# TSBB15
# Computer Vision

## Lecture 6
## Clustering and Learning

# Why learning?

- Learning is a very important part of Computer Vision:

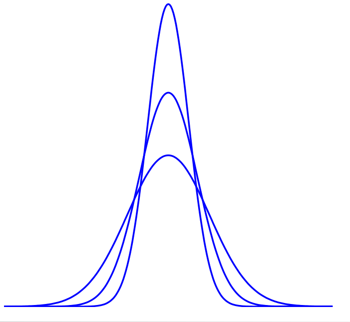
1. Parameter tuning
2. Adaptation to changing conditions
3. Finding patterns in data

# Parameter tuning

- Most Computer Vision systems are complex pieces of software.
- The more complex a system is, the more parameters it has.

# Parameter tuning

- Most Computer Vision systems are complex pieces of software.
- The more complex a system is, the more parameters it has. E.g. filter sizes, thresholds for detection etc. These need to be tuned!

# Parameter tuning

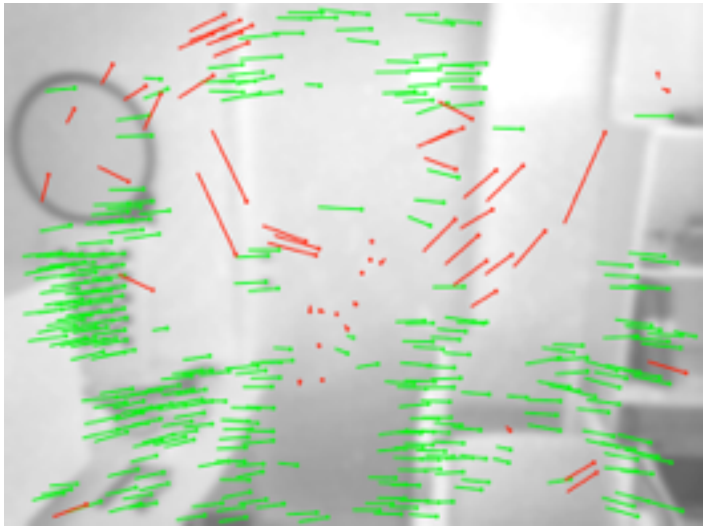
- Tuning in brief:

  1. Give examples of the desired behaviour of an algorithm.
  2. Look for the parameters that produce the desired behaviour.

  If you let the computer look for the parameters, tuning becomes learning.

# Parameter tuning

- Example:
  Automatically decide which motion vectors are good($\mathbf{v} \in G$) and which are bad($\mathbf{v} \in B$).



- Look for **tracker parameters** that maximise:
  $J(p_1,...,p_N) = |G|/(|G|+|B|)$

# Adaptation

- Computer Vision systems that are deployed in live situations face changing conditions. E.g. different illumination at night and during the day.

# Adaptation

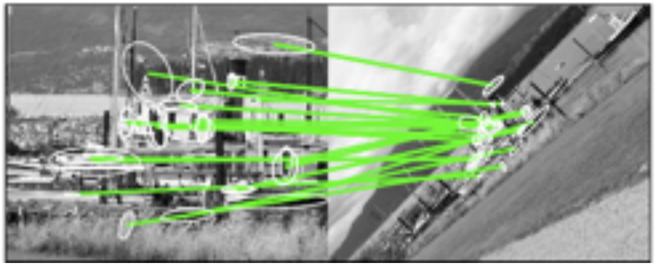
- Computer Vision systems that are deployed in live situations face changing conditions. E.g. different illumination at night and during the day.
- In order to cope with changes, a vision system needs to be adaptive.

# Adaptation

- Computer Vision systems that are deployed in live situations face changing conditions. E.g. different illumination at night and during the day.
- In order to cope with changes, a vision system needs to be adaptive.
- Example: Background models introduced later in this lecture.

# Finding patterns in data

- Recognition and matching (LE 8) uses learned features (or tuned).



- Applications such as: object recognition, object tracking, image captioning etc.

# Learning systems

- Batch learning: *learn once, use forever*

- Online learning: *learn continuously*

# Learning systems

- Batch learning: *learn once, use forever*
  Is used to automatically tune parameters, features, classifiers etc.
- Online learning: *learn continuously*
  Is used to automatically adapt e.g. classifiers and trackers to changing conditions.

# Today's topics

- Learning paradigms
- K-means clustering
- Mixture models and EM
- Background models
- Meanshift clustering
- Generalised Hough Transforms (GHT)
- Channel clustering

# Learning paradigms

- Different learning situations/paradigms:

  **Supervised learning**
  **Reinforcement learning**
  **Unsupervised learning**

- Covered in depth in:
  TBMI26 Neural Networks and Learning Systems

# Learning paradigms

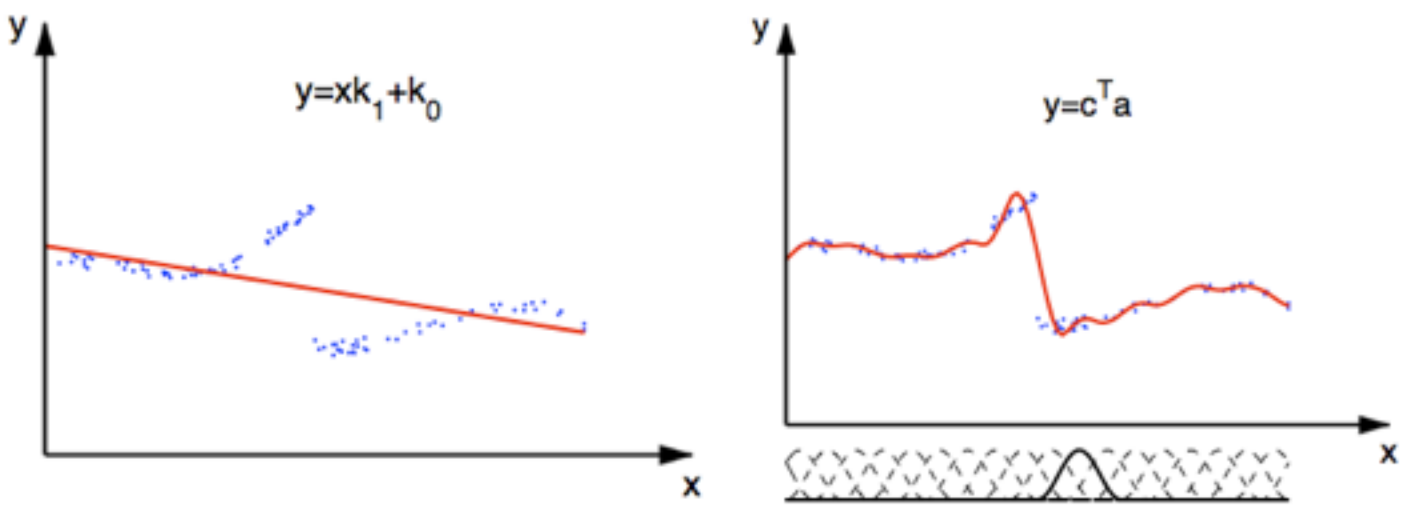
- Different learning situations/paradigms:

  **Supervised learning**
  **Reinforcement learning**
  **Unsupervised learning** ←this lecture

- Covered in depth in:
  TBMI26 Neural Networks and Learning Systems

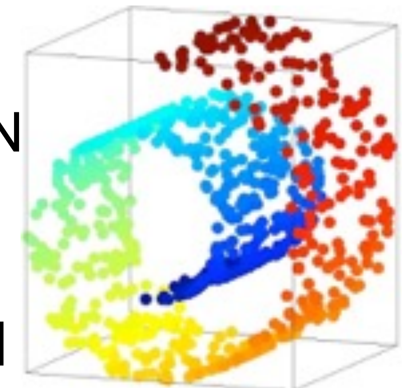
# Learning paradigms

- **Supervised learning**
  learn $y=f(\mathbf{x})$ from examples $\{\mathbf{x}_n,\mathbf{y}_n\}_1^N$
  = function approximation

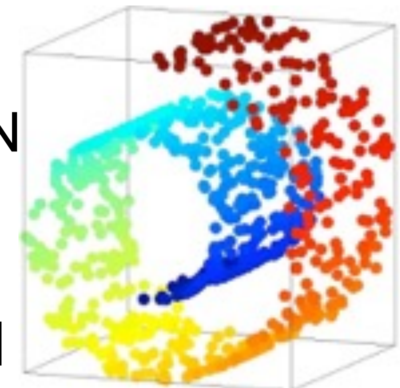


# Learning paradigms

- **Unsupervised learning**
  learn y=f(**x**) from examples $\{\mathbf{x}_n\}_1^N$
  =*manifold learning* or *clustering*
  - Manifold learning finds low dimensional representations of high dimensional data. E.g. coordinates on a surface in nD.



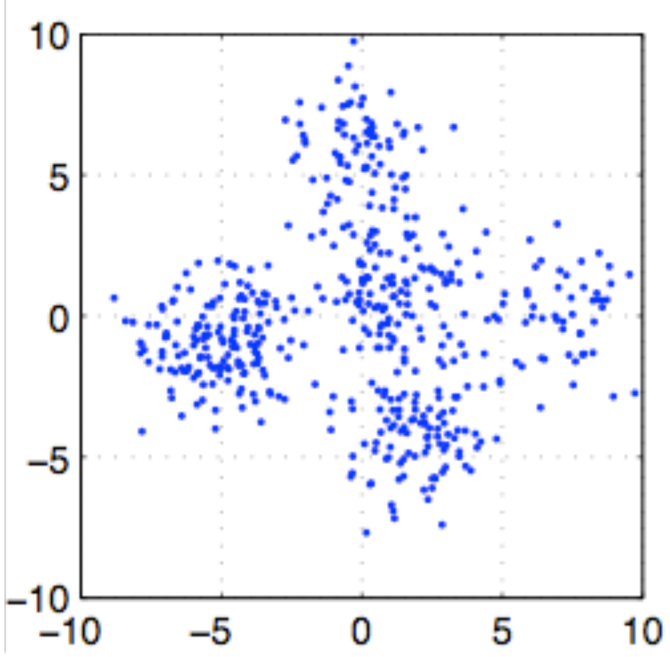
# Learning paradigms



- **Unsupervised learning**
  learn y=f(**x**) from examples $\{\mathbf{x}_n\}_1^N$
  =*manifold learning* or *clustering*
  - Manifold learning finds low dimensional representations of high dimensional data. E.g. coordinates on a surface in nD.
- This lecture is mainly about clustering.
- $y \in \mathbb{N}$, i.e. each sample $\mathbf{x}_n$ is assigned a cluster *label*.

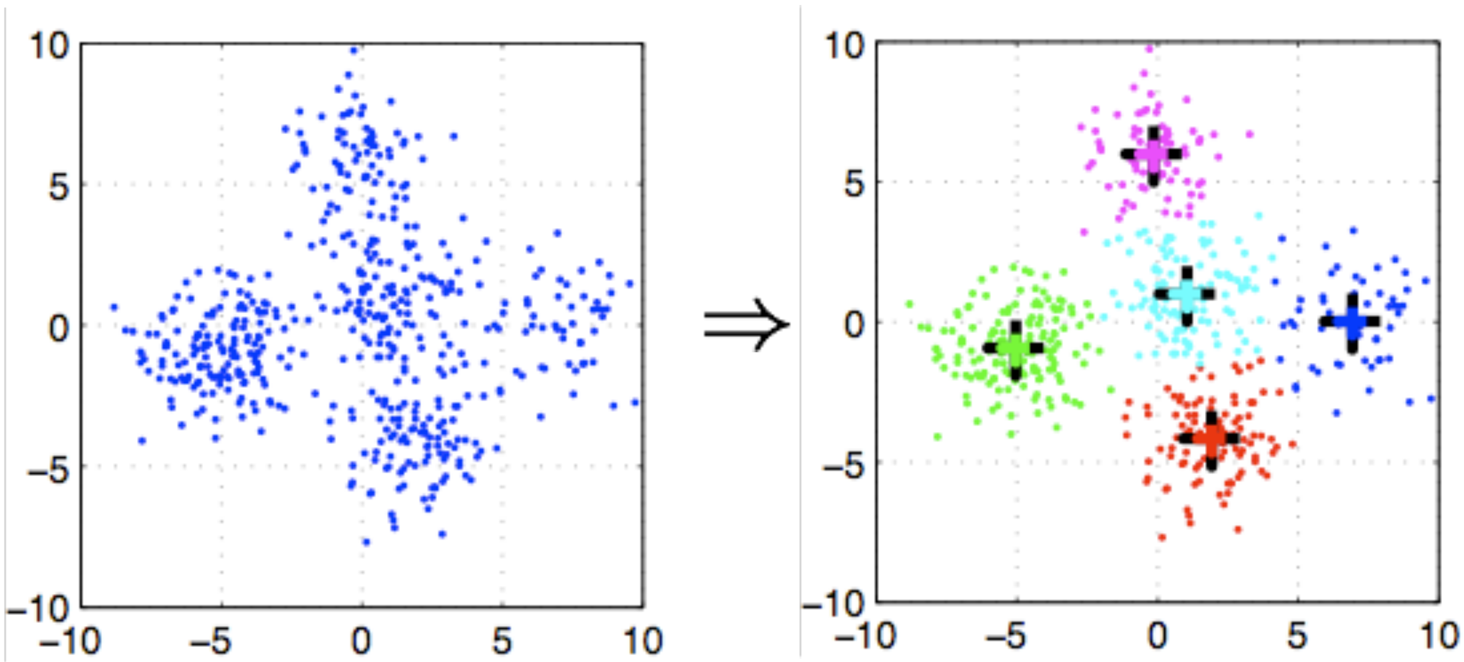
# Clustering

– Our input is a set of data points $\{\mathbf{x}_n\}_1^N$



# Clustering

– Each data point $\{\mathbf{x}_n\}_1^N$ is assigned a cluster label $y \in [1 \ldots K]$, and a prototype $\{\mathbf{p}_k\}_1^K$



# Clustering

- A good clustering has small distances between prototypes and samples within that cluster:

$$J(\mathbf{p}_1, \ldots \mathbf{p}_K) = \sum_{k=1}^{K} \sum_{n=1}^{N} \delta[y_n = k] ||\mathbf{x}_n - \mathbf{p}_k||^2$$

# Clustering

- A good clustering has small distances between prototypes and samples within that cluster:

$$J(\mathbf{p}_1, \ldots \mathbf{p}_K) = \sum_{k=1}^{K} \sum_{n=1}^{N} \delta[y_n = k] ||\mathbf{x}_n - \mathbf{p}_k||^2$$

- NP-complete problem.
- K-means clustering [MacQueen'67] is a useful heuristic.

# K-means clustering

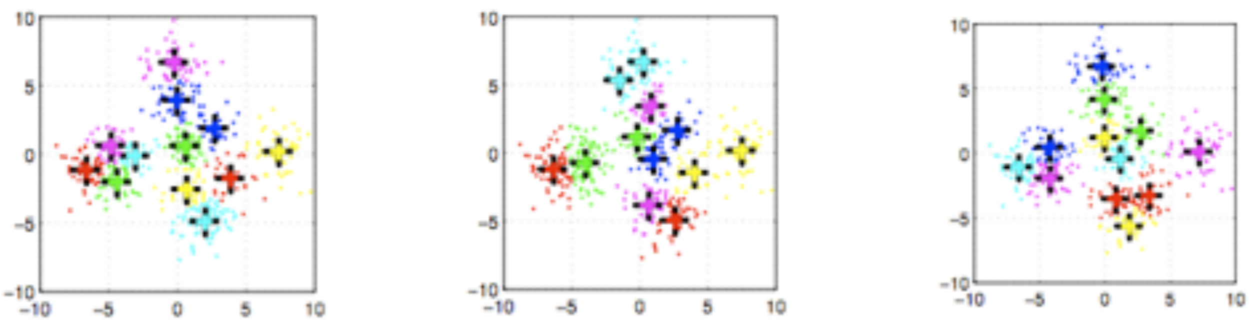
1. Pick random sample points as cluster prototypes.
2. Assign cluster labels $\{y_n\}_1^N$ to samples $\{\mathbf{x}_n\}_1^N$ according to prototype distances $d_k^2 = ||\mathbf{x}_n - \mathbf{p}_k||^2$
3. Assign prototypes as averages of samples within cluster: $$\mathbf{p}_k = \frac{1}{|\{y_n = k\}|} \sum_{n=1}^{N} \delta[y_n = k]\mathbf{x}_n$$
4. Repeat 2-3 until labels stop changing.

# K-means clustering

- K-means finds a *local min* of the cost:

$$J(\mathbf{p}_1, \ldots \mathbf{p}_K) = \sum_{k=1}^{K} \sum_{n=1}^{N} \delta[y_n = k] ||\mathbf{x}_n - \mathbf{p}_k||^2$$

- Issue 1:Bad repeatability:



- Issue 2:What is the value of K?

# Fuzzy K-means clustering

- Fix (partial) for repeatability:
- Replace binary indicator function $\delta[y_n = k]$ with a continuous weight, $w_{kn}$, for each sample.

$$J(\mathbf{p}_1, \ldots \mathbf{p}_K) = \sum_{k=1}^{K} \sum_{n=1}^{N} w_{kn} ||\mathbf{x}_n - \mathbf{p}_k||^2$$

- Smoother cost fcn $\Rightarrow$ fewer local min.
- Called *fuzzy k-means* or *fuzzy c-means*.

# Fuzzy K-means clustering

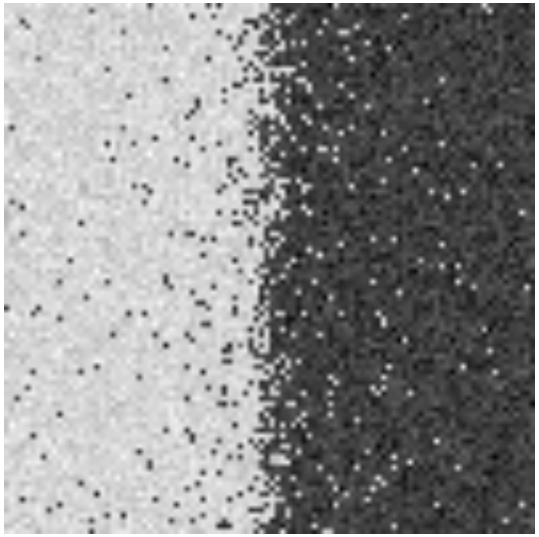
1. Pick random sample points as cluster prototypes.
2. Assign weights, $w_{kn}$, to samples $\{\mathbf{x}_n\}_1^N$ according to $w_{kn} = 1/(||\mathbf{x}_n - \mathbf{p}_k||^2 + \epsilon)$
3. Assign prototypes as weighted averages of samples: $$\mathbf{p}_k = \frac{1}{\sum_{n=1}^N w_{kn}} \sum_{n=1}^N w_{kn}\mathbf{x}_n$$
4. Repeat 2-3 until labels stop changing.

# K-means problems

- Fix for the local min problem:
  - Run the algorithm many times, and pick the solution with the lowest $J$.
- Steps 2,3 can be seen as special cases of the EM-algorithm [Dempster et al. 77]
- more on this soon.
- First we need to introduce *mixture models*.

# Mixture models

- A *generative model* for data that may come from several distributions.
- E.g. pixel values at a step edge with uncertain location:



# Mixture models

- We model the probability density of pixel intensity *I* as:

$$p(I) = \sum_{k=1}^{K} p(I|\Gamma_k)P(\Gamma_k)$$

# Mixture models

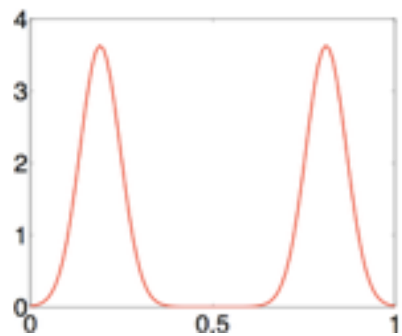
- We model the probability density of pixel intensity *I* as:

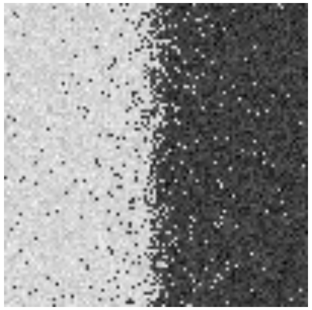
$$p(I) = \sum_{k=1}^{K} p(I|\Gamma_k)P(\Gamma_k)$$

- *Mixture probabilities*:

$$\sum_{k=1}^{K} P(\Gamma_k) = 1$$

e.g. $P(\Gamma_1)=P(\Gamma_2)=0.5$ gives this $p(I)$:



# Mixture models

- We model the probability density of pixel intensity *I* as:

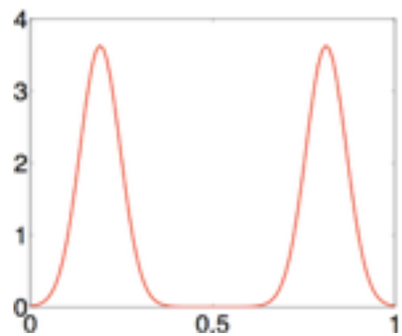
$$p(I) = \sum_{k=1}^{K} p(I|\Gamma_k)P(\Gamma_k)$$

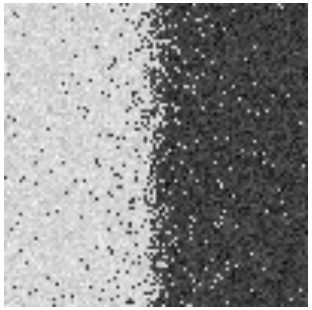
- *Mixture components*: e.g.

$$p(I|\Gamma_k)$$

$$p(I|\Gamma_k) = \frac{1}{\sqrt{2\pi}\sigma_k} e^{-0.5(I-\mu_k)^2/\sigma_k^2}$$

- Gaussian mixture model





# Mixture models

- Gaussian mixture components:

$$p(I|\Gamma_k) = \frac{1}{\sqrt{2\pi}\sigma_k} e^{-0.5(I-\mu_k)^2/\sigma_k^2}$$

- Notation conditioned on the parameters:

$$p(I|\mu_k, \sigma_k) = \frac{1}{\sqrt{2\pi}\sigma_k} e^{-0.5(I-\mu_k)^2/\sigma_k^2}$$

- Also the mixture probabilities are parameters:

$$P(\Gamma_k) = \pi_k \,, \text{ where } \sum_k \pi_k = 1$$

# Expectation Maximisation

- Given a set of measurements, $\{I_n\}_1^N$ how do we estimate the parameters of the mixture distribution *p(I)*?

$$p(I) = \sum_{k=1}^{K} p(I|\Gamma_k)P(\Gamma_k)$$

# Expectation Maximisation

- Given a set of measurements, $\{I_n\}_1^N$ how do we estimate the parameters of the mixture distribution *p(I)*?

$$p(I|\{\pi_k, \mu_k, \sigma_k\}_1^K) = \sum_{k=1}^{K} \pi_k p(I|\mu_k, \sigma_k)$$

- This can be done with the EM algorithm.
- Note similarities with K-means below.

# Expectation Maximisation

1. Postulate a mixture distribution.

2. E: Compute partial memberships, $w_{kn}$, with $\sum_{k=1}^{K} w_{kn} = 1$ to samples $\{I_n\}_1^N$, using the mixture distribution.
3. M: Use partial memberships to estimate mixture distribution parameters.

4. Repeat 2-3 until convergence.

# Expectation Maximisation

- For the mixture:

$$p(I|\{\pi_k, \mu_k, \sigma_k\}_1^K) = \sum_{k=1}^{K} \pi_k p(I|\mu_k, \sigma_k)$$

- The E-step becomes:

$$\tilde{w}_{kn} = \pi_k p(I_n|\mu_k, \sigma_k)$$

$$w_{kn} = \tilde{w}_{kn} / \sum_{l=1}^{K} \tilde{w}_{ln}$$

# Expectation Maximisation

- For the mixture:

$$p(I|\{\pi_k, \mu_k, \sigma_k\}_1^K) = \sum_{k=1}^{K} \pi_k p(I|\mu_k, \sigma_k)$$

- The E-step becomes:

$$\tilde{w}_{kn} = \pi_k p(I_n|\mu_k, \sigma_k)$$

$$w_{kn} = \tilde{w}_{kn} / \sum_{l=1}^{K} \tilde{w}_{ln}$$

- What is $p(I_n|\mu_k, \sigma_k)$?

# Expectation Maximisation

- The M-step becomes:

$$\pi_k = P(\Gamma_k) = \frac{1}{N} \sum_{n=1}^{N} w_{kn}$$

- and, assuming a Gaussian mixture:

$$\mu_k = \frac{1}{\sum_{n=1}^{N} w_{kn}} \sum_{n=1}^{N} w_{kn} I_n$$

$$\sigma_k^2 = \frac{1}{\sum_{n=1}^{N} w_{kn}} \sum_{n=1}^{N} w_{kn} (I_n - \mu_k)^2$$

# Expectation Maximisation

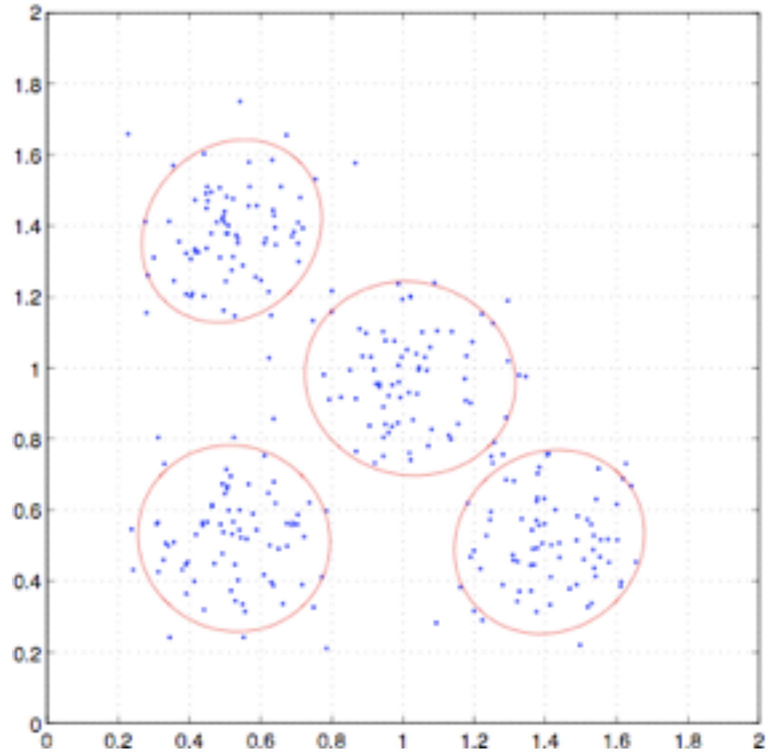
- Generalizes to higher dimensions.
- e.g. in 2D we have 5 parameters in each mixture component:

$$\mu = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix} \quad \boldsymbol{\Sigma} = \begin{pmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{12} & \sigma_{22} \end{pmatrix}$$

- Just like K-means,
  EM also finds a local min.

# Expectation Maximisation

- Demo for 2D case:

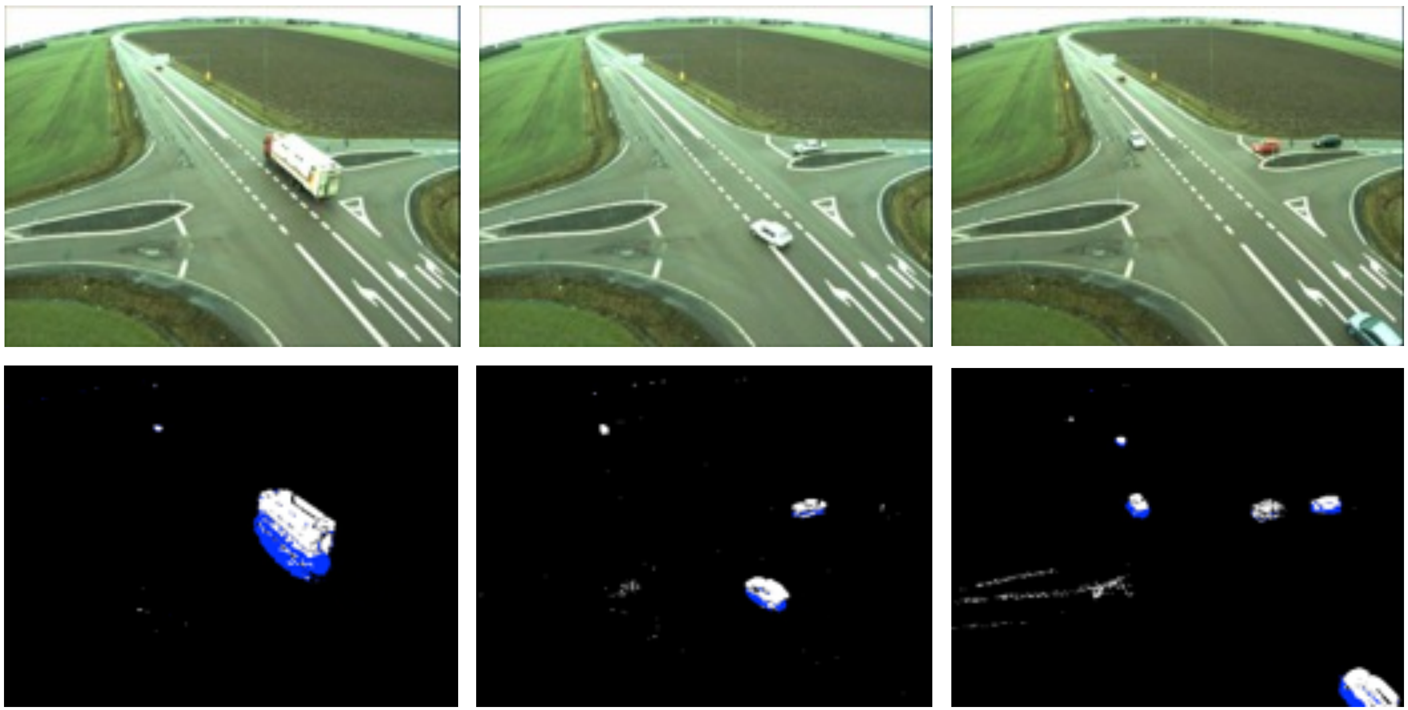


# Background modelling

- A popular application of mixture models is **background modelling** (SHB 16.5.1):
  - Estimate a mixture model for the image *in each pixel.*
  - Pixel values far from the mixture are seen as foreground pixels.
  - Popular way track e.g. people and cars in **stationary** surveillance cameras.
  - Fast compared to motion estimation.

# Background modelling

- Background modelling+shadow detection



- CVL Master thesis of John Wood 2007

# Background modelling

- Samples now arrive one at a time.
- EM uses a batch update:

$$\mu_k = \frac{1}{\sum_{n=1}^{N} w_{kn}} \sum_{n=1}^{N} w_{kn} I_n$$

$$\sigma_k^2 = \frac{1}{\sum_{n=1}^{N} w_{kn}} \sum_{n=1}^{N} w_{kn} (I_n - \mu_k)^2$$

- On-line update is needed

# Background modelling

- Samples now arrive one at a time.
- On-line update:

$$\mu_k[n] = (1-\alpha)\mu_k[n-1] + \alpha I_n$$

$$\sigma_k^2[n] = (1-\alpha)\sigma_k^2[n-1] + \alpha(I_n - \mu_k[n-1])^2$$

$$\pi_k[n] = (1-\alpha)\pi_k[n-1] + \alpha w_{kn}$$

- How to design $\alpha(w_{kn}, \pi_k)$ can be investigated in project 1.

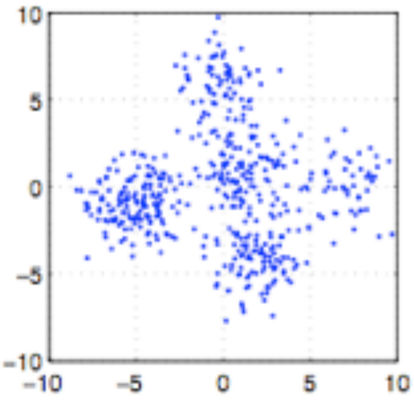
# Mean-shift Clustering

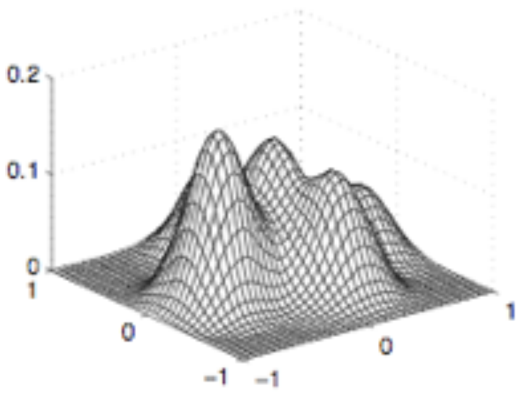
- A proper solution to the local min problem is to find *all* local minima.
- Two steps:
  - Mean-shift filter (mode seeking)
  - Clustering

# Kernel density estimate

- For a set of sample points $\{\mathbf{x}_n\}_1^N$ we define a continuous PDF-estimate as:

$$p(\mathbf{x}) = \frac{1}{Nh^d} \sum_{n=1}^{N} K\left(\frac{\mathbf{x}_n - \mathbf{x}}{h}\right)$$



# Kernel density estimate

- For a set of sample points $\{\mathbf{x}_n\}_1^N$ we define a continuous PDF-estimate as:

$$p(\mathbf{x}) = \frac{1}{Nh^d} \sum_{n=1}^{N} K\left(\frac{\mathbf{x}_n - \mathbf{x}}{h}\right)$$

- K() is a kernel, e.g. $K(\mathbf{x}) = c \exp\left(-\mathbf{x}^T\mathbf{x}/2\right)$
- h is the kernel scale.

# Mode seeking

- By *modes* of a PDF, we mean the local peaks of the kernel density estimate.
  - These can be found by gradient ascent, starting in each sample.
  - If we use the Epanechnikov kernel,

$$K_E(\mathbf{x}) = \begin{cases} c(1 - \mathbf{x}^T\mathbf{x}) & \text{if } \mathbf{x}^T\mathbf{x} \leq 1 \\ 0 & \text{otherwise.} \end{cases}$$

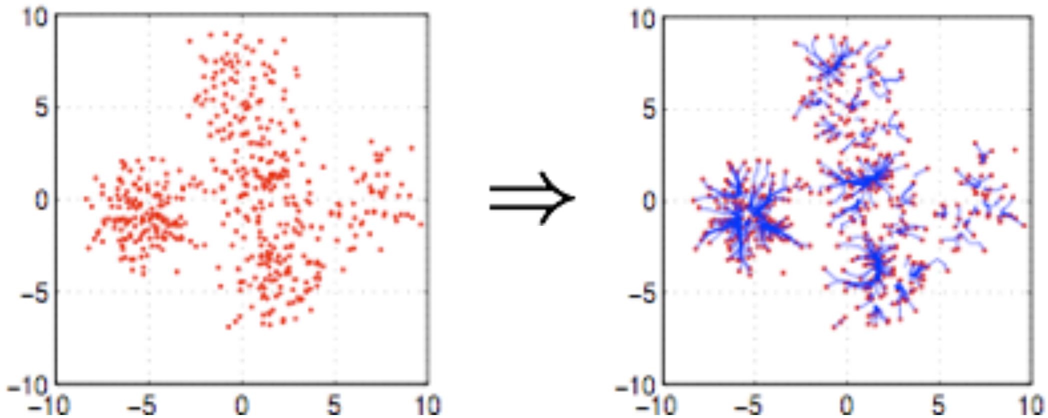
    a particularly simple gradient ascent is possible.

# Mean-shift filtering

- Start in each data point, $\mathbf{m}_n = \mathbf{x}_n$
- Move to position of local average

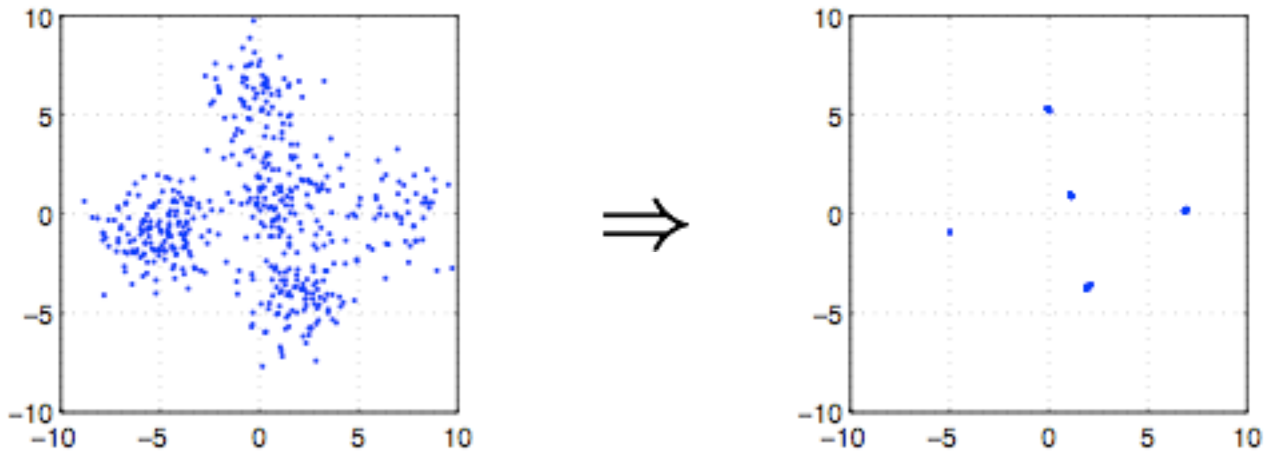
$$\mathbf{m}_n \leftarrow \mathrm{mean}_{\mathbf{x}_n \in S(\mathbf{m}_n)}(\mathbf{x}_n)$$

- Repeat step 2 until convergence.



# Mean-shift clustering

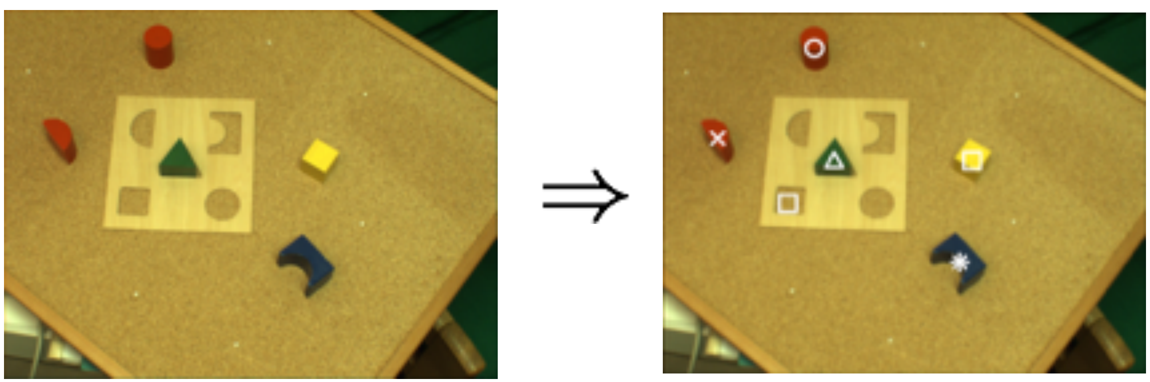
- After convergence of the mean-shift filter, all points within a certain distance (e.g. *h*) are said to constitute one cluster.



# Pose estimation

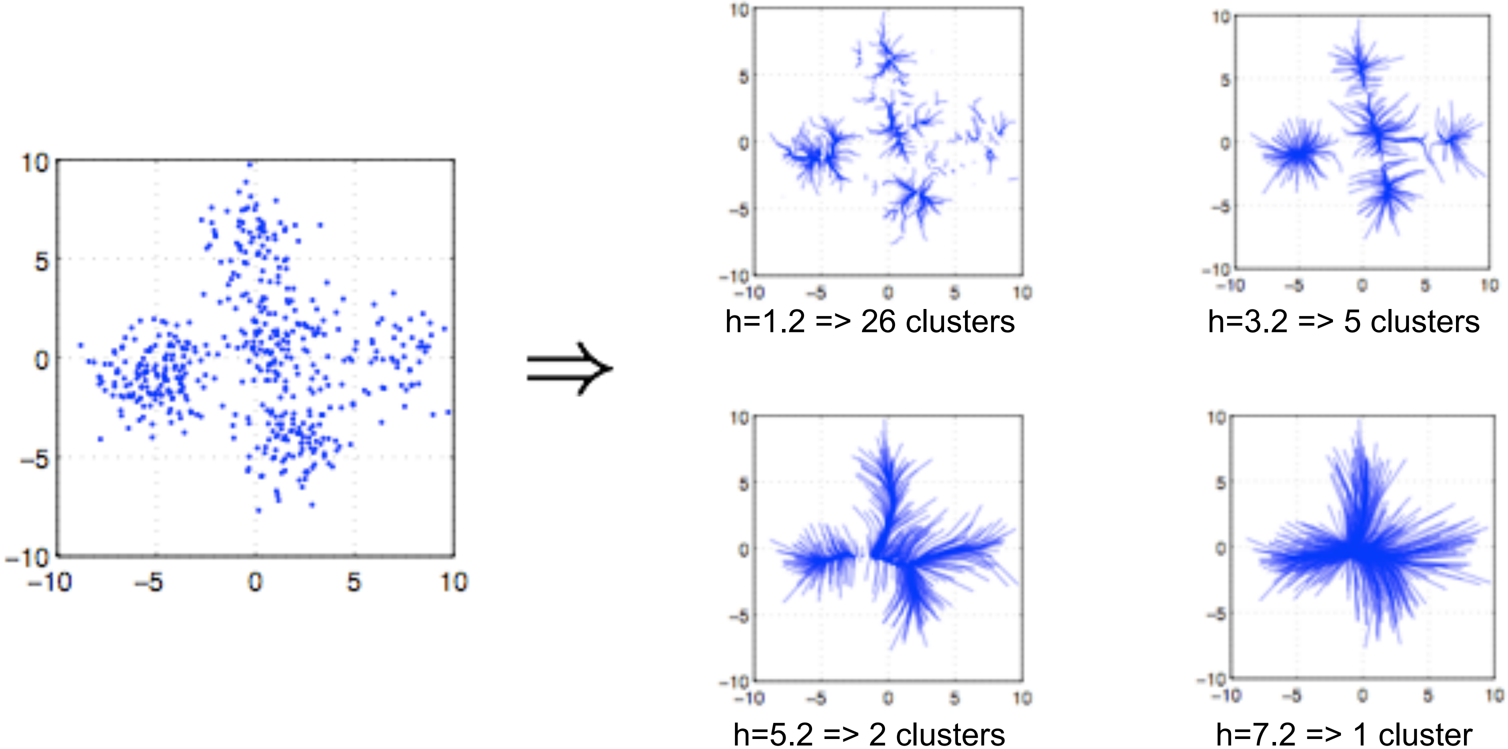
- Mean-shift can be used for "continuous voting" in pose estimation.
- Each local invariant feature (e.g. SIFT or MSER) will cast a vote (sample point)

$$\mathbf{x} = \begin{pmatrix} x_0 & y_0 & \alpha & s & \varphi & \theta & \mathtt{type} \end{pmatrix}^T$$



# Mean-shift

- Choice of kernel scale affects results



# Mean-shift

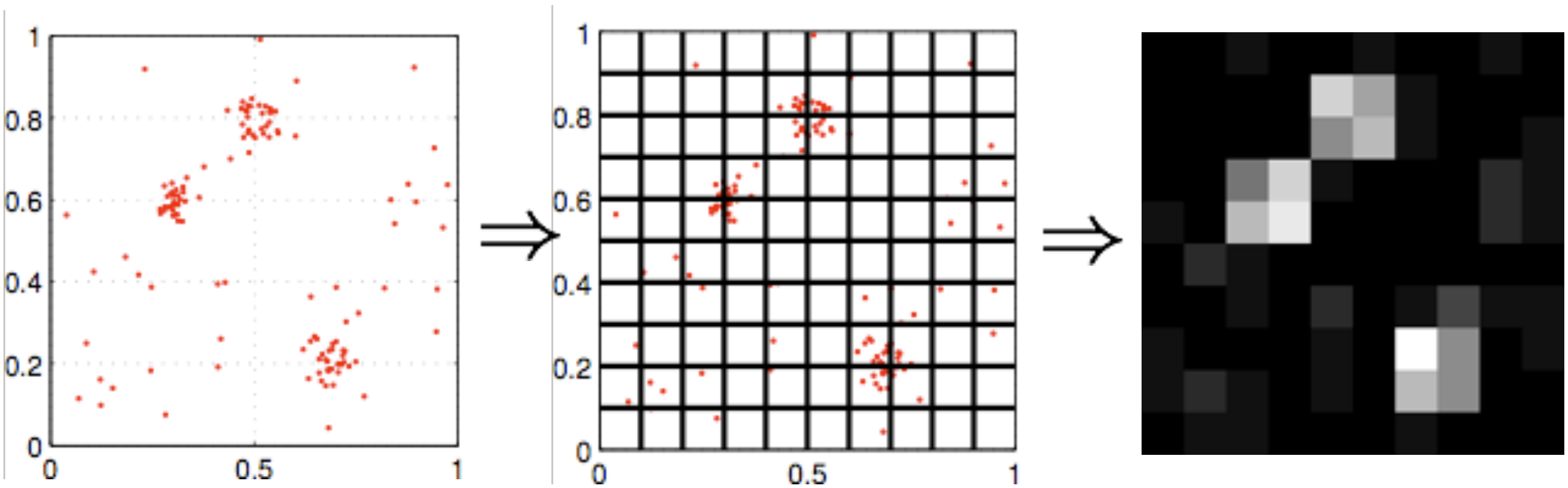
- For the Epanechnikov kernel, the algorithm is quite fast.
- The Gaussian kernel is another popular choice.
- There is also a scale adaptive version of meanshift, that works in a manner similar to EM in each iteration (slower).

# Generalised Hough Transform

- Another way to find modes of a PDF is to quantize the parameter space into accumulator cells.
- Each sample then casts a vote in one or several cells.
- This is called the *Generalised Hough Transform* (GHT).

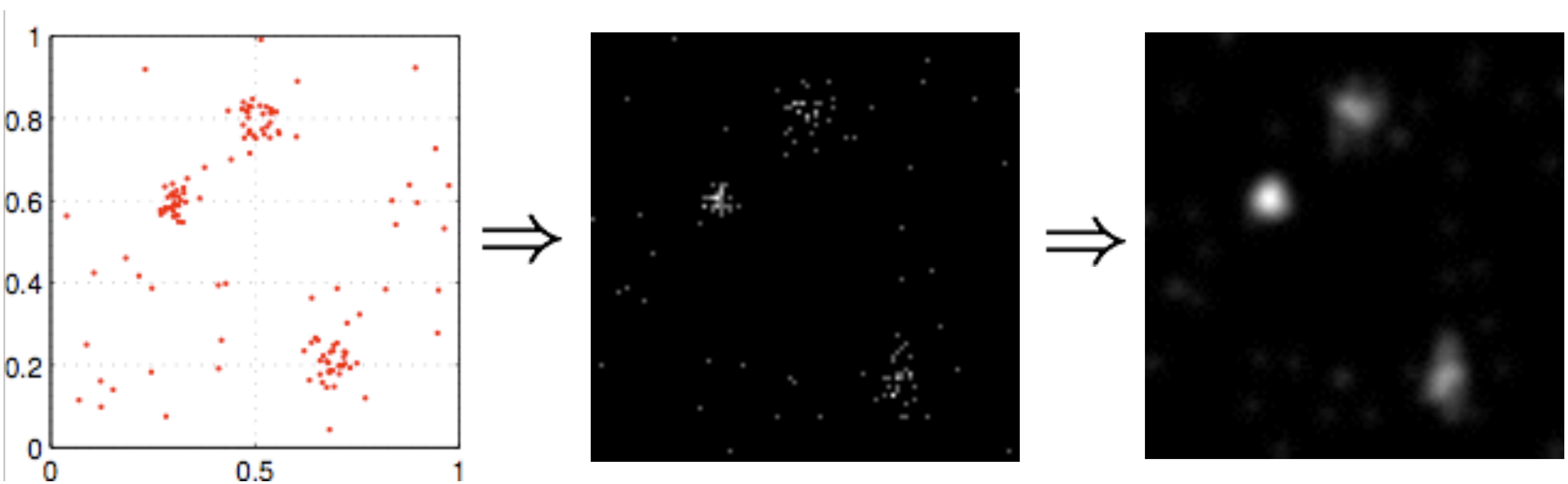
# Generalised Hough Transform

- Non-iterative ⇒ constant time complexity.



# Generalised Hough Transform

- Quantisation can be dealt with by increasing the number of cells, and blurring.

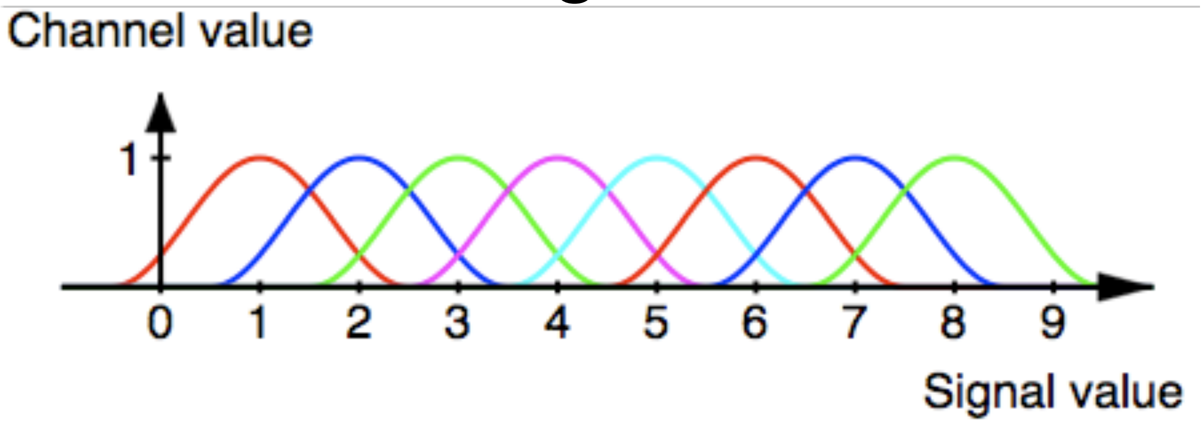


# Channel Representation

- A similar technique is to use averaging in *channel representation*.
  - By first quantizing, and then blurring, we are actually introducing aliasing of the PDF.
  - Better to directly sample the kernel density estimate at regularly sampled positions.
  - Density of samples is regulated by the kernel scale.

# Channel Representation

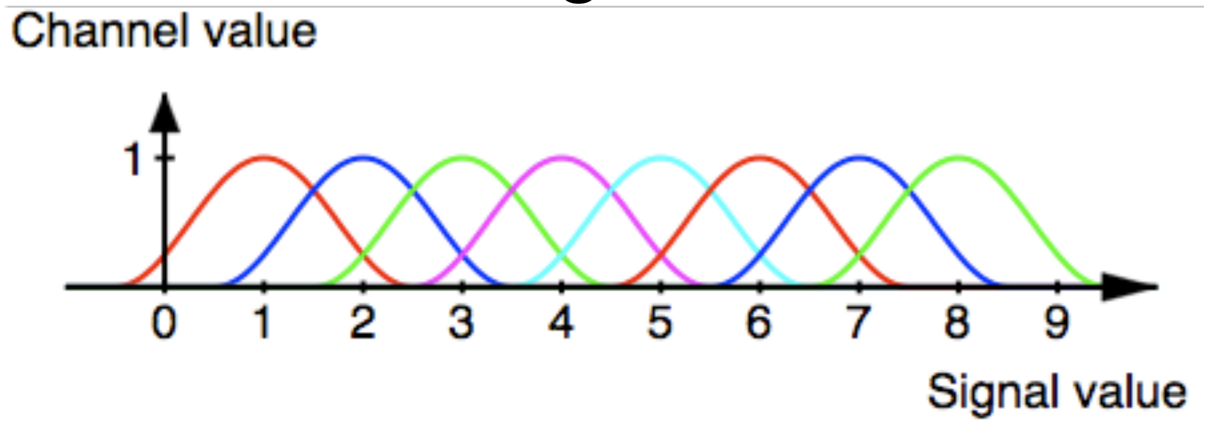
- Channel encoding




$$x = 4 \Rightarrow \text{enc}(x) = \mathbf{x} = \begin{bmatrix} B(x-1) & B(x-2) & \ldots & B(x-8) \end{bmatrix}^T$$

# Channel Representation

- Channel encoding



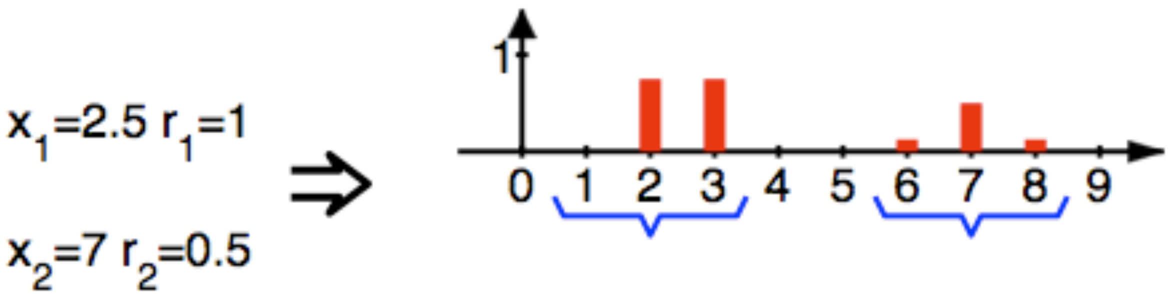

$$x = 4 \Rightarrow \mathrm{enc}(x) = \mathbf{x} = [0 \quad 0 \quad 0.25 \quad 1 \quad 0.25 \quad 0 \quad 0 \quad 0]^T$$

- Channel decoding

$$\hat{x} = \mathrm{dec}(\mathbf{x})$$

# Channel Representation

– A local decoding is necessary in order to decode a multi-valued channel representation.



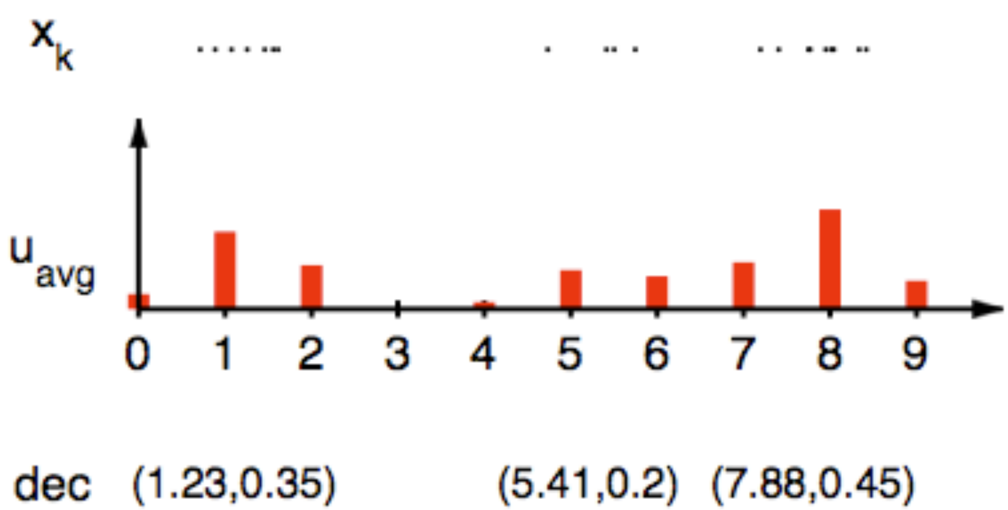

– That is

$$\hat{x}_1 = \mathrm{dec}(x_1 \ldots x_3) \qquad \hat{x}_2 = \mathrm{dec}(x_6 \ldots x_8)$$

– Decoding formula depends on the kernel.

# Channel Clustering

- Channel encode data points, $\mathbf{x}_n = \mathrm{enc}(x_n)$
- Average channel vectors $\bar{\mathbf{x}} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n$
- Compute all decodings $(\hat{x}, \hat{r})$



# Channel Clustering

- The decoding step computes *location*, *density*, and *standard deviation* at mode.
- Optimal decoding is expensive, but fast heuristic decodings exist.
- It can be shown [Forssén 04] that averaging in channel representation is equivalent to a regular sampling of a kernel density estimator.

# Summary

- This was a quick overview of clustering, and related techniques.
- The main purpose with learning is to make Computer Vision systems adapt to data.
- The alternative, to manually tune parameters, works for small static problems, but does not scale and cannot adapt to changes.

# Course events this week

- Thursday: Lab1
  Material on the course web page.
  Preparation is necessary to finish on time.

- Friday: Projects start
  Introductory lecture
  Assignments into groups (4/5 per group)