

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

DEPARTAMENTO DE INFORMÁTICA Y SISTEMAS



TESIS DOCTORAL

**SISTEMA BASADO EN CONOCIMIENTO PARA SEGMENTACIÓN
DE IMÁGENES. DESARROLLOS Y APLICACIONES**

JORGE CABRERA GÁMEZ

Las Palmas de Gran Canaria, 1994

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

DOCTORADO EN INFORMÁTICA

DEPARTAMENTO DE INFORMÁTICA Y SISTEMAS
PROGRAMA DE PERCEPCIÓN ARTIFICIAL Y APLICACIONES

***SISTEMA BASADO EN CONOCIMIENTO PARA
SEGMENTACIÓN DE IMÁGENES.
DESARROLLOS Y APLICACIONES.***

Tesis Doctoral presentada por D. Jorge Cabrera Gámez.

Dirigida por el Dr. D. Antonio Falcón Martel.

Codirigida por el Dr. D. Francisco M. Hernández Tejera.

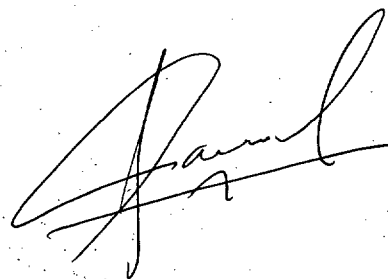
El Director



El Codirector



El Doctorando



Las Palmas de Gran Canaria, a 1 de Julio de 1994.

A Verónica por su comprensión y estímulo,
a Raquel y a Jorge por su paciencia,
a mi familia por su incondicional apoyo durante estos años.

AGRADECIMIENTOS.

Muchas son las personas que de una u otra forma han colaborado en el desarrollo de este trabajo, a todas ellas mi agradecimiento. En especial, quisiera agradecer a mis compañeros del Grupo de Inteligencia Artificial y Sistemas de la Universidad de Las Palmas de Gran Canaria, Antonio Falcón y F. Mario Hernández su alentadora dirección, a Juan Méndez su inspiración y acertadas sugerencias, a Javier Lorenzo, Modesto Castrillón y Daniel Hernández su valiosa colaboración en la construcción de SVEX. Finalmente, he de agradecer a Gabriel de Blasio el compañerismo demostrado en el quehacer docente y a Rafael Nebot la excelente programación del Editor de SVEX.

INDICE

INDICE.

Extended Abstract.

Capítulo I. Segmentación e interpretación de imágenes.

I.1 Introducción.	43
I.2 La segmentación de imágenes como un proceso de simbolización.	45
I.3 Métodos y técnicas.	49
I.3.1. Detección de contornos.	50
I.3.2. Detección de regiones.	56
I.3.2.1. Segmentación basada en el espacio de propiedades.	56
I.3.2.2. Crecimiento de regiones.	61
I.3.2.3. División y fusión.	64
I.3.2.4. Segmentación de regiones iterativa. Relajación.	70
I.4. Sistemas expertos para análisis de imágenes.	71
I.4.1 Sistemas consultores para proceso de imágenes.	73
I.4.2 Sistemas basados en conocimiento para generación de programas de proceso de imágenes.	74
I.4.3 Sistemas de segmentación de imágenes guiados por objetivos.	77
I.4.4 Sistema basado en reglas para segmentación de imágenes.	78
I.5. Sistemas de interpretación de imágenes.	82
I.5.1 ACRONYM.	83
I.5.1.1. Arquitectura.	84
I.5.1.2. Representación del conocimiento	87
I.5.1.3. Control.	89
I.5.2 SPAM.	91
I.5.2.1. Arquitectura.	91
I.5.2.2. Representación del conocimiento	93
I.5.2.3. Control.	95
I.5.3 SIGMA.	97
I.5.3.1. Arquitectura.	98
I.5.3.2. Representación del conocimiento	101
I.5.3.3. Control.	103
I.5.4 Schema System (VISIONS).	105
I.5.4.1. Arquitectura.	105
I.5.4.2. Representación del conocimiento	108
I.5.4.3. Control.	110
I.5.5 Discusión y conclusiones sobre líneas generales de diseño.	113

Capítulo II. Elementos para el diseño de un sistema multinivel de segmentación de imágenes.

II.1	Introducción.	117
II.2	Hacia una metodología en el diseño de sistemas Percepto-Efectores.	118
II.2.1	La organización en niveles.	120
II.2.2	La estructura de un nivel.	126
II.2.3	La representación simbólica.	130
II.2.4	Las transformaciones entre niveles.	134
II.3	Propuesta de máquina multinivel para segmentación de imágenes.	136
II.3.1	Organización por niveles y arquitectura del sistema.	137
II.3.2	En el nivel de los pixels.	141
II.3.2.1	Arquitectura del Procesador de Pixels. Lenguaje de definición.	143
II.3.2.2	Descripción de objetos. Procesador BU.	145
II.3.2.3	Sistema de control. Procesador TD.	159
II.3.3	En el nivel de los segmentos.	159
II.3.3.1	Arquitectura del Procesador de Segmentos. Lenguaje de definición.	161
II.3.3.2	Descripción de objetos. Procesador BU.	163
II.3.3.2.1	Del pixel al segmento.	164
II.3.3.2.2	Descripción simbólica de los segmentos.	168
II.3.3.3	Sistema de control. Procesador TD.	176

Capítulo III. Sobre el diseño y desarrollo.

III.1	Introducción.	185
III.2	Consideraciones sobre algunos aspectos de la implementación.	186
III.2.1	El presegmentador "Flood".	186
III.2.1.1	La transformada Watershed.	187
III.2.1.2	Adaptación de la transformada Watershed en el presegmentador "Flood".	193
III.2.1.3	Objetos del lenguaje.	199
III.2.2	Un Algoritmo de Análisis de Componentes Conexas (AACC).	203
III.2.2.1	Introducción.	203
III.2.2.2	Una propuesta de AACC.	205
III.2.2.2.1	Detección de esquinas en mapas binarios.	206
III.2.2.2.2	Detección de esquinas en mapas multivaluados.	207
III.2.2.2.3	Procesado de pares de esquinas. Operaciones sobre bordes.	210
III.2.2.2.4	Registro de vecindades.	215
III.2.2.2.5	Extensiones al algoritmo original.	220
III.2.2.2.6	Análisis experimental del AACC propuesto.	231

III.3 Herramientas de desarrollo.	238
III.3.1 Proceso de generación de una aplicación. (compilador y librerías)	238
III.3.2 Un prototipo de editor icónico.	241
III.3.3 Una herramienta de adquisición de muestras.	242
III.3.4 Depuración de programas.	244

Capítulo IV. Aplicaciones.

IV.1 Introducción.	247
IV.2 Aplicaciones a nivel de pixels. Casos de estudio.	248
IV.2.1 Control de calidad en peletería.	249
IV.2.2 Detección de movimiento.	253
IV.3 Aplicaciones a nivel de segmentos. Casos de estudio.	260
IV.3.1 Localización e identificación de piezas industriales.	260
IV.3.2 Segmentación de contornos.	281
IV.3.3 Segmentación desde propiedades morfológicas.	287
IV.3.4 Segmentación de imágenes de exterior.	291
IV.4 Discusión.	306

V. Conclusiones, principales aportaciones y posibles extensiones.

V.1 Introducción.	309
V.2 Conclusiones y principales aportaciones.	309
V.3 Posibles extensiones y líneas de desarrollo.	315

Referencias.

Apéndice A: Pseudocódigo de las partes más relevantes del AACC.

Apéndice B: Especificación BNF del procesador de Pixels.

Apéndice C: Especificación BNF del módulo de presegmentación "Flood".

Apéndice D: Especificación BNF del procesador de Segmentos.

Apéndice E: Programas utilizados en los casos de estudio.

EXTENDED ABSTRACT

This thesis describes the main outlines of a methodology for the design of perception-action systems and its application to the development of SVEX, a multilevel knowledge-based system for image segmentation.

The organization of this extended abstract is as follows. The first part introduces the experimental context of this work and summarizes the results of a bibliographical survey of related work in a set of design guidelines for Image Understanding Systems (IUS). The second part describes the proposed methodology for the design of perception-action systems that has guided the design of SVEX. The SVEX's architecture and main characteristics are the topics of the third section. The next section describes some aspects of SVEX's implementation considered as relevant contributions. The last two sections are devoted, respectively, to a description of the experiments used for testing SVEX, and a presentation of the conclusions, main contributions and future proposals of this work.

1. IMAGE UNDERSTANDING SYSTEMS AND IMAGE SEGMENTATION.

The first part of this thesis is aimed at giving an overview of the context in which the experimental part of this project has been developed. It provides a review of the three following topics, which we consider the most important in the development of an image segmentation system:

- A) The development of methods and techniques for segmenting images, and their conception as basic tools to carry out the segmentation of an image.
- B) The design of a flexible form for the expression of explicit segmentation knowledge, to allow the adaptation of the segmentation system to new environments, without major changes in the structure of the component processes of the system.

C) The definition and organization of a control strategy such that explicit high-level knowledge can be used by the upper levels of a system to control the course of segmentation.

The analysis of these questions divides the first chapter into three parts. In considering the first question, we have reviewed the most used segmentation techniques. The second part is devoted to the description of image processing expert systems. A review of these expert systems is in order, as they have tried different approaches to articulate, in a consistent framework, the heuristic knowledge about image processing techniques, aimed at obtaining more powerful and autonomous systems. Some of these expert systems have been incorporated into several Image Understanding Systems (IUS) as segmentation modules. The third part of this chapter analyzes some the best known knowledge-based IUS's, following a structured approach that considers for each system its architecture, the knowledge representation used, and its control strategy. There is evidence of the existence of a set of common design guidelines used to overcome the main problems found in designing an IUS.

The effort invested in the development of Image Understanding Systems (IUS's) has failed so far to determine which is the best design to produce an optimum IUS: a system that should be robust, reliable, easily adaptable to new tasks, efficient, etc. However, this effort seems to have produced some consensus about the main problems that must be considered by the designer, and the capabilities an IUS should possess to face those problems [Rise-84][Rise-87][Mats-90][Hans-88][Drap-89][Jain-91]. These ideas may be summarized in the following set of "design guidelines" for IUS's:

A) Organization by levels

The organization of IUS's into levels, following the nature of the information grain used at each level, is an organization principle present in the vast majority of these systems. The most-used scheme is that defining three levels: the low level, defined around the pixel as the information grain, the intermediate level that uses regions, lines or volumes, and the high level that operates with definitions of objects or models.

B) Bottom-up and top-down control.

The context or knowledge base must guide the system's active processes at each level to limit the searching space where the interpretation takes place, as well as increase reliability and computational efficiency. Control strategies driven exclusively by the data, common among the first IUS's, have shown to be unreliable. It seems conclusive that data-driven analysis must integrate with knowledge-driven in a uniform way in an interpretation scheme. For example, it is interesting that the activation of new hypotheses during the interpretation may command a resegmentation of certain zones in the image using new parameters or operators.

C) Knowledge representation.

An adequate choice for knowledge representation determines the ease of building the system and its efficiency. The different formalisms used for knowledge representation (frames, rules, knowledge sources in blackboard systems, ...) in IUS's are epistemologically equivalent; the main differences are found in the efficiency, explicitness of knowledge, and the ease of implementation or extension of the system. The knowledge base must contain the description of the models of the world and their expected appearance in an image; it should also include heuristic knowledge about image processing operations and segmentation algorithms. The description of objects' attributes must be done in terms that should be intelligible at each IUS level: properties of pixels, properties of segments (lines, regions or volumes), properties of objects, etc.

The elaboration of world models and the acquisition of application domain knowledge, pose a serious obstacle in the development of image understanding systems [Conn-87] [Niem-90b] [Fich-92]. This is demonstrated, among other reasons, by the fact that the best known IUS's have been applied only in one or two domains. To make progress in the development of these systems, it is necessary to achieve advances in the automation of the processes of model acquisition or learning.

D) The interpretation of an image as a dynamic process of hypothesis activation.

Segmentation and interpretation errors have a dramatic effect upon the reliability and efficiency of an IUS. The simultaneous consideration of multiple hypotheses about the objects that may be present in a scene when many of these hypotheses may be erroneous, imposes a needless overload on the system and degrades the effective use of knowledge during the interpretation. This requires the utilization of a control schema that will drive the interpretation from "the evident" to "the uncertain". In other words, the elaboration of the interpretation must be based on a mechanism that starts from an initial set of trusted or reliable hypotheses ("islands of reliability" [Hans-88]) and progressively refines it by chain-activation of new hypotheses from verified ones. The first assumption of this scheme is that it represents a good method for improving efficiency, since the effort of trying new hypotheses is ruled by their plausibility. Moreover, the combination of this control scheme with the concept of object hypotheses as active agents, provides a 'smart' form of organizing the control in a distributed mode.

E) Information fusion and uncertainty management.

A ubiquitous problem in artificial perception systems is the transformation of numerical measures or features, defined in the signal domain, into symbols or diagnostics in the symbolic domain. This problem has two important aspects. The first deals with the conversion of numerical features into symbols, where it is necessary to consider the fuzzy nature of the whole process, such that resulting symbols come out with an associated reliability factor or membership degree to a symbol class. The second is closely related to the first, and includes the definition of mechanisms for integration of multiple knowledge sources within a level or from different levels (integration of data from different or multiple sensors, diagnostics obtained from different operators, combination of evidences arising from different hypotheses) [Abid-92] [Agga-93]. Due to the fuzzy or uncertain nature of symbol assignments, these evidence-combination mechanisms need to have the capability of dealing with uncertainty.

F) A reliable segmentation.

The success of an IUS relies heavily upon the achievement of a reliable segmentation, the part of the system dedicated to this task probably being one of the most important in the whole system. The designer of such a segmentation module needs to bear in mind the following main points:

Objective guided. The segmentation module must be able to attend requests of high symbolic content, emitted from upper levels of the system. These requests are made in terms that are intelligible to the segmentation module and may include restrictions stated as spatial relations. For example, it might attend a request stated as: "*Inside the window (3,15,100,100), try to find a red circle that encloses a black square.*" The aforementioned requirements about the necessity of mechanisms for information fusion and uncertainty control are especially important at the segmentation phase. Diagnostics provided by the segmentation module must have associated an uncertainty or reliability measure, such that the transition between the numerical and the symbolic domains performs smoothly.

Adaptive and stable segmentation module. An IUS that is to operate in changing environments requires of a segmentation module capable of dealing with images of different quality and characteristics. Given a segmentation task, it should adapt its operation, that is, its selection of operators and parameter adjustment, to the image characteristics. This requires a means of measuring the segmentation correctness that is really an unsolved problem in Computer Vision. This capability will be very important in achieving stable segmentations, in the sense of obtaining similar segmentations for images of similar content but different spectral characteristics.

2. A DESIGN METHODOLOGY FOR PERCEPTION-ACTION SYSTEMS.

Computer Vision Systems (CVS) customarily form part of more complex systems that include some form of actuation over the world, sometimes as robotics systems or as general automatons. In this sense, a CVS may be considered as part of a more general perception-action system.

A Perception-Action System can be defined as a system that operates in an environment with the aid of sensors and the action of its actuators or effectors. Equivalent conceptions might be those of Autonomous Intelligent System or Autonomous Agent [Elfe-91][Maes-90][Brook-86]. In this thesis, we propose a qualitative version of a methodology for the design of these systems, stated as a few design principles or "Rules". We do not make any claims about the superiority of this methodology, as we think its validation must be done a posteriori, if the systems conceived using it tend to be better, more efficient or easier to build.

About the levels of a system.

The first principle in the methodology is revolves around the utilization of levels of abstraction. The organization of a system into levels is a common "Rule" to reduce design complexity. A level of abstraction is made up of a set of symbols that interact by means of specific procedures or operators, as suggested by the non-formal expression:

$$\text{LEVEL} = \text{SYMBOLS} + \text{OPERATORS}$$

The level organization we propose is based on conceiving a perception-action system as a machine where each level builds a representation from the data it receives from a "virtual world" and acts upon it. The virtual world each level "perceives" can be the physical world or the world

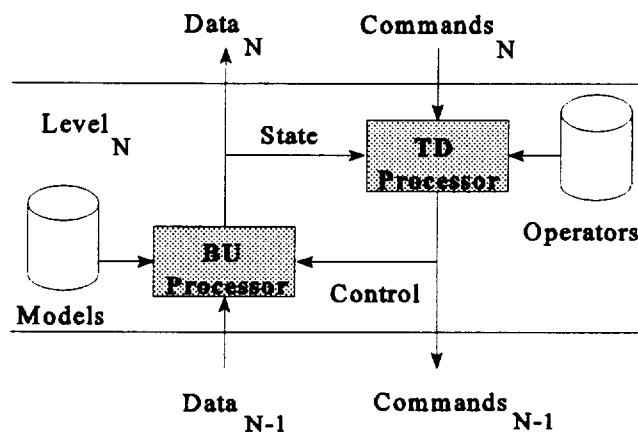


Figure 1. Generic architecture model for a level in a Perception-Action system.

representation produced by the previous level. Similarly, the actions executed by a level upon its virtual world can be orders directed to the effectors of the system or to a lower level. According to this paradigm, the different levels of a perception-action system are functionally equivalent though they operate at different representations. The functions of a level are:

1. To transform the representation of its virtual world into a more abstract one delivered to the next upper level.
2. To transform orders received from the upper level into actions that can be executed in its virtual world.
3. If the representation of its virtual world meets certain conditions, then a level can execute actions over its virtual world independently of the actions ordered from upper levels. This establishes a mechanism that can be considered a sort of "reflex act" that provides the system with adaptability and reactivity at different levels.

The first principle can be summarized in the practical following "Rule":

Rule 1. To design a perception-action system, organize the system by levels, using the paradigm of a machine that operates in a virtual world.

About the structure of a level.

If the machine's functions at each level are formally the same, but applied over different data, then the internal organization of each level's machine may formally be the same but processing different data.

Rule 2. The design of each level can provide a certain conceptual economy by proposing the same qualitative architecture for all the levels. Nevertheless, the specific details at different levels cannot be identical as they process different data.

Rule 3. An architectural model for each level based on the utilization of units comprising a BU (Bottom-Up) and a TD (Top-Down) processor, interconnected as indicated in Figure 1, possesses all the necessary elements to develop different paradigms of perception-action machines.

The objectives of each of these processors are the following:

BU processor (Bottom-Up). It is a unit for diagnosis, codification or abstraction. It transforms the world representation received as data using some type of coding into another more compact though complete representation. The transformation is based on the detection of some pattern or model in the initial representation and its substitution by a reference to the detected model. It is really a recodification of the message using a new code such that the message is shorter but has the same descriptive power about the world. The action of BU processor is determined by control orders emitted by the TD processor and the descriptions of the models contained in a model base.

TD processor (Top-Down). It is a unit for control, planning and decoding. It receives orders or requests from the upper level, expressed in an abstract form, and transforms them into sequences of simpler and more concrete orders taking into account the state of its virtual world. Some of these simpler orders are commands for the BU processor while others are forwarded to the lower level. It may also trigger actions when the state of its virtual world meets certain conditions. The transformation of orders is done using a set of operators and knowledge about the structure of the virtual world on which it operates.

As stated by the third rule, it is possible to define different paradigms of perception-action machines based on BU-TD modules. From the system engineer's practical point of view, the existence of a TD processor allows for dynamic linkages between a BU-TD module and several similar modules located in the inferior level. This makes possible the distribution of control and computation and is an elemental mechanism to provide a system using this architecture with coarse grain parallelism. The communication among BU-TD modules in different levels facilitates

cooperation between levels and modules, without the necessity for centralized or hierarchical control, with its consequent loss of reactivity, since there are generalized reflex acts that can be triggered by the TD processor of each module. Following these ideas, Figure 2 depicts a three-level system where the BU-TD modules are conceived as level processing units. These units can accept requests from the upper level to compute diagnostics or do the same with one or several units placed at the inferior level. Some of these units receive data from sensors and others control effectors with or without supervision from the upper level. In this example, the BU-TD modules may have assigned specific tasks (receive data from sensors, control an effector, etc.) or act merely as processing units with the capability of downloading tasks upon request from upper level modules.

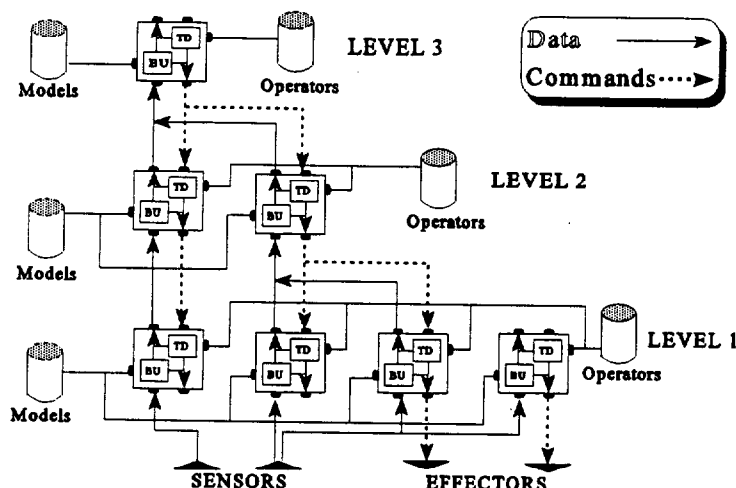


Figure 2. A three level perception-action system based on BU-TD modules.

About the symbolic representation.

The construction of machines with "intelligent" behavior has evolved from its early times towards the consideration of symbol processing vs. number processing machines. This orientation may be well-founded in that symbolic representations provide the most suitable environment for the performance of advanced processes for reasoning and world representation at a more abstract level than that of sensors and effectors. Although this consideration is valid for the vast majority

of situations, there may be others in which a numerical machine may be more appropriate, e.g. a connectionist machine.

We think that the advantages of a symbolic representation are due to two assumptions:

1. Economy of a representation that uses symbols. We think that the utilization of symbols, understood as intense descriptions, provokes a condensation of the representation and saves computational resources.
2. More human-like qualitative reasoning. We think that human reasoning of the qualitative type correlates better with certain symbolic formalisms than with others of numerical nature.

The elements that define what should be considered as a symbol in a perception-action system are based on the following considerations:

1. A symbol is a reference, indirection or pointer to an exhaustive or extended representation. This reference is more intense, and thus is of a smaller size. For example, labeling a set of aligned points as a straight line builds a more intense representation of these points. Nevertheless, this intense representation maintains the essential characteristics of the initial set of points.
2. A symbol can be manipulated independently of its extended description, using formalisms that consider it as elemental. Going on with the same example, a set of aligned points can undergo a process for parallelism detection with other sets of points, without needing to use the extended set of points, but by means of a process that uses attributes of its intense representation like its orientation.

In our opinion, the main advantage of using symbols is the efficiency achieved as a result of performing operations in the intense representation. The fourth rule summarizes these ideas:

Rule 4. In order to facilitate the construction of each level, the use of the most explicit symbolic constructions is recommended. This should deliver an easier incorporation of knowledge about the level's tasks. Also, it should result in a greater efficiency as the representation so obtained is more compact, or intense.

In this sense, the use of symbolic constructions found normally in Knowledge Engineering as rules, objects, hierarchies of objects and fuzzy operators is recommended. BU processors can hold constructions similar to classifiers and forward rules, that we shall call F rules, and must be interpreted as follows:

F Rule: $\text{Condition}_1 \wedge \dots \wedge \text{Condition}_N \rightarrow \text{Conclusion}$

where both conditions and conclusion belong to the logical or symbolic domain. Associated with the TD processor, two other types of rules are possible: forward rules related to reflex acts, here called R rules; and backward rules named B rules. These rules must be interpreted as:

R Rule: $\text{Condition}_1 \wedge \dots \wedge \text{Condition}_N \rightarrow \text{Action}$

B Rule: $\text{Objective} \rightarrow \text{Condition}_1 \wedge \dots \wedge \text{Condition}_N \wedge \text{Action}_1 \wedge \dots \wedge \text{Action}_L$

where the Actions belong to the procedure domain and the Objectives to the logic domain. Rules B are the basic elements to provide the TD processor with the capability of planning guided by objectives.

About transformations between levels.

The outlined architecture of a perception-action machine is clearly conceived around the symbol and its meaning in each level as an elemental attribute that can be manipulated with independence of its intense description. The organization of a system of this type into multiple levels, as advocated by the methodology, poses the hard problem of finding the transformation that will serve to derive the entities of one level from the entities of the previous level. A situation

of this type is found, for example, in a vision system made up of two levels that use pixels and aggregations of pixels (that is, segments), respectively as the entities to be described symbolically at each level. A transformation of the entities at the pixel level (pixels), is needed to obtain the entities that are used at the next level, the segment level. Obviously, this transformation must rely on the symbolic description obtained at the pixel level.

Generally, there is not a pre-established way to define the transformation between two levels. Nevertheless, this transformation should observe two premises. First, it must consider the derivation of the entities of a level from a fuzzy tessellation of the symbolization space of the previous level. Second, the new entities so obtained must represent a condensation of the description, both in data volume or number of entities and in symbolic content. These considerations are summarized in a final rule:

Rule 5. The organization into levels of a perception-action system must be done with special consideration to the transformation that the representation undergoes when passing from one level to the next. This transformation should establish an explicit relationship between the symbols at both levels and allow for obtaining the upper level entities by means of a condensation of the lower level entities based on their symbolization.

3. LEVELS OF ORGANIZATION AND SYSTEM ARCHITECTURE.

The proposed methodology has been applied in designing a multilevel knowledge-based vision system (SVEX). Our proposal for defining the levels that integrate a computer vision system is based on considering the entities that have significant attributes for describing the elements of a scene. These entities are the basic elements (tokens) or information units that can be described symbolically within a level. Accordingly, SVEX is made up of three levels, as illustrated in Figure 3, each of them related respectively to:

- A) **Pixels** or elementary information units in an image.
- B) **Segments** or aggregations of pixels, somewhere referred also as regions or areas, which verify some uniformity definition in their properties.
- C) **Objects** or relational structures, units built from the evidence of possible spatial aggregations of segments.

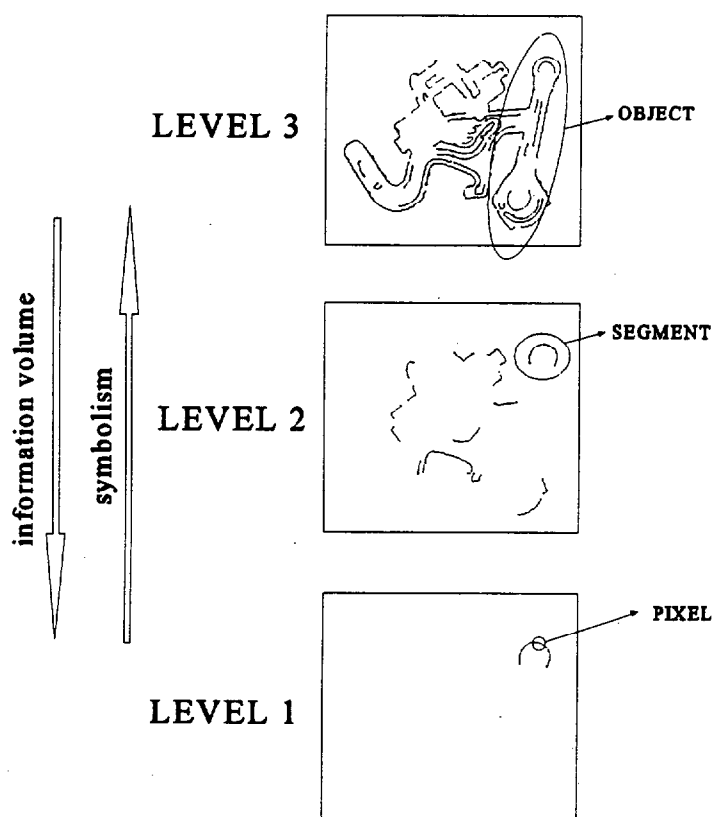


Figure 3. Levels definition in SVEX.

In this computational structure, the symbolic computations needed at each level in the process of assignment to classes can be associated with a virtual "processor", which acts as a dedicated processor. Depending on the nature of the units of information, or tokens, used at each level we will speak of three different processors: Pixel, Segment and Relational or Object processors.

The first of these processors produces a set of diagnostics or memberships of the pixels in certain pixel classes. These diagnostics offer cues for the confirmation of segmentation hypotheses [Ravi-88]. The results produced by the Pixel Processor are diagnostic maps where every pixel has a degree of membership in a pixel class. The evaluation of this degree of membership is performed by a process of conversion of numeric into symbolic features using classifiers (see Figure 5). In the new symbolic domain it is possible to define new classes as instances of previously defined classes or as the result of class combinations by means of rules. At this level, segmentation is straightforward, defining areas of uniformity as aggregations of spatially connected pixels that have a similar assignment to a pixel class. Sometimes, this output is all that is needed in applications such as those of inspection or quality control.

The output of the Segment Processor consists of pixel aggregations or segments, defined by shape factors or property uniformity, along with the corresponding assignment to segment classes, their spatial localizers and a region adjacency graph. At this level, the representation of these tokens is managed in both a numeric and a symbolic way, and the processes that take place use both representations. The output at this level is enough for applications that need only to detect and locate simple forms in a scene in the sense of Niemann's postulates [Niem-90a].

Finally, in applications where the detection and location of complex forms are in order, the Object Processor must be used. This basically does a matching between the graph built from the segments supplied by the Segment Processor and those representing the object models looking for possible correspondences [Hern-89]. At this level, the managed entities are complex forms defined by spatially-ordered aggregations of the simpler forms provided by the Segment Processor. In this case, the computation is symbolic in essence.

The experimental development of this work is restricted to the ambit of problems related to segmentation of images and considers only the first two levels mentioned above: the pixel and the segment levels. The design objectives for these levels have been the following:

- Define a reduced set of objects and operators to allow for the elaboration of programs capable of solving segmentation problems at the pixel and segment levels.
- Design a system that clearly identifies data dependencies and where the data flow follows a defined model.
- Define a working environment where it is possible to combine high-level processes, based on symbolic elements as hierarchies of classes and rules, and the management of fuzzy logic concepts, with low-level processes devoted to numerical computations.

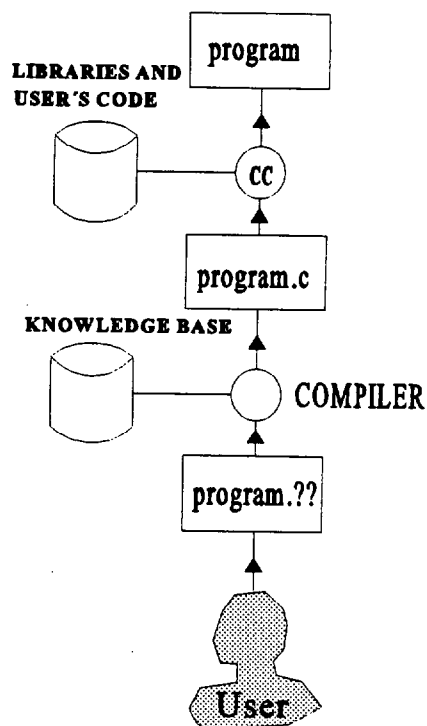


Figure 4. Process for creating an application.

- Define an invariant architecture, such that a change in the segmentation problem to solve should only be reflected in variations in the programming of the system, leaving the structure of the processes unchanged.
- Define a programming language specific to the proposed architecture and the corresponding compiler. The language should be of declarative type to make as explicit as possible the knowledge that is used for solving a segmentation problem.

The generic process of building a program, also sometimes referred to as an application, is schematized in Figure 4. The user utilizes this language to define the program and can include other files written by vision experts to build an executable program. This is done in two steps. First, the language's compiler translates the program and the included code from the knowledge base into C language structures. Then, a general purpose C compiler compiles these structures

and links it with procedure libraries and other user-specific modules. In this way, a change in the application only requires changing the program as expressed by the language. The structure of the processes and the system architecture remain invariant.

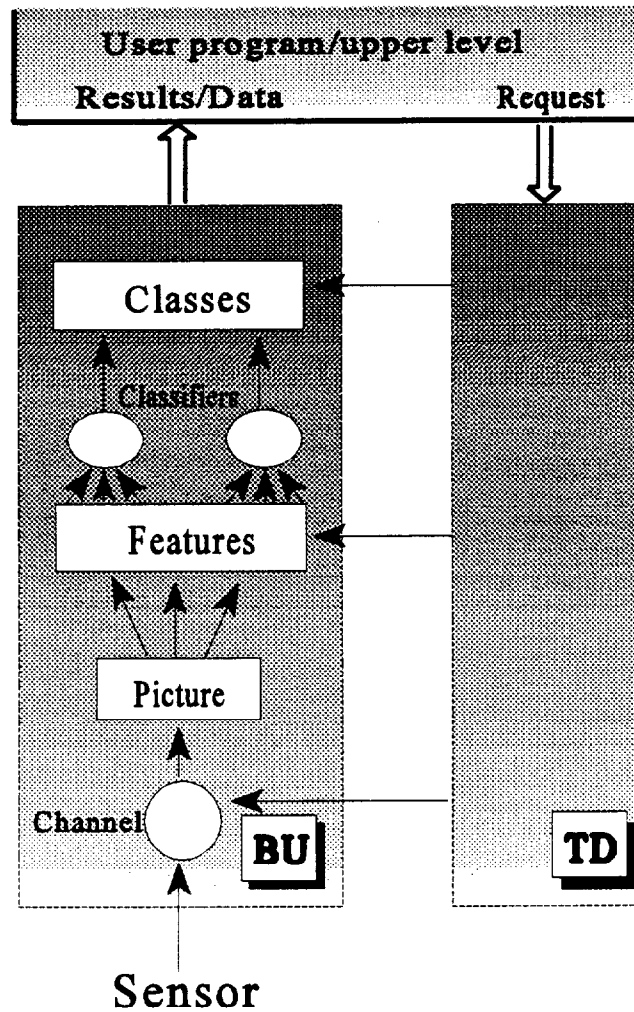


Figure 5. BU-TD Processors and domains at the pixel level.

3.1 AT THE PIXEL LEVEL: Pixel Processor architecture and definition language.

In the next section, we will describe how the methodological "rules" have been used to design the first, or pixel, level of SVEX. The initial requirements imposed upon the BU and TD processors at this level are the following:

1. The BU processor receives images as input data from camera sensors and produces symbolic diagnostic images as output data. It will use explicit knowledge about the models of class diagnosis.
2. The TD processor receives requests for computing pixel diagnoses. This processor transforms these requests into execution orders addressed to the BU processor and commands for camera controllers.

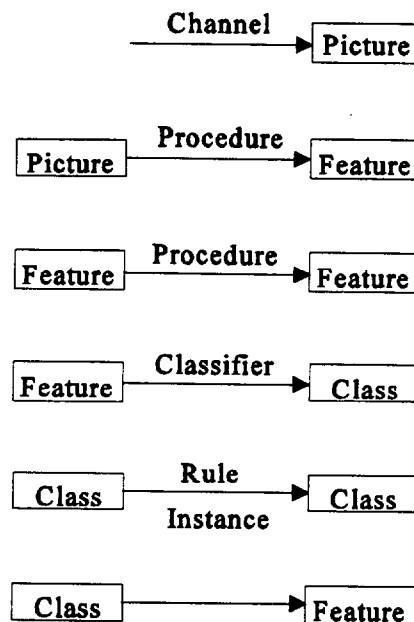


Figure 6. Image types and their possible transformations by means of operators.

At the pixel level, the BU processor uses two representations. The first is numerical, and takes the form of feature maps (gradient, color, variance, ...) obtained initially by image processing algorithms from sensor data. The second representation manages symbolic maps or symbol images that are generated from feature maps by symbolization processes. In this work, the value of a pixel in a symbolic map represents the membership degree of that pixel to a certain pixel class, such as *Green Class* or *High Texture Class* [Wils-88a]. With these elements it is possible to perform symbolic computation based on class hierarchies and interclass dependencies expressed

in the form of rules. The use of class memberships makes possible the use of fuzzy classifiers and certainty factors in rules. The control of the uncertainty associated with the computed pixel diagnosis is one of the most important tasks of the BU processor. The uncertainty is transmitted from classes, classifiers and rules to other classes following a process described in chapter II.

As it has been stated, the Pixel Processor operation is structured over two domains. The first, the feature domain, utilizes feature images and operators [Hara-91] called *procedures*. The procedures are applied over images or other features to generate new features. The second domain is the class domain where a set of operators (classifiers, rules and hierarchies) generates class images (class membership maps) from features or other class images. Figure 5 shows the Pixel Processor's architecture detailing the domains used in the BU processor. The image types and the operators that can transform them are depicted in Figure 6.

To simplify the usage of the Pixel Processor from upper levels, its control is goal-oriented. The computations are triggered when the TD processor receives a request for computing one or more pixel classes and transforms it into a sequence of orders that are directed to the BU processor. These orders will be in correspondence with the program designed for computing a requested pixel class. The program is written in a special-purpose declarative language using the following reduced set of objects: *Channel, Picture, Procedure, Feature, Classifier, Class, Rule and Interface*. The "Picture" object identifies the input image. The "Feature" objects are image features obtained from the image or from other features by the application of "Procedures". The "Class" objects are symbolic images that can be combined using "Rule" objects to define logical conditions. Finally, The "Interface" object contains the names of the classes that are accessible from upper levels, indeed the classes the Pixel Processor can be asked for. These objects are explained in greater detail in chapter II.

3.2. AT THE SEGMENTS LEVEL: Segment Processor's architecture and definition language.

The second level in this system is conceived around the Segment Processor whose basic goal is the **partition** of the image under analysis into a set of pixel aggregations or **segments**

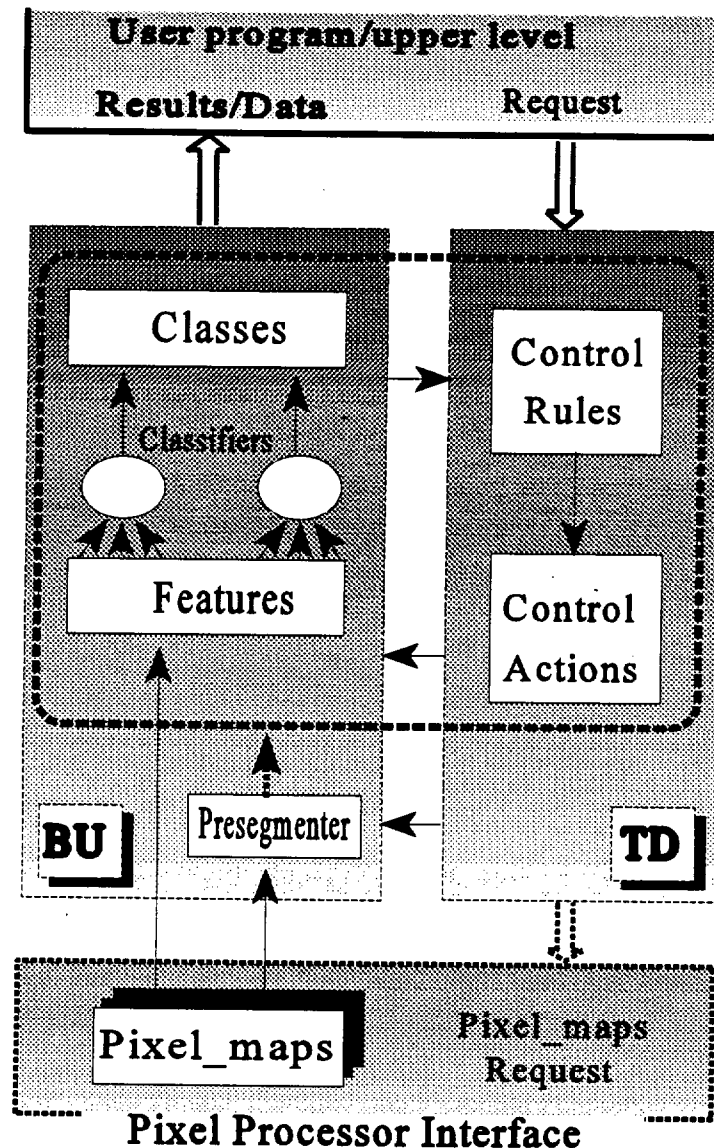


Figure 7. BU-TD processors and domains at the segment level.

qualified by their membership degrees to symbolic classes of segments ("red-square", "straight-line", ...). We consider the **segment** as a connected aggregation of pixels defined by shape and/or property criteria. Within the Segment Processor's context, a **partition** of an image is a list of segments that constitute a complete image tessellation, along with their corresponding assignment to symbolic segment classes, their spatial localizers, and the spatial relations among neighbor segments. The use of segments as the information grain in this level poses two problems that were not present at the pixel level. First, it is necessary to obtain the spatial definitions of the segments from the pixel maps requested from the Pixel Processor; that is, to define this level's entities from

the symbols of the next lower level, prior to their symbolic characterization in terms of segment classes. The second problem is that, while the Pixel Processor operates with predefined entities of stationary spatial limits, (pixels), at the Segment Processor level, it should be possible to allow entities to undergo changes in their spatial definition (merging or splitting) as a result of refinement processes acting over the segmentation.

The architecture of this level is conceptually identical to that of the Pixel Processor (Rule 2) already described, based on the BU and TD units. However, the aforementioned requirements introduced some oddities peculiar to this level in the functionality of BU and TD units. Their specifications at the segment level are the following:

1. The BU processor receives as input data pixel diagnostic maps, or pixel classes, previously requested by the TD processor from the Pixel Processor; and produces symbolic diagnostics for the Partition's segments as output data. Its action has two different phases: acquisition of the entities through the definition of the initial partition, and the symbolic description of the segments obtained, in terms of segment classes.
2. The TD processor receives diagnostic requests about segments, and transforms these requests into sequences of commands for the BU processor to compute the diagnoses, and in requests for pixel diagnoses directed to the Pixel Processor. The TD processor can be programmed using control rules to execute actions that modify the image partition. The premises of these rules are established in terms of segment diagnoses computed by the BU processor and may include conditions about spatial relations among neighbor segments.

From the point of view of functionality, the Segment Processor's architecture (Figure 7) is organized in three blocks, each one addressing a specific objective. The first of these blocks is charged with the obtainment of the initial partition. This is the objective of the presegmenter module that is part of the BU processor. Different methods can be used to achieve this goal so the presegmenter module has been conceived as an interchangeable part within the Segment Processor. The second functional block has the goal of computing the segment diagnoses. This

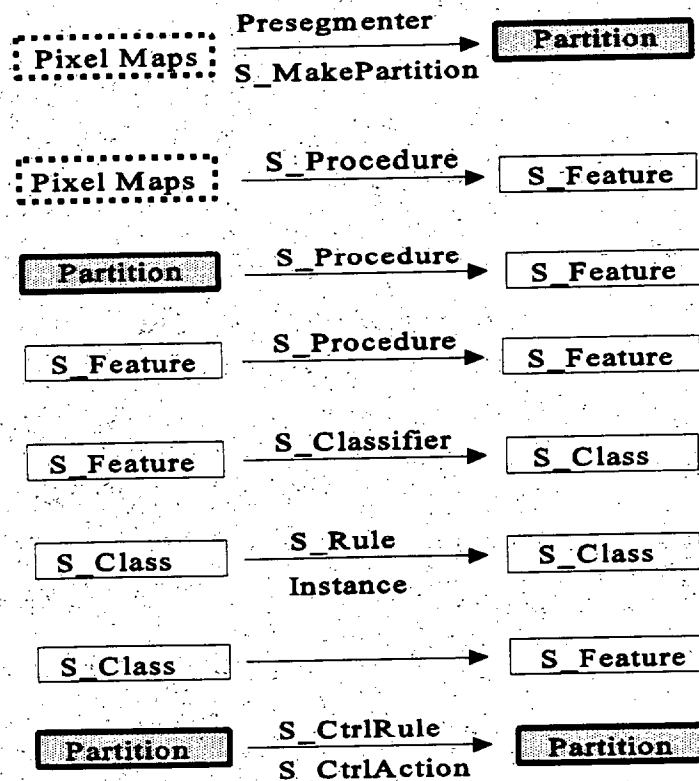


Figure 8. Data types and their possible transformations by means of operators.

block is structured by the distinction between a numerical domain, where the segments are described by features, and another symbolic domain based on classes, a replica of the scheme used with the Pixel Processor with only slight, though important, differences, like the utilization in the Segment Processor of conditions describing spatial relations among neighboring segments. The third functional block is located at the TD processor and manages the diagnostics requests and controls the Segment Processor. This control includes evaluating the partition's state from a set of rules that may trigger actions to modify the spatial definition of the segments.

The Segment Processor is programmed in a fashion similar to the Pixel Processor using a new set of objects that reflects the aforementioned differences. There exists a small subset of objects that are used with the presegmenter and whose exact characteristics will depend upon the selected presegmenter module. Independently of the presegmenter selected, the "S_MakePartition" object defines the TD processor's control over the action of the presegmenter.

Extended Abstract

Other objects like "S_Feature", "S_Procedure", "S_Classifier", "S_Class" and "S_Rule" are equivalent to their homonyms in the Pixel Processor, the only difference being that they are now applied to segments instead of pixels. The most important differences are the introduction of the "S_Condition" object to express premises and the possibility of computing features ("S_Feature") from a larger set of sources. In this case, once the first partition has been obtained, a segment's feature can be computed from the pixels (Pixel_Maps) included in the segment, from other features (S_Features), from a class (S_Class) that is transformed into a feature or directly from the partition if the feature depends on the spatial definition of the segment, such as shape, size, etc. Related to the performance of the TD processor, the "S_CtrlRuleSet", "S_CtrlRule" and "S_CtrlAction" make possible the control of the course of building a final partition. Finally, the "S_Interface" is used to declare the segment classes that will be visible to an upper level; and the "P_Interface" object indicates the pixel classes that are to be requested from the Pixel Processor. Figure 8 shows the various data types (inside boxes) and their transformations by the operators located above the arrows. The objects placed inside boxes of continuous border and white background represents descriptors of numerical or symbolic nature of every partition segment. Data in dotted boxes are pixel maps obtained from the Pixel Processor and used to define the initial partition or to compute segment features. The partition can be envisioned as another data type that can be updated by means of control rules and their associated actions.

4. SOME ASPECTS RELATED TO THE IMPLEMENTATION OF SVEX.

Chapter III explains some elements of SVEX's implementation that deserve a more detailed description, as they represent important contributions arising from this work. One of them is the Watershed Transform and the way it is used by the Segment Processor to define a primary partition of an image from several diagnostic maps produced by the Pixel Processor. This is an important part of the system as it is the method used to define segments from pixels. Another major contribution of this work is a Connected Component Analysis Algorithm (CCAA) used to obtain detailed topological information from the partition of the image. This CCAA is an extension of the algorithm presented in [Mand-90] and produces for each connected component in the partition the code of its external border and possible internal borders, an ordered list of the

neighbors as found during a walk along each border, and a list of the pixels making up the component. A second part of this chapter outlines the process of programming SVEX and presents some tools developed as programming aids. These include an iconic editor for editing programs; a tool for collecting "samples", such as pixels or segments that belong to a certain symbolic class, which is used to discover properties of that class; and other utilities that are used as program "tracers" at the pixel and segment levels.

4.1 PRESEGMENTER OPERATING PRINCIPLES.

Different segmentation methods can be used to achieve the presegmenter's goals and no one is definitively superior to the others. Consequently, the presegmenter module has been conceived as an interchangeable part of the Segment Processor so that different applications can employ different segmentation strategies. In this work, we describe the presegmenter module named "Flood", used with all the experiments, which is based on a region growing method called the Watershed transform. The features of this method make it fit naturally within the frame set by the Pixel and Segment Processors. As we will explain below, different diagnostic maps can be used to produce a segmentation of an image and the types of objects expected in the image can be characterized directly in terms of the requested diagnoses.

The Watershed transform.

The presegmenter uses a region-growing algorithm to build a color map or primary segmentation of the image. It is inspired by the Watershed transform, a tool developed in the field of mathematical morphology and first introduced by H. Digabel and C. LantuÇjoul [Diga-78] for processing of binary images. It was extended to grayscale images and applied to segmentation problems by S. Beucher and C. LantuÇjoul [Beuc-79]. The Watershed transform can be easily explained using a simile based on an immersion process of a topographic surface as is presented in [Beuc-90] and in [Vinc-91]. Let I be a digital grayscale image and let F be also a digital grayscale image representing the topography of I . For example, F can be the absolute value of I 's gradient. In this way, homogeneous regions in I will be mapped into plateaus, or **minima** of F , surrounded by higher elevation points (see Figure 9) which form closed paths. These closed paths, which separate different minima, are called watershed lines or simply **watersheds**; clearly, their

Extended Abstract

positions correspond to the contours present in the original image I. The area delimited by each watershed and containing exactly one minimum defines the **catchment basin** of that minimum. The purpose of the algorithm doing the Watershed transform is to tessellate the image into these catchment basins, corresponding to almost homogeneous regions in the original image I.

The same simile may help in understanding how the algorithm proceeds. Figure that the minima of the topographic surface have been pierced and the surface is slowly immersed into a lake. Coming out from the lowest altitude minima, water will progressively start filling the catchment basins of these minima. A restriction is imposed that waters coming from different minima can't

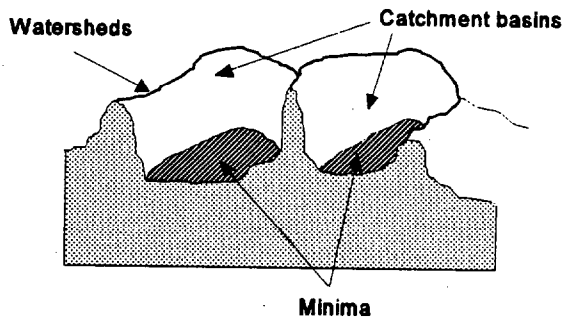


Figure 9. Minima, catchment basins and watersheds.

merge; so, at each pixel where this mix would take place a 'dam' is built. At the end of this immersion process, each minimum is surrounded by dams, which delimit the associated catchment basin. These dams correspond to the watersheds of F.

The Watershed transform as it has been described until now does not work properly when applied to image segmentation [Beuc-90][Vinc-91], as an oversegmentation is produced. This problem is caused by the many minima (many of them irrelevant) normally found in real images. Obviously, a modification is needed to apply the Watershed transform successfully in solving segmentation problems. The classic approach is to use **markers** to select sets of connected points that will be set as the minima to be used by the Watershed transform. These markers are selected using knowledge about the nature of the images and the objective of the segmentation. To summarize, the use of the Watershed Transform to segment images comprises the following three major steps:

1. Define the image's topographic relief over which the flood will take place.
2. Employ a set of markers to select those points that correspond to the seed or prototype

points of relevant parts of the image. Use these marked points to modify the gradient of the image such that connected marked points will be the only minima.

3. Apply the Watershed transform using the modified gradient of the image. This will produce a partition of the image into regions (i.e., catchment basins) that will be taken as the primary segmentation of the image, also referred in this work as a color map.

Acquisition of the topographic profile:

The Watershed transform normally uses the gradient of the image or a similar measure as the definition of the topographic profile. In our case, we may need one or more diagnostic maps to segment an image. So we may have several maps to define the relief of the topographic surface. To provide the system with a reasonable degree of flexibility, we may allow the user to build the relief of the surface by means of a linear combination of the profiles obtained from different pixel maps. The user can define these 'profile components' using different gradient operators over the pixel maps. In this way, it is possible, for example, to build the profile of the topographic surface by combining the profiles obtained from one or more pixel maps using different operators. Also, the user can weigh the contribution of the pixel maps' profiles by adjusting their weights in the definition of the topographic profile.

Point labeling: selection of markers.

As was mentioned above, the application of the Watershed transform to image segmentation requires the selection of sets of connected points using markers. Every class of segment expected in the initial partition must have associated a marker to "mark" its associated minima. Classes of segments are named **prototypes** in the framework of the presegmenter and must be defined by the user in terms of the pixel maps requested from the Pixel Processor. This set of pixel maps constitutes a representation space where each pixel map is assigned a dimension. Prototypes are defined in this space, giving a definition of the hypervolume that contains the samples of each prototype. In general, prototype k will be defined as follows:

$$HV_k = F(Pm_1, Pm_2, \dots, Pm_n, \alpha_{ki}, \beta_{kj}, \dots, \xi_{kn})$$

Extended Abstract

Where Pm_h stands for pixel map 'h', F is a function that defines the hypervolume containing the samples or prototype points of prototype K , and $\alpha_{ki}, \beta_{kj}, \dots, \xi_{kn}$, are the arguments of function F , typically centroid coordinates and dispersion of this prototype in each map. For example, prototypes can be specified in the following simple form, which has proved powerful enough during the experiments. For each prototype a 'mean' or characteristic value of the diagnosis and a tolerance or dispersion around this value are given for every pixel map used in the prototype definition, so that in this case, the hypervolume is just a hypercube.

Coming back to the selection of markers, it is obvious that markers must be based on the detection of prototype points (i.e., points enclosed by the prototype's hypervolume). Although this has been adopted as the basic idea for selection of markers, it may be complemented with other heuristics, depending on the nature of the segmentation problem. For example, these can be based on the homogeneity and/or extension of a set of connected prototype points, the 'height' of these points in the topographic relief, etc.

Objects for Presegmenter programming.

The presegmenter can be programmed using the objects that every presegmenter module incorporates into the Segment Processor object-oriented language. The "Flood" presegmenter, based on the Watershed transform, adds four objects to the language of the Segment Processor. The "Prototype" object permits the specification of the segment types expected in a primary partition of the image. Each "Prototype" object includes the definition of the corresponding prototype in terms of pixel diagnostic values, the selection of a "Marker" object to locate the minima of this prototype,

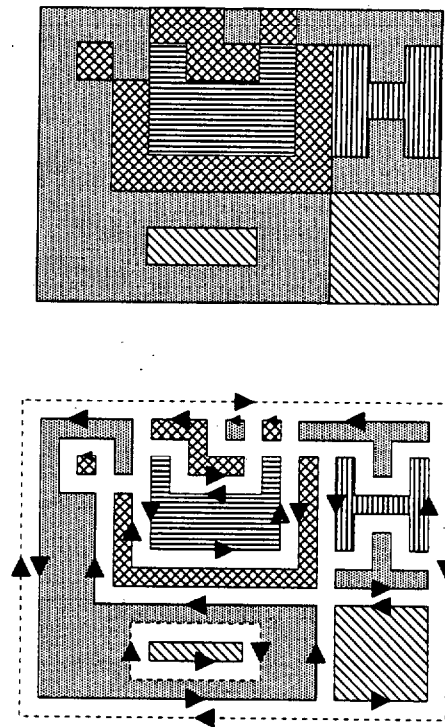


Figure 10. Connected components in a color map.

and some other details, such as the definition of connectivity to be used with this prototype (4- or 8- connectivity). "Marker" objects are used to facilitate for each prototype the localization of its minima. They normally rely on homogeneity and/or extension criteria, but the user can incorporate new "Markers" based on different criteria. There is a "Pixel_Map" object associated with each pixel or diagnostic map used by the presegmenter. Its declaration includes the selection of a "Profiler" object and a weight: these determine how the associated pixel map contributes to the attainment of the topographic relief used during the flooding process. Each "Profiler" object contains the definition of a gradient or similar operator used with a pixel map.

4.2 CONNECTED COMPONENTS ANALYSIS.

Once the presegmenter has produced a color map, it is necessary to extract from it the relational information that will be added to the color map in order to build the initial partition of the image. This is done using a procedure that performs a connected-components analysis (CCA) over the color map. The algorithm used is an adaptation of that presented in [Mand-90] with some extensions and enhancements. It extracts the connected components of a **multivalued image**, returning for each component **its border and ordered lists of all neighbors (as they are found by walking around the outer and, possibly, the inner borders), the coordinates of the component's pixels, and the minimum bounding rectangle**. This is done by scanning the color map in a single pass, using a 2x2 window.

Every component is defined in the color map by the area bounded by an external contour and a set of possible internal contours. In this way, the color map can be envisioned as a puzzle, with two contours running in opposite directions in the virtual space separating each component, as illustrated in Figure 10 and in Figure 11. Contours are given a direction such that the area of the component is always kept at the

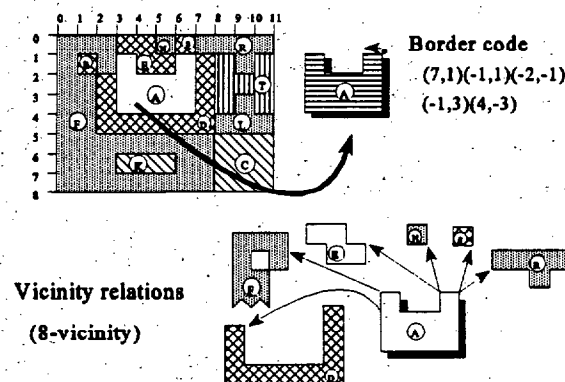


Figure 11.

Extended Abstract

left-hand side in the direction of advance. From this rule, it follows that the outer contours are tracked counterclockwise and inner contours clockwise.

The algorithm's strategy for solving the problem of finding connected components in multicolor images (i.e., color maps) has two parts: First, multicolor corners (C-corners) are decomposed into binary corners (B-corners), by scanning the image using a 2x2 window. Then, B-corners found in a line scan are linked by pairs. This permits the performance of a connected-component analysis during a sequential raster scan. Table I shows the possible combinations of B-corner pairs and how they can be decomposed into pairs of binary corners.

2x2 windows	B-Corners	B-Corner pair

Table I. List of possible binary corner pairs.

When we deal with multicolor images, up to four different colors can appear in a 2x2 window. Table II depicts the possible C-corners. Colored corners can be classified into five groups depending on the topology of the corner. The first column in Table II contains the names of these groups, by which they will be designated hereafter. The different types of X corners are not discriminated by the original algorithm and they all are mapped as X4 corners because the CCA is based on 4-connectivity instead of 8-connectivity. So, only the mapping of X2 and X3 corners, as depicted in Table II, is used by our version of this algorithm, as will be detailed below.

Topological corner classification	Colored corners		Upper Contour		Lower Contour	
	window	CC-type	window	BC-type	window	BC-type
L2 Corners						
L Corners two colors						
T3 Corners						
T Corners three colors						
X4 Corners						
X3 Corners three colors						
X2 Corners two colors						

Table II. Colored corners: topological classes and decomposition into binary corners.

The algorithm's major steps are the following:

I. *Corner detection.*

Scan the color map line by line using a 2x2 window. From left to right, each C-corner is detected and identified. Its type, position (column), and color (as defined by the color of the lowest left pixel), are registered to be used in the second phase.

II. *Corner processing.*

1. C-corners are decomposed into binary corners: one B-corner for every component (i.e., color) that appears in the C-corner.
2. Vicinity relations are registered in precise order.
3. B-corners are combined by pairs, defining in this way the basic contour elements.
4. Open contours are manipulated as a result of processing corner pairs. The possible operations with borders are: opening, closing, shifting and merging.

Extended Abstract

A tricky part of the algorithm is how vicinity relations are recorded, because when these relations are defined, the final identity of the contours is unknown. For example, a neighbor contour can disappear if it is merged into another contour. This and other details are explained in detail in chapter III.

In the context of this work, situations may arise where it is important to consider extending the capabilities of the described algorithm to deal with diagonal connectivity and diagonal vicinity. This is the case when the image may contain one-pixel wide components (such as contours), together with other more complex components. The algorithm in its original version uses 4-connectivity so that an 8-connected component will be split into 4-connected portions and, even worse, these portions will not be related as neighbors. In the next section, we will present our attempt to extend the capabilities of the original algorithm to consider 8-connectivity. Clearly the starting point is a more careful analysis of X corners, since possible diagonal connections will originate from them, precisely from X3 and X2 corners. The connection of diagonal pixels with the same color in X3 corners does not pose any problem and the algorithm can easily be modified to accomplish this task. In contrast, X2 corners present two possible and equivalent diagonal connections, and some extra processing is required to solve for the correct diagonal connection.

The extension of this CCA algorithm to include 8-connectivity and 8-vicinity poses two major difficulties that can be summarized as follows:

1. When an X2 corner is encountered, it is not possible to resolve the two possible diagonal connections exclusively from the topological information gathered during the scan of the image. In its original formulation, the algorithm does not set any diagonal connection, since 4-connectivity is used, and this type of corner is processed as an X4 corner. An extension of the CCA algorithm to deal with 8-connectivity must be capable of selecting one direction for the diagonal connection. (Incidentally, it should be noted that the use of a larger window does not solve this question at all.) The solution proposed for binary images [Pavl-82] is to consider the background as 4-connected and the

foreground or object as 8-connected. Although clearly this solution is not applicable to multicolor images, it offers some cues for a possible solution.

2. Although not so problematic, the extension of the algorithm to relate as neighbors pixels that are 8-connected may give rise to self-vicinity problems. The problem of self-vicinity appears when 8-connected pixels belonging to the same component are related as neighbors. An example of a self-vicinity occurrence is shown in Figure 12. In this example, the component with B color would be a

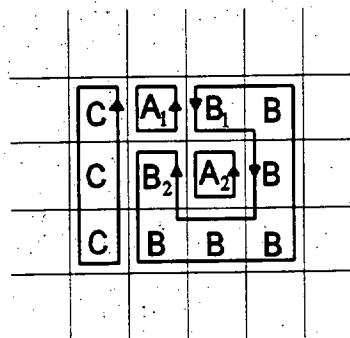


Figure 12. Example of self-vicinity.

neighbor of itself across the two A-colored components, because when this relation was set both B-colored pixels were not part of the same component. Within the framework of this algorithm, it is very simple to correct for these anomalous vicinity relations during the closing of a contour by checking the identity of the neighbors.

The extension of the algorithm to deal with 8-connectivity and 8-vicinity must solve both of these problems. In the next section, we will briefly describe two different strategies to solve the diagonal connections in ambiguous situations.

The first possible solution, let's call it A_strategy, for the problem of diagonal connectivity uses only information extracted from the image, and it is based on the following observation: Very often, the extension of 4-connectivity to 8-connectivity is necessary because an image contains 'contour-like' components, and these are normally included in other, more extensive, components. (Note that even in an image containing only 'contour-like' components they would be included in the background component.) When this situation occurs, it is usually possible to solve an ambiguous diagonal connection in a certain direction using a heuristic strategy: If the contours running upwards and downwards at the position of the ambiguous corner are of the same type (i.e., outer), it is based upon the type of last corner detected in the same line; or, if the contours are of different type, the diagonal connection is set between the pixels having the same color as

the outer contour. When both contours are of the same type and the type of last corner detected is not included in the case considered, the X2 corner is processed as an X4 corner, as done by the original algorithm, so that no diagonal connection is set. Clearly, this strategy is not a general solution for this problem, but we have found that it suffices to solve most of the situations encountered, **using only information extracted from the image.**

The second approach to solving the same problem, which we will call B_strategy, can be used when the color assignment in the color map follows a semantic scheme. This solution is an extension of that adopted for binary images when the background and foreground colors are known. The solution is very simple and consists in setting foreground-background relations for every possible pair of colors, such that pixels of 'foreground' color would be diagonally connected if an X2 corner of 'foreground' and 'background' colors is found. In this way, the algorithm can resolve all the possible diagonal connections. Our system makes use of this possibility to extend the original algorithm. For the sake of flexibility, the user can instruct the CCA algorithm which definition of connectivity (8-connectivity or 4-connectivity) to use, both for connectivity analysis and for vicinity analysis. It is possible, for example, to use 4-connectivity to extract components but 8-connectivity to get the neighbor components. If 8-connectivity is used for component extraction, the use of B_strategy fits naturally in our system, since the color of the components has a semantic definition in terms of prototypes as the result of presegmenter action. The user can specify a precedence list, in terms of prototype names, from which 'foreground-background' pairs will be built. This list need not be exhaustive; those pairs not specified in the precedence list will be resolved as X4 corners.

The described CCA algorithm has been compared with the connected-components labeling algorithm presented in [Yang-92]. This algorithm is, as far as we know [Lumi-83a][Caps-84][Same-86], the most efficient non-parallel algorithm for connected-components labeling (CCL), and, furthermore, it is well suited for hardware implementation. However, the comparison is hard to perform, for several reasons. First, we think a distinction between CCA and CCL algorithms must be made, since CCL algorithms produce only a small fraction of the results offered by a CCA algorithm. CCA algorithms produce not only a color map, but also extract

detailed topological information about the component and its vicinity relations. In contrast, the only objective of most CCL algorithms is the acquisition of a color map containing the labeled components. Most of the algorithms described in the literature are CCL algorithms [Lumi-83a][Same-86], that must apply, *a posteriori*, some sort of border following over the produced color map to extract the borders of a component and locate its vicinity points with neighbor components. The second question that makes this comparison difficult is that most CCL algorithms use 4-connectivity and have been conceived to deal only with binary images. Surprisingly, multicolor component labeling has rarely been studied [Dani-82] [Mand-90]. In view of this situation, it is clear that the comparison between the CCL algorithm proposed in [Yang-92] and the described CCA algorithm is not precise, and must be considered as a worst-case analysis from the point of view of our algorithm. Several comparisons were carried out using sets of 256x256 and 512x512 binary images, constructed so as to have different numbers of connected components and corners. In spite of the former observations, the CCA algorithm was shown to be faster when the number of components (corners) in the image was less than approximately 2000 (10000). Images manipulated with CCA algorithms are normally much less complex than this, so the proposed algorithm may be of great interest and have a wide application.

5. SVEX'S EXPERIMENTAL RESULTS.

The final part of this thesis is devoted to the presentation of the results achieved using SVEX in different segmentation problems used as study cases. Some of the experiments are solved at the pixel level while others require processing at the segment level. Defect detection in leather pieces, moving part detection, and outdoor scene segmentation are the experiments described at the pixel level. At the segment level, the experiments include situations of quality-control in biscuit manufacture and glass bottle reception in a bottling plant. Also at the segment level, there are some experiments on contour detection and segmentation into types of contour pieces, multiconnected particle segmentation, and detection and identification of industrial pieces.

6. CONCLUSIONS, MAIN CONTRIBUTIONS AND FUTURE PROPOSALS.

The most significant contributions of this work fall into two distinct categories; on one hand, the theoretical basis and methodology used provide a computational structure for Image

Understanding Systems characterized by its flexibility, on the other hand, the tools constructed to implement this structure provide elements of immediate practical application. In the following sections, we present the conclusions and principal contributions obtained during this research. Thereafter, we propose some further extensions and improvements of the image-segmentation system (SVEX) which we have developed.

6.1 CONCLUSIONS AND PRINCIPAL CONTRIBUTIONS.

1. In order to guide the progress of this work, we performed a structured, comprehensive review of the problem of image segmentation from three different perspectives. First, we reviewed the most prevalent segmentation techniques, seen as 'basic tools' or fundamental resources. Then we examined various expert systems for image analysis which have contributed to the development of more robust and independent methods of segmentation and processing, with the intention of organizing and condensing the existing heuristic knowledge represented by these methods. In the final phase of the review, we have considered the segmentation process in the context of Image Understanding Systems (IUS). We have described the best-known IUS's in terms of their architecture, their representation of the knowledge base, and the organization of their control systems. As a result of this review, we have developed a series of guiding design principles which experience has shown to be useful in any IUS, and we have summarized the central problems to be resolved in its design.

2. We have proposed a basic methodology for the development of perception-action systems expressed in a set of 'Rules', or design principles. This methodology is based upon the concept of a multilevel system, following the paradigm of a data-processing machine evolving in a 'virtual world'. The architectural model we propose for each level is based upon BU-TD units, formed by a Bottom-Up (BU) processor devoted to diagnosis or abstraction, and a Top-Down (TD) processor, acting as a control and planning element. The use of BU-TD units in the various levels of the system facilitates the design of different models of perception-action machines, at the same time that it establishes a basic framework for the distribution of the work of computation and control.

3. The presence of the TD processor allows for the elaboration of control strategies based not only upon the evolution of the description generated by the BU processor, but also upon knowledge generated at a higher level of the system. In the first case, it is possible to develop a type of control called 'reflex acts', which permit the system to react upon its virtual world independently of orders received from higher levels. The second control scheme, based upon orders or requests received from higher levels, makes possible a complementary mechanism for the use of knowledge from higher levels to guide the work of lower-level units.
4. We have evaluated the proposed methodology *a posteriori*, following is application to the design of specific perception-action systems; that is, the methodology is considered valid when, on the average, its use has produced more robust, efficient systems, or has facilitated their construction. In this sense, the knowledge-based vision system (SVEX) developed within the framework of this thesis may be considered as a first experiment whose results seem to validate the usefulness of the methodology employed.
5. One of the benefits of applying this design methodology to SVEX has been the production of a clearly-defined computational structure, applied at both the pixel and segment levels. This structure includes a general mechanism for the integration of multiple data sources (different sensors, numeric or symbolic characteristics, etc.) and a model for the control of uncertainty in the symbolic domain. The computational structure we have developed permits symbolic calculations at any level, including the lowest level. This represents a substantial difference compared to more classical designs, where the terms "high level" versus "low level" are considered equal to "symbolic" versus "numeric" computation.
6. We have developed a declarative programming language that facilitates the explicit expression of the knowledge necessary for the operation of SVEX. Its characteristics permit the rapid and efficient adaptation of SVEX to new tasks or applications. Specifically, we have determined that the use of this language has reduced significantly the time required to develop new applications, when compared to that spent in developing the same application using a general-purpose language. The language uses a reduced set of data types and operators, and integrates low-level

Extended Abstract

numeric processes with high-level constructions such as rules and hierarchies of classes, in addition to permitting the management of uncertainty.

7. The levels contained in SVEX are based upon elements that are situated close to the observer (pixels, segments, objects, ...), which establishes a clear semantic base and facilitates the definition of the set of operators at each level. We emphasize that a similar principle of clarification of the elements to be processed at each level has been used in the development of specific architectures, such as the IUA [Weem-89] [Weem-92]. The organization of SVEX based upon this scheme is modular and scalable. Thus it is possible to solve vision tasks that require pixel-level diagnostics using only the Pixel-level Processor, or the Pixel and Segment Processors when the task requires the localization of simple forms, and so on.

8. By design, the two levels present in the implementation of SVEX described in this work may attend to requests coming from the next higher level or from an application program. Thus it is possible within SVEX to combine, in a single request to a Segment Processor, directions to several Pixel Processors, each one calculating different diagnostics from a single image. This permits the effective distribution of the computation over the various levels of the system, and facilitates the integration of SVEX into other more general systems or specific applications.

9. The transformation of the representation of the image, necessary for a change in level, has been identified as a fundamental problem to be resolved in multilevel systems. In SVEX, this is the mission of the presegmentation module of the Segment Processor, charged with the generation of a first partition of the image using the diagnostic maps produced by the Pixel Processor. Since there is no known universally-valid segmentation technique, applicable in any context, the presegmentation module has been conceived as an interchangeable part of the Segment Processor. Thus, different presegmentation modules may be flexibly used, depending upon the topology of the image, and the goal of the segmentation.

10. The Watershed transform has served as the inspiration for a versatile and flexible presegmentation module ("Flood"), which has proven valid for producing the segmentation of an

image using multiple diagnostic maps. This presegmentation module in particular, and the general method upon which it is based, constitute a potent segmentation tool of broad applicability.

11. We have produced a rapid and adaptable algorithm for Connected Component Analysis (CCA). For each connected component in a multicolor map, the proposed algorithm extracts the external border, and possible internal borders, ordered lists of all neighbors (as they are found by walking around each border), and the minimum bounding rectangle. This information may also include lists of the constituent pixels of each component. Furthermore, the algorithm permits the selection of 8- or 4-connectivity for the analysis of vicinity.

12. This algorithm has been compared with the connected-components labeling (CCL) algorithm presented in [Yang-92], which may be considered one of the most efficient non-parallel algorithms, based on the results published [Dani-82][Lumi-83a][Caps-84][Same-86]. We have concluded on the basis of our studies that the proposed CCA is faster up to a reasonably-high number of corners (about 10000) and components (approximately 2000), in images of 512 x 512 pixels in size. The CCL is only applicable to binary images, and generates a label map containing only 4-connected objects. This result is not comparable to that obtained with the proposed CCA, since it lacks explicit information: component borders, vicinity information, etc. In this sense, the comparison represents a worst-case analysis from the standpoint of the proposed algorithm. Thus we consider that the CCA proposed in this work represents an important contribution, with a wide range of applicability.

13. We have designed and implemented an initial set of programming-support tools which make of SVEX a system for the development of applications in computer vision. The most important members of this toolbox are:

- A Sampler, which permits the collection of samples belonging to a qualified class, at pixel or segment level, forming the first element of a learning system.
- An iconic editor that assists in the rapid writing of programs free from typographic or syntactic errors.
- A tool for the visualization and analysis of maps of characteristics and diagnostics at the level

of the Pixel Processor (ViewMap), and another tool of similar characteristics at the segmental level, integrated in the Segment Processor, which permits program tracing at this level (Xdebug).

14. The basic objective of these experiments has been to demonstrate a non-traditional approach to the resolution of the problem of image segmentation, using a variety of relatively simple experiments. The use of the computation scheme imposed by the concept of pixel- and segment-processors creates at times a certain lack of freedom compared to the traditional method of programming vision systems using a general-purpose language. However, this lack of freedom is offset by the definition of a more structured way of confronting the problem of Computer Vision, which puts the emphasis on the organization of the useful knowledge base, and places all the rest in a secondary role.

15. Concerning the progress of the experiments, we have arrived at three conclusions: First, we have established the capacity of the model to resolve a variety of real problems of image segmentation, using relatively simple methods of general utility. Thus we have resolved problems of segmentation using a heterogeneous set of images (grayscale, color, or sequential), in different contexts (quality control, industrial applications, landscape images, etc.). Some of the cases studied have suggested the desirability of incorporating in SVEX various methods of feedback between the pixel and segment levels. Second, SVEX and its associated set of tools should be viewed as an environment for the development of applications in Computer Vision that delivers results at a lower cost of development than that seen in a general programming environment. Finally, we have demonstrated the existence of an important bottleneck in the acquisition of the knowledge base that highlights the need for a learning tool to automate, among other things, the selection of the most relevant image features, and the symbolic definition of classes based upon them.

16. The experiments have also shown certain aspects of SVEX which, although not fully developed in the present implementation, are potentially very important. One of these is the possibility of defining a set of primitive or elemental concepts, not only at the pixel level (to be "red", to be "sky", to be "foliage", ...), but also at segment level (to be "square", to be

"straight",....). These elemental concepts form the model base at each level, and may be combined into definitions of new concepts or more complex classes in different contexts. We have also demonstrated the possibility in SVEX of organizing the interpretation using a scheme where we progress from the most obvious to the most uncertain, or from the most general to the most detailed, emitting certain diagnostics as soon as a hypothesis concerning a zone of the image has been confirmed. This permits the breakdown of complex analyses into successive stages, using the knowledge base to direct and focus the analysis. To strengthen these aspects of the process will require improvement of the implementation related to the system's control scheme and the integration of the various levels. The implementation of the Object Processor is an essential element in reaching this objective, because only at this level is it possible to have contextual information sufficient to permit a knowledge-based direction of the interpretation/segmentation of the image.

6.2 FUTURE IMPROVEMENTS AND LINES OF DEVELOPMENT.

We have already suggested the most important extensions of SVEX in the above conclusions concerning the prototype developed in this project, and, most importantly, in the analysis of the experiments.

E1. *Provide SVEX with greater adaptability, especially at the pixel level.* In order to produce a more robust and flexible system, we must provide mechanisms such as accommodation, selection of parameters for the procedures, procedure selection, etc. These mechanisms should be effective at all levels, but it is at the pixel level that they are most necessary, in order to adapt the calculation of diagnostics to the characteristics of the input signal. This requires evaluation, not only of the overall results of using the procedure, but also of the quality of the segmentation produced. Various solutions to this problem have been proposed [Nazi-84][Mats-90][Clem-93][Lied-91][Drap-89], but it continues to be an open question.

E2. *Include in SVEX the control of camera parameters.* We foresee the incorporation in SVEX of mechanisms of control of camera parameters, such as focus, diaphragm aperture, zoom, and camera orientation. This would result in greater adaptability, permitting the development of

Extended Abstract

systems based upon the paradigm of active vision, and widening SVEX's field of action to include vision problems not considered up to the present time.

E3. *Modify the definition of the decision functions associated with the classifiers.* The present implementation only permits the definition of decision functions between extremes (total confirmation or denial). Relaxing this restriction would permit the definition of decision functions (classifiers) that would produce confirmation or denial in either exclusive or relative terms.

E4. *Investigate other rules for the combination of evidence.* The two schemes for the incremental combination of the evidence produced by classifiers and rules, used in the two levels of SVEX to produce diagnostics, are the most common rules of combination used in expert systems [Gord-84]. There are circumstances that demonstrate that neither of these combination rules is optimal. We must, therefore investigate other combination rules. One interesting possibility is to abandon functionally-defined combination rules, and use instead tables for the incremental combination of evidence.

E5. *Inclusion of new segmentation modules.* The presegmentation module based on the Watershed transform has shown its versatility in the experiments performed up to the present. However, this does not exclude the inclusion of new presegmentation modules based upon other segmentation techniques capable of managing multiple diagnostic maps [Ross-89][Lim-90][Moli-92]. One aspect of the present module that could be improved is its necessity for the exhaustive definition of the "prototype" zones that appear in the image to be segmented. In environments where the image content is not completely defined *a priori*, this can pose an important problem, when a clearly-defined zone invades a zone corresponding to an unknown prototype. One possible solution to this problem would be to incorporate the capacity for automatic discovery of significant groups of connected points in the representation space defined by the diagnostic maps. A question closely related to the above, and which has also been suggested by other authors [Fuji-90], is whether it is necessary that the partition generated by the presegmenter be exhaustive; that is, that the set of segments defined in the initial partition form a complete tessellation of the image. This possibility, which would be of interest especially in the initial stages of the interpretation,

where the intent is to provide the most robust initial hypotheses possible, would require the redefinition of the spatial relations between segments, and would add a new dimension to the problem.

E6. Strengthen the capacity of the Segment Processor for relational processing. In the present implementation of the Segment Processor, the use of spatial relation between segments has been only partially explored. Among the possible extensions, we will point out those that seem most interesting and promising. One of these is the extension of the "Features" associated with the segments, which presently numerically qualify the entire segment, to include information related to vicinities. In the present implementation it is not possible to qualify numerically attributes of a relational or binary nature associated with the vicinity relations of the segment; for example, the average contrast along the common border between a segment and each of its neighbors [Nazi-84][Ross-89]. The inclusion of these binary characteristics can also be extended to the symbolic domain in a natural way. Another interesting possibility would consist of the ability to calculate in a selective manner certain characteristics that depend either on the shape or the nature (diagnostics) of neighboring segments. In this way, it would be possible to select the neighbors that would participate in the calculation, depending upon the satisfaction of a certain premise.

E7. Definition of spatial localizers based upon attributes. The spatial localizers used in the present implementation of the Segment Processor are purely positional (Above, Below, etc....). An interesting extension would be the inclusion of localizers that could include as an argument an attribute, numeric or symbolic, associated with the segment. By means of this kind of localizer it would be possible to describe conditions such as the neighbor "most square" or "most similar in size"

E8. Definition of new control actions in the Segment Processor. It would be interesting to include new control actions designed to deal with the appearance of inconsistencies in the interpretation. In the present implementation it is possible to implement control rules that will give notification of the presence of inconsistencies, such as when a segment labeled 'tree' is included within another

Extended Abstract

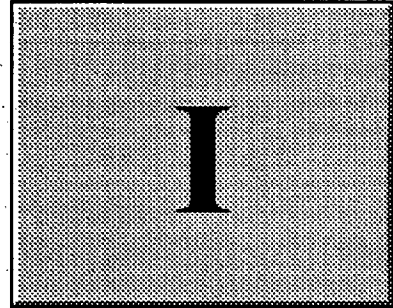
labeled 'highway'; but it is not possible to include any control action that will resolve such inconsistencies of interpretation [Fuji-90][McKe-89][Drap-89].

E9. *Implementation of the Object Processor.* As we have noted repeatedly during the discussion of our conclusions, the development and implementation of the Object Processor is a fundamental objective in the extension of SVEX, not only because it would represent the completion of a proposed model of an image-interpretation system applicable to more complex tasks, but also because its development will permit the addition to SVEX of more versatile and powerful control strategies. Furthermore, options suggested by the analysis of the experimental results, such as the search for diagnostics in parts of the image on the basis of prior hypotheses, the application of specific control rules to specific sets of segments, etc., and, in particular, the orientation of the interpretation process from the standpoint of the object knowledge base (Top-Down control), will only be possible once this processor exists.

E10. *Automation of the learning process.* Several authors [Niem-90b][Conn-87][Fich-92][Pell-94][McKe-89] have pointed out that the process of learning and acquisition of knowledge about models strongly constrain the practical realization of knowledge-based vision systems. The problem of learning has various facets, from the elaboration of tools for automated acquisition of qualified models (Sampler), and the selection of optimum procedures for numeric characterization, to the automatic generation of abstraction procedures that permit the transformation of data into symbolic categories, and the generation of symbolic rules.

CAPÍTULO

Segmentación e Interpretación de imágenes.



I.1 INTRODUCCIÓN.

El proceso de segmentación de imágenes ha sido considerado durante las dos últimas décadas como el resultado de una cierta etapa previa que posibilitaba el aislamiento de zonas en una imagen con significado relevante. El término "segmentación de imágenes" ha sido extensamente utilizado con distintos significados. En el contexto que nos ocupa lo entenderemos como aquel proceso que permite dividir la imagen en diferentes partes, regiones, que poseen uniformidad en alguna, o algunas, cualidades significativas para el observador. De esta definición no formal se desprende [Pavl-77] que el problema de la segmentación está relacionado con aspectos de percepción psicológica y en general no será posible encontrar una solución puramente analítica. Por ello, muchos de los algoritmos utilizados en segmentación incorporan la heurística y están orientados a resolver problemas específicos en entornos definidos.

Para disponer de una cierta efectividad en la solución a los problemas de segmentación, se necesita no sólo de conocimiento de los elementos a segmentar, sino además de una estrategia que nos indique "cómo" efectuarla. Estas cuestiones han estado planteadas implícita o explícitamente, desde las primeras definiciones y metodologías [Pavl-77], [Zuck-76], [Rose-82].

Capítulo I

Lamentablemente, constatamos que no existe una teoría, sólidamente contrastada sobre la segmentación, lo que nos obliga a abordarla desde las herramientas que disponemos. Sin embargo, las herramientas tienen las desventajas propias de las redes del pescador las cuales definen en gran medida las características de lo que se va a pescar en función del tipo de tamiz. No obstante, las implicaciones psicológicas y propias del dominio del observador atribuibles a los segmentos de una escena dejan la puerta abierta no sólo a planteamientos fundamentados en el dominio de la señal, sino a aproximaciones que permitan aprovechar la ventaja de la utilización en este nivel de conocimiento sobre la presencia de determinadas entidades en la escena.

Desde estos presupuestos vemos cuando menos surgir tres problemas de estudio:

- a) El desarrollo de técnicas y métodos de segmentación de imágenes como herramientas básicas para llevar a cabo la segmentación de una imagen.
- b) Cómo incorporar conocimiento explícito en segmentación de forma flexible, de manera que pueda adaptarse a contextos diferentes, y que esta adaptación no suponga cambios cualitativamente importantes en la estructuración de los procesos implicados en un sistema de segmentación.
- c) Cómo definir y planificar el control de manera que desde niveles superiores pueda dirigirse la segmentación utilizando conocimiento de alto nivel.

Estos tres problemas van a definir la organización de este primer capítulo. En primer lugar, daremos una definición formal de la segmentación de imágenes y describiremos brevemente las técnicas más empleadas en segmentación. A continuación, consideraremos la integración de estos métodos en sistemas expertos para proceso de imágenes, los cuales suponen una contribución importante en la línea de diseñar sistemas de segmentación y proceso de imágenes más robustos y autónomos, por cuanto intentan condensar y articular el conocimiento heurístico existente del uso de métodos y técnicas de proceso de imágenes. Algunos de estos sistemas expertos han constituido los módulos de segmentación de sistemas de interpretación de imágenes (Image Understanding Systems - IUS). Se realizará una descripción estructurada de algunos de los sistemas de interpretación de imágenes más conocidos, donde se pondrá de manifiesto la

arquitectura de los mismos, la representación del conocimiento empleada y cómo se organiza el control de la interpretación. Esta revisión constatará la existencia de una serie de guías o principios de diseño comunes para tales sistemas e identificará los problemas centrales encontrados en su diseño. Todo ello para presentar el contexto de este trabajo, sirviendo de preámbulo al segundo capítulo, donde se describe la propuesta de un sistema multinivel para segmentación de imágenes.

I.2 LA SEGMENTACION DE IMAGENES COMO UN PROCESO DE SIMBOLIZACION.

Para definir formalmente el concepto de segmentación [Pavl-77], consideremos una imagen $I(x,y)$ definida como una colección de pixels sobre un cierto dominio espacial, R , y sea Y un subconjunto de R que contiene al menos dos pixels. Entonces un predicado de uniformidad $P(Y)$ se define como aquel que: a) asigna el valor de verdadero o falso a Y dependiendo solo de las propiedades de $I(x,y)$ en la región o segmento Y . b) Adicionalmente, para ser considerado predicado de uniformidad, $P(Y)$ deberá verificar la propiedad de que si Z es un subconjunto no vacío de Y , entonces siempre que $P(Y)$ sea cierto, $P(Z)$ deberá serlo. Así por ejemplo, serán predicados de uniformidad:

$P_1(Y)$ = cierto "si el valor de la intensidad en dos pixels cualesquiera de Y es el mismo".

$P_2(Y)$ = cierto "si el valor de la intensidad en dos pixels cualesquiera de Y solo difieren en una cantidad dada".

Mientras que los siguientes predicados no satisfacen la definición anterior:

$P_3(Y)$ = cierto "si Y tiene al menos 10 pixels"

$P_4(Y)$ = cierto "si el máximo valor de $I(x,y)$ en Y excede de un valor dado".

En el primer caso, $P_3(Y)$ porque no depende de los valores de $I(x,y)$ sino del recuento de pixels sobre Y , y en el segundo, porque el predicado aludido puede no ser cierto en subconjuntos de Y .

Formalmente, definido un predicado de uniformidad $P(Y)$ en R , una partición de R en subconjuntos no vacíos R_1, R_2, \dots, R_n se considera una segmentación si se verifican:

$$R = \bigcup_{i=1..n} R_i \quad (I-1)$$

$$R_i \cap R_j = \emptyset \quad (I-2)$$

$$P(R_i) = \text{cierto} \quad \forall i \quad (I-3)$$

$$P(R_i \cup R_j) = \text{falso} \quad \forall i \neq j \quad R_i, R_j \text{ adyacentes.} \quad (I-4)$$

La segmentación puede considerarse como un proceso de simbolización entendida en el contexto de lo que así se considera en Reconocimiento de Formas o en Visión por Ordenador [Wils-88a]. Los símbolos de este proceso están vinculados al hecho de considerar la segmentación como la separación de una imagen en un conjunto de regiones disjuntas cada una de ellas con propiedades constantes, separadas por límites bien definidos. Sin efectuar consideraciones sobre la naturaleza de las propiedades relevantes utilizadas en una tarea de segmentación (bien sea nivel de gris, color o textura), entendemos que, en el contexto que nos ocupa, el resultado de un proceso de segmentación es una descripción simbólica que contiene dos componentes: una componente de clase, indicativa de las propiedades relevantes de la región o segmento, y una componente de posición, que define dónde la región aparece en la imagen. Conectamos así con el paradigma de representación estructural, donde cada forma viene representada por un conjunto de primitivas (tokens) y por una serie de relaciones entre ellas. En esta aproximación, cada primitiva está definido por dos conjuntos:

$$T = \langle \Omega, \Lambda \rangle \quad (I-5)$$

Donde el primer conjunto $\Omega = \{\omega_1, \dots, \omega_n\}$, representa los parámetros de identidad, mientras el segundo $\Lambda = \{\lambda_1, \dots, \lambda_m\}$, los correspondientes parámetros de localización espacial. El primer conjunto se corresponderá con un conjunto de etiquetas y sus respectivos grados de

pertenencia a clases de primitivas. Este conjunto es el elemento de trabajo de dos procesos, el que denominamos proceso Abstractor o de creación de estas clases o categorías desde datos numéricos y el proceso Referenciador que es aquel que realiza la transposición o sustitución de datos numéricos por dichas clases.

La acción que realiza este último proceso será la de referenciación o simbolización mediante la cual asignaremos símbolos al conjunto de datos. La distinción esencial entre los símbolos y las señales o datos (imágenes) de las cuales se obtienen, es que cada símbolo es una referencia a una clase de señales o datos.

Las características de las primitivas están definidas en función de valores numéricos de sus propiedades o en base a su pertenencia a categorías generales de entidades. Esta última es una de las características destacadas de los modelos de percepción artificial avanzados, esto es, la utilización de representaciones basadas en símbolos o clases abstractas de entidades, en contraposición con modelos más simples que utilizan fundamentalmente representaciones numéricas. El principal problema en el tratamiento simbólico estriba en la correspondencia operacional de los símbolos, es decir, en el paso de la frontera de una representación numérica a otra simbólica.

El grado de simbolismo de una representación podemos asimilarlo a la cantidad de referencias a clases que contiene, lo que está en relación con el número de procesos de referenciación, explícitos o implícitos, que ha sufrido la representación. En el caso general, las características de los primitivas definidas en base a clases constituyen jerarquías. Así una representación muy idónea es la de estructuras de "frames" [Mins-75], de tal forma que los primitivas heredan las propiedades de sus super-clases, y el proceso de referenciación se corresponde con el de matching en tales sistemas. Por ejemplo, si tenemos la siguiente representación de "frames", con clases representativas de las entidades *circulo*, *cuadrado*, *rojo*, *verde*, *cuadrado-rojo* y *circulo-verde*, de tal forma que los colores se describen por sus coordenadas RGB, y las figuras por una característica *f*.

círculo-verde:
 forma = círculo
 color = verde

cuadrado-rojo:
 forma = cuadrado
 color = rojo

círculo:
 Es-una = figura
 f = 10

cuadrado:
 Es-una = figura
 f = 3

verde:
 Es-un = color
 r-g-b = 0.0,1.0,0.0

rojo:
 Es-un = color
 r-g-b = 1.0,0.0,0.0

Una primitiva, primitiva-15, en un cierto nivel de representación, podrá pasar de una descripción numérica a una simbólica, con ciertos coeficientes de certeza, de la forma siguiente:

Representación-1

primitiva-15:
f = 8
r-g-b = 0.1,0.9,0.2

Representación-2

primitiva-15:
forma = círculo/0.7
color = verde/0.8

Representación-3

primitiva-15:
Es-un = círculo-verde/0.7

Para salvar el salto operacional entre el dominio de la señal, o numérico, y el simbólico, o de clases, son posibles varias alternativas. En el contexto de la obtención las diversas primitivas, consideraremos que tal proceso puede admitir una división en dos niveles que poseen

cuantitativamente atributos diferenciales. Cada uno de estos niveles está caracterizado por sus propios elementos básicos y, conforme se asciende en los mismos, aparece un grado de simbolización más elevado. Estos niveles están relacionados respectivamente con:

- a) Los pixels, o unidades elementales de información en la imagen
- b) Los segmentos o agregaciones de pixels que poseen una cierta unidad, o en los cuales se verifican predicados de uniformidad.

Los procesos a nivel de pixels son fundamentalmente numéricos y en el mismo se desarrollan procedimientos sobre las imágenes tales como detección de gradiente, convoluciones y otras operaciones propias del dominio de la señal. La salida de este nivel es un conjunto de diagnósticos o asignaciones a clases de pixels. La segmentación de una imagen en este nivel se corresponderá con la agregación de pixels espacialmente conexos que tengan una similar agregación a una cierta clase de pixels.

En el nivel de segmentos, la salida son agregaciones de pixels vertebradas en función de factores de forma o por ser uniformes en propiedades. Se puede corresponder con una lista de regiones con sus respectivas asignaciones a clases de segmentos, sus localizadores espaciales y el correspondiente grafo de regiones adyacentes. En este nivel la representación de las primitivas se gestiona a la vez en forma numérica y simbólica y los procesos que en él acontecen participan de ambos tipos de representación [Niem-90a]. Existen diferencias entre la unidad de representación utilizada en este nivel y la correspondiente a nivel de pixels; mientras este último maneja entidades predefinidas de límites espaciales estacionarios, en el nivel de segmentos las entidades o primitivas tienen carácter morfológico.

I.3 MÉTODOS Y TÉCNICAS.

Por su interés práctico, clásicamente [Rise-77] se distinguen dos tipologías de primitivas resultados de un proceso de segmentación como el descrito, las que derivan de regiones y aquellas otras que se obtienen desde elementos de contornos. Ambos tipos de primitivas son complementarios y pueden cooperar entre sí, no solo en el análisis de la imagen en cuestión sino

en el proceso de segmentación en si mismo. En general estas primitivas se originan debido a factores tales como la geometría local de los objetos de la escena, la reflectancia de las superficies visibles, la iluminación de la escena o el punto de observación. Su obtención se ve condicionada por el conjunto de variaciones irrelevantes que aparecen en la imagen de la escena. Estas fluctuaciones y variaciones incluyen típicamente a elementos tales como posición relativa, movimiento de la cámara, los cambios en la iluminación ambiente, ruido en los procesos de adquisición, etc.. Las técnicas de detección de regiones y contornos que a continuación analizaremos, tratan de minimizar la presencia de este ruido y de proporcionar la localización e identificación de la primitiva en cuestión en las mejores condiciones posibles.

I.3.1 DETECCION DE CONTORNOS.

Un contorno se puede definir como un cambio brusco registrado en los valores de la intensidad o textura de una imagen en un entorno de la misma. Los contornos en muchas circunstancias se corresponden con el lugar geométrico de los puntos frontera entre regiones. Una de las principales cualidades de la utilización de contornos es la de permitir definir la segmentación a partir de transiciones por lo que se precisa un volumen de datos reducido. Una imagen discretizada posee una elevada redundancia, de manera que una gran parte de la información contenida no se utiliza en las etapas de alto nivel en sistemas de visión por ordenador. La detección de contornos es un medio de generar una descripción compacta que mantiene o preserva gran parte de la información estructural de la imagen.

Los requerimientos exigibles a un buen detector de contornos pasan no solamente porque responda adecuadamente a la estructura verdadera del contorno, sino además porque sea insensible a ruido y presente una realización computacional eficiente y poca costosa.

Existen buenas recopilaciones de algoritmos para la detección de contornos [Davi-75][Nalw-86]. Entre las diversas aproximaciones a la detección de contornos distinguiremos primeramente aquellas que se fundamentan en modelizar la imagen como un campo aleatorio y tratar de detectar los contornos desde los cambios de diversas propiedades estadísticas que caracterizan a los mismos [Huan-88][Nahl-77]. Entre los modelos utilizados en esta

aproximación, denominada estocástica, se encuentran los basados en la utilización de los campos de Markov [Hans-82] y los modelos autoregresivos. En general, la evaluación se hace computacionalmente más costosa dependiendo de la complejidad de los modelos que se empleen y de la tipología de la imagen en cuestión.

En otras aproximaciones se realiza la búsqueda de elementos en la imagen que sean indicativos del ajuste de un borde ideal mediante una función escalón. Como consecuencia, la detección de los posibles contornos se establece evaluando las formas correspondientes utilizando la convolución con máscaras de orientaciones diversas. La orientación del contorno se toma como la de la máscara que da la mejor correspondencia en un punto dado, y la magnitud de esta correspondencia como la medida del valor del borde. Una alternativa a este último método reside en expandir las funciones imagen y máscara en términos de un conjunto de funciones ortogonales básicas y utilizar la suma de los cuadrados de las diferencias entre coeficientes que se correspondan como medida de error [Huec-71][Neva-77]. Lógicamente, para efectuar los cálculos será necesario truncar la expansión a unos pocos términos, lo cual dará una medida de error que sólo es aproximada.

La aproximación más recurrida es la utilización de filtros, bien de respuesta impulsional finita (FIR) o infinita (IIR). En un primer gran grupo de técnicas de detección de contornos se encuentran los operadores de diferencias [Rose-82], que consideran la imagen como una función continua de la intensidad y asocian la presencia de contornos a cambios en dicha función. La detección se establece utilizando operadores tales como el gradiente, la laplaciana o la diferencia de promedios ampliamente tratados en la literatura [Gonz-77][Rose-82][Jain-89]. Un estudio comparativo de los detectores de bordes más populares basados en diferencias finitas puede encontrarse en [Flec-92].

