Computer Vision

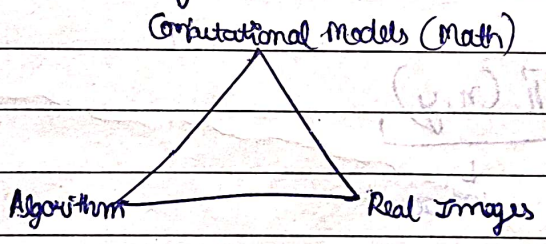
1. Introduction

→ Computer programs that can interpret images.
Understanding the representation of image

→ Applications:

- Images & movies
  ex- Surveillance
  Building 3D representations → Earth, Maries
  Motion capture → Pirates of Caribbean, COI
- OCR and Face Recognition.
  ↓
  Scanner → Adobe, Supermarket
- Object Recognition
- Special Effects & 3D Modeling
- Smart Cars
- Sports → Cricket, soccer.
- Vision Based Interaction → MS Kinect, Nintendo Wii
- Security and surveillance.

Note: Image Processing & computer vision is different.



2. Image Processing for CV

Image is a function, f or I, from R^2 to R

Blurry Image = Smooth function

$f(x,y)$ gives the intensity or value at position (x, y)

Practically, define image over a rectangle $f: [a, b] \times [c\ d] \rightarrow [\min, \max]$

with a finite range.

Note: 0 to 255 is dumb measurement (0 to 1 should be better)

for color images

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

* pixel → picture element.

- Image Blending

$$g(x) = (1-\alpha) f_0(x) + \alpha f_1(x) + \gamma$$

(usually zero)

dst = cv2.addWeighted(img1, 0.7, img2, 0.3, 0)

→ Digital Images:

In computer vision we typically operate on digital images.

Sample → the 2D space on a regular grid.

Quantize → each same (round to "nearest integer")

Images, thus are represented as matrix of integers.

→ Noise in Images

$$\vec{I}'(x,y) = \vec{I}(x,y) + \underbrace{\vec{n}(x,y)}_{\text{noise}}$$

Types of Noise

- Salt & pepper: Random occurences of black & white pixels
- Impulse: random occurences of white pixel.
- Gaussian: variations in intensity drawn from a Gaussian normal distribution

↓

* noise = randn(size(im)) * sigma ← better depends on range of image

output = im + noise

Image Filtering → used to
→ Enhance images
→ Extract information
→ Detect patterns

⇒ Removing noise from an image:

Approach 1: (Moving Average) 1-D
Replace each pixel with an average of all the values in its neighbourhood.

- Assumptions
1. The "true" value of pixels are similar to the true value of pixels nearby.
2. The noise added to each pixel is done independently.

Approach 2: (Weighted Moving image) 1-D
Can add weights to our moving average
So, we use an odd number of non-uniform weighted mask
Eg. [1, 4, 6, 4, 1] to all nearby neighbourhood

For 2-D

Approach 1: Moving Average

Just take some 2×2 or 3×3 matrix and average it.

* Co-relation filtering - uniform weights
Say the averaging window size is $(2K+1) \times (2K+1)$

$$G[i,j] = \frac{1}{(2K+1)^2} \sum_{u=-K}^{K} \sum_{v=-K}^{K} F[i+u, j+v]$$

Nested loop

↓ & averaging it.

☆ Co-relating filtering - nonuniform weights

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]\, F[i+u, j+v]$$

This is called cross-relation, denoted $G = H \otimes F$

This $H[u,v]$ is called filter "kernel" or "mask" which is just the matrix of weights in the linear combination.

So, what makes a good kernel?

Approach 2: Gaussian Filter

$H(u,v) \rightarrow$

1	2	1
2	4	2
1	2	1

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

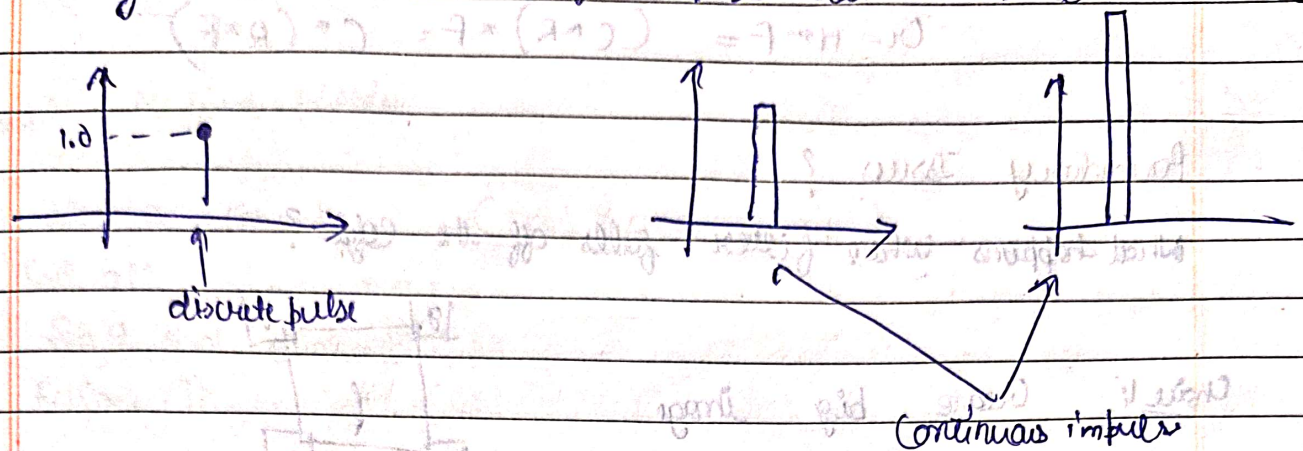
This kernel is approximation of Gaussian function

Note: Variance (σ^2) or standard deviation (σ) determines extent of smoothing
Also, Size of Kernel (u, v) is not variance

⇒ Impulse Noise

In discrete world, Impulse is just a value of 1 at a single location.

while in continuous world, impulse is an idealized func^n that is very narrow and very tall so that it has a unit area.



- **Convolution**

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]\, F[i-u, j-v]$$

$$G = H * F$$

Also, Img * Impulse = Img

→ Properties of Convolution:

a) Linear & Shift invariant

b) $f * g = g * f$ (Commutative)

c) $(f * g) * h = f * (g * h)$ (Associative)

d) $f * e = f$ (Identity)

e) Differentiation $\frac{\partial}{\partial x}(f * g) = \frac{\partial f}{\partial x} * g$

A Big Note: What would be the computational complexity?

$\underbrace{W * W}_{\text{filter}} * \underbrace{N * N}_{\text{Imag}}$ to get the convolution

so, if some filter is seperable i.e we can create convolution by $W * N * N + W * N * N = 2 * W * N * N \ll W^2 N^2$

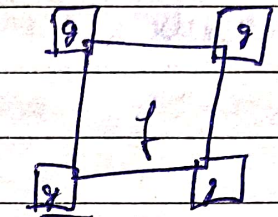
It can be done because

$$G = H * F = (C * R) * F = C * (R * F)$$

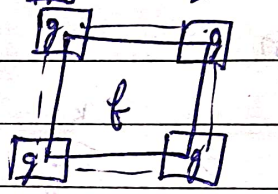
Boundary Issues ?

What happens when filter falls off the edge ?

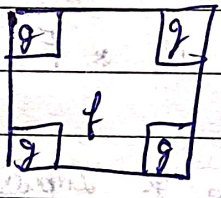
Choice 1: Create big image



Choice 2: Create same image



Choice 3: Create smaller image



Methods ? (In openCV) [For same image]

i) cv2.BORDER_CONSTANT → just clips out extra & replace with some color

ii) cv2.BORDER_WRAP → Circular wrapping

iii) cv2.BORDER_REPLICATE → Last element is replicated throughout (No obvious edges)

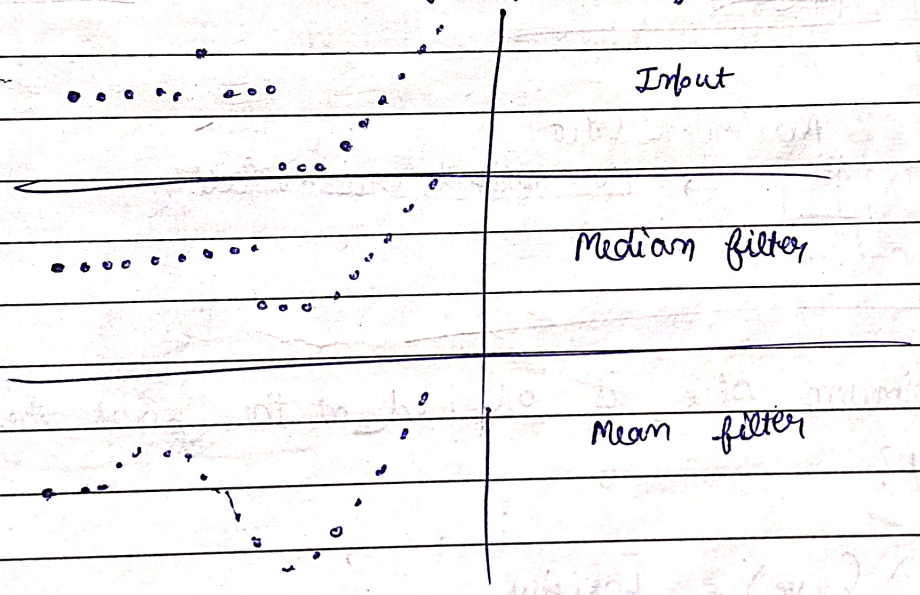
iv) cv2.BORDER_REFLECT → Border will be mirror reflection of the border elements.

Now, talking about filter for noises like salt & pepper noise, impulse noise

Approach 3: Median Filter

- Select a Kernel (odd × odd = odd)
- Sort all the numbers
- Select mid element
- Replace it

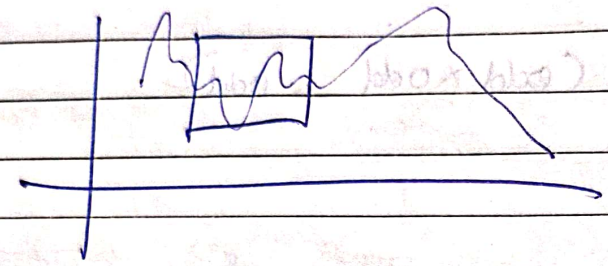
It is also called edge preserving bc

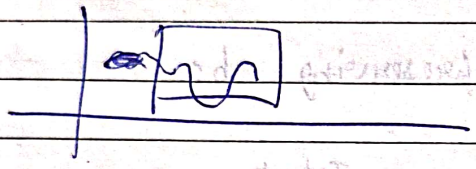


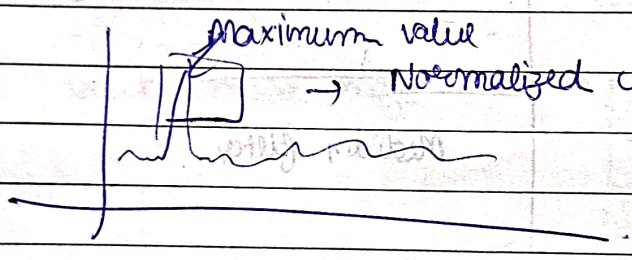
Now, we move on to finding properties of pixels locally in the image

⇒ Template Matching :

- 1D (nx) correlation







Why maximum value is attained at the point they match completely?

(+ve) x (+ve) = positive
(-ve) x (-ve) = (+ve)

Well, the filter max out each signal when they match exactly.

☆ We can now use this trick to find objects in the image (Template Matching)

⇒ Here, this doesn't work well for different scales & orientations.

So, it's just an appln of using filters as template

⇒ Edge Detection : Gradients

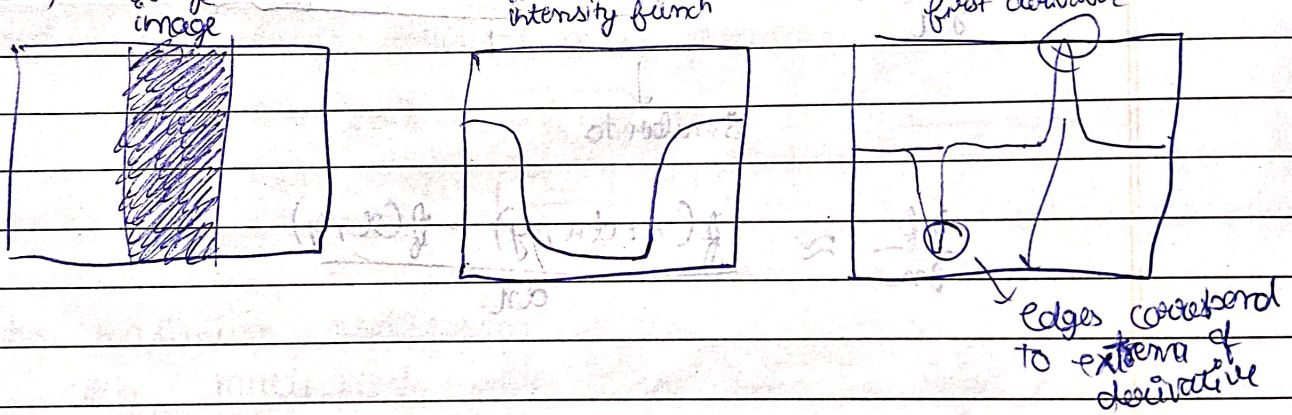
Edges in an image is important, they can easily convey information with reduced images.

Motive → $f(x, y) \xrightarrow{\text{reduce}}$ reduced set of pixels or curves that are important elements of the picture.

Now, edges look like steep cliffs / change, so look for a neighborhood with strong signs of change.
But, there are some ambiguity

- neighborhood size.
- how to detect change

Now, change is derivative (as an edge is a place of rapid change in the image intensity func")



→ To, find these peaks (extrema of derivate), we run it through some operators (differential operators to be exact)

- Differential operators - when applied to some image returns some derivatives (duh?)
- Model these "operators" as masks / kernels that compute the image gradient func".
- Threshold the this gradient func" to select the edge pixels.

But, what is gradient?

Gradient is a vector that's made up of the partial derivatives

The gradient of an image: $\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$

In other words, the gradient points in the direction of most rapid increase in intensity, & magnitude of vector is how much it's changing as a function of a unit step in that direction.

For computers, we calculate discrete gradient

$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+1,y) - f(x,y)}{1}$$

↓ Similar to

$$\frac{\partial f}{\partial x} \approx \frac{f(x+dx, y) - f(x,y)}{dx}$$

Some operators

1. Sobel operator

$$S_x = \frac{1}{8} * \begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array} \qquad S_y = \frac{1}{8} * \begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline \end{array}$$

It is a joint gaussian smoothing plus differentiation operator, so it is more resistant to noise.

$$\nabla I = [g_x \; g_y]^T$$

→ Gradients to edges:

Primary edge detection steps:

1. Smoothing derivatives to suppress noise and compute gradient.
2. Threshold to find "regions" of "significant" gradient
3. "Thin" to get localized edge pixels
4. and link or connect edge pixels.

So, to do that we use canny edge detection algorithm.
It's stages are:

1. Filter image with derivative of Gaussian
2. Find magnitude & orientation of gradient

$$\text{Edge-gradient } (G) = \sqrt{G_x^2 + G_y^2}$$

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

3. Non-maximum suppression
   Thin multi-pixel wide "ridges" down to single pixel width.
   In short, the result you get is binary image with "thin edges".

4. Linking and Thresholding (Hysteresis):
   - Define two thresholds: low and high
   - Use the high threshold to start edge curves and the low threshold to continue them

☆ Some concerns

1. What's the value of σ in gaussian filter

   Canny with $\sigma = 1$ (fine features) (detailed edge image)
   Canny with $\sigma = 2$ (less features) (large scale edges)

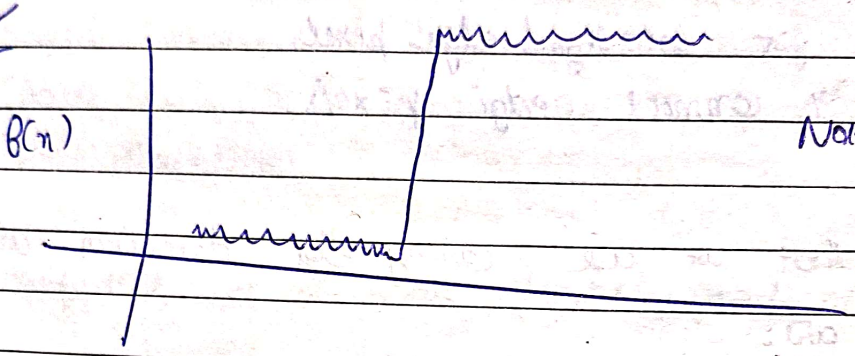
2. What is the kernel size to be taken?

