Final Project: Simplified Gabor Wavelet Edge Detection

Course number: CEG 7580

Student: Ryan Arnold

Date Due: 12/08/21

Date submitted: 12/04/21

Declaration Statement:

Contents

1 Abstract

Many conventional edge detection algorithms are widely implemented and demonstrate accuracy, including the Canny, Sobel, and LoG (Laplacian of Gaussian) methods. One drawback of these methods, however, is that they can only account for limited spatial directional information. Some are also not suited for high-demand algorithms that require high performance for computer vision and artificial intelligence applications. One approach to this problem is to employ a filter using the imaginary component of a Gabor Wavelet. The Gabor Wavelet is a Gaussian function augmented by a sinusoidal component, and accounts for sampling frequency and orientation. This filter alone, however, is not computationally efficient. To combat this, a novel approach proposed by Jiang et al. involves the derivation of a Simplified Gabor Wavelet (SGW), that simultaneously offers detection accuracy and offers computational efficiency [1]. In this report, I attempted to replicate the image processing results presented by Jiang et al., using their novel approach.

2 Technical Discussion

The first step of this algorithm involved computing and quantizing Gabor Wavelet Kernels to create a filter bank. The basis function driving the kernels was the 2D imaginary component of the Gabor wavelet, shown in Equation (1).

$$G(x, y) = exp\left[-\frac{x^2 + y^2}{2\sigma^2}\right] \sin[\omega(x\cos\theta + y\sin\theta)] \tag{1}$$

The Spatial Kernels are computed as a function of orientation (θ), pixel sampling frequency (ω), and Gaussian spread (σ); $f(\theta, \omega, \sigma)$. Jiang et al., determined a set of eight pairs of orientation and frequency for optimal edge detection using the SGW approach. The values of σ can be manually varied and selected by maintaining $\sigma\omega \leq 1$. The pairs of orientation and frequency used for each of the eight kernels are shown in Table 1 below [1].

Table 1: Orientation and frequency based Kernel Pairings.

Kernel #	Orientation (rad)	Frequency (pixel cycle / second)
1	0	0.3π
2	$\pi/4$	0.3π
3	$\pi/2$	0.3π
4	$3\pi/4$	0.3π
5	0	0.5π
6	$\pi/4$	0.5π
7	$\pi/2$	0.5π
8	$3\pi/4$	0.5π

After computing the kernels, they all had to be properly quantized in order to simplify the computations, and to pronounce the presences of step-like features that appear as edges. The method of quantization involved computing the maximum and minimum values of a given kernel, and the magnitude of largest element of the kernel. Given a number of input levels, n_l, the values of the quantization levels could be computed according to Equation (2), where A is the maximum magnitude of the kernel [1].

$$q_+(k) = \frac{A}{2n_l + 1} \cdot 2k \qquad q_-(k) = -\frac{A}{2n_l + 1} \cdot 2k \tag{2}$$

Once all the levels where computed, it was possible to map a quantized value to each region of the kernel using equally spaced points between the levels. Furthermore, there would then only be a total of n_l different values contained in the kernel. Note that the center value is strategically set to zero. See Figure 1 below for a schematic of this with a 1D analog Gabor Wavelet [2].

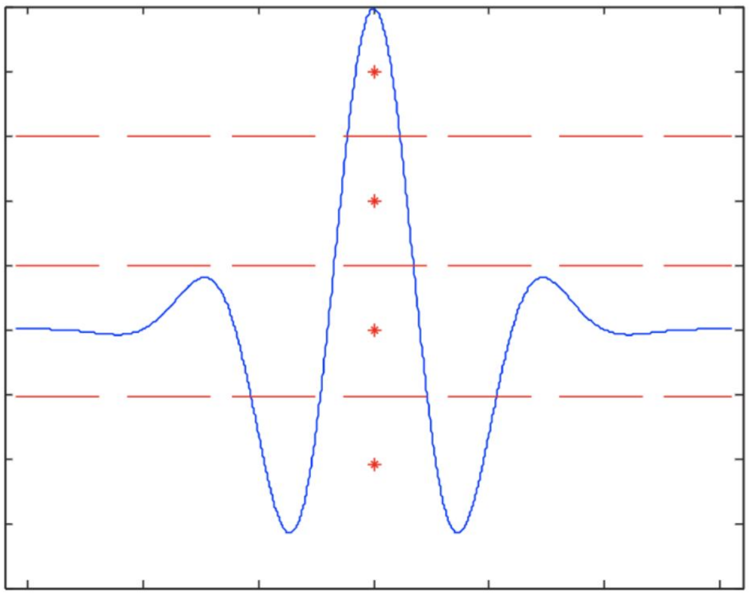

Figure 1: 1D Quantized Gabor Wavelet Scheme.

