2.1B：Introduction to MatLab Image Library

Chap. 1: Fundamental

1.1: input and output　　1.2: Data type and matrices　　1.3: Image arithmatics

1.1: Image Input, output and display

1.1.1: Read and Display

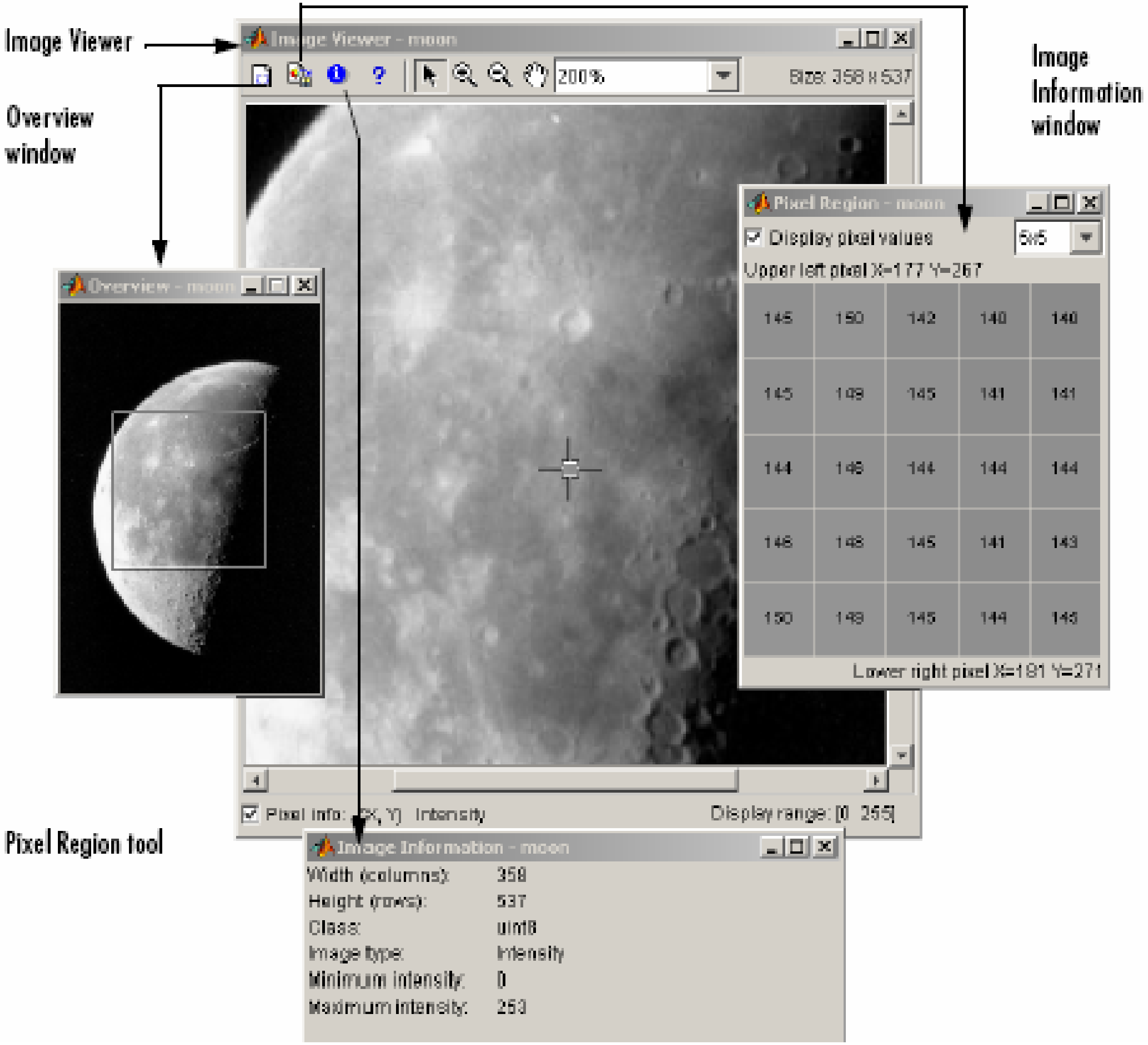
```
clear;close all;clc;    % Clear all variables in the workspace and close figure windows.
I = imread('pout.tif');    % Reads an image from a file named ‘pout.tif’ and stores it %
                           in an array named ‘I’.
imshow(I)                  % Display the image in the array (matrix) I.
```

- There are two function for displaying images on the screen: `imshow` and `imveiw`.