Dentro de esta última aproximación se incluyen los operadores planteados en el marco de las teorías de Marr sobre el proceso visual [Marr-80][Marr-82]. Según Marr subyacen dos ideas en la detección. La primera de ellas es que los cambios de intensidad en una imagen ocurren a diferentes escalas, y por tanto para detectarlos es preciso emplear operadores de diferentes

tamaños. La segunda de estas ideas es que un cambio brusco de intensidad dará lugar a una punta o valle en la primera derivada, o de un modo equivalente, a un cruce por cero en su segunda derivada. El procedimiento sugerido se fundamenta en la obtención de los cambios de signo que se producen en la aplicación del operador laplaciana a una imagen previamente convolucionada con un núcleo operador gaussiano. Esta operación es equivalente a la de convolución de la imagen con una máscara que es la laplaciana de la función gaussiana (LoG):

$$\begin{aligned} Z(x,y) &= \nabla^2 \int_A G(x-x',y-y')I(x',y')dA = \\ &= \int_A (\nabla^2 G)(x-x',y-y')I(x',y')dA \end{aligned} \quad (I-6)$$

Donde $Z(x,y)$ es una imagen en la cual los puntos en cuyo entorno se producen cambios de signo, se corresponden bastante fidedignamente con puntos de un contorno real. $I(x,y)$ representa la imagen original que describe los niveles de iluminación de la escena y $G(x,y)$ es una gaussiana bidimensional que define al núcleo de convolución cuya Laplaciana tiene por expresión:

$$\nabla^2 G(x,y) = \frac{1}{2\pi\sigma^4} \left[\frac{x^2+y^2}{\sigma^2} - 2 \right] \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \quad (I-7)$$

Este núcleo tiene área nula, esto es no dará respuesta si no existen cambios de intensidad, propiedad útil para la detección. Para una aplicación eficiente de este operador se utiliza una factorización del núcleo LoG, en la forma:

$$\nabla^2 G(x,y) = h_1(x) h_2(y) + h_2(x) h_1(y) \quad (I-8)$$

Donde:

$$h_1(v) = \frac{1}{\sigma^2\sqrt{2\pi}} \left[\left(\frac{v}{\sigma}\right)^2 - 1 \right] \exp\left(-\frac{v^2}{2\sigma^2}\right) \quad (I-9)$$

$$h_2(v) = \frac{1}{\sigma^2\sqrt{2\pi}} \exp\left(-\frac{v^2}{2\sigma^2}\right) \quad (I-10)$$

Esta factorización permite convertir la convolución bidimensional en un conjunto de convoluciones monodimensionales lo que reduce el costo computacional. Implementaciones computacionalmente más eficientes pueden encontrarse en [Chen-87][Sota-89]. Los contornos se obtienen detectando aquellos puntos que presentan cambio en el signo en la respuesta a este operador. De acuerdo con la teoría de Marr los cambios de intensidad se producirán para varias resoluciones y se obtendrán convolucionando operadores LoG de diferentes frecuencias (\square) detectando en la respuesta aquellos puntos en los cuales se produce un cruce por cero. El problema se plantea en términos de la integración de la información suministrada por detectores de ancho variable a lo largo de las diferentes escalas [Will-90][Lu-89][Lu-92].

Formalmente, las condiciones que debe cumplir un buen detector de contorno pueden resumirse en las siguientes [Cann-86]:

- a) Detección correcta, es decir, debe ser baja la probabilidad de ignorar como perteneciente al contorno a un punto del contorno real y también baja la de etiquetar como del contorno a un punto no perteneciente a contornos reales.
- b) Localización correcta, en la medida de lo posible, los contornos detectados deben coincidir con los contornos debidos a transiciones entre regiones reales, ajustándose al centro de la transición.
- c) Unicidad de la detección, esto es que un contorno real solo debe detectarse una vez, lo cual significa que deben desecharse aquellos procedimientos de detección que generen dos o más contornos en cada transición.

Esta tercera condición es necesaria porque la formalización del primer criterio no deja explícitamente definida la ausencia de múltiples respuestas en la señal de salida ante una única transición. Para el diseño de operadores que verifiquen las condiciones anteriores, resulta interesante disponer de medidas del grado de adecuación del detector bajo diseño a las mismas. Canny [Cann-86], proporciona una función de evaluación algebraica, de manera que aplicando

Capítulo I

herramientas de cálculo variacional sea posible encontrar el operador que maximice la mencionada función de evaluación. El problema se plantea en términos de obtener aquellos funcionales que permitan evaluar el comportamiento del operador desde los criterios intuitivos mencionados anteriormente, dejando así la puerta abierta a la utilización de distintas alternativas y herramientas.

Un cuarta condición se genera desde la necesidad de una implementación eficiente que reduzca los tiempos de detección. En general, los esquemas en que se utiliza un filtro de respuesta infinito (IIR) truncado e implementado mediante una máscara de convolución, produce costes computacionales que crecen con el tamaño de la máscara. Por ello resulta particularmente interesante aquellos filtros que permitan una realización recursiva explícita [Deri-87][Sark-91][Shen-92].

Para entender mejor como diseñar operadores con estas características, consideremos [Cann-86] una función de salto monodimensional $I(x)$ definida por:

$$I(x) = A u(x) + n(x) \quad (\text{I-11})$$

Donde $A u(x)$ representa un escalón de amplitud A y $n(x)$ ruido blanco de varianza n_0^2 . Si asumimos que la detección del flanco se realiza mediante la convolución de la señal $I(x)$ con la función-filtro incógnita $f(x)$, que se desea cumpla las condiciones anteriores, el posible contorno coincidirá con el máximo de la salida de este filtro:

$$O(x_0) = \int_{-\infty}^{\infty} I(x) f(x_0 - x) dx \quad (\text{I-12})$$

La primera de las condiciones se verificará por aquellas funciones que maximicen la relación señal ruido (SNR) la cual se define como cociente de la respuesta al escalón aislado y la raíz cuadrada de la media cuadrática de la respuesta del ruido:

$$SNR = \frac{A}{n_o} \frac{\int_{-\infty}^{\infty} f(x) dx}{\sqrt{\int_{-\infty}^{\infty} f^2(x) dx}} = \frac{A}{n_o} \Sigma \quad (I-13)$$

A Σ se le denomina índice de detección. La segunda condición se corresponde con minimizar la varianza σ^2 de la posición de cruce por cero, esto es maximizar el criterio de localización (L) definido como el recíproco de σ :

$$L = \frac{A}{n_o} \frac{|f'(0)|}{\sqrt{\int_{-\infty}^{\infty} f^2(x) dx}} = \frac{A}{n_o} \Lambda \quad (I-14)$$

Donde Λ representa al índice de localización. Para limitar la aparición de respuestas múltiples deberemos limitar el número de picos en la respuesta lo que se corresponde con una baja probabilidad de la existencia de mas de un flanco. Para plantear esta condición, la distancia entre picos en la respuesta ruidosa de $f(x)$, x_{max} se iguala a una cierta fracción k del ancho del operador W :

$$x_{max} = kW = 2\pi \sqrt{\frac{\int_{-\infty}^{\infty} f^2(x) dx}{\int_{-\infty}^{\infty} f^2(x) dx}} \quad (I-15)$$

Desde los índices de localización, detección y limitación del número de picos Canny [Cann-86] realiza una combinación de los mismos maximizando el producto $\Sigma \Lambda$ bajo las restricciones del tercer criterio, lo que le permite encontrar el conjunto de funciones $f(x)$ deseado. La extensión al caso bidimensional, del detector de Canny y otras funciones obtenidas en esta

línea [Deri-87], se efectúa construyendo dos máscaras en dos direcciones, una para la dirección x y otra en la y , formadas por el operador monodimensional $f(x)$ alineado en la dirección x (o la y en su caso) y con una función de proyección paralela a la dirección y (o la x en su caso). Posteriormente se convoluciona la imagen con ambas máscaras en las dos direcciones y se estima la dirección del flanco desde las respectivas salidas. El siguiente paso reside en efectuar sobre la magnitud de la salida una operación de supresión de no-máximos en dicha dirección y una operación de umbralizado con histéresis. Por último, de entre las múltiples funciones que en mayor medida verifiquen las condiciones anteriores resultan particularmente interesantes aquellas que admiten una implementación recursiva eficaz [Deri-87] [Sark-91].

I.3.2 DETECCION DE REGIONES.

Las distintas técnicas de obtención de primitivas que representan una agregación de pixels formada según un cierto criterio, pueden clasificarse en los siguientes tipos:

- a) Segmentación basada en el espacio de propiedades.
- b) Crecimiento de regiones.
- c) División y fusión.
- d) Segmentación iterativa: relajación.

Tal taxonomía no es estricta y representa una extensión de la establecida en [Hara-85]. No obstante, existen técnicas, o variantes de las mismas, que utilizan procedimientos que son híbridos de algunos de los grupos mencionados.

I.3.2.1 SEGMENTACION EN REGIONES BASADA EN EL ESPACIO DE PROPIEDADES.

En este grupo de técnicas se utilizan agrupamientos en el espacio de medidas para definir particiones en dicho espacio. De esta forma, a cada pixel en el dominio espacial se le asigna una etiqueta que se corresponde con la partición efectuada en el espacio de medidas. Los segmentos o regiones de la imagen se definen entonces como las componentes conexas de los pixels que tienen la misma etiqueta. La precisión del proceso dependerá de como estén separados las

representaciones de los objetos en el espacio de las propiedades. Típicamente el método presenta buenos resultados en aquellos casos en que se tienen pocos objetos y que estos poseen propiedades de los pixels claramente diferenciadoras (tonos de gris sobre fondo uniforme, colores diferentes, estadísticos locales diferenciales, etc.)

Dentro de este grupo, una de las técnicas más utilizada es el umbralizado. Dada una imagen $f(x,y)$, definida como una aplicación $f: N \times N \rightarrow G$ donde G representa un conjunto de posibles valores, un umbral u y un par de niveles $\Lambda(\lambda_1, \lambda_2)$, el resultado del proceso de umbralizado es una función $f_t: N \times N \rightarrow \Lambda$ definida por:

$$f_t(x,y) = \begin{cases} \lambda_1 & \text{si } f(x,y) > u \\ \lambda_2 & \text{si } f(x,y) \leq u \end{cases}$$

El problema reside en determinar el valor de u basado en un cierto criterio, para lo cual se han descrito en la literatura un amplio espectro de técnicas [Saho-88]. En general estos métodos se clasifican en dependientes de los puntos y dependientes de la región. En el primer grupo se encuentran aquellos en los que el valor del umbral se determina solamente desde el valor $g \in G$ de cada pixel. Por el contrario, si esta determinación se establece desde alguna propiedad local que depende del entorno del pixel trataremos con métodos dependientes de la región. La técnica de umbralizado puede además aplicarse a toda la imagen, estaremos en el caso de umbralizado global, o a una parte de la misma definida según una cierta estrategia, en cuyo caso hablaremos de umbralizado local.

Las técnicas que utilizan histogramas como elemento de partida para realizar la segmentación se fundamentan en que las zonas homogéneas de la imagen se corresponden con agrupamientos en el espacio de medida, de esta forma el proceso de segmentación se realiza sencillamente estableciendo una correspondencia de los agrupamientos detectados en el espacio de propiedades sobre el dominio espacial. Para imágenes monoespectrales la evaluación del histograma es una operación directa. El proceso de agrupamiento en el espacio de propiedades puede ser realizado en este caso determinando los valles en el histograma y definiendo los

Capítulo I

agrupamientos a partir del intervalo entre valles. Entre las numerosas técnicas sugeridas para realizar este proceso destacamos la propuesta por Ohlander [Ohla-78] que utiliza la idea de agrupamiento de forma recursiva. Se comienza definiendo una máscara que abarca a todos los pixels y dada cualquier máscara se realiza la evaluación del histograma de la imagen enmascarada. Posteriormente se efectúa un proceso de agrupamiento en el espacio de propiedades de la imagen y los pixels se identifican con el agrupamiento con el que se relaciona. Si sólo existe un agrupamiento entonces el proceso de enmascaramiento termina. Si hay mas de un agrupamiento entonces cada componente conectada de todos los pixels de un mismo agrupamiento se utiliza para generar una nueva máscara. Durante las sucesivas iteraciones las sucesivas máscaras que se han ido obteniendo seleccionarán los pixels de la imagen. Este proceso de división se repite hasta que no sea posible generar nuevas máscaras.

En el contexto de la umbralización global y los métodos dependientes de propiedades de los puntos destacaremos tres tipos de técnicas, las basadas en medidas de entropía, las que preservan momentos y las de error mínimo. Las primeras consideran el histograma de una imagen como una fuente de símbolos y obtienen el umbral óptimo aplicando conceptos de teoría de la información [Pun-81][Kapu-85]. Las del segundo computan los valores de umbral de tal manera que los momentos de la imagen a ser umbralizada se mantienen en la de salida [Tsai-85]. Por último, en los métodos de error mínimo [Hall-79][Kitt-86], se considera al histograma como una estima de la función de densidad de probabilidad de la mezcla de población que comprende los pixels de los objetos y del fondo en la escena, y se introduce una función criterio que permite realizar la anterior estima.

Entre los métodos de umbralizado que utilizan técnicas dependientes de las propiedades de la región se encuentran la utilización de propiedades estadísticas para la definición del umbral [Ahuj-78][Dera-83] y las que derivan de la utilización de métodos de relajación [Rose-81]. Estas últimas se fundamentan en primero considerar una primera clasificación probabilística de los pixels según sus valores de nivel de intensidad en dos clases (claro y oscuro) e ir progresivamente ajustando la mismas de acuerdo con las probabilidades de un cierto entorno del pixel. El proceso

itera hasta que la clasificación del pixel no sufre cambios, o algún parámetro de medida global del número de cambios no experimenta una variación notable.

Otro conjunto de técnicas conceptualmente interesantes son las que conciben el problema de la segmentación de imágenes como un problema de naturaleza difusa o borrosa (fuzzy), e intentan abordarlo aplicando las ideas propias de la teoría de conjuntos borrosos [Zade-65][Zade-75][Zade-83]. La idoneidad de esta aproximación para tratar algunos problemas en el ámbito de la Visión Artificial se basa en el hecho de que los conceptos que se manejan en este dominio suelen ser borrosos en el sentido estar definidos de manera vaga e imprecisa y/o incorporar incertidumbre en su detección. Así, por ejemplo, la clasificación de un pixel como "objeto" o "fondo" en un problema de umbralizado, la decisión de considerar una porción de un contorno como esquina o una cierta región como "cuadrada", llevan normalmente asociado una incertidumbre o imprecisión en la decisión. Un enfoque tradicional, *duro (crisp)*, en el que la asignación de una cierta etiqueta se establece de manera dicotómica, es o no es, produce en el momento de la asignación una decisión determinante e irreversible que "olvida" la incertidumbre con la que se tomó dicha decisión. Esto supone indiscutiblemente una pérdida de información y convierte cada uno de estos pasos en una decisión crítica. Lo que se intenta mediante la aplicación de las ideas de la teoría de conjuntos borrosos es precisamente que esta información, presente en forma de incertidumbre, se preserve y pueda ser utilizada por los diferentes niveles de procesamiento de un sistema de visión. La aplicación de estas ideas a la segmentación de imágenes ha producido fundamentalmente dos conjuntos de técnicas definidas en el espacio de propiedades: técnicas de umbralizado borroso para segmentación de imágenes en niveles de gris y técnicas de agrupamiento (clustering) borroso.

Las técnicas de umbralizado borroso se fundamentan en la utilización de una función de pertenencia, normalmente de tipo S, para clasificar los pixels como pertenecientes al prototipo o clase "objeto" presente en la imagen. La selección o ajuste de la función de pertenencia apropiada se realiza mediante la minimización de diferentes medidas borrosas que miden en la imagen segmentada bien la "confusión" resultante en niveles de gris (entropía global, índice de compactación o de borrosidad) o la indefinición en la forma de las regiones resultantes

(compactación difusa) [Rose-84][Pal-88][Murt-90][Pal-93]. La segmentación final "dura" se establece definiendo un umbral sobre el grado de pertenencia (α -corte).

En el campo del reconocimiento de formas, las ideas sobre conjuntos borrosos han inspirado numerosos algoritmos de agrupamiento o clustering [Bezd-81], normalmente bastante efectivos aunque de elevado costo computacional. Entre estas técnicas, la más utilizada en la segmentación de imágenes ha sido la extensión borrosa del algoritmo K-medias. Este algoritmo se basa en la minimización iterativa de una función objetivo definida como la suma ponderada de la distancia de los pixels al centro de la clase prototipo o agrupamiento. Un extremo local de la función objetivo se considera un agrupamiento óptimo o segmentación de la imagen. La función objetivo a minimizar está dada por:

$$W_m(U,V) = \sum_{k=1}^n \sum_{i=1}^c (U_{ik})^m \|x_k - v_i\|_A^2 \quad (I-17)$$

donde $X = \{ x_1, x_2, \dots, x_n \}$ es un conjunto finito de datos en R^d , $x_k \in R^d$, $1 \leq k \leq n$, es un vector d -dimensional de medidas; $\|\cdot\|_A$ es una norma definida por un producto interno, $\|Q\|_A = Q^T A Q$, A es una matriz $[d \times d]$ definida positiva; $V = \{ v_1, v_2, \dots, v_c \}$ es un conjunto de centros de clases, $v_i \in R^d$, $1 \leq i \leq c$ representa el centro de la clase i -ésima; U_{ik} indica el grado de pertenencia del dato k -ésimo a la clase i -ésima, y $m \in [1, \infty)$ controla la borrosidad del proceso de agrupamiento, siendo ésta directamente proporcional al valor de m , de manera que si $m=1$, el algoritmo de agrupamiento se convierte en el K-medias "duro". Diferentes versiones del algoritmo K-medias borroso han sido utilizadas en la segmentación de imágenes en contextos diversos, como imágenes aéreas [Triv-86] o imágenes del cerebro obtenidas por resonancia magnética [Hall-92]. Lim y Lee [Lim-90] han aplicado una versión del algoritmo K-medias borroso a la segmentación de imágenes en color con buenos resultados. El método propuesto realiza la segmentación en dos etapas. La primera genera una primera aproximación a la segmentación de la imagen mediante la detección de máximos significativos en los histogramas de las componentes de color. Esta detección requiere del filtrado de los histogramas de las componentes de color en el espacio de escalas [Witk-84] y la posterior definición de zonas de indeterminación alrededor de los mínimos locales entre máximos. Este procedimiento genera una cubicación del espacio de representación,

componentes de color, donde los volúmenes ocupados por los diferentes máximos definen el conjunto de clases o prototipos de color. Los pixels incluidos en los diferentes volúmenes se clasifican como pertenecientes al correspondiente prototipo. Esta primera etapa deja sin clasificar los pixels que se encuentran en las zonas de indefinición alrededor los mínimos de los histogramas. La clasificación de estos pixels es el objetivo de la segunda etapa para producir la segmentación definitiva de la imagen. Para ello se emplea una versión del algoritmo K-medias borroso donde la función de pertenencia, U_{ik} , se define como:

$$U_{ik} = \frac{1}{\sum_{j=1}^c \left(\frac{\|x_k - v_i\|}{\|x_k - v_j\|} \right)^{\frac{2}{m-1}}} \quad \forall i, \forall k \quad (I-18)$$

donde el centroide o prototipo de la clase i-ésima se obtiene de:

$$v_i = \frac{\sum_{k=1}^n (U_{ik})^m x_k}{\sum_{k=1}^n (U_{ik})^m} \quad (I-19)$$

I.3.2.2 CRECIMIENTO DE REGIONES.

Dentro de este concepto se incluyen los métodos que alcanzan la segmentación final en una imagen desde la agregación de pixels vecinos en un proceso iterativo basado en medidas de similaridad [Zuck-76]. En estos esquemas un pixel dado se añadirá a un segmento o región existente si, mediante alguna medida de similaridad, la región resultado no varía sustancialmente sus parámetros de homogeneidad. En este proceso podrá producirse la unión de regiones de grado de similaridad alto. Dentro este contexto se encuentran los métodos de crecimiento de regiones por encadenamientos simples, híbridos y a promedios [Hara-85].

Los esquemas de crecimiento de regiones por *encadenamiento simple* consideran cada pixel como un nodo de un grafo. Los pixels vecinos que se consideran suficientemente similares

Capítulo I

se unen mediante un arco. La partición de la imagen está constituida por los conjuntos de pixels que son espacialmente conexos. Los esquemas de crecimiento de regiones por *encadenamiento híbrido*, más robustas que los anteriores, se caracterizan por asignar a cada pixel un vector de propiedades obtenido a partir de una ventana centrada en el pixel. La similaridad entre pixels implica entonces que sus respectivos entornos son similares en un cierto sentido. La similaridad se establece como una función sobre los valores de los pixels vecinos. Los métodos de crecimiento de regiones por *encadenamientos a promedios* recorren la imagen de una forma predeterminada comparando el valor del pixel con la media de agrupaciones de pixel vecinas ya creadas. Si el valor del pixel es suficientemente parecido al valor medio de una región vecina, el pixel se une a la región actualizándose la media. Si existen varias regiones vecinas en condiciones de integrar el pixel, éste se une al de media más próxima al valor del pixel. Sin embargo, si la diferencia entre dos regiones competidoras es pequeña, entonces se define una nueva región por la unión de las regiones y el pixel. Por el contrario, si ninguna región vecina es suficientemente similar en valor medio, el pixel se constituye como una nueva región.

Un método clásico de crecimiento de regiones, que puede ser englobado en el último grupo descrito en el párrafo anterior, es el debido a Brice y Fenema [Bric-70]. El método comprende dos etapas. En la primera, se definen *regiones atómicas* por agrupación de pixels 4-conexos de igual amplitud. A continuación se aplican dos reglas heurísticas para fusionar regiones atómicas entre las exista un frontera débil. Así, si R_1 y R_2 , son dos regiones vecinas con perímetros P_1 y P_2 , C es la longitud de la frontera común entre ambos segmentos y D es la fracción de C donde la diferencia de amplitud a ambos lados de la frontera es menor que un cierto umbral ϵ_1 , entonces R_1 y R_2 se fusionan si:

$$\frac{D}{\text{MIN}(P_1, P_2)} > \epsilon_2 \quad (\text{I-20})$$

donde ϵ_2 es una constante (típicamente, $\epsilon_2 = 1/2$). Esta regla heurística impide la unión de regiones vecinas de tamaño parecido, pero permite que las regiones más pequeñas sean absorbidas por las más grandes. La segunda regla elimina las fronteras débiles que persistan después de la aplicación de la primera regla. Explícitamente, dos regiones vecinas se fusionan si:

$$\frac{D}{C} > \epsilon_3 \quad (I-21)$$

donde ϵ_3 es otra constante (típicamente, $\epsilon_3 = 3/4$). Este método proporciona segmentaciones relativamente buenas en el caso de escenas simples con escaso número de objetos y poca textura.

Un método de crecimiento de regiones de especial relevancia en el marco de esta tesis es el basado en la transformada Watershed. Desarrollada inicialmente en el campo de la morfología matemática [Diga-78] para el proceso de imágenes binarias y extendida posteriormente a problemas de segmentación de imágenes de niveles de gris [Beuc-79]. Esta

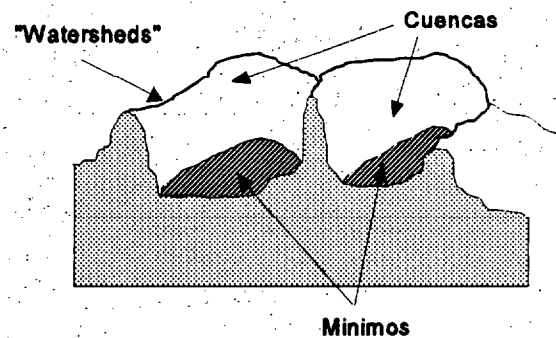


Figura I.1 Cuencas y límites de cuencas (watersheds).

transformación puede explicarse utilizando un símil basado en un proceso de inundación de una superficie topográfica [Beuc-90][Vinc-91]. Consideremos una imagen definida por una cierta función $I(x,y)$ que representa los niveles de intensidad y además otra función $F(x,y)$ que representa la topografía de $I(x,y)$. Para esta función F pueden elegirse varias alternativas como por ejemplo que los valores de $F(x,y)$ se correspondan con los valores absolutos del gradiente de $I(x,y)$. En este símil, las regiones homogéneas de $I(x,y)$ se corresponderán con mesetas o mínimos de $F(x,y)$, rodeadas de puntos de más alta cota que formarán caminos cerrados. Estos caminos o contornos que separan los diferentes mínimos se denominan líneas divisorias de aguas, o "watershed" y el área delimitada por cada "watershed" y su mínimo define la cuenca o embalse del mismo. El propósito del algoritmo de la transformada Watershed es dividir la imagen en estas cuencas o embalses que se corresponderán con regiones homogéneas de $I(x,y)$. Este símil puede ayudar a entender como progresa el algoritmo asumiendo que el mínimo de la superficie topográfica se horada y que se "inunda" lentamente con líquido la misma. El proceso hará que progresivamente la inundación se extienda por toda la cuenca o embalse. Para alcanzar la segmentación se impone la restricción de que el líquido que proviene de diferentes cuencas no pueda mezclarse, construyéndose en aquellos pixels límites una "presa". Al final de este proceso de inmersión cada

mínimo estará rodeado por "presas" que delimitan las cuencas, correspondiéndose estas "presas" con los watershed de $F(x,y)$ (Figura I.1).

La transformación Watershed tal y como ha sido descrita, cuando se aplica a la segmentación de imágenes [Beuc-90][Vinc-91] puede producir un número elevado de regiones y, por tanto, una sobresegmentación de la imagen. Esto se debe a la presencia de múltiples mínimos que son de escasa relevancia, pero que conducen a la aparición un alto número de pequeñas regiones. En este sentido, es necesario modificar la transformación Watershed para solventar este problema. La solución clásica es la utilización de marcadores para seleccionar los conjuntos de puntos conectados que conforman un mínimo. Estos marcadores se seleccionan desde el conocimiento de las imágenes y el objetivo de la segmentación. Los puntos marcados se utilizan entonces para modificar el gradiente de la imagen, tal que los puntos marcados conectados sean los únicos mínimos. Posteriormente se aplica la transformación Watershed utilizando el gradiente modificado de la imagen para obtener una partición de la imagen. En el capítulo III daremos una descripción más completa de la transformada Watershed dentro del contexto de este trabajo, así como de su utilización como método para producir una partición inicial de una imagen desde varios mapas de pixels.

I.3.2.3 DIVISION Y FUSION.

La técnica de división y fusión (Split & Merge) propuesta por Horowitz y Pavlidis [Horo-76] combina las técnicas de crecimiento y división de regiones. En ella, se comienza con una partición inicial de la imagen, donde no se asegura que se cumplan los predicados correspondientes a las ecuaciones (I-3) y (I-4), y se intenta obtener una segmentación de la imagen que verifique éstos además de los definidos por las ecuaciones (I-1) y (I-2).

El procedimiento para obtener la segmentación de la imagen comienza con la obtención de una partición inicial de la imagen en regiones cuadradas (R_i ; $i=1..n$ t.q. $\log_4 n \in \mathbb{N}$), como se muestra en la figura I.2. Este proceso se efectúa dividiendo toda la imagen en cuatro regiones, y a su vez cada una de éstas en otras cuatro, y así sucesivamente hasta alcanzar el tamaño deseado. El mecanismo descrito puede ser visto como la construcción de un árbol cuaternario

hasta un cierto nivel que define el tamaño de las regiones R_i (Figura I.3), donde el nodo raíz se hace corresponder a toda la imagen, y cada nodo hijo a una subdivisión de la misma.

Una vez obtenida la partición inicial, el siguiente proceso es el de fusión, en el cual se procede a unir todas las regiones que teniendo igual tamaño, posean el mismo nodo padre en el árbol y su unión resulte uniforme. Para determinar la uniformidad de la región resultante de la fusión, se utiliza un cierto predicado de uniformidad $P(Y)$. En una segunda fase, se determina la uniformidad de cada una de las regiones que no han sido fusionadas en el proceso anterior. Si una región no es uniforme, se divide en cuatro y se vuelve a realizar el proceso sobre cada una de las regiones resultantes. Esta división continúa hasta que todas las regiones sean uniformes según el predicado $P(Y)$ utilizado. Durante la fusión solo se unen regiones que comparten el mismo nodo padre en el árbol, por lo tanto es necesario llevar a cabo un proceso de agrupamiento. Este proceso une regiones que siendo espacialmente adyacentes en la imagen, residen en diferentes ramas del árbol. Esta situación es posible debido a que en la partición inicial de la imagen un objeto de la misma puede ser fragmentado en diferentes partes, perteneciendo cada una de ellas a una rama diferente del árbol. Después de realizado el proceso de segmentación pueden aparecer pequeñas regiones debido a la existencia de ruido en la imagen o zonas estrechas de transición entre grandes regiones. La calidad visual de la segmentación obtenida se puede mejorar

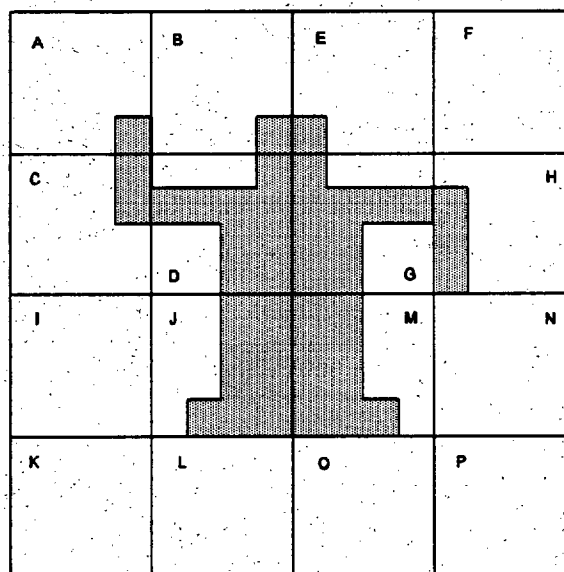


Figura I.2 Partición unicial de la imagen en el proceso de división y fusión.

eliminando estas pequeñas regiones mediante un proceso de filtrado. Este proceso consiste en fusionar cada una de estas regiones con la más similar entre las adyacentes, entendiéndose como medida de similitud, por ejemplo, la diferencia de niveles de gris promedio.

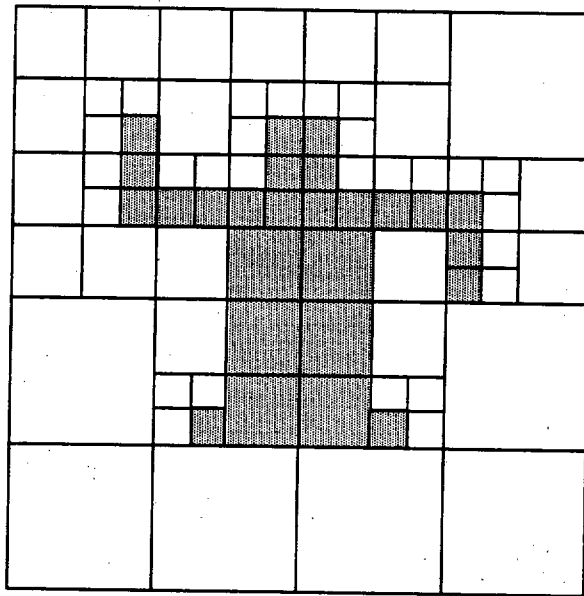


Figura I.3 Partición resultante del proceso de división, aplicado sobre la imagen de la figura I.2.

Se pueden realizar algunas modificaciones al método original, estas se fundamentan principalmente en el uso de un árbol cuaternario como estructura de datos, y en alterar el orden de los procesos que intervienen en el método. Esta alteración del orden conlleva a que el proceso de división de las regiones R_i que se consideran no uniformes, se realice después de obtenida la partición inicial.

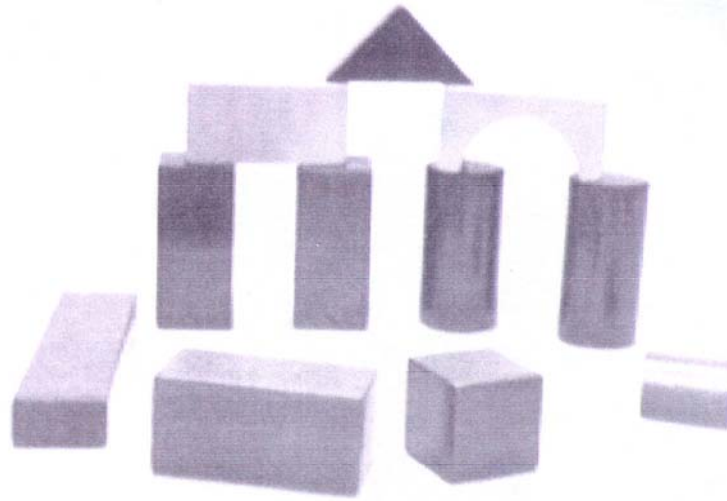
Como resultado del proceso de división se obtiene una partición de la imagen, en la que todas las regiones resultantes son uniformes, cumpliendo por tanto la condición que refleja la ecuación (I-1). A continuación se procede a la fusión de todas las regiones adyacentes, y cuya

unión resulte uniforme, según un predicado de uniformidad $Q(Y)$, que no tiene porque ser el mismo que al utilizado en el proceso de división. El uso del árbol cuaternario como estructura de datos permite la utilización de los procedimientos propuestos por Samet [Same-82] para la búsqueda de regiones adyacentes. Estos procedimientos no son computacionalmente costosos, con lo que la eficiencia de todo el proceso se ve incrementada.

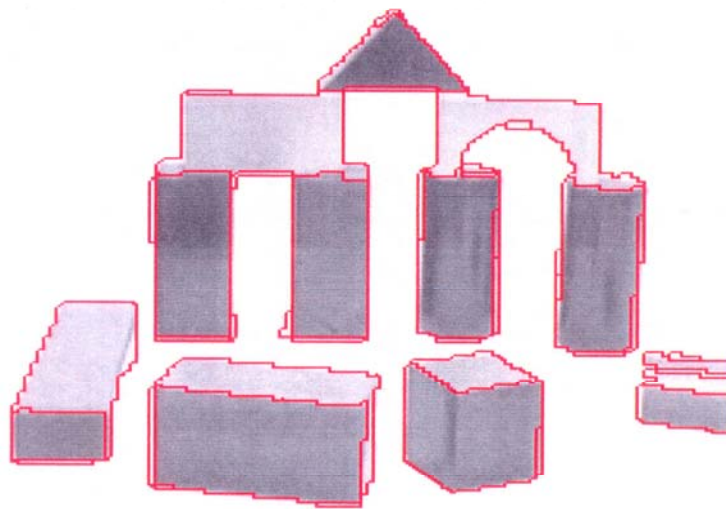
En las figuras I.4 y I.5 se puede ver el resultado de esta técnica con las modificaciones citadas, aplicada a una imagen del mundo de bloques y a otra de piezas industriales. En estos casos el predicado de uniformidad utilizado en el proceso de división modeliza las regiones de una imagen como una superficie plana a la que se intenta ajustar un plano. Como medida de calidad del ajuste se ha empleado la correlación normalizada [Lore-94]. En el proceso de división se utiliza una medida de similaridad entre regiones que pondera la diferencia de niveles promedio con la orientación relativa de los planos ajustados a cada región.

El uso de este predicado de división permite reducir el efecto de dientes de sierra que aparece en los bordes de las regiones. Sin embargo, para evitar una sobresegmentación de la imagen, en el proceso de fusión el predicado de uniformidad empleado no es tan riguroso. Por último, se ha aplicado al resultado del proceso de fusión, un filtrado con el fin de eliminar pequeñas regiones sin significado en la escena. Estas modificaciones permiten la eliminación de la fase de agrupamiento del algoritmo original [Lore-94]. Esto se consigue debido a que la fase de división se hace antes que la de fusión, y en esta última en lugar de unir regiones que posean el mismo nodo padre en el árbol, se unen regiones que siendo adyacentes en la imagen, su fusión de como resultado una región uniforme.

Además de la técnica de división y fusión existen otras técnicas que también se basan en la utilización de un árbol como estructura de datos principal. Tal es el caso de las pirámides adaptativas, una técnica propuesta por Montanvert [Mont-91][Moli-92] en la que se utiliza un árbol donde cada nodo posee un número de nodos hijos no constante, a diferencia de árbol cuaternario donde cada nodo posee cuatro nodos hijo.



a)

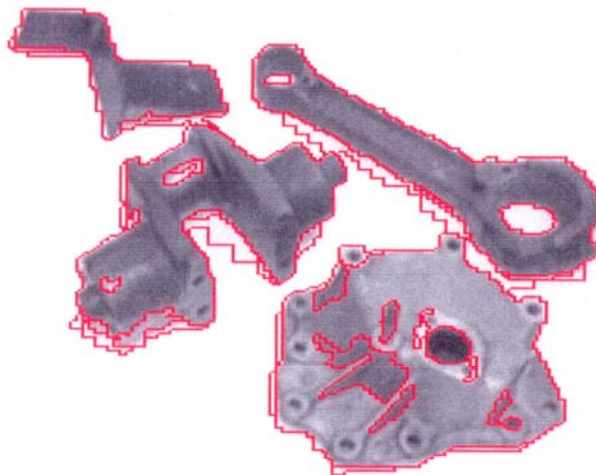


b)

Figura I.4 Ejemplo de segmentación utilizando la técnica de división y fusión, con las modificaciones realizadas sobre el método original, aplicada a una escena del mundo de los bloques. a) Imagen original. b) Resultado de la segmentación.



a)



b)

Figura I.5 Ejemplo de segmentación utilizando la técnica de división y fusión, con la modificaciones realizadas sobre el método original, aplicada a una escena con piezas industriales. a) Imagen original. b) Resultado de la segmentación.

I.3.2.4 SEGMENTACION DE REGIONES ITERATIVA. RELAJACION.

Las técnicas de relajación fueron originalmente utilizadas por Waltz para la descripción de sólidos y más tarde extendidas por Zucker y Rosenfeld a una amplia variedad de aplicaciones. El método se fundamenta en la utilización de técnicas locales y de modificación de la asignación de un pixel a una cierta clase de pixel desde las asignaciones de sus vecinos en un proceso iterativo convergente.

Para ilustrar el proceso seguiremos el esquema propuesto en [Rose-82] donde se parte de un conjunto de n pixels, $\{A_1, \dots, A_n\}$, que se desean clasificar en m clases $\{C_1, \dots, C_m\}$. Se asume interdependencia en las asignaciones de pixels a clases, es decir, para cada par de asignaciones a clases $A_i \in C_j$ y $A_h \in C_k$, se dispone de una medida cuantitativa de la compatibilidad de este par, $c(i,j,h,k) \in [-1,+1]$. Si $p_{ij}^{(0)}$ es una estima inicial de la probabilidad de que $A_i \in C_j$, Rosenfeld propone un método iterativo para computar las estimas de probabilidad $p_{ij}^{(r)}$ ($r=1,2,\dots$) a partir de las probabilidades iniciales y las compatibilidades. Este esquema esta basado en ajustar los valores actuales de cada p_{ij} a partir de los valores en la iteración anterior de las otras "p" y de los valores de las "c". Intuitivamente podemos pensar que si p_{hk} es alto y $c(i,j,h,k)$ es asimismo alto, deberemos incrementar p_{ij} puesto que es compatible con la alta probabilidad del evento $A_h \in C_k$. Análogamente ocurre si $c(i,j,h,k)$ es baja, en cuyo caso deberemos hacer decrecer p_{ij} . Dentro de las posibles alternativas [Pavl-77][Rose-82], Rosenfeld plantea el esquema de promedios siguiente para realizar la iteración $r+1$:

$$p_{ij}^{(r+1)} = \frac{p_{ij}^{(r)}(1+q_{ij}^{(r)})}{\sum_{j=1}^m p_{ij}^{(r)}(1+q_{ij}^{(r)})} \quad (I-22)$$

Donde:

$$q_{ij}^{(r+1)} = \frac{1}{n-1} \sum_{h=1, h \neq i}^n \left[\sum_{k=1}^m c(i,j,h,k) p_{hk}^{(r)} \right] \quad (I-23)$$

Para completar el esquema iterativo es necesario definir los coeficientes de compatibilidad. En general si se utiliza el proceso de relajación en la clasificación de pixels, sólo se tendrá en cuenta el entorno del pixel dado. También puede considerarse que los distintos coeficientes de compatibilidad son espacialmente invariantes, esto es, dependerán solamente de la posición relativa del pixel. Estos coeficientes pueden ser definidos a partir del punto de aplicación de muchas formas [Rose-82], incluyendo no linealidades, direccionalidad, etc. El final del proceso de iteración se establece desde medidas de variación de las asignaciones de la partición, bien mediante funcionales basados en la energía o entropía de la imagen final o en el cómputo de los cambios netos producidos.

Un subconjunto importante de algoritmos de segmentación que se engloban dentro de este grupo son aquellos que utilizan modelos de interacción espacial como los campos de Markov aleatorios (Markov Random Fields, MRF) o los campos de Gibbs aleatorios (Gibbs Random Fields, GRF) para modelizar la imagen [Gema-84][Marroq-87][Dube-90][Geig-91]. Por su propia concepción, estos métodos encuentran su mejor campo de aplicación en problemas de segmentación asociados con la discriminación entre texturas, en los que las imágenes son susceptibles de ser modeladas como procesos estocásticos.

I.4 SISTEMAS EXPERTOS PARA ANÁLISIS DE IMÁGENES.

No existe en el área de la Visión Artificial una metodología de análisis de imágenes que sea decididamente superior a otros en cualquier contexto. Como se ha visto en los epígrafes anteriores, se han ideado numerosas técnicas con características y entornos de aplicación muy diversos, surgidos, con frecuencia, de trabajos realizados para resolver problemas de visión en dominios concretos. Por otra parte, es frecuente que los métodos contengan parámetros libres que deben ser fijados por el usuario, sin que exista una definición explícita de la relación de estos parámetros con las características de las imágenes utilizadas como datos. En definitiva, no disponemos en este campo de una metodología basada en principios fundados que permitan establecer qué algoritmos deben emplearse para resolver determinadas tareas y cómo deben ajustarse sus parámetros.

Existe, en cambio, una amplia experiencia heurística que posibilita a los expertos en este campo estimar qué técnicas y con qué parámetros es factible acometer un determinado problema de visión, en función de la naturaleza de las imágenes y de las restricciones impuestas por los objetivos del análisis. Esto ha propiciado, en los últimos años, el desarrollo de sistemas expertos para análisis de imágenes capaces, en mayor o menor medida, de seleccionar un algoritmo o una secuencia óptima de algoritmos de proceso de imágenes, de inicializar sus parámetros a partir de una descripción de los objetivos y de las imágenes a tratar, y ajustar luego estos parámetros evaluando el resultado obtenido. Para cumplir estos objetivos, los sistemas expertos para análisis de imágenes deben ser capaces de proporcionar soluciones a lo siguientes problemas [Mats-90][Clém-93]:

1. *Determinación de la calidad de imagen.* La medida y descripción de la calidad de una imagen es una tarea difícil, usualmente fundada en el dominio de la señal, que en determinados casos puede ser definida globalmente, para toda la imagen, y en otras requerir una determinación local al variar a lo largo de la imagen.
2. *Selección de los operadores apropiados.* Con frecuencia, una cierta tarea de análisis de imágenes puede ser realizada por diferentes operadores (métodos/técnicas). La selección del operador óptimo dependerá de los objetivos de la tarea, la calidad de la imagen y de las características del operador.
3. *Determinación de los parámetros óptimos.* El funcionamiento de los diversos operadores en las tareas de análisis suele estar parametrizado. Su idoneidad será, en general, muy dependiente de la selección de los parámetros.
4. *Combinación de operadores elementales.* Normalmente, es necesario combinar varios operadores simples para realizar una cierta tarea. Por ejemplo, una forma común de extraer regiones en una imagen es realizar una detección de bordes, a continuación conectar los bordes y, finalmente, detectar los contornos cerrados.
5. *Ensayos de prueba-y-error.* La adecuación de los operadores y parámetros seleccionados no puede, en general, establecerse a priori, por lo que es necesario ajustar su selección en base a ensayos de prueba-y-error.
6. *Evaluación de los resultados del análisis.* La robustez y el grado de autonomía de este

tipo de sistemas dependerán en gran medida de su capacidad de evaluar los resultados producidos. Es éste un requisito indispensable para poder establecer un mecanismo de realimentación que permita bien el refinamiento del análisis o la adaptación del sistema a un entorno cambiante mediante la selección de nuevos parámetros y/o nuevos operadores.

Nótese, incidentalmente, que la evaluación de una segmentación dentro del propio módulo de segmentación es un problema difícil, por cuanto la bondad de una segmentación puede depender de aspectos perceptuales que no pueden ser definidos con facilidad en este nivel.

Los sistemas expertos para proceso de imágenes pueden ser clasificados grosso modo en cuatro categorías no exclusivas [Mats-90]:

1. Sistemas consultores para proceso de imágenes.
2. Sistemas basados en conocimiento para generación de programas de proceso de imágenes.
3. Sistemas de segmentación de imágenes guiados por objetivos.
4. Sistemas basados en reglas para segmentación de imágenes.

I.4.1 SISTEMAS CONSULTORES PARA PROCESO DE IMÁGENES.

Se incluyen en este apartado el conjunto de sistemas de proceso de imágenes que son capaces de asistir a un usuario inexperto en la selección de los operadores idóneos y de sus parámetros utilizando un sistema de interacción basado en consultas. *EXPLAIN* [Tana-88] es un sistema experto para proceso de imágenes en el que el usuario describe el problema mediante una serie de palabras claves, como *mejora (enhancement)* o *segmentación*, y califica las propiedades de la imagen a procesar (color, monocromática, contraste, nivel de ruido,...). El sistema descompone la tarea en el encadenamiento de una serie de etapas de proceso de imágenes utilizando reglas de producción. Los algoritmos a ejecutar en cada etapa y sus parámetros se seleccionan por interacción con el usuario que detalla el objetivo y la calidad de la imagen. El sistema ejecuta el algoritmo y muestra los resultados al usuario. En caso de que los resultados no sean evaluados positivamente por el usuario, se seleccionan otros parámetros y eventualmente

otro algoritmo equivalente. Una característica notable de este sistema es su capacidad para aprender por interacción con un experto.

I.4.2 SISTEMAS BASADOS EN CONOCIMIENTO PARA GENERACIÓN DE PROGRAMAS DE PROCESO DE IMÁGENES.

El objetivo de este tipo de sistemas es la generación automática de programas mediante la utilización de librerías de procedimientos de proceso de imágenes. El sistema basa su funcionamiento en el conocimiento de las características de los módulos de la librería, tipos de datos que utiliza y produce, etc. El usuario, en general, sólo tiene que escribir un programa abstracto de especificaciones sin necesidad de conocer los detalles de los módulos contenidos en las librerías de procesos.

En esta categoría de sistemas se incluye *Expert Assistant* [John-87], un sistema de generación de comandos para análisis de imágenes astronómicas. Es independiente del subsistema de proceso de imágenes utilizado, actuando de facto como una interface inteligente entre este último y el usuario. Descrito en un problema de calibración de una cámara CCD, genera una secuencia de comandos de proceso a partir de una descripción simbólica de los datos (tipos de datos, modos de operación del instrumento y etapas del análisis) hecha por el astrónomo. Este sistema no hace en ningún momento uso de los datos.

Toriu et al. [Tori-87] han desarrollado un sistema de propósito general orientado a usuarios no expertos que permite asistirles en el desarrollo de algoritmos para aplicaciones específicas mediante un paquete de proceso de imágenes denominado FIVIS. Para un objetivo predefinido (tal como: segmentación, mejora_de_calidad o extracción_de_características) y una descripción simbólica de los atributos de la imagen (tipo de imagen, textura, nivel de ruido,...), el sistema selecciona mediante reglas de producción una estrategia de procesamiento. Los módulos de proceso que la integran, descritos mediante frames en la base de conocimiento, se seleccionan por comparación entre sus características y las de la imagen. El sistema ejecuta el algoritmo y muestra los resultados al usuario pero no los evalúa.

El sistema DIA-ES (Digital Image Analysis - Expert System) [Tamu-84] permite de manera semiautomática diseñar algoritmos para aplicación determinadas. En [Sato-88] se ilustra su uso en el desarrollo de algoritmos de reconocimiento de formas. DIA-ES se compone de dos subsistemas: el procesador sintáctico y el procesador semántico. Para una imagen y un problema determinado, el procesador semántico construye un árbol de operaciones que describe

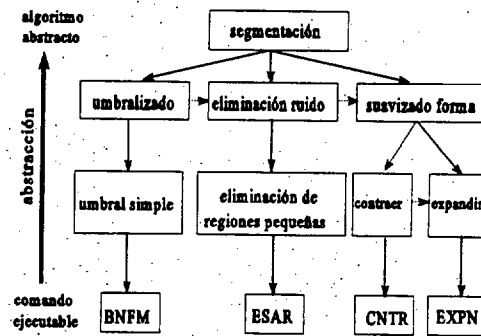


Figura I.6. Árbol de operaciones [Tamu-84].

el proceso de análisis de la imagen. Los diferentes niveles del árbol son equivalentes a niveles de abstracción de la tarea a realizar (ver figura I.6), correspondiéndose las hojas del árbol con llamadas a los módulos contenidos en la librería de procedimientos. Todos los mecanismos de inferencia para la generación de un plan y su modificación se asocian con operaciones de manipulación de la estructura del árbol de operaciones. El usuario es quien lleva a cabo la evaluación de los resultados y el ajuste de parámetros. El procesador semántico produce como resultado una secuencia de comandos que es interpretada por el procesador sintáctico [Saka-85] para generar un programa optimizado. Este procesador sintáctico está concebido para utilizar la librería de proceso de imágenes SPIDER [Tamu-83].

En [Bail-88] se describe otro sistema experto para facilitar el desarrollo de algoritmos para proceso de imágenes a usuarios inexpertos. En una primera fase, el sistema interroga al usuario para determinar la naturaleza de la tarea a desarrollar. A continuación, selecciona de su base de conocimiento y mediante reglas un plan que descompone la tarea en una serie de subobjetivos o etapas de procesado. Para cada etapa, el sistema selecciona un operador y lo aplica sobre los datos. El usuario juzga los resultados de cada operador; y si la evaluación es negativa, se intenta otro operador o secuencia de operaciones. Por último, cuando todos los resultados son satisfactorios, el sistema genera el algoritmo correspondiente.

CONNY [Lied-92] es un sistema que genera automáticamente un programa de análisis de imágenes basado en regiones y líneas. Utiliza una definición de objetivos dada por el usuario, un conjunto de imágenes test y el conocimiento sobre los operadores y las estrategias de proceso de imágenes contenidos en su base de conocimiento. El resultado es un algoritmo, con los mejores parámetros posibles, definido como un encadenamiento de operadores. Un aspecto destacado de este sistema es que explora todas las posibles estrategias contenidas en la base de conocimiento que sean adecuadas para cubrir el objetivo fijado, seleccionando al final, la mejor estrategia. Una parte fundamental de *CONNY* es su subsistema de evaluación [Blöm-89][Lied-91] fundamentado en el proceso de imágenes test cuyas propiedades se expresan en términos de lenguaje natural.

OCAPY [Clém-93] es un sistema experto diseñado para automatizar tareas de procesamiento de imágenes independientemente del dominio de la aplicación y del software para proceso de imágenes utilizado. Un aspecto interesante de *OCAPY* es el modelo de representación del conocimiento que utiliza, basado en la identificación de los conceptos relevantes (*objetivo, operador, contexto, petición, datos o argumentos de los operadores, selección de operadores, evaluación de resultados, inicialización de parámetros y ajuste de parámetros*) y sus relaciones. El conocimiento estático o descriptivo (*objetivos, operadores, contextos, peticiones y argumentos de los operadores*) se representa por frames. El conocimiento heurístico, que interviene en la selección y evaluación de operadores, y en la inicialización y ajuste de parámetros, se representa mediante reglas de producción. Las reglas de producción se organizan en pequeñas bases en función de su semántica (reglas de selección de operadores, reglas de evaluación, etc...) que se asocian a los frames con los que guardan alguna relación. Por ejemplo, las reglas contenidas en un frame que representa a un determinado operador serán las que sirvan para inicializar y ajustar sus parámetros exclusivamente. El modo de operación es similar a los otros sistemas descritos y comprende tres fases. En la primera de ellas, se elabora un plan a partir del objetivo fijado por el usuario, quien también especifica el contexto de la imagen para guiar al sistema en la elaboración del plan. La segunda fase controla la ejecución del plan mediante ensayos de tipo prueba-y-error. La última fase comprende la evaluación de los resultados y, si es necesario, la modificación de los parámetros. Cuando un operador fracasa después de que sus parámetros hayan recorrido el rango de valores permitidos sin alcanzar una evaluación positiva, el sistema posee un mecanismo de

replanificación que posibilita la generación de una nueva secuencia de operadores. PROGAL es un sistema experto dedicado a la clasificación automática de galaxias que emplea a OCAPÍ como subsistema de proceso de imágenes. El objetivo es automatizar la clasificación de galaxias empleando criterios morfológicos a partir de imágenes obtenidas con diferentes sensores y telescopios. La diversidad de las imágenes requiere que el procesamiento (secuencia de operadores y selección de parámetros) se acomode al tipo y características de cada imagen.

Un campo de interés Visión Artificial es la elaboración automática de programas de reconocimiento de objetos basados en modelos. En este tipo de problemas, los sistemas de reconocimiento se componen básicamente de dos elementos: un "motor" de reconocimiento y una librería de modelos. El motor de reconocimiento reconoce los objetos en una imagen comparando características extraídas de esta imagen con las contenidas en las descripciones de los modelos presentes en la librería de modelos. Para ello puede emplear numerosas técnicas que no describimos aquí por razones de extensión (véase por ejemplo [Grim-90]). Para cada objeto, la librería de modelos contiene una descripción en términos de las características útiles para su localización en una imagen. La generación de la descripción de los modelos es considerado hoy en día como un problema fundamental que actúa de cuello de botella, limitando la aplicación de los sistemas de visión basados en modelos en entornos industriales [Niem-90b][Chen-92]. Los trabajos en este campo se orientan hacia la generación automática de librerías de modelos a partir de descripciones CAD o de vistas de los objetos [Goad-83][Conn-87][Ikeu-87][Hans-89][Chen-92], y hacia la generación de programas de reconocimiento (vision programming) [Ikeu-88][Ikeu-91] definido un cierto motor de reconocimiento.

I.4.3 SISTEMAS DE SEGMENTACIÓN DE IMÁGENES GUIADOS POR OBJETIVOS.

Dentro de este apartado se incluyen sistemas como GOLDIE [Kohl-87][Kohl-88] y LLVE [Mats-90], diseñados para ser integrados en sistemas de interpretación de escenas como subsistemas de segmentación y capaces de atender peticiones de segmentación de regiones o líneas expresadas en términos de alto nivel (p.e. "busca una región triangular de color rojo"). Como los sistemas expertos de análisis de imágenes mencionados anteriormente, éstos incorporan conocimiento sobre los tipos de imágenes y sobre los operadores que es posible aplicar para cubrir

un objetivo; así como un conjunto considerable de reglas heurísticas que posibilitan la composición de estrategias de combinación de operadores y de selección y modificación de parámetros. GOLDIE, una de las fuentes de conocimiento del nivel intermedio operativas en el Schema System, realiza resegmentaciones de regiones o líneas actuando localizadamente sobre zonas de la imagen. LLVE, que representa una parte esencial del sistema SIGMA, se describirá en el apartado I.5.3.

I.4.4 SISTEMA BASADO EN REGLAS PARA SEGMENTACIÓN DE IMÁGENES.

Un sistema paradigmático en cuanto a la explicitud del conocimiento utilizado es el descrito como un sistema experto en la segmentación a bajo nivel por [Nazi-84][Levi-85a][Levi-85b]. La denominación de bajo nivel no está referida a las herramientas utilizadas que son reglas de producción con muy alto nivel simbólico, sino más bien a que la segmentación está conducida por los datos, es decir por las propiedades de las imágenes mismas, en contraposición de los sistemas conducidos por modelos en los que los procesos de segmentación están conducidos por tareas de visión jerárquicamente superiores a la de segmentación.

Los objetivos en este caso se centran en la construcción de un sistema que cumpla básicamente tres condiciones: la primera es que integre un número considerable de heurísticas de segmentación, la segunda es que el conocimiento heurístico que guía la evolución de la segmentación debe figurar en una forma totalmente explícita, y la tercera es que se incluyan parámetros que definan la calidad de la segmentación y que gobiernen las estrategias del sistema.

El sistema se organiza en la forma que se muestra en la figura I.7. Las reglas integran la base de conocimiento o memoria permanente y se encuentran organizadas por el tipo de conocimiento que representan: sobre líneas, sobre regiones, etc. Al comenzar la segmentación se construyen en la base de datos o memoria volátil mapas de regiones y líneas. Un módulo supervisor coordina la selección de las estrategias de control, conjunto de reglas, regla y foco de atención, y un intérprete de reglas intenta disparar alguna de las reglas sobre el foco de atención. El foco de atención puede ser una línea, una región o un área compuesto de varias regiones. Cuando se produce el disparo de una regla, y en función de su naturaleza, la acción

desencadenada resultará en la invocación de módulos especializados en tareas de bajo nivel referidas a líneas, regiones o áreas. Estos módulos provocan la modificación/actualización de la memoria volátil.

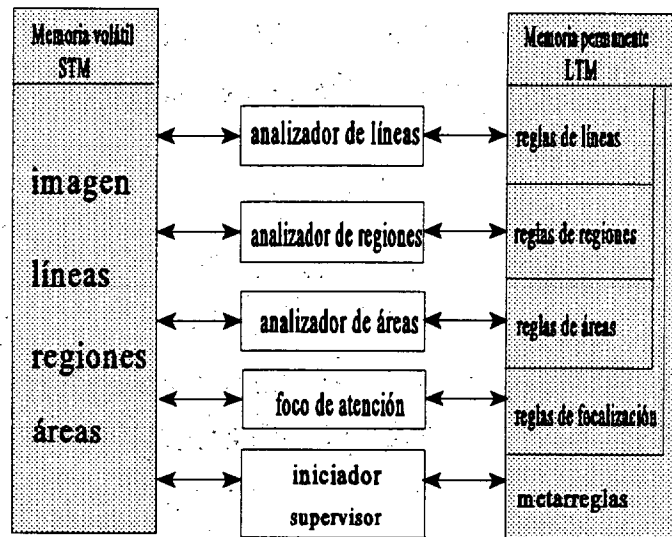


Figura I.7. Organización del sistema experto para segmentación de Nazif y Levine.

Como ya se ha indicado, uno de los objetivos perseguidos por este sistema es el de facilitar la explicitud del conocimiento heurístico utilizado. Con tal fin, todas las declaraciones del sistema se realizan mediante reglas sencillas y regulares en la forma:

SI CONDICIÓN Y CONDICIÓN ... Y CONDICIÓN ENTONCES ACCIÓN.

Dentro de CONDICIÓN pueden incluirse diversas condiciones lógicas referidas a entidades de la imagen y a sus adyacentes espaciales. Las entidades consideradas en las reglas son: regiones, líneas y áreas. Las áreas se consideran como agrupaciones de regiones de tal forma que se clasifican en alguno de los siguientes tipos: "uniforme", "de textura" o "delimitada por líneas". Dentro del campo de ACCIÓN se incluye la ejecución de diversos procedimientos de modificación o evaluación de la segmentación. A título de ejemplo mostramos las siguientes reglas. La primera de las cuales es utilizada para el análisis de regiones, mientras que la siguiente lo es para agrupación de líneas:

REGLA(906):

- SI: (1) El **ÁREA** de la **REGIÓN** es **ALTA**
(2) El **GRADIENTE PROMEDIO** de la **REGIÓN** es **NO BAJO**
(3) El **HISTOGRAMA** de la **REGIÓN** es **BIMODAL**

ENTONCES: (1) **DIVIDE** la **REGIÓN** en función del **HISTOGRAMA**

REGLA(1503):

- SI: (1) El **FINAL** de la **LÍNEA** es **ABIERTO**
(2) La **DISTANCIA** a la **LÍNEA** de **ENFRENTA** es **NO ALTO**
(3) Las **LÍNEAS** tienen **IGUAL DIRECCIÓN PROMEDIO**

ENTONCES: (1) **UNE** las **LÍNEAS** mediante **EXPANSIÓN ADELANTE**

Las reglas utilizadas por el sistema se dividen en varios grupos, entre los que destacamos los de análisis y control. El primer grupo a su vez se divide en las relativas a regiones, líneas y áreas. El segundo grupo de control incluye reglas para definir el foco de atención, es decir la entidad sobre la que se centra el proceso en el próximo ciclo del sistema, así como reglas de estrategias que controlan las acciones que deben realizarse. Igualmente se incluyen meta-reglas o reglas que controlan los conjuntos de reglas que deben aplicarse. El siguiente es un ejemplo de regla de foco de atención:

REGLA(201):

- SI: (1) El **GRADIENTE** de la **LÍNEA** es **ALTO**
(2) La **LONGITUD** de la **LÍNEA** es **ALTA**
(3) La **MISMA REGIÓN** a **IZQUIERDA** y **DERECHA**

ENTONCES: (1) **TRABAJA** sobre la **REGIÓN** a la **IZQUIERDA** de la **LÍNEA**

En esta regla se codifica el hecho de que trabajando en una línea larga y de alto gradiente, es decir una línea de alta fiabilidad, detectamos que se encuentra inmersa en una región. En tal caso juzgamos que posiblemente la región este mal segmentada, y procedemos seguidamente a trabajar sobre tal región para proceder a una segmentación posiblemente más adecuada.

Las meta-reglas son las encargadas de lanzar procesos de inicialización al comienzo de la segmentación. Por ejemplo:

META-REGLA(1):	
SI:	(1) NO EXISTEN REGIONES
ENTONCES:	(1) INICIALIZA REGIONES
META-REGLA(2):	
SI:	(1) NO EXISTEN LÍNEAS
ENTONCES:	(1) INICIALIZA LÍNEAS
META-REGLA(3):	
SI:	(1) NO EXISTEN ÁREAS
ENTONCES:	(1) GENERA ÁREAS

Las meta-reglas crean también la estrategia de control adecuada. Por ejemplo, si existe un área integrada por regiones que en promedio presentan una baja uniformidad, entonces la prioridad de las reglas del analizador de regiones se ajusta para otorgar mayor prioridad a las reglas de división frente a las reglas de fusión de regiones. Las meta-reglas paran el análisis una vez que se ha completado la segmentación.

Otra de las características destacables del sistema es la utilización de un vector de parámetros de calidad asociado a cada área y que se utiliza con el tipo de éstas para gobernar las estrategias del sistema. El vector de parámetros de calidad P se define como sigue:

$$P = [U_1, \dots, U_m, C_1, \dots, C_m, H_1, \dots, H_m, T, R, L]$$

La dimensionalidad del vector es $M=3m+3$ donde m es el número de características de la imagen, básicamente las componentes de color. Las componentes U_i del vector de calidad son medidas de uniformidad de las regiones del área. Las componentes C_i se corresponden con medidas de contraste de región, mientras que las medidas H_i lo son del contraste de las líneas inmersas en el área. Las restantes componentes H, R, L se corresponden respectivamente con la conectividad de líneas, el número normalizado de regiones del área y el número normalizado de líneas en el área.

I.5 SISTEMAS DE INTERPRETACIÓN DE IMÁGENES.

La interpretación de una imagen es un proceso que utiliza conocimiento en cada etapa [Jain-91] el cual puede referirse tanto a las propiedades de las entidades como a la forma de realizar los procesos. Se precisa conocimiento referente a las condiciones del entorno e iluminación, a la geometría y propiedades de los objetos, a procesos de formación de imágenes, *y aquel relativo a los operadores de proceso con sus propiedades y ámbitos de aplicación.* La utilización de conocimiento es general en casi todos los sistemas de visión, pero una de las características distintivas es el grado en que tal conocimiento se encuentra en forma explícita, es decir, no integrado en los propios procesos sino externos a ellos y además en formatos altamente modulares y próximos al lenguaje humano. En estos últimos el conocimiento no forma parte del sistema, sino que el sistema lo utiliza, de tal forma que la modificación del conocimiento no modifica la arquitectura del sistema, en contraposición a aquellos otros en los que el conocimiento está implícito.

Los primeros sistemas de interpretación de imágenes (IUS), en sintonía con las ideas de Marr [Marr-82], que defendía la hipótesis de que las primeras etapas del procesamiento visual debían estar conducidas exclusivamente por los datos, consideraron la segmentación de imágenes casi como una tarea de preproceso que debía discurrir sin intervención del conocimiento acerca del dominio de la escena. Se enfatizaban aspectos de más alto nivel como la representación del conocimiento más idónea, el control de la interpretación, la combinación de evidencias, etc. En ciertos sistemas, como en SPAM y ACRONYM, incluso, la segmentación se realizaba externamente al sistema de interpretación mediante un programa genérico de segmentación que era absolutamente independiente del propio IUS. Pronto se constató que esta concepción de la segmentación como una mera etapa de preproceso limitaba muy seriamente el rendimiento del IUS. Esta observación propició el que en los sucesivos diseños se considerara la segmentación como una de las tareas más importantes a realizar. Se han investigado diferentes mecanismos que posibilitan el que el módulo de segmentación pueda interactuar con los niveles superiores del IUS; mecanismos que permitan, en definitiva, orientar la segmentación en base al conocimiento de alto nivel activado en función del estado de la interpretación. Realizaciones concretas de estas ideas

son los sistemas de segmentación guiados por objetivos presentes en sistemas como SIGMA y SCHEMA.

De acuerdo con el planteamiento que, sobre el desarrollo de este capítulo, hacíamos en la introducción, estudiaremos ahora algunos de los sistemas de interpretación de imágenes más conocidos. El objetivo es conocer cómo se organiza y planifica el control de un IUS para que desde niveles superiores pueda dirigirse la segmentación utilizando conocimiento de alto nivel. Con este fin, describiremos los diferentes sistemas de manera estructurada, considerando en cada caso la arquitectura del sistema, la representación del conocimiento que emplea y cómo se organiza el control de la interpretación.

I.5.1 ACRONYM.

El sistema de interpretación de imágenes ACRONYM [Broo-79b,83,84] aborda el problema, actualmente abierto, de utilizar modelos tridimensionales de objetos descritos con independencia del punto de vista para dirigir su localización e identificación en una imagen. Aspectos destacados de ACRONYM son la representación del conocimiento utilizada para la descripción de los modelos y la forma en que se realiza el razonamiento geométrico [Broo-81] por medio de restricciones sobre parámetros de los modelos. ACRONYM ha sido aplicado en la tarea de identificar modelos de aviones en imágenes aéreas de aeropuertos.

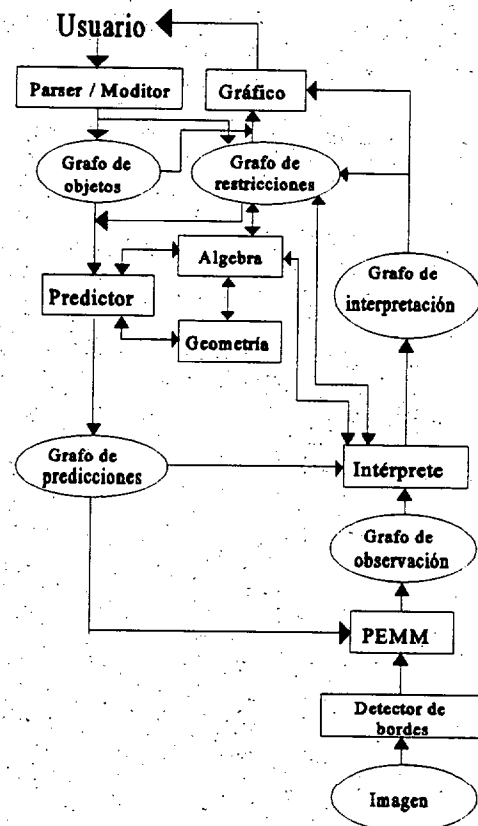


Figura I.8. Organización de ACRONYM.

I.5.1.1 ARQUITECTURA.

La arquitectura de ACRONYM está basada en una concepción de los sistemas de visión basados en modelos y contiene cuatro componentes: modelado, predicción, descripción e interpretación. La figura I.8 muestra la organización de ACRONYM.

a) Modelado.

El sistema consta de una parte orientada a la definición de los objetos a reconocer. Esta definición puede realizarse bien mediante un lenguaje de descripción geométrico que resulta adecuado para describir las clases de objetos y sus relaciones espaciales (Parser, en la figura), o mediante un editor de sólidos (Moditor), más adecuado para el modelado geométrico de objetos específicos. Ambos producen la misma representación para los objetos. Un módulo de representación gráfica 3D (Graphics) permite visualizar la definición de los modelos.

Los objetos se describen mediante el uso de conos generalizados [Binf-71][Neva-82]. Las clases de objetos se definen por conjuntos de restricciones sobre los parámetros geométricos de los modelos. Estas restricciones se expresan en forma de desigualdades algebraicas y se describen con más detalle en el apartado dedicado a la representación del conocimiento. La definición volumétrica de los modelos, tipo de cono generalizado y parámetros, y sus relaciones espaciales definen el *grafo de objetos*. Los elementos de volumen, las formas simples, definen los nodos del grafo, mientras que las relaciones espaciales y las relaciones de subpartes definen los arcos. Las relaciones entre clases de objetos se representan por un *grafo de restricciones*, en el que los nodos son conjuntos de restricciones impuestas sobre la definición de los elementos de volumen (conos generalizados), y los arcos dirigen la inclusión de subclases.

b) Predicción.

El módulo de predicción (Predictor, Geometría y Algebra) utiliza técnicas de razonamiento geométrico para predecir la apariencia de características observables de los modelos que sean invariantes con la posición. Por ejemplo, si una característica es colineal en el modelo, también lo será en la imagen, de esta forma, características que sean paralelas en el modelo

también serán paralelas en la imagen si son observables. La conectividad es otro invariante que se utiliza en la predicción. El módulo de predicción no predice la apariencia de un objeto desde todos los posible puntos de vista, sino predice la aparición de determinadas características que permitan identificar los objetos y determinar su posición y orientación. Esta predicción se efectúa desde el seguimiento de los niveles de descripción utilizados en el grafo de objetos, de manera que sólo se realiza la predicción sobre una subparte de un objeto cuando la parte principal ha sido hipotetizada. En concreto, la predicción se realiza para cada cono generalizado en las cinco etapas siguientes:

1. Se predicen todos los contornos posibles del cono generalizado por medio de un conjunto de reglas.
2. Se examina la posible orientación del cono generalizado relativa a la cámara para decidir qué contornos serán visibles y cómo se proyectarán en la imagen.
3. Se establecen relaciones espaciales entre los contornos visibles.
4. Se predicen las formas que generarán los contornos.
5. Se determinan cuáles deben ser las restricciones a verificar durante el proceso de interpretación.

Una vez que se han predicho las formas correspondientes a cada cono generalizado, las relaciones entre formas producidas por conos independientes se predicen utilizando los mismos mecanismos del punto 3.

El módulo de predicción produce dos resultados. El primer de ellos es un programa que guía la acción del módulo de bajo nivel (PEMM, Prototype Edge Mapping Module), detalla los parámetros necesarios y un conjunto de funciones heurísticas que guían la obtención de franjas (ribbons) y elipses a partir de contornos, principal cometido de este módulo. El otro resultado es el *grafo de predicción*. Los nodos de este grafo son predicciones de características en la imagen, y los arcos especifican las relaciones que deben verificarse entre ellos en la imagen.

c) Descripción.

La descripción de la imagen se realiza en dos partes. La primera es la obtención de bordes rectos mediante el algoritmo descrito en [Neva-80]. Sobre este resultado opera el módulo PEMM que emplea el programa producido por el módulo predictor, como ya se ha indicado. El objetivo es obtener una descripción de las formas presentes en la imagen a partir de los contornos. Para ello, primero se encadenan contornos y, a continuación, se intentan detectar franjas y elipses por ajuste a los contornos producidos [Broo-79a]. Éstas son las primitivas de forma que se utilizan para producir un grafo de la misma naturaleza que el grafo de predicción, el *grafo de observación*, donde los nodos representan las formas detectadas, y los arcos las relaciones espaciales entre ellos.

d) Interpretación.

La interpretación se efectúa desde la comparación de los grafos de predicción y observación. Esta comparación no se realiza por ajuste de las medidas de las características, extraídas de la imagen a las contenidas en el grafo de predicción, sino que se utilizan las medidas obtenidas de la imagen para imponer restricciones sobre los parámetros de los modelos tridimensionales cuya presencia en la imagen se hipotetiza. El sistema de manipulación de restricciones (Algebra, en la figura I.8) se invoca para comprobar que las restricciones que se intentan imponer en la interpretación son consistentes con lo que ya se ha descubierto sobre los modelos. Las restricciones que resultan ser coherentes se introducen en el grafo de restricciones para ser utilizadas en sucesivas interpretaciones. Esto se hace realiza de hecho por el manejador de restricciones mediante la estimación de nuevos límites que acoten el rango de variación de los parámetros de los modelos. De esta forma, interpretaciones locales restringen los parámetros de cámara y el tamaño del objeto o solamente relaciones entre éstos. Las interpretaciones locales se combinan para producir interpretaciones más globales si todas las restricciones impuestas por las primeras son consistentes entre sí. A medida que los objetos son identificados se introducen nuevas restricciones al extenderse el análisis (predicción) a detalles más finos de los objetos.

I.5.1.2 REPRESENTACIÓN DEL CONOCIMIENTO.

El conocimiento empleado por ACRONYM es de dos tipos. El primero, específico al dominio del problema, describe los objetos tridimensionales a identificar en las imágenes por medio de frames [Mins-75]. El segundo, permite manipular geoméricamente los modelos y predecir cómo se proyectan en la imagen bidimensional los objetos tridimensionales. Para expresar este conocimiento que es independiente del dominio de la aplicación se emplean reglas de producción.

La representación de los objetos se realiza de forma jerárquica, procediendo desde una descripción grosera de la forma de un clase de objetos hasta los detalles más finos de objeto concreto. El formalismo utilizado es el representar de los objetos en un grafo de objetos, donde cada nodo es un frame que contiene la descripción de un objeto o de alguna de sus partes, y sus relaciones espaciales. Como ya se ha indicado, ACRONYM utiliza conos

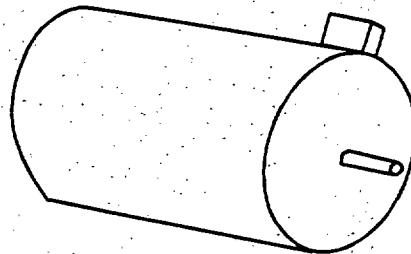


Figura I.9. Modelo de motor eléctrico.

generalizados para describir las formas tridimensionales. Un cono generalizado se define por un eje, una sección recta perpendicular a éste y una regla (regla de barrido) que expresa la forma en que la sección recta varía al recorrer el citado eje. Así, por ejemplo, el cuerpo de un motor eléctrico simple, como el mostrado en la figura I.9, se podría definir como un frame:

Nodo: MOTOR_ELECTRICO
Clase: CONO_SIMPLE
Eje: RECTO_LONGITUD_8
Regla de barrido: CONSTANTE
Sección recta: CIRCULO_RADIO_2.5

Este nodo así definido describiría un modelo concreto de motor eléctrico. Es posible, en cambio establecer un modelo más general de motor eléctrico utilizando dos mecanismos. De una

Capítulo I

parte, se pueden declarar los slots del frame como otros frames, De otra, *cuantificadores* en vez de constantes para especificar las dimensiones. Los cuantificadores toman la forma de desigualdades, las cuales se conservan en el grafo de restricciones. Por ejemplo, si la longitud del motor está comprendida entre 6.0 y 9.0 unidades de longitud y el radio entre 2.0 y 3.0:

$$6.0 \leq \text{LONGITUD_MOTOR} \leq 9.0$$
$$2.0 \leq \text{RADIO_MOTOR} \leq 3.0$$

Con estas consideraciones, un modelo genérico de motor eléctrico [Broo-84] se declararía de la manera siguiente:

Nodo: CONO_MOTOR_ELÉCTRICO_GENÉRICO
Clase: CONO_SIMPLE
Eje: Z0014
Regla de barrido: REGLA_BARRIDO_CONSTANTE
Sección recta: Z0013

Nodo: Z0014
Clase: EJE
Tipo: RECTO
Longitud: LONGITUD_MOTOR

Nodo: REGLA_BARRIDO_CONSTANTE
Clase: REGLA_BARRIDO
Tipo: CONSTANTE

Nodo: Z0013
Clase: SECCIÓN_RECTA
Tipo: CÍRCULO
Radio: RADIO_MOTOR

El grafo de objetos tiene dos subgrafos naturales definidos por los dos tipos posibles de arcos direccionales: *subparte*(sub_part) y *fijación* (affixment). Un arco de tipo subparte relaciona dos nodos espacialmente conexos, cada uno conteniendo la definición de un cono generalizado, entre los que existe una relación direccional como *parte_principal* - *subparte*. El subgrafo constituido por relaciones del tipo subparte establece una estructura arbórea que permite describir los objetos con un nivel de detalle progresivo a medida que se desciende por el árbol. Cada objeto

(como generalizado) tiene definido su propio sistema de coordenadas local. Los arcos de tipo fijación relacionan los sistemas de coordenadas de diferentes objetos (nodos) e incluyen un producto de transformaciones simbólicas de coordenadas que permiten transformar el sistema de coordenadas del objeto apuntado por el arco al sistema de coordenadas del objeto original. Siguiendo con el ejemplo anterior, la descripción completa del motor que se muestra en la figura I.9, requeriría describir los dos elementos conexos al cuerpo del motor, en este caso, un terminal y el eje. Los nodos que contuvieran la representación de estos elementos estarían conectados por arcos de tipo subparte al nodo motor, donde se asume que la descripción más primitiva de este objeto se identifica con el elemento cilíndrico que constituye el cuerpo del motor. Arcos de tipo *fijación* permitirían transformar los sistemas de coordenadas utilizados en los objetos "eje" y "terminal" al utilizado en el cuerpo del motor. Esta organización progresiva de la descripción tiene interés al permitir que el sistema inicie la búsqueda de subpartes sólo cuando ha encontrado una instancia del objeto al que se conectan las subpartes.

I.5.1.3 CONTROL.

Aunque técnicamente muy complejo, el control de la interpretación en ACRONYM es sin embargo conceptualmente sencillo. La interpretación de una imagen se inicia a partir del primer nivel de descripción de los posibles modelos y del conjunto inicial de restricciones que acotan los posibles valores de los cuantificadores. A partir de éstos se establece un esquema iterativo (predicción - descripción - interpretación - predicción) que involucra niveles de detalle de los modelos cada vez más finos a medida que los objetos se identifican. Nótese que, si bien la interpretación utiliza resultados previos para intentar una interpretación cada vez más fina, ésta ocurre de manera dirigida esencialmente por los datos: se extraen características de la imagen y se comparan con los modelos. No existe una utilización de evidencias parciales para dirigir por conocimiento la búsqueda en la imagen de partes de los modelos no detectadas en primera lugar. Esto hace que los errores cometidos en la fase de descripción de la imagen, una de las partes más débiles de ACRONYM [Broo-84], sean tan determinantes.

El verdadero "motor" de la interpretación se sitúa en el módulo de predicción, que se conduce básicamente por encadenamiento hacia atrás (backward chaining) de unas 280 reglas de

producción. Este conjunto de reglas contiene el conocimiento que capacita al sistema para realizar razonamientos geométricos y predecir la apariencia de características de los modelos. Los dos elementos fundamentales del módulo de predicción son el desarrollo del análisis geométrico mediante el establecimiento de restricciones, y su manipulación/simplificación por el sistema de manejo de restricciones.

Uno de los aspectos más destacados de ACRONYM es la forma en la que se desarrolla el razonamiento sobre la posición de los objetos en el espacio. Como ya se ha indicado, cada objeto se describe como una jerarquía de subpartes, cada subparte descrita mediante un frame, con un árbol de *fijaciones* asociado, que describe las relaciones espaciales entre las partes. Supongamos ahora, por ejemplo, que se desea determinar si un objeto, se encuentra en el campo de visión de la cámara. Para ello se establecen restricciones que deben satisfacerse en el caso de que el objeto se encuentre efectivamente en el campo de visión de la cámara. Los valores de los parámetros, en este caso posicionales, que satisfacen el conjunto de restricciones constituyen el *conjunto de verificación (satisfying set)*. El *sistema de manipulación de restricciones* es el encargado de obtener tal conjunto. Si el conjunto es vacío, entonces el conjunto de restricciones es inconsistente, o en otras palabras, evidencia una contradicción. Sin embargo, debido a que el procedimiento de prueba no es completo, tan sólo se puede establecer su verificación negativa y no su recíproco. Como ejemplo, consideremos la oclusión de objetos en el campo de visión de una cámara. El volumen definido por el campo de visión puede representarse por una pirámide infinita de sección recta circular que se extiende a partir del punto focal de la cámara. Para establecer si los objetos se encuentran situados dentro de dicho volumen se añaden las restricciones adecuadas. Ahora para determinar oclusiones, se seleccionan pares de objetos y se incorporan nuevas restricciones que aseguren que un objeto nunca ocluye al otro. Si el conjunto de verificación es vacío, se concluye por contradicción *definitivamente* que el objeto está al menos parcialmente ocluido. Por contra, debido a la parcialidad de la prueba, si el conjunto no es vacío el objeto *puede* ser visible.

I.5.2 SPAM.

SPAM (System for Photo interpretation of Aerial Airports using Maps) es un sistema basado en reglas concebido inicialmente para interpretar imágenes de aeropuertos [McKe-85] con la utilización de conocimientos de ingeniería civil en el diseño de aeropuertos, y extendido con posterioridad a la interpretación de imágenes aéreas de suburbios [McKe-89].

I.5.2.1 ARQUITECTURA.

En su concepción inicial [McKe-85], SPAM fue diseñado de manera que el conocimiento propio del dominio de aplicación se mezclaba con el conocimiento que guiaba las fases de interpretación/control. Posteriormente, esta primera versión se rediseñó [McKe-89] para adaptar el sistema a diferentes dominios mediante la incorporación de una base de conocimiento adecuada. En esta última versión de SPAM, se incorporaron al sistema una serie de módulos, orientados a la adquisición del conocimiento (ISCAN), a la incorporación del conocimiento propio del dominio de aplicación al sistema de interpretación (RULEGEN) y un módulo de evaluación de la interpretación (SPATS). La articulación del sistema en sus diferentes partes se muestra en la figura I.10.

Nótese que el término SPAM se utiliza en dicha

figura para nombrar al módulo de interpretación de imágenes o al conjunto del sistema. Describiremos a continuación, y de forma breve, el módulo de adquisición del conocimiento, el compilador de reglas y el módulo de evaluación para centrarnos después en el módulo interpretación.

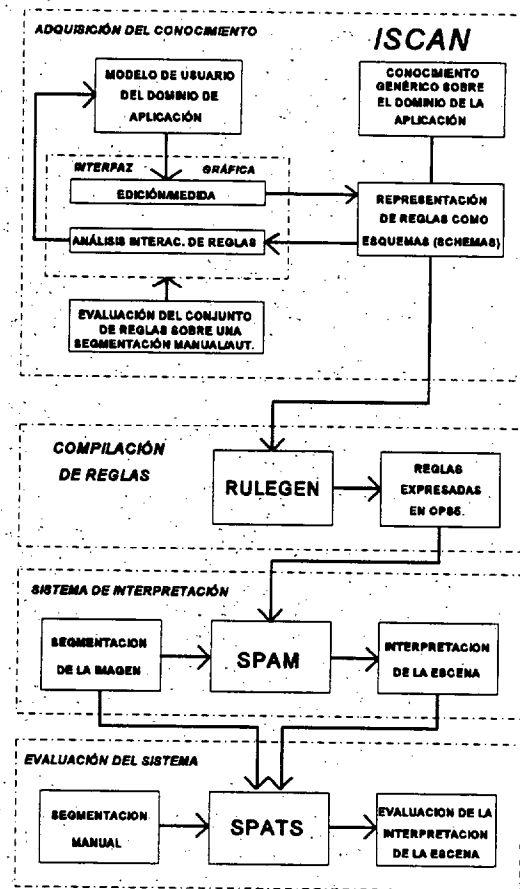


Figura I.10. Arquitectura de SPAM.

El módulo ISCAN es una herramienta que permite la adquisición de conocimiento mediante la interacción con el usuario. Contiene un editor que posibilita al usuario la adición, eliminación o modificación de los esquemas (schemas) que estructuran el conocimiento propio del dominio de la aplicación. Mediante este módulo, el usuario puede analizar interactivamente la evaluación de esquemas sobre las regiones definidas en una determinada segmentación de la imagen.

El módulo RULEGEN es un compilador que transforma el conocimiento expresado en forma de esquemas en reglas de producción expresadas en OPS5. Este compilador analiza la correcta definición de las jerarquías de clases y reglas, y estructura éstas últimas de acuerdo con el conocimiento procedimental que posee sobre la estructura de control de SPAM.

SPATS permite evaluar la bondad de la interpretación de la escena producida por el módulo de interpretación SPAM. Para ello, compara la interpretación resultante con otra, obtenida a partir de una segmentación manual de la imagen, que se toma como interpretación de referencia. Esta interpretación de referencia contiene la identificación de cada región presente en la segmentación y su asignación a un cierto modelo [McKe-84]. En el caso de utilizar un programa de segmentación, el módulo SPATS permite identificar la influencia de una segmentación imperfecta sobre la interpretación final, y en el caso de una segmentación manual, evaluar el conjunto de reglas que constituyen el conocimiento utilizado por SPAM sobre unos datos iniciales perfectos.

El compilador RULEGEN genera el módulo de interpretación (SPAM) utilizando la base de conocimiento de esquemas. En este módulo se lleva a cabo la interpretación de una escena en base a la serie de primitivas que se definen a continuación:

1. *Regiones*. Son las zonas que resultan de la segmentación de la imagen. Quedan definidas por su localización espacial y por atributos de extensión, forma, textura, propiedades espectrales, etc.

2. *Fragmentos*. Un fragmento es la interpretación de una región de acuerdo con su posible clasificación como instancia de una clase. Una región puede ser asignada a diferentes clases generando así múltiples fragmentos. Por ejemplo, en escenas de aeropuertos, clases posibles son "lineal" y "compacto", que tienen como subclases las clases "pista de aterrizaje", "pista de aproximación" y "carretera", y "edificio terminal" y "hangar" respectivamente.

3. *Áreas funcionales*. Los fragmentos se agrupan, según la compatibilidad de las hipótesis que representan, en áreas funcionales o interpretaciones parciales de la escena. La definición de áreas funcionales divide la imagen en zonas que pueden estar sobrepuestas ya que un mismo fragmento puede estar incluido en varias áreas funcionales. Así por ejemplo, una posible área funcional en escenas de aeropuertos es "edificio terminal", que se compone de fragmentos de tipo "edificio terminal", "zona de aparcamientos" y "carretera".

4. *Modelos*. Un modelo se define por una colección de áreas funcionales mutuamente compatibles. En escenas de aeropuertos, los modelos son la expresión en áreas funcionales de aeropuertos reales contenidos en una base de datos.

I.5.2.2 REPRESENTACIÓN DEL CONOCIMIENTO.

En SPAM, el conocimiento se representa por esquemas (schemas) cuya tipología refleja el conocimiento requerido en las diferentes las fases de interpretación. El compilador RULEGEN utiliza esta tipología para sintetizar el conjunto de reglas e identificar aquellas que deben utilizarse en cada fase de interpretación. La organización por tipos de las reglas utilizadas en SPAM se muestra en la figura I.11. Los tipos de esquemas posible son:

a) *Definiciones de las clases de fragmentos*. Estos esquemas declaran la forma y el rango de valores posibles para los atributos de una clase de fragmento. Por ejemplo, la definición de la subclase "casa" puede expresarse con el siguiente esquema:

Clase = "casa"
Dependencias-región = ""
Dependencias-fragmento = "objeto_tipo compacto && hipótesis desconocida"
Restricción-forma = "area && 50.00 ≤ valor ≤ 150.00"
Restricción-forma = "elipse_longitud && 12.00 ≤ valor ≤ 18.00"
Restricción-forma = "elipse_anchura && 10.00 ≤ valor ≤ 20.00"
Restricción-forma = "elipse_linealidad && 0.00 ≤ valor ≤ 3.50"

b) *Condiciones sobre compatibilidad local.* El conocimiento que se declara en estos esquemas cuantifica relaciones espaciales entre fragmentos, llamados hipótesis en la sintaxis de este tipo de esquemas, como "cerca_de", "paralelo", etc, y le asigna una cierta credibilidad o certeza. La cuantificación se realiza imponiendo un rango para los valores posibles de la medida de relación espacial. A continuación incluimos un ejemplo de este tipo de esquemas que sirve para sintetizar una regla que establece que las casas suelen alinearse paralelamente a la carretera en una escena de suburbios:

Nombre-Regla = "las casas son paralelas a las carreteras"
Certeza = "0.8"
Hipótesis = "casa && carretera"
Geometría = "orientación"
Subtipo = "paralelo"
Límites = "0.00 ≤ valor ≤ 0.50"

c) *Definiciones de áreas funcionales.* El conocimiento contenido en la definición de un área funcional enumera los tipos de fragmentos que la componen. Un área funcional requiere siempre de un tipo de fragmento (Region-semilla), que debe estar siempre presente, y de uno o más de los fragmentos que se declaran en la definición. Así, el área funcional "terminal" se define como constituida por la hipótesis seminal "edificio terminal" y las hipótesis adicionales: "zona de aparcamientos" y "carretera".

Nombre-AreaFuncional= "terminal"
Región-Semilla = "edificio terminal"
Definición = "zona de aparcamientos && carretera"

d) *Generación de modelos.* En la última fase de la interpretación, SPAM combina áreas funcionales para formar modelos de la escena completa. Durante este proceso, es

frecuente que surjan interpretaciones conflictivas que deben ser resolverse antes de obtener un modelo final consistente. El conocimiento acerca de los conflictos que se pueden presentar y su mecanismo de resolución, se declara utilizando este tipo de esquemas. Por ejemplo, supongamos que una misma región integre un área funcional como hipótesis de "hangar" y de otra bajo la hipótesis de "zona de aparcamientos". El siguiente esquema indica qué método debe utilizarse para resolver este conflicto, consistente en la invocación de un programa de análisis estéreo para decidir entre ambas hipótesis en base a la altura:

Conflicto = " hangar && zona de aparcamientos"
 Resolución = "función && estéreo"

Aquellos conflictos que no están incluidos se resuelven aplicando una estrategia de resolución por defecto que tiene en cuenta la certeza de las hipótesis en conflicto y el grado en que las respectivas áreas funcionales son compatibles con el modelo propuesto para la escena.

I.5.2.3 CONTROL.

El compilador RULEGEN utiliza el tipo de esquema que origina cada regla para estructurar el conjunto de reglas en subconjuntos dependiendo de las reglas que se pueden aplicar en las diferentes fases que comprende la interpretación de una escena en SPAM (figura I.11). SPAM inicia la interpretación de la escena a partir de una segmentación inicial y ejecuta un proceso de interpretación secuencial constituido por las siguientes fases:

a) *Definición de los fragmentos.* Se relaciona cada región con una posible interpretación en términos de las clases definidas a partir exclusivamente de propiedades

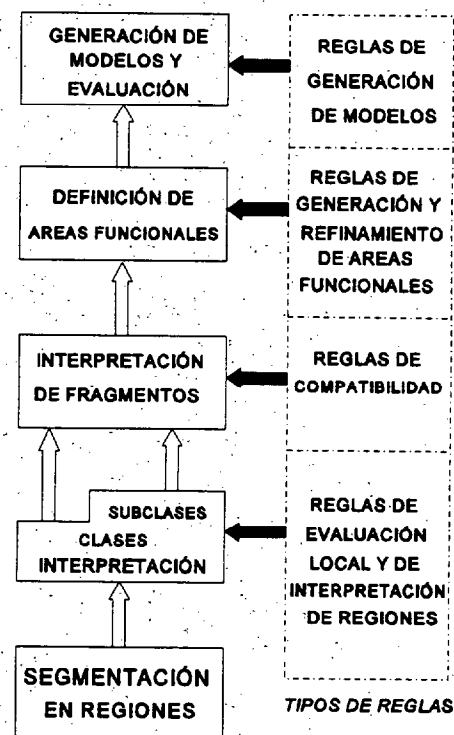


Figura I.11. Fases de la interpretación en SPAM.

locales (forma 2D, textura, información 3D, etc). Se asocia a cada fragmento una certeza acerca de la bondad de la interpretación.

b) *Análisis local de compatibilidades.* Se verifica la compatibilidad entre pares de fragmentos que sostienen una determinada relación espacial. Una verificación positiva refuerza la fiabilidad asociada a ambos fragmentos.

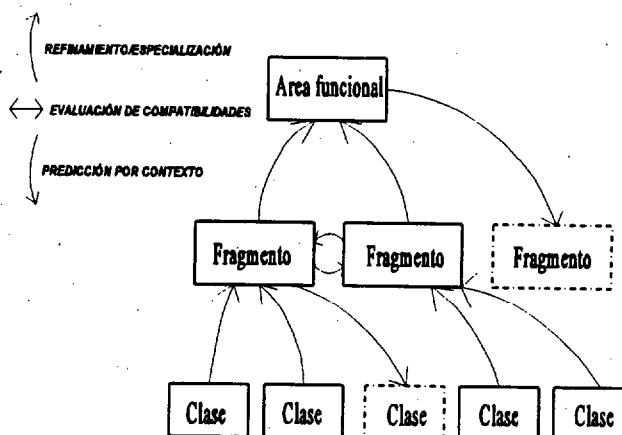


Figura I.12. Principio elemental de interpretación en SPAM.

c) *Definición de áreas funcionales.* Se agrupan los fragmentos mutuamente compatibles en áreas funcionales. La extensión espacial de un área funcional queda establecida por la envolvente convexa (convex hull) de los fragmentos incluidos. El área funcional recibe una certeza inicial basada en la certeza promedio de los fragmentos que la integran. Esta certeza se modifica si el área funcional contiene o interseca fragmentos que tienen una interpretación incompatible con ella.

d) *Generación de modelos.* Esta fase comienza cuando se han generado un número suficiente de áreas funcionales. Las áreas funcionales se agrupan para producir los modelos presentes en la escena (p.e. diferentes tipos de aeropuertos). Los conflictos que surgen al combinar áreas funcionales se resuelven teniendo en cuenta la fiabilidad asociada a las áreas funcionales en conflicto o mediante la utilización de una estrategia específica declarada por el usuario. La interpretación finaliza cuando todas las áreas funcionales han sido incluidas en algún modelo.

La interpretación en SPAM es básicamente ascendente o conducida por los datos. Sin embargo, no excluye la posibilidad de una vuelta atrás o descenso a una fase de interpretación más temprana. La figura I.12 muestra el esquema elemental que articula el proceso de interpretación en SPAM. La compatibilidad entre hipótesis permite combinarlas para crear contextos que susciten predicciones sobre hipótesis no presentes. En SPAM, estas predicciones tienen lugar

desde un nivel de interpretación superior a otro inferior. Por ejemplo, en escenas de aeropuertos, conjuntos de pistas de aterrizaje y de pistas de espera mutuamente compatibles pueden ser combinadas para crear un área funcional del tipo "pista de aterrizaje". Las reglas que definen este tipo de área funcional podrían indicar que suelen aparecer también zonas cubiertas de césped. En ese caso, se emitiría la predicción de que ciertas regiones incluidas en el área funcional podrían ser un buen lugar para buscar fragmentos del tipo "zonas de césped".

I.5.3 SIGMA.

SIGMA [Mats-90] es un sistema de interpretación de imágenes diseñado para interpretar imágenes aéreas de suburbios urbanos. La concepción de SIGMA está fundamentada en la distinción de dos niveles conceptuales para los IUS: *los dominios de la imagen y de la escena*. Esta diferenciación se realiza en base al tipo de conocimiento que se requiere en cada uno de estos niveles y determina la arquitectura del sistema. En SIGMA los análisis ascendente (bottom-up) y descendente (top-down) se encuentran integrados en un esquema de razonamiento por acumulación de evidencias que controla el ciclo de interpretación. A diferencia de otros IUS, SIGMA posee un módulo elaborado para realizar la segmentación primaria de una imagen utilizando conocimiento de técnicas de proceso de imágenes.

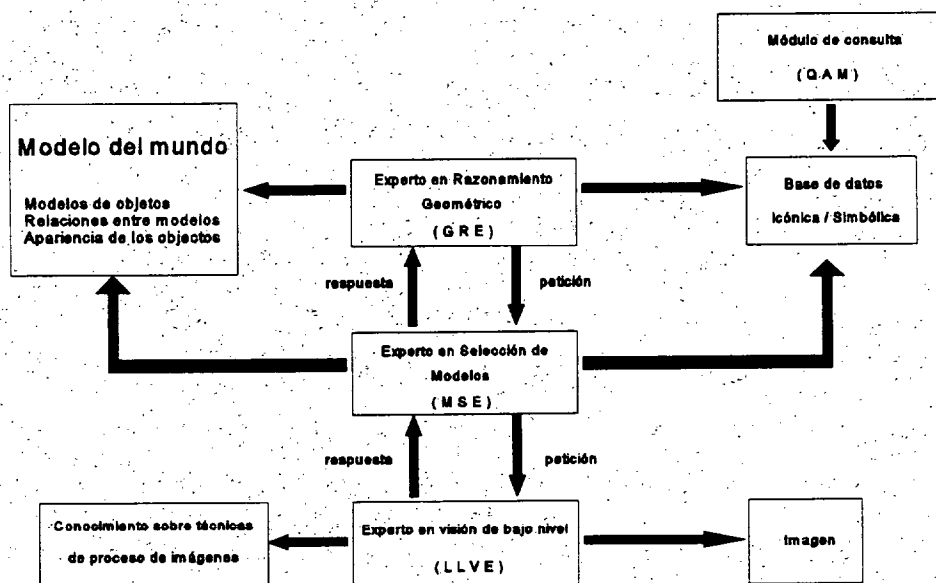


Figura I.13. Arquitectura de SIGMA [Mats-90].

