# Efficient Greedy Estimation of Mixture Models Through a Binary Tree Search

Nicola Greggio[a,c,*], Alexandre Bernardino[c], Paolo Dario[a,b], José Santos-Victor[c]

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

**Abstract**

Unsupervised data clustering can be addressed by the estimation of mixture models, where the mixture components are associated to clusters in data space. In this paper we present a novel unsupervised classification algorithm based on the simultaneous estimation of the mixture's parameters and the number of components (complexity). Its distinguishing aspect is the way the data space is searched. Our algorithm starts from a single component covering all the input space, and iteratively splits components according to breadth first search on a binary tree structure that provides an efficient exploration of the possible solutions. The proposed scheme demonstrates important computational savings with respect to other state-of-the-art algorithms, making it particularly suited to scenarios where the performance time is an issue, such as in computer and robot vision applications. The initialization procedure is unique, allowing a deterministic evolution of the algorithm, while the parameter estimation is performed with a modification of the Expectation Maximization algorithm. To compare models with different complexity we use the Minimum Message Length information criteria that implements the trade-off between the number of components and data fit log-likelihood. We validate our new approach with experiments on synthetic data, and we test and compare to related approaches its computational efficiency in data intensive image segmentation applications.

*Keywords:* Unsupervised clustering, Expectation Maximization, Image Processing, Robotics, Machine Vision.

## 1. Introduction

Data clustering is a topic of major interest in many disciplines, having an extremely wide application range. Several techniques for clustering have been developed during the last decades, including Kohonen maps [1] [2], Growing Neural gas [3], [4], k-means [5], to Independent component analysis [6], [7], and mixture models [8].

To cope with the dynamics of changing environments, robotics applications must process in real-time huge amounts of sensory data. For example, image processing algorithms play a key role in many practical applications and demand significant computational resources due to huge amounts of data.

A paradigmatic application of clustering in robot vision is image segmentation. Several applications in humanoid robots [9], [10], rescue robots [11], or soccer robots [12] rely on some sort on image segmentation [13]. Additionally, many other fields of image analysis depend on image segmentation algorithms: video surveillance, medical imaging and database retrieval are some examples [14], [15].

In this paper we propose an unsupervised data clustering algorithm for data intensive algorithms, moving towards practical applications in robotics and computer vision. We propose computational efficiency improvements to the state of the art unsupervised clustering algorithms and illustrate the achieved performances and computational gains both in synthetic data and on image segmentation applications. Our goal is not to improve the state of the art in image segmentation, but to demonstrate efficiency gains with respect to alternative approaches. The state-of-the-art in image segmentation technology is much beyond simple data clustering, but our methodology can certainly be useful for segmentation techniques requiring clustering algorithms.

Our approach to data clustering is based on the Expectation Maximization algorithm applied to Gaussian mixtures, which allows to approximate arbitrary probability distribution functions. Fitting a mixture model to the distribution of the given data is equivalent, in some applications, to the identification of the clusters with the mixture components [8]. The selection of the right model complexity, i.e. the number of mixture components, is a critical issue. In fact, this is an ill-posed problem with multiple possible solutions depending on a trade-off between data fit and complexity. The higher the number of components in the mixture is, the better the data fit will be. Unfortunately, increasing the number of components will lead both to data overfitting and to increase in the computational burden. Therefore, finding the best compromise between precision, generalization and speed is an essential concern. A common approach to address this compromise is to try different hypotheses for the number of components in the mixture, and then selecting the best one according to some appropriate model selection criteria. In such approaches, the best model is typically selected by executing independent runs of the EM algorithm for many

*Corresponding author
*Email address:* ngreggio@isr.ist.utl.pt (Nicola Greggio)

different initializations of the number of components, and then evaluating each of the solutions with criteria that penalize complex models. Some examples are the Akaike Information Criterion (AIC) [16], the Schwarz's Bayesian Information Criterion [17], the Rissanen Minimum Description Length (MDL) [18], and Wallace and Freeman Minimum Message Length (MML) [19]. However, all of these criteria, in order to be effective, have to be evaluated for every possible number of models under comparison. Thus, it is a combinatorial problem where there is no guarantee of finding an optimal global solution, except with exhaustive search methods.

### *1.1. Our contribution*

Our work follows an incremental model selection approach. Instead of executing independent EM runs spanning a wide range of parameters, and evaluating each one with a model selection criteria, we change the model complexity on-the-fly, while a modified EM algorithm optimizes the mixture parameters. The search for the best number of components in the mixture starts with a single component and progressively adds new components according to a binary tree structure, i.e. forking one of the existing components. Using a breadth-first search scheme on the binary tree structure allows an efficient exploration of the search space and provides "backtracking" capabilities, i.e. if the addition of a new element does not prove useful, we can go back to the best previous solution and continue exploration on a different part of the search space. Such a strategy allows a good search efficiency allied to a good coverage of the possible solution's space, achieving better fits that alternative algorithms, in average. The current paper is an extension of the contents presented in [20]. We compare the proposed method both with our previous work [21] and with the method in Figueiredo and Jain [22].

## 2. Related Work

Due to the complexity of the model selection problem, many algorithms have proposed to escape the classical exhaustive search methods by adopting an incremental approach. Most of these algorithms derive from the original EM formulation, but they are capable of modifying their complexity during the learning process. Here, it is possible to distinguish three different categories: those starting with a low number of components and only increase their number as the algorithm progresses; those starting with a high number and annihilating components along time; and those both incrementing and reducing the number of components. Typically new component are added by random initialization (birth operator) or by dividing existing components (split operation), whereas components are removed either by annihilation (death operator) or joining other existing components (merge operator). Split-and-merge algorithms have been widely used in computer vision, pattern recognition and signal processing [23] [24], and they are often more efficient than exhaustive, random or genetic algorithm approaches. Richardson and Green used split-and-merge operations together with birth-and-death operations to develop a reversible jump Markov chain Monte Carlo (RJMCMC) algorithm for fully Bayesian analysis of univariate gaussians mixtures [25]. The novel RJMCMC methodology elaborated by Green is attractive because it can preferably deal with parameter estimation and model selection jointly in a single paradigm. BrieGy, a random-sweep Metropolis Hastings method, constructs the dimension matching transform with the reversible jump methodology [26]. Then, Ueda *et Al.* proposed a split-and-merge EM algorithm (SMEM) to alleviate the problem of local convergence of the EM method [27]. They analyzed the implications of merge and split operations in terms of their "well-posedness". While merging two components is trivial (the point set in the merged component is the union of the original ones), there are many possible ways to split a component, which results in a ill-posed problem. Merge-only approaches, thus, alleviate this problem. In this class of methods, the approach proposed by Figueiredo and Jain in 2002 is especially interesting [22]. They start with a large number of components and impose a Dirichlet prior on the mixing weights that drives to zero the weights of the components that get low support from data during the steps of the algorithms. Huang *et Al.* proposed a refinement of the method using a new penalized likelihood method that is continuous when components are annihilated and shows good analytic properties [28]. Despite the remarkable accuracy and robustness of merge-only methods, they suffer from an high computational burden given that many EM iterations are performed with a large number of components.

With the aim of reducing as much as possible the required computational demands, split-only approaches start with a single component and add new ones along time according to different criteria. In 2002 Vlassis and Likas introduced a greedy algorithm [29] that starts with a single component covering all the data, and sequentially splits it in two new ones. The parameters of these two components are then adjusted by local EM iterations. The method takes $O(n^2)$ operations, where $n$ is the number of input data samples. Subsequently, Verbeek et al. developed another greedy method [30] by starting 'partial' EM searches, each of them with different initializations. The total complexity for the algorithm to learn a sequence of mixtures composed by $k$ components is $O(k^2 n)$, where $n$ is as before.

In our previous work [21] we proposed a split-only method whose decisions were made in accordance to the state of the gaussian components and a set of adaptive thresholds. In this paper we propose an approach that introduces significant improvements in reducing the number of tuning parameters and performing a better exploration of the search space, leading to a more efficient, robust, and easy to use methodology.

## 3. Mixture Learning Algorithm

Due to the nature of the mixture learning problem with model selection, finding globally optimal solutions is very hard except for problems of very small dimension, where exhaustive search techniques can be applied. All non exhaustive algorithms in literature are greedy in the sense that, at each step, they take local/immediate optimal decisions. Likewise, our approach is

greedy, but we propose a search strategy that covers and explores the state space in a more effective way, using a coarse-to-fine paradigm. To represent this coarse-to-fine approach we use a binary tree data structure. Each tree's node represents a mixture component, while the descendants of a node represent the mixture components that arise from splitting the parent. The proposed algorithm adopts a breadth search approach with branch pruning, based on the following observations:

1. When a component is split in two new ones, the new components usually converge to a smaller size than the original one, i.e. covering a smaller part of the state-space or, in other words, having a smaller covariance. This provides a coarse-to-fine interpretation to the binary tree representation.
2. If, at a certain stage of the algorithm, splitting a component does not lead to improvements, it is likely that a similar result will hold if that same component is split at a latter stage.

Although these heuristics are not valid for every case, we have empirically verified their frequent occurrence in our domain of study. Therefore, based on point 1, we propose an exploration of the state-space using a breadth-first search of the binary tree representation, where coarser regions of the search space are analyzed first. Then, based on point 2, when the expansion of one branch does not improve the solution, we revert to the previous solution, and that branch is not expanded anymore in the future. This rule reflects the intuition that a future expansion of this component would result in a similar outcome, thus not being worth the computation. Together, these two heuristics, create a good balance between computational complexity and coverage of the search space, that we exploit and experimentally evaluate in this paper.

An overview of our algorithm is described in pseudocode 1. The first step consists in initializing the first component, as described in sec. 3.1. Then, it splits (sec. 3.6) each component following a breadth-first scheme over a binary tree structure (sec. 3.7).

We propose a split rule that creates a new component with the same covariance (replication) of the parent but with a mean value perturbed in multiple dimensions. We denote this rule by "multidimensional replication".

During each replication step, an EM like optimizing procedure re-computes the mixture parameters (sec. 3.3). Each time the EM procedure converges to a (local) minimum, a cost function (5) is evaluated and a decision is taken on whether to keep this solution or try a different one. When there are no nodes eligible to have children, the algorithm stops.The following sections describe all these points in detail.

### *3.1. Parameters' initialization*

Let a dataset $X$ be composed of $N$ samples of dimension $d$:

$$X = \{x_n, n = 1 \cdots N\}, \quad x_n \in \mathbb{R}^d \tag{1}$$

The starting component is initialized with the mean and the covariance of the whole data set:

$$\begin{aligned} \mu &= \frac{1}{N}\sum_{n=1}^{N} x_n \\ \Sigma &= \frac{1}{N}\sum_{n=1}^{N} (x_n-\mu)(x_n-\mu)^T \end{aligned} \tag{2}$$

### *3.2. Model Selection Criterion*

A generic mixture distribution can be defined as:

$$p(x \mid \vartheta) = \sum_{k=1}^{K} w_k p(x \mid \vartheta_k), \quad \sum_{k=1}^{K} w_k = 1 \tag{3}$$

where $K$ is the number of components of the mixture, and $w_k$ are the class prior weights of each component $k$. A Gaussian mixture model is a mixture where all components are Gaussian:

$$p(x \mid \vartheta_k) = \frac{\exp^{-\frac{1}{2}(x-\mu_k)^T \Sigma_k^{-1}(x-\mu_k)}}{\sqrt{(2\pi)^d \mid \Sigma_k \mid}} \tag{4}$$

where $\mu_k$, $\Sigma_k$ are the mean and variance of the $k^{th}$ Gaussian component, jointly denoted as $\vartheta_k$. The union of $K$, $w_k$ and $\vartheta_k$ is denoted $\vartheta$ and completely characterizes the mixture.

One of the most critical parameters to estimate in a mixture is its complexity, given by the number of components $K$. Several information criteria have been proposed for determining the *best* mixture complexity. A comprehensive survey on the most well-known criteria can be found in [31]. We adopted the minimum message length (MML) criterion. This criterion is equivalent to imposing a Dirichlet prior on the distribution of the class weights $w_k$ that favours low complexity models [22]. The optimal set of parameters of the mixture model then results from the following optimization problem (c.f. [22], Eq. (14)):

$$\vartheta_{opt} = \arg\min_{\vartheta} \frac{P}{2}\sum_{k=1}^{K} \log\frac{N\cdot w_k}{12} + \frac{K}{2}\left(P+1+\log\frac{n}{12}\right) - L_X(\vartheta) \tag{5}$$

where $P$ is the number of parameters specifying each component, which in the case of a normal distribution are the mean and the covariance matrix parameters, $P = d + d(d+1)/2$; and $L_X(\vartheta)$ is the log-likelihood of the data given the model parameters:

$$L_X(\vartheta) = \sum_{n=1}^{N} \log p(x_n \mid \vartheta) \tag{6}$$

The relationship to the original MML and MDL criteria has been discussed in [22].

### *3.3. The modified EM*

The EM iterations applied in our work follow a modified version proposed in [22]. The E-Step is computed as in the standard case:

$$\pi_k^n = \frac{w_k^{(t)} \cdot p_k(x_n)}{\sum_{k=1}^{K} w_k^{(t)} \cdot p_k(x_n)} \tag{7}$$

where the $p_k(x_n)$ is the likelihood of a data sample $x_n$ being generated by component $k$.

The M-Step however differs in the class prior weights update:

$$w_k^{(t+1)} = \frac{\left(\sum_{n=1}^{N} \pi_k^i\right) - \frac{P}{2}}{\sum_{j=1}^{K}\left[\left(\sum_{n=1}^{N} \pi_j^n\right) - \frac{P}{2}\right]} \tag{8}$$

The difference arises from adopting in (5) improper Dirichlet priors for the class probabilites [32]:

$$p(w_1, \cdots, w_K) \propto \exp^{-\frac{P}{2}\sum_{k=1}^{K} \log w_i} \tag{9}$$

Finally, the means and covariances of each class are computed in the standard way.

### *3.4. The Stopping Procedure*

We define the stopping criterion for the EM algorithm not only when the total distribution log-likelihood ceases increasing, but also when all individual components stabilise. This is assessed by checking the evolution of the partial log-likelihood functions:

$$L_X(\vartheta_k) = \sum_{n=1}^{N} \log p(x_n \mid \vartheta_k) \tag{10}$$

We define the percentage increment of the partial log-likelihood of a component as:

$$\Lambda_k^{(t)}(\vartheta_k) = \left| \frac{L_X^{(t)}(\vartheta_k) - L_X^{(t-1)}(\vartheta_k)}{L_X^{(t)}(\vartheta_k)} \right| \cdot 100 \tag{11}$$

When the average percentage increment in partial log-likelihood is lower than a small value $\delta$, the algorithm stops:

$$\sum_{k=1}^{K} \Lambda_k(\vartheta_k) \leqslant c \cdot \delta \tag{12}$$

In our experiments (see sec. 5) we use $\delta = 0.1\%$

### *3.5. Ill-Conditioned components*

It may happen that during the EM steps the computation leads to a ill-conditioned component. We solved this by controlling the determinant of each component's covariance matrix every iteration. If it goes under a limit (e.g. $10e^{-4}$) this means that the component may represent (actually cover) a small amount of data or that the component is too elongated. An ill-conditioned component generates a higher MML value (therefore unsatisfactory), since that component does not contribute enough to the input data description. In this case we reject the current mixture configuration and go along the binary tree for replicating another component.

### *3.6. Component Replication*

Splitting a mixture component is one of the standard operators in the algorithms that try to optimize the model complexity incrementally. The rationale is to divide each existing component in two new ones covering roughly the same area as the

**Algorithm 1:** FSAEM: Pseudocode

```
   input : data set X, replication parameter ε (sec. 3.6)
   output: mixture description ϑ (sec. 3.2)
1  Parameters' initialization; (sec. 3.1);
2  Run EM (sec. 3.3) and evaluate minimum message length criterion (eq. 5) with the single component mixture configuration;
3  best mixture = mixture (Backup the mixture);
4  while (stop = 0) do
5      level = last tree level;
6      left_NODE = 2^(level−1) (most left last level node, being the root = 1);
7      right_NODE = (2^level − 1) (most right last level node, being the root = 1);
8      for (node = left_NODE to right_NODE) do
9          Replicate component node (sec. 3.6);
10         Run EM on current mixture (sec. 3.3) ;
11         for (All mixture components) do
12             if (Ill − Conditioned component) then
13                 mixture = best mixture (Restore previous mixture);
14         if (log − likelihood percentage increment < δ) (sec. 3.4) then
               if (All the single components are stable) (eq. 12) then
15                 Stop the EM computation;
16         current mml = Evaluate minimum message length (eq. 5);
17         if (current mml < last mml) then
               (Case A: The current information criterion is better than the best one ⇒ backup the current mixture as the best one.);
18             best mixture = mixture (Backup (bu) mixture);
19             improvement = true;
           else if (current mml ⩾ last mml) then
               if The component has been replicated along all its covariance matrix dimensions then
                   (Case B: The current information criterion is worse than the best one ⇒ bad replication. Return to the previously bu mixture.);
20                 mixture = best mixture (Restore previous mixture);
               else
                   Replicate along another component's covariance matrix eigenvector (go to line 9);
21         if (improvement ≠ 0) then
22             level = level + 1;
23         else if (improvement = 0) then
24             stop = 1;
```

parent, and search for the new best mixture parameters starting from that configuration. However, the division of a new component is a ill-posed problem since it can be done in many different ways. The simplest possible parameter free way would be to duplicate one component in two new ones having identical mean and covariance, and half the prior probability of the original component. This would maintain the log-likelihood of the data but increase the complexity of the model, so the overall cost function would increase its value. Therefore, a new run of the EM algorithm would be necessary to try to reduce the cost. However, the initialization of the two new components with exactly the same parameters creates a fixed point in the cost function that will prevent the algorithm from progressing[1]. To solve this situation in practice, the two components must be initialized with different parameters. The classical way to tackle this problem is to use the so called called split operator: The original component's coverage is divided among the two new ones by using distinct means and covariance matrices in order to describe the original data with low overlap. However, there is few intuition on how to choose the new mean vector and covariance matrices. Instead, we adopt a simpler procedure, that we found more effective than the splitting approaches in our experiments:

- The new components are initialized with mean and covariance identical to the original one except for a perturbation of the mean along one of the components' principal directions (represented by the covariance matrix's eigenvectors). The prior probability of each new component is assigned to half of the prior probability of the parent. Since the two new components are very similar to the original one, this operation is denoted "replication".
- The mean perturbation is applied sequentially along each components' principal dimensions until improvements are obtained or all dimensions have been explored. We denote this operation "multidimensional replication".

Still, the perturbation parameter, $\epsilon$, is an empirical value in our algorithm we must tune. According to our experiments, $\epsilon$ depends mostly in the dimensionality and amplitude range of the data. Although we did not study the problem in depth, we have noticed this parameter's optimal value does not change significantly for different samples of the same type of data. Therefore, $\epsilon$ can be tuned off-line in a few samples of data of the same type. We will report the values we used in our experiments in sec. 5.1.

Then, the modified EM is run for optimizing the mixture. Finally, the algorithm decides whether keeping the new components or reverting them to the original one, based on the cost function evaluated by (5).

### *3.7. Binary Tree Search*

To represent the state of the search algorithm we adopt a binary tree structure. The current mixture components are defined at each time step by the leaves of the tree. Beyond the parameters of the corresponding component, a node of the tree contains a tag indicating whether that specific component has been previously replicated without success, so it would not be replicated again. The optimization procedure corresponds to an exploration of the tree until a configuration can not be improved anymore. We adopt a breadth-first search exploration:

- The initial tree starts with the root, representing the initial component covering all the dataset covariance;
- Each node either has no children (it is a leaf) or two children;
- Only the leaves correspond to active components
- When a component is replicated, the corresponding node is expanded into two children. One of the children is an exact copy of the parent and the other is perturbed in position, as explained before.
- When a multidimensional replication step finishes without improvement, the solution is backtracked to the original component. Otherwise the current solution is accepted.
- The next node to replicate is the left-most, least-depth node that has not previously been replicated.

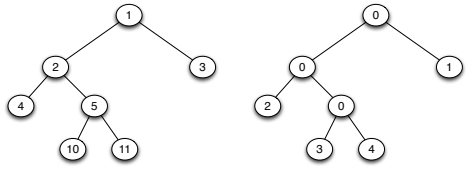
The breadth-first nature of the tree search algorithm results in a coarse to fine exploration of the state-space. Since, in general, the first levels of the tree represent coarser components, these are explored during the first stages of the optimization. Then, at later stages, increasingly finer decompositions are tested.

The actual implementation of the binary tree is made with an implicit data structure in an array (*Ahnentafel list*). Nodes are stored linearly from top to bottom and from left to right. This structure needs no pointers and allows easy indexing to any of the tree nodes. In Fig. 1 we show an example of a binary tree representing a certain configuration and the structure actually implemented in memory. Because parents are not necessary after successful optimization of their children, more efficient structures could be used. Since our tree search algorithm is breath-first, an alternative implementation could be made via a list-like structure: The parents are removed from the head, replicated and, after successful optimization, the two resulting components are moved to the tail. If unsuccessful, the parent is moved to the tail and marked. The algorithm stops when all the elements are marked. We note that, depending on the search algorithms used on a tree, different data structures may be used.

#### *3.7.1. Example*

To exemplify the binary tree representation and its relationship to the mixture components, we show in Fig. 2 a run of the algorithm in a 2D point set generated by a mixture with three Gaussians. Each of the sub-figures shows the state of the mixture estimate and the associated binary tree, at the steps of the algorithm when the new components are added or removed from the tree. The grey nodes correspond to the active components in the mixture, while the white nodes are used as backup for backtracking operations. In the proposed algorithm, only

[1] In practice the algorithm will exit the fixed point due to numerical noise, but this may take a large number of iterations.




(a) Binary tree indices (b) Binary tree contents

Figure 1: Binary tree example: On the left we show the nodes' indices as they are stored in a linear array; on the right we show the corresponding number of the mixture component, which points to a structure with the component parameters. The 0 represents and empty nodes corresponding to parent nodes that have been replicated and replaced by their descendants. On the given example, the tree's actual implementation is via the array: $[0,0,1,2,0,0,0,0,0,3,4,0,0,0,0]$.

the immediate parents of the active nodes will be necessary for backtracking. Whenever a component is expanded and none of its expansions improves the solution, it is marked with an asterisk, $(*)$. These nodes will not be further elected for expansion. The numbers in the tree node represent the order the components were created in the process. At the end of the optimization, components 2, 10 and 11 constitute the solution. Note that iteration 158 proposed an unsuccessful expansion of component 3 along the principal dimension, while in iteration 163 the expansion of the same component along the second dimension succeeded. This illustrates the advantage of using a multidimensional replication strategy, without which the algorithm might have been stopped prematurely.

## 4. Other finite mixture learning algorithms

In this section we provide a brief description about other two approaches, based on finite mixture model estimation, namely: (i) a previous work of ours (FASTGMM) [21], and (ii) the algorithm of Figueiredo and Jain[22], that we denote FIGJ. These methods will be used for comparison purposes in the experimental results.

### *4.1. FASTGMM*

The FASTGMM algorithm has been proposed in [21] and uses a split-only approach. FASTGMM also starts with a single component and sequentially splits components until no further improvements are obtained. The main difference to the current proposal is that in FASTGMM the component to split is chosen in an irreversible way based on a few control parameters. Each component in the mixture is characterised by the following variables:

- Likelihood - the data likelihood for that component, i.e. with other components removed.
- Age - a variable that measures how long the component's likelihood does not increase significantly;
- Area - the determinant of the covariance matrix, which represents the "size" of the component.

A set of per-component time varying thresholds are considered:

- $\Lambda_{TH}$ - for determining a significant increase in likelihood;
- $A_{TH}$ - for triggering the split process based on the component's age;
- $\xi_{TH}$ - for deciding to split a gaussian based on its area.

At each time step, all components whose age is larger than $A_{TH}$ and have not varied in likelihood more than $\Lambda_{TH}$ are elected for possible split. The largest component of the eligible set (the one with highest covariance matrix determinant) whose area exceeds $\xi_{TH}$ is split in two components with smaller covariance and with means on major semi-axes middle points. The thresholds are adapted at each iteration to prevent the algorithm from stalling. When a component is split, its thresholds are reset. The algorithm stops when the global log-likelihood no longer increases. For more details please check [21].

### *4.2. FIGJ*

The work of Figueiredo and Jain [22] (FIGJ) can be considered as a merge-only approach. A large number of components is initially selected via random sampling on the available dataset. As the algorithm progresses, components are evolved and annihilated whenever their prior probability is too low, which implicitly merges the deleted component with nearby components.

To address the ill-posedness of random sampling initialization, the authors fixed a desired minimum probability of successful initialization, $\varepsilon$, i.e. the probability that after $k$ samples from the mixture, none is generated from a component with less than $\alpha_{min}$ prior probability (sec. 6.1 of [22]). This leads to the rule:

$$k > \frac{\ln \varepsilon}{\ln (1 - \alpha_{min})} \tag{13}$$

The means of the components are assigned to the value of the sampled points and the covariances are initialized to $\sigma^2 I$, where $\sigma^2$ a fixed fraction of the mean of the variances along each data dimension. For instance, suppose that to describe a data mixture with about 12 classes, we decide to set the initial components with the same prior probability, i.e. $1/12$. Then, for a probability of successful initialization (draw at least one sample from each component) of $90\% \implies 1 - \varepsilon = 0.9 \implies \varepsilon = 0.1$ leads to $c_{min} = 26,4 \cong 27$ as the minimum number of components, i.e. which is more than double the real number components. This may lead to a computational burden larger than other approaches.

## 5. Experiments and Discussions

In this section we describe the experiments performed to validate the proposed algorithm, which we denote FSAEM. We compare to our previous approach (FASTGMM), and to Figueiredo and Jain (FIGJ) algorithm.

We performed three experiments:

1. Clustering synthetic data with know generating mixture

2. Generic image segmentation
3. Object segmentation in simplified backgrounds

The first experiment is evaluated quantitatively with distance to the ground truth mixture. The second experiment is evaluated both quantitatively with data log-likelihood and qualitatively on the characteristics of the segmentation. The third experiment is evaluated qualitatively on human judgement about the correct result. The latter has the purpose of assessing the applicability of the algorithms to object manipulation in table-top scenarios. All the experiments are benchmarked quantitatively on the computation time.

### *5.1. Setting the Parameters*

The only critical parameter in our algorithm is $\varepsilon$, i.e. the amount of perturbation that is given to the mean of each component after replication (see sec. 3.6). For the experiments, we have performed a grid search for the best mixture log-likelihood, in the range 10E-6 to 10E1, with steps of 0.1 in the exponent. The "optimal" values found were $\varepsilon = 2.8$ for the 2D data, and $\varepsilon = 0.1$ for the color images. With respect to the other two algorithms (FASTGMM and FIGJ) we followed the guidelines presented in the corresponding publications.

### *5.2. Synthetic Data Clustering*

In the first batch of experiments, we artificially created sets of bi-dimensional data points by sampling known (ground truth) Gaussian mixture distributions. Each set is composed by 2000 2D points using mixtures with different number of components. We present 3 cases: i) a mixture with 3 low overlap components; ii) 8 high overlapping components; and iii) 16 low overlap components. We applied the algorithms from Figueiredo and Jain's approach (FIGJ), our previous technique (FASTGMM), and our new approach (FSAEM) on the data sets. We evaluated the output (estimated density) in terms of distance to ground truth and the computation time of the algorithms. All implementations are in non-optimized MATLAB code.

We evaluate the difference between the generated mixture and the estimated one by a distance metric. Several metrics have been proposed in the literature to measure distance between Gaussian mixture distributions, being the most common ones the Kullback-Leibler distance, the Earth Mover's distance, and the Normalized L2 distance [33]. For computational reasons, we have chosen the Normalized L2 distance because it can be computed in closed form [34] with Gaussian mixtures with more than one component, on the contrary to the Kulback-Leibler and the Earth Mover's distances [33].

Let us consider two Gaussian mixture distributions, indexed by $m$:

$$p_m(x) = \sum_{i=1}^{K_m} w_{mi}\mathcal{N}(\mu_{mi}, \Sigma_{mi}), \quad \sum_{i=1}^{K_m} w_{mk} = 1, \quad m = 1,2 \quad (14)$$

where $\mathcal{N}$ is the multivariate Gaussian density function.

The normalized L2 distance was introduced in [33] as:

$$d_{nL2}(\tilde{p}_1, \tilde{p}_2) \triangleq \int_x (\tilde{p}_1(x) - \tilde{p}_2(x))^2 \, \mathrm{d}x$$

where the input distributions are normalised to unit L2-norm:

$$\tilde{p}_m(x) = \frac{1}{Z_m} p_m(x), \quad Z_m = \sqrt{\int_x p_m(x)^2 \, \mathrm{d}x}$$

Because distributions are normalized ($\int_x \tilde{p}_m(x)^2 \, \mathrm{d}x = 1$), we get:

$$d_{nL2}(\tilde{p}_1, \tilde{p}_2) = 2\left(1 - \int_x (\tilde{p}_1(x)\tilde{p}_2(x)) \, \mathrm{d}x\right) \quad (15)$$

To compute the normalisation terms and the closed form expression of the distance for Gaussian mixtures, the formulas for the product of two Gaussian distributions are used [34], (Eqs. 5.1 and 5.2):

$$\mathcal{N}(\mu_a, \Sigma_a) \cdot \mathcal{N}(\mu_b, \Sigma_b) = z_{a \cdot b}\mathcal{N}(\mu_{a \cdot b}, \Sigma_{a \cdot b})$$

with

$$\begin{aligned} \Sigma_{a \cdot b} &= \left(\Sigma_a^{-1} + \Sigma_b^{-1}\right)^{-1} \\ \mu_{a \cdot b} &= \Sigma_{a \cdot b}\left(\Sigma_a^{-1}\mu_a + \Sigma_b^{-1}\mu_b\right) \\ z_{a \cdot b} &= \mid 2\pi(\Sigma_a + \Sigma_b) \mid^{-\frac{1}{2}} \exp^{-\frac{1}{2}(\mu_a - \mu_b)^T(\Sigma_a + \Sigma_b)^{-1}(\mu_a - \mu_b)} \end{aligned}$$

Thus, for the Gaussian mixtures, the normalisation terms can be computed by:

$$Z_m = \left(\sum_{i=1}^{K_m}\sum_{j=1}^{K_m} w_{mi}w_{mj} \frac{\exp^{-\frac{1}{2}(\mu_{mi} - \mu_{mj})^T(\Sigma_{mi} + \Sigma_{mj})^{-1}(\mu_{mi} - \mu_{mj})}}{\mid 2\pi(\Sigma_{mi} + \Sigma_{mj}) \mid^{\frac{1}{2}}}\right)^{\frac{1}{2}}$$

The L2 distance (15) between two mixtures (14) then becomes:

$$d_{nL2}(p_1, p_2) = 2\left(1 - \frac{\sum_{i=1}^{K_1}\sum_{j=1}^{K_2} w_{1i}w_{2j}z_{1i \cdot 2j}}{Z_1 Z_2}\right) \quad (16)$$

Fig. 3 shows the output of the test. Each row corresponds to one of the datasets (3, 8, and 16 components respectively), and each column corresponds to a different algorithms (FIGJ, FASTGMM, and FSAEM respectively). Both FSAEM and FIGJ successfully find a number of components close to the ground truth, and have similar data fits, as shown in Fig. 3 and Tab. 1. Quantitatively, the log-likelihood and normalised L2 distance values in FIGJ and FSAEM are very similar, without significant differences. However, FSAEM has significantly better computational times. FASTGMM shows a less stable behaviour in all aspects. Despite being the fastest algorithm, it clearly presents worse results in the estimation of the number of components and normalised L2 distance. For the mixture with 3 components, FASTGMM actually shows the best log-likelihood values, but this is an artefact due to the overestimation of the number of components (the log-likelihood tends to improve as the number of components grow, whereas the normalised L2 distance is immune to this fact).

### *5.3. Image Segmentation*

In the second set of experiments we use real color image data, considering both pictures taken from photographic cameras and pictures taken from video streaming cameras (webcams and

| Input | Algorithm | # Initial gaussians | # Detected gaussians | Actual gaussian number | # Iterations | Elapsed Time [s] | Log-likelihood | Normalized L2 Distance [$\times 10^{-3}$] |
|---|---|---|---|---|---|---|---|---|
| 3-gau: | FIGJ | 16 | **3** | 3 | **309** | 9.55 | −7249.1 | **2.7** |
| | FASTGMM | 1 | 4 | | 670 | **2.07** | **−7231.7** | 202.4 |
| | FSAEM | 1 | **3** | | 1106 | 4.59 | −7249.0 | 3.1 |
| 8-gau: | FIGJ | 23 | **7** | 8 | **225** | 14.69 | −8397.5 | **9.8** |
| | FASTGMM | 1 | 5 | | 257 | **0.85** | −8959.7 | 199.1 |
| | FSAEM | 1 | **7** | | 2055 | 11.11 | **−8397.4** | 10.2 |
| 16-gau: | FIGJ | 48 | 14 | 16 | **364** | 34.82 | −8150.8 | 39.1 |
| | FASTGMM | 1 | 8 | | 476 | **2.94** | −8693.8 | 311.9 |
| | FSAEM | 1 | **16** | | 2659 | 23.02 | **−8133.7** | **33.7** |

Table 1: Experimental results on synthetic data.

| $(R,G,B,x,y)$ Color Segmentation Results | | | | | | |
|---|---|---|---|---|---|---|
| Input | Algorithm | # Initial components | # Detected components | # Iterations | Elapsed time [s] | Log-likelihood |
| 1 | FIGJ | 6 | 6 | 304 | 308.476 | -4.5e5 |
| | FASTGMM | 1 | 7 | 590 | 75.355 | -4.3e5 |
| | FSAEM | 1 | 6 | **104** | **8.288** | **-3.9e5** |
| 2 | FIGJ | 18 | 18 | 206 | 113.051 | -4.6e5 |
| | FASTGMM | 1 | 5 | 328 | 102.416 | -4.3e5 |
| | FSAEM | 1 | 5 | **64** | **47.665** | **-4.3e5** |
| 3 | FIGJ | 2 | 2 | **18** | **2.603** | -4.6e5 |
| | FASTGMM | 1 | 3 | 121 | 4.02 | -4.1e5 |
| | FSAEM | 1 | 3 | 58 | 3.303 | **-3.5e5** |
| 4 | FIGJ | 18 | 18 | 233 | 151.035 | -4.5e5 |
| | FASTGMM | 1 | 8 | 231 | 19.515 | -4.2e5 |
| | FSAEM | 1 | 6 | **117** | **9.990** | **-4.1e5** |
| 5 | FIGJ | 24 | 24 | **351** | 605.001 | -5.1e5 |
| | FASTGMM | 1 | 10 | 631 | 99.515 | -4.6e5 |
| | FSAEM | 1 | 12 | 581 | **78.27** | **-4.4e5** |
| 6 | FIGJ | 16 | 16 | 352 | 242.379 | **-2.6e5** |
| | FASTGMM | 1 | 12 | 260 | 130.416 | -3.2e5 |
| | FSAEM | 1 | 5 | **152** | **16.348** | -4.3e5 |
| 7 | FIGJ | 18 | 18 | **305** | 220.185 | -5.9e5 |
| | FASTGMM | 1 | 21 | 810 | 193.59 | **-4.5e5** |
| | FSAEM | 1 | 19 | 735 | **161.534** | -4.9e5 |
| 8 | FIGJ | 5 | 5 | 503 | 18.557 | -4.4e5 |
| | FASTGMM | 1 | 3 | 354 | 37.059 | -4.2e5 |
| | FSAEM | 1 | 3 | **245** | **17.354** | **-3.9e5** |
| 9 | FIGJ | 18 | 18 | 273 | 237.854 | -5.3e5 |
| | FASTGMM | 1 | 6 | 246 | 37.913 | -4.9e5 |
| | FSAEM | 1 | 7 | **213** | **26.259** | **-4.9e5** |
| 10 | FIGJ | 14 | 14 | **183** | 89.011 | -4.7e5 |
| | FASTGMM | 1 | 7 | 320 | 68.422 | -4.2e5 |
| | FSAEM | 1 | 8 | 283 | **33.461** | **-4.1e5** |
| 11 | FIGJ | 13 | 12 | **159** | 47.778 | -4.4e5 |
| | FASTGMM | 1 | 8 | 329 | 35.269 | -4.1e5 |
| | FSAEM | 1 | 8 | 230 | **23.645** | **-3.7e5** |
| 12 | FIGJ | 13 | 13 | 180 | 144.418 | -6.9e5 |
| | FASTGMM | 1 | 4 | 193 | 30.269 | -6.4e5 |
| | FSAEM | 1 | 5 | **133** | **21.348** | **-6.4e5** |

Table 2: Experimental results on real images segmentation.

robot cameras). Each image generates a 5-dimensional data set, where each data point is composed by the three components of the *RGB* color space and its two dimensional pixel coordinates. An input point $p_i$ has the form: $p_i = (R_i, G_i, B_i, x_i, y_i)$.

Again, we use the algorithms FIGJ, FASTGMM and FSAEM to cluster the data points. However, the distance to the original mixture cannot be evaluated in this case, since no ground truth data is available. Therefore, we rely on a qualitative evaluation through the data likelihood and a visual assessment of the obtained segmentation. Again, we note that the purpose of this work is not to improve the state-of-the-art of image segmentation, but to compare the ability of our algorithm to be quicker than other state-of-the-art in Gaussian mixture estimation, while being competitive in terms of performance. A visual inspection of the segmentation results and the assessment of the data log-likelihood will be the means to check similarity between different methods.

Fig. 4 shows the experimental results. For each row, from left to right, we show the original image, the results of the FIGJ, FASTGMM, and FSAEM. Table 2 presents numeric details, namely the final number of components, the data log-likelihood and the computation time. From Fig. 4 we can notice the similarity in output between the different methods, with FIGJ often resulting in a more detailed segmentation (higher number of components). However this tendency for over-segmentation does not bring much added value to practical applications since objects become fractioned. In Table 2, we can notice that FSAEM often produces slightly better data fits that the other methods and surpasses FIGJ about 8 $\times$ in computational performances, being now the clear winner.

### *5.4. Object Segmentation*

Finally, in the last experiments, we evaluate the ability to segment objects in a simplified background scenario, typical of robotic manipulation settings. We have used images taken by the cameras of our robotic platform (the iCub), in different lighting conditions. The scenario is composed by multiple objects of interest with simple characteristics (blob like shapes and uniformly coloured) laying on a table. As it was observed in the previous experiment, the algorithm FIGJ proved too expensive for eventual real-time applications dealing with image data, so it was not used in this last set of experiments.

The purpose of this experiment is to evaluate the ability of color based mixture model estimation to be used as a pre-

| $(R,G,B,x,y)$ Object Segmentation Results | | | | | | | |
|---|---|---|---|---|---|---|---|
| Image number | Algorithm | Detected number of Gaussian components | Number of iterations | Elapsed Time [s] | Percentage time difference [%] | Final log-likelihood | Percentage difference on log-likelihood [%] |
| (1) | FASTGMM | 2 | 38 | **1.244** | -37.1 | **-2.8e5** | -55.5 |
| | FSAEM | 2 | **26** | 1.705 | | -4.3e5 | |
| (2) | FASTGMM | 3 | **21** | **0.933** | -129.2 | **-3.1e5** | -12.9 |
| | FSAEM | 2 | 38 | 2.135 | | -3.5e5 | |
| (3) | FASTGMM | 11 | 332 | 46.743 | **95.5** | **-4.0e5** | -2.4 |
| | FSAEM | 3 | **31** | **2.080** | | -4.1e5 | |
| (4) | FASTGMM | 11 | 285 | 34.017 | **82.4** | **-3.9e5** | -4.2 |
| | FSAEM | 4 | **106** | **5.983** | | -4.0e5 | |

Table 3: Experimental results on real robotic images. Segmentation performed in the $(R,G,B,x,y)$ color space.

processing step in the pipeline of a simple object segmentation algorithm. The output of the mixture model clustering is post-processed with a smoothing gaussian blur and a connected components labelling in order to obtain compact and connected objects on the scene.

Results are shown in Fig. 5. The obtained connected components are highlighted with a cross identifying their centroid and an ellipse showing their second order geometrical moments. The first three columns on the left correspond to the FASTGMM method and the other columns present the FSAEM version. We have chosen images with significant differences in brightness, contrast and number of objects but we observe that both algorithms perform similarly. FSAEM is able to better detect the relevant objects in images (2) and (4), since FASTGMM misses a couple of them. However, FSAEM detects one spurious object in image (3) and has problems in the left boundary of image (1). Despite these problems could be mitigated via a better tuning on the connected component parameters, the purpose of this analysis is to comprove that both approaches are viable to tackle this problem. The quantitative results presented in Tab. 3 demonstrate that generally FSAEM runs faster in average. However, in images (1) and (2), the elapsed time is similar. This is due to the low contrast of these images that produce a segmentation with a low number of components, therefore both algorithms finish early. We note however that the number of components is not identical to the number of the detected objects due to the post-processing step via the connected components algorithm. For images (3) and (4), a much larger number of components was obtained and the computational advantages of FSAEM are significant.

### *5.5. Final considerations*

In this section we applied the proposed mixture model estimation algorithm to three different problems: Clustering 2D data points, image segmentation and object segmentation. The results of the proposed method were confronted with other two state-of-the-art algorithms on gaussian mixture model estimation. From the obtained results two main observations can be made: (i) the proposed method is consistently more computationally efficient that the others, and (ii) there is no clear advantage of any of the algorithms in terms of output quality. In some cases one algorithm shows slightly better output quality than the others, but there is no systematic trend we could argue about. The proposed method also presents the advantage of having a simple initialization procedure, a single parameter to tune, and a deterministic behavior. As such, we believe FSAEM is an approach worth exploring in practical applications requiring real-time clustering and segmentation methods, as well as for further advances on the theoretical aspects of mixture model estimation.

## 6. Conclusion

This work presented a new unsupervised clustering algorithm that estimates a finite mixture model by means of a modification of the Expectation Maximization algorithm. The algorithm determines both the model complexity and the mixture parameters in an incremental fashion. It starts from a single component describing all the data set and evolves by replicating components in a binary tree structure in order to cover a significant part of the search space. The resulting mixture is evaluated with the Minimum Message Length model selection criterion. The binary tree organization of the computations is a key feature of our approach. It improves the space coverage in a way that leads to systematic gains in efficiency with respect to other approaches of the same purpose. We performed experiments involving synthetic data and real images, comparing our new approach against the alternative approaches. We showed that our new algorithm performs considerably faster while achieving competitive accuracy in data description. In future work we will study more systematic ways of tuning the single free parameter of the algorithm, looking forward for fully unsupervised and automated mixture model estimation methods.

## Acknowledgements

This work was supported by the Portuguese FCT agency through project PEst-OE/EEI/LA0009/2013, and the European Commission project POETICON++ (FP7-ICT-288382).

[1] T. Kohonen, Analysis of a simple self-organizing process., Biological Cybernetics 44 (1982) 135–140.

[2] T. Kohonen, Self-organizing formation of topologically correct feature maps., Biological Cybernetics 43 (1982) 59–69.

[3] B. Fritzke, A growing neural gas network learns topologies., Adv ances in Neural Inform ation Processing Systems 7 (NIPS'94), MIT Press, Cambridge MA (1995) 625–632.

[4] J. Holmström, Growing Neural Gas - Experiments with GNG, GNG with Utility and Supervised GNG, Master's thesis, Uppsala University Department of Information Technology Computer Systems Box 337, SE-751 05 Uppsala,, 2002.

[5] J. B. MacQueen, Some methods for classification and analysis of multivariate observations., Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability. (1967) 281–297.

[6] P. Comon, Independent component analysis: a new concept?, Signal Processing, Elsevier 36 (1994) 287–314.

[7] A. Hyvärinen, J. Karhunen, E. Oja, Independent component analysis, New York: John Wiley and Sons ISBN 978-0-471-40540-5 (2001).

[8] G. McLachlan, D. Peel, Finite mixture models., John Wiley and Sons (2000).

[9] L. Montesano, M. Lopes, A. Bernardino, J. Santos-Victor, Learning object affordances: From sensory motor maps to imitation, IEEE Trans. on Robotics 24 (2008).

[10] N. Greggio, A. Bernardino, C. Laschi, J. Santos-Victor, P. Dario, Real-time 3d stereo tracking and localizing of spherical objects with the icub robotic platform, Journal of Intelligent & Robotic Systems (2011) 1–30. 10.1007/s10846-010-9527-3.

[11] S. Carpin, M. Lewis, J. Wang, S. Balakirsky, C. Scrapper, Bridging the gap between simulation and reality in urban search and rescue", in: Robocup 2006: Robot Soccer World Cup X.

[12] N. Greggio, G. Silvestri, E. Menegatti, E. Pagello, Simulation of small humanoid robots for soccer domain., Journal of The Franklin Institute - Engineering and Applied Mathematics 346 (2009) 500–519.

[13] M. Vincze, Robust tracking of ellipses at frame rate, Pattern Recognition 34 (2001) 487–498.

[14] J. G. G. Dobbe, G. J. Streekstra, M. R. Hardeman, C. Ince, C. A. Grimbergen, Measurement of the distribution of red blood cell deformability using an automated rheoscope, Cytometry (Clinical Cytometry) 50 (2002) 313–325.

[15] H. Shim, D. Kwon, I. Yun, S. Lee, Robust segmentation of cerebral arterial segments by a sequential monte carlo method: Particle filtering, Computer Methods and Programs in Biomedicine 84 (2006) 135–145.

[16] Y. Sakimoto, M. Iahiguro, G. Kitagawa, Akaike information criterion statistics, KTK Scientific Publisher, Tokio (1986).

[17] G. Schwarz, Estimating the dimension of a model, Ann. Statist. 6 (1978) 461–464.

[18] J. Rissanen, Stochastic complexity in statistical inquiry., Wold Scientific Publishing Co. USA (1989).

[19] C. Wallace, P. Freeman, Estimation and inference by compact coding, J. Royal Statistic Soc. B 49 (1987) 241–252.

[20] N. Greggio, A. Bernardino, J. Santos-Victor, Sequentially greedy unsupervised learning of gaussian mixture models by means of a binary tree structure, 11-th International Conference on Intelligent Autonomous Systems (IAS-11) - Aug 30, Sept 1 (2010).

[21] N. Greggio, A. Bernardino, C. Laschi, P. Dario, J. Santos-Victor, Fast estimation of gaussian mixture models for image segmentation, Machine Vision and Applications (2011) 1–17. 10.1007/s00138-011-0320-5.

[22] A. Figueiredo, A. Jain, Unsupervised learning of finite mixture models, IEEE Trans. Patt. Anal. Mach. Intell. 24 (2002).

[23] L. Xu, Vector quantization, cluster number selection and the emalgorithms, International Conference on Neural Networks and Signal Processing, Nanjing, China (1995) 149–152.

[24] Z. Zhang, C. Chen, J. Sun, K. Chan, Em algorithms for gaussian mixtures with split-and-merge operation, Pattern Recognition 36 (2003) 1973 – 1983.

[25] S. Richardson, P. Green, On bayesian analysis of mixtures with an unknown number of components (with discussion), J. R. Stat. Soc. Ser. 59 (1997) 731–792.

[26] P. Green, Reversible jump markov chain monte carlo computation and bayesian model determination, Biometrika 82 (1995) 711–732.

[27] N. Ueda, R. Nakano, Y. Ghahramani, G. Hiton, Smem algorithm for mixture models, Neural Comput 12 (2000) 2109–2128.

[28] T. Huang, H. Peng, K. Zhang, Model selection for gaussian mixture models, http://arxiv.org/abs/1301.3558 (2013).

[29] N. Vlassis, A. Likas, A greedy em algorithm for gaussian mixture learning, Neural Processing Letters 15 (2002) 77–87.

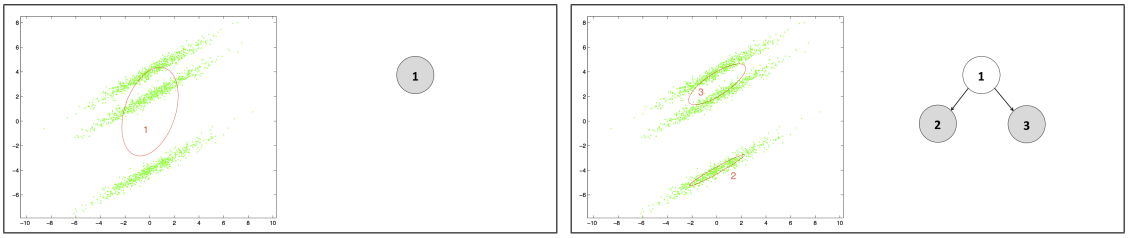
[30] J. Verbeek, N. Vlassis, , B. Krose, Efficient greedy learning of gaussian mixture models, Neural Computation 15 (2003) 469–485.

[31] A. Lanterman, Schwarz, wallace and rissanen: Intertwining themes in theories of model order estimation, Int'l Statistical Rev. 69 (2001) 185–212.

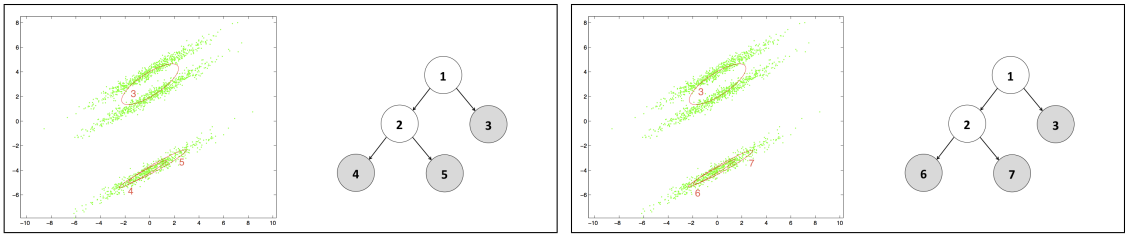
[32] B. J., A. Smith, Bayesian Theory, Chichester UK: John Wiley and Sons, 1994.

[33] J. H. Jensen, D. Ellis, M. G. Christensen, S. H. Jensen, Evaluation distance measures between gaussian mixture models of mfccs, Proc. Int. Conf. on Music Info. Retrieval ISMIR-07 Vienna, Austria (October, 2007) 107–108.

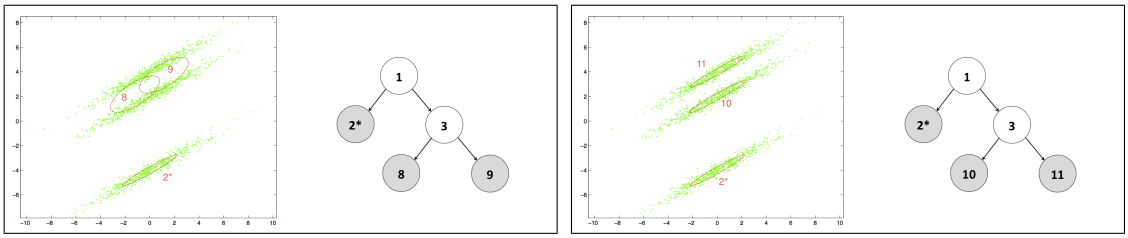
[34] P. Ahrendt, The Multivariate Gaussian Probability Distribution., Technical Report, http://www2.imm.dtu.dk/pubdb/p.php?3312, 2005.



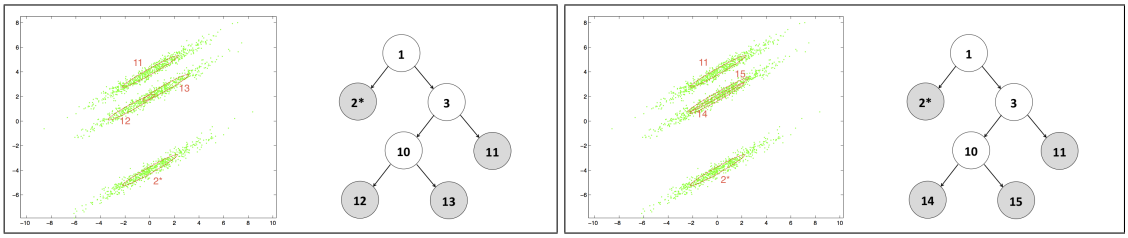

(a) Iteration 1 (b) Iteration 9



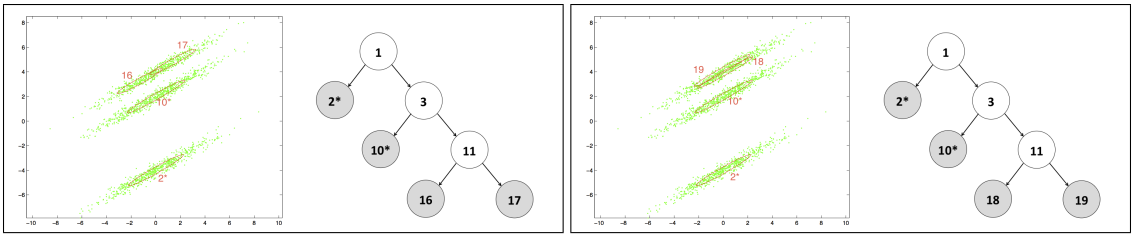

(c) Iteration 59 (d) Iteration 137



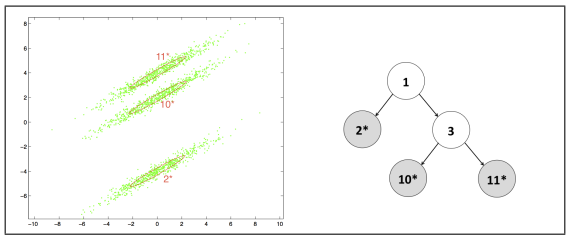

(e) Iteration 158 (f) Iteration 163




(g) Iteration 177 (h) Iteration 216




(i) Iteration 226 (j) Iteration 241




(k) Final solution

Figure 2: Illustration of the evolution of the algorithm. Each subfigure represents a stage when new components are created or removed. In the left of each subfigure it is represented the current state of the estimated mixture. The ellipses represent each of the components of the estimated mixture. In the right of each subfigure it is shown the current structure of the search tree. Greyed nodes correspond to the active mixture components. Nodes with an asterisk are marked as frozen because they have been expanded previously without success.

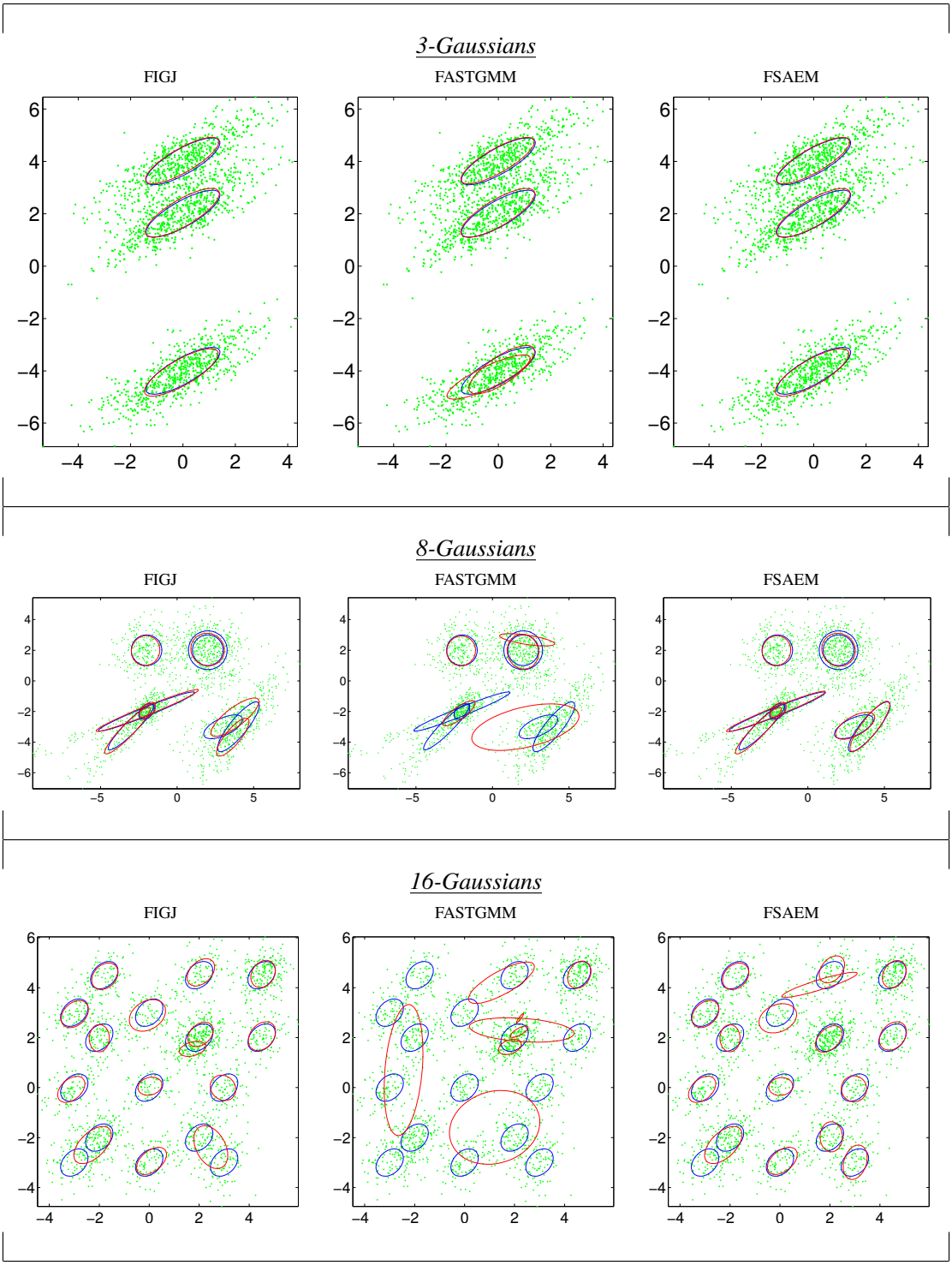



Figure 3: Results of input clustering. Two-dimensional points, generated by the mixtures depicted in blue are represented as black dots in the 2D plane. The estimated mixtures are represented in red. One can observe the equivalence between the tested methods on three different data sets, generated by 3, 8 and 16 Gaussian mixtures, respectively.

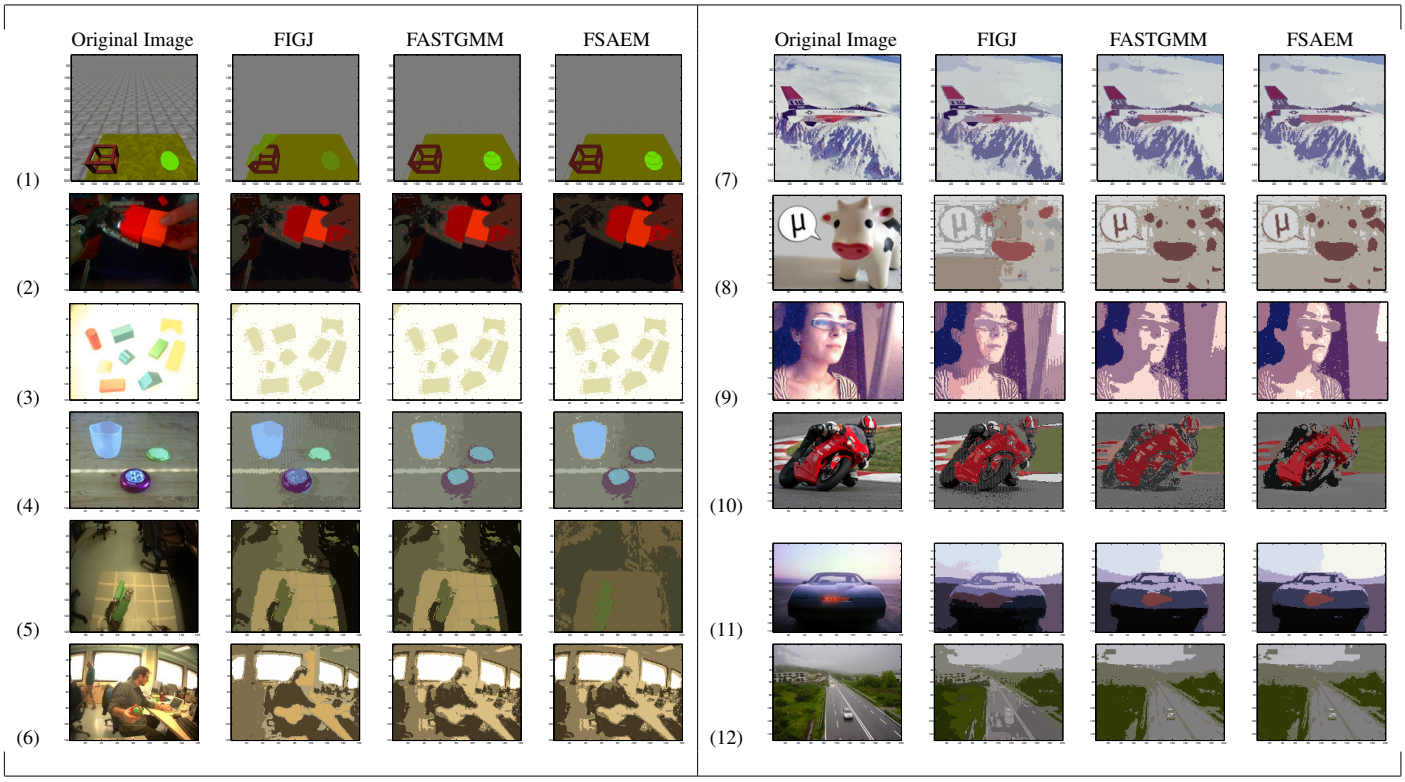



Figure 4: Color images segmentation: From image (1) to (6) we tested the algorithms on real images captured by our robotic platform RobotCub's cameras, and from (7) to (12) we exploit the algorithms' possibilities on general real images.

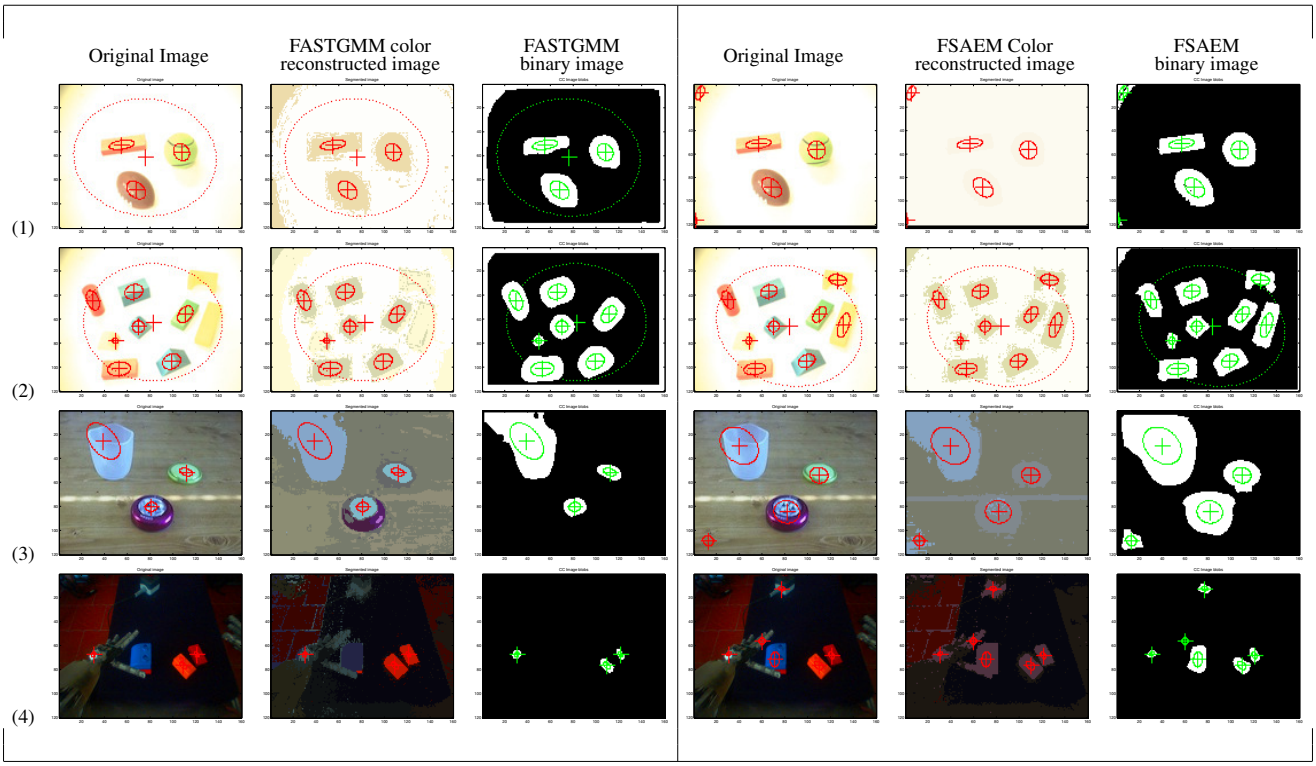



Figure 5: Image segmentation in the $(R,G,B,x,y)$ color space. FASTGMM results are shown on the left, while FSAEM outcomes are represented on the right. For each image, there is a subset composed by the original image, the color image reconstruction, and the binary image labeled with the connected components, respectively. Each image contains the objects of interest highlighted in red for the color output, and green for the binary output. The objects have been marked with their mean and covariance, represented as a regular ellipse in 2D, obtained with the connected components labelling.