0.1 Bayes Filter

- Bayes Filter accounts for both robot state and perception data
- To use Bayes filter, the state must be discrete, usually represented by a grid
- Each grid cell, contains the **belief** (probability that the true state of the system is x_t)

Algorithm 1: Discrete Bayes Filter Algorithm

```
Inputs: Bel(x), d;
η = 0;
if d is a perceptual data item z then
    for all x do
        Bel'(x) = P(z|x)Bel(x);
        η = η + Bel'(x);
    end
    for all x do
        Bel'(x) = η^{-1} Bel'(x);
    end
end
else if d is an action data item u then then
    for all x do
        Bel'(x) = Σ_{x'} P(x|u, x')Bel(x');
    end
end
Return Bel'(x)
```

0.2 Kalman Filter

- Kalman filter used when state space is continuous variables
- They key idea is to represent everything with *gaussians*
- Univariate and multivariate gaussians
- We stay in "Gaussian world" as long as we start with Gaussians and perform only linear transformations
- Estimate state x of a *discrete-time* controlled process governed by linear stochastic difference equation:

  $$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t$$

  and sensor measurement

  $$z_t = C_t x_t + \delta_t$$

  where:

x_t = current state
A_t = matrix describing how state changes from $t-1$ to t without controls
B_t = matrix that describes how control u_t changes the state from $t-1$ to t
C_t = matrix that describes how to map state x_t to an observation z_t
ϵ_t = process noise normally distributed with covariance R_t
δ_t = measurement noise normally distributed with covariance Q_t

---

Algorithm 2: Kalman Filter

Prediction: use dynamics to predict what will happen;
$\bar{\mu}_t = A_t\mu_{t-1} + B_t u_t$;
$\bar{\Sigma}_t = A_t\Sigma_{t-1}A_t^T + R_t$;
Correction: use sensor measurement to correct prediction;
$K_t = \bar{\Sigma}_t C_t^T (C_t\bar{\Sigma}_t C_t^T + Q_t)^{-1}$;
$\mu_t = \bar{\mu}_t + K_t(z_t - C_t\bar{\mu}_t$;
$\Sigma_t = (I - K_t C_t)\bar{\Sigma}_t$;
return μ_t, Σ_t

- Comments:
  - Highly efficient: only need to compute matrix multiplication
  - Optimal for linear Gaussian systems, but most robotics systems are nonlinear

0.3 Extended Kalman Filter (EKF)

- Most robotics problem deal wth nonlinear dynamics and sensors

$$x_t = g(u_t, x_{t-1})$$

$$z_t = h(x_t)$$

- EKF trick: use a *local linear approximation* by computing the Jacobians of g and h

$$x_t = g(u_t, x_{t-1}) \approx g(u_t, \mu_{t-1}) + G_t(x_{t-1} - \mu_{t-1})$$

$$z_t = h(x_t) \approx h(\bar{\mu}_t) + H_t(x_t - \bar{\mu}_t)$$

where
$G_t = \frac{\partial g(u_t, \mu_{t-1})}{\partial x_{t-1}}$
$H_t = \frac{\partial h(\bar{\mu}_t)}{\partial x_t}$

- Comments
  - Highly efficient
  - Not optimal, because it is an approximation of the nonlinear function
  - Can diverge if nonlinearities are large (e.g. close to beacon)
  - Everything must be a Gaussian
  - Cannot be used if transition is non-linear, e.g. multimodal distributions

0.4 Unscented Kalman Filter

- Key idea:
  1. Sample a set of sigma points from Gaussian distribution
  2. Pass sigma points through function
  3. Re-estimate Gaussian

Algorithm 3: Unscented Kalman Filter

$\mathcal{X}_{t-1}$ = Compute sigma points using μ_{t-1} and Σ_{t-1};
$\bar{\mathcal{X}}_t^* = g(u_t, \mathcal{X}_{t-1})$ // Apply dynamics ;
$\bar{\mu}_t = \sum_{i=0}^{2n} w_m^{[i]} \bar{\mathcal{X}}_t^{*[i]}$ // Estimate new mean from sigma points;
$\bar{\Sigma}_t = \sum_{i=0}^{2n} w_c^{[i]} (\bar{\mathcal{X}}_t^{*[i]} - \bar{\mu}_t)(\bar{\mathcal{X}}_t^{*[i]} - \bar{\mu}_t)^T + R_t$;
$\bar{\mathcal{X}}_t$ = Compute sigma points using $\bar{\mu}_t$ and $\bar{\Sigma}_t$;
$\bar{\mathcal{Z}}_t = h(\bar{\mathcal{X}}_t)$ // Apply sensor model;
TODO;

- Comments
  - Highly efficient
  - Better approximation than EKF: accurate in first two terms of Taylor expansion
  - Derivative-free, does not require any Jacobians

0.5 Particle Filter

- No need to make assumptions of distributions unlike Kalman Filter which assumes that all error is Gaussian
- Sample from the implicit distribution of the state at any given time
- At time t, distribution is represented by the M samples of the robot's states

$$X_t = x_t^{[1]}, x_t^{[2]}, ..., x_t^{[M]}$$

$$W_t = w_t^{[1]}, w_t^{[2]}, ..., w_t^{[M]}$$

- If starting location is know, can initialize particles around start, else scatter particles through the environment

Algorithm 4: Particle Filter

X_0 = Sample M particles from $P(X_0)$;
$t = 0$;
while *True* **do**
    $t++$;
    u_t = action();
    z_t = sensor();
    $S_t = X_t = \{\}$;
    for m = *1 to M* **do**
        sample $x_t^{[m]} \sim p(x_t | u_t, x_{t-1}^{[m]}$ from $X_{t-1})$;
        $w_t^{[m]} = p(z_t | x_t^{[m]})$;
        $S_t = S_t \cup (x_t^{[m]}, w_t^{[m]})$;
    end
    for m = *1 to M* **do**
        draw i with probability $\sim w_t^{[i]}$;
        add $x_t^{[i]}$ from S_t to X_T;
    end
end

- Sampling strategies
  - Need to preserve diversity of particles, inject random particles
  - Low-variance sampling
  - Stratified sampling
- Comments
  - Represent distribution with a set of particles
  - Approximate arbitrary probability distributions
  - More particles = better approximation but more computation cost