Robot Races on Friday Nov 14 from 3-5 p.m.
- The take home exam will be pushed back 1 week, now due after Robot Race.
- Check out Robot probability on website (Bayes Filter)

A Probabilistic Filter combines probabilities of Robot Motion and Sensing to produce a distribution over states indicating probability of robot being at a particular location.

Random Variables: State \( x \).

Sensor measurement \( z \)

Control \( u \) is a constant (known)

Two underlying (known) distributions:

- Motion Model \( p(x'|x,u) \)
  - Probability of reaching next state \( x' \) given that robot applied control \( u \) in state \( x \)
- Measurement Model \( p(z|x) \)
  - Probability of observing sensor reading \( z \) given robot in state \( x \)
  - Typically encodes knowledge of world or map.

The Bayes Filter

Let \( \text{Bel}(x) \) represent belief robot is at state \( x \).

1. Apply motion model: Execute control \( u \) and set for each \( x' \):

\[
\text{Bel}(x') = \sum_{x} \text{Bel}(x) \cdot p(x' | x, u)
\]

2. Apply sensor model: Observe measurement \( z \) and set for each \( x \):

\[
\text{Bel}(x) = \frac{\Pi p(z | x) \cdot \text{Bel}(x)}{\sum_{x'} p(z | x') \cdot \text{Bel}(x')}
\]

(Arte to make sure \( \text{Bel}(x) \) sums to 1.)
Bel(x) is basically p(x) - actually p(x_{t+1} | v_{1:t}, z_{t:t})
Bel(x) is p(x_{t+1} | v_{1:t}, z_{1:t-1})
P(x') = \sum_x P(x' | x) P(x) (Theorem of total probability)
P(x | z) = \frac{P(z | x) P(x)}{P(z)} = \sum_x P(z | x') P(x')
- P(x') is Bel(x)
- P(z) = ??

Runtime stuff
N states
M sensor readings

O(N^2) \quad Bel(x') = \sum_x Bel(x) \cdot p(x' | x, v)
O(N) \quad Bel(x) = \prod_x p(z | x) Bel(x)

10/10/14
- Review from Homework #7
  - \( H = -kp \dot{x} - k_d x \)
  - \( N \frac{k_d}{s} (m) \frac{k_p}{s} (m/s) \)
  - k_p (m/s)
- LQR [super popular] [Building blocks]
  - Matrices
    - A, B, Q, R
  - Pick time
    - N to +
  - Out comes optimal gains to apply
Lab 4
- Test code one test at a time, not everything at once
- Parameter tuning
- Start slow, then ramp up speed
- Timing to shave robot
- Charge the robot! \(^3\) (laptop?)

Bayes' filter cont'd
- Motion model
  \[ P(x'|x, u) \]
  Example: \( x \in \{1, 2, 3, \ldots, 20\} \)
  \( u = 1 \)
- Measurement model
  \[ P(z|x) \]
  \( P(\text{triggered} | \text{door}) = 95\% \)
  \( P(\text{no trigger} | \text{door}) = 5\% \)
  \( P(\text{trigger} | \text{no door}) = 10\% \)
  \( P(\text{no trigger} | \text{no door}) = 90\% \)

Random sampling
- Suppose I have a function \( \text{rand()} \) which returns an integer from 0 to a large \( \text{rand-\max} \) with equal probability.
- How can I use \( \text{rand()} \) to implement \( \text{rand()} \) which returns a floating point number uniformly distributed in \([0, 1)\)?

Python:
```python
def rand():
    return float(rand()) / (rand_max)
```

Assume I have an " unfair" coin with:
- \( p(\text{heads}) = .7 \)
- \( p(\text{tails}) = .3 \)

How can I use \( \text{rand()} \) to implement \( \text{uniform-}\text{flip} \)?
def unfair_flip():
    if random() > 0.3
        return 'heads'
    else:
        return 'tails'

Now do the same with an unfair_die_roll() with

\[ P(X) \]

\[ x = 1, 2, 3, 4, 5, 6 \]

def unfair_die_roll(): → linear search \Oh(n)
    p = fair()  
    if p < 1.0/21  // 1
        return 1
    elif p < 3.0/21 // 1+2
        return 2
    elif p < 9.0/21 // 1+2+3
        return 3

\[ \frac{1}{21}, \frac{3}{21}, \frac{6}{21}, \frac{10}{21}, \frac{15}{21}, \frac{21}{21} \] → binary search runtime \Oh(\log n)

find smallest element \geq p

Continuous Probability Distributions

**Discrete**
\[ x \in X \ (finite \ set) \]
\[ 0 \leq p(x) \leq 1 \]
\[ \sum_x p(x) = 1 \]

**Continuous**
\[ x \in \mathbb{R} \]
\[ p(x) \geq 0 \]
\[ \int_{-\infty}^{\infty} p(x)dx = 1 \]
Example: uniform distribution on $[0,1)$

$$p(x) = \begin{cases} 1 & x \in [0,1) \\ 0 & \text{otherwise} \end{cases}$$

Is $P(0,1,2,3,4,5,6,7,8)$ a probability distribution function? Explain:

Probability that $x \in [a,b]$ is

$$\int_a^b p(x) \, dx$$