Announcements

- Final exam will be at 7pm on Wednesday December 14th
  - Date of last class
  - 1.5 hrs long
- I won’t ask anything about the last few classes.

Recap: Reasoning Over Time

- Stationary Markov models
  \[ P(X_t) \] (\( t = 1 \) to \( t \))

- Hidden Markov models
  \[ P(E_t | X_t) \]

Recap: Filtering

Elapse time: compute \( P(X_t | e_{1:t-1}) \)

Observe: compute \( P(X_t | e_t) \)

\[
P(x_t | e_{1:t}) = \sum_{x_{t-1}} P(x_{t-1} | e_{1:t-1}) \cdot P(x_t | x_{t-1})
\]

Example: State Representations for Robot Localization

Grid based approaches (Markov localization)

Particle Filtering

- Sometimes |X| is too big to use exact inference
  - |X| may be too big to even store B(X)
  - E.g. X is continuous
  - |X|^2 may be too big to do updates

- Solution: approximate inference
  - Track samples of X, not all values
  - Samples are called particles
  - Time per step is linear in the number of samples
  - But: number needed may be large
  - In memory: list of particles, not states

- This is how robot localization works in practice
Representation: Particles

• Our representation of \( P(X) \) is now a list of \( N \) particles (samples)
  • Generally, \( N \ll |X| \)
  • Storing map from \( X \) to counts would defeat the point
• \( P(x) \) approximated by number of particles with value \( x \)
  • So, many \( x \) will have \( P(x) = 0! \)
• More particles, more accuracy
  • For now, all particles have a weight of 1

Particle Filtering: Elapse Time

• Each particle is moved by sampling its next position from the transition model
  \[ x' = \text{sample}(P(X'|x)) \]
  • This is like prior sampling – samples' frequencies reflect the transition probs
  • Here, most samples move clockwise, but some move in another direction or stay in place
  • This captures the passage of time
    • If we have enough samples, close to the exact values before and after (consistent)

Particle Filtering: Observe

• Slightly trickier:
  • Don’t do rejection sampling (why not?)
  • We don’t sample the observation, we fix it
  • This is similar to likelihood weighting, so we downweight our samples based on the evidence
    \[ w(z) = P(e|x) \]
    \[ B(X) \propto P(e|X)B'(X) \]
  • Note that, as before, the probabilities don’t sum to one, since most have been downweighted (in fact they sum to an approximation of \( P(e) \))

Particle Filtering: Resample

• Rather than tracking weighted samples, we resample
  • \( N \) times, we choose from our weighted sample distribution (i.e. draw with replacement)
  • This is equivalent to renormalizing the distribution
  • Now the update is complete for this time step, continue with the next one

Particle Filter Algorithm

\[ \text{Bel}(x) = \eta \int \int \int \text{Bel}(x_0) \ p(x_1|x_0,u_0) \ p(z|u_0) \ dx_1 \]

Robot Localization

• In robot localization:
  • We know the map, but not the robot’s position
  • Observations may be vectors of range finder readings
  • State space and readings are typically continuous (works basically like a very fine grid) and so we cannot store \( B(X) \)
  • Particle filtering is a main technique
Robot Motion Model

Proximity Sensor Model

Laser sensor

Sonar sensor
SLAM

• SLAM = Simultaneous Localization And Mapping
  • We do not know the map or our location
  • Our belief state is over maps and positions!
  • Main techniques: Kalman filtering (Gaussian HMMs) and particle methods

Example: State Representations for Robot Localization

Robotic Cars

• DARPA Grand Challenge
• DARPA Urban Challenge
• http://www.youtube.com/watch?v=SQFEmR50HAc
Kalman Filters - Equations

Recursive filter for estimating state of linear dynamical system from noisy measurements

\[ P(x_{t+1} | x_t) = N(Ax_t, \Gamma) \]
\[ P(x_t | x_t) = N(Cx_t, \Sigma) \]

where:
- \( x_t \): State at time \( t \)
- \( A \): State transition matrix (n x n)
- \( C \): Measurement matrix (m x n)
- \( w_t \): Process noise (\( \epsilon \mathbb{R}^n \))
- \( v_t \): Measurement noise (\( \epsilon \mathbb{R}^m \))
- \( \Sigma \): Process noise covariance matrix
- \( \Gamma \): Measurement noise covariance matrix

