Albert-Ludwigs-Universität Freiburg | Institut für Informatik
Lecture: Mobile robotics II | Exercise: Prof. Dr. Kai Arras
Winter term 2011/2012 | Lab instructor: Dipl.-Inf. Jonas Rist

Exercise: First Steps in Octave/Matlab

Exercise 1: Getting Started

1. Download and install Octave onto your computer (e.g. from `http://www.gnu.org/software/octave/download.html`). There are pre-built binaries for most common operating systems.

2. Start the program. You should see a prompt in a shell such as `octave-3.4.0:1>`.

3. Verify the installation by typing `octave-3.4.0:1> 1+1` (or alike). You should get an answer of the form `ans = 2`.

4. Verify the graphical output by typing `octave-3.4.0:2> plot(1:10)`. A window should pop up showing a diagonal line from one to ten.

5. Download the `librobotics` library from `http://srl.informatik.uni-freiburg.de/downloads`. Unpack and add it to your path by using the command `addpath('yourpath/librobotics');`.

6. Verify the system by typing `octave-3.4.0:3> help drawrobot`. A help text should appear that describes the interface of the `librobotics`-specific command `drawrobot`.

7. You are ready to go!

Exercise 2: Vectors and Matrices

1. **Vectors:** Create a row vector: `a = [1 2 3 4 5]`. Display the transpose by typing `a'`. Create a column vector: `b = [0; 1; 2; 3; 4]`. Display the transpose. Without further notice we will assume all vectors being column vectors.

2. **Ranges:** Create a range of values from 1 to 10: `r1 = 1:10`. Introduce a non-integer stride: `r2 = 1:0.2:10`. Create a linearly spaced row vector `r3` of equally spaced points using `linspace`. Get help, `help linspace`, then define a range of 100 points between 0 and 2π including the limits 0 and 2π. Try to suppress unwanted outputs to the shell using semicolon `;`.

3. **Vector operations:** Multiply vector `a` with itself: `a*a`. Explain the result. Multiply `a` elementwise with itself: `a.*a`. Try also: `a.^3`. Let us now compute the inner product of two vectors by typing `a*b`, the result is a scalar. Compute the cross product `cross(a(1:3),b(1:3))`, the result is a vector. Finally, compute the outer product, the result is a matrix. Assign the result to `M`.

4. List the variables that are on your workspace by typing `whos`. There should be (among others perhaps) `a, b, r1, r2, r3, M`.

5. **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 `3:end` to index the columns.

6. **Matrix operations:** Compute the rank of matrix `M`, explain the result. Compute the determinant of `M`, explain the result. To look for built-in commands, use tab completion and the help system.

7. **Relational operators:** Assign all elements of `M` greater than 9 the value -1.

8. **Size:** Get familiar with the `size` command, display the sizes of `a`, `b`, `r3`, `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`.

9. **Solving linear equation systems:** Let us now define matrix `A = [1 2 3; 4 5 6; 7 8 10]` and redefine vector `b` to be a 3-by-1 column vector of ones. To solve the system $Ax = b$, we can use the built-in backslash operator \. Given that the number of equations equals the number of unknowns, we obtain the exact solution by `x = A\b`. Verify correctness by computing the residual vector `r = b - A*x`. The same operator also applies for over- and underdetermined equation systems, see below.

Exercise 3: Plotting in 2D

1. Define a range of x-values, let's say between -4 and 4. Compute sine, cosine, arctangent, and a 3rd order polynomial on this interval. For the polynomial, use elementwise operations `y = x+0.3*x.^2-0.05*x.^3`. Open a figure window by typing `figure(1)` and plot the functions into the same window.

2. Add title, axis labels, and a legend.

3. Familiarize yourself with `plot`. Note the line style options that the command offers.

Exercise 4: Functions and Scripts, More Plotting

1. **Functions:** Start a text editor of your choice. We will now define our first function, `plotcircle`. The function shall take three arguments, the x- and y-coordinates of the center of a circle and its radius. Since function m-files must be named after the function they contain, save the file as `plotcircle.m`.

