A Parallel Algorithm for Tracking of Segments in Noisy Edge Images

P.E. López-de-Teruel
Dpto. Ingeniería y Tecnología de Computadores

A. Ruiz
Dpto. Informática y Sistemas

J.M. García
Dpto. Ingeniería y Tecnología de Computadores

Universidad de Murcia, Campus de Espinardo, s/n
30080 Murcia (Spain)
E-mail: {pedroe,jmgarcia}@ditec.um.es, aruiz@dif.um.es

Abstract

We present a parallel implementation of a probabilistic algorithm for real time tracking of segments in noisy edge images. Given an initial solution –a set of segments that reasonably describe the input binary edge image–, the algorithm efficiently updates the parameters of these segments to track the movements of objects in the image in successive image frames. The proposed method is based on the EM algorithm –a technique for parameter estimation of statistical distributions in presence of incomplete data–, used here to estimate the parameters of a mixture density. The algorithm is highly susceptible of parallelization, because of the uncoupled nature of the computations needed on its main data structures. This property is exploited in order to make an efficient version for parallel distributed memory environments, under the message passing paradigm. We carefully describe the details of the implementation, and finally, we show an evaluation of the algorithm in a NOW (Network Of Workstations), using the standard Message Passing Interface (MPI) library. Our evaluation shows that the reached speedup is very close to the ideal optimum.

1. Introduction

Describing an edge image in terms of straight segments is a standard medium level processing technique in computer vision systems. Its main goal is to find some kind of elemental structure in the previously segmented image, as an intermediate step where further high level processing is needed, such as object recognition or extraction of 3D information.

Although finding segments is a basic problem treated in almost all generic machine vision references [2][5], most of the available methods are based in just two different approaches, namely, the *feature space transformation* methods, and the *aggregational* methods. *Feature space* techniques are based on the well-known Hough transform [4], where lines are expressed in some convenient parametric form, and a single transformation is applied to all the input edge points, each of them *voting* for an unique or a set of line candidates in the transformed parameter space. *Aggregational methods*, on the other side, try to group edge elements into extended boundaries by looking for contour lines, that are processed later to be piecewise approximated by segments [10]. In principle, both kind of techniques are thought for static images. So, to be applied in moving scenes, they have to start from scratch in each new frame.

In this paper we adopt a completely different approach to the task of segment finding, reformulating it as an statistical parameter estimation problem. The key idea is to deal with the binary input image (previously segmented in order to find edges of objects with any standard filtering technique) as a random sample of points in a bidimensional space, drawn from a probability density function that is modeled like a mixture of 'segment-like' elemental densities. Figure 1 shows the idea: The problem of segment detection is transformed into the problem of estimating the parameters of the mixture that most likely generated the corresponding input edge image. The EM algorithm [8] will be used for this task.

Our approach has several advantages over the traditional methods: First, the *dynamics of the model*: the algorithm is specially useful in the tracking of moving segments in a temporal sequence of images, using the previously obtained solution and the current input to obtain the new set of segments. Second, the ability to cope with *low-quality input edge images*, as noise points are, as we will see, explicitly contained in the probabilistic model. And finally, the susceptibility for massive *parallelization*, because of the regular nature of the computations performed by the algorithm.