Mathematics

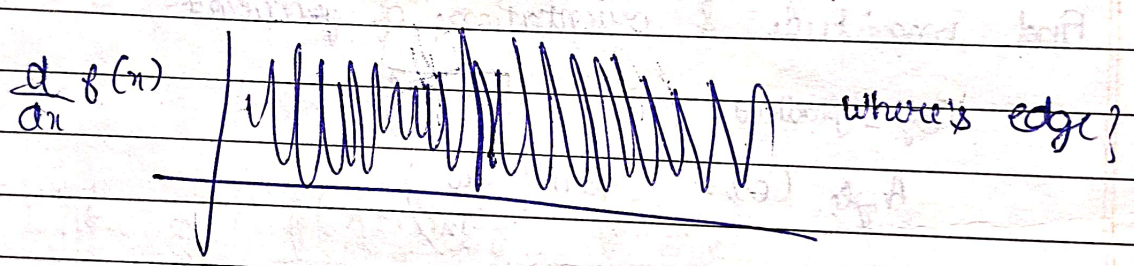
f → Image func^n.

h → filter (Gaussian)

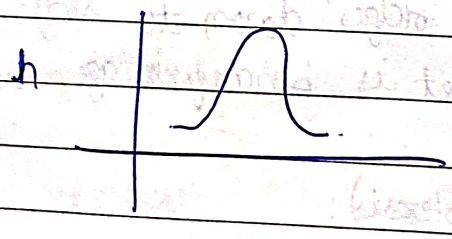
For 1-D

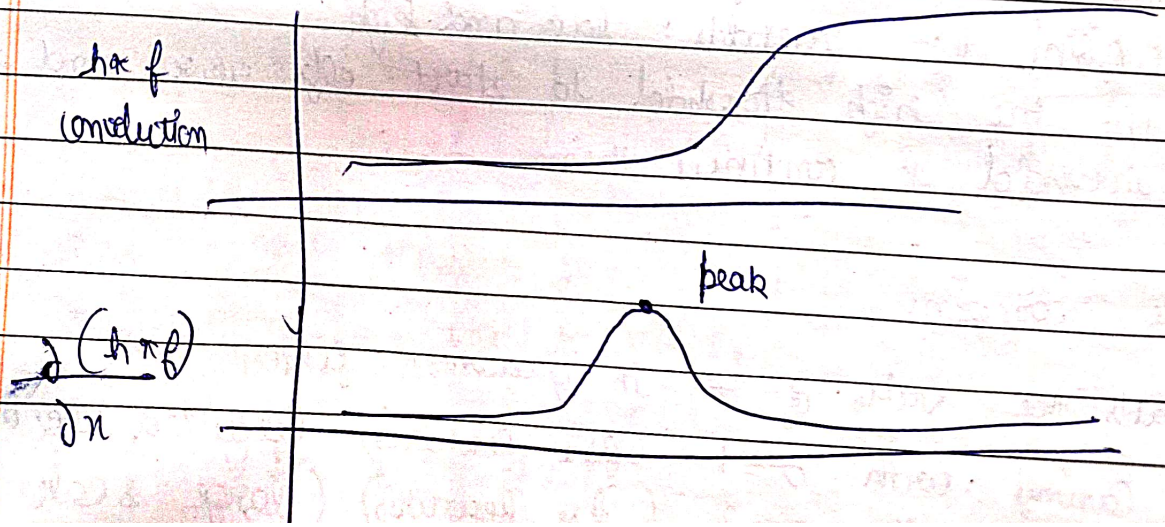


Apply derivative operator

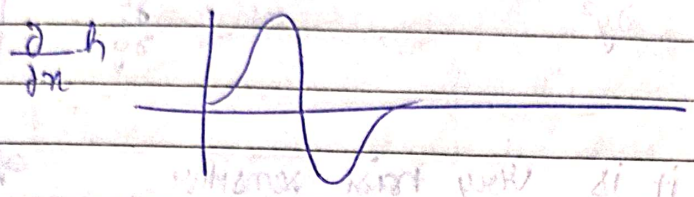


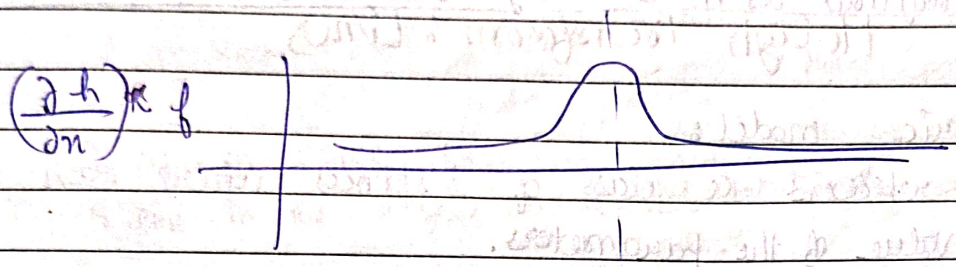
So, we need to remove noise



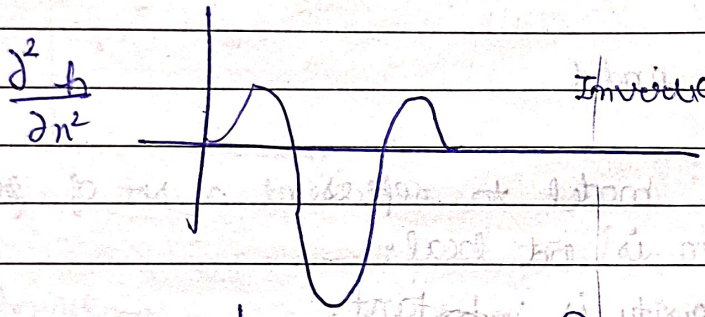


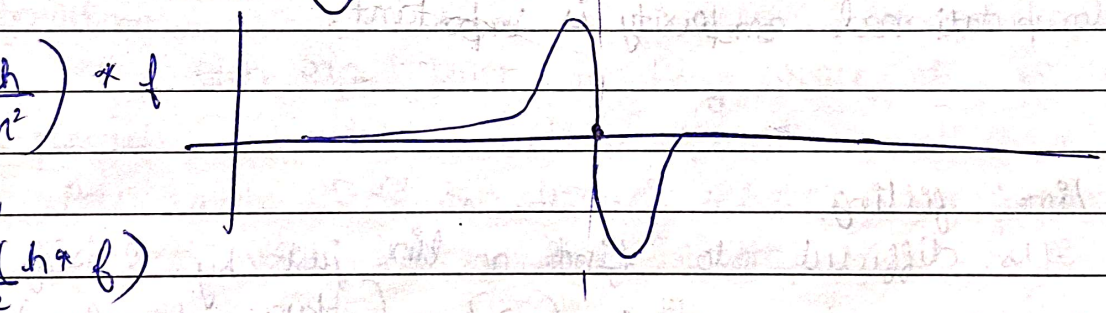
Also, $\frac{\partial (h * f)}{\partial x} = \left(\frac{\partial h}{\partial x}\right) * f$





Now, to find peaks, we take more derivatives





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

For 2-D (It is a single 2D edge detection filter)
It's a little bit harder, as there's more than one direction to take our derivative.

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

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

$\frac{\partial^2 h}{\partial (x,y)^2}$ is Mexican Hat.

$$\nabla^2 h = I_{xx} + I_{yy}$$
$$= \left([1 \; -2 \; 1] + \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix} \right) * I$$
$$= \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} * I$$

Laplacian ←

So, we use something called Laplacian operator

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

where $\frac{\partial^2}{\partial x^2} = [1 \; -2 \; 1]$

$\frac{\partial^2}{\partial y^2} = \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix}$

Central diff approx to second derivative

Laplacian operator

* Smoothing required as it is very noise sensitive.

Hough Transform : Lines

⇒ Parametric model:

It can represent a class of instances where each is defined by a value of the parameters.

Ex- lines, circles, parameterized template

→ Fitting a parametric model

- Choose a parametric model to represent a set of features.
- Membership criterion is not local:
- Computational complexity is important.

I. Line fitting

It's difficult to find a line just by looking at the points. so we use voting (:) (Hough transform)

It is a general technique where we let the features vote for all models that are compatible with it.

1. Cycle through features, each casting votes for model parameters.
2. Look for model parameters that recieve a lot of votes.

→ Hough space

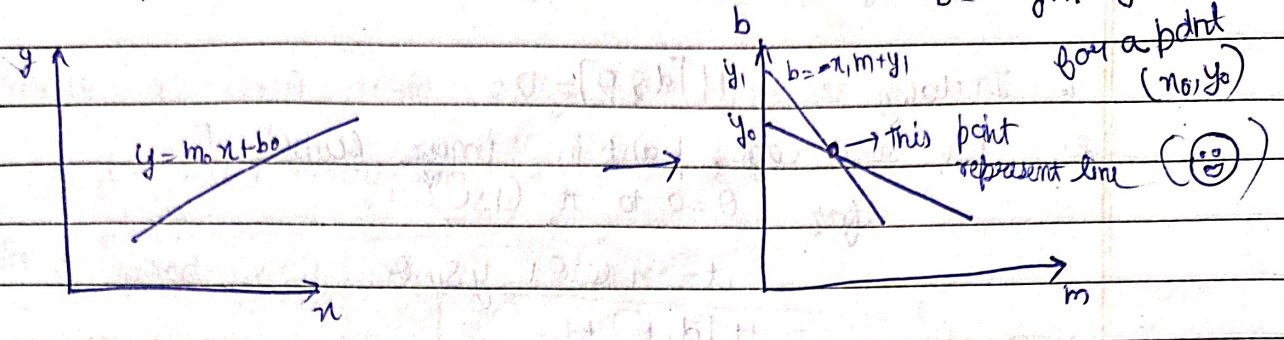
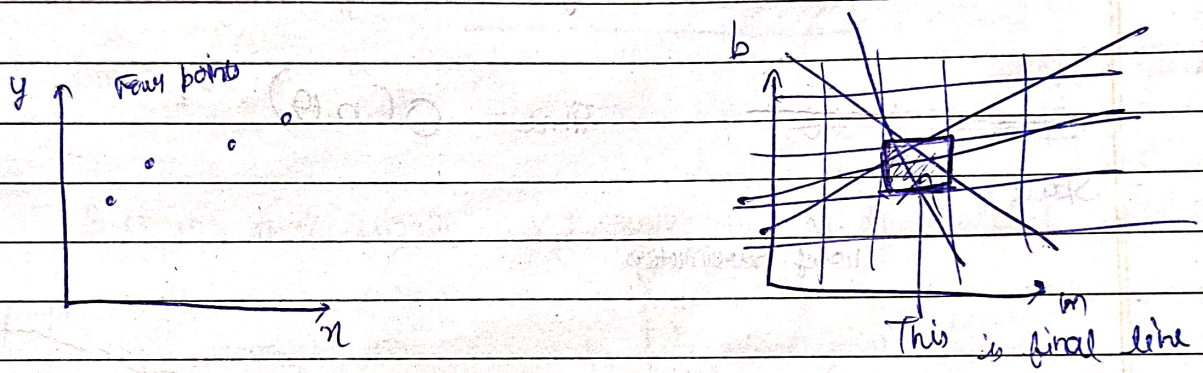


Image Space

Hough (parameter) space

A point in image space is line in Hough space
A line in the image correspond to Hough space point

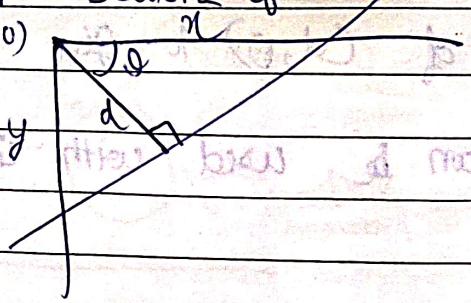


Algorithm:

1. Let each edge points in image space vote for a set of possible parameters in Hough space.
2. Accumulate votes in discrete set of bins; parameters with the most votes indicate line in image space.

• A little gotcha

If line is vertical $m \to \infty$, so we use polar representation of a line. to prevent any mathematical errs.



d: ler distance from line to origin

θ: angle the ler makes with x-axis

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

- The polar representation of image causes the Hough space to have sinusoidal waves for each point.

→ Basic Hough transform algorithm

1. Initialize $H[d, \theta] = 0$
2. For each edge point in image func^n F
   for $\theta = 0$ to π (180°)
   $d = x\cos\theta + y\sin\theta$
   $H[d, \theta) ++$
3. Find the value(s) of (d, θ) where $H[d, \theta]$ is max.
4. The detected line is given by
   $d = x\cos\theta - y\sin\theta$

~~Space = O(n θ)~~ Time = $O(n\theta)$

Space = $O(K^n)$
↓
no. of parameters

There are some extensions to the above algo., that helps with real line segments etc.

I. Using the gradient (knowing the alignment at (x, y) in image space
   instead of using $\theta = 0$ to π
   use, θ = grad. or $(-35$ to $55)$

II. Give more votes for stronger edges.

III. Change the sampling of (d, θ) to give more/less resolution.

IV. The same procedure can be used with circles, squares or any other shape

II. Circle

Consider a circle with center (a, b) & radius r

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

Method 1: For a fixed radius r

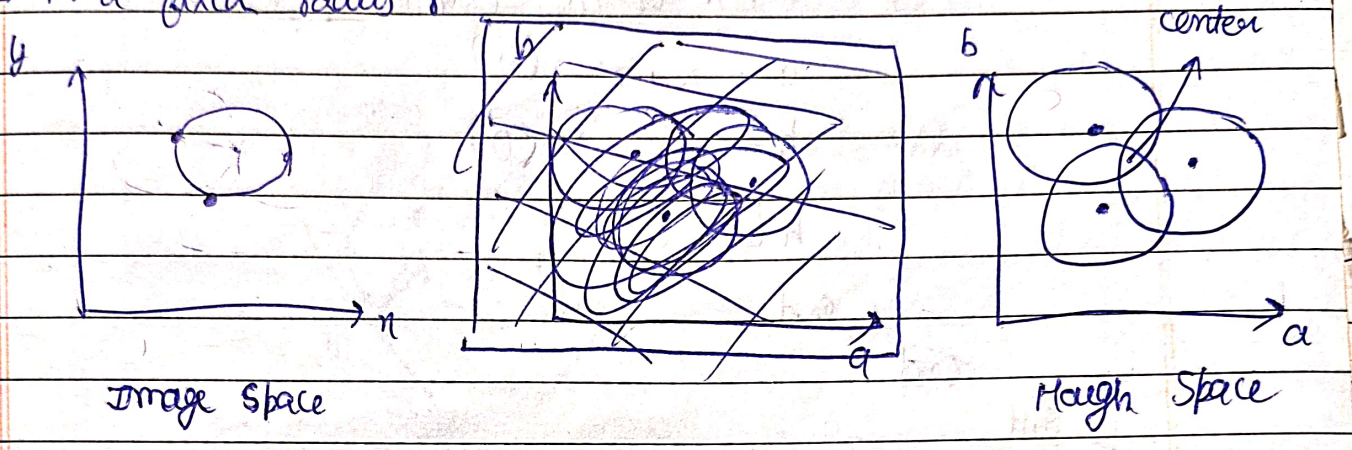
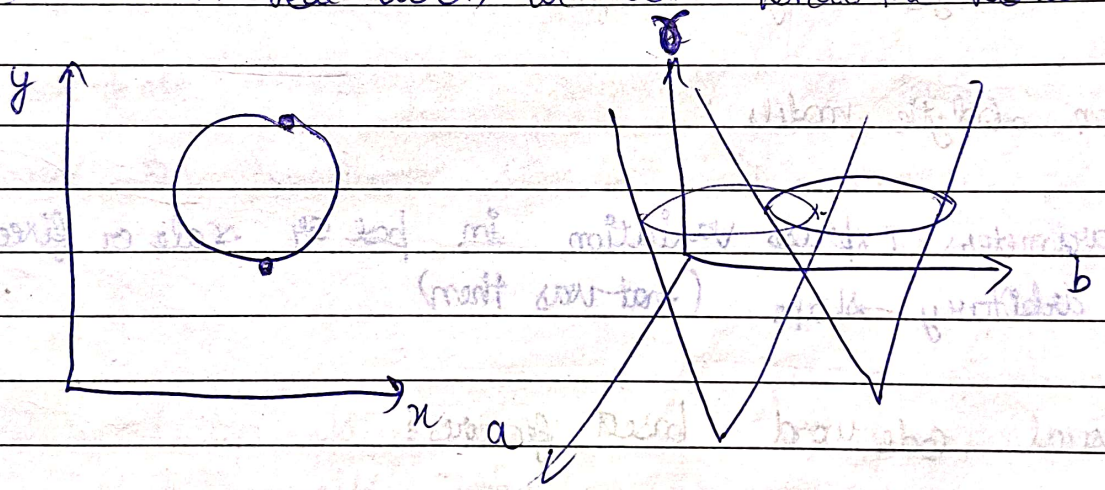


Image Space

Hough Space

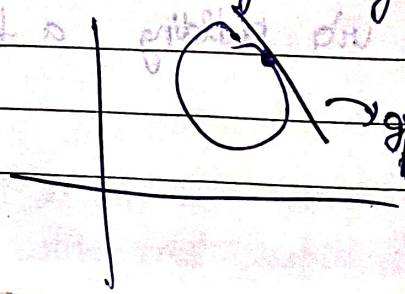
Method 2: But in real word, we don't know the radius so



The Hough space increases drastically. (RANSAV solves it)

Method 3: Unknown r, with gradients (reduces hough space)

Vote only along a line or circle.



i) If r is known line

ii) If r is un-known circle

→ Algorithm

1. For every edge pixel (x, y)
   For each possible radius value r:
      For each possible gradient direction θ:
         % % or use estimated gradient
         a = x + r cos(θ)
         b = y + r sin(θ)
         H[a, b, r] += 1
      end
   end
end

III. Generalized Hough Transform

⇒ Non analytic models :

Parameters express variation in pose or scale or fixed but arbitrary shape (that was then)

⇒ Visual code-word based features :

Not edges but detected templates learned from models (this is "now")

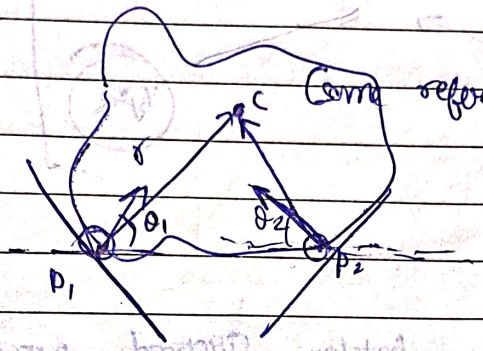
Moving back to generalized Hough transform

In this, voting is done via building a hough table.

how to build Hough table?

Training

1. At each boundary point, compute displacement vector: $r = c - p_i$.
2. Measure the gradient angle θ at the boundary point.
3. Store that displacement in a table indexed by θ.



Recognition

1. At each boundary point, measure the gradient angle θ.
2. Look up all displacements in θ displacement table.
3. Vote for a center at each displacement.

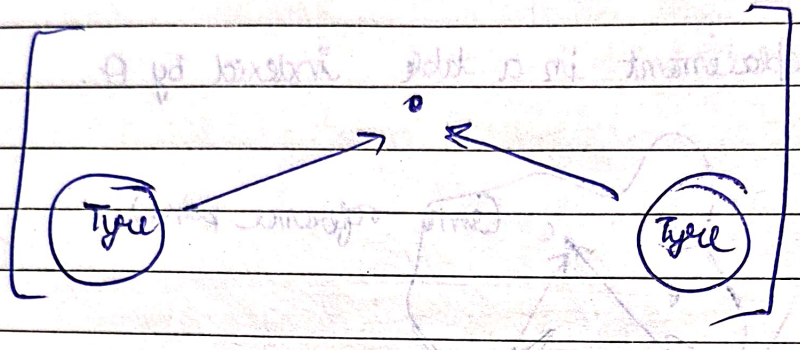
Algorithm

If orientation is known:
    do recognition point
else
    For each edge point
        For each possible master θ^*
            Compute gradient direction θ
            New $\theta' = \theta - \theta^*$
            For θ' retrieve displacement vectors to vote for reference point.

