Lab 2 - Noise Removal

Computational Vision & Imaging

Student ID: XXXXXXX
Random Student, School of Computer Science,
University of Birmingham, UK, B15 2TT

***Abstract*—This document is a formal solution for the Lab 2 Formative Assignment based on Noise Removal. It shows how Gaussian filters can be applied to remove noise form the image and how different parameters (e.g., standard deviation, kernel size) impact the output. It also performs tests on time and checks how Laplacian filter works.**

***Index Terms*—Lab 2, formative assignment, computer vision, imaging, noise removal, filters, kernels, Sobel, Laplacian, Gaussian, MATLAB.**

I. Introduction

CONTINUING the previous topic of *Edge Detection*, we were shown that it can be enhanced by removing the noise from the image. In other words, a filter can be applied to average out abrupt gradient changes which will smoothen the noise. After that, an edge filter can be applied and we can hope to see better results. Often, they depend on the parameters of the noise reduction filter, which most commonly is chosen as Gaussian filter.

II. Setup

We continue the setup of the 1st lab with our working image "shakey.150.gif". In MATLAB, we are additionally provided with Gaussian 3x3 and 5x5 filter. Specific values are not necessary important, what should be noted is that such filters have largest weight on the middle value and smaller weights on the surrounding values. It can be generated using formula 1 (note that $1 \leq i, j \leq (2k+1)$).

$$H_{ij} = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(i-(k+1))^2 + (j-(k+1))^2}{2\sigma^2}\right) \quad (1)$$

We are provided with "N.m" function which calculates a *Gaussian* vector given a mean, a sigma value and a range. The *Gaussian* kernel then can be produced as an outer product of the 2 vectors. We are also given a *Laplacian* filter which is shown by equation 2.

$$\text{Laplacian} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad (2)$$

Laplacian is essentially the sum of second order derivatives. For change in x, we take the difference between the differences involving the center and adjacent pixels for that row, for change in y - involving centre and adjacent pixels in that column. I. e., in our 3-by-3 case, the second order derivative with respect to x and y can be shown by Equation 3 and Equation 4.

$$\begin{aligned}(\nabla^2_{x^2} I)_{h,w} &= (I_{h,w+1} - I_{h,w}) - (I_{h,w} - I_{h,w-1}) \\ &= I_{h,w-1} - 2I_{h,w} + I_{h,w+1}\end{aligned} \quad (3)$$

$$(\nabla^2_{y^2} I)_{h,w} = I_{h-1,w} - 2I_{h,w} + I_{h+1,w} \quad (4)$$

We are also required to use *zerocrossing* method at some point which is provided by MATLAB via built-in "edge" function. *Zerocrossing* essentially means detecting those points where a sign of a value changes. Regarding our functions in assignment 1, we will reuse magnitude and error calculations.

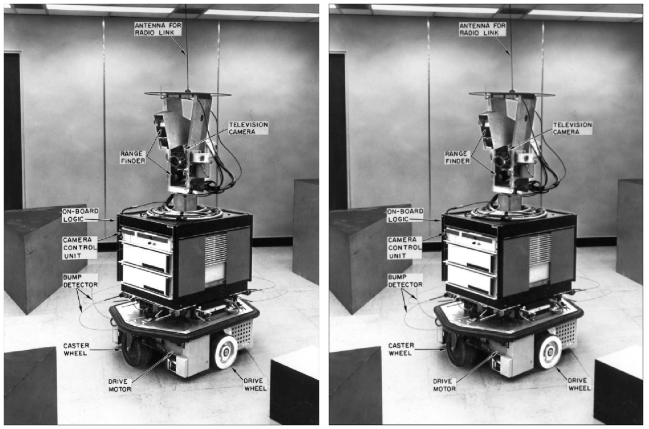
III. Analysis Part 1

A. Step 1

Step 1 asks to experiment with the provided 3x3 and 5x5 *Gaussian* filters.

Question: can you see any difference between them?

Answer: the convolved image with the 5x5 filter is more blurry, thus smoother, than with the 3x3 filter. This is because a larger filter is applied to a larger local area and it makes it more similar to the surrounding pixels because more pixels influence the convolved value than a few specific ones.



