ASSIGNMENT 2

CMSC 178
Digital Image Processing

Submitted to:
Mr. Dhong-Fhel K. Gom-os

Submitted by:
Lyka Jeanine L. Panchacala
Wineff Chen Zhen Y. Gulane

April 2023

Overview
The assignment was aimed at providing a practical learning opportunity in digital image manipulation using MATLAB, while also reinforcing our knowledge of the foundational concepts in image processing covered in the course. As part of the task, we were required to modify various functions to carry out basic yet essential techniques such as color correction, contrast enhancement, and noise filtering. The main objective was to acquire hands-on experience with these methods.

Exercise 2A – (5%) – De-convolution using a Wiener Filter

1. Background of the Problem

Color balancing is a critical aspect of image processing, but achieving accurate color balance can be challenging, especially when working with images captured under non-standard lighting conditions or different cameras or devices. One approach to color balancing is to use a reference chart, and in this exercise, three simple functions are required:

- `get_chart_values.m` – extracts samples from the 24-color chart image; and
- `chart_correct.m` – calculates the mapping between the ideal and less than ideal cases using the extracted values from both charts.
- `apply_rgb_map.m` – applies this mapping to an image to correct for color imbalance.

These functions can be useful in various applications that require accurate color representation, such as photography, graphic design, and other fields where visual content is critical. AAA AAA
AAA
AAA AAA AAA
AAA AAA AAA AAA AAA AAA AAA AAA AAA AAA AAA AAA AAA AAA AAA

2. Procedure

We used the `wiener_deblur.m` function for image deblurring using the Wiener deconvolution algorithm. This function is already supplied however it is incomplete and cannot be used. We need to modify the `wiener_deblur.m` function to implement the algorithmic process of Wiener Deconvolution. Inside the `wiener_deblur.m` function, these following functions are used: `floor()`, `padarray()`, `zeros()`, `abs()`, `ifft2()`, `real()`, and `ifftshift()`. The remaining functions, `edgetaper()` and `fft2()`, are already supplied. The `floor` function is used to get the floor values of the padding size. The `padarray` function is used to pad zeros to the blur kernel. The `zeros` function is used to instantiate an array of zeros. The `absolute` function is used to ensure that only magnitude is used for the inverse filter. The `inverse fast fourier transformation 2` function is used to convert an image from spatial domain to the frequency domain. The `real` function is used to extract the real values used for the final deblurred image. And the `inverse fast fourier transform shift` or `inverse zero-frequency shift` is used to shift the zero-frequency component of the DFT to the center of the array effectively shifting the image back to its original position putting the image back in the top-left corner.

The algorithm Wiener deconvolution is used to remove the blurriness from an image caused by a known blur kernel. The code modifications we have added focuses on the main steps through using zero padding, inverse filtering, image reconstruction, and spatial delay handling. The first step is to add zero padding. To begin, the initial step involves applying zero padding to the blur kernel. This adjustment ensures that the kernel matches the size of the input image. This serves the purpose of convenient element-wise multiplication in the frequency domain. Although, Wiener deconvolution does not require zero padding, this step simply adds ease or convenience when dealing with the next steps. We then compute for the Fast Fourier Transform representation of the blur kernel. Next step is to compute wiener inverse filter through this formula:

$$F_{\text{inv}} = \frac{F_I}{F_B} \frac{|F_b|^2}{|F_B|^2 + k}$$

the F_{inv} can then be calcuated through element-wise multiplication and division in `MATLAB`. Other way of solving this is to iterate each pixels and calculate their values. But using element-wise operation is much more efficient in vector operations compared to iterating in loops. The next step is to convert the image back to the frequency domain from the spatial domain and then acquire its real values that will be used to display the image. The reason why real values are extracted is because complex numbers in the Wiener inverse filter may include negative and imaginary values, but in the frequency domain, negative values are not utilized as the focus is typically on magnitude and phase information. Displaying the complex numbers would probably alter the image however testing the image example in this exercise did not alter much of the

image and still looked similar with real values being extracted. Once the wiener inverse filter is calculated and extracted for display, the last step is to fix the spatial delays or zero values due to the zero padding.

3. Results and Discussion

In this section, we present the outcomes of our modified `wiener blur` function and analyze how the outputs are influenced by the selected parameters or configurations. The sample image inputs we used are already provided from this laboratory exercise and their corresponding outputs are showcased to demonstrate the function's behavior. There are a total of three (3) tests for `deconv_test.m` function with varying parameters: `K=0.01`, `K=0.9`, and `K=0.000001`. The following figures below are the original image inputs:

Figure 1: Sample Input Image 1

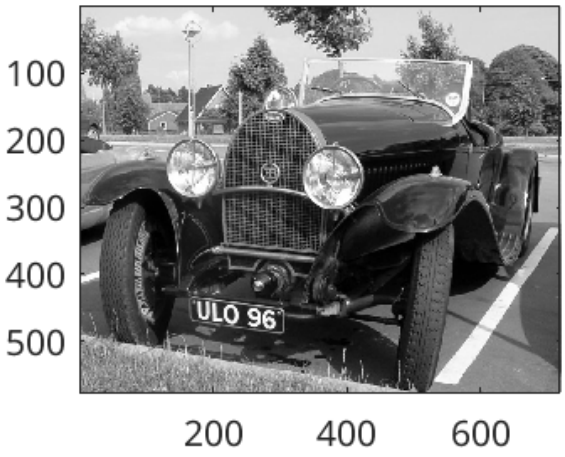


Figure 2: Sample Input Image 2

Since only the parameter K value has been changed, these figures below will be used as the default corrupted data of the two input images:

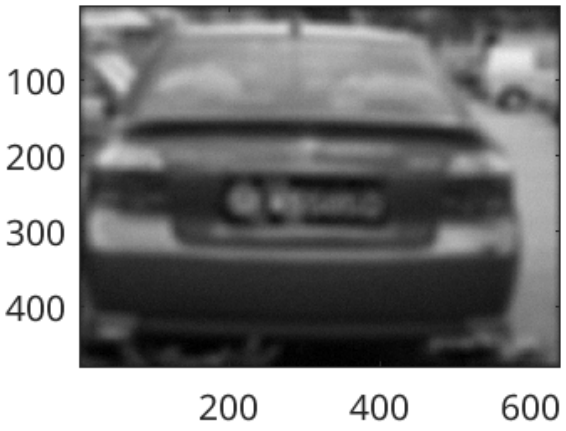


Figure 3: Corrupted Data of Input Image 1

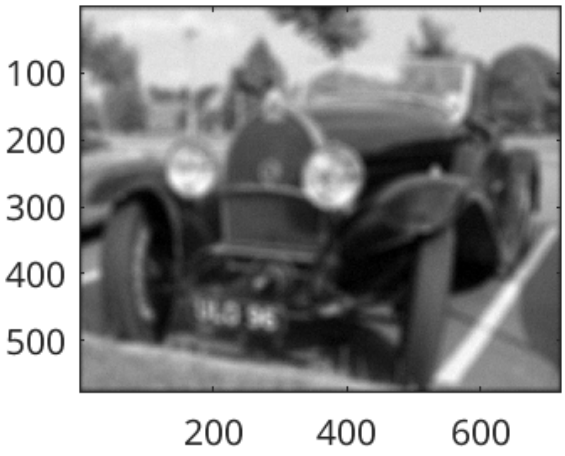


Figure 4: Corrupted Data of Input Image 2

The following below are the tests cases:

- **Parameter K = 0.01** – This test only modifies the K parameter value to 0.01 which is the default parameter K from the exercise.

  Figure 5 and Figure 6 are the guessed PSF values used for deblurring the image.

