Líneas de Nivel y Shape Context

Tesis de Licenciatura

23 de marzo de 2010

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Alumno:	Francisco Gómez Fernández (LU: 158/03)	fgomez@dc.uba.ar
Director:	Mariano H. Tepper	mtepper@dc.uba.ar
Co-Directora:	Marta E. Mejail	marta@dc.uba.ar

Resumen

A lo largo de este trabajo, nos esforzamos por responder el principal cuestionamiento existente en todo proceso de reconocimiento de formas: ¿una forma está o no presente en una imagen?

No existe una solución trivial a este problema, y para poder abordarlo dividimos el proceso de reconocimiento de formas en tres etapas: detección de características, elaboración de descriptores, y búsqueda de correspondencias, en donde hemos puesto nuestro foco en la detección de características.

En el contexto de la detección de características presentamos un nuevo procedimiento para calcular las regiones extremas máximamente estables de una imagen basado en el árbol de inclusión de formas o componentes de una imagen, que impone un mínimo de restricciones sobre el árbol de entrada. Cuando es aplicado al árbol de formas de una imagen interpolada bilinealmente presenta especiales cualidades para detectar los objetos bien contrastados, y lo denominamos como criterio de selección de formas máximamente estables.

Con el objetivo de completar el proceso de reconocimiento de formas, hemos propuesto un reconocedor basado en *shape context*, denominado *shape contex morfológico* por su cualidad de trabajar con formas. El *shape context morfológico* nos da la posibilidad de enfrentar el problema de la búsqueda de formas en imágenes al poseer la importante propiedad de describir una forma de manera semi-local. Por lo tanto, es adecuado para trabajar en el proceso de reconocimiento de formas en donde los objetos deben ser puestos en correspondencia individualmente, en contraposición de los enfoques globales que abarcan toda (o casi) la imagen. Este hecho es corroborado experimentalmente con los resultados presentados, en donde, se puede ver que el *shape context morfológico* permite establecer correspondencia entre formas de dos imágenes en donde hay también muchas otras formas presentes.

Abstract

Throughout this work, we strive to answer the main question any shape recognition method: is a shape present or not in a picture?

There is no trivial solution to this problem and in order to address we divide the pattern recognition process into three stages: feature detection, encoding of descriptors, and matching. We put our focus on feature detection.

In the context of feature detection, we present a new process for calculating the *maximally stable extremal regions* of an image based on the inclusion tree of shapes or components of an image, which imposes minimal restrictions on the input tree. When it is applied to the tree of shapes of a bilinear interpolated image, it is particularly good of detecting well-contrasted objects, and it is called *maximally stable shapes* criterion.

In order to complete the process of shape recognition, we have proposed a shape recognizer based on *shape context*, called *morphological shape context* because of its ability to work with shapes. The semi-locality property in morphological shape context gives us the possibility of addressing the problem of searching shapes in images. Therefore, it is appropriate for working in shape recognition where individual objects present in the image must be matched separately, as opposed to global approaches that cover all (or almost all) the

image. This fact is confirmed experimentally with the results presented, where we can see that the morphological shape context allows us to match shapes in contexts where other shapes are present.

Índice general

Capítulo 1

Introducción

En este trabajo abordamos el problema de la detección de características en imágenes en el marco de un proceso de reconocimiento de formas. El proceso de reconocimiento de formas es parte del reconocimiento de patrones en imágenes, cuyo objetivo principal es decidir si un objeto está presente o no en una escena.

Un sistema de reconocimiento de formas por visión en computadora puede ser dividido en tres partes: detección de características, elaboración de descriptores, y búsqueda de correspondencias.

El proceso de detección de características es una fase muy importante que, en muchos casos, puede determinar el éxito o fracaso de todo el sistema. Una característica puede describirse como una una parte "interesante" de una imagen, que le da cualidad distintiva, es decir, que sirve para distinguirla de sus semejantes.

Una imagen –capturada a través de un lente óptico– plasma en un medio físico, una escena del mundo real. Una escena está compuesta por los objetos que se encuentran presentes en ella. Por lo tanto, si lo que se desea es trabajar con objetos, es necesario contar con una representación de la imagen que posea mayor información de la que nos puede brindar su conjunto de píxeles.

La percepción visual recae en lo bordes de los objetos de la imagen [Att54], y por ende, en éstos se concentra la información de la imagen. Los bordes en una imagen coinciden con partes de líneas de nivel [DMM08], de esta manera una representación de una imagen constituida por el conjunto de líneas de nivel de ella, nos brinda la base para manipular los objetos visuales.

Por otro lado, los factores que influyen en la adquisición y captura de una imagen de una escena, e.g. cámara, proceso de digitalización, iluminación, medio de reproducción, etc., no afectan la percepción visual de la misma, es decir, independientemente del contraste de la imagen, los objetos percibidos se mantienen inalterados [Wer38]. Estas premisas nos compelen a utilizar representaciones de una imagen que sean invariantes a cambios de contraste.

El mapa topográfico de una imagen establece una representación completa e invariante a cambios de contraste, la cual está formada por el conjunto de líneas de nivel de ella. Las líneas de nivel de una imagen de definen como las fronteras topológicas de las componentes conexas de los conjuntos de nivel.

Las componentes conexas de los conjuntos de nivel de una imagen pueden organizarse en una estructura arbórea de inclusión, obtenida del umbralado de los niveles de gris. Esta estructura se denomina árbol de componentes y posee

la importante característica de que las componentes de adecuan muy bien a los objetos de la escena.

El árbol de componentes presenta la desventaja de que es incapaz de representar los objetos claros y oscuros de una imagen simultáneamente, ya que, la inclusión entre componentes se define por su nivel de gris, y no por su inclusión geométrica. Con el ánimo de superar este inconveniente surge el árbol de formas, en el cual una forma esta incluida en otra si yace en su interior, otorgándonos una herramienta sólida para poder hablar de la estructura geométrica de la imagen.

El árbol de componentes y el de formas, ambos constituyen una representación completa e invariante a cambios de contraste. La representación es completa debido a que es posible reconstruir la imagen original desde sus elementos, e invariante a cambios de contraste, ya que ambos están formados a partir de los conjuntos de nivel de una imagen, y éstos se mantienen bajo cambios alteraciones del contraste.

En este trabajo, desarrollamos un nuevo detector de características denominado formas máximamente estables, que establece un criterio de selección sobre el árbol de formas de una imagen, inspirado en el método de regiones extremas máximamente estables [MCMP02].

En el proceso de reconocimiento de formas una vez obtenidas las características más salientes de la imagen, se procede a la codificación o representación de las mismas. En este trabajo proponemos fusionar el descriptor *shape context* [BMP02] con las líneas de nivel en lo que denominamos *shape context morfológico*. El descriptor *shape context* es poderoso y redundante descriptor de formas que por su naturaleza nos brinda una descripción semi-local, en contraposición de los descriptores globales ([RC96], [KH90]) que actúan sobre la totalidad de la imagen, permitiéndonos que los objetos individuales presentes en la imagen puedan ponerse en correspondencia separadamente.

En el último paso del proceso, es cuándo debemos responder a la pregunta: ¿dos formas son semejantes o no? Esta pregunta se nos presenta cuando debemos establecer correspondencias entre dos conjuntos de descriptores, en donde uno de ellos representa la forma a buscar y el otro la forma candidata. Luego, un objeto puede estar representado por muchas formas, y también, pueden existir numerosos candidatos a comparar. Por lo tanto debemos poder establecer un criterio para medir la semejanza entre descriptores y de este modo elegir los más similares. Este es un problema conocido bajo el nombre de *matching en grafos bipartitos*.

La etapa de *matching*, generalmente, es la menos estudiada de todo el proceso de reconocimiento de formas. La mayoría de los métodos usan una técnica de vecinos más cercanos para corresponder los conjuntos de descriptores.

En este trabajo implementamos un proceso un criterio *a contrario* de *matching* para la correspondencia de descriptores *shape context*, que se caracteriza por ser robusto y estable, y además nos brinda la ventaja de poder evaluar independientemente la calidad de cada una de las correspondencias establecidas entre los descriptores, manejando un único parámetro.

Los resultados de nuestro proceso de reconocimiento de formas muestran que el *shape context morfológico* es apropiado para trabajar en situaciones en las que el proceso de reconocimiento de formas *shape context* [BMP02] no tiene éxito.

La tesis está organizada de la siguiente manera.

En el capítulo 2, comenzaremos con una revisión de los conceptos básicos de la morfología matemática, para luego introducirnos en las descripciones topológicas de una imagen: el árbol de componentes, árbol de formas y el árbol de líneas de nivel o mapa topográfico.

En el capítulo 3, nos abocaremos a la tarea de detección de características en el desarrollo del detector de formas máximamente estables, inspirado en las regiones extremas máximamente estables. También estudiaremos el detector de contornos significativos que está muy bien condicionado para la detección de características, ya que obtiene un subconjunto mínimo de formas coincidentes con la percepción visual humana.

En el capítulo 4, estudiamos el uso de los contornos significativos y las formas máximamente estables como dos métodos de detección de características aplicados en un nuevo proceso reconocimiento de formas denominado *shape context morfológico*, basado en *shape context*. También se presentan los resultados de utilizar el proceso de reconocimiento de formas *shape context morfológico*, en aplicaciones de búsqueda en video por contenido, establecimiento de correspondencias entre cuadros de video, y establecimiento de correspondencias para un posterior apareamiento estéreo entre cámaras.

Capítulo 2

Topología de una Imagen

En imágenes naturales, el contraste depende de muchos factores: del tipo de cámara, del proceso de digitalización, de la iluminación, de la pantalla en que se la muestre, etc. A pesar de estos factores que afectan el contraste, la percepción de la imagen se mantiene inalterada. En otras palabras, la información geométrica en una imagen prima sobre la información de contraste. La invariabilidad a cambios de contraste, originalmente fue propuesta como un principio gestáltico por Wertheimer [Wer38].

La *morfología matemática* se encarga de estudiar las características de las imágenes que son invariables a cambios de contraste, en vez de preocuparse por sus niveles de gris. Los conjuntos de nivel constituyen los elementos más básicos de la morfología matemática; a través de éstos es posible crear una representación de una imagen, completa e invariable a cambios de contraste [Mat75] [Ser83]. Una alternativa de esta representación se obtiene al considerar los contornos[1] de los conjuntos de nivel, las líneas de nivel, en lo que se denomina *mapa topográfico* [CCM99].

La segmentación de una imagen, usualmente, se define como una partición en regiones conexas, donde son de especial interés las relaciones entre estas regiones. Una vez segmentada la imagen, se debe buscar una adecuada descripción para la topología resultante. Si consideramos los conjuntos de nivel, éstos no representan una segmentación de la imagen, sin embargo, poseen un orden y, por ende, se podrían describir con una lista ordenada. Esto tiene una gran desventaja para la representación de los objetos de una imagen y es la falta de *localidad*, ya que los conjuntos de nivel no están conectados. De aquí surge la necesidad de utilizar las componentes conexas de los conjuntos de nivel.

Las componentes conexas de los conjuntos de nivel superior o inferior pueden representarse mediante una estructura de árbol ordenada por inclusión. Sin embargo, considerar únicamente los conjuntos de nivel superior o inferior por separado no es suficiente para describir topológicamente la imagen, debido a que unos están adaptados a los objetos claros y otros a los oscuros.

En el árbol de formas [Mon00], se mezclan conjuntamente las componentes conexas de los conjuntos de nivel superior e inferior en una estructura de árbol ordenadas por inclusión, en donde se manejan adecuadamente los huecos. El árbol de formas describe una correcta topología de la imagen.

[1] Traducción libre del inglés *boundary*.

En este capítulo, comenzaremos con una revisión de los conceptos básicos de la morfología matemática, para luego introducirnos en las descripciones topológicas de una imagen: el árbol de componentes, árbol de formas y el árbol de líneas de nivel o mapa topográfico, que serán utilizados intensamente en los siguientes capítulos.

2.1. Imágenes

Vamos a definir a una imagen como una función $u : D \to V$, donde el dominio D representará las posiciones de los puntos de la imagen, y el codominio V serán sus niveles de gris o intensidades.

Diremos que la imagen u es *digital* o *discreta* cuando $D \subset \mathbb{Z}^2$, y cuando $D \subset \mathbb{R}^2$ diremos que la imagen es de dominio continuo. En el contexto de imágenes digitales, los puntos de la imagen son también llamados píxeles.

Generalemente $V \subset \mathbb{Z}$, ó $V \subset \mathbb{R}$ y, en este último caso, diremos que la imagen posee valores reales o *flotantes*.

También vamos tener en cuenta cómo fueron capturadas las imágenes que analizaremos. Una imagen *natural* será aquella que provenga de la captura de una escena del mundo real con una cámara digital o análogica, y una imagen *sintética* será aquella producida artificialmente por el hombre o la computadora.

2.2. Conjuntos de nivel

Los conjuntos de nivel superior de una imagen son los conjuntos de puntos cuyo nivel de gris es mayor o igual a un cierto umbral. Análogamente, se definen los conjuntos de nivel inferior como los conjuntos de puntos cuyo nivel de gris es menor o igual a un cierto umbral. Ver figura 2.1.

Definición 2.1. El **conjunto de nivel** superior de valor $\lambda \in V$ de u se denomina $\mathcal{X}_\lambda u$ y se define como:

$$\mathcal{X}_\lambda u = \{x \mid u(x) \geq \lambda\}$$

De igual manera, para el conjunto de nivel inferior:

$$\mathcal{X}^\lambda u = \{x \mid u(x) < \lambda\}$$

El objetivo de tomar la comparación por menor estricto $<$ para los conjuntos de nivel inferior, es lograr consistencia entre ambos: $D \setminus \mathcal{X}^\lambda u = \mathcal{X}_\lambda u$. Esta es una convención, y, se puede ver que para imágenes discretas es equivalente tomar la desigualdad no estricta en ambas definiciones.

Los conjuntos de nivel de una imagen nos brindan una representación completa de la imagen. Por lo tanto, es posible reconstruir la imagen desde sus conjuntos de nivel superiores o inferiores.

Figura 2.1: (a) Imagen original compuesta por un círculo gris y un cuadrado negro. Conjunto de nivel inferior correspondiente: (b) al color gris y (c) al negro. Conjunto de nivel superior correspondiente: (d) al color negro y (e) al gris.

Proposición 2.1 (Principio de superposición)**.** Una imagen puede ser reconstruida a partir de sus conjuntos de nivel inferior:

$$\forall x,\, u(x) = \inf\{\lambda \mid x \in \mathcal{X}^{\lambda} u\}$$

o de los conjuntos de nivel superior:

$$\forall x,\, u(x) = \sup\{\lambda \mid x \in \mathcal{X}_{\lambda} u\}$$

El proceso de reconstrucción de una imagen sugerido por el principio de superposición consiste en ir superponiendo los conjuntos de nivel como si fueran capas, una sobre la otra, para finalmente obtener la imagen original; de aquí el nombre.

Otra propiedad importante de los conjuntos de nivel es que son invariables a cambios de contraste. El cambio de contraste en una imagen se puede definir como la aplicación de una función continua y monótona (decreciente o creciente) de todos los valores de los píxeles de la imagen de entrada. La imagen así transformada tendrá la misma familia de conjuntos de nivel que antes.

Proposición 2.2 (Invariabilidad de contraste)**.** Sean $u : D \to V$, $v : D \to V$, tales que existe una función $g : V \to V$ estrictamente creciente, tal que $v = g \circ u$, es decir, $\forall x \in D, v(x) = g(u(x))$ entonces:

$$\forall \lambda \in V, \mathcal{X}^{\lambda} u = \mathcal{X}^{g(\lambda)} v \quad \text{y} \quad \forall \lambda \in V, \mathcal{X}_{\lambda} u = \mathcal{X}_{g(\lambda)} v$$

La propiedad anterior establece que en las imágenes u y v existen los mismos conjuntos de nivel, pero asociados a distintos niveles.

Algo interesante para destacar de los conjuntos de nivel es la propiedad de inclusión entre ellos que se muestran en las siguientes proposiciónes.

Proposición 2.3 (Inclusión de $\mathcal{X}^{\lambda}$)**.** Sean $\lambda_1, \lambda_2, \ldots\ \lambda_{n-1}, \lambda_n$, tal que $\lambda_i < \lambda_{i+1}$ $\forall i\ 0 < i < n$, se cumple que $\mathcal{X}^{\lambda_1} \subseteq \mathcal{X}^{\lambda_2} \subseteq \ldots\ \subseteq \mathcal{X}^{\lambda_{n-1}} \subseteq \mathcal{X}^{\lambda_n}$

Proposición 2.4 (Inclusión de $\mathcal{X}_{\lambda}$)**.** Sean $\lambda_1, \lambda_2, \ldots\ \lambda_{n-1}, \lambda_n$, tal que $\lambda_i > \lambda_{i+1}$ $\forall i\ 0 < i < n$, se cumple que $\mathcal{X}_{\lambda_n} \subseteq \mathcal{X}_{\lambda_{n-1}} \subseteq \ldots\ \subseteq \mathcal{X}_{\lambda_2} \subseteq \mathcal{X}_{\lambda_1}$

Un conjunto de nivel inferior estará incluido en los conjuntos de nivel inferior de mayor nivel de gris a él; es decir, los píxeles de valor de gris menor a λ, también serán menores a $\lambda + k$, para un λ fijo y para todo k positivo.

Similarmentes, los conjuntos de nivel superior estarán incluidos en los conjuntos de nivel superior de menor nivel.

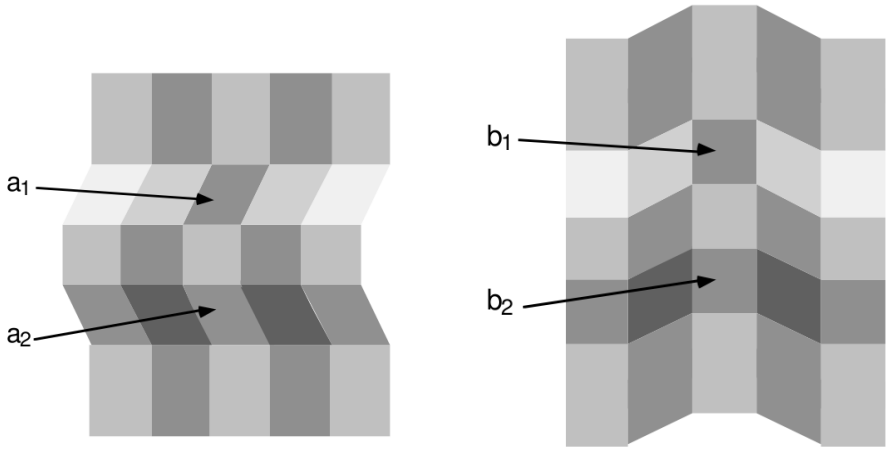
En la figura 2.2 se ven los conjuntos de nivel inferior para una imagen real en donde se puede ver la inclusión de los mismos. En la figura 2.2b, se muestran en negro los puntos pertenecientes al conjunto de nivel 127 y, en 2.2c, el conjunto de nivel 192, donde se puede apreciar que incluye los puntos de 2.2b.

(a) (b) (c)

Figura 2.2: (a) Imagen original. Sus conjuntos de nivel $\mathcal{X}^{127}$ en la subfigura (b) y $\mathcal{X}^{192}$ en la subfigura (c), donde se puede apreciar que $\mathcal{X}^{127} \subset \mathcal{X}^{192}$

Los conjuntos de nivel tienen una desventaja importante y es que no son suficientemente compatibles con nuestra percepción visual en la representación de lo que consideramos objetos visuales. El ojo humano tiene facilidad para comparar intensidades de luz, pero estas comparaciones no parecen ser globales: para dos regiones adyacentes podemos decir con facilidad cuál es más oscura o más clara, pero si no son adyacentes esta decisión no siempre puede darse (ver figura 2.3). Además, cada región homogénea en la imagen es percibida como un objeto y, por lo tanto, no es partida por el ojo humano. El hecho de que dos regiones disjuntas formen parte del mismo conjunto de nivel no es una información relevante para el análisis de imágenes. Estas ideas nos impulsan a trabajar con las componentes conexas de los conjuntos de nivel.

Antes de continuar, debemos precisar la noción de conexión que utilizamos.



(a) (b)

Figura 2.3: En ambas figuras los niveles de gris de los cuadrados indicados con flechas son iguales, pero el contexto hace que sean percibidos como distintos. En (a), el nivel de girs de los cuadrados a_1 y a_2 es idéntico, pero la geometría y el contexto de la figura hace que a_2 sea percibido como más claro. Nuevamente, en la figura (b), el nivel de gris de b_1 y b_2 es el mismo, pero el cuadrado indicado como b_1 se percibe como más oscuro. Figuras extraídas del artículo de Adelson [Ade93].

Definición 2.2. Sean $p, q \in D$, se define $A \subset D$ una relación de adyacencia entre puntos como

$$pAq \Leftrightarrow p \in \Gamma(q)$$

donde $\Gamma : D \to \mathcal{P}(D)$ se denomina *vecindad* y $\mathcal{P}(D)$ denota el conjunto de partes de D.

La función de vecindad $\Gamma(q)$ devuelve el conjunto de puntos adyacentes a q en la imagen u definida en el dominio D. El problema de definir la adyacencia o conexión entre los puntos de una imagen está intrínsicamente ligado al marco de estudio en el que trabajemos. Muchas de las propiedades vistas a lo largo de este capítulo fueron estudiadas en un marco mucho más general utilizando herramientas teóricas de la morfología matemática [Mon00].

A lo largo de este trabajo, se trabajará con relaciones de adyacencia simétricas, es decir $pAq \Rightarrow qAp$.

2.3. Árbol de Componentes

Las componentes conexas de los conjuntos de nivel pueden organizarse en una estructura jerárquica de inclusión determinada por su nivel de gris, llamada árbol de componentes. El árbol de componentes constituye una representación completa de una imagen e invariable a cambios de contraste y, gracias a su estructura arborea de inclusión, podemos obtener información muy valiosa con respecto a los objetos presentes en la imagen.

Definición 2.3. Decimos que un conjunto Y es **conexo** si y sólo si $(\forall p, q \in Y)(\exists a_1 ... a_i ... a_n \in Y)(pAa_1 \wedge ... \wedge a_iAa_{i+1} \wedge ... \wedge a_nAq)\ \forall i,\ 0 < i < n$, donde A es una relación de adyacencia definida en D.

El conjunto Y también se denomina **región**.

Vale resaltar que esta defnición de conjunto conexo es demasiado restrictiva para muchas de las propiedades que veremos a continuación, sin embargo no presenta limitaciones en los casos prácticos de aplicación del presente trabajo.

Definición 2.4. Decimos que $Y \subseteq D$ es una **componente conexa** de X si $Y \subseteq X$ es un conjunto conexo y maximal. Es decir, no existe $Z \mid Y \subsetneq Z \subsetneq X$ y Z es conexo.

Dado un punto x en $\mathcal{X}^\lambda u$ llamamos $\mathrm{cc}(\mathcal{X}^\lambda u, x)$ a la componente conexa del conjunto de nivel $\mathcal{X}u$ que contiene a x. Por convención, si $x \notin \mathcal{X}u, \mathrm{cc}(\mathcal{X}u, x) = \emptyset$

Por lo tanto, de la proposición 2.1 se deriva la fórmula de reconstrucción para las componentes conexas de los conjuntos de nivel de una imagen:

$$\forall x,\ u(x) = \inf\{\lambda \mid \mathrm{cc}(\mathcal{X}^\lambda u, x) \neq \emptyset\}$$
$$\forall x,\ u(x) = \sup\{\lambda \mid \mathrm{cc}(\mathcal{X}_\lambda u, x) \neq \emptyset\}$$

Los mismo ocurre con la invariabilidad a cambios de contraste y la carecterística de formar una representación completa de una imagen como se ve en la fórmulas de reconstrucción anteriores.

La propiedad de inclusión de los conjuntos de nivel se traduce naturalmente a sus componentes conexas, de forma que se pueden representar todas las componentes conexas de los conjuntos de nivel superior o inferior en una estructura de árbol ordenada por inclusión.

En la figura 2.4, vemos las componentes conexas de los conjuntos de nivel de una imagen y su árbol de inclusión de componentes asociado, tanto para los conjuntos de nivel superior como para los de nivel inferior.

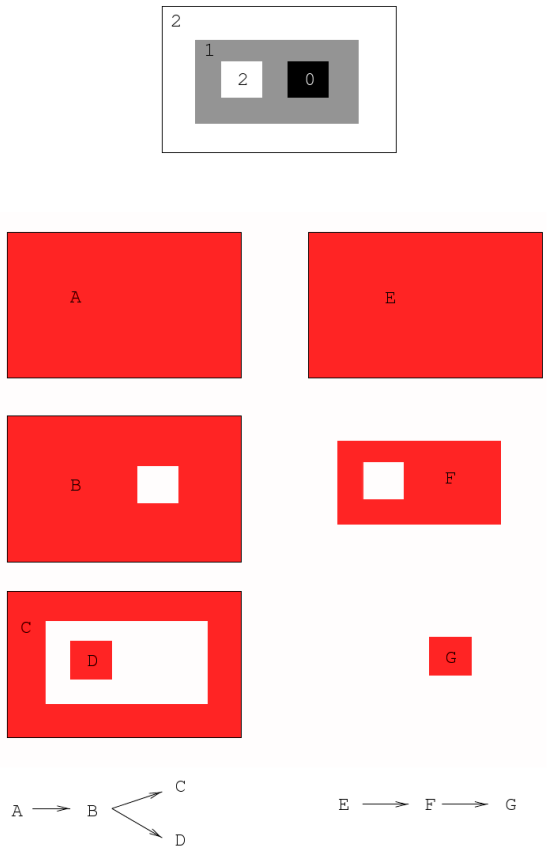


Figura 2.4: Arriba: imagen sintética con 3 niveles de gris: 0,1 y 2. En la columna izquierda se muestran las componentes conexas de lo conjuntos de nivel superior, $\mathcal{X}_0, \mathcal{X}_1, \mathcal{X}_2$, y su árbol correspondiente. En la columna derecha se ven las componentes conexas de los conjuntos de nivel inferior, $\mathcal{X}^2, \mathcal{X}^1, \mathcal{X}^0$, y su árbol correspondiente. Figuras extraídas del trabajo de Monasse [Mon00].

El árbol de componentes superior se obtiene de las componentes conexas de los conjuntos de nivel superior de una imagen, de la misma manera se obtiene el árbol de componentes inferior. Es sabido que los conjuntos de nivel superior favorecen a los objetos claros de la imagen, mientras que, los de nivel inferior, a los oscuros. Es por esto que será de nuestro interés obtener tanto el árbol de componentes superior como el inferior.

En este trabajo la implementación elegida para la construcción del árbol de componentes de una imagen fue la desarrollada por Najman y Couprie [NC06] en donde se propone un algoritmo con complejidad de orden lineal amortizada.

2.3.1. Extracción del árbol de componentes

La idea del algoritmo de extracción del árbol de componentes se basa en un proceso emergente que utiliza nociones topográficas de las imágenes [NC06].

Primero, vamos a ver la imagen como una superficie, en donde a cada punto del plano de la imagen le corresponde un valor de altitud definido por el nivel de gris (ver figura 2.5). Luego, supondremos que la superficie está totalmente cubierta por agua, y que gradualmente el agua se va escurriendo hasta desaparecer; durante este proceso, irán apareciendo islas que irán creciendo y uniéndose unas con otras a medida que el agua que las separaba va desapareciendo. Estas islas son las que consideraremos como componentes de la imagen.

Nótese que el proceso de escurrir el agua deja al descubierto las componentes correspondientes a los conjuntos de nivel superior. De la misma forma, para obtener las componentes de los conjuntos de nivel inferior se puede pensar en un proceso, en donde a la superficie se la va inundando con agua gradualmente. Las lagunas que se van formando corresponden a las componentes inferiores de la imagen.

En el proceso emergente, las islas que irán apareciendo estarán representadas por árboles parciales que iremos guardando en la colección de conjuntos disjuntos. Al finalizar el algoritmo la colección contendrá un único árbol, y éste será justamente el árbol de componentes buscado.

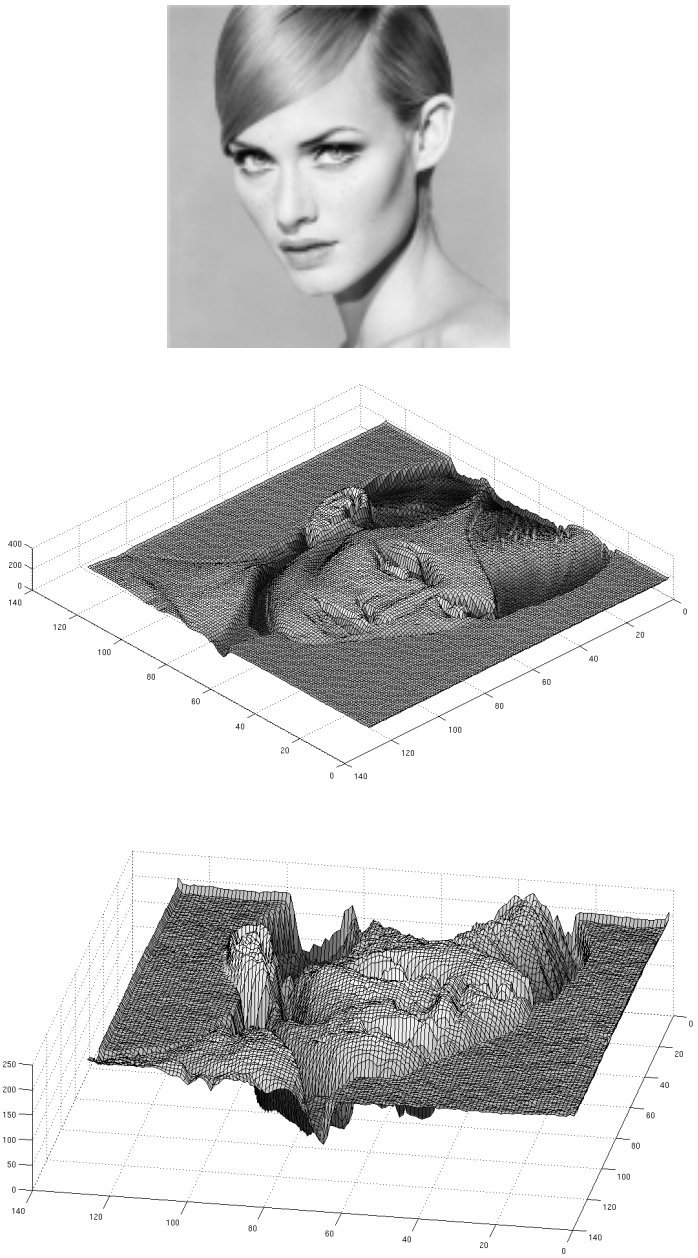


Figura 2.5: Representación de la imagen original (arriba) en donde cada píxel es un punto en el plano y su valor de gris es la altitud.

