Jackie Chan
301310345

Assignment 1 Report

Project Instructions

All source code is contained within the a1 folder. There is one script file corresponding to each component of the assignment, from a1_1 for question 1 to a1_4 for question 4.

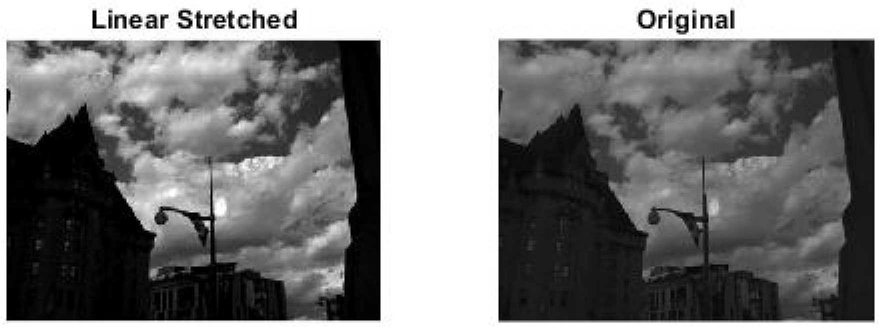
Within the a1 folder are also folders containing images for each question, named q(n)_images, where n is the question number. These folders also contain images comparing the results of different filters and operators within each question.

In order to run the script files, simply click the run button within MATLAB.

Observations

1. Linear Stretch

For this component, I simply implemented the formula provided in lecture notes in order to stretch the values within the image ‘dark.tif’. This improved the contrast of the image by increasing frequency of lower and higher gray-level values.



A direct comparison of the linear stretched image and the original image

The linear stretched image has a noticeably increased contrast. However, the level of detail in darker regions is reduced, as many lower gray-level values are reduced to 0, resulting in less clarity in the darker regions.

The result above was produced with a b value of 170, and an a value of 22.

2. Smoothing

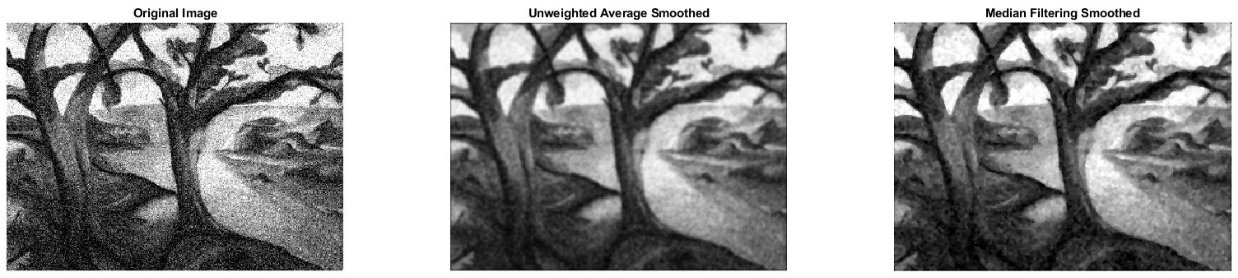
Smoothing methods were implemented and tested on tree images of varying Gaussian noise and salt and pepper noise.

To smooth the images with Gaussian noise, I implemented Unweighted Averaging smoothing by convolving the image with uniform filters of different sizes. In the end, a 3×3 filter produced the best results for unweighted averaging.

For the images with salt and pepper noise, I implemented K-Nearest Neighbour Averaging, by iterating through each pixel within the image and taking the mean of the K nearest neighbours. After testing, I found that a K value of 4 produced the best results.

I also tested and compared Median Filtering, on both the images with Gaussian noise and the images with salt and pepper noise. In both cases, Median Filtering seemed to produce the best results of the 3 methods.

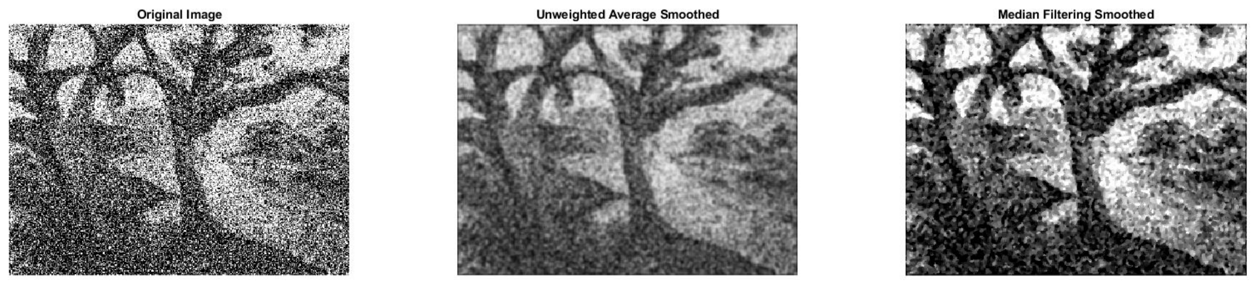
Each of unweighted averaging, k-nearest neighbour averaging, and median filtering were tested with multiple iterations, and the best results came with 3 iterations of each method.



