2D Image Processing
(Extended) Kalman and particle filter

Prof. Didier Stricker
Kaiserlautern University
http://ags.cs.uni-kl.de/

DFKI – Deutsches Forschungszentrum für Künstliche Intelligenz
http://av.dfki.de

Some slides based on S. Thrun and K. Smith
Outline

- Recap: Kalman filter algorithm
- Extended Kalman filter algorithm
- Particle Filter
Recap: linear Gaussian state-space model

- **Motion model:**
  - Linear stochastic difference equation in $x$
  - Evolution of $x_t$ based on previous state $x_{t-1}$ and control input $u_t$

  $$x_t = A_t x_{t-1} + B_t u_t + \varepsilon_t \iff p(x_t \mid x_{t-1}, u_t) = N(x_t; A_t x_{t-1} + B_t u_t, R_t)$$

  $$\varepsilon_t \sim N(0, R_t)$$

- **Measurement model:**
  - Linear stochastic equation in $x$
  - Describes, how measurements $z_t$ are related to the state

  $$z_t = C_t x_t + \delta_t \iff p(z_t \mid x_t) = N(z_t; C_t x_t, Q_t)$$

  $$\delta_t \sim N(0, Q_t)$$
Recap: Kalman filter

- Bayes update rule + linear Gaussian models
  → Kalman filter

\[
bel(x_t) = \eta \ p(z_t|x_t) \int p(x_t|x_{t-1}, u_t) \ bel(x_{t-1}) \ dx_{t-1}
\]

\[
\begin{align*}
N(x_t; \mu_t, \Sigma_t) & \quad \text{posterior} \\
N(z_t; C_t x_t, Q_t) & \quad \text{likelihood} \\
N(x_t; A_t x_{t-1} + B_t u_t, R_t) & \quad \text{motion model} \\
N(x_{t-1}; \mu_{t-1}, \Sigma_{t-1}) & \quad \text{posterior at } t-1
\end{align*}
\]
Recap: Kalman filter algorithm

1. **Kalman** filter$(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$:

2. **Prediction**:  
   \[
   \mu_t = A_t \mu_{t-1} + B_t u_t
   \]
   \[
   \Sigma_t = A_t \Sigma_{t-1} A_t^T + R_t
   \]

3. **Correction**:  
   \[
   K_t = \Sigma_t C_t^T (C_t \Sigma_t C_t^T + Q_t)^{-1}
   \]
   \[
   \mu_t = \mu_t + K_t(z_t - C_t \mu_t)
   \]
   \[
   \Sigma_t = (I - K_t C_t) \Sigma_t
   \]

4. **Requirements**:
   - Initial belief is Gaussian
   - Linear Gaussian state-space model

9. **Return** $\mu_t, \Sigma_t$
Constant velocity model

Assuming that the object follows a const. velocity model:

😊 Better prediction
😊 Still problems in case of accelerations
😊 Cont’d estimate, even if we don’t have measurements
Kalman filter limitations

- Applies to linear Gaussian models
- Many visual tracking problems are nonlinear, e.g. as soon as we move to tracking in 3D
Humanoid robot catches flying balls

- Several cameras observe flying balls
- The 3D trajectory is estimated
- A robot is supposed to catch the balls
  → nonlinear estimation problem

Courtesy of U. Frese
Extended Kalman filter

Slides based on S. Thrun
Nonlinear state-space model, Gaussian noise

- **Motion model:**
  \[ x_t = A_t x_{t-1} + B_t u_t + \varepsilon_t \Rightarrow g(x_{t-1}, u_t, \varepsilon_t) \]
  \[ \varepsilon_t \sim N(0, R_t) \]

- **Measurement model:**
  \[ z_t = C_t x_t + \delta_t \Rightarrow z_t = h(x_t, \delta_t) \]
  \[ \delta_t \sim N(0, Q_t) \]
  \[ z_t = h(x_t) + \delta_t \]
Propagate Gaussian through linear function

\[ X \sim N(\mu, \Sigma) \]
\[ Y = AX + B \]
\[ \Rightarrow Y \sim N(A\mu + B, A\Sigma A^T) \]
Propagate Gaussian through nonlinear function

- PDF obtained from 500,000 Monte Carlo samples + histogramming
- Then sample mean and covariance
Bayes update rule

