_{kl} + I_t}{\lambda^{-1} + I_x^2 + I_y^2} I_x, \\ \hat{v}_{kl} &= \bar{v}_{kl} - \frac{I_x \bar{u}_{kl} + I_y \bar{v}_{kl} + I_t}{\lambda^{-1} + I_x^2 + I_y^2} I_y \end{aligned}}$$

(new value) (old average)

When $\lambda$ is small $\Rightarrow \lambda^{-1}$ is big.

$\Rightarrow \hat{u}_{kl} = \bar{u}_{kl}$

$\hat{v}_{kl} = \bar{v}_{kl}$ { only smoothness }

## Horn-Schunck Optical Flow Algorithm:

1. Precompute image gradients, $I_y$, $I_x$.
2. Precompute temporal gradients, $I_t$.
3. Initialize flow field, $u=0$, $v=0$
4. while not converged, compute flow field update for each pixel using $\hat{u}_{kl}$, $\hat{v}_{kl}$ formula above.

## Pyramidal Optimal Flow Estimation

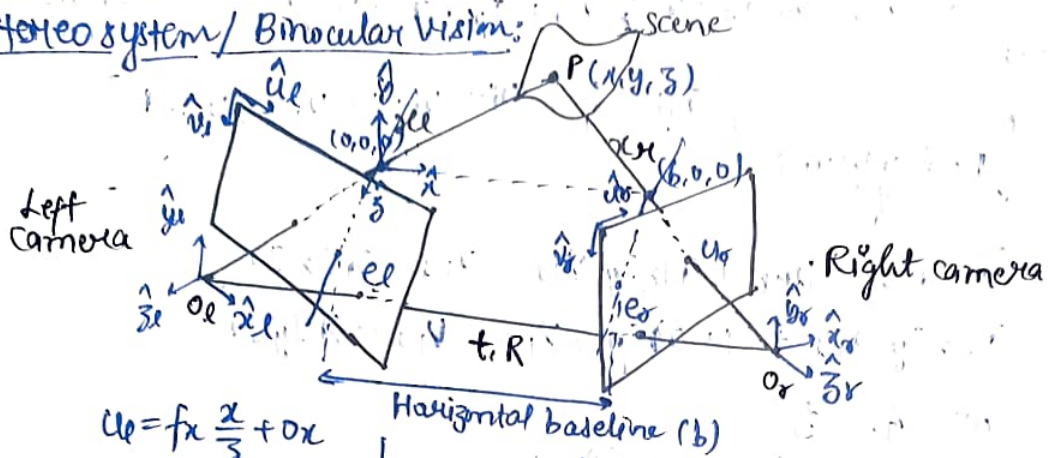
Pyramids: 2 types: ① Low pass or Gaussian → Convolve image with a small gaussian kernel. → subsample to get lower resolution image. → Repeat for more levels → A sequence of low-pass filtered images.

② Band pass or Laplacian → highlight edges at different spatial scales.

↳ less sensitive to illumination changes, but more noisy than using Gaussians.

# Stereo Vision

## Stereo system / Binocular Vision:




$u_l = f_x \frac{x}{z} + o_x$

$v_l = f_y \frac{y}{z} + o_y$

$u_r = f_x \frac{x-b}{z} + o_x$

$v_r = f_y \frac{y}{z} + o_y$ ; $f_x, f_y, b, o_x, o_y$ are known.

Solving for $(x, y, z)$,

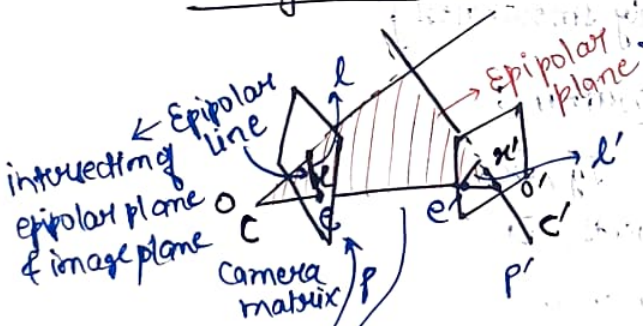
$$x = \frac{b(u_l - o_x)}{u_l - u_r}, \quad y = \frac{b f_x (v_l - o_y)}{f_y (u_l - u_r)}, \quad \boxed{z = \frac{b f_x}{u_l - u_r}}$$

Disparity : $(u_l - u_r) \propto$ Baseline $(b)$

Depth $z \propto \frac{1}{\text{disparity}}$

Disparity = 0 → points infinitely far away from the cameras

## Triangulation:




→ Given set of matched points $\{x_i, x_i'\}$, camera matrices $P, P'$

↓

Estimate the 3D point $X$.