  `Imshow`: access to figure annotation and printing capabilities.

  `Imview`: displays in a separate Java-based window and provide additional tools in navigating and inspecting around an image, especially large images.

Image viewer and related tools



☑ Pixel info: (X, Y) Intensity

Display range: [0 255]

☑ Display pixel values 5x5

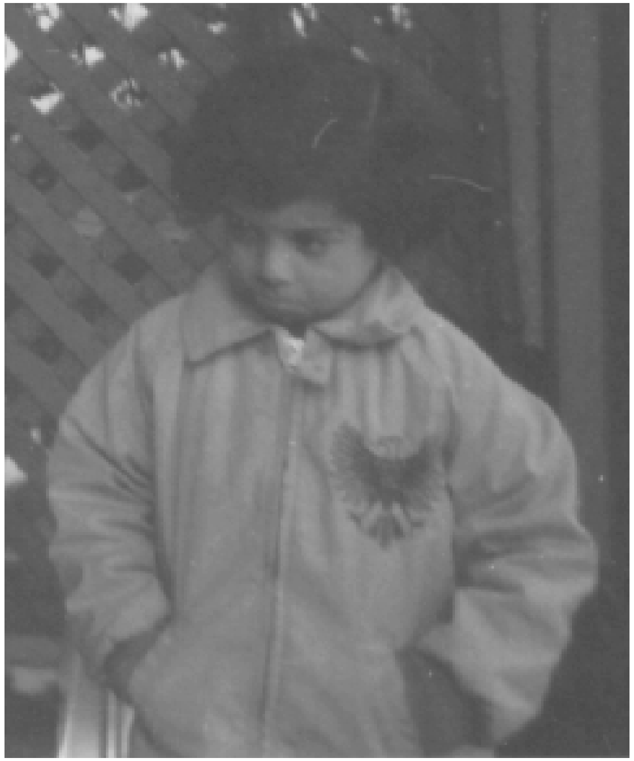
Upper left pixel X=177 Y=267

145	150	142	140	140
145	149	145	141	141
144	148	144	144	144
146	148	145	141	143
150	149	145	144	146

Lower right pixel X=181 Y=271

Width (columns):	358
Height (rows):	537
Class:	uint8
Image type:	intensity
Minimum intensity:	0
Maximum intensity:	253

Imshow

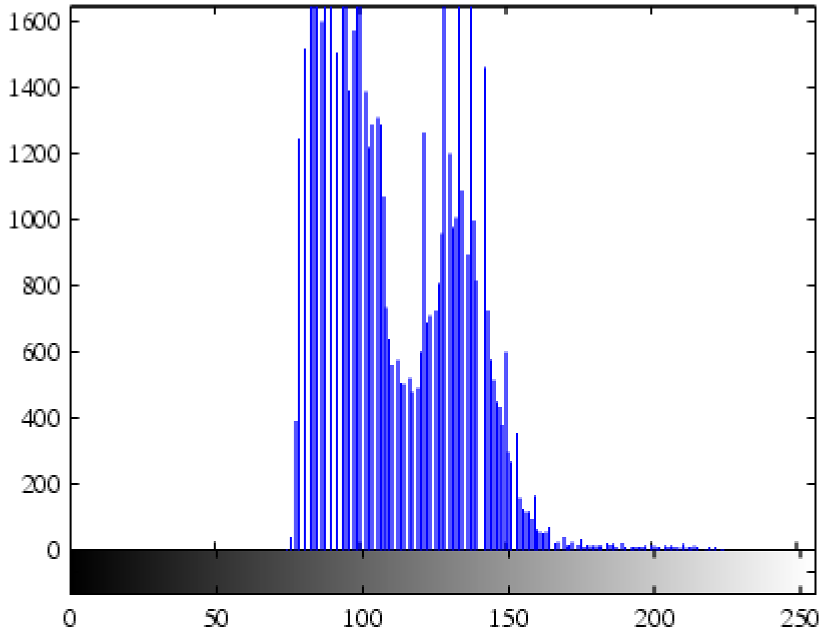


On the left is the image ‘pout.tif’ saved in the ‘I’ matrix and displayed using the MatLab command

```
Imshow(I)
```