After fully quantizing the kernels, I obtained the output shown in Figure 2. This replicates Figure 3 in the reference paper [1].

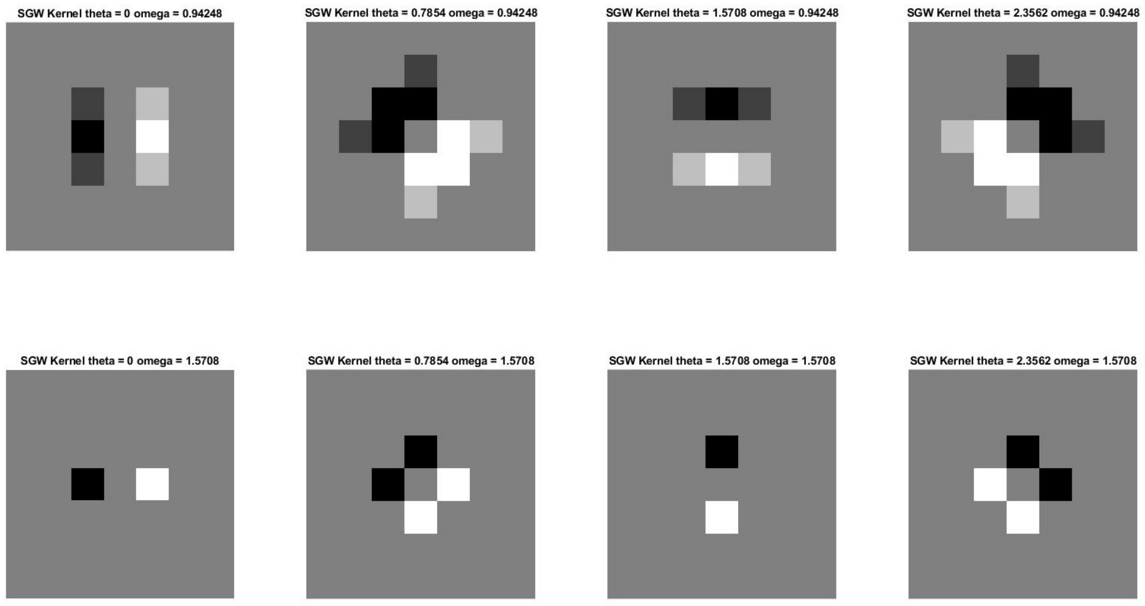
SGW Kernel theta = 0 omega = 0.94248
SGW Kernel theta = 0.7854 omega = 0.94248
SGW Kernel theta = 1.5708 omega = 0.94248
SGW Kernel theta = 2.3562 omega = 0.94248
SGW Kernel theta = 0 omega = 1.5708
SGW Kernel theta = 0.7854 omega = 1.5708
SGW Kernel theta = 1.5708 omega = 1.5708
SGW Kernel theta = 2.3562 omega = 1.5708

Figure 2: Quantized Imaginary Components of Gabor Wavelet at orientation/frequency pairs.

After quantizing all eight of the kernels, each kernel was separately applied to the input images via a spatial convolution, according to Equation (3). The Features produced by the convolutions, denoted by $\phi_{\omega,\theta}$, were then compared to each other at each pixel position, (x_c, y_c). The max pixel across all 8 features was taken at each position, forming a composite feature array, $\phi^{''}_{\omega,\theta}$, according to Equation (4).

$$\phi(x, y) = G(x, y) \otimes I(x, y) \tag{3}$$

$$\phi^{''}_{\omega,\theta} = max\left\{\phi^{'}_{\omega_i,\theta_j}(x_c, y_c), i = 0, 1 \ and \ j = 0, ..., 3\right\} \tag{4}$$

An advantage of using the quantized kernels, is since most of the elements of the kernel are zero, the number of computational operations is greatly reduced. For example, refering to the first kernel in the upper left of Figure 2, at $\omega = 0.3\pi$ (0.94248), the operations are reduced to Equation (5). Note, that since the kernels were symmetric, the magnitude of the positive and negative levels were the same, and are denoted by q_1 and q_2, and differ in sign only.