A comparison of unweighted averaging and median filtering on a tree image with low Gaussian noise

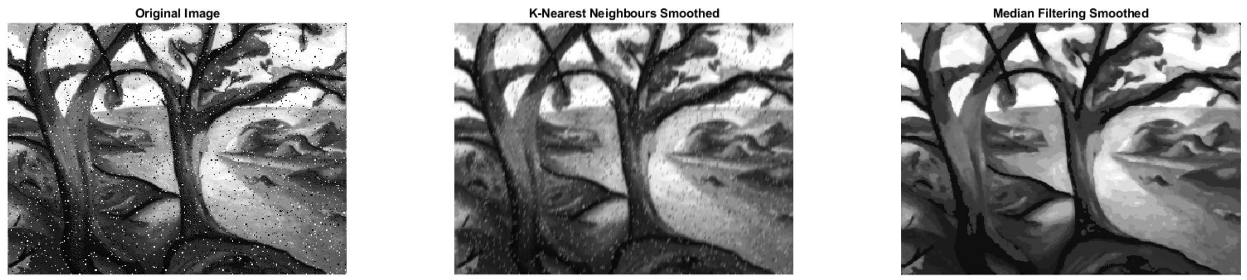
Unweighted averaging was fairly effective at removing noise in an image with fairly little Gaussian noise. However, the image becomes heavily blurred, and detail is heavily reduced in the resulting image. Median filtering performed better than unweighted averaging here, but the resulting image is still blurry, although more detail is kept than with unweighted averaging.

Jackie Chan
301310345



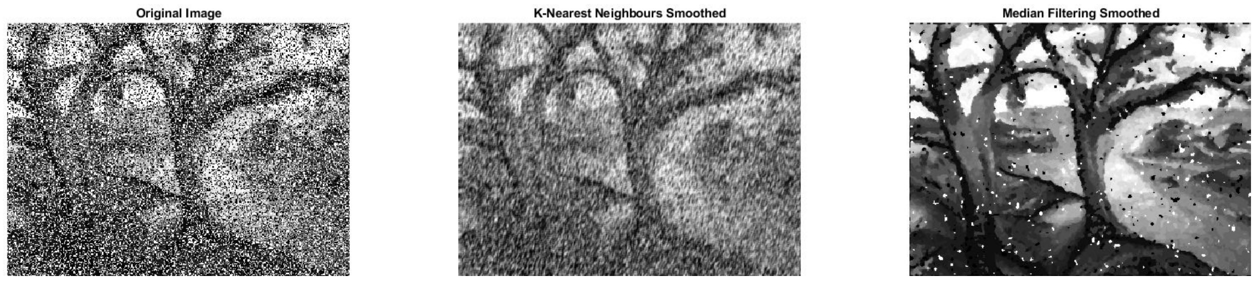
A comparison of unweighted averaging and median filtering on a tree image with high Gaussian noise

Both unweighted averaging and median filtering and not very effective at removing large amounts of Gaussian noise. However, median filtering is able to retain and improve the contrast of the original image, while unweighted averaging decreases contrast and leaves the image more washed out.



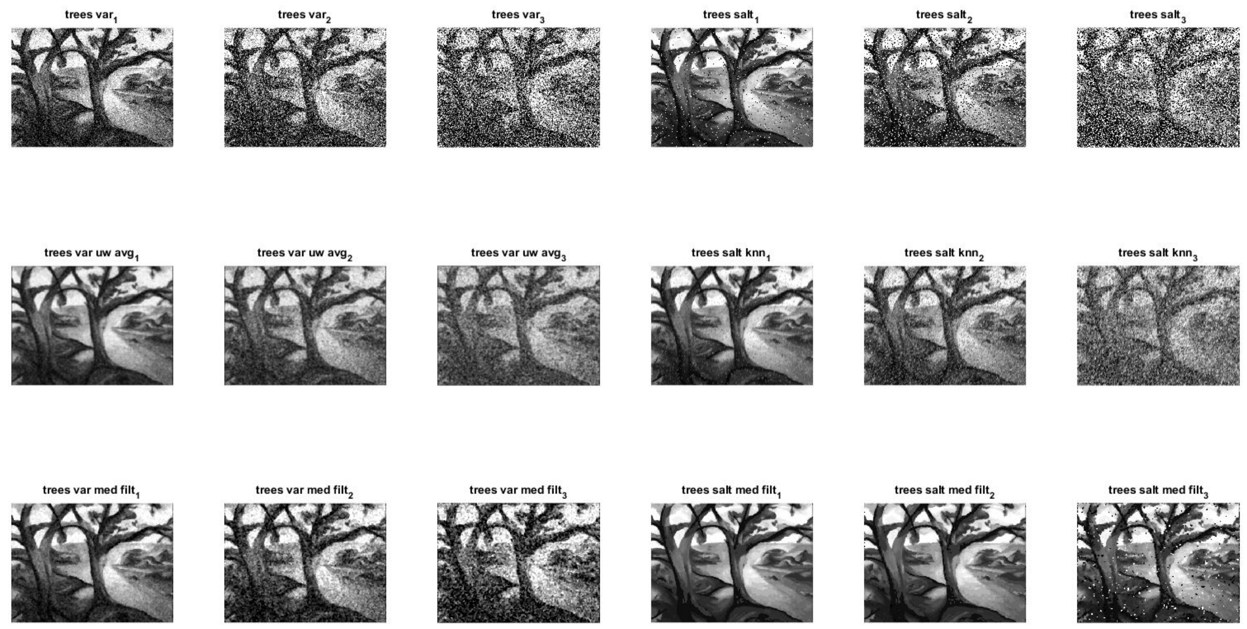
A comparison of K-nearest neighbour averaging and median filtering on a tree image with low salt and pepper noise

