Bayesian Deblurring with a Known Gaussian Filter

Abstract – A Bayesian algorithm was made to deblur an image that has been blurred with a known Gaussian filter. Bayes theorem is formulated using per-pixel Gaussian Difference and a Gaussian Neighborhood prior to introduce dependencies between neighboring pixels. The formulation is then solved using Metropolis Hastings MCMC. Finally, we look at the results under different blur conditions and make comparisons to existing deblurring algorithms.

Introduction

Images can often end up blurred unintentionally. We would like to de-blur the image and regain the original information. The general problem of deblurring is extremely difficult as the type of blur, the size of the blur, or any information about how that image was produced is unknown. A simpler version of this problem can be solved where the original image is blurred with a known Gaussian Blur. To further simplify the problem, a grayscale image is used instead of three color dimensions.

Method

Fundamentally, an image is burred by convolving it with a blur kernel. This kernel can vary in size but in this case, it will be a 2D Gaussian of the form:

$$N(x, y, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

To visualize this kernel, the 2D Gaussian function with a σ of one is sampled over a range of $\pm 3\sigma$, to produce the kernel image in Figure 1.

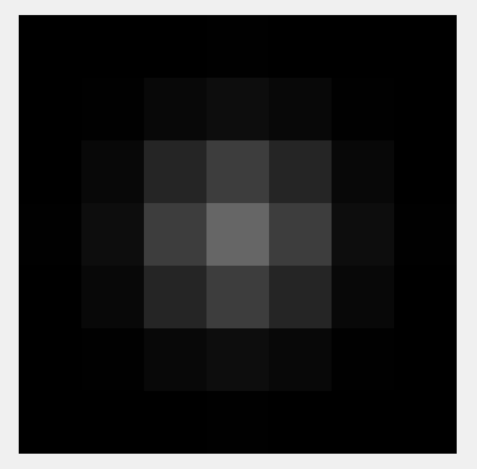

Figure 1: Gaussian Kernel

Figure 1 represents the blur kernel to be convolved with a clear image. The forward function of blurring where K is the Gaussian Blur Kernel, Y is the original clear image, and X is the final blurred image is written here:

$$X = Y * K$$

Figure 2: Clear Image (Left), Blurred Image (Right)

Figure 2 shows the clear image on the left and a blurred image on the right with a blur kernel of size 9x9, and a sigma of 2. The goal of this project is to find the original image Y given a kernel K and a blurred image X. Formulating this as a Bayes' problem appears as:

$$p(Y \mid X, K, I) = P(Y|I) * \frac{p(X|Y, K, I)}{Z}$$

Given the blurred image X, the kernel K, and any other prior information, the odds of that hypothesized image Y can be found. The right side include the prior for a hypothesized image, the likelihood based on the input blurred image, and the normalizing evidence term Z. This formula may seem simple but due to the images being matrices with 100's of rows and

columns, this makes the problem explode in dimensionality. One solution is to reframe the problem around a single pixel where i and j index the row and column.

$$p(Y_{ij} \mid X_{ij}, K, I) = p(Y_{ij} | I) * \frac{p(X_{ij} | Y_{ij}, K, I)}{Z_{ij}}$$

Doing this does have drawbacks though. By treating each pixel as an independent atomic object, we lose sight of the bigger picture, literally. Pixels are not independent and they do have important relationships with each other. This will be addressed by adding back some dependence between pixels in the prior. Now that the form of the problem is stated mathematically, the right hand side functions must be defined.

Likelihood Function

For simplicity, a Gaussian likelihood is used. This method compares the hypothesized pixel value and the blurred pixel value. By taking the difference of the two, an error quantity is found and input into a standardized Gaussian centered at zero. So an error of zero means the pixel has the highest probability of being correct with the odds tapering off at the error increases in magnitude.

$$p(X_{ij} | Y_{ij}, K, I) = \frac{1}{\sqrt{2\pi}\sigma_L} \exp\left(-\frac{(X_{ij} - (K * Y)_{ij})^2}{2\sigma_L^2} \right)$$

First a hypothesis image generated and blurred with the known blur filter. If the guess image was the original image, it would blur exactly to X and maximize the likelihood. One problem arises from the fact that more than one hypothesized image can maximize the likelihood and some of these images will not be correct. To shrink the search space, a prior function is used.