$$\phi'_{0,0}(x_c, y_c) = q_1(I(x_c-1, y_c+1) + I(x_c+1, y_c+1) - I(x_c-1, y_c-1) - (x_c+1, y_c-1)) \\ + q_2(I(x_c, y_c+1) - I(x_c, y_c-1)) \quad (5)$$

Once the composite feature array was collected, the data had to be thresholded. This was done using a double threshold hysteresis-like scheme. First, two thresholds were selected: T1 and T2. T2 was by default set to be 2T1. T1 was solved using the minimax method. I modified this method, and used a constatn of proportionality to scale the threshold to each image dataset. The minimax algorithm produced by Donoho et al. [3]. The method entailed referencing a precomputed lookup table, which provides a global threshold as a function of N, the number of input samples. The thresholding approach was to select the two greatest feature matrices out of the eight. let the maximum and second maximum be denoted as ϕ'_1 and ϕ'_2. If $\phi'_1 + \phi'_2 > T2$, the pixel position is classified as a strong edge. If both $\phi'_1 < T1$ and $\phi'_2 < T2$, then the pixel position is discarded and set to zero. The remaining cases (at least one greater than T1), are set to weak edges. The resulting matrix (containing either a zero, weak, or strong edge) was linked using a hysteresis approach. If a weak pixel is surrounded by a strong edge within a pre-defined neighborhood mask (I employed an 8-connectivity square neighborhood), it is replaced as a strong edge. If the weak edge is exclusively surrounded by weak edges and zeroes, it is discarded. The result becomes a binary mask, and can be plotted as an edge extracted image. Figure 3 below demonstrates this concept, where the white pixels are strong edges, the gray pixels are weak edges, and the black pixels are zeroes [4].

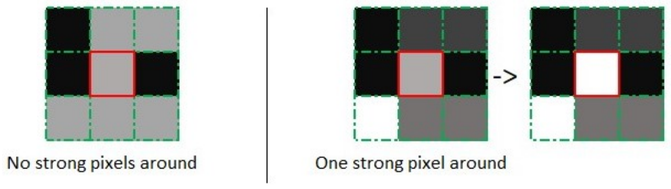

Figure 3: Linking Weak and Strong Edges using Hysteresis Illustration [4].

The aforementioned methodolgy describes the SGW method. The SGW Efficient algorithm outlined by Jiang et al., takes a slightly modified approach. Jiang et al., mention that edges are most commonly detected when $\theta = 0 \; and \; \pi/2$ and $\omega = 0.3\pi$. Therefore, only the kernels corresponding to those conditions were computed at first (i.e., $\phi^{'}_{0,0}$ and $\phi^{'}_{0,2}$, first and third kernels in the top row of Figure 2). The phis were then compared to the thresholds: T1 and T2. If both were less than T1, then the pixel was set to zero, and disregarded as being an edge. If their sum exceeded T2, then the pixel was automatically marked as an edge. If only one was greater than T1, then the other six kernels were considered, and the process was iterated until the pixel was categorized. This made is so that the number of convolutions was reduced by not needing to use all eight kernels in most cases. Both the conventional SGW and efficient SGW approaches are implemented in the SGW.m script included in the project code attachments.

In addition to applying the SGW algorithms to the images, the effects of orientation, frequency, and number of quantization levels were examined. Rather than assembling a filter bank combining all of these effects, filter banks were assembled by varying each effect exclusively. This was useful in seeing how each parameter affected edge assessment. For accuracy and performance reference, other methods were used as a basis of edge detection comparison. The other methods included: Gabor Wavelet Filtering, Canny Method, LoG (Laplacian of Gaussian) Method, and the Sobel Method. Finally, the runtimes of the SGW and SGW Efficient method were computed and compared using Matlab's built-in *tic()* and *toc()* statements.