$x = PX$ (homogeneous coordinates)

$x = \alpha P X$, → differ by a scalar factor

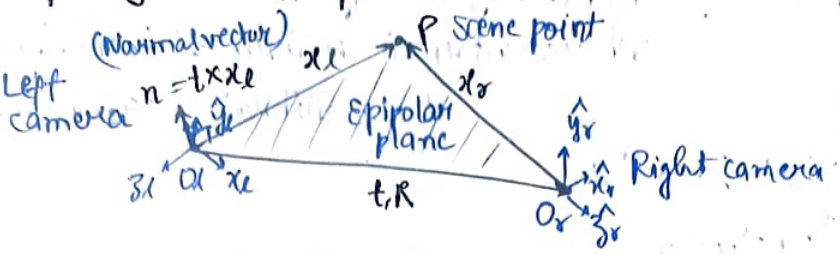
$x \times PX = 0$ (cross product = 0)

$\Downarrow$

$AX = 0$

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \alpha \begin{bmatrix} P_1 & P_2 & P_3 & P_4 \\ P_5 & P_6 & P_7 & P_8 \\ P_9 & P_{10} & P_{11} & P_{12} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

# Computing extrinsic parameters of two cameras:




$$x_l \cdot (t \times x_l) = 0$$

$$\Rightarrow \begin{bmatrix} x_l & y_l & z_l \end{bmatrix} \begin{bmatrix} t_y z_l - t_z y_l \\ t_z x_l - t_x z_l \\ t_x y_l - t_y z_l \end{bmatrix} = 0$$

$$\Rightarrow \begin{bmatrix} x_l & y_l & z_l \end{bmatrix} \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix} \begin{bmatrix} x_l \\ y_l \\ z_l \end{bmatrix} = 0$$

Substitute $x_l = R x_r + t$. → skew-symmetric

$t_{3\times1}$ : position of R.C in LC's frame

$R_{3\times3}$ : orientation of L.C. in RC.'s frame

$$\Rightarrow \begin{bmatrix} x_l & y_l & z_l \end{bmatrix} \left( \underbrace{\begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}}_{E = \begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{bmatrix}} \begin{bmatrix} x_r \\ y_r \\ z_r \end{bmatrix} + \underbrace{\begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix} \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}}_{t_x t = 0} \right) = 0$$

$$\Rightarrow \begin{bmatrix} x_l & y_l & z_l \end{bmatrix} E \begin{bmatrix} x_r \\ y_r \\ z_r \end{bmatrix} = 0 \Rightarrow \text{, where } \boxed{E = T_x R} \rightarrow \text{Essential matrix}$$

$$\Rightarrow \boxed{x_l^T E x_r = 0} \rightarrow \text{Relates 3D position of scene w.r.t. LC to its 3D position w.r.t. RC.}$$

→ Potential matches to $x$ lie on the epipolar line $l'$.

→ Epipolar constraint reduces search to a single line.

$$\boxed{Ex = l'}$$

$$\boxed{x'^T E x = 0} \quad \left[\because x'^T l' = 0,\ Ex = l'\right] \rightarrow \text{Maps a point to a line (different from homography)}$$

$$\boxed{x' = R(x - t)} \quad \text{(Rigid motion)}$$

$x, t, x'$ are coplanar $\Rightarrow x^T (t \times x) = 0$

$\Rightarrow (x - t)^T (t \times x) = 0$

$\Rightarrow (x'^T R)(t \times x) = 0$

$\Downarrow$

$$\boxed{\begin{aligned} x'^T E x &= 0 \\ E &= (R[t_x]) \end{aligned}}$$

## Properties of Essential Matrix:

Longuet-Higgins equation: $x'^T E x = 0$

Epipolar lines: $x^T l = 0,\ x'^T l' = 0$

$l' = Ex,\ l = E^T x'$

Epipoles: $e'^T E = 0,\ Ee = 0$

Assumption: 2D points expressed in camera c.s. (i.e., intrinsic matrices are identities).

**Fundamental Matrix**: Generalization of essential matrix, where intrinsic matrix ≠ identity. ↳ uncalibrated cameras.

$\hat{x}'^T E \hat{x} = 0$

$\hat{x}' = K'^{-1} x'$; $\hat{x} = K^{-1} x$.

$\Rightarrow K'^{-T} E K^{-1} x = 0$

$\Rightarrow x'^T (K'^T E K^{-1}) x = 0$

$\Rightarrow \boxed{x'^T F x = 0}$ → Properties of essential matrix applicable.

$F = K'^T E K^{-1}$

$= K'^{-T} [t_\times] R K^{-1}$ → dependent on both intrinsic & extrinsic parameters.

Solve for F: $x_m'^T F x_m = 0$ → for M matches image points.