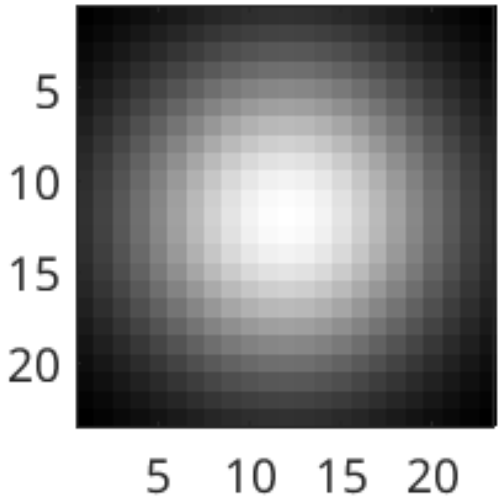


Figure 5: PSF of Sample Input Image 1

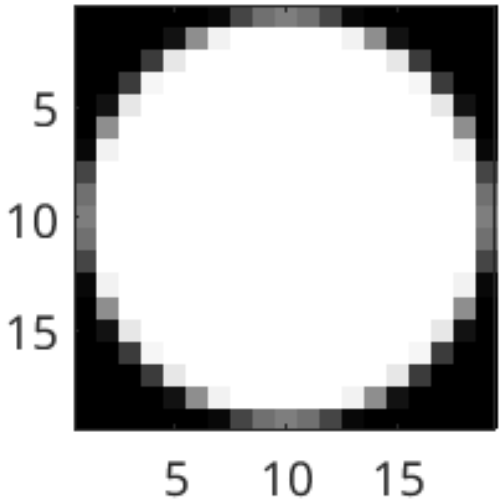


Figure 6: PSF of Sample Input Image 2

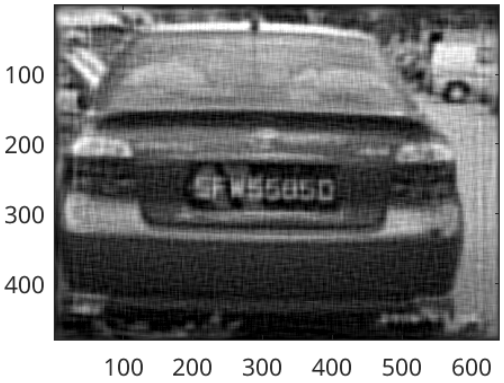


Figure 7: Deconvuled of Sample Input Image 1

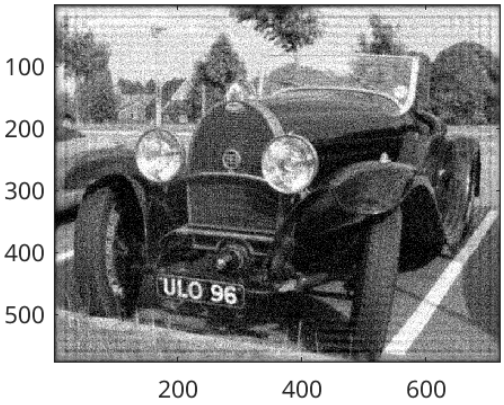


Figure 8: Deconvuled of Sample Input Image 2

Based on these two images Figure 7 and Figure 8, the license plate numbers can be clearly seen but as you can see there are a bit of blocks pattern on the image. But for Figure 7, the license plate number is a bit harder to read. Thus, we have tested Figure 1 with K value of `0.001`.

The results shows that the deconvoled output figures of the sample input images will have different parameter K values and their PSF can differ depending on the images. We have tested both K values for the same images but in order to depict the image with clarity, we further tested 1 with another K value. The result was far clearer in comparison with the default K value. Basing on this test case result, we decided to further test on checking the result of a higher parameter K.

- **Parameter K = 0.9** – In this test, we tried to acquire the result of the deconvoled image if K is `0.9`.

  To much of our surprise, Figure 10 and Figure 11 has changed. We did not expect that changing the parameter K value would alter the result of the corrupted image. As it turns out, modifying `K` was actually the noise standard deviation which may also be the blur kernel.

  Figure 12 and Figure 13 exhibits noticeable degradation in terms of clarity. Due to the presence of extensive image noise, only a limited portion of the image quality was successfully restored using our wiener deblur function. The substantial loss of data caused by the noise renders a complete recovery of the image impossible.

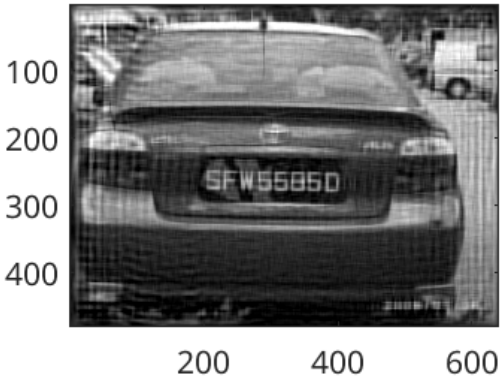


Figure 9: Deconvolution of Sample Input Image 1 at K=0.001

The license plate number in Figure 9 is now much clearer compared to the image with `K=0.01`.

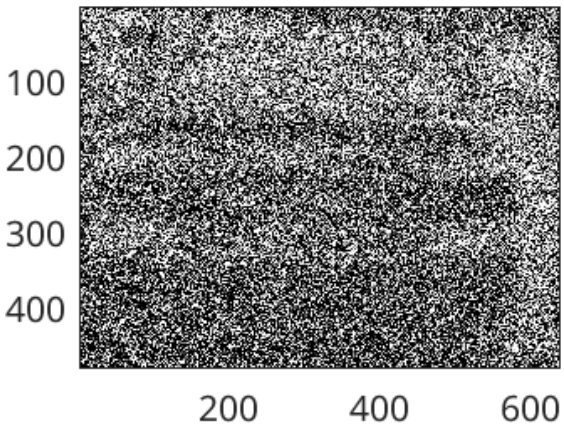


Figure 10: Corrupted Data of Sample Input Image 1

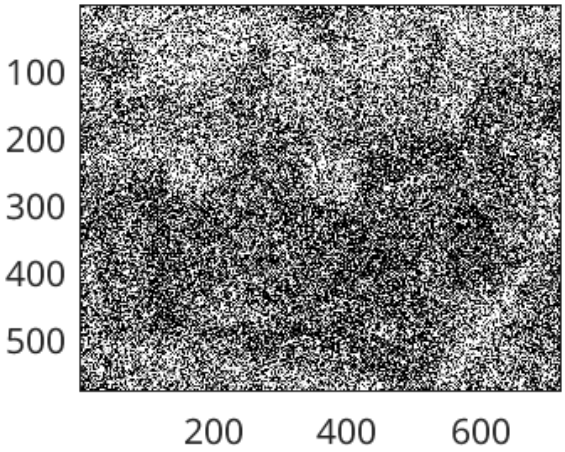


Figure 11: Corrupted Data of Sample Input Image 2

- **Parameter K = 0.000001** – In this test, we tried to acquire the result of the deconvoled image if K is `0.000001`.

  Figure 14 and Figure 15 demonstrate minimal noticeable noise, closely resembling Figures Figure 3 and Figure 4, which represent the corresponding corrupted data. However, upon closer examination of the output images, line-like artifacts are apparent along the X and Y axes. These artifacts indicate the challenges faced by our wiener deblur function in effectively restoring the image to its original clarity.

- **Parameter K = 0.01 (without deblurring)** – In this test, similar parameter K value is used to the first test however we do not blur the image.

  Figure 16 and Figure 17 skips the blurring process and directly undergo to the deblurring process. The output images do not resemble the input images but instead they exhibit distortions resembling the visual effects associated with LSD usage.

- **Parameter K = 0.01 (laplacian and sobel blur filter)** – In this test, similar parameter K value is used to the first test however we have changed the blur filter being used for the input images.

  We applied different blur filters to the images: Laplacian blur filter in Figure 18 and Sobel filter in Figure 19. The purpose of using these filters was to observe their effect on the images and compare it with the Gaussian filter. As shown in the figures, it is evident that a significant amount of details has been lost due to the blurring process. The blurring caused a considerable degradation in the image quality, making the recovery of the original image nearly impossible unlike Gaussian filter.