3 Results

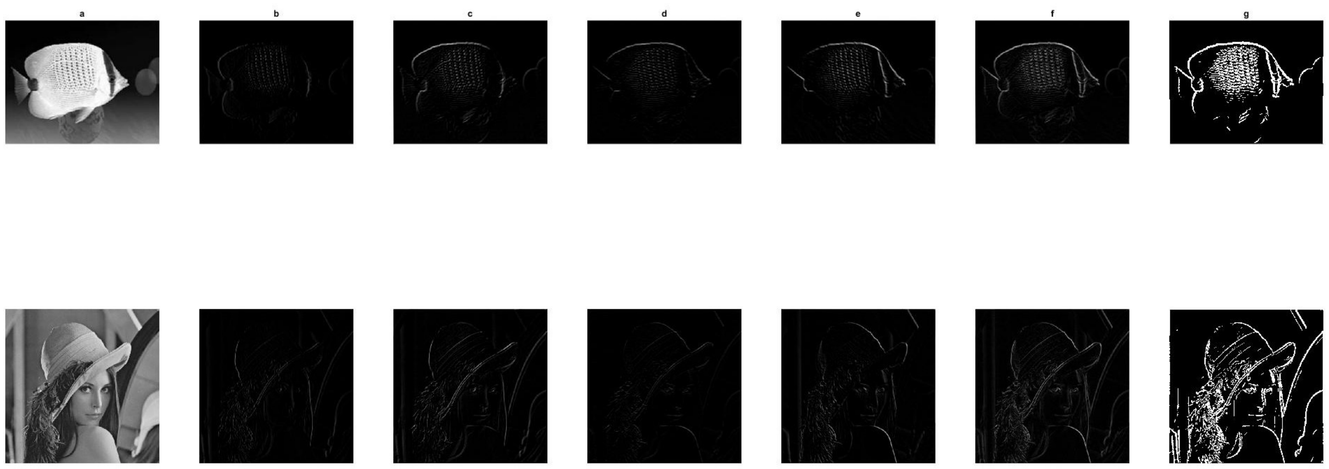

Figure 4: Replication of Figure 4 in Journal Paper. (a) Original Image (b) $\theta = 0$ (c) $\theta = \pi/4$, (d) $\theta = \pi/2$, (e) $\theta = 3\pi/4$,(f) Maximum of SGW Features,(g) Results after Thresholding [1]

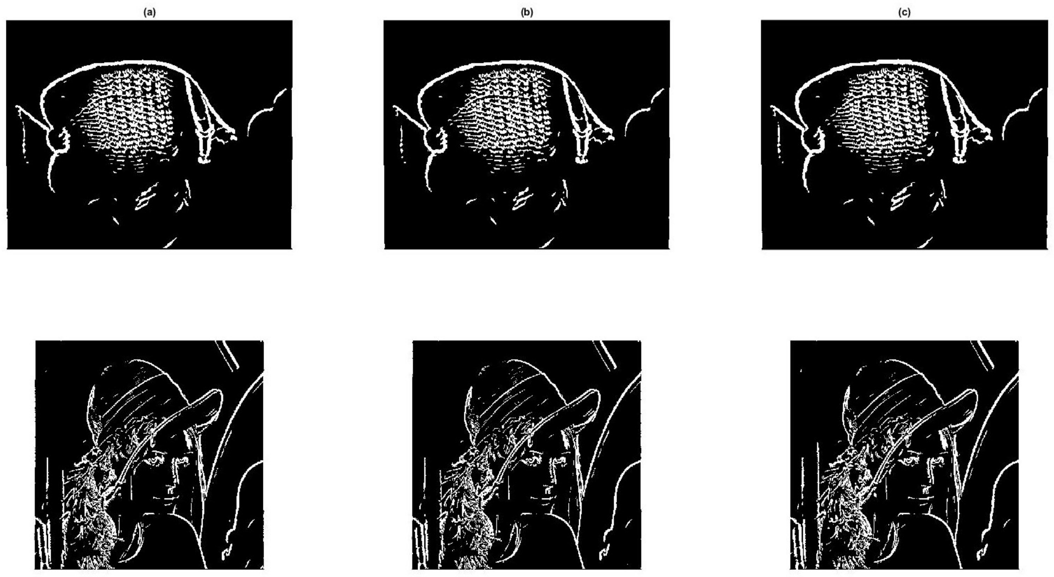

Figure 5: Replication of Figure 5 in Journal Paper. $\omega = 0.3\pi$ (a) 3 quantization levels (b) 5 quantization levels (c) 7 quantization levels [1]