1.1.2: Check intensity distribution and Modify the image

```
figure, imhist(I)    % figure command holds the image shown before.
                     % imhist command shows the intensity distribution (histogram)
                     % of the image I.
```



The histogram of image ‘I’ shown by using the following command:

```
figure, imhist(I)
```

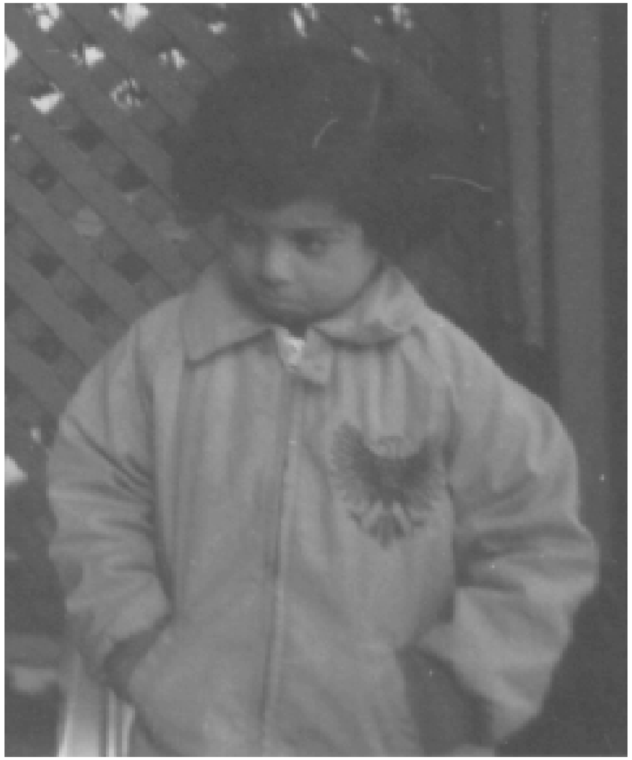
The intensity is ‘condensed’ in a narrow range.

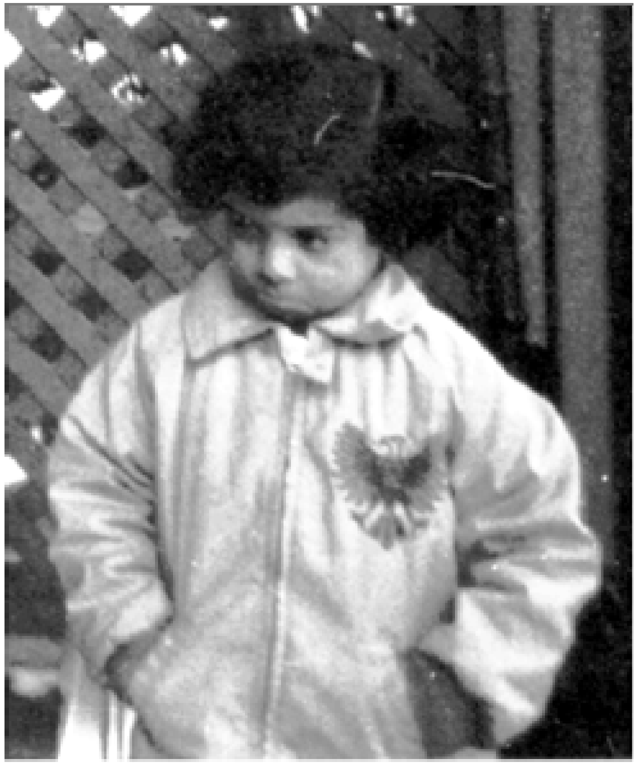
-

I2 = histeq(I); % `histeq` function spreads the intensity values over the full range so that the contrast is enhanced as will be seen later.

imview close all **% To close all java-based windows opened by `imview`.**

figure, imshow(I2) % To show image ‘I2’ as is given in the next page.





The left image ‘I2’ now has a better contrast than the original ‘I1’ due to ‘stretch’ histogram.

1.1.3：Save a matrix as an image and check an image in the workspace

```
imwrite (I2, 'pout2.jpg');    % Save the matrix ‘I2’ in the format ‘JPEG’ under the
                              %name ‘pout2’.
```

```
imfinfo('pout2.jpg')        % Check the properties of ‘pout2.jpg’ (e.g. colour or
                            % gray image, format, size, etc.)
```

1.2. Data types and matrices

When processing, MatLab save each image as a matrix. For gray image, it is a $M \times N$ matrix；while for color image, it is a $M \times N \times 3$ matrix.

