CV Practical - 1

Name: Ritu Pal

Enrollment No.: 230297

Batch: A3

1. Reading, displaying, and writing an image using OpenCV

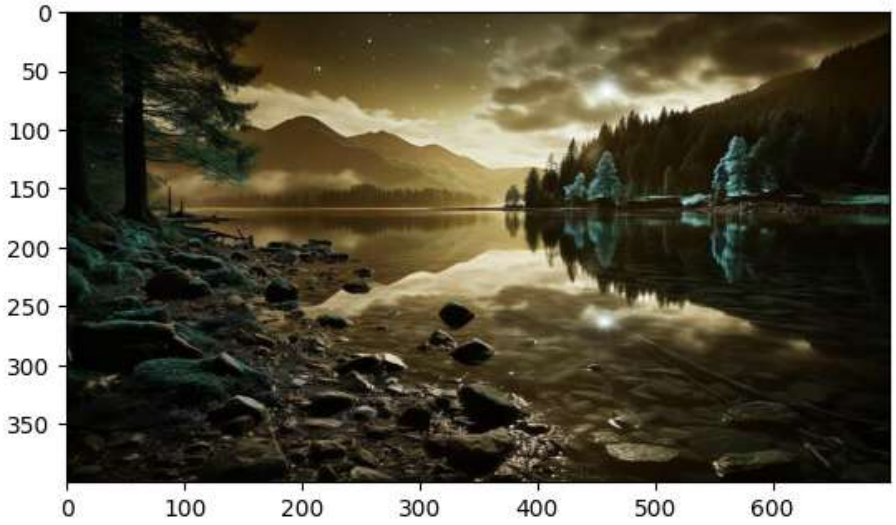
```
import cv2 as cv
from google.colab.patches import cv2_imshow
from google.colab import files

# Upload image
uploaded = files.upload()

# Read uploaded image
img = cv.imread('test2.jpg')

# Show image
cv2_imshow(img)

cv.imwrite("saved_test2.jpg", img)

print("Image saved as saved_test2.jpg")
```

Choose files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving test2.jpg to test2 (3).jpg
```

```
import matplotlib.pyplot as plt
plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x7e7d88ee2850>
```



2. Convert the image in another format using OpenCV

```
image=cv.imread("test2.jpg")
cv.imwrite("output.png",image)
print("Image has been converted and saved as output_image.png")
```

```
Image has been converted and saved as output_image.png
```

3. Perform the image resizing

```
cv.resize(img,(250,250))
```

```
ndarray (250, 250, 3) show data
```

4. Convert a colored image into grayscale using OpenCV

```
gray_img=cv.cvtColor(img, cv.COLOR_BGR2GRAY)
cv2_imshow(gray_img)
```

```
color_img=cv.cvtColor(gray_img, cv.COLOR_GRAY2BGR)
cv2_imshow(color_img)
```

5. Scaling, rotation and shigting operation on the image using OpenCV

```
cv.rotate(img, cv.ROTATE_90_CLOCKWISE)
```

ndarray (700, 400, 3) show data

6. Playing a video

```
from google.colab import files

# Upload video file
uploaded = files.upload()

# Get uploaded filename
for fn in uploaded.keys():
    video_path = fn
print(" Uploaded video:", video_path)
```

Choose files v1.mp4

- **v1.mp4**(video/mp4) - 2553608 bytes, last modified: 19/08/2025 - 100% done

```
Saving v1.mp4 to v1 (1).mp4
 Uploaded video: v1 (1).mp4
```

```
import cv2
from google.colab import files
from IPython.display import HTML
```

```
from base64 import b64encode

# Upload video
uploaded = files.upload()
for fn in uploaded.keys():
    video_path = fn

# Re-encode video to play in Colab
cap = cv2.VideoCapture(video_path)
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
out = cv2.VideoWriter("output.mp4", fourcc, 20.0,
                      (int(cap.get(3)), int(cap.get(4))))

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break
    out.write(frame)

cap.release()
out.release()

# Play video in notebook
mp4 = open("output.mp4",'rb').read()
data_url = "data:video/mp4;base64," + b64encode(mp4).decode()
HTML(f"""
<video width=500 controls>
      <source src="{data_url}" type="video/mp4">
</video>
""")
```

Choose files v1.mp4

- **v1.mp4**(video/mp4) - 2553608 bytes, last modified: 19/08/2025 - 100% done

```
Saving v1.mp4 to v1 (2).mp4
```

0:00

7. Extract images from the video.

