Basics of MATLAB

Nishith Tirpankar

January 27, 2012

1 Basics

1.1 Starting Matlab

Windows users need to navigate to the MATLAB install directory and execute the Matlab application **matlab** from the **$MATLAB/bin/** directory. Linux users can run the same by running **./matlab** from the **$MATLAB/bin/** directory.

Matlab is installed on all the CADE lab machines. If you wish to remotely work on the cade machines perform the following steps:

1. SSH with or without the X option to any cade machine with the command
   ssh -X username@lab1-4@eng.utah.edu
2. Navigate to the matlab install directory
   cd /usr/local/bin
3. Run matlab using:
   ./matlab

1.2 Getting Help for Matlab

The best source of information is the MATLAB help which is insstalled with the orignal product. If you are using the graphical interface navigate to the help menu and click on "Product Help". Search for the command or keyword. If you are running the console version of MATLAB, and want the details regarding a particular function, use the command

help functionName.

Another excellent source is the mathworks website `http://www.mathworks.com/help/techdoc/`. The website has a examples, tutorials, user guides and a lot more.

1.3 Reading and writing images

Images are treated as matrices in MATLAB. The first simple command that you need to know to know is:

imageMatrix = imread('fileName');

This command reads the image file and writes it out into the matrix "imageMatrix". If the image file is grayscale, the matrix has dimensionality $[n \times m]$. If the image is a 3 channel color image, then the dimensionality of the matrix is $[n \times m \times 3]$. n is the height of the image while m is the width.

If you wish to modify the value at a certain pixel location you need to access it by its index. Note that the matrix indexing starts from 1 and not 0. For example if you wish to change the pixel value at $(10, 10)$ of the "imageMatrix" then we can do it as:

imageMatrix(10,10) = 30;

There are various ways of selecting subsets of matrices to operate upon. For a better description go to `http://www.mathworks.com/company/newsletters/articles/Matrix-Indexing-in-MATLAB/matrix.html`.

1.4 Converting a color image to grayscale

On some occassions, you may require to convert a color image to grayscale. This can be done using the following two ways:

rgbImage = imread('someColorImage');
grayImage1 = rgb2gray(rgbImage);
grayImage2 = 0.2989*rgbImage(:,:,1) + 0.5870*rgbImage(:,:,2) + 0.1140*rgbImage(:,:,3);

This first method is an inbuilt matlab function. The second command we explicitly use the colorspace conversion transform. Depending upon the colorspace we are dealing with, we can use the relevant colorspace conversion transform.

1.5 Displaying images

MATLAB graphics are generally displayed in graphics object called "figure". The command to invoke a new object is simply **figure**. The overloaded calls of the function allow you to access various properties and the handle to the graphics object. You can display various objects in a "figure". Let us look at the methods to display images in the figure. There are two major ways of displaying images. The first simple way is:

image(imageMatrix);
OR
imshow(imageMatrix);

The above method does not scale the image data to occupy the entire colormap. If you wish to scale the data appropriately you can use:

imagesc(imageMatrix);

The above commands display a single image in a figure window. To display multiple objects in the same figure use:

subplot(n,m,i); imagesc(imageMatrix1);
subplot(n,m,j); imagesc(imageMatrix2);
subplot(n,m,l); imagesc(imageMatrix3);

The command "subplot" breaks the figure window into an $n \times m$ grid of axes. Each axis object is counted from left to right, top to bottom starting from the top left.

1.6 Writing images to files

To write matrix data into common image file formats use the command:

imwrite(imageMatrix, 'fileName', 'fileFormat') This command will write the image matrix to the file specified in the desired file format.

1.7 Data Types

When performing any type of operation on variables be aware of its data type. Many issues in MATLAB scripts arise due to incorrect usage of a specific data type. **imread** outputs the data type **uint8** or **uint16**. As a general rule try to convert your data to a double precision floating point before performing any operations on it. To convert a "uint8" matrix to a "double":

imageMatrix = double(imageMatrix);

To convert it back:

imageMatrix = uint8(imageMatrix);

2 Read cursor position and values

```
[x,y] = ginput(n)
```

enables you to select n points from the current axes and returns the x- and y-coordinates in the column vectors x and y, respectively. You can press the Return key to terminate the input before entering n points.

```
[x,y] = ginput
```

gathers an unlimited number of points until you press the Return key.
Link:
More information about this function

3 Point operation

3.1 Thresholding an image (binarize) with specified threshold

Example: Suppose we have a image which size is 3×5, we want to threshold this image using 0.5. The following code does it and we set all pixels which are larger than the threshold as 1, the pixels which are smaller or equal to the threshold as 0.

```
I = rand(3,5);
I(I>0.5) = 1;
I(I<=0.5) = 0;
```

