Image Processing with MATLAB

1. Aims and Objectives:

1.1 Introduction

MATLAB is a very powerful, high level language; It is also very easy to use. It comes with a wealth of libraries and toolboxes, that you can use directly, so that you don't need to program low level functions. It enables us to display very easily results on graphs and images. To get started with it, you need to understand how to manipulate and represent data, how to find information about the available functions and how to create scripts and functions to generate programs.

You have already had an introduction to MATLAB in Semester 1 in the course Signal Processing with MATLAB. This course seeks to build on that experience by exposing you to using MATLAB to manipulate and process Images.

The aim of the course is to build on the semester 1 course on Image Processing and to apply the methods learned there to real images.

1.2 Organisation and Assessment

The course is organised into 10 **compulsory** laboratories. The laboratories will take place from 9am-12noon every Wednesday in weeks 1-10 of semester 2. In addition, a 2hour lab slot is booked from 10-12noon for use by students although this slot will be unsupervised. If you cannot attend the Wednesday laboratory for any reason them you must inform Prof McLaughlin by email as soon as possible.

These notes are grouped into 5 sessions that students will be expected to work through the exercises described in these notes at their own pace. The sessions are as follows:

Session 1:	MATLAB Revision, manipulation of images (Image Inversion and Fourier Transform)
Session 2:	Fourier Analysis of Images
Session 3:	Histogram Techniques Applied to Image Analysis
Session 4:	Image Filtering to remove noise
Session 5:	Hough Transform

Students will be required to keep a laboratory notebook which will be submitted at the end of the semester. In each of the sessions described in these notes there are a series of exercises that students are asked to work through. Students will record all comments and observations on these exercises in their laboratory notebooks in sufficient detail that someone reading the notebook could reproduce their results.

The day-book is a diary, a journal, a day by day, hour by hour, record of the work which you do in the pursuit of the objectives of the practical work. In it, you should make an accurate and comprehensive record of your activities, the resources used, the results

obtained and such notes on interpretation, planning and other organisation of the work, that another person, conversant with the area of your work, could reconstruct your activities, in full.

Thus, the laboratory notebook records failure as well as success.

* It records good ideas and bad ideas.
* It records leads which produce the big breakthrough, as well as those which do not.
* It records decisions to do things and decisions to not do things.
* It records things which are undecided.
* It records how tests were set up.
* It records raw results, as tables, technical sketches, graphs, &c.
* It records references to sources of ideas and information.

It is a chronological record; one made at the time and place of the activity.
It is not a formal report, neatly written up after the event, edited to produce a consistent and logical flow of ideas and actions. It is life in the raw. It contains working notes, made as thoughts and plans are conceived, modified in real time as the ideas develop. It is a book of rough working. Nonetheless, the records should be legible.

The records in a laboratory notebook must be dated and the pages are numbered.
By following your laboratory notebook, another person should be able to reconstruct (literally, if necessary) all the work which you have performed, in the order in which you performed it. You should expect your day-book to be inspected and commented on through the course of the semester by the lab supervisor or laboratory demonstrator. When this happens, they will sign and date the book at the latest record.

Assessment will be based on the following breakdown of marks.

10% for attendance.
30% for the quality of record keeping in the laboratory notebook based on the criteria discussed above.
4 x 15% that is 15% for each one of four coursework submissions spread throughout the semester.

Note: All work must be the students own work and plagiarisms and copying will be subject to penalty.

The timetable for submitted assessments will be as follows:

Week 4: Assessed Coursework exercise on Fourier Analysis of images
Week 8: Assessed Coursework on Histogram Techniques
Week 6: Assessed Coursework on Image Filtering
Week 10: Assessed Coursework on Hough Transform

For each of these assessments the students will submit a 1 page description of the problem and their approach to the solution. Flow charts where appropriate illustrating how they have structured their solution. Programmes used to solve the problem.
All solutions must be submitted via WebCT by the 5pm on the Friday of the weeks in which an assignment is due.

Session 1

2. What is MATLAB?
MATLAB is a high performance language and software for technical computing. It allows

- Maths and computation
- Algorithm development
- Modelling, simulation and prototyping
- Data analysis and visualisation
- Scientific and engineering graphics

It is an interactive system (all data always in memory) with the following key facts:

- The basic data element is an array
- No declaration (or prototyping of functions) required
- All data represented as doubles
- In script mode or command line mode, data is always accessible and in memory.

3, Basic Commands and Getting Started

3.1 Knowing where you are:

Use the `pwd` command to know where you are
Use the `dir` command to list the files in your current directory
Use the `cd` command to change directory
Now go to your home directory to be able to save today's work

3.2 The help command:

You can get help on all commands in matlab by typing (beware, matlab is case sensitive!):

```
>> help command
```

For instance, try typing
```
>> help help
```
And read what pops up in the screen…That should give you plenty of information on how to
find information on topics. For instance, if you are looking for the cosine function, type
```
>> help cos
```
Note that if you press the "up" and "down" arrows on your keyboard, this enables you to recall previous commands.

3.3 The lookfor command

If you do not know what is the name of the matlab command you want to use the `lookfor` command is what you are looking for.