Peak in this Hough space (now x, y, θ^*) is reference point with most supporting edges.

- Modern Approach:

Instead of using indexed displacements by gradient orientation index by "visual codeword".



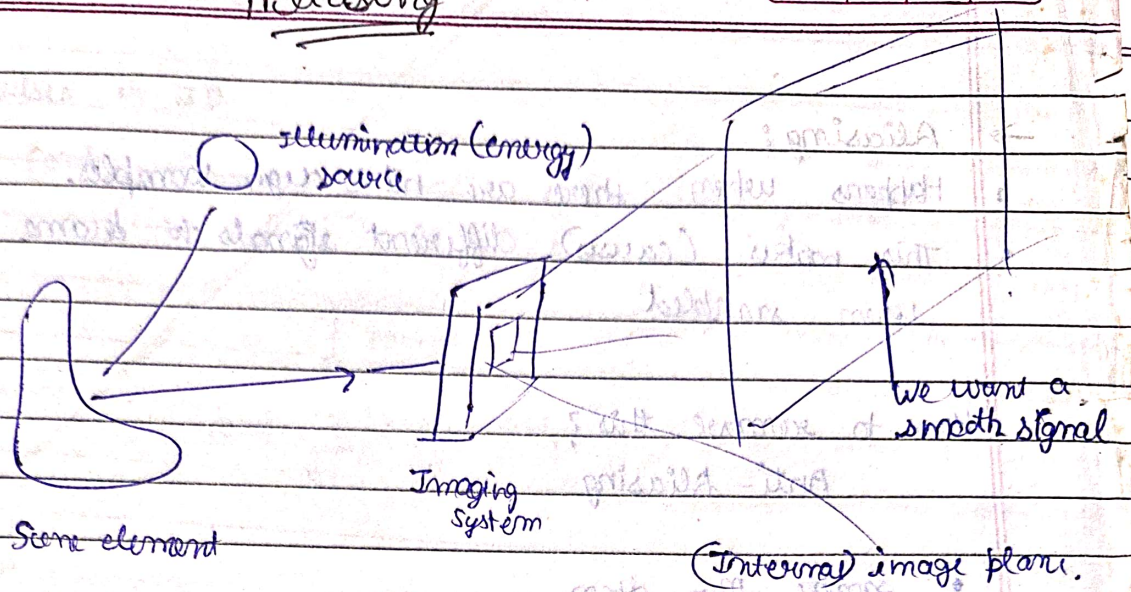
Training

1. Build codebook of patches around extracted (interest points) using clustering.
2. Map the patch around each "interest point" to closest codebook entry.
3. For each codebook entry, store all displacements relative to object center.

Recognition

1. Find all the "interest point" patches & identify visual codeword.
2. Look up all the displacements in codbook.
3. Vote for center at each displacement.

Aliasing

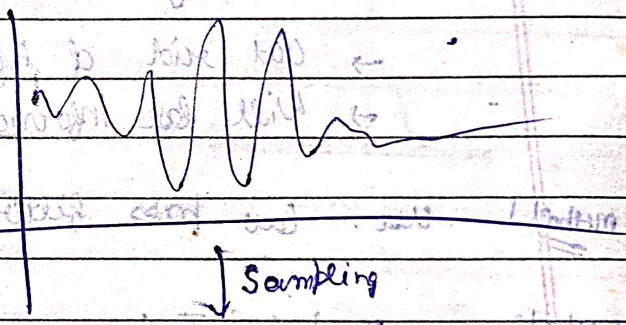


→ Sampled Representations

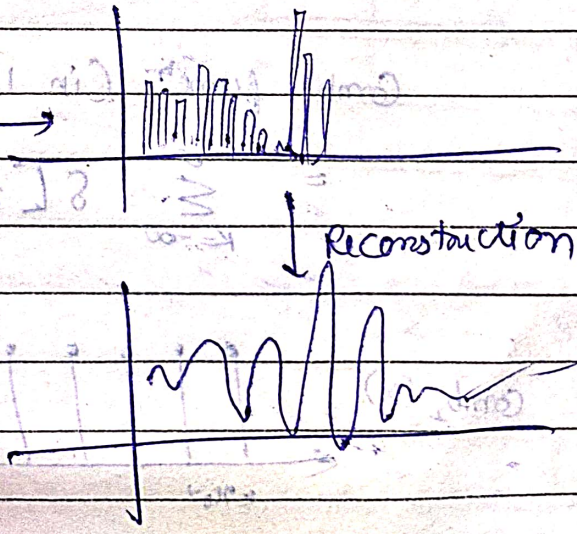
- Sampling → How to store & compute with continuous funcⁿ.

* If we do undersampling, information is lost.

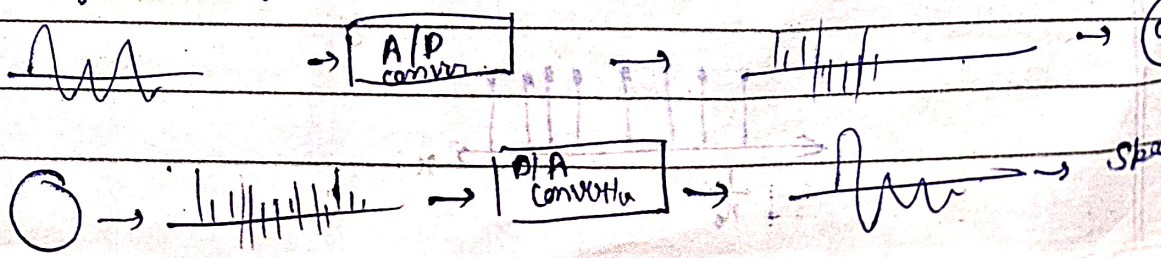
But, it was indistinguishable from lower frequency.



Goal → "guess" what this funcⁿ did in between.



- Sampling in digital audio



→ Aliasing:

- Happens when there are not enough samples.
- This makes (causes) different signals to become indistinguishable when sampled

How to remove this?

Anti-Aliasing

- Sample More often
  - → But this can't go on forever
- Make the signal less "wiggly"
  - → Get rid of higher frequencies
  - → Will lose information

Method 1 Use low pass filters (Image is stored already)

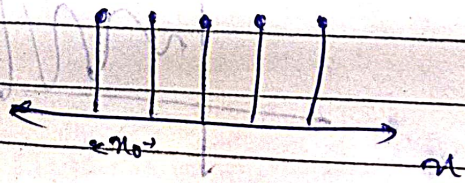
Method 2 Impulse Train

Comb func^n (in 1D)

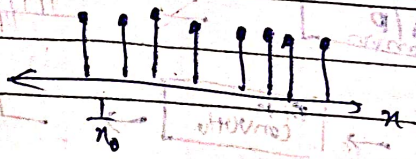
$$= \sum_{k=-\infty}^{\infty} \delta[x - kn_0]$$

$\delta(x) = 1$ if $x = 0$

$\delta(x) = 0$ otherwise



↕ Fourier Transform (Scaling property)



Page No.
Date

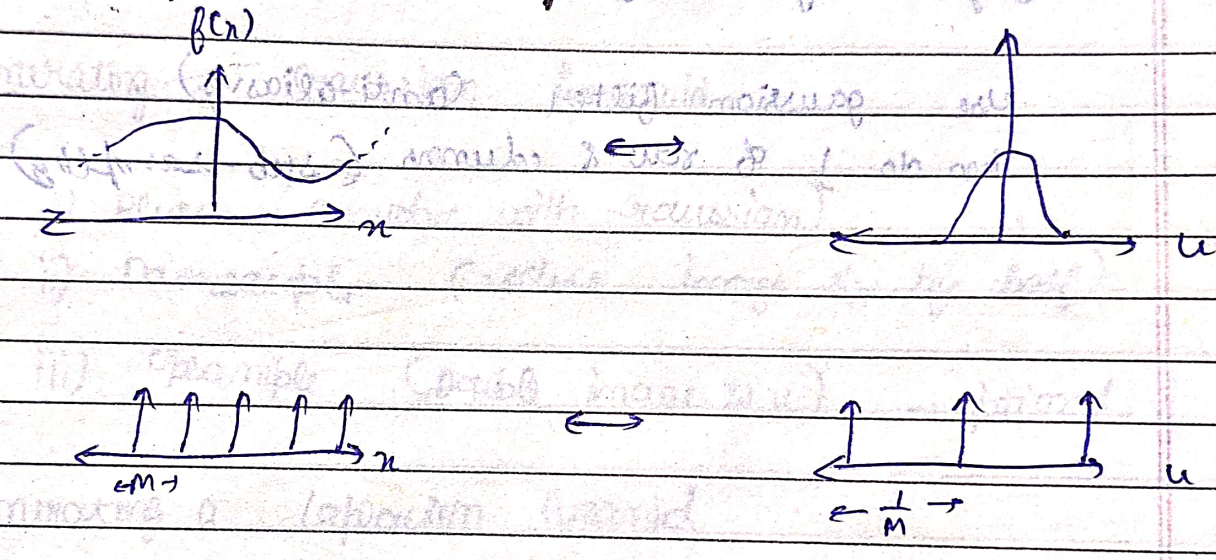
Impulses in 2D

$$\text{comb}_{M,N}(x,y) = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} \delta(x - kM, y - lN)$$

$$\updownarrow \text{ Fourier}$$

$$\text{comb}_{\frac{1}{M},\frac{1}{N}}(u,v)$$

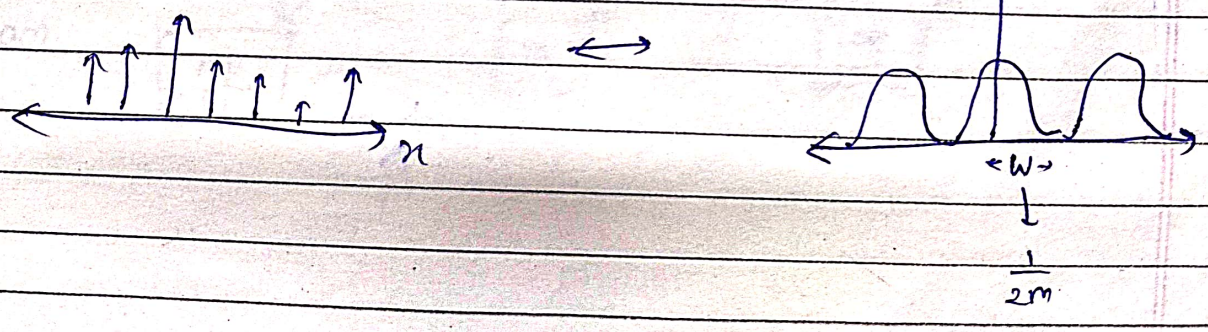
Ie Sampling law frequency



· Multiply (sample)

$f(x) \cdot \text{comb}_M(x)$

$F(u) * \text{comb}_{\frac{1}{M}}(u)$



If there is no overlap.
$W < \frac{1}{2M}$, the original signal can be recovered from its sample by low pass filtering.

Page No.
Date

II. Sampling high frequency:

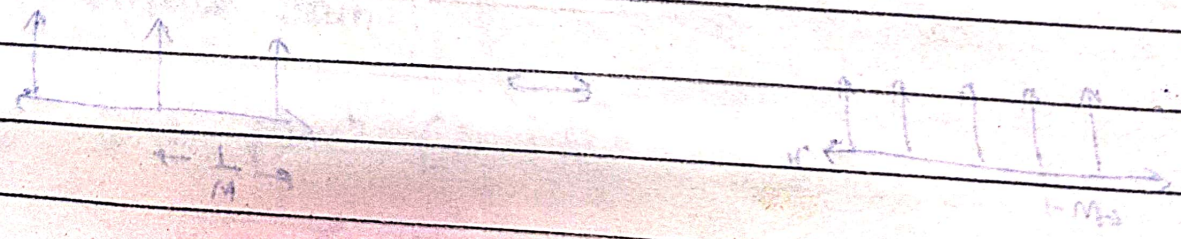
→ Apply an anti-aliasing filter

→ Then do comp func^n. (sub

→ Aliasing in Images:

Resizing image by $\frac{1}{2}$

Use gaussian filter (anti-aliasing)
Then do $\frac{1}{2}$ of row & column. (sub-sampling)



Page No.
Date

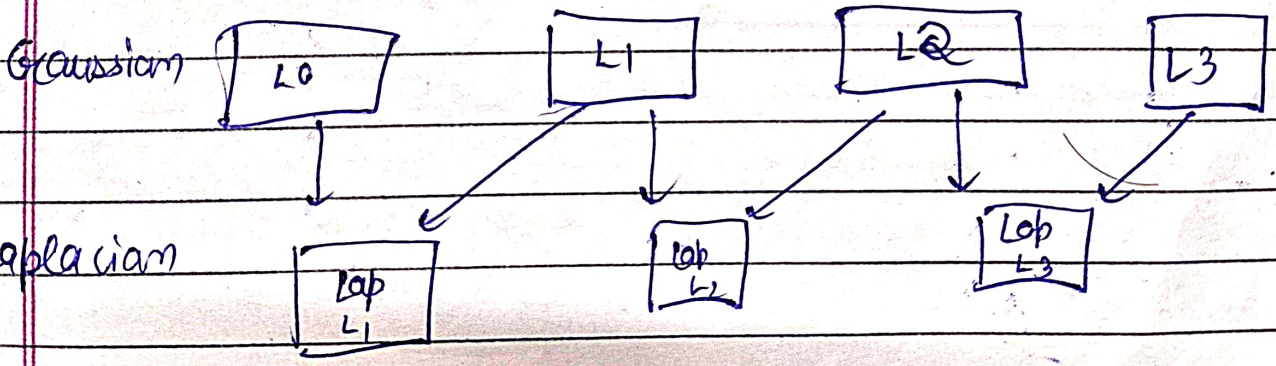
⇒ Image Pyramids

- There are two kinds of Image Pyramids
  - i) Gaussian Pyramid
  - ii) Laplacian Pyramids

We use these pyramids, when we need to work with images of different resolution of the same image.

- Generating a Gaussian Pyramid
  - i) Blur (Convolve with Gaussian)
  - ii) Downsample (reduce image size by half)
  - iii) Upsample (Double image size) → optional

- Generating a Laplacian Pyramid



→ Reduce & Expand :

$[1\ 4\ 6\ 4\ 1] / 16$ → Reduce] Repeatable 5-tap filter

$[\frac{1}{8}\ \frac{1}{2}\ \frac{3}{4}\ \frac{1}{2}\ \frac{1}{8}]$ → Expand] 3-tap filter

Earlier times in 80s, working on image was studying of frequency stuff and signals. Still, some thing needs to be understood for CS in 2020s.

Fourier Transform

Idea: Every image/signal can be decomposed into set of things that describe things.

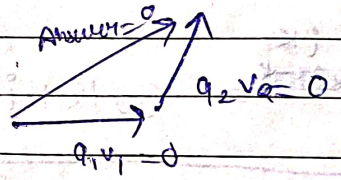
It is an important image processing tool which is used to decompose an image into its sine and cosine components.

⇒ Decomposing an image: basis sets $(\hat{i}, \hat{j})$

Suppose that $B = \{v_1, v_2, \ldots, v_n\}$ is a finite subset of a vector space V over a field F (such as real or complex numbers). Then B is a basis if it satisfies:

- Linear independence

  For all $a_1, \ldots a_n \in F$, if $a_1 v_1 + \cdots + a_n v_n = 0$, then necessarily $a_1 = a_2 = \cdots a_n = 0$



- Spanning property

  For every x in V it is possible to choose $a_1, \ldots, a_n \in F$ s.t.

  $x = a_1 v_1 + \cdots + a_n v_n = 0$

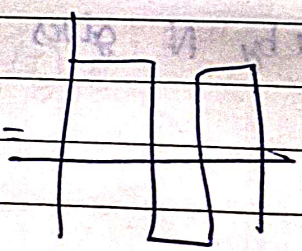
We need a good basis set, One is fourier basis set.

⇒ Fourier Theory:

Any periodic func^n can be written as a sum of weighted sines and cosines of different frequencies.

$$g(t) = A_1 \sin(\omega_1 t + \phi_1) + A_2 \sin(\omega_2 t + \phi_2) + \cdots$$

e.x.

Square wave =  $= A \sum_{K=1}^{\infty} \frac{1}{K} \sin(2\pi K t)$

* for decomposing our signal, we want to understant the frequency ω of our signal, phase is irrelevant as for CV we are not interested in reconstructing the image.

$f(x)$ → [Fourier Transform] → $F(\omega) = A\sin(\omega x + \phi)$

$f(x)$ ↓ Image func^n

$F(\omega)$ ↓ center of mass in both x & y directions

where $F(\omega) = R(\omega) + i\, I(\omega)$

$A = \pm \sqrt{R(\omega)^2 + I(\omega)^2}$

$\phi = \tan^{-1}\left(\frac{I(\omega)}{R(\omega)}\right)$

$$F(u) = \int_{-\infty}^{\infty} f(x)\, e^{-i2\pi ux}\, dx$$ (Fourier Transform)

$$f(x) = \int_{-\infty}^{\infty} F(u)\, e^{i2\pi ux}\, du$$ (Inverse Fourier Transform)

$$e^{ik} = \cos k + i \sin k$$

★ $F(\omega)$ gives us the basis set.

How?

$$\int_{-\infty}^{\infty} \sin(ax+\phi)\, \sin(bx+\phi)\, dx = 0, \text{ if } a \neq b$$

Independent

★ $f(x)\, e^{-i2\pi ux}$ gives the winding machine exactly like 3Blue1Brown

& integrating it (& dividing by N) gives centre of mass and ignoring divide by N gives

→ Discrete FT

$$F(K) = \frac{1}{N} \sum_{x=0}^{N-1} f(x)\, e^{-i \frac{2\pi k x}{N}}$$

where K is cycles per (Period of signal)

For 2-D

$$F(u,v) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x,y)\, e^{-i2\pi(ux+vy)}\, dx\, dy \quad \frac{1}{2}$$

$$F(k_x, k_y) = \frac{1}{N} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y)\, e^{\frac{-i\,2\pi\,(k_x x + k_y y)}{N}}$$

Thus, we get the basis vector.

