Image Processing and Edge Detection Techniques

Emma Hughson

301356242

Instructions for running matlab files: Each question in the assignment is associated with its own matlab file. Each matlab file contains a function with a parameter to input a specified image of your choice.

For question 1, there are three parameters: image, lower percentile, and high percentile. The low percentile parameter allows you to change the amount of the histogram that is determined for the min value in g(x,y) = (f(x,y) – min)/(max-min))\*255. While the high percentile parameter allows you to change the amount of the histogram that is determined for the max value in g(x,y) = (f(x,y) – min)/(max-min))\*255.

For question 2, the only parameter that is considered is the image that is being used to run the function on. You can go into the function and alter the parameters for imgaussfilt(), medfilt2(), and imboxfilt().

For question 3, the only parameter that is considered is the image that is being used to run the function on. You can go into the function and alter the parameters for imsharpen(), as well as kernel values.

For question 4, there are two files one for part a and one for part b. In part a, the parameter is just an image. You can alter the function parameters for sobel, prewitt, and Laplacian functions. For part b the parameters are image, sigma, size of filter, maximum threshold, minimum threshold.

Image Processing and Edge Detection Techniques

**1. Contrast Stretching vs. Histogram Equalization**

Contrast is the measure of dynamic range of an image’s histogram. Dynamic range can be further thought of as the maximum pixel value minus the minimum pixel value. Linear stretching is a technique that, when applied to images, can enhance the contrast of an image [1]. Usually linear stretching is implemented when an image lacks contrast. Significant lack of contrast can lead to misidentification of objects and loss of spatial relationships. Linear stretching involves linearly expanding the original pixel values for every pixel in an image [2]. By doing this, subtle variations in data become more present.

The first task was to write a function that linearly stretched a gray scale image. In this function, Min-Max Linear Contrast stretching was applied. In this way, the minimum values and the maximum values are given new predefined values that take advantage of the availability of brightness in the image. Using the original histogram values of the image, an optimal contrast was achieved using the minimum value 0.004 percent and the maximum value of 95th percentile of the histogram. After finding the minimum and maximum values, each pixel of the new linearly stretched image was calculated by using g(x,y) = (f(x,y) – min)/(max-min))\*255. The formula is multiplied by 255 because that is number of intensity levels an image has.

 The original image (fig.1) has had no linear stretching techniques applied to it. The image is dull and lacks definition of objects’ spatial relationships. After applying Min-Max Linear Contrast stretching, the objects in the image (fig. 2) become more defined and the clouds in the sky became crisper and more realistic. This result is due impart to the enhancement of the histograms dynamic range, making a more well-balanced image.

 Another technique to enhance an image’s contrast is histogram equalization. Histogram equalization is a nonlinear technique that spreads out the most frequent intensity values [3]. Spreading out the most frequent intensity values causes the intensities of the image to be more well-distributed. In other words, the contrast is increased where the intensity values are the most populated and reduces the contrast in areas of the images that consists of either very light or very dark pixel values [2]. The resulting image (fig. 3), after applying histogram equalization, makes the spatial relationship amongst objects even more defined than applying Min-Max Linear Contrast stretching.

*Figure 2*. Contrast stretching.

*Figure 1*. Original image.

The reason that Min-Max Linear Contrast stretching results in a less defined image is due to the unreliability of contrast stretching. Linear stretching maps the maximum intensity to the maximum value and the minimum intensity to the minimum value. Therefore, if the range of the pixel values covers the entire range of possible pixel values initially then the output of linear stretching would have produced nothing [4]. Histogram equalization remaps the pixel values, therefore equalizing brightness values and avoiding the limitation accompanied with contrast stretching.

*Figure 3*. Histogram equalization.

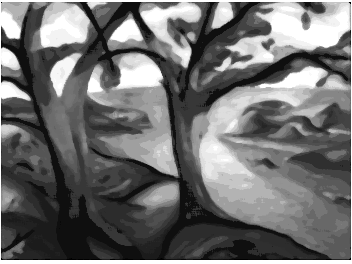
**2. Smoothing vs. Median Filtering**

Smoothing filters are primarily used to reduce the noise that is contained within an image [5]. In images, the pixel values have varying values that represent the color of a given pixel. Sometimes within these varying values there is a very different value, or outlier, which cause noise in the image. Using a smoothing filter, these outliers can be removed/supressed from the image [7]. These filters are constructed from low pass filters (LPF) which takes the average of nearby pixels in a mask. The LPF remap the frequencies so that the highest frequencies occur at edges. These masks are considered the kernel and can range in size, such as a 3x3 or 5x5 array.

