DIP

DIP : Coursera
(Northwestern & Duke)

- szeliski.org/Book

(cse478@lists.iiit.ac.in)
(Monday : 3:30 – 5:00)

- Project!
  - ↳ Paper → DIP

- Matrix → Bullet Scene → Mesh of cameras → To control space time
  (How was it shot?)
- Who was the artist of the painting (Slide)

Types of Images

- EM Spectrum
  - PET Scans, Bone Scans } Gamma ray imaging
  - Radar } Microwaves
  - MRI } Radiowaves
- Ultrasound
- Computer Generated Imagery

Note: Multispectral images, Stereo Images, Multiview images

* Note: Single Image Haze Removal → Paper
  Black/White to Color → Paper

- Cinematic grading
- Feature Detection & Stitching / SIFT Features
- Segmentation → Medical Imaging
- Compression of Images.
- Inpainting, Special effects
- Biometrics.
- Gesture recognition.

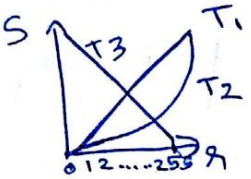
———— × ————

Intensity Transformations:

- Point Operations (Do not consider neighbouring pixels)

⇒ Input pixel value (r) $\xrightarrow{T}$ output pixel value(s)

ex:



- T_2 will increase contrast b/w dark & bright
- T_3 = negative transform (Inversion kind of)

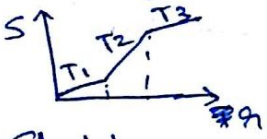
* • Log transforms help if few intensities with large difference are present. (Scale of visualization changes) After transform we use max & min to divide into new 255 intervals.

⇒ Gamma transforms ⟶ $S = cr^{\gamma}$

$\gamma \rightarrow 0.04$ to 25 or more

Piecewise Transformation:

- For a certain range use T_1, then T_2 and so on.



↑ Transform functions

→ Contrast Stretching

* → Bit Plane Slicing : (Examples in slides)

- For every pixel select bit i, if its 1, put 255 in image else put 0 in image.

⇒ So for the 8 bits, we get 8 bit planes

- This can be used for compression or we can store data on 1st bit plane, and add this to image − actual 1st bit plane (least significant) and send it.

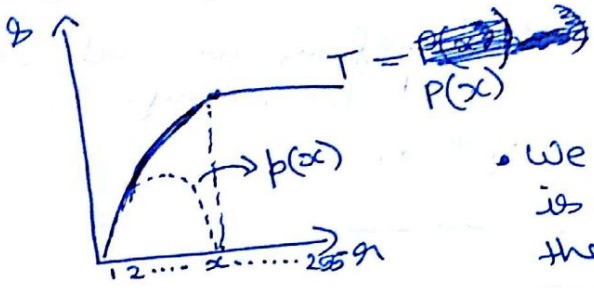
---

*Histograms:

- Probability Density $\Rightarrow p(x)$
- Cumulative distribution function $= \int_{-\infty}^{a} p(x)\,dx = P(a)$

1) * $\Rightarrow$ Histogram Equalization: (ex: in slides)

* • If the peaks of histogram are not spaced out evenly throughout, then we spread it across the entire histogram.
  • We do this by using the cumulative distribution function as the transform function.



- We can see that if there is a peak at $x=20$, and then lots of zeros and another small value at $x=100$.
  $\Rightarrow$ They get mapped close to each other after transform.

Note:
$\Rightarrow p_s(s)\,ds = p_r(r)\,dr$ } The area under prob density curve remains same after transform. (If the T is monotonic)
$\longrightarrow$ (1)

- $S = T(r) = (L-1)\int_0^r p_r(w)\,dw$ (Since CDF will be in range of 0 to 1, we multiply with L-1 $\Rightarrow$ (Generally 255))
  $\longrightarrow$ (2)

- From (1),
  $p_s(s) = p_r(r)\cdot\frac{dr}{ds} = p_r(r)\cdot\frac{1}{(L-1)\cdot p_r(r)} = \frac{1}{L-1}$

$\therefore p_s(s) = \frac{1}{L-1}$, So the histogram is distributed evenly (Assuming continuous scale)