However, elements of image matrices usually are integer elements >=0, while elements of ordinary matrices are usually floating points. Since they have different data types, operations defined for ordinary matrices (floating point data) cannot be applied to matrices of images (positive integers), neither can operations defined on matrix images be applied to ordinary matrices.

Hence one should always be aware of the data type of a matrix (representing image or ordinary floating-point data) in applying MatLab functions.

In general, operations which we are familiar, such as 「+」、「-」、「*」、「conv2」 and so on are defined for ordinary matrices (in the format of floating-point).

On the other hand, operations applicable to images (matrices with positive integer elements) generally begins with two characters 「im」, such as「imread」、「imshow」、「imadd」、「imsubtract」.

1.2.1: Types of matrices and legal operations

There are several types of matrices to save an image

- uint8 and uint16：save the image data as 8-bit or 16-bit unsigned integers (i.e., all elements are semi-positive definite integers).
- double：save image data as 64-bit floating-point numbers.
- logical：a matrix with all elements as 0's (off pixels) and 1's (on pixels).

Example:

`A=[2 -4; 0 8];B=double(A);C=uint8(A);BW=im2bw(A);` returns

$$A=\begin{bmatrix}2 & -4\\ 0 & 8\end{bmatrix},\ B=\begin{bmatrix}2.0 & -4.0\\ 0.0 & 8.0\end{bmatrix},\ C=\begin{bmatrix}2 & 0\\ 0 & 8\end{bmatrix},\ BW=\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$$

Most MATLAB functions accept only `double` (double-precision) format and reject `uint8` or `uint16` data, while `find`, `all`, `any`, `conv2`, `convn`, `fft2`, `fftn`, and `sum` functions accept all three types.

For instance, basic MATLAB arithmetic operators do not accept `uint8` or `uint16` data. If you attempt to add two `uint8` images, `A` and `B`, you get an error, such as

- `C = A + B`
- `??? Function '+' not defined for variables of class 'uint8'.`

To perform addition (or other arithmetic operations) of two `uint8` images, `A` and `B`, you have to use「`imadd`」(or「`imsubtract`」、「`immultiply`」、「`imdivide`」)

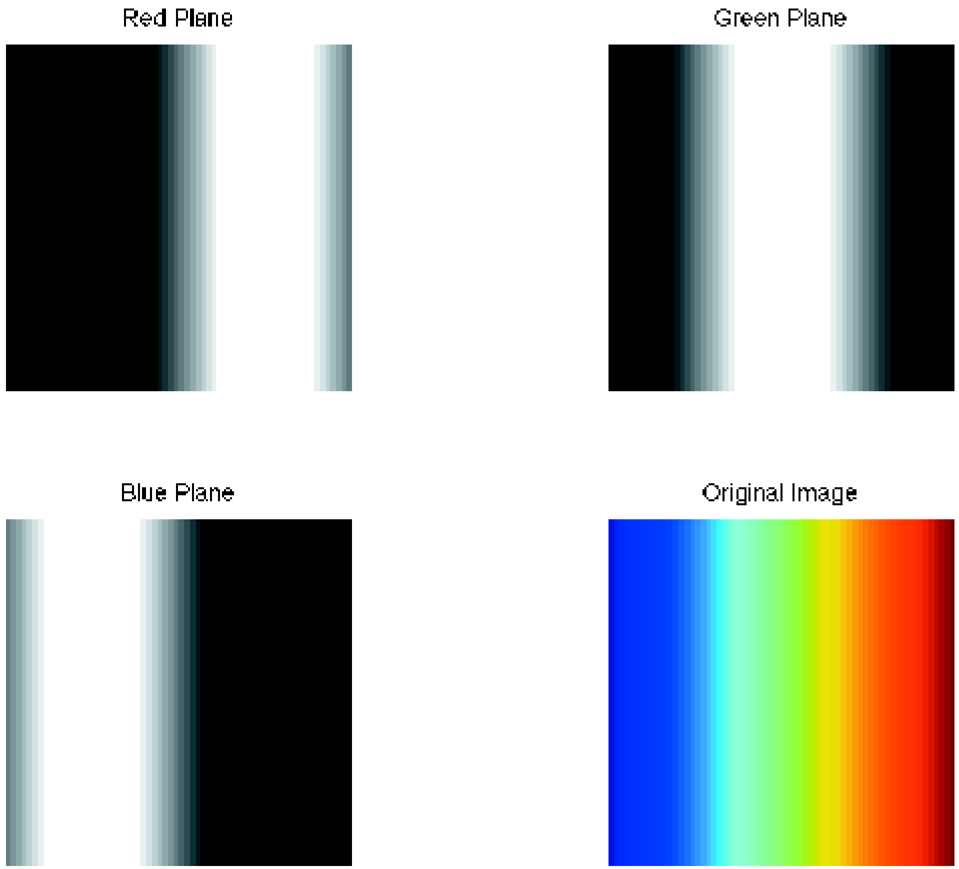
1.2.2: Types of matrices for image storages

