Albert-Ludwigs-Universität Freiburg, Institut für Informatik
Prof. Wolfram Burgard, Dr. Daniel Büscher, Lukas Luft
Lecture: Robot Mapping
Winter term 2018/2019

Sheet 1

Topic: Octave

Due: November 8, 2018

General Notice

- The exercises should be solved in groups of two students.
- We will be using Octave for the programming exercises.

Octave is a command line program for solving numerical computations. The software is an open-source imitation of its commercial counterpart MATLAB and mostly compatible with it. The software is freely available from `www.octave.org`. It is compatible with Linux, Mac OS, and Windows. Install Octave on your system in order to solve the programming assignments. A quick guide to Octave is given in the Octave cheat sheet which is available on the website of this lecture.

Note that Octave is known to crash or produce faulty figures on Linux with proprietary Nvidia graphics card drivers. This can be prevented with the following command before starting Octave (tested on Ubuntu 16.04):

```
export LD_PRELOAD=/usr/lib/x86_64-linux-gnu/libOSMesa.so.6
```

If you rather want to use MATLAB, which is freely available to all students of the University of Freiburg, you are responsible for adapting the example code, if necessary.

Exercise 1: Get familiar with Octave

The purpose of this exercise is to familiarize yourself with Octave and learn basic commands and operations that you will need when solving the programming exercises throughout this course.

Go through the provided Octave cheat sheet and try out the different commands. As pointed out in the sheet, a very useful Octave command is `help <function>`. Use it to get information about the correct way to call any Octave function.

Exercise 2: Implement an odometry model

Implement an Octave 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 Octave 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:

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

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_model.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:

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

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 *avconv* (`sudo apt-get install libav-tools`) or *mencoder*. With *avconv* you can use the following command to generate the animation from inside the `plots` directory:

```
avconv -r 10 -i odom_%03d.png -b 500000 odom.mp4
```

Exercise 3: Homogeneous transformations

A robot pose in a given frame is compactly represented as:

$$\mathbf{p} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix}$$

Alternativeley, homogeneous coordinates can be used. This simplifies transformations as they are modeled in matrix form:

$$\mathbf{T} = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & x \\ \sin\theta & \cos\theta & y \\ 0 & 0 & 1 \end{pmatrix}$$

(a) Implement two Octave funtions:

- **v2t** takes as input the vector form of the robot pose and outputs the corresponding homogeneous transformation.
- **t2v** takes as input a homogeneous transformation representing the robot pose in 2-D space and outputs the corresponding compact vector.

Test your implementation by chaining together four different transformations of your choice.

(b) Given two robot poses $\mathbf{x}_1 = (x_1, y_1, \theta_1)^T$ and $\mathbf{x}_2 = (x_2, y_2, \theta_2)^T$, how do you get the relative transformation from $\mathbf{x}_1$ to $\mathbf{x}_2$?

(c) A 2-D point (which we will often call "landmark" or "point of interest") in the plane can be expressed by its x and y values. In order to get a 3×1 vector and to multiply it with 3×3 matrices representing the robot poses, we need to express the landmark in homogeneous coordinates. This means adding a scaling value of 1:

$$\mathbf{poi} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}.$$

Given a robot pose $\mathbf{x}_t$ and an $< x, y >$ observation $\mathbf{z}$ of a landmark relative to $\mathbf{x}_t$,

$$\mathbf{x}_t = \begin{pmatrix} 1 \\ 1 \\ \pi/2 \end{pmatrix}, \quad \mathbf{z} = \begin{pmatrix} 2 \\ 0 \end{pmatrix},$$

compute the location of the landmark.