Estructuras de datos y variables utilizadas

Para poder alcanzar el orden de complejidad lineal, se utiliza una estructura de conjuntos disjuntos basada en el procedimiento *union-find* de Tarjan [TVL84].

Antes de continuar, haremos un breve repaso sobre la estructura de conjuntos disjuntos. Esta estructura está explicada con mayor detalle en el apéndice B.

El problema de los conjuntos disjuntos consiste en mantener una colección $\mathcal{Q}$ de subconjuntos disjuntos de un conjunto D. Es decir, dado $\mathcal{Q} = \{Q_1, \ldots, Q_n\}$ se cumple que $Q_i \cap Q_j = \emptyset$ con $1 \leq i \neq j \leq n$.

Para poder identificarlos, cada conjunto Q_i de $\mathcal{Q}$ está representado por un único elemento de Q_i denominado *elemento canónico*. Por lo tanto, dos elementos que poseen el mismo representante están en el mismo conjunto.

Las operaciones principales que realizaremos sobre los conjuntos disjuntos son las siguientes:

- FIND(x): devuelve el elemento canónico del conjunto de $\mathcal{Q}$ que contiene a x. Útil para saber si dos elementos están en el mismo conjunto.
- LINK(x, y): se eliminan X e Y y se agrega a $\mathcal{Q}$ el nuevo conjunto $Z = X \cup Y$ donde X e Y son los conjuntos representados por los elementos canónicos x e y respectivamente. Como requisito se debe cumplir que $x \neq y$
- MAKESET(x): agrega el conjunto $\{x\}$ a $\mathcal{Q}$ asumiendo que $\{x\}$ no está ya en $\mathcal{Q}$.

Los algoritmos presentados a continuación utilizan varias estructuras auxiliares que asumen que los píxeles de la imagen de entrada u están numerados de 0 a $N-1$ de arriba a abajo y de izquierda a derecha, donde N es la cantidad de píxeles de u. Las estructuras utilizadas son las siguientes:

- Node: cada nodo del árbol de componentes con atributos *level* y *area*, accesibles como *n.level* y *n.area* respectivamente, para un nodo n del árbol[2], que representa el nivel de gris y el área de una componente.
- *nodes*: un arreglo de nodos indexado por píxel. Permite saber para un píxel a que nodo pertenece.
- Q_{node}: una colección de conjuntos disjuntos, en donde los nodos pertenecientes a la misma componente y con mismo nivel son agrupados juntos. Denotaremos con $\text{Find}_{\text{node}}$, $\text{Link}_{\text{node}}$ y $\text{MakeSet}_{\text{node}}$ a las operaciones sobre Q_{node}.
- Q_{tree}: una colección de conjuntos disjuntos, en donde se irán alojando los árboles parciales extraídos de la imagen los cuales conformarán el árbol de componentes. Denotaremos con $\text{Find}_{\text{tree}}$, $\text{Link}_{\text{tree}}$ y $\text{MakeSet}_{\text{tree}}$ a las operaciones sobre Q_{tree}.
- *roots*: un arreglo que contiene las raíces de los árboles parciales en Q_{tree}, indexado por el elemento canónico de cada conjunto disjunto de Q_{tree}. Esta asociación es necesaria porque el elemento canónico de cada árbol en Q_{tree} no es necesariamente su raíz.
- *root*: nodo raíz del árbol de inclusión de formas.

[2]Para consultar la estructura de implementación del árbol y sus algoritmos, dirigirse al apéndice A.

- *smallest*: arreglo de nodo indexado por píxel, donde para cada píxel el arreglo contiene el nodo en el árbol correspondiente a la componente conexa más chica que lo contiene.
- $u : D \to V$: la imagen de entrada, donde para cada píxel p se accede a su nivel de gris como $u(p)$.

Descripción del algoritmo

El pseudocódigo del procedimiento que construye el árbol de componentes se detalla en el algoritmo 2.1. Éste calcula solamente el árbol de componentes superior, pero podemos notar que es posible calcular con el mismo algoritmo sin modificaciones, el árbol de componentes inferior de una imagen, si simplemente invertimos el valor de gris de sus píxeles: $\forall x,\ u(x) = u_{max} - u(x)$ donde u_{max} es el máximo nivel de gris de la imagen u. Una alternativa que no requiere modificar la imagen es cambiar el algoritmo, para que recorra los píxeles en orden creciente de nivel de gris.

La relación de vecindad entre píxeles (función Γ) adoptada en el trabajo de Najman y Couprie [NC06] es de 4-adyacencia (ver figura 2.6).

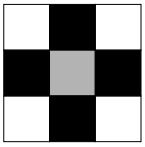


Figura 2.6: En negro se puede ver los píxeles vecinos al píxel central en una relación de 4-adyacencia.

El algoritmo puede ser dividido en tres etapas: inicialización, ciclo principal y finalización. Ésta división contribuye a una mejor comprensión del algoritmo y además nos brinda las herramientas para comprender las propiedades invariantes del ciclo principal, que nos servirán para entender la correctitud y un mejor análisis del orden de complejidad.

Incialización

En el algoritmo 2.1, en las líneas de 2 a 5, se inicializan los conjuntos disjuntos Q_{node} y Q_{tree} mediante el procedimiento MakeSet (algoritmo B.3), como n conjuntos disjuntos cada uno conteniendo un único elemento: él mismo. Asimismo, se inicializa el arreglo *roots* y el arreglo *nodes*.

El ordenamiento de los píxeles por su nivel de gris (línea 1) puede realizarse eficientemene utilizando el algoritmo BucketSort, descripto en el libro de Cormen et. al. [CLRS03].

Ciclo principal

En el ciclo principal (líneas 6 a 18) del algoritmo, se recorren todos los píxeles en orden decreciente de nivel de gris. Para cada píxel p se examinan sus vecinos ya visitados: sea q un vecino ya visitado de p se cumple que $u(q) \geq u(p)$. Esto es debido a que recorremos en orden decreciente de nivel de gris.

Tanto para p como para q se obtiene por medio de Q_{tree} el árbol parcial al cual pertenecen, y gracias a *lowestNode* y Q_{node} se encuentra la componente que los agrupa.

Luego, si p y q están en la misma componente (línea 12) no se debe hacer nada. Si no se compara su nivel de gris (línea 13)):

- Si p y q tienen el mismo nivel de gris deben pertenecer a la misma componente, por lo tanto se deben unir sus componentes como una sola. La función MergeNodes (algoritmo **??**) es la encargada de esta tarea, y además de actualizar la colección de nodos Q_{node} y el árbol de componentes.

- Si no, ocurre que el nivel de gris de p es estrictamente inferior al de q, de este modo la componente que contiene a q debe ser hija de la de p, ya que el árbol de componentes que estamos construyendo es de tipo superior, y por lo tanto las componentes de mayor nivel de gris estáran incluidas en las de menor nivel de gris.

Finalmente, se deben unir los árboles parciales que contienen a p y q (línea 17), ya que en este paso p y q están en el mismo árbol.

Finalización

Al terminar la iteración del ciclo principal, se habrán comparado todos los píxeles con sus vecinos y agrupado en el mismo árbol, que será el único presente en la colección de conjuntos disjuntos Q_{tree}, por lo que resta buscar la raíz del árbol de componentes para cualquier píxel (línea 19). Por último, se establece para cada píxel de la imagen cuál es la componente (o nodo) más chica del árbol que la contiene (líneas 20 y 21); esto resulta de especial interés para las aplicaciones de procesamiento del árbol de componentes, que analizamo en este trabajo.

Para una explicación más detallada y un ejemplo de ejecución del algoritmo consultar el trabajo de Najman y Couprie [NC06].

Complejidad

A continuación denominaremos con n a la cantidad de píxeles de la imagen de entrada u.

Se puede ver fácilmente que tanto la inicialización como la finalización tiene un orden de complejidad de $O(n)$, ya que se iteran n veces y se ejecutan operaciones de tiempo constante; y los píxeles pueden ser ordenados en tiempo lineal para imágenes con valores enteros chicos mediante BucketSort [CLRS03].

Luego, en el ciclo principal, se puede ver que para cada pixel se visitan únicamente sus vecinos no procesados, obteniendo un total de $\frac{k}{2} \times n$ iteraciones del ciclo exterior más el ciclo interior, para una k-vecindad (k es constante y generalmente se utiliza 4-adyacencia u 8-adyacencia); llamemos m a éste número. Por lo tanto, tendremos un total de operaciones del orden de m y, como se

ve en el apéndice B, se puede probar que el orden de complejidad pertenece a $O(m \times \alpha(m,n))$, donde α es una función que crece muy lentamente y se asimila a una función constante para cualquier caso práctico. De esta manera, siguiendo un análisis amortizado de la complejidad, el ciclo principal tiene un orden cuasi lineal.

Concluyendo, vimos que en las tres etapas del algorimo: incialización, ciclo principal y finalización, el orden de complejidad es de $O(n)$ y, por lo tanto, el orden total de complejidad del algoritmo 2.1 es de $O(n)$.

```
ÁrbolDeComponentes(Image u)
 1  Ordenar los puntos de u en orden decreciente de nivel de gris
 2  for p ∈ D
 3       MakeSet_tree(p), MakeSet_node(p)
 4       nodes[p] ← MakeNode(u(p))
 5       roots[p] ← p
 6  for p ∈ D en orden decreciente de nivel de gris
 7       curTree ← Find_tree(p)
 8       curNode ← Find_node (roots[curTree])
 9       for q ∈ Γ(p) / u(q) ≥ u(p)
10            adjTree ← Find_tree(q)
11            adjNode ← Find_node(roots[adjTree])
12            if curNode = adjNode
13                 if nodes[curNode].level = nodes[adjNode].level
14                      curNode ← MergeNodes(adjNode, curNode)
15                 else
16                      nodes.addChild(nodes[adjNode])
17                 curTree ← Link_tree(adjTree, curTree)
18                 roots[curTree] ← curNode
19  root ← roots[Find_tree(Find_node (0))]
20  for p ∈ D
21       smallest(p) ← nodes[Find_node(p)]
22  return < root, smallest >
```

Algoritmo 2.1: Construcción del árbol de componentes.

```
MergeNodes(Int node1, Int node2)
 1  tmpNode ← Link_node(node1, node2)
 2  if tmpNode = node2
 3       for n ∈ nodes[node1].children
 4            nodes[node2].AddChild(n)
 5       tmpNode2 ← node1
 6  else
 7       for n ∈ nodes[node2].children
 8            nodes[node1].AddChild(n)
 9       tmpNode2 ← node2
10  nodes[tmpNode].area ← nodes[tmpNode].area + nodes[tmpNode2].area
11  return tmpNode
```

Algoritmo 2.2: Procedimiento que une dos componentes del árbol. Algoritmo auxiliar de 2.1.

```
MakeNode(Int level)
1  Crear un nuevo nodo n con una lista vacía de hijos
2  Definir n.level ← level, n.area ← 1
```

Algoritmo 2.3: Procedimiento auxiliar del algoritmo 2.1

2.4. Árbol de formas

El árbol de inclusión de formas es una representación completa y no redundante de una imagen. Es completa porque la información de las formas es suficiente para recontruir la imagen; y carece de redundancia debido a que, si se remueve alguna forma del árbol, es imposible reconstruirla; o, con la reconstrucción, se obtiene una imagen distinta.

El árbol está compuesto por formas que representan los "objetos" que se espera encontrar en la imagen, donde una forma es hija de otra en el árbol si está incluida en ella.

Las formas conservan las mismas propiedades que las componentes conexas de los conjuntos de nivel: localidad e invariabilidad a cambios de contraste.

Las estructuras de los árboles de componentes inferior y superior de una misma imagen pueden ser muy distintas, y además no existe un vínculo entre ambas (ver figura 2.4). Sin embargo, podemos relacionarlas mediante la noción de *hueco*. La información de los huecos no está presente en el árbol de componentes, no obstante lo huecos pueden ser vistos como componentes conexas dentro de otra componente (ver figura 2.7). De aquí surge lo que se denomina una *forma* y puede ser vista como una componente conexa de un conjunto de nivel que tiene los huecos rellenados. La relación "ser un hueco de" entre componentes conexas de los conjuntos de nivel se traduce a "está contenida en" cuando trabajamos con formas.

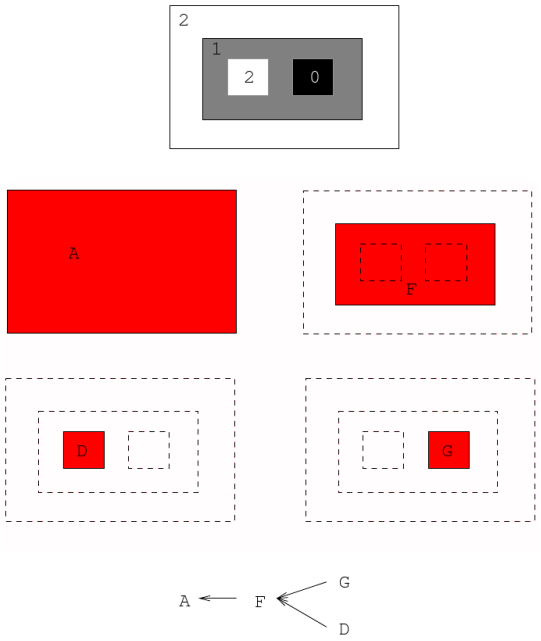


Figura 2.7: Arriba de todo se puede ver la imagen sintética usada. A la componente F se le rellenan lo huecos (componente D). Las componentes D y G están incluídos en F como se ve en el árbol. La imagen se representa con único árbol de inclusión. En este ejemplo, en contraposición al de la figura 2.4, hay un único elemento en el árbol por "objeto" percibido en la imagen.

Para poder realizar una definición formal de hueco y forma, es necesario contar con un operador de saturación. El operador de saturación es el operador que transforma las componentes conexas de los conjuntos de nivel en formas, es decir, "rellena" los huecos de las componentes conexas de los conjuntos de nivel. Las propiedades y definiciones que incluyen este operador de saturación, requieren un conocimiento extensivo de topología y morfología matemática, que escapan de los alcances de este trabajo[3].

Definición 2.5. Diremos que *sat* es un operador de saturación en D si $sat : \mathcal{P}(D) \to \mathcal{P}(D)$, donde $\mathcal{P}(D)$ es el conjunto de partes de D, y se cumplen:

1. $\forall A \subset D, D \setminus sat(A)$ es $\emptyset$ o una componente conexa de $D \setminus A$
2. *sat* es un operador creciente con respecto a la inclusión de subconjuntos de D
3. $\forall A \subset D, sat(D \setminus sat(A)) = D$ ó $\emptyset$
4. $sat \circ sat = sat$

Definición 2.6. Sea $sat : \mathcal{P}(D) \to \mathcal{P}(D)$ un operador de saturación, y sea $A \subseteq D$. Llamamos **huecos** a las componentes conexas incluidas en $sat(A)$ y denominamos **exterior** de A al conjunto $D \setminus A$.

[3]En el trabajo doctoral de Monasse [Mon00], se puede encontrar el soporte matemático para corrobar las nociones y propiedades dadas en esta sección.

Luego, una forma es una componente conexa de un conjunto de nivel "saturada". De la definición anterior, se llega a la siguiente identidad:

$$sat(A) = A \cup \bigcup_{H \in \mathcal{H}_A} H$$

donde las uniones son disjuntas y $\mathcal{H}_A$ denomina el conjunto de huecos de A.

El último punto de la definición 2.5 establece que el operador de saturación es idempotente: $sat \circ sat = sat$, es decir, una vez que los huecos de una componente fueron rellenados, no quedan huecos por rellenar.

Se puede probar que $\mathcal{H}_A$ son las componentes conexas acotadas del complemento de A, y que la única componente conexa no acotada de A es su exterior.

Definición 2.7. Dada un imagen u definimos una **forma** de tipo inferior como $sat(\mathrm{cc}(\mathcal{X}^\lambda u, x))$, y una forma de tipo superior como $sat(\mathrm{cc}(\mathcal{X}_\lambda u, x))$.

Otra propiedad importante de los huecos es que si $A \subset D$ es acotado y H es un hueco de A, entonces $sat(H) = H$, es decir, un hueco es también una forma.

Definición 2.8. El **nivel** de una forma S es el valor λ tal que $S = sat(\mathrm{cc}(\mathcal{X}^\lambda u, x))$, si es de tipo inferior, o $S = sat(\mathrm{cc}(\mathcal{X}_\lambda u, x))$ si es de tipo superior.

Como veremos en la siguiente sección, el algoritmo principal de extracción del árbol de formas, extrae la rama que contiene un punto p, sólo si p pertenece a una región extrema. Aunque esto no es una limitación, precisamos definir lo que llamaremos región extrema.

Definición 2.9. El **borde (exterior)** o **vecindad** de una región se define como

$$\partial Q = \{q \in D \setminus Q : \exists p \in Q : qAp\}$$

En otras palabras, podemos decir que el borde de una región Q es el conjunto de píxeles que no están en Q y son adyacentes al menos a un píxel de Q.

Definición 2.10. Una **región extrema** $Q \subset D$ es una región tal que

$$\forall p \in Q, q \in \partial Q : u(p) > u(q) \quad \text{ó} \quad \forall p \in Q, q \in \partial Q : u(p) < u(q)$$

En esta sección, veremos un resumen de los capítulos 2 y 3 de la tesis doctoral de Monasse [Mon00], con el objetivo de presentar el árbol de formas y el algoritmo para su obtención, que serán usados inténsamente a lo largo del presente trabajo. También presentaremos una nueva variante de la implementación del árbol de formas sugerida por Monasse.

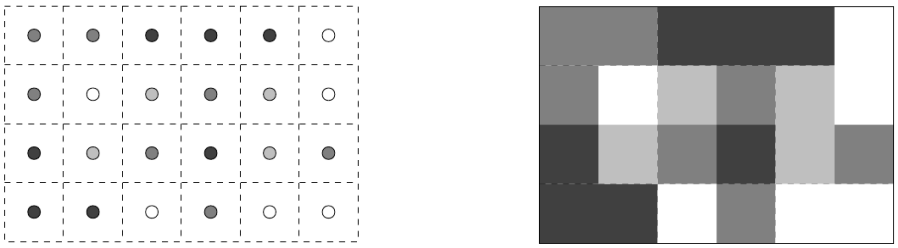
Modelo de interpolación

El trabajo hecho para el árbol de inclusión de formas está basado en imágenes de dominio continuo. El árbol de inclusión de formas tiene su equivalente para imágenes digitales, pero antes es necesario encontrar las definiciones de forma y hueco en este contexto.

Este es un problema frecuente en el campo de procesamiento de imágenes, ya que las imágenes de dominio continuo son estudiadas teóricamente, pero en

la práctica contamos con imágenes de dominio discreto. Por lo tanto, tenemos dos opciones: una es interpretar la imagen discreta como si tuviera dominio continuo, y la otra es redefinir las nociones usadas en el dominio continuo. Esta última decisión es muy difícil, ya que implica reformular todas las propiedades y definiciones en el dominio discreto, y no hay garantía de que sigan valiendo. Por lo tanto, para poder utilizar los resultados obtenidos por el dominio continuo, se optó por utilizar la primera alternativa. Ésta implica un paso de interpolación, que no consiste en trabajar con una imagen continua teniendo una imagen discreta, sino en definir una imagen con dominio continuo basada en los valores de los píxeles de la imagen discreta. La interpolación adoptada es bastante cruda y suficiente para asegurar que la imagen es una función semi-continua superiormente, y deducible de la imagen original de manera invariable a contraste. Vale aclarar que no se utiliza cualquier imagen semi-continua superiormente sino aquella que proviene de la interpolación de una imagen discreta.

El dominio de definición de una imagen digital puede verse como una grilla rectangular equiespaciada donde los valores de gris están concentrados en los centros de los píxeles. El resultado de esta interpolación es fácil de establecer: un punto de la imagen que tenga coordenadas no enteras, es asociado al valor del punto más cercano de la imagen digital (ver figura 2.8).



(a) (b)

Figura 2.8: Imagen discreta en (a) y su imagen continua asociada en (b).

La definición de conexión entre los puntos en la imagen está encapsulada en la función Γ del algoritmo 2.6 que veremos a contiuación, con el objetivo de presentar un algoritmo idependiente del modelo de interpolación usado. Esto nos permitirá utilizar (prácticamente) sin modificaciones el algoritmo de extracción del árbol de formas para imágenes interpoladas bilinealmente (ver sección 2.7).

En el trabajo de Monasse [Mon00] se considera dos tipos de adyacencia distinta: para las conjuntos de nivel inferior se utiliza 4-adyacencia y para los de nivel superior 8-adyacencia, con el fin de evitar los problemas mencionados en la siguiente sección.

Conexión entre puntos

El tipo de conexión entre puntos de una imagen está dado por el tipo de adyacencia que se defina, los más comunes son 4-adyacencia y 8-adyacencia cuando pensamos en una imagen digital como una grilla.

En una grilla rectangular, un pixel se dice que está 4 u 8 conectado cuando tiene las mismas propiedades que uno de sus 4 u 8 vecinos más cercanos,

respectivamente (ver figura 2.9).

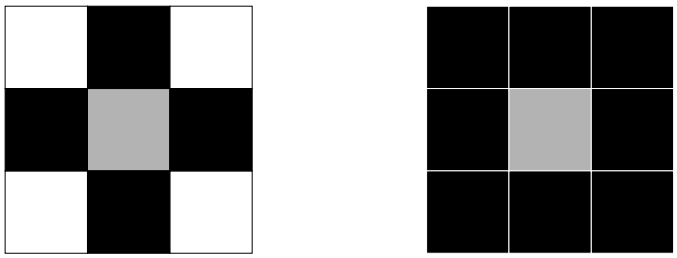


Figura 2.9: Distintos tipos de adyacencia. En negro se representan los píxeles vecinos al píxel en gris. Izquierda: 4-adyacencia. Derecha: 8-adyacencia

Si usamos únicamente 4-adyacencia no podemos obtener un árbol de inclusión de formas, ya que, como se ve en la figura 2.10, hay dos formas que se intersectan y ninguna de las dos está incluida en la otra. En cambio, si utilizamos únicamente la 8-adyacencia, sí podemos obtener un árbol de inclusión de formas. Sin embargo, existen dificultades asociadas con estas definición de adyacencia, ya que se obtiene una definición anti intiuitiva de lo que es un hueco. Como se ve en la figura 2.11, utilizando 4-conectividad, los segmentos 1,2,3 y 4 serían clasificados como disjuntos, aunque sean percibidos como un anillo conectado. Bajo 8-conectividad, estos segmentos están conectados, pero el hueco interno (por ejemplo el píxel A) está también 8-conectado con el exterior (por ejemplo con el píxel B).

Si utilizamos 4-adyacencia para los objetos y 8-adyacencia para el fondo es posible calcular localmente los huecos que se producen en una región, como veremos en la siguiente sección. En cambio, si utilizamos únicamente 8-adyacencia esto no es posible, y se deben computar con un procedimiento con orden de complejidad lineal, que puede producir un cuello de botella en el desempeño del método de extracción del árbol de formas.

En el trabajo de Monasse [Mon00], este problema es resuelto al considerar los dos tipos de adyacencia. Para las componente conexas de los conjuntos de nivel inferior se utiliza 4-adyacencia para los objetos y 8-adyacencia para el fondo, y al revés para las componentes conexas de los conjuntos de nivel superior.

Otra alternativa es usar grillas triangulares o hexagonales [4], donde se puede definir 3 ó 6-adyacencia. Igualmente, hay otras dificultades prácticas que surjen de trabajar con grillas no rectangulares.

[4]Existe un trabajo del autor Yuqing Song [Son07], en donde se estudia el árbol de líneas de nivel para imágenes definidas en grillas hexagonales con relación de 6-adyacencia entre los píxeles.

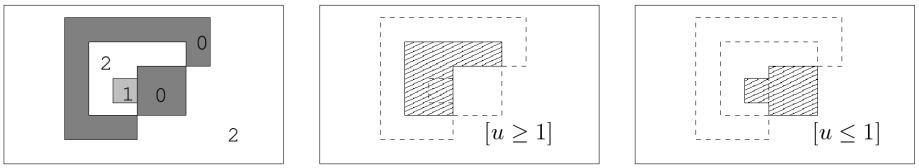


Figura 2.10: Si consideramos únicamente 4-adyacencia no podemos construir un árbol de inclusión de formas. Izquierda: imagen original. Centro: componente conexa del conjunto de nivel superior con nivel 1. Derecha: componente conexa del conjunto de nivel inferior con nivel 1. Ninguna de las dos componentes tiene un hueco, por lo tanto son formas, pero ninguna de ellas está incluida en la otra.

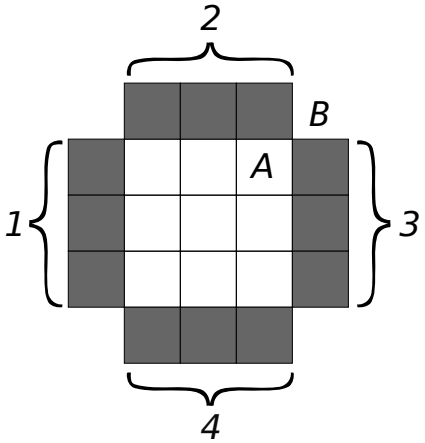


Figura 2.11: Si consideramos únicamente 8-adyacencia, en la figura el píxel A está conectado con el exterior, y por lo tanto no hay hueco.

Cálculo de huecos

El hecho de usar 8-adyacencia los conjuntos de nivel superior y 4-adyacencia los conjuntos de nivel inferior, nos permite calcular localmente la variación de la cantidad de huecos. Esta variación se calcula para todo punto que se agrega a la región. La idea es que, mediante la observación de la configuración de los píxeles de la región en una vecindad de 3×3 alrededor del píxel que se está agregando a la región, poder decidir cuántos nuevos huecos se formarán y cuántos de perderán.

Para poder calcular ésto, contamos con una serie de patrones de configuraciones 3×3 como se ve en la figura 2.12. Cada patrón corresponde a una configuración posible de píxeles pertenecientes (indicados con color negro) o no (indicados con color blanco) a la región. Los píxeles que no importan que estén o no en la región se muestran en color gris. El píxel que se agrega está ubicado en el centro del patrón, y en la figura 2.12, podemos ver en esa ubicación qué cantidad de huecos se agregan o restan a los ya calculados hasta el momento. En la figura 2.12, se presentan las configuraciones para los patrones de un conjunto

con 4-adyacencia. Además de los patrones de la figura 2.12, se deben contemplar los patrones para un conjunto con 8-adyacencia, y más aún, se deben tener en cuenta los patrones que se generan de las rotaciones de éstos, y los que son simétricos horizontalmente o verticalmente.

En el algoritmo 2.5 (ExtractBranch), utilizamos la función #Holes, que calcula la cantidad de huecos de una región mediante los patrones de 4 y 8-adyacencia, dependiendo el caso.

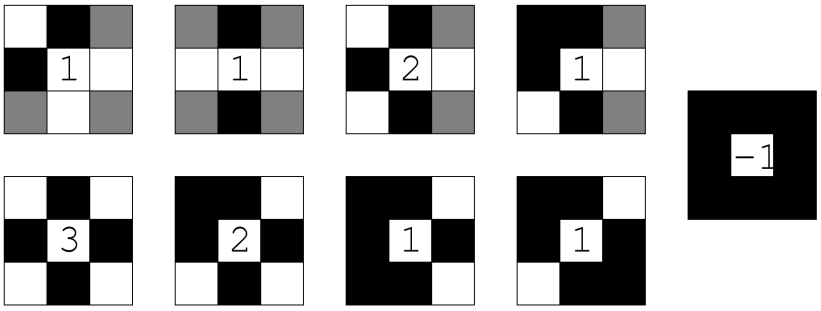


Figura 2.12: Patrones de las configuraciones posibles de los píxeles de un conjunto con 4-adyacencia al agregar un punto centrado en el patrón. En negro se representan los puntos que están dentro del conjunto, en blanco lo que no pertenecen y en gris los que no importan si están o no. El número en el centro del patrón indica la cantidad de huecos que varía.

