**Department of Electrical Engineering and   
Computer Science**

**Faculty Member:** Dr. Mohsin Kamal **Dated:** 19/10/2023

**Semester:** 7th **Section:** BEE 12C

**CS-477 Computer Vision**

Lab 6: Pixel Connectivity and Noise Reduction Filters

**Group Members**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | **PLO4 - CLO4** | | **PLO5 -CLO5** | **PLO8 -CLO6** | **PLO9 -CLO7** |
| **Name** | **Reg. No** | **Viva / Quiz / Lab Performance** | **Analysis of Data in Lab Report** | **Modern Tool Usage** | **Ethics and Safety** | **Individual and Teamwork** |
|  |  | **5 Marks** | **5 Marks** | **5 Marks** | **5 Marks** | **5 Marks** |
| Muhammad Ahmed Mohsin | 333060 |  |  |  |  |  |
| Muhammad Umer | 345834 |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

**Table of Contents**

[2 Pixel Connectivity and Noise Reduction Filters 3](#_Toc148728328)

[2.1 Introduction 3](#_Toc148728329)

[2.2 Objectives 3](#_Toc148728330)

[2.3 Software 3](#_Toc148728331)

[3 Lab Tasks 4](#_Toc148728332)

[3.1 Task 1 4](#_Toc148728333)

[3.2 Task 2 6](#_Toc148728334)

[3.3 Task 3 9](#_Toc148728335)

[4 Conclusion 13](#_Toc148728336)

# Pixel Connectivity and Noise Reduction Filters

## Introduction

In this lab, we will explore three important image processing techniques: connected component labeling, arithmetic mean filtering, and order statistic filtering. These techniques are widely used in a variety of applications, such as medical imaging, computer vision, and remote sensing.

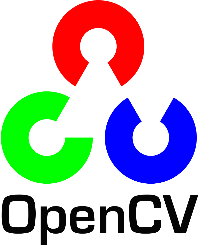
## Objectives

The objective of this lab is:

* To perform connected component labeling of a binary image.
* Learn how to implement the arithmetic mean filter, as well as some of its variations, such as the contra-harmonic mean, the harmonic mean, and the geometric mean filters.
* Learn how to perform order statistic filtering, including median, min, max, midpoint, and alpha-trimmed mean filters!

## Software

OpenCV is a library that focuses on image processing and computer vision. An image is an array of colored squares called pixels. Each pixel has a certain location in the array and color values in BGR format. By referring to the array indices, the individual pixels or a range of pixels can be accessed and modified. OpenCV provides many functions for resizing, rotating, and placing objects in images. Rotation involves computing a 2-D rotation matrix which is applied for the transformation of the image.



# Lab Tasks

## Task 1

1. Read “cc.png” from directory.
2. Apply connected component labelling using 4 connectivity and count total number of objects in the list. (**which means write your own function from scratch!**)

HINT1: The image is a binary image with background (black portion) has numeric value of 1 while the white objects have numeric values of 255

HINT2: You can use **two-pass algorithm** from the Algorithm 1 (which doesn’t use a specialized data structure and simply use two for loops!)

1. Use OpenCV’s **cv2.connectedComponents** function and admire the underlying workings.

### TASK 1 CODE STARTS HERE ###

img = cv2.imread("lab6\_cc.png", cv2.IMREAD\_GRAYSCALE)

img = img / 255

binary = img.astype(np.uint8)

rows, cols = binary.shape

labels = np.zeros((rows, cols), *dtype*=np.uint16)

*# Equivalence table*

equivalence = {}

current\_label = 1

for i in range(1, rows):

    for j in range(1, cols):

        if binary[i, j] == 0:

            continue

        label\_neighbors = [labels[i - 1, j], labels[i, j - 1]]