Prior Functions

Choosing a prior for an image is a lot less simple. It can be one of the most important steps when there are so many possible hypothesized images. Reading into some of the literature shows a variety of priors. Most methods try to find commonalities in images that some hypothesized image should have. These generally use broad image datasets to find generalized image statistics. For example, [1, 2, 3] uses image gradient statistics from an image dataset to produce an informed prior. This is how a single pixel can become aware of the whole image. This concept can be done directly using a neighborhood prior found in [4]. This method compares some hypothesized pixel to its neighbors above, below, left and right. The idea is that pixels close to each other are often similar. A prior can be defined that takes the different between these pixels and correlates a probability with that error.

$$p(Y_{ij}|N_{ij}, I) \propto \exp\left(-\frac{1}{2\sigma_P^2}\sum_{m,n\in N_{ij}}(Y_{ij}-Y_{m,n})^2\right), \qquad \text{where } N_{ij} \text{ is the neighborhood of } Y_{ij}$$

In this case, a Gaussian is used though any symmetrically increasing function around Y_{ij} could work, such as the absolute value.

The complete Bayes theorem with the evidence and Gaussian Coefficients taken as a constant of proportionality results in:

$$p(Y_{ij} \mid X_{ij}, K, I) \propto \frac{1}{\sigma_L^2\sigma_P^2}\exp\left(-\frac{1}{2\sigma_P^2}\sum_{m,n\in N}(Y_{ij}-Y_{m,n})^2\right) * \exp\left(-\frac{(X_{ij}-(K*Y)_{ij})^2}{2\sigma_L^2}\right)$$

The sigma's of each Gaussian are chosen through experimentation resulting in $\sigma_L = 1, \sigma_P = 100$. A lower sigma means a larger uncertainty or a lower cost to guessing wrong. The likelihood was the most important factor in making sure the image generated actually produced a blurred image that matched the given blurred image. The likelihood alone cannot make the image clearer as it would produce images that blur properly but look random to the eye. This is where the softer guidance of the prior helped. It was found that a larger sigma helped. Keep in mind that the pixel values are in between 0 and 255. A sigma of 100 allows for decent variation in the pixel value or about 40% of the possible range. Conversely, a sigma of one would mean that a deviation of pixel intensity of 2% of its range would be nearly impossible leading to negligible changes.

Numerical Technique

Despite simplifying the problem, solving Bayes' Theorem analytically would be difficult if not impossible. The prior's dependence with neighboring pixels makes this extra difficult as well as needing to do this for every pixel in a given image. For this reason a Markov Chain Monte Carlo technique is used to approximate the posterior distribution for each pixel. The technique used will be the Metropolis-Hastings algorithm. This algorithm works by slightly modifying a previous guess, testing if the new guess is better, and then accepting this with some uniform probability. Imagine a given hypothesized pixel value is y. First, a sample from a normal distribution scaled by some step size (s), is taken to produce a guess (g).

$$g = y + s \cdot \mathcal{N}(x)$$

The step size determines how fast pixels values are explored. The larger it is, the faster different values are attemped, but the easier it is to jump past an optimal value. The smaller it is, the a precise guess can be made to help find the optimal peaks, but the easier it becomes to get stuck in a local optimum.

