Particle Filters
Bayesian Filter

- Estimate state \( x \) from data \( Z \)
  - What is the probability of the robot being at \( x \)?
- \( x \) could be robot location, map information, locations of targets, etc...
- \( Z \) could be sensor readings such as range, actions, odometry from encoders, etc...
- This is a general formalism that does not depend on the particular probability representation
- Bayes filter **recursively** computes the posterior distribution:
  \[
  Bel(x_T) = P(x_T | Z_T)
  \]
Iterating the Bayesian Filter

• Propagate the motion model:

\[ Bel(x_t) = \int P(x_t \mid a_{t-1}, x_{t-1}) Bel(x_{t-1}) dx_{t-1} \]

Compute the current state estimate before taking a sensor reading by integrating over all possible previous state estimates and applying the motion model.

• Update the sensor model:

\[ Bel(x_t) = \eta P(o_t \mid x_t) Bel(x_t) \]

Compute the current state estimate by taking a sensor reading and multiplying by the current estimate based on the most recent motion history.
Mobile Robot Localization  
(Where Am I?)

- A mobile robot moves while collecting sensor measurements from the environment.
- Two steps, action and sensing:
  - Prediction/Propagation: what is the robots pose $\mathbf{x}$ after action $A$?
  - Update: Given measurement $\mathbf{z}$, correct the pose $\mathbf{x}'$
- What is the probability density function ($pdf$) that describes the uncertainty $\mathbf{P}$ of the poses $\mathbf{x}$ and $\mathbf{x}'$?
State Estimation

- Propagation

\[ P(x_{t+1}^- \mid x_t, \alpha) \]

- Update

\[ P(x_{t+1}^+ \mid x_{t+1}^-, z_{t+1}) \]
Traditional Approach  Kalman Filter

- Optimal for linear systems with Gaussian noise
- Extended Kalman filter:
  - Linearization
  - Gaussian noise models
- Fast!
Monte-Carlo State Estimation

(Particle Filtering)

• Employing a Bayesian Monte-Carlo simulation technique for pose estimation.
• A particle filter uses $N$ samples as a discrete representation of the probability distribution function ($pdf$) of the variable of interest:

$$S = [\tilde{x}_i, w_i : i = 1 \cdots N]$$

where $x_i$ is a copy of the variable of interest and $w_i$ is a weight signifying the quality of that sample.

In our case, each particle can be regarded as an alternative hypothesis for the robot pose.
The particle filter operates in two stages:

- **Prediction**: After a motion ($\alpha$) the set of particles $S$ is modified according to the action model

$$S' = f(S, \alpha, \nu)$$

where ($\nu$) is the added noise.

The resulting *pdf* is the **prior** estimate before collecting any additional sensory information.
Particle Filter (cont.)

• **Update:** When a sensor measurement \((z)\) becomes available, the **weights** of the particles are updated based on the likelihood of \((z)\) given the particle \(x_i\)

\[
w'_i = P(z \mid \tilde{x}_i) w_i
\]

The **updated particles** represent the posterior distribution of the moving robot.
Remarks:

- **In theory**, for an infinite number of particles, this method models the true *pdf*.
- **In practice**, there are always a finite number of particles.
Resampling

For finite particle populations, we must focus population mass where the PDF is substantive.

- Failure to do this correctly can lead to divergence.
- Resampling needlessly also has disadvantages.

One way is to estimate the need for resampling based on the variance of the particle weight distribution, in particular the coefficient of variance:

\[
CV_t^2 = \frac{\text{var}(w_t(i))}{E^2(w_t(i))} = \frac{1}{M} \sum_{i=1}^{M} (Mw_t(i) - 1)^2
\]

\[
ESS_t = \frac{M}{1 + CV_t^2}
\]
Prediction: Odometry Error Modeling

- **Piecewise linear motion**: a simple example.
- **Rotation**: Corrupted by Gaussian Noise.
- **Translation**: Simulated by multiple steps. Each step models translational and rotational error.

**Single step**:
Small *rotational* error (drift) before and after the translation.

*Translational* error proportional to the distance traveled.