I.5.3.1 ARQUITECTURA.

Este sistema está estructurado en base a cuatro *módulos expertos*, tal y como se muestra en la figura I.13: el módulo de razonamiento geométrico (GRE), el módulo de selección de modelos (MSE), el módulo de visión de bajo nivel (LLVE) y el módulo de consulta (QAM). Los tres primeros son módulos de análisis dedicados a construir la interpretación de la escena mientras que el último permite la consulta interactiva de los resultados de la interpretación. A continuación esbozaremos las características más destacadas de los tres módulos de análisis.

a) *Módulo de razonamiento geométrico (GRE).*

Constituye el módulo más importante del sistema y tiene como misión construir una descripción de la escena estableciendo relaciones espaciales entre objetos. Utiliza como fuente de conocimiento un modelo jerárquico del mundo que contiene la definición de los objetos como estructuras geométricas y las posibles relaciones espaciales entre las mismas. La definición de los modelos se realiza en un marco de representación orientado a objetos, donde la definición de una clase de objeto contiene posibles relaciones con otros objetos. Así, cuando en el proceso de interpretación un objeto es reconocido como perteneciente a una clase se crea una instancia de ésta que es capaz de generar hipótesis sobre otros posibles objetos relacionados. Por ejemplo, cuando una instancia de la clase "casa" es identificada genera hipótesis sobre posibles "casas" vecinas y sobre el "camino" que las conecta. De esta forma, cada instancia es capaz de suscitar hipótesis acerca de su entorno, siendo la principal tarea de GRE la coordinación de estas inferencias locales para construir una interpretación de la escena globalmente consistente. Las instancias de objetos y las hipótesis generadas por ellos constituyen lo que en SIGMA se denominan *evidencias*. Todas las evidencias parciales que se obtienen durante la interpretación se almacenan en una estructura denominada *Symbolic/Iconic database*. Cada evidencia se describe de dos maneras diferentes: una icónica que representa su localización espacial y otra simbólica que describe sus atributos y relaciones con otras evidencias. GRE agrupa evidencias que son mutuamente consistente en lo que se denominan *situaciones*. Éstas definen por su extensión espacial un entorno local o contexto que se emplea como foco de atención durante el análisis si GRE selecciona esa situación.

b) Módulo de selección de modelos (MSE).

Este módulo se activa por GRE para razonar acerca del aspecto o presentación más probable de un objeto que se desea detectar en la imagen. Después de determinar el aspecto, MSE activa al LLVE para extraer una o más características (features) de la imagen de acuerdo con la presentación seleccionada. Si estas tareas se completan con éxito, MSE genera una instancia del objeto en cuestión que es devuelto a GRE como respuesta a su petición. La determinación de la presentación de un objeto se lleva a cabo en MSE en tres pasos.

1. *Selección de un modelo especializado.* En SIGMA, los modelos de los objetos se organizan en diferentes niveles de especialización como se muestra en la figura I.14. Por ejemplo, la clase genérica "casa" puede descomponerse mediante relaciones del tipo UN-TIPO-DE (A-KIND-OF) en las clases de objetos "casa-rectangular" y "casa-en-forma-de-L". Así, cuando GRE comanda la búsqueda de un instancia de "casa", MSE la traduce a la búsqueda de formas rectangulares y en forma de L.

2. *Descomposición de modelos compuestos.* El objeto reclamado por GRE puede ser un objeto compuesto de múltiples partes, en cuyo caso MSE descompone el objeto en sus partes constituyentes. Por ejemplo, cuando la clase a buscar es "coche", MSE descompone esta clase en "ruedas", "carrocería", etc... y ordena a LLVE la búsqueda de características propias de estas clases. En SIGMA los objetos compuestos se representan por clases de objetos que están enlazados mediante relaciones del tipo PARTE-DE (PART-OF) con las clases que constituyen sus diferentes partes.

3. *Determinación de la presentación 2D.* Una vez que un cierto modelo de objeto con una forma determinada ha sido seleccionado, MSE determina su posible presentación bidimensional a partir de un modelo de cámara conocido a priori.

Tras el proceso descrito, MSE genera una petición a LLVE que contiene las características que deben extraerse de la imagen en la zona definida como foco de atención y sus propiedades (determinadas como un conjunto de restricciones sobre los posibles valores de las características). A continuación se muestra un ejemplo de petición a LLVE consistente en

Capítulo I

encontrar en la ventana (40,75,240,320) un rectángulo de área comprendida entre 100 y 200 pixels.

```
((GOAL (AND (EQUAL IMAGE-FEATURE-TYPE RECTANGLE)
              (AND (LESSP AREA 200)
                    (GREATERP AREA 100))))))
```

```
(LOCATION (AND (EQUAL START-I 40)
               (EQUAL START-J 75)
               (EQUAL END-I 240)
               (EQUAL END-J 320))))
```

c) *Módulo de visión de bajo nivel (LLVE).*

Está encargado de realizar la segmentación de la imagen para extraer las características demandadas por MSE, esto es, líneas y regiones que cumplan una serie de condiciones. El módulo genera un plan de segmentación automáticamente encadenando diferentes procesos y operadores y seleccionando los parámetros óptimos. Dado, por ejemplo, un objetivo como el indicado en el ejemplo anterior, LLVE primero extrae el área indicada por la ventana y confecciona un plan en la forma:

Detección de bordes → Encadenamiento de bordes → Detección de contornos cerrados → Aproximación poligonal → Rectángulo. A continuación, selecciona un operador y sus parámetros para cada etapa del plan: detector de bordes de Sobel para detección de bordes, 8-connectividad para el encadenamiento de bordes y así sucesivamente.

La selección de los operadores se basa en una cierta medida de la calidad de la imagen en la ventana considerada, que también es utilizada en el ajuste de parámetros mediante ensayos de tipo prueba-y-error. El esquema de prueba-y-error es asimismo empleado en la selección de operadores alternativos o en la confección de un nuevo plan cuando uno de los operadores de un plan o el plan en su conjunto fallan.

I.5.3.2 REPRESENTACIÓN DEL CONOCIMIENTO.

El conocimiento tal y como se emplea en SIGMA se puede clasificar en tres modalidades atendiendo a su funcionalidad: conocimiento del dominio de la imagen, conocimiento del dominio de la escena y conocimiento acerca de cómo se establece la correspondencia entre la escena y el dominio de la imagen. El conocimiento necesario en el dominio de la imagen está relacionada con el tipo de características a extraer de una imagen, esto es, mediante qué procedimientos y cómo es posible agruparlas para construir una descripción estructural de la misma. En el dominio de la escena, el conocimiento que se requiere incluye las propiedades intrínsecas de los objetos en el mundo y las relaciones entre éstos. Se describe mediante una terminología propia de este nivel: nombres de los objetos y sus partes constituyentes, las relaciones geométricas entre las partes, sus dimensiones, etc. El conocimiento acerca de cómo se establece la correspondencia entre el dominio de la escena y el dominio de la imagen implica conocer cuál es el aspecto, esto es cómo se presentan los objetos del mundo en una imagen a partir del conocimiento que de estos objetos se tiene en el dominio de la escena. Estos tres tipos de conocimiento son los empleados por SIGMA en los tres módulos de análisis. GRE utiliza conocimiento propio del dominio de la escena, MSE lo tiene acerca la correspondencia de los objetos de una escena en una imagen y el conocimiento que emplea LLVE es propio exclusivamente del dominio de la imagen.

El conocimiento que emplean GRE y MSE es específico al problema de visión que se intenta resolver, en este caso la interpretación de imágenes de suburbios urbanos. Como se aprecia

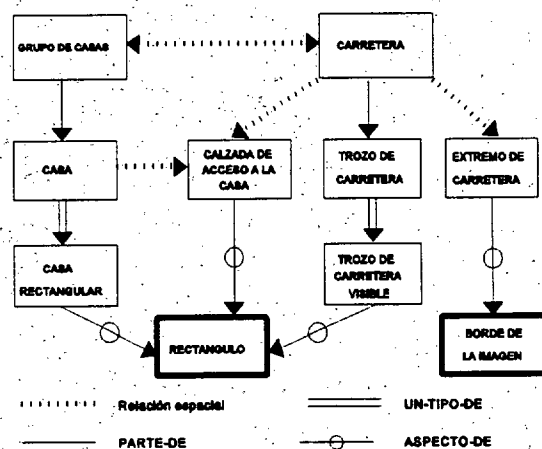


Figura I.14. Modelo del mundo utilizado por SIGMA

en la figura I.13, estos módulos necesitan de un modelo del mundo que incluya la definición de los objetos, las relaciones entre los objetos y cómo se pueden presentar los objetos en una imagen. La figura I.14 muestra el modelo del mundo utilizado por SIGMA, donde las clases de objetos en el dominio de la escena están representadas por rectángulos de borde continuo, y por borde discontinuo las clases propias del dominio de la imagen. Las clases de objetos en el dominio de la escena son estructuras que se definen en base a tres tipos de información: atributos (slots) de los objetos (centroide, descriptores de forma,...); reglas (rules) que contienen información de control y sobre relaciones espaciales entre las clases de objetos; y enlaces (links) que permiten relacionar jerárquicamente clases de objetos en base a relaciones del tipo "UN-TIPO-DE", "ASPECTO-DE" Y "PARTE-DE".

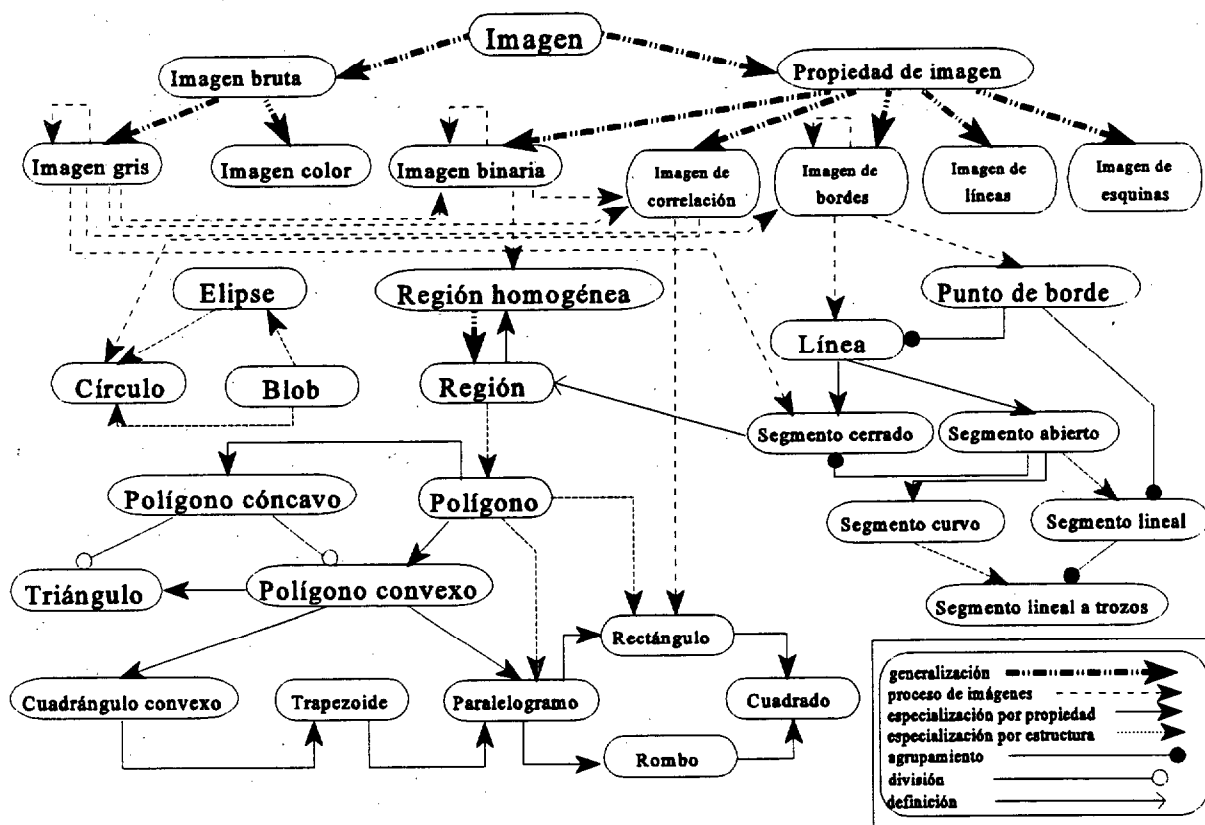


Figura I.16. Organización del conocimiento en LLVE [Mats-90].

Por otra parte, el conocimiento introducido en LLVE es independiente del dominio de la aplicación y se utiliza para llevar a cabo automáticamente la segmentación de la imagen. Este conocimiento es de dos tipos:

1. *Conocimiento acerca de los conceptos fundamentales en segmentación:* tipos de características extraíbles de una imagen y tipos de operadores posibles para proceso de imágenes.
2. *Conocimiento acerca de las técnicas de segmentación de imágenes:* cómo combinar adecuadamente los operadores.

El primer tipo de conocimiento se representa en SIGMA mediante la red conceptual que se muestra en la figura I.15, mientras que el segundo, que es de tipo eminentemente heurístico, se condensa en un conjunto de reglas de producción.

I.5.3.3 CONTROL.

El ciclo de operación en SIGMA comprende tres fases bien diferenciadas tal y como se muestra en la figura I.16. En una primera fase, SIGMA activa un proceso de segmentación inicial diseñado para extraer características que se correspondan con presentaciones de los objetos fácilmente detectables. En base a estas características, se genera un conjunto semilla de instancias de objetos que se insertan en la *base de datos de iconos y símbolos*. En una segunda fase, y a partir de las hipótesis que este primer conjunto de instancias produce, se construye de forma gradual la interpretación de la escena. Para ello en SIGMA se itera el siguiente ciclo interpretativo bajo el control de GRE:

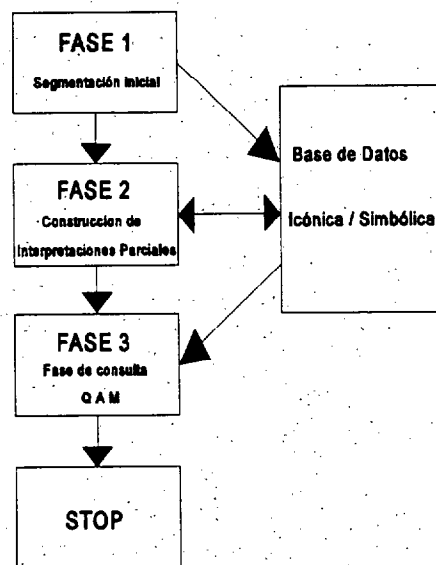


Figura I.16. Ciclo de interpretación de SIGMA.

1. *Generación de hipótesis.* Cada instancia de objeto presente en la base de datos genera hipótesis sobre objetos con los que guarda relación espacial utilizando para ello las reglas contenidas en la definición de la clase.
2. *Acumulación de evidencias.* Se examina la consistencia de las diferentes piezas de evidencia (hipótesis e instancias) contenidas en la base de datos. Aquellas que son consideradas mutuamente consistentes se constituyen en una "situación".
3. *Foco de atención.* El mecanismo de focalización se basa en calcular la fiabilidad asociada a cada situación y seleccionar la situación la de mayor fiabilidad. En la implementación descrita en [Mats-90], la fiabilidad se asigna de manera uniforme para todas las instancias de objetos, fiabilidad igual a 5, y para todas las hipótesis, fiabilidad igual a 1.
4. *Resolución de la situación seleccionada.* A partir de las hipótesis contenidas en la situación se construye "una hipótesis compuesta" que intenta verificar la existencia en la situación de una instancia de un determinado objeto. Si la situación contiene dicha instancia, la *base de datos de iconos y símbolos* se modifica para reflejar la verificación de esta hipótesis compuesta. En caso contrario, se activa un análisis descendente, que involucra a los módulos MSE y LLVE, dirigido a la detección de objetos que no se detectaron en la segmentación inicial de la imagen. En este caso, la segmentación se restringe a la ventana definida como foco de atención.

Las nuevas instancias de objetos suscitadas en este análisis se utilizan para actualizar la información contenida en la *base de datos de iconos y símbolos* y para generar nuevas hipótesis, repitiéndose el ciclo a partir de este punto. Cuando GRE encuentra instancias que son inconsistentes con la interpretación que las hipotetizó, la interpretación se divide en dos, conteniendo cada una instancias que sean consistentes. Mediante este proceso SIGMA se pueden mantener múltiples interpretaciones de la escena. GRE itera el ciclo de interpretación hasta que no se generan nuevas hipótesis y las que se generaron han sido verificadas o refutadas, seleccionando como interpretación final aquella que contiene un mayor número de instancias de objetos. En la última fase, el control se transfiere al módulo de consulta QAM mediante el cual es posible obtener información acerca de la interpretación resultante.

I.5.4 SCHEMA SYSTEM.

El "Schema System" [Drap-89] es un sistema basado en conocimiento para la interpretación de escenas, resultado de la evolución durante los últimos 10 años del sistema VISIONS [Rise-84][Hans-88] que integra actualmente los niveles bajo e intermedio del sistema. El Schema System presenta la particularidad de haber promovido el desarrollo de una arquitectura hardware específica al mismo, denominada "The Image Understanding Architecture" (IUA) [Weem-89][Weem-92]. Este sistema ha sido aplicado en diferentes tareas, entre ellas, las de interpretación de escenas de exteriores de viviendas unifamiliares y de carreteras a partir de imágenes en color estáticas.

I.5.4.1 ARQUITECTURA.

La arquitectura del Schema System (VISIONS) está orientada a conseguir un sistema donde la interpretación de escenas se realice de forma eminentemente distribuida en un sistema de cómputo paralelo organizado en tres niveles (IUA).

1. *Nivel bajo.* Los procesos de bajo nivel producen *eventos simbólicos*, regiones y líneas, a partir de los datos asociados a los pixels.
2. *Nivel intermedio.* Los procesos de nivel intermedio son los encargados de generar la interpretación simbólica intermedia (ISR) mediante la asociación de tipos simbólicos (tokens) a las regiones, líneas o estructuras de eventos en base a los atributos que definen los tokens. La reorganización de la ISR se produce tanto conducida por los datos (bottom-up) como por estrategias que se ejecutan desde el nivel alto (top-down) al activarse una hipótesis de objeto.
3. *Nivel alto.* La base de conocimiento, también llamada "Memoria permanente" (Long Term Memory, LTM) consiste en una red semántica de esquemas que contienen una componente declarativa y otra procedimental. La red semántica está organizada jerárquicamente en base a relaciones PARTE-DE (PART-OF) y ES-UN (IS-A). La identificación de una clase de objeto en la imagen produce la creación de una instancia del esquema asociado. Estas instancias actúan como sistemas expertos autónomos, dedicado cada uno a la detección de una clase de objeto, que cooperan en la tarea de interpretación

mediante el intercambio de mensajes a través de una pizarra (blackboard). Las instancias crean procesos de alto nivel que focalizan la atención en agregaciones de eventos por medio de hipótesis generadas por reglas. El conocimiento procedimental contenido en las porciones de la red que se activan dirigen la reorganización de la representación intermedia, refinando las hipótesis activadas y, cuando se confirman, generando otras nuevas mediante la activación de esquemas relacionados. El proceso de interpretación construye una red en la "Memoria volátil" (Short-Term Memory, STM) que se compone de instancias en la imagen de porciones de la red contenida en la LTM y que constituye, en último término, la interpretación de la escena. Dado que la operación a este nivel está absolutamente determinada por la representación del conocimiento y la estrategia de control utilizados, pospondremos a estos apartados una descripción más pormenorizada de este nivel.

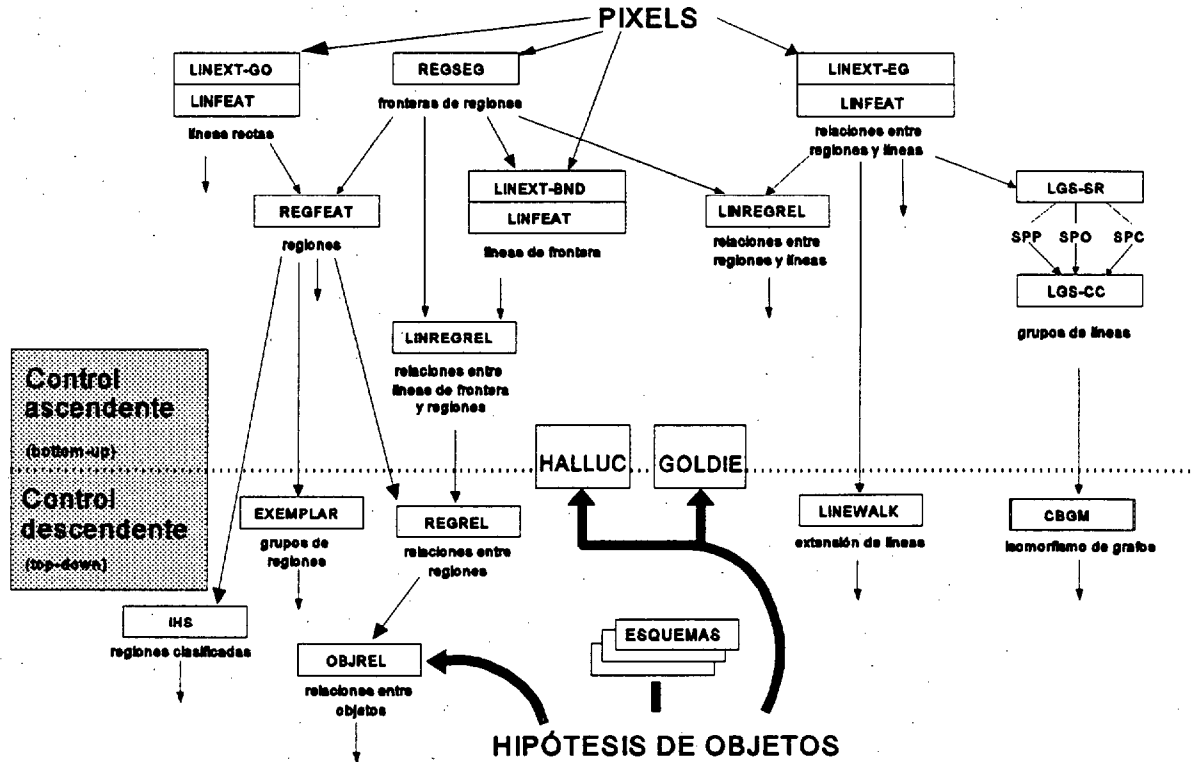


Figura I.17. Flujo de datos: desde los píxeles a las hipótesis de objetos [Drap-89]. Las flechas que no apuntan a una fuente de conocimiento son examinadas directamente por los esquemas

Los niveles bajo e intermedio del Schema System están basados en los procedimientos que utilizaba VISIONS para llevar a cabo la extracción de tokens y construcción de la representación simbólica inicial de la imagen. Éstos se organizan como "fuentes de conocimiento" parametrizadas, de manera que pueden ser ajustadas por las estrategias de control de los esquemas. Las fuentes de conocimiento descritas en [Drap-89] se agrupan según el nivel de representación en el que operan. Así se distinguen fuentes de conocimiento de nivel bajo e intermedio (figura I.17).

Las primeras realizan operaciones sobre los datos asociados a los pixels que, en primera aproximación, son conducidos por los datos o bien utilizan parámetros por defecto. Una vez que los esquemas han reunido información sobre el contenido de la imagen, los procedimientos asociados a las fuentes de conocimiento pueden ejecutarse nuevamente y de forma selectiva sobre zonas de la imagen con parámetros más discriminantes. Las fuentes de conocimiento de nivel bajo extraen tokens de regiones y líneas, y calculan características de estos tokens. Las empleadas en este nivel son:

- Segmentación en regiones mediante histogramas localizados (REGSEG).
- Extracción de características asociadas a regiones (REGFEAT).
- Extracción de líneas rectas por orientación del gradiente (LINEXT-GO).
- Extracción de líneas rectas por agrupamiento de bordes (LINEXT-EG).
- Extracción de líneas rectas por agrupamiento de bordes que sean frontera entre regiones (LINEXT-BND).

Las fuentes de conocimiento de nivel intermedio operan conducidas por los modelos a través de las estrategias de control asociadas a los esquemas activados. Actúan sobre los tokens producidos por las fuentes de conocimiento de bajo nivel para producir tokens más abstractos por agrupación. Las fuentes de conocimiento que nos encontramos en este nivel se agrupan en cinco categorías atendiendo a su funcionalidad:

1. *Clasificación basada en características.*

- Sistema para la generación de hipótesis iniciales basadas en características (IHS).
- Obtención de regiones similares en características (EXEMPLAR).

2. *Agrupamiento y organización perceptuales*

- Sistema para agrupamiento de líneas en base a relaciones espaciales binarias (LGS-SR) como aproximadamente paralelas (SPP), colineales (SPC) o perpendiculares (SPO).
- Sistema para agrupamiento de líneas en componentes conexas (LGS-CC).
- Extensión de líneas (LINEXT-EG y LINEWALK).

3. *Comparación de modelos geométricos.*

- Comparador de grafos basado en restricciones (CBGM).

4. *Relaciones entre tokens.*

- Relaciones entre regiones y líneas (LINREGREL), entre regiones (REGREL) y entre objetos (OBJREL).

5. *Resegmentación dirigida por conocimiento.*

- Sistema de nivel intermedio guiado por objetivos (GOLDIE) [Kohl-87] [Kohl-88]. Realiza la resegmentación de partes de la imagen invocando fuentes de conocimiento de nivel bajo. Es invocado por las estrategias de control de los esquemas.
- Sistema para la creación de tokens guiado por conocimiento (HALLUC).

I.5.4.2 REPRESENTACIÓN DEL CONOCIMIENTO.

El Schema System representa el conocimiento propio de un dominio como una estructura jerárquica de esquemas organizada en forma de red semántica. Cada nodo de la red o esquema se relaciona con otros bien por medio de relaciones del tipo PARTE-DE (PART-OF) y ES-UN (IS-A), figura I.18, o porque existe entre ellos una dependencia contextual o de invocación, figura I.19. Por ejemplo, la consideración de una región como poste telefónico activa el esquema que representa a los hilos del tendido telefónico. Un esquema modeliza una posible apariencia de un objeto: la apariencia de un coche circulando de noche es completamente diferente a la apariencia

de ese mismo coche cuando circula de día. La información contenida en un esquema se expresa en términos de tokens extraíbles de la imagen y sus atributos, y en particular, expresa las combinaciones de tokens y atributos que pueden ser evidencia de la presencia de un objeto en la imagen. Todo esquema se compone de tres elementos: el espacio de confirmación (endorsement space), una función de certeza y unas estrategias de control. El espacio de confirmación cualifica las evidencias devueltas por las fuentes de conocimiento u obtenidas de la pizarra como evidencia-positiva, evidencia-negativa o como ausencia de información. Por ejemplo, en un dominio de interpretación de escenas de carreteras, el esquema correspondiente a "arcén de carretera" declara seis posibles confirmaciones para atributos de la región: *color-correcto*, *color-neutro*, *color-incorrecto*, *textura-correcta*, *textura-neutra* y *textura-incorrecta*, e impone que la región debe ser próxima a otra que haya sido confirmada como "carretera" con una certeza "creíble". La confirmación de esta última condición se obtiene consultando la sección de la pizarra global correspondiente al esquema "carretera" y comprobando la existencia de una región vecina de estas características. Por otro lado, la fuente de conocimiento encargada de generar las hipótesis iniciales (IHS) compara las medidas de color y textura con las medidas de referencia para el objeto "arcén de carretera", y devuelve para cada una un valor entre -10 (error absoluto) y 10 (ajuste perfecto). Este valor se cualifica entonces como "correcto", "neutro" o "incorrecto" en base a una cierta división del rango [-10,10]. La función de certeza realiza la combinación de las confirmaciones declaradas en el espacio de confirmación para calificar la hipótesis como "fuertemente confirmada", "confirmada", "creíble", "poco creíble" o "sin evidencia". Siguiendo

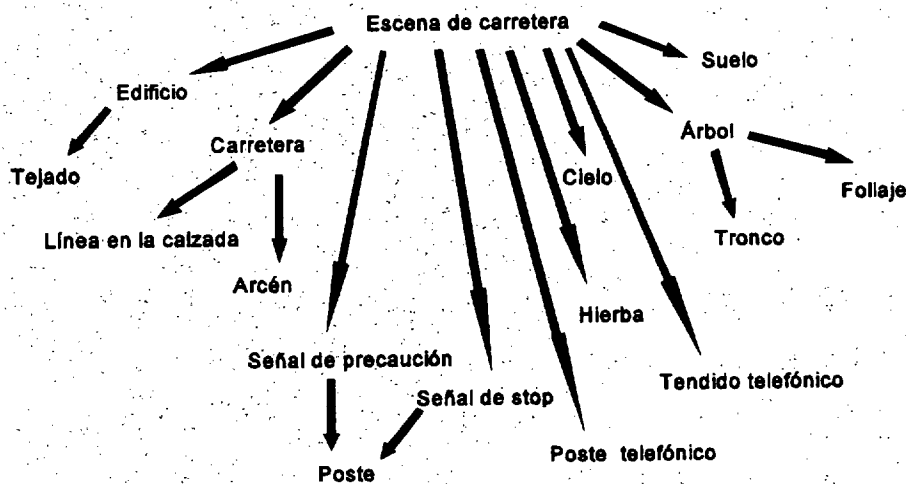


Figura I.18. Red de relaciones PARTE-DE para escenarios de carreteras [Drap-89].

con el ejemplo anterior, una región próxima a otra calificada como carretera, puede ser "fuertemente confirmada" como arcén si sus medidas de color y textura han sido calificadas como "correctas", o sólo ser "creíble" si su textura es "correcta" pero su color es "neutro". Por último, las estrategias de control sirven para articular la búsqueda de las confirmaciones; esto es, qué fuentes de conocimiento deben ser invocadas, en qué orden, qué hacer cuando las fuentes de conocimiento devuelven valores contradictorios, etc... Por ejemplo, el esquema "CASA" puede tener una estrategia para reconocer casas a una cierta distancia y otra diferente para cuando la casa esté próxima.

La base de datos de la Representación Simbólica Intermedia (ISR) es un sistema que gestiona el almacenamiento, recuperación y compartición de tokens obtenidos y/o utilizados por las fuentes de conocimiento. ISR organiza los tokens como frames y tiene incorporada la posibilidad de indexar los tokens por tipo, valor y localización espacial.

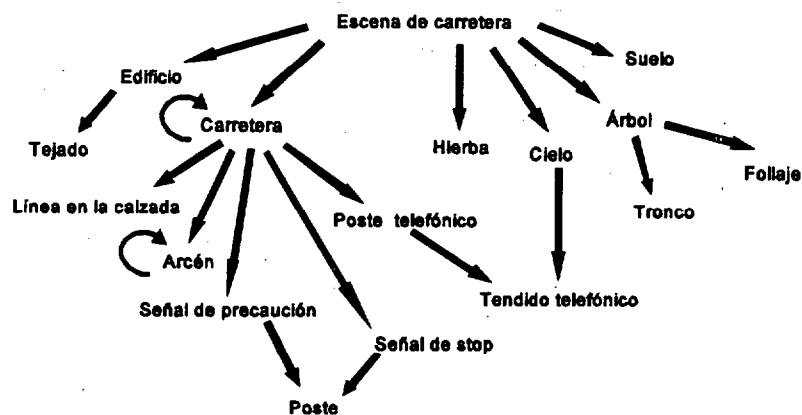


Figura I.19. Red de invocaciones para escenarios de carreteras [Drap-89].

I.5.4.3 CONTROL.

La organización del control en este sistema está condicionada por el intento de explotar el paralelismo de grano grueso que puede existir en la fase de análisis semántico o de alto nivel en el análisis de escenas. Este paralelismo se organiza alrededor de las instancias de esquemas en dos niveles diferenciados. Éstas se ejecutan de manera concurrente permitiéndose la comunicación asíncrona entre ellas por medio de una pizarra global. Además, en un nivel interno a cada instancia

de esquema, se pueden desarrollar múltiples estrategias simultáneamente, lo que establece un segundo nivel de paralelismo. En este segundo nivel, la vía de comunicación que permite la cooperación entre las estrategias de un esquema es también una pizarra local.

La estrategia de interpretación de una escena es conceptualmente simple. El sistema posee inicialmente una serie de expectativas acerca de los objetos que espera encontrar en la imagen. Estas "expectativas" se traducen en un conjunto semilla de instancias de esquemas que se encuentran activados al principio de la interpretación. Estos esquemas suelen ser muy generales, definiendo simplemente el contexto de la escena, v.g.: "escena de carretera" o "escena de casas". A medida que estas instancias predicen la existencia de otros objetos, invocan los esquemas asociados a estos objetos, los cuales invocarán a su vez a otros esquemas. El conjunto de instancias que obtienen la necesaria confirmación de sus hipótesis constituye la interpretación de la imagen. La interpretación finaliza cuando se ha asignado a cada región de la imagen la etiqueta de un objeto o ha sido marcada como desconocida si no ha sido posible asignarle una interpretación.

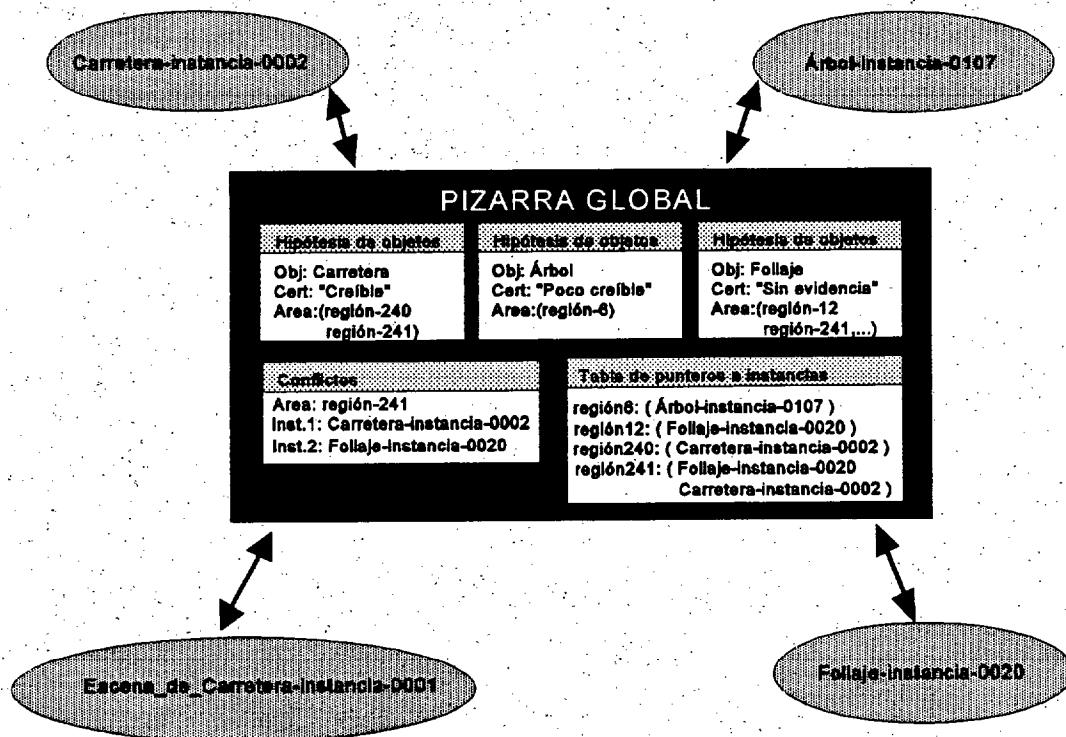


Figura I.20. La pizarra global del sistema [Drap-89].

Las instancias de esquemas actúan como subsistemas expertos dedicados a la localización de una clase de objetos. Durante la fase de interpretación, los esquemas interactúan entre ellos mediante la publicación y lectura de mensajes en una pizarra común a todas las instancias, llamada "pizarra global". Los mensajes permiten a un esquema publicar sus contribuciones y acceder a las contribuciones de otros esquemas. La publicación de mensajes tiene lugar por secciones, correspondiendo cada sección de la pizarra a una clase de objeto. Las contribuciones se realizan por medio mensajes del tipo *hipótesis de objetos*, los cuales contienen:

- la clase de objeto.
- la identidad de la instancia que publica la hipótesis.
- relaciones como parte/subparte con otras hipótesis de objetos.
- una lista de las porciones de la imagen explicadas o relacionadas con esta hipótesis.
- un valor de certeza para la hipótesis

Otro tipo de mensaje importante es el de detección de conflictos, que se produce cuando dos hipótesis incompatibles se superponen espacialmente. Cuando una instancia publica una hipótesis, comprueba al mismo tiempo la posible existencia de otra hipótesis que esté en conflicto con la primera. Si se encuentra, se envía un mensaje de conflicto a la instancia de esquema que publicó la hipótesis conflictiva. El conflicto se resuelve entonces por competencia entre los dos instancias. Cuando el conflicto es predecible, por ejemplo porque es fácil identificar el tronco de un árbol como un poste de teléfono, pueden incluirse estrategias de resolución de este tipo de conflictos en la propia definición de los esquemas. En otro caso, la certeza asociada a las hipótesis se utilizan para resolver el conflicto.

Cada instancia de un esquema, durante la interpretación de la escena, desarrolla y mantiene hipótesis internas acerca de posibles instancias de su clase de objetos en la escena. Estas hipótesis son creadas, utilizadas y modificadas por las estrategias de control activas en la instancia de esquema. El mecanismo que permite el uso concurrente de estas hipótesis es una "pizarra local". El empleo de una pizarra local establece un filtro que evita trasladar a la pizarra global hipótesis débiles o pendientes de confirmación, aumentando efectivamente la distribución del control en el

sistema. La pizarra local también está dividida por secciones, correspondiendo cada sección a un nivel diferente de representación de los tokens incluidos. Por ejemplo, la hipótesis de "tronco de árbol", puede contener inicialmente líneas rectas, a continuación, generar tokens que representen agrupamientos de líneas rectas que verifiquen el ser paralelas y, finalmente, regiones que estén delimitadas por los grupos de líneas paralelas. Una estrategia de control especial, común a todas las instancias de esquemas, es la estrategia de *Mantenimiento de Hipótesis de Objetos* (OHM). Esta estrategia monitoriza la actividad de otras estrategias activas en la instancia de un esquema y actualiza los mensajes de hipótesis de objetos dirigidos a la pizarra global. Asimismo, es la encargada de detectar y resolver conflictos entre hipótesis internas.

I.5.5 DISCUSIÓN Y CONCLUSIONES SOBRE LÍNEAS GENERALES DE DISEÑO.

El esfuerzo invertido en el desarrollo de sistemas de interpretación de imágenes no ha conseguido determinar qué diseño es el mejor para lograr un IUS óptimo, esto es, robusto, fiable, rápidamente adaptable a diferentes tareas, eficiente, etc. Sí, en cambio, parece haber producido un cierto consenso acerca de cuáles son los problemas fundamentales que el diseño debe considerar, y cuáles deberían ser, en definitiva, las capacidades que un IUS debería poseer para poder enfrentar aquellos problemas [Rise-84][Rise-87][Mats-90][Hans-88][Drap-89][Jain-91]. A continuación, y sin ánimo de exhaustividad, expondremos las pautas de diseño que se desprenden de este análisis y consideramos fundamentales en un IUS.

a) Estructuración por niveles.

La estructuración de los sistemas de interpretación de imágenes por niveles, atendiendo a la naturaleza del grano de información empleada en cada nivel, es un principio de organización presente en la práctica totalidad de estos sistemas. La organización por niveles más común es la que define tres niveles: nivel bajo que se asocia a procesos donde el grano de información es el pixel; el nivel intermedio que utiliza regiones, líneas o volúmenes; y el nivel alto que maneja las definiciones de los objetos o modelos.

b) Vías de control guiadas por los datos (bottom-up) y por los objetivos (top-down).

El contexto o conocimiento debe guiar los procesos activos en el sistema en cada uno de los niveles, limitando efectivamente el espacio de búsqueda en el que se desarrolla la interpretación y aumentando la eficiencia computacional y la fiabilidad. Aquellos esquemas exclusivamente conducidos por los datos, frecuentes en los primeros IUSs, han demostrado ser insuficientes. No obstante, parece concluyente que los análisis conducidos por los datos y los conducidos por conocimiento deben integrarse en el esquema de interpretación de manera uniforme. Es muy interesante, por ejemplo, que la activación de nuevas hipótesis en el transcurso de la interpretación pueda dirigir la resegmentación de determinadas zonas de la imagen utilizando nuevos operadores y parámetros.

c) Representación del conocimiento.

Una elección adecuada, para representar el conocimiento que el IUS debe utilizar, determina la facilidad con la que se puede construir el sistema y su eficiencia cuando opera. Los diferentes modelos de representación utilizados (frames, reglas, fuentes de conocimiento en sistemas "blackboard",...) son epistemológicamente equivalentes, y las diferencias se encuentran en la eficiencia, explicitud del conocimiento y en la facilidad de implementación/extensión del sistema. El conocimiento debe contener la descripción de los modelos presentes en el mundo y cómo se presentan los modelos en una imagen. También se incluirá el conocimiento propio del dominio de la imagen, esto es, acerca de técnicas de proceso de imágenes y segmentación. Los atributos de los objetos deben describirse en los términos inteligibles a cada nivel del IUS, es decir, propiedades de los pixels, propiedades de agrupamientos de pixels o segmentos, propiedades de objetos, etc.

La elaboración de los modelos del mundo, la adquisición del conocimiento propia del dominio de aplicación, supone un obstáculo en el desarrollo de sistemas de interpretación de imágenes [Conn-87][Niem-90b][Fich-92][Pelle-94]. Así lo demuestra el hecho de que los sistemas más conocidos tan sólo se hayan aplicado en un dominio o dos a lo sumo. Para progresar en el desarrollo de estos sistemas es necesario avanzar en la automatización de los procesos de adquisición de modelos (v.g. aprendizaje).

d) La interpretación como un proceso dinámico de activación de hipótesis.

Los errores de segmentación e interpretación tienen una importancia decisiva sobre la eficiencia y fiabilidad de todo IUS. La consideración simultánea de múltiples hipótesis sobre los objetos presentes en la escena cuando muchas de estas hipótesis pueden ser erróneas, supone una sobrecarga inútil del sistema y limita la utilización efectiva del conocimiento en la interpretación. Esto exige la utilización de un esquema de control de la interpretación que progrese de "lo más evidente" a "lo más confuso". En otras palabras, la construcción de la interpretación debe basarse en un mecanismo de refinamiento progresivo que parta de un conjunto inicial de hipótesis muy fiables ("islas de fiabilidad" [Hans-88]), que se refina por la activación en cadena de nuevas hipótesis sugeridas por otras que ya han sido confirmadas. La asunción de este esquema supone en primera instancia un factor de economía computacional pues ordena el esfuerzo según la plausibilidad de las hipótesis. Además, la combinación de este esquema de control con la concepción de las hipótesis de objetos como "agentes activos" introduce una forma elegante de organizar el control de manera distribuida.

e) La fusión de información y el manejo de la incertidumbre.

Un problema central en los sistemas de percepción artificial es la transformación de las medidas numéricas o características definidas en el dominio de la señal, a los símbolos o diagnósticos propios del dominio simbólico. Este problema tiene dos aspectos importantes. El primero se refiere a la conversión de características numéricas en símbolos donde es necesario considerar el carácter borroso (fuzzy) de aquellas, de manera que se asocie a los símbolos un cierto índice de fiabilidad o de pertenencia a una clase. El segundo aspecto está íntimamente relacionado con el anterior y tiene que ver con la definición de mecanismos que posibiliten la integración de múltiples fuentes de conocimiento dentro de un nivel o entre diferentes niveles (diferentes dispositivos sensores, diagnósticos obtenidos por diferentes operadores, combinación de evidencias producidas por diferentes hipótesis, etc) [Abid-92][Agga-93]. Debido al carácter borroso o incierto de las asignaciones a símbolos, es necesario que los mecanismos de combinación de evidencias sean capaces de manejar incertidumbres.

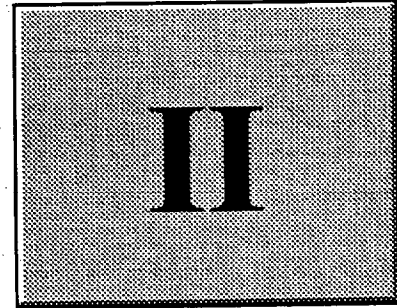
f) Una segmentación fiable.

El éxito de todo sistema de interpretación de imágenes descansa en gran medida en la obtención de una segmentación fiable, siendo muy probablemente la parte dedicada a esta tarea de las más importantes en el IUS. El diseño del módulo de segmentación de un IUS debe contemplar las siguientes cuestiones:

Guiado por objetivos. El módulo de segmentación debe ser capaz de atender peticiones de alto contenido simbólico realizadas desde los niveles superiores del sistema. Estas peticiones se realizarán en términos de conceptos que deben ser inteligibles al módulo de segmentación, pudiendo incluir restricciones en forma de relaciones espaciales. Por ejemplo, debería poder atender una solicitud como: "en la ventana (3,15,100,100), encuentra un círculo rojo con un cuadrado negro en su interior". Los requerimientos comentados anteriormente sobre la necesidad de establecer mecanismos de fusión de la información y de manejo de incertidumbre tienen en la fase de segmentación especial importancia. Los diagnósticos que proporcione el módulo de segmentación deben tener asociado una medida de fiabilidad, de tal manera que se establezca una transición suave entre el dominio numérico o de la imagen y el dominio simbólico o de las clases.

Sistema de segmentación adaptable y estable. Si un IUS debe actuar en un entorno cambiante o manejar imágenes de diferente calidad, es necesario que la segmentación también se adapte a estos cambios. El módulo de segmentación debe ser un sistema experto para segmentación de imágenes en el sentido comentado en el epígrafe I.4, con capacidad para elaborar un plan de acuerdo con los objetivos fijados, y de adaptarlo a las características de la imagen mediante la selección de los operadores y parámetros adecuados. En consecuencia, establecido un conjunto de hipótesis, el resultado final de la segmentación debe ser estable, es decir, se deberán incluir mecanismos que definan la bondad de una segmentación y permitan guiar su consecución.

CAPÍTULO



Elementos para el diseño de un sistema multinivel de segmentación de imágenes.

II.1 INTRODUCCIÓN.

En un proceso de descubrimiento tratamos de explicar lo desconocido, de resolver dudas racionales sobre la realidad y las relaciones que existen entre los elementos que la componen. En nuestra realidad cotidiana como observadores y actores del mundo, a través de la imagen y su significado componemos un modelo de ésta y lo utilizamos en los distintos niveles que constituyen el proceso percepción-acción. El estudio de este proceso forma parte de una de las líneas de investigación a largo plazo del Grupo de Inteligencia Artificial y Sistemas de la Universidad de Las Palmas de Gran Canaria. En ella se trata de hallar solución a ciertos problemas prácticos de interés para encontrar nuevas metodologías e instrumentos de actuación que nos permitan el proceso de descubrimiento, que deberá surgir del contacto íntimo con la realidad y del conocimiento profundo de los procesos percepto-efectores. En el ámbito de los procesos perceptivos visuales, planteamos la definición de una metodología de desarrollo de los sistemas de visión fundamentada en la incorporación de métodos de alto nivel derivados de la Ingeniería del Conocimiento. La metodología conlleva tareas de implementación que se desarrollan de arriba

a abajo, de tal forma que se construyen las capas de más alto nivel, identificando módulos de proceso elementales que sean simples y que, mediante la definición de un conjunto de operaciones también elementales de interacción entre estos módulos, permitan cubrir un cierto espectro de aplicaciones. Estas tareas exigen la necesaria reflexión para evitar errores derivados de una visión parcial y subjetiva de la realidad y de una inadecuada selección del modelo percepto-efector. En cualquier caso un elemento de seguridad o fiabilidad lo establece la concretización que supone la realización práctica y sobre esta base habrá que valorar la bondad y corrección de la metodología propuesta.

En general, los sistemas de Visión por Ordenador suelen ser parte de sistemas más complejos que incluyen actuaciones sobre el medio físico, en algunos casos mediante sistemas robóticos o bien mediante automatismos generales. Desde este punto de vista, un sistema de visión por computador es parte de un **sistema percepto-efector**. Creemos necesario establecer previamente algunos elementos generales o metodología para el desarrollo de sistemas percepto-efectores que ayuden a definir la organización general de tales sistemas y la del sistema de visión en particular.

Con este objetivo, la primera parte de este capítulo se dedica a la exposición de los principios de una metodología de diseño de sistemas percepto-efectores (SPE). La segunda parte describe el resultado de aplicar estos principios al diseño de un sistema adaptable para segmentación de imágenes estructurado en dos niveles: el nivel de los pixels y el nivel de los segmentos.

II.2 CONTRIBUCIONES A UNA METODOLOGÍA PARA EL DISEÑO DE SISTEMAS PERCEPTO-EFECTORES

Como paso previo a cualquier otra consideración, nos plantearemos un conjunto de reflexiones que surgen como posibles respuestas a las siguientes preguntas:

1. ¿Que es una Metodología de Diseño?
2. ¿Que es un Sistema Percepto-Efector?

Consideramos a una metodología como un conjunto de "recetas" o "reglas" que definen una forma de proceder para alcanzar un objetivo. Una metodología generalmente incluye un modelo que permite por un lado, resolver casos prácticos mediante la particularización de los elementos del modelo, y por otro lado proporciona alguna explicación de casos concretos previos a la misma. Una metodología para el diseño de sistemas debe incluir algún modelo en forma de arquitectura o lenguaje de definición, así como ciertos principios heurísticos y sugerencias prácticas.

Un Sistema Percepto-Efector (SPE) constituye un sistema que se desenvuelve e interacciona en un entorno mediante la acción de sensores y efectores. Otras denominaciones equivalentes son las de: *Sistema Autónomo Inteligente* o *Agente Autónomo*. La concepción que exponemos de SPE tiene sus orígenes en experiencias de las áreas de Percepción, Control e Inteligencia Artificial. Una concepción más global de esta disciplina, y quizás más especulativa, ha dado en denominarse en los últimos tiempos como *Vida Artificial* [Mey-91].

Abordamos ahora una versión cualitativa de la metodología. La metodología debe ser aplicable, en principio y al menos desde un punto de vista teórico, a la construcción de sistemas sencillos de control, hasta sistemas complejos de percepción y acción. La aplicación práctica inmediata se centrará en problemas de percepción artificial, control industrial y robótica. Para que una arquitectura de SPE pueda ser eficiente se deben incluir, en alguna medida, procesos de Percepción, Razonamiento, Toma de Decisiones, Aprendizaje, Planificación y Control. Dada esta diversidad de técnicas que deben utilizarse en alguna medida, una denominación adecuada para un SPE eficiente puede ser la de un **Sistema Integrado de Inteligencia Artificial**. Los SPE deben poseer un modelo del entorno o mundo en el que se desenvuelven, así como conocimiento acerca de como debe realizarse la percepción del mundo y como debe ser manipulado el mismo. La metodología es el resultado, por un lado, de la integración de diversas concepciones teóricas en la organización de sistemas y tecnologías de Inteligencia Artificial, y por otro lado de una inducción a partir de sistemas concretos particularmente estructurados en las áreas de Visión Artificial [Rise-84] [Rise-87] [Aloi-88] [Hern-89] [Hern-90] [Mend-92], Robótica [Broo-86]

[Broo-91a] [Broo-91b] [Maes-90] [Elfe-91], y propuestas previas de integración general percepto-efectora [More-86] [More-87].

La presentación se desarrolla en base a varios principios, axiomas, conjeturas o recetas cualitativas que pensamos son de utilidad para los fines propuestos. La validez de tales principios se establece *a posteriori* si mediante su aplicación es posible construir más eficientemente sistemas. La eficiencia, como es lógico, debe entenderse como promedio o tendencia general.

II.2.1 LA ORGANIZACIÓN EN NIVELES.

El primero de los principios fundamentales de la metodología se centra en la utilización de niveles de abstracción. Su utilización constituye una "receta" muy utilizada para reducir la complejidad de diseño. Un nivel de abstracción está constituido por un conjunto de símbolos, conceptos o representaciones que interactúan entre sí mediante procedimientos específicos. La introducción de la organización por niveles genera sistemas estructurados que son más simples de construir.

La organización en niveles es una herramienta que suele utilizarse con relativa frecuencia, en diversas ramas científicas y tecnológicas, cuando se trata de analizar o sintetizar sistemas complejos. Así, en el caso de análisis, el concepto de nivel responde a una forma de ordenar en jerarquías una realidad existente de hechos. Mientras en el caso de síntesis, constituye una forma de simplificar el proceso en base a construir capas de complejidad reducida. Podemos considerar varios elementos teóricos que conducen a la adopción de una organización en base a niveles, en primer lugar, la existencia de casos notables dentro de la tecnología de computadores, en los que la organización en niveles ha demostrado ser fructífera. Ejemplos destacados son los de tecnología hardware, sistemas operativos, redes de computadores o entornos gráficos como XWindow. En segundo lugar, ciertas consideraciones referentes a sistemas complejos naturales o artificiales, por un lado de tipo filosóficas como las de D.R. Hofstadter [Hofs-89], y por otro lado metodológicas como las de Moreno y Mira [More-84] [Mira-84], que comentaremos seguidamente desde nuestra propia óptica.

En el primero de los trabajos de Moreno y Mira [More-84] se establecen consideraciones de cómo debe trabajarse coherentemente en Biocibernética cuando se utilizan diversos niveles. Adoptamos estas consideraciones para otra área de trabajo como es la construcción de SPE, porque creemos que los criterios allí contenidos son generales para amplias áreas de la ciencia y la tecnología. En un sentido no formal, podemos resumir tales consideraciones, desde nuestra óptica, en una fórmula simple como la siguiente:

$$\text{Nivel} = \text{Símbolos} + \text{Operadores}$$

Con tal expresión, queremos destacar el hecho de que lo que define a un nivel es la elección de un conjunto de símbolos que representan las entidades de su dominio, y un álgebra o conjunto de formalismos u operadores que realizan las transformaciones de los símbolos. Desde nuestra concepción podemos resumir las consideraciones metodológicas en la forma siguiente:

1. No existe cambio de nivel cuando no existe cambio del conjunto de los símbolos y/o de sus operadores.
2. No es lícito utilizar, en un nivel, símbolos que no se han declarado explícitamente en el mismo, o que forman parte de otro nivel.
3. La interacción de un nivel con otro se realiza sobre la base de transformaciones de significado de los símbolos de cada nivel. Por ejemplo, se realiza transformación de significado cuando se convierte la tensión eléctrica de un conductor de un circuito con transistores, a un valor booleano. El primer símbolo, la tensión eléctrica, es válido para un nivel como el de transistores, resistencias y leyes Ohm y Kirchhoff, pero el mismo símbolo puede transformarse en otro válido para el segundo nivel, como el que contiene variables y operadores booleanos.

El segundo trabajo considerado [Mira-84] contiene una discusión, que consideramos central, sobre las cualidades de las representaciones en las diferentes capas de un sistema biocibernético. Igualmente adoptamos estas consideraciones para un SPE, y le añadimos un

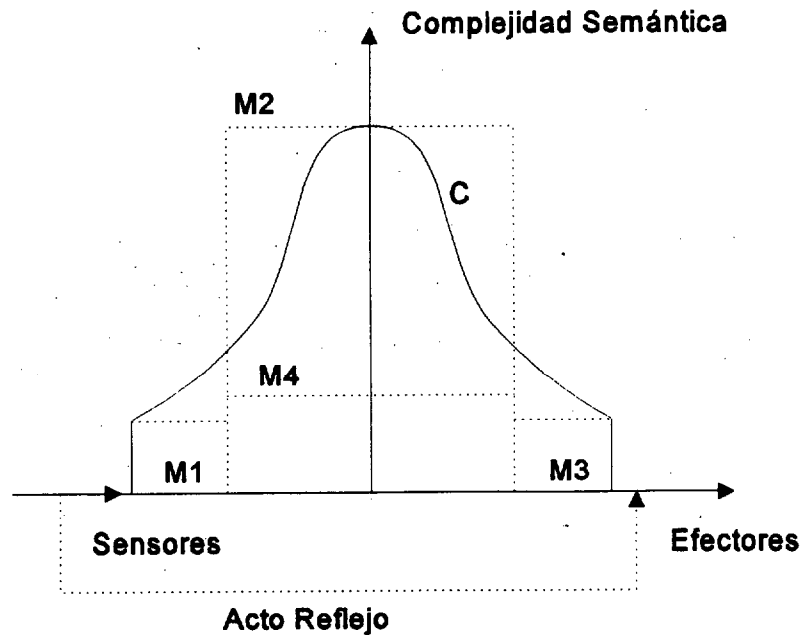


Figura II.1. Diagrama de evolución de la complejidad y abstracción simbólica de los datos en un SPE.

conjunto de interpretaciones que creemos útiles. La figura II.1 contiene una ilustración de las consideraciones que seguidamente realizamos:

1. Un SPE puede considerarse como una máquina en la que existe un flujo de datos procedentes de los sensores hacia los efectores. Estos datos que en la parte izquierda de la gráfica se refieren a representaciones del mundo, es decir a percepción, en la parte derecha se refieren a ordenes de control y a planificación. La curva C representa el nivel de complejidad semántica, o grado de abstracción, de los datos en el trayecto considerado. Éste recorre desde el grado de entrada determinado por los sensores, hasta el grado de salida definido por los efectores, pasando por un grado máximo en las zonas centrales. Este nivel máximo de abstracción representa, según pensamos, el factor que define la potencialidad de un SPE en cuanto se refiere a nivel de "inteligencia" o prestaciones del sistema.

2. La construcción de SPE puede abordarse mediante la utilización de conjuntos discretos de niveles. Comentaremos el caso, bastante común, en el que se suele utilizar dos niveles que pueden denominarse nivel alto y bajo. Consideremos a la máquina M1 como una máquina de bajo nivel relacionada con los sensores. La máquina M3, igualmente de bajo nivel, realiza la misma función con los efectores. Cuando se trata de construir el alto nivel existen dos alternativas:

A. Un nivel alto que contiene el máximo de complejidad semántica adecuada para el sistema, implementado mediante una máquina M2. En este caso las prestaciones potenciales de las máquinas M1-M2-M3 son altas, pero el salto cualitativo en grado de abstracción entre M1 y M3 por un lado y M2 por otro, puede dificultar la realización práctica de esta solución.

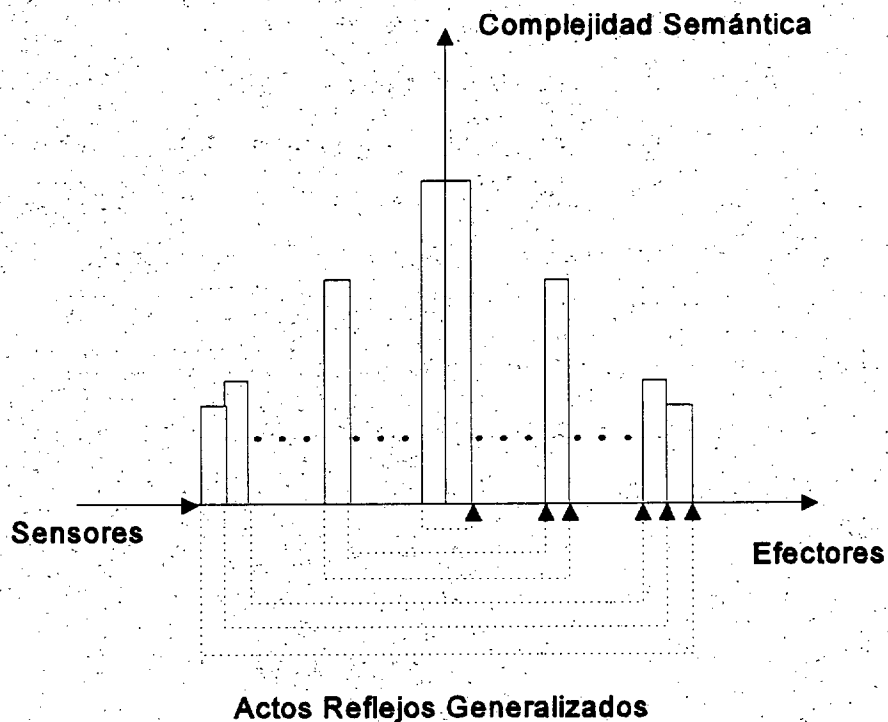


Figura II.2. Realización de un SPE en base a un conjunto de máquinas que implementan niveles con saltos semánticos incrementales y actos reflejos generalizados.

B. Un nivel alto que presenta un salto del grado de abstracción que puede considerarse abordable en la práctica, implementado por una máquina M4. En este caso la posible viabilidad del conjunto M1-M4-M3 tiene su contrapartida en la limitación de la potencialidad del sistema.

Estas consideraciones definen una dicotomía presente en el diseño de SPE que enfrenta por un lado a la realizabilidad, entendida como la superación del salto semántico entre niveles que influye en el diseño práctico; y por otro lado a la potencialidad, entendida como el grado máximo de abstracción en las zonas centrales del SPE. El acto reflejo suele ser un mecanismo presente en los seres vivos, y que cualitativamente constituye una acción directa de los sensores sobre los efectores. Interpretamos tal mecanismo como una solución dispuesta por la evolución para proporcionar una realizabilidad eficiente conservando la potencialidad del sistema. Este mecanismo debe utilizarse sólo en casos extraordinarios, por cuanto rompe con el control que las máquinas centrales M2 o M4 poseen sobre la evolución del sistema.

La trayectoria intelectual que nos conduce al establecimiento del primer principio de la metodología se basa, por un lado en la modificación de la forma práctica de utilizar un símil de la curva de la figura II.1, y por otro en una reflexión sobre el paradigma de organización en niveles en áreas tecnológicas. Consideramos que la construcción de SPE debe abordarse con niveles que contengan saltos semánticos que en la práctica se puedan superar, y con un número de niveles que posibilite la potencialidad que se estime necesaria. Para proporcionar cierto nivel de eficacia se puede utilizar el concepto de **Acto Reflejo Generalizado**, según se ilustra en la figura II.2. Esta consideración del número de niveles como referente de la potencialidad "inteligente" de un sistema, coincide cualitativamente con la expuesta por D.R. Hofstadter [Hofs-89] que considera la organización en múltiples niveles como una necesidad teórica para la acción inteligente. Nosotros, más modestamente, lo consideramos como una necesidad para la realización práctica de sistemas.

La utilización de niveles está ampliamente extendida en diversa ramas tecnológicas. Entre otras destacamos la de diseño estructurado de computadores donde se utilizan niveles de

abstracción asociados, por ejemplo, a transistores, puertas lógicas y bloques MSI. La tecnología de computadores utiliza el paradigma general para organizar los niveles de **red de dispositivos discretos interconectados constituyendo una estructura**, ya sea al nivel de transistor como al de MSI.

La organización de los niveles de un SPE que proponemos, se basa a un paradigma consistente en el concepto de **máquina percepto/efectora que interactúa con un mundo virtual**. El mundo físico lo consideramos un entorno del que se extrae una representación mediante sensores y sobre el que actuamos mediante efectores. Un mundo virtual puede ser bien el mundo físico, o bien el de otra máquina que proporciona datos referidos a la representación de su mundo y que puede recibir ordenes, tal y como muestra la figura II.3. Las funciones de la máquina de un nivel son:

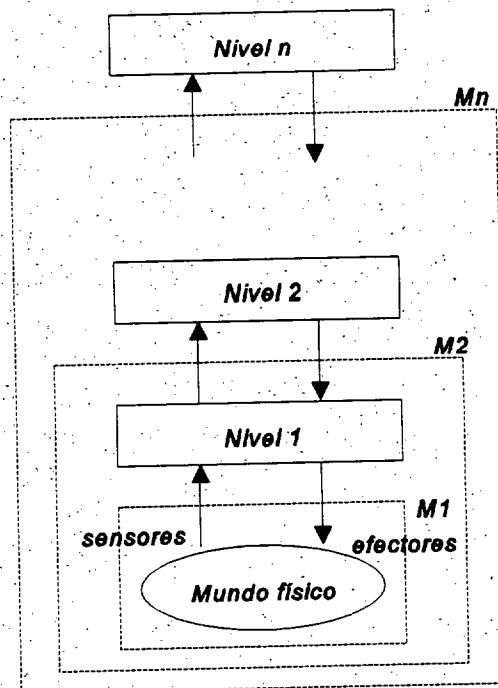


Figura II.3. Ilustración gráfica del Paradigma de estructuración de un SPE en base a Niveles y sus Mundos Virtuales asociados.

Capítulo II

1. Transformar la representación de su mundo virtual en otra más abstracta que suministra al nivel superior.
2. Transformar un conjunto de acciones u órdenes abstractas que recibe del nivel superior en acciones u órdenes más concretas ejecutables sobre su mundo virtual.
3. Si la representación de su mundo virtual presenta ciertas condiciones, entonces realizar ciertas acciones sobre su mundo, independientemente de las acciones ordenadas en niveles superiores.

Podemos resumir el primer principio de la metodología mediante la siguiente "Receta" de tipo práctico:

Receta 1. Para diseñar un SPE, estructurar en niveles el sistema basándose en el paradigma de máquina y mundo virtual. La definición de los niveles es arbitraria. En determinados casos puede basarse en las representaciones del mundo y el conjunto de acciones que se establecen sobre cada una de ellas.

II.2.2 LA ESTRUCTURA DE UN NIVEL.

Los siguientes principios de la metodología se centran en cómo debe ser la organización de los niveles previamente considerados. Lo que nos conduce a la segunda receta que es una consecuencia directa del paradigma básico previo. Se puede expresar cualitativamente como sigue:

"si las funciones de las máquina de cada nivel son formalmente las mismas, pero aplicadas a datos diferentes, entonces la organización interna de cada máquina puede ser formalmente la misma, pero procesando datos diferentes".

En forma de receta práctica se expresa como sigue:

Receta 2. El diseño de cada nivel puede abordarse con cierta economía conceptual en base a suponer que la arquitectura cualitativa de los niveles es común a todos ellos, no así los detalles específicos de cada nivel, dado que tratan símbolos diferentes.

En una organización por niveles cada uno de ellos debe poseer su propio funcionamiento diferenciado, lo que significa que los criterios de construcción deben ser específicos. Cada nivel trata con datos diferentes y por tanto debe poseer procesos diferentes. Nuestro planteamiento es que a pesar de ello, la organización y arquitectura cualitativa de un nivel no difiere manifiestamente del resto. La definición de la arquitectura cualitativa constituye la siguiente receta, que está inspirada en los trabajos de Moreno y Mira [More-86] [More-87] sobre integración multisensorial y efectora, y en los de R.A. Brooks [Broo-86] [Broo-91a] [Broo-91b] relativos a agentes autónomos.

Receta 3. Un modelo de arquitectura para cada nivel basada en la utilización de las Unidades o Procesadores fundamentales BU y TD, e interconectados como se indica en la figura II.4, contiene los elementos necesarios para poder implementar múltiples paradigmas de máquinas percepto-efectoras.

El conjunto de estos dos procesadores constituye lo que denominaremos módulo BU-TD y se definen de la siguiente forma:

Procesador BU (Bottom-Up). Constituye una unidad de diagnóstico, codificación o abstracción. Recibe datos correspondientes a una representación del mundo en un tipo de codificación, y los transforma en otra representación generalmente menos extensa y completa a efectos prácticos. La transformación se fundamenta en detectar ciertos patrones o modelos en la representación inicial y sustituir tal presencia por una referencia al modelo detectado, es decir, recodificar el mensaje en base a un nuevo código, de tal forma que el mensaje es más corto pero a efectos útiles igualmente descriptivo del mundo o entorno. La tarea de la unidad BU se realiza en la forma determinada por ciertas ordenes de control y en base a los modelos y al conocimiento de estos.

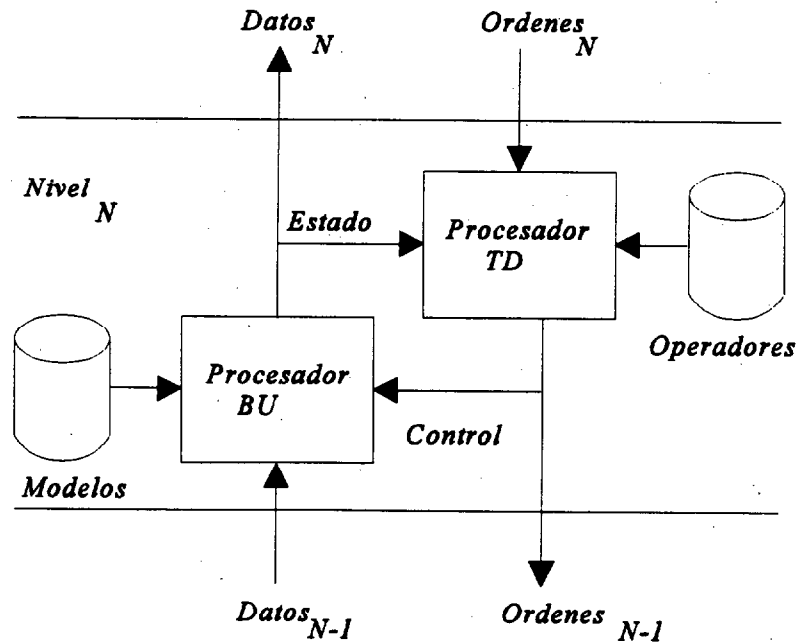


Figura II.4. Modelo de arquitectura genérica de un nivel en un SPE.

Procesador TD (Top-Down). Constituye una unidad de control, planificación, decodificación u operacionalización. Recibe ordenes de alto nivel, por tanto expresadas en forma abstracta, y transforma las mismas en órdenes concretas teniendo en cuenta el estado actual del entorno. Igualmente desencadena acciones cuando en el estado actual se verifican ciertas condiciones, tal proceso conforma lo que denominamos *Acto Reflejo Generalizado*. La transformación u operacionalización de las ordenes se realiza en base a disponer de un conjunto de operadores y al conocimiento de cómo se estructura el mundo virtual en el que opera. El procesador TD es recíproco del BU y se corresponde con el de sistema de control y planificación.

Como ya se ha indicado, es posible desarrollar diversos paradigmas de máquinas SPE basadas en módulos BU-TD. Desde el punto de vista práctico del ingeniero de sistemas, la existencia del procesador TD permite el enlace dinámico de un módulo BU-TD con varios

módulos de un nivel inferior. Esto posibilita una distribución del cómputo y del control a la vez que representa un mecanismo elemental para proveer a un sistema basado en esta arquitectura con capacidad de procesamiento paralelo de grano grueso. La comunicación entre módulos BU-TD de diferentes niveles permite la cooperación y compartición de resultados entre niveles, pero sin que esto conlleve una centralización o jerarquización del control y la consecuente pérdida de reactividad en un nivel gracias a la disponibilidad de Actos Reflejos Generalizados en cada módulo. La figura II.5 pretende ilustrar una posible realización con estos principios de un sistema de tres niveles donde los módulos BU-TD se conciben como unidades de proceso de un nivel. Estas pueden aceptar peticiones desde niveles superiores para ejecutar tareas y proporcionar diagnósticos, controlar la atención de peticiones a uno o varios módulos del nivel inferior o, en el caso de estar situadas en el nivel inferior, ejercer el control de efectores con o sin supervisión desde el nivel superior. En este ejemplo los módulos BU-TD pueden tener asignadas tareas específicas (recibir datos de un sensor, controlar un efector,...) o actuar como unidades de proceso de un nivel con capacidad para cargar tareas bajo demanda de módulos del nivel superior.

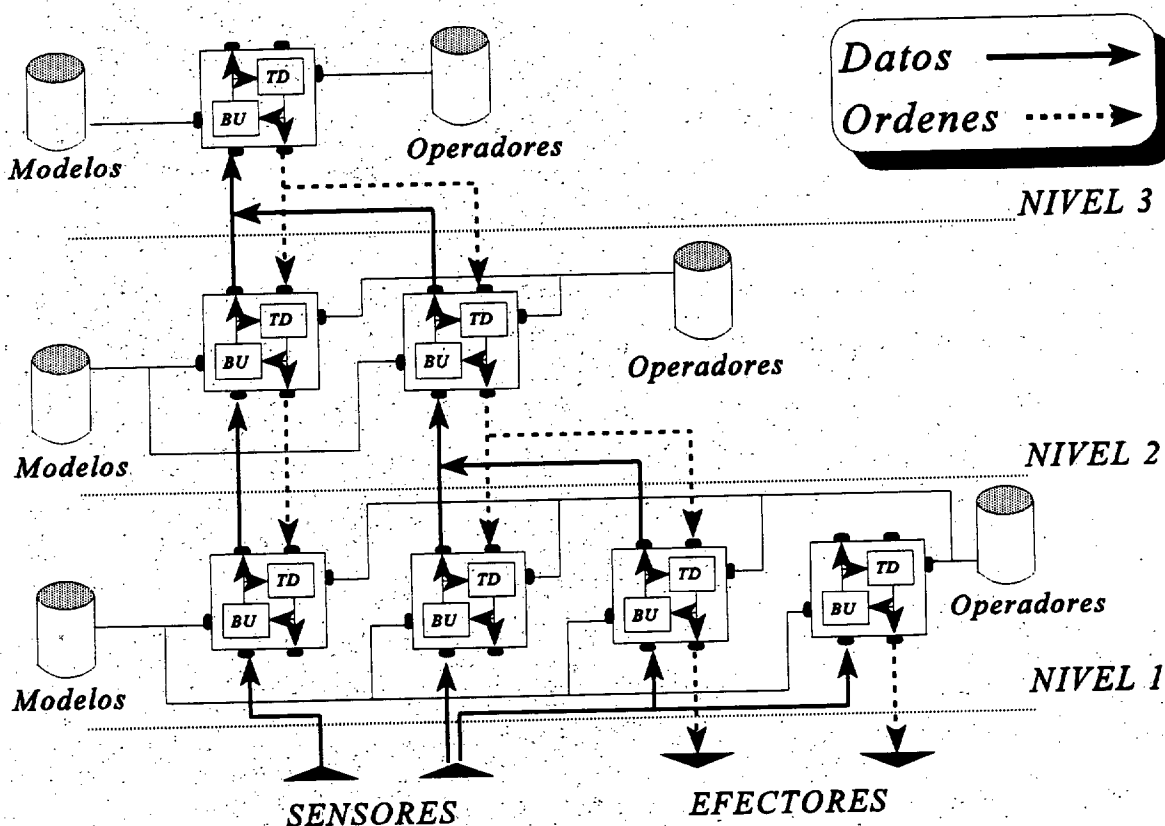


Figura II.5. Un posible SPE de tres niveles con múltiples unidades BU-TD por nivel.

II.2.3 LA REPRESENTACIÓN SIMBÓLICA.

La construcción de máquinas con comportamientos asociados a la inteligencia ha derivado, desde sus comienzos, hacia la consideración de máquinas procesadoras de símbolos en contraposición de las máquinas de números. Esta orientación puede ser debida a que las representaciones simbólicas proporcionan el entorno más adecuado para realizar procesos avanzados, tanto de razonamiento como de representación del mundo, en cualquier nivel más abstracto que en el de sensores y efectores. Aunque esta consideración es de tipo general, no debe ocultar el hecho de que en determinadas condiciones pueda ser más adecuada una máquina básicamente numérica, como por ejemplo una de naturaleza conexionista.

Creemos que las ventajas de la representación simbólica se deben a estos dos supuestos:

1. Economía de la representación mediante símbolos. Creemos que la utilización de símbolos, entendidos como descripciones intensas, redundan en primer lugar, en una economía de recursos.
2. Posibilidad de razonamiento cualitativo más próximo al humano. Creemos que el razonamiento humano, cuando es de tipo cualitativo, conecta mejor con ciertos formalismos de tipo simbólico que con los de naturaleza numérica.

Se puede conjeturar que existe algún principio básico de economía de recursos que obliga a la utilización de descripciones intensas en las máquinas inteligentes, ya sean naturales o artificiales. Una cuestión subyacente en tal principio e importante en el contexto de los SPE es la definición y aclaración de que significado tiene un símbolo, es decir de la semántica del símbolo. Parafraseando a W.S. McCulloch nos preguntamos: ¿Qué es un símbolo?.

Nuestra concepción de lo que es un símbolo proviene de un enfoque más empírico que teórico, y es el apropiado a un ingeniero o arquitecto de máquinas inteligentes. Nuestra concepción de símbolo es de tipo positivista y establecemos que debe **existir una base operacional directa o indirecta para todo símbolo** representado y utilizado por un SPE. Una base operacional implica una tabla de significados como la sugerida por Moreno y Mira [More-

84]. En último extremo, la base operacional de los símbolos debe referirse a los datos de los sensores y efectores de la máquina. Aunque en cuanto a disciplina científica no somos estrictamente seguidores de D. Hume [Russ-64], esta visión de la construcción de SPE es básicamente positivista. Se descarta otras concepciones de símbolo, y por ello puede interpretarse como una versión "degradada" o restrictiva de su semántica.

La "degradación" de los símbolos a la categoría de resúmenes intensos, en último extremo de sensores y efectores, significa que deben existir simultáneamente procesos de codificación-decodificación del símbolo. Es decir procesos de construcción de símbolos a partir de datos y el proceso inverso de representación del símbolo como dato. Esta "degradación" elimina otras posibles semánticas del símbolo, pero en lo que respecta a los fines que perseguimos, lo hace muy valioso para construir máquinas. Igualmente utilizamos una versión "degradada" de los conceptos extenso vs intenso, referida únicamente a criterios de volumen de información. Los elementos que definen lo qué debe considerarse como un símbolo en un SPE se basan en las siguientes consideraciones:

1. Un símbolo es una referencia, indirección o puntero a una representación exhaustiva o extensa. La referencia suele ser reducida, es decir intensa. Por ejemplo, catalogar a un conjunto de puntos alineados como un segmento recto, es construir una representación resumida, posiblemente en una simple etiqueta, pero que conserva lo sustancial del conjunto inicial de puntos.
2. Un símbolo puede ser manipulado independientemente de su descripción extensa, en base a formalismos que lo consideran elemental. En el ejemplo anterior, un conjunto de puntos alineados puede ser sometido a un proceso de detección de paralelismo con otro dado, sin necesidad de utilizar el conjunto extenso de puntos, sino mediante un proceso definido sobre su representación intensa que haga uso de atributos de esta última como puede ser la orientación.

Capítulo II

Lo económico y ventajoso de un símbolo consiste, a nuestro juicio, en la economía relativa que supone realizar operaciones en la representación intensa. Podemos expresar la cuarta receta en la forma siguiente:

Receta 4. Para facilitar la construcción de cada nivel se recomienda utilizar el máximo de construcciones simbólicas altamente explícitas. Ello posibilita por un lado, la incorporación más sencilla de conocimiento sobre la tarea del nivel. Por otro lado, puede proporcionar cierta economía de recursos.

En tal sentido, se recomienda la utilización de construcciones simbólicas usualmente utilizadas en la Ingeniería del Conocimiento como reglas, objetos, jerarquías de objetos y operadores borrosos. Los procesadores BU pueden contener construcciones similares a clasificadores y a reglas de activación hacia-delante, que denominaremos Reglas F, que deben interpretarse en la forma siguiente:

Regla F: Condición₁ \wedge ... Condición_N \rightarrow Conclusión

Donde tanto las condiciones y conclusiones pertenecen al dominio lógico. Los procesadores TD pueden contener construcciones similares a los operadores en planificación y a reglas de activación hacia-delante relacionadas con *Actos Reflejos*, que denominaremos Reglas R, y reglas de activación hacia-atrás, que denominaremos Reglas B. Ambas reglas deben interpretarse en la forma siguiente:

Regla R: Condición₁ \wedge ... Condición_N \rightarrow Acción

Regla B: Objetivo \rightarrow Condición₁ \wedge ... Condición_M \wedge Acción₁ \wedge ... Acción_L

Donde las Acciones pertenecen al dominio de los procedimientos y los Objetivos al dominio lógico. Las reglas B son el elemento básico para dotar al procesador TD con la capacidad de planificar o elaborar planes guiados por objetivos. La interacción y conflictos entre

las acciones desencadenadas por las Reglas R y B constituye un problema que en alguna forma es similar al resuelto por las denominadas "subsumption architectures" o arquitecturas reactivas de R.A. Brooks [Broo-91a] [Broo-91b]. Una explicación sintética de esta arquitectura se puede realizar a partir de la figura II.6. En esta arquitectura cada máquina de nivel de proceso trabaja en paralelo e independientemente una de otra, sin posibilidad de compartir datos, resultados o variables. Cada nivel recibe los datos de los sensores y produce ordenes sobre los actuadores, sin embargo se considera que las ordenes deben ejecutarse en base a cierta jerarquía que define el nivel. Existe un elemento (S, en la figura), básicamente un autómata, que inhibe las acciones determinadas por un nivel superior en favor de las acciones determinada por uno inferior. Más que una arquitectura jerárquica con niveles, esta arquitectura podría definirse como de caminos independientes con criterios de prioridad. El objetivo fundamental de las arquitecturas reactivas es el dotar a las máquinas con un sistema de control totalmente descentralizado que tenga una gran velocidad de reacción, lo cual debe conferirle adaptabilidad, robustez y autonomía. Estas arquitecturas han sido aplicadas con éxito en el desarrollo de robots móviles que realizan actividades de cierta complejidad sobre la base de procesos de percepción-actuación muy simples. Tras el éxito alcanzado a finales de los ochenta y principio de los noventa, el desafío al que se enfrentan los defensores de esta concepción para los SPE es la aplicación de sistemas que incorporen esta arquitectura en tareas más complejas que tradicionalmente han requerido un alto nivel de simbolización.

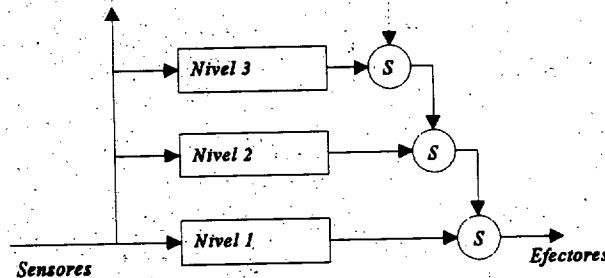


Figura II.6. Ilustración gráfica de la "subsumption architecture" de R.A. Brooks.

II.2.4 LAS TRANSFORMACIONES ENTRE NIVELES.

La arquitectura de máquina percepto-efectora esbozada en los epígrafes anteriores está claramente concebida alrededor del símbolo y de su significación en cada nivel como atributo elemental para ser manipulado formalmente con independencia de su descripción intensa. También se ha discutido la conveniencia de organizar un SPE en múltiples niveles como recurso para posibilitar una mayor potencialidad semántica y hemos indicado la relación existente entre el salto semántico que se produce entre dos niveles consecutivos y su influencia sobre la realizabilidad del sistema. La existencia de este salto semántico plantea desde el punto de vista conceptual y operacional la necesidad de tener en cuenta la transformación de las entidades consideradas elementales (cuantos) en cada nivel sobre la base de su simbolización. Para ilustrar estas consideraciones, elijamos un sistema concreto tal como uno orientado a la comprensión del lenguaje natural que contenga varios niveles. Estos niveles estarían definidos por los símbolos que manejen y a su vez éstos describirán las entidades que se consideran granos de información o cuantos en cada nivel. Así este sistema podría estructurarse en niveles orientados a la señal, al fonema, a la palabra, etc. En cada nivel, el proceso de simbolización consistiría básicamente en intentar asociar modelos a las entidades cuya definición se establece en ese nivel, v.g. identificación de sonidos, fonemas o palabras, bien a partir de los símbolos definidos en el nivel anterior o a partir de la combinación de símbolos definidos en el propio nivel. La mayor dificultad

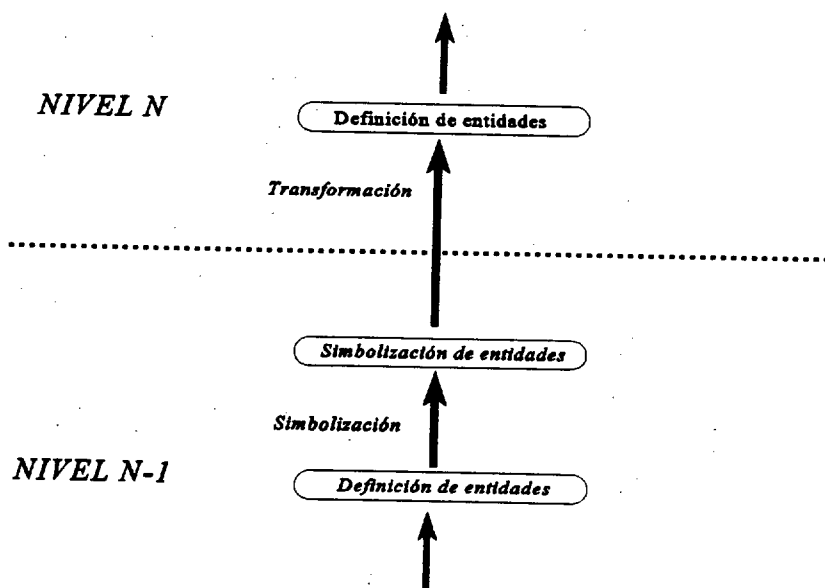


Figura II.7. Procesos de simbolización y transformación de entidades entre niveles.

de este enfoque por niveles se centra en encontrar la transformación que permita obtener operacionalmente las entidades de un nivel a partir de las entidades del nivel inferior sobre la base de su descripción simbólica. Este proceso se repite de nivel en nivel y se representa esquemáticamente en la figura II.7. La transformación entre dos niveles, N-1 y N, puede definirse simbólicamente como:

$$T_N \langle e_1, e_2, \dots, e_k | s_1, s_2, \dots, s_j \rangle_{N-1} = \langle e_1, e_2, \dots, e_m | \cdot \rangle_N$$

donde $\langle e_1, e_2, \dots, e_k | s_1, s_2, \dots, s_j \rangle_{N-1}$ resume la descripción alcanzada en el nivel N-1 mediante la asignación de símbolos $\{s_j\}$ al conjunto de entidades $\{e_k\}$, y $\langle e_1, e_2, \dots, e_m | \cdot \rangle_N$ representa el conjunto de entidades del nivel N generadas a partir de aquellas por la transformación T_N . La transformación T se formula a partir del conocimiento contenido en la base de modelos del nivel y se realiza, por tanto, condicionada por los símbolos definidos en él. Esto es, el conocimiento incluido en la base de modelos de un nivel debe hacer explícita la definición de entidades elementales propias a partir de los símbolos de las entidades del nivel inferior y de la naturaleza de los símbolos del nivel en cuestión.

En general, no existe una regla preestablecida para obtener o definir la transformación T entre dos niveles concretos. No obstante, esta transformación deberá verificar al menos dos requisitos básicos:

1. Deberá considerar la definición de las entidades del nivel N a partir de una cierta teselación del espacio de simbolización del nivel N-1, que será, en general, de carácter difusa.
2. Deberá suponer una compactación de la descripción. Esta compactación tiene dos facetas: una de volumen de datos o número de entidades, en cuanto que las entidades del nivel N se definirán en general por agrupamiento de las entidades del nivel N-1; y otra de compactación del contenido simbólico, por cuanto que los símbolos del nivel N contendrán a otros del nivel anterior, esto es, los símbolos del nivel N-1 son como "matices" entre los elementos que constituyan las entidades del nivel N.

Capítulo II

Todo lo anterior nos lleva a formular de manera no formal la siguiente receta:

Receta 5: La organización por niveles de un SPE debe realizarse con especial consideración hacia la transformación de la representación entre niveles. Dicha transformación deberá establecer una relación explícita entre los símbolos correspondientes a dos niveles que permita la generación de las entidades del nivel superior por compactación de las entidades del nivel inferior en función de su simbolización.

II.3 PROPUESTA DE MÁQUINA MULTINIVEL ADAPTABLE.

La metodología que se ha trazado en el epígrafe anterior nos servirá ahora de base para diseñar un sistema de interpretación de imágenes basado en conocimiento (Image Understanding Systems-IUS). El análisis de diferentes IUS (ver epígrafe I.5) y la receta 1 sugieren la

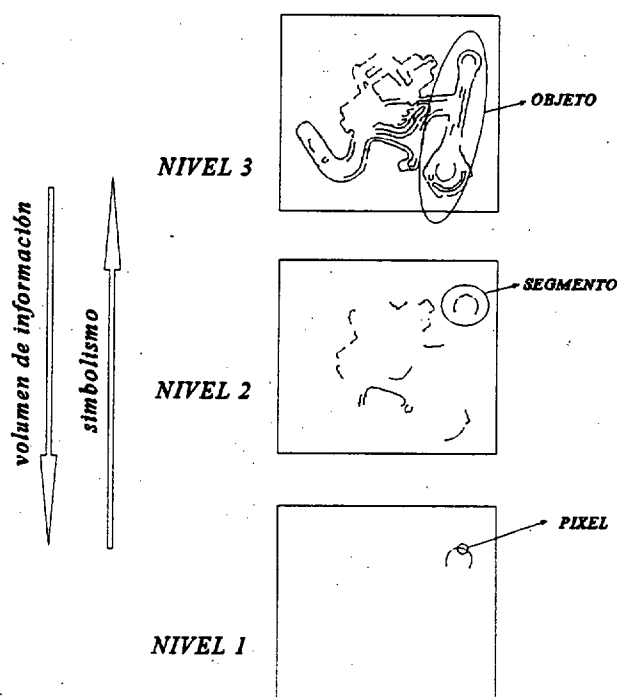


Figura II.8. Definición de niveles para el análisis de formas complejas.

estructuración por niveles como una herramienta útil para diseñar sistemas prácticos. Frecuentemente, un IUS se organiza como una estructura de tres niveles, que podemos denominar como: alto, intermedio y bajo [Rise-84] [Rise-89] [Mats-84], donde cada nivel define una cierta representación de las imágenes y unos procesos que actúan sobre esas representaciones. Usualmente, en el nivel alto, la representación de imagen se basa en símbolos que se asocian con hipótesis referentes a objetos del mundo real. Los procesos típicos de este nivel pueden requerir la comparación de objetos expresados en forma simbólica. El nivel intermedio utiliza representaciones de imágenes basadas en líneas, regiones y superficies, tanto en forma numérica como simbólica. Los procesos característicos de este nivel pueden ser: etiquetado, separación y unión de regiones y/o contornos así como otros procesos relacionados con la segmentación. En este nivel se combinan procedimientos numéricos con representaciones y procesos simbólicos [Nazi-84][Mats-90][Drap-89][Niem-90a]. En el nivel bajo, o nivel de pixel, la representación de las imágenes es básicamente numérica mediante matrices de pixels, y los procedimientos nivel involucran procesos básicos como detección de gradientes, convoluciones y otros operadores orientados al proceso de la señal. La función de este nivel es la de proporcionar indicios y pistas intermedias, tales como bordes, texturas, colores o superficies. Estos indicios proporcionan datos iniciales, así como un camino para la confirmación de las hipótesis de segmentación [Ravi-88]. En este nivel se supone un menor uso del procesamiento simbólico que en los niveles superiores.

II.3.1 ORGANIZACIÓN POR NIVELES Y ARQUITECTURA DEL SISTEMA.

En función del grado simbólico de la representación surgen distintos niveles de tratamiento del problema de representación en una escena. En nuestra propuesta, distinguiremos tres niveles (figura II.8) asociados a entidades que pensamos poseen cualidades significantes para la descripción de los elementos de una escena. Cada uno de estos niveles está caracterizado por sus propios elementos básicos (tokens) o **granos de información** y, conforme ascendemos en los mismos, aparece un grado de simbolización más elevado. Estos niveles están relacionados respectivamente con:

Capítulo II

- A. Los **pixels**, o unidades elementales de información en la imagen.
- B. Los **segmentos** o agregaciones de pixels que poseen una cierta unidad, o en los cuales se verifican predicados de uniformidad.
- C. Los **objetos**, que representan unidades formadas desde la evidencia de la existencia de agregaciones espaciales de segmentos.

En la estructura de máquina que proponemos, para cada nivel existe un procesador (figura II.9) que realiza el tratamiento simbólico en el mismo, efectuándose los procesos de referenciación o de asignación de clases teniendo en cuenta la tipología de la representación en ese nivel. Así distinguimos, consecuentemente con cada tipología de tokens, los **procesadores de Pixels, Segmentos y Relacional o de Objetos**. Cada uno de estos trabaja con los elementos propios de su nivel, con la particularidad que son flexibles en el sentido de la comodidad de inclusión del conocimiento explícito de cada aplicación.

La salida del Procesador de Pixels es un conjunto de diagnósticos o asignaciones a clases de pixels. Estas asignaciones proporcionan pistas para la confirmación de hipótesis de segmentación [Ravi-88]. Los procesos en este nivel son fundamentalmente numéricos y en él se desarrollan procedimientos sobre las imágenes tales como detección de gradiente, convoluciones y otras operaciones propias de proceso de señal. Aquí la segmentación es inmediata pues las correspondientes zonas de uniformidad en la salida se corresponderán con la agregación de pixels espacialmente

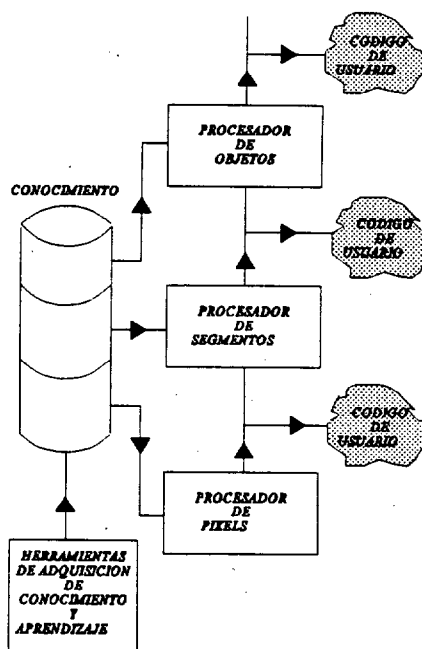


Figura II.9. Esquema de Arquitectura Adaptable.

conexos que tengan similar asignación a una cierta clase de pixels. Esta salida es propia de ciertas aplicaciones como las de inspección o control de calidad.

La salida del Procesador de Segmentos son agregaciones de pixels vertebradas en función de factores de forma o por ser uniformes en propiedades. Dichos elementos son una lista de regiones con sus respectivas asignaciones a clases de segmentos o diagnósticos simbólicos, sus localizadores espaciales y el correspondiente grafo de regiones adyacentes. En este nivel, la representación de los tokens se gestiona a la vez en forma numérica y simbólica, y los procesos que en el acontecen participan de ambos tipos de representación. La salida es de utilidad para aquellas aplicaciones de reconocimiento donde se precise establecer una detección y localización de formas simples en la escena en el sentido de los postulados de Nieman [Niem-90a].

Por último para el análisis de formas complejas, se hace necesaria la utilización del Procesador de Objetos, que realiza una comparación entre los grafos de la escena proporcionados por el Procesador de Segmentos y el de los modelos de objetos, buscando las posibles correspondencias [Hern-89]. En este nivel las entidades que se manejan son formas complejas esto es, definidas por un conjunto de formas simples suministradas por el Procesador de Segmentos, y un conjunto de relaciones entre ellas. En este caso, el tratamiento es fundamentalmente simbólico.

El desarrollo experimental de este trabajo de investigación se circunscribe al ámbito de problemas relacionados con la segmentación de imágenes e involucra los dos primeros niveles antes mencionados: el nivel de pixels y el nivel de los segmentos. Los objetivos de diseño para estos niveles son los siguientes:

- Definir un conjunto reducido de objetos y operadores que permitan construir aplicaciones generales de segmentación a nivel de pixels y de segmentos.
- Construir un sistema que defina e identifique claramente las dependencias entre datos y donde el flujo de estos se pueda determinar con facilidad.

Capítulo II

- Definir un marco de trabajo donde sea posible combinar procesos de alto nivel, con construcciones simbólicas como jerarquía de clases, reglas y conceptos de lógica difusa; con procesos de bajo nivel, esto es la inclusión de procesos fundamentalmente numéricos.

- Definir una arquitectura invariante de manera que un cambio en la aplicación sólo se refleje en variaciones de la programación del sistema, quedando inalterable la estructura de procesos.

- Definir un lenguaje de programación específico a la arquitectura propuesta y el correspondiente compilador.

El lenguaje de propósito especial que proponemos y que describiremos a lo largo de los siguientes epígrafes, permite al usuario expresar su aplicación y añadir en los términos del lenguaje aquel conocimiento explícito que estime relevante. El proceso genérico de construcción de una aplicación está esquematizado en la figura II.10. El usuario define cada aplicación en este lenguaje, y puede incluir otros ficheros escritos por expertos en visión para producir un aplicación totalmente ejecutable. Esta labor se desarrolla en dos pasos: la primera trata de la traducción del código de conocimiento a estructuras del lenguaje C, y el segundo paso realiza la compilación de estas estructuras a código máquina, así como el enlace con librerías de procedimientos y otros códigos específicos del usuario. La ventaja de la adaptabilidad permitirá que un cambio en la aplicación sólo se refleje en una variación del conocimiento explícito necesario, expresado con el lenguaje. Tal cambio no supondrá variación alguna en la estructura de procesos ni en la arquitectura del sistema, que queda invariante.