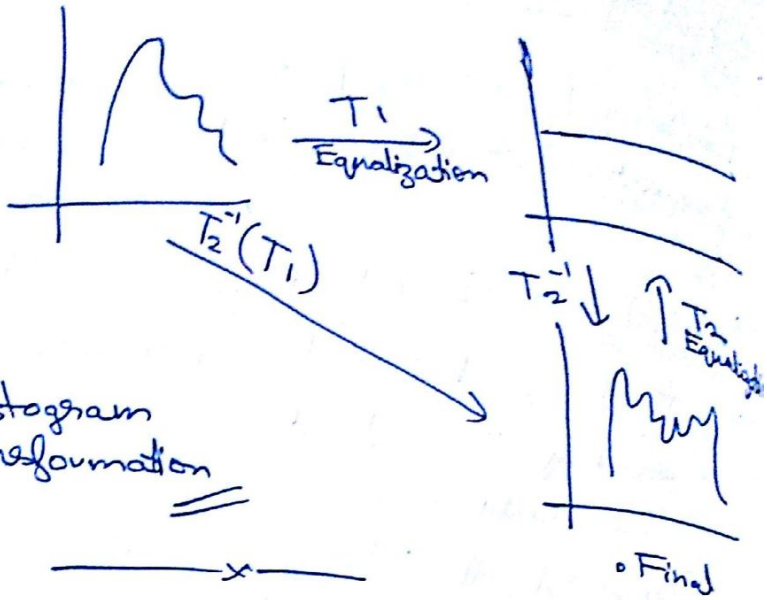
$\therefore$ This is an ideal transformation function.

---

*Note: From histogram A → histogram B

*

- First



⇒ This is histogram transformation

- Final

3

- On taking the inverse transform, we will sometimes end up mapping one pixel to many. So we need to take care of this.

---

*Spatial Filtering:

- Either 4 neighbours or 8 neighbours.

1) Paintfill: (Bucket Fill Algo)

```
pf(x, y, cur_c, rep_c):
    if (I(x,y) == cur_c):
        pf(x+1, y, cur_c, rep_c)
        pf(x-1, y, cur_c, rep_c)
        pf(x, y+1, cur_c, rep_c)
        pf(x, y-1, cur_c, rep_c)
    else:
        return
```

- If cur_c == rep_c, we directly return.
- If x,y is outside the image, we return.

2) Averaging Filter → Blur,

$$\Rightarrow \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \Rightarrow \frac{1}{n^2}\begin{bmatrix} 1 & \cdots & 1 \\ \vdots & & \vdots \\ 1 & \cdots & 1 \end{bmatrix}$$ (n times)

---

*3) Gaussian Filter: → Blur

$$G(x,y) = e^{-\frac{(x-u)^2}{2\sigma_x^2} - \frac{(y-v)^2}{2\sigma_y^2}}$$

- No need to normalize since the image automatically gets normalized.

$$\Rightarrow G(x,y) = e^{-\frac{(x^2+y^2)}{2\sigma^2}}, \text{ If } (u,v) = (0,0)$$

ex: $\frac{1}{x}\begin{bmatrix} e^{-1} & e^{-1/2} & e^{-1} \\ e^{-1/2} & 1 & e^{-1/2} \\ e^{-1} & e^{-1/2} & e^{-1} \end{bmatrix}$, If $\sigma = 1$

- x = Sum of $e^{-1} + e^{-1/2} + e^{-1} + e^{-1/2} + 1 + e^{-1/2} + e^{-1} + e^{-1/2} + e^{-1}$

Note: Smoothing linear filters,

- To remove graing noise, blur with gaussian then do treshdding.

---

*4) Differentiating Filter: → [-1 1]

- 1st order derivative, $\frac{\partial f}{\partial x} = F(x+1) - F(x)$
- 2nd order derivative, $\frac{\partial^2 f}{\partial x^2} = F(x+2) - F(x+1) - F(x+1) + F(x)$

$= F(x+2) - 2F(x+1) + F(x)$

→ $F(x+1) - 2F(x) + F(x-1)$

↓ [1 -2 1]

- In 2D, we do both x & y derivatives.

