Albert-Ludwigs-Universität Freiburg
Lecture: Introduction to Mobile Robotics
Summer term 2015

Institut für Informatik

Prof. Dr. Wolfram Burgard
Dr. Gian Diego Tipaldi
Dr. Barbara Frank

Sheet 6

Topic: Discrete Filter, Particle Filter

Submission deadline: June 18, 2015

Submit to: `mobilerobotics@informatik.uni-freiburg.de`

Exercise 1: Discrete Filter

In this exercise you will be implementing a discrete Bayes filter accounting for the motion of a robot on a 1-D constrained world.
Assume that the robot lives in a world with 20 cells and is positioned on the 10th cell. The world is bounded, so the robot cannot move to outside of the specified area. Assume further that at each time step the robot can execute either a *move forward or a move backward* command. Unfortunately, the motion of the robot is subject to error, so if the robot executes an action it will sometimes fail. When the robot moves forward we know that the following might happen:

1. With a 25% chance the robot will not move
2. With a 50% chance the robot will move to the next cell
3. With a 25% chance the robot will move two cells forward
4. There is a 0% chance of the robot either moving in the wrong direction or more than two cells forwards

Assume the same model also when moving backward, just in the opposite direction.
Since the robot is living on a bounded world it is constrained by its limits, this changes the motion probabilities on the boundary cells, namely:

1. If the robot is located at the last cell and tries to move forward, it will stay at the same cell with a chance of 100%
2. If the robot is located at the second to last cell and tries to move forward, it will stay at the same cell with a chance of 25%, while it will move to the next cell with a chance of 75%

Again, assume the same model when moving backward, just in the opposite direction.
Implement in octave a discrete Bayes filter and estimate the final belief on the position of the robot after having executed 9 consecutive *move forward* commands and 3 consecutive *move backward* commands. Plot the resulting belief on the position of the robot.

Hints: Start from an initial belief of:

$$bel = [zeros(10, 1); 1; zeros(9, 1)]\,;$$

You can check if your implementation has bugs by noting that the belief needs to sum to one (within a very small error, due to the limited precision of the computer). If it doesnt there is a mistake.
Careful about the bounds in the world, those need to be handled ad-hoc.

General Notice

In the following exercises you will implement a complete particle filter. A framework containing the motion model and the measurement model is provided to you, so you can concentrate on the implementation of the filter itself. The following folders are contained in the tarball:

data This folder contains files representing the world definition and sensor readings used by the filter.

octave This folder contains the particle filter framework with stubs for you to complete.

plots The framework uses this folder to store images generated by the visualization.

To run the particle filter, change into the directory `octave` and launch the *Octave* program. Inside *Octave*, load `particle_filter.m` to start the particle filter. Running the particle filter may take some time. While the particle filter is running, plots visualizing the state of the filter are generated and stored in the `plots` directory. *Note: You first have to complete resample.m and mean position.m in order to get the filter working correctly.* We use the *librobotics* library for some of the visualization. All functions defined in the library are available in the framework.

Some implementation tips:

- Turn off the visualization to speed up the computation by commenting out the `plot_state(...` line in the file `particle_filter.m`.
- While debugging run the filter only for a few steps. You can do this by replacing the for-loop in `particle_filter.m` by something along the lines of `for t = 1:50`.
- The command `repmat` allows you to replicate a given matrix in many different ways and is magnitudes faster than using for-loops.
- When converting implementations containing for-loops into a vectorized form it often helps to draw the dimensions of the data involved on a sheet of paper.
- 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`, `sqrt`, and many others.

Exercise 2: Theoretical Considerations

(a) Particle filters use a set of weighted state hypotheses, which are called particles, to approximate the true state x_t of the robot at every time step t. Think of three different techniques to obtain a single state estimate $\bar{x}_t$ given a set of N weighted samples $S_t = \{\langle x_t^{[i]}, w_t^{[i]} \rangle \mid i = 1, \ldots, N\}$.

(b) How does the computational cost of the particle filter scale with the number of particles and the number of dimensions in the state vector of the particles? Why can a large dimensionality be a problem for particle filters in practice?

Exercise 3: Measurement Model and Resampling

A particle filter consists of three steps listed in the following:

(a) Sample new particle poses using the motion model.

(b) Compute weights for the new particles using the sensor model.

(c) Compute the new belief by sampling particles proportional to their weight with replacement.

The motion model (a) is already implemented in the provided framework. In this exercise you are asked to implement both (b) and (c).

(b) Complete the function file `measurement_model.m`. This function should implement the update step of a particle filter, using a *range-only* sensor.

It takes as input a set l of landmarks, a set z of *independent* landmark observations and a set x of particles.

- l: A struct array representing a landmark map of the environment, where each landmark $l(i)$ has an id $l(i).id$ and a position $l(i).x, l(i).y$.
- z: A struct array containing a number of landmark observations, where each observation $z(i)$ has an id $z(i).id$ and a range $z(i).range$.
- x: A matrix of size $N \times 3$, where N is the number of particles, $x(:, 1)$ represents the x-coordinate and $x(:, 2)$ the y-coordinate of each particle. The orientation $x(:, 3)$ is not used in this exercise, but will be of importance on the next exercise sheet where a complete particle filter is implemented.

It should return a vector of weights that has the same size as the number of particles. See slide 15 of the particle filter lecture for the definition of the weight w. Instead of computing a probability, it is sufficient to compute the likelihood $p(z|x, l)$. The measurement standard deviation is $\sigma_r = 0.2$. Try to avoid loops by using matrix operations where possible. *Hint*: The template for `measurement_model.m` already shows how to get a landmark with a certain id.

(c) Complete the function file `resample.m` by implementing stochastic universal sampling.

It takes as an input a set of particles to sample from particles in the same format as the aforementioned x and a set of weights w, which is a column vector containing a weight for each particle. The function should return a new set of particles, again, in the same format of x.

Exercise 4: Visualization

In Exercise 1 (a) you described three ways to obtain a single state estimate from the particle cloud. Implement one of these methods in the file `mean_position.m`, which is used to draw the pose of the robot in the visualization. *Note: You can use the function* `average_angle`. With *ffmpeg* you can use the following command to generate an animation from inside the `plots` folder:

```
ffmpeg -r 10 -b 500000 -i pf_%03d.png pf.mp4
```