For a given guess the acceptance ratio (AR) is found which is simply the ratio of the posteriors. As the evidence is a constant across all hypthoeses, it cancels out simplifiying the process:

$$\text{AR} = \frac{p(g|I) * p\left(X_{ij}|g, K, I\right)}{p(y|I) * p\left(X_{ij}|y, K, I\right)}$$

This is clamped between 0 and 1 with $min(1, AR)$

If the AR is closer to one the guess is better, and opposite for an AR closer to zero. So far, this procedure will be able to climb to the nearest optimal value for each pixel. This is a problem since we it could get stuck at a local optimum instead of finding the global one. To avoid getting stuck, the algorithm must accept worse guesses some of the time to move past local optimums. This is implemented with randomness by sampling from the uniform distribution. If this sample is less than the clamped Acceptance Ratio the guess is accepted. A better AR closer to one has a high chance of being accepted but could be forgone.

$$\text{Accept} = U[0,1] < \min(1, AR)$$

Producing the Final Image

As the algorithm runs, the error reaches an asymptote and a general image structure is found. Due to the random nature of a Markov Chain Monte Carlo Algorithm, each pixel will still have some small acceptance rate which produces noisy changes over every iteration. Because the Markov Chain approaches the posterior, the mean of the algorithm's samples can be taken to approximate the mean of the posterior. This will eliminate the noise of the algorithm to produce a clearer final image. Note, the initial guess is seeded with the blurred image to avoid a complete reconstruction from nothing. To exit the algorithm's loop, a maximum number of iterations of 100,000 is chosen which was large based on experimentation. Of course the algorithm could settle at a solution way before the maximum number of iterations. The RMS Error of each iteration is used as an error metric. If the RMS isn't changing significantly between iterations, the algorithm will end.

$$\text{RMS Error} = \sqrt{\frac{1}{i \cdot j}\sum_{j}\sum_{i}\left(Y_{ij}\right)^2}$$

This is done by measuring the difference in the RMS between every 1000 iterations to avoid local variations between iterations. Experimentally, a difference of less than 0.05 is considered to be settled and will break the loop. As a final note, the RMS error often spikes up, the goes down in the first thousand iterations (See Figure 6). The initial error spike is just the Markov Chain "burning in" approaching the posterior. As a result the initial error and some error 1000

iterations later could be similar but doesn't represent a real solution. To avoid breaking the loop early, a 2000 iteration grace period is used to allow the algorithm to explore the search space.

Results

The algorithm is tested for a series of different images and blur sigma's. To start, a Gaussian blur that is 9 by 9 with a sigma of 2 used. In Matlab, a Gaussian Blur Kernel is sized as $2(2\sigma) + 1$, which makes the kernel an odd size and extends to 2σ of the Gaussian. The evolution of an image can be observed in Figure 3, Figure 4, and Figure 5. These represent a snapshot of the algorithm at 250, 1000, and 5000 iterations. Early on, the algorithm has got the general image completed already as it is seeded with the blurred image, but it is quite noisy at this point. As the algorithms settles out the noise averages away and the images become clearer for 1000 and 5000 iterations. These figures also show the blurred guess image which is what the likelihood function compares with. The difference of the blurred images in the bottom right of each figure. It is mostly noise as the algorithm has gotten the structure correct already. In Figure 6, the error's progress over the iterations is shown. It spikes in the beginning as the numerical algorithm burns in. As the algorithm further progresses the RMS Error drops until it settles. Due to the log axis, the last tail appears to have a more dramatic slope than it really does.