For instance, if you are looking for the cosine or related functions, type
```
>> lookfor cosine
```

matlab comes back with all matlab functions related to cosine with a short description as:

```
ACOS Inverse cosine.
ACOSH Inverse hyperbolic cosine.
COS Cosine.
COSH Hyperbolic cosine.
TFFUNC time and frequency domain versions of a cosine modulated
Gaussian pulse.
DCT2 Compute 2-D discrete cosine transform.
DCTMTX Compute discrete cosine transform matrix.
DCTMTX2 Discrete cosine transform matrix.
IDCT2 Compute 2-D inverse discrete cosine transform.
CHIRP Swept-frequency cosine generator.
DCT Discrete cosine transform.
FIRRCOS Raised Cosine FIR Filter design.
IDCT Inverse discrete cosine transform.
DCT Discrete cosine transform.
IDCT Inverse discrete cosine transform.
dctold.m: %DCT Discrete cosine transform.
idctold.m: %IDCT Inverse discrete cosine transform.
BLKACOS This block defines an output angle that is the arccosine of
the input.
BLKCOS This block defines the output as the cosine of the input.
BLKCOSASIN This block defines the cosine of an angle whose sine is u.
BLKCOSATAN This block defines the cosine of an angle whose tangent is
u1/u2.
```

Now type help cos to get details on the cosine function.

```
>> help cos
```

Note that if you press the "up" and "down" arrows on your keyboard, this enables you to recall previous commands.

4. Creating a vector or a matrix:

As mentioned before, matrices (and vectors) are the basic element of a matlab program. matlab does see matrices as other languages see numbers and can perform operations in matrices directly without for loops, which comes handy for image processing...
To find out more about vectors and matrices try and type the following and observe the results.

Type :
```
>> x = [1 2 3 4 5 4 3 2 1];
```

This will create a vector of size 1 by 9 called `x` which contains the numbers specified above

```
>> x
```

This will display the contents of `x` on the screen

```
>> who
```

This will tell you about all the matrices/vectors you have in your current workspace.

```
>> whos
```

This will provide more details about these matrices/vectors.

```
>> y = [6; 7; 8; 9; 0; 9; 8; 7; 6]
```

Create a new matrix called `y` of size 9 by 1. *(note that is there is a semicolon at the end of the instruction, matlab doesn't expand the result of the instruction)*

```
>> y'
```

Take the transpose of y

```
>> z = [1 2 3; 4 5 6; 7 8 9; 0 1 2]
```

Create a new matrix called `z`

5. Operators:

NOTE: MOST OF THEM WORK ON MATRICES DIRECTLY.

5.1 The colon operator:

A way to define quickly some vectors is to use the ":" operator. Type for instance (and observe the results):

```
>> a = 0:10
>> b = -5:10
>> c = 0:2:10
>> d = 10:-2:-5
>> e = 0:0.01:4.2
>> f = -pi:0.01:pi
```

You can see that these instructions are of the form *lower limit: increment: upper limit*. If no increment is given, it is assumed the increment is 1.
If you wish to define elementary vectors that only contain ones and zeros, then the commands

`zeros` and `ones` are useful. Also `randn` to create a matrix of random numbers. Look at the help on *`zeros`*, *`ones`*, *`randn`*. To illustrate `ones` and `zeros` try:

Try:

```
>> g = zeros(2,4)
>> h = ones(5,3)
```

5.2 Arithmetic operators:

These are the standard operators. (`+` plus, `-` minus, `/` divide, `*` multiply, `^` exponent) Try

```
>> 3^2
>> y'
>> x+y'
>> x*y
>> 3*x
```

The operators (/ , * and ^) have also got a matrix counterpart (respectively `./` , `.*` and `.^`), which apply the operation element by element. For instance, compare: `x*y` and `x.*y`. Type:

```
>> x^2
>> x.^2
>> x.*x
>> x*x
```

if you type "`whos`" now, you will see that you have a lot of variables in the workspace.

Type

`clear`, and `whos` again and you will see that your workspace has been cleared.

5.3 Matrices and operators:

Type

```
>> A = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]
>> sum(A) % displays the sum of the columns of A.
```

This will display on the screen

```
>> 34 34 34 34
```

To investigate further

```
>> sum(sum(A)) % displays the sum of A
>> A' %Transpose conjugate (if complex number) else transpose.
>>sum(A') % Sum of the lines of A (columns of A'). Note that functions can be
```

combined very easily.

Type as given below to find out what is in element (4,2)

```
>> A(4,2)
```

However if you try

```
>> A(4,5)
```

You will get “ERROR: index exceeds matrix dimension!”

7. The functions available and the toolboxes

Type:

```
>> help elfun
```
% elementary functions
```
>> help specfun
```
% special functions
```
>> help elmat
```
% elementary math functions to see some of what's there. Try out some of the functions.

For instance, type:

```
>> t =-pi:0.01:pi;
>> c = cos(t);
>> s = sin(t);
```

Important toolboxes are available:

Image processing toolbox - for more information `>> help images`
Signal processing toolbox - for more information `>> help signal`
Statistics toolbox - for more information `>> help stats`

6. Plotting functions:

Type

```
>> figure
>> plot(s)
>> figure
>> plot(t,s)
>> figure
>> plot(t,s,'g',t,c,'r')
```

Type `help plot` if you haven't already done it to see more possibilities.
Try and modify the axis, put a title and labels on the x- and y- axes…
Add a title and a label on the x-axis.

7. Image processing:

Try

```
im1 = imread('peppers.png'); % Reads an image and puts it in im1
im2 = randn(384,512); % Creates a 384x512 normally
% distributed image (gaussian noise)
% of the size of im1.
whos % Note that images are stored in uint8 to
% save memory space
```

Try

```
imshow(im1); % Display matrices as images
imshow(im2);
```

Try

```
figure(1)
imshow(im1); % Display matrices as images
figure(2)
imshow(im2); % Same thing with 2 windows.
```

Play with the figure menus and especially the export button that allows saving images and figures. Type help figure for more information.

