Variations on a particle filter

Geoff Gordon

November 6, 2001

Overview

What is state tracking? Why is it important/interesting?

Markov tracking

- Kalman filters
- particle filters
- various ways to combine them

HKPFs for robot localization and people tracking

Review and conclusion

The tracking problem

Robots move around, sense environment

Maintain belief about state of world

Poses of robots, locations of people, map of obstacles, . . .

Usually in support of planning—good tracking can make planning much easier

Why is tracking hard?

State is usually high-dimensional

World models are nonlinear and discontinuous

$\Rightarrow$ belief is often a complicated PDF

Our tracking problem

An example to keep in mind: 2 robots, 3-5 people

Robots have x, y, θ; people have $x, y, \dot{x}, \dot{y}$

Minimum of 18 dof

Nonlinear motion model (error in θ causes banana-shaped belief)

Nonlinear observation model (*e.g.*, ring-shaped posterior)

Discrete hidden variables (don't know which object your laser bounced off of)

Algorithms for tracking

Lots!

All the ones I'll talk about are based on Markov equations for belief update

Can solve Markov equations exactly for small state spaces

- called the forward algorithm for HMMs
- familiar from speech recognition, text classification, etc.

Our problem is that state spaces are too big

Setup for tracking

State x_t, observation y_t

Assume x_t is sufficient for state of world

- *i.e.*, knowing x_t makes previous time steps irrelevant for predicting y_t and x_{t+1}
- called Markov assumption

Write $y_{(t)}$ for $y_1 \ldots y_t$

Tracking means computing $\mathrm{Bel}_t \equiv P(x_t | y_{(t)})$

Markov equations

Use Bayes' rule, law of total probability, Markov assumption:

$$
\begin{aligned}
\mathsf{Bel}_t &= P(x_t|y_{(t)}) \\
&= \alpha P(y_t|x_t, y_{(t-1)})P(x_t|y_{(t-1)}) \\
&= \alpha P(y_t|x_t, y_{(t-1)}) \int P(x_t|x_{t-1}, y_{(t-1)})P(x_{t-1}|y_{(t-1)})dx_{t-1} \\
&= \alpha P(y_t|x_t) \int P(x_t|x_{t-1})\mathsf{Bel}_{t-1}dx_{t-1}
\end{aligned}
$$

Markov tracking

To implement Markov tracking, must represent Bel_t in a way that allows *state propagation* and *evidence incorporation*

Propagation is $\widehat{\mathsf{Bel}}_t = \int P(x_t|x_{t-1})\mathsf{Bel}_{t-1}dx_{t-1}$ (uses *motion model*)

Incorporation is $\mathsf{Bel}_t = \alpha P(y_t|x_t)\widehat{\mathsf{Bel}}_t$ (uses *observation model*)

Kinds of Markov tracking

Will look at 2 rough groups: Kalman filtering, particle filtering

Kalman filters are smooth and deterministic

Particle filters are discontinuous and randomized

Kalman filter

Suppose motion and observation models are linear with Gaussian noise—that is,

$$\begin{aligned} x_{t+1} &= Fx_t + Gv_t \\ y_t &= Hx_t + Uw_t \end{aligned}$$

where v_t and w_t are independent unit-variance normal

Then we can solve Markov equations analytically

Resulting update is Kalman filter; involves lots of matrix multiplications which I won't repeat here

Kalman filter example

A robot with NSEW actions and GPS sensor

Extended Kalman filter

What if motion, observation models are "nearly" linear:

$$
\begin{aligned}
x_{t+1} &= f(x_t, v_t) \\
y_t &= h(x_t, w_t)
\end{aligned}
$$

where v_t and w_t are normal as before but f and h can be nonlinear (but must be near linear on scale of uncertainty in x, v, w)

Can linearize f, h around $(E(x_t), 0)$ then do Kalman filter

Called the extended Kalman filter

EKF example

Same as Kalman filter example, with different story

We have the correct motion model (including theta), but we linearized it

Sensors tell bearing and distance to a known point, but we linearized them

Particle filters

Kalman filters are fine if models are "nearly" linear and uncertainty is "small"

But if Bel_t is significantly non-Gaussian, we're stuck

So, represent Bel_t with weighted sample set instead

Can handle arbitrary (non-Gaussian, multi-modal) beliefs

But the more complicated or higher-dimensional the belief is, the more samples we need

The basic particle filter

Given:

  Bel_t a list of (x, p) pairs

  y an observation to incorporate

Normalize ps to sum to 1

$\text{Bel}_{t+1} := \phi$

Repeat $|\text{Bel}_t|$ times:

  Pick an x at random according to ps

  Pick state noise $v \sim N(0, I)$

  $x^+ := f(x, v)$

  $p^+ := P(y = g(x^+, w))$

  $\text{Bel}_{t+1} = \text{Bel}_{t+1}, (x^+, p^+)$

Basic particle filter cont'd

State propagation is sampling

Evidence incorporation is importance weighting

Bel_t is always a sample from true posterior: expected frequency of x_t is

$$P(x_t|x_{t-1})P(y_t|x_t)$$

(first term from sampling, second from weighting)

Extremely simple to implement

Particle filter example

Can use true motion and observation models

Results in non-Gaussian (bimodal) posterior

Particle filter example #2

Too much information gets us down to few particles ($500 \to 3$)

Particles induce spurious correlations in high-D

Franken-filters

(Extended) Kalman filters need near-Gaussian beliefs, but work well in high-dimensional spaces

Particle filters can have arbitrary beliefs, but need unreasonable numbers of particles for high-dimensional spaces

Can we get the best of both worlds?

Will describe three tricks:

- mixing together several filters
- models that depend on discrete hidden variables
- explicitly controlling size of particles

Combination does what we want

Mixing filters

Represent belief w/ mixture of Gaussians

Linearize around center of each one, use EKF equations separately

Only difficulty: mixture weights $p_{i,t}$

Depend on $P(y|i)$ (how well component i explains observation)

Mixture weights

$$p_{i,t} = p_{i,t-1} P(y|i)$$

$$P(y|i) = \int P(y|x,i)P(x|i)dx = \int N(y; \widehat{y}(x), \Sigma_{\mathsf{obs}})N(x; \mu_i, \Sigma_i)dx$$

If the two Gaussians overlap, $P(y|i)$ is big

Mixture weight equations

$$
\begin{aligned}
kP(x) &= \left|\frac{P}{2\pi}\right|^{\frac{1}{2}} e^{-\frac{1}{2}(x-\mu)^\top P(x-\mu)} \left|\frac{Q}{2\pi}\right|^{\frac{1}{2}} e^{-\frac{1}{2}(x-\nu)^\top Q(x-\nu)} \\
&= \left|\frac{P}{2\pi}\right|^{\frac{1}{2}} \left|\frac{Q}{2\pi}\right|^{\frac{1}{2}} e^{-\frac{1}{2}A} \\
A &= (x-\mu)^\top P(x-\mu) + (x-\nu)^\top Q(x-\nu) \\
&= x^\top(P+Q)x - 2x^\top(P\mu + Q\nu) + (\mu^\top P\mu + \nu^\top Q\nu) \\
R &= P + Q \\
\lambda &= R^{-1}(P\mu + Q\nu) \\
A &= (x-\lambda)R(x-\lambda) + (\mu^\top P\mu + \nu^\top Q\nu - \lambda^\top R\lambda) \\
\int_X e^{-\frac{1}{2}x^\top Rx} dx &= \left|\frac{R}{2\pi}\right|^{-\frac{1}{2}}
\end{aligned}
$$

$$k = \left|\frac{P}{2\pi}\right|^{\frac{1}{2}} \left|\frac{Q}{2\pi}\right|^{\frac{1}{2}} \left|\frac{R}{2\pi}\right|^{-\frac{1}{2}} e^{-\frac{1}{2}(\mu^\top P\mu+\nu^\top Q\nu-\lambda^\top R\lambda)}$$

Mixture of EKFs example

Better than using just one EKF

Still linearizes too much

Mixture Kalman filters

Suppose we have some discrete hidden variables in motion, observation models

For example, which object did we see?

Suppose models are conditionally linear/Gaussian:

$$\begin{aligned} x_{t+1} &= f(x_t, v_t, z_t) \\ y_t &= h(x_t, w_t, z_t) \end{aligned}$$

with f, h linear in x, v, w when z is fixed

MKFs cont'd

If we knew z_t for all t, could just use Kalman filter

Don't know z_t, so each Gaussian at step $t-1$ produces several children at step t

One mixture component for each possible history $z_{(t)}$

Mixture weights calculated as before, scaled by $P(z_{(t)})$

MKFs cont'd

In exact posterior, number of mixture components grows exponentially

So throw away some mixture components by sampling

Looks like a particle filter but individual particles are Gaussians

Nonlinearity

What if models are nonlinear even once hidden variables are known?

If only part of model is nonlinear, can treat that part as another hidden variable to be sampled ("extended mixture Kalman filter" – only difference from MKF is whether you want to track the hidden variables)

Or, can linearize around center of each particle (MKF article doesn't suggest this, don't know why – call it "mixture extended Kalman filter")

MEKF example

Uses true state, action models

Robot goes forward enough to have substantial uncertainty, sees 1 of 2 obstacles, produces bimodal posterior

Note inaccurate motion model, inaccurate incorporation of observation

Inaccurate for same reason M of EKF is inaccurate—linearization is too severe

Problems with MEKF

EKF update always picks a single point to linearize around

Can cause significant inaccuracy if uncertainty is high

- because of prior state uncertainty
- because of inherent noise in model

Can happen even if true posterior is nearly Gaussian (*e.g.*, if motion model has non-Gaussian uncertainty which is then removed by observation)

Controlling particle size

Motion model: want to approximate distribution of $f(x, v)$ given distribution of x

EKF linearizes f around $E(x)$

Particle filter samples x, v then "linearizes" around each sample

Controlling particle size

What if we approximate dist'n of x, v by mixture of Gaussians and linearize around each one?

Makes Gaussians smaller, so less affected by nonlinearity

Can interpolate between PF and EKF behavior

Similar trick for observation model

Call this "hybrid Kalman particle filter"

Aside: mixtures of Gaussians

A Gaussian mixture of Gaussians is Gaussian

That is, if you pick a random Gaussian displacement from some mean, then construct a new Gaussian centered on that point with a given covariance, then repeat infinitely many times, you've produced a mixture of Gaussians which is itself a Gaussian

Covariance of the final Gaussian is the sum of the displacement's covariance and the new Gaussian's covariance

So we can take any Gaussian particle and decompose it into a mixture of Gaussians

Aside cont'd

That's how HKPF splits up its particles to make them smaller

There are infinitely many ways to make the split; I took the simple one of splitting the original covariance Σ into $\alpha\Sigma$ and $(1-\alpha)\Sigma$ (here α is a free parameter controlling how small we want to keep our particles)

HKPF example

Same as MEKF example but works better

Final HKPF algorithm

Given:

  Bel_t a list of $(\bar{x}, \Sigma, p)$ pairs

  y an observation to incorporate

Normalize ps to sum to 1

$\text{Bel}_{t+1} := \phi$

Repeat $|\text{Bel}_t|$ times:

  Pick an $\bar{x}, \Sigma$ at random according to ps

  Pick $x \sim N(\bar{x}, \alpha\Sigma)$

  Pick $v \sim N(0, \alpha I)$

  $(F, G) := (\frac{\partial f}{\partial x}, \frac{\partial f}{\partial v})|_{x,v}$

  $\bar{x}^+ := f(x, v)$

  $\Sigma^+ :=$ Kalman forward update of $(1 - \alpha)\Sigma$ for F, G

Pick $w \sim N(0, \alpha I)$
Pick z according to $P(z|\bar{x}^+, \Sigma^+)$
$(H, U) := (\frac{\partial h}{\partial x}, \frac{\partial h}{\partial w})|_{\bar{x}^+, w, z}$
$\bar{x}^{++}, \Sigma^{++}$:= Kalman observation update of $\bar{x}^+, \Sigma^+$ using H, U
p^+ := Kalman evidence
$\mathrm{Bel}_{t+1} = \mathrm{Bel}_{t+1}, (\bar{x}^{++}, \Sigma^{++}, p^+)$

Note: above description of how to select z is oversimplified, we usually need to use knowledge of problem to come up with a better proposal distribution

Review

Kalman filters and particle filters cover different parts of design space:

- Linear: Kalman filter
- Locally linear: EKF
- Low-dimensional: particle filter

Review

For non-Gaussian beliefs in more than a few dimensions, must use a combination ("Rao-Blackwellized particle filters")

- Linear with mixture prior: M of KF
- Locally linear with mixture prior: M of EKF
- Conditionally linear: MKF or EMKF
- Conditionally locally linear: MEKF
- Not linear enough for MEKF: HKPF

Conclusion

For a given mode, a few Gaussians fill it better than a lot of particles

But need particles for keeping track of distinct modes, handling nonlinearity

HKPFs should let us solve our tracking problem

Future research:

- integrating with our robots
- sparse covariance matrices for efficient representation of a few correlations
- scaling up further—maybe whole maps could be particles

Interesting filters I didn't mention

Unscented Kalman filters [Julier & Uhlmann]

Different proposal distributions in particle filters:

- proposing from the observation [Thrun]
- extended Kalman particle filters
- unscented particle filters [van der Merwe et al]

Conditional particle filters [Montemerlo et al]

Map learning w/ RBPFs [Murphy et al]