K-nearest neighbours averaging is able to remove some of the salt and pepper noise in the image, but still leaves significant marks behind on the image. Median filtering, however, excels at smoothing salt and pepper noise images, leaving no noise behind and retaining the details of the image.



A comparison of K-nearest neighbour averaging and median filtering on a tree image with high salt and pepper noise

For images with large amounts of salt and pepper noise, K-nearest neighbours averaging is ineffective and fails to smooth the image. Median filtering continues to excel, however, and is very effective at removing a majority of the noise in the image, and maintaining the detail behind the image. However, not all noise is smoothed out.



A comparison of the results from all 3 methods

Overall, it seemed that unweighted averaging performed better with Gaussian noise than salt and pepper noise, and K-nearest neighbour averaging performed better with salt and pepper noise than Gaussian noise. However, median filtering outperformed both of the other methods, producing decent results for images with Gaussian noise, and excelling at smoothing images with salt and pepper noise.

3. Sharpening

In order to implement sharpening, I tested various sharpening filters on the 'peppers.png' and 'flower-glass.tif' images. Each sharpening filter was tested on both the luminance values of the image, and all 3 RGB values of the image. An unsharp masking filter was also tested, with the imsharp method.

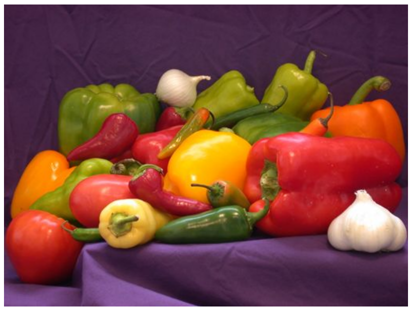
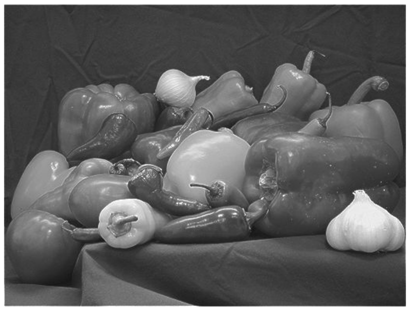
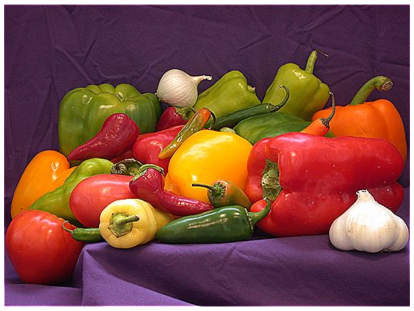
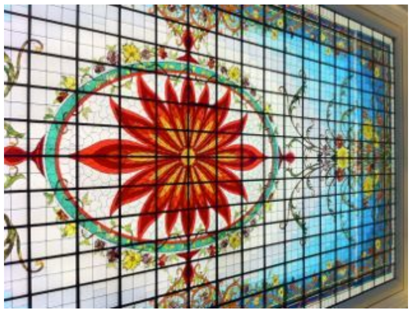
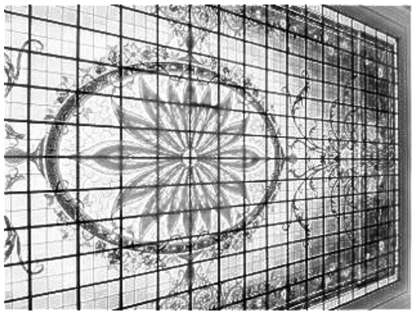
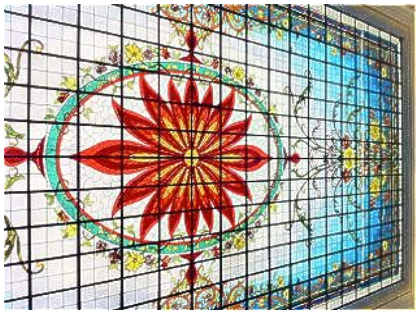
The filters used are as follows:

$$\text{filter}_1 = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}, \text{filter}_2 = \begin{bmatrix} 0 & -1/2 & 0 \\ -1/2 & 3 & -1/2 \\ 0 & -1/2 & 0 \end{bmatrix},$$

$$\text{filter}_3 = \begin{bmatrix} 0 & -2 & 0 \\ -2 & 9 & -2 \\ 0 & -2 & 0 \end{bmatrix}, \text{filter}_4 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Filter$_1$ and filter$_3$ produced the most noticeable results, which will be discussed in more detail.

Jackie Chan
301310345

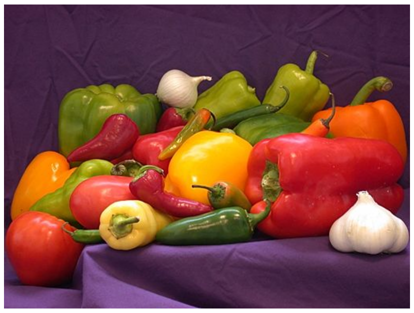
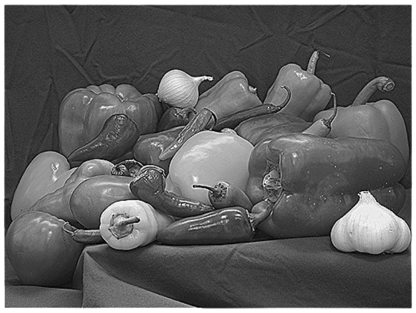
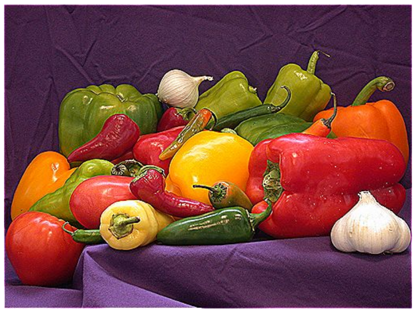
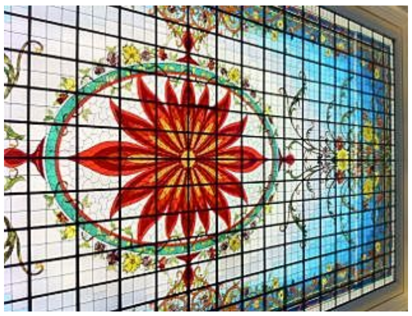
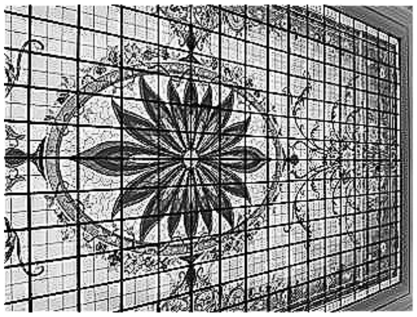
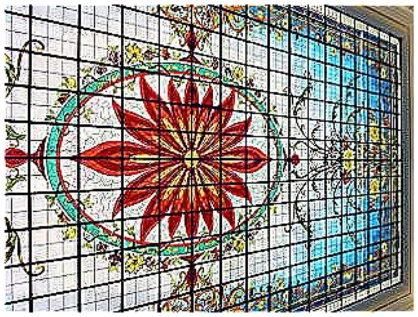







Original

Filter$_1$ on luminance alone

Filter$_1$ on all RGBs

Filter$_1$ is also the Laplacian operator. After convolving the image with filter$_1$, the result was then subtracted from the original image in order to achieve the sharpened image. Filter$_1$ was an effective sharpening filter for both luminance and RGB values, with noticeably sharper edges for both images. However, filter$_1$ removes a lot of the contrast on the flower pattern when applied to luminance alone, resulting in a more washed out image.