Images are stored as matrices in MatLab, hence the above types of matrices corresponds to types of images

- Binary images: data are stored using matrices of the format `logical`
- Gray images: data are stored as matrices in the format `uint8`、`uint 16`、`double`
- Colour images: data are stored as a $M \times N \times 3$ `uint8`、`uint 16`、`double` matrices with the 1st , 2nd and 3rd columns represents R、G、B components of an image.

Example

```
•   RGB=reshape(ones(64,1)*reshape(jet(64),1,192),[64,64,3]);
•   R=RGB(:,:,1); G=RGB(:,:,2); B=RGB(:,:,3);
•   imshow(R),figure,imshow(G),figure,imshow(B),figure,imshow(RGB)
```



- Indexed image: Image data consists of R、G、B components as (`X, map`).

  To `Imread` or `imwrite` an indexed image you need to key in (`X, map`)

Image Type	Storage Class	Data type and Interpretation
Binary	`logical`	zeros (0) and ones (1)
Indexed	`double`	integers in the range [1, *p*]
	`uint8` or `uint16`	integers in the range [0, *p*-1]
Intensity[1]	`double`	floating-point. Typical value range [0, 1].
	`uint8` or `uint16`	Array of integers. Typical range [0, 255].
RGB	`double`	m*n*3 floating-point values in the range [0, 1].
	`uint8` or `uint16`	m*n*3 integers in the range [0, 255].

- [1] For intensity images the colormap is typically grayscale.

1.2.3: Conversion between data types of matrices and image types

- Conversion between data types

You can use `uint8`, `double` and `im2bw` to convert data types of a matrix before using a proper operation

Example:

```
A=[2.4 4.8; 2.5 6]; B=uint8(A); C=double(B); BW=im2bw(A)
```

Returns

$$A = \begin{bmatrix} 2.4 & -4.8 \\ 2.5 & 6 \end{bmatrix},\ B = \begin{bmatrix} 2 & 0 \\ 2 & 6 \end{bmatrix},\ C = \begin{bmatrix} 2.0 & 0.0 \\ 2.0 & 6.0 \end{bmatrix},\ BW = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

- Conversion between image types

You can convert image type (data storage type) using the following functions:

Function	**Description**
gray2ind	Create an indexed image from a grayscale intensity image
grayslice	Create an indexed image from a grayscale intensity image by thresholding
im2bw	Create a binary image from an intensity image, indexed image, or RGB image
ind2gray	Create a grayscale intensity image from an indexed image
ind2rgb	Create an RGB image from an indexed image
mat2gray	Create a grayscale intensity image from data in a matrix, by scaling the data
rgb2gray	Create a grayscale intensity image from an RGB image
rgb2ind	Create an indexed image from an RGB image

The followings are some example algorithms for gray images

Function	**Description**
imcomplement	Complement an image（適用 binary & gray images，用 1 or 255 去減）
imadd	Add two images（pixel-by-pixel 亮度相加）
imsubtract	Subtract two images（pixel-by-pixel 亮度相減）
imabsdiff	Absolute difference of two images（先「-」再求差的絕對值「｜｜」
imdivide	Divide two images (element-by-element division)
immultiply	Multiply two images(element-by-element multiplication)
imlincomb	Compute linear combination of two images

Homework I-A

(1) **Read** several gray images captured by an industrial camera into the workspace of MatLab.

