Fuzzy and Possibilistic Shell Clustering Algorithms and Their Application to Boundary Detection and Surface Approximation—Part I

Raghu Krishnapuram, *Member, IEEE*, Hichem Frigui, and Olfa Nasraoui

Abstract— **Traditionally, prototype-based fuzzy clustering algorithms such as the Fuzzy C Means (FCM) algorithm have been used to find "compact" or "filled" clusters. Recently, there have been attempts to generalize such algorithms to the case of hollow or "shell-like" clusters, i.e., clusters that lie in subspaces of feature space. The shell clustering approach provides a powerful means to solve the hitherto unsolved problem of simultaneously fitting multiple curves/surfaces to unsegmented, scattered and sparse data. In this paper, we present several fuzzy and possibilistic algorithms to detect linear and quadric shell clusters. We also introduce generalizations of these algorithms in which the prototypes represent sets of higher-order polynomial functions. The suggested algorithms provide a good trade-off between computational complexity and performance. Since the objective function used in these algorithms is the sum of squared distances, the clustering is sensitive to noise and outliers. We show that by using a possibilistic approach to clustering, one can make the proposed algorithms robust.**

I. Introduction

CLUSTERING methods have been used extensively in pattern recognition and computer vision [27]. Objective function based clustering methods are one particular class of clustering methods in which a criterion function is iteratively minimized until a global or local minimum is reached. Objective function based clustering can be either hard (crisp) or fuzzy, depending on whether each feature vector belongs exclusively to one cluster or to all clusters to different degrees. In general, the performance of fuzzy algorithms is superior to that of the corresponding hard versions, and they have a lower tendency to get stuck in local minima [4].

In objective function based clustering algorithms, each cluster is usually represented by a prototype, and the sum of distances from the feature points to the prototypes is used as the objective function. This method has been traditionally used to detect "compact" or "filled" clusters in feature spaces, whose prototypes are typically represented by cluster centers and cluster covariance matrices. The Fuzzy C Means (FCM) algorithm [4] and its derivatives [12], [20], [23] may be used to find clusters that resemble filled hyper spheres or filled hyper ellipsoids. Lately this approach has been extended to the case of hollow or shell-like clusters by using shells (manifolds) for prototypes and measuring the distances to the shells rather than to the cluster centers. Coray seems to have been the first to suggest the use of this idea to find circular clusters [10]. More recently, Davé's Fuzzy C Shells (FCS) algorithm [13] and the Adaptive Fuzzy C-Shells (AFCS) algorithm [16] have proven to be successful in detecting circular and elliptical shapes. These algorithms are computationally rather intensive since one needs to solve coupled nonlinear equations to update the shell parameters in every iteration [5]. They also assume that the number of clusters is known. A computationally simpler Fuzzy C Spherical Shells algorithm for clustering hyperspherical shells and an unsupervised version to be used when the number of clusters is unknown have also been introduced [36]. Extensions to more general quadric shapes have also been proposed [16], [31], [32]. One problem with the proposed extensions is that they use a highly nonlinear algebraic distance, which results in unsatisfactory performance when the data are scattered [16], [32]. Finally, none of the above shell clustering algorithms can deal with situations in which the clusters include lines/planes and there is much noise. In this paper, we address these drawbacks in more detail and present new fuzzy and possibilistic algorithms to overcome these drawbacks.

The algorithms proposed in this paper can be used for simultaneously fitting a given number of parameterized curves/surfaces to an unsegmented data set. They are particularly useful for boundary detection and surface approximation in computer vision, especially when the edges are jagged or when the range data is sparse and noisy. In Section X, we qualitatively compare the shell clustering approach with the more traditional generalized Hough transform approach for boundary detection. In Part II of this paper, we discuss problems associated with conventional boundary detection and range image segmentation methods in more detail and present unsupervised shell clustering algorithms that use a new cluster validity measure to overcome these problems.

In Section II, we briefly describe prototype-based fuzzy clustering. In Sections III–VII, we introduce several fuzzy shell clustering algorithms. Although the algorithms proposed in these sections are specifically designed to seek clusters that can be described by segments of second-degree curves, (or by segments of shells of hyperquadrics), they can be generalized easily to deal with shells of more complex types. In Section VIII, we present one such generalization in which the prototypes correspond to sets of higher-order polynomial

Manuscript received February 3, 1993; revised April 25, 1994. This work was supported in part by the Alexander von Humboldt Foundation, Germany.

The authors are with the Department of Electrical and Computer Engineering, University of Missouri-Columbia, Columbia, MO 65211 USA.

IEEE Log Number 9406652.

functions. In Section IX, we describe a possibilistic approach to clustering, which has the advantage that the partition and the prototype estimates are much less sensitive to noise when compared with the fuzzy approach.

II. Prototype-Based Fuzzy Clustering

Let $\mathbf{X} = \{\mathbf{x}_j \mid j = 1 \cdots N\}$ be a set of feature vectors in an n-dimensional feature space with coordinate-axis labels $[x_1, x_2, \cdots, x_n]$, where $\mathbf{x}_j = [x_{j1}, x_{j2}, \cdots, x_{jn}]^T$. Let $B = (\beta_1, \cdots, \beta_C)$ represent a C-tuple of prototypes each of which characterizes one of the C clusters. Each β_i consists of a set of parameters. In the following, we use β_i to denote both cluster i and its prototype. Let u_{ij} represent the grade of membership of feature point $\mathbf{x}_j$ in cluster β_i. The $C \times N$ matrix $\mathbf{U} = [u_{ij}]$ is called a constrained fuzzy C-partition matrix if it satisfies the following conditions [4], [25]

$$u_{ij} \in [0,1] \text{ for all } i, \quad 0 < \sum_{j=1}^{N} u_{ij} < N \text{ for all } i, j \quad \text{and}$$
$$\sum_{i=1}^{C} u_{ij} = 1 \text{ for all } j. \tag{1}$$

The problem of fuzzily partitioning the feature vectors into C clusters can be formulated as the minimization of an objective function $J(B, \mathbf{U}; \mathbf{X})$ of the form

$$J(B, \mathbf{U}; \mathbf{X}) = \sum_{i=1}^{C} \sum_{j=1}^{N} (u_{ij})^m d^2(\mathbf{x}_j, \beta_i). \tag{2}$$

In the above equation, $m \in [1, \infty)$ is a weighting exponent called the fuzzifier, and $d^2(\mathbf{x}_j, \beta_i)$ represents the distance from a feature point $\mathbf{x}_j$ to the prototype β_i. Minimization of the objective function with respect to $\mathbf{U}$ subject to the constraints in (1) gives us [4]

$$\left.\begin{aligned} u_{ij} &= \frac{1}{\sum_{k=1}^{C} \left(\frac{d^2(\mathbf{x}_j, \beta_i)}{d^2(\mathbf{x}_j, \beta_k)} \right)^{\frac{1}{m-1}}} \quad \text{if } I_j = \emptyset \\ u_{ij} &= 0 \quad i \notin I_j \\ \textstyle\sum_{i \in I_j} u_{ij} &= 1 \quad i \in I_j \end{aligned}\right\} \quad \text{if } I_j \neq \emptyset \tag{3}$$

where $I_j = \{i \mid 1 \leq i \leq C, d^2(\mathbf{x}_j, \beta_i) = 0\}$. Minimization of $J(B, \mathbf{U}; \mathbf{X})$ with respect to B varies according to the choice of the prototypes and the distance measure. For example, in the FCM algorithm, the clusters are usually assumed to be compact and spherical in shape, and each of the prototypes is described by the cluster center $\mathbf{c}_i$. If the distance measure is Euclidean or an inner product induced norm metric, these centers may be updated in each iteration using [4]

$$\mathbf{c}_i = \frac{1}{N_i} \sum_{j=1}^{N} (u_{ij})^m \mathbf{x}_j \tag{4}$$

where

$$N_i = \sum_{j=1}^{N} (u_{ij})^m. \tag{5}$$

In the Gustafson–Kessel (G-K) algorithm, the prototypes consist of the cluster centers $\mathbf{c}_i$ and the covariance matrices $\mathbf{C}_i$ [23]. It uses the distance measure $d^2(\mathbf{x}_j, \mathbf{c}_i) = |\mathbf{C}_i|^{1/n} (\mathbf{x}_j - \mathbf{c}_i)^T \mathbf{C}_i^{-1} (\mathbf{x}_j - \mathbf{c}_i)$. The centers are updated as above, and the covariance matrices are updated by

$$\mathbf{C}_i = \frac{1}{N_i} \sum_{j=1}^{N} (u_{ij})^m (\mathbf{x}_j - \mathbf{c}_i)(\mathbf{x}_j - \mathbf{c}_i)^T. \tag{6}$$

The G-K algorithm and the unsupervised fuzzy partition–optimum number of clusters algorithm due to Gath and Geva [20], assume that the clusters are compact ellipsoids and allow each cluster to have a different size and orientation. They can also be used to detect linear clusters in 2-D and planar clusters in 3-D, since these are extreme cases of ellipsoids [35].

The general form of prototype-based clustering algorithms is given below.

PROTOTYPE-BASED FUZZY CLUSTERING

Fix the number of clusters C; fix $m, m \in [1, \infty)$*;*
Initialize the fuzzy C-partition $\mathbf{U}$;
REPEAT
Update the parameters of each cluster prototype;
Update the partition matrix $\mathbf{U}$ *by using* (3);
UNTIL$(\|\Delta \mathbf{U}\| < \varepsilon)$;

The hard (crisp) versions of these algorithms are easily obtained by changing the updating rule for the memberships so that they are always binary. In other words, one uses

$$u_{ij} = \begin{cases} 1 & \text{if } d^2(\mathbf{x}_j, \beta_i) < d^2(\mathbf{x}_j, \beta_k) \quad \text{for all } k \\ 0 & \text{otherwise.} \end{cases}$$

Ties are broken arbitrarily. In practice, the hard versions do not perform as well as their fuzzy counterparts. Due to space limitation, we do not deal with the hard algorithms in this paper.

III. The Fuzzy C Quadric Shells (FCQS) Algorithm

This algorithm [31], [32] assumes that each cluster resembles a hyperquadric surface, and the prototypes β_i consist of parameter vectors $\mathbf{p}_i$ which define the equations of the hyperquadric surfaces. The general equation for such a hyperquadric surface is

$$\mathbf{p}_i^T \mathbf{q} = 0 \tag{7}$$

where

$$\begin{aligned} \mathbf{p}_i^T &= [p_{i1}, p_{i2}, \cdots, p_{in}, p_{i(n+1)}, \cdots, \\ &\qquad p_{ir}, p_{i(r+1)}, p_{i(r+2)}, \cdots, p_{i(r+n)}, p_{is}]. \\ \mathbf{q}^T &= [x_1^2, x_2^2, \cdots, x_n^2, x_1 x_2, \cdots, \\ &\qquad x_{n-1} x_n, x_1, x_2, \cdots, x_n, 1] \end{aligned} \tag{8}$$

and $s = \frac{n(n+1)}{2} + n + 1 = r + n + 1$. We may define the algebraic (or residual) distance from a point $\mathbf{x}_j$ to a prototype β_i as

$$d^2(\mathbf{x}_j, \beta_i) = d_{Qij}^2 = \mathbf{p}_i^T \mathbf{q}_j \mathbf{q}_j^T \mathbf{p}_i = \mathbf{p}_i^T \mathbf{M}_j \mathbf{p}_i \tag{9}$$

where

$$\mathbf{M}_j = \mathbf{q}_j\mathbf{q}_j^T, \text{ with } \mathbf{q}_j^T = \left[x_{j1}^2, x_{j2}^2, \cdots, x_{jn}^2, x_{j1}x_{j2}, \cdots, x_{j(n-1)}x_{jn}, x_{j1}, x_{j2}, \cdots, x_{jn}, 1\right].$$

To obtain a fuzzy C-partition of the data, we may minimize the objective function in (2) with d_{Qij}^2 as the underlying distance measure. Since the objective function is homogeneous with respect to $\mathbf{p}_i$, however, we need to constrain the problem to avoid the trivial solution. Some of the possibilities are

$$\begin{aligned} &\text{i) } \|\mathbf{p}_i\|^2 = 1, \quad \text{ii) } p_{i1}^2 + \cdots + p_{i(r+n)}^2 = 1, \\ &\text{iii) } p_{i1} = 1, \quad \text{iv) } p_{is}^2 = 1, \quad \text{and} \\ &\text{v) } \|p_{i1}^2 + p_{i2}^2 + \cdots + p_{in}^2 + \tfrac{1}{2}p_{i(n+1)}^2 \\ &\qquad + \tfrac{1}{2}p_{i(n+2)}^2 + \cdots + \tfrac{1}{2}p_{ir}^2\|^2 = 1. \end{aligned} \tag{10}$$

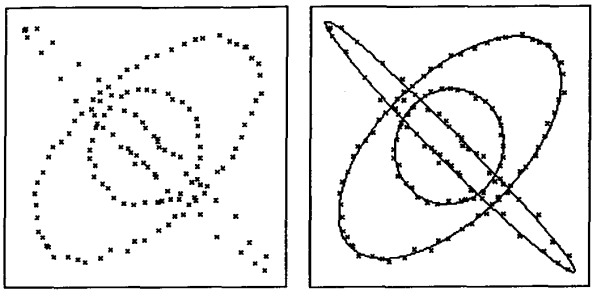
Constraints i) [41] and ii) [21] do not make the distance measure invariant to translation and rotation of the data. They do allow the solution to be linear or planar. Constraint iii) was used by Chen [9] and Krishnapuram *et al.* [36] for circles and by Davé and Bhaswan [16] for quadric curves. It precludes linear solutions and can lead to instabilities or poor performance when the data points are approximately linear (planar) as in the case of partial circles (spheres) with very large radii [42]. Moreover, in the case of noncircular (nonspherical) quadrics, this constraint makes the distance measure rotation-variant, which is undesirable. Constraint iv) [2], [6], [11] has the problem that no conic that passes through the origin satisfies it. The last constraint was imposed by Bookstein [7], [45] and has the advantage that the resulting distance measure is invariant to rigid transformations of the prototype. It does not allow, however, the solution to be linear or planar. Pratt showed that this is a great disadvantage and proposed a new quadratic constraint for the case of circles [42]. Taubin [46] generalized this idea for fitting an implicit polynomial curve to a data set. This constraint will be discussed in Section VI. Bookstein's Constraint, i.e., Constraint v) above, is in our experience the best compromise between computational complexity and performance. Agin [1] also discusses some of these issues.

If we define

$$a_{ik} = \begin{cases} p_{ik} & 1 \le k \le n \\ \dfrac{p_{ik}}{\sqrt{2}} & n+1 \le k \le r \end{cases}, \quad \text{and}$$
$$b_{ik} = p_{i(r+k)} \quad 1 \le k \le s-r$$

then constraint v) in (10) simplifies to $\|a_i\|^2=1$. It is easily verified [19], [31] that the solution is

$$\begin{aligned} \mathbf{a}_i &= \text{eigenvector of } (\mathbf{F}_i - \mathbf{G}_i^T\mathbf{H}_i^{-1}\mathbf{G}_i) \text{associated} \\ &\quad \text{with the smallest eigenvalue, and} \\ \mathbf{b}_i &= -\mathbf{H}_i^{-1}\mathbf{G}_i\mathbf{a}_i \end{aligned} \tag{11}$$



(a) (b)

Fig. 1. (a) A data set consisting of two ellipses and a circle. (b) The prototypes found by the FCQS algorithm superimposed on the dataset.

where

$$\mathbf{F}_i = \sum_{j=1}^{N}(u_{ij})^m\mathbf{R}_j, \quad \mathbf{G}_i = \sum_{j=1}^{N}(u_{ij})^m\mathbf{S}_j,$$
$$\mathbf{H}_i = \sum_{j=1}^{N}(u_{ij})^m\mathbf{T}_j,$$
$$\mathbf{R}_j = \mathbf{r}_j\mathbf{r}_j^T, \quad \mathbf{S}_j = \mathbf{r}_j\mathbf{t}_j^T, \quad \mathbf{T}_j = \mathbf{t}_j\mathbf{t}_j^T,$$
$$\mathbf{r}_j^T = \left[x_{j1}^2, x_{j2}^2, \cdots, x_{jn}^2, \sqrt{2}x_{j1}x_{j2}, \cdots, \sqrt{2}x_{jk}x_{jl}, \cdots, \sqrt{2}x_{j(n-1)}x_{jn}\right], \text{ and } \mathbf{t}_j^T = [x_{j1}, x_{j2}, \cdots, x_{jn}, 1].$$

It is to be noted that $\mathbf{H}_i^{-1}$ exists as long as there are at least $n+1$ noncollinear feature points in the data set. Thus, in the FCQS algorithm, the parameters are updated using (11), and the memberships are updated using (3) except that $d^2(\mathbf{x}_j, \beta_i)$ is replaced by d_{Qij}^2.

Shell clustering algorithms are quite sensitive to how one initializes the C-partition. In our experience, a few (typically 10) iterations of the FCM algorithm followed by a few (typically 10) iterations of the G-K algorithm and a few (typically five) iterations of the Fuzzy C Spherical Shells algorithm [36] provides a good initialization for the FCQS algorithm. While using the FCM algorithm for initialization, a value of 3.0 for the fuzzifier m seems to give good results. This is because the FCM is not really meant for shell clusters and a higher value of m gives a fuzzier partition, which is more desirable for initialization purposes. In all other algorithms, a value of 2.0 for m works best in practice.

Fig. 1 shows a typical example of the results obtained by the FCQS algorithm with a synthetic data set containing about 200 points. Fig. 1(a) shows the 200×200 image of the original data set. A uniformly distributed noise with an interval of 3.0 was added to the x and y locations of the data points so that the points do not lie on ideal curves. Fig. 1(b) shows the resulting prototype curves superimposed on the original data set. The number of clusters was assumed to be known. The algorithm typically converges in about 20 iterations and the CPU time on a Sun Sparc 1 workstation is less than 10 s.

Since the FCQS algorithm uses the algebraic distance given by (9) which is highly nonlinear in nature, the membership assignments are not very meaningful. Moreover, when there are curves (surfaces) of highly-varying sizes, the algebraic

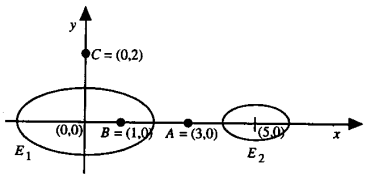


Fig. 2. An example to illustrate the sensitivity of the algebraic distance to the size of the curve as well as location of the feature point with respect to the curve. Points A, B, and C are all geometrically equidistant from the bigger ellipse, and point A is equidistant from both ellipses. However, the algebraic distance does not reflect this.

distance is biased towards smaller curves (surfaces), and for a particular curve (surface) it is biased towards points inside the curve (surface) as opposed to points outside. Thus, the distance measure gives rather eccentric and highly curved fits if the data is scattered as the prototypes try to enclose more points inside the curve. The distance also is sensitive to the placement of the feature point with respect to the curve. Consider for example, two ellipses E_1 and E_2 as shown in Fig. 2. Ellipse E_1 is centered at (0, 0) and its major and minor semi-axes have the values 2 and 1. Ellipse E_2 is centered at (5, 0) and its major and minor semi-axes have the values 1 and 1/2. Consider points $A = (3,0)$, $B = (1,0)$ and $C = (0,2)$. Note that all three points are equidistant from E_1 in the Euclidean sense, and point A is equidistant from both E_1 and E_2. Keeping in mind constraint iv) in (10), we can write the expressions for the algebraic distances of a point $\mathbf{x} = (x, y)$ from E_1 and E_2 as $d^2(\mathbf{x}, E_1) = (16/17)\{x^2/4 + y^2 - 1\}^2$ and $d^2(\mathbf{x}, E_2) = (1/17)\{(x-5)^2 + 4y^2 - 1\}^2$. Thus, we see that $d^2(A, E_1) = 25/17, d^2(B, E_1) = 9/17, d^2(C, E_1) = 144/17$, and $d^2(A, E_2) = 9/17$. This clearly illustrates the bias of this distance measure as discussed above. Another problem with the nonlinear distance is that it makes the fit of each curve rather sensitive to the presence of other curves, which sometimes leads to unstable hyperbolic fits. An example is shown in the next section.

Here we would like to note that although the distance used in the AFCS algorithm [16] for elliptical clusters is not biased towards points inside the curve, it is still sensitive to the size of the curve as well as the placement of the feature point with respect to the curve. For example, the expressions for the AFCS distance of a point $\mathbf{x} = (x, y)$ from E_1 and E_2 can be written as $d^2(\mathbf{x}, E_1) = \{[x^2/4 + y^2]^{1/2} - 1\}^2$ and $d^2(\mathbf{x}, E_2) = \{[(x-5)^2 + 4y^2]^{1/2} - 1\}^2$. Thus, we see that $d^2(A, E_1) = 1/4, d^2(B, E_1) = 1/4, d^2(C, E_1) = 1$ and $d^2(A, E_2) = 1$. In the next sections we introduce modifications to the basic FCQS algorithm to mitigate this problem, keeping in mind that one needs to keep the computational complexity as low as possible.

IV. Modifications to the Fuzzy C Quadric Shells Algorithm

One possible way to alleviate the problem due to the nongeometric nature of d^2_{Qij} is to use the geometric (perpendicular) distance denoted by d^2_{Pij} between the point $\mathbf{x}_i$ and the shell β_i. To compute d^2_{Pij} we first rewrite (7) as $\mathbf{x}^T\mathbf{A}_i\mathbf{x} + \mathbf{x}^T\mathbf{b}_i + c_i = 0$. Then the distance d^2_{Pij} can be obtained by minimizing $\|\mathbf{x}_j - \mathbf{z}\|^2$ subject to

$$\mathbf{z}^T\mathbf{A}_i\mathbf{z} + \mathbf{z}^T\mathbf{b}_i + c_i = 0 \tag{12}$$

where $\mathbf{z}$ is a point on the quadric β_i. By using a Lagrange multiplier λ, the solution is found to be

$$\mathbf{z} = \frac{1}{2}(\mathbf{I} - \lambda\mathbf{A}_i)^{-1}(\lambda\mathbf{b}_i + 2\mathbf{x}_j). \tag{13}$$

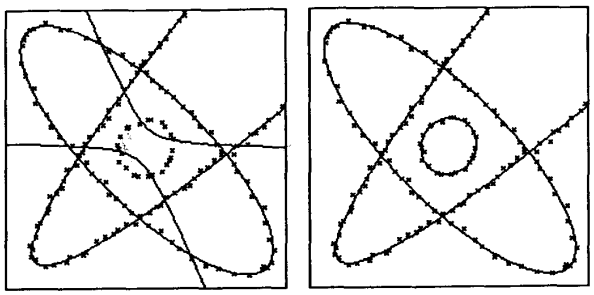
Substituting (13) in (12) yields a quartic (fourth-degree) equation in λ in the 2-D case (see Appendix A for details), and has at most four real roots $\lambda_k, k = 1, \cdots, 4$. The four roots can be computed using the standard closed-form solution. For higher dimensions, the equation is of sixth degree or higher, and iterative root finding techniques need to be used. For each real root λ_k so computed, we calculate the corresponding $\mathbf{z}$ vector $\mathbf{z}_k$ using (13). Then, we compute d^2_{Pij} using

$$d^2_{Pij} = \min_k \|\mathbf{x}_j - \mathbf{z}_k\|^2. \tag{14}$$

Minimization of the objective function in (2) with respect to $\mathbf{p}_i$ when d^2_{Pij} is used as the underlying distance measure can be achieved only by using iterative techniques such as the Levenberg–Marquardt algorithm [39], [46]. To overcome this problem, we may assume that we can obtain approximately the same values for $\mathbf{p}_i$ by using (11), which will be true if all the feature points lie reasonably close to the hyperquadric shells. This leads to a modified FCQS algorithm, in which the memberships are computed using d^2_{Pij}, but the parameters are updated using d^2_{Pij}. An alternative is to use the Levenberg–Marquardt algorithm after initializing it with the solution obtained by (11) in each iteration. This is implementationally and computationally more complex, however, and is recommended only for small data sets. We have observed that this can increase the CPU time by an order of magnitude, although the overall number of iterations required for the FCQS algorithm to converge is somewhat lower. Moreover, our simulations indicate that the performance of the modified FCQS algorithm is adequate for most computer vision applications. The initialization procedure recommended for the FCQS algorithm can also be used for the modified version with good results.

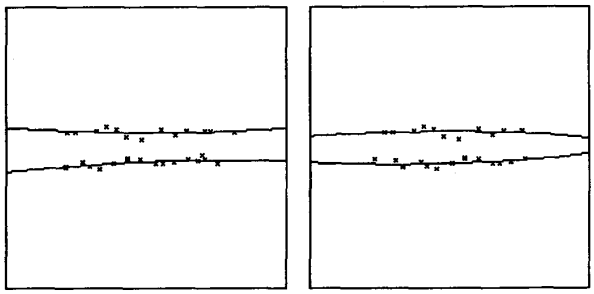
Fig. 3(a) shows a data set with three curves for which the original version of the FCQS algorithm fails. It can be seen that due to the presence of other curves, the fit for the circle becomes distorted, resulting in a hyperbola. Fig. 3(b) shows the result of the modified FCQS algorithm, illustrating the advantage of the more meaningful membership assignments in the modified version. It is to be noted that in the modified algorithm, although the memberships are based on geometric distances, the parameters are still estimated by minimizing the algebraic distance. This may give poor fits when the data is highly scattered. An example of this behavior is shown in Section VI. Another problem with the modified FCQS algorithm is that d^2_{Pij} has a closed-form solution only in the 2-D case. In higher dimensions, solving for d^2_{Pij} is not trivial.

Henceforth we will simply use the acronym FCQS to denote the modified version.



(a) (b)

Fig. 3. An example illustrating the advantage of the modified FCQS algorithm over the FCQS algorithm on a data set containing a circle, a parabola, and an ellipse. Prototypes of clusters found by (a) the FCQS algorithm and (b) the modified FCQS algorithm. The modified algorithm avoids the hyperbolic fit.



(a) (b)

Fig. 4. Examples illustrating the tendency of the FCQS algorithm to fit pathological prototypes to scattered linear data. (a) A "flat" hyperbolic fit for two parallel lines. (b) An extremely elongated elliptical fit for two parallel lines.

V. Line Detection Using the FCQS Algorithm

The FCQS algorithm can be used to find linear clusters, even though the constraint forces all prototypes to be of second degree. This is because in practice this algorithm fits a pair of coincident lines for a single line, a hyperbola for two intersecting lines, and a very "flat" hyperbola (see Fig. 4(a)), or an elongated ellipse (see Fig. 4(b)), or a pair of lines for two parallel lines. Hyperbolas and extremely elongated ellipses occur rarely in practice. When the data set contains many linear clusters, the FCQS algorithm characterizes them variously as hyperbolas, extremely elongated ellipses, etc. In this case, we can group all the points belonging to such pathological clusters into a data set and then run a line finding algorithm such as the G-K algorithm (see [30]) on this data set with an appropriate initialization. The parameters of the lines can be determined from the centers and the covariance matrices of the clusters. The line detection algorithm summarized below may be used after the FCQS algorithm converges.

THE LINE DETECTION ALGORITHM

Set χ, the set of all data points in linear clusters to $\emptyset$;
Set number of lines C to 0;
FOR *each cluster β_i* **DO**
IF *β_i is a pair of coincident lines* **THEN**
Add all points assigned to cluster β_i to the data set χ;
$C = C + 1$;
Initialize the new linear prototype as the one of the two coincident lines;
END IF;
IF *β_i is a nonflat hyperbola* **OR** *a pair of intersecting lines* **OR**
a pair of parallel lines **THEN**
Add all points assigned to cluster β_i to the data set χ;
$C = C + 2$;
Initialize the new linear prototypes as the asymptotes of the hyperbola or as the individual lines making up the pair of lines;
END IF;
IF *β_i is an ellipse with a very large major axis to minor axis ratio* **THEN**
Add all points assigned to cluster β_i to the data set χ;
$C = C + 2$;
Initialize the new linear prototypes as the two tangents to the ellipse at the two ends of the minor axis;
END IF;
IF *β_i is a hyperbola with a very large conjugate axis to transverse axis ratio* **THEN**
Add all points assigned to cluster β_i to the data set χ; $C = C + 2$;
Initialize the new linear prototypes as the two tangents to the hyperbola at its two vertices;
END IF;
END FOR;
Run the G-K algorithm on the data set χ with C clusters using the initialization for the prototype of each cluster.

Appendix B summarizes the various conditions that one needs to check to determine the nature of the second degree curve. The initialization procedures for the various cases in the line detection algorithm are described in Appendix C. Since the initialization is excellent, the G-K algorithm converges in a couple of iterations. The above algorithm successfully handles the pathological cases shown in Fig. 4.

VI. The Fuzzy C Plano-Quadric Shells (FCPQS) Algorithm

When the exact distance is too complex to compute, one could use what is known as the "approximate distance" (first-order approximation of the exact distance) given by [24], [46]

$$d^2(\mathbf{x}_j, \beta_i) = d^2_{Aij} = \frac{d^2_{Qij}}{|\nabla d_{Qij}|^2} = \frac{\mathbf{p}_i^T \mathbf{M}_j \mathbf{p}_i}{\mathbf{p}_i^T [D(\mathbf{q}_j) D(\mathbf{q}_j)^T] \mathbf{p}_i}. \quad (15)$$

where ∇d_{Qij} is the gradient of the functional $\mathbf{p}_i^T \mathbf{q}$ in (7) evaluated at $\mathbf{x}_j$ and the matrix $D(\mathbf{q}_j)$ is the Jacobian of $\mathbf{q}$ in (8) evaluated at $\mathbf{x}_j$ [46]. In other words, the approximate distance is simply the algebraic distance divided by the gradient magnitude. The objective function to be minimized in this

case may be formulated as

$$J_A(B,\mathbf{U};\mathbf{X}) = \sum_{i=1}^{C}\sum_{j=1}^{N}(u_{ij})^m d_{Aij}^2 = \sum_{i=1}^{C}\sum_{j=1}^{N}(u_{ij})^m \frac{\mathbf{p}_i^T\mathbf{M}_j\mathbf{p}_i}{\mathbf{p}_i^T[D(\mathbf{q}_j)D(\mathbf{q}_j)^T]\mathbf{p}_i}. \quad (16)$$

A suitable constraint needs to be chosen while minimizing (16) with respect to $\mathbf{p}_i$. One may use the constraint proposed by Taubin [46] given by

$$\frac{1}{N}\sum_{j=1}^{N}\mathbf{p}_i^T[D(\mathbf{q}_j)D(\mathbf{q}_j)^T]\mathbf{p}_i = 1. \quad (17)$$

This constraint has the advantage that it takes the data points into account and allows the degenerate case of lines and planes. It is meant for fitting a single curve to a given set of data points. Hence, we need to generalize it to the fuzzy case and to fit C curves simultaneously. This is achieved by changing the constraint to [33]

$$\mathbf{p}_i^T\left[\sum_{j=1}^{N}(u_{ij})^m[D(\mathbf{q}_j)D(\mathbf{q}_j)^T]\right]\mathbf{p}_i = N_i \quad \text{for } i = 1,\cdots,C, \quad \text{or} \quad (18)$$

$$\mathbf{p}_i^T\mathbf{D}_i\mathbf{p}_i = N_i \quad \text{for } i = 1,\cdots,C \quad (19)$$

where N_i is as in (5), and

$$\mathbf{D}_i = \sum_{j=1}^{N}(u_{ij})^m[D(\mathbf{q}_j)D(\mathbf{q}_j)^T]. \quad (20)$$

Minimization of (16) with respect to $\mathbf{p}_i$ subject to (19) yields a complicated equation which cannot be solved for $\mathbf{p}_i$ explicitly. To avoid an iterative solution, we make the following assumptions:

i) All data points are reasonably close to some cluster, i.e., the u_{ij} are close to being hard. This assumption is valid when the data is not very scattered. (It is reasonable for the noisy case if we use possibilistic memberships, as will be discussed in Section IX.)

ii) The magnitude of the gradient at all points $\mathbf{x}_j$ that have a high membership in β_i is approximately constant, i.e., $|\nabla d_{Qij}|^2 = \mathbf{p}_i^T D(\mathbf{q}_j)D(\mathbf{q}_j)^T\mathbf{p}_i \approx 1$.

We now discuss the second assumption for the special case of hyperspheres. Hyperspheres are described by

$$\mathbf{p}_i^T\mathbf{q} = [p_{i1}, p_{i2}, \cdots, p_{i(n+1)}, p_{i(n+2)}] \times [(x_1^2 + x_2^2 + x_3^2 + \cdots + x_n^2), x_1, \cdots, x_n, 1]^T = 0. \quad (21)$$

Here $\mathbf{p}_i$ represents a prototype parameter vector. Let $\mathbf{x}_j = [x_{1j},\cdots,x_{nj}]^T$ denote a feature point, and let $\mathbf{q}_j = [(x_{1j}^2 + x_{2j}^2 + x_{3j}^2 + \cdots + x_{nj}^2), x_{1j},\cdots,x_{nj},1]^T$. Then we have

$$|\nabla d_{Qij}|^2 = (2p_{i1}x_{j1} + p_{i2})^2 + \cdots + (2p_{i1}x_{jn} + p_{i(n+1)})^2 = 4p_{i1}(p_{i1}x_{j1}^2 + \cdots + p_{i1}x_{jn}^2 + p_{i2}x_{j1} + \cdots + p_{i(n+1)}x_{jn}) + (p_{i2}^2 + \cdots + p_{i(n+1)}^2).$$

If a feature vector $\mathbf{x}_j$ lies on the prototype of cluster β_i, then it must satisfy (21) and the above expression becomes

$$|\nabla d_{Qij}|^2 = \mathbf{p}_i^T D(\mathbf{q}_j)D(\mathbf{q}_j)^T\mathbf{p}_i = -4p_{i1}p_{i(n+2)} + (p_{i2}^2 + \cdots + p_{i(n+1)}^2) = \text{constant} = \mathbf{K}.$$

When the memberships are almost hard, however, substituting the above result in (18), we get $N_iK \approx N_i$, or $K \approx 1$. Thus, assumption ii) is valid for circles (spheres) when $\mathbf{q}_j$ corresponds to a point on the curve (surface). It is obviously valid for the case of planes regardless of the location of $\mathbf{x}_j$ since $p_{i1} = 0$. It can also be shown that the assumption is valid for cylinders and rectangular hyperboloids. For other quadric shapes, this assumption does not hold. This issue will be discussed further in Section VII.

If $\mathbf{p}_i^T D(\mathbf{q}_j)D(\mathbf{q}_j)^T\mathbf{p}_i \approx 1$, we may ignore the denominator in (16). The simplified objective function is given by

$$J_A(B,\mathbf{U};\mathbf{X}) = \sum_{i=1}^{C}\sum_{j=1}^{N}(u_{ij})^m\mathbf{p}_i^T\mathbf{M}_j\mathbf{p}_i = \sum_{i=1}^{C}\mathbf{p}_i^T\mathbf{E}_i\mathbf{p}_i$$

where

$$\mathbf{E}_i = \sum_{j=1}^{N}(u_{ij})^m\mathbf{M}_j.$$

The above objective function is essentially the same as the one used in the FCQS algorithm, however, the constraint used is different. Minimization of the simplified objective function subject to the constraint in (19) leads to

$$\mathbf{E}_i\mathbf{p}_i = \lambda_i\mathbf{D}_i\mathbf{p}_i. \quad (22)$$

It is easily verified that

$$D(\mathbf{q}_j) = \begin{bmatrix}\mathbf{A}_1\\ \hline \mathbf{A}_2\\ \hline \mathbf{A}_3\end{bmatrix}, \text{ where } \mathbf{A}_1 = \begin{bmatrix} 2x_{j1} & 0 & \cdots & 0\\ 0 & 2x_{j2} & \cdots & 0\\ 0 & 0 & \cdots & 0\\ \cdot & \cdot & \cdots & \cdot\\ \cdot & \cdot & \cdots & \cdot\\ \cdot & \cdot & \cdots & \cdot\\ 0 & 0 & \cdots & 2x_{jn}\end{bmatrix},$$

$$\mathbf{A}_2 = \begin{bmatrix} x_{j2} & x_{j1} & \cdots & 0\\ x_{j3} & 0 & \cdots & 0\\ 0 & 0 & \cdots & 0\\ \cdot & \cdot & \cdots & \cdot\\ \cdot & \cdot & \cdots & \cdot\\ \cdot & \cdot & \cdots & \cdot\\ 0 & 0 & \cdots & x_{j(n-1)}\end{bmatrix}, \text{ and } \mathbf{A}_3 = \begin{bmatrix} 1 & 0 & \cdots & 0\\ 0 & 1 & \cdots & 0\\ 0 & 0 & \cdots & 0\\ \cdot & \cdot & \cdots & \cdot\\ \cdot & \cdot & \cdots & \cdot\\ \cdot & \cdot & \cdots & \cdot\\ 0 & 0 & \cdots & 1\\ 0 & 0 & \cdots & 0\end{bmatrix}.$$

Since the last row of $D(\mathbf{q}_j)$ is always equal to $[00\cdots0]$, $\mathbf{D}_i$ is singular, and the above generalized eigenvector problem cannot be converted to regular eigenvector problem, however, we may solve it using the QZ algorithm [46]. Care must be exercised while solving (22), because the matrices $\mathbf{D}_i$ and $\mathbf{E}_i$ are highly unbalanced. Several methods for balancing matrices

are available in the literature. Thus, (22) gives us the prototype update equation for the FCPQS algorithm. The membership update equation in the FCPQS algorithm is identical to (3), except that $d^2(\mathbf{x}_j, \beta_i)$ is replaced by d^2_{Aij}. In the special case of hyperspherical prototypes, this algorithm may be called the Fuzzy C Plano-Spherical Shells algorithm.

The FCPQS algorithm requires solving a six-dimensional generalized eigenvector problem (in the 2-D case) as opposed to a three-dimensional regular eigenvector problem. In addition to its computational complexity, we noticed that the FCPQS algorithm in general requires more iterations to converge than the FCQS algorithm. When the data set contains only one noiseless linear (planar) cluster among other nonlinear (nonplanar) clusters, the FCPQS algorithm does detect and characterize the linear (planar) cluster correctly. On the other hand, if the linear (planar) cluster is scattered or noisy, or if multiple linear (planar) clusters are present, the lines (planes) may be "overfitted" by a quadric curve (or surface). (One can always obtain a lower error of fit by fitting an extremely elongated ellipse to a scattered line.) The FCPQS algorithm also sometimes combines a pair of linear clusters into a single quadric cluster, a result similar to that of the FCQS algorithm. In such cases, the line detection procedure described in the previous section needs to be applied. Thus, in the 2-D case, this algorithm does not seem to have any advantage over the FCQS algorithm.

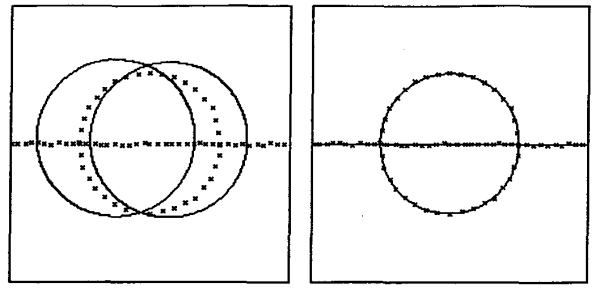
The real advantage of the FCPQS algorithm over the FCQS algorithm is in higher dimensions. It may be recalled (from Section IV) that the exact distance to a quadric has no closed-form solution in higher dimensions. Thus, (the modified version of) the FCQS algorithm is quite impractical. The performance of the FCPQS algorithm, however, is quite good, and its computational complexity is reasonably low. (See [46] for a discussion on the computational complexity of the QZ algorithm.) This is the primary reason we chose this algorithm to develop an unsupervised algorithm for surface fitting. This algorithm will be explained in Part II of this paper.

In the 3-D case, we noticed that the smallest eigenvalue solution of (22) may sometimes correspond to an "overfitted" prototype or to a surface prototype that almost never occurs in real images. Examples are hyperboloids of two sheets, hyperbolic cylinders, and imaginary quadric surfaces. Therefore, we accept the smallest eigenvalue solution of (22) only if it represents the parameter vector of an "acceptable" surface type, otherwise we keep checking the next eigenvectors (assuming that they are organized in ascending order of the eigenvalues) until we find the first one that represents an "acceptable" surface prototype. By an "acceptable" surface prototype, we mean the following: real ellipsoids, hyperboloids of one sheet, real quadric cones, elliptic paraboloids, real elliptic cylinders, parabolic cylinders, and (pairs of) planes. The different types of quadric surfaces and their identification conditions are listed in Appendix B.

In the 3-D case, many quadric surfaces such as cones, cylinders, and planes are not bounded surfaces, and have an infinite extent. Therefore, a given cluster having the prototype of one of these unbounded surface types will attract many other points belonging to other clusters if the points lie on the (infinite) extension of the given cluster. To avoid this problem, one may use a heuristically modified distance which is a convex combination of the approximate distance to the shell d_{A^2} and the Euclidean distance d_{E^2}. This modified distance d_{M^2} can be formulated as

$$d_{M^2} = (1 - \varepsilon) d_{A^2} + \varepsilon d_{E^2}$$

where ε is chosen such that the second term becomes comparable to the first term only for large d_{E^2}.The value of ε was of the order of 10^{-4} in our application. The objective function can be easily reformulated using the modified distance. The Euclidean distance d_{E^2} is measured from the statistical center (mean) of the cluster, and the center is updated in every iteration using (4). We recommend using the modified distance only as a method to refine the results obtained by the FCPQS algorithm. Thus, this distance may be used for a couple of iterations after the FCPQS algorithm converges.



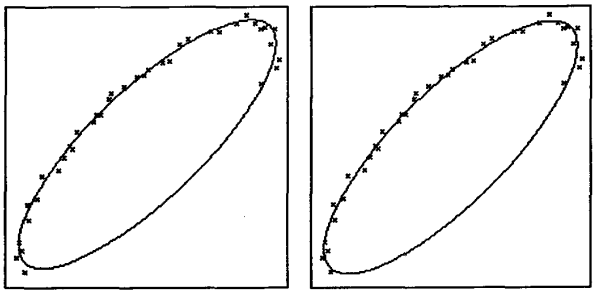
(a) (b)

Fig. 5. (a) Result of the Fuzzy C Spherical Shells algorithm on a data set containing a line and a circle. The prototypes are shown superimposed on the data set. The constraint used in this algorithm does not allow the degenerate case of lines. (b) Result of the Fuzzy C Plano-Spherical Shell algorithm.

Figs. 5(a) and 5(b) show the results of the Fuzzy C Spherical Shells algorithm [36] and the Fuzzy C Plano-Spherical Shells algorithm, respectively, on a data set consisting of a circle and a line. The prototypes obtained are superimposed on the original data. Unlike the Fuzzy C Spherical Shells algorithm, the Fuzzy C Plano-Spherical Shells algorithm with the constraint in (19) is able to characterize both the line and the circle correctly. Figs. 6(a) and 6(b) show the fit obtained by the FCQS and FCPQS algorithms for a scattered ellipse. (It is to be noted that the modification to the FCQS algorithm presented in Section IV has no effect if there is only one cluster in the data set, since all memberships are equal to 1.0.) It can be seen that the FCPQS fit is much better, even though the assumptions made in arriving at (22) are not valid for an ellipse. The sum squared error (measured using d_{P^2} for the two cases are 301.2 and 254.7, respectively. In the 2-D case, we observed that the FCPQS algorithm takes 20% to 30% more iterations to converge compared to the FCQS algorithm. The CPU requirements per iteration are about 10% to 20% higher. In higher dimensions, the difference will be considerably higher.

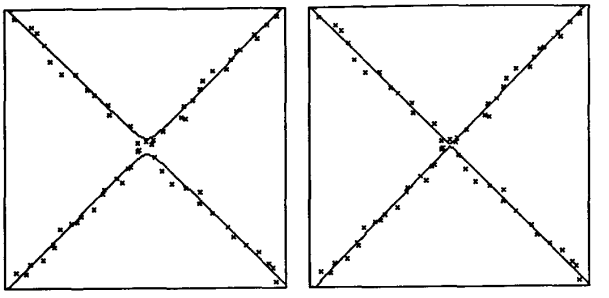
VII. A Weighting Procedure to Improve Fits

As discussed in the previous section, the assumption that $|\nabla d_{Qij}|^2 \approx$ constant for all $\mathbf{x}_j$ having a high membership in



(a) (b)

Fig. 6. An example illustrating the advantage of the FCPQS algorithm over the FCQS algorithm for scattered data. (a) Prototype found by the FCQS algorithm for a scattered ellipse. (b) Prototype found by the FCPQS algorithm for the same scattered ellipse.



(a) (b)

Fig. 7. Effect of the reweight procedure on the FCQS algorithm. The prototype for two intersecting lines found by the FCQS algorithm (a) without reweight and (b) with reweight.

β_i is always true for linear (planar) prototypes regardless of the location of the feature point $\mathbf{x}_j$ with respect to the prototype. Otherwise, this assumption is correct only for certain types of shells (such as circles and rectangular hyperbolas in 2-D, or spheres and cylinders in 3-D), even then only if all the feature points lie close to one of the shells. Thus, this assumption is invalid for ellipses and parabolas in 2-D, and ellipsoids and many other quadric shapes in 3-D. Therefore, the fit will be biased towards points where the gradient magnitude $|\nabla d_{Qij}|$ is high. One solution is to weight the distance measure to improve the fit [42], [46]. We achieve this by minimizing the weighted objective function given by

$$J_{AW}(\mathbf{B},\mathbf{U},\mathbf{W};\mathbf{X}) = \sum_{i=1}^{C}\sum_{j=1}^{N}(u_{ij})^m w_{ij}\mathbf{p_i^T}\mathbf{M}_j\mathbf{p}_i$$

subjected to

$$\mathbf{p}_i^T(\mathbf{D}_W)_i\mathbf{p}_i = N_i$$

where

$$(\mathbf{D}_W)_i = \sum_{j=1}^{N}(u_{ij})^m W_{ij}\ D(\mathbf{q}_j)D(\mathbf{q}_j)^T.$$

Since the purpose of introducing the weights w_{ij} is to reduce the bias due to the omission of the denominator in (16), ideally these weights should be chosen as

$$w_{ij} = \left[\mathbf{p}_i^T[D(\mathbf{q}_j)D(\mathbf{q}_j)^T]\mathbf{p}_i\right]^{-1}. \tag{24}$$

With this choice of w_{ij}, however, the objective function in (23) becomes identical to the one in (16), which cannot be minimized easily. To simplify the problem, we may treat the w_{ij} as constants which are updated in every iteration according to (24) using the parameter values $\mathbf{p}_i$ from the previous iteration. Using this heuristic, the parameters can be obtained by solving

$$(\mathbf{E}_W)_i\mathbf{p}_i = \lambda_i(\mathbf{D}_W)_i\mathbf{p}_i, \quad \text{where}$$
$$(\mathbf{E}_W)_i = \sum_{j=1}^{N}(u_{ij})^m W_{ij}\mathbf{M}_j. \tag{23}$$

Since this reweight procedure is heuristic, it is not guaranteed that the fit obtained after reweighting will always be better than the original fit. Therefore, in each iteration, we compute the parameter vector $\mathbf{p}_i$ both with and without the weights and accept the parameter vector $\mathbf{p}_i$ resulting from the reweight procedure only when the error of fit decreases. The sum of geometric or approximate distances for each individual cluster may be used as a measure of the error of fit.

The reweight procedure can also be adopted for the FCQS algorithm with the same choice of weights with good results. Following the same steps that were used above, it can be shown that the solution with the reweight procedure for the FCQS algorithm (subject to Bookstein's constraint) is given by the following equations instead of (11).

$\mathbf{a}_i$ = eigenvector of $\left((\mathbf{F}_W)_i - (\mathbf{G}_W)_i^T(\mathbf{H}_W)_i^{-1}(\mathbf{G}_W)_i\right)$ associated with the smallest eigenvalue, and

$$\mathbf{b}_i = -(\mathbf{H}_W)_i^{-1}(\mathbf{G}_W)_i\ \mathbf{a}_i$$

where

$$(\mathbf{F}_W)_i = \sum_{j=1}^{N}(u_{ij})^m w_{ij}\mathbf{R}_j,$$
$$(\mathbf{G}_W)_i = \sum_{j=1}^{N}(u_{ij})^m w_{ij}\mathbf{S}_j$$

and

$$(\mathbf{H}_W)_i = \sum_{j=1}^{N}(u_{ij})^m w_{ij}\mathbf{T}_j.$$

The FCQS algorithm with the reweight procedure produced the same fit for the scattered ellipse of Fig. 6 as the one produced by the FCPQS algorithm. Fig. 7(a) shows the results of the FCQS algorithm on a data set with two intersecting lines. In this figure, the fit is poor around the intersection, because the gradient magnitude is zero at the point of intersection. Fig. 7(b) shows the result of the FCQS algorithm with the reweight procedure. An obvious improvement in the fit can be seen.

In the case of the FCPQS algorithm, the use of the reweight procedure does not have much effect in 2-D, however, its effect can be significant in 3-D. Therefore, we recommend the use of the FCPQS algorithm with reweight in the 3-D case.

VIII. Generalization to Prototypes Represented by Sets of Higher-Order Polynomial Functions

Consider the set of zeros of $\mathbf{f}(\mathbf{x})$ defined by

$$f_1(\mathbf{x}) = 0; f_2(\mathbf{x}) = 0; \cdots, f_k(\mathbf{x}) = 0. \tag{25}$$

Here $\mathbf{x}^T = [x_1, x_2, \cdots, x_n]$ is an n-dimensional coordinate vector, as before. It is to be noted that $\mathbf{f}(\mathbf{x})$ is a vector consisting of k functions, all of which have to be simultaneously satisfied. Thus, if $n = 2$ and $k = 1$, we have a planar curve, if $n = 3$ and $k = 1$, we have a 3-D surface, and if $n = 3$ and $k = 2$, we have a space curve. For example, a circle of radius 1 on the xy-plane in 3-D is defined by the equations $x^2 + y^2 - 1 = 0$, and $z = 0$. Let

$$\mathbf{F}(\mathbf{x}) = [F_1(\mathbf{x}), F_2(\mathbf{x}), \cdots, F_h(\mathbf{x})]^T$$

be a vector of h polynomials. Each of the $F_i(\mathbf{x})$ is a polynomial of degree p or less in each of the coordinate axis labels $x_1, x_2, \cdots, x_n$. For example, when $n = 2$ and $k = 1$, and $p = 2$, $\mathbf{F}(\mathbf{x})$ can be chosen to be $\mathbf{q}T = [x_1^2, x_2^2, \cdots, x_n^2, x_1x_2, \cdots, x_{n-1}x_n, x_1, x_2, \cdots, x_n, 1]$ as in (8), in which case each element of $\mathbf{F}(\mathbf{x})$ is a monomial. For a suitable choice of $\mathbf{F}(\mathbf{x})$, we can write (25) as

$$\mathbf{f}(\mathbf{x}) = \mathbf{P}\mathbf{F}(\mathbf{x}) = 0$$

where $\mathbf{P}$ is a $k \times h$ matrix of parameters consisting of the coefficients of the functions $f_i(\mathbf{x})$. Thus $\mathbf{P}$ represents the prototype parameters, where each prototype is a set of functions $\mathbf{f}(\mathbf{x})$. We can construct an objective function based on C such prototypes as

$$J(\mathbf{\Pi}, \mathbf{U}; \mathbf{X}) = \sum_{i=1}^{C}\sum_{j=1}^{N}(u_{ij})^m \|\mathbf{P}_i\mathbf{F}(\mathbf{x}_j)\|^2 \tag{26}$$

where $\mathbf{\Pi} = (\mathbf{P}_1, \cdots, \mathbf{P}_C)$ represent a C-tuple of prototypes. Here $\|\mathbf{P}_i\mathbf{F}(\mathbf{x}_j)\|^2$ represents the algebraic distance from $\mathbf{x}_j$ to prototype $\mathbf{P}_i$. The above objective function can be written as

$$\begin{aligned} J(\mathbf{\Pi}, \mathbf{U}; \mathbf{X}) &= \sum_{i=1}^{C}\sum_{j=1}^{N}(u_{ij})^m \mathrm{Tr}[\mathbf{P}_i\mathbf{F}(\mathbf{x}_j)\mathbf{F}^T(\mathbf{x}_j)\mathbf{P}_i^T] \\ &= \sum_{i=1}^{C}\sum_{j=1}^{N}(u_{ij})^m \mathrm{Tr}[\mathbf{P}_i\mathbf{M}_j\mathbf{P}_i^T] \\ &= \sum_{i=1}^{C}\mathrm{Tr}[\mathbf{P}_i\mathbf{E}_i\mathbf{P}_i^T] \end{aligned}$$

where

$$\mathbf{M}_j = \mathbf{F}(\mathbf{x}_j)\mathbf{F}^T(\mathbf{x}_j), \quad \text{and} \quad \mathbf{E}_i = \sum_{j=1}^{N}(u_{ij})^m\mathbf{M}_j.$$

The above objective function can be minimized subject to the constraint

$$\begin{aligned} \mathbf{P}_i^T\left[\sum_{j=1}^{N}(u_{ij})^m[\mathbf{D}(\mathbf{x}_j)\mathbf{D}(\mathbf{x}_j)^T]\right]\mathbf{P}_i &= N_i\mathbf{I}_k \\ \text{for } i = 1, \cdots, C, \quad &\text{or} \\ \mathbf{P}_i^T\mathbf{D}_i\mathbf{P}_i = N_i\mathbf{I}_k \quad \text{for } i = 1, \cdots, C & \end{aligned} \tag{27}$$

where $\mathbf{I}_k$ is a $k \times k$ identity matrix, and

$$\mathbf{D}_i = \sum_{j=1}^{N}(u_{ij})^m[D(\mathbf{x}_j)D(\mathbf{x}_j)^T].$$

$D(\mathbf{x}_j)$ is the Jacobian matrix $D(\mathbf{x})$ evaluated at $\mathbf{x}_j$ given by

$$D(\mathbf{x}_j) = \begin{bmatrix} \frac{\partial F_1(\mathbf{x}_j)}{\partial x_1} & \cdots & \frac{\partial F_h(\mathbf{x}_j)}{\partial x_1} \\ \vdots & \cdots & \vdots \\ \frac{\partial F_1(\mathbf{x}_j)}{\partial x_n} & \cdots & \frac{\partial F_h(\mathbf{x}_j)}{\partial x_n} \end{bmatrix}.$$

It can be shown [46] that the solution to the minimization of (26) subject to (27) is given by the eigenvectors corresponding to the least k eigenvalues of the generalized eigenvector problem given below.

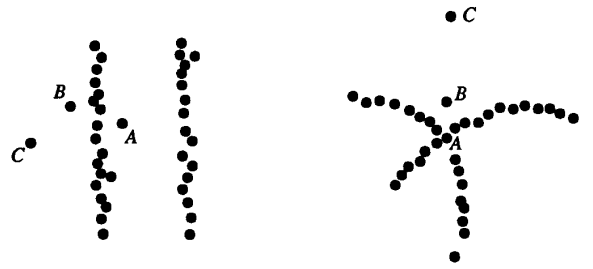
$$\mathbf{p}_i\mathbf{E}_i = \lambda_i\mathbf{p}_i\mathbf{D}_i.$$

Each of the eigenvector solutions $\mathbf{p}_i$ gives us one row of $\mathbf{P}_i$. As in the case of the FCPQS algorithm, the use of the constraint in (27) gives us fits that correspond to the approximate distance, even though the objective function uses the algebraic distance, especially when reweighting is used.

It is to be noted that in this general case, each prototype is represented by a set of higher-order polynomials. In this sense, the above algorithm is also a generalization of the Fuzzy C Regression Models (FCRM) introduced by Hathaway and Bezkek recently [25]. The FCRM considers two-dimensional prototypes of the form $y = f(x)$, where $f(x)$ is a polynomial. In other words, y is explicitly considered as a dependent variable, and higher powers of y do not appear in this model. The FCRM uses the algebraic distance, and the implicit constraint that the coefficient of y is one.

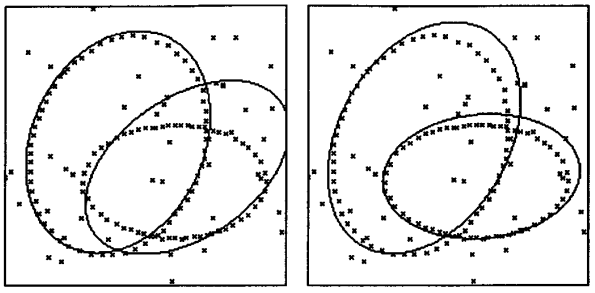
IX. Possibilistic Memberships for Robust Clustering

Fuzzy clustering algorithms do not always estimate the prototype parameters of the clusters accurately. The main source of this problem is the probabilistic constraint used in fuzzy clustering, which states that the memberships of a data point across all clusters must sum to one. [See (1)]. This problem has been discussed in detail in [34]. Here we present two simple examples to illustrate its drawbacks as related to shell clustering. Fig. 8(a) shows a situation where there are two linear clusters. Fuzzy clustering would produce very different (asymmetric) memberships in cluster 1 for points A and B, even though they are equidistant from the prototype. Similarly, point A and point C may have equal membership values in cluster 1, even though point C is far less typical of cluster 1 than point A. The resulting fit for the left cluster would thus be skewed. Fig. 8(b) presents a situation with two intersecting circular shell clusters. In this case, intuitively, one point A might be considered a "good" member of both clusters, where as point B might be considered a "poor" member, and point C an outlier. Here again, the constraint in (1) would force points A, B, and C have memberships of 0.5 in both clusters. The membership values cannot distinguish between a moderately a typical member and an extremely atypical member, because the membership of a point in a class is a relative number. In otherwords, the memberships that result from the constraint in (1) denote degrees of sharing rather than degrees of typicality.



(a) (b)

Fig. 8. Disadvantages of constrained fuzzy memberships. (a) A data set with two linear clusters. Membership values in cluster 1 for points $A$and B will be different, even though they are equidistant from the prototype. Point A and point B may have equal membership values in cluster 1, even though point C is far less typical of cluster 1 than point A. (b) A data set with two intersecting circular shell clusters. The memberships of points A, B, and C are all 0.5 in the two clusters, even though point B is less typical of the clusters, and point c is an outlier.



(a) (b)

Fig. 9. The effect of noise points on shell clustering. Prototypes for two noisy ellipses found by (a) the Hard C Quadric Shells algorithm and (b) the Fuzzy C Quadric Shells algorithm.

Therefore, noise points, which are often quite distant from the primary clusters, can drastically influence the estimates of the class prototypes, and hence the final partition.

One established technique for reducing the effect of noise on least-squares fits is to use weights that are inversely related to the distance of the point to the prototype [24], [37], [48]. Although one may interpret memberships as weights, the memberships generated by fuzzy clustering are not inversely related to the distance, since they are relative numbers. Other ways to deal with noise in clustering maybe found in [29], [47]. Davé also discusses another effective way to deal with the noise problem (although not the relative membership problem) by introducing the concept of a noise cluster [14], [15].

If there is only one cluster in the data, there is no difference between crisp and fuzzy clustering. Otherwise, the effect of noise points on the final partition is more drastic in the crisp case, because the membership of noise points tends to be distributed among all classes in the fuzzy case. Since the sum of the memberships of a point is constrained to be equal to one, however, this difficulty still remains with a lesser degree in the fuzzy case. Fig. 9 shows the effect of noise on a data set containing two elliptic clusters. As can be seen, the result of the fuzzy case (Fig. 9(b)) is poor but better than that of the hardcase (Fig. 9(a)).

One can cast the clustering problem into the framework of possibility theory [18], [49] by relaxing the constraint in (1) and reformulating the objective function in (2) as [34]

$$\mathbf{J}_m(\mathbf{B},\mathbf{U};\mathbf{X}) = \sum_{i=1}^{C}\sum_{j=1}^{N}(u_{ij})^m d^2(\mathbf{x}_j,\beta_i) + \sum_{i=1}^{C}\eta_i\sum_{j=1}^{N}(1-u_{ij})^m \tag{28}$$

where η_i are suitable positive numbers. The first term in (28) requires that the distance from the feature vectors to the prototypes be as low as possible, whereas the second term forces u_{ij} to be as large as possible, thus avoiding the trivial solution. It is easy to show [34] that $\mathbf{U}$ may be a global minimum of $\mathbf{J}_m(\mathbf{B},\mathbf{U};\mathbf{X})$ only if the memberships are updated by

$$u_{ij} = \frac{1}{1+\left(\frac{d^2(\mathbf{x}_j,\beta_i)}{\eta_i}\right)^{\frac{1}{m-1}}}. \tag{29}$$

Thus, the updated value of u_{ij} depends only on the distance of $\mathbf{x}_j$ from β_i, and not on the distance of $\mathbf{x}_j$ from all other prototypes, which is a desirable result. The prototypes are updated in the same manner as in the corresponding fuzzy algorithms.

It is best to initialize the possibilistic algorithms with the corresponding fuzzy algorithms. Since the value of η_i determines the distance at which the membership value of a point in a cluster becomes 0.5, it should relate to the overall size and shape of cluster β_i. When the nature of the clusters is known, values for η_i may also be fixed ***apriori***. For example, in the case of shell clustering algorithms, the values for η_i may be set equal to the square of the expected thickness of the shells. Another possibility is to use the fuzzy intra-cluster distance to compute η_i, i. e.,

$$\eta_i = \frac{K}{N_i}\sum_{j=1}^{N}(u_{ij})^m d^2(\mathbf{x}_j,\beta_i). \tag{30}$$

Typically K is chosen to be one. Several other ways to estimate the η_i are given in [34].

Although the fits themselves are not very sensitive to the exact values of η_i, some of the shell cluster validity measures (to be discussed in Part II of this paper) do depend on the choice of η_i. For this reason, the best approach is to compute approximate values for the η_i from the initial fuzzy partition using (30), and after the possibilistic algorithm converges, run a few more iterations of the algorithm with a fixed value of η_i. This will ensure that the validities of all the clusters are measured on a uniform scale. The fixed value of η_i is the expected thickness of the shell clusters. A good value for η_i in boundary detection applications is about two. The possibilistic shell clustering algorithm is summarized below.

THE POSSIBILISTIC SHELL CLUSTERING ALGORITHM

Fix the number of clusters C; fix m, $m \in [1,\infty)$;
Initialize C-partition $\mathbf{U}$ using the corresponding fuzzy algorithm;

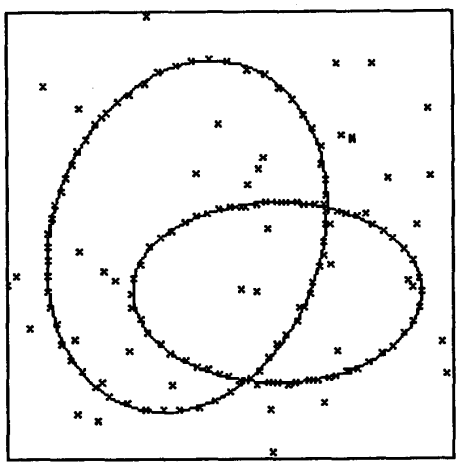


Fig. 10. Advantage of unconstrained memberships in the possibilistic approach. Prototypes for two noisy ellipses found by the PCQS algorithm.

Estimate η_i *using* (30);
REPEAT
Update the prototypes using **U**;
Compute **U** *using* (29);
UNTIL $(\|\Delta U\| < \varepsilon_1)$;
{The remaining part of the algorithm is optional and is to be used only when validity measures need to be computed}
Fix the values of η_i *to the expected thickness of the shells*;
REPEAT
Update the prototypes using **U**;
Compute **U** *using* (29);
UNTIL $(\|\Delta \mathbf{U}\| < \varepsilon_2)$;

Possibilistic versions of the FCQS and FCPQS algorithms can be derived very easily by merely changing the membership updating equation from (3) to (29). We will hence forth refer to these possibilistic versions as the Possibilistic C Quadric Shells (PCQS), and the Possibilistic C Plano-Quadric Shells (PCPQS) algorithms, respectively. Fig. 10 shows the result of the PCQS algorithm on the noisy data set in Fig. 9 for which both hard and fuzzy clustering failed. This result shows that the possibilistic approach makes the clustering process robust.

X. Conclusions and Recommendations

In this paper, we presented several fuzzy and possibilistic shell clustering algorithms. The FCQS algorithm uses a constraint on the second-degree terms and does not allow the degenerate case of linear prototypes. This problem can be overcome by using a procedure to extract linear clusters from certain types of pathological clusters. Although the constraint used in the FCPQS algorithm theoretically allows the detection of linear clusters, in practice it often overfits second-degree curves to linear segments. Also, this algorithm requires us to solve a six-dimensional generalized eigenvector problem (in the 2-D case), compared to the three-dimensional regular eigenvector problem in the FCQS algorithm. Therefore, in our experience, the (modified) FCQS algorithm and its possibilistic version are the best choices for most applications in 2-D. The reweight procedure presented in Section VII is particularly important in the 3-D case, and since the computation of the exact distance is too expensive, the FCPQS algorithm and its possibilistic version are the only viable algorithms in this case.

The existing fuzzy clustering methods use relative memberships, which cannot always distinguish between good members and poor members. On the other hand, if one takes the view that the membership of a point in a class has nothing to do with its membership in other classes, then one can achieve membership distributions that correspond more closely to the notion of typicality. The resulting possibilistic algorithms are naturally more immune to noise. One disadvantage of the possibilistic approach is that one needs to estimate the bandwidths η_i. In most practical applications of shell clustering, the expected thickness of the shell clusters is known, and this is not a major drawback. It is also to be remembered that the possibilistic algorithms need a good initialization. Thus, the fuzzy algorithms will always be useful.

Traditionally, the generalized Hough transform (GHT) [3], [26], [38] has been used to detect shapes when the boundaries/surfaces are noisy or sparse. One disadvantage of the GHT is that its computational complexity is $O(N \times N_{p1} \times N_{p2} \cdots \times N_{p_{s-1}})$, where N is the total number of points in the image to be processed, N_{p_i} is the number of quantization levels of the i-th parameter, and s is the total number of parameters. The memory requirement of the GHT is $O(N_{p1} \times N_{p2} \cdots \times N_{p_s})$. Since the accuracy of the parameter values is determined by the number of quantization levels, N_{p_i} cannot be too small. (In contrast, the accuracy of the parameter values in shell clustering is limited only by computer precision.) Some researchers have used hierarchical resolutions to mitigate this problem [43]. In the case of a general second-degree curve in 2-D, we need five parameters to describe the curve. The speed of the GHT can be improved only if we make certain assumptions about the curve, (i.e., if the curve is circular, elliptic etc.), and if the gradient information is available [8], [16], [26], [38]. Also, in spite of recent advances [28], [44], if the edge points are somewhat scattered around the ideal curve (or surface), then peak detection is very difficult in multidimensional Hough space due to bin splitting. Moreover, the detection of small segments is virtually impossible, since small peaks in the GHT are lost in the bias. The GHT also suffers from a high probability of spurious peaks [22]. Most importantly, peaks in the GHT correspond to "majority fits" and not "best fits."

The computational complexity of all the algorithms presented in this paper is $O(NCK)$, where N is the number of points, C is the number of clusters, and K is the number of iterations. If we have a good initialization procedure, the number of iterations K can be kept low. This compares very favorably with the complexity of the GHT. The memory requirements of these algorithms, which is $O(NC)$ is very low compared to those of the GHT. A more thorough comparison with the GHT is possible only for specific types of curves. An excellent comparison of the shell clustering approach with GHT for the case of circles and ellipses may be found in [16]. In [17] Davé and Fu also show how the GHT with a crude discretization of parameter space can provide

a good initialization for the shell clustering algorithms, thus significantly reducing the computational burden. This is a fuzzy generalization of the method suggested by O'Gorman and Clowes [40], who obtain a crude segmentation of the data set using the HT and then fit lines to obtain more accurate results. This approach is particularly viable when the parameter space is of low dimensionality.

Finally, we would like to note that to our knowledge, no general proof of convergence has been presented for any of the shell clustering algorithms, although in practice these algorithms always seem to converge. This is an important topic that needs to be researched in the future.

Appendix A
Derivation of the Geometric Distance from a Point to a Quadric

In this appendix we derive the closed form solution of the exact distance from a point $\mathbf{x}_j$ to the curve β_i in the 2-D case. We first note that in (12), $\mathbf{A}_i, \mathbf{b}_i$ and c_i are given by

$$\mathbf{A}_i = \begin{bmatrix} p_{i1} & p_{i(n+1)}/2 & \cdot & \cdot & \cdot & p_{i(2n-1)}/2 \\ p_{i(n+1)}/2 & p_{i2} & \cdot & \cdot & \cdot & \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & p_{ir}/2 \\ p_{i(2n-1)}/2 & \cdot & \cdot & \cdot & p_{ir}/2 & p_{in} \end{bmatrix},$$

$$\mathbf{b}_i = \begin{bmatrix} p_{i(r+1)} \\ \cdot \\ \cdot \\ \cdot \\ p_{i(r+n)} \end{bmatrix}, \quad \text{and} \quad c_i = p_{is}.$$

We first rotate the cluster prototype β_i and the point $\mathbf{x}_j$ so that the matrix $\mathbf{A}_i$ becomes diagonal. This does not change the distance. The angle of rotation in the 2-D case is given by

$$\alpha_i = \frac{1}{2}\tan^{-1}\left(\frac{p_{i3}}{p_{i1} - p_{i2}}\right). \tag{A1}$$

Equations (12) and (13) can now be written as

$$\mathbf{z}'^T\mathbf{A}_i'\mathbf{z}' + \mathbf{z}'^T\mathbf{b}_i' + c_i' = 0 \tag{A2}$$

$$\mathbf{z}' = \frac{1}{2}(\mathbf{I} - \lambda\mathbf{A}_i')^{-1}(\lambda\mathbf{b}_i' + 2\mathbf{x}_j'). \tag{A3}$$

where $\mathbf{x_j}'$ and $\mathbf{z}'$ denote the locations of points $\mathbf{x}_j$ and $\mathbf{z}$ after rotation. It is easily verified that

$$\mathbf{x}_j = \mathbf{R}_i\mathbf{x}_j' \quad \text{and} \quad \mathbf{z} = \mathbf{R}_i\mathbf{z}'$$

$$\mathbf{A}_i' = \mathbf{R}_i^T\mathbf{A}_i\mathbf{R}_i = \begin{bmatrix} p_{i1}' & 0 \\ 0 & p_{i2}' \end{bmatrix},$$

$$\mathbf{b}_i' = \mathbf{R}_i\mathbf{b}_i = \begin{bmatrix} p_{i4}' \\ p_{i5}' \end{bmatrix} \quad \text{and} \quad c_i' = c_i, \tag{A4}$$

where

$$\mathbf{R}_i = \begin{bmatrix} \cos\alpha_i & \sin\alpha_i \\ -\sin\alpha_i & \cos\alpha_i \end{bmatrix}.$$

Since $\mathbf{A}_i'$ is a diagonal matrix, $(\mathbf{I} - \lambda\,\mathbf{A}_i')^{-1}$ can be easily inverted, and from (A3) we obtain

$$\mathbf{z}'^T = \left[\frac{\lambda p_{i4}' + 2x_{j1}'}{2(1 - \lambda p_{i1}')}, \frac{\lambda p_{i5}' + 2x_{j2}'}{2(1 - \lambda p_{i2}')}\right]. \tag{A4}$$

Substituting (A4) into (A2) yields the following quartic equation in λ

$$C_4\lambda^4 + C_3\lambda^3 + C_2\lambda^2 + C_1\lambda^1 + C_0 = 0 \tag{A5}$$

where

$$\begin{aligned} C_4 &= 4p_{i1}'p_{i2}'(4p_{i1}'p_{i2}'p_{i6}' - p_{i2}'p_{i4}'^2 - p_{i1}'p_{i5}'^2) \\ C_3 &= 8p_{i1}'p_{i2}'(p_{i4}'^2 + p_{i5}'^2) + 8(p_{i1}'^2p_{i5}'^2 + p_{i2}'^2p_{i4}'^2) \\ &\quad - 32p_{i1}'p_{i2}'p_{i6}'(p_{i1}' + p_{i2}') \\ C_2 &= 16p_{i1}'p_{i2}'(p_{i2}'x_{j1}^2 + p_{i1}'x_{j2}^2) - 4p_{i1}'p_{i4}'^2 \\ &\quad - 4p_{i2}'p_{i5}'^2 + 16p_{i2}'p_{i4}'(p_{i2}'x_{j1} - p_{i4}') \\ &\quad + 16p_{i1}'p_{i5}'(p_{i2}'x_{j2} - p_{i5}') \\ &\quad + 16p_{i6}'(p_{i1}'^2 + p_{i2}'^2 + 4p_{i1}'p_{i2}') \\ C_1 &= -32p_{i1}'p_{i2}'(x_{j1}^2 + x_{j2}^2) + 8(p_{i4}'^2 + p_{i5}'^2) \\ &\quad - 32(p_{i2}'p_{i4}'x_{j1} + p_{i1}'p_{i5}'x_{j2}) - 32p_{i6}'(p_{i1}' + p_{i2}') \\ C_0 &= 16(p_{i1}'x_{j1}^2 + p_{i2}'x_{j2}^2 + p_{i4}'x_{j1} + p_{i5}'x_{j2} + p_{i6}'). \end{aligned}$$

For each real root of (A5), we calculate the corresponding $\mathbf{z}_k$ using (A4). Finally we compute d_{Pij}^2 using (14).

Appendix B
Summary of Second Degree Curve and Surface Types

A. Two-Dimensional Case

The nature of the graph of the general quadratic equation in x_1 and x_2 given by

$$p_1x_1^2 + p_2x_2^2 + p_3x_1x_2 + p_4x_1 + p_5x_2 + p_6 = 0$$

is described in Table I in terms of the values of

$$\Delta = \begin{vmatrix} p_1 & \frac{1}{2}p_3 & \frac{1}{2}p_4 \\ \frac{1}{2}p_3 & p_2 & \frac{1}{2}p_5 \\ \frac{1}{2}p_4 & \frac{1}{2}p_5 & p_6 \end{vmatrix}, \quad J = \begin{vmatrix} p_1 & \frac{p_3}{2} \\ \frac{p_3}{2} & p_2 \end{vmatrix}$$

$$I = p_1 + p_2, \quad \text{and} \quad K = \begin{vmatrix} p_{i1} & \frac{1}{2}p_{i4} \\ \frac{1}{2}p_{i4} & p_{i6} \end{vmatrix} + \begin{vmatrix} p_{i2} & \frac{1}{2}p_{i5} \\ \frac{1}{2}p_{i5} & p_{i6} \end{vmatrix}.$$

B. Three-Dimensional Case

The nature of the graph of the general quadratic equation in x_1, x_2, and x_3 given by

$$\begin{aligned} &p_1x_1^2 + p_2x_2^2 + p_3x_3^2 + p_4x_1x_2 + p_5x_1x_3 \\ &\quad + p_6x_2x_3 + p_7x_1 + p_8x_2 + p_9x_3 + p_{10} = 0 \end{aligned}$$

is described in Table II.

TABLE I
TWO-DIMENSIONAL QUADRATIC CURVE TYPES

Case	Δ	J	Δ / I	K	Conic
1	$\neq 0$	>0	<0		real ellipse
2	$\neq 0$	>0	>0		imaginary ellipse
3	$\neq 0$	<0			hyperbola
4	$\neq 0$	0			parabola
5	0	<0			real intersecting lines
6	0	>0			conjugate complex intersecting lines
7	0	0		<0	real distinct parallel lines
8	0	0		>0	conjugate complex parallel lines
9	0	0		0	coincident lines

In Table II, the expressions for $\rho_3, \rho_4, \Delta, k_1, k_2$ and k_3 are

$$\rho_3 = \text{rank } e, \text{ with } e = \begin{bmatrix} p_1 & p_4/2 & p_5/2 \\ p_4/2 & p_2 & p_6/2 \\ p_5/2 & p_6/2 & p_3 \end{bmatrix},$$

$$\rho_4 = \text{rank } E, \text{ with } E = \begin{bmatrix} p_1 & p_4/2 & p_5/2 & p_7/2 \\ p_4/2 & p_2 & p_6/2 & p_8/2 \\ p_5/2 & p_6/2 & p_3 & p_9/2 \\ p_7/2 & P_8/2 & p_9/2 & p_{10} \end{bmatrix},$$

Δ = determinant of E, and

k_1, k_2 and k_3 are the roots of $\begin{vmatrix} p_1-x & p_4/2 & p_5/2 \\ p_4/2 & p_2-x & p_6/2 \\ p_5/2 & p_6/2 & p_3-x \end{vmatrix} = 0.$

APPENDIX C
CONVERSION OF PATHOLOGICAL PROTOTYPES TO LINES

By checking the conditions satisfied by the parameters of the prototype according to Table I, we may determine if the cluster is a pathological case (i.e., if it is a hyperbola, a pair of lines or an extremely elongated ellipse). If a cluster is a pathological case, it is converted to lines. This is done using the line detection algorithm in Section V. The initialization is carried out by determining the equations of the line(s) that might best describe the pathological clusters. Then each of the points belonging to the pathological clusters is crisply assigned to the line prototype to which it is closest. This provides a very good initial partition for the G-K algorithm. The procedures for identifying the line prototypes are described below.

A. Line Detection from a Hyperbola or a Pair of Intersecting Lines

Each hyperbola should be split into two lines using the following procedure, provided it is not a very "flat" hyperbola. The case of a very "flat" hyperbola will be discussed later.

TABLE II
THREE-DIMENSIONAL QUADRATIC SURFACE TYPES

Case	ρ_3	ρ_4	Sign of Δ	Nonzero k's same sign ?	Quadric Surface
1	3	4	-	yes	Real ellipsoid
2	3	4	+	yes	Imaginary ellipsoid
3	3	4	+	no	Hyperboloid of one sheet
4	3	4	-	no	Hyperboloid of two sheets
5	3	3		no	Real quadric cone
6	3	3		yes	Imaginary quadric cone
7	2	4	-	yes	Elliptic paraboloid
8	2	4	+	no	Hyperbolic paraboloid
9	2	3		yes	Real elliptic cylinder
10	2	3		yes	Imaginary elliptic cylinder
11	2	3		no	Hyperbolic cylinder
11	2	2		no	Real intersecting planes
13	2	2		yes	Imaginary intersecting planes
14	1	3			Parabolic cylinder
15	1	2			Real parallel planes
16	1	2			Imaginary parallel planes
17	1	1			Coincident planes

Before running the G-K algorithm, the linear prototypes are initialized to be the asymptotes of the hyperbola. Finding the equations of these asymptotes is quite simple if the matrix $\mathbf{A}_i$ is diagonalized as in Appendix A. After rotation, the equation of each hyperbola becomes

$$p'_{i1}x_1^2 + p'_{i2}x_2^2 + p'_{i4}x_1 + p'_{i5}x_2 + p'_{i6} = 0$$

where $p'_{i1}, p'_{i2}, p'_{i4}$ and p'_{i5} are given by (A4) in Appendix A. It is easy to show that the two asymptotes of the hyperbola defined by the above equation are given by

$$x_2 - c_{i1}x_1 + c_{i0} = 0, \quad \text{and} \quad x_2 + c_{i1}x_1 + c_{i2} = 0$$

where

$$ci_1 = \sqrt{-\frac{p'_{i1}}{p'_{i2}}}, \quad c_{i0} = \frac{p'_{i4}}{2p'_{i1}}\sqrt{-\frac{p'_{i1}}{p'_{i2}}} + \frac{p'_{i5}}{2p'_{i2}} \quad \text{and}$$

$$c_{i2} = -\frac{p'_{i4}}{2p'_{i1}}\sqrt{-\frac{p'_{i1}}{p'_{i2}}} + \frac{p'_{i5}}{2p'_{i2}}.$$

After the prototypes have been computed, they are rotated back to their original space.

Sometimes, the FCQS algorithms will fit a pair of intersecting lines instead of a hyperbola. When cluster β_i is a pair of intersecting lines, the above equations characterize the lines themselves instead of the asymptotes of the hyperbola, thus making the initialization even better.

B. Line Detection from a Pair of Parallel Lines or a Very Elongated Ellipse or a Very Flat Hyperbola

If cluster β_i is a pair of parallel lines, after the prototype is rotated, the two lines can be either parallel to the x_1 axis or to the x_2 axis. If the lines are parallel to the x_1 axis, then $p'_{i1} \approx p'_{i4} \approx 0$, and the equations of the two lines are given by

$$x_2 = c_{i1} \quad \text{and} \quad x_2 = c_{i2}$$

where

$$c_{i1} = \frac{p'_{i5} - \sqrt{p'^2_{i5} - 4p'_{i2}p'_{i6}}}{2p'_{i2}}, \quad \text{and}$$
$$c_{i2} = \frac{p'_{i5} + \sqrt{p'^2_{i5} - 4p'_{i2}p'_{i6}}}{2p'_{i2}}.$$

On the other hand, If the lines are parallel to the x_2 axis, then $p'_{i2} \approx p'_{i5} \approx 0$, and the equations of the two lines are given by

$$x_1 = c_{i3} \quad \text{and} \quad x_1 = c_{i4}$$

where

$$c_{i3} = \frac{p'_{i4} - \sqrt{p'^2_{i4} - 4p'_{i1}p'_{i6}}}{2p'_{i1}}, \quad \text{and}$$
$$c_{i4} = \frac{p'_{i4} + \sqrt{p'^2_{i4} - 4p'_{i1}p'_{i6}}}{2p'_{i1}}.$$

When the two lines are not exactly parallel, but form a small angle between them, the FCQS algorithm will sometimes fit one very elongated ellipse or a very flat hyperbola instead of a pair of parallel lines. An ellipse can be categorized as very elongated if

$$\frac{\text{Major Axis Length}}{\text{Minor Axis Length}} > C_L \tag{C1}$$

where

$$\text{Major Axis Length} = 2\frac{1}{\sqrt{p'_{i1}\left[\frac{p'^2_{i4}}{4p'_{i1}} + \frac{p'^2_{i5}}{4p'_{i2}} - p'_{i6}\right]}},$$
$$\text{Minor Axis Length} = 2\frac{1}{\sqrt{p'_{i2}\left[\frac{p'^2_{i4}}{4p'_{i1}} + \frac{p'^2_{i5}}{4p'_{i2}} - p'_{i6}\right]}},$$

and C_L is chosen to be about 10. Similarly, we may also assume that a hyperbola can be classified as very flat if

$$\frac{\text{Conjugate Axis Length}}{\text{Transverse Axis Length}} > C_L. \tag{C2}$$

If the transverse axis is parallel to the x_1 axis, i.e., if

$$\frac{1}{p'_{i2}}\left[\frac{p'^2_{i4}}{4p'_{i1}} + \frac{p'^2_{i5}}{4p'_{i2}} - p'_{i6}\right] < 0 \tag{C3}$$

then

$$\text{Conjugate Axis Length} = 2\sqrt{-\frac{1}{p'_{i2}}\left[\frac{p'^2_{i4}}{4p'_{i1}} + \frac{p'^2_{i5}}{4p'_{i2}} - p'_{i6}\right]},$$

and

$$\text{Transverse Axis Length} = 2\sqrt{-\frac{1}{p'_{i1}}\left[\frac{p'^2_{i4}}{4p'_{i1}} + \frac{p'^2_{i5}}{4p'_{i2}} - p'_{i6}\right]}.$$

in this case, Condition (C2) reduces to

$$-\frac{p'_{i1}}{p'_{i2}} > C_L^2.$$

On the other hand, if the inequality in (C3) is reversed, then the above expressions for conjugate and transverse axes lengths are interchanged, and the negative sign inside the root appears in the expression for the transverse axis length. In this case, Condition (C2) reduces to

$$-\frac{p'_{i2}}{p'_{i1}} > C_L^2.$$

When one of the FCQS algorithm fits either a very elongated ellipse or a very flat hyperbola, the equations for the lines will be computed using the equations derived for parallel lines.

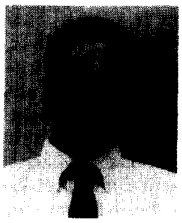
ACKNOWLEDGMENT

The authors are grateful to the anonymous reviewers for their valuable comments, which improved the presentation and contents of this paper considerably.

REFERENCES

[1] G. J. Agin, "Fitting ellipses and general second-order curves," Dept. of Comput. Sci., Carnegie Mellon Univ., Res. Rep., Jul. 1987.
[2] A. Albano, "Representation of digitized contours in terms of conic arcs and straight-line segments," *Computer Graphics Image Processing*, vol. 3, pp. 23–33, 1974.
[3] D. H. Ballard, "Generalizing the Hough transform to detect arbitrary shapes," *Pattern Recognition*, vol. 13, no. 2, pp. 111–122, 1981.
[4] J. C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms*. New York: Plenum, 1981.
[5] J. C. Bezdek and R. H. Hathaway, "Numerical convergence and interpretation of the fuzzy C-shells clustering algorithm," *IEEE Trans. Neural Networks*, vol. 3, no. 5, pp. 787–793, Sept. 1992.
[6] R. H. Biggerstaff, "Three variation in dental arch form estimated by a quadratic equation," *J. Dental Res.*, vol. 51, p. 1509, 1972.
[7] F. L. Bookstein, "Fitting conic sections to scattered data," *Computer Vision, Graphics, Image Processing*, vol. 9, pp. 56–71, 1979.
[8] D. Casasent and R. Krishnapuram, "Curved object location by Hough transformations and inversions," *Pattern Recognition*, vol. 20, no. 2, pp. 181–188, 1987.
[9] D. S. Chen, "A data-driven intermediate level feature extraction algorithm," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. PAMI-11, no. 7, pp. 749–758, July 1989.
[10] C. Coray, "Clustering algorithms with prototype selection," in *Proc. Hawaii Intern. Conf. Syst. Sci.*, Jan. 1981, pp. 945–955.
[11] D. B. Cooper and N. Yalabick, "On the computational cost of approximating and recognizing noise-perturbed straight lines and quadratic arcs in the plane," *IEEE Trans. Comput.*, vol. 25, no. 10, pp. 1020–1032, Oct. 1976.
[12] R. N. Davé, "Use of the adaptive fuzzy clustering algorithm todetect lines in digital images," in *Proc. SPIE Conf. Intell. Robots and Computer Vision*, SPIE vol. 1192, no. 2, pp. 600–611, 1989.
[13] ——, "Fuzzy shell-clustering and application to circle detection in digital images," *Int. J. Gen. Syst.*, vol. 16, pp. 343–355, 1990.
[14] ——, " Characterization and detection of noise in clustering," *Pattern Recognition Lett.*, vol. 12, no. 11, pp. 657–664, 1992.
[15] ——, "Robust fuzzy clustering algorithms," *Proc. Second IEEE Conf. Fuzzy Syst.*, San Francisco, Mar.–Apr. 1993, pp. 1281–1286.
[16] R. N. Davé and K. Bhaswan, "Adaptive fuzzy C-shells clustering and detection of ellipses," *IEEE Trans. Neural Networks*, vol. 3, no. 5, pp. 643–662, Sept. 1992.
[17] R. Davé and T. Fu, "Robust shape detection using fuzzy clustering: Practical applications," to appear in *Fuzzy Sets and Systems, Special Issue on Pattern Recognition and Computer Vision*, 1994.
[18] D. Dubois and H. Prade, *Possibility Theory: An Approach to Computerized Processing of Uncertainty*. New York: Plenum,1988.
[19] O. D. Faugeras and M. Hebert, "The representation, recognition, and positioning of 3D shapes from range data," in *Techniques for 3D*

Machine Perception, A. Rosenfeld, Ed. Amsterdam, The Netherlands: Elsevier, 1986, pp. 113–148.
[20] I. Gath and A. B. Geva, "Unsupervised optimal fuzzy clustering," *IEEE Trans. PAMI*, vol. 11, no. 7, pp. 773–781, Jul. 1989.
[21] R. Gnanadesikan, *Methods for Statistical Data Analysis of Multivariate Observations*. New York: Wiley, 1977.
[22] W. E. L. Grimson and D. P. Huttenlocher, "On the sensitivity of the Hough transform for object recognition," *IEEE Trans. PAMI*, vol. 12, no. 3, pp. 255–274, 1990.
[23] E. E. Gustafson and W. C. Kessel, "Fuzzy clustering with a fuzzy covariance matrix," in *Proc. IEEE CDC*, San Diego, CA, 1979, pp. 761–766.
[24] R. M. Haralick and L. G. Shapiro, *Computer and Robot Vision*, vol. I. Reading, MA: Addison–Wesley, 1992, Appendices.
[25] R. J. Hathaway and J. C. Bezdek, "Switching regression models and fuzzy clustering," *IEEE Trans. Fuzzy Syst.*, vol. 1, no. 3, Aug. 1993, pp. 195–204.
[26] J. Illingworth and J. Kittler, "A survey of Hough transforms," *Computer Vision, Graphics Image Processing*, vol. 44, no. 1, Oct. 1988, pp. 87–116.
[27] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*. Englewood Cliffs, NJ: Prentice-Hall, 1988.
[28] J.-M. Jolion, P. Meer, and S. Bataouche, "Robust clustering with applications in computer vision," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 13, no. 8, pp. 791–801, Aug. 1991.
[29] J.-M. Jolion and A. Rosenfeld, "Cluster detection in background noise," *Pattern Recognition*, vol. 22, no. 5, pp. 603–607, 1989.
[30] R. Krishnapuram and C.-P. Freg, "Fitting an Unknown Number of Lines and Planes to Image Data through Compatible Cluster Merging," *Pattern Recognition*, vol. 25, no. 4, 1992, pp. 385–400.
[31] R. Krishnapuram, H. Frigui, and O. Nasraoui, "New fuzzy shell clustering algorithms for boundary detection and pattern recognition," in *Proc. SPIE Conf. Robotics and Computer Vision*, Boston, Nov. 1991, SPIE vol. 1607, pp. 458–465.
[32] ——, "Quadratic shell clustering algorithms and the detection of second degree curves," *Pattern Recognition Lett.*, vol. 14, no. 7, Jul. 1993, pp. 545–552.
[33] ——, "A fuzzy clustering algorithm to detect planar and quadric shapes," in *Proc. N. Am. Fuzzy Inform. Process. Soc. Workshop*, Puerto Vallarta, Mexico, vol. I, Dec. 1992, pp. 59–68.
[34] R. Krishnapuram and J. M. Keller, "A possibilistic approach to clustering," *IEEE Transactions on Fuzzy Systems*, vol. 1, no. 2, May 1993, pp. 98–110.
[35] ——, "Fuzzy and possibilistic clustering methods for computer vision," in *Neural Fuzzy Syst.*, S. Mitra, M. Gupta, and W. Kraske, Eds., SPIE Institute Series, vol. IS–12, 1994, pp. 133–159.
[36] R. Krishnapuram, O. Nasraoui and H. Frigui, "The fuzzy C spherical shells algorithms: A new approach," *IEEE Trans. on Neural Networks*, vol. 3, no. 5, Sept. 1992, pp. 663–671.
[37] D. G. Lowe, "Fitting parametrized three-dimensional models to images," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 13, no. 5, pp. 441–450, May 1991.
[38] V. Milenkovic, "Multiple resolution search techniques for the Hough transform in high dimensional parameter spaces," in A. Rosenfeld, Ed., *Techniques for 3D Machine Perception*. Amsterdam, The Netherlands: Elsevier, 1986, pp. 231–255.
[39] J. J. Moore, "The Levenberg-Marquardt algorithm: Implementation and theory," in *Numerical Analysis*, G. A. Watson, Ed., Lecture Notes in Mathematics. Berlin: Springer–Verlag, 1977, pp. 105–116.
[40] F. O'Gorman and M. B. Clowes, "Finding picture edges through collinearity of feature points," *IEEE Trans. Comput.*, vol. 25, 1976, pp. 133–142.
[41] K. Paton, "Conic sections in chromosome analysis," *Pattern Recognition*, vol. 2, no. 1, pp. 39–51, Jan. 1970.
[42] V. Pratt, "Direct least squares fitting of algebraic surfaces," *Computer Graphics*, vol. 21, no. 4, pp. 145–152, 1987.
[43] J. Princen, J. Illingworth, and J. Kittler, "A hierarchical approach to line extraction based on the Hough transform," *Computer Vision, Graphics and Image Processing*, vol. 52, 1990, pp. 57–77.
[44] ——, "Hypothesis testing: A framework for analyzing and optimizing the Hough transform performance," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 16, no. 4, pp. 329–341, Apr. 1994.
[45] R. D. Sampson, "Fitting conic sections to 'very scattered' data: An iterative refinement of Bookstein algorithm," *Computer Vision and Image Processing*, vol. 18, pp. 97–108, 1982.
[46] G. Taubin, "Estimation of planar curves, surfaces, and nonplanar space curves defined by implicit equations with application to edge and range image segmentation," *IEEE Trans. on Pattern Anal. Machine Intell.*, vol. 13, no. 11, pp. 1115–1138, Nov. 1991.
[47] I. Weiss, "Straight line fitting in a noisy image," in *Proc. IEEE Conf. Computer Vision Pattern Recognition*, 1988, pp. 647–652.
[48] P. Whaite and F. P. Ferrie, "From uncertainty to visual exploration," *IEEE Trans. Patt. Anal. Machine Intell.*, vol. 13, no. 10, pp. 1038–1049, Oct. 1990.
[49] L. A. Zadeh, "Fuzzy sets as a basis for a theory of possibility," *Fuzzy Sets and Systems*, vol. 1, 1978, pp. 3–28.



Raghu Krishnapuram (S'83-M'84) received the B.Tech. degree in electrical engineering from the Indian Institute of Technology, Bombay, in 1978. He obtained the M.S. degree in electrical engineering from Louisiana State University, Baton Rouge, in 1985 and the Ph.D. degree in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, in 1987.

Dr. Krishnapuram was with Bush India, Bombay for a year where he participated in developing electronic audio entertainment equipment. From 1979 to 1982, he was a deputy engineer at Bharat Electronics Ltd., Bangalore, India, manufacturers of defense equipment. He is currently an Associate Professor in the Electrical and Computer Engineering Department at the University of Missouri, Columbia. In 1993, he visited the European Laboratory for Intelligent Techniques Engineering (ELITE), Aachen, Germany, as a Humboldt Fellow. His current research interests are many aspects of computer vision and pattern recognition as well as applications of fuzzy set theory and neural networks to pattern recognition and computer vision.

Hichem Frigui received the B.S. degree in electrical and computer engineering in 1990 and the M.S. degree in electrical engineering in 1992, both from the University of Missouri, Columbia.

From 1992 to 1994 he worked with IDEE, Tunis, where he participated in the development of banking software applications. He is currently pursuing the Ph.D. degree in electrical engineering at the University of Missouri, Columbia.

His current research interests include pattern recognition, computer vision, fuzzy set theory, and artificial intelligence.

Olfa Nasraoui received the B.S. degree in electrical and computer engineering, and the M.S. degree in electrical engineering, in 1990 and 1992, respectively, from the University of Missouri, Columbia.

She worked as a Software Engineer with IDEE, Tunis, from 1992 to 1994 and is currently pursuing the Ph.D. degree in electrical engineering at the University of Missouri, Columbia.

Her current research interests include pattern recognition, computer vision, neural networks, and applications of fuzzy set theory.