2.4.1. Extracción del árbol de formas

El algoritmo para la extracción del árbol de formas de una imagen se llama FLST *Fast Level Set Transform*; fue propuesto originalmente por Monasse y Guichard [MG00] y luego Monasse desarrolló una versión más eficiente [Mon00] que es la que le da el nombre al método. En esta sección, se detalla una variante de la implementación del FLST.

La idea del algoritmo extracción del árbol de inclusión de formas se basa en la técnica de crecimiento de regiones. La misma consiste en agregar puntos a una región inicial vacía mientras sea extrema, no posea huecos y no incluya a todos los puntos de la imagen. De esta manera, las regiones crecidas con este criterio irán constituyendo las formas del árbol. Las formas se extraen de la imagen encadenadas en una rama, y se agregan a una colección de árboles parciales en donde se van guardando las formas extraídas hasta el momento ordenadas por inclusión.

Al finalizar la extracción de las formas, la colección de árboles parciales contendrá un único elemento, que será el árbol de formas buscado.

En esta subsección, detallaremos el algoritmo de extracción del árbol de formas de una imagen. La implementación aquí presentada es alternativa a la expuesta por Monasse [Mon00]; aunque no presenta diferencias radicales, proponemos estructuras de datos alternativas a las sugeridas en el mencionado trabajo. También, decidimos detallar algunos procedimientos que no fueron especificados en el trabajo de Monasse; como por ejemplo, el proceso de armado del árbol a partir de las ramas extraídas.

Estructuras de datos y variables utilizadas

Los algoritmos presentados a continuación utilizan varias estructuras auxiliares que asumen que los píxeles de la imagen de entrada u están numerados de 0 a $N-1$ de arriba a abajo y de izquierda a derecha, donde N es la cantidad de píxeles de u.

Las estructuras de datos y variables utilizadas globalmente por todos los algoritmos se presentan a continuación.

- Node: cada nodo del árbol de formas con atributos *level*, *area* y *type*, accesibles como *n.level*, *n.area* y *n.type* respectivamente, para un nodo n del árbol[5].
- $u : D \to V$: imagen de entrada, donde para cada píxel p se accede a su nivel de gris como $u(p)$. Esta imagen es modificada, por lo tanto se debe realizar una copia si se desea conservar la original.
- *tags*: arreglo de booleanos indexado por píxel, donde para cada píxel indica si fue examinado o no.
- $\mathcal{Q}$: un conjunto disjunto en donde se van guardando los árboles parciales, con operación Find, link y MakeSet.
- *roots*: arreglo de nodo que contiene las raíces de los árboles parciales en $\mathcal{Q}$, indexado por el elemento canónico de cada conjunto disjunto de $\mathcal{Q}$. Esta asociación es necesaria porque el elemento canónico de cada árbol en $\mathcal{Q}$ no es necesariamente su raíz.
- *smallest*: arreglo de nodo indexado por píxel, donde para cada píxel contiene el nodo en el árbol correspondiente a la forma más chica que lo contiene.
- `frame`: el conjunto de puntos que forma el borde la imagen.

Es importante nombrar dos estructuras más que poseen significativa relevancia, y que son utilizadas de forma auxiliar para poder implementar la técnica de crecimiento de regiones eficientemente.

- Region R: arreglo que representa el conjunto de puntos de la rama que se está construyendo.
- Neighborhood $\mathcal{N}$: un conjunto que representa todos los vecinos de la región que se está creciendo, donde $\mathcal{N}_{min}$ y $\mathcal{N}_{max}$ representan el mínimo y máximo elemento de $\mathcal{N}$, respectivamente.

Descripción del algoritmo

En el algoritmo 2.4 se describe el pseudocódigo del proceso que construye el árbol de formas de una imagen.

Como hicimos con el algoritmo de extracción del árbol de componentes, dividiremos el proceso en tres etapas: inicialización, ciclo principal y finalización. Debido a la dificultad que presenta describir el ciclo principal, también lo dividiremos en etapas con el único fin de realizar una descripción más organizada.

[5]Para consultar la estructura de implementación del árbol y sus algoritmos, dirigirse al apéndice A.

Inicialización

En el ciclo de la línea 1 del algoritmo 2.4 se definen todos los puntos de la imagen como no marcados (*tags* en FALSE). Luego se define como nula la raíz del árbol que contiene a cada punto de la imagen, y se prepara la colección de conjuntos disjuntos que contendrá los árboles parciales (procedimiento MakeSet).

Notar que no es necesario inicializar el arreglo *smallest*, ya que durante la ejecución del algoritmo se extraen todas las formas de la imagen, y por lo tanto, no queda ningun píxel sin asociar a una forma.

Ciclo principal

El ciclo principal del algoritmo recorre todos los píxeles no marcados de la imagen que sean extremos locales (línea 5); y para estos píxeles se llama al procedimiento ExtractBranch (ver algoritmo 2.5).

Vamos a definir entonces el predicado LocalExtremum(p), que verifica si el punto p es un extremo local.

Definición 2.11. Un punto $p \in D$ se define como **extremo local** si cumple:

$$\forall q \in \Gamma(p), u(q) \geq u(p) \ \vee \ \forall q \in \Gamma(p), u(q) \leq u(p)$$

y además $\exists r \in \Gamma(p), u(r) \neq u(p)$.

Vale destacar que, si se consideran distintos tipos de adyacencia para los conjuntos de nivel superior e inferior, el extremo local debe respetar este hecho.

La idea del algoritmo ExtractBranch es extraer las formas de la imagen ni bien sean detectadas, removerlas y luego agregarlas al árbol resultante. Para remover una forma de una imagen se modifica el valor de gris en la imagen de todos los puntos de la forma, y se asigna como nuevo valor el menor nivel de gris de sus vecinos –si la forma es inferior– o el mayor si la forma es superior.

La idea de buscar los extremos locales está basada en que se puede probar que una componente conexa de un conjunto nivel inferior contiene una región mínima, donde una región mínima está formada únicamente por mínimos locales. Equivalentemente, una componente conexa de un conjunto de nivel superior contiene una región máxima, y se puede probar, que está formada únicamente por máximos locales[6].

Los píxeles que son marcados no necesitan volver a ser examinados aunque sean extremos locales, ya que corresponden a ramas que ya fueron extraídas, o formas que poseen un hueco y por ende no tienen asociada una forma terminal.

Extracción de las ramas Para lograr un mejor entendimiento del algoritmo 2.5 (ExtractBranch), veremos que, se puede probar que toda forma contiene una región extrema sin huecos (la cual es también una forma). Por lo tanto, se pueden caracterizar las hojas del árbol de formas como regiones extremas sin huecos, también denominadas formas terminales. En esta propiedad se basa fuertemente el algoritmo de construcción del árbol de formas, el cual mediante

[6]Una región mínima o máxima también puede verse como una componente conexa de nivel λ, en la que sus vecinos tienen todos un nivel mayor estricto o menor estricto a λ.

un procedimiento *bottom-up* denominado ExtractBranch (ver algoritmo 2.5), extrae la ramas del árbol de formas empezando por las hojas y, después sube por las ramas que luego formarán parte del árbol de formas resultante.

La extracción de una rama puede verse como una sucesión de extracción y remoción de formas, en la que cada una, a su vez, se convierte en forma terminal antes de ser removida. Para remover una rama entera basta con remover su forma más grande.

En la figura 2.13, vemos un ejemplo en donde se remueve una forma de una imagen. Como la forma que se remueve es una componente conexa de un conjunto de nivel superior, para eliminarla definimos su nivel de gris como el máximo de sus vecinos.

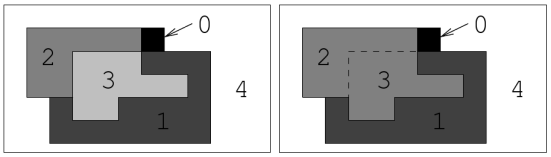


Figura 2.13: Izquierda: imagen antes de remover la forma correspondiente al nivel 3. Derecha: resultado de remover la forma indicada.

En cada ejecución de ExtractBranch, lo primero que se hace es inicializar las variables que se utilizarán para crear una nueva rama (líneas 1 a 5). Luego, se itera hasta que se alcance alguna de las condiciones de corte (variable *grow* en FALSE); en cada iteración se "crece" la región R con el siguiente nivel de gris g, y si la región corresponde a una forma, se añade al árbol.

Recordar que cada región que se extrae debe ser extrema y no poseer huecos para ser considerada una forma. Luego, en vez de remover la forma y extraer otra región extrema en p, se aumenta la región actual R con los píxeles vecinos del nivel más cercano de gris (líneas 16 y 19 dependiendo del tipo de forma) que es equivalente a remover la forma, y crecer nuevamente la región en p.

El procedimiento que "crece" la región se denomina GrowIsoLevelRegion y está definido en el algoritmo 2.6. El mismo mantiene una estructura de vecinos $\mathcal{N}$ que sirve para obtener los puntos vecinos que están a un mismo nivel de gris g y sobre todo nos permite identificar si la región crecida es extrema, es decir, se cumple $\forall q \in \mathcal{N}, u(q) \leq g$ ó $\forall q \in \mathcal{N}, u(q) \geq g$, por lo tanto basta con verificar que $\mathcal{N}_{min} < g < \mathcal{N}_{max}$ para ver si una región no es extrema, ya que todos lo puntos de R se consideran a nivel g.

Cuando se guarda una nueva forma en el árbol, líneas 11, 15 y 18 del algoritmo 2.5, debemos agregarla a la rama que estamos construyendo mediante ExtractBranch.

Añadir una nueva forma al árbol El algoritmo 2.8, es el encargado de agregar al árbol la nueva forma encontrada, representada por la región R.

La primera instrucción realizada (línea 1) consiste en crear una nueva forma y definirla como padre de la forma previamente extraída (algoritmo 2.7 MakeBranchNode).

Luego, se recorren todos lo puntos de la nueva forma (ciclo en la línea 8). Como los nuevos puntos son agregados en orden en el arreglo R, se puede saber

la posición a partir de la cual se encuentran (líneas 2 a 6).

Para cada punto p de la nueva forma (línea 10) tal que, pertenece a un árbol parcial distinto al que se está construyendo representado por currentTree, debe ser agregado agregado a éste. Luego, tenemos dos casos:

- El punto p pertenece a una forma agregada previamente a otro árbol parcial actTree, entonces la región R incluye a esta forma y por ende el árbol parcial actTree cuya raíz se representa con actNode, debe ser hija del árbol que se está construyendo, cuya raíz está representada por currentNode (línea 13).
- El punto p puede pertenecer a un árbol parcial representado únicamente por él, considerado como un punto aislado (anode en NULL), y por lo tanto debemos definirle su forma más pequeña (arreglo *smallest*) como la nueva forma (línea 15).

Luego de ambos casos los puntos se encuentran en el mismo árbol, y por lo tanto se actualiza la colección de árboles parciales mediante la función Link en la línea 16. La nueva forma pasa a ser la forma actual (línea 7) y ésta pasa a ser la raíz del nuevo árbol (línea 17).

La instrucción de la línea 4 del algoritmo 2.8, debe incluirse debido a un detalle de implementación. Esto ocurre en la primera iteración de ExtractBranch, ya que el primer punto que se agrega es p, el punto en donde se empieza a extraer la rama. Por lo tanto, el mismo estará en el árbol actual y nunca se le definirá su forma más chica.

Condiciones de corte El algoritmo ExtractBranch termina cuando se alcanza alguna de las siguientes condiciones de corte:

1. $\mathcal{N}_{min} < g < \mathcal{N}_{max}$: el punto p no pertenece a una región extrema (línea 8).
2. $\#\text{Holes}(R) > 0$: el punto p pertenece a una región extrema, pero contiene un hueco (línea 8).
3. $R \cap \texttt{frame} \neq \emptyset$ `and` $\#R \geq N/2$: el punto p pertence a una región extrema que toca el borde y tiene tamaño mayor a la mitad del área de la imagen (línea 10), por lo que se encontró la raíz. También se podría chequear que $\texttt{frame} \subseteq R$, para verificar si se encontró la raíz del árbol, pero no puede ocurrir que esta condición se cumpla y la región no posea huecos o tenga un área mayor a $N/2$.

En los casos 1 y 3, termina porque se encontró otra forma que no contiene a P, pero que está contenida en la forma más pequeña que contiene a p, y, por lo tanto, se debe reinicializar el proceso en otra región extrema. Luego, el caso 2 interrumpe el proceso debido a que, los huecos encontrados estarán incluidos en la región R y, por ende, se debe reiniciar el procedimiento para extraer la rama (o subárbol) que contiene a los huecos.

Finalización

Como hemos visto, al finalizar la extracción de las formas, la colección de árboles parciales contendrá un único elemento que será el árbol de formas buscado, por lo tanto para obtener la raíz del árbol basta con obtener el elemento

canónico de cualquier punto de la imagen, y consultar su raíz mediante el arreglo *roots* (línea 7).

Por otro lado, al recorrer la imagen todas las ramas del árbol asociadas a los píxeles visitados son extraídas y entonces todos los píxeles de la imagen tendrán asociado su nodo en el arreglo *smallest*.

Hemos visto que para remover una forma de una imagen se modifica el valor de gris en la imagen de todos los puntos de la forma, asignando como nuevo valor el menor nivel de gris de sus vecinos, si la forma es inferior o el mayor si la forma es superior, por lo tanto, al finalizar el algoritmo, la imagen habrá sido modificada con un valor de gris constante en los valores de todos sus píxeles, correspondiente al valor de gris de la raíz del árbol.

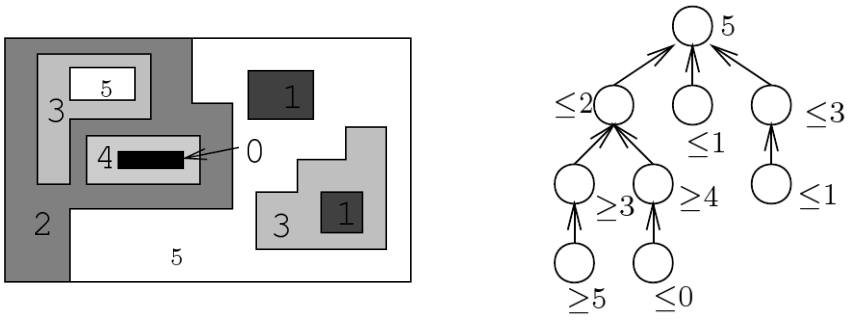
Puede probarse que el algoritmo es correcto ya que:

- Cada rama del árbol se puede extraer al remover sucesivamente formas terminales.
- No es necesario hacer más de un recorrido por toda la imagen, ya que durante la recorrida todas las ramas asociadas a los píxeles visitados son extraídas.
- Luego de una recorrida completa de la imagen no quedan más ramas, por lo que se extrajeron todas las formas de la imagen.

El primer ítem justifica la correctitud del algoritmo ExtractBranch. El segundo punto muestra que es adecuado recorrer la imagen como lo hace el algoritmo 2.4. Por último el tercer punto determina que cuando el algoritmo finaliza efectivamente se extrajeron todas las formas de la imagen.

Para una demostración de estas aseveraciones y un ejemplo de ejecución del algoritmo consultar el trabajo doctoral de Monasse [Mon00].

En la figura 2.14, vemos el árbol de formas asociado a una imagen sintética. La raíz del árbol corresponde a la componente conexa del conjunto de nivel con valor 5, que es el que posee mayor área. Luego, podemos notar tres elementos aislados en la imagen de la figura (a), que serán las ramas del árbol que podemos ver en el árbol de la figura (b).



(a) (b)

Figura 2.14: (a) Imagen original. (b) Árbol de formas asociado.

Orden de complejidad

Encontrar formas consiste en comparar los valores de los píxeles, por lo tanto la medida de tomada para el cálcuo de la complejiad es la cantidad de comparaciones entre píxeles.

Es difícil estimar cuántas veces un píxel va a ser comparado con sus vecinos, ya que un píxel puede ser vecino de muchas formas. Más aún se puede ver experimentalmente que el tiempo de ejecución no depende solamente del tamaño de la imagen, sino también de sus **contenidos** [Mon00], es decir, cantidad de formas y disposición de éstas en la imagen.

Para establecer una cota superior al orden de complejidad, se propone comparar el algoritmo de extracción del árbol de formas con el algoritmo básico de extracción de componentes conexas. Éste consiste en recorrer todos los posibles niveles de gris de la imagen y, para cada uno, binarizar la imagen y extraer en tiempo lineal las componentes conexas. Entonces, como cada píxel tiene que ser comparado con 2 ó 3 vecinos dependiendo del tipo de adyacencia considerada, tenemos una cantidad de operaciones del orden de $2 \times g \times n$ donde g es la cantidad de niveles de gris distintos de la imagen, y n su cantidad de píxeles. Por lo tanto, si la imagen tiene un rango de valores pequeño, es decir $g << n$ –por ejemplo para valores representados con 8 bits– tendremos que $g = 255$ y, entonces, el orden de complejidad, en el peor caso, será del orden de $O(n)$. En cambio, si la imagen es flotante, podría haber tantos valores de gris como puntos en la imagen, ocasionando que el orden de complejidad sea de $O(n^2)$ en el peor caso.

```
ÁrbolDeFormas(Image u)
1  for p ∈ D
2        tags[p] ← FALSE
3        roots[p] ← NULL
4        MakeSet(p)
5  for p ∈ D / ¬tags[p] ∧ LocalExtremum(p)
6        ExtractBranch(p)
7  root ← roots[Find(0)]
8  return < root, smallest >
```

Algoritmo 2.4: Ciclo principal para la contrucción del árbol de formas.

```
ExtractBranch(Point p)
 1  currentTree ← Find(p) ▷ Representante del árbol actual
 2  currentNode ← NULL
 3  R ← ∅, grow ← TRUE
 4  𝒩 ← {p} ▷ Se inicializa la vecindad con un único punto
 5  g ← u(p) ▷ Nivel de gris actual
 6  while grow
 7       GrowIsoLevelRegion(g, R, 𝒩)
 8       if (𝒩_min < g < 𝒩_max) or #Holes(R) > 0
 9            grow ← FALSE
10       elseif R ∩ frame ≠ ∅ and #R ≥ N/2 ▷ R es la raíz
11            AddShape(R, g, currentNode, ROOT)
12            grow ← FALSE
13       else ▷ Agrego una nueva forma al árbol
14            if g ≥ 𝒩_max
15                 AddShape(R, g, currentNode, SUPERIOR)
16                 g ← 𝒩_max
17            else
18                 AddShape(R, g, currentNode, INFERIOR)
19                 g ← 𝒩_min
20  for q ∈ R ▷ Se remueve la rama de la imagen
21       u(q) ← g
22       tags[q] ← TRUE
```

Algoritmo 2.5: Procedimiento que construye el árbol de formas a partir de sus ramas.

```
GrowIsoLevelRegion(Float g, Region R, Neighborhood 𝒩)
1  for q ∈ 𝒩 / u(q) = g
2       R ← R ∪ {q}
3       𝒩 ← 𝒩 \ {q}
4       for r ∈ Γ(q) / r ∉ R
5            𝒩 ← 𝒩 ∪ {r}
```

Algoritmo 2.6: Procedimiento que agrega a la región R todos los puntos vecinos con nivel de gris g

```
MakeBranchNode(Node child, Region R, Float g, Type t)
1  Crear un nuevo nodo n
2  n.area ← #R, n.level ← g, n.type ← t
3  if child ≠ NULL
4       n.addChild(child)
5  return n
```

Algoritmo 2.7: Agrega un nuevo nodo a la rama.

```
AddShape(Region R, Float g, Node currentNode, Type t)
 1  newNode ← MakeBranchNode(currentNode, R, g, t)
 2  if currentNode = NULL ▷ Primera iteración de ExtractBranch
 3       i ← 0
 4       smallest[R[0]] ← newNode
 5  else
 6       i ← currentNode.area
 7  currentNode ← newNode
 8  for p ∈ R[i..n] ▷ Itero por los puntos recién agregados
 9       actTree ← Find(p)
10       if actTree ≠ currentTree ▷ p es un punto aislado o está en un árbol
11            anode ← roots[actTree]
12            if anode ≠ NULL
13                 currentNode.addChild(anode)
14            else
15                 smallest[p] ← currentNode
16            currentTree ← Link(actTree, currentTree) .
17  roots[currentTree] ← currentNode
```

Algoritmo 2.8: Agrega una nueva forma a la rama actual.

2.4.2. Forma Vs Componente

Manejar simultáneamente los conjuntos de nivel inferior y superior de una imagen en dos árboles distintos es un problema. Como ambos no están relacionados, si borro un nodo de alguno de los árboles, ninguno representará la misma imagen. No hay una solución fácil a este problema y, para corregir esta situación, se debe volver a extraer el árbol de la imagen reconstruida.

Además, la información de inclusión sólo está disponible para componentes del mismo tipo; esto es de esperarse, ya que las componentes de distinto tipo no están anidadas y, por lo tanto, no se puede tener una única estructura con las componentes de nivel superior e inferior.

Hemos visto una imagen se puede reconstruir tanto desde los conjuntos de nivel superior como de los conjuntos de nivel inferior; por consiguiente, la información de las dos estructuras al mismo tiempo es redundante y, sin embargo,

no es posible conocer con estas dos estructuras la relación "ser un hueco" en un objeto.

Estos problemas no se encuentran al considerar el árbol de inclusión de formas, en donde la redundancia entre las componentes de nivel inferior y superior es eliminada automáticamente al trabajar con formas, y el problema de un hueco en una componente se ve sencillamente resuelto como una forma que está incluida dentro de otra.

2.5. Almacenamiento de los píxeles de formas y componentes

Es posible guardar el conjunto de píxeles de cada forma o componente del árbol de inclusión de manera eficiente sin repetir información. La idea es aprovechar la estructura de inclusión del árbol, y guardar un único arreglo común a todas las elementos el árbol con todos los píxeles de la imagen, agrupados por el ordenamiento que surge al recorrer en preorden (ver definición A.11) el árbol.

Para esto se enumeran los píxeles en un arreglo A con un orden inducido por el recorrido en preorden de las formas, tal que los píxeles pertenecientes a una misma forma estén en un intervalo de A. Luego, cada forma tiene que almacenar únicamente el índice del arreglo en donde empieza el intervalo en A, ya que la longitud del intervalo estará dada por el área de la forma. En cada intervalo primero se encuentran los píxeles propios de cada forma y luego los píxeles de las formas hijas.

Además del árbol de inclusión, convenientemente se genera una relación entre píxel y el elemento más chico que lo contiene (ya sea componente o forma, dependiendo de la estructura considerada), en un arreglo de nodo indexado por píxel, con el objetivo principal de lograr la reconstrucción de la imagen eficientemente (ver sección 2.6). Sin embargo, tener este arreglo y el conjunto de píxeles perteneciende a cada nodo (como se explicó arriba) es redundate, ya que a partir de uno se puede deducir el otro y viceversa. Además, tener que mantener ambas estructuras puede perjudicar el desempeño de los algortimos de procesamiento sobre el árbol de inclusión, es por esto que se propone un algoritmo de reconstrucción alternativo 2.10.

Se puede demostrar que toda forma tiene al menos un píxel propio, es decir, un píxel que no pertenece a ninguna forma que esté estrictamente incluida (ver [Mon00]). Esto sirve para justificar que todo píxel tiene al menos una forma que lo contiene, y por lo tanto que siempre se puede construir el arreglo que establece la relación entre píxel y la forma más chica que lo contiene. Otra propiedad importante que se deduce es que se puede establecer un mapeo inyectivo entre la formas y el dominio de la imagen, mostrando que no puede haber más formas que píxeles. Lo mismo sucede para las componentes conexas de los conjuntos de nivel.

2.6. Reconstrucción

Hemos vimos que, tanto el árbol de componentes como el árbol de formas, logran una representación completa de una imagen, es decir, es posible recontruir la imagen original desde el árbol de componentes o de formas. Además de la

estructura de árbol que indica la inclusión entre los elementos, para las dos representaciones se obtiene un arreglo (*smallest*) de nodo indexado por píxel, donde cada píxel contiene el nodo en el árbol correspondiente al elemento más chico que lo contiene. Con la ayuda de este arreglo, es posible reconstruir la imagen original mediante un procedimiento lineal en la cantidad de píxeles de la imagen (ver algoritmo 2.9).

```
ReconstrucciónPíxelPorPíxel(Árbol de inclusión T, Arreglo smallest)
1  Sea u una imagen de tamaño dado por T
2  for p ∈ D
3        u(p) ← smallest[p].level
4  return u
```

Algoritmo 2.9: Algoritmo de reconstrucción píxel por píxel que utiliza la relación entre píxel y elemento del árbol más chico que lo contiene.

Existe también otro procedimiento para reconstruir la imagen original que resulta menos eficiente, pero hace uso de la información del conjunto de píxeles perteneciente a cada nodo del árbol, y puede ser útil si se pierde (o no se desea mantener) la relación entre píxel y elemento más chico que lo contiene. Éste se detalla en el algoritmo 2.10, y es esencialmente la aplicación del principio de superposición dado en la proposición 2.1. Tiene un orden de complejidad cuadrático en la cantidad de píxeles de la imagen.

```
ReconstrucciónPorSuperposición(Árbol de inclusión T)
1  Sea u una imagen de tamaño dado por T
2  for n ∈ T / en preorden
3        for p ∈ n.pixels
4              u(p) ← n.level
5  return u
```

Algoritmo 2.10: Algoritmo de reconstrucción que aplica el principio de superposición.

2.7. Árbol de líneas de nivel bilineales

En la sección 2.4, estudiamos el árbol de inclusión de formas de una imagen, el cual utiliza los conjuntos de nivel. En el árbol de formas se guarda la información de inclusión entre las componentes conexas de los conjuntos de nivel que no poseen huecos, también denominadas formas. La información geométrica de una imagen puede ser reducida a los bordes topológicos de las componentes conexas de los conjuntos de nivel, llamada líneas de nivel.

El mapa topográfico de una imagen [CCM99], definido como el conjunto de todas sus líneas de nivel, constituye una representación completa de una imagen. Asimismo, las líneas de nivel heredan las principales propiedades de las

componentes conexas de los conjuntos de nivel: resultan invariables a cambios de contraste y, puede organizarse en una estructura jerárquica de inclusión, llamada árbol de líneas de nivel.

En esta sección, daremos un repaso del trabajo de Lisani et. al. [LMR01], y nos abocaremos al estudio de las líneas de nivel *bilineales*, es decir, las líneas de nivel que provienen de una imagen interpolada bilinealmente; con el objetivo de presentar el árbol de líneas de nivel bilineales.

2.7.1. Líneas de Nivel

Una línea de nivel (o curva de nivel) se puede definir como el borde topológico de las componentes conexas de los conjuntos de nivel.

Definición 2.12. El **interior** de una línea de nivel cerrada es la porción del plano encerrada por la línea de nivel. Si la curva es abierta es necesario "cerrarla" para poder decidir que parte del plano está adentro y cuál afuera. Una curva abierta puede ser cerrada a través del largo del borde de la imagen que posea menor longitud (ver figura 2.15).

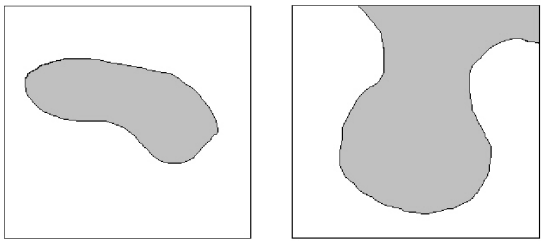


Figura 2.15: Interior de una línea de nivel representado en color gris. Izquierda: la línea de nivel no intersecta el borde de la imagen. Derecha: la línea de nivel toca el borde y por lo tanto es cerrada a lo largo del borde que posee menor longitud.

Una forma también puede definirse como la unión de una línea de nivel con su interior.

Definición 2.13. Llamamos **línea de nivel reducida** L^b, asociada a una línea de nivel L, al contorno de la forma construida a partir de L y su interior.

Luego, el **nivel** y el **tipo** de una línea de nivel reducida L^b, es el nivel de gris y tipo de la forma asociada.

El término *línea de nivel* es más apropiada para L^b que para L. Esto es porque L^b puede ser descripta naturalmente como una curva donde el interior se ubica a la izquierda de la curva, y el exterior a la derecha. Además, L^b y L poseen la misma información desde el punto de vista de la estructura del árbol, ya que que poseen la misma forma asociada (ver figura 2.16).

De aquí en más, nos referiremos a las a líneas de nivel reducidas L^b como líneas de nivel L.

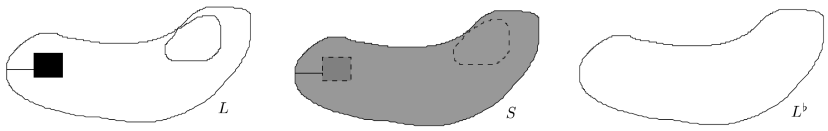


Figura 2.16: Izquierda: línea de nivel L. Centro: forma asociada a la línea de nivel L. Derecha: línea de nivel reducida L^b.

Se puede probar que en cualquier nivel λ el número de líneas de nivel con nivel λ es finito. Por lo tanto, si consideramos un número finito de niveles, tendremos un número finito de líneas de nivel.

Proposición 2.5. Si L es una línea de nivel con nivel λ, el exterior es uniformemente $< \lambda$ ó $> \lambda$.

Esta propiedad implica que el interior de una línea de nivel contiene un extremo local, hecho que se utiliza fuertemente en el algoritmo de extracción del árbol de líneas de nivel bilineales.

Efecto de pixelado

El dominio de definición de una imagen digital puede verse como una grilla rectangular equiespaciada donde los valores de gris están concentrados en los centros de los píxeles (ver figura 2.8a).