In normalized box filtering, all the pixels within the kernel are summed together, and then each value inside the array is divided by the summation of values. Afterwards, the center value of the kernel is replaced by this average [6]. In fig. 4 and 5 below, the effects of using normalized box filtering are presented on the salt-and-pepper noise image. As the size of the kernel increase, the image becomes smoother, but also causes the image to become blurred. This negative side effect could be caused by pixels in the kernel that are not representative, biasing the mean value. As well, this effect may also impact the edge of the objects in the image because of the large difference in values between the edges. This large difference will be changed by replacing edge values with values that are not representative of the edge. On the gaussian noise images, the smoothed image performs very well in early iterations, removing noise. But a similar effect happens as the iteration increase and the image loses definition becoming more blurred.

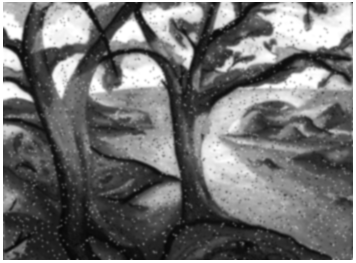
Another way of smoothing images is using Gaussian Filtering. Gaussian Filtering takes the weighted average of the surrounding pixels within a mask with further pixels carrying less weight. However, instead of summing the original values of the image and normalizing the array like normalized box filtering, each pixel value is re-evaluated using the following formula: G(x, y) = . In the formula x and y are the distance from the center of the kernel. In fig. 6 and 7 below, gaussian filtering is shown with increasing filter size on the salt-and-pepper noise images. Larger values of filter size cause the image to remove some of the noise and the blurring effect is not present like the normalized box filtering. But after 5 iterations noise is still present. On the gaussian noise image, the effect is superior to normalized box filtering. The first few iterations reduce the noise without blurring the image. If sigma values were used instead for the gaussian filter implementation, the blurring effect would remove the noise more but increasing sigma would result in a blurred effect similar to normalized box filtering (fig. 18). Incorporating the sigma value controls the Gaussian distributions shape; larger sigma values have flatter distributions than smaller sigma values which have a defined peak [9]. But the larger filter sizes results in a wider range of blurring. This is due to larger sigma values causing the gaussian distribution to become flatter, therefore losing edge definition.

Finally, median filtering can also be applied for smoothing images. Unlike the previous two filtering techniques mentioned, median filtering removes the noise while also preserving edges. Median filtering takes the median value of the kernel and replaces the center value with this median value [10]. In fig. 8 and 9, median filtering is presented on the salt-and-pepper noise image. The edges remain present as the kernel size increases and the noise is removed in one iteration. However, the definition becomes flatter due to the median value of the kernel replacing the center pixel value. Over the duration of median filtering, very high values and very low values become lost because they are replaced with median values that surround it, degrading the image. On the gaussian noise image, median filtering performs better than both gaussian and normalized box filtering. The blurring effect is present, but the images is not as degraded compared to when using just the salt-and-pepper noise image (fig. 10). This might show that median filtering is greatly affected by outliers in the data and performs better under gaussian noise.

****

*Figure 9*. median filter, iteration 5 on salt-and-pepper image.

*Figure 4*. normalized box smoothing, iteration 1 on salt-and-pepper image.

****

*Figure 7*. gaussian filter, iteration 3 on salt-and-pepper image.

*Figure 5*. normalized box smoothing, iteration 5 on salt-and-pepper image.

*Figure 8*. median filter, iteration 1

on salt-and-pepper image.

*Figure 6*. gaussian filter, iteration 1 on salt-and-pepper image.

*Figure 18. gaussian filter using sigma instead of filter size*, iteration 1 on salt-pepper-noise.

image.

*Figure 10*. median filter, iteration 5 on gaussian noise

image.

**3. Sharpening Images**

Sharpening an image involves high pass filtering to emphasize details within an image. Similar to low pass filtering in section 2, high pass filtering involves the use of a kernel that amplifies noise instead of reducing it. This is done by allowing high intensity values to pass the filter, while low intensity values are not [11]. One approach is to use a simple sharpening filter. This is done by adding a source mask to an edge detector mask. This mask will look like [0, -1, 0; -1, 5, -1; 0, -1, 0]. Another approach is the unsharp technique to sharpen an image. This approach involves subtracting a blurred version of the image from the original, increasing contrast [12]. Unsharp masks can be implemented using imsharpen(image, radius, amount) in matlab. The imsharpen() function uses a gaussian mask to implement sharpening. Another way of implementing unsharp masks is by using a Laplacian mask such as kernel = [-1, -1, -1; -1, 8, -1; -1, -1, -1]. Using a Laplacian mask supposedly extracts high intensity values directly [13].

