

Parallel Recognition and Classification of Objects

C.A. Rodrigo Felice¹, C.A. Fernando Ruscitti¹, Lic. Marcelo Naiouf², Eng. Armando De Giusti³

*Laboratorio de Investigación y Desarrollo en Informática⁴
Departamento de Informática - Facultad de Ciencias Exactas
Universidad Nacional de La Plata*

Abstract

The development of parallel algorithms for an automatic recognition and classification of objects from an industrial line (either production or packaging) is presented. This kind of problem introduces a temporal restriction on images processing, a parallel resolution being therefore required.

We have chosen simple objects (fruits, eggs, etc.), which are classified according to characteristics such as shape, color, size, defects (stains, loss of color), etc. By means of this classification, objects can be sent, for example, to different sectors of the line.

Algorithms parallelization on a heterogeneous computers network with a PVM (Parallel Virtual Machine) support is studied in this paper.

Finally, some quantitative results obtained from the application of the algorithm on a representative sample of real images are presented.

¹ Computer Sciences Graduate. Department of Computer Sciences, Faculty of Exact Sciences, UNLP.

² Full-time Co-Chair Professor. LIDI. Department of Computer Sciences, Faculty of Exact Sciences, UNLP. E-mail: mnaiouf@info.unlp.edu.ar

³ Principal Researcher of the CONICET. Full-time Chair Professor. Department of Computer Sciences, Faculty of Exact Sciences, UNLP. E-mail: degiusti@info.unlp.edu.ar

⁴ Calle 50 y 115 - 1er piso - (1900) La Plata - Buenos Aires
Tel/Fax: +54 21 227707
E-mail: lidi@info.unlp.edu.ar

1. Introduction

There is an increasing interest on handling and decision-making processes automation, which have so far been almost exclusively relied on human expertise. A very wide range of these activities is based on visual perceptions of the world surrounding us; therefore images acquisition and processing are a necessary requirement for automation, and there is a close relation to the study area Computer Vision [Har92] [Jai95].

When classification processes are not automated they are carried out by means of different methods:

- *Mechanical*: used only when the objects are geometrically regular. Also, these methods may not be very accurate, and they only allow size-based differentiation.
- *Electronic*: oriented to color detection, and application-specific. They use photocells which send information to programmable logical controllers, which in turn process them.
- *Manual*: trained staff carries out the classification in a visual way, therefore human limitations influence the results: they are not accurate, they are training-dependent, and subject to personal criteria. In addition to this, routine tasks are difficult to be kept constant during long periods of time, and efficiency is reduced with time. In order to solve this, staff has to be changed at short periods of time, costs being thus incremented.

Computer vision systems attempt to imitate the human visual process in order to automate highly difficult tasks as regards accuracy and time, or highly routinary ones. It is currently widely used, since the technological advance in the area results in the automation of processes so far performed using some of the previously mentioned methods.

The purpose of a computer vision system is the creation of a model of the real world from images: to recover useful information about a scene from its 2D projections. There are techniques developed in other areas which are used to recover information from images. Some of these techniques are Images Processing (generally at the early stages of a computer vision system, where it is used to improve specific information and eliminate noise) [Gon92] [Bax94] [Hus91] [Jai89], Computer Graphics (where images are generated from geometric primitives), Pattern Recognition (which classifies symbolic and numerical data) [Sch92], etc.

A particular kind of problems of interest is the process of recognition and classification of objects coming from an industrial line (production or packaging) [Law92]. The incorporation of computers to this kind of problem tends to increase both quantity and quality of the obtained products and at the same time it reduces expenses. These systems carry out simple tasks such as the processing of images obtained under controlled lighting conditions and from a fixed point of view. In addition to this they must operate in real time, which means that specialized hardware and solid algorithms must be used in order to reduce the possibility of failures, and they must be flexible so that they can be speedily adapted to changes in the production process and to inspection requirements [Lev90]. This temporal restriction naturally leads to a parallel resolution of the problem in order to properly fulfill the requirements.

For this study, a transporting belt over which objects are transported was taken as the classification model. Somewhere in the run there is a video camera taking images of the objects which are in turn received by a computer. From these images, the application classifies the object and makes decisions, which eventually are translated into signals which send the object to different sectors according to its characteristics (quality, color, size, etc.), or discard it.

The decision-making process always requires that the objective be known: decisions must be made at each stage of a vision system. Emphasis is made on automatic operation maximization of each stage, and systems need use (implicit and explicit) knowledge to do this. System efficacy and efficiency are usually governed by the employee's knowledge quality.

A computational solution would not present the problems of the mechanical, electronic or manual methods; since it allows the analysis of a wide range of characteristics, is easily adaptable to different types of objects and does not have human biologic limitations. However, the existing object-recognition applications have performance limitations restricting their

application to very specific cases where answering time is not determining, or to research projects where there are specialized computers with a great computational speed available.

In an attempt to overcome these limitations, this study has as its main objective the development of parallel recognition and classification algorithms, which will allow to obtain results in acceptable times using a heterogeneous processors network.

2. Computer vision

Computer vision develops basic algorithms and theories by means of which useful information about the world can be automatically extracted and analyzed from images. For a better understanding of the problems and goals constituting the purpose of images processing, it is necessary to comprehend the complexity of our own human vision system.

Even though visual perception, objects recognition, and the analysis of the collected information seems to be an instantaneous process for the human being, who does it in a "natural" way, our visual system is not yet fully understood. It involves physical sensors which get the light reflected by the object (creating the original image), filters which clean the image (perfecting it), sensors which get and analyze the particular details of the image, so that, after its recognition and classification, decisions can be made according to the gathered information. In these terms, the complexity of the apparently simple process of seeing, which includes a numerous amount of subprocesses, all of them carried out in a very short time, can be appreciated.