⇒ Fourier Transform & Convolution:

$$g = f * h$$

$$G(u) = \int_{-\infty}^{\infty} g(x)\, e^{-i2\pi ux}\, dx$$

$$= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(\tau)\, h(x-\tau)\, e^{-i2\pi ux}\, d\tau\, dx$$

$$= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \left[f(\tau) e^{-i2\pi u\tau}\, d\tau \right] \left[h(x-\tau) e^{-i2\pi u(x-\tau)} dx \right]$$

$$x - \tau = x'$$

$$= \int_{-\infty}^{\infty} \left[f(\tau) e^{-i2\pi u\tau}\, d\tau \right] \int_{-\infty}^{\infty} \left[h(x') e^{-i2\pi ux'} dx' \right]$$

$$G(u) = F(u)\, H(u)$$

Convolution in spatial domain ⇔ multiplication in Frequency Domain

$g = f * h \quad \longleftrightarrow \quad G = F \cdot H$

$g = f \cdot h \quad \longleftrightarrow \quad G = F * H$

★ So, it's not uncommon to calculate F & H and multiplying them & then using IFT to get g.

FFT is Fast Fourier Transformation.

→ Smoothing & Blurring:

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

$$H(u) = e^{-\frac{1}{2}(2\pi u)^2 \sigma^2}$$] Also Gaussian.

But, σ is multiplied instead of dividing.

$G(u) = F(u) \cdot H(u)$] Thus, Attenuates Higher frequency.

↓IFT

(g)

⇒ Properties:

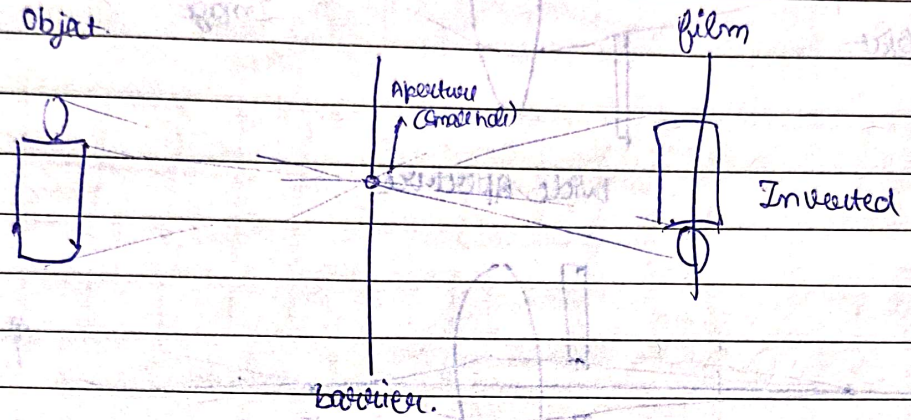
Linearity	$c_1 f(x) + c_2 g(x)$	$c_1 F(u) + c_2 G(u)$
Scaling	$f(ax)$	$\frac{1}{\lvert a \rvert} F\left(\frac{u}{a}\right)$
Differentiation	$\frac{d^n f(x)}{dx^n}$	$(i 2\pi u)^n F(u)$

3. Camera Models & Views:

Now, Image is a 2-D representation of 3D plane.

How to form an image?

a) Pinhole Perspective

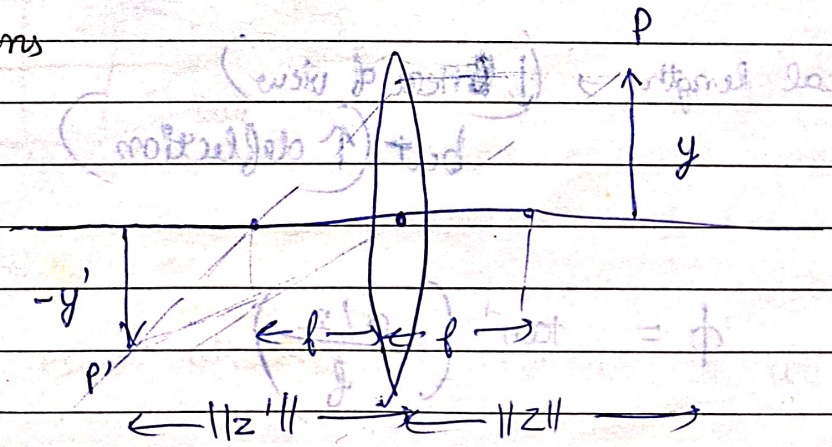


↓ Aperture size → Crisp Image

But if hole is too small, diffraction takes place

↓↓ aperture → ↓ crisp Image

b) Thin lens



$$\frac{-y'}{y} = \frac{\|z'\|}{\|z\|} \quad \& \quad \frac{-y'}{y} = \frac{\|z'\| - f}{f}$$

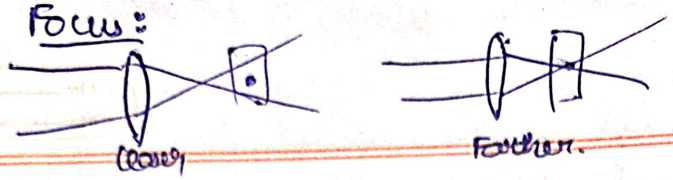
$$\Rightarrow \frac{\|z'\|}{\|z\|} = \frac{\|z'\| - f}{f}$$

$$\Rightarrow \frac{1}{f} = \frac{1}{u} + \frac{1}{v}$$

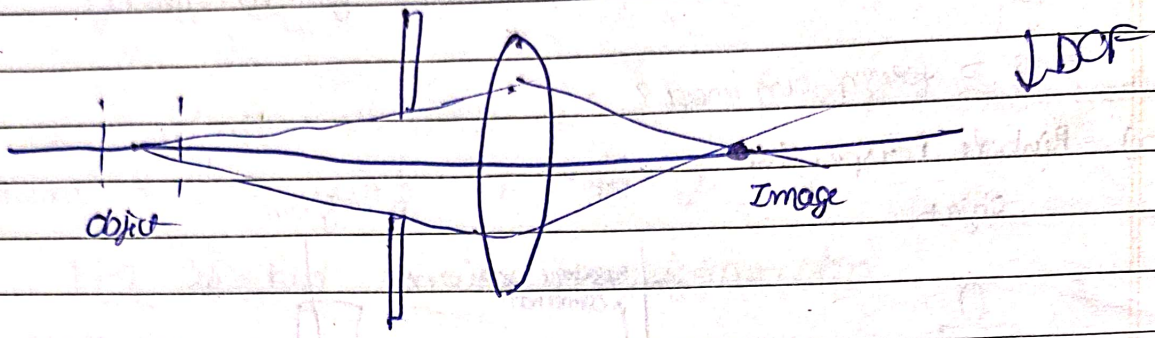
$u \to \|z\|$

$v \to \|z'\|$

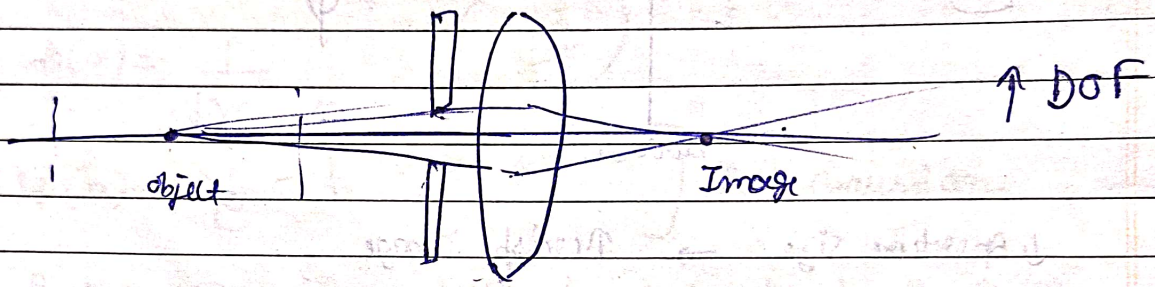
→ **Focus:**



→ **Depth of field:**



Wide Aperture



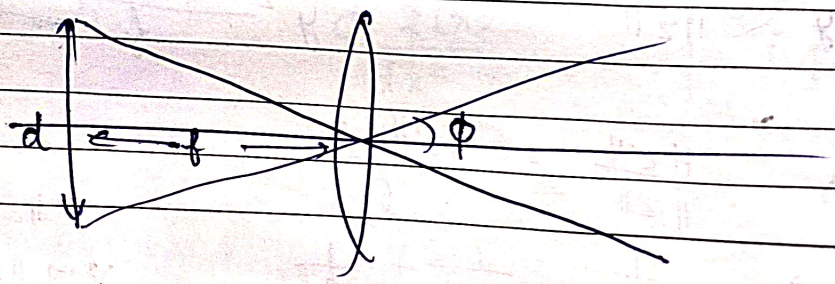
Small Aperture

→ **Field of view (zoom):**

↑ focal length → (↓ Field of view)
but (↑ deflection)

FOV ⇒ $\phi = \tan^{-1}\left(\frac{d/2}{f}\right)$

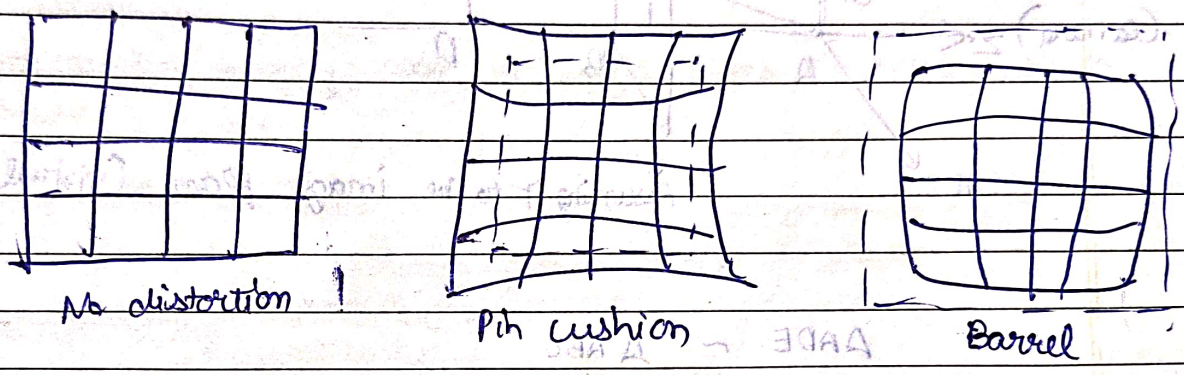
d → "retina" size or sensor size



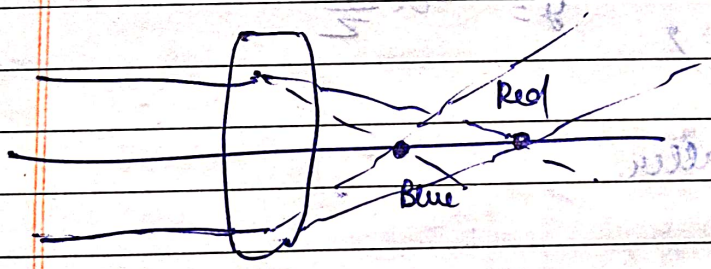
✱ Zooming & moving are not the same.

⇒ Problems of lens:
Lens are not perfectly thin

1. Geometric Distortion



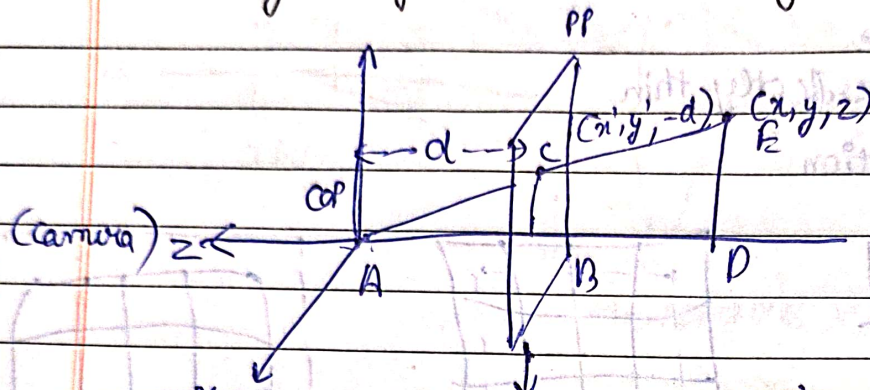
2. Radial distortion
3. Chromatic Aberration



4. Vignetting
Dark edges around the corner in two lens system.

⇒ ~~Projecting~~ Perspective Imaging:

I. Modeling Projection - Co-ordinate system



$\triangle ADE \sim \triangle ABC$

$$\frac{x'}{X} = \frac{y'}{Y} = \frac{-d}{z}$$

$$\Rightarrow x' = -d\frac{X}{z} \quad , \quad y' = -d\frac{Y}{z}$$

∴ Distant objects are smaller.

Now $\frac{x}{z}$ & $\frac{y}{z}$ is not a linear transformation

So, add one more coordinate

$$(x,y) \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad (x,y,z) \Rightarrow \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Converting from homogenous coordina

~~For 2-D~~

$x = \frac{x}{w} \quad y = \frac{y}{w}$

~~For 3-D~~

$x = \frac{x}{w}, \quad y = \frac{y}{w}, \quad z = \frac{z}{w}$

3D point to 2D image plane

$$\begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} fx \\ fy \\ z \end{bmatrix} = \boxed{x' = \frac{fx}{z}} \quad \boxed{y' = \frac{fy}{z}}$$

Here now, it is homogenous

In perspective imaging, all parallel lines meet up at a vanishing point
e.g. Train tracks.

Mathematically,
A line in 3-space is represented as

$$x(t) = x_0 + at$$
$$y(t) = y_0 + bt$$
$$z(t) = z_0 + ct$$

$$x'(t) = \frac{fx}{z} = \frac{f(x_0 + at)}{z_0 + ct}$$

$$y'(t) = \frac{fy}{z} = f\frac{(y_0 + bt)}{z_0 + ct}$$

as $t \to \pm\infty$, for $c \neq 0$

$$x'(t) \to \frac{fa}{c}$$
$$y'(t) \Rightarrow \frac{fb}{c}$$

almost all
So, parallel (or not) lines converge to the same point.

⑩ ⇒ Other Projection operators:

3-d point position

1. Perspective : $(x, y, z) \to \left(\frac{fx}{z}, \frac{fy}{z}\right)$
2. Weak Perspective: $(x, y, z) \to \left(\frac{fx}{z_0}, \frac{fy}{z_0}\right)$
3. Orthographic: $(x, y, z) \to (x, y)$

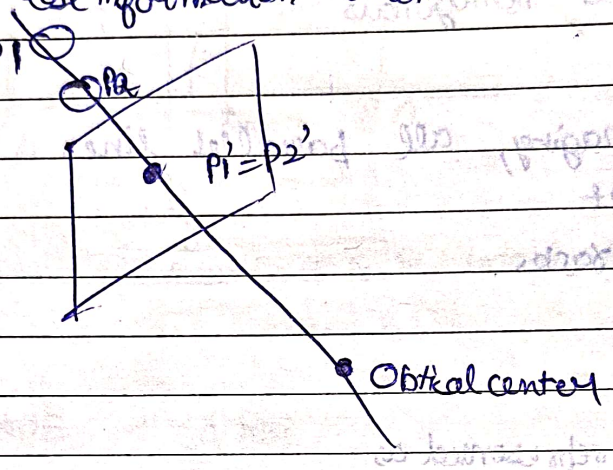
For State of the art algo.

visit → vision.middlebury.edu/stereo

⇒ Stereo - Geometry (Geometry of Multiple views)

The depth of a scene point along with structure is not directly accessible in a single image.

B/c we lose information when converting an image from 3D to 2D



The trick to solving this would be to look at it through a different perspective.

e.g. two eyes in animals, (vs)

* But, I can find depth with a single eye (Am I non-human?)

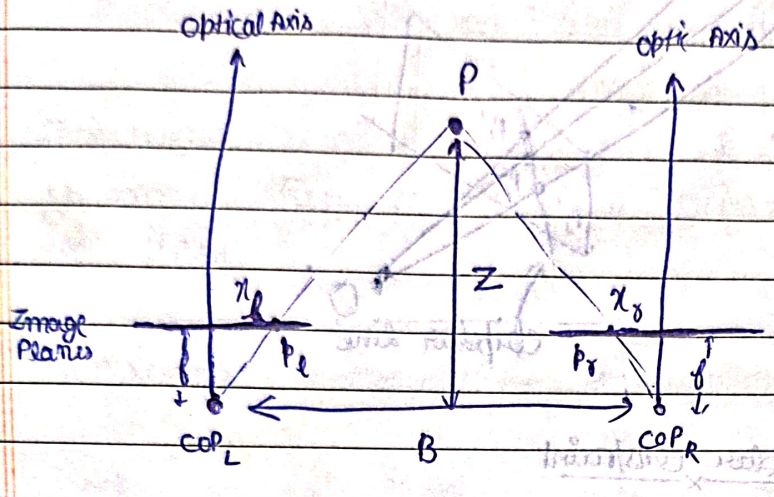
Hey dumb-head, there are other factors such as shading (shadow), texture, focus/de-focus, motion.

