Digital Image Processing

BCSE 403L AI + TAI

- Manis Saha (22BCE1538)
- Vedant Kumar (22BCE1632)

Team Leader

Vedant Kumar

PART - A:

a) Harris corner detector

- A corner is an ~~intest~~ point where two edges meet, where the intensity of the image changes significantly in at least two directions. To illustrate this, consider three types of regions in an image
  - Flat regions: In these areas, pixel intensity remains relatively constant. if you move a small window across the image, there will be minimal change in intensity
  - Edge: In an edge, intensity variation is significant in one direction but remains constant in the perpendicular direction. Moving a window along the edge direction results in little change, but moving perpendicular to the edge causes a noticeable intensity difference.
  - Corners: At a corner, intensity varies in multiple directions. This means that moving a small window in any direction results in a significant change in intensity.

The Harris corner Detector analyzes the intensity variations in an image using derivatives. The key idea is to measure how the intensity of an image patch changes when shifted in different directions. This is captured using an approx. of the second-order Taylor series expansion.

Given an image $I(x,y)$, the intensity change $E(u,v)$ for a small shift (u,v) can be approximated as:

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

where $w(x,y)$ is a window function (often a Gaussian function) that gives more importance to the center pixels.

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

where the M matrix

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

I_x, I_y = gradient of image along x and y directions.

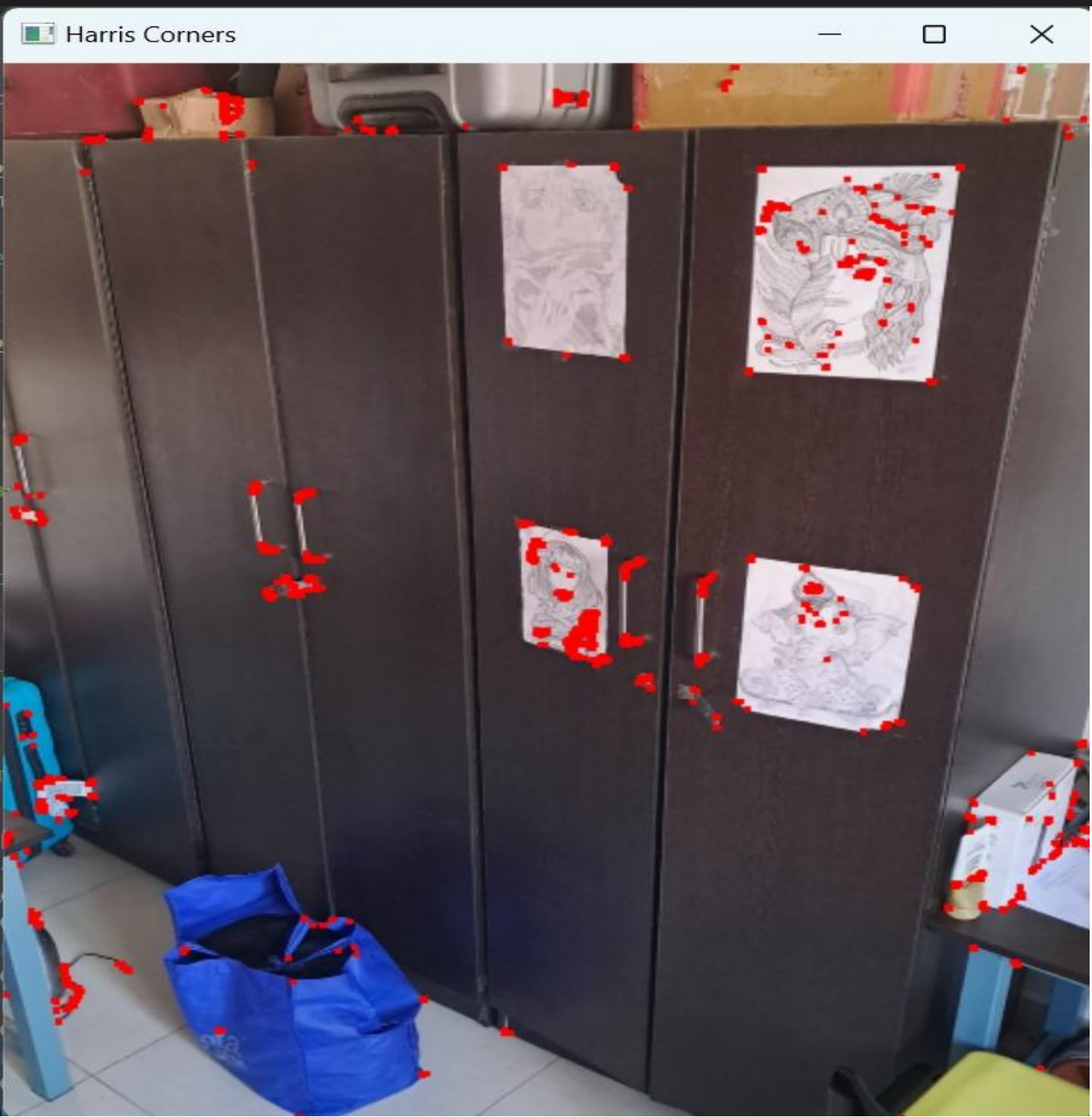
$$R = \det(M) - K(\text{trace}(M))^2$$

- $\det(M) = \lambda_1 \lambda_2$ is the determinant of M
- $\text{trace}(M) = \lambda_1 + \lambda_2$ is sum of its eigen values
- K is an empirical constant, typically range 0.04 to 0.06

- If both λ_1 & λ_2 are small, it's a flat region
- If one eigen value is large and the other is small, it is an edge.
- If both eigen values are large, the region is classified as corner

Original Image:

```python
import cv2
import numpy as np

img = cv2.imread('room.jpg')
img = cv2.resize(img,(480,576))
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
gray = np.float32(gray)

dst = cv2.cornerHarris(gray, blockSize=2, ksize=3, k=0.04)

dst = cv2.dilate(dst, None)
img[dst > 0.01 * dst.max()] = [0, 0, 255]

cv2.imshow('Harris Corners', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



b)

An edge in an image represents a boundary where there is a significant change in pixel intensity. This change can be due to variations in illumination, texture, or object boundaries.

~~Detecting edges~~ is

Edges can be classified into 3 main types.

- Step edges - A sudden intensity change at a boundary
- Ramp edges - A gradual change in intensity over a region
- Roof edges - intensity peaks at a point before gradually decreasing.

Marr - Hildreth edge detection method consist of following steps:

1. Gaussian Smoothing:

Gaussian filter smooths the image by averaging pixel values based on a Gaussian function.

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

where σ is the standard deviation, which controls the level of smoothing. A larger σ results in more blurring.

The image $I(x,y)$ is convolved with this Gaussian function to produce a smoothed version $I_s(x,y)$:

$$I_s(x,y) = G(x,y) * I(x,y)$$

2. Applying the laplacian operator.

The second step is to compute the laplacian of the imag. The laplacian is a second-order derivative operator given by:

$$\nabla^2 I_s(x,y) = \frac{\partial^2 I_s}{\partial x^2} + \frac{\partial^2 I_s}{\partial y^2}$$

This operator highlights region of rapid intensity change and is useful for detecting edges.

3. Detecting Zero-Crossings

Edges are located where the laplacian of the smoothed image changes sign, These points are called zero-crossings, indicate strong edges. To detect zero-crossings, the laplacian-filtered image is scanned for neighboring pixels that have opp. signs.

4. Thresholding

Not all zero-crossings correspond to meaningful edges. A threshold is applied to filter out weak edges and retain only the most significant ones

```python
import cv2
import numpy as np

img = cv2.imread('room.jpg', cv2.IMREAD_GRAYSCALE)
img = cv2.resize(img,(480,576))

sigma = 1.4
blurred = cv2.GaussianBlur(img, (5,5), sigma)
laplacian = cv2.Laplacian(blurred, cv2.CV_64F)

laplacian_abs = np.uint8(np.absolute(laplacian))

_, edges = cv2.threshold(laplacian_abs, 30, 255, cv2.THRESH_BINARY)

cv2.imshow('Marr-Hildreth Edge Detection', edges)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

c) canny edge Detection

- The canny edge detection method ensures that edges are detected accurately uniquely, and with minimal noise interference

steps involved in canny edge detection

1 Noise Reduction using Gaussian Blur

Gaussian filters are used to smoothen an image to reduce the noise present.

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

standard deviation σ determines the level of smoothing. A larger σ results in stronger blurring.