The purpose and logical structure of a computerized vision system is essentially the same as a human one. From an image caught by a sensor, all the necessary analyses and processes are carried out in order to recognize the image and the objects forming it. There are several considerations to be made when designing a vision system: What kind of information is to be extracted from the image? Which is the structure of this information in the image? Which "a priori" knowledge is needed to extract this information? Which kinds of computational processes are required? Which are the required structures for data representation and knowledge?

In the required processing there are four main aspects:

- Pre-processing of the image data
- Detection of objects characteristics
- Transformation of iconic data into symbolic data
- Scene interpretation

Each of these tasks requires different data representations, as well as different computational requirements. Thus the following question arises: Which architecture is needed to carry out the operations?

Many architectures have pipeline-developed computers, which provides a limited operations concurrence at a good data transference rate (throughput). Others present a connected mesh of processors because their images mapping is efficient, or they increase the mesh architecture with trees or pyramids because they provide the operations hierarchy which is thought to involve the biologic vision system. Some architectures are developed with more general parallel computers based on shared memory or connected hypercubes.

The problems presented in a vision system occur because the observation units are not analysis units. The unit of an observed digital image is the pixel, which presents value and position properties, but the fact that both position and value of a pixel are known does not give information about an object recognition, its shape, orientation, length of any distance in the object, its defects degree, or information about the pixels which form the object under study.

Objects recognition

The process allowing to classify the objects of an image is called computational inspection and recognition, and it involves a variety of stages which successively transform iconic data in order to recognize information. Objects detection in real time, with only the processed information from the image available, is a difficult task, since recognition and classification capacities depend on numerous factors. Among them we can mention: similarity between objects from the analyzed set and those to be detected, different possible angles used for objects visualization, type of sensors involved, amount of distortion and noise, algorithms and processes used, architecture of the used computers, real time specific restrictions, confidentiality levels required to make the final decision

An "ideal" objects recognition system may suppose that the set of objects is observed by a perfect sensor, with no noise or distortion transmission. A more realistic approach would consider a different situation. The image is created by a mathematical process applied to the sensed objects, and is then cleaned or restored in order to classify the features which will be needed for future processes. Segmentation divides it into mutually exclusive pieces to facilitate the recognition process. Useful data are selected, and relevant aspects given by the sensors are registered (altitude, speed, etc.) in order to carry out the classification process. Segmented objects are checked against a pre-determined class in order to know whether the objects being looked for are present or not, the detection step being thus completed.

The more widely known recognition methodologies generally include the following stages:

1. *Image formation*: it is the digital representation of the data captured by the sensor. It includes sampling determination and gray levels quantization, which define the image resolution.

2. *Conditioning*: the original image is enhanced and restored in order to obtain a "better" one, "better" meaning whatever the observer considers as quality in an image. Contrast enhancement, smoothing and sharpening techniques can be used. Restoration consists of the reconstruction of a degraded image, with an "a priori" knowledge of the cause of the degradation (movement, focus, noise, etc.).

3. *Events labeling*: it is based on the model defining the informational pattern with an events arrangement structure (or regions), where each event is a connected set of pixels. Labeling determines to which class of event each pixel belongs, labeling pixels belonging to the same event with identical labels. Examples of labeling are thresholding (with high or low pixel values), borders detection, corners search, etc.

4. *Events grouping*: it identifies the events by means of a simultaneous collection or identification of very connected pixel sets belonging to the same type of event. It may be defined as an operation of connected components (if labels are symbolic), a segmentation operation (if labels are in gray levels), or a borders linking (if labels are border steps). After this stage, the entities of interest are no longer pixels but sets of pixels.

5. *Properties extraction*: it assigns new properties to the set of entities generated by the grouping stage. Some of them are center, area, orientation and moments. It can also determine relations between pixel groups, as for example if they touch each other, if they are closed, if one group is inside another, and so on.

6. *Matching*: the image events or regions are identified and measured by means of the previous processes, but in order to assign a meaning to them it is necessary to carry out a conceptual organization which allows to identify a specific set as an imagined instance of some previously known object. The matching process determines the interpretation of some related event sets, by associating these events with some object. Simple figures will correspond to primitive events, and measure properties from a primitive event will frequently be suitable for the figure recognition. Complex figures will correspond to sets of primitive events. In this case, a properties vector of each observed event and the relation between them will be used.

Except for image formation, the rest of the stages can be applied several times at different processing levels.

The management of non-restricted environments is one of the current difficulties, since the existing recognition and computer vision algorithms are specialized and only perform some of the necessary transformation steps.

3. Problem description and sequential solution

Initially, egg-transporting belts were taken as a model of industrial processes, since the regular characteristics of these objects facilitates the classification process. Then more complex cases, such as fruits, bottles, ceramics, etc., were considered. The application classifies the objects and makes decisions which may be translated as signals, which send the objects to different sectors of the production line according to their characteristics, or discard them. For the classification process, different characteristics are considered, such as area, color, maximum diameter, perimeter, centroid, and the presence of defects on the surface (stains, sectors with loss of color, etc.).

As already mentioned, the goal is the automation of objects classification by using distributed algorithms in heterogeneous computers networks. For this purpose, the sequential resolution of the problem is first described, and then its parallelization is considered.

