Unsupervised Learning of Finite Gaussian Mixture Models (GMMs): A Greedy Approach

Nicola Greggio[1,2,*], Alexandre Bernardino[1], and José Santos-Victor[1]

[1] Instituto de Sistemas e Robótica, Instituto Superior Técnico
1049-001 Lisboa, Portugal
[2] ARTS Lab - Scuola Superiore S.Anna, Polo S.Anna Valdera
Viale R. Piaggio, 34 - 56025 Pontedera, Italy

[*]nicola.greggio@ieee.org

Abstract. In this work we propose a clustering algorithm that learns on-line a finite gaussian mixture model from multivariate data based on the expectation maximization approach. The convergence of the right number of components as well as their means and covariances is achieved without requiring any careful initialization. Our methodology starts from a single mixture component covering the whole data set and sequentially splits it incrementally during the expectation maximization steps. Once the stopping criteria has been reached, the classical EM algorithm with the best selected mixture is run in order to optimize the solution. We show the effectiveness of the method in a series of simulated experiments and compare in with a state-of-the-art alternative technique both with synthetic data and real images, including experiments with the iCub humanoid robot.

Index Terms - Image Processing, Unsupervised Learning, (Self-Adapting) Gaussians Mixtures, Expectation Maximization, Machine Learning, Clustering

1 Introduction

Nowadays, computer vision and image processing are involved in many practical applications. The constant progress in hardware technologies leads to new computing capabilities, and therefore to the possibilities of exploiting new techniques, for instance considered to time consuming only a few years ago. Image segmentation is a key low level perceptual capability in many robotics related application, as a support function for the detection and representation of objects and regions with similar photometric properties. Several applications in humanoid robots [11], rescue robots [2], or soccer robots [6] rely on some sort on image segmentation [15]. Additionally, many other fields of image analysis depend on the performance and limitations of existing image segmentation algorithms: video surveillance, medical imaging and database retrieval are some examples [4], [12]. Two main principal approaches for image segmentation are adopted: Supervised and unsupervised. The latter one is the one of most practical interest. It may be defined as the task of segmenting an image in different regions based on some similarity criterion among each region's pixels.

One of the most widely used distributions is the Normal distribution. Due to the central limit theorem, any variable that is the sum of a large number of independent factors is likely to be normally distributed. For this reason, the normal distribution is used throughout statistics, natural science, and social science as a simple model for complex phenomena. If we model the entire dataset by a mixture of gaussians, the clustering problem, subsequently, will reduce to the estimation of the gaussians mixture's parameters.

1.1 Related Work

Expectation-Maximization (EM) algorithm is the standard approach for learning the parameters of the mixture model [8]. It is demonstrated that it always converges to a local optimum [3]. However, it also presents some drawbacks. For instance, EM requires an *a-priori* selection of model order, namely, the number of components to be incorporated into the model, and its results depend on initialization. The higher the number of components within the mixture, the higher will be the total log-likelihood. Unfortunately, increasing the number of gaussians will lead to overfitting and to an increase of the computational burden.

Particularly in image segmentation applications, where the number of points is in the order of several hundred thousand, finding the best compromise between precision, generalization and speed is a must. A common approach to choose the number of components is trying different configurations before determining the optimal solution, e.g. by applying the algorithm for a different number of components, and selecting the best model according to appropriate criteria.

Adaptive mixture models can solve the problem of the original EM's model selection. It was originally proposed in 2000 by Li and Barron [10], and subsequently explored in 2003 by Verbeek et al. in [14]. They developed a deterministic greedy method to learn the gaussians mixture model configuration [14]. At the beginning a single component is used. Then, new components are added iteratively and the EM is applied until it reaches the convergence.

Ueda *et Al.* proposed a split-and-merge EM algorithm to alleviate the problem of local convergence of the EM method [13]. Subsequently, Zhang *et Al.* introduced another split-and-merge technique [16]. Merge an split criterion is efficient in reducing number of model hypothesis, and it is often more efficient than exhaustive, random or genetic algorithm approaches. To this aim, particularly interesting is the method proposed by Figueiredo and Jain, which goes on step by step until convergence using only merge operations [5].

1.2 Our contribution

We propose an algorithm that simultaneously determines the number of components and the parameters of the mixture model with only split operations. In [7] we previously proposed a split and merge technique for learning finite Gaussian mixture models. However, the principal drawbacks were the initialization, with particular regards to the beginning number of mixture classes, and the superimposition of the split and merge operations. The particularly of our new model is that it starts from only one mixture component progressively adapting the mixture by splitting components when necessary.

In a sense, we approach the problem in a different way than [5]. They start the computation with the maximum possible number of mixture components. Although that work is among the most effective to date, it becomes too computationally expensive for image segmentation applications, especially during the first iterations. It starts with the maximum number of components, decreasing it progressively until the whole space of possibilities has been explored, whereas our method starts with a single component and increases its number until a good performance is attained.

1.3 Outline

The paper is organized as follows. In sec. 3 we introduce the proposed algorithm. Specifically, we describe its formulation in sec. 3.1, the initialization in sec. 3.2, the component split operation in sec. 3.4, and the decision thresholds update rules in sec. 3.5. Furthermore, in sec. 4 we describe our experimental set-up for testing the validity of our new technique and in sec. 5 we discuss our results. Finally, in sec. 6 we conclude.

2 Expectation Maximization Algorithm

2.1 EM Algorithm: The original formulation

A common usage of the EM algorithm is to identify the *"incomplete, or unobserved data"* $\mathcal{Y} = (\vec{y}^1, \vec{y}^2, \ldots, \vec{y}^k)$ given the couple $(\mathcal{X}, \mathcal{Y})$ - with $\bar{x}$ defined as $\mathcal{X} = \{\vec{x}^1, \vec{x}^2, \ldots, \vec{x}^k\}$, also called *"complete data"*, which has a probability density (or joint distribution) $p(\mathcal{X}, \mathcal{Y}|\bar{\vartheta}) = p_{\bar{\vartheta}}(\mathcal{X}, \mathcal{Y})$ depending on the parameter $\bar{\vartheta}$. More specifically, the *"complete data"* are the given input data set $\mathcal{X}$ to be classified, while the *"incomplete data"* are a series of auxiliary variables in the set $\mathcal{Y}$ indicating for each input sample which mixture component it comes from. We define $E'(\cdot)$ the expected value of a random variable, computed with respect to the density $p_{\bar{\vartheta}}(\bar{x}, \bar{y})$.

We define $Q(\bar{\vartheta}^{(n)}, \bar{\vartheta}^{(n-1)}) = E' L(\bar{\vartheta})$, with $L(\bar{\vartheta})$ being the log-likelihood of the observed data:

$$L(\bar{\vartheta}) = \log p_{\bar{\vartheta}}(\mathcal{X}, \mathcal{Y}) \tag{1}$$

The EM procedure repeats the two following steps until convergence, iteratively:

– E-step: It computes the expectation of the joint probability density:

$$Q(\bar{\vartheta}^{(n)}, \bar{\vartheta}^{(n-1)}) = E'[\log p(\mathcal{X}, \mathcal{Y}|\bar{\vartheta}^{(n-1)})] \tag{2}$$

– M-step: It evaluates the new parameters that maximize Q:

$$\bar{\vartheta}^{(n+1)} = arg \max_{\bar{\vartheta}} Q(\bar{\vartheta}^{n}, \bar{\vartheta}^{(n-1)}) \tag{3}$$

The convergence to a local maxima is guaranteed. However, the obtained parameter estimates, and therefore, the accuracy of the method greatly depend on the initial parameters $\hat{\vartheta}^0$.

2.2 EM Algorithm: Application to a Gaussians Mixture

When applied to a Gaussian mixture density we assume the following model:

$$\begin{aligned} p(\overline{x}) &= \sum_{c=1}^{nc} w_c \cdot p_c(\overline{x}) \\ p_c(\overline{x}) &= \frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma_c|^{\frac{1}{2}}} e^{-\frac{1}{2}(\overline{x}-\overline{\mu}_c)^T |\Sigma_c|^{-1}(\overline{x}-\overline{\mu}_c)} \end{aligned} \tag{4}$$

where $p_c(\overline{x})$ is the component prior distribution for the class c, and with d, $\overline{\mu}_c$ and Σ_c being the input dimension, the mean and covariance matrix of the gaussians component c, and nc the total number of components, respectively.

Consider that we have nc classes C_{nc}, with $p(\overline{x}|C_c) = p_c(\overline{x})$ and $P(C_c) = w_c$ being the density and the *a-priori* probability of the data of the class C_c, respectively. In this setting, the unobserved data set $\mathcal{Y} = \left(\vec{y}^1, \vec{y}^2, \ldots, \vec{y}^N\right)$ contains as many elements as data samples, and each vector $\vec{y}^i = \left[y_1^i, y_2^i, \cdots, y_c^i, \cdots y_{nc}^i\right]^T$ is such that $y_c^i = 1$ if the data sample x^i belongs to the class C_c and $y_c^i = 0$ otherwise. The expected value of the c^{th} component of the random vector $\overline{y}$ is the class C_c prior probability $E^{'}(y_c) = w_c$.

Then the E and M steps become, respectively:

E-step:

$$\begin{aligned} P\left(y_c^i = 1|\vec{x}^i\right) &= P\left(C_c|\vec{x}^i\right) \\ &= \frac{p\left(\vec{x}^i|C_c\right) \cdot P(C_c)}{p\left(\vec{x}^i\right)} = \frac{w_c \cdot p_c\left(\vec{x}^i\right)}{\sum_{c=1}^{nc} w_c \cdot p_c\left(\vec{x}^i\right)} \\ &\triangleq \pi_c^i \end{aligned} \tag{5}$$

For simplicity of notation, from now on we will refer to $E^{'}\left(y_c|x^i\right)$ as π_c^i. This is probability that the x^k belongs to the class C_c.

M-step:

$$\begin{aligned} \overline{\mu}_c^{(n+1)} &= \frac{\sum_{i=1}^{N} \pi_c^i \vec{x}^i}{\sum_{i=1}^{i} \pi_c^i} \\ \Sigma_c^{(n+1)} &= \frac{\sum_{i=1}^{N} \pi_c^i \left(\vec{x}^i - \overline{\mu}_c^{(n+1)}\right)\left(\vec{x}^i - \overline{\mu}_c^{(n+1)}\right)^T}{\sum_{i=1}^{N} \pi_c^i} \end{aligned} \tag{6}$$

Finally, *a-priori* probabilities of the classes, i.e. the probability that the data belongs to the class c, are reestimated as:

$$w_c^{(n+1)} = \frac{1}{N} \sum_{i=1}^{N} \pi_c^i, \quad with \;\; c = \{1, 2, \ldots, nc\} \tag{7}$$

3 FASTGMM: Fast Gaussian Mixture Modeling

Our algorithm starts with a single component and only increments its number as the optimization procedure progresses. With respect to the other approaches, our is the one with the minimal computational cost.

The key issue of our technique is looking whether one or more gaussians are not increasing their own likelihood during optimization. Our algorithm evaluates the current likelihood of each single component c as (8):

$$\Lambda_{curr\,(c)}\left(\vartheta\right) = \sum_{i=1}^{N} log\left(w_c \cdot p_c\left(\overline{x_i}\right)\right) \tag{8}$$

In other words, if their likelihood has stabilized they will be split into two new ones and check if this move improves the likelihood in the long run. For our algorithm we need to introduce a state variable related to the state of the gaussian component:

– Its age, that measures how long the component's own likelihood does not increase significantly (see sec. 3.1);

Then, the split process is controlled by the following adaptive decision thresholds:

– One adaptive threshold Λ_{TH} for determining a significant increase in likelihood (see sec. 3.5);
– One adaptive threshold A_{TH} for triggering the split process based on the component's own age (see sec. 3.5);
– One adaptive threshold ξ_{TH} for deciding to split a gaussian based on its area (see sec. 3.4).

It is worth noticing that even though we consider three thresholds to tune, all of them are adaptive, and only require a coarse initialization.

These parameters will be fully detailed within the next sections.

3.1 FASTGMM Formulation

Our algorithm's formulation can be summarized within three steps:

– Initializing the parameters;
– Splitting a gaussian;
– Updating decision thresholds.

Each mixture component c is represented as follows:

$$\bar{\vartheta}_c = \rho\left(w_c, \bar{\mu}_c, \Sigma_c, \xi_c, \Lambda_{last\,(c)}, \Lambda_{curr\,(c)}, a_c\right) \tag{9}$$

where each element is described in tab. I. In the rest of the paper the index notation described in tab. I will be used.

Here, we define two new elements, the area (namely, the covariance matrix determinant) and the age of the gaussians, which will be described later.

During each iteration, the algorithm keeps memory of the previous likelihood. Once the re-estimation of the vector parameter ϑ has been computed in the EM step, our algorithm evaluates the current likelihood of each single component c as:

If a_i overcomes the age threshold A_{TH} (i.e. the gaussians i does not increase its own likelihood for a predetermined number of times significally - over Λ_{TH}), the algorithm decides whether to split this gaussians depending on whether their own single area overcome ξ_{TH}.

The whole algorithm pseudocode is shown in Algorithm 1.

Symbol	Element
w_c	*a-priori* probabilities of the class c
$\bar{\mu}_c$	mean of the gaussian component c
Σ_c	covariance matrix of the gaussian component c
ξ_c	area of the gaussian component c
$\Lambda_{last(c)}$	log-likelihood at iteration $t-1$ of the gaussian component c
$\Lambda_{curr(c)}$	log-likelihood at iteration t of the gaussian component c
a_c	*age* of the gaussian component c
c	single mixture component
nc	total number of mixture components
i	single input point
N	total number input points
d	single data dimension
D	input dimensionality

Table 1: Symbol notation used in this paper

3.2 <u>Parameters initialization</u>

The decision thresholds $(\cdot)_{INIT}$ will be initialized as follows:

$$\begin{aligned} \xi_{TH-INIT} &= \xi_{data}; \\ L_{TH-INIT} &= k_{LTH}; \\ A_{TH-INIT} &= k_{ATH} \end{aligned} \tag{10}$$

with k_{LTH} and k_{ATH} (namely, the minimum amount of likelihood difference between two iterations and the number of iterations required for taking into account the lack of a likelihood consistent variation) relatively low (i.e. both in the order of 10, or 20). Of course, higher values for k_{LTH} and smaller for k_{ATH} give rise to a faster adaptation, however adding instabilities.

At the beginning, before starting with the iterations, ξ_{TH} will be automatically initialized to the Area of the whole data set - i.e. the determinant of the covariance matrix relative to all points, as follows:

$$\begin{aligned} \mu_{data,d} &= \frac{1}{N}\sum_{i}^{N} x_d^N \\ \Sigma_{data,i} &= \langle \bar{x_i} - \bar{\mu}_{data} \rangle \langle \bar{x_i} - \bar{\mu}_{data} \rangle^T \end{aligned} \tag{11}$$

where N is the number of input data vectors $\bar{x}$, and D their dimensionality.

3.3 <u>Gaussian components initialization</u>

The algorithm starts with just only one gaussian. Its mean will be the whole data mean, as well as its covariance matrix will be that of the whole data set.

That leads to a unique starting configuration.

Algorithm 1: FASTGMM: Pseudocode

```
1  Parameter initialization;
2  while (stopping criterion is not met) do
3      $\Lambda_{curr\,(c)}$, evaluation, for $c = 0, 1, \dots, nc$;
4      Whole mixture log-likelihood $L\left(\bar{\vartheta}\right)$ evaluation;
5      Re-estimate priors $w_c$, for $c = 0, 1, \dots, nc$;
6      Recompute center $\vec{\mu}_c^{(n+1)}$ and covariances $\Sigma_c^{(n+1)}$, for $c = 0, 1, \dots, nc$;
       - Evaluation whether changing the gaussians distribution structure;
7      for ($c = 0$ to nc) do
8          if ($a_c > A_{TH}$) then
9              if (($\Lambda_{curr\,(c)} - \Lambda_{last\,(c)}$) $< \Lambda_{TH}$) then
10                 $a_c += 1$;
                   - General condition for changing satisfied; now checking
                   those for each component;
11                 if ($\Sigma_c > \xi_{TH}$) then
12                     if ($c < maxNumComponents$) then
13                         $\underline{\text{split gaussians}} \rightarrow \underline{\text{split}}$;
14                         $nc += 1$;
15                         reset $\xi_{TH} \leftarrow \frac{\xi_{TH-INIT}}{nc}$;
16                         reset $\Lambda_{TH} \leftarrow L_{TH-INIT}$;
17                         reset $a_A, a_B \leftarrow 0$ - with A, B being the new two
                           gaussians;
18                         return;
19                 $\xi_{TH} = \xi_{TH} \cdot (1 + \alpha \cdot \xi)$;
20     Optional: Optimizing selected mixture;
```

3.4 Splitting a Gaussian

When a component's covariance matrix area overcomes the maximum area threshold ξ_{TH} it will split. As a measure of the area we adopt the matrix's determinant. This, in fact, describes the area of the ellipse represented by a gaussian component in 2D, or the volume of the ellipsoid represented by the same component in 3D.

It is worth noticing that the way the component is split greatly affects further computations. For instance, consider a 2-dimensional case, in which an *elongated* gaussian is present. Depending on the problem at hand, this component may approximating two components with diverse configurations: Either covering two smaller data distribution sets, placed along the longer axis, or two ovelapped sets of data with different covariances, etc. A reasonable way of splitting is to put the new means at the two major semi-axis' middle point. Doing so, the new components will promote non overlapping components and, if the actual data set reflects this assumption, it will result in faster convergence.

To implement this split operation we make use of the singular value decomposition. A rectangular n x p matrix A can be decomposed as $A = USV^T$, where the columns of U are the left singular vectors, S (which has the same dimension as A) is a diagonal matrix with the singular values arranged in descending order, and V^T has rows that are the right singular vectors. However, we are not interested in the whole set of eigenvalues, but only the bigger one, therefore we can save some computation by evaluating only the first column of U and the first element of S.

More precisely, A gaussian with parameters $\overline{\vartheta}_{OLD}$ will be split in two new gaussians A and B, with means:

$$\begin{aligned} &\overline{\Sigma}_{OLD} = USV^T \\ &\overline{u}_{MAX} = U_{*,1}; \quad s_{MAX} = S_{1,1} \\ \overline{\mu}_A = \overline{\mu}_{OLD} + \frac{1}{2} s_{MAX} \overline{u}_{MAX}&; \quad \overline{\mu}_B = \overline{\mu}_{OLD} - \frac{1}{2} s_{MAX} \overline{u}_{MAX} \end{aligned} \tag{12}$$

where $\overline{u}_{MAX}$ is the first column of U, and s_{MAX} the first element of S.

The covariance matrices will then be updated as:

$$S_{1,1} = \frac{1}{4} s_{MAX}; \quad \Sigma_A = \Sigma_B = USV^T \tag{13}$$

while the new *a-priori* probabilities will be:

$$w_A = \frac{1}{2} w_{OLD}; \quad w_B = \frac{1}{2} w_{OLD} \tag{14}$$

The decision thresholds will be updated as explained in sec. 3.5.

Finally, their ages, a_A and a_B, will be reset to zero.

3.5 Updating decision thresholds

The decision thresholds are updated in two situations:

A. When a mixture component is split;
B. When each iteration is concluded.

These two procedures will be explained in the following.

- Single iteration: The thresholds Λ_{TH}, and ξ_{TH} vary at each step with the following rules:

$$\begin{aligned} \Lambda_{TH} = \Lambda_{TH} - \frac{\lambda}{nc^2} \cdot \Lambda_{TH} = \Lambda_{TH} \cdot \left(1 - \frac{\lambda}{nc^2}\right) \\ \xi_{TH} = \xi_{TH} - \frac{\alpha_{MAX}}{nc^2} \cdot \xi_{TH} = \xi_{TH} \cdot \left(1 - \frac{\alpha_{MAX}}{nc^2}\right) \end{aligned} \tag{15}$$

with *nc* is the number of current gaussians, λ, and α_{MAX} are the coefficients for the likelihood and area change evaluation, respectively. Using high values for λ, and α_{MAX} results in high convergence speed. However, a faster convergence is often associated to instability around the optimal point, and may lead to a divergence from the local optimum. We can say that α_{MAX} can be interpreted as the *speed* the mixture components are split. In normal conditions, ξ_{TH} will become closer to the *area* of the bigger component's determinant step-by-step at each iteration. Then, it will approach the split threshold, allowing the splitting procedure.

Following an analog rule, Λ_{TH} will decrease step by step, approaching the current value of the global log-likelihood increment. This will allow the system to avoid some local optima, by varying its configuration whether a stationary situation occurs. Moreover, dividing λ and α_{MAX} by the square of *nc* consents to reduce the variation of the splitting threshold according to the number of components increases with a parabolic curve. This favorites the splitting when a low number of components is present, while avoiding a diverging behavior in case of an excessive amount of splitting operations.

Finally, every time a gaussians is added these thresholds will be reset to their initial value (see next section).

- After gaussian splitting: The decision thresholds will be updated as follows:

$$\xi_{TH} = \frac{\xi_{TH-INIT}}{nc}; \quad \Lambda_{TH} = L_{TH-INIT} \tag{16}$$

where nc_{OLD} and nc are the previous and the current number of mixture components, respectively. Substantially, this updates the splitting threshold to a value that goes linearly with the initial value and the actual number of components used for the computation.

3.6 Optimizing the selected mixture

This is an optional procedure. Once the best, or chosen, mixture, is saved, there are two possibilities:

1. Keeping the chosen mixture as the final result;
2. Optimizing the chosen mixture with the original EM algorithm.

The first one is the fastest but less accurate, while the second one introduces new computations ensuring more precision. It may happen that FASTGMM decides to increase the number of components even though the EM has not reached its local maximum, due to the splitting rule. In this case current mixture can still be improved by running the EM until it achieves its best configuration (the log-likelihood no longer increases).

Whether applying the first or second procedure is a matter of what predominates in the *”number of iterations vs. solution precision”* compromise at each time.

3.7 Computational complexity evaluation

We refer to the pseudocode in algorithm 1, and to the notation presented in sec. 3.1.The computational burden of each iteration is:

– the original EM algorithm (steps 3 to 6) takes $O(N \cdot D \cdot nc)$ for each step, for a total of $O(4 \cdot N \cdot D \cdot nc)$ operations;
– our algorithm takes $O(nc)$ for evaluating all the gaussians (step 7 to 7);
– our split (step 13) operation requires $O(D)$.
– the others take $O(1)$.
– the optional procedure of optimizing the selected mixture (step 20) takes $O(4 \cdot N \cdot D \cdot nc)$, being the original EM.

Therefore, the original EM algorithm takes:

– $O(4 \cdot N \cdot D \cdot nc)$, while our algorithm adds $O(D \cdot nc)$ on the whole, or $O(4 \cdot N \cdot D \cdot nc)$, giving rise to $O(4 \cdot N \cdot D \cdot nc) + O(D \cdot nc) = O(4 \cdot N \cdot D \cdot nc + D \cdot nc) = (nc \cdot D \cdot (4N + 1))$ in the first case;

– $2 \cdot O(4 \cdot N \cdot D \cdot nc) + O(D \cdot nc) = O(8 \cdot N \cdot D \cdot nc + D \cdot nc) = (nc \cdot D \cdot (8N + 1))$ in the second case, with the optimization procedure.

Considering that usually $D << N$ and $nc << N$, and that the optimization procedure is not essential, our procedure does not add a considerable burden, while giving an important improvement to the original computation in terms of self-adapting to the data input configuration at best. Moreover, it is worth noticing that even though the optimization procedure is performed, this starts very close to the optimal mixture configuration. In fact, the input mixture is the result of the FASTGMM computation, rather than a generic random or k-means initialization (as it happens with the simple EM algorithm, generally).

4 Experiments

Since now we use the following notation:

– FASTGMM: Our algorithm;
– FIGJ: [5].

4.1 Synthetic data

We tested it by classifying different input data sets randomly generated by a known gaussians mixture. The same input sets have been proposed to [5]. Each distribution has a total of 2000 points, but arranged with different mixture distributions, with 3, 4, 8, and 16 Gaussian components.

The output of the two algorithms is shown in Fig. 1. Each subplot set is composed by the graphical output representation for the 2-D point distribution (top) and the 3-D estimation mixture histogram (bottom). The data plots show the generation mixture (blue) and the evaluated one (red). On the left the data result from our approach is shown, while on the right those of [5], relative to the same input data set. Moreover, the 3D histograms at the bottom in each subfigure represent: The generated mixture, our algorithm's estimated one, and that estimated by [5], respectively.

We can see that our algorithm is capable to learn the input data mixture starting from only one component with an accuracy comparable with those of [5].

4.2 Colored real images

We segmented the images as 3-dimensional input in the (R,G,B) space. The color image segmentation results are shown in Fig. 2.The set of images is divided into two groups: Some general images, on the left (from (1) to (6)), and some images taken by the iCub's cameras, on the right (from (7) to (12)). For each group we show the original images, those obtained with [5], and those obtained with our algorithm on the left, in the middle, and on the right, respectively.

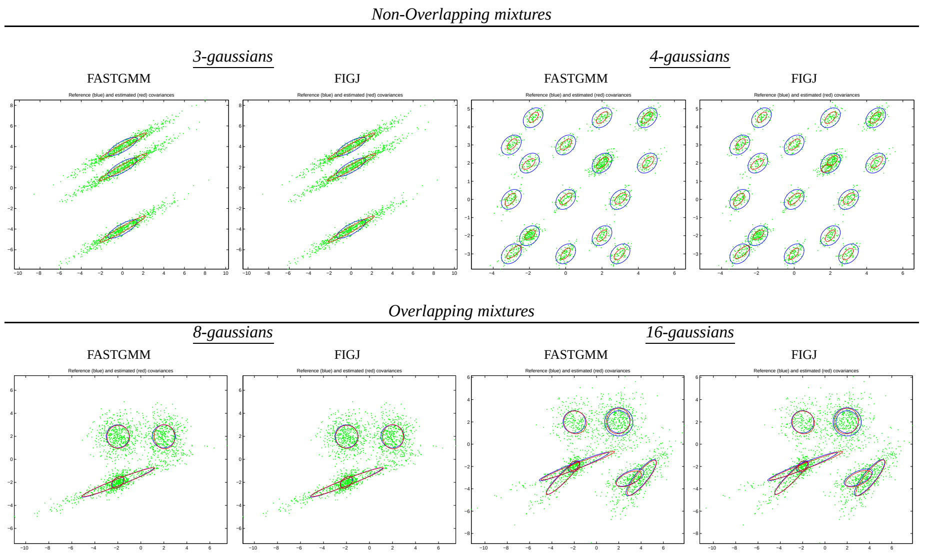

Fig. 1: For each plot set: Generation mixture (blue) and the evaluated one (red) for FASTGMM and FIGJ on the same input sets. Moreover, the 3D histograms at the bottom in each subfigure represent: The generated mixture, our algorithm's estimated one, and that estimated by [5], respectively.

Table 2: Experimental results on 2D synthetic data.

Input	Algorithm	# Initital gaussians	# Detected gaussians	Actual gaussian number	# Iterations	Elapsed Time [s]	Diff time FASTGMM with opt vs FIGJ%	Log-likelihood	Diff lik FASTGMM with opt vs FIGJ %	Normalized L2 Distance without optimization	Normalized L2 Distance with optimization	Crashed
	FSAEM		3		76	3.99716		-8420.917867				
3	Optimization	1	3	3	130	6.151567	-53.89844289	-8379.161274	0.495867477	5.770135	3.918034	no
	FSAEM + Opt.		3		206	10.148727		-8379.161274				
	FIGJ	16	3		277	29.433288		-9524.692099		3.670464	3.670464	no
	FSAEM		4		101	5.615204		-7573.101881				
4	Optimization	1	4	4	186	12.531248	-123.166389	-7405.078438	2.218687212	10.670613	0.07519	no
	FSAEM + Opt.		4		287	18.146452		-7405.078438				
	FIGJ	16	4		205	13.52505		-8729.761818		0.076403	0.076403	no
	FSAEM		9		276	5.750431		-8599.51				
8	Optimization	1	9	8	199	4.428076	22.99575458	-8598.17	0.015582283	0.196817	1.971166	no
	FSAEM + Opt.		9		475	10.178507		-8598.17				
	FIGJ	16	7		333	48.156629		-9798.154848		0.14491	0.14491	no
	FSAEM		16		501	26.667825		-8165.436422				
16	Optimization	1	16	16	202	12.848394	51.82061529	-8160.778985	0.057038433	0.251515	1.033934	no
	FSAEM + Opt.		16		703	39.516219		-8160.778985				
	FIGJ	20	13		363	63.740854		-9540.91802		2.98916	2.98916	no

5 Discussion

5.1 Synthetic data

In table 2 the results of FASTGMM and FIGJ applied to the selected images are shown.

5.1.1 - Evaluated number of components: There are substantially no differences in the selected number of components. Both our approach and [5] perform well on low mixture components, while having the tendency of underestimating the best number when it increases, with exception for our approach that overestimates it with the 8-component synthetic data and acting exactly with the 16-components. However, it is worth considering that even though it approaches the actual number correctly, this not necessary means that the components are in the right place. For instance, two components may be regarded as only one, while a single one can be considered as a multiple one. Nonetheless, it happens for both algorithm (see Fig. 1), suggesting that a perfect algorithm is hard to find.

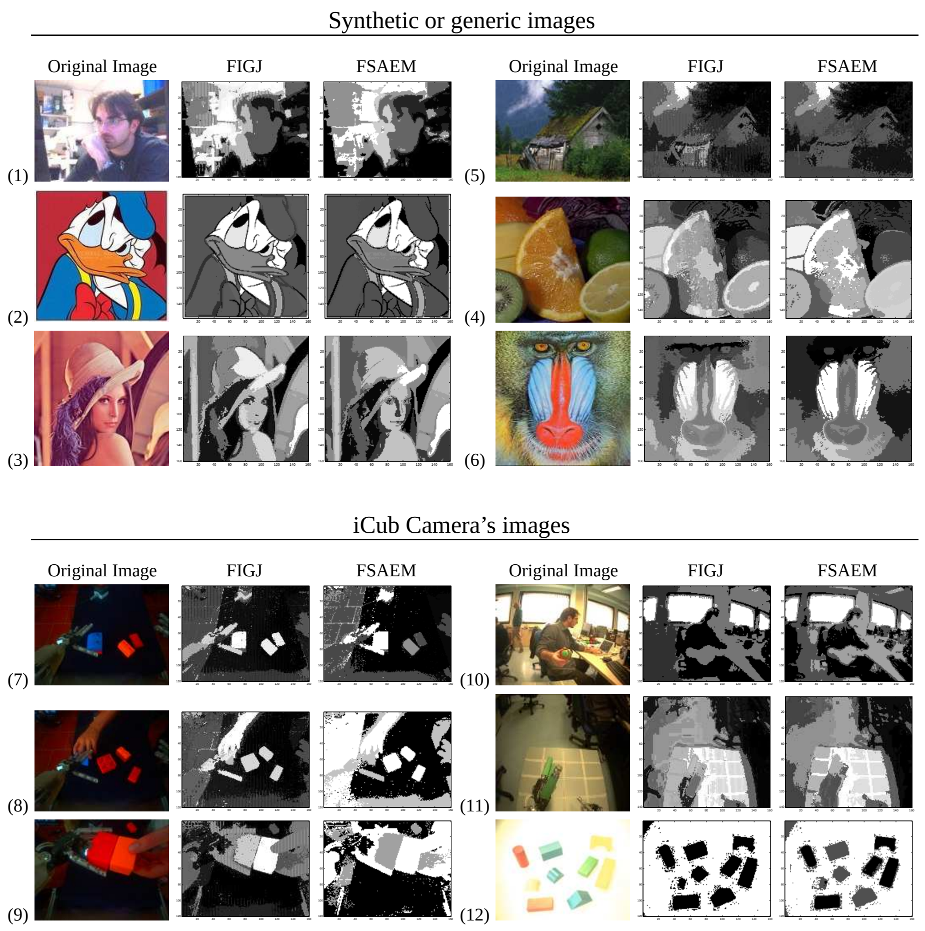

Fig. 2: Color images segmentation. From image (1) to (6) we tested the algorithms on well-known images, or synthetic ones, and from (7) to (12) we exploit the algorithms possibilities on real images captured by our robotic platform iCub's cameras. Note: FIGJ has not been able to segment image (13) also starting with merely 2 components, due to some internal covariances ill-posedness problems.

Input	Algorithm	# Initial gaussians	# Detected gaussians	# Iterations	Elapsed Time [s]	Diff time FASTGMM with opt vs FIGJ %	Log-likelihood	Diff lik FASTGMM with opt vs FIGJ	Diff lik FASTGMM with opt vs FIGJ	Crashed
	FASTGMM		9	551	71.460507		-235130.6216			
1	Optimization	1	9	23	22.131293	130.9699636	-234692.3977	0.186374662	17.03330409	no
	FASTGMM + Opt.		9	700	93.5918		-234692.3977			
	FIGJ	16	16	422	307.454885		-274668.2675			yes, 17
	FASTGMM		9	426	68.31949		-301594.6385			
2	Optimization	1	9	3	1.950391	2.854809074	-301594.6384	6.29984E-09	39.70671245	no
	FASTGMM + Opt.		9	429	70.269882		-301594.6384			
	FIGJ	19	19	308	441.170848		-421347.9543			yes, 20
	FASTGMM		14	426	80.365553		-314931.44			
3	Optimization	1	14	374	88.057387	109.5710584	-314551.5352	0.120630946	11.16531396	no
	FASTGMM + Opt.		14	800	168.422941		-314551.5352			
	FIGJ	30	25	572	1611.816189		-349672.2017			no
	FASTGMM		12	451	75.660802		-235735.0604			
4	Optimization	1	11	103	21.39836	28.28196296	-235729.2834	0.00245062	28.67314092	no
	FASTGMM + Opt.		11	554	97.059162		-235729.2834			
	FIGJ	18	19	329	500.708511		-303320.2731			yes 19
	FASTGMM		13	451	80.165239		-311585.8059			
5	Optimization	1	13	246	57.748543	72.03688746	-311167.3045	0.134313369	10.60749104	no
	FASTGMM + Opt.		13	697	137.913782		-311167.3045			
	FIGJ	30	27	611	2110.020131		-344174.3484			no
	FASTGMM		7	276	34.977055		-226138.1494			
6	Optimization	1	7	36	5.860168	16.7543208	-226138.0738	3.34203E-05	17.57803149	no
	FASTGMM + Opt.		7	312	40.837223		-226138.0738			
	FIGJ	16	16	420	272.227272		-265888.6956			yes, 17
	FASTGMM		7	576	72.744922		-281176.5478			
7	Optimization	1	7	14	2.610093	3.588007146	-281176.5441	1.30736E-06	15.72288044	no
	FASTGMM + Opt.		7	590	75.355015		-281176.5441			
	FIGJ	11	11	267	106.453566		-325385.5959			yes, 12
	FASTGMM		8	426	54.119749		-205718.9337			
8	Optimization	1	8	46	6.478271	11.97025322	-205718.7793	7.50364E-05	26.47107115	no
	FASTGMM + Opt.		8	472	60.59802		-205718.7793			
	FIGJ	11	11	228	90.200195		-260174.7438			yes, 12
	FASTGMM		4	251	25.685892		-211551.64			
9	Optimization	1	4	3	1.341008	5.220795914	-211551.64	0	15.41441064	no
	FASTGMM + Opt.		4	254	27.026901		-211551.64			
	FIGJ	13	13	313	137.77516		-244161.0785			yes, 14
	FASTGMM		3	201	18.822216		-230138.8367			
10	Optimization	1	3	63	6.427956	34.15089913	-230136.557	0.000990588	14.70275567	no
	FASTGMM + Opt.		3	264	25.250172		-230136.557			
	FIGJ	14	13	275	106.626624		-263972.9727			yes, 14
	FASTGMM		11	451	67.534808		-210899.0185			
11	Optimization	1	10	180	31.981125	47.35502469	-210657.4353	0.114549212	17.68706256	no
	FASTGMM + Opt.		10	631	99.515933		-210657.4353			
	FIGJ	24	22	514	624.913104		-247916.5477			no
	FASTGMM		3	151	16.899695		-218447.7912			
12	Optimization	1	3	23	2.27548	13.4646217	-218447.7848	2.96364E-06	16.0195988	no
	FASTGMM + Opt.		3	174	19.943671		-218447.7848			
	FIGJ	12	12	260	130.416222		-253442.2435			yes, 13

Table 3: Experimental results on real images segmentation.

5.1.2 - Elapsed time: It is important to distinguish the required number of iterations from the elapsed time. FASTGMM employs fewer iterations than FIGJ without making use of the optimization process, while more in the other case. At a first glance, this may suggest a whole FASTGMM slower computation than FIGJ. However, the whole elapsed time that occurs for running our procedure is generally less than FIGJ's. Nevertheless, we made FIGJ starting with a reasonable number of components, just a few more than the optimum, so that they do not affect its performance negatively. FASTGMM's better performance is due to the fact that our approach, growing in the number of components, computes more iterations than FIGJ but with a small number of components per iteration. Therefore it runs each iteration faster, while slowing only at the end due to the augmented number of components.

5.1.3 - Mixture precision estimation: It is possible to see that FASTGMM usually achieves a higher final log-likelihood than FIGJ. This suggests a better approximation of the data mixture. However, a higher log-likelihood does not strictly imply that the extracted mixture covers the data better than another one. This is because it is based on the probability of each component, which may be more or less exact, being not deterministic. Nevertheless, it is a good index on the probability that such mixture would be better.

A deterministic approach is to adopt a unique distance measure between the generation mixture and the evaluated one. In [9] Jensen *et Al.* exposed three different strategies for computing such distance: The Kullback-Leibler, the Earh Mover, and the Normalized L2 distance. The first one is not symmetric, even though a symmetrized version is usually adopted in music retrival. However, this measure can be evaluated in a close form only with mono-dimensional gaussians. The second one also suffers analog problems of the latter. The third choice, finally is symmetric, obeys to the triangle inequality and it is easy to compute, with a comparable precision with the other two. We then used the last one. Its expression states [1]:

$$\begin{aligned}
z_c N_x\left(\bar{\mu}_c, \bar{\Sigma}_c\right) &= N_x\left(\bar{\mu}_a, \bar{\Sigma}_a\right) \cdot N_x\left(\bar{\mu}_b, \bar{\Sigma}_b\right) \\
&\textit{where} \\
\bar{\Sigma}_c = \left(\bar{\Sigma}_a^{-1} + \bar{\Sigma}_b^{-1}\right)^{-1} \quad &\textit{and} \quad \bar{\mu}_c = \bar{\Sigma}_c\left(\bar{\Sigma}_a^{-1}\bar{\mu}_a + \bar{\Sigma}_b^{-1}\bar{\mu}_b\right) \\
z_c &= |2\pi\bar{\Sigma}_a\bar{\Sigma}_b\bar{\Sigma}_c^{-1}|^{\frac{1}{2}} e^{-\frac{1}{2}(\bar{\mu}_a-\bar{\mu}_b)^T\bar{\Sigma}_a^{-1}\bar{\Sigma}_c\bar{\Sigma}_b^{-1}(\bar{\mu}_a-\bar{\mu}_b)} \\
&= |2\pi\left(\bar{\Sigma}_a+\bar{\Sigma}_b\right)|^{\frac{1}{2}} e^{-\frac{1}{2}(\bar{\mu}_a-\bar{\mu}_b)^T\left(\bar{\Sigma}_a+\bar{\Sigma}_b\right)^{-1}(\bar{\mu}_a-\bar{\mu}_b)}
\end{aligned} \tag{17}$$

5.2 Colored real images

It is salient to report that in [5] it has not been performed any experiment on real images segmentation. In fact, their result only concern different families of synthetic data. Contrariwise, we want to focus more on image processing, due to its relevant importance in several different scientific fields, like robotics and medicine, as mentionned within the introduction.

As we pointed out in the previous section (sec. 4.2), to compare a detected mixture versus a generic image is not possible quantitatively, only qualitatively. This is due to the high number of colors present within the image. It is obvious that with more components the image is better reconstructed. However, it is possible to visually recognize a pattern even with fewer components, although with less accuracy. Therefore, what algorithm gives the best result is again a matter of what compromise is better, in terms of computational complexity and result accuracy.

Moreover, we have to report a problem that has not been addressed in the original work [5]. This crashes with some images when the number of components increases too much. Than means that it is not able to finish the computation. Therefore, as reported on tab. 3 we had to start it with a relatively low number of gaussians. However, even though it is able to finish the computation, it very often returns a mixture having the same starting number of components as the best one. Moreover, it explores the whole solution space, from the input mixture to one with only one element. This makes pointless the usage of such approach: Segmenting an image with the original EM instead of [5] will give the same result in less time.

5.3 FASTGMM Optimization Procedure

We reported our results both using the optimization procedure, and not. Since one of the most prominent key feature of our approach is its fast computation, together with its simple implementation, the optimization process may seem worthless or too much time demanding. However, by comparing its performances against those of [5], our algorithm still remain faster (see sec. 5.1.2). The difference in terms of final mixture precision is not so evident at a first glance, both referring to the final log-likelihood and to the normalized L2 distance, although present. Nevertheless, the required time for the EM optimization step is important, since sometimes it approaches (and overcome) the splitting part. Here, selecting whether optimizing or not is merely a question of performance requiring. If one claims for the fastest algorithm it is advisable to not use the optimization, even though it may lead to some improvements to the final mixture. Otherwise, FASTGMM gives a good precision maintaing a better computational burden than FIGJ.

5.4 Limitation of the proposed algorithm

The bigger issue with our approach is the α_{MAX} parameter tuning. This cause FASTGMM being less general than FIGJ in input domain. If α_{MAX} is too small the input description may be underestimated, or overestimated if it is too high. This does not mean that it cannot perform in general purposes, but only that it has to be tuned for getting precise results. However, this makes FASTGMM suitable for a first data description due to its great velocity. Once a first input analysis has been performed, it can be fine tuned to have a better data description. Moreover, we demonstrated that if well tuned, FASTGMM is able to segment the input data even better than FIGJ.

6 Conclusion

In this paper we proposed a unsupervised algorithm that learns a finite mixture model from multivariate data on-line. The algorithm can be applied to any data mixture where the EM can be used. We approached the problem from the opposite way of [5], i.e. by starting from only one mixture component instead of several ones and progressively adapting the mixture by adding new components when necessary. Our algorithm starts from a single mixture component and sequentially *growing* both increases the number of components and adapting their means and covariances. Therefore, its initialization is unique, and it is not affected by different possible starting points like the original EM formulation. Moreover, by starting with a single component the computational burden is low at the beginning, increasing only whether more components are required. Finally, we presented the effectivity of our technique in a series of simulated experiments with synthetic data, artificial, and real images, and we compared the results against [5].

Acknowledgements

This work was supported by the European Commission, Project IST-004370 RobotCub and FP7-231640 Handle, and by the Portuguese Government - Fundação para a Ciência e Tecnologia (ISR/IST pluriannual funding) through the PIDDAC program funds and through project BIO-LOOK, PTDC / EEA-ACR / 71032 / 2006.

References

1. Ahrendt, P.: The multivariate gaussian probability distribution. Tech. rep., http://www2.imm.dtu.dk/pubdb/p.php?3312 (January 2005)
2. Carpin, S., Lewis, M., Wang, J., Balakirsky, S., Scrapper, C.: Bridging the gap between simulation and reality in urban search and rescue". In: Robocup 2006: Robot Soccer World Cup X (2006)
3. Dempster, A., Laird, N., Rubin, D.: Maximum likelihood estimation from incomplete data via the em algorithm. J. Royal Statistic Soc. 30(B), 1–38 (1977)
4. Dobbe, J.G.G., Streekstra, G.J., Hardeman, M.R., Ince, C., Grimbergen, C.A.: Measurement of the distribution of red blood cell deformability using an automated rheoscope. Cytometry (Clinical Cytometry) 50, 313–325 (2002)
5. Figueiredo, A., Jain, A.: Unsupervised learning of finite mixture models. IEEE Trans. Patt. Anal. Mach. Intell. 24(3) (2002)
6. Greggio, N., Silvestri, G., Menegatti, E., Pagello, E.: Simulation of small humanoid robots for soccer domain. Journal of The Franklin Institute - Engineering and Applied Mathematics 346(5), 500–519 (2009)
7. Greggio, N., Bernardino, A., Santos-Victor, J.: A practical method for self-adapting gaussian expectation maximization. International Conference on Informatics in Control, Automation and Robotics (ICINCO 2010), Funchal, Madeira - Portugal (June 15-18 2010)
8. Hartley, H.: Maximum likelihood estimation from incomplete data. Biometrics 14, 174–194 (1958)
9. Jensen, J.H., Ellis, D., Christensen, M.G., Jensen, S.H.: Evaluation distance measures between gaussian mixture models of mfccs. Proc. Int. Conf. on Music Info. Retrieval ISMIR-07 Vienna, Austria pp. 107–108 (October, 2007)
10. Li, J., Barron, A.: Mixture density estimation. NIPS, MIT Press 11 (2000)
11. Montesano, L., Lopes, M., Bernardino, A., Santos-Victor, J.: earning object affordances: From sensory motor maps to imitation. IEEE Trans. on Robotics 24(1) (2008)
12. Shim, H., Kwon, D., Yun, I., Lee, S.: Robust segmentation of cerebral arterial segments by a sequential monte carlo method: Particle filtering. Computer Methods and Programs in Biomedicine 84(2-3), 135–145 (December 2006)
13. Ueda, N., Nakano, R., Ghahramani, Y., Hiton, G.: Smem algorithm for mixture models. Neural Comput 12(10), 2109–2128 (2000)
14. Verbeek, J., Vlassis, N., , Krose, B.: Efficient greedy learning of gaussian mixture models. Neural Computation 15(2), 469–485 (2003)
15. Vincze, M.: Robust tracking of ellipses at frame rate. Pattern Recognition 34, 487–498 (2001)
16. Zhang, Z., Chen, C., Sun, J., Chan, K.: Em algorithms for gaussian mixtures with split-and-merge operation. Pattern Recognition 36, 1973 – 1983 (2003)