Introduction to Computational and Biological Vision
Matlab Tutorial

General
Matlab stands for Matrix Laboratory and is an interpreted language and technical computing IDE by MathWorks. It can be used as an interactive numerical shell and it's particularly strong when dealing with matrices.
This tutorial will cover the most basic knowledge you should have in order to fulfill this course's programming assignments.

Matrices
In Matlab, every numeral is seen as a matrix; that is, $a = 4$ defines a as being a 1x1 matrix, where $a(1,1) = 4$. An array or vector of length n is a 1xn matrix. To build a matrix we define its lines separating them by ';', as follows:

$$>> A = [1, 2; 3, 4]$$

$$A = \begin{matrix} 1 & 2 \\ 3 & 4 \end{matrix}$$

The regular mathematical operations are used as the matrix ones. So if we define a new matrix $B = [1, 2; 1, 2]$ and we multiply A by B, we get:

$$>> A * B$$

$$ans = \begin{matrix} 3 & 6 \\ 7 & 14 \end{matrix}$$

To perform cell-wise operations, we have to precede the operator by a '.'. So, if we want to multiply every cell of A by its corresponding cell in B, we do:

$$>> A. * B$$

$$ans = \begin{matrix} 1 & 4 \\ 3 & 8 \end{matrix}$$

The inverse matrix of A is given by $inv(A)$. *Reminder:* $inv(A) * A = I$.

$$>> inv(A)$$

$$ans = \begin{matrix} -2 & 1 \\ 1.5 & -0.5 \end{matrix}$$

The transpose matrix of A is given by A'.

$$>> A'$$

$$ans = \begin{matrix} 1 & 3 \\ 2 & 4 \end{matrix}$$

Both the transpose and the invert operators don't change the value of the variable and if we want to use this result afterwards, we have to assign it to a variable, as $C = inv(A)$ for example.
Special 'pre-made' matrices:

- $zeros(m)$ and $zeros(m, n)$ build a square mxm matrix and a mxn one, respectively, where all the cell values are 0.

$$>> zeros(2, 1)$$

$$ans = \begin{matrix} 0 \\ 0 \end{matrix}$$

- $ones(m)$ and $ones(m, n)$ will have the same effect but with cell value of 1.

$$>> ones(2)$$

$$ans = \begin{matrix} 1 & 1 \\ 1 & 1 \end{matrix}$$

- $eye(m)$ will build the mxm Identity matrix

$$>> eye(3)$$

$$ans = \begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix}$$

A fast way to build vectors that have equal-spaced elements is by defining the first of the elements, the interval between every two consecutive elements, and the last element of the vector. So, $v = x : d : y$ will return the vector $v = [x, x + d, x + 2d, ..., y - d, y]$.

$$>> v = 1 : 1 : 5$$

$$v = \begin{matrix} 1 & 2 & 3 & 4 & 5 \end{matrix}$$

If we don't report the interval d, it is supposed to be $d = 1$; so $v = 1 : 5$ will have the same result as $v = 1 : 1 : 5$.

Loops and Functions

- *for loop*:

```
for var = vec
   statement-1
   statement-2
   ...
   statement-n
end
```

where *vec* is a vector that contains all the ordered values that *var* will receive during the loop.

Remember: when programming in Matlab we'll prefer to use matrices instead of simple loops. So, if we want to build a new matrix from the values of two other matrices, instead of looping over an index and calculating the values cell by cell, we'll just do the operations needed over the existing matrices. For example, instead of

```
for t=1:size(C)
   A(t)=B(t)+C(t)
end
```

we'll simply use $C = A + B$.
Note: in Matlab, the letter i serves as an indication of the complex part of a number. In order to avoid confusion it shouldn't be used as a varible. For example, the code

```
for i=1:10
   a= 1+2*i
end
```

will generate a very different output from

```
for i=1:10
   a= 1+2i
end
```

- *if condition*:

```
if cond
   do-if-true-statements
else
   do-if-false-statements
end
```

- *function*:

```
function [ReturnedValue1, ..., ReturnedValueN] = functionName(arg1, ..., argm)
```

Dealing with matrices
The call to $size(A)$ returns the number of rows and columns of the matrix A in an array.

$$>> [m, n] = size(A)$$

$$m = 2$$

$$n = 2$$

where m is the amount of rows and n of columns.
To have access to a cell in the matrix we use the parenthesis. Please, note that in Matlab the first index of the matrices is 1, not 0 as in other well-known programming languages.

$$>> A(1, 2)$$

$$2$$

$$>> A(1, 1) = 0$$

$$A = \begin{matrix} 0 & 2 \\ 3 & 4 \end{matrix}$$

In order to get a sub-matrix from a matrix M, we use $M(m1 : m2, n1 : n2)$. So,

to receive the sub-matrix of M contained between lines 3 and 5 and columns 1 and 4, we use $M(3:5, 1:4)$. If we do not want to specify one of these dimensions, we use a ':'. So $M(1:5, :)$ will return the sub-matrix of M consisting of rows 1 to 5 and all the columns of M.

Dealing with Pictures
In order to read a picture:

$$>> I = imread('pictureName.jpg');$$

If the picture is black and white, I will be a two dimensional matrix with cell values between 0 and 255. The elements of the matrix will be 8-bit unsigned integers (Uint8).
If the picture has colors, then I will be tri-dimensional and composed by the red, green and blue matrices that together describe the picture. We can get each one of the RGB matrices through I, for example: $red = I(:, :, 1)$.
For image processing we'll want to work with a matrix of doubles. So, we'll cast:

$$>> I = double(I);$$

And, if needed we may cast back:

$$>> I = uint8(I);$$

The $figure$ command opens a new graphic window, in which we can show a image or a graph.
The command $imshow(I)$ will show the image I in the last graphic window opened, while *imagesc* will scale the image and display it.
To save the image, we may use: $imwrite(I,'pictureName.jpg')$.