All errors drawn from a Normal Distribution.
Odometry Error Modeling
Odometry Error Modeling

\[(\sigma_{\text{trans}} = 5, \sigma_{\text{drift}} = 1)\]
Odometry Error Modeling

\[ (\sigma_{\text{trans}} = 1, \sigma_{\text{drift}} = 5) \]

Distance traveled: 100 cm
Distance traveled: 200 cm
Distance traveled: 300 cm
Distance traveled: 400 cm
Odometry Error Modeling

$(\sigma_{\text{trans}}=10, \sigma_{\text{drift}}=10)$
Odometry Error Modeling

\[ (\sigma_{\text{trans}} = 5, \sigma_{\text{drift}} = 3) \]

- Distance traveled: 100 cm
- Distance traveled: 200 cm
- Distance traveled: 300 cm
- Distance traveled: 400 cm
Prediction-Only Particle Distribution

Sample Trajectory of the Mapping Robot (solid: Tracker, dash-dot: Odometer)
Propagation of a discrete time system ($\delta t = 1$ sec)

\[
x_{i}^{t+1} = x_{i}^{t} + (v_{t} + w_{v_{t}})\delta t \cos \phi_{i}^{t}
\]

\[
y_{i}^{t+1} = y_{i}^{t} + (v_{t} + w_{v_{t}})\delta t \sin \phi_{i}^{t}
\]

\[
\phi_{i}^{t+1} = \phi_{i}^{t} + (\omega_{t} + w_{\omega_{t}})\delta t
\]

Where $w_{v_{t}}$ is the additive noise for the linear velocity, and $w_{\omega_{t}}$ is the additive noise for the angular velocity.
Continuous motion example

- \( Dt=1 \text{ sec} \)
- Plotting 1 sample/sec all the particles every 5 sec
- Constant linear velocity
- Angular velocity changes randomly every 10 sec
Continuous motion example
Prediction Examples Using a PF

Piecewise linear motion

(Translation and Rotation)

– Command success 70%
– Start at [-8,0,0]
– Translate by 4m
– Rotate by 30°
– Translate by 6m
Start \([-8,0,0^\circ]\)
Translate by 4m
Rotate by 30°
Translate by 6m
Propagation

- Known position, known orientation
- Bounded linear velocity \([0.5 \ 0.7]\) m/sec
- Bounded angular velocity
- Run 200 sec.
- Plotting every twenty fifth sec.
Bounded Velocities

\[ \omega \in [-0.01, 0.01] \text{ rad/sec} \]

\[ \omega \in [-0.1, 0.1] \text{ rad/sec} \]

\[ \omega \in [-0.2, 0.2] \text{ rad/sec} \]
Propagation

- Known position, unknown orientation
- Bounded linear velocity $[0.5 \ 0.7]$ m/sec
- Bounded angular velocity $[-0.1 \ 0.1]$ rad/sec
- Run 200 sec.
- Plotting every twenty fifth sec.
Propagation
Propagation

- Known position, known orientation
- Bounded linear velocity \([0.0 \ 0.5] \text{ m/sec}\)
- Bounded angular velocity \([-0.01 \ 0.01] \text{ rad/sec}\)
- Run 200 sec.
- Plotting every twenty fifth sec.

- For a particle to stay at the origin, it has to draw zero velocity 25 times in the row.
Bounded velocities
Update Examples Using a PF
Environment with two red doors
(uniform distribution)
Environment with two red doors

(Sensing the red door)
Sensing four walls
Four possible areas
Update Range only

\[ w_i^t = w_i^{t-1} \cdot \frac{1}{\sqrt{2\pi}\sigma_\rho^2} e^{-\frac{(\rho_i - \rho_r)^2}{2\sigma_\rho}} \]
Update Range only
Update Range only
Update Range only
$w_i^t = w_i^{t-1} \cdot \frac{1}{\sqrt{2\pi\sigma_\phi^2}} e^{-\frac{(\phi_i - \phi_r)^2}{2\sigma_\phi^2}}$
Update Bearing only
Update Bearing only
Update Bearing only
Update Bearing only
Update Bearing only