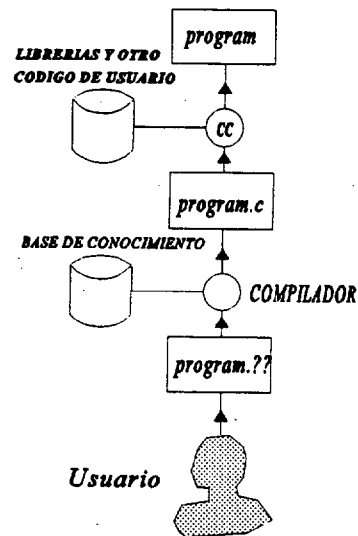


Figura II.10. Proceso para la generación de una aplicación.

Los procesadores que se proponen para los dos primeros niveles poseen una estructura modular, y los distintos módulos que la componen están asociados a objetos que se definen en este lenguaje de propósito especial. Estos objetos poseen una serie de atributos relacionados con las funciones del mismo y se incluyen en el lenguaje mediante definiciones de la forma:

```
Define      tipo_objeto  nombre_objeto
            atributo1
            .....
            atributon

EndDefine
```

II.3.2 EN EL NIVEL DE LOS PIXELS.

Exponemos cómo las recetas de la metodología considerada han sido utilizadas para diseñar el nivel más bajo de este Sistema de Interpretación de Imágenes. Las especificaciones que se establecen inicialmente sobre los procesadores BU y TD en el nivel de los pixels son las siguientes:

1. El procesador BU recibe datos de entrada consistentes en imágenes procedentes de sensores de cámara y proporciona como datos de salida, imágenes de diagnóstico simbólico. Utilizará conocimiento explícito referente a los modelos de las clases de diagnósticos.
2. El procesador TD recibe órdenes consistentes en la petición de diagnósticos, y transforma las mismas en secuencias de órdenes de ejecución de tareas del procesador BU, así como órdenes hacia los controladores de cámaras. La transformación de órdenes es consecuencia de las relaciones de dependencia funcional expresadas en la definición del procesador BU, por ello, y dado que no se implementan actos reflejos, el procesador TD no requiere definición explícita.

Capítulo II

Como ya se ha dicho, no existe una solución general para construir IUS, más bien, se utilizan soluciones dependientes del dominio. La inteligencia Artificial y la Ingeniería del Conocimiento son disciplinas donde la resolución de problemas en forma específica y oportunista es una parte importante de los conceptos nucleares [Fox-90] que pueden ser aplicadas con éxito en dominios con representación simbólica, en los cuales se puede expresar más fácilmente la heurística y la experiencia humana.

En el nivel de los pixels se emplean dos representaciones. Una primera numérica, en forma de mapas de características (gradiente, color, ...), obtenida a partir de los datos proporcionados por los sensores mediante algoritmos de proceso de imágenes. La segunda representación maneja mapas de símbolos o imágenes simbólicas generadas por procesos de simbolización de los mapas de características. En este trabajo, consideramos que las imágenes simbólicas son mapas en los cuales el valor de cada pixel es el grado de pertenencia del pixel a una tipología o clase de pixel, como por ejemplo *PixelVerde* o *PixelAltoGradiente* [Wils-88a]. Con estos elementos, es posible realizar computación simbólica, dado que la representación se basa en clases y tipos que se pueden considerar como símbolos más que números. Como consecuencia, se pueden utilizar operaciones relacionadas con jerarquías de clases y relaciones entre ellas, tales como reglas o producciones. La utilización de grados de pertenencia a clases posibilita la utilización de clasificadores borrosos y coeficientes de certeza en las reglas.

El objetivo primero de este apartado consiste en describir un procesador BU en el cual las representaciones principales son imágenes de características, esto es datos numéricos, e imágenes de clases, esto es datos simbólicos (figura II.11). En este trabajo, el nivel alto vs. el bajo, no es equivalente a computación simbólica vs. numérica, dado que una hipótesis sustancial del trabajo consiste en que la computación simbólica es posible en cualquier nivel de un IUS. Sin embargo, a efectos de eficiencia, es en el nivel de los pixels donde se demanda una mayor capacidad de cómputo numérico.

II.3.2.1 ARQUITECTURA DEL PROCESADOR DE PIXELS Y LENGUAJE DE DEFINICIÓN.

Tal y como hemos planteado, el entorno a describir se compone de dos dominios. El primero, dominio de las características, utiliza imágenes de características y operadores [Hara-91] denominados procedimientos. Los operadores se aplican sobre las imágenes o sobre otras características para producir nuevas características. El segundo, dominio de las clases, utiliza imágenes de clases y operadores, que denominados: clasificadores, reglas y jerarquías, computan clases desde características o desde otras clases. La figura II.12 muestra todos los tipos de imágenes y los operadores que las transforman.

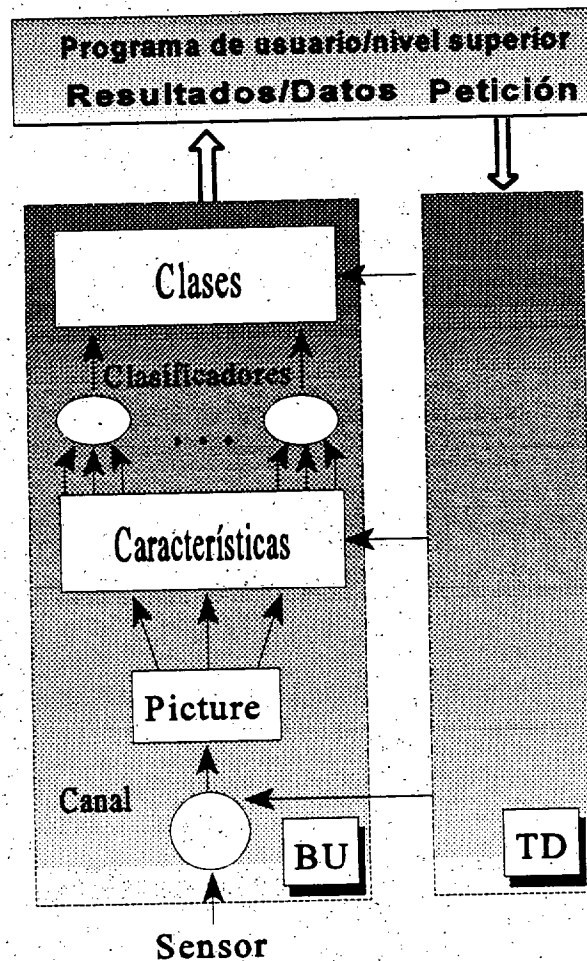


Figura II.11. Estructura de procesadores y dominios en el nivel de pixels.

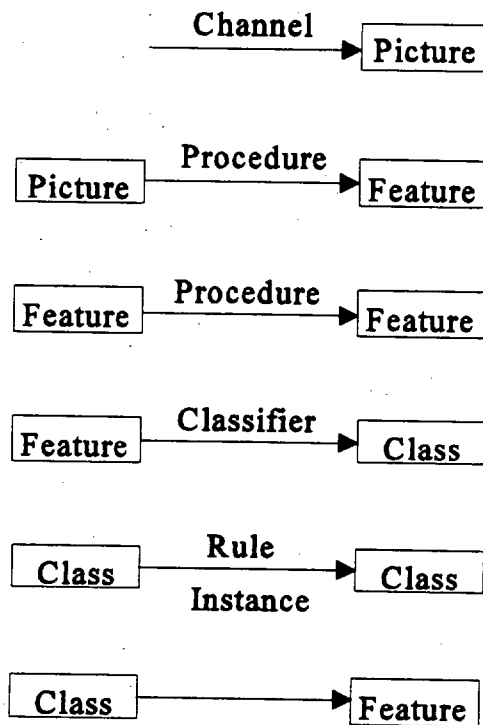


Figura II.12. Tipos de imágenes y su transformación mediante operadores. Las imágenes se encuentran en cajas y los operadores sobre las líneas de transformación.

Para facilitar el uso del procesador de pixels desde niveles superiores, el control del Procesador de Pixels está orientado por objetivos. Los procesos de computación se desencadenan cuando se solicita una petición de cómputo de una clase. Esta petición se recibe por el procesador TD y se transforma en acciones de cómputo concretas en el BU. Los módulos BU y TD se definen en un lenguaje de propósito especial en el que los operadores "Procedure" y "Channel" son externos y deben definirse en algún lenguaje eficiente de propósito general o bien en algún lenguaje orientado al proceso de imágenes [Ritt-90]. El entorno que estudiamos puede ser útil en dos tipos de aplicaciones:

1. Para definir un procesador de bajo nivel en un IUS general.
2. Para construir un sistema autónomo que no necesita un procesador de alto nivel. Frecuentemente un diagnóstico sobre pixels utilizado por un código específico de usuario es suficiente para algunas aplicaciones.

Sintéticamente, los tipos de objetos utilizados en el nivel del Procesador de Pixels son: "Channel", "Picture", "Procedure", "Feature", "Classifier", "Class", "Rule" e "Interface". El tipo "Picture" identifica la imagen de entrada. El tipo "Feature" está relacionado con propiedades o características de la imagen, obtenidas por la aplicación de "Procedures". El tipo "Class" se aplica a imágenes simbólicas, sobre las que se pueden imponer condiciones lógicas para definir otras clases por medio del objeto "Rule". Por último, el tipo "Interface" define los nombres de las clases que son visibles a los niveles superiores.

II.3.2.2 DESCRIPCIÓN DE OBJETOS. PROCESADOR BU.

El procesador BU se define en base al conjunto de objetos enunciados previamente. Describiremos ahora mediante ejemplos cómo se declaran y cuál es el significado de sus atributos o campos. Un objeto de tipo "Channel" se corresponde con la declaración de un sistema de adquisición de imágenes, canal o fuente de las mismas. Cada objeto Channel debe contener el nombre de un procedimiento que ejecuta el procesador TD para realizar la adquisición de una imagen. Una definición de un objeto de tipo Channel podría ser la siguiente:

```
Define Channel DT2803
  Type:      Grey
  SizeY:     256
  SizeX:     256
  Level:     64
  Homothety: Homothety3_2
  Function:  DTAcquire
EndDefine
```

En este ejemplo se define un canal de entrada de imágenes en blanco y negro, con un tamaño de 256x256 pixels, 6 bits de precisión, es decir 64 niveles, y una razón de aspecto u

Capítulo II

homotecia espacial de 1.5. El token *Homothety3_2* está predefinido en el lenguaje, y forma parte de un conjunto de posibles tokens (*Homothety4_3*, *NoHomothety*) que identifican las razones de aspecto presentes en la práctica totalidad de los digitalizadores de imágenes. Cuando el procesador TD debe realizar la adquisición de una imagen mediante este canal, ejecuta el código de la función denominada *DTAcquire*, que debe incorporarse en el momento del montaje de la aplicación. Se han definido diversos tipos de canales o "Channels": *Grey*, *Rgb*, *Hsi*, *Sequence* y *Set*. Los tipos *Rgb* y *Hsi* corresponden a adquisición de imágenes en color. El tipo *Sequence* identifica una secuencia temporal de imágenes, y *Set* se corresponde con varias imágenes en blanco y negro de una misma escena adquiridas, por ejemplo, con diferentes tipos de fuentes de luz. El tipo *Grey* comprende una única imagen, mientras que los demás incluyen diversas imágenes, denominadas "slices", y que pueden ser utilizadas separadamente.

El objeto "**Picture**" sólo puede declararse una vez en la aplicación, define la imagen de trabajo y contiene el nombre del canal que debe ser utilizado. Si el canal está orientado a la utilización de ficheros, entonces se debe especificar el nombre del mismo. Existen otros atributos opcionales, tales como comandos de visualización, factor de zoom o para etiquetar la presentación visual. Un ejemplo de declaración de un objeto "Picture" sería:

```
Define Picture BlocksPicture
  Channel:      DTFile
  File:         "/blocks.pic"
  Show
  Zoom:         2
  Label:        "Es una imagen de bloques"
EndDefine
```

Los objetos "**Procedure**" definen la forma de utilización de los procedimientos. Estos últimos han de escribirse en cualquier lenguaje de programación de propósito general. Los procedimientos pueden aplicarse al objeto "Picture" para obtener objetos del tipo "Feature", que contienen imágenes de cualidades o características numéricas. Los procedimientos también pueden aplicarse a características ("Features") para obtener otras. En el primer caso, de aplicación a "Picture", debe incluirse un atributo que indique si el procedimiento se puede aplicar a

componentes ("slices") individuales, o debe aplicarse a todo el objeto "Picture" en conjunto. En el primer caso el procedure se denomina *Sliced*, y en el segundo *NoSliced*. Por ejemplo, en la computación del gradiente en una imagen en color, puede utilizarse un procedimiento *Sliced* para ser aplicado individualmente a cada una de las componentes de la imagen, y a partir de los tres resultados calcular el gradiente total. Por el contrario, para la computación de la intensidad de luz en una imagen en color, debe utilizarse un procedimiento *NoSliced* que la evaluará a partir de las tres componentes cromáticas. Este último procedimiento no es aplicable a componentes individuales. El lenguaje utiliza también un atributo para la inclusión de parámetros de los procedimientos. Para aumentar la eficiencia, algunos procedimientos pueden computar varias características. Un ejemplo destacado, lo constituye la computación de funcionales estadísticos, donde la computación de la varianza produce automáticamente también la media. En estos casos, el acceso a las imágenes de características individuales se realiza mediante un *selector*, que constituye un índice dentro del objeto "Feature". Los próximos ejemplos muestran Procedures aplicados a "Picture" y a "Feature":

```
Define Procedure      LocalStatistics
  Function:          Stat2
  AppliedTo:         Picture
  Sliced
  ParamNum:          1
  Selector:           mean, variance;
EndDefine
```

```
Define Procedure      Hypotenuse
  Function:           Hypot
  AppliedTo:          Feature
  ArgNum:             2
  ParamNum:           0
EndDefine
```

El primer ejemplo define un procedimiento denominado *LocalStatistics* que es computado por aplicación de la función Stat2. Se aplica a componentes individuales de una imagen y requiere de un parámetro, lo cual está indicado por el atributo *ParamNum*; su salida son dos características seleccionadas mediante los selectores: *mean* y *variance*. El segundo ejemplo, define un

Capítulo II

procedimiento denominado *Hypotenuse* calculado mediante la función *Hypot*. En este caso, (atributo *ArgNum*), se aplica a dos imágenes de características y no necesita parámetros.

Los objetos "**Feature**" definen imágenes de propiedades o características, como por ejemplo:

```
Define Feature GreenStatistics
  From:      Picture
  Slice:     Green
  Procedure: LocalStatistics
  Parameter: 16
EndDefine
```

```
Define Feature      Gradient5x5Total
  From:             Feature
  FeatureList:     Gradient5x5.column, Gradient5x5.row;
  Procedure:       Hypotenuse
EndDefine
```

En el primer caso se utiliza el procedimiento definido en el ejemplo previo, para computar la estadística local en un entorno o ventana de 16x16 centrado alrededor de cada pixel. Se aplica a una imagen de tipo Rgb, dado que se utiliza la componente *Green*. Los resultados se utilizan mediante selectores en la forma de *GreenStatistics.mean* y *GreenStatistics.variance*, de acuerdo con la definición del procedimiento *LocalStatistics*. El segundo ejemplo define una característica denominada *Gradient5x5Total* desde otras dos mediante un procedimiento que computa la hipotenusa geométrica. Las características pueden evaluarse también a partir de la conversión automática de imágenes de clases simbólicas en imágenes numéricas, como muestra el ejemplo siguiente:

```
Define Feature Numeric_Image
  From: Class
  ClassName: Symbolic_Image
EndDefine
```

Los objetos "Classifier" son básicamente son funciones discriminantes y de decisión, y aportan una mayor riqueza de significado. Utilizan imágenes de tipo numérico para computar una de tipo simbólico y son, por tanto, el instrumento para acceder al dominio simbólico desde el dominio numérico. En una imagen simbólica el valor de cada pixel constituye el grado de pertenencia del mismo a una clase. El rango del grado de pertenencia se establece en el intervalo entero [0-100]. El objeto "Classifier" se compone de dos procesos, tal y como muestra la figura II.13, el primero, denominado funcional ("Functional"), define las características que se utilizan y cómo se combinan. El segundo proceso, denominado decisión ("Decision"), es el que transforma el resultado del anterior en un rango borroso de [0-100]. En esencia, el objeto "Classifier" es una implementación de procedimientos de clasificación incluyendo decisiones borrosas mediante el cual se pueden implementar decisiones más complejas [Rise-87].

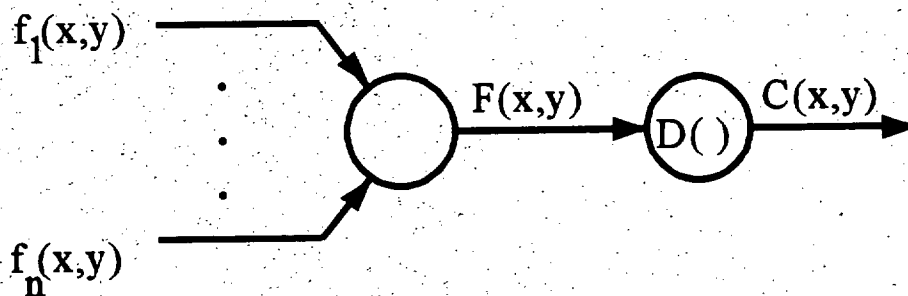


Figura II.13. El objeto "Classifier" incluye dos procesos, el primero computa una combinación de Features, y el segundo la transforma a un rango de lógica borrosa.

En el contexto del sistema, se permiten tres tipos de funcionales, denominados *Unitary*, *Lineal* y *Quadratic*. El primero utiliza una única característica, mientras que los demás utilizan combinaciones de primer y segundo orden respectivamente. Sea $f_i(x,y)$, $i=1, \dots, n$, un conjunto de características, $F(x,y)$ un funcional computado desde ellas, y $C(x,y)$ la imagen resultado suministrada por el clasificador. Se tiene:

$$C(x,y) = D(F(x,y)) \quad (\text{II-1})$$

Capítulo II

Donde $D(u)$ es una función de decisión. El funcional *Lineal* es una combinación de primer orden, que se calcula como sigue:

$$F(x, y) = \sum_{i=1}^{i=n} a_i f_i(x, y) \quad (\text{II-2})$$

Donde $\{a_i\}$ representa un conjunto de coeficientes numéricos. El funcional *Quadratic* es una combinación de segundo orden de la forma:

$$F(x, y) = \sum_{i=1}^{i=n} \sum_{j=1}^{j=n} [f_i(x, y) - a_i] b_{ij} [f_j(x, y) - a_j] \quad (\text{II-3})$$

Donde $\{a_i\}$ y $\{b_{ij}\}$ son conjuntos de coeficientes numéricos. En el lenguaje, se incluyen varios modelos de funciones de decisión, expresados como $D_M(u; p_1, \dots, p_n)$, donde M es el tipo de modelo (SS, SP, T, R, P, E, C y TR) y p_1, \dots, p_n son sus parámetros. Los modelos SS y SP se basan en la función tipo-S de conjuntos borrosos [Zade-81] [Zade-75], definida por $S(u; \alpha, \beta)$ en la forma:

$$S(u; \alpha, \beta) = \begin{cases} 0 & u \leq \alpha \\ 2 \left(\frac{u - \alpha}{\beta - \alpha} \right)^2 & \alpha < u \leq \frac{\alpha + \beta}{2} \\ 1 - 2 \left(\frac{\beta - u}{\beta - \alpha} \right)^2 & \frac{\alpha + \beta}{2} < u < \beta \\ 1 & \beta \leq u \end{cases} \quad (\text{II-4})$$

Y la función de decisión $D_{SS}(u; \alpha, \beta)$ queda como:

$$D_{SS}(u; \alpha, \beta) = 100S(u; \alpha, \beta) \quad (\text{II-5})$$

Esta definición se selecciona en el lenguaje por la palabra *Interval*, donde los parámetros α y β definen el intervalo de borrosidad entre la decisión nula(0) y total(100). También es posible otro tipo de decisión mediante la utilización de la palabra *Centre*, en cuyo caso α es el parámetro de decisión central y β es el intervalo de borrosidad:

$$D_{SS}(u; \alpha, \beta) = 100S(u; \alpha - \beta, \alpha + \beta) \quad (\text{II-6})$$

En este caso si: $\beta=0$, se obtiene una decisión de tipo "duro". El modelo SP está basado en la función Π definida desde la función S como sigue [Zade-75]:

$$\Pi(u; \alpha, \beta) = \begin{cases} S(u; \alpha, \frac{\alpha + \beta}{2}) & u \leq \frac{\alpha + \beta}{2} \\ 1 - S(u; \frac{\alpha + \beta}{2}, \beta) & \frac{\alpha + \beta}{2} < u \end{cases} \quad (\text{II-7})$$

mientras que la decisión SP de tipo *Interval* es:

$$D_{SP}(u; \alpha, \beta) = 100\Pi(u; \alpha, \beta) \quad (\text{II-8})$$

donde los parámetros α y β definen el intervalo de decisión no nula. La decisión de tipo *Centre* se define como:

$$D_{SP}(u; \alpha, \beta) = 100\Pi(u; \alpha - \beta, \alpha + \beta) \quad (\text{II-9})$$

donde α es el centro y máximo de la decisión, y β es el intervalo de borrosidad. El modelo T es de tipo umbral:

$$D_T(u; \alpha) = \begin{cases} 0 & u \leq \alpha \\ 100 & u > \alpha \end{cases} \quad (\text{II-10})$$

Capítulo II

El modelo R es de tipo borroso, con una curva de transición lineal como sigue:

$$D_R(u; \alpha, \beta) = \begin{cases} 0 & u \leq \alpha \\ 100 \frac{u - \alpha}{\beta - \alpha} & \alpha < u < \beta \\ 100 & \beta \leq u \end{cases} \quad (\text{II-11})$$

El modelo P se corresponde con una decisión "dura" de tipo pulso:

$$D_P(u; \alpha, \beta) = \begin{cases} 0 & u \leq \alpha \\ 100 & \alpha < u < \beta \\ 0 & \beta \leq u \end{cases} \quad (\text{II-12})$$

El modelo E se basa en una exponencial, como sigue:

$$D_E(u) = 100 \exp(-|u|) \quad (\text{II-13})$$

que en conjunción con un funcional cuadrático permite definir un clasificador de tipo gaussiano, es decir:

$$C_G(x, y) = 100 \exp\left(-\sum_{i=0}^{i=n} \sum_{j=0}^{j=n} [f_i(x, y) - a_i] b_{ij} [f_j(x, y) - a_j]\right) \quad (\text{II-14})$$

El modelo C es una función de decisión formalmente definida como:

$$D_C(u; \alpha) = \begin{cases} 0 & u \neq \alpha \\ 100 & u = \alpha \end{cases} \quad (\text{II-15})$$

Dada la cuantización espacial y numérica propia de las imágenes digitales, esta función de decisión debe implementarse mediante la detección de cruces sobre el valor del parámetro α . Este modelo se utiliza, entre otras aplicaciones, para obtener los cruces por cero en la detección de contornos [Marr-82].

El modelo TR es un clasificador trapezoidal que se define formalmente como:

$$D_{TR}(u; \alpha, \beta, \gamma, \delta) = \begin{cases} 0 & u \leq \alpha \\ 100 \frac{u - \alpha}{\beta - \alpha} & \alpha \leq u \leq \beta \\ 100 & \gamma \leq u \leq \delta \\ 0 & u \geq \delta \end{cases} \quad (\text{II-16})$$

Las funciones de los objetos "Classifier" guardan cierta analogía con las propiedades de ciertas neuronas artificiales [Lipp-87] [Gall-88], dado que algunas de los modelos más simples de neuronas se componen de funcionales lineales y decisiones de tipo umbral. Un red neuronal por capas puede considerarse como un conjunto de capas, en la cual la primera evalúa decisiones simples a partir de los datos, y las siguientes capas evalúan decisiones derivadas y más complejas. En nuestro entorno de trabajo, los clasificadores y clases pueden ser vistos como un sistema de decisión con múltiples capas, en el cual la primera capa, o de clasificadores, generaliza la primera capa de un sistema neuronal, y las siguientes capas se implementan mediante otras herramientas, tales como reglas e "instances". Como ejemplo, una neurona formal simple con una decisión de tipo umbral, es decir modelo T, se puede expresar como [Lipp-87]:

$$y = T\left(\sum_{i=1}^N w_i f_i - \theta\right) \quad (\text{II-17})$$

Este modelo de neurona se puede expresar en el lenguaje como:

```

Define Classifier SimpleNeuron
  FeatureList: f1, ..., fN;
  Functional: Lineal
  Coefficient: w1, ..., wN;
  Decision: Threshold
  Sign: Positive
  Level: θ
EndDefine
    
```

El objetivo principal del procesador BU es obtener las asignaciones a clases, sintetizadas en los objetos "Class". Una clase puede ser evaluada desde diversas vías, o fuentes, como son:

Capítulo II

clasificadores, reglas y particularizaciones (instances). Cada una de estas vías define una contribución que debe combinarse con las otras para la evaluación total de una clase. La definición de una clase incluye los nombres de todas las fuentes mediante los atributos denominados *Instances*, *Classifiers* y *Rules*. En el sistema se definen dos leyes para realizar la combinación de fuentes. La ley utilizada en cada clase se selecciona mediante un atributo opcional: *Logic*, que es la de defecto, y *Evidence*. El próximo ejemplo contiene la definición de una clase denominada *Green* que se computa desde varias fuentes. En la primera de estas se utilizan dos clases que son particularizaciones de la citada y que se denominan: *GreenGrass* y *GreenTree*, ambas asociadas a tipos especiales de la cualidad verde. La segunda fuente la constituye un clasificador denominado *GreenFromData*, que evalúa la contribución a la clase a partir de la imagen de entrada. La tercera, y última, fuente es una regla denominada *GreenEnvironment*, que computa una contribución basándose en el entorno. Todas las contribuciones se combinan mediante la ley *Evidence*.

```
Define Class Green
  Instances:   GreenGrass, GreenTree;
  Classifiers: GreenFromData;
  Rules:       GreenEnvironment;
  Combine:     Evidence
  Show
    Overlapped: 70
    Label: "Esta es la clase verde"
EndDefine
```

El atributo *Show* es opcional, y proporciona una forma de visualización de los resultados de la clase, en tal caso, se pueden utilizar estos otros: *Overlapped* y *Label*. Con el primero el resultado es visualizado superpuesto con la imagen original en los pixels en los que el valor de la clase exceda el valor umbral definido.

Los objetos "**Rule**" son construcciones lógicas, similares a la cláusulas de Horn [Kowa-79]. Una regla define una contribución a una clase en un pixel basada en los valores de otras clases en el mismo pixel o en sus vecinos. Cada regla tiene un nombre, una conclusión denominada *Target*, un factor de certeza opcional en el rango [0-100], y una condición compuesta

de predicados interconectados mediante operadores And. Un predicado se define desde una clase, y puede incluir datos adicionales, tales como cualificadores lingüísticos y un entorno. La definición sintáctica de un predicado es como sigue:

```
predicado ::= [ Not ] [ lingüístico ] nombreclase [ ( entorno ) ]  
lingüístico ::= Very | MoreLess  
entorno ::= NB nb | AllNB | AnyNB  
nb ::= 0|1|2|3|4|5|6|7
```

Donde el cualificador *Not* constituye la negación lógica, mientras los cualificadores *Very* y *MoreLess* se corresponden con los operadores lingüísticos borrosos *very* y *more-or-less* [Zade-81] [Zade-83]. Si no se define el entorno, entonces la evaluación se realiza sobre el pixel de trabajo, en otro caso, el término NB define qué vecino es utilizado para la evaluación, mediante una numeración ampliamente utilizada [Pavl-77]. Si se incluye el término *AllNB*, entonces se utiliza una combinación And de todos los vecinos y el pixel de trabajo, mientras si se incluye el término *AnyNB*, entonces se utiliza una combinación Or.

El próximo ejemplo muestra un objeto "Rule" denominado *ContourRule*, que define una contribución a cada pixel de la clase denominada *GoodContour*. La contribución se evalúa desde la clase *SimpleContour* en el mismo pixel y desde la clase *HighVariance* cualificada y con el término *AnyNB*:

```
Define Rule ContourRule  
  Target:      GoodContour  
  Condition:   SimpleContour And Very HighVariance (AnyNB)  
  Certainty:  80  
EndDefine
```

Un aspecto esencial de la actividad del procesador BU es el control de la incertidumbre, la cual se transforma y asigna desde las clases, clasificadores y reglas hacia otras clases. Los procesos básicos se muestran en la figura II.14. El valor intrínseco de una clase es el determinado desde las fuentes de datos, tales como clasificadores y reglas. Si la clase posee particularizaciones, entonces el citado valor intrínseco debe normalizarse para obtener el valor final. Esta normalización está basada en considerar los casos particulares como subconjuntos borrosos. A continuación se

detallan los pasos exactos que se siguen para el cómputo de una clase, en los cuales se ha rechazado explícitamente la existencia de dependencias cíclicas:

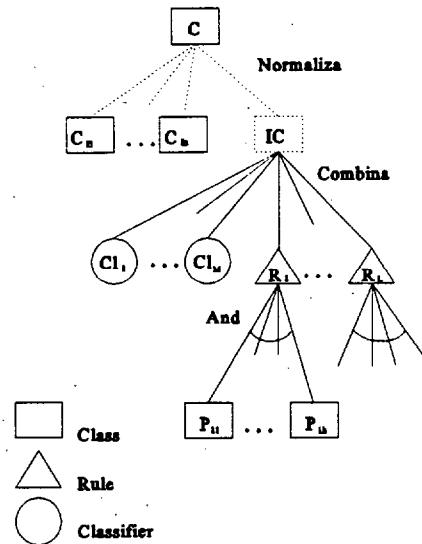


Figura II.14. Árbol de dependencias generales entre clases. La imagen IC es el valor intrínseco computado a partir de clasificadores y reglas.

Paso 1: Computar todos los clasificadores, casos particulares y clases incluidas en las condiciones de las reglas de una clase.

Paso 2: Computar todos los predicados en las condiciones de reglas. Esto se realiza mediante los siguientes sub-pasos. Sea $C_c(x,y)$ el valor de la clase de la condición, y $C_p(x,y)$ el valor de salida del predicado.

Paso 2.1: Computar el valor de entorno $C_e(x,y)$ como sigue:

$$C_e(x,y) = \begin{cases} C_c(x+\Delta(n),y+\Delta y(n)) & \text{si NB n} \\ \underset{u=0..7}{\text{Min}}[C_c(x,y), \underset{u=0..7}{\text{Min}}[C_c(x+\Delta x(u),y+\Delta y(u))] & \text{si AllNB} \\ \underset{u=0..7}{\text{Max}}[C_c(x,y), \underset{u=0..7}{\text{Max}}[C_c(x+\Delta x(u),y+\Delta y(u))] & \text{si AnyNB} \\ C_c(x,y) & \text{en caso contrario} \end{cases} \quad (\text{II-18})$$

Paso 2.2: Aplicar el cualificador lingüístico. Sea $C_q(x,y)$ su resultado:

$$C_q(x,y) = \begin{cases} \frac{C_o^2(x,y)}{100} & \text{si Very} \\ \sqrt{100C_o(x,y)} & \text{si MoreLess} \\ C_o(x,y) & \text{en caso contrario} \end{cases} \quad (\text{II-19})$$

Paso 2.3: Aplicar el operador Not:

$$C_p(x,y) = \begin{cases} 100 - C_q(x,y) & \text{si Not} \\ C_q(x,y) & \text{en caso contrario} \end{cases} \quad (\text{II-20})$$

Paso 3: Computar el resultado de la reglas, en base a los predicados y a la certeza de la misma. Si $C_{pk}(x,y)$, $k=1..h$, es la certeza de cada predicado, y C es la certeza de la regla, entonces el resultado de la regla, $C_r(x,y)$, es:

$$C_r(x,y) = \frac{C}{100_{k=1..h}} \text{Min}[C_{pk}(x,y)] \quad (\text{II-21})$$

Paso 4: Combinar los resultados de clasificadores y reglas, proporcionando el valor intrínseco. El valor intrínseco $IC(x,y)$ se evalúa desde las certezas $C_{Lk}(x,y)$, $k=1..m$, de los m clasificadores y reglas en conjunto. Esta evaluación se realiza incrementalmente en la forma siguiente:

$IC(x,y) := 0;$

for $k=1$ to m

$IC(x,y) := \text{COM}(IC(x,y), C_{Lk}(x,y));$

La función $COM()$ se define como sigue:

$$COM(u,v) = \begin{cases} u+v - \frac{uv}{100} & \text{si Logic} \\ 50[EV(u/50-1, v/50-1)+1] & \text{si Evidence} \end{cases} \quad (II-22)$$

Donde $EV(a,b)$ es:

$$EV(a,b) = \begin{cases} a+b-ab & a \geq 0 \wedge b \geq 0 \\ a+b+ab & a \leq 0 \wedge b \leq 0 \\ \frac{a+b}{1-\min(|a|, |b|)} & (a > 0 \wedge b < 0) \vee (a < 0 \wedge b > 0) \end{cases} \quad (II-23)$$

La ley Evidence utiliza el dominio $[-1,+1]$ y está basada en la Teoría de la Evidencia, en la forma en que es utilizada por algunos sistemas basados en conocimiento [Gord-84]. Para realizar la transformación entre los dos dominios, se utiliza la siguiente relación: $L=50(\lambda+1)$, donde $L \in [0, 100]$ y $\lambda \in [-1, +1]$.

Paso 5: Normalizar el valor intrínseco utilizando los valores de los casos particulares. Sea $C(x,y)$ el valor de la clase, y $C_{rk}(x,y)$, $k=1..n$, la certeza de los n casos particulares. La normalización es de la forma siguiente:

$$C(x,y) = \text{Max}[C(x,y), \text{Max}_{k=1..n}[C_{rk}(x,y)]] \quad (II-24)$$

Esta ley se basa en la consideración de que si un conjunto borroso A es un subconjunto de otro B , $A \subseteq B$, entonces los grados de pertenencia de un pixel, p , $u_A(p)$ y $u_B(p)$, deben verificar [Zade-65]: $u_A(p) \leq u_B(p)$.

II.3.2.3 SISTEMA DE CONTROL. PROCESADOR TD.

El sistema de control, tal y como se ha definido en apartados previos, está orientado a la consecución de objetivos y se puede describir como un sistema de petición y respuesta. Las peticiones incluyen solicitudes de cálculo de clases, y las respuestas suministran tales peticiones. Existen dos tipos de peticiones: las de solicitud de clases y las de inicialización o reset del procesador de píxel. Esta última provoca la ejecución del código del objeto "Channel", una vez borrados todos los datos de ciclos de operación anterior. Las clases que pueden ser solicitadas se deben incluir en un objeto denominado "**Interface**", que se declara una sola vez en la aplicación, en la forma siguiente:

```
Define Interface
  Class: pixel_class_list;
EndDefine
```

La lista de clases explicita las reconocidas o "visibles" para el nivel superior. Una petición de evaluación de una de ellas inicia el funcionamiento del procesador TD. Este analiza la red de dependencias entre clases y características, ordenando al procesador BU la ejecución de los operadores necesarios. El procesador TD actúa básicamente como un controlador del flujo de datos, por lo que no necesita detalles explícitos de funcionamiento.

II.3.3 EN EL NIVEL DE LOS SEGMENTOS.

El siguiente nivel dentro del sistema lo constituye el Procesador de Segmentos cuyo cometido básico es proporcionar la partición de la imagen en un conjunto de segmentos o agregaciones de píxels con sus respectivos grados de pertenencia a clases de segmentos ("cuadrado_rojo", "línea_recta", ...). Consideramos al **segmento** como *una agregación conexas de píxels definida por criterios de forma o de uniformidad en propiedades*. Para clarificar la terminología, denominaremos **Partición a la lista de segmentos** que constituyen *una teselación completa de la imagen y que incluye las asignaciones simbólicas de los segmentos a clases y las relaciones espaciales entre segmentos*. La utilización del segmento como grano de información

en este nivel plantea dos dificultades que no se presentan en el nivel de los pixels. De una parte, la necesidad de obtener la definición espacial de los segmentos a partir de los mapas de diagnóstico o clases de pixels obtenidos del Procesador de Pixels como paso previo a su descripción simbólica en términos de clases de segmentos. De otra, mientras el Procesador de Pixels maneja entidades predefinidas, de límites espaciales estacionarios, en el Procesador de Segmentos es muy interesante permitir el que las entidades puedan sufrir cambios durante el proceso de refinamiento de la segmentación como consecuencia de la fusión o división de segmentos.

La arquitectura sobre la que se ha desarrollado el Procesador de Segmentos es conceptualmente idéntica a la ya descrita para el Procesador de Pixels, basada en las unidades BU y TD. Los requerimientos anteriores, no obstante, introducen en la funcionalidad de los procesadores particularidades propias de este nivel. La distribución de tareas entre los procesadores BU y TD es la siguiente:

1. El procesador BU recibe como datos de entrada mapas de diagnósticos, clases de pixels, que previamente han sido solicitados por el procesador TD al Procesador de Pixels; y produce como datos de salida, diagnósticos simbólicos asociados a los segmentos que constituyen la Partición. Su acción tiene dos etapas diferenciadas: una, la obtención del grano de información, esto es, la definición de la Partición inicial; y otra, la descripción simbólica de los segmentos así obtenidos en términos de clases de segmentos.
2. El procesador TD recibe órdenes en forma de peticiones de diagnósticos sobre segmentos, y transforma las mismas en secuencias de órdenes de ejecución de tareas para el procesador BU, así como en órdenes de solicitud de diagnósticos dirigidas al Procesador de Pixels. El procesador TD puede ser programado para modificar la Partición de la imagen ejecutando acciones de control disparadas por reglas. Las premisas de las reglas se establecen en términos de los diagnósticos sobre clases de segmentos producidos por el procesador BU y pueden incluir condiciones sobre relaciones espaciales entre segmentos. Las acciones de control de modificación de la Partición constituyen los "actos reflejos" habilitados en el nivel de los segmentos.

II.3.3.1 ARQUITECTURA DEL PROCESADOR DE SEGMENTOS. LENGUAJE DE DEFINICIÓN.

Desde el punto de vista funcional, la arquitectura del Procesador de Segmentos (figura II.15) se organiza en tres grandes bloques con cometidos específicos. El primero está dedicado a la definición de la Partición inicial y es el cometido del módulo de presegmentación integrado en el procesador BU. Este módulo está concebido como una parte intercambiable dentro del Procesador de Segmentos pues en principio son posibles diferentes aproximaciones a la hora de

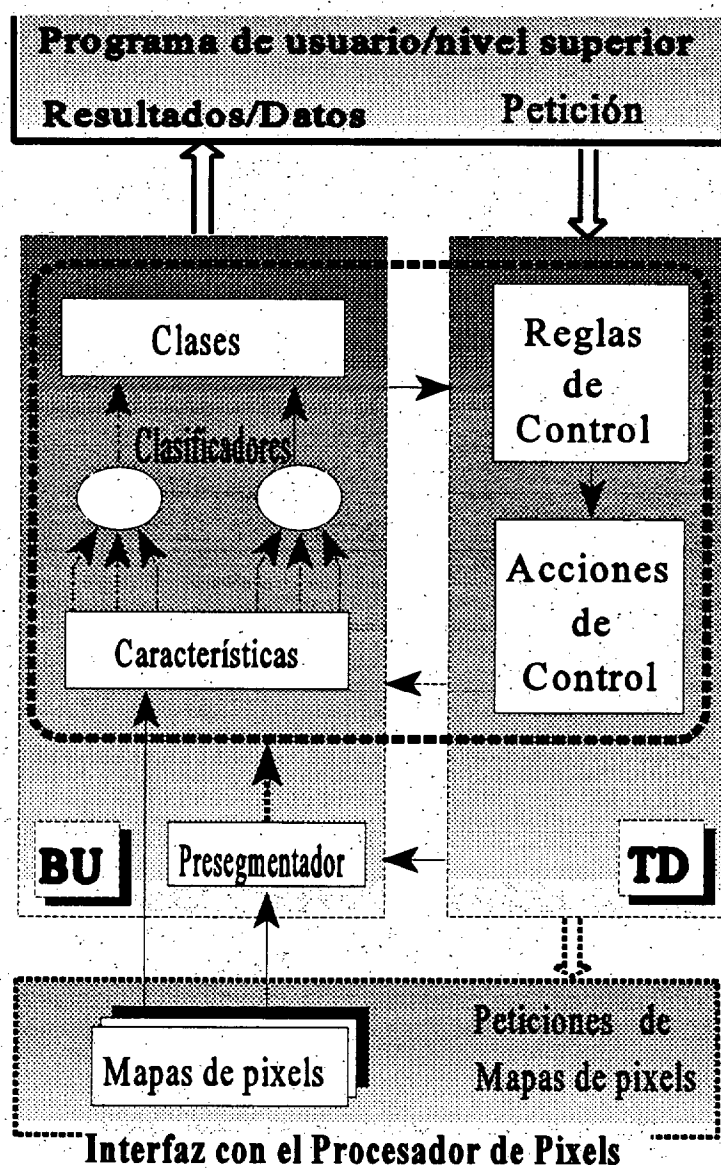


Figura II.15. Estructura de procesadores y dominios en el nivel de los segmentos.

establecer una Partición inicial o segmentación primaria de la imagen. El segundo tiene como cometido la definición de la estructura computacional de obtención de los diagnósticos de segmentos. Esta basado en la distinción de un dominio numérico o de características (Features) y otro simbólico o de clases (Class), y es idéntico al empleado en el Procesador de Pixels, salvo matices como la imposición de condiciones sobre relaciones espaciales entre segmentos. El tercer bloque funcional se ubica en el procesador TD y gestiona la atención de peticiones de diagnósticos y el control del Procesador de Segmentos. Éste incluye la evaluación del estado de la Partición desde un conjunto de reglas de control que pueden producir acciones que modifiquen la definición espacial de los segmentos.

La programación del Procesador de Segmentos es similar a la del Procesador de Pixels, introduciéndose para ello un nuevo conjunto de objetos que reflejan las características funcionales comentadas anteriormente. Así, existe un conjunto de objetos que define la actuación del

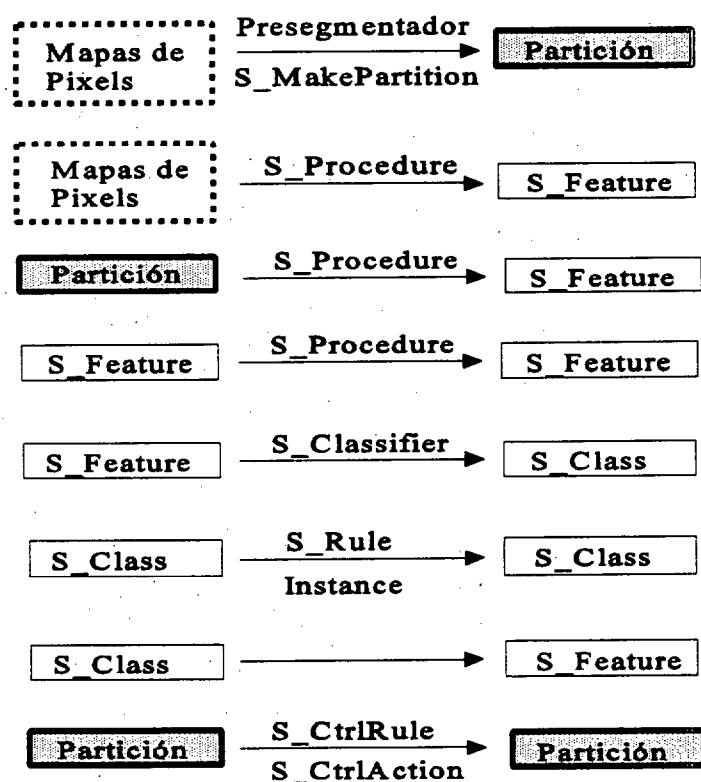


Figura II.16. Tipos de datos y su transformación mediante operadores.

presegmentador y cuya tipología dependerá del presegmentador empleado. Independientemente del presegmentador que se seleccione, el objeto "S_MakePartition" define desde el procesador TD el control de la acción del presegmentador. Los objetos "S_Feature", "S_Procedure", "S_Classifier", "S_Class" y "S_Rule" son equivalentes a sus homónimos en el Procesador de Pixels, con la salvedad de que se aplican en la descripción de segmentos en vez de pixels. Las diferencias más destacables son la introducción del objeto "S_Condition" para expresar premisas y la obtención de características ("S_Features") desde diferentes tipos de datos. Así, una vez definida la Partición, las características de un segmento pueden ser obtenidas desde los diagnósticos de los pixels que lo integran (Mapas de pixels), desde otras características (S_Features), desde una clase (S_Class) o desde la propia definición del segmento (Partición) cuando la característica está asociada a la definición espacial del segmento (v.g. forma, tamaño, etc...). Relacionados con el procesador TD, los objetos "S_CtrlRuleSet", "S_CtrlRule" y "S_CtrlAction", posibilitan la definición de acciones de control sobre la partición. Finalmente, el objeto "S_Interface" establece los diagnósticos que son visibles al nivel superior, y el objeto "P_Interface" indica las clases de pixels que es necesario solicitar al Procesador de Pixels. La figura II.16 muestra los diferentes tipos de datos (en cuadros) y su transformación mediante los operadores dispuestos sobre las flechas. Los tipos de datos encuadrados con trazo continuo y fondo blanco representan descriptores, de naturaleza numérica o simbólica, de cada segmento de la partición. Los datos enmarcados en trazo discontinuo son mapas de diagnósticos obtenidos del Procesador de Pixels, que se utilizan para definir la partición inicial o para computar propiedades de los segmentos a partir de diagnósticos asociados a los pixels que integran cada segmento. La partición o teselación de la imagen puede ser considerada como otro tipo de dato, actualizable mediante el disparo de reglas de control y las acciones de modificación asociadas a éstas.

II.3.3.2 DESCRIPCIÓN DE OBJETOS. PROCESADOR BU.

La unidad BU del Procesador de Segmentos tiene dos cometidos fundamentales: la definición del nuevo conjunto de entidades de este nivel a partir de los símbolos definidos para el nivel anterior y la atribución simbólica de las entidades así definidas. Los objetos del lenguaje involucrados en la programación del procesador BU se organizan según esta distribución de

cometidos. Por este motivo, consideraremos en primer lugar la problemática ligada a la obtención de la Partición inicial y a continuación describiremos los objetos que participan en la descripción simbólica de los segmentos.

II.3.3.2.1 DEL PIXEL AL SEGMENTO.

La organización de un SPE por niveles, donde cada nivel establece un conjunto propio de símbolos y operadores, es un ingrediente fundamental de la metodología presentada al principio de este capítulo para el diseño de sistemas percepto-efectores. Esta aproximación a la organización de tales sistemas plantea, sin embargo, la necesidad de definir transformaciones de símbolos y entidades entre niveles para alcanzar una mayor elaboración semántica y compactación de la descripción a medida que se progresa de un nivel al siguiente. Aunque no existe una regla general que indique cómo se ha de plantear la transformación entre dos niveles dados, sí es posible imponer una serie de requerimientos más o menos generales al conjunto de posibles transformaciones. De acuerdo con la receta 5 presentada con esta metodología de diseño (epígrafe II.2.4), la transformación debe establecer de manera explícita la definición de las entidades de un nivel a partir de la descripción simbólica de las entidades del nivel anterior.

La aplicación de estas consideraciones al pasar del nivel de los pixels al de los segmentos plantea el problema de la definición de las entidades elementales o segmentos, esto es el grano de información del nuevo nivel, a partir de los mapas de diagnóstico o símbolos producidos en el nivel de pixels. Este es el cometido básico del módulo de presegmentación incluido en el procesador BU. Desde el punto de vista operativo, el problema a resolver es el de segmentar una imagen mediante algún método donde sea posible combinar información obtenida de diferentes mapas de pixels.

En un sistema basado en conocimiento, una aproximación natural a la definición de los segmentos en términos de diagnósticos de pixels es la definición de "prototipos" o "zonas bien definidas", también llamadas en algunos sistemas [Hans-88] "islas de fiabilidad". La definición de estas zonas puede realizarse básicamente de dos maneras. En la primera, que podría denominarse

como definición explícita, la especificación de las zonas prototipo se establece en términos de los grados de pertenencia de los pixels a ciertas clases. La definición de estas zonas responde a los objetivos de la segmentación y es parte fundamental del conocimiento depositado en la base de modelos del nivel de segmentos. Con frecuencia en su definición se deben tomar en consideración no sólo los valores de diagnóstico de los pixels, sino también criterios de extensión y grado de uniformidad. La clasificación de los pixels y su agrupamiento para definir segmentos es un problema de naturaleza esencialmente difusa, de manera que esta aproximación tiene asociada una situación característica que gráficamente puede ser visualizada como una imagen donde se presentan "zonas prototípicas o bien definidas" sumidas en un halo más o menos extenso de indefinición. El problema importante que implica esta aproximación es el de la adscripción de los pixels situados en las zonas de indefinición a alguno de los prototipos vecinos. Este problema admite diferentes soluciones que pueden basarse en un esquema de crecimiento de regiones [Vinc-91] o en técnicas de agrupamiento difuso [Lim-90]. La otra posibilidad, que podemos llamar de definición por extensión, se basaría en detectar agrupamientos en el espacio de propiedades multidimensional generado por los mapas de pixels que se utilicen. Los criterios de definición que intervienen son entonces de extensión y uniformidad suficientes. Dentro de este marco es posible utilizar diversos métodos de segmentación consistentes con la utilización de múltiples mapas de características, como las basadas en el análisis de histogramas locales propuestas en [Ohla-76] y [Ross-89]. El mayor inconveniente de estas técnicas reside en que suelen producir una sobresegmentación de la imagen y una pobre delineación de los objetos. La propuesta presentada en [Lim-90], que describimos en el primer capítulo, podría ser un ejemplo de método híbrido, donde las zonas "prototipo" se definen por la situación de los máximos en los histogramas de los mapas de características, pero los pixels situados en los valles de los histogramas, en las zonas de peor definición, se adscriben al "prototipo" más próximo utilizando el algoritmo K-medias difuso definido en el espacio de propiedades.

Sólo en algunas casos, la definición de los segmentos no plantea dificultades pues la segmentación puede resolverse al nivel de los pixels y el papel que debe jugar el presegmentador o no es necesario o es trivial. Tal es el caso en problemas donde la partición de la imagen se puede realizar a partir de la clasificación de los pixels de manera exclusiva en un cierto conjunto de

Capítulo II

clases [Blanz-89][Fich-92][Hild-92]. El resultado es un mapa o conjunto de mapas donde cada pixel está asignado a una cierta clase, de tal forma que los segmentos quedan definidos por los agrupamientos conexos de pixels que comparten una misma etiqueta. En estas situaciones la función reservada al Procesador de Segmentos es la de la caracterización de los segmentos resultantes por criterios de forma o relacionales, y el posible refinamiento de la segmentación resultante, por ejemplo, mediante la eliminación/fusión de segmentos que se juzguen irrelevantes. Es necesario resaltar, que la acción del Procesador de Pixels en la descripción simbólica de los pixels puede ser rica y compleja, habida cuenta de los recursos presentes en dicho procesador. La pregunta que surge naturalmente de lo anterior es: ¿cuándo es necesaria entonces la acción del presegmentador?; o planteada en otros términos, ¿cuándo no es posible reducir la segmentación de una imagen a la clasificación de sus pixels más una posible etapa posterior de fusión?. La respuesta a esta última pregunta, desde un punto de vista estrictamente posibilista, sería "rara vez". Ahora bien, si la pregunta admite la matización de dónde o cómo es más viable o eficiente producir un resultado equivalente, la respuesta cambia radicalmente. Fundamentalmente porque para evitar errores en la segmentación, este planteamiento debe producir una gran sobresegmentación de la imagen en la primera etapa de clasificación de los pixels para luego proceder a fusionar los segmentos irrelevantes con sus vecinos lo que normalmente supone un gran coste computacional.

En definitiva la cuestión clave es cómo hacer intervenir las zonas mejor definidas en la imagen, en la segmentación de las zonas donde la indefinición es mayor. A nuestro juicio, la solución más natural dentro de un sistema basado en conocimiento es la que hemos presentado como de definición explícita. Por otra parte, la característica más interesante de los métodos de definición por extensión es su adaptabilidad; y la más desfavorable, el escaso control que se tiene sobre de la fragmentación en las zonas de transición donde la indefinición es mayor. Si la fragmentación es excesiva, el Procesador de Segmentos deberá invertir un esfuerzo considerable en eliminar los pequeños segmentos. Ambas aproximaciones deben resolver en esencia el mismo problema: el etiquetado de los pixels situados en las zonas de mayor ambigüedad. Sin embargo en la primera, la adscripción de estos pixels se realiza con anterioridad a la definición de los segmentos, evitándose el cómputo necesario para describir los pixels, o

pequeños grupos de éstos, como segmentos por lo que se alcanza un resultado equivalente de manera mucho más eficiente.

El análisis previo ha conducido en el transcurso de este trabajo de investigación a la selección de la aproximación por definición explícita para la elaboración de un módulo de presegmentación inspirado en la transformada Watershed [Vinc-91] denominado Flood. Esta opción no excluye la adaptación de otros métodos y el desarrollo de otros módulos en el futuro. En consecuencia, el presegmentador ha sido concebido como una parte intercambiable del Procesador de Segmentos donde se pueden incluir diferentes estrategias de segmentación en función de la aplicación. Se trata de aprovechar la ventaja de métodos bien conocidos en segmentación de imágenes [Pav77], [Har85], [Ros82], incorporados de una forma flexible, que permitan al usuario el diseño, desde un conjunto de opciones, de aquella que mejor se adapte a su aplicación. Para lograr esta flexibilidad, la elección de un cierto módulo de presegmentación se refleja en la introducción de objetos específicos en el lenguaje de programación del Procesador de Segmentos. Retrasaremos la descripción del presegmentador desarrollado en este trabajo y de los objetos necesarios para su programación al capítulo tres, pues como opción específica no constituye una parte fundamental dentro de la unidad BU del Procesador de Segmentos.

La segmentación inicial producida en el presegmentador se trata como un mapa de colores y se transforma posteriormente en una Partición inicial mediante un análisis de componentes conectadas, que extrae del mapa de color información espacial exhaustiva que utiliza durante la descripción simbólica de los segmentos. Dicha información detalla los bordes interiores y exterior de cada componente conectada o segmento y la lista ordenada de los segmentos vecinos según se encuentran al recorrer el borde. Incluye también las coordenadas de los pixels que integran el segmento y la ventana rectangular mínima que lo incluye. El análisis de componentes conexas es una pieza fundamental en la concepción y funcionamiento del Procesador de Segmentos (unidad BU) que se invoca cada vez que la Partición sufre una modificación. El algoritmo utilizado se describirá en el Capítulo III.

II.3.3.2.2 DESCRIPCIÓN SIMBÓLICA DE LOS SEGMENTOS.

La estructura de clases de segmentos y su evaluación se define siguiendo un camino análogo al establecido para el Procesador de Pixels, así estarán presentes objetos tales como "S_Procedure", "S_Feature", "S_Classifier", "S_Rule" o "S_Class". Cuya composición y significado guardan paralelismo con sus homónimos respectivos, aunque aplicados a segmentos o zonas uniformemente conexas de la imagen. El único matiz diferencial importante reside en la introducción del objeto "S_Condition" que posibilita la declaración independiente de premisas. Esto no supone ninguna modificación en el esquema de combinación de las premisas y es tan sólo una facilidad de programación. Ya que de esta manera una premisa declarada como una "S_Condition" puede ser utilizada en múltiples reglas. Una condición en el marco del Procesador de Segmentos (S_Condition) se entiende como el encadenamiento de un conjunto de premisas conectadas por conjunciones lógicas tipo "And". El siguiente ejemplo ilustra una definición sencilla de este tipo:

```
Define S_Condition   RedSquare
      Premise:       Is( Red , x) And Is( Square , x)
EndDefine
```

Las condiciones así definidas se utilizan en el Procesador de Segmentos en los diversos tipos de reglas, tanto aquellas que definen la estructura de clases ("S_Rule"), como en reglas de control ("S_CtrlRule"), o en acciones de control ("S_CtrlAction").

Las premisas utilizadas en este nivel se enriquecerán con la inclusión de relaciones espaciales, las cuales cobran un significado singular en el Procesador de Segmentos y, en cierta medida, condicionan su diseño. La definición sintáctica de la premisa de una condición se establece simbólicamente como sigue:

```

premise ::= prem_atom_true | prem_list
premise_atom_true ::= Is ( True, [qualifier1] x )
premise_list ::= prem_atom { And prem_atom }
premise_atom ::= Is( property { And property }, [qualifier1] x )
property ::= [qualifier2] [order] sclassname
qualifier1 ::= [neig][qualifier2][spatial]
neig ::= All | Any
qualifier2 ::= [Not][linguistic]
linguistic ::= Very | MoreLess
spatial ::= Above | Below | Left | Right | Beside | InBeside | Neib | ExNeib |
            InNeib | Contains | user_defined_localizer
order ::= Gt (number) | Eq (number) | Lt (number)
    
```

Los cualificadores *All* y *Any* hacen referencia al entorno y operan en conjunción con los cualificadores de relaciones espaciales que analizaremos posteriormente. Se incluyen además relaciones de orden (*Gt*, *Eq*, *Lt*) que afectan al grado de la propiedad sobre la cual se establece el diagnóstico. Se permite asimismo la inclusión de lista de propiedades And-conectadas. La sintaxis de escritura de una premisa de esta naturaleza se fundamenta en la expresión elemental:

Is(*property* , *x*)

Donde *x* denota el segmento foco de atención (cuya referencia puede omitirse, pero que mantendremos por claridad en la escritura) y *property* es el diagnóstico sobre el que se condiciona a *x*. Así la premisa:

Is(*Green* , *x*)

Sería la expresión formal del hecho "El segmento foco de atención es verde", y la evaluación de esta premisa devolvería el grado en que el segmento foco de atención satisface el diagnóstico asociado a la clase "Green". El sentido de una premisa puede modificarse mediante el uso de los modificadores *Not*, *Very* y *MoreLess*, ya presentados en el Procesador de Pixels. Además la propiedad sobre la cual se establece la premisa puede ser compuesta y contener un conjunto de propiedades elementales o diagnósticos conectados por conjunciones lógicas tipo "And".

Is(*Very Green* And Not *Red* , **x)**

Es fundamental en el marco de este procesador establecer premisas sobre la vecindad del segmento foco de atención, para obtener una descripción del papel particular que juegan las relaciones espaciales. La sintaxis básica para incorporar relaciones espaciales es:

Is(*property* , *spatial* **x)**

Donde *spatial* representa a un elemento del lenguaje que se asocia con una relación espacial de los tipos: *Above*, *Below*, *Left*, *Right*, *Beside*, *Neib*, *InBeside*, *ExNeib*, *InNeib* y *Contains*. Así la expresión :

Is(*Green*, *Below* **x)**

Se correspondería con la condición "Algún segmento de los que están debajo del foco de atención es verde". Por otra parte, una premisa expresada como

Is(**True , *Contains* **x**)**

actúa exclusivamente como un selector del segmento en cuyo interior se situa el segmento foco de atención. Una premisa de este tipo sólo está permitida cuando se utiliza en una acción de control ("S_CtrlAction") para seleccionar algún vecino del segmento foco de atención. Algunos localizadores tienen acotado su ámbito de aplicación. Así, localizadores como *Above*, *Below*, *Left*, *Right*, *Beside*, *ExNeib* y *Contains* se evalúan sólo sobre los vecinos "externos" de un segmento dado. Por el contrario, los localizadores *InBeside* e *InNeib* motivan la evaluación exclusivamente sobre los segmentos contenidos dentro de uno dado, es decir, sobre sus vecinos "interiores". Finalmente, *Neib* hace que se evalúe la condición sobre cualquier vecino del segmento foco de atención. También resulta útil establecer la condición, no sobre algún vecino (opción de defecto) sino sobre todos los adyacentes, lo que se hará utilizando el modificador *All*. Así, si deseamos expresar condiciones como: "Todos los segmentos contenidos en el segmento foco de atención

no son rojos" o " Todos los segmentos que no están debajo del foco de atención son muy cuadrados", tendríamos respectivamente las premisas:

Is(Not Red , All InNeib x)

Is(Very Square, All Not Bellow x)

La evaluación del predicado "A Above B" devuelve la medida, en el intervalo [0,100], en que el segmento A, vecino del B, se sitúa "por encima" de este. Consideremos las siguientes magnitudes:

- L_{AB} : Medida de la longitud del contorno común entre los segmentos A y B.
- $C_y(A)$: Ordenada del centroide del segmento A.
- $C_y(B)$: Ordenada del centroide del segmento B.
- $L_{AB}[y < C_y(B)]$: Medida de la longitud del contorno común entre A y B tal que se sitúa por encima del centroide de B. Nótese que la ordenada crece hacia la parte inferior de la imagen.
- $L_{AB}[y < C_y(A)]$: Medida de la longitud del contorno común entre A y B tal que se sitúa por encima del centroide de A.

Definiremos numéricamente la relación como:

$$R_{A \text{ Above } B} = 50 \left(1 + \frac{L_{AB}[y < C_y(B)] - L_{AB}[y < C_y(A)]}{L_{AB}} \right) \quad (\text{II-25})$$

Por extensión de esta definición establecemos en el rango [0,100] las correspondientes a:

Below:

$$R_{A \text{ Below } B} = 50 \left(1 + \frac{L_{AB}[y > C_y(B)] - L_{AB}[y > C_y(A)]}{L_{AB}} \right) \quad (\text{II-26})$$

Left:

$$R_{A \text{ 'Left' } B} = 50 \left(1 + \frac{L_{AB}[x < C_x(B)] - L_{AB}[x < C_x(A)]}{L_{AB}} \right) \quad (\text{II-27})$$

Right:

$$R_{A \text{ 'Righth' } B} = 50 \left(1 + \frac{L_{AB}[x > C_x(B)] - L_{AB}[x > C_x(A)]}{L_{AB}} \right) \quad (\text{II-28})$$

El predicado de naturaleza espacial "A Beside B" es indicativo del grado en que un segmento A es vecino de otro B. La existencia de esta relación se justifica por la necesidad de condicionar ciertas acciones sobre el vecino más importante. El grado de vecindad se determina de la proporción entre la longitud del contorno común entre A y B, L_{AB} , y la longitud total del contorno del segmento vecino B que no es vecino del marco de la imagen, L_B , en el intervalo $[0,100]$.

$$R_{A \text{ Beside } B} = 100 \frac{L_{AB}}{L_B} \quad (\text{II-29})$$

Para los casos en que A es un segmento interno del segmento B, la relación espacial incluida en el predicado "A InBeside B" se evalúa de forma análoga, pero considerando que L_B representa al contorno interno del segmento B que engloba al segmento A además de otros posibles segmentos internos.

Los elementos *Neib*, *ExNeib* e *InNeib*, se utilizan para señalar a cualquier vecino del foco de atención, independientemente de su posición o importancia relativa, e indican, respectivamente, si un segmento es vecino de otro, si es un vecino exterior o si es un vecino interior, devolviendo un único valor (100) en caso de evaluación positiva. El término *Contains* selecciona el segmento que contiene al segmento foco de atención si existe. Es decir, la evaluación del predicado de naturaleza espacial "A Contains B" devolverá 100 si B es un segmento interior al segmento A y 0 en cualquier otro caso.

El diseñador puede también redefinir la forma en que se computan estos localizadores o definir otros nuevos. Para ello debe incluir un objeto del tipo "User_Localizer" con el nombre del nuevo localizador. Con este objeto se indica al compilador la incorporación del nuevo localizador y su ámbito de evaluación que puede abarcar los vecinos externos, los internos o ambos (atributo *Scope*). Por otro lado, el código que implementa el localizador debe ser compilado y enlazado con el resto de la aplicación. Por ejemplo, la siguiente definición sería necesaria para incorporar un nuevo localizador denominado *My_Left*:

```
Define User_Localizer My_Left
    Scope: Outers
EndDefine
```

La utilización de predicados que incluyen relaciones espaciales entre segmentos permiten establecer también relaciones de orden, esto es comparaciones entre vecinos para una propiedad o diagnóstico dado. Las relaciones de orden permitidas son: *Gt* (Mayor), *Eq* (Igual) y *Lt* (Menor), cuya evaluación, establecida de manera difusa, trataremos a continuación.

Para ello, considérense dos segmentos A y B y sea SC un determinado diagnóstico en base al cual se desea establecer una relación de orden del tipo *GT* entre A y B, que designaremos por la expresión ' $SC_A GT SC_B$ ', siendo SC_A y SC_B los valores que toma el diagnóstico sobre A y B respectivamente. Sea u una variable definida como la diferencia entre estos dos valores y sea δ un parámetro indicativo de la semianchura del intervalo en u que separa un resultado verdadero (100) de otro falso (0). Entonces el juicio sobre " $SC_A GT(\delta) SC_B$ " se resuelve de acuerdo con:

$$R_{SC_A GT(\delta) SC_B} = 100 S(u; -\delta, \delta) \quad (II-30)$$

Donde $S(u; \alpha, \beta)$ es la función definida en II.5 para el intervalo $[0,1]$. El parámetro δ controla la incertidumbre en el juicio y es dependiente del diagnóstico utilizado para establecer la relación de orden.

Análogamente se define la evaluación de la relación del tipo *Eq* entre A y B como:

$$R_{SC_A Eq(\delta) SC_B} = 100 \Pi(u; -\delta, \delta) \quad (II-31)$$

Siendo la función $\Pi(u; \alpha, \beta)$ la definida en II.8. Por último la relación del tipo *Lt* se establece como:

$$R_{SC_A Lt(\delta) SC_B} = - 100 S(u; -\delta, \delta) \quad (II-32)$$

De esta forma será posible expresar hechos del tipo "Algún vecino del foco de atención es más negro con incertidumbre 10", como:

$Is(Gt(10) Black, Neib x)$

donde *Black* es una *S_Class* o diagnóstico representativo de la clase segmentos negros.

La evaluación de las premisas se realiza según los pasos indicados a continuación. Este esquema resume el proceso de evaluación de una premisa, la correspondiente evaluación de una *S_Condition* se llevará a cabo realizando una operación que devuelva el mínimo de los resultados obtenidos para cada una de las premisas que la integran.

Paso 1: Localización.

- Paso 1.1:** Si no existe ningún localizador espacial selecciona el foco de atención y salta al paso 2.
- Paso 1.2:** Se evalúa el localizador espacial sobre la vecindad del segmento foco de atención. La vecindad puede estar constituida por los vecinos internos, externos o ambos dependiendo del localizador empleado.
- Paso 1.3:** Se ejecutan, si existen los modificadores *Very* y *MoreLess* sobre los resultados obtenidos en 1.2.

Paso 1.4: Si es necesario se niegan (*Not*) y se construye una lista con los segmentos vecinos que, tras el proceso de localización, se evalúen positivamente.

Paso 2: Evaluación.

Paso 2.1: Sobre cada uno de los segmentos seleccionados en la etapa anterior, se evalúa la lista de propiedades (premisas elementales) en la forma ya explicada para el Procesador de Pixels, salvo el matiz de la inclusión de los predicados de orden. La evaluación de estos predicados sobre un determinado diagnóstico produce un resultado que es tratado internamente como un nuevo diagnóstico.

El resultado de esta etapa es la asignación, a cada uno de los segmentos seleccionados del mínimo de los diagnósticos que conforman la lista de propiedades.

Paso 3: Selección.

Paso 3.1: Si no existe en la premisa un localizador espacial, se presenta como resultado el producido en el paso 2 sobre el segmento foco de atención.

Paso 3.2: Si existe localizador espacial, la evaluación de la premisa será el resultado de realizar una de las siguientes operaciones:

- Si el modificador utilizado con el localizador espacial es *Any*, tomar el máximo de los diagnósticos obtenidos en el segundo paso
- En otro caso (*All*) tomar el mínimo de los diagnósticos que se construyeron en el paso 2.

Análogamente al Procesador de Pixels, el control de la certidumbre es un aspecto central en la actividad del procesador BU. La combinación de evidencias para definir una clase de segmentos (*S_Class*) desde clasificadores, reglas y particularizaciones utiliza un esquema idéntico al descrito para el Procesador de Pixels.

II.3.3.3 SISTEMA DE CONTROL. PROCESADOR TD.

El procesador TD tiene en el nivel de los segmentos un papel mucho más activo que en el nivel de los pixels. En este último, el flujo de datos y de ejecución de procesos procedía de abajo a arriba quedando claramente definida la evaluación de estructuras de datos tipo imagen de las clases de pixels. El Procesador de Segmentos opera según dos fases diferenciadas. Una directa, ascendente, ejecutada por la unidad BU, de características similares al modo de trabajo del Procesador de Pixels en la que se calculan los grados de pertenencia a clases de segmentos de aquellos que están presentes en la partición inicial. La otra queda bajo la supervisión del procesador TD y permite desarrollar estrategias que modifican la partición de una imagen desde un conjunto de reglas de control. Estas reglas (*S_CtrlRules*), son diferentes de las utilizadas en la generación de clases de pixels o de segmentos (*S_Rule*), y se estructuran a partir de una condición o lista de condiciones (*S_Condition*), una acción de control (*S_CtrlAction*) a ejecutar si se dispara la regla y, si se requiere, una condición adicional que actúa como selector de algún segmento vecino. Las reglas de control se agrupan en conjuntos (*S_CtrlRuleSet*) de intersección no necesariamente vacía, posibilitando la modularización del conocimiento. Las acciones de control se ejecutan cuando se verifican las condiciones correspondientes a la regla. Estas acciones son:

- Unión (*Merge*) del segmento que constituye el foco de atención con el segmento vecino que verifica en mayor grado cierta condición utilizada como selector.
- Inclusión (*Include*) del segmento foco de atención en el segmento vecino que verifica en mayor grado una cierta condición. Este tipo de acción está ideada para tratar regiones con brillos, sombras o en general que sufren modificaciones anómalas de ciertas

propiedades, de manera que es posible evitar que determinadas propiedades de segmentos se desvirtúen.

- División (*Split*) de un segmento en los dos segmentos a partir del cual se formó. Es decir, la división de un segmento sólo puede deshacer uniones previas, hasta alcanzar el nivel definido por la Partición inicial.

- Bloqueo (*Lock*) de una región de manera que no pueda sufrir ningún cambio.

- Desbloqueo (*Unlock*).

El control del Procesador de Segmentos se establece desde la interface con el Procesador Relacional, donde la unidad TD atiende la demanda de una serie de diagnósticos sobre segmentos. El análisis de esta demanda provoca las correspondientes peticiones al Procesador de Pixels de los diagnósticos de pixels necesarios. A continuación, la unidad TD utiliza entonces la información contenida en el objeto "S_MakePartition" para activar la acción del presegmentador y obtener una primera Partición. Obtenida ésta, la unidad TD ordena a la unidad BU la evaluación de las características y los grados de pertenencia a las clases solicitadas de los segmentos presentes en la Partición. Posteriormente, ésta es analizada por la unidad TD por medio de las reglas de control lo cual puede provocar la modificación de la misma. En los procesos de agregación/desagregación que se producen por modificaciones de la Partición, la unidad TD indica a la unidad BU la lista de segmentos cuya descripción simbólica es necesario actualizar. La evaluación de condiciones que incorporen relaciones espaciales provoca que determinados segmentos puedan sufrir variaciones en sus propiedades debido a la presencia de relaciones espaciales. De aquí que dicha lista incluya, no sólo los segmentos directamente implicados en la acción de control, sino también a aquellos cuyos vecinos, o ellos mismos, han sufrido alguna variación en su composición. Tras restaurar la información topológica de los segmentos implicados, la unidad BU actualiza los diagnósticos para completar la descripción de la nueva Partición y el ciclo de control se repite. La unidad TD detiene la ejecución de este ciclo cuando se presenta alguno de los siguientes eventos:

Capítulo II

- a) Si no existen reglas de control definidas, cuando se evalúan todas las clases solicitadas para la partición inicial.
- b) Cuando no exista el disparo de ninguna regla de control.

Como se ha indicado, el control del presegmentador se realiza a través de la información contenida en el objeto "S_MakePartition". En él se indican en primera instancia las clases de pixels que son utilizadas por el presegmentador, el módulo de presegmentación seleccionado, el programa a ejecutar por el presegmentador y, si fuera necesario, un posible conjunto de parámetros numéricos para el presegmentador. La segunda parte del objeto "S_MakePartition" determina las definiciones de conectividad y vecindad a emplear durante el análisis de componentes conexas que extrae la información topológica de la Partición y delimita espacialmente los segmentos. Éstos pueden definirse como 4 u 8 conectados y, de manera independiente, puede decidirse si las relaciones de vecindad deben extenderse a segmentos adyacentes diagonalmente u 8-vecinos o solamente pueden ser considerados como vecinos los 4 adyacentes. Finalmente, puede indicarse si se desea visualizar la Partición inicial resultante y con que factor de ampliación. Un ejemplo de declaración sería:

```
Define S_MakePartition
  P_Class:          Blue, Brown, GreenHighTexture;
  Presegmenter:    Flood
  ProgramName:     LookingForTrees
  Parameter:       2, 1;
  Connectivity:    8-Connectivity
                  DiagonalPrecedence: Leaves;
  Neighborhood:    4-Connectivity
  Show
                  Zoom: 2
EndDefine
```

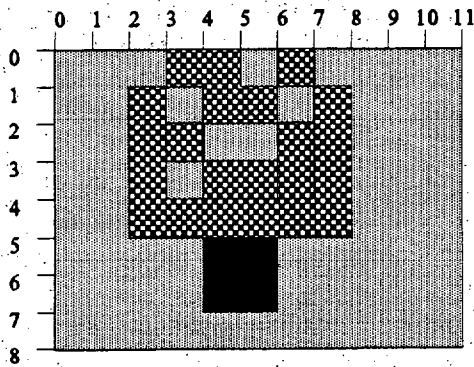


Figura II.17.

En esta definición, la acción del módulo presegmentador denominado "Flood" ejecuta el programa *LookingForTrees*. La obtención de la Partición por este programa utiliza los tres mapas de pixels indicados: *Blue*, *Brown* y *GreenHighTexture*. Los segmentos resultantes podrán ser 8-conectados y se consideraran vecinos sólo los 4-adyacentes.

La consideración de 8-conectividad puede propiciar que aparezcan situaciones de ambigüedad en la definición espacial de los segmentos durante el análisis de componentes conexas. Considérese como ilustración la situación mostrada en el mapa de colores de la figura II.17. En puntos como (1,3), (1,7), etc... no es trivial decidir si los pixels con la textura de cuadros forman parte de un mismo segmento, o, por el contrario, son los pixels de color gris los que deben conectarse en diagonal. El objeto "S_MakePartition" ofrece la posibilidad de especificar una cierta precedencia a la hora de establecer conexiones diagonales por medio de una lista (*DiagonalPrecedence*) en la que se detallan en orden decreciente los prototipos de segmentos de mayor precedencia. En este caso, las zonas etiquetadas como "Leaves" en el presegmentador, que podrían corresponderse con los pixels de textura cuadricular de la figura, se conectarán diagonalmente cuando se presente una situación de ambigüedad con zonas de cualquier otro tipo. En el capítulo tres, al estudiar la problemática ligada al análisis de componentes conectadas se estudiarán éstas y otras cuestiones relacionadas con más detalle.

El objeto "S_CtrlRule" permite definir una regla de control y establecer así las condiciones que controlan el desencadenamiento de una cierta acción de control para modificar la Partición. Su declaración deberá incluir los siguientes elementos:

- a) La lista de condiciones (*S_Condition*) que se necesitan para desencadenar una determinada acción de control.
- b) El nombre de una acción de control.

- c) Un umbral de disparo de la regla,
- d) Un modo de evaluación de la regla.

El modo de evaluación puede ser incondicional (Unconditional) o condicional (Conditional). Cuando el modo de evaluación es incondicional, el disparo de la regla y la ejecución de la acción de control asociada no está sometida a ningún tipo de condición a posteriori. Por el contrario, si se selecciona el modo condicional, la acción de control puede deshacerse para restituir la Partición anterior si el segmento o segmentos involucrados no han mejorado la evaluación de un conjunto de diagnósticos (S_Classes) combinados como objetivo (Objective). Las reglas de control cuya evaluación es incondicional o condicional son equiparables, respectivamente, a las reglas R y B descritas en la propuesta de metodología para el diseño de SPE descrita en este trabajo (epígrafe II.2.3). Realmente, las reglas de evaluación condicional son una implementación bastante elemental de las reglas B cuyo potencial puede ser muy superior.

Ejemplos de la sintaxis propuesta para este tipo de objeto serían:

```
Define S_CtrlRule NeibCirclePieces
  Condition: CircularBorderCond And CircularBorderNeibCond
  Action: MergeCirclePieces
  Evaluation: Conditional
  Objective: Not Square And Circle
  Threshold: 50
EndDefine
```

```
Define S_CtrlRule NotGreenRule
  Condition: Not GreenCond
  Action: SplitNotGreen
  Evaluation: Unconditional
  Threshold: 10
EndDefine
```

Donde en el primer caso, la regla de control *NeibCirclePieces* permite la unión (mediante una acción de control definida desde un objeto tipo "S_CtrlAction" denominado *MergeCirclePieces*) de un segmento que parece ser un fragmento de círculo con otro segmento

vecino de características similares. La evaluación de la regla se realiza de manera condicional. Concretamente, la unión de los dos segmentos se anulará si el segmento resultante no maximiza la suma de los diagnósticos "no ser cuadrado" y "ser círculo" respecto de cualquiera de los dos segmentos originales. En el segundo, se define la regla *NoGreenRule* que se disparará cuando, superándose el umbral, el segmento no verifique la condición *GreenCond*.

Las reglas de control pueden ser agrupadas en conjuntos modulares de reglas mediante el objeto "S_CtrlRuleSet". Su definición incluye el conjunto de reglas, un umbral de disparo por defecto y una estrategia de resolución de conflicto entre las reglas. Las estrategias de selección de la regla activa que se incluyen por defecto son las dos posibilidades más básicas: "la primera mejor" (*FirstBest*) y "la mejor primera" (*BestFirst*). En el primer caso, el orden de declaración de las reglas en el objeto S_CtrlRuleSet asigna un orden de precedencia a las reglas, siendo la más precedente la primera. Sólo se intentará el disparo de la regla n-ésima cuando no sea posible disparar la regla precedente sobre ningún segmento de la partición. La regla se evalúa simultáneamente sobre todos los segmentos de la partición. Esto produce una lista ordenada que contiene la identidad de los segmentos en los que la evaluación de la regla es superior al umbral de disparo. La acción de control asociada a la regla se intenta ejecutar consecutivamente sobre los segmentos contenidos en la lista, empezando por aquellos sobre los que la regla tiene una mejor evaluación. En el segundo esquema de resolución, se asigna a todas las reglas la misma precedencia de forma que todas las reglas se evalúan simultáneamente sobre todos los segmentos. La selección de la regla y el segmento sobre el que se aplica pasa también por la elaboración y recorrido de una lista ordenada similar a la descrita anteriormente. Un ejemplo de objeto de este tipo sería:

```
Define S_CtrlRuleSet      BigAreaPartition
  Strategy:              FirstBest
  CtrlRule:              NoGreenRule, NeibCirclePieces ;
  Threshold:             80
EndDefine
```

En esta definición se ha seleccionado "la primera mejor" como estrategia de selección sobre el conjunto de dos reglas *NoGreenRule* y *NeibCirclePieces*, siendo la primera la de mayor precedencia.

Las acciones de control para modificar la partición se describen mediante objetos "S_CtrlAction". Las acciones de control se dividen en dos grupos según requieran la selección de otro segmento o no. En el primer caso se incluyen las acciones de control *Merge* e *Include*; en el segundo, las acciones *Split*, *Lock* y *Unlock*. La propuesta de sintaxis-definición para este objeto se expresa como sigue:

```
Define S_CtrlAction CtrlActionName  
[ActionElement]  
EndDefine
```

```
ActionElement ::= Action: Merge Locus | Include Locus  
Split | Lock | Unlock
```

```
Locus ::= Target: [Not] ConditionName  
Threshold: Integer
```

La acción *Merge*, implica la unión del segmento foco de atención con aquel vecino que verifique en mayor medida la condición establecida en el campo *Target* y no esté bloqueado, es decir no se haya ejecutado sobre él una acción *Lock*. La evaluación de esta condición puede estar sometida a un posible umbral.

La acción *Include* es equivalente a la acción *Merge* excepto en que los pixels de los segmentos incluidos sólo se utilizan en el cómputo de características (S_Features) que no están basadas en diagnósticos de pixels. Esta acción resulta útil en situaciones donde un segmento pueda haber sido subdividido durante la creación de la partición porque presente a nivel de pixel diferentes zonas (brillos, sombras, reflejos, manchas, etc...). La inclusión de los fragmentos en el segmento principal actualizará los diagnósticos que estén basados en la definición espacial del segmento, pero no aquellos que se computen a partir de diagnósticos de pixels.

La acción *Split* ordena la división del segmento foco de atención en dos segmentos, deshaciendo la última fusión origen de dicho segmento cuando esto es posible. Es decir, mediante la acción *Split* solo es posible alcanzar el nivel de división existente en la partición inicial.

Es interesante hacer notar, que el disparo de una regla no depende exclusivamente de que la evaluación de la condición sobre un cierto segmento supere el umbral de disparo al ser evaluada. Aún cuando el modo de evaluación sea incondicional, la regla puede fallar si por cualquier motivo la acción de control no puede realizarse. Esta situación puede presentarse, por ejemplo, al fallar la evaluación de la condición de selección del segmento vecino en el caso de las acciones "Merge" o "Include"; o porque se ha intentado la acción "Split" sobre un segmento que no ha sufrido previamente una unión con otro segmento. En este sentido, puede afirmarse que la evaluación de una regla de control ocurre en dos fases, condicionada la segunda a la viabilidad de la acción de control propuesta.

Para permitir la comunicación del Procesador de Segmentos con el Relacional o con un programa específico de aplicación, se introduce el objeto "S_Interface", que establece los diagnósticos que le son solicitados al Procesador de Segmentos, permitiendo lanzar los sucesivos procesos de cómputo en los procesadores de Pixels y de Segmentos. De forma equivalente, el objeto "P_Interface" contiene las clases de pixels que deben ser solicitadas al Procesador de Pixels para posibilitar la definición de la Partición y de características. La declaración de estos objetos es idéntica a la del objeto Interface del Procesador de Pixels.

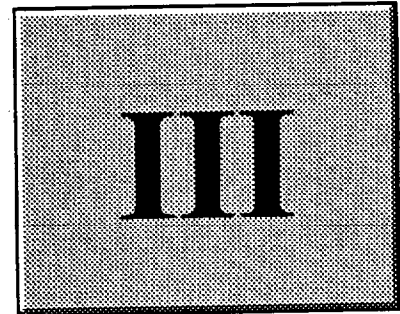
```
Define S_Interface
      Class: segment_class_list;
EndDefine
```

Las estrategias de resolución de conflictos implementadas hasta ahora son, evidentemente, muy sencillas. Sin embargo, la organización del procesador TD permite la incorporación de otras estrategias más sofisticadas [Rich-91] donde, por ejemplo, se pudieran incluir meta-reglas que evaluaran el estado de la Partición o se posibilitara el que la precedencia de las reglas pudiera ser evaluada dinámicamente, etc. Si bien ciertos esquemas podrían ser de un gran interés, esta posible

Capítulo II

sofisticación debe ser atemperada para no introducir una complejidad que puede resultar innecesaria o no resultar aplicable en este nivel. La operación del Procesador de Segmentos está centrada en el segmento y su entorno, de forma que el control que se tiene sobre el estado global de la Partición a partir de la evaluación de reglas sobre los segmentos es necesariamente limitada. En coherencia con esta idea, se ha considerado que las reglas de control y las acciones asociadas habían de tener una orientación localista y ser, por tanto, de naturaleza sencilla. Las reglas o acciones de control más complejas que se encuentran en numerosos sistemas de interpretación de imágenes (agrupación de segmentos en áreas activas, la realización de agrupamientos perceptuales, ...) están basadas en entidades más complejas constituidas a partir de los segmentos (como los segmentos se constituyen a partir de los pixels) que son propias de un nivel superior. Así también, aunque es posible definir varios conjuntos de reglas de control, se ha considerado que la conmutación desde un paquete de reglas a otro - sólo un conjunto de reglas de control puede estar activo en un instante dado - no debe realizarse desde el propio nivel de los segmentos sino desde un nivel superior, donde sea posible establecer criterios de activación desde una perspectiva más amplia y mecanismos de activación/selección más ricos. Una posibilidad interesante compatible con este esquema podría considerar la definición de distintos tipos de áreas de atención y la aplicación sobre ellas de conjuntos de reglas de control especializados.

CAPÍTULO



Sobre el diseño y la implementación.

III.1 INTRODUCCIÓN.

Este capítulo se dedica fundamentalmente a la descripción de elementos relacionados con la implementación de SVEX. Aunque concebidos para cubrir objetivos concretos en el diseño del sistema, su posible interés y aplicabilidad trasciende los límites de este trabajo y en esta creencia se ha pormenorizado su descripción. Uno de estos elementos es el presegmentador "Flood" que se utiliza para producir una segmentación inicial de la imagen a partir de los mapas de diagnóstico obtenidos del Procesador de Pixels. El segundo elemento de la implementación que se describe en este capítulo es un algoritmo de análisis de componentes conectadas (AACC) que se utiliza en SVEX como pieza clave para el análisis topológico y relacional de la partición lo que permite extraer de forma eficiente información detallada sobre cada componente (bordes, coordenadas de los pixels que la integran, la lista ordenada de sus vecinos, el rectángulo circunscrito menor, etc...).

Este capítulo incluye también una descripción del entorno del que se ha dotado a SVEX para facilitar el desarrollo de aplicaciones. Este entorno está integrado por la primera generación de un conjunto de herramientas que incluye los compiladores del lenguaje, un editor icónico, una

herramienta asistida para la "recolección" de muestras, esto es, pixels o segmentos pertenecientes a una cierta clase simbólica, y otras utilidades orientadas a facilitar la depuración de programas en SVEX.

III.2 CONSIDERACIONES SOBRE ALGUNOS ASPECTOS DE LA IMPLEMENTACIÓN.

En la primera parte de esta sección se discuten los fundamentos del módulo de presegmentación "Flood" y se describe cómo puede programarse para generar la segmentación inicial de una imagen combinando múltiples diagnósticos de pixels. En la segunda parte, se pormenorizará acerca del algoritmo de análisis de componentes conexas empleado en SVEX para obtener información topológica y relacional de los segmentos incluidos en la partición de la imagen y analizaremos sus principales características. El Apéndice A contiene el pseudocódigo de las partes más importantes del algoritmo.

III.2.1 EL PRESEGMENTADOR "FLOOD".

El módulo de presegmentación ha sido concebido como una parte intercambiable del Procesador de Segmentos para permitir la utilización de diferentes estrategias de segmentación en la generación de la partición inicial de la imagen en función de la naturaleza del problema. En este apartado describiremos uno de los posibles módulos de presegmentación, denominado "Flood", inspirado en la Transformada Watershed. El presegmentador obtiene una partición inicial (mapa de color) de la imagen utilizando múltiples mapas de diagnóstico proporcionados por el Procesador de Pixels. La partición se organiza en base a la definición de una tipología de segmentos que describe las clases de segmentos sobre los que se articula la segmentación de la imagen.

En este epígrafe se describirá en primer lugar la transformada Watershed como fundamento conceptual de las operaciones del presegmentador "Flood". Se tratarán a continuación dos conceptos que determinan la operación del presegmentador como son la

definición del "perfil topográfico" y la selección de "marcadores" y se describirán los objetos que se incorporan al lenguaje para permitir la programación del presegmentador.

III.2.1.1 LA TRANSFORMADA WATERSHED.

En el epígrafe I.3.2.2, dedicado a la descripción de métodos de segmentación de imágenes basados en crecimiento de regiones, se presentó la transformada Watershed de manera no formal utilizando un símil basado en la inmersión de un perfil topográfico en un lago. Dada su relevancia dentro del marco de este trabajo, se ofrece en esta sección una descripción más detallada y formal basada en el mismo símil.

Definiciones básicas.

Sea I una imagen digital multivaluada en niveles de gris y sea F otra imagen digital multivaluada que represente la topografía de I , como por ejemplo, el módulo del gradiente de I , y cuyo dominio de definición denotaremos por $D_F \subset Z^2$. Asumimos que tanto I como F toman valores discretos (grises) en un rango dado $[0, N]$, donde N es un entero positivo arbitrario:

$$F \begin{cases} D_F \Rightarrow \{0, 1, \dots, N\} \\ p \Rightarrow F(p) \end{cases}$$

Sea G una rejilla digital, $G \subset Z^2 \times Z^2$, de cualquier tipo: cuadrada 4 u 8 conectada o hexagonal 6-conectada.

Definición 1. Un camino P de longitud L entre dos pixels p y q en una imagen F es una $(L+1)$ -tupla de pixels $(p_0, p_1, \dots, p_{L-1}, p_L)$ tal que $p_0=p$, $p_L=q$ y $\forall_i \in [1, L]$, $(p_{i-1}, p_i) \in G$. Denominamos $L(P)$ a la longitud de un camino dado P y $N_G(p)$ al conjunto de pixels vecinos de un pixel p con respecto a G , esto es:

$$N_G(p) = \{p' \in Z^2, (p, p') \in G\}$$

Capítulo III

Definición 2. La distancia geodésica, $d_A(x,y)$, entre dos pixel x e y en un cierto subconjunto de pixels conexos $A \subset G$ (figura III.1) es la ínfima de las longitudes de los caminos que, estando completamente incluidos en A , unen x con y .

$$d_A(x,y) = \inf \{ L(P), \text{ siendo } P \text{ un camino de } x \text{ a } y \text{ totalmente incluido en } A \}$$

Definición 3. Sea el conjunto $B, \{B_1, B_2, \dots, B_k\}$, de las componentes conectadas de A . La zona de influencia, $iz_A(B_i)$, de una componente conectada B_i de A es el lugar de los puntos de A cuya distancia geodésica a B_i es más pequeña que su distancia geodésica a cualquier otra componente de B .

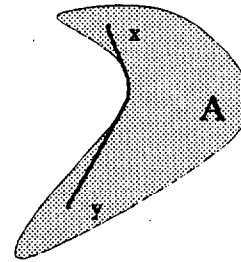


Figura III.1 Distancia geodésica entre x e y en el interior de A .

$$iz_A(B_i) = \{ p \in A, \forall_j \in [1,k] / \{i\}, d_A(p,B_i) < d_A(p,B_j) \}$$

El conjunto de los puntos de A que no se encuentran en alguna zona de influencia geodésica, constituye el "esqueleto de zonas de influencia" de B dentro de A , denominado $SKIZ_A(B)$:

$$SKIZ_A(B) = A / IZ_A(B) \text{ con } IZ_A(B) = \bigcup_{i \in [1;k]} iz_A(B_i)$$

Definición 4. Un mínimo de F de altura h es una meseta de pixels de valor h desde los que es imposible alcanzar un punto de altura inferior sin encontrar una colina.

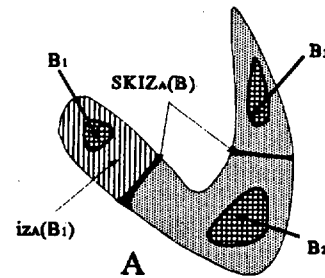


Figura III.2 Esqueleto de zonas de influencia de B dentro de A y zona de influencia geodésica de la componente B_1 .

$$\forall_p \in M, \forall_q \in M \text{ tal que } F(q) \leq F(p)$$

$$\forall_p \in (p_0, p_1, \dots, p_{L-1}, p_L) \text{ tal que } p_0 = p \text{ y } p_L = q$$

$$\exists i \in [1,L] \text{ tal que } F(p_i) > F(p_0)$$

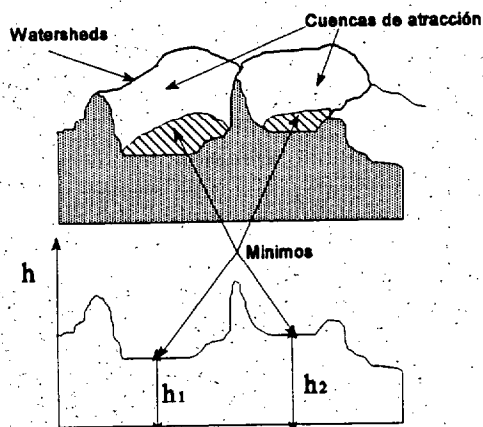


Figura III.3 Mínimos, cuencas y watersheds de una superficie.

Un mínimo es un área conectada, conjunto de pixels iso-intensidad cuyo nivel de gris (valor) es estrictamente menor que el de sus pixels vecinos.

Definición 5. La **cuenca de atracción** (catchment basin), $C(M)$, asociada a un mínimo M es un conjunto de pixels $p \in D_F$ tal que si dejáramos caer una gota de agua desde p fluiría hacia abajo sobre el relieve, siguiendo un cierto camino descendente denominado "corriente" de p que eventualmente termina en M (figura III.3). Las líneas que separan las diferentes cuencas, cada una asociada con un único mínimo, se les denomina "**líneas de vertiente**" o "**watersheds**" de F .

Las líneas de vertiente de F definen las "cordilleras" que rodean completamente a cada cuenca se corresponden, en la imagen original I , con contornos o fronteras entre regiones. El propósito del algoritmo de la transformada Watershed es el de teselar o dividir la imagen en este conjunto de cuencas de atracción que - en principio - se corresponden con zonas aproximadamente homogéneas de la imagen I .

Algoritmo "inundación".

Para comprender como progresa el algoritmo, imaginemos que la superficie que representa el perfil topográfico se ha perforado en las zonas de mínimos y se sumerge lentamente en un lago. Las aguas del lago inundarán progresivamente las cuencas de atracción de los mínimos, empezando por los de menor cota. Supongamos que se impone la condición de que las aguas que

Capítulo III

provengan de diferentes mínimos no pueden mezclarse, de manera que en aquellos pixels donde se produce el encuentro de aguas de diferentes mínimos se levanta un "dique". Al final de este proceso de inmersión, cada mínimo estará completamente rodeado por un dique que delimitará su cuenca de atracción y cuya posición corresponderá con la de las líneas de vertiente o watersheds.

Para expresar de manera más formal y precisa este proceso, sea I la imagen bajo estudio y F la imagen que representa su perfil topográfico. Llamemos h_{\min} al valor más pequeño de F en el dominio D_F y sea $T_h(F)$ el conjunto de pixels de F con altura inferior a h :

$$T_h(F) = \{ p \in D_F, F(p) \leq h \}$$

Asimismo, sea $C(M)$ la cuenca asociada al mínimo M y definamos $C_h(M)$ como el subconjunto de $C(M)$ integrado por los puntos que tienen una altura menor o igual a h :

$$C_h(M) = \{ p \in C(M), F(p) \leq h \} = C(M) \cap T_h(F)$$

Para simular el proceso de inmersión comenzaremos con el conjunto $T_{h_{\min}}(F)$, de los puntos que primero alcanzará el agua cuando se inicie la inundación y son el conjunto inicial de un proceso recursivo, esto es:

$$X_{h_{\min}} = T_{h_{\min}}(F)$$

Donde $X_{h_{\min}}$ contiene los puntos de F que integran los mínimos de altitud más baja. Consideremos a continuación el conjunto umbral de F al nivel $h_{\min}+1$, $T_{h_{\min}+1}(F)$. Obviamente $X_{h_{\min}} \subseteq T_{h_{\min}+1}(F)$.

Si Y es una de las componentes conectadas de $T_{h_{\min}+1}(F)$ existen tres posibles relaciones de inclusión entre Y e $Y \cap X_{h_{\min}}$:

a) $Y \cap X_{h_{\min}} = \emptyset$. En este caso, Y es un nuevo mínimo de F y, de acuerdo con las definiciones anteriores, se trata de una "meseta" en el nivel h+1.

$$\forall p \in Y, \begin{cases} p \in X_{h_{\min}} \Rightarrow F(p) \geq h_{\min} + 1 \\ p \in Y \Rightarrow F(p) \leq h_{\min} + 1 \end{cases}$$

Más aún, los pixels que rodean a Y no pertenecen a $T_{h_{\min}+1}(F)$ y tienen, por tanto, una altura estrictamente superior a $h_{\min}+1$. Este nuevo mínimo también debe ser considerado como

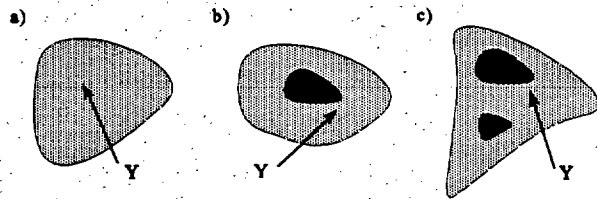


Figura III.4 Las tres posibles relaciones de inclusión entre Y e $Y \cap X_{h_{\min}}$.

"agujereado" de manera que su cuenca también se inundará progresivamente.

b) $Y \cap X_{h_{\min}} \neq \emptyset$ y está conectado. En este caso, Y se corresponde con los pixels que conforman la cuenca asociada con el mínimo $Y \cap X_{h_{\min}}$ y tienen una altura menor o igual a $h_{\min}+1$:

$$Y = C_{h_{\min}+1}(Y \cap X_{h_{\min}})$$

c) $Y \cap X_{h_{\min}} \neq \emptyset$ y no está conectado.