- **Problem:**
  We do not stay in the Gaussian world, if motion and/or measurement models are nonlinear functions of the state

\[
bel(x_t) = \eta \ p(z_t|x_t) \int p(x_t|x_{t-1}, u_t) \ bel(x_{t-1}) \ dx_{t-1}
\]

- **No general closed-form solution for Bayes filter**
- **Approximate solutions:**
  - Keep the functions and approximate distributions \(\mapsto\) Unscented Kalman filter, particle filter
  - Linearize the functions and use again the Kalman filter
EKF linearization

Monte Carlo transform + sample statistics → blue Gaussian
First Order Taylor approximation → red Gaussian

Mismatch = linearization error
First order Taylor approximation (multivariate)

\[ y = f(x) \approx f(a) + \frac{\partial f(a)}{\partial x} (x - a) \]

With vectors:

\[ y = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix}, \quad x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}, \quad a = \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix} \]

- **Jacobian**: matrix of all 1st-order partial derivatives of vector/scalar-valued function with respect to another vector. Let: \( f : \mathbb{R}^n \rightarrow \mathbb{R}^m \) with component functions

\[ y_1 = f_1(x_1, \ldots, x_n), \ldots, y_m = f_m(x_1, \ldots, x_n) \]

\[ \frac{\partial f}{\partial x} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix} \]
EKF linearization of state-space model (additive noise)

- Linearize $g()$ and $h()$ with First Order Taylor Expansion
- Linearization points: best available estimate
- Prediction (linearize around $\mu_{t-1}$):
  \[
g(u_t, x_{t-1}) \approx g(u_t, \mu_{t-1}) + \frac{\partial g(u_t, \mu_{t-1})}{\partial x_{t-1}} (x_{t-1} - \mu_{t-1})
  
  g(u_t, x_{t-1}) \approx g(u_t, \mu_{t-1}) + G_t (x_{t-1} - \mu_{t-1})
  \]

- Correction (linearize around $\bar{\mu}_t$):
  \[
h(x_t) \approx h(\bar{\mu}_t) + \frac{\partial h(\bar{\mu}_t)}{\partial x_t} (x_t - \bar{\mu}_t)
  
  h(x_t) \approx h(\bar{\mu}_t) + H_t (x_t - \bar{\mu}_t)
  \]
Extended Kalman filter: additive noise

1. **Extended_Kalman_filter**($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):

2. **Prediction**:

   3. $\bar{\mu}_t = g(u_t, \mu_{t-1})$
   
   4. $\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$

3. $\mu_t = A_t \mu_{t-1} + B_t u_t$

4. $\Sigma_t = A_t \Sigma_{t-1} A_t^T + R_t$

5. **Correction**:

   6. $K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$

   7. $\mu_t = \bar{\mu}_t + K_t (z_t - h(\bar{\mu}_t))$

   8. $\Sigma_t = (I - K_t H_t) \bar{\Sigma}_t$

9. **Return** $\mu_t, \Sigma_t$

   \[ H_t = \frac{\partial h(\bar{\mu}_t)}{\partial x_t} \quad G_t = \frac{\partial g(u_t, \mu_{t-1})}{\partial x_{t-1}} \]
Linearization of state-space model (non-additive noise)

- **Linearization points**: best estimate and 0 (assuming zero-mean noise)
  - **Prediction** (linearize around $\mu_{t-1}$ and 0):
    \[
    g(u_t, x_{t-1}, \varepsilon_t) \approx g(u_t, \mu_{t-1}, 0) + \frac{\partial g(u_t, \mu_{t-1}, 0)}{\partial x_{t-1}} (x_{t-1} - \mu_{t-1}) + \frac{\partial g(u_t, \mu_{t-1}, 0)}{\partial \varepsilon_t} \varepsilon_t
    \]
    \[
    g(u_t, x_{t-1}, \varepsilon_t) \approx g(u_t, \mu_{t-1}) + G_t (x_{t-1} - \mu_{t-1}) + W_t \varepsilon_t
    \]

- **Correction** (linearize around $\bar{\mu}_t$ and 0):
  \[
  h(x_t, \delta_t) \approx h(\bar{\mu}_t, 0) + \frac{\partial h(\bar{\mu}_t, 0)}{\partial x_t} (x_t - \bar{\mu}_t) + \frac{\partial h(\bar{\mu}_t, 0)}{\partial \delta_t} \delta_t
  \]
  \[
  h(x_t, \delta_t) \approx h(\bar{\mu}_t, 0) + H_t (x_t - \bar{\mu}_t) + V_t \delta_t
  \]
Gauss approximation formula (1st order) derived

- Approximate a nonlinear transform using a first order Taylor approx.:
  - $p(x)$, a PDF with mean $\mu_x$ and covariance $\Sigma_x$
  - $y = f(x)$, a nonlinear function of $x$

The 1st order Gauss approximation for mean $\mu_y$ and covariance $\Sigma_y$ is:

$$\mu_y = E(y) = E(f(x)) \approx E\left(f(\mu_x) + \frac{\partial f(\mu_x)}{\partial x} (x - \mu_x)\right)$$

$$= E\left(f(\mu_x)\right) + \frac{\partial f(\mu_x)}{\partial x} E(x - \mu_x) = f(\mu_x)$$

$$\Sigma_y = cov(y) = cov(f(x)) = E\left((f(x) - \mu_y)(f(x) - \mu_y)^T\right)$$

$$\approx E\left(\left(f(\mu_x) + \frac{\partial f(\mu_x)}{\partial x} (x - \mu_x) - f(\mu_x)\right)(\mu_y - \mu_x)\right)^T$$

$$= \frac{\partial f(\mu_x)}{\partial x} E\left(\frac{(x - \mu_x)(x - \mu_x)^T}{\Sigma_x}\right) \frac{\partial f(\mu_x)}{\partial x}^T$$
Extended Kalman filter: non-additive noise

1. **Extended_Kalman_filter**($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):

2. **Prediction:**
   3. $\overline{\mu}_t = g(u_t, \mu_{t-1})$
   4. $\overline{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + W_t R_t W_t^T$

5. **Correction:**
   6. $K_t = \overline{\Sigma}_t H_t^T (H_t \overline{\Sigma}_t H_t^T + V_t Q_t V_t^T)^{-1}$
   7. $\mu_t = \overline{\mu}_t + K_t (z_t - h(\overline{\mu}_t))$
   8. $\Sigma_t = (I - K_t H_t) \overline{\Sigma}_t$

9. **Return** $\mu_t, \Sigma_t$

$$G_t = \frac{\partial g(u_t, \mu_{t-1}, 0)}{\partial x_{t-1}}$$

$$W_t = \frac{\partial g(u_t, \mu_{t-1}, \varepsilon_t)}{\partial \varepsilon_t}$$

$$H_t = \frac{\partial h(\overline{\mu}_t, 0)}{\partial x_t}$$

$$V_t = \frac{\partial h(\overline{\mu}_t, \delta)}{\partial \delta_t}$$
EKF linearization: problems

- Bigger uncertainty → bigger linearization error
EKF linearization: problems
EKF linearization: problems

- Higher local nonlinearity $\rightarrow$ bigger linearization error
EKF linearization: problems
(Extended) Kalman filter: limitations

No multimodal distributions
(Extended) Kalman filter: limitations

- Unimodal distributions fail for unpredicted motion

Prediction too far from actual location to recover
Extended Kalman filter: summary

- Motion and measurement functions are linearized, hence approximated $\Rightarrow$ Not optimal (we keep Gaussian representations, but $bel(x)$ is not Gaussian)
- If the functions are highly nonlinear (difficult to quantize), the estimation may become unstable
- Good results, if functions are approx. linear around mean
- Less certain estimate is more affected by linearization errors

Kalman Filter was optimal (minimum variance estimator) for linear Gaussian models.
EKF is not an optimal Bayesian tracker!

- However, it is successfully used in many applications
Monte Carlo (particle) approximation

- How can we represent an arbitrary probability density?

\[ p(x) \approx \sum_{n=1}^{N} w_t^{(n)} \delta \left( x_t - x_t^{(n)} \right) \]

Non-parametric representation, as a set of weighted particles

\[ \left\{ x_t^{(n)}, w_t^{(n)} \right\}_{n=1}^{N}, w_t^{(n)} \in [0,1], \sum_n w_t^{(n)} = 1 \]

PDF of faces appearing in the image
Filter methods (rules-of-thumb)

Bayes Filter

- Kalman Filter
  - Linear Gaussian models
  - Nonlinear models, Gaussian noises
- Unscented Kalman Filter
- Extended Kalman Filter

Kalman Filter banks

- (Non)linear models, Gaussian noises, multimodal

Particle Filter

- Highly nonlinear models, non-Gaussian noises, multimodal
Particle filters

- Go by many names:
  - Sequential Monte Carlo Methods
  - Sequential importance resampling (SIR)
  - Bootstrap filters
  - Condensation trackers
  - Survival of the fittest

- Originally used for problems in
  - Statistics
  - Fluid mechanics
  - Statistical mechanics
  - Signal processing

- Introduced to the Computer Vision community by

  Michael Isard and Andrew Blake,
  CONDENSATION – Conditional Density Propagation for Visual Tracking,
Introductory example

- Mobile robot localization in a known environment based on motion (odometry) and sensing (door detection)
- The robot knows a map of the environment (corridor with 3 indistinguishable doors)
- Wanted: position of the robot in the map
Initially, the state is unknown.

Measurement $z_1$: here is a door.

Particles close to doors have higher weights.
Resampling:

Generate a new particle set by drawing particles (with replacement) with probabilities according to their weights.
(Particles with high weights will be duplicated)

Odometry input $u_1$: 1m forward.

Move and diffuse (with noise samples) particles according to the motion model.
Measurement $z_2$: here is a door.

Update weights by multiplying with measurement likelihood.

Resampling:

Generate a new particle set by drawing particles (with replacement) with probabilities according to their weights.
Particle filter algorithm: overview

- **Initialization:**
  - Randomly draw particles from the state-space.

- **Measurement update:**
  - Multiply weights with measurement likelihood (plausibility).

- **Resampling:**
  - Randomly draw new particles from the old set with probabilities proportional to the weights.

- **Time update:**
  - Move particles according to the motion model. This includes diffusing the particles by adding random noise.
Particle filter algorithm: overview

- **Question:** What does the robot know after each step?
- **Initialization:**
  - The robot knows nothing.
- **Measurement update:**
  - The robot knows that some hypotheses (particles) are more plausible.
- **Resampling:**
  - The robot knows the same, but in a different representation. It concentrates its resources on the more likely areas of the state-space.
- **Time update:**
  - The robot knows: „If I was previously here, I am now here“. However, uncertainty grows.
Particle filter

- **Advantages:**
  - Easy to implement.
  - Not restricted to Gaussian densities.
  - Handles highly nonlinear functions and multimodal distributions.
  - Very successful in practical applications.

- **Disadvantages:**
  - Needs many particles in high-dimensional state-spaces.
  - Precision highly depends on number of particles.

![Target distribution (Gaussian mixture)](image)
![Too few samples](image)
![Added samples](image)
Particle filter in more details

Along concrete tracking example

Slides based on K. Smith
Example: tracking based on color histograms

- Track girl in pink
What is a particle?

- A “sample” of the posterior

- Particles contain a
  - state estimate/hypothesis
  - weight

\[ s_t^n \triangleq (x_t^n, w_t^n) \]

- Summing the particles gives an approximation to the target distribution

\[ p(x_t | Z_t) \approx \sum_{n=1}^{N} w_t^n \delta(x_t - x_t^n) \]
What is a particle?

- Each particle contains a
  - state estimate
  - weight

\[ s_t^n \triangleq (x_t^n, w_t^n) \]

\[ x_t^n \triangleq \begin{pmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{pmatrix} \]

Const. velocity model, fixed bounding box size (could be improved by estimating also the bounding box size)
**SIR particle filter**

- **Begin** with weighted samples from t-1
  - **Resample:** draw samples according to \( \{w_{t-1}\}^{n=1:N} \)
- **Move:** apply motion model (no noise)
- **Diffuse:** apply noise to spread particles
- **Measure:** weights are assigned by likelihood response
- **Finish:** density estimate
**SIR particle filter**

- **Begin** with weighted samples from $t-1$
- **Resample**: draw samples according to $\{w_{t-1}\}_{n=1}^{N}$
- **Move**: apply motion model (no noise)
- **Diffuse**: apply noise to spread particles
- **Measure**: weights are assigned by likelihood response
- **Finish**: density estimate
SIR particle filter

- **Begin** with weighted samples from $t-1$

$$p(x_{t-1} | Z_{t-1}) \approx \sum_{n=1}^{N} w^n_{t-1} \delta(x_{t-1} - x^n_{t-1})$$
Previous estimate

- Receive posterior estimate from previous time step

\[ \{x_{t-1}^n, w_{t-1}^n\}_{n=1}^N \]

\[
p(x_{t-1} | Z_{t-1}) \approx \sum_{n=1}^N w_{t-1}^n \delta (x_{t-1} - x_{t-1}^n)
\]
**SIR particle filter**

- **Resample**: draw samples according to \( \{w_{t-1}\}^{n=1:N} \)

  N new samples are drawn from the previous set **with replacement**.

  New samples are assigned **uniform weights**.
Resample

- N new samples are drawn from the previous set with replacement to prevent degeneracy.
- Repeated samples occur by design.

\[
\begin{align*}
\{x_{n}^{t-1}, w_{n}^{t-1}\}_{n=1}^{N} & \quad \Rightarrow \quad \{x_{n}^{t}, w_{n}^{t}\}_{n=1}^{N} \\
\text{New sample set is given uniform weights}
\end{align*}
\]
Resample

- \( N \) new samples are drawn from the previous set with replacement to prevent degeneracy.
- Repeated samples occur by design.

\[
\begin{align*}
\{x_{t-1}^n, w_{t-1}^n\}_{n=1}^N & \quad \rightarrow \quad \{x_t^n, w_t^n\}_{n=1}^N \\
\text{Weighted sampling with replacement} & \quad \rightarrow \quad \text{New sample set is given uniform weights} \\
\text{Samples still in same place, but more likely ones are duplicated}
\end{align*}
\]
Degeneracy

- Failing to resample results in **degeneracy**.
  - Iteratively propagating the particles and assigning weights tends to make a few samples dominate the rest.
Resampling

- Given: Set \( S \) of weighted samples.

- Wanted: Random sample, where the probability of drawing \( x^i \) is given by \( w^i \).

- Typically done \( n \) times with replacement to generate new sample set \( S' \).
Naive solution: independent resampling

- Draw $n$ times independently from the particle set
- Like a roulette wheel
- **Question:** Where is the problem?
  - Two states $A, B$
  - Two particles $w_1 = w_2$
  - 1000 times resampling
  - What happens?
- One of the states is lost
- This is quite likely (50% per resampling)
- Filter believes that it knows the state
Better solution: systematic resampling

- Roulette wheel with $n$ pointers
- Linear time complexity
- Easy to implement, low variance
- Also called: stochastic universal sampling
Systematic resampling: algorithm

- Divide roulette wheel into numbers $0..\sum w_k$
- Distribute indices uniformly with interval $\sum \frac{w_k}{n}$
- First index points to random number $\alpha_0 \in \left[0..\sum \frac{w_k}{n}\right]$
- Next index $\alpha_i = \alpha_0 + i \sum \frac{w_k}{n}$
- Determine particle interval $j(i)$ according to index $\alpha_i$
  - First interval with cumulative weight $> \alpha_i$
  - $j(i) = \min\{j \mid \alpha_i < \sum_{k=0..j} w_k\}$
- Repeat for $i = 0..n-1$
Systematic resampling: pseudo code

1. Algorithm `systematic_resampling(S,n)`: 

2. \( S' = \emptyset, c_1 = w^1 \)  
   \text{Initialize empty particle set}

3. For \( i = 2 \ldots n \)  
   \( c_i = c_{i-1} + w^i \)  
   \text{Generate cumulative weights}

4. \( u_1 \sim U[0,n^{-1}], i = 1 \)  
   \text{Generate random number}

5. For \( j = 1 \ldots n \)  
   \text{Draw samples with replacement…}

6. While ( \( u_j > c_i \) )  
   \( i = i + 1 \)  
   \text{Search for 1st interval with cdf > } u_j

7. \( S' = S' \cup \{ x^i, n^{-1} > \} \)  
   \text{Insert into new set (normalized weight)}