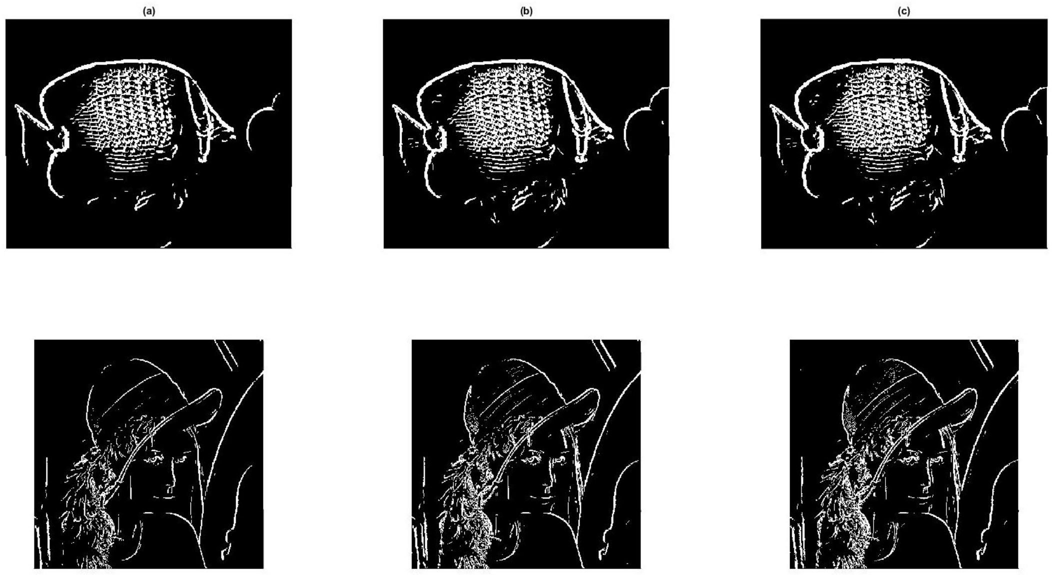

Figure 6: Replication of Figure 6 in Journal Paper. Examination of number of levels and frequency. $\omega = 0.5\pi$ (a) 3 quantization levels (b) 5 quantization levels (c) 7 quantization levels [1]

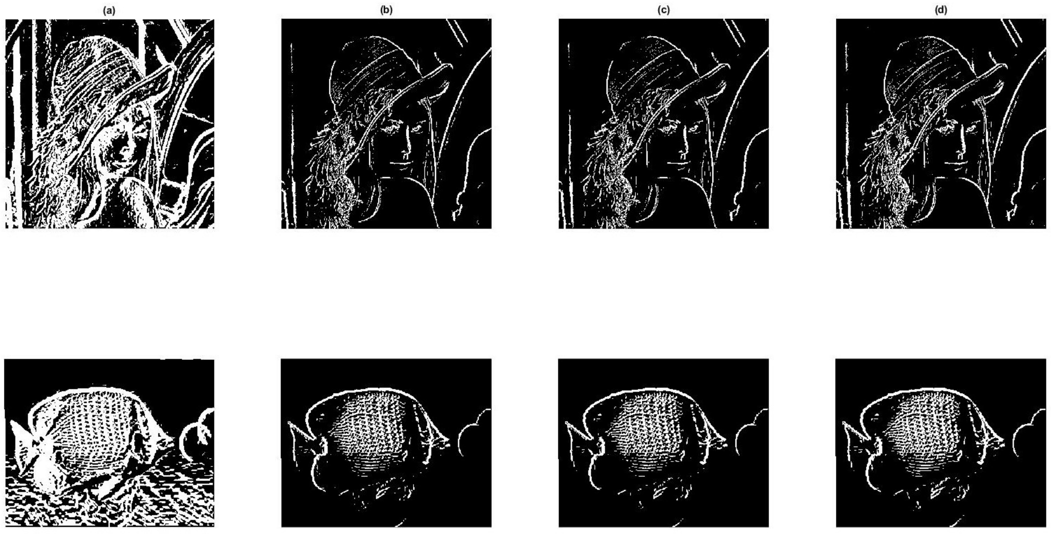

Figure 7: Replication of Figure 7 in Journal Paper. Examination of frequency parameter. (a) $\omega = 0.125\pi$ (b) $\omega = 0.3\pi$ (c) $\omega = 0.5\pi$ (d) $\omega = 0.625\pi$ [1]