En este último caso, Y contiene diferentes mínimos de F. Llamemos Z_1, Z_2, \dots, Z_n a estos posibles mínimos y sea Z_i uno cualquiera de ellos. En esta circunstancia, la mejor elección posible para $C_{h_{\min}+1}(Z_i)$ la define la zona de influencia de Z_i dentro de Y:

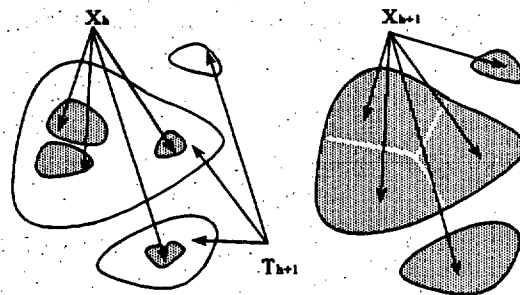


Figura III.5 Relación de recurrencia entre X_h y X_{h+1} .

$$C_{h_{\min}+1}(Z_i) = iz_Y(Z_i)$$

Estas relaciones de inclusión se muestran en la figura III.4. El análisis de estas relaciones permite establecer la recursión como:

$$X_{h_{\min}+1} = \min_{h_{\min}+1} \bigcup_{IZ_{T_{h_{\min}+1}(F)}}(X_{h_{\min}})$$

La figura III.5 ilustra la relación de recurrencia entre dos niveles sucesivos. Esta relación recursiva, aplicada entre dos niveles consecutivos cualesquiera, conduce a una nueva definición de la cuencas de atracción por inmersión.

Definición 6. El conjunto de cuencas de atracción de una imagen F es igual al conjunto $X_{h_{\max}}$ obtenido de la siguiente relación de recursión:

1. $X_{h_{\min}} = T_{h_{\min}}(F)$,
2. $\forall h \in [h_{\min}, h_{\max}-1]$, $X_{h+1} = \min_{h+1} \bigcup_{IZ_{T_{h+1}(F)}}(X_h)$

Las líneas de vertiente o watersheds de F corresponden al complementario de este conjunto en D_F , es decir, al conjunto de puntos de D_F que no pertenecen a ninguna cuenca.

Utilización de la transformada Watershed en la segmentación de imágenes.

Como ya se indicó en el epígrafe I.3.2.2, la aplicación directa de la transformada Watershed, tal y como ha sido descrita hasta ahora, en problemas de segmentación de imágenes no resulta adecuada debido a la excesiva sobresegmentación de la partición resultante. La solución habitual consiste en la utilización de "marcadores" para definir conjuntos conexos de puntos que fijen los mínimos a emplear en la transformada Watershed. En general, los marcadores seleccionados para una imagen reflejan el conocimiento a priori que se tiene sobre el contenido de la imagen y su caracterización en un cierto conjunto de clases o tipos de segmentos. De esta forma se consigue que las zonas seleccionadas por el marcador de una clase sean zonas relevantes que se ajusten a la descripción de determinadas clases de segmentos. Este esquema básico de definición y selección de los mínimos permite el control sobre el grado de detalle de la segmentación resultante.

En resumen, la aplicación de la transformada Watershed en la segmentación de imágenes comprende dos fases:

1. Utilización de un conjunto de marcadores para seleccionar los puntos prototipo de las clases de segmentos, y modificación del gradiente de la imagen para que los puntos marcados sean los únicos mínimos.
2. Aplicación de la transformada Watershed desde el gradiente modificado de la imagen. Esto producirá la teselación de la imagen en términos de las cuencas de atracción de los mínimos definidos en la primera fase.

III.2.1.2 ADAPTACIÓN DE LA TRANSFORMADA WATERSHED EN EL PRESEGMENTADOR "FLOOD".

Definición de la superficie topográfica.

La transformada Watershed emplea el gradiente de la imagen o una medida similar para definir la superficie topográfica sobre la cual se produce la inundación. En el caso del presegmentador, la segmentación se puede realizar a partir de múltiples mapas de diagnósticos, que también pueden intervenir en la definición del perfil "topográfico" de la imagen (figura III.6). En aras de proveer al sistema con un alto grado de flexibilidad, resulta interesante permitir que la definición de dicho perfil pueda establecerse desde diferentes mapas de pixels empleando operadores variados. En este sentido, cada mapa de pixels que se declara en el presegmentador puede definir un perfil elemental mediante la aplicación de un determinado operador o procedimiento sobre el mapa en cuestión.

Perfil "Topográfico" de la imagen.

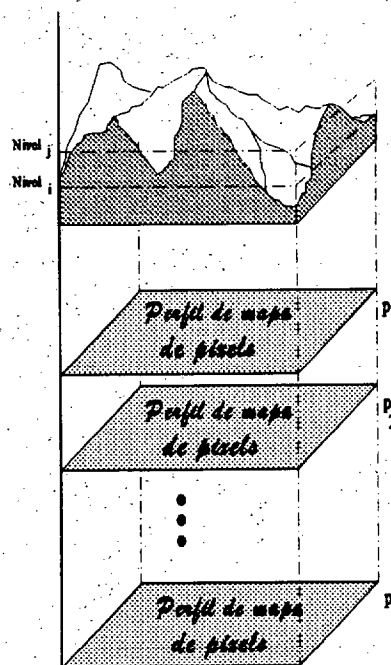


Figura III.6 Definición del perfil topográfico de una imagen desde múltiples mapas de pixels.

Capítulo III

La definición del perfil topográfico de la imagen se establece entonces por la combinación lineal de los perfiles elementales de acuerdo con un peso asignado a cada mapa. La posibilidad de ponderar las contribuciones de los perfiles de cada mapa resulta útil en casos en los que alguno de los mapas de diagnóstico se ha obtenido en el Procesador de Pixels a partir de medidas estadísticas utilizando grandes ventanas alrededor de cada pixel. En estos casos, los diagnósticos suelen ser deficientes en zonas próximas a los bordes de las regiones de la imagen, por lo que pueden resultar interesante atenuar o eliminar su contribución al perfil de la imagen.

Selección de marcadores.

La selección de marcadores es, sin lugar a dudas, el elemento más importante en la actuación del presegmentador como mecanismo de transformación de una representación basada en los pixels a otra basada en los segmentos. La selección de los marcadores a utilizar está condicionada por el conocimiento que sobre la naturaleza de las clases de segmentos potencialmente presentes en la imagen se tiene a priori. Este conocimiento se articula desde ambas representaciones, la de pixels y la de segmentos, para describir las clases de segmentos en base a propiedades de sus pixels y posibles criterios relacionados con la distribución espacial de estos. La definición de estas clases se establece, por tanto, combinando dos aspectos. Uno, propio del dominio de los pixels, determina la combinación de diagnósticos que define las propiedades que deben verificar los pixels de una cierta clase de segmento. El otro, más cercano al dominio de los segmentos, utiliza criterios de densidad de puntos, uniformidad y forma para estimar como relevante una cierta agrupación conexas de puntos e identificarla con una clase de segmentos.

En el contexto del presegmentador, cada una de las clases de segmentos contempladas en la partición inicial define un "*Prototipo*". Este concepto sirve para denotar a un pixel como característico de un tipo de segmento y en su definición se integran los dos aspectos antes aludidos. En primer lugar, es necesario caracterizar las propiedades de los pixels en términos de los mapas de diagnóstico solicitados al Procesador de Pixels. Este conjunto de mapas constituye un espacio de representación donde se asocia a cada diagnóstico una dimensión. Cada prototipo está representado en este espacio por un hipervolumen que encierra la distribución de diagnósticos

característica de este tipo de pixels. En general, el hipervolumen asignado al prototipo k-ésimo se define simbólicamente como:

$$HV_k = F(Pm_1, Pm_2, \dots, Pm_n, \alpha_{ki}, \beta_{kj}, \dots, \zeta_{kn})$$

donde los Pm_h representan los mapas de diagnóstico que integran el espacio de representación; F es una cierta función que define el hipervolumen ocupado por cada prototipo y $\alpha_{ki}, \beta_{kj}, \dots, \zeta_{kn}$, son los argumentos de la función F que definen al prototipo k-ésimo en dicho espacio, p.e. coordenadas y dispersiones en cada uno de los mapas. En esta implementación, la función F asocia a cada prototipo un hiperparalelepípedo recto, cuyo centroide se sitúa en los valores "medios" de cada dimensión o diagnóstico y cuya elongación en cada uno de las direcciones indica la dispersión característica de este prototipo en torno a su valor medio. Para cada prototipo, la definición del hipervolumen correspondiente en el espacio de representación delimita al marcador asociado el ámbito de búsqueda de puntos prototipo en dicho espacio.

Mediante lo anterior es posible decidir sobre la pertenencia de un pixel a un determinado prototipo sobre la base de los diagnósticos asociados a ese pixel exclusivamente. Esta definición parcial puede ser enriquecida mediante la consideración de restricciones sobre la distribución espacial de los pixels incluidos en el hipervolumen de un prototipo. Estas restricciones permiten utilizar en la búsqueda de las clases de segmentos determinado conocimiento acerca de la extensión mínima del segmento, de la uniformidad de su definición espacial, de su posición en el perfil de la imagen o incluso sobre su forma. Como quiera que no existe una definición de marcador que resulte óptima en la definición de todo tipo de prototipos o clases de segmentos, el presegmentador "Flood" permite la selección de un tipo de marcador por cada prototipo de un conjunto o librería de marcadores, lo que en último término se traduce en una llamada a un procedimiento específico de análisis.

El algoritmo.

El algoritmo empleado en el presegmentador "Flood" ha simplificado algunos aspectos de la transformada Watershed tal y como ha sido formulada más arriba. Los elementos

fundamentales del mismo son los siguientes:

1. Cálculo del perfil de los mapas de pixels y combinación lineal de los mismos para definir el perfil que gobernará el proceso de inundación.
2. Para cada prototipo, definición de los mínimos de dicho prototipo mediante la actuación de un marcador sobre los mapas de diagnóstico que componen el espacio de representación.
3. Inundación desde los mínimos.

- Se crean dos pilas FIFO (*pila1* y *pila2*) de dimensión igual al número de pixels de la imagen no incluidos en los mínimos.

- Se define el número mínimo de prototipos adyacentes (*NumMin*) para que un pixel pueda ser etiquetado.

- Se introducen en una de las pilas (*pila1*) todos los pixels adyacentes a mínimos de algún prototipo.

Desde *NIVEL*= altura mínima del perfil hasta *NIVEL*= altura máxima hacer:

Mientras *pila1* no esté vacía hacer

Extraer *pixel* de *pila1*.

if (*pixel* está situado a una altura > *NIVEL*)

then introducir *pixel* en *pila2*.

else

Recuperar las etiquetas de pixels prototipos adyacentes a *pixel*.

if (el número de pixels prototipos adyacentes < *NumMin*)
then introducir *pixel* en *pila2*.

else

if (todos los pixels adyacentes son del mismo prototipo)

then dar a *pixel* la etiqueta del prototipo

else emplear estrategia de selección del prototipo óptimo y dar etiqueta a *pixel*.

end-if

Introducir en *pila1* los pixels adyacentes a *pixel*.

end-if

end-if

end-mientras

Intercambiar *pila1* y *pila2*.

end-desde.

Este algoritmo supone una simplificación respecto del esquema básico de la transformada Watershed pues omite el uso de la distancia geodésica para resolver el etiquetado de puntos en situaciones donde $Y \cap X_{\text{NIVEL}} \neq \emptyset$ y es multiconectada. Se ha sustituido por un esquema oportunista del tipo "el primero el mejor" más simple, pero sobre el que la implementación permite un cierto grado de control. Así es posible especificar en función del prototipo si los pixels adyacentes son 4 u 8 conectados y el número mínimo (NumMin, típicamente 1 o 2) de pixels adyacentes que deben ser prototipo para que un pixel no etiquetado pueda recibir una etiqueta. Esto hace que el crecimiento de las regiones resulte más competitivo.

Finalmente, el etiquetado de aquellos pixels que se encuentran rodeados por varios prototipos, esto es pixels que se corresponderían aproximadamente con las líneas watershed, puede resolverse empleando diferentes estrategias para seleccionar un prototipo de entre sus vecinos. Las estrategias que se han implementado son las más elementales y se corresponden con la selección del prototipo más numeroso o con la selección del prototipo "más similar". La similitud entre un pixel y un prototipo se calcula como la suma cuadrática de las distancias que en cada dimensión separan el diagnóstico del pixel del extremo más próximo del intervalo de definición del prototipo.

El mínimo número de prototipos adyacentes para resolver el etiquetado de un pixel (*NumMin*) y la estrategia de resolución del etiquetado son dos de los tres parámetros que es posible ajustar en el funcionamiento del presegmentador "Flood". El tercer parámetro es la altura máxima a la que se debe detener la inundación. Por defecto, es el nivel máximo del perfil, normalizado siempre a 255, en cuyo caso la partición es "completa", es decir, libre de zonas sin etiquetar. La posibilidad de detener la inundación a un cierto nivel puede resultar de utilidad en algunos contextos [Fuji-90], como cuando lo que se pretende es generar una partición "grosera", que muestre las zonas mejor definidas o más evidentes de la imagen, sin necesidad de perfilar exactamente la forma de los segmentos ni sus relaciones de adyacencia.



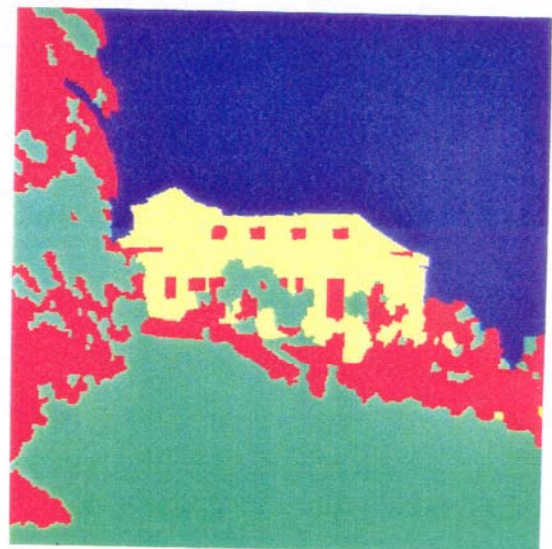
A



B



C



D

Figura III.7 Diferentes etapas en la acción del presegmentador "Flood". A) Imagen original. B) Mínimos correspondientes a diferentes prototipos. C) Un estado intermedio de la inundación. D) Partición final.

La figura III.7 muestra en forma de secuencia las sucesivas etapas de la acción del presegmentador sobre una determinada imagen. La imagen original (A) se procesa en el nivel de los pixels para generar un conjunto de mapas de diagnóstico ("verde", "cielo", ...) a partir de los cuales se definen los prototipos del presegmentador. La figura III.7 (B) contiene los mínimos seleccionados por los "marcadores" de cada prototipo a partir de los cuales comienza la "inundación". Los apartados C y D de la misma figura muestran, respectivamente, el aspecto de la partición cuando la inundación ha progresado hasta un cierto nivel y el resultado final, en el que la inundación ha progresado hasta el máximo nivel del perfil. En el capítulo IV puede encontrarse una descripción completa acerca de los diagnósticos de pixels y de los prototipos empleados en la segmentación de esta imagen.

III.2.1.3 OBJETOS DEL LENGUAJE.

Como ya se indicó al describir los objetos del Procesador de Segmentos, la selección de un cierto módulo de presegmentación en ese procesador supone la introducción en el lenguaje de SVEX del conjunto de objetos asociado a la programación de dicho módulo. Los objetos introducidos por el presegmentador "Flood" son: "*Profiler*", "*Pixel_Map*", "*Marker*" y "*Prototype*".

El objeto "**Profiler**" se asocia a un procedimiento para calcular el perfil de un mapa de pixels. El presegmentador "Flood" se aplica en la definición de particiones donde los segmentos pueden ser tanto "regiones" como "contornos". En el primer caso, los procedimientos que exploran los mapas de pixels son de naturaleza claramente bidimensional como es el caso de cualquier procedimiento que calcule el gradiente de una imagen. En el segundo caso, el cálculo del "perfil de la imagen" normalmente sólo tiene sentido sobre la estructura lineal de los contornos por lo que resulta interesante poder definir procedimientos que tengan en cuenta esta particularidad. Para ilustrar esta última posibilidad, imaginemos una situación en la que se desea generar una partición de los contornos extraídos de una imagen en segmentos tipo "recto" y tipo "curvo". En este caso, el mecanismo de operación ofrecido por el presegmentador "Flood" es aún válido si el perfil de la "imagen de contornos" se define adecuadamente. Por ejemplo, una medida equivalente al gradiente y que puede servir para definir el perfil de un mapa de contornos es la

Capítulo III

curvatura. Por todo esto, los objetos que en el presegmentador representan a procedimientos de análisis o exploración, v.g. los objetos "Profiler" y "Marker", incorporan en su declaración si se aplican sobre contornos o sobre ventanas. Las siguientes declaraciones de objetos "Profiler" pueden servir como ejemplo de la sintaxis de este tipo de objetos:

```
Define Profiler Gradient
  Procedure: WinGradient
  AppliedTo: Window
  ParamNum: 1
EndDefine

Define Profiler ContourProfile
  Procedure: ContourCurvature
  AppliedTo: Contour
  ParamNum: 1
EndDefine
```

En el primer caso, el objeto *Gradient* representa a un operador para calcular el perfil de un mapa de pixels que utiliza un procedimiento, *WinGradient*, que se aplica sobre una ventana y requiere de la especificación de un parámetro. En el segundo ejemplo, el objeto "Profiler" *ContourProfile* denota a un procedimiento, *ContourCurvature*, que define el perfil de un mapa de pixels a partir del cálculo de la curvatura de los contornos contenidos en dicho mapa, para lo cual también requiere de un parámetro.

El objeto "**Pixel_Map**" declara los mapas de diagnóstico utilizados en la síntesis de la partición inicial de la imagen. Son diagnósticos de pixels que previamente han sido solicitados al Procesador de Pixels desde la unidad de control del Procesador de Segmentos. Pueden ser utilizados por el presegmentador para definir una contribución al perfil de la imagen y/o en la descripción de los prototipos. La siguiente declaración ilustra la sintaxis del objeto "Pixel_Map":

```
Define Pixel_Map GreenMeanMap
  Class: GreenMean
  Profiler: Gradient
  Weight: 10
  Parameter: 5;
EndDefine
```

En esta declaración el "Pixel_Map" GreenMeanMap se obtiene desde el mapa de diagnóstico GreenMean y contribuye a definir el perfil topográfico con un peso (Weight) o importancia relativa de 10; cuyo significado último depende de los pesos asignados a otros mapas. El perfil elemental de este mapa se obtiene de la actuación del procedimiento indicado por el objeto "Profiler", aquí denominado Gradient, al que se le pasa un parámetro.

El objeto "Marker" define un procedimiento que puede servir como "marcador" de un determinado prototipo. Su sintaxis es análoga a la del objeto "Profiler", aunque su funcionalidad es absolutamente diferente. Como ocurre con el objeto "Profiler", el objeto "Marker" también puede ser aplicado tanto sobre contornos como sobre ventanas, dependiendo del diseño del procedimiento. Como ejemplo de este tipo de objetos considérese la declaración del objeto *WinStandardMarker*, donde se aplica un procedimiento denominado *WinStandard* para marcar los pixels que se identifican como "mínimos" de un cierto prototipo. Este procedimiento explora los mapas de pixels mediante una ventana cuya dimensión lateral se pasa como parámetro.

```
Define Marker WinStandardMarker
  Procedure: WinStandard
  AppliedTo: Window
  ParamNum: 3
EndDefine
```

El objeto "Prototype" es el elemento central del esquema de operación del presegmentador "Flood" pues define las clases de segmentos o "prototipos" que pueden aparecer en la primera partición de la imagen. Considérese la siguiente declaración de un objeto "Prototype":

```
Define Prototype Green_Textured_Region
  MapList: GreenMean, HighTexture ;
  Centroid: 80, 80;
  Dispersion: 30, 20;
  Marker: WinStandardMarker
  Parameter: 10, 50, 100;
  Mode: Merge
  Color: Cyan
  Connectivity: 8-Connectivity
```

```
Show
      Zoom:      2
      PictureFile: OutdoorPicture
EndDefine
```

Los tres primeros campos están relacionados con la definición del prototipo en términos de los diagnósticos de pixels que le son relevantes, en este caso *GreenMean*, *HighTexture*, declarados en el campo "MapList". La ubicación del centroide del hiperparalelepípedo asignado a este prototipo se declara en "Centroid" y en "Dispersion" se define la semilongitud del intervalo de definición del prototipo alrededor del centroide definido para cada diagnóstico. En este caso los pixels que tuvieran un grado de pertenencia entre 50 y 100 a la clase de pixels *GreenMean* y entre 60 y 100 a la clase *HighTexture* estarían situados dentro del espacio de definición de este prototipo. Para definir los mínimos de este prototipo se emplea el marcador *WinStandardMarker* al que se le pasan tres parámetros. "Mode" y "Connectivity", los cuales determinan el comportamiento de este prototipo durante el proceso de crecimiento de regiones o de inundación. El primero, "Mode", especifica si se desea que las cuencas de atracción de este prototipo compartan un mismo color en el mapa de colores o no (Not_Merge). En el ejemplo anterior, el token "Merge" indica que todas cuencas de este prototipo participan del mismo color, declarado en "Color", de las cuales aquellas que sean adyacentes integrarán un único segmento en la partición. El parámetro "Connectivity" permite fijar la definición de adyacencia (4 u 8 conectividad) para los pixels de este prototipo. La última parte de la declaración fija parámetros opcionales, relacionados con la visualización de este prototipo al finalizar la acción del presegmentador. Éstos son el factor de aumento, "Zoom", y la imagen ("PictureFile") sobre la que se superpone la definición del prototipo para su visualización.

La utilización del objeto "S_MakePartition" permite sincronizar la declaración y acción del presegmentador con el resto del Procesador de Segmentos, tal y como se discutió en la descripción de los objetos de este Procesador. Común a todos los posibles módulos de presegmentación, el objeto "S_MakePartition" permite fijar aquellos parámetros relacionados con la acción de los posibles presegmentadores. En el caso del presegmentador "Flood", se pueden especificar tres parámetros en el campo "Parameter" para controlar la segmentación. Éstos son el mínimo número de prototipos adyacentes para que pueda resolverse el etiquetado de un pixel

pixel durante la inundación (denominado NumMin) en la descripción del algoritmo), la estrategia de resolución en el caso de adyacencia de diferentes prototipos y el nivel máximo del perfil (altura máxima) al que se desea detener la inundación.

III.2.2 UN ALGORITMO DE ANÁLISIS DE COMPONENTES CONEXAS.

III.2.2.1 INTRODUCCIÓN.

Una vez que el módulo de presegmentación ha generado una segmentación primaria de la imagen, v. g. un mapa de colores, es necesario extraer de la misma la información topológica y relacional relevante para producir la partición inicial de la imagen. Sobre cada segmento o componente conexas (conjunto conexo de pixels del mismo color o etiqueta) contenida en el mapa de colores, esta información se concreta en los siguientes elementos:

- a. Una codificación del borde externo del segmento y de los posibles bordes internos.
- b. Una lista ordenada de los segmentos vecinos encontrados al recorrer cada borde del segmento (externo e internos). Los registros de vecindad incluyen la identidad del segmento vecino, detallando el borde sobre el que tiene lugar la vecindad, su color en el mapa de colores y las coordenadas del primer punto del borde común a ambos segmentos.
- c. Las coordenadas de los pixels que constituyen el segmento.

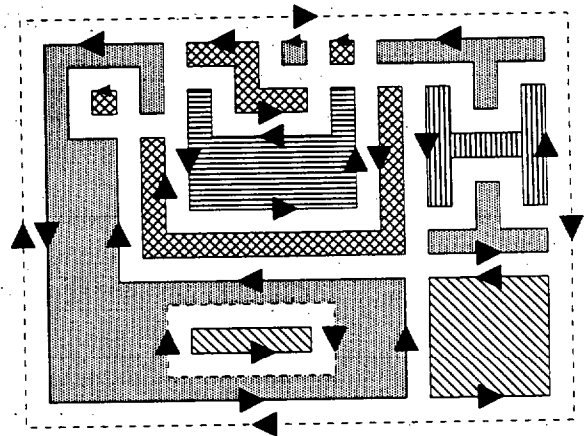
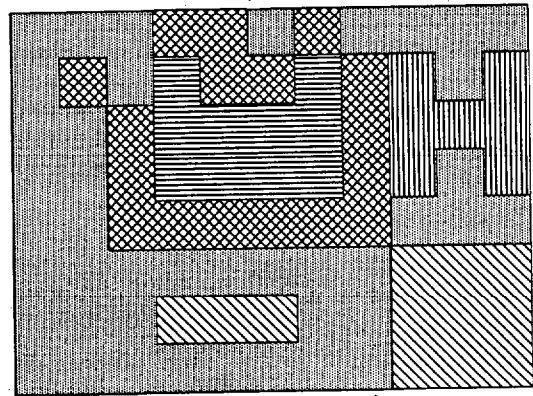


Figura III.8 Mapa de colores y su descomposición en componentes conexas.

d. La ventana rectangular que circunscribe al segmento. La ventana está orientada paralelamente a los ejes de coordenadas o marcos de la imagen.

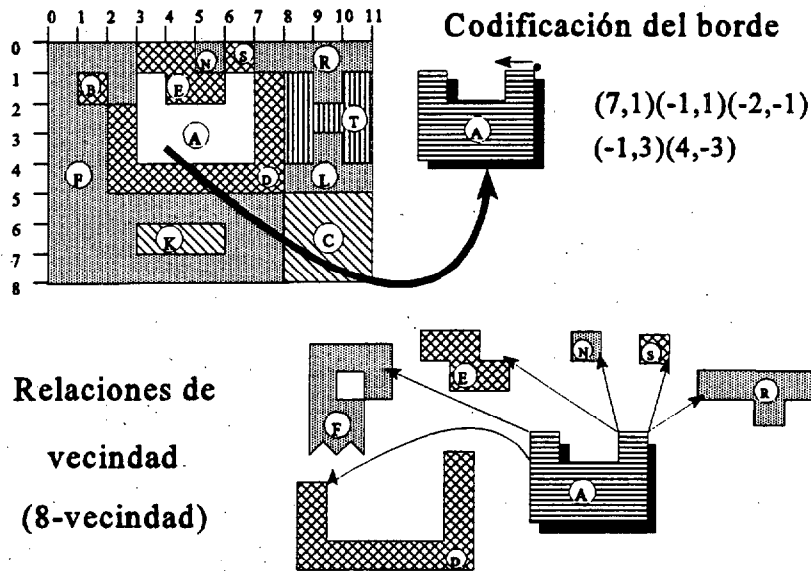


Figura III.9 Codificación del borde de la componente A y registro de sus vecinos.

La obtención de esta información es el cometido característico de los Algoritmos de Análisis de Componentes Conexas (AACC) [Rose-66]. Los AACC descritos en la literatura pueden ser clasificados en base a una serie de elementos, entre ellos: su naturaleza (secuencial o paralela), el tipo de mapas que maneja (binarios o multivaluados), la definición de conectividad empleada (4 u 8 conectividad) y su ámbito de aplicación (imágenes 2D o 3D). La mayor parte de estos trabajos [Lumi-83a][Same-86][Yang-92] se circunscriben al desarrollo de algoritmos secuenciales para análisis de mapas bidimensionales, binarios y 4-conectados sobre arquitecturas seriales (Von Neumann). Un problema que se ha considerado frecuentemente ha sido el del análisis de imágenes que por su extensión no pueden ser almacenadas en memoria [Lumi-83a][Dins-85]. Algunos de estos trabajos [Yang-88] han estado orientados a la implementación en hardware de este tipo de algoritmos. Sin embargo, el análisis de componentes conexas en imágenes multivaluadas ha sido escasamente tratado [Dani-82][Mand-90]. En otros trabajos se exploran posibles algoritmos susceptibles de ser ejecutados sobre arquitecturas masivamente paralelas (e.g. un procesador por pixel) [Shil-82][Tuck-86][Mano-89]. El mayor interés en este

campo se centra actualmente en el desarrollo de algoritmos para tratamiento de mapas tridimensionales producidos normalmente por dispositivos de imaginería médica (TAC, RMN, ...) [Lumi-83b][Udup-90][Thur-92].

III.2.2.2 UNA PROPUESTA DE AACC.

El AACC que describiremos aquí es una extensión del algoritmo propuesto en [Mand-90]. Opera sobre mapas multivaluados o multicoloreados de manera secuencial, recorriendo el mapa con una ventana de 2x2 pixels una sola vez. El algoritmo devuelve como resultado una codificación del borde externo y de los posibles bordes internos de cada componente, y la lista ordenada de vecinos encontrados al recorrer cada borde. Las extensiones que se describen a continuación posibilitan, además, la obtención de la lista de pixels que componen el área de cada componente y la detección de la ventana que la circunscribe. En su versión original el algoritmo utilizaba exclusivamente la 4-conectividad para definir tanto la conectividad como la vecindad de una componente. En la versión que describimos, el usuario puede seleccionar 4 u 8 conectividad para las definiciones de conectividad y vecindad de manera independiente. La parte superior de la figura III.8 muestra un mapa de colores conteniendo diferentes componentes. Consideraremos en lo que sigue que el borde o contorno de cada componente discurre en el espacio virtual de separación entre los pixels, y el sentido de recorrido es tal que los pixels de la componente a la que pertenece el borde se sitúan siempre a la izquierda según el sentido de avance. Este convenio permite diferenciar claramente los bordes internos de los bordes externos por el sentido en el que rodean a la componente: horario y antihorario, respectivamente. En la parte inferior de la misma figura, se han reducido en tamaño las componentes para mostrar los bordes. Nótese que los bordes internos están trazados con línea discontinua y los externos con línea continua. Para la correcta aplicación de este algoritmo, es necesario considerar idealmente que el mapa de colores está inmerso en una componente externa cuyo borde interno equivale al borde o marco del mapa de colores. En aras de una mayor claridad, en este ejemplo se ha empleado 4-conectividad para definir las diferentes componentes. La figura III.9 muestra para una componente (A) la definición de su borde externo empleando el código de contorno [Bart-89] y sus relaciones de 8-vecindad. El código de contorno contiene las coordenadas absolutas del punto inicial (7,1) y los sucesivos desplazamientos horizontales y verticales necesarios para recorrer el borde. Nótese que las

Capítulo III

coordenadas verticales aumentan hacia la parte inferior de la página. Las relaciones de vecindad se registran en el orden en el que se encuentran al avanzar sobre un borde en el sentido apropiado, de manera que una componente puede aparecer múltiple veces la lista de vecinos de otra. Cuando se registra una vecindad, se salvan las coordenadas absolutas del punto de contacto, la identidad del borde vecino y su color. Las relaciones de vecindad entre dos componentes, tales como A y D en el ejemplo, no son simétricas en el sentido de que no se registran en el mismo punto. Así la componente D aparece como vecino de la A en el punto (3,2) y A aparece como vecino de D en el punto de coordenadas (7,1). Salvo en el caso de un borde interno, las relaciones de vecindad se registran siempre en puntos donde la vecindad cambia (punto de contacto múltiple). La relación de vecindad entre el borde interno de F y el borde externo de K no puede detectarse por "cambio de vecindad" y ha de establecerse un mecanismo específico que detecte estas situaciones. En estos casos, las coordenadas del punto de vecindad puede escogerse de manera arbitraria (punto de inicio del contorno, esquina inferior derecha, etc.).

III.2.2.2.1 DETECCIÓN DE ESQUINAS EN MAPAS BINARIOS.

Para fijar algunos conceptos, considérese un mapa binario donde se requiere localizar y describir sus componentes 4-conectadas. Para tal fin, basta desplazar sobre cada línea de la imagen una ventana 2x2 y detectar posibles tipos de "esquinas". Definiremos una esquina como la transición de un borde horizontal a un borde vertical o viceversa. Estas transiciones ocurren siempre por pares, de manera que una transición vertical-horizontal del borde de una componente en una línea debe corresponderse con otra horizontal-vertical sobre la misma línea. Si el análisis considera sólo 4-conectividad, sólo son posibles los 8 tipos de pares de *esquinas binarias o esquinas-B* mostradas en la figura III.10, donde el borde

Ventana 2x2	Esquina-B	Par esquinas

Figura III.10 Lista de posibles pares de esquinas binarias.

que se ha considerado ha sido el de la componente de color negro (objeto). Los pares de esquinas-B son los elementos sobre los que es posible basar la representación de los bordes de las componentes durante un análisis secuencial (raster scan) de la imagen. En esencia, esta representación se fundamenta en asociar acciones específicas (creación o cierre de un borde, fusión de dos bordes, prolongación de un borde, etc...) a la aparición de diferentes tipos de pares de esquinas-B. Posteriormente consideraremos en detalle las operaciones que es necesario definir para manipular la representación de las componentes y sus bordes.

III.2.2.2.2 DETECCIÓN DE ESQUINAS EN MAPAS MULTIVALUADOS.

Al recorrer un mapa multivaluado con una ventana 2x2 es posible encontrar hasta 4 etiquetas o colores diferentes, siendo posibles 15 configuraciones o distribuciones de estos colores

Clasificación topológica de esquinas	EsquinasC		Borde superior		Borde inferior	
	ventana	tipo	ventana	esquinaB	ventana	esquinaB
Esquinas L2						
Esquinas L dos colores						
Esquinas T3						
Esquinas T tres colores						
Esquinas X4						
Esquinas X3 tres colores						
Esquinas X2 dos colores						

Figura III.11 Clases de esquinas-C y su descomposición en esquinas-B correspondientes al borde superior e inferior.

Capítulo III

en la ventana. Clasificaremos las combinaciones posibles en tres grupos de **esquinas coloreadas** o **esquinas-C** (figura III.11): esquinas en L con sólo dos colores, esquinas en T con tres colores y esquinas en X con dos, tres o cuatro colores.

La idea básica del algoritmo consiste en considerar que las esquinas-C pueden descomponerse durante el análisis de componentes en pares de esquinas binarias o esquinas-B. Las dos últimas columnas de la figura III.11 muestran cómo se formula la descomposición de esquinas-C, donde se produce una transición horizontal-vertical o viceversa, en dos o más esquinas-B que corresponden a un borde "superior", dirigido de izquierda a derecha, y otro borde "inferior" dirigido de derecha a izquierda. Como antes, el pixel "blanco" en estas columnas indica que el color de estos pixels puede ser cualquiera ("no-importa"), siendo la componente de interés la marcada en negro. En resumen, la idea es considerar que en el espacio virtual entre dos componentes se sitúan siempre dos bordes, uno por cada componente, que discurren en sentidos opuestos y que, llegados a una esquina, siempre es posible distinguirlos como borde superior e inferior.

En el contexto en el que se sitúa este trabajo, resulta necesario extender las capacidades del algoritmo original para poder tratar conectividades y vecindades en diagonal. Este es el caso, por ejemplo, cuando las componentes en una imagen son contornos (v.g. estructuras 8-conexas de un pixel de ancho). En su formulación original, el algoritmo sólo considera 4-conectividad y 4-vecindad, de manera que componentes con conexiones en diagonal resultarían divididas en esos puntos, y lo que es más desfavorable, los diferentes trozos no serían considerados como vecinos. Por tanto, la introducción de los tipos de esquinas X3 y X2 responde al intento de extender este algoritmo para poder considerar la 8-conectividad y también, aunque no de forma directa, 8-vecindad. En cualquier caso, el uso de la 8-conectividad o de la 8-vecindad es opcional y el algoritmo puede realizar el análisis de componentes conexas empleando 4 u 8-conectividad y 4 u 8-vecindad de manera independiente. Cuando se emplea 4-conectividad, las esquinas X3 y X2 se tratan como esquinas tipo X4 aunque algunos pixels puedan compartir el mismo color. La característica excepcional que introduce la consideración de las esquinas X3 y X2 es la realización de una conexión en diagonal que es posible expresar en los mismos términos que permiten tratar

los otros tipos de esquinas tal y como muestra la parte inferior de la figura III.11. La conexión diagonal en el caso de esquinas X3 está claramente definida y no plantea ningún problema, uniéndose los pixels diagonales de igual color. Por el contrario, el caso de de las esquinas X2 es más complicado ya que a partir de la información contenida en la ventana 2x2 resultan dos posibles direcciones para establecer la conexión en diagonal. Es necesario, por tanto, proponer un mecanismo que permita

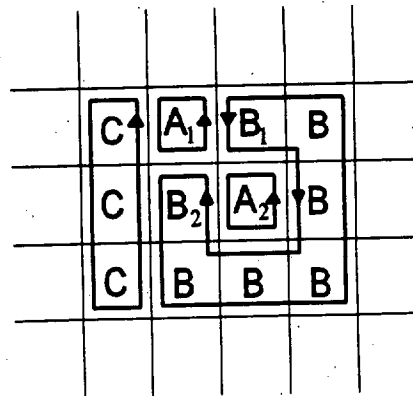


Figura III.12

seleccionar una cierta dirección y resolver la ambigüedad. Nótese, incidentalmente, que el uso de una ventana de mayores dimensiones no resuelve este problema. La consideración conjunta de 8-conectividad y 8-vecindad plantea también un problema adicional que surge del hecho de relacionar como vecinos pixels 8-conectados puede provocar la aparición de inconsistencias. Esta situación se ilustra en el mapa de colores de la figura III.12, donde las letras indican el color de cada componente y los subíndices sirven para localizar pixels concretos. En primer lugar, será necesario impedir el establecimiento de relaciones de vecindad a través de uniones en diagonal por razones evidentes. Este sería el caso cuando los pixels B₁ y B₂ resultaran conectados, lo que impediría la vecindad diagonal entre A₁ y A₂. Otro problema que puede aparecer es el de la autovecindad. Por ejemplo, tal situación se presenta si no se estableciera ninguna unión en diagonal, y los pixels B₁ y B₂, pertenecientes a la misma componente se consideraran vecinos. Esta circunstancia se produce debido a que cuando se analiza la esquina formada por los pixels A₁, A₂, B₁ y B₂ estos dos últimos pixels están rodeados por dos bordes diferentes, que se funden en la siguiente línea. Este problema puede corregirse fácilmente durante el cierre de un borde comprobando la identidad de los vecinos, de manera que cuando un borde resulta ser vecino de sí mismo se elimina el registro de esta vecindad. Pospondremos a la presentación del algoritmo la propuesta de un mecanismo que permita resolver las conexiones en diagonal de esquinas X2.

Una vez presentados los conceptos fundamentales que utiliza el algoritmo, podemos realizar ahora un esbozo del mismo. El algoritmo propuesto para análisis de componentes conexas

Capítulo III

en imágenes multivaluadas opera sobre cada línea en dos fases diferenciadas dedicadas a la detección de esquinas y procesado de las mismas, sintéticamente:

Fase A. Detección de esquinas.

Operación: Se desplaza sobre el mapa de colores una ventana de 2x2 pixels, de izquierda a derecha, para detectar esquinas-C e identificar su tipo. Se registran también para cada esquina su posición (columna) y color, que se toma del color del pixel inferior izquierdo.

Fase B. Procesado de esquinas.

Operaciones: b.1 Las esquinas-C se descomponen en esquinas-B: una esquina-B por cada componente que aparezca en la esquina-C.
b.2 Las relaciones de vecindad se registran en un orden preciso.
b.3 Las esquinas-B se combinan por pares para obtener los elementos que definen el borde de las componentes y los bordes abiertos se manipulan como resultado de procesar los pares de esquinas. Las operaciones posibles son **creación, cierre, desplazamiento y fusión de bordes.**

Para que este esquema pueda ser aplicado por igual a todas las líneas del mapa de colores, y se detecten correctamente los bordes de componentes situados en la primera o en la última línea, así como en los extremos laterales, es necesario añadir alrededor del mapa de colores un marco de un pixel de ancho con un color reservado. Esta secuencia de operaciones (A,B) se repite para cada línea una sola vez, comenzando por la primera línea del mapa de colores y terminando por la línea inferior del marco.

III.2.2.2.3 PROCESADO DE PARES DE ESQUINAS. OPERACIONES SOBRE BORDES.

Consideremos el mapa de colores mostrado en la figura III.8 y realicemos un corte horizontal a través de una línea cualquiera. En las columnas donde confluyen dos componentes aparecen siempre dos bordes, uno ascendente y otro descendente. A cada borde "abierto", esto es, aquellos cuya extensión se prolonga por debajo de la línea, le corresponden exactamente un

borde descendente y otro ascendente. La ordenación de bordes ascendentes y descendentes permanece inalterada de una línea a la siguiente hasta la aparición de una nueva esquina. Cuando se produce este evento, los sucesos posibles son: a) desaparición de un borde ascendente y otro descendente al cerrarse un borde o porque dos bordes, aparentemente independientes, resultan ser parte de un único borde y se fusionan; b) creación de un nuevo par de esquinas al iniciarse un borde o la traslación horizontal de un borde a una nueva columna.

En definitiva, es evidente que la manifestación progresiva de las componentes durante el análisis secuencial línea a línea tiene un correlato en los procesos que se producen sobre el conjunto de bordes ascendentes y descendentes. Por este motivo el algoritmo organiza gran parte de su actividad alrededor de dos vectores de punteros a estructuras de borde de dimensión igual al número de columnas de la imagen más uno. Estos vectores se denominarán *UpEdge* y *DownEdge*, y contendrán la identidad (puntero) del borde ascendente o descendente, respectivamente, situado en la columna indicada por el índice de posición dentro del vector. Así, por ejemplo, para la línea 3 de la figura III.9, los elementos *DownEdge[3]* y *UpEdge[7]* apuntan ambos al borde A. Cuando no existe ningún borde en la columna *i*, los elementos *UpEdge[i]* y *DownEdge[i]* son nulos.

El algoritmo realiza el análisis de conectividades y vecindades mediante la descomposición de las esquinas-C en esquinas binarias y su agrupamiento por pares según lo establecido anteriormente. Los posibles pares de esquinas-B, mostrados en la figura III.10, determinan los procesos que se realizan durante el análisis para obtener el borde externo y posibles bordes internos de cada componente. Los procesos asociados a los posibles pares de esquinas-B son:



Creación de un borde externo. Para ello se introducen apuntadores a este borde en *DownEdge[i]* y *UpEdge[j]*. El color de la componente se registrará a partir del color de la esquina derecha (pixel inferior izquierdo).

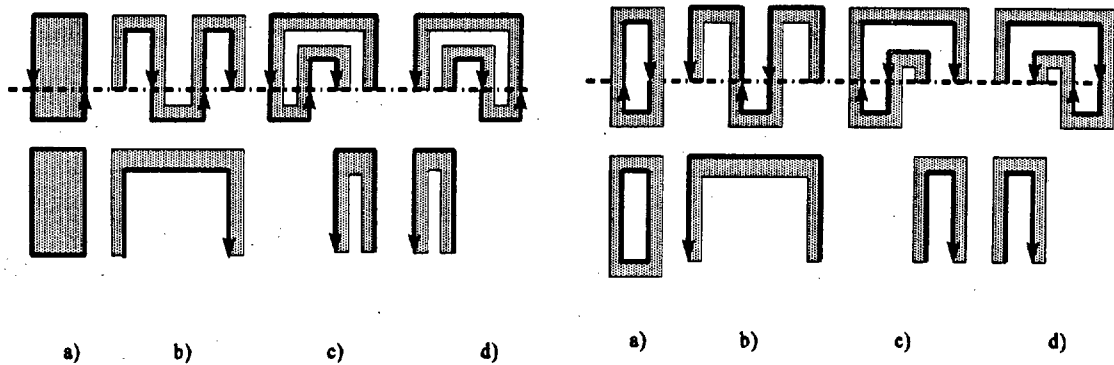


Figura III.13 Posibles situaciones de \lrcorner [Mand-90]. Figura III.14 Posible situaciones de \llcorner [Mand-90].

- \lrcorner : Creación de un borde interno. Se introducen apuntadores a este borde en $DownEdge[j]$ y $UpEdge[i]$. El color de la componente a la que se asocia este borde interno se toma como el color de la esquina izquierda.
- \llcorner : Provoca un desplazamiento horizontal del puntero al borde en $UpEdge$. Esto es, $UpEdge[i] - DownEdge[j]$ y $Edge[j] - NULL$.
- \lrcorner : Provoca un desplazamiento horizontal del puntero al borde en $DownEdge$. Esto es, $DownEdge[i] - DownEdge[j]$ y $DownEdge[j] - NULL$.
- \lrcorner : Provoca un desplazamiento horizontal del puntero al borde en $DownEdge$. Esto es, $DownEdge[j] - DownEdge[i]$ y $DownEdge[i] - NULL$.
- \llcorner : Provoca un desplazamiento horizontal del puntero al borde en $UpEdge$. Esto es, $UpEdge[j] - DownEdge[i]$ y $DownEdge[i] - NULL$.
- \lrcorner : La aparición de este par de esquinas puede presentarse en cualquiera de las situaciones ilustradas en la figura III.13. Las operaciones a realizar en cada caso son:

- a) Cierre del borde externo de una componente.
 - b) Dos bordes internos se fusionan en uno.
 - c,d) Un borde interno resulta ser parte de un borde externo. El borde de interno se fusiona con el externo, realizándose la unión por la cabeza (c) o por la cola (d) del borde externo.
- En todos los casos, $UpEdge[j] \neq NULL$ y $DownEdge[i] \neq NULL$.

↙ : La aparición de este par de esquinas puede presentarse en cualquiera de las situaciones

ilustradas en la figura III.14. Las operaciones a realizar en cada caso son:

- a) Cierre del borde interno de una componente.
- b) Dos bordes externos se fusionan en uno.
- c,d) Un borde externo resulta ser parte de un borde interno. El borde de externo se fusiona con el interno, realizándose la unión por la cabeza (d) o por la cola (c) del borde interno.

En todos los casos, $UpEdge[i] \neq NULL$ y $DownEdge[j] \neq NULL$.

Quando se considera 8-conectividad, la actualización de los vectores $UpEdge$ y $DownEdge$ para los pares de esquinas ↙, ↘, ↙↘, ↘↙, tal y como se ha indicado anteriormente no resulta adecuada. Como ejemplo, consideremos la situación representada en la figura III.15 donde las letras mayúsculas denotan la identidad de los bordes y las letras en minúsculas la localización de la columna. La figura muestra la secuencia de esquinas encontradas al procesar una línea. La esquina en la columna i , de tipo X2, provoca entre otras acciones una conexión en diagonal del borde C al procesar un par ↙ sobre el borde superior; como consecuencia $UpEdge[i]$ se modifica para apuntar a C. En la columna j , la primera acción es procesar el par ↙ (fusionar A y B) donde

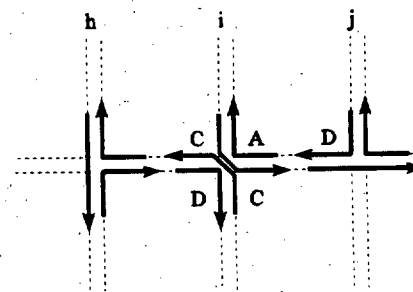


Figura III.15

Capítulo III

será necesario actualizar los punteros en $UpEdge[i]$ y $DownEdge[j]$. Sin embargo, $UpEdge[i]$ no apunta ya a A sino a C por lo que no se debe modificar su valor. En otras palabras, al procesar los pares $\leftarrow \downarrow, \leftarrow \downarrow$ se anulará $UpEdge[i]$ sólo si la identidad del borde apuntado por él y el borde que se procesa son la misma. De manera similar, al procesar los pares $\leftarrow \downarrow, \leftarrow \downarrow$ se establece un criterio análogo pero sobre el puntero $DownEdge[i]$.

línea i:

Crear borde externo entre C y F

$h:(F,i) ; t:(F,i)$

línea j:

Añadir elemento desde h en la cabeza

cabeza: $[CF,ji]$ $h:(C,j)$

Añadir elemento desde t en la cola

cola: $[FE,ij]$ $t:(E,j)$

línea k:

Añadir elemento desde h en la cabeza

cabeza: $[CF,ji][DC,kj]$ $h:(D,k)$

Añadir elemento desde t en la cola

cola: $[EF,jk]$ $[FE,ij]$ $t:(F,k)$

línea m:

Añadir elemento desde h en la cabeza

cabeza: $[CF,ji][DC,kj][CD,mk]$ $h:(C,m)$

Añadir elemento desde t en la cola

cola: $[FC,km]$ $[EF,jk]$ $[FE,ij]$ $t:(C,m)$

Borde completo: inicio: (F,i) recorrido: $[CF,ji][DC,kj][CD,mk][FC,km][EF,jk][FE,ij]$

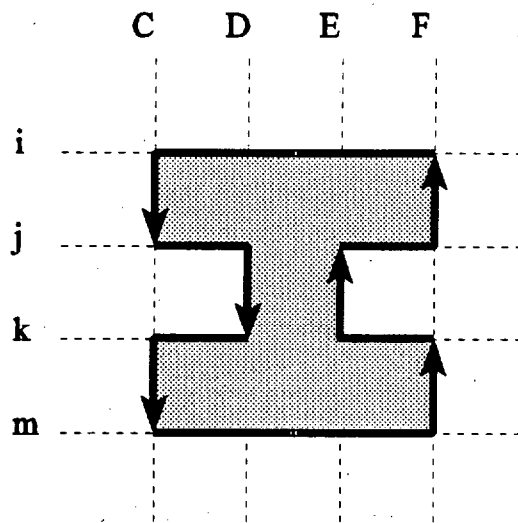


Figura III.16 Evolución de la definición del borde de una componente a través de cuatro líneas.

La descripción de los bordes de cada componente se van construyendo dinámicamente a medida que se procesan los pares de esquinas-B. Los bordes se definen de manera incremental mediante elementos de borde, constituidos por pares de números que miden el desplazamiento horizontal y vertical necesario para alcanzar la siguiente esquina desde una dada. Debido a que el borde se define a partir del recorrido de los extremos ascendente y descendente, es necesario discriminar los elementos de borde que se obtienen de uno y otro extremo. Los elementos de borde que se obtienen de la progresión del extremo descendente se añaden en la cabeza de la lista

de elementos. Mientras que los que se obtienen del recorrido del extremo ascendente se añaden en la cola de la lista. De esta forma, en el momento de cerrar el borde de una componente la lista de elementos de borde está correctamente ordenada, comenzando por el primer desplazamiento a partir del extremo superior derecho (izquierdo) del borde externo (interno).

III.2.2.4 REGISTRO DE VECINDADES.

Una de las características interesantes de este AACC es la forma en la que se registran las relaciones de vecindad entre las componentes, por cuanto se realiza durante el mismo análisis secuencial en el que se obtiene la definición de las mismas. Durante el procesado de las esquinas se construye para cada borde una lista de bordes vecinos. Cuando se cierra el borde de una componente, su lista vecindad contiene la identidad de los bordes vecinos en el orden en el que se encuentran al recorrer dicho borde desde el punto inicial (esquina superior derecha si el borde es exterior). Al igual que ocurría con los elementos de borde, las vecindades pueden establecerse durante el recorrido de cualquiera de los extremos, ascendente o descendente, por lo que para producir una lista ordenada es necesario distinguir las vecindades que deben registrarse por la cabeza o por la cola de la lista de vecindades.

El registro de vecindades simultáneo con la definición de los bordes de las componentes plantea dificultades relacionadas con la identidad de los bordes vecinos. Durante la discusión sobre la obtención de los pares de esquinas-B y los procesos asociados con ellos, se analizó cómo pueden presentarse fenómenos de fusión de bordes. Esto implica que la identidad final de un borde no se establece de forma definitiva hasta el momento en que se cierra. Para evitar esta dificultad es necesario habilitar un mecanismo que permita comunicar la identidad final de un borde a todos los bordes que lo tenían como vecino. Un ejemplo de situación sencilla donde se presenta este

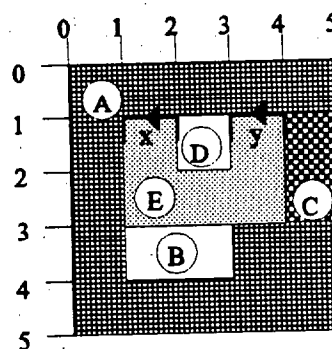


Figura III.17

fenómeno de cambio de identidad de un elemento de borde es la presentada en la figura III.17. En el análisis de la línea 1, los elementos de borde marcados como x e y de la componente E dan origen a dos bordes externos. Al concluir el análisis de la línea 1, x tiene un vecino, la componente A, e y tiene como vecinos a las componentes A y D. Al mismo tiempo x aparece como vecino de D e y como vecino de A y C. Durante el análisis de la línea 2, el borde D se cierra y los bordes x e y se fusionan en un único borde con la identidad de y. En este momento existen bordes (D) que incluyen una relación de vecindad con un borde cuya identidad ha cambiado (x). La verdadera identidad de un borde sólo se conoce con seguridad en el momento en el que se cierra. Cuando en la línea 3, se produzca el cierre del borde de la componente E, será posible colocar su verdadera identidad en las listas de vecinos de bordes adyacentes. Nótese que algunos de estos bordes pueden haberse cerrado con anterioridad, tal es el caso de D, o pueden estar aún abiertos como A, B y C.

LISTAS DE OBJETOS

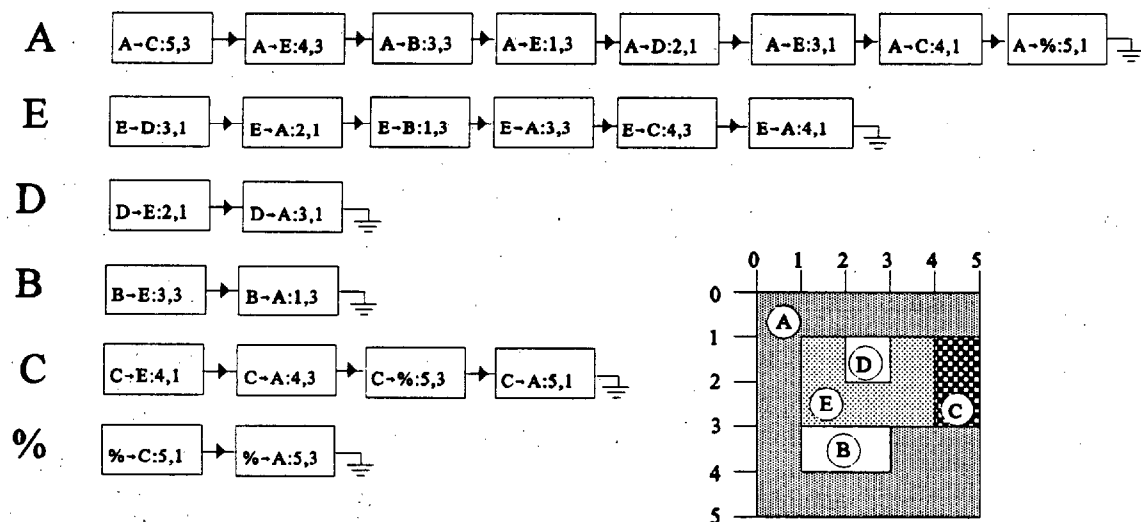


Figura III.18 Lista de objetos.

Una solución al problema descrito es la adoptada en [Mand-90] y que también se ha conservado en esta implementación. Consiste en mantener por cada borde abierto o activo dos listas encadenadas llamadas **listas de objetos** y **de sujetos** respectivamente. La primera contiene la lista de bordes que "son" vecinos del borde en cuestión. La segunda es la lista de bordes que tienen al borde en cuestión como vecino, esto es, en sus listas de objetos. Para fijar ideas, consideremos de nuevo el borde de la componente E en la figura III.17. La lista de objetos de este borde se compondrá simplemente de los registros de vecindad de la componente E con los bordes vecinos, que en el momento de cerrar el borde, sería la mostrada en la figura III.18. Las coordenadas que se incluyen con cada registro de vecindad son las del punto de contacto entre ambos bordes al recorrer el borde al que pertenece la lista de objetos con el sentido adecuado. Tal y como muestra la lista, un borde puede aparecer en la lista de objetos de otro borde repetidas veces aunque nunca de manera consecutiva. Es necesario advertir que en el ejemplo aludido no es necesario distinguir entre componente y borde, puesto que no aparecen componentes con bordes interiores. Si este no fuese el caso, habría que indicar en estas listas la identidad del borde y la componente al que pertenece. La lista de sujetos de la componente E se detalla en la figura III.19 y contiene los bordes que tienen a esta componente como vecino. En estas figuras el símbolo % representa al borde interior equivalente al marco del mapa de colores. De la comparación entre las listas de objetos y sujetos de cualquiera de las componentes es inmediato extraer algunas observaciones. La primera es que los registros de vecindad contenidos en la figura III.19 y en la figura III.18 son exactamente las mismas aunque enlazadas de diferente forma. Esto quiere decir que a la hora de confeccionar estas listas no es necesario crear dos registros de vecindad, uno para la lista de objetos y otro para la lista sujetos, sino que basta con crear un único registro y enlazarlo de manera diferenciada en cada lista. Nótese también que las relaciones de vecindad entre dos bordes vecinos, aunque recíprocas, no son simétricas en tanto que se establecen en coordenadas diferentes. Por último, es evidente que las listas de objetos y sujetos no existen simultáneamente para todas las componentes tal y como se detallan en las figuras.

El registro de una vecindad supone la creación dinámica de un elemento del tipo *estructura descripción de vecindad*, que consta de los siguientes campos:

```
struct neighborhood_descrip {
    short    col, row;           // Coordenadas del punto de contacto.
    char     StateOfObj;        // Estado del borde objeto ABIERTO | CERRADO
    char     StateOfSubj;       // Estado del borde sujeto ABIERTO | CERRADO
    int      ObjIdent;          // Identidad del borde objeto (*).
    int      ObjColor;          // Color del borde objeto (*).
    long     LocoOfObjRef;       // Dirección donde se depositará la identidad final
                                // del objeto cuando se cierre (*).

    struct neighborhood_descrip * NextObj; // Puntero al siguiente elemento en la lista
                                           // de objetos.

    struct neighborhood_descrip * NextSubj; // Puntero al siguiente elemento en la lista
                                           // de sujetos.
}
```

Los registros de vecindad no se liberan hasta que los dos bordes involucrados, como sujeto y objeto, no se cierran. Cuando se presenta esta circunstancia, el registro de vecindad se libera y las listas de sujetos y de objetos en las que participa se reconectan. Para clarificar este particular consideremos, por ejemplo, el registro de la vecindad "B es vecino de A en el punto (3,3)" que involucra dos pasos. Primeramente se inicializa un elemento del tipo *estructura de descripción de vecindad* y se incluye en la lista de objetos del borde sujeto (A). A continuación, este mismo elemento se incluye en la lista de sujetos del borde objeto (B). Supongamos ahora que el borde B se cierra antes que el borde A. Al cerrar el borde B es necesario realizar dos tareas: comunicar la identidad final y color a los bordes de la lista de sujetos de B y especificar una dirección donde los bordes de la lista de objetos de B que aún estén abiertos puedan comunicar sus identidades finales cuando se cierren. Para realizar la primera de ellas, se recorre la lista de sujetos del borde B para comunicar su identidad final y color. La comunicación de esta información se realiza de forma diferente según el estado, campo *StateOfSubj*, de los bordes de la lista de sujetos. Si los bordes sujetos están cerrados, las señas de identidad del borde objeto se salvan en la dirección indicada por *LocoOfObjRef*, se vuelve a conectar la lista de sujetos y se libera el espacio en memoria ocupado por el elemento de vecindad. En caso contrario, se actualiza

el campo *StateOfObj* a CERRADO y la identidad y color del objeto se escriben en los campos *ObjIdent* y *ObjColor*. A continuación es necesario salvar la lista de vecindades de B, esto es su lista de objetos. Para ello se recorre esta lista y se comprueba cuál es el estado del objeto (*StateOfObj*). Si el borde objeto está aún abierto, *StateOfSubj* se cambia a CERRADO y en *LocoOfObjRef* se escribe la dirección donde se comunicará la identidad y el color del borde objeto cuando se cierre. Si el borde objeto está cerrado su identidad se conoce, la información contenida en el elemento de vecindad se salva, la lista de objetos se vuelve a conectar y se libera el espacio en memoria ocupado por el elemento de vecindad. Para eliminar la posibilidad de que se registren **autovecindades** cuando se utiliza 8-vecindad según lo comentado anteriormente, antes de salvar los elementos contenidos en la lista de objetos se comprueba que las identidades del objeto y del sujeto son diferentes.

Las relaciones de vecindad se determinan durante el procesamiento de las esquinas-C, por ello con frecuencia es necesario especificar una relación entre dos bordes con anterioridad a la

LISTAS DE SUJETOS

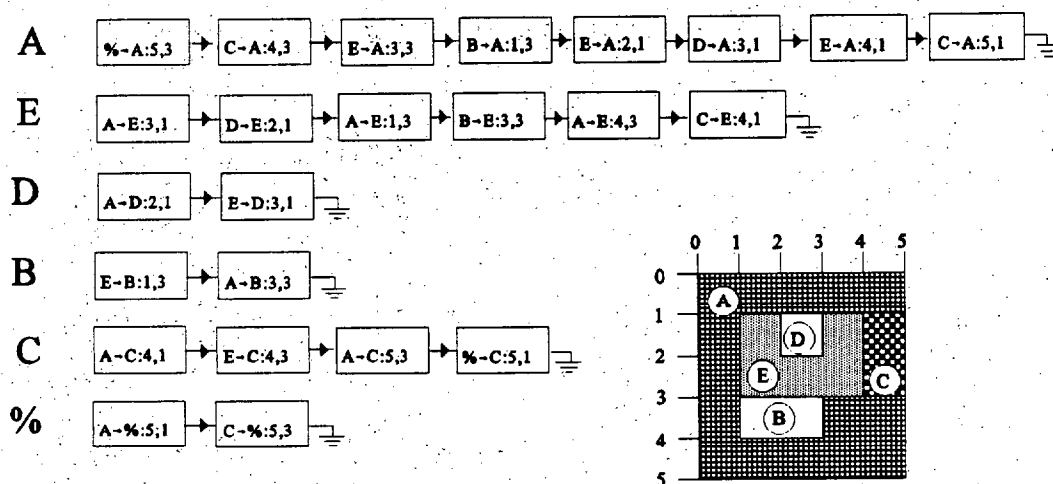


Figura III.19 Listas de sujetos.

creación de uno de ellos, o de ambos. Por esta razón se emplean dos bordes "auxiliares" - tres en el caso de utilizar 8-vecindad - (denominados dummy1, dummy2 y dummydiag en el pseudocódigo que se presenta en el Apéndice A), los cuales sirven para identificar bordes aún no creados en las relaciones de vecindad. Las vecindades acumuladas en estos bordes auxiliares se transfieren a las listas de vecindad adecuadas una vez que los bordes correspondientes han sido creados.

III.2.2.2.5 EXTENSIONES AL ALGORITMO ORIGINAL.

Describiremos a continuación algunas extensiones realizadas sobre el algoritmo original. Algunas de éstas son meras clarificaciones a puntos que, a nuestro juicio, no quedaban suficientemente desarrollados en la propuesta original del algoritmo; o son extensiones útiles como la obtención de la ventana circunscrita a cada componente o la obtención de los pixels que integran una componente. Otras extensiones, por el contrario, como la utilización de 8-conectividad, plantean algunos problemas adicionales que requieren una exposición más detallada.

a) Obtención de la ventana circunscrita a cada componente.

La adaptación del algoritmo descrito para realizar el análisis de componentes conectadas sobre una ventana rectangular del mapa de colores resulta de mucha utilidad. Esta posibilidad es muy interesante dentro del marco de actuación del Procesador de Segmentos pues, como ya se explicó en el capítulo II, durante la elaboración de la partición final de una imagen pueden llevarse cabo acciones de fusión o de división de segmentos. Cuando estas acciones tienen lugar es imprescindible actualizar la información (bordes, vecindades, etc...) relativa a los segmentos sobre los que se producido la acción y, probablemente, las listas de vecindad de los segmentos vecinos. En otras palabras, es necesario efectuar un nuevo análisis de componentes conexas localizado sobre los segmentos que han sido modificados por la acción de control y su entorno de vecindad. Con esta perspectiva es evidente que la obtención de las ventanas circunscritas a los bordes de cada componente es muy interesante, fundamentalmente por razones de eficiencia, aunque la ventana circunscrita puede por sí misma servir como elemento descriptivo de la componente. La obtención de la ventana asociada con cada uno de los bordes de cada componente no plantea ninguna dificultad dentro del esquema propuesto y se tratará siempre de

una ventana rectangular orientada paralelamente al marco de la imagen. El lado superior de la ventana se fija en el momento en que se crea el borde y se actualiza siempre que se producen fusiones de bordes. El lado inferior se determina en el momento del cierre del borde. La situación de los lados verticales se fija cuando se crea el borde y se actualiza siempre que se producen traslaciones de los extremos ascendente o descendente (véase la parte del pseudocódigo donde se procesan los pares de esquinas-B) o se efectúan fusiones entre bordes. En el momento de cerrar el borde, los lados de la ventana no coincidentes con el marco de la imagen se trasladan un pixel hacia el exterior para que los límites de la componente queden en el interior de la ventana, rodeados por al menos un pixel de las componentes vecinas. De esta manera, los puntos de vecindad de la componente quedan incluidos en la ventana.

b) Obtención de las coordenadas de los pixels de cada componente.


La obtención de las coordenadas de los pixels que constituyen cada componente es otra de las tareas que encajan de manera sencilla dentro de la estructura de este AACC y que resulta de interés. Para ello basta añadir el siguiente código dentro del bucle que analiza todas las líneas del mapa de colores, a continuación de haber procesado las esquinas detectadas en la línea activa, de manera que los vectores *UpEdge* y *DownEdge* están actualizados:

```
col_start = primera_columna_del_mapa;
col_end = col_start;
do {
    down_border = DownEdge[col_start];
    while (UpEdge[++col_end] == NULL);
    AppendAreaElem(down_border, col_start, (col_end - col_start));
    col_start = col_end;
} while (col_start < última_columna_del_mapa);
```

La función *AppendAreaElem* añade un "elemento de área" en el borde apuntado por *down_border*. Un elemento de área se compone de tres números: columna, fila y longitud de recorrido (*col_end - col_start*) o número de pixels consecutivos que pertenecen a la misma componente. Evidentemente, lo que se obtiene de forma directa con la propuesta que aquí se hace, es la codificación del área de las componentes en forma de código de recorrido sobre cada línea. El procedimiento de obtención es aplicable tanto para bordes externos como para bordes





internos. El área de una componente se recuperará entonces de la unión de todos los elementos de área de su borde externo y de sus posibles bordes internos. Ésto plantea la cuestión adicional de relacionar los bordes internos con el correspondiente borde externo o lo que es igual con una componente del mapa de colores.

c) Obtención del borde externo correspondiente a un borde interno.

La idea básica es que el borde externo de una componente y los posibles bordes internos comparten el mismo color y que el borde externo se cerrará en algún momento con posterioridad a los bordes internos. El procedimiento seguido es análogo al sugerido en [Mand-90] y consiste en poner la referencia del borde interno que se cierra en el primer borde externo del mismo color encontrado en el vector *UpEdge*, descrito en el apartado III.2.2.2.3, a partir de la posición ocupada por la esquina derecha de  hacia la derecha. Puede ocurrir que este borde resulte ser un falso borde externo, en cuyo caso se deberá repetir la misma operación para localizar el borde externo de la componente cuando se cierre este nuevo borde interno. En algún momento se encontrará el borde externo de la componente. Éste, al cerrarse, contendrá la lista con las referencias de todos sus bordes internos, a los que comunicará su identidad, y los correspondientes elementos de área.

d) La relación de vecindad entre un borde interno y su único vecino interior.

Como se ha explicado con anterioridad, el registro de vecindades se realiza en los puntos donde se produce "un cambio de vecino", o lo que es igual, donde confluyen al menos tres bordes. Cuando una componente es la única vecina interior de otra que la rodea completamente, no existen estos puntos de cambio de vecindad al recorrer el borde externo de la componente incluida o el borde interno de la componente inclusora. De no habilitarse un mecanismo específico que detecte y corrija estas situaciones, estos bordes no se van a relacionar como vecinos. Al respecto se propone una solución que consiste en comprobar la lista de objetos de un borde externo en aquellas situaciones en las que se puede producir secuencialmente el cierre de un borde externo y, a continuación, el cierre de un borde interno. En estas situaciones, la ausencia de registros de vecindad en lista de objetos de cualquiera de los bordes será el indicativo de que el borde interno

tiene como único vecino al borde externo y viceversa. Estas circunstancias pueden aparecer al procesar esquinas-C del tipo  o del tipo . En ambos casos, si el par de esquinas-B situado en el borde superior es  (lo que supondrá el cierre de un borde externo), el situado sobre el borde inferior es  (lo que supondrá el cierre de un borde interno) y la lista de objetos de cualquiera de estos bordes está vacía, entonces se registra una relación de vecindad entre ambos. El punto de vecindad se establece arbitrariamente sobre la esquina inferior derecha común a ambos bordes.

e) Resolución uniones diagonales en esquinas-C del tipo X2.

En su versión original, el algoritmo utilizaba sólo 4-conectividad para definir la extensión de las componentes y todas las esquinas X (X2, X3 y X4) se trataban como esquinas tipo X4, evitando así las uniones en diagonal. La consideración de la 8-conectividad es problemática cuando se consideran esquinas X2. Ante una esquina de este tipo no es posible escoger una dirección para realizar la unión en diagonal exclusivamente desde los colores de los pixels contenidos en la ventana 2x2. Una solución a esta dificultad, adoptada en el caso de imágenes binarias [Pavl-82], consiste normalmente en definir el "fondo" como 4-conectado y el "objeto" como 8-conectado, de manera que sólo pueden conectarse diagonalmente los pixels etiquetados como "objeto". Evidentemente esta solución no es aplicable directamente en imágenes multivaluadas, pero supone una idea interesante que puede ser adaptada en ciertas situaciones. A continuación desarrollaremos dos propuestas de solución a este problema.

La primera, que llamaremos **solución_A**, no es una solución general en el sentido que no garantiza el que todas las uniones en diagonal se resuelvan adecuadamente. Sin embargo, consideramos interesante presentarla pues se basa exclusivamente en la información topológica extraída del mapa de colores en el momento de procesar la esquina tipo X2 y es suficiente para tratar algunos problemas. Concretamente, esta solución está pensada específicamente para tratar los casos en los que componentes 8-conectadas de tamaño reducido están incluidas en el interior

de otra componente más extensa. En estas situaciones, es posible resolver conexiones diagonales ambiguas utilizando el siguiente algoritmo:












Definiciones:

UpBorder: identidad del borde ascendente en la columna de la esquina.

DownBorder: identidad del borde descendente en la columna de la esquina.

CornerType: clasificación del esquina.

CornerColor: color de la esquina: color del pixel inferior izquierdo en la ventana 2x2.

```
if (CornerType == X2) {
    if (UpBorder y DownBorder son del mismo tipo (v.g. externos)) {
        LastCornerType = tipo de la última esquina detectada en la línea;
        switch (LastCornerType) {
             :      CornerType =  ;
            break;
             :      CornerType =  ;
            break;
             :      CornerType =  ;
            break;
             :      CornerType =  ;
            break;
            default:      CornerType =  ;
        }
    }
    else {
        Selecciona el borde externo;
        if ( el color del borde externo == CornerColor)
            CornerType =  ;
        else
            CornerType =  ;
    }
}
```

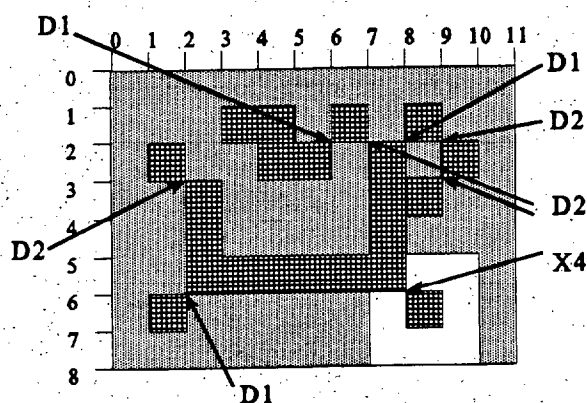




Figura III.20 Resolución de esquinas X2 mediante el algoritmo de la solución_A.

El algoritmo propuesto considera básicamente dos situaciones. La primera de ellas se presenta cuando los bordes ascendente y descendente situados en la columna de la esquina X2 son del mismo tipo, es decir, externos. En este caso se considera el tipo de la última esquina-C detectada en la línea. Si éste no es de los tipos considerados, la esquina X2 se tratará como una esquina X4 evitando así cualquier conexión en diagonal. Nótese que esta parte del algoritmo opera durante la detección de las esquinas-C y antes de que ninguna de las esquinas detectadas en la línea se procese. La segunda situación se presenta cuando los bordes ascendente y descendente son uno interno y el otro externo. En este caso, se resuelve la esquina X2 para conectar diagonalmente los pixels del borde externo. La figura III.20 muestra la actuación de este algoritmo sobre un mapa de tres colores. D1 y D2 indican que la esquina X2 se ha resuelto, respectivamente como  o como . La mayoría de las situaciones involucran a un borde externo y otro externo, y el algoritmo tiende a conectar en diagonal el borde externo. Nótese la ambigüedad inherente a situaciones como las que se dan en las esquinas situadas en las posiciones (8,6) y (9,3) de la citada figura.
















En general, no es posible resolver estas situaciones de ambigüedad de manera estable si no se asocia alguna semántica al coloreado de los pixels, en el caso de que ésta exista o sea posible definirla. Esta consideración nos lleva a proponer la segunda solución, que llamaremos **solución_B**, aplicable en aquellos problemas donde el etiquetado de los pixels obedece a una

cierta semántica. Esta solución se inspira en la solución del caso binario, donde se resuelve las conexiones diagonales ambiguas definiendo el color del fondo como 4-conectado y el color del objeto o primer plano como 8-conectado. La idea es muy simple y consiste en establecer relaciones de "fondo-objeto" entre todos los pares de posibles colores. La conexión diagonal de una esquina X2 se establece entre los pixels cuyo color se corresponde con el color del "objeto" para el par de colores presentes en la ventana 2x2.



La implementación de este AACC utilizada en el **Procesador de Segmentos** incorpora la **solución_B** como fórmula básica para resolver las esquinas X2, aunque con algunas particularidades. Como ya se explicó, el módulo de presegmentación basado en la transformada Watershed produce una partición de la imagen en términos de **prototipos**, en cuya declaración se puede especificar si éste se considera 4 u 8-conectado. También, el objeto **S_MakePartition** puede contener una lista de precedencias en términos de nombres de prototipo para resolver conexiones diagonales en esquinas X2. En cuanto al uso que se hace del AACC en el Procesador de Segmentos hay que distinguir dos momentos diferenciados: el primero, que se produce sólo en una ocasión cuando el AACC analiza el mapa de colores completo al finalizar la acción del presegmentador; y el segundo, que comprende la aplicación del algoritmo sobre ventanas de la partición cada vez que, tras una acción de unión o división de segmentos, es necesario restaurar la información sobre vecindades y conectividades de la partición. En cada una de estas fases, la detección del tipo de esquinas-C es diferente.

En la primera aplicación del AACC se utiliza información sobre la conectividad definida para los prototipos y, si es necesario, la especificación de precedencias diagonales. El siguiente pseudocódigo resume la parte del algoritmo de detección de esquinas dedicado a la asignación de un tipo a las esquinas X.

```

switch(Window2x2) {
case  : CornerType =  ;
break;
case  : if (el prototipo del pixel inferior izquierdo es 8-conectado)
        CornerType =  ;
    else
        CornerType =  ;
break;
case  : if (el prototipo del pixel superior izquierdo es 8-conectado)
        CornerType =  ;
    else
        CornerType =  ;
break;
case  : if (el prototipo del pixel inferior izquierdo NO es 8-conectado) {
        if (el prototipo del pixel superior izquierdo NO es 8-conectado)
            CornerType =  ;
        else
            CornerType =  ;
    }
    else {
        if (el prototipo del pixel superior izquierdo NO es 8-conectado)
            CornerType =  ;
        else {
            Obtener la precedencia asignada a este par de prototipos;
            if (la precedencia de ambos prototipos es la misma)
                CornerType =  ;
            else if (la precedencia favorece el prototipo del pixel superior izquierdo)
                CornerType =  ;
            else
                CornerType =  ;
        }
    }
break;
}
}

```

Un resultado muy importante de la fase de detección de esquinas-C en esta aplicación del algoritmo es una matriz, denominada "matriz de conexión", de las mismas dimensiones del mapa de colores (no incluido en el pseudocódigo anterior). En ella, el elemento (i,j) registra el tipo de enlace diagonal que se establecerá en la posición equivalente de la partición durante el análisis de conectividades. Evidentemente, las uniones diagonales sólo serán posibles en las posiciones ocupadas por esquinas  y . En las sucesivas actuaciones del AACC sobre la partición, se actualiza la información contenida en la matriz como consecuencia de las acciones que se realicen sobre los segmentos; pero fundamentalmente, la utilización de la matriz de conexiones como referencia permite resolver posibles conexiones en diagonal preservando la topología ya establecida. El algoritmo dedicado a la detección de esquinas X en estas ocasiones se basa fundamentalmente en la matriz de conexiones. En esta fase de detección de esquinas del AACC no se introducen nuevas conexiones diagonales, las únicas modificaciones que se realizan sobre la información contenida en la matriz son la eliminación de conexiones diagonales que han sido destruidas en el transcurso de alguna acción de división/unión de segmentos. Estas acciones pueden producir la creación de nuevas conexiones diagonales, como se observa en el ejemplo de

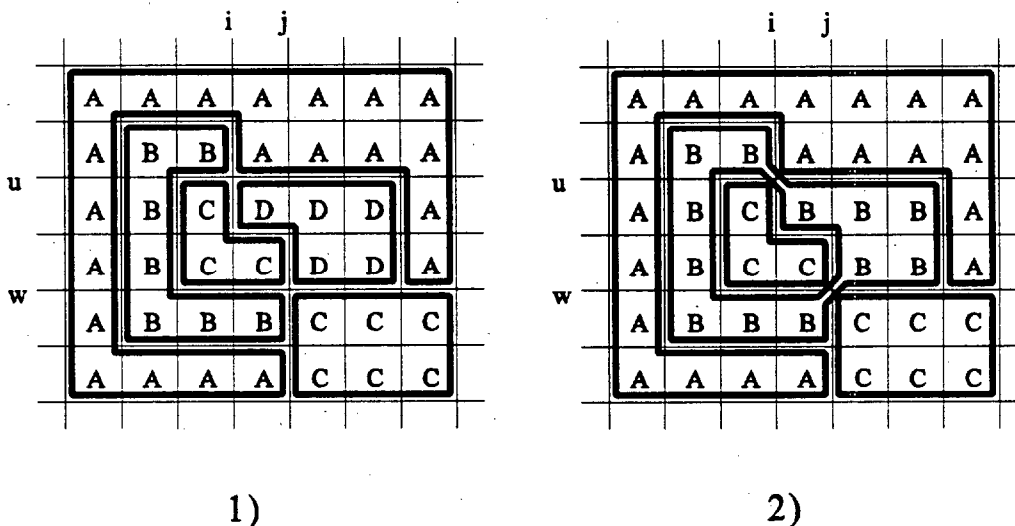


Figura III.21 Modificación de las conexiones diagonales por la fusión de los segmentos B y D.

la figura III.21. La creación de estas nuevas conexiones diagonales requieren la modificación de los contenidos de la matriz de conexiones antes del análisis de conectividad.

Para ilustrar estos comentarios considérese el caso mostrado en la figura III.21. Tras la primera actuación del ACCC, el mapa está constituido por cinco componentes cuyo color se representa por letras. Las componentes de color C se han definido como 4-conectadas, siendo el resto 8-conectadas. En el análisis de vecindad, que no tiene ninguna consecuencia apreciable en la figura, se ha empleado 8-vecindad. Como ilustra el apartado 1) de la figura, en la partición resultante no se han establecido conexiones diagonales, ya que la definición explícita de las componentes de color C como 4-conectadas ha evitado la conexión diagonal en la posición (j,w) . Supongamos ahora que en algún momento se produce la fusión de la componente de color B y con la de color D (apartado 2) de la figura). Esto provoca la creación de dos conexiones en diagonal en las posiciones (i,u) y (j,w) y la actualización de la matriz de conexiones diagonales con anterioridad al análisis del AACC. Estas conexiones diagonales suponen una modificación, no sólo en la definición de la nueva componente, sino también en sus relaciones de vecindad y en las de sus vecinos. Concretamente, y entre otras modificaciones, se inhibe la vecindad entre las componentes de color A y de color C en la posición (i,u) , y entre las componentes de color C en (j,w) . La actuación del AACC sobre la ventana definida por la extensión de las componentes fusionadas debe servir para actualizar esta información. Durante la fase de detección de esquinas-C, el AACC va a encontrar en el punto (j,w) una esquina de tipo X2 al recorrer el mapa de colores con una ventana 2×2 . A diferencia de lo que ocurría durante la primera aplicación del AACC, ahora las esquinas X2 y X3 que se detecten van a producir conexiones en diagonal o no (se consideran entonces como esquinas X4) dependiendo de la información contenida en la matriz de conexiones diagonales. En el ejemplo de la figura III.21, la consulta a la matriz de conexiones resuelve la esquina X2 que se encuentra en la posición (j,w) . La conexión diagonal entre los pixels de color C debe evitarse pues ya existe otra conexión en sentido opuesto entre los pixels de color B. El pseudocódigo que sigue resume la parte del algoritmo de detección de esquinas-C dedicado a la consideración de las esquinas de tipo X.













Definiciones:

DiagLink(columna, fila) : contenido de la matriz de conexiones diagonales en la posición (columna, fila). Cuando DiagLink(col,row) es igual a \emptyset indica que no existen conexiones diagonales en esa posición.

col,row : indican la posición actual de la ventana y por tanto de la esquina.

```
switch(Window2x2) {
```








```

case  : CornerType =  ;
    if (DiagLink(col,row) ==  $\emptyset$  )
        break;
    DiagLink(col,row) =  $\emptyset$  ;
    break;
case  : if (DiagLink(col,row) ==  $\emptyset$  )
        CornerType =  ;
    else if (DiagLink(col,row) ==  ) {
        DiagLink(col,row) =  $\emptyset$  ;
        CornerType =  ;
    }
    else
        CornerType =  ;
    break;
case  : if (DiagLink(col,row) ==  $\emptyset$  )
        CornerType =  ;
    else if (DiagLink(col,row) ==  ) {
        DiagLink(col,row) =  $\emptyset$  ;
        CornerType =  ;
    }
    else
        CornerType =  ;
    break;





```



```

case  : switch (DiagLink(row,col)) {
    case  : CornerType =  ; break;
    case  : CornerType =  ; break;
    case  : CornerType =  ; break;
}
}

```

Se observa cómo el algoritmo descrito evita la introducción de nuevas conexiones diagonales durante la fase de detección de esquinas y tan sólo elimina conexiones diagonales que han quedado obsoletas, debido a la dinámica de unión/división de los segmentos. Los procesos encargados de controlar las uniones entre segmentos son los que modifican la matriz de conexiones cuando es necesario introducir en algún punto nuevas conexiones diagonales. Finalmente, resulta interesante analizar el caso de una posterior división de la unión B-D en los segmentos originales B y D. Consideremos cómo trataría el algoritmo descrito las esquinas que se detectan en las posiciones (i,u) y (j,w). En (i,u) aparece una esquina X4 y el contenido de DiagLink se cambia de  a . En (j,w), en cambio, aparece una esquina X3, siendo el contenido de DiagLink en esa posición . Según la especificación dada para este caso, la esquina se trata como una esquina X4 y DiagLink se cambia a . De esta manera el uso de la matriz de conexiones evita en la posición (j,w) un enlace diagonal entre los pixels de igual color (C) que resultaría incorrecto al tratarse de dos componentes independientes.

III.2.2.2.6 ANÁLISIS EXPERIMENTAL DEL AACC PROPUESTO.

El algoritmo de análisis de componentes conectadas descrito en los epígrafes anteriores ha sido estudiado desde un punto de vista experimental para analizar su comportamiento en función del tamaño de la imagen, su complejidad (dependiente del número de esquinas y componentes de la imagen) y su naturaleza (binarias o multivaluadas). Además, el algoritmo propuesto se ha comparado con el algoritmo de etiquetado de componentes conectadas (AECC) descrito en [Yang-92]. Después de una revisión sobre la literatura publicada [Lumi-83a][Caps-84][Same-86] a este respecto, y en base a los tiempos de cómputo publicados, se ha estimado este

algoritmo como el AECC no-paralelo más eficiente para imágenes bidimensionales que tiene como característica interesante la de haber promovido una implementación hardware [Yang-88]. Sin embargo, la comparación entre ambos algoritmos exige establecer una serie de consideraciones.

En primer lugar, es necesario plantear una distinción entre *un algoritmo de análisis de componentes conectadas (AACC)* y *un algoritmo de etiquetado de componentes conectadas (AECC)*. Ambos extraen las componentes conectadas presentes en un mapa de colores, pero un AACC, además, extrae información topológica y relacional detallada sobre las componentes (bordes, area, vecinos, etc...), tal y como se ha visto a lo largo de las secciones anteriores. Los AECC suelen generar como resultado otro mapa donde se asocia a cada pixel un índice que identifica la componente a la que pertenece, sobre el que es necesario aplicar posteriormente algún algoritmo de seguimiento de bordes para extraer los bordes de las componentes e información sobre vecindades. Es evidente, por tanto, que un AECC produce sólo una fracción de la información obtenida de un AACC, lo que hace que la comparación de dos de estos algoritmos deba considerarse siempre como muy desfavorable para el AACC. La segunda cuestión que hace difícil la comparación entre el AACC propuesto y el algoritmo de Yang, reside en las diferentes capacidades de ambos respecto al tipo de imágenes que pueden procesar. Al igual que la mayoría de los AECC publicados, el algoritmo de Yang sólo trata imágenes binarias y 4-conectadas, mientras que al AACC propuesto puede trabajar con imágenes multivaluadas que pueden ser 4 u 8 conectadas. Para "amortiguar" esta última diferencia, en la comparación entre ambos algoritmos se han utilizado dos series de imágenes binarias que contenían, respectivamente, imágenes de 256x256 y de 512x512 pixels. Ambas series estaban integradas por imágenes cuya complejidad, entendida como el número de esquinas y contornos de componentes, cubría un rango aproximado de cuatro órdenes de magnitud. En este primer análisis, se ajustaron los parámetros del AACC propuesto para que el análisis de conectividades y vecindades fuera 4-conectado y no se incluyera el área de las componentes. Los tiempos que se presentan han sido medidos con el comando *time* de UNIX sobre una estación de trabajo HP 9000 serie 700, modelo 720. En las gráficas se relaciona el tiempo tardado en analizar la imagen, eje de ordenadas, en relación al logaritmo en base diez del número de contornos detectados en la imagen. Los tiempos

que se muestran en todas las gráficas representan el resultado medio de seis análisis de la misma imagen. Por el método empleado para la generación de estas series de imágenes, el número de esquinas y el número de contornos presentes en una imagen mantenía una relación aproximadamente constante por lo que en las gráficas se ha omitido la representación en función del número de esquinas.

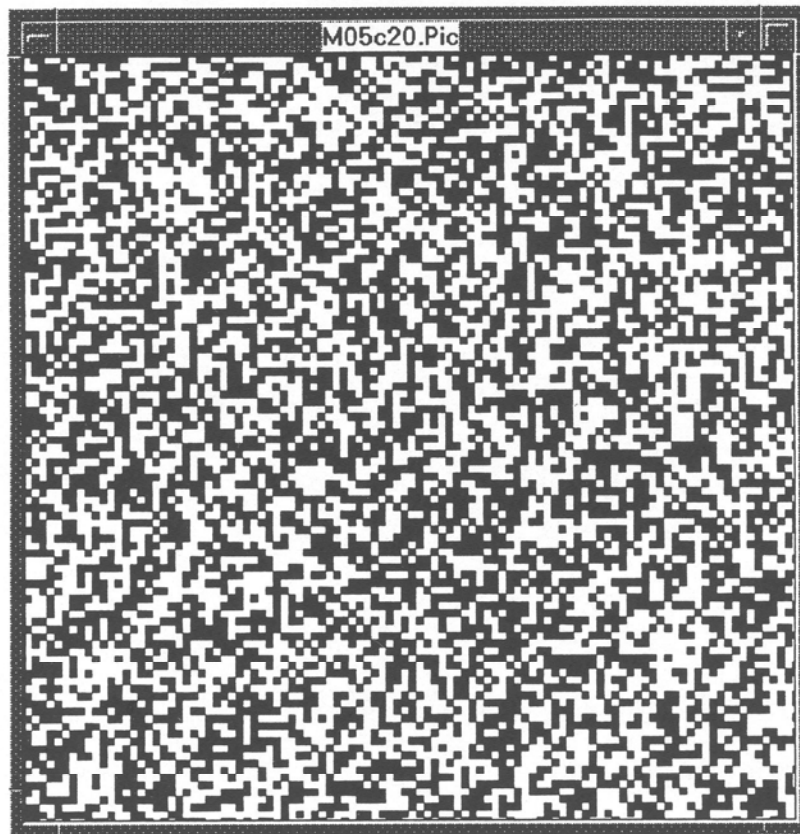


Figura III.22 Imagen binaria (512x512, 6700 esquinas, 1502 contornos) utilizada en los experimentos.

Yang vs. AACC.

Las figuras III.23-26 ilustran el resultado del análisis comparativo entre ambos algoritmos. Las figuras III.23 y III.24 muestran la dependencia del algoritmo de Yang en función del tamaño de la imagen y de la complejidad de la misma. Como puede apreciarse, este algoritmo depende de manera aproximadamente constante del tamaño de la imagen y presenta un espectacular comportamiento, aproximadamente lineal, con el número de contornos de la imagen. Por contra, el AACC propuesto no depende del tamaño de la imagen, pero sí de la complejidad de la misma. Nótese además la diferente escala en el eje de tiempos de ambas gráficas. Las figuras III.25 y III.26 muestran la superposición de las curvas obtenidas por ambos algoritmos en cada una de la series. Puede apreciarse como el AACC es más rápido que el algoritmo de Yang hasta un cierto punto donde ambas curvas se cruzan. En el caso de la serie de 512x512, este punto se sitúa aproximadamente en los 2000 contornos (10000 esquinas) y 1.2 segundos. La figura III.22 muestra una de las figuras empleadas en los experimentos (512x512, 6700 esquinas y 1502 contornos) y puede servir para obtener una impresión "visual" de la complejidad de una imagen con este número de esquinas y contornos.

Comparativa del AACC utilizando diferentes configuraciones.

Una segunda fase de este estudio experimental se dedicó a investigar el rendimiento del algoritmo propuesto en función de diferentes configuraciones. Los resultados de este estudio se muestran en la figura III.27 donde se incluyen como referencia los resultados obtenidos de la actuación del algoritmo de Yang sobre la serie de 512x512. Las configuraciones del AACC estudiadas fueron: a) extracción del área de las componentes, 8-conectividad y 8-vecindad, b) sin extracción del área de las componentes, 4-conectividad y 4-vecindad y c) la misma configuración de b) pero inhibiendo la escritura en disco de los resultados del análisis. De la inspección de las curvas de la figura III.27 es posible extraer algunas conclusiones. En primer lugar el algoritmo mantiene un rendimiento aproximadamente constante para las configuraciones a) y b), excepto en el caso de imágenes con un gran número de contornos. Este menor rendimiento se debe en parte al mayor coste computacional del análisis del algoritmo en la configuración a), pero también influye el que el volumen de información generado es superior, información que debe ser transferida al disco. Como ejemplo del volumen que puede llegar a alcanzar, considérese que el

análisis de una imagen de 233465 esquinas y 133059 contornos supone la escritura en disco de 15Mb. La influencia de este aspecto sobre el rendimiento del algoritmo queda patente en la curva correspondiente a la configuración c), en la que se ha inhibido la escritura en disco. Puede apreciarse como la reducción del tiempo empleado por el algoritmo es importante, siendo aproximadamente de la mitad cuando el número de contornos es relativamente elevado.

La figura III.28 muestra los resultados de aplicar el algoritmo en la configuración b) del apartado anterior sobre una serie de imágenes binarias y sobre otra de imágenes multivaluadas (10 colores diferentes). Ambas curvas están prácticamente superpuestas, lo que demuestra que el algoritmo no es dependiente del número de colores presentes en la imagen.

Finalmente, la gráfica de la figura III.29 muestra los porcentajes de tiempo consumidos por las tres partes del algoritmo que ocupan la mayor parte del tiempo de ejecución, función del número de contornos. Éstas son la detección de esquinas, la reserva de memoria dinámica (malloc) y el tiempo dedicado a la escritura en disco. Estas medidas representan los valores medios de seis análisis obtenidas con el comando *prof* de Unix. Para este análisis se utilizó el algoritmo en la configuración b). Los resultados muestran como la parte del código dedicada a la detección de esquinas ocupa la mayor parte del tiempo mientras que el número de contornos es inferior a 125, aproximadamente. A partir de este punto, el algoritmo consume la mayor parte del tiempo en llamadas relacionadas con la escritura en el disco (write, fwrite, seek y fseek de C) de la información generada. Estos resultados inducen a pensar que el rendimiento del algoritmo de análisis de componentes conectadas podría mejorar, fundamentalmente cuando se emplee sobre imágenes con un gran número de contornos, si se utilizan técnicas de programación que eviten la escritura de los resultados en disco. El Apéndice A contiene el pseudocódigo de las partes más relevantes del AACC propuesto.

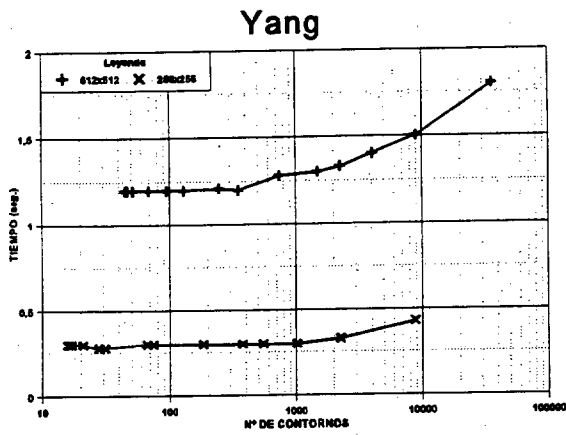


Figura III.23

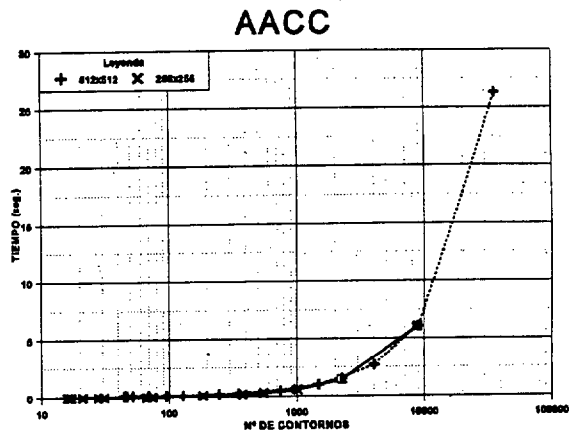


Figura III.24

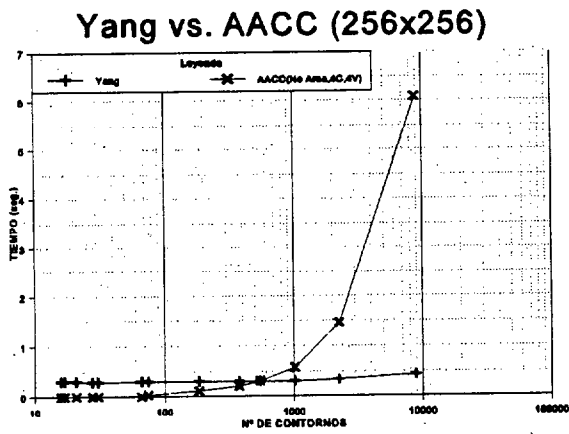


Figura III.25

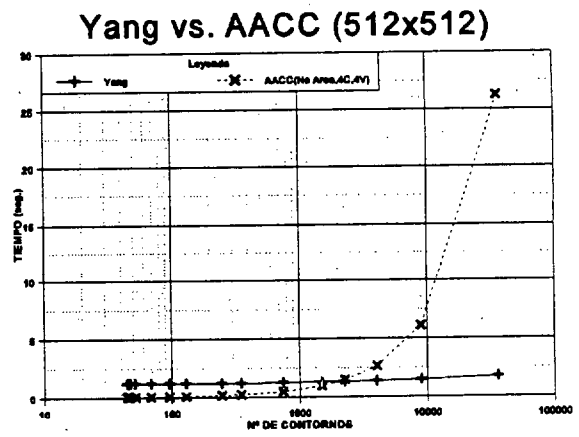


Figura III.26

COMPARATIVA AACC (512x512)

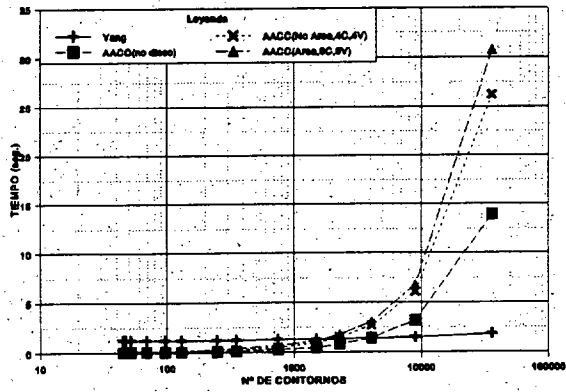


Figura III.27

AACC: BINARIAS vs. MULTIVALUADAS

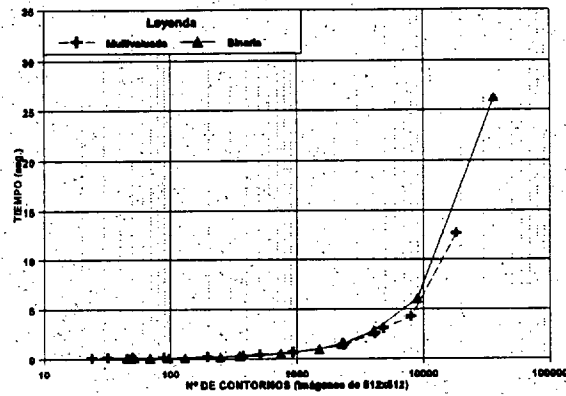


Figura III.28

Tiempo de ocupación (%)

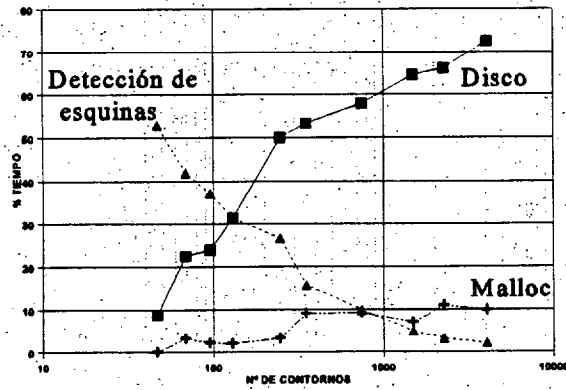


Figura III.29

III.3 HERRAMIENTAS DE DESARROLLO.