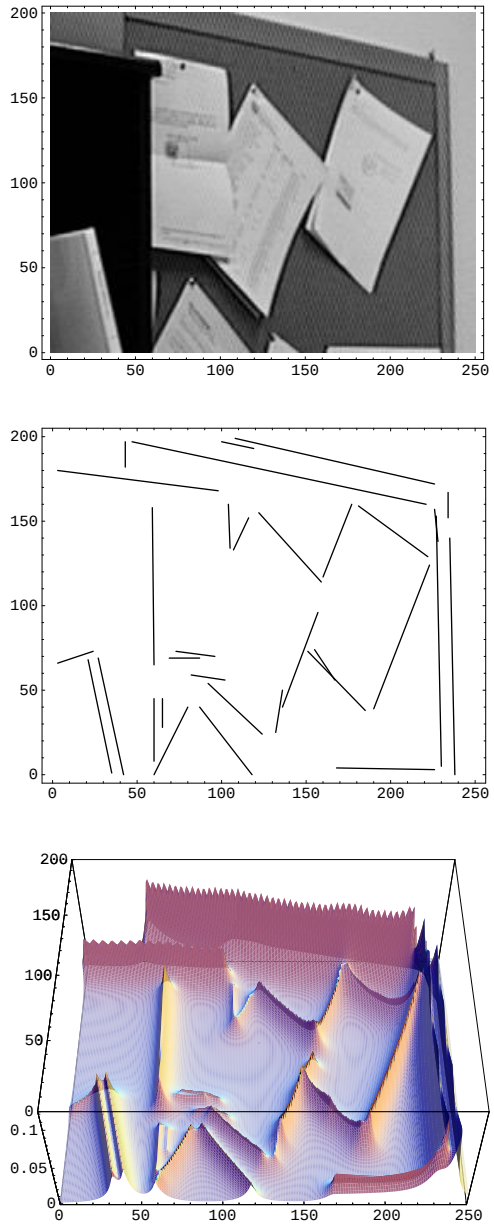

Figure 1. A sample image, a set of segments describing its edge points, and the corresponding mixture *pdf*.

2. Theoretical Foundations

As we have already stated, we use a probabilistic approach to estimate the parameters of the segments in a new image, given a prior solution and a new set of input points. The aim is to exploit the ability of EM of finding structure in data: Each point has influence in the calculation of each segment, but progressively, it is taken into account just for computing the component that most likely generated it, losing its influence on the calculus of the rest of components. In this section we describe the mathematical details of the adaptation of the EM algorithm to efficiently cope with segment tracking in noisy environments.

2.1. First Step: Weighted Assignation of Points to Segments

First of all, we need an elemental component of the mixture to model individual segments. To define the probability density function (*pdf*) of each component, we use a previous and straightforward definition of $d((x,y),S)$, the distance from a point (x,y) to a segment S: it will be the distance of the point to the nearest extreme of the segment, if the projection of (x,y) along the direction of S falls out of the segment, and simply the perpendicular distance of (x,y) to S otherwise. Using this previous definition, the following equation defines a proper *pdf* in the XY plane domain:

$$p\{(x,y)|S\} = \frac{1}{(d((x,y),S)+1)^3(\pi+L)} \qquad (1)$$

where L is the length of S. It can be shown that the above function is defined over the whole domain $\Re^2$, it integrates to one, and it is inversely proportional in each point (x,y) to the distance of the point to the segment S.

Given the previous definition, we can compute the *a posteriori* probability q_{ie} that an input point (x_e,y_e) has been generated by a segment S_i. This is analogous to the E-step (*expectation step*) of the classic EM algorithm:

$$\begin{gathered} q_{ie}^{(t+1)} = p\{S_i^{(t)}|(x_e,y_e)^{(t+1)}\} = \\ \frac{P_i^{(t)} p\{(x_e,y_e)^{(t+1)}|S_i^{(t)}\}}{\sum_{j=1}^{N} P_j^{(t)} p\{(x_e,y_e)^{(t+1)}|S_j^{(t)}\}}, \\ \forall i \in \{1,\ldots,N\}, e \in \{1,\ldots,M^{(t+1)}\} \end{gathered} \qquad (2)$$

Here, N is the number of segments, M is the number of points, and P_i is the *a priori* probability of each segment S_i. Values with superscript (t) correspond to the t^{th} image in the sequence. Note that, here, N is fixed, but $M^{(t+1)}$, the number of input points, could change in time.

2.2. Second Step: The Adjustment of Segments

Now we can recompute the new extreme points of each segment S_i using all the input points (x_e,y_e), weighted by their probabilities q_{ie} of being part of such segment. In order to do it, we first compute the new *a priori* probability P_i, mean (row) vector $\vec{\mu}_i$, and covariance 2×2 matrix Σ_i

of each component. This is analogous to the M-step (*maximization step*) of the classic EM algorithm:

$$P_i^{(t+1)} = \frac{1}{M^{(t+1)}} \sum_{e=1}^{M^{(t+1)}} q_{ie}^{(t+1)} \tag{3}$$

