CIS 390 Fall 2015
Robotics: Planning and Perception
Kostas Daniilidis

Project 2 Phase 1 Equations

1 Introduction

Here we will briefly go over the equations for the particle filter

2 Input to the System

As before with the Kalman filter we have our three dimensional state vector $\mathbf{x}^t$

$$\mathbf{x}^t = \begin{pmatrix} x^t \\ y^t \\ \theta^t \end{pmatrix}$$

We are also given a map of the world, which consists of obsacles and April Tag positions and orientations. For the purposes of state estimtion we will ignore the obstacles and only consider the April Tags as a set of tuples, say m of them:

$$W = \{(x_i, y_i, \theta_i, d_i)\}_{i=1}^m$$

Where x_i, y_i, and θ_i give the position and orientation and d_i gives the ID of the tag. Note that the d_i may not be distinct for different i.

3 Particle Filtering

3.1 Initializing particles

Unlike the Kalman filter, we do not assume that we move with simple linear equations and instead we use the full model. However we need to keep an estimate of the ambiguity or the error of our estimate, and so we use particles. First you create n particles distributed with some initial distribution over the (x, y, θ). This could be uniformly at random over a region, or normally distributed with an expected mean and covariances. Initially the points are weighted equally. The number of particles n you choose is up to you - the more you have the more accurate it will be but the more computational burden will be put on your machine.

3.2 Propogating particles

Now that we have our initial particles in a set $\{p_i = (x_i, y_i, \theta_i)\}_{i=1}^n$, we can propogate them. Since we are storing particles we can propogate them in a nonlinear fashion using the particle's current position and the control input, using the equation

$$f(\mathbf{x}^t, \mathbf{u}^t) = \begin{pmatrix} x_t + v\Delta t \cos\theta_t \\ y_t + v\Delta t \sin\theta_t \\ \theta_t + \omega\Delta t \end{pmatrix} \tag{1}$$

Where $\mathbf{u}^t = (v, \omega)^T$ is the commanded velocity and the commanded angular velocity.

3.3 Reweighting particles

Once propogated we need to compute the weights for the points. We can measure how accurate the point after we get measurements of the tags $\{(x_i, y_i, \theta_i, j_i)\}$ for some k we see. We use the following pseudocode to compute the weights for a particle of with position (X, Y, Θ):

```
function GetWeight($X, Y, \Theta$)
    Initialize weights $w_i$ to zero
    for all $(x_i, y_i, \theta_i, d_i)$ in visible tags do
        // Find the closest match to the tag we see
        for all $(x_j, y_j, \theta_j, d_j)$ in $W$ where $d_j = d_i$ do
            Let $(\hat{x}_j, \hat{y}_j, \hat{\theta}_j)$ be the $(x_j, y_j, \theta_j)$ in the particle frame
            $w_i \leftarrow \max(w_i, p_N(x_j - x_i, y_j - y_i, \theta_j - \theta_i))$ // Pick the more likely one
        end for
    end for
    Return $\prod_i w_i$
end function
```

Where the likelihood function p_N is the pdf of a normal:

$$p_N(x, y, \theta) = \exp\left(-(ax^2 + by^2 + c\theta^2)/2\right)$$

Do this for all the particles to find the weights, then normalize so that the weights sum to one.

3.4 Resampling

When points get very low in weight, it is no longer worth keeping them. Therefore we resample them on occation after reweighting them. Though there are better resampling methods, we will use a "roulette wheel" model to resample, for simplicity. The idea is to resample based on the weights by picking with probability proportional to the weights. Given the weights of the particles $\{w_i\}$ the algorithm is as follows:

```
function Resample($\{w_i\}_{i=1}^n$, $\{p_i = (x_i, y_i, \theta_i)\}_{i=1}^n$)
    Let $S$ be the set of new particles (initialized as empty)
    Let $c_j = \sum_{k=1}^{j} w_k$ // Compute the cumulative distribution, with $c_n = 1$
    for all $i$ from 1 to $n$ do
        Pick $u$ uniformly at random from 0 to 1
        Find the largest $j$ such that $c_j \leq u$
        Place particle $p_j$ into $S$
    end for
    Return $S$
end function
```