In order to focus on the development of distributed recognition algorithms, bidimensional true color images of the objects were used, their acquisition by means of a video camera being left for a future extension. In order to obtain the images used, hen eggs were photographed on a dark background with natural light. These photographs were then scanned using a Genius 9000 scanner with a resolution of 300 dpi.

When the first tests were run using this set of photographs, it was observed that natural light caused reflections and shadows on the surface of the objects, thus preventing a precise determination of their color. On these basis, a second set of photographs was taken, using artificial light from different angles: this set of photographs was significantly better. Fruit (lemons, oranges, and tomatoes) and ceramics (mosaics and tiles) samples were also obtained. Finally, scanned images were stored as JPEG files to reduce storage space. The implemented algorithms will now be described.

3.1. Threshold

Threshold (a labeling operation on a gray-leveled or colored image) was the first algorithm implemented, and it was used to differentiate the background of the image from the objects in it. It distinguishes between pixels with a high gray or color value from pixels with a low gray or color value. It is one of the most widely used methods to extract a shape or a particular feature of interest from an image. The threshold binary operator produces a black and white image.

It is very important for this operation that the threshold value be properly determined, otherwise the following stages may fail. There are various methods to automatically choose the ideal threshold value, one of them is based on the image histogram. To find a value which will separate in an optimal way the dark background from the shiny object (or vice versa), it would be desirable to know dark and shiny pixels distribution. Thus, the separating value could be determined as the value where the probability of assigning the wrong label to a background pixel is equal to the probability of assigning the wrong label to an object pixel. The difficulty here lies in the fact that the independent distribution of dark and shiny pixels is generally not known at the beginning of the process.

The algorithm simply runs over each pixel of the image and replaces it by the value 255 (white) when its value is higher than the pre-established threshold value (VDT), or by the value 0 (black) when it is not. It should be noted that the algorithm is VDT dependent. According to the results obtained with the first processed images, the ideal VDT was noticed to depend on

lighting and background conditions; therefore, the same value is good for sets obtained under similar conditions. If the VDT is wrongly determined, the algorithm may fall into errors such as taking background pixels as object pixels or as separate objects (when the VDT is lower than the ideal one), or taking object pixels as background ones.

3.2. Connected components labeling. First version

The algorithm consists on the labeling of the connected components of a binary image, followed by measurements over the regions found. Connected components labeling algorithms group together all pixels belonging to the same region and assign an only label to them (it is a grouping operation). The region is a more complex unit than the pixel, and presents a larger set of properties (shape, position, gray level statistics, etc.), so for each region an n-tuple of those properties can be built. One way of distinguishing defective objects or objects with different characteristics is to distinguish the different regions according to their properties.

The speed and storage cost of the algorithm executing this labeling operation is extremely important, and there are different variants to carry out this stage. Usually one row of the image is processed at a time, and new labels are assigned to the first level of each component, with attempts to extend the label of the pixel to its neighbors to the right and downwards.

An algorithm, proposed by Haralick, does not use an auxiliary storage to produce the labeled image. It consists of an initialization step plus a top-down labels-propagation sequence, followed by a bottom-up propagation, until there are no changes in the labels.

A different possibility is the “cyclic algorithm”, based on the connected components classic algorithm for graphs described by Rosenfeld and Pfaltz. It only goes through the images twice, but it requires a large global table to store equivalencies. The first run carries out labels propagation. When the situation arises that two different labels can be propagated to the same pixel, the smallest one is propagated, and each equivalence found is added to the table. Each entry of the equivalence table is an ordinate pair (label, equivalent label).

After the first run, the “equivalence classes” are found by means of the transitive closing of the table equivalencies set. Each equivalence class is assigned to an only label, usually the lower (or the older) of the class. The second run assigns to each pixel the equivalence class label corresponding to the label assigned to the pixel during the first run. The main drawback this approach presents is that the larger the images, the larger the global equivalencies table, so for large images with many regions the table turns out to be too large.

The initial idea was to process the image based on the threshold, but these results were not enough because the image generally presents imperfections (usually produced by lighting effects) which were considered as objects. On the other hand, the implementation of a labeling algorithm would allow to classify several objects from one image, as well as to process control marks in order to, for example, determine objects size in a standard measurement unit.

The labeling operator also generates a table of regions showing the amount of pixels belonging to each of the regions. By analyzing this table we can obtain information to determine which is the biggest object in the image (the interest object), which are the very small objects representing noise or imperfections, and which are the objects of the pre-established size which in this case are control marks.

The first implementation performed two runs for each horizontal line of the image. During the first run, from left to right, it labeled pixels according to its upper and previous neighbors; and during the second run, from right to left, it solved the equivalencies generated in this line taking into account only the pixel to the right. This algorithm worked very well with circular or polygonal objects, but not so well with horseshoe-shaped regions with their opening oriented towards their upper part, since the equivalencies detected in one line did not propagate towards the upper ones.

3.3. Connected components labeling. Second version

In order to solve the problems presented by the first version, a new implementation was developed to carry out a first run over the image, line by line and in only one direction. The equivalencies presented are stored in a table generated during the first run. Then a second run is done in the same direction as the first one (even though this is not important) during which, for each pixel, its label is changed by the one corresponding to its label in the equivalencies table (if it had one).

The problem of this algorithm is the maintenance of the equivalencies table, since as equivalencies are added to the table a graph determining the equivalencies of each label is generated. In order to know these equivalencies, a transitive closure of this graph is necessary, which is not a trivial task, and depending on the shape of the image the algorithm performance could be decreased. Nevertheless, it presents a proper functioning for any geometric shape.