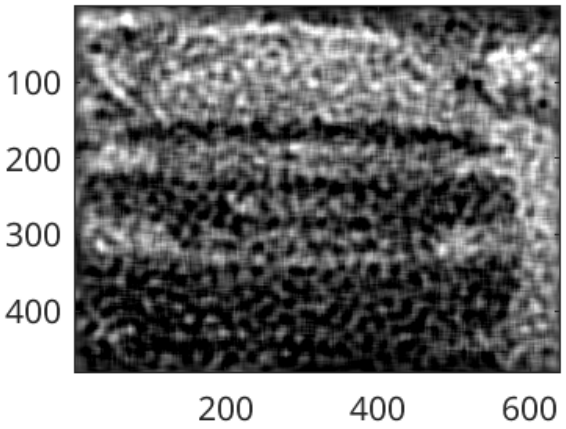


Figure 12: Deconvuled Sample Input Image 1

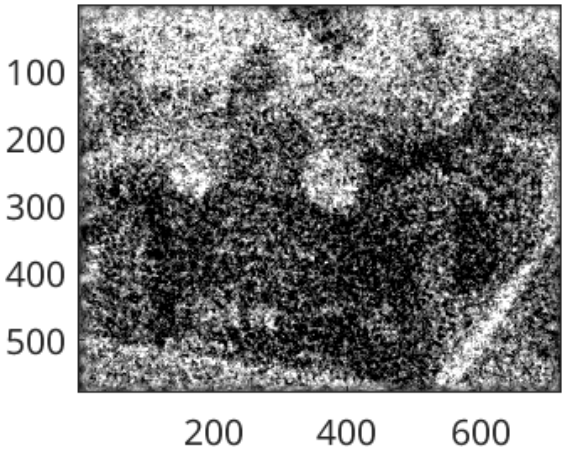


Figure 13: Deconvuled Sample Input Image 2

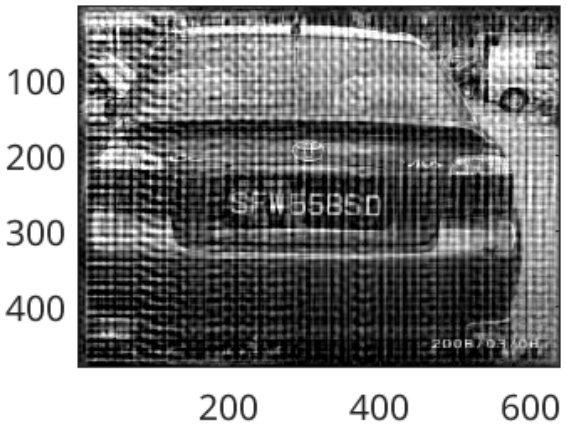


Figure 14: Deconvoled Sample Input Image 1

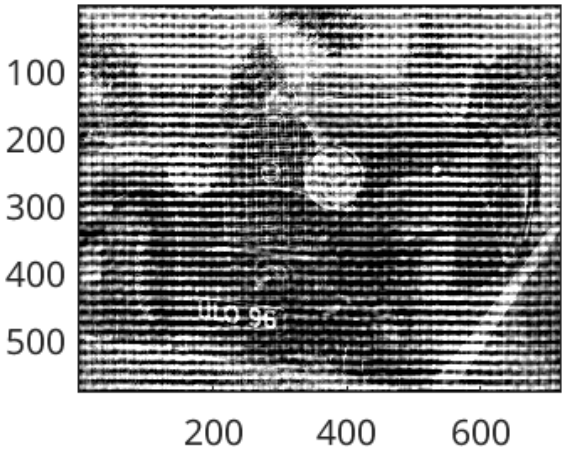


Figure 15: Deconvoled Sample Input Image 2

As a proof to that statement, we present Figure 22 and Figure 23. These figures show the deconvolution results after applying the Wiener deblur algorithm to the blurred images. Despite the attempt to deblur the images, the resulting outputs still exhibit substantial artifacts and the images being black. These artifacts are remnants of the severe degradation introduced by Laplacian and Sobel filters.

Deblurring Six Plate Numbers

In order to further test our implemented function, we tested this against six plate numbers that have been blurred. It is assumed that Gaussian filtered was applied. Our goal for this test is to guess which `K` parameter value works best in deblurring the plate numbers. Figure 24 shows the blurred plate numbers that needs to be deblurred using our implemented wiener deconvolution function and Figure 25 is the PSF guess. We tested guessing `K` at `0.01`, `0.001`, `0.003`, and `0.005`.

Based on our test results, the parameter with the value of `K` would be `0.003` since all the six plate numbers can be read. These six numbers are: `EN 1113`, `DT 1735`, `CD 7984`, `BT 9824`, `WYG 734`, and `IT 0420`.

Overall, these sample test cases highlight the expected output of our modified wiener blur function similarly to the laboratory exercise's outputs. The variation of the parameter K has a significant impact on both the corrupted data and the deconvolved image. When the K value approaches 1, excessive noise becomes evident in the output. Conversely, selecting a K value close to 0 minimizes the presence of noise,

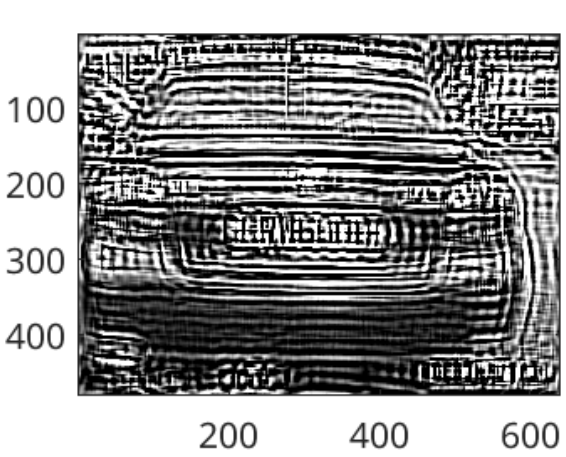


Figure 16: Deconvoled Sample Input Image 1

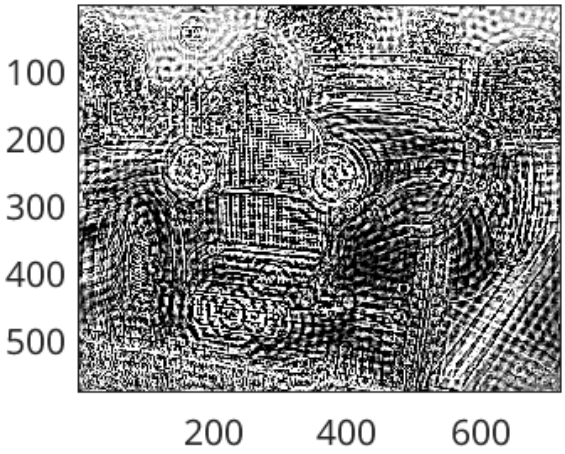


Figure 17: Deconvoled Sample Input Image 2

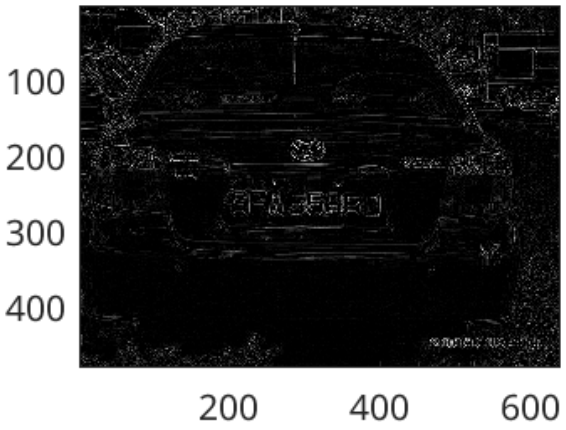


Figure 18: Corrupted Data of Sample Input Image 1

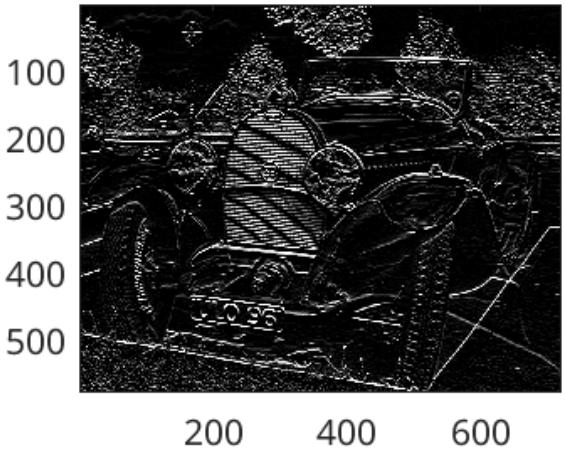


Figure 19: Corrupted Data of Sample Input Image 2

but introduces line-like artifacts in the resulting image. It is important to note that deblurring the image is a necessary prerequisite for the proper functioning of the Wiener Deconvolution algorithm.

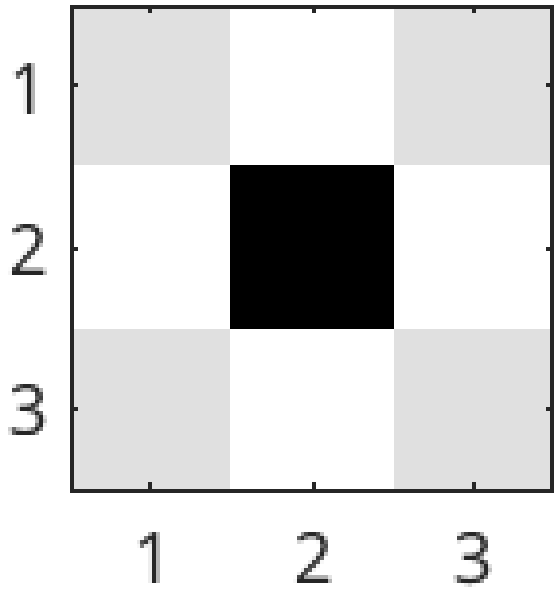


Figure 20: PSF of Sample Input Image 1

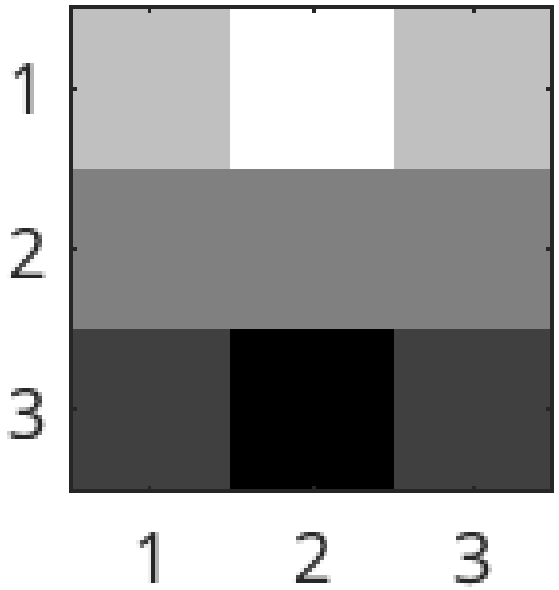


Figure 21: PSF of Sample Input Image 2

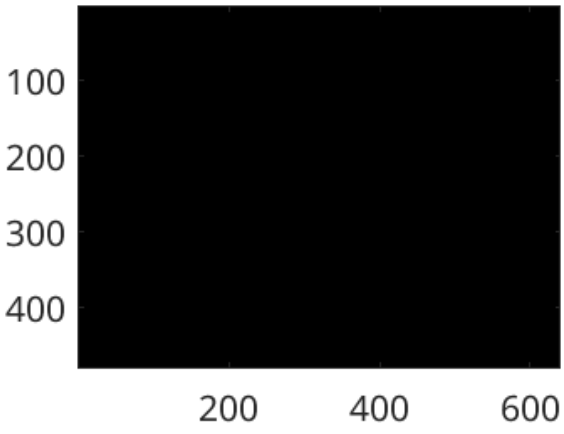


Figure 22: Deconvoled Sample Input Image 1

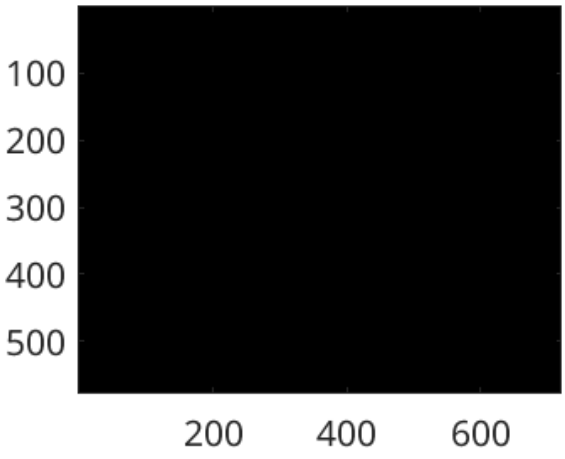


Figure 23: Deconvoled Sample Input Image 2

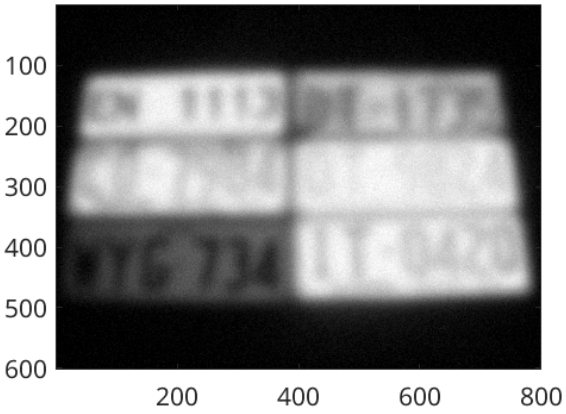


Figure 24: Blurred Six Plate Numbers

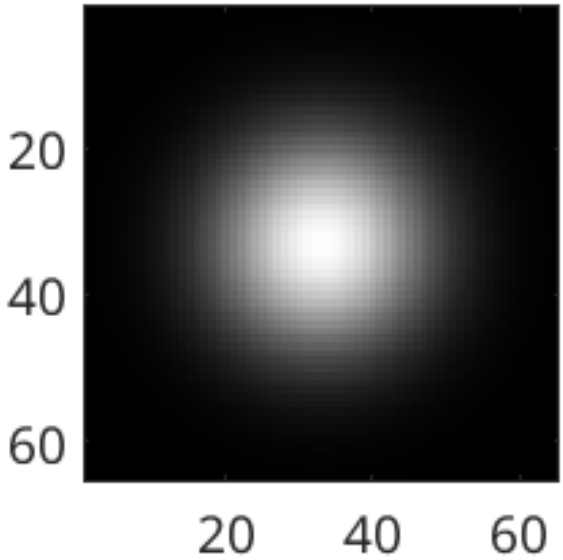


Figure 25: PSF Guess for Six Plate Numbers

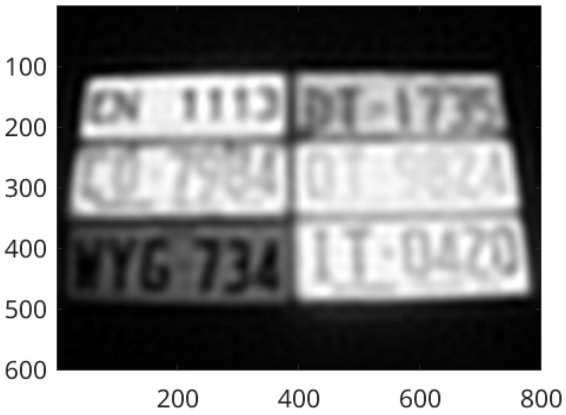


Figure 26: K = 0.01

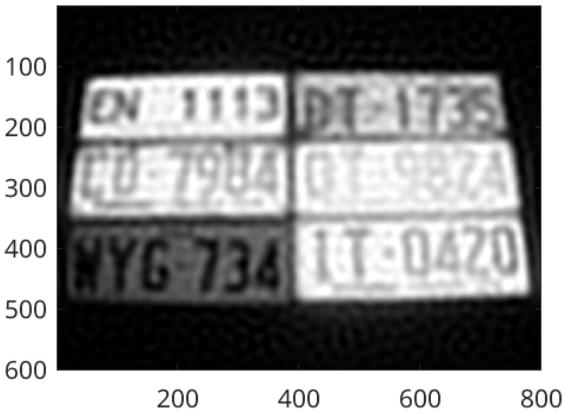


Figure 27: K = 0.001

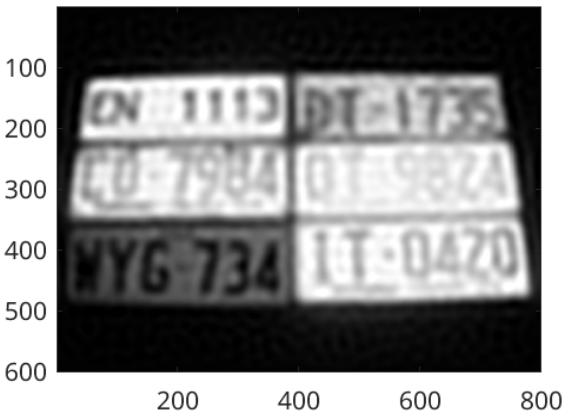


Figure 28: K = 0.003

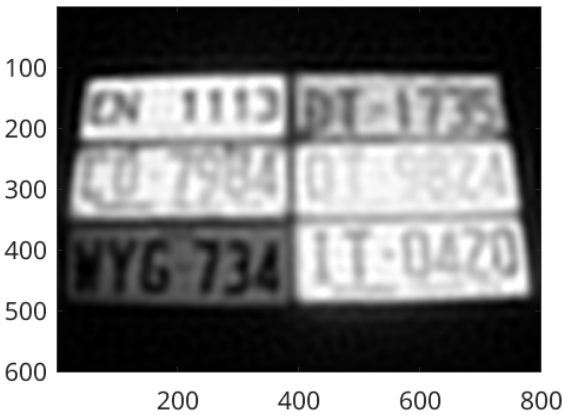


Figure 29: K = 0.005

4. Comments and Conclusion

The `wiener_deblur.m` function is implemented to deblur an input image using the Wiener deconvolution algorithm. The optimal output can be achieved by selecting suitable parameters or configurations. The key parameter in this function is the regularization parameter `K`. Adjusting the value of `K` allows controlling the trade-off between noise reduction and clarity of the deblurred image. Generally, a smaller value of `K` results in less noise but may introduce artifacts or line-like distortions in the output. On the other hand, a larger value of `K` would result in an increased noise standard deviation, leading to higher levels of noise in the output image. There is no best parameter or configurations since it would depend on the blurr level intensity of the input image. Experimenting the values for different images would be the best result.

One limitation of the our modified function is that it assumes the blurring model being used, such as a Gaussian blur kernel. While it may work well for images blurred by this model, it may not be as effective for images blurred using different kernels or complex blurring mechanisms which is evident in our Results and Discussions.

Exercise 2B – (5%) – Motion Image Compression

1. Background of the Problem

2. Procedure

There are two functions, `jpeg_8x8.m` and `simple_mpeg.m`, that are supplied from this laboratory exercise as a guide to modify two incomplete functions, `djpeg_8x8.m` and `simple_dmpeg.m`, that effectively decode an image and a video. We first focused on `djpeg_8x8.m` that implements the decoding process for the image. It first copies the DC coefficient to the quantized coefficient array. The first and second element or index in the array is the DC while the rest are the AC coefficients. Which is to be used for iterating the array in zig-zag pattern where the AC coefficient index is assigned to the quantized array while ensuring the direction. Where its next step is to acquire its quantization scale factor to determine the level of compression applied to the coefficients in the array. We then perform the inverse quantization step in the JPEG decoding process. It reconstructs the original DCT coefficients from the quantized coefficients by applying the quantization table and the Q scale factor. The result would be then a matrix that represents a reconstructed DCT coefficients which will be used for IDCT. In the final step, we use the inverse discrete cosine transform to convert the frequency coefficients back into pixel values. This process, performed with the help of the idct() function from the signal processing toolbox, allows us to reconstruct the 8x8 image block tile to its original form.

After performing the decoding process for images, the next step of the laboratory exercise is to implement the decoding of MPEG video formats. We modified `simple_dmpeg.m` to iterate over the 8x8 block in an image. Two frames are compared to each other and analyzed to see whether there is a change in pixels. Blocks that had similar coefficient values are ignored and blocks that do have differences are extracted. The intention of this function is not to iterate over each frames in the video but instead to compare two frames. It is the job of `mpeg_test.m` to iterate each frames. Once the decompression of the frame is done, an output is generated.

3. Results and Discussion

In this section, we show the results of `djpeg_8x8.m` and `simple_dmpeg.m` and analyze its outputs based on given parameter configurations. Sample input images are already provided. Using Figure 30 and Figure 38 as inputs, we where able to acquire its decompressed image shown on Figure 32 and Figure 33. Their difference is shown on Figure 34 and Figure 35.

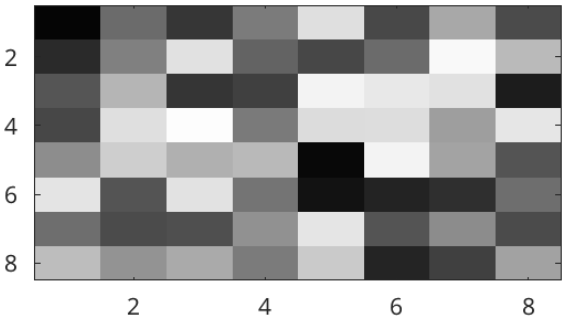


Figure 30: JPEG Input 1

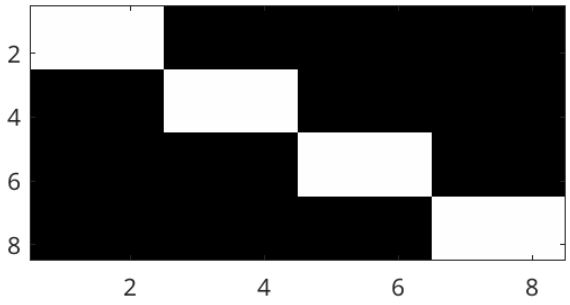


Figure 31: JPEG Input 2

Continuing on to the next test, we are subjected into testing the `Q` value first before testing the `tolerance` level. The test values used for `Q` are `1`, `25`, `50`, `80`, and `99`. We will be using Figure 36a and Figure 36b as the input and reference images for all the test cases, respectively, unless stated. By examining Figure 37a and Figure 41a, we can observe that the quality of the image frame is controlled by the parameter `Q`. A higher value of `Q` corresponds to higher image quality, while a lower value of `Q` corresponds to lower image quality. It is evident from Figure 44 that using a tolerance level of `50.0` results in disoriented pixels or artifacts. However, when the tolerance level is lowered, as shown in Figure 43, the image appears cleaner and clearer.

In summary, the parameter Q in the code determines the quality factor for image encoding. A higher value of Q leads to lower compression, preserving more image details and resulting in higher image quality.

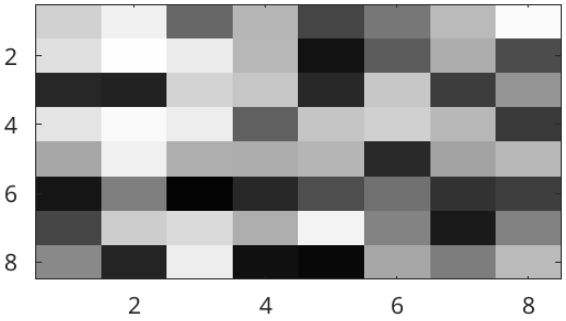


Figure 32: Decompressed Input 1

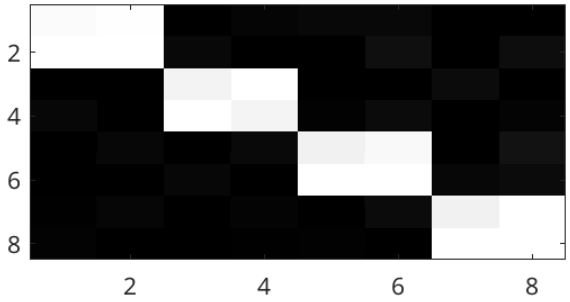


Figure 33: Decompressed Input 2

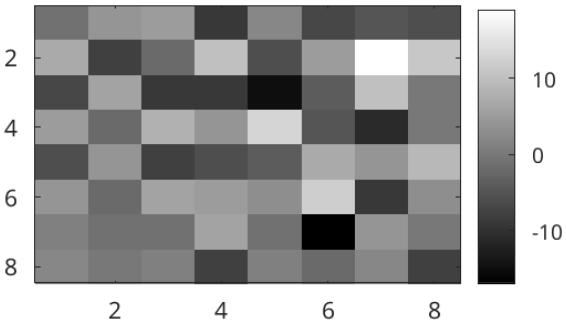


Figure 34: Input 1 Difference (rms 8.02)

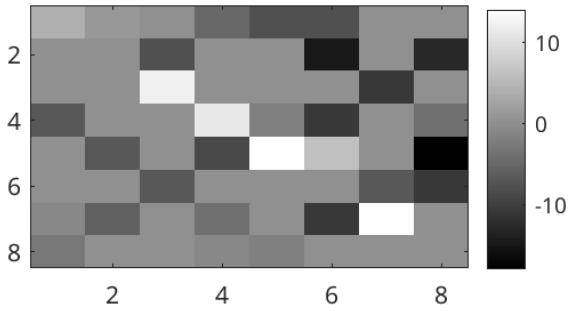


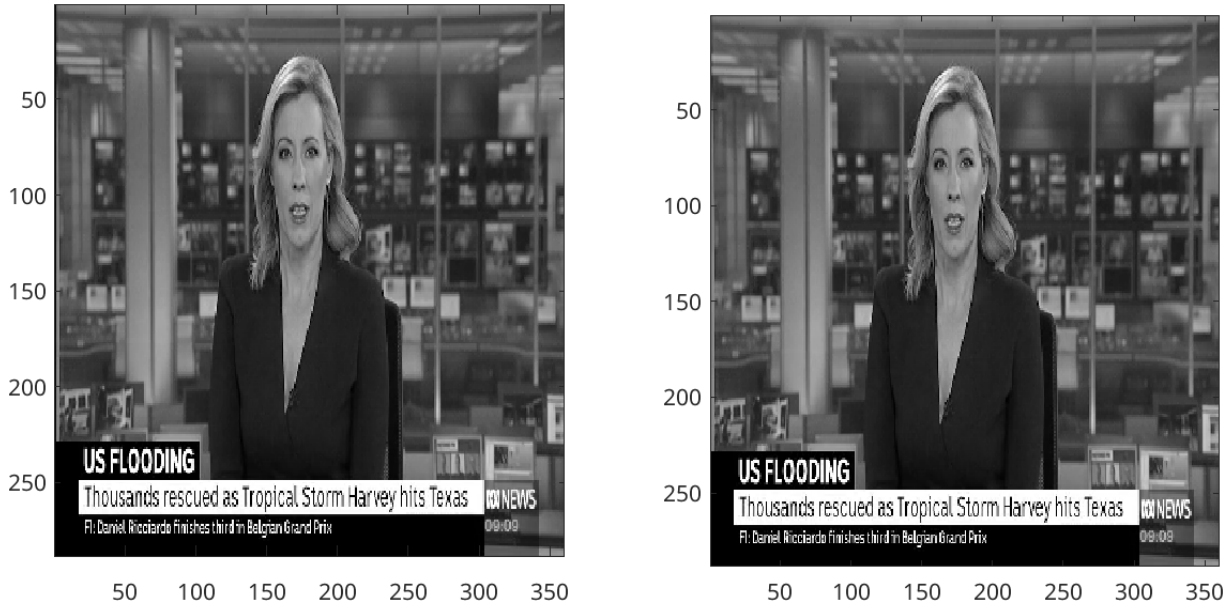
Figure 35: Input 2 Difference (rms 6.24)

Conversely, a lower value of Q increases compression, sacrificing some image details for higher efficiency. However, when the transmitted data gets reconstructed the image data or details are lost. The tolerance level, on the other hand, determines the level of change or difference between consecutive frames that will be considered significant enough to encode and transmit which is why having lower tolerance value would compress smaller compared to having high tolerance value which compresses huge data. Having a higher tolerance value would result image artifacts with deformed pixels.

4. Comments and Conclusion

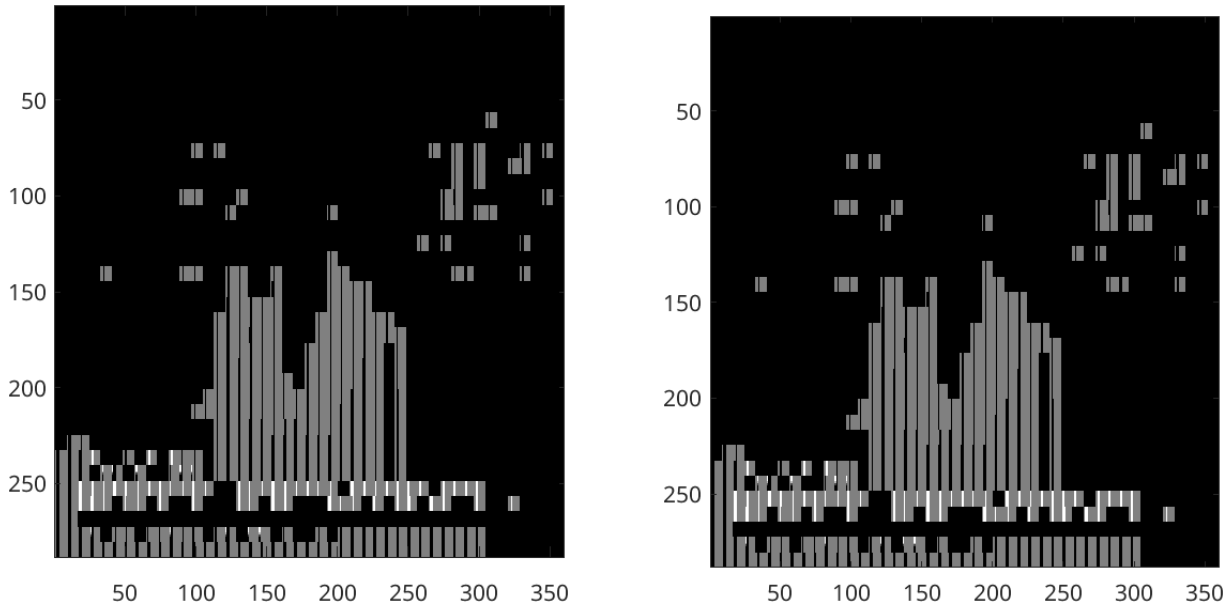
The completion of the `djpeg_8x8.m` and `simple_dmpeg.m` functions enables the decoding of JPEG images and MPEG videos, respectively. The decoding process involves quantization, inverse quantization, IDCT, and comparison of frames. The experiments conducted using different parameter configurations demonstrated the influence of the `Q` value and tolerance level on image quality and compression efficiency. A higher `Q` value resulted in higher image quality with less compression, while a lower `Q` value sacrificed image details for better compression efficiency. Lower tolerance levels produced cleaner images with fewer artifacts, while higher tolerance levels resulted in more noticeable artifacts.

These findings emphasize the importance of choosing appropriate parameter values in image and video decoding to balance the trade-off between image quality and compression efficiency. The completed functions provide a foundation for further research and development in the field of image and video processing.



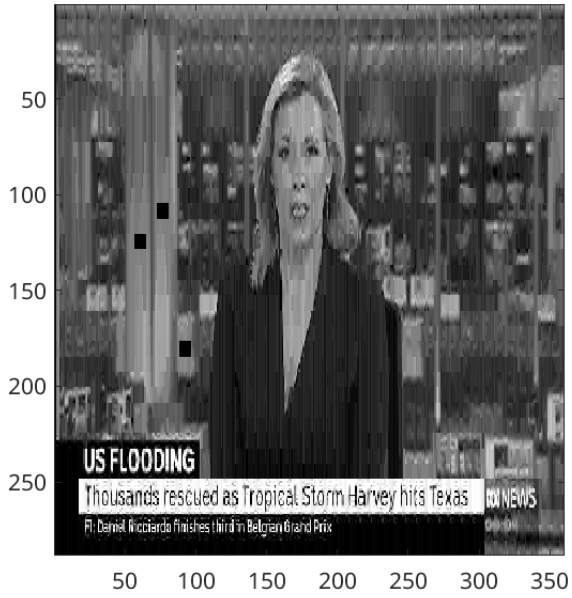
(a) Input 1 (103680 bytes total) (b) Reference

Figure 36: Input 1 at any Q and Tolerance values

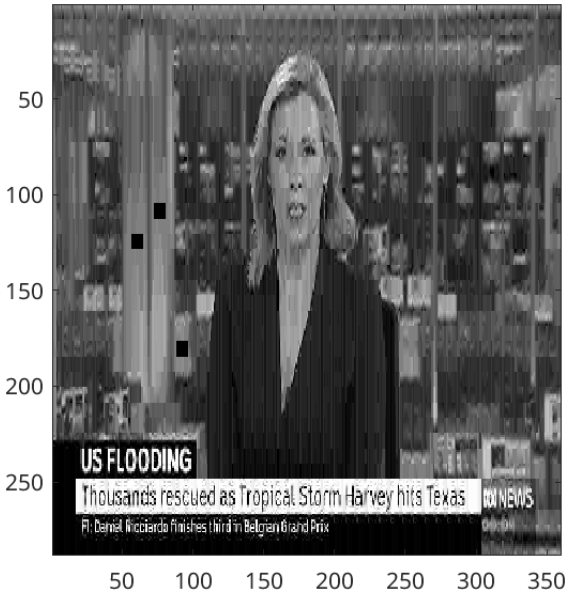


(a) Transmitted Data (516 bytes) (b) Reconstructed (516 bytes total)

Figure 37: Q = 1, Tolerance = 5.00

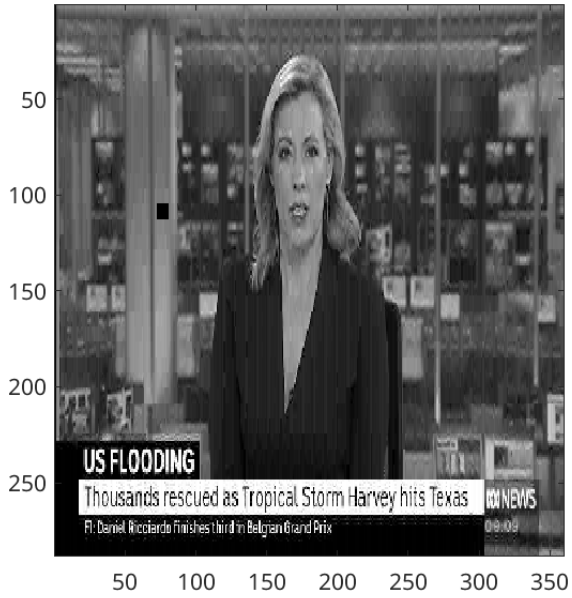


(a) Transmitted Data (15729 bytes)

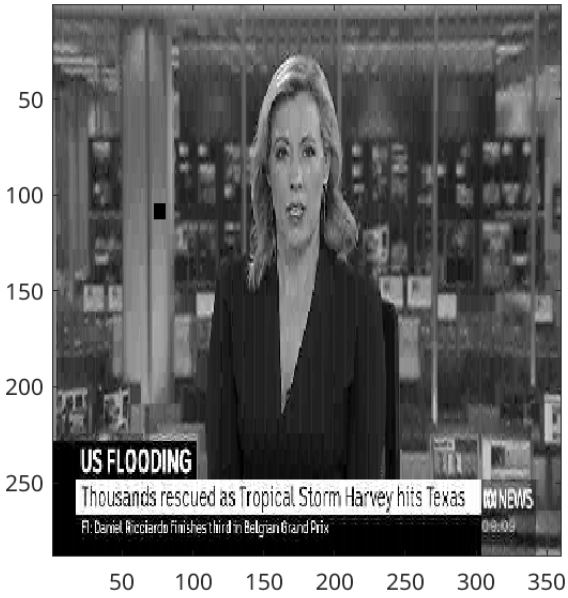


(b) Reconstructed (15729 bytes total)

Figure 38: Q = 25, Tolerance = 5.00

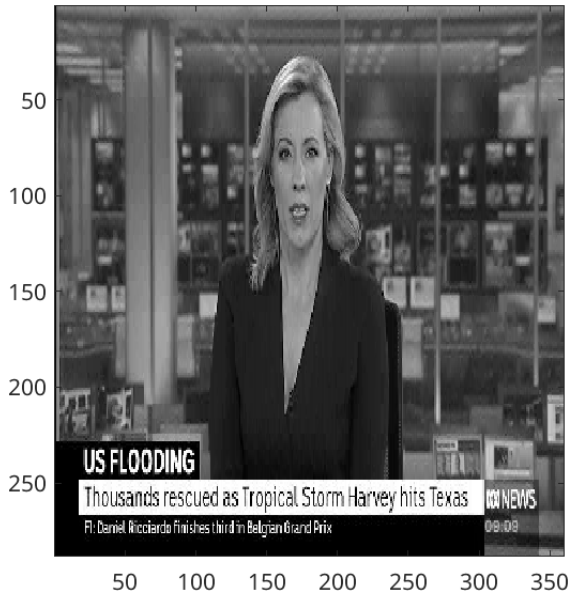


(a) Transmitted Data (21862 bytes)

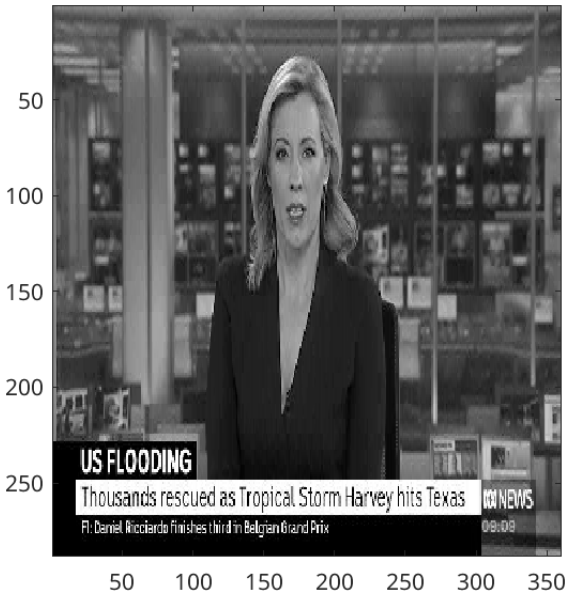


(b) Reconstructed (21862 bytes total)

Figure 39: Q = 50, Tolerance = 5.00



(a) Transmitted Data (31383 bytes)



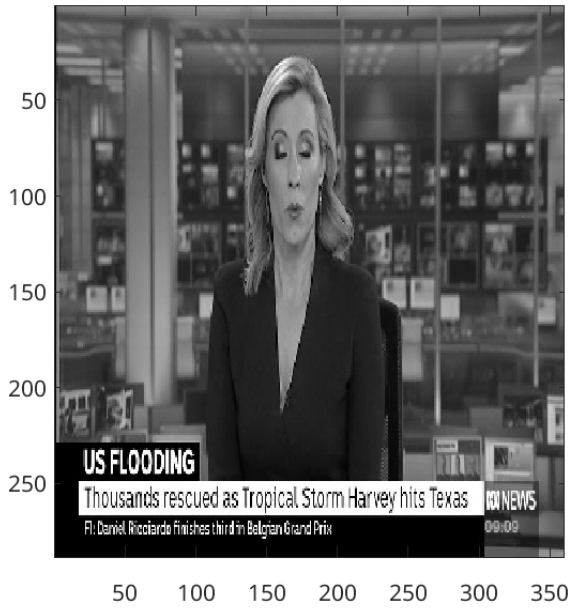
(b) Reconstructed (31383 bytes total)

Figure 40: Q = 80, Tolerance = 5.00

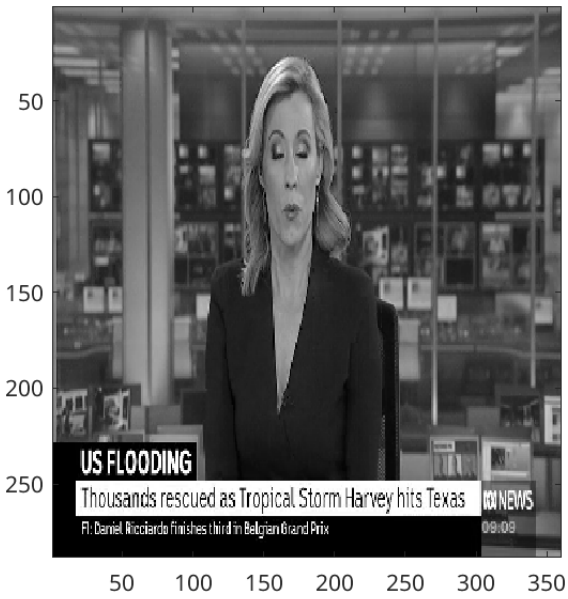
(a) Transmitted Data (83173 bytes)

(b) Reconstructed (83173 bytes total)

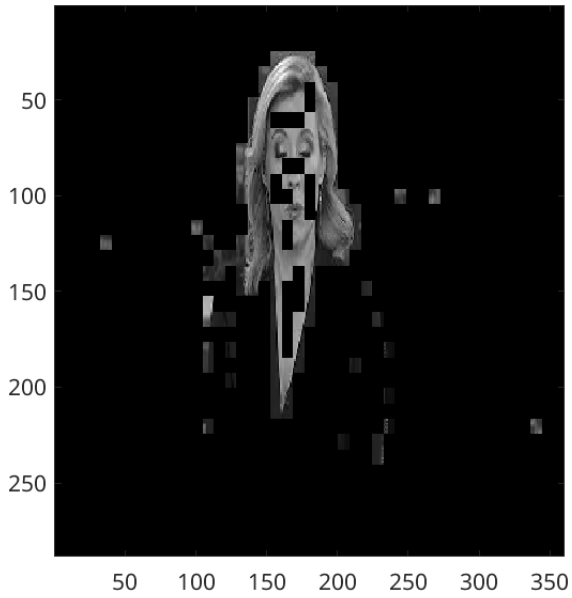
Figure 41: Q = 99, Tolerance = 5.00



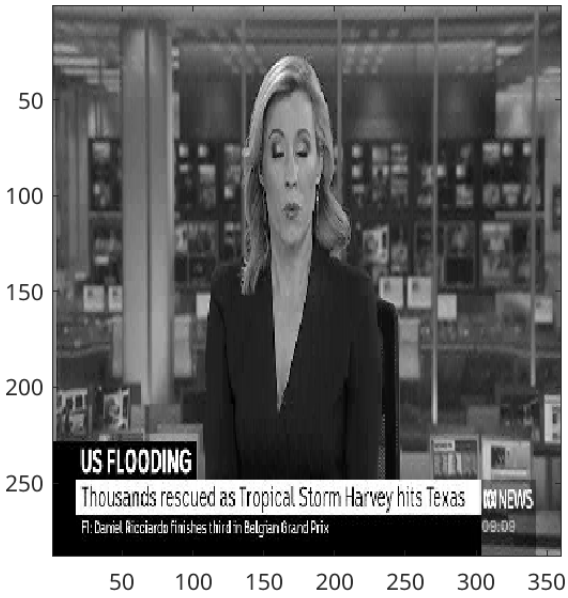
(a) Frame 8



(b) Reference



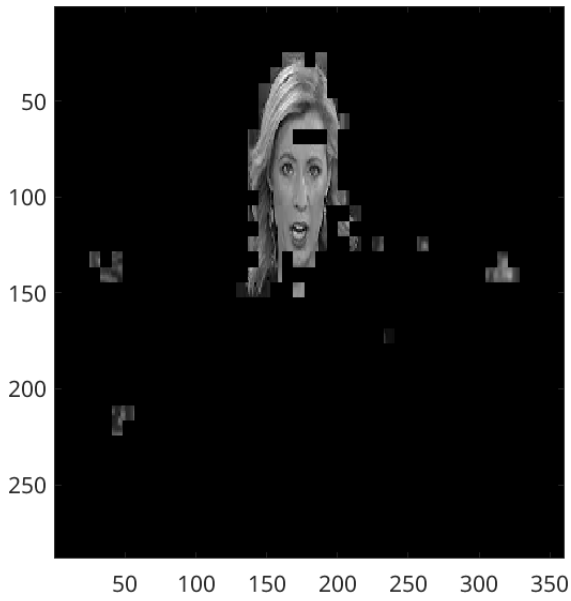
(c) Transmitted Data (3674 bytes)



(d) Reconstructed (58437 bytes total)

Figure 42: Q = 80, Tolerance = 5.00

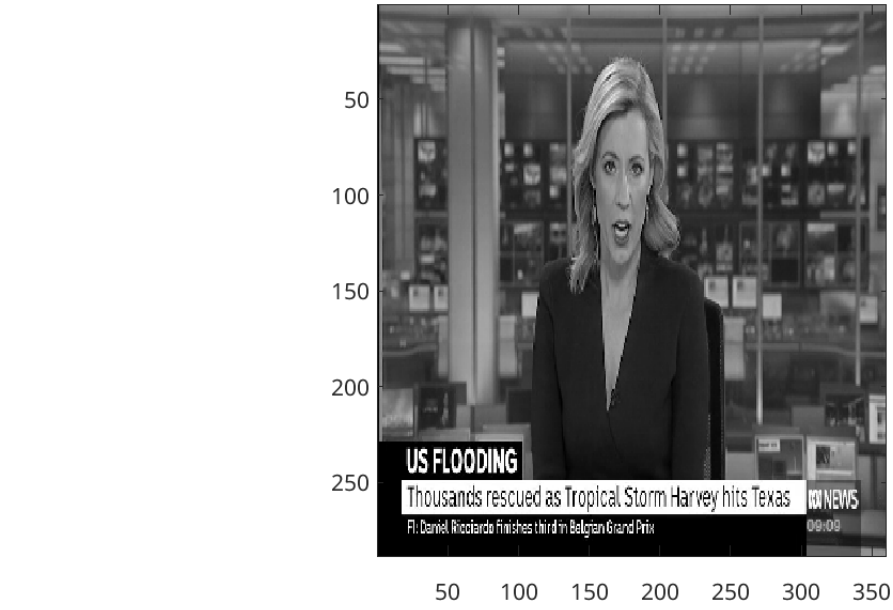
(a) Input 100 (10368000 bytes total)



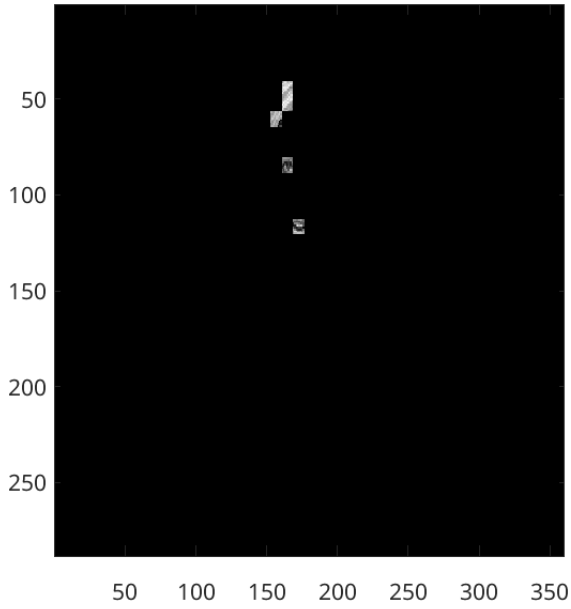
(b) Transmitted Data (2843 bytes)

(c) Reconstructed (552853 bytes total)

Figure 43: Q = 80, Tolerance = 5.00



(a) Input 100 (10368000 bytes total)



(b) Transmitted Data (638 bytes)

(c) Reconstructed (347376 bytes total)

Figure 44: Q = 80, Tolerance = 50.00

Exercise 2C – (2%) – Written Questions

- **2C.1** (0.5%)

  A. Considering the absence of lens distortion, the optimal transformation to capture the relationship between images in this situation is a projective linear transformation. This type of transformation is particularly suited for scenarios where points in an image plane need to be mapped to points in a 3D space, and vice versa. It enables the projection of 3D points onto the 2D image plane, as illustrated in the lecture material on spatial transformations. When it comes to celestial objects like Jupiter, the projective linear transformation is an ideal choice for capturing their projection onto a 2D image plane. The camera records the celestial body's appearance in a 2D format, with the transformation preserving the relative positions and shapes of objects within the scene.

  When the telescope is adjusted or moved, it captures different views of the celestial body, altering the perspective of the scene being recorded by the camera. Each change in perspective results in a unique mapping of the 3D scene onto the 2D image plane. Consequently, the captured image presents a specific viewpoint of the camera, showcasing the celestial body from a particular vantage point.

  B. In the given scenario, where the transformation of the document itself is the focus rather than the room, the most appropriate transformation would be an `affine linear transformation`. This choice is based on the requirement to maintain the document's proportions and parallelism when it is photographed and viewed from different angles and positions. Other linear transformations are not suitable for this situation. Like for example, projective transformation, not preserve parallel lines and involves planar scaling, which is not applicable in this case where the document is observed from various positions within the room. Isometry, another type of transformation, preserves distances, but it is not feasible in this context as the observer moves to different locations in the room. Linear-conformal transformation, which preserves shape, is also unsuitable as the picture may be taken from different angles, resulting in a change in shape. Translation, which is a simple rigid transformation involving movement, is not relevant here since the viewpoint remains fixed.

  Therefore, the `affine linear transformation` stands out as the most appropriate choice for maintaining the document's proportions and parallelism while allowing for variations in viewing angles and positions. This makes sure that the document appears consistent and accurately represents its original geometry despite changes in perspective.

- **2C.2** (0.5%)

  Given the matrix transformation of linear-conformal, we equate the given matrix representation of the linear transformation and then solve for the values.

$$T = \begin{bmatrix} sR_2 & t \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} s(\cos(\theta)) & s(\sin(\theta)) & t_x \\ s(-\sin(\theta)) & s(\cos(\theta)) & t_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1.7321 & 1 & 10 \\ -1 & -1.7321 & -20 \\ 0 & 0 & 1 \end{bmatrix}$$

  Then we have these system of equations:

  $t_x = 10$; $t_y = -20$; $s(\cos(\theta)) = 1.7321$; $s(\sin(\theta)) = 1$

  Solving for the rotation:

  $s = \frac{1.7321}{\cos\theta}$; $s = \frac{1}{\sin\theta}$

  Equating those into:

$$\frac{1.7321}{\cos\theta}\sin\theta = 1$$
$$1.7321\tan\theta = 1$$
$$\tan\theta = \frac{1}{1.7321}$$
$$\theta = \arctan(\frac{1}{1.7321})$$
$$\theta = 30^\circ$$

Using the θ value we acquired, we can now get the scale value:

$$s = \frac{1.7321}{cos\theta}$$
$$s = \frac{1.7321}{cos(30^\circ)}$$
$$s \approx 2.000056803 = 2$$

Thus, our translation values are:

$$t_x = 10$$
$$t_y = -20$$
$$\theta = 30^\circ$$
$$s = 2$$

- **2C.3** (0.5%)

  We will use Equation 1 for the bilinear interpolation:

$$\begin{aligned} I(p) = &[(1-a)(1-b)(I(x,y))] \\ &+ [(1-a)(b)(I(x,y+1))] \\ &+ [(a)(1-b)(I(x+1,y))] \\ &+ [(a)(b)(I(x+1,y+1))] \end{aligned} \tag{1}$$

  In order to get the values a and b for the bilinear estimates for pixel (20.6, 10.2), we can use Equation 2 and Equation 3:

$$\begin{aligned} a &= p_x - \lfloor p_x \rfloor \\ &= 20.6 - \lfloor 20.6 \rfloor \\ &= 20.6 - 20 \\ a &= 0.6 \end{aligned} \tag{2}$$

$$\begin{aligned} b &= p_y - \lfloor p_y \rfloor \\ &= 10.2 - \lfloor 10.2 \rfloor \\ &= 10.2 - 10 \\ b &= 0.2 \end{aligned} \tag{3}$$

  Now that a and b values have been determined, we can substitute those values into Equation 1. Thus, we get Equation 4:

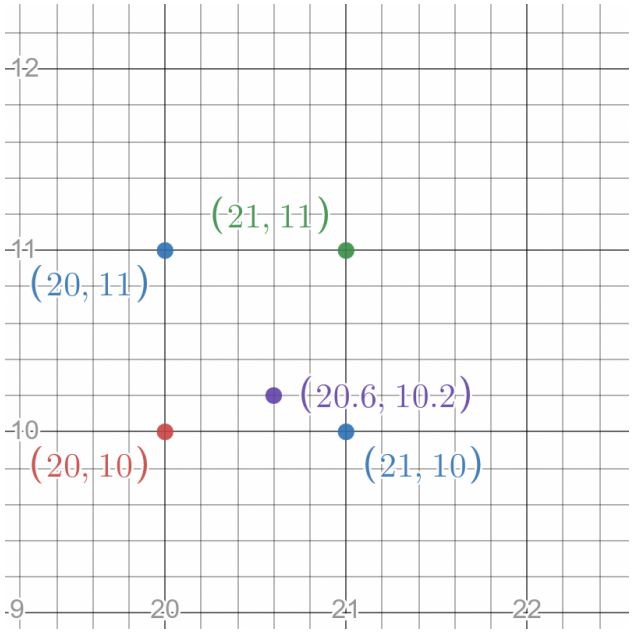


Figure 45: Nearest neighboring pixels of (20.6, 10.2)

$$
\begin{aligned}
I(p) &= [(1-0.6)(1-0.2)(128)] + [(1-0.6)(0.2)(64)] + [(0.6)(1-0.2)(64)] \\
&\quad + [(0.6)(0.2)(32)] \\
&= [(0.4)(0.8)(128)] + [(0.4)(0.2)(64)] + [(0.6)(0.8)(64)] + [(0.6)(0.2)(32)] \\
&= 40.96 + 5.12 + 30.72 + 3.84 \\
I(p) &= 80.64
\end{aligned} \tag{4}
$$

Thus, the bilinear estimate for the pixel $(20.6, 10.2)$ is `80.64` and its nearest neighbor is $(21, 10)$ shown on Figure 45. However, if we follow the formula

- **2C.4** (0.5%)

  Based on a detailed analysis of the spectral characteristics and their corresponding outputs, we can draw specific conclusions regarding the relationship between `Spectra 1`, `Spectra 2`, and their respective outputs.

  Firstly, `Spectra 1` can be attributed to the generation of `Output 2` due to its distinctive composition of black and white. This composition is reflected in the resulting image, where the cameraman is prominently highlighted in white against a mostly black background with subtle gray shades. The presence of blacks and whites in `Spectra 1` contributes to the stark contrast between the subject and the background, emphasizing the cameraman's features. Notice that `Spectra 1` shows a relatively smoother profile without any prominent stripes, implying a more uniform background. This aligns with the characteristics of `Output 1`, which exhibits a consistent and evenly distributed response

  As for `Spectra 2`, it is found to be responsible for the creation of `Output 1`. This conclusion is supported by the predominantly gray background observed in the resulting image. The grayscale background, with a central black line indicating the presence of dark edges around objects, aligns with the spectral characteristics of `Spectra 2`. The absence of distinct black and white regions suggests a more uniform distribution of gray tones, resulting in a less contrasting image.