ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institute for Dynamic Systems and Control
IDSC
Institut für Dynamische Systeme und Regelungstechnik

151-0566-00 **Recursive Estimation** (Spring 2021)

Programming Exercise #2 Topic: Particle Filtering

Issued: May 5, 2021 Due: **June 2, 2021**

Particle Filter to Track a Mobile Robot

Your task is to design a Particle Filter (PF) that tracks a mobile robot, which is moving in a closed room with a partially known contour. The coordinates of the robot are denoted by $(x_\mathrm{r}, y_\mathrm{r})$ and the heading by $\varphi \in [-\pi, \pi]$. The contour is defined by the points $\{p_1, \ldots, p_{10}\}$ and the uncertain wall offset κ given in Table 1. A distance sensor is mounted on the robot, pointing in the same direction as the robot is heading. This sensor measures the distance between the robot and the first opposing wall straight in front of the robot. The robot is controlled with a known input, which prevents the robot from driving into any wall. A sketch of the system is provided in Figure 1.

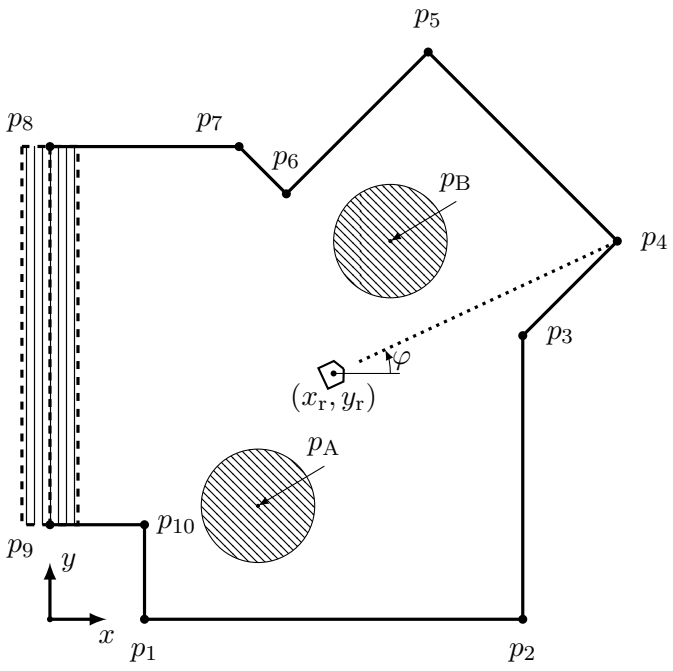

Figure 1: Sketch of the robot moving in a closed room with a partially known contour. The distance measurement is indicated by the dotted line between the robot and the opposing wall in front of it. The shaded circles denote the locations where the robot is initially located. The shaded rectangle on the left indicates the region where the unknown vertical wall is placed.

Point	x[m]	y[m]
p_1	0.5	0.0
p_2	2.5	0.0
p_3	2.5	1.5
p_4	3.0	2.0
p_5	2.0	3.0
p_6	1.25	2.25
p_7	1.0	2.5
p_8	κ	2.5
p_9	κ	0.5
p_{10}	0.5	0.5
p_A	1.1	0.6
p_B	1.8	2.0

Table 1: Coordinates of the contour points and center points of the initial distributions.

The position of the robot at time instant k is denoted by $(x_\mathrm{r}[k], y_\mathrm{r}[k])$, its heading by $\varphi[k]$, and the fixed (but unknown) x-coordinate of p_8 and p_9 by $\kappa[k]$. The dynamics are described by the following discrete time process model:

$$\begin{aligned} x_\mathrm{r}[k] =& x_\mathrm{r}[k-1] + (u_\mathrm{f}[k-1] + v_\mathrm{f}[k-1])\cos(\varphi[k-1]) \\ y_\mathrm{r}[k] =& y_\mathrm{r}[k-1] + (u_\mathrm{f}[k-1] + v_\mathrm{f}[k-1])\sin(\varphi[k-1]) \\ \varphi[k] =& \varphi[k-1] + u_\varphi[k-1] + v_\varphi[k-1] \\ \kappa[k] =& \kappa[k-1], \end{aligned}$$

where $u[k] = (u_\mathrm{f}[k], u_\varphi[k])$ is the known control input in the forward and angular directions at time instant k. The input $u_\mathrm{f}[k]$ may also be negative, which allows the robot to drive both forwards and backwards. The uniform process noise in the forward and angular directions is denoted by

$$v_\mathrm{f}[\cdot] \sim \mathcal{U}\left(-\frac{\sigma_\mathrm{f}}{2}, \frac{\sigma_\mathrm{f}}{2}\right), \quad v_\varphi[\cdot] \sim \mathcal{U}\left(-\frac{\sigma_\varphi}{2}, \frac{\sigma_\varphi}{2}\right),$$

with a known constant of σ_f and σ_φ.

The distance sensor mounted on the robot provides a measurement of the distance between the robot and the opposing wall at each discrete time instant, k. The distance measurement can be described by the following measurement equation:

$$z[k] = \sqrt{(x_\mathrm{r}[k] - x_\mathrm{c}[k])^2 + (y_\mathrm{r}[k] - y_\mathrm{c}[k])^2} + w[k], \tag{1}$$

where $(x_\mathrm{c}[k], y_\mathrm{c}[k])$ denotes the point of intersection of the distance sensor ray with the first opposing wall, which can be inferred from the position and heading of the robot and the contour of the room. You can assume that the robot always has sufficient distance to the wall. The measurement is corrupted by additive measurement noise $w[k]$ with the PDF as displayed in Figure 2. The known constant ϵ is in the range $[0, 0.02]$.

