CE4003/CZ4003/CPE412/CSC408 : Computer Vision

Lab 1: Point Processing + Spatial Filtering + Frequency Filtering + Imaging Geometry

1. Objectives

This laboratory aims to introduce image processing in MATLAB context. In this laboratory you will:

a. Become familiar with the MATLAB and Image Processing Toolbox software package.
b. Experiment with the point processing operations of contrast stretching and histogram equalization.
c. Evaluate how different Gaussian and median filters are suitable for noise removal.
d. Become familiar with the frequency domain operations
e. Understand imaging geometry.

2. Experiments

2.1 Contrast Stretching

Obtain the image **mrt-train.jpg** from the edveNTUre website under Course Documents/ Laboratory Material/Images.

a. Input the image into a MATLAB matrix variable by executing:

```
>> Pc = imread('mrt-train.jpg');
>> whos Pc
```

The **whos** command is to show whether the image is read as an RGB or gray-scale image. Notice that this is a 320x443x3 uint8 matrix indicating a colour image with byte values. You will need to convert this into a grayscale image by:

```
>> P = rgb2gray(Pc);
```

b. View this image using **imshow**. Notice that the image has poor contrast. Your initial task will be to investigate different methods for improving the image appearance using point processing. In point processing, the image transformation is carried for each pixel independently of neighbouring pixels.

c. Check the minimum and maximum intensities present in the image:

```
>> min(P(:)), max(P(:))
```

Contrast stretching involves linearly scaling the gray levels such the smallest intensity present in the image maps to 0, and the largest intensity maps to 255.

d. Next, write *two lines* of MATLAB code to do contrast stretching. Hint: This involves a subtraction operation followed by multiplication operation (note that for images which contain uint8 elements, you will need to use **imadd**, **imsubtract**, etc. for the arithmetic instead of the usual operators; alternatively you can convert to a double-valued matrix first using **double**, but you will need to convert back via **uint8** prior to using **imshow** — see below).

Check to see if your final image P2 has the correct minimum and maximum intensities of 0 and 255 respectively by using the **min** and **max** commands again.

e. Finally, redisplay the image by

>> imshow(P2);

Note again that if you choose to convert your byte images to doubles first, you will need to use the **uint8** command is necessary to convert the P2 double values back to byte values. Otherwise, **imshow** detects P2 to be double and assumes that the intensity range is between 0.0 and 1.0.

An alternative is to use the **imshow** as such:

>> imshow(P2,[]);

This does automatic contrast stretching of the display but does not affect the input matrix. This allows you to ignore whether you are displaying byte or double-valued images.

2.2 Histogram Equalization

You will be using the inbuilt command **histeq** to do histogram equalization. Histogram equalization involves nonlinearly mapping some gray levels to other levels, in order to create a resultant histogram which is approximately uniform.

a. Display the image intensity histogram of P using 10 bins by

>> imhist(P,10);

Next display a histogram with 256 bins. What are the differences?

b. Next, carry out histogram equalization as follows:

>> P3 = histeq(P,255);

Redisplay the histograms for P3 with 10 and 256 bins. Are the histograms equalized? What are the similarities and differences between the latter two histograms?

c. Rerun the histogram equalization on P3. Does the histogram become more uniform? Give suggestions as to why this occurs.

2.3 Linear Spatial Filtering

In linear spatial filtering, typically an input image f(x,y) is convolved with a small spatial filter h(x,y) to create an output image g(x,y),

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} f(x-s, y-t)\, h(s,t) = f(x,y) \otimes h(x,y)$$

The function h(x,y) can be considered to be the impulse response of the filter system. Another common name for h(x,y) is the point-spread function (PSF).

a. Generate the following filters:

$$h(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

(i) Y and X-dimensions are 5 and σ = 1.0
(ii) Y and X-dimensions are 5 and σ = 2.0

Normalize the filters such that the sum of all elements equals to 1.0. View the filters as 3D graphs by using the **mesh** function. These filters are Gaussian averaging filters.

b. Download the image 'ntu-gn.jpg' from edveNTUre and view it. Notice that this image has additive Gaussian noise.

c. Filter the image using the linear filters that you have created above using the **conv2** function, and display the resultant images. How effective are the filters in removing noise? What are the trade-offs between using either of the two filters, or not filtering the image at all?

d. Download the image 'ntu-sp.jpg' from edveNTUre and view it. Notice that this image has additive speckle noise.

e. Repeat step (c) above. Are the filters better at handling Gaussian noise or speckle noise?

2.4 Median Filtering

Median filtering is a special case of order-statistic filtering. For each pixel, the set of intensities of neighbouring pixels (in a neighbourhood of specified size) are ordered. Median filtering involves replacing the target pixel with the median pixel intensity. Repeat steps (b)-(e) in Section 3, except instead of using *h(x,y)* and **conv2**, use the command **medfilt2** with different neighbourhood sizes of 3x3 and 5x5. How does Gaussian filtering compare with median filtering in handling the different types of noise? What are the tradeoffs?

2.5 Suppressing Noise Interference Patterns

Occasionally with poor television reception, parallel lines will be seen on the screen corrupting the desired image. These are interference patterns. Here we explore how bandpass filtering can be used to suppress such interference.

a. Download the image 'pck-int.jpg' from edveNTUre and display it from MATLAB. Notice the dominant diagonal lines destroying the quality of the image.

b. Obtain the Fourier transform F of the image using **fft2**, and subsequently compute the power spectrum S. Note that F should be a matrix of complex values, but S should be a real matrix. Display the power spectrum by

```
>> imagesc(fftshift(S.^0.1));
>> colormap('default');
```

fftshift is used to shift the origin of the Fourier transform to the centre of the image. Note that the origin corresponds to the DC (or zero frequency) component. The power to 0.1 is only used to nonlinearly scale the power spectrum such that it is easier to visualize the frequency components.