Using the imsharpen() function with a radius of 2 (i.e., radius is the standard deviation of the Gaussian low pass filter) and amount of 2 (i.e., amount controls the sharpening effect), results in a more defined image than the original one. Running a simple sharpening mask results in a more defined image than the original image but is subtly less defined than when Unsharp is used. As well, when the radius of the gaussian filter is increased in the imsharpen() function, the sharpening effect covers a wider region. While increasing the kernel value of simple sharpening mask results in a high contrast and noisy image. This demonstrates the versatility and robustness of the unsharp technique compared to using a simple sharpening mask. Unsharp techniques superiority is because it involves combining a blurred image and the original image. Simply using a sharpening mask involves only the original image and a sharpening mask. Therefore, this blurred image is the solution to producing a more defined image. In addition to using a Gaussian mask, using a Laplacian mask was also used for the unsharp mask [13]. Using Laplacian to subtract from the original image created a blurry undefined image. As such, Laplacian masks are not superior to using gaussian masks for unsharp or using a simple sharpening mask.

After conducting the various sharpening filters on the original image, the image was broken down into its RGB channels. After this separation, a simple sharpening mask and imsharpen(), with the same parameter values as before, were applied to each channel. This technique was also applied to the images YUV channels, which consists of luminance, chroma blue, and chroma red. However, instead of applying a sharpening technique on all channels, it was applied only on luminance. Once each channel was sharpened, they were put back together to create one image. At an initial glance, it seems that using unsharp technique created a more defined image than using a simple sharpening mask. However, in both images halos around certain objects were present (fig. 11), with simple sharpening mask having less of a halo effect than unsharp. This halo effect is possibly due to the increase in contrast when sharpening techniques are applied to each RGB channel. After running unsharp and simple sharpening mask on just the luminance channel alone, however, these halos were removed (fig. 12). The remove of the halos using the Luminance channels is most likely due to the effects of sharpening on a non-color channel.

In conclusion, using unsharp technique on images produces more defined images than a simple sharpening mask. As well, as the kernel was increased on the simple sharpening mask, the picture resulted in more noise and higher contrast. While, increasing the radius values of the gaussian filter for unsharp, results in a wider sharpening effect but does not introduce high contrast or noise like using a simple sharpening mask. As such, the unsharp approach using a gaussian mask is a better technique to sharpen images than a simple sharpening mask or using a Laplacian mask.



*Figure 12*. running unsharp and simple sharpening mask on luminance channel alone.

*Figure 11*. running unsharp and simple sharpening mask on RGB channels.

**4. Edge Detection**

Edge detection is an image processing technique that determines the boundaries of objects within an image. This detection is done by finding the changes in intensity values within a given image [14]. Using first derivative gradient operators, regions that have a high spatial gradient are emphasized by approximating absolute gradient magnitude for each pixel in an image [16]. Edge detection simplifies image data, decreasing the amount of processing required [21]. There are several different edge detection techniques. The techniques discussed here are Prewitt, Sobel, Laplacian, and Canny Edge Operator.

**4.1. Prewitt**

Prewitt operator detects horizontal and vertical edges by using first order derivates that detect vertical and horizontal directions and magnitudes [15]. Prewitt operator has 8 operators that cover 8 possible orientations. The masks calculate the difference between the right versus the left edge of a pixel, hence why the middle row or column of each mask contains 0’s. Running Prewitt operator on a noisy image resulted in an image that emphasized the noise, resulting in an image that looked blizzard-like. Behind the noise there is an outline of the images in the original picture, but these edges are hard to make out through the noise in the foreground. This demonstrates the sensitivity Prewitt operator has to noise, and also emphasizes the importance of removing noise beforehand.

**4.2. Sobel**

Sobel operator detects edges by also calculating first order derivatives for horizontal and vertical directions. These edges can be approximated by using similar masks to Prewitt’s. These masks also cover 8 orientations, but primarily is used to detect X and Y directions [16]. Using the gradient result, the direction of change can be determined indicating whether an edge is present or not. Running Sobel operator on the noisy and smooth images resulted in similar results to the Prewitt operator. Again, the masks are sensitive to noise. The output image produced by Sobel amplifies the noise that was present in the original noisy image. Comparing the smooth images, Sobel’s output resulted in brighter areas being more emphasized than Prewitt’s output. As well, Prewitt resulted in more fuzzy edges compared to Sobel’s, where edges were subtly more defined.

