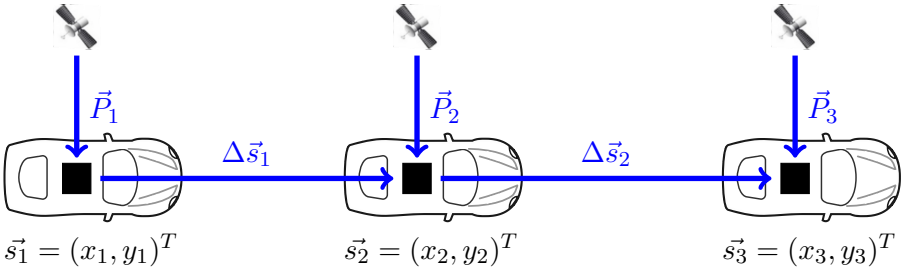
Lecture Automotive Vision 2019

Institut für Mess- und Regelungstechnik, Karlsruher Institut für Technologie
Dr. Martin Lauer, M.Sc. Annika Meyer, M.Sc. Danial Kamran

This excercise sheet covers the *particle filter* as an alternative state estimation algorithm to estimate the state and uncertainty of a dynamic system. The first task resembles the state estimation task from the previous excercise sheet using a particle filter in this case. The second task applies the particle filter to localize a robot on a soccer field.

Particle Filter

The objective of this exercise task is the implementation of a *particle filter* for the simplified simulated scenario to estimate the two-dimensional position of a vehicle that was introduced in assignment 4[1]. The state of the vehicle is expressed as $\vec{s}_k = (x_k,\ y_k)^T$ at the discrete time step k. The motion of the vehicle from time step k to $k+1$ is given by an odometer as $\Delta\vec{s}_k = (\Delta x_k,\ \Delta y_k)^T$ with uncertainty matrix Σ_k^s. A global positioning sensor measures the global position of the vehicle $P_k = (X_k,\ Y_k)$ with uncertainty Σ_k^p each point in time.



Exercise (5.1):

The Matlab script *pf_simple.m* implements the recursive state estimation for the described scenario above. It applies the functions `pf_init` for particle initialization, `pf_predict` for state prediction, `pf_update` for state innovation and `pf_resample` for particle resampling.

Implement the four functions `pf_init`, `pf_predict`, `pf_update` and `pf_resample`.

[1] For a reference of the Matlab variables, please refer to the excercise sheet of assignment 4.

The following auxiliary functions are provided:

`compute_particle_statistics`	This function computes the weighted mean and covariance of the particle set.
`simulate_particle_filter`	This function simulates the behavior of the particle filter graphically. For details, please refer to the comments in the file.

Also, please mind the useful Matlab functions listed in Table 3 at the end of this excercise sheet.

Exercise (5.2):

Test the particle filter with a varying number of particles. How does the numbers of particles influence the quality of estimation? What is a possible drawback of using a large number of particles?

Robot Localization

Now, we want to use a particle filter for a self-localization task in a more challenging scenario as in the previous example, namely for self-localization of an autonomous soccer playing robot. The robot shown in Figure 1 has a holonomic drive so that it can drive in any direction and turn itself at the same time. It is equipped with odometers which measure the present velocity and yaw rate of the robot, however, they are highly affected by noise and imprecision. Furthermore, the robot is equipped with an omnidirectional camera that allows it to perceive the soccer field and detect points on the white field markings. The geometry of the soccer field and the field markings are known.

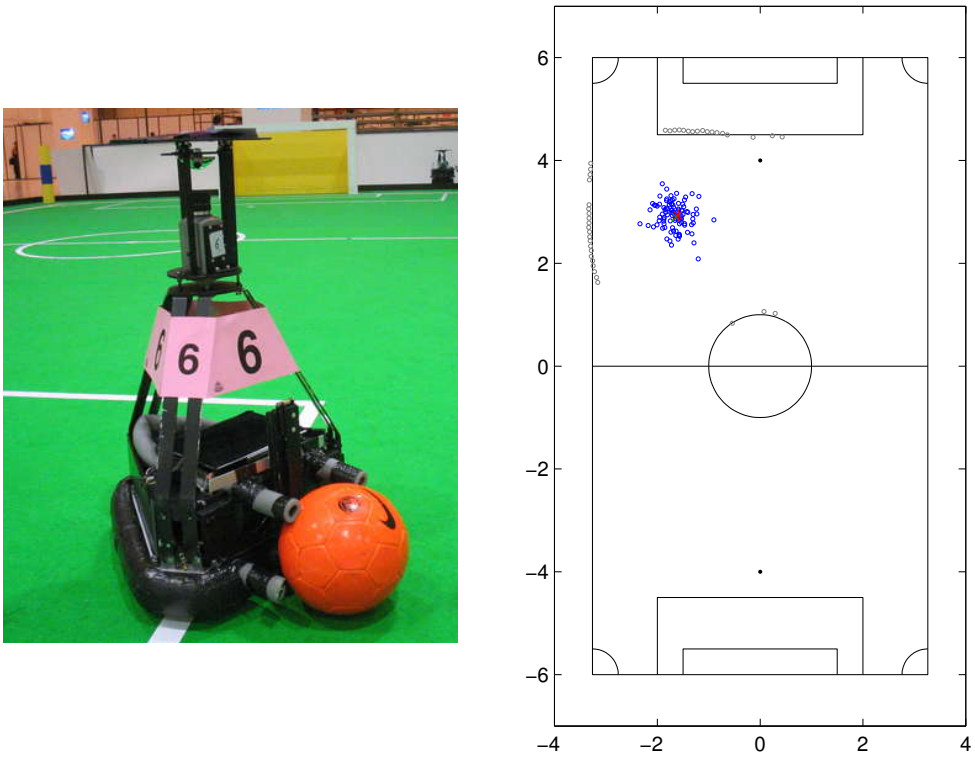


Figure 1: Left: Picture of a soccer robot. Right: Screenshot from the visualization of the particle filter for robot localization. The black lines represent the field markings, the blue circles represent the positions that are represented by the particles of the filter. The green cross illustrates the position which was calculated as the weighted mean of the particles and which is supposed to be the estimated robot position. The red asterisk represents the reference position. The gray circles illustrate the observed points on field markings if we assume the weighted mean of the particles as the true pose of the robot.