Also, we have two eyes

Stereo:

- The image from one eye is a little different than the image from the other eye.
- Think of shape from "motion" b/w two views.
- Infer 3-D shape of scene

- Basic Geometry



$p_l = x_l \qquad p_r = -x_r$

Now (p_l, P, p_r) & (C_L, P, C_R) are similar

So, $$\frac{B - x_l + x_r}{B} = \frac{Z - f}{Z}$$

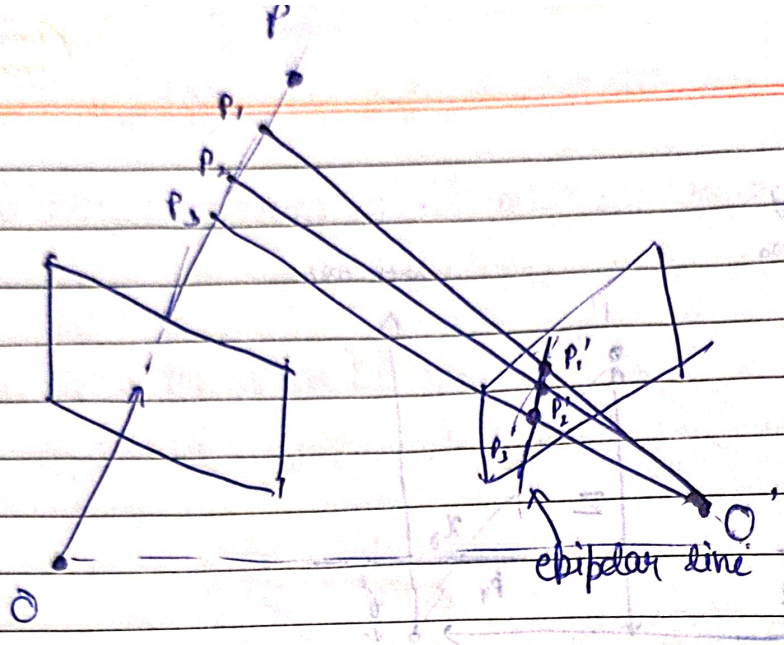
$$\Rightarrow \quad Z = f \frac{B}{x_l - x_r}$$

★ $x_l - x_r$ = disparity

& disparity $\propto \frac{1}{\text{depth}}$

Now, we need to find disparity but we don't know either of x_l or x_r.

we will talk about epipolar geometry.



Epipolar constraint

↓

Given a calibrated stereo rig, the set of possible matches for the point p is constrained to lie on the associated epipolar line l'.

Now, did we solve the correspondence problem?

↓

matching point in θ'

→ No, there are still a lot of points on the line

We need other "soft" constraints to help identify corresponding points

- Similarity
- Uniqueness
- Ordering
- Disparity gradient is limited.

1. Similarity

Assume image regions for the matches are similar in appearance.

For each pixel/ window in the left image

- compare with every pixel/window on the same epipolar line in right image
- Pick position with minimum match cost. (SSD, normalized correlation)

Cross-correlation might not work good.
It depends a lot on window size.

2. Uniqueness
No more than one match in right image for each window in left image.

3. Ordering
abc in left image is abc in right image.
won't always hold e.g. Consider transparent object.

e.g. Close one eye each time with both index fingers in front of nose and in a line.

* There are a lot of algorithms solving it, but there are still some challenges that stereo isn't possible to covers

- Low-contrast; textureless image regions.
- Occlusions
- Violation of brightness constancy (e.g. specular reflections)
- Larger Baseline.
- Camera Calibration errors.

Algo 1: Scaline (DP) → Does Have streching marks.
Algo 2: Full 2-D geid.
as Energy Minimization

$$E_{data} = \sum_i \left(W_1(i) - W_2(i + D(i)) \right)^2$$

$$E_{smooth} = \sum_{\substack{\text{neighbors} \\ i,j}} \rho\left(D(i) - D(j) \right)$$

$$\min. E = \alpha E_{data}(I_1, I_2, D) + \beta E_{smooth}(D)$$

World Coords → Camera Coords → Film Coords → Pixel Coords

rotation + Translation

$x' = f\frac{X}{Z}$

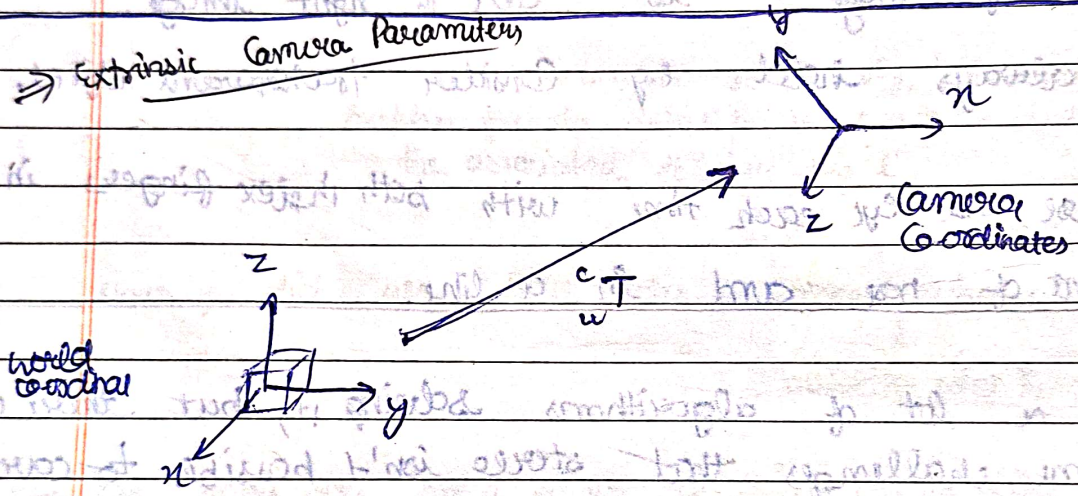
$y' = f\frac{Y}{Z}$

⇒ Geometric Camera Calibration: We want to use the camera to tell us things about the world. So, we need the relationship between coordinates in the world & co-ordinates in the image.

I. Extrinsic Camera parameters:
For arbitrary world co-ordinate system to the camera's 3D co-ordinate system. (Camera pose)

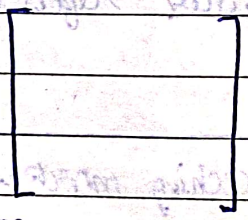
II. Intrinsic Camera parameters
From the 3D co-ordinates in the camera frame to the 2D image plane via projection.

⇒ Extrinsic Camera Parameters



$^{C}T_{W}$

* A rigid body has 6 degrees of freedom (cube)



Translation

⇒ a point A in world co-ordinate is translated to point B in camera co-ordinate with a relation of

$$^{B}P = {}^{B}(O_A) + {}^{A}P$$

Commutative ✓

$^{B}(O_A)$ → origin A expressed in B frame

We can express the same eqⁿ using matrix multiplication, with the help of homogenous co-ordinates that means linear transformation.

$$\begin{bmatrix} {}^{1}P_B \\ 1 \end{bmatrix} = \begin{bmatrix} I & {}^{B}(O_A) \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} {}^{A}P \\ 1 \end{bmatrix}$$

(3×3 Identity, 3×1)

$$P_B = {}^{A}P + {}^{B}(O_A)$$

Rotation

$$\vec{OP} = \begin{bmatrix} i_A & j_A & K_A \end{bmatrix} \begin{bmatrix} {}^{A}n \\ {}^{A}y \\ {}^{A}z \end{bmatrix} = \begin{bmatrix} i_B & j_B & K_B \end{bmatrix} \begin{bmatrix} {}^{B}n \\ {}^{B}y \\ {}^{B}z \end{bmatrix}$$

$${}^{B}P = \left({}^{B}_{A}R\right) {}^{A}P$$

$$\begin{bmatrix} i_A \cdot i_B & j_A \cdot i_B & K_A \cdot i_B \\ i_A \cdot j_B & j_A \cdot j_B & K_A \cdot j_B \\ i_A \cdot K_B & j_A \cdot K_B & K_A \cdot K_B \end{bmatrix}$$

Also, by homogenous co-ordinates.

$$\begin{bmatrix} {}^{B}P \\ 1 \end{bmatrix} = \begin{bmatrix} {}^{B}_{A}R & 0 \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} {}^{A}P \\ 1 \end{bmatrix}$$

Commutative ✗

Now, let's do a total rigid transformation,

→ A point 'A' in system is rotated and then translated (offset) in the A system to get point B in B system

Using homogenous co-ords

Transformation

$$\begin{bmatrix} {}^{B}p \\ 1 \end{bmatrix} = \begin{bmatrix} {}^{B}_{A}R & {}^{B}O_{A} \\ 0^{T} & 1 \end{bmatrix} \begin{bmatrix} {}^{A}p \\ 1 \end{bmatrix} = {}^{B}_{A}T \begin{bmatrix} {}^{A}p \\ 1 \end{bmatrix}$$

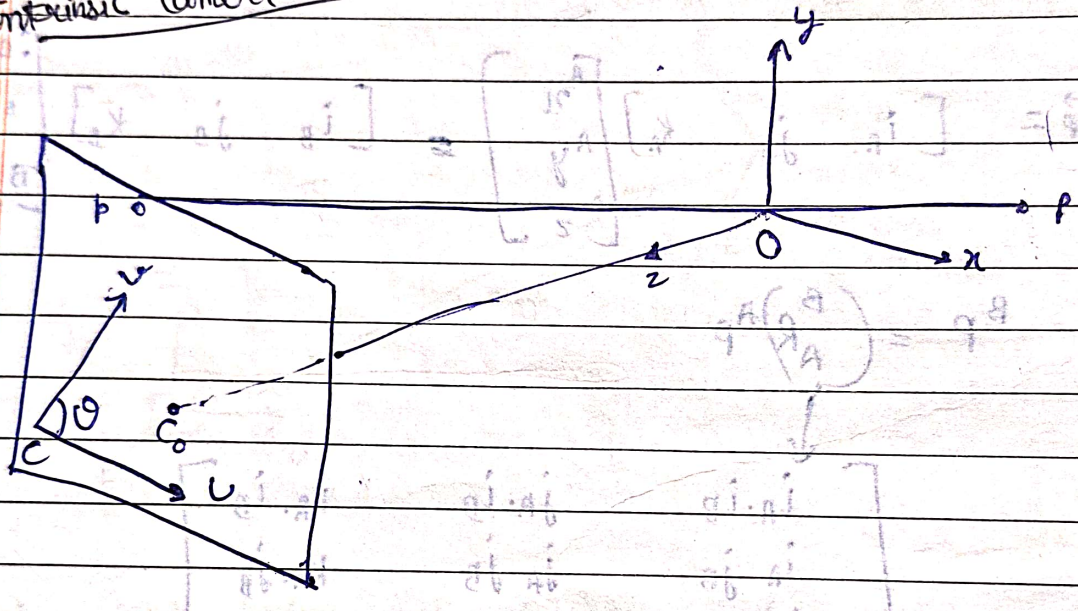
Point in Camera Frame

Extrinsic parameter matrix.

Point in world Frame.

6 degrees of freedom

⇒ Intrinsic Camera Parameter



$$x' = u = \alpha \frac{x}{z} + u_0$$

$$y' = v = \beta \frac{y}{z} + v_0$$

B/c of is not in mm
α is there
Now $\alpha \neq \beta$ so, we have two degrees of freedom.

Origin of camera co-ordinate system is at corner C of seting and not at it's center so u_0 & v_0

4 degrees of freedom.

(Principal point)

Also, the camera co-ordinate system may be skewed due to some manufacturing error, so angle θ b/w two image axes is not equal to 90°.

$$u = \alpha \frac{x}{z} - \alpha \cot\theta \frac{y}{z} + u_0$$

$$v = \frac{\beta}{\sin\theta} \frac{y}{z} + v_0.$$

So, 5 degrees of freedom.

$$\begin{bmatrix} zu \\ zv \\ z \end{bmatrix} = \underbrace{\begin{bmatrix} \alpha & -\alpha\cot\theta & u_0 & 0 \\ 0 & \frac{\beta}{\sin\theta} & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{K} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$\vec{p'} = K \; {}^{c}\vec{p}$$

$$\Rightarrow \text{Actual point} = \frac{\vec{p'}}{z}$$

→ Combining extrinsic & intrinsic calibration parameters

$$\Rightarrow \vec{p'} = K \begin{pmatrix} {}^{c}_{w}R & {}^{c}_{w}\vec{t} \end{pmatrix} {}^{w}\vec{p}$$

$$\rightarrow \vec{p'} = M \; {}^{w}\vec{p}$$

$\vec{p'}$ ↓ pixel coords

${}^{w}\vec{p}$ ↑ World coords

where

$$M_{(3\times4)} = \begin{bmatrix} f & s & x_c' \\ 0 & af & y_c' \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R_{3\times3} & 0_{3\times1} \\ 0_{1\times3} & 1 \end{bmatrix} \begin{bmatrix} I_{3\times3} & T_{3\times1} \\ 0_{1\times3} & 1 \end{bmatrix}$$

intrinsics Projection Rotation Translation

⇒ CAMERA CALIBRATION:

I. Calibration using known points (Direct linear transformation)

Place a known object in the scene

- Identify correspondence between image and scene
- Compute mapping from scene to image.

⇒ Now, we need to find all 11 degrees of freedom. It is done using SVD tricks

- Find the m that minimizes $\|Am\|$ subject to $\|m\| = 1$
- Let $A = UDV^T$ (SVD)

U, V: orthogonal matrices; D: diagonal matrix

- Therefore minimizing $\|UDV^T m\|$.
- But, $\|UDV^T m\| = \|DV^T m\|$ & $\|m\| = \|V^T m\|$

U & V are unit vectors

- Thus, minimize $\|DV^T m\|$ subject to $\|V^T m\| = 1$.

Let $y = V^T m$

So, $\|Dy\|$ is minimized s.t. $\|y\| = 1$

Also, D is diagonal with decreasing values.

So $\|Dy\|$ minimum is when $y = (0, 0, \ldots, 0, 1)^T$

Since $y = V^T m$, $\boxed{m = Vy}$ as V is orthogonal ($V^{-1} = V^T$)

↓ This gives last column in V.

★ & $A^TA = V^TD^2V$ (eigen decomposition)

The final soln is:

Given $Am = 0$, find the eigen vector of A^TA with smallest eigen value, that's m.

⇒ After finding M, we should be able to find things like the camera center from M

Two ways: Pure way, Easy Way

$M = [Q \mid m_4]$ $\quad Q = 3\times3$, $m_4 = 3\times1$

$C = -Q^{-1} m_4$

II. Multi-plane Calibration.

Advantages

- Requires a plane only (checker-board)
- Don't need to know positions/orientations
- Direct code available in open CV

Zhengyou Zhang (Good of Calibration)

⇒ Multiple Views

⇒ Image to Image Projections

I

$$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \approx \begin{bmatrix} sx' \\ sy' \\ s \end{bmatrix}$$

6 degree of freedom

Affine transformation

	D.O.F.
↳ Translation	2
↳ Rotation	1
↳ Scale	1
↳ Skew	2

II

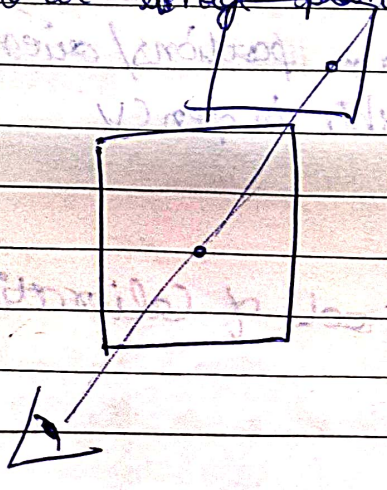
$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} w'x' \\ w'y' \\ w' \end{bmatrix} \approx \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$$ (Divide by w')

Homography

8 degree of freedom (This transformation maps for any linear between any two planes

⇒ Homographies and mosaics

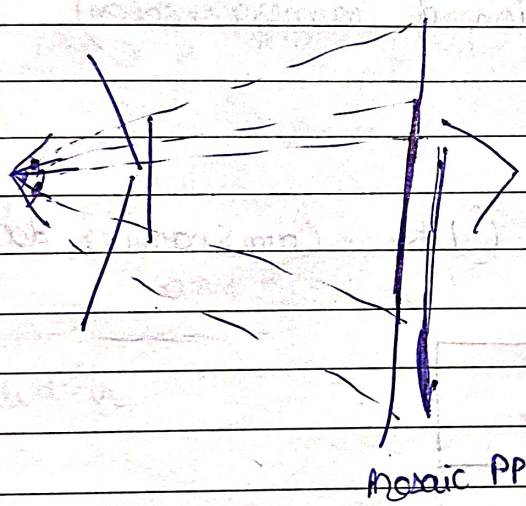
Rather than thinking of this as a 3D reprojection on 2D plane, we surely can think of it a a 2D image warp from one image plane to another image plane.



This gives an idea of panoramas.

→ How to stitch together a panorama.

- Take a sequence of images from the same position.
  > rotate the camera about it's optical center.
- Compute transformation b/w second image plane & first
- Transform the second image to overlap with the first.
- Blend the two together to create a mosaic
  (Repeat, if more than two)



The mosaic has a natural interpretation in 3D:

- The images are reprojected onto a common plane
- The mosaic is formed on this plane.

(Works for views upto 180°, Beyond that need to use sequence that "bends the rays" or map onto a different surface, say, a cylinder)

→ Now, how to solve homography?

- Since, homography is transformation b/w any two planes,

$$P' = HP$$

$$\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

(8 degrees)

- Requires at-least 4 points for 8 eqs.

Non-homogenous way

$Ah = b$ (h is vector of 8 degree of freedom)

Solve for h by $\min \|Ah - b\|^2$ using least-squares.

OR

Homogenous way

- Solve using SVD

→ Applications of Homography (Not mosaics dumby dumb)

i) Video motion (Computation of camera motion)

ii) Image Rectification

Rectifying slanted views (Like CamScanner does)

How do we do this?