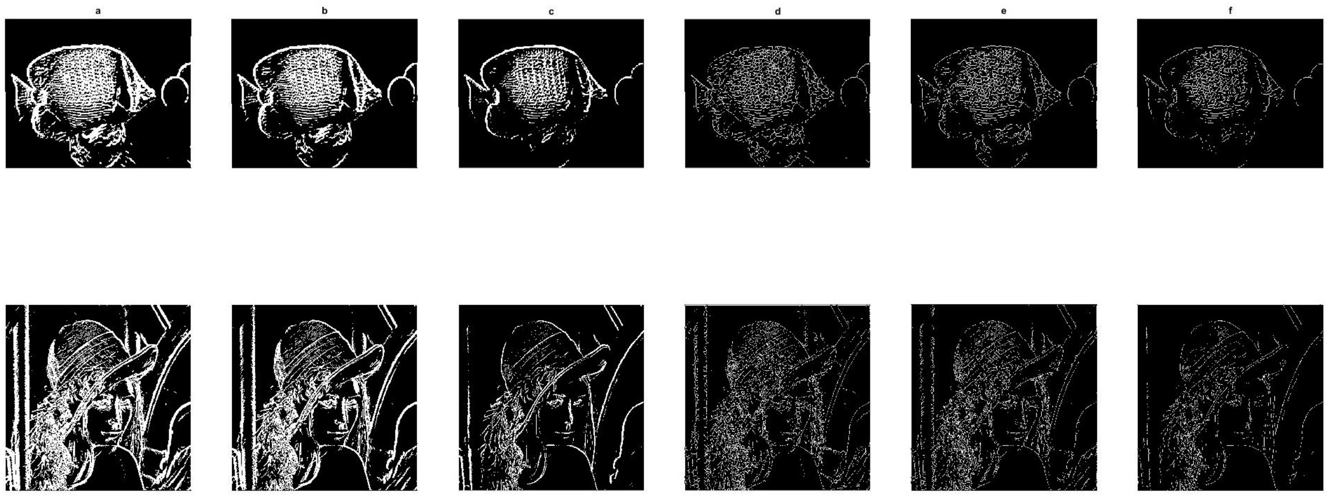

Figure 8: Replication of Figure 8 in Journal Paper. Examination of frequency parameter. (a) GWs (Gabor Wavelets) (b) SGWs (c) SGWs efficient (d) GWs thinned (e) SGWs thinned (f) SGWs efficient thinned. [1]

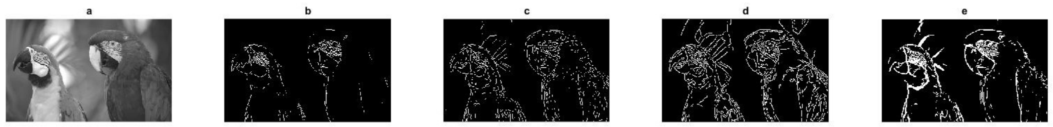

Figure 9: Replication of Figure 10 in Journal Paper. Comparison to other methods (Parrots). (a) Original Image (b) Sobel Edge Detector (c) LoG Edge Detector (d) Canny Edge Detector (e) SGW Edge Detector. [1]

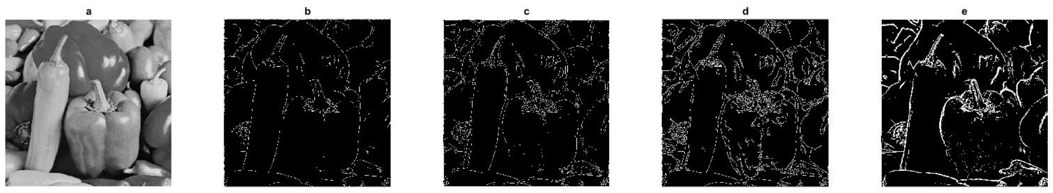

Figure 10: Replication of Figure 10 in Journal Paper. Comparison to other methods (Peppers). (a) Original Image (b) Sobel Edge Detector (c) LoG Edge Detector (d) Canny Edge Detector (e) SGW Edge Detector. [1]

