Acta Cybernetica 15 (2001) 151–162.

Parallelization of an algorithm for the automatic detection of deformable objects*

J.M. González-Linares†, N. Guil†, E.L. Zapata†
P.M. Ortigosa‡, and I. García‡

Abstract

This work presents the parallelization of an algorithm for the detection of deformable objects in digital images. The parallelization has been implemented with the message passing paradigm, using a master-slave model. Two versions have been developed, with synchronous and asynchronous communications.

1 Introduction

Numerous real life applications of image analysis need to detect the presence and localization of certain objects, which suffer deformations due to several factors, like sampling errors or the own flexibility of the material. Diverse methods that allow to recover these objects exist, like free-form models based on "snakes" [1] or the parametric models [2], [3]. Free-form models have the disadvantage that they cannot be used reliably in an automatic environment, and parametric models need an initial segmentation of the images.

In this work an algorithm has been selected that combines the Generalized Hough Transform (GHT, [4], [5]) with a deformation model [6], to form an objective function that is minimized by a stochastic global optimizer. The GHT provides an automatic mechanism and is highly immune to occlusions, noise and cluttering, thus solving the problems associated with other methods. The whole method can be represented by the Bayesian rule, where the prior information is formed by a template of the object to be detected and a group of deformations that are applied on the template. The likelihood is obtained from the likeness measure that provides the GHT, while the *a posteriori* information is given by the application of the Bayes rule. The inference on this bayesian model is carried out by a maximum *a*

*This work has been supported by the Ministry of Education of Spain (CICYT TIC99-0361).

†Departamento de Arquitectura de Computadores, Campus de Teatinos, Universidad de Málaga, E-29080 Málaga (Spain). E-mail: gonzalez,nico,ezapata@ac.uma.es.

‡Departamento de Arquitectura de Computadores y Electrónica, Universidad de Almería, E-04120 Almería (Spain). E-mail: pilar,inma@iron.ualm.es.

posteriori estimator (MAP), therefore an optimization algorithm named UEGO (Universal Global Evolutionary Optimizer, [7]) have been selected. This optimization algorithm allows to calculate the global maximum in an efficient way.

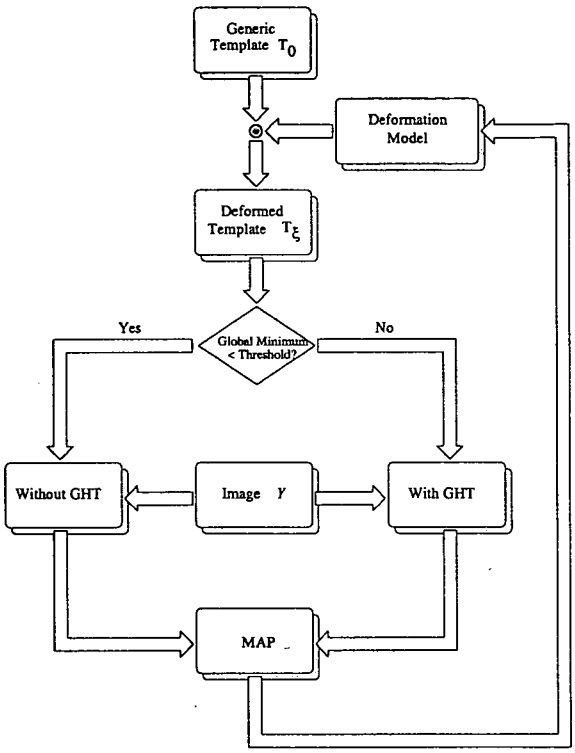

Figure 1: Outline of the method for the automatic detection of deformable objects

The computational complexity of this method is very high, since the evaluation of the GHT requires a lot of computation. The GHT allows to calculate the parameters that define the similarity Euclidean transformations (rotation, scale and displacement), but these parameters are not necessary to be computed when a good enough solution, below a given threshold, is found. To reduce the computational requirements two objective functions are used; the first one evaluates the GHT while the second not, so its computational complexity is much smaller. This second objective function can be applied when the global minimum found is below a threshold. The outline of operation of the method can be seen in Figure 1. Even with this simplification, the computational complexity remains quite high, therefore its parallelization is very interesting to obtain reasonable computational times.

Figure 2 shows one of the test images used in the evaluation of the algorithm. The top-left image presents a complex scene where the guitar is to be detected.

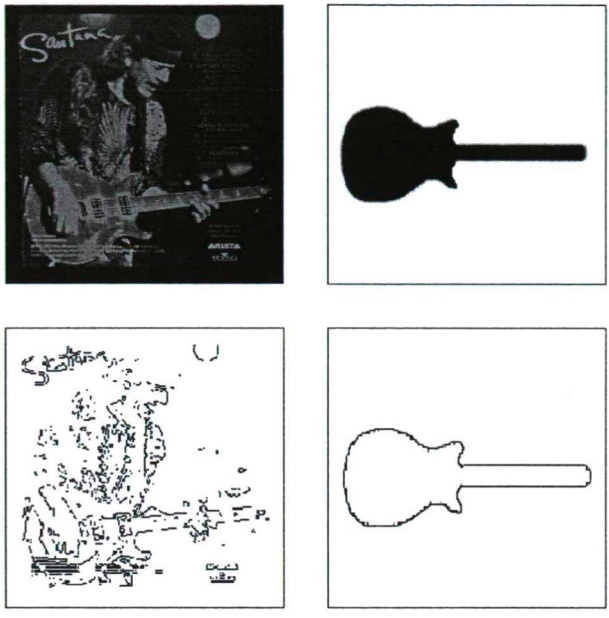

Figure 2: *Test image and contour edges*

The top-right image corresponds to the template used in the detection and the two bottom images are the contour edges obtained with the Canny detector [8]. The deformation model is applied over the template contour to obtain a deformed template T_ξ. Then, the GHT obtains the rotation, scale and displacement parameters of T_ξ in the image. After applying these parameters over the deformed template, a likeness measure can be computed:

$$\mathcal{E} = \frac{1}{n_T} \sum (1 + \Phi(x,y)|\cos(\beta(x,y))|) \tag{1}$$

where

$$\Phi(x,y) = -\exp\left(-\rho\sqrt{(\delta_x^2 + \delta_y^2)}\right), \tag{2}$$

and n_T is the number of edge points in the template, δ_x, δ_y is the distance between the contour points of the deformed template and image, $\beta(x,y)$ is the difference between their gradient angles, and ρ is a parameter to control the smoothness of the function. The value of $\mathcal{E}$ is normalized between 0 and 1, with 0 corresponding to a perfect match.

The energy value provided by (1) is used as an objective function that is minimized by UEGO. Experimental results show that near a minimum the similarity transformation parameters remain equal, thus the GHT can be skipped to speed up the process. Several intermediate results along with the final result obtained are presented in Figure 3. In the bottom-right image the final result can be seen,

Figure 3: *Example of the results obtained with the algorithm*

where the contour of the deformed, rotated, scaled and displaced template has been superimposed on the original image.

2 UEGO

UEGO is a stochastic optimization algorithm that looks for groups of solutions to optimize them by means of a particular method.

In the used implementation SASS (Single Agent Stochastic Search [9]) has been selected as local optimizer. This method presents the advantage that the objective function is not required to be continuous or differentiable. The operation consists in generating a sequence of random paths. These paths start from a search point that suffer a perturbation with a random variable. This random variable can have different probability distributions, being the Gaussian distribution the most efficient and the one that has been implemented. SASS is quite efficient to find local minima, but it presents the disadvantage of being trapped in local minima. The introduction of SASS inside an evolutionary algorithm as UEGO allows solving this problem.

The basic concept in UEGO is the one of *species* that is defined as a region (window) in the search space. Its *center* and its *radius* determine this region, where the center is a solution or point in the search space, and the radius is a positive number that indicates the width of the region. This definition assumes that the

concept *distance* exists inside the search space. UEGO algorithm creates a list of species, and each species is optimized by means of a certain method (SASS).

The role of this window is to 'localize' the optimizer that is called by a species and can 'see' only its window, so every new sample is taken from here. This means that the largest step made by the optimizer in a given species is not larger than the radius of the given species. If the value of a new solution is better than that of the old center, the new solution becomes the center and the window is moved.

The radius of a species is not arbitrary; it is taken from a list of decreasing radii, the *radius list*. The radii decrease in a regular fashion in geometrical progression. The first element of this list is always the diameter of the search space (r_1), which will ensure that the largest species always contains the whole space independently of its center. The diameter is given by the largest distance between any two possible solutions according to the distance mentioned above, and it is an input parameter. If the radius of a species is the ith element of the list, then we say that the *level* of the species is i.

During the optimization process, a list of species is kept by UEGO. The algorithm is in fact a method for managing this *species-list* (i.e. creating, deleting and optimizing species).

The creation of the list of species consists in, initially, to calculate a random point inside the search space. This point will be the center of a new species whose radius comes determined by the diameter of the search space r_1. The list of species is finite and limited by the input parameter M that indicate the maximum number of species. This initial list of species is optimized by SASS.

In the species list not only the species optimization is carried out, but also three additional types of processes ([10]). The first process is a mutation process (species creation mechanism), where a species can be divided in two or more species if more than one local minimum exists inside the species.

The second process type is one of fusion. If the centers of two species are separated by a smaller distance than the radius associated with the actual level i, the two species combine into a single one. The center of the new species will correspond to the best (minimal) center of both species, and its radius will be the biggest one.

The third process carried out is elimination. It has already been indicated that the maximum number of species is limited by a parameter. If the number of species, as a consequence of the previous processes, overcomes this parameter, the species that have been created more recently are eliminated.

All these processes, together with the optimization of the species, are carried out in an iterative way like the one indicated in Figure 4. The number of iterations l (levels) is another input parameter. Two further input parameters exist: the allowed maximum number of objective function evaluations (N) and a threshold ν that controls the maximum distance a species can travel in the level i. Although five input parameters exist, only four of them are necessary, since the fifth parameter will be obtained by means of the application of some principles. In [10], all these principles are described in detail.

```
Initialize species
Optimize species
For i=2...1,
    Create species
    Fuse species
    Eliminate species
    Optimize species
    Fuse species
```

Figure 4: *The* UEGO *code*

3 Parallelization of the method

The high computational requirements of the optimization algorithms have raised the appearance of numerous parallelization strategies. One of the most common strategies consists of carrying out global parallelization following a *master-slave* model. The *master* processor takes charge of making global decisions and distributing the populations of points that must be evaluated. On the other hand the *slave* processors evaluate the objective function in those populations of points. Another common strategy is the *coarse grain* parallelization, where the different processors execute the same algorithm of optimization on different subpopulations in an independent way. Though the executions are independent, intermediate results are sometimes exchanged. In the *fine grain* strategy the evaluations of the objective function are distributed among the processors. This strategy is quite common when working with massively parallel machines.

In this work, two parallelization strategies of UEGO based on a *master-slave* model have been evaluated. The first one (PSUEGO, [11]) presents several synchronism points so that the *slaves* processors send the evaluation of their lists of species to the *master* processor. The second version (PAUEGO) eliminates the synchronism points to avoid waiting times.

3.1 Parallel synchronous implementation (PSUEGO)

The *slaves* processors only need to receive the two own features of any species (i.e. its center and its radius), from the *master* processor, to be able to run the *optimize* and *create_species* procedures, so the amount of information involved in the communication procedures is quite small.

The *optimize* and *create_species* procedures do not need any additional information; each procedure only depends on a species and does not depend on another parameters or species. For this reason, these procedures can be run independently in several *slaves* processors at the same time.

In the initialization phase, the *master* processor forms the initial species list containing not a single point, but NP random points, where NP is the number

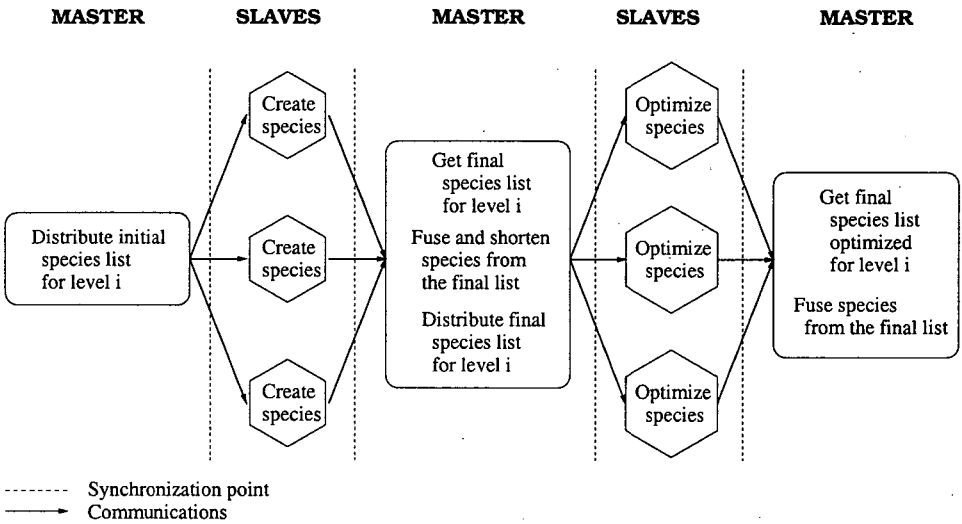

Figure 5: Synchronous version of UEGO

of *slave* processors. These NP points will be the centers of the first NP species. Then it distributes the species among the *slave* processors so all species can be optimized simultaneously. When they finish, they send the results to the *master* processor so that it forms the list of species that will be used in the following level. Then an iterative process (l levels) is carried out whose operation outline for this synchronous version can be observed in Figure 5. The oval boxes represent instructions that are executed in the *master* processor and the hexagonal boxes in the *slave* processors. The vertical dotted lines indicate synchronism points and the continuous lines indicate communications where the arrow shows the direction of the communication.

Initially, the *master* processor distributes the list of species among the *slave* processors. The *slave* processors pick up the species and evaluate them, trying to create new species. Meanwhile, the *master* processor stays in a wait state until **all** the *slave* processors finish creating species. Once the *master* processor has received all the new species, it applies the fusion and elimination processes to them in order to complete the final species list at this level. Later, it distributes this list among the *slave* processors, which take charge of optimizing each of their assigned species. When all species have been optimized, they are sent to the *master* processor that applies a fusion process and forms the species list that will be used in the following iteration.

When evaluating the objective function, the algorithm decides what objective function type must be used depending on the value of the reached minimum. If the *slave* processors act independently, there will be processors that evaluate more often the objective function with GHT than others, which will evaluate the objective

function without GHT a bigger amount of times. To avoid this computational unbalance, the *slave* processors send the rotation, scale and displacement parameters to the *master* processor, computed for their best minimum, every time they communicate with the *master* processor. The *master* processor will select the best minimum so far and, if it is below the threshold, it will communicate to all *slave* processors the parameters that should evaluate the objective function without GHT. Using this technique all the *slave* processors approximately evaluate the same number of times the objective functions with and without GHT.

Anyway, the distribution of the computational load is not well balanced. The evaluation of the objective function is carried out in the species creation and optimization processes, which are only executed in the *slave* processors. Therefore, the *master* processor is most of the time waiting results from the *slave* processors. Due to the fact that the performance of this synchronous version is very low, the elimination of some synchronism points is necessary. In this way the *master* processor can also work in the creation and optimization processes and the processors can distribute the computational load in a more dynamic way.

3.2 Parallel asynchronous implementation (PAUEGO)

The second parallel strategy (PAUEGO) is intended to solve some of the previous problems. In this new implementation the load has been balanced forcing the *master* processor to optimize and create species while the *slave* processors are working. Another change consists of the reduction of several points of synchronization. Now the *master* processor can start to carry out the synchronous operations over the species before it has received all the information, so the idle time can be reduced considerably.

The first modification that has been carried out with regard to the synchronous version is located in the initialization phase. Now every *slave* processor has a species initialization procedure; in particular, every *slave* processor chooses two or more points as centers of new species and later on, it optimizes them. Once the species is optimized, the *slave* processor creates and sends a new sublist of species for each of the optimized species to the *master* processor. On the other hand, the *master* processor only initializes a single species (not NP species like in the case of PSUEGO), and later it optimizes the species and creates new species from the optimized one. Once this new sublist has been created, the *master* processor is prepared to receive any information (sublists of species) from any *slave* processor. If at some time the *master* processor does not receive any sublist, then it begins to fuse the lists of species. This fusion process stops when the *master* processor receives any sublist from any *slave* processor, and goes on when no more reception of information from any *slave* processor is produced at that time.

Once the *master* processor has received all the sublists created by the *slave* processors and the *fuse* procedure has been applied, it starts the iterative loop. Figure 6 shows how the work is distributed among the processors. The meaning of the used symbols is the same as in Figure 5.

The iterative part of PAUEGO for the *slave* processors does not have any modifi-

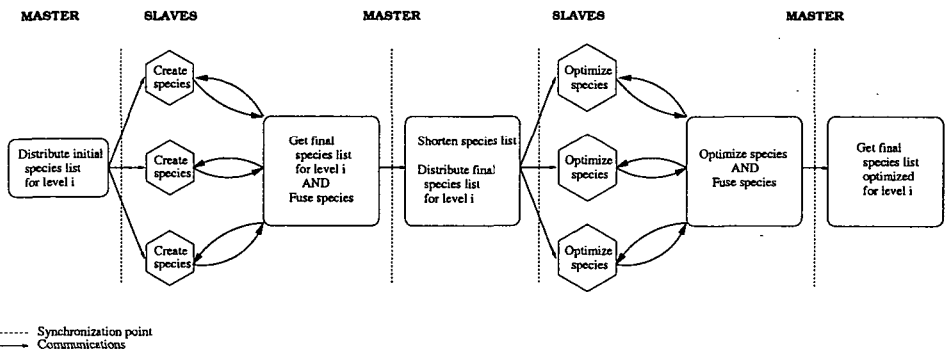

Figure 6: Asynchronous version of UEGO

cation with respect to the iterative part of PSUEGO, but there are some changes for the *master* processor. In this version, the *master* processor always checks the arrival of information (a new created sublist of species or an optimized species) from the *slave* processors and if any information has arrived it sends a new species. Otherwise the *master* processor contributes to the optimization process in such a way that when the algorithm is in the species creation stage, the *master* processor fuses the received species, and when the algorithm is in the optimization phase, then the *master* processor optimizes a species. These species must be species that have not been sent to any *slave* processor in this optimization phase. These processes executed by the *master* processor are always interrupted when new information arrives to the *master* processor, and later they are resumed.

As in the previous version, all the processors communicate the rotation, scale and displacement parameters, obtained for the best minimum, so that all processors do a similar number of evaluations of the GHT.

4 Results

The two versions of the algorithm have been tested on a SGI/CRAY T3E machine, with 32 DEC 21164 (Alpha EV-5) processors at 300 MHz and 128 MB of RAM memory each. The implementation has been carried out using the PVM message passing library [12]. To obtain the speedup, both versions have been executed 20 times on a group of five different images, being carried out a total of 100 executions. Each of these tests has been made for 2, 4, 8 and 16 processors. In addition, the sequential algorithm has been executed on the same machine to be able to compare it with the results obtained by the parallel versions. The DEC 21164 processors have two cache levels. The first level has a cache of 8 KB for the data and another of 8 KB for the instructions, and the second level has a cache of 96 KB for data and instructions. This relatively small size of the cache penalizes the performance of the algorithm in comparison with a processor like the Pentium II whose second

level cache has a size of 512 KB. The spatial locality of the data takes advantage of the cache of 512 KB. For this reason, the sequential algorithm executed on the Pentium II can be up to three or four times quicker than on the Alpha 21164.

The selected group of images presents different characteristics, in the sense that there are images where the objective function with GHT is applied few times and in others the opposite happens. This is because for some images a good optimum better than the threshold is reached very quickly and in others it is not possible to reach it. Due to these differences, the computation times are very disparate, varying from few seconds to several minutes. Nevertheless, the obtained speedup remains approximately constant independent of the computational complexity.

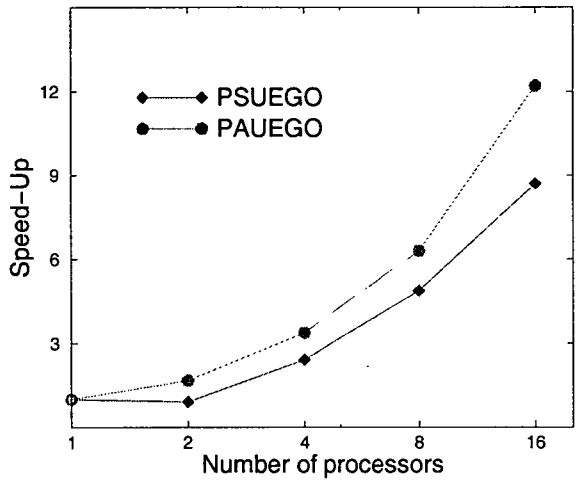

Figure 7: *Speedup for* PSUEGO *and* PAUEGO

In Figure 7 the average results obtained for both parallelizations are presented. The results with one processor correspond to the sequential version. From these results the following considerations are made:

- The distribution of the computational load is made at the species level, then it is convenient that the number of species is a multiple of the number of processors to obtain a good balance. Although the concrete number of species depends on the input data (the image and the template), the parameter M can be used to limit the number of species.

- The objective functions for this concrete application present numerous local optima [13], therefore the number of evaluations of the objective function for each species has a high variation. Typically, the total number of function evaluations is between 2000 and 3000. Then, although the distribution of species is well balanced, the computational load for each processor is not necessarily such.

- The selection of the objective function type that is evaluated depends on the best approximate global optimum. If several species are evaluated in parallel, it is possible to find global optima that are better than the given threshold before the sequential version. For this reason the average number of evaluations of the objective function with GHT in the parallel versions can be smaller to that of the sequential version.

The results obtained for PSUEGO show a low speedup because the *master* processor does not collaborate in the evaluation of the objective function. In addition, the distribution of the load is not well balanced, so the processors are waiting to each other most of the time (30%−40% of the total time). With PAUEGO the results improve because of the dynamic distribution of the load and because the *master* processor also works in the species creation and optimization processes. Although the distribution of the species is well balanced, the processors spend a 10% − 20% of the total time in waits and communications. The reason for this unbalance is that the evaluation of different species can have quite different computational complexities.

5 Conclusions

The parallelization of an algorithm for the automatic detection of deformable objects has been presented. A *master-slave* model has been selected and two versions, a synchronous one and an asynchronous one, have been implemented. The asynchronous version obtains a better speedup thanks to the dynamic distribution of the load and the participation of the *master* processor in the tasks with more computational complexity. The obtained speedup does not come closer to the ideal one due to the granularity of the problem. A distribution of the computational load with a finer grain would imply that the granularity is at the level of function evaluations instead of species. It would allow a better computational balance, but the number of communications would be much higher and the speedup would not improve.

References

[1] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models", *International Journal of Computer Vision*, 1(4), 1988, 321–331.

[2] L.H. Staib and J.S. Duncan, "Boundary finding with parametrically deformable models", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(11), 1992, 1061–1075.

[3] A.L. Yuille, P.W. Hallinan, and D.S. Cohen, "Feature extraction from faces using deformable templates", *International Journal of Computer Vision*, 8(2), 1992, 133–144.

[4] D.H. Ballard, "Generalizing the Hough Transform to Detect Arbitrary Shapes", *Pattern Recognition*, 14(2), 1981, 111–122.

[5] N. Guil, J.M. González-Linares, and E.L. Zapata, "Bidimensional shape detection using an invariant approach", *Pattern Recognition*, 32(6), 1999, 1025–1038.

[6] Y. Amit, U. Grenander, and M. Piccioni, "Structural image restoration through deformable template", *Journal of the American Statistical Association*, 86(414), 1991, 376–387.

[7] M. Jelasity, P.M. Ortigosa, and I. García, "UEGO, an Abstract Clustering Technique for Multimodal Global Optimization", Accepted for publication in the *Journal of Heuristics*.

[8] John Canny, "A computational approach to edge detection", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6), 1986, 679–698.

[9] D.C. Karnopp, "Random search techniques for optimization problems", *Automatica*, 1, 1963, 111–121.

[10] Pilar M. Ortigosa, *Stochastic Global Optimization Methods. Parallel Processing.*, PhD Thesis, University of Málaga, 1999.

[11] P.M. Ortigosa, I. García, and M. Jelasity, "Two Approaches for Parallelizing the UEGO Algorithm", Accepted for publication in *Mátraháza Optimization Days*, Kluwer Academic Publishers, 2001.

[12] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, *PVM 3 Users guide and reference manual*, Technical Report, Oak Ridge National Laboratory, 1993.

[13] J.M. González-Linares, N. Guil, E.L. Zapata, P.M. Ortigosa, and I. García, "Deformable Shapes Detection by Stochastic Optimization", in Proceedings of the *IEEE Int'l Conf. on Image Processing (ICIP'2000)*, Vancouver, Canada, September 10–13, 2000.