Kidnapped Vehicle Project

1. Introduction

In this project, we estimate the position of an object by means of a particle filter. As in any Bayesian filter, the filtering process consist of two basic steps:

1. Prediction: in each time step we apply motion provided by the sensor and assuming a bicycle motion model.
2. Update: we compute the posterior probability by measuring how well does the prediction match the map landmarks (this is an *a priori* information).

In the next section, I will briefly describe some of the implementation details.

2. Implementation

I have followed the Udacity lecture in order to implement the particle filter. Namely, the following function have been implemented:

- *init()* : It initializes the particles and sets their position according to the initial GPS reading. This position is drawn from a pdf (probability density function) given by the reading itself and the measurement error of the sensor (GPS in this case).
- *prediction()* : It applies the estimated motion to each of the particles. Bicycle motion model is assumed and the motion measurement are assumed to be normally distributed.
- *findNearestNeighbour()* : It performs data association (via nearest neighbour). For each observation (after transforming its coordinates to the global basis) we find the nearest landmark. However, several observations can share the same nearest landmark. In order to avoid such duplicities (or multiplicities), we perform the so called cross-check approach. This means, that for each observation we select its nearest landmark and we also check that this observation is the nearest (among all of them) to the selected landmark.
- *updateWeights()* : It updates the weights of each particle according to how well their observations match the available map landmarks.
- *resample()* : This is the actual update step where particles with low weights (poor landmark matching) are more likely to be discarded and particles with high weights (good landmark matching) are likely to increase their relative population.

All the implementations are located in the *src* folder in the files *particle_filter.h* and *particle_filter.cpp*.

3. Results

The code meets the specifications as indicated by the simulator. Currently, the number of particle is set to 100. Nevertheless, the specifications are also met with much lower number of particles. For instance, the filter passes the requirements with 10 particle only even though the corresponding estimation error is higher than using 100 particles. On the other hand, the lower the number of particles the lower the computational burden. It is a pity that we do not have access to the ground truth data since it would be interesting to see the evolution of the estimation error with respect to the number of particles (done in an automated way and not by hand).

4. Discussion

In this project we have seen the performance of a particle filter. The filtering process is relatively simple yet it yields high quality results.

Finally, and as a side note, I would like to express my concerns regarding the documentation and the instructions provided in this project. It is very deficient with respect to the previous ones. One example (out of many):

SetAssociations() function:

- Its return type is set as Particle but the function does not return anything (it actually issues a compilation warning).
- It is here where it is explained what the parameter *associations* means (and not in the *Particle* class).
- Idem for *sense_x*, *sense_y* which, in addition, is not very clear what they are supposed to represent.