⇒ Projective Geometry

In case of homogeneous co-ordinates

- Line joining two points p_1 & p_2 is given by

$$l = p_1 \times p_2$$

$$p_1 = [x_1 \; y_1 \; 1]$$

$$p_2 = [x_2 \; y_2 \; 1]$$

- Similarly, intersection of two lines

$$p_{12} = l_1 \times l_2$$

$$l_1 = [a_1 \; b_1 \; c_1]$$

$$l_2 = [a_2 \; b_2 \; c_2]$$

Before that,

Projective line (in image plane)

A line is a plane of rays through origin by the normal $\vec{l} = (a, b, c)$

All rays (x, y, z) satisfying:

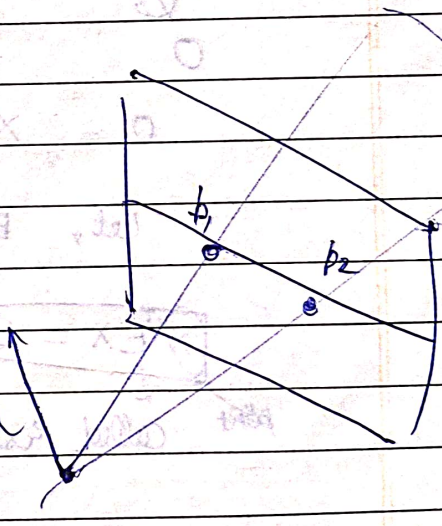
$$ax + by + cz = 0$$

Point & line duality

- A line l is a homogeneous 3-vector

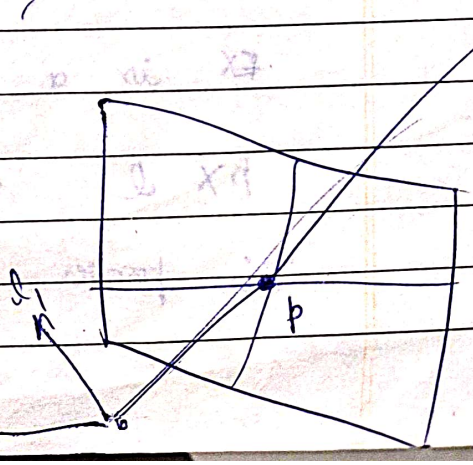
l is $\perp$ to p_1 & p_2

$$l = p_1 \times p_2$$



- A point p is $\perp$ or to l_1 & l_2

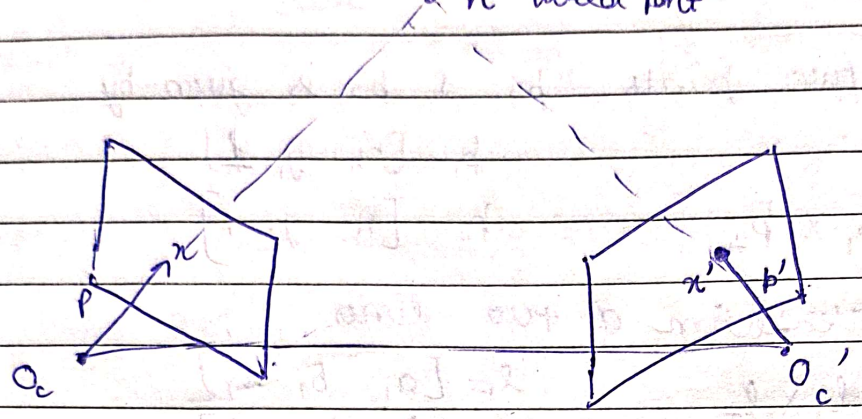
$$p = l_1 \times l_2$$



(Go to top now)

⇒ Essential Matrix

Consider again the stereo problem,



In case of calibrated cameras

$$X_c' = RX_c + T$$

$$TX\,X' = TXRX + TXT$$

$$TX\,X' = TXRX$$

$$X'.(TX\,X') = X'.(TXRX)$$

$$0$$

$$0 = X'.(TXRX)$$

Let, $E = TXR$

$$\boxed{X'^{T} EX = 0}$$ or $$X E X'^{T} = 0$$

point called essential Matrix

EX is a vector l (line)

$$pX\,l = \text{line}$$

series of

This proves a point, lies on a epipolar line

⇒ Fundamental Matrix

Motivation:

- estimate epipolar geometry from a set of point correspondence between two uncalibrated cameras.

We know, $p_{im} = K_{int} \underbrace{\Phi_{ext} P_w}_{P_c}$

$$\Rightarrow p_{im} = K_{int} P_c$$

$$\Rightarrow P_c = K_{int}^{-1} p_{im} \quad (K_{int} \text{ is invertible})$$

P_c: Camera → Image.

this gives the ray in space (as it is homogeneous)

and for two cameras

$$P_{c,left} = K_{int,left}^{-1} p_{im,left}$$

$$P_{c,right} = K_{int,right}^{-1} p_{im,right}$$

$$P_{c,right}^T E P_{c,left} = 0 \quad \text{(Look at it)}$$

$$\left(K_{int,r}^{-1} p_{im,r}\right)^T E \left(K_{int,l}^{-1} p_{im,l}\right) = 0$$

$$p_{im,r}^T \underbrace{\left(K_{int,r}^{-1} E K_{int,l}^{-1}\right)}_{F} p_{im,left} = 0$$

$$\boxed{p_{im,r}^T F p_{im,left} = 0}$$

$\Rightarrow l = F p'$ is the epipolar line in the b image associated with b' & vice versa ($l' = F^T b$)

→ Properties of Fundamental Matrix.

- Epipoles found by $Fp' = 0$ & $F^T p = 0$
  ↓
  a point which exist on every epipolar line

☆ In parallel images planes, epipoles are at infinity.

- F is singular (mapping from homogeneous 2-D point to 1-D family so rank 2)

- If we estimate fund. matrix from correspondences in pixel coords, can re-construct epipolar geometry without intrinsics or extrinsic parameters.

→ How to find it? (Note: FM gives relation btw Any two views That gives an idea of video motion)

- Each point generates one constraint or single eq^n on F.

$$p^T F p' = 0$$

Not Famous.

Requires 8 point atleast to solve for F (9 variables)

divide 1 variable

↓

8 variables left

Now, do OLS or orthogonal least squares.

More Used

SVD approach (very minimal errors)

F = Find eigen vector of $A^T A$ with minimum eigen value or OLS

Do another SVD (Since rank of F is 2)

$F = U D V^T$

$\hat{D}$ = set the last singular value to zero.

→ Applications of Stereo:

- Stereo Rectification
- Photo-Synth.

→ Summary:

- For 2-views, there is a geometric relationship between rays in one view & rays to the other → epipolar geometry.
- These relationships can be captured algebraically as well
  - → Calibrated : Essential Matrix
  - → Uncalibrated: Fund. Matrix
- This relation can be estimated from point correspondences

→ Another problem?

How to find these point correspondences?

Manually....

Nah - - -

We will use features to find these.

→ $\begin{bmatrix} u' & v' & 1 \end{bmatrix} F \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = 0$

u_i, v_i'

→ $\begin{bmatrix} u_i' u_i & u_i' v_i & u_i' & v_i' u_i & v_i' v_i & v_i' & u_i & v_i & 1 \end{bmatrix} \begin{bmatrix} f \end{bmatrix} = 0$

N points

$Ax = 0$

→ $\hat{F} = U \hat{D} V^T$

Computer Vision

Features & Matching

⇒ Introduction to "features":

To get a transformation of one Image to another image, we need corresponding points, and we do that using "LOCAL FEATURES".

Goal: Find points that are:

→ Found in other images

→ Found precisely

→ Found reliably.

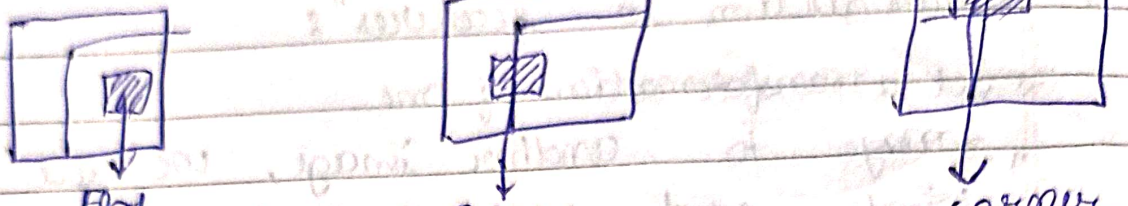
→ Characteristics of Good Features:

- Precision / Repeatability
  The same feature can be found in several images despite geometric & photometric transformations.

- Saliency / Matchability
  Each feature has a distinctive property.

- Compactness / efficiency
  Many fewer features than image pixels.

⇒ Finding Corners:



No change in all directions

No change along the edge direction

significant change in all direction with small shift

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

Now, in this eq^n, we want to look for small shifts
↳ (Derivative)
second-order Taylor expression of $E(u,v)$ about $(0,0)$

For 1-D $\quad F(\delta x) \approx F(0) + \delta x \frac{dF(0)}{dx} + \frac{1}{2}\delta x^2 \frac{d^2F(0)}{dx^2}$

For 2-D

$$E(u,v) \approx E(0,0) + [u\ v]\begin{bmatrix} E_u(0,0) \\ E_v(0,0) \end{bmatrix} + \frac{1}{2}[u\ v]$$

$$\times \begin{bmatrix} E_{uu}(0,0) & E_{uv}(0,0) \\ E_{uv}(0,0) & E_{vv}(0,0) \end{bmatrix}$$

$$\times \begin{bmatrix} u \\ v \end{bmatrix}$$

Now,

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

$$E_{uu}(u,v) = \sum_{x,y} 2w(x,y)\, I_x(x+u, y+v)\, I_x(x+u, y+v)$$

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

$$E_{uv}(u,v) = \quad " \quad I_y \quad " \quad I_x$$

$$" \quad I_{xy} \quad "$$

Putting all in eq^n, we get

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

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

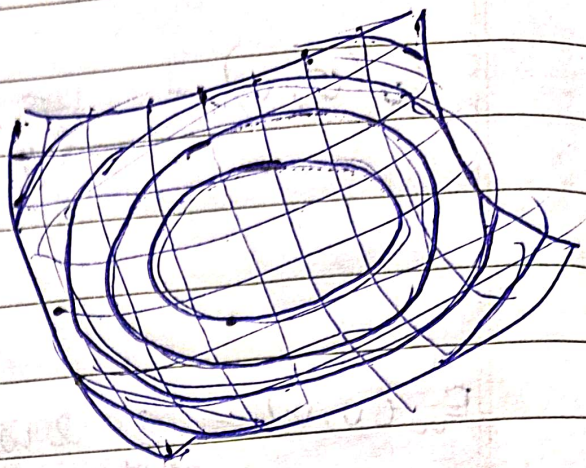
$$\Rightarrow \boxed{E(u,v) \approx [u\ v]\, M \begin{bmatrix} u \\ v \end{bmatrix}}$$

Interpreting it, (Consider a slice of $E(u,v)$)

$$\boxed{\sum I_x^2 u^2 + 2\sum I_x I_y\, uv + \sum I_y^2 v^2 = k}$$

The eqⁿ is of ellipse.

* Treat gradient vectors as a set of (dx, dy) points with a center of mass defined as being at (0,0)
* Fit an ellipse to that set of points via scatter matrix.



Now again consider the M matrix (second moment)

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

Case I: Gradients are either horizontal or vertical but not slanted

$$I_x I_y = 0$$

$$M = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$$

λ_1, λ_2 → a & b for ellipse

Case II: Gradient is only horizontal or only vertical

either $\lambda_1 = 0$ or $\lambda_2 = 0$

(Not a good corner)

Note: λ_1 & λ_2 determine the corner based on the property of diagonalization of M matrix and concluding them as eigen values

How?

Watch video again.

λ_1 & λ_2 are eigen-values of M.

Case I: λ_1 & λ_2 are too small
Flat region

Case II: $\lambda_1 \gg \lambda_2$ or $\lambda_2 \gg \lambda_1$
Edge

Case III: λ_1 & λ_2 are large, $\lambda_1 \sim \lambda_2$
Corner.

But, Harris algorithm makes use of response func^n

$$R = \det(M) - \alpha \, \text{trace}(M)^2$$

$$R = \lambda_1 \lambda_2 - \alpha (\lambda_1 + \lambda_2)^2$$

and α: constant (0.04 to 0.06)

So, $|R|$ small → Flat
$R < 0$ → Edge
$R \gg 0$ → Corner

→ Harris detector: Algorithm

1. Compute gaussian derivatives at each pixel.
2. Compute second moment matrix M in a gaussian window around each pixel.
3. Compute corner response func^n R.
4. Threshold R.
5. Find local maxima of response function (non-maximum suppression)

Thus, we get good features on both images.

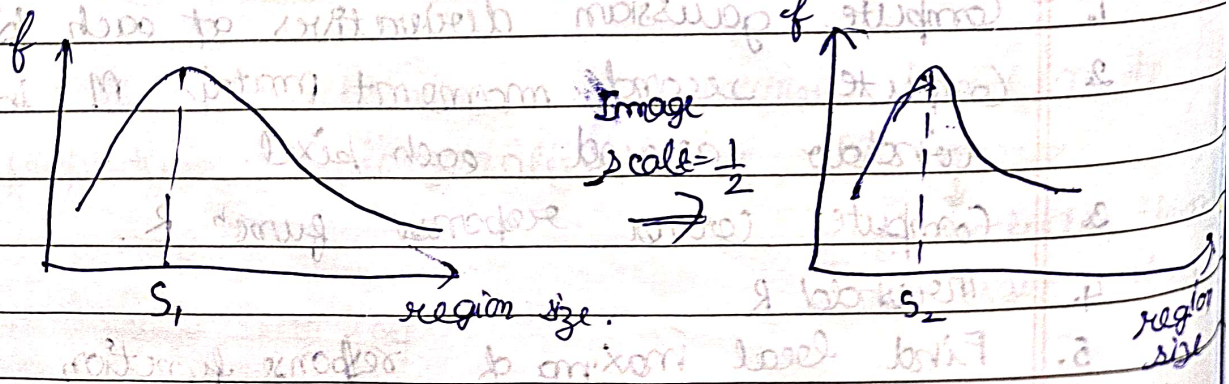
- Properties of Harris detector:

1. In-variant to Rotation.
2. Intensity — Invariant to Addition
   — Threshold Issue in Multiplication but invariant.
3. Not invariant to scale.

How to do scale invariance?

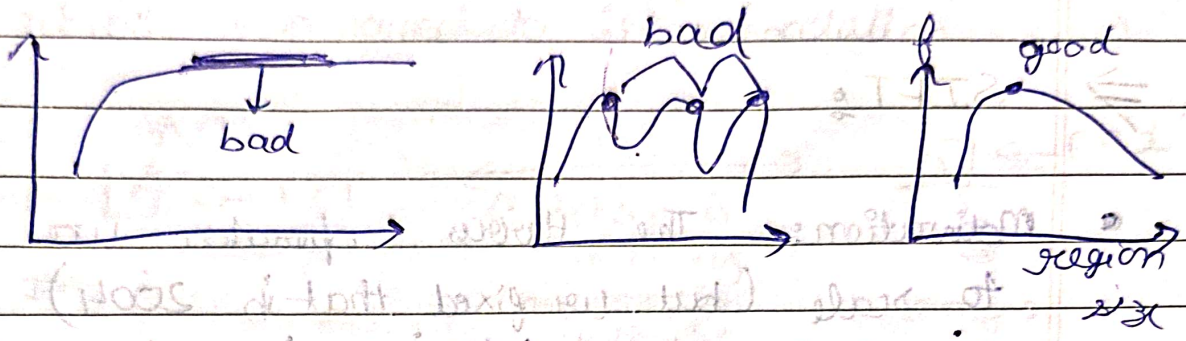
Soln:
Design a func^n on the region (circle), which is "scale invariant" - not affected by the size but will be the same for "corresponding regions", even if they are at different scales.

One Method:
At a point, compute the scale invariant func^n over different size neighborhood (different scales)



we get scale size S_1 & S_2

Another problem, we need a good func^n for scale invariant detection, that has one stable sharp peak.



* A good func^n would be a one which responds to contrast.

$$L = \sigma^2 \left(G_{xx}(x,y,\sigma) + G_{yy}(x,y,\sigma) \right)$$

↓

Laplacian of Gaussian - LoG

It is a little costly to find.

we instead use DOG, that is essentially the same

$$DoG = G(x,y,K\sigma) - G(x,y,\sigma)$$

↓

Difference of Gaussian

Both Kernels are invariant to scale & rotation

Now, we got interest points but we still need to match them distinctively, that is the work of descriptor.

⇒ SIFT:

- Motivation: The Harris operator was not invariant to scale (but we fixed that in 2004) and co-relation was not invariant to rotation
  ↓
  (Normal Bruteforce matching)

- Develop

Done ✓ Detector → invariant to scale & rotation
Descriptor → Robust to variations in viewing conditions.

- Idea of SIFT

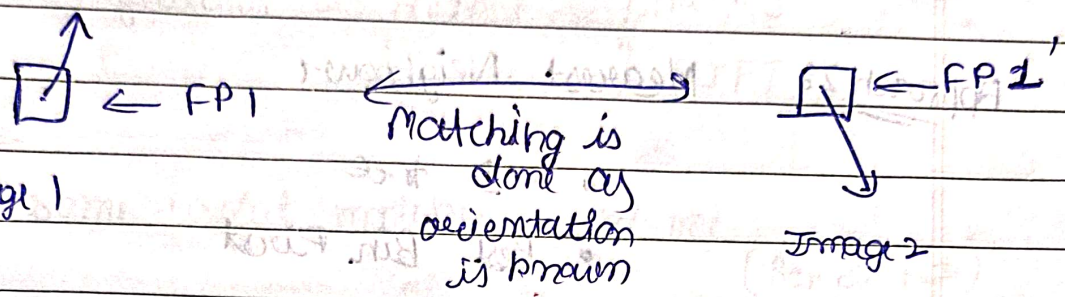
Match enough SIFT features that are invariant to scale, transformations, photometry etc. to do computer vision recognition.

- Overall SIFT procedure

1. Scale-space extrema detection } Or we Harris-Laplace or other method
2. Keypoint localization
3. Orientation Assignment
4. Keypoint Description

→ Orientation Assignment

An orientation is assigned to each keypoint to achieve invariance to image rotation.



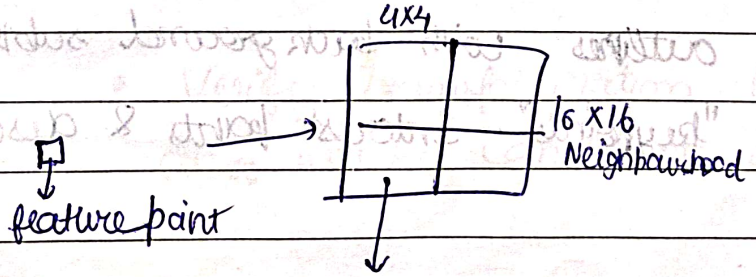
→ Keypoint Descriptors

Next is compute a descriptor for the local image region about each keypoint that is:

- Highly distinctive
- As invariant as possible

But first Normalization...

- Rotate to standard orientation
- Scale the window size



Histogram with 8 bins = 128 bin values

called as SIFT feature vector

Now, we know how to describe them but how about matching? (We still not done ☺)

Approach 1: n^2 Brute force

Approach 2: Nearest Neighbour
- K-D tree
- Best Bin First

Approach 3: Wavelet-Based Matching (Hashing)
↓
- Compute a short (3-vector) descriptor from the neighbourhood
- Quantize each value into 10 (overlapping) bins
  10^3 entries.

⇒ 3D object Recognition.

Train
1. Extract outlines with background subtraction
2. Compute "keypoints" - interest points & descriptors.