*# Assign new label*

        if label\_neighbors[0] == 0 and label\_neighbors[1] == 0:

            labels[i, j] = current\_label

            current\_label += 1

*# Assign label from neighbor*

        else:

            if label\_neighbors[0] == 0:

                labels[i, j] = label\_neighbors[1]

            elif label\_neighbors[1] == 0:

                labels[i, j] = label\_neighbors[0]

            else:

                labels[i, j] = label\_neighbors[1]  *# Assign the left label*

*# Add to equivalence table*

                if label\_neighbors[0] != label\_neighbors[1]:

                    equivalence[label\_neighbors[0]] = label\_neighbors[1]

*def* get\_equivalent\_label(*label*, *equivalence*=equivalence):

    while label not in equivalence or equivalence[label] != label:

        if label not in equivalence:

            equivalence[label] = label

        label = equivalence[label]

    return label

*# Second pass to apply final equivalent labels*

for i in range(rows):

    for j in range(cols):

        if labels[i, j] != 0:

            labels[i, j] = get\_equivalent\_label(labels[i, j])

*# OpenCV's connectedComponents*

\_, labels\_opencv = cv2.connectedComponents(binary, *connectivity*=4)

plt.rcParams["font.family"] = "STIXGeneral"

fig, ax = plt.subplots(1, 2, *figsize*=(10, 10))

ax[0].imshow(

    labels \* 1200, *cmap*="nipy\_spectral"

)  *# Multiply by 1200 to make the colors more visible*

ax[0].set\_title("Custom Connected Components")

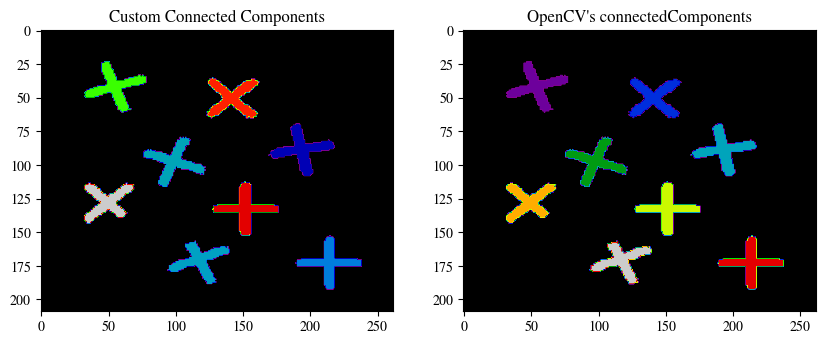
ax[1].imshow(labels\_opencv, *cmap*="nipy\_spectral")

ax[1].set\_title("OpenCV's connectedComponents")

plt.show()

### TASK 1 CODE ENDS HERE ###

### TASK 1 SCREENSHOT STARTS HERE ###



### TASK 1 SCREENSHOT ENDS HERE ###

## Task 2

Corrupt the input images with different types of noise models such as:

1. ‘Gaussian’
2. ‘Poisson'
3. ‘Salt & Pepper’
4. ‘Speckle’
5. ‘Salt-only’
6. ‘Pepper-only’

### TASK 2 CODE STARTS HERE ###

from skimage.util import random\_noise

img = cv2.imread("lab6\_pears.png", cv2.IMREAD\_GRAYSCALE)

gaussian = random\_noise(img, *mode*="gaussian", *var*=0.01)

poisson = random\_noise(img, *mode*="poisson")

salt\_pepper = random\_noise(img, *mode*="s&p", *amount*=0.1)

speckle = random\_noise(img, *mode*="speckle")

salt = random\_noise(img, *mode*="salt", *amount*=0.1)

pepper = random\_noise(img, *mode*="pepper", *amount*=0.1)

*# Stix*

plt.rcParams["font.family"] = "STIXGeneral"

for i, noise in zip(

    ["Gaussian", "Poisson", "Salt & Pepper", "Speckle", "Salt", "Pepper"],

    [gaussian, poisson, salt\_pepper, speckle, salt, pepper],

):