```
import cv2
import os
from google.colab import files

# Upload video
uploaded = files.upload()

# Open video
cap = cv2.VideoCapture('v1.mp4')
```

```
if not cap.isOpened():
    print("Error opening video stream or file")
else:
    # Create folder to save frames
    output_folder = 'video_frames'
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    frame_count = 0
    while cap.isOpened():
        ret, frame = cap.read()
        if ret:
            frame_filename = os.path.join(output_folder, f"frame_{frame_count:04d}.jpg")
            cv2.imwrite(frame_filename, frame)
            frame_count += 1
        else:
            break

cap.release()
print(f"Extracted {frame_count} frames to {output_folder}")
```

Choose files v1.mp4

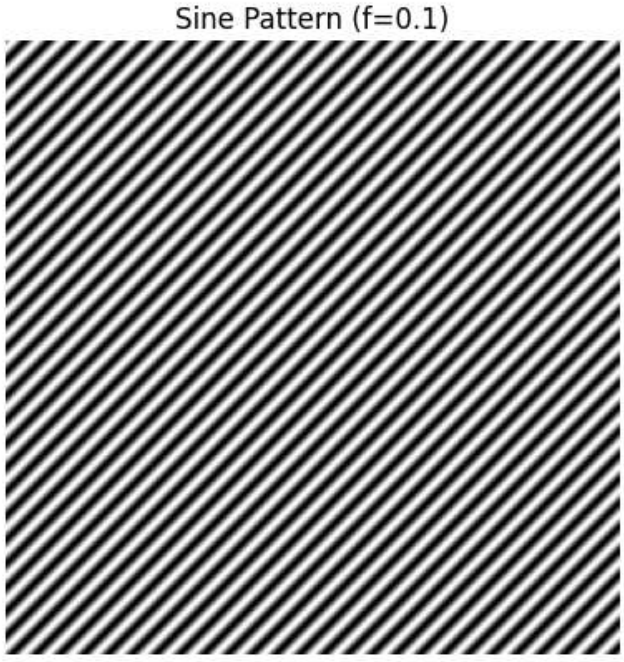
- **v1.mp4**(video/mp4) - 2553608 bytes, last modified: 19/08/2025 - 100% done

```
Saving v1.mp4 to v1 (3).mp4
Extracted 363 frames to video_frames
```

8. Using the following formula f (i, j) = sin (2 π f (i + j)) where i and j indices of a pixel, draw an image with different frequencies (input from the user).

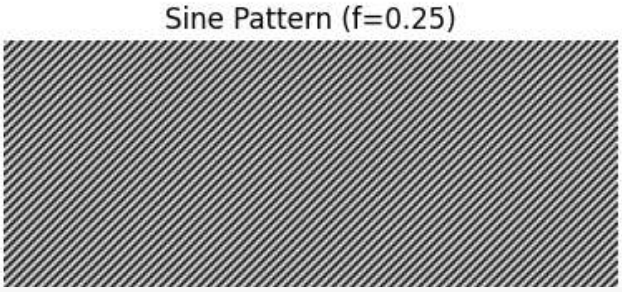
```
import numpy as np
import matplotlib.pyplot as plt

# User input frequency
f = float(input("Enter frequency value: "))

# Image size
rows, cols = 256, 256

# Generate pixel values
img = np.zeros((rows, cols), dtype=np.float32)

for i in range(rows):
    for j in range(cols):
        img[i, j] = np.sin(2 * np.pi * f * (i + j))

# Normalize to [0,255] and convert to uint8
img_norm = cv2.normalize(img, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)

