Albert-Ludwigs-Universität Freiburg
Institut für Informatik
Lecture: Robot Mapping
Winter term 2012
PD Dr. Cyrill Stachniss

Sheet 1

Topics: Octave

Submission deadline: Nov 5, 2012

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 on the following Monday before class at the latest. 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 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. You can generate an animation from the saved images using *ffmpeg* or *mencoder*. With *ffmpeg* you can use the following command to generate the animation from inside the `plots` directory:

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