Se describirán a continuación y de manera sucinta, un conjunto de herramientas que integran el sistema de desarrollo del que se ha dotado a SVEX. La importancia de dotar a los sistemas de visión basados en conocimiento de un sistema de desarrollo adecuado ha sido señalada por algunos autores como fundamental para potenciar el desarrollo y la aplicabilidad de sistemas de visión basados en conocimiento en diferentes contextos [McKe-89][Hans-88]. Es habitual que la decisión de diseñar y construir las herramientas que deben integrar dicho entorno se produzca una vez que se ha acumulado una cierta experiencia en la utilización del sistema, se hayan afianzado sus modos de operación y se hayan detectado qué partes o tareas son susceptibles de utilizar algún tipo de herramienta específica. SVEX no ha sido una excepción a esta regla, por lo que el conjunto de herramientas que describiremos a continuación se encuentran necesariamente en una primera fase de realización y no integran, por el momento, un sistema de desarrollo homogéneo común a los diferentes niveles que componen SVEX.

III.3.1 PROCESO DE GENERACIÓN DE UNA APLICACIÓN.

La generación de una aplicación en SVEX refleja la estructuración por niveles del sistema. Así, un programa escrito para SVEX se compone -en principio- de uno o dos módulos en función de que dicha aplicación requiera sólo del Procesador de Pixel o de los Procesadores de Pixels y de Segmentos, respectivamente. Tanto el Procesador de Pixels como el de Segmentos disponen de un compilador propio encargado de la generación del módulo correspondiente. La organización de los compiladores y el proceso que se sigue en cada nivel son muy similares.

La figura III.30 esquematiza este proceso en el caso de la generación de una aplicación en el Procesador de Pixels. El usuario o programador edita un programa (*MiAplicación.ks*) e invoca al compilador de este procesador (KC). El compilador analiza el programa y lo traduce a estructuras de datos en lenguaje C generando el programa *MiAplicación.c*, que es compilado mediante un compilador C estándar y enlazado con las librerías que contienen el código correspondiente al Procesador de Pixels u otro código de usuario. Durante la compilación, el compilador KC resuelve las referencias a objetos no incluidos en el programa (*.ks) empleando una base de conocimiento. La base de conocimiento a la que el compilador KC acude por defecto,

contiene la definición de todos los tipos de manejadores de placa y formatos de ficheros de imágenes conocidos por el Procesador de Pixels (objetos "Channel"), y de todos los procedimientos (objetos "Procedure") conocidos por el Procesador de Pixels.

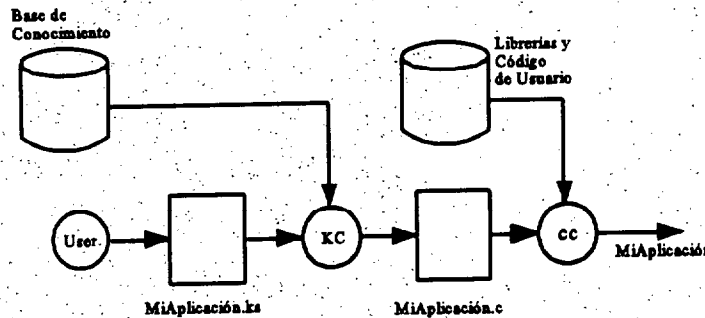


Figura III.30 Pasos para construir una aplicación. El compilador KC traduce el código fuente a estructuras en lenguaje C.

El proceso seguido para la generación del módulo correspondiente al Procesador de Segmentos es completamente análogo al anterior, sin más que cambiar el fichero que contiene el programa del Procesador de Segmentos (*MiAplicación.sc*) e invocar el compilador de este nivel (SC). La base de conocimiento consultada por este compilador por defecto contiene la declaración de los procedimientos (S_Procedures) conocidos en el nivel de los segmentos. El compilador SC tiene, sin embargo, dos particularidades respecto del compilador descrito para el Procesador de Pixels. De una parte, la invocación del compilador mediante la opción -d provoca la inclusión en el programa ejecutable de un módulo de depuración que se describirá en el epígrafe III.3.4. La otra particularidad está relacionada con la posibilidad de selección del módulo de presegmentación. Esto requiere de la introducción de un compilador subordinado al compilador SC que analice el programa del presegmentador seleccionado. En la implementación actual el código correspondiente al presegmentador debe estar delimitado en el programa del Procesador de Segmentos por la secuencia de tokens "&{{" y "}}&" que indican al compilador SC el comienzo y el final, respectivamente, de dicho código. Al finalizar el análisis léxico- sintáctico del programa (*.sc), excluido el código correspondiente al presegmentador, el compilador SC invoca al compilador encargado de analizar el código relativo al presegmentador, y que en el caso del

Capítulo III

presegmentador "Flood", se denomina FC. Este último compilador genera un programa ejecutable que representa al módulo de presegmentación del Procesador de Segmentos, y comunica ciertos resultados de la compilación al compilador SC para permitir la sincronización del módulo de presegmentación con el resto del Procesador de Segmentos durante el tiempo de ejecución. Con vistas a facilitar el diseño y depuración de programas, el compilador FC del presegmentador "Flood" puede invocarse directamente sobre un programa (*.fc) que contenga el código del presegmentador incluido normalmente en los programas del Procesador de Segmentos, en cuyo caso el programa que resulta puede ser ejecutado sin requerir de la intervención del Procesador de Segmentos. El compilador FC comparte la organización y filosofía de los compiladores KC y SC. En concreto, la base de conocimiento que incluye por defecto, contiene la declaración de objetos "Marker" y "Profiler" incluidos en el presegmentador "Flood".

Todos los compiladores que se han mencionado comparten un único diseño y utilizan las utilidades "lex" y "yacc" del S.O. UNIX para realizar, respectivamente, el análisis léxico y sintáctico durante la compilación.

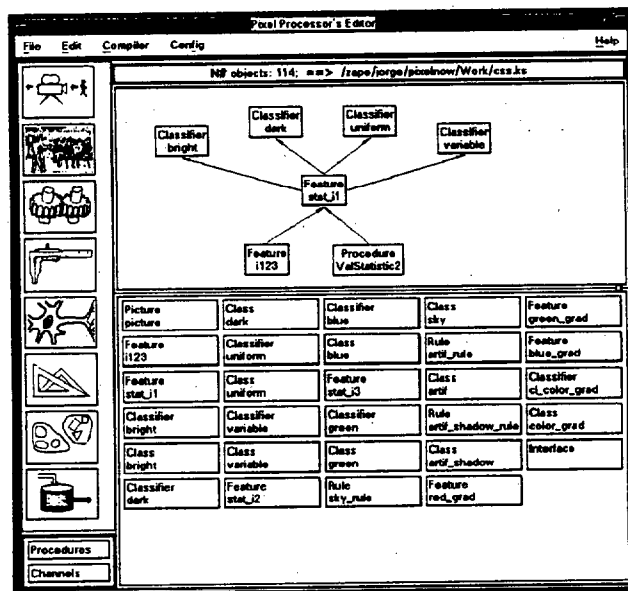


Figura III.31. Sesión de trabajo con el editor icónico.

III.3.2 UN PROTOTIPO DE EDITOR ICÓNICO.

En la línea de disponer un entorno de desarrollo "amigable" para el usuario, se ha desarrollado un editor icónico que permite editar, compilar y ejecutar programas del Procesador de Pixels [Nebo-94]. De acuerdo con las características declarativas presentes en el lenguaje de programación de SVEX, el editor está basado en el paradigma de "rellenado de fichas" (form-filling) que asemeja la declaración de un objeto al relleno de una ficha cuya estructura refleja la sintaxis del objeto en cuestión. La declaración de los objetos se facilita así considerablemente pues no es necesario recordar con precisión la sintaxis de cada objeto, los campos que son optativos, las selecciones por defecto, etc. Esto redundará en una edición más rápida y segura, libres de errores léxicos y sintácticos.

Las características más reseñables del editor son las siguientes:

Visualización de dependencias. Cada vez que se declara un objeto se incorpora un botón que lo representa a la ventana de edición. Al hacer click con el ratón sobre este botón, se visualizan en la parte superior de la ventana un grafo que muestra los objetos que intervienen en la definición del objeto en cuestión, así como los objetos en cuya definición interviene directamente (figura III.31). Esta característica permite al usuario, por una parte visualizar la estructura del programa que está editando y, por otra, acceder directamente a la definición de los objetos que dependen de uno dado.

Sistema de ayuda. El editor dispone de un sistema básico de ayuda, con enlaces de tipo hipertexto, permitirá al usuario moverse entre temas relacionados mediante un sencillo doble-click. El sistema de ayuda disponible en la actualidad versa fundamentalmente sobre la definición de los objetos y el significado de sus campos. En el futuro, contendrá también una descripción de los objetos "Channel" disponibles y sobre los procedimientos (objetos "Procedure") contenidos en la librería de procedimientos.

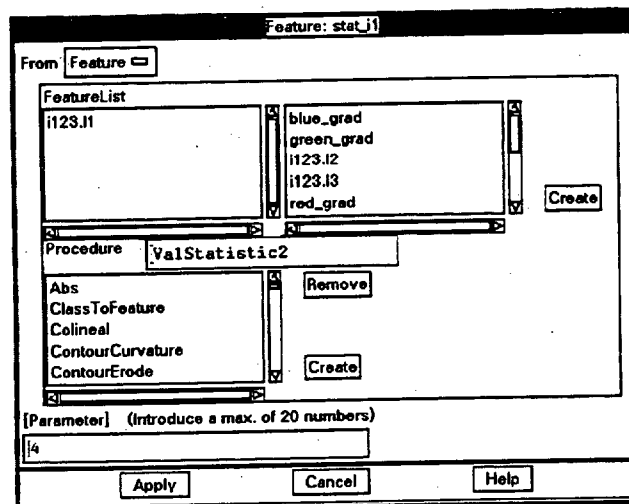


Figura III.32 Edición de un objeto "Feature" del Procesador de Pixels.

Compilación y ejecución desde el Editor. Desde el editor es posible compilar el programa y, eventualmente, ejecutarlo. Esto evita el tener que salir del editor para ejecutar un programa una vez que éste ha sido editado.

Una característica interesante en el diseño e implementación del editor es la reducción de los posibles campos presentes en la definición de los objetos a siete tipos diferentes ("GetString", "ChooseNFromM", "Toggle", ...) a los que se ha asociado un cierto aspecto gráfico en la interfaz del editor. Esto ha permitido un diseño del editor sumamente claro y estructurado, lo que hace relativamente fácil la modificación puntual de la sintaxis de un objeto. Pero lo más interesante es que abre la puerta a la posibilidad de generar automáticamente la interfaz gráfica de un editor de este tipo, a partir de la especificación sintáctica de un lenguaje declarativo que incluya el tipo asignable a cada campo.

III.3.3 UNA HERRAMIENTA DE ADQUISICIÓN DE MUESTRAS.

El objetivo de esta herramienta, denominada "Sampler", es la de servir como elemento de adquisición de muestras pertenecientes a una clase semántica definida por el usuario, tanto en el nivel de pixels como en el de los segmentos. La principal finalidad de las muestras de una clase es la de ser utilizadas en la elaboración del proceso abstractor que define dicha clase semántica

a partir de las propiedades (Features) de las muestras y desde su clasificación por medio de clasificadores. En el futuro, la herramienta Sampler formará parte de un módulo de aprendizaje que permitirá automatizar la definición de las clases a partir de la selección cualificada de muestras y un conjunto de características (features) que se consideren relevantes.

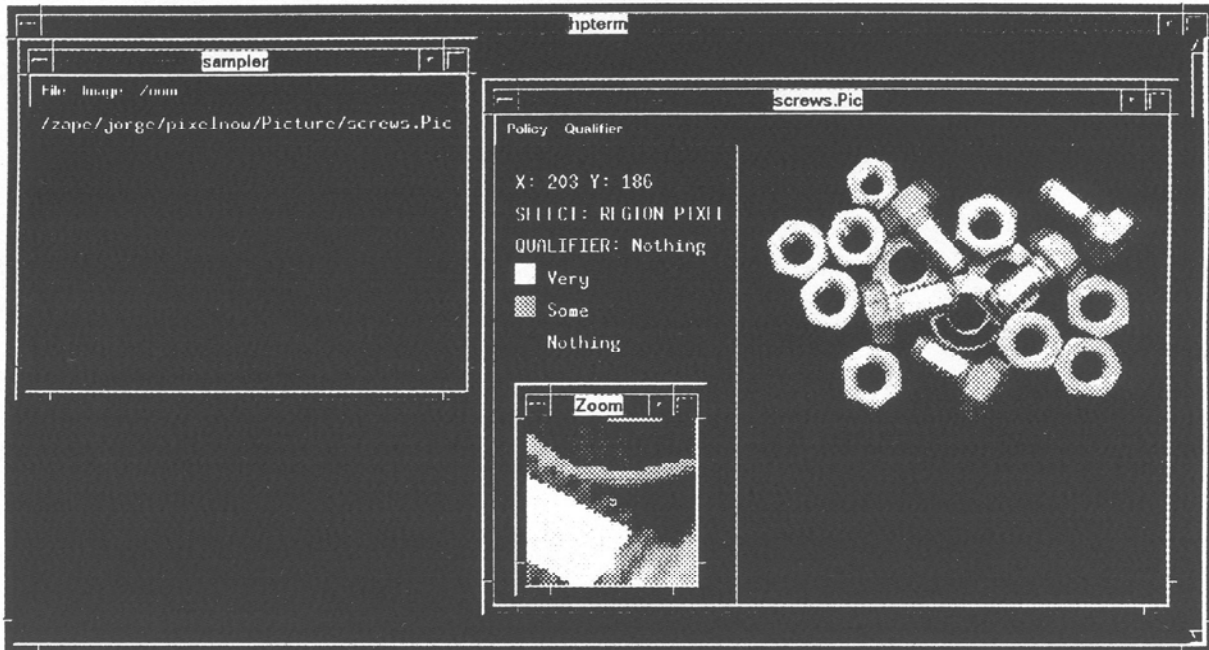


Figura III.33 Captura de muestras con la herramienta Sampler.

La figura III.33 muestra una típica sesión de trabajo con la herramienta Sampler en entorno Motif (X Window). El proceso de adquisición de las muestras es interactivo: el usuario selecciona una serie de imágenes de las que piensa extraer muestras representativas de una cierta clase de pixels o segmentos. A continuación, sobre cada imagen y por medio de un apuntador (ratón), realiza la selección de los pixels o segmentos y asigna un grado de pertenencia a la clase (Very, Some, Nothing en la figura III.33). La inspección de la imagen se facilita mediante una ventana de zoom variable. El proceso de selección de las muestras puede llevarse a cabo en forma de pixels individuales, asignando a cada uno de ellos un grado de pertenencia, o por grupos de pixels encerrados por un polígono y teniendo todos ellos el mismo grado de pertenencia a la clase. También es posible realizar la selección de muestras directamente sobre segmentos, lo que es necesario para que esta herramienta sea utilizable también en el nivel del Procesador de Segmentos.

Además de la obtención de muestras, la herramienta tiene la posibilidad de incorporar un visualizador de características y clases. La primera posibilidad permite al usuario juzgar la utilidad y potencia descriptiva de diferentes características para definir una clase. En el caso de las clases, su visualización es fundamental como mecanismo de realimentación en el proceso de su definición o aprendizaje.

III.3.4 DEPURACIÓN DE PROGRAMAS.

Entre las utilidades para la depuración de programas destacamos la que hemos denominado "ViewMap" que se puede emplear tanto en el nivel de los pixels como en el presegmentador, y la que hemos denominado "Xdebug" que se encuentra integrada en el Procesador de Segmentos. Ninguna de ellas puede ser calificada de depurador (debugger) pues son meros "inspectores" de resultados, por lo que sería más exacto llamarles "traceadores".

ViewMap permite inspeccionar mapas generados por un programa del Procesador de Pixels, es, por tanto, una herramienta que opera off-line, una vez que el programa ha concluido su ejecución. Permite visualizar mapas de características (features) o de diagnósticos (clases) en tres modos de consulta: puntual, poligonal y por intervalo. En el modo de consulta puntual presenta la opción de conocer las coordenadas y el valor en el mapa del punto seleccionado por un apuntador (ratón). En el modo de consulta poligonal, permite conocer el valor medio, la dispersión y el rango de variación del mapa en el interior de un área poligonal seleccionada. Finalmente, el modo de consulta por intervalo permite al usuario conocer la distribución de puntos que en un mapa se encuentran en un determinado rango. En general, ViewMap puede utilizarse en la inspección tanto de mapas de características o de diagnóstico, como de imágenes. La herramienta ViewMap es operativa en el entorno X Window y utiliza sólo recurso a nivel de Xlib.

El compilador del Procesador de Segmentos dispone de una opción de depuración (-d) que incluye en el código ejecutable un módulo de depuración llamado "Xdebug". La selección de esta opción permite la consulta on-line de los campos de datos de cualquier objeto declarado en el programa del Procesador de Segmentos que sea evaluable sobre un segmento. El usuario selecciona los objetos que desea consultar en la ventana "Xdebug" (figura III.34), e indica, a continuación, el segmento a consultar, señalándolo con el ratón en la ventana "Partition" que muestra el mapa de colores de la Partición. Dependiendo de si el programa incluye reglas y acciones de control o no, son posibles, respectivamente, dos modos de consulta o sólo uno. En el primer caso, el punto de ruptura para consulta (breakpoint) puede establecerse antes de la finalización del programa (cuando ya no es posible el disparo de ninguna regla de control) o cada vez que se produce una actualización de los diagnósticos tras la acción de una regla de control. Cuando no se incluyen reglas de control, la consulta se realiza antes de la finalización del programa cuando los diagnósticos ya han sido obtenidos. En cualquier caso, la selección de esta opción provoca que se genere un fichero de texto ("nombre del programa".debug) en el directorio indicado por la variable de entorno TMP. Este fichero contiene para cada segmento los valores

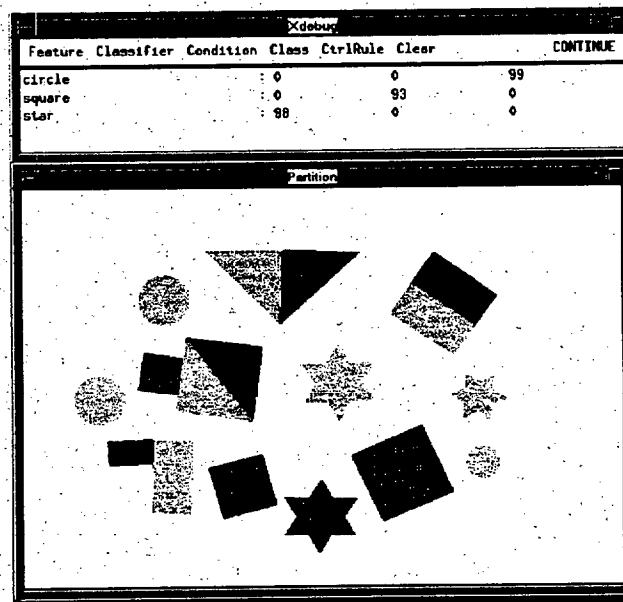
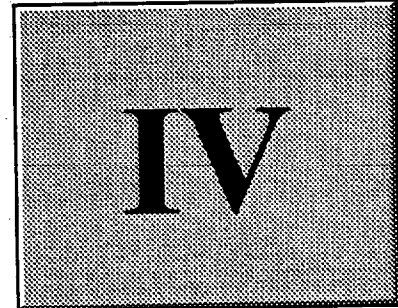


Figura III.34 Consulta de diagnósticos de segmentos durante la ejecución de un programa en el Procesador de Segmentos mediante la herramienta Xdebug.

Capítulo III

de todos los objetos, evaluables sobre un segmento, declarados en el programa: "S_Features", "S_Classifiers", "S_Conditions", "S_Classes", "S_Rules" y "S_CtrlRules". Cuando el programa incorpora reglas y acciones de control, esta información se reproduce tras cada iteración o modificación de la Partición, indicando la regla de control que se disparó, su grado de verificación y el resultado de la acción de control en el caso de reglas de evaluación condicional. El registro de esta información permite, por ejemplo, conocer la utilidad de las diferentes reglas de control en base a su frecuencia de disparo, lo que resulta relevante en la depuración de cualquier sistema de control basado en reglas. La utilización de la opción de depuración en el Procesador de Segmentos requiere del entorno X Window y utiliza recursos a nivel de Motif.

CAPÍTULO



Aplicaciones

IV.1 INTRODUCCIÓN.

La validación de cualquier desarrollo técnico o científico se sustenta en la experimentación, en la necesaria conexión con lo empírico. En tal sentido, SVEX se ha utilizado en la resolución de un conjunto de problemas característicos de la Visión Artificial. El ámbito de los casos de estudio o aplicaciones presentados en este capítulo es heterogéneo y se organiza en dos apartados. El primero de éstos está dedicado a la descripción de dos casos de estudio (detección de defectos en pieles y detección de movimiento) que se resuelven utilizando sólo el primer nivel del sistema, esto es el Procesador de Pixels. El otro apartado, epígrafe IV.3, abarca un conjunto más amplio de aplicaciones en los que se utilizan los dos niveles de SVEX. Los primeros tres casos presentados están inspirados en problemas característicos en entornos de naturaleza industrial y comprenden situaciones de control de calidad (fabricación de galletas y recepción de botellas en una planta de lavado) y de localización de piezas. El cuarto caso de estudio ilustra el uso del Procesador de Pixels y de Segmentos en un problema de segmentación

Capítulo IV

e identificación de contornos. El problema de identificación de partículas en sedimentología constituye el tema de la siguiente aplicación, la cual servirá para mostrar la conveniencia de establecer mecanismos de realimentación entre niveles en la resolución de determinados problemas. Finalmente, se describe la aplicación de SVEX en la segmentación de una serie de imágenes de exteriores en color. Para todos los casos de estudio se ha incluido el código utilizado en la programación del sistema, bien en el propio cuerpo de este capítulo como en los primeros casos de ambas secciones, o de forma sumaria en el apéndice E. Ésto proporciona una idea clara y detallada del estilo de programación propuesto ante ejemplos concretos.

El último apartado de este capítulo está dedicado a discutir las cuestiones más relevantes surgidas del desarrollo y análisis de los casos de estudio.

IV.2 APLICACIONES A NIVEL DE PIXELS. CASOS DE ESTUDIO.

Se describen en este epígrafe dos casos de estudio para ilustrar la utilización de SVEX en tareas que requieren sólo del Procesador de Pixels. En ambos casos, utilizaremos un procedimiento de evaluación de "Features" o características asociadas a estadísticas locales alrededor de cada pixel. El procedimiento de evaluación de las características se puede definir utilizando un proceso de convolución con un núcleo $W(x,y)$, que es no nulo y constante en una región, R , centrada en el origen de coordenadas:

$$W(x,y) = \begin{cases} 1 & (x,y) \in R \\ 0 & (x,y) \notin R \end{cases} \quad (\text{IV-1})$$

Verificándose que:

$$\int_R W(x',y') dx' dy' = A \quad (\text{IV-2})$$

La característica equivalente a la media estadística, $\mu(x,y)$, se define como sigue, en función de la entrada $I(x,y)$:

$$\mu(x,y) = \frac{1}{A_R} \int W(x-x',y-y') I(x',y') dx' dy' \quad (IV-3)$$

La característica varianza, $\sigma^2(x,y)$, se define como:

$$\sigma^2(x,y) = \frac{1}{A_R} \int W(x-x',y-y') [I(x',y') - \mu(x,y)]^2 dx' dy' = \frac{1}{A_R} \int W(x-x',y-y') I^2(x',y') dx' dy' - \mu^2(x,y) \quad (IV-4)$$

La varianza tiene valores altos en regiones con baja uniformidad. Es utilizada bien para detectar regiones con alto gradiente en imágenes sin texturas, o bien para comparar texturas. La característica "offset", $off(x,y)$, se utiliza como medida de discrepancia local, y se define como:

$$off(x,y) = I(x,y) - \mu(x,y) = I(x,y) - \frac{1}{A_R} \int W(x-x',y-y') I(x',y') dx' dy' \quad (IV-5)$$

Esta función es similar a la utilizada en diversos modelos de las etapas primarias de proceso de información visual biológica [Mor80a] [Mor80b]. En las aplicaciones que mostramos en este apartado, la función offset se utiliza para la detección de contornos por medio del proceso de detección de cruces por cero.

IV.2.1 CONTROL DE CALIDAD EN PELETERÍA.

Constituye un caso muy sencillo, en el cual la aplicación está relacionada con una tarea de verificación de calidad consistente en la detección de defectos en piezas de piel previamente a su corte y confección. Las zonas defectuosas se caracterizan por poseer una textura anormal y proporcionan la pista, basada en la experiencia o conocimiento de la aplicación, que marca la pauta para la selección de los procedimientos que deben ser utilizados. En la figura IV.1 se muestra el correspondiente diagrama de flujo de datos, donde los objetos "Channel" y "Procedure" apropiados para esta aplicación, se definen como sigue:

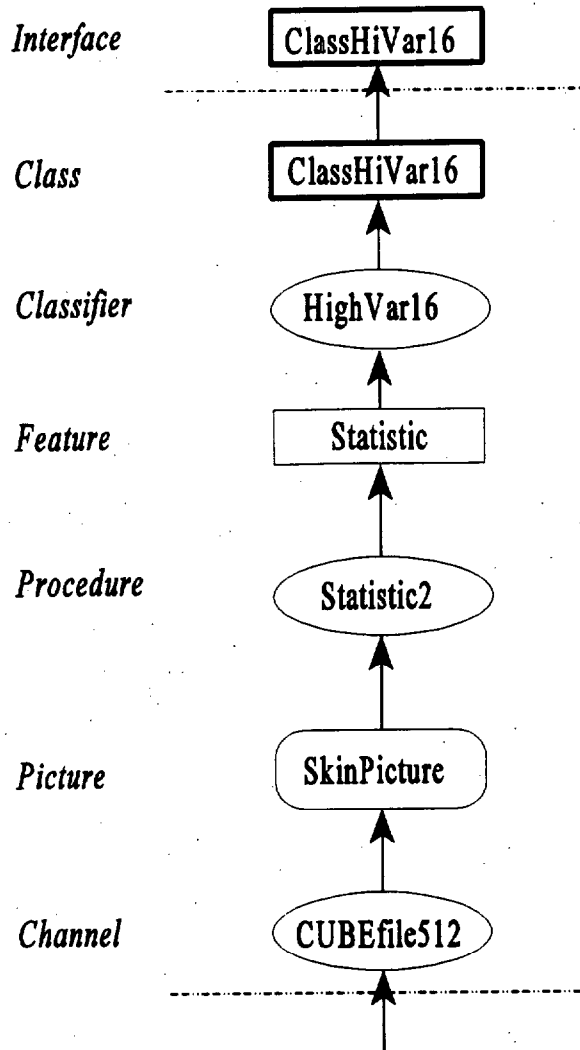


Figura IV.1 Diagrama de flujo de datos en el caso de estudio sobre control de defectos en peletería.

```

Define Channel CUBEfile512
  Type:      Grey
  SizeY:    512
  SizeX:    512
  Level:    512
  Homothety: Homothety3_2
  Function:  LoadPicture
EndDefine
  
```

```
Define Procedure PicStatistic2
  Function:    PicStatistic2
  AppliedTo:   Picture
  Sliced
  ParamNum:   1
  Selector:    mean, variance;
EndDefine
```

Los objetos que completan la definición de este programa para el Procesador de Pixels se declaran a continuación:

```
Define Picture SkinPicture
  Channel:     CUBEfile512
  File:        "../Picture/skinA04.Pic"
  Show
EndDefine
```

```
Define Feature Statistic
  From:        Picture
  Procedure:   PicStatistic2
  Parameter:   16;
EndDefine
```

```
Define Classifier HighVar16
  FeatureList: Statistic.variance;
  Functional:  Unitary
  Decision:   Threshold
              Sign: Positive
              Level: 500.0
EndDefine
```

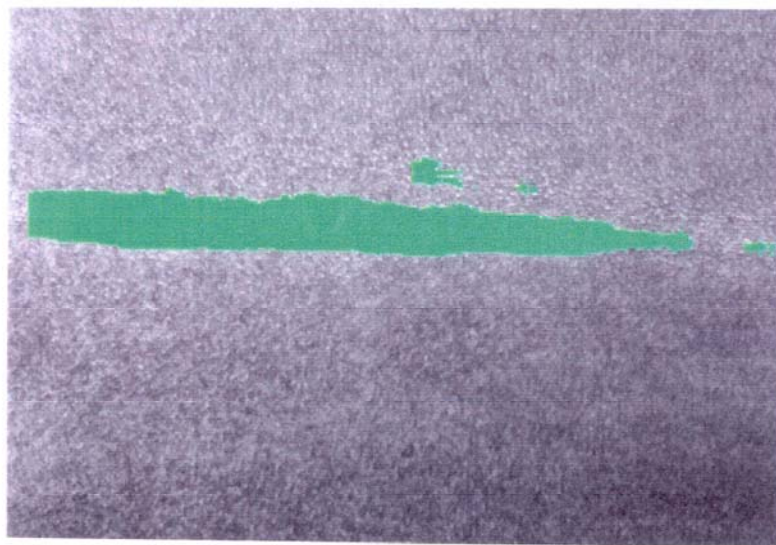
```
Define Class ClassHiVar16
  Classifier:  HighVar16
  Show
              Overlapped: 70
EndDefine
```

```
Define Interface
  Class:      ClassHiVar16;
EndDefine
```

La figura IV.2 muestra la imagen original (A) y la clase resultado denominada "*ClassHiVar16*" superpuesta con la original (B).



A



B

Figura IV.2 Control de defectos en peletería. El objetivo consiste en detectar regiones con defectos en la textura de la piel. A) Imagen original. B) Imagen resultado mostrando (verde) las zonas de textura anómala. Son zonas cuyo grado de pertenencia a la clase "ClassHiVar16" supera el umbral 70.

IV.2.2 DETECCIÓN DE MOVIMIENTO.

Este ejemplo ilustra como detectar los contornos de un objeto móvil en una secuencia de dos imágenes, o "instantáneas" (slices), de la escena mostrada en la figura IV.4. Las instantáneas de una secuencia de N imágenes se denominan como *Time0*, *Time1*, ..., *TimeN* que son tokens del lenguaje. Más que detectar los cambios entre las dos imágenes en el nivel de la señal, los cambios son detectados al nivel simbólico, correlacionando localmente las clases correspondientes a los contornos de las dos instantáneas, lo que implica como tarea previa la detección de los contornos en cada instantánea. La figura IV.3 muestra el flujo de datos de esta aplicación y las figuras IV.5.

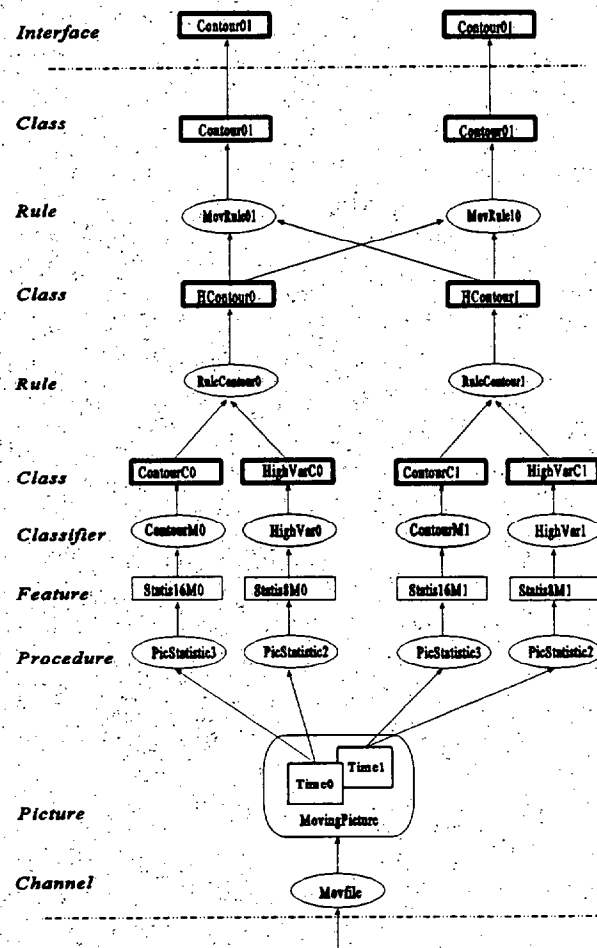


Figura IV.3 Flujo de datos en el caso de estudio sobre detección de movimiento.

Capítulo IV

y siguientes muestran los resultados del caso. En esta aplicación la secuencia de imágenes es leída de un canal definido como sigue:

```
Define Channel MOVfile
  Type:      Sequence 2
  SizeY:     512
  SizeX:     512
  Level:     512
  Homothety: Homothety3_2
  Function:  LoadPicture
EndDefine
```

El objeto "Picture" se define como sigue:

```
Define Picture MovingPicture
  Channel:   MOVfile
  File:     "../Picture/Moving.Pic"
  Show
EndDefine
```

Para computar el offset se utiliza una estadística local en un entorno de 16x16. Si la imagen es de tipo *Sequence* y el procedimiento es *Sliced*, entonces se debe especificar el slice sobre el cual se realizan las evaluaciones. El procedimiento "*PicStatistic3*" evalúa las medias y offset locales.

```
Define Feature Statis16M0
  From:      Picture
  Slice:     Time0
  Procedure  PicStatistic3
  Parameter: 16;
EndDefine
```

```
Define Classifier ContourM0
  FeatureList: Statis16M0.offset;
  Functional:  Unitary
  Decision:   Cross
  Level:     0.0
EndDefine
```



```
Define Class Contour0
  Classifier: ContourM0
  Show
EndDefine
```

Los contornos seleccionados son los que se presentan en regiones con valores altos de la varianza local en entornos de 8x8. El procedimiento "*PicStatistic2*", utilizado en el caso de estudio anterior, evalúa la media y varianza. A partir de la varianza se obtiene la clase correspondiente a los pixels de alta variación mediante una decisión de tipo umbral.

```
Define Feature Statis8M0
  From:      Picture
  Slice:     Time0
  Procedure: PicStatistic2
  Parameter: 8;
EndDefine
```

```
Define Classifier HighVar0
  FeatureList: Statis8M0.variance;
  Functional:  Unitary
  Decision:   Threshold
  Sign:       Positive
  Level:     50.0
EndDefine
```

```
Define Class HighVarC0
  Classifier: HighVar0;
  Show
EndDefine
```

La clase de contornos de buena calidad se obtiene por medio de la siguiente regla:

```
Define Rule RuleContour0
  Target:      HContour0
  Condition:   HighVarC0 And Contour0
EndDefine
```

Capítulo IV

```
Define Class HContour0
  Rule:      RuleContour0;
  Show
EndDefine
```

Un código similar al descrito se define para detectar contornos en el "slice" denominado *Time1*. Las clases objetivo del problema se denominan "*Contour01*" y "*Contour10*", cuyas definiciones son las siguientes:

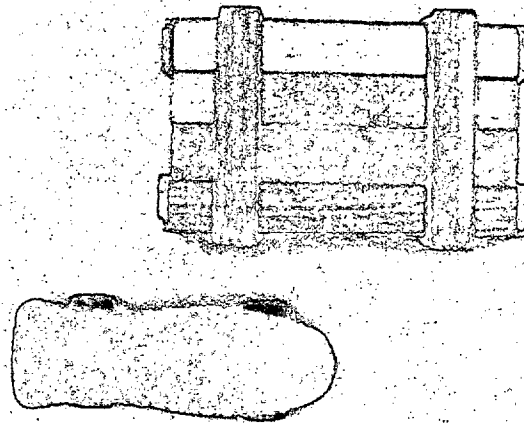
```
Define Rule MovRule10
  Target:    Contour10
  Condition: HContour0 And Not HContour1(AnyNB)
EndDefine
```

```
Define Rule MovRule01
  Target:    Contour10
  Condition: Not HContour0 (AnyNB) And HContour1
EndDefine
```

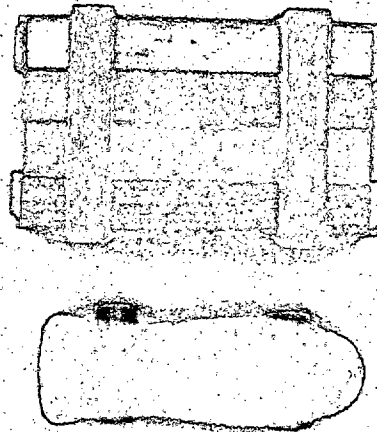
```
Define Class Contour01
  Rule:      MovRule01;
  Show
  Overlapped: 70
EndDefine
```

```
Define Class Contour10
  Rule:      MovRule10;
  Show
EndDefine
```

```
Define Interface
  Class: Contour01, Contour10;
EndDefine
```

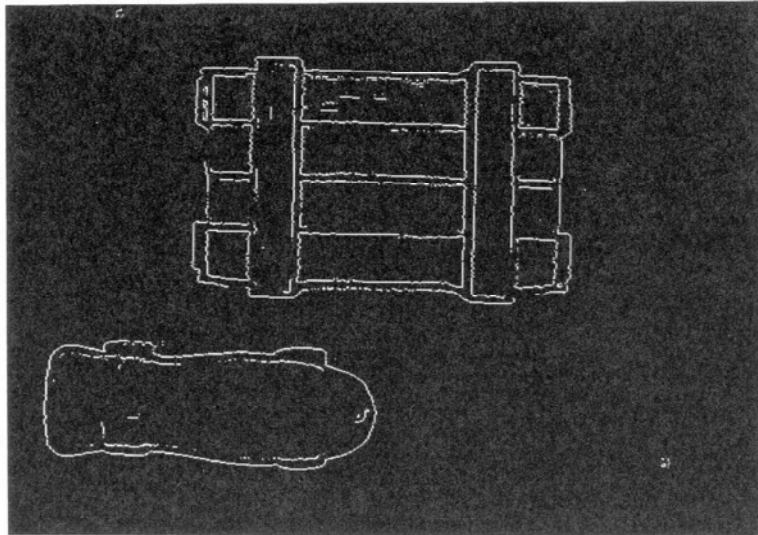


A

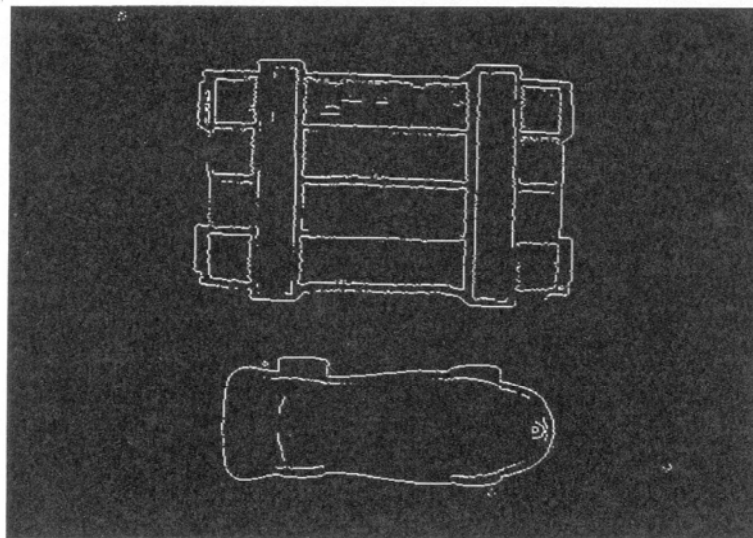


B

Figura IV.4 Instantáneas de la secuencia utilizada en la detección de objetos en movimiento. A) Instantánea Time0. B) Instantánea Time1.

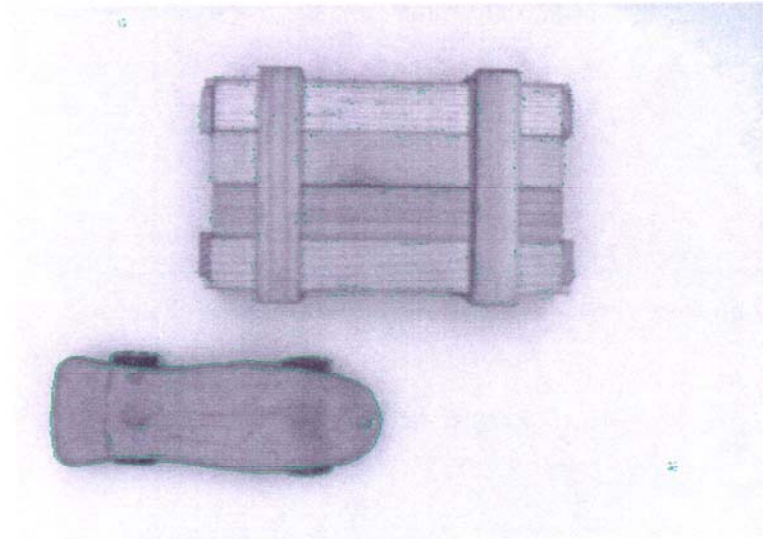


A

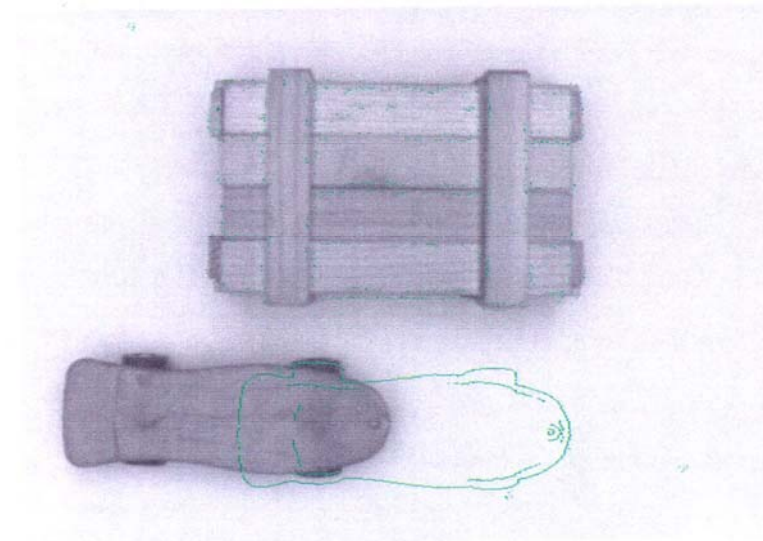


B

Figura IV.5 Contornos seleccionados en ambas instantáneas. A) Clase *HContour0*. B) Clase *HContour1*



A



B

Figura IV.6 Clases resultados. A) Contour10. B) Contour01 superpuesta con la instantánea Time0.

IV.3 APLICACIONES A NIVEL DE SEGMENTOS. CASOS DE ESTUDIO.

Se describen en esta sección un conjunto de experimentos fruto de la aplicación de SVEX en una variedad de problemas. A diferencia de las aplicaciones de la sección anterior, los casos de estudio presentados en este epígrafe implican a los niveles de pixels y de segmentos del sistema.

Todos los casos incluyen los listados de la aplicación correspondientes al Procesador de Pixels (*.ks) y de Segmentos (*.sc), si bien en algunos casos, y por razones de extensión, el código se ha trasladado al Apéndice E. En aras de una mayor claridad, el código correspondiente al presegmentador (*.fc) se muestra siempre separado del resto del programa del Procesador de Segmentos.

IV.3.1 LOCALIZACIÓN E IDENTIFICACIÓN EN ENTORNOS INDUSTRIALES.

A. Control de calidad en un proceso de fabricación de galletas.

En el primer ejemplo de este apartado se efectúa un control de calidad en un proceso de fabricación de galletas, articulado sobre los dos niveles de SVEX. En el nivel de Procesador de Pixels, se realiza un umbralizado trapezoidal de la imagen original para producir el diagnóstico "Biscuit" (en base al nivel de gris característico de las galletas en el entorno del problema). El procedimiento "Value" empleado en este nivel permite definir una "Feature" directamente a partir de la propia imagen. El presegmentador utiliza este diagnóstico para producir una partición de la imagen según dos prototipos: "biscuit" y "background". Para definir los mínimos de estos prototipos se emplea el marcador "WinStandardMarker" que localiza en el mapa de pixels "Image" los pixels que verifican las siguientes tres condiciones: a) el pixel está incluido en el intervalo de definición del prototipo, b) el pixel ocupa el centro de un ventana MxM que contiene al menos K pixels que verifican la condición (a), y c) el pixel en cuestión se encuentra a una "altura" inferior a H en el perfil de la imagen. Los tres parámetros que se detallan en la selección de este marcador corresponden respectivamente a M, el lado de la ventana, K y H. Finalmente, el Procesador de Segmentos verifica la calidad de la galleta comprobando que tiene el tamaño suficiente y que su forma es rectangular. Para este último cometido se utiliza una medida de la

curvatura del contorno externo de cada galleta, el cual se umbraliza para contabilizar el número de esquinas o puntos de alta curvatura. Al procedimiento "Corners", utilizado para tal fin, se le pasan tres parámetros que determinan, respectivamente: el número de puntos del contorno que intervienen en el cálculo de la curvatura (M-curvatura), la longitud mínima del segmento y el umbral de curvatura a partir del cual se considera que un punto del borde externo del segmento es una esquina. La galleta que "supera" el test de calidad, "Good_biscuit", es aquella que tiene un tamaño suficiente, exactamente cuatro esquinas y no tiene agujeros o zonas interiores (el fondo tiene cuatro esquinas sobre su perímetro externo y tantos agujeros como galletas). La figura IV.7 muestra el flujo de datos a través de los niveles de pixels y de segmentos y las figura IV.8 contiene la imagen original, los resultados intermedios y final del proceso.

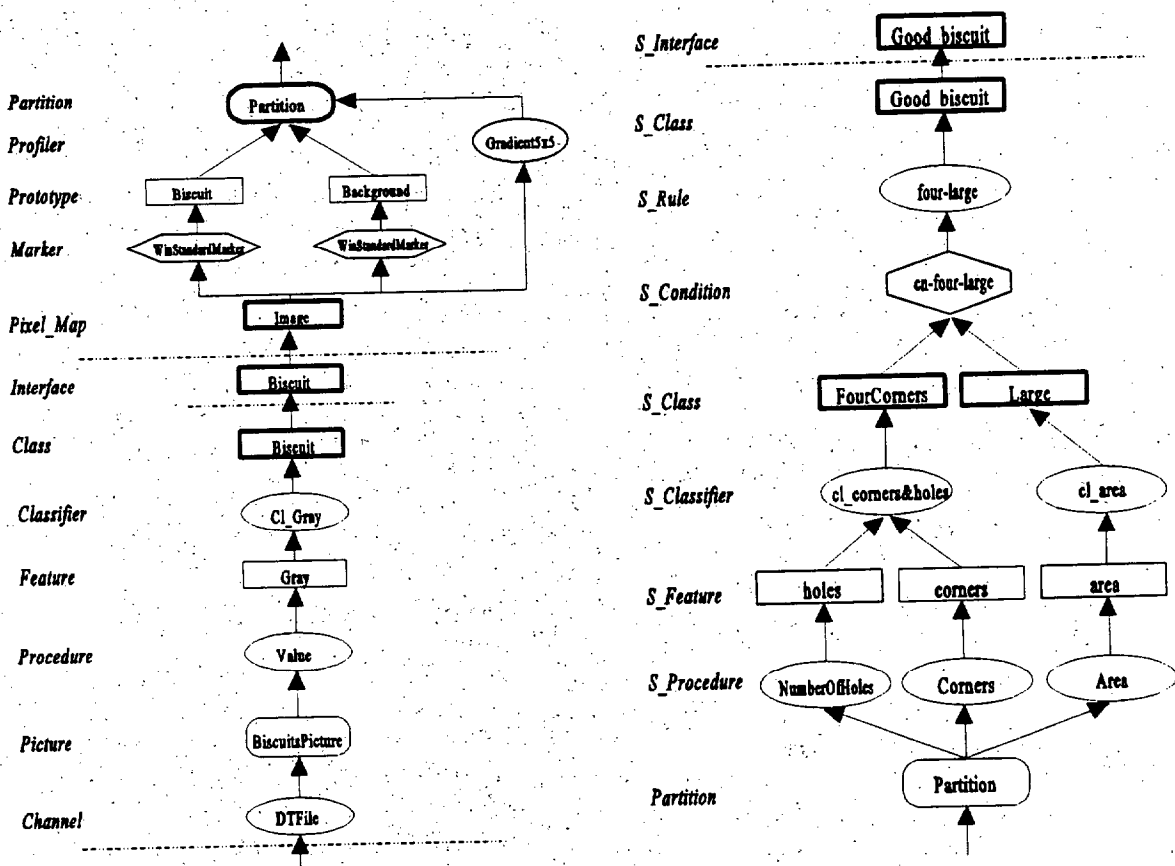


Figura IV.7 Flujo de datos en SVEX en el ejemplo sobre control de calidad de un proceso de fabricación de galletas.

BISCUITS.KS

Define Channel *DTfile*
Type: Grey
SizeY: 256
SizeX: 256
Level: 64
Homothety: Homothety3_2
Function: LoadPicture

EndDefine

Define Picture *BiscuitsPicture*
Channel: *DTfile*
File : *"../Picture/biscuits.Pic"*

EndDefine

Define Feature *Gray*
From: Picture
Procedure: *Value*

EndDefine

Define Procedure *Value*
Function: *Value*
AppliedTo: Picture
Sliced
ParamNum: 0

EndDefine

Define Classifier *Cl_Gray*
FeatureList: *Gray;*
Functional: Unitary
Decision: Trapezoid
Level: *10, 30, 130, 200*

EndDefine

Define Class *Biscuit*
Classifier: *Cl_Gray;*
Show

EndDefine

Define Interface
Class: *Biscuit;*

EndDefine

BISCUITS.FC

Define Pixel_Map *Image*
P_Class: *Biscuit*
Profiler: *Gradient5x5*
Weight: 1

EndDefine

Define Profiler *Gradient5x5*
Procedure: *Gradient5x5*
AppliedTo: Window
ParamNum: 0

EndDefine

Define Marker *WinStandardMarker*
Procedure: *WinStandard*
AppliedTo: Window
ParamNum: 3

EndDefine

Define Prototype *biscuit*
MapList: *Image;*
Centroid: 75;
Dispersion: 45;
Marker: *WinStandardMarker*
Mode: Merge
Color: Maroon
Parameter: 5, 15, 20;
Connectivity: 8-Connectivity
Show

EndDefine

Define Prototype *background*
MapList: *Image;*
Centroid: 0;
Dispersion: 5;
Marker: *WinStandardMarker*
Mode: Merge
Color: Green
Parameter: 3, 9, 10;
Connectivity: 8-Connectivity

EndDefine

BISCUITS.SC

Define P_Interface

Class: *Biscuit;*

EndDefine

Define S_MakePartition

P_Class: *Biscuit;*

Presegmenter: *"FC"*

ProgramName: *"biscuits"*

Connectivity: *4-Connectivity*

Neighborhood: *8-Connectivity*

Show

EndDefine

Define S_Feature *area*

From: *Partition*

Procedure: *Area*

EndDefine

Define S_Procedure *Area*

Function: *Area*

AppliedTo: *Partition*

ArgNum: *0*

ParamNum: *0*

EndDefine

Define S_Classifier

cl_area

Feature: *area;*

Functional: *Unitary*

Decision: *Threshold*

Sign: *Positive*

Level: *9000*

EndDefine

Define S_Feature

corners

From: *Partition*

Procedure: *Corners*

Parameter: *13, 30, 3;*

EndDefine

Define S_Procedure *Corners*

Function: *Corners*

AppliedTo: *Partition*

ArgNum: *0*

ParamNum: *3*

EndDefine

Define S_Feature

holes

From: *Partition*

Procedure: *NumberOfHoles*

EndDefine

Define S_Procedure *NumberOfHoles*

Function: *NumberOfHoles*

AppliedTo: *Partition*

ArgNum: *0*

ParamNum: *0*

EndDefine

Define S_Classifier

cl_corners&holes

Feature: *corners, holes;*

Functional: *Lineal*

Coefficient: *1, 100;*

Decision: *Trapezoid*

Sign: *Positive*

Level: *3.5, 3.5, 4.5, 4.5*

EndDefine

Define S_Class

FourCorners

Classifier: *cl_corners&holes;*

EndDefine

Define S_Class

Large

Classifier: *cl_area;*

EndDefine

Define S_Class

Good_biscuit

Rule: *four_large;*

Show

Level: *80*

EndDefine

Define S_Rule

four_large

Target: *Good_biscuit*

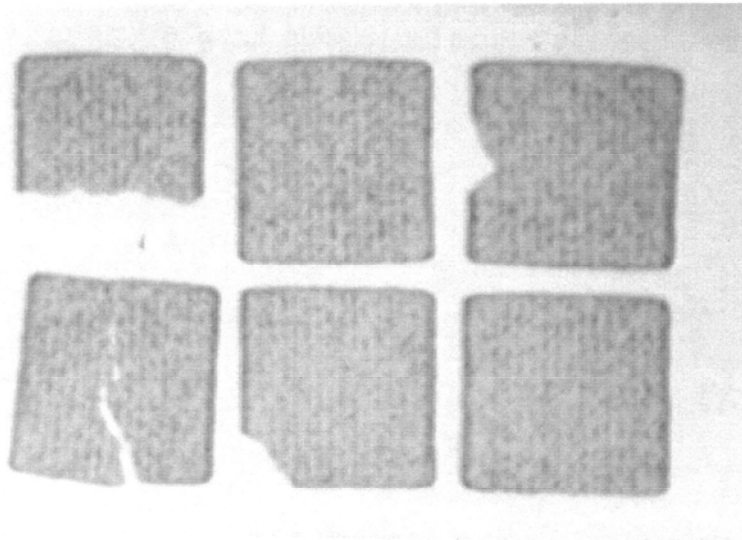
Condition: *cn_four_large*

EndDefine

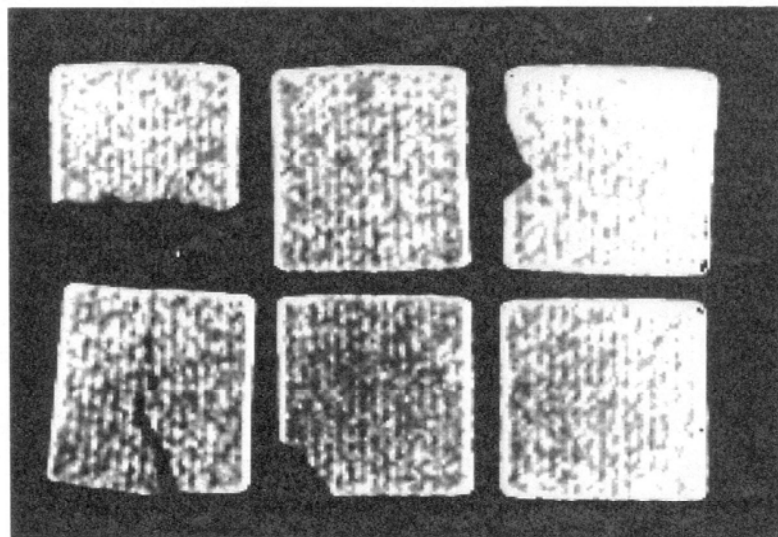
Capítulo IV

```
Define S_Condition  cn_four_large  
  Premise:  
  Is (Large And FourCorners4, X)  
EndDefine
```

```
Define S_Interface  
  Class:  Good_biscuit;  
EndDefine
```

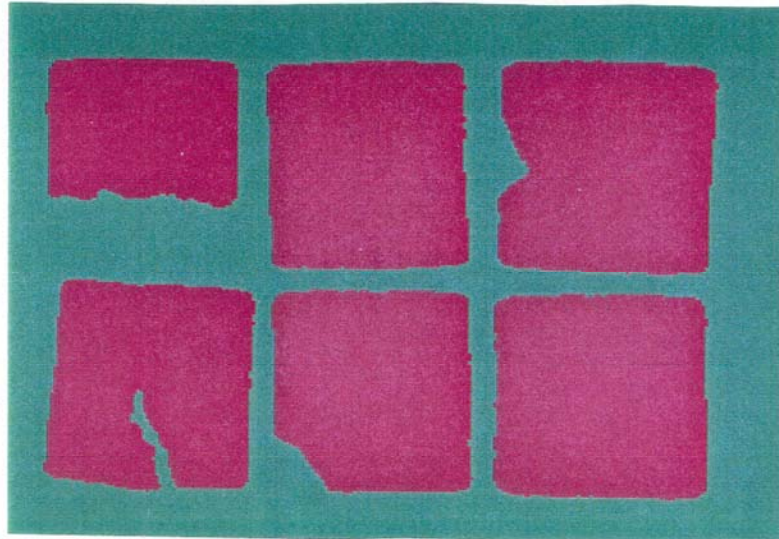


A

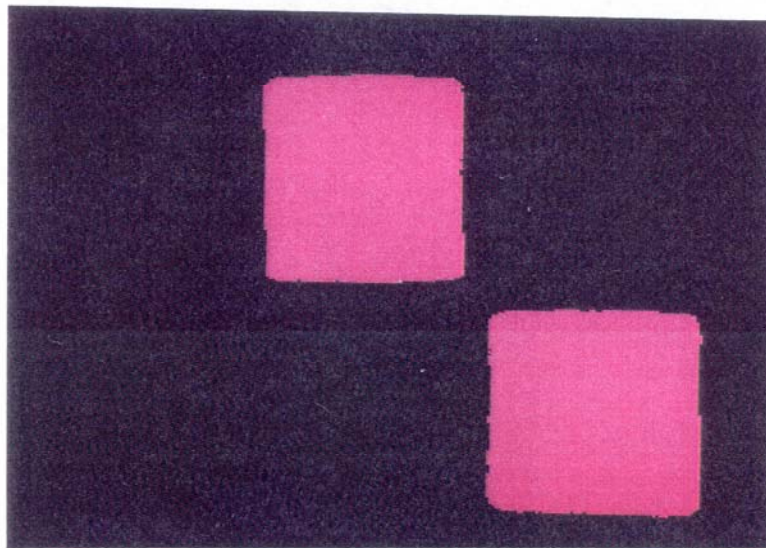


B

Figura IV.8 A) Imagen original del caso de estudio. B) Mapa de diagnóstico de la clase "Biscuit", donde el grado de pertenencia a esta clase de pixels es directamente proporcional al nivel de gris.



C



D

Figura IV.8 (cont.) C) Partición inicial generada por el presegmentador. D) Resultado final donde se muestran las galletas que han superado el control de calidad. Son aquellas cuyo grado de pertenencia a la clase de segmentos "Good_biscuit" es mayor que el umbral fijado (80).

B. Control de calidad en una planta de lavado de botellas.

En este ejemplo se trata de detectar botellas, que en una vista cenital, tengan fragmentado o astillado el cuello de la botella. La figura IV.10 muestra dos escenas características de este entorno, conteniendo ambas una botella en buenas condiciones y otra defectuosa. El proceso de análisis de estas imágenes, orientado a detectar las botellas que no presentan defectos, se organiza sobre los Procesadores de Pixels y de Segmentos como se describe a continuación. El Procesador de Pixels normaliza la imagen de entrada a 100 niveles. El presegmentador trabaja con la imagen normalizada para aislar tres tipos de regiones o prototipos (figura IV.11): fondo o prototipo "Background" (visualizado en verde), cuerpo de botella o "Body" (rojo) y cuello de botella o "Top" (azul). El segmentador analiza los segmentos resultantes y los clasifica, mediante las características: radio cuadrático medio y dispersión, que se obtienen del borde externo de cada segmento, en los diagnósticos:

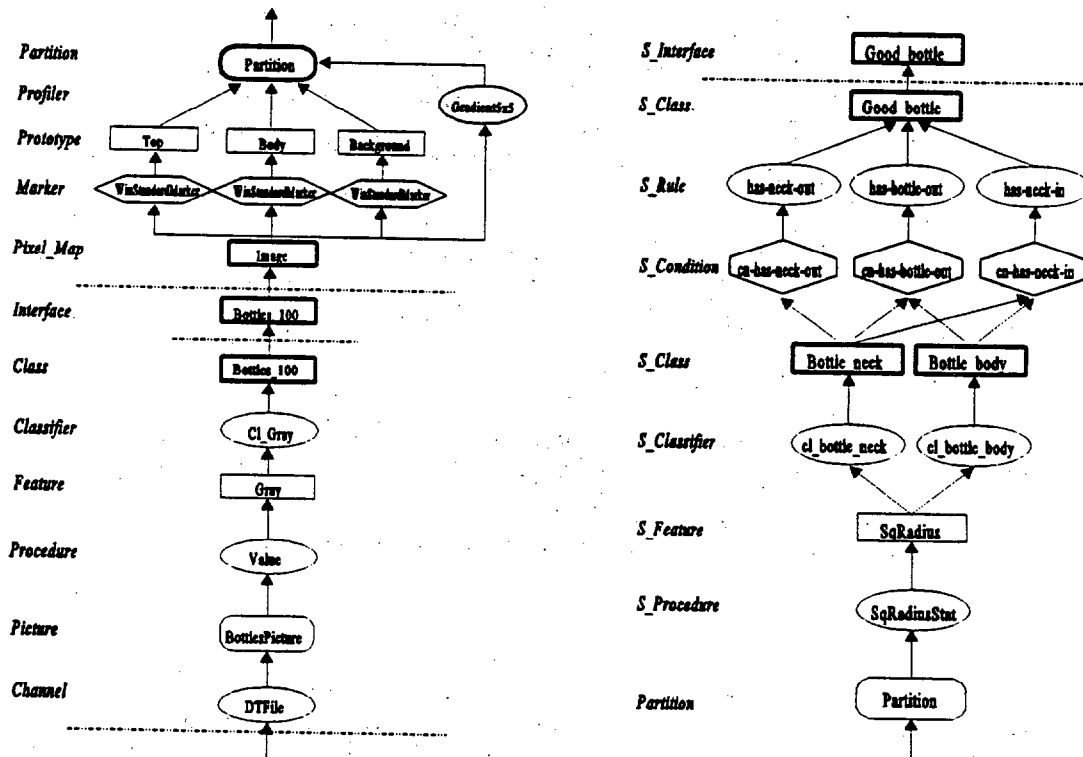


Figura IV.9. Flujo de datos en el caso de estudio sobre detección de defectos en botellas.

"cuerpo de botella" ("*Bottle_body*" en el código adjunto) y "cuello de botella" (*Bottle_neck*). El diagnóstico final que corresponde a las botellas que no presentan defectos es el resultado de la combinación de tres reglas que capturan la dependencia estructural de las diferentes partes que componen una botella en la partición inicial. La definición de estas dependencias es posible gracias al uso de localizadores espaciales en las premisas de los objetos "S_Condition". La figura IV.12 muestra las botellas seleccionadas superpuestas sobre cada una de las imágenes originales. En el código que sigue no se incluyen objetos como "*Channel DTfile*", "*Marker WinStandardMarker*", etc..., cuya declaración es idéntica al del caso presentado anteriormente.

BOTTLES.KS

```
Define Picture BottlesPicture
  Channel: DTfile
  File :   "../Picture/bottles.Pic"
EndDefine
```

```
Define Feature gray
  From:       Picture
  Procedure:  Value
EndDefine
```

```
Define Classifier cl_gray
  FeatureList: gray;
  Functional:  Unitary
  Decision:   Ramp
  Sign:       Positive
  Level:      0, 256
EndDefine
```

```
Define Class Bottles_100
  Classifier: cl_gray;
  Show
EndDefine
```

```
Define Interface
  Class:      Bottles_100;
EndDefine
```

BOTTLES.FC

```
Define Pixel_Map Image
  P_Class:      Bottles_100
  Profiler:     Gradient5x5
  Weight:       1
EndDefine
```

```
Define Prototype Top
  MapList:      Image;
  Centroid:     15;
  Dispersion:   15;
  Marker:       WinStandardMarker
  Mode:         Merge
  Color:        Navyblue
  Parameter:    5, 15, 40;
  Connectivity: 8-Connectivity
EndDefine
```

```
Define Prototype Body
  MapList:      Image;
  Centroid:     65;
  Dispersion:   10;
  Marker:       WinStandardMarker
  Mode:         Merge
  Color:        Red
  Parameter:    5, 10, 40;
  Connectivity: 8-Connectivity
EndDefine
```

Capitulo IV

Define Prototype *Background*

MapList: *Image*;
Centroid: *95*;
Dispersion: *5*;
Marker: *WinStandardMarker*
Mode: *Merge*
Color: *Green*
Parameter: *3, 9, 10*;
Connectivity: *8-Connectivity*

EndDefine

BOTTLES.SC

Define S_MakePartition

P_Class: *Bottles_100*;
Presegmenter: *"FC"*
ProgramName: *"bottles"*
Connectivity: *4-Connectivity*
Neighborhood: *8-Connectivity*

EndDefine

Define P_Interface

Class: *Bottles_100*;

EndDefine

Define S_Feature *SqRadio*

From: *Partition*
Procedure: *SqRadiusStat*

EndDefine

Define S_Procedure *SqRadiusStat*

Function: *SqRadiusStat*
AppliedTo: *Partition*
ArgNum: *0*
ParamNum: *0*
Selector: *mean, dispersion*;

EndDefine

Define S_Classifier *cl_bottle_body*

Feature: *sqradio.mean,*
sqradio.dispersion;
Functional: *Quadratic*
Mean: *5.2e3, 200*;
Variance: *1, 0, 0, 1*;

Decision: *Threshold*

Sign: *Negative*

Level: *4.0e4*

EndDefine

Define S_Classifier *cl_bottle_neck*

Feature: *sqradio.mean,*
sqradio.dispersion;

Functional: *Quadratic*

Mean: *2.3e3, 200*;

Variance: *1, 0, 0, 1*;

Decision: *Threshold*

Sign: *Negative*

Level: *2.6e4*

EndDefine

Define S_Class *Bottle_neck*

Classifier: *cl_bottle_neck*;

EndDefine

Define S_Class *Bottle_body*

Classifier: *cl_bottle_body*;

EndDefine

Define S_Class *Good_bottle*

Rule: *has_neck_in,*
has_bottle_out,
has_neck_out;

Show

Level: *80*

Overlapped: *"bottles_100"*

EndDefine

Define S_Rule *has_neck_in*

Target: *Good_bottle*

Condition: *cn_has_neck_in*

EndDefine

Define S_Rule *has_bottle_out*

Target: *Good_bottle*

Condition: *cn_has_bottle_out*

EndDefine

```
Define S_Rule has_neck_out  
  Target: Good_bottle  
  Condition: cn_has_neck_out  
EndDefine
```

```
Define S_Condition cn_has_neck_out  
  Premise: Is(Bottle_neck, Contains X)  
EndDefine
```

```
Define S_Condition cn_has_neck_in  
  Premise: Is (Bottle_body, X) And  
           Is (Bottle_neck, InNeib X)  
EndDefine
```

```
Define S_Condition cn_has_bottle_out  
  Premise: Is (Bottle_neck, X) And  
           Is(Bottle_body, Contains X)  
EndDefine
```

```
Define S_Interface  
  Class: Good_bottle;  
EndDefine
```

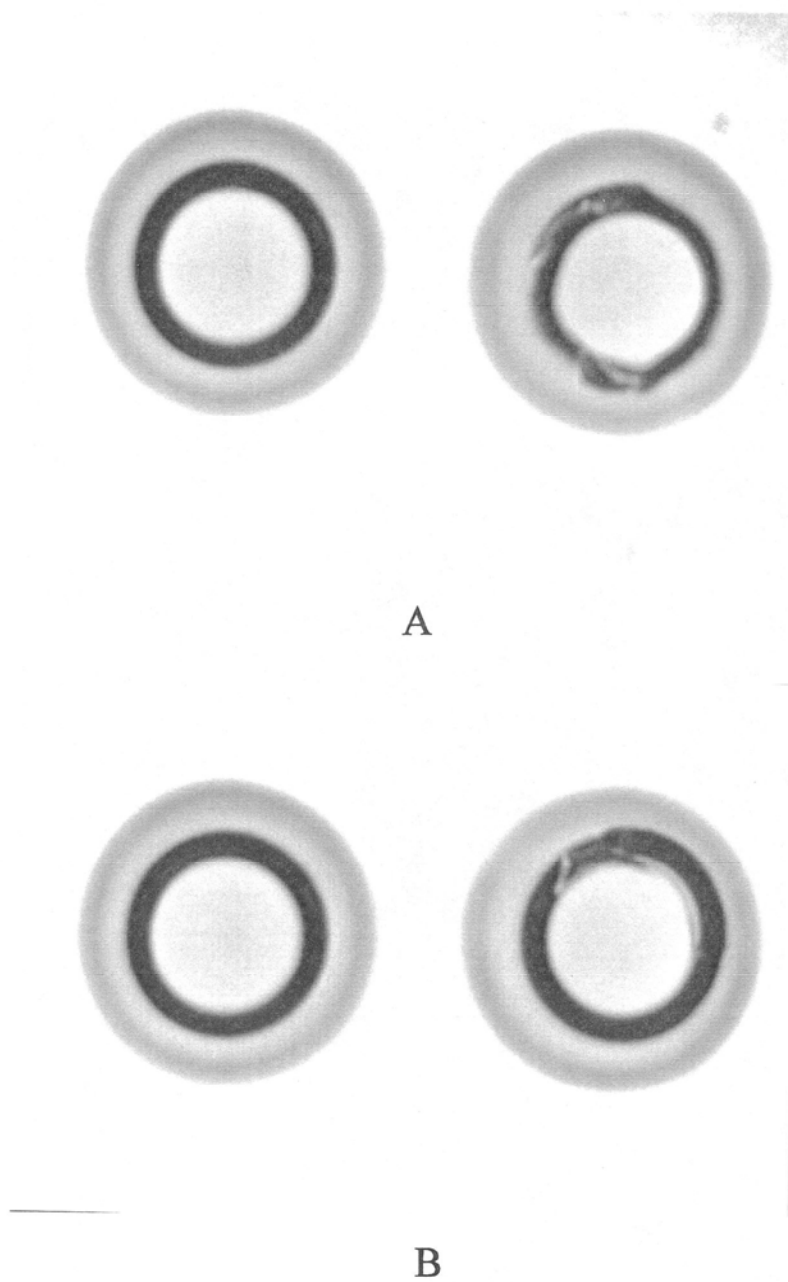
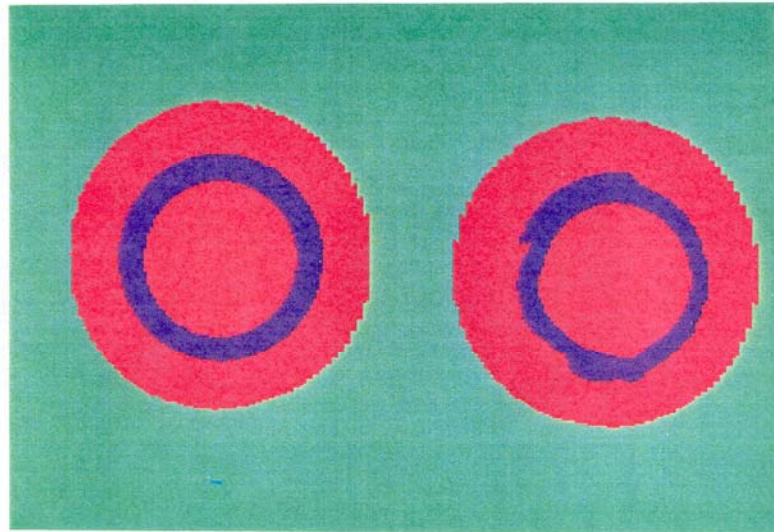
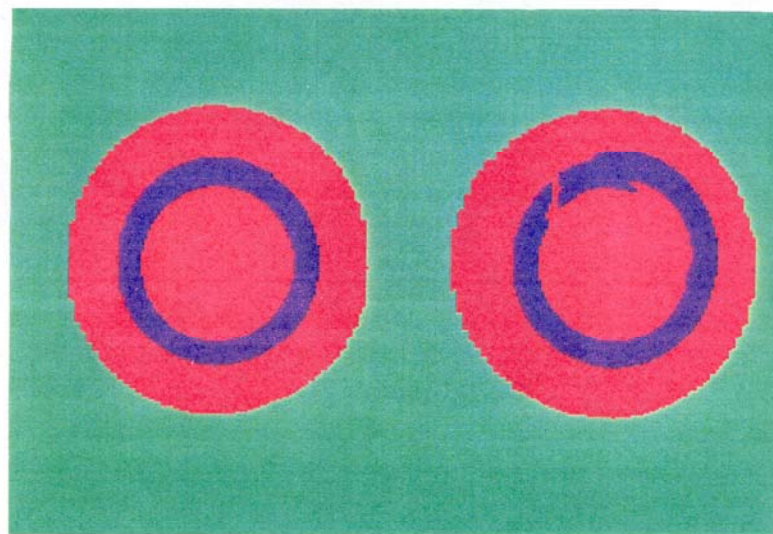


Figura IV.10 A) y B) Imágenes originales utilizadas en este caso de estudio, conteniendo cada una de ellas una botella en buenas condiciones y otra con diferentes defectos en el cuello.

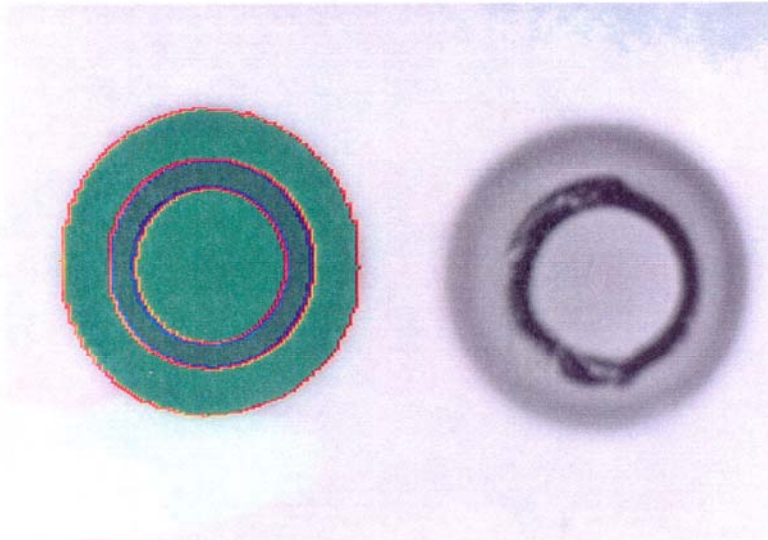


A

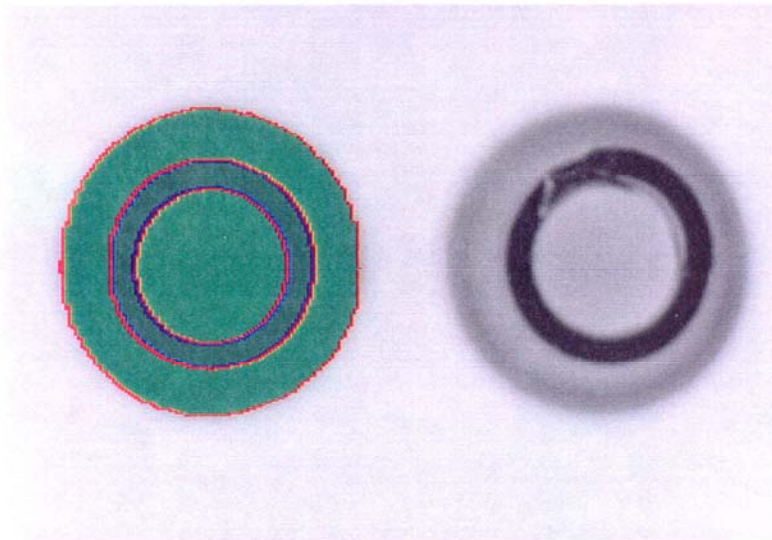


B

Figura IV.11 A) y B) Particiones iniciales generadas por el presegmentador para las imágenes mostradas en la figura IV.10.A) y B), respectivamente.



A



B

Figura IV.12 A) y B) Resultados finales donde se muestra la localización de la botella en buenas condiciones en cada una de las imágenes originales.

C. Identificación y Localización de Piezas Industriales.

El último caso de estudio que se presenta dentro de este apartado dedicado a la localización e identificación en entornos industriales es, fundamentalmente, un ejemplo de cómo se puede organizar la segmentación e interpretación de una imagen por etapas, progresando de lo "grueso", o más evidente, a lo "fino" o más incierto. Es este caso se parte de una disposición de piezas como la mostrada en la figura IV.14-A, con el objetivo de localizar sobre ella los posibles tornillos y tuercas. Para ello se organiza un análisis que consta de dos etapas diferenciadas. La primera de éstas está dirigida a localizar en la imagen las zonas correspondientes al brazo o rosca de cada tornillo, su zona más característica en este tipo de imagen. En la segunda fase, se procede a aislar las otras piezas presentes en la imagen, cuya delimitación es más confusa. Este esquema en dos etapas simplifica el diseño de aquellos programas que deben dirigir la segmentación, pues en cada una de ellas el objetivo, esto es "lo que se busca", está claramente diferenciado del "resto". Este hecho redundará también en una mayor robustez y estabilidad de la segmentación.

En la primera fase se parte de la imagen original para obtener dos diagnósticos en el nivel de los pixels, uno de ellos, "screws100", es simplemente una renormalización de la imagen de intensidades en 100 niveles. El segundo identifica a los pixels situados en zonas que presentan una determinada frecuencia espacial, característica de las roscas de los tornillos. Como medida de la frecuencia espacial se toma el número máximo de cruces por cero (Procedure *Dir4CrossDist*) detectados en el mapa de "offset" de la imagen. La evaluación del número de cruces por cero se realiza en una ventana centrada en cada pixel y a lo largo de las cuatro direcciones principales. El resultado es el mapa diagnóstico denominado "OffZeroCross" mostrado en la figura IV.14-B donde las zonas de mayor respuesta se localizan efectivamente sobre la rosca de los tornillos. A continuación, el presegmentador define el perfil de la imagen combinando este mapa de diagnósticos con el mapa "screws100" antes aludido y extrae las zonas en las que se encuentra el brazo roscado del tornillo, prototipo "screw_arm" (figura IV.15-A). Las pequeñas zonas, irrelevantes por su tamaño, que pueden aparecer en esta primera segmentación se filtran por las reglas de control del Procesador de Segmentos para producir, finalmente un conjunto de

Capítulo IV

segmentos identificados como clase de segmentos "screw_arm" (figura IV.15-B). El flujo de datos de esta primera etapa está resumido en la figura IV.13.

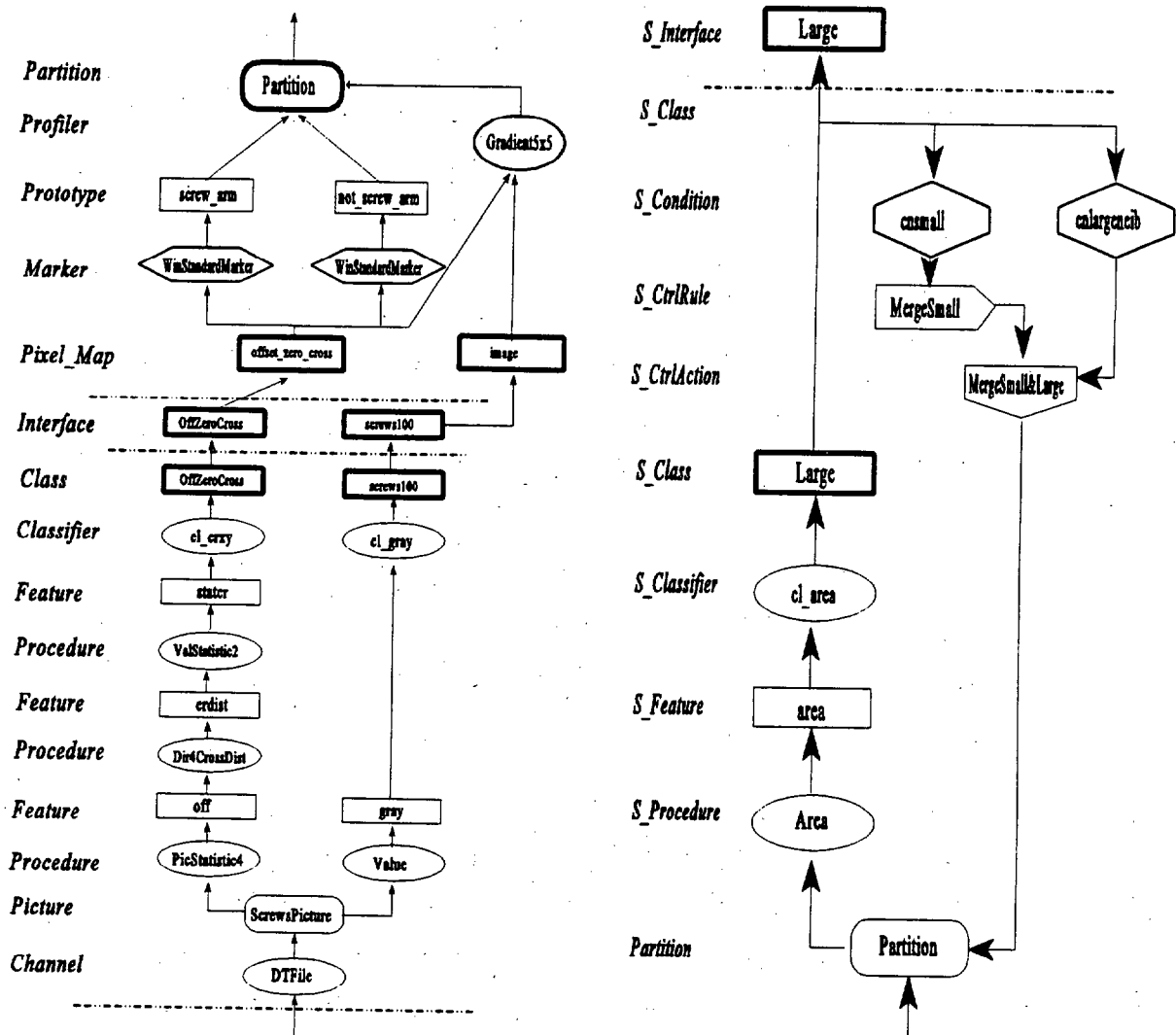


Figura IV.13 Flujo de datos en la primera fase del caso de estudio. Como resultado de esta fase se produce la localización de los brazos de los tornillos presentes en la imagen.

Finalizada esta primera etapa, un programa de usuario (que no se muestra en los diagramas) recibe del Procesador de Segmentos la clase de segmentos "screw_arm" y a partir de ellos define un nuevo mapa de pixels que contiene los pixels que pertenecen a alguno de estos segmentos. A continuación comienza la segunda fase del análisis, activándose un nuevo programa

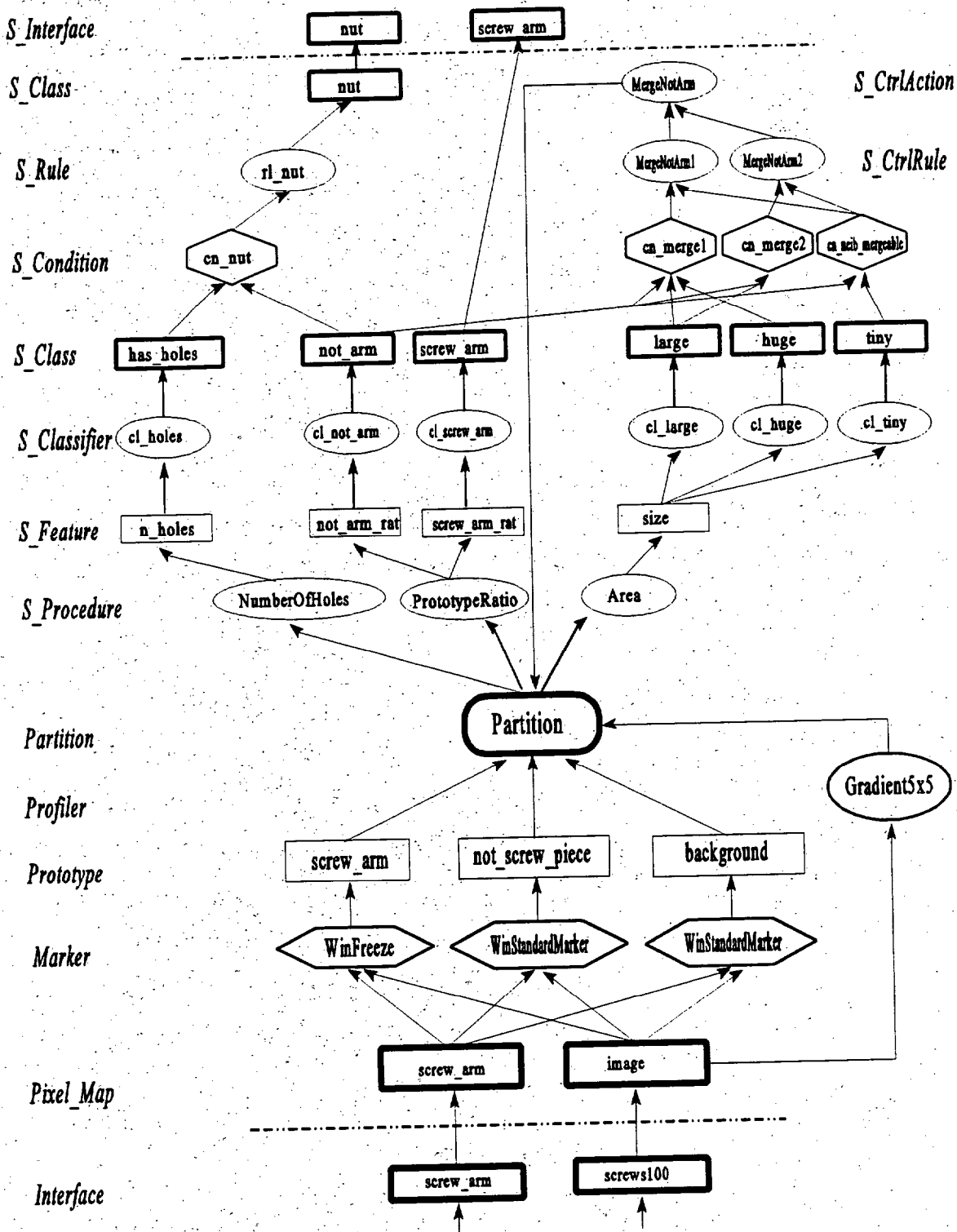
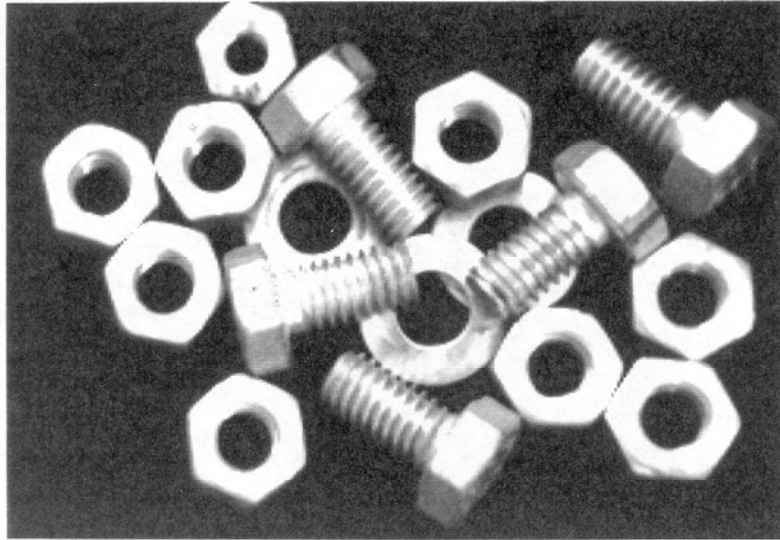


Figura IV.13 (cont.) Flujo de datos en el Procesador de Segmentos durante la segunda fase de la interpretación.

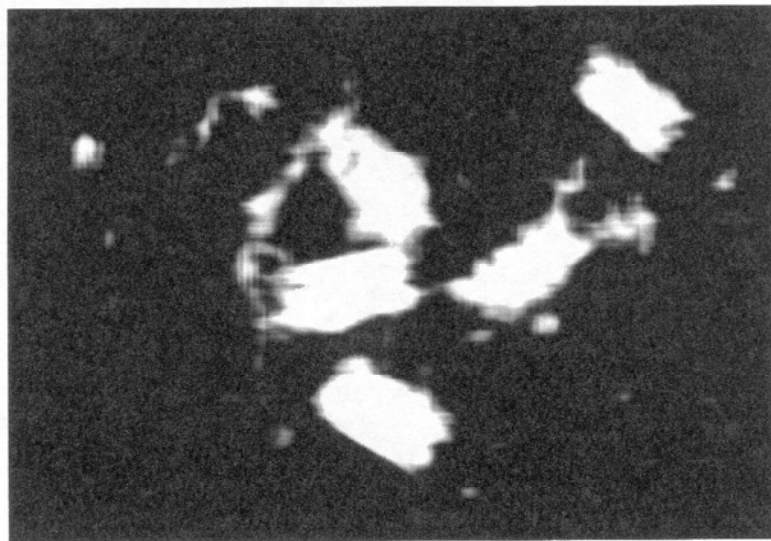
Capítulo IV

del Procesador de Segmentos. Utilizando el nuevo mapa de pixels ("screw_arm") y el mapa "screws100", se genera en el presegmentador una nueva partición inicial. El motivo de realimentar el resultado anterior sobre el presegmentador es el de "imponer" sobre la nueva partición la interpretación parcial obtenida en la fase anterior, la cual representa una primera evidencia de la presencia de determinadas clases [Drap-89]. El objetivo de esta segunda actuación del presegmentador es, por una parte, aislar zonas homogéneas correspondientes a piezas que no se hayan identificado como rosca de tornillo, y por otra impedir que las zonas identificadas en la primera fase, cuya localización es aceptable, participen en el proceso de inundación o crecimiento de regiones. Para lograr el primer objetivo se utiliza la opción *Not_Merge* en la declaración del prototipo "not_screw_piece" (véase el código en el Apéndice E), lo que impide que las cuencas de atracción de diferentes mínimos de un mismo prototipo puedan fundirse. Este hecho resulta en la delimitación de zonas homogéneas separadas por un cierto nivel de perfil, que en la imagen corresponde a bordes de las piezas o a sombras y reflejos. La inhibición de las zonas roscadas (prototipo "screw_arm") en el proceso de inundación se consigue empleando el marcador "WinFreeze". La partición resultante de este proceso se muestra en la figura IV.16-A.

Esta segunda partición se analiza para extraer un conjunto de diagnósticos de segmentos que conduzcan finalmente a la identificación de los tornillos y las tuercas presentes en la imagen. La estrategia seguida en la interpretación de esta segunda partición en el Procesador de Segmentos consiste primero en localizar los segmentos correspondientes al interior de las tuercas y, a continuación, fusionar pequeñas zonas irrelevantes con otras más significativas. La partición final resultante es la mostrada en la figura IV.16-B y los segmentos identificados como "brazo de tornillo" o como "tuerca" están indicados en la figura IV.17. El código que se ha utilizado en la implementación de este caso de estudio se incluye en el Apéndice E por razones de extensión.

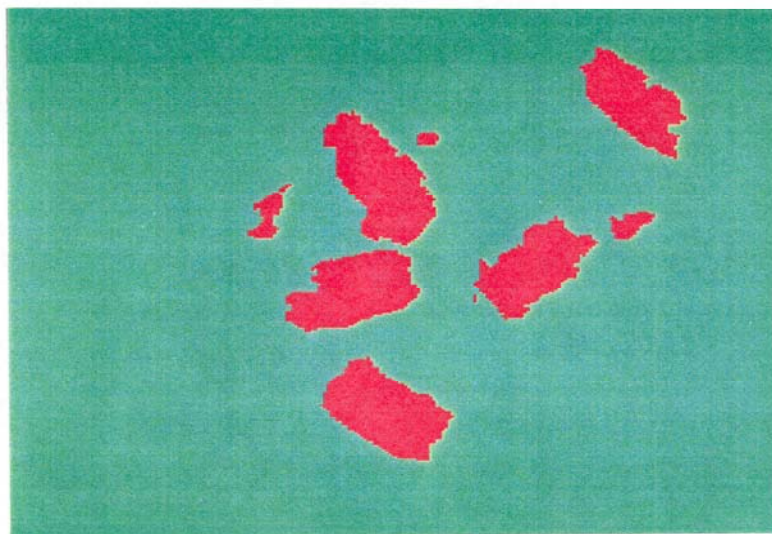


A

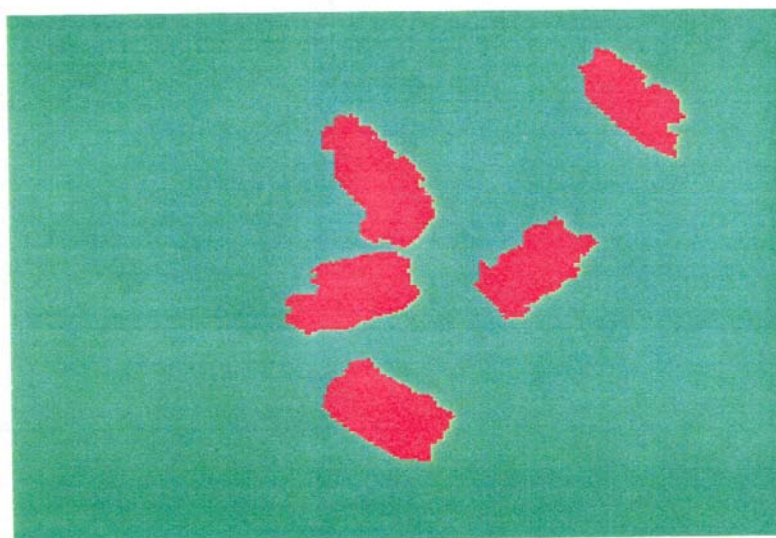


B

Figura IV.14 A) Imágen original. B) Mapa de diagnósticos de píxels OffZeroCross.

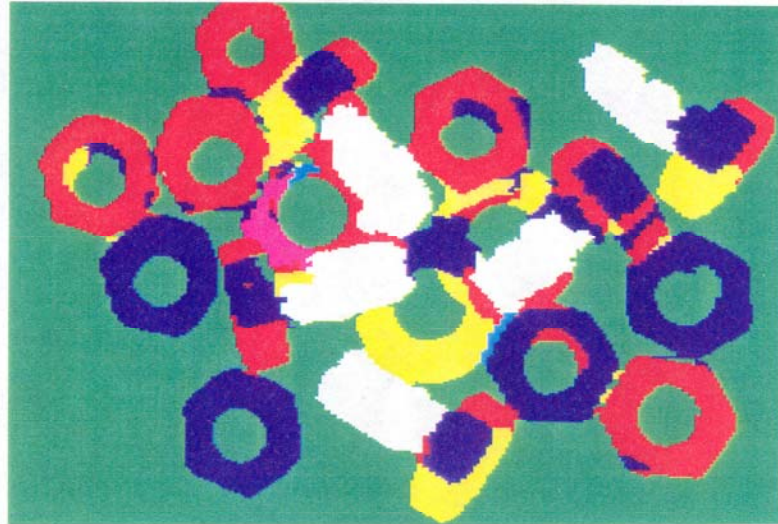


A

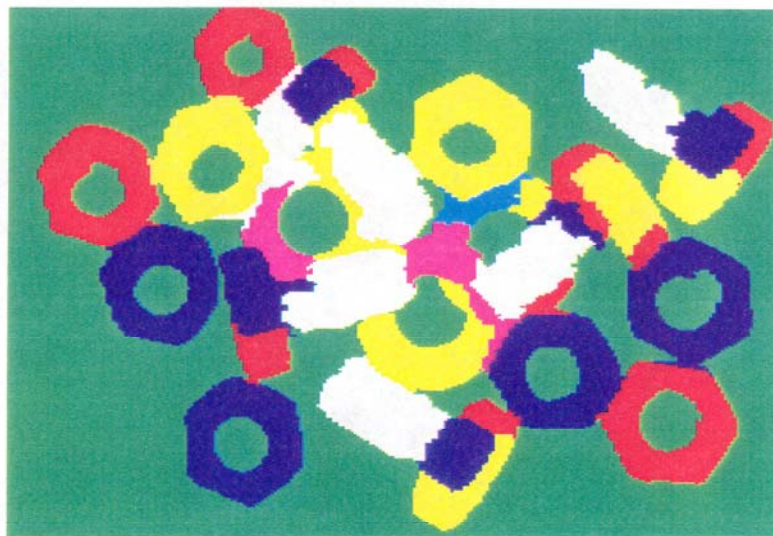


B

Figura IV.15 A) Partición inicial que muestra en rojo las zonas identificadas como posibles "brazos roscado" de tornillo. B) Partición resultante de la primera acción del Procesador de Segmentos orientada a refinar la partición inicial eliminando los segmentos de pequeña extensión.