Try

```
subplot(2,1,1)
imshow(im1); % Display matrices as images
subplot(2,1,2)
imshow(im2); % Same thing with 2 images in one
% window.
```

Try

```
clf; % Clear figure
imshow(im1(:,:,1)) % Displays Red component of the image
```

Try

```
clf;
subplot(3,1,1)
imshow(im1(:,:,1)) % Displays Red component of the image
title('Red Component');
subplot(3,1,2)
imshow(im1(:,:,2)) % Displays Green component of image
title('Green Component');
subplot(3,1,3)
imshow(im1(:,:,3)) % Displays Blue component of the image
title('Blue Component');
```

BE CAREFUL, OPERATORS DO NOT WORK ON UINT8 TYPE

Try

```
im1 = double(rgb2gray(im1)); % Converts image to grey level image and to
% double type.
```

7.1 Operators and images:
Addition:

Try

```
ima = im1 + im2;
imshow(ima);
```

Substraction:

Try
```
ima = im1 - im2;
imshow(ima);
```

Multiplication:

Try
```
ima = im1*im2;
```
This is a matrix multiplication which cannot be applied to multiply images.
The correct way to do this is:
```
ima = im1 .* im2;
imshow(ima);
```

Other operators

/ Matrix division
^ Power operator
‘ Complex transpose

The combination of . and any other operator makes it an element to element operator. For instance .* or ./ multiplies and divides each element of the first image by the element of the second.

8. Workspace and saving results:

As matlab is interpreted, the data are always in the workspace and can be saved. The workspace can be seen as an area of memory accessible from the command line.

To see it: `whos`
To clear it: `clear`
To save it: `save name` Do help save to learn more about save.

To load it `load name` Do help load to learn more about load.

YOU CANNOT SAVE ANYWHERE. YOU NEED TO HAVE WRITE PERMISSION.

Session 2

1. Matlab resources:

During this session, you will learn:

1- Basic flow control and programming language
2- How to write scripts (main functions) with matlab
3- How to write functions with matlab
4- How to use the debugger
5- How to use the graphical interface
6- Examples of useful scripts and functions for image processing

After this you will know most of what you need to know about Matlab and should definitely know how to go on learning about it on your own. So the "programming" aspect won't be an issue any more, and you will be able to use matlab as a tool to help you with you image processing. Programming languages don't come any easier!

2. Matlab resources:

- Language: High level matrix/vector language with
  - Scripts and main programs
  - Functions
  - Flow statements (for, while)
  - Control statements (if,else)
  - data structures (struct, cells)
  - input/ouputs (read,write,save)
  - object oriented programming.
- Environment
  - Command window. You are now familiar with it
  - Editor
  - Debugger
  - Profiler (evaluate performances)
- Mathematical libraries
  - Vast collection of functions
- API
  - Call C functions from matlab
  - Call Matlab functions form C programs
- **GUI tool**
  - Allows to design easily Graphical User Interfaces to your programs.

3. Scripts and main programs

In matlab, scripts are the equivalent of main programs. The variables declared in a script are visible in the workspace and they can be saved. Scripts can therefore take a lot of memory if you are not careful, especially when dealing with images. To create a script, you will need to start the editor, write your code and run it.

4. How to use the editor

Just type `edit` in the command line and the editor starts.

This should start the editor with a new file. Write the following in the file and save the file as `readimage.m` in your local directory. To go to your local directory, use the `cd` command as seen last week.

```
% This is my first script in matlab
% This script reads in an image, displays it and changes the
colormap

% First read an image from matlab images
ima = imread('moon.tif');
% Now display it
clf;              % Close figure 1
figure(1);        % First create a figure
image(ima) ;      % Display but do not resize or rescale.
                  % Useful only of image is in [0,256]
disp('Press a key to continue');
pause;            % Waits until a key is pressed to continue
imagesc(ima);     % Display and rescale the image.
                  % Useful for floating point images.
colormap(gray);   % Uses the gray colormap
pause(5);         % Wait 5 seconds and display the image
imshow(ima);      % Normal matlab function from image processing
                  % toolbox
% Now write the image to disk
disp('Now writing image in jpeg format');
imwrite(ima, 'moon.jpg','jpg');  % Write the image in JPEG format

% various formats are supported (see help imwrite)

%CAREFUL IMSHOW ONLY WORKS WITH UINT8 IMAGES
```

5. Scripts and Functions:

You've already written your first elementary script. Standard branching and looping instructions can be used as well: "for" loops, "if" statements etc… To find out about their syntax, type "help" for these different instructions.

