7060 Image Sensors & Processing

Assignment 2

April 2019

(Assignment Value: 12%)

Dr D.Gibbins

Overview

This assignment will give you some hands-on experience with FFT filtering, JPEG compression and some written exercises on spatial transforms. This covers material from weeks 4 to 6.

IMPORTANT:

Only the following routines from the image processing toolbox may be used for **testing** purposes: *`imshow()`* *`imnoise()`* *`imfilter()`*. The function *`edgetaper()`* may only be used in exercise 2A. Using other routines from the image processing toolbox (unless approved by me in class or tutorials) may result in zero marks for that component of the assignment work.

You will also need to use functions such as `fft2()` `ifft2()` `real()` `imag()` `abs()` `fftshift()` and `ifftshift()` from the signal processing toolbox in order to complete exercise 2A. If you have not used these MATLAB functions before these functions are covered in tutorial 6 which can be found on MyUni.

Assignment Submission

Assignments are to be submitted via MyUni as a ZIP archive file containing the key MATLAB functions and a Word or PDF document summarising your results with comments on the performance of the processing steps employed.

Marks will be deducted for late submission.

Source Materials

Source code, test functions and example imagery for this assignment are located on the MyUni website (https:// myuni.adelaide.edu.au).

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

Your company has also given you the task of working on an automated number-plate recognition system for identifying illegally parked cars. Unfortunately the images of the vehicles and their number plates are blurred and as a result the character recognition algorithms you are trying to use are not working. There is also a small amount of noise in the imagery just to make things harder.

Fortunately for you, the camera manufacturer is aware of the problem and can supply you with functions to generate filters which match the type of blurring effects seen in the data. As a result you have decided to try using a Wiener filter to de-blur the imagery.

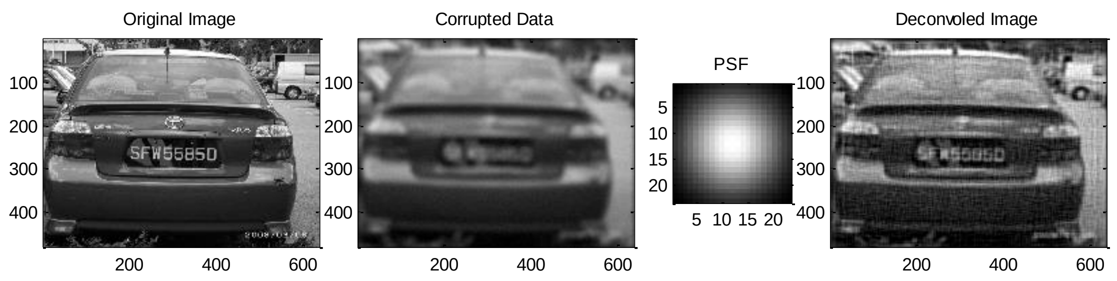
STEP 1: Modify the supplied M-file `wiener_deblur(I,B,k)` such that given a blurry image I, an estimate of the blurring function B, and a noise term k, the function implements a simple Wiener inverse filter (see lecture 9) of the form:

$$F_{inv} = \frac{F_I}{F_B}\left(\frac{|F_B|^2}{|F_B|^2 + k}\right)$$

← The division operations shown in this expression represent cell by cell division of the coefficients NOT matrix division

Where F_I , F_B and F_{inv} are the Fourier representations of the input image I, blurring function B (padded to the same size as I) and the inverse filtered image respectively. The k term (proportional to the noise standard deviation times the number of pixels in the image), is used to dampen the inverse operation and importantly to reduce sensitivity to any image noise present. Increasing this value reduces sensitivity to noise but at the cost of less inverse filtering. Lowering this value can improve detail but at the cost of increased image artefacts.

IMPORTANT: In real-world applications to reduce edge ringing you need to smooth off the image near the image boundaries. This is a little too complicated for this assignment, so rather than do this yourself the supplied code uses the matlab function **edgetaper.m**.



An illustration of this Weiner filtering process is shown above.

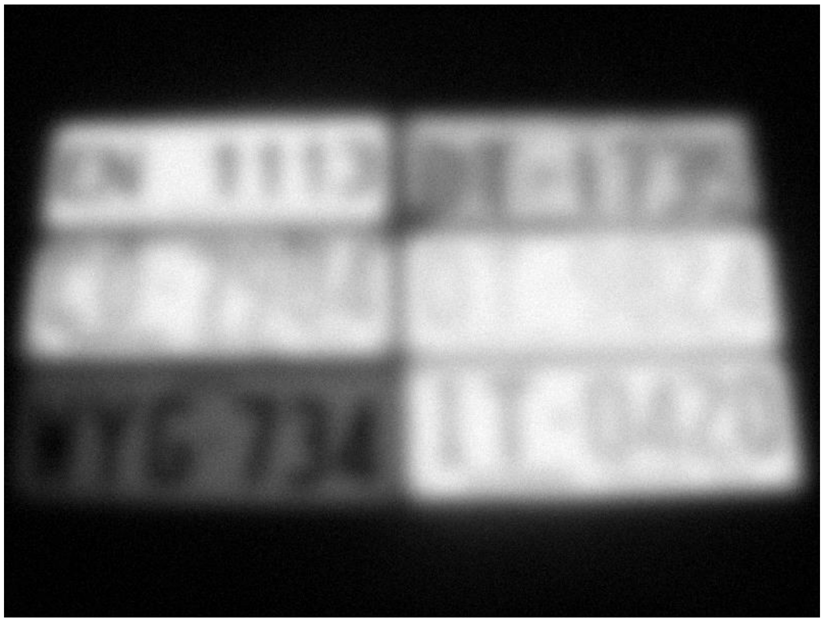
The functions `blur_image` and `noisy_image` are provided as is a copy of the `fspecial` function.

STEP 2: Use the test script `deconv_test.m` provided to check your results. Also try the method out for different values of K examples and different blurring functions. Write up your results.

STEP 3: You have also been given a blurred image of six number-plates to identify (example3.png) This image was blurred using a known Gaussian filter however the exact noise level is unknown.

Your task is to attempt to figure out, by trial and error what fine tuning of the 'k' term will give you enough detail to be able to make out the registration numbers. The test script `deconv_test2` has been supplied to you and this will prompt you for a variance and weighting to try.

A blurry image of the number-plates image used in `deconv_test2` is shown below:



Can you successfully deblur the image figure out what the six number plates are?

Hints / Advice:

- The MATLAB functions *fft2*, *ifft2*, *fftshift* and *ifftshift* can be used here to go to/from the Fourier domain (and shift the coefficients tho' this is not necessary to solve this exercise). Note that *ifft2* returns a complex image. The functions *real* and *imag* can be used to get the real and imaginary components. The function *abs* returns the magnitude of a complex number. The inverse may also contain values outside of the range of the original image data (use min and max to fix this). Note also that these functions leave the DC term at location (0,0) in the image (ie. it is not in the centre as shown in the lecture materials).
- The blurring function will probably be smaller than the input image so you will need to zero pad the blurring filter (point spread function) to make it the same size as the image before computing the Fourier coefficients. Zero padding the blurring function will however add a spatial delay to the filter so the output may be shifted by several rows and columns. You will need to handle this in your solutions.
- Make sure you understand the difference between the operations "/" and "./" in MATLAB. The latter is cell by cell division.
- You may also need to take care of any small or zero values in the spectrum. This is especially the case for the $1/F_B$ component of the expression.
- Do <u>not</u> remove the line I = edgetaper(I,B) from the solution code this is included to prevent the edge ringing effects we have discussed in class.
- It is recommended that you initially test your de-convolution function in the absence of noise.

Exercise 3C – Motion Image Compression (5%)

As the final coding exercise you are going to complete the implementation of a simplistic motion jpeg (MPEG) encoding scheme for greyscale imagery. This scheme uses a simplistic jpeg compression of the imagery in 8x8 blocks (as covered in the Week 5 lecture) but in addition only encodes the scene changes from each image frame which further reduces the amount of information required to encode the sequence of images. For this exercise you will be required to complete the jpeg and mpeg reconstruction steps (using the supplied jpeg and mpeg encoding steps as a guide).

The process used to select which 8x8 blocks are encoded is as follows:

1. *For each 8x8 block in the new image:*
    a. *Estimate the mean square error between the 8x8 block from the new image and the old image.*
    b. *If the mean square error is greater than some given tolerance (or the current frame is a key-frame) then jpeg compress the 8x8 block and add the coefficients to the list of AC and DC coefficients to be Huffman encoded. Otherwise pass a block of zero coefficients to the list to be encoded*
2. *Huffman encode the list of coefficients.*

In the above process blocks that are not jpeg encoded are represented by zero valued dc and ac coefficients. Here we are taking advantage of the fact that the Huffman encoding step will be able to significantly compress of any sequences of 64 zeros in the coefficient list. This bank of zero coefficients can be used in the decoding stage to indicate an 8x8 block which does not require updating.

The dc coefficients are stored as an NxM array (1/8th the size of the original image). The ac coefficients are stored as an NMx63 array. These two sets of data are vectorised and passed to the final Huffman encoding step.

The process used for jpeg compression of the 8x8 blocks of image data is similar to that shown in lectures and is as follows:

1. *Perform a dc shift (subtract 128) of the block of greyscale data*
2. *Compute the DCT of the 8x8 block of data (8x8 array*
3. *Use the supplied Q (quality) term to determine the scale factor to apply to the base quantisation matrix (see below)*
4. *Perform jpeg quantisation of the DCT coefficients using the base quantisation matrix (see below) and the scale factor*
5. *Extract the dc term from the quantised DCT coefficients (first element of 8x8 array)*
6. *Extract out the ac terms into a 63 element list in zig-zag order (see below)*

In the above the base quantisation matrix for the 8x8 block of DCT coefficients is as follows:

16	**11**	**10**	**16**	**24**	**40**	**51**	**61**
12	**12**	**14**	**19**	**26**	**58**	**60**	**55**
14	**13**	**16**	**24**	**40**	**57**	**69**	**56**
14	**17**	**22**	**29**	**51**	**87**	**80**	**62**
18	**22**	**37**	**56**	**68**	**109**	**103**	**77**
24	**35**	**55**	**64**	**81**	**104**	**113**	**92**

49	**64**	**78**	**87**	**103**	**121**	**120**	**101**
72	**92**	**95**	**98**	**112**	**100**	**103**	**99**

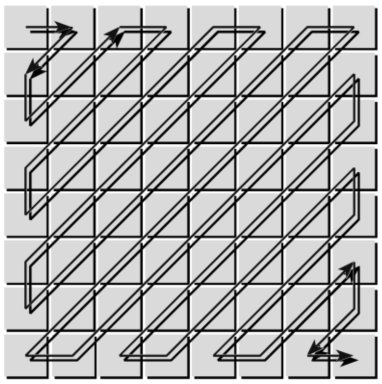
Here the quantisation increases as the DCT frequency increases (diagonally from top-left to bottom right).

The quantisation scale factor in step 3 is calculated as follows:

If Q<=50 then scale is 50/Q else scale is (2-Q/50)

Where Q ranges from 1 to 99 (do not use values of 0 or 100).

Finally, extraction of the quantised coefficients (step 6) uses the zig-zag pattern as shown below:



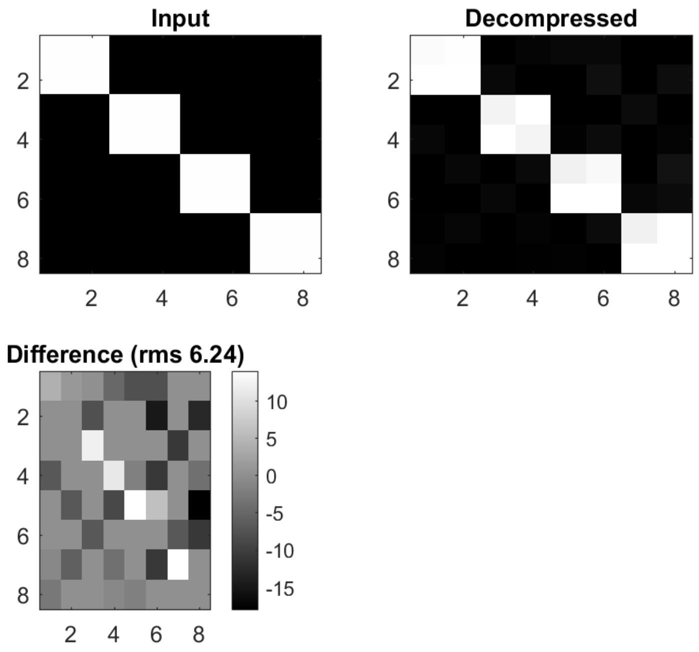
In this assignment the code for performing the Huffman encoding and decoding (Huff06.m), the 8x8 jpeg encoding (jpeg_8x8.m) and the mpeg encoding (simple_mpeg.m) has been supplied. Your task is to reverse the process and reconstruct the image by modifying the supplied functions djpeg_8x8.m and simple_dmpeg.m. These functions can be tested using the two test functions jpeg_8x8_test.m and mpeg_test.m.

STEP 1: Using **jpeg_8x8.m** as a guide modify **djpeg_8x8.m** such that given a dc component and 63 ac components it reconstructs the original 8x8 image patch. In essence you will need to reverse the processing steps defined on the previous page and unzip the elements back into a 8x8 array, reverse the quantisation step, and invert the dct using the function **idct()** and remove the 128 offset.

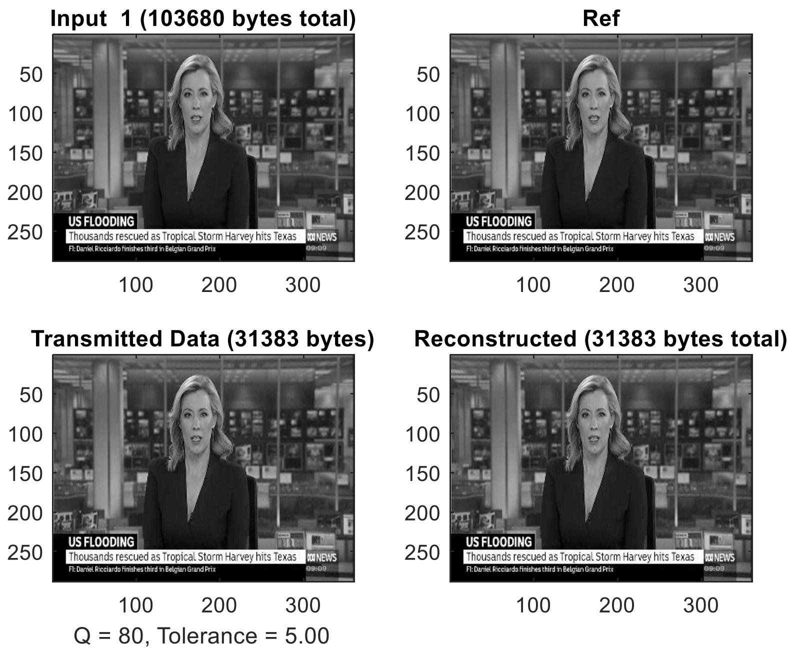
STEP 2: Check your solution using the **jpeg_8x8_test** script. Alter the Q factor and observe the effect on image quality as Q decreases. For a Q of 80 errors should be around a maximum of +/-10 greylevels for the test patches.

STEP 3: Using **simple_mpeg.m** as a guide modify **simple_dmpeg.m** such that given a string of Huffman encoded data and the previous reconstructed image it decompresses the jpeg encoded data and reconstruct the new image frame. Use the fact that regions of the image not requiring updating should have all of their ac and dc terms set to zero (as noted on the previous page).

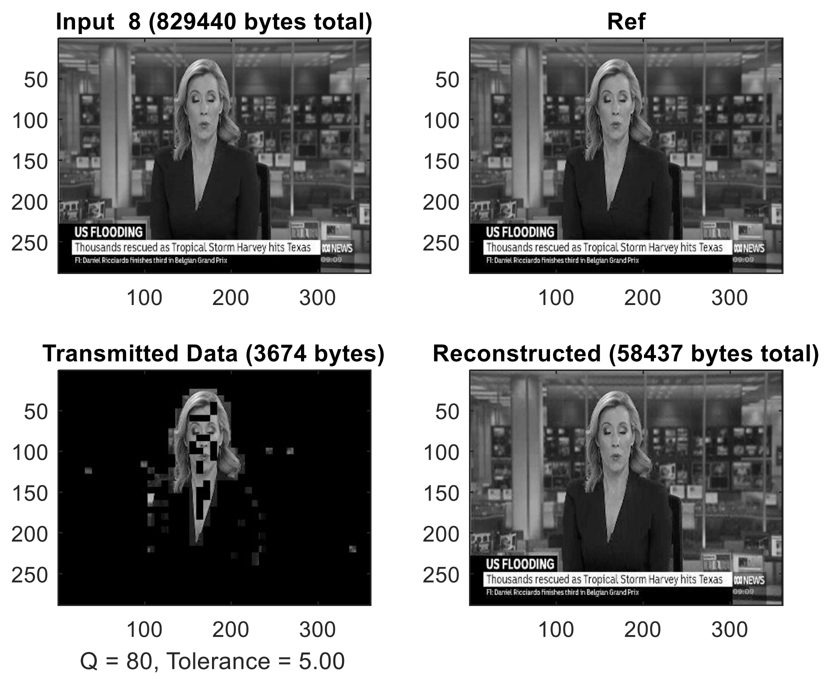
STEP 4: Check your solution using the **mpeg_test** script. Once you have your code working, alter the Q factor and tolerance values and observe the effect on the total number of bytes and reconstructed image quality. A movie file with an example result for a Q of 80 and a change tolerance of 5.0 has been included. If you wish to calculate bit error rates etc, or use your own imagery then feel free to create a modified version of the **mpeg_test** function (if you do then please include these files in your submission). If your code is working correctly you should get a result of around 553,000 bytes total for the 100 image sequence (Q=80, Tolerance=5).



Above: example output from jpeg_8x8_test.m (Step 2).


Input 1 (103680 bytes total)
Ref
Transmitted Data (31383 bytes)
Reconstructed (31383 bytes total)
50
100
150
200
250
300
Q = 80, Tolerance = 5.00

Above: Example compression result for the first image frame using ***mpeg_test.m*** *for a Q of 80 and a change tolerance of 5. The current image and the current reconstruction are shown in the top-left and bottom-right respectively. The current byte totals are also shown.*


Input 8 (829440 bytes total)
Ref
Transmitted Data (3674 bytes)
Reconstructed (58437 bytes total)
Q = 80, Tolerance = 5.00
50
100
150
200
250
100
200
300
US FLOODING
Thousands rescued as Tropical Storm Harvey hits Texas
NEWS

Above: Example compression result for a later image frame (frame 8) using ***mpeg_test.m****. The Reference image (top right) is the encoders version of the current image based on updates from detected changes. The transmitted data (lower left) is the detected changes between the current and previous images. The bottom-right image is the reconstruction based on the previous reconstruction and the new updated regions (PLEASE NOTE: Your own results may differ slightly from the example shown here)*

Exercise 2C – (2%) – Written Questions

2C.1 (0.5%) - In terms of the simple linear transformations ‘translation’, ‘isometric’, ‘linear conformal’, ‘affine’ and ‘projective’. Ignoring lens distortion, which transformations best describe the relationships between images in the following situations (you must explain your reasoning for each your answers.):

A. A camera mounted on a telescope looking up at a celestial body (eg. Jupiter). The telescope can tilt up and down and left and right (assume the scene itself is static).

B. A series of photographs of a document sitting on a desk taken from different positions throughout the room (just consider the transformation of the document itself, not the rest of the room).

2C.2 (0.5%) – Linear transformations are transformations of the form:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = T \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Assuming the following matrix T represents a ‘linear conformal’ transformation what are the underlying <u>translation</u>, <u>rotation</u> and <u>scale</u> values of this transformation?

$$T = \begin{bmatrix} 1.7321 & 1 & 10 \\ -1 & 1.7321 & -20 \\ 0 & 0 & 1 \end{bmatrix}$$

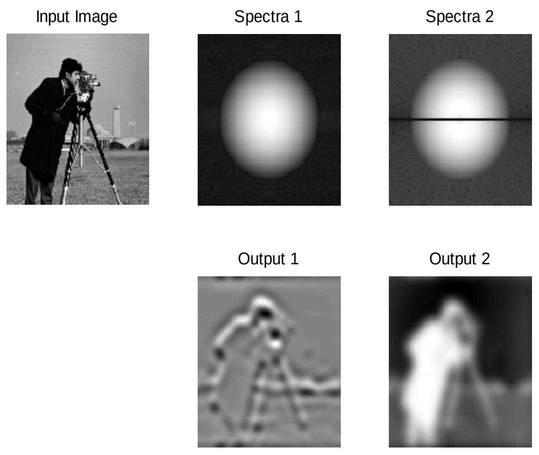
You must clearly state your reasoning for each of the four estimates.

2C.3 (0.5%) – Bilinear interpolation uses a 2x2 patch of image data to estimate pixel values after a transformation has been applied. If the 2x2 patch of an image for coordinates (20,10) to (21,11) is given by the greylevel values:

128	64
64	32

What are the nearest and bilinear estimate for pixel (20.6, 10.2) ?

2C.4 (0.5%) – An image has been processed using two different filters. The spectra of each filter is shown along with the two outputs. Based on the filter spectra carefully explain which filter produced which output:



Q: Which spectra gave rise to which output?