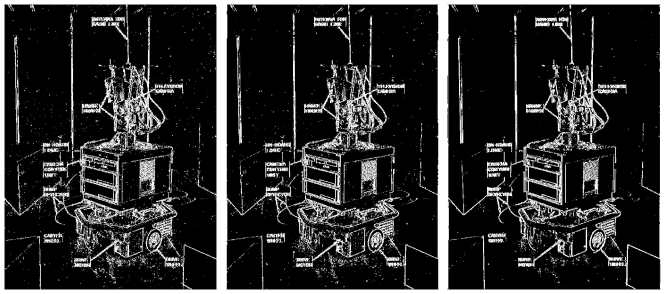
(a) 3x3 Gaussian filter (b) 5x5 Gaussian filter

Fig. 1: Image convolved with different Gaussian filters

This is a solution for the Formative Assignment by Hamid Dehghani
The report was submitted to Canvas on Feb 23 and reviewed by a TA

Task: try applying an edge filter to each and thresholding.

Results: the edge map acquired by applying an edge filter on a 5x5 Gaussian map has less noise. This is due to the fact that the larger filter smoothens the image better than the small filter (i.e., the same reason as above).



(a) SobelX (b) 3x3 + SobelX (c) 5x5 + SobelX

Fig. 2: Pure image and 2 blurred maps from figure Figure 1 convolved with a horizontal Sobel filter and thresholded at 40

B. Task 1

Task 1 asks to compare the pure image convolved with an edge filter with blurred images convolved with an edge filter.

Question: can you describe the effect in comparison with applying the edge filter to the image directly?

Answer: considering the same threshold, if we we apply the edge filter directly on the image, there will be more noise in the acquired map, i.e., visually more false positives are captured as edges. In contrast, smoothening the image with a *Gaussian* filter prior to applying an edge filter averages the noise out and keeps the edges more standing out. However, a trade-off can be seen as well between edge details and noise. The effects can be seen in Figure 2.

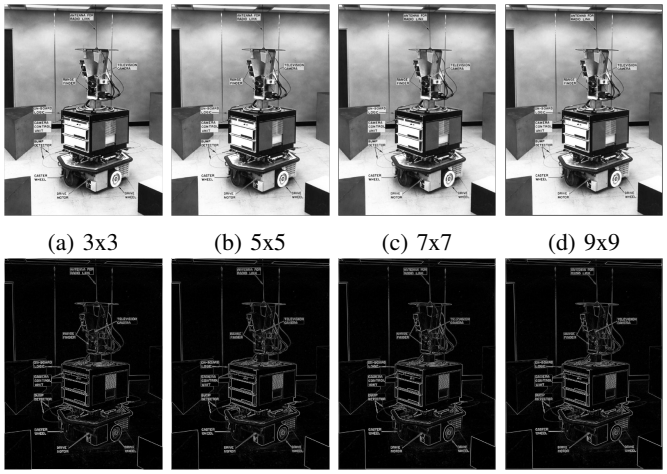
IV. Analysis Part 2

A. Step 2

Step 2 asks to create different Gaussian masks and see what happens to the output as the parameters of the generating function change. It also asks to apply an edge filter and inspect the results.

Question: what happens to the image as you increase the size of the mask? What happens as you increase the size of **s** (σ)?

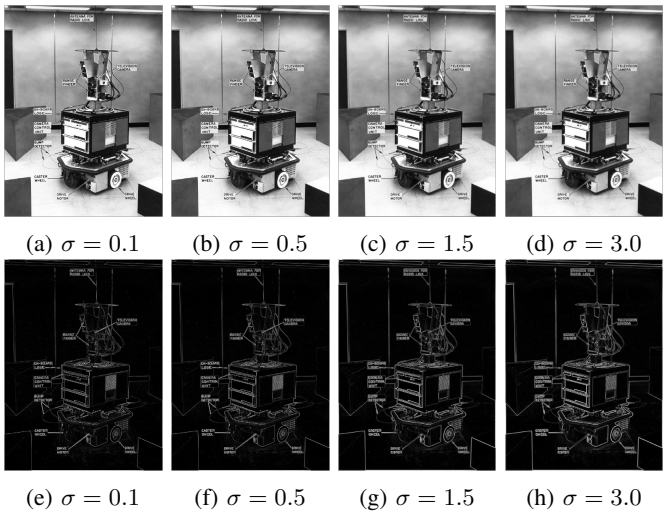
Answer: as we increase the size of the mask, the image becomes more blurry (similar reason explained in subsection III-A) This effect can be observed in Figure 3. As we increase standard deviation, the image also becomes more blurry because the weight values become more spread around the mean, thus surrounding cells play a bigger role in the local area when it is convolved with the kernel. This effect can be observed in Figure 4.