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

* → $\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ ⇒ Laplacian Filter

* → $\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ A variation of laplacian

—— x ——

Note: We can sharpen an image by taking
* the laplacian & adding it to the image.

—— x ——

*5) Unsharp Masking — (Highboost Filtering)
* (Sharpening) (Example in slides)

- Original Image = F(x,y)
- Gaussian = G(x,y)

⇒ h(x,y) = F(x,y) − G(x,y) } If edges etc, this has a value, else its 0.

- We now do F'(x,y) = F(x,y) + k · h(x,y)
- So we boost the sharpness of the image

—— x ——

Note: Robert & Sobel detectors (edges etc)

—— x ——

6) Max & Min Filters → for pepper & salt noise.

Median Filter → for both salt & pepper noise.

—— x ——

Note: There are other averaging filters as well.

Bilateral Filtering: (Gaussian without losing details)

- $I^{filtered}(x) = \frac{1}{W_p} \sum_{x_i \in \Omega} I(x_i)\, g_s(\|x - x_i\|)$

$$W_p = \sum g_s(\|x - x_i\|)$$

$$g_s = e^{-\frac{(x-x_i)^2}{2\sigma_s^2}}$$

- Use a neighbourhood of pixels along with the gaussian.

⇒ If we want to give more weight to pixels with closer intensity,

$$I^{filtered}_{x \to (i,j)}(x) = \frac{1}{W_p} \sum_{x_i \in \Omega} I(x_i)\, g_s(\|x - x_i\|) \cdot g_r(\|I(x) - I(x_i)\|)$$

$$g_s = e^{-\frac{((i-k)^2 + (j-l)^2)}{2\sigma_s^2}}$$

$$g_r = e^{-\frac{(I(i,j) - I(k,l))^2}{2\sigma_r^2}}$$

- When we do this, we ensure that edges and other details don't get blurred whereas normal patches with small changes do.

ex: $a = \begin{bmatrix} 100, 230, 80 \\ 80, (240), 90 \\ 87, 235, 79 \end{bmatrix}$ ← Image1

$b = \begin{bmatrix} 100, 90, 80 \\ 80, (85), 90 \\ 87, 93, 79 \end{bmatrix}$ ← Image2

$g_r^a = \begin{bmatrix} 0 & 0.67 & 0 \\ 0 & 1 & 0 \\ 0 & 0.9 & 0 \end{bmatrix}$ at $I(i,j) = 240$

$g_r^b = \begin{bmatrix} 0.41 & 0.9 & 0.9 \\ 0.9 & 1 & 0.9 \\ 0.78 & 0.77 & 0.86 \end{bmatrix}$ at $I(i,j) = 85$

- So in Image1, we only use values along the edge for blurring, so we don't lose detail.

---

* Human Vision & Color perception:

- Human eye → Cornea, Retina
  → Fovea → Responsible for most of our sight → Almost all cones here.
- Accommodation of the eye, how it adjusts to focus on far or near objects.
- * Retina → Rods (Majority of cells) → At low luminance → Other vision aid
  → Cones → R, G, B → Visible light
  → LMS Cones (Long, Medium, Short)
- Blind Spot → Where the nerves go in.
- Scotopic, Mesopic, Photopic vision.
  Scotopic → Rods
  Mesopic → Rods + Cones
  Photopic → Cones

* ⇒ Weber Law: Intensity changes, we are more sensitive to them at high intensities.

⇒ Luminous efficiency → Describes average spectral sensitivity of our eye based on luminance.
  ↳ Purkinje shift

Perception (Humans)

* • Opponent theory: Red-Green, Yellow-Blue, Black-White } We perceive all colors using this.
  - → R+G+B = Black/White
  - → R−G = Red/Green
  - → R+G−B = Yellow/Blue
- Trichromatic theory ⇒ RGB, this can't explain afterimages, so we came up with opponent theory.

