CSE598C Project1 Ideas Fall 2012

As discussed in class, particle filtering is a method for implementing Bayes filtering for more general distributions than Kalman filtering (for example) can handle. The goal of this project is to make the concepts involved in implementing an SIR particle filter more clear and concrete.

I have written a very simple particle filter implementation called samplePFcode.m to get you started. The code performs SIR filtering to track a single target, demonstrated on a soccer video sequence. Motion prediction is based on a "constant position + noise" motion model, meaning state vector is just [x,y] location with no velocity variables. This simple model is the same as assuming zero velocity, i.e. objects don't move. Of course, objects DO move, and therefore the noise model must do all the work to handle object motion. For observations, I am using ground truth bounding boxes that are supplied with the sample dataset to simulate the output of a (perfect) object detector. Again, it is an unreasonable assumption, but something to get you started. Initialization is done by clicking on an object in the first frame. Look at the code and run it several times to get a feel for what it does (we will also do this in class). Also dive into the code, line by line, to make sure you understand how the underlying SIR algorithm is being implemented here. Play around with the standard deviation sigma values for the different noise terms to see what the effect is when you increase and decrease their values.

What I want you to do for the project is to make the tracker better! I have given a few ideas below, but you are welcome to come up with your own ideas.

1) Add a more sophisticated motion prediction model, for example the "constant velocity + noise" model we discussed in class. Note that each particle state vector must now include a velocity term. Find some cases to show where your version with constant velocity motion model works better than my version using constant position.

2) Make the code halt gracefully when the object being tracked leaves the field of view. That is, determine how you might know when the object is not being tracked anymore, perhaps by looking at the magnitudes of the highest likelihood scores.

3) In practice, you won't have a perfect object detector! Simulate a real detector by adding noise to ground truth bounding box positions before you use them, and also by adding new random bounding boxes (to simulate false positives) and deleting some bounding boxes at random (to simulate false negatives). Try pushing the limits of how much you can mess up the detection data before your tracker starts to show a noticeable degradation in performance.

4) One of the sample files I have given you, soccerboxesusage.m, has some code that demonstrates how to do background subtraction on this dataset. Modify the PF tracker to use foreground/background binary masks to compute observation likelihoods, which will result in a much more practical tracker that you could easily use on other video data taken from a stationary camera. You will have to play around a little to determine what a good

likelihood function would be. One idea is to keep around the width and height information about the bounding box of the object you are tracking. Then, to evaluate a sample at location (x,y), examine the rectangular region centered at (x,y) and count how many pixels are on (foreground pixels) in that box vs off (background pixels). We would expect the likelihood should be higher if there is a higher proportion of foreground pixels. If you wanted to, you could also try adding bounding box width and height to the state vector, along with a "motion" model for each of them, so that the tracker can modify size and shape of the bounding box over time. There is room for significant creativity here!

5) Take a serious look at the Aralampulam tutorial paper and choose one or more other variants of particle filtering that look appealing to you, and modify the code to implement those algorithms. Ideally, you would be able to show that the new algorithm works significantly better than my simple version.

Finally, other project ideas related to particle filtering are also welcome. For example, I have put a pdf of a chapter on "Tracking with Non-Linear Dynamic Models" online. Section 2.2 has a VERY good description of both SIS and SIR particle filters. It also shows experiments comparing SIS and SIR with the Kalman filter, on an example where the Kalman filter can compute the optimal answer against which to evaluate the mean and variance estimates produced by SIS and SIR over time. [This is very similar to the experiment that I kept referring to in lecture as something that would be interesting to try!] So another project idea is to implement SIS, SIR and Kalman filtering for the example given in the book, and to try to reproduce their experimental results.