Image Processing

- *threshold*:
$$>> A = I > 50;$$
For every cell in I, the correspondent cell in A will have value 1 if the original cell in I had a value bigger than 50; otherwise, the cell value will be 0. Please, note that A is a binary matrix.

- *2D convolution* $a * b$:
$$>> C = conv2(A, B);$$
There are three types of convolution in Matlab: *full* ,*valid* and *same*. *Full* is the default method and when not specified Matlab will do a full convolution, that is, it'll return the full two-dimensional convolution. *Valid* will return only elements of the convolution that were calculated without need of the zeros used for padding the edges. Finally, *same* will return just the central part of the convolution that has the same size as A.

- *edge detectors*:
edge is the Matlab built-in command for edge detection and it returns a binary matrix representing the edges found in the picture. It can run

several different types of edge detection, such as *prewitt*, *canny*, *sobel*, *zerocross*, etc.

$$>> E = edge(I,' sobel');$$

The Psychtoolbox

The Psychophysics Toolbox, or Psychtoolbox, is a set of compiled functions (a.k.a. mex files) for Matlab that allows it to control the computer's hardware. Most of its functions are written in C (callable from Matlab). The kind of interface provided by this toolbox is most needed when trying to accurately display stimuli (timing and colors) and collect precise response data (reaction times and user inputs).
The first thing one should learn when using a new tool is how to get help:

```
help Psychtoolbox      % prints an overview of the toolbox
help Screen            % gives a summary of the 'Screen' function
help PsychDemos        % returns a list of demo programs
Screen(OpenWindow?)    % gives the usage and a summary of an OpenWindow call
```

Here are some of the basic routines of the psychtoolbox:

- Rush - executes a piece of code with little or no interruption, blocking OS interrupts for a few seconds. It can be used when the timing of a stimulus presentation is critical.
- Screen - is the most used routine of the toolbox. Almost all needs related to the stimuli display are attended by this function.
  Note: when starting to use the toolbox it is easy to get stuck with the screen 'frozen' after using this command. In order to get back to the Matlab IDE, one can blindly type clear Screen.
- KbWait, KbCheck - waits for/checks a keyboard input.
- Snd - plays a sound.

Let's take a look on some code written using the toolbox:

```
[winPtr,winCoords] = Screen(0,'OpenWindow');
                     % 0 = main monitor
                     % winPtr is a window pointer;
                     % winCoords will hold the rectangle coordinates
                     % for the upper an the lower corners of the screen
                     % (0,0) is the upper left corner

xCenter = winCoords(3)/2;
yCenter = winCoords(4)/2;
white = WhiteIndex(winPtr); % the pixel value for the 'white' color at winPtr
black = BlackIndex(winPtr); % same for black
gray = (white+black)/2;
escKeyCode = KbName('ESCAPE'); % defines the 'esc' key to be a known value;
OKBeep = sin(1:3000);          % defines a sinusoidal wave

% we'll load some of the mex funcs to memory now:
GetSecs;   % gets the system time (since the computer started)
```

```
KbCheck;   % gets the keyboard status
Snd('Play',zeros(1,100))  % doesn't play anything
% loading finished
HideCursor;               % we don't want to see the cursor anymore

for picIndex = 1:200
  picFileName = strcat('pictures/pic',int2str(textureIndex),'.jpg');
  picMatrix  = imread(textureFileName);
                           % reads the picture
  textureArray(picIndex) = Screen('MakeTexture',winPtr,picMatrix);
                           % makes a texture from the picture
end;

Screen(winPtr, 'FillRect', gray);
                           % fills the offscreen window buffer with gray
Screen(winPtr, 'Flip');    % displays the offscreen buffer contents by
                           % setting the onscreen buffer to the offscreen one
                           % and creates a new blank offscreen buffer;

%now the monitor is displaying just gray pixels

KbWait;                    % waits for any keyboard input
                           % returns the time waited until the input

Screen('FillRect',winPtr,[255 0 0],[xCenter yCenter xCenter+100 yCenter+100])
                           % draws a rectangle of color RGB=[255 0 0]
Screen(winPtr, 'Flip');    % displays the red rectangle
keyIsDown = 0;
while ~keyIsDown
  [keyIsDown,reactTime,keyCode]=KbCheck();
                          % checks for keyboard input
end;
k= find(keyCode);         % finds the code of the key pressed

if (k(1) == escKeyCode)
  Screen(winPtr,'FillRect',white);
  msg= sprintf('Experiment completed. Thanks for participating!');
  Screen('DrawText',winPtr,msg,0,0,black);
  Screen(winPtr, 'Flip');
else
  for t = 1:size(textureArray) % rapidly presents a series of pics
    Screen('DrawTexture', winPtr, textureArray(t));
    Screen(winPtr,'Flip');
  end;
  Screen(winPtr,'FillRect',white);
  msg= sprintf('Press another key');
  Screen('DrawText',winPtr,msg,0,0,black);
  Screen(winPtr, 'Flip');
end;
```

```
Snd('Play',OKBeep);
Screen('CloseAll');
ShowCursor;
return;
```