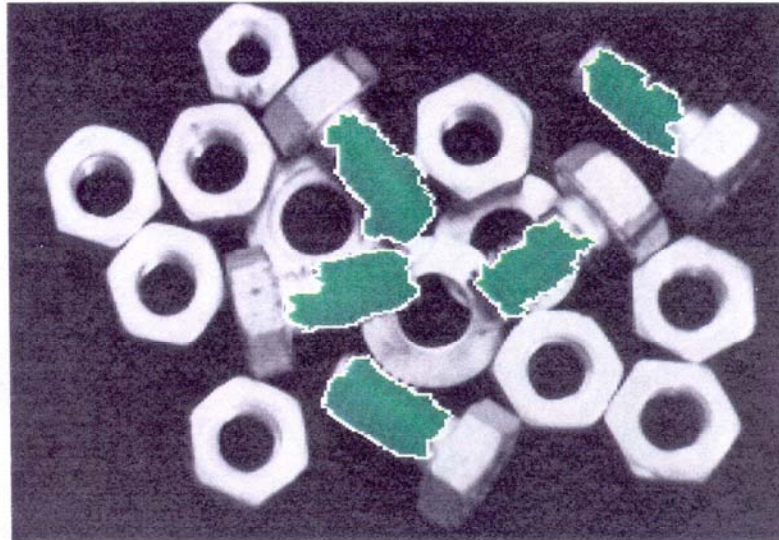


A

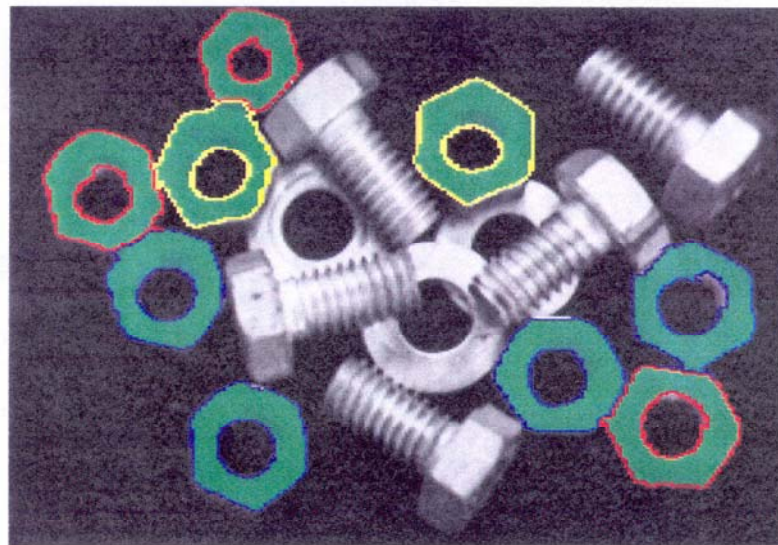


B

Figura IV.16 A) Segunda partición inicia orientada a obtener una segmentación que detalle todas las piezas presentes en la imagen original (Fig.IV.14-A). B) Partición final.



A



B

Figura IV.17 A) Segmentos identificados como "tornillos" en la partición final.
B) Segmentos etiquetados como "tuercas".

IV.3.2 SEGMENTACIÓN DE CONTORNOS.

En el caso de estudio descrito en el apartado IV.2.2, dedicado a la detección de movimiento, se describió cómo era posible articular un proceso de detección de contornos en el nivel de los pixels en base a operaciones estadísticas simples. En el caso de estudio que se presenta a continuación, se utiliza exactamente el mismo esquema para detectar los contornos presentes en la imagen en color (RGB) que muestra la figura IV.19. La única diferencia radica en que este proceso de detección se aplica aquí sobre cada una de las componentes de la imagen. El objetivo final es el de generar una segmentación de los contornos en dos tipologías de segmentos: "líneas rectas" y "arcos".

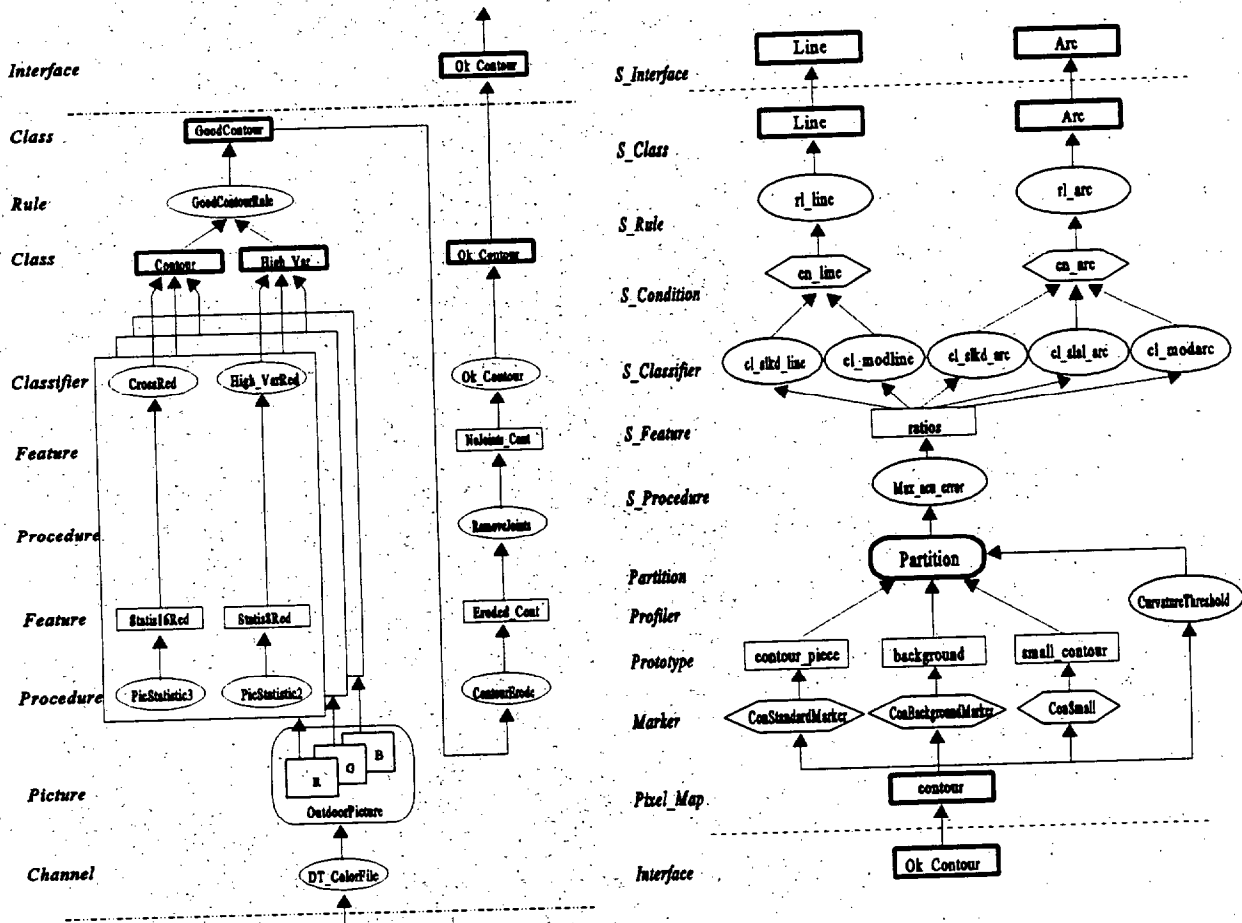


Figura IV.18 Flujo de datos en el caso de estudio sobre segmentación de contornos.

Capítulo IV

El primer paso es la detección de contornos que se realiza en el nivel de los pixels siguiendo un proceso análogo al descrito en el apartado IV.2.2. La única diferencia estriba que en este caso la detección de cruces por cero y de zonas de alta variación espacial se lleva a cabo sobre cada una de las componentes de la imagen y los resultados se combinan en las clases de pixels "Cross" y "HighVar". La clase resultante, "GoodContour", se realimenta como característica para sufrir un proceso de adelgazamiento de contornos y de eliminación de "puntos múltiples" (branch points). Como resultado de este proceso se obtiene el diagnóstico final "Ok_Contour" (figura IV.20).

Este caso de estudio ilustra la aplicación del presegmentador "Flood" en la segmentación de un mapa de contornos. En estas circunstancias, la acción combinada de los operadores que definen el perfil, basados típicamente en alguna medida de la curvatura de los contornos, y de los marcadores de los diferentes prototipos, sirve para definir un escenario donde el proceso de inundación se produce normalmente a lo largo de los contornos. En el ejemplo que se describe, el objetivo es segmentar los contornos en los puntos de máxima curvatura de manera que los segmentos que aparezcan en la partición puedan ser clasificados como "lineas" o "arcos". Para generar el perfil básico del mapa de contornos se utiliza el "Profiler" denominado *CurvatureThreshold* sobre el mapa de diagnóstico "Ok_Contour". Este operador requiere de tres parámetros: número de puntos del soporte para el cálculo de la M-Curvatura, la longitud mínima de los contornos para los que se calcula la curvatura y la curvatura umbral en valor absoluto equivalente a curvatura nula. El perfil así generado es nulo en todos los puntos que no forman parte de algún contorno o en aquellos que la longitud del contorno al que pertenecen es menor que la longitud umbral declarada en *CurvatureThreshold*. En cualquier caso, éstos son puntos cuyo etiquetado como "fondo" es evidente. Para ello se definen dos prototipos, "background" y "small_contour", que detectan estos tipos de puntos en el mapa de contornos "Ok_Contour" con la ayuda de los marcadores, "ConBackGroundMarker" y "ConSmall". Este último marcador necesita de un parámetro que indique la longitud máxima del segmento que debe ser considerado pequeño o irrelevante. Nótese en el código que acompaña a esta descripción cómo a ambos prototipos se les asigna un misma etiqueta de color lo que produce de hecho su fusión en un único segmento en la partición resultante. Los marcadores empleados con estos prototipos producen

una modificación del perfil en estos puntos situándolos por encima del nivel máximo normalizado (255) lo que inhibe de hecho su participación en el proceso de inundación. La definición del prototipo "contour_piece" responde a los objetivos trazados en este caso para el presegmentador. La detección de los mínimos de este prototipo se realiza con el marcador "ConStandardMarker" al que se fijan dos parámetros. El primero de estos es el número mínimo de puntos "contorno" que estando situados por debajo de un cierto nivel umbral rodean a un cierto punto "contorno", y el segundo de los parámetros es el aludido nivel umbral. Este marcador permite detectar zonas de contorno de baja curvatura que tengan una cierta relevancia perceptual en razón de su extensión. La partición resultante se muestra en la figura IV.21. Los puntos correspondientes a los prototipos "small_contour" y "background" aparecen en verde oscuro constituyendo el fondo de la imagen. Puede observarse que el proceso de inundación permite resolver con bastante exactitud el etiquetado de los contornos en las zonas de mayor curvatura.

La identificación final de los segmentos resultantes como "línea" o "arco" se lleva a cabo mediante un subconjunto de los descriptores presentados en [Wu-92]:

a) *Proporción entre la longitud del segmento y la distancia entre sus puntos extremos (SLKD).*

$$SLKD = S / D$$

donde S representa la longitud del segmento calculada a partir de la codificación del borde en código de cadena y D es la distancia euclídea entre los puntos extremos del segmento. Este descriptor debe ser próximo a uno en el caso de una línea y mayor que uno en otro caso.

b) *Proporción entre la longitud del segmento y la longitud del arco que une sus puntos extremos (SLAL).*

$$SLAL = S / L$$

donde S tiene el mismo significado y L se corresponde con la longitud del arco de circunferencia que une los puntos extremos del segmento. Si el segmento es un arco de circunferencia de forma que su longitud puede calcularse a partir de

$$L = R\theta$$

Capítulo IV

donde R es el radio del arco y θ es el ángulo definido por la dos cuerdas que unen el centro del arco con sus extremos, entonces R y θ se obtienen de las siguientes expresiones cuya deducción es inmediata:

$$R = \frac{(D/2)^2 + h^2}{2h}$$
$$\Theta = 2 \arctan \left(\frac{D/2}{R-h} \right)$$

siendo h es la distancia entre el punto medio (x_{cc}, y_{cc}) de la cuerda que une los extremos del segmento y el punto medio del segmento (x_{cs}, y_{cs}) , es decir,

$$h = \sqrt{(x_{cs} - x_{cc})^2 + (y_{cs} - y_{cc})^2}$$

c) Error acumulado (*modline*, *modarc*).

Este descriptor tiene dos expresiones en función de que el segmento sea una línea o un arco. En el primer caso (*modline*) se obtiene como la suma de las distancias de todos los puntos del segmento a la recta que une los extremos del mismo, y, en el segundo, como la suma de las distancias de todos los puntos del segmento al arco de circunferencia obtenido en el descriptor anterior. Las expresiones correspondientes son:

$$E_{modline} = \sum_{i=1}^N \frac{|Ax_i + By_i + C|}{\sqrt{A^2 + B^2}}$$
$$E_{modarc} = \sum_{i=1}^N \left| \sqrt{(x_i - x_c)^2 + (y_i - y_c)^2} - R \right|$$

donde N es el número de puntos del segmento, y A , B y C son los coeficientes de la recta que une los puntos extremos del mismo. La combinación de estos descriptores permite definir de forma robusta las clases de segmentos "Line" y "Arc" con los resultados que se muestran, respectivamente, en las figuras IV.22-A y IV.22-B.



Figura IV.19 Imagen original en el caso de estudio sobre segmentación de contornos.

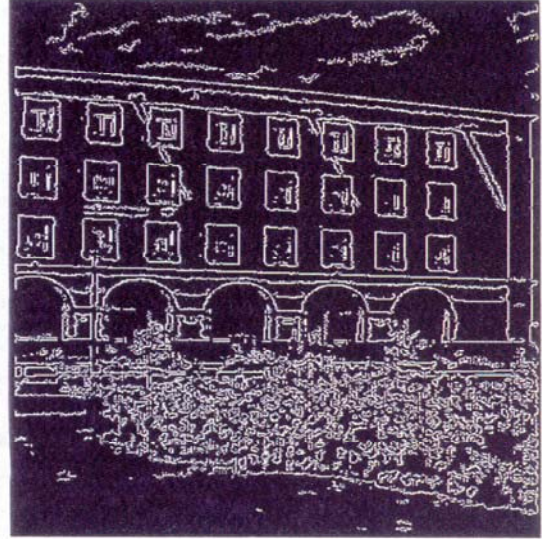


Figura IV.20 Mapa de diagnóstico de pixels "Ok_Contour".

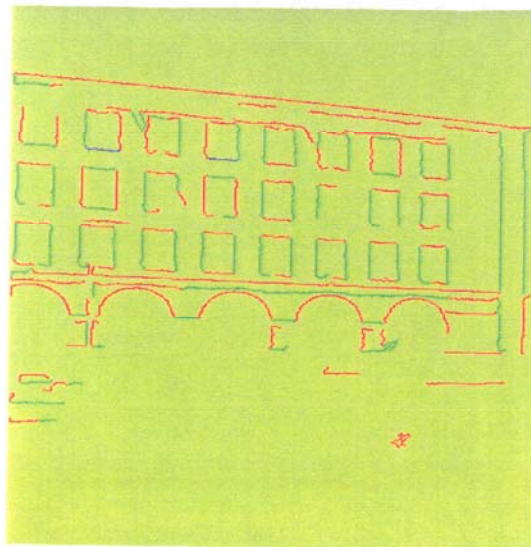


Figura IV.21 Partición resultante generada por el presegmentador.

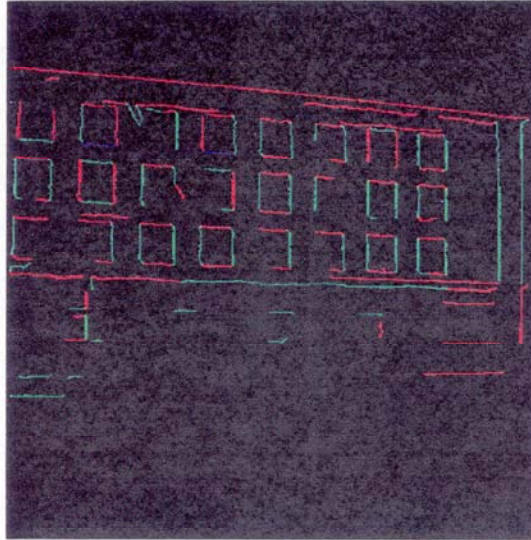


Figura IV.22-A Segmentos de la clase "Line" identificados en la partición.

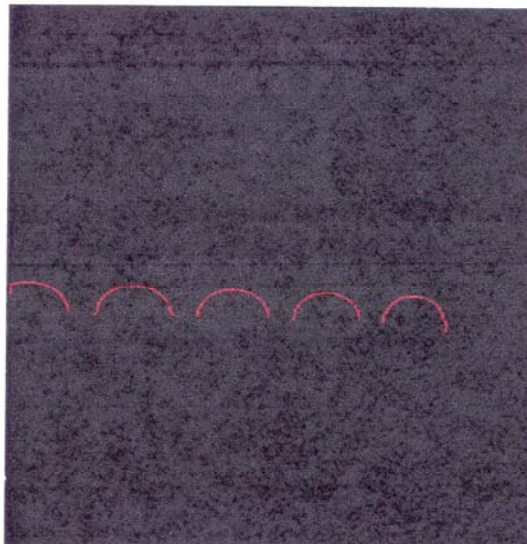


Figura IV.22-B Segmentos de la clase "Arc" identificados en la partición.

IV.3.3 SEGMENTACIÓN DESDE PROPIEDADES MORFOLÓGICAS.

El presente caso de estudio está inspirado en un problema real tomado del dominio de la sedimentología. Éste consiste en la obtención de la forma aproximada de las partículas presentes en una muestra como la que se presenta en la figura IV.25-A, para obtener un conjunto de medidas como el radio, forma aproximada, etc. La dificultad radica, como puede apreciarse en la citada figura, en la adyacencia de las partículas. Esto hace que el primer paso del análisis deba enfocarse hacia la recuperación de la forma aproximada de dichas partículas.

Como en el caso descrito anteriormente sobre localización de piezas industriales (tornillos y tuercas) SVEX permite la articulación de una estrategia en dos etapas. La primera y más sencilla está dirigida a aislar el conjunto de partículas del fondo de la imagen. Tarea aparentemente trivial, un simple umbralizado no resulta adecuado ya que las zonas de la imagen que corresponden a las partículas no son absolutamente homogéneas, debido a la forma tridimensional de las mismas. El esquema seguido está ilustrado en la figura IV.23, donde se describe el flujo de datos en el Procesador de Pixels, y en la parte inferior izquierda de la figura IV.24, donde se muestra la organización del presegmentador que produce la partición inicial de la figura IV.25-B.

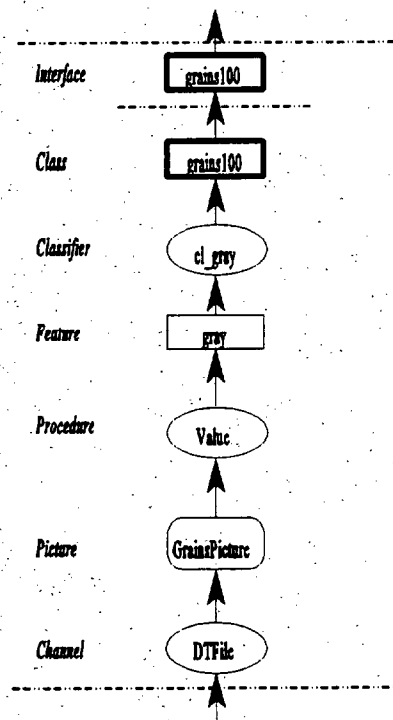


Figura IV.23 Flujo de datos en el Procesador de Pixels para este caso de estudio.

Una vez obtenida esta primera segmentación, un programa de usuario convierte la partición inicial en un nuevo mapa de pixels y activa un segundo programa del Procesador de Segmentos, organizado como se indica en la figura IV.24. En esta ocasión, el presegmentador "Flood" obtiene el perfil a partir del gradiente morfológico de las zonas que en la partición inicial

Capítulo IV

corresponden a las partículas. Durante el proceso de inundación se hace crecer a las partículas desde los mínimos del gradiente, al tiempo que el marcador seleccionado para el prototipo "background" impide que el fondo participe en el proceso de inundación. El resultado de este proceso o partición final es el mostrado en la figura IV.26-A, donde aparecen delineadas aproximadamente las partículas "visualmente" más evidentes. Las zonas peor delimitadas son las que corresponden a la frontera entre dos partículas, donde evidentemente la incertidumbre acerca de la forma originaria es absoluta. Finalmente, el Procesador de segmentos computa un conjunto de descriptores de formas y selecciona aquellas partículas de aspecto redondeado, tal y como muestra la figura IV.26-B.

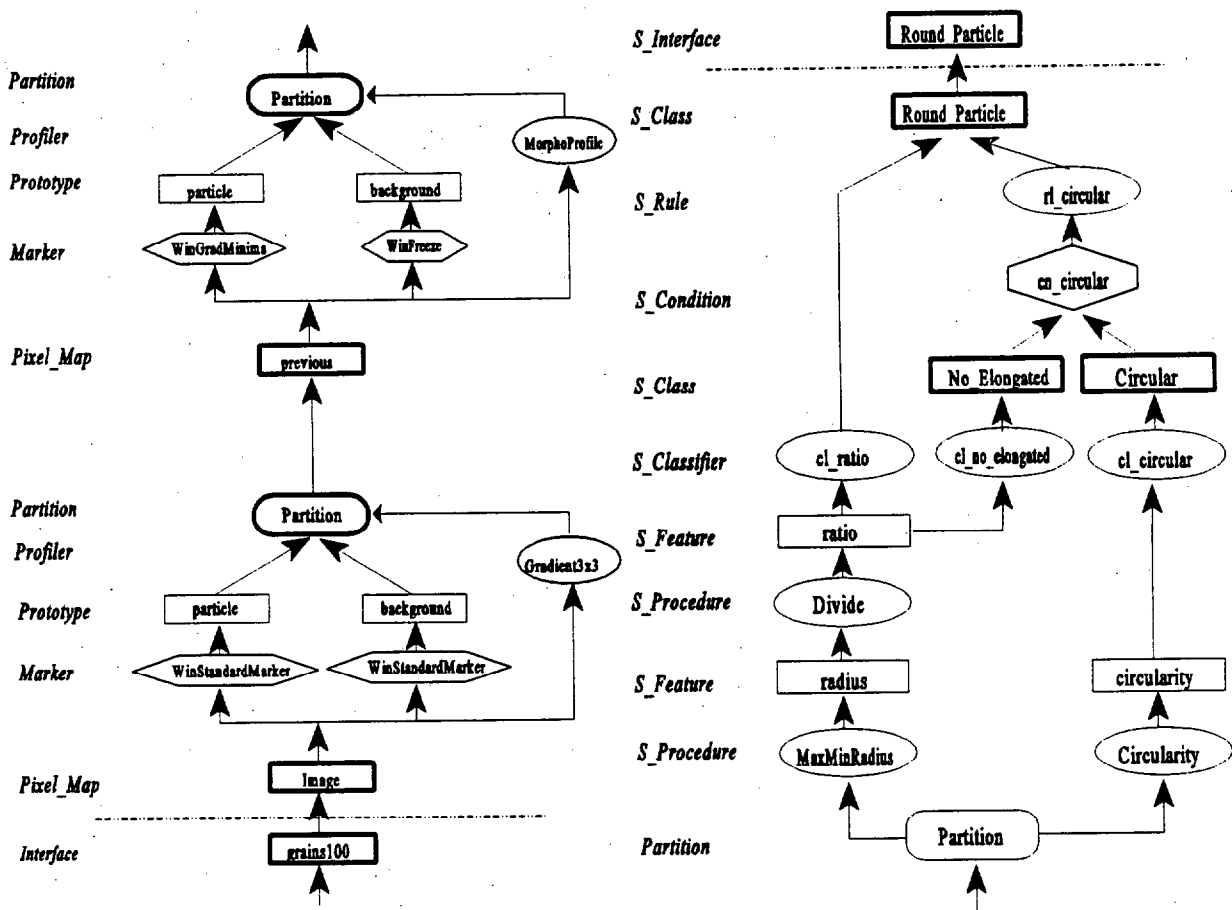
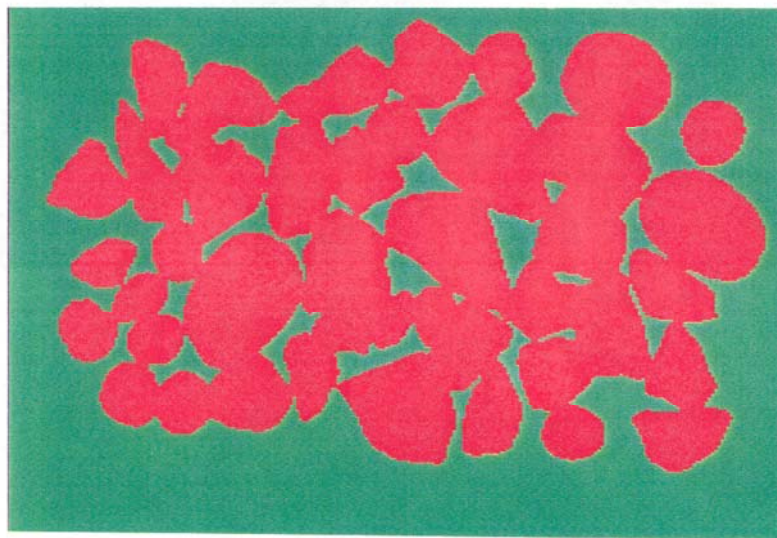


Figura IV.24 Flujo de datos en el Procesador de Segmentos para este caso de estudio.

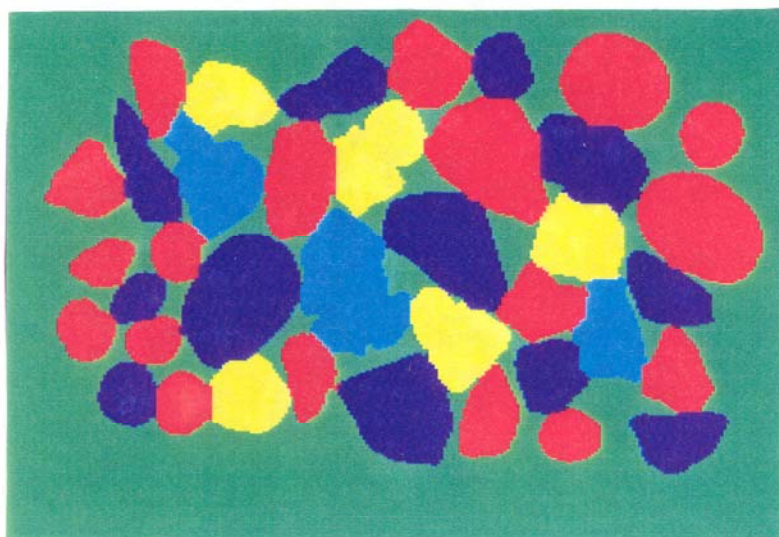


A

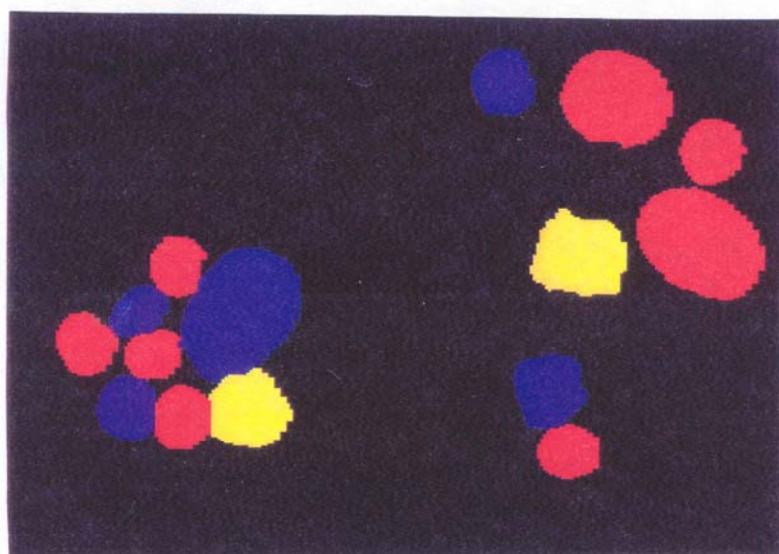


B

Figura IV.25 A) Imagen original del caso de estudio. B) Primera partición.



A



B

Figura IV.26 A) Partición final. B) Identificación de partículas redondeadas.

IV.3.4 SEGMENTACIÓN DE IMÁGENES DE EXTERIOR.

El último de los casos de estudio que se presenta es el más rico y complejo de cuantos se han descrito en este trabajo. Pretende ser una primera exploración de SVEX en la segmentación de imágenes que, aún perteneciendo a un mismo contexto, presentan un aspecto y contenido muy diferente. El conjunto de imágenes utilizado está integrado por las seis imágenes de exterior (RGB) que se muestran en la figura IV.31. El análisis de estas imágenes no está dirigido a obtener una segmentación minuciosa y detallada, sino a lograr una caracterización robusta de los elementos más relevantes de las mismas que permita orientar la búsqueda de otros diagnósticos sobre hipótesis fiables. Para generar los resultados que se muestran en las figuras se ha utilizado en todos los casos el mismo programa y con la misma selección de parámetros. Por razones de extensión el código de estos programas se ha incluido separadamente en el Apéndice E.

El Procesador de Pixels proporciona cinco mapas de diagnóstico, entre ellos el denominado "*color_grad*" que representa el gradiente de la imagen RGB y se utiliza como perfil en el presegmentador. Los restantes mapas de pixels son los correspondientes a los diagnósticos "*green*", "*blue*", "*artif*" y "*artif_shadow*". Los dos primeros identifican respectivamente pixels verdes o azules, normalmente asociados a las zonas de follaje o hierba y de cielo de las imágenes. Los diagnósticos "*artif*" y "*artif_shadow*" se asocian con zonas de las imágenes que no presentan un color definido sino que son, respectivamente, claras y oscuras. La definición de estas clases de pixels está basada en la obtención de las componentes I1, I2 e I3 a partir de las componentes RGB. El flujo de datos seguido en el nivel de los pixels se ilustra en la figura IV.27.

El presegmentador toma el mapa de diagnóstico "*color_grad*" como el perfil de la imagen y se limita simplemente a realizar un crecimiento de regiones combinando los mapas de diagnóstico obtenidos del Procesador de Pixels y mencionados anteriormente. Un aspecto a destacar es el uso de la opción *Not_Merge* en la definición de los prototipos con el objetivo de obtener una partición inicial con un cierto nivel de detalle. La figura IV.28 contiene el esquema seguido en la definición de los prototipos y de la partición inicial. Las particiones resultantes se muestran en la figura IV.32.

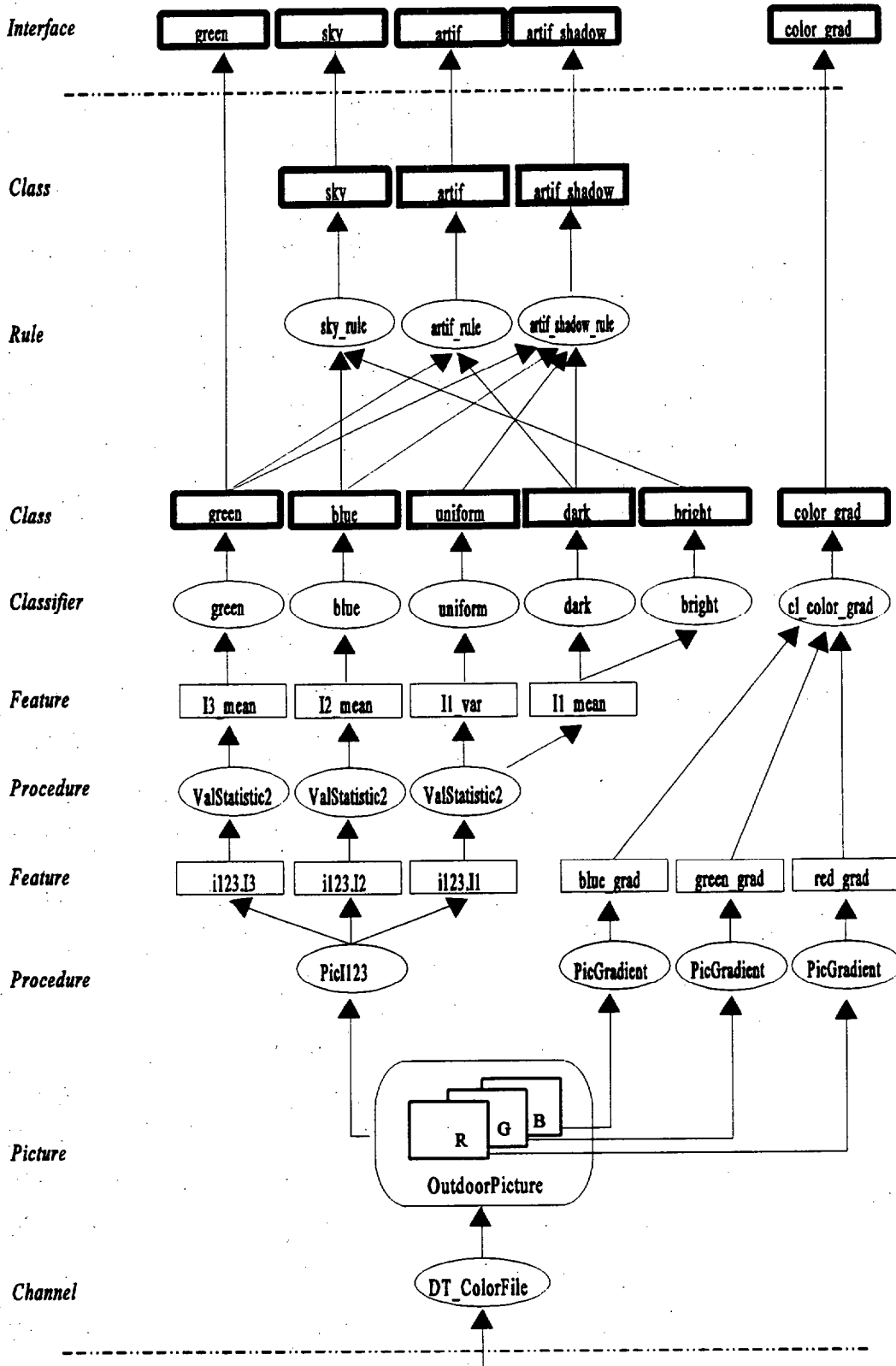


Figura IV.27 Flujo de datos en el Procesador de Pixels.

Las clases de segmentos que se definen en el Procesador de segmentos están dirigidas a identificar en las imágenes entidades como ventanas ("window"), carreteras ("road"), zonas verdes ("green"), cielo ("sky") y construcciones ("front"). La definición de algunas de estas clases como

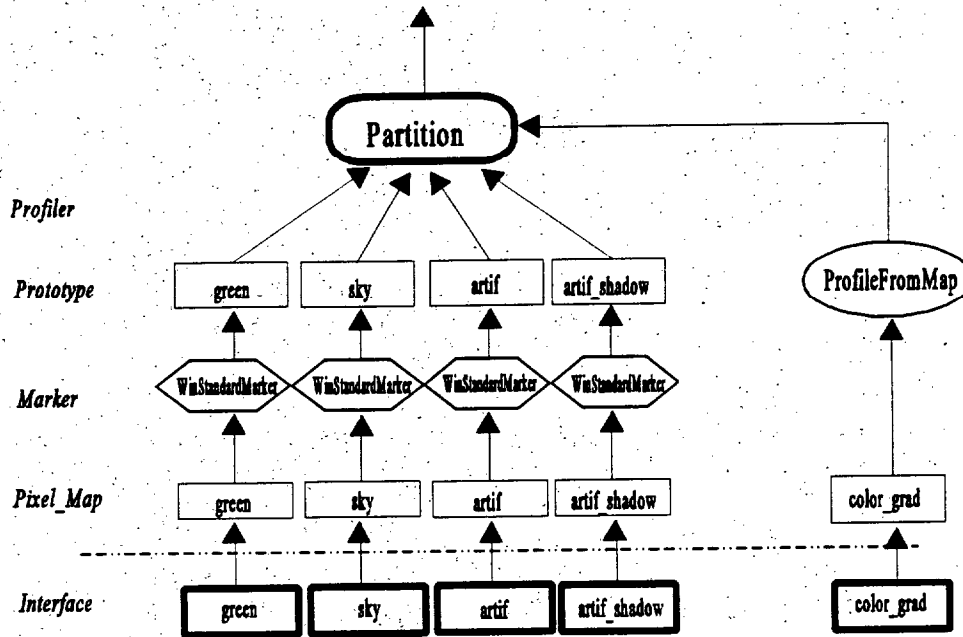


Figura IV.28 Flujo de datos en el presegmentador.

"green" o "sky" se obtiene básicamente de los prototipos utilizados en el presegmentador, mientras que otras clases, como "front", se derivan a partir de relaciones espaciales entre clases de segmentos menos elaboradas. En el caso que se describe, las reglas y acciones de control juegan un papel destacado a la hora de rehacer la partición inicial, normalmente bastante fragmentada. La figura IV.29 muestra las reglas de control (S_CtrlRules) y las acciones asociadas (S_CtrlActions) ordenadas de mayor a menor precedencia. La misma figura incluye las condiciones (S_Conditions) que participan en la definición de dichas reglas de control. Algunas de estas reglas son de evaluación incondicional (véase el Apéndice E), mientras que otras, como la regla "FindWindow", se condicionan a que el resultado de la acción de control maximice un

Capítulo IV

cierto diagnóstico. En este caso, el segmento originado por la fusión de otros dos debe ser "más ventana" que cualquiera de sus antecedentes. La figura IV.32 muestra las particiones resultantes tras la acción del Procesador de Segmentos, mientras que las figuras IV.34-37 contienen los segmentos identificados como pertenecientes a algunas de las clases de mayor contenido semántico como "front", "window", etc...

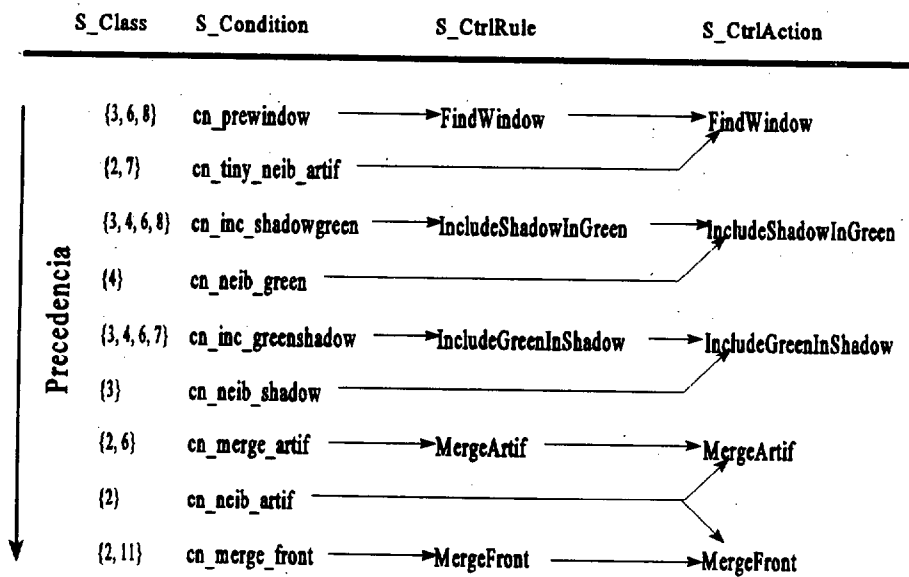


Figura IV.29 Organización del control en el Procesador de Segmentos. Los números entre llaves representan a clases que intervienen en la definición de las condiciones, según lo indicado en la figura IV.30. La flecha indica el sentido decreciente de la precedencia de las reglas de control.

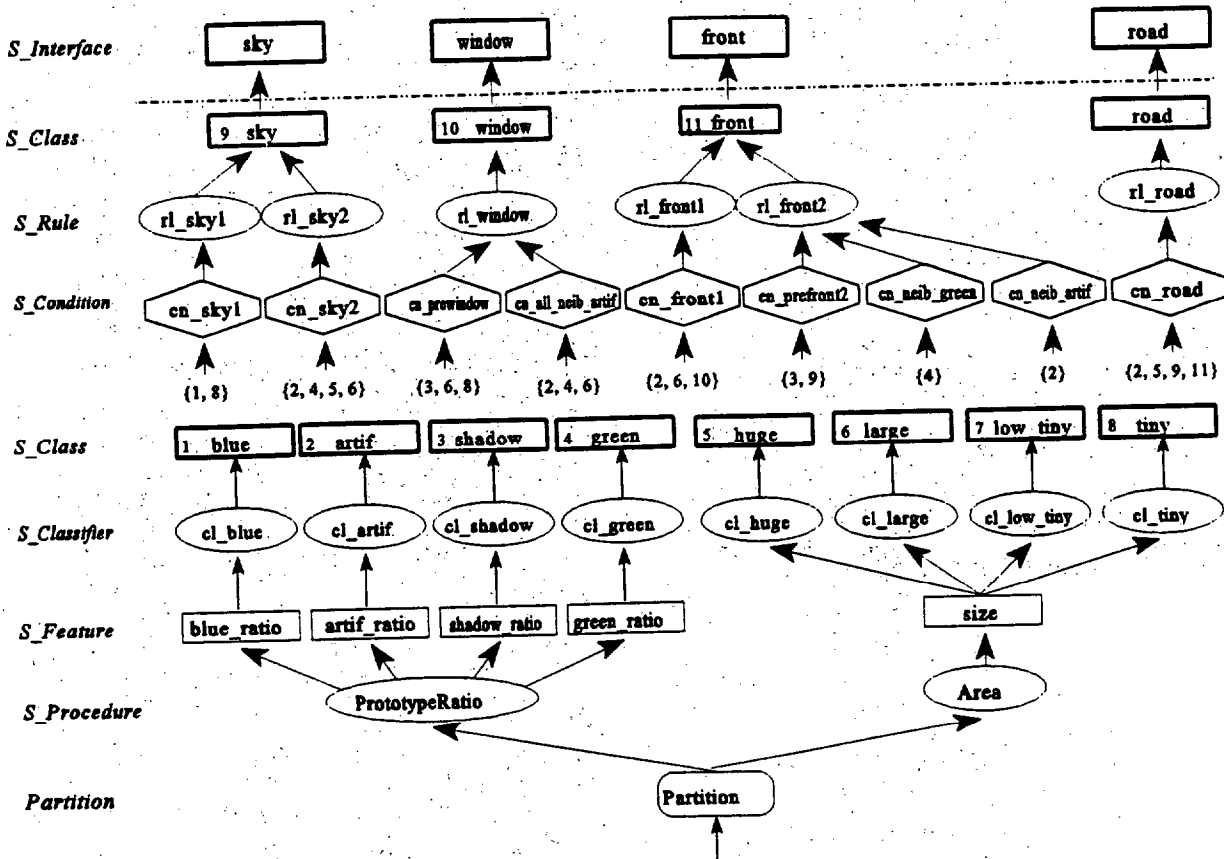


Figura IV.30 Definición de las clases de segmentos empleadas en este caso de estudio. Sólo se muestran en la figura las condiciones ("S_Conditions") que intervienen en la definición de alguna clase. En aras de una mayor claridad, las clases que participan en cada condición se indican por la secuencia de índices situados en la base de cada condición. El índice correspondiente a cada clase es el número que aparece dentro de los rectángulos.



A



B



C

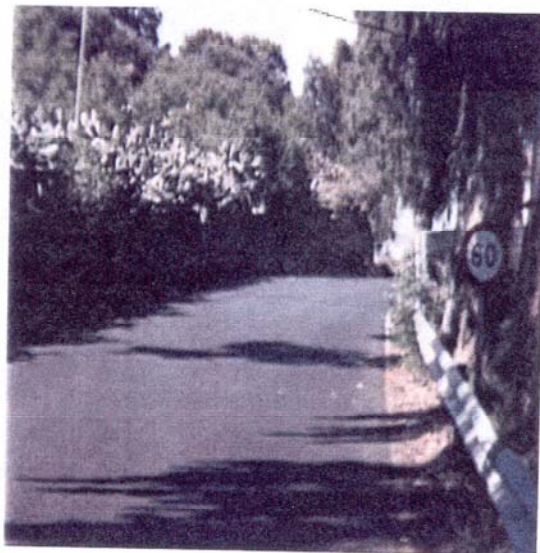


D

Figura IV.31 Imágenes utilizadas en este caso de estudio.

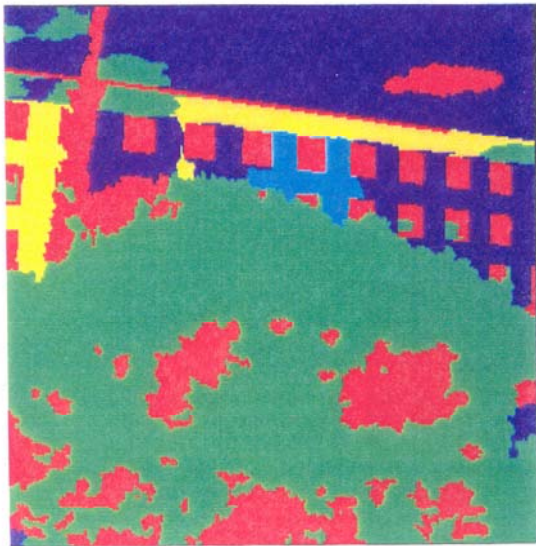


E

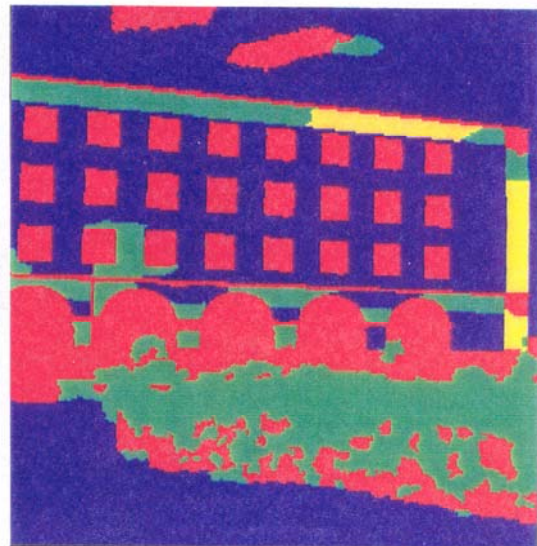


F

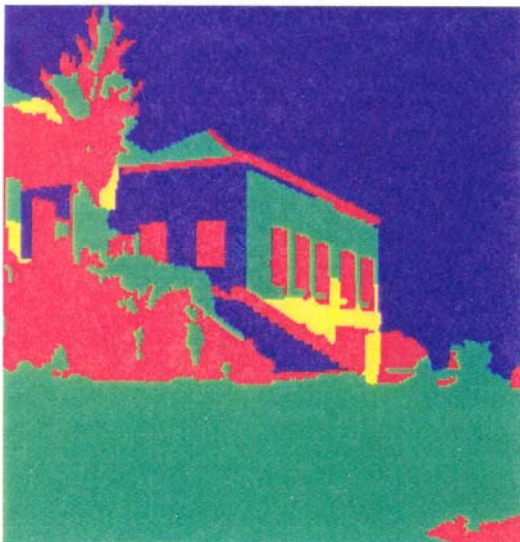
Figura IV.31 (cont.) Imágenes utilizadas en este caso de estudio.



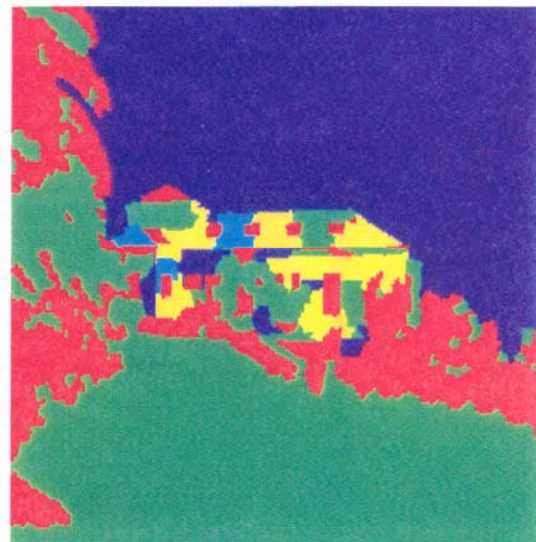
A



B

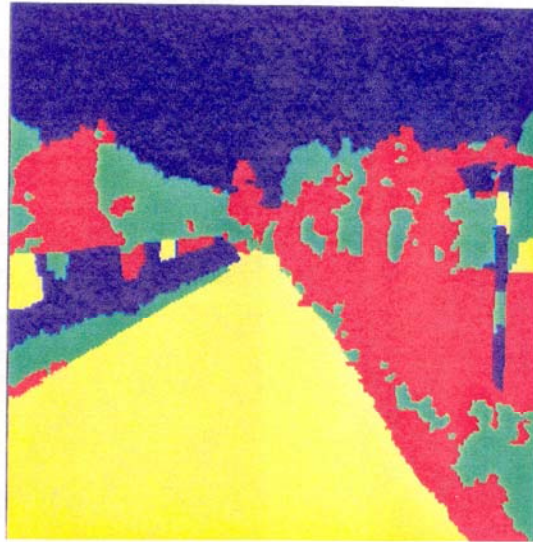


C

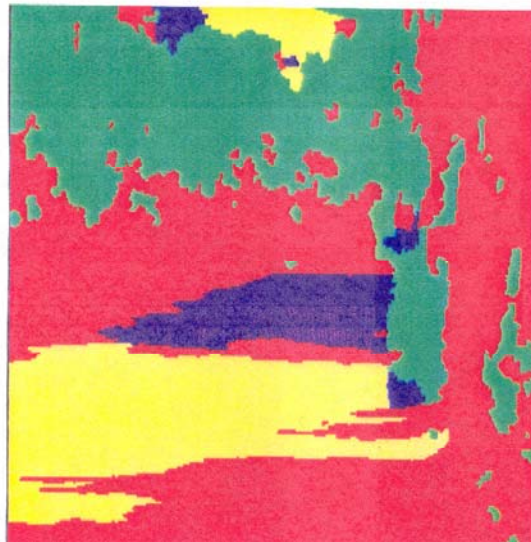


D

Figura IV.32 Particiones iniciales generadas por el presegmentador de las imágenes mostradas en la figura IV.31.

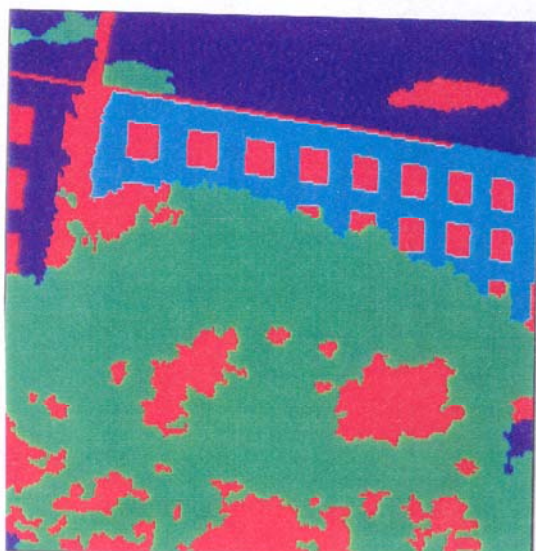


E

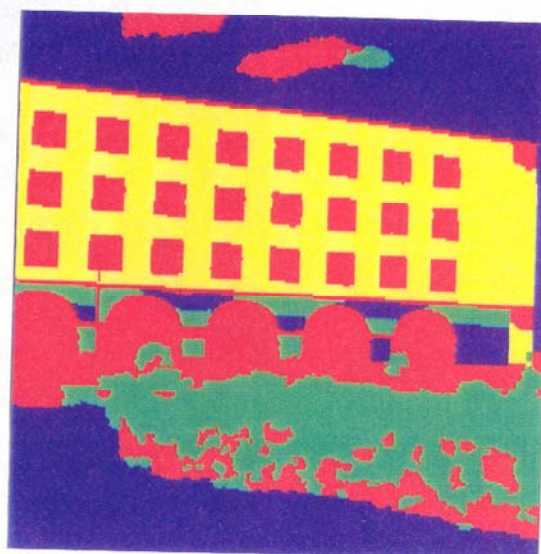


F

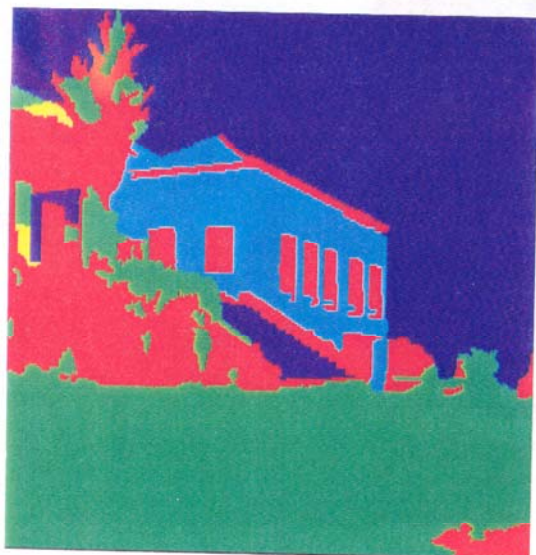
Figura IV.32 (cont.) Particiones iniciales generadas por el presegmentador de las imágenes mostradas en la figura IV.31.



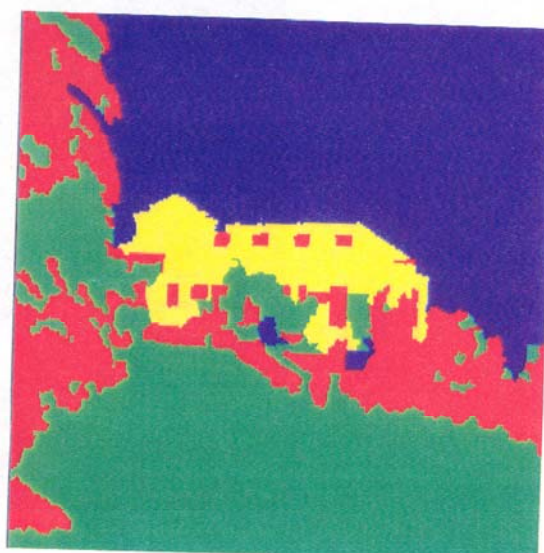
A



B



C

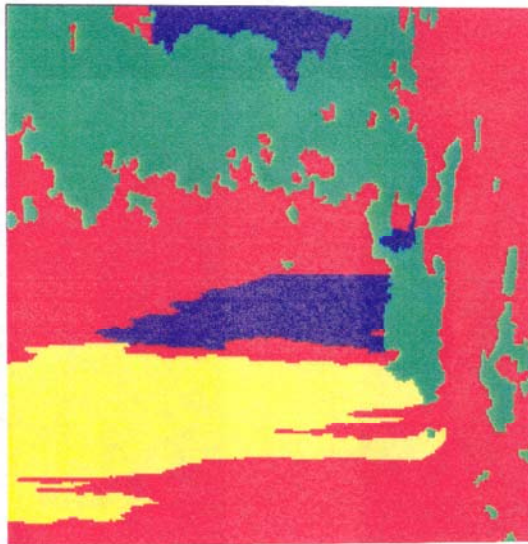


D

Figura IV.33 Particiones finales producidas por el Procesador de Segmentos.



E

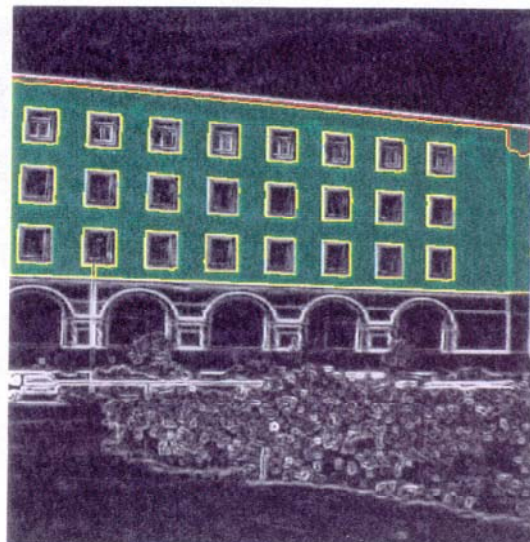


F

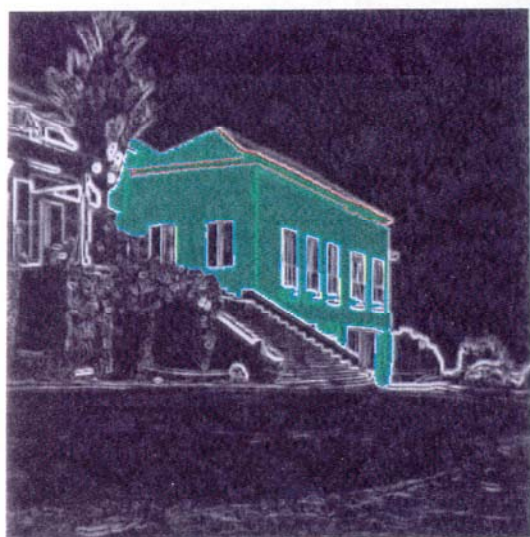
Figura IV.33 (cont.) Particiones finales producidas por el Procesador de Segmentos.



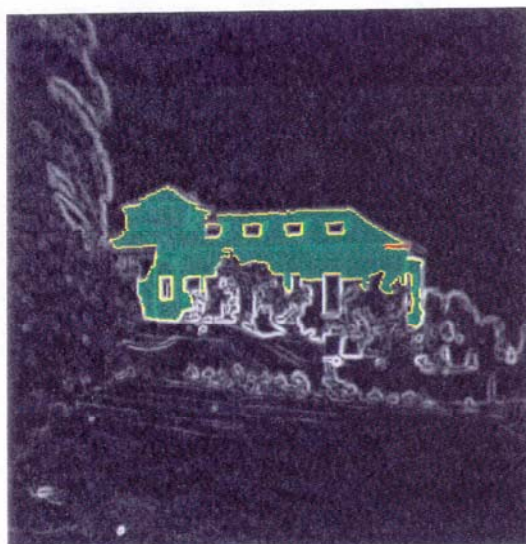
A



B

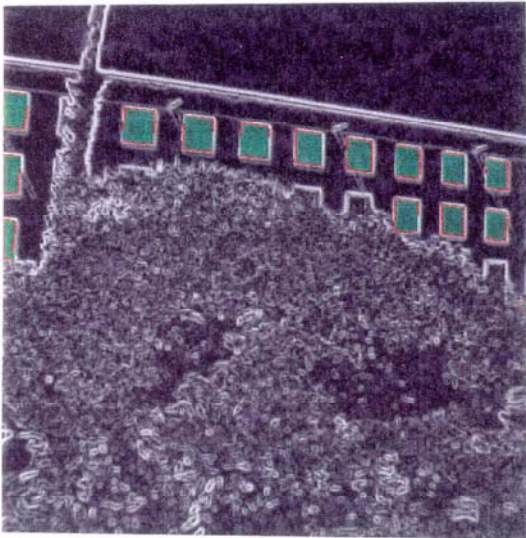


C

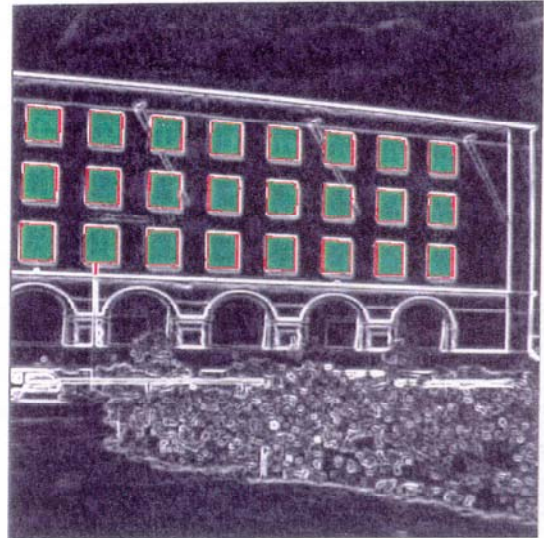


D

Figura IV.34 Segmentos de las clase "front" encontrados en algunas de las imágenes analizadas.



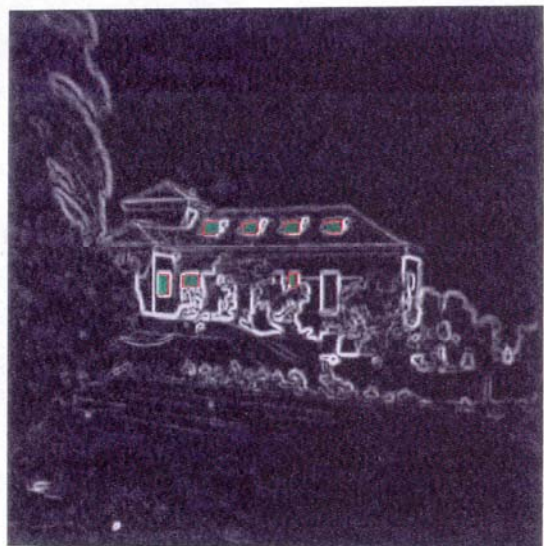
A



B

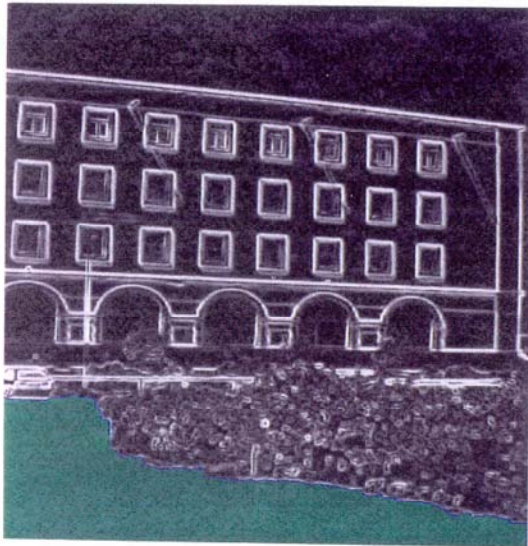


C

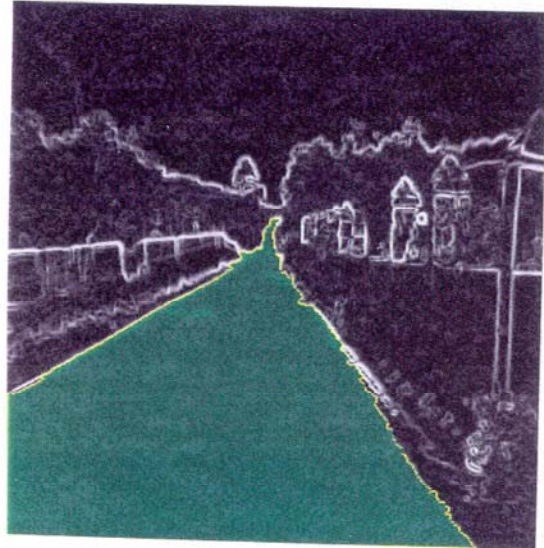


D

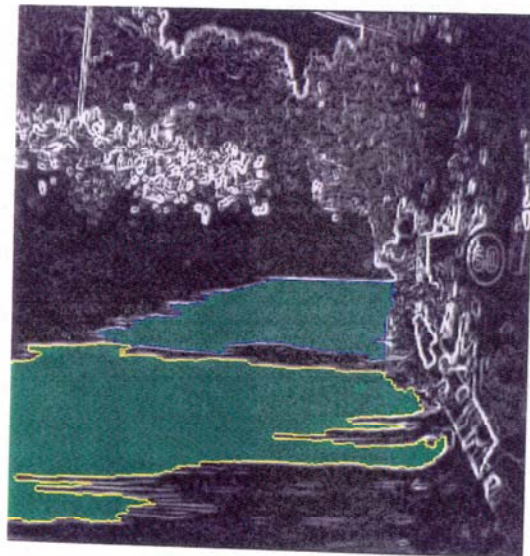
Figura IV.35 Segmentos de la clase "window" encontrados en algunas de las imágenes analizadas.



A

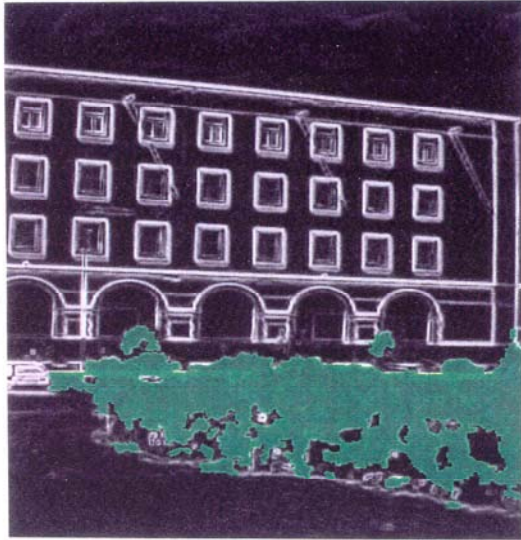


B

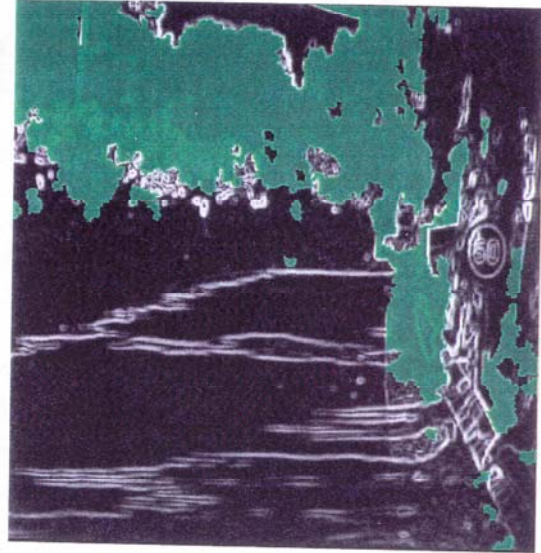


C

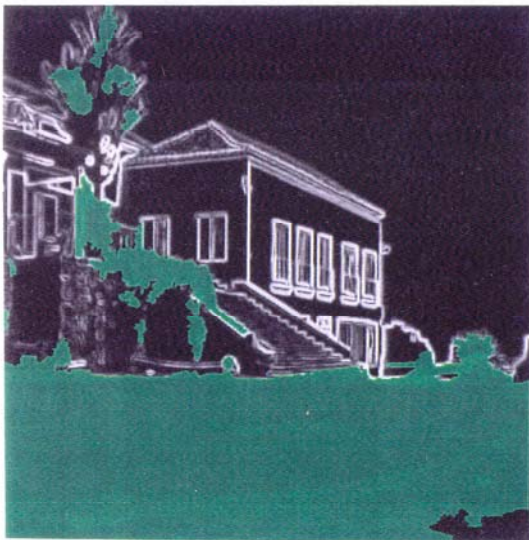
Figura IV.36 Segmentos de las clase "road" encontrados en algunas de las imágenes analizadas.



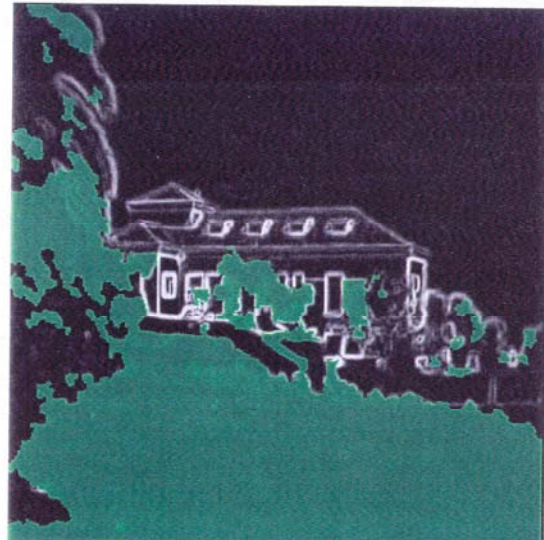
A



B



C



D

Figura IV.37 Segmentos de la clase "green" encontrados en algunas de las imágenes analizadas.

IV.4 DISCUSIÓN.

A través de los casos de estudio presentados en este capítulo se ha intentado ilustrar la aplicación de SVEX en la resolución de diferentes problemas de segmentación de imágenes. El objetivo de estos experimentos no ha sido el de demostrar la superioridad de SVEX frente a otros sistemas o algoritmos de segmentación, sino el de mostrar la aplicación de un nuevo esquema que permite resolver problemas de segmentación en contextos muy diferentes mediante mecanismos relativamente sencillos, y cuyas características más importantes responden a los objetivos de diseño trazados para SVEX (epígrafe II.3.1): organización por niveles, combinación de procesamiento simbólico con procesamiento numérico en cualquier nivel del sistema, manejo de la incertidumbre, explicitud del conocimiento necesario, definición de una arquitectura invariante donde el flujo de datos está claramente definido, disponibilidad de un lenguaje de programación que permite el desarrollo de aplicaciones con facilidad y rapidez, etc.

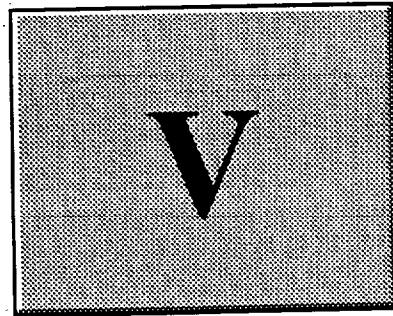
El desarrollo de estos experimentos ha servido también para mostrar algunas facetas de este sistema que, aunque no se han desarrollado plenamente en este prototipo, son potencialmente interesantes. Una de ellas es la posibilidad de definir un conjunto de conceptos primitivos, tanto a nivel de píxel ("ser-borde", "ser-azul", ...) como de segmentos ("ser-cuadrado", "ser-recto", ...), que constituirían una base de modelos y operadores susceptibles de ser utilizados en diferentes situaciones. En otros experimentos (casos IV.3.1.C y IV.3.3) se ha sugerido la conveniencia y utilidad de establecer mecanismos de realimentación (control top-down) entre los diferentes niveles del sistema. Mediante éstos es posible descomponer la solución de un problema, la segmentación e interpretación de una imagen, en etapas sucesivas organizadas para progresar desde lo más "evidente" a lo más "incierto". En cualquier caso, estas características de SVEX alcanzarán su verdadera potencialidad una vez que se incorpore al sistema el Procesador de Relacional o de Objetos actualmente en desarrollo.

Los aspectos del sistema sobre los que es necesario avanzar, según se ha puesto de manifiesto durante el desarrollo de los experimentos, son fundamentalmente dos. El primero es la falta de adaptabilidad del sistema ante cambios en la formación o contenido de las imágenes. Si bien es cierto que la arquitectura de SVEX permite lograr una cierta robustez en la obtención

de diagnósticos mediante una adecuada selección de procedimientos y definición de los clasificadores, tal y como se ha demostrado en el experimento sobre segmentación de imágenes de exteriores, esta adaptabilidad es de tipo estático lo que con frecuencia no es suficiente. Sería interesante incorporar mecanismos de acomodación que permitieran diseñar procesos de abstracción o de definición de clases menos rígidos y más adaptables. El otro aspecto mejorable es común a todos los sistemas basados en conocimiento y se refiere al coste de la adquisición del conocimiento necesario para programar SVEX. La selección de las características más relevantes, el ajuste de los clasificadores, la definición de prototipos, etc... son tareas cuyo coste se reduciría de forma notable si se dispusiera de un entorno para la adquisición automatizada del conocimiento.

En definitiva, el desarrollo de los casos de estudio presentados a lo largo de este capítulo ha supuesto la aplicación de SVEX en la resolución de un conjunto heterogéneo de problemas en Visión Artificial. La diversidad de contextos en los que se han desarrollado los casos de estudio, la calidad de los resultados obtenidos y la facilidad relativa con la que se han desarrollado demuestran la versatilidad de SVEX y validan, en líneas generales, el modelo sobre el que se ha desarrollado este sistema.

CAPÍTULO



Conclusiones , principales aportaciones y posibles extensiones.

V.1 INTRODUCCIÓN.

Las contribuciones más significativas de este trabajo se sitúan en dos aspectos diferenciados. De una parte, el soporte teórico y metodológico utilizado posibilita la construcción de una estructura computacional para el tratamiento de los problemas de Visión por Computador caracterizada por su flexibilidad. De otra, las herramientas construidas para hacer efectiva esta estructura permiten disponer de elementos de utilidad práctica inmediata. En primer lugar se presentarán las conclusiones y principales aportaciones obtenidas en el desarrollo de este trabajo de investigación. Posteriormente, se propondrán algunas extensiones y futuras mejoras del sistema de segmentación de imágenes (SVEX) que se ha desarrollado.

V.2 CONCLUSIONES Y PRINCIPALES APORTACIONES.

1. Con el objetivo de orientar el desarrollo de este trabajo, previamente se ha realizado una revisión estructurada e integradora del problema de la segmentación de imágenes desde tres perspectivas diferentes. Primeramente, se han revisado las técnicas más empleadas en

segmentación de imágenes en su consideración de "herramientas básicas" o recursos elementales. A continuación, se han analizado diferentes sistemas expertos para proceso de imágenes, los cuales suponen un contribución importante en la línea de diseñar sistemas de segmentación y proceso de imágenes más robustos y autónomos, por cuanto intentan condensar y articular el conocimiento heurístico existente acerca del uso de métodos y técnicas de proceso de imágenes. En la última parte de esta revisión se ha considerado la segmentación en el contexto de Sistemas de Interpretación de Imágenes (IUS). Se ha realizado una descripción de los sistemas de interpretación de imágenes más conocidos, estructurada alrededor de la arquitectura, representación del conocimiento y organización del control empleadas en dichos sistemas. Como aportación de esta revisión, se ha planteado la existencia de una serie de guías o principios de diseño que la experiencia ha evidenciado como útiles en cualquier IUS, y se han resumido los problemas centrales que han de ser resueltos en su diseño.

2. Se ha propuesto una metodología básica para el desarrollo de sistemas percepto-efectores expresada en un conjunto resumido "recetas" o principios de diseño. Esta metodología está basada en la concepción de un sistema en múltiples niveles, en consonancia con el paradigma de una máquina procesadora de datos que evoluciona en un mundo virtual. En ella se propugna un modelo de arquitectura para cada nivel basado en unidades BU-TD, integradas por un procesador Bottom-Up (BU), que actúa como una unidad de diagnóstico o abstracción, y otro procesador Top-Down (TD), concebido como una unidad de control y planificación. La utilización de unidades BU-TD en los diferentes niveles de un sistema posibilita la realización de diferentes modelos de máquinas percepto-efectoras, al tiempo que establece un marco básico para la distribución del cómputo y el control.

3. La existencia del procesador TD permite la articulación de esquemas de control basados, tanto en la evaluación del estado de la descripción generada por el procesador BU, como en conocimiento de más alto nivel activado desde las capas superiores del sistema. En el primer caso, es posible establecer un control de tipo reactivo en forma de "actos reflejos" que permiten al sistema actuar sobre su mundo virtual con independencia de las órdenes recibidas desde niveles superiores. El segundo esquema de control, basado en órdenes o peticiones que se reciben desde

niveles superiores, habilita un mecanismo complementario para utilizar conocimiento de alto nivel en la orientación de tareas en niveles inferiores.

4. La validación de la metodología propuesta se produce a posteriori tras su aplicación al diseño de diferentes sistemas percepto-efectores, esto es, la metodología resultará validada cuando en promedio su utilización haya conducido a la obtención de sistemas más robustos y eficientes o haya facilitado su construcción. En tal sentido, el sistema de visión basado en conocimiento (SVEX) desarrollado en el marco de esta tesis puede considerarse un primer experimento cuyos resultados parecen avalar la utilidad de la metodología empleada.

5. Uno de los logros obtenidos en la aplicación de esta metodología al diseño de SVEX ha sido la de producir una estructura computacional claramente definida, que se ha aplicado tanto en el nivel de los pixels como en el de los segmentos. Dicha estructura establece desde su diseño un mecanismo general de integración de múltiples fuentes de datos (diferentes sensores, características numéricas o simbólicas, ...) y un modelo de control de la incertidumbre en el dominio simbólico. La estructura computacional desarrollada permite el cálculo simbólico en cualquier nivel, incluso en el nivel más bajo. Esto representa una diferencia sustancial frente a concepciones más clásicas donde la consideración "alto nivel vs. bajo nivel" se ha considerado equivalente a la confrontación computación simbólica vs. numérica.

6. Se ha desarrollado un lenguaje de programación declarativo que posibilita la expresión explícita del conocimiento necesario para la actuación de SVEX. Las características del lenguaje permiten la adaptación de SVEX a nuevas tareas o aplicaciones de forma rápida y eficiente. En concreto, se ha constatado como la utilización de este lenguaje reduce notablemente el tiempo de desarrollo de aplicaciones, si se le compara con el coste de desarrollo de la misma aplicación utilizando un lenguaje de propósito general. El lenguaje está basado en un conjunto reducido de tipos de datos y operadores, e integra procesos numéricos de bajo nivel con constructores de alto nivel como son reglas y jerarquías de clases, además de permitir la gestión de la incertidumbre.

7. La definición de los niveles que integran SVEX está basada en entidades próximas al observador (pixels, segmentos, objetos, ...) lo que establece una semántica clara y facilita la definición del conjunto operadores en cada nivel. Es necesario resaltar que ha sido un principio similar de clarificación de las entidades a procesar en cada nivel lo que ha definido el desarrollo de arquitecturas específicas como la IUA [Weem-89] [Weem-92]. La organización de SVEX resultante de esta concepción es modular y escalable. Así, es posible resolver tareas de visión que requieran diagnósticos a nivel de pixels aplicando sólo el Procesador de Pixels; o los procesadores de Pixels y de Segmentos cuando sea necesaria la localización de formas simples, etc.

8. Por su diseño, los dos niveles de SVEX presentes en la implementación descrita en este trabajo están orientados a la atención de peticiones de diagnósticos procedentes del nivel superior o de un programa de aplicación. En este sentido, está contemplada en SVEX la posibilidad de enlazar bajo petición a un Procesador de Segmentos con varios Procesadores de Pixels, cada uno computando diagnósticos independientes sobre una misma imagen. Esto permite la distribución efectiva del cómputo en los diferentes niveles del sistema y facilita la integración de SVEX en otros sistemas más generales o en aplicaciones específicas.

9. Se ha identificado la transformación de la representación, obligada por un cambio de nivel, como un problema fundamental a resolver en el diseño de sistemas multinivel. En SVEX, éste es el cometido del módulo de presegmentación del Procesador de Segmentos, encargado de generar una primera partición de la imagen en términos de segmentos a partir de los mapas de diagnóstico elaborados por el Procesador de Pixels. Como quiera que no se conoce una técnica de segmentación de validez universal, es decir, aplicable con superioridad en cualquier contexto, el módulo de presegmentación se ha concebido como una parte intercambiable del Procesador de Segmentos, flexibilizándose así la utilización de diferentes módulos de presegmentación en función de la tipología de la imagen y el objetivo de la segmentación.

10. La transformada Watershed ha servido para inspirar un módulo de presegmentación ("Flood") versátil y flexible, que ha demostrado ser válido para producir la segmentación de una imagen a partir de múltiples mapas de diagnósticos. En tal sentido, el módulo de presegmentación en

particular, y la técnica sobre la que se basa en general, constituyen una herramienta de segmentación potente y de amplia aplicación.

11. Se ha producido un algoritmo de análisis de componentes conectadas (AACC) rápido y adaptable. Para cada componente conectada presente en un mapa multicolor, el algoritmo propuesto extrae el borde externo y los posibles bordes internos, la lista ordenada de las componentes vecinas según se encuentran al recorrer cada borde y el rectángulo mínimo que la circunscribe. Esta información puede ser complementada con la lista de los pixels que conforman cada componente. Además, el algoritmo permite la selección de 8 o 4-conectividad de forma independiente tanto para el análisis de conectividades como para el análisis de vecindades.

12. Este algoritmo se ha comparado con el algoritmo de etiquetado de componentes conectadas (AECC) propuesto en [Yang-92], el cual puede ser considerado como uno de los más eficientes a partir de los resultados publicados en la literatura [Dani-82][Lumi-83a][Caps-84][Same-86] para algoritmos no paralelos. Del estudio realizado se ha concluido que el AACC propuesto es más rápido que el AECC hasta un número razonablemente alto de esquinas (alrededor de 10000) y componentes (aproximadamente 2000) en imágenes de tamaño 512x512 pixels. Debe tenerse en cuenta que el AECC es aplicable sólo a imágenes binarias y genera, como resultado, una mapa de etiquetas conteniendo componentes 4-conectadas. Este resultado no es comparable con el que se obtiene de la aplicación del AACC propuesto, pues no se extrae del mapa de etiquetas generado información explícita alguna: bordes de las componentes, información sobre vecindades, etc. En este sentido, el análisis de la comparación debe considerarse como una estimación pesimista del rendimiento del algoritmo propuesto. Por todo lo anterior, consideramos el AACC propuesto en este trabajo como una contribución importante con un amplio rango de aplicación.

13. Se ha diseñado e implementado un conjunto inicial de herramientas de apoyo a la programación que configuran el entorno de SVEX como sistema de desarrollo para aplicaciones en visión. Los elementos más importantes de este conjunto son:

- Una herramienta (Sampler) que permite "recolectar" muestras de manera cualificada, tanto a nivel de pixels como de segmentos, y que constituye el primer elemento de un sistema de aprendizaje.
- Un editor icónico que permite la escritura de programas de manera asistida, rápida y libre de errores léxicos o semánticos.
- Una herramienta de visualización y análisis de mapas de características y de diagnósticos a nivel del Procesador de Pixels (ViewMap), y otra herramienta de características similares en el nivel de segmentos que, integrada en el Procesador de Segmentos, permite realizar la traza de un programa en este nivel (Xdebug).

14. El objetivo fundamental de los experimentos ha sido mostrar, mediante una variedad de problemas relativamente sencillos, un enfoque no tradicional de resolución de problemas de segmentación de imágenes. La utilización del esquema de cómputo impuesto por la propia concepción de los procesadores de pixels y de segmentos puede producir en ocasiones una cierta falta de libertad frente a la forma tradicional de programación de sistemas de visión utilizando un lenguaje de propósito general. Sin embargo, esta falta de libertad se ve compensada por la definición de una nueva forma más estructurada de abordar la resolución de problemas de Visión por Computador, que pone el énfasis en la organización del conocimiento útil al problema y convierte en absolutamente secundario el resto.

15. Sobre el desarrollo de los experimentos cabe establecer fundamentalmente tres conclusiones. En primer lugar, se ha establecido la capacidad del modelo propuesto para resolver una variedad de problemas reales de segmentación mediante mecanismos de utilidad general relativamente sencillos. Así, se han resuelto problemas de segmentación desde un conjunto heterogéneo de imágenes (imágenes en niveles de gris, en color o secuencias) en diferentes contextos (control de calidad, aplicaciones industriales, imágenes de exterior, etc...). Algunos de los casos estudiados han sugerido el interés de incorporar en SVEX diferentes mecanismos de realimentación entre el nivel de pixels y de segmentos. En segundo lugar, cabe considerar a SVEX y el conjunto de herramientas anexas como un entorno para el desarrollo de aplicaciones en Visión por Computador con un coste de desarrollo inferior al obtenido en un entorno de programación

general. Por último, se ha constatado la existencia de un cuello de botella importante en lo referente a la adquisición del conocimiento, por lo que resultaría del mayor interés disponer de una herramienta de aprendizaje que automatizara, entre otras tareas, la selección de las características (Features) más relevantes y la definición simbólica de las clases en términos de estas características.

16. Los experimentos han servido también para mostrar algunas facetas de SVEX que, aunque no han sido desarrolladas plenamente en esta implementación, son potencialmente muy importantes. Una de ellas es la posibilidad de definir un conjunto de conceptos primitivos o elementales, tanto a nivel de pixels (ser "rojo", ser "cielo", ser "follaje", ...) como a nivel de segmentos (ser "cuadrado", "recto", ...). Estos conceptos elementales constituyen la primera capa de la base de modelos de cada nivel y pueden combinarse en la definición de nuevos conceptos o clases más complejas en diferentes contextos. Se ha mostrado también como es posible organizar la interpretación en SVEX mediante un esquema donde se progresa de lo más evidente a lo más incierto o de lo general al detalle, lanzando determinados diagnósticos sobre zonas concretas de la imagen, una vez que se han verificado ciertas hipótesis. Esto permite la descomposición de análisis complejos en etapas sucesivas y la utilización del conocimiento para guiar y localizar dicho análisis. Con todo, potenciar de estas capacidades requiere avances en la implementación en aspectos relacionados con el control del sistema y con la integración de los diferentes niveles. La implementación del Procesador de Objetos es un elemento esencial en la consecución de este objetivo, pues sólo desde este nivel es posible tener una información contextual amplia que permita guiar desde el conocimiento la interpretación/segmentación de la imagen.

V.3 POSIBLES EXTENSIONES Y LÍNEAS DE DESARROLLO.

Las principales extensiones de SVEX han sido ya sugeridas en las conclusiones sobre el prototipo desarrollado en este trabajo y, sobre todo, en el análisis de los experimentos.

E1. *Dotar a SVEX de una mayor adaptabilidad, fundamentalmente en el nivel de los pixels.* En la línea de conseguir un sistema de interpretación más robusto y adaptable, es necesario dotar a SVEX de mecanismos tales como acomodación, selección de parámetros para procedimientos, selección de procedimientos, etc. Estos mecanismos de adaptación deben resultar efectivos en cualquier nivel, pero es en el de los pixels donde son más necesarios para adecuar el cálculo de diagnósticos a las características de la señal. La adaptabilidad requiere de la evaluación, tanto de los resultados obtenidos de la aplicación de un cierto procedimiento, como de la segmentación obtenida. Es este un problema sobre el que se han ensayado diferentes esquemas [Nazi-84][Mats-90][Clem-93][Lied-91][Drap-89], pero que continúa absolutamente abierto.

E2. *Incorporar en SVEX el control de los parámetros de cámara.* Se ha previsto la incorporación en SVEX de actuadores para el control de parámetros de cámara como el enfoque, la apertura de diafragma, el control del zoom y la orientación de la cámara. Esta posibilidad redundaría en una mayor adaptabilidad del sistema, permitiría articular esquemas de control basados en el paradigma de visión activa y, en último término, ampliaría el campo de aplicación de SVEX a problemas de visión no considerados hasta ahora.

E3. *Modificar la definición de las funciones de decisión asociadas a los clasificadores.* La implementación actual sólo permite la definición de funciones de decisión entre los valores extremos de evidencia (máxima confirmación o desconfirmación). La relajación de esta circunstancia permitiría definir funciones de decisión (clasificadores) que produjeran confirmación o disconformidad de manera exclusiva y/o en grados intermedios.

E4. *Investigar otras reglas de combinación de evidencias.* Los dos esquemas de combinación incremental de evidencias producidas por clasificadores y reglas, utilizados en los dos niveles de SVEX en el proceso de obtención de diagnósticos, son las reglas de combinación más comunes en sistemas expertos [Gord-84]. Existen circunstancias que demuestran que ninguna de estas reglas de combinación es óptima. Se requiere, por tanto, del estudio y análisis de otras reglas de combinación. En tal sentido, una posibilidad interesante es la de abandonar la definición funcional

de reglas de combinación y utilizar, en cambio, tablas para la combinación incremental de evidencias.

E5. Incorporación de nuevos módulos de presegmentación. El módulo de presegmentación basado en la transformada Watershed ha demostrado su versatilidad en los experimentos desarrollados hasta ahora. Sin embargo, esto no excluye la incorporación de nuevos presegmentadores basados en otras técnicas de segmentación capaces de manejar múltiples mapas de diagnósticos [Ross-89][Lim-90][Moli-92]. Un aspecto del presegmentador actual susceptible de mejora es la necesidad que se tiene de definir exhaustivamente las diferentes zonas "prototipo" que aparecen en la imagen a segmentar. En entornos donde el contenido de la imagen no esté completamente definido a priori, esto puede suponer un problema importante al invadir anormalmente las zonas de clara definición las zonas correspondientes a prototipos no definidos. Una posible solución a este problema pasaría por incorporar la capacidad de descubrir automáticamente agrupaciones significativas de puntos conexos en el espacio de representación definido por los mapas de diagnóstico. Una cuestión muy ligada a la anterior, y que ha sido sugerida también por otros autores[Fuji-90], es si es necesario que la partición generada en el presegmentador sea exhaustiva, es decir, que el conjunto de segmentos definidos por la partición inicial forme una teselación completa de la imagen. Esta posibilidad, que resultaría interesante sobre todo en las primeras fases de la interpretación donde la segmentación intentaría proporcionar hipótesis iniciales lo más robustas posibles, obligaría a reformular la definición de las relaciones espaciales entre segmentos y aporta otra dimensión al problema.

E6. Potenciar la capacidad de procesamiento relacional del Procesador de Segmentos. En la implementación actual del Procesador de Segmentos se ha explorado la utilización de relaciones espaciales entre segmentos sólo de forma parcial. De las posibles extensiones, se destacan las que parecen más interesantes y prometedoras. Una de ellas es la extensión de las características (Features) asociadas a los segmentos, que en la actualidad cualifican numéricamente al segmento de forma íntegra, para permitir incluir características relativas a vecindades. En la implementación actual, no es posible cualificar numéricamente atributos de naturaleza relacional o binaria que se asocian con las relaciones de vecindad del segmento, como, por ejemplo, el contraste medio a lo

largo de la frontera común entre un segmento y cada uno de sus vecinos. Características similares han sido utilizadas en numerosos sistemas de segmentación [Nazi-84][Ross-89]. La consideración de este tipo de características binarias se prolonga de forma natural al dominio simbólico. Otra extensión interesante consistiría en posibilitar que determinadas características que dependieran bien de la forma, bien de características o diagnósticos de los segmentos vecinos pudieran ser calculadas de forma selectiva. De esta forma sería posible seleccionar los vecinos a intervenir en el cálculo, los cuales deberían satisfacer una determinada premisa.

E7. Definición de localizadores basados en atributos. Los localizadores empleados en la implementación actual del Procesador de Segmentos son meramente posicionales (Encima, Debajo, etc...). Una extensión interesante es la inclusión de localizadores que pudieran tomar como argumento un atributo, numérico o simbólico, asociado al segmento. Mediante este tipo de localizadores sería posible establecer condiciones, por ejemplo, sobre el vecino "más cuadrado" o "más similar en tamaño".

E8. Definición de nuevas acciones de control en el Procesador de Segmentos. Pudiera resultar de interés el establecimiento de nuevas acciones de control orientadas a tratar la aparición de inconsistencias en la interpretación. En la implementación actual es posible articular reglas de control que evidencien la aparición de inconsistencias, como que un segmento etiquetado como árbol está incluido dentro de otro que está etiquetado como carretera; pero no es posible asociarles ninguna acción de control que resolviera estos conflictos de interpretación [Fuji-90][McKe-89][Drap-89].

E9. Implementación del Procesador de Objetos. Como ya se ha manifestado en diferentes ocasiones a lo largo de la exposición de estas conclusiones, el desarrollo e implementación del Procesador de Objetos es un objetivo fundamental en la extensión de SVEX. No sólo porque significaría completar el modelo propuesto para un sistema de interpretación de imágenes lo que permitiría la aplicación de SVEX en tareas más complejas, sino también porque su desarrollo posibilitaría la incorporación de esquemas de control más versátiles y potentes. En este sentido, opciones surgidas del análisis de los experimentos como la búsqueda selectiva de diagnósticos

sobre partes de la imagen en base a hipótesis previas, la aplicación de reglas de control específicas sobre determinados conjuntos de segmentos, etc..., y ,en definitiva, la orientación de la interpretación desde el conocimiento de objetos (control Top-Down), sólo son realizables desde la existencia de este procesador.

E10. *Automatización del aprendizaje.* Numerosos autores [Niem-90b][Conn-87][Fich-92][Pell-94][McKe-89] han señalado el proceso de aprendizaje y adquisición del conocimiento sobre los modelos como un factor que constriñe en gran medida la realización práctica de sistemas de visión basados en conocimiento. El problema del aprendizaje tiene numerosas facetas que van desde la elaboración de herramientas para la adquisición automatizada de muestras cualificadas (Sampler), y la selección de los procedimientos de caracterización numérica idóneos, a la generación automática de procesos abstractores que permitan la transformación de datos en categorías simbólicas y la obtención de reglas simbólicas.

REFERENCIAS

- [Abid-92] Abidi M.A., González R.C. (Eds.), *"Data Fusion in Robotics and Machine Intelligence"*, Academic Press, 1992.
- [Agga-93] Aggarwal J.K. (Ed.), *"Multisensor Fusion for Computer Vision"*, NATO ASI Series, Series F: Computer and Systems Sciences, vol. 99, Springer-Verlag, 1993.
- [Ahu-78] Ahuja N., Rosenfeld A., *"A note on the use of second-order statistic for threshold selection"*, IEEE Trans. Systems Man Cybernetic, 8, pp. 895-899, 1978.
- [Aloi-88] Aloimonos J., Weis I., Bandyopadhyay A., *"Active Vision"*, Int. Journal of Computer Vision, Vol. 1, n° 4, pp. -, 1988.
- [Ball-82] Ballard D., Brown C.M., *"Computer Vision"*, Prentice Hall, 1982.
- [Bail-88] Bailey D.G., *"Research on computer-assisted generation of image processing algorithms"*, in Proceedings of IAPR Workshop on Computer Vision-Special Hardware and Industrial Applications, Tokio, pp. 294-297, 1988.
- [Bart-89] Bartneck N., *"A General Data Structure for Image Analysis based on a Description of Connected Components"*, Computing Vol. 42, pp. 17-34, 1989.
- [Beuc-79] Beucher S., Lantuéjoul C., *"Use of Watersheds in contour detection"*. Int. Workshop on Image Processing, Real-Time edge and motion Detection/Estimation. CCETT/INSA. IRISA Report n°132, p. 2.1-2.12. Rennes, France, Sept 17-21, 1979.
- [Beuc-90] Beucher S., *"Segmentation Tools in Mathematical Morphology"*. SPIE vol. 1350, Image Algebra and Morphological Image Processing. pp. 70-84, 1990.
- [Bez-81] Bezdek J.C., *"Pattern Recognition with Fuzzy Objective Function Algorithms"*, Plenum Press, Nueva York, 1981.
- [Bin-71] Binford T.O., *"Visual Perception by Computer"*, Invited paper at IEEE Systems Science and Cybernetics Conference, Miami, Diciembre 1971.
- [Blan-89] Blanz W.E., Sanz J.L.C., Petkovic D., *"Control-Free Low-Level Image Segmentation: Theory, Architecture and Experimentation"*, en Advances in Machine Vision, Sanz J.L.C. Ed., Springer-Verlag, pp.254-282, 1989.
- [Blöm-89] Blömer A., Liedtke C.E., *"Use of Natural Spoken Terms for the Evaluation of Knowledge-Based Image Interpretation System"*, Third Int. Conference on Image Processing and its Applications, Warwick, 1989.
- [Bric-70] Brice C.R., Fenema C.L., *"Scene Analysis using Regions"*, Artificial Intelligence, 1, pp. 205-226, 1970.
- [Brol-89] Brolio J., Draper B.A., Beveridge J.R., Hanson A.R., *"ISR: A Database for Symbolic Processing in Computer Vision"*. Computer vol. 22, n° 12, pp. 22-30, Diciembre 1989.
- [Broo-79a] Brooks R.A., *"Goal-Directed Edge Linking and Ribbon Finding"*, Proceedings ARPA Image Understanding Workshop, Menlo Park, pp. 72-76, Abril 1979.

- [Broo-79b] Brooks R.A., Greiner R., Binford T., *"The ACRONYM Model-Based Vision System"*. Proceedings of the Sixth IJCAI, Tokio, Japon, pp. 105-113, 1979.
- [Broo-81] Brooks R.A., *"Symbolic Reasoning Among Three-Dimensional Interpretations of Two-Dimensional Images"*. Artif. Intell. 17, pp. 285-348, 1981.
- [Broo-83] Brooks R.A., *"Model-based Three-Dimensional Interpretations of Two-Dimensional Images"*. IEEE Trans. on Pattern Anal. and Machine Intell., Vol 5, n° 2, pp. 140-150, 1983.
- [Broo-84] Brooks R.A., *"Model-Based Computer Vision"*. Computer Science: Artificial Intelligence N°14, H.S. Stone Ed., UMI Research Press, Ann Arbor (Michigan), 1984.
- [Broo-86] Brooks R.A., *"A Robust layered Control System For A Mobile Robot"*, IEEE Journal of Robotic and Automation, Vol. 2, n° 1, pp. 14-23, 1986.
- [Broo-91a] Brooks R.A., *"Elephants Don't Play Chess"*, en Designing Autonomous Agents, Ed. P. Maes, MIT Press, pp. 3-15, 1991.
- [Broo-91b] Brooks R.A., *"Challenges for Complete Creature Architectures"*, en Proc. First Int. Conference on Simulation of Adaptive Behavior, Ed. Meyer y Wilson, MIT Press, pp. 434-443, 1991.
- [Cann-86] Canny J., *"A Computational Approach to Edge Detection"*, IEEE Trans. on Pattern Anal. and Machine Intell., Vol 8, n° 6, pp. 679-698, 1986.
- [Caps-84] Capson D.W., *"An improved algorithm for the sequential extraction of boundaries from a raster scan"*, Comp. Vision, Graph. and Image Proc., 28, pp. 109-125, 1984.
- [Clém-93] Clément V., Thonnat M., *"A Knowledge-Based Approach to Integration of Image Processing Procedures"*, CVGIP: Image Understanding, Vol. 57, n° 2, pp. 166-184, 1993.
- [Conn-87] Connell J.H., Brady M., *"Generating and Generalizing Models of Visual Objects"*, Artificial Intell. 31, pp. 159-183, 1987.
- [Chen-87] Chen J.S., Huertas A., Medioni G., *"Fast Convolution with Laplacian of Gaussian Masks"*, IEEE Trans. on Pattern Anal. and Machine Intell., Vol 9, n° 7, pp 584-590, 1987.
- [Chen-92] Chen C., Mulgaonkar P.G., *"Automatic Vision Programming"*, CVGIP: Image Understanding, Vol. 55, n° 2, pp. 170-183, 1992.
- [Cheng-92] Cheng Y., Kashyap R.L., *"Recursive Fusion Operators: Desirable Properties and Illustrations"*, en "Data Fusion in Robotics and Machine Intelligence", Abidi M.A. and González R.C. (Eds), Academic Press, pp. 245-265, 1992.
- [Dani-82] Danielson, P.E., *"An improved segmentation and coding algorithm for binary and nonbinary images"*, IBM J. Res. Develop. Vol. 26, n° 6, pp. 698-707, Noviembre 1982.
- [Davi-75] Davis L.S., *"A Survey on Edge Detection Techniques"*, Comp. Graph. and Image Proc., 4, pp 248-270, 1975.
- [Dera-83] Deravi F., Pal S.K., *"Gray level thresholding usind second order statistics"*, Pattern Recong. Letters, 1, pp. 417-422, 1983.
- [Deri-87] Deriche R., *"Using Canny's Criteria to Derive a Recursively Implemented Optimal Edge Detector"*, Int. Journal of Computer Vision, Vol 1, pp. 167-187, 1987.

Referencias

- [Diga-78] Digabel H., Lantuéjoul C., "*Iterative Algorithms*". Proc. 2nd European Symp. Quantitative Analysis of Microstructures in Material Sciences, Biology and Medicine. Caen, France, Oct.1977, J.L. Chermant, Ed. Stugart, Germany,: Riederer Verlag, pp. 85-99, 1978.
- [Dins-85] Dinstein I., Yen D., Flickner M., "*Handling memory overflow in connected component labeling applications*", IEEE Trans. on Pattern Anal. and Machine Intell., Vol 7, n° 1, 1.985.
- [Drap-89] Draper B.A., Collins R.T., Brolio J., Hanson A.R., Riseman E.M., "*The Schema System*", Int. Journal of Computer Vision, 2, pp. 209-250, 1989.
- [Dube-90] Dubes R.C., Jain A.K., Nadabar S.G., Chen C.C., "*MRF Model-Based Algorithms For Image Segmentation*", Proc. 10th Int. Conf. on Pattern Recognition, IEEE Computer Soc. Press, pp. 808-814, Junio 1990.
- [Elfe-91] Elfes A., "*Dynamic Control of Robot Perception using Stochastic Spatial Models*", Int. Workshop on Information Processing in Mobile Robots, Springer-Verlag, New York, 1991.
- [Falc-86] Falcón A., Méndez J., Hernández F.M., "*Segmentación*", en Visión Artificial, editado por Asociación Española de Informática y Automática (AEIA), pp. 4.1-4.37, 1986.
- [Fich-92] Fichera O., Pellegretti P., Roli F., Serpico S.B., "*Automatic Acquisition of Visual Models for Image Recognition*", in Proc. 11th Int. Conf. on Pattern Recognition, The Hague, The Netherlands, IEEE Computer Soc. Press, pp. 95-98, 1992.
- [Flec-92] Fleck M., "*Some Defects in Finite-Difference Edge Finders*", IEEE Trans. on Pattern Anal. and Mach. Intell., Vol 14, n° 3, pp. 337-345, 1992.
- [Fox-90] Fox M.S., "*AI and Expert System Myths, Legends and Facts*", IEEE Expert, Vol. 5, n°. 1, pp. 8-20, 1990.
- [Fuji-90] Fujimori T., Kanade T., "*An Approach to Knowledge-Based Interpretation of Outdoor Natural Color Road Scenes*", en "Vision and Navigation: The Carnegie Mellon Navlab", Thorpe C.E. (Ed.), Kluwer Academic Publishers, pp.39-81, 1990.
- [Gall-88] Gallant S.I., "*Connectionist Expert Systems*", Com. ACM, Vol. 31, n°. 2, pp. 152-169, 1988.
- [Geig-91] Geiger D., Yuille A., "*A Common Framework for Image Segmentation*", Int. Journal of Computer Vision, vol. 6, n° 3, pp. 227-243, 1991.
- [Gema-84] Geman S., Geman D., "*Stochastic Relaxation: Gibbs Distributions and the Bayesian Restoration of Images*", IEEE Trans. on Pattern Anal. and Mach. Intell. Vol. 6, pp. 721-741, 1984.
- [Goad-83] Goad C., "*Special purpose automatic programming for 3D model-based vision*", en Proceedings DARPA Image Understanding Workshop, pp. 94-104, Junio 1983.
- [Gonz-77] Gonzalez R., Wintz P., "*Digital Image Processing*", Addison Wesley, 1977.
- [Gord-84] Gordon J., Shortliffe E.H., "*The Dempster-Shafer Theory of Evidence*", en "Rule-Based Expert Systems", Ed. Buchanan y Shortliffe, Addison-Wesley, Reading, Mass., pp. 272-292, 1984.
- [Grim-90] Grimson W.E.L., "*Object Recognition by Computer: The Role Of Geometric Constraints*", MIT Press, Massachusetts, 1990.
- [Hall-79] Hall E.L., "*Computer Image Processing and Recognition*", Academic Press, 1979.

- [Hall-92] Hall L.O., Bensaid A.M., Clarke L.P., Velthuizen R.P., Silbiger M., Bezdek J.C., "A comparison of neural network and fuzzy clustering techniques in segmenting magnetic resonance images of the brain", IEEE Trans. Neural Networks, Vol. 3, n° 5, pp. 672-681, 1992
- [Hans-82] Hansen F.R., Elliot H., "Image Segmentation Using Simple Markov Field Models" Comp. Vision, Graph., and Image Proc., 20, pp. 101-132, 1982.
- [Hans-88] Hanson A.R., Riseman E.M., "The VISIONS Image-Understanding System". In Advances in Computer Vision, Vol I, pp. 1-114, Ed. C. Brown, Lawrence Erlbaum Associates Publishers, New Jersey, 1988.
- [Hans-89] Hansen C., Henderson T.C., "CAGD-based computer vision", IEEE Trans. on Pattern Anal. and Mach. Intell., Vol 11, n° 11, pp. 1181-1194, 1989.
- [Hara-85] Haralick R., Shapiro L.G., "Survey: Image Segmentation Techniques", Comp. Vision, Graph. and Image Proc., 29, pp 100-132, 1985.
- [Hara-91] Haralick R.M. Shapiro L.G., "Glossary of Computer Vision Terms", Pattern Recognition, Vol. 24, n° 1, pp. 69-93, 1991.
- [Hern-89] Hernández F.M., Mendez J., Falcón A., "SVEX: Un Modelo de Sistema de Visión Estructural", Report Dep. Informática y Sistemas, Univ. Las Palmas de Gran Canarias, Febrero 1989.
- [Hern-90] Hernández F.M., Mendez J., Falcón A., "A Model for Structural Vision System", en Proc. of International Workshop on Computer Aided System Theory (EUROCAST'89), F. Pichler, R. Moreno-Díaz (Eds), pp 310-321, Springer-Verlag, Berlin, 1990.
- [Hild-92] Hild M., Shirai Y., Asada M., "Initial Segmentation for Knowledge Indexing", In Proc. of Inter. Conf. on Pattern Recognition, IEEE Computer Soc. Press, pp.587-590, 1992.
- [Hofs-89] Hofstadter D.R., "Gödel, Escher, Bach: Un Eterno y Grácil Bucle", Ed. Tusquets, Barcelona, 1989.
- [Horn-89] Horn B.P., Brooks M.J., (Edts.), *Shape from Shading*, MIT Press, 1989
- [Horo-76] Horowitz S. L. and Pavlidis T., "Picture Segmentation by a Tree Traversal Algorithm", Journal of the ACM, 23, pp. 368-388, 1976.
- [Huan-88] Huang J.S., Tseng D.H., "Statistical Theory of Edge Detection", Comp. Vision, Graph., and Image Proc., 43, pp 337-346, 1988.
- [Huec-71] Hueckel M.H., "An Operator Which Locates Edges in Digitized Pictures", Journal of ACM, 18, pp 113-125, 1971
- [Ikeu-87] Ikeuchi K., "Generating an interpretation tree from a CAD model for 3-D object recognition in bin-picking tasks", Int. Journal of Computer Vision, Vol. 1, n° 2, pp. 145-165, 1987.
- [Ikeu-88] Ikeuchi K., Kanade T., "Towards automatic generation of object recognition program", Proc. IEEE, Vol. 76, n° 8, pp. 1016-1035, 1988.
- [Ikeu-91] Ikeuchi K., Hong K.S., "Determining Linear Shape Change: Toward Automatic Generation of Object Recognition Programs", CVGIP: Image Understanding Vol. 53, n° 2, pp. 154-170, 1991.
- [Jain-89] Jain A.K., "Fundamentals of Digital Image Processing", Prentice-Hall Inter., 1989.

Referencias

- [Jain-91] Jain R.C., Binford T.O., "*Ignorance, Myopia and Naiveté in Computer Vision Systems*", CVGIP: Image Understanding, 53, 112-117, 1991.
- [John-87] Johnston M.D., "*An expert system approach to astronomical data analysis*", en Proceedings of Goddard Conf. on Space Applications of Artificial Intelligence and Robotics, pp. 1-17, 1987.
- [Kapu-85] Kapur J.N., Shao P.K., Wong A.K.C., "*A New method for gray-level picture thresholding using the entropy of histogram*", Comp. Vision, Graph. and Image Proc., 29, pp. 273-285, 1.985.
- [Kitt-86] Kittler J., Illingworth J., "*Minimum Error Thresholding*", Pattern Recog., 19, pp. 41-47, 1.986.
- [Kohl-87] Kohl C.A., Hanson A., Riseman E., "*A goal-directed low level executive for image interpretation*", in Proceedings 10th Int. Joint Conf. Artificial Intelligence, Milán, pp. 811-814, Agosto 1987.
- [Kohl-88] Kohl C.A., "*GOLDIE: A goal-directed low level executive for image interpretation*", COINS Tech. Report 88-22 (Ph. D. thesis) Dept. of Computer and Information Science, Univ. of Massachusetts at Amherst, 1988.
- [Kowa-79] Kowalski R., "*Logic for Problem Solving*", Elsevier Pub. Co., New York, 1979.
- [Lang-92] Langton C.G., "*Artificial Life II*", Addison-Wesley Pub., 1992.
- [Levi-85a] Levine M., Nazif A., "*Rule-Based Image Segmentation: A Dynamic Control Strategy Approach*", Comp. Vision, Graph., and Image Proc., 32, pp 104-126, 1.985.
- [Levi-85b] Levine M.D., Nazif A.M., "*Dynamic Measurement of Computer Generated Image Segmentations*", IEEE Trans. on Pattern Anal. and Machine Intell., Vol 7, n° 2, pp. 155-164, 1985.
- [Lied-91] Liedtke C.E., Blömer A., Gahm Th., "*Knowledge Based Configuration of Image Segmentation Processes*", Int. Journal of Imaging Systems and Technology, 1991.
- [Lied-92] Liedtke C.E., Blömer A., "*Architecture of the Knowledge-Based configuration System for Image Analysis CONNY*", in Proceedings of IEEE 11th Int. Conf. on Pattern Recognition, The Hague, The Netherlands, IEEE Computer Soc. Press, pp. 375-378, 1992.
- [Lim-90] Lim Y.W., Lee S.U., "*On the color image segmentation algorithm based on the thresholding and the fuzzy C-Means techniques*", Pattern Recognition, Vol. 23, n° 9, pp. 935-952, 1990.
- [Lipp-87] Lippmann R.P., "*An Introduction to Computing with Neural Nets*", IEEE Acoustic Signal and Speech Processing Magazine, Vol. 4, n° 2, pp. 4-22, 1987.
- [Lore-94] Lorenzo-Navarro J., Hernández-Tejera M., "*Image Segmentation using a modified Split&Merge Technique*", Cybernetics and Systems: An International Journal, Vol. 25, pp.137-162, 1994.
- [Lu-89] Lu Y., Jain R.C., "*Behavior of edges in scale space*", IEEE Trans. on Pattern Anal. and Mach. Intell., Vol 11, n° 4, pp. 337-356, 1989.
- [Lu-92] Lu Y., Jain R.C., "*Reasoning about edges in scale space*", IEEE Trans. on Pattern Anal. and Mach. Intell., Vol 14, n° 4, pp. 450-468, 1992.
- [Lumi-83a] Lumia R., Shapiro L., Zuñiga O., "*A new connected components algorithm for virtual memory computers*", Comp. Vision, Graph. and Image Proc., 22, pp. 287-300, 1.983.

- [Lumi-83b] Lumia R, "A new three-dimensional connected components algorithms", *Comp. Vision, Graph. and Image Proc.*, 23, pp. 207-217, 1.983.
- [Maes-90] P. Maes, "Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back", MIT Press, 1990.
- [Mand-90] Mandler E., Oberländer M.F., "One-Pass encoding of connected components in multivalued images", *Proc. of 10th Int. Conf. on Pattern Recognition*, Vol. II, pp. 64-69, IEEE Computer Society Press, 1990.
- [Mano-89] Manohar M., Ramapryian H.K., "Connected component labeling of binary images on a mesh connected massively parallel processor", *Comp. Vision, Graph. and Image Proc.*, 45, pp 133-149, 1.989.
- [Marr-80] Marr D., Hildreth E. "Theory of Edge Detection", *Proc. Royal Soc., Londres*, vol B207, pp. 187-217, 1980.
- [Marr-82] Marr D., "Vision", W.H. Freeman, Nueva York, 1.982.
Marr D., "La Visión", Alianza Editorial, 1985.
- [Marroq-87] Marroquín J.L., "Deterministic Bayesian estimation of Markovian random fields with applications to computational vision", *Proc. First Int. Conference on Computer Vision*, Londres, Junio 1987,
- [Mats-84] Matsuyama T., "Knowledge Organization and Control Structure in Image Understanding", en *Proc. VIIth Int. Conf. on Pattern Recognition*, Montreal, 1118-1127, 1984.
- [Mats-90] Matsuyama T., Hwang V.S., "SIGMA: A Knowledge-Based Aerial Image Understanding System". Plenum Publishing Corporation. New York, 1990.
- [McKe-84] McKeown D.M. Jr., Denlinger J.L., "Map guided feature extraction from aerial imagery". *Proceedings of the Second IEEE Computer Society Workshop on Computer Vision: Representation and Control*. 1984
- [McKe-85] McKeown D.M. Jr., Harvey W.A. Jr., McDermott J. "Rule-Based Interpretation of Aerial Imagery". *IEEE Trans. on Pattern Anal. and Mach. Intell.*, Vol 7, n° 5, pp. 570-585, 1985.
- [McKe-89] McKeown D.M. Jr., Harvey W.A. Jr., Wixson L.E.. "Automatic Knowledge Acquisition for Aerial Image Interpretation". *Computer Vision, Graphics and Image Processing*, 46, pp. 37-81, 1989.
- [Mend-92] Méndez J., Falcón A., Hernández F.M., "Procesador de Pixel Basado en Ingeniería del Conocimiento", Report Dept. Informática y Sistemas, Univ. Las Palmas de Gran Canaria, 1992.
- [Mend-94] Mendez J, Falcón A., Hernández F., Cabrera J., "A Development Tool for Computer Vision Systems at Pixel Level", *Cybernetic & Systems* (Pend. Publicación), 1.994.
- [Meye-91] Meyer J.A., Wilson S.W., "From animals to animats", *Proc. of the First Int. Conf. on Simulation of Adaptive Behavior*; MIT Press, 1991.
- [Mins-75] Minsky M., "A Framework for Representing Knowledge", en "The Psicocology of Computer Vision", Wiston et al. (eds), 1.975.
- [Mira-84] Mira-Mira J., Moreno-Díaz R., "Un Marco Teórico para Interpretar la Función Neuronal a Altos Niveles", en "Biocibernética: Implicaciones en biología, medicina y tecnología", R. Moreno-Díaz y J. Mira (Eds), Ed. Siglo Ventiuno S.A., Madrid, 1984.

Referencias

- [Miro-92] Miró J., Mira J., Moreno-Díaz R., "Teoría del Conocimiento", Mesa Redonda, Departamento de Informática y Sistemas, Univ. de Las Palmas de Gran Canaria, 15 de Mayo de 1.992.
- [Moli-92] Molion J. M. and Montanvert A., "The Adaptive Pyramid: A Framework for 2D Image Analysis", CVGIP: Image Understanding, Vol. 55, pp. 339-348, 1992.
- [Mont-91] Montanvert A., Meer P., Rosenfeld A., "Hierarchical Image Analysis Using Irregular Tessellations", IEEE Trans. on Pattern Anal. and Mach. Intell., Vol 13, nº 4, pp. 307-316, 1991.
- [More-80a] Moreno-Díaz R., Rubio E., "A Model for Nonlinear Processing in Cat's Retina", Biol. Cybernetics, nº 37, pp. 25-31, 1.980.
- [More-80b] Moreno-Díaz R., Rubio Royo F., "A Theoretical Proposal to Account for Visual Computation in a Frog's Retina", Int. J. Bio-Med. Comp., nº 11, pp. 415-426, 1.980.
- [More-84] Moreno-Díaz R., Mira J., "La Teoría de Automatas y la Teoría de Sistemas en Biocibernética", en "Biocibernética: Implicaciones en biología, medicina y tecnología", Ed. Moreno-Díaz y Mira, Ed. Siglo Ventiuno S.A., Madrid, 1984.
- [More-86] R. Moreno-Díaz, "El sistema Visual Integrado en el Comportamiento Artificial", en "Visión Artificial", editado por la Asociación Española de Informática y Automática (AEIA), 1986.
- [More-87] R. Moreno-Díaz, J. Mira-Mira, "Architectures for Integrating Artificial Perception and Action", en Proc. Interkibernetik'87, pp. 1-11, Tarragona, 1987.
- [Murt-90] Murthy C.A., Pal S.K., "Fuzzy Thresholding: mathematical framework, bound functions and weighted moving average technique", Pattern Recognition Lett., 11, pp. 197-206, 1990.
- [Nahi-77] Nahi N.E., Jahanshahi M.H., "Image Boundary Estimation", IEEE Trans. Comput, Vol 26, pp. 772-781, Julio 1.977.
- [Nalw-86] Nalwa V.S., Binford T.O., "On Detecting Edges", IEEE Trans. on Pattern Anal. and Mach. Intell., Vol 8, pp. 699-714, 1.986.
- [Nazi-84] Nazif A., Levine M., "Low Level Image Segmentation: A Expert System", IEEE Trans. on Pattern Anal. and Mach. Intell., Vol 6, nº 5, pp. 555-577, 1.984.
- [Nebo-94] Nebot R., "Un editor integrado para el Procesador de Pixels", Proyecto de Diplomatura, Escuela Universitaria de Informática de la Universidad de Las Palmas de Gran Canaria, 1994.
- [Neva-80] Nevatia R., Babu K.R., "Linear Feature Extraction and Description", Computer Graphics and Image Processing, 13, pp. 257-269, 1980.
- [Neva-82] Nevatia R., "Machine Perception". Prentice-Hall, Englewood Cliffs, NJ, 1982.
- [Niem-90a] Niemann H., "Pattern Analysis and Understanding", Springer-Verlag, Berlin, 1990.
- [Niem-90b] Niemann H., Brünig H., Salzbrunn R., Schröder S., "A Knowledge-Based Vision System for Industrial Applications", Machine Vision and Applications, 3, pp 201-229, 1.990.
- [Ohla-78] Ohlander R., Price K., Reddy D.R., "Picture Segmentation Using a Recursive Region Splitting Method", Comp. Vision, Graph. and Image Proc., 8, pp. 313-333, 1.978.

- [Ohta-85] Otha Y., *"Knowledge-based Interpretation in Outdoor Natural Color Scenes"*, Pitman Pub., Londres, 1985.
- [Pal-88] Pal S.K., Rosenfeld A., *"Image enhancement and thresholding by optimization of fuzzy compactness"*, Pattern Recognition Lett., 7, pp. 77-86, 1988.
- [Pal-93] Pal N.R., Pal S., *"A review on image segmentation techniques"*, Pattern Recognition Vol. 26, n° 9, pp. 1277-1294, 1993.
- [Pavl-77] Pavlidis T., *"Structural Pattern Recognition"*, Springer Verlag, 1.977.
- [Pelle-94] Pellegretti P, Roli F., Serpico S.B., Vernazza G., *"Supervised Learning of Descriptions for Image Recognition Purposes"*, IEEE Trans. on Pattern Anal. and Mach. Intell., Vol 16, n° 1, pp. 92-98, 1.994.
- [Pun-81] Pun T., *"Entropic Thresholding: A New Approach"*, Comp. Vision, Graph. and Image Proc., 16, pp. 210-239, 1.981.
- [Ravi-88] Ravishankar R., Jain R., *"Knowledge Representation and Control in Computer Vision Systems"*, IEEE Expert, Vol. 3, n° 2, pp. 64-79, 1988.
- [Rich-91] Rich E., Knight K., *"Artificial Intelligence"*, McGraw-Hill, 2ª Edición, 1991.
- [Rise-77] Riseman E.M., Arbib M.A., *"Computational Techniques in the Visual Segmentation of Static Scenes"*, Comp. Vision, Graph., and Image Proc., 6, pp. 221-276, 1.977.
- [Rise-84] Riseman E.M., Hanson A.R., *"A Methodology for the Development of General Knowledge-Based Vision Systems"*, Proc. IEEE Workshop on Principles of Knowledge-Based Systems, pp. 159-170, Denver, Diciembre 1984.
- [Rise-87] Riseman E.M., Hanson A.R., *"A Methodology for the Development of General Knowledge-Based Vision Systems"*, en "Vision, Brain and Cooperative Computation", Ed. Arbib y Hanson, MIT Press., Cambridge Mass., pp. 285-328, 1987.
- [Rise-89] Riseman E.M., Hanson A.R., *"Computer Vision Research at the University of Massachusetts. Themes and Progress"*, Int. Journal of Computer Vision, n° 2, pp. 199-207, 1.989.
- [Ritt-90] Ritter G.X, Wilson J.N. and Davidson J.L., *"Image Algebra: An Overview"*, Computer Vision, Graphic and Image Processing, Vol. 49, pp. 297-331, 1990.
- [Rose-66] Rosenfeld A., Pfaltz J.L., *"Sequential operations in digital picture processing"*, Journal of ACM Vol. 12, n° 4, pp. 471-494, 1966.
- [Rose-81] Rosenfeld A., Smith R.C., *"Thresholding Using Relaxation"*, IEEE Trans. on Pattern Anal. and Mach. Intell., Vol. 3, pp. 598-606, 1.981.
- [Rose-82] Rosenfeld A., Kak A.C., *"Digital Picture Processing"*, (Vol. I y II), Academic Press, 1.982.
- [Rose-84] Rosenfeld A., *"The fuzzy geometry of image subsets"*, Pattern Recognition Lett., Vol. 2, pp. 311-317, 1984.
- [Ross-89] Ross Beveridge J., Griffith J., Kohler R., Hanson A.R., Riseman E.M., *"Segmenting Images Using Localized Histograms and Region Merging"*, Int. Journal of Computer Vision, 2, pp.311-347, 1989.

Referencias

- [Russ-64] Russell B., *"La Sabiduria de Occidente"*, Ed. Aguilar, Madrid, 1964.
- [Saho-88] Sahoo P.K., Soltani S., Wong K.C., *"A Survey of Thresholding Techniques"*, *Comp. Vision, Graph., and Image Proc.*, 41, 233-260, 1988.
- [Saka-85] Sakaue K., Tamura H., *"Automatic generation of image processing programs by knowledge-based verification"*, in *Proceedings of IEEE Conf. on Computer Vision and Pattern Recognition*, San Francisco, pp. 189-192, 1985.
- [Same-82] Samet H., *"Neighbor Finding Techniques for Images Represented by Quadrees"*, *Computer Graphics and Image Processing*, 18, pp. 37-57, 1982.
- [Same-86] Samet H., Tamminen M., *"An improved approach to connected component labeling of images"*, in *Proc Int. Conf. on Computer Vision and Pattern Recog., Miami Beach, Florida, IEEE Comp. Soc. Press, pp. 312-318, 1986.*
- [Sark-91] Sarkar S., Boyer K.L., *"On Optimal Infinite Impulse Response Edge Detection Filters"*, *IEEE Trans. on Pattern Anal. and Mach. Intell.*, Vol 13, n° 11, pp 1154-1171, 1991.
- [Sato-88] Sato H., Kitamura Y., Tamura H., *"A knowledge-based approach to vision algorithm design for industrial parts feeder"*, in *Proceedings of IAPR Workshop on Computer Vision-Special Hardware and Industrial Applications*, Tokio, pp. 413-416, 1988.
- [Shen-92] Shen J., Castan S., *"An Optimal Linear Operator for Step Edge Detection"*, *Comp. Vision, Graph. and Image Proc.*, 54, pp. 112-133, 1992.
- [Shil-82] Shiloach Y., Vishkin U., *"An $O(\log)$ parallel connectivity algorithm"*, *J. Algorithms*, Vol. 3, pp. 57-67, 1982.
- [Sota-89] Sotak G.E., Boyer K.L., *"The Laplacian of Gaussian Kernel: A Formal Analysis an Desing Procedure for Fast Accurate Convolution and Full-Frame Output"*, *Comp. Vision, Graph. and Image Proc.*, 48, pp. 147-189, 1989.
- [Tamu-83] Tamura H. et al., *"Design and Implementation of SPIDER: A transportable image processing software package"*, *Comp. Vision, Graph. and Image Proc.*, 23, pp. 273-294, 1983.
- [Tamu-84] Tamura H., Sakaue K., *"DIA (Digital Image Analysis) - Expert System: An approach to future vision system design"*, *Int. Symp. on Image Processing and its Applications*, Tokio, 1984.
- [Tana-88] Tanaka T., Sueda N., *"Knowledge acquisition in image processing expert system EXPLAIN"*, in *Proceedings of Int. Workshop on Artificial Intelligence for Industrial Applications*, Hitachi City, pp. 267-272, 1988.
- [Thur-92] Thurfjell L., Bengtsson E., Nordin B., *"A new three-dimensional connected components labeling algorithm with simultaneous object feature extraction capability"*, *GVGIP: Graphical Models and Image Processing*, Vol. 54, n° 4, pp. 357-364, 1992.
- [Tori-87] Toriu T., Iwase H., Yoshida M., *"An expert system for image processing"*, *FUJITSU Sci. Tech. J.*, Vol. 23, n° 2, pp. 111-118, 1987.
- [Triv-86] Trivedi M., Bezdek J.C., *"Low-Level segmentation of aerial imags with fuzzy clustering"*, *IEEE Trans. Syst. Man Cybern.*, Vol. 16, n° 4, pp. 589-598, 1986.

- [Tuck-86] Tucker L.W., "Labeling connected components on a massively parallel tree machine", In Proc. of Int. Conf. on Computer Vision and Pattern Recognition, Miami Beach, Florida, IEEE Comp. Soc. Press, 1986.
- [Udup-90] Udupa J., Ajjanagade V., "Boundary and object labeling in three-dimensional images", Comp. Vision, Graph. and Image Proc., 51, pp. 355-369, 1.990.
- [Vinc-91] Vincent L., Soille P., "Watersheds in Digital Spaces: An efficient algorithm based on immersion simulations". IEEE Trans. on Pattern Anal. and Mach. Intell., Vol 13, n° 6, pp. 583-598, 1991.
- [Weem-89] Weems C.C., Levitan S.P., Hanson A.R., Riseman E.M., Shu D.B., Nash J.G., "The Image Understanding Architecture". Int. Journal of Computer Vision, 2, pp. 251-282, 1989.
- [Weem-92] Weems C.C., Riseman E.M., Hanson A.R., "Image Understanding Architecture: Exploiting Potential Parallelism in Machine Vision". Computer, vol 25, n° 2, pp. 65-68. Febrero 1992.
- [Will-90] Williams D., Shah M., "Edge Contours Using Multiple Scales", Comp. Vision, Graph. and Image Proc., 51, pp. 256-274, 1.990.
- [Wils-88a] Wilson R., "Is Vision a Pattern Recognition Problem?", en "Pattern Recognition", Ed. J. Kittler (Ed), 1-25, Springer-Verlag, 1988.
- [Wils-88b] Wilson R., Spann M., "Image Segmentation and Uncertainty", Research Studies Press Ltd., 1.988.
- [Witk-84] Witkin A.P., "Scale space filtering: a new approach to multi-scale description", Image Understanding, S. Ullman and W. Richards, Eds., pp. 79-95, Ablex Publishing, N.J., 1984.
- [Wu-92] Wu Q.M., Rodd M.G., "Boundary Feature Extraction and Structural Verification of Objects", en "Advances in Machine Vision", C. Archibald and E. Petriu (Eds.), World Scientific Series in Computer Science Vol.23, World Scientific, pp.35-62, 1992.
- [Yang-88] Yang X. D., "Design of fast connected components hardware", in Proc. Int. Conf. Computer Vision and Pattern Recognition, Ann Arbor, MI, 1988.
- [Yang-92] Yang X. D., "An improved algorithm for labeling connected components in a binary image", in Computer Vision and Image Processing, Shapiro L. and Rosenfeld A. (Eds.), Academic Press, 1992.
- [Yuil-86] Yuille A., Poggio T., "Scaling Theorems for Zero Crossings", IEEE Trans. on Pattern Anal. and Mach. Intell., Vol 8, n° 1, pp. 15-25, 1.986.
- [Zade-65] Zadeh L.A., "Fuzzy Set", Information and Control, Vol. 8, pp. 338-353, 1965.
- [Zade-75] Zadeh L.A., "Calculus of Fuzzy Restrictions", en "Fuzzy Sets and their Applications to Cognitive and Decision Processes", Zadeh, Fu K.S., Tanaka T., Shimura M. (Eds.), Academic Press, Londres, 1975.
- [Zade-81] Zadeh L.A., "PRUF-a meaning representation language for natural languages", en "Fuzzy Reasoning and its Applications", Mandani y Gaines (Eds.), Academic Press, pp. 1-66, London, pp. 1-39, 1981.
- [Zade-83] Zadeh L.A., "Commonsense Knowledge Representation based on Fuzzy Logic", IEEE Computer, Vol. 16; n° 10, pp. 61-65, 1983.
- [Zuck-76] Zucker S.W., "Region Growing: Childhood and Adolescence", Comp. Vision, Graph. and Image Proc., Vol. 5, pp. 382-399, 1.976.

Apéndice A

Pseudocódigo de la partes más relevantes del AACC.




En las páginas que siguen se presentan sin solución de continuidad las partes más importantes del algoritmo de análisis de componentes conectadas. Algunos detalles de implementación han sido descritos previamente a lo largo de la presentación del algoritmo.

Variables globales


integer cur_col;	Columna activa de la imagen.
integer Line;	Línea activa de la imagen.
struct border * LeftCorn1Adr;	Punteros al borde de la esquina izquierda del borde pointer superior (1) e inferior (2) de la línea activa.
struct border * LeftCorn2Adr;	
struct border * UpEdge[];	UpEdge[i] (DownEdge[i]) es un puntero al borde ascendente (descendente) en la columna i.
struct border * DownEdge[];	
struct corner Corner[];	Vector cuyos elementos recogen información sobre las esquinas-C detectadas en la línea activa (tipo, columna, color).



procedure Process_C_Corners (número de esquinas-C)


```



begin
  reset dummy1, dummy2;
#ifdef NEIB8
  reset dummydiag;
#endif
  for ( i = 0; i < número de esquinas-C ; i++ ) {
    cur_col = Corner[i].col;
    tipo_de_esquina-C = Corner[i].type;
    down_border = DownEdge[cur_col];
    up_border = UpEdge[cur_col];
    switch (case tipo_de_esquina-C ) {
      case :
        LeftCorn1 = ; LeftCol1 = cur_col;
        LeftCorn2 = ; LeftCol2 = cur_col;
        LeftCol2 = LeftCol1 = cur_col;
        Color1 = Corner[i].Color;
        break;
    }
  }


```



case  :


```
EnterNeighborAtTail(up_border, down_border);
EnterNeighborAtHead(down_border, &dummy2);
EnterNeighborAtHead(&dummy2, up_border);
LeftCorn1 = ;
LeftCorn2 = ;
LeftCorn1Adr = down_border;
LeftCol2 = LeftCol1 = cur_col;
break;
```


case  :

```
LeftCorn1 = ;
LeftCorn1Adr = down_border;
LeftCorn2 = ;
LeftCorn2Adr = up_border;
LeftCol2 = LeftCol1 = cur_col;
break;
```

case  :

```
EnterNeighborAtTail(&dummy2, &dummy1);
Color2 = Corner[i].Color;
ProcessCornerPair(LeftCorn2, LeftCol2, , Color2);
up_border = UpEdge[cur_col];
TransferNeighborsToTail(up_border, &dummy2);
EnterNeighborAtHead(&dummy1, &dummy2);
EnterNeighborAtHead(&dummy2, up_border);
LeftCorn2 = ;
LeftCol2 = cur_col;
break;
```

case  :

```
EnterNeighborAtTail(&dummy2, up_border);
#ifdef NEIB8
EnterNeighborAtTail(&dummy2, down_border);
#endif
EnterNeighborAtTail(up_border, down_border);
#ifdef NEIB8
EnterNeighborAtTail(up_border, &dummydiag);
EnterNeighborAtHead(&dummydiag, up_border);
#endif
TransferNeighborsToTail(up_border, &dummy1);
ProcessCornerPair(LeftCorn1, LeftCol1, , -1);
```

```

        Color2 = Corner[i].Color;
        ProcessCornerPair(LeftCorn2, LeftCol2, ↖, Color2);
        up_border = UpEdge[cur_col];
        down_border = DownEdge[cur_col];
        TransferNeighborsToTail(up_border, &dummy2);
#ifdef NEIB8
        EnterNeighborAtHead(down_border, up_border);
#endif

        EnterNeighborAtHead(down_border, &dummy2);
        EnterNeighborAtHead(&dummy2, up_border);
#ifdef NEIB8
        TransferNeighborsToTail(&dummy2, &dummydiag);
#endif

        LeftCorn1 = ↘;
        LeftCorn1Adr = down_border;
        LeftCorn2 = ↙;
        LeftCol1 = LeftCol2 = cur_col;
        break;
#ifdef CONN8
    case ↗:
        TransferNeighborsToTail(up_border, &dummy1);
        TransferNeighborsToHead(down_border, &dummy2);
        if((up_border == LeftCorn1Adr)&&(LeftCorn1 == ↘)&&(up_border no tiene
            vecinos)) { /* up_border es el único vecino interno de down_border */
            EnterNeighborAtTail(up_border, down_border);
            EnterNeighborAtHead(down_border, up_border);
        }
        ProcessCornerPair(LeftCorn1, LeftCol1, ↗, -1);
        ProcessCornerPair(LeftCorn2, LeftCol2, ↖, -1);
        LeftCorn1 = ↗;
        LeftCorn2 = ↙;
        LeftCol1 = LeftCol2 = cur_col;
        Color1 = Corner[i].Color;
        break;
    case ↘:
        ProcessCornerPair(LeftCorn1, LeftCol1, ↘, Color1);
        Color2 = Corner[i].Color;
        ProcessCornerPair(LeftCorn2, LeftCol2, ↖, Color2);
        LeftCorn1 = ↘;
        LeftCorn1Adr = down_border;
        LeftCorn2 = ↗;
        LeftCorn2Adr = up_border;

```

```
LeftCol1 = LeftCol2 = cur_col;  
up_border = UpEdge[cur_col];  
down_border = DownEdge[cur_col];  
TransferNeighborsToHead(down_border, &dummy1);  
TransferNeighborsToTail(up_border, &dummy2);  
break;
```

#endif

case ↗:

```
EnterNeighborAtTail(&dummy2, up_border);  
EnterNeighborAtTail(up_border, down_border);  
EnterNeighborAtHead(down_border, &dummy2);  
TransferNeighborsToTail(up_border, &dummy1);  
ProcessCornerPair(LeftCorn1, LeftCol1, ↗, -1);  
LeftCorn1 = ↘;  
LeftCorn1Adr = down_border;  
LeftCol1 = cur_col;  
break;
```

case ↘:

```
ProcessCornerPair(LeftCorn1, LeftCol1, ↘, Color1);  
Color2 = Corner[i].Color;  
ProcessCornerPair(LeftCorn2, LeftCol2, ↘, Color2);  
up_border = UpEdge[cur_col];  
down_border = DownEdge[cur_col];  
TransferNeighborsToHead(down_border, &dummy1);  
TransferNeighborsToTail(up_border, &dummy2);  
break;
```

case ↖:

```
EnterNeighborAtTail(&dummy2, up_border);  
EnterNeighborAtTail(up_border, down_border);  
TransferNeighborsToTail(up_border, &dummy1);  
ProcessCornerPair(LeftCorn1, LeftCol1, ↖, -1);  
Color2 = Corner[i].Color;  
ProcessCornerPair(LeftCorn2, LeftCol2, ↖, Color2);  
up_border = UpEdge[cur_col];  
down_border = DownEdge[cur_col];  
TransferNeighborsToTail(up_border, &dummy2);  
EnterNeighborAtHead(down_border, up_border);  
break;
```

```

case  $\updownarrow$ :
    TransferNeighborsToTail(up_border, &dummy1);
    TransferNeighborsToHead(down_border, &dummy2);
    if((up_border == LeftCorn1Adr)&&(LeftCorn1 ==  $\downarrow$ )&&(up_border no tiene
    vecinos)) { /* up_border es el único vecino interno de down_border */
        EnterNeighborAtTail(up_border, down_border);
        EnterNeighborAtHead(down_border, up_border);
    }
    ProcessCornerPair(LeftCorn1, LeftCol1,  $\updownarrow$ , -1);
    ProcessCornerPair(LeftCorn2, LeftCol2,  $\updownarrow$ , -1);
} /* end switch */
} /* end for */
end;

```

procedure EnterNeighborAtHead (sujeto, objeto)

```

begin
    El borde apuntado por "sujeto" tiene una relación de vecindad con el borde apuntado por
    "objeto" en el punto (cur_col, Line). El registro de esta vecindad se introduce en la cabeza
    de la lista de objetos del borde "sujeto" y en la cabeza de la lista de sujetos del borde
    "objeto".
end

```

procedure EnterNeighborAtTail (sujeto, objeto)

```

begin
    El borde apuntado por "sujeto" tiene una relación de vecindad con el borde apuntado por
    "objeto" en el punto (cur_col, Line). El registro de esta vecindad se introduce en la cola
    de la lista de objetos del borde "sujeto" y en la cola de la lista de sujetos del borde
    "objeto".
end

```

procedure TransferNeighborsToHead (destino, origen)

```

begin
    Añade la lista de objetos (sujetos) del borde apuntado por "origen" en la cabeza de la lista
    de objetos (sujetos) del borde apuntado por "destino".
end

```

procedure TransferNeighborsToTail (destino, origen)

```

begin
    Añade la lista de objetos (sujetos) del borde apuntado por "origen" en la cola de la lista
    de objetos (sujetos) del borde apuntado por "destino".
end

```


procedure ProcessCornerPair (leftBcorn, left_col, rightBcorn, color)

begin

definir par_de_esquinas a partir de leftBcorn y rightBcorn;

switch (par_de_esquinas)

case ↖ : /* crear borde externo */

InitBorder (left_col, cur_col, OUTER, color);

break;

case ↗ : /* crear borde interno */

InitBorder (left_col, cur_col, INNER, color);

break;

case ↘ :

border1 = UpEdge[cur_col];

AppendBorderElementAtTail(border1, left_col, Line);

ShiftUp(cur_col, left_col);

if (border1->type == INNER)

border1->window_left_side = min(border1->window_left_side, left_col);

break;

case ↙ :

border1 = LeftCorn1Adr;

AppendBorderElementAtHead(border1, left_col, Line);

DownEdge[cur_col] = border1;

if (border1 == DownEdge[left_col]) {

DownEdge[left_col] = NULL;

}

DownEdge[cur_col]->Down = cur_col;

if (border1->type == INNER)

border1->window_right_side = max(border1->window_right_side, cur_col);

break;

case ↖ :

border1 = LeftCorn2Adr;

AppendBorderElementAtTail(border1, cur_col, Line);

UpEdge[cur_col] = border1;

if (border1 == UpEdge[left_col]) {

UpEdge[left_col] = NULL;

}

UpEdge[cur_col]->Up = cur_col;

if (border1->type == OUTER)

border1->window_right_side = max(border1->window_right_side, cur_col);

break;

```

case ↙ :
    border1 = DownEdge[cur_col];
    AppendBorderElementAtHead(border1, cur_col, Line);
    ShiftDown(cur_col, left_col);
    if ( border1→type == OUTER )
        border1→window_left_side = min(border1→window_left_side, left_col);
    break;

case ↘ :
    border1 = LeftCorn1Adr;
    border2 = UpEdge[cur_col];
    if ( border1 == DownEdge[border1→Down] ) {
        DownEdge[border1→Down] = NULL;
    }
    UpEdge[cur_col] = NULL;
    AppendBorderElementAtHead(border1, left_col, Line);
    AppendBorderElementAtTail(border2, left_col, Line);
    if ( border1 == border2 ) { /* cerrar un borde externo */
        CloseBorder(border1);
    }
    else {
        if ( border1→type == INNER ) && ( border2→type == INNER )
            MergeBorderAtHead(border2, border1);
        else if ( border1→type == OUTER )
            MergeBorderAtHead(border2, border1);
        else
            MergeBorderAtTail(border1, border2);
    }
    break;

case ↙ :
    border1 = LeftCorn2Adr;
    border2 = DownEdge[cur_col];
    if ( border1 == UpEdge[border1→Up] ) {
        UpEdge[border1→Up] = NULL;
    }
    DownEdge[cur_col] = NULL;
    AppendBorderElementAtHead(border2, cur_col, Line);
    AppendBorderElementAtTail(border1, cur_col, Line);
    if ( border1 == border2 ) { /* cerrar un borde interno */
        CloseBorder(border1);
    }

```

```
else {
    if ( border1→type == OUTER ) && ( border2→type == OUTER )
        MergeBorderAtHead(border1, border2);
    else if (border1→type == INNER )
        MergeBorderAtTail(border2, border1);
    else
        MergeBorderAtHead(border1, border2);
}
} /* end switch */;
end;
```

```
procedure ShiftUp (old_col, new_col)
begin
    UpEdge[new_col] = UpEdge[old_col];
    UpEdge[old_col] = NULL;
    UpEdge[new_col]→Up = new_col;
end
```

```
procedure ShiftDown (old_col, new_col)
begin
    DownEdge[new_col] = DownEdge[old_col];
    DownEdge[old_col] = NULL;
    DownEdge[new_col]→Down = new_col;
end
```

Apéndice B

Descripción BNF del Compilador de Procesador de Pixels

***** GENERAL *****

```
<pixp> ::= <definition> { <definition> }
<definition> ::= Define <body> EndDefine
<body> ::= <channel> | <picture> | <procedure> | <feature>
<body> ::= <classifier> | <rule> | <class> | <interface>
```

***** OBJETOS *****

```
<channel> ::= Channel <name> <channelattr>
<picture> ::= Picture <name> <pictureattr>
<procedure> ::= Procedure <name> <procedureattr>
<feature> ::= Feature <name> <featureattr>
<classifier> ::= Classifier <name> <classifierattr>
<rule> ::= Rule <name> <ruleattr>
<class> ::= Class <name> <classattr>
<interface> ::= Interface <interattr>
```

***** CHANNEL *****

```
<channelattr> ::= <type> <sizey> <sizeX> <level> <homothety> <function>
<type> ::= Type : <channeltype>
<sizey> ::= SizeY : <integer>
<sizeX> ::= SizeX : <integer>
<level> ::= Level : <integer>
<homothety> ::= Homothety : <homotype>
<function> ::= Function : <name>
<channeltype> ::= Grey | Rgb | Hsi | Secuence <integer> | Set <integer>
<homotype> ::= NoHomothety | Homothety3_2
```

***** PICTURE *****

```
<pictureattr> ::= <acquire> [ <file> ] [ <showpic> ]
<acquire> ::= Channel : <name>
<file> ::= File : <string>
<showpic> ::= Show [ <label> ] [ <zoom> ]
<label> ::= Label : <string>
<zoom> ::= Zoom : <integer>
```

Descripción BNF del Compilador del Procesador de Pixels

***** PROCEDURE *****

```
<procedureattr> ::= <function> <appl> <paramnum> [ <selector> ]
<appl> ::= AppliedTo: <arg>
<arg> ::= Picture [ <sliced> ] | Feature ArgNum: <integer> | Class
<sliced> ::= Sliced | NoSliced : <channeltype>
<paramnum> ::= ParamNum : <integer>
<selector> ::= Selector : <namelist>;
```

***** FEATURE *****

```
<featureattr> ::= <featureattr1> | >featureattr2>
<featureattr1> ::= From: Class <class_name>
<class_name> ::= Class : <name>
<featureattr2> ::= <from> <proc> [ <param> ]
<from> ::= From: <proced>
<proced> ::= Picture [ <slice> ] | Feature FeatureList: <featurelist>;
<proc> ::= Procedure : <name>
<slice> ::= Slice : <slicetype>
<param> ::= Parameter : <numberlist>;
<slicetype> ::= Red | Green | Blue
<slicetype> ::= Hue | Saturation | Intensity
<slicetype> ::= Time0 | Time1 | Time2
<slicetype> ::= Pic0 | Pic1 | Pic2
<featurelist> ::= <featur> { , <featur> }
<featur> ::= <name> | <name> . <name>
<namelist> ::= <name> { , <name> }
<numberlist> ::= <number> { , <number> }
<number> ::= <integer> | <float>
```

***** CLASSIFIER *****

```
<classifierattr> ::= <feat> <path> <functional> <decision>
<feat> ::= FeatureList : <featurelist1>;
<featurelist1> ::= <featur> { , <featur> }
<functional> ::= Functional : <functype>
<decision> ::= Decision : <dectype>
<functype> ::= <lineal> | <cuadratic> | Unitary
<lineal> ::= Lineal Coefficient : <numberlist>;
<cuadratic> ::= Cuadratic Mean : <numberlist>; Variance : <numberlist>;
<dectype> ::= <sigmodes> | <sigmoidep> | <threshold> | <cross> | <gaussian> | <ramp>
```

Appendice B

***** DECISION TYPES *****

<sigmodes> ::= Sigmoide-S <sign> <mode> <param2>
<sigmodep> ::= Sigmoide-P <mode> <param2>
<threshold> ::= Threshold <sign> <param1>
<cross> ::= Cross <param1>
<gaussian> ::= Gaussian
<ramp> ::= Ramp <sign> <param2>
<sign> ::= Sign : <sigtype>
<mode> ::= Mode : <modetype>
<param2> ::= Level : <number> , <number>
<param1> ::= Level : <number>
<sigtype> ::= Positive | Negative
<modetype> ::= Centered | Interval

***** RULE *****

<ruleattr> ::= <target> <condition> [<ertain>]
<target> ::= Target : <name>
<condition> ::= Condition : <premise>
<premise> ::= <atomic> { And <atomic> }
<atomic> ::= [Not] [Very | MoreLess] <predicate>
<predicate> ::= <name> | <name> (<pixel>)
<pixel> ::= NB <number> | AnyNB | AllNB
<ertain> ::= Certainty : <integer>

***** CLASS *****

<classattr> ::= [<instances>] [<classifier>] [<rules>] [<combine>] [<showclass>]
<instances> ::= Instances : <namelist>;
<classifier> ::= Classifier : <namelist>;
<rules> ::= Rule : <namelist>;
<combine> ::= CombineRule : <comb>
<comb> ::= Evidence | Logic
<showclass> ::= Show [<label>] [<zoom>] [<overlap>] [<whatslice>]
<overlap> ::= Overlapped : <integer> [<overlapmode>]
<overlapmode> ::= Filter_Fill
<whatslice> ::= Slice : <slicetype>

***** INTERFACE *****

<interattr> ::= <interclass> [<path>]
<interclass> ::= Class : <internamelist>;
<internamelist> ::= <name> { , <name> }

Apéndice C

Descripción BNF del Compilador del Presegmentador Flood

***** GENERAL *****

<flood> ::= <definition> { <definition> }
<definition> ::= Define <body> EndDefine
<body> ::= <s_makepartition> | <pixel_map> | <profiler> | <marker> | <prototype>

***** OBJETOS *****

<s_makepartition> ::= S_MakePartition <s_partition_attr>
<pixel_map> ::= Pixel_Map <name> <pixel_map_attr>
<profiler> ::= Profiler <name> <profiler_attr>
<marker> ::= Marker <name> <marker_attr>
<prototype> ::= Prototype <name> <prototype_attr>

***** S_MAKEPARTITION *****

<s_partition_attr> ::= <classlist1> <module> <program> [<protolist>]
[<parameter>] <connectivity> <neighborhood> [<showpartition>]
<classlist1> ::= P_Class : <namelist> ;
<module> ::= Presegmenter : <string>
<program> ::= ProgramName : <string>
<protolist> ::= Prototype : <namelist> ;
<connectivity> ::= Connectivity : <connectivitytype>
<connectivitytype> ::= 8-Connectivity [<precedence>] | 4-Connectivity
<precedence> ::= DiagonalPrecedence : <prot_name_list> ;
<prot_name_list> ::= * name | <namelist>
<neighborhood> ::= Neighborhood : 8-Connectivity | 4-Connectivity
<showpartition> ::= Show [<zoom>]

NOTA: El asterisco en <prot_name_list> indica que el prototipo cuyo nombre se da tiene la prioridad más baja cuando se resuelven conexiones diagonales ambiguas en la partición.

***** PIXEL_MAP *****

<pixel_map_attr> ::= <pclass> <profile> <weight> [<parameter>]
<pclass> ::= P_Class : <name>
<profile> ::= Profiler : <name>
<weight> ::= Weight : <integer>
<parameter> ::= Parameter : <numberlist> ;

Apéndice D

Descripción BNF del Compilador del Procesador de Segmentos

***** GENERAL *****

```
<segp> ::= <definition> { <definition> }
<definition> ::= Define <body> EndDefine
<body> ::= <p_interface> | <s_makepartition> | <s_procedure> | <s_feature> |
          <s_classifier> | <s_condition> | <s_rule> | <s_class> | <s_interface> |
          <s_localizer> | <s_ctrlrule> | <s_ctrlruleset> | <s_ctrliaction>
```

***** OBJETOS *****

```
<p_interface> ::= P_Interface <p_interface_attr>
<s_makepartition> ::= S_MakePartition <s_partition_attr>
<s_procedure> ::= S_Procedure <name> <s_procedure_attr>
<s_feature> ::= S_Feature <name> <s_feature_attr>
<s_classifier> ::= S_Classifier <name> <s_classifier_attr>
<s_condition> ::= S_Condition <name> <s_condition_attr>
<s_condition> ::= S_Localizer <name> <s_localizer_attr>
<s_rule> ::= S_Rule <name> <s_rule_attr>
<s_class> ::= S_Class <name> <s_class_attr>
<s_interface> ::= S_Interface <s_interface_attr>
<s_ctrlruleset> ::= S_CtrlRuleSet <s_ctrlruleset_attr>
<s_ctrlrule> ::= S_CtrlRule <name> <s_ctrlrule_attr>
<s_ctrliaction> ::= S_CtrlAction <name> <s_ctrliaction_attr>
```

***** P_INTERFACE *****

```
<p_interface_attr> ::= <classlist>
<classlist> ::= Class : <namelist>;
```

***** S_MAKEPARTITION *****

```
<s_partition_attr> ::= <classlist> <module> <program> [ <protolist> ]
                   [ <param> ] <connectivity> <neighborhood> [ <show> ]
<classlist> ::= P_Class: <namelist>;
<module> ::= Presegmenter : <presegname>
<presegname> ::= <name> | None
<protolist> ::= Prototype : <namelist>;
<program> ::= ProgramName : <file>
<connectivity> ::= Connectivity : <connectivitytype>
<connectivitytype> ::= <connect8> | 4-Connectivity
<connect8> ::= 8-Connectivity [ <precedence> ]
<precedence> ::= DiagonalPrecedence : <prot_namelist>;
<prot_namelist> ::= <last_name> | <namelist>
```

Apéndice D

<last_name> ::= * <name>
<neighborhood> ::= Neighborhood : <vicinity_type>
<vicinity_type> ::= 8-Connectivity | 4-Connectivity
<show> ::= Show [<zoom>]

NOTA: DiagonalPrecedence debe contener los nombres de los prototipos.
Caso de que sólo se deban especificar el prototipo de menor
precedencia, utilizar la sintaxis: "DiagonalPrecedence: * prot_name ;"

***** S_PROCEDURE *****

<s_procedure_attr> ::= <function> <appl> <paramnum> [<selector>]
<function> ::= Function: <name>
<appl> ::= AppliedTo: <arg> <argnum>
<arg> ::= P_Class | S_Feature | Partition
<argnum> ::= ArgNum: <integer>
<paramnum> ::= ParamNum: <integer>
<selector> ::= Selector: <namelist>;

***** S_FEATURE *****

<s_feature_attr> ::= From: <comefrom>
<comefrom> ::= <p_class_f> | <s_feature_f> | <s_class_f> | <partition_f>

<p_class_f> ::= P_Class List: <namelist>; <proc>
<s_feature_f> ::= S_Feature List: <featurelist>; <proc>
<s_class_f> ::= S_Class List: <name>;
<partition> ::= Partition [<protolist>] <proc>

<proc> ::= Procedure: <name> [<param>]
<param> ::= Parameter: <numberlist>;

<featurelist> ::= <featur> { , <featur> }
<featur> ::= <name> | <name>. <name>
<namelist> ::= <name> { , <name> }
<numberlist> ::= <number> { , <number> }
<number> ::= <integer> | <float>

***** S_CLASSIFIER *****

<s_classifier> ::= <feat> <functional> <decision>
<feat> ::= Feature: <featurelist>;
<functional> ::= Functional: <functype>
<decision> ::= Decision: <dectype>
<functype> ::= <lineal> | <quadratic> | Unitary
<lineal> ::= Lineal Coefficient: <numberlist>;
<quadratic> ::= Quadratic Mean: <numberlist>; Variance: <numberlist>;
<dectype> ::= <sigmodes> | <sigmodep> | <threshold> | <gaussian> | <ramp> | <trapezoid>

<sigmodes> ::= Sigmoide-S <sign> <mode> <param2>
<sigmodep> ::= Sigmoide-P <mode> <param2>
<threshold> ::= Threshold <sign> <param1>

Descripción BNF del Compilador del Procesador de Segmentos

<gaussian> ::= Gaussian
<ramp> ::= Ramp <sign> <param2>
<trapezoid> ::= Trapezoid <sign> <param4>

<sign> ::= Sign : <sigtype>
<mode> ::= Mode : <modetype>
<param1> ::= Level : <number>
<param2> ::= Level : <number> , <number>
<param4> ::= Level : <number> , <number> , <number> , <number>
<sigtype> ::= Positive | Negative
<modetype> ::= Center | Interval

***** S_CONDITION *****

<s_condition_attr> ::= Premise : <premlist>
<premlist> ::= <prematomlist> | <prematontrue>
<prematomlist> ::= <prematom> { And <prematom> }
<prematontrue> ::= Is(TRUE, [<qualifier1>] x)

<prematom> ::= Is(<property> { And <property> } , [<qualifier1>] x)
<property> ::= [<qualifier2>] [<order>] <name>
<qualifier1> ::= <neig> [<qualifier2>] <spatial>

<neig> ::= All | Any
<qualifier2> ::= [Not] [<linguistic>]
<linguistic> ::= Very | MoreLess
<spatial> ::= Above | Below | Left | Right | Beside | InBeside |
 Neib | ExNeib | InNeib | Contains | <UserDefined>
<order> ::= Gt (<integer>) | Eq (<integer>) | Lt (<integer>)
<UserDefined> ::= <name>

***** S_LOCALIZER *****

<s_localizer_attr> ::= Scope : <scope>
<scope> ::= Outers | Inners | Both

***** S_RULE *****

<s_rule_attr> ::= <target> <condition> [<certain>]
<target> ::= Target: <name>
<condition> ::= Condition : <premise>
<premise> ::= <atomic> { And <atomic> }
<atomic> ::= [Not] <name>
<certain> ::= Certainty : <integer>

Apéndice D

***** S_CLASS *****

```
<s_class_attr> ::= [ <instance> ] [ <classifier> ] [ <rule> ]
                [ <combine> ] [ <showclass> ]
<instance> ::= Instance : <namelist> ;
<classifier> ::= Classifier : <namelist> ;
<rule> ::= Rule : <namelist> ;
<combine> ::= CombineRule : <comb>
<comb> ::= Evidence | Logic
<showclass> ::= Show <level> [ <label> ] [ <zoom> ] [ <overlap> ]
<level> ::= Level : <integer>
<label> ::= Label : <string>
<zoom> ::= Zoom : <integer>
<overlap> ::= Overlapped: <file>
```

NOTA: file debe ser el nombre (sin extensión .Pic) de un fichero Pic ubicado en el directorio \$PIXHOME/Interface.

***** S_INTERFACE *****

```
<s_interface_attr> ::= <classlist>
<classlist> ::= Class : <internamelist> ;
<internamelist> ::= <name> { , <name> }
```

***** S_CTRLRULESET *****

```
<s_ctrlruleset_attr> ::= <strategy> <ctrlrules> <threshold>
<strategy> ::= Strategy : <strat1>
<strat1> ::= FirstBest | BestFirst
<ctrlrules> ::= CtrlRule : <name> { , <name> } ;
```

***** S_CTRLRULE *****

```
<s_ctrlrule_attr> ::= <ctrlrulelist> <action> <evaluation> <threshold>
<ctrlrulelist> ::= Condition : <conditionlist> ;
<conditionlist> ::= [Not] <name> { And [Not] <name> }
<evaluation> ::= Evaluation : <evaluationmode>
<evaluationmode> ::= Unconditional | <conditional>
<conditional> ::= Conditional <diagnostics>
<diagnostics> ::= Diagnostic: <diagnostic_list> ;
<diagnostic_list> ::= [Not] <classname> { And [Not] <classname> }
<action> ::= Action : <name>
```

Descripción BNF del Compilador del Procesador de Segmentos

***** S_CTRLACTION *****

```
<s_ctrlaction_attr> ::= Action : <pos_actions>
<pos_actions> ::= <merge> | <include> | Split | Lock | Unlock
<merge> ::= Merge <locus>
<include> ::= Include <locus>
<locus> ::= <target_cond> <threshold>
<target_cond> ::= [ Not ] <name>
<conditionlist> ::= [Not] <name> { And [Not] <name> }
```

NOTA: <name> es el nombre de una s_condition que debe contener un localizador.
Esta s_condition puede ser del tipo *Is(True, [Not][qualificador] localizador x)*
y debe servir para seleccionar un segmento vecino como candidato para una acción
MERGE o INCLUDE.

NOTA: La S_Condition utilizada en <target_cond> debe cumplir las siguientes condiciones:

- Debe contener un único localizador y contener sólo una premisa.
- Esta premisa puede ser del tipo *Is(True, ... x)*.
- El modificador ALL no puede utilizarse en s_conditions utilizadas a su vez como objetivos (targets).

Apéndice E

Programas utilizados en los casos de estudio y no incluidos en el capítulo IV.

1. Localización e identificación de piezas industriales (Epígrafe IV.3.1 apartado C).

SCREWS.KS

```
Define Picture ScrewsPicture
  Channel: DTfile
  File : "../Picture/screws.Pic"
  Show
EndDefine
```

```
Define Feature gray
  From: Picture
  Procedure: Value
EndDefine
```

```
Define Feature crdist
  From: Feature
  FeatureList: off.offset;
  Procedure: Dir4CrossDist
  Parameter: 12,10;
EndDefine
```

```
Define Feature off
  From: Picture
  Procedure: PicStatistic4
  Parameter: 4;
EndDefine
```

```
Define Feature statcr
  From: Feature
  FeatureList: crdist;
  Procedure: ValStatistic2
  Parameter: 5;
EndDefine
```

```
Define Classifier cl_gray
  FeatureList: gray;
  Functional: Unitary
  Decision: Ramp
  Sign: Positive
  Level: 0,255
EndDefine
```

```
Define Classifier cl_crxy
  FeatureList: statcr.mean;
  Functional: Unitary
  Decision: Ramp
  Sign: Positive
  Level: 3.5,5.5
EndDefine
```

```
Define Class OffZeroCross
  Classifier: cl_crxy;
  Show
  Overlapped: 80
EndDefine
```

```
Define Class screws100
  Classifier: cl_gray;
EndDefine
```

```
Define Interface
  Class: OffZeroCross,
  screws100;
EndDefine
```

SCREWS.FC

```
Define Pixel_Map offset_zero_cross
  P_Class: OffZeroCross
  Profiler: Gradient5x5
  Weight: 1
EndDefine
```

```
Define Pixel_Map image
  P_Class: screws100
  Profiler: Gradient5x5
  Weight: 2
EndDefine
```

```

Define Prototype screw_arm
  MapList:      offset_zero_cross;
  Centroid:     95;
  Dispersion:   5;
  Marker:       WinStandardMarker
  Mode:         Merge
  Color:        Red
  Parameter:    5, 25, 88;
  Connectivity: 8-Connectivity
  Show
    PictureFile: screws
EndDefine

```

```

Define Prototype not_screw_arm
  MapList:      offset_zero_cross;
  Centroid:     30;
  Dispersion:   40;
  Marker:       WinStandardMarker
  Mode:         Merge
  Color:        Green
  Parameter:    0, 0, 180;
  Connectivity: 4-Connectivity
EndDefine

```

SCREWS.SC

```

Define P_Interface
  Class:      OffZeroCross,
             screws100;
EndDefine

```

```

Define S_MakePartition
  P_Class:    OffZeroCross,
             screws100;
  Presegmenter: "FC"
  ProgramName: "screws"
  Connectivity: 8-Connectivity
  Neighborhood: 8-Connectivity
EndDefine

```

```

Define S_Feature area
  From:      Partition
  Procedure: Area
EndDefine

```

```

Define S_Classifier cl_area
  Feature:    area;
  Functional: Unitary
  Decision:   Threshold
  Sign:       Positive
  Level:      150
EndDefine

```

```

Define S_Class Large
  Classifier: cl_area;
EndDefine

```

```

Define S_CtrlRuleSet
  Strategy:   FirstBest
  CtrlRule:   MergeSmall;
  Threshold:  90
EndDefine

```

```

Define S_CtrlRule MergeSmall
  Condition:  cnsmall
  Action:     MergeSmall&Large
  Threshold:  90
EndDefine

```

```

Define S_CtrlAction MergeSmall&Large
  Action:     Merge
  Target:     cn_largeneib
  Threshold:  90
EndDefine

```

```

Define S_Condition cnsmall
  Premise:    Is (Not Large, X)
EndDefine

```

```

Define S_Condition cnlargeneib
  Premise:    Is (Large, Neib X)
EndDefine

```

```

Define S_Interface
  Class:      Large;
EndDefine

```

NUTS.FC

```

Define S_MakePartition
  P_Class:    screw_arm, screws100;
  Presegmenter: "FC"
  ProgramName: "nuts"
  Prototype:  screw_arm,
             not_screw_piece;
  Parameter:  1.0, 2.0, 256;
  Connectivity: 4-Connectivity
  Neighborhood: 4-Connectivity
  Show
EndDefine

```

```

Define Pixel_Map screw_arm
  P_Class:    screw_arm
  Profiler:   None
  Weight:     0
EndDefine

```

Apéndice E

```
Define Pixel_Map image
  P_Class: screws100
  Profiler: Gradient5x5
  Weight: 1
EndDefine

Define Prototype screw_arm
  MapList: screw_arm;
  Centroid: 100;
  Dispersion: 1;
  Marker: WinFreeze
  Mode: Merge
  Color: White
  Parameter: 0,0,256;
  Connectivity: 4-Connectivity
EndDefine

Define Prototype background
  MapList: screw_arm, image;
  Centroid: 0, 0;
  Dispersion: 0, 9;
  Marker: WinStandardMarker
  Mode: Merge
  Color: Green
  Parameter: 3,9,100;
  Connectivity: 4-Connectivity
EndDefine

Define Prototype not_screw_piece
  MapList: screw_arm, image;
  Centroid: 0, 60;
  Dispersion: 0, 45;
  Marker: WinStandardMarker
  Mode: Not_Merge
  Parameter: 10,36,35;
  Connectivity: 8-Connectivity
EndDefine

NUTS.SC

Define S_MakePartition
  P_Class: screw_arm, screws100;
  Presegmenter: "FC"
  ProgramName: "nuts"
  Prototype: screw_arm,
             not_screw_piece,
             background;
  Parameter: 1.0,2.0;
  Connectivity: 4-Connectivity
  Neighborhood: 4-Connectivity
  Show
EndDefine

Define P_Interface
  Class: screw_arm, screws100;
EndDefine

Define S_Feature size
  From: Partition
  Procedure: Area
EndDefine

Define S_Classifier cl_tiny
  Feature: size;
  Functional: Unitary
  Decision: Sigmoide-S
  Sign: Negative
  Mode: Interval
  Level: 190, 195
EndDefine

Define S_Class tiny
  Classifier: cl_tiny;
EndDefine

Define S_Classifier cl_large
  Feature: size;
  Functional: Unitary
  Decision: Sigmoide-S
  Sign: Positive
  Mode: Interval
  Level: 250, 260
EndDefine

Define S_Class large
  Classifier: cl_large;
EndDefine

Define S_Classifier cl_huge
  Feature: size;
  Functional: Unitary
  Decision: Sigmoide-S
  Sign: Positive
  Mode: Interval
  Level: 1000, 1010
EndDefine

Define S_Class huge
  Classifier: cl_huge;
EndDefine

Define S_Feature screw_arm_rat
  From: Partive
  Prototype: screw_arm;
  Procedure: PrototypeRatio
EndDefine
```



```
Define S_Classifier cl_screw_arm
  Feature: screw_arm_rat;
  Functional: Unitary
  Decision: Sigmoide-S
    Sign: Positive
    Mode: Interval
    Level: 0.0, 0.8
EndDefine
```

```
Define S_Class screw_arm
  Classifier: cl_screw_arm;
EndDefine
```

```
Define S_Feature not_arm_rat
  From: Partition
  Prototype: not_screw_piece;
  Procedure: PrototypeRatio
EndDefine
```

```
Define S_Classifier cl_not_arm
  Feature: not_arm_rat;
  Functional: Unitary
  Decision: Sigmoide-S
    Sign: Positive
    Mode: Interval
    Level: 0.0, 0.8
EndDefine
```

```
Define S_Class not_arm
  Classifier: cl_not_arm;
EndDefine
```

```
Define S_Feature n_holes
  From: Partition
  Procedure: NumberOfHoles
EndDefine
```

```
Define S_Classifier cl_holes
  Feature: n_holes;
  Functional: Unitary
  Decision: Threshold
    Sign: Positive
    Level: 0.5
EndDefine
```

```
Define S_Class has_holes
  Classifier: cl_holes;
EndDefine
```

```
Define S_Rule rl_nut
  Target: nut
  Condition: cn_nut
  Certainty: 100
EndDefine
```

```
Define S_Class nut
  Rule: rl_nut;
EndDefine
```

```
Define S_Interface
  Class: nut, screw_arm;
EndDefine
```

```
Define S_CtrlRuleSet
  Strategy: FirstBest
  CtrlRule: MergeNotArm1, MergeNotArm2;
  Threshold: 70
EndDefine
```

```
Define S_CtrlRule MergeNotArm1
  Condition: cn_merge1 And cn_neib_mergeable;
  Action: MergeNotArm
  Evaluation: Unconditional
  Threshold: 70
EndDefine
```

```
Define S_CtrlRule MergeNotArm2
  Condition: cn_merge2 And cn_neib_mergeable;
  Action: MergeNotArm
  Evaluation: Unconditional
  Threshold: 70
EndDefine
```

```
Define S_Condition cn_nut
  Premise: Is(has_holes And not_arm, X)
EndDefine
```

```
Define S_Condition cn_merge1
  Premise: Is(not_arm And large And Not huge, X)
EndDefine
```

```
Define S_Condition cn_merge2
  Premise: Is(not_arm And large, X)
EndDefine
```

```
Define S_Condition cn_neib_mergeable
  Premise: Is(not_arm And tiny, Neib X)
EndDefine
```

```
Define S_CtrlAction MergeNotArm
  Action: Merge
  Target: cn_neib_mergeable
  Threshold: 70
EndDefine
```

Apéndice E

2. Segmentación de contornos (Epígrafe IV.3.2)

CONTOURS.KS

```
Define Picture picture
  Channel: DT_COLOR_file
  File : "../Picture/outer.Pic"
  Show
EndDefine

Define Feature Statis16Red
  From: Picture
  Slice: Red
  Procedure: PicStatistic3
  Parameter: 16;
EndDefine

Define Feature Statis16Green
  From: Picture
  Slice: Green
  Procedure: PicStatistic3
  Parameter: 16;
EndDefine

Define Feature Statis16Blue
  From: Picture
  Slice: Blue
  Procedure: PicStatistic3
  Parameter: 16;
EndDefine

Define Classifier CrossRed
  FeatureList: Statis16Red.offset,
  Functional: Unitary
  Decision: Cross
  Level: 0.0
EndDefine

Define Classifier CrossGreen
  FeatureList: Statis16Green.offset,
  Functional: Unitary
  Decision: Cross
  Level: 0.0
EndDefine

Define Classifier CrossBlue
  FeatureList: Statis16Blue.offset,
  Functional: Unitary
  Decision: Cross
  Level: 0.0
EndDefine

Define Class Contour
  Classifier: CrossRed, CrossBlue,
  CrossGreen;
  Show
  Overlapped: 70
EndDefine

Define Feature Statis8Red
  From: Picture
  Slice: Red
  Procedure: PicStatistic2
  Parameter: 8;
EndDefine

Define Feature Statis8Green
  From: Picture
  Slice: Green
  Procedure: PicStatistic2
  Parameter: 8;
EndDefine

Define Feature Statis8Blue
  From: Picture
  Slice: Blue
  Procedure: PicStatistic2
  Parameter: 8;
EndDefine

Define Classifier High_VarRed
  FeatureList: Statis8Red.variance;
  Functional: Unitary
  Decision: Threshold
  Sign: Positive
  Level: 80.0
EndDefine

Define Classifier High_VarGreen
  FeatureList: Statis8Green.variance;
  Functional: Unitary
  Decision: Threshold
  Sign: Positive
  Level: 80.0
EndDefine

Define Classifier High_VarBlue
  FeatureList: Statis8Blue.variance;
  Functional: Unitary
  Decision: Threshold
  Sign: Positive
  Level: 80.0
EndDefine
```

```

Define Class High_Var
  Classifier: High_VarRed,
             High_VarGreen,
             High_VarBlue;
EndDefine

Define Rule GoodContourRule
  Target: GoodContour
  Condition: Contour And High_Var
EndDefine

Define Class GoodContour
  Rule: GoodContourRule;
  Show
EndDefine

Define Feature Feat_Contour
  From: Class
  Class: GoodContour
EndDefine

Define Feature Eroded_Cont
  From: Feature
  FeatureList: Feat_Contour,
  Procedure: ContourErode
EndDefine

Define Feature NoJoints_Cont
  From: Feature
  FeatureList: Eroded_Cont,
  Procedure: RemoveJoints
EndDefine

Define Classifier Ok_Contour
  FeatureList: NoJoints_Cont,
  Functional: Unitary
  Decision: Threshold
  Sign: Positive
  Level: 50
EndDefine

Define Class Ok_Contour
  Classifier: Ok_Contour;
  Show
EndDefine

Define Interface
  Class: Ok_Contour,
EndDefine

```

CONTOURS.FC

```

Define S_MakePartition
  P_Class: Ok_Contour,
  Presegmenter: "FC"
  ProgramName: "contours"
  Prototype: contour_piece,
             background;
  Parameter: 1,0,2,0;
  Connectivity: 8-Connectivity
  Neighborhood: 8-Connectivity
  Show
EndDefine

Define Pixel_Map contour
  P_Class: Ok_Contour
  Profiler: CurvatureThreshold
  Weight: 1
  Parameter: 6,20,2;
EndDefine

Define Prototype contour_piece
  MapList: contour,
  Centroid: 99;
  Dispersion: 1;
  Marker: ConStandardMarker
  Mode: Not_Merge
  Parameter: 8, 60;
  Connectivity: 8-Connectivity
  Show
  PictureFile: Ok_Contour
EndDefine

Define Prototype background
  MapList: contour,
  Centroid: 0;
  Dispersion: 0;
  Marker: ConBackGroundMarker
  Mode: Merge
  Color: Darkgreen
  Parameter: 1;
  Connectivity: 4-Connectivity
EndDefine

Define Prototype small_contour
  MapList: contour,
  Centroid: 99;
  Dispersion: 1;
  Marker: ConSmall
  Mode: Merge
  Color: Darkgreen
  Parameter: 40;
  Connectivity: 8-Connectivity
EndDefine

```

Apéndice E

CONTOURS.SC

```
Define P_Interface
  Class:      Ok_Contour,
EndDefine

Define S_MakePartition
  P_Class:    Ok_Contour,
  Presegmenter: "FC"
  ProgramName: "contours"
  Connectivity: 8-Connectivity
  Neighborhood: 8-Connectivity
  Show
    Zoom: 1
EndDefine

Define S_Feature ratios
  From:      Partition
  Procedure:  Max_acu_error
EndDefine

Define S_Classifier cl_slkd_line
  Feature:    ratios.slkd;
  Functional: Unitary
  Decision:   Trapezoid
    Sign: Positive
    Level: 0.9, 0.9, 1.25, 1.35
EndDefine

Define S_Classifier cl_modline
  Feature:    ratios.modline;
  Functional: Unitary
  Decision:   Threshold
    Sign: Negative
    Level: 800
EndDefine

Define S_Classifier cl_slkd_arc
  Feature:    ratios.slkd;
  Functional: Unitary
  Decision:   Trapezoid
    Sign: Positive
    Level: 1.5, 1.5, 1.9, 2
EndDefine

Define S_Classifier cl_slal_arc
  Feature:    ratios.slal;
  Functional: Unitary
  Decision:   Trapezoid
    Sign: Positive
    Level: 1.15, 1.18, 1.25, 1.30
EndDefine

Define S_Classifier cl_modarc
  Feature:    ratios.modarc;
  Functional: Unitary
  Decision:   Threshold
    Sign: Negative
    Level: 90
EndDefine

Define S_Class line_slkd
  Classifier: cl_slkd_line;
EndDefine

Define S_Class line_modline
  Classifier: cl_modline;
EndDefine

Define S_Rule rl_line
  Target:     Line
  Condition:  cn_line
EndDefine

Define S_Condition cn_line
  Premise:    ls(line_slkd,X) And
              ls(line_modline,X)
EndDefine

Define S_Class Line
  Rule:       rl_line;
  Show
    Level: 80
EndDefine

Define S_Class arc_slal
  Classifier: cl_slal_arc;
EndDefine

Define S_Class arc_slkd
  Classifier: cl_slkd_arc;
EndDefine

Define S_Class arc_modarc
  Classifier: cl_modarc;
EndDefine

Define S_Class Arc
  Rule:       rl_arc;
  Show
    Level: 80
EndDefine

Define S_Rule rl_arc
  Target:     Arc
  Condition:  cn_arc
EndDefine
```

Define S_Condition *cn_arc*
Premise: Is(*arc_slal*,X) And
 Is(*arc_slkd*,X) And
 Is(*arc_modarc*,X)

EndDefine

Define S_Interface
Class: *Line, Arc*;
EndDefine

3. Segmentación desde propiedades morfológicas (Epígrafe IV.3.3).

GRAINS.KS

Define Picture *GrainsPicture*

Channel: *DTfile*
File : *"../Picture/grains.Pic"*
Show
EndDefine

Define Feature *gray*

From: *Picture*
Procedure: *Value*
EndDefine

Define Classifier *cl_gray*

FeatureList: *gray;*
Functional: *Unitary*
Decision: *Ramp*
Sign: *Positive*
Level: *0,256*
EndDefine

Define Class *grains100*

Classifier: *cl_gray;*
Show
EndDefine

Define Interface

Class: *grains100;*
EndDefine

GRAINS.1.FC

Define S_MakePartition

P_Class: *grains100;*
Presegmenter: *"FC"*
ProgramName: *"grains.1"*
Parameter: *1.0,2.0;*
Connectivity: *4-Connectivity*
Neighborhood: *4-Connectivity*
Show

Zoom: *2*

EndDefine

Define Pixel_Map *gray*

P_Class: *grains100*
Profiler: *Gradient3x3*
Weight: *1*
EndDefine

Define Prototype *particle*

MapList: *gray;*
Centroid: *15;*
Dispersion: *15;*
Marker: *WinStandardMarker*
Mode: *Not_Merge*
Parameter: *5, 10, 155;*
Connectivity *4-Connectivity*
Show
PictureFile: *grains100*
EndDefine

Define Prototype *background*

MapList: *gray;*
Centroid: *70;*
Dispersion: *35;*
Marker: *WinStandardMarker*
Mode: *Merge*
Color: *Green*
Parameter: *1, 1, 255;*
Connectivity: *4-Connectivity*
EndDefine

GRAINS.2.FC

Define S_MakePartition

P_Class: *previous_part;*
Presegmenter: *"FC"*
ProgramName: *"grains.2"*
Parameter: *1.0,1.0;*
Connectivity: *8-Connectivity*
Neighborhood: *8-Connectivity*
Show

Zoom: *2*

EndDefine

Define Pixel_Map *previous*

P_Class: *previous_part*
Profiler: *MorphoProfile*
Weight: *1*
Parameter: *8;*
EndDefine

Define Prototype *particle*

MapList: *previous;*
 Centroid: *0;*
 Dispersion: *2;*
 Marker: *WinGradMinima*
 Mode: *Not_Merge*
 Parameter: *14, 9, 222;*
 Connectivity: *8-Connectivity*
 Show
 PictureFile: *grains100*

EndDefine

Define Prototype *background*

MapList: *previous;*
 Centroid: *100;*
 Dispersion: *2;*
 Marker: *WinFreeze*
 Mode: *Merge*
 Color: *Green*
 Parameter: *1, 1, 256;*
 Connectivity: *8-Connectivity*

EndDefine

GRAINS.SC

Define S_MakePartition

P_Class: *grains100,*
previous_part;
 Presegmenter: *"FC"*
 ProgramName: *"grains.2"*
 Parameter: *1.0,1.0;*
 Connectivity: *8-Connectivity*
 Neighborhood: *8-Connectivity*
 Show
 Zoom: *2*

EndDefine

Define P_Interface

Class: *grains100;*

EndDefine

Define S_Feature *circularity*

From: *Partition*
 Procedure: *Circularity*

EndDefine

Define S_Classifier *cl_circular*

Feature: *circularity;*
 Functional: *Unitary*
 Decision: *Threshold*
 Sign: *Positive*
 Level: *0.54*

EndDefine

Define S_Feature *radius*

From: *Partition*
 Procedure: *MaxMinRadius*

EndDefine

Define S_Feature *ratio*

From: *S_Feature*
 List: *radius.max, radius.min;*
 Procedure: *Divide*

EndDefine

Define S_Classifier *cl_ratio*

Feature: *ratio;*
 Functional: *Unitary*
 Decision: *Ramp*
 Sign: *Negative*
 Level: *1.2, 2*

EndDefine

Define S_Classifier *cl_no_elongated*

Feature: *ratio;*
 Functional: *Unitary*
 Decision: *Threshold*
 Sign: *Negative*
 Level: *2.00*

EndDefine

Define S_Class *No_Elongated*

Classifier: *cl_no_elongated;*

EndDefine

Define S_Class *Circular*

Classifier: *cl_circular;*

EndDefine

Define S_Rule *rl_circular*

Target: *Round_Particle*
 Condition: *cn_circular*

EndDefine

Define S_Condition *cn_circular*

Premise: *Is(No_Elongated And*
Circular, X)

EndDefine

Define S_Class *Round_Particle*

Classifier: *cl_ratio;*
 Rule: *rl_circular;*
 Show

Level: *70*

EndDefine

Define S_Interface

Class: *Round_Particle;*

EndDefine

4. Segmentación de imágenes de exteriores (Epígrafe IV.3.4).

OUTDOOR.KS

Define Picture *picture*
 Channel: DT_COLOR_file
 File : "../Picture/outdoor.Pic"
 EndDefine

Define Feature *i123*
 From: Picture
 Procedure: *Pic123*
 EndDefine

Define Feature *stat_i1*
 From: Feature
 FeatureList: *i123.I1;*
 Procedure: *ValStatistic2*
 Parameter: 4;
 EndDefine

Define Classifier *bright*
 FeatureList: *stat_i1.mean;*
 Functional: Unitary
 Decision: Sigmoide-S
 Sign: Positive
 Mode: Interval
 Level: 130,135
 EndDefine

Define Class *bright*
 Classifier: *bright;*
 EndDefine

Define Classifier *dark*
 FeatureList: *stat_i1.mean;*
 Functional: Unitary
 Decision: Sigmoide-S
 Sign: Negative
 Mode: Interval
 Level: 70,75
 EndDefine

Define Class *dark*
 Classifier: *dark;*
 EndDefine

Define Classifier *uniform*
 FeatureList: *stat_i1.variance;*
 Functional: Unitary
 Decision: Sigmoide-S
 Sign: Negative
 Mode: Interval
 Level: 120,125
 EndDefine

Define Class *uniform*
 Classifier: *uniform;*
 EndDefine

Define Classifier *variable*
 FeatureList: *stat_i1.variance;*
 Functional: Unitary
 Decision: Sigmoide-S
 Sign: Positive
 Mode: Interval
 Level: 205,210
 EndDefine

Define Class *variable*
 Classifier: *variable;*
 EndDefine

Define Feature *stat_i2*
 From: Feature
 FeatureList: *i123.I2;*
 Procedure: *ValStatistic2*
 Parameter: 4;
 EndDefine

Define Classifier *blue*
 FeatureList: *stat_i2.mean;*
 Functional: Unitary
 Decision: Sigmoide-S
 Sign: Negative
 Mode: Interval
 Level: -30,-25
 EndDefine

Define Class *blue*
 Classifier: *blue;*
 EndDefine

Define Feature *stat_i3*
 From: Feature
 FeatureList: *i123.I3;*
 Procedure: *ValStatistic2*
 Parameter: 4;
 EndDefine

Define Classifier *green*
 FeatureList: *stat_i3.mean;*
 Functional: Unitary
 Decision: Sigmoide-S
 Sign: Positive
 Mode: Interval
 Level: 0,5
 EndDefine


```

Define Class green
  Classifier: green;
EndDefine

Define Rule sky_rule
  Target: sky
  Condition: bright And blue
  Certainty: 90
EndDefine

Define Class sky
  Rule: sky_rule;
EndDefine

Define Rule artif_rule
  Target: artif
  Condition: uniform And Not dark And
             Not blue And Not green
  Certainty: 90
EndDefine

Define Class artif
  Rule: artif_rule;
EndDefine

Define Rule artif_shadow_rule
  Target: artif_shadow
  Condition: dark And Not green
  Certainty: 90
EndDefine

Define Class artif_shadow
  Rule: artif_shadow_rule;
EndDefine

Define Feature red_grad
  From: Picture
  Slice: Red
  Procedure: PicGradient1
  Parameter: 5;
EndDefine

Define Feature green_grad
  From: Picture
  Slice: Green
  Procedure: PicGradient1
  Parameter: 5;
EndDefine

Define Feature blue_grad
  From: Picture
  Slice: Blue
  Procedure: PicGradient1
  Parameter: 5;
EndDefine

```

```

Define Classifier cl_color_grad
  FeatureList: red_grad, green_grad,
              blue_grad;
  Functional: Lineal
  Coefficient: 0.3333, 0.3333,
              0.3333;
  Decision: Ramp
  Sign: Positive
  Level: 0,255
EndDefine

Define Class color_grad
  Classifier: cl_color_grad;
EndDefine

Define Interface
  Class: green, sky,
        color_grad, artif,
        artif_shadow;
EndDefine

```

OUTDOOR.FC

```

Define S_MakePartition
  P_Class: color_grad, green,
           artif, sky,
           artif_shadow;
  Presegmenter: "FC"
  ProgramName: "outdoor"
  Prototype: green, artif, sky,
            artif_shadow;
  Parameter: 1,0,2,0;
  Connectivity: 4-Connectivity
  Neighborhood: 4-Connectivity
  Show
EndDefine

Define Pixel_Map graylevel
  P_Class: color_grad
  Profiler: ProfileFromMap
  Weight: 1
  Parameter: 2
EndDefine

Define Pixel_Map green
  P_Class: green
  Profiler: None
  Weight: 0
EndDefine

Define Pixel_Map sky
  P_Class: sky
  Profiler: None
  Weight: 0
EndDefine

```

Apéndice E

Define Pixel_Map *artif*

P_Class: *artif*
Profiler: *None*
Weight: *0*

EndDefine

Define Pixel_Map *artif_shadow*

P_Class: *artif_shadow*
Profiler: *None*
Weight: *0*

EndDefine

Define Prototype *green*

MapList: *green, sky, artif,*
artif_shadow;
Centroid: *80,0,0,0;*
Dispersion: *30,0,0,0;*
Marker: *WinStandardMarker*
Mode: *Merge*
Color: *Green*
Parameter: *3, 4, 50;*
Connectivity: *4-Connectivity*

EndDefine

Define Prototype *artif*

MapList: *green, sky, artif,*
artif_shadow;
Centroid: *0,0,80,0;*
Dispersion: *0,0,30,0;*
Marker: *WinStandardMarker*
Mode: *Merge*
Color: *Yellow*
Parameter: *3, 4, 50;*
Connectivity: *4-Connectivity*

EndDefine

Define Prototype *artif_shadow*

MapList: *green, sky, artif,*
artif_shadow;
Centroid: *0,0,0,80;*
Dispersion: *0,0,0,30;*
Marker: *WinStandardMarker*
Mode: *Merge*
Color: *Red*
Parameter: *3, 4, 50;*
Connectivity: *4-Connectivity*

EndDefine

Define Prototype *sky*

MapList: *green, sky, artif,*
artif_shadow;
Centroid: *0,80,0,0;*
Dispersion: *0,30,0,0;*
Marker: *WinStandardMarker*
Mode: *Merge*

Color: *Blue*
Parameter: *3, 4, 50;*
Connectivity: *4-Connectivity*

EndDefine

OUTDOOR.SC

Define P_Interface

Class: *color_grad, artif,*
artif_shadow, sky,
green;

EndDefine

Define S_MakePartition

P_Class: *color_grad, artif,*
artif_shadow, sky,
green;
Presegmenter: *"FC"*
ProgramName: *"outdoor"*
Prototype: *sky, artif, artif_shadow,*
green;
Parameter: *1.0,2.0;*
Connectivity: *4-Connectivity*
Neighborhood: *4-Connectivity*
Show

EndDefine

Define S_Feature *size*

From: *Partition*
Procedure: *Area*

EndDefine

Define S_Classifier *cl_tiny*

Feature: *size;*
Functional: *Unitary*
Decision: *Sigmoide-S*
Sign: *Negative*
Mode: *Interval*
Level: *50, 60*

EndDefine

Define S_Class *tiny*

Classifier: *cl_tiny;*

EndDefine

Define S_Classifier *cl_low_tiny*

Feature: *size;*
Functional: *Unitary*
Decision: *Sigmoide-S*
Sign: *Negative*
Mode: *Interval*
Level: *200, 300*

EndDefine

```
Define S_Class low_tiny
  Classifier: cl_low_tiny;
EndDefine
```

```
Define S_Classifier cl_large
  Feature: size;
  Functional: Unitary
  Decision: Sigmoide-S
  Sign: Positive
  Mode: Interval
  Level: 3000, 3500
EndDefine
```

```
Define S_Class large
  Classifier: cl_large;
EndDefine
```

```
Define S_Classifier cl_huge
  Feature: size;
  Functional: Unitary
  Decision: Sigmoide-S
  Sign: Positive
  Mode: Interval
  Level: 13000, 13500
EndDefine
```

```
Define S_Class huge
  Classifier: cl_huge;
EndDefine
```

```
Define S_Feature green_ratio
  From: Partition
  Prototype: green;
  Procedure: PrototypeRatio
EndDefine
```

```
Define S_Classifier cl_green
  Feature: green_ratio;
  Functional: Unitary
  Decision: Sigmoide-S
  Sign: Positive
  Mode: Interval
  Level: 0.0, 0.8
EndDefine
```

```
Define S_Class green
  Classifier: cl_green;
EndDefine
```

```
Define S_Feature sky_ratio
  From: Partition
  Prototype: sky;
  Procedure: PrototypeRatio
EndDefine
```

```
Define S_Classifier cl_blue
  Feature: sky_ratio;
  Functional: Unitary
  Decision: Sigmoide-S
  Sign: Positive
  Mode: Interval
  Level: 0.0, 0.8
EndDefine
```

```
Define S_Class blue
  Classifier: cl_blue;
EndDefine
```

```
Define S_Feature artif_ratio
  From: Partition
  Prototype: artif;
  Procedure: PrototypeRatio
EndDefine
```

```
Define S_Classifier cl_artif
  Feature: artif_ratio;
  Functional: Unitary
  Decision: Sigmoide-S
  Sign: Positive
  Mode: Interval
  Level: 0.0, 0.8
EndDefine
```

```
Define S_Class artif
  Classifier: cl_artif;
EndDefine
```

```
Define S_Feature shadow_ratio
  From: Partition
  Prototype: artif_shadow;
  Procedure: PrototypeRatio
EndDefine
```

```
Define S_Classifier cl_shadow
  Feature: shadow_ratio;
  Functional: Unitary
  Decision: Sigmoide-S
  Sign: Positive
  Mode: Interval
  Level: 0.0, 0.8
EndDefine
```

```
Define S_Class shadow
  Classifier: cl_shadow;
EndDefine
```

```
Define S_Rule r1_sky1
  Target: sky
  Condition: cn_sky1
EndDefine
```

Apéndice E

```
Define S_Rule rl_sky2
  Target:      sky
  Condition:   cn_sky2
  Certainty:  90
EndDefine

Define S_Class sky
  Rule:        rl_sky1, rl_sky2;
EndDefine

Define S_Rule rl_window
  Target:      window
  Condition:   cn_prewindow And
              cn_all_neib_artif
  Certainty:  90
EndDefine

Define S_Class window
  Rule:        rl_window;
  Show:
  Level:      50
  Overlapped: "color_grad"
EndDefine

Define S_Rule rl_front1
  Target:      front
  Condition:   cn_front1
  Certainty:  100
EndDefine

Define S_Rule rl_front2
  Target:      front
  Condition:   cn_prefront2 And
              cn_neib_artif And Not
              cn_neib_green
  Certainty:  100
EndDefine

Define S_Class front
  Rule:        rl_front1, rl_front2;
EndDefine

Define S_Rule rl_rule
  Target:      road
  Condition:   cn_road
  Certainty:  90
EndDefine

Define S_Class road
  Rule:        rl_road;
EndDefine

Define S_Interface
  Class:       sky, road, front, window;
EndDefine

Define S_CtrlRuleSet
  Strategy:    FirstBest
  CtrlRule:    FindWindow,
              IncludeShadowInGreen,
              IncludeGreenInShadow,
              MergeArtif, MergeFront,
  Threshold:   70
EndDefine

Define S_CtrlRule FindWindow
  Condition:   cn_prewindow And
              cn_tiny_neib_artif,
  Action:      MergeWindow
  Evaluation:  Conditional
  Diagnostic:  window;
  Threshold:   85
EndDefine

Define S_CtrlRule IncludeShadowInGreen
  Condition:   cn_inc_shadowgreen;
  Action:      IncludeShadowInGreen
  Evaluation:  Unconditional
  Threshold:   70
EndDefine

Define S_CtrlRule IncludeGreenInShadow
  Condition:   cn_inc_greenshadow;
  Action:      IncludeGreenInShadow
  Evaluation:  Unconditional
  Threshold:   70
EndDefine

Define S_CtrlRule MergeArtif
  Condition:   cn_merge_artif;
  Action:      MergeArtif
  Evaluation:  Unconditional
  Threshold:   70
EndDefine

Define S_CtrlRule MergeFront
  Condition:   cn_merge_front;
  Action:      MergeArtif
  Evaluation:  Unconditional
  Threshold:   70
EndDefine

Define S_Condition cn_prefront2
  Premise:     Is(shadow, X) And
              Is(sky, Any Neib X)
EndDefine

Define S_Condition cn_sky1
  Premise:     Is(blue And Not tiny, X)
EndDefine
```

Define S_Condition *cn_sky2*
 Premise: Is(*artif* And *large*, X) And
 Is(*huge* And *green*, Beside X)

EndDefine

Define S_Condition *cn_all_neib_artif*
 Premise: Is(*artif* And *large*, Any Beside X)
 And Is(Not *green*, Any Beside X)
 And Is(Not *sky*, All Neib X)

EndDefine

Define S_Condition *cn_prewindow*
 Premise: Is(*shadow* And
 Not Very *large*
 And Not *tiny*, X)

EndDefine

Define S_Condition *cn_road*
 Premise: Is(*artif* And *huge*, X) And
 Is(Not Very *front*, X) And
 Is(Not *sky*, All Neib X)

EndDefine

Define S_Condition *cn_front1*
 Premise: Is(*artif* And *large*, X) And
 Is(*window*, Any Neib X)

EndDefine

Define S_Condition *cn_inc_shadowgreen*
 Premise: Is(*large* And *green*, Beside X)
 And Is(Very *tiny* And *shadow*, X)

EndDefine

Define S_Condition *cn_inc_greenshadow*
 Premise: Is(*large* And *shadow*, Beside X)
 And Is(Very *tiny* And *green*, X)

EndDefine

Define S_Condition *cn_neib_shadow*
 Premise: Is(*shadow*, Any Neib X)

EndDefine

Define S_Condition *cn_neib_green*
 Premise: Is(*green*, Any Neib X)

EndDefine

Define S_Condition *cn_neib_artif*
 Premise: Is(*artif*, Any Neib X)

EndDefine

Define S_Condition *cn_tiny_neib_artif*
 Premise: Is(*artif* And *low_tiny*,
 Any Neib X)

EndDefine

Define S_Condition *cn_merge_artif*
 Premise: Is(Not *large* And *artif*, X) And
 Is(*large* And *artif*, Any Neib X)

EndDefine

Define S_Condition *cn_merge_front*
 Premise: Is(*artif*, X) And
 Is(*front*, Any Neib X)

EndDefine

Define S_CtrlAction *MergeWindow*
 Action: Merge
 Target: *cn_tiny_neib_artif*
 Threshold: 85

EndDefine

Define S_CtrlAction *MergeArtif*
 Action: Merge
 Target: *cn_neib_artif*
 Threshold: 70

EndDefine

Define S_CtrlAction *IncludeShadowInGreen*
 Action: Include
 Target: *cn_neib_green*
 Threshold: 70

EndDefine

Define S_CtrlAction *IncludeGreenInShadow*
 Action: Include
 Target: *cn_neib_shadow*
 Threshold: 70

EndDefine