3.4. Average color calculation

After the identification of the pixels forming the object to classify (which is the larger region resulting from the labeling process) the average color is determined simply by adding up the colors of these pixels and dividing the result by the amount of pixels. To obtain a satisfactory result the object must predominantly be of a uniform color. In addition to this, the image must have been acquired under lighting conditions not altering the characteristics of the object.

3.5. Stains identification

One of the objectives of the application is to determine the existence of defects on the object to be classified. To do this, an algorithm able to detect these defects or stains on the main object identified by the labeling process was developed. This algorithm works in a way similar to the threshold process, but to modify the value of each pixel by black or white, not one but two values are used: if the pixel value is within the determined range, 0 is assigned (black), since this is the object; otherwise 255 is assigned (white) because this would be a stain. The two values mentioned are determined according to the object average color and to a parameter showing the so-considered normal variation in color.

When analyzing the results obtained with this algorithm, it was noticed that if the image presented shade variations due to lighting differences on the surface of the object, very illuminated or very dark areas were considered as stains. To overcome this, another algorithm assuming that the predominant color of the object presents different intensities was implemented. Thus, in order to determine if a pixel is a defect of the object or not, its color must be similar to the average color, and a linear variation in the three components (red, green and blue) is allowed. In this case, even though the color is not the same, it is considered to be normal (not belonging to a defect of the object). The following calculation is done:

1° - Normalization of the RGB components of the average color:

$$\text{Component}_i \text{ Normalized} = \text{Component}_i \text{ Average} * 100 / 255 \quad \forall i \in \{\text{red, green, blue}\}$$

2° - For each pixel of the object, calculate:

$$\text{Component}_i \text{ Difference} = (\text{Pixel Component}_i * 100 / 255) - \text{Normalized Component}_i$$

3° - In the ideal case, the three Difference values would be equivalent for object color pixels, and different for defect pixels. Since this is not always the case, a tolerance limit is used for variation between these Difference values.

The result is a binary image with the stains found on the object in white. On this image a labeling process is carried out in order to calculate the amount of defects and their size.

3.6. Borders detection

By analyzing the border or perimeter of the object relevant information can be determined, such as shape, diameters, etc. This is the reason why this facility was considered as an useful feature for the application.

A pixel of a given region is considered to belong to its border if some neighboring pixel (using connectivity 4) does not belong to the region. As pixels are found, their coordinates are stored on a list in order to process them later in a quicker way. The algorithm finds both outer and inner borders - in case the region has holes, which must be taken into account when using the coordinates of these pixels for moments calculation.

3.7. Centroid or center of mass determination

One of the first order moments of a region is its centroid or center of mass. In the case of circles or squares, it coincides with the geometrical center. The importance of this value lies in the fact that, along with the border, it may help to determine interesting features of the objects, as well as in the fact that, if there were marks on the image to facilitate the calculation of distances, these distances are measured from the center of the marks, therefore the center has to be determined. The calculation is as follows:

$$\text{Centroid of } R = C_{xy} = \frac{\text{Sum } (x, y)}{\forall (x,y) \in R} / \text{Area } (R)$$

3.8. First order moments calculation

There are two first order moments which are suitable to identify the general shape of an object. They are:

Average distance from the center to the perimeter:

$$\mu_r = (1/K) * S (| (x,y) - C_{xy} |), \quad \forall (x,y) \in \text{perimeter, and } K \text{ the amount of pixels.}$$

Standard deviation of μ_r :

$$s_r^2 = (1/K) * S (| (x,y) - C_{xy} | - \mu_r)^2$$

4. Parallel languages and algorithms

Factors affecting the performance of an algorithm in a particular architecture depend on the parallelism degree and on scheduling overhead and task synchronization. For a task-level parallelism, the greater the parallelism, the higher the synchronization and scheduling cost, which leads to the non-obtention of a linear increase in speed when increasing the amount of processors. The selection of an algorithm for the resolution of a particular problem is strongly influenced by hardware architecture and available software tools [Cof92] [Hee91] [Hwa93] [Lei92] [Mor94].

In order to choose the parallelism model to be used (data, task, or systolic parallelism), it should be taken into account that not every kind of parallelism will be suitable to solve a specific problem in an efficient way. More often than not, algorithms are developed without any references to a particular architecture, and they use more than one parallelism model, which makes them difficult to implement.

The paradigm of data parallelism is explicitly synchronized and mapped according to the SIMD programming model. Even though this is an optimal model, for example, for low level images processing, the way in which high level reasoning tasks are to be implemented or represented is not obvious. Task-level parallelism requires the algorithm to be partitioned in sub-tasks, and these, along with data, to be distributed among processors; therefore the system should be configured to allow communication and synchronization among processors. For the systolic model, the algorithm is temporarily partitioned and each stage computes a partial result which is then passed on to the following stage.

For a language to be parallel, it should support some of the parallelism and communication concepts. Parallelism can be at a processes level (Ada), at an objects level (Emeral), at a sentences level (Occam) [CSA90] [Hoa85], at an expressions level (functional languages), or at a clauses level (Parlog). Communication may be either point-to-point or broadcast.

Parallel Virtual Machine (PVM)

It is a set formed by software tools and libraries which emulate in a flexible way a framework for general-purpose heterogeneous concurrent computations on interconnected computers with different architectures [PVM96]. The main objective of PVM is to allow parallel computations to be run on a set of computers. The PVM basic principles are:

- User-configured computers pool (it may even be modified during running time by adding or removing computers)
- Transparent access to hardware (however, facilities of specific computers may be exploited by placing certain tasks in the more suitable computers).
- Process-based computing. The processing unit is the task, and multiple tasks can be run in an only processor.
- Explicit message-passing model. Tasks cooperate by sending and receiving explicit messages from one to the other.
- Heterogeneity support (in terms of computers, networks and applications).
- Multiprocessors support. It uses native facilities for messages passing in multiprocessors to take advantage of their specific hardware.

