



University of Trento

# Part IV: Motion Tracking

Nicola Conci  
[nicola.conci@unitn.it](mailto:nicola.conci@unitn.it)

# Introduction and motivations

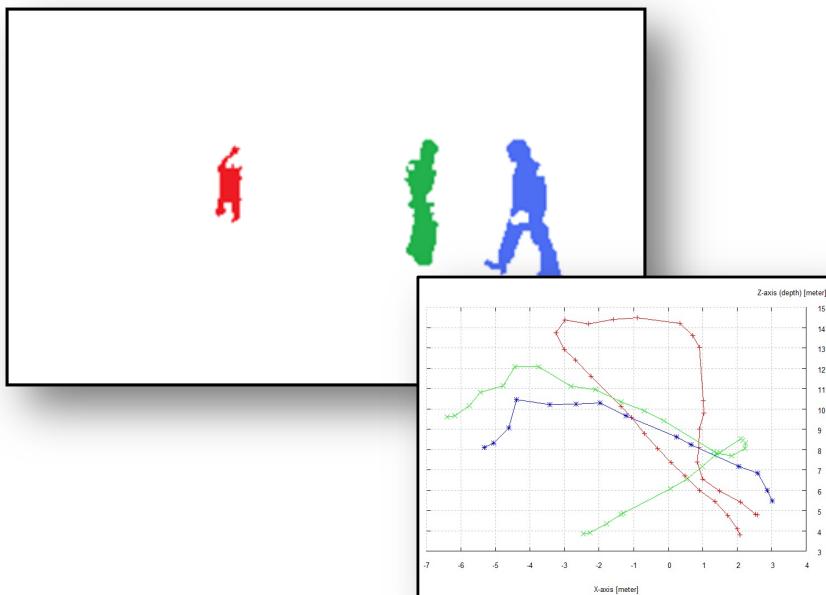


- Follow one or more moving objects for:
  - People monitoring
  - Traffic monitoring and analysis
  - Biological applications (cells tracking)
- High Level analysis for
  - Discovery of activities
  - Behavior understanding
  - Detection of threats
- More in general
  - Understand WHAT is moving in a scene
  - Understand HOW it moves/interacts with the environment

# Object Tracking



- On the basis of the applications requirements
  - Track the 2D coordinates (centroid)
  - Track in 3D (more cameras are required)
  - Determine the position of complex objects (human body articulations)
- Examples:



# Object Tracking: applications



- Monitoring and surveillance
  - Motion classification
  - Identification of anomalous/suspicious behaviors
  - Follow a trajectory
- Human Machine Interfaces
  - Interact with a device removing physical barrier (mouse, keyboard)
  - Natural language understanding
- Virtual Reality
  - Immersive presence
  - Animation of virtual characters
- Mining and retrieval
  - Browse databases containing specific motion patterns

# Benefits



- In HCI, control PC (or systems in general) → no need for additional tools
- In surveillance, Automated / Semi-automated systems → reduce the stress of human operators
- Virtual reality, computer animation → animate and drive the avatar
- But also
  - Training of athletes,
  - Gait disorders detection
  - Medical applications
  - ...

# 2D Tracking



- Motion in the image plane
- Sometimes it is enough
- Different approaches
  - **Region-based** → set of pixels that share similar features (color)
  - **Contour-based** → determine position and shape of an object over time. Useful to track deformable objects.
  - **Feature-based** → select meaningful points (contours, corners)
  - **Template-based** → use specific models (hands, faces, eyes)

# Tracking: Region-based



- Tracking regions with uniform appearance is a good method for real-time applications:
  - Fast (20-30 fps)
  - Good tradeoff quality/speed
- A region can be represented by the projection of an area with similar color on the image plane
- For example obtained from segmentation (e.g. after background suppression)

# Tracking: Region-based



- **Colors** of regions **must be different** in order to be distinguished
- It is **unstable in presence of variable illumination**
- If applied to situations with variable illumination, appropriate compensation techniques have to be applied, i.e.:
  - Use HS or HSV
  - Use a normalized RGB space
- Ok indoor, troubles outdoor

# Tracking: Region-based



- What do I want to track?
  - Any type of moving object
  - Skin vs non-skin (hand and face tracking)
  - Areas with certain colors
- How?
  - Color thresholding if uniform
  - Color histograms
- Problems
  - Color changes over time (illumination, posture)
  - Acquired models of objects need to be updated

# Tracking: Region-based



- A possible approach:
  - Divide the object to be tracked into regions
  - Each region is associated to a color vector (average for all pixels in the region) or histogram
  - Compute the color at each frame
  - If the ratio between the reference and actual values is close to 1, the match is good

# Tracking: Region-based



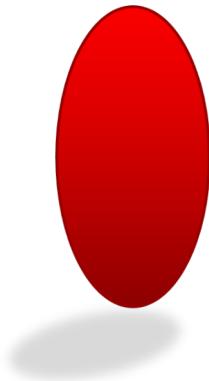
- How to use histograms
  - For each moving object compute the histogram
  - At each step evaluate the histogram of the tracked region  $O^t$  and compare it with the reference model  $O^r$  for each region  $i$
  - Similarity can be evaluated using:
    - Bin-by-bin comparison (intersection) 
$$\bigcap(O_i^t, O_i^r) = \sum_{n=1}^U \min\{O_{i,n}^r, O_{i,n}^t\}$$
    - SSD 
$$SSD(O_i^t, O_i^r) = \sum_{n=1}^U (O_{i,n}^r - O_{i,n}^t)^2$$
  - Bins should be neither too few, nor too many

See A. Bovik, Chapter 7

# Note: Shadows



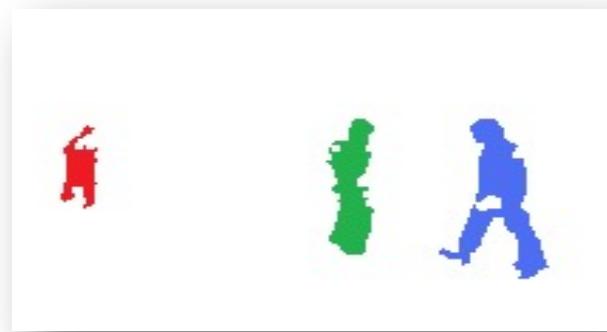
- Shadows are source of noise → false positives
- A shadow does not correspond to the motion of a real object
  - Variation of the luminance
  - Chrominance remains (ideally) unaltered
  - → for a proper tracking shadows should be removed before tracking using a suitable algorithm



# Blobs extraction



- Aggregation of a set of pixels that share common features
- An object can be made up by several blobs (head, torso, legs, ...)
- Features include also position
  - pixels with similar color but far (in x,y) from the object must be discarded
- Typical application in combination with background suppression

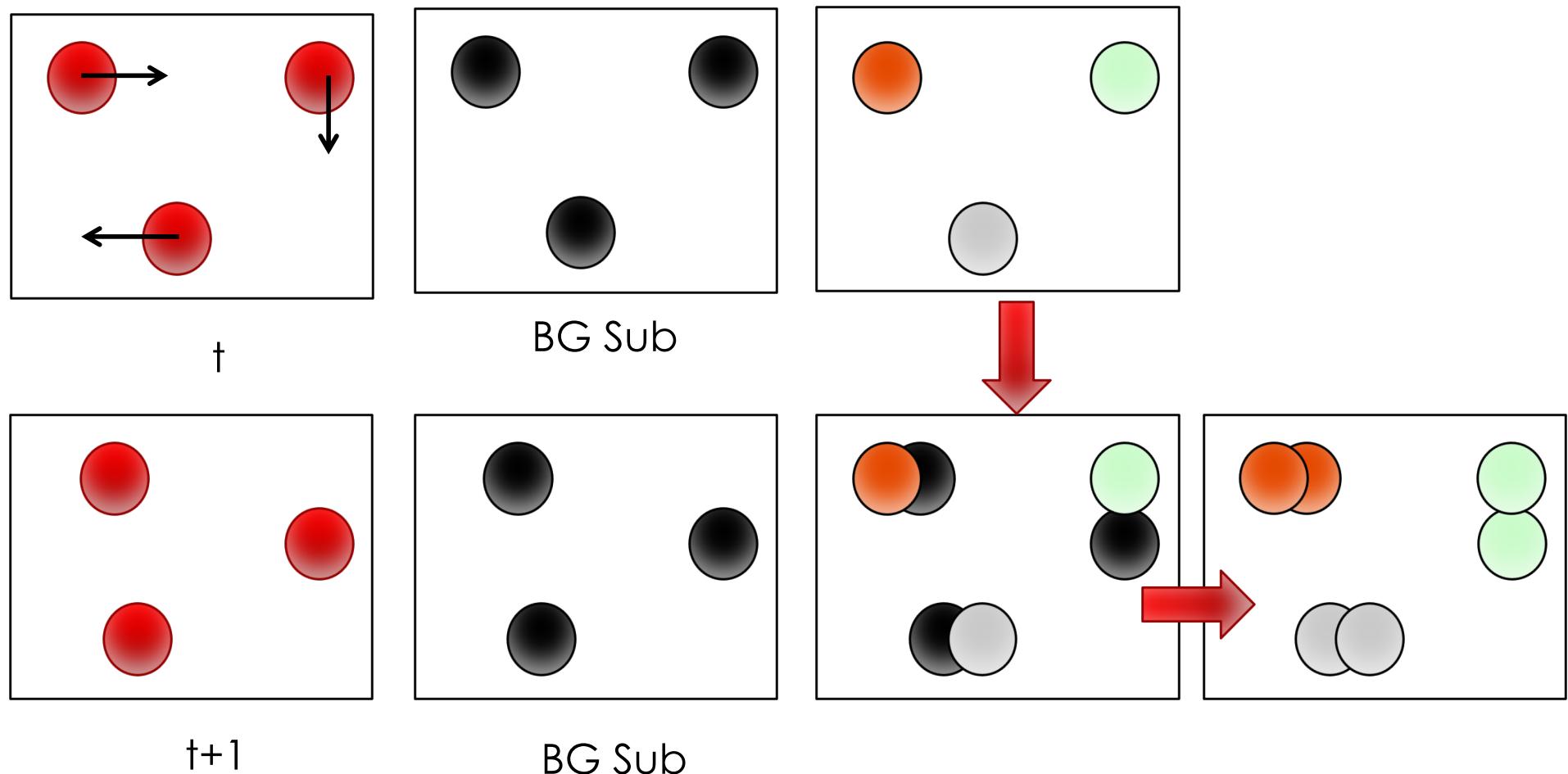


# Target association



- Procedure common to all trackers, not only region-based
- In general it is worth noting that detection is not carried out on a frame-basis
- This could be too demanding in terms of computational resources
- ONCE detected, targets are followed on a proximity basis
- Example:
  1. Background subtraction informs about the presence of motion
  2. Histograms characterize each moving objects
  3. Unless occlusions occur, the target in the next frame should be the closest blob

# Target association



# Target association

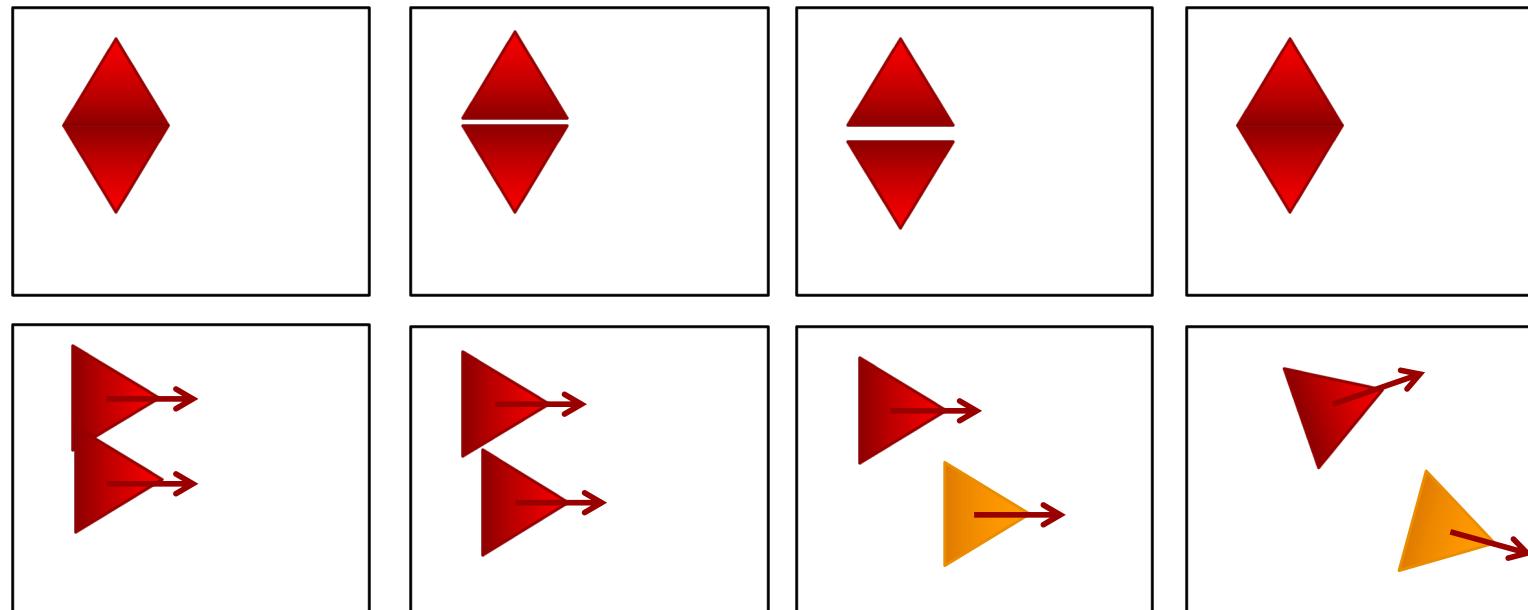


- Association can be performed as:
  - Overlapping blob (issues with scale → depends on objects size)
  - Centroid with minimum distance (as above)
  - Overlapping bounding boxes (may fail due to perspective)
  - Bounding box with minimum distance
  - Bounding box to centroid distance
- When association has been completed
  - For each object/blob update the appearance model to account for small variations
  - In presence of occlusions, the last saved model can be used to disambiguate

# Splitting



- If objects are identified as single blobs, no problem!
- However, background suppression may return ambiguous results
  1. Objects are split into several small blobs (not enough separation btw BG and FG)
  2. Two objects enter the scene together, then separate



# Splitting

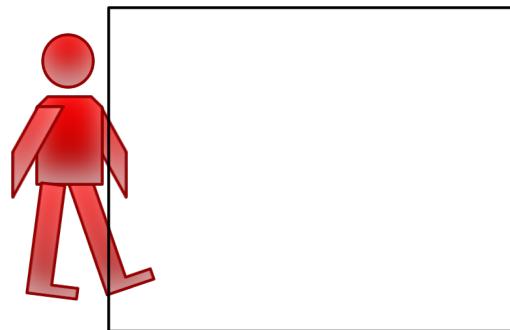


- Before saying it is case 1. or 2. evidence needs to be accumulated
  - Impossible to understand what's going on immediately
  - A temporal interval is needed to tell if:
    - It is necessary to keep the object merged even though it is fragmented
    - A new object has to be created

# Merging



- When two objects move together consistently, then perhaps they're the same object.
- Example:
  - A person's arm and foot enter the scene first and are detected as two well-separated FG blobs
  - Then also the rest of the body enters and a single blob is created
- Blobs are close and they might share similar properties



# Criteria for splitting and merging

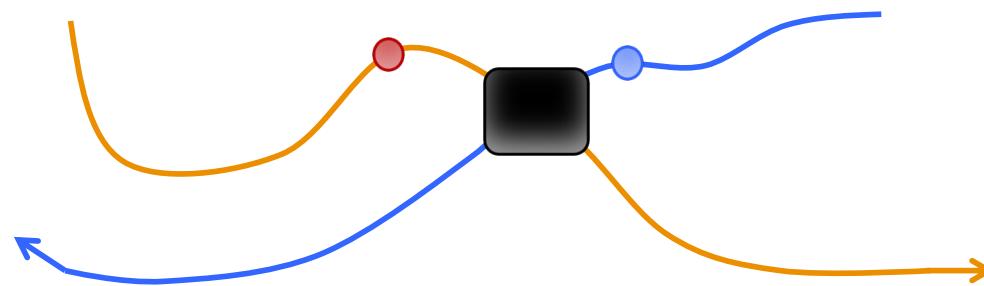


- Observation is the key
- Need to monitor the regions of interest and evaluate consistency in terms of:
  - Direction of motion
  - Distance between centroids/bounding boxes
  - Temporal range in which the phenomenon is observed
  - Velocity
  - Matching in features

# Occlusions



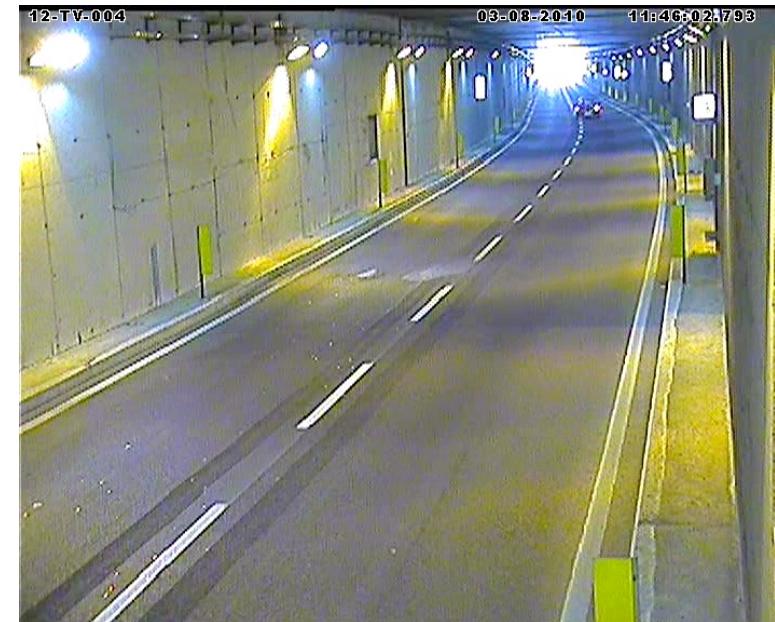
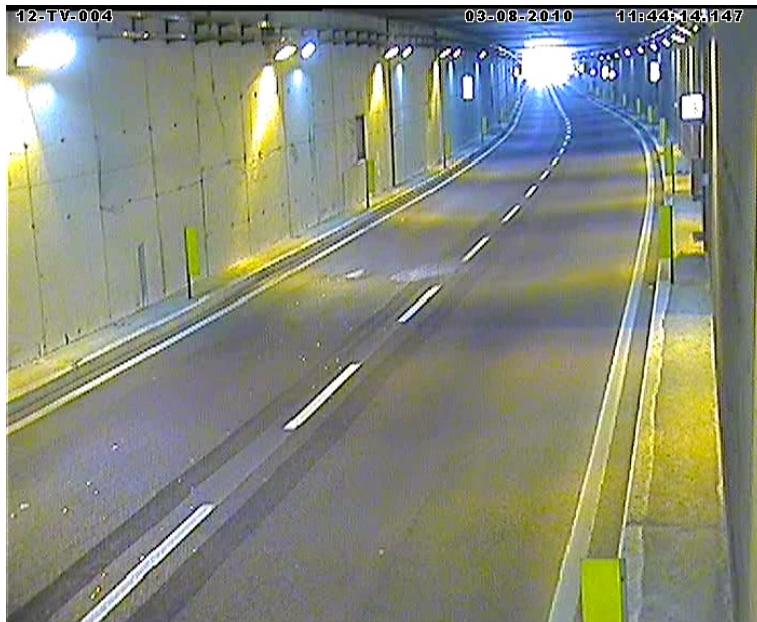
- Moving objects occlude each other while crossing
  - One or more objects disappear from the scene
  - Bigger blobs appear as a result of the occlusion, with properties that do not belong to any of the models acquired previously
- How to resolve the occlusion?
  - Need to re-associate “A to A” and “B to B”
  - Histograms are a good way out



# Occlusions



- It's an “anomalous” (though very frequent) situation
- Objects overlap and the acquired models are not reliable anymore
- Model update should be **avoided** during occlusions



# Tracking: Feature-based



- The objective is to retrieve the motion information of a set of features
- Considering:
  - $A=\{A(0), A(1), \dots, A(j-1)\}$  is a set of images
  - $m_i(x_i, y_i)$   $i=[0, j-1]$  the position of the feature in the image plane in each frame
- Objective:
  - determine the displacement vector  $d_i=(dx_i, dy_i)$  that best estimates the position of the feature in the next frame  $m_{i+1}(x_{i+1}, y_{i+1})=m_i+d_i$
- If needed, points can be grouped and objects can be represented using the bounding box or the convex hull



# What features?



- A good feature point has distinctive characteristics:
  - Brightness
  - Contrast
  - Texture
  - Edges
  - Corners
  - Points with high curvature

# Good features to track



- For each candidate point, compute:

$$Z = \begin{bmatrix} \sum_w J_x^2 & \sum_w J_x J_y \\ \sum_w J_y J_x & \sum_w J_y^2 \end{bmatrix}$$

- $J_x$  and  $J_y$  are the gradients evaluated on the point in  $x$  and  $y$  direction within  $W$  ( $n \times n$  window)
- A good feature point is where the smallest eigenvalue of  $Z$  is larger than a specified threshold
- In practice, it highlights corner points and textures

# Good features to track



- Eigenvalues should be above the image noise
- Small eigenvalues imply strong similarity within the window
- A large and a small eigenvalue means unidirectional patterns
- If both eigenvalues are large, the point is of high interest (salt&pepper texture, corners)

# How to track them?



- Must ensure that the same points are tracked throughout the video!
- Ideally we would expect that

$$A_i(Dm - d) = A_{i+1}(m)$$

- where
  - $A_i, A_{i+1}$  successive frames
  - $m$  is the 2D position of the feature point
  - $D$  is the deformation matrix (affine transformation model)
  - $d$  is the displacement vector (translational model)

# How to track them?



- However:
  - Due to noise the equality does usually not hold
  - Motion across successive frames is assumed to be small → a translational model is a good approximation
- Feature dissimilarity measure is used to quantify the change of appearance between the first and current frame

$$\varepsilon = \iint_W [A_i(m-d) - A_{i+1}(m)]^2 \omega(m) dm$$

- $\omega$  is a weighting function (e.g., Gaussian to emphasize the center of the window)
- When the feature dissimilarity grows too large, the feature point should be abandoned

# How to track them?



- To minimize the residual we differentiate w.r.t. unknowns ( $d$ )

$$e = 2 \iint_W [A_i(m) - A_{i+1}(m)] g(m) \omega(m) dm$$

- and

$$g(m) = \begin{bmatrix} \frac{\partial(A_i(m) - A_{i+1}(m))}{\partial x} \\ \frac{\partial(A_i(m) - A_{i+1}(m))}{\partial y} \end{bmatrix}$$

- In this case the solution for the displacement vector can be expressed by the  $2 \times 2$  linear system of equations (see paper for details):

$$Zd = e$$

# The Lucas-Kanade optical flow



- Two-frame differential method for optical flow estimation developed by Bruce D. Lucas and Takeo Kanade (1981)
- Consider  $u = [u_x, u_y]$  in frame I and  $v = [v_x, v_y]$  in frame J
- The goal is to find  $d$  that satisfies  $v = u + d$  such as I and J are similar (translational model)
- Because of the aperture problem, **similarity** must be defined in 2D
- $d$  is the vector that minimizes

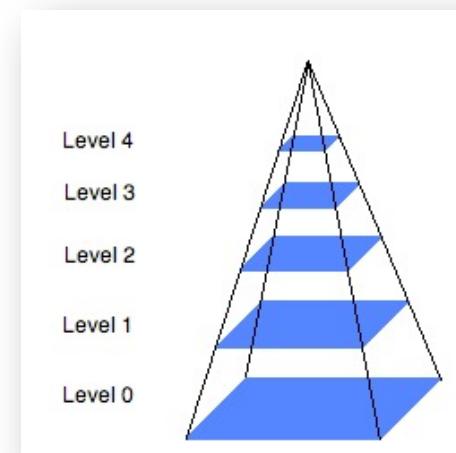
$$\epsilon(d) = \epsilon(d_x, d_y) = \sum_{x=u_x - \omega_x}^{u_x + \omega_x} \sum_{y=u_y - \omega_y}^{u_y + \omega_y} (I(x, y) - J(x + d_x, y + d_y))^2.$$

- $\omega$  is the integration window

# Pyramidal implementation



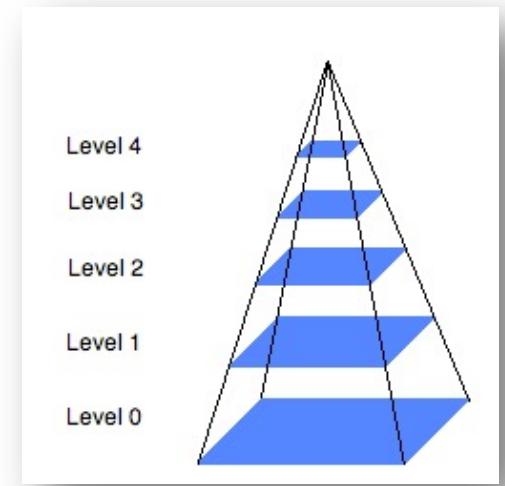
- The two key components to any feature tracker are **accuracy** and **robustness**
- **Accuracy** relates to the local sub-pixel accuracy attached to tracking → small integration window preferable to limit smoothness and preserve detail information (two image patches moving rapidly in different directions)
- **Robustness** relates to the sensitivity of tracking with respect to changes of light and big motions → a large window is preferable
- → Pyramidal implementation



# Pyramidal implementation



- Level 0 is the image at original resolution
- Level 4 (in this example) is the image at lowest resolution
- The L-th level is defined as a linear combination of the elements in the previous level



$$\begin{aligned} I^L(x, y) = & \frac{1}{4} I^{L-1}(2x, 2y) + \\ & \frac{1}{8} (I^{L-1}(2x - 1, 2y) + I^{L-1}(2x + 1, 2y) + I^{L-1}(2x, 2y - 1) + I^{L-1}(2x, 2y + 1)) + \\ & \frac{1}{16} (I^{L-1}(2x - 1, 2y - 1) + I^{L-1}(2x + 1, 2y + 1) + I^{L-1}(2x - 1, 2y + 1) + I^{L-1}(2x + 1, 2y - 1)). \end{aligned}$$

# Pyramidal implementation



- At each level of the pyramid an initial guess  $\mathbf{g}$  of the flow is computed at the lower level, which is then refined at the current level.

$$\epsilon^L(\mathbf{d}^L) = \epsilon^L(d_x^L, d_y^L) = \sum_{x=u_x^L - \omega_x}^{u_x^L + \omega_x} \sum_{y=u_y^L - \omega_y}^{u_y^L + \omega_y} (I^L(x, y) - J^L(x + g_x^L + d_x^L, y + g_y^L + d_y^L))^2$$

- $\mathbf{g}$  is used to pre-translate the image patch in the second image  $J$
- $\mathbf{d}$  should be small
- The information is then propagated at the upper level

$$\mathbf{g}^{L-1} = 2(\mathbf{g}^L + \mathbf{d}^L).$$

- Overall the displacement becomes  $\mathbf{d} = \sum_{L=0}^{L_m} 2^L \mathbf{d}^L.$

# Bayesian tracking



- Idea:
  - Estimate the state of a system over discrete time steps
  - At each step, noisy measurements
  - The state is represented by the x-y coordinates, velocity and acceleration along each dimension
  - 6D vector
  - Noise is typically smaller than the information about the state

# Bayesian tracking



- The state can be defined as

$$x_k = f_k(x_{k-1}, w_{k-1})$$

- Linking the measurement with the state vector:

$$z_k = h_k(x_k, v_k)$$

- $w_k$  is the process noise
- $v_k$  is the measurement noise
- $f_k$  and  $z_k$  are in general nonlinear functions defined in

$$f_k : \Re^{n_x} \times \Re^{n_w} \rightarrow \Re^{n_x} \text{ and } h_k : \Re^{n_x} \times \Re^{n_v} \rightarrow \Re^{n_z}$$

# Bayesian tracking



- System and measurement should be available in probabilistic form
- Every time the measurement is available, the estimation can be computed
- It is an online method
  - for every step  $k$  an estimate can be computed based on the previous observations  $z_k$  up to instant  $k$

# Bayesian tracking



- The initial pdf of the state vector is given,  $p(x_0|z_0)$
- $z_0$  contains no measurement
- Goal is to compute  $p(x_k|z_k)$  at time  $k$
- The process consists of two steps
  - Prediction
  - Update

# Bayesian tracking - Prediction

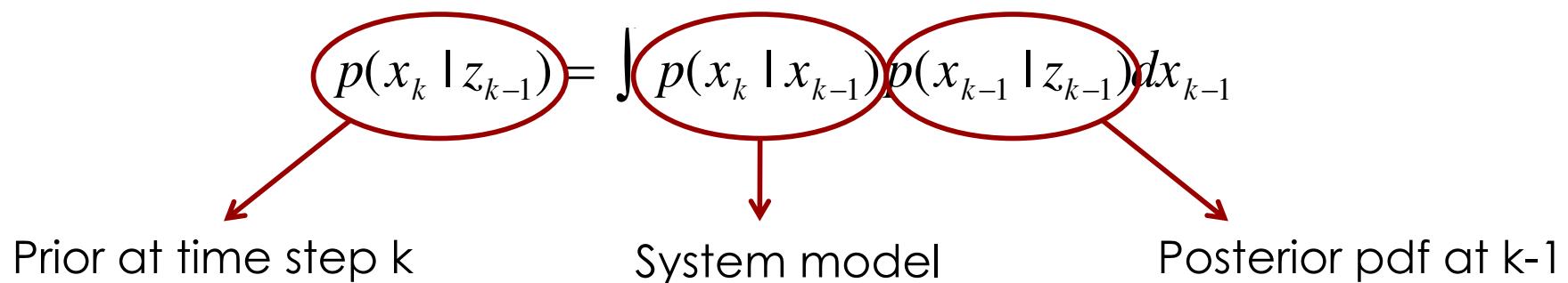


- From the previous representation:

$$x_k = f_k(x_{k-1}, w_{k-1}) \rightarrow p(x_k | x_{k-1})$$

$$z_k = h_k(x_k, v_k) \rightarrow p(z_k | x_k)$$

- The posterior pdf at  $k-1$  is propagated forward in time using the **system model**



# Bayesian tracking - Update



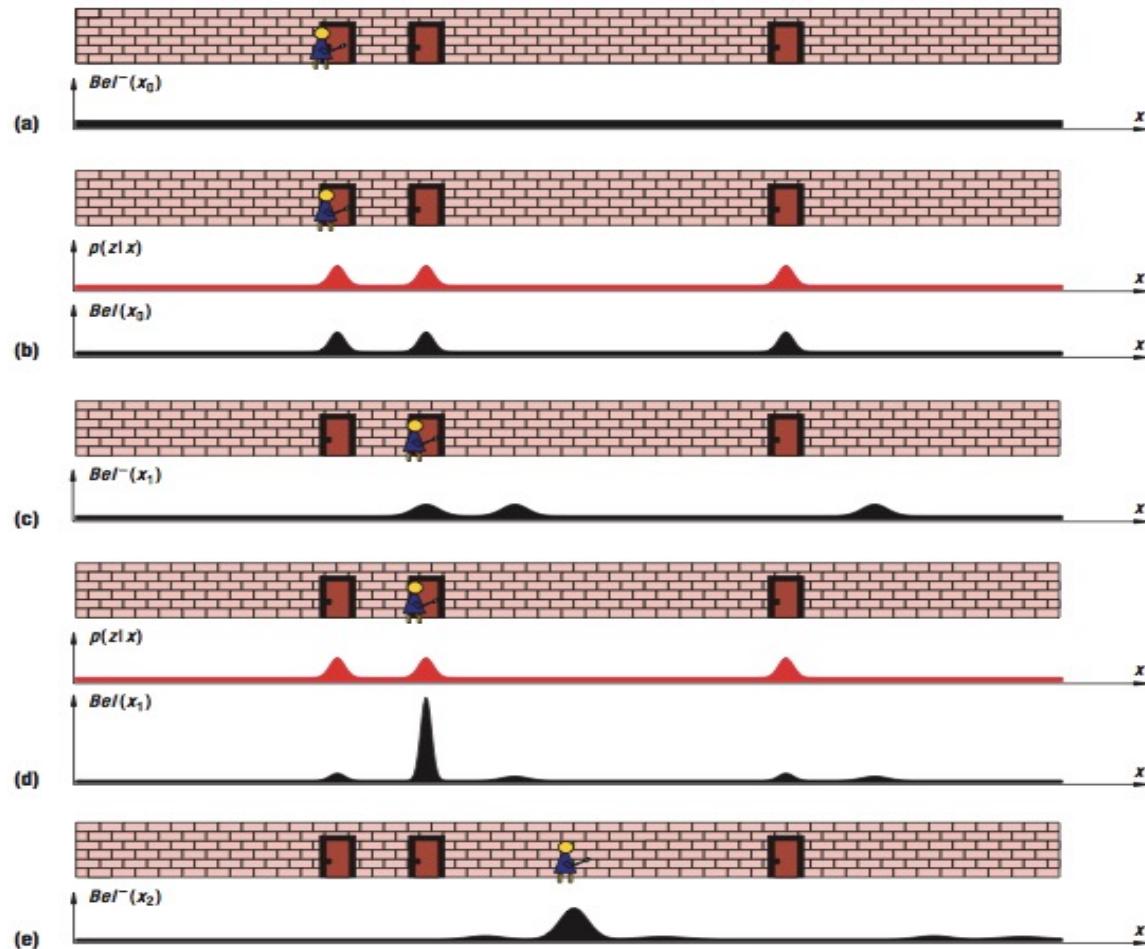
- Using the Bayes theorem it is possible to obtain the desired pdf  $p(x_k | z_k)$

$$p(x_k | z_k) = \frac{p(z_k | x_k)p(x_k | z_{k-1})}{p(z_k | z_{k-1})}$$

- Where  $p(z_k | z_{k-1})$  is used for normalization and computed as

$$p(z_k | z_{k-1}) = \int p(z_k | x_k)p(x_k | z_{k-1})dx_k$$

# Bayesian tracking – Toy Example



- User has a door “sensing device”
- User walks at typical walking speed
- (a) position is unknown
- (b) the sensor senses a door
- User can be in any of the three positions
- (c) User moves
- (d) another door is sensed
- (e) user moves again

D. Fox, J. Hightower, L. Liao, D. Schulz, G. Borriello, “Bayesian Filters for Location Estimation”, IEEE Pervasive Computing, 2003.



# The Kalman filter in a nutshell



- In line with Bayesian tracking:
  - Take a measurement
  - Measurement is subject to error
  - Derive the state of the system from the measurement
  
- We start from:

$$z_1, \sigma_{z1}^2$$

$$\hat{x}_1 = z_1$$

$$\sigma_1^2 = \sigma_{z1}^2$$

# The Kalman filter in a nutshell



- Add a second measurement
- Combine the two
- Iterate...

$$\begin{aligned} z_2, \sigma_{z2}^2 \\ \hat{x}_2 = ? \\ \sigma_2^2 = ? \end{aligned}$$

$$\hat{x}_2 = \hat{x}_1 + K_2(z_2 - \hat{x}_1)$$

$$K_2 = \frac{\sigma_{z1}^2}{\sigma_{z1}^2 + \sigma_{z2}^2}$$

$$\frac{1}{\sigma_2^2} = \frac{1}{\sigma_{z1}^2} + \frac{1}{\sigma_{z2}^2}$$

- The algorithm works online
- It's a weighted average

$$\sigma_2^2 = \frac{\sigma_{z1}^2 \sigma_{z2}^2}{\sigma_{z1}^2 + \sigma_{z2}^2}$$

# The Kalman filter in a nutshell



- Not all the difference between two measurements is noise
- Motion occurs
- Need to include a motion model, taking into account for example position and velocity
- The process:  
    Loop {
  1. Predict the new state and the uncertainty
  2. Correct using the new measurement  
    }

# KF in practice



- KF provides a computationally efficient solution to the least squares method
- Assuming that  $w_k$  and  $v_k$  are normal distributions (zero-mean,  $Q_k$  and  $R_k$  covariance), then
- The state and measurement model can be written as:

$$x_k = A_k x_{k-1} + B_k u_k + w_{k-1}$$

$$z_k = H_k x_k + v_k$$

# KF in practice



$$x_k = A_k x_{k-1} + B_k u_k + w_{k-1}$$

$$z_k = H_k x_k + v_k$$

- $x_k$  is the current state
- $x_{k-1}$  is the previous state
- $A_k$  is the state transition matrix
- $w_k$  is the process noise
- $z_k$  is the actual measurement
- $H_k$  is the measurement matrix
- $v_k$  is the measurement noise
- $B_k$  and  $u_k$  refer to additional and optional control input



# KF in practice

- We assume that the process noise and measurement noise are not changing over time, plus
  - Gaussian
  - Zero mean

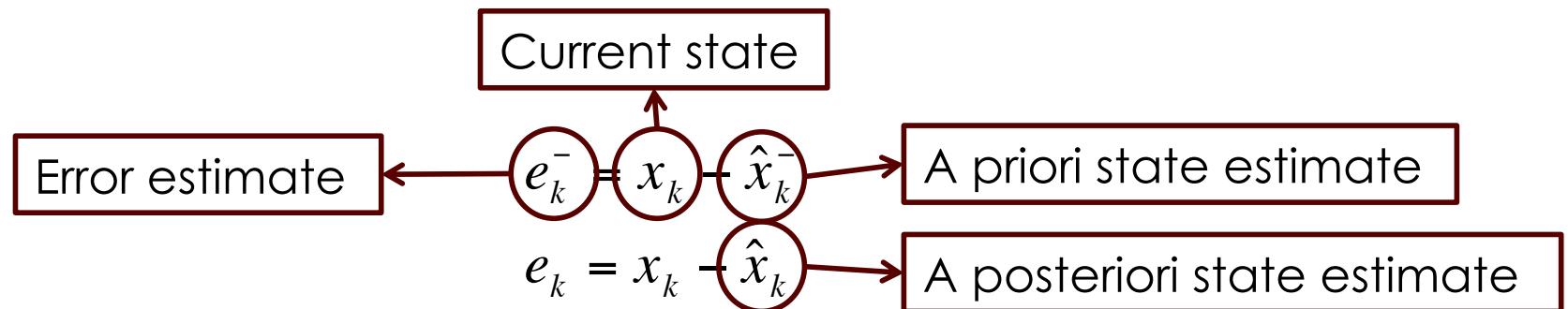
$p(w) = N(0, Q)$  Process noise covariance

$p(v) = N(0, R)$  Measurement noise covariance

# Predict-and-correct stages



- During the **first** phase the **current state estimate** together with the **error estimate** are propagated forward in time
- In the second stage a new measurement is taken to modify the two estimations
- Evaluation of
  - a priori estimate (based on the past measurements)
  - a posteriori estimate (as soon as measurement  $z_k$  is available)



# Predict-and-correct stages



- Once the error estimates have been computed, determine the error covariance by:

$$P_k^- = E[e_k^- e_k^{-T}]$$

$$P_k = E[e_k e_k^T]$$

- For the prediction stage we have then (discarding the optional control parameter):

$$\hat{x}_k^- = A_k \hat{x}_{k-1}$$

$$P_k^- = A_k P_{k-1} A_k^T + Q_{k-1}$$

# Predict-and-correct stages



- The update stage begins with the computation of the “gain” of the KF, which minimizes the a posteriori error covariance

$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + R_k)^{-1}$$

- The gain is used to modify the a priori estimate and to compute the a posteriori state estimate:

$$\hat{x}_k = \hat{x}_k^- + K_k (z_k - H_k \hat{x}_k^-)$$

- And to compute the a posteriori error covariance:

$$P_k = (I - K_k H_k) P_k^-$$

# How to:



- Predict the position of a point following a certain trajectory in (x,y) plane

$$x_k = A_k x_{k-1} + w_{k-1}$$

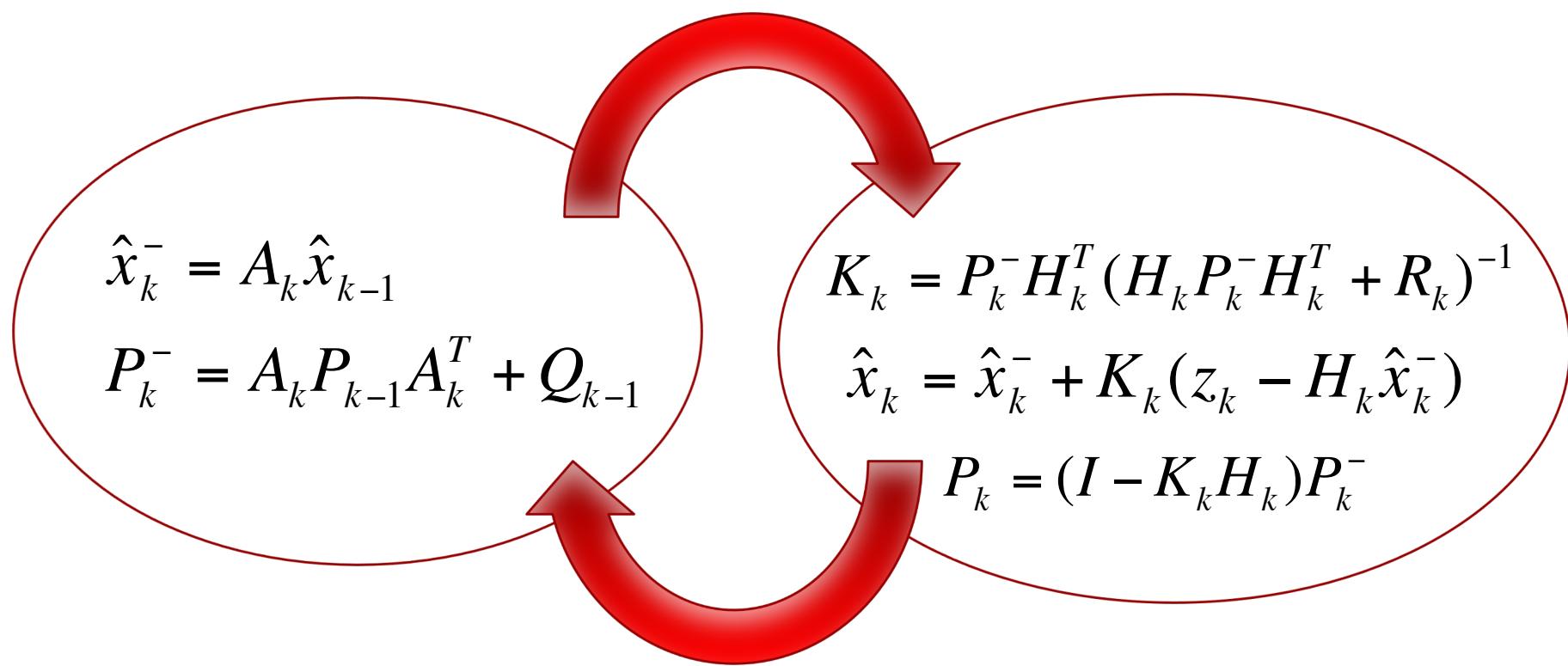
$$z_k = H_k x_k + v_k$$

- $x$ ,  $w$ ,  $z$ , and  $v$  are  $2 \times 1$  vectors
- $A$  and  $H$  are  $2 \times 2$
- Given  $w$  and  $v$  compute  $Q$  and  $R$  (in some cases determined empirically)
- During initialization
  - $x_0 = Hz_0$
  - $P_0$  diagonal matrix with reasonable values for covariance (maximum allowed displacement?)



# How to:

- Now predict ... and correct:



# Implementation of the KF



- Estimate a constant function  $y=c$  (voltage)



- Measurements we take are corrupted by white noise (quantization error due to conversion)
- Problem statement is easy, assuming A and H are both =1

$$x_k = x_{k-1} + w_k$$

$$z_k = x_k + v_k$$

Example taken from “An introduction to the Kalman Filter” by Greg Welch and Gary Bishop

# Filter equations



- Our a priori state estimate is given by the previous state

$$\hat{x}_k^- = \hat{x}_{k-1}$$

$$P_k^- = P_{k-1} + Q$$

- And the measurement update

$$K_k = P_k^- (P_k^- + R)^{-1}$$

$$\hat{x}_k = \hat{x}_k^- + K_k (z_k - \hat{x}_k^-)$$

$$P_k = (1 - K_k) P_k^-$$

- We assume the process noise is small
- To make the algorithm converge we have to set a “wrong” initial value
- $P \neq 0$  must be chosen as well

# Matlab code



```
% State transition matrix A and measurement matrix H are unitary  
% Assume the measurement error is normal (white) with sigma=0.1  
% Assume the process error variance is small 1e-5
```

```
x=1*ones(1,200); % the constant value to predict is 1
```

```
R=1; Q=1e-5;
```

```
N=size(x); % Let's take a number of measurements N
```

```
% Set priors
```

```
x_est = zeros(1,size(x,2));
```

```
P_est = zeros(1,size(x,2));
```

```
x_est(1) = 0.5;
```

```
P_est(1) = 0.5;
```

```
z = zeros(1,size(x,2)); % Set vector for measurements
```

```
z(1)=5;
```

```
for n=2:size(x,2)
```

```
    x_prior = x_est(n-1);
```

```
    P_prior = P_est(n-1) + randn*Q;
```

```
    z(n) = x(n) + randn*sqrt(R);
```

```
    K = P_prior / (P_prior + R);
```

```
    x_est(n) = x_prior + K*(z(n) - x_prior);
```

```
    P_est(n) = (1-K)*P_prior;
```

```
end
```

```
plot(1:size(x,2),z, 'r+');
```

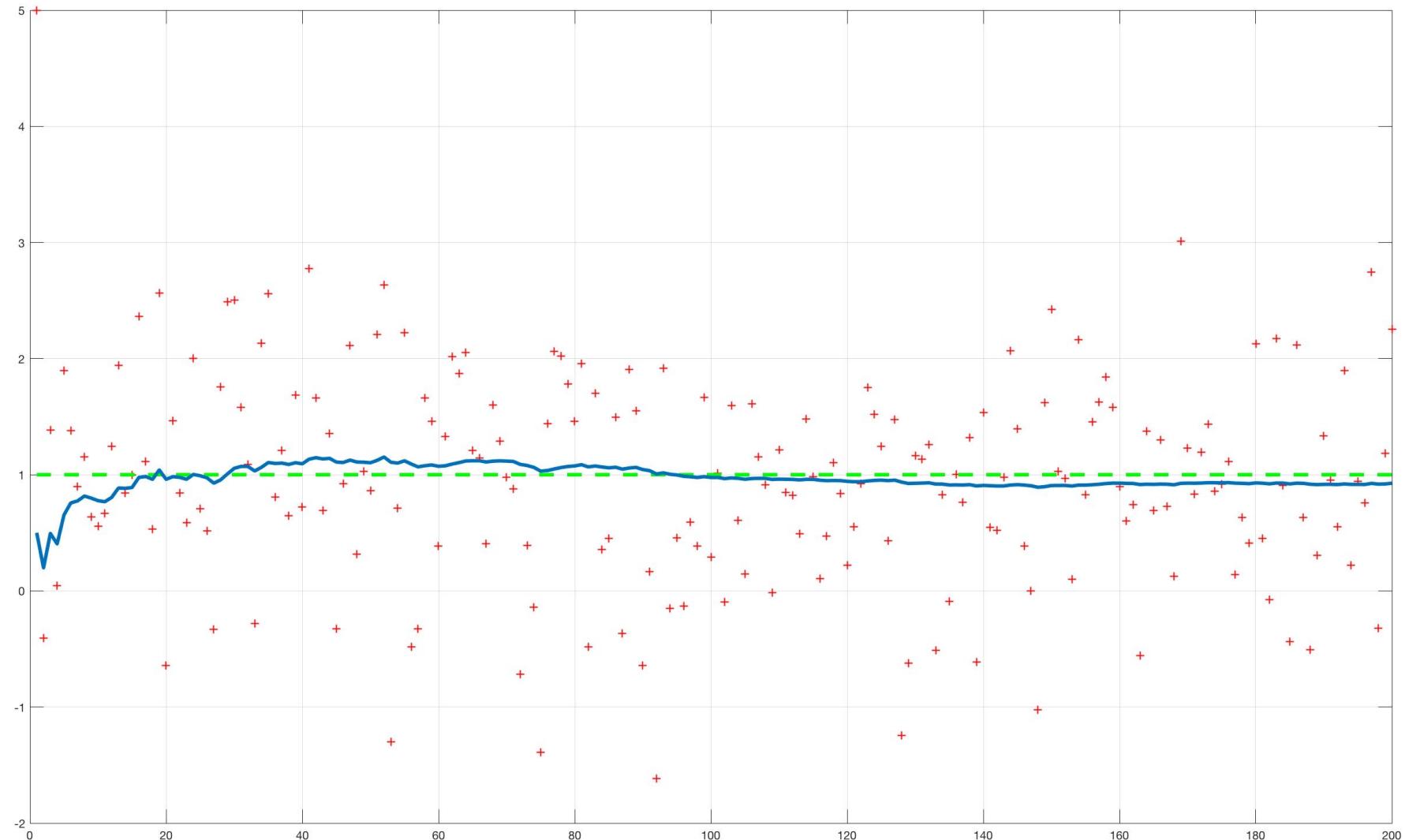
```
hold on;
```

```
grid on;
```

```
plot(1:size(x,2),[x], 'g--'); plot(1:size(x,2),x_est, '-');
```



# Matlab code



# The extended KF (EKF)



- Assumption in KF: state and measurement are linear
- In general wrong in object tracking
- “Adapt” the filter to linearize mean and covariance
- The process is still defined through  $f_k$  and  $h_k$ , which are in this case not linear

$$x_k = f_k(x_{k-1}, w_{k-1})$$

$$z_k = h_k(x_k, v_k)$$

# The EKF



- Use partial derivatives to linearize the estimation
- Partial derivatives of:
  - Process (A)
  - Measurements (H)
- Since we don't know the values of noise we can assume here for simplicity that state and measurement are

$$\tilde{x}_k = f(\hat{x}_{k-1}, 0)$$

$$\tilde{z}_k = h(\tilde{x}_k, 0)$$

# EKF prediction



- Using the same notation as for KF, the prediction stage is given by:

$$\tilde{x}_k = f(\hat{x}_{k-1}, 0)$$

$$P_k^- = A_{k-1} P_{k-1} A_{k-1}^T + W_{k-1} Q_{k-1} W_{k-1}^T$$

- $A_k$  is the Jacobian matrix of partial derivatives of  $f$  with respect to  $x_k$
- $W_k$  is the Jacobian matrix of partial derivatives of  $f$  with respect to  $w_k$
- $Q_k$  is the process noise covariance matrix

# EKF update



- The update step becomes:

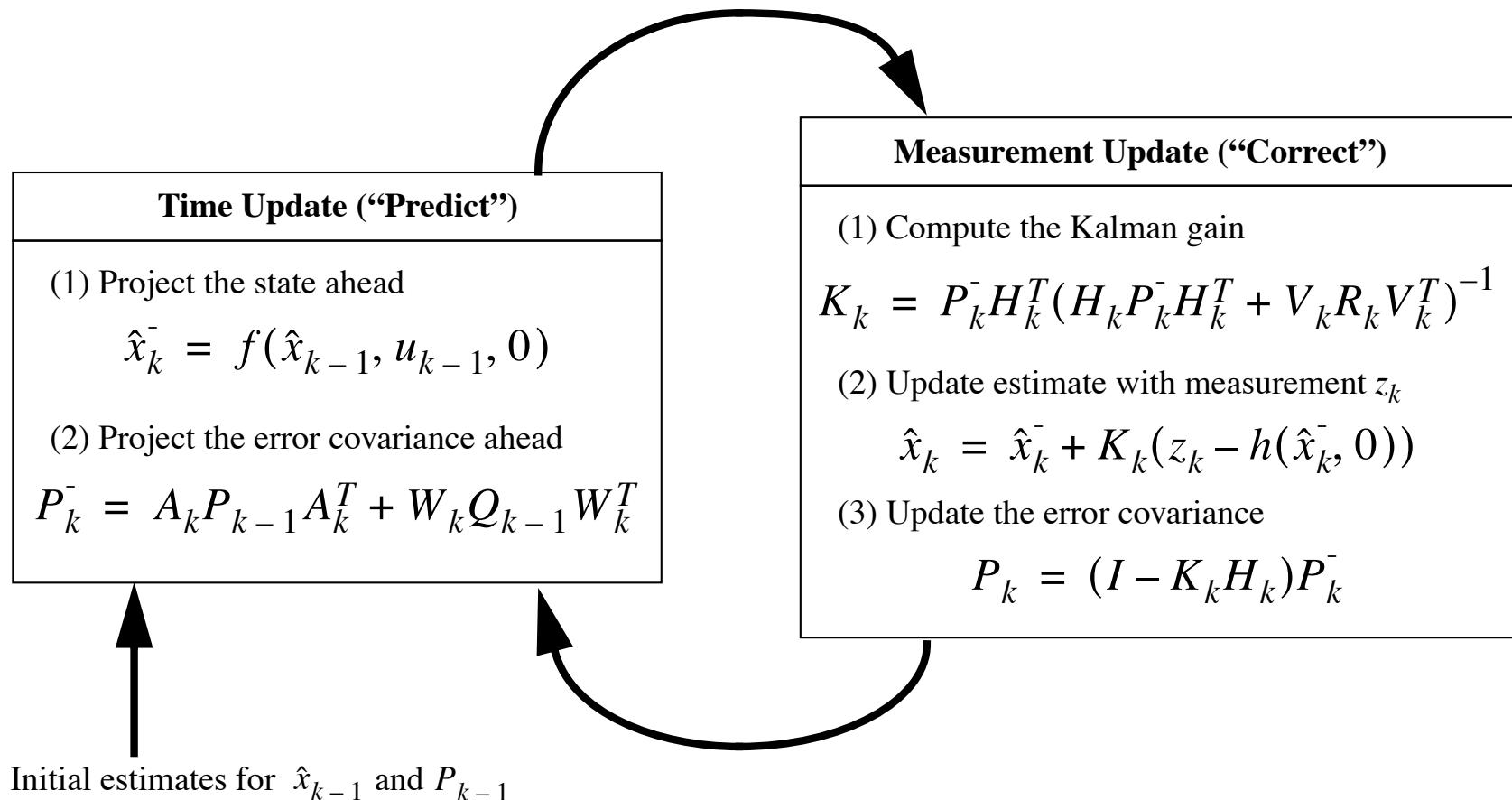
$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + V_k R_k V_k^T)^{-1}$$

$$\hat{x}_k = \tilde{x}_k + K_k (z_k - h_k(\tilde{x}_k, 0))$$

$$P_k = (I - K_k H_k) P_k^-$$

- $H_k$  is the Jacobian matrix of partial derivatives of  $h$  with respect to  $x_k$
- $V_k$  is the Jacobian matrix of partial derivatives of  $h$  with respect to  $v_k$
- $R_k$  is the measurement noise covariance matrix

# EKF predict-update



# Particle filters



- KF and EKF assume the posterior probability to be Gaussian
- In case it is not Gaussian performances are reduced
- PFs can do the job
- Main concept:
  - Represent alternative solutions as a set of samples
  - Each sample with a weight
  - The more the samples, the closer the optimal Bayesian estimate

# Properties



- While KF is an optimal solution to the estimation problem, PFs provide an approximate solution
- Can be applied to either linear and non-linear problems
- Can handle multimodal distributions
- Multiple hypothesis on the process state

# How PFs work



- Take a set of points associated with the corresponding weights:

$$\{x_k^i\}_{i=1}^N, \{w_k^i\}_{i=1}^N$$

- The a posteriori pdf is represented by:

$$p(x_k | z_k) \approx \sum_{i=1}^N w_{k-1}^i \delta(x_k - x_{k-1}^i)$$

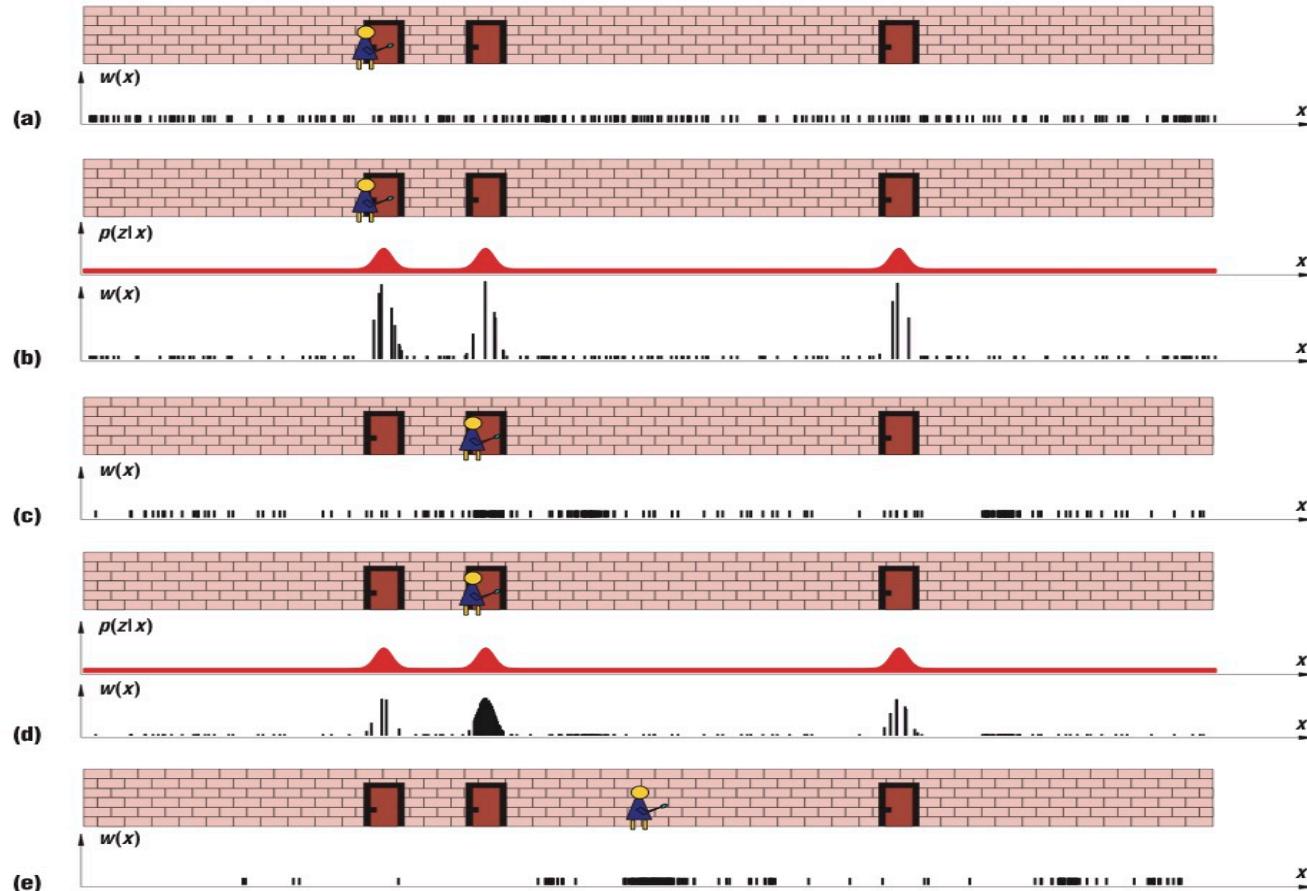
# PFs – SIR algorithm



- How do we select  $\{x_k^i\}_{i=1}^N, \{w_k^i\}_{i=1}^N$  for the pdf approximation?
- Select **PROPOSAL DISTRIBUTION**  $x_k^i \approx \pi(x_k | x_{k-1}^i)$  with  $i = 1 \dots N_s$
- Draw  $\pi(x_k | x_{k-1})$  (*Usually*  $= p(x_k | x_{k-1})$ )
- Update the weights  $\hat{w}_k^i = w_{k-1}^i p(z_k | x_k^i)$
- Normalize  $w_k^i = \hat{w}_k^i / \sum_{i=1}^{Ns} (\hat{w}_k^i)^2$
- Resample:
  - (i) draw  $N_s$  particles with probability proportional to their weight;
  - (ii) set uniform weight for the new set

$$w_k^i = 1 / N_s$$

# PFs – Toy Example



D. Fox, J. Hightower, L. Liao, D. Schulz, G. Borriello, “Bayesian Filters for Location Estimation”, IEEE Pervasive Computing, 2003.