Particle Filter Tutorial

Carlos Esteves and Daphne Ippolito

November 3, 2016
Outline

1 Introduction
2 Prediction
3 Update
4 Resample
Outline

Introduction

Prediction

Update

Resample
Introduction

- The Following material is from Ioannis Rekleitis "A Particle Filter Tutorial for Mobile Robot Localization"
Outline

1. Introduction

2. Prediction

3. Update

4. Resample
model motion step as rotation, followed by a translation

Figure 1: Arbitrary motion $[\Delta x, \Delta y]^T$ of robot $R_i$. At time $t = k - 1$ the pose is $[x, y, \theta]^T$, after the motion at time $t = k$ the pose is $[x', y', \theta_k]^T$. The robot first rotates to orientation $\theta_k$ and then translates by $\rho_k$. 
Rotation model

\[ \hat{\theta}_{k+1} = \hat{\theta}_k + \delta\hat{\theta} + N(\mu_{\text{rot}}, \sigma_{\text{rot}}\delta\hat{\theta}) \]
Translation model

- Translation is more complicated, because each translation step may also introduce rotation errors.
- So we model a translation of $\rho$ in $K$ steps, and add orientation noise before and after each step.

```
Input: Set of $M$ Particles: $S$; Translation distance: $\rho$
$\delta \rho = \frac{\rho}{K}$
for (j = 1 to M) do { For each particle}
  for (k = 1 to K) do { At each of K steps}
    $E_{trs} = \text{rand\_N}(M_{trs} \ast \delta \rho, \sigma_{trs} \ast \delta \rho)$;
    $E_{drft} = \text{rand\_N}(M_{drft} \ast \delta \rho, \sigma_{drft} \ast \delta \rho)$;
    $\hat{\theta}_j = \hat{\theta}_j + E_{drft}$;
    $x[j] = x[j] + (\delta \rho + E_{trs} \ast \cos(\hat{\theta}_j))$;
    $y[j] = y[j] + (\delta \rho + E_{trs} \ast \sin(\hat{\theta}_j))$;
    $E_{drft} = \text{rand\_N}(M_{drft} \ast \delta \rho, \sigma_{drft} \ast \delta \rho)$;
    $\hat{\theta}_j = \hat{\theta}_j + E_{drft}$;
  end for
  $S'[j] = [x[j], y[j], \hat{\theta}_j]^T$;
end for
Return($S'$)
```

**Algorithm 2:** Forward Translation with Noise; $\text{rand\_N}(M, \sigma)$ is a pseudo-random number generator drawing samples from a Normal distribution with mean $M$ and standard deviation $\sigma$; procedures are noted as underlined text, Comments are inside curly brackets “{comment}”. The variables $M_{trs}$ and $M_{drft}$ represent the mean error and are experimentally derived.
Figure 2: The effect of $\sigma_{tr,s}, \sigma_{dr,ft}$ for the forward translation: (a) $\sigma_{tr,s} = 5\text{cm/m}, \sigma_{dr,ft} = 1^\circ/\text{m}$ (b) $\sigma_{tr,s} = 1\text{cm/m}, \sigma_{dr,ft} = 5^\circ/\text{m}$.
How to estimate the standard deviations?

- compare odometry with laser range finder

Figure 9: Measuring the odometry error on carpet.
Error in rotation (speed and angle)

Figure 11: Error in rotation relative to the odometer for different angles and for different speeds (“o” speed 10, “x” speed 50, “+” speed 90, lines connect the mean values).
Error in rotation (different surfaces)

Figure 13: Error distribution from the odometry measurement for different surfaces (rotation of 90°).
Figure 16: Error distribution after translation of 120cm. Plastic surface. 43 samples.
function predict(pf, odo)

    % Straightforward code:
    %
    % for i=1:pf.nparticles
    %     x = pf.robot.f( pf.x(i,:), odo)' + sqrt(pf.Q)*pf.randn(3,1);
    %     x(3) = angdiff(x(3));
    %     pf.x(i,:) = x;
    %
    % Vectorized code:

    randvec = pf.randnn(pf.nparticles,3);
    pf.x = pf.robot.f( pf.x, odo) + randvec*sqrt(pf.Q);
    pf.x(:,3) = angdiff(pf.x(:,3));
end
Outline

1. Introduction
2. Prediction
3. Update
4. Resample
note that in our case the sensor is aligned with the robot orientation \( \hat{\theta} = 0 \)
and the "observed robot" is an April tag