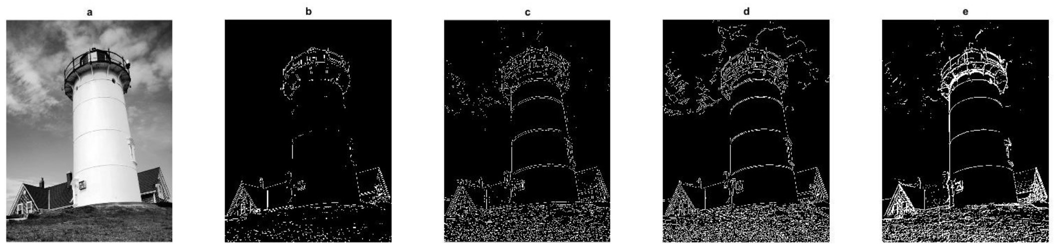

Figure 11: Replication of Figure 10 in Journal Paper. Comparison to other methods (lighthouse). (a) Original Image (b) Sobel Edge Detector (c) LoG Edge Detector (d) Canny Edge Detector (e) SGW Edge Detector. [1]

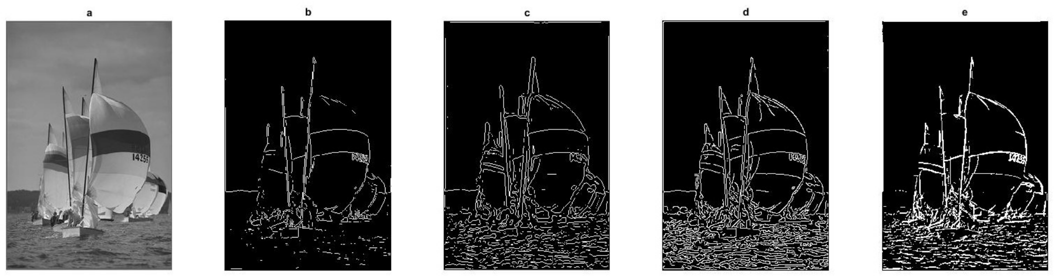

Figure 12: Replication of Figure 10 in Journal Paper. Comparison to other methods (Sailboat). (a) Original Image (b) Sobel Edge Detector (c) LoG Edge Detector (d) Canny Edge Detector (e) SGW Edge Detector. [1]

Table 2: Runtime Analysis Results, modeled after Table III [1].

Algorithm #	3 quant. levels (s)	5 quant. levels (s)	7 quant. levels (s)
SGW (without efficient scheme)	6.032	6.0339	6.019
SGW (with efficient scheme)	5.4248	5.4075	5.4198
GW		5.9533	
Percent Increase GW -> SGW eff.	8.8777 %	9.1678 %	8.9621 %

4 Discussion

The primary Challenge in replicating the results produced by Jiang et al., was the fact that many of their parameters were merely suggestive, and not explicitly defined. The two main examples of this were the selection of σ (Gaussian spread) and $T1$. Since an inequality was provided ($\sigma\omega \leq 1$), I was able to deduce proper values for σ. However, for $T1$, they only mention they used the minimax method. After investigating the minimax method, it entails returning a parameter (λ^*) [2] as a function of the number of samples. Although, how this parameter is to be used as a threshold was unclear. There were no units attached to the parameter, and its magnitude was higher than all of the filter bank features ($\phi_{\omega,\theta}$) that I computed. I made a compromise by instead scaling to the maximum value of the maximum feature ($\phi''_{\omega,\theta}$). I defined a proportional constant (α), and then applied it to the max of the feature maximum to retrieve $T1$. Through trial and error, I decided on using a value of $\alpha = 0.04$. This obviously deviated from the minimax method, which likely caused slight deviation in my results compared to theirs [1]. However, from my observation, these deviations were minimal.

Figure 4 demonstrates how the SGW algorithm is able to discriminate between different orientation features within the input images. It was evident from the results that there were edges that spanned the four orientations examined. Figure 5 and 6 demonstrated how the number of quantization levels affected the results. The tradeoff of quantization levels is that as the number of quantization levels increases, the accuracy increases at the expense of computational load. The selection of the number of quantization levels depends on the application. Both Figures 5 and 6 show that as the quantization levels increase, the edges become more granular and detailed. Figure 7 examines how the frequency parameter influenced the results. Since frequency (ω) is representative of the pixel sampling frequency, it was expected that as the frequency is increased, more detail would be extracted in the edges. Going from left to right in Figure 7, this was exactly the case. In the the extreme case of $\omega = 0.125\pi$, the edges were smudged and too thick. Increasing the frequency made the edges thinner and subsequently more distinct. Figure 8 shows the three different Gabor Wavelet approaches, coupled with the results of applying morphological thinning to the images. Note that Jiang et al.'s thinning method was not defined, so I used the built-in Matlab methods. Hence, my thinned results are slightly different than theirs. Also, the results of my GW method compared to SGW were also slightly different. I attribute this to the fact that I was not able to replicate their exact approach of using an fft/ifft and transfer function method [1]. I instead applied a spatial convolution, and I attribute this to causing some dissimilarity. It is also important to note that since the SGW methods are quantized, they are going to be inherently different from the non-quantized GW approach. Figures 9 - 12 provide a summary of comparing the SGW approach to conventional reference methods (Canny, LoG, Sobel, and traditional GW filtering). Overall, most of my figures closely resemble those in the reference paper, with the exception of a few discrepancies that I attempted to explain [1].