(2) **Check** the **properties** of the above input images (type, format and size)

(3) Examine the **histogram** of each image.

(4) Choose a proper **threshold value** for each image and **write a program** to change the gray image into a binary image.

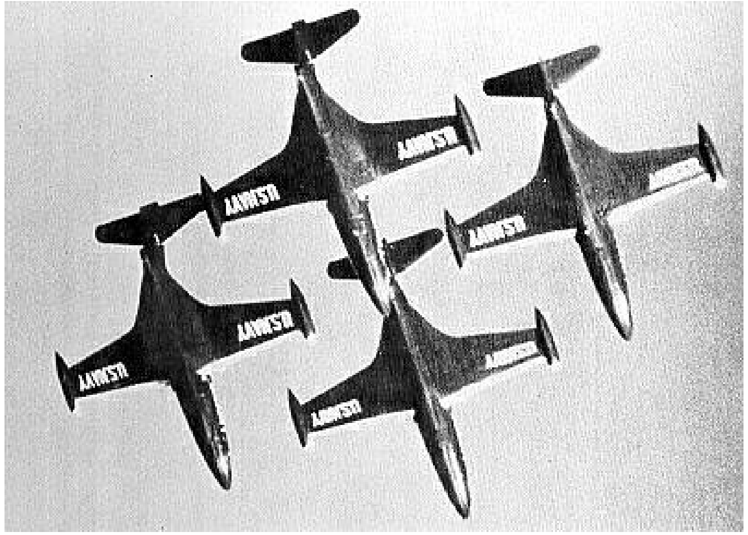
(5) Display the **binary images** one-by-one using ‘**Imshow**’.

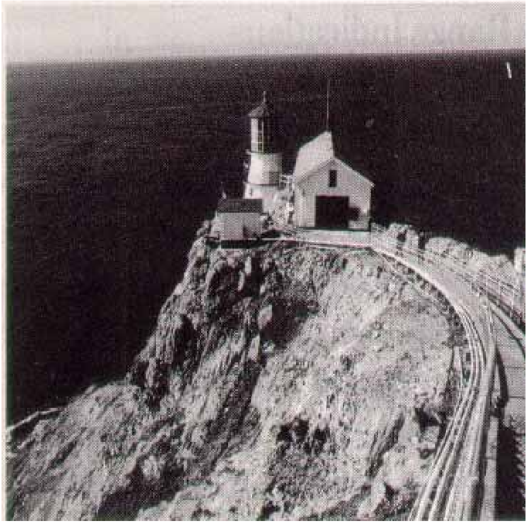
(6) Examine the gray level of each image by using ‘**Imviewer**’, and explain why the results in step (4) are.

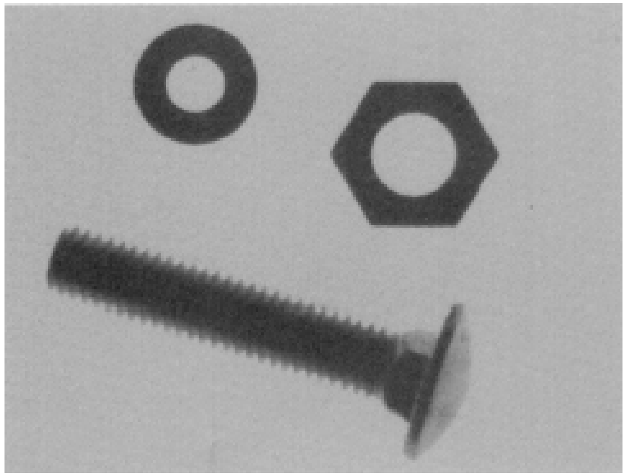
(7) Applying the **arithmetic algorithms** in page 14 to each gray images and see how they work by using ‘**Imshow**’ and ‘**Imviewer**’. ■

Homework I-B

Please use histogram and thresholding to segment objects in the following windows.







For more Information regarding MatLab

You may find the following website useful:

(a) Homepage of MatLab Image Processing Toolbox

http://www.mathworks.com/access/helpdesk/help/toolbox/images/images.html

(b) You can download a user's guide from

http://www.mathworks.com/access/helpdesk/help/pdf_doc/images/images_tb.pdf

(c) You can also find a list of MatLab functions in the Image Processing Tool box at

http://www.mathworks.com/access/helpdesk/help/toolbox/images/referenc.html#f