Deskewing of Underwater Images: Replication Study

Brent Wasilow
School of Computer Science
California State Polytechnic University
Pomona, California 91768 USA
btwasilow@cpp.edu

***Abstract*—We provide an independent replication study of the article "Deskewing of Underwater Images". We implement a modified version of the Lucy-Richardson expectation maximization blind deconvolution algorithm in conjunction with various supporting Matlab functions. We approach experimental evaluation using an image capturing technique more indicative of realistic image capturing scenarios. We note that while the results provided by Seemakurthy and Rajagopalan provide reasonable image reconstruction metric increases, it is characteristically afforded through the use of highly constrained and unrealistic testing conditions. Specifically, the amount of wave construction/destruction, camera shake, wave period, bounding box, object depth, edge contrast, sub-object bounding contrast/boxes, and clarity of water are too highly constrained to provide replicable and extendible results. We also provide an explanation as to how unintended and inevitable changes in many of the previously mentioned variables affected our reconstructed images and the image quality metrics therein.**

I. Introduction

It is important to indicate that this article, and every other article referenced therein on the subject of deskewing underwater images, is largely suboptimal in terms of readability. The better part of four weeks were spent trying to decipher the concept of how underwater images are skewed, how each individual solution works in its entirety, and most importantly how to implement any given solution. The most important aspects, the parameters associated with deskewing an image, were often devoid of explanation and information, leaving a frustrating trial-and-error approach as to how best to replicate results accurately.

Part of this can be attributed to the fact that the authors were in the process of obtaining a patent, and therefore could not provide assistance. This reflected in the level of specific implementation detail throughout the paper. More importantly, having been a part of the peer-review process, as well as publishing my own papers, I hold the upmost importance toward creating a paper that unfolds logically; ultimately guiding the reader through each step in a simplistic manner. The same can be said about the referenced articles throughout.

This led me through a chaotic search across dozens of papers, thousands of internet searches, and yet I was still confused. Finally, after figuring out how all of the pieces of minutia fit together, I noticed that the paper was ill-written with reference to intuitive idea development. Therefore, my description and critique of this article will mimic the same though process that lead me to the epiphany of how this paper ultimately solves the problem of deskewing underwater images. In contrast, I found their solution to be highly novel and creative. Specifically, the authors' method of adding constrains in order to transform their problem domain into one that already ahas a solutions, is highly imitable.

II. Image Formation in Still Water

The first and simplest question to ask is, "How is an image formed when capturing a scene through a completely still surface of water?" According to Snell's Law, as shown in Figure 1, the angle of an incoming ray of light is related to the angle of refraction of the ray of light through a medium. This is described mathematically by Equation 1,

$$\frac{sin(\theta_1)}{sin(\theta_2)} = \frac{n_2}{n_1} \tag{1}$$

where n_1 is the refraction index of the top medium and n_2 is the refraction index of the bottom medium. Based on this equation, one can image a camera's imaging plane parallel to the surface of the water and shooting rays that are perpendicular to the surface of the water.

Therefore, $\theta_1 = 0$. Letting the refractive indices of air and water be $n_1 = 1$ and $n_2 = 1.3$, respectively, we can solve for Equation 1 as follows in Equation 2:

$$sin(\theta_2) = \frac{sin(\theta_1)}{n_2} = \frac{sin(0)}{1.3} = 0 \tag{2}$$

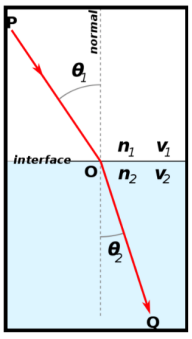

Fig. 1. Snell's Law

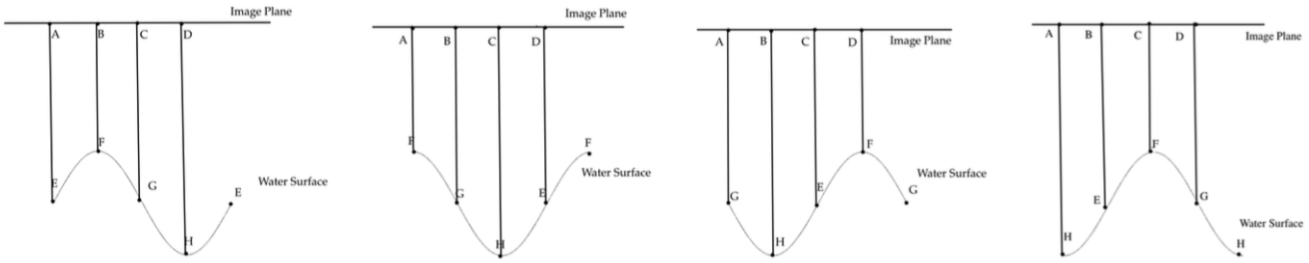

Fig. 2. Motion Blur Due to the Presence of Water

Therefore, $\theta_2 = 0$. This means that in completely still water, all light rays move through in a straight line. Ultimately, a picture captured in this way would not show any indication of being underwater.

III. Image Formation in Flowing Water

The next logical question is, "How does the imaging process change in the presence of flowing water?" The article delved deeply into how to simplify the image formation process using sine waves, however it is rather esoteric and unnecessary for this description. The most important aspect is that a deformed image can be represented by Equation 3:

$$I(x,t) = I_g(x + w(x,t)) \tag{3}$$

where x is the pixel being deformed at time t by a warping factor, w. This warping factor was shown to be directly equal to the gradient of the wave on the surface of the water that x comes into contact with. A clear example is given in Figure 3, whereby one can see that a pixel being indexed directly perpendicular to the water surface, achieves a deformation due to the gradient of the wave. Moreover, the amount of deformation is small if the gradient is small, and the amount of deformation is large if the gradient is large.

IV. Skewing Due to Flowing Water

Based on the previous explanation, one can deduce that the cause of skew is due to flowing water. During image capture through a still water surface, each pixel on the image plane will pass through without any deformation – causally attributed to the lack of a water gradient. However, during image capture through flowing water, each pixel on the image plane will come into contact with numerous different gradients. Some pixels will pass through undeformed, whereas others will be highly deformed. This deformation is the cause of skew. Some pixels correctly index the scene, and some do not. Logically, we can ask the question, "How do we remove skew?" We note that this is Problem 1.

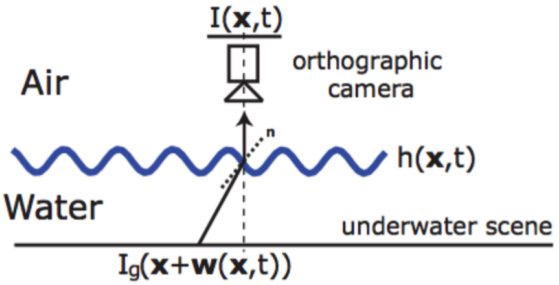

Fig. 3. Imaging An Underwater Scene

V. Motion Blur Due to Flowing Water

Due to the non-immediate nature of image capture, namely camera exposure time, we must take into account an equally-likely image deformation scenario – motion blur. Therefore, one must ask the question, "How does the length of time for any given camera exposure effect the image formation process?" Due to the dynamic nature of the water surface, the wave formation process will change during camera exposure. This is illustrated by Figure 2.

Capturing a scene through a dynamic water surface using a non-infinitesimal camera exposure time will result in skew and a corresponding motion blur. It is important to note that the idea of skew is embedded in the motion blur. This is important because the skewing is caused by the non-flat water surface and the motion blur is caused by the change in the non-flat water surface during exposure time. So, we can ask, "How do we remove not only skew, but also the corresponding motion blur?" We note that this is Problem 2.

