State estimator ~ perception ~ sensor function

State estimation

**insert figure of arbitrary system**

State update dynamics x' = f(x,u,t)

Sensor update z = h(x,u,t)

**insert figure of state estimator**

Figure out state using sensor inputs and know inputs to state

What is the state of the state estimator?

b' = f(b,u,z,t) input

x^ = h(b,u,z,t) output. This output is our "best guess state estimation"

x^: best guess at x

X^max liklihood (mode) }

X^min-max (median) } defined in terms of the probability distribution of x.

X^min-variance (mean) }

**insert figure of mean, median, and mode**

State Estimator State

Belief(x') = P(x = x')

-This gives a PDF of x

-P is the probability that our best guess for x (x') is equal to the actual value of x, given over all possible values of x'

example: 1-D point

**insert figure of single point and distributions**

In order to apply a Kalman filter, the system must have

1) Be linear

2) Have Gaussian noise and uncertainty

Therefore, all possible beliefs will also follow a Gaussian distribution.

To fully define a Gaussian distribution, only mean and variance are required.

Discrete State

example: **insert picture of discrete state**

x[t+1] = x[t] + u[t] + d[t]

noise:

To quantify the noise, we need to give a PDF of the noise, specified over u.

An example PDF is:

d= {-1 up to 10%}

{0 up to 80%}

{1 up to 10%}

This example PDF implies that the noise will lead to a reading that is 1 to the left 10% of the time, a reading of the particle being unchanged 80% of the time, and a reading that is 1 to the right 10% of the time.

How to describe the belief state:

Bel(x) = [p(x=1), p(x=2), p(x=3), p(x=4), p(x=5)]

The belief state is described by the probabilities of each state.

The particle starts in the middle,

Bel0(x) = (0, 0, 1, 0, 0), u(0) = 0

Bel1(x) = (0, 0.1, 0.8, 0.1, 0), u(1) = 0

Bel2(x) = (0.01, 0.16, 0.66, 0.16, 0.01)

Note: in Bel2(x), to find values (x1, x2, x3, x4, x5)

x1 = P(1)\*0.8 + P(2)\*0.1 = 0\*0.8 + 0.1\*0.1 = 0.01

x2 = P(1)\*0.1 + P(2)\*0.8 + P(3)\*0.1 = 0\*0.1 + 0.1\*0.8 + 0.8\*0.1 = 0.16

x3 = P(2)\*0.1 + P(3)\*0.8 + P(4)\*0.1 = 0.1\*0.1 + 0.8\*0.8 + 0.1\*0.1 = 0.66

x4 = P(3)\*0.1 + P(4)\*0.8 + P(5)\*0.1 = 0.8\*0.1 + 0.1\*0.8 + 0\*0.1 = 0.16

x5 = P(4)\*0.1 + P(5)\*0.8 = 0.1\*0.1 + 0\*0.8 = 0.01

Bayesian Filter

Belief state update with input u(t).

Belt+1(x) = f( Belt(x), u(t) )

Belt+1(x) =

z[t] = x[t] + n[t]

Note: The noise, n[t], does not change where we are, just what information we know about where we are.

z[0] = 3. This is the measurement that we ended up in state 0.

Bel1+(x) = P(x|z) = , where P(x) = Belt-(x)

How to apply Baysian filter for finite states to infinite number of states?

Make a lot of samples, and run dynamics update on each sample.

**Insert graphs of dots**

Particle Filter

­­­­1) Spawn N particles according to the knowledge of initial state and with equal probability

2) Run dynamics update to each particles' state

3) Update the measurement to particle probability

Loop between steps 2 and 3 each time.

This is a basic, minimum, particle filter.

Note:

The Bayesian filter requires state storage and operations.

The requirements for storage and operation grow with size

In a particle filter, N dictates the runtime.

Problems with particle filters:

Loss of particles

Run the risk of too little particles left

With the loss of particles comes a loss of information