Image Processing with MATLAB

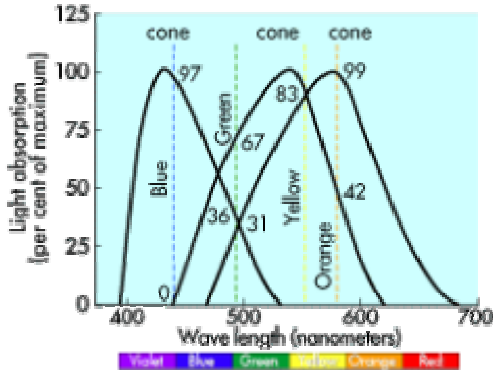
- **What is digital image processing?** Transforming digital information representing images
- **Motivating problems:**
  - Improve pictorial information for human interpretation
    - Remove noise
    - Correct for motion, camera position, distortion
    - Enhance by changing contrast, color
  - Process pictorial information by machine.
    - Segmentation - dividing an image up into constituent parts
    - Representation - representing an image by some more abstract models
    - Classification
  - Reduce the size of image information for efficient handling.
    - Compression with loss of digital information that minimizes loss of "perceptual" information. JPEG and GIF, MPEG,
    - Multiresolution representations versus quality of service
- **How do we see?**
  - Lens focuses an image on the retina (like a camera).
  - Pattern is affected by distribution of light receptors (rods and cones) --see Figure 2.2 of CIP.
  - The (6-7 million) cones are in the center of the retina (fovea) and are sensitive to color - each connected to own neuron.
  - The (75-150 million) rods are distributed everywhere, connected in clusters to a neuron
  - Unlike ordinary camera, the eye is flexible.
  - Range of intensity levels supported by the human visual system is 10^{10}.
  - Uses brightness adaptation to set sensitivity.

Color Vision

The color-responsive chemicals in the cones are called **cone pigments** and are very similar to the chemicals in the rods. The retinal portion of the chemical is the same, however the scotopsin is replaced with photopsins. Therefore, the color-responsive pigments are made of retinal and photopsins. There are three kinds of color-sensitive pigments:

- Red-sensitive pigment
- Green-sensitive pigment
- Blue-sensitive pigment

Each cone cell has one of these pigments so that it is sensitive to that color. The human eye can sense almost any gradation of color when red, green and blue are mixed.



In the diagram above, the wavelengths of the three types of cones (red, green and blue) are shown. The **peak absorbancy** of blue-sensitive pigment is 445 nanometers, for green-sensitive pigment it is 535 nanometers, and for red-sensitive pigment it is 570 nanometers.

(Taken from www.howstuffworks.com)

Pixels:

Pixel based images are created by painting programs such as Adobe **Photoshop** and Microsoft's Paintbrush. These are the most common of digital images. Photographs, for example, are described by breaking an image up into a mosaic of colour squares (pixels). Depending on their final destination, the number of pixels used per inch varies (PPI or DPI). On the web where big file sizes mean long downloads, only 72 pixels per inch are required since monitors only display 72 ppi. For publishing, anywhere from 200-1200 ppi is required depending on the press and desired quality. Laser printers usually print at anywhere from 300 to 600 dpi. That is why they produce such sharp images.

MATLAB stores most images as two-dimensional arrays (i.e., matrices), in which each element of the matrix corresponds to a single pixel in the displayed image. For example, an image composed of 200 rows and 300 columns of different colored dots would be stored in MATLAB as a 200-by-300 matrix. Some images, such as RGB, require a three-dimensional array, where the first plane in the 3rd dimension represents the red pixel intensities, the second plane represents the green pixel intensities, and the third plane represents the blue pixel intensities.

To reduce memory requirements, MATLAB supports storing image data in arrays of class uint8 and uint16. The data in these arrays is stored as 8-bit or 16-bit unsigned integers. These arrays require one-eighth or one-fourth as much memory as data in double arrays. An image whose data matrix has class uint8 is called an 8-bit image; an image whose data matrix has class uint16 is called a 16-bit image.

Indexed Images

An indexed image consists of a data matrix, *X*, and a colormap matrix, *map*. *map* is an *m*-by-3 array of class `double` containing floating-point values in the range [0, 1]. Each row of *map* specifies the red, green, and blue components of a single color. An indexed image uses "direct mapping" of pixel values to colormap values. The color of each image pixel is determined by using the corresponding value of `X` as an index into `map`. The value 1 points to the first row in `map`, the value 2 points to the second row, and so on. You can display an indexed image with the statements:

```
image(X); colormap(map)
```

A colormap is often stored with an indexed image and is automatically loaded with the image when you use the `imread` function. However, you are not limited to using the default colormap--you can use any colormap that you choose. The description for the property `CDataMapping` describes how to alter the type of mapping used.
The next figure illustrates the structure of an indexed image. The pixels in the image are represented by integers, which are pointers (indices) to color values stored in the colormap.