2. Computing Image Gradients (Edge strength and Direction)

Edges correspond to regions of high intensity change. The gradient is computed using Sobel operators to obtain derivatives in both x & y directions:

$$G_x = \frac{\partial I}{\partial x}, \quad G_y = \frac{\partial I}{\partial y}$$

The gradient magnitude G is given by:

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

The gradient direction θ (angle of the edge)

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

3. Non-Maximum Supression

After computing gradient magnitude and direction, a technique called non-maximum suppression is applied to thin the edges.

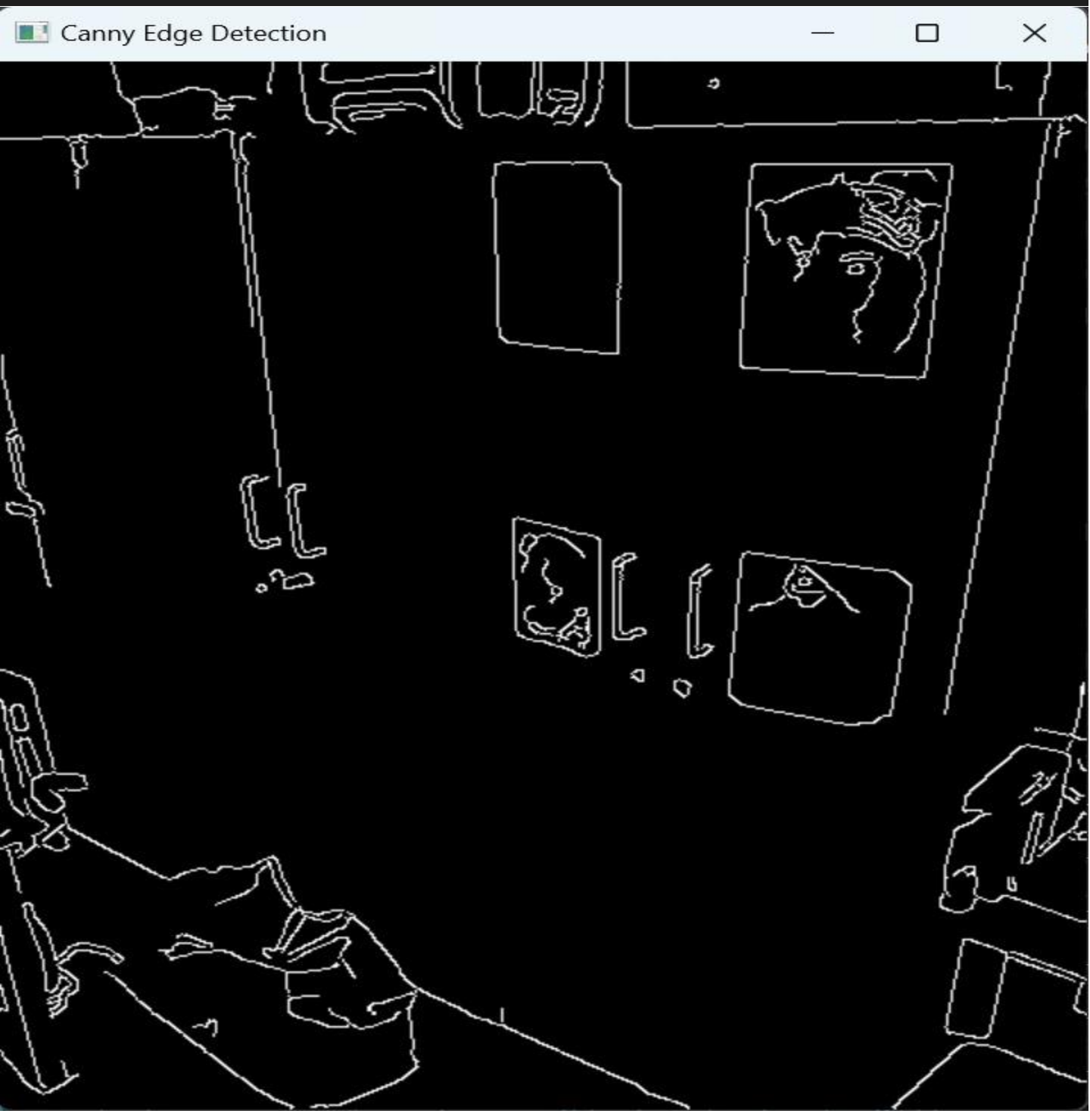
- For each pixel, the gradient direction is used to check the two neighboring pixels along the edge direction.
- If the current pixel is not the maximum compared to its neighbors, it is suppressed (set to zero).
- This step ensures that edges are represented as thin, sharp lines.

4. Hysteresis Thresholding

To determine which edges are important, two threshold values are used

- Upper Threshold (T_high): Pixels with gradient magnitudes above this value are definately edges.
- Lower Threshold (T_low): Pixels below this value are discarded.
- Intermediate pixels (b/w 2 Threshold) are considered edges only if they are connected to strong edges.

```python
import cv2
import numpy as np

img = cv2.imread('room.jpg', cv2.IMREAD_GRAYSCALE)
img = cv2.resize(img,(480,576))
sigma = 1.4
blurred = cv2.GaussianBlur(img, (5,5), sigma)

lower_threshold = 50
upper_threshold = 150
edges = cv2.Canny(blurred, lower_threshold, upper_threshold)

cv2.imshow('Canny Edge Detection', edges)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



d) Image Pyramids

- Image Pyramids work by progressively reducing the resolution of an image, forming a hierachical structure. Two main types of pyramids are:

1. Gaussian Pyramid - used for image smoothing and downsampling
2. Laplacian Pyramid - Used for reconstructing images and highlighting fine details.

The idea behind image pyramids is to construct multiple layers of an image, where each subsequent layer is smaller, downsampled to form a sequence of images. Each level in the pyramid has fewer pixel than previous.

1. Gaussian Pyramid (Down sampling / up sampling)

The Gaussian pyramid is created by successively applying a Gaussian filter and downsampling. to form a sequences of images.

Downsampling: The image is first blurred using a Gaussian filter, Then every second pixel is removed to reduce the resolution.

Upsampling: The image can also be expanded to a higher resolution by interpolating missing pixel values.

The down sampled image G_{i+1} at level $i+1$ is computed as

$$G_{i+1}(x,y) = \sum_{m=-2}^{2} \sum_{n=-2}^{2} w(m,n)\, G_i(2x+m, 2y+n)$$

where $w(m,n)$ represents the Gaussian filter weights.

2. Laplacian Pyramid (Edge emphasis & reconstruction)

The laplacian pyramid is derived from the Gaussian Pyramid and is used for image sharpening, compression, and Blending Each level of the laplacian pyramid is computed as the difference between a Gaussian pyramid level and it's upsampled version of the next level:

$$L_i = G_i - \text{upsample}(G_{i+1})$$

Since the Laplacian pyramid captures the differences between different resolutions, it is useful for edge enhancement and for feature extraction.

image_pyramids.py > ...

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt


img = cv2.imread('room.jpg')
img = cv2.resize(img, (480, 576))


gp = [img]
for i in range(3):
    img = cv2.pyrDown(img)
    gp.append(img)


lp = [gp[-1]]
for i in range(3, 0, -1):
    gaussian_expanded = cv2.pyrUp(gp[i])
    laplacian = cv2.subtract(gp[i-1], gaussian_expanded)
    lp.append(laplacian)


gp_rgb = [cv2.cvtColor(g, cv2.COLOR_BGR2RGB) for g in gp]
lp_rgb = [cv2.cvtColor(l, cv2.COLOR_BGR2RGB) for l in lp]


def plot_pyramid(images, title):
    fig, axes = plt.subplots(1, 4, figsize=(12, 6))
    fig.suptitle(title, fontsize=16, fontweight='bold')

    for i, ax in enumerate(axes):
        ax.imshow(images[i], aspect='auto')
        ax.set_xticks([])
        ax.set_yticks([])
        ax.set_title(f'Level {i}', fontsize=12)

    plt.tight_layout()
    plt.show()

plot_pyramid(gp_rgb, "Gaussian Pyramid")
plot_pyramid(lp_rgb, "Laplacian Pyramid")

```

Figure 1

Gaussian Pyramid

Level 0 | Level 1 | Level 2 | Level 3

Figure 1

Laplacian Pyramid

Level 0 | Level 1 | Level 2 | Level 3

PART-B

a)

Q 2.5)

a) image size = 2048 x 2048 pixels

Print size = 5cm x 5cm

Resolution in pixels per mm (px/mm):

$$\frac{2048}{5\text{cm} \times 10\text{mm/cm}} = \frac{2048}{50} = 40.96 \text{ px/mm}$$