# Show the image
plt.imshow(img_norm, cmap="gray")
plt.title(f"Sine Pattern (f={f})")
plt.axis("off")
plt.show()
```

```
Enter frequency value: 0.1
```



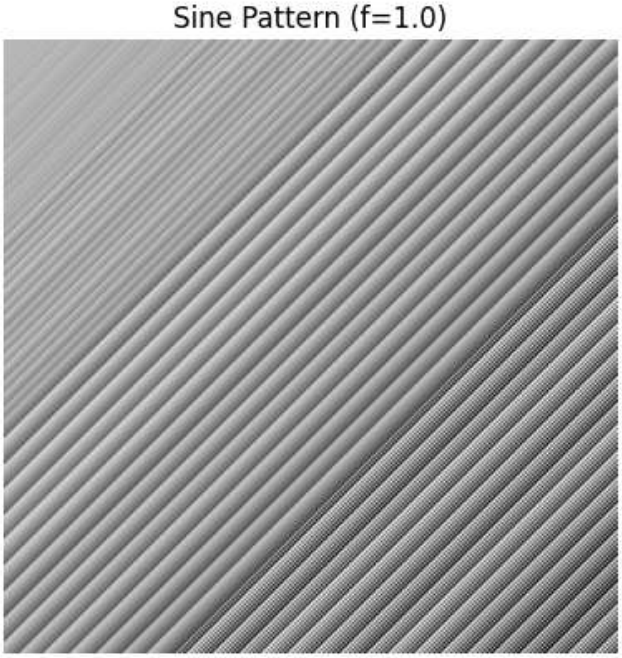

```
import numpy as np
import matplotlib.pyplot as plt

# User input frequency
f = float(input("Enter frequency value: "))

# Image size
rows, cols = 256, 256

# Generate pixel values
img = np.zeros((rows, cols), dtype=np.float32)

for i in range(rows):
    for j in range(cols):
        img[i, j] = np.sin(2 * np.pi * f * (i + j))

# Normalize to [0,255] and convert to uint8
img_norm = cv2.normalize(img, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)

# Show the image
plt.imshow(img_norm, cmap="gray")
plt.title(f"Sine Pattern (f={f})")
plt.axis("off")
plt.show()
```

```
Enter frequency value: 0.25
```




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

# User input frequency
f = float(input("Enter frequency value: "))

# Image size
rows, cols = 256, 256

# Generate pixel values
img = np.zeros((rows, cols), dtype=np.float32)

for i in range(rows):
    for j in range(cols):
        img[i, j] = np.sin(2 * np.pi * f * (i + j))

# Normalize to [0,255] and convert to uint8
img_norm = cv2.normalize(img, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)

# Show the image
plt.imshow(img_norm, cmap="gray")
plt.title(f"Sine Pattern (f={f})")
plt.axis("off")
plt.show()
```

```
Enter frequency value: 1
```



CV Practical - 2

Name: Ritu Pal

Enrollment No.: 230297

Batch: A3

In [ ]:
```python
from google.colab.patches import cv2_imshow
import cv2
import numpy as np

# Upload image
from google.colab import files
uploaded = files.upload()

img = cv2.imread('test2.jpg')

# Scaling
scaled_img = cv2.resize(img, None, fx=1.5, fy=1.5, interpolation=cv2.INTER_LINEAR)

# Rotation
(h, w) = img.shape[:2]
center = (w // 2, h // 2)
rotation_matrix = cv2.getRotationMatrix2D(center, angle=45, scale=1.0)
rotated_img = cv2.warpAffine(img, rotation_matrix, (w, h))

# Shifting
shift_matrix = np.float32([[1, 0, 50], [0, 1, 100]])
shifted_img = cv2.warpAffine(img, shift_matrix, (w, h))
```

Choose files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving test2.jpg to test2 (3).jpg
```

In [ ]:
```python
# Display using cv2_imshow (no need to convert BGR to RGB here)
cv2_imshow(scaled_img)
```

In [ ]:
```
cv2_imshow(rotated_img)
```

In [ ]:
```
cv2_imshow(shifted_img)
```

Image Reflection (Flipping)

In [ ]:
```
# Horizontal reflection
horizontal_flip = cv2.flip(img, 1)