The computational model is based on the notion that one application consists of several tasks, and that each of them is responsible for a part of the work. It supports task and data parallelism (even mixed). Depending on the function, tasks can be run in parallel and they may need synchronization and data exchange by means of messages.

The PVM system supports C, C++ and Fortran languages. These languages have been chosen on the grounds that most applications are written using C or Fortran and that new applications are based on object-oriented methodologies and languages. User library interface for C and C++ is implemented as functions. The programming general paradigm of an application using PVM is the following: the user writes one or more sequential programs in the supported languages which contain calls to the library. Each program is considered a task integrating the application. For the execution, the user generally starts an instance of the "master" task from a computer included in the pool, which will in turn execute other tasks, the result being a set of processes performing local calculations and exchanging messages with others to solve the problem.

5. Algorithms parallelization

To carry out a parallel processing of the images, a "master/slave" model was used. The master process is responsible of dividing processing activities among a certain number of slave or client processes, and of coordinating their calculation. Clients are in charge of performing the processing activities themselves.

There is parallelization at a data level: the image is partitioned in sub-images which are sent to the clients (all of them identical) in charge of the processing activities. Some algorithms are easily parallelized, whereas for others special care must be taken as regards master participation in the processing of activities to coordinate clients. Each of the developed algorithms will now be analyzed.

5.1. Threshold

It does not require any special consideration, since it only analyzes each pixel of the region and compares it with the limit value. Therefore, the result obtained is the same independently of the amount of clients used.

5.2. Labeling

It presents parallelization difficulties mainly due to the fact that regions may be divided in two or more sub-images, and also to the fact that labels must be unique. In order to solve

these problems, the two runs carried out during the already described labeling process were divided.

Each client carries out the first run on its sub-image in the same way as the conventional algorithm does. It then sends its local labeling table to the master process. To solve the conflict generated when a region is divided into two or more sub-images, the lines corresponding to the image division point are sent to two clients, as the last line of one of the sub-images to one of them and as the first line of the other sub-image to the other. Client processes send these two special lines to the master with the labels assigned. The master concatenates the partial labeling tables sent by the clients into a global table and simultaneously checks the lines in common. In case there is a conflict (the clients involved labeled the pixels in that line with different labels) it adds the corresponding equivalence to the global table.

Once this information is processed, the master sends to the clients the corresponding portion of the labeling table, which may now have new equivalencies. The clients receive this table and carry out the second run taking the modified table as the starting point. When the global table is created, the problem of two different regions with the same label is also solved, since in the global table partial labels are modified by adding a number to them in order to obtain different labels.

The master process of this algorithm performs a part of the processing activities: it unifies labeling tables and it solves conflicts on overlapping image lines. This may not be efficient in cases where the amount of processors is very large (for instance, one processor per image line), since the master would be processing too many lines, which would result in a loss of the advantages provided by parallelism. Since this application was created to be used in a local network with a limited number of conventional computers, this is not a problem (for instance, if there are 5 computers and a 200-line image, the master would only process five lines, i.e., the 2.5% of the image). This in turn reduces the use of the communications network since otherwise clients would have to communicate among them and with other process providing unique labels.

5.3. Average value calculation

For a distributed calculation of the average color, the previously described algorithm was used, but now each client sends its partial result to the master, which in turn calculates the average among all clients, taking into account that this average must be weighed according to the number of pixels considered in each process.

5.4. Borders detection

In order to determine perimeter pixels in parallel, the fact that the image is divided must be taken into account. This means that the pixels of the region located on the first and last lines cannot be considered to belong to the perimeter of the region because there may be intermediate partitionings of the image.

In order to solve this inconvenient, processes need to know which portion of the image they are processing. These portions may be either the first one (which means that the first line of this sub-image is the limit of the complete region), an intermediate one (neither the first nor the last lines are the limit of the complete image), or the last one (the last line is the limit of the complete image). It should also be taken into account that the lines where the image is divided are processed twice, and it is necessary, in case the border of the region falls there, to avoid counting it twice.

To do this, an algorithm similar to the conventional one was developed, with the difference that this one takes into account every one of the presented cases. This algorithm solves the already mentioned problems and therefore allows the identification of pixels belonging to the perimeter of the region without being duplicated and avoids taking as part of the border pixels from the borders of inner partitions of the original image if they are not borders.

6. Implementation and development environment

In the case of the implementation, there were certain features to be taken into account, namely that it should be clear and easily extensible, and that PVM was available to be used with C and Fortran. This is the reason why it was developed in C++ (using PVM and on a Linux environment), since it provides object-oriented programming possibilities [Rum91], which facilitates the development of a clear, modular and extensible code, and, in addition to this, it is possible from C++ to access PVM routines developed in C.

As already mentioned, the application is oriented to become a process applied in real time on the images obtained with a video recorder, and results are sent to devices deciding the direction of the objects according to their characteristics. However, a graphic environment had to be used during the development in order to observe the results of the images at each processing step in order to determine algorithms correction.

The Linux X-Windows graphic system, developed on a client-server architecture [Uma93], allows, in a transparent way, to see data generated on any of the network computers on a screen. Even though interface elements can be easily handled by using the API of the X-Windows system, it is not so simple to show true color images as the ones used by the application.