3.2 Scaling intensity values.

Example: Suppose we have a image which size is 4×7, the range of the image is $[0, 1]$, we want to rescale the image range to $[0, 255]$. The following is the code to do this.

```
I = rand(4,7);% original image which range is between 0 and 1
Inew = I*255;
```

3.3 Others

Example: If we have two matrix (A, B) and we want to do component-wise matrix multiplication, we need to do the following. (The sizes of A, B are the same.)

```
C = A .* B;
```

If we have two matrix (A, B) and we want to do standard matrix multiplication, we need to do the following. (Now, the sizes of A, B may not the same, we just need to make sure A is $m \times n$ and B is $n \times l$).

```
C = A*B;
```

4 Filtering

4.1 Neighborhood filtering

Convolution of an mask with an image is a useful operation. This process is called spatial filtering or neighborhood filtering. Let us have a look at a simple filtering function that convolves any $n \times n$ filter where n is odd with an image of arbitrary width and height. Note that we are only dealing with 2 dimensional images here.

```
%Function: signedFilterImage
%parameters:    I = 2D array containing image data
%               maskSize = size of the Square mask applied
%               mask = Actual mask array of size maskSize x maskSize.
%               Account for the normalisation considering the division
%               factor.
%returns:   signedFilter = 2D array containing filtered image data. Please
%                          note that no border padding is done; hence the
%                          border pixels are left unfiltered
```

```
function [signedFilter] = signedFilterImage(I, maskSize, mask)

Idouble = double(I);
maskdouble = double(mask);

tempImage = double(I);

for i = ((maskSize-1)/2)+1 : (size(I,1) - ((maskSize-1)/2))
    for j = ((maskSize-1)/2)+1 : (size(I,2) - ((maskSize-1)/2))
        subImage = Idouble(i-((maskSize-1)/2) : i+((maskSize-1)/2), ...
    j-((maskSize-1)/2) : j+((maskSize-1)/2));
        filterResult = subImage .* maskdouble;

        pixelValue = double(0);
        for p = 1:size(filterResult,1)
            for q = 1:size(filterResult,2)
                pixelValue = pixelValue + filterResult(p,q);
            end
        end

        tempImage(i,j) = abs(pixelValue);
    end
end

signedFilter = uint8(tempImage);

end
```

This function will take an image, a $n \times n$ filter mask and the value of n to output a convolved image.

4.2 Averaging neighbourhood filter

The simple averaging neighborhood filter is as follows:

$$\begin{pmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{pmatrix}$$

Figure 1 shows the result of application of the above filter mask on the image of the capitol building. As can be seen from the figure 1, the averaging mask has smoothed the image.

4.3 Edge detection

Another application of filtering is edge detection. This can be done by convolving two separate filters that detect the x and y gradients and then computing the magnittude of the resultant. The filter masks used are:

$$\begin{pmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

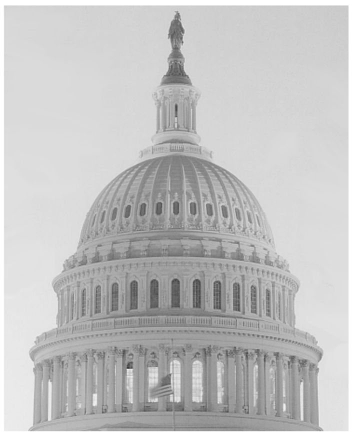
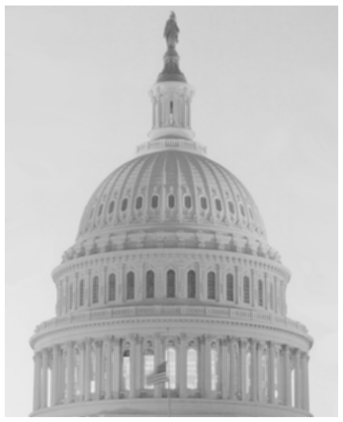

Figure 1: **Left:**Orignal image of the capitol building **Right:**Averaging filter applied to the capitol image.

to compute the x gradient and

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & -1 & 0 \end{pmatrix}$$

for computing the y gradient.

Figure 2 shows the result of the gradient filter applied to the capitol image.

4.4 Overlaying binary map using color overlay

In the example in section 4.3, we can find the edge map by simply thresholding the gradient image. In order to correlate the input image with the edge map we overlay the thresholded result onto the orignal image in red. Assuming that “imageMatrix“ is the orignal image and ”imageEdge“ is the thresholded edgemap we can create an overlay by:

```
imageOverlay(:,:,1) = imageMatrix+imageEdge;
imageOverlay(:,:,2) = imageMatrix;
imageOverlay(:,:,3) = imageMatrix;
```

The overlay image contains the edgemap overlaid in Red color on the orignal image.