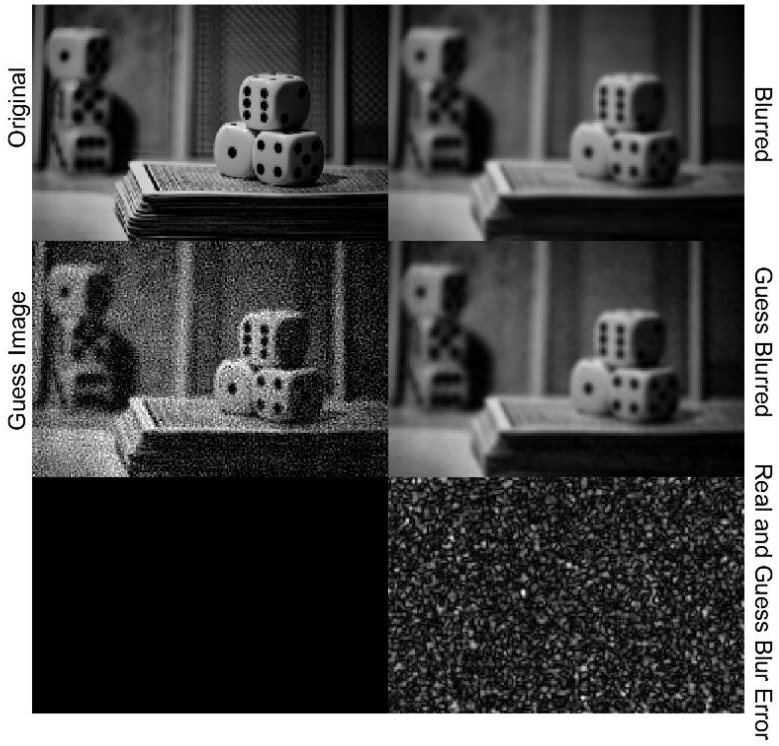

Figure 3: Algorithm at 250 iterations

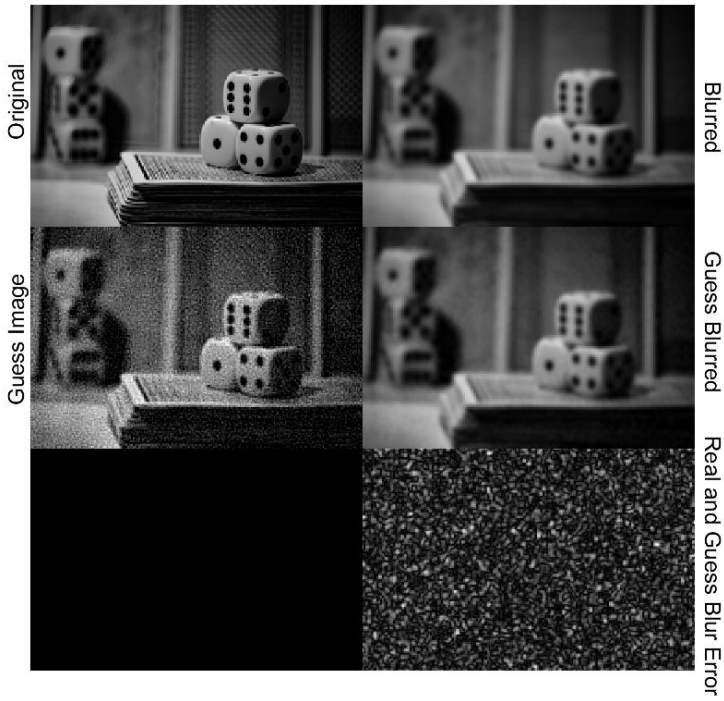

Figure 4: Algorithm at 1000 iterations

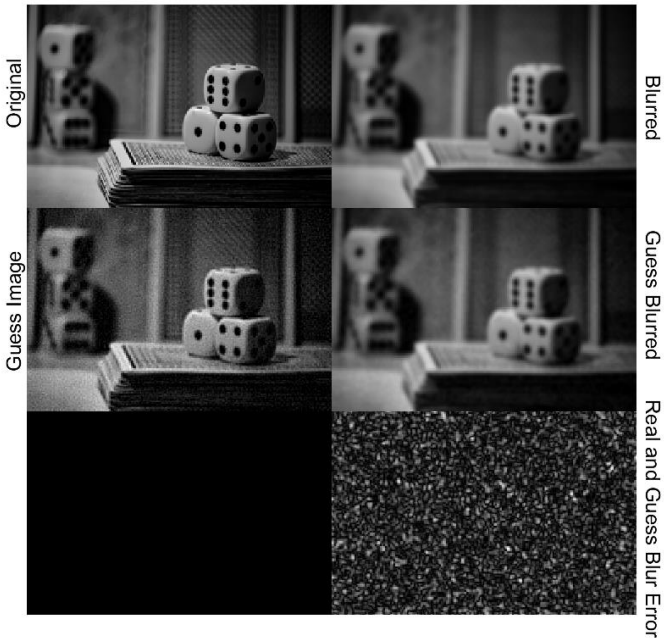

Figure 5: Algorithm at 5000 iterations

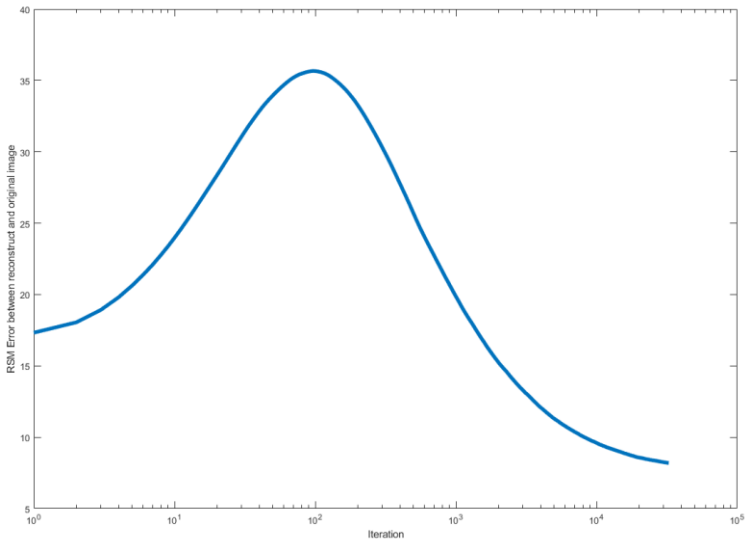

Figure 6: Plot of the RMS error over the number of iterations (log plot)

Figure 7 and Figure 8 has a table of different test images. The top row represents the original clear image, the second row represents the original blurred image, and the final row is the reconstructed image. Under each column, the RMS Error between the original and reconstructed image is shown, and the number of iterations it took to complete. Visually, the reconstructed images have more details than the blurred but still aren't as detailed as the original. The reconstructed images also have some noise in them. Despite this, the reconstruction clears the image up nicely. The initial and final RMS Errors in Table 1 demonstrate this. Initial being the error between the correct and blurred image, and final being the error between the correct and reconstructed image.