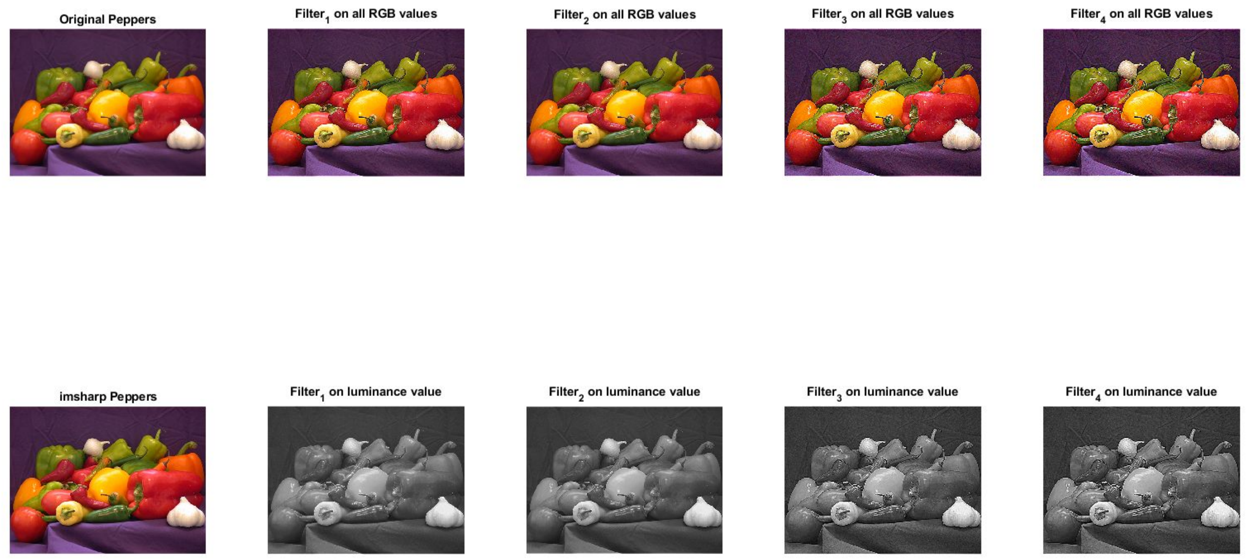







Unsharp mask

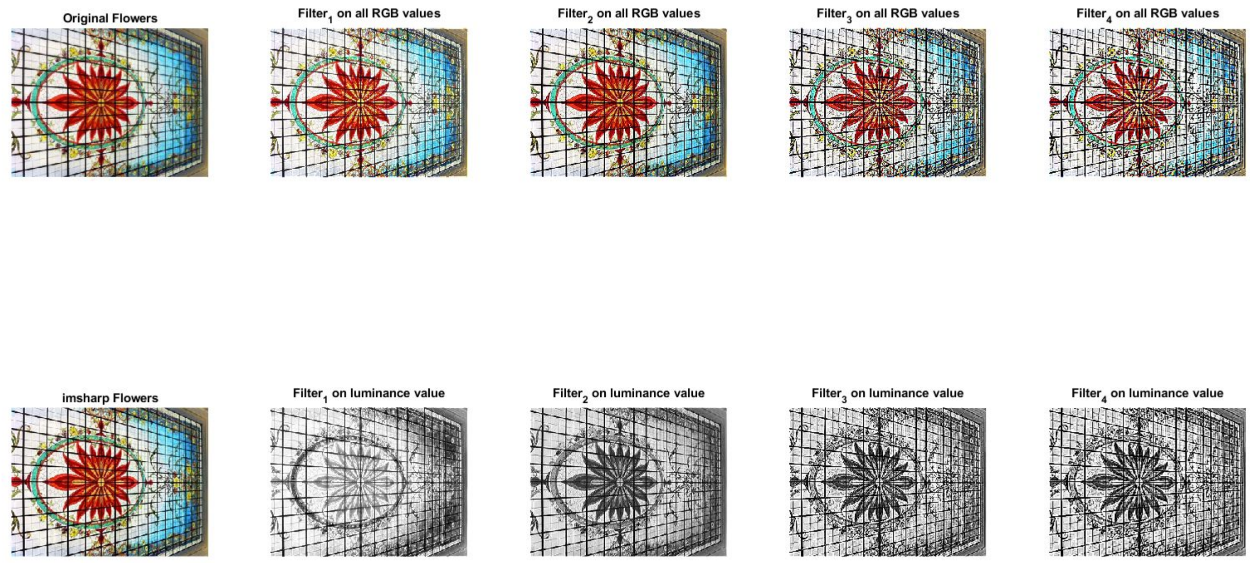
Filter$_3$ on luminance alone

Filter$_3$ on all RGBs

Filter$_3$ had a much more pronounced effect on each image, and produces a much more dramatic sharpening than the imsharp function provided by MATLAB. Also, filter$_3$ maintains the contrast in the flower-glass image even when applied to luminance alone. However, filter$_3$ is also considerably noisier than filter$_1$, especially when applied to the flower-glass image.