Notice that there are two distinct, symmetric frequency peaks that are isolated from the central mass. These frequency components correspond to the interference pattern.

c. Redisplay the power spectrum *without* **fftshift**. Measure the actual locations of the peaks. You can read the coordinates off the *x* and *y* axes, or you can choose to use the **ginput** function.

d. Set to zero the 5x5 neighbourhood elements at locations corresponding to the above peaks *in the Fourier transform F*, not the power spectrum. Recompute the power spectrum and display it as in step (b).

e. Compute the inverse Fourier transform using **ifft2** and display the resultant image. Comment on the result and how this relates to step (c). Can you suggest any way to improve this?

f. Download the image `primate-caged.jpg' from edveNTUre which shows a primate behind a fence. Can you attempt to "free" the primate by filtering out the fence? You are not likely to achieve a clean result but see how well you can do.

2.6 Undoing Perspective Distortion of Planar Surface

The goal in this part of the experiment is to take an original slanted view of a book, and compute the projective transformation to warp the image to a frontal view. The final warped image should have the scale 1 pixel (image) = 1 mm (book dimension).

The following paragraph provides the basic knowledge on 2D planar projective transformation. A 2D planar projective transform is expressed via a 3x3 matrix:

$$\begin{bmatrix} kx_{im} \\ ky_{im} \\ k \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ 1 \end{bmatrix}$$

where X_w and Y_w are the coordinates of a point in the input image, while x_{im} and y_{im} are the coordinates of the transformed point in the output image. This can be alternatively expressed in algebraic form:

$$x_{im} = \frac{m_{11}X_w + m_{12}Y_w + m_{13}}{m_{31}X_w + m_{32}Y_w + 1}, \mathrm{y}_{im} = \frac{m_{21}X_w + m_{22}Y_w + m_{23}}{m_{31}X_w + m_{32}Y_w + 1}.$$

Since there are only 8 degrees of freedom in the projective matrix (i.e., 8 unknowns), we can compute this matrix if we know the correspondence between 4 or more points in the input image and the output image. Using the correspondences and the above equations, the following matrix equation may be set up

$$\begin{bmatrix} X_w^1 & Y_w^1 & 1 & 0 & 0 & 0 & -x_{im}^1 X_w^1 & -x_{im}^1 Y_w^1 \\ 0 & 0 & 0 & X_w^1 & Y_w^1 & 1 & -y_{im}^1 X_w^1 & -y_{im}^1 Y_w^1 \\ X_w^2 & Y_w^2 & 1 & 0 & 0 & 0 & -x_{im}^2 X_w^2 & -x_{im}^2 Y_w^2 \\ 0 & 0 & 0 & X_w^2 & Y_w^2 & 1 & -y_{im}^2 X_w^2 & -y_{im}^2 Y_w^2 \\ . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . \\ X_w^n & Y_w^n & 1 & 0 & 0 & 0 & -x_{im}^n X_w^n & -x_{im}^n Y_w^n \\ 0 & 0 & 0 & X_w^n & Y_w^n & 1 & -y_{im}^n X_w^n & -y_{im}^n Y_w^n \end{bmatrix} \begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{31} \\ m_{32} \end{bmatrix} = \begin{bmatrix} x_{im}^1 \\ y_{im}^1 \\ x_{im}^2 \\ y_{im}^2 \\ . \\ . \\ x_{im}^n \\ y_{im}^n \end{bmatrix}. \quad (*)$$

We can write the above equation in the form of Au = v, where A and v are the 8x8 matrix and 8x1 vector respectively, containing corner data, while u is the 8x1 vector of projective transformation parameters to be computed.

This equation allows the computation of an unknown projective transformation through simple matrix inversion (for 4 correspondences) or pseudo-inversion (for >4 correspondences).

Now please start to go through the following steps.

a. Download `book.jpg' from the edveNTUre website as a matrix P and display the image. The image is a slanted view of a book of A4 dimensions, which is 210 mm x 297 mm.

b. The **ginput** function allows you to interactively click on points in the figure to obtain the image coordinates. Use this to find out the location of 4 corners of the book, remembering the order in which you measure the corners.

```
>> [X Y] = ginput(4)
```

Also, specify the vectors x and y to indicate the four corners of your desired image (based on the A4 dimensions), in the same order as above.

c. Set up the matrices required to estimate the projective transformation based on the equation (*) above.

```
>> u = A \ v;
```

The above computes $u = A^{-1}v$, and you can convert the projective transformation parameters to the normal matrix form by

```
>> U = reshape([u;1], 3, 3)';
```

Write down this matrix. Verify that this is correct by transforming the original coordinates

```
>> w = U*[X'; Y'; ones(1,4)];
>> w = w ./ (ones(3,1) * w(3,:))
```

Does the transformation give you back the 4 corners of the desired image?

d. Warp the image via

```
>> T = maketform('projective', U');
>> P2 = imtransform(P, T, 'XData', [0 210], 'YData', [0 297]);
```

e. Display the image. Is this what you expect? Comment on the quality of the transformation and suggest reasons.

3 REPORT

You are expected to write a report towards the end of the semester for coursework grading. The report should be submitted through edventure. In the report, (1) Please complete all questions; (2) incorporate the processing results; (3) incorporate all source code; and (4) add some comments and analysis if you feel necessary. Because detailed information has been given in this document, the exercises are not very difficult. Please do not copy & paste from your classmates, which can be easily identified.