To simplify the development, a graphics processing application simple as the Gimp library was used, its advantage being the possibility of adding plug-in processes. Therefore, it was decided to develop the graphic application as a Gimp plug-in and thus be able to use their functionality to show images in a transparent way. Gimp allows to open an image in different formats (Jpeg, Tiff, Targa, Gif, etc.), to show it, and to apply a series of processes to it, implemented in a plug-in way. These processes are configured by adding a file (.gimprc), and Gimp allows the user to select one of the processes on the list to apply it to the loaded image. Once the process is selected, Gimp executes it and sends the image to be processed by means of pipes. The process is run independently from Gimp, and it may communicate with Gimp through a simple API which allows to show new images, data input dialogs, etc.

Implementation consists of two applications: a graphic one (as already said, mainly for system testing and development) and a batch-type one (which can be used in combination with the acquisition stage and with the stage sending the objects to the appropriate destiny).

The graphic application is a Gimp plug-in which, when run over a previously read image, presents a dialog box which allows to specify processing parameters. These parameters are: amount of processes to be used, threshold value, size of the defects to be taken into account as related to the object, difference in color of one region as compared with the average color to consider it a defect, whether moments are to be calculated or not, and whether partial results are to be shown (only for debugging). Then processing is done, and a dialog box with the results appears at the end of the process: surface (amount of pixels) of the main object, average color of the object, maximum diameter, (optional) values of the moments μ_r and s_r^2 , amount of stains (defects), surface of the defects, centroid of the object, center of reference marks (if they are any), and processing time. On the other hand, the batch application was run, which was useful to carry out performance measurements. The process is very simple and it allows to use the same values used for the graphic application as parameters in the command line. Processing results are seen on the screen at the end of the process. To load Jpeg images, a library provided with Linux distribution was used (libjpeg).

In order to abstract the main objects of the application, which are RGB images formed by pixels and algorithms applied on these images, classes were defined. Classes were also defined in order to encapsulate calls to PVM (instances and process communication), so that this library could be later on changed by a similar one without modifying the rest of the code. The encapsulating of algorithms as classes allows the addition of new algorithms in a very simple way and following a clear scheme, in addition of which it allows to generate lists of algorithms to be sequentially applied to an image.

Classes to abstract images and pixels. The *Himage* class encapsulates an RGB image, which is a sequence of pixels, each of which has three color components. These pixels are encapsulated in the *Hpixel* class. The *Himage* class also encapsulates a cursor (*Cursor*) which allows to run over the pixels in a tidy way.

Classes to abstract algorithms. As already expressed, the algorithms used are encapsulated in classes. In order to do this, there is a virtual class which is the superclass of all algorithms (*HimageAlgo*), which defines a standard interface for all of them. The use of cursors to run over the image facilitates in a significant way the implementation of algorithms, since they allow to access the image internal information independently and transparently from its format. It should also be mentioned that, since it is possible to define several cursors at the same time on the same image, the implementation of more complex algorithms such as labeling or borders detection is also simplified, since, for instance, four additional cursors to run over the four connected neighbors of the pixel being processed can be defined.

Classes to abstract master and client processes. The parallelism model chosen was the master-client one: there is a master process which coordinates processing activities, and several client processes which carry out processing activities. This caused the definition of two classes to encapsulate these processes (*HMaster* and *HClient*). The general idea of the application is that an instance of an *HMaster* process is created, all necessary parameters are set (image to process, number of processes among which the processing activities will be distributed, threshold values, etc.), and the processing task begins by calling a particular method of the object (`process()`). This method instantiates the corresponding client processes according to the parameters, partitions the image among these processes and then coordinates their partial results in order to obtain the final result. On the other hand, client processes, when run, create an instance of the *HClient* class and begin the process by calling the `process()` method. This method receives the results, processes them, and synchronically exchanges them with the instantiating master process.

Classes to abstract PVM processes. There are basically two types of processes with different communication requirements. On the one hand, we have the master process, which knows a set of clients with which it communicates; and on the other hand there are client processes, which only interact with their instantiating master. In order to model these two cases, a class called *HPvmProxi* was created, which represents a remote process. The master sees each client as an instance of this class, and client processes in turn have an only instance representing the master. This class has the necessary methods to communicate with the remote process, and it particularly keeps its identifier in order to send and receive messages to and from it. Since PVM has functions to send different kinds of data (ints, longs, floats, etc.), this class has overcharged *send* and *receive* methods with a particular implementation for each type of data. On the other hand, a class called *HPvmProcessArray*, which basically manages a proxies number to facilitate the implementation of the master, was defined.

7. Results obtained

The fact that the application was developed on a heterogeneous processors network with limited resources must be taken into account when evaluating the obtained results. In addition to this, since a Linux environment was used, the fact that Linux performs a particular scheduling of processes which distributes CPU time among the application running and management processes of own resources of the operating system must also be considered. This distribution can be modified by altering processes priorities, in spite of which the application does not have a 100% of the processing time.

Two Pentium (133 MHz) processors, with 32 and 48 RAM Mb, and a 80486 DX4 (100 MHz) with 12 Mb were used for the tests. In order to carry out measurements two different network configurations were built:

- The 32 Mb Pentium with the 80486, using 10 Mbits/sec EN-2000 compatible network boards.
- The two Pentiums with the same type of network boards.

Successive runs of the application, changing for each of them the number of client processes, the number of hosts, and the image to process, were carried out. In addition to this, each case was run several times in order to minimize the memory scheduling effects of the Linux. As the development of tests advanced, the running time of the application significantly varied between runs with an even or odd number of client processes, when the number of hosts of the virtual machine was even. When analyzing each test it was observed that PVM distributed in an equitable way the processes it executed among the hosts of the virtual machine. Thus, when the number of clients to run was odd, for example three, PVM run two client processes in one host and one client together with the master in the other host.

