**Human-Oriented Robotics** Winter term 2013/2014
Prof. Dr. Kai Arras University of Freiburg
Lab instructors: Timm Linder, Luigi Palmieri, Billy Okal Department of Computer Science

# Exercise: Matlab/Octave Basics

### Exercise 1: Getting Started

1. Make sure that Matlab or Octave/Gnuplot is properly installed. Start the program, Matlab should open its IDE, Octave should display a shell prompt such as `octave:1>`.
2. For Octave users: verify the installation and the interplay with gnuplot. Type `octave:2> plot(1:10,1:10)` (or another plot command). A window should pop up showing a diagonal line from one to ten. In case of errors, return to the start.
3. Download the `librobotics` library from `http://srl.informatik.uni-freiburg.de/downloads`. Unpack it, put it at a place that makes sense (you will use it throughout the course) and add it to your path by using the command `addpath`
4. Verify the system by typing `help drawrobot`. A help text should appear.
5. You are ready to go!

### Exercise 2: Vectors and Matrices

1. **Vectors:** Create two vector `a = [1 2 3 4 5]`, `b = [0; 1; 3; 6; 10]`. Display the transpose of `a` and `b` and apply a couple of vector functions on `a` and `b`: compute, for instance, the vector of cumulative sums for `a` and the vector of differences of `b` using `diff`. Remember, you can suppress unwanted output to the shell using semicolon.
2. **More vector operations:** Multiply vector `a` with itself: `a*a`. Explain the result. Multiply `a` elementwise with itself, then take the third power of each element of `a`. Compute the inner product of `a` and `b`. Compute the outer product of `a` and `b` and assign the result to `M`.
3. **Workspace:** List the variables that are on your workspace by typing `whos`. There should be (among others perhaps) `a, b, M`.
4. **Matrices:** Get the 2nd row of `M`, then get its 4th column by using the colon operator `:`. Get a submatrix from `M`, e.g. the one that contains the 1st, 3rd and 5th row and the three last columns. Note that you can use the keyword `end` in the expression to index the columns.
5. **Matrix operations:** Invert matrix `M` and explain the result. To look for built-in commands, use tab completion and the help system.
6. **Relational operators:** Assign all elements of `M` greater than 9 the value -1.
7. **Size:** Get familiar with the `size` command, display the sizes of `a`, `b`, `M`. Make use of the second argument of `size`. Create a matrix of ones in the size of `M`, create a matrix of normally distributed random numbers in the size of `M`.

**Exercise 3: Plotting in 2D**

1. **Multiple plots:** Define a range of x-values, let's say 200 values between -4 and 4. Compute sine, cosine, arctangent, and the 3rd order polynomial $y = x+0.3\,x^2-0.05\,x^3$ on this interval. Plot the functions into the same window.
2. **Annotations:** Add title, axis labels, and a legend.
3. **Line styles:** Familiarize yourself with the `plot` command including its line style options.

**Exercise 4: Functions and Scripts, More Plotting**

1. **Functions:** We will now define our first function, `plotcircle` that plots a circle. The function shall take three arguments, the x- and y-coordinates of the circle center and the radius. **Hint**: Create first a range of angles then define vectors of the circle's x- and y-values. Set the property `'LineWidth'` to 4 of the `plot` command.
2. **Scripts:** Create another file, the script from which we call `plotcircle.m`. We use UpperCamelCase notation for scripts, so call it `PlotCircleDemo.m`. In the script, open a new figure and write an example call of `plotcircle`. Use the command `axis` to adjust the axes and the aspect ratio if needed.
   Redefine the function `plotcircle` to take an forth input argument `color`. The argument should be a 3-by-1 row vector of RGB-values. Extend the plot command in `plotcircle` by the `'Color'` property.
   Then write a for-loop in the script with 100 randomized radii, positions and RGB-colors and create some post-modern art. Finally, turn the axes off.

**Exercise 5: 2D Range Data Segmentation**

We want to segment a 2D laser scan using a simple jump-distance criterion. This is useful if we wanted to classify segments, for example, to find people or different objects in such data.

1. **Get and plot raw data:** Read the laser points from `scan.txt` into a matrix `scan`. The readings are in polar coordinates, angles are in the first column, ranges in the second one. Put them into row vectors `phi` and `rho`. Convert the points into Cartesian coordinates using `pol2cart`, then plot.
2. **Preprocess scan:** The scan contains several erroneous readings with a maximum range of 8 meters. This can happen, for instance, when the laser return signal is too weak due to specular reflection or infrared-absorbing surfaces. Filter out all laser points whose ranges are greater than 7.5 meters. Plot the filtered scan.
3. **Find break points:** Find the break points in the scan. A break point is where the range values of two neighboring points differ ("jump") by more than 0.3 meters. **Hint**: Use commands `diff` and `find`.
4. **Build segment data structure:** Build a array of struct for segments with the following fields: a unique segment identifier, the segment's begin and end indices, the points that constitute the segment and the segment's center of gravity.
5. **Plot segments:** Plot all segments using a different randomized color for each segment. Annotate the segments with their identifier at the respective center of gravity-position using the command `text`.
6. **Filter segments:** Redo steps 4 and 5 but filter out segments that have fewer than 3 points.