The final piece of this project entailed performing a runtime analysis of the different algorithms. This was the most difficult component to replicate, because there were so many factors that made it difficult to perform parallel comparisons. For example, the reference paper was written in 2009, so the hardware was most likely drastically different. It very well could have meant that my hardware may have had negligible runtime differences between methods due to the hardware alone. Another issue I encountered was that I was using the Matlab *nlfilter()* function. This function produced a gui that displayed a loading bar. The instancing of a graphical object may have confouned the runtimes. In fact, I ommitted the Canny runtime because the built-in Matlab implementation did not have a loading bar, and in general, seemed to be more optimized than my code. I realized this when the Canny runtimes were orders of magnitude smaller. For the GW, SGW, and SGW Efficient results, I programmed them very similarly. In spite of omitting the Canny runtimes, according to Table 2, I was able to show that the SGW Efficient scheme improved the runtime over the GW filtering method by about 9%. My results were not the same as in [1] for the runtime, so to get more consistency with that, I would have to reassess my approach of measuring the runtime and the programming environment. Nevertheless, I still demonstrated that the efficient SGW scheme improved runtime.

References

[1] W. Jiang, K. Lam and T. Shen, "Efficient Edge Detection Using Simplified Gabor Wavelets," in IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), vol. 39, no. 4, pp. 1036-1047, Aug. 2009, doi: 10.1109/TSMCB.2008.2011646.

[2] W. P. Choi, S. H. Tse, K. W. Wong, and K. M. Lam, “Simplified Gabor wavelets for human face recognition,” Pattern Recognit., vol. 41, no. 3, pp. 1186–1199, Mar. 2008.

[3] Ideal Spatial Adaptation by Wavelet Shrinkage Author (s): David L . Donoho and Iain M . Johnstone Published by : Oxford University Press on behalf of Biometrika Trust Stable URL : https://www.jstor.org/stable/2337118 REFERENCES Linked references are ava. (1994). 81(3), 425–455.

[4] Sahir, S. (2019). Canny Edge Detection Step by Step in Python — Computer Vision | by Sofiane Sahir | Towards Data Science. Towardsdatascience, 1–15. https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123.

5 Appendix

Program Listings

Script File Listing:

Project Root

Main.m

gen_ figures.m

bookeeping

> Filter.m

> gen_ file_ list.m

> gen_ params.m

> init_ out_ struct.m

> parse_ inputs

> Scheme.m

filtering

> apply_ GW_ filt.m

> apply_ paper _ methods.m

> apply_ ref _ methods.m

> double_ thresh.m

> GaborImag.m

> apply_ paper _ methods.m

> gen_ inter_ data.m

> GenGWBank.m

> GetThresholds.m

> hysteresis_ link.m

> hysteresis3d.m

> minimax.m

> QuantizeWavelet.m

> SGW.m

fourier

> freq_ filter_ image.m

> get_ freq_ transfer_ fun.m

> parse_ inputs.m

image_ handling

> load_ image.m

> LoadImages.m

> shift_ image _ values.m

inputs

> find_ files_ from_ pattern.m

> fish_test.tif

> Lenna.tif

> lighthouse.tif

> parrots.tif

> peppers.tif

> sailboat.tif

Instructions to Run Scripts

The most important detail in setting up this project to be functional is to ensure that all of the supplied image files are stored in the same directory as the accompanying scripts. The algorithms assume that the files will be in the same directory to run properly. The Main.m script calls all routines in the same script. Therefore, it is recommended to run the Main.m script to produce all of the figures at once. The main concepts of the paper are implemented in the scripts listed under the **filtering** sub-section above.