Test
1. Find possible matches
2. Search for consistent solution
   - such as affine

Date
Page

⇒ Feature - Based Alignment :

→ Overall strategy

- Compute features - detect } Harris
  describe } SIFT
- Find some useful matches - kd-tree
  BBF (Best Bin First)
- Compute & apply the best transformation - Affine
  Homography

Algorithm:

1. Extract features
2. Compute putative matches → "closest descriptor"
   Not known how to do
3. Loop until happy:
   - Hypothesize transformation T from some matches
   - Verify Transformation (search for other matches consistent with T) - mark best

Here step 2 isn't encountered yet, so tackling it now.

* We know exhaustive search, Hashing & Nearest Neighbor techniques gives the best match, how do we know it is a good one?

How do we know a good match is found?

Problem 1 → Feature Space Outlier Rejection:

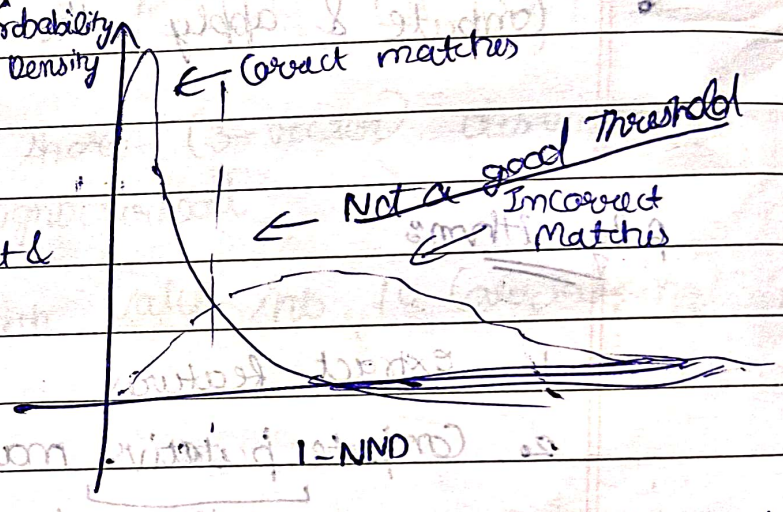
- Let's not match all features, but only those that have "similar enough" matches?
- How can we do it?

SSD (patch1, patch2) < threshold

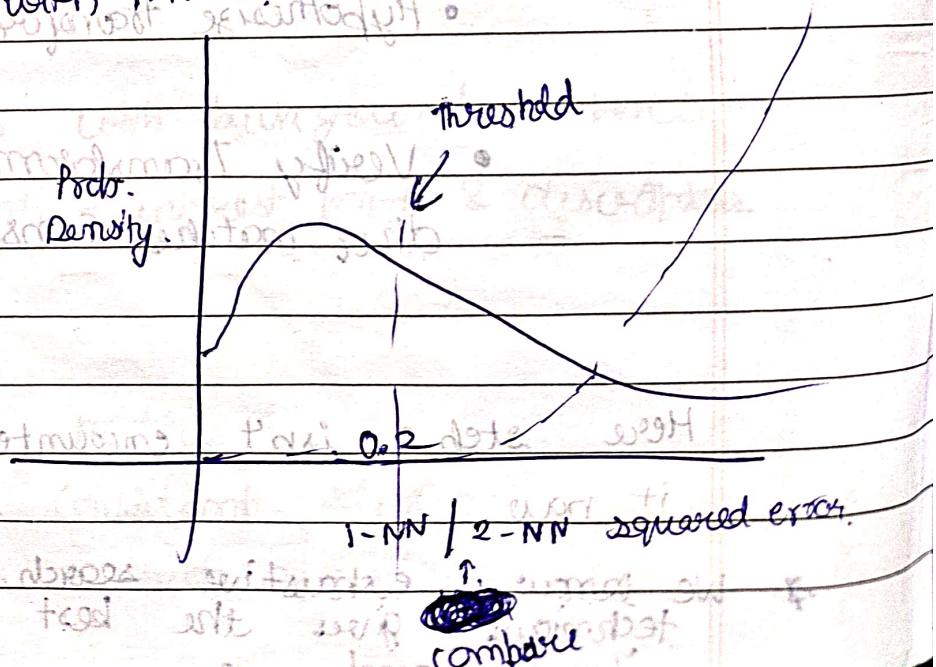
↓ also unknown

But, how to threshold it?

A point's Best matches are sometimes correct & sometimes not.



So, we use Lowe's method to use 2 NND and compare it with 1-NND.



Problem 2

Even when pick best match, still lots (& lots) of wrong matches - "outliers", because we did the distinctive property thing.

Essentially, we need to fit a model s.t. the effect of outliers is minimized.

⇒ RANSAC (Random Sample Consensus)

- Randomly pick some points to define your line (model). Repeat enough times until you find a good line (model) - or with many inliers.
- It somehow, copes with a large proportion of outliers.
- Every general model has a minimal set, ↑ min. points to calculate model
  - for line it is 2
  - for image transformations, 3 points
    - Translation: One pair of matched points
    - Homography (for plane): 4
    - Fundamental Matrix: 8 point pairs (in reality, 7)

Algorithm

1. Randomly selects s points (or point pairs) to form a sample.
2. Instantiate the model.
3. Get consensus set C_i - the points within error bounds (distance threshold) of the model.
4. If $|C_i| > T$, terminate & return model
5. Repeat for N trials, return model with max. C_i.

Parameters

1. s → Initial no. of points → known
2. Distance threshold t. → Un-known.
   - Choose t so prob. for inlier is high 95%.
   - 0 mean gaussian noise with std. dev. σ

$$t^2 = 3.84\,\sigma^2$$

3. Number of trials N.
   - Choose N so that, with probability p, at-least one random sample set is free from outliers (e.g. $p = 0.99$)
   - Need to set N based upon the outlier ratio ϵ

→ Calculating N

1. $s \rightarrow$ no. of points to compute solution
2. $p \rightarrow$ prob. of success.
3. $e \rightarrow$ proportion outliers, so % inliers $= (1-e)$
4. $P(\text{sample set with all inliers}) = (1-e)^3$
5. $P(\text{sample set will have at-least one outlier}) = (1 - (1-e)^3)$
6. $P(\text{all } N \text{ samples have outlier}) = (1-(1-e)^3)^N$
7. We want $P(\text{all } N \text{ samples have outlier}) < (1-p)$

$$\Rightarrow (1-(1-e)^3)^N < (1-p)$$

$$\Rightarrow \boxed{N > \frac{\log(1-p)}{\log(1-(1-e)^3)}}$$

⇒ Lastly, do re-compute least-squares H estimate on all of the inliers.

Adaptive algorithm

1. Set $N = \infty$, sample_cnt = 0, $e = 1.0$

2. while N > sample_cnt

Choose s & count the no of inliers

Set $e_0 = 1 - \frac{\text{no. of inliers}}{\text{No. of Points}}$

if $e_0 < e$, set $e = e_0$ & recompute N from e:

$$N = \frac{\log(1-p)}{\log(1-(1-e)^s)}$$

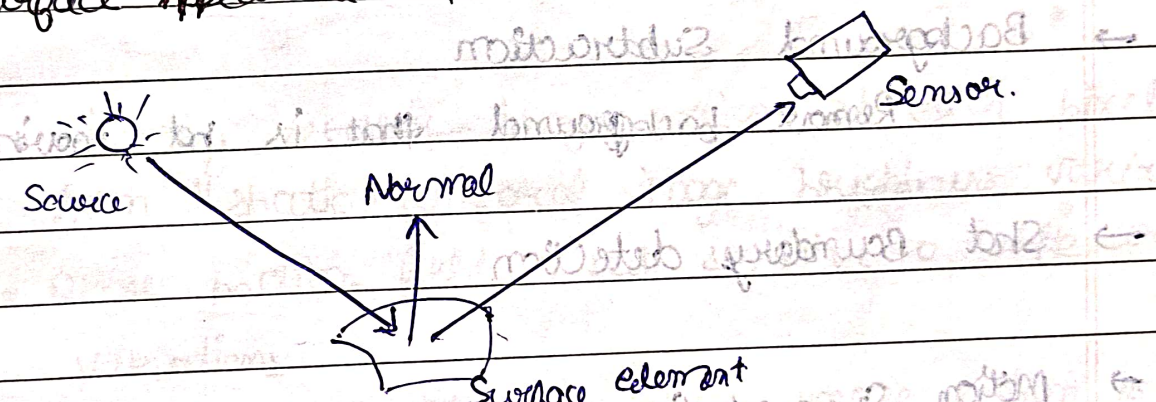
sample_cnt ++.

PHOTO METRY

factors affecting image:

1. Lights, Surfaces & Shadows
2. Reflections
3. Refractions
4. Inter-reflections
5. Scattering.

→ Surface Appearance



- Image Intensity = f (normal, surface reflectance, illumination)
- Surface reflection depends on both the viewing & illumination directions.

⇒ Introduction to Motion

- A video is a sequence of frames captured over time - usually quickly.

$I(x, y, t)$
(space, time)

- Motion Applications:

→ Background Subtraction
Remove background that is not moving.

→ Shot Boundary detection

→ Motion Segmentation
Segment the video into multiple coherently moving objects.

Goal: Manufacture information in motion like brain does.

→ Estimating 3D structure
→ Learning dynamic model
→ Recognizing events & activities
→ Improving video quality.

→ Motion estimation techniques:

I. Feature-based Methods (Done ✓)

- Extract visual features & track them over multiple frames.
- Sparse motion fields, but more robust tracking
- Suitable when image motion is large.

II. Direct, dense methods

- Directly recover image motion at each pixel from "spatio-temporal image brightness variation".
- Dense motion fields, but sensitive to appearance variations
- Suitable for video & when image motion is small.

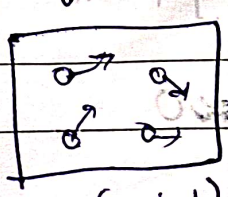
State of the art → Combination of both above.

→ Dense Flow : Brightness Constraint

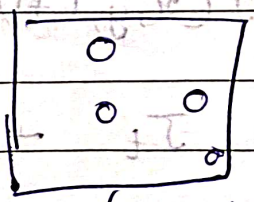
i) Optic Flow

It is the apparent motion of objects or surfaces.

Problem Defn:



I(x, y, t)



I (x, y, t+1)

Q. How to estimate pixel motion from image 1 to 2.

⇒ Solve pixel correspondence problem

Given a pixel in $I(x,y,t)$, look for nearby pixels of the same color in $I(x,y,t+1)$

This is the optic flow problem.

Assumptions

- color constancy $\left[I(x,y,t) = I(x+u, y+v, t+1) \right]$ — (1)

For grayscale, this is brightness constancy

- small motion. [u, v are less than 1 px, or smooth]

$$I(x+u, y+v) = I(x,y) + \frac{\partial I}{\partial x} u + \frac{\partial I}{\partial y} v + \cdots$$

↑ higher order

— (2)

Combining both eqn

$$I(x+u, y+v, t+1) - I(x,y,t) = 0$$

$$I(x,y,t+1) + I_x u + I_y v - I(x,y,t) \approx 0$$

(Assume t ~ small)

$$\Rightarrow \left[I(x,y,t+1) - I(x,y,t) \right] + I_x u + I_y v \approx 0$$

$$\Rightarrow \boxed{I_t + I_x u + I_y v \approx 0}$$

$0 \approx I_t + \nabla I \cdot \langle u, v \rangle$

If $u, v \to 0$

this becomes exact eqn.

$$I_t + \nabla I \cdot \langle u, v \rangle = 0$$

↓

Brightness constancy constraint eqn.

* 2 un-knowns, but 1 eqn.
(u, v)

similar to

$au + bv + c = 0$

I_x, I_y, I_t [(so optical flow is "constrained" to be on a line]

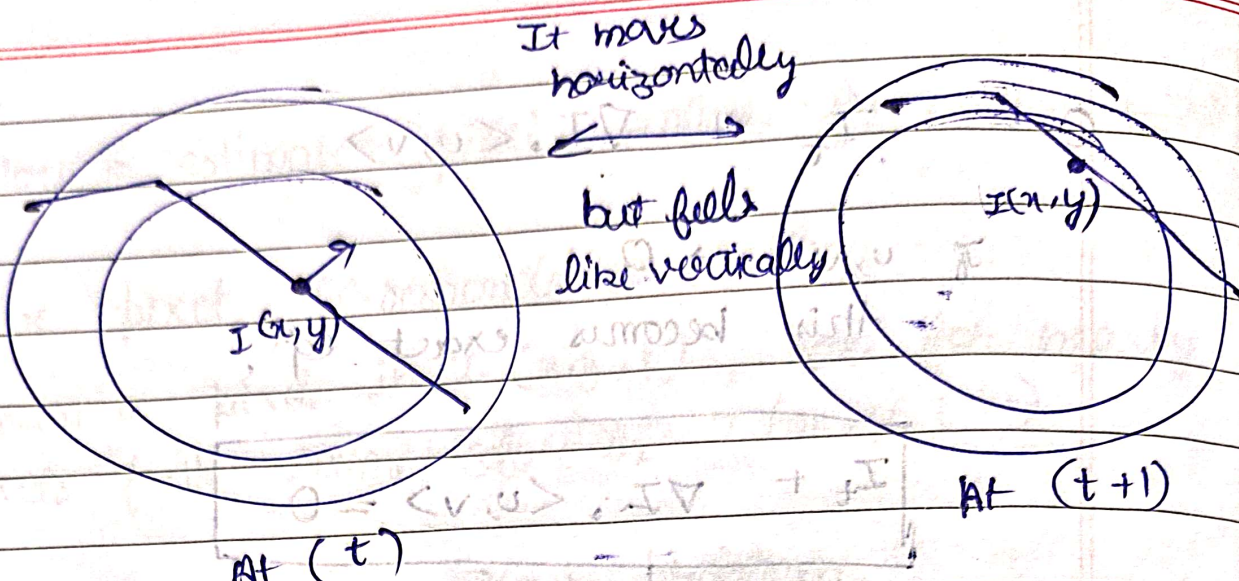
Intuitively, this constraint means that

- The component of the flow in gradient direction is determined (called Normal Flow). I_x, I_y

- The component of the flow parallel to an edge is unknown

↓

called aperture problem



So, optical flow can't be computed from just brightness constancy constraint, we need additional constraints such as:

Global
1. Smooth optical flow (motion shall be consistent in whole image.)

- Formulate error in optical flow constraint

$$e_c = \iint_{imag} (I_x u + I_y v + I_t)^2 \, dx \, dy$$

- Smoothness constraint

$$e_s = \iint_{imag} (u_x^2 + u_y^2) + (v_x^2 + v_y^2) \, dx \, dy$$

Find (u, v) at each img. point that minimizes:

$$e = e_s + \lambda e_c$$

Local

e.g.

1. The flow field is smooth locally

Method (Lucas-Kanade) Pretend the pixel's neighbors have the same (u, v)

$$A d = -b$$

for a 5×5 window

25 eqn, 2 unknown

do least squares & done

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ \vdots & \vdots \\ I_x(p_{25}) & I_y(p_{25}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = -\begin{bmatrix} I_t(p_1) \\ \vdots \\ I_t(p_{25}) \end{bmatrix}$$

$$A \quad d = -b$$

$$A^T A \quad d = -A^T b$$

$$\begin{bmatrix} \Sigma I_x I_x & \Sigma I_x I_y \\ \Sigma I_x I_y & \Sigma I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = -\begin{bmatrix} \Sigma I_x I_t \\ \Sigma I_y I_t \end{bmatrix}$$

- This is solvable when $A^T A$ is invertible

which means

λ_1 & λ_2 should not be too small

$\frac{\lambda_1}{\lambda_2}$ must not be too large where $\lambda_1 > \lambda_2$

→ Errors in Lucas-Kanade (B/C of assumptions)

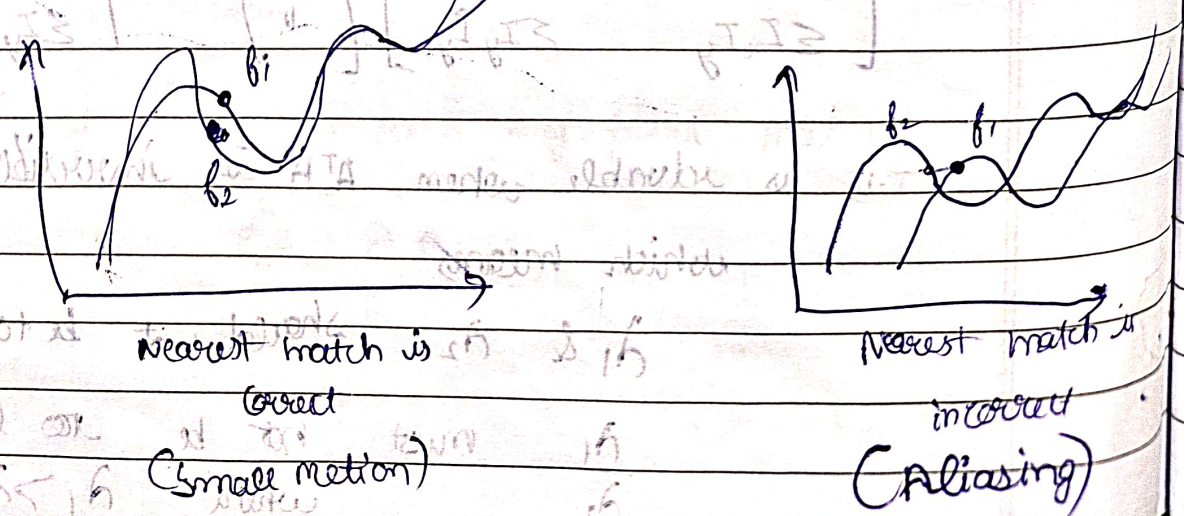
- A point doesn't move like its neighbors we made
  → Motion segmentation
- The motion is large (larger than a pixel) - Taylor doesn't hold.
  → Not-linear : Iterative Refinement
  → Local Minima : Coarse to fine estimate
- Brightness constancy doesn't hold
  → Exhaustive neighborhood search required.

i) Iterative Refinement

Algorithm

1. Estimate velocity at each pixel by solving Lucas-Kanade equations.
2. Warp I_t towards I_{t+1} using the estimated flow field.
3. Repeat until convergence.