5

Note: Image Morphing

Geometric Operations:

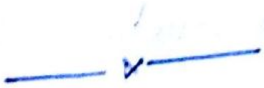
1) Translation: $x' = x + dx$
$y' = y + dy$

2) Scale: $x' \rightarrow x \cdot S_x$
$y' \rightarrow y \cdot S_y$

3) Shear: $x' \rightarrow x + ay$
$y' \rightarrow y + bx$

4) Rotation: $\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix}$

5) Flipping



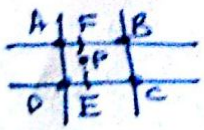
* Image Warping: (Issue)

- If we try to find (x', y') for each (x, y) some (x, y) could map to same (x', y') or x', y' may be floating points. This is a problem.

⇒ So for each (x', y') use inverse transform
* to get (x, y), $dest(x', y') = source(x, y)$
↓
resample − src(x, y)

$\Rightarrow x = f_x^{-1}(x', y')$
$y = f_y^{-1}(x', y')$

- If x, y are floats, we can perform linear interpolation to get the result.



$P = (x, y)$

- We interpolate A, B for F, D, C for E & F, E for P.
- Now we have the color value for (x, y) and we just assign this interpolated color to (x', y').

Homogeneous coords : (~~Affine transforms~~)

- $\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$

⇒ Example affine transforms in slides.

---

* Affine Transforms:

- Preserves collinearity (all points on a line, remain on a line)
  - ↳ Preserve ratios of distances between points on a line.
- Translation, Scaling, ~~etc~~ rotation etc.

—— x ——

- Note: 2D projective transform (Homography)

Note: Image stitching.
Note: Non linear image warps

—— x ——

6) Ripple:

$$x = x' + a_x \sin\left(\frac{2\pi \cdot y'}{\tau_x}\right)$$

$$y = y' + a_y \sin\left(\frac{2\pi \cdot x'}{\tau_y}\right)$$

7) Twist:

$\beta = \tan^{-1}(dy/dx) + \alpha\left(\frac{r_{max}-r}{r_{max}}\right)$

$$x = \begin{cases} x_c + r\cos\beta, & r \le r_{max} \\ x' & r > r_{max} \end{cases}$$

$dx = x' - x_c$
$dy = y' - y_c$
$r = \sqrt{dx^2 + dy^2}$

$$y = \begin{cases} y_c + r\sin\beta, & r \le r_{max} \\ y' & r > r_{max} \end{cases}$$

8) Spherical:

- Similar formulas
- Examples in slides.

Note: Warp grids, no parameters, but content aware warps

—— x ——

6

Image Resampling:

I) Down sampling:

- To make an image half the size of the original image, we can't just drop rows & columns since the frequency might be lower than nyqist rate so our image would be bad. (Aliasing)

⇒ A solution for this would be to apply a * gaussian filter and then do the down sampling. This would reduce the high frequency components so our sampling frequency would be decent enough. So we would not get aliasing.

—— x ——

*II) Gaussian Pyramid:

- $I_1 \rightarrow I_2 \rightarrow I_3 \rightarrow \dots$ } Gaussian pyramid

($\downarrow$ = Blur, $\nearrow$ = Sub sample)

$I_1 * \quad I_2 * \quad \vdots$

⇒ Laplacian Pyramid:

*
- In the gaussian pyramid take G_{n+1} → Expand 2x, (Upsample) then $L_i = G_i - G_{i+1}$ Exp

- The last layer $L_n = G_n$.
- This is similar to the output we get from the Laplacian filter on the image.

- The laplacian pyramid can be used to reconstruct any layer of the original image completely since we are storing the change between layers basically.

———x———

Note: Gaussian Pyramid is useful for multiscale
** detection. Given image & search pattern, ~~downsample them~~ & create the gaussian pyramid, start from the topmost (smallest) layer & get approx location. Now go to lower layer (larger) and search only the area which which seemed similar in the previous layer.

———x———

Note: Lasso Blending

*Gaussian Pyramid Applications (And Laplacian Pyramids) (Slides for examples)

1) Multiscale detection.

2) Blending Images.

* ↳ Generate laplacians for source & destination image.
  ↳ At each layer apply a mask, make the mask more blurred at higher levels of pyramid and less blurred at lower levels.
  ↳ Then generate laplacian pyramid for the resulting image & generate the resulting image back.