VI. Problem Transformation

The problem of removing both skew and motion blur is incredibly difficult. First and foremost, skewing is a complex mathematical process that is highly difficult to reverse given a skewed image. Information about the waveform is required – rate, direction, and type – and is usually difficult to infer. Moreover, motion blur is a highly complex process to reverse. This is made even more difficult by the fact that each point on our image will experience different blurring characteristics. Combining these two problems and trying to find a solution is next to impossible.

The idea is to try and constrain this problem so that we can reduce it into one that is much easier to solve. Specifically, is there a way to remove only the skew component *or* the blur component? The authors make an intuitive leap by imposing a long camera exposure time on the image capturing process.

Fig. 4. Motion Blur and Skew Due to a Small Exposure Time

Fig. 5. Motion Blur Only Due to a Long Exposure Time

If a long exposure time is used, each pixel will experience the exact same set of skews.

This is due to the cyclic nature of waves. With a small exposure time, one pixel may be skewed by the first quarter of the waveform, while another pixel is skewed by the second quarter of the waveform. Because these two pixels encounter different waveform positions, they encounter different skews, thereby allowing the concept of skew to even take place. This is illustrated in Figure 4.

If each pixel encounters exactly the same set of skews as every other pixel in the image, then there is no skew, only motion blur – depicted in Figure 5. Thus the solution is to provide a long camera exposure time to eliminate the skewing portion of the image, leaving only a uniform blur. The blur is uniform due to the fact that every pixel is being skewed exactly the same way, therefore each pixel is also being blurred exactly the same way.

VII. Image Blur

Having transformed the problem into a constrained domain whereby our image is left with only motion blur, we can begin by understanding how to deblur an image. In order to understand how to deblur an image, we need to understand how to blur an image. A blurred image is the combination of a starting image that is presumable devoid of blur, a blurring kernel, and a set of noise characteristics. The blurring kernel is a given type of modification that is performed to each pixel, or set of pixels, in the image. A simplistic blur kernel is the averaging kernel of size 3×3. This filter uses the 8 neighbors surrounding the center pixel, adds them together, and divides by 9. This will produce a uniform averaging blur.

Moreover, the newly convoluted image that was created using the original image and the kernel is then added to a noise function that produces a characteristic set of noise for that image. This can be either salt and pepper noise or various artifacts. When an image is captured from a camera there is generally a noise component to it – a set of distorting artifacts embedded in the lenses capturing the image, or artifacts embedded in the scene such as water distorting light paths.

When capturing an image with a camera, a blur may also be introduced. This blur is generally caused by camera movement, exposure time, or scene movement. This dictates the type of blur caused and has an impact on deblurring. This is due to the fact that some images are blurred differently with respect to certain portions of the image, or some may be blurred uniformly, which is generally what happens when capturing an image using a long exposure time. Now that we have a blurred image the next question one can ask is, "How do we deblur a blurred image?"

VIII. Image Deblurring

Once we have our blurred image we can try and deblur it. This process can be achieved if we know the blur kernel (*i.e.,* the type of blurring that was done) and the noise that was embedded in the image during image capture. The process of deblurring can be inversely performed by dividing the blur kernel by the image in the Fourier domain. This is due to the fact that convolution can be translated into division in the Fourier domain. Therefore, given a blurred image corresponding to Equation 4:

$$g = f * h + n \tag{4}$$

in which g is our blurred image, f is our clear image, h is the blur kernel, n is the noise, and * is the convolution operator, we can perform deblurring using Equation 5:

$$F = \frac{(G - N)}{H} \tag{5}$$

in which the capital letters signify the same variables from Equation 4, but translated into the Fourier domain.

However, this process almost never provides realistic image deblurring due the fact that Equation 5 assumes that the precise set of noise characteristics are known or given. Therefore, numerous techniques have been developed to estimate a deblurred image given a blurred image. The most popular technique is the Richardson–Lucy algorithm which accounts for the noise characteristics by using the Poisson distribution.

In its most simplistic form it can be represented by Equation 6:

$$f(k+1) = f(k) \ .^* \left[\frac{g}{f(k) * h} \right] \tag{6}$$

where $f(k+1)$ is the new image at the $(k+1)$-th iteration, $f(k)$ is the image output of the previous iteration, * is the convolution operation, $.^*$ is the image correction operation, g is the original blurred image, and h is the blur kernel.

This algorithm iteratively tries to deblur an image by computing a correction factor. In Equation 6 the output of the previous iteration is blurred using the kernel and compared to the original blurred image. This correction factor is factored into the blurred image for this iteration, providing a new output image. Basically, each iteration tries to correct the previous iteration until the point at which $\frac{g}{f(k)*h} = 1$, or in other words, the previous image and the current image are equivalent, implying that $f(k)$ would be the clear deblurred image.

However, we must explore the scenario whereby we do not know how the image was blurred. This is usually the case as images are captured by human beings under inherently chaotic environments. Typically images are subjected to various noise characteristics and a variety of blur characteristics. Equally as important, these blur characteristics can be uniform (*e.g.,* consistent movement over a long camera exposure time) or non-uniform (*e.g.,* car accelerating from a stop while the camera man inadvertently applies shake during image capture). Assuming we are not given the blur characteristics, how can we appropriately deblur an image?

IX. Blind Deblurring

Given an unknown set of noise and blur characteristics embedded into any given image, deblurring an image requires solving a two variable equation. In order to solve for two variables that are mutually dependent on each other, we can use expectation maximization. Because the two variables are mutually dependent on one another, they must be modified incrementally and in such a way as to jointly modify them so that they maintain the same mutually dependent link.

An adept example is trying to study for a test for a class that has not been attended. You know that you need x knowledge to answer y problems. You make an initial guess at what chapters you think you need to study, such as chapters 1, 2, and 4. Once you study those chapters you know problems y_1 will most likely be on the test. However, based on those problems you know that you should also study chapter 6. Due to the addition of chapter 6, you have a good guess that problem set y_2 will appear on the test. You continue on this path until you maximization your personal expectation. This is in contrast to reading the entire book as a method of studying for a test.

This same process can be applied to image deblurring and it is the key process provided the authors in the article. Because we do not know the blurring characteristics of the image, we must take an initial guess as to the blur kernel. Moroever, because the authors demonstrated that a long camera exposure produces uniformly blurred images, we can assume that a single blur kernel was used across the entire blurring process. If a shorter exposure time was used, not only would we have skew in the image, we would also have non-uniform blur. In order to deblur an image that is non-uniformly blurred, we would have to solve for a set of blur kernels, not just one. This would no longer be solving a two variable problem, and would instead diverge into a hundred or thousand variable problem.

The article provides a mathematical implementation of the algorithm used to deblur an image. It is almost exactly the same as various other implementations discussed in related blind deblurring articles. Firstly, the noise characteristics are discussed and factored in as Gaussian noise, which is very common amongst images generated using a digital camera. An initial guess is created for our blur kernel, otherwise known as a Point Spread Function (PSF). This represents the blur characteristics of our image and is the most important step, since it determines the neighborhood of pixels that will be modified as well as how they will be modified.

Unfortunately, the authors do not provide any concrete indication of their estimates. However, when emailing one of the authors, he mentioned that he chose numerous different PSF estimates and then later refined the initial estimate depending on the results produced. The blurred image we want to deblur is then transformed into a latent image using their prediction step. This step requires the image to be shock filtered, bilaterally filtered, and subjected to gradient magnitude thresholding.