2. **Hint**: After defining the function header, create a range of angles and call `plot` with two properly defined vectors of x- and y-values.

3. **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 window, write an example call of `plotcircle` and execute the script. Use the command `axis` to adjust the axes or the aspect ratio if needed. Redefine the function to take an additional color input argument `col`. The argument should be a 3-by-1 row vector of RGB-values. In the function, change the plot command to `plot(x,y,'Color',col)`. Then write a for-loop in the script with randomized radii, positions and RGB-colors and create some post-modern art.

4. **Solving linear equation systems (cont.):** We will now solve an overdetermined linear equation system using the pseudoinverse A^+ of a matrix. The pseudoinverse computes a 'best fit' solution to a overdetermined system of linear equations in a least-squares sense. We will use this to fit of a circle to points in Cartesian coordinates.

   Given a set of n points $\{(x_i, y_i)\}_{i=1}^n$, we parameterize the problem by the vector of unknowns $x = (x_c \quad y_c \quad x_c^2 + y_c^2 - r_c^2)^T$ with x_c, y_c and r_c being the circle center and radius. We then establish the overdetermined equation system $Ax = b$ with elements

   $$A = \begin{pmatrix} -2x_1 & -2y_1 & 1 \\ -2x_2 & -2y_2 & 1 \\ \vdots & \vdots & \vdots \\ -2x_n & -2y_n & 1 \end{pmatrix} \quad b = \begin{pmatrix} -x_1^2 - y_1^2 \\ -x_2^2 - y_2^2 \\ \vdots \\ -x_n^2 - y_n^2 \end{pmatrix}$$

   and, assuming A has full rank, solve it by

   $$x = A^+ b \quad \text{with} \quad A^+ = (A^T A)^{-1} A^T.$$

   The first step is to create the matrix that holds our points in Cartesian coordinates. In a new script, called `FitCircleDemo.m`, type `P = [0 2 5 7 9 3; 7 7 8 7 5 7];` All x-values are in the first, all y-values are in the second row. Build up the matrix `A` and vector `b`. Preallocate the elements using the command `zeros`. Then solve the system as shown above and assign the result to `x1`. Solve the system using the command `pinv` and assign the result to `x2`. Solve the system with the backslash operator, assign it to `x3`. Compare the results.

5. In the script, open a figure window and plot the points. If you want them to display as large green dots, type `plot(x,y,'g.','MarkerSize',20);`. Add `hold on` to avoid that subsequent plot commands erase the figure content again. Extract the circle parameters from x and plot the best fit circle. Does the result make sense?

Exercise 5: Plotting using librobotics, Printing

1. Finally, we want to use `librobotics` to generate a figure of a robot that drives through an environment modeled as a set of landmarks and store it as an .eps file. Create a new script file, give it an appropriate name, and make sure the path is properly set to include the library. In the first lines, you should open a figure window, and allow superimposed plotting using `hold`. Set the random seed with `rand('seed',2);`.

2. Generate ten randomly distributed 2D landmark poses: `x = 5*rand(3,10);`. Plot a reference frame at each pose with label `'E'`, size 0.5 and black color (command `drawreference`). We assume the landmark positions to be uncertain, modeled as normally distributed random variables. To visualize their uncertainty, we plot, at the same positions than the landmarks, Gaussian error ellipses at a significance level of 0.95 with covariance matrices generated by `diag(0.02*rand(3,1))`. Use `drawprobellipse` and, say, blue color.

3. Plot the robot and a (hypothesized) trajectory: `drawtransform([1 5],[3 2.6],'\','',[0 0.5 0]);` and `drawrobot([2.95 2.4 -pi/2],'k');`. You might want to normalize the axis now using `axis equal;`. Execute the script.

4. We finally generate an .eps file because we want to include it into a paper or report that we are writing. This is done using the `print` command. In the shell, type `print -depsc myfilename.eps`. Look at the file with an eps-viewer or alike. The command can also be part of your script, then the arguments form a comma-separated list of strings: `print('-depsc','myfilename.eps');`.