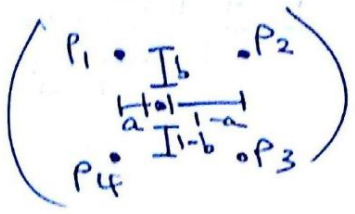
———x———

III) Upsampling:

1) • Nearest neighbour interpolation can be done, just add rows & columns inbetween & then copy the nearest neighbour there.

2) Bilinear interpolation: (2D)

• Consider a large blank image, and fill values considering as multiplied image creates of this size.



• We can find any value inbetween.
• 2 linear interpolations together seem like quadratic.

3) Gaussian | Sinc | Linear interpolation: (Example in slides)

• Select one of these filters ∧, ∩, etc then just convolve it with the original.

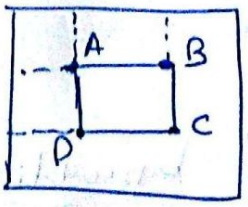
⇒ For images we use a ~~3D filter~~ 3D looking filter

———x———

7
Color Image Processing:

• 2D cumulative image sum → Integral image.

$$I(x,y) = i(x,y) + I(x-1,y) + I(x,y-1) - I(x-1,y-1)$$



• Cumulative sum ABCD

⇓

$$I(C) - I(D) - I(B) + I(A)$$

• From cumulative sum we can get mean.

⇒ we can do something similar for variance,

$$\text{Variance } \sigma^2 = \frac{1}{n}\sum (i(x,y) - \mu)^2$$

$$= \frac{1}{n}\sum \left(i(x,y)^2 + \mu^2 - 2\mu i(x,y)\right)$$

- Similar to above but maintain sum of squares as well.

$\Rightarrow$ Variance $= \frac{1}{N}\left(S_2 - \frac{S_1^2}{N}\right)$, S_2 = Cumulative squared sum

S_1 = Cumulative sum.

(Of a region ABCD)

Note: Polya Gamma (Lecture on Math)

---

*Chromatic Adaptation:

- Small changes in lighting conditions etc, our brain automatically adjusts and we can still recognize objects etc.

Additive & Subtractive colors:

- Additive → RGB (Digital)
- Subtractive → MCY (Painters, Printers etc)

*Color Matching Functions:

- Using RGB.

$\Rightarrow$ Give input color which needs to be matched with primaries

$\Rightarrow$ Not all colours can be matched by adding RGB so we add R to the input & then it can be matched.

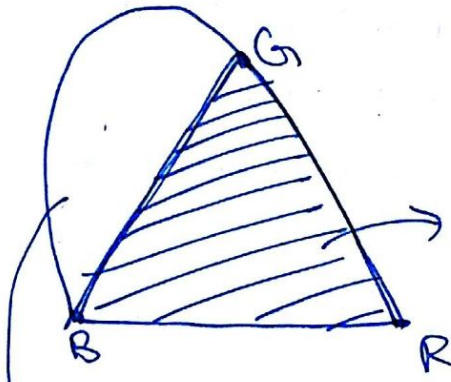
XYZ-Color Space

- Because of the issue where we need to subtract red

we apply a transformation to move to XYZ space

- Here we can generate any colours.
- We also have CIE LAB color space, HSI, HSV etc.

CIE Chromaticity Diagram:



- The entire spectrum of colors
- The spectrum we can generate

- We cannot create these colors from just RGB, and we require adding R to that range to get it within the Δle.

⇒ McAdam Ellipses → regions in the chromaticity diagram which contains colors indistinguishable to the human eye.

—x—

8

LMS Color Space: (Closest to human vision)

- From RGB → LMS using a transform.

⇒ Chromatic Adaptation based white balancing can be done.

—x—

* White Balancing:

- Choose a point which we know is actually white. Then see how the rgb at that point need to be scaled to get 255, 255, 255. Apply this to the entire image.

⇒ $\frac{255}{r}, \frac{255}{g}, \frac{255}{b}$ ⇒ Apply to every pixel p.

*Von Kries Method: (White Balancing in LMS)

- Naive white balancing might not always work

⇒ Convert to LMS, then get the scales in a similar method as before. Then apply this to the image and convert back to RGB image.

- This works because LMS is similar to our perception of light.

---

*Pseudo color image processing: (Give a color)

1) Depth channels are hard to visualize so we add color to it making it like red → green → blue near → far

- Example in slides.