8. \( u_{j+1} = u_j + n^{-1} \)  
   \text{Next index}

9. Return \( S' \)
SIR particle filter: predict

- Apply the (non-)linear motion model $p(x_t|x_{t-1}, u_t)$ to every particle!

\[ x_t = F_t x_{t-1} + B_t u_t + w_t \]
\[ x_t = f(x_{t-1}, u_t, w_t) \]

- **Move**: apply motion model (no noise)

- **Diffuse**: apply noise to spread particles
Motion model (here linear Gaussian)

- Apply the motion model $p(x_t|x_{t-1})$ to every particle!

$$x_t = F_{t}x_{t-1} + w_t$$

Linear motion noise model

$$
\begin{pmatrix}
  x_t \\
  y_t \\
  \dot{x}_t \\
  \dot{y}_t
\end{pmatrix} =
\begin{pmatrix}
  1 & \Delta t \\
  1 & \Delta t \\
  1 & \\
  1 & 
\end{pmatrix}
\begin{pmatrix}
  x_{t-1} \\
  y_{t-1} \\
  \dot{x}_{t-1} \\
  \dot{y}_{t-1}
\end{pmatrix} + w_t
$$

$w_t \sim N(0, Q_t)$
SIR particle filter: measure

- Obtain a measurement $z_t$ for each state estimate $x_t$.

- Evaluate likelihood that $x_t$ gave rise to $z_t$ using measurement model.

  $$ p(z_t \mid x_t) $$

- **Measure**: weights are proportional to the measurement likelihood

  $$ p(x_t \mid Z_t) $$
Measurement model

- Obtain measurement $z_t$ for each state estimate $x_t$. 
Measurement model

- Obtain measurement $z_t$ for each state estimate $x_t$. 
Measurement model (based on color histograms)

- Obtain measurement $z_t$ for each state estimate $x_t$.

LAB space:
- designed to approximate human vision
  - L $\Rightarrow$ lightness
  - A,B $\Rightarrow$ color

Measurements are obtained by converting pixel values within bounding boxes to LAB color space, and concatenating to form an AB channel histogram.
Measurement model

- Obtain measurement $z_t$ for each state estimate $x_t$.

- Evaluate likelihood that an $x_t$ gave rise to $z_t$ using measurement model.

$$p(z_t | x^n_t) \propto e^{-\lambda \text{dist}_{KL}(z_t,c)}$$

Measurement model compares $z_t$ to a known color model $c$ using the **Kullback-Leibler divergence**.
Measurement model

- Obtain measurement $z_t$ for each state estimate $x_t$.

- Evaluate likelihood that an $x_t$ gave rise to $z_t$ using measurement model.

\[ p(z_t|x_t^n) \propto e^{-\lambda \text{dist}_{KL}(z_t,c)} \]

Measurement model compares $z_t$ to a known color model $c$ using the Kullback-Leibler divergence.
Excursion: Kullback-Leibler divergence

- Non-symmetric measure of the difference between two probability distributions $P$ and $Q$
  - How much information is lost, if $Q$ is used to approximate $P$
  - $P$ assumed “true” distribution, $Q$ assumed approximation
- For discrete probability distributions (histograms can be interpreted as such):
  \[
  dist_{KL}(Q, P) = \sum_i \ln \left( \frac{P(i)}{Q(i)} \right) P(i)
  \]
- Likelihood function:
  \[
  p(z_t | x^n_t) \propto e^{-\lambda dist_{KL}(z_t, c)}
  \]

Determines, how peaked the likelihood function is
- Measured color model (histogram)
- Known color model (histogram)
Measurement model

- Obtain measurement $z_t$ for each state estimate $x_t$.

- Evaluate likelihood that an $x_t$ gave rise to $z_t$ using measurement model.

- Weights are updated with the likelihood response:
  $$w_t^n \propto p(z_t|x_t^n) \cdot w_{t-1}^n$$
Measurement model

- Obtain measurement $z_t$ for each state estimate $x_t$.

- Evaluate likelihood that an $x_t$ gave rise to $z_t$ using measurement model.

- Weights are updated with the likelihood response:
  
  $$w_t^n \propto p(z_t|x_t^n) \cdot w_{t-1}^n$$

- After resampling, this corresponds to:
  
  $$w_t^n \propto p(z_t|x_t^n)$$
SIR particle filter

- **Begin** with weighted samples from t-1
  - **Resample**: draw samples according to \( \{w_{t-1}\}^{n=1:N} \)
  - **Move**: apply motion model (no noise)
  - **Diffuse**: apply noise to spread particles
  - **Measure**: weights are assigned by likelihood response
- **Finish**: density estimate
Obtaining a solution

- So far, we do not have an explicit state estimate, we have a cloud of particles!

- How do we extract an answer? It depends…
  - Compute a **weighted mean**
  - Confidence: inverse variance
  - For discrete labels, this does not work!
    - Use the mode?
Obtaining a solution

- Weights assumed normalized (sum up to 1)
- Weighted sample mean:
  \[ \mu_t = \sum_{i=1}^{n} w_t^i x_t^i \]
- Weighted sample covariance:
  \[ P_t = \sum_{i=1}^{n} w_t^i (x_t^i - \mu_t)(x_t^i - \mu_t)^T \]
Particle filter: relation to Bayes update rule

\[ Bel(x_t) = \eta p(z_t|x_t) \int p(x_t|x_{t-1}, u_{t-1}) Bel(x_{t-1}) dx_{t-1} \]

- Draw \( x^i_{t-1} \) from \( Bel(x_{t-1}) \) (resample)
- Draw \( x^i_t \) from \( p(x_t|x^i_{t-1}, u_{t-1}) \) (move and diffuse)

Importance factor/weight for \( x^i_t \) (measure):

\[
 w^i_t = \frac{\text{target distribution}}{\text{proposal distribution}} = \frac{\eta p(z_t|x_t) \int p(x_t|x_{t-1}, u_{t-1}) Bel(x_{t-1}) dx_{t-1}}{\int p(x_t|x_{t-1}, u_{t-1}) Bel(x_{t-1}) dx_{t-1}} \propto p(z_t|x_t)
\]
Importance sampling principle

Indicator function: is 1, if argument true

\[ E_f [I(x \in A)dx] = \int f(x)I(x \in A)dx = \int \frac{f(x)}{g(x)} g(x)I(x \in A)dx = E_g [w(x)I(x \in A)] \]

Compensate for the mismatch by weighting the samples with: \( w = \frac{f}{g} \)
Particle filters in action

- Example: head tracking (likelihood based on skin color)

Video of K. Smith
Particle filter: people tracking example

Uses an adaptive classifier based on Adaboost → past lecture
Particle filters in action

Michael Isard and Andrew Blake
Particle filters in action

- Tracking a ball

Particle filter recovers
1. multi-modal
2. random sampling
Summary: particle filter algorithm

- Particle filters are an implementation of recursive Bayesian filtering.
- They represent the posterior by a set of weighted samples.
- In the context of localization, the particles are propagated according to the motion model.
- They are then weighted according to the likelihood of the observations.
- In a resampling step, new particles are drawn with a probability proportional to their weight.
Summary: particle filters

- Represents arbitrary (multimodal) densities
- Handle nonlinear, non-Gaussian systems
- Concentrate particles on interesting regions (by resampling)
- Number of samples is important
  - Use as few as necessary (for efficiency)
  - But use enough to do a good job exploring the state space
- Complexity grows exponentially with dimensionality of the state space

Further readings: advanced particle filters are described, e.g. in Probabilistic Robotics by S. Thrun
Oral exams

▪ Exam period 13.09.2015 – 22.09.2015 (13:30 to 17:30)

▪ Please email to Leivy Kaul (kaul@dfki.uni-kl.de) to get an appointment

▪ [http://ags.cs.uni-kl.de/teaching/oral-examinations/](http://ags.cs.uni-kl.de/teaching/oral-examinations/)
Thank you!
EKF application example

- Set-up: Two fully calibrated cameras observe a flying ball
- Measurements: 2D ball positions in each image (feature level)
- Wanted: (predicted) 3D position of ball
EKF application example

**Given:**
- Set of measurements and ground truth positions (for validation)
- Gravity vector, $g$
- Intrinsic and extrinsic camera parameters
- Sample time, $\Delta t$

**Tuning parameters:**
- Process noise covariance
- Measurement noise covariance
- Initial estimate and covariance
Image formation

**Assuming perfect pinhole:**

\[ p_s = \Pi(K \cdot R(p + T)) \]

\[ \Pi \left( \begin{pmatrix} x & y & z \end{pmatrix}^T \right) := \begin{pmatrix} x/z & y/z \end{pmatrix}^T \]

In our example, we assume no rotation \((R = I)\)
State-space model

- **State:** \( x_t = (p_w, v_w)^T \) 3D position and velocity of ball in world frame

- **Dynamic model:**
  - Constant acceleration due to gravity, \( g \), with white noise acceleration, \( \varepsilon \)
  - **Linear Gaussian model**
    \[
    \begin{pmatrix}
    p_w \\
    v_w
    \end{pmatrix}_t = 
    \begin{pmatrix}
    I_3 & I_3 \Delta t \\
    0 & I_3
    \end{pmatrix} 
    \begin{pmatrix}
    p_w \\
    v_w
    \end{pmatrix}_{t-\Delta t} 
    + 
    \begin{pmatrix}
    I_3 \Delta t^2 / 2 \\
    I_3 \Delta t
    \end{pmatrix} g_w 
    + 
    \begin{pmatrix}
    I_3 \Delta t^2 / 2 \\
    I_3 \Delta t
    \end{pmatrix} \varepsilon_t 
    \]
    \[
    \text{cov}(B_t \varepsilon_t) = B_t R_t B_t^T
    \]

- **Measurement model:** 2D ball positions in images
  - **Nonlinear Gaussian model with additive noise**
    \[
    z = h(x) + \delta
    \]
  
  \[
  \Rightarrow \begin{pmatrix}
  p_{s1} \\
  p_{s2}
  \end{pmatrix} = 
  \begin{pmatrix}
  \prod(K(p_w + T_1)) \\
  \prod(K(p_w + T_2))
  \end{pmatrix} + \begin{pmatrix}
  \delta \\
  \delta
  \end{pmatrix}
  \]

EKF required + linearization of measurement model
EKF linearization (1\textsuperscript{st} order Taylor approximation)

- Motion model is already linear
- Measurement model needs to be linearized with respect to state, \( x \)
- Linearization point: predicted state, \( \bar{\mu} \)

\[ z = h(x) \quad h(x) \approx h(\bar{\mu}) + \frac{\partial h(\bar{\mu})}{\partial x} (x - \bar{\mu}) \]

Jacobian
Measurement Jacobian (4x6)

- Develop Jacobian for one camera, then duplicate:
  \[
  \text{pix} = \Pi(K(p_w + T)) = \Pi((x\ y\ z)^T) := \left(\frac{x}{z}\frac{y}{z}\right)^T
  \]

- Apply chain rule:
  \[
  \frac{\partial \Pi(g(x))}{\partial x} = \frac{\partial \Pi(u)}{\partial u} \cdot \frac{\partial g(x)}{\partial x}
  \]
Measurement Jacobian (4x6)

\[
\frac{\partial \Pi(g(x))}{\partial x} = \frac{\partial \Pi(u)}{\partial u} \cdot \frac{\partial g(x)}{\partial x}
\]

\[u = g(x) = K(p_w + T), \quad \text{pix} = \Pi(u) := \begin{pmatrix} u_x & u_y & u_z \end{pmatrix}^T\]

\[
H_t := \frac{\partial h}{\partial x} = \begin{pmatrix} J_\Pi \cdot K & 0_{2 \times 3} \\ J_\Pi \cdot K & 0_{2 \times 3} \end{pmatrix} J_\Pi
\]
Extended Kalman filter (adapted to example)

1. **Extended_Kalman_filter**($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):

2. **Prediction**:

3. \[ \bar{\mu}_t = A_t \mu_{t-1} + B_t (u_t + \varepsilon_t) \]

4. \[ \bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + B_t R_t B_t^T \]

5. **Correction**:

6. \[ K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1} \]

7. \[ \mu_t = \bar{\mu}_t + K_t (z_t - h(\bar{\mu}_t)) \]

8. \[ \Sigma_t = (I - K_t H_t) \bar{\Sigma}_t \]

9. **Return** $\mu_t, \Sigma_t$

- **Motion model is linear**
- **Nonlinear measurement model has additive noise**