Kyung Hee University
Lecturer: Jiyoung Jung
Lecture: SWC331 Robot Programming
Fall 2020

Sheet 1

Topics: Matlab

General Notice

We will be using Matlab for the programming exercises. Matlab is a command line program for solving numerical computations. Campus license for faculty and students in Kyung Hee University is available. All you have to do is to make a MathWorks account with your KHU email address. It is available for Linux, Mac OS, and Windows. Install Matlab on your system in order to solve the programming assignments. A quick guide to Matlab is given in the Matlab cheat sheet.

Exercise 1: Getting familiar with Matlab

The purpose of this exercise is to familiarize yourself with Matlab and learn basic commands and operations that you will need when solving the programming exercises throughout this course.
Go through the provided Matlab cheat sheet and try out the different commands. As pointed out in the sheet, a very useful Matlab command is `help`. Use it to get information about the correct way to call any Matlab function.

Exercise 2: Implementing an odometry model

Implement an Matlab function to compute the pose of a robot based on given odometry commands and its previous pose. Do not consider the motion noise here.
For this exercise, we provide you with a small Matlab framework for reading log files and to visualize results. To use it, call the `main.m` script. This starts the main loop that computes the pose of the robot at each time step and plots it in the map. Inside the loop, the function motion command is called to compute the pose of the robot. Implement the missing parts in the file `motion_command.m` to compute the pose $\mathbf{x}_t$ given $\mathbf{x}_{t-1}$ and the odometry command $\mathbf{u}_t$. These vectors are in the following form:

$$\mathbf{x}_t = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \qquad \mathbf{u}_t = \begin{pmatrix} \delta_{rot1} \\ \delta_{trans} \\ \delta_{rot2} \end{pmatrix}$$

where δ_{rot1} is the first rotation command, δ_{trans} is the translation command, and δ_{rot2} is the second rotation command. The pose is represented by the 3×1 vector x in `motion_command.m`. The odometry values can be accessed from the struct u using

u.r1, *u.t*, and *u.r2* respectively.
Compute the new robot pose according to the following motion model:

$$x_t = x_{t-1} + \delta_{trans} \cos(\theta_{t-1} + \delta_{rot1})$$
$$y_t = y_{t-1} + \delta_{trans} \sin(\theta_{t-1} + \delta_{rot1})$$
$$\theta_t = \theta_{t-1} + \delta_{rot1} + \delta_{rot2}$$

Test your implementation by running the `main.m` script. The script will generate a plot of the new robot pose at each time step and save an image of it in the plots directory. While debugging, run the program only for a few steps by replacing the for-loop in `main.m` by something along the lines of for `t = 1:20`. You can generate an animation from the saved images using *VideoWriter*. You can use the following command to generate the animation from inside the plots directory:

```
writerObj = VideoWriter('odom.avi');
open(writerObj);
for k=1:330
    filename = sprintf('../plots/odom_%03d.png',k);
    thisimage = imread(filename);
    writeVideo(writerObj, thisimage);
end
close(writerObj);
```