Since 1 line pair consist of two pixels, resolution per mm is

$$\Rightarrow \frac{40.96 \text{ px/mm}}{2} = 20.48 \text{ lp/mm}$$

b) Resolution in dpi for a 2x2 inch print size

Resolution in pixels per inch:

$$\frac{2048}{2 \text{ inches}} = 1024 \text{ dpi}$$

$$\Rightarrow 1024 \text{ dpi}$$

Q2.6)

CCD sensor size = 7×7 mm

Resolution = 1024×1024 pixels

object distance d_o = 0.5 m

lens focal length = 35 mm

Using The thin-lens equation:

$$\frac{1}{f} = \frac{1}{d_o} + \frac{1}{d_i}$$

image distance d_i:

$$d_i = \frac{f d_o}{d_o - f}$$

Substituting values:

$$d_i = \frac{(35\text{ mm})(500\text{ mm})}{500 - 35}$$

$$d_i = \frac{17500}{465} \approx 37.63\text{ mm}$$

magnification M is:

$$M = \frac{d_i}{d_o} = \frac{37.63}{500} \approx 0.0753$$

$$FOV = \frac{\text{sensor size}}{M}$$

$$= \frac{7\text{mm}}{0.0753} \approx 93\text{ mm}$$

The pixel size on the sensor is:

$$\frac{7mm}{1024} \approx 0.00684 \text{ mm/Pixel}$$

The pixel size in the object plane:

$$\text{Pixel size} = \frac{0.00684}{M}$$

$$\approx 0.091 \text{ mm}$$

Calculate the line pairs per millimeter (lp/mm)

$$= \frac{1}{2 \times \text{Pixel size}}$$

$$= \frac{1}{2 \times 0.091}$$

$$\approx 5.49 \text{ lp/mm}$$

Q2.7)

Object distance (d_o) = 1m

Object plane = 5 x 5 cm

focal length = 200mm

Req. resolution => 5 lp/mm

=>

$$\frac{1}{f} = \frac{1}{d_o} + \frac{1}{d_i}$$

$$d_i = \frac{f d_o}{d_o - f}$$

$$d_i = \frac{200 \times 1000}{1000 - 200} = 250 \text{ mm}$$

$$M = \frac{di}{do} = \frac{250}{1000} = 0.25$$

$$\text{img size} = M \times \text{obj. size}$$
$$= 0.25 \times 50\,mm$$
$$= 12.5\,mm$$

$\therefore$ req. CCD size is $12.5 \times 12.5\,mm$

Number of pixels:

$$\text{Pixel size} \leq \frac{1}{2 \times 5} = \frac{1}{10} = 0.1\,mm/\text{pixel}$$

no. of pixel along one dimension is:

$$\frac{\text{img size}}{\text{pixel size}} = \frac{12.5}{0.1} = 125 \text{ pixel}$$

For square CCD chip, the total required sensing elements:

$$125 \times 125 = 15,625 \text{ Pixels}$$

Q2.9)

No. of images = 500

img size = 1024 x 1024 px

intensity = 256

Transmission format = each packet consist of:

start bit = 1 bit

Data = 8bit

stop bit = 1 bit

Total bit Byte = 10 bits

Baud rate

a) 3 M-baud = 3×10^6 bits / sec

b) 30 G-baud = 30×10^9 bits/sec

No. of pixel in each img = 1024 x 1024

= 1048576 px

No. of bits per img = 8x1024x1024

Transmission overhead = 10 bits per byte

Total transmitted bits = $\frac{8 \times 1024 \times 1024 \times 10}{8}$

= 10485760

for 500 images

= 10485760 x500

= 5242880000

Transmission time

a) using 3M-baud modem

Time = 5242880000 / 3×10^6

= 29.13 min

b) using 30 G-baud

T = 5242880000 / 30×10^9

= 175 millisec

Q2.10)

no. of horizontal lines = 1125 (vertical resolution)

Aspect ratio = 16:9

Frame rate = $\frac{60}{2}$ = 30 fps

color depth per pixel = 24 bits

Movie duration 2hrs

Horizontal resolution = $\frac{16}{9} \times 1125$

= 2000 px

Total pixel per frame = horizontal px × vertical px

= 2000 × 1125

= 225×10^4 px

Total bits per frame = $225 \times 10^4 \times 24$

= 54×10^6

Total frame = 30 × 60 × 120

= 216,000

∴ Total bits required

= $216 \times 10^3 \times 54 \times 10^6$

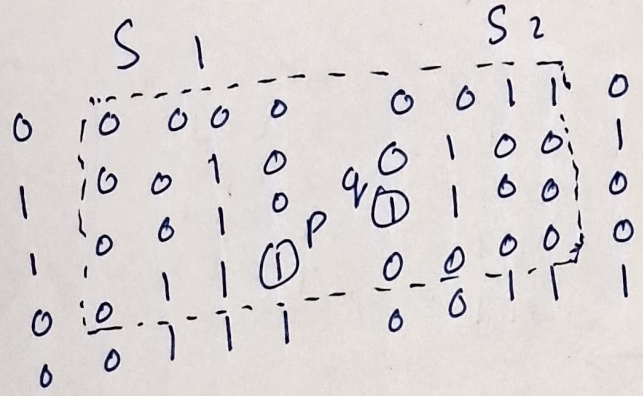
= 1.1664×10^{13} bits

Q 2.14)

given:

set V = {1}.

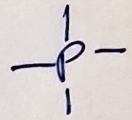
This means we are only considering pixels with value 1 for connectivity



Rules

4 adjacent:

a) q should be in the set of N4(P)



as · q is not in the set of N4(P)

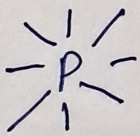
it is not 4-adjacent

b) 8 adjacent:

q. should be in the set of N8(P)

as q is the set of N8(P)

it is 8 - adjacent



c) m adjacent

1. N4(P) intersection N4(q) should ≠ to {V}
2. q in ND(P)

as both condition satisfy

it is m-adjacent

Q 2.18)

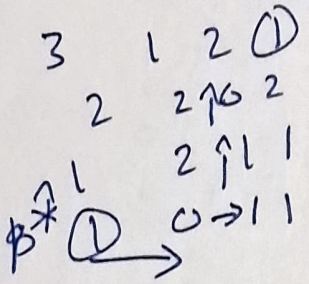
a) To find shortest 4-, 8-, m- path between P & Q

(i) V = {0,1}

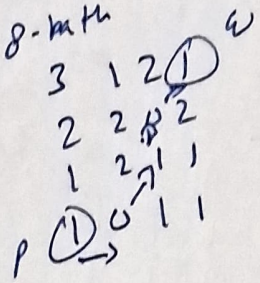
(ii) V = {1,2}

⇒ (i) V{0,1}

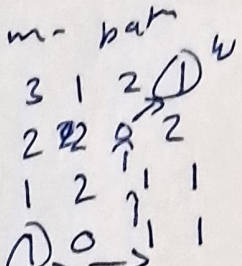
4-path



8-path



m-path



For 1st item

$N_4(P) = \{0,1\}$

2nd item

$N_4(P) \neq \{\}$

as no path is found

∴ Not possible

∴ path len = 4

at last node

$N_4(P) = \{2,2\}$

$N_4(Q) = \{2,2\}$

intersection = {2,2}

∴ m-path is possible

and len = 5

$V = \{1, 2\}$

4-Path

3	1	1→2→1		a
↑2→2	↑	0	2	
↑1	2	1	1	
P 1	0	1	1	

8-Path

3	1	2→1		a
2	2	↑ 0	2	
1	↑ 2	1	1	
P 1	→ 0	1	1	

m-Path

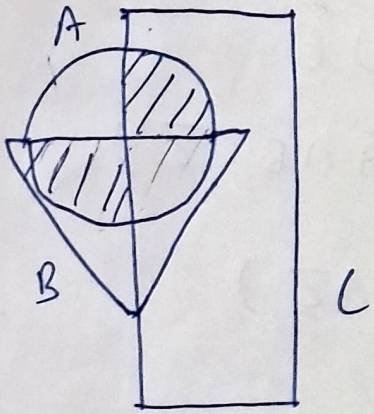
3	↑1→2→1			a
↑2→2	0	2		
↑1	2	1	1	
P 1	0	1	1	

$\therefore$ Path length $= 4$

Path length $= 4$

$\therefore$ $\text{len}(4 \cap 4(P)) = 4$

(Q 2.32)

