Albert-Ludwigs-Universität Freiburg, Institut für Informatik
Dr. Gian Diego Tipaldi, Prof. Wolfram Burgard
Lecture: Robot Mapping
Winter term 2015

Sheet 1

Topics: Octave

Submission deadline: Oct. 26, 2015
Submit to: `robotmappingtutors@informatik.uni-freiburg.de`

General Notice

The exercises should be solved in groups of two students. In general, assignments will be published on Monday and should be submitted until the deadline. The source code of programming exercises should be submitted via email.

We will be using Octave for the programming exercises. Octave is a command line program for solving numerical computations. Octave is mostly compatible with MATLAB and is freely available from `www.octave.org`. It is available for 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.

Exercise 1: Getting 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. Ask for help by email whenever you need it. As pointed out in the sheet, a very useful Octave command is `help`. Use it to get information about the correct way to call any Octave function.

Exercise 2: Implementing 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 -b 500000 -i odom_%03d.png odom.mp4
```

Exercise 3: Homogeneous transformations

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

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

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

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

(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 an homogeneous transformation representing the robot pose in the 2D space and outputs the corresponding compact vector

Test your implementation chaining four different transformations of your choice.

(b) Given two robot poses $p_1 = (x_1, y_1, \theta_1)^T$ and $p_2 = (x_2, y_2, \theta_2)^T$, how do you get the relative transformation from p_1 to p_2?

(c) A 2D point (that 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 be able to make multiplications with the 3×3 matrices representing the robot poses, we need to express the landmarks in homogeneous coordinates, that in this case means putting the scaling value to 1.

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

Given a robot pose $p1$ and an $< x, y >$ observation z of a landmark relative to $p1$.

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

Compute the location of the landmark.