The bilateral filtering smooths the image while preserving edges. The shock filter then takes those edges and created defined, discrete edges. Finally, the gradient magnitude thresholding removes gradients that are larger than a given threshold. Ultimately, this process produces an image with strong edges and lines that represent the overall depiction of the structure of the image. The initial PSF is then used in conjunction with the latent image received after the three filtering operations to estimate a new PSF. This new PSF is estimated by taking the latent image and trying to minimize the amount of energy in a function they provide. The function factors in numerous partial derivates of the image, the PSF, and various constants that need to be experimentally determined. This function is marked as Equation 23 in the article and is the underlying mechanism behind the Matlab function `deconvblind()`.

The new PSF is then used in conjunction with the latent image to determine the new latent image. The new latent image is the product of minimizing Equation 26 in the article. The process involves using partial derivates, constants, the PSF, and gradients. The new latent image is then compares to the first latent image. If the latent images are very similar, below a specific threshold, or a maximum number of iterations is reached, then the algorithm is stopped. This means that the image does not differ in any observable measure between this iteration and the last, therefore the image can no longer be modified and has reached a local maximum. In other words, the initial PSF estimate is used with the blurred image to create

a new image. The new image is compared to the blurred image. Depending on the comparison, the PSF estimate is modified so that the next iteration will produce a better image.

Moreover, the authors provided a method for reconstructing blurred images from circular ripples. They indicate that blur due to circular ripples is no longer inform but can be transformed as such. They indicate that modifying the images and translating them into the polar domain allows the pixels to organized so as to make them space-invariant. However, one must take into account the center of the originating ripples so as to determine how to organize the new images' pixels.

X. Paper Results

The authors described the results of their implementation on both unidirectional cyclic waves and circular waves using both real images and synthetically generated images. The captured images were generated by using a container filled with water with dimensions $4 \times 2 \times 1$ and placing an image under the surface of the water. The unidirectional cyclic waves were generated using a fan and the circular waves were generated using drops of water. The water movement characteristics were outlined as being between 1–2 cm in height with a spatial frequency of 4 cycles. Moreover, they tested their experiments using an exposure time of 2 seconds. The synthetically generated images were created by subjecting clear images to blur using an equation that represented shallow water wave characteristics.

The algorithm was applied to the images degraded by the movement of the water and the resulting images were gathered. In order to quantify their algorithms' effectiveness, they used three image quality metrics. These three image quality metrics will be discussed in §XI, which outlines my experimental results. Moreover, the authors compared their algorithm to other algorithms that have been implemented in the scientific community to solve this problem. The resulting image comparisons are givenin Figures 7 – 14 at the end of the paper.

We note that all figures performed better using the authors' algorithm. Not only were the images clearer from a visual perspective, but they also had higher metric values in comparison to the blurred image and the other proposed image deblurring methods from the literature. What is most intriguing is that the results of [8] and [18] follow more in line with my personal experimental evaluation when attempting to implement the algorithm in this paper.

XI. Implementation & Results

This section will outline the input images, the output images, the Matlab implementation, and a discussion about each piece and how it all fits together. Following the discussion, I will include various images representing a ground truth image, a blurred image, and the corresponding reconstructed images using various different parameters.

A. Images

The input images were taken using a high resolution Canon camera that was rigged/suspended over a bathtub. The bathtub was filled with water and a Listerine mouthwash bottle was placed, and anchored down using weights, at the bottom. The water was allowed to become sedentary, whereby images were captured with varying exposure times using a blowdryer placed 4 feet away, creating unidirectional cyclic waves. For circular ripples, water droplets were dripped into the tub 6 inches from the camera center.

The subsequently mentioned directories contain all of the images captured using the methods outlined in "Deskewing of Underwater Images". The input images are stored in the following directories:

1) `circular`
2) `ground`
3) `ucw`

The `circular` directory contains images captured in which the surface of the water was being distorted by circular ripples originating a few inches from the camera center.

The image exposure times range from $\frac{1}{100}$, $\frac{1}{20}$, 1, 2, or 3 seconds. However, I only included the 1, 2, and 3 second exposure time images in the experimental results – allowing brevity. The darker images constitute the shorter exposure times and the brighter more blurred images constitute the longer exposure times as the lighting intensity is compounded when averaging exposure frames.

When looking through the `circular` directory, notice the circular skew corresponding to shorter exposure times and how that skew is slowly eliminated as the exposure time lengthens and becomes a circular uniform blur.

The images stored in the `ucw` directory represent all images captured on an underwater scene in which the surface of the water was being distorted by uniform cyclic waves. These waves were created using a blow dryer originating several feet from the camera center. The images range from exposure times of $\frac{1}{100}$, $\frac{1}{20}$, 1, or 2 seconds. As with the circular images, the darker images represent the shorter exposure times and the brighter images represent the longer exposure times. Similarly, notice the very subtle linear skews with the shorter exposure times and a more uniform blur without skew on the longer exposure times.

Finally, the `ground` directory represents our ground truth image which will be the reference image for our image quality metrics. More specifically, that means that the reference image will be used in conjunction with each image in the `circular` and `ucw` directories. These metrics will constitute the control metrics. Then, the algorithm to deblur the images will be run on each image and the resulting images will be compared against the reference image again. The two values for the original image and the deblurred image will be compared to determine if the image quality has increased or not.

As a final aside, these three directores are used as inputs to the Matlab code which is located under the `src` directory.

B. Matlab Code

All of the Matlab code is located under the `src` directory and contains everything necessary to create the experimental results. There are four Matlab files:

1) `nmi.m`
2) `metrics.m`
3) `generateMetrics.m`
4) `determineBestDeconvBlindValues.m`

Each Matlab file is properly documented with a high-level description of what the function does as well as inline comments about what each specific portion of code divulges.

nmi.m: The Matlab file `nmi.m` implements the Normalized Mutual Information (NMI) image quality metric that was described in the paper. This metric is used to determine how mutually dependent two images are. In other words, NMI is a metric that allows us to generate a single value that determines that image I_1 is linked to image I_2 by x amount.

`nmi.m` defines a Matlab function that takes in two images, A and a reference image ref. A histogram is then created that contains all of the mutually dependent values across the two images. Specifically, this means that if two pairs of color are seen then we increment the number of occurrences in our histogram. So, if we see a color value of 255 and 232 for images ref and A, respectively at location i and j then we increment the occurrences. The joint histogram is then normalized by dividing by the sum to get the probability of each occurrence. We then grab all of the non-zero elements of the histogram since the entropy calculation does not work for zero due to the use of logarithms. We then calculate the joint entropy between the two images. This process is repeated with each image separately, retrieving their respective entropy values. The NMI is then calculated using each individual entropy value added together and divided by the joint entropy. This value is then returned by the function to be used in the `metrics.m` Matlab file.

metrics.m: The Matlab file `metrics.m` implements a function that returns the three image quality metrics used in the paper:

1) Peak Signal-to-Noise Ratio (PSNR)
2) Structural Similarity Index Metric (SSIM)
3) Normalized Mutual Information (NMI)

Matlab includes implementations for both PSNR and SSIM, but not NMI, hence the separate implementation of the NMI metric. A reference was merely coded to include these three metrics to return the appropriate values given input images. We provide a brief description of PSNR and SSIM.

The Peak Signal-to-Noise Ratio represents the value of corruption between two images, A and ref. Specifically, this calculation is used to determine the quality of reconstruction between our reference image and the original image vs. the deblurred image. I will expand on this metric later during the result discussion since I believe this metric choice was poorly conceived for our purposes.

The third metric used was the Structural Similarity Index Metric. This metric takes into account the spatial and inter-pixel dependence correlation. That is to say that structures in an image are usually highly-correlated to their colors and the ones next to it – their neighbors. As an example, the pink dumbbell in Figure 4 has a very high inter-pixel dependence with the tan color the bathtub. Therefore, any compressed or blurred image will change that inter-pixel dependence, showing a lack of quality or blurriness. However, this method takes into account the fact that our reference image is a perfectly reconstructed and ground truth image when compared to our ucw and circular images. I will explain why this concept affected my results heavily in the subsequent results portion.