Las líneas de nivel de una imagen discreta están restringidas a la grilla de la imagen.

Una línea de nivel se define como el contorno de una componente conexa de un conjunto de nivel. Por lo tanto, las líneas de nivel heredan las propiedades de invariabilidad a cambios de contraste.

Cuando se trabaja con las líneas de nivel de una imagen digital, se observa un efecto de *pixelado*. Ésto se debe a que los puntos que describen las líneas de nivel sólo toman valores discretos.

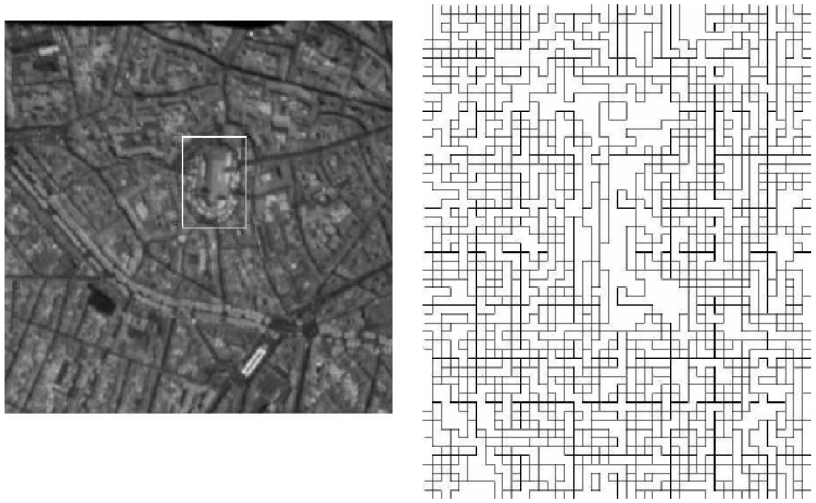
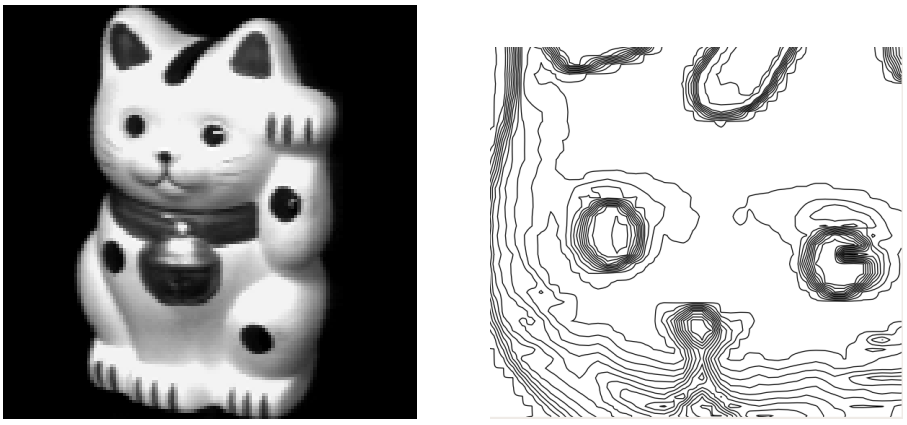


Figura 2.17: A la derecha podemos ver las líneas de nivel de la selección de la imagen de la izquierda.

Para evitar el problema del pixelado se utiliza la interpolación de los valores discretos de la imagen con el fin de obtener una imagen continua. De esta forma, las líneas de nivel de la imagen continua serán más suaves que para la imagen digital. Además es posible obtener mediciones con precisión de subpíxel sobre las líneas de nivel, ya que ellas no están sujetas a pertenecer a la grilla discreta de la imagen digital. En la figura figura 2.18, podemos ver un ejemplo de líneas de nivel extraídas de una imagen interpolada bilinealmente.



(a) (b)

Figura 2.18: En la figura (b) podemos ver un detalle de las líneas de nivel de la figura (a). Las líneas de nivel fueron tomadas en niveles no enteros a un paso de cuantización de 20.

2.7.2. Líneas de nivel e interpolación bilineal

La interpolación bilineal es una extensión de la interpolación lineal para funciones de dos variables definidas en un dominio discreto. La interpolación bilineal busca una aproximación de la función $f(x, y)$ dada un grilla bidimensional de valores de la función f. La forma general de f es:

$$f(x, y) = axy + bx + cy + d$$

De esta forma, para cada conjunto de 4 píxeles adyacentes (que también llamaremos **Qpíxel**, ver figura 2.19) podremos calcular una función bilineal. Calculando la interpolación bilineal de todos los Qpíxeles de la imagen digital, obtendremos una imagen continua bilineal definida de a pedazos.

Los parámetros a, b, c y d son definidos como el valor de gris de los cuatro píxeles (i, j), $(i+1, j)$, $(i, j+1)$ y $(i+1, j+1)$ del Qpíxel a interpolar (ver figura 2.19).

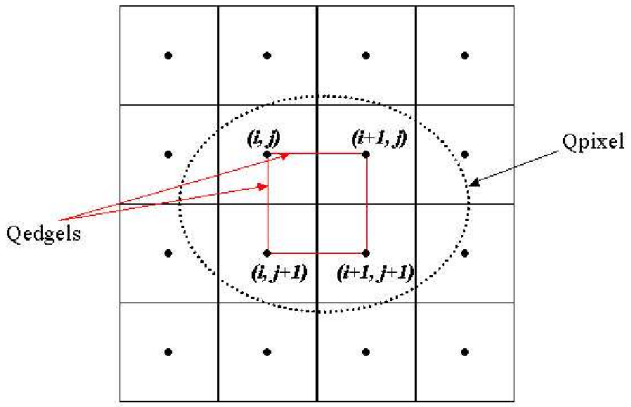


Figura 2.19: Definición de Qpixel y Qeje

La continuidad de los valores de gris en Qpíxeles contiguos está garantizada por las propiedades de la interpolación bilineal; sin embargo, esto mismo no ocurre con el gradiente.

La ecuación de una línea de nivel dentro de un Qpíxel esta dada por:

$$a(x - x_s)(y - y_s) + (\lambda_s - \lambda) = 0$$

Ésta es la ecuación de una hipérbola cuando $x_s = -\frac{c}{a}$, $y_s = -\frac{b}{a}$, y $\lambda_s = d - \frac{bc}{a}$ exceptuando el caso degenerado $a = 0$.

Por lo tanto, las líneas de nivel de una imagen digital interpolada bilinealmente de a pedazos (que llamaremos líneas de nivel bilineales), serán una concatenación de pedazos de hipérbolas. Ésto lo podemos ver en la figura 2.20, en donde se muestra una imagen interpolada bilinealmente y algunas de líneas de nivel.

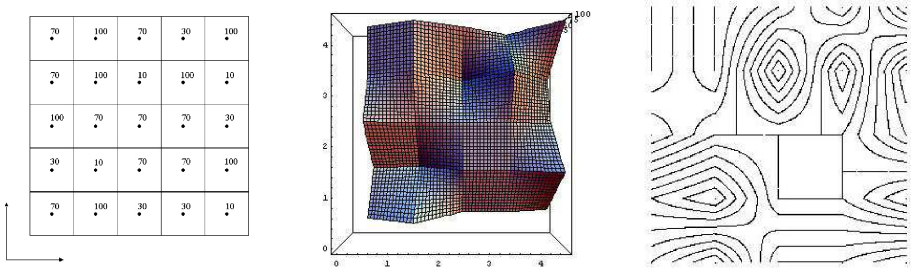


Figura 2.20: Izquierda: imagen original. Centro: vista de la interpolación bilineal de la imagen. Derecha: algunas de sus líneas de nivel.

Cuando $\lambda = \lambda_s$ la línea de nivel se compone de dos líneas perpendiculares que se cruzan en el punto (x_s, y_s) denominado **punto de ensilladura**, y a su nivel de gris λ_s **nivel de ensilladura**.

Codificación del borde

Como una forma es un conjunto conexo, los Qpíxeles adyacentes a una forma puede ser ordenados en una cadena en donde cada eslabón es adyacente al siguiente (ver figura 2.21). Es importante resaltar que la información de la cadena de Qpíxeles adyacentes a una forma S, es equivalente a conocer los centros de los píxeles pertenecientes a S. Además, dada una cádena de Qpíxeles de una formas es posible calcular la línea de nivel de la forma que representa.

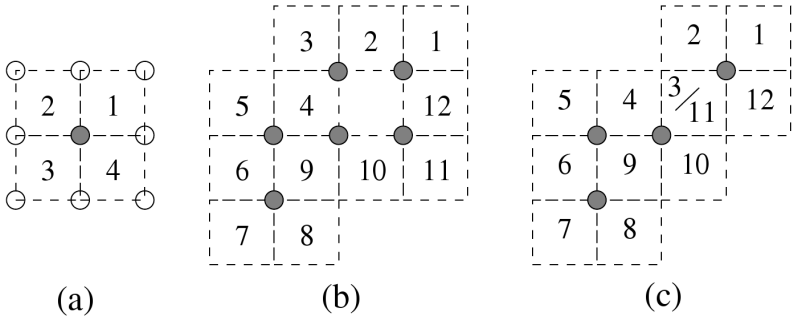


Figura 2.21: Ejemplos de cadenas de Qpíxeles. Los centros de los píxeles se representan en gris, y los números indican el orden de los Qpíxeles en la cadena. En (a) hay un único punto en la forma; en (b) se puede ver una configuración tradicional, y en (c) se puede ver una candena en donde se produjo una 8-conexión entre dos píxeles a través de un punto de ensilladura. Éste Qpíxel se incluye dos veces en la cadena.

A pesar de la interpolación, el efecto de pixelado puede aparecer cuando las líneas de nivel de un Qpíxel forman líneas rectas siguiendo los Qejes. Ésto puede evitarse si para las líneas de nivel no se toman los valores de gris de la imagen

original. Por ejemplo, si los niveles de gris de la imagen están en el conjunto $\{0 \dots 255\}$. una forma de evitarlos es considerar valores no enteros.

2.7.3. Conexión entre puntos

Los vecinos de un centro de un píxel p son sus píxeles 4-adyacentes, más los puntos de ensilladura tal que alguna esquina de su Qpíxel sea p.

Los vecinos de un punto de ensilladura q son los centros de píxeles que son equinas del Qpíxel que contiene a q.

Para poder calcular localmente los huecos podemos utilizar los patrones de 4-conexión entre píxeles de una región, como vimos en la sección 2.4, con el agregado de la conexión por puntos de ensilladura. Por lo tanto, dos centros de píxel están conectados si son 4-adyacentes o, si están opuestos en la diagonal de un Qpíxel y, al menos alguno de los otros dos píxeles de las esquinas o el punto de ensilladura del Qpíxel pertenece al conjunto.

2.7.4. Extracción del árbol de líneas de nivel bilineales

El algoritmo aquí presentado es una variante del algoritmo de extracción de formas (ver sección 2.4), que calcula el árbol fundamental de líneas de nivel bilineales de una imagen, en donde las línea de nivel asociadas a las formas del árbol constituyen el mapa topográfico de la imagen organizado en un árbol de líneas de nivel.

El árbol fundamental de líneas de nivel bilineales puede ser utilizado para calcular el árbol de líneas de nivel bilineales correspondiente a cualquier nivel de cuantización de la imagen.

El algoritmo extrae las líneas de nivel en puntos críticos, es decir, extremos y puntos de ensilladura. Estos niveles corresponden a un cambio en la topología de las líneas de nivel [CK98]. Todas las líneas de nivel que contienen puntos críticos pertenecen al árbol fundamental de líneas de nivel bilineales.

Las variantes introducidas al algoritmo de extracción de formas, se pueden ver en el pseudocódigo 2.11.

Las modificaciones principales son las siguientes:

- La imagen de entrada $u : D \to V$ es interpolada bilinealmente para obtener una función continua, donde $D = D_p \cup D_s$ tal que D_p son las posiciones de los centros de los píxeles de u, y D_s son las posiciones de los puntos de ensilladura. Un punto es tanto un centro de un píxel como un punto de ensilladura.
- Un extremo local se calcula únicamente en base a sus 4-vecinos.
- Solamente se recorren los centros de los píxeles D_p en el ciclo principal del algoritmo.
- La conexión entre puntos y la función de adyacencia Γ, se definen como se indica en la sección 2.7.3.
- Como el algoritmo extrae formas (interiores de las líneas de nivel) de la imagen, mediante un postprocesamiento se deben calcular las líneas de nivel mediante la cadena de Qpíxeles, como se explicó en la sección 2.7.2.

Luego, todas las propiedades, estructuras de implementación, y procedimientos algorítmicos se mantienen intactos, gracias al planteo genérico de éstos.

```
ÁrbolDeLíneasDeNivel
1  for p ∈ D
2        tags[p] ← FALSE
3        roots[p] ← NULL
4        MakeSet(p)
5  for p ∈ D_p / ¬tags[p] ∧ LocalExtremum(p)
6        ExtractBranch(p)
7  root ← roots[Find(0)]
8  return < root, smallest >
```

Algoritmo 2.11: Algoritmo para la contrucción del árbol de líneas de nivel.

Reconstrucción

Como hemos visto anteriormente, el conjunto de líneas de nivel constituye una representación completa de una imagen. Por lo tanto, es posibible recuperar la imagen desde ellas.

El proceso de reconstrucción teniendo el árbol de líneas de nivel es idéntico al explicado en la sección 2.6, con la única excepción de que es necesario obtener el interior de las líneas de nivel - si no se lo tiene ya - antes de iniciar el proceso.

Cuantización del árbol fundamental de líneas de nivel bilineales

El proceso de obtención del árbol fundamental de líneas de nivel bilineales para obtener el árbol de líneas de nivel bilineales para cualqueir cuantización es el siguiente:

- Para cada forma S en el árbol fundamental de líneas de nivel bilineales, se deben buscar los niveles de cuantización entre el nivel de S y el de su padre S'.
- Luego, se crea una forma por cada nivel, la línea de nivel asociada pasa por la misma cadena de Qpíxeles que S.
- Para cada nueva forma, su orden en el árbol de líneas de nivel bilineales resultante es el mismo que el de sus formas "menos grandes" en el árbol fundamental de líneas de nivel bilineales.

El árbol de líneas de nivel bilineales resultante se puede considerar un muestreado del árbol fundamental de líneas de nivel bilineales.

Vale aclarar que el proceso de cuantización puede elminar algunas de las formas presentes en el árbol fundamental de líneas de nivel bilineales, ocasionando que la imagen no pueda ser reconstruida desde sus líneas de nivel.

Capítulo 3

Detección de características

Un sistema de reconocimiento de formas por visión en computadora puede ser dividido en tres partes: detección de características, elaboración de descriptores, y búsqueda de correspondencias.

El proceso de detección de características es una fase muy importante que, en muchos casos, puede determinar el éxito o fracaso de todo el sistema. Una característica puede describirse como una una parte "interesante" de una imagen, que le da cualidad distintiva, es decir, que sirve para distinguirla de sus semejantes.

Es por esto que las características deben cumplir con ciertas propiedades, como por ejemplo, estabilidad, repetibilidad y robustez. La detección de las características deber ser estable a variaciones de las condiciones de la imagen y parámetros del método. La repetibilidad establece que se deben encontrar las mismas características en dos o más imágenes distintas de una misma escena; y la robustez es la propiedad que tienen las características de ser detectadas bajo situaciones de ruido y/o transformaciones de la imagen.

Dividiremos la detección de características en dos etapas: segmentación y selección.

La segmentación es un proceso por el cual se desea obtener una representación de la imagen que nos brinde mayor información que el mapa de bits que la conforma. En el capítulo anterior, vimos tres métodos de representación de una imagen: el árbol de formas, el árbol de líneas de nivel y el árbol de componentes; que serán usados para la etapa de segmentación a lo largo del presente capítulo.

La selección está motivada por la necesidad de condensar la gran cantidad de información que posee la representación de una imagen, para elegir con algún criterio las características más salientes.

En este capítulo, nos abocaremos a la tarea de detección de características en el desarrollo del detector de formas máximamente estables o *maximally stable shapes*, inspirado en las regiones extremas máximamente estables. También estudiaremos el detector de contornos significativos que está muy bien condicionado para la detección de características, ya que obtiene un subconjunto mínimo de formas coincidentes con la percepción visual humana.

3.1. Contornos Significativos

Las líneas de nivel extraídas de una imagen natural suelen ser muchas y redundantes; en especial, si provienen de una imagen interpolada bilinealmente, son todavía más que para una imagen discreta. Esto también ocurre por efectos de la adquisición y/o codificación digital. Contar con una elevada cantidad de líneas de nivel perjudica el desempeño de los algoritmos de procesamientos que realizaremos sobre ellas.

Por lo tanto, nuestro objetivo será reducir el número de líneas de nivel, quedándonos con un subconjunto representativo que posea la menor cantidad de información necesaria para entender la imagen. En el trabajo de Desolneux et. al [DMM01], se presenta un método de selección de líneas significativas (ver figura 3.1), que se basa en el principio de percepción de Helmholtz de la teoría Gestáltica [Kan79]. Una línea de nivel significativa se detecta como una curva contrastada con pocas chances de ser encontrada en una situación de ruido.

(a) (b) (c)

Figura 3.1: (a) Imagen extraída de la base de datos COIL-20 [NNM96]. (b) Líneas de nivel de la imagen en (a) tomadas en 255 niveles de gris distintos. En total hay 1666 líneas de nivel. (c) Líneas de nivel significativas. En total hay 473 líneas de nivel.

Sea L una línea de nivel de la imagen u y $x_0, x_1, \ldots, x_{n-1}$ denotan n puntos de L, con distancia geodésica (dentro de la curva) de dos píxeles, considerados estadísticamente independientes en el modelo *a contrario* de ruido [DMM01]. Si ℓ es la longitud de la curva L, n será igual a $\ell/2$.

Definición 3.1. El **contraste** en un punto x de la imagen u se define como la norma del gradiente en x:

$$c(x) = |\nabla u(x)|$$

Consideraremos que los contrastes en los puntos $c(x_i)$ son variables aleatorias independientes.

Nota. El contraste se puede calcular mediante diferencias finitas en una vencindad de 2×2, para no violar la independencia de los contrastes de los puntos independientes.

Sea $H(\mu)$ la distribución de probabilidad de la norma del gradiente como variable aleatoria de la imagen u.
$H(\mu)$ modela la probabilidad de que un punto dentro de la imagen tenga un contraste mayor a μ. Debemos notar que necesitamos tener previamente calculada la distribución $H(\mu)$; ésta puede ser estimada como el histograma de los contrastes de la imagen u.

La probabilidad de que la curva L tenga un contraste mayor a μ será:

$$P[c(x_0) \geq \mu] \times P[c(x_1) \geq \mu] \times \cdots \times P[c(x_{n-1}) \geq \mu] = H_c(\mu)^n$$

Definición 3.2. El **número de falsas alarmas** de una línea de nivel L se define como:

$$NFA(L) = N_{\ell\ell} \times [H_c(\min_{x \in L} c(x))]^n$$

donde $N_{\ell\ell}$ es la cantidad de líneas de nivel de la imagen u

El NFA mide la significación o relevancia perceptual de una curva y cuánto más chico es, la curva será más significativa ya que su probabilidad de ser generada por ruido será muy chica (principio de Helmholtz).

Definición 3.3. Una línea de nivel L se dice **ε-significativa** si:

$$NFA(L) < \varepsilon$$

Por lo tanto, el NFA de una línea de nivel nos permite establecer una medida sobre su importancia perceptual, y también poder determinar cuantitativamente con qué grado de seguridad una curva fue aceptada o rechazada. De esta forma, podemos realizar una selección en nuestro árbol de líneas de nivel, en donde cada curva será elegida si posee suficiente relevancia, es decir, si su NFA es menor a un ε dado.

Vale destacar que en [DMM01] el objetivo principal del artículo está puesto en detectar los bordes de una imagen. Como explican los autores, los bordes de una imagen coinciden con pedazos de curvas de nivel, por lo tanto si nuestra preocupación es detectar bordes, podremos seleccionar de las líneas de nivel de una imagen las partes o pedazos más significativos, y definir de esta forma un borde ε-significativo de una manera similar a como lo hicimos con las líneas de nivel. Hay que tener en cuenta que cuando dividimos una curva de nivel, sus partes pierden varias propiedades que tienen las curvas de nivel, como la inclusión y la posibilidad de poder reconstruir la imagen original.

3.1.1. Contornos significativos maximales

Una complicación que trae la selección de líneas de nivel ε-significativas es que suelen encontrarse redundantemente, en grupos paralelos formando *bordes gordos* (ver figura 3.1c) ubicados en una misma rama dentro del árbol de líneas de nivel. Esto motiva el desarrolo de las líneas de nivel ε-significativas maximales.

Definición 3.4. En un árbol de líneas de nivel, una **rama monótona** o **sección monótona** es una secuencia de líneas de nivel L_i con $i = 1 \dots k$ tal que:

- L_i es el único hijo de L_{i-1} para $i = 2 \dots k$

- L_i tienen el mismo tipo, es decir, son todas superiores o inferiores para $i = 1 \dots k$
- La secuencia es maximal, es decir, no está incluida propiamente en otra rama monótona.

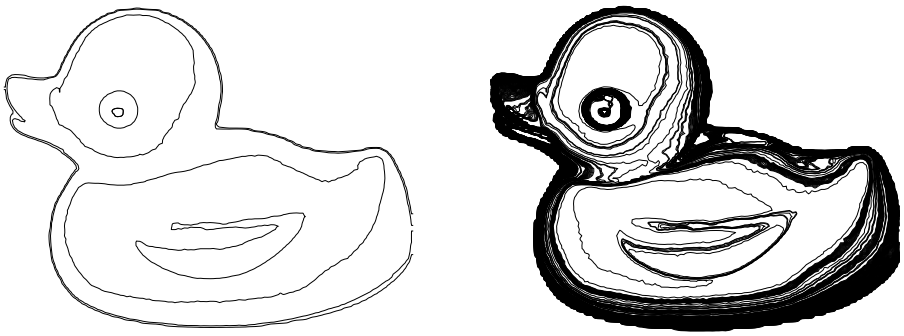
En vez del segundo punto, se puede pedir equivalentemente que las líneas de nivel estén ordenadas todas de forma creciente o decreciente, según su nivel de gris, ya que es una propiedad que se deduce de una rama monótona del árbol de líneas de nivel [LMR01].

Definición 3.5. Las **líneas de nivel ε-significativas maximales** –o también llamadas **líneas de nivel óptimas**– se obtienen seleccionando dentro cada rama monótona las de menor NFA.

Luego, puede probarse [CMS05] la validez de la siguiente propiedad, que nos permite dar una interpretación del NFA.

Proposición 3.1. La esperanza del número de curvas ε-significativas en un conjunto aleatorio E de curvas es inferior a ε.

En la figura 3.2 se puede ver un como se reducen las líneas de nivel significativas de una imagen al imponer la restricción de maximalidad. Los grupos paralelos de líneas de nivel son eliminados conservando la curva más significativa de ellos.



(a) (b)

Figura 3.2: (a) Líneas de nivel significativas máximales. Hay un total de 9 curvas para un valor de $\varepsilon = 1$. (b) Líneas de nivel significativas. Hay un total de 473 curvas. Puede apreciarse cómo las líneas de nivel significativas se agrupan formando bordes gordos.

Vale aclarar que este criterio de maximalidad es un criterio de simplificación de ramas monótonas, como también existen o pueden desarrollarse otros criterios que tengan en cuenta otras propiedades entre las curvas, como distancia [LMR01], regularidad [Mon00], etc. En particular, la elección de sólo una curva por rama, en mucho casos, puede ser muy restrictiva, ya que generalmente las ramas monótonas suelen ser de gran longitud.

3.1.2. Contornos significativos contrastados

Las definición 3.2 tiene algunas desventajas. Por un lado, el uso de un mínimo o alguna otra medición puntual, puede ser muy inestable en presencia de ruido. Por el otro, requiere que la curva *entera* sea improbable que haya sido generada por ruido. Como hemos dicho, los bordes de los objetos coinciden con pedazos de líneas de nivel, por lo tanto, veremos un nuevo modelo que propone modificar la definición del número de falsas alarmas de una curva, para obtener detecciones adicionales de curvas que posean partes bien contrastadas.

Sea L una línea de nivel de la imagen u y $x_0, x_1, \ldots, x_{n-1}$ denotan n puntos de L, con distancia geodésica igual a dos, y sea $c(x_i)$ el contraste en el punto x_i (ver definición 3.1).

Denotamos con M el vector de valores $c(x_i)$ ordenado ascendentemente y con μ_k $(0 \leq k \leq n-1)$ al k-ésimo valor M.

La definición 3.3, dada en el trabajo de Desolneux et. al. [DMM01], consiste en rechazar adecuadamente la hipótesis nula:

$\mathcal{H}_0$: los valores $c(x)$ son i.i.d., extraídos de una imagen de ruido con el mismo histograma de gradientes de la imagen u.

Siguiendo las ideas propuestas en el trabajo de Meinhardt et. al. [MZFC08], para una curva dada, la probabilidad sobre $\mathcal{H}_0$ –que al menos k entre los n valores μ_j mayores a μ– está dada por la cola de la ley binomial $\mathcal{B}(n, k, H_c(\mu))$. La función beta incompleta regularizada I puede verse como una interpolación de la cola de la binomial en el dominio continuo y puede ser computada mucho más rápido que ésta. Entonces, es interesante y más conveniente extender este modelo al caso continuo usando la función beta incompleta regularizada

$$I(H_c(\mu); l_1(k), l_2(k))$$
$$l_1(k) = \frac{l}{2}\frac{n-k}{n} \qquad l_2(k) = 1 + \frac{l}{2}\frac{k}{n}$$

Esto representa la probabilidad sobre $\mathcal{H}_0$ que, para una curva de longitud l, algunas partes con longitud total mayor o igual que $l_1(k)$ tengan un contraste mayor o igual a μ.

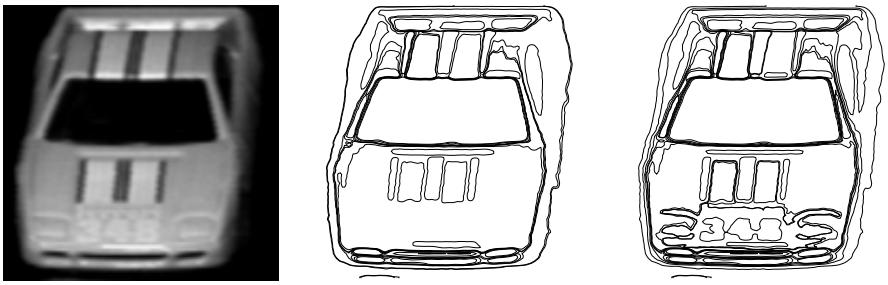
Definición 3.6. Sea $\mathcal{C}$ un conjunto finito de N_{ll} líneas de nivel. Una línea de nivel $C \in \mathcal{C}$ es ε-significativa si

$$\text{NFA}(C) \equiv N_{ll} \cdot K \cdot \min_{0 \leq k < K} I(H_c(\mu_k); l_1(k), l_2(k)) < \varepsilon$$

Este número es llamado el **número de falsas alarmas** (NFA) de C.

Esta definición, analogamente, cumple la proposición 3.1. Luego, el refinamiento propuesto en la definición 3.6 no es más que una relajación de la definición clásica dada por Desolneux et. al. [DMM08], cuyo objetivo es evitar subdetecciones, lo que permite que algunas partes (hasta $k < K$ de lo n puntos) de la curva estén poco contrastadas.

En la figura 3.3, podemos ver un ejemplo, en donde se puede apreciar la detección de nuevas líneas de nivel que no estaban suficientemente bien contrastadas según la definición 3.5.



(a) (b) (c)

Figura 3.3: (a) Imagen original extraída de [NNM96]. (b) Líneas de nivel significativas maximales. (c) Líneas de nivel significativas maximales bien contrastadas de a pedazos. Podemos destacar la aparición de la línea de nivel correspondiente al contorno de los números en la parte inferior del juguete, entre otras.

La elección del valor de K no puede ser hecha directamente, ya que es altamente dependiente de la longitud y el contraste de cada curva. Entonces, el valor de K tiene que ser elegido como una función de la longitud de la curva y del contraste sobre ésta.

Siguiendo la definición 3.6, fijamos el valor de K como:

$$\hat{K}_\varphi \equiv \arg\max_{i<n} \left(\frac{\sum_{j=0}^{i} \mu_j}{\sum_{j=0}^{n-1} \mu_j} < \varphi \right)$$

donde $\varphi \in [0, 1]$ es un nuevo parámetro del algoritmo de detección.

Esta elección de K es, de hecho, adaptativa a la longitud y contraste de cada línea de nivel. También es bastante estable para valores de $\varphi < 0{,}05$. Valores más grandes nos llevan a sobredetecciones y, en general, no aparecen líneas de nivel perceptualmente significantes.

3.2. Regiones extremas máximamente estables

Las regiones extremas máximamente estables –del inglés *maximally stable extremal regions* (MSER) [MCMP02]– son elementos de una imagen (subconjuntos conexos de los píxeles de la imagen) que se caracterizan por ser altamente distinguibles por su contraste con el fondo.

El concepto de las MSER puede ser explicado informalmente mediante un proceso análogo a Watershed [MCMP02], y muy similar al que vimos en la sección 2.3.

