Mobile Robotics Winter term 2024/2025

Lecturer: Prof. Kai Arras
Lab instructors: Till Hielscher, Dennis Rotondi, Fabio Scaparro
Submission instructions: see box
Submission and voting deadline in ILIAS: Oct. 25, 14:00

Socially Intelligent Robotics Lab
Institute for Artificial Intelligence
Faculty of Computer Science
University of Stuttgart

Submission Instructions: For the solution of theoretical tasks, use a header with your name and Matrikelnummer on each sheet and combine all files (pictures, scans, LaTeX-ed solutions) into a single `.pdf` document. For code solutions, if not stated otherwise, your code should execute correctly when called from a single `.m` or `.mlx` script (external functions are ok as long as they are called from the script). Each file you submit must include a header with your name and Matrikelnummer. Please add comments to make your code readable and to indicate to which task and subtask it refers to. For submission, all files should be included in a single `.zip` archive named as: `Ex01_YourLastname_Matrikelnummer.zip`. Remember to vote on the tasks that you solved and be ready to present them.

Exercise 01: MATLAB

Exercise 01.1: Getting Started

a) Make sure that Matlab (we recommend Matlab, as you have a license for it from the university) or Octave/Gnuplot is properly installed. Start the program, Matlab should open its IDE, Octave should display a shell prompt such as `octave:1>`.

b) For Octave users: verify the installation and the interplay with gnuplot. Type `octave:2> plot(1:10,1:10)` (or any another plot command). A window should pop up showing a diagonal line from one to ten. In case of errors, return to the start.

c) You are ready to go!

For this exercise create a script `Ex01_Matlab.m` and follow the tasks in this script if not stated otherwise.
At the top put a comment with your name and your student registration number (Matrikelnummer).
Label the tasks and subtasks with comments.

Exercise 01.2: Vectors and Matrices

a) **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 semicolons.

b) **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`.

c) **Workspace:** List the variables that are on your workspace by typing `whos`. There should be (among others perhaps) `a, b, M`.

d) **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.

e) **Matrix operations:** Invert matrix `M` and explain the result. To look for built-in commands, use tab completion and the help system.

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

g) **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 01.3: Plotting in 2D

a) **Multiple plots:** Define a range of x-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.

b) **Annotations:** Give the plots different colors, add title, axis labels, and a legend.

c) **Line styles:** Familiarize yourself with `plot`: line types, plot symbols and predefined colors.

Exercise 01.4: Functions and Scripts, More Plotting

a) **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.

b) **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 a forth input argument `color`. The argument should be a 1-by-3 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 01.5: Calculate π

We want to calculate π using a Monte Carlo approach. If a circle of radius r is inscribed into a square with side length equal to $2r$, then the area of the circle is πr^2, while the area of the square is $(2r)^2$. The ratio of the two areas equals $\frac{\pi}{4}$.

If you randomly generate N uniformly distributed points in the square, approximately $N\frac{\pi}{4}$ points fall into the circle. The value of π can then be approximated by $\frac{4 \cdot K}{N}$, with K being the number of points falling into the circle.

a) **Generate samples:** Generate N points uniformly distributed in the range $[-1, 1] \times [-1, 1]$

b) **Count inliers:** Determine the number of points K that fall into the unit circle.

c) **Compute π:** Calculate an estimate for $\hat{\pi}$ and the error $e = |\hat{\pi} - \pi|$.

d) **Plot:** Plot the circle, plot the square and plot the circle inliers and outliers in two different colors. Give the figure a title that contains the resulting $\hat{\pi}$. Use `plot` to plot the square and familiarize yourself with the `scatter` command for coloring the points. Define an own colormap if needed.

Exercise 01.6: 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.

a) **Get and plot raw data:** Load the laser points from `scan.txt` into a matrix `scan`. The readings are in polar coordinates, angles in the first column, ranges in the second. Put them into row vectors `phi` and `rho`. Convert the points into Cartesian coordinates using `pol2cart`, then plot.

b) **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.

c) **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`.

d) **Build segment data structure:** Build a array of struct for segments with the following fields: a unique identifier, the segment's begin and end indices, the points of the segments and the segment's center of gravity.

e) **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`.

f) **Filter segments:** Redo steps 4 and 5 but filter out segments that have less than 3 points.