

Albert-Ludwigs-Universität Freiburg, Institut für Informatik  
PD Dr. Cyrill Stachniss  
Lecture: Robot Mapping  
Winter term 2013

## Sheet 7

Topic: Grid Maps

Submission deadline: December, 16

Submit to: [robotmappingtutors@informatik.uni-freiburg.de](mailto:robotmappingtutors@informatik.uni-freiburg.de)

### Exercise: Occupancy Mapping Algorithm Implementation

Implement the occupancy grid mapping algorithm as presented in the lecture. To support this task, we provide a small *Octave* framework (see course website). The framework contains the following folders:

**data** contains the recorded laser scans and known robot poses at each time step.

**octave** contains the grid maps framework with stubs to complete.

**plots** this folder is used to store images.

The below mentioned tasks should be implemented inside the framework in the directory **octave** by completing the stubs:

- Implement the functions in `prob_to_log_odds.m` and `log_odds_to_prob.m` for converting between probability and log odds values.
- Implement the function in `world_to_map_coordinates.m` for converting the  $(x, y)$  world frame coordinates of a point to its corresponding coordinates in the grid map. You might find the *Octave* functions `ceil` and `floor` useful.
- Implement the function in `inv_sensor_model.m` to compute the update to the log odds value of each cell in the map for a particular laser scan measurement.

After implementing the missing parts, you can run the occupancy grid mapping framework. To do that, **change into the directory octave** and launch *Octave*. Type `gridmap` to start the main loop (this may take some time). The script will produce plots of the state of the resulting maps and save them in the **plots** directory. You can use the images for debugging and to generate an animation. For example, you can use `ffmpeg` from inside the **plots** directory as follows:

```
ffmpeg -r 10 -b 500000 -i gridmap_%03d.png gridmap.mp4
```

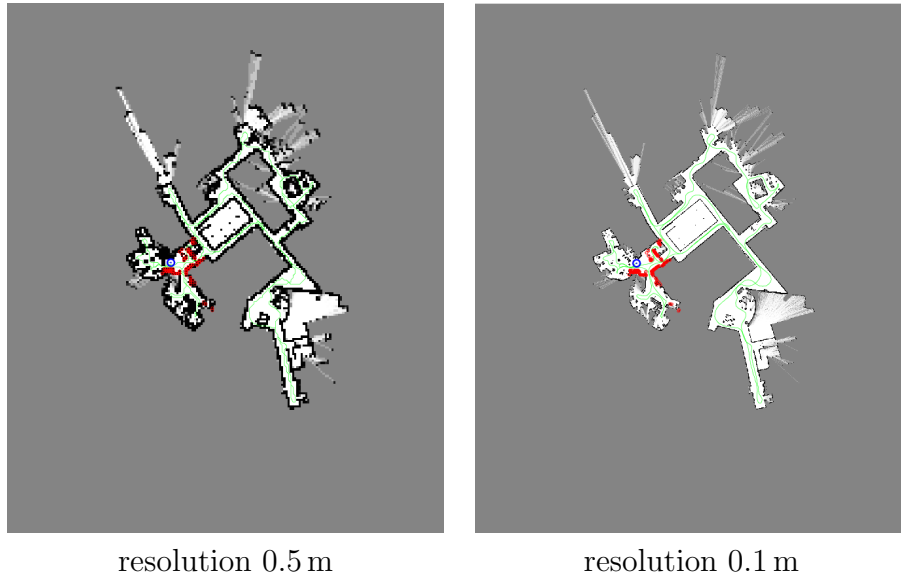


Figure 1: Examples for the final result of the occupancy mapping algorithm.

Figure 1 depicts the example images of the resulting maps using grid sizes of 0.5 m and 0.1 m.

Some implementation tips:

- Use an inverse sensor model corresponding to laser range finders (see lecture slides). The corresponding  $p_{free}$  and  $p_{occ}$  values are specified in the `gridmap.m` script. Use  $p_{occ}$  to update the occupancy value of cells that laser beam endpoints hit and  $p_{free}$  for all other cells along the beam. Use the function `robotlaser_as_cartesian.m` to compute the Cartesian coordinates of the endpoints of a laser scan. The provided `bresenham.m` function can be used for computing the cells that lie along a laser beam in map coordinates.
- Compute all occupancy value updates in log odds (not probabilities) so they can be added directly to the map.
- Test your implementation with a grid size of 0.5m. Once you are satisfied with your results, you can run the algorithm with an increased resolution (e.g. 0.1m), as this will take considerably more time.
- While debugging, run the algorithm only for a few steps by replacing the for-loop in `gridmap.m` by something along the lines of `for t = 1:10`.
- Many of the functions in *Octave* can handle matrices and compute values along the rows or columns of a matrix. Some useful functions that support this are `sum`, `log`, `sqrt`, `sin`, `cos`, and many others.

### Exercise: Particle Filter

- (a) Describe briefly the main differences between the particle filter and the Extended Kalman filter for state estimation.
- (b) Discuss briefly the advantages of the low variance re-sampling strategy.

### Exercise: Particle Filter Implementation

First, implement the prediction step of a particle by sampling the motion of a robot given the distribution  $p(x_t | u_{t-1}, x_{t-1})$  and second, implement the re-sampling step.

- (a) Implement the function in `prediction_step.m`, which samples a motion for each particle according to the motion model and the given noise parameters.
- (b) Implement the function in `resample.m`, which re-samples the set of particles utilizing the low variance re-sampling method.

To support this task, we provide a small *Octave* framework (see course website). The above-mentioned tasks should be implemented inside the framework in the directory `octave` by completing the stubs. After implementing the missing parts, you can test your solutions by running the script in `motion.m` for the prediction step and `resampling.m` for the re-sample step. The script `motion.m` will produce plots of the position of the particles and save them in the `plots` directory.

Some implementation tips:

- The function `normrnd( $\mu$ ,  $\sigma$ )` allows to draw samples from a Gaussian with mean  $\mu$  and standard deviation  $\sigma$ .
- The function `unifrnd( $a$ ,  $b$ )` generates random samples from the uniform distribution on  $[a, b]$ .
- Many of the functions in *Octave* can handle matrices and compute values along the rows or columns of a matrix. Some useful functions that support this are `sum`, `cumsum`, `sqrt`, `sin`, `cos`, and many others.