Computer Vision 2
WS 2018/19

Part 8 – Beyond Kalman Filters
13.11.2018

Prof. Dr. Bastian Leibe
RWTH Aachen University, Computer Vision Group
http://www.vision.rwth-aachen.de

Course Outline

- Single-Object Tracking
- Bayesian Filtering
  - Kalman Filters, EKF
  - Particle Filters
- Multi-Object Tracking
- Visual Odometry
- Visual SLAM & 3D Reconstruction
- Deep Learning for Video Analysis

Topics of This Lecture

- Recap: Kalman Filter
  - Basic ideas
  - Kalman filter for 1D state
  - General Kalman filter
  - Limitations
  - Extensions
- Particle Filters
  - Basic ideas
  - Propagation of general densities
  - Factored sampling

Today: Beyond Gaussian Error Models

Recap: Tracking as Inference

• The hidden state consists of the true parameters we care about, denoted \( X \).
• The measurement is our noisy observation that results from the underlying state, denoted \( Y \).
• At each time step, state changes (from \( X_{t-1} \) to \( X_t \)) and we get a new observation \( Y_t \).

• Our goal: recover most likely state \( X_t \), given
  - All observations seen so far.
  - Knowledge about dynamics of state transitions.

Recap: Tracking as Induction

• Base case:
  - Assume we have initial prior that predicts state in absence of any evidence: \( f(X_0) \)
  - At the first frame, correct this given the value of \( Y_0 = y_0 \)
• Given corrected estimate for frame \( t \):
  - Predict for frame \( t+1 \)
  - Correct for frame \( t+1 \)
Recap: Prediction and Correction

- **Prediction:**
  \[ P(X_t | y_{t-1}, y_{t-2}, \ldots, y_0) = \int P(X_t | X_{t-1})P(X_{t-1} | y_{t-1}, y_{t-2}, \ldots, y_0) \, dX_{t-1} \]

- **Correction:**
  \[ P(X_t | y_0, \ldots, y_T) = \frac{P(Y_T | X_T)P(X_T | y_0, \ldots, y_T)}{P(Y_T | y_0, \ldots, y_T)} \]

Recap: Linear Dynamic Models

- **Dynamics model**
  - State undergoes linear transformation \( D \), plus Gaussian noise
  \[ x_t \sim N(Dx_{t-1}, \Sigma_d) \]

- **Observation model**
  - Measurement is linearly transformed state plus Gaussian noise
  \[ y_t \sim N(Mx_t, \Sigma_m) \]

Example: Constant Velocity (1D Points)

- State vector: position \( p \) and velocity \( v \)
  \[ x_t = \begin{bmatrix} p_t \\ v_t \end{bmatrix}, \quad p_t = p_{t-1} + (M)v_{t-1} + \varepsilon \\
  v_t = v_{t-1} + \xi \]
  \[ x_t = Dx_{t-1} + \text{noise} = \begin{bmatrix} 1 & \Delta T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p_{t-1} \\ v_{t-1} \end{bmatrix} + \text{noise} \]

  - Measurement is position only
  \[ y_t = Mx_t + \text{noise} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} p_t \\ v_t \end{bmatrix} + \text{noise} \]

Example: Constant Acceleration (1D Points)

- State vector: position \( p \) and velocity \( v \)
  \[ x_t = \begin{bmatrix} p_t \\ v_t \end{bmatrix}, \quad p_t = p_{t-1} + v_{t-1} + \varepsilon \\
  v_t = v_{t-1} + \Delta \varepsilon \]
  \[ x_t = Dx_{t-1} + \text{noise} = \begin{bmatrix} 1 & \Delta T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p_{t-1} \\ v_{t-1} \end{bmatrix} + \text{noise} \]

  - Measurement is position only
  \[ y_t = Mx_t + \text{noise} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} p_t \\ v_t \end{bmatrix} + \text{noise} \]
Beyond Kalman Filters

The calculations are easy (all the integrals can be done in closed form).

Substitute variables to transform this into linear system.

Measurement is position only.

The predicted/corrected state distributions are Gaussian.

You only need to maintain the mean and covariance.

Example: Constant Acceleration (1D Points)

- State vector: position \( p \), velocity \( v \), and acceleration \( a \).

\[
\begin{bmatrix}
  p_i \\
  v_i \\
  a_i
\end{bmatrix}
= \begin{bmatrix}
  p_{i-1} + (\Delta t)v_{i-1} + \frac{1}{2}(\Delta t)^2a_{i-1} + \xi \\
  v_{i-1} + (\Delta t)a_{i-1} + \zeta \\
  a_{i-1} + \xi + \zeta
\end{bmatrix}
\]

(greek letters denote noise terms)

\[
x_i = D_x x_{i-1} + \text{noise} = \begin{bmatrix} 1 & \Delta t & \frac{1}{2}(\Delta t)^2 \end{bmatrix} \begin{bmatrix} p_{i-1} \\
  v_{i-1} \\
  a_{i-1}
\end{bmatrix} + \text{noise}
\]

- Measurement is position only.

\[
y_i = M_x y_i = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_i \\
  v_i \\
  a_i
\end{bmatrix} + \text{noise}
\]

General Motion Models

- Assuming we have differential equations for the motion
  
  - E.g. for (undamped) periodic motion of a linear spring

\[
\frac{d^2 p}{dt^2} = -p
\]

- Substitute variables to transform this into linear system

\[
\begin{bmatrix}
  p_1 \\
  p_2 \\
  p_3
\end{bmatrix}
= \begin{bmatrix}
  p_{i-1} + (\Delta t)p_{i-2} + \frac{1}{2}(\Delta t)^2p_{i-3} + \xi \\
  p_{i-2} + (\Delta t)p_{i-3} + \zeta \\
  p_{i-3} + \xi + \zeta
\end{bmatrix}
\]

Then we have

\[
x_i = \begin{bmatrix} 1 & \Delta t & \frac{1}{2}(\Delta t)^2 \end{bmatrix} \begin{bmatrix} p_{i-1} \\
  p_{i-2} \\
  p_{i-3}
\end{bmatrix}
\]

The Kalman Filter

- Kalman filter
  
  - Method for tracking linear dynamical models in Gaussian noise

- The predicted/corrected state distributions are Gaussian
  
  - You only need to maintain the mean and covariance.
  
  - The calculations are easy (all the integrals can be done in closed form).

Kalman Filter for 1D State

- Want to represent and update

\[
P(x_i|y_0,\ldots,y_{i-1}) = N(\mu^+_i,(\sigma_i^+)^2)
\]

\[
P(x_i|y_0,\ldots,y_i) = N(\mu^+_i,(\sigma_i^+)^2)
\]

The Kalman Filter

Know corrected state from previous time step, and all measurements up to the current one.

- Predict distribution over next state.

\[
P(x_{i+1}|y_0,\ldots,y_i)
\]

Time advances: \( t \to t+1 \)

Mean and std. dev. of predicted state:

\[
\mu^+_i,\sigma^+_i
\]

Propagation of Gaussian densities

Shifting the mean

Bayesian update

Increasing the variance

Bayesian update
1D Kalman Filter: Prediction

- Have linear dynamic model defining predicted state evolution, with noise:
  \[ X_t = N(dx_{t-1}, \sigma^2) \]
- Want to estimate predicted distribution for next state
  \[ P(X_t | y_0, \ldots, y_{t-1}) = N(\mu_t^*, (\sigma^*_t)^2) \]
- Update the mean:
  \[ \mu_t^* = d\mu_{t-1} \]
- Update the variance:
  \[ (\sigma^*_t)^2 = (\sigma^*_{t-1})^2 + (d\sigma^*_{t-1})^2 \]

1D Kalman Filter: Correction

- Have linear model defining the mapping of state to measurements:
  \[ Y_t = N(mx_t, \sigma^2_m) \]
- Want to estimate corrected distribution given latest measurement:
  \[ P(X_t | y_0, \ldots, y_t) = N(\mu_t^*, (\sigma^*_t)^2) \]
- Update the mean:
  \[ \mu_t^* = \frac{\mu_t^* \sigma^2_m + m y_t (\sigma^*_t)^2}{\sigma^2_m + m^2 (\sigma^*_t)^2} \]
- Update the variance:
  \[ (\sigma^*_t)^2 = \frac{\sigma^2_m (\sigma^*_t)^2}{\sigma^2_m + m^2 (\sigma^*_t)^2} \]

Prediction vs. Correction

\[ \mu_t^* = \frac{\mu_t^* \sigma^2_m + m y_t (\sigma^*_t)^2}{\sigma^2_m + m^2 (\sigma^*_t)^2} \]
\[ (\sigma^*_t)^2 = \frac{\sigma^2_m (\sigma^*_t)^2}{\sigma^2_m + m^2 (\sigma^*_t)^2} \]
- What if there is no prediction uncertainty \((\sigma^*_{t-1})^2 = 0\)?
  \[ \mu_t^* = \mu_{t-1} \]
  The measurement is ignored!
- What if there is no measurement uncertainty \((\sigma^*_m)^2 = 0\)?
  \[ \mu_t^* = \frac{y_t}{m} \]
  The prediction is ignored!

Recall: Constant Velocity Example

State is 2D: position + velocity
Measurement is 1D: position

Constant Velocity Model

- o state
- x measurement
- * predicted mean estimate
- + corrected mean estimate
- bars: variance estimates before and after measurements

Constant Velocity Model

- o state
- x measurement
- * predicted mean estimate
- + corrected mean estimate
- bars: variance estimates before and after measurements
### Constant Velocity Model

- **x** measurement
  - * predicted mean estimate
  - + corrected mean estimate
- Bars: variance estimates before and after measurements

### Kalman Filter: General Case (>1dim)

**Predict**

- $\hat{x}_t^p = D_x^t \hat{x}_t^{p-1}$
- $\Sigma_t^p = D_x^t \Sigma_{x_t}^p D_x^t + \Sigma_z^p$

**Correct**

- $K_t^c = \Sigma_t^c M_t^c (M_t^c \Sigma_t^c M_t^c + \Sigma_m^c)^{-1}$
- $\hat{x}_t = \hat{x}_t^{p} + K_t^c (y_t - M_t^c \hat{x}_t^{p})$
- $\Sigma_t^c = (I - K_t^c M_t^c) \Sigma_t^p$

For derivations, see F&P Chapter 17.3

### Remarks

- **Try it!**
  - Not too hard to understand or program
- **Start simple**
  - Experiment in 1D
  - Make your own filter in Matlab, etc.
- **Note: the Kalman filter “wants to work”**
  - Debugging can be difficult
  - Errors can go unnoticed

### Summary: Kalman Filter

- **Pros:**
  - Gaussian densities everywhere
  - Simple updates, compact and efficient
  - Very established method, very well understood
- **Cons:**
  - Unimodal distribution, only single hypothesis
  - Restricted class of motions defined by linear model

### Topics of This Lecture

- **Recap: Kalman Filter**
  - Basic ideas
  - Kalman filter for 1D state
  - General Kalman filter
  - Limitations
  - Extensions
- **Particle Filters**
  - Basic ideas
  - Propagation of general densities
  - Factored sampling
Extension: Extended Kalman Filter (EKF)

• Basic idea
  – State transition and observation model don’t need to be linear functions of the state, but just need to be differentiable.
    \[ x_t = g(x_{t-1}, v_t) \]
    \[ y_t = h(x_t) + \delta_t \]
  – The EKF essentially linearizes the nonlinearity around the current estimate by a Taylor expansion.

• Properties
  – Unlike the linear KF, the EKF is in general not an optimal estimator.
    • If the initial estimate is wrong, the filter may quickly diverge.
    • Still, it’s the de-facto standard in many applications
      • Including navigation systems and GPS

Recap: Kalman Filter – Detailed Algorithm

• Algorithm summary
  – Assumption: linear model
    \[ x_t = A x_{t-1} + v_t \]
    \[ y_t = C x_t + w_t \]
  – Prediction step
    \[ x_t^* = A x_{t-1} + K_t y_t \]
    \[ \Sigma_t^* = A \Sigma_{t-1} A^T + \Sigma_v \]
  – Correction step
    \[ K_t = \Sigma_t^* H_t^T (H_t \Sigma_t^* H_t^T + \Sigma_w)^{-1} \]
    \[ \Sigma_t^* = (I - K_t H_t) \Sigma_t^* \]

Extended Kalman Filter (EKF)

• Algorithm summary
  – Nonlinear model
    \[ x_t = g(x_{t-1}) + \delta_t \]
    \[ y_t = h(x_t) + \delta_t \]
  – Prediction step
    \[ x_t^* = g(x_{t-1}^*) \]
    \[ \Sigma_t^* = G_t \Sigma_{t-1}^* G_t^T + \Sigma_v \]
    \[ G_t = \frac{\partial g(x)}{\partial x} \mid_{x=x_t} \]
  – Correction step
    \[ K_t = \Sigma_t^* H_t^T (H_t \Sigma_t^* H_t^T + \Sigma_w)^{-1} \]
    \[ x_t^* = x_t^* + K_t (y_t - h(x_t^*)) \]
    \[ \Sigma_t^* = (I - K_t H_t) \Sigma_t^* \]

Kalman Filter – Other Extensions

• Unscented Kalman Filter (UKF)
  – Used for models with highly nonlinear predict and update functions.
  – Here, the EKF can give very poor performance, since the covariance is propagated through linearization of the non-linear model.
  – Idea (UKF): Propagate just a few sample points ("sigma points") around the mean exactly, then recover the covariance from them.
  – More accurate results than the EKF’s Taylor expansion approximation.

• Ensemble Kalman Filter (EnKF)
  – Represents the distribution of the system state using a collection (an ensemble) of state vectors.
  – Replace covariance matrix by sample covariance from ensemble.
  – Still basic assumption that all prob. distributions involved are Gaussian.
  – EnKFs are especially suitable for problems with a large number of variables.

甚至更多扩展

Even More Extensions

• Switching linear dynamical system (SLDS)
  \[ z_t \sim \pi_{z_{t-1}} \]
  \[ y_t = A(z_{t-1}) x_{t-1} + v_t \]
  \[ y_t = C z_t + v_t \]
  \[ z_t \sim N(0, \Sigma(z_{t-1})) \]

• Switching Linear Dynamic System (SLDS)
  – Use a set of \( k \) dynamic models \( A^{(1)}, \ldots, A^{(k)} \), each of which describes a different dynamic behavior.
  – Hidden variable \( z_t \) determines which model is active at time \( t \).
  – A switching process can change \( z_t \) according to distribution \( \pi_{z_{t-1}} \)

Topics of This Lecture

• Recap: Kalman Filter
  – Basic ideas
  – Kalman filter for 1D state
  – General Kalman filter
  – Limitations
  – Extensions

• Particle Filters
  – Basic ideas
  – Propagation of general densities
  – Factored sampling

Today: only main ideas
Formal introduction next lecture
When Is A Single Hypothesis Too Limiting?

- Consider this example: say we are tracking the face on the right using a skin color blob to get our measurement.

### Propagation of General Densities

**Factored Sampling**

- Idea: Represent state distribution non-parametrically
  - Prediction: Sample points from prior density for the state, \( P(X_t) \)
  - Correction: Weight the samples according to \( P(Y_t | X_t) \)

\[
P(X_t | y_1, \ldots, y_t) = \frac{P(y_t | X_t) P(X_t | y_1, \ldots, y_{t-1})}{\int P(y_t | X_t) P(X_t | y_1, \ldots, y_{t-1}) dX_t}
\]

### Particle Filtering

- Many variations, one general concept:
  - Represent the posterior pdf by a set of randomly chosen weighted samples (particles)

- Randomly Chosen = Monte Carlo (MC)
- As the number of samples become very large – the characterization becomes an equivalent representation of the true pdf.

- Idea:
  - We want to use sampling to propagate densities over time (i.e., across frames in a video sequence).
  - At each time step, represent posterior \( P(X_t | Y_t) \) with weighted sample set.
  - Previous time step’s sample set \( P(X_{t-1} | Y_{t-1}) \) is passed to next time step as the effective prior.
Particle Filtering – Visualization

Code and video available from http://www.robots.ox.ac.uk/~misard/condensation.html

Particle Filtering Results

• Some more examples

http://www.robots.ox.ac.uk/~misard/condensation.html

Obtaining a State Estimate

• Note that there’s no explicit state estimate maintained, just a “cloud” of particles
• Can obtain an estimate at a particular time by querying the current particle set
• Some approaches
  − “Mean” particle
  − Weighted sum of particles
  − Confidence: inverse variance
  − Really want a mode finder—mean of tallest peak

Condensation: Estimating Target State

State samples (thickness proportional to weight) Mean of weighted state samples

Summary: Particle Filtering

• Pros:
  − Able to represent arbitrary densities
  − Converging to true posterior even for non-Gaussian and nonlinear system
  − Efficient: particles tend to focus on regions with high probability
  − Works with many different state spaces
    • E.g. articulated tracking in complicated joint angle spaces
  − Many extensions available
Summary: Particle Filtering

- **Cons / Caveats:**
  - #Particles is important performance factor
    - Want as few particles as possible for efficiency.
    - But need to cover state space sufficiently well.
  - Worst-case complexity grows exponentially in the dimensions
    - Multimodal densities possible, but still single object
    - Interactions between multiple objects require special treatment.
    - Not handled well in the particle filtering framework (state space explosion).

References and Further Reading

- A good tutorial on Particle Filters

- The CONDENSATION paper