Exercise (5.3):

Implement a particle filter that solves the self-localization task for the soccer robot. The basic idea is to use the odometer values of the robot in the prediction step of the filter and the observed points on field markings in the innovation step. We assume that the noise of the odometers follows a Gaussian distribution with certain standard deviations σ_x and σ_y for the translation error and σ_ϕ for the error in the yaw angle. Furthermore, we assume that the distance between the observed position of a field marking and the closest field marking also follows a Gaussian distribution with a certain standard deviation σ_{obs}.

The file *rc_particle_filter.m* contains an incomplete implementation of an appropriate particle filter including visualization tools. The incomplete sub functions `pf_predict` and `pf_innovate` should implement the prediction step and the innovation step of the particle filter, respectively. Complete them following the ideas described above and test your implementation! You might make use of the function `mapdistance` that calculates for a point (x, y) on the soccer field the distance to the closest field marking.

The filter initializes the particles by random sampling from a uniform distribution over one half of the soccer field. The function `rc_particle_filter` also visualizes the present state of the particle filter with top view plots as depicted in Figure 1. Mind the useful Matlab functions listed in Table 3.

You find test data in the file *rwalk.mat*. It contains one long data sequence (L) and seven shorter subsequences (L1–L7). For each point in time when a measurement was made the sequences contain a struct with elements that are described in table 1. The struct `field_geometry` describes the geometry of the soccer field (e.g. length and width of the field).

`time`	The point in time when the measurement was taken (in seconds).
`observation`	The set of points that have been observed. Each row represents one observed point. The two columns describe the x- and y-coordinate of the observed point in a robocentric coordinate system (length unit: meters).
`odometry`	The first and second entry contain the sensed velocity of the robot in x- and y-direction (physical unit: $\frac{m}{s}$). The third entry contains the yaw rate (unit: $\frac{rad}{s}$).
`reference`	The position and yaw angle of the robot that were determined using a more elaborated self-localization technique and which are used as ground truth to assess the quality of the particle filter solution. Do not use the reference data in your particle filter implementation.

Table 1: Elements of the struct that contains the measurements and reference poses for the robot localization task.

Exercise (5.4):

Vary the parameters of the particle filter and observe its behavior. Is it able to find the correct robot position? How long does it take until the robot has found its position? How does the tracking accuracy vary? How much scattered is the particle cloud? Which effect do varying sizes of the particle set have on the computation time? Start with the parameter set that is provided in Table 2.

`num_particles`	100	The size of the particle set.
`num_uniform`	0	See description of the next sub task, keep it zero in this sub task.
`sigma_xy`	0.1	The measurement noise of the odometry in translation.
`sigma_yaw`	0.1	The measurement noise of the odometry in angle
`sigma_observation`	0.3	The measurement noise of the detected points on the field markings.

Table 2: Parameters for the soccer robot experiments. Start with those values, observe the behavior of the particle filter, and vary them.

Exercise (5.5):

The function `rc_particle_filter` allows to change the standard behavior of the particle filter by setting the parameter *num_uniform* to a value larger than zero. If this parameter is unequal to zero, only `num_particles - num_uniform` particles are resampled from the present particle set in the resampling step while the remaining *num_uniform* particles are generated randomly from a uniform distribution over one half of the soccer field. This should enable the filter to relocalize if the tracking has failed and the robot has lost its position or if the true position has not yet been found during initialization.

Test the behavior of the particle filter for different values of *num_uniform*. How does an increase of *num_uniform* affect the ability to localize itself? How does the tracking accuracy change?

Exercise (5.6):

While the data in file *rwalk.mat* have been recorded under lab conditions in which the robot was driving smoothly without interaction with other robots, real games look differently. The robot collides with other robots so that it is turned or shifted unexpectedly or it changes its velocity so quickly that it tilts which causes erroneous landmark measurements. Moreover, the soccer field is twice as large. The file `attacker.mat` contains data that have been recorded on a robot during a soccer game.

Try your particle filter solution on this data set and check whether your solution still works properly. Check the runtime of your approach. The camera provides 30 images per second. Would your approach be able to be processed in real time?

`mvnrnd`	This function generate random variates from a multivariate normal distribution with given expectation value and covariance matrix. Mind the fact that this function generates row vectors and the expectation value also needs to be provided as row vector.
`mvnpdf`	This function evaluates the probability density function of a multivariate normal distribution with given expectation value and covariance matrix. Mind the fact that this function takes row vectors as input.
`mnrnd`	This function generates variates of a multinomial probability distribution which can be used to simulate random sampling with replacement (n is the number of repetitions and p is a vector with selection probabilities).
`normrnd`	This function generates random variates from a uniform normal distribution
`normpdf`	This function evaluates the probability density function of a univariate normal distribution

Table 3: Table of Matlab function which might be useful for this assignment. See the help function of Matlab for further details.