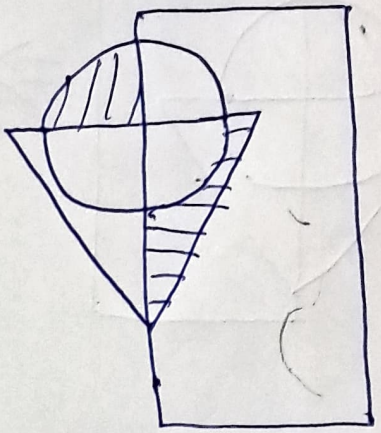


a) $(A \cap B \cap \bar{C}) \cup (A \cap \bar{B} \cap C)$

$\Rightarrow A \cap (B \cap \bar{C} \cup \bar{B} \cap C)$

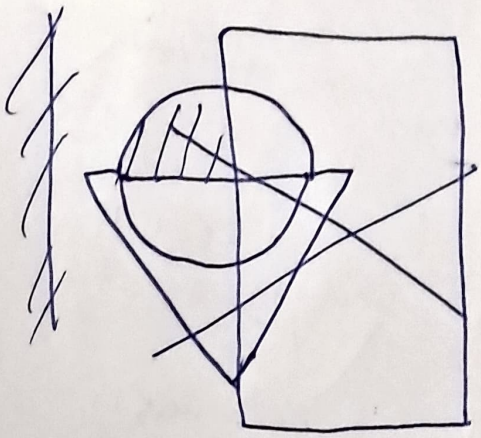
$\Rightarrow A \cap (B \oplus C)$

~~$(A \cap B) \cup (A \cap C)$~~

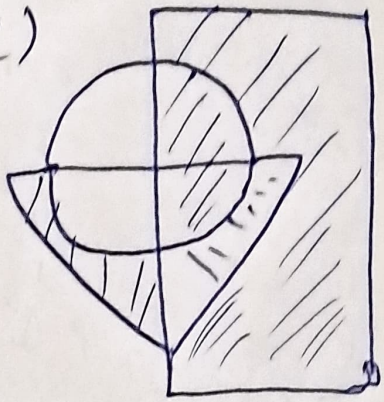


b) $(A \cap \bar{B} \cap \bar{C}) \cup (B \cap C \cap \bar{A})$

$\Rightarrow (A \oplus B) \cap (\bar{A} \oplus C)$



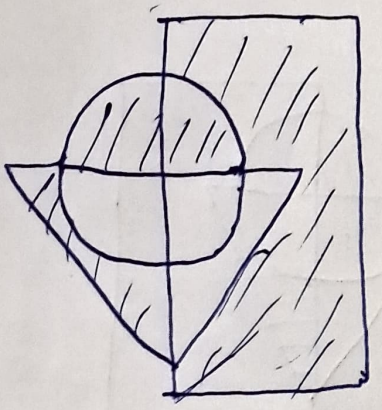
c)



$\Rightarrow C \cup (\bar{A} \cap B \cap \bar{C})$

$= C \cup (\bar{A} \cap B)$

d) ——



$\Rightarrow$

$\bar{B} \cup (\bar{A} \cap B \cap \bar{C})$

$\Rightarrow \bar{B} \cup (\bar{A} \cap \bar{C})$

Q2.36)

nomogeneous Trans formatia matia is giua by:

$$T = \begin{bmatrix} a & b & tx \\ c & d & ty \\ 0 & 0 & 1 \end{bmatrix}$$

- (tx, ty) represents translation

2 ~~3b~~)

a) scaling and Translation

$$S = \begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$TS = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} sx & 0 & tx \\ 0 & sy & ty \\ 0 & 0 & 1 \end{bmatrix}$$

b) scaling, Translation and Rotation

$$R = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$TRS = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} sx\cos\theta & -sy\sin\theta & tx \\ sx\sin\theta & sy\cos\theta & ty \\ 0 & 0 & 1 \end{bmatrix}$$

c) Vertical sheer, Scaling, Translation and Rotation

$$Sh = \begin{bmatrix} 1 & Shx & 0 \\ Shy & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- composite transformation is

$$TRSSh = \begin{bmatrix} Sx\cos\theta + Shx\sin\theta & -Sy\sin\theta + Shx\,Sy\cos\theta & tx \\ Sx\sin\theta + Shy\cos\theta & Sy\cos\theta + Shy\,Sy\sin\theta & ty \\ 0 & 0 & 1 \end{bmatrix}$$

d) Scaling first, then translation:

$$TS = \begin{bmatrix} Sx & 0 & tx \\ 0 & Sy & ty \\ 0 & 0 & 1 \end{bmatrix}$$

. Scale the object, then move it.

2. Translation first, then scaling:

$$ST = \begin{bmatrix} Sx & 0 & Sx.tx \\ 0 & Sy & Sy.ty \\ 0 & 0 & 1 \end{bmatrix}$$

translation is also scaled

Q2.32)

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & a_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

a) $S = \begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$, $S^{-1} = \begin{bmatrix} 1/sx & 0 & 0 \\ 0 & 1/sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$

b) $T = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix}$, $T^{-1} = \begin{bmatrix} 1 & 0 & -tx \\ 0 & 1 & -ty \\ 0 & 0 & 1 \end{bmatrix}$

c) $Sh = \begin{bmatrix} 1 & shx & 0 \\ shy & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, $Sh^{-1} = \begin{bmatrix} \frac{1}{1-shx\,shy} & \frac{-shx}{1-shx\,shy} & 0 \\ \frac{-shy}{1-shx\,shy} & \frac{1}{1-shx\,shy} & 0 \\ 0 & 0 & 1 \end{bmatrix}$

d) Inverse Rotation

$$R = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad R^{-1} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

e) $$TR = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta & -\sin\theta & tx \\ \sin\theta & \cos\theta & ty \\ 0 & 0 & 1 \end{bmatrix}$$

$$(TR)^{-1} = R^{-1}T^{-1}$$

$$= \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -tx \\ 0 & 1 & -ty \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta & \sin\theta & -tx\cos\theta - ty\sin\theta \\ -\sin\theta & \cos\theta & tx\sin\theta - ty\cos\theta \\ 0 & 0 & 1 \end{bmatrix}$$

3.4 b) Find all the bit planes on the following 4-bit image!

0	1	8	6
2	2	1	1
1	15	14	12
3	6	9	10

PART - B.

Manis Saha
22BCE1538
Vedant Kumar
22BCE1632

Solution

Convert Each Pixels to 4-bit Binary Representation

0000	0001	1000	0110
0010	0010	0001	0001
0001	1111	1110	1100
0011	0110	1001	1010

Extract Bit Planes

a) Bit Plane 3 (Most Significant Bit - MSB).

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

b) Bit Plane 2 $\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}$

c) Bit Plane 1 $\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix}$

d) Bit Plane 0 (Least Significant Bit - LSB)

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

3.11 Assume continuous intensity values, and suppose that the intensity values of an image have the PDF $p_r(r) = 2r/(L-1)^2$ for $0 \le r \le L-1$, and $p_r(r) = 0$ for other values of r.

a) Find the transformation function that will map the input intensity values, r, into values, s, of a histogram-equalized image.

b) Find the transformation function that (when applied to the histogram-equalized intensities, s) will produce an image whose intensity PDF is $p_z(z) = 3z^2/(L-1)^3$ for $0 \le z \le L-1$ and $p_z(z) = 0$ for other values of z.

c) Express the transformation function from (b) directly in terms of r, the intensities of the input image.