This distribution of processes causes an increase of running times: the host processing the two clients takes more time than the other host because the master process needs little running time. In the case of three clients, as the image to be processed is equally distributed among the three of them, the final processing time will be the time the host with two clients needs to process two thirds of the image. As opposed to this, if the total number of clients is four, the final processing time will be the time used by the host with two clients and the master to process half the image, which is quite lower than that of the test with three clients.

An example of the difference in running times between runs with even and odd numbers of processes can be seen on the following table (Note: time is always expressed in seconds):

Processes	1	2	3	4	5	6	7	8	9	10
Time	6.18	5.85	5.48	5.1	5.77	5.55	5.95	5.76	6.43	6.28

As a consequence of this analysis, it was decided to carry out tests with an even number of processes (in addition to runs with only one client process), so that the results obtained showed in a clear way the difference in processing time when increasing the number of clients. (However, if the tests are run with an odd number of processes, variation in running time is percentually similar).

The first tests consisted in running the application on a virtual machine with an only host (on one of the Pentiums 133 with 48 Mb RAM). The resolution of the image chosen for these tests was 264 x 384 pixels in 24 color bits. The results obtained were the following:

Processes	1	2	4	6	8	10
Time	4.36	3.695	3.47	3.71	3.865	4.33

The following series of measurements was done after adding a new host to the previous configuration (the other Pentium), processing the same image. The results were:

Processes	1	2	4	6	8	10
Time	4.3	2.84	2.675	2.7	3.025	3.55

Figure 1 shows running times of runs carried out with one and two CPUs. The improvement obtained considering the best times for each case was 1.29 seconds.

In order to evaluate in a better way the behavior of the application with different work loads, the same measurements were done on a bigger image (522 x 84 pixels on 24 color bits). The results obtained when running the application on a virtual machine with an only host were:

Processes	1	2	4	6	8	10
Time	17.9	15.94	14.01	14.84	15.13	15.66

When adding another host to the virtual machine, the following results were obtained:

Processes	1	2	4	6	8	10
Time	16.95	8.206	10.77	10.84	11.05	11.6

Figure 2 shows running times for runs carried out with one and two CPUs. The improvement obtained, considering the best times for each case, was 1.7 seconds.

In order to study the performance of the application in heterogeneous environments, a network was configured with the 133 MHz Pentium and the 80486. First the application was run on a virtual machine with an only host (the 80486), the following results being obtained:

Processes	1	2	5	6	7
Time	10.41	9.95	10.03	10.23	10.38

By adding a host to the virtual machine (the 133 MHz Pentium) running times were markedly reduced, only if when there was an odd number of client processes, the highest number (half of the processes plus one) were run on the fastest host. The results were:

Processes	1	2	5	6	7
Time	7.12	6.65	4.7	5.82	5.76

Figure 3 shows running times of runs performed with one and two heterogeneous CPUs. The improvement obtained considering the best times for each case is 2.11 seconds.

The following test was carried out on the same network configuration, but with a 522 x 804 pixels image in 24 bits, with a significant performance improvement expected when comparing the run on the 80486 as an only host (case I) with another run with the Pentium 133 as a second host (case II). The best running-times obtained were 35.9 seconds (for case I) and 15.3 seconds (for case II). The improvement obtained was around 2.37 seconds.

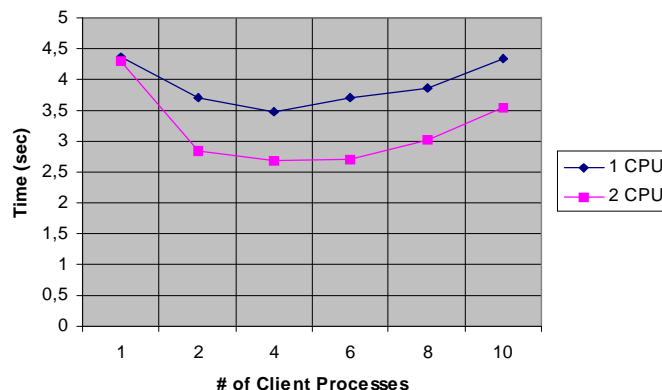


Figure 1.

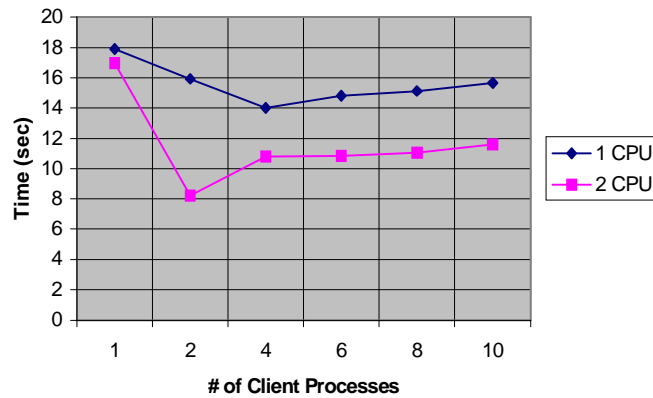


Figure 2.

