NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

BSc THESIS

A Novel Probability-based Data Clustering Algorithm for Detecting Elongated Clusters with Application to the Line Detection Problem

Kyriakos A. Stylianopoulos

Supervisors: **Sergios Theodoridis**, Professor
Konstantinos Koutroumbas, Dr.

ATHENS

JULY 2019

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Ένας Νέος, Πιθανοτικού Υπόβαθρου, Αλγόριθμος Ομαδοποίησης Δεδομένων για την Ανάδειξη Επιμήκων Ομάδων, με Εφαρμογή στο Πρόβλημα της Ανίχνευσης Γραμμικών Στοιχείων σε Εικόνα

Κυριάκος Α. Στυλιανόπουλος

Επιβλέποντες: **Σέργιος Θεοδωρίδης**, Καθηγητής
Κωνσταντίνος Κουτρούμπας, Δρ.

ΑΘΗΝΑ

ΙΟΥΛΙΟΣ 2019

BSc THESIS

A Novel Probability-based Data Clustering Algorithm for Detecting Elongated Clusters with Application to the Line Detection Problem

Kyriakos A. Stylianopoulos
S.N.: 1115201400194

Supervisors: **Sergios Theodoridis**, Professor
Konstantinos Koutroumbas, Dr.

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Ένας Νέος, Πιθανοτικού Υπόβαθρου, Αλγόριθμος Ομαδοποίησης Δεδομένων για την Ανάδειξη Επιμήκων Ομάδων, με Εφαρμογή στο Πρόβλημα της Ανίχνευσης Γραμμικών Στοιχείων σε Εικόνα

Κυριάκος Α. Στυλιανόπουλος
Α.Μ.: 1115201400194

Επιβλέποντες: **Σέργιος Θεοδωρίδης**, Καθηγητής
Κωνσταντίνος Κουτρούμπας, Δρ.

ABSTRACT

Line detection is the process of identifying straight lines or line segments in a given image. Potential applications are commonly found in a variety of fields, such as analysis of geospatial data, image compression and road extraction, to name a few. In this dissertation an approach to the above problem based on probabilistic clustering is explored. A variation of the Gaussian probability distribution centered around a line segment is defined accordingly for the two dimensional space in order to model the line segments in the image under study and an algorithm, called Probabilistic Line Segment CLustering (PLSC) that follows the Mixture Decomposition approach is proposed. The process of finding the optimal positioning of the line segments is carried out by an iterative Expectation-Maximization-like procedure in which the segments are gradually moved in order to fit the actual edges of the image using a heuristic rule. In order to find the appropriate number of segments/clusters, the algorithm starts with an overestimation of it and progressively reduces it via appropriate elimination and unification mechanisms. Toward supporting the value of the proposed method, experimental results have been carried out and discussed in which it is shown that the current method is able to appropriately identify clusters in multiple scenarios. The algorithm can perform mostly comparably and in some cases, even favorably with regard to a selection of relevant published methods.

SUBJECT AREA: Pattern Recognition

KEYWORDS: Image Processing, Clustering, Probability Theory, Line Detection, Expectation-Maximization Algorithm

ΠΕΡΙΛΗΨΗ

Ένα σημαντικό πρόβλημα που απαντάται σε διάφορα πεδία εφαρμογών, όπως η ανάλυση γεωχωρικών δεδομένων (geospatial data analysis), η συμπίεση εικόνας (image compression) και η εξαγωγή δρόμων (road extraction), μεταξύ άλλων παραδειγμάτων, είναι αυτό της ανίχνευσης ευθείων γρμμών ή ευθυγράμμων τμημάτων σε μια δεδομένη εικόνα. Στη διατριβή αυτή, προτείνεται μια νέα προσέγγιση στο παραπάνω πρόβλημα, η οποία βασίζεται στην πιθανοτική ομαδοποίηση (probabilistic clustering). Πιο συγκεκριμένα, ορίζεται μια νέα κατανομή πυκνότητας πιθανότητας η οποία αποτελεί παραλλαγή της Γκαουσσιανής κατανομής πιθανοτήτων (Gaussian probability distribution), στην οποία το κέντρο της δεν είναι πλέον σημείο, αλλά ένα ευθύγραμμο τμήμα, με στόχο τη μοντελοποίηση ευθυγράμμων τμημάτων. Στη συνέχεια, το σύνολο των δεδομένων σημείων θεωρείται ότι προέρχεται από μία κατανομή που εκφράζεται ως ένα σταθμισμένο άθροισμα (μίξη)επιμέρους κατανομών και ο στόχος είναι ο προσδιορισμός αυτών των κατανομών, κάθε μία από τις οποίες μοντελοποιεί και μια (γραμμική) συστάδα (linear cluster). Προτείνεται ένας αγόριθμος, ο οποίος ονομάζεται *Αγλόριθμος Πιθανοτικής Συσταδοποίησης Ευθυγράμμων Τμημάτων (Probabilistic Line Segment Clustering algorithm – PLSC)* και ακολουθεί τη λογική της Αποδόμησης Μίξης (Mixture Decomposition). Η διαδικασία εύρεσης βέλτιστων τοποθετήσεων των ευθυγράμμων τμημάτων (κέντρων των κατανομών πιθανοτήτων) φέρεται εις πέρας μέσω μιας επαναληπτικής διαδικασίας παρόμοιας του αλγορίθμου Αναμενόμενης Τιμής - Βελτιστοποίησης (Expectation Maximization), κατά την οποία, τα τμήματα μετακινούνται σταδιακά με σκοπό να ταιριάξουν στις γραμμικές ομάδες που σχηματίζονται από τα δεδομένα, βάσει ενός ευρετικού κανόνα (heuristic rule). Ο αλγόριθμος δεν απαιτεί εκ των προτέρων γνώση του αριθμού των συστάδων. Αντί αυτού, ξεκινά κάνοντας μία υπερεκτίμηση του πλήθους τους και σταδιακά τις μειώνει μέσω κατάλληλων μηχανισμών απαλοιφής και συνένωσης. Με σκοπό την τεκμηρίωση της αξίας της προτεινώμενης μεθόδου, διεξήχθησαν αρκετά πειράματα, τα αποτελέσματα των οποίων δείχνουν ότι η τρέχουσα μέθοδος είναι ικανή να αναγνωρίσει σε πολύ ικανοποιητικό βαθμό συστάδες τόσο σε απλούστερες όσο και σε πολυπλοκότερες περιπτώσεις. Ο αλγόριθμος μπορεί να αποδώσει παρόμοια και, σε μερικές περιπτώσεις, καλύτερα απολέσματα συγκρινόμενος με ένα επιλεγμένο πλήθος σχετικών δημοσιευμένων μεθόδων.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Αναγνώριση Προτύπων

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Επεξεργασία Εικόνας, Ομαδοποίηση, Θεωρία Πιθανοτήτων, Ανίχνευση Γραμμικών Στοιχείων, Αλγόριθμος Αναμενόμενης Τιμής - Μεγιστοποίησης

To my dearest family, for each one helped me in their own, unique way.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

PREFACE

This thesis is my final work as an undergraduate student for the Department of Informatics and Telecommunications of the National and Kapodistrian University of Athens. It has stemmed from my devotion throughout my years of studies, combined with collaboration and invaluable assistance from my supervisors. This dissertation has been my main focus during my final year and it constitutes my first research attempt in Academia. Its subject area lies on the intersection of a number of scientific domains that had previously attracted my interest and I am grateful I was able to work on those topics.

In truth, this work could not have been completed without a strong support. For that, I would like to, first and foremost, thank the Department and especially, both of my supervisors – Prof. Sergios Theodoridis for accepting to oversee the whole venture and Dr. Konstantinos Koutroumbas from the National Observatory of Athens for proposing the subject, educating and working closely with me. Without his involvement in each and every stage of this process, none of this would have been possible. He was able to provide me with guidance whenever needed while also granting me freedom for approaching the work from my own perspective. This experience over the past year was priceless for me and for that, I thank him deeply.

On equal importance, I would like to express my gratitude towards my family. My mother, father and brother, all gave me support, encouragement and immeasurable aid in various aspects, not only for my thesis but throughout my course of studies. Their motivating attitude was instrumental to my completion of this dissertation. I would also like to personally thank my grandparents, uncle and closest friends, whose interest and enthusiasm kept me going in the past few months.

1. INTRODUCTION

1.1 The Process of Edge Detection

In the context of computer vision and digital image processing, feature detection is the stage where semantic information from input images is extracted. The resulting information is often subsequently utilized in solving pattern recognition, classification and image editing problems. A popular kind of feature detection is the detection of discontinuities (i.e., sharp changes) in the brightness of neighbouring points in an image domain. Those points are termed edge points and subsequently, the task of identifying the lines they form is termed edge detection. The motivation behind edge detection is that discontinuities in pixels are assumed to indicate changes in the visual elements of an image (e.g. [1]), containing thus information about its content. In this dissertation, the focus lies on the special case of edge detection, in which, edges form straight lines (or line segments) in the image. An example of edge detection is shown in figure 1.1:. Two different algorithms are applied to the original image providing different results. In (c) the number of true edge pixels detected is greater than the result in (b) but so is the number of false edges. This figure helps to realize that edge detection by itself is not a trivial task.

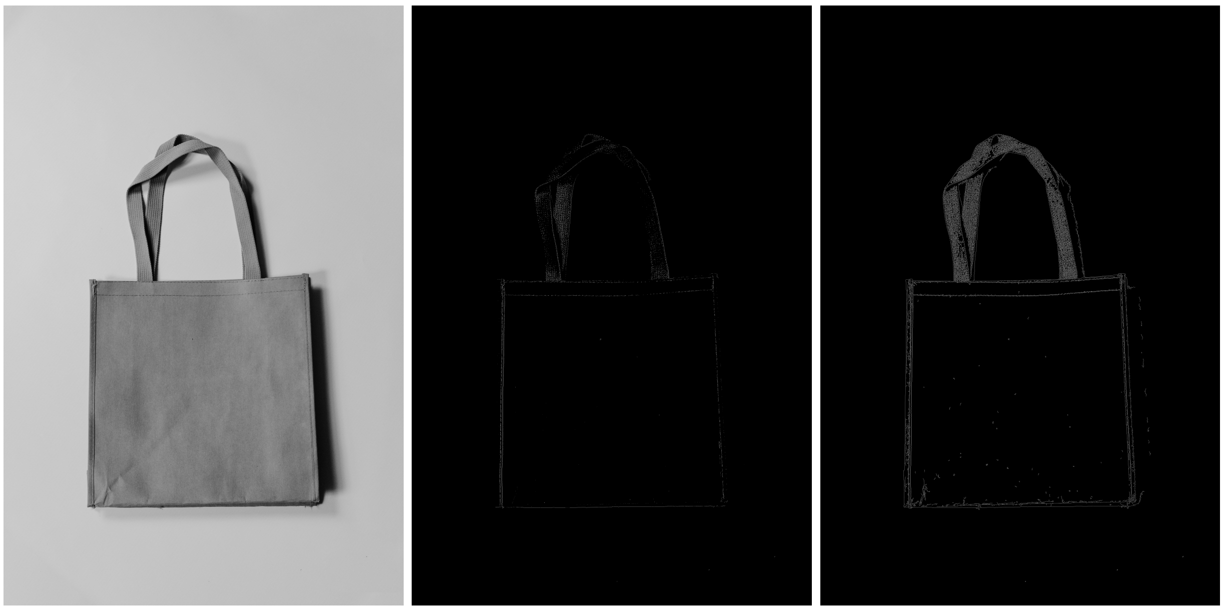

(a) original greyscale image **(b) edge points with Sobel's method [2]** **(c) edge points with Canny edge detector [3]**

Figure 1.1: An example of edge detection.

In the following paragraphs, the basic methodologies of edge detection and line extraction are synopsized.

1.1.1 Overview of Edge Detection Algorithms

Typical edge detection methods can be regarded as processes consisted of the three following discrete steps: [1]
Smoothing – the process of applying noise reduction and regularization techniques in the

[1]The reader may find a detailed review of edge detection algorithms in [4].

image to filter out sharp changes in pixel values and facilitate numerical computations. The operation is being carried out with a convolution of the image with a suitable filter, such as the Gaussian filter [5] in the case where the image under study is contaminated with Gaussian noise and the median filter in the case where the image is contaminated with salt-and-pepper noise [6].

Differentiation – the step in which, edge pixels are located by computing first or second order derivatives. Approaches in literature may estimate the magnitude and direction of the gradient at each pixel (e.g. with the use of Prewitt's [7] or Sobel's [2] masks) or zero-crossings of a second-order derivative measure, such as the Laplacian operator used in [8].

Edge labeling – as a final step, the aim is to localize detected edges and discard false ones. While in the simplest form, a threshold of the gradient value can be used to discard unwanted edges, additional operations, such as edge thinning or edge tracing, must be carried out to provide satisfactory results.

A widely used edge detection algorithm, the *Canny edge detector*, has been introduced in [9]. According to this, a Gaussian smoothing filter is first applied, that is shown to approximate the optimal filter for ideal edge step (i.e., single pixels) extraction. The intensity gradient is computed by such operators, as those discussed above. As a thinning policy, suppression of points with non local maximum gradient magnitude along the direction of the gradient is used. Finally, a high and a low threshold values are both used for eliminating false edges. Specifically, edges with magnitude above the high threshold are kept by default and edges with magnitude between the high and low thresholds are kept only if they are linked to some edge with value above the high threshold.

1.1.2 Detecting Lines and Segments

An important early stage step in recognizing the content of an image, in a variety of computer vision applications, is the detection of straight lines and/or line segments. The main reason behind this is that silhouettes of man made objects are often consisted of straight edges.

The task of tracing edge segments itself can be thought of as a subproblem of edge detection. Conceptually, it can be described as detecting edge regions of the image under the assumption that those regions have approximately the form of straight lines. Algorithms that perform straight line detection take as input a set of edge pixels already discovered by a standard edge detection process and output either a parametric representation of the detected lines or groups of points each one comprising a line – or both.

In the context of image processing, a vast amount of algorithms have been proposed for detecting lines. Most notably, *Hough transform* was introduced in [10]. The algorithm works by defining a transformation between the image domain and a parametric space, termed *Hough space*. Points in the image domain are transformed to lines in the parametric space and points in the Hough space can be interpreted as lines in the original image. Collinear points will result in intersected lines in the parametric space. As a result, image lines can be detected by finding Hough space points, on which, a certain amount of lines intersect. Then one can map those points back to the image domain to get the corresponding lines. Over the years a number of variations on the original method have been explored. For instance, in [11] it shown that the idea can be modified to detect distinct line segments, rather than lines and in [12], a generalized version is proposed for identifying arbitrary shapes.

Another proposed method is given [13]. The algorithm works by first identifying regions of image with gradients with similar directions and then by grouping pixels of such regions together (encapsulating them in rectangles in order to finally construct edge line segments). Additionally, in [14], a simple approach is described that uses a convolution-based technique similar to the ones discussed in section 1.1.1 with filters that detect horizontal, vertical and oblique edge points. When the detection phase is over, an edge linking routine can be applied to group points to distinct lines.

1.2 Clustering Approaches to Line Detection

In recent decades, machine learning has provided an alternative important direction in the field of image understanding. Traditional unsupervised learning algorithms are commonly applied in different areas of computer vision [2] (e.g. in image segmentation [15]). In the context of this work, the focus lies on clustering algorithms and especially their applications in line detection.

Formally, clustering can be defined as the process of finding groups (clusters) of adequately similar objects within a data set – i.e. a collection of data points. Typically, data points are represented as vectors of numerical values. Each numerical value represents the measurement of an observed property, termed a *feature*. Thus data points are regarded as feature vectors in a vector or metric space. Subsequently, the measure of similarity between points is often defined via a norm or distance function on the feature space.

The specific problem of identifying edge segments in images can be regarded as a clustering problem, in which the data set is comprised of edge points (i.e. $\boldsymbol{x} \in \mathbb{R}^2$) forming elongated clusters and the features used for the representation of each edge point are simply its spatial coordinates (i.e. $\boldsymbol{x}_1$, $\boldsymbol{x}_2$) of the pixels.

A number of different clustering categories exist based on the philosophy the corresponding algorithms follow to group the data points. *Hard clustering* is used to describe algorithms that assign each data point to a single cluster. In contrast, *fuzzy* and *possibilistic clustering* algorithms assign points to all clusters with different (nonnegative) ”degrees of membership” or ”compatibility”. Their difference is that under the fuzzy paradigm, the degrees of membership for each single point must add up to 1, while possibilistic algorithms pose no such restrictions. In addition to the above, we have the *probabilistic clustering* family which exhibits close affinity with fuzzy clustering, where degrees of membership are interpreted as probabilities or likelihoods of points belonging to clusters.

1.2.1 Elongated Clusters

”Elongated clusters” is a term given to clusters, the points of which, have feature values that are strongly linearly correlated. That is to say, the variance of the values for a certain feature is significantly greater than the variance of the values for the rest of the features. The term is used rather informally. An adequate but still loose definition for two linearly correlated features can be given terms of the *Pearson correlation coefficient*.

[2]Of course, other types of machine learning algorithms apart from unsupervised learning have been used in the domain of image processing but they lie out of the scope of this work.

In statistical terms, given a population associated with random variables A and B, Pearson's coefficient of correlation is defined as:

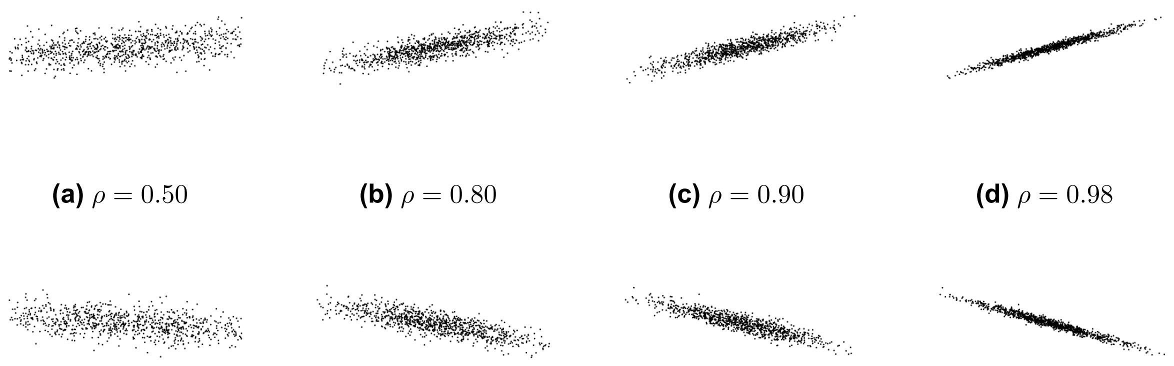
$$\rho_{A,B} = \frac{cov(A,B)}{\sigma_A, \sigma_B}$$

where, $cov(A,B)$ denotes the covariance of random variables A and B and σ_A, σ_B denote the respective standard deviations. In case there is no confusion about the random variables, the subscripts are often emitted. $\rho_{A,B}$ takes values in $[-1,1]$. Values close to 1 indicate a strong positive linear correlation between A and B, values close to -1 indicate a strong negative correlation and in case the value is 0, no correlation is indicated.

With that in mind, *we define a set of points to be elongated if*

$$|\rho| \geq thresh$$

In this context, each cluster is treated as a population and the features of the data set as random variables. $thresh \in [0,1]$ is an arbitrary threshold value that depends on our interpretation of the underlying problem. A visual representation of elongated clusters of points in the two dimensional plane along with the corresponding coefficients is shown in figure 1.2:. In each picture, 1000 points have been produced from the two-dimensional Gaussian distributions. Their elongated shapes are the result of different covariance matrices used in each case. In (a) through (d), ρ is positive and the sets exhibit positive correlation - the values in the vertical axis tend to grow as values of the horizontal axis grow. The greater the value of ρ, the stronger the correlation. In (e) through (h), negative ρ values indicate negative correlation with the smallest values corresponding to strong correlations.



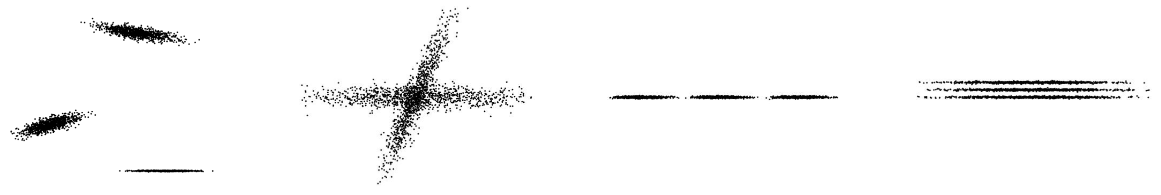
(a) $\rho = 0.50$ **(b)** $\rho = 0.80$ **(c)** $\rho = 0.90$ **(d)** $\rho = 0.98$

(e) $\rho = -0.50$ **(f)** $\rho = -0.80$ **(g)** $\rho = -0.90$ **(h)** $\rho = -0.98$

Figure 1.2: Elongated sets of points with the corresponding Pearson Coefficients.

The task of identifying elongated clusters in a set of points entails fundamentally new complications. As a brief example, four cases are depicted in figure 1.3:. The first case in (a) is rather straightforward. For the rest of the subfigures, different algorithms may fail to detect all the distinct segments and erroneously classify points in a single cluster. The intersection of clusters depicted in (b) may confuse some approaches, while in (c) and (d), it is difficult to be determined whether there are three distinct clusters or a single one, affected by noise.

In the following section, clustering approaches to elongated data are presented.



(a) A simple case (b) intersecting clusters (c) colinear clusters (d) parallel clusters

Figure 1.3: Four cases of elongated or linearly shaped clusters.

1.2.2 Related work

The literature of clustering methods suitable for elongated shaped data usually entails modifications of well studied algorithms or problems, such as Lloyd's or k-Means algorithm [16], [17]. In [18], Philips and Rosenfeld introduced one such variation able to fit a specified number of lines in a data set. They compute the principal axis for each cluster as an update step. Then, for each point x its perpendicular distance from the principal axis of each cluster is computed and x is assigned to the cluster corresponding to the minimum distance. Building upon this idea in [19], Yin proposed three improvements; one of those, titled "Algorithm B" in the article, suggests a fast approximation procedure for finding the principal axis for a set of points. In this work, a redefinition of this method, suitable for the context of probabilistic clustering, is used for the same purpose. From a similar point of view, another variant of k-Means was explored in [20], defining line segments as cluster representatives. The distances of the points to the segments are then calculated by a simple geometrical model and the update of the segments relies on principal component analysis of the assigned points and on their projections on segment axes. That geometric approach and the intuition behind the update mechanism is close to the algorithm proposed in the next chapter.

Under the fuzzy clustering taxonomy, a variation of the Fuzzy c-Means algorithm is introduced in [21], aptly titled "Fuzzy c-Lines". This algorithm uses parametric form of lines as cluster prototypes. The assignment step calculates the degrees of membership of points to clusters based on a notion of *point to line distance* and the parameters of each line are next updated via a mixture of weighted sums and eigenvector calculations.

Considering the family of possibilistic clustering algorithms, the SPCLS algorithm is introduced in [22], with an intended application in line detection, as part of image processing. SPCLS opts for representing elongated clusters by line segments, specifically by keeping track of the resulting endpoints. The algorithm is fitted under a general possibilistic sparsity framework, also introduced in the article. The term sparsity commonly refers to an algorithm's mechanism for being unaffected by distant points (such as outliers). The update of the line segments is carried out via a function optimization process and the procedure is also accompanied by a means of eliminating clusters that do not represent any point well enough.
The proposed algorithm of this thesis shares with SPCLS the following: a) the initialization stage – since it is suitable for elongated clusters, b) the use of line segment endpoints for representing the line segments (cluster representatives), c) an equivalent definition of the distance between a point and a line segment and d) the intend of eliminating unnecessary clusters (though the respective policies are not related).

Furthermore, a closely related method to the proposed in the present work can be found in [23]. There, an algorithm is proposed that also makes use of Mixture Models and the

Expectation-Maximization process for extracting line segments in a probabilistic framework. In order to completely define the optimization function, the authors use the affine transformation model, which is common in remote sensing applications. Their algorithm is proposed in the context of road extraction and experiments are applied using geospatial imagery.

The structure of the present study is as follows. Chapter 2 describes the EM-algorithm in the frame of the mixture decomposition problem. In Chapter 3 the proposed algorithm is fully analyzed, while in Chapter 4 extensive experimentation is conducted in order to assess its performance. Finally, Chapter 5 contains a discussion on the proposed method while some directions for further research are also provided.

2. EXPECTATION-MAXIMIZATION AND MIXTURE MODELS

Since the proposed method includes a variation of Expectation-Maximization (EM), this chapter is devoted to a brief presentation of this algorithm. Combined with the problem of Mixture Models, a methodology for applying EM in probabilistic clustering is also discussed.

2.1 The EM Algorithm

EM was first introduced in [24] and can be used for Maximum Likelihood Estimation (MLE) of parameters of a probability distribution function (pdf) given a set of observations. It is as an iterative optimization method that falls under the general category of gradient-based optimizers.

The EM algorithm is applied to problems in which the observed data set is (or can be considered as) *incomplete*. In detail, we assume a data set of *complete* samples $\boldsymbol{y} \in Y \subseteq R^{l_1}$ that was generated from an (unknown) multivariate probability distribution function $p_{\boldsymbol{y}}(\boldsymbol{y};\boldsymbol{\theta})$ that is parameterized by an unknown vector of parameters, $\boldsymbol{\theta}$. Those points however cannot be observed directly. Instead, we can observe a set of samples $\boldsymbol{x} \in X \subseteq R^{l_2}$, where $l_2 \leq l_1$. We assume that each $\boldsymbol{x}$ is related to a corresponding $\boldsymbol{y}$ by a relation $g : Y \subseteq R^{l_1} \rightarrow X \subseteq R^{l_2}$, so that $g(\boldsymbol{y}) = \boldsymbol{x}$. Under this framework, Y and X are often termed as the complete and incomplete data sets respectively.

The *likelihood* of Y is defined as:

$$p(Y;\boldsymbol{\theta}) = \prod_{i=1}^{N} p_{\boldsymbol{y}}(y_i;\boldsymbol{\theta}) \tag{2.1}$$

The intend is to find the maximum log-likelihood estimation of parameter $\boldsymbol{\theta}$ given the complete data:

$$\hat{\boldsymbol{\theta}}_{MLE} = arg \underset{\boldsymbol{\theta}\in\Omega}{max}\{log\ p_{\boldsymbol{y}}(\boldsymbol{Y};\boldsymbol{\theta})\} \tag{2.2}$$

In the above, Ω is the parametric space of $\boldsymbol{\theta}$ and the natural logarithm is used, since it is a monotonically increasing function, in order to convert the product of (2.1) to a sum of logarithms of probability distributions. This is extremely helpful when dealing with exponential distributions, such as the Gaussian or the one presented in section 3.2.2.

However, as stated before, Y is not observable. Therefore the likelihood of Y is not directly accessed and, as a consequence, the optimization problem in Equation (2.2) cannot be solved directly. Instead, the incomplete data set, X, is the only one at our disposal. Thus, a way out of this situation is to consider the expected value of the logarithmic likelihood function over the complete, unobserved data set, conditioned to the observed samples and our current estimation of the parameters, in the place of likelihood of Y (see Equation (2.3) below). Note that the former is a function of $\boldsymbol{\theta}$ and it has been given the name ”Q-function”. Here is the entrance point of EM, which performs iterative maximization on the above cost function.

The iterative process makes at $t = 0$ an initial guess about the value of $\boldsymbol{\theta}^{(0)}$ and then it repeats the following two steps:

- **E-step:** Using $\boldsymbol{\theta}^{(t)}$, calculate the conditional probability distribution $p_{\boldsymbol{y}}(Y|X;\boldsymbol{\theta}^{(t)})$ and then calculate:
$$Q(\boldsymbol{\theta};\boldsymbol{\theta}^{(t)}) = E_{Y|X,\boldsymbol{\theta}^{(t)}}[log\ p_{\boldsymbol{y}}(Y;\boldsymbol{\theta})] \tag{2.3}$$
- **M-step** Find the value of $\boldsymbol{\theta}^{(t+1)}$ that maximizes $Q(\boldsymbol{\theta};\boldsymbol{\theta}^{(t)})$. That is,
$$\boldsymbol{\theta}^{(t+1)} = arg\ \underset{\theta\in\Omega}{max}\{Q(\boldsymbol{\theta};\boldsymbol{\theta}^{(t)})\} \tag{2.4}$$
Assuming differentiability for Q, $\boldsymbol{\theta}^{(t+1)}$ is set equal to the solution of:
$$\frac{\partial Q(\boldsymbol{\theta};\boldsymbol{\theta}^{(t)})}{\partial\boldsymbol{\theta}} = 0 \tag{2.5}$$

The names of the steps come from the words Expectation and Maximization respectively. In the E-step, the value of Q-function does not affect the M-Step and so there is no need for explicitly computing it. During that step, the values that need to be calculated are problem-specific and they are the ones that play a role in the maximization process. The procedure stops when an appropriate condition is fulfilled; usually when $\|\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^{(t)}\| < \epsilon$, for some small $\epsilon > 0$ that denotes the tolerance of convergence. Citing results of theoretical studies on this method, such as the ones that can be found in [25], it is shown that a) iterations do not reduce the value of the likelihood function *(Monotonicity Theorem)* and b) the algorithm converges to a local maximum or a saddle point. This convergence result is typically adequate in many instances.

2.2 Mixtures Models and Clustering

A typical use case of EM is when a data set X of size N is assumed to have been generated by the effect of m distinct distributions so that each data point has been produced by one of them. In this case, the distribution (pdf) that generates the data is expressed as a weighted sum of distributions (pdfs):

$$p(\boldsymbol{x}_i) = \sum_{j=1}^{m} p_j(\boldsymbol{x_i}|j)Pr_j\ , \quad subject\ to\ \sum_{j=1}^{m} Pr_j = 1 \tag{2.6}$$

Due to its definition, this distribution is also known as a Mixture Model. Under this model, we may treat the sample X of data points as the incomplete data set. Subsequently, the complete data set can be expressed as:

$$Y = \{(\boldsymbol{x}_i, c_i) \mid \forall\boldsymbol{x}_i \in X,\ c_i \in \{1,...,m\}, i = 1,...,N\} \tag{2.7}$$

In that context, each c_i is a latent variable indicating the distribution that produced its associated $\boldsymbol{x}_i$. Under the above formalization, using the elementary property of conditional probabilities, we can deduct that:

$$p_{\boldsymbol{y}}(\boldsymbol{y}_i;\boldsymbol{\theta}) = p_{\boldsymbol{x}}(\boldsymbol{x}_i, c_i;\boldsymbol{\Theta}) = p_{\boldsymbol{x}}(\boldsymbol{x}_i|c_i;\boldsymbol{\theta})Pr_{c_i} \tag{2.8}$$

Pr_{c_i}, or simpler, Pr_j, for $j = 1,...,m$ denote the *a priori* probabilities of the distinct distributions of our model. Since we have no way of knowing the values of Pr_j, they are treated as part of the parameter vector, along with the specific parameters of each distribution.

For the rest of this work, $\boldsymbol{\theta}_j$ will be used to denote the parameters associated with the j-th distribution. Subsequently, $\Theta = [\boldsymbol{\theta}_1^T, \ ..., \ \boldsymbol{\theta}_m^T]^T$ will denote the concatenated vector of the parameters of all distributions. Similarly, vector $\boldsymbol{Pr} = [Pr_1, \ ..., Pr_m]^T$ will hold information about the *a priori* of all distributions.

The parameters that affect the Q-function are now both Θ and $\boldsymbol{Pr}$. The expectation is now over the space space of all possible distributions, c_j and as a result, utilizing Equation (2.8) the Q-function of Equation (2.3) for the problem of Mixture Models can be expressed as:

$$Q(\Theta, \boldsymbol{Pr}; \Theta^{(t)}, \boldsymbol{Pr}^{(t)}) = \sum_{i=1}^{N} \sum_{j=1}^{m} P(c_j|\boldsymbol{x}_i; \boldsymbol{\theta}_j^{(t)}) log(p(\boldsymbol{x}_i|c_j; \boldsymbol{\theta}_j^{(t)})Pr_j) \tag{2.9}$$

Assuming that all individual pdfs are of the same form, we have simplified the notation from $p_{\boldsymbol{x}}()$ to $p()$. Next, we compute $P(c_j|\boldsymbol{x}_i; \boldsymbol{\theta}_j^{(t)})$ using the extended form of the *Bayes Theorem*:

$$P(c_j|\boldsymbol{x}_i; \boldsymbol{\theta}_j^{(t)}) = \frac{p(\boldsymbol{x}_i|c_j; \boldsymbol{\theta}_j^{(t)})Pr_j^{(t)}}{\sum_{k=1}^{m} p(\boldsymbol{x}_i|c_k; \boldsymbol{\theta}_k^{(t)})Pr_k^{(t)}}, \ \forall i = 1, ..., M \ , \ \forall j = 1, ..., m \tag{2.10}$$

Hence, we can now derive more specific equations for the maximization step:

- Since we interpret Pr_j as probabilities, the process of finding $Pr_j^{(t)}$ that maximize the Q-function is now a constrain optimization problem with the constrains being:

$$\sum_{j=1}^{m} Pr_j = 1 \ \text{ and } \ Pr_j \geq 0, \ j = 1, ..., m \tag{2.11}$$

  Such a problem can be solved [26], resulting in the following closed form equation:

$$Pr_j^{(t)} = \frac{1}{N} \sum_{i=1}^{N} P(c_j|\boldsymbol{x}_i; \boldsymbol{\theta}^{(t)}) \tag{2.12}$$

- Finally, considering Equations (2.5) and (2.9), assuming $p()$ is differentiable and that $\boldsymbol{\theta}_i$, $\boldsymbol{\theta}_j$ $(i \neq j)$ are independent from each other, $\boldsymbol{\theta}_j$ can be computed by solving:

$$\sum_{i=1}^{N} \sum_{j=1}^{m} P(c_j|\boldsymbol{x}_i; \boldsymbol{\theta}^{(t)}) \frac{\partial}{\partial \boldsymbol{\theta}_j} log(p(\boldsymbol{x}_i|c_j; \boldsymbol{\theta}_j)) = \boldsymbol{0} \tag{2.13}$$

The above framework fits nicely with probabilistic clustering. More specifically, one can assume that each cluster is associated with a specific distribution and that data points are drawn by those distributions (one point from each distribution). Finding an optimal Θ value means that we have the optimal overall distribution, whose constituent pdfs represent the clusters. This procedure is often called a Mixture Decomposition. If needed, one can then use those pdfs (as they are defined utilizing the final values of the respective parameter vectors) to deterministically assign points to clusters (i.e. perform hard clustering) with the following profound rule:

$$\boldsymbol{x_i} \rightarrow c_j = arg \max_{j=1,...,m} \{Pr_j \ p(c_j|\boldsymbol{x_i}; \boldsymbol{\theta}_j)\} \ , \ i = 1, ..., N \tag{2.14}$$

A very common application of Mixture Decomposition is to model each cluster with a distinct multivariate Gaussian distribution. In this case, each $\boldsymbol{\theta}_j$ is comprised of the mean, μ_j, and covariance matrix, Σ_j of the corresponding pdf. It is worth noting here that this Gaussian Mixture Model (GMM), for the special case of covariance matrices of the form $\Sigma_j = \sigma^2 I$ – where I is the identity matrix, that result to spherical distributions, is very closely related to k-means clustering.

In [26], the Generalized Mixture Decomposition Algorithmic Scheme (GMDAS) which is actually the utilization of the EM and Mixture Models, in the frame of clustering is given. Expressed in pseudocode with necessary notation changes, GMDAS is shown in Algorithm 1:. The proposed in the present thesis method, described in the next chapter is also expressed using a modified version of this scheme.

Algorithm 1: GMDAS

```
Require: Data set $X$
Require: Initial values: $\Theta^{(0)}$, $Pr_1^{(0)}$, $Pr_2^{(0)}$, ... ,$Pr_m^{(0)}$
 1: $t \leftarrow 0$
 2: repeat
 3:
 4:    // E-Step
 5:    for $j \leftarrow 1\ to\ m$ do
 6:       for $i \leftarrow\ \ to\ N$ do
 7:          Compute $P(c_j|\boldsymbol{x}_i;\Theta^{(t)})$ from (2.10)
 8:       end for
 9:    end for
10:
11:    // M-Step
12:    for $j \leftarrow 1\ to\ m$ do
13:       Compute $\boldsymbol{\theta}_j^{(t+1)}$ by solving (2.13)
14:       Compute $Pr_j^{(t+1)}$ from (2.12)
15:    end for
16:
17:    $t \leftarrow t + 1$
18: until $||\Theta^{(t)} - \Theta^{(t-1)}|| < \epsilon$
```

3. THE PROBABILISTIC LINE SEGMENTS CLUSTERING (PLSC) ALGORITHM

3.1 Introduction

In this chapter, a novel probabilistic clustering algorithm for detecting line segments in images, called Probabilistic Line Segment Clustering (PLSC) is presented. The algorithm uses line segments (defined by their endpoints) to represent elongated clusters in the image. A two-dimensional probability distribution centered around a line segment is used to model each linear cluster and to quantify the membership between data points and clusters. It is expressed so as to loosely fit under the GMDAS, presented in Paragraph 2.2, while it also exhibits a number of additional key features.

For one, it is supposed to start with more clusters than those actually exist in the image. Using a pair of elimination policies (to be disclosed later in detail), the iterative scheme is able to identify false line segments and deal with them accordingly by either merging them together or by removing them. Secondly, the maximization step no longer applies in this framework in a strict sense. This is, in the one hand, due to the fact that since the number of clusters changes at certain iterations, there is no correspondence between the respective Q-functions and, on the other hand, due to difficulties in coming up with closed form solutions when attempting to find maximizing values of the parameter vector. Instead of directly finding parameter values via maximization of the expected log-likelihood of the underlying distributions, an approach similar to Least Square Error (LSE) regression is used in each iteration in order to reposition line segments. This approach of fitting line segments to points and then deciding whether merges should happen resulted in two procedures under the names of FIT and $MERGE$. In order not to deviate completely from the maximization spirit, during consecutive iterations in which the number of clusters remains the same, the set of segments that produced the best Q value is stored along with additional information. $MERGE$ then attempts to unify those stored segments rather than segments of the current iteration.

While PLSC can be used in any type of clustering problem, in this work, it is expressed in the context of detecting line segments in images. In particular, the data set is always assumed to be two dimensional, where each edge pixel is represented by its two spatial coordinates in the image. The edge pixels are identified by a process such as those described in the first chapter. This choice, while strict, allows for a more concise description of certain components such as the initialization, the optimization stage, as well as a clearest use of the required hyper-parameters.

The rest of this chapter continues as follows: Section 3.2 defines the probability distribution function to be used by the algorithm, Section 3.3 deals with the initialization process, Section 3.4 discusses relevant difficulties in performing the maximization step of EM and Section 3.5 proposes alternative update procedures. Finally, the algorithm is described in full in Section 3.6.

3.2 The Proposed Probability Distribution Function

In order for PLSC to fit under a Mixture Models framework, an appropriate probability distribution function (pdf) needs to be defined. Since that pdf incorporates elements from the Euclidean geometry, relevant prerequisites are presented first.

3.2.1 Point - Segment Distance

The aim of this paragraph is to provide a definition of the distance between a line segment and a point. That distance is central in formulating the probability distribution function of the following subsection.

Given a line segment, L, and a point $\boldsymbol{x} \in \mathbb{R}^2$ the *point-segment distance* of $\boldsymbol{x}$ from L is defined as: [1]

$$D(\boldsymbol{x}, L) = \underset{\boldsymbol{p} \in L}{min} \|\boldsymbol{x} - \boldsymbol{p}\|$$

That definition, however, is of no direct use from a computational point of view. Thus, the above distance is expressed in a more analytic form. At first, a few simple definitions are needed in order to build upon.

Given a line H expressed in Cartesian coordinates as: $ax + by + c = 0$, the distance of a point $\boldsymbol{x}(x_0, y_0)$ from H is defined as:

$$d(\boldsymbol{x}, H) = \frac{|ax_0 + by_0 + c|}{\sqrt{a^2 + b^2}} \tag{3.1}$$

When H passes through two known points $\boldsymbol{a}(x_1, y_1)$, $\boldsymbol{b}(x_2, y_2)$, it can be derived that

$$\begin{aligned} a &= y_2 - y_1 \\ b &= x_2 - x_1 \\ c &= x_1 y_2 - y_1 x_2 \end{aligned} \tag{3.2}$$

Substituting the above in (3.1), we obtain

$$d(\boldsymbol{x}, H) = \frac{|(y_2 - y_1)x_0 + (x_2 - x_1)y_0 + x_2 y_2 - y_2 x_1|}{\sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}} \tag{3.3}$$

Additionally, the orthogonal projection of a point, $\boldsymbol{x}'$ onto a line H, $\boldsymbol{x}'_H$ that passes through $\boldsymbol{a}$ and $\boldsymbol{b}$ needs to be defined. An algebraic notation will be used as it provides a simpler formula. Points are thus treated as vectors. H can be expressed through the equation $\boldsymbol{w}^T \boldsymbol{x} + w_0 = 0$, for $\boldsymbol{w} \in \boldsymbol{R}^2,\ w_0 \in \boldsymbol{R}$, and since it passes through $\boldsymbol{a}(x_1, y_1)$ and $\boldsymbol{b}(x_2, y_2)$, $\boldsymbol{w}$ and w_0 can be expressed as:

$$\boldsymbol{w} = [y_2 - y_1,\ x_2 - x_1]^T \tag{3.4}$$

$$w_0 = x_1 y_2 - y_1 x_2 \tag{3.5}$$

Then, the orthogonal projection of any point $\boldsymbol{x}'$ onto H, $\boldsymbol{x}'_H$, is given by the following expression:

$$\boldsymbol{x}'_H = proj_H\{\boldsymbol{x}'\} = \boldsymbol{x}' - \frac{\boldsymbol{w}^T \boldsymbol{x}' + w_0}{\|\boldsymbol{w}\|^2} \boldsymbol{w} \tag{3.6}$$

[1] Actually, this is the general definition of a point from an arbitrary set of points.

Finally, given a line segment $L(\boldsymbol{a}, \boldsymbol{b})$ and a point $\boldsymbol{x'}$ that belongs in the carrier line of L , $\boldsymbol{x'}$ belongs in L if and only if the following condition is true:

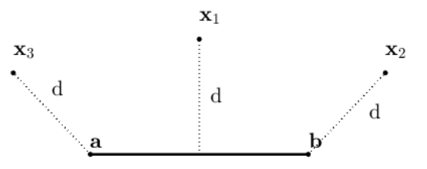
$$(\|\boldsymbol{x}'\| \leq \|\boldsymbol{a}\| \textbf{ AND } \|\boldsymbol{x}'\| \geq \|\boldsymbol{b}\|) \textbf{ OR } (\|\boldsymbol{x}'\| \geq \|\boldsymbol{a}\| \textbf{ AND } \|\boldsymbol{x}'\| \leq \|\boldsymbol{b}\|) \tag{3.7}$$

Using the above prerequisites, the distance between a point $\boldsymbol{x}$ and a line segment $L(\boldsymbol{a}, \boldsymbol{b})$ can be expressed as:

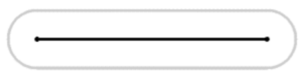
$$D(\boldsymbol{x}, L) = \begin{cases} d(\boldsymbol{x}, H_L) & \text{, if } \boldsymbol{x}_H \in L \\ min\{\|\boldsymbol{x} - \boldsymbol{a}\|\,,\ \|\boldsymbol{x} - \boldsymbol{b}\|\} & \text{, otherwise} \end{cases} \tag{3.8}$$

A geometrical illustration of the distance definition is presented in figure 3.1. More specifically, 3.1:a, helps illustrate the computational process implied by Equation (3.8). All three points have the same distance, d, from the segment, L, with endpoints $\boldsymbol{a}$ and $\boldsymbol{b}$. In detail, the projection of $\boldsymbol{x}_1$ lies onto the segment, so the first branch of Equation (3.8) is activated in this case. On the contrary, points $\boldsymbol{x}_2$ and $\boldsymbol{x}_3$ are not projected orthogonally onto L. In those cases, $D(\boldsymbol{x}_2, L)$ is computed as $\|\boldsymbol{x}_2 - \boldsymbol{b}\|$ and $D(\boldsymbol{x}_3, L)$ as $\|\boldsymbol{x}_3 - \boldsymbol{a}\|$, through the second branch of Equation (3.8).

Figure 3.1:b shows a contour line of the distance function from a line segment of fixed endpoints. (i.e. a curve in which all points have the same distance from the segment). The shape of the contour line slightly resembles an ellipse, without actually being one. That shape is referred to as a "cylinder" in [20], however "*capsuloid*" may constitute a more appropriate term.



(a) Three points of equal distance from a segment that comprise three distinct cases of Equation (3.8).



(b) A contour line of the distance seen as special a shaped, two-dimensional curve

Figure 3.1: Geometrical overview of the point to segment distance.

3.2.2 A Probability Distribution Centered Around Line Segments

The assumption under which this algorithm is to be used is that data points form elongated clusters. In order to assign meaningful probabilities of each data point belonging to each cluster, the underlying probability distribution needs to take into consideration that assumption. Thus, the proposed pdf peaks around a line segment and has an exponential decay around it, the rate of which can be parametrized.

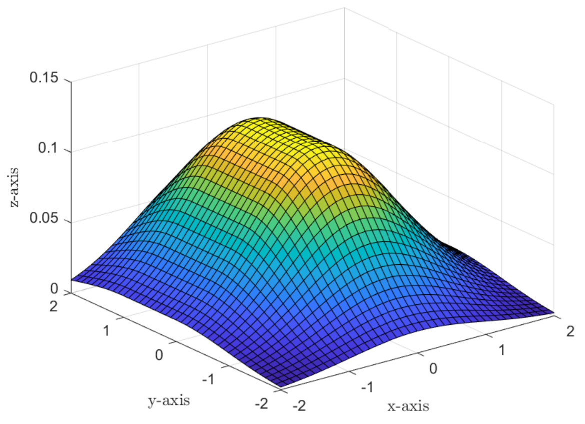
Given a point $\boldsymbol{x} \in \mathbb{R}^2$, a line segment $L(\boldsymbol{a}, \boldsymbol{b})$ and some $\sigma > 0$, the probability distribution function is defined as:

$$p(\boldsymbol{x}; L, \sigma) = \frac{1}{2\pi\sigma^2 + \sqrt{2\pi} l_L \sigma} exp(-\frac{1}{2\sigma^2} D^2(\boldsymbol{x}, L)) \tag{3.9}$$

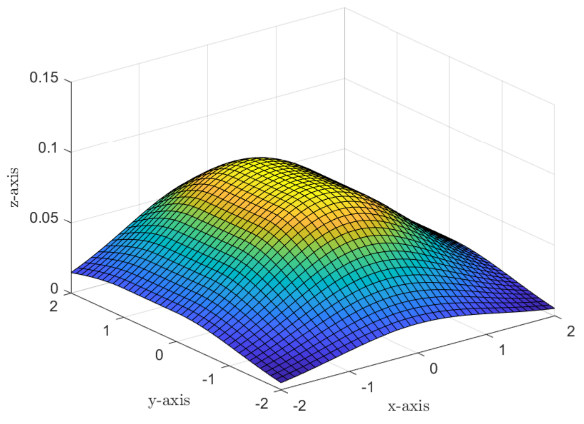
in which, l_L is the length of line segment L, namely $\|\boldsymbol{a} - \boldsymbol{b}\|$ and distance $D(L, \boldsymbol{x})$ is defined in Equation (3.8). The value of σ determines how abruptly the values of the probability decrease as points move away from the central segment. It a sense, it holds information about the variance of the distribution around the line segment. Since in the application of line detection, edge segments are commonly expected to have approximately equal widths, σ is treated as a hyper-parameter and its value is selected so that it prevents numerical underflow in the implementation of the algorithm. That underflow may arise in cases of distant points, when computing the exponent part of the equation.

The proof that $p(\boldsymbol{x}, L, \sigma)$ in Equation (3.9) actually represents a probability distribution function (i.e. its values are non negative and it integrates to 1) can be found in ANNEX I.

In Figure 3.2:, two surface plots are shown that map the values of a probability distribution centered around a line segments with endpoints at $(0, 0)$ and $(0, 1)$. In 3.2:a, σ is set to 1 while in 3.2:b the value of σ is 1.1, resulting in differences in both the maximum values of the respective probabilities and their slopes. Higher σ values lead to lower peaks and slower decays.



(a) $\sigma = 1$



(b) $\sigma = 1.1$

Figure 3.2: Surface plots of areas of two probability density functions centered around the same line segment and using different σ's.

An interesting property of this distribution is that when $\boldsymbol{a} = \boldsymbol{b}$, l_L becomes 0 and $D(\boldsymbol{x}, L)$ becomes $\|\boldsymbol{x} - \boldsymbol{a}\|$, allowing (3.9) to be rewritten as:

$$p(\boldsymbol{x}; L, \sigma) = \frac{1}{2\pi\sigma^2} exp(-\frac{1}{2\sigma^2} \|\boldsymbol{x} - \boldsymbol{a}\|^2) \tag{3.10}$$

which is the two-dimensional Gaussian Distribution for the special case where $\Sigma = \sigma^2 I_2$ [2] and $\boldsymbol{\mu} = \boldsymbol{a}$.

[2] I_2 denotes the 2×2 identity matrix.

3.3 Initialization and Number of Clusters

A crucial part of clustering algorithms that require the number of clusters, m, as a parameter, like EM, is how to find an appropriate m value for different problems. PLSC deals with that issue by starting with more clusters than those physically formed in the data and continues by either eliminating or merging together clusters until no further changes concerning the number of clusters can be performed. At this point, it is assumed that the algorithm has learned the correct number of clusters for a given data set. In order for that approach to work, initial line segments need to be positioned in a way so that there is at least one (and preferably more) segments near each physical cluster. In the special case of intersecting segments, a suitable initialization needs to ensure that there will be at least one initial segment in each side of each intersection.

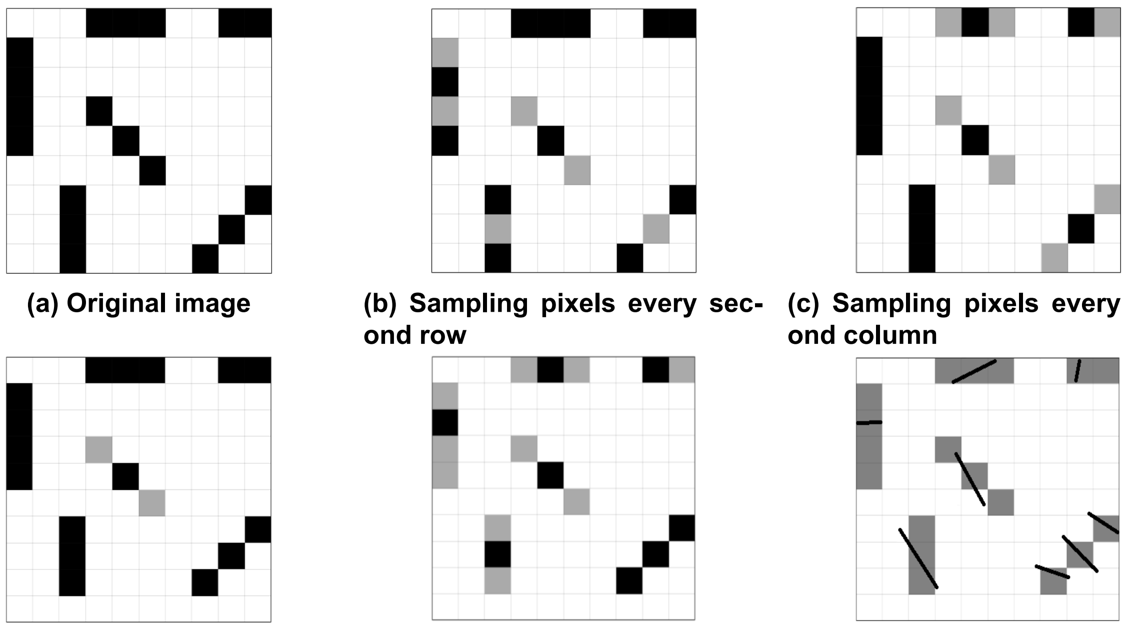
The quality of the results of PLSC greatly depends on whether the initialization procedure tasked with finding initial segments is able to fulfill the above conditions. A suitable initialization method that this work uses without any modification is discussed in [22], where a clustering algorithm for line segment detection in images is proposed that also contains a stage of eliminating false clusters. That method is applied explicitly on images consisted of binary edge pixels. It works by first sampling every T-th column and T-th row of the image (where T is a user-defined parameter). Edge pixels found by either process are marked in a new image with the same size as the original. After that, the new image is scanned both horizontally and vertically for existing line segments. Every time one is found, its corresponding pixels are erased apart from its midpoint. This process results to an image of isolated edge pixels. For each such point, a small line segment with random orientation and length is centered at those coordinates, representing a cluster. Note that the use of decreased or increased values of T results in more or fewer initial segments, respectively. While the initialization also uses another parameter, ρ to control the length of the generated segments, its value is left to $\sqrt{2}$, as proposed in [22].

An illustration of the initialization process is shown in figure 3.3: with $T = 2$. The original image is shown in (a) with black pixels on white background. Figure (b) is the result of sampling all pixels of every second row (shown in black). Non sampled pixels are shown in gray for reference purposes to the original image only. Similarly, in (c) the result of sampling pixels of every second column is displayed. In (d), pixels from (b) and (c) are added in an element-wise manner [3] and in (e) horizontal and vertical segments are located, having all their pixels removed apart from the middle one (or previous to middle in case of even sized segments). Initial segments are produced in (f) centered at the points left in (e). Their direction and size are random. Notice that multiple segments may be produced for the same actual segment. This was intended to happen. Instead, a problem would arise if a whole segment was skipped altogether by the sampling process.

3.4 Parameter Updates Under GMDAS

In order to maximize the value of the Q-function, EM needs to find at each iteration the value of $\Theta^{(t)}$ that maximizes the current instance of $Q(\Theta, \boldsymbol{Pr}; \Theta^{(t)}, \boldsymbol{Pr}^{(t)})$. This is normally carried out by computing the partial derivatives of $Q(\Theta, \boldsymbol{Pr}; \Theta^{(t)}, \boldsymbol{Pr}^{(t)})$ w.r.t. $\boldsymbol{\theta}_j$ and setting

[3]Since pixels are either black or white, their values are treated as binary with 1 indicating an edge pixel (black) and 0 a background pixel (white). In that sense, the *OR* operation is actually performed instead of addition.



(a) Original image **(b) Sampling pixels every second row** **(c) Sampling pixels every second column**

(d) Two previous images added together **(e) Shrinkage of vertical and horizontal segments to points** **(f) Initial random segments**

Figure 3.3: Initialization procedure of SPCLS and PLSC

them equal to 0, as shown in Equation (2.5). In the case of PLSC, the parameters concerning line segments are treated as a set of segments $C = \{L_1(\boldsymbol{a}_1, \boldsymbol{b}_1), ..., L_m(\boldsymbol{a}_m, \boldsymbol{b}_m)\}$ that represent the respective clusters. Subsequently, since a line segment is defined via four parameters, the coordinates of its endpoints, the parameter vector, $\boldsymbol{\Theta}$, is defined as :

$$\boldsymbol{\Theta} = [\boldsymbol{a}_1^T,\ \boldsymbol{b}_1^T,\ ...,\ \boldsymbol{a}_m^T,\ \boldsymbol{b}_m^T]^T \in \mathbb{R}^{4m} \tag{3.11}$$

Those two notations may be used interchangeably when referring to line segments for the rest of the chapter as the vector representation is useful in equations while sets facilitate algorithmic operations. The specific form of the Q-function, derived from Equation (2.9) and using the newly defined pdf of Equation (3.9), can now be fully expressed as:

$$Q(\boldsymbol{\Theta}, \boldsymbol{Pr}; \boldsymbol{\Theta}^{(t)}, \boldsymbol{Pr}^{(t)}) = \sum_{i=1}^{N}\sum_{j=1}^{m} P(c_j|\boldsymbol{x}_i; \boldsymbol{\theta}_j^{(t)})\Big(-log(2\pi\sigma^2 + \sqrt{2\pi}l_{L_j}\sigma) - \frac{1}{2\sigma^2}D^2(L_j, \boldsymbol{x_i}) + logPr_j\Big) \tag{3.12}$$

Attempting to take the appropriate partial derivatives of each coordinate-element of $\boldsymbol{\Theta}$ does not provide with closed form solutions of the maximizing Equation (2.13) [4] . This is a setback since convergence analysis of the EM and the *Monotonicity Theorem* both assume that at each time step, t, $\boldsymbol{\Theta}^{(t)}$ has values that maximize $Q(\boldsymbol{\Theta}; \boldsymbol{\Theta}^{(t-1)}, \boldsymbol{Pr}^{(t-1)})$.

One naive solution would be to use a global optimization method for high dimensional vector spaces such as DIRECT [27], Stochastic Gradient Descent [28], or its more recent variant, Adam [29]. There are however inherent problems with the use of such techniques, namely their prohibitive computation time and the extensive parameter tuning required. This solution is, for all practical purposes, unfeasible, as PLSC would be required to solve an iterative optimization problem in high dimensions for each iteration of the GDMAS.

Finally, it is worth mentioning that even if computing optimal $\boldsymbol{\Theta}^{(t)}$ values was possible, a strict maximization process still cannot be carried out. This is because the algorithm

[4]Another problem with the M step is that $Q(\boldsymbol{\Theta})$ is not differentiable in all of its domain. This second implication could be overcome, but another direction is taken for that step altogether.

intends to eliminate clusters at certain iterations, resulting to $m^{(t_1)} < m^{(t_2)}$ for some $t_1 > t_2$. Since the vector space of $\mathbf{\Theta}^{(t)}$, $\Omega^{(t)} \subseteq \mathbb{R}^{4m^{(t)}}$ depends on the number of clusters of the current iteration, it is also the case that $\Omega^{(t_1)} \not\equiv \Omega^{(t_2)}$. As a result, comparing values of Q-functions that have the dimensions of their domains changing during the iterative process is nonsensical.

Because of those two facts, PLSC necessarily deviates from the classical EM algorithm, as far as the maximization step is concerned. The Expectation step and the part of the M-step that focuses on updating the prior probabilities are still performed but updates of $\mathbf{\Theta}$ are being carried out using a heuristic approach. The relevant procedures are presented in the next paragraph.

3.5 FIT and MERGE Procedures

FIT and $MERGE$ procedures substitute the maximization step. The first heuristic repositions the segments so as to best fit the data points, according to their a posteriori probabilities of belonging to the clusters, while the second reduces decisively the number of detected clusters.

The parts of those two processes are discussed in detail in the next subsections. PLSC uses the related algorithms as subroutines. During each iteration, PLSC calls FIT first. Then, if no clusters are eliminated due to this procedure (see 3.5.2), the $MERGE$ step can be applied in order to try to find segments that can be merged together. It must be noted that even though those heuristic seem intuitive, there is no theoretical guarantee that they optimize criteria, neither that they force the algorithm to converge in every possible case. Experimental results, however, indicate that the methods described next have a strong proposition of value.

3.5.1 Defining Line Segments from Data Points

While the problem of finding the line that best fits a set of data is often solved using some form of LSE fitting, the approach adopted in the present framework follows a slight modification of a method proposed in [19], titled *Algorithm B* because of its execution time: While the time complexity of both strategies for a set of n points is $\Theta(n)$, *Algorithm B* involves fewer calculations. More crucially, it is also able, once the line has been computed and a single point is either added or removed from the set of interest, to reconfigure the line in constant time.

Algorithm B works by arbitrarily splitting a set of n points, S, into two sets of equal size (or almost equal in the case n is an odd number). Those sets are given the names S_f and S_r and for simplicity it is assumed to contain $n_f = \lceil \frac{n}{2} \rceil$ and $n_r = \lfloor \frac{n}{2} \rfloor$ points, accordingly. The centroid vectors $\boldsymbol{c}_f$, $\boldsymbol{c}_r$ of those sets are computed as:

$$\boldsymbol{c_f} = \frac{1}{n_f} \sum_{\boldsymbol{x} \in S_f} \boldsymbol{x} \tag{3.13}$$

and

$$\boldsymbol{c_r} = \frac{1}{n_r} \sum_{\boldsymbol{x} \in S_r} \boldsymbol{x} \tag{3.14}$$

Then, line H is defined by these two points and can be expressed from the Equation (3.2).

In PLSC, three modifications of this technique are introduced. The first one concerns the splitting policy of points into S_f and S_r. PLSC opts for sorting the points in S with regard to their values of the axis (x or y) of the greatest variance. When this is done, the first half of the points is assigned to S_f, while the second half constitutes S_r. Note that this change comes at the cost of computational time due to the sorting involved.

The second modification is simply a distinction between probabilistic and hard clustering. In the original algorithm, the hard clustering philosophy is followed and set S contains all the data points that have been assigned to a given cluster. In the GMDAS however, no explicit assignment of points to clusters is performed. Because of that, for a given cluster c_j, the set S_j is defined to be the set of points of the data set whose maximum posterior probability is given for c_j. For the rest of the work, the term "*cluster points*" is used excessively when referring to S. Formally:

$$S_j = \{\boldsymbol{x} \in X \mid arg \max_{k=1,...,m} \{p(c_k|\boldsymbol{x})\} = j\} \quad j = 1, ..., m \tag{3.15}$$

The final change allows for weights to be used when determining the exact orientation of each line. The posterior probabilities act as a weight for each data point. Since in probabilistic clustering it is assumed that every point potentially belongs to any cluster, using S_j alone for finding the optimal line would be restrictive. Instead, the entire data set, X is used to compute $\boldsymbol{c_{f,j}}$, $\boldsymbol{c_{r,j}}$. This is carried out via a two-step procedure. More specifically, $\boldsymbol{c_{f,j}}$, $\boldsymbol{c_{r,j}}$ are initially estimated via Equations (3.13) and (3.14), where S_f, S_r are defined as described before. Let $\boldsymbol{c_{f,j}^{ini}}$, $\boldsymbol{c_{r,j}^{ini}}$ denote the values of $\boldsymbol{c_{f,j}}$, $\boldsymbol{c_{r,j}}$ respectively, as they are computed by Equations (3.13) and (3.14) . Then, based on these estimations, every data point, $\boldsymbol{x}_i \in X$, is classified to one of two sets according to the following nearest neighbor rule:

$$\boldsymbol{x_i} \in \begin{cases} X_{f,j} & , \ if \ \|\boldsymbol{x}_i - \boldsymbol{c_{f,j}^{ini}}\| \leq \|\boldsymbol{x}_i - \boldsymbol{c_{r,j}^{ini}}\| \\ X_{r,j} & , \ otherwise \end{cases} \tag{3.16}$$

Then the centroids are recomputed via the weighted averages of those sets:

$$\boldsymbol{c_{f,j}} = \frac{\sum_{\boldsymbol{x} \in X_{f,j}} p(c_j|\boldsymbol{x})\boldsymbol{x}}{\sum_{\boldsymbol{x} \in X_{f,j}} p(c_j|\boldsymbol{x})} \tag{3.17}$$

$$\boldsymbol{c_{r,j}} = \frac{\sum_{\boldsymbol{x} \in X_{r,j}} p(c_j|\boldsymbol{x})\boldsymbol{x}}{\sum_{\boldsymbol{x} \in X_{r,j}} p(c_j|\boldsymbol{x})} \tag{3.18}$$

It is noted now that the hyperplane H is defined by $\boldsymbol{c_{f,j}}$ and $\boldsymbol{c_{r,j}}$

The final part of the fitting process is to determine the endpoints of each line segment. This can be accomplished by first projecting all points of S_j onto H. In case H is not a vertical line (not perpendicular to the x axis), the endpoints can be located as the projected points with the lowest and highest x coordinate. If the line is vertical, the endpoints are simply the projected points with the lowest and highest y coordinate.

The exact algorithm for finding the best fit line segment for a given cluster, c, is given in Algorithm 2:. It takes as input the complete data set X, cluster points S and the posterior probabilities, $P(c|x_i)$, for each data point belonging to that cluster. It returns points $\boldsymbol{a}, \boldsymbol{b}$ that constitute the endpoints of the computed line segment. This algorithm is to be used at each iteration of PLSC for estimating each line segment.

Algorithm 2: FIT procedure for segment of cluster c

```
Require: $X = \{x_1, ..., x_N\}$
Require: $S = \{s_1, ..., s_k\}$
Require: $P(c|x_i)$ , $i = 1, ..., N$
 1: // Create $S_f$ and $S_r$ sets based on the sorted data data set
 2: $\bar{x} \leftarrow \frac{1}{k}\sum_{i=1}^{k} s_{i,x}$   ,   $\bar{y} \leftarrow \frac{1}{k}\sum_{i=1}^{k} s_{i,y}$
 3: $\sigma_x \leftarrow \sqrt{\sum_{i=1}^{N}(x_{i,x} - \bar{x})^2}$   ,   $\sigma_y \leftarrow \sqrt{\sum_{i=1}^{N}(x_{i,y} - \bar{y})^2}$
 4: if $\sigma_x > \sigma_y$ then
 5:    Sort all $s_i$ by their x coordinates.
 6: else
 7:    Sort all $s_i$ by their y coordinates.
 8: end if
 9: $n_f \leftarrow \lceil \frac{k}{2} \rceil$   ,   $n_r \leftarrow \lfloor \frac{k}{2} \rfloor$
10: $S_f \leftarrow \{s_i \mid i = 1, ..., n_f\}$   ,   $S_r \leftarrow \{s_i \mid i = n_f + 1, ..., k\}$
11:
12: Compute $c_f^{ini}, c_r^{ini}$ from Eq. (3.13), Eq. (3.14).
13:
14: // Split the entire data set according to the points' proximity to centroids
15: $X_f \leftarrow \emptyset$   ,   $X_r \leftarrow \emptyset$
16: for $i = 1$ $to$ $N$ do
17:    if $\|x_i - c_f^{ini}\| \leq \|x_i - c_r^{ini}\|$ then
18:       $X_f \leftarrow X_f \cup \{x_i\}$
19:    else
20:       $X_r \leftarrow X_r \cup \{x_i\}$
21:    end if
22: end for
23:
24: // Define principal axis with respect to the new, weighted centroids
25: Compute $c_f$, $c_r$ from Eq. (3.17), (3.18).
26: Define $H$ from Eq. (3.2) based on $c_f, c_r$ .
27:
28: // Project cluster points onto line to determine endpoints
29: for $i = 1$ $to$ $k$ do
30:    $s'_i \leftarrow proj_H\{s_i\}$
31: end for
32: if $H$ is not vertical then
33:    $a \leftarrow arg \min_{i=1,..,k}\{s'_{i,x}\}$
34:    $b \leftarrow arg \max_{i=1,..,k}\{s'_{i,x}\}$
35: else
36:    $a \leftarrow arg \min_{i=1,..,k}\{s'_{i,y}\}$
37:    $b \leftarrow arg \max_{i=1,..,k}\{s'_{i,y}\}$
38: end if
39:
40: return $a, b$
```

3.5.2 Implicit Cluster Elimination Mechanism

By design, FIT procedure also helps in eliminating clusters that fail to represent adequately any part of the edge image. This takes place before the execution Algorithm 2:. It is remarked that, in order for probabilistic line fitting to be performed for a given cluster c_j, the set S_j as defined in Equation (3.15) must contain at least two data points. Otherwise, no endpoints of the line segment can be determined. When $|S_j| < 2\ ,\ j = 1, ..., m$, Algorithm 2: fails to calculate the endpoints. Because of that c_j must be eliminated. That means that at each iteration, before the FIT step is executed, all clusters with less than two cluster points are removed. Interpreting the above condition, it is equivalent to say that *a cluster c_j is eliminated if there is at most one data point that belongs with higher probability to c_j than to any other cluster, c_k, for $j, k = 1, ..., m$.* Such clusters are by definition superfluous and their elimination under the probabilistic framework is intuitively justified.

3.5.3 Merging Clusters

Another mechanism of PLSC is the one that decides on the merging of two line segments. Such segments must be both *"sufficiently close"* and *"sufficiently collinear"*. In effect, using this mechanism is instrumental to the general policy of PLSC of starting with an overestimation of the true number of clusters and continue by reducing it until it is no longer possible to do so.

At first, it is important to define the terms "*sufficiently close*" and "*sufficiently collinear*". Two line segments $L_1(\boldsymbol{a}_1, \boldsymbol{b}_1)$, $L_2(\boldsymbol{a}_2, \boldsymbol{b}_2)$ are considered sufficiently close if:

$$min\{D(L_1, \boldsymbol{a}_2), D(L_1, \boldsymbol{b}_2), D(L_2, \boldsymbol{a}_1), D(L_2, \boldsymbol{b}_1)\} \leq \epsilon_{gap} \tag{3.19}$$

for a user-specified parameter ϵ_{gap}. In the context of segment detection in images, this hyper-parameter is expressed in terms of pixels and thus, it is expected to be easy for an appropriate value to be determined. Such value must be less than the minimum distance between pixels of different actual segments of the input image. The notation $D(L_1, L_2)$ is used for denoting that distance.

Continuing, L_1, L_2 are considered sufficiently collinear if:

$$|sim\_cos(L_1, L_2)| \geq \epsilon_{ang} \tag{3.20}$$

where $sim\_cos(L_1, L_2)$ is the cosine similarity expressed in terms of line segments $L_1(\boldsymbol{a}_1, \boldsymbol{b}_1)$, $L_2(\boldsymbol{a}_2, \boldsymbol{b}_2)$ rather than vectors. It is computed by simply applying the cosine similarity to the vectors that occur when subtracting the endpoints of each segment:

$$sim\_cos(L_1, L_2) = \frac{(\boldsymbol{b}_1 - \boldsymbol{a}_1) \cdot (\boldsymbol{b}_2 - \boldsymbol{a}_2)}{\|\boldsymbol{b}_1 - \boldsymbol{a}_1\|\ \|\boldsymbol{b}_2 - \boldsymbol{a}_2\|} \tag{3.21}$$

In Equation (3.20), ϵ_{ang} is a hyper-parameter that represents the lowest angular difference allowed for two segments to be considered collinear. Usually, good values for ϵ_{ang} tend to reside in $[0.95, 1]$.

The merging policy can be simply worded as: *Clusters c_i, c_j, represented by $L_i(\boldsymbol{a}_i, \boldsymbol{b}_i)$, $L_j(\boldsymbol{a}_j, \boldsymbol{b}_j)$ respectively are chosen for merging if a) L_j is the segment closest to L_i **and** b) L_i, L_j are sufficiently collinear **and** c) L_i, L_j are sufficiently close.* Let it be noted that a single cluster may be a candidate for merging (i.e. fulfills all of the above properties) in more than one pairs. The merging policy will simply merge only the first of those pairs.

The algorithm for determining possible merges given a set of clusters C, represented by their line segments, is presented in Algorithm 3:. It returns a set of clusters containing newly merged clusters.

The merging itself is simply the process of returning a new line segment, $L(\boldsymbol{a}, \boldsymbol{b})$ that fits well with regard to the data points contained in the clusters represented from both of the original segments. For that purpose, the Fit procedure is applied whenever a merge happens.

The implementation uses a set of indices, called $MERGED$, in order to keep track of which clusters have already been merged. For each cluster, c_j, that has not been yet merged with any other, it checks if any other, not yet merged, cluster satisfies both conditions (3.19) and (3.20). From those that do, it finds the one with the minimum distance from c_j, say c_q, and then merges c_j and c_q together. This produces a new segment L as described above and the algorithm adds it to the set of clusters to be returned, C'. After all clusters have been examined, it iterates them once again in order to append to C' all of the original clusters that have not been affected by merging.

Let it be noted that this implementation does not find necessarily optimal merges, i.e. those segments that have the highest angular similarity or affinity. It rather focuses on finding ”adequate” merges without increasing the computational complexity. The rationale behind this decision is that in case where there are many candidates for merging, then all of them together ultimately constitute a single line segment and so, the ordering of the merges is not important – candidates may still be merged in future iterations, assuming that segments do not deviate substantially at each iteration.

A final point to be made is that in the context of PLSC, $MERGE$ considers for merging the segments that have produced the best value of Q-function (for the current number of clusters) and not (necessarily) the segments of the current iteration. This makes the algorithm more stable as it will not attempt to merge segments from an iteration in which the segments are not fitted accurately enough on the data points.

Algorithm 3: MERGE procedure

```
Require: $\epsilon_{gap}, \epsilon_{ang}$
Require: $C = \{L_1(a_1, b_1), ..., L_m(a_m, b_m)\}$
Require: $X = \{x_1, ..., x_N\}$
Require: $P(c_j|x_i)$ , $i = 1, ..., N$ , $j = 1, ..., m$
 1: $MERGED \leftarrow \{\}$
 2: $C' \leftarrow \emptyset$ , $Pr' \leftarrow \emptyset$
 3: for $j = 1\ to\ m$ do              // for each cluster
 4:    if $j \notin MERGED$ then   // as long as it has not already been merged
 5:       $min\_dist \leftarrow \infty$
 6:       $k^* \leftarrow -1$
 7:       for $k = 1\ to\ m$ do        // for each potential pair
 8:          if $j \neq k$ and $k \notin MERGED$ then
 9:
10:             // Check for sufficiently close and sufficiently collinear conditions
11:             if $|sim\_cos(L_j, L_k)| \geq \epsilon_{ang}$ and $D(L_j, L_k) \leq \epsilon_{gap}$ then
12:                $min\_dist \leftarrow min\{min\_dist, D(L_j, L_k)\}$
13:                // Save closest segment
14:                if $min\_dist = D(L_j, L_k)$ then
15:                   $k^* \leftarrow k$
16:                end if
17:             end if
18:
19:          end if
20:       end for
21:       if $k^* \neq -1$ then           // pair found – perform merging
22:          Compute $S_j$,$S_{k*}$ from (3.15).
23:          $S_{new} \leftarrow S_j \cup S_{k*}$
24:          $a, b \leftarrow FIT(X, S_{new}, P)$
25:          $C' \leftarrow C' \cup \{L(a, b)\}$
26:          $MERGED \leftarrow MERGED \cup \{j, k^*\}$
27:       end if
28:    end if
29: end for
30:
31: // Append all unaffected clusters to returning structures
32: for $j = 1\ to\ m$ do
33:    if $j \notin MERGED$ then
34:       $C' \leftarrow C' \cup \{L_j(a_j, b_j)\}$
35:    end if
36: end for
37: return $C'$
```

3.6 PLSC Algorithm: The Complete Description

At this point, all aspects of the proposed method have been discussed. Written in pseudocode notation, PLSC is presented in algorithmic form in Algorithm 4:. It is assumed that the processes related to locating edge pixels of the input image and finding initial segments have been carried out prior to the execution of PLSC. To avoid unnecessary repetitions and to confine the length of the code, equations used throughout the algorithm are referenced and procedures FIT and $MERGE$ are called. The initialization policy is an important but not an integral part of this method and since it is given in detail in [22], it is not included.

The algorithm starts every loop by computing the posterior probabilities of data points over the clusters. Then the value of Q-function is computed. If either the number of clusters was reduced in the previous iteration or the current value of Q is the highest found so far (for the current number of clusters), current segments and probabilities are stored. The iteration then proceeds to the FIT Step. Cluster points for each cluster are computed and those clusters that have at least two cluster points have their segments repositioned using the FIT procedure. The rest are eliminated. The algorithm keeps track of how many clusters are left after the completion of this procedure. If no clusters were eliminated in that step, PLSC applies the $MERGE$ procedure in order to identify and merge suitable segments. In case $MERGE$ was able to unify segments, the set of segments for the next iteration is substituted by the set C' resulting from the $MERGE$ call.It is important to note here that the segments that are considered for merging are the ones that produced the highest Q value for the current number of clusters. Because of that, if one or more clusters were eliminated during the previously applied FIT procedure, the saved segments that produced the highest Q value are no longer valid, and so the $MERGE$ does not apply. Finally, the prior probabilities of the clusters for the next iteration are computed using the general closed form formula for the Mixture Decomposition and the termination criterion is checked. If it is not met, the process proceeds onto the next iteration.

The iterative scheme is terminated when either of the following happens:

a) Coordinates of cluster segments during two consecutive iterations (were the number of clusters was not reduced) vary by an amount less than ϵ. This indicates convergence. That variance is expressed in terms of the L_1 metric of the difference between $\Theta^{(t-1)}$ and $\Theta^{(t)}$ and convergence is implied when its value is less than the respective user defined tolerance, ϵ. The choice of L_1 is due to the fact that it is affected equally by both larger and close to zero values.

b) Iterations exceed a user defined threshold. This is used as a remedy to the fact that convergence is not guaranteed.

Upon termination, C^* contains the endpoints of segments that maximized the Q-function for the final value of m, m^*. The posterior probabilities are stored in P^*.

Algorithm 4: Probabilistic Line Segments Clustering (PLSC)

```
Require: $X, C^{(0)}, \sigma, \epsilon, \epsilon_{gap}, \epsilon_{ang}, max\_iters$
 1: $N \leftarrow |X|$ , $m^{(0)} \leftarrow |C^{(0)}|$
 2: $Q^* \leftarrow -\infty$ , $m^* \leftarrow \infty$
 3: $t \leftarrow 0$
 4: $Pr_j^{(0)} \leftarrow 1/m$ , $j = 1, ..., m^{(0)}$
 5: $P^{(0)}(c_j|x_i) \leftarrow 0$ , $i = 1, ..., N,\ j = 1, ..., m^{(0)}$
 6: repeat
 7:   for $i = 1\ to\ N$ do        // E-Step
 8:     for $j = 1\ to\ m^{(t)}$ do
 9:       Compute $P^{(t)}(c_j|x_i)$ from Eq. (2.10) using $\Theta^{(t)}, X, Pr^{(t)}, \sigma$.
10:     end for
11:   end for
12:
13:   Compute $Q^{(t)}$ from Eq. (3.12) using $\Theta^{(t)}, X, Pr^{(t)}, P^{(t)}, \sigma$.        // Save best Q value
14:   if $Q^{(t)} > Q^*$ or $m^{(t)} < m^*$ then
15:     $Q^* \leftarrow Q^{(t)}$
16:     $C^* \leftarrow C^{(t)}$ , $m^* \leftarrow m^{(t)}$
17:     $P^* \leftarrow P^{(t)}$
18:   end if
19:
20:   $C^{(t+1)} \leftarrow \emptyset$ , $m^{(t+1)} \leftarrow 0$        // $FIT$ Step
21:   for $j = 1\ to\ m^{(t)}$ do
22:     Compute $S_j^{(t)}$ from Eq. (3.15) using $X, P^{(t)}$.
23:     if $|S_j^{(t)}| \geq 2$ then
24:       $a_j^{(t+1)}, b_j^{(t+1)} \leftarrow FIT(X, S_j^{(t)}, P^{(t)}(c_j|X))$.
25:       $C^{(t+1)} \leftarrow C^{(t+1)} \cup \{L_j^{(t+1)}(a_j^{(t+1)}, b_j^{(t+1)})\}$
26:       $m^{(t+1)} \leftarrow m^{(t+1)} + 1$
27:     end if
28:   end for
29:
30:   if $m^{(t+1)} = m^{(t)}$ then        // No eliminations due to $FIT$ - proceed to $MERGE$ Step
31:     $C' \leftarrow MERGE(\epsilon_{gap}, \epsilon_{ang}, C^*, X, P^*)$
32:     if $|C'| < m^{(t)}$ then        // Some clusters were merged successfully
33:       $C^{(t+1)} \leftarrow C'$        // Update clusters of next iteration
34:       $m^{(t+1)} \leftarrow |C'|$
35:     end if
36:   end if
37:
38:   for $j = 1\ to\ m^{(t+1)}$ do        // Maximization of a priori probabilities
39:     Update $Pr_j^{(t+1)}$ from Eq. (2.12) using $P^{(t)}, N$.
40:   end for
41:
42:   $t \leftarrow t + 1$
43: until $(m^{(t)} = m^{(t-1)}$ and $\|\Theta^{(t)} - \Theta^{(t-1)}\|_1 < \epsilon)$ or $t \geq num\_iters$
```

Note that the two ways of representing line segments, i.e $C = \{L_1(\boldsymbol{a}_1, \boldsymbol{b}_1), ..., L_1(\boldsymbol{a}_m, \boldsymbol{b}_m)\}$ and $\boldsymbol{\Theta} = [\boldsymbol{a}_1^T, \boldsymbol{b}_1^T, ..., \boldsymbol{a}_m^T, \boldsymbol{b}_m^T]^T$, are treated as equivalent notations and used interchangeably in the algorithm. A full description of the variables used can be found in the next tables: In Table (3.1:) input variables along with user parameters are defined and explained appropriately and, similarly, explanation of internal variables of PLSC is presented in Table (3.2:).

Table 3.1: Input variables and user parameters of PLSC

X	The set of edge pixels represented by their spatial coordinates of the input image.
$C^{(0)}$	Initial set of pairs of endpoints of cluster segments, $C = \{(a_1, b_1), ..., (a_m, b_m)\}$.
σ	The expansion term of the defined pdf.
ϵ	Tolerance of variance when checking the termination condition.
ϵ_{gap}	Tolerance of segment proximity during Merge procedure.
ϵ_{ang}	Tolerance of segment collinearity during Merge procedure.
max_iters	Maximum number of iterations the algorithm may take.

Table 3.2: Internal variables of PLSC

$\bullet^{(t)}$	Value of any variable during the t-th iteration.
N	The size of X i.e. number of edge pixels.
m	Number of clusters.
Q^*	Highest value of Q-function for a certain value of m.
t	Iteration counter.
Pr	Vector of prior probabilities of clusters.
P	$N \times m$ array with cell i, j being the posterior probability that the data point x_i belongs to cluster c_j, i.e. $P(x_i \vert c_j)$.
i	Counter of data points – to be used while looping.
j	Counter of clusters or segments – to be used while looping.
$Q^{(t)}$	Value of Q-function at the t-th iteration.
C^*	Set of segments that produced the highest Q value.
P^*	Posterior probabilities that produced the highest Q value.
m^*	Number of clusters that produced the highest Q value.
S_j	Set containing data points of cluster c_j.
a_j, b_j	Endpoints of j-th segment.
C'	Set of segments that was returned by a call $MERGE$ procedure.
Θ	The vector containing the coordinates of the endpoints of line segments.

The source code for a MATLAB® implementation is provided in ANNEX II. The implementation is not focused on strictly transferring the algorithm into a programming language. Instead, it is split in multiple code files while using the object oriented programming paradigm so as to allow for ease in maintenance, experimentation, and future modifications.

4. EXPERIMENTAL RESULTS

This section focuses on the evaluation of the performance of the proposed algorithm (PLSC), based on a number of experimental data sets. Most sets are either artificially generated images or hand drawn pictures. A synthetic data set is also used as an example while an image resulted from an observation in the area of solar physics is presented for showcasing the application of the algorithm in different scientific domains. Some of the data sets are used for comparing PLSC against other approaches in literature. Such a comparison has been published in [22]. With the author's consent, original imagery and resulting pictures from the article are also displayed, allowing for side by side comparisons of their respective output images. Finally, failure or less successful cases are given and discussed appropriately.

Algorithms used as benchmarks include a variation of Hough Transform (HT) [30] for segment detection as discussed in [22], the Fuzzy c-lines (FCL) [21], as well as SPCLS algorithm [22]. The experiments are next enumerated and discussed accordingly, accompanied by figures showing input images and output line segments for each algorithm.

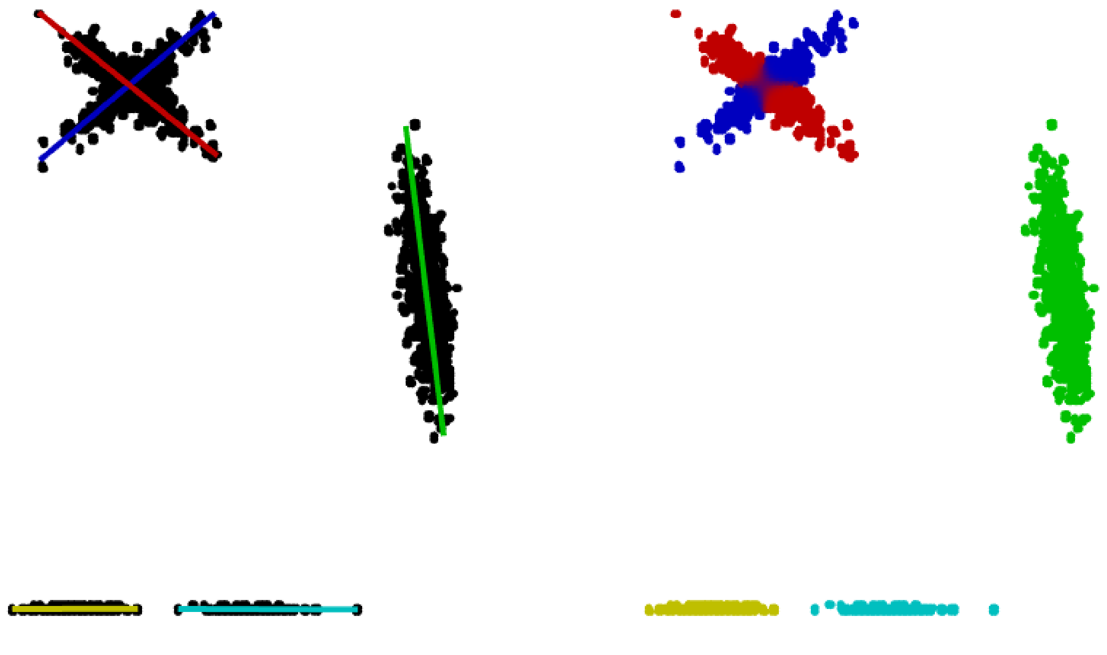
4.1 Experiments

We consider first an experiment involving an artificially generated data set. This is useful for illustrating the probabilistic nature of PLSC as well as how segments are fitted onto clusters. The next experiment involves a relatively simple configuration of line segments, in order to give a first ”flavor” of the performance of the algorithm on images, before we proceed with the comparison with the above mentioned algorithms.

Experiment 1 – A synthetic data set comprised of a mixture of samples from different Gaussian probability distributions. Points of figure 4.1: have been produced by five distinct normal distributions, with different means and covariance matrices. PLSC is applied to that mixed data set and the resulting segments are drawn and colored in figure 4.1:a. The carriers of those segments approximate closely the principal directions, along which the data points of the respective clusters are spread. Additionally, this example illustrates the probabilistic assignment of points to clusters that is central to the basic idea of Mixture Decomposition. The RGB value of the color of each pixel in Figure 4.1:b indicates the likelihood of each point belonging to each cluster. The more similar the color of a pixel to that of a segment in Figure 4.1:a, the higher the probability that the data point belongs to the specific cluster. For this specific experiment, it can be observed that when clusters are sufficiently far away from each other, the effect of probability functions onto points of other clusters is negligible. Only points that are near the area of intersection have a high probability in belonging to two clusters.

Experiment 2 – A hand drawn image of a floppy disk is shown in figure 4.2:. The original image, 4.2:a, essentially depicts two distorted rectangles. The result of PLSC, displayed in figure 4.2:b by overlaying resulting segments onto the original pixels, illustrates that the proposed method identifies all appropriate edges. Two of the detected segments, appearing in top right corners of both rectangles, are the result of the method's effort to classify rounded corners that are the effects of noise.

Experiment 3 – Figure 4.3:a: Three straight lines intersecting at the same point. Comparative results are shown in figures 4.3:a through 4.3:e. All considered algorithms, including



(a) Points sampled from Gaussian distributions and resulting segments of PLSC

(b) Visual representation of likelihoods of data points belonging to clusters after the termination of the algorithm

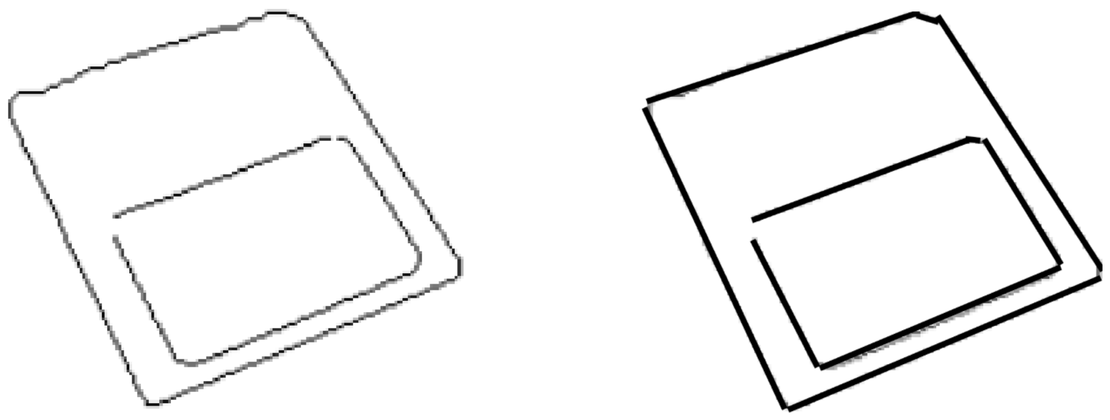
Figure 4.1: Experiment 1: A synthetic data set

PLSC, can detect those lines flawlessly.

Experiment 4 – Figure 4.4:a depicts eight straight lines, all sharing a common midpoint, forming thus an asterisk. This case is found to be more demanding than the one of the previous experiment, as points of distinct lines are now closer to each other. Especially near the intersection point, the same area is occupied by pixels from all segments. HT (4.4:b) and FCL (4.4:c) face difficulties in detecting all segments. The first one fails to identify half of them, while FCL detects even fewer and not correctly located segments. Both SPCLS and PLSC manage to identify all clusters correctly, as shown in figures 4.4:d and 4.4:e respectively.

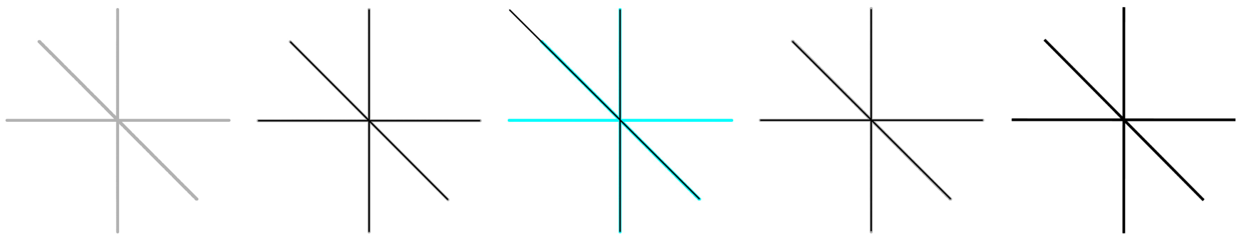
Experiment 5 – A configuration where distinct line segments need to be identified rather than lines that cut through the image. The original image is presented in figure 4.5:a. The modified Hough Transform detects accurately only a small fraction of the segments appearing in the original image (figure 4.5:b). The FCM (figure 4.5:c) fails totally in capturing even a single line segment, mainly due to the fact that it does not involve any notion of endpoints (figure 4.5:c). On the contrary, the approaches that use line segments as cluster representatives, SPCLS (Figure 4.5:d) and PLSC (Figure 4.5:e), are able to detect parts of the image with greater accuracy – albeit with a number of misidentified segments, in case of PLSC.

From now on, we consider more involved configurations where only line segments are present. Thus, based on the results obtained from the previous experiments, the performance of the Hough Transform and the FCL algorithms is expected to be poor. Therefore, they are not considered in the subsequent experiments.



(a) Edge pixels image (b) PLSC

Figure 4.2: Experiment 2: A drawing of a floppy disk



(a) Edge pixels image (b) HT (c) FCL (d) SPCLS (e) PLSC

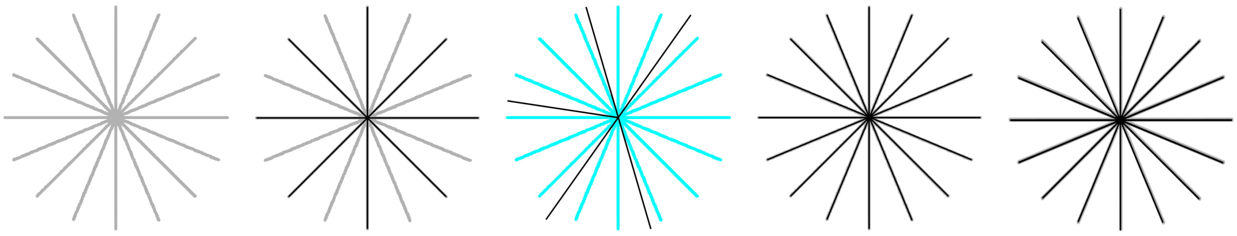
Figure 4.3: Experiment 3: Three intersecting lines

Experiment 6 – A drawing of books placed on a table. This illustration, depicted in figure 4.6:a, contains a large number of segments that can be considered as noise with regard to the ones conveying information about the content of the image. The results of PLSC (4.6:c) are adequate in capturing important edges of the image but a number of them is distorted due to those noisy clusters. In contrast, SPCLS, shown in 4.6:b seems less affected due to its possibilistic nature combined with the sparsity framework in which it is applied. Overall, the performances of the above algorithms in this experiment are comparable, since some segments that have been detected by one of them have not been detected by the other and vice versa. However, SPCLS gives a slightly smaller number of misplaced segments.

Experiment 7 – A pattern of geometric shapes is displayed in figure 4.7:a. A number of smaller segments appear that are positioned with their endpoints touching larger ones. This challenging layout is a failure case for PLSC, with the resulting segments depicted in 4.7:c. In this scenario, the initialization needed to be fine grained enough in order to generate initial representatives for all segments. This results to a vast amount of initial clusters. Even with extremely strict thresholds while merging, the algorithm ends up erroneously merging many of those together. The method of this work compares unfavorably to SPCLS shown in 4.7:b.

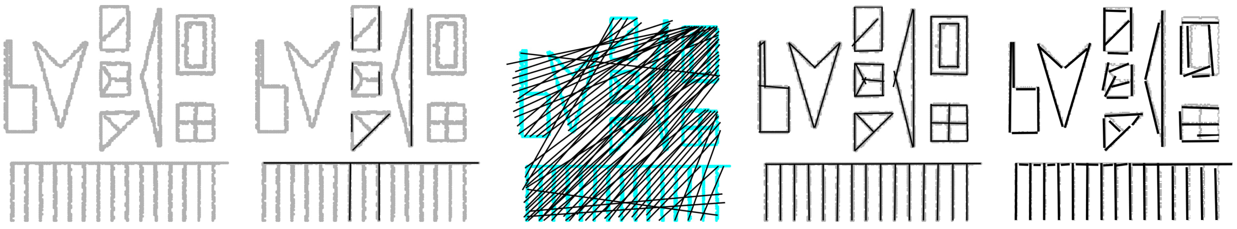
In the three last experiments, PLSC failed to converge. Instead, the execution stopped due to exceeding the limit of iterations. A number of misplaced segments continued to move during the final iterations, preventing the termination condition from being fulfilled. This behavior was exhibited in all three cases.

Experiment 8 – The image of figure 4.8:a is a solar observation image. Dark tubes of the image correspond to spicules (i.e. jets) in the atmosphere of the Sun. The intend is to locate and represent spicules of the image using line segments. Following the same process as in [22], a thresholding step is first applied so as to only keep pixels corresponding



(a) Original Image **(b) HT** **(c) FCL** **(d) SPCLS** **(e) PLSC**

Figure 4.4: Experiment 4: Eight intersected lines



(a) Original image **(b) HT** **(c) FCL** **(d) SPCLS** **(e) PLSC**

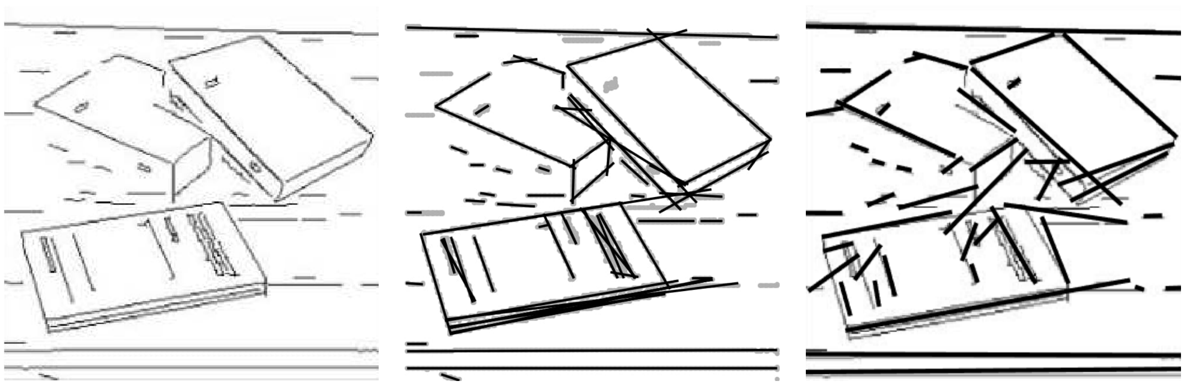
Figure 4.5: Experiment 5: A case of line segments

to spicules, resulting in figure 4.8:b. Then the clustering algorithm can be applied . PLSC is able to identify and assign segments to all distinct spots appeared in the image. The resulting segments are shown in figure 4.8:d with the ones resulted from SPCLS appearing in 4.8:c for reference purposes. Both methods identify appropriately all areas with the resulting segments positioned so as to indicate the general direction of the dark areas in the original image.

4.2 Discussion

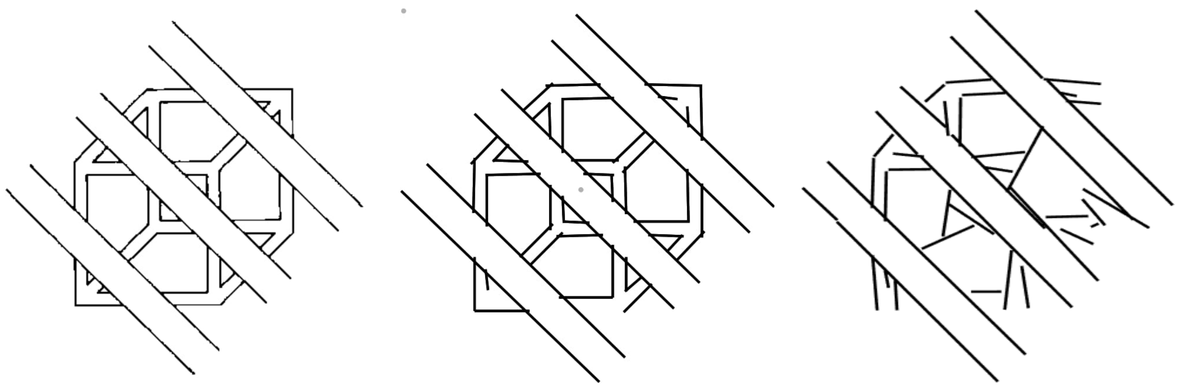
For further analysis of the performance of PLSC, table 4.1: provides details regarding input data sets, parameter values and execution results for each experiment.. Parameters are denoted with the names assigned to them during the discussion of the algorithm in section 3.6. Execution time concerns the $MATLAB^{®}$ implementation provided in ANNEX II and was measured on a machine using Intel 8th Generation CPU and 16GB of RAM. Apart from the parameters displayed on the table, PLSC requires two additional input variables from the user that affect the termination condition: ϵ, the tolerance and max_iters, the maximum number of iterations the algorithm is allowed to perform. In all experiments, those values were set to 10^{-3} and 100 respectively.

Using the information provided in table 4.1:, a few observations are in order. A great importance lies in the initial clusters. Their number needs to be considerably greater than the actual number of the physical clusters formed by the data. Because of that, the value of the initialization parameter, T needs to be configured properly. As shown in the table, its value does not directly correspond neither to the dimensions of the image nor the edge pixels contained there. Its value is rather affected by the size of the smallest clusters as well the number of intersections between image segments, as the initialization needs to locate initial cluster representatives in all ”*sub-segments*”. A disadvantage with small T values is that the initialization will produce multiple segments even in clusters that do not intersect with any others. Under appropriate parameter values, those segments will be eventually merged together but this process results to many iterations and increased execution time.



(a) Original image **(b) SPCLS** **(c) PLSC**

Figure 4.6: Experiment 6: A drawing of books



(a) Original image **(b) SPCLS** **(c) PLSC**

Figure 4.7: Experiment 7: A geometrical pattern

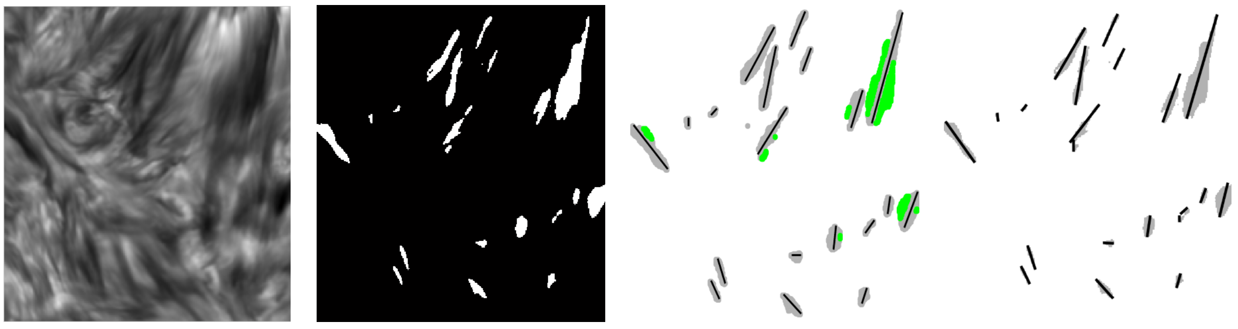
Table 4.1: Execution Details of PLSC per experiment

	Input Image			Parameters				Results	
Exp. Number	**Image Dimensions**	**Data Points**	**Actual Segments**	T	ϵ_{gap}	ϵ_{ang}	σ	**Clusters (Initial / Final)**	**Iterations / Execution Time (mins)**
1	209×264	2118	5	50	10	0.80	2	17/5	13 / 0.45
2	646×499	10199	8	60	3	0.95	3	53/10	21 / 8.25
2	553×553	1494	3	80	40	0.99	3	22/3	8 / 0.34
4	553×553	3913	8	80	40	0.99	3	58/8	17 / 2.43
5	225×225	5381	56	15	1	0.99	2	254/62	100^* / 29.63
6	227×226	3966	$45 - 50$	15	3	0.99	1	251/39	100^* / 28.85
7	383×386	6249	52	28	1	0.99	1	141/39	100^* / 34.17
8	477×434	9889	$18 - 22$	20	10	0.98	1	91/21	78 / 16.05

* The algorithm stopped due to limit in number of iterations without converging.

For most of the parameters involved in PLSC, finding proper values is rather straightforward. Firstly, ϵ needs little modification. The value of 0.1, used in the presented experiments and during the development process, is found to be a good fit. Similarly, max_iters does not require any fine tuning - a relative high value will not normally prevent the iterative process from converging. In that respect, while convergence analysis is not included in this work, *Experiments 5 - 7* indicate that PLSC is unable to converge in cases where actual segments are not correctly identified.

Additionally, values for parameters σ, the ”variance” around the line segment representa-



(a) Original image **(b) After thresholding (shown in negative)** **(c) SPCLS** **(d) PLSC**

Figure 4.8: Experiment 8: A solar observation containing spicules

tives and ϵ_{ang}, the tolerance of collinearity, can be found effortlessly. Namely, σ requires values that are just enough so as to prevent floating point underflow from happening. Since underflow may happen when computing values of the pdf for points that are far away from the respective segments, an appropriate value for σ is mainly affected by the size of the image. In the experiments, that value was initially set to 1 or 2, and in case underflow was detected, that value was incremented, normally by 1. High values of σ will result in high likelihood values for far away points, deteriorating the results. Using the lowest value that does not result in numerical errors is recommended. In a similar way, ϵ_{ang} may start wih a value as high as 0.99 and in case merges fail to happen, that threshold can be relaxed. Those cases typically correspond to images in which elongated clusters appear rather than segments (*Experiments 1,8*), or segments do appear but their pixels do not strictly form straight lines but are rather noisy (*Experiment 2*).

The final parameter is ϵ_{gap}, , the distance threshold between different clusters. It has a principal effect on the overall performance of the algorithm. Its value should be large enough so as to allow for segments to be merged correctly but also small enough in order to prevent merging segments corresponding to different clusters. Since this is a threshold of distance between segments, its values are measured in pixel units. That means fitting values may be identified by observing the input image directly. Values that are smaller than the smallest pixel gap between two different line segments, may constitute appropriate choices. Difficulties in determining optimal values may be found in cases that combine intersecting clusters with clusters that are parallel to each other and in close proximity (e.g. *Experiments 5, 6, 7*) . Exhibiting the same limitation with T, using a single threshold for the entirety of the image may be too restrictive. A few modifications were explored that accounted for other variables such as cluster thickness during experimentation on this work, but none was able to provide consistently better results; thus they are not included in the dissertation.

5. CONCLUSION

5.1 Summary

In this dissertation, a clustering algorithm, called *Probabilistic Line Segment Clustering (PLSC)* is proposed, which is suitable for identifying linear clusters formed in a data set. It has been applied in the identification of line segments formed by edge pixels in digital images. In the present set up, each cluster is represented by a pdf especially tailored for representing distribution of data points around a line segment and the entire data set is represented by a weighted sum of these pdfs. The challenge here is to decompose the sum of all pdfs, so as to identify the individual pdfs corresponding to the line segments formed by the data. PLSC falls under the probabilistic framework. It is based on the paradigm of Mixture Decomposition of the Expectation Maximization-like algorithm. However, PLSC deviates from classical EM philosophy in the maximization step where a regression-like fitting procedure as well as a unification policy are applied instead of a strict direct maximization. However, this feature gives the flexibility to bypass the requirement of the classical EM for knowing the exact number of clusters (pdfs). In order for the appropriate number of segments to be determined, the algorithm starts with more clusters than those actually formed by the data points and (hopefully) ends up with a set of line segments, fitted to the clusters underlying in the data set.

5.2 Evaluation

Experimental results of the proposed PLSC algorithm were presented and discussed. The data sets used were originated from a relevant publication [22]. A comparison of the algorithm of this work with the ones examined in that article was conducted. The results have shown that the algorithm compares equally or favorably to a couple of benchmark algorithms from the literature. When comparing to a recent, state of the art approach, PLSC is able to produce comparable results most of the time, even though it is outperformed in a few cases.

To the best of the author’s knowledge, this work constitutes the first proposed probabilistic clustering algorithm that can be used for detecting line segments in any type of digital image. Previously related works have either explored approaches of alternative taxonomies or focused on special cases of imagery.

5.3 Future Work

As it may have been clear by the description of the proposed algorithm, there is significant room for improvements. Future work may concern a number of issues. At first, an implementation allowing for speeding up the execution time of the algorithm can be provided. Such implementation may use sophisticated data structures and policies in order to reduce the computations needed at each iteration as well as the number of iterations itself. The latter may be attributed e.g. to utilization of the fact that an update in one area of the image does not affect significantly the rest areas of it. Thus, not all clusters may need to be updated at each iteration. Furthermore, using this spatial locality, along with the identification of tasks that can be executed without any need of information exchange, the algorithm may be implemented in a form suitable for parallel execution.

Local sensitive hyper-parameters may also be explored since they can encode information that varies throughout different parts of the image. Literature of traditional edge detection techniques may provide a starting point for approaches for finding local values for the involved hyper-parameters. A more sophisticated initialization that will be able to estimate areas of intersections and produce multiple clusters only in those areas would also improve computational performance. Additionally, imposing the notion of sparsity, similarly to [22], may be a suitable improvement.

Finally, theoretical analysis may also contribute in establishing more rigorously the behavior of the algorithm. Relevant results on convergence or modifications of the procedure that can be proven to maximize the expected likelihood function could be of use as they could help in fully incorporating the algorithm in the Expectation - Maximization framework.

TABLE OF TERMINOLOGY

Ξενόγλωσσος Όρος	**Ελληνικός Όρος**
Cluster	Συστάδα / Ομάδα
Clustering	Συσταδοποίηση / Ομαδοποίηση
Elongated Clusters	Επιμήκεις Συστάδες
Expectation - Maximization	Αναμενόμενη Τιμή - Μεγιστοποίηση
Gaussian Probability Distribution	Γκαουσσιανή Κατανομή Πιθανότητας
Geospatial Data Analysis	Ανάλυση Γεωχωρικών Δεδομένων
Heuristic Rule	Ευρετικός Κανόνας
Image Compression	Συμπίεση Εικόνας
Image Processing	Επεξεργασία Εικόνας
Line Detection	Εντοπισμός Γραμμικών Στοιχείων
Mixture Decomposition	Αποδόμηση Μίξης
Pattern Recognition	Αναγνώριση Προτύπων
Probability Theory	Θεωρία Πιθανοτήτων
Probabilistic clustering	Πιθανοτική Ομαδοποίηση / Συσταδοποίηση
Probabilistic Line Segment Clustering	Πιθανοτική Ομαδοποίηση Ευθυγράμμων Τμημάτων
Road Extraction	Εξαγωγή Δρόμων

ABBREVIATIONS - ACRONYMS

E	Expectation
EM	Expectation - Maximization
FCL	Fuzzy c-Lines
GMDAS	General Mixture Decomposition Algorithmic Scheme
GMM	Gaussian Mixture Model
HT	Hough Transform
LSE	Least Square Error
M	Maximization
MLE	Maximum Likelihood Estimation
pdf	Probability Density Function
PLSC	Probabilistic Line Segment Clustering
RGB	Red - Green - Blue
SPCLS	Sparse Possibilistic Clustering of Line Segments

ANNEX I. STUDY OF THE PROPOSED PROBABILITY DISTRIBUTION FUNCTION

In order to prove that Equation (3.9) is, in fact, a probability distribution function, given a line segment L with endpoints $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^2$, some $\sigma > 0$ and the definition of $D(\boldsymbol{x}, L)$ from Equation (3.8), the following properties must hold:

$$p(\boldsymbol{x}; L, \sigma) \geq 0 \ , \ \forall \boldsymbol{x} \in \mathbb{R}^2 \tag{1}$$

and

$$\int_{\mathbb{R}^2} p(\boldsymbol{x}, L, \sigma) d\boldsymbol{x} = 1 \tag{2}$$

The first property is relatively straightforward to show. Given that:

$$\sigma > 0 \tag{3}$$

$$exp(x) > 0 \ , \ \forall x \in \mathbb{R} \tag{4}$$

$$l_L = \|\boldsymbol{a} - \boldsymbol{b}\| \geq 0 \tag{5}$$

$$D(\boldsymbol{x}, L) : \mathbb{R}^6 \to \mathbb{R} \tag{6}$$

we have:

$$2\pi\sigma^2 + \sqrt{2\pi} l_L \sigma > 0 \Rightarrow \frac{1}{2\pi\sigma^2 + \sqrt{2\pi} l_L \sigma} > 0 \tag{7}$$

and

$$exp\Big(- \frac{1}{2\sigma^2} D^2(\boldsymbol{x}, L) \Big) > 0 \tag{8}$$

Combining (7) and (8) we reach the conclusion that:

$$p(\boldsymbol{x}; L, \sigma) = \frac{1}{2\pi\sigma^2 + \sqrt{2\pi} l_L \sigma} exp\Big(- \frac{1}{2\sigma^2} D^2(\boldsymbol{x}, L) \Big) > 0 \tag{9}$$

For the integral property, let us first compute the circumference, A_r, of the curve that is formed by all points that have a constant (squared) distance of r^2 from L, for some $r \geq 0$. That integral can be computed geometrically from figure I.1:

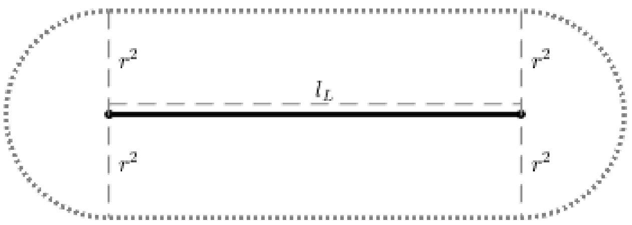


Figure I.1 The circumference of all points that have a distance of r^2 from L

$$A_r \equiv \int_{D(\boldsymbol{x}, L) = r^2} dA_r = 2\pi r + 2 l_L \tag{10}$$

Now, let us define:

$$f(\boldsymbol{x}) = exp\Big(- \frac{D(\boldsymbol{x}, L)}{2\sigma^2} \Big) \tag{11}$$

and subsequently:

$$erf(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} dt \tag{12}$$

with $\lim\limits_{z \to +\infty} erf(z) = 1$.

In order to prove that property (2) holds, it will suffice to show that:

$$I \equiv \int_{\mathbb{R}^2} f(\boldsymbol{x}) d\boldsymbol{x} = 2\pi\sigma^2 + \sqrt{2\pi} l_L \sigma \tag{13}$$

The proof for (13) starts as follows:

$$\begin{aligned} I &\equiv \int_{\mathbb{R}^2} f(\boldsymbol{x}) d\boldsymbol{x} = \int_{\mathbb{R}^2} exp\Big(- \frac{D(\boldsymbol{x}, L)}{2\sigma^2} \Big) d\boldsymbol{x} = \int_0^{+\infty} \Big[\int_{D(\boldsymbol{x},L)=r^2} exp\Big(- \frac{r^2}{2\sigma^2} \Big) dA_r \Big] dr = \\ &= \int_0^{+\infty} exp\Big(- \frac{r^2}{2\sigma^2} \Big) \Big[\int_{D(\boldsymbol{x},L)=r^2} dA_r \Big] dr \overset{(10)}{=} \int_0^{+\infty} exp\Big(- \frac{r^2}{2\sigma^2} \Big) [2\pi r + 2 l_L] dr = \\ &= \underbrace{\int_0^{+\infty} 2\pi r \; exp\Big(- \frac{r^2}{2\sigma^2} \Big) dr}_{\boldsymbol{A}} + 2 l_L \underbrace{\int_0^{+\infty} exp\Big(- \frac{r^2}{2\sigma^2} \Big) dr}_{\boldsymbol{B}} \end{aligned} \tag{14}$$

Dealing with A and B separately we have:

$$\begin{aligned} A &= \int_0^{+\infty} 2\pi r \; exp\Big(- \frac{r^2}{2\sigma^2} \Big) dr = 2\pi \int_0^{+\infty} r \; exp\Big(- \frac{r^2}{2\sigma^2} \Big) dr \\ &= \pi \int_0^{+\infty} exp\Big(- \frac{r^2}{2\sigma^2} \Big) dr^2 \overset{z=r^2}{=} \pi \int_0^{+\infty} exp\Big(- \frac{z}{2\sigma^2} \Big) dz \\ &= 2\pi\sigma^2 \int_0^{+\infty} exp\Big(- \frac{z}{2\sigma^2} \Big) d\Big(\frac{z}{2\sigma^2} \Big) \overset{y=\frac{z}{2\sigma^2}}{=} 2\pi\sigma^2 \int_0^{+\infty} exp(-y) dy \\ &= 2\pi\sigma^2 \Big[- exp(-y) \Big]_0^{+\infty} = 2\pi\sigma^2 (0+1) = 2\pi\sigma^2 \end{aligned} \tag{15}$$

$$\begin{aligned} B &= \sigma \int_0^{+\infty} exp\Big(- \frac{1}{2} \Big(\frac{r}{\sigma} \Big)^2 \Big) d\Big(\frac{r}{\sigma} \Big) \overset{y=\frac{r}{\sigma}}{=} \sigma \int_0^{+\infty} exp\Big(- \frac{y^2}{2} \Big) dy \\ &= \sqrt{2}\sigma \int_0^{+\infty} exp\Big(- \Big(\frac{y}{\sqrt{2}} \Big)^2 \Big) d\Big(\frac{y}{\sqrt{2}} \Big) \overset{t=\frac{y}{\sqrt{2}}}{=} \sqrt{2}\sigma \int_0^{+\infty} exp(-t^2) dt \\ &= \sqrt{2}\sigma \frac{\sqrt{\pi}}{2} \frac{2}{\sqrt{\pi}} \int_0^{+\infty} exp(-t^2) dt \overset{(12)}{=} \sqrt{2}\sigma \frac{\sqrt{\pi}}{2} \lim_{z \to +\infty} erf(z) = \sigma \frac{\sqrt{\pi}}{\sqrt{2}} \end{aligned} \tag{16}$$

Substituting values of A and B from (15) and (16) into (14) we are left with:

$$I = 2\pi\sigma^2 + 2 l_L \sigma \frac{\sqrt{\pi}}{\sqrt{2}} = 2\pi\sigma^2 + \sqrt{2\pi} l_L \sigma \tag{17}$$

which concludes the proof.

ANNEX II. MATLAB® CODE

The source code of a MATLAB® implementation of PLSC is presented next. The source is split over a number of files. The only function that needs to be run by the user is `PLSC()`, found in `PLSC.m`. All `.m` files need to be in MATLAB's working directory or in the path. A minimal example of how to run `PLSC()` can be found in `run_PLSC.m` script. You may type `help PLSC` for details on input and output variables.

- File `run_PLSC.m`

```matlab
% Minimal example of how to use PLSC. Substitute variables appropriately in
% order to run PLSC on a desired image with the corresponding parameters


% Substitute here accordingly
image_name = "some_image.png"; % Desired image name (or path if needed)
T          = 20;               % Initialization parameter of PLSC
e          = 0.1;              % Coveregence tolerance. 0.1 is usually good
e_gap      = 5;                % Distance threshold for merging segments
e_ang      = 0.99;             % Angular similarity threshold for merging
sigma      = 2;                % Variance of distribution.
max_iters  = 100;              % Threshold of iterations in case PLSC does
                               % not converge
verbose    = true;             % Flag indicating whether PLSC should print
                               % messages per iteration as well as plot
                               % current segments. Set to false if only the
                               % result is needed.

% read image file from disk
Im = imread(image_name);

% keep only greyscale information
Im = Im(:,:,1);

% Assuming image has a white background, convert it so as to be used from
% PLSC. Skip next line if image has a black backgounrd.
Im = 1-Im;

% Vector that will contain concatenated endpoints of identified segments
theta = [];

% Structure with additional information
INFO = [];

% Run PLSC here
[theta, INFO] = PLSC(Im, T, e, e_gap, e_ang, sigma, max_iters, verbose);

% use internal plotting function to display segments. You may plot those
% segments any way you like.
plot_state(INFO.clusters, 101, [], [], [], "Final Result");
```

- File `PLSC.m`

```
function [theta, INFO] = PLSC(Im, T, e, e_gap, e_ang, s, max_iters,
    verbose)
    % PLSC - Perform probabilistic line segment clustering on an image to
    % locate its segments.
    %
    % [theta, INFO] = PLSC(Im, T, e, e_gap, e_ang, s, max_iters, verbose)
    % applies PLSC algorithm on black and white image of edge points, Im.
    % Im should be a two dimensional array representing an edge
    % image. Elements with positive value are considered edge pixels and
    % elements with 0 are considered background.
    % PLSC uses appropriate parameters T, e (epsilon), e_gap (epsilon_gap
    ),
    % e_ang( epsilon_ang), s (sigma) and max_iters. Optional variable
    % verbose may have a boolean value, true indicating that information
    % will be printed on console and figures will be shown per iteration.
    % It is set to false by default.
    % The function returns the endpoints of  the final segments with
    their
    % endpoints concatenated to theta vector. Structure INFO has the
    % following fields:
    %   INFO.clusters     : An array of resulting Cluster objects
    %   INFO.posteriors   : An N x m array, namely P(c_j | x_i )
    %   INFO.num_clusters : The number of segments finally identified

    if nargin < 7
        error("Wrong usage. Type help PLSC for instructions");
    elseif nargin < 8
        verbose = false;
    end

    rho = sqrt(2);

    [I,J]=find(Im>0);
    X=[I,J];

    Im = (1-Im);
    Im = Im';


    if verbose
        [rows, cols] = size(Im);
        fprintf("Initializing image with dimensions %dx%d, containing %d
    datapoints\n", rows, cols, length(X));
    end

    segments = SPCLS_initialization(Im, T, rho);

    non_zero_segments = [];
    for i=1:length(segments)
        if segments(i,:) ~= zeros(1,4)
            non_zero_segments = [non_zero_segments; segments(i,:)];
        end
    end
    segments = non_zero_segments;
    m = length(segments);

    if verbose
        fprintf("Initialization produced %d clusters\n", m);
    end

    PARAMS.num_iterations           = max_iters;
    PARAMS.initial_sigma            = s;
    PARAMS.stopping_criterion       = @(a,b,c)
   rate_of_clusters_movement_criterion(a,b,c);
    PARAMS.rate_of_movement_tol     = e;
    PARAMS.compute_Q                = true;
    PARAMS.verbose                  = verbose;
    PARAMS.unification_ang_tol      = e_ang;
    PARAMS.unification_gap_tol      = e_gap;
    PARAMS.iters_wo_merge           = 3;
    PARAMS.T                        = T;

    tic;
    [classification, clusters, num_clusters, ~] = GMDAS(X, m, segments,
   PARAMS);
    if verbose
        toc;
    end

    INFO.clusters = clusters;
    INFO.posteriors = classification;
    INFO.num_clusters = num_clusters;

    theta = serialize_to_theta_vector(clusters);
end
```

- File `centroid.m`

```
function [c] = centroid(X, weighted)
% c = centroid(X)
% Get the centroid vector of some vectors.
% INPUT:
%   X   : The row-vectors for computing their cetnroid.
%         An N*l array of numbers where N is the number of vectors and l
   is
%         the number of dimensions.
% OUTPUT :
%   c     : A ROW vector (of dimension l) which is the centroid of vectors
%           in X.

    [N, l] = size(X);
    c = zeros(l,1);

    if nargin == 1
        for i=1:l
            c(i) = sum(X(:,i)) / N;
        end
    elseif nargin == 2 && weighted == true
        for i=1:N
            c = c + X(i,:)/norm(c-X(i,:));
            c = c / N;
        end
    end

    c = c';
end
```

- File `Cluster.m`

```
classdef Cluster < handle
    % A class modeling the cluster representation of the algorithm while
    % also containing the functionality for updating its parameters and
    % estimating pdf.

   properties
       id
       lineSegment
       aPriori
       sigma
       color
   end

   methods

       function obj = Cluster(id, x1, x2, sigma, aPriori)
           % Create a new Cluster object

           if nargin == 0
               obj.id = 0;
               obj.sigma = 0;
               obj.aPriori = 0;
               return
           end


           obj.id            = id;
           obj.sigma         = sigma;
           obj.aPriori       = aPriori;
           obj.lineSegment   = LineSegment(x1, x2);
           obj.color         = rand(1,3);
       end


       function update_a_priori_probability(obj, N, cluster_posteriors)
           % Update its a priori probability computed as 1/N * Sum{P(Cj|x
    )}
          obj.aPriori = sum(cluster_posteriors) / N;
       end

       function update_line_segment(obj, Y)
           % Update line segment information. Currently using the fast
           % method of random centroids

           [Ni,~] = size(Y);
           if Ni == 0
               obj.aPriori = 0;
               obj.lineSegment = LineSegment([0,0],[0,0]);
               return
           end

           carrier = principal_axis(Y);
           [a, b] = line_segment_endpoints(Y, carrier);
```

```
            obj.lineSegment = LineSegment(a,b);

            if a ~= b
                plot_state([obj], -1, Y, [], obj.id);
                a = a;
            end
        end


        function bool = is_eliminated(obj)
            bool = any(isnan(obj.lineSegment.A));
            bool = bool | any(isnan(obj.lineSegment.B));
            bool = bool | (obj.aPriori == 0);
            bool = bool | (obj.sigma == 0);
        end
    end
end
```

- File `compute_posterior_probabilities.m`

```
function P_Cj_xi = compute_posterior_probabilities(X, m, clusters)
    [N,l] = size(X);
    P_Cj_xi = -1*ones(N,m);
    nominators = zeros(1,m);
    for i=1:N
        for j=1:m
            x = X(i,:);
            c = clusters(j);
            if ~c.lineSegment.isNull()
                P_x_C = pdf(x, c.sigma, c.lineSegment);
            else
                P_x_C = 0;
            end
            nominators(j) = P_x_C * c.aPriori;
        end

        denominator = sum(nominators);
        for j=1:m
            aPosteriori = nominators(j)/denominator;
            if (isnan(aPosteriori))
                aPosteriori = 0;
            end

            P_Cj_xi(i,j) = aPosteriori;
        end
    end
end
```

- File `cosine_similarity.m`

```
function [d] = cosine_similarity(L1,L2)

    u = L1.A - L1.B;
    v = L2.A - L2.B;
    if all(v==0) || all(u==0), d = 1; end % zero vectors are treated as Null segments. Explicitly allowing them to be merged.
    if length(v) ~= length(u), error("Cosine similarity cannot be computed for vectors of different dimensions"); end

    d = dot(u,v) / (norm(u)*norm(v));
end
```

- File GMDAS.m

```
function [classification, clusters, m_, frames] = GMDAS(X,m, segments, PARAMS)
    [N,l] = size(X);

    %
    % Clusters Initializations
    %
    aPriori = 1/m;
    clusters = Cluster.empty(0,m);
    for j=1:m
        x1 = segments(j, 1:l);
        x2 = segments(j, l+1:2*l);
        clusters(j) = Cluster(j, x1, x2, PARAMS.initial_sigma, aPriori);
    end


    %
    % Auxiliary variables initialization
    %
    converged                  = false;
    frames                     = [];
    t                          = 0;
    Q_val                      = 0;
    confidence                 = 0;
    movement_rate              = 0;
    str                        = "";
    best_Q                     = -Inf;
    best_clusters              = Cluster.empty(0,m);
    best_posteriors            = zeros(N,m);
    best_t                     = 0;

    if PARAMS.verbose, frames = plot_state(clusters, 0, X, [], frames, "initial clusters"); end

    %
    % Iterative procedure
    %
    while ~converged && t <= PARAMS.num_iterations

        %
        % E - Step
        %
        posteriors = compute_posterior_probabilities(X,m,clusters);
        best_cluster_per_point = hard_clustering(posteriors);
        if PARAMS.compute_Q, Q_val = Q_function(X,clusters,posteriors); end


        %
        % Save best Q
        %
        if (Q_val >= best_Q)
            best_clusters = copy_clusters(clusters);
            best_Q = Q_val;
            best_posteriors=posteriors;
            best_t=t;
        end


        if PARAMS.verbose
            str =sprintf("%d: Q: %.0f | Theta rate: %.2f | clusters: %d %s",...
                t, Q_val, movement_rate, length(clusters), str);
            frames = plot_state(clusters, t, X, best_cluster_per_point, frames, str);
            fprintf("%s\n",str);
            str = "";
        end


        %
        % Apply FIT and MERGE procedures
        %
        clusters = maximization(clusters, X, posteriors);
        if m == length(clusters)
            [new_clusters, ~ , str] = merge_clusters(clusters, X, posteriors, str, PARAMS);
            if length(new_clusters) ~= m
                clusters = copy_clusters(new_clusters);
            end
        end
        m = length(clusters);

        t = t+1;
        [converged, movement_rate] = PARAMS.stopping_criterion(t, clusters, PARAMS);
    end




    classification = compute_posterior_probabilities(X,length(clusters), clusters);
    if PARAMS.verbose
        frames = plot_state(clusters, t, [], [], frames, "Final result");
    end
    m_ = m;

end
```

- File `hard_clustering.m`

```
function [best_cluster_per_point] = hard_clustering(posteriorProbabilities)
    [N,m] = size(posteriorProbabilities);
    best_cluster_per_point = -1 * zeros(N,1);
    for i = 1:N
        [max_p, argmax] = max(posteriorProbabilities(i,:));
        if max_p > 0
            best_cluster_per_point(i) = argmax;
        else
            best_cluster_per_point(i) = 0;
        end
    end
end
```

- File `maximization.m`

```
function [new_clusters] = maximization(clusters, X, posteriors)
    m = length(clusters);
    new_aPrioris = zeros(1,m);
    [N,l] = size(X);
    best_cluster_per_point = hard_clustering(posteriors);
    for j=1:m
        c = clusters(j);
        new_aPrioris(j) = sum(posteriors(:,j))/N;
    end

    new_clusters = [];
    for j=1:m
        c = clusters(j);
        cluster_points = X(best_cluster_per_point==j,:);
        if length(cluster_points)<2, continue, end
        c.lineSegment = principal_axis(cluster_points, X, posteriors(:,j)
   );
        c.lineSegment.fit_endpoints(cluster_points);
        c.aPriori = new_aPrioris(j);
        new_clusters = [new_clusters c];
    end


end
```

- File `merge_clusters.m`

```
function [new_clusters, unified_ids, str_] = merge_clusters(clusters, X,
   posteriors, str, PARAMS)
    ANGULAR_TOL               = PARAMS.unification_ang_tol;
    GAP_TOL                   = PARAMS.unification_gap_tol;
    m                         = length(clusters);
    unified_ids               = [];
    deleted_clusters_indexes = [];
    new_clusters              = [];

    for i=1:m
        if any(deleted_clusters_indexes==i), continue, end
        min_dist = Inf;
        best_j = 0;

        for j=1:m
            if i==j, continue, end
            if any(deleted_clusters_indexes==j), continue, end
            sim =abs(cosine_similarity(clusters(i).lineSegment, clusters(j).lineSegment));
            if sim >= ANGULAR_TOL && are_closeby(clusters(i).lineSegment, clusters(j).lineSegment, GAP_TOL)

                dist = min_line_segment_distance(clusters(i).lineSegment, clusters(j).lineSegment);
                if dist < min_dist
                    min_dist = dist;
                    best_j = j;
                end
            end
        end


         if min_dist < Inf
            if PARAMS.verbose
                fprintf("Unifying %d and %d\n", clusters(i).id, clusters(best_j).id);
            end
            best_cluster_per_point = hard_clustering(posteriors);
            C1_points = X(best_cluster_per_point == i,:);
            C2_points = X(best_cluster_per_point == best_j,:);

            if length(C1_points) > length(C2_points), C1 = clusters(i);
            else,                                     C1 = clusters(best_j);
            end

            C_new_points = [C1_points; C2_points];
            L = C1.lineSegment;
            L.fit_endpoints(C_new_points);
            aPriori =  clusters(i).aPriori+clusters(best_j).aPriori;
            C_new = Cluster(C1.id, L.A, L.B, C1.sigma, aPriori);
            C_new.color = C1.color;

            new_clusters = [new_clusters C_new];
            deleted_clusters_indexes = [deleted_clusters_indexes i best_j];
        end
    end

    unified_ids = arrayfun(@(c)(c.id), clusters(deleted_clusters_indexes));

    old_clusters = copy_clusters(clusters);
    old_clusters(deleted_clusters_indexes) = [];
    new_clusters = [old_clusters new_clusters];

    if isempty(unified_ids)
        str_ = sprintf("%s | No unifications", str);
    else
        str_ = sprintf("%s | %d unifications", str, length(unified_ids)/2);
        reshape(unified_ids, 2, length(unified_ids)/2); unified_ids = unified_ids';
    end
end

```

```

function d = min_line_segment_distance(L1, L2)
    d1 = point_segment_distance(L1.A, L2);
    d2 = point_segment_distance(L1.B, L2);
    d3 = point_segment_distance(L2.A, L1);
    d4 = point_segment_distance(L2.B, L1);

    d = min([d1,d2,d3,d4]);
end


function bool = are_closeby(L1, L2, TOL2)
    bool = min_line_segment_distance(L1, L2) <= TOL2;
end
```

- File `pdf.m`

```
function [p] = pdf(x, s, L)
    if L.isNull(), p=0; return; end
    D_x_L = point_segment_distance(x, L);
    D_x_L = D_x_L ^2;
    p = (1 / (2 * pi *s^2 + sqrt(2*pi)*L.length()*s))*exp(-D_x_L/(2*s^2))
    ;
end
```

- File `plot_state.m`

```
function  F = plot_state(clusters, t, X, classification, F_old, info,
    SINGLE)
    m = length(clusters);


    if nargin == 7
        figure('Name',num2str(t));
        C = clusters(1);
        centroid = (C.lineSegment.A + C.lineSegment.B) ./ 2;
        darker_color = 0.8 * C.color;
        plot([C.lineSegment.A(1) C.lineSegment.B(1)],[C.lineSegment.A(2)
    C.lineSegment.B(2)], 'color', darker_color,  'LineWidth', 3);
        text(centroid(1),centroid(2),num2str(C.id));
        hold on
        scatter(X(:,1), X(:,2), 'MarkerEdgeColor', C.color);
        hold off
    else
        figure('Name',num2str(t));
        grid on
        if ~isempty(X)
            for i=1:m
                C = clusters(i);
                Y = X(classification==i, :);
                scatter(Y(:,1), Y(:,2), 'MarkerEdgeColor', C.color);
                hold on;
            end

            Y = X(classification==0, :);
            if isempty(classification), Y = X; end
            scatter(Y(:,1), Y(:,2), 'MarkerFaceColor', [0,0,0]);
            hold on;


        end
        for i=1:m
            C = clusters(i);
            centroid = (C.lineSegment.A + C.lineSegment.B) ./ 2;
            hold on
            darker_color = 0.8 * C.color;
            plot([C.lineSegment.A(1) C.lineSegment.B(1)],[C.lineSegment.A
    (2) C.lineSegment.B(2)], 'color', darker_color,  'LineWidth', 3);
            text(centroid(1),centroid(2),num2str(C.id));
        end

        dim = [0 1 0 0];
        annotation('textbox',dim,'String',info,'FitBoxToText','on');

        hold off
    end
    F = [F_old getframe(gcf)];
end
```

- File `point_segment_distance.m`

```
function [d] = point_segment_distance(p, L)
    % [d] = point_segment_distance(point, lineSegmentStart,
    lineSegmentEnd)
    % Calculate distance from a point to line segment as described in
    % [Koutr 18]
    % INPUT
    %  p                : An N-dimension vector
    %  L                : A LineSegment object
    % OUTPUT
    %  d                : The distance from `point` to the line segment

    if L.isNull()
        d = Inf;
    else
        p_proj = L.project(p);
        if L.belongs(p_proj)
            d = norm(p-p_proj);
        else
            d = min( norm(p-L.A), norm(p-L.B) );
        end
    end
end
```

- File `principal_axis.m`

```
function [axis_] = principal_axis(S, X, cluster_posteriors)
    % [axis_] = principal_axis(S)
    % Compute princial axis of a set of points using  Algorithm B

    [N,l] = size(S);
    if N==0 || N == 1 % no points for this line segment
        cf = zeros(1,1);
        cs = zeros(1,1);
        axis_ = LineSegment(cf, cs);
        return;
    end

    stds_by_col = std(S);                           % compute std in each dimension
    [~, largest_dev_col] = max(stds_by_col);        % find dimension of largest std
    S = sortrows(S, largest_dev_col);               % sort corpus in accordance to that dimension

    Nf = ceil(N/2);                                 % number of points in the first subgroup
    Ns = N - Nf;                                    % number of points in the second subgroup

    cf = centroid(S(1:Nf, :));                      % centroid of first subgroup
    cs = centroid(S(Nf+1:N, :));                    % centroid of second subgroup

    axis_ = LineSegment(cf, cs);

    weighted_cf = [0 0];
    weighted_cs = [0 0];

    Nf = 0;
    Ns = 0;

    for i=1:length(X)
        x = X(i,:);

        if norm(x-cf) < norm(x-cs)
            times = cluster_posteriors(i)*1000;
            for t=1:times
                weighted_cf = weighted_cf + x;
            end
            Nf = Nf+times;
        else
            times = cluster_posteriors(i)*1000;
            for t=1:times
                weighted_cs = weighted_cs + x;
            end
            Ns = Ns+times;
        end
    end

    weighted_cf = weighted_cf / Nf;
    weighted_cs = weighted_cs / Ns;

    axis_ = LineSegment(weighted_cf, weighted_cs);
    axis_.fit_endpoints(S);

end
```

- File `Q_function.m`

```
function q_value = Q_function(X, clusters, posteriors)
    Q_t_plus_1 = 0;
  N = length(X);
  m = length(clusters);
    for i=1:N
    for j=1:m
      x = X(i,:);
      s = clusters(j).sigma;
      P = clusters(j).aPriori;
            L = clusters(j).lineSegment;
      P_j_given_x = posteriors(i,j);
            prev = Q_t_plus_1;
      Q_t_plus_1 = Q_t_plus_1 + ( P_j_given_x * log(pdf(x,s,L)*P) );
            if isnan(Q_t_plus_1)
                Q_t_plus_1 = prev;
            end
    end
    end
    q_value = Q_t_plus_1;
end
```

- File `rate_of_clusters_movement_criterion.m`

```
function [stop, movement_rate] = rate_of_clusters_movement_criterion(t,
    clusters, PARAMS)
    TOL = PARAMS.rate_of_movement_tol;
    persistent previous_theta;
    persistent previous_variance;

    theta = serialize_to_theta_vector(clusters);

    if t==1
        previous_theta = theta;
        previous_variance = 1;
        stop = false;
        movement_rate = nan;
        return;
    else
        if length(theta) ~= length(previous_theta)
            stop = false;
            movement_rate = nan;
        else
            variance = norm(previous_theta-theta, 1);
            movement_rate = abs(variance-previous_variance);

            if movement_rate <= TOL
                stop = true;
            else
                stop = false;
            end
            previous_variance = variance;
        end
        previous_theta = theta;
        return;
    end
```

```

end
```

- File `serialize_to_theta_vector.m`

```
function theta = serialize_to_theta_vector(clusters)
    m = length(clusters);
    theta = [];
    k=1;
    for i=1:m
        c = clusters(i);
        if c.is_eliminated(), continue, end
        theta(k) = c.lineSegment.A(1);
        theta(k+1) = c.lineSegment.A(2);
        theta(k+2) = c.lineSegment.B(1);
        theta(k+3) = c.lineSegment.B(2);
        k = k+4;
    end
end
```

- File `SPCLS_initialization.m`

```
function [segments] = SPCLS_initialization(pixelImage, T, rho)
    A = ~pixelImage;
    [rows, cols] = size(A);
    B = zeros(rows, cols);
    for i = 1:T:rows
       B(i,:) = A(i,:);
    end
    C = zeros(rows, cols);
    for j = 1:T:cols
        C(:,j) = A(:,j);
    end
    D = ~((B + C) == 0);
    E = zeros(rows, cols);
    centers(1,:) = [0,0] ;
    k = 1;
    for i=1:rows
        for j=1:cols
            if D(i,j) == 1
                [x1, y1, len_hor, D] = locate_horizontal_segment(D, [i,j
    ]);
                [x2, y2, len_vert, D] = locate_vertical_segment(D, [i,j])
    ;
                if x1 == x2 && y1 == y2
                    E(x1, y1) = 1;
                    centers(k,:) = [y1,x1];
                    k = k + 1;
                else
                    if len_hor > 1
                        E(x1,y1) = 1;
                        centers(k,:) = [y1,x1];
                        k = k + 1;
                    end
                    if len_vert > 1
```

```
                        E(x2,y2) = 1;
                        centers(k,:) = [y2,x2];
                        k = k + 1;
                    end
                end
            end
        end
    end
    if  length(centers) < 2 || length(centers(:,1)) < 2
        error("Initialization could not locate any segments. Bad image or
    high T value");
    end
    segments = convert_to_segments(centers, rho);
end

function [x, y, len, D_new] = locate_horizontal_segment(D, start_)
    D_new = D;
    [~, cols] = size(D);
    i0 = start_(1); j0 = start_(2);
    len = 1;
    for j=j0+1:cols
        if D(i0,j) == 1
            len = len+1;
            D_new(i0,j) = 0;
        else
            break
        end
    end
    [x, y] = find_midpoint(start_, [i0, j0+len-1]);
end

function [x, y, len, D_new] = locate_vertical_segment(D, start_)
    D_new = D;
    [rows, ~] = size(D);
    i0 = start_(1); j0 = start_(2);
    len = 1;
    for i=i0+1:rows
        if D(i,j0) == 1
            len = len+1;
            D_new(i,j0) = 0;
        else
            break
        end
    end
    [x, y] = find_midpoint(start_, [i0+len-1, j0]);
end

function initial_segments = convert_to_segments(centers, rho)
    initial_segments = zeros(size(centers)*2);
    for k=1:length(centers)
        m = centers(k,:);
        v = rand(1,2);
        a = m - rho*v;
        b = m + rho*v;
        initial_segments(k,:) = [a,b];
    end
end

function [midpoint_x, midpoint_y] = find_midpoint(start_, end_)
    midpoint_x = ceil( (start_(1) + end_(1)) / 2 );
    midpoint_y = ceil( (start_(2) + end_(2)) / 2 );
end
```

- File `copy_clusters.m`

```
function curr_clusters = copy_clusters(clusters)
    for i=1:length(clusters)
        c_old = clusters(i);
        c_new = Cluster(c_old.id, c_old.lineSegment.A, c_old.lineSegment.
   B, c_old.sigma, c_old.aPriori);
        c_new.color = c_old.color;
        curr_clusters(i) = c_new;
    end
end
```

REFERENCES

[1] T. Kanade, “Region segmentation: Signal vs semantics,” *Computer Graphics and Image Processing*, vol. 13, pp. 279–297, 08 1980.

[2] E. P. Lyvers and O. R. Mitchell, “Precision edge contrast and orientation estimation,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 10, pp. 927–937, Nov. 1988.

[3] J. F. Canny, “Finding edges and lines in images,” 1983.

[4] D. Ziou and S. Tabbone, “Edge detection techniques - an overview,” *INTERNATIONAL JOURNAL OF PATTERN RECOGNITION AND IMAGE ANALYSIS*, vol. 8, pp. 537–559, 1998.

[5] T. Poggio, H. Voorhees, and A. Yuille, “A regularized solution to edge detection,” *Journal of Complexity*, vol. 4, no. 2, pp. 106 – 123, 1988.

[6] P. Heinonen, I. Defée, A. Nieminen, and Y. Neuvo, “Median based algorithms for image enhancement and edge detection,” *IFAC Proceedings Volumes*, vol. 19, no. 9, pp. 33 – 39, 1986. 1st IFAC Workshop on Digital Image Processing in Industrial Applications, Espoo, Finland, 10-12 June 1986.

[7] J. M. Prewitt, “Object enhancement and extraction,” *Picture processing and Psychopictorics*, vol. 10, no. 1, pp. 15–19, 1970.

[8] H. E. Marr D. and B. Sydney, “Theory of edge detection,” *Proceedings of the Royal Society of London. Series B. Biological Sciences*, vol. 207, 2 1980.

[9] J. F. Canny, “A computational approach to edge detection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, pp. 679–698, 1986.

[10] P. V. Hough, “Method and means for recognizing complex patterns,” 12 1962. US Patent 3,069,654.

[11] J. Matas, C. Galambos, and J. Kittler, “Robust detection of lines using the progressive probabilistic hough transform,” *Computer Vision and Image Understanding*, vol. 78, no. 1, pp. 119 – 137, 2000.

[12] D. Ballard, “Generalizing the hough transform to detect arbitrary shapes,” *Pattern Recognition*, vol. 13, no. 2, pp. 111 – 122, 1981.

[13] R. G. Von Gioi, J. Jakubowicz, J.-M. Morel, and G. Randall, “Lsd: A fast line segment detector with a false detection control,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 4, pp. 722–732, 2010.

[14] R. C. Gonzalez and R. E. Woods, *Digital Image Processing (3rd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006.

[15] N. Dhanachandra, K. Manglem, and Y. J. Chanu, “Image segmentation using k -means clustering algorithm and subtractive clustering algorithm,” *Procedia Computer Science*, vol. 54, pp. 764 – 771, 2015.

[16] S. P. Lloyd, “Least squares quantization in pcm,” *IEEE Transactions on Information Theory*, vol. 28, pp. 129–137, 1982.

[17] J. MacQueen, “Some methods for classification and analysis of multivariate observations,” in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, (Berkeley, Calif.), pp. 281–297, University of California Press, 1967.

[18] T.-Y. Phillips and A. Rosenfeld, “An isodata algorithm for straight line fitting,” *Pattern Recognition Letters*, vol. 7, no. 5, pp. 291 – 297, 1988.

[19] P.-Y. Yin, “Algorithms for straight line fitting using k-means1this research was supported in part by the national science council of r.o.c., under contract nsc-86-2621-e-130-002-t.1,” *Pattern Recognition Letters*, vol. 19, no. 1, pp. 31 – 41, 1998.

[20] J. C. R. Thomas, “A new clustering algorithm based on k-means using a line segment as prototype,” in *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications* (C. San Martin and S.-W. Kim, eds.), (Berlin, Heidelberg), pp. 638–645, Springer Berlin Heidelberg, 2011.

[21] J. Bezdek, C. Coray, R. Gunderson, and J. Watson, “Detection and characterization of cluster substructure i. linear structure: Fuzzy $c$lines,” *Siam Journal on Applied Mathematics - SIAMAM*, vol. 40, 04 1981.

[22] K. D. Koutroumbas, “Introducing sparsity in possibilistic clustering: A unified framework and a line detection paradigm,” *IEEE Transactions on Fuzzy Systems*, vol. 26, pp. 2886–2898, 10 2018.

[23] T. Long, W. Jiao, G. He, and W. Wang, “Automatic line segment registration using gaussian mixture model and expectation-maximization algorithm,” *Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of*, vol. 7, pp. 1688–1699, 05 2014.

[24] A. P. Dempster, N. M. Laird, and D. B. Rubin, “Maximum likelihood from incomplete data via the em algorithm,” *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 39, no. 1, pp. 1–22, 1977.

[25] M. R. Gupta and Y. Chen, “Theory and use of the em algorithm,” *Foundations and Trends® in Signal Processing*, vol. 4, no. 3, pp. 223–296, 2011.

[26] S. Theodoridis and K. Koutroumbas, “Pattern recognition. 2003,” *Elsevier Inc*, 2009.

[27] D. R. Jones, *Direct global optimization algorithmDirect Global Optimization Algorithm*, pp. 725–735. Boston, MA: Springer US, 2009.

[28] L. Bottou, “Large-scale machine learning with stochastic gradient descent,” in *Proceedings of COMP-STAT’2010* (Y. Lechevallier and G. Saporta, eds.), (Heidelberg), pp. 177–186, Physica-Verlag HD, 2010.

[29] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[30] P. E. H. Richard O. Duda, “Pattern classification and scene analysis,” *The Library Quarterly*, vol. 44, no. 3, pp. 258–259, 1973.