Image Processing Lab

Spatial Filtering

GROUP-10

Submitted by:

Aditya Narayan (13EC35010)

Adarsh Kumar Kosta (13EC35008)

Objective

Write C/C++ modular functions/subroutines to design spatial filters - mean, median, gradient, Laplacian, Sobel kernels (horizontal, vertical, diagonal) on a stack of grayscale images (say, 15 images per stack).

Use OpenCV (or) ImageJ for image reading, writing and GUI development only. Use the OpenCV tracker (slider) functionality to show outputs for varying sizes of neighborhoods.

You may have different sliders to select:
(i) Image
(ii) Filter
(iii) Neighborhood size

(a) Input: Path to the stack of images. Input stack should contain the (provided) noisy images, and may also contain the normal test images, e.g. jetplane.jpg, lake.jpg, livingroom.jpg, mandril_gray.jpg, pirate.jpg, walkbridge.jpg

(b) Output: Filtered stack of images should be shown beside input stack in the same pane of GUI with a slider to vary filter/kernel size/change image.

Introduction

A spatial filter is an image operation where each pixel value I(u, v) is changed by a function of the intensities of pixels in a neighborhood of (u, v).

Let **$H : R_H \rightarrow [0, K-1]$**

$$I'(u,\ v)\ =\ \sum_{(i,j)\varepsilon R_H} I(u+i,\ \ v+j)\ .\ H(i,\ j)$$

This is known as a correlation of **I** and **H.**

H is the filter “kernel” or “matrix”.

Notice that the kernel H is just a small image!

Depending on which filter is being used, the kernel H changes.

A few examples of the various kernels used and emplyed in this experiment are explained below.

Mean Filter

Mean filtering is a simple, intuitive and easy to implement method of smoothing images, i.e. reducing the amount of intensity variation between one pixel and the next. It is often used to reduce noise in images.

The idea of mean filtering is simply to replace each pixel value in an image with the mean (`average') value of its neighbors, including itself. This has the effect of eliminating pixel values which are unrepresentative of their surroundings.

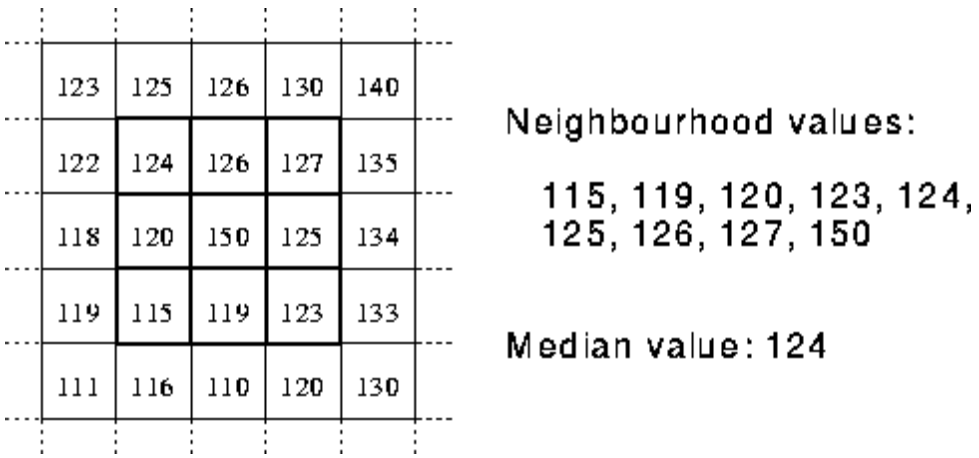
Mean filtering is usually thought of as a convolution filter. Like other convolutions it is based around a kernel, which represents the shape and size of the neighborhood to be sampled when calculating the mean. Often a 3×3 square kernel is used, as shown in Figure 1, although larger kernels (e.g. 5×5 squares) can be used for more severe smoothing. (Note that a small kernel can be applied more than once in order to produce a similar but not identical effect as a single pass with a large kernel.)