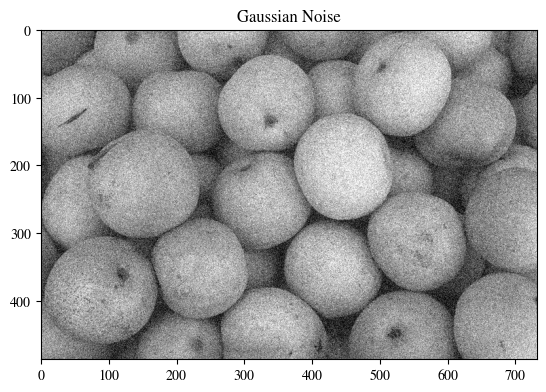
    plt.imshow(noise, *cmap*="gray")

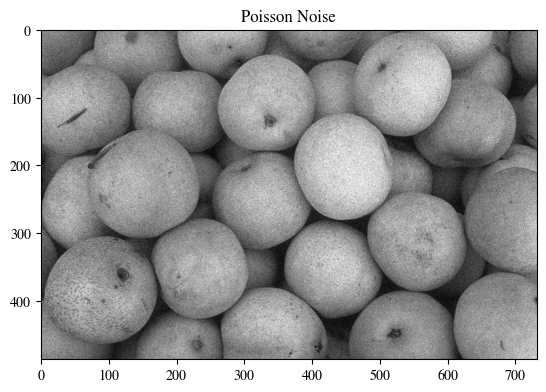
    plt.title(*f*"{i} Noise")

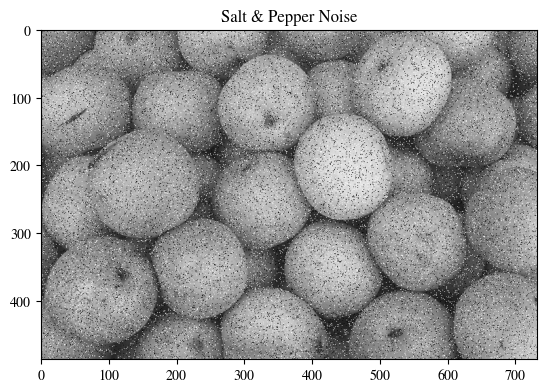
    plt.show()

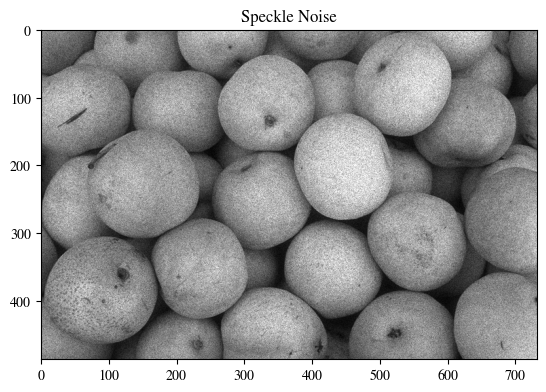
### TASK 2 CODE ENDS HERE ###

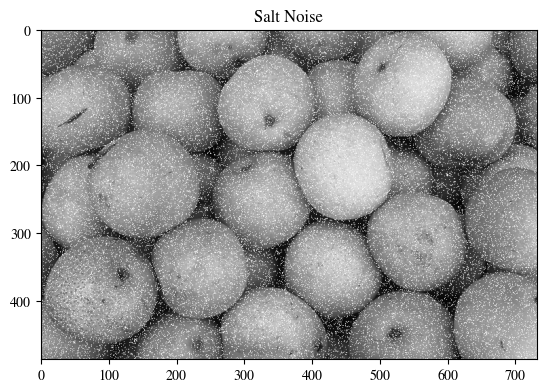
### TASK 2 SCREENSHOT STARTS HERE ###

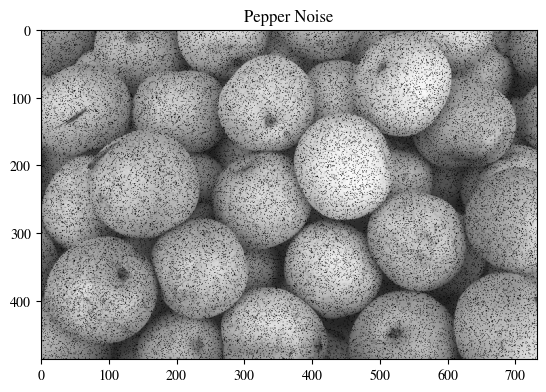












### TASK 2 SCREENSHOT ENDS HERE ###

## Task 3

Apply different kinds of noise removal filters as given below.