Process dynamics (motion model)

\[ x_t = Ax_{t-1} + w_t \]
\[ y_t = Cx_t + v_t \]
\[ w_t \sim N(0, \Gamma) \]
\[ v_t \sim N(0, \Sigma) \]

Measurements (observation model)

\[ N(\mu,m,V) = \frac{1}{2\pi V} \exp\left(-\frac{1}{2}(x-m)^T V^{-1}(x-m)\right) \]

Kalman Filters - Update

Predict state, covariance

\[ \hat{x}_t = Ax_{t-1} \]
\[ P_t = AP_{t-1}A^T + \Gamma \]

Compute Gain

\[ K_t = P_t C^T ( CP_t C^T + \Sigma)^{-1} \]

Compute Innovation

\[ J_t = \hat{y}_t - C \hat{x}_t \]

Update

\[ \hat{x}_t = \hat{x}_t - K_t J_t \]
\[ P_t = (I - K_t C)P_t \]

Kalman Filter - Example

\[ x_t = Ax_{t-1} + B + w_t \]
\[ y_t = Cx_t + v_t \]
\[ w_t \sim N(0, \Gamma) \]
\[ v_t \sim N(0, \Sigma) \]

\[ A = [1] \]
\[ B = [u] \]
\[ C = [1] \]
\[ D = [1] \]

Predict

\[ \hat{x}_t = Ax_{t-1} + B \]
\[ P_t = AP_{t-1}A^T + \Gamma \]
### Kalman Filter – Example

Predict:
- $\hat{x}_t = A\hat{x}_{t-1} + B$
- $P_t = AP_{t-1}A^T + \Gamma$

Compute Innovation:
- $J_t = \hat{y}_t - C\hat{x}_t$

Compute Gain:
- $K_t = P_tC^T(CP_tC^T + \Sigma)^{-1}$

Update:
- $\hat{x}_t = \hat{x}_t - K_tJ_t$
- $P_t = (I - K_tC)P_t$

### Kalman Filter Applications

- Apollo guidance computer
- Cruise missiles
- Airplane autopilot
- Robotics
- Finance

### Continuous State Approaches

- Perform very accurately if the inputs are precise (performance is optimal with respect to any criterion in the linear case).
- Computational efficiency.
- Requirement that the initial state is known.
- Inability to recover from catastrophic failures
- Inability to track Multiple Hypotheses the state (Gaussians have only one mode)

### Discrete State Approaches

- Ability (to some degree) to operate even when its initial pose is unknown (start from uniform distribution).
- Ability to deal with noisy measurements.
- Ability to represent ambiguities (multi modal distributions).
- Computational time scales heavily with the number of possible states (dimensionality of the grid, number of samples, size of the map).
- Accuracy is limited by the size of the grid cells/number of particles-sampling method.
- Required number of particles is unknown

### Best Explanation Queries

- Query: most likely seq:
  \[ \arg \max_{x_{1:T}} P(x_{1:T} | e_{1:T}) \]
**State Path Trellis**

- State trellis: graph of states and transitions over time

\[
X_1 \quad X_2 \quad \cdots \quad X_N
\]

- Each arc represents some transition \( x_{t-1} \to x_t \)
- Each arc has weight \( P(x_t|x_{t-1})P(e_t|x_t) \)
- Each path is a sequence of states
- The product of weights on a path is the seq’s probability
- Can think of the Forward (and now Viterbi) algorithms as computing sums of all paths (best paths) in this graph

**Viterbi Algorithm**

\[
x_{1:T}^* = \arg \max_{x_{1:T}} P(x_{1:T}, x_{1:T}, e_{1:T}) = \arg \max_{x_{1:T}} P(x_{1:T}, e_{1:T})
\]

\[
m_t[x_t] = \max_{x_{1:t-1}} P(x_{1:t-1}, x_t, e_{1:t})
= \max_{x_{1:t-1}} P(x_{1:t-1}, e_{1:t-1})P(x_t|x_{t-1})P(e_t|x_t)
= P(e_t|x_t) \max_{x_{1:t-1}} P(x_t|x_{t-1}) \max_{x_{1:t-2}} P(x_{1:t-2}, e_{1:t-2})
= P(e_t|x_t) \max_{x_{1:t-1}} m_{t-1}[x_{t-1}]
\]

**Example**

**Andrew Viterbi**

[Qualcomm logo]