$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$

Median Filter

The median filter is normally used to reduce noise in an image, somewhat like the mean filter. However, it often does a better job than the mean filter of preserving useful detail in the image.

Like the mean filter, the median filter considers each pixel in the image in turn and looks at its nearby neighbors to decide whether or not it is representative of its surroundings. Instead of simply replacing the pixel value with the mean of neighboring pixel values, it replaces it with the median of those values. The median is calculated by first sorting all the pixel values from the surrounding neighborhood into numerical order and then replacing the pixel being considered with the middle pixel value. (If the neighborhood under consideration contains an even number of pixels, the average of the two middle pixel values is used.) Figure illustrates an example calculation.



Gradient Filter

As evident by the name, this ltering operation is used to send directional change in intensity in the image. This helps in detection of edges in the image. Edge detection helps in feature extraction and sharpening of the image.
The filter kernels are illustrated below for horizontal and vertical gradients.

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \quad G_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

Laplacian Filter

The Laplacian is a 2-D isotropic measure of the 2nd spatial derivative of an image. The Laplacian of an image highlights regions of rapid intensity change and is therefore often used for edge detection (see zero crossing edge detectors). The Laplacian is often applied to an image that has first been smoothed with something approximating a Gaussian smoothing filter in order to reduce its sensitivity to noise, and hence the two variants will be described together here. The operator normally takes a single gray level image as input and produces another gray level image as output.

The Laplacian *L(x,y)* of an image with pixel intensity values *I(x,y)* is given by:

$$L(x, y) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

Since the input image is represented as a set of discrete pixels, we have to find a discrete convolution kernel that can approximate the second derivatives in the definition of the Laplacian. Two commonly used small kernels are shown below.

0	−1	0
−1	4	−1
0	−1	0

−1	−1	−1
−1	8	−1
−1	−1	−1

Sobel Filter

The Sobel operator, sometimes called the Sobel–Feldman operator or Sobel filter, is used in image processing and computer vision, particularly within edge detection algorithms where it creates an image emphasising edges. It is named after Irwin Sobel and Gary Feldman, colleagues at the Stanford Artificial Intelligence Laboratory (SAIL).

The Sobel–Feldman operator is based on convolving the image with a small, separable, and integer-valued filter in the horizontal and vertical directions and is therefore relatively inexpensive in terms of computations. On the other hand, the gradient approximation that it produces is relatively crude, in particular for high-frequency variations in the image.

The operator uses two 3×3 kernels which are convolved with the original image to calculate approximations of the derivatives – one for horizontal changes, and one for vertical. If we define A as the source image, and Gx and Gy are two images which at each point contain the horizontal and vertical derivative approximations respectively, the computations are as follows:

$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}$$

Algorithm

The algorithm of applying various of the above filters with varying kernel sizes on multiple images involves many steps and subsequent analysis.

For this, we use a trackbar system using which we select the three options:

(i) Image - List of images on which filtering operation can be performed

(ii) Filter - Filter to be used out of the available ones.

(iii) Neighborhood size - Kernel size of the filter (3, 5, 7).

The trackbars are created and manage using the **cv::createTrackbar** function of OpenCV.

The filter kernels for all the filters to be used with different kernel sizes are predefined in the code as follows.

Kernel MEAN_3 = {
{ 1, 1, 1 },
{ 1, 1, 1 },
{ 1, 1, 1 },
};
	Kernel MEAN_5 = {
	{ 1, 1, 1, 1, 1 },
	{ 1, 1, 1, 1, 1 },
	{ 1, 1, 1, 1, 1 },
	{ 1, 1, 1, 1, 1 },
	{ 1, 1, 1, 1, 1 },
	};
		Kernel MEAN_7 = {
		{ 1, 1, 1, 1, 1, 1, 1 },
		{ 1, 1, 1, 1, 1, 1, 1 },
		{ 1, 1, 1, 1, 1, 1, 1 },
		{ 1, 1, 1, 1, 1, 1, 1 },
		{ 1, 1, 1, 1, 1, 1, 1 },
		{ 1, 1, 1, 1, 1, 1, 1 },
		{ 1, 1, 1, 1, 1, 1, 1 },
		};

