****

**Lab Manual**

**Computer Engineering – Artificial Intelligence**

**B. Tech. Year – II, 5th Semester, Academic Year (2023)**

**Subject Code: 01AI0504**

*Subject Name: Digital Image Processing*

*Name: Basid Al Siddik Shammo*

*Enrollment Number: 92100151052*

**Aim-** Study the of Relationships between Pixels and transformation of an image

**Description-**

**Pixel Relationships in an Image:**

* An image consists of a grid of pixels, where each pixel represents a small portion of the image's content. The arrangement of pixels creates the visual content that we perceive as an image.
* In digital images, pixels are typically represented by values that correspond to colors or grayscale intensities. In color images, each pixel may have three color channels (red, green, and blue), while grayscale images have only one channel representing the intensity.

**Spatial Relationships:**

* Spatial relationships refer to the arrangement of pixels with respect to their positions in the image. Understanding these relationships is crucial for various image processing tasks.
* Common spatial relationships include neighboring pixels, which are pixels located next to each other horizontally or vertically. These relationships are used in tasks like edge detection and noise reduction.

**Transformation of an Image:**

* Image transformations are operations applied to an image to alter its appearance or content. These transformations can be both geometric and non-geometric in nature.
* Geometric transformations involve changing the spatial arrangement of pixels in the image. Common geometric transformations include scaling, rotation, translation, and shearing.
* Non-geometric transformations involve changes in pixel values without altering the spatial arrangement. Common non-geometric transformations include contrast adjustments, brightness adjustments, and color space conversions.

**Affine Transformations:**

* Affine transformations are a class of geometric transformations that preserve parallel lines and ratios of distances between points. These transformations include translation, rotation, scaling, and shearing.
* Affine transformations are widely used in computer vision for tasks like image registration, image alignment, and object detection.

**Perspective Transformations:**

* Perspective transformations are geometric transformations that involve the distortion of the image to simulate changes in perspective or viewpoint.
* Perspective transformations are often used in computer graphics and augmented reality applications to render virtual objects realistically into real-world scenes.

**Image Registration:**

* Image registration is the process of aligning two or more images of the same scene taken at different times, from different viewpoints, or by different sensors. It involves finding the spatial relationships between the pixels in the images to align them properly.
* Image registration is crucial in various applications, including medical imaging, remote sensing, and panoramic image stitching.

**Task- I**

**Code-**

import cv2

import numpy as np

def display\_neighbors(image):

    four\_point\_kernel = np.array([[0, 1, 0], [1, 0, 1], [0, 1, 0]])

    eight\_point\_kernel = np.ones((3, 3), dtype=np.uint8)

    diagonal\_kernel = np.array([[1, 0, 1], [0, 0, 0], [1, 0, 1]])

    neighbors\_4 = cv2.filter2D(image, -1, four\_point\_kernel)

    neighbors\_8 = cv2.filter2D(image, -1, eight\_point\_kernel)

    neighbors\_diag = cv2.filter2D(image, -1, diagonal\_kernel)

    neighbors\_combined = np.hstack((neighbors\_4, neighbors\_8, neighbors\_diag))

    cv2.imshow("Original Image", image)

    cv2.imshow("Filtered Neighbors", neighbors\_combined)

    cv2.waitKey(0)

    cv2.destroyAllWindows()

    #img = cv2.imread('architecture1.jpg')

img = cv2.imread('architecture1.jpg', cv2.IMREAD\_GRAYSCALE)

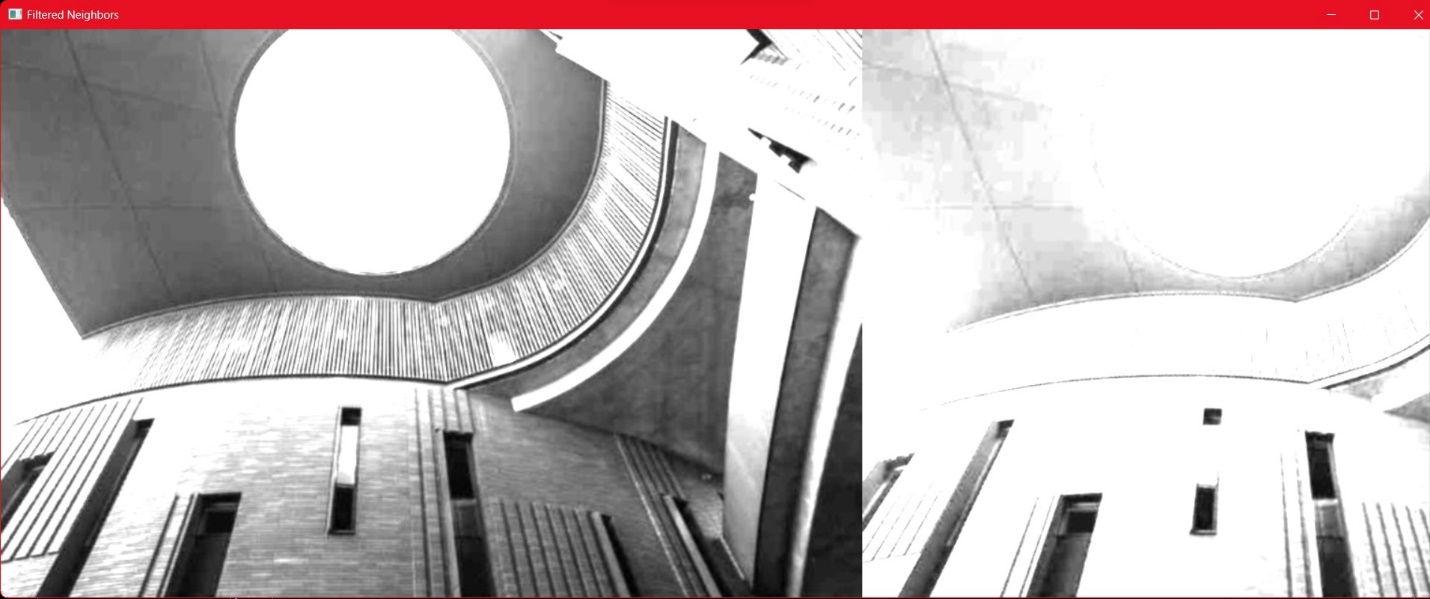
if img is not None:

display\_neighbors(img)

**Result (Original Image)-**

****

**Result (Filtered Neighbors)-**

****

**Task- II**

**Code-**

import cv2

import numpy as np

def scale\_and\_rotate\_image(img\_path, scaling\_factors, rotation\_angles):

    img = cv2.imread(img\_path)

    height, width = img.shape[:2]

    scaled\_images = []

    for scale in scaling\_factors:

        new\_width = int(width \* scale)

        new\_height = int(height \* scale)

        scaled\_image = cv2.resize(img, (new\_width, new\_height))

        scaled\_images.append(scaled\_image)

        rotated\_images = []

        for angle in rotation\_angles:

            for scaled\_image in scaled\_images:

                center = (scaled\_image.shape[1] // 2, scaled\_image.shape[0] // 2)

                rotation\_matrix = cv2.getRotationMatrix2D(center, angle, 1.0)

                rotated\_image = cv2.warpAffine(scaled\_image, rotation\_matrix, (scaled\_image.shape[1], scaled\_image.shape[0]))

                rotated\_images.append(rotated\_image)

        return rotated\_images

img\_path = 'architecture1.jpg'

scaling\_factors = [0.5, 2, 3]

rotation\_angles = [45, 60]

result\_images = scale\_and\_rotate\_image(img\_path, scaling\_factors, rotation\_angles)

cv2.imshow('Original image', cv2.imread(img\_path))

for i, result\_image in enumerate(result\_images):

    cv2.imshow(f'Result {i+1}', result\_image)

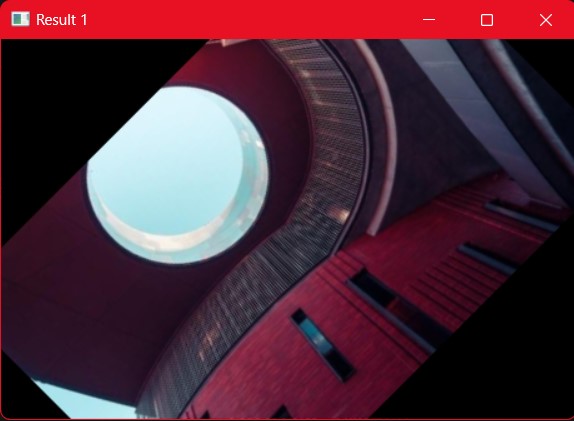
cv2.waitKey(0)

cv2.destroyAllWindows()

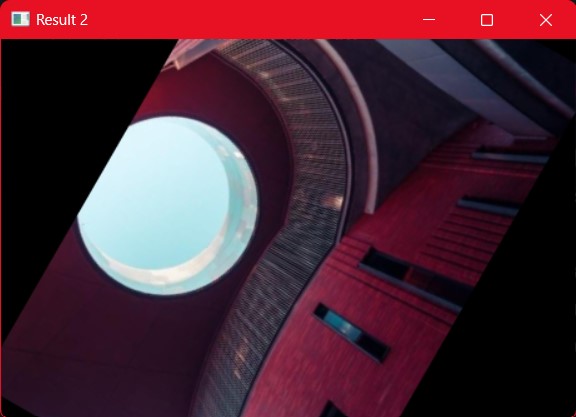
**Result (Original Image)-**



**Result 1-**

****

**Result 2-**

****

**Conclusion-**