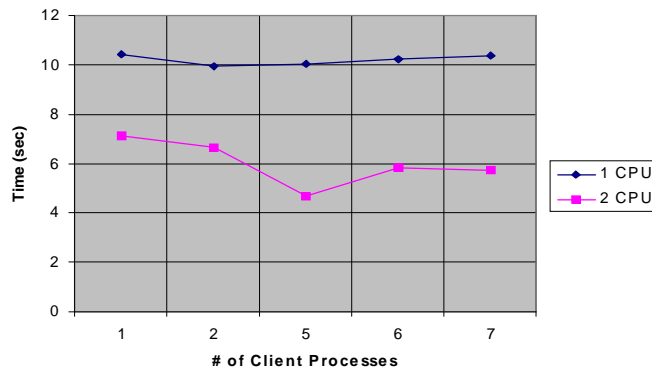


Figure 3.

8. Conclusions and future lines of work

It would be convenient to evaluate separately the tools used and the results obtained. As regards the tools, Linux turned out to be very stable, which is translated into an efficient hardware resources management. In addition to this, it provides facilities for networks with a TCP/IP configuration (this characteristic is very important if we consider the orientation of the application), it gives a version of the gnu gcc compiler for C and C++ (with highly satisfactory services), and a good PVM implementation. In short, environment selection proved to be suitable for an application with these characteristics (interconnection, concurrency, and handling of very large data volumes), in addition of which it allows to exploit portability.

As regards X-Windows, there are free-ware libraries which simplify its use, it allows to run applications, and results can be seen in different computers, which facilitates the execution of tests. Among the negative characteristics, we can mention the following: it has a complicated and non-friendly API, it is hard to configure in order to use high graphic applications, and documentation is oriented to advanced users.

On the other hand, PVM makes the communication between distributed processes and the migration from small to large configurations - since the configuration of the virtual machine is independent of the application using it - completely transparent. Its main drawback is the fact that it does not allow to define protocols in the Occam way to send complex data as a structure, which results in the need for several communications, each of them with a different type.

The language used (C++) allows to define inheritance between classes and data encapsulation, which in turn allows to make a clear and easily extensible implementation; and due to its low level characteristic, it also allows an optimal disposition of the available resources, even though it requires excessive care when handling memory and the debugger

provided by the version of the compiler used is non-friendly, error detection being thus complicated.

The results obtained during the performance measurement stage suggest that the implementation of solutions using gross-grain parallelism results in significant improvements only when the processes involved are complex (that is, if they require a long processing time) and there are no excessive communication needs during running time. This is due to the fact that the cost of communications is significant in relation to processing speed, since a LAN was used.

Even though the results obtained are highly satisfactory as regards the objectives set at the beginning of the research, alternative lines of work can be proposed in order to obtain improvements in certain aspects, such as:

- Implementation of a processes manager which can dynamically detect the hosts which are faster than the virtual machine (in order to avoid an unbalanced distribution of tasks among processors with different speeds), and assign to them a higher processing load, which is translated into more image to process. Thus, PVM equitable distribution of processes would be avoided, the available CPU time being thus better exploited.

- Establishment of a way of assigning to the application processes as high an execution priority as possible, so that Linux does not devote too much CPU time to its internal scheduling processes.

- Use of other network configurations and communication systems (for example optical fiber) in order to reduce data transference times between processes.

In addition to these improvements, there are aspects which were not considered, such as the module for image acquisition from the transporting belt, its digitalization and improvement for processing, or the final stage of the application in charge of evaluating the results obtained (when processing a certain image) and on this basis make decisions (for example, sending the object to its destiny according to classification results).

9. Bibliography

[Bax94] G. A. Baxes, "Digital Image Processing. Principles and Applications", John Wiley & Sons Inc., 1994.

[Cof92] M. Coffin, "Parallel programming- A new approach", Prentice Hall, Englewood Cliffs, 1992.

[CSA90] "OCCAM", Computer System Architects, 1990.

[Gon92] R. C. González, R. E. Woods, "Digital Image Processing", Addison-Wesley Publishing Comp., 1992.

[Har92] R. M. Haralick, L. G. Shapiro, "Computer and Robot Vision", Addison-Wesley Publishing Company, 1992.

[Hee91] D. W. Heermann, A. N. Burkitt, "Parallel Algorithms in Computational Science", Springer-Verlag, 1991.

[Hoa85] C. A. R. Hoare, "Communicating Sequential Processes", Prentice-Hall, 1985.

[Hus91] Z. Hussain, "Digital Image Processing", Ellis Horwood Limited, 1991.

[Hwa93] K. Hwang, "Advanced Computer Architecture. Parallelism, Scalability, Programmability", McGraw Hill, 1993.

[Jai89] A. Jain, "Fundamentals of Digital Image Processing", Prentice Hall Inc., 1989.

[Jai95] R. Jain, R. Kasturi, B. G. Schunck, "Machine Vision", McGraw-Hill International Editions, 1995.

- [Law92] H. Lawson, "Parallel processing in industrial real time applications", Prentice Hall 1992.
- [Lei92] F. T. Leighton, "Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes", Morgan Kaufmann Publishers, 1992.
- [Lev90] S. Levi, A. Agrawala, "Real Time System Design", McGraw-Hill Inc, 1990.
- [Mor94] H. S. Morse, "Practical Parallel Computing", AP Professional, 1994.
- [PVM96] "Parallel Virtual Machine", World Wide Web.
- [Rum91] J. Rumbaugh, M. Blaha, M. Premerlani, W. Lorensen, "Object-Oriented Modeling and Design", Prentice Hall, Englewoods Cliff, 1991.
- [Sch92] R. Schalkoff, "Pattern Recognition. Statistical, Structural and Neural Approaches", 1992.
- [Uma93] A. Umar, "Distributed Computing and Client-Server Systems", P T R Prentice Hall, 1993