Original Peppers
Filter$_1$ on all RGB values
Filter$_2$ on all RGB values
Filter$_3$ on all RGB values
Filter$_4$ on all RGB values
imsharp Peppers
Filter$_1$ on luminance value
Filter$_2$ on luminance value
Filter$_3$ on luminance value
Filter$_4$ on luminance value

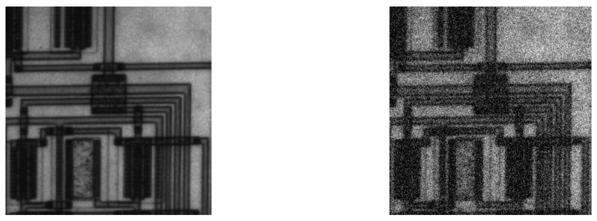
A comparison of the effects of each filter on the peppers image


Original Flowers
Filter$_1$ on all RGB values
Filter$_2$ on all RGB values
Filter$_3$ on all RGB values
Filter$_4$ on all RGB values
imsharp Flowers
Filter$_1$ on luminance value
Filter$_2$ on luminance value
Filter$_3$ on luminance value
Filter$_4$ on luminance value

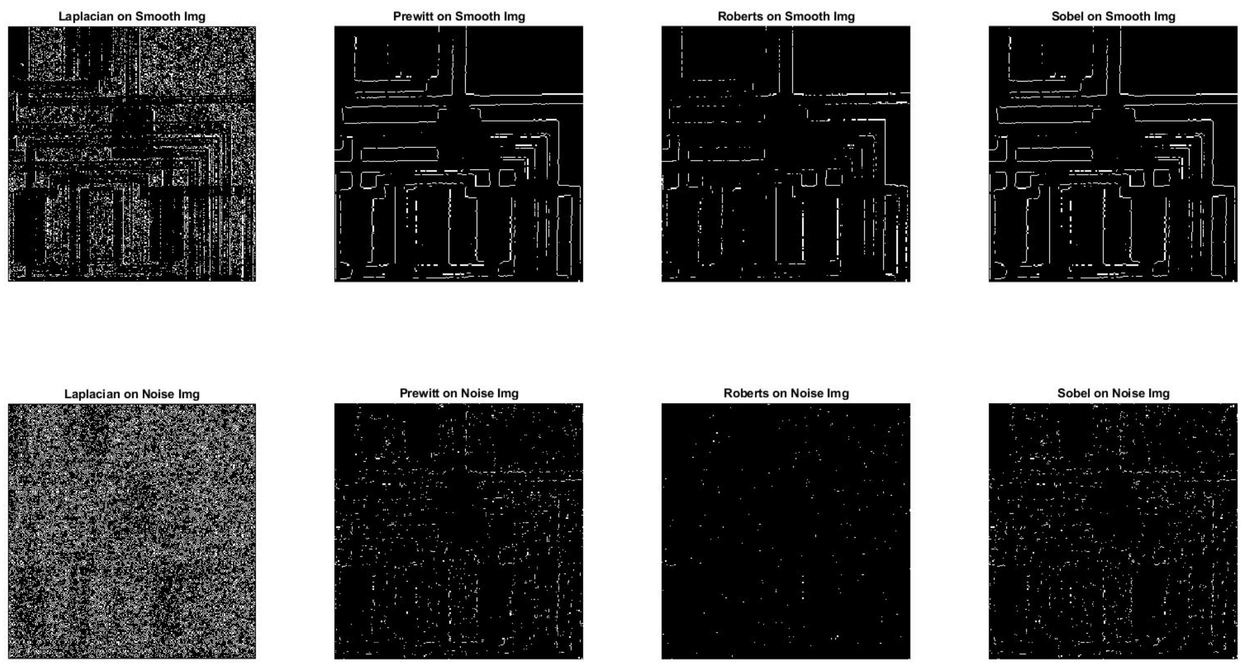
A comparison of the effects of each filter on the flower-glass image

4. Edge Detection

The Laplacian zero-crossing, Prewitt, Sobel, and Roberts operators were tested for the first component of this section. Each operator was tested on a smooth circuits image (‘circuit.tif’), and a circuits image with Gaussian noise (‘circuit_var002.tif’).



The smooth and noisy circuit images tested on, respectively



A comparison of the different operators on a smooth circuits image and a noisy circuits image