II. Coarse to fine Refinement.



Q. How to eliminate this problem of large motion?

Reduce the resolution until it moves only in range of 1 px motion.

Coarse image ← fine image

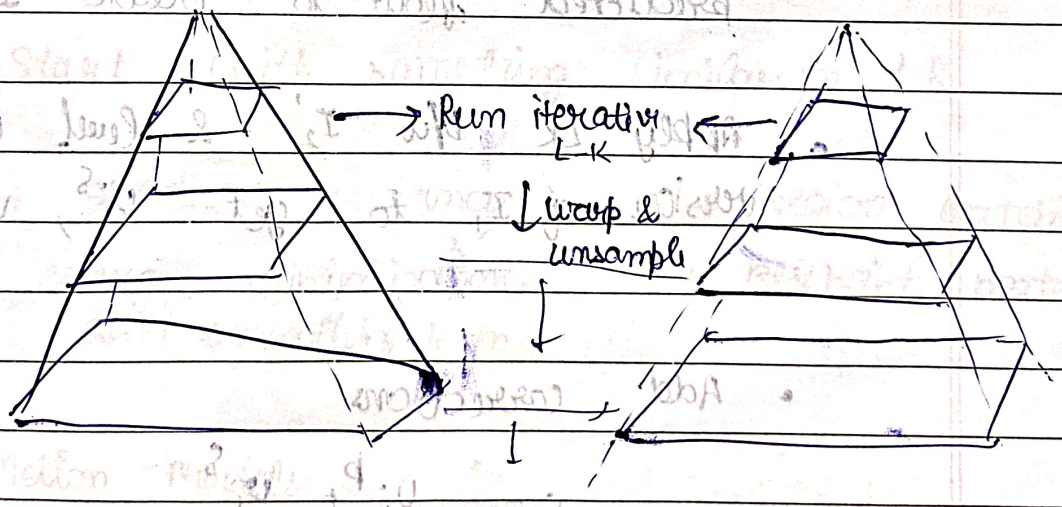
This is how it is done.

Hierarchical L-K

i) Create a gaussian pyramid

At top, it is a blury (coarse) picture

ii) Run iterative L-K on the top level images.



iii) Move downward, & repeat until you get the exact motion.

Before moving forward, let's take a detour to learn about aliasing & Image pyramids.

Formal Algorithm

1. Compute Iterative LK at level K
2. Init. $u_{K+1}, v_{K+1} = 0$ at size of level K+1
3. for each level i = K to 0
   - Upsample (Expand) u_{i+1}, v_{i+1} to create u_i^p, v_i^p flow field of one twice resolution as level i+1
   - Multiply u_i^p, v_i^p by 2
   - Warp level i Gaussian version of I_2 w/ to predicted flow to create I_2'
   - Apply LK b/w I_2' & level i Gaussian version of I_1 to get u_i^s, v_i^s (correction in flow)
   - Add corrections

$$u_i = u_i^p + u_i^s$$
$$v_i = v_i^p + v_i^s$$

⇒ Sparse LK:

- The lucas Kanade alg. described gives a dense field, (u, v) everywhere.
- But we said that we only want to solve LK where eigen-values are well behaved.

Sparse LK ⇒ hierarchical LK applied to good feature locations.

⇒ Today's State of the Art Methods:

Start with something similar to LK

+ gradient constancy
+ energy minimization with smoothing term
+ region matching
+ keypoint matching

⇒ Motion Models

There are 4 type of motion in real world:

1. Translation — 2 Un-known
2. ~~Linear~~ (Similarity) — 4
3. Affine — 6
4. Homography / Perspective — 8

Date
Page

Crewical Motion

Now, in real world

$$V = \omega \times R + T$$

$$\begin{bmatrix} V_x \\ V_y \\ V_z \end{bmatrix} = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + \begin{bmatrix} V_{Tx} \\ V_{Ty} \\ V_{Tz} \end{bmatrix}$$

In image, velocity can be defined from

$$x = f\frac{X}{Z}$$

$$y = f\frac{Y}{Z}$$

$$\frac{dx}{dt} = u = v_x = f\frac{V_x}{Z} - x\frac{V_z}{Z}$$

$$\frac{dy}{dt} = v = v_y = f\frac{V_y}{Z} - y\frac{V_z}{Z}$$

$$\begin{bmatrix} u(x,y) \\ v(x,y) \end{bmatrix} = \frac{1}{Z(x,y)} A(x,y) \cdot T + B(x,y) \cdot \Omega$$

↓ Trans. ↓ Rotation.

$$A(x,y) = \begin{bmatrix} -f & 0 & x \\ 0 & -f & y \end{bmatrix}$$

$$B(x,y) = \begin{bmatrix} xy/f & -(f+x^2)/f & y \\ (f^2+y^2)/f & -xy/f & -x \end{bmatrix}$$

⇒ Introduction to Tracking

- Recognizing same object through frames in different timeline is called tracking.

Some great tracking ex.

i) Surveillance camera

ii) Camera Tracking under occlusions

→ Tracking Challenges:

One idea to do tracking is optic flow b/w first & second, second & third & so so. but..

i) It's hard to compute optical flow.

ii) There can be large displacements since could be moving rapidly
↓
Probably dynamics need to taken into account

iii) errors would drift

iv) Occlusions

Date
Page

- Shi - Tomasi Feature Tracker

Algo

1. From frame to frame, track with LK and a pure translational model
   - Robust for small disp.

2. Check consistency of tracks by affine registration to the first (or earlier) observed instance of the feature.
   - Affine model is more accurate for larger displacement.
   - Comparing to the first or early frame helps to minimize drift.

* This above method will fail for large disp.

- Tracking with dynamics

Key idea: Given a model of expected motion, predict where objects will occur in next frame, even before seeing the image.
   - Restrict search for the object
   - Improved estimates since measurement noise is reduced by trajectory smoothness.

Assumptions

- object do not disappear & reappear in diff. places in the scene
- Camera is not moving instantaniously to new viewpoint
- Gradual change in pose between camera and scene.

⇒ Tracking as inference:

Hidden state (x): True parameters we care about

Measurement (Y): Noisy observation of underlying state.

At each time step t, state changes (from X_{t-1} to X_t) and we get a new observation Y_t.

Our goal: Estimate distribution of state X_t given

→ all observations seen so far

→ Knowledg about dynamics of state transitions.

→ Steps of Tracking

i) Prediction: What is the next state of the object given past measurements?

$P(X_t \mid Y_0 = y_0, \ldots, Y_{t-1} = y_{t-1})$

ii) Correction: Compute an updated estimate of the state from prediction & measurements

$$P(X_t \mid Y_0 = y_0, \ldots, Y_{t-1} = y_{t-1}, Y_t = y_t)$$

($Y_t = y_t$: New measurement)

Tracking → The process of propagating this posterior distribution of state given measurements across time.

Some Assumptions

i) Only the immediate past matters

$$P(X_t \mid X_0, \ldots, X_{t-1}) = P(X_t \mid X_{t-1})$$

($P(X_t \mid X_{t-1})$: Dynamics model)

ii) Measurements depend only on the current state

$$P(Y_t \mid X_0, Y_0, \ldots X_{t-1}, Y_{t-1}, X_t) = P(Y_t \mid X_t)$$

($P(Y_t \mid X_t)$: Observation model)

→ Tracking as Induction

Base Case:
- Assume we have some initial prior that predicts state in the absence of any evidence: $P(X_0)$
- At the first frame, correct this, given value of $Y_0 = y_0$

Prediction Step

Given: $P(X_{t-1} \mid y_0, \ldots, y_{t-1})$

Guess: $P(X_t \mid y_0, \ldots, y_{t-1})$

$$= \int P(X_t, X_{t-1} \mid y_0, \ldots, y_{t-1})\, dX_{t-1}$$

↳ Law of Total Probability - Marginalization.

$$= \int P(X_t \mid X_{t-1}, y_0, \ldots, y_{t-1}) P(X_{t-1} \mid y_0, \ldots, y_{t-1})\, dX_{t-1}$$

Using assumption

$$= \int \underbrace{P(X_t \mid X_{t-1})}_{\text{dynamics model}} \underbrace{P(X_{t-1} \mid y_0, \ldots, y_{t-1})}_{\text{Corrected estimate from previous step}} dX_{t-1}$$

Correction Step

Given: Predicted value $P(X_t \mid y_0, \ldots, y_{t-1})$ & y_t

Compute $P(X_t \mid y_0, \ldots, y_t)$

$$= \frac{P(y_t \mid X_t, y_0, \ldots, y_{t-1}) \overbrace{P(X_t \mid y_0, \ldots, y_{t-1})}^{\text{predicted estimate}}}{P(y_t \mid y_0, \ldots, y_{t-1})}$$

(Using Bayes Rule)

$P(y_t \mid X_t)$ ← Observed Model

(Using assumption)

→ Linear Dynamics model

Dynamics Model: State undergoes linear transformation plus Gaussian Noise

$$X_t \sim N(D_t X_{t-1}, \Sigma_{d_t})$$

- X_t: Normal Distribution
- $D_t X_{t-1}$: Mean
- Σ_{d_t}: Gaussian Noise / Co-variance

→ Linear Measurement model

Observation Model: Measurement is linearly transformed state plus Gaussian Noise.

$$y_t \sim N(M_t X_t, \Sigma_{m_t})$$

⇒ Kalman Filter — Gives a tradeoff b/w uncertainty of prediction & measurement.
It is a method of tracking linear dynamic systems with Gaussian Noise.

Kalman Prediction

Mean at time t = dynamics multiplier d
&
Corrected mean at time (t-

variance at t = d^2 & Corrected variance at (t-
+ additional nois

Nomenclature

μ_t^- , σ_t^- → Mean & variance before measurement

μ_t^+ , σ_t^+ → vice versa

Kalman Correction (For 1-D)

$$\mu_t^+ = \frac{\mu_t^- \sigma_m^2 + m y (\sigma_t^-)^2}{\sigma_m^2 + m^2 (\sigma_t^-)^2}$$

$$(\sigma_t^+)^2 = \frac{\sigma_m^2 (\sigma_t^-)^2}{\sigma_m^2 + m^2 (\sigma_t^-)^2}$$

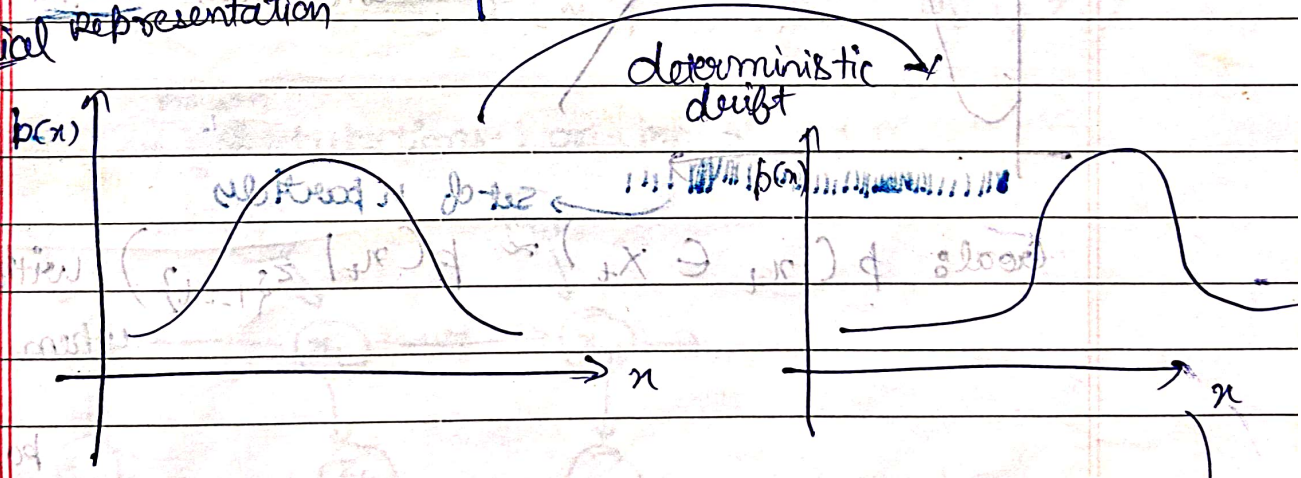
$$\mu_t^+ = \mu_t^- + K\left(\frac{y_i}{m} - \mu_t^-\right)$$

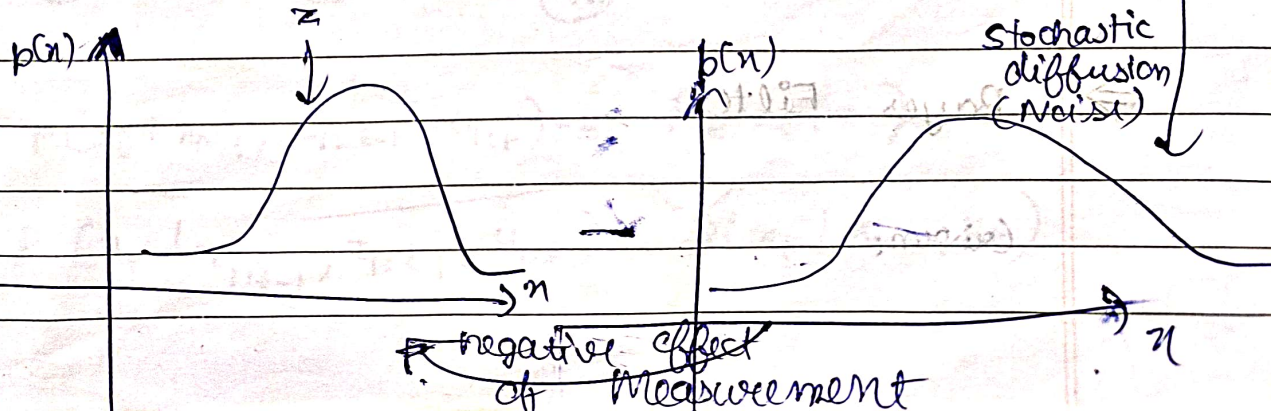
Residual

Kalman Gain

* Also, variance lowers

Pictorial representation





negative effect of Measurement

Pros

- Simple updates, compact & efficient

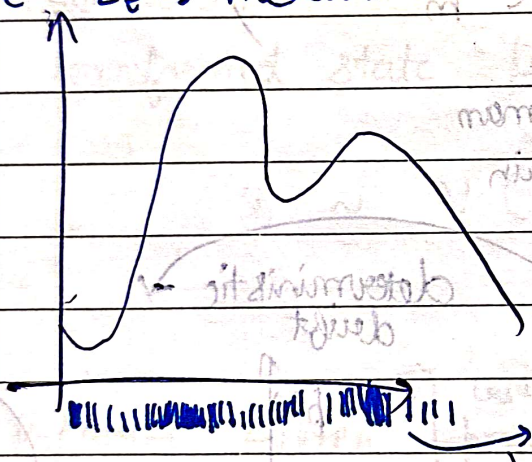
Cons

- Unimodal distribution, only single hypothesis
- Restricted class of motions defined by linear models
  - Extensions called "extended KL filter"

What about Non-Gaussian?

↳ Particle Filters [.. YAY ☺]

Here z_t → measurements.



Density is represented by both where the particles are and their weight.

→ set of n particles

Goal: $p(x_t \in X_t) \approx p(x_t | z_{1:t})$ with equality when $n \to \infty$

t particles

⇒ Bayes Filter

Given:

1. Prior probability of the system state $p(x)$

2. Action (dynamical system) model:
$$p(x_t \mid u_{t-1}, x_{t-1})$$

3. Sensor Model (likelihood) $p(z \mid x)$

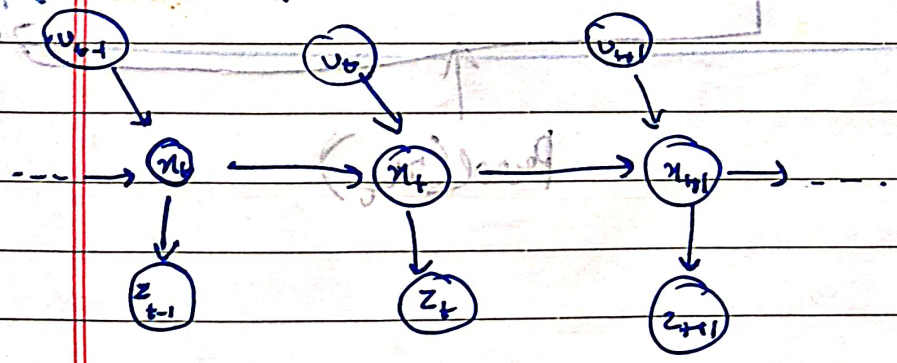
4. Stream of observations z and action data u:
$$data_t = \{u_1, z_2, \ldots, u_{t-1}, z_t\}$$

Wanted:

1. Estimate of the state X at time t.

2. Posterior of the state is also called belief:
$$Bel(x_t) = p(x_t \mid u_1, z_2, \ldots, u_{t-1}, z_t)$$

$u_{1:t}$ = known perturbations or inputs



$$p(z_t \mid x_{0:t}, z_{1:t-1}, u_{1:t}) = p(z_t \mid x_t)$$

$$p(x_t \mid x_{1:t-1}, z_{1:t-1}, u_{1:t}) = p(x_t \mid x_{t-1}, u_t)$$

→ Bayes Filter

$$Bel(x_t) = P(x_t \mid u_{1:t-1}, z_{2:t})$$

$$= \eta P(z_t \mid x_t, u_1, z_2, \ldots, u_{t-1}) P(x_t \mid u_1, z_2, \ldots, u_{t-1})$$

(Bayes)

$$= \eta P(z_t \mid x_t) P(x_t \mid u_1, z_2, \ldots, u_{t-1})$$

↑ Sensor Ind.

Total Prob. of "prior" $= \eta P(z_t \mid x_t) \int P(x_t \mid u_1, z_2, \ldots, u_{t-1}, x_{t-1})$

$$P(x_{t-1} \mid u_1, z_2, \ldots, u_{t-1}) \, dx_{t-1}$$

↑ Markov Ind.

$$= \eta P(z_t \mid x_t) \int P(x_t \mid u_{t-1}, x_{t-1}) \, Bel(x_{t-1}) \, dx_{t-1}$$

$$\boxed{Bel(x_t) = \eta P(z_t \mid x_t) \int P(x_t \mid u_{t-1}, x_{t-1}) \, Bel(x_{t-1}) \, dx_{t-1}}$$