Exercise: Diversity Preservation in Monte-Carlo Localization

August 11, 2010

1 Introduction

Monte-Carlo Localization (MCL) is a probabilistic method to estimate the position of a robot based on the robot's sensors readings and its odometry. MCL uses a Particle Filter for the estimation. A particle filter represents the probability distribution by a set of particles. The density of particles represents the probability. In Monte-Carlo Localization, a map of the environment is known. The robot's sensor information is used to determine the most likely position in the map.

A Participle Filter shows remarkable similarities with a Genetic Algorithm: The fitness of every participle is determined by the resemblance of the robot's observation and the expected observation, the fittest particles survive and reproduce, and mutation is applied in the form of noise on the transition model.

Initially, when nothing is known about the position of the robot, the particles are randomly distributed over the map. Every particle has a x- and y-position and an orientation. The position of the robot is then estimated by iteratively going through three steps:

- Step 1: The robot's displacement since the last time step is measured using its odometry. The same transition (position and orientation) is applied to all the particles, with the addition of translational and rotational noise.
- Step 2: The probability of every particle is calculated by comparing the robot's sensory observation to the expected observation. This expected observation is calculated for every particle using the map of the environment. This is the observation that the robot is expected to make when being at the position of the particle. Some uncertainty in the sensor model is taken into account.
- Step 3: The particle population is resampled based on the probabilities of the particles. Particles with higher probabilities have a higher chance to end up in the new population. Improbable particles will likely by erased from the population.

However, there is a problem with the standard Particle Filter and therefore with Monte-Carlo Localization. This is the problem of premature convergence in the case of ambiguous situation, and thus multiple possible solutions to the estimation problem. Due to genetic or random drift, the particle population will quickly converge to one of the solutions. Since this solution might be the incorrect solution, this premature convergence is not desirable. The genetic drift occurs because of the randomness in the resampling function and the competition between particles of different solutions due to the limited number of resources, that is, the limited number of particles in the population. This causes that particles in one niche can replace particles in another niche, therefore weakening the other niche.

Within the fields of Theoretical Biology and Genetic Algorithms, the phenomenon of genetic drift has been studied. This has resulted in a number of so-called *niching methods* to preserve the diversity in the particle population. The key idea of the niching method is to prevent competition between particles of different solutions (in biology called niches) or to add a balancing force that counteracts the random drift. In [Kootstra and de Boer, 2009], the application of the niching methods to solve the problem of premature convergence in MCL has been studied.

2 Exercise

The Java program `Localization.class` in the directory `exercise_2` implements a simple simulator of a robot driving through an environment. The position of the robot is estimated using Monte-Carlo Localization. The program can be run with `java Localization`. In `manual.html` you can read a short description of the different inputs and buttons of the program.

- Use the ambiguous world without a niching method and run the Monte-Carlo Localization (MCL). Although the world is highly symmetric and provides four ambiguous situations, the particle filter will soon loose one of the solutions, and will eventually converge to a single solution. The reason for this is the random drift, caused by the randomness in the sampling process.
- Restart the simulation with different number of particles in the population. See what happens
- Randomly distributing a proportion of the particles over the environment is a good method to recover from a kidnap (try!). It is sometimes proposed as a method to preserve diversity in the population. See if that is the case.
- To truly preserve diversity, a number of niching methods have been proposed [Kootstra and de Boer, 2009]. One of the niching methods is implemented in this version of the simulation, the closest-of-the-worst method. It replaces the normal resampling method with a method that tries to avoid competition between different niches (solutions). It is doing that by biasing towards replacing nearby particles. Run the MCL with this niching method. Are the four solutions maintained?

- The method has a problem with *free particles*, particles that are not considered for replacement, although their probability is low. Look at the algorithm (either in the paper on in the function `resampleUsingClosestOfTheWorst()` in `Localization.java` (line 990)), and try to explain the free particles. Also suggest a solution to the problem.

You will now implement another niching method to preserve the diversity. Both sharing and frequency dependent selection alter the original probability value so that smaller groups have a fitness advantage. This will produce a balancing force that eliminates the genetic drift. The fitness advantage is based on the distance between the particles. Particles that have a low average distance are likely to be in a larger group and therefore get a fitness disadvantage. Particles with a high average distance are more isolated and therefore receive an advantage.

- Open `Localization.java`.
- The class `Hypotheses` implements the particle filter (a particle is sometimes termed an hypothesis). The main cycle is in the function `nextCycle()` (line 1082). It consists of first applying the transition model to the particles to move the particles in accordance with the robot. Next the sensor model is applied to calculate the probabilities of every particle based on the robot's sonar readings. Finally the particles are resampled. The probabilities are calculated in `getProbabilities()` (line 856).
- We will implement frequency dependent selection to solve the premature convergence of the particle filter. Read the about it in the method section in the [Kootstra and de Boer, 2009] (`kootstra09ras.pdf` in the exercise directory).
- We will Implement the frequency-dependent selection method in the function `applyMyNichingMethod()` (line 848). First calculate the distance of every particle to every other particle. You can use `((Hypothesis)hypotheses.get(i)).pos` to get the particles position. This will give a `Position` object with fields `x`, `y`, and `a`. You can compile the code with `javac Localization.java`.
- Determining the sharing of a particle by considering the distance to all other particles results in a quadratic complexity of the algorithm with respect to the number of particles. Good results can however be obtained by estimating the sharing by only considering the distance to a random subset of all the particles. Experiment with the size of the subset and see what gives good diversity preservation while being computationally efficient.
- Load the 'complex' world and see how the particle filter with niching method behaves. The diversity maintaining force can result in a *ghost cluster*. This is not desirable in non-symmetric environments. However, you can see that the ghost clusters are not very stable over time, because of inconsistencies with the robot's observations.

- Design and implement a method to deal with the ghost clusters. Make sure that the desirable diversity-maintaining abilities in the symmetrical world don't go lost.

References

[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.