(a) 3x3 (b) 5x5 (c) 7x7 (d) 9x9

(e) 3x3 (f) 5x5 (g) 7x7 (h) 9x9

Fig. 3: Experiments with different Gaussian filter sizes while keeping $\sigma = 1$. The top row contains pure images on which the Gaussian filters were applied. The bottom row contains the top row images but on which the horizontal and vertical Sobel filters were applied and the magnitude was calculated



(a) $\sigma = 0.1$ (b) $\sigma = 0.5$ (c) $\sigma = 1.5$ (d) $\sigma = 3.0$

(e) $\sigma = 0.1$ (f) $\sigma = 0.5$ (g) $\sigma = 1.5$ (h) $\sigma = 3.0$

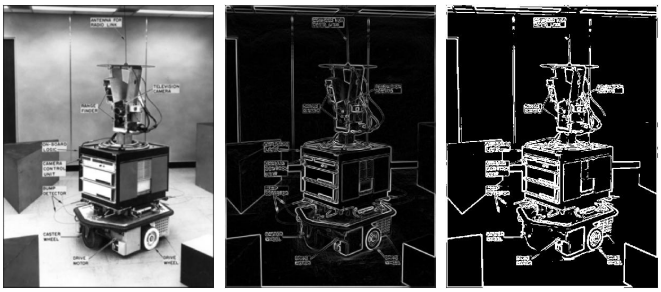
Fig. 4: Experiments with different σ values while keeping the filter size 5x5. The top row contains pure images on which the Gaussian filters were applied. The bottom row contains the top row images but on which the horizontal and vertical Sobel filters were applied and the magnitude was calculated

Task: now apply gradient operators such as the Sobel operators to the blurred images.

Results: the results can be seen in bottom rows of Figure 3 and Figure 4. They compliment the explanation in subsection III-B where a trade-off between edge details and noise was mentioned.

Question: what happens to the edges in the heavily blurred case after applying gradient operators, such as Sobel operators?

Answer: thicker edges stand out more and thinner edges almost disappear. This is because thin edges may sometime be confused with noise which gets averaged out after heavily blurying an image. Strong edges, on the other hand remain standing out therefore the gradient operator captures the changes and emphasizes them.



(a) Gaussian (b) Sobel (c) Threshold

Fig. 5: A heavily blurred image with a Gaussian filter of size 9x9 and $\sigma = 3$. The magnitude of the Sobel operators was computed and the acquired map was thresholded at 40

B. Task 2

Task 2 just asks some reassurance questions to which the explanations could mainly be applied from the previous subsection.

Question: what is the effect of increasing the size of the Gaussian Filter (3x3 versus 5x5 for example)?

Answer: this was answered in subsection IV-A. In the particular case of 3x3 vs 5x5, 3x3 filter leaves more details (thus more noise) than 5x5. So the image with 3x3 filter is less blurry. It can also be seen that, after applying the threshold, there are more false positives in the background captured, but also more true positives within thin edges.

Question: what is the effect of changing the standard deviation **s** (σ)? Why do you see what you see?

Answer: this was answered above in subsection IV-A.

V. Analysis Part 3

A. Step 3

Step 3 asks to check how the performance differs between applying 2 1D *Gaussian* filters (in sequence - first, a horizontal filter over every image row, then a vertical filter over every image column) and 1 2D *Gaussian* filter.

Question: can you detect differences in the CPU times as the mask sizes increase?