Figure 3 shows the result of overlaying.

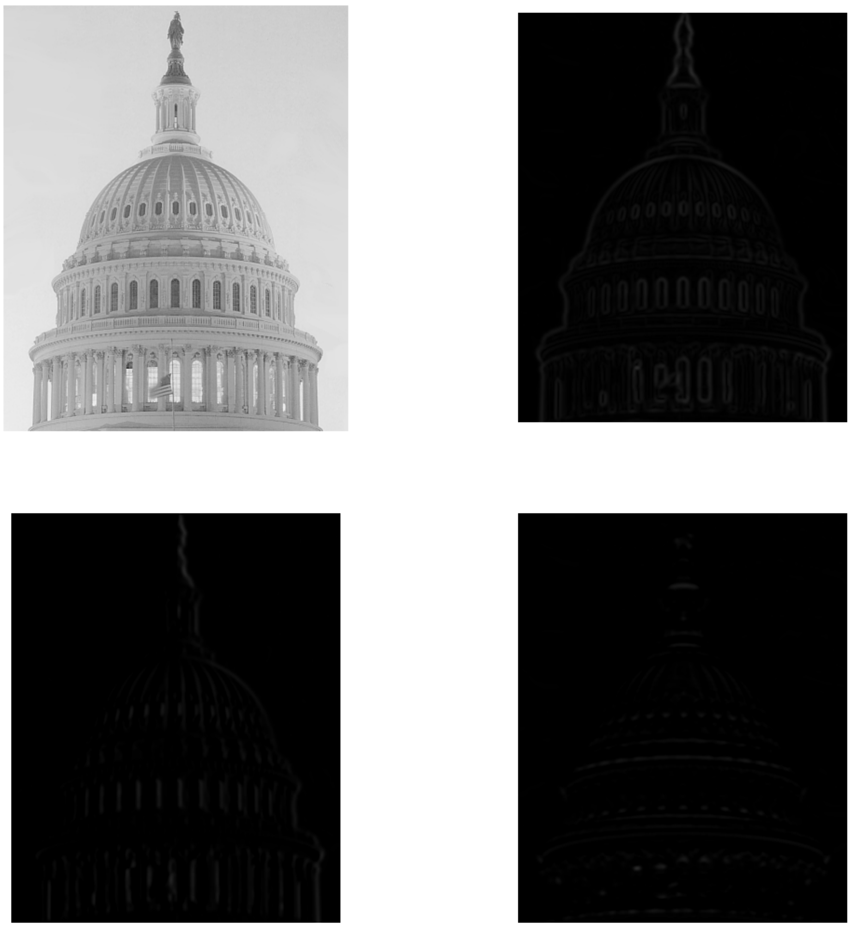

Figure 2: **Top Left:**Orignal image of the capitol building **Top Right:**Magnittude of the gradient.**Bottom Left:**Gradient along the X axis. **Bottom Right:**Gradient along the Y axis.

5 Math operations

5.1 Introduce SVD (preparation of camera calibration)

Example, we have a matrix X which size is $m \times n$, we want to do singular value decomposition.

```
[U,S,V] = svd(X);
```

Link:
More information about this function in MATLAB website
SVD in wikipedia

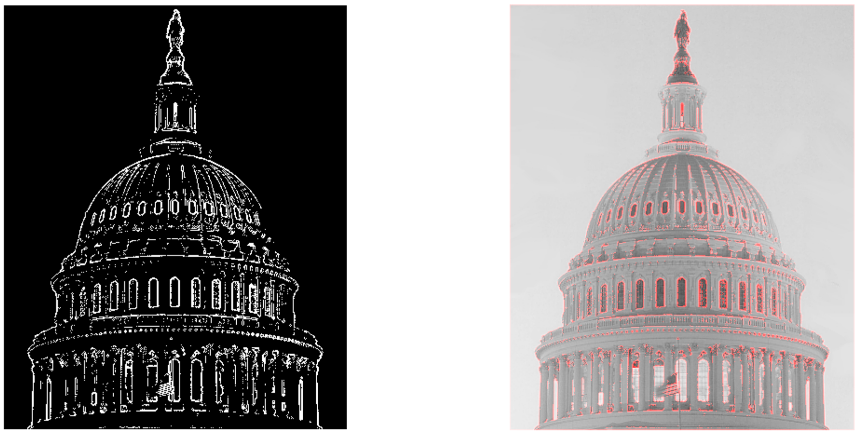

Figure 3: **Left:**Edge map of the capitol image **Right:** Edge map overlay on the orignal.