Machine Vision And Applications manuscript No.
(will be inserted by the editor)

Fast Estimation of Gaussian Mixture Models for Image Segmentation

Nicola Greggio · Alexandre Bernardino · Cecilia Laschi · Paolo Dario · José Santos-Victor

Received: date / Accepted: date

Abstract *The Expectation-Maximization algorithm has been classically used to find the maximum likelihood estimates of parameters in probabilistic models with unobserved data, for instance, mixture models. A key issue in such problems is the choice of the model complexity. The higher the number of components in the mixture, the higher will be the data likelihood, but also the higher will be the computational burden and data overfitting. In this work we propose a clustering method based on the expectation maximization algorithm that adapts on-line the number of components of a finite Gaussian mixture model from multivariate data. Or method estimates the number of components and their means and covariances sequentially, without requiring any careful initialization. Our methodology starts from a single mixture component covering the whole data set and sequentially splits it incrementally during expectation maximization steps. The coarse to fine nature of the algorithm reduce the overall number of computations to achieve a solution, which makes the method particularly suited to image segmentation applications whenever computational time is an issue. We show the effectiveness of the method in a series of experiments and compare it with a state-of-the-art alternative technique both with synthetic data and real images, including experiments with images acquired from the iCub humanoid robot.*

N. Greggio*, C. Laschi, P. Dario
ARTS Lab - Scuola Superiore S.Anna, Polo S.Anna Valdera
Viale R. Piaggio, 34 - 56025 Pontedera, Italy
Tel.: +39-049-8687385
Fax: +39-049-8687385
E-mail: nicola.greggio@sssup.it

N. Greggio, Alexandre Bernardino, José Santos-Victor
Instituto de Sistemas e Robótica, Instituto Superior Técnico
1049-001 Lisboa, Portugal

Paolo Dario
CRIM Lab - Scuola Superiore S.Anna, Polo S.Anna Valdera
Viale R. Piaggio, 34 - 56025 Pontedera, Italy

Keywords Image Processing · Unsupervised Learning · Self-Adapting Gaussians Mixtures · Expectation Maximization · Machine Learning · Clustering

1 Introduction

Image segmentation is a key low level perceptual capability in many robotics related applications, as a support function for the detection and representation of objects and regions with similar photometric properties. Several applications in humanoid robots [27], rescue robots [3], or soccer robots [12], [14] rely on some sort on image segmentation. Additionally, many other fields of image analysis depend on the performance and limitations of existing image segmentation algorithms: video surveillance [6], medical imaging [8], face recognition [23], and database retrieval [31] are some examples. In this work we study and develop a fast method for fitting gaussian mixtures to a set of data, with applications to image segmentation.

Unsupervised image segmentation is a typical clustering problem: Image pixels must be grouped into ”classes” according to some similarity criteria (pixel color, proximity, etc.). The classes, also called *clusters*, are detected automatically, i.e. without any input by an external agent. This method finds applications in many fields, such as in image processing (e.g. face-color segmentation [11] or posture recognition [2]) , sound analysis [18], segmentation of multivariate medical images (a wide overview can be fond here [20]).

The techniques for unsupervised learning range from Kohonen maps [21] [22], Growing Neural gas [10], [15], k-means [25], to Independent component analysis [5], [16], etc. Particularly interesting is the Expectation Maximization algorithm applied to Gaussian mixtures which allows to model complex probability distribution functions. Fitting a mixture model to the distribution of the data is equivalent, in some applications, to the identification of the clusters with the mixture components [26].

1.1 Related Work

Expectation-Maximization (EM) algorithm is the standard approach for learning the parameters of the mixture model [13]. It is demonstrated that it always converges to a local optimum [7]. 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.

Different approaches can be used to select the best number of components. These can be divided into two main classes: *off-line* and *on-line* techniques.

The first ones evaluate the best model by executing independent runs of the EM algorithm for many different initializations and number of components, and evaluating each estimate with criteria that penalize complex models (e.g. the Akaike Information Criterion (AIC) [30] and the Rissanen Minimum Description Length (MDL) [29]). These, in order to be effective, have to be evaluated for every possible number of models under comparison. Therefore, it is clear that, for having a sufficient search range the complexity goes with the number of tested models as well as the model parameters.

The second ones start with a fixed set of models and sequentially adjust their configuration (including the number of components) based on different evaluation criteria. Pernkopf and Bouchaffra proposed a Genetic-Based EM Algorithm capable of learning gaussians mixture models [28]. They first selected the number of components by means of the minimum description length (MDL) criterion. A combination of genetic algorithms with the EM has been explored.

Ueda *et Al.* proposed a split-and-merge EM algorithm to alleviate the problem of local convergence of the EM method [32]. Subsequently, Zhang *et Al.* introduced another split-and-merge technique [34]. 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 [9].

A different approach has been explored by Ketchantang *et Al.*, by using Pearson mixture model (PMM) in conjunction with Gaussian copula instead of pure Gaussian mixture distributions [19] The model is combined with a Kalman filter in order to predict the object's position within the next frame. However, despite its validity in tracking multiple objects and its robustness against different illumination contexts, the required computational burden is higher than the corresponding Gaussian mixture based models.

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. The particularity of our model is that it starts from only one mixture component progressively adapting the mixture by splitting components when necessary.

Our formulation guarantees the following advantages. First, the initialization is very simple. Diversely from the standard EM algorithm, or any EM technique that starts with more than one component, where initialization is often random, in our case the initialization is deterministic: The initial component is a single gaussian that best fits the whole data set. Moreover, the splitting criterion is also deterministic, as it will be explained later. Hence, the whole algorithm is deterministic: By applying the same algorithm to the same input data we will always get the same results. Second, it is a technique with computational cost lower than other approaches. The algorithm complexity will be analyzed later in the paper and compared to the alternatives.

In a sense, we approach the problem in a different way to Figueiredo and Jain. 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 Real-Time Applications

The algorithm described in this paper was developed keeping in mind the need for real-time operation, having image segmentation for robots as our first objective. Due to the large resolution of modern frame grabbers and cameras, image segmentation requires fast techniques in order to satisfy the real-time demands of robotic applications, ranging from simple tracking to autonomous vehicle guidance and video surveillance. Since there are no efficient global solutions to the general problem of unsupervised estimation of mixture models, we have developed a greedy approach where the design choices are taken to specifically address the image segmentation problem achieving simultaneously fast performance and results competitive with the state-of-the-art. For evaluation purposes, we also perform experiments on 2D synthetic data, where the method's performance can be more easily visualized and compared. However, we stress that the main aspect considered in writing the algorithm's specification is a fast performance in image segmentation. Several results on this domain, both with generic images and with images taken in our robotic platform, are presented and compared with the state-of-the-art.

1.4 Outline

The paper is organized as follows. In sec. 2 we summarize the main results of the classical Expectation Maximization algorithm. 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. 5 we describe our experimental set-up for testing the validity of our new technique and in sec. 6 we discuss our results. Finally, in sec. 7 we draw the main conclusions of this work.

2 Expectation Maximization Algorithm

The Expectation-Maximization algorithm serves to find the maximum likelihood estimates of a probabilistic model with unobserved data. A common usage of the EM algorithm is to identify the *"incomplete, or unobserved data"*:

$$\mathscr{Y} = \left(\vec{y}^1, \vec{y}^2, \ldots, \vec{y}^j\right) \tag{1}$$

given the couple $(\mathscr{X}, \mathscr{Y})$ - with $\mathscr{X}$ defined as:

$$\mathscr{X} = \{\vec{x}^1, \vec{x}^2, \ldots, \vec{x}^N\} \tag{2}$$

also called *"complete data"*, which has a probability density (or joint distribution) $p\left(\mathscr{X}, \mathscr{Y} | \bar{\vartheta}\right) = p_{\bar{\vartheta}}\left(\mathscr{X}, \mathscr{Y}\right)$ depending on the parameter $\bar{\vartheta}$. More specifically, the *"complete data"* are the given input data set $\mathscr{X}$ to be classified, while the *"incomplete data"* are a series of auxiliary variables in the set $\mathscr{Y}$ 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 $p_{\bar{\vartheta}}\left(\mathscr{X}, \mathscr{Y}\right)$.

We define:

$$Q\left(\bar{\vartheta}^{(n)}, \bar{\vartheta}^{(n-1)}\right) = E^{'} L\left(\bar{\vartheta}^{(n-1)}\right) \tag{3}$$

with $L\left(\bar{\vartheta}^{(n-1)}\right)$ being the log-likelihood of the observed data at step $n-1$:

$$L\left(\bar{\vartheta}^{(n-1)}\right) = \log p\left(\mathscr{X}, \mathscr{Y} | \bar{\vartheta}^{(n-1)}\right) \tag{4}$$

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

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

$$Q\left(\bar{\vartheta}^{(n)}, \bar{\vartheta}^{(n-1)}\right) = E^{'}\left[\log p\left(\mathscr{X}, \mathscr{Y} | \bar{\vartheta}^{(n-1)}\right)\right] \tag{5}$$

- M-step: It evaluates the new parameters that maximize Q; this, according to the ML estimation, is:

$$\bar{\vartheta}^{n} = arg\max_{\bar{\vartheta}} Q\left(\bar{\vartheta}^{n}, \bar{\vartheta}^{(n-1)}\right) \tag{6}$$

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 $\bar{\vartheta}^0$.

2.1 **EM Algorithm: Application to a gaussians Mixture**

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

$$\begin{aligned} p\left(\bar{x}\right) &= \sum_{c=1}^{nc} w_c \cdot p_c\left(\bar{x}\right) \\ p_c\left(\bar{x}\right) &= \frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma_c|^{\frac{1}{2}}} e^{-\frac{1}{2}(\bar{x}-\bar{\mu}_c)^T |\Sigma_c|^{-1} (\bar{x}-\bar{\mu}_c)} \end{aligned} \tag{7}$$

where $p_c\left(\bar{x}\right)$ is the component prior distribution for the class c, and with d, $\bar{\mu}_c$ 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.

Let us consider nc classes C_{nc}, with $p\left(\bar{x}|C_c\right) = p_c\left(\bar{x}\right)$ and $P\left(C_c\right) = w_c$ being the density and the *a-priori* probability of the data of the class C_c, respectively. Then:

$$p\left(\bar{x}\right) = \sum_{c=1}^{nc} P\left(C_c\right) \cdot p\left(\bar{x}|C_c\right) = \sum_{c=1}^{nc} w_c \cdot p_c\left(\bar{x}\right) \tag{8}$$

In this setting, the unobserved data set $\mathscr{Y} = \left(\bar{y}^1, \bar{y}^2, \dots, \bar{y}^N\right)$ 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 $\bar{y}$ is the class C_c prior probability:

$$E'(y_c) = w_c \tag{9}$$

The algorithm main stages are:
- ***a)*** consider the whole data set $\bar{D} = (\mathscr{X}, \mathscr{Y})$, where $\vec{x^i}$ are known but $\vec{y^i}$ are unknown;
- ***b)*** E-step: for each data sample evaluate its class posterior probabilities $P\left(y_c^i = 1|\vec{x^i}\right), c = 1 \cdots nc$:

$$\begin{aligned} P\left(y_c^i = 1|\vec{x^i}\right) &= P\left(C_c|\vec{x^i}\right) \\ &= \frac{p\left(\vec{x^i}|C_c\right) \cdot P(C_c)}{p\left(\vec{x^i}\right)} = \frac{w_c \cdot p_c\left(\vec{x^i}\right)}{\sum_{c=1}^{nc} w_c \cdot p_c\left(\vec{x^i}\right)} \\ &\triangleq \pi_c^i \end{aligned} \tag{10}$$

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 $\vec{x^i}$ belongs to class C_c.
- ***c)*** M-step: re-estimate the parameter vector $\bar{\vartheta}$, which at the $n+1$ iteration will be $\bar{\vartheta}^{(n+1)}$. This, in case of a gaussians mixture distribution, i.e. $p_c(\bar{x})$ is a gaussians density, we have $\bar{\vartheta} = (w_c, \bar{\mu}_c, \Sigma_c)$. Then, we evaluate the means and the covariances by weighting each data sample by the degree in which it belongs to the class as:

$$\begin{aligned} \bar{\mu}_c^{(n+1)} &= \frac{\sum_{i=1}^N \pi_c^i \vec{x^i}}{\sum_{i=1}^i \pi_c^i} \\ \Sigma_c^{(n+1)} &= \frac{\sum_{i=1}^N \pi_c^i \left(\vec{x^i} - \bar{\mu}_c^{(n+1)}\right)\left(\vec{x^i} - \bar{\mu}_c^{(n+1)}\right)^T}{\sum_{i=1}^N \pi_c^i} \end{aligned} \tag{11}$$

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

$$w_c^{(n+1)} = \frac{1}{N}\sum_{i=1}^N \pi_c^i, \quad with \;\; c = \{1, 2, \dots, nc\} \tag{12}$$

2.2 EM Algorithm: Model Selection Criteria

Deciding the best number of components in a mixture is a critical issue. The more components there are, the higher the log-likelihood will be. However a high number of components increases the computational cost of the algorihtm and the risk of a data overfitting. On the other side, a low number of components may not allow for a good enough description of the data. Several information criteria have been developed in order to evaluate the best mixture complexity (number of components) subject to the specific input data. They evaluate the best model by executing independent runs of the EM algorithm for many different initializations and number of components. Then, after each run they produce a number, the value of the information criterion itself, characterizing the overall description. Each criterion has its own characteristics. For a detailed overview about the most used ones, refer to [24]. A common feature is that all of them tend to penalize complex models. Some examples are the Akaike Information Criterion (AIC) [30], the Rissanen Minimum Description Length (MDL) [29], and the Wallace Minimum Message Length [33]. However, in order to be effective, they have to be evaluated for every possible number of models under comparison. Therefore, it is clear that, for wide enough search range, the complexity goes with the number of tested models as well as the model parameters. The following section describes our approach to perform an efficient exploration of the model space, not only avoiding exaustive search but also reducing as most as possible the model computational costs.

3 FASTGMM: Fast Gaussian Mixture Modeling

In this section we describe the rational of our algorithm. It has been mainly designed to perform an efficient search for the number of mixture components. Whereas the classical approach is to perform an exhaustive search of the number of components, doing independent EM runs with different initializations, a new class of algorithms has been developed in the last decade in the attempt to speed up the computations. The basic idea is to incrementally estimate the mixture parameters and the number of components simultaneously. The number of components is incremented or decremented at certain stages of the optimization procedure but the values of the mixture parameters are incrementally changed and not reinitialized. In this class of algorithms we refer to Figueiredo and Jain [9] (only decrement the number of components) and to Ueda [32] (increment/decrement). 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.

We distinguish two main features in our algorithm: The splitting and the stopping criteria. 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 (13):

$$\Lambda_{curr(c)}(\vartheta) = \sum_{i=1}^k log\left(w_c \cdot p_c(\bar{x}_i)\right) \tag{13}$$

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. To each Gaussian component we also associate a *age* variable in order to control ow long the component's own likelihood does not increase significantly (see sec. 3.1). 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:

$$\bar{\vartheta}_c = \rho\left(w_c, \bar{\mu}_c, \Sigma_c, \xi_c, \Lambda_{last\,(c)}, \Lambda_{curr\,(c)}, a_c\right) \tag{14}$$

where each element is described in tab. 3.1. In the rest of the paper the index notation described in tab. 3.1 will be used.

Two important elements are the area (namely, the covariance matrix determinant) and the age of a Gaussian component, respectivelly ξ_c and a_c.

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
k	total number input points
d	single data dimension
D	input dimensionality

Table 1 Symbol notation used in this paper

During each iteration, the algorithm keeps in memory the previous likelihood ($\Lambda_{last\,(c)}$). Once the re-estimation of the vector parameter $\bar{\vartheta}$ has been computed in the EM step, a key decision is taken: If a component's own likelihood does not increase more than Λ_{TH} for a predetermined number of times and its area exceeds ξ_{TH}, this component will be split in two. Both thresholds Λ_{TH} and ξ_{TH} are time varying, like an annealing schedule, to promote gradual splits. If the splits do not improve the whole log-likelihood significantly, the algorithm stops.

The whole algorithm pseudocode is shown in Algorithm 3.1.

Algorithm 3.1 FASTGMM: Pseudocode

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

3.2 **Parameters initialization**

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

$$\begin{aligned} \xi_{TH-INIT} &= \xi_{data} \\ L_{TH-INIT} &= k_{LTH} \\ A_{TH-INIT} &= k_{ATH} \end{aligned} \tag{15}$$

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:

$$\begin{aligned} \mu_{data,d} &= \frac{1}{k}\sum_{i}^{k} x_d^i \\ \Sigma_{data,i} &= \langle \bar{x_i} - \bar{\mu}_{data}\rangle \langle \bar{x_i} - \bar{\mu}_{data}\rangle^T \end{aligned} \tag{16}$$

where k is the number of input data vectors $\bar{x}$.

3.3 Gaussian components initialization

The algorithm starts with a single 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.

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 be approximating two components with diverse configurations: Either covering two smaller data distribution sets, placed along the longer axis, or two overlapped sets of data with different covariances, etc. So, splitting a component is a ill posed problem and the best way depends on the problem at hand. In this paper we make a choice suited to applications in color image segmentation whose purpose is to obtain components with lower overlap. For this case 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. In fact, in image segmentation applications, points are composed of the R,G,B color values (or other color space) and spatial (x,y) coordinates. Whereas the color components can overlap (there may be many pixels with the same color), spatial components are intrinsically non-overlapping ((x,y) coordinates are unique). Thus, in the context of image segmentation applications it makes sense to favor non-overlapping components, thus empirically justifying the proposed rule.

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 biggest 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 $\bar{\vartheta}_{OLD}$ will be split in two new gaussians A and B, with means:

$$\begin{aligned} \bar{\Sigma}_{OLD} &= USV^T \\ \bar{u}_{MAX} &= U_{*,1}; \quad s_{MAX} = S_{1,1} \\ \bar{\mu}_A &= \bar{\mu}_{OLD} + \frac{1}{2} s_{MAX} \bar{u}_{MAX} \\ \bar{\mu}_B &= \bar{\mu}_{OLD} - \frac{1}{2} s_{MAX} \bar{u}_{MAX} \end{aligned} \tag{17}$$

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

The covariance matrices will then be updated as:

$$\begin{aligned} S_{1,1} &= \frac{1}{4} s_{MAX} \\ \Sigma_A = \Sigma_B &= USV^T \end{aligned} \tag{18}$$

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

$$w_A = \frac{1}{2} w_{OLD} \qquad w_B = \frac{1}{2} w_{OLD} \tag{19}$$

It is worth noticing that the first split occurs immediately after the initialization. Otherwise the algorithm would have to wait until the first Gaussian gets old before adding the new components, which is clearly unnecessary.

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:

$$\begin{aligned}\Lambda_{TH} &= \Lambda_{TH} - \frac{\lambda}{nc^2}\cdot\Lambda_{TH}\\ &= \Lambda_{TH}\cdot\left(1-\frac{\lambda}{nc^2}\right)\\ \xi_{TH} &= \xi_{TH} - \frac{\alpha_{MAX}}{nc^2}\cdot\xi_{TH}\\ &= \xi_{TH}\cdot\left(1-\frac{\alpha_{MAX}}{nc^2}\right)\end{aligned} \qquad (20)$$

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 for α_{MAX} the corresponding thresholds Λ_{TH} and ξ_{TH} decrease. Therefore, since a component splits when its covariance matrix area overcomes the threshold ξ_{TH}, decreasing α_{MAX} will increase the split frequency and promote a faster convergence. In an analogous form, λ controls the minimum increase in log-likelihood a component must have to be split. Therefore, the higher λ is, the easier the system will promote a split. In tandem, α_{MAX} and λ control the convergence speed together. However, fast convergence is often associated to instability around the optimal point, and may even lead to a divergence from the local optimum.

Notice that Λ_{TH} and ξ_{TH} decrements are inversely proportional to the square of the number of components. The rationale is to promote splits in models with low complexity and prevent the fast growing of the number of components in models with high complexity.

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:

$$\begin{aligned}\xi_{TH} &= \frac{\xi_{TH-INIT}}{nc}\\ \Lambda_{TH} &= L_{TH-INIT}\end{aligned} \qquad (21)$$

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.

All these rules, although empirical, have proven successful in the estimation of mixture components, as will be shown in the results. The sole parameters to cotrol the process are α_{MAX} and λ. A practical rule to tune these parameters is given in the following section.

3.5.1 ***Tuning α_{MAX} and λ***

Tuning paramenters α_{MAX} and λ depend mostly on the dimensionality and type of data and the application requirements for precision *vs* computation time. According to our experience, these parameters can be tuned only once for each specific type of data. Particular instantiations of data with the same characteristics do not require retuning. For instance, the 2D input data of sec. 5 uses the same paramenter and the same happens for the image data sets. Therefore we propose a methodology to tune the parameters depending on the demands of the application: focussing more on the stability or on the accuracy. If precision is preferred, starting with low values assures a low frequency of split and a better convergence to a local optimum. Then, α_{MAX} and λ should be gradually increased until a satisfactory performance. The best way is to use a test input set, and comparing the output of our algorithm with "ground truth" for different runs, increasing α_{MAX} and λ each time. In the second case, it is better to follow the opposite way: starting with higher values of α_{MAX} and λ than before, and then reducing them at each run. It is not possible to establish a priori the best starting values, because they depend on the type of data (dimensionality, scale). In our experiments we used $\lambda = 20$ for all the tests, while $\alpha_{MAX} = 1.5$ for the 2D input data, and $\alpha_{MAX} = 1$ for the images.

3.6 **Ill-Conditioned components**

We experienced that during the EM steps, the computation sometimes leads to a ill-conditioned component. This happens when a Gaussian becomes too "enlongated", which can be tested through its conditioning number (i.e. the ratio between the biggest and smallest eigenvalue) becoming too large. It may happen in several situations with different kinds of input data, from the classical 2-dimensional points, to 3 dimensional (e.g. xyz cartesian points, or RGB color images), and this may "crash" the algorithm. However, this problem happens more frequently with images than with simple input points. Many images we tried with the Figueiredo and Jain's algorithm cannot have been segmented due to this problem.

To address this problem we evaluate the class' covariance matrix conditioning number, and if it is higher than a limit (e.g. $10e^{+10}$) we stop the EM computation. In fact, since that component does not contribute enough to the input data description, we directly reject this component.

3.7 **Optimizing the selected mixture**

After FASTGMM algorithm stops, we may keep the chosen mixture as the final result or we may perform an aditional EM step to refine the solution. This is an optional procedure. The former choice is the fastest but less accurate, while the latter one introduces new computations but ensures more precision.

Why to choose the second possibility?
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.

3.8 **Stopping criterion**

The algorithm stops when the log-likelihood does not increase over a minimum value. This is a common used approach [9] [32]. However, rather than consider the absolute value of the log-likelihood, as in the two previous techniques, we considered a percentage variation. In fact, describing different input data sets with their *best* mixture give rise to different values for the final log-likelihood. Therefore, instead of fixating an absolute value as the minimum increment, a percentage amount allows a more general approach. In this work, we stop the EM computation when the log-likelihood does not increase more than 0.1%.

3.9 **Computational complexity evaluation**

We refer to the pseudocode in algorithm 3.1, 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 8 to 27);
- our split (step 15) operation requires $O(D)$.
- the others take $O(1)$.
- the optional procedure of optimizing the selected mixture (step 29) 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 Application to Color Image Segmentation

This section mainly focuses on the application of our approach for segmenting real colored images. Since the algorithm is suitable for this kind of task, it is relevant to dedicate a brief section about it.

Each pixel of the image is represented as a 5-dimensional vector, where the dimensions are the (R,G,B) color space and the (x,y) pixel coordinates. The input data set is composed by all the pixels of the image. After the Gaussian mixture estimation process, we will have several components, each one representing a Gaussian distribution in the 5D joint (R,G,B,x,y) space, represented by a mean color, a mean spacial coordinate and associated covariances. Each pixel vector is assigned a likelihood of belonging to each of these distributions. We choose the maximum likelihood assignment. In the experimental results we show segmentation results (see Fig. 2). In those images, each pixel is colored with the mean RGB value of its maximum likelihood cluster.

5 Experimental Validation

The experiments performed in this paper aim at a careful comparison with a state-of-the-art unsupervised learning technique [9] which is based on similar principles (mixture model estimation) and makes publicly available the corresponding source code. We compare both approaches in synthetic data (artificially generated with a known mixture), and to some real images (taken by a webcam or by our robotic platform). The comparison criteria include the computational cost (both algorithms have MATLAB implementations), the number of components, and robustness to operating conditions. In the experiments performed with real images we also show the output of a successful image segmentation algorithm based on the *meanshift algorithm* [4], using the EDISON binary package [1]. Although based on different principles, this method is one the best with respect to image segmentation. However it is a non adaptive method: it relies on two main parameters, the spatial and the color bandwidth,

[1] http://coewww.rutgers.edu/riul/research/code/EDISON/

that rougly define the neighborhood in space and color to define clusters [4], and that we have manually tuned to achive a number of clusters similar to ours. The available implementation is coded in C and therefore we cannot compare the performance in terms of the computation time. We used it as an example of one of the best quality image segmentation methods currently available.

5.1 **Synthetic data**

In order to evaluate our algorithm's performance with ground truth data, we tested it by classifying different input data sets randomly generated by a known gaussians mixture. The same input sets have been proposed in [9]. Each distribution has a total of 2000 points, but arranged with different mixture distributions. Even though our procedure can be applied to any input dimensionality, we choose to show the results for 2-dimensional input because they are easier to represent. As the ratio *(# gaussians)/(# data points)* increases, it becomes harder to reach a good solution in a reasonable number of steps. Therefore, we are interested in evaluating how our algorithm behaves when the model complexity gradually increases.

Since [9] depends on the initialization, to have a fair comparison, we adopted a commonly used approach: select 10 different initial random conditions and report those giving the highest likelihood. The output of the two algorithms is shown in Fig. 2. 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). Here we show the representation for different mixtures of 3, 4, 5, 8, 12, 14, and 16 gaussian components. The data plots show the generated mixture (blue) and the evaluated one (red). The data on the left results from our approach, while on the right we show the results of [9], relative to the same input data set. The 3D plots at the bottom in each subfigure represent, respectivelly, the generated mixture, our algorithm's estimation, and the estimation given by the algorithm in [9].

We can see that our algorithm is capable of identifying the input data mixture starting from only one component with an accuracy comparable with that of [9].

Here we performa a detailed quantitative comparison with [9]. In table 2 we show a detaile quantitative comparison with [9] (from now on denoted FIGJ). The table contains:

- The number of initial mixture components;
- The number of detected components;
- The actual number of components, i.e. that of the generation mixture;
- The number of total iterations;
- The elapsed time;
- The percentage difference in time for our algorithm with the optimization process;
- The final log-likelihood;
- The percentage difference in final log-likelihood for our algorithm with the optimization process;
- The normalized L2 distance to the generation mixture without optimization;
- The normalized L2 distance to the generation mixture with optimization (only for FASTGMM).

In the following we discuss the main differences between the two algorithms.

5.1.1 Evaluated number of components

There are substantially no differences in the selected number of components. Both our approach and [9] perform well on low number of mixture components, while having the tendency of underestimating them when the number increases. An exception is our approach that overestimates the 8-components case but correctly estimates the 16-components case. However, it is worth considering that even though it estimates well the number of components, the parameters may differ. For instance, two components may be regarded as only one, while a single one can be considered as a multiple one. This behavior is present in both algorithm (see Fig. 2), suggesting that a perfect algorithm is hard to find.

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, because most iterations are performed with lower number of components. The elapsed time of the algorithm in [9] strongly depends on the initial number of components. The more they are, the slower it is. [9], on the other side, starts in the opposite way, i.e. with the maximum allowed components. The rule to define this number relies on a desired minimum probability of successful initialization of $1-\varepsilon$, with the limit of having at least the following amount of starting components c_{min} (sec. 6.1 of [9]):

$$c_{min} > \frac{\ln \varepsilon}{\ln(1-\alpha_{min})} \tag{22}$$

where $\alpha_{min} = min\{\alpha_1, \alpha_2, \cdots, \alpha_c\}$ is the probability of the component that will be more probability left out of the initialization, for its excessively small probability. Then, claiming a probability of successful initialization of 90% $\Longrightarrow 1-$

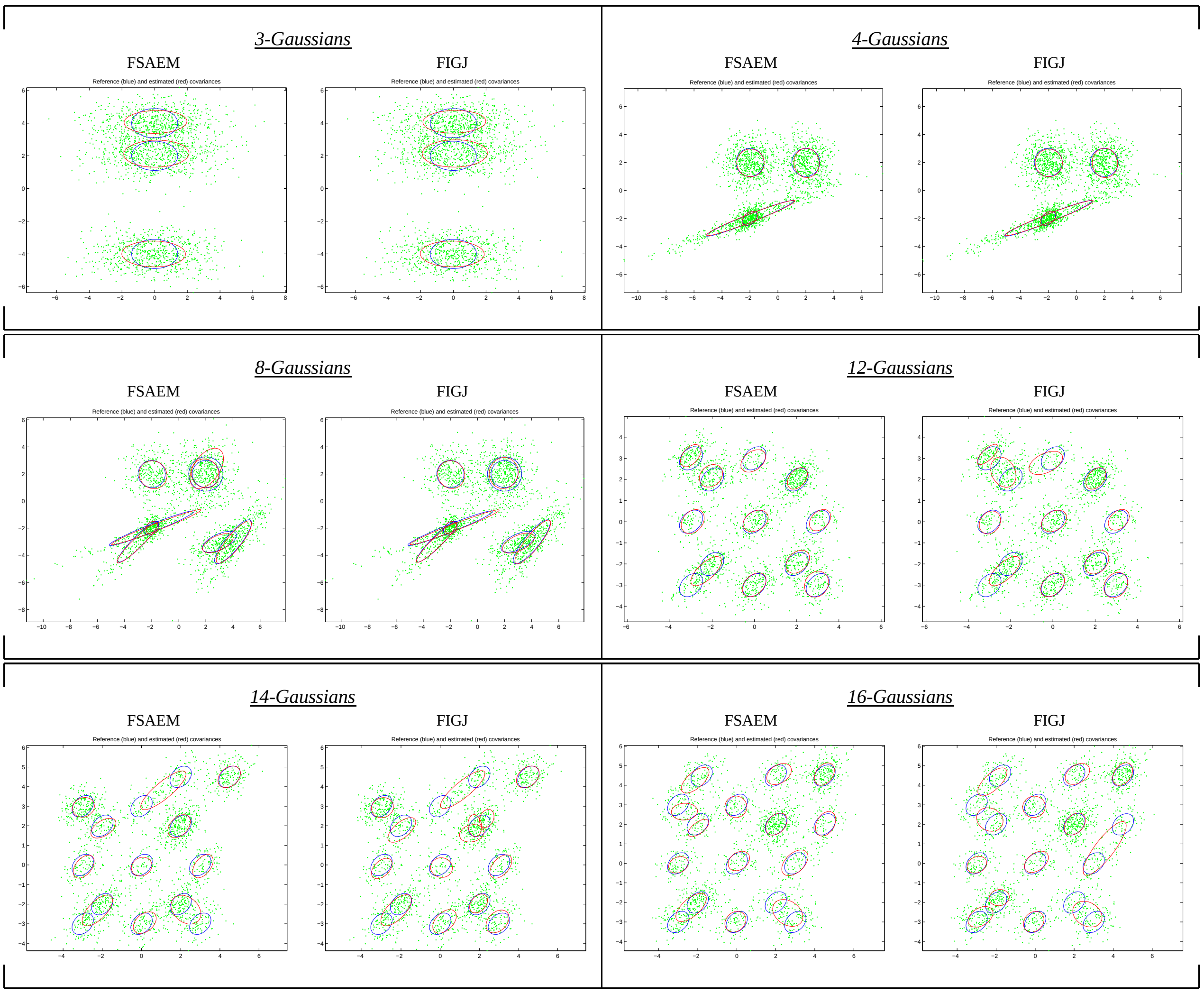

Fig. 1 For each plot set: Generation mixture (blue) and the evaluated one (red) for FASTGMM and FIGJ on the same input sets.

Input	Algorithm	# Initial 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
3-gau:	FASTGMM	1	3	3	76	3.99716	-53.89844289	-8420.917867	0.495867477	5.770135	3.918034	no
	Optimization		3		130	6.151567		-8379.161274				
	FASTGMM + Opt.		3		206	10.148727		-8379.161274				
	FIGJ	16	3		277	29.433288		-9524.692099		3.670464	3.670464	no
4-gau:	FASTGMM	1	4	4	101	5.615204	-123.166389	-7573.101881	2.218687212	10.670613	0.07519	no
	Optimization		3		186	12.531248		-7405.078438				
	FASTGMM + Opt.		4		287	18.146452		-7405.078438				
	FIGJ	16	4		205	13.52505		-8729.761818		0.076403	0.076403	no
8-gau:	FASTGMM	1	9	8	276	5.750431	22.99575458	-8599.51	0.015582283	0.196817	1.971166	no
	Optimization		8		199	4.428076		-8598.17				
	FASTGMM + Opt.		9		475	10.178507		-8598.17				
	FIGJ	16	7		333	48.156629		-9798.154848		0.14491	0.14491	no
12-gau:	FASTGMM	1	11	12	376	18.496127	22.99575458	-7536.534442	0.049105156	1.069315	0.284287	no
	Optimization		12		117	5.593468		-7532.833615				
	FASTGMM + Opt.		11		493	24.089595		-7532.833615				
	FIGJ	16	11		340	12.882045		-8922.220193		1.116857	1.116857	no
14-gau:	FASTGMM	1	12	14	351	17.651956	16.07843346	-8008.490211	0.157717218	5.906032	1.884532	no
	Optimization		14		280	14.813798		-7995.859443				
	FASTGMM + Opt.		12		631	32.465753		-7995.859443				
	FIGJ	20	12		419	30.707378		-9465.594783		3.639617	3.639617	no
16-gau:	FASTGMM	1	16	16	501	26.667825	51.82061529	-8165.436422	0.057038433	0.251515	1.033934	no
	Optimization		16		202	12.848394		-8160.778985				
	FASTGMM + Opt.		16		703	39.516219		-8160.778985				
	FIGJ	20	14		363	63.740854		-9540.91802		2.98916	2.98916	no

Table 2 Experimental results on synthetic data.

$\varepsilon = 0.9 \Longrightarrow \varepsilon = 0.1$ we get $c_{min} = 26,4 \cong 27$ as the minimum number of component, i.e. more than double the actual necessary components.

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. This is 16 for the 3 to 12 mixture components, and 20 for the 14 and 16 ones (see tab. 2). 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.

Finally, FASTGMM stops when the best amount of components to cover the whole data distribution is found. FIGJ, instead, once the optimum has been found, continues to try all the remaining configurations, to that with just one component (which is our starting mixture), so that exploiting all the possible solutions. That contributes to slow the FIGJ 's performance. Generally, our approach performs faster than [9].

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. A deterministic approach is to adopt a unique distance measure between the generated mixture and the evaluated one. In [17] Jensen *et Al.* exposed three different strategies for computing such distance: The Kullback-Leibler, the Earth Mover, and the Normalized L2 distance. The first one is not symmetric, and can onlyy be evaluated in closed form for unidimensional gaussians. The second one suffers from analogous problems. The third choice is symmetric, obeys to the triangle inequality and it is easy to compute. We ue the latter to perform the comparison. Its expression states [1]:

$$\begin{gathered}
z_c N_x\left(\bar{\mu}_c, \bar{\Sigma}_c\right) = N_x\left(\bar{\mu}_a, \bar{\Sigma}_a\right) \cdot N_x\left(\bar{\mu}_b, \bar{\Sigma}_b\right) \\
\textit{where} \\
\bar{\Sigma}_c = \left(\bar{\Sigma}_a^{-1} + \bar{\Sigma}_b^{-1}\right)^{-1} \quad \textit{and} \quad \bar{\mu}_c = \bar{\Sigma}_c\left(\bar{\Sigma}_a^{-1}\bar{\mu}_a + \bar{\Sigma}_b^{-1}\bar{\mu}_b\right) \\
\begin{aligned}
z_c &= |2\pi\bar{\Sigma}_a\bar{\Sigma}_b\bar{\Sigma}_c^{-1}|^{\frac{1}{2}} e^{-\frac{1}{2}(\bar{\mu}_a-\bar{\mu}_b)^T\bar{\Sigma}_a^{-1}\bar{\Sigma}_c\bar{\Sigma}_b^{-1}(\bar{\mu}_a-\bar{\mu}_b)} \\
&= |2\pi\left(\bar{\Sigma}_a+\bar{\Sigma}_b\right)|^{\frac{1}{2}} e^{-\frac{1}{2}(\bar{\mu}_a-\bar{\mu}_b)^T\left(\bar{\Sigma}_a+\bar{\Sigma}_b\right)^{-1}(\bar{\mu}_a-\bar{\mu}_b)}
\end{aligned}
\end{gathered} \tag{23}$$

5.2 **Colored real images**

Here we evaluate the performance of our algorithm when applied to the image segmentation problem. We perform a detailed comparison with [9] (to our knowledge this is the first time the algorithm of [9] is applied to image segmentation) and a qualitative comparison with an implementation of the *meanshift* algorithm [4]. Given the different principle and implementation of this algorithm it is difficult to perform a quantitative comparison in terms of computation time or model complexity. Anyway, since this is one the most successfull image segmentation algorithms, it provides a quality criteria benchmark to take into account. We segmented the images as 5-dimensional input in the (R,G,B,x,y) space. The color image segmentation results are shown in Fig. 2. The set of images is divided into two groups: Some general images (from (1) to (3)) and some images taken by the iCub's cameras (from (4) to (6)). For each group we show the original images, the segmentation provided by the EDISON implementation of the *meanshift* algorithm, those obtained with the [9], and those obtained with our algorithm, from left to right, respectively.

From a visual inspection of the figures it is hard to say what is the best segmentation. The *meanshift* output is probably more visually appealing. This method has been specially designed for image segmentation and contains a few post-processing steps that improve the overall quality of the results. Using raw mixture model estimation, as our method and FIGJ, there are no clear qualitative advantages of one over the other.

Table 3 shows quantitative results of the image segmentation runs using FASTGMM, FIGJ and FASTGMM with the final optimization step. If computational speed if an important requirement in the application at hand, we may say that our algorithm is better for image segmentation, because it is fast, its initialization is simple and produces comparable results. On the negative side on the α_{MAX} parameter (see sec. 3.5). [9] instead, it requires the *a-priori* definition of the maximum number of mixture components, that is the starting number of components. The higher it is the longer it will take for the input segmentation. If it is set too small, the solution space may not be appropriatelly explored, giving rise to undersegmentation. In addition, some problems arise with the evaluation of the covariances matrix for some real images. It happens that some of them may become ill-conditioned, making FIGJ “crash” when too many components are computed.

6 **Discussion**

In this section we highlight other aspects of the proposed algorithm and provide an overall comparison of the advantages and disadvantages with respect to FIGJ.

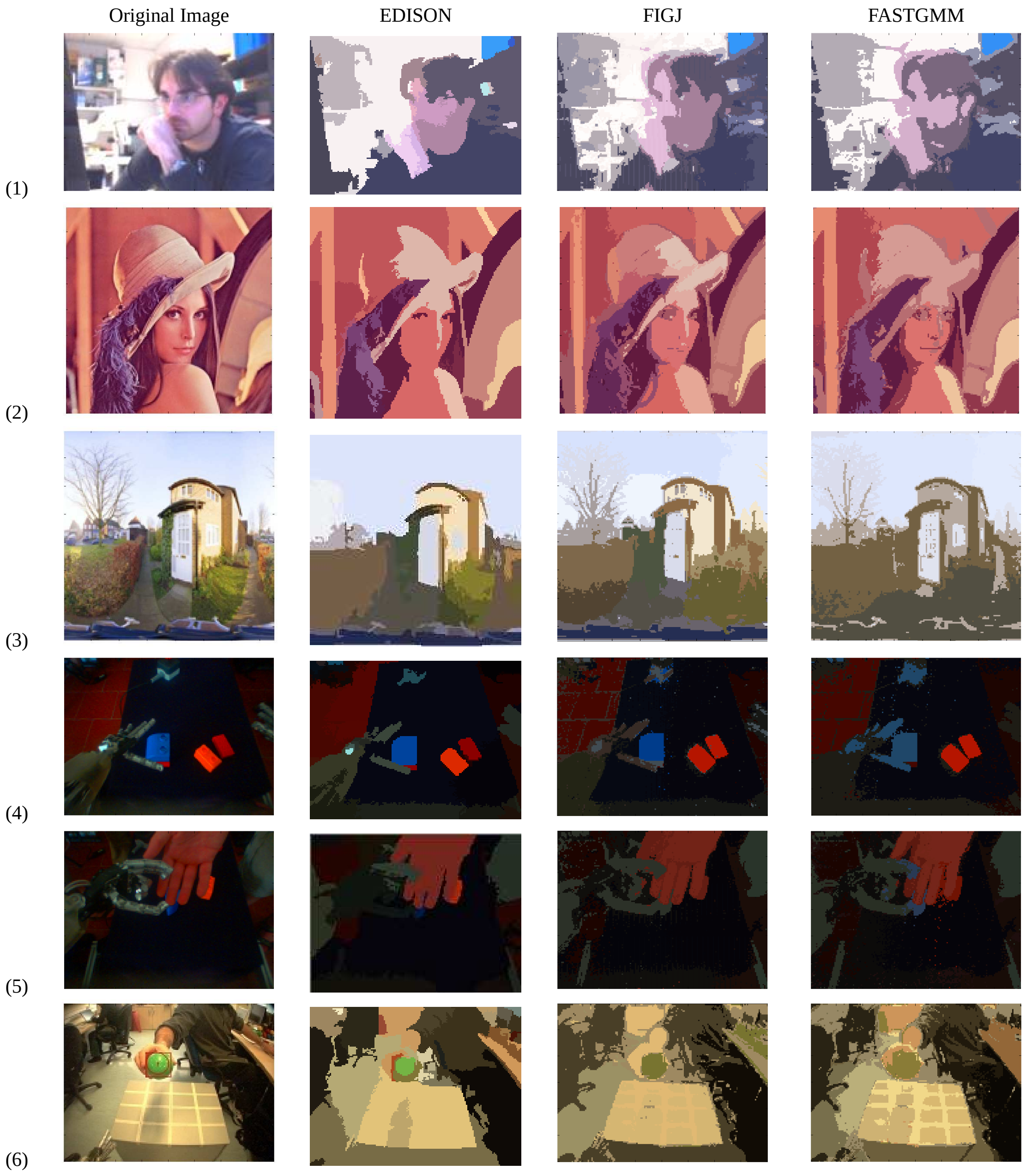

Fig. 2 Color image segmentation results. We divide these images into two groups: Some general images, on the top (lines (1) to (3)), and some images taken by the iCub's cameras, on the bottom (lines (4) to (6)). For each group we show the original images, those obtained with EDISON, those obtained with FIGJ, and those obtained with our algorithm, from left to right, respectively.

6.1 **FASTGMM Optimization Procedure**

We reported our results with and without the optimization procedure. 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 computational demanding. However, by comparing its performance against those of [9], our algorithm still remains faster (see sec. 5.1.2). The difference in terms of final mixture precision is not so evident at a first glance, both in the final log-likelihood and the normalized L2 distance, but allows some improvements in the final segmentation. 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 and a better computational cost.

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
1	FASTGMM	1	9	551	71.460507	130.9699636	-235130.6216	0.186374662	17.03330409	no
	Optimization		9	23	22.131293		-234692.3977			
	FASTGMM + Opt.		9	700	93.5918		-234692.3977			
	FIGJ	16	16	422	307.454885		-274668.2675			yes, 17
2	FASTGMM	1	14	426	80.365553	109.5710584	-314931.44	0.120630946	11.16531396	no
	Optimization		14	374	88.057387		-314551.5352			
	FASTGMM + Opt.		14	800	168.422941		-314551.5352			
	FIGJ	30	25	572	1611.816189		-349672.2017			no
3	FASTGMM	1	7	276	34.977055	16.7543208	-226138.1494	3.34203E-05	17.57803149	no
	Optimization		7	36	5.860168		-226138.0738			
	FASTGMM + Opt.		7	312	40.837223		-226138.0738			
	FIGJ	16	16	420	272.227272		-265888.6956			yes, 17
4	FASTGMM	1	7	576	72.744922	3.588007146	-281176.5478	1.30736E-06	15.72288044	no
	Optimization		7	14	2.610093		-281176.5441			
	FASTGMM + Opt.		7	590	75.355015		-281176.5441			
	FIGJ	11	11	267	106.453566		-325385.5959			yes, 12
5	FASTGMM	1	3	151	16.899695	13.4646217	-218447.7912	2.96364E-06	16.0195988	no
	Optimization		3	23	2.27548		-218447.7848			
	FASTGMM + Opt.		3	174	19.943671		-218447.7848			
	FIGJ	12	12	260	130.416222		-253442.2435			yes, 13
6	FASTGMM	1	11	451	67.534808	47.35502469	-210899.0185	0.114549212	17.68706256	no
	Optimization		10	180	31.981125		-210657.4353			
	FASTGMM + Opt.		10	631	99.515933		-210657.4353			
	FIGJ	24	22	514	624.913104		-247916.5477			no

Table 3 Experimental results on real images segmentation.

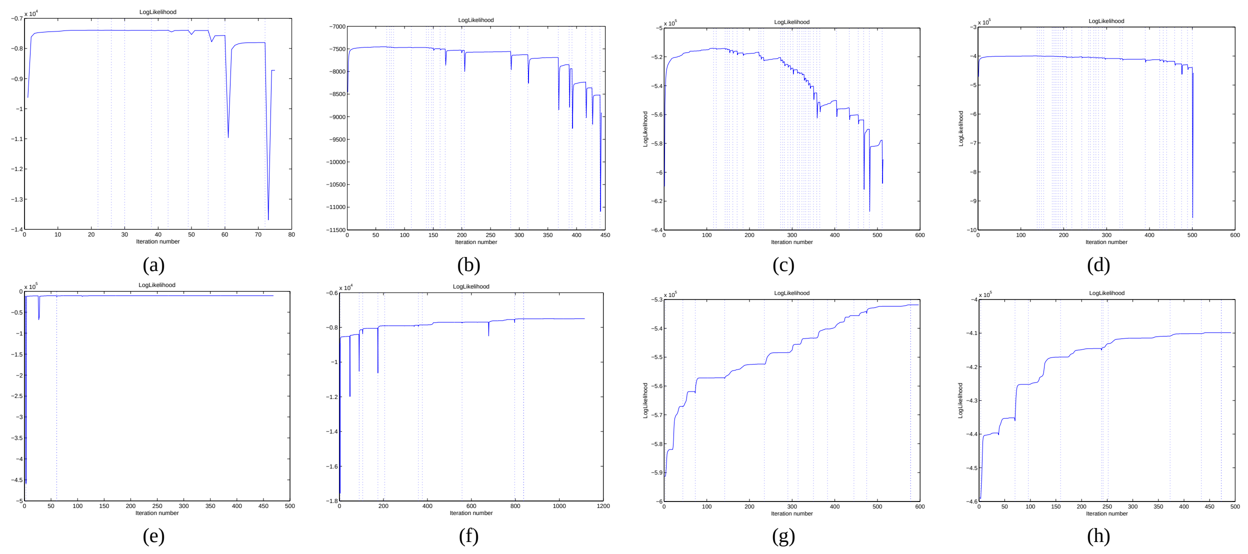

Fig. 3 The final log-likelihood evolution as function of the number of iterations of two different kinds of input data used within the experiments: the 4 mixture components (a) - FIGJ and (e) FASTGMM, the 12 mixture components - (b) FIGJ and (f) FASTGMM, the image (4) - (c) FIGJ and (g) FASTGMM, and the image (5) - (d) FIGJ and (h) FASTGMM.

6.2 **Log-Likelihood**

We approach the problem conversely than in [9]. Instead of starting with the maximum number of mixture components, we start with just only one. This will generate two opposite evolutions of the cost functions. FIGJ is characterized by a decreasing exponential curve, that raises at the end. Our approach, instead, is characterized by an increasing exponential curve. Fig. 3 shows four examples of these output for each algorithm.

Here it is possible to notice spikes in FIGJ curve, corresponding to component annihilation. When the log-likelihood increases, it means that the current mixture description is worse than the previous one.

Similarly, when our algorithm splits a gaussian, i.e. adding a component, a spike that decreases the log-likelihood appears on the curve. Finally, when the log-likelihood does not increase significantly the computation stops.

6.3 **Overall comparison: Which algorithm is the best one?**

There’s not a unique answer to this question. Both approaches has advantages and disadvantages. These are summarized in tab. 4.

Generally, our algorithm is faster than the [9]. However, both techniques rely on some *a-priori* parameters: FASTGMM α_{MAX} and λ, and FIGJ the maximum number of com-

Algorithm	Pro	Contro	Application Domain
FASTGMM	1. Deterministic. Independent runs of the algorithm will produce the same result. 2. Initialization is unique. No parameters required. 3. Low computational burden.	1. Requires tuning parameters for controlling the convergence.	1. High dimensional data (e.g. Image Segmentation, Robotics) 2. On-line and real-time applications 3. Applications requiring a low number of components (e.g. low bandwidth coding).
FIGJ	1. Does not require parameter tuning for controlling convergence. 2. Better results on average in synthetic data generated by gaussian mixtures	1. Stochastic. Different runs will lead to different results. 2. Requires the specification of the initial number of components. 3. High computational cost. 4. Showed unreliable performance in image segmentation.	1. Low-dimensional data. 2. Off-line applications. 3. Applications requiring a high number of components (e.g. high quality coding).

Table 4 Comparison between FASTGMM and FIGJ: Advantages and drawbacks of both algorithms.

ponents. These greatly affect the algorithms, the first for the precision most, and the second for the computational burden most.

Our approach is better when the time computation is a strict requirement. If the computation time is not a problem, FIGJ can be applied to synthetic data with more precise results than FASTGMM.

FASTGMM performed especially well on image segmentation. FIGJ, on the other side, did well in synthetic data, but not for real images. This is because of three main points: its low robustness to numerical errors produced crashes due to singular matrices when a certain number of components is analyzed (usually from 18, 20) as reported in tab. 3; higher computation time; and inaccurate selection of number of components.

7 Conclusion

In this paper we proposed a unsupervised algorithm that estimates on-line the parameters and number of components of a finite mixture model from multivariate data. The main focus was on computational speed. Our methodology starts with a single mixture component and incrementally adds new components until a good compromise between precision and speed is achieved. This promotes a faster computation that alternative methods that start with many components and gradually refine them. From this class is the algorithm of [9], for which we performed a detailed comparison in the application to simple 2D mixtures or to more complex image segmentation scenarios. Our methos has several advantages: is fast, its initialization is unique, a rough tuning of design parameters is sufficient for good performance. We describe in detail the operation of the algorithm and its variants, provide rules for tuning the important parameters and show, through simulated and real data sets, the advantages of our algorithm in what regards computation time and fitting score to ground truth.

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 (2005)
2. Bueno, J.I., Kragic, D.: Integration of tracking and adaptive gaussian mixture models for posture recognition. The 15th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN06), Hatfield, UK (2006)
3. 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)
4. Comaniciu, D., Meer, P.: Mean shift: A robust approach toward feature space analysis. IEEE Trans. Pattern Anal. Machine Intell **24**, 603–619 (2002)
5. Comon, P.: Independent component analysis: a new concept? Signal Processing, Elsevier **36**(3), 287–314 (1994)
6. Davis, J.W., Morison, A.W., Woods, D.D.: An adaptive focus-of-attention model for video surveillance and monitoring. Machine Vision and Applications **18**(1), 41–64 (2006)
7. 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)
8. 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)
9. Figueiredo, A., Jain, A.: Unsupervised learning of finite mixture models. IEEE Trans. Patt. Anal. Mach. Intell. **24**(3) (2002)
10. Fritzke, B.: A growing neural gas network learns topologies. Adv ances in Neural Inform ation Processing Systems 7 (NIPS'94), MIT Press, Cambridge MA pp. 625–632 (1995)
11. Greenspan, H., Goldberger, J., Eshet, I.: Mixture model for face-color modeling and segmentation. Pattern Recognition Letters (22), 1525–1536 (2001)

12. 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)
13. Hartley, H.: Maximum likelihood estimation from incomplete data. Biometrics **14**, 174–194 (1958)
14. Henderson, N., King, R., Middleton, R.H.: An application of gaussian mixtures: Colour segmenting for the four legged league using hsi colour space. Proceedings of the 2008 Australasian Conference on Robotics & Automation - Canberra, Australia **(ISBN 978-0-646-50643-2)** (2008)
15. Holmström, J.: Growing neural gas - experiments with gng, gng with utility and supervised gng (2002)
16. Hyvärinen, A., Karhunen, J., Oja, E.: Independent component analysis. New York: John Wiley and Sons **ISBN 978-0-471-40540-5** (2001)
17. 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)
18. Jinachitra, P.: Constrained em estimates for harmonic source separation. Proceedings of the 2003 International Conference on Multimedia and Expo - Volume 1 pp. 617 – 620 (2003)
19. Ketchantang, W., Derrode, S., Martin, L., Bourennane, S.: Pearson-based mixture model for color object tracking. Machine Vision and Applications **19**(5-6), 457–466 (2008)
20. Key, J.: The em algorithm in medical imaging. Stat Methods Med Res **6**(55), 55–75 (1997)
21. Kohonen, T.: Analysis of a simple self-organizing process. Biological Cybernetics **44**(2), 135–140 (1982)
22. Kohonen, T.: Self-organizing formation of topologically correct feature maps. Biological Cybernetics **43**(1), 59–69 (1982)
23. Kouzani, A.Z.: Classification of face images using local iterated function system. Machine Vision and Application **19**(4), 223–248 (2008)
24. Lanterman, A.: Schwarz, wallace and rissanen: Intertwining themes in theories of model order estimation. Int'l Statistical Rev. **69**, 185–212 (2001)
25. MacQueen, J.B.: Some methods for classification and analysis of multivariate observations. Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability. pp. 281–297 (1967)
26. McLachlan, G., Peel, D.: Finite mixture models. John Wiley and Sons (2000)
27. Montesano, L., Lopes, M., Bernardino, A., Santos-Victor, J.: Learning object affordances: From sensory motor maps to imitation. IEEE Trans. on Robotics **24**(1) (2008)
28. Pernkopf, F., Bouchaffra, D.: Genetic-based em algorithm for learning gaussian mixture models. IEEE Trans. Patt. Anal. Mach. Intell. **27**(8), 1344–1348 (2005)
29. Rissanen, J.: Stochastic complexity in statistical inquiry. Wold Scientific Publishing Co. USA (1989)
30. Sakimoto, Y., Iahiguro, M., Kitagawa, G.: Akaike information criterion statistics. KTK Scientific Publisher, Tokio (1986)
31. 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 (2006)
32. Ueda, N., Nakano, R., Ghahramani, Y., Hiton, G.: Smem algorithm for mixture models. Neural Comput **12**(10), 2109–2128 (2000)
33. Wallace, C., Freeman, P.: Estimation and inference by compact coding. J. Royal Statistic Soc. B **49**(3), 241–252 (1987)
34. Zhang, Z., Chen, C., Sun, J., Chan, K.: Em algorithms for gaussian mixtures with split-and-merge operation. Pattern Recognition **36**, 1973 – 1983 (2003)