As a final aside, we note that the three metrics PSNR, SSIM, and NMI are returned in an array that will be used by other functions to produce the desired experimental evaluation results.

generateMetrics.m: The Matlab file `generateMetrics.m` is our starting Matlab function (*Main*). It will be typed into the Matlab console and will return several files and images that will give us an idea of the results of our implementation. Specifically, this function performs the algorithm outlined in the paper so as to retrieve a higher quality image – hopefully. Then, each of the metrics are generated using the reconstructed image in conjunction with the ground truth image. We note that the metrics were also generated using the blurred image (*i.e.,* before reconstruction) and the ground truth image to provide a comparison to the newly restored image metrics. These two values are then compared to determine if the newly reconstructed image is of higher quality than the original image, metrically.

The function performs this process iteratively by looping through the `ground` directory to gather each reference image – we use different reference images to correlate different results. The ground truth image is read in and resized to produce time-efficient computational results. The Canon camera produces pictures at resolution of 4000×2500. By shrinking them to a quarter of their original size, we can dramatically increase the algorithms running time. However, we note that cutting out those pixels provides a potential for modified results. This was necessary to provide a quick verification, as spending an hour of processing time for each image was not conducive to achieving that goal.

The `ucw` directory was then looped through. Each image was read in, resized by a quarter, and passed into the metrics function as provided in the Matlab file `metrics.m`. The metrics function takes the reference image (*i.e.,* the ground truth image) and the ucw image, at this iteration, and generates the three appropriate metrics – PSNR, SSIM, and NMI. These are our control metrics. We call the `determineBestDeconvBlindValues` function using these three metrics, the reference image, and the input image. This function will be explained next. The same process is then performed for each file in the `circular` directory.

determineBestDeconvBlindValues.m: The Matlab file `determineBestDeconvBlindValues.m` provides a function that takes in the input image (*blurred image*), a reference image (*ground truth image*), the three metrics gathered from using the ground truth image in conjunction with the blurred image, and the name of the input image file so we can write out an appropriately named modified file. This function's purpose was to run the algorithm provided in the paper (*modified appropriately where we saw fit*) to retrieve a reconstructed image with better quality.

`determineBestDeconvBlindValues` function defines a list of PSFs. The PSFs will be looped through to determine the best starting PSF estimate to use with the algorithm. We note that the algorithm is highly susceptible to the starting size of the PSF and therefore we loop through a set of basal PSFs to determine, procedurally, the one that works best for any given image. Each PSF from the list is used in conjunction with the input image A as arguments to the Matlab function `edgetaper`. This function blurs the edges of the input image so as to reduce ringing and additional artifacts around the edge of the image caused by the steep drop in pixel values.

The tapered image is then passed to the `deconvblind` function provided in the Matlab toolbox. This function takes in numerous parameters including a blurred image, an initial PSF estimate, the number of iterations of the algorithm to perform, a damper value for cutting out noise, and a weighted array to determine which pixels to include in the calculation. For the implementation, we used the input image, the initial PSF, a procedurally determined number of iterations, and a varying number of damper values.

The entirety of the article rests on the input parameters to this functions. Due to the sparse amount of information offered by the authors, trying to reproduce their exact results was extremely difficult. Therefore, we applied procedural determination of variables using variations applied through looping. This is why we chose a set of basal PSFs to iterate through, a range of iterations to run, a range of damper values, and a range of weight images which represented all of the sharp contrast edges in the image.

The initial PSF was more so affected by the size and not the starting values since the purpose of the algorithm is to deconstruct the PSF. Therefore, we chose matrices consisting of all 1's in various sizes. The iterations parameter determines the number of times the PSF and image will be updated in the underlying expectation maximization (EM) algorithm. The resulting image is highly influenced by this number. Higher numbers of iterations compounded ringing artifacts and noise not otherwise accurately accounted for. Therefore, an optimal number of iterations needed to be selected. Moreover, the damper value determined the variance from the output image and the input image for that specific iteration. Image changes that were large were discarded so as to not include the compounding problems associated with the underlying algorithm (*noise and ringing*).

In summation, reproduction of the results by determining the correct algorithmic inputs, was a chaotic trial-and-error mess. Even with all of the various iterations, no output image every struck us as being successfully reconstructed, especially when viewing the authors results which were considerably clearer/sharper.

Once the image was reconstructed using the input arguments to the function, I displayed the image and wrote it to a file so it could be viewed comparatively to the original blurred image. Then, the newly reconstructed image was passed as an input to the metrics method in order to determine the PSNR, SSIM, and NMI. This represents the reconstructed images' quality in comparison to the reference image and will be used against the previously mentioned control metrics. Finally, all of the image metrics were written to a file so as to keep track of which parameters performed in what way.

C. Output

The output of running the algorithm from the Matlab command line using the following command – `generateMetrics` – is included in the `output` directory. This directory contains the following three subdirectories:

1) `ucw`
2) `circular`
3) `metrics`

The `ucw` and `circular` directory contain the output images written during the process of reconstructing the image. Both of these directories will have subdirectories which will hold a set of reconstructed images for the top image using various parameters to the deconvolution algorithm. The image files will be labeled with the name of their initial blurred image filename, the size of the PSF, the number of iterations, and the damper value. An example is as follows: `IMG_6145-3-50-0.JPG`. The `metrics` directory will hold the metrics gathered for each of the images and their appropriate reference image.

XII. Results

Our results were very lackluster. None of the results matched those of the article. In the article, the authors show tables of values for each of their tests representing how well their algorithm performed. The most important aspect of their table was the original blurred image score for the three metrics, as well as the final scores for their algorithms' output. Across all of their tests, both synthetic and real, the authors were able to demonstrate large increases in the value of the metrics. This meant that their algorithm produced clearer images, or ones that were closer to the original reference image.

For our experiment, non of the metrics produced scores higher than the original blurred image. For example, the three metrics calculated using `IMG_6145` with the ground truth image `IMG_6144`, were 28.020606, 0.933892, and 1.274324. However, all of the reconstructed images of `IMG_6145` performed as follows, using the same ground truth image:

IMG_6145.JPG	28.020600	0.933892	1.274324
IMG_6145-3-50-0.JPG	23.536508	0.796028	1.165512
IMG_6145-5-50-0.JPG	15.428022	0.345165	1.058983
IMG_6145-7-50-0.JPG	14.112679	0.290317	1.048837
IMG_6145-9-50-0.JPG	14.047674	0.282816	1.049137

As is apparent, none of the images metrics perform better than the original blurred metrics. We provide some clues as to why this may be the case. For one, the ground truth image is not an exact unblurred representation of IMG_6145. There are subtle differences in spatial locations due to camera movement, distortion, and noise. Therefore, the metric gained from using IMG_6145 in conjunction with the ground truth image, is incapable of being correlated against the reconstructed images. Moreover, while a reconstructed image may be clearer to the human eye, it may mathematically be much different than the ground truth image, indicatively represented by a lower metric value. Moreover, all of our experimental images were gathered in full RGB color. The authors strategically chose grayscale images, thereby potentially compounding our results – errors compounded over a triplet of color values in deference to a single grayscale value.

When looking at many of the images, we notice that they appear to be much clearer than the initial blurred images. Therefore, we place heavy skepticism on the use of the previously mentioned three image quality metrics in our testing. Much of this was further supported by our research into what these three metrics mean, and more importantly how they are computed. For most purposes, these metrics give poor indicators of *perceptual* enhancement in an image.