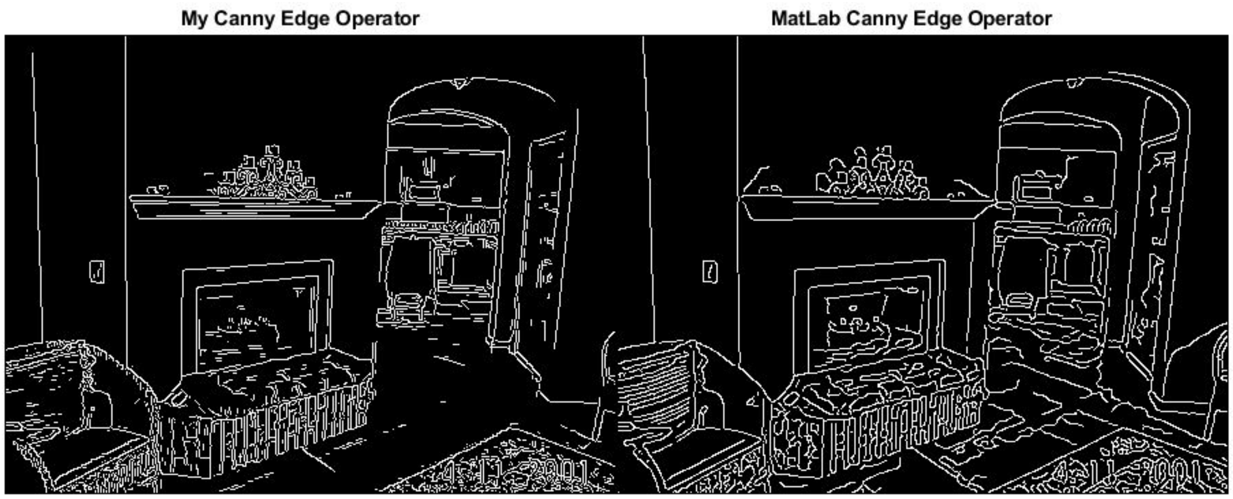
Of the 4 operators, the Prewitt and Sobel operators performed the best on the smooth circuits image, producing fairly clear edges. The Laplacian operator picked up large amounts of noise, while the Roberts operator was unsuccessful in capturing many major edges.

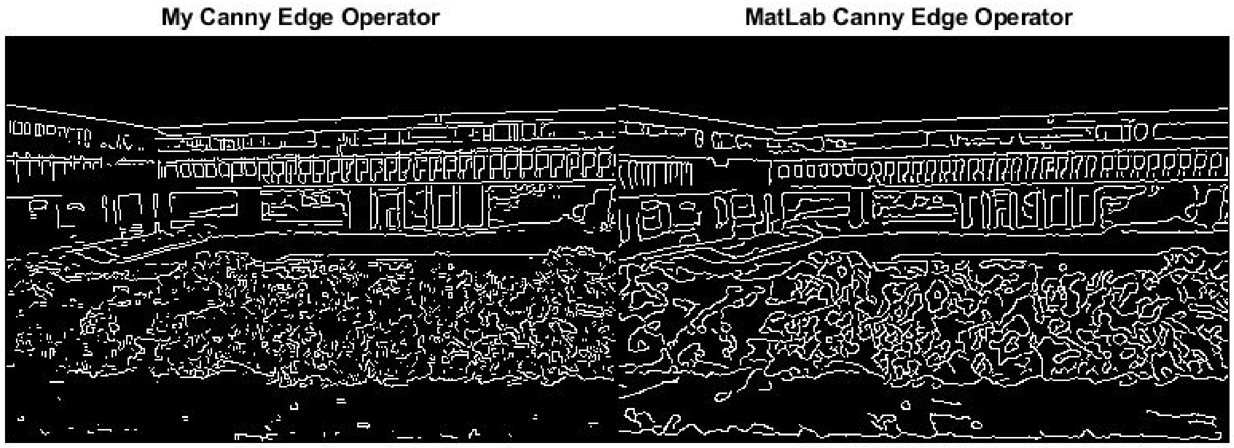
However, all 4 operators performed poorly with the noisy circuits image, failing to produce any clean edges and instead producing large amounts of noise. The Laplacian and Roberts operators performed the worst, either producing very high amounts of noise or not producing any discernible edges.