\( \dot{v} \in [0, 0.5] \text{ m/sec}, \dot{w} \in [-0.05, 0.05] \), Bearing only update
Update Bearing only
Update Range and Bearing

\[ w^t_i = w^{t-1}_i \cdot \frac{1}{\sqrt{2\pi}\sigma^2_\phi} e^{-\frac{(\phi_i - \phi_r)^2}{2\sigma^2_\phi}} \cdot \frac{1}{\sqrt{2\pi}\sigma^2_\rho} e^{-\frac{(\rho_i - \rho_r)^2}{2\sigma^2_\rho}} \]
Update Compass only

\[ w_i^t = w_i^{t-1} \cdot \frac{1}{\sqrt{2\pi\sigma_\theta^2}} e^{-\frac{(\theta_i - \theta_r)^2}{2\sigma_\theta^2}} \]
Update Compass only
Update Compass only
Cooperative Localization

- Pose of the moving robot is estimated relative to the pose of the stationary robot. Stationary Robot observes the Moving Robot.

Robot Tracker Returns: \(<\rho, \theta, \phi>\)

\[
\mathbf{x}_{m_{est}}(k + 1) = 
\begin{pmatrix}
    x_{m_{est}} \\
    y_{m_{est}} \\
    \theta_{m_{est}}
\end{pmatrix} =
\begin{pmatrix}
    x_s + \rho \cos(\theta + \theta_s) \\
    y_s + \rho \sin(\theta + \theta_s) \\
    \pi - (\phi - (\theta + \theta_s))
\end{pmatrix}
\]
Laser-Based Robot Tracker

Robot Tracker Returns:
\[ <\rho,\theta,\phi> \]
Tracker Weighting Function

\[ W = \frac{1}{\sqrt{2\pi\sigma^2_\rho}} e^{-\frac{(\rho-\rho_i)^2}{2\sigma^2_\rho}} \times \frac{1}{\sqrt{2\pi\sigma^2_\theta}} e^{-\frac{(\theta-\theta_i)^2}{2\sigma^2_\theta}} \times \frac{1}{\sqrt{2\pi\sigma^2_\phi}} e^{-\frac{(\phi-\phi_i)^2}{2\sigma^2_\phi}} \]

(\sigma_\rho = 3, \sigma_\theta = 3, \sigma_\phi = 2)
Example: Prediction
Example: Update
Example: Prediction
Example: Update
Variations on PF

- Add some particles uniformly
- Add some particles where the sensor indicates
- Add some jitter to the particles after propagation
- Combine EKFs to track landmarks
Keep in Mind:

- The number of particles increases with the dimension of the state space.
Complexity results for SLAM

• $n =$ number of map features
• Problem: naïve methods have high complexity
  – EKF models $O(n^2)$ covariance matrix
  – PF requires prohibitively many particles to characterize complex, interdependent distribution
• Solution: exploit conditional independencies
  – Feature estimates are independent given robot’s path
Generating Random Numbers

From a uniform RNG produce samples following the Normal distribution: The most basic form of the transformation looks like:

\[ y_1 = \sqrt{-2 \ln(x_1)} \cos(2 \pi x_2) \]
\[ y_2 = \sqrt{-2 \ln(x_1)} \sin(2 \pi x_2) \]

The **polar form** of the Box-Muller transformation is both faster and more robust numerically. The algorithmic description of it is:

```c
float x1, x2, w, y1, y2;
do {
x1 = 2.0 * ranf() - 1.0; x2 = 2.0 * ranf() - 1.0;
w = x1 * x1 + x2 * x2;
} while ( w >= 1.0 );
w = sqrt( (-2.0 * ln(w) ) / w );
y1 = x1 * w;
y2 = x2 * w;
```

See: http://www.taygeta.com/random/gaussian.html
Rao-Blackwellization

Figure from [Montemerlo et al – Fast SLAM]
RBPF Implementation for SLAM

• 2 steps:
  – Particle filter to estimate robot’s pose
  – Set of low-dimensional, independent EKF’s (one per feature per particle)

• E.g. FastSLAM which includes several computational speedups to achieve $O(M \log N)$ complexity (with $M$ number of particles)
Questions

• For more information on PF:

http://www.cim.mcgill.ca/~yiannis/ParticleTutorial.html