A. Methodology

Our experimental methodology consisted of 25 iterations of the algorithm, a damper value of 0, and PSF sizes of 3, 5, and 7 – all values of 1. The higher PSF sizes produces larger amounts of ringing characteristics associated with overestimating the size of the PSF. The most optimal results for the images was produces at PSF sizes of 3 and 5. Very little ringing was produces and some small amounts of blur were reconstructed. We tried various other trials ranging from small numbers of iterations to large numbers, and small numbers of damping values to large numbers. The larger damping values output images with lesser ringing characteristics. However, the images were also smoother, as though an averaging filter was performed across the image. The larger iterations also compounded noise and ringing characteristics, especially so for larger PSF sizes.

Our goal was to find a set of procedurally discernible properties for determining the best image reconstruction parameters. That is to say, one would upload an image and the algorithm would produce the best reconstructed image possible. However, we noticed that each image is prone to differing noise, motion, saturation, skew, and blur characteristics. Moreover, our images has a combination of large saturation borders as well as very small letters/word spacing mixed in with large letters/word spacing. The algorithm was able to reconstruct blurring in large objects throughout the image using larger PSF sizes such as 7 and 9. However, the smaller letters and contrast borders would become muddled and start to produce ring characteristics. Similarly, the smaller PSF sizes produced clearer results for the small sizes letters/words and contrast borders, but failed to deblur larger portions of the image.

We note that not only is image homogeneity important for deblurring, but also constraining experimental values. Our results were achieved using the experimental methods outlined in the article. However, we noticed that the unidirectional cyclic waves were in fact not unidirectional. If the image was uniformly blurred, then the reconstruction process would have produced more optimal results. This is indicated in the associated presentation where we synthetically blur an image then reconstruct that blurred image. We note that we receive almost perfect results. However, when deblurring with non-uniform blur, unknown noise characteristics, and using an unknown blur kernel, it seems almost impossible to reconstruct the image, let alone produce anything noticeably better.

While implementing the algorithm, we had to determine the numerous different parameters which were essential to the algorithms performance. The authors unfortunately did not provide any information about the exact parameters used and therefore we were unable to reproduce their results as adequately. However, we did investigate numerous other methods, such as blurring the image before deconvolution, adding the edge/weight array, and trying different filters. While we did not place the different filters in the runnable code (commented out), we ran through numerous estimate kernels using the Matlab function fspecial.

Our initial though was that the blur afforded by the image capture process would be uniform linear motion blur. Since the waves are moving cyclically in a linear way, the motion translated across the exposure time should have been uniformly linear in the direction of the waves. However, we were unable to produce satisfactory results using the motion filter and instead coded the basic PSF filter of all ones. The motion filters take in a length as well as a magnitude for translation. Our thinking was to visually estimate the pixel translation length from the blur as well as the direction of the blur from the waves. Most of the images were determined, using individual by-hand measurements, to be blurred anywhere from 5-10 pixels in an angle of roughly 10°. We appropriately used these values as inputs in generating the kernel to be used to perform the subsequent deconvolution.

Most of our frustration with producing clearer images revolved around trying to deblur the images adequately. The article is merely about translating a skewed image into a uniformly blurred image using a longer exposure time. Therefore it would have been preferable for authors to include information about their specific deblurring parameters since blind deconvolution is a very general problem requiring hand-tuned settings.

Based on our implementation, results, and subsequent analysis, we believe there to be a solution to this problem. However, we do not believe that this solution is realistic.

Our experimentation had a rigged camera to avoid camera shake and a highly controlled set of wave generation methods. In a realistic scenario, a camera would not be rigged to remove camera shake, objects under the surface would be moving, impurities would produce noise in both the water and camera, and surface movement would not be perfectly linear or circular.

The results of the authors was dependent on highly-controlled experiments, whereas our results were probably on the more realistic side in which multiple wave sources are compounding off of the corners of the bathtub creating non-linear translations, varying-wave heights, and camera movement that was not uniform. All of this combined together caused a space-variant blur instead of a spatially-invariant blur. This reason is why most of the images could not be accurately reproduced. Perhaps with a stricter setup and more constrained experiment we could have accurately reproduced the results of the authors, but this constraining would not be conducive to realistic experimentation.

We chose not to implement the synthetic blur caused by wave motion due to how synthetic it really was. Our testing produced near-perfect results when creating synthetic blur and deblurring with our custom implementation. This procedure was performed by the authors using the equation for shallow water waves. Using said blur equation, they synthetically blurred their images. These images were perfectly uniform and therefore reconstruction was almost perfect because of the lack of variability in image capture (given there was not actually a real image capture process). These results were useless since it is in no way characteristic of a real image capture, much less one that was representative of a scene imaged through flowing water.

We rant the tests on the `circular` directory but found even worse results due to the fact that we could not properly modify the images into the polar domain. Moreover, wave construction and deconstruction further compounded under the circular ripple testing causing metrically and perceptually worse results. We therefore chose to leave out these experimental results and only have output images and metrics for the ucw images. As a final aside, we provide various side-by-side image comparisons in Figure 6 for different reconstructions below, as well as a corresponding table, Table I, labeling all of the metrics for the associated images:

XIII. Conclusion

The conclusion goes here.

References

[1] K. Seemakurthy and A. N. Rajagopalan, "Deskewing of Underwater Images," in *IEEE Transactions on Image Processing*, vol. 24, no. 3, March 2015, pp. 1046–1059.