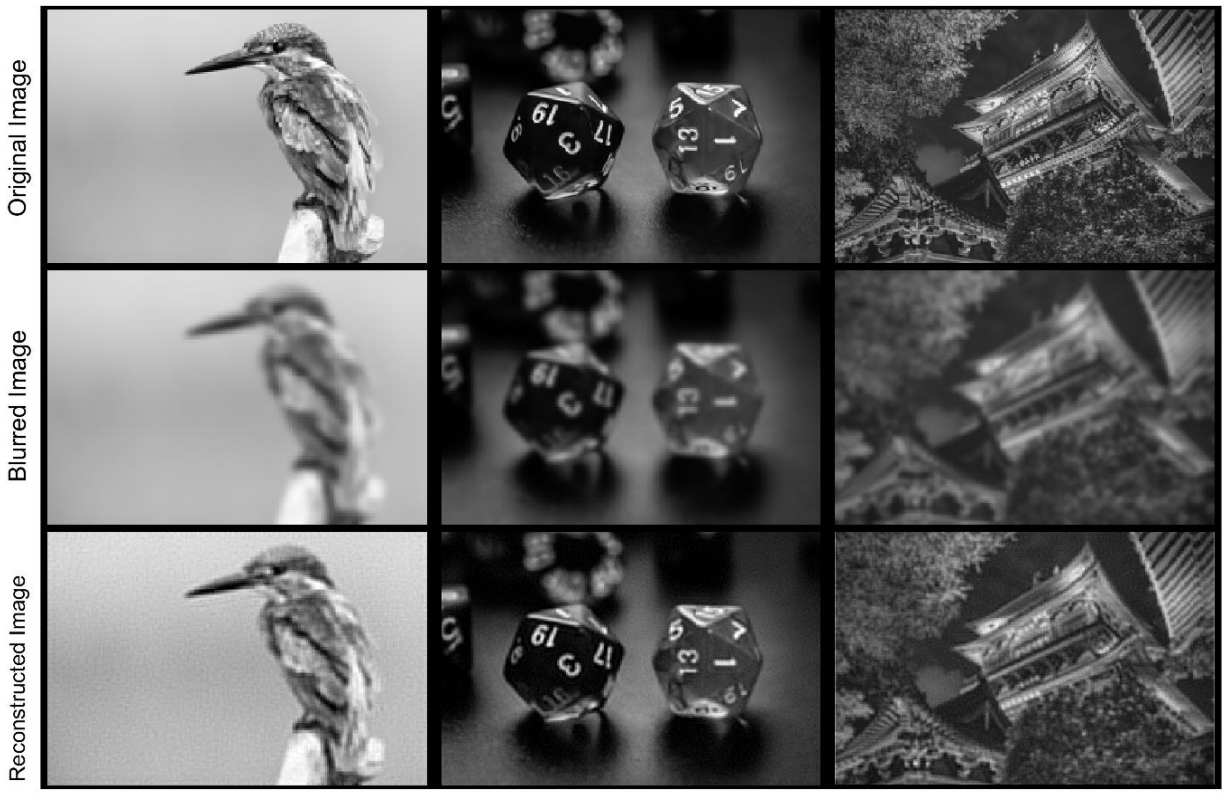

Figure 7: Image Set 1 containing the original, blurred, and reconstructed images.

Figure 8: Image Set 2 containing the original, blurred, and reconstructed images

Table 1	**Bird**	**Two Dice**	**Pagoda**	**Dice Rolled**	**Dice Stacked**	**Colosseum**
RMS Blurred	13.84	14.05	18.98	14.66	13.07	13.87
RMS Reconstructed	8.85	7.43	13.23	9.25	8.29	9.71

To get a more quantifiable understanding of how this algorithm performs it is compared to existing deblurring/deconvolution algorithms. Matlab has a few built in including a Blind Deblurring Algorithm, the Lucy-Richardson Algorithm, and another Regularized Filter Deconvolution algorithm. Figure 9 shows each algorithms reconstruction including this reports', and the far left shows the original image. The bottom row shows the difference between the reconstruction and original image, amplified so the differences can be seen. Finally, the RMS Error is displayed on the bottom. Our algorithm performs on par with existing ones, and in this particular case, actually performs the best.