Solution a) Compute the CDF for Histogram Equalization

The transformation function of histogram equalization is:

$$s = T(r) = \int_0^r P_r(r)\,dr$$

Given PDF: $P_r(r) = \frac{(L-r)^2}{12}, \quad 0 \le r \le L-1$

Computing the cumulative distribution function (CDF):

$$T(r) = \int_0^r \frac{(L-r')^2}{12}\,dr'$$

$$T(r) = \frac{1}{12}\int_0^r (L^2 - 2Lr' + r'^2)\,dr'$$

$$T(r) = \frac{1}{12}\left[L^2 r - Lr'^2 + \frac{r'^3}{3}\right]_0^r$$

$$T(r) = \frac{1}{12}\left[L^2 r - Lr^2 + \frac{r^3}{3}\right]$$

Thus, the transformation function for histogram equalization is:

$$s = T(r) = \frac{L^2 r - Lr^2 + r^3/3}{12}$$

b) Compute the CDF of the desired PDF

Given target PDF: $P_z(z) = \frac{(L-z)^3}{2(L-1)^3}, \quad 0 \le z \le L-1$

The transformation function for obtaining a histogram-matching transformation is:

$$z = G(s) = \int_0^s P_z(z)\,dz$$

$$G(s) = \int_0^s \frac{(L-z')^3}{2(L-1)^3}\,dz'$$

Expanding: $G(s) = \frac{1}{2(L-1)^3}\int_0^s (L-z')^3\,dz'$

Using the integral formula:

$$\int (a-x)^n\,dx = \frac{(a-x)^{n+1}}{-(n+1)}$$

Applying it: $G(s) = \frac{1}{2(L-1)^3} \times \frac{(L-z')^4}{-4}\Big|_0^s$

$$G(s) = \frac{1}{-8(L-1)^3}\left[(L-s)^4 - L^4\right]$$

Rearranging: $z = G(s) = L - (L^4 - 8(L-1)^3 s)^{1/4}$

c) Express the Final Transformation Function in Terms of r

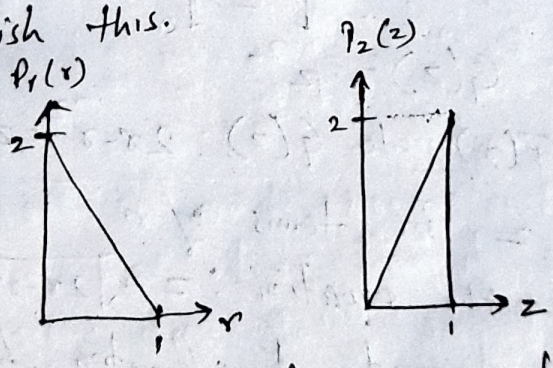
Since $s = T(r)$ and $z = G(s)$, we substitute $T(r)$ into $G(r)$:

$$z = L - \left(L^4 - 8(L-1)^3 \times \frac{L^2 r - Lr^2 + r^3/3}{12}\right)^{1/4}$$

Thus, the direct transformation from r to z is:

$$z = L - \left(L^4 - \frac{8(L-1)^3}{12}\left(L^2 r - Lr^2 + r^3/3\right)\right)^{1/4}$$

3.12 An image with intensities in the range $[0, 1]$ has the PDF, $p_r(r)$, shown in the following figure. It is desired to transform the intensity levels of this image so that they will have the specified $p_z(z)$ shown in the figure. Assume continuous quantities, and find the transformation (expressed in terms of r and z) that will accomplish this.



Solution From the given image, we observe the probability density function (PDFs) of the input intensity $p_r(r)$ and the desired output intensity $p_z(z)$.

→ $p_r(r) = 2(1-r)$, $0 \le r \le 1$

→ $p_z(z) = 2z$, $0 \le z \le 1$

Compute the CDF of $p_r(r)$

The cummulative distribution function (CDF) is:

$$T(r) = \int_0^r p_r(r')\,dr'$$

$$T(r) = \int_0^r 2(1-r')\,dr'$$

$$T(r) = 2\int_0^r (1-r')\,dr'$$

$$T(r) = 2\left[r' - \frac{r'^2}{2}\right]_0^r$$

$$T(r) = 2\left(r - r^2/2\right)$$

$$T(r) = 2r - r^2$$

Since histogram equalization transforms r into s using $s = T(r)$, we set: $s = 2r - r^2$

Compute the Inverse of $G(s)$

The required intensity transformation follows:

$$s = T(r) = G^{-1}(z)$$

The CDF of the desired output PDF $p_z(z)$ is:

$$G(z) = \int_0^z p_z(z')\,dz'$$

$$G(z) = \int_0^z 2z'\,dz'$$

$$G(z) = 2\left[\frac{z'^2}{2}\right]_0^z$$

$$G(z) = z^2$$

Equating $T(r)$ to $G(z)$: $2r - r^2 = z^2$

Solve for z in terms of r

Rearrange the equation: $z = \sqrt{2r - r^2}$

Thus, the required transformation function is:

$$z = \sqrt{2r - r^2}$$

3.18 You are given the following kernel and image:

$$w = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \qquad f = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

a) Give a sketch of the area encircled by the large ellipse in Fig. 3.28 when the kernel is centered at point (2,3) (2nd row, 3rd col) of the image

shown above. Show specific values of w and f.

b) Compute the convolution $w * f$ using the minimum zero padding needed. Show the details of your computations when the kernel is centered on point (2,3) of f; and then show the final full convolution result.

c) Repeat (b), but for correlation, $w \star f$.

Solution a) ~~Starting~~ When the kernel is centered at (2,3) (2nd row, 3rd column in 1-based indexing), the 3×3 window of the image covered by the kernel is:

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

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

b) Convolution is defined as:

$$(f * w)(x,y) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} w(i,j) f(x-i, y-i)$$

We multiply the kernel with the corresponding image region:

$(1\times0) + (2\times0) + (1\times0) + (2\times0) + (4\times1) + (2\times0) + (1\times0) + (2\times1) + (1\times0)$

$= 0+0+0+0+4+0+0+2+0 = \underline{\underline{6}}$.

Performing similar calculations at all valid positions, the full convolution output is:

$$\begin{bmatrix} 0 & 0 & 2 & 0 & 0 \\ 0 & 2 & 6 & 2 & 0 \\ 0 & 4 & 10 & 4 & 0 \\ 0 & 2 & 6 & 2 & 0 \\ 0 & 0 & 2 & 0 & 0 \end{bmatrix}$$

c) The correlation operation is similar to convolution, except the kernel is not flipped. Since our kernel is symmetric, the result remains the same as convolution:

$$\begin{bmatrix} 0 & 0 & 2 & 0 & 0 \\ 0 & 2 & 6 & 2 & 0 \\ 0 & 4 & 10 & 4 & 0 \\ 0 & 2 & 6 & 2 & 0 \\ 0 & 0 & 2 & 0 & 0 \end{bmatrix}$$

3.21 Given the following kernel and image:

$$w = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \qquad f = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

a) Give the convolution of the two.

b) Does your result have a bias?

Solution a) Convolution Result =

$$= \begin{bmatrix} \Sigma(w \cdot f) & \Sigma(w \cdot f) & \Sigma(w \cdot f) \\ \Sigma(w \cdot f) & \Sigma(w \cdot f) & \Sigma(w \cdot f) \\ \Sigma(w \cdot f) & \Sigma(w \cdot f) & \Sigma(w \cdot f) \end{bmatrix}$$

$= (1\times1)+(2\times1)+(1\times1)+(2\times1)+(4\times1)+(2\times1)+(1\times1)+(2\times1)+(1\times1)$

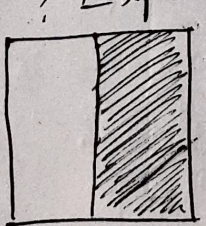
$= 1+2+1+2+4+2+1+2+1 = 16.$

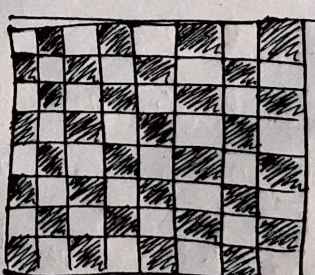
Convolved Output $= \begin{bmatrix} 16 & 16 & 16 \\ 16 & 16 & 16 \\ 16 & 16 & 16 \end{bmatrix}$

b). No bias included.

3.26 The two images shown in the following figure are quite different, but their histogram are the same. Suppose that each image is blurred using 3×3 box kernel.

a) Would the histograms of the blurred images still be equal? Explain.





b) If your answer is no, either sketch the two histogram or give two tables, detailing the histogram components.

Solution: a) No, the histograms of the blurred images would not remain the same after applying the 3x3 box kernel.

→ Reason: Blurring using 3x3 box kernel involves averaging the pixel values in a neighborhood, which reduces high-frequency details (sharp transitions between light and dark regions). Although the two images might have the same histogram before blurring, the blurring operation will smooth out local variations in pixel intensity, potentially changing the distribution of pixels values in each image.

→ Effect of Blurring: The image that was originally a sharp transition (white or dark) will have its edges softened, while the image with a matrix of black and white boxes will lose some of its grid-like structure. The histograms of these blurred images will differ because the blurring process affects how intensities are spread across the image.

b) Since the histograms of the two images would differ after blurring, here is a general outline of how the histograms could differ:

→ Before blurring:

- Image 1 (white to dark transition): The histogram would likely have higher intensity values on the light side and lower intensity values on the dark side, with a steep drop-off in between.
- Image 2 (black and white matrix): The histogram would have spikes at the black (low intensity) and white (high intensity) values, showing more distinct peaks in comparison to Image 1.

→ After Blurring:

- Image 1 (blurred): The sharp transition between white and dark will be less pronounced, and the histogram will become more spread out, likely showing a smoother gradient.
- Image 2 (blurred): The grid-like structure will become less distinct, and the histogram may show more intermediate intensity values (gray levels), with a smoother distribution than the sharp peaks of the original image.

Example Histograms:

Intensity	Image 1 (Before Blurring)	Image 2 (Before Blurring)
0	Low	High
1	Low	Low
255	High	Low

After applying the blur (with the box kernel), the histograms for both images might shift and spread out differently:

Intensity	Image 1 (After Blurring)	Image 2 (After Blurring)
0	Moderate	Moderate
1	Moderate	Moderate
255	Moderate	Moderate.

3.27 An image is filtered four times using a Gaussian kernel of size 3×3 with a standard deviation of 1.0. Because of the associative property of convolution, we know that equivalent results can be obtained using a single Gaussian kernel formed by convolving the individual kernels.

a) What is the size of the single Gaussian kernel?

b) What is its standard deviation?

Solution a) When multiple Gaussian filters are applied sequentially, the result filter is still a Gaussian, but with an increased standard deviation. However, the size of the kernel does not simply add up. Instead, for practical purposes, the kernel size is typically chosen based on the standard deviation (σ). using the rule: Kernel size $\approx 6\sigma+1$

Since each of the four applications uses a 3×3 kernel, the final equivalent kernel size remains determined by the increased standard deviation, not just the sum of individual kernel sizes. Thus, while the standard deviation changes, the size of the resulting Gaussian kernel should be large enough to cover the new effective standard deviate, likely requiring a larger kernel size than 3×3. A more precise calculation would require checking the combined spread, but a reasonable estimate suggests at least a 7×7 and 9×9 kernel.

b) The standard deviate of the combined Gaussian filter follows: $\sigma_{final} = \sqrt{\sigma_1^2+\sigma_2^2+\sigma_3^2+\sigma_4^2}$

Given that each Gaussian filter has $\sigma=1.0$:

$$\sigma_{final} = \sqrt{1^2+1^2+1^2+1^2} = \sqrt{4} = 2.0$$

Thus, the equivalent single Gaussian filter has a standard deviation of 2.0.

3.28 An image is filtered with three Gaussian lowpass kernels of size 3×3, 5×5, 7×7, and standard deviations 1.5, 2, and 4, respectively. A composite filter, w, is formed as the convolution of these three filters.

a) Is the resulting filter Gaussian? Explain.

b) What is its standard deviation?

c) What is its size?

Solution a) Yes, the resulting filtering is still Gaussian.

→ The convolution of multiple Gaussian filters results in another Gaussian filter due to the associativity property of convolution and the Gaussian property of stability under convolution.

→ This means that applying multiple Gaussian filters sequentially is equivalent to applying a single Gaussian filter with a new, combined standard deviation.

b) The standard deviation of the composite Gaussian filter follows the formula:

$$\sigma_{final} = \sqrt{\sigma_1^2 + \sigma_2^2 + \sigma_3^2}$$

$$\sigma_{final} = \sqrt{1.5^2 + 2^2 + 4^2} = \sqrt{2.25 + 4 + 16} = \sqrt{22.25} \approx 4.71$$

Thus, the final S.D. is 4.71.

c) Gaussian kernel size is typically determined using the empirical formula: Kernel size $= 6\sigma + 1$

For $\sigma_{final} = 4.71$, Kernel size $= 6(4.71) + 1$

$= 28.26 \approx 29$

Thus, the final kernel size is $\approx 29 \times 29$.

3.29 Discuss the limiting effect of repeatedly filtering an image with a 3×3 low pass filter kernel. You may ignore border effects.

Solution. Repeatedly filtering an image with a 3×3 low pass filter (such as a box filter or Gaussian filter) has the following limiting effects:

1. Progressive Smoothing - Each application reduces high-frequency details, progressively blurring edges and textures.

2. Convergance to a Uniform Image - As filtering continues, pixels values gradually become uniform, leading to a completely smooth image with minimal contrast.

3. Gaussian Approximation - If the filter is Gaussian, repeated applications result in an equivalent Gaussian filter with increasing standard deviation, effectively widening the blur.

4. Loss of High-Fregency Information - Fine details and noise are removed faster than larger structures, leading to feature loss over time.

5. Asymptotic Behaviour - After many iterations, the image approaches a steady-state where further filtering has negligible effect. Ignoring border effects, the final result is a nearly constant grey image.

3.30 In Fig 3.42(b) the corners of the estimated shading pattern appear darker or lighter than their surrounding areas. Explain the reason for this.

Solution: The effect observed in Fig 3.42(b), where the corners of the estimated shading pattern appear darker or lighter than their surrounding, is due to the boundary effects caused by filtering operations, particularly in lowpass filtering or Gaussian smoothing.

Reasons:

1. Edge Padding or Truncation:

→ When applying a filter, especially a Gaussian or box filter, at image boundaries, there may be insufficient neighboring pixels to contribute to the convolution, leading to artificially lower or higher values at the corners.

→ If zero-padding is used, it results in darker corner due to missing intensity contributions.

2. Accumulated Averaging Effects:

→ The filtering process involves local averaging, and at the corners, fewer neighboring pixels contribute, leading to a different intensity than the central regions.

3. Uneven Illumination Estimation:

→ The estimated shading may assume a smooth gradient, but the lack of pixel data at the corners leads to discrepancies, making them appear artificially darker or lighter.

Conclusion:

This effect is a direct result of boundary artifacts in filtering operations, where insufficient data at the edges causes deviations in the estimated shading intensity.

```python
# MANIS SAHA (22BCE1538) & VEDANT KUMAR (22BCE1632)
# OpenCV Program to display Original Images used in PART-C

import cv2
from google.colab.patches import cv2_imshow

img1 = cv2.imread('/image1.jpg')
img2 = cv2.imread('/image2.jpg')

if img1 is None:
    print("Error: image1.jpg not found or cannot be read.")
    exit()

if img2 is None:
    print("Error: image2.jpg not found or cannot be read.")
    exit()

print("Original Image 1:")
cv2_imshow(img1)

print("Original Image 2:")
cv2_imshow(img2)
```

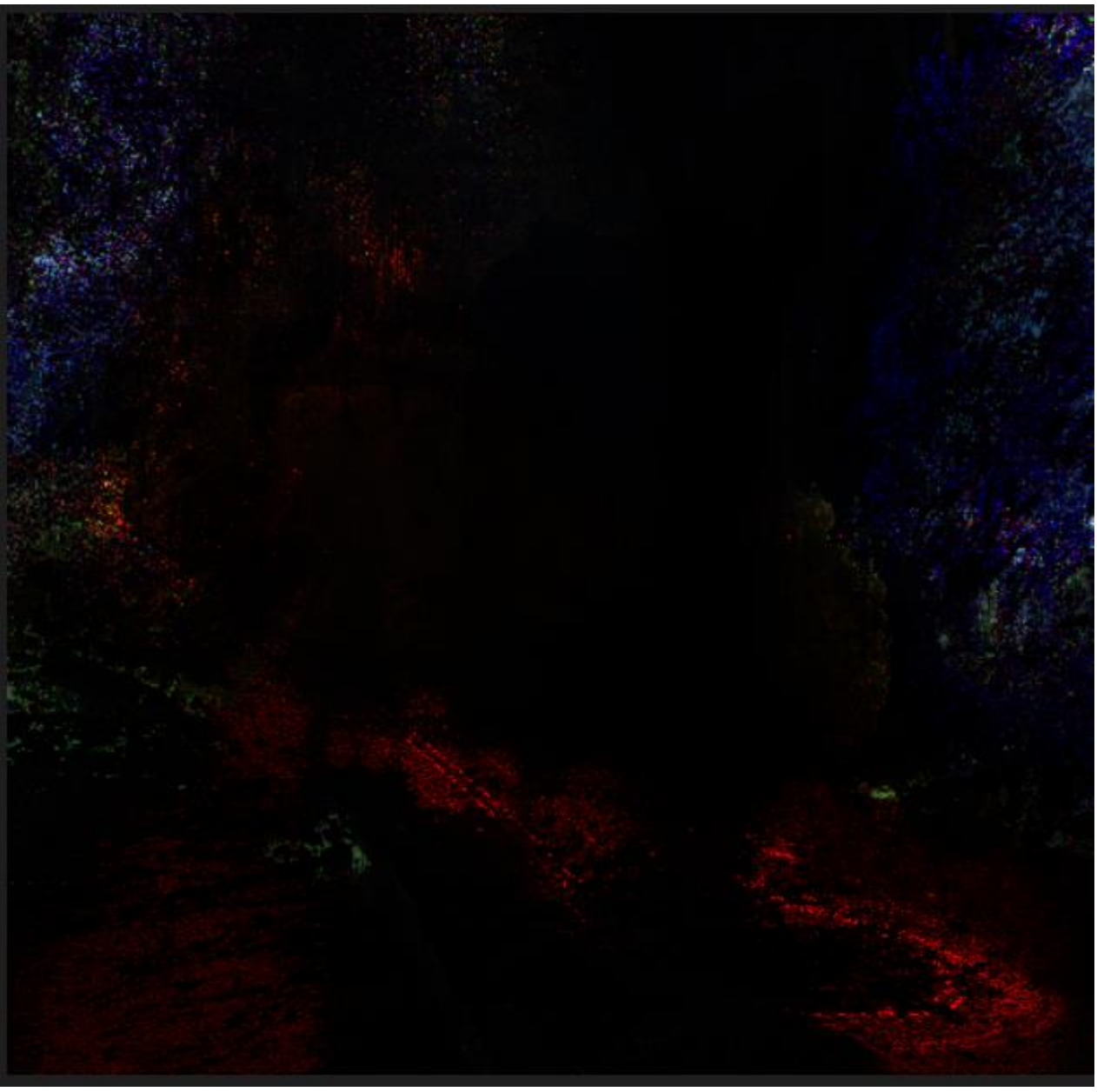
```python
# [a] arithmetic operations on images [ addition, subtraction, multiplication and division]

import cv2
import numpy as np
from google.colab.patches import cv2_imshow

img1 = cv2.imread('/image1.jpg')
img2 = cv2.imread('/image2.jpg')

if img1 is None:
    print("Error: image1.jpg not found or cannot be read.")
    exit()

if img2 is None:
    print("Error: image2.jpg not found or cannot be read.")
    exit()

img1 = cv2.resize(img1, (500, 500))
img2 = cv2.resize(img2, (500, 500))

add_result = cv2.add(img1, img2)
sub_result = cv2.subtract(img1, img2)
mul_result = cv2.multiply(img1, img2, scale=1.0/255)
img2_safe = np.where(img2 == 0, 1, img2)
div_result = cv2.divide(img1, img2_safe)

cv2_imshow(add_result)
cv2_imshow(sub_result)
cv2_imshow(mul_result)
cv2_imshow(div_result)
```

Addition:

Subtraction:

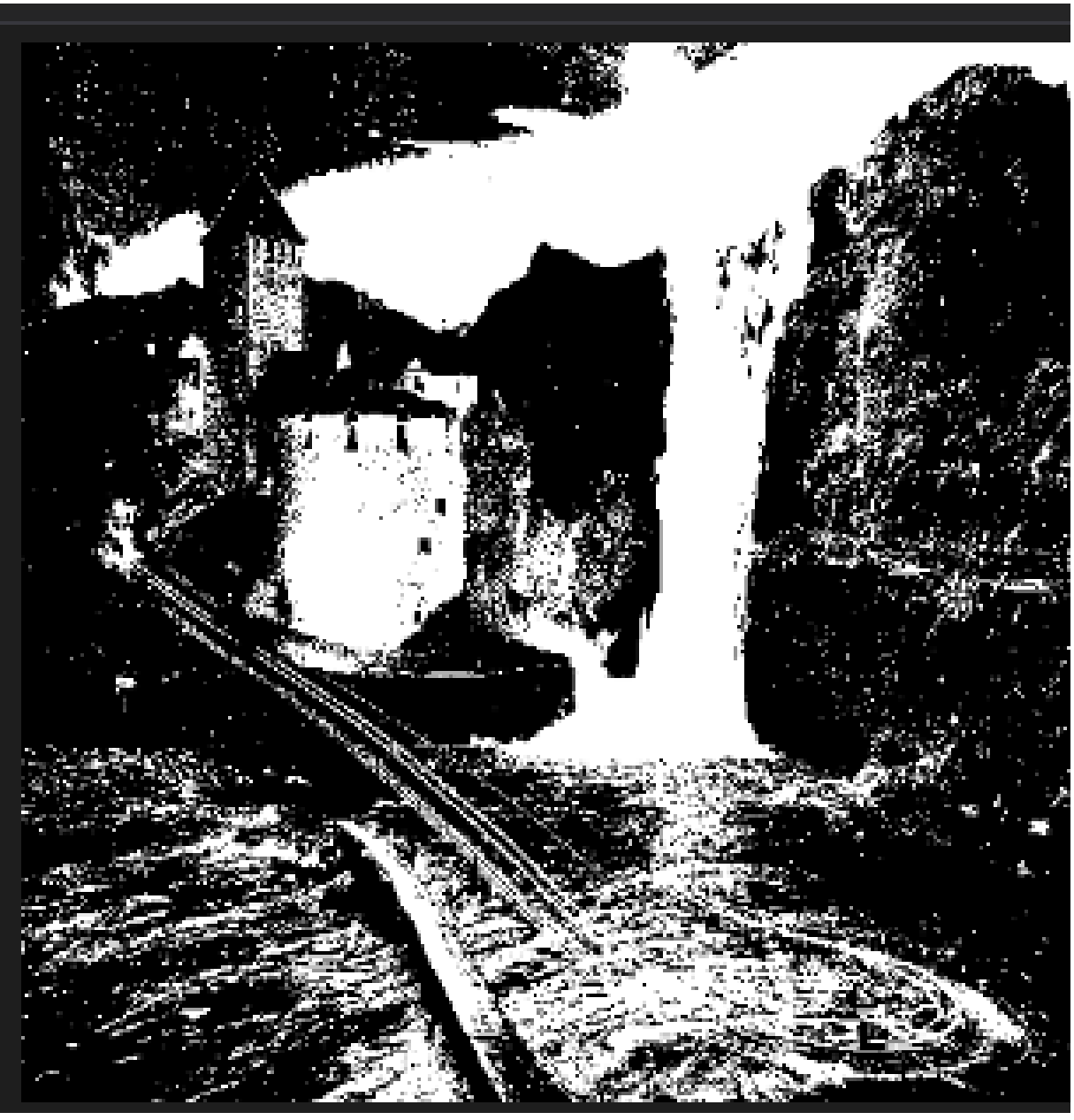
Multiplication:

Division:

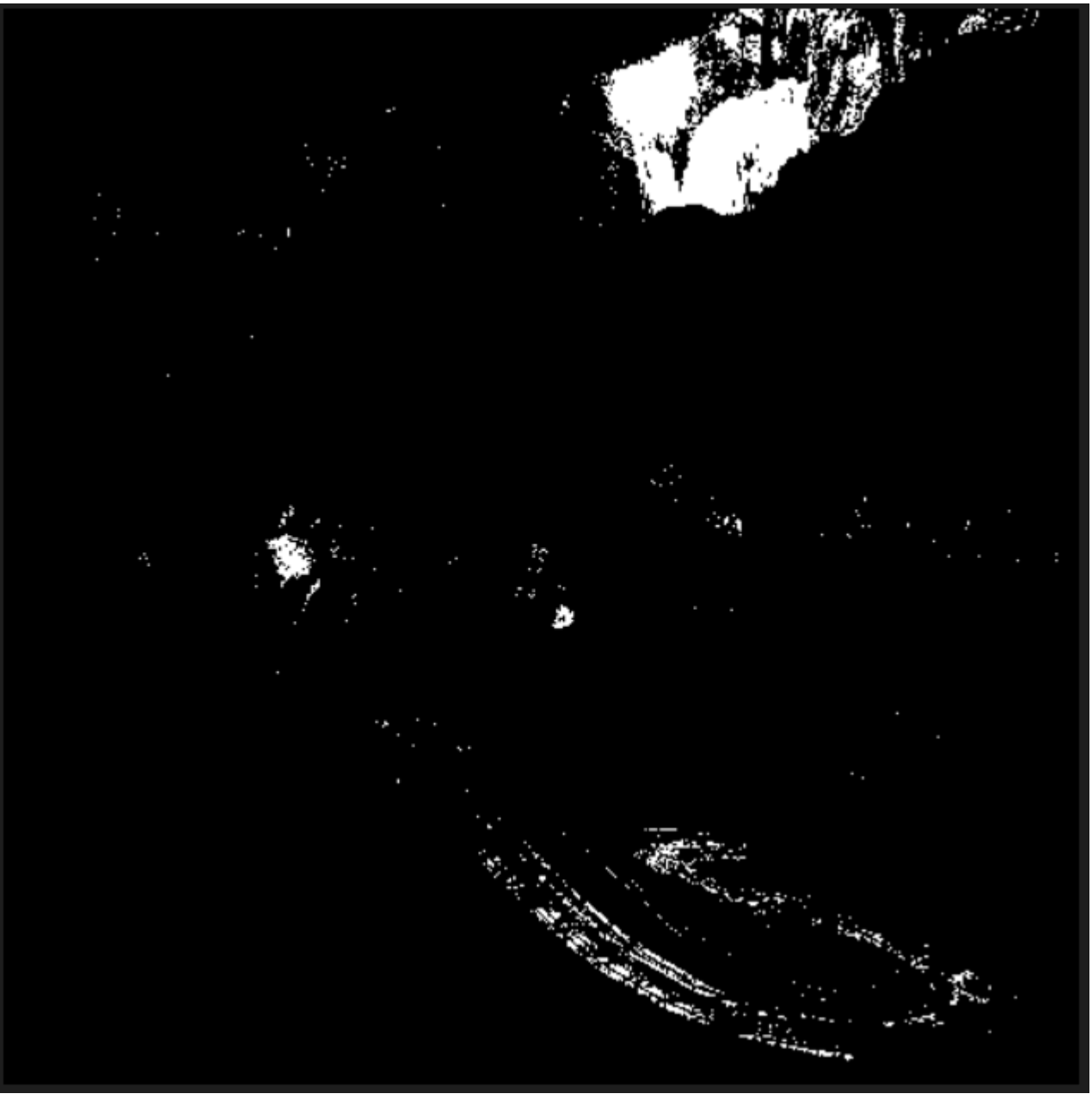


```python
# [b] Set operations on images [union, intersection, negation, XOR]

import cv2
import numpy as np
from google.colab.patches import cv2_imshow

img1 = cv2.imread('/image1.jpg', cv2.IMREAD_GRAYSCALE)
img2 = cv2.imread('/image2.jpg', cv2.IMREAD_GRAYSCALE)

if img1 is None:
    print("Error: image1.jpg not found or cannot be read.")
    exit()
if img2 is None:
    print("Error: image2.jpg not found or cannot be read.")
    exit()

img1 = cv2.resize(img1, (500, 500))
img2 = cv2.resize(img2, (500, 500))

_, img1_bin = cv2.threshold(img1, 128, 255, cv2.THRESH_BINARY)
_, img2_bin = cv2.threshold(img2, 128, 255, cv2.THRESH_BINARY)

union_result = cv2.bitwise_or(img1_bin, img2_bin)
intersection_result = cv2.bitwise_and(img1_bin, img2_bin)
negation_result = cv2.bitwise_not(img1_bin)
xor_result = cv2.bitwise_xor(img1_bin, img2_bin)

cv2_imshow(union_result)
cv2_imshow(intersection_result)
cv2_imshow(negation_result)
cv2_imshow(xor_result)
```

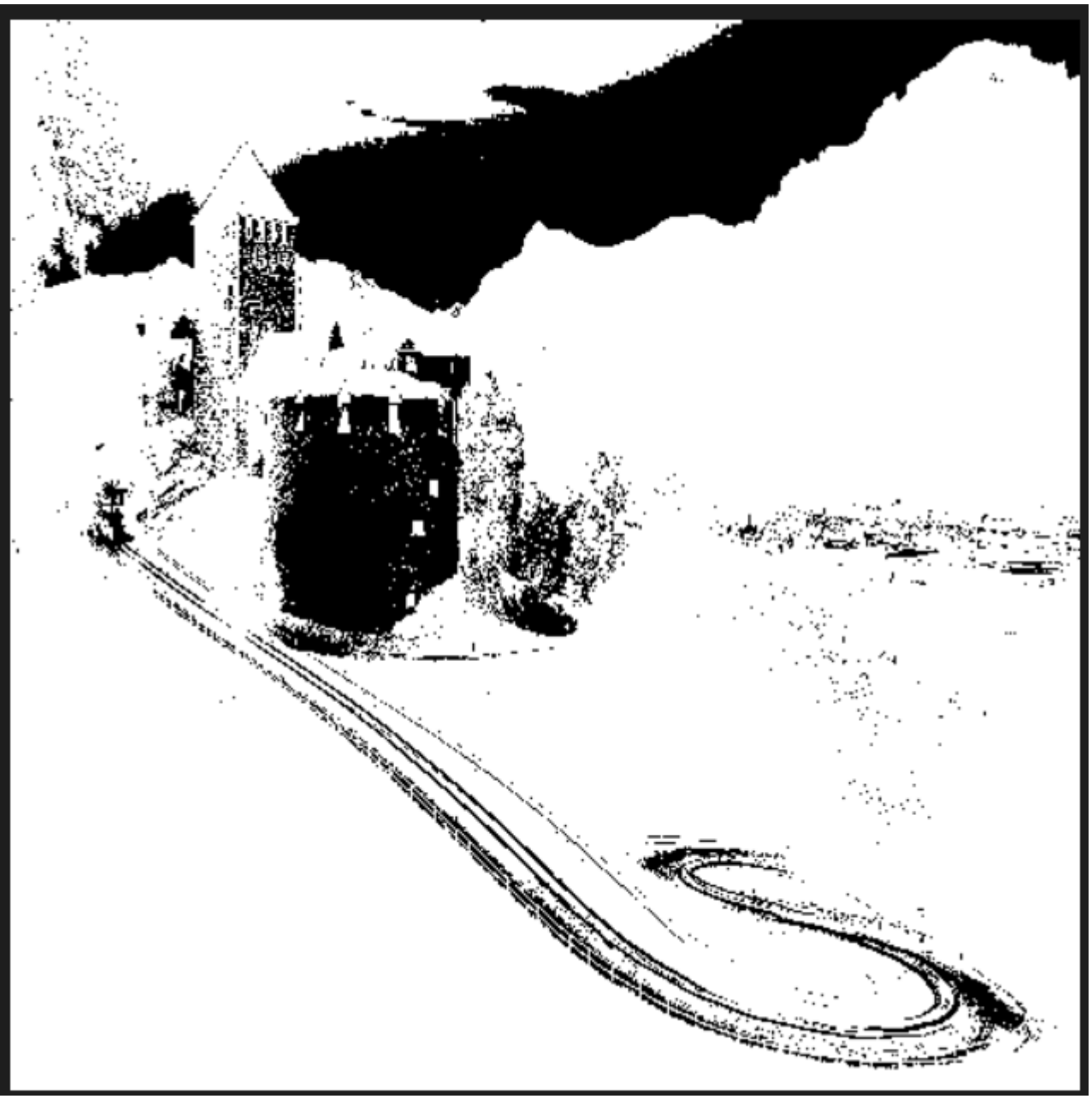
Union:



Intersection:



Negation:



XOR:

```python
# [c] transformation operations [Translation, Rotation and shear]

import cv2
import numpy as np
from google.colab.patches import cv2_imshow

img = cv2.imread('/image1.jpg')

if img is None:
    print("Error: image.jpg not found or cannot be read.")
    exit()

img = cv2.resize(img, (500, 500))
rows, cols, _ = img.shape

tx, ty = 100, 50
translation_matrix = np.float32([[1, 0, tx], [0, 1, ty]])
translated_img = cv2.warpAffine(img, translation_matrix, (cols, rows))

angle = 45
center = (cols // 2, rows // 2)
rotation_matrix = cv2.getRotationMatrix2D(center, angle, 1)
rotated_img = cv2.warpAffine(img, rotation_matrix, (cols, rows))

shear_factor = 0.3
shear_matrix = np.float32([[1, shear_factor, 0], [shear_factor, 1, 0]])
sheared_img = cv2.warpAffine(img, shear_matrix, (cols + int(shear_factor * rows), rows))

cv2_imshow(translated_img)
cv2_imshow(rotated_img)
cv2_imshow(sheared_img)
```

```python
# [d] Histogram equalization

import cv2
import numpy as np
from google.colab.patches import cv2_imshow

img = cv2.imread('/image2.jpg', cv2.IMREAD_GRAYSCALE)

if img is None:
    print("Error: image.jpg not found or cannot be read.")
    exit()

equalized_img = cv2.equalizeHist(img)

cv2_imshow(img)
cv2_imshow(equalized_img)
```