Digital Image Processing: 525 U0920 / ESOE 5096

Computer Assignment 1

Introduction to Image Display and Manipulation

Due: the week after next, 9:00pm

*Hand in online via **NTU COOL***

Total score: 180

This assignment will introduce you to some of the image display techniques and image processing commands using Matlab. The codes that this course may use include the methods in *Signal Processing* and *Image Processing* related toolboxes. You are free to use any platform to program, but you may opt to use Windows due to its simplicity and availability.

What to turn in:

Matlab source **codes** that perform each individual experiment and the associated **results** and **explanation** if any. Please consolidate all contents into one single file in *pdf* (preferred) or *doc*(*x*). Write down your *name*, *student ID*, and *department* on the top of the file.

Introduction

In the Matlab prompt, type in the command

```
w = imread('wombats.tif');
```

This command takes the gray values of all the pixels in the grayscale image, whose filename is *wombats.tif*, and puts them all into a matrix *w*. We can display this matrix as a grayscale image using

```
figure, imshow(w);
```

where *figure* creates a new figure on the screen and *imshow* displays the matrix *w* as an image. An image matrix *a* can be written to an image file with the *imwrite* function. For example,

```
imwrite(a, gray(256), 'flower.tif', 'TIFF');
```

which write the matrix *a* to a 256 grayscale level TIFF file with the name *flower.tif*.

1. Spatial Resolution (35%)

Read the *lena.bmp* file and use the *imresize* function to make change in spatial resolution. For example, `imresize(x, 1/2);`

will halve the size of the image if **x** is the matrix of the *lena* image. Show me the results of the following commands applying to the *lena* image: (20%)

(a) `imresize(imresize(x, 1/4), 4);`

(b) `imresize(imresize(x, 1/8), 8);`

(c) `imresize(imresize(x, 1/16), 16);`

(d) `imresize(imresize(x, 1/32), 32);`

Will you obtain the same result? State the effective resolution of each command and explain why.

Now, use different interpolation methods below to resize the image to 1/8 of the original dimension and show me the results: (15%)

(e) *nearest*

(f) *bilinear*

(g) *bicubic*.

2. Bit Planes (35%)

Read the *cameraman.tif* file, denoted as **x**, and make it a matrix of type *double*, denoted as **xd**, which means we can perform arithmetic on the values. We now isolate the bit planes by simply dividing the matrix by successive powers of 2, throwing away the remainder, and seeing if the final bit is 0 or 1. We do this with the *mod* function:

```
c0 = mod(xd, 2);
c1 = mod(floor(xd/2), 2);
c2 = mod(floor(xd/4), 2);
…
c7 = mod(floor(xd/128), 2);
```

where c0 is called the least significant bit plane and c7 the most significant bit plane. Show me the 8 bit planes of the *cameraman* image. (16%)

The most significant bit plane c7 is actually a binary image that maps original intensities between 0 and 127 to 0 and maps all levels between 128 and 255 to 1. We can create this binary image by thresholding the image at level 127 using

```
ct = x > 127;
```

Are they (c7 and ct) identical? You can verify your answer by using

```
all(c7(:) == ct(:));
```

What will you get by executing this command? (4%)

We can recover the original image using:

```
xc = 2*(2*(2*(2*(2*(2*(2*c7+c6)+c5)+c4)+c3)+c2)+c1)+c0;
```

Now, reconstruct the image using the following bit planes:

(a) c7

(b) c6 and c7

(c) c4 to c7

Show me the reconstructed images and display the difference images between the reconstructed images and the original image. What do you observe? (15%)

3. Histogram Operation (30%)

Read the *pollen.tif* file and show me the image associated with its histogram using the *imhist* function.

(a) Apply the histogram equalization to this image using the *histeq* function and show me the result associated with its equalized histogram. (5%)

(b) Perform histogram stretching to this image using the *imadjust* function and show me the result associated with its adjusted histogram. You will need to manually select the parameters in *imadjust* in order to make the adjusted image more appealing. Explain why you choose such parameters and the corresponding transform function for this image. (10%)

Repeat all processes for *aerial.tif.*

4. Transformations and Registration (80%)

The affine transform is a fundamental geometric transformation tool that can scale, rotate, translate, or shear an image depending on the element of the affine matrix. Read the *peppers.bmp* image and denote it as **f**. We now apply geometric transformations to this image via the affine transform using

```
sx = 0.85;
sy = 1.15;
T1 = [sx  0   0
      0   sy  0
      0   0   1];
t1 = affine2d(T1);
fs = imwarp(f, t1);
```

and

```
theta = pi/6;
T2 = [cos(theta)     sin(theta)        0
     -sin(theta)        cos(theta)        0
      0              0              1];
t2 = affine2d(T2);
```

```
fsr = imwarp(fs, t2);
```

and

```
alpha = 0.75;
T3 = [1          0      0
     alpha       1      0
      0          0      1];
t3 = affine2d(T3);
fss = imwarp(fs, t3);
```

Obviously, **fss** and **fsr** are created based on **fs**. Display the transformed images, **fs**, **fsr**, and **fss** associated with the original image **f**. (8%)

Image registration is an important application of digital image processing. One simple method is to use tie points as described in lecture. You can use some Matlab tools to demonstrate the registration of the transformed images. For example, you can first use *cpselect* to select the matching points as

```
cpselect(fs, f)
```

This means that you are going to register the target image **fs** to the source image **f**. You can pick 4 pairs for the registration. After you complete the selections, you can export points to workspace and close *cpselect*. Now, your workspace should have two new variables called *base_points* and *input_points*, respectively. You can save these variables to files for further use as

```
save('base_points.mat', 'base_points');
save('input_points.mat', 'input_points');
```

Next, compute the transform that makes **fs** fits **f** using

```
tietform = fitgeotrans(input_points, base_points, 'affine');
```

Finally, execute the transformation (registration) the image **fs** using:

```
fs2 = imwarp(fs, tietform);
```

The output image **fs2** is the registered image. Show the registered image **fs2**, the target image **fs**, the source image **f**, and the difference image **fs2 - f**. What do you observe? Increase the number of tie points using 8 pairs. Do the results look better?
Repeat the process for the target images **fsr** and **fss**. (12%×2×3)

Hint: Use Matlab *help* to figure out how to use the functions that you are not familiar with, e.g., **help imhist**. You may also use the *subplot* function to well organize the results for each experiment.