**Assignment 2**

CP467: Image Processing & Recognition

Professor: Dr. Zia Ud Din

Due Date: Friday October 20, 2023. 11:59 p.m.

Chandler Mayberry 190688910

Table of Contents

[Introduction 3](#_Toc148728600)

[Task 1 3](#_Toc148728601)

[Averaging Smoothing Filter (Filter size: 3\*3) 3](#_Toc148728602)

[Gaussian Smoothing Filter (Filter size: 7\*7, sigma=1, mean=0) 4](#_Toc148728603)

[Sobel Sharpening Filter (Filter size: 3\*3, both horizontal and vertical) 4](#_Toc148728604)

[Task 2 5](#_Toc148728605)

[Averaging Smoothing Filter Built-in (Same as Task 1) 5](#_Toc148728606)

[Gaussian Smoothing Filter Built-in (Same as Task 1) 5](#_Toc148728607)

[Sobel Sharpening Filter Built-in (Same as Task 1) 6](#_Toc148728608)

[Comparison of Task 1 and Task 2 Results 6](#_Toc148728609)

[Task 3 9](#_Toc148728610)

[Marr-Hildreth Edge Detector 9](#_Toc148728611)

[Canny Edge Detector 10](#_Toc148728612)

[Compare Results of Edge Detectors 10](#_Toc148728613)

[Task 4 11](#_Toc148728614)

[Group Adjacent Pixels 11](#_Toc148728615)

[Comparison of Grouped Adjacent Pixel Results: 14](#_Toc148728616)

# Introduction

All the code for this assignment may be run using the a02\_190688910.py file in the source code folder. Running this file will perform the operations and display the resulting images to the user using python OpenCV.imshow() method.

# Task 1

## Averaging Smoothing Filter (Filter size: 3\*3)

An averaging smoothing filter is a simple image blurring technique to reduce the sharpness and bridge the smaller details of an image. This operation is rather simple, the new value for each pixel in the image is the average of its surrounding neighborhood (adjacent pixel values). In the case of the operation implemented below, the filter size specified was 3\*3. See the code below:

A computer screen shot of a program code

Description automatically generated

Figure 1: Averaging Smoothing Filter Implementation

## Gaussian Smoothing Filter (Filter size: 7\*7, sigma=1, mean=0)

A Gaussian smoothing filter is another smoothing technique that reduces the noise within an image and produces a blurred image. This was introduced as a preprocess to edge detection in an image as it reduces the amount of noise (large value differences in neighboring pixels). The logic is relatively straight forward, for a 7\*7 region with sigma value one and a mean of zero a normalized height map matrix is created as seen on slide 23-24, Lecture 5. This height map essentially gradients the values from lighter to darker from the center to edges of each pixel matrix (size 7\*7). This height map is then applied to all the pixels and adjacent neighbors in the matrix, causing smoother transitions between gray values in the resulting image. See the code below:

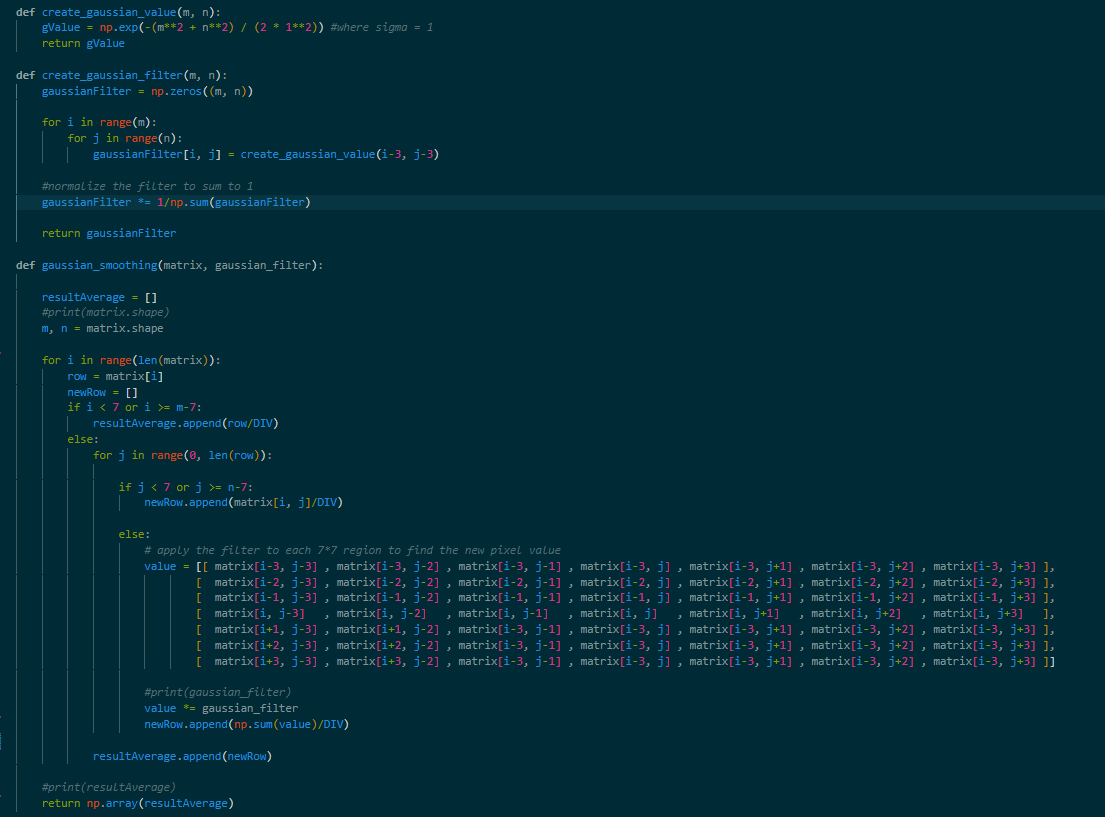


Figure 2: Gaussian Smoothing Filter Implementation

## Sobel Sharpening Filter (Filter size: 3\*3, both horizontal and vertical)

The general purpose of the Sobel sharpening filter is to bring out finer detail within the image. This operation is simple, the Sobel filter is predefined both vertically and horizontally in a 3\*3 matrix. Both the vertical and horizontal matrices are applied at each pixel and surrounding neighbors independently. In this operation it is required that both vertical and horizontal Sobel filtering is used, this combination is done by taking the magnitude of each value across both arrays. See the code below:

A computer screen shot of text

Description automatically generated

Figure 3: Sobel Sharpening Filter Implementation

# Task 2

## Averaging Smoothing Filter Built-in (Same as Task 1)

Averaging smoothing filter using built-in function. See the code below:



Figure 4: Averaging Smoothing Filter Built-in

## Gaussian Smoothing Filter Built-in (Same as Task 1)

Gaussian smoothing filter using built-in function. See the code below:



Figure 5: Gaussian Smoothing Filter Built-in

## Sobel Sharpening Filter Built-in (Same as Task 1)

Sobel sharpening filter using built-in functions. See the code below:

A screen shot of a computer code

Description automatically generated

Figure 6: Sobel Sharpening Filter Built-in

# Comparison of Task 1 and Task 2 Results

Comparing the results of the built-in functions to those that I implemented, it is extremely clear that the built-in operations are significantly faster than the scratch implementation of the algorithms. Either than that, it seems the images produced for Gaussian and Averaging smoothing were nearly identical to the built-ins from visual inspection. The only difference I see are the results of the Sobel sharpening filter, where the built-in does a significantly better job than that of the scratch implementation. This is perhaps due to my combination of values after calculating two separate matrix values for horizontal and vertical, where the built-in may do something cleverer to combine the images equally in cv2.addWeighted. Please see results below and in the folder included to see enlarged results:

A person using a camera

Description automatically generated

Figure 7: Original Cameraman

A person using a camera

Description automatically generated

Figure 8: Averaging Smoothing Filter Result

A person standing next to a camera

Description automatically generated

Figure 9: Gaussian Smoothing Filter Result

A person using a camera

Description automatically generated

Figure 10: Sobel Sharpening Filter Result

A person using a camera

Description automatically generated

Figure 11: Averaging Smoothing Built-in

A person using a camera

Description automatically generated

Figure 12: Gaussian Smoothing Built-in

A person using a tripod

Description automatically generated

Figure 13: Sobel Sharpening Built-in

# Task 3

## Marr-Hildreth Edge Detector

Marr-Hildreth edge detector is an edge detection algorithm that results in a black and white image, where the white values indicate the edges of objects in an image. Marr-Hildreth was implemented with built-ins as follows from the slides; first smooth the image using a gaussian filter, then apply Laplacian and combine. Then normalize into 255 range, get the zero crossings, and apply a threshold within bounds 0-255 to get the result. The threshold I finalized was 40-255 as it produced the best edge detection result. See the code and result below:

A computer screen shot of text

Description automatically generated

Figure 14: Marr-Hildreth Edge Detector Built-ins

A screen shot of a person with a camera

Description automatically generated

Figure 15: Marr-Hildreth Result

## Canny Edge Detector

Canny edge detector is also an edge detection algorithm that produces similar results to Marr-Hildreth. This is the most universally used edge detector in image processing as well as computer vision as stated in Lecture 5. The threshold I found to be the best was between 120-180, where the camera man and his camera are outlined without too much detail. See the code and result below:



Figure 16: Canny Built-in

A person with a camera on a tripod

Description automatically generated

Figure 17: Canny Result

## Compare Results of Edge Detectors

As seen above, the results for the Marr-Hildreth implementation using built-ins are far worse than the result of the Canny built-in. This is perhaps due to an incorrect implementation of Marr-Hildreth or truly representing the significant improvement using Canny edge detection is. I found that no matter the threshold values used in Marr-Hildreth the image would result with a lot of noise in terms of random white pixels and poor localization of edges (keeping straight lines). The higher the threshold resulted in more exaggerated edges and less detail in the image for Marr-Hildreth. In comparison, Canny results in a nice edge map where the edges are well defined with a lot less noise regardless of a specific threshold.

# Task 4

## Group Adjacent Pixels

Group adjacent pixels is a simple concept that involves filling in color and/or labeling pixels based on their grouping between edges. For example, think of the bucket tool within Microsoft paint on each group of pixels in an image.

**Note:**

I noticed when I ran the function it seemed that the edge maps produced by both my Canny and Marr-Hildreth implementations ended up with gaps between edges causing large grouping of pixels between objects in the image (such as the cameraman and the foreground). Therefore, I wanted to increase the size of the edges to better show the grouping and [applied a dilation function](https://docs.opencv.org/3.4/d4/d86/group__imgproc__filter.html) to the final result of Canny and Marr-Hildreth. See the code and results below:

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

Figure 18: Group Adjacent Pixels Implementation

A screen shot of a video camera

Description automatically generated

Figure 19: Marr-Hildreth Grouped Result



Figure 20: Marr-Hildreth Dilated Grouped Result

A person with a camera on a tripod

Description automatically generated

Figure 21: Canny Grouped Result

A person using a camera

Description automatically generated

Figure 22: Canny Dilated Grouped Result

## Comparison of Grouped Adjacent Pixel Results:

With both the dilated and un-dilated edge maps, Canny produces more connected regions than Marr-Hildreth. My Marr-Hildreth implementation had a lot of issues with random pixel placement and poor localization of edges as mentioned above. Canny edge detection also produced more image details in observation.