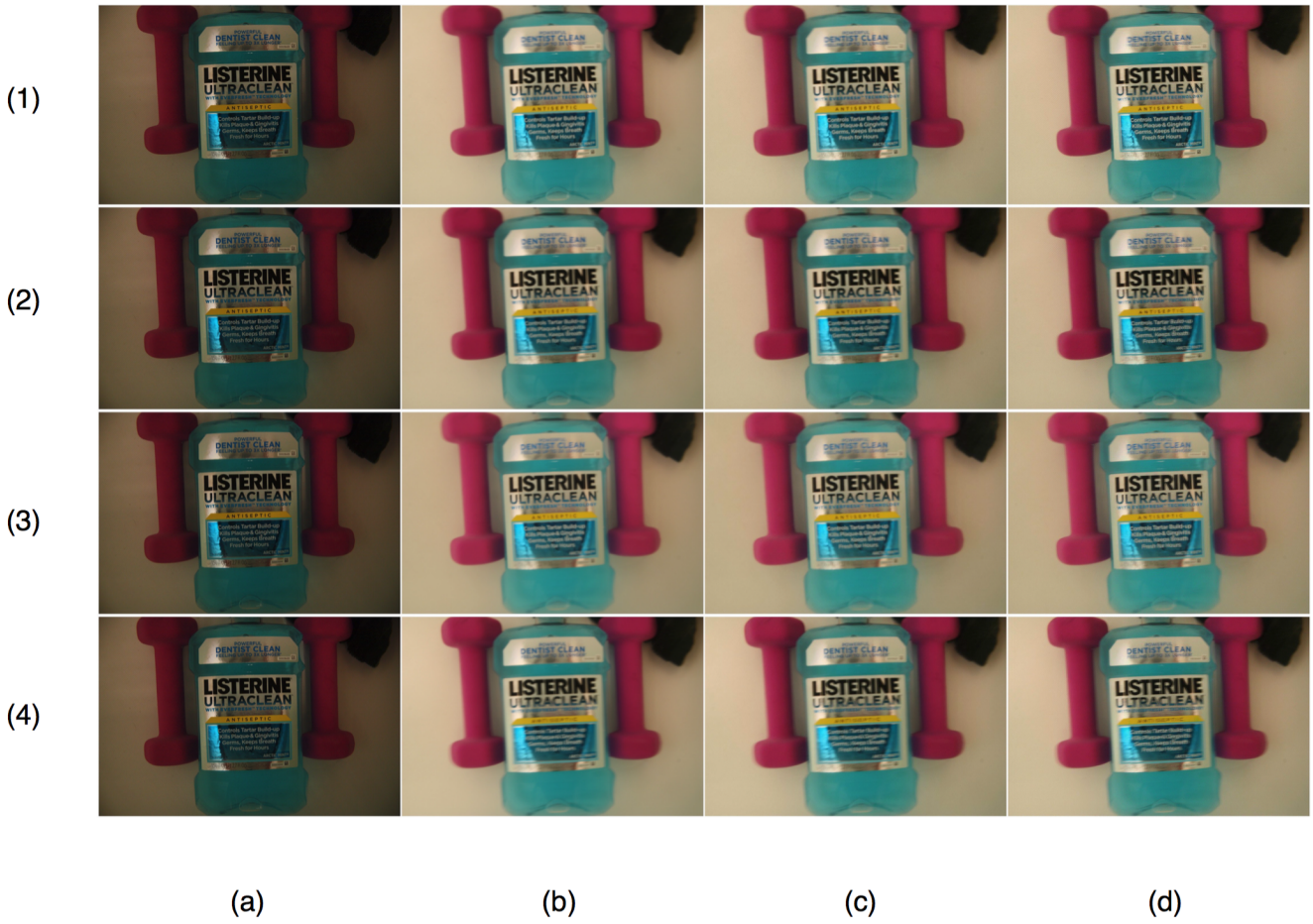

Fig. 6. Our results (a) Ground Truth Image 6144 (b) Blurred Image Collected From Experiment (c) Reconstructed w/ PSF of size 3 (d) Reconstructed w/ PSF of size 5. Image Name by Row Number: (1) 6159 (2) 6160 (3) 6161 (4) 6162.

TABLE I
PSNR, SSIM, AND NMI FOR FIGURE 6.

	PSNR			SSIM			NMI		
Image	a + b	a + c	a + d	a + b	a + c	a + d	a + b	a + c	a + d
6159	10.806 862	10.719 338	10.571 414	0.610 099	0.542 131	0.485 497	1.133 250	1.111 775	1.103 551
6160	11.586 727	11.478 433	11.290 926	0.633 861	0.556 288	0.488 592	1.129 367	1.107 987	1.099 585
6161	11.294 075	11.196 501	11.007 415	0.608 377	0.540 066	0.477 911	1.140 761	1.115 673	1.104 857
6162	11.099 537	11.011 150	10.839 135	0.624 076	0.552 610	0.488 479	1.126 947	1.107 736	1.099 456

Image 6144 or (a) is used as a reference for each and every image since it is the ground truth image. If the images are difficult to view, they can be viewed independently in their original unmodified size in the input and output images directory.

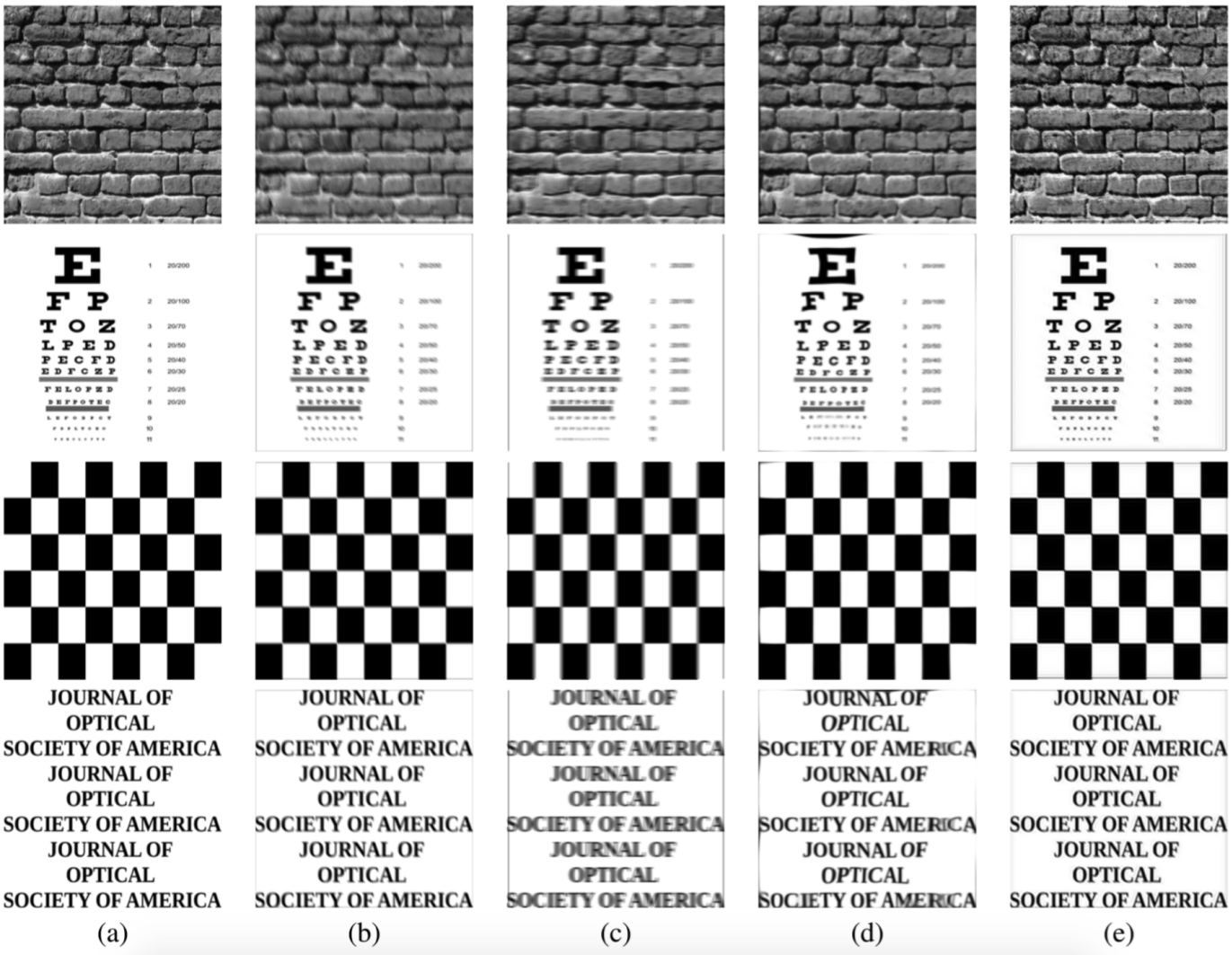

Fig. 7. Synthetically Generated UCW Blur and Associated Results. (a) ground truth image (b) synthetically blurred images (c) results from [8] (d) results from [18] (e) authors' results.

	PSNR (in dB)				SSIM				NMI			
	Blur	Tian [8]	Oreifej [18]	Algorithm 1	Blur	Tian [8]	Oreifej [18]	Algorithm 1	Blur	Tian [8]	Oreifej [18]	Algorithm 1
Brickwall	19.6513	22.4240	21.0315	**23.7990**	0.5572	0.7259	0.7197	**0.8492**	1.1208	1.1559	1.1452	**1.1672**
Eyechart	22.9896	21.4142	19.4327	**23.5653**	0.9190	0.8827	0.8563	**0.9772**	1.3892	1.4257	1.3575	**1.4499**
Checkerboard	23.7942	19.2969	22.0372	**24.5278**	0.9114	0.8215	0.9179	**0.9280**	1.4778	1.6368	1.5327	**1.7404**
Text	20.5040	15.0205	13.9449	**22.9061**	0.9232	0.7934	0.7796	**0.9400**	1.3384	1.2836	1.2359	**1.3387**

Fig. 8. Resulting Image Quality Metrics for Synthetically Generated Blur from UCW.

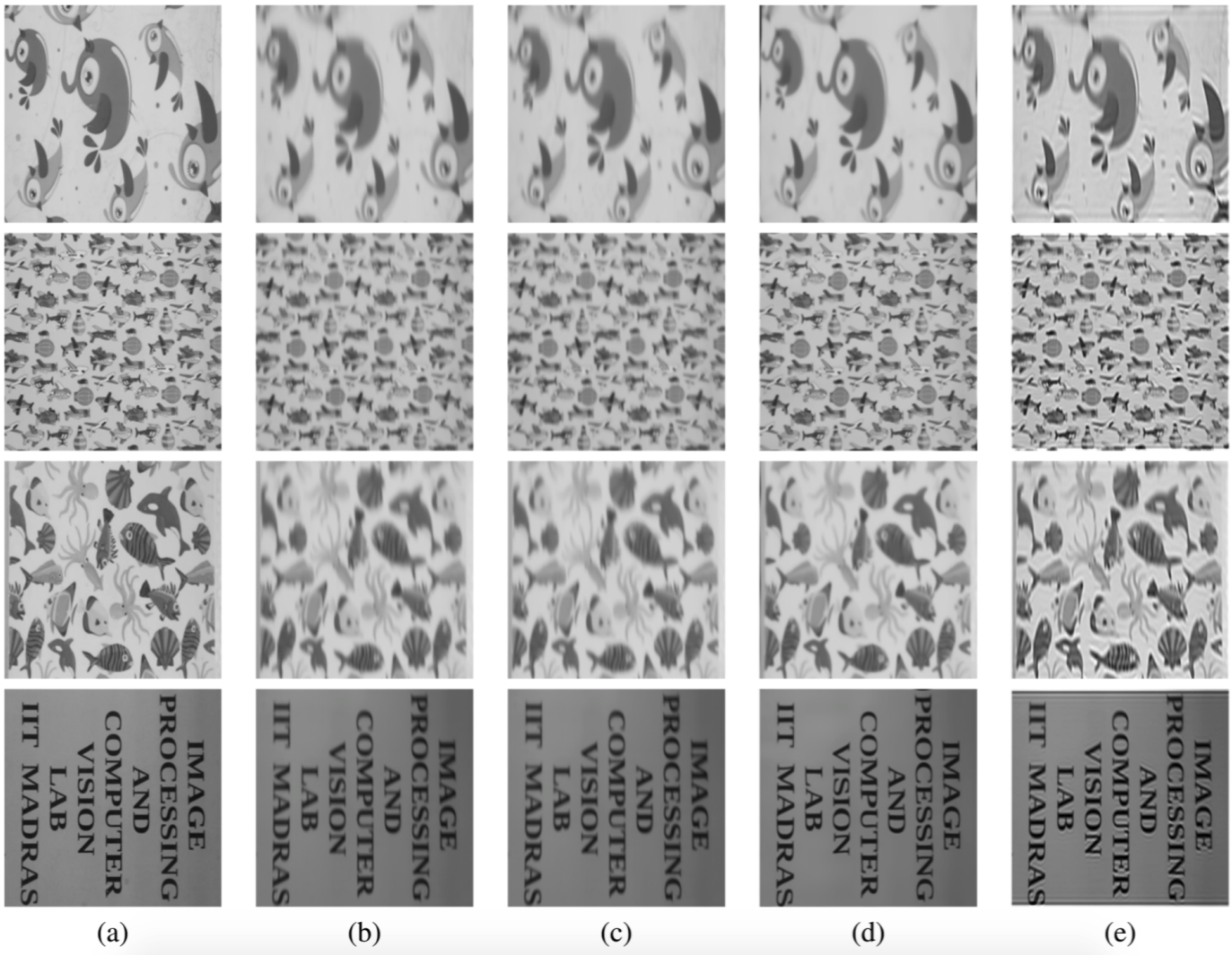

Fig. 9. Real Indoor UCW Experiments and Associated Results. (a) ground truth image (b) blurred images with long exposures (c) results from [8] (d) results from [18] (e) authors' results.

	PSNR (in dB)				SSIM				NMI			
	Blur	Tian [8]	Oreifej [18]	Algorithm 1	Blur	Tian [8]	Oreifej [18]	Algorithm 1	Blur	Tian [8]	Oreifej [18]	Algorithm 1
Birds	27.2184	27.6205	27.9718	**28.7190**	0.8652	0.8734	0.8849	**0.8913**	1.2028	1.2028	1.2090	**1.2829**
Planes	22.0421	23.3494	22.0531	**25.4122**	0.6881	0.7590	0.7892	**0.8506**	1.2354	1.2463	1.2437	**1.2617**
Fishes	24.2461	23.8653	23.7464	**25.6521**	0.7819	0.7874	0.7650	**0.8364**	1.1538	1.1573	1.1450	**1.1650**
IPCV	23.5051	23.3494	23.3995	**24.7142**	0.8198	0.8766	0.8868	**0.9140**	1.1707	1.1488	1.1707	**1.1802**

Fig. 10. Resulting Image Quality Metrics for Real Indoor Blur from UCW.

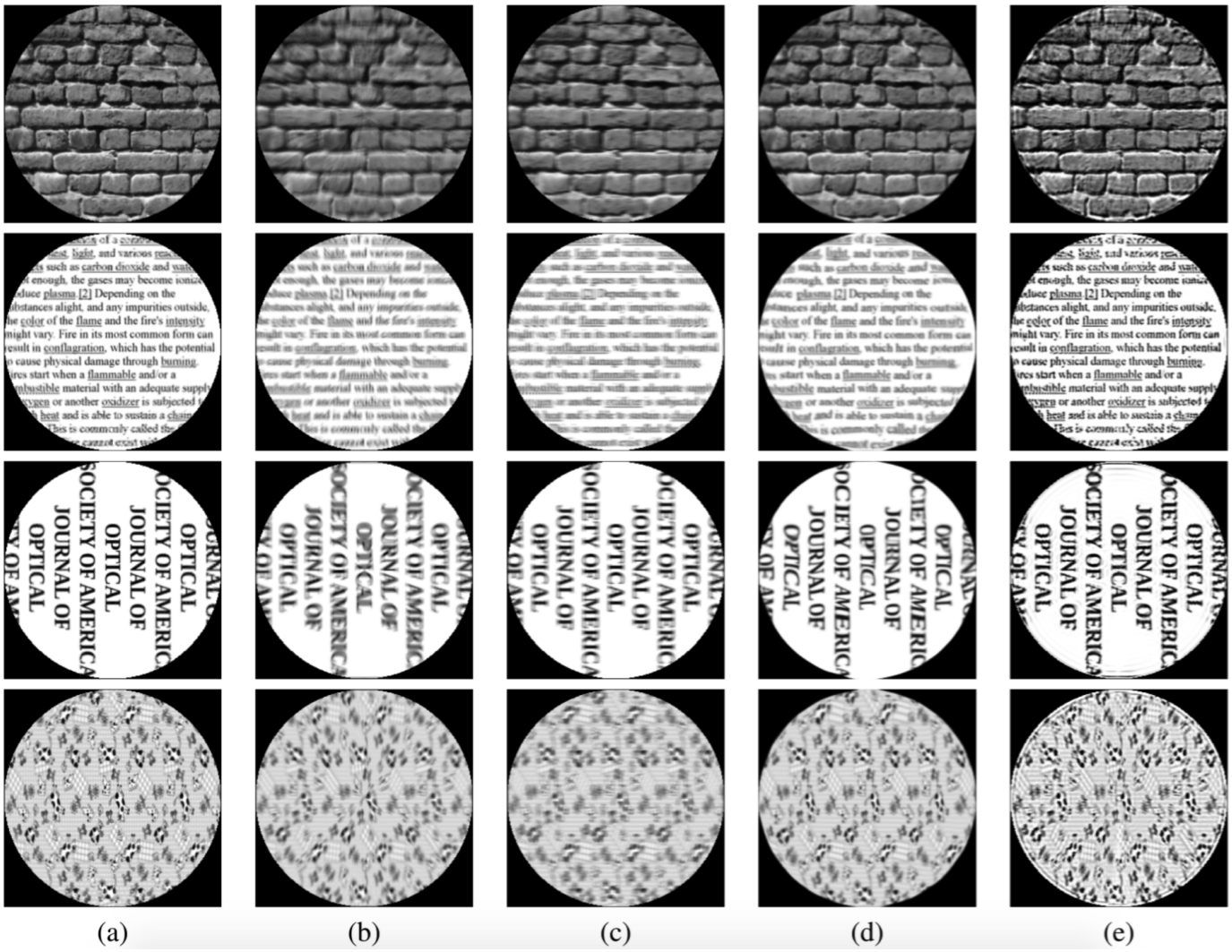

Fig. 11. Synthetically Generated Circular Ripple Blur and Associated Results. (a) ground truth image (b) blurred images with long exposures (c) results from [8] (d) results from [18] (e) authors' results.

	PSNR (in dB)				SSIM				NMI			
	Blur	Tian [8]	Oreifej [18]	Algorithm 2	Blur	Tian [8]	Oreifej [18]	Algorithm 2	Blur	Tian [8]	Oreifej [18]	Algorithm 2
Brickwall	21.9779	22.3291	22.7864	**23.2619**	0.7801	0.8708	0.8437	**0.8791**	1.2157	1.2554	1.2296	**1.2673**
Text	18.4052	17.0964	17.4712	**18.6798**	0.7582	0.7000	0.7213	**0.9066**	1.2426	1.2451	1.2177	**1.3036**
JOSA	16.4220	18.9146	11.3665	**20.7420**	0.8124	0.8634	0.6825	**0.9186**	1.3226	1.3286	1.2997	**1.3299**
Texture	20.8349	19.3000	18.9889	**21.2364**	0.8031	0.7034	0.7297	**0.8637**	1.2074	1.1832	1.1848	**1.3034**

Fig. 12. Resulting Image Quality Metrics for Synthetically Generated Blur from Circular Ripples.

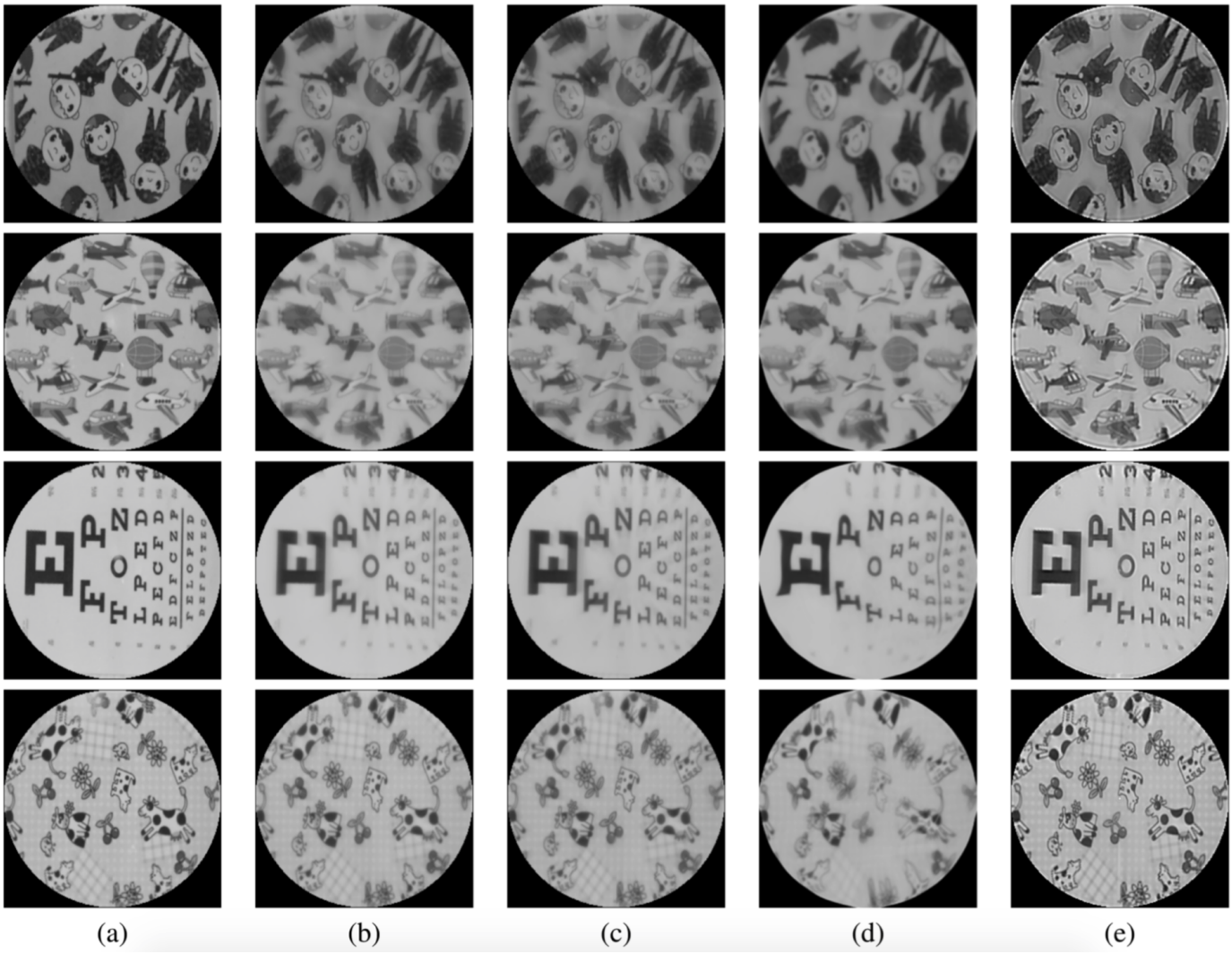

Fig. 13. Real Indoor Circular Ripple Blur and Associated Results. (a) ground truth image (b) blurred images with long exposures (c) results from [8] (d) results from [18] (e) authors' results.

	PSNR (in dB)				SSIM				NMI			
	Blur	Tian [8]	Oreifej [18]	Algorithm 2	Blur	Tian [8]	Oreifej [18]	Algorithm 2	Blur	Tian [8]	Oreifej [18]	Algorithm 2
Soldiers	23.2161	22.3897	22.0637	**24.7994**	0.9225	0.9060	0.8626	**0.9405**	1.2784	1.2695	1.2760	**1.2877**
Planes	26.4353	25.1718	24.6342	**27.2596**	0.9366	0.9390	0.9034	**0.9401**	1.3073	1.2947	1.2809	**1.3804**
Eyechart	23.1920	21.9767	20.1264	**24.8690**	0.9454	0.9325	0.8797	**0.9520**	1.3072	1.3081	1.3090	**1.3133**
Texture	23.8547	22.8205	21.6755	**24.3010**	0.9290	0.8928	0.8481	**0.9319**	1.2029	1.2780	1.2544	**1.2847**

Fig. 14. Resulting Image Quality Metrics for Real Indoor Blur from Circular Ripples.