5.1 if statement:

the if statement is written as follows:

```
if (condition1)
```

```
     expression1
elseif (condition2)
     expression2
else
     expression3
end
```

5.2 Logical operators:

`A==B`	**equality for scalars**
`isequal(A,B)`	**equality for vectors, matrices, structures**
`A~=B`	**difference**
`A>B`	**superior. Will return an array of boolean for arrays with 1 for elements where the condition is satisfied and 0 elsewhere.**
`A<B`	**inferior. Will return an array of boolean for arrays with 1 for elements where the condition is satisfied and 0 elsewhere.**
`A>=B`	
`A<=B`	
`isempty(A)`	**Check if the variable exists and is not empty.**
`isinf(A)`	**Returns 1 if A is Inf (infinity), 0 otherwise.**
`isnan(A)`	**Returns 1 if A is not a number (NaN), O otherwise**[1]**.**

5.3 Switch and case:

```
switch (expression)
     case value1
          action1;
     case value2
          action2;
     otherwise
          default action;
     end
```

5.4 For loops

```
for index = 1 : m
     r(index) = index;
end
```

Example:

[1] Note that NaN and Inf are defined. Example A = Inf

Try

```
ima = zeros(100,100);
for i = 1:100
     for j = 1:100
          ima(i,j) = i+j;
     end
end
imagesc(ima)    % imshow is buggy and sometimes does
not                    % work properly. In that case use
imshow
                       % then imagesc or imagesc  directly.
```

5.5 while loops

```
while (expression)
     Action
end
```

5.6 Functions

Functions differ from scripts in that they take explicit input and output arguments. Type "help" function to know more and see examples. As all other matlab commands, a function can be called within a script of from the command line.

The syntax of a function is as follows:

function **[out1,out2,...,outn] =**
function_name**(in1,in2,...,inn)**

Tips:

nargin: **gives the number f input arguments**
allow defaults parameters
allow variable number of arguments

% This is the symbol for comments

As an example, here is a basic function that reads in an image, displays it and finds the min and max of the image. **The function must be saved into a file of the same name as the function to be accessible from the command line.** In our case we save it as `min_max.m`.

```
function [mini,maxi] = min_max(imageName)
     ima = imread(imageName);   % reads in an image of name
                                % imageName
     imshow(ima);
     ima = rgb2gray(ima);       % Converts image to grey image
     imshow(ima);               % Display the image
     mini = min(min(ima));      % Get the min of the image
```

```
maxi = max(max(ima));      % Get the max of the image
```

Nothing else is required!

Try

```
>>[mini,maxi] = min_max('peppers.png');
```

Try to modify this function to get the maximum and minimum of each color

6. Debugging a program

Debugging is easy in matlab as it is an interpreted language. In order to start the debug mode, you just have to type in the command line:

```
>>dbstop if error              % Will stop for errors
>>dbstop if warning            % Will stop for warnings
```

Alternatively, you can go into the editor and set the options and breakpoints where you want. In any case, the editor will be invoked and show the line where the error has occurred.
In this case the `K` symbol appears in the command line.
As matlab is interpreted you can change the values of variables and continue the execution using:

```
K>>dbstep           % continues by one line
K>>dbcont           % continues until next stop
```

To see all the options of the debugger, use `help dbstop`.

To quit the debugger, use `dbquit`
To remove breakpoints and debugger options, use `dbclear`

Finally you can use the keyboard functions. This does not start the debugger but stops the execution where the keyboard has been typed. You can see the variables and resume execution typing dbcont.

7. Function/Command Duality

Command line	**In function or script**
`load August17.dat`	`load('August17.dat')`
`save January25.dat`	`save('January25.dat');`
`cd H:\matlab\`	`cd('H:\matlab')`

8. General Tips

8.1 Vectorization:

```
x = 0;
for k = 1 : 1001
   y(k) = log10(x);
   x = x +0.01;
end
plot(x,y);
```

```
x = 0:0.01:10;
y = log10(x);
```

10 times faster!

8.2Preallocation:

```
r = zeros(32,1);
for n = 1 : 32
   r(n) = n;
 end
```

5 times faster!

8.3 Matrices

`zeros(m,n)`	Creates a mxn matrix filled with zeros
`ones(m,n)`	Creates a mxn matrix filled with ones
`rand(m,n)`	Creates a mxn matrix following a uniform distribution in [0,1]
`randn(m,n)`	Creates a mxn matrix following a normal Gaussian distribution (mu = 0, sigma = 1)

9. Full exercises to attempt:

Exercise 1: Image inversion:

Example program that reads in an image and invert it.

Exercise 2: Fourier Transform:

Example program that takes the Fourier transform and display it.
Same for the inverse Fourier Transform.

These programs must be downloaded from the Image Processing module on vision:

http://www.see.ed.ac.uk/~sml/Image_Processing/

Please stick this page into your bookmarks as it will be useful for the remainder of the course.

Please download them and try them out.

9.1 Image inversion

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Image inversion
%
%       ima_out = invert(ima)
%
% This function inverts the input image
%
%       ima : real input image
%       ima_out : output real
%
%
% Yvan Petillot, December 2000
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function ima_out = invert(ima)

  % First convert ima to double
  ima = double(ima);
  % Checks that ima is a gray level image
  if ndims(ima) > 2
    % RGB image?
    if ndims(ima) == 3
      ima = double(rgb2gray(uint8(ima));
    else
      display('Unknown format, Cannot guarantee result');
    end
  end
  % Create new figure
  figure(1)
```

```
  % Display ima
  imagesc(ima);
  colormap(gray);
  axis(image);
  % Rescales ima between [0, 255]
  mini = min(min(ima));
  maxi = max(max(ima));
  ima_out = (ima-mini)/(maxi-mini)*255;
  % Invert ima_out
  ima_out = 255-ima_out;
  % Display result
  % Create new figure
  figure(2)
  imagesc(ima_out);
  colormap(gray);
  axis image;
  axis off;
```

Try using this function with various images.