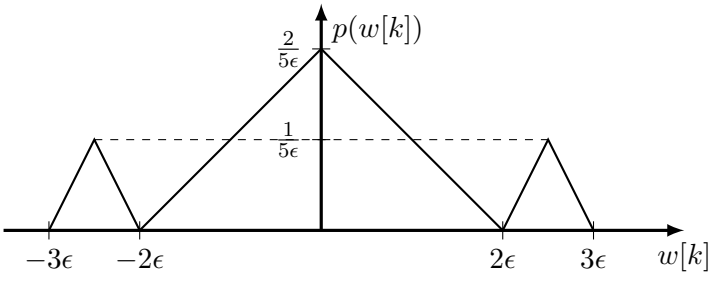

Figure 2: Probability density function of the measurement noise.

At the initial time sample $k = 0$, the robot is located at $(x_\mathrm{r}[0], y_\mathrm{r}[0]) = (x_0, y_0)$ with heading $\varphi[0] = \varphi_0$, and contour offset $\kappa[0] = \kappa_0$. The initial position (x_0, y_0) is uniformly distributed in the two shaded circles with centers located at p_A and p_B and radius d, as displayed in Figure 1. The known constant d is in the range $[0.1, 0.2]$. The initial heading φ_0 is uniformly distributed in $[-\frac{\pi}{4}, \frac{\pi}{4}]$, while κ_0 is uniformly distributed in $[-l, l]$, where the known constant l is in the range $[0.1, 0.2]$.

The random variables $x_0, y_0, \varphi_0, \kappa_0, \{v_\mathrm{f}[\cdot]\}, \{v_\varphi[\cdot]\}$ and $\{w[\cdot]\}$ are mutually independent.

Objective

The objective is to design a PF that estimates the location and heading of the robot, as well as the offset of the left wall from its nominal position. Your estimator will be called at every discrete time step k. The estimator has access to the previous posterior particles, the control inputs $u_{\mathrm{f}}[k-1], u_{\varphi}[k-1]$ and the measurement $z[k]$. Furthermore, the constants $\{p_1, \ldots, p_7, p_{8,y}, p_{9,y}, p_{10}\}$, p_{A}, p_{B}, d, l, σ_r, σ_{φ} and ϵ are known to the estimator. Note that $p_{8,y}$ and $p_{9,y}$ denote the y-coordinate of the points p_8 and p_9, respectively.

Provided Matlab Files

A set of Matlab files is provided on the class website.

`run.m`	Matlab function that is used to execute a simulation of the true system, run the estimator, plot the results, and report the root-mean squared tracking error.
`Estimator.m`	Matlab function template to be used for your implementation of the PF.
`Simulator.p`	Matlab function used to simulate the motion of the robot and measurements. This function is called by `run.m`, and is obfuscated (i.e. its source code is not readable).
`EstimatorConst.m`	Constants known to the estimator.
`SimulationConst.m`	Constants used for the simulation. These constants are *not* known to the estimator.

Task

Implement your solution for the PF in the file `Estimator.m`. Your code has to run with the Matlab script `run.m`. You *must* use *exactly* the function definition as given in the template `Estimator.m` for the implementation of your estimator.
The number of particles is not defined in the problem, and you should tune this number, together with a roughening method, to achieve an acceptable performance for your estimator. Furthermore, it is possible with the given measurement model that all your particles have zero measurement likelihood and therefore, all particle weights are zero as well. Your PF must handle this case appropriately.
The file `run.m` also outputs a performance measure based on a mean distance error (see the code for how it is computed).
The number of particles will affect the computation time of your implementation of the PF. The file `run.m` also outputs an average computation time for a single update of your PF. Your implementation should run with an average computation time for a single update below 0.3 seconds on the TA's laptop[1]. Points can be deducted for exceeding this value.
You are only allowed to use the basic MATLAB installation without any additional toolboxes. The `run.m` script will output an error whenever your estimator does not comply with this rule. In this case, the error message will contain the additional toolboxes on which your code depends. To determine whether a given function (e.g. the function `mean()`) depends on an additional toolbox, you may use the following instructions:

```
[~, pList] = matlab.codetools.requiredFilesAndProducts('mean');
{pList.Name}'
```

In case the function only depends on the basic MATLAB language (as is the case for the function `mean()`), the output should look as follows:

[1] Core i7 CPU running at 2.8GHz, with 16GB of RAM.

```
ans =
   {'MATLAB'}
```

While the style of your code is not relevant for evaluation, points will be deduced for severe violation of common sense programming techniques, which result for example in considerably increased computation times.
Note also that while there are built-in Matlab functionalities that provide solutions to find the intersection of the robot's distance sensor ray with the opposing wall (e.g. `polyshape` and `intersect`), their computational runtime may not be optimal for the given problem.

Evaluation

To evaluate your solution, we will test your PF on the given problem data. Moreover, we will make suitable modifications to the parameters in `EstimatorConst.m` and `SimulationConst.m` and test the robustness of your estimator for different values of the constants d, l, and ϵ for the ranges defined above. The constants $\{p_1, \ldots, p_7, p_{8,y}, p_{9,y}, p_{10}\}$, p_A, p_B, σ_r, and σ_φ are fixed and will not be modified during evaluation.

Deliverables

Your submission must follow the instructions reported in the `Deliverables.pdf` ***file provided, as grading is automated. Submissions that do not conform will have points deducted.***