Imaginemos todas las posibles imágenes obtenidas de aplicar umbrales sobre los niveles de gris de una imagen u. Nos referiremos a los píxeles como "negros", si están por abajo de un cierto umbral, o como "blancos" si están por arriba. Luego, si armamos un video de las imágenes umbraladas u_t, donde el cuadro (*frame*) t correspondiente al umbral t, y lo reproducimos en orden creciente de los umbrales, primero vermos una imagen en blanco. Subsiguientemente irán apareciendo y creciendo "manchas" negras correspondientes a mínimos locales en intensidad. En un determinado momento, dos regiones correspondientes a

mínimos locales se juntarán formando un mancha más grande. Finalmente, la última imagen del video será completamente negra. Consecuentemente, las manchas negras del video que varíen poco en una secuencia de cuadros del video, concordarán con las regiones máximamente estables.

El conjunto de todas las componentes conexas de todos los cuadros del video es el conjunto de regiones máximas; las regiones mínimas pueden ser obtenidas invertiendo la intensidad de los puntos de la imagen u, y empezar otra vez el proceso.

Procedamos ahora con la definición formal. Hemos visto en la sección 2.1 que una imagen u se puede definir como una función $u : D \to V$. Repasemos la definición 2.10.

Definición 3.7. Una **región extrema** $Q \subset D$ es una región tal que

$$\forall p \in Q, q \in \partial Q : u(p) > u(q) \quad \text{ó} \quad \forall p \in Q, q \in \partial Q : u(p) < u(q)$$

donde ∂Q representa el borde exterior o vecindad de la región Q (ver definición 2.9).

Para el primer caso, la región extrema es de mínima intensidad y, para el segundo, de máxima intensidad.

Definición 3.8 ([MCMP02])**.** Sean $Q_1, \ldots Q_{i-1}, Q_i, \ldots$ una secuencia anidada de regiones extremas, es decir, $Q_i \subset Q_{i+1}$. La región extrema Q_{i^*} es **máximamente estable** sii $q(i) = \dfrac{|Q_{i+\Delta} \setminus Q_{i-\Delta}|}{|Q_i|}$ tiene un mínimo local en i^*. $\Delta \in V$ es un parámetro del método.

La función q se denomina **función de estabilidad**, $|\cdot|$ denota cardinalidad y $\cdot \setminus \cdot$ denota diferencia.

En la definición 3.8, para cada Q_i, el subíndice i representa el nivel de gris de la región extrema, y por ende, la función $q(i)$ mide la variaciación de área con respecto al nivel de gris, para la secuencia de regiones extremas, tomando una ventana de análisis de $[i - \Delta, i + \Delta]$. Nótese que los subíndices de la secuencia no se repiten y las regiones están ordenadas por inclusión en base a éstos.

Si una región extrema es de mínima intensidad y máximamente estable, entonces la denominaremos MSER-, y, en el caso de que sea de máxima intensidad, la denominaremos MSER+.

En las figuras 3.4b y 3.4c podemos ver ejemplos de MSER detectadas para la imagen original 3.4a. Se puede apreciar cómo las MSER+ favorecen la detección de regiones claras sobre fondo oscuro y, las MSER-, a las oscuras sobre fondo claro.

Es habitual que para la creación de descriptores se utilicen las MSER+ y MSER- simultáneamente, considerando la unión como el conjunto de características de la imagen.

(a) (b) (c)

Figura 3.4: Ejemplo de detección de MSER. (a) Imagen original extraida de la Amsterdam Library of Object Images (ALOI). (b) MSER- y (c) MSER+ detectadas de la imagen original.

Las MSER tienen muchas propiedades interesantes. Entre ellas poseen invariabilidad a: cambios afín del contraste, rotación, traslación, y cambios de escala. También poseen la importante cualidad de estabilidad, que es una propiedad inherente a ellas. En la sección 3.2.4 haremos un repaso de las principales propiedades de las MSER.

Debido a sus mencionadas virtudes, son muy utilizadas en la práctica para la extracción de características de imágenes y elaboración de descriptores, como por ejemplo, aplicaciones de apareamiento estereo [MCMP02], búsqueda de objetos y escenas en videos [SZ03], seguimiento de rostros y patentes de automóviles [DB06], entre otras.

3.2.1. Función de Estabilidad

La definición 3.8, que presenta el criterio máxima estabilidad para regiones extremas, tiene varios puntos para comentar. A continuación la analizaremos en detalle.

Sea $S = Q_1, \dots Q_{i-1}, Q_i, \dots$ una secuencia de regiones extremas, definimos el conjunto $\mathcal{I} = \{i : Q_i \in S\}$.

Como hemos dicho en la sección anterior, el subíndice i de Q_i representa el nivel de gris de la región extrema. Luego, la imagen u está definida como $u : D \to V$ entonces $i \in V$ y por ende $\mathcal{I} \subset V$, permitiéndonos establecer este criterio sobre una gran variedad de imágenes (ver sección 2.1).

Luego, no hay una restricción sobre el parámetro Δ, es decir, podemos suponer $\Delta \in \mathbb{R}$. Esta suposición nos trae un inconveniente, si estamos trabajando con imágenes digitales donde $V \in \mathbb{Z}$, o simplemente si $\mathcal{I}$ no es un conjunto compacto, y está dado porque siempre podremos encontrar un valor de Δ tal que $Q_{i+\Delta} \notin S$.

Por ejemplo, sean $\Delta = 0{,}5$ e $\mathcal{I} = \{0, 1, \dots, 255\}$, entonces podemos ver claramente que $Q_{i\pm\Delta} \notin S$. Este ejemplo nos induce a buscar una interpolación de los valores $\mathbb{I}$ para solucionar este problema. El modelo de interpolación sugerido es análogo al visto en 2.4, y supone que la función es semicontinua definida de a tramos. Esta idea está basada en suponer que las variaciones de área se

producen en un único valor de gris, y que éste se mantiene constante hasta la siguiente variación, obteniendo una función de variación de áreas $Q(i)$ definida de a pedazos en el dominio continuo de los números reales.

Ahora bien, si analizamos en detalle la función de estabilidad de la definición 3.8

$$q(i) = \frac{|Q_{i+\Delta} \setminus Q_{i-\Delta}|}{|Q_i|}$$

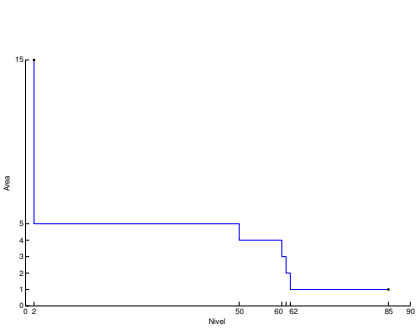
podemos ver que, para dos valores distintos de i, podría tener el mismo valor de $q(i)$, es decir, la región en esos niveles se mantiene constante. Por lo tanto, necesitamos evaluar la función de estabilidad únicamente en los niveles de gris en donde se producen las variaciones de área para una región dada. Esto niveles se pueden representar por el conjunto de índices:

$$\mathcal{I} = \{i : Q_i \in S\} \cup \{i + \Delta : Q_i \in S\} \cup \{i - \Delta : Q_i \in S\}$$

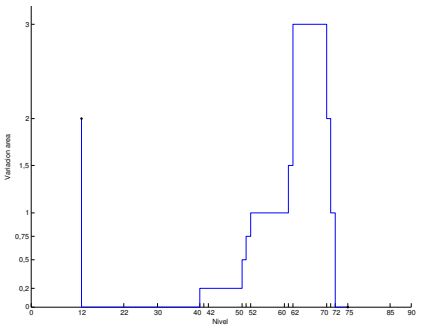
En la figura 3.5, graficamos la función de estabilidad para el crecimiento de una región R. Nótese que el mínimo local entre el rango de valores $(12, 40]$ no es único, y cualquiera de ellos podría ser elegido como el correspondiente a una MSER. En este trabajo se decidió elegir el de mayor nivel de gris, para el ejempo, el mínimo local se produce en el valor 40.

Nivel	2	5	60	61	62	85
Área	15	5	4	3	2	1

(a)



(b)



(c)

Figura 3.5: (a) Valores de crecimiento de una región R. (b) Gráfico de área en función del nivel de gris para (a) con $\Delta = 10$. (c) Gráfico de la función de estabilidad. Variación de área en función del nivel de gris para (a) con $\Delta = 10$.

Máxima variabilidad

Con el objetivo de controlar las variaciones de la función de estabilidad se impone una cota superior a los valores de la función de estabilidad q. La idea es evitar detectar mínimos locales *inestables*, producidos por una variación abrupta en el área para un crecimiento de una forma.

Por lo tanto, se agrega un nuevo parámetro al método denominado *máxima variabilidad* ($maxVar$).

Luego, para un árbol de formas $\mathcal{T}$, y una formas máximamente estables $F_i \in \mathcal{T}$, si $q(i) \geq maxVar$ es elmininada de $\mathcal{T}$.

3.2.2. Extracción de las MSER

A continuación, proponemos un nuevo método de extracción de las MSER de una imagen. Este método está pensado para ser aplicado sobre cualquiera de los árboles de inclusión detallados en el capítulo 2; éstos son: el árbol de componentes, el árbol de formas, y el árbol de líneas de nivel.

Los árboles a los que enfocaremos nuestro estudio serán el árbol de componentes y el árbol de formas, ya que, el árbol de líneas de nivel lo podemos considerar como un árbol de formas, debido a que, como vimos en la sección 2.7, las líneas de nivel pueden ser definidas como los contornos de las formas, y además, para calcular las MSER sólo necesitamos el interior de las líneas de nivel (cantidad de puntos y nivel de la curva).

Una forma posible de obtener las MSER de una imagen es realizando el proceso descripto en el trabajo de Matas et al. [MCMP02]; no obstante podemos notar que el árbol de componentes visto en la sección 2.3, posee toda la información necesaria sobre las regiones para poder encontrar las MSER. En el artículo de Donoser et al. [DB06], se sugiere un método de extracción de las MSER a partir del árbol de componente.

Cabe destacar la principal diferencia de nuestro método con el procedimiento descripto en el trabajo de Donoser et al. [DB06]. Éste último propone calcular el valor de la función de estabilidad para cada nodo a medida que se construye el árbol de componentes, y luego, una vez obtenido el árbol, seleccionar los nodos, que sean mínimos locales para la función de estabilidad. Por el contrario, nuestro procedimiento comienza un con árbol de componentes previamente construido y busca las MSER, verificando la propiedad de estabilidad de las regiones a medida que lo recorre con el objetivo de encontrar mínimos locales. Esto nos brinda la posibilidad de aplicar el criterio de MSER sobre cualquier árbol de inclusión, y obtener las MSER ordenadas por inclusión en una estructura de árbol.

La idea del algoritmo de extracción de las MSER consiste en recorrer todas las ramas del árbol de inclusión, y detectar los nodos del árbol más "estables". Podemos ver a una rama del árbol como el crecimiento de la región más chica de ella. Esto es debido a que cada nodo del árbol de entrada tiene asociado un nivel de gris y un valor de área. Los elementos del árbol están ordenados por inclusión, y a su vez en las ramas del árbol, cada elemento está ordenado por nivel de gris. De esta manera, en cada rama a medida que cambia el nivel de gris, tenemos una variación de área, desde la región más chica correspondiente a la hoja de la rama, hasta la raíz del árbol correspondiente a la región más grande. Esta variación de área para distintos niveles de gris, es justamente la que se controla al buscar los mínimos locales de la función de estabilidad.

Vale destacar que este procediento es genérico, y permite calcular las MSER+ y MSER- para un árbol de componente superio o inferior respectivamente, y si el árbol de entrada es un árbol de formas podremos calcular las formas máximamente estables, como veremos en la sección 3.3.

Estructuras de datos y variables utilizadas

Las estructuras utilizadas son las siguientes:

- $\mathcal{T}$: árbol de inclusión que contiene formas, componentes o líneas de nivel. Cada elemento del árbol posee atributos *level*, *area* y *mark*, accesibles como *node.level*, *node.area* y *node.mark* respectivamente, si *node* es un nodo del árbol. Donde *level* y *area* representan el nivel de gris y el cardinal de la región del nodo. Los nodos que posean este atributo en TRUE, serán los nodos del árbol correspondientes a las regiones extremas máximamente estables.
- R: arreglo de nodos que representa una rama del árbol $\mathcal{T}$.
- L: matriz de dimensión $5 \times n$ conteniendo los niveles de gris en los que se evaluará la función de estabilidad para la rama R de longitud n. En el algoritmo 3.2 (FillMatrix se detalla su uso y definición.
- N: matriz de dimensión $5 \times n$ de referencias a nodos del árbol $\mathcal{T}$, indexada igual que L. En cada posición N_{ij} se encuentra el nodo de la rama R correspondiente al nivel L_{ij}. En el algoritmo 3.2 (FillMatrix se detalla el uso y definición.
- $\Delta \in \mathbb{R}$: parámetro del método.

Luego, en el algoritmo 3.2 (FillMatrix se detalla el uso y definición de L y de N.

Descripción del algoritmo

En el pseudocódigo del algoritmo 3.1, mostramos el procedimiento que calcula las regiones extremas máximamente estables de un árbol de inclusión de formas, componentes o líneas de nivel.

La idea del algoritmo es recorrer todas las ramas del árbol de entrada, donde cada rama representa el crecimiento de la región correspondiente al nodo hoja. Luego, mediante la función de estabilidad, analizar el crecimiento de la región con el objetivo de buscar mínimos locales. Finalmente, los nodos de la rama del árbol en donde se produzcan éstos mínimos locales, corresponderán a las MSER buscadas.

Inicialización

El primer paso del algoritmo 3.1, como se puede ver en la línea 2, es definir en FALSE todas las marcas de los nodos del árbol de entrada.

Si los nodos del árbol de entrada no poseen este atributo, se puede crear una copia del árbol que agregue este campo para cada nodo. Adicionalmente, si es necesario guardar el valor de la función de estabilidad para cada elemento del árbol, se debe agregar un nuevo campo a cada nodo para tal fin.

Ciclo principal

A continuación detallamos el ciclo principal (línea 3) del algoritmo MserPruner, el cual consiste en recorrer las ramas del árbol de entrada.

Para obtener las ramas del árbol se lo recorre en profundidad (ver definición A.11), a medida que se guarda en el arreglo R las referencias a los nodos hasta llegar a una hoja. Es conveniente que los nodos en R se guarden en orden creciente de nivel de gris. Vale notar que, para el caso del árbol de componentes superior, este es el orden en que se encuentran los nodos desde la raíz hasta la hoja de la rama; para el caso del árbol de componentes inferior es el orden inverso.

La primera instrucción del algoritmo 3.3, llama al procedimiento que llena las matrices auxiliares (FillMatrix algoritmo 3.2). La matriz L es usada para recorrer en orden creciente los valores en donde será evaluada la función de estabilidad, y junto con la matriz N poder acceder eficientemente al nodo del árbol que le corresponde al valor de la función.

FillMatrix El objetivo de este procedimiento es llenar la matrices L y N para ser usadas por el algoritmo ComputeMSER. Vale la pena resaltar, que la forma de iterar de este algoritmo es igual a la de ComputeMSER, y que ambos podrían ser integrados en uno sólo. Sin embargo, se decidió hacer esta separación para favorecen la claridad y comprensión del mismo.

La matriz L se llena con los valores en donde será evaluada la función de estabilidad, como vimos en la sección 3.2.1, estos valores serán los que estén a una distancia de Δ y 2Δ, de cada nivel de la rama R (líneas 2 a 7). Luego, para cada valor de L, definimos en la misma posición de la matriz N, una referencia al nodo del árbol que corresponde para dicho valor.

Para llenar la matriz N, recorremos la matriz L en orden creciente de nivel de gris. Este orden creciente, es llevado a cabo, mediante 5 variables: p, q, r, s, t, una por cada fila de la matriz, para indicar la columna actual, ya que se necesitan recorrer también las filas 1 y 5 (ver figura 3.6).

En cada iteración se calcula el mínimo entre los 5 valores de la matriz dados por $L_{1p}, L_{2q}, L_{3r}, L_{4s}, L_{5t}$, logrando el orden creciente. Para pasar al siguiente valor se incrementa la variable correpondiente al mínimo (línea 16), y se vuelve a repetir el proceso. Se termina de iterar cuando llegamos a R_{max}, que es el último valor usado por la función de estabilidad.

	t	s	r	q	p	
$\cdots L_1$					L_{1p}	$\cdots$
$\cdots L_2$				L_{2q}		$\cdots$
$\cdots L_3$			L_{3r}			$\cdots$
$\cdots L_4$		L_{4s}				$\cdots$
$\cdots L_5$	L_{5t}					$\cdots$

	t	s		q | r	p	
$\cdots L_1$					L_{1p}	$\cdots$
$\cdots L_2$				L_{2q}		$\cdots$
$\cdots L_3$				L_{3r}		$\cdots$
$\cdots L_4$		L_{4s}				$\cdots$
$\cdots L_5$	L_{5t}					$\cdots$

Figura 3.6: Izquierda: configuración de la matriz L. Derecha: configuración resultante de haber elegido como el mínimo al valor en L_{3r}

Luego, mediante la variable a mantenemos la posición actual del arreglo R, y como L recorremos en orden de nivel de gris, basta con verificar sólo una desigualdad de la fórmula 3.2.2 en el algoritmo (ver línea 11) para obtener el valor correcto de N para la posición de L.

Para cada posición N_{ij} se define el nodo a-ésimo de la rama R del árbol de la siguiente manera:

$$N_{ij} = R_a \ / \ R_{a-1}.level < L_{ij} \leq R_a.level$$

excepto para la primera posición de R:

$$N_{ij} = R_1 \ / \ L_{ij} = R_1.level$$

Para el resto de las posiciones el valor es indefinido, ya que no se itera por éstos.

ComputeMSER Una vez inicializadas las matrices L y N (línea 1 del algoritmo 3.3), se procede a recorrer la matriz L en orden creciente, del mismo modo que se explicó para FillMatrix con la excepción de que sólo necesitamos recorrer las tres filas del medio.

Este orden creciente es llevado a cabo, mediante tres índices: q, r, s, una por cada fila de la matriz, en donde será evaluada la función de estabilidad, para indicar la columna actual. En cada iteración se calcula el mínimo entre los tres valores de la matriz dados por L_{2q}, L_{3r}, L_{4s}, y de esta manera se logra el orden creciente. Para pasar al siguiente valor se incrementa la variable correpondiente al mínimo (línea 14), y se vuelve a repetir el proceso (ver figura 3.6). Se termina de iterar cuando llegamos a $R_{max} - \Delta$, que es el último valor en donde se evaluará la función de estabilidad.

En cada iteración, verificamos si es posible evaluar la función de estabilidad (línea 6). Luego, para econtrar los mínimos locales de la función de estabilidad, vamos a usar 3 variables: q_0, q_1, diff. La variable q_0 representará el valor anterior de la función de estabilidad, y la variable q_1 el valor actual que se calcula en la línea 8. Asimismo, es conveniente guardar una referencia a los nodos del árbol correspondientes a q_0 y q_1, llamados n_0 y n_1. De esta manera, encontraremos un mínimo local cuando el valor actual q_1, sea mayor que el anterior q_0, y la función sea decreciente (línea 12). Para poder saber si la función es decreciente o no, mantenemos un nueva variable *diff*, en la que guardaremos la diferencia $q_1 - q_0$ de la iteración anterior (ver línea 13); por ende, conociendo el signo de esta diferencia podemos saber si la función está creciendo (signo positivo) o está decreciendo (signo negativo). Por lo tanto, el mínimo local encontrado en la función de estabilidad está en q_0; y consiguientemente marcamos su nodo asociado n_0 como MSER (línea 12).

Finalización

En el ciclo de la línea 5 del algoritmo 3.1, se eliminan[1] todos los nodos del árbol que no hayan sido marcados como MSER por el procedimiento ComputeMSER, de esta manera, se modifica el árbol de entrada conservando únicamente los nodos que corresponden a regiones extremas máximamente estables. Para elminar un nodo se llama al algoritmo A.1 (RemoveNode), que en tiempo constante lo remueve del árbol.

Orden de Complejidad

El orden de complejidad del algoritmo 3.1 (MserPruner) está medido en función de la cantidad de nodos del árbol de entrada $\mathcal{T}$; llamemos m a éste número.

[1] En el apéndice A, se explicitan los algoritmos sobre árboles y las estructuras de implementación usadas.

Como podemos ver, las etapas de inicialización y finalización del algoritmo tienen orden $O(m)$ ya que, realizan procedientos que recorren todos los nodos del árbol y realizan operaciones $O(1)$.

Aisladamente podemos observar que los procedimientos ComputeMSER (algoritmo 3.3 y FillMatrix (algoritmo 3.2) tienen orden $O(n)$, donde n es la longitud de la rama; esto es debido a que en ambos, el ciclo *while* recorre los elementos de L sin repertirlos, y esta matriz tiene un tamaño total de $5 \times n$.

Luego, si suponemos que cada rama del árbol se obtiene haciendo un recorrido independiente desde la raíz hasta cada hoja, para luego invocar el procedimiento ComputeMSER con dicha rama, la cantidad de operaciones del ciclo principal la podemos acotar por: $n_{max} \times B$, donde n_{max} es la longitud de la rama más larga y B la cantidad de ramas del árbol.

Luego, si suponemos que los árboles no son degenerados en promedio, es decir, no poseen ramas largas en donde los nodos poseen un único hijo, podemos decir que la longitud de la rama más larga coincide con la altura del árbol, y se puede probar que para todo árbol de m nodos, su altura h, puede sera acotada superiormente por $log(m)$ [CLRS03], por lo tanto: $n_{max} \leq log(m)$.

Esta asumisión la podemos realizar, ya que, los árboles de formas o componentes extraídos de imágenes naturales, tienen tendencia a ser completos, y no suelen presentarse en configuraciones degeneradas.

Por otra lado, la cantidad de ramas coincide con la cantidad de hojas en un árbol, y la cantidad de hojas puede ser acotada superiormente por la cantidad de nodos del árbol, por ende: $B \leq m$.

Concluyendo, $n_{max} \times B \leq log(m) \times m$, entonces podemos deducir que la cantidad de operaciones del ciclo principal es del orden de $O(m \times log(m))$ en el caso promedio.

```
MserPruner(Tree 𝒯)
1  for Node n ∈ 𝒯
2      n.mark ← FALSE
3  for Branch R ∈ 𝒯
4      ComputeMSER(R)
5  for Node n ∈ 𝒯 / n.mark = FALSE
6      RemoveNode(n)
```

Algoritmo 3.1: Cálculo de Las MSER a partir del árbol de inclusión de una imagen

```
FillMatrix(Branch R[1..n])
 1  R_min ← min_{R_i.level}(R), R_max ← max(R)
 2  for i = 1 to n
 3       L_1i ← R_i.level − 2Δ
 4       L_2i ← R_i.level − Δ
 5       L_3i ← R_i.level
 6       L_4i ← R_i.level + Δ
 7       L_5i ← R_i.level + 2Δ
 8  p, q, r, s, t, i, j, a ← 1
 9  while L_ij ≤ R_max
10       < i, j > ← arg min(L_1p, L_2q, L_3r, L_4s, L_5t)
11       if L_ij ≤ R_a.level
12            N_ij ← R_a
13       else
14            a ← a + 1
15            N_ij ← R_a
16       for k ∈ {p, q, r, s, t}
17            if k = j
18                 k ← k + 1
19  return < L, N >
```

Algoritmo 3.2: Procedimiento que completa las matrices auxiliares del algoritmo ComputeMSER

```
ComputeMSER(Branch R[1..n])
 1  < L, N > ← FillMatrix(R)
 2  R_min ← min(R), R_max ← max(R)
 3  q, r, s, i, j ← 1, q_0, q_1, diff ← 0
 4  while L_ij ≤ R_max − Δ
 5       < i, j > ← arg min(L_2q, L_3r, L_4s)
 6       if L_ij − R_min ≥ Δ
 7            q_0 ← q_1
 8            q_1 ← (N_{i−1,j}.area − N_{i+1,j}.area)/N_ij.area
 9            nq_0 ← nq_1
10            nq_1 ← N_ij
11            if q_1 > q_0 and diff < 0
12                 nq_0.mark ← TRUE
13            diff ← q_1 − q_0
14       for k ∈ {q, r, s}
15            if k = j
16                 k ← k + 1
```

Algoritmo 3.3: Procedimiento para calcular los mínimos locales de la función de estabilidad en una rama.

Ejemplo de aplicación

A continuación veremos un ejemplo de aplicación del algoritmo 3.1 (MSER).

En la figura 3.7, presentamos el árbol que utilizaremos para este ejemplo.

Vamos a suponer que estamos recorriendo la rama que se ve resaltada en la figura 3.7, y nos enfocaremos a estudiar el comportamiento del algoritmo 3.3 (ComputeMSER) sobre esta rama.

El algoritmo ComputeMSER utiliza dos matrices auxiliares L y N. En la tabla 3.2, se muestra el contenido de estas matrices que fueron llenadas mediante el procedimiento FillMatrix como explicamos anteriormente. En L se pueden ver los niveles de gris en donde necesitaremos evaluar la función de estabilidad para la rama R (ver tabla 3.1), y en N los nodos correspondientes para dichos valores de gris (para mayor claridad en la tabla 3.2 mostramos sólo el área de los nodos).

En la figura 3.9, vemos un gráfico para la rama R en donde se representa para cada nivel de gris el área de los nodos de R. Nótese que el área disminuye a medida que aumenta el nivel de gris, sin embargo, este hecho no es un impedimento para calcular la función de estabilidad.

En la tabla 3.2, vemos resaltado con negrita los valores de las posiciones L_{2q}, L_{3r}, L_{4s} y N_{2q}, N_{3r}, N_{4s}, luego de tres iteraciones del ciclo principal (línea 4) del algoritmo ComputeMSER, y además, $q_0 = 0, q_1 = 2$, $diff = 2$ y n_1 apunta al nodo "(50) 5". Por lo tanto, en la primera instrucción del cuerpo del ciclo (línea 5), el mínimo entre estos tres valores es 40 y su posición $< 2, 2 >$. Luego, como 40 es mayor a 2 (R_{min}), las siguientes cuatro instrucciones consisten en calcular el nuevo valor de la función de estabilidad y guardar el anterior en q_0. El nuevo valor es igual a $q_1 = (N_{1,2} - N_{3,2})/N_{22} = 0$, por lo tanto, no cumple la condición de mínimo local (línea 11). El resto de las variables se actualiza en consecuencia: $q_0 = 2, q_1 = 0$, $diff = -2$ y, n_0 y n_1 ambos apuntan al nodo "(50) 5".

Si hacemos una nueva iteración podremos comprobar que ahora el mínimo pasa a ser 50, y el nuevo valor de la función de estabilidad $q_1 = 0{,}2$. De esta manera, dados los valores de las variables del ciclo anterior, podemos comprobar que la condición de ser mínimo local (línea 11) se cumple para q_0, y por ende, el nodo "(50) 5" es marcado como MSER (línea 12).

En el resto de las iteraciones, se puede comprobar mediante el gráfico de la función de estabilidad para la rama R (ver figura 3.10), que no se encontrarán otros mínimos locales. El árbol resultante, una vez terminado el algoritmo ComputeMSER, lo podemos ver en la figura 3.8.

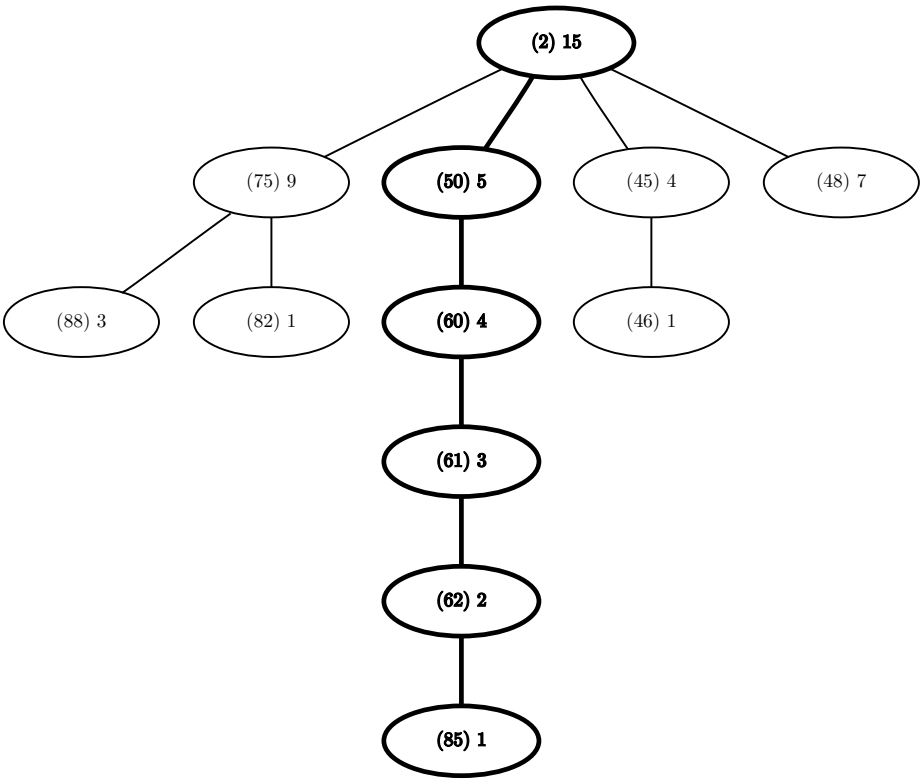


Figura 3.7: Árbol de inclusión donde en cada nodo se muestra el nivel de gris entre paréntesis "()", y el área a continuación.

Nivel	2	5	60	61	62	85
Área	15	5	4	3	2	1

Cuadro 3.1: Rama R del árbol de la figura 3.7.