**4.3. Laplacian**

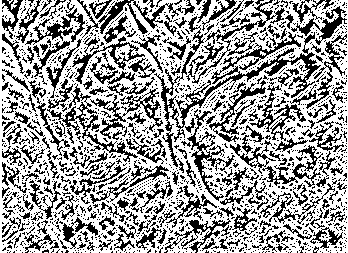
Laplacian operator, unlike Prewitt and Sobel, uses second order derivatives of pixel values to determine edges in an image. Detection occurs by locating zero crossing in the second derivative [21]. As well, Laplacian operator only uses one mask to detect edges instead of 8 [18]. The Laplacian mask considers a wider area of pixels than Prewitt and Sobel. The disadvantage of using the Laplacian operator is that it has difficulties detecting orientation of edges [19]. After running the Laplacian operator on the smooth image this limitation is clearly present. A few edges are faintly present in the resulting image illustrating Laplacian operators’ inability to detect curves and corners. Another limitation, that is also present in the output of the noisy image, is that Laplacian is also very sensitive to noise in an image.

**4.4. Canny Edge Operator**

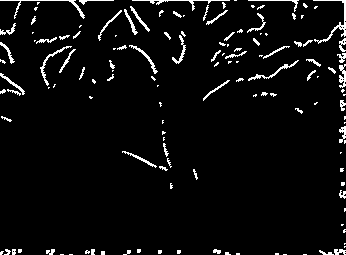
Canny Edge operator consists of four steps: apply smoothing filter, apply gradient operator, non-maximum suppression, and hysteresis thresholding. The smoothing portion of the operator is to remove noise from the image. After removing noise from the image, a gradient operator (e.g., Sobel) is then used to determine direction and magnitude of the gradient for each pixel. Non-maximum suppression is then used to remove unwanted pixels which are most likely not part of any edge. This is done by checking both sides of a pixel to determine if said pixel is the maximum value. If the pixel is the maximum value, it is then declared an edge. Finally, hysteresis thresholding is applied to the pixels in the image. Edges are finally determined by comparing the edges to a maximum and minimum threshold. If the pixel’s intensity is greater than the maximum threshold, the pixel is considered an edge. If the pixel’s intensity is between minimum and maximum threshold, all neighbouring edges are checked to see if they are above the maximum threshold. If any of the neighbouring pixels are edges, then the pixel is labelled an edge. If the pixel is below the minimum threshold it is not an edge [20].

After running the canny edge operator on noisy images, the output resulted in an image that outlined edges of objects present in the original image. Scaling the maximum threshold upwards, the edges become more defined and less noisy (fig. 13; fig. 14; fig. 15). Increasing the maximum threshold too much, however, decreases the amount of detected edges with more reduced noise (fig. 16). The perfect range for maximum threshold, for the image 'trees\_salt004.tif', was between 50 and 100. Altering the sigma value also resulted in varying sizes in edges. The larger sigma was, the thicker the edges, while a smaller sigma value resulted in thinner edges. The perfect result occurred when sigma was 1.60, maximum threshold was 85, and minimum threshold was 5 (fig. 17). Scaling the minimum threshold did not have much of an effect compared to changing maximum threshold. Maximum threshold impacts edge detection more than minimum threshold because values exceeding maximum threshold are given edge values, therefore, demonstrating the importance of maximum threshold in determining the presence of edges in and image. Canny edge detector is also less sensitive to noise than other edge detecting methods, like the gradient operators discussed prior. This insensitivity is most likely due to the smoothing operator that is placed on the image prior to conducting non-maximum suppression and hysteresis.

In conclusion, canny edge detector is a superior method to detecting edges when compared to Sobel, Prewitt, and Laplacian operators. The ability to alter parameters, tweaking the image to perfection, provides an upper hand for the canny edge detector. Insensitivity to noise and increased accuracy compared to other edge detecting methods also gives canny edge detector superiority over other methods.







*Figure 14*. canny edge detector with maximum threshold of 50.

*Figure 17*. canny edge detector with maximum threshold of 85, minimum threshold of 5, and sigma value of 1.60.

*Figure 13*. canny edge detector with maximum threshold of 10.

*Figure 15*. canny edge detector with maximum threshold of 100.

*Figure 16*. canny edge detector with maximum threshold of 200.

References

[1] "Image Processing, Linear stretch and OpenCV - LinuxConfig.org", *Linuxconfig.org*, 2020. [Online]. Available: https://linuxconfig.org/image-processing-linear-stretch-and-opencv. [Accessed: 14- Jan- 2020].

[2] S. Al-amri, N. Kalyankar and S. Khamitkar, "Linear and Non-linear Contrast Enhancement Image", *IJCSNS International Journal of Computer Science and Network Security*, vol. 10, no. 2, 2010. Available: https://pdfs.semanticscholar.org/e326/d1d603cfe0a5e2094eda30577d67fcd4d77b.pdf. [Accessed 16 January 2020].

