Today:

- Bayes Filter, cont'd
- Sampling from distributions

Demo: bayes_filter_demo.py

Robot is in a discrete state.

It has a belief that it is in any particular location. These beliefs need to sum to 1.

Robot updates its beliefs according to a 2-part rule:

1. \( \text{bel}(x') = \text{bel}(x) \sum_x p(x' | x, u) \)

2. \( \text{bel}(x) = \eta \text{bel}(x) p(z | x) \)

Motion model included in part 1:

\( p(x' | x, u) \)

Given that you knew where robot was previously and what controls were given, to have it execute, what is the probability that it's in a specific current state?

Measurement model included in part 2:

\( p(z | x) \)

We can represent both the measurement and motion models as tables.
η: normalization constant
defined s.t. \( \sum_x \text{bel}(x) = 1 \)

\[
\eta = \frac{1}{\sum_x \text{bel}(x) p(z|l|x)}
\]

How can we justify \( 1 + 2 \)?

1: summing across all possible \( x \)’s -
   this is TTP
2: Bayes’ Rule

Formal argument for why all of this works:

[Inductive Basis]

Dependency Graph:

```
A -> B
```

''B is dependent on A''

time

- next state depends on previous state
- measurement depends on current state

\[
\text{bel}(x_t) = p(x_{t+1} | z_{o:t}, u_{o:t})
\]

\[
= \frac{p(z_t | x_t, z_{o:t-1}, u_{o:t}) p(x_{t+1} | z_{o:t-1}, u_{o:t})}{p(z_t | z_{o:t-1}, u_{o:t})}
\]

using \( p(a | b, c) = \frac{p(b | a, c) p(b | c)}{p(a | c)} \)
1. $P(Z_t | X_t, Z_{0:t-1}, U_{0:t})$
   prob. of measurement given a state $X_t$, all previous measurements, and all previous controls

   we can simplify this dependency to
   $P(Z_t | X_t)$ if we know $X_t$, since this renders $U_{0:t}$ and $Z_{0:t-1}$ as irrelevant.

2. define $P(X_t | Z_{0:t-1}, U_{0:t})$ as $bel(x_t)$

3. we can ignore denominator by requiring
   that $\sum_x bel(x_t) = 1$

   combining 1, 2, and 3

   $bel(x_t) = \eta P(Z_t | X_t) bel(x_t)$

   repeating 2:

   $bel(x_t) = P(X_t | Z_{0:t-1}, U_{0:t})$

   by theorem of total probability

   note that A is equivalent to
   $bel(x_t) = P(X_t | Z_{0:t}, U_{0:t})$, just with $t = t-1$

   so, rewrite this as to $bel(x_{t-1})$
Also note that $\mathbb{B}$ can be rewritten without $w_{t+1}$ or $z_{0:t+1}$, because it's not a dependency:

$$p(x_{t+1}, x_t, u_t)$$

Now,

$$\text{bel}(x_t) = \sum_{x_{t-1}} p(x_{t+1}, x_t, u_t) \cdot \text{bel}(x_{t-1})$$

We need to unwind the inductive case to a start point $x_0$, where there were no measurements or previous controls. We need to define a base case (initial distribution over the states) ourselves.

Sketchy point: We're assuming the controls are independent of previous states; measurements more realistic dependency graph:

![Graph diagram]

We can justify saying the controls are random using the Central Limit Theorem.
Practically

Set of states must be finite and discrete.

Moving Bayes Filter into a continuous state space requires discretization

Hicks 2nd Floor:

100 m x 20 m

at 5 cm x 5 cm

- 2000 x 400 cells
- 80,000 states \( (n^2 \approx 6.4 \times 10^9) \)

In general:

\( n \) states (total)

\( M \) possible measurements

\( k \) possible controls

What is runtime of each step of Bayes' Filter?

(big-O notation)

1. Motion step:

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

for each \( x' \) in \( N \):

for each \( x \) in \( N \):

sum probabilities via look-up table for measurement model

we don't consider \( k \) because we only issued one control - assume we execute 1 control per step.

This is \( O(n^2) \).
2. Measurement Step: \[ m(x) = \eta \operatorname{Bel}(x) \cdot \operatorname{Bel}(x)p(z_1|x) \]

for each \( x \in \mathbb{X} \), compute \( \operatorname{Bel}(x)p(z_1|x) \), accumulate values for \( \eta \), compute \( \eta \),

for each \( x \in \mathbb{X} \), compute final values now that we know \( \eta \),

overall: \( O(2 \cdot n) = O(n) \)

This is assuming we have only one sensor.

Similar to \( k \), we don't consider \( m \) because we obtained a single known measurement.

We want to reduce this to something \( \ll O(n^2) \).

E.g., using Gaussians reduces this to \( O(1) \).
Random Sampling on a Computer

We can simulate a random integer using:
```
Numpy.random.randint(MAX_RAND + 1)
```
where MAXRND is a defined global constant.

to get a random float:
```
(float - rand())
```
```
rand
```
```
float(MAX_RAND) <- forces float division
```

We're using a binary computer — under the hood, we're still choosing randomly from a discrete set of integers. Computers can't do continuous random sampling b/c they're already discretized.

Unfair Coin Simulation:

\[
P(H) = 0.3 \\
P(T) = 0.7
\]

\[
P(\text{Unfair = heads}) = \text{float}_\text{rand()} < 0.3
\]

does \(<\) vs. \(\leq\) matters?
- doesn't matter b/c in a continuous distribution, \(P(\text{exactly 0.3}) = 0\)
- matters b/c this is discretized on a computer.

See prob_sampling.py for histogram

More trials will get is closer to the
defined distributions
takeaways: Computers are discrete, we can fake everything using integers

Unfair Die Sampling:

<table>
<thead>
<tr>
<th></th>
<th>Individual</th>
<th>Cumulative</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P(1)$</td>
<td>1/21</td>
<td>1/21</td>
</tr>
<tr>
<td>$P(2)$</td>
<td>2/21</td>
<td>3/21</td>
</tr>
<tr>
<td>$P(3)$</td>
<td>3/21</td>
<td>6/21</td>
</tr>
<tr>
<td>$P(4)$</td>
<td>4/21</td>
<td>10/21</td>
</tr>
<tr>
<td>$P(5)$</td>
<td>5/21</td>
<td>15/21</td>
</tr>
<tr>
<td>$P(6)$</td>
<td>6/21</td>
<td>1</td>
</tr>
</tbody>
</table>

$r = 0.533$

Goal: Find $i$ s.t. Cumulative sum $\geq r$

How to make this better?

- Vectorization instead & for-loop