1. Arithmetic mean
2. Geometric mean
3. Harmonic mean
4. Contra harmonic mean, the contra harmonic mean filter is used for filtering an image with either salt or pepper noise (but not both).
5. Max filters
6. Min filters

### TASK 3 CODE STARTS HERE ###

import scipy.ndimage as ndimage

img = cv2.imread("lab6\_pears.png", cv2.IMREAD\_GRAYSCALE)

gaussian = random\_noise(img, *mode*="gaussian", *var*=0.01)

poisson = random\_noise(img, *mode*="poisson")

salt\_pepper = random\_noise(img, *mode*="s&p", *amount*=0.1)

speckle = random\_noise(img, *mode*="speckle")

salt = random\_noise(img, *mode*="salt", *amount*=0.1)

pepper = random\_noise(img, *mode*="pepper", *amount*=0.1)

*# Arithmetic mean*

arithmetic\_mean = ndimage.uniform\_filter(gaussian, *size*=3)

*# Geometric mean*

kernel\_size = 3

geometric\_mean = cv2.pow(

    cv2.GaussianBlur(gaussian, (kernel\_size, kernel\_size), 0), 1.0 / kernel\_size

)

*# Harmonic mean*

*def* harmonic\_mean(*img*, *size*):

    num = img.size

    denom = 1 / (img + np.finfo(*float*).eps)

    kernel = np.ones(size)

    result = num / cv2.filter2D(denom, -1, kernel)

    return result

harmonic\_mean = harmonic\_mean(gaussian, (3, 3))

*# Contra harmonic mean*

*def* contraharmonic\_mean(*img*, *size*, *Q*):

    num = np.power(img, Q + 1)

    denom = np.power(img, Q)

    kernel = np.ones(size)

    result = cv2.filter2D(num, -1, kernel) / cv2.filter2D(denom, -1, kernel)

    return result

window\_size = (3, 3)

contra\_harmonic\_salt = contraharmonic\_mean(salt, window\_size, -1)

contra\_harmonic\_pepper = contraharmonic\_mean(pepper, window\_size, 1)

*# Max filter*

max\_filter = ndimage.maximum\_filter(pepper, *size*=3)

*# Min filter*

min\_filter = ndimage.minimum\_filter(salt, *size*=3)

### TASK 3 CODE ENDS HERE ###

### TASK 3 SCREENSHOT STARTS HERE ###

*# Gaussian and Arithmetic, Geometric, and Harmonic mean*

fig, ax = plt.subplots(2, 2, *figsize*=(10, 7))

ax[0, 0].imshow(gaussian, *cmap*="gray")

ax[0, 0].set\_title("Gaussian Noise")

ax[0, 1].imshow(arithmetic\_mean, *cmap*="gray")

ax[0, 1].set\_title("Arithmetic Mean")