Matriz L

$R_{-2\Delta}$	-18	30	40	41	42	65
$R_{-\Delta}$	-8	**40**	50	51	52	75
R	2	**50**	60	61	62	85
$R_{+\Delta}$	12	**60**	70	71	72	95
$R_{2\Delta}$	22	70	80	81	82	105

Matriz N (áreas)

		5	5	5	5	1
$R_{-\Delta}$		**5**	5	4	4	1
R	15	**5**	4	3	2	1
$R_{+\Delta}$	5	**4**	1	1	1	
	5	1	1	1	1	

Cuadro 3.2: Matrices L y N para la rama del árbol de la figura 3.7. En la matriz N solo se muestran las áreas de los nodos.

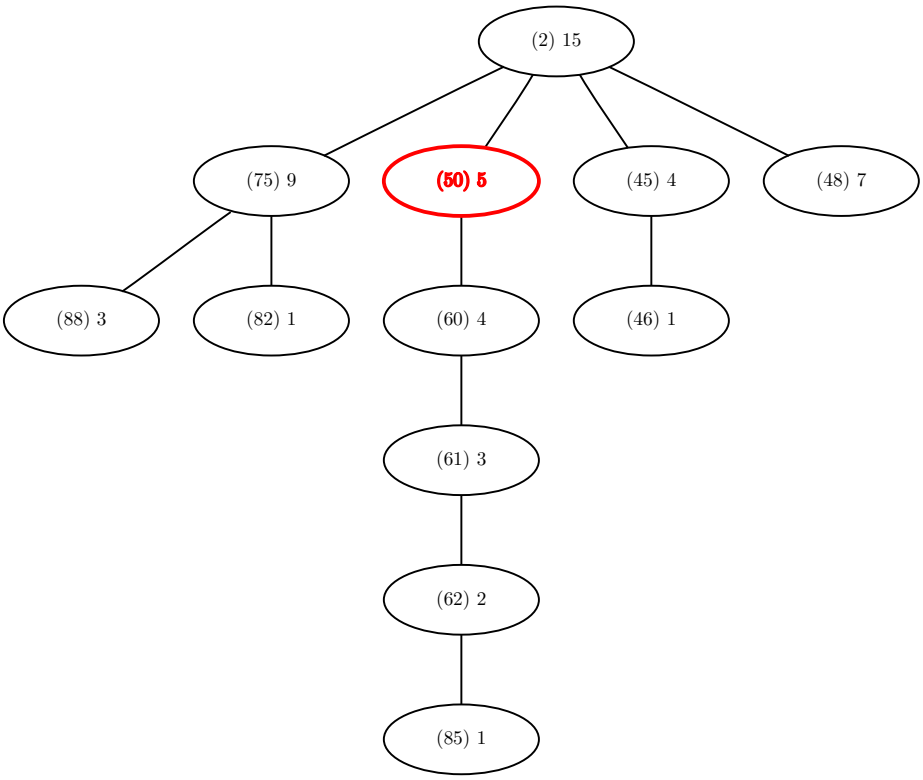


Figura 3.8: Árbol de MSER correspondiente al aplicar el algoritmo 3.1 sobre el árbol de la figura 3.7.

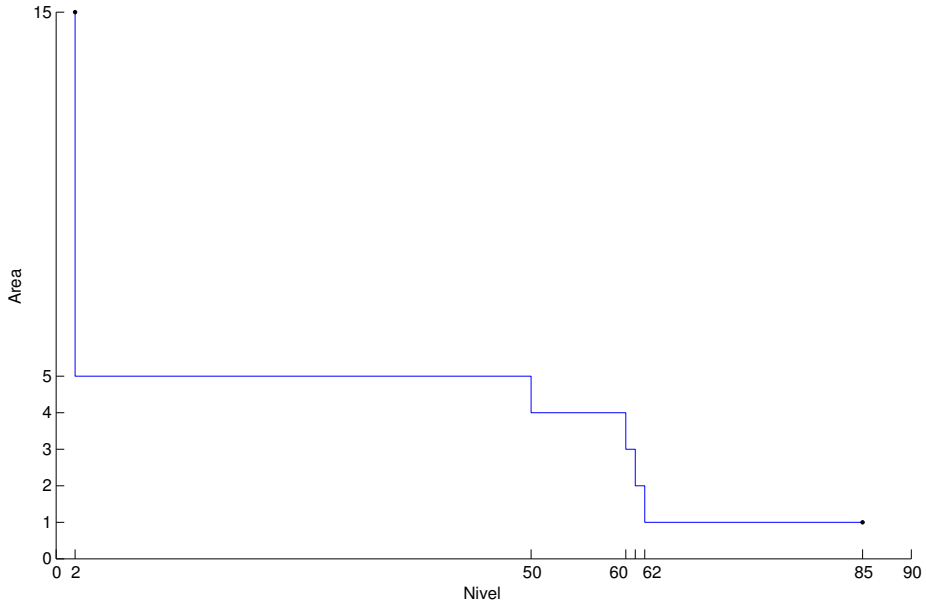


Figura 3.9: Gráfico de área en función del nivel de gris para la rama del árbol de la figura 3.7, utilizando un valor $\Delta = 10$.

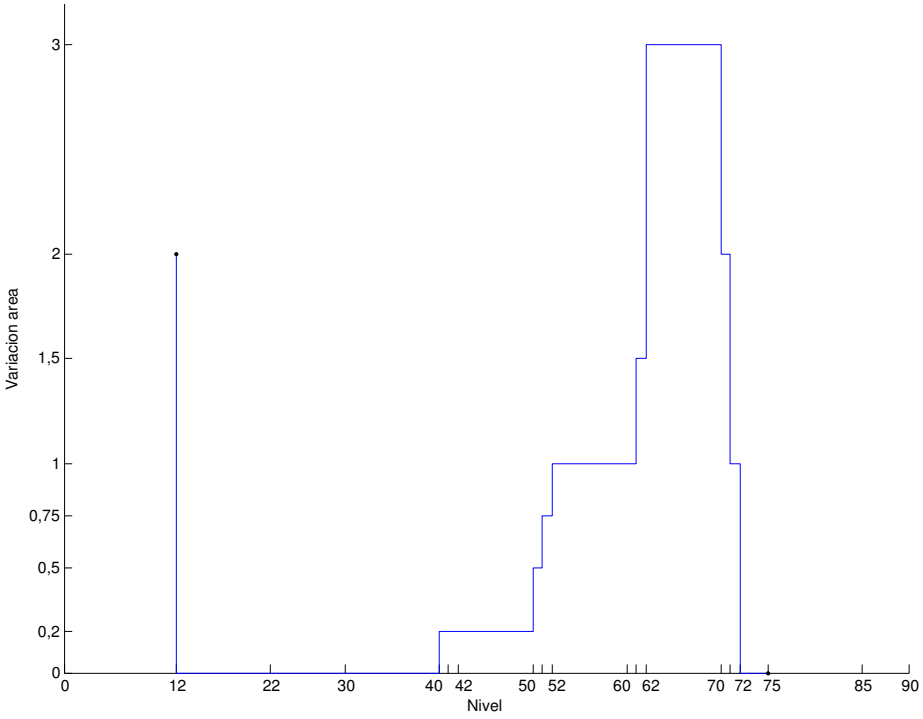


Figura 3.10: Gráfico de la función de estabilidad. Variación de área en función del nivel de gris para la rama del árbol de la figura 3.7, utilizando un valor $\Delta = 10$.

3.2.3. Influencia del parámetro Delta Δ

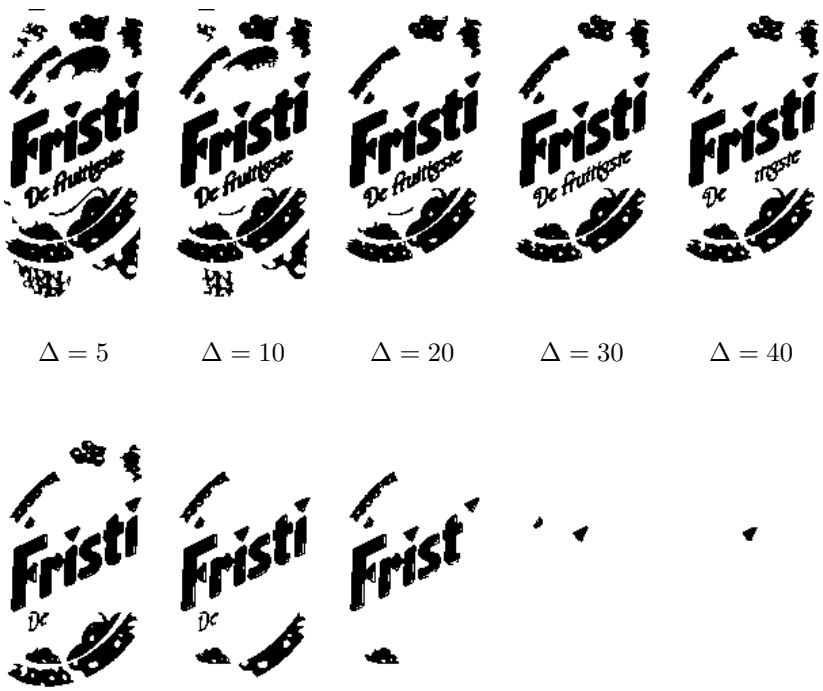
En esta sección, analizaremos la influencia del parámetro Δ en la detección de MSER de nivel inferior (MSER-) calculadas sobre el árbol de componente de una imagen.

En la figura 3.11, cada región obtenida se graficó con color negro para observarlas en conjunto. La secuencia de imágenes fue obtenida luego de haber variado el parámetro Δ desde 5 hasta 90, valores mayores a 90 no lograron detecciones. Además, filtramos por área conservando las mayores a 10 y menores a 3000 píxeles, y fijamos el umbral de máxima variabilidad en 3.

Podemos analizar mejor los resultados de la figura 3.11 si miramos bien la imagen original 3.4a. Primero, vemos que las detecciones se condicen con las características más salientes de la figura, éstas son las letras y los racimos de frutas, ya que al calcular las MSER- estamos favoreciendo la detección de regiones oscuras contrastadas (con fondo claro). En particular, los puntos de la letra *i* de la palabra *Fristi*, se conservan para grandes valores de Delta, esto se debe a su alto contraste y diferencia de área con la región que lo incluye. Luego, el cabo del racimo de uvas y las letras debajo de la palabra *Fristi* poseen un contraste no tan marcado con su fondo y, por eso, se dejan de detectar para valores inferiores.

También podemos ver en las frutas de la zona inferior de la imagen - partes

de ellas - logran un alto contraste por las líneas blancas onduladas que las atraviesan, ocasionando que sean detectadas para altos valores de Delta.



$\Delta = 5$ $\Delta = 10$ $\Delta = 20$ $\Delta = 30$ $\Delta = 40$

$\Delta = 50$ $\Delta = 60$ $\Delta = 70$ $\Delta = 80$ $\Delta = 90$

Figura 3.11: MSER-: variación del Δ entre 5 y 90

$\Delta = 5$ $\Delta = 10$ $\Delta = 20$ $\Delta = 30$

$\Delta = 40$ $\Delta = 50$ $\Delta = 60$ $\Delta = 70$

Figura 3.12: MSER+: variación del Δ entre 5 y 70

Iteración de MSER

En la figura 3.13, vemos la iteración del criterio de MSER. La primera iteración se consigió aplicando el criterio al árbol de componentes y, las subsiguientes, aplicándolo al árbol resultante de la iteración anterior.

Podemos ver con este ejemplo como se refuerza la aseveración de estabilidad sobre las regiones que se conservan.

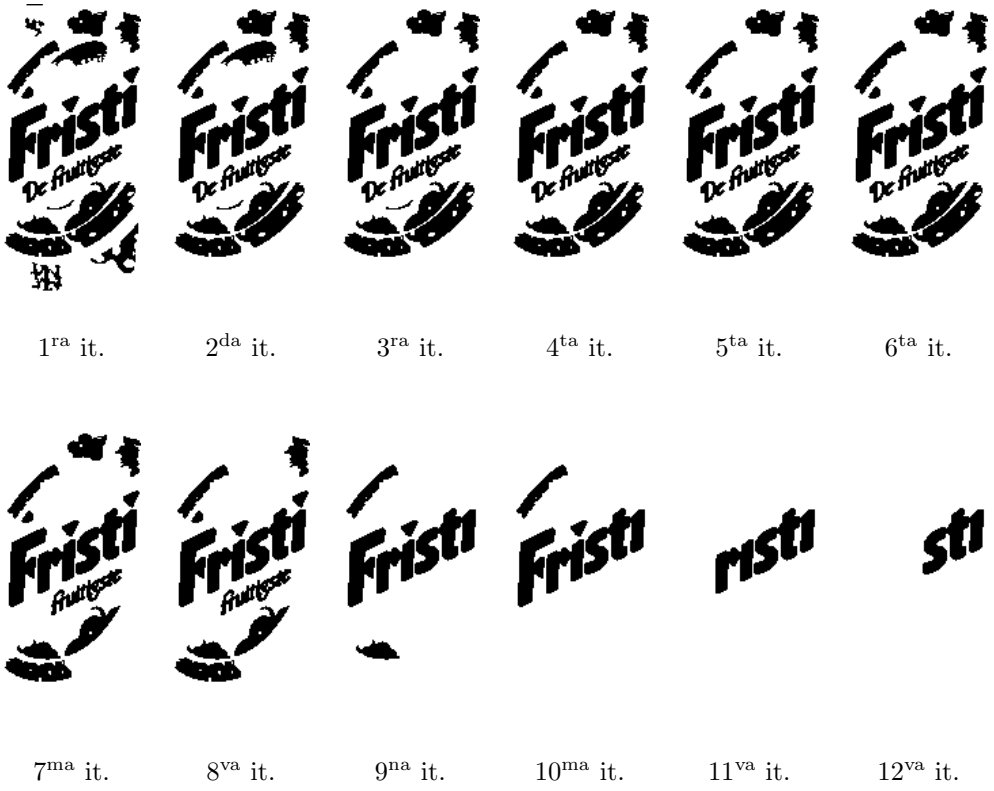


Figura 3.13: Aplicación iterativa del criterio de MSER.

3.2.4. Propiedades de las MSER

En esta sección, haremos un breve repaso de las principales propiedades de la *regiones extremas máximamente estables* (MSER) calculadas sobre el árbol de componente de una imagen.

Para todos los ejemplos, calculamos las MSER- con el parámetro $\Delta = 10$. En éstas, se filtraron las MSER grandes, conservando las de área entre 10 y 3000 píxeles. También se descartaron las regiones muy inestables, en las cuales el valor de la función de estabilidad excedía a 3.

Con estos mismos parámetros, también se obtuvieron las MSER de los ejemplos 3.4b y 3.4c, que tendremos que tener presentes para las comprobación de las propiedades.

Brevemente éstas son:

- Invariabilidad a cambios afín del contraste
- Invariabilidad a rotación y traslación
- Invariabilidad a cambios de escala
- Estabilidad

La estabilidad de las MSER es una propiedad inherente a ellas, ya que son elegidas solamente las regiones extremas cuyo soporte prácticamente no cambia en un rango de umbrales. Ésta es una de las propiedades más importantes de las MSER y, gracias a ella, es que el métodos es tan exitoso y usado en aplicaciones prácticas. Esta propiedad la evaluaremos en las sección 3.2.3 y la tendremos en especial consideración para todos los análisis efectuados.

Invariabilidad a cambios afín del contraste

Vemos en las figuras 3.15 y 3.14. En ambas, se ve a la izquierda la imagen alterada con un cambio de contraste y, a la derecha, sus MSER resultantes. Si bien hay diferencias con la imagen 3.4b en las regiones detectas, las más estables permanecen inalteradas.

Figura 3.14: Izquierda: Imagen alterada con un cambio de contraste. Derecha: sus MSER.

Figura 3.15: Izquierda: Imagen alterada con un cambio de contraste. Derecha: sus MSER.

Invariabilidad a rotación y traslación

Toda transformación de la imagen que preserve áreas mediante una constante multiplicativa, también preservará las MSER. Esto incluye transformaciones afines. En la imagen 3.17 vemos el resultado de aplicar una transformación de similaridad a la imagen original 3.4a, y luego calcular sus MSER. Podemos apreciar como las MSER permanecieron inalteradas completamente.

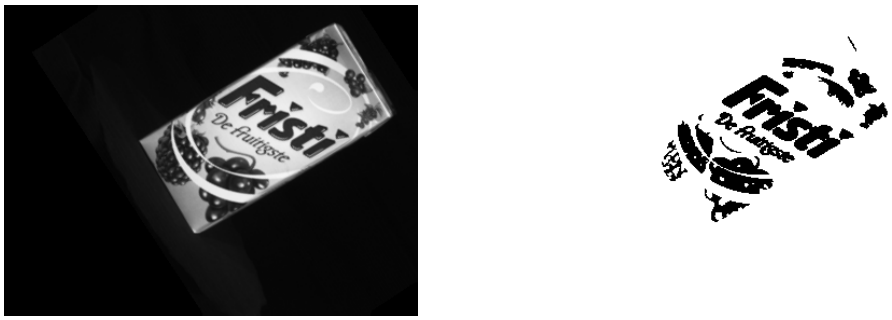


Figura 3.16: Izquierda: Imagen alterada con una rotación y traslación. Derecha: sus MSER.

Invariabilidad a cambios de escala

Al escalar la imagen, si bien cambian los tamaños de las regiones, la relación entre las áreas de las regiones se conserva. Esta propiedad es justamente la que se verifica al buscar una MSER, y de aquí la facilidad de ser detectadas para distintos cambios de escala. Sin embargo, existe la posibilidad de incrementar aún la invariabilidad a cambios de escala haciendo un análisis de multi-resolución [for07].

Figura 3.17: Izquierda: Composición con 4 imágenes a una escala del 25 % de su tamaño orignal. Derecha: sus MSER.

3.3. Formas máximamente estables

En el trabajo de Cao et al. [CLM+08], se esquematiza un proceso para obtener un descriptor *MSER mejorado*: se propone extraer las MSER del árbol de formas de la imagen, y luego codificarlas utilizando una normalización global geométrica.

Vale notar que en [CLM+08], si bien se utiliza el árbol de formas para extraer las MSER, no se detalla su procedimiento y tampoco se analiza experimentalmente.

En está sección, analizaremos la aplicación del criterio de MSER en el árbol de formas extraído de una imagen interpolada bilinealmente, con el objetivo

de extraer características para elaborar descriptores de formas. Las formas que cumplen con este criterio son denominadas formas máximamente estables.

3.3.1. Extracción de las formas máximamente estables

En la sección 3.2.2, hemos visto un procedimiento genérico que permite calcular las regiones máximanente estables de una imagen, sobre un árbol de inclusión. Para el caso de un árbol de inclusión de formas (ver sección 2.4), permite calcular las formas que tendrán la propiedad de ser máximamente estables. Aquí el crecimiento de una región extrema R se puede ver como el crecimiento de su forma asociada. En el árbol de formas (ver sección 2.4), los nodos están ordenadas por inclusión y por ende, una rama representa el crecimiento de la forma más chica, correspondiente a una hoja.

En la figura 3.18, se pueden ver ejemplos de formas máximamente estables, en donde, para cada forma detectada indicamos con un color distinto su interior (figura 3.18b); también, en la figura 3.18c mostramos los contornos (líneas de nivel) de las mismas formas detectadas. Los contornos de las formas se calculan con un postprocesamiento como se indica en la sección 2.7.

(a) (b) (c)

Figura 3.18: (a) Image extraída de [NNM96]. (b) Áreas de las formas máximamente estables para $\Delta = 10$. (c) Contornos de las formas máximamente estables para $\Delta = 10$.

Las formas máximamente estables tienen la ventaja sobre las MSER, de que no se realiza dos veces el cálculo para detectar los objetos claros y los oscuros. En el árbol de formas, los ojetos presentes en una imagen son caracterizados por formas, y por ende, los objetos claros coinciden con las formas de nivel superior que poseen “fondo” oscuro, y los oscuros coinciden con las formas de nivel inferior con “fondo” claro. Por lo tanto, obtenemos en una única estructura representados a todos los objetos de una imagen. Sin embargo, la desventaja que se presenta es que la implementación del árbol de formas de una imagen interpolada bilinealmente es difícil de conseguir; y posee un orden de complejidad levemente mayor al de extracción de árbol de componentes (necesario para obtener las MSER).

3.3.2. Influencia de los parámetros

La cantidad de parámetros para la selección de las regiones extremas máximamente estables en la definición inicial es uno: el Delta Δ. Sin embargo, es muy importante hacer una selección entre las MSER detectadas, ya que es muy común que aparezcan MSER redundantes. Además, otro problema que se presenta y debe ser manejado es el de controlar la máxima variación de área permitida para evitar detecciones espúreas. Por lo tanto, en las siguiente secciones, introducimos dos nuevos parámetros para solucionar estos problemas.

Delta Δ

En la sección 3.2.3, hicimos un repaso de la influencia del parámetro Δ en la detección de las MSER. Para poder establecer una adecuada comparación con las formas máximamente estables, es necesario primeramente, analizar la influencia del mismo.

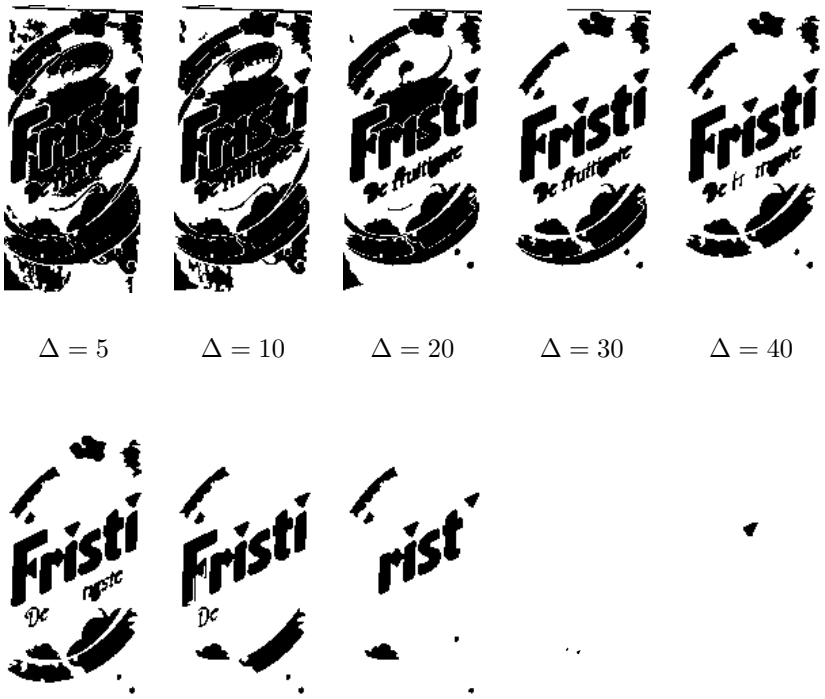
En la figura 3.19, vemos las formas detectadas para distintos valores de Δ. Éstas se graficaron en color negro para observarlas en conjunto. Además, se filtraron las de área grande, conservando las de tamaño entre 10 y 3000 píxeles. También se descartaron las regiones muy inestables, en las cuales el valor de la función de estabilidad excedía a 3.

Los resultados fueron obtenidos variando el parámetro Δ desde 5 hasta 90, valores mayores a 90 no lograron detecciones.

Al comparar los resultados obtenidos con la imagen original (ver figura 3.4a), podemos observar que las formas detectadas coinciden con lo objetos más contrastados de la imagen. Además, se puede notar que tanto las formas contratadas con fondo claro y oscuro están presentes simultáneamente.

Algo para destacar, es que las letras presentes en la figura son detectadas para muchos valores de delta por su alto contraste.

En la figura 3.20, se grafican los contornos de la formas obtenidas en la figura 3.19. El objetivo de mostrar los contornos, es obsevar como se localizan las formas máximamente estables, ya que, como veremos en la sección 4.2, sobre éstas curvas en donde se calcularán los descriptores.

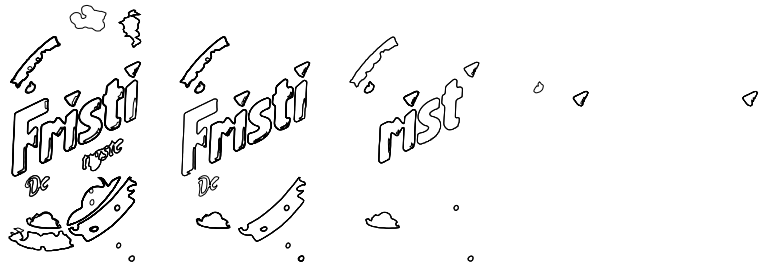


$\Delta = 5$ $\Delta = 10$ $\Delta = 20$ $\Delta = 30$ $\Delta = 40$

$\Delta = 50$ $\Delta = 60$ $\Delta = 70$ $\Delta = 80$ $\Delta = 90$

Figura 3.19: Variación del Δ entre 5 y 90

$\Delta = 5$ $\Delta = 10$ $\Delta = 20$ $\Delta = 30$ $\Delta = 40$



$\Delta = 50$ $\Delta = 60$ $\Delta = 70$ $\Delta = 80$ $\Delta = 90$

Figura 3.20: Variación del Δ entre 5 y 90

Redundancia de Regiones

Como ya vimos en la sección 3.1, las líneas de nivel (los bordes de los conjuntos de nivel), en una imagen natural, es decir, una escena de la naturaleza obtenida mediante un lente fotográfico y luego condificada, presenta mucha redundancia sobre los objetos que el ojo humano percibe. Esto se puede justificar por la dificultad que tiene un medio análogico (lente óptico) de representar altas frecuencias, comúnmente presentes en los bordes de los objetos.

Esta redundancia se transmite en las detecciones de las formas máximamente estables. Para evitar esta redundancia, presentamos un criterio para filtrar las formas máximamente estables que sean *similares*. El criterio de similitud elegido está basado en la relación entre áreas. Éste criterio consiste en eliminar las formas que se consideran similares a su padre en el árbol de formas máximamente estables.

El procedimiento es el siguiente, dado un árbol de formas $\mathcal{T}$, sea una rama $F_1, \ldots, F_n \in \mathcal{T}$ donde F_1 es la raíz y F_n es una hoja.
Luego, dos formas F_i, F_j, con $1 \leq i < j \leq n$, detectadas como máximamente estables, tal que F_i ansestro de F_j en el árbol $\mathcal{T}$, y no existe F_k máximamente estable, con $i < k < j$. Por lo tanto, si

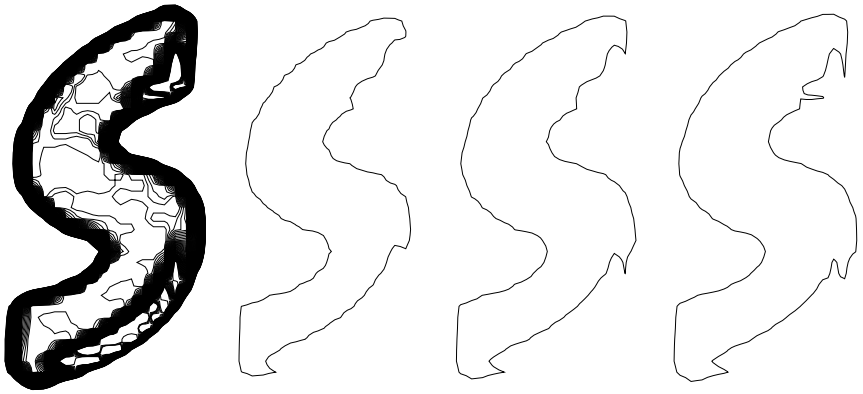
$$\frac{\#F_j}{\#F_i} > minSim$$

la forma F_j es considerada *similar* a F_i, y se elimina de $\mathcal{T}$. Luego, $minSim$ es un nuevo parámetro del método.

Nótese que vale: $0 < \frac{\#F_j}{\#F_i} < 1$, ya que el árbol de formas $\mathcal{T}$, es un árbol de inclusión (ver sección 2.4).

A continuación mostramos un ejemplo de detección de formas máximamente estables detectadas para una rama del árbol de formas, extraídas de la figura 3.4a, con $\Delta = 10$. En la figura 3.21a, podemos ver los contornos de todas la formas de dicha rama. Luego, en las figuras 3.21 y 3.22 se ven algunas de las formas máximamente estables detectadas en donde se puede apreciar la similitud entre ellas, y por ende, la redundancia presentada en las detecciones.

En la figura 3.23, graficamos la variación de área y la función de estabilidad para las formas de la rama, en donde, se puede apreciar con cruces los valores en donde se producen las formas máximamente estables.



(a) (b) (c) (d)

Figura 3.21: (a) Contornos de las formas de una rama del árbol de formas de la figura 3.4a. *Formas máximamente estables:* (b) nivel 64 y área 260 píxeles, (c) nivel 79 y área 288 píxeles, (d) nivel 92 y área 309 píxeles. En todos los casos se definió $\Delta = 10$.

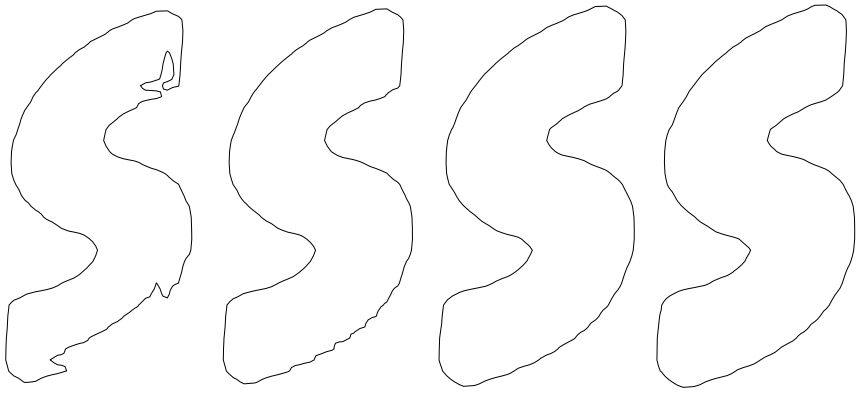
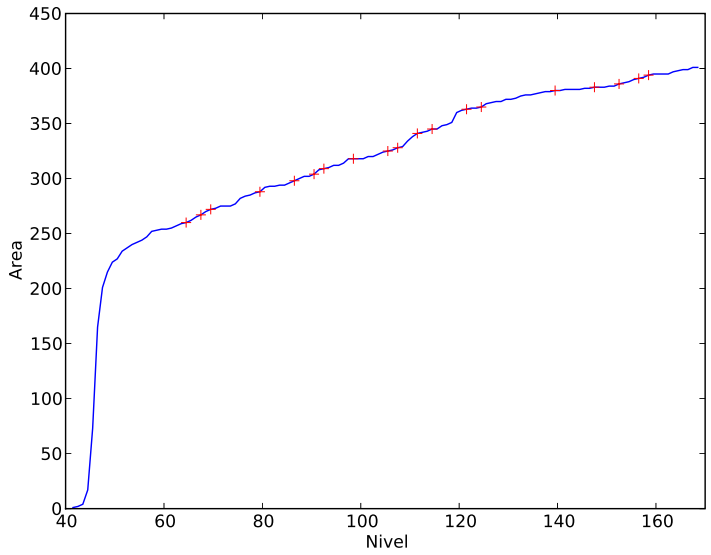
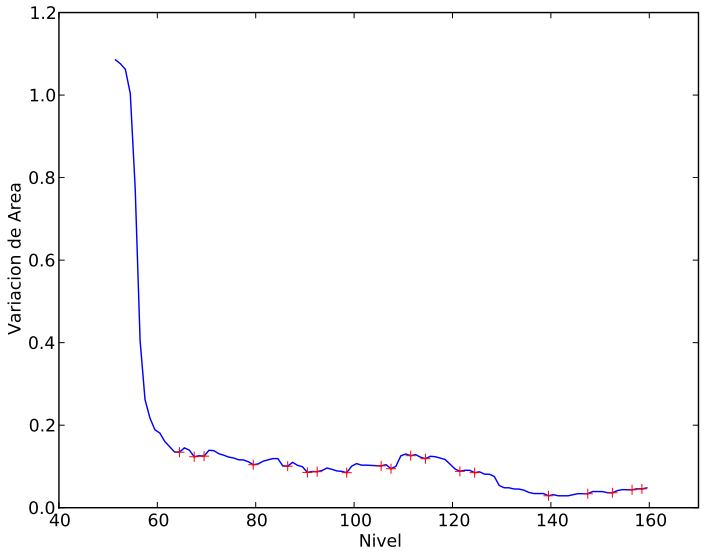


Figura 3.22: *Formas máximamente estables:* (a) nivel 107 y área 328 píxeles, (b) de nivel 121 y área 363 píxeles, (c) de nivel 147 y área 383 píxeles, (d) de nivel 158 y área 394 píxeles. En todos los casos se definió $\Delta = 10$.



(a)



(b)

Figura 3.23: Gráficos para la rama de la figura 3.21a con $\Delta = 10$. (a) Gráfico de área en función de nivel de gris. (b) Función de estabilidad. La cruz indica dónde se producen los mínimos de la función de estabilidad (las formas máximamente estables).

3.4. Evaluación de las formas máximamente estables

En esta sección, realizamos una evaluación empírica de los puntos en común y diferencias entre las formas máximamente estables y las regiones extremas máximamente estables (MSER) por un lado, y las formas máximamente estables y los contornos significativos por el otro. Con el objetivo de comparar el método como detector de características de una imagen.

Primero, realizamos la comparación más natural, que es cotejar las detecciones entre las regiones extremas máximamente estables obtenidas sobre el árbol de formas (formas máximamente estables) y sobre el árbol de componente (MSER).

Luego, comparamos las formas máximamente estables con los contornos significativos, ya que éstos últimos, serán usados como detector de caracterísiticas en el siguiente capítulo. Esta comparación es más difícil de establecer debido a la diferente naturaleza de ambos métodos.

En las secciones siguientes nos referiremos a las formas máximamente estables por su sigla FME.

3.4.1. Comparación con MSER±

El objetivo de esta comparación es complementar las comparaciones hechas hasta el momento entre las FME y las MSER, que se comentaron en las secciones previas.

Llamaremos MSER± al conjunto de MSER extraídas de los conjuntos de nivel superior (MSER+), junto con el conjunto de MSER extraídas de los conjuntos de nivel inferior (MSER-).

Debido a la distinta naturaleza de los dos métodos, tenemos que encontrar una forma de compararlos que no sea injusta para ninguno de los dos, es decir, que el análisis no es esté sesgado en desmedro o privilegio de alguno de los métodos. Por esto se decidió hacer la comparación de la unión de las MSER+ y MSER-, contra las FME de un modo cualitativo.

Otro punto importante a tener en cuenta en la comparación es la diferencia entre forma y componente. Esto ya fue discutido en la sección 2.4.2,

Figura 3.24: Izquierda: Detalle de la imagen original. Medio: MSER-. Derecha: MSER+

Figura 3.25: Izquierda: MSER±. Centro: FME. Derecha: Contornos significativos.

En la figura 3.24, podemos ver las MSER+ y MSER- de área entre 10 y 1500 píxeles de la imagen. Ambas fueron obtenidas con $\Delta = 10$, también fueron filtradas las regiones muy inestables ($maxVar > 3$) y las MSERs muy similares

($minSim = 0{,}7$). Cada color distinto denota una región distinta. Con los mismos parámetros fueron obtenidas las FME.

Podemos apreciar que, a grandes rasgos, las regiones detectadas son similares. Salta a la vista que las detecciones no son las mismas, pero que las formas en la imagen de las FME tienen su componente homóloga entre las MSER±. Esto ocurre con las componentes simples, convexas, regulares, con pocos (o ningún) agujero, es decir, se asemejan al área delimitada por una forma (ver sección 2.4.2). Hay un total de 81 MSER± y 87 FME.

En la sección 2.4.2, discutimos la diferencia entre las dos representaciones de la imagen dadas por el árbol de formas y el de componentes. Vimos que la idea de ambas representaciones es similar; luego, para una componente o una forma en su respectivo árbol, si está suficientemente contrastada, estará incluida en una componente o forma mayor con una diferencia notable en los valores de gris (necesario para lograr el contraste). Esto ocasionará que sea detecta como máximamente estable, ya que para un rango de valores de gris (dependiendo el delta) permanecerá con el mismo valor de área.

Por otro lado, un hueco es una forma que posee inversión de contraste con respecto a la forma que la incluye. Esta inversión, hace que la variación de área para el hueco se mantenga estable entre los niveles de gris que la separan de la forma que la incluye, propiciando que sea detectada como máximamente estable en el árbol de formas. En el árbol de componentes, esta diferencia de niveles de gris se presentará en el árbol superior o inferior, según a cuál pertenezca la componente correspondiente al hueco, y también propiciará ser detectada como máximamente estable.

Esto nos da la pauta de que la influencia del parámetro Δ en ámbos métodos es la misma,

mientras que, si bien estos tipos de regiones son detectadas por ambos métodos, los valores de variación de área no serán los mismos, ocasionando que el parámetro de máxima variación de área ($maxVar$) no tenga la misma influencia.

Algo para notar entre las imágenes son los huecos que se presentan en la letra “F” y “S” de la imagen de MSER. En el árbol de formas, los píxeles pertenecientes a los huecos son parte de la forma que lo incluye, esto no es así en el árbol de componentes. Por lo tanto, si se remueven los huecos de una componente, se presentarán “porosidades” ocasionadas por el faltante de píxeles.

Por otro lado, la conexión entre los píxeles afecta la diferencia entre las detecciones, ya que tiene una relación muy directa con el área en la conformación de una componente o forma (bilineal), entonces se producirán distintos valores de variación de área y, por ende, otras regiones estables serán reconocidas o filtradas como inestables.

Las regiones de abajo (color rosa) y arriba (color violeta) de las letras *Fristi* en la imagen 3.25 –que fueron detectadas como FME pero no tienen su componente homóloga en la imagen de las MSER– son producto de lo discutido anteriormente, ya que están cerca de ser filtradas como inestables: 2.6 y 2.14 de variación de área correspondientemente.

3.4.2. Comparación con contornos significativos

En esta sección, comparamos brevemente las formas máximamente estables (FME) con los contornos significativos (CS). Hay que aclarar que, debido a las diferencias en los planteos teóricos de ambos métodos, es muy difícil realizar una comparación exhaustiva; igualmente, se pueden destacar algunos puntos interesantes.

Las FME de la figura 3.26 fueron obtenidas con un valor de $\Delta = 10$, $minSim = 0{,}7$, $maxVar = 3$, sin filtrar por área y cuantizando previamente el árbol de formas. Para la cuantización, se eligió el mismo paso que para los contornos significativos de la misma imagen. Los contornos significativos fueron obtenidos con un valor de $\epsilon = 1$.

Vale aclarar que, en un árbol cuantizado, el nivel de gris puede mantenerse constante y variar el área. Esto es un problema a la hora de calcular la función de estabilidad, ya que la variación de área deja de ser función (pierde la propiedad inyectiva). La decisión tomada para evitar este problema fue que, al presentarse esta situación, se consideren dos dominios distintos de la función de área, y se calcule separadamente la función de estabilidad en ambos.

Figura 3.26: Izquierda: Contornos FME. Derecha: Contornos Significativos.

Ambos métodos tienen un punto muy fuerte de comparación, y es que pueden ser vistos como procesos de filtrado (o selección) sobre el árbol de formas de una imagen interpolada bilinealmente.

Luego, si bien los métodos son muy distintos, ambos seleccionan curvas contrastadas. Por un lado, como vimos en la sección 3.2.1, elegir las formas máximamente estables puede ser visto como una selección de las formas que poseen mayor contraste a lo largo de su contorno. Por otro lado, para los contornos significativos, explícitamente se calcula el contraste sobre cada curva, y luego la selección es sobre las mejor contrastadas y perceptualmente más interesantes.

Se puede ver en la teoría de percepción visual de la Gestalt [Kan79] que la percepción visual humana no se basa solamente en distinguir objetos con alto contraste, por lo que un método (como el de contornos significativos) que emplee mayores herramientas de la percepción visual (además del contraste), para la detección de formas interesantes, será más coherente con las características salientes que deseamos obtener sobre una imagen.

Podemos apreciar, para los contornos significativos de la figura 3.26, que las líneas de nivel que se presentan en paquetes encuentran su homólogo en las formas máximamente estables, ya que son curvas bien contrastadas a lo largo de todo su contorno y se detectan por ambos métodos. Luego, para las FME hay algunas detecciones presentes y otras ausentes con respecto a los CS; esto es debido a la diferencia en el cálculo de contraste y significación de las curvas por los dos métodos.

Capítulo 4

Reconocimiento de formas

En el reconocimiento de patrones en imágenes, el objetivo principal es decidir si un objeto está presente o no en una escena. En el caso del reconocimiento de formas, un objeto es representado por una o varias formas, y la escena constituye la base de datos o el conjunto de formas en dónde buscar.

Una solución categórica a este problema es muy difícil de conseguir, por ello es que se establecen criterios para medir, con cierto grado de precisión, cuando dos formas son semejantes, para luego poder encontrar la más parecida en toda la base de datos.

Podemos dividir el proceso de reconocimiento de formas principalmente en tres etapas:

1. **Detección:** extracción de las características (formas) presentes en una imagen.

2. **Codificación:** elaboración de descriptores de las formas obtenidas en el paso previo.

3. **Matching** o Correspondencia: estimación del grado de similitud entre los descriptores y comparación de ellos.

En este capítulo, estudiamos el reconocimiento de formas en imágenes mediante el uso de los contornos significativos y las formas máximamente estables, como dos métodos de detección de características (ver capítulo 3). Para la etapa de codificación de formas utilizamos el descriptor *shape context*; y usamos un *matching a-contrario* entre los descriptores para encontrar las formas más parecidas. El proceso así descripto nos permite desarrollar un nuevo reconocedor de formas que lo denominamos *shape context morfológico*, y veremos sus virtudes en aplicaciones de búsqueda en video por contenido y establecimiento de correspondencias.

Como pudimos ver en la sección 2.7, las líneas de nivel son los bordes topológicos o contornos de las formas. Utilizar curvas para representar un objeto nos brinda la posibilidad de calcular propiedades locales a éste. En particular, nosotros utilizaremos el *shape context* sobre las líneas de nivel que por su naturaleza nos brinda un descriptor semi-local de la forma, en contraposición de los descriptores globales ([RC96], [KH90]) que actúan sobre la totalidad de la imagen. Por lo tanto, los descriptores globales son más apropiados para la tarea

de buscar imágenes similares que para ser usados en el reconocimiento de formas. En particular, en el reconocimiento de formas, en menester que los objetos individuales presentes en la imagen puedan ser correspondidos separadamente.

Los resultados de nuestro proceso de reconocimiento de formas muestran que el *shape context morfológico* es apropiado para trabajar en situaciones en las que el proceso de reconocimiento de formas *shape context* [BMP02] no tiene éxito.

4.1. Shape context

En esta sección, repasamos el trabajo de Belongie et al. [BMP02], en donde se propone un proceso de reconocimiento de formas que utiliza un novedoso descriptor de formas denominado *shape context*.

Mientras no queden ambigüedades, al proceso de reconocimiento de formas del trabajo de Belongie et al. [BMP02] lo llamaremos simplemente *shape context*, aunque la denominación *shape context* es la usada en el mencionado trabajo para refererise al descriptor utilizado.

Shape context implementa el proceso de reconocimiento de formas de la siguiente manera:

1. **Detección:** mediante algún detector de bordes (e.g. Canny [Can86]).

2. **Codificación:** se desarrolla el descriptor *shape context*. El *shape context* en un punto modela la distribución de las posiciones relativas de los demás puntos con respecto a él.

3. **Matching:** se establece el apareamiento entre los *shape context* resolviendo el *problema de asignación óptima* [GY05]. Luego, se estima la disimilitud entre las formas, y se deciden las correspondencias mediante una clasficación de vecinos más cercanos.

Para el primer paso del proceso las formas de los objetos se representan como un conjunto finito de puntos muestreados a lo largo de los contornos del objeto. Éstos son obtenidos mediante cualquier detector de bordes. Preferentemente, los puntos se eligen con un espaciado uniforme y, asumiendo que un contorno es suave de a pedazos, al tomar una cantidad suficientemente grande de puntos, se obtiene una buena aproximación de la forma.

El segundo paso consiste en generar los descriptores *shape context* en los puntos muestreados. La generación de los descriptores *shape context* la describiremos extensivamente en la sección 4.1.1.

La motivación detrás de esta técnica consiste en medir la similitud entre formas, ya que formas similares tendrán *shape context* similares, para luego lograr un reconocedor a nivel de categorías.

En el tercer paso, la idea es encontrar las mejores correspondencias entre los *shape context* de las formas[1]. Este es un problema conocido que se denomina

[1] El problema de buscar correspondencias entre dos conjuntos de elementos –esto es, dado un criterio de asignación, buscar para un elemento de un conjunto cuál es el que le corresponde en el otro– tiene su origen en la teoría de grafos bajo el nombre de *matching en grafos bipartitos* [GY05]. La palabra *matching*, aquí traducida como *correspondencia* no suele tener una traducción unívoca, y muchas veces, en la bibliografía del tema, se opta por no traducirla, con el fin de no perder el sentido que le otorga la palabra al problema. También podemos encontrar

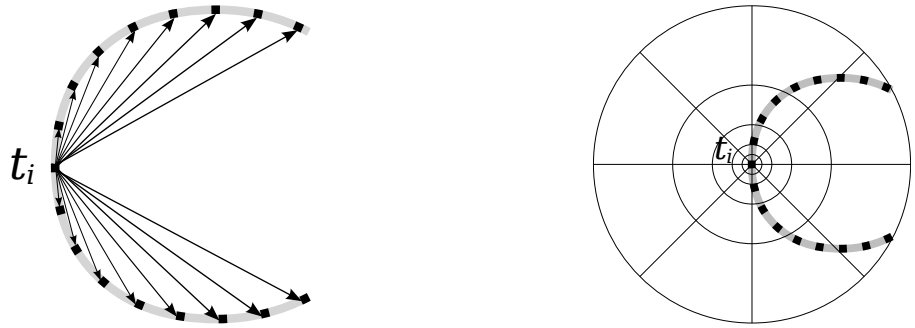
"matching de peso mínimo en grafos bipartitos" o "problema de la asignación óptima en grafos bipartitos", que posee una solución eficiente.

Para refinar las correspondencias, se establece una transformación de alineamiento. Luego, la disimilitud entre dos formas puede ser calculada como el error de apareamiento entre los puntos que se corresponden, junto con un término que mide la magnitud de la transformación de alineamiento.

Finalmente, se utiliza la técnica de vecinos más cercanos [CH67] para hacer el reconocimiento de formas.

4.1.1. Descriptor *shape context*

La primera idea surgida en el desarrollo del descriptor *shape context* [BMP02] fue pensarlo como un conjunto de vectores desde un punto determinado al resto (ver figura 4.1a).



(a) (b)

Figura 4.1: Descriptor *shape context* en el punto t_i de la curva semejante a la letra *C*. Los puntos negros denotan el muestreo. (a) Primera aproximación para la creación del descriptor *shape context*. (b) Casilleros del histograma en el espacio log-polar alrededor del punto t_i.

El problema que se presentaba es que el descriptor así representado es demasiado detallado, siendo que las formas y los puntos muestreados pueden variar dentro de una categoría de objetos del mismo tipo. Por esto, surgió la idea final, que fue considerar para cada punto la distribución de las coordenadas relativas a los otros puntos, y se lo llamó *shape context*.

El descriptor *shape context* en un punto es modelado por un histograma de las posiciones relativas a los demás puntos muestreados. Los casilleros del histograma son tomados uniformemente en el espacio log-polar, haciendo al descriptor más sensitivo a las posiciones de los puntos más cercanos (ver figura 4.1b).

Sea $\mathcal{T} = \{t_1, \ldots, t_n\}$ el conjunto de muestras tomadas de la forma de entrada. Para cada $t_i \in \mathcal{T}$, $1 \leq i \leq n$, la distribución de las distancias a los $n-1$ puntos

conjugaciones y adaptaciones del vocablo al castellano, como suele ocurrir ubicuamente en el ámbito computacional e informático. En este trabajo, utilizaremos las dos terminologías en pos de la claridad y mejor comprensión del tema.

restantes en $\mathcal{T}$ es modelada relativa a cada t_i como un histograma log-polar. Llamamos el *shape context* de t_i (SC_{t_i}) a esta distribución.

Formalmente, sea un espacio polar $[0, 2\pi]\times\mathbb{R}$ y sea $\Theta = [0 = \alpha_0, \alpha_1, \ldots, \alpha_A = 2\pi]$ una partición de $[0, 2\pi]$, y $\Delta = [0 = d_0, d_1, \ldots, d_D]$ sea una partición del rango $[0, d_D]$, donde d_D es la distancia al punto más lejano en $\mathcal{T}$ relativo a t_i. Luego, $\Theta \times \Delta$ constituye una partición de $[0, 2\pi] \times \mathbb{R}$.

Denotaremos como $h_{t_i}(\alpha_k, d_m)$ al número de puntos en el casillero del histograma $[\alpha_{k-1}, \alpha_k) \times [d_{m-1}, d_m) \in \Theta \times \Delta$, $0 < k \leq A, 0 < m \leq D$. Esta cantidad puede ser definida como:

$$h_{t_i}(\alpha_k, d_m) = \#\{t_j \in \mathcal{T}, j \neq i \ : \\ \alpha_{k-1} \leq \arg(t_j - t_i) < \alpha_k, d_{m-1} \leq ||t_j - t_i||_2 < d_m\}$$

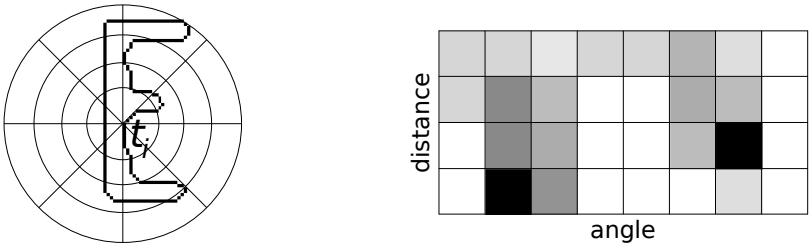
donde $\arg(v)$ es el ángulo del vector v.

El ***shape context*** de t_i denominado como SC_{t_i} se define como la versión normalizada de h_{t_i}.

$$SC_{t_i}(\alpha_k, d_m) = \frac{h_{t_i}(\alpha_k, d_m)}{N}$$

$$N = \sum_{\alpha_a \in \Theta, d_b \in \Delta} h_{t_i}(\alpha_a, d_b)$$

En la figura 4.2, podemos ver un ejemplo de un descriptor *shape context* en un punto de la forma que representa el carácter "E" junto con su histograma asociado.



(a) (b)

Figura 4.2: (a) Muestra en el espacio polar sobre una forma representando al caracter "E". (b) Histograma resultante. Mientras más oscuro sea un casillero, denotará más frecuencia de puntos.

El conjunto de los *shape context* para cada punto muestreado en una forma constituye un poderoso y redundante descriptor.

Propiedades del descriptor *shape context*

Algunas propiedades importantes del *shape context* son:

- Invariante a traslación. Esta propiedad es intrínseca en el *shape context*, ya que todas las mediciones son tomadas con respecto a puntos del objeto.

- Invariante a rotación. Es posible calcular el *shape context* en un punto se utilizando un eje cartesiano cuya abscisa esté ubicada tangencialmente al borde del objeto en el punto. De esta manera, los ángulos que determinan los casilleros del histograma se obtienen con respecto al objeto y no a la imagen; para alcanzar la propiedad invariante a rotación.
- Invariante a cambios de escala. Se obtiene al normalizar las distancias radiales.
- Robustez frente a pequeñas perturbaciones geométricas y distorsiones de la forma. Esto se debe a la utilización de un histograma grosero (*coarse* en inglés), y la posterior refinación de las correspondencias.

4.1.2. *Matching* en grafos bipartitos

En vez de buscar correspondencias entre los puntos muestreados de las formas, se busca para cada punto muestreado de una forma, cuál es el punto muestreado en la otra que tiene el *shape context* más similar. Este es un problema de *matching* en grafos bipartitos, y la idea es minimizar el costo total de la correspondencia entre todos los puntos, uno a uno, entre las dos formas.

Luego, el costo C_{ij} entre dos puntos p_i y p_j se puede definir como la distancia χ^2 entre los *shape context* en los puntos.

$$C_{ij} = d(SC_{p_i}, SC_{q_j}) = \frac{1}{2} \sum_{\alpha_a \in \Theta, d_b \in \Delta} \frac{\left[SC_{p_i}(\alpha_a, d_b) - SC_{p_j}(\alpha_a, d_b)\right]^2}{SC_{p_i}(\alpha_a, d_b) + SC_{p_j}(\alpha_a, d_b)} \quad (4.1)$$

Tomar la distancia χ^2 es una decisión natural, ya que cada *shape context* representa distribuciones modeladas como histogramas.

Luego, teniendo el conjunto de costos C_{ij} entre todos los pares de puntos p_i, en la primera forma, y q_i en la segunda, el objetivo es minimizar:

$$H(\pi) = \sum_{i=1}^{n} C_{i,\pi(i)}$$

donde $\pi(i)$ es una permutación de i.

Éste es uno de los problemas fundamentales de la optimización combinatoria caracterizado como "matching de peso mínimo en grafos bipartitos" o "problema de la asignación óptima en grafos bipartitos", y posee solución de orden de complejidad $O(n^3)$ mediante el método Húngaro [PS98].

Refinación de correspondencias

En el trabajo de Belongie et al. [BMP02], se realiza una búsqueda de una transformación de alineamiento con el objetivo principal de refinar y extender a toda la forma las correspondencias entre los puntos obtenidas por la comparación de los *shape context* entre las formas.

La búsqueda de una transformación de alineamiento está motivada por la observación de que las formas que están relacionadas no son idénticas, éstas pueden ser deformadas para ser alineadas usando tranformaciones simples de

coordenadas (ver figura 4.3). Esto mismo ya lo estudió D'Arcy Thompson en su trabajo orignal *On Growth and Form* [Tho17].

En este trabajo no abarcaremos este tema, y proponemos utilizar el método RANSAC para refinar las correspondencias de la etapa de *matching* como veremos en la sección 4.2.2.

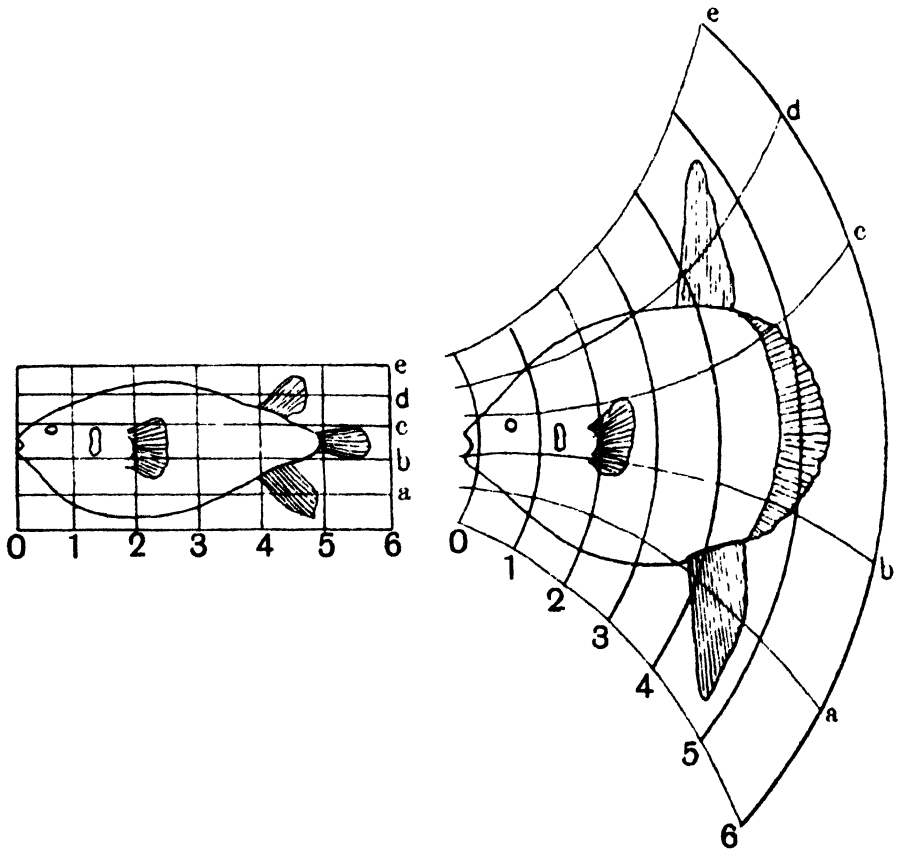


Figura 4.3: Transformación de alineamiento entre dos figuras de un pez. Cada cuadrante de la figura izquierda está relacionado mediante una transformación con su cuadrante homólogo en la figura derecha. Figura extraída del libro de D'Arcy Thompson [Tho17].

4.1.3. Desventajas del descriptor *shape context*

En el artículo de Belongie et al [BMP02] en la definición del descriptor *shape context*, se tomaron varias decisiones que le otorgaron muchas ventajas y desventajas para trabajar en diversas áreas de procesamiento de imágenes. Algunas de estas decisiones pueden ser mejoradas o adaptadas según sea el marco en el que deseemos trabajar.

Primero, la detección de características de la imagen, que tiene como objetivo econtrar los objetos interesantes presentes en ella, es realizada mediante el mapa de bordes, obtenido a través el detector de bordes Canny [Can86]. El mapa de

bordes de una imagen tiene algunos problemas: varios parámetros tienen que ser ajustados manualmente, dependiendo del contraste o ruido de la imagen; y además, los bordes son representados como un conjunto de puntos-borde los cuales deben ser agrupados a posteriori para formar curvas, lo cual no es una tarea trivial.

Segundo, la estapa de muestreo es realizada considerando que el mapa de bordes corresponde a un proceso de Poisson [BMP02]. Este modelo produce un muestreo no determinístico, que ocasiona que distintas corridas del proceso de muestreo, pueden dar resultados diferentes. La consecuencia inmediata es que dos descriptores obtenidos de la misma imagen, adquiridos en distintas ocasiones, pueden no ser iguales. En pocas palabras, se introduce un ruido de perturbarción[2] en el descriptor. En la figura 4.4, se ven los efectos del ruido de perturbación, haciendo que $d(SC_{t_i}, SC_{t_j}) \approx 0{,}11 \neq 0$, donde $d(\cdot,\cdot)$ es la distancia χ^2 definida en la ecuación 4.1.

Desde nuestro punto de vista, la principal falencia del *shape context* es que hereda las debilidades del mapa de bordes que hemos mencionado previamente. Este hecho, tiene un gran impacto en la codificación de la forma: no hay una diferenciación intrínseca entre lo que es global y lo que no. Un ejemplo de esto se puede ver en la figura 4.4, donde $d(SC_{t_i}, SC_{t_k}) \approx 0{,}3$ se ubica claramente por encima del ruido de perturbación $d(SC_{t_i}, SC_{t_j}) \approx 0{,}11$. En pocas palabras, una pequeña modificación de la forma tiene un gran impacto en la distancia.

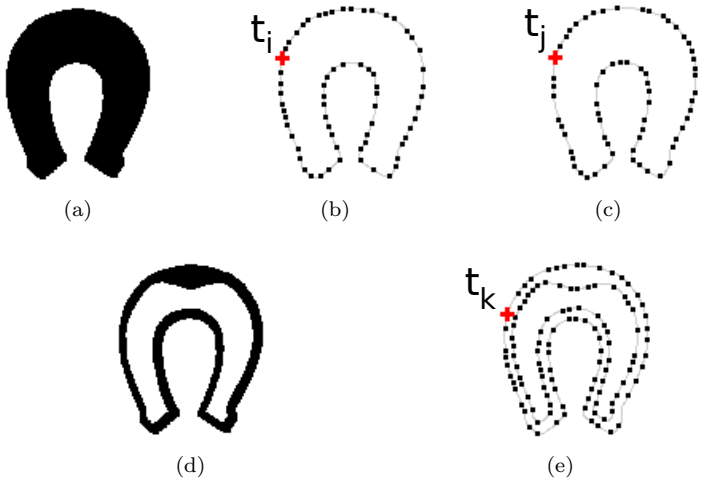
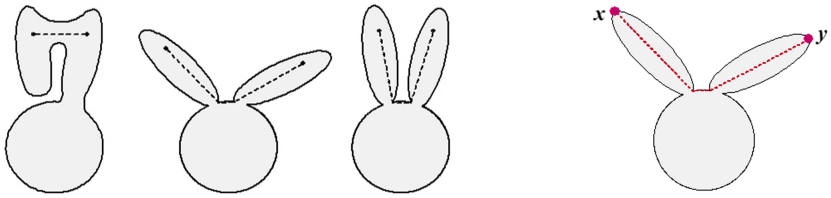


Figura 4.4: (a) Imagen *horsehoe1*; (b) puntos muestreados para *horsehoe1*; (c) otros puntos muestreados de *horsehoe1*, con el mismo proceso de muestreo que en (b); (d) Imagen *horsehoe2*; (e) puntos muestreados de *horsehoe2*, con el mismo proceso de muestreo que en (b) y (c). Los puntos t_i, t_j y t_k están en la misma posición en la imagen.

Tercero, la definición del histograma que modela el *shape context* en cada punto (sección 4.1.1), se realiza dividiendo uniformemente el espacio log-polar;

[2]Traducción libre del término inglés *jitter*.

por ende, se utiliza la distancia euclídea para saber a qué casillero del histograma corresponde cada punto. La distancia euclídea no es la relación métrica más adecuada para capturar la estructura de las formas. Las formas de los objetos pueden poseer articulación y/o variaciones leves en su composición como se puede ver en la figura 4.5a. En el trabajo de H. Ling y D. W. Jacobs [LJ07] proponen utilizar la "distancia interior" para crear un descriptor *shape context* que responda a éstos cuestionamientos. La distancia interior es definida como la longitud del camino más corto **dentro** de la forma entre los puntos muestreados (ver figura 4.5b). El descriptor así propuesto, conserva la misma estructura del *shape context* con la excepción de que es más robusto a comparar formas que poseen articulación en sus partes. En este trabajo no incluiremos la distancia interior en la elaboración del descriptor *shape context*, sin embargo, utilizamos una agrupación de los casilleros del histograma log-polar para mejorar la etapa de *matching*, como veremos en la siguente sección 4.2.



(a) (b)

Figura 4.5: (a) Ejemplos de un objeto con articulaciones y cambios leves en su estructura. (b) Detalle de la distancia interior entre los puntos x e y. Figuras extraídas del artículo H. Ling y D. W. Jacobs [LJ07].

Cuarto, para la comparación entre *shape context* se utiliza la distancia χ^2 (ver sección 4.1.2). Este ítem es dependiente de la etapa de matching y de la forma en que se crea el histograma. La distancia χ^2 es muy usada para la comparación de histogramas pero no es la única; es posible establecer distancias entre histogramas utilizando correlación, intersección [SB91], *earth moverś distance* [RGT97], *Bhattacharyya distance* [Bha43], etc. La elección de la misma tiene gran influencia en la decisión de determinar la similitud entre formas. En nuestro trabajo no exploramos esta área, y conservamos la distancia χ^2 propuesta por Belongie et al [BMP02] para la comparación de *shape context*.

Estas desventajas que presenta el *shape context* son las motivaciones de la elaboración del descriptor *shape context morfológico* el cual estudiaremos en la sección 4.2.

4.2. Shape context morfológico

Con el objetivo principal de superar los incovenientes que presenta *shape context*, presentamos un nuevo proceso de reconocimiento de formas llamado

shape context morfológico (SCM) [3], el cual consiste en fusionar el descriptor *shape context* con las líneas de nivel de una imagen, e incorporar una novedosa etapa de *matching* basada en el modelo de detección *a-contrario* [CLM+08].

El *shape context morfológico* implementa el proceso de reconocimiento de formas de la siguiente manera:

1. **Detección:** mediante la extracción de las líneas de nivel de una imagen interpolada bilinealmente, también llamadas líneas de nivel bilineales.

2. **Codificación:** mediante el descriptor *shape context*. El *shape context* en un punto modela la distribución de las posiciones relativas de los demás puntos con respecto a él.

3. **Matching:** mediante un matching *a-contrario* con el conjunto de descriptores *shape context*, en donde, es posible evaluar la confiabilidad de cada correspondencia obtenida.

El primer paso consiste en seleccionar un conjunto representativo de líneas de nivel del mapa topográfico de una imagen (ver sección 2.7), con el objetivo de encontrar los objetos presentes, ya que éstos pueden ser representados por pedazos de líneas de nivel [DMM01]. Obsérvese en la figura 4.6 que el uso de líneas de nivel continuas extraídas de una imagen interpolada bilinealmente (sección 2.7), nos provee de mucha información detallada y resuelve el problema de conectar los puntos-borde del mapa de bordes de la imagen para construir curvas.

Como hemos visto en el capítulo 3, las líneas de nivel de una imagen suelen ser muy numerosas y redundandes, por ende, es necesario hacer una selección de las mismas para trabajar con un número reducido de características. En esta sección, proponemos utilizar los contornos significantivos (ver sección 3.1) por un lado, y los contornos de las formas máximamente estables (sección 3.3) por el otro, dando lugar a dos variaciones del *shape context morfológico*, denominadas:SCM_{CS} para el primer caso, y SCM_{FME} para el último. Luego, este conjunto de selección de líneas de nivel es el que será usado en la etapa de codificación.

El uso de la teoría de la detección *a-contrario* –inspirada por la teoría de la Gestalt [Kan79]– en la técnica de contornos significativos, provee un significado teórico y efectivo al hecho de seleccionar parámetros, al mismo tiempo que se controla la relación contraste/ruido [DMM08].

[3] El *shape context morfológico* fue presentado en el artículo "M. Tepper, F. Gómez Fernández, P. Musé, A. Almansa y M. Mejail. *Morphological Shape Context: Semi-locality and Robust Matching in Shape Recognition.* 14th Iberoamerican Congress on Pattern Recognition. CIARP 2009" [TGFP+09]. En este trabajo además de su descripción presentamos una extensión del mismo del basado en las *formas máximamente estables.*

(a) (b) (c) (d)

Figura 4.6: (a) Imagen original. (b) Área Seleccionada. (c) Filtro Canny aplicado sobre (a) (detalle del área seleccionada). (d) Contornos significativos extraídos de (a) (detalle del área seleccionada).

Para el segundo paso, utilizamos el descriptor *shape context* tal cual lo vimos en la sección 4.1.1, con la modificación fundamental de que el descriptor *shape context* es restringido a los puntos de cada línea de nivel, es decir, el histograma que modela al *shape context* en un punto de una línea de nivel, sólo considera los puntos de esa misma línea. Esta modificación, junto con la propiedad de disyunción entre líneas de nivel bilineales (nunca se intersectan), nos permite hablar de una cualidad de descripción semi-local de una forma, en cada uno de los puntos de ella.

Asimismo, cuando lidiamos con curvas, el proceso de muestreo es realizado de una forma muy natural mediante la parametrización de la longitud del arco; consecuentemente, es eliminado el ruido de perturbación (*jitter*) introducido en *shape context* al muestrear aleatoreamente los puntos de las formas.

De esta manera, solucionamos los problemas que se presentan en el primer y segundo paso del proceso reconocmiento de formas *shape context* (ver sección 4.1.3).

En el tercer paso, implementamos un proceso de *matching* robusto y estable dentro del marco de trabajo *a-contrario*, que por sus buenas características está muy bien condicionado para el establecimiento de correspondencias entre formas. La etapa de *matching*, generalmente, es la menos estudiada de todo el proceso de reconocimiento de formas. La mayoría de los métodos usan una técnica de vecinos más cercanos para corresponder los conjuntos de descriptores. En esta sección, presentamos un criterio *a contrario* de *matching* para la correspondencia de descriptores *shape context*, que nos brinda la ventaja de poder evaluar independientemente la calidad de cada una de las correspondencias establecidas entre los descriptores, manejando un único parámetro.

Los resultados experimentales obtenidos (ver sección 4.3) muestran que este nuevo proceso de reconomiento de formas es apropiado para ser usado situaciones en las que *shape context* no lo es, tales como la búsqueda en video por contenido. También se observa que la semi-localidad es una característica clave en el *shape context morfológico*, útil para comparar formas en imágenes donde hay también muchas otras formas presentes.

4.2.1. Descriptor *shape context morfológico*

La idea principal de este descriptor es explotar los beneficios de la representación de la imagen definida por el mapa topográfico (sección 2.7) y fusionarla con el descriptor *shape context*.

En *shape context*, cada forma es codificada en conjunto de puntos aislados $\mathcal{T} = \{t_1, ..., t_n\}$ muestreados desde el mapa de bordes de una imagen. El objetivo principal de la modificación introducida por el descriptor *shape context morfológico* consiste en restringir la codificación de cada forma a su contorno.

Por lo tanto, la modificación del descriptor *shape context* es la siguiente:

Redefinamos $\mathcal{T} = \{t_1, ..., t_n\}$ como el conjunto de puntos muestreados de una línea de nivel L. Se calcula el *shape context* en cada punto t_i, $1 \leq i \leq n$, como la distribución de las posiciones relativas a t_i de los puntos $t_j \in \mathcal{T} \setminus \{t_i\}$.

De este modo el descriptor *shape context morfológico* conserva las virtudes del descriptor *shape context* tales como la propiedad invariante a traslacación, rotación, cambio de escala, y robustez frente a pequeñas alteraciones de la forma.

El conjunto de descriptores *shape context morfológico* tomados de una línea de nivel nos brinda un rico descriptor de la forma asociada. Por ende, la totalidad descriptores *shape context morfológico* calculados del conjunto de líneas de nivel seleccionadas de una imagen, es útil para encontrar individualmente correspondencias entre formas, o partes de ellas, con otra imagen descripta de la misma manera. Vale destacar que esta utilidad no es posible de obtener con el *shape context* tradicional.

En el libro de Cao et al. [CLM$^+$08], el descriptor LLD (*Level Line Descriptor*) fue diseñado para detectar que dos imágenes compartan *exactamente* la misma forma. El "invariante perceptual" es solamente introducida en la etapa de *matching*. Este no es nuestro objetivo. Nosotros queremos mantener el "invariante perceptual" intrínseco en el *shape context*, y ser capaces de detectar cuándo dos imágenes comparten dos formas *similares*, independientemente del algoritmo de *matching*.

4.2.2. *Matching a-contrario*

En esta sección, utilizamos el marco de trabajo *a-contrario* para el *matching* de formas codificadas mediante el descriptor *shape context morfológico*.

Con el objetivo de decidir cuando dos formas son semejantes o no, en este marco se intenta establecer un método para fijar un umbral de aceptación o rechazo y poder evaluar la calidad de la decisión.

El modelo *a-contrario* no utiliza información a priori, en cambio, construye un modelo estadístico empírico sobre el conjunto de formas posibles a comparar (base de datos). Está basado en el principio de Helmholtz que establece que cualquier detección obtenida en una situación de ruido debe ser considerada como irrelevante [DMM08].

Sean $\{SC_i | 1 \leq i \leq n\}$ y $\{SC'_j | 1 \leq j \leq m\}$ dos conjuntos de *shape context* de dos formas diferentes. Nuestro objetivo es ver si ambas son parecidas. Las distancias $d(SC_i, SC'_j)$ entre cada par de *shape context*, pueden ser vistas como observaciones de una variable aleatoria D que sigue un algún proceso aleatorio desconocido.

Lo que realmente necesitamos es realizar un test de hipótesis para cada par $d(SC_i, SC'_j)$, definido como:

$\mathcal{H}_1$*: $d(SC_i, SC'_j)$ es una observación causal, esto es, porque las formas son parecidas.*
$\mathcal{H}_0$*: $d(SC_i, SC'_j)$ es una observación producida por azar, esto es, porque la base de datos en grande.*

Por un lado, $P(D|\mathcal{H}_0)$ puede ser modelada relativamente fácil, incluso si el modelo no es perfectamente realístico. Por el otro, no es posible modelar $P(D|\mathcal{H}_1)$ debido a que no asumimos otra información sobre las formas aparte del conjunto observado de características (los *shape context*). Por lo tanto, no es posible realizar completamente el test de hipótesis ya que no podemos controlar los error de tipo II[4].

Sin embargo, controlar los errores de tipo I, esto es, el número de falsas correspondencias bajo la hipótesis $\mathcal{H}_0$, es suficiente para realizar una correcta respuesta a nuestro problema de decidir cuándo dos formas son semejantes o no. En otras palabras, al obtener probabilidades chicas bajo la hipótesis $\mathcal{H}_0$ es raro que las observaciones ocurran al azar, sino contrariamente, son causales.

La idea es aplicar este marco de estudio para el establecimiento de correspondencias entre *shape context*.

A continuación definimos la distancia entre dos *shape context* y estimamos la probabilidad de ocurrencia de una correspondencia bajo la hipótesis $\mathcal{H}_0$.

Formalmente, sea $\mathcal{F} = \{F^k | 1 \leq k \leq M\}$ una base de datos de M formas. Para cada forma $F^k \in \mathcal{F}$ tenemos un conjunto $\mathcal{T}^k = \{t_j^k | 1 \leq j \leq n_k\}$ de n_k puntos muestreados.

Sea $SC_{t_j^k}$ el *shape context* de t_j^k, $1 \leq j \leq n_k$, $1 \leq k \leq M$.

Luego, cada SC_{t_j} es dividido en C secciones independientes $SC_{t_j}^{(c)}$ con $1 \leq c \leq C$ (ver figura 4.7) debido a que se asume que los puntos están distribuidos uniformemente, con el objetivo de mejorar la calidad del *matching* [TAG+09].

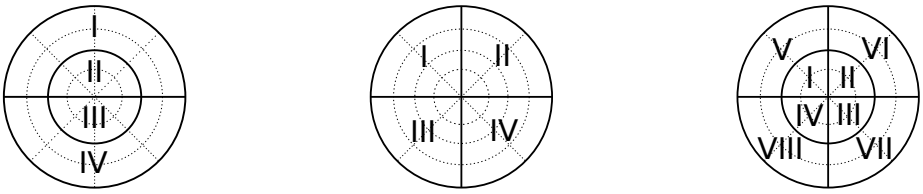


Figura 4.7: Diferentes formas de partir el *shape context*. Las líneas punteadas separan los casilleros y las lineas continuas separan los grupos de casilleros. La partición de la derecha fue la elegida para todos los experimentos realizados.

Sea Q la forma a buscar y q un punto de Q. Se define la semejanza entre *shape context* como:

$$d_j^k = \max_{1 \leq c \leq C} d_j^{k(c)}$$
$$d_j^{k(c)} = d(SC_q^{(c)}, SC_{t_j^k}^{(c)})$$

donde $d(\cdot, \cdot)$ es alguna distancia adecuada, en particular usamos la χ^2.

[4]Los errores de tipo I y II en estadística, definen respectivamente, la posibilidad de tomar un suceso verdadero como falso, o uno falso como verdadero.

El proceso de *matching* consiste en rechazar adecuadamente la hipótesis nula:
$\mathcal{H}_0$*: las distancias* $d_j^{k(c)}$ *son realizaciones de* C *variables aleatorias independientes* $D^{(c)}$, $1 \leq c \leq C$.

Definición 4.1. El par (q, t_j^k) es una **correspondencia ε-significativo** en la base de datos $\mathcal{F}$ si:

$$\mathrm{NFA}(q, t_j^k) \equiv \left(\sum_{k'=1}^{M} n_{k'} \right) \cdot \prod_{c=1}^{C} P(D^{(c)} \leq d_j^{k(c)} \mid \mathcal{H}_0) < \varepsilon$$

Este número es llamado el número de falsas alarmas (NFA) del par (q, t_j^k)

Esto nos provee de una simple regla para decidir cuando un par (q, t_j^k) es correspondido o no. Por un lado, esta es una clara ventaja sobre otros métodos de matching, ya que, tenemos una valoración individual para la calidad de cada correspondencia posible. Por otro lado, el umbral está tomado sobre la probabilidad en vez de directamente sobre las distancias. Definir el umbral directamente sobre las distancias d_j^k (o $d_j^{k(i)}$ para el caso) es muy difícil, ya que las distancias no tienen un sentido absoluto. Si todas las formas en la base de datos son parecidas, el umbral sería muy restrictivo; en cambio, si difieren bastante entre ellas, un umbral un poco más relajado sería suficiente.

Fijar umbrales sobre las probabilidades es más robusto y estable. Es más estable debido a que el mismo umbral sirve para distintas configuraciones de base de datos; y más robusto, por el hecho de que controlamos explícitamente las falsas detecciones. Como fue probado en el trabajo de Cao et al. [CLM+08], el número esperado de correspondencias ε-significativos en un conjunto aleatorio de correspondencias aleatorios es menor que ε.

El *matching* de formas, como fue descripto hasta ahora, sólo permite corresponder formas relativamente simples y semi-locales. Formas más complejas son representadas mediante grupos de formas que son geométricamente organizadas de la misma manera en ambas imágenes. Estos grupos son detectados como un paso agregado de *clustering*. En este trabajo, no abordaremos este paso agregado, ya que nuestro objetivo es enfocarnos en las etapas 1 y 2 del proceso de reconocimiento de formas. Sin embargo, proponemos utilizar el método RANSAC (ver sección 4.2.2) para elminar las correspondencias erroneas.

RANSAC

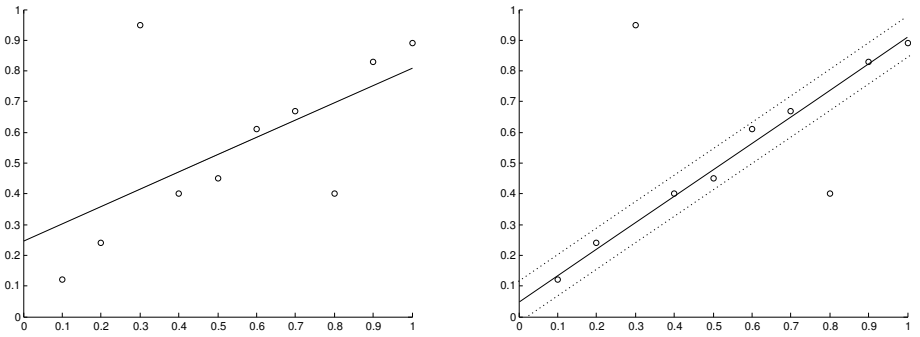
El método RANSAC (Random Sample Consensus) [FB81], es un estimador robusto, general y muy usado en la práctica con execelentes resultados. La estimación robusta se basa en que es tolerante a *outliers* y es general, ya que admite diferentes modelos y distancias en el algoritmo.

El objetivo del método es que a partir de un subconjunto minimal con la información suficiente para determinar un modelo (definido a priori), buscar el conjunto maximal de puntos consistentes con éste. Los puntos encontrados se denominan *inliers*.

RANSAC es un procedimiento que es opuesto a las técnicas comunes de refinación de correspondencias. En vez de usar toda la información posible para obtener una solución inicial y luego ir eliminando los puntos inválidos, RANSAC en principio usa un conjunto con la menor cantidad de información posible, y

luego, lo agranda gradualmente con información coherente. Este conjunto se denomina conjunto de consenso.

Por ejemplo, un modelo puede ser una recta y, la información provista, un conjunto de puntos en el plano; nuestro objetivo sería encontrar la recta que mejor aproxima el conjunto de puntos. Ver figura 4.8.



(a) Cuadrados Mínimos (b) RANSAC

Figura 4.8: En la figura a), vemos un ejemplo de ajuste de puntos mediante una recta usando el método de cuadrados mínimos. Este ajuste no es tan bueno como se puede ver en la figura b). Aquí la línea obtenida (línea continua) ajustó mejor el conjunto de puntos descartando los *outliers*. También podemos ver, con líneas punteadas la delimitación de la zona que el umbral de distancia fija donde debe caer un punto para ser considerado *inlier*.

Para nuestro caso, tenemos que el modelo es una transformación de similaridad, y la información provista es un conjunto de pares de puntos correspondientes $\{x_i \leftrightarrow x_i'\}$. Entonces, el procedimiento, al finalizar, nos determinará una transformación T y un conjunto de consenso $C_s \subseteq \{x_i \leftrightarrow x_i'\}$ que contendrá la mayor cantidad de correpondencias que cumplan que $T(x_i) = x_i'$, con un cierto margen de error. Se necesitan al menos dos pares de correspondencias para poder estimar una transformación de similaridad. Por lo tanto, se toman conjuntos aleatorios de 2 muestras.

Tanto la cantidad de correspondencias que deben ser coherentes con la transformación, como así también el margen de error tolerable, son parámetros controlables por el algoritmo y más adelante hablaremos sobre ellos.

Procedimiento

A continuación, presentamos los pasos del algoritmo de RANSAC.

Uno de los requisitos fundamentales es que el conjunto de datos de entrada S tenga más de las muestras necesarias para instanciar el modelo. Por ejemplo, si el modelo es una recta, se necesitan dos o más puntos; si el modelo es una transformación de similaridad, dos o más correpondencias, etc.

El algoritmo posee tres parámetros: el umbral de tolerancia t de distancia de un punto al modelo permite establecer cuando un punto se ajusta o no al modelo, para una distancia definida por éste. El segundo, la cantidad (o porcentaje) de *inliers* R, es un valor que establece una cota sobre la cantidad suficiente de datos de entrada que ajusta el modelo. El tercero, la cantidad de iteraciones máxima

N, necesaria para establecer una cota de iteración, al no alcanzar la cantidad de *inliers* pedida.

RANSAC

1: Sea S el conjunto de datos de entrada
2: **for** $i = 1$ **to** N **do**
3: Elegir aleatoriamente un subconjunto minimal de k muestras de S e instanciar el modelo con este subconjunto.
4: Determinar el conjunto de puntos S_i que están a una distancia menor que t del modelo. S_i es el conjunto de consenso de las muestras y define los *inliers* de S.
5: Si el tamaño de S_i (la cantidad de *inliers*) es mayor que un valor definido R, reestimar el modelo usando todos los puntos de S_i y terminar.
6: Si el tamaño de S_i es menor que R, ir al paso 3
7: **end for**
8: Elegir el mayor conjunto de consenso S_i, y reestimar el modelo usando todos los puntos del subconjunto S_i.

Algoritmo 4.1: Pseudo-código del algoritmo RANSAC

4.3. Resultados

En esta sección, mostramos el desempeño del *shape context morfológico* con tres ejemplos distintos. Para ellos, aplicamos las dos variantes SCM_{CS} y SCM_{FME} con el objetivo de observar las ventajas de trabajar con líneas de nivel, y el *matching* robusto de este nuevo proceso de reconocimiento de formas.

Los ejemplos muestran que la semi-localidad en el *shape context morfológico* es una característica clave para buscar formas en contextos donde hay muchas otras presentes: cuando hay formas muy similares que presentan pequeñas diferencias, cuando la imagen buscada no está presente o está rodeada por una larga cantidad de otras formas, y cuando distintas objetos ocluyen el mismo fondo. El *shape context morfológico* provee un nuevo enfoque para lidiar con situaciones en las que *shape context* no puede aplicarse.

En los dos algoritmos *a contrario*: el detector de contornos significativos y el *matching*, usar un $\varepsilon = 1$ hubiera sido suficiente; sin embargo, definimos en $\varepsilon = 10^{-10}$ para el primero y $\varepsilon = 10^{-3}$ para el segundo con la finalidad de mostrar el grado de confianza alcanzado sin afectar los resultados. Para el caso de los contornos significativos definimos el parámetro $\phi = 0{,}02$.

Para el algoritmo de selección de formas máximamente estables, se utilizaron los siguientes parámetros: $\Delta = 10$, $minSim = 0{,}7$, $maxVar = 3$, ya que fueron los que mostraron mejores resultados a lo largo del trabajo.

4.3.1. *Matching* bajo cambios de perspectiva

En las figuras 4.9 a 4.11 veremos que las bondades del descriptor *shape context* son potenciadas con los detectores de características usados: contornos significativos y formas máximamente estables; que como hemos establecido, otorgan una buena representanción de los objetos más salientes de una escena.

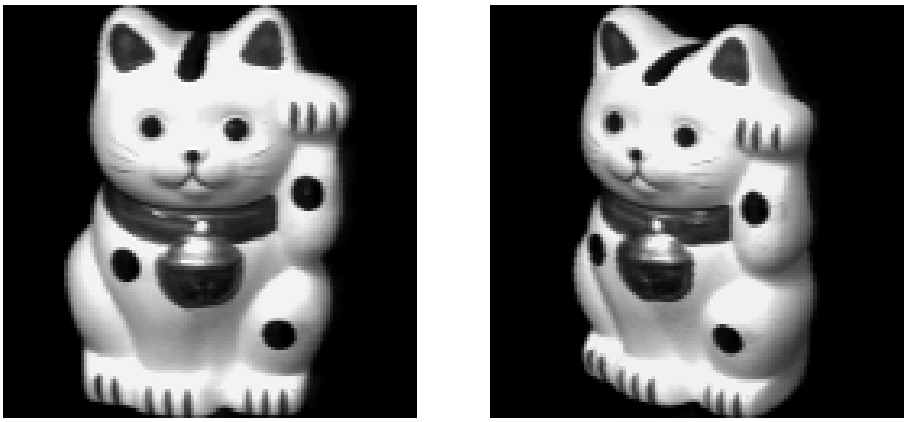
En la 4.9, podemos ver dos imágenes de una misma escena las cuales están relacionadas mediante una transformación proyectiva de la cámara, y por ende, las formas presentes también estarán relacionadas mediante una transformación. Debido a la adquisión de la imágenes la pose del objeto no cambia [NNM96], es decir, podemos saber que están relacionadas mediante una transformación afín. Esta inforamción no es usada en los ejemplos presentados a continuación, sin embargo, en el *shape context morfológico* utilizamos el procedimiento RANSAC para eliminar las correspondencias que no puedan ser relacionadas mediante una tranformación de similaridad.

En las figuras 4.10a y 4.11a, mostramos las correspondencias obtenidas de haber aplicado el proceso SCM_{FME} y SCM_{CS} respectivamente. Para poder apreciar específicamente en qué posiciones se corresponden los puntos en las imágenes, superponemos las correspondencias sobre las imágenes y sus líneas de nivel seleccionadas.

Para la figura 4.10a se encontraron 32 correspondencias entre las dos imágenes. En las imágenes de las figuras 4.9a y 4.9b se detectaron 33 y 31 formas máximamente estables, respectivamente. Pueden apreciarse sus contornos en la figura 4.10b.

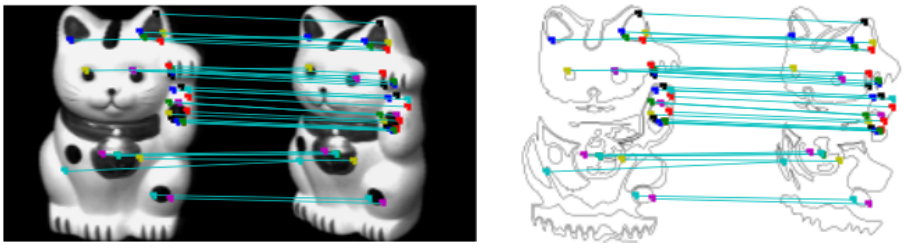
En la figura 4.11a se encontraron 23 correspondencias entre las dos imágenes. En las imágenes de las figuras 4.9a y 4.9b se detectaron 24 y 23 contornos significativos, respectivamente. Éstos pueden apreciarse en la figura 4.11b. La totalidad de las correspondencias detectadas mediante SCM_{CS} pueden verse en la figura 4.12. Aquí, se encontraron 153 correspondencias que mediante el procedimiento RANSAC (ver sección 4.2.2) se eliminan las erróneas (y algunas acertadas también).

Como pudimos ver en los ejemplos, las formas máximamente estables y los contornos significativos son detectados bajo situaciones de traslación y rotación de la formas de la imagen, permitiéndonos describirlas con un descriptor que posee estas propiedades invariantes; y luego, mediante un *matching* robusto y estable nos permite ponