



# Image Analysis and Object Recognition

## Exercise 6

**Clustering using Expectation Maximization (EM)  
and  
Maximum-Likelihood (ML) image segmentation**

SS 2014

(Course notes for internal use only!)

- Exercise dates
  - 15.04.2014 → Introduction + Exercise 1
  - 29.04.2014 → Exercise 2
  - 13.05.2014 → Exercise 3
  - 27.05.2014 → Exercise 4
  - 10.06.2014 → Exercise 5
  - **24.06.2014 → Exercise 6**
  - 08.07.2014 → Final Meeting / Summary / Discussion
- Every 2<sup>nd</sup> Week, **17:00 – 18:30**

# Ex 5 - Descriptor Computation Algorithm

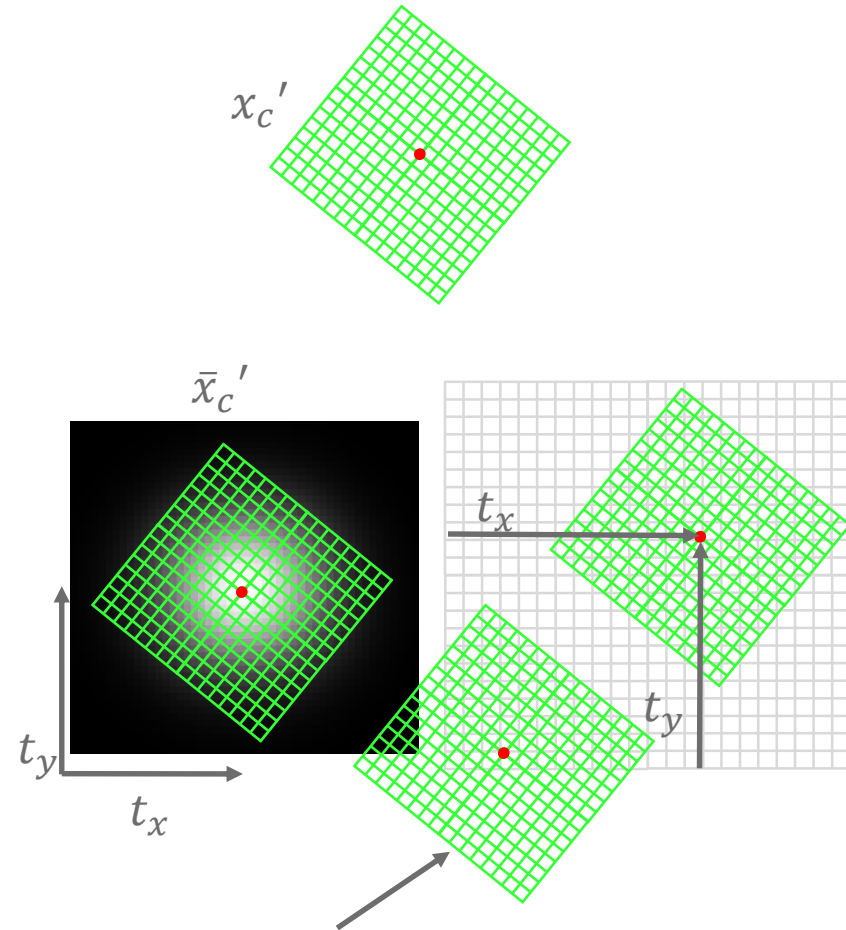
- compute the  $49 \times 49$  Gaussian filter ( $\sigma = 8$ )
- **for** all scales
  - **for** all points in current scale
    - Initialize arrays  $x_c, y_c$  with local image coordinates for a  $16 \times 16$  neighborhood
    - Rotate local coordinates  $x_c$  and  $y_c$  using main orientation  $\theta_m$
    - Determine nearest scale  $\sigma_n$
    - Sample Gaussian window, magnitude and angle image at  $\sigma_n \rightarrow G_s, M_s, \theta_s$
    - Compute weighted gradient magnitude  $M_w = G_s * M_s$
    - Rotate magnitude angles of  $\theta_s$  by  $\theta_{s,rot} = \theta_s - \theta_m$
    - Build the 16 histograms with 8 bins (each bin covers  $360/8 = 45$  degrees)
    - Build descriptor vector  $\rightarrow$  rearrange values of Histograms (128 bins) to a vector
    - Normalize the vector
  - **end**
- **end**

# Nearest Scale $\sigma_n$

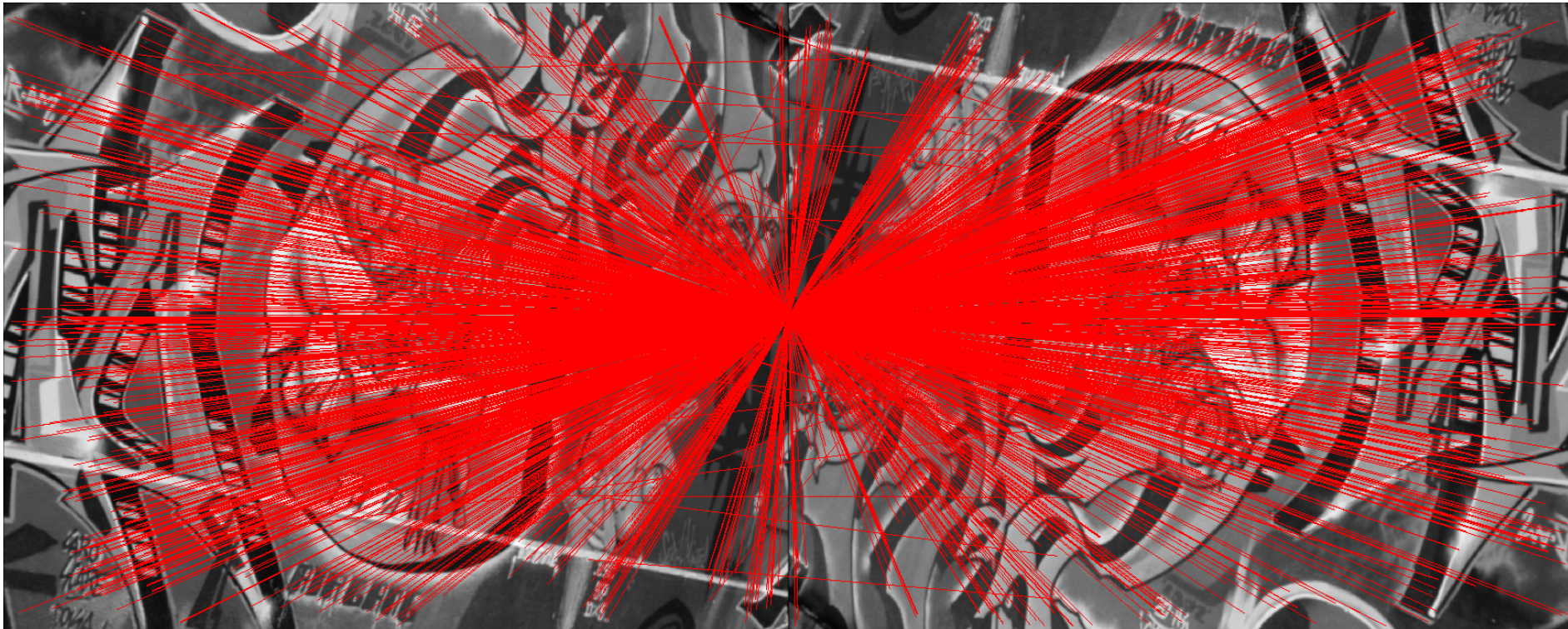
- **Nearest scale**  $\sigma_n$  of extrema  $r$ : Already available
  - Get keypoints in current scale  $\sigma$  :  $ex = Extrema\{\sigma\}$
  - Get single keypoint  $r$  in current scale:  $ex(r, :) = [y, x, \sigma_n, \theta_m, m_m]$
- Initial scale where a keypoint was found and stored may be **different** from the scale after its refinement (Step 3: Sub-pixel and scale localization)
- **Example:**  $Sigmas = [2.0, 2.5, 3.2, 4.0]$ 
  - Current scale in loop:  $\sigma = 2.0$
  - Current extremum  $r$  in scale  $\sigma = 2.0$  :  $ex(r, :) = [239.5, 369.4, 4.0, 14.3, 0.1]$ 
    - $\sigma_n = 4.0$  → Sample angles and magnitudes from that scale
- But for most keypoints:  $\sigma_n =$  current scale  $\sigma$  in loop

# Sampling using $x_c'$ and $y_c'$

- Translate  $x_c'$  and  $y_c'$  by  $t_x$  and  $t_y$ 
  - $\bar{x}_c' = \text{round}(x_c' + t_x)$ ;
  - $\bar{y}_c' = \text{round}(y_c' + t_y)$ ;
- Gaussian:  $t_x = 49/2$ ,  $t_y = 49/2$
- Angles / Magnitudes:  $t_x = x$ ,  $t_y = y$ 
  - $ex(r, :) = [y, x, \sigma_n, \theta_m, m_m]$
- Use  $\bar{x}_c'$  and  $\bar{y}_c'$  as indices
  - $ind = \text{sub2ind}(\text{array\_size}, \bar{y}_c', \bar{x}_c')$ ;
  - $\theta_s = \text{AnglePyramid}\{\sigma\}(ind)$ ;
- **Caution:**  $\text{sub2ind}$  fails, if  $\bar{y}_c'$  or  $\bar{x}_c'$  exceed image dimensions!



# Results Ex 5





## Exercise 6

**A:** Expectation Maximization (EM) Clustering

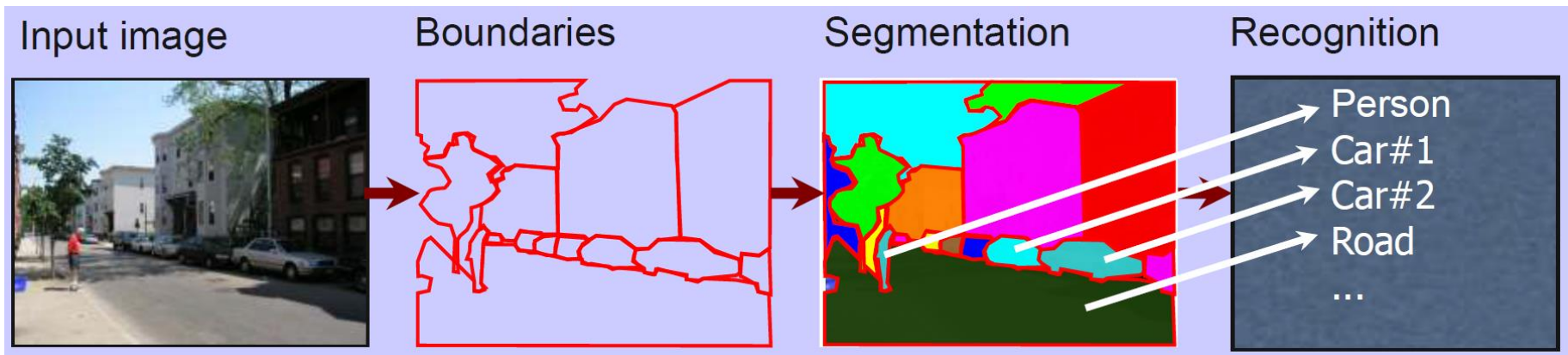
**B:** Maximum Likelihood (ML) image segmentation  
(**Application Task**)



# Motivation

## Segmentation

- Partitioning an image into a collection of connected set of pixels

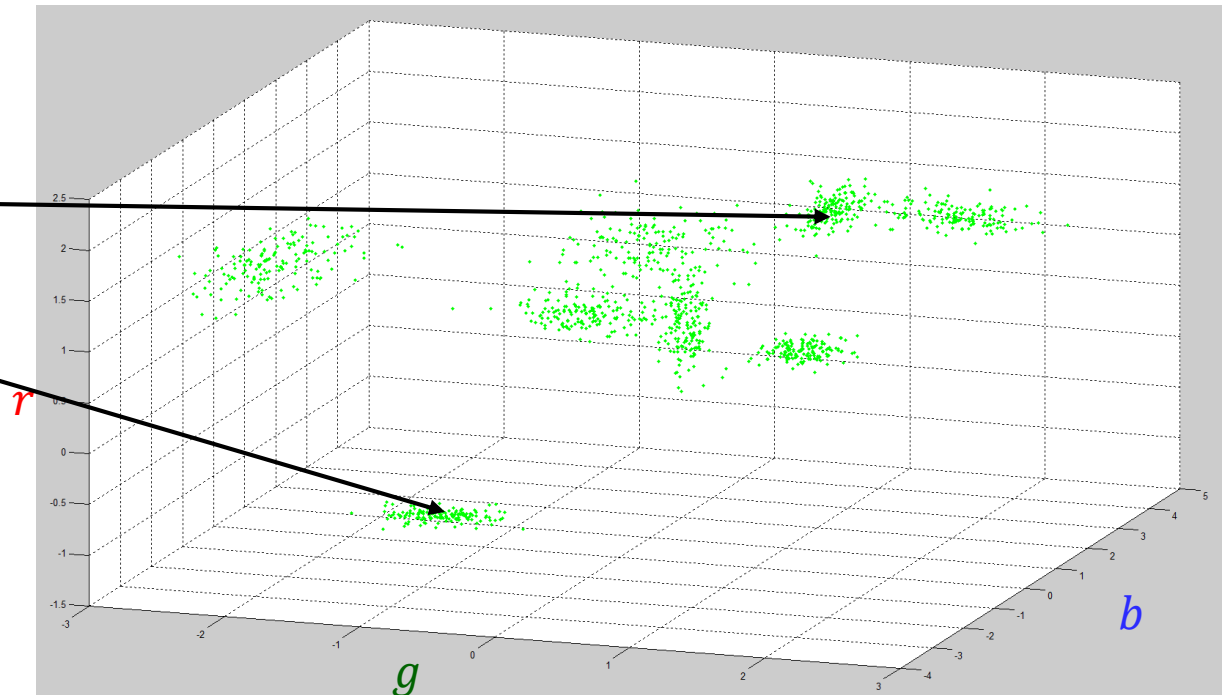
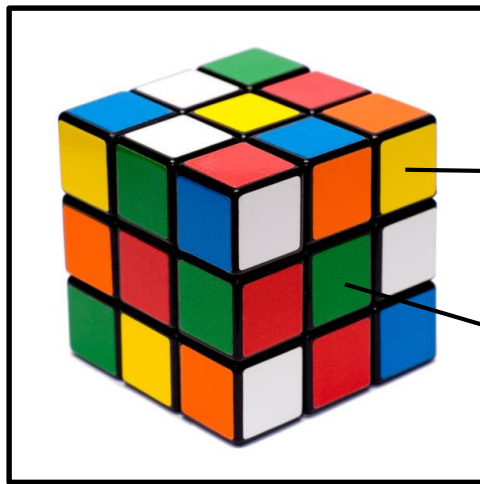


- Needed for understanding the content of a scene



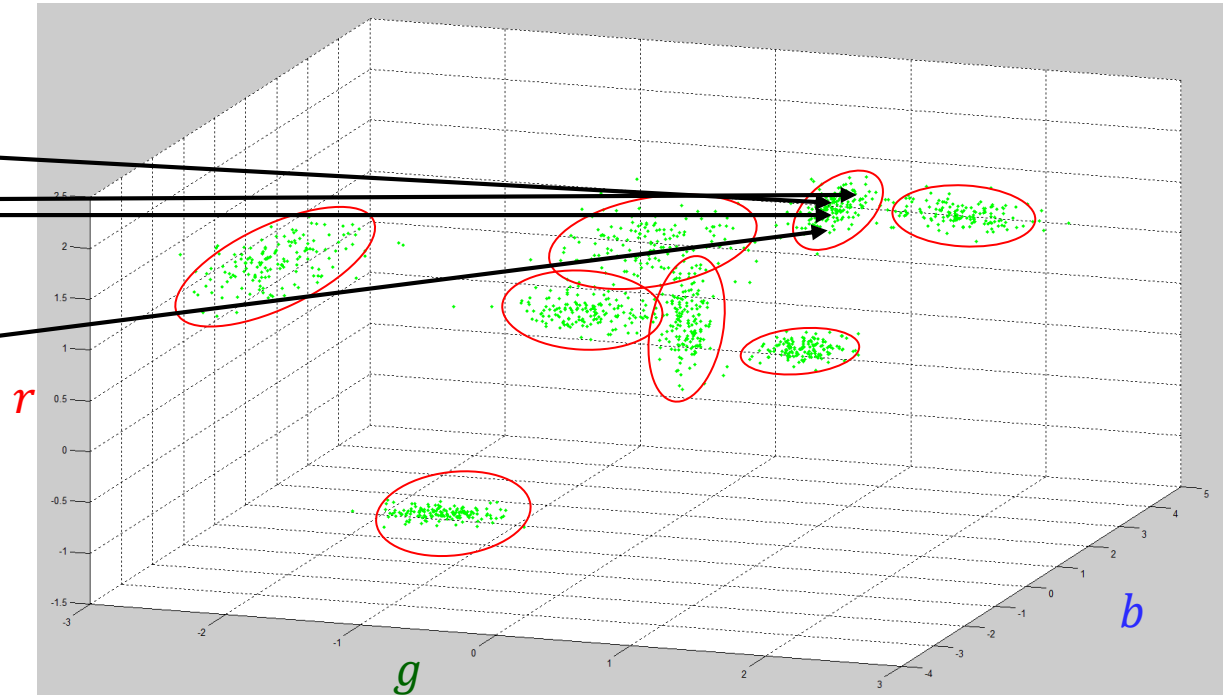
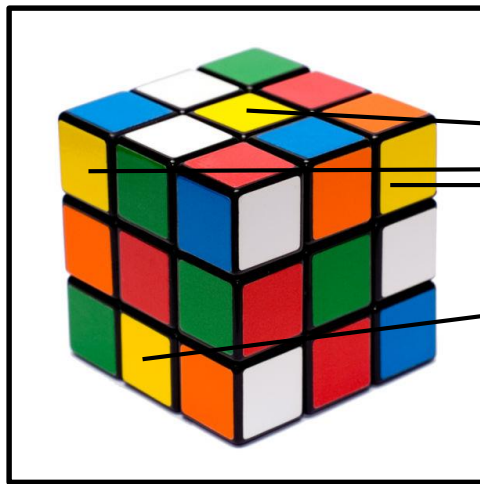
# Feature Spaces

- Given: 3-channel image (rgb)
  - Each channel represents one dimension of a feature space
  - Each pixel of the image maps to a point in that space



# Idea of Clustering

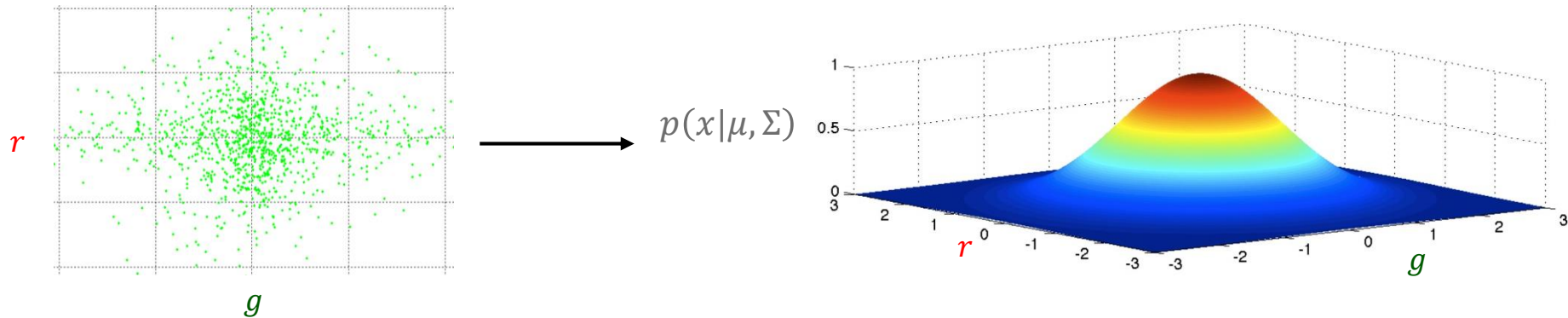
- Group all similar points in feature space to „clusters“
- Each cluster contains pixels with similar spectral properties
  - Members of a cluster belong to the same segment or segment class



# Parameterization of Data

- Cluster representation using multivariate Gaussian functions

$$p(x|\mu, \Sigma) = \frac{1}{(2\pi)^{\frac{d}{2}} \cdot |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x - \mu)^T \Sigma^{-1}(x - \mu)\right)$$



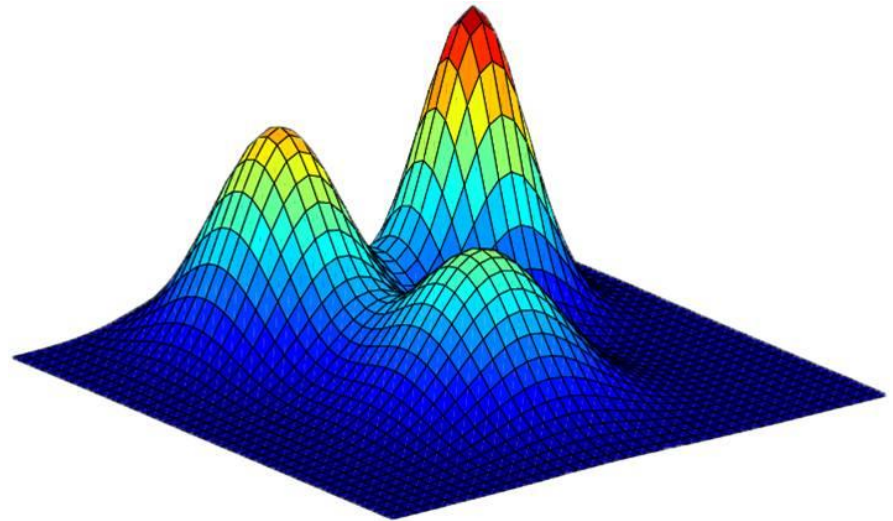
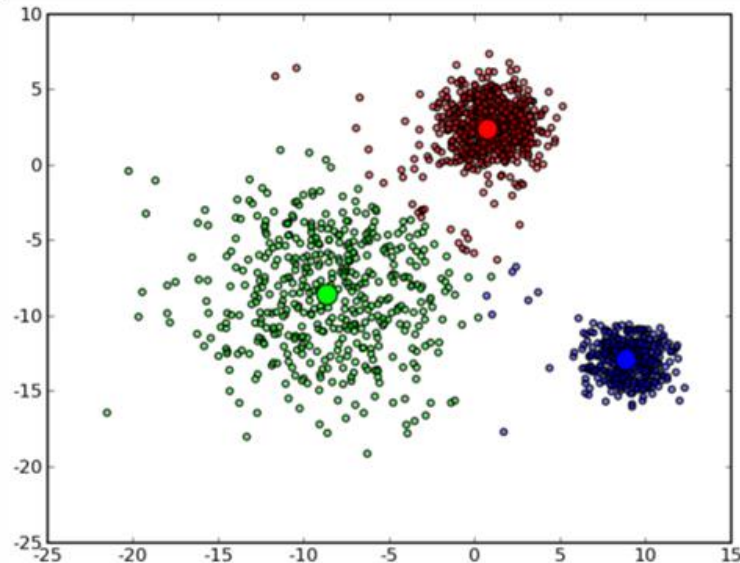
- $p(x|\mu, \Sigma)$ : „Probability of  $x$  given the parameters  $\mu, \Sigma$ “
- $x$ : Feature vector, e.g. rg-information of a pixel  $x = (r, g)$
- $\mu$ : **Center of the Gaussian function, here  $\mu = (0, 0)$**
- $\Sigma$ : **Covariance matrix (symmetric), here of size  $2 \times 2$  (2 dimensions)**
- $d$ : Number of feature-space dimensions, here  $d = 2$
- $|\Sigma|$ : Determinant of the covariance matrix
- $\Sigma^{-1}$ : Inverted covariance matrix

# Gaussian Mixture Models

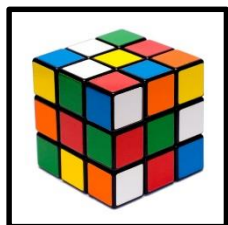
- Mixture-Model: Weighted sum of  $N_c$  elementary Gaussians

$$p(x|\Omega) = \sum_{c=1}^{N_c} \alpha_c p(x|\mu_c, \sigma_c), \quad \Omega = \{\alpha_1, \mu_1, \sigma_1, \dots, \alpha_{N_c}, \mu_{N_c}, \sigma_{N_c}\}$$

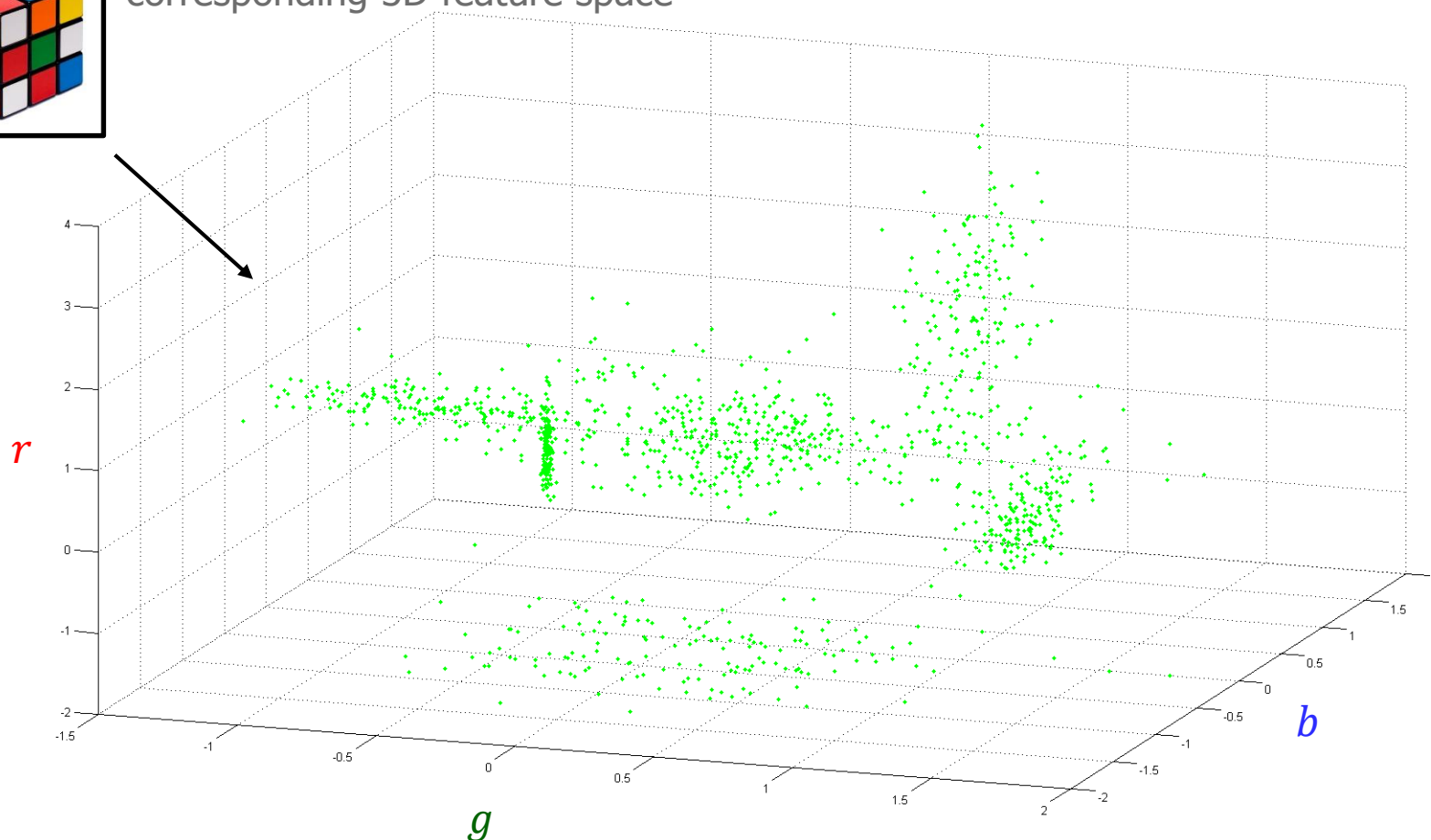
- $N_c = 3$  example in 2D:



# 3D Gaussian Mixture Model Example



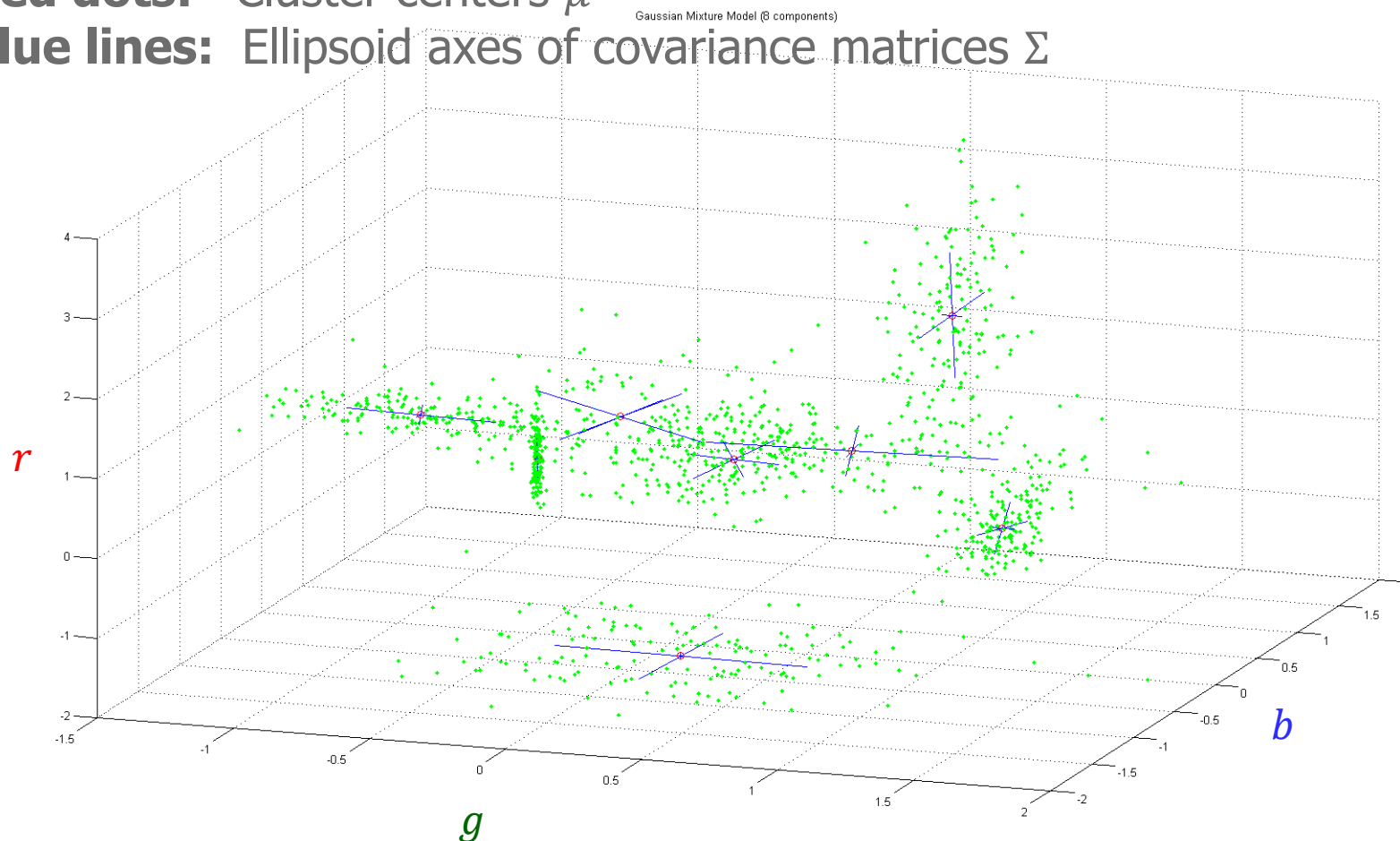
**Remember:** Each pixel of an rgb-image represents a point in the corresponding 3D feature space



# 3D Gaussian Mixture Model Example

**Red dots:** Cluster centers  $\mu$

**Blue lines:** Ellipsoid axes of covariance matrices  $\Sigma$





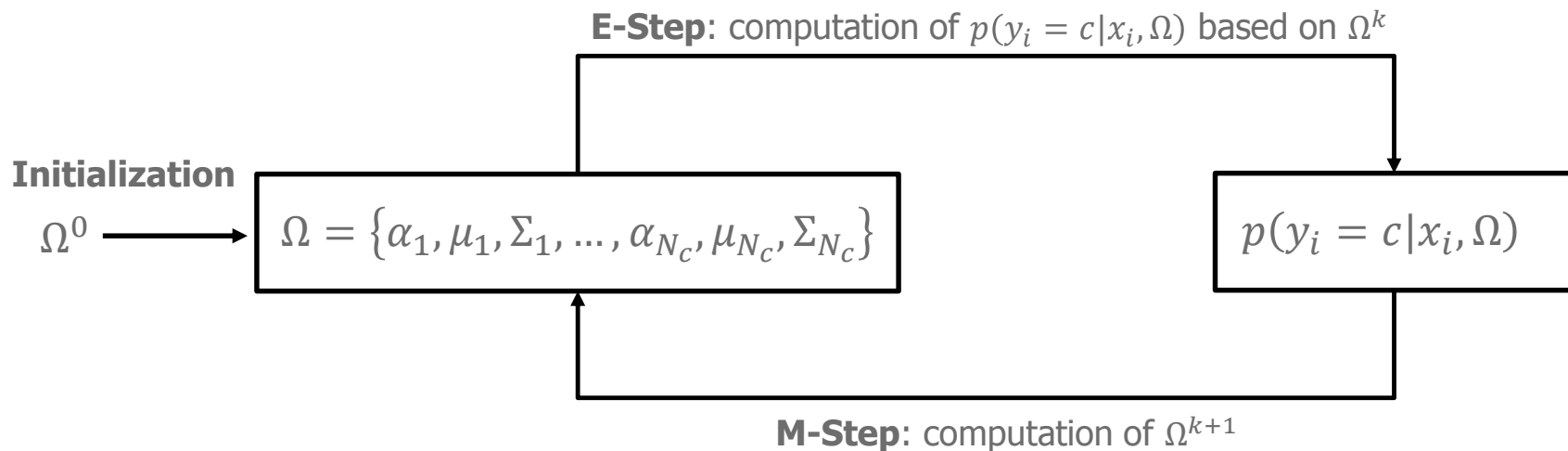
# Parameter Estimation for GMMs

- **Given:**  $d$ -dimensional feature space consisting of  $N_x$  feature vectors  $\{x_1, \dots, x_{N_x}\}$
- **Wanted:** Set of parameters  $\Omega = \{\alpha_1, \mu_1, \Sigma_1, \dots, \alpha_{N_c}, \mu_{N_c}, \Sigma_{N_c}\}$  for mixture model with  $N_c$  components
  - $\alpha_c$ : Weight for a cluster, where  $\sum_{c=1, \dots, N_c} \alpha_i = 1.0$
  - $\mu_c$ : Mean vector for a cluster
  - $\Sigma_c$ : Covariance matrix for a cluster
- **Approach:** Use probabilities
 
$$p(y_i = c | x_i, \Omega_c), \quad y_i \in \{1, \dots, N_c\},$$
 which describe the membership of a feature space point  $x_i$  to a cluster  $y_i$ 
  - This enables parameter estimation!!

# Parameter Estimation

## Expectation-Maximization (EM):

- Iterative Algorithm



- **E-Step:** Compute probabilities of all clusters  $c$  for each vector  $x$  using the current model parameters  $\Omega^k$
- **M-Step:** Computation of new model parameters  $\Omega_c^{k+1}$  using the probabilities  $p(y_i = c | x_i, \Omega_c^k)$  from E-Step

# E-Step

Compute membership probabilities for each vector  $x_i$  and component  $c$  using the current model parameters  $\Omega^k$

$$p(y_i = c | x_i, \Omega_c^k) = \frac{\alpha_c p(x_i | \mu_c^k, \Sigma_c^k)}{\sum_{j=1}^{N_c} \alpha_j p(x_i | \mu_j^k, \Sigma_j^k)}$$

„Prior probability“ (scalar value: weight)  $\rightarrow \alpha_c$

„Likelihood“ (Gaussian function)  $\rightarrow p(x_i | \mu_c^k, \Sigma_c^k)$

„Evidence“ (normalization)  $\rightarrow \sum_{j=1}^{N_c} \alpha_j p(x_i | \mu_j^k, \Sigma_j^k)$

- Bayes' theorem
  - „Probability of a cluster  $c$  given  $x_i$  and the parameters  $\Omega_c$ “
- Result: Matrix of size  $N_c \times N_x$ 
  - $N_c$ : Number of clusters
  - $N_x$ : Number of feature vectors

# Computations in E-Step 1/2

**Computation of:**

$$p(y_i = c | x_i, \Omega_c^k) = \frac{\alpha_c p(x_i | \mu_c^k, \Sigma_c^k)}{\sum_{j=1}^{N_c} \alpha_j p(x_i | \mu_j^k, \Sigma_j^k)}$$

$\longleftarrow$  numerator  
 $\longleftarrow$  denominator

**Problem:** Values  $p(x_i | \mu_c^k, \Sigma_c^k)$  often very small  $\rightarrow$  computational problems!

$\rightarrow$  Use log-values, and compute  $\log(p(y_i = c | x_i, \Omega_c^k))$

$\rightarrow$  Gaussian parametrization

$$p(x_i | \mu_c, \Sigma_c) = \frac{1}{(2\pi)^{\frac{d}{2}} \cdot |\Sigma_c|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x_i - \mu_c)^T \Sigma_c^{-1}(x_i - \mu_c)\right)$$

$\rightarrow$  The log-value for the numerator is:

$$\log(\alpha_c \cdot p(x_i | \mu_c^k, \Sigma_c^k)) = \log(\alpha_c) - \frac{1}{2} \left[ \log(|\Sigma_c^k|) + (x_i - \mu_c^k)^T \Sigma_c^{-1}(x_i - \mu_c^k) \right] + t$$

$\rightarrow$  Value  $t$  constant  $\rightarrow$  can be ignored in all computations

**Computation of:**  $\log(p(y_i = c | x_i, \Omega_c^k)) = \log\left(\frac{\alpha_c p(x_i | \mu_c^k, \Sigma_c^k)}{\sum_{j=1}^{N_c} \alpha_j p(x_i | \mu_j^k, \Sigma_j^k)}\right)$

$$= \log(\text{numerator}) - \log(\text{denominator})$$

• The log value for the numerator is:

- $$\log(\alpha_c \cdot p(x_i | \mu_c^k, \Sigma_c^k)) = \log(\alpha_c) - \frac{1}{2} [\log(|\Sigma_c^k|) + (x_i - \mu_c^k)^T \Sigma_c^{-1} (x_i - \mu_c^k)]$$

- Computation of denominator: no rule for sum of log-values ☹️
- Use numerator in the following way:

$$\begin{aligned} Num_{c,i} &= \log(\alpha_c \cdot p(x_i|\mu_c^k, \Sigma_c^k)) \\ Denom_i &= \log\left(\sum_{j=1}^{N_c} \alpha_j p(x_i|\mu_j^k, \Sigma_j^k)\right) = \log\left(\sum_{j=1}^{N_c} \exp(Num_{j,i})\right) \end{aligned}$$

- Final result:  $\log(p(y_i = c|x_i, \Omega_c^k)) = Num_{c,i} - Denom_i$

# M-Step

Compute new model parameters  $\Omega^{k+1}$  for each cluster  $c$  based on the probabilities  $p(y_i = c|x_i, \Omega_c^k)$  of E-Step (**no *log*-values!**):

$$N_p = \sum_{i=1}^{N_x} p(y_c = c|x_i, \Omega_c^k)$$

→ Number of feature points in cluster  $c$

$$\alpha_c = \frac{N_p}{N_x}$$

→ Weight for each cluster

$$\mu_c^{k+1} = \frac{1}{N_p} \sum_{i=1}^{N_x} x_i p(y_i = c|x_i, \Omega_c^k)$$

→ Mean vector (centroid) for each cluster

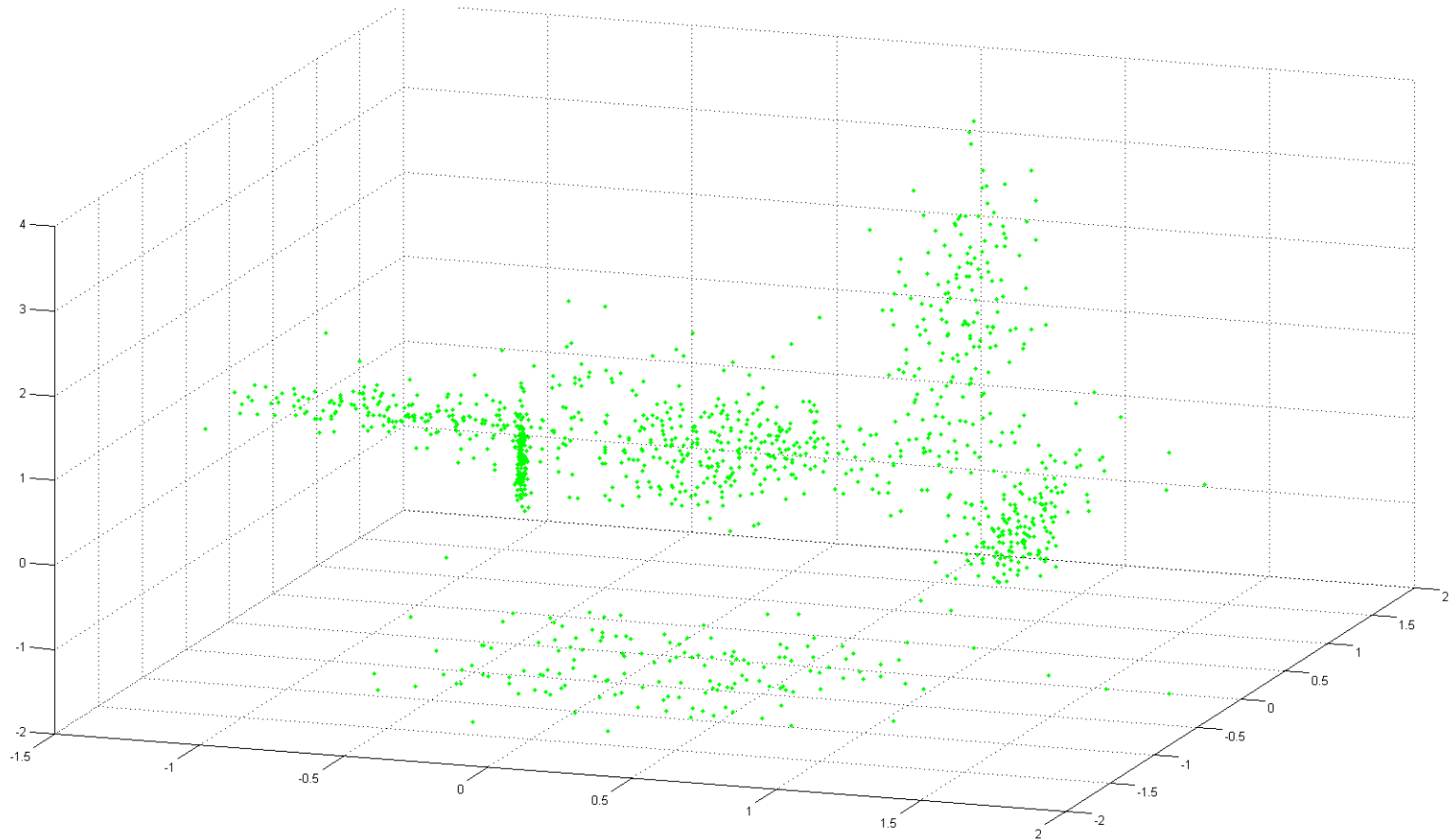
$$\Sigma_c^{k+1} = \frac{1}{N_p} \sum_{i=1}^{N_x} (x_i - \mu_c^{k+1})(x_i - \mu_c^{k+1})^T p(y_i = c|x_i, \Omega_c^k) \rightarrow \text{Covariance matrix for each cluster}$$



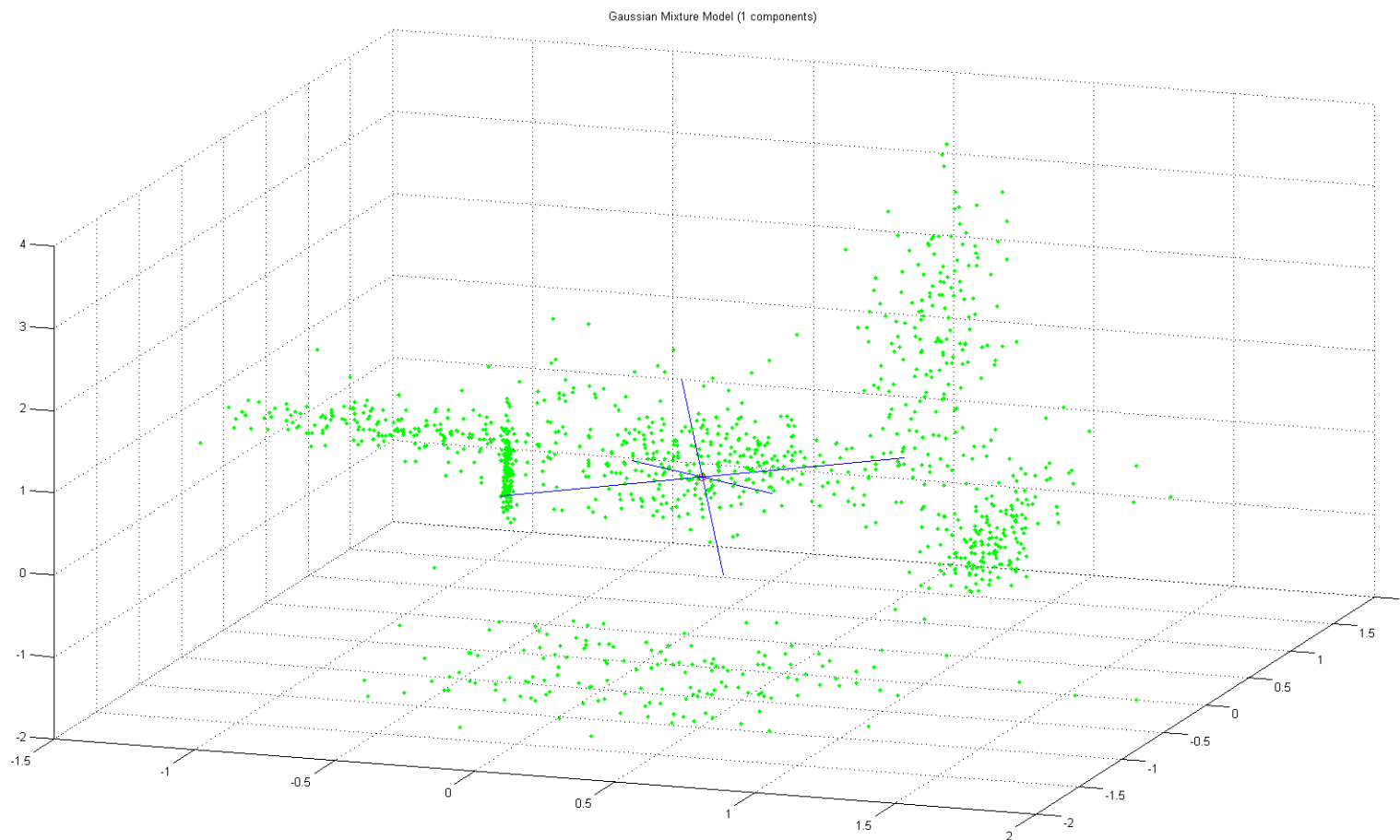
# Model Initialization

- EM derived using iterative gradient climbing
  - Converges to a **local** maximum
  - Quality of model depends on initialization
- Often applied: Random initialization
  - Estimation repeated several times
  - Problematic when number of components is large
- **Here:** Iterative splitting of components
  - Algorithm starts with one component (trivial)
  - After a local maximum is found a new component is added by splitting the „biggest“ cluster into two new ones
  - Repeated, till the desired number of components is reached

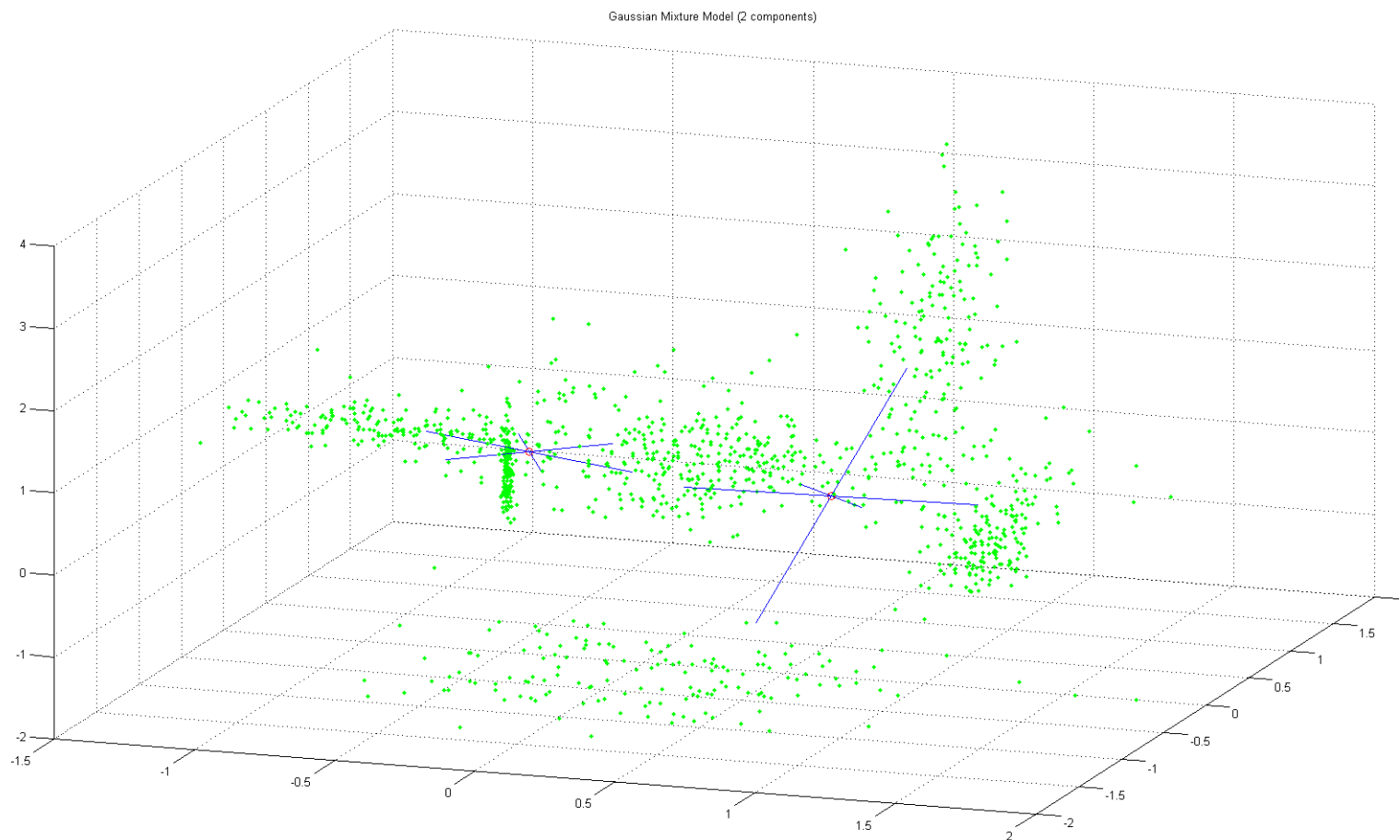
# Model Initialization: Feature Space



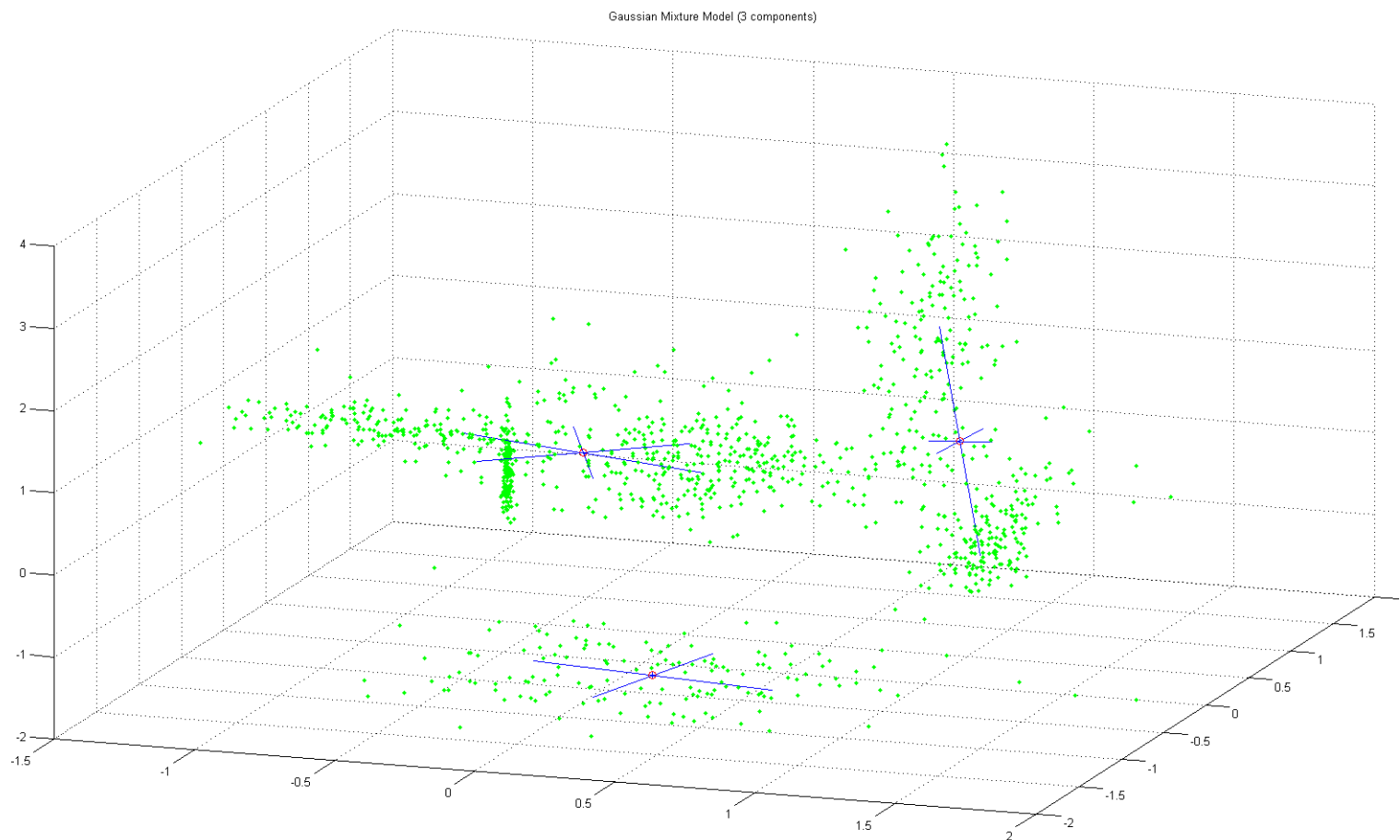
# Result for 1 Cluster



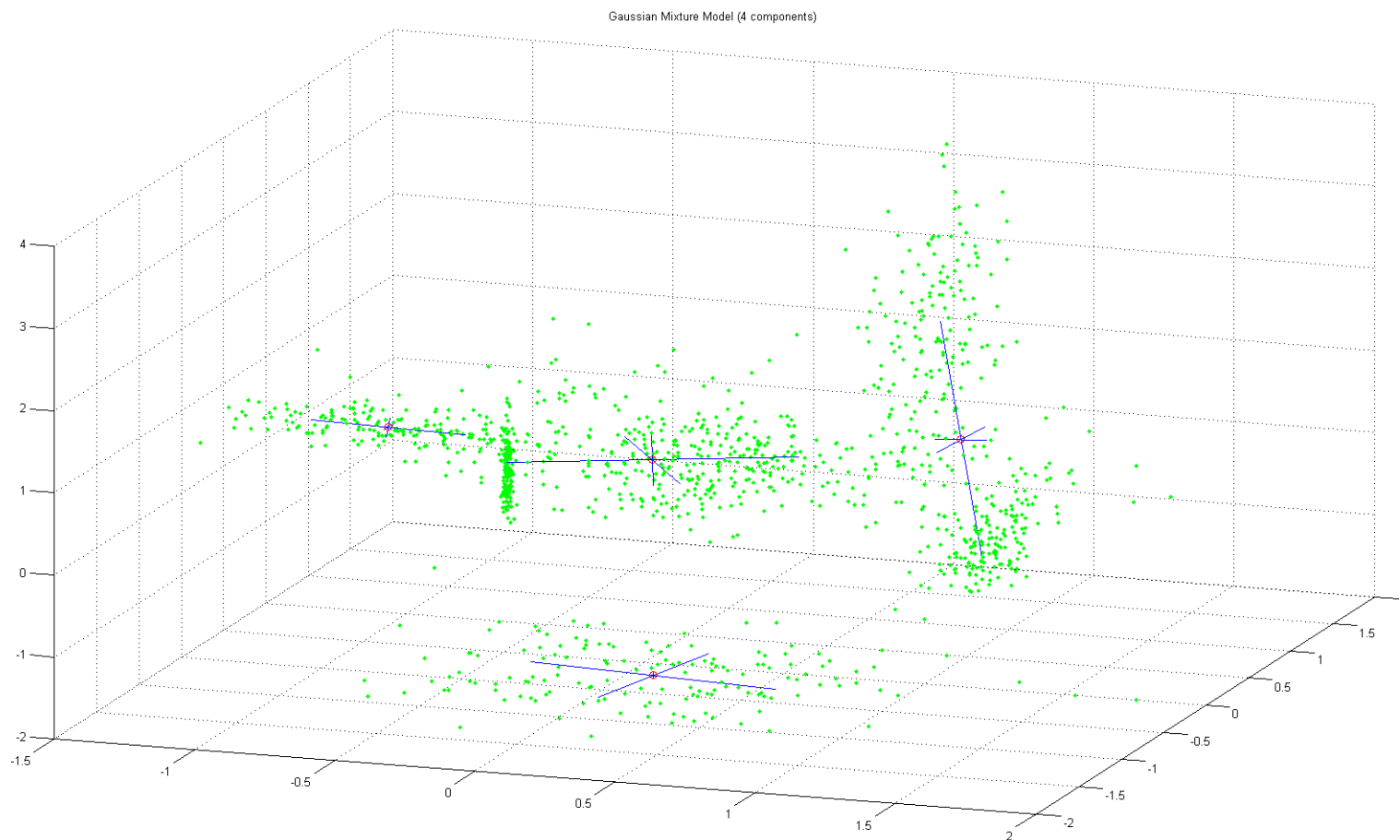
# Result for 2 Cluster



# Result for 3 Cluster

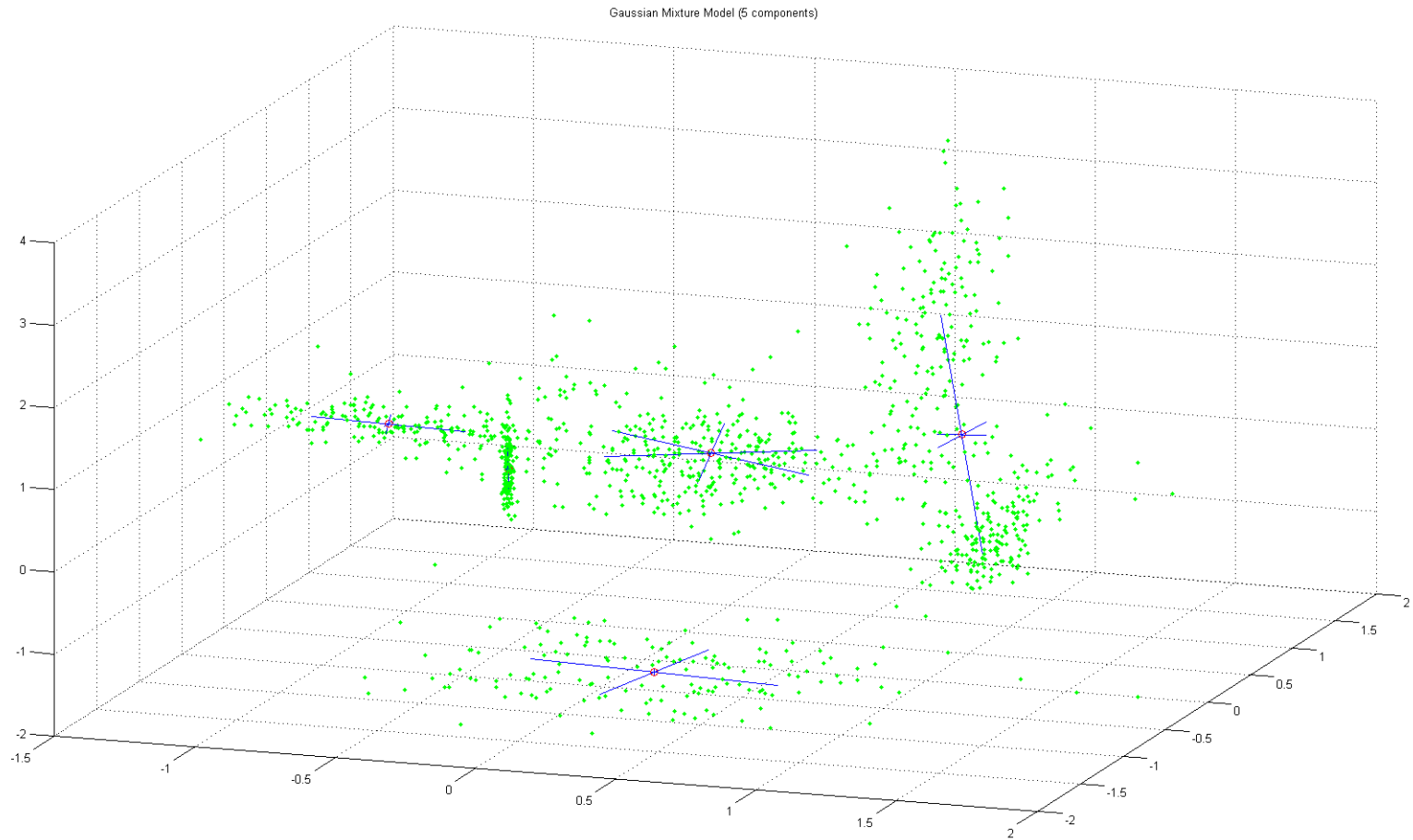


# Result for 4 Cluster

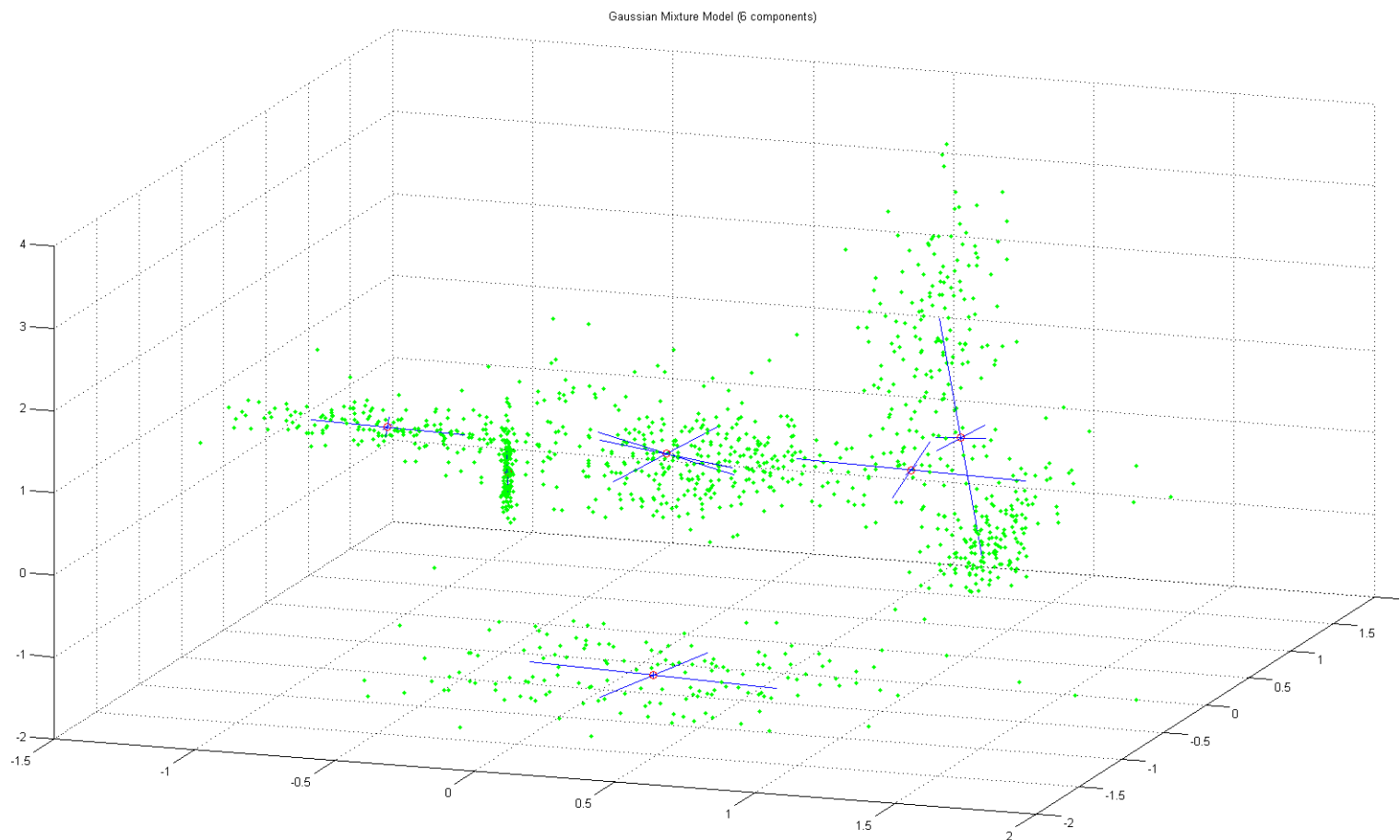




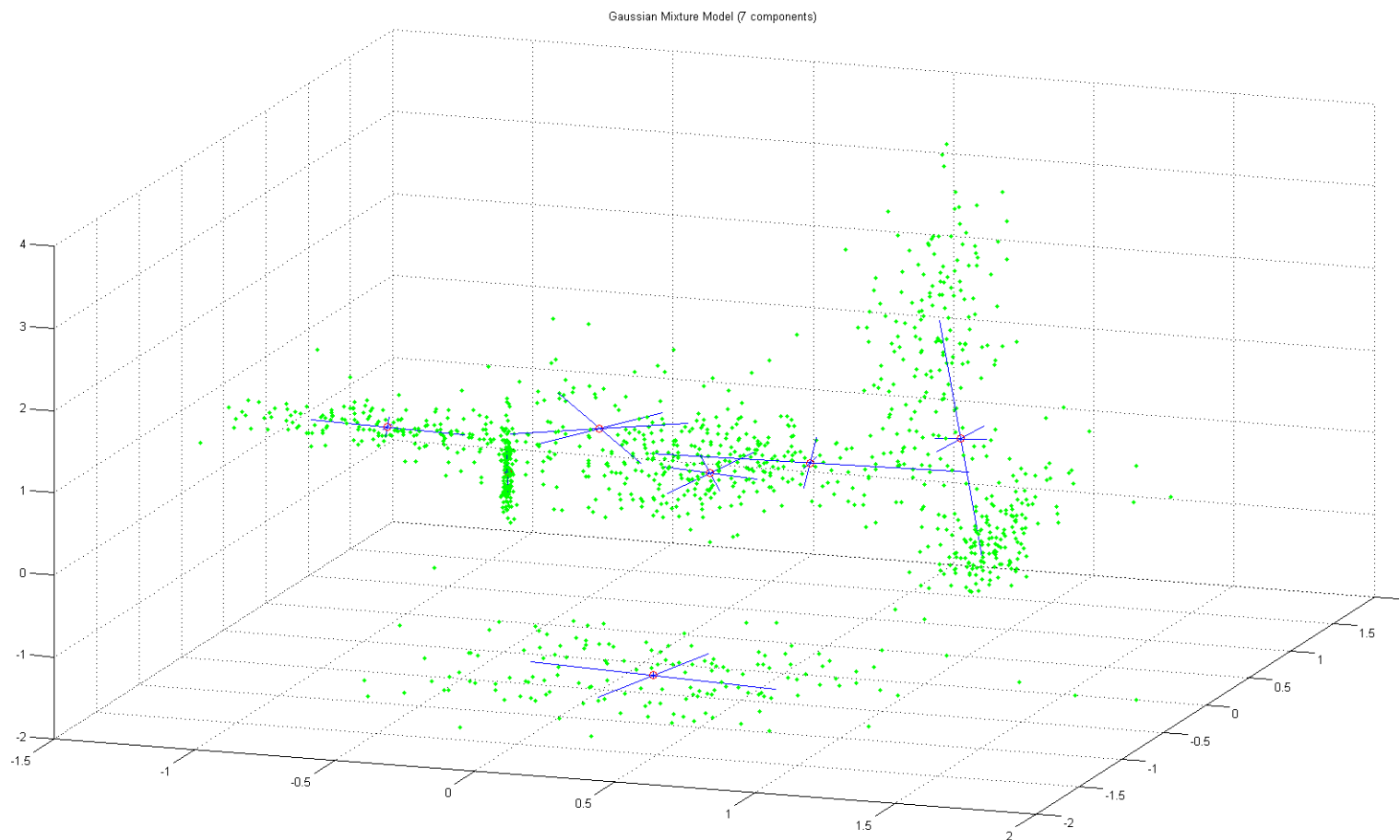
# Result for 5 Cluster



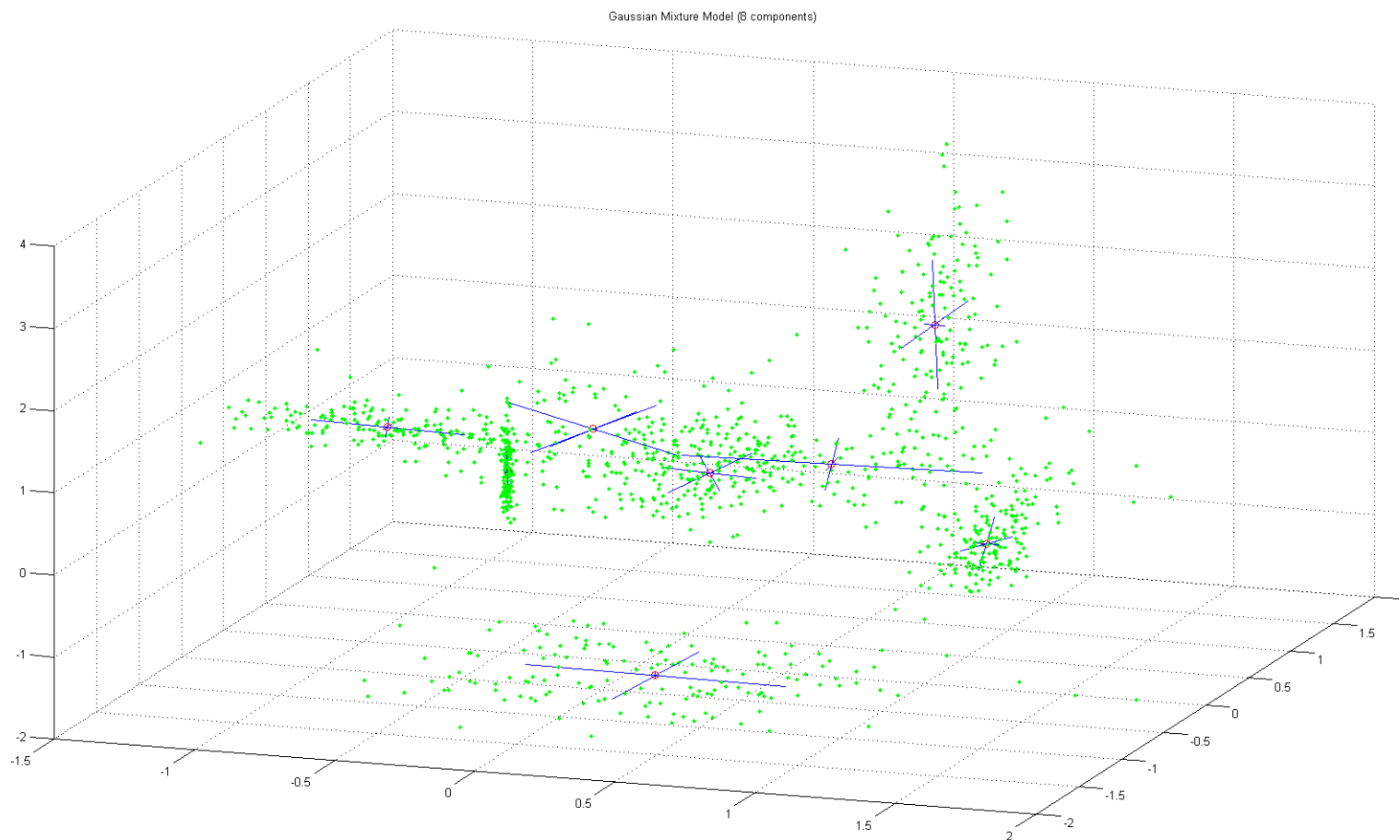
# Result for 6 Cluster



# Result for 7 Cluster



# Result for 8 Cluster



# Algorithm Outline (Just an information)

- Initialization of  $\Omega_0$ :  $\mu \leftarrow [0,0,0]^T$ ,  $\Sigma \leftarrow I$  (identity matrix)
- For  $c = 1, \dots, N_c$ 
  - $lastPX \leftarrow -\infty$
  - While  $(\log(PX) - lastPX > \varepsilon)$ 
    - $lastPX \leftarrow PX$
    - $p(y|x, \Omega) \leftarrow$  **E-Step**
    - $\Omega \leftarrow$  **M-Step**
  - Endwhile
  - $\Omega \leftarrow$  Initialize a new component
- Endfor

$$PX: p(x|\Omega) = \prod_{i=1}^{N_x} \sum_{c=1}^{N_c} \alpha_i p(x_i|\Omega_c)$$

- Scalar value: Probability of a dataset  $x$
- This value will be maximized during EM
- Used for decision when to stop iterating

# Task A

Algorithm is already implemented, but there are 3 functions missing. **Implement these functions:**

a. Function

$LnVectorProb = \text{CalcLnVectorProb}(\text{model}, \text{trainVect});$

calculates

$$\log(\alpha_c \cdot p(\mathbf{x}_i | \mu_c^k, \Sigma_c^k)) = \log(\alpha_c) - \frac{1}{2} [\log(|\Sigma_c^k|) + (\mathbf{x}_i - \mu_c)^T \Sigma_c^{-1} (\mathbf{x}_i - \mu_c)]$$

→ Log-probability of all feature vectors in all components

→  $N_c \times N_x$  array containing all possible  $\log(\alpha_c p(\mathbf{x}_i | \mu_c^k, \Sigma_c^k))$

→ Needed in E-step and also for other computations



# Task A

Algorithm is already implemented, but there are 3 functions missing. **Implement these functions:**

b. Function

$LnCompProb = \text{GmmEStep}(\text{model}, \text{trainVect})$

calculates

$$\log \left( p(y_i = c | x_i, \Omega_c^k) \right) = \log \left( \frac{\alpha_c p(x_i | \mu_c^k, \Sigma_c^k)}{\sum_{j=1}^{N_c} \alpha_j p(x_i | \mu_j^k, \Sigma_j^k)} \right)$$

- Also  $N_c \times N_x$  array
- Use **CalcLnVectorProb** for the nominator (*model*, *trainVect* only needed for this call!)
- Use *log*-values for computation and for the result!
- *LnCompProb* needed in M-step (function *GmmMStep*)

# Task A

Algorithm is already implemented, but there are 3 functions missing. **Implement these functions:**

## C. Function

*model* = **GmmMStep**(*model*, *trainVect*, *LnCompProb*)

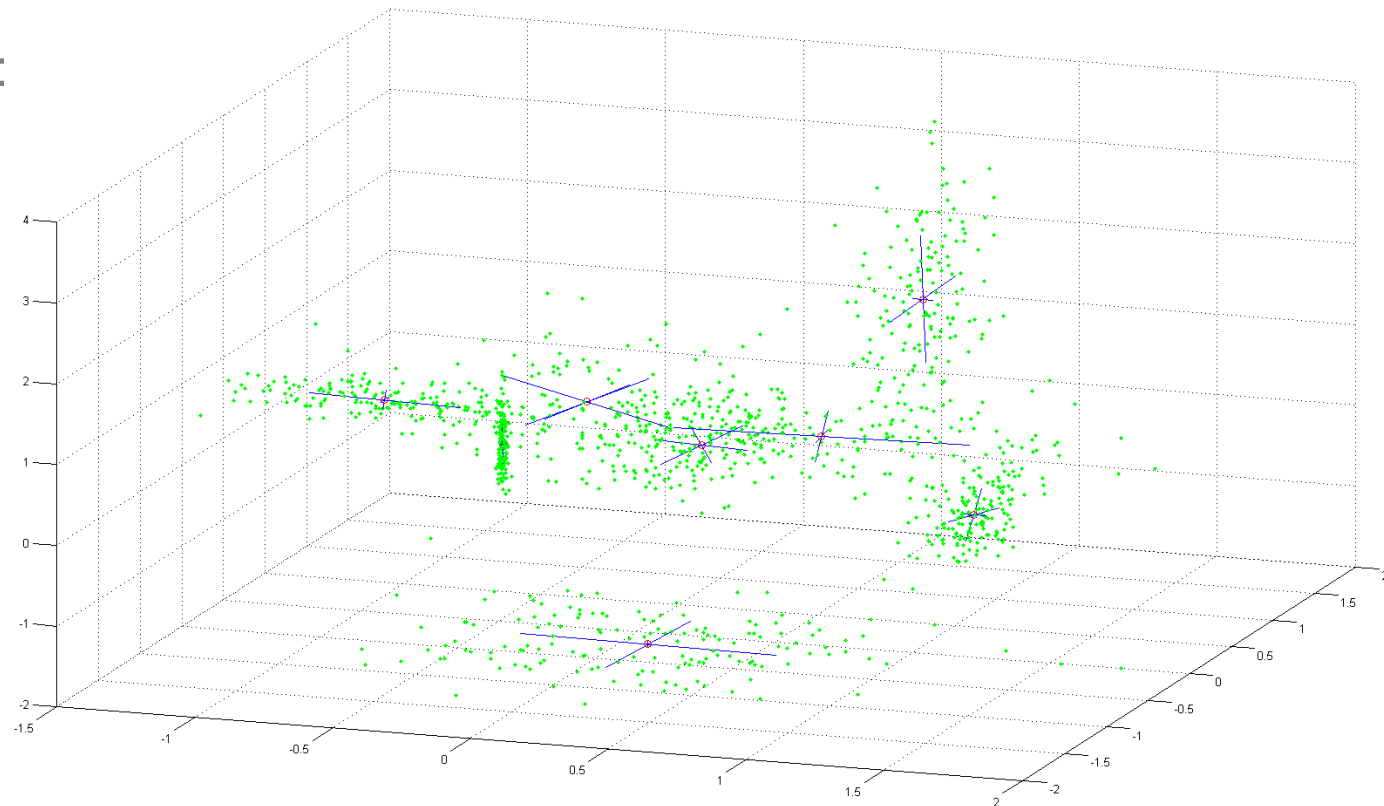
→ calculates the new model parameters  $\Omega^{k+1}$

→ Use *exp*-function to convert the values  $\log\left(p(y_i = c | x_i, \Omega_c^k)\right)$  (*LnCompProb*) before computations

# Task A

## d. Test your implementation using provided function *TestGaussMixEM*:

- Generates  $n_{comp}$  normally distributed point clouds
- Applies *LearnGaussMixModel.m*
- Example Result:



# Task B: ML-Image Segmentation

**Idea:** computation of  $\log$ -Likelihood  $\log(\alpha_c p(x_i | \mu_c, \Sigma_c))$  for each pixel (feature vector)  $\rightarrow$  store the index of the cluster with the maximum value for each pixel

$$\log(\alpha_1 p(x_i | \mu_1, \Sigma_1)) = -130$$

$$\log(\alpha_2 p(x_i | \mu_2, \Sigma_2)) = -190$$

$$\log(\alpha_3 p(x_i | \mu_3, \Sigma_3)) = -110$$

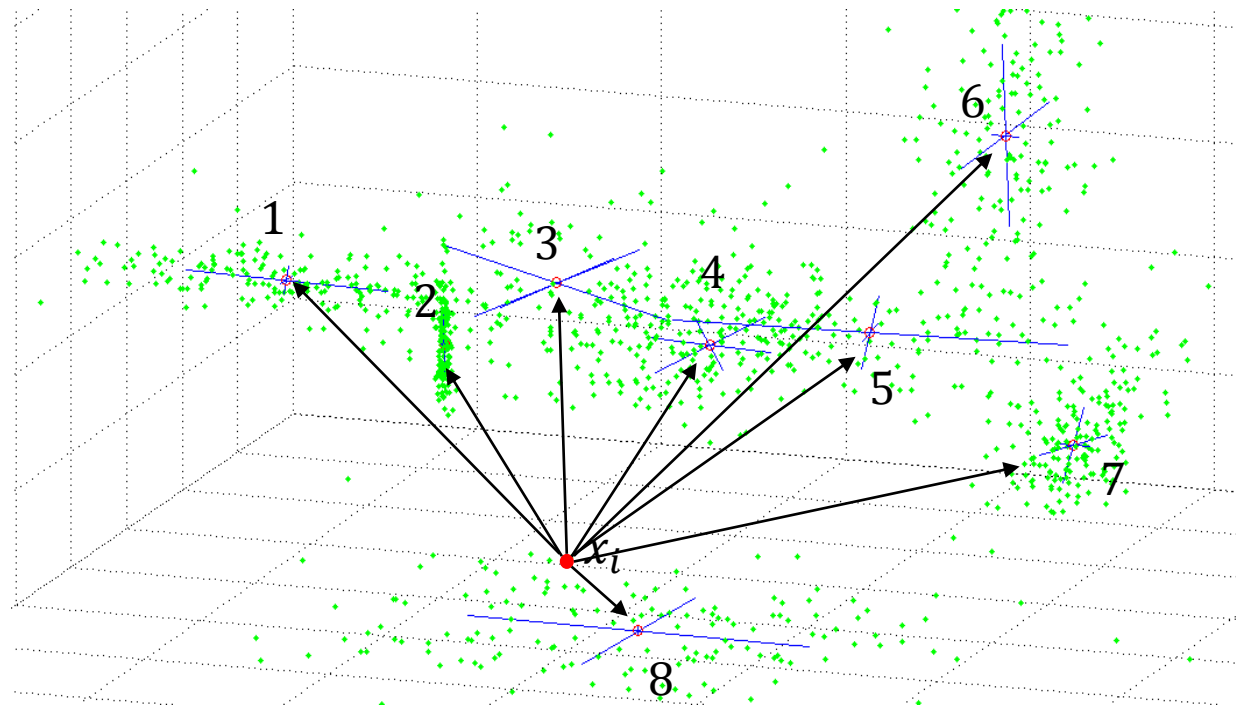
$$\log(\alpha_4 p(x_i | \mu_4, \Sigma_4)) = -90$$

$$\log(\alpha_5 p(x_i | \mu_5, \Sigma_5)) = -91$$

$$\log(\alpha_6 p(x_i | \mu_6, \Sigma_6)) = -221$$

$$\log(\alpha_7 p(x_i | \mu_7, \Sigma_7)) = -111$$

$$\log(\alpha_8 p(x_i | \mu_8, \Sigma_8)) = -16$$



$\rightarrow$  According to ML-criterion, pixel  $x_i$  is member of cluster 8!

# Task B: ML-Image Segmentation

**a.** **Copy** your implemented function *CalcLnVectorProb* from Task A into the given file *ApplyGaussMixEM.m*

→ Needed for the ML-Segmentation

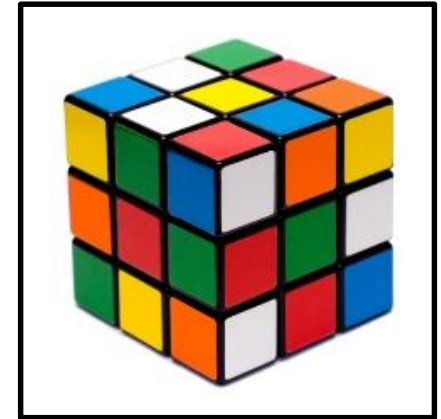
**b.** **Run** *ApplyGaussMixEM* and use the provided image *inputEx6.jpg* as input

→ Generates a subset of feature vectors from image

→ Learns a GMM

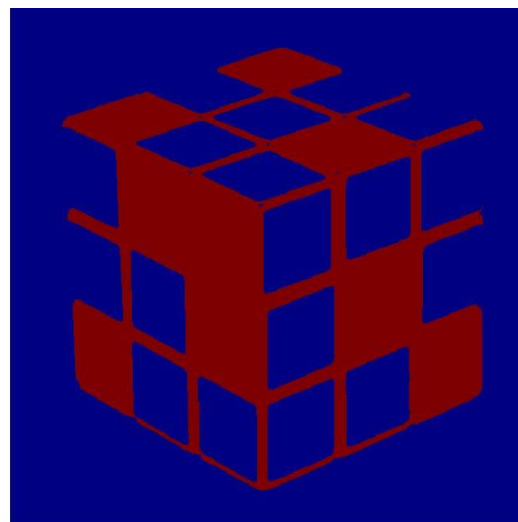
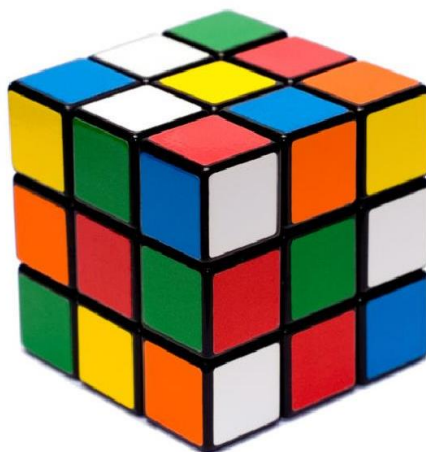
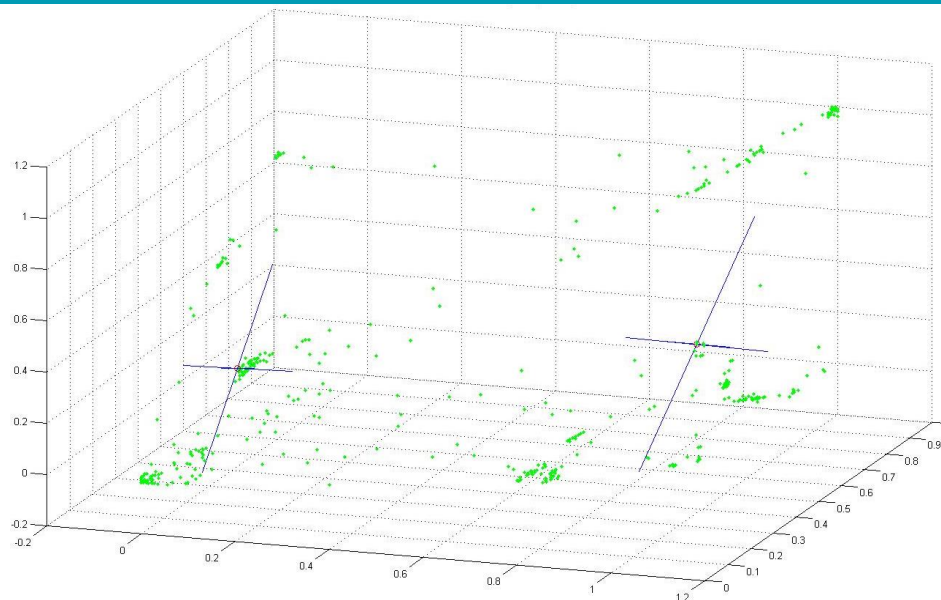
→ Classifys all pixels using ML-criterion

→ Plots the segmentation result



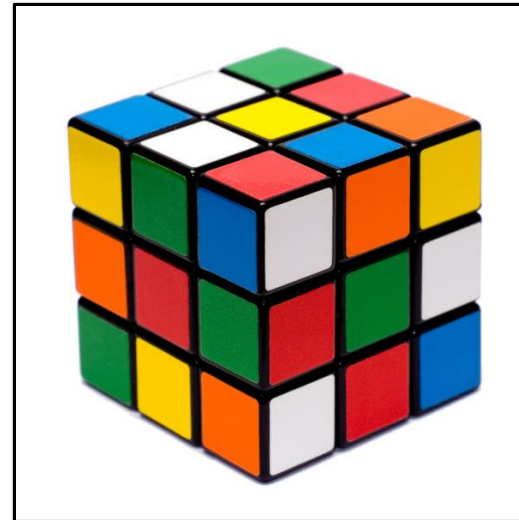
**c.** **Vary** the parameter  $n_{comp}$  (see source code of *ApplyGaussMixEM*). Which number is suitable for the given image? Can you observe any problems in segmenting the image?

# Task B: Example result, $n_{comp} = 2$



# Provided Data (See CV-Homepage)

- **Matlab** source
  - *LearnGaussMixModel.m* → For implementation of A
  - *TestGaussMixEM.m* → For testing of A
  - *ApplyGaussMixEM.m* → For classification in B
- **Octave** source
  - *ApplyGaussMixEM\_octave.m*
  - *LearnGaussMixModel.m*, *TestGaussMixEM.m* also work in Octave
- **Image:** *inputEx6.jpg*



# Task A: Implementation Details

## Given:

- *LearnGaussMixModel.m*
  - EM-algorithm outline
  - Initialization and Splitting
  - Visualization
  - **Function dummies for implementation:**
    - *CalcLnVectorProb*
    - *GmmEStep*
    - *GmmMStep*



# Task A: Implementation Details

## Inputs *CalcLnVectorProb*:

- *model*: Structure with current model parameters (**already present**)
  - *model.weight*: Vector with cluster weights
  - *model.weight(c)*: Weight  $\alpha_c$  of component  $c$   $\log \left( \alpha_c \cdot p(x_i | \mu_c^k, \Sigma_c^k) \right)$
  - *model.mean*: Mean vectors  $\mu$  of all components
  - *model.mean(c,:)*: Mean vector  $\mu_c$  (3 elements) of component  $c$
  - *model.covar*: Covariance matrices  $\Sigma$  of all components
  - ***squeeze(model.covar(c,:,:))***: Covariance matrix  $\Sigma_c$  ( $3 \times 3$  elements) of component  $c$
- *trainVect*:
  - Feature (or training) vectors (**given**)
  - *trainVect(i,:)*: 3-element feature vector

# Task A: Implementation Details

## Inputs *GmmEStep*:

- *model*: Structure with current model parameters (see above)
- *trainVect*: Feature (or training) vectors (see above)

## Inputs *GmmMStep*:

- *model*: Structure with current model parameters (see above)
- *trainVect*: Feature (or training) vectors (see above)
- *LnCompProb*: Output of E-Step

# Rules for Log-Value Computations

- $\log(a \cdot b) = \log(a) + \log(b)$
- $\log(a + b)$ : no mathematical rule using  $\log(a)$  and  $\log(b)$ !  
→  $\log(\exp(a) + \exp(b))$
- $\log\left(\frac{a}{b}\right) = \log(a) - \log(b)$
- $\log(\exp(a)) = a$
- $\log(a^b) = b \cdot \log(a)$



**Thank you!**