Imread:

`A = imread(filename,`*`fmt`*`)` reads a grayscale or truecolor image named `filename` into `A`. If the file contains a grayscale intensity image, `A` is a two-dimensional array. If the file contains a truecolor (RGB) image, `A` is a three-dimensional (`m`-by-`n`-by-3) array.

DEFINITIONS:

Rotation:

```
B = IMROTATE(A,ANGLE,METHOD)
```

A is your image.
ANGLE is the angle (in degrees) you want to rotate your image in the counter clockwise direction.
METHOD is a string that can have one of
these values:
If you omit the METHOD argument, IMROTATE uses the default method of 'nearest'. **Note:** to rotate the image clockwise, specify a negative angle.

The returned image matrix B is, in general, larger than A to include the whole rotated image. IMROTATE sets invalid values on the periphery of B to 0.

Scaling:

IMRESIZE Resizes the image.

IMRESIZE resizes an image of any type using the specified interpolation method. Supported interpolation methods include:

'nearest' (default) nearest neighbor interpolation?

'bilinear' bilinear interpolation?

'bicubic' bicubic interpolation ?

`B = IMRESIZE(A,M,METHOD)` returns an image that is M times the size of A. If M is between 0 and 1.0, B is smaller than A. If M is greater than 1.0, B is larger than A. If METHOD is omitted, IMRESIZE uses nearest neighbor interpolation.

`B = IMRESIZE(A,[MROWS MCOLS],METHOD)` returns an image of size MROWS-by-MCOLS. If the specified size does not produce the same aspect ratio as the input image has, the output image is distorted.

```
a= imread('image.fmt'); % put ur image in place of image.fmt.
» B = IMRESIZE(a,[100 100],'nearest');
» imshow(B);
» B = IMRESIZE(a,[100 100],'bilinear');
» imshow(B);
» B = IMRESIZE(a,[100 100],'bicubic');
» imshow(B);
» B = IMROTATE(a,45);
» imshow(B);
```

Edge Tracing

```
b=rgb2gray(a); % convert to gray. WE can only do edge tracing for gray images.

 edge(b,'prewitt');
 edge(b,'sobel');
 edge(b,'sobel','vertical');
 edge(b,'sobel','horizontal');
 edge(b,'sobel','both');
```

FINALLY

So that's enough of definitions, but it was necessary to get you familiar with the language! The primary object of this lab is to get you comfortable with MATLAB and prepare you for future classes.
Lets get started!

Download an image into your U:\ee186 folder. Choose any image you like!

Now using the instructions from the previous lab, read the image and store it as any variable you want to. For example, I will be using 'A' throughout this lab!

Let's Rotate the image

```
>>B=imrotate(A,30);
>>Imshow(B)
```

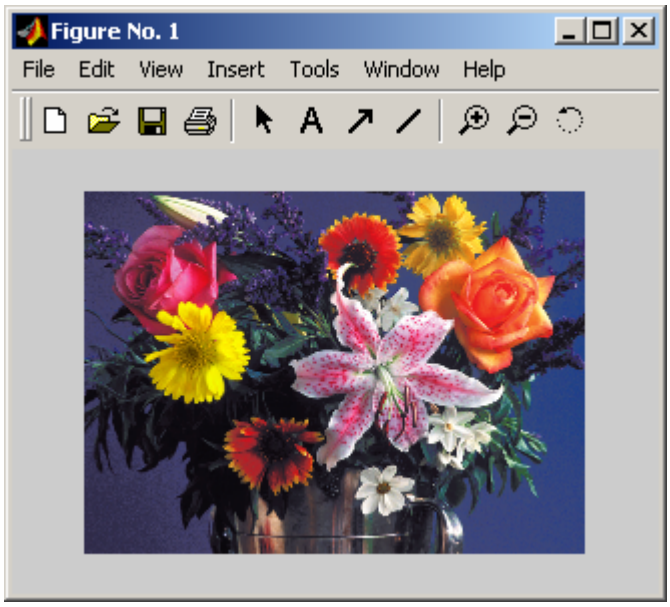
That code above, rotates the image by 30 degrees remember it is counterclockwise.

Ok now rotate the image by 90 and 270 degrees, from the clock wise direction. See what happens.