$$\vec{\mu}_i^{(t+1)} = \frac{1}{M^{(t+1)} P_i^{(t+1)}} \sum_{e=1}^{M^{(t+1)}} q_{ie}^{(t+1)} (x_e, y_e) \tag{4}$$

$$\Sigma_i^{(t+1)} = \frac{1}{M^{(t+1)} P_i^{(t+1)}} \cdot \sum_{e=1}^{M^{(t+1)}} q_{ie}^{(t+1)} ((x_e, y_e) - \vec{\mu}_i^{(t+1)})^T \cdot ((x_e, y_e) - \vec{\mu}_i^{(t+1)}) \tag{5}$$

$$\forall i \in \{1, \ldots, N\}$$

Now we can assume that the points in each segment present a strong correlation between x and y, as they should be straight segments. Then, we can calculate the main (i.e., major) eigenvalue, λ_{i1}, and the corresponding eigenvector $\vec{v}_{i1}$, of each covariance matrix Σ_i:

$$\lambda_{i1} = \max\{(\sigma_{11})_i + (\sigma_{22})_i \pm \sqrt{((\sigma_{11})_i + (\sigma_{22})_i)^2 - 4((\sigma_{11})_i (\sigma_{22})_i - (\sigma_{12})_i^2)}\} \tag{6}$$

$$\vec{v}_{i1} = (\frac{\lambda_{i1} - 2(\sigma_{22})_i}{2(\sigma_{12})_i}, 1) \tag{7}$$

These give us the direction of the segments. The minor eigenvalue in each component, λ_{i2}, corresponding to the other eigenvector $\vec{v}_{i2}$ (orthogonal to $\vec{v}_{i1}$), should be much smaller, and can be rejected (in fact, it would be zero in a perfect segment). Using the values of λ_{i1} and $\vec{v}_{i1}$, it is easy to compute the extreme points (x_{i1}, y_{i1}) and (x_{i2}, y_{i2}) for the segment S_i:

$$\{(x_{i1}, y_{i1}), (x_{i2}, y_{i2})\} = \mu_i \pm \sqrt{3\lambda_{i1}} \frac{\vec{v}_{i1}}{|\vec{v}_{i1}|} \tag{8}$$

In the above formulae, we assume that the projection of points into their corresponding segment are uniformly distributed along its direction $\vec{v}_{i1}$, and obviously centered in $\vec{\mu}_i$. Given that the eigenvalue λ_{i1} is exactly the variance of the sample points projected along that direction, and that the variance of an unidimensional uniform distribution in an interval of length L is $\frac{L^2}{12}$, it is straightforward to see that the length of the segment S_i is exactly $L = \sqrt{12\lambda_{i1}}$. Thus, we multiply $\frac{L}{2} = \sqrt{3\lambda_{i1}}$ by the normalized eigenvector $\frac{\vec{v}_{i1}}{|\vec{v}_{i1}|}$, and respectively subtract and add it to $\vec{\mu}_i$ in order to obtain the extremes of the segment.

2.3. Noise Treatment

Noise points are a problem in our approach. In most edge images, many spurious points appear, caused by noise in the sensor, textures, reflects, limitations of the filtering technique, and so on. If these points are not treated in a special way, they could drastically disturb the solutions, moving the segments away from their correct locations. To solve this problem, we introduce in the mixture an additional uniform bidimensional component to capture these noise points. That is, the mixture is not constituted only by the N elemental segment-like components, but also has an additional uniform component across the XY plane, with the same limits than the whole image frame. This component gives a constant *a priori* probability $p\{(x, y)|Noise\} = \alpha$ to each input point of having been produced by noise. The constant α can be modified in order to cope with different amounts of noise. The algorithm remains the same, except that now we have to compute a new $q_{N+1\,e}$ value for each point, the *a posteriori* probability that it has been generated by the uniform component. Empirical tests confirmed that the solution is satisfactory, with noise points being captured by the uniform component during the EM algorithm iteration, while aligned points were adequately tracked by the rest of segment-like components.