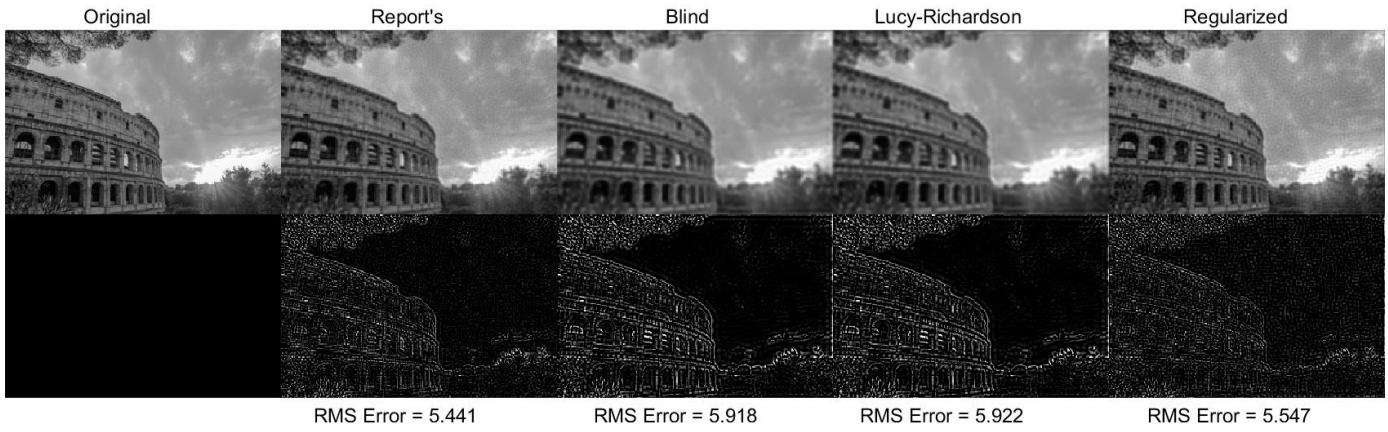

Figure 9: Comparison of Matlab deblurring algorithms to this report's. The top row is the reconstructed image. The bottom row is the exaggerated difference between reconstructed and actual.

The same tests are rerun with a 5x5 kernel and a sigma of one to see how it performs at lower blur levels. Again Figure 10 and Figure 11 show the original, blurred, and reconstructed images with the RMS and number of iterations at the bottom. Because the blur is weaker, a lot more detail are in the final images which is why the RMS Error is much smaller. The algorithm isn't perfect and as information is lost during the blur, the reconstruction is still somewhat noisy and less detailed. Same as before, the initial and final RMS values are in Table 2 below.

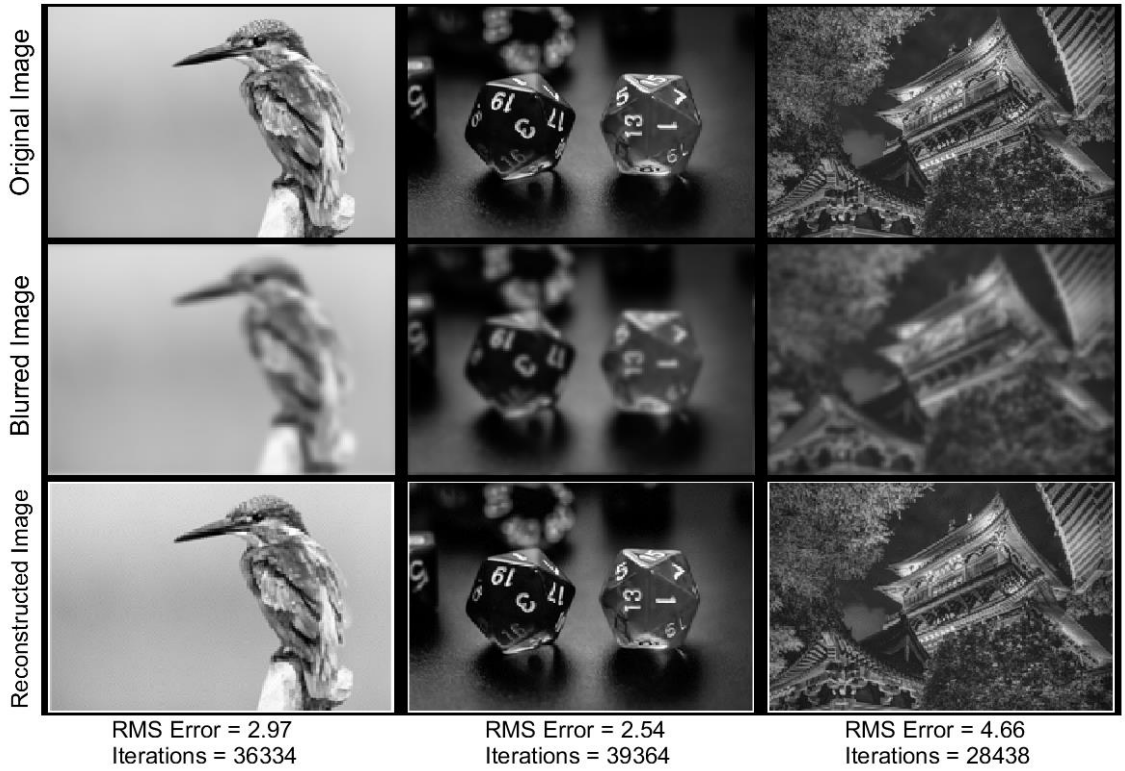

Figure 10: Image set 1 of the original image, blurred image and reconstructed images.