2) Intensity Slicing:

- If intensity at a point > threshold, give it yellow otherwise blue.
- Helps identifying what we require.
  ex: pipe leaks.

3) Transformations:

- For each R, G, B channels use some transformation like a $|\sin x|$ etc to generate colour value.
- Useful for scans in airport luggage checks etc where color helps easily visualize xray images.

4) Multi Spectral:

- If you have an infrared channel available etc, replace it with one channel, say r & g, b → From gray scale

Note: Chroma Keying → Greenscreen to video.

Note: Alpha matting

---

* Fourier Transforms:

1) Vector space: $S = \{v_i\}_{i=1:N}$

$v_i + v_j \in S$

$\alpha v_i \in S$

Inverse additive and a few other properties.

2) Span: A set S spans vector space V if any $v_i \in V$

$v_i = \alpha_1 v^1 + \alpha_2 v^2 + \dots + \alpha_n v^n$

where $v^i \in$ Span

(Linear combination)

3) Linear independence:

If, $\alpha_1 v_1 + \alpha_2 v_2 + \dots + \alpha_n v_n = 0$

then $\alpha_k = 0 \quad \forall k$

⇒ A set of vectors are linearly independent if

4) Inner product:

$$\langle v_1, v_2 \rangle = \sum_{i=0}^{d} v_1[i] \cdot \overline{v_2[i]}$$

- Inner product of 2 vectors.

5) Orthogonal:

- Two vectors are orthogonal if inner product of them is 0.

6) Basis:

- A set of linearly independent vectors S which span the given vector space V is called the basis.

Note: If vectors are orthogonal ⟶ they are linearly independent.
(Inverse need not be true)

---

7) Theorem of orthogonal decomposition:

- $V = \sum_{k=0}^{U-1} \alpha_k V_k$, $V_k \in$ Orthogonal Basis.

$$\Rightarrow \alpha_k = \frac{\langle V, V_k \rangle}{\langle V_k, V_k \rangle}$$

Proof:

$$\langle V, V_m \rangle = \sum_{k=0}^{N-1} \alpha_k \langle V_k, V_m \rangle = \alpha_m \langle V_m, V_m \rangle$$

$$\therefore \frac{\langle V, V_m \rangle}{\langle V_m, V_m \rangle} = \alpha_m.$$

8) Complex Exponentials

- $e^{j\Phi} = \cos\Phi + j\sin\Phi$

$$E_{N,K} \text{ (Vector)} = \begin{bmatrix} e^{j \cdot \frac{2\pi K \cdot 0}{N}} \\ e^{j \cdot \frac{2\pi K \cdot 1}{N}} \\ \vdots \\ e^{j \cdot \frac{2\pi K (U-1)}{N}} \end{bmatrix}$$

} Discrete complex exponential.

- The set of vectors $E_{N,i}\ \forall\ i \in N$ form an orthogonal basis.

Proof $\langle E_{N,k}, E_{N,l} \rangle = \sum_{n=0}^{N-1} e^{j \cdot \frac{2\pi k \cdot n}{N}} \cdot e^{\frac{j \cdot 2\pi l \cdot n}{N}}$

$$= \sum_{n=0}^{N-1} \left(e^{j \cdot \frac{2\pi (k-l)}{N}} \right)^n \quad \text{G.P}$$

$$= \frac{1 - 3^n}{1 - 3}$$

$$= \frac{1 - \left(e^{j \cdot \frac{2\pi (k-l)}{N}} \right)^N}{1 - e^{\frac{j \cdot 2\pi \cdot (k-l)}{N}}}$$

$\because e^{j \cdot 2\pi (k-l)} = 0$ if $k \neq l$

So they are orthogonal.