Figure 7: Observation.
Update pose estimate and weights

- pose from measurement

\[
x_{\text{mest}}(k + 1) = \begin{bmatrix} x_{\text{mest}} \\ y_{\text{mest}} \\ \hat{\theta}_{\text{mest}} \end{bmatrix} = \begin{bmatrix} x_s + \rho \cos(\hat{\phi} + \hat{\theta}_s) \\ y_s + \rho \sin(\hat{\phi} + \hat{\theta}_s) \\ \pi + \hat{\phi} + \hat{\theta}_s - \hat{\theta} \end{bmatrix}
\]  

(14)

- estimate measurement for particle \( i \)

\[
z_i = \begin{bmatrix} \hat{\rho}_i \\ \hat{\theta}_i \\ \hat{\phi}_i \end{bmatrix} = \begin{bmatrix} \sqrt{dx_i^2 + dy_i^2} \\ \text{atan2}(dy_i, dx_i) - \hat{\theta}_s \\ \text{atan2}(-dy_i, -dx_i) - \hat{\theta}_m \end{bmatrix}
\]

(8)

- update weights (the \( \sigma \) are the sensor noise)

\[
P(x_{m_i}^{k+1}|x_s, z) = \frac{1}{\sqrt{2\pi\sigma_\rho}} e^{-\frac{(\rho - \hat{\rho}_i)^2}{2\sigma_\rho^2}} \cdot \frac{1}{\sqrt{2\pi\sigma_\theta}} e^{-\frac{(\theta - \hat{\theta}_i)^2}{2\sigma_\theta^2}} \cdot \frac{1}{\sqrt{2\pi\sigma_{\hat{\theta}}}} e^{-\frac{(\hat{\theta} - \hat{\theta}_i)^2}{2\sigma_{\hat{\theta}}^2}}
\]

(9)
Updating weights
function observe(pf, z, jf)
    % Straightforward code:
    % for p = 1:pf.nparticles
    %    % what do we expect observation to be for this particle?
    %    % use the sensor model h(.)
    %    z_pred = pf.sensor.h( pf.x(p,:), jf);
    %
    %    % how different is it
    %    innov(1) = z(1) - z_pred(1);
    %    innov(2) = angdiff(z(2), z_pred(2));
    
    %    % get likelihood (new importance). Assume Gaussian but any PDF works!
    %    % If predicted obs is very different from actual obs this score will be low
    %    % ie. this particle is not very good at predicting the observation.
    %    % A lower score means it is less likely to be selected for the next generation...
    %    % The weight is never zero.
    %    pf.weight(p) = exp(-0.5*innov'*inv(pf.L)*innov) + 0.05;
    % end
    
    % Vectorized code:
    invL = inv(pf.L);
    z_pred = pf.sensor.h( pf.x, jf);
    z_pred(:,1) = z(1) - z_pred(:,1);
    z_pred(:,2) = angdiff(z(2), z_pred(:,2));
    
    LL = -0.5*[invL(1,1); invL(2,2); 2*invL(1,2)];
    e = [z_pred(:,1).^2 z_pred(:,2).^2 z_pred(:,1).*z_pred(:,2)]*LL;
    pf.weight = exp(e) + pf.w0;
end
Select with replacement

```
Input: double W[N]

Require: \( \sum_{i=1}^{N} W_i = 1 \)

\( Q = \text{cumsum}(W); \) \{ calculate the running totals \( Q_j = \sum_{i=0}^{j} W_i \} \)

\( t = \text{rand}(N+1); \) \{ \( t \) is an array of \( N+1 \) random numbers. \}

\( T = \text{sort}(t); \) \{ Sort them (\( O(n \log n) \) time) \}

\( T(N+1) = 1; i=1; j=1; \) \{ Arrays start at 1 \}

while \( (i \leq N) \) do
    if \( T[i] < Q[j] \) then
        Index[i]=j;
        i=i+1;
    else
        j=j+1;
    end if
end while

Return(Index)
```

**Algorithm 3:** Select with Replacement Resampling Algorithm; functions are noted as underlined text, Comments are inside curly brackets “{“.”
function select(pf)

    % particles with large weights will occupy a greater percentage of the
    % y axis in a cumulative plot
    CDF = cumsum(pf.weight)/sum(pf.weight);

    % so randomly (uniform) choosing y values is more likely to correspond to
    % better particles...
    iSelect = pf.rand(pf.nparticles,1);

    % find the particle that corresponds to each y value (just a look up)
    iNextGeneration = interp1(CDF, 1:pf.nparticles, iSelect, 'nearest', 'extrap');

    % copy selected particles for next generation..
    pf.x = pf.x(iNextGeneration,:);
end