Figure 11: Image set 2 with original image, blurred image, and reconstructed image

Table 2	**Bird**	**Two Dice**	**Pagoda**	**Dice Rolled**	**Dice Stacked**	**Colosseum**
RMS Blurred	8.60	7.96	13.17	9.10	8.09	9.29
RMS Reconstructed	2.97	2.54	4.66	2.97	2.73	3.37

The same comparisons as in Figure 9 here in Figure 12. In this case, the report's algorithm is on par with existing solutions, though doesn't perform the best.

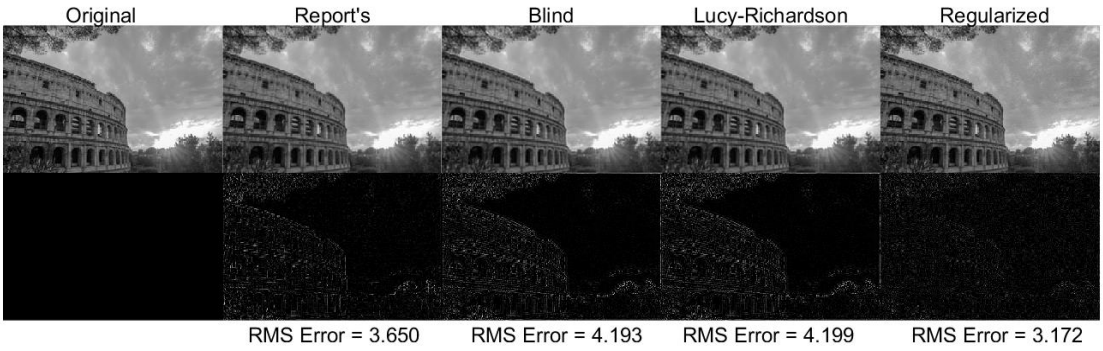

Figure 12: Comparison of this report's algorithm with other Matlab Deblurring algorithms

Conclusion

A Bayesian formulation of the deblurring problem was created with known Gaussian blur. Using Metropolis-Hastings to iterate all of the pixels, pixel posteriors were approximated. The results of the deblurring worked well as measured by the RMS Error tested on six images. Visually, the images had more details than the blurred but still has some noise and blur compared to the true original image. Comparing the algorithm to existing deblurring algorithm found this report's algorithm to be comparable in performance using the RMS Error metric.

References

[1] H. Yang, X. Su, J. Wu, and S. Chen, “Non-blind image blur removal method based on a Bayesian hierarchical model with hyperparameter priors,” *Optik*, vol. 204, p. 164178, Feb. 2020, doi: 10.1016/j.ijleo.2020.164178.

[2] D. Krishnan and R. Fergus, “Fast Image Deconvolution using Hyper-Laplacian Priors,” in *Advances in Neural Information Processing Systems*, Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta, Eds., Curran Associates, Inc., 2009. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2009/file/3dd48ab31d016ffcbf3314df2b3cb9ce-Paper.pdf

[3] A. Levin, R. Fergus, F. Durand, and W. Freeman, “Deconvolution using natural image priors,” *Mass. Inst. Technol. Comput. Sci. Artif. Intell. Lab.*, [Online]. Available: https://groups.csail.mit.edu/graphics/CodedAperture/SparseDeconv-LevinEtAl07.pdf

[4] J. Besag, “Digital Image Processing: Towards Bayesian image analysis,” *J. Appl. Stat.*, vol. 16, no. 3, pp. 395–407, Jan. 1989, doi: 10.1080/02664768900000049.