[3] S. Sudhakar, "Histogram Equalization", *Medium*, 2020. [Online]. Available: https://towardsdatascience.com/histogram-equalization-5d1013626e64. [Accessed: 14- Jan- 2020].

[4] K. Nithya, K. Karthik and K. Ramasamy, "Comparison of Contrast Enhancement Technique with Partitioned Iterated Function System", *International Journal of Scientific & Engineering Research*, vol. 5, no. 5, 2014. [Accessed 18 January 2020].

[5] "Smoothing an Image", *Northstar-dartmouth.edu*, 2005. [Online]. Available: https://northstar-www.dartmouth.edu/doc/idl/html\_6.2/Smoothing\_an\_Image.html. [Accessed: 18- Jan- 2020

[6] A. Mordvintsev and A. K, "Smoothing Images", *Opencv-python-tutroals.readthedocs.io*, 2013. [Online]. Available: https://opencv-python-tutroals.readthedocs.io/en/latest/py\_tutorials/py\_imgproc/py\_filtering/py\_filtering.html. [Accessed: 18- Jan- 2020].

[7] R. Leach, *Journals & Books Fundamental Principles of Engineering Nanometrology*, 2nd ed. Elsevier Inc., 2014, pp. 241-294.

[8]"Gaussian blur", *En.wikipedia.org*. [Online]. Available: https://en.wikipedia.org/wiki/Gaussian\_blur. [Accessed: 18- Jan- 2020].

[9]"Blurring images – Image Processing with Python", *Datacarpentry.org*. [Online]. Available: https://datacarpentry.org/image-processing/06-blurring/. [Accessed: 19- Jan- 2020].

[10] M. Ohki, M. Zervakis and A. Venetsanopoulos, "3-D Digital Filters", *Control and Dynamic Systems*, vol. 69, pp. 49-88, 1995. Available: https://doi.org/10.1016/S0090-5267(05)80038-6 [Accessed 19 January 2020].

[11][Online]. Available: http://ceng503.cankaya.edu.tr/uploads/files/file/Digital%20Image%20Processing-3(1).pdf. [Accessed: 20- Jan- 2020].

[12]"Sharpening Filters", *Theobjects.com*. [Online]. Available: https://www.theobjects.com/dragonfly/dfhelp/30/Content/07\_Image%20Processing/Sharpening%20Filters.htm. [Accessed: 24- Jan- 2020].

[13]"Spatial Filters - Unsharp Filter", *Homepages.inf.ed.ac.uk*. [Online]. Available: http://homepages.inf.ed.ac.uk/rbf/HIPR2/unsharp.htm. [Accessed: 25- Jan- 2020].

[14]"Edge Detection", *Mathworks.com*. [Online]. Available: https://www.mathworks.com/discovery/edge-detection.html. [Accessed: 25- Jan- 2020].

[15]"Prewitt Operator - Tutorialspoint", *Tutorialspoint.com*. [Online]. Available: https://www.tutorialspoint.com/dip/prewitt\_operator.htm. [Accessed: 25- Jan- 2020].

[16] S. Gupta and S. G. Mazumdar, “Sobel Edge Detection Algorithm,” *International Journal of Computer Science and Management Research*, vol. 2, no. 2, pp. 1578–1583, Feb. 2013.

[17] “Edge Detection”, cs.auckland.ac.nz. [online]. Available: <https://www.cs.auckland.ac.nz/compsci373s1c/PatricesLectures/Edge%20detection-Sobel_2up.pdf> [Accessed: 25- Jan- 2020].

[18] U. Sinha, “The Sobel and Laplacian Edge Detectors,” *AI Shack*. [Online]. Available: https://aishack.in/tutorials/sobel-laplacian-edge-detectors/. [Accessed: 25-Jan-2020].

[19] M. Roushdy, “Comparative study of edge detection algorithms applying on the grayscale noisy image using morphological filter,” *GVIP Journal*, vol. 6, no. 4, pp. 17–21, Dec. 2006.

[20] “Canny Edge Detection¶,” *OpenCV*. [Online]. Available: https://opencv-python-tutroals.readthedocs.io/en/latest/py\_tutorials/py\_imgproc/py\_canny/py\_canny.html. [Accessed: 25-Jan-2020].

[21] S. Kaur and I. Singh, “Comparison between Edge Detection Techniques,” *International Journal of Computer Applications*, vol. 145, no. 15, pp. 15–18, 2016.