Kernel GRADIENT_H_3 = {
{-1, -1, -1 },
{ 0, 0, 0 },
{ 1, 1, 1 },
};
	Kernel GRADIENT_H_5 = {
	{ -1, -1, -1, -1, -1 },
	{ -2, -2, -2, -2, -2 },
	{ 0, 0, 0, 0, 0 },
	{ 2, 2, 2, 2, 2 },
	{ 1, 1, 1, 1, 1 },
	};
		Kernel GRADIENT_H_7 = {
		{ -1, -1, -1, -1, -1, -1, -1 },
		{ -2, -2, -2, -2, -2, -2, -2 },
		{ -3, -3, -3, -3, -3, -3, -3 },
		{ 0, 0, 0, 0, 0, 0, 0 },
		{ 3, 3, 3, 3, 3, 3, 3 },
		{ 2, 2, 2, 2, 2, 2, 2 },
		{ 1, 1, 1, 1, 1, 1, 1 },
		};

Kernel GRADIENT_V_3 = {
{ 1, 0, -1 },
{ 1, 0, -1 },
{ 1, 0, -1 },
};
	Kernel GRADIENT_V_5 ={
	{ -1, -2, 0, 2, 1},
	{ -1, -2, 0, 2, 1},
	{ -1, -2, 0, 2, 1},
	{ -1, -2, 0, 2, 1},
	{ -1, -2, 0, 2, 1}
	};
		Kernel GRADIENT_V_7 = {
		{ -1, -2, -3, 0, 3, 2, 1},
		{ -1, -2, -3, 0, 3, 2, 1},
		{ -1, -2, -3, 0, 3, 2, 1},
		{ -1, -2, -3, 0, 3, 2, 1},
		{ -1, -2, -3, 0, 3, 2, 1},
		{ -1, -2, -3, 0, 3, 2, 1},
		{ -1, -2, -3, 0, 3, 2, 1},
		};

Kernel LAPLACIAN_3 = {
{ -1, -1, -1},
{ -1, 8, -1},
{ -1, -1, -1},
};
	Kernel LAPLACIAN_5 = {
	{ -1, 3, -4, -3, -1},
	{ -3, 0, 6, 0, -3},
	{ -4, 6, 20, 6, -4},
	{ -3, 0, 6, 0, -3},
	{ -1, -3, -4, -3, -1},
	};
		Kernel LAPLACIAN_7 = {
		{ -2, -3, -4, -6, -4, -3, -2},
		{ -3, -5, -4, -3, -4, -5, -3},
		{ -4, -4, 9, 20, 9, -4, -4},
		{ -6, -3, 20, 36, 20, -3, -6},
		{ -4, -4, 9, 20, 9, -4, -4},
		{ -3, -5, -4, -3, -4, -5, -3},
		{ -2, -3, -4, -6, -4, -3, -2},
		};

Kernel SOBEL_H_3 = {
{ 1, 2, 1},
{ 0, 0, 0},
{ -1, -2, -1},
};
	Kernel SOBEL_H_5 = {
	{ 1, 4, 7, 4, 1},
	{ 2, 10, 17, 10, 2},
	{ 0, 0, 0, 0, 0},
	{ -2, -10, -17, -10, -2},
	{ -1, -4, -7, -4, -1},
	};
		Kernel SOBEL_H_7 = {
		{ 1, 4, 9, 13, 9, 4, 1},
		{ 3, 11, 26, 34, 26, 11, 3},
		{ 3, 13, 30, 40, 30, 13, 3},
		{ 0, 0, 0, 0, 0, 0, 0},
		{ -3, -13, -30, -40, -30, -13, -3},
		{ -3, -11, -26, 34, -26, -11, -3},
		{ -1, -4, -9, -13, -9, -4, -1},
		};