$= 0$ // if $k \neq l$

Fourier Transform:

$$\begin{bmatrix} X_0 \\ X_1 \\ \vdots \\ X_{N-1} \end{bmatrix} = \frac{1}{N} \begin{bmatrix} e^{j\frac{2\pi \cdot 0 \cdot 0}{N}} & e^{j\frac{2\pi \cdot 0 \cdot 1}{N}} & \cdots & e^{j\frac{2\pi \cdot 0 \cdot (N-1)}{N}} \\ \vdots & \vdots & \cdots & \vdots \\ e^{j\frac{2\pi (N-1) \cdot 0}{N}} & \cdots & \cdots & e^{j\frac{2\pi (N-1)(N-1)}{N}} \end{bmatrix} \begin{bmatrix} V(0) \\ V(1) \\ \vdots \\ V(N-1) \end{bmatrix}$$

$$= \frac{1}{N} \cdot F_{NN} \cdot V_N$$

(We get this from $X_k = \frac{\langle V, V_k \rangle}{\langle V_k, V_k \rangle}$ — vector we form; basis vector)

$$\Rightarrow \boxed{V[k] = \sum_{u=0}^{N-1} X_u \cdot e^{j\frac{2\pi ku}{N}}} \longrightarrow \text{DFT}$$

$$\boxed{X_u = X_u = \frac{1}{N} \sum_{k=0}^{N-1} V[k] \cdot e^{-j\frac{2\pi ku}{N}}} \longrightarrow \text{IDFT}$$

---

ex: Consider $f(x) = 5 + 2\cos(2\pi x - 90) + 3\cos 4\pi x$

$\Rightarrow$ Sample at $0, \frac{1}{4}, \frac{2}{4}, \frac{3}{4}$ } Nyquist freq.

$\Rightarrow f = [8 \;\; 4 \;\; 8 \;\; 0]$

$$\begin{bmatrix} 20 \\ -4j \\ 12 \\ 4j \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} 8 \\ 4 \\ 8 \\ 0 \end{bmatrix} = \text{Fourier transform of } f.$$

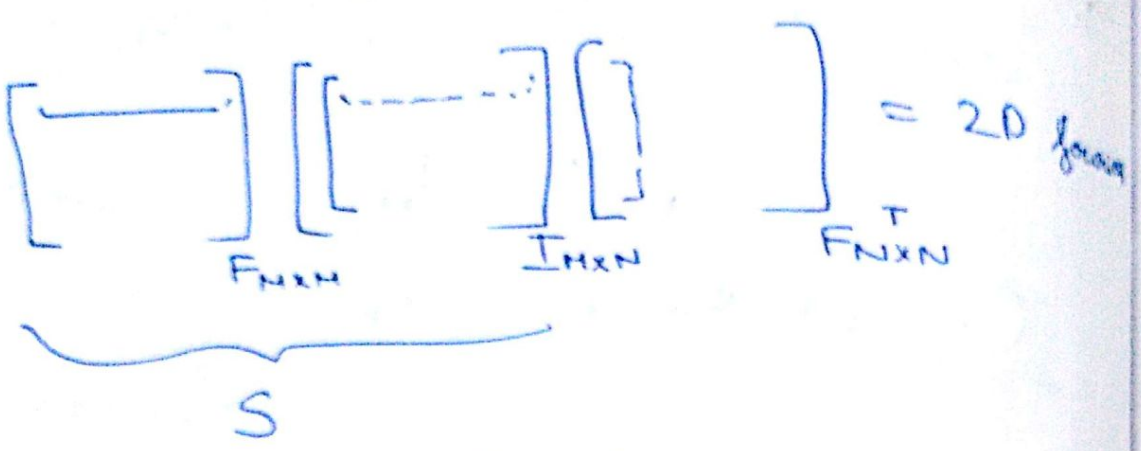
$\Rightarrow F[k] = F^*[N-k]$ $\quad (F[0] = F^*[3])$

- All the other properties also apply, we can get back f from F and so on.

---

*Fourier Transform on Images:

- Apply fourier transform on all the columns first and then apply on the rows next.



- In S we would a fourier transform of columns. ~~This would be of~~. We then do a fourier transform of rows to get final output

⇒ We can pad the image & fourier matrices if we want to have more precise basis.

Note: DFT is really useful if there are patterns in the images which you need to use / remove (noise).

---

* Magnitude Spectrum & Phase Spectrum: (of Fourier transform)

$$|F(u)| = [R^2(u) + I^2(u)]^{1/2}$$

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

- These are usually visualized as images of fourier transforms. (Since complex numbers are hard to visualize)