3. Improving the Performance: Parallelization of the Algorithm

In this section, we show how we take advantage of the symmetry in the involved computations in the EM algorithm for mixtures [7]. The first clear data parallelism can be found in the E-step: We can distribute the processing of the $p\{(x_e, y_e)|S_i\}$ and the subsequent q_{ie} values (eqs. (1) and (2)) matrix by input points, indexed by e. This distribution avoids the need for communications among processors in the normalizations needed by eq. (2). This solves the E-Step of the algorithm in an efficient way, at a minimal communication cost (only the segment parameters must be sent to all the processors; the input points can be partitioned among them).

A second source of parallelism can be exploited in the M-Step, when parameters of the mixture are recomputed using the q_{ie} matrix: Once that we have the q_{ie} values calculated, but distributed among the processors, we can compute the *intermediate parameters* P_{ip}, $\vec{\mu}_{ip}$ and Σ_{ip} for each component i on each processor p, but taking into account only the points assigned to the corresponding processor, using the eqs. (3), (4) and (5), conveniently modified as follows:

$$Parameter_{ip}^{(t+1)} = \frac{1}{\frac{M^{(t+1)}}{P} P_i^{(t+1)}} \sum_{e=(p-1)\frac{M^{(t+1)}}{P}+1}^{p\frac{M^{(t+1)}}{P}} \ldots, \tag{9}$$

$$\forall p \in \{1, \ldots, P\}$$

where p indexes the processors, from 1 to P (total number of available processors). Note that these computations

can still be performed under the initial data distribution, without the need for additional communications. But, of course, what we need are the final values of P_i, $\vec{\mu}_i$ and (σ_i). So, to complete the M-Step, we have to gather the intermediate values obtained in (9) in all the processors, and using them, compute the final values. This is the second communication point of the algorithm, after the initial distribution:

$$P_i^{(t+1)} = \frac{1}{P} \sum_{p=1}^{P} P_{ip}^{(t+1)} \tag{10}$$

$$\vec{\mu}_i^{(t+1)} = \frac{1}{PP_i^{(t+1)}} \sum_{p=1}^{P} P_{ip}^{(t+1)} \vec{\mu}_{ip}^{(t+1)} \tag{11}$$

$$\Sigma_i^{(t+1)} = \frac{1}{PP_i^{(t+1)}} \sum_{p=1}^{P} P_{ip}^{(t+1)} (\Sigma_{ip}^{(t+1)} + \vec{\mu}_{ip}^{(t+1)T} \cdot \vec{\mu}_{ip}^{(t+1)}) \tag{12}$$

$$\forall i \in \{1, \ldots, N\}$$

Eqs. (10) and (11) are fairly straightforward, while eq. (12) uses the well known property that the total covariance of a set of subsamples is the mean of the covariances plus the covariance of the means of the subsamples.

For the sake of simplicity, we have supposed that $M^{(t+1)}$ is a multiple of P. But if this condition does not hold, the only change that must be made is to weight each intermediate value with its corresponding proportion of points assigned to that processor, i.e. $\frac{M_p^{(t+1)}}{M^{(t+1)}}$, instead of the value $\frac{1}{P}$, that appears in eqs. (10), (11) and (12), and $M_p^{(t+1)}$ instead of $\frac{M^{(t+1)}}{P}$ in eq. (9)[1].

In (10), (11) and (12), a new data partition can be done to take advantage of parallelism, in this case by segments, indexed by i. Each processor is assigned a subset of segments $i = (p-1)\frac{N}{P}, \ldots p\frac{N}{P}$. We can also compute this way the extremes of the segments, using eqs. (6), (7), and (8). The final results have to be broadcasted again to all the processors, to begin with the next iteration. This is the third and last communication point of the parallel algorithm.

Figure 2 shows in pseudo-code the *message-passing* program for segment detection and tracking, in which the three communication points are emphasized. In fact, we have centered the discussion in the inner loop, corresponding to EM iteration (tracking of segments). If the noise component grows too much, then perhaps there are new objects in the scene, or some old ones have disappeared, and the current number of segments may have became inadequate. In this case, we have to reinitialize the solution with the external detection algorithm.

[1]Of course, $M_p^{(t+1)}$ is the number of input points assigned to processor p, usually $\lfloor \frac{M^{(t+1)}}{P} \rfloor + 1$ for processors $p = 1, \ldots, M^{(t+1)} \bmod P$ and $\lfloor \frac{M^{(t+1)}}{P} \rfloor$ for processor $p = M^{(t+1)} \bmod P + 1, \ldots, P$. This is the assignation of points that presents a better load balancing, assumed that the P processors are identical.

Input: *(Array of changing M sample points)*
(x_e, y_e), for $e = 1, \ldots, M$.
Output: *(Array of changing N segments)*
$((x_{i1}, y_{i1}), (x_{i2}, y_{i2}))$, for $i = 1, \ldots, N$.

Main Iteration: (*Infinite Loop*)
Repeat
Input: (*From Lower Levels of the Vision System*)
·Read current list of edge points (x_e, y_e), for $e = 1, \ldots, M$.
Initialization: (*Segment Detection*)
·Compute an initial solution $((x_{i1}, y_{i1}), (x_{i2}, y_{i2}))$, for $i = 1, \ldots, N$, with some standard (possibly parallel) algorithm (i.e. Hough).
·Assign $P_i := \frac{1}{N+1}$, for $i = 1, \ldots, N+1$.
EM Iteration: (*Segment Tracking*)
Repeat
Process 0:
Input: (*From Lower Levels of the Vision System*)
·Read current list of edge points (x_e, y_e), for $e = 1, \ldots, M$.
Data Distribution: (*First Communication Point*)
·Send points (x_e, y_e), $e = ((p-1)\frac{M}{P} + 1, \ldots, p\frac{M}{P})$ to respective processors p, for $p = 1, \ldots, P$
·Broadcast $((x_{i1}, y_{i1}), (x_{i2}, y_{i2}))$, for $i = 1, \ldots, N$, and P_i, for $i = 1, \ldots, N+1$, to all P processors.
For all processes $p = 1, \ldots, P$**:**
E step: (*Q Matrix Computation*)
·Compute $p\{(x_e, y_e)|S_i\}$, for $i = 1, \ldots, N+1$, and $e = ((p-1)\frac{M}{P} + 1, \ldots, p\frac{M}{P})$, using eqs. (1) and $p\{(x_e, y_e)|S_i\} = \alpha$ (for $i = N+1$).
·Compute q_{ie}, for $i = 1, \ldots, N+1$, and $e = ((p-1)\frac{M}{P} + 1, \ldots, p\frac{M}{P})$, using eq. (2).
M1 step: (*Intermediate Parameters Computation*)
·Compute P_{ip}, $\vec{\mu}_{ip}$, and Σ_{ip}, for $i = 1, \ldots, N$, using eqs. (3), (4) and (5), but adapted to the corresponding subset of points by eq. (9).
Intermediate Gathering: (*Second Communication Point*)
·Gather $P_{ip'}$, $\vec{\mu}_{ip'}$, and $\Sigma_{ip'}$ from the rest of processes $p' = 1, \ldots, P, p' \neq p$.
M2 step: (*Final Parameters Computation*)
·Compute P_i, $\vec{\mu}_i$, and Σ_i, for $i = (p-1)\frac{N}{P} + 1, \ldots, (p)\frac{N}{P}$, using eqs. (10), (11) and (12).
·Compute $((x_{i1}, y_{i1}), (x_{i2}, y_{i2}))$, for $i = (p-1)\frac{N}{P} + 1, \ldots, p\frac{N}{P}$, for $i = 1, \ldots, N$, using eqs. (6), (7), and (8).
Process 0:
Segments Gathering: (*Third Communication Point*)
·Gather $((x_{i1}, y_{i1}), (x_{i2}, y_{i2}))$, for $i = 1, \ldots, N$ and P_i $\forall i = 1, \ldots, N+1$.
Output: (*To Upper Levels of the Vision System*)
(…Use output in higher levels…)
Until $P_{N+1} > Threshold$.
Until FALSE

Figure 2. Parallel message passing algorithm for segment tracking (pseudo-code).

4. Evaluation

We conclude with some performance results obtained executing an MPI implementation [7] of our proposed parallel algorithm in a cluster of workstations [1]. The cluster had the following technical characteristics: Pentium MMX 200 MHz, 256 KB cache, 64 MB RAM nodes, communicated by a Fast Ethernet 3Com 100 Mbps, and MPICH 1.0.13 for Linux as the MPI library [3] implementation.

Figure 3 shows the processing times *vs* the number of processors for several problem sizes $M \times N$, after 100

iterations of E and M steps. The reduction in execution time as we increment the number of workstations participating in the computation (P) is very promising: The reached speedups are not very far from the ideal optimum P. The implementation, therefore, shows to be highly scalable, mainly due to the reductions performed in the size of communications, and the symmetric distribution of the computations (good *load balancing*). Using the 7 processors simultaneously, up to 5 frames per second were processed for images with $M = 2000$ edge points and $N = 50$ segments, typical values for usual image sizes. Of course, the higher the values of M and N, the less the frames per second that the system was capable to process, but, also, the better the reached *speedup*. This is mainly due to the fact that, as we increase the size of the problem, the increment in the quantity of parallel computations is more important than the increment in the size of communications, the traditional bottleneck in NOWs.

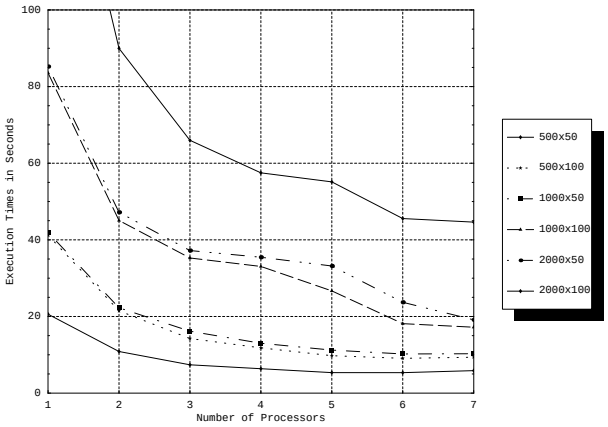

Figure 3. Performance measures for 100 EM iterations on several (N, M) input values.

5. Conclusions

In this paper, we have presented an innovative parallel algorithm to track segments in real time computer vision environments, using a probabilistic technique. We have also shown how it can be efficiently implemented in distributed memory parallel machines. The algorithm shows very good results in both speedup and scalability, as it takes advantage of a careful load balancing and minimization of communications. As a result, for small/medium image sizes, the number of frames per seconds that can be processed allows for real time application in tracking of moving scenes.

The technique is specially appropriate for clusters of PCs, where the shared communication medium does not suppose a severe bottleneck [9]. In these clusters, very good performance/cost ratios can be achieved if the parallel algorithms are specifically designed for those distributed memory environments [6]. This suggests the interesting possibility of using this kind of parallel machines in computer vision, with standard PCI image acquisition cards. In this way, we can significantly improve the overall processing speed of our computer vision system, obtaining an interesting performance/cost ratio, and real-time processing of moving scenes.

6. Acknowledgements

This work has been partially supported by the Spanish CICYT under grants TIC97-0897-C04-02 and TIC98-0559.

References

[1] T. Anderson, D. Culler, and D. Patterson. A case for NOW. *IEEE Micro*, 15(1):55–64, 1999.

[2] E. Davids. *Machine Vision, 2^{nd} Edition*. McGraw-Hill, 1997.

[3] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message Passing Interface*. MIT Press, 1994.

[4] P. Hough. Methods and means for recognising complex patterns. *U.S. Patent 3 069 654*, 1962.

[5] R. Jain, R. Kasturi, and B. Schunk. *Machine Vision*. McGraw-Hill, 1995.

[6] B. Lester. *The Art of Parallel Programming*. Prentice Hall, 1993.

[7] P. López-de-Teruel, J. García, and M. Acacio. The parallel EM algorithm and its applications in computer vision. *Proceedings of the PDPTA'99, CSREA Press*, 1999.

[8] G. McLachlan and T. Krishnan. *The EM Algorithm and Extensions*. John Wiley and Sons, 1997.

[9] J. Piernas, A. Flores, and J. García. Analyzing the performance of MPI in a cluster of workstations based on fast ethernet. *4^{th} European PVM/MPI Users' Group Meeting. Lecture Notes in Computer Science*, 1332:17–24, 1997.

[10] P. Rosin. Techniques for assesing polygonal approximation of curves. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19:659–666, 1997.