# Assignment: Adaptive Population Size Using KLD Sampling

August 12, 2010

## 1 Introduction

For a particle filter to effectively find and track the position of the robot, it needs to have a sufficient amount of particles. The problem is that this sufficient amount depends on the situation. Initially, for instance, when the position of the robot is unknown, a high number of particles is needed to cover the whole environment. Once the robot position is found, fewer particles are needed. Also when the particle filter is faced with an ambiguous situation, more particles are needed to track the solutions.

In [Kootstra and de Boer, 2009], a method called local selection is introduced. This method lets particles 'reproduce' when the fitness (probability) of those particles is high. However, when the fitness is low, particles will be taken out of the population. This makes that the number of particles is dependent on the situation.

Another method to adapt the population size is proposed in [Fox, 2003]. There the Kullback-Leibner distance is used to measure the spread of particles in the population. Based on the spread, more or less particles will be used in the next time step. This method is called KLD-sampling. In this exercise, you will implement KLD-sampling and compare it to local selection.

## 2 Exercise

First read the paper [Fox, 2003] in `fox03kld_sampling.pdf`. Don't focus too much on the mathematics, but focus on the practical implementation. Table 2 at page 14 gives the pseudo-code for KLD-sampling.

In the exercise folder, you will find the full Java code for the particle filter plus diversity-maintenance methods. At the bottom-right corner you can add values for T. This is the threshold used in the local-selection method. The value is used to initialize the amount of energy of every particle. But more importantly it is the threshold on the amount of energy for reproduction of a particle. If the energy of a particle exceeds the threshold, a duplicate of the particle will be placed in the map. If on the other hand the energy

drops below zero, the particle will be erased from the population. In this way, the size of the particle population adapts to the environment. Start the code and look at the behavior of the local-selection method.

We will have a quick look at the code, to point out some elements that you will need for the implementation.

- The function `resampleUsingSUS()` (line 1115) is used for resampling the particle population based on the weights. The variable `nrSamples` gives the number of particles that will end up in the new particle population. In the function this is set to `nrHypotheses`. For KLD-sampling, you want to make a copy of the function with the difference that the number of samples are set using the Kullback-Leibner distance.

- For implementing an adaptive population size, look at the function `applyLocalSelection()` (line 1273), which implements the local-selection method. Here the `newHypotheses` is first emptied in line 1284 and then new particles are added to the new population in, for instance, line 1301. You should do a similar thing for the KLD-sampling.

- To add a new niching method to the selection list. Add another selection item after line 200, and add another static variable `NICHING_KLD` after line 2157. You can then use `if(nichingMethod==Vehicle.NICHING_KLD)` in the code to place the code specific for KLD-sampling.

- The main loop of the particle filter is in the function `nextCycle()` (line 1337). Since the main-loop as described in table 2 is a bit different, you will have to add code to the function to execute in case the KLD method is selected as niching method. Use an if-then-else statement around lines 1342-1347 and implement the KLD sampling. Lines 1350-1365 should always be executed, no matter which niching method.

- You will select one particle at the time from the current set of particles. You can use `sampleUsingSUS(1)` to get a weight-proportional sample. Due to the different order in the KLD-method (first sampling, then applying the transition and sensor model), the will be a problem at the first iteration, since the probabilities of the particles are not yet set. Create therefore an initialization function that sets the probabilities of every particle to 1.0 using `((Hypothesis)hypotheses.get(i)).setProbability(p)` and fills the array `cumulativeP` accordingly with the cumulative probabilities ($[1.0, 2.0, \ldots]$).

- Instead of applying the motion and sensor model to all particles at once, you will have to apply it for the individual particles. Look at the functions `applyMotionModel` (line 1309) and `applySensorModel` (line 1330).

- Implement the method using the pointers given above. You can use $z_{1-d} = 1.65$ for 95% confidence.

# References

[Fox, 2003] Fox, D. (2003). Adapting the sample size in particle filters through kld-sampling. *International Journal of Robotics Research*, 22(12):985–1004.

[Kootstra and de Boer, 2009] Kootstra, G. and de Boer, B. (2009). Tackling the premature convergence problem in monte-carlo localization. *Robotics and Autonomous Systems*, 57(11):1107–1118.