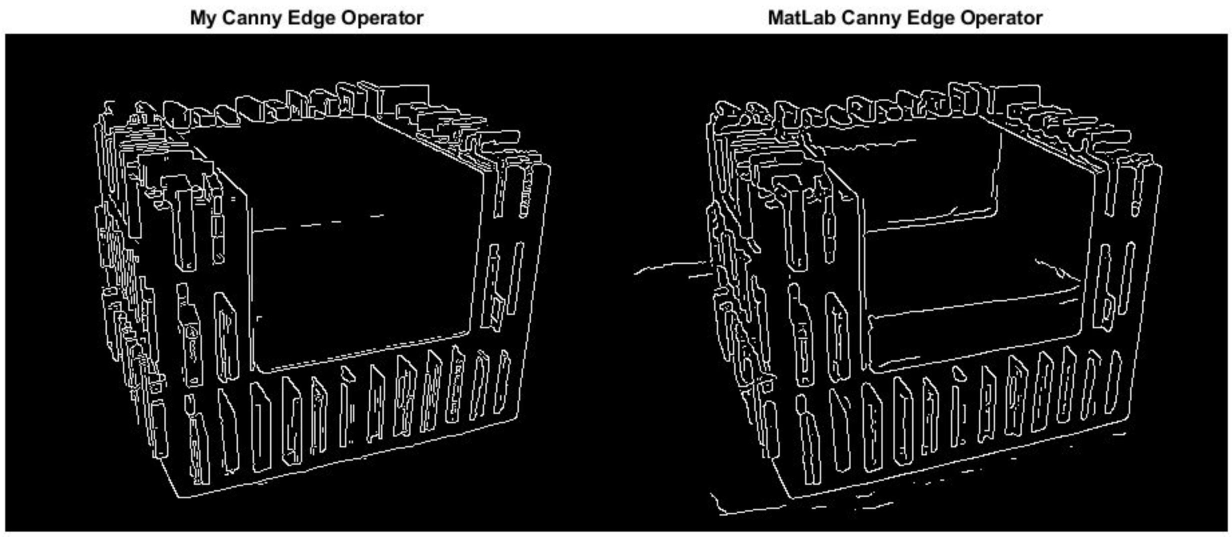
Canny Edge Operator

The Canny Edge operator was implemented by following the algorithm provided in class. It was tested on the circuits image, as well as the ‘sfu.jpg’, ‘room.tif’, and ‘sofa.tif’ images.

Jackie Chan
301310345

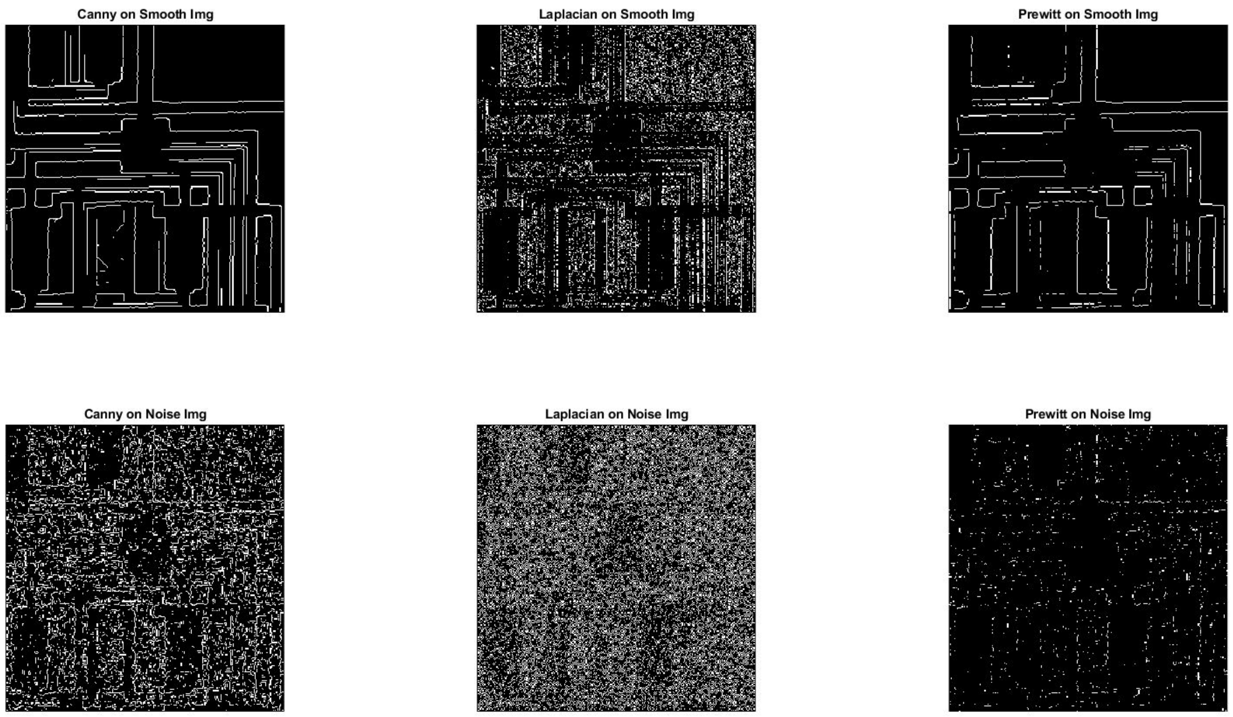






The Canny Edge Operator performed very well for all 3 of the room, SFU, and sofa images. However, my Canny Edge Operator implementation retains some more noise than the Canny Edge Operator method provided by MATLAB.

Jackie Chan
301310345



A comparison of the Canny Edge Operator on smooth and noisy circuits images with Laplacian and Prewitt operators

The Canny Edge Operator continues to perform well with the circuits images, producing very clear and defined edges for the smooth circuits image. However, the Canny operator produces large amounts of noise when applied to the noisy circuits image, but not nearly as much as the Laplacian operator. The edges produced by the Canny operator on the noisy image can still be seen beneath the noise.