$\Rightarrow [x_m' \; y_m' \; 1] \begin{bmatrix} f_1 & f_2 & f_3 \\ f_4 & f_5 & f_6 \\ f_7 & f_8 & f_9 \end{bmatrix} \begin{bmatrix} x_m \\ y_m \\ 1 \end{bmatrix} = 0$ [F → 3×3]

$\Rightarrow x_m x'_m f_1 + x_m y'_m f_2 + x_m f_3 + y_m x'_m f_4 + y_m y'_m f_5 + y_m f_6 + x'_m f_7 + y'_m f_8 + f_9 = 0$ ⇒ One correspondence gives one equation.

Setup a homogeneous linear system with 9 unknowns.

Each point pair contributes only one scalar equation.

Need atleast 8 points. ⇒ 8 Point Algorithm.

⇒ $Ax = 0$ → solve using SVD.

**8-point algorithm:**

⓪ (Normalize points)

① Construct the M×9 matrix A.

② Find the SVD of A.

③ Entries of F are the elements of column of V corresponding to the least singular value.

④ (Enforce rank 2 constraint on F) → SVD

⑤ (Un-normalize F)

**Enforcing Rank constraint**: Given matrix F. Find F' of rank K that is closest to F.

$\min_{F'} \|F - F'\|^2$ ⇒ Compute SVD of F ⇒ $F = U \Sigma V^T$

⇒ Form $\Sigma'$ by replacing all but largest singular value in $\Sigma$ with 0.

⇒ ∴ $F' = U \Sigma' V^T$.

# Image segmentation

↳ Partition an image into a collection of set of pixels (segments).
  ↳ Meaningful regions, shapes. ....
↳ Techniques: Thresholding; region based methods (region growing), clustering (K-means), graph-based methods (graph-cut, random walk), shape-based methods (level set, active contours), energy minimization methods (MRF), Machine Learning based methods.

**Image Binarization** → One global threshold T for mapping scalar image I to binary image.

$$J(x,y) = \begin{cases} 0, & \text{if } I(x,y) < T \\ 1, & \text{otherwise.} \end{cases}$$

↳ Global threshold can be identified by an optimization strategy aiming at creating connected regions and at reducing the no. of small-sized regions, called artifacts.

**Thresholding:**

**Histogram:**


**Difficulties:**
→ Valley may be broad.
→ No of minima.
→ Noise; no visible valley.
→ Multi-modal histogram.

**Otsu Thresholding:** Uses gray-value histogram to provide best threshold such that overlap b/w 2 classes (object and background pixels) is minimized.

$$\sigma_b^2 = P_1(\mu_1 - \mu)^2 + P_2(\mu_2 - \mu)^2 = P_1 P_2 (\mu_1 - \mu_2)^2$$

$\sigma_b^2$: b/w-class variance
$P_1, P_2$: class probabilities
$\mu_i$: means of object and bg classes.

**Algorithm:** Compute histogram $H_I$ for $u = 0, \ldots, G_{max}$
Let $T_0$: increment for potential threshs., $u = T_0$, $T = u$, $s_{max} = 0$.
while $u < G_{max}$ do
  compute $c_I(u)$, $\mu_i(u)$ for $i = 1, 2$.
  compute $\sigma_b^2 = c_I(u)[1 - c_I(u)][\mu_1(u) - \mu_2(u)]^2$
  if $\sigma_b^2(u) > s_{max}$ then
    $s_{max} = \sigma_b^2$, $T = u$
  set $u = u + T_0$.

# $P_1 = \sum_{i=0}^{u} P(i)$, $P_2 = \sum_{i=u+1}^{G_{max}} P_i$, $\mu_1 = \sum_{i=0}^{u} i p(i)/P_1$, $\mu_2 = \sum_{i=u+1}^{G_{max}} i p(i)/P_2$.

$C_I$ : Relative cumulative histogram of image I.

$P_1 \approx C_I(u)$, $P_2 \approx 1 - C_I(u)$.

u: Chosen threshold.

↳ Performs well when histogram has a bimodal distribution with a deep and sharp valley b/w 2 peaks.

↳ Performing badly in case of heavy noise, small objects size, inhomogeneous lighting, larger inter-class & intra-class variance.

## Region-based Segmentation

**Region Growing:** For segment generation in grayscale or color images, start at one seed pixel, $(x, y, I(x, y))$, and add recursively adjacent pixels that satisfy a "similarity criterion" with pixels contained in the so far grown region around seed pixel.

## Region Splitting and Merging Segmentation

**Region Splitting:** Starts with the whole image as a single region and subdivides it into subsidiary regions recursively while a condition of homogeneity is not satisfied.

**Region merging:** Opposite to splitting, to avoid over-segmentation.

↳ Start with a small region, and merge the regions that have similar characteristics (gray level, variance, etc.).

## Clustering based Segmentation

↳ Organizing data into classes such that – high intra-class similarity, low inter-class similarity.

↳ Finding the class label & the no. of classes directly from the data.

↳ Cluster by features.

### ↳ Mean shift segmentation

↳ Partitioning image into meaningful regions or objects by grouping pixels with similar features, like color.