Resize the image

Like mentioned in the definitions there are two ways to resize an image.

```
>> C=imresize(A,0.5);
>> imshow(C)
```



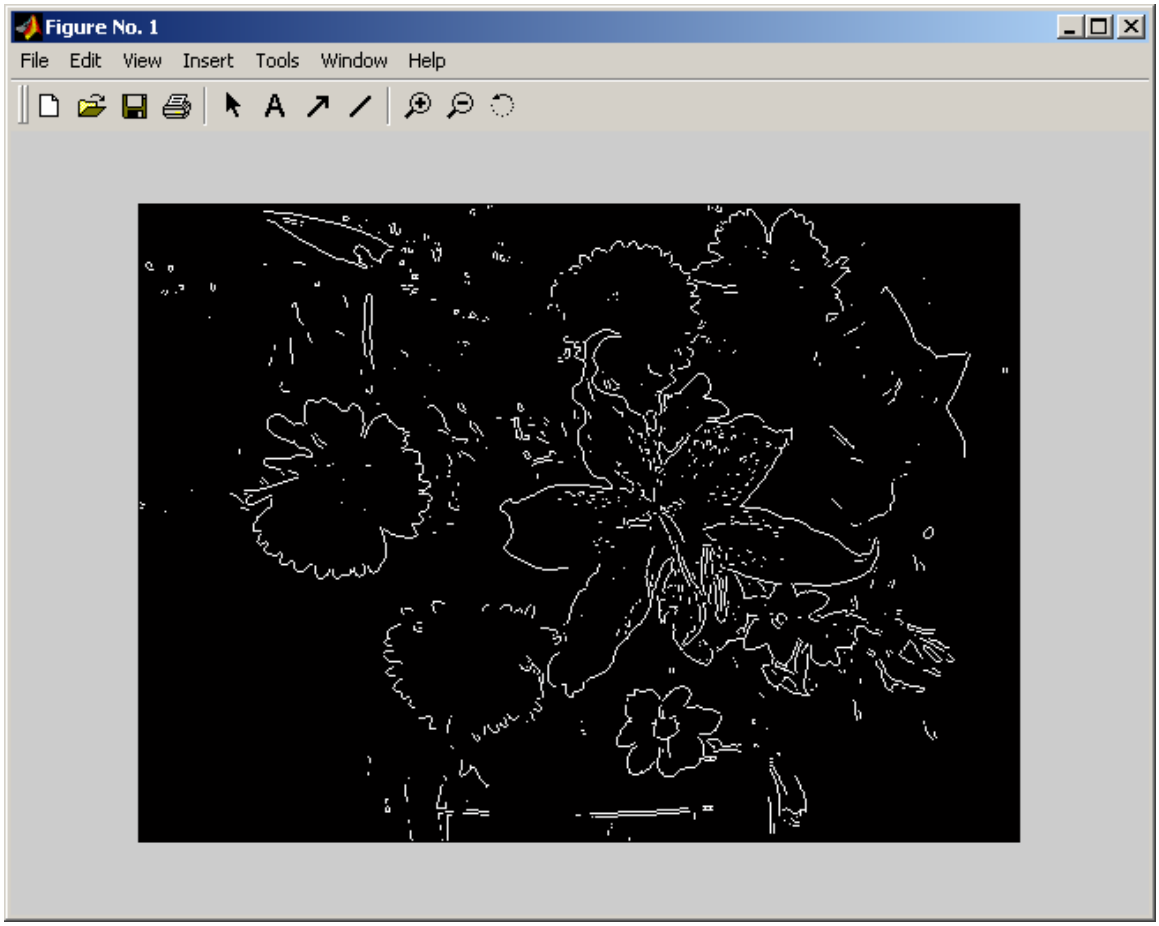
This is what happened to the flower image when I shrunk it by half!

Try the alternate method (`IMRESIZE(A,[MROWS MCOLS],METHOD))` and notice what happens. Have fun with it; do not be scared of it! Find out the difference between bilinear, bicubric and nearest.

Edge tracing

We can only do edge tracing using gray scale images (i.e images without color).

>> BW=rgb2gray(A);
>> edge(BW,'prewitt')



That is what I saw!

>>edge(BW,'sobel','vertical')
>>edge(BW,'sobel','horizontal')
>>edge(BW,'sobel','both')

Notice what happens when you do each of these edge tracings and make a note of it in your Lab report. In your own language try and explain what exactly happens to the image once each of these lines of code are executed.

- **What exactly is the use of digital image processing as Electrical Engineers?**
- **What is the use of edge tracing, resizing or rotating an image?**
- **Could you team up with another group and identify how can each of the operation you did in today's laboratory be used in a practical case. For example one can say that if I am designing a auto pilot to land an unmanned plane, one way to detect the exact location of the landing strip is by first doing and edge detection and then some additional processing to see what are the lines that converge at the distance etc.**

Keep saving every line of code you execute and save every image you create. It will be a great addition to your portfolio. Remember its your portfolio that talks about you as an engineer!