Kernel SOBEL_V_3 = {
{ -1, 0, 1 },
{ -2, 0, 2 },
{ -1, 0, 1 },
};
	Kernel SOBEL_V_5 = {
	{ -1, -2, 0, 2, 1},
	{ -4, -10, 0, 10, 4},
	{ -7, -17, 0, 17, 7},
	{ -4, -10, 0, 10, 4},
	{ -1, -2, 0, -2, 1},
	};
		Kernel SOBEL_V_7 = {
		{ -1, -3, -3, 0, 3, 3, 1},
		{ -4, -11, -13, 0, 13, 11, 4},
		{ -9, -26, -30, 0, 30, 26, 9},
		{ -13, -34, -40, 0, 40, 34, 13},
		{ -9, -26, -30, 0, 30, 26, 9},
		{ -4, -11, -13, 0, 13, 11, 4},
		{ 1, -3, -3, 0, 3, 3, 1},
		};

Kernel SOBEL_D_3 = {
{ 0, 1, 2},
{ -1, 0, 1},
{ -2, -1, 0},
};
	Kernel SOBEL_D_5 = {
	{ 0, 1, 2, 3, 4},
	{ -1, 0, 1, 2, 3},
	{ -2, -1, 0, 1, 2},
	{ -3, -2, -1, 0, 1},
	{ -4, -3, -2, -1, 0},
	};
		Kernel SOBEL_D_7 = {
		{ 0, 1, 2, 3, 4, 5, 6},
		{ -1, 0, 1, 2, 3, 4, 5},
		{ -2, -1, 0, 1, 2, 3, 4},
		{ -3, -2, -1, 0, 1, 2, 3},
		{ -4, -3, -2, -1, 0, 1, 2},
		{ -5, -4, -3, -2, -1, 0, 1}
		{ -6, -5, -4, -3, -2, -1, 0}
		};

Depending on the choice in the trackbar system the code executes on the selected image, with the selected filter and kernel size.
The Graphical User Interface is user-friendly and allows real-time changes in any of the three parameters, displaying the outputs automatically.

The code has structures and typedefs used to represent the various data types used.

The functions which carry out the task of filtering are described as below:

convolute: Performs convolution of the input image with the selected filter kernel
Input: Input Image to be filtered, FIlter Kernel to be used (size and type both included)
Output: Filtered image

```
for i = 0...IMAGE_HEIGHT-1
    for j = 0...IMAGE_WIDTH-1
        sum = 0; den = 0;
        for k = -KERNEL_HEIGHT/2...KERNEL_HEIGHT/2
            for l = -KERNEL_WIDTH/2...KERNEL_WIDTH/2
                sum += input_img[i+k-KERNEL.size()/2][j+l-KERNEL.size()/2] * kernel[k][l];
                if kernel[k][l] is positive
                    den += kernel[k][l];
        Output_img[i][j] = abs(sum/den);
```

Median: Performs median filtering on an input image with given kernel size
Input: Input image, Kernel Size (3, 5, 7)
Output: Median filtered output image

```
for i = 0...IMAGE_HEIGHT-1
    for j = 0...IMAGE_WIDTH-1
        for pixels in neighbourhood
            Add pixel to array neighbourhood
        Sort array neighbourhood
        Output_img[i][j] = median(neighbourhood array)
```

callBack: Analyses the trackbar and calls the convolute() function with appropriate arguments