Answer: for small matrix size (3, 5 and 7) 1 2D convolution performs faster than 2 1D convolutions. However for the kernel sizes of 9 and above, the time taken to perform the convolution

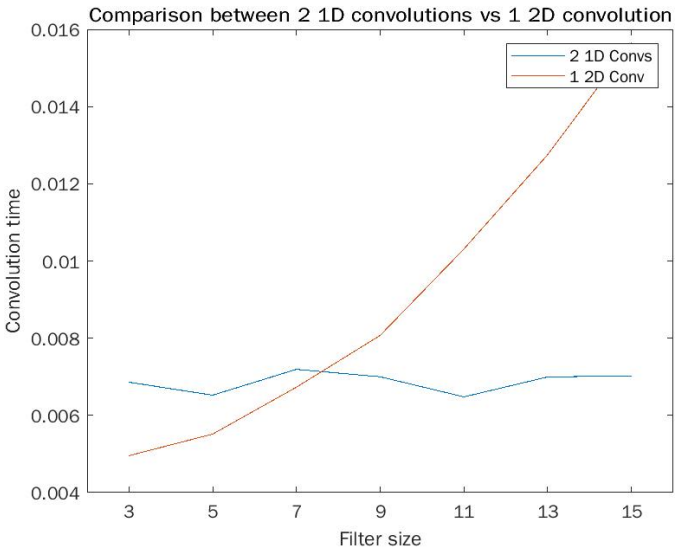


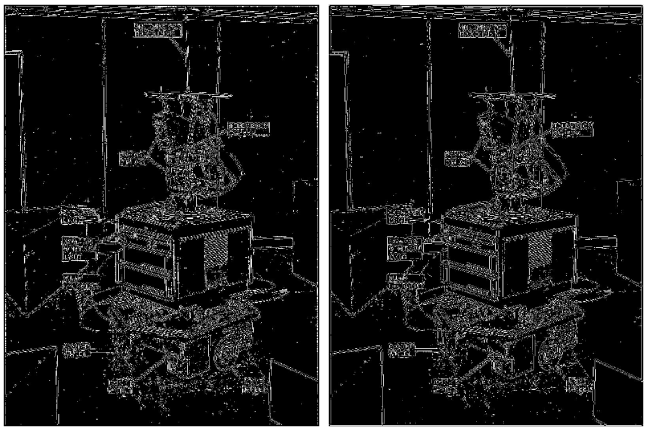
Fig. 6: A comparison of time taken to perform full convolution between applying 2 1D Gaussian filters and 1 2D Gaussian filter

for a 2D case dramatically increases compared to 1D case (e.g., twice the duration when filter size is 15).
Question: are there any effects due to small floating point errors?

Answer: there are very small relative errors, i.e., the range is from 10^{-16} to 10^{-18}. This means the convolution result is almost the same.

B. Step 4

Step 4 asks to produce an edge map after applying a *Laplacian* filter where edges are determined by *zerocrossing*.



(a) Laplacian + zerocross (b) LoG + zerocross

Fig. 7: Second order gradient inspection method to determine the edges: Laplacian is applied to the first image and Laplacian of Gaussian (Laplacian convolved over 5x5 Gaussian filter) to the second one. Edges are determined by zerocrossing

Question: why does it produce a poor result compared to the other operators?

Answer: the poor result is due to finite difference approximation. In other words, it is not always clear where exactly the edge is, it could be one pixel further or closer. It is because zero crossing determines a thin line between pixels with values with opposing signs and the values themselves could be close to 0 meaning higher uncertainty.

C. Task 3

Task 3 asks to produce a **Laplacian of Gaussian** operator and see how it affect the image.

Question: How could you combine the idea of the *Laplacian* operator with the idea of *Gaussian* smoothing?

Answer: first we find the gradient of the *Gaussian* kernel (via convolution with *Laplacian* filter). Then we take the **LoG** filter, apply it to the image and then apply the *zerocrossing* to determine the edges. It still doesn't produce great results due to the reason in subsection V-B. The result and comparison with a pure *Laplacian* can be seen in Figure 7.

REFERENCES

[1] Canny, J., *A Computational Approach To Edge Detection*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 8(6):679–698, 1986.
[2] Leventhal, Daniel (Autumn 2011). "Image processing". University of Washington. Retrieved 2019-12-01.