AARHUS
UNIVERSITET

Computer Vision & Machine Learning

Alexandros Iosifidis
@
Department of Electrical and Computer Engineering
Aarhus University

Visual Tracking

A definition:

- Given a “target” in an image of an image sequence (or frame in a video), visual tracking corresponds a process defining the location (and size) of that target (object) in all remaining images (frames)

What is a target?

Visual Tracking

A definition:

- Given a “target” in an image of an image sequence (or frame in a video), visual tracking corresponds a process defining the location (and size) of that target (object) in all remaining images (frames)

A target can be:

- “Blips” in RADAR (radio detection and ranging) signals
- Objects defined by their bounding boxes
- Objects defined by their contours

The term object is used to define any type of targets, e.g. objects, humans, animals, etc.

Visual Tracking

A definition:
- Given a “target” in an image of an image sequence (or frame in a video), visual tracking corresponds a process defining the location (and size) of that target (object) in all remaining images (frames)

A target can be:
- “Blips” in RADAR (radio detection and ranging) signals
- Objects defined by their bounding boxes
- Objects defined by their contours

The term object is used to define any type of targets, e.g. objects, humans, animals, etc.

We will focus on object tracking in image sequences, when these are defined by their bounding box.

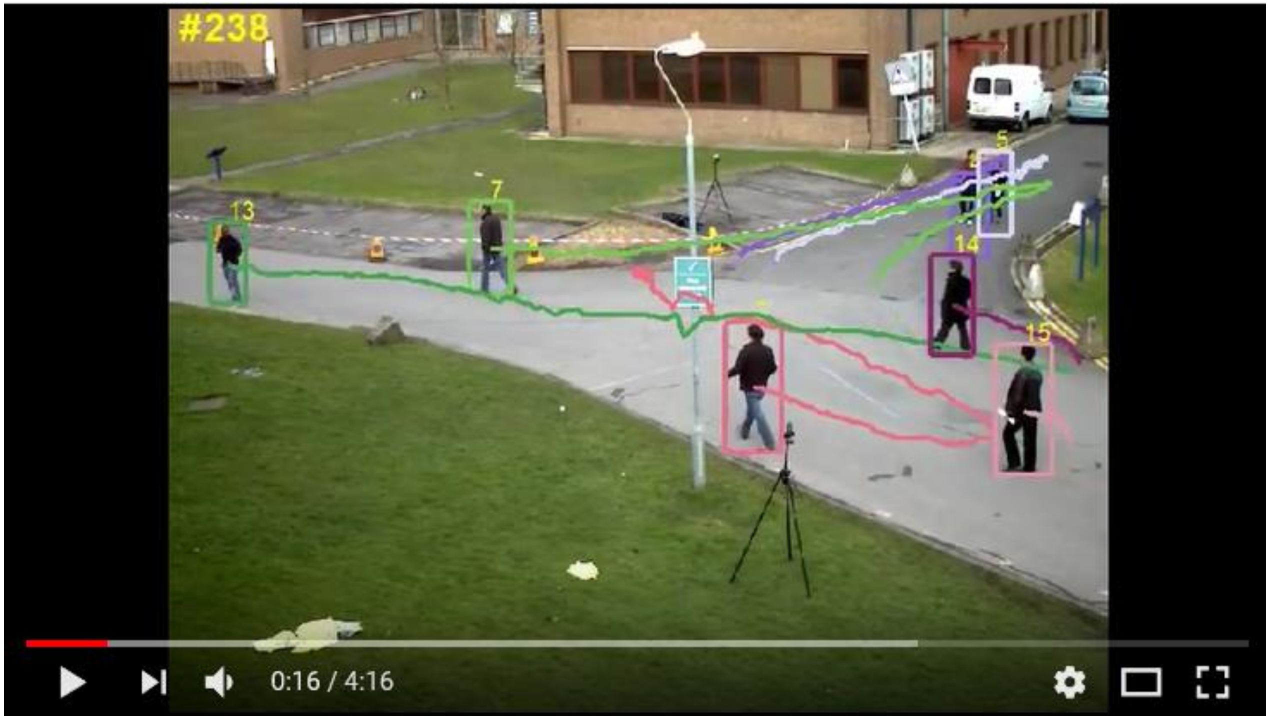
Some examples

https://www.youtube.com/watch?v=CigGvt3DXIw

Some examples

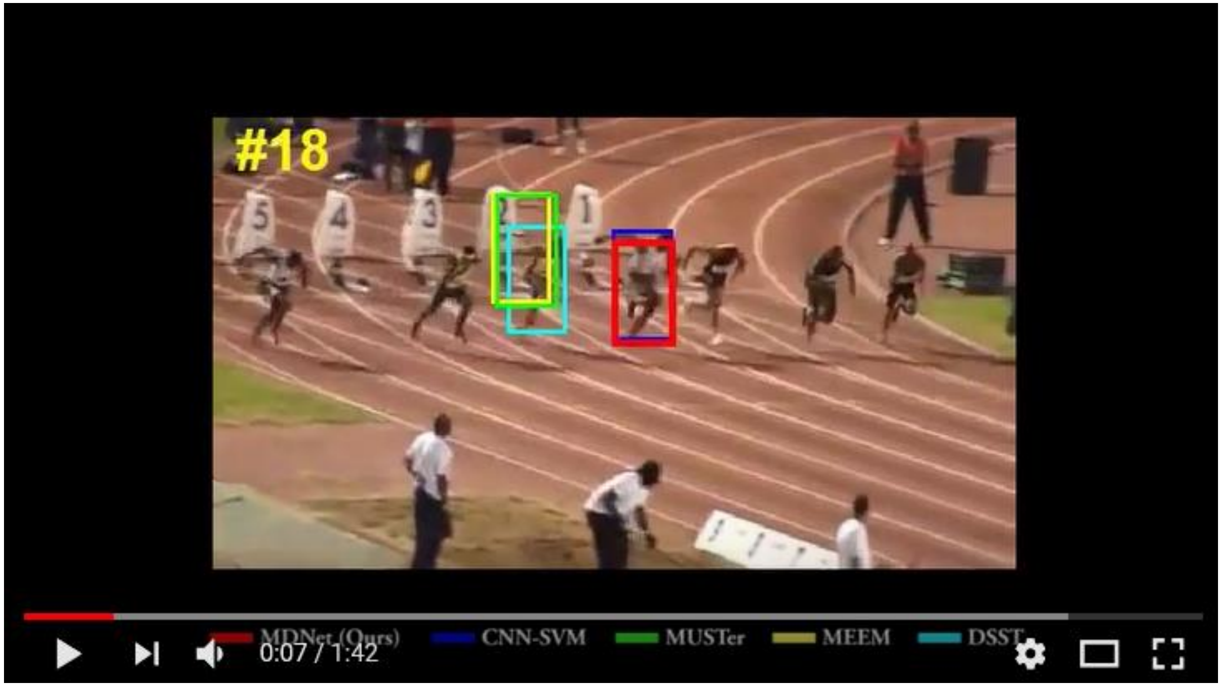
https://www.youtube.com/watch?v=xOGQ1IP7pt8

Some examples



https://www.youtube.com/watch?v=InAUnU596UE

Some examples



https://www.youtube.com/watch?v=InAUnU596UE

Visual Tracking

Initialization:

- By the user (draw a bounding box in the first image of the image sequence)
- Color-based initialization
- Object recognition based initialization
- Foreground detection-based initialization

Foreground detection

Foreground detection-based initialization

- Motion/Change detection

$$D_b(x, y, t) = (I(x, y, t) - I(x, y, t-1))^2$$

- Background subtraction

1. Creation of a ‘background image’ by accumulating information for a short period (10-15 secs) $BCK(x,y)_C^t = \alpha_C * BCK(x,y)_C^{t-1} + (1-\alpha_C) * I(x,y)^t$
2. Model each pixel of the BCK image using three Gaussians (R,G,B colors) with mean color values and covariances the values used to calculate BCK
3. If the color of a pixel in a new frame is not inside a range of distances from the mean vector of the corresponding pixel in BCK → foreground
4. The mean color value and the covariance of the BCK image pixel Gaussians are updated at every frame using the background pixels in I(x,y)

Foreground detection

Classification of a pixel as foreground:

$$\delta Br(x,y) = \|BCK(x,y)\| - I'(x,y) = \|BCK(x,y)\| - \frac{I(x,y) * BCK(x,y)}{\|BCK(x,y)\|}$$

$$\delta Cr(x,y) = \cos^{-1}\left(\frac{I(x,y) * BCK(x,y)}{\|I(x,y)\|\|BCK(x,y)\|}\right)$$

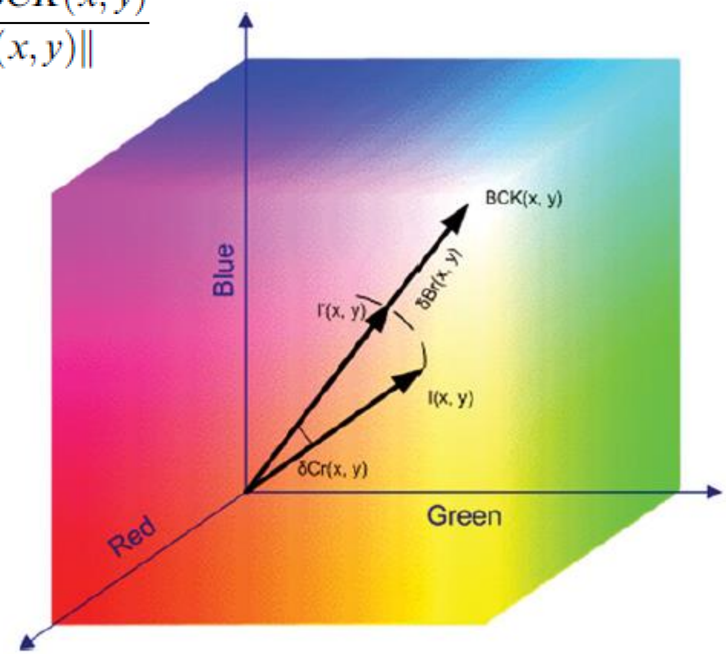
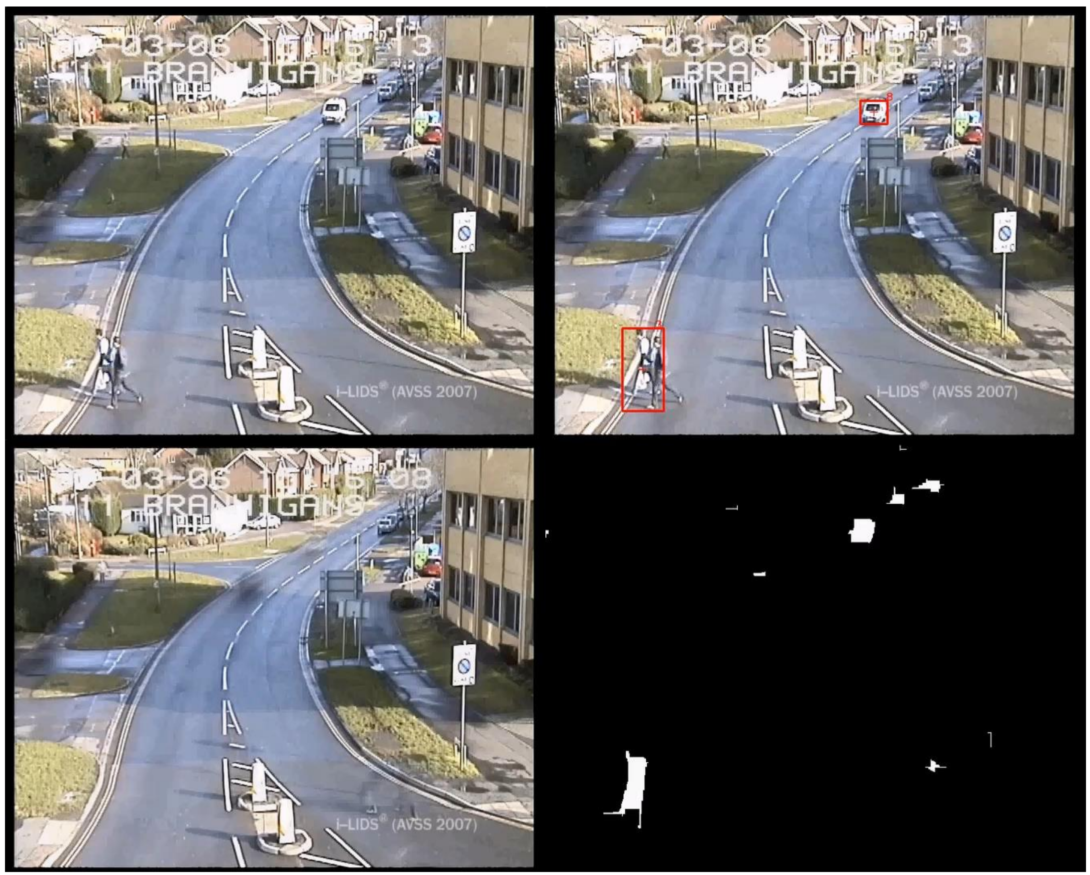


Figure 3. The physical meaning of Chromatic and Brightness distortion.

Foreground detection

Input video

Output video



BCK

D_b

Visual Tracking

In the following we will assume that the initialization has already been done

This approach is followed in order to focus on the properties of the tracking methods

We will describe:

- Kalman Filters
- Particle Filters
- Mean Shift
- Kanade, Lucas, Tomasi (KLT) tracker
- Detect, Predict, Track

Visual Tracking

In the following we will assume that the initialization has already been done

This approach is followed in order to focus on the properties of the tracking methods

We will describe:

- Kalman Filters
- Particle Filters
- Mean Shift
- Kanade, Lucas, Tomasi (KLT) tracker
- Detect, Predict, Track

Point-wise predictions

Template predictions

Optical flow based predictions

State of the Art object tracker

Kalman filters

Used for:

- The prediction of the ‘state’ of a point/vector

The point/vector can represent:

- The location ($\mathbf{x}$ = [p_x,p_y] coordinates) of a point (e.g. in RADAR signals)
- The location and speed ($\mathbf{x}$ = [p_x,p_y,u_x,u_y] vector) of a point
- The location, speed and size ($\mathbf{x}$ = [p_x,p_y,u_x,u_y,w,h] of an object centered at pixel [p_x,p_y] and having size $w \cdot h$ pixels

Kalman filters

At each time instance t, the Kalman filter uses the transition model

$$\mathbf{x}_t = \mathbf{A}\mathbf{x}_{t-1} + \mathbf{B}\mathbf{u}_{t-1} + \mathbf{q}_{t-1}$$

q is the noise associated with the state transition process
A is a matrix relating the state at time t-1 to the state at time t
u is a control input (that can incorporate external knowledge to the prediction
B is a matrix relating the control input at time t-1 with the state at time t

All variables are considered to be sampled from a Gaussian distribution.
Hence, the distributions can be characterized by a mean and covariance.

Kalman filters

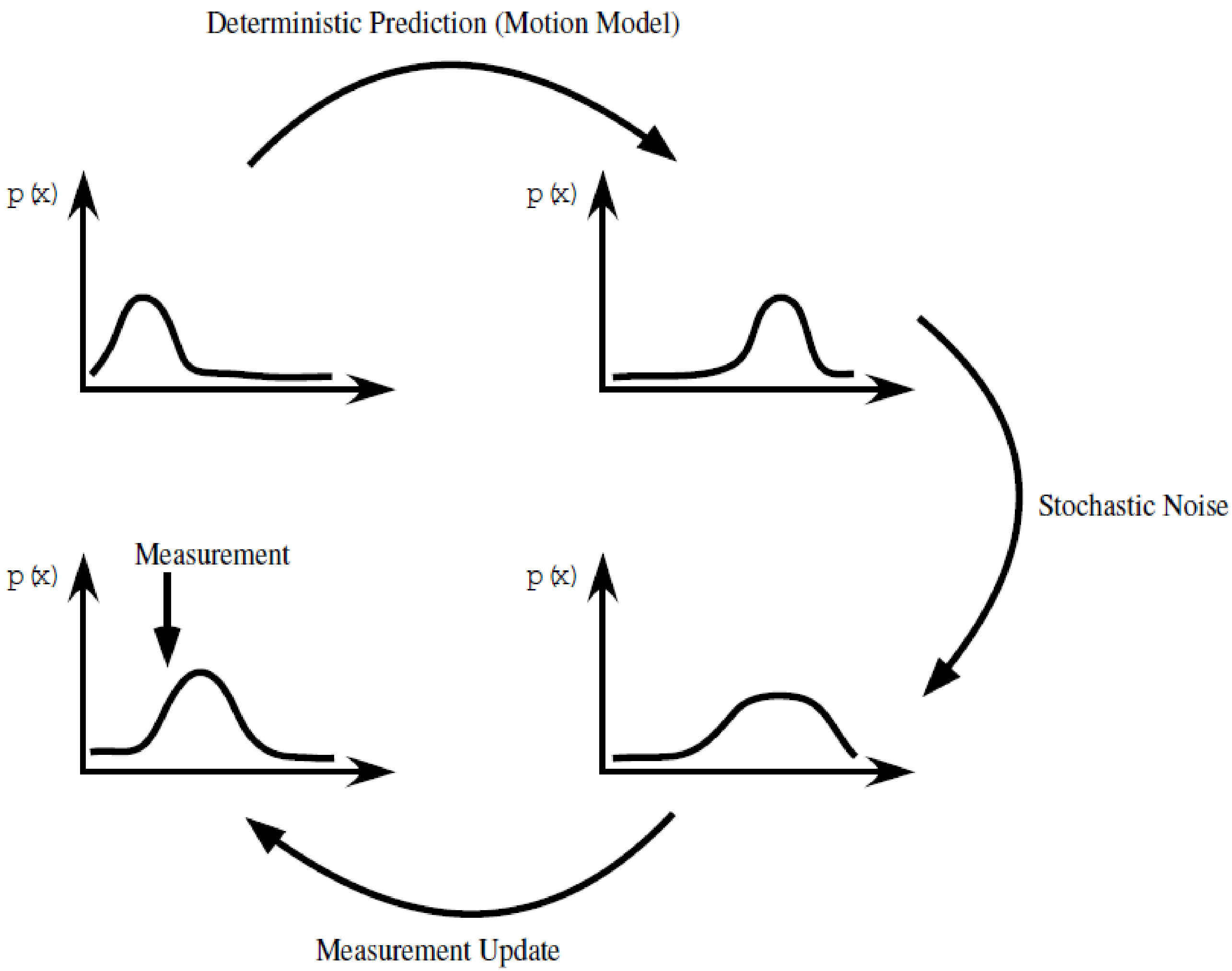


Figure 2.5: The main steps performed when Kalman filtering.

Kalman filters

At the time instance t:

- The state (a priori) estimate/prediction is given by

$$\hat{\mathbf{x}}_t^- = \mathbf{A}\hat{\mathbf{x}}_{t-1} + \mathbf{B}\hat{\mathbf{u}}_{t-1}$$

A: a matrix relating the state at time t-1 to the state at time t
u: a control input (that can incorporate external knowledge to the prediction)
B: a matrix relating the control input at time t-1 with the state at time t

The control input **u** and the corresponding system/matrix **B** is not necessary to be used (if no external information Is available)

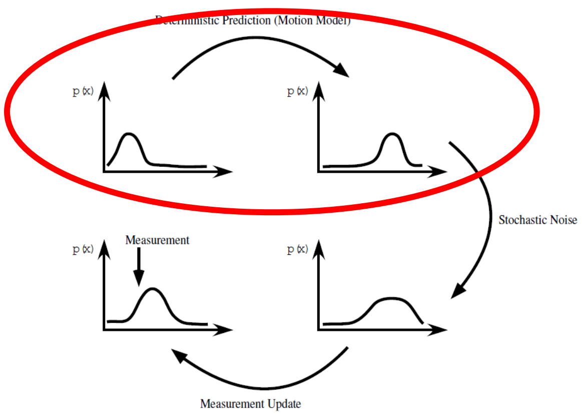


Figure 2.5: The main steps performed when Kalman filtering.

Kalman filters

At the time instance t:

- The state (a priori) estimate/prediction is given by

$$\hat{\mathbf{x}}_t^- = \mathbf{A}\hat{\mathbf{x}}_{t-1} + \mathbf{B}\hat{\mathbf{u}}_{t-1}$$

A: a matrix relating the state at time t-1 to the state at time t

u: a control input (that can incorporate external knowledge to the prediction)

B: a matrix relating the control input at time t-1 with the state at time t

For a state **x** = [p_x,p_y,u_x,u_y] , the matrix **A** can have the form

$$\mathbf{A} = \begin{array}{c} \\ \\ \end{array}\begin{array}{cc} \begin{array}{cccc} p_x & p_y & u_x & u_y \end{array} & \\ \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} & \begin{array}{c} p_x \\ p_y \\ u_x \\ u_y \end{array} \end{array}$$

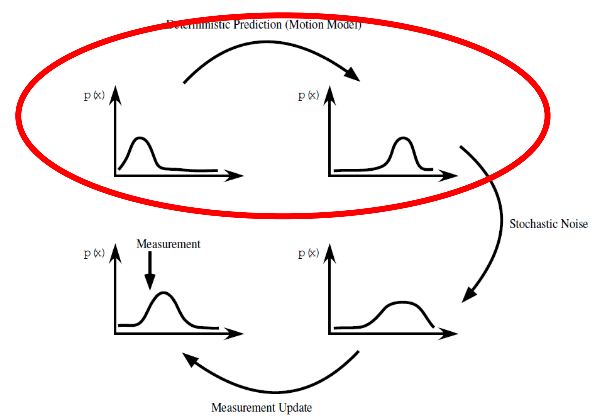


Figure 2.5: The main steps performed when Kalman filtering.

Kalman filters

At the time instance t:

- The covariance (a priori) estimate/prediction is given by

$$\hat{\mathbf{P}}_t^- = \mathbf{A}\mathbf{P}_{t-1}\mathbf{A}^T + \mathbf{Q}$$

Q: the covariance of the noise associated with the state prediction process

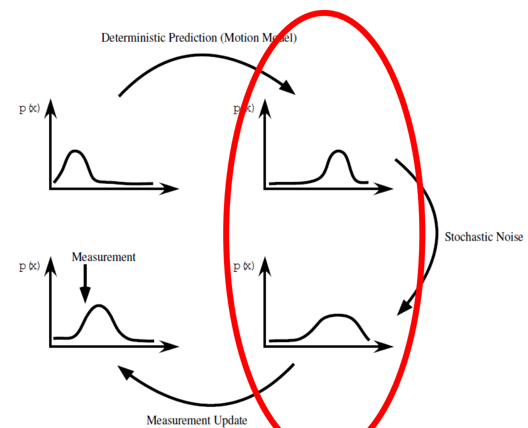


Figure 2.5: The main steps performed when Kalman filtering.

Kalman filters

At the time instance t:

- The correction stage involves the measurement of the state at time t. The measurement is given by

$$\mathbf{z}_t = \mathbf{H}\mathbf{x}_t + \mathbf{r}_t$$

r: the noise of the measurement

H: a matrix that can map the state at time t to the measurement at time t

In the simplest case (e.g. when the measurement involves all elements in the state, **H** is the identity matrix

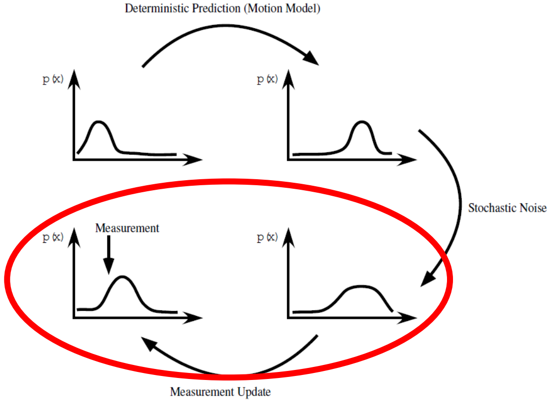


Figure 2.5: The main steps performed when Kalman filtering.

Kalman filters

At the time instance t:

- The correction stage involves the measurement of the state at time t. The measurement is given by

$$\mathbf{z}_t = \mathbf{H}\mathbf{x}_t + \mathbf{r}_t$$

- The (a posteriori) estimate is given by

$$\hat{\mathbf{x}}_t = \hat{\mathbf{x}}_t^- + \mathbf{K}_t\left(\mathbf{z}_t - \mathbf{H}\hat{\mathbf{x}}_t^-\right)$$

where $\mathbf{K}_t = \mathbf{P}_t^-\mathbf{H}^T\left(\mathbf{H}\mathbf{P}_t^-\mathbf{H}^T + \mathbf{R}\right)^{-1}$

R: the noise of the measurement process

$\left(\mathbf{z}_t - \mathbf{H}\hat{\mathbf{x}}_t^-\right)$: the 'innovation' term expressing the difference between the predicted position and the measurement

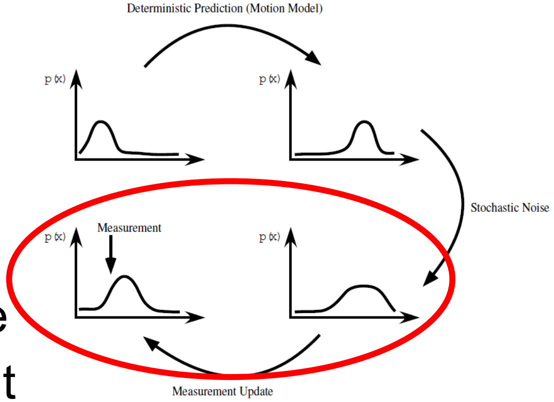


Figure 2.5: The main steps performed when Kalman filtering.

Kalman filters

At the time instance t:

- The correction stage involves the measurement of the state at time t. The measurement is given by

$$\mathbf{z}_t = \mathbf{H}\mathbf{x}_t + \mathbf{r}_t$$

- The (a posteriori) estimate of the covariance is given by

$$\mathbf{P}_t = \left(\mathbf{I} - \mathbf{K}_t\mathbf{H}\right)\mathbf{P}_t^{-}$$

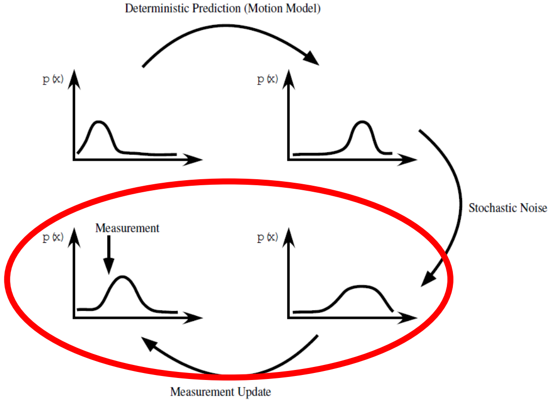


Figure 2.5: The main steps performed when Kalman filtering.

Kalman filters

Observations:

- If R = 0, then: $\mathbf{K}_t = \mathbf{P}_t^- \mathbf{H}^T (\mathbf{H}\mathbf{P}_t^- \mathbf{H}^T + \mathbf{R})^{-1} \Rightarrow \mathbf{K}_t = \mathbf{H}^{-1}$

  which leads to $\hat{\mathbf{x}}_t = \hat{\mathbf{x}}_t^- + \mathbf{K}_t (\mathbf{z}_t - \mathbf{H}\hat{\mathbf{x}}_t^-) \Rightarrow \hat{\mathbf{x}}_t = \mathbf{H}^{-1}\mathbf{z}_t$

  Thus, when there is no measurement error, the prediction of the Kalman filter is equal to the measurement

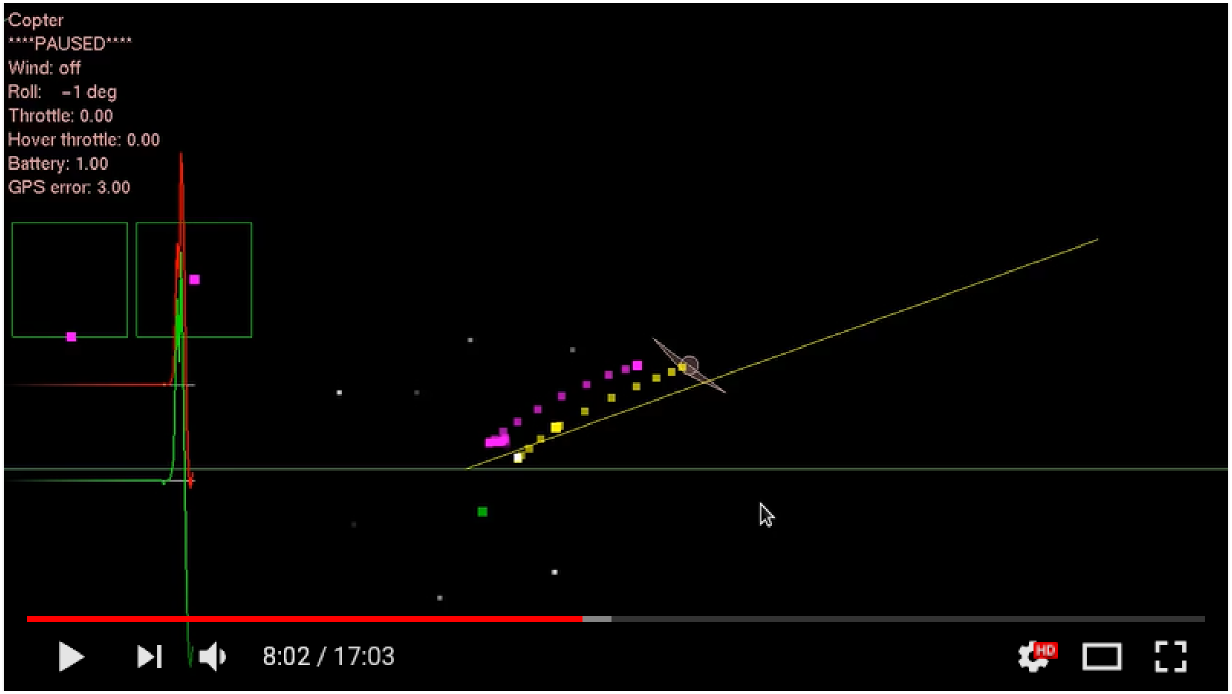
- If R → ∞, then $\mathbf{K}_t = \mathbf{P}_t^- \mathbf{H}^T (\mathbf{H}\mathbf{P}_t^- \mathbf{H}^T + \mathbf{R})^{-1} \Rightarrow 0$

  which leads to $\hat{\mathbf{x}}_t = \hat{\mathbf{x}}_t^- + \mathbf{K}_t (\mathbf{z}_t - \mathbf{H}\hat{\mathbf{x}}_t^-) \Rightarrow \hat{\mathbf{x}}_t = \hat{\mathbf{x}}_t^-$

  Thus, when there is a big measurement error, the prediction relies only on the predicted position

Kalman filters

Example:



https://www.youtube.com/watch?v=nNWWLJZRxAU

Kalman filters

Advantages:

- It can incorporate measurements from many sources to improve upon the state estimation
- It is a recursive method: it doesn't require storing all previous information. Instead the previous information is implicitly incorporated in the state of the previous step

Disadvantages:

- Some systems are not well-described by linear equations.
- It assumes unimodal distribution for the measurements. In several cases, like when tracking many targets (which are similar to each other) it can fail

Particle filters

The particle filters (also known as the CONDENSATION algorithm) improve over the Kalman filters in that:

- their prediction and update equations need not be linear
- they can exploit distributions that can be multimodal

The removal of these limitations makes Particle filters good candidates for tracking in cluttered environments

However, they are dependent on sampling methods

Particle filters

Particle filters use a set of (multiple) observations $\mathbf{z}_t$

Given these observations, they try to estimate the conditional probability

$$p\left(\mathbf{x}_t \middle| \mathbf{z}_t\right)$$

which is the probability distribution describing the current state, given all the measurements that have been observed

Particle filters

Particle filters use a set of (multiple) observations $\mathbf{z}_t$

Given these observations, they try to estimate the conditional probability

$$p\left(\mathbf{x}_t | \mathbf{z}_t\right)$$

which is the probability distribution describing the current state, given all the measurements that have been observed

Using the Bayes' rule: $$p\left(\mathbf{x}_t | \mathbf{z}_t\right) = kp\left(\mathbf{z}_t | \mathbf{x}_t\right) p\left(\mathbf{x}_t\right)$$

scaling factor

Particle filters

In order to obtain the multiple measurements, a sampling process is applied, leading to a set of N samples (also called particles) $\left\{\mathbf{s}_t^{(1)} \ldots, \mathbf{s}_t^{(N)}\right\}$

These samples are then weighted according to

$$\pi_t^n = \frac{p\left(\mathbf{z}_t | \mathbf{x}_t = \mathbf{s}_t^{(n)}\right)}{\sum_{j=1}^{N} p\left(\mathbf{z}_t | \mathbf{x}_t = \mathbf{s}_t^{(j)}\right)}$$

where $p\left(\mathbf{z}_t | \mathbf{x}_t = \mathbf{s}_t^{(n)}\right)$ is the probability of observing a measurement, given that the state is $\mathbf{x}_t = \mathbf{s}_t^{(n)}$

The sample with the highest probability is selected as the predicted state

Particle filters

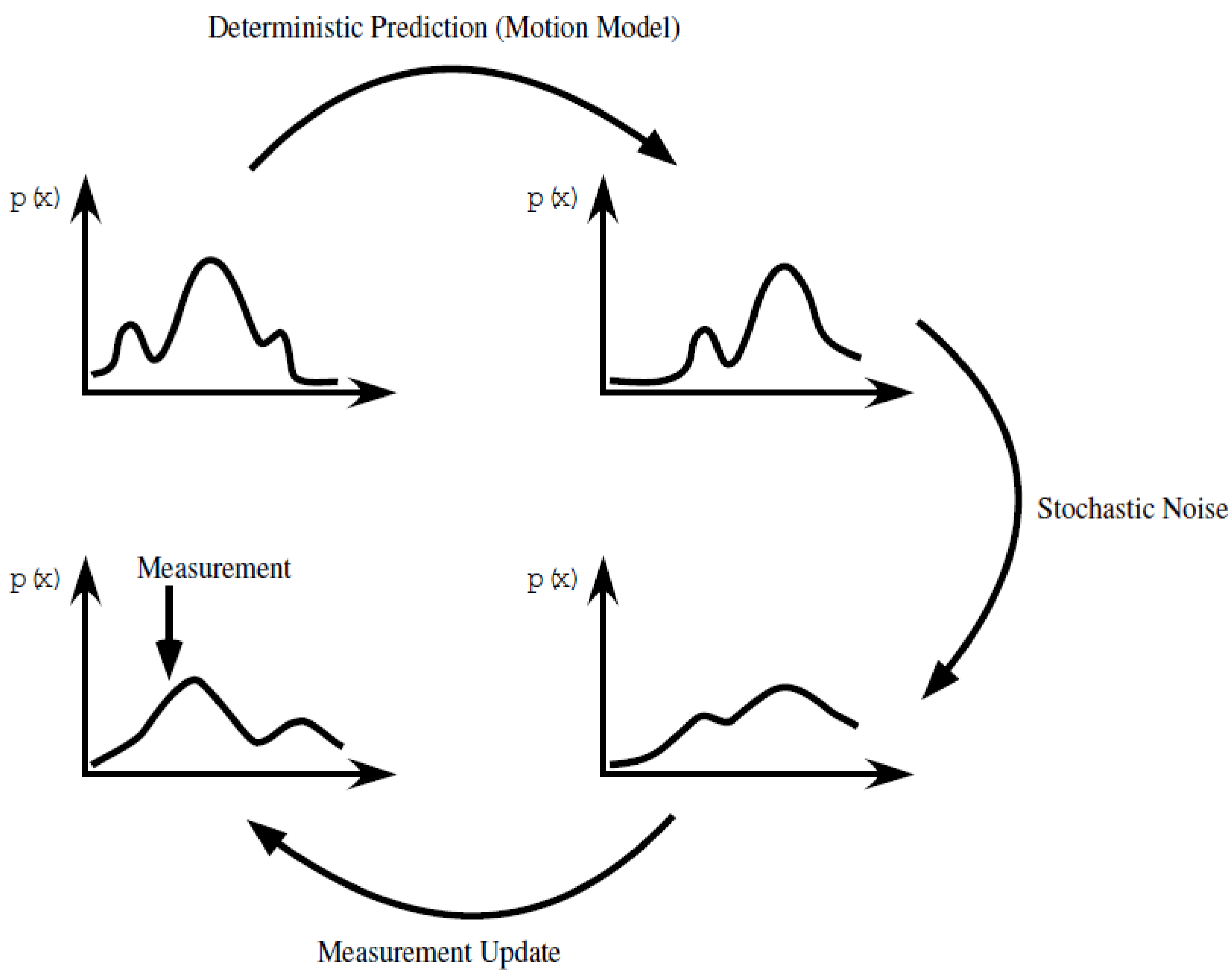
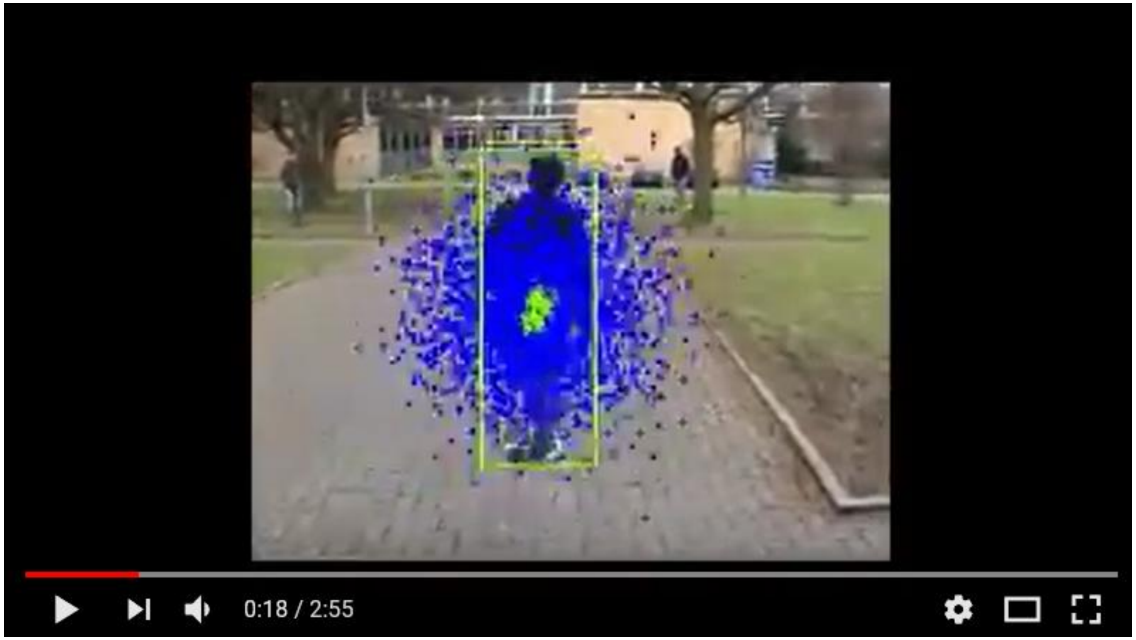


Figure 2.6: The main steps performed when particle filtering.

Particle filters

Example:



https://www.youtube.com/watch?v=O1FZyWz_yj4

Mean Shift Tracking

Used for tracking objects in the level of templates (bounding boxes)

Steps applied by template matching trackers:

1. Initialize the template on the target region (first frame)
2. Predict where the target will appear in the next frame
3. Create candidates in the neighborhood of the predicted position
4. Find the candidate that is more similar to the template (this is the prediction)
5. (Optionally) update the template using the best candidate
6. Go to step 2

Mean Shift Tracking

Main components of Mean Shift Algorithm:

- Representation of the template and candidate regions using histograms
- Use of a histogram distance/similarity metric
- Use of a method for locating the best candidate match at every frame

Mean Shift Tracking

Histogram definition:

- Use of color values
- Partition the range of colors in a number of bins
- Count the votes of each bin for the template (bounding box) of interest. The number of votes per bin can be obtained by

$$q_u = C \sum_{i=1}^{n} \delta \left[b \left(\mathbf{x}_i^* \right) - u \right]$$

where $\mathbf{x}_i^*$ = [p_x,p_y], C is a scaling value (so that the sum of all elements in the histogram will equal to 1 and δ is the Dirac delta function

Mean Shift Tracking

Histogram definition:

In Mean Shift tracking a modified version of histogram is used:

- Pixels in the center of the bounding box are more probable to belong to the object of interest
- Pixels in the borders of the bounding box are more probable to belong to background

Thus, the contribution of each pixel in the histogram is weighted based on its spatial location in the bounding box.

Mean Shift Tracking

Histogram definition:

Thus, the contribution of each pixel in the histogram is weighted based on its spatial location in the bounding box.

This is done by using a kernel function k(·):

$$\hat{q}_u = C \sum_{i=1}^{n} k\left(\left|\left|\mathbf{x}_i^*\right|\right|^2\right) \delta\left[b\left(\mathbf{x}_i^*\right) - u\right]$$

When evaluating several target candidates, the histograms for the target candidates are evaluated using:

$$\hat{p}_u = C \sum_{i=1}^{n} k\left(\left|\left|\frac{\mathbf{y} - \mathbf{x}_i^*}{h}\right|\right|^2\right) \delta\left[b\left(\mathbf{x}_i^*\right) - u\right]$$

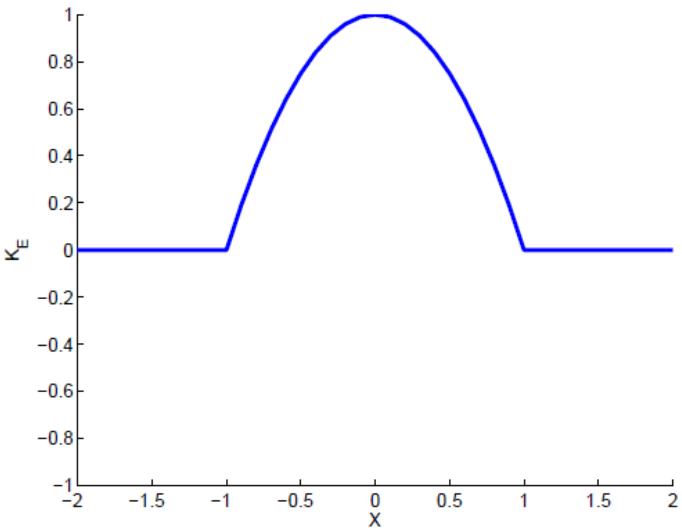
where **y** is the center of the target candidate's tracking window and h is the bandwidth of the kernel

Mean Shift Tracking

Histogram definition:

The kernel function k(·) can have several forms. The most popular one is:

$$K_E(x) = \begin{cases} \frac{1}{2} c_d^{-1} (d+2) \left(1 - ||\mathbf{x}||^2\right), & \text{if} \quad ||\mathbf{x}|| < 1 \\ 0, & \text{otherwise} \end{cases}$$



Mean Shift Tracking

<u>Histogram distance/similarity metric:</u>

After defining a histogram of the template (object) and the candidates, we evaluate the similarity of each candidate with the template by calculating the Bhattacharyya coefficient:

$$\hat{\rho}(\mathbf{y}) \equiv \hat{\rho}(\hat{\mathbf{p}}(\mathbf{y}), \hat{\mathbf{q}}) = \sum_{u=1}^{m} \sqrt{\hat{p}_u(\mathbf{y})\, \hat{q}_u}$$

Mean Shift Tracking

Target position estimation:

After defining a histogram of the template (object) and the candidates, we evaluate the similarity of each candidate with the template by calculating the Bhattacharyya coefficient:

$$\hat{\rho}(\mathbf{y}) \equiv \hat{\rho}(\hat{\mathbf{p}}(\mathbf{y}), \hat{\mathbf{q}}) = \sum_{u=1}^{m} \sqrt{\hat{p}_u(\mathbf{y})\, \hat{q}_u}$$

In order to minimize the distance between the two histograms (**p** and **q**) ρ needs to be maximized with respect to the target position **y**

Mean Shift Tracking

<u>Target position estimation:</u>

After defining a histogram of the template (object) and the candidates, we evaluate the similarity of each candidate with the template by calculating the Bhattacharyya coefficient:

$$\hat{\rho}(\mathbf{y}) \equiv \hat{\rho}(\hat{\mathbf{p}}(\mathbf{y}), \hat{\mathbf{q}}) = \sum_{u=1}^{m} \sqrt{\hat{p}_u(\mathbf{y})\, \hat{q}_u}$$

In order to minimize the distance between the two histograms (**p** and **q**) ρ needs to be maximized with respect to the target position **y**

In order to do this, we use an approximation

$$w_i = \sum_{u=1}^{m} \delta\left[b(\mathbf{x}_i) - u\right] \sqrt{\frac{\hat{q}_u}{\hat{p}_u(\hat{\mathbf{y}}_0)}}$$

$$\rho(\hat{\mathbf{p}}(\mathbf{y}), \hat{\mathbf{q}}) \approx \frac{1}{2} \sum_{u=1}^{m} \sqrt{\hat{p}_u(\hat{\mathbf{y}}_0)\, \hat{q}_u} + \frac{C_h}{2} \sum_{i=1}^{n_h} w_i k\left(\left\|\frac{\mathbf{y} - \mathbf{x}_i^*}{h}\right\|^2\right)$$

Mean Shift Tracking

Target position estimation:

After defining a histogram of the template (object) and the candidates, we evaluate the similarity of each candidate with the template by calculating the Bhattacharyya coefficient:

$$\hat{\rho}(\mathbf{y}) \equiv \hat{\rho}(\hat{\mathbf{p}}(\mathbf{y}), \hat{\mathbf{q}}) = \sum_{u=1}^{m} \sqrt{\hat{p}_u(\mathbf{y})\,\hat{q}_u}$$

In order to minimize the distance between the two histograms (**p** and **q**) ρ needs to be maximized with respect to the target position **y**

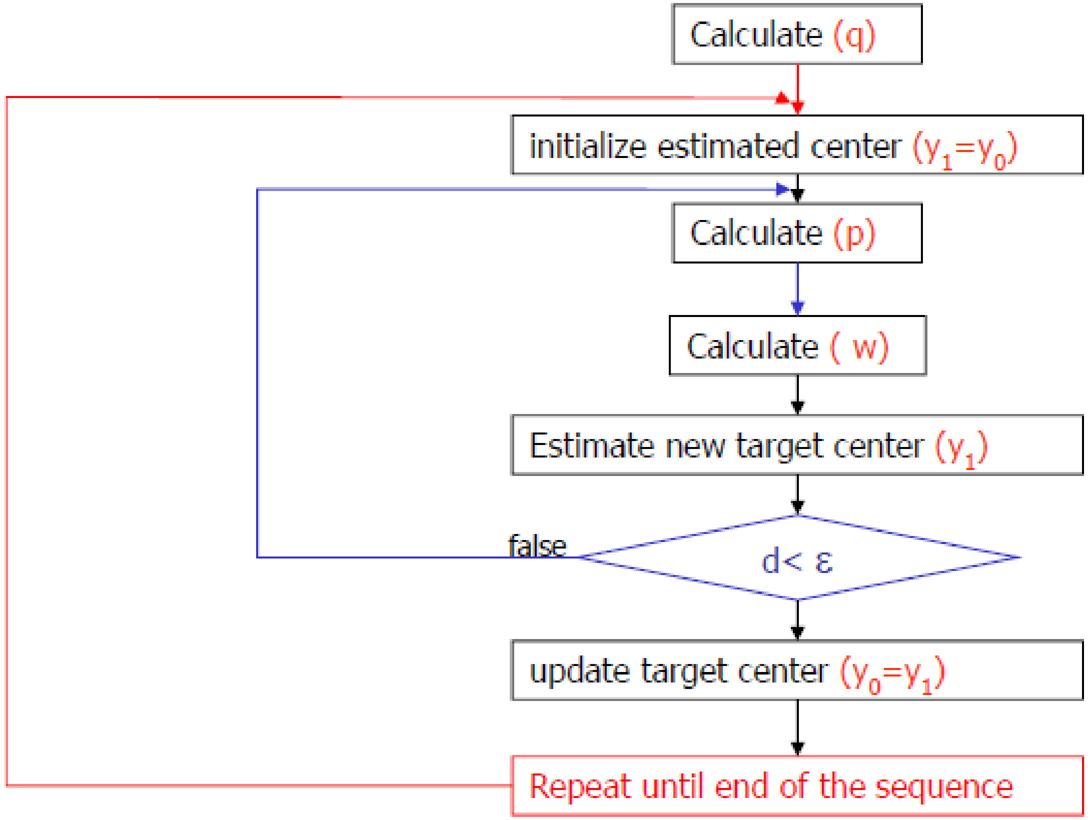
The optimal value for y is:

$$\hat{\mathbf{y}}_1 = \left[\frac{\sum_{i=1}^{n_h} \mathbf{x}_i^* w_i g\left(\left\| \frac{\hat{\mathbf{y}}_0 - \mathbf{x}_i^*}{h} \right\|^2 \right)}{\sum_{i=1}^{n_h} w_i g\left(\left\| \frac{\hat{\mathbf{y}}_0 - \mathbf{x}_i^*}{h} \right\|^2 \right)} \right]$$

where $g(x) = k'(x)$

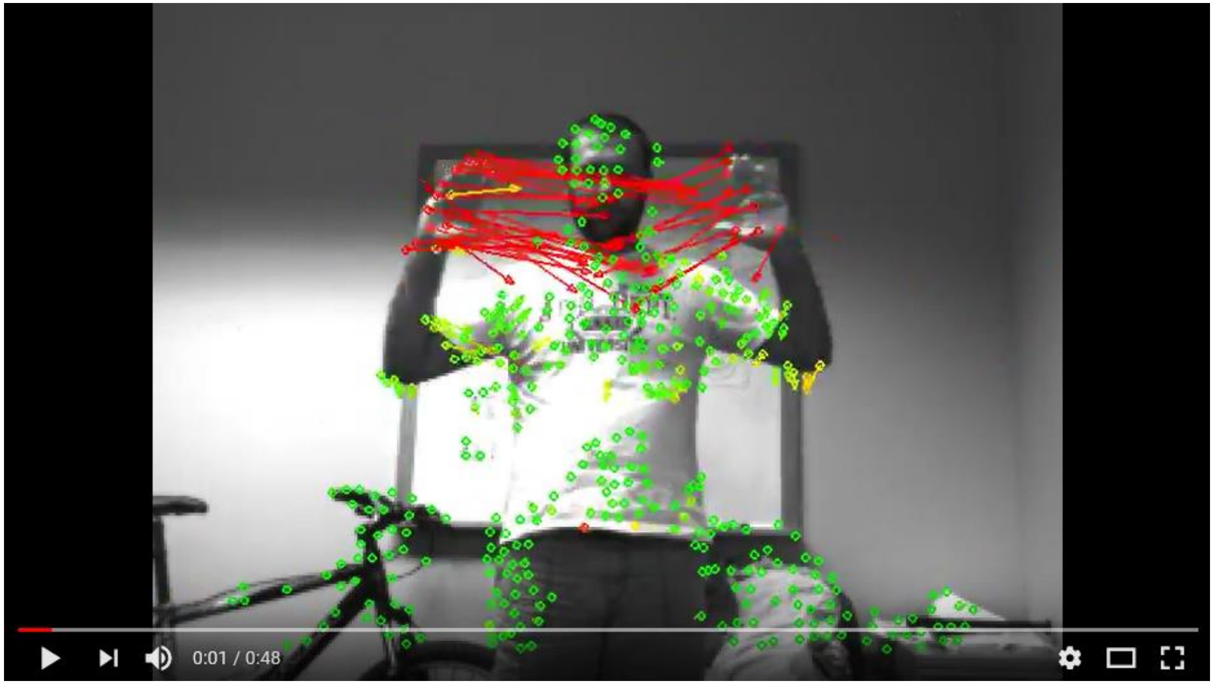
Mean Shift Tracking

Algorithm:



KLT tracking

Examples:



https://www.youtube.com/watch?v=E86NLzNbuL8

KLT tracking

Examples:

https://www.youtube.com/watch?v=PZW0UgPrsdk

KLT tracking

Problem statement:

- Given two images I and J (consecutive frames)
- Let $\mathbf{u} = [u_x, u_y]^T$ be a point in I
- Find the location in J of point $\mathbf{u}$
  - if the displacement is given by $\mathbf{d} = [d_x, d_y]^T$, then

$$\mathbf{v} = \mathbf{u} + \mathbf{d} = [u_x + d_x,\ u_y + d_y]^T$$

  - $\mathbf{d} = [d_x, d_y]^T$ is the optical flow at $\mathbf{d}$

KLT tracking

Problem statement:

- Given two images I and J (consecutive frames)
- Let **u** = $[u_x,u_y]^T$ be a point in I
- Find the location in J of point **u**
  - if the displacement is given by **d** = $[d_x,d_y]^T$, then

$$\mathbf{v} = \mathbf{u} + \mathbf{d} = [u_x+d_x,\ u_y+d_y]^T$$

  - **d** = $[d_x,d_y]^T$ is the optical flow at **d**
- In order to find v, we search in a window of size $(2\omega_x+1) \times (2\omega_y+1)$
- Residual function ε(d)

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

KLT tracking

Pyramid-based KLT:

- When the displacement of a pixel is larger than the search window size, KLT fails
- To handle such situations, we exploit image pyramids

KLT tracking

Pyramid-based KLT:

- When the displacement of a pixel is larger than the search window size, KLT fails
- To handle such situations, we exploit image pyramids
- We denote by I^0 the image at the 0^{th} level of the pyramid
- Images at higher level l=1,…,L (usually L is equal to 2-4) are obtained by downsizing images at previous levels

$$I^0 \rightarrow I^1 \rightarrow I^2 \rightarrow I^3 \rightarrow \ldots \rightarrow I^L$$

KLT tracking

Pyramid-based KLT:

- When the displacement of a pixel is larger than the search window size, KLT fails
- To handle such situations, we exploit image pyramids
- We denote by I^0 the image at the 0^{th} level of the pyramid
- Images at higher level l=1,…,L (usually L is equal to 2-4) are obtained by downsizing images at previous levels

$$I^0 \rightarrow I^1 \rightarrow I^2 \rightarrow I^3 \rightarrow \ldots \rightarrow I^L$$

- A point $\mathbf{u}_0$ on the image I^0 can be found on I^L at:

$$u^L = \frac{u^0}{2^L}$$

$\mathbf{d}^{Lm}$ is computed at the pyramid level L_m

↓

$\mathbf{d}^{Lm-1}$ is computed with an initial guess of $\mathbf{d}^{Lm}$ at L_{m-1}

↓

This continues up to the level 0

KLT tracking

Pyramid-based KLT:

- Let $\mathbf{g}^L = [g^L_x, g^L_y]^T$ be the initial guess at level L ($\mathbf{g}^L$ is available from level L to level L+1)

- Residual pixel displacement vector $\mathbf{d}^L = [d^L_x, d^L_y]^T$ that minimizes the new image matching error ε^L

$$\epsilon^L(\mathbf{d}^L) = \epsilon^L(d^L_x, d^L_y) = \sum_{x=u^L_x-\omega_x}^{u^L_x+\omega_x} \sum_{y=u^L_y-\omega_y}^{u^L_y+\omega_y} \left(I^L(x,y) - J^L(x+g^L_x+d^L_x, y+g^L_y+d^L_y)\right)^2$$

- The window of size $(2\omega_x+1)$x $(2\omega_y+1)$ is constant for all pyramid levels

- $\mathbf{g}^L$ is used to pre-translate the image patch in 2nd image J

- $\mathbf{d}^L$ is small and easy to compute with KLT algorithm

KLT tracking

Pyramid-based KLT:

- Assume that $\mathbf{d}^L$ is computed, new initial guess $\mathbf{g}^{L-1}$ at level L-1

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

- Initial guess for the deepest level of the pyramid (lowest resolution) L_m

$$\mathbf{g}^{L_m} = [0 \quad 0]^T$$

- The final optical flow solution **d**

$$\mathbf{d} = \sum_{L=0}^{L_m} 2^L \, \mathbf{d}^L$$

KLT tracking

Standard KLT optical flow computation:

- The goal is to find $\mathbf{d}^L$ that minimizes the matching function ε^L.
  (we drop the superscript since this is applied in all levels)

$$\forall (x, y) \in [p_x - \omega_x, p_x + \omega_x] \times [p_y - \omega_y, p_y + \omega_y],$$

$$A(x, y) \doteq I^L(x, y),$$

$$B(x, y) \doteq J^L(x + g_x^L, y + g_y^L)$$

- Let us change the displacement vector and the point vector

$$\overline{\nu} = [\nu_x \quad \nu_y]^T = \mathbf{d}^L$$

$$\mathbf{p} = [p_x \quad p_y]^T = \mathbf{u}^L$$

KLT tracking

Standard KLT optical flow computation:

- The goal is to find **v** that minimizes the matching function

$$\varepsilon(\overline{\nu}) = \varepsilon(\nu_x, \nu_y) = \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \left(A(x,y) - B(x+\nu_x, y+\nu_y)\right)^2$$

- To find **v** $\left.\frac{\partial \varepsilon(\overline{\nu})}{\partial \overline{\nu}}\right|_{\overline{\nu}=\overline{\nu}_{\text{opt}}} = [0 \quad 0]$

$$\frac{\partial \varepsilon(\overline{\nu})}{\partial \overline{\nu}} = -2 \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \left(A(x,y) - B(x+\nu_x, y+\nu_y)\right) \cdot \left[\frac{\partial B}{\partial x} \quad \frac{\partial B}{\partial y} \right]$$

KLT tracking

<u>Standard KLT optical flow computation:</u>

- We substitute $B(x+v_x, y+v_y)$ by its 1st order Taylor expansion at the point **v** = $[0,0]^T$

$$\frac{\partial \varepsilon(\overline{\nu})}{\partial \overline{\nu}} \approx -2 \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \Big(\underbrace{A(x,y) - B(x,y)}_{\text{frame difference}} - \underbrace{\begin{bmatrix} \frac{\partial B}{\partial x} & \frac{\partial B}{\partial y} \end{bmatrix}}_{\text{Image gradient}} \overline{\nu} \Big) \cdot \underbrace{\begin{bmatrix} \frac{\partial B}{\partial x} & \frac{\partial B}{\partial y} \end{bmatrix}}_{\text{Image gradient}}$$

$$G \doteq \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

and

$$\overline{b} \doteq \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \begin{bmatrix} \delta I \, I_x \\ \delta I \, I_y \end{bmatrix}$$

$$\frac{1}{2} \left[\frac{\partial \varepsilon(\overline{\nu})}{\partial \overline{\nu}} \right]^T \approx G\overline{\nu} - \overline{b}.$$

$$\overline{\nu}_{\text{opt}} = G^{-1} \, \overline{b}$$

where δI is a temporal image derivative

$$\delta I(x,y) \doteq A(x,y) - B(x,y)$$

KLT tracking

Standard KLT optical flow computation:

- It is valid only when the pixel displacement is small (due to the first order Taylor approximation)

- In order to handle situations where the pixels displacement is higher, iterative version is necessary

KLT tracking

Iterative KLT

Goal: Let $\mathbf{u}$ be a point on image I. Find its corresponding location $\mathbf{v}$ on image J

Build pyramid representations of I and J: $\{I^L\}_{L=0,\ldots,L_m}$ and $\{J^L\}_{L=0,\ldots,L_m}$

Initialization of pyramidal guess: $\mathbf{g}^{L_m} = [g_x^{L_m} \; g_x^{L_m}]^T = [0 \; 0]^T$

for $L = L_m$ **down to** 0 **with step of** -1

Location of point $\mathbf{u}$ on image I^L: $\mathbf{u}^L = [p_x \; p_y]^T = \mathbf{u}/2^L$

Derivative of I^L with respect to x: $I_x(x,y) = \dfrac{I^L(x+1,y) - I^L(x-1,y)}{2}$

Derivative of I^L with respect to y: $I_y(x,y) = \dfrac{I^L(x,y+1) - I^L(x,y-1)}{2}$

Spatial gradient matrix: $G = \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \begin{bmatrix} I_x^2(x,y) & I_x(x,y)\,I_y(x,y) \\ I_x(x,y)\,I_y(x,y) & I_y^2(x,y) \end{bmatrix}$

Initialization of iterative L-K: $\overline{\nu}^0 = [0 \; 0]^T$

for $k = 1$ **to** K **with step of** 1 (or until $\|\overline{\eta}^k\| <$ accuracy threshold)

Image difference: $\delta I_k(x,y) = I^L(x,y) - J^L(x + g_x^L + \nu_x^{k-1}, y + g_y^L + \nu_y^{k-1})$

Image mismatch vector: $\overline{b}_k = \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \begin{bmatrix} \delta I_k(x,y)\,I_x(x,y) \\ \delta I_k(x,y)\,I_y(x,y) \end{bmatrix}$

Optical flow (Lucas-Kanade): $\overline{\eta}^k = G^{-1}\,\overline{b}_k$

Guess for next iteration: $\overline{\nu}^k = \overline{\nu}^{k-1} + \overline{\eta}^k$

end of for-loop on k

Final optical flow at level L: $\mathbf{d}^L = \overline{\nu}^K$

Guess for next level $L-1$: $\mathbf{g}^{L-1} = [g_x^{L-1} \; g_y^{L-1}]^T = 2\,(\mathbf{g}^L + \mathbf{d}^L)$

end of for-loop on L

Final optical flow vector: $\mathbf{d} = \mathbf{g}^0 + \mathbf{d}^0$

Location of point on J: $\mathbf{v} = \mathbf{u} + \mathbf{d}$

Solution: The corresponding point is at location $\mathbf{v}$ on image J

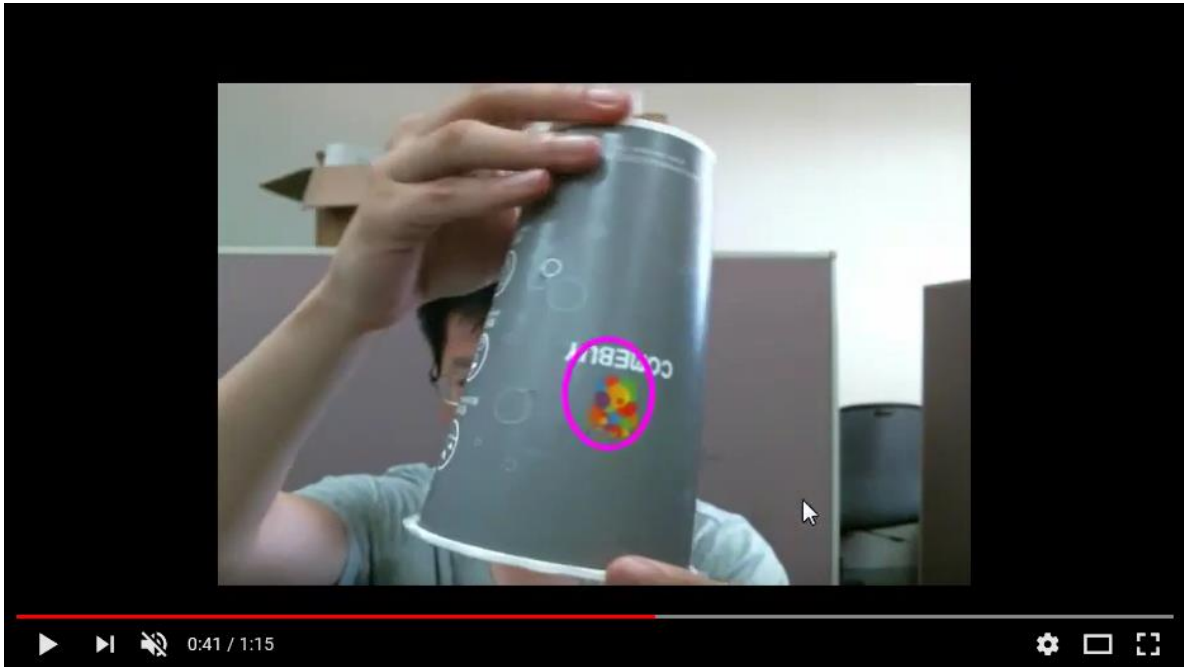
Tracking-Learning-Detection

Three components:

- Tracker: estimates the object's motion between consecutive frames under the assumption that the frame-to-frame motion is limited and the object is visible.
- Detector: performs full scanning of each frame independently to localize all appearances that have been observed and learned in the past.
- Learning: observes performance of the tracker and the detector
  - estimates detector's errors and generates training examples to avoid these errors in the future
  - Through learning, the detector generalizes to more object appearances and discriminates against the background.

Tracking-Learning-Detection

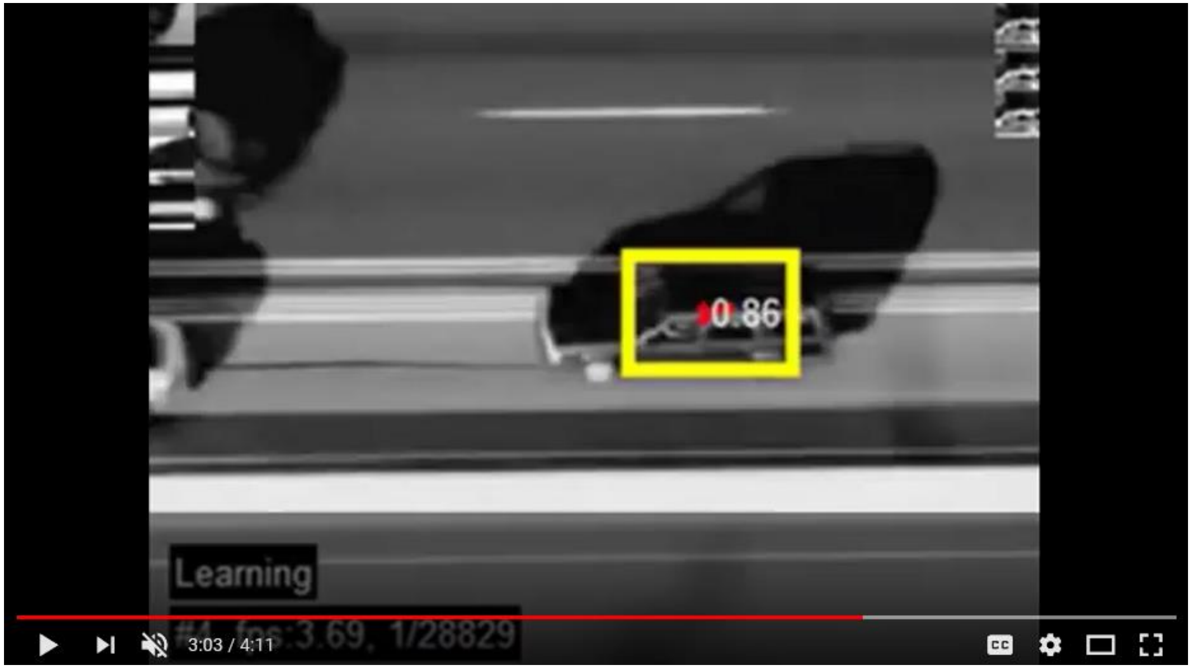
Examples:



https://www.youtube.com/watch?v=guksgD0Sxyc

Tracking-Learning-Detection

Examples:



https://www.youtube.com/watch?v=1GhNXHCQGsM

Tracking-Learning-Detection

Three components:

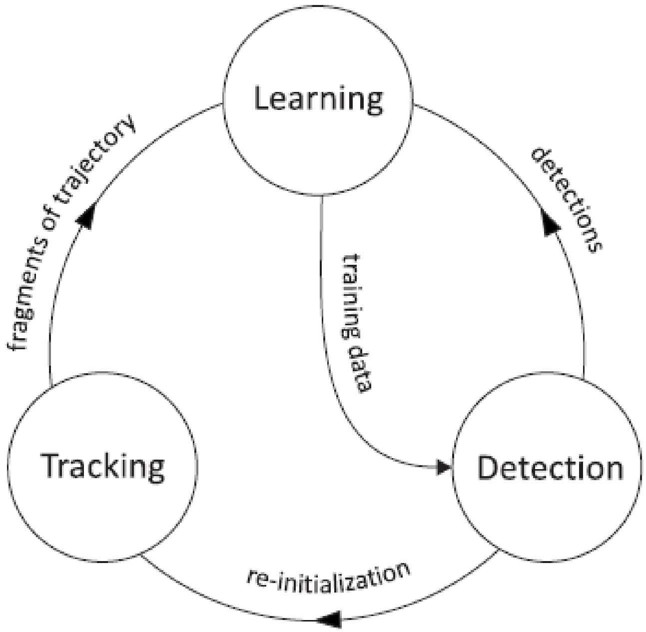


Fig. 2. The block diagram of the TLD framework.

Tracking-Learning-Detection

P-N learning:

- It is used to improve the performance of an object detector by online processing of the video stream.

In every frame:

- it evaluates the current detector
- identifies its errors
- updates the detector to avoid these errors in the future

The main idea of P-N learning is to use two (independent) “experts”

- P-expert identifies only false negatives
- N-expert: identifies only false positives.

Tracking-Learning-Detection

P-N learning:

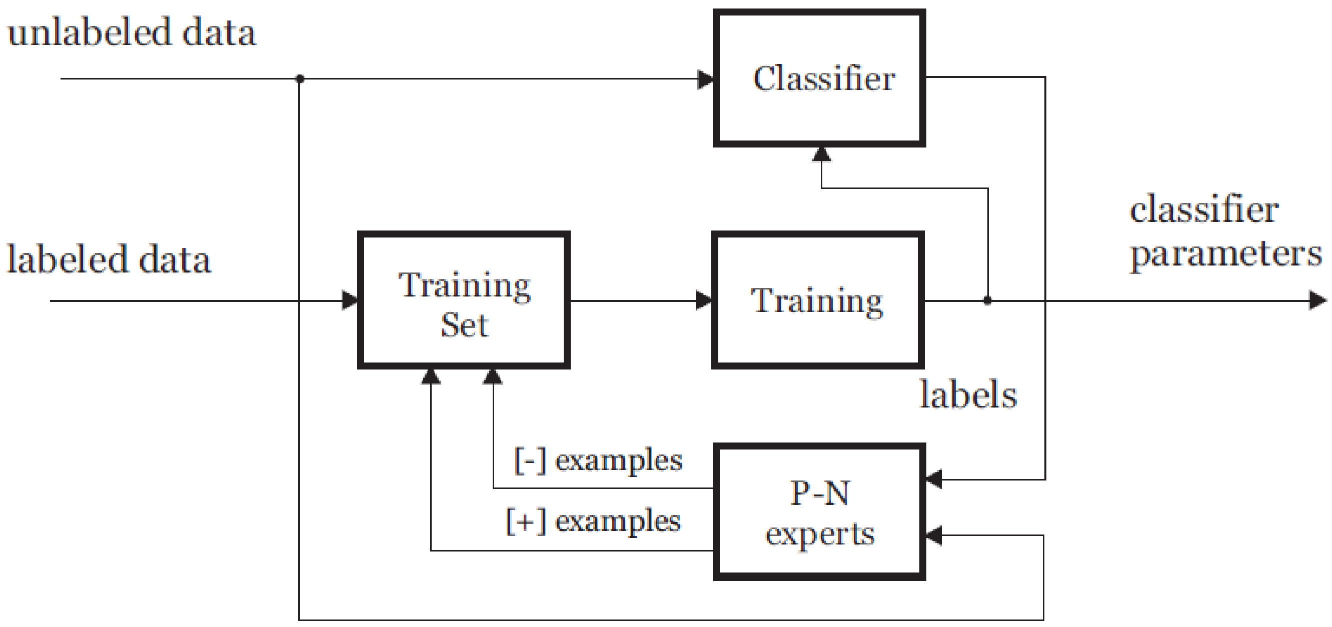


Fig. 3. The block diagram of the P-N learning.

Tracking-Learning-Detection

<u>P-N learning:</u>

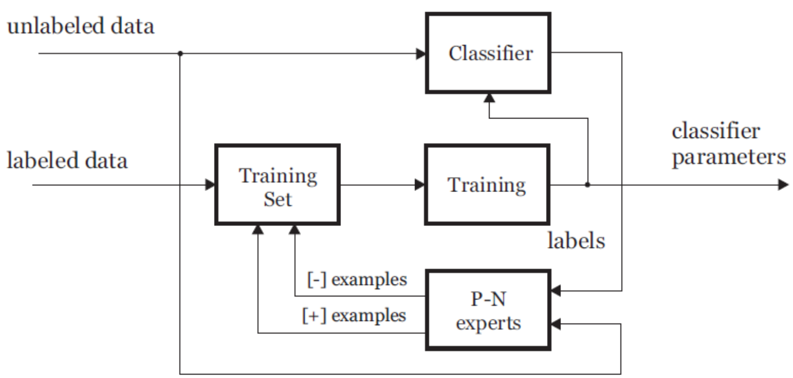
Some definitions:

- $X = \{x_1,\ldots,x_N\}$ is a set of samples
- $Y = \{-1,1\}$ is a set of labels

Using X and Y we can define:

- $L_l = \{(x_1, y_l), \ldots, (x_N, y_l)\}$ is a labeled training set with l samples
- $X_u = \{x_1,\ldots,x_u\}$ is an unlabeled set with u>>l samples (l+u=N)

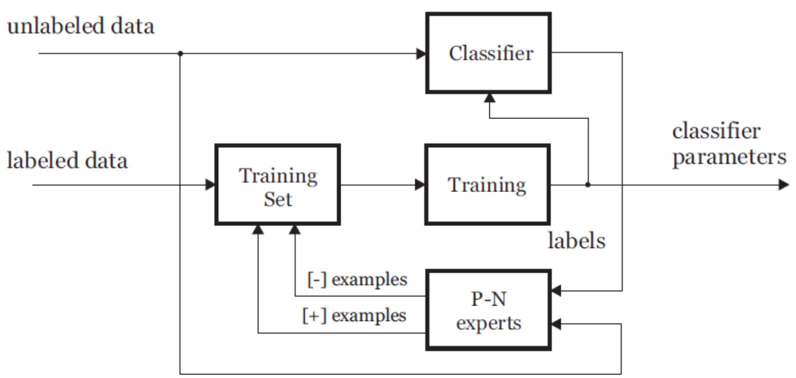
Using L_l and X_u the task of N-P learning is to learn a mapping $X \rightarrow Y$



Tracking-Learning-Detection

<u>P-N learning blocks:</u>

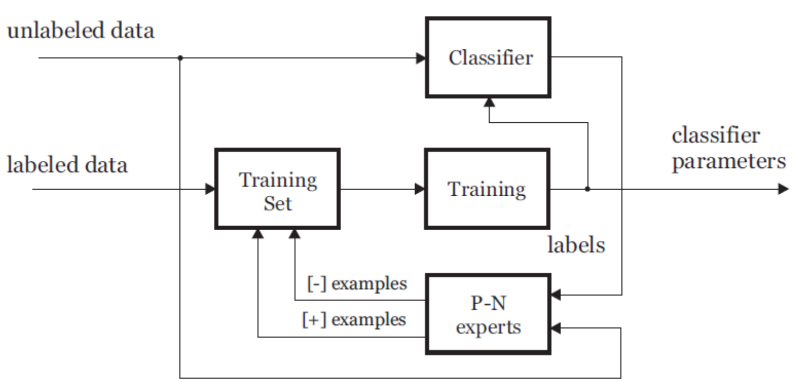
- a classifier to be trained
- a labeled training set
- classifier training using the labeled set
- P-N experts (functions) that generate positive and negative training samples during the learning process



Tracking-Learning-Detection

P-N learning processing steps:

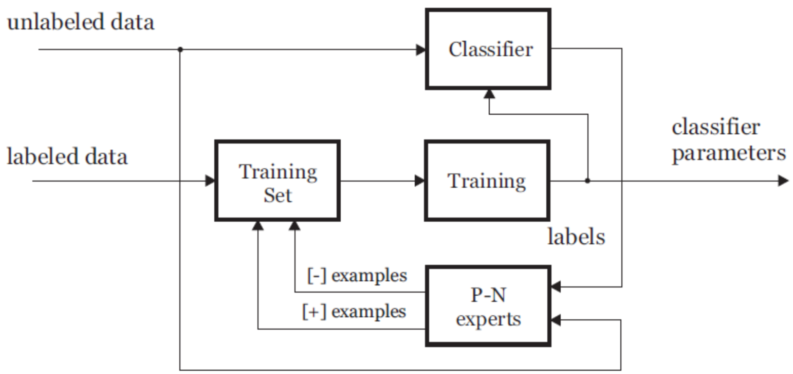
1. Initial classifier training using the labeled set L. This generates the classification model M(θ_0) with parameters θ_0



Tracking-Learning-Detection

P-N learning processing steps:

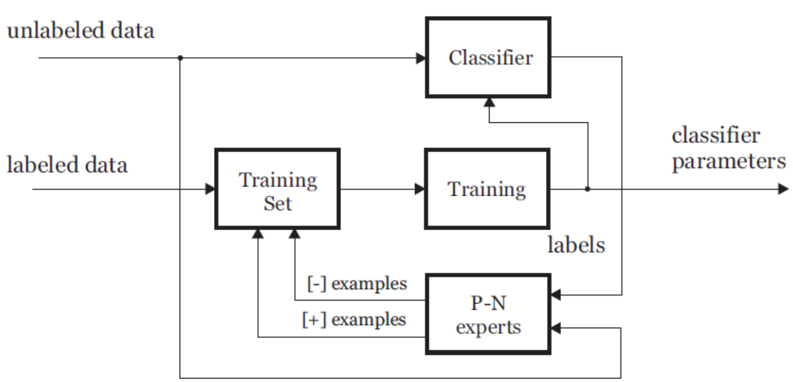
1. Initial classifier training using the labeled set L. This generates the classification model $M(\theta_0)$ with parameters θ_0
2. Application of an iterative process for t = 1,…,T:
   2.1 Use $M(\theta_{t-1})$ to classify the unlabeled set X_u



Tracking-Learning-Detection

P-N learning processing steps:

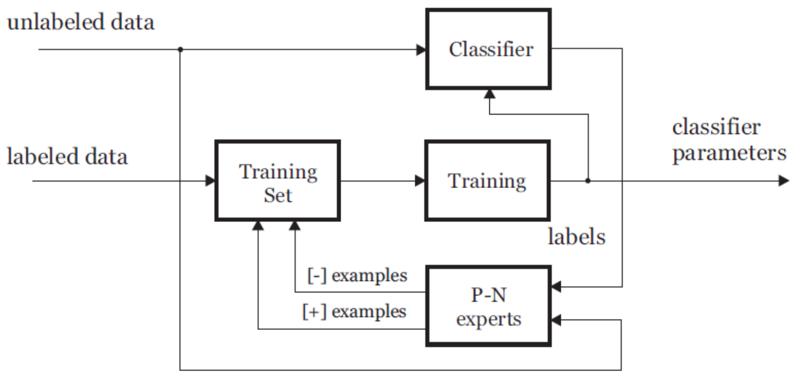
1. Initial classifier training using the labeled set L. This generates the classification model $M(\theta_0)$ with parameters θ_0
2. Application of an iterative process for t = 1,…,T:
   2.1 Use $M(\theta_{t-1})$ to classify the unlabeled set X_u
   2.2 Obtain the predicted labels $y_u(t)$ for all samples in X_u



Tracking-Learning-Detection

P-N learning processing steps:

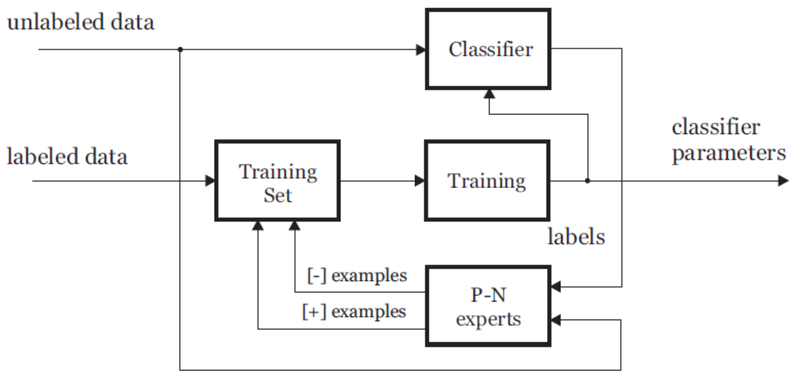
1. Initial classifier training using the labeled set L. This generates the classification model $M(\theta_0)$ with parameters θ_0
2. Application of an iterative process for t = 1,…,T:
   - 2.1 Use $M(\theta_{t-1})$ to classify the unlabeled set X_u
   - 2.2 Obtain the predicted labels $y_u(t)$ for all samples in X_u
   - 2.3 Analyze the classification results using the P and N experts



Tracking-Learning-Detection

P-N learning processing steps:

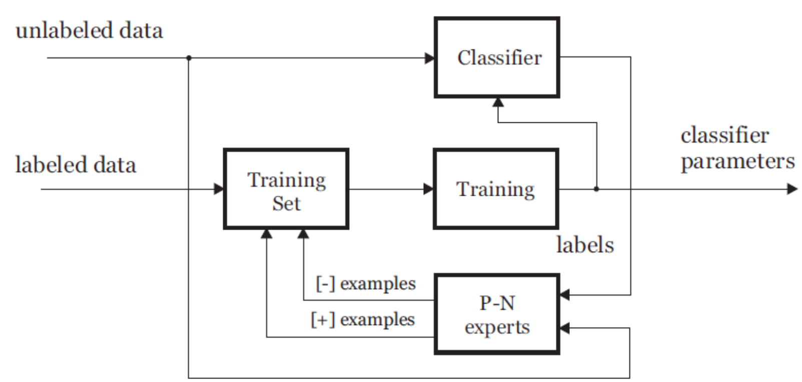
1. Initial classifier training using the labeled set L. This generates the classification model $M(\theta_0)$ with parameters θ_0
2. Application of an iterative process for t = 1,…,T:
   2.1 Use $M(\theta_{t-1})$ to classify the unlabeled set X_u
   2.2 Obtain the predicted labels $y_u(t)$ for all samples in X_u
   2.3 Analyze the classification results using the P and N experts
   2.4 P and N experts identify samples that have been miss-classified ($X_m(t)$)



Tracking-Learning-Detection

P-N learning processing steps:

1. Initial classifier training using the labeled set L. This generates the classification model $M(\theta_0)$ with parameters θ_0
2. Application of an iterative process for t = 1,…,T:
   - 2.1 Use $M(\theta_{t-1})$ to classify the unlabeled set X_u
   - 2.2 Obtain the predicted labels $y_u(t)$ for all samples in X_u
   - 2.3 Analyze the classification results using the P and N experts
   - 2.4 P and N experts identify samples that have been miss-classified ($X_m(t)$)
   - 2.5 Augment the labeled set L by adding the set $X_m(t)$ and changed labels
3. Termination if convergence

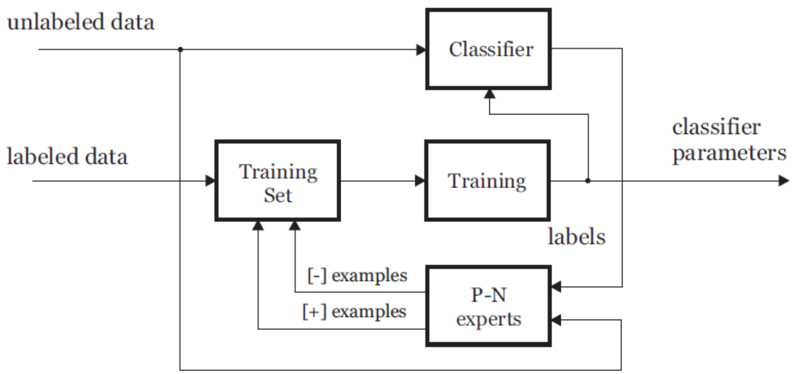


Tracking-Learning-Detection

P-N learning processing steps:

1. Initial classifier training using the labeled set L. This generates the classification model $M(\theta_0)$ with parameters θ_0
2. Application of an iterative process for t = 1,…,T:
   - 2.1 Use $M(\theta_{t-1})$ to classify the unlabeled set X_u
   - 2.2 Obtain the predicted labels $y_u(t)$ for all samples in X_u
   - 2.3 Analyze the classification results using the P and N experts
   - 2.4 P and N experts identify samples that have been miss-classified ($X_m(t)$)
   - 2.5 Augment the labeled set L by adding the set $X_m(t)$ and changed labels
3. Termination if convergence

The P-N experts play a crucial role in the above process



Tracking-Learning-Detection

P-N experts exploit the following ideas:

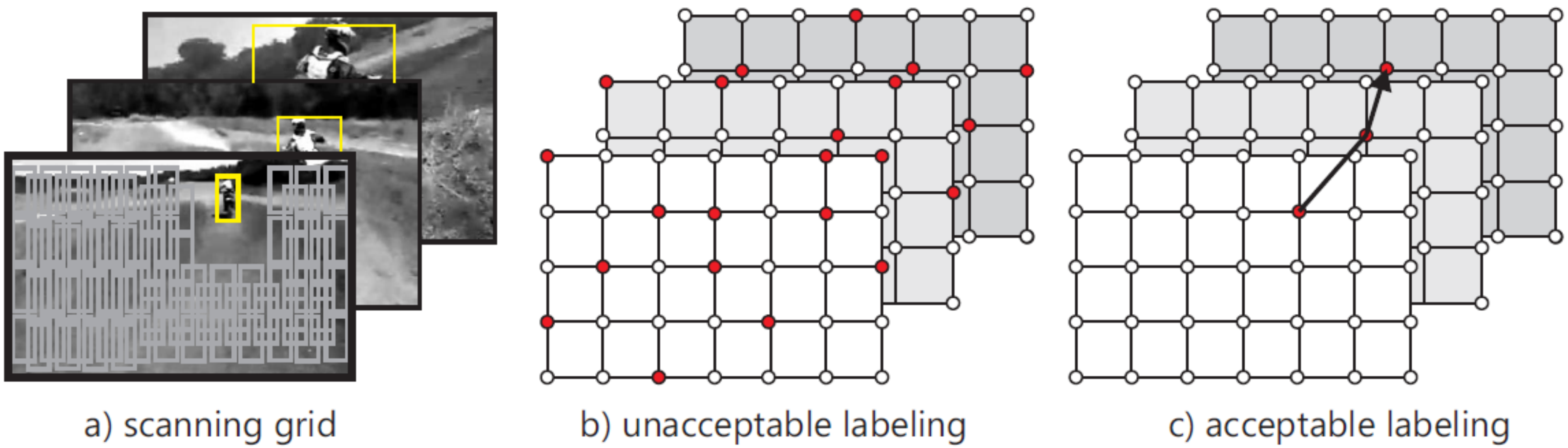


Fig. 6. Illustration of a scanning grid and corresponding volume of labels. Red dots correspond to positive labels.

Tracking-Learning-Detection

P-N experts exploit the following ideas:

P expert:

- exploits the **temporal structure** in the video and assumes that the object moves along a trajectory.
- It remembers the location of the object in the previous frame and estimates the object location in current frame using a tracker.
- If the detector labeled a location belonging to an object trajectory as negative (false negative error), the P-expert generates a positive example.

Tracking-Learning-Detection

P-N experts exploit the following ideas:

N expert:

- exploits the **spatial structure** in the video and assumes that the object can appear at a single location only for each frame
- It analyzes:
  - all responses of the detector in the current frame
  - the response produced by the tracker

  and selects the one that is the most confident.
- Patches that are not overlapping with the selected patch are labeled as negative.

The selected (maximally confident) patch re-initializes the location of the tracker.

Tracking-Learning-Detection

Example of P-N experts decisions:

- Yellow box: tracker's prediction (P experts most probable output)
- Black boxes: used by N expert to correct the prediction errors

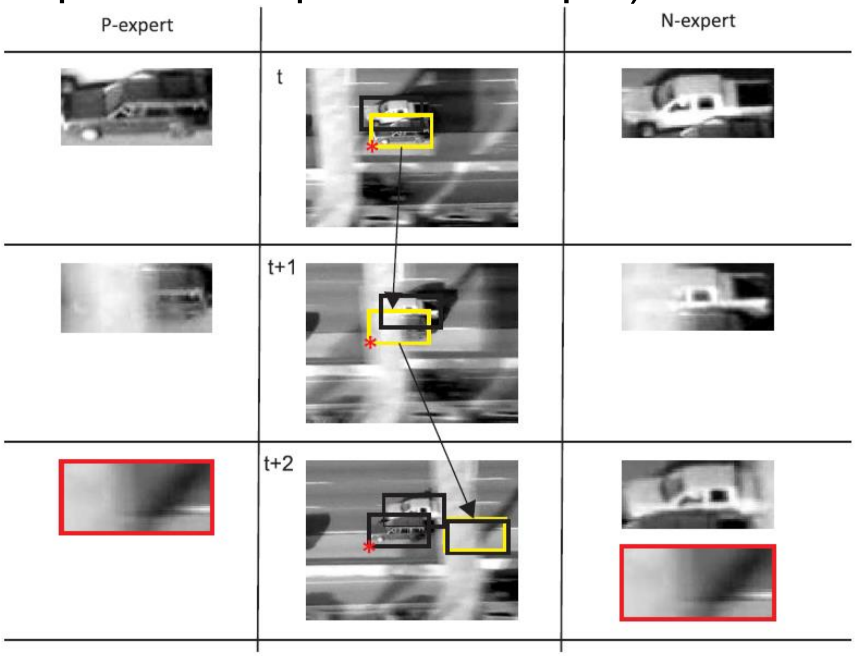


Fig. 7. Illustration of the examples output by the P-N experts. The third row shows error compensation.

Tracking-Learning-Detection

Object model:

- It represents the object and the background (the surroundings of the object)
- It is a set of positive and negative patches (positive patches are ordered w.r.t. time)

$$M = \{p_1^+, p_2^+, \ldots, p_m^+, p_1^-, p_2^-, \ldots, p_n^-\}$$

- All patches are rescaled to a fixed size BB (15x15 pixels)

Tracking-Learning-Detection

Given the object model M and a patch p, we define several similarity measures:

- Similarity with the positive nearest neighbor $S^+(p, M) = \max_{p_i^+ \in M} S(p, p_i^+)$

- Similarity with the negative nearest neighbor $S^-(p, M) = \max_{p_i^- \in M} S(p, p_i^-)$

- Similarity with the positive nearest neighbor considering 50% earliest positive patches $$S^+_{50\%}(p, M) = \max_{p_i^+ \in M \wedge i < \frac{m}{2}} S(p, p_i^+)$$

- Relative (positive-negative) similarities (ranging from 0 to 1 – higher values means highest confidence that the patch depicts the object)

$$S^r = \frac{S^+}{S^+ + S^-} \qquad S^c = \frac{S^+_{50\%}}{S^+_{50\%} + S^-}$$

Tracking-Learning-Detection

In all the above we use the Normalized Correlation Coefficient:

$$\mathrm{S}(p_i, p_j) = 0.5(\mathrm{NCC}(p_i, p_j) + 1)$$

$$NCC(p_i, p_j) = \frac{p_i^T p_j}{\|p_i\| \|p_j\|}$$

The relative similarity is used to define a Nearest Neighbor classifier:
- A patch p is classified as positive if $S^r(p, M) > \theta_{\mathrm{NN}}$
- A classification margin is defined as $S^r(p, M) - \theta_{\mathrm{NN}}$

The parameter θ_{NN} is used in order to update the model M

A patch is added to the collection (the model is updated) if:
- its label predicted by the NN classifier is different from the label given by the P-N experts
- its classification margin is smaller than a parameter value λ (e.g. λ = 0.1)

Tracking-Learning-Detection

Object detector:

- Uses a sliding window with:
  - scale step of 1.2
  - horizontal step of 10% of the width of the initial bounding box
  - vertical step of 10% of the height of the initial bounding box
  - minimum bounding box size of 20 pixels
- In an image of 240x320 pixels, this process leads to around 50k windows

Tracking-Learning-Detection

Object detector:

- Uses a sliding window with:
  - scale step of 1.2
  - horizontal step of 10% of the width of the initial bounding box
  - vertical step of 10% of the height of the initial bounding box
  - minimum bounding box size of 20 pixels
- In an image of 240x320 pixels, this process leads to around 50k windows
- Uses a cascaded classifier with three steps:
  1. rejects all patches with gray-value variance smaller than 50% of variance of the object's patch. The calculation of the variance is fast and rejects around 50% of windows
  2. An ensemble of fast (binary) randomized classifiers based on binarized features
  3. Nearest Neighbor classifier using the object model M

Tracking-Learning-Detection

Tracker:

- Uses point predictions between consecutive video frames (KLT tracker)
- Failure detection if

$$\mathrm{median}|d_i - d_m| > 10$$

where d_i is the displacement of point i and d_m is the median displacement

Tracking-Learning-Detection

Tracker:
- Uses point predictions between consecutive video frames (KLT tracker)
- Failure detection if

$$\text{median}|d_i - d_m| > 10$$

where d_i is the displacement of point i and d_m is the median displacement

Integrator:
- Integrates the bounding boxes predicted by the tracker and the detector.
- The final bounding box is the one that provides the maximum S^c similarity value

$$S^c = \frac{S^+_{50\%}}{S^+_{50\%} + S^-}$$

Tracking-Learning-Detection

Learning component:

- Initialization in the first frame
- Given the initial bounding box of the object (given by the user) we generate 200 positive patches
  - we select 10 bounding boxes on the scanning grid that are closest to the initial bounding box
  - For each bounding box, we generate 20 warped versions by applying geometric transformations (shift of 1%, scale change of 1%, in-plane rotation of 10^{o}) and add them after adding Gaussian noise (σ=5) on pixels
- Negative patches are generated from the surrounding of the initial bounding box, so that they do not overlap with it

Tracking-Learning-Detection

Learning component:

- P-expert: tries to identify reliable patches in the current frame using the object model M
- In every frame
  - It outputs a decision about the reliability of the current location using the S^c similarity
  - If the current location is reliable, the P-expert generates 100 positive examples that update the object model and the ensemble classifier
    - we select 10 bounding boxes on the scanning grid that are closest to the current bounding box
    - for each bounding box, we generate 10 warped versions by geometric transformations (shift 1%, scale change 1%, in-plane rotation 5^o) and add them after adding Gaussian noise (σ=5) on pixels

Tracking-Learning-Detection

Learning component:

- N-expert: generates negative training examples
  - discovers clutter in the background against which the detector should discriminate
  - assumes that the object can occupy at most one location in the image. Therefore, if the object location is known, its neighborhood (overlap < 0.2) is labeled as negative
  - It is applied if the trajectory is reliable (at the same time as P-expert)
  - For the update of the object detector and the ensemble classifier, we consider only those patches that were not rejected by the variance and the ensemble classifier.