```
>> ima = imread('kids.tif');
>> ima_out = invert(ima);
```

Note in this example, the use of ndims to find out the number of dimensions of the input array.

Additional Exercise:

Could you alter this script to invert each colour and generate a inverted RGB image? Try it in your spare time...

9.2 Fourier Transform

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% 2 Dimensional Fourier Transform
%
%       [imafft,mag,phi] = fft2d(ima_in)
%
% This function calculates the complex Fourier transform
% and the associated magnitude and phase (mag, phi) of the input real
% image ima_in.
%
%       ima_in : real input image
%       imafft : output complex image
%                real and imaginary part can be accessed through real
%                and imag function (see help real, imag)
%
%       mag,phi: magnitude and phase of the fourier transform
%
% Yvan Petillot, December 2000
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [imafft,mag,phi] = fft2d(ima_in)
```

```
imafft = fftshift(fftn(single((fftshift(ima_in))));
mag = abs(imafft);
phi = angle(imafft);
%Display results
clf;       %Clear display
figure(1); %Create figure
title('Demo of the fourier transform')
subplot(1,3,1);  %Separate figure in 3 and selects 1st partition
imagesc(ima_in); % Scale image to [0 255] range and display it
title('Original image');
axis image;     % Display equal on x and y axes
colormap(gray);  % Sets gray scale colormap;
axis off;
subplot(1,3,2);  % Selects the second partition
imagesc(log(mag+1));    % Display magnitude
title('Magnitude of the spectrum');
axis image;     % Display equal on x and y axes
axis off;
subplot(1,3,3);  % Selects the second partition
imagesc(phi);    % Display phase
axis image;     % Display equal on x and y axes
title('Phase of the spectrum');
axis off;
drawnow;         % Ready to dipslay: Do it
```

Try using this function with various images.

```
>> ima = imread('peppers.png');
>> ima = rgb2gray(ima);
>> ima_out = fft2d(ima);
```

Note that `ima_out` is an array of complex numbers stored as real and imaginary parts. `real(ima_out)` returns the real part while `imag(ima_out)` returns the imaginary part. Magnitude and phase can be obtained from the real and imaginary part using the `abs` and `angle` functions. This functions, as always in matlab work directly on arrays.

Additional Exercise:
What would happen if the image was rotated?
Try it using the `imrotate` function to rotate the image with and without the crop parameter (see `help imrotate`) on the 'circuit.tif' image.
Try

```
>> ima = imread('circuit.tif');
>> ima1 = fft2d(ima);
>> ima2 = fft2d(imrotate(ima,30));
```

9.2.1 Inverse Fourier Transform

This function will perform the inverse Fourier Transform. First try to take the Fourier Transform and then the inverse Fourier Transform to return to the original image.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% 2 Dimensional Inverse Fourier Transform
%
%       [ima] = ifft2d(ima_in)
%
% This function calculates the complex Inverse Fourier transform
%  of the complex image ima_in.
%
%       ima_in : complex input image
%       ima : output (possibly complex) image
%             real and imaginary part can be accessed through real
%             and imag function (see help real, imag)
%
%
% Yvan Petillot, December 2000
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function ima = ifft2d(ima_in)

  ima = fftshift(ifft2(fftshift(ima_in)));
  %Display results
  clf;       %Clear display
  figure(1); %Create figure
  imagesc(abs(ima)); % Scale image to [0 255] range and display it
  axis square;     % Display equal on x and y axes
  colormap(gray);  % Sets gray scale colormap;
  drawnow;         % Ready to dipslay: Do it
```

9.2.2 Generic Fourier Transform Function

This is simply to illustrate how matlab functions can be called from other functions. This function can perform either the Fourier Transform or the inverse, depending on the setting of the direction argument.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% 2 Dimensional generic Fourier Transform
%
%       [ima_out] = fft2(ima_in,direction)
%
% This function calculates the complex Fourier transform
% of the input image ima_in. If direction is set to 1, the direct fft
% is performed, otherwise, the inverse one performed.
%
%       ima_in : input image. Can be real or imaginary
%       direction: optional parameter. By default, direct Fourier
transform is assumed
%       imafft : output image
%
%
```

```
% Yvan Petillot, December 2001
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [ima_out] = fft(ima_in,direction)

if nargin == 1
   direction = 1;
end

if (direction == 1)
   [ima_out,mag,phi] = fft2d(ima_in);
else
   ima_out = ifft2d(ima_in);
end
```

This is essentially a demo of the nargin parameter

Session 2: Fourier Analysis and Manipulations

1. Objectives:

We will investigate the notion of spectrum and simple filtering in the frequency domain. As we will see, in all cases, we can interpret the effect of these filters in the frequency domain easily while their effect is not obvious in the time domain.

During this session, you will learn & practice:

1- More programming in matlab
2- More examples of functions and scripts
3- Fourier transform and its applications
4- Simple Filtering techniques
5- Phase information and its relation to image structure

2. Resources required:

In order to carry out this session, you will need to download images and matlab files from the Image Processing module on vision.

http://www.see.ed.ac.uk/~sml/Image_Processing/

Please download the following images:

- Image of Lena
- Landsat image
- sonar image
- sar image

We will also use the circuit and flowers images (circuit.tif, peppers.png).
You are of course encouraged to try these programs out on images of your choice.
Please download the following functions and script:

- `fft2d.m` (same as last week)
- `ifft2d.m` (same as last week).
- `phase_only.m`
- `random_phase.m`
- `random_magnitude.m`
- `quant_fft.m`
- `move_image.m`
- `SimpleFiltering.m`

Session 3: Histogram Manipulation

1. Objectives:

We will investigate the properties of the histogram and its various uses for image preprocessing, visualisation and segmentation. During this session, you will learn & practice:

1- More programming in matlab
2- More examples of functions and scripts
3- Histogram manipulation in matlab
4- Simple segmentation techniques

2. Resources required:

In order to carry out this session, you will need to download images and matlab files from the Image Processing module on vision:

http://www.see.ed.ac.uk/~sml/Image_Processing/

Please download the following images:

- Image of Lena
- Landsat image

We can also use other images – for example circuit image. (circuit.tif). Note that a full list of all the images supplied as part of matlab can be accessed through `help images/imdemos`.