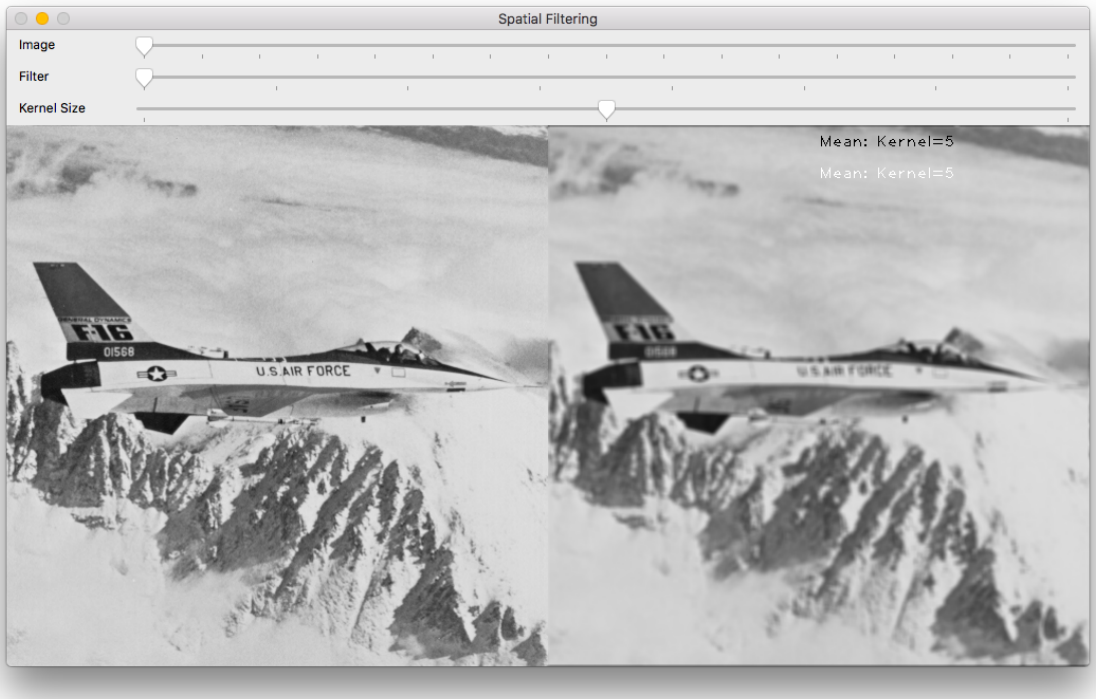
Input: Trackbar data

Output: Analysed values of the three parameters (Image, Filter type, Kernel size)

- **Cascaded Switch Cases to determine the parameters and call the convolute() function accordingly**
- **In case of Median Filter the Median() function is called since it doesn't have a filter kernel and is implemented in code.**

Results

Mean Filter:



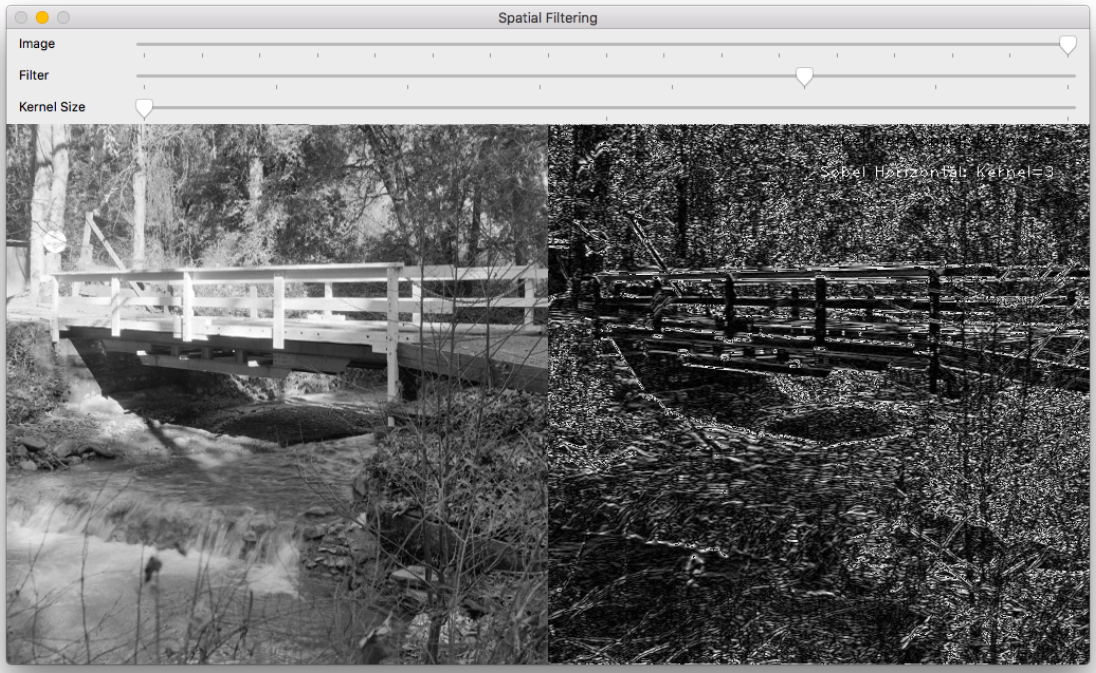
Median Filter:

Laplacian Filter:

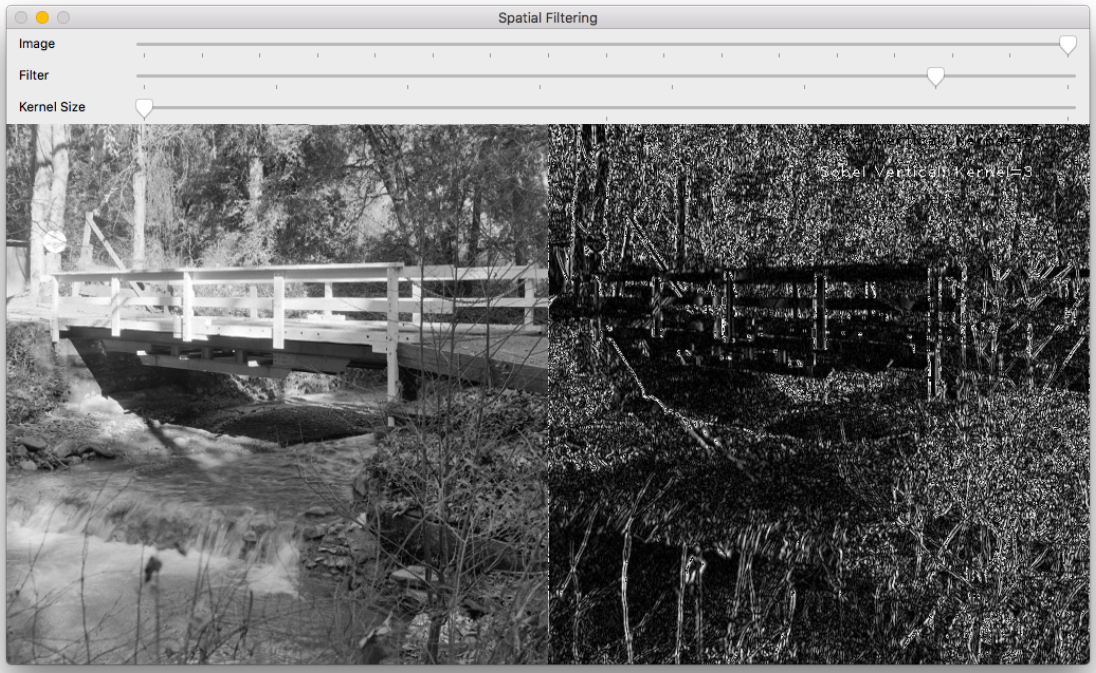
Horizontal Gradient:

Vertical Gradient:

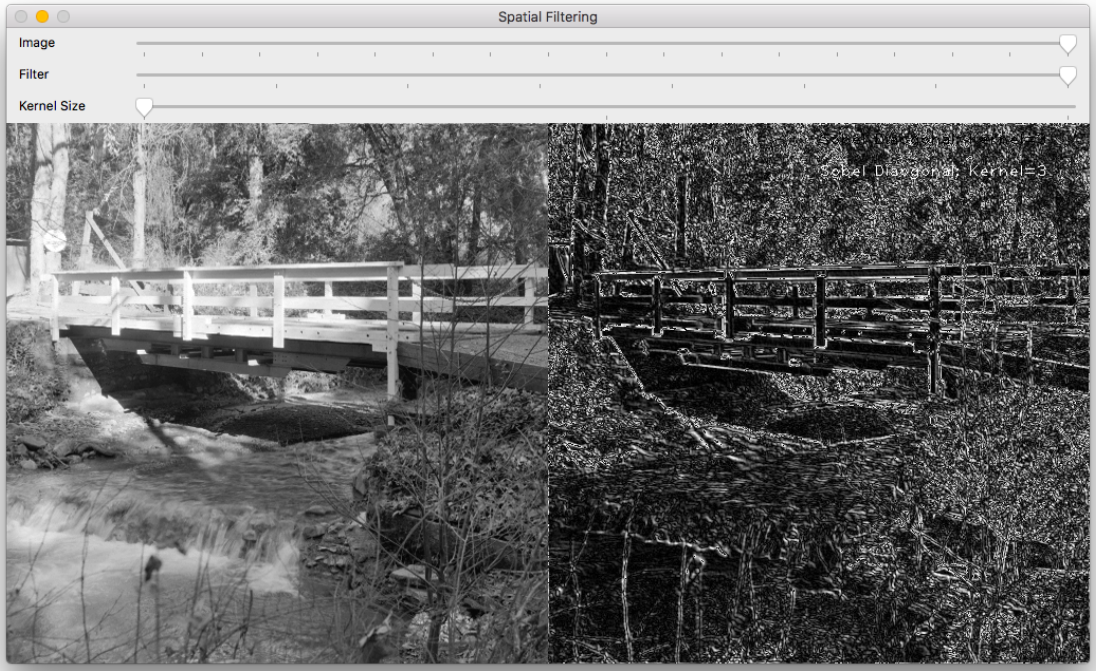
Horizontal Sobel:



Vertical Sobel:



Diagonal Sobel:



Analysis

Aditya Narayan [13EC35010]

1. The mean filter is used as a blur tool to round off edges in an image. It is useful when images have noise other than salt and pepper noise.

2. The median filter works exceptionally well with images containing salt and pepper noise. The outlier pixels in the image are removed by making a median of the pixels in the neighbourhood, as the outliers always the highest intensity or the lowest intensity pixels.

3. The edge detection filters such as gradient, laplacian and sobel filters perform terrible on images with noise. It is recommended to de-noise the images using a low-pass filter such as mean filter or the median filter.
   Canny edge detection algorithm is built upon the sobel filter and utilizes the gaussian blur filter to remove noise from an image before applying the edge detection.

4. The laplacian filter, being isotropic, can detect edges in all directions. The gradient and sobel filters can be made isotropic by superimposing the filtered outputs from horizontal, vertical and diagonal axes.

Analysis

Adarsh Kumar Kosta

[13EC35008]

1. The Mean filter is basically an averaging low pass filter. It is useful as a smoothing filter to filter out noise from a noisy image. The average is taken amongst the neighbourhood of the target pixel.

2. The Median Filter is much more useful in filtering noise as it does not take average of pixel intensities but replaces the target pixel value with the most statistically significant value. It thus preserves details in the image like edges which are lost in case of a Mean filter.

3. The Gradient filter detects edges in the image. It employes a first order filter kernel and thus produces lines where the intensity goes from high to low to high. Gradient filter is more effective in detecting gray level steps compared to 2nd order masks. There are separate masks for finding Horizontal, vertical and diagonal edges.

4. The Laplacian filter employs a 2nd order filter mask for filtering. It is useful for detecting sharp boundaries in the image. The boundaries can be curved too, i.e. the Laplacian mask is isotropic and can detect edges in any direction. The laplacian mask is highly susceptible to noise, it being a 2nd order filter.

5. The Sobel filter uses a weighted 1st order filter mask. It is similar to the gradient filter mask but with higher weights to the inner pixels in the mask. Thus it produces a more profound edge boundaries compared to gradient filter.