# Vertical reflection
vertical_flip = cv2.flip(img, 0)

# Both directions
both_flip = cv2.flip(img, -1)

# Display
print("Horizontal Flip:")
cv2_imshow(horizontal_flip)

print("Vertical Flip:")
cv2_imshow(vertical_flip)

print("Both Directions Flip:")
cv2_imshow(both_flip)
```

```
Horizontal Flip:
```

Vertical Flip:

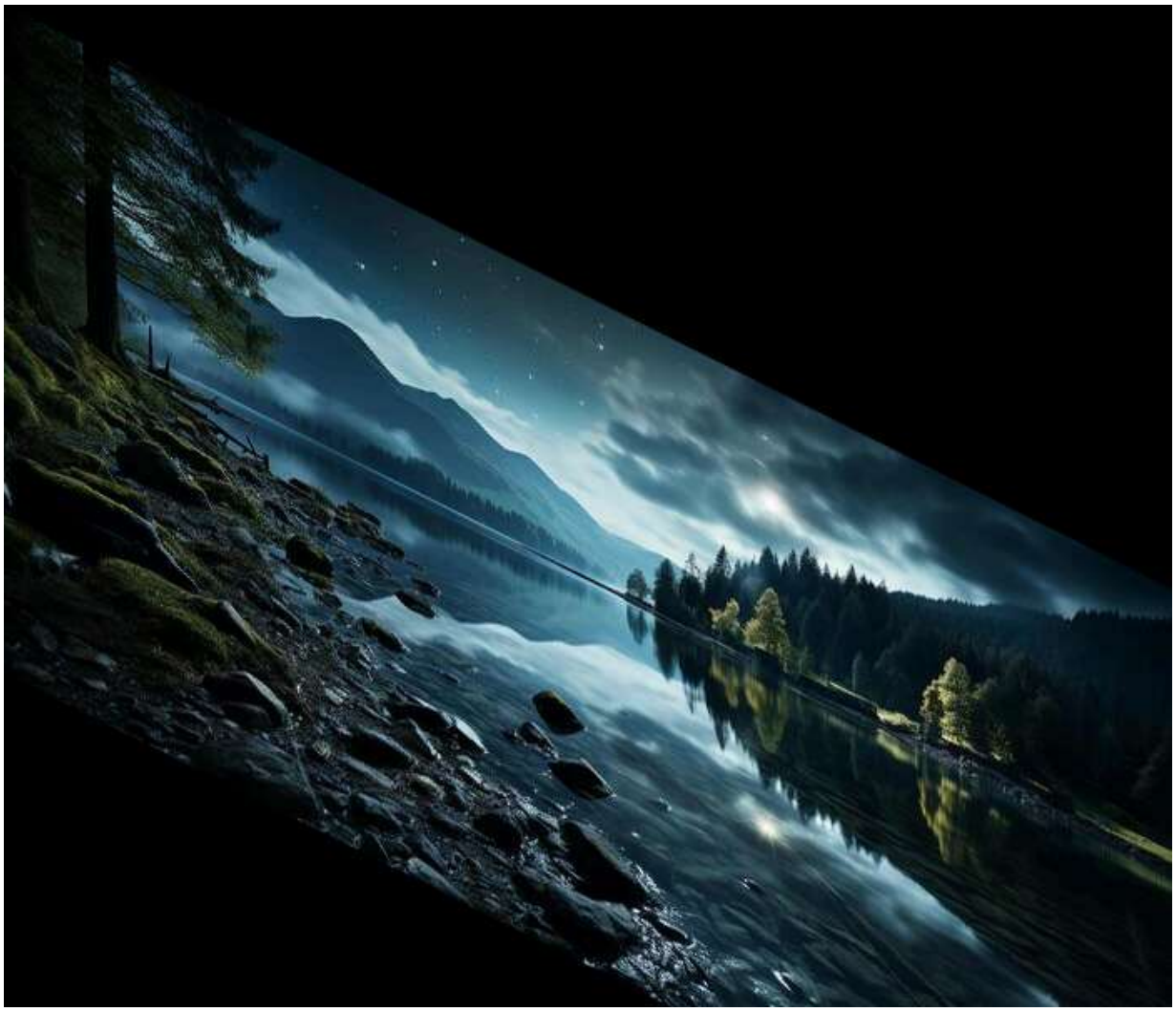
Both Directions Flip:

Image Shearing

In [ ]:
```
# Shearing in X-direction
shear_matrix_x = np.float32([[1, 0.5, 0], [0, 1, 0]])
sheared_img_x = cv2.warpAffine(img, shear_matrix_x, (int(w * 1.5), h))

# Shearing in Y-direction
shear_matrix_y = np.float32([[1, 0, 0], [0.5, 1, 0]])
sheared_img_y = cv2.warpAffine(img, shear_matrix_y, (w, int(h * 1.5)))

# Display
print("Sheared in X-direction:")
cv2_imshow(sheared_img_x)

print("Sheared in Y-direction:")
cv2_imshow(sheared_img_y)
```

```
Sheared in X-direction:
```

```
Sheared in Y-direction:
```



Affine Transformation

In [ ]:
```
# Points from original image
pts1 = np.float32([[50, 50], [200, 50], [50, 200]])

# Corresponding points for affine transform
pts2 = np.float32([[10, 100], [200, 50], [100, 250]])

# Get affine matrix and apply it
affine_matrix = cv2.getAffineTransform(pts1, pts2)
affine_transformed = cv2.warpAffine(img, affine_matrix, (w, h))

# Display
print("Affine Transformed Image:")
cv2_imshow(affine_transformed)
```

```
Affine Transformed Image:
```

Perfomance On Second Image

In [ ]:
```
from google.colab import files
uploaded = files.upload()
```

Choose files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving car1.jpeg to car1 (5).jpeg
```

In [ ]:
```
import cv2
from google.colab.patches import cv2_imshow

img = cv2.imread("car1.jpeg")  # Make sure the name matches exactly
cv2_imshow(img)
```

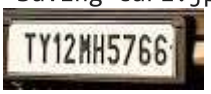
In [ ]:
```
print(img.shape)  # (height, width, channels)
```

```
(213, 237, 3)
```

In [ ]:
```
from google.colab import files
import cv2
import numpy as np
from google.colab.patches import cv2_imshow

# ===== Step 1: Upload Image =====
uploaded = files.upload()
filename = list(uploaded.keys())[0]
img = cv2.imread(filename)

# ===== Step 2: Crop Number Plate (adjusted for 213x237 image) =====
# Coordinates within range
y1, y2 = 110, 150   # vertical range
x1, x2 = 80, 180   # horizontal range
number_plate = img[y1:y2, x1:x2]

cv2_imshow(number_plate)
print("Original Number Plate")
```

Choose files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving car1.jpeg to car1 (6).jpeg
```




```
Original Number Plate
```

In [ ]:
```
print(img.shape)  # (height, width, channels)
```

```
(213, 237, 3)
```

Performance On Third Image

In [11]:
```
from google.colab import files
import cv2
import numpy as np
from google.colab.patches import cv2_imshow

# ===== Step 1: Upload Image =====
uploaded = files.upload()
filename = list(uploaded.keys())[0]
img = cv2.imread(filename)

cv2_imshow(img)
print("Original Image")

# ===== Step 2: Rotation of full image =====
(h, w) = img.shape[:2]
center = (w // 2, h // 2)
rotation_angle = -25  # Change this for different rotation
rotation_matrix = cv2.getRotationMatrix2D(center, rotation_angle, 1.0)
rotated_img = cv2.warpAffine(img, rotation_matrix, (w, h))

cv2_imshow(rotated_img)
print(f"Rotated full image by {rotation_angle} degrees")
```

```
# ===== Step 3: Crop Number Plate (adjust coordinates after rotation) =====
y1, y2 = 110, 150  # vertical range
x1, x2 = 70, 190   # horizontal range
number_plate = rotated_img[y1:y2, x1:x2]

cv2_imshow(number_plate)
print("Cropped Number Plate after rotation")
```

Choose files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving car2.jpeg to car2 (10).jpeg
```


```
Original Image
```

Rotated full image by -25 degrees

Cropped Number Plate after rotation

CV Practical - 3

Name: Ritu Pal

Enrollment No.: 230297

Batch: A3

3. Arithmetic and Bitwise Operations:Addition, Subtraction, Multiplication, Division: useful for tasks like image blending, background subtraction, and image enhancement.Bitwise Operations (AND, OR, NOT, XOR): useful for masking, extracting specific features and creating composite images.

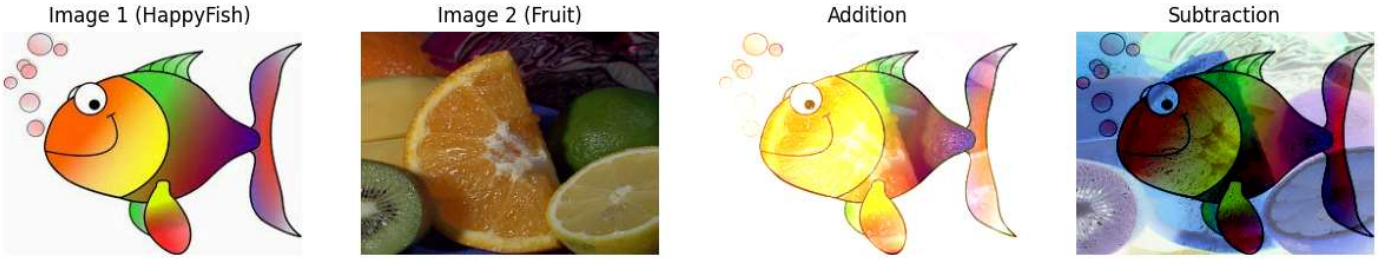
```
import cv2
import numpy as np
from google.colab import files
from matplotlib import pyplot as plt

# Upload two images at once
uploaded = files.upload()

# List of uploaded files
filenames = list(uploaded.keys())
if len(filenames) < 2:
    print("Please upload two images!")
else:
    img1_path = filenames[0]
    img2_path = filenames[1]
    img1 = cv2.imread(img1_path)
    img2 = cv2.imread(img2_path)

    # Resize second image to match first
    img2 = cv2.resize(img2, (img1.shape[1], img1.shape[0]))

    # Arithmetic Operations
    add = cv2.add(img1, img2)
    sub = cv2.subtract(img1, img2)
    mul = cv2.multiply(img1, img2)
    div = cv2.divide(img1.astype(float), img2.astype(float) + 1)
    div = np.clip(div, 0, 255).astype(np.uint8)

    # Bitwise Operations
    gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
    gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
    _, mask1 = cv2.threshold(gray1, 127, 255, cv2.THRESH_BINARY)
    _, mask2 = cv2.threshold(gray2, 127, 255, cv2.THRESH_BINARY)

    bitwise_and = cv2.bitwise_and(mask1, mask2)
    bitwise_or = cv2.bitwise_or(mask1, mask2)
    bitwise_not = cv2.bitwise_not(mask1)
    bitwise_xor = cv2.bitwise_xor(mask1, mask2)

    # Show function
    def show_images(images, titles):
        plt.figure(figsize=(15, 10))
        for i, (img, title) in enumerate(zip(images, titles)):
            plt.subplot(2, 4, i + 1)
            if len(img.shape) == 3:
                img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
                plt.imshow(img)
            else:
                plt.imshow(img, cmap='gray')
            plt.title(title)
            plt.axis('off')
        plt.show()

    show_images(
        [img1, img2, add, sub, mul, div, bitwise_and, bitwise_not],
        ['Image 1 (HappyFish) ', 'Image 2 (Fruit)', 'Addition', 'Subtraction',
         'Multiplication', 'Division', 'Bitwise AND', 'Bitwise NOT']
    )
```

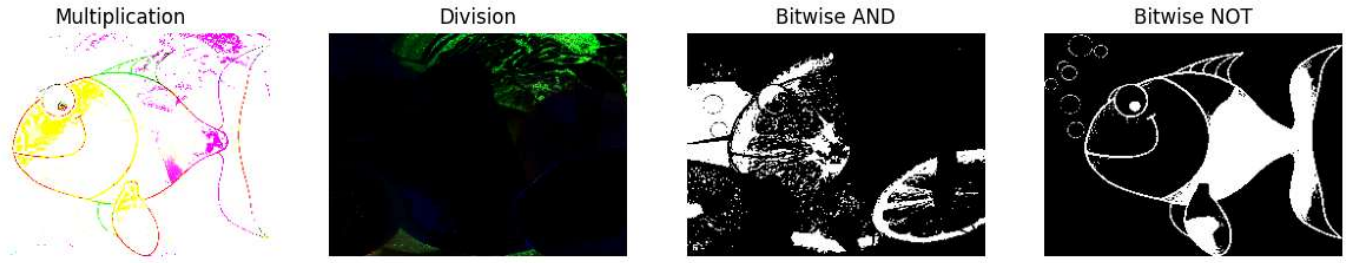
,

Choose files 2 files

- **HappyFish.jpg**(image/jpeg) - 8283 bytes, last modified: 19/08/2025 - 100% done
- **fruits.jpg**(image/jpeg) - 82429 bytes, last modified: 19/08/2025 - 100% done

```
Saving HappyFish.jpg to HappyFish (2).jpg
Saving fruits.jpg to fruits (1).jpg
```





1. Compute the Homography matrix for a given 4 data points without SVD and transformed the point using the computed homography matrix. (51,791) → (1,900 (63,143) → (1,1) (444,211) → (501,1) (426,719) → (501,900)

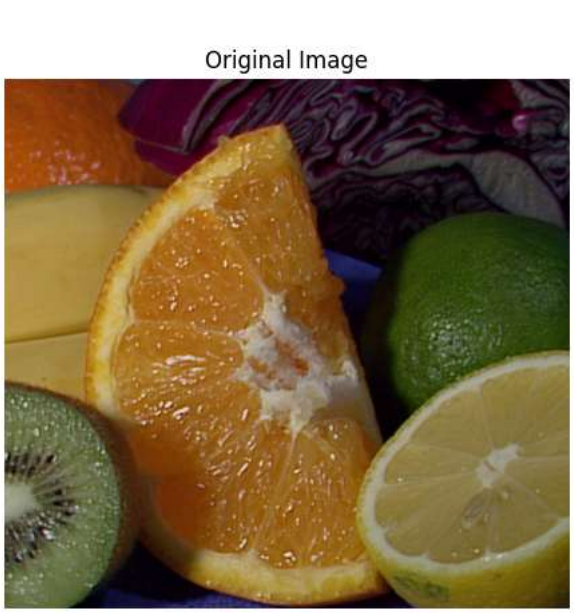
```
import numpy as np
import cv2
from matplotlib import pyplot as plt
from google.colab import files

# Upload image
print("Upload your source image:")
uploaded = files.upload()

# Get uploaded image filename
img_name = list(uploaded.keys())[0]
print(f"Uploaded image: {img_name}")

# Load the uploaded image
img_src = cv2.imread(img_name)
if img_src is None:
    raise FileNotFoundError(f"Image file '{img_name}' not found or cannot be loaded.")

# Your point correspondences
pts_src = np.array([
    [51, 791],
    [63, 143],
    [444, 211],
    [426, 719]
], dtype=np.float64)

pts_dst = np.array([
    [1, 900],
```

```
    [1, 1],
    [501, 1],
    [501, 900]
], dtype=np.float64)

# Construct matrix A and vector b for Ah = b
A = []
b = []

for (x, y), (xp, yp) in zip(pts_src, pts_dst):
    A.append([x, y, 1, 0, 0, 0, -xp*x, -xp*y])
    b.append(xp)
    A.append([0, 0, 0, x, y, 1, -yp*x, -yp*y])
    b.append(yp)

A = np.array(A)
b = np.array(b)

# Solve for h (8 unknowns)
h = np.linalg.solve(A, b)

# Append 1 to get homography matrix
H = np.append(h, 1).reshape(3, 3)
print("Computed homography matrix H:")
print(H)

# Warp the source image using homography
width = int(max(pts_dst[:, 0]))
height = int(max(pts_dst[:, 1]))

img_warped = cv2.warpPerspective(img_src, H, (width, height))

# Show original and warped images
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(cv2.cvtColor(img_src, cv2.COLOR_BGR2RGB))
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title('Warped Image')
plt.imshow(cv2.cvtColor(img_warped, cv2.COLOR_BGR2RGB))
plt.axis('off')

plt.show()
```

```
Upload your source image:
```
Choose files fruits.jpg

- **fruits.jpg**(image/jpeg) - 82429 bytes, last modified: 19/08/2025 - 100% done

```
Saving fruits.jpg to fruits (3).jpg
Uploaded image: fruits (3).jpg
Computed homography matrix H:
[[ 9.79081952e-01  1.80888635e-02 -6.33104064e+01]
 [-2.30322178e-01  1.28740037e+00 -1.68629492e+02]
 [-5.40599557e-04 -5.22948563e-05  1.00000000e+00]]
```



2. Compute the Homography matrix for a given 4 data points using DLT and transformed the point using the computed homography matrix. (51,791) → (1,900) (63,143) → (1,1) (444,211) → (501,1) (426,719) → (501,900)

```
# Step 1: Upload image
from google.colab import files
uploaded = files.upload()  # Upload your image here

# Step 2: Import necessary libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Step 3: Load the uploaded image
img_name = list(uploaded.keys())[0]
img_src = cv2.imread(img_name)

if img_src is None:
    raise ValueError(f"Failed to load image '{img_name}'. Check the file and path.")

# Step 4: Define source and destination points
pts_src = np.array([
    [51, 791],
    [63, 143],
    [444, 211],
    [426, 719]
], dtype=np.float64)

pts_dst = np.array([
    [1, 900],
    [1, 1],
    [501, 1],
    [501, 900]
], dtype=np.float64)

# Step 5: Define the DLT function to compute homography
def compute_homography_dlt(src_pts, dst_pts):
    n = src_pts.shape[0]
```

```
    # n = src_pts.shape[0]
    A = []
    for i in range(n):
        x, y = src_pts[i]
        xp, yp = dst_pts[i]
        A.append([-x, -y, -1, 0, 0, 0, xp*x, xp*y, xp])
        A.append([0, 0, 0, -x, -y, -1, yp*x, yp*y, yp])
    A = np.array(A)
    U, S, Vt = np.linalg.svd(A)
    h = Vt[-1, :]
    H = h.reshape(3, 3)
    H /= H[2, 2]
    return H

# Step 6: Compute homography matrix
H = compute_homography_dlt(pts_src, pts_dst)
print("Computed Homography matrix H:\n", H)

# Step 7: Warp the image using the homography
width = int(max(pts_dst[:, 0]))
height = int(max(pts_dst[:, 1]))
img_warped = cv2.warpPerspective(img_src, H, (width, height))

# Step 8: Display the original and warped images
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.title("Original Image")
plt.imshow(cv2.cvtColor(img_src, cv2.COLOR_BGR2RGB))
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title("Warped Image")
plt.imshow(cv2.cvtColor(img_warped, cv2.COLOR_BGR2RGB))
plt.axis('off')

plt.show()
```

Choose files fruits.jpg

- **fruits.jpg**(image/jpeg) - 82429 bytes, last modified: 19/08/2025 - 100% done

```
Saving fruits.jpg to fruits (4).jpg
Computed Homography matrix H:
 [[ 9.79081952e-01  1.80888635e-02 -6.33104064e+01]
 [-2.30322178e-01  1.28740037e+00 -1.68629492e+02]
 [-5.40599557e-04 -5.22948563e-05  1.00000000e+00]]
```