You are of course encouraged to try these programs out on images of your choice.

3. Histogram manipulation:

As already seen in class, the histogram of an image is a simple way to see how the gray level (i.e. the information) is spread over the available range of grey levels. This in turn can be used to highlight problems during the acquisition (clipping due to over exposure, low contrast). The histogram can also be modified to make better use of the available range for further processing and display.
There is another way to see the histogram in the light of probability theory. An image can be seen as a random field (array) of pixels, each pixel being a random variable. Assuming these pixels follow the same statistical law, then the histogram can be interpreted as an estimator of the probability density function (pdf) of this law. This fact has been used for texture classification using local histograms for example (here, the texture are modelled locally by the histogram).

Matlab has several functions available to calculate and process histograms:

• `imhist`: Display the histogram of an image.
Synthax: `hist = imhist(ima,nb_boxes)`.
`imhist` works with uint8 types images.
• `histeq`: histogram equalisation and specification.
Syntax: `ima1 = histeq(ima,hgram)` where `hgram` is a specified histogram. If `hgram` is not present, histogram equalisation is performed.

To Try:-

Please load an image (for example landsat.tif).
Display the image in a figure;

Now use the `imhist` and `histeq` functions

Comments?

Use the `histeq` function to specify histograms for various images and compare the result with the desired histogram.

Can you explain the difference?

Can you improve the results?
What is the influence of the number of boxes?

Now have a look at the `imadjust` function and see how it is related to `histeq`.

4. Matlab Demos

Matlab has a number of demos for Image Enhancement. Investigate the use of Histogram Equalisation and Simple transformation functions using the demo **"Intensity Adjustment and Histogram Equalisation".** (At the prompt type `demos` and select the Image Processing Toolbox) Also investigate the demo **"Contrast Enhancement Techniques**". Work through the steps provided to see the use of the `imadjust`, `histeq` and `adapthisteq` functions.

Session 4: Filtering

1. Objectives:
We will demonstrate various aspects of filtering, including spatial and frequency domains and we will analyse the frequency response of various filters and their interpretation.

During this session, you will learn & practice:

1- Filter synthesis in space and frequency domain
2- Filter response analysis
3- Various examples of commonly used filters
4- Noise removal using various filters

2. Resources required:

In order to carry out this session, you will need to download images and matlab files from

http://www.see.ed.ac.uk/~sml/Image_Processing/

Please download the following images:

- Image of Lena
- Landsat image

We will also use the circuit and flowers images (circuit.tif, flowers.tif).
You are of course encouraged to try these programs out on images of your choice.
Please download the following functions and script:

- `SimpleFiltering.m`
- `gene_filters.m`
- `Analyse_Filter.m`
- `Deconvolution.m`

3. Simple Filtering:

The simplest form of filtering are high and low pass filters applied directly in the frequency domain. This is illustrated in the `SimpleFiltering` program as you might have time to see in tutorial 2.
It is interesting to note that a finite frequency response (as it is the case for ideal low pass and band-pass filtering) leads to an infinite space signal (image) and that these filters are in general not realizable as they are not causal.

The `SimpleFiltering()` function applies either a low pass of high pass filter to the image with the specified cut-off frequency. The syntax for the function is

```
ima_out = SimpleFiltering(ima,type,cutoff)
```

where

`ima`: is the input image
`type`: (is either 0 or 1), if specified as 0 will apply a high pass filter, and if 1 will apply a low pass
`cutoff`: cutoff frequency in the range [0,1]. 0 is corresponds to the minimum frequency and 1 the maximum frequency.
`ima_out`: the output real It will produce 3 figures, the first showing the original image, the 2nd the filtered frequency response and the 3rd the filtered image.

For example, to perform a high pass filter, try:-

```
ima = imread('lena.tif');
ima_out= SimpleFiltering(ima, 0, 0.5);
```

to perform a low pass filter

```
ima = imread('lena.tif');
ima_out= SimpleFiltering(ima, 1, 0.5);
```

repeat the above altering the cut-off frequency, and noting the effect.
This type of filtering is more of the type *feature extraction* as we define the response of the filter in function of the type of characteristics of the image we want to extract. More efficient filters (both in terms of response and computation time) have been proposed over the years to perform these type of operations.

4. Spatial domain filtering

Spatial domain filters are generally defined as small windows (or images) that are convolved with the original image (see your notes on filtering). In the spatial domain, the action of a filter can be seen by looking at the structure of the filter. For instance a mean filter will just replace the value of the current pixel by the mean of its neighbour. This is typically a low pass filtering operation as we will show shortly. Conversely, a Prewitt or Sobel filter acts as a derivative and tends to locate edges. This is typically a high-pass / band-pass filtering operation.

To examine the use of such spatial domain filters two functions are required. First run `gene_filters` this will create a set of masks for a range of filters as small matrices labelled as given below. The second function `Analyse_Filter()` will then apply the specified filter to the image.