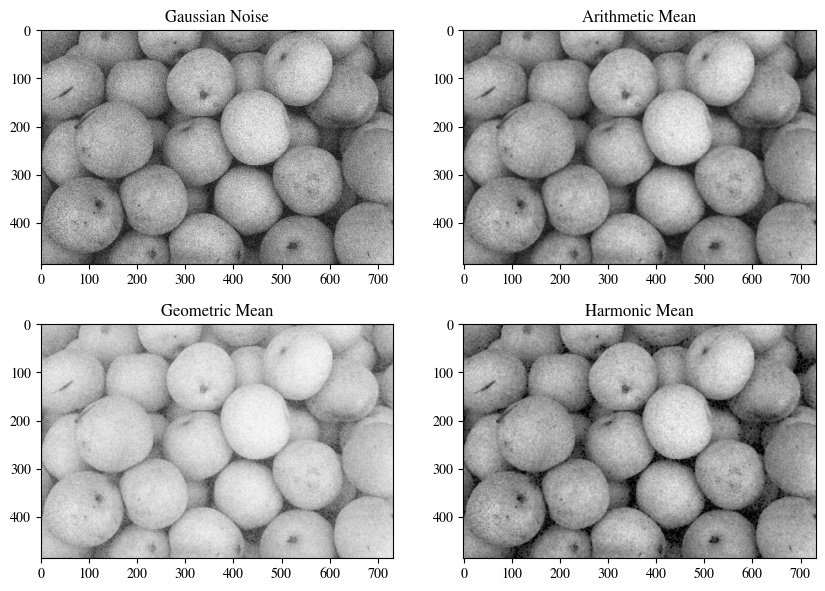
ax[1, 0].imshow(geometric\_mean, *cmap*="gray")

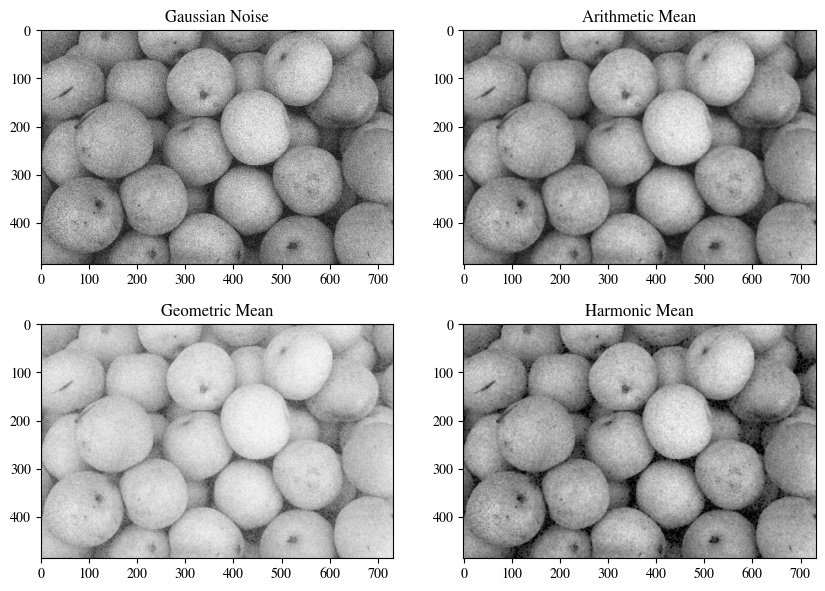
ax[1, 0].set\_title("Geometric Mean")

ax[1, 1].imshow(harmonic\_mean, *cmap*="gray")

ax[1, 1].set\_title("Harmonic Mean")

plt.show()





*# Salt and Contra Harmonic*

fig, ax = plt.subplots(1, 2, *figsize*=(10, 10))

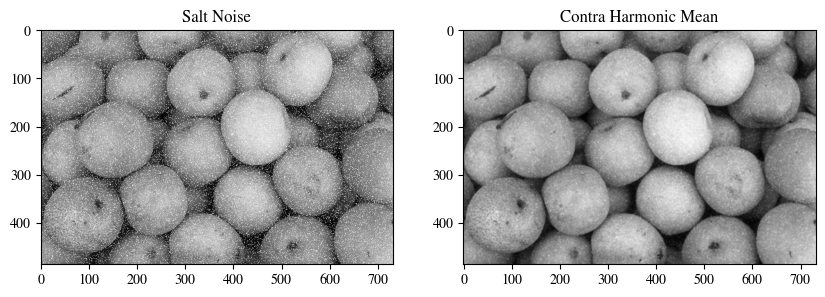
ax[0].imshow(salt, *cmap*="gray")

ax[0].set\_title("Salt Noise")

ax[1].imshow(contra\_harmonic\_salt, *cmap*="gray")

ax[1].set\_title("Contra Harmonic Mean")

plt.show()



*# Pepper and Contra Harmonic*

fig, ax = plt.subplots(1, 2, *figsize*=(10, 10))

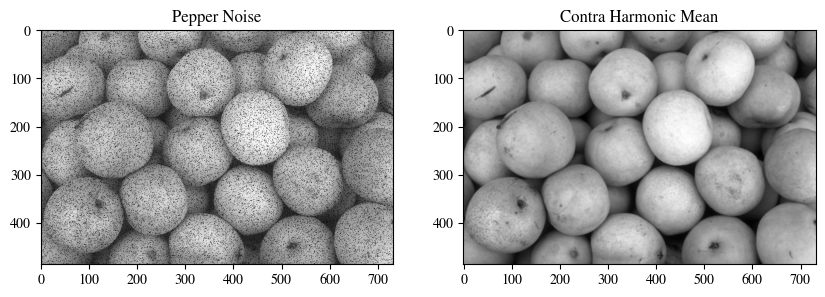
ax[0].imshow(pepper, *cmap*="gray")

ax[0].set\_title("Pepper Noise")

ax[1].imshow(contra\_harmonic\_pepper, *cmap*="gray")

ax[1].set\_title("Contra Harmonic Mean")

plt.show()



*# Salt and Max*

fig, ax = plt.subplots(1, 2, *figsize*=(10, 10))

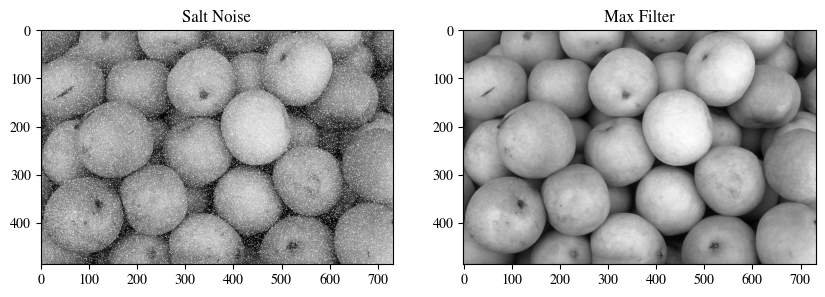
ax[0].imshow(salt, *cmap*="gray")

ax[0].set\_title("Salt Noise")

ax[1].imshow(max\_filter, *cmap*="gray")

ax[1].set\_title("Max Filter")

plt.show()



*# Pepper and Min*

fig, ax = plt.subplots(1, 2, *figsize*=(10, 10))

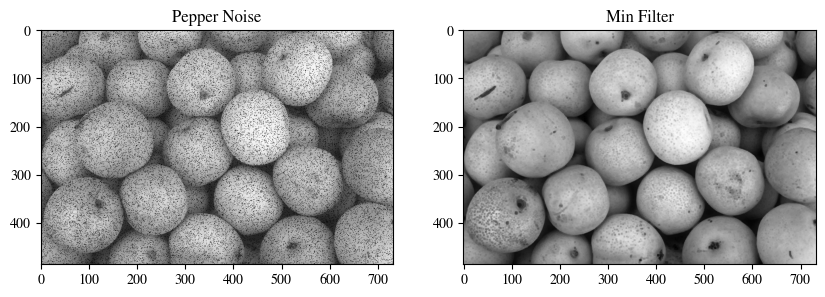
ax[0].imshow(pepper, *cmap*="gray")

ax[0].set\_title("Pepper Noise")

ax[1].imshow(min\_filter, *cmap*="gray")

ax[1].set\_title("Min Filter")

plt.show()



### TASK 3 SCREENSHOT ENDS HERE ###

# Conclusion

In this lab, we learned how to perform connected component labeling, implement the arithmetic mean filter with its variants, and perform order statistic filtering. These techniques are useful for image processing and analysis tasks such as image segmentation, object recognition, and noise reduction. We also learned about the different parameters that can be used to control these filters. By adjusting these parameters, we can tailor the filtering process to meet the specific needs of our application.