4.1 Low pass filtering:

Using the `gene_filters` function, you will have generated the following filters as matrices as labelled below (type `whos` to see the list of active matrices):

`fmean3` [3x3] mean filter

`fmean5` [5x5] mean filter
`fmean11` [11x11] mean filter
`fgaussian5_1` [5x5] Gaussian filter with standard deviation 1. See `help fspecial.`
`fgaussian7_05` [7x7] Gaussian filter with standard deviation 0.5.
`fgaussian7_3` [7x7] Gaussian filter with standard deviation 3.

These filters are all low pass filters. The interest of the Gaussian filter lies in the fact that the Fourier transform of a Gaussian is a Gaussian and therefore the frequency response of the filter is very smooth with no virtually no sidelobes.

Use the function `Analyse_Filter(image,filter)` to see the effect of each filter On the image and their associated frequency response. This will only work on grey level images but could be easily extended to colour images.

For example, to examine the fmean3 filter,

```
>> gene_filters
>> ima = imread('circuit.tif);
>> ima2 = Analyse_Filter(ima, fmean3);
```

`Analyse_Filter()` will display the original image, the frequency response of the image and the resulting filtered image. It will also plot the frequency response of the filter as a 3D plot in a separate window.
You should now repeat the procedure above the other filters. (Note: you don't have to repeat the `gene_filters` and `imread` – only the `Analyse_Filter` stage)

4.2 High / Band-Pass pass filtering and edge detection:

Using the `gene_filters` function, you will also have generated the following filters:

`flog` [5x5] Laplacian of Gaussian filter
`fprewittx` [3x3] Prewitt filter in the x direction
`fsobelx` [3x3] Sobel filter in the x direction

These filters are all band-pass or high-pass filters used to perform edge detection
Use the function `Analyse_Filter(image,filter)` again, to see the effect of each filter on the image and their associated frequency response. This will only work on grey level images but could be easily extended to colour images. You can see that these filters are not always ideal and have a fairly smooth frequency drop-off while you would prefer a quick one for better performance. The last filter (`ftop`) is designed using an optimal design technique (Remez) mainly used in 1D signal processing using FIR filtering theory. The cut-off frequencies can be accurately chosen and the response of the filter is more controllable.

Try it on various images.

All these filters do not take into account noise in the images and are not adapted to the spectral content of the specific image we are dealing with. This is a major drawback in some applications and can be dealt with using adaptive filtering (Wiener for instance). All the filters considered here are linear filters. Some more advanced filters, usually nonlinear, have also been developed to perform feature extraction. It is useful to note filters such as Canny for edge detection.

5. Non-linear filters and Noise removal

So far we have used filters merely as feature extraction tools (edges, low frequencies). We have also only considered *linear filters* where the resulting image is obtained by convolution of the input image with the filter, making the analysis simple.
But filters are also used to perform noise removal on images corrupted by various sources of noise. The noise can be additive (Gaussian noise, salt and pepper noise) or multiplicative (Speckle noise). In each case a filter adapted to the problem must be selected and we will see a few examples of well adapted filters for each case.

4.1 Gaussian additive noise:

Using the `imnoise` function, it is possible to add a specific type of noise to an image.

For example you could do:

```
ima = imread('circuit.tif');
ima1 = imnoise(ima,'gaussian',0.2,0.01);
```

Note that the mean and variance are in the range [0,1] as a percentage of the max gray level. The `imnoise()` function is a matlab function and for Gaussian noise it requires the mean and variance of the noise to be specified.

It can be shown that Gaussian or mean filters are best used to remove this type of noise. Try the `Analyse_Filter` function again to see how you can denoise the image. For example, you can try

```
ima2 = Analyse_Filter(ima1, fmean3);
```

You should also try the other filters (`fmean5, fmean11, fgaussian5_1` etc). However, there is no theoretical backing to this assertion and adaptive filtering is a better solution in most case.

4.2 Salt and Pepper noise:

Using the `imnoise` function, it is possible to add a specific type of noise to an image. For example you could do:

```
ima = imread('circuit.tif');
ima1 = imnoise(ima,'salt & pepper',0.02);
```

It can be shown that median filters are best used to remove this type of noise.

Try the `medfilt2` function (`help medfilt2`) and see how you can denoise the image.

```
ima2= medfilt2(ima1);
```

4.3 Speckle noise:

This is a multiplicative noise, very frequent in Radar and Satellite imagery. It is very difficult to remove as it is multiplicative and very often correlated to the signal. Try:

```
ima = imread('circuit.tif');
ima1 = imnoise(ima,'speckle',0.02);
```

And try various filters on it to remove noise.

5. Demos - "Noise Reduction Filtering"

The issues of filtering for noise reduction can also be examined using the Matlab Demo – **"Noise Reduction Filtering".**

7. Adaptive Filters

As noted above an adaptive filter can often provide better noise reduction. One such filter is the Wiener filter. The matlab function `wiener2()` performs 2-D adaptive noise-removal filtering. It lowpass filters an intensity image that has been degraded by constant power additive noise. It uses a pixel-wise adaptive Wiener method based on statistics estimated from a local neighbourhood of each pixel.

```
ima_out = wiener2(ima,[M N],NOISE)
```

filters the image using pixel-wise adaptive Wiener filtering, using neighbourhoods of size M-by-N to estimate the local image mean and standard deviation. If you omit the [M N] argument, M and N default to 3. The additive noise
(Gaussian white noise) power is assumed to be NOISE.

To illustrate the difference between the mean and Wiener filters, try the following:-

```
ima = imread('circuit.tif');
ima1 = imnoise(ima,'gaussian',0.2,0.01);
ima2 = wiener2(ima, [3,3]);
ima3 = Analyse_Filter(ima1, fmean3);
figure(1);
subplot(2,2,1);
imshow(ima);
title('Original Image');
subplot(2,2,2);
```

```
imshow(ima1);
title('Noise Added');
subplot(2,2,3);
imshow(ima2);
title('Wiener Filtered Image');
subplot(2,2,4);
imshow(ima3);
title('Mean Filtered Image');
```

8. Edge Detection

A range of different edge detectors including Roberts Cross, Prewtitt, Sobel, Laplacian of Gaussian and Canny were introduced in class. Use the matlab demo "**Edge Detection**" to investigate the difference between them.

9. Additional Advanced Work - Deconvolution:

Imagine that the real scene that you are after has been distorted by the imaging system that you are using (out of focus lens, atmospheric perturbation,…). What you see is the result of the convolution of the real scene by a known (or unknown) transfer function. When the transfer function is known, you can, in the frequency domain, divided the spectrum of the observed image by the spectrum of the transfer function and recover the original image. To illustrate this, try the `deconvolution` program. Edit the program and change the level of noise to see the effect of the instabilities in the filter.
However, direct deconvolution is very sensitive to noise as you might not have access to the exact transfer function and also because you are dividing by the spectrum of the transfer function (inverse problem). For frequencies where the signal in the image is small and the noise signal is important, you are going to amplify the noise…

Better techniques can be used such as again adaptive filtering (Wiener).
If the convolution kernel (transfer function) is not known, then we are confronted to a problem called blind deconvolution where we are trying to find the convolution kernel from the data by imposing constraints on the type of images we should observe. This is an on-going area of research and is outside the scope of this course.

Session 5: The Hough Transform

1. **Objectives:**
2.

We will demonstrate various aspects of developing Hough transform code in MATLAB. In particular we will discuss:

1. How to discretize r and θ.
2. How to calculate the Calculate for each foreground pixel (x,y) in the image the values of $r=x\cos\theta + y\sin\theta$ for the selcted values of θ.
3. How to create an accumulator array whose sizes are the number of angles θ and values r in the chosen discretizations.
4. How to step through the values of r and update the acculator array as you proceed.

2. Resources required:

In order to carry out this session, you will need to download images and matlab files from

http://www.see.ed.ac.uk/~sml/Image_Processing/

Please download the following images:

- Image of an electronic circuit
- square.tif image

You are of course encouraged to try these programs out on images of your choice.

3. Steps to implement the algorithm

3.1 Discretizing r and θ

Observe that even though a particular pair (r, θ) corresponds to only one line,a given line can be parameterised in a variety of ways. For example, the line x+y=-3 can be parameterised as $\theta=5\pi/4$ and $r=\sqrt{(3/2)}$. Alternatively if we use $\theta=\pi/4$ and $r=-\sqrt{(3/2)}$. So there is a choice of where the values of θ can be restricted to $-\pi/2\leq\theta\leq\pi/2$and let r have both positive and negative values, or we may let θ take any value chosen from the range $0\leq\theta\leq2\pi$ and restrict r to non-negative values.

Question: Bearing in mind what we will wish to do with the values what would be the most sensible option?

We can choose any discrete set of values we like, but let us take all the integer degree values in the given range and convert them to radians for the purposes of calculation.

```
>>angles=[-90:180]*pi/180;
```

3.2 Calculating the r values

If the image is binary then we can find the positions of all the foreground pixels with a simple application of the *find* function.

>>[x,y]=find(im);

If the image is not binary, we can create a binary edge image by a variety of techniques discussed in the lecture course.

Once you have a binary image it is straightforward to calculate the *r* values as follows:

*>>z=floor(x*cos(angles)+y*sin(angles));*

Where *floor* is used to obtain integer values.

3.3 Forming the accumulator array

If an image is *m x n* pixels the if the origin is to be at the top left corner then the maximum value of r will be $\sqrt{(m^2+n^2)}$. The size of the accumulator array can then be set at $\sqrt{(m^2+n^2)}*270$. However, we are only interested in positive values of *r* given the choice of the range of θ. By finding the largest positive value of *r* and use that as the dimension of the array.

>>rmax=max(r(find(r>0)));
>>acc=seros(rmax+1,270);

3.4 Updating the accumulator array

We must now step through the array of *r* values, and for each value (r, θ) increase the corresponding accumulator value by one.

>>if (r(i,j)>0, acc(r(i,j)+1,i)=acc(r(i,j)+1,i)+1;

Note:this is not a very efficient method of creating the accumulator array.

4. Exercises

Simplifications and details

Use the image files square.tif and circuits.tif in the following and use the methods discussed above to develop a hough transform..

Only one line must be found in the image. The line should correspond to the maximum of the accumulator. The input image is square, that is, it has an equal number of rows and columns. The size of the accumulator should be the same as the image.

Hints for implementing the algorithm

Do not try to build the complete functionality at once. Build small parts which you can test with simple data.

Step 1:
You can use the matlab function `find`. Try `help find` in Matlab. Notice that in Matlab the first index refers to the row (y-coordinate) of a matrix, and the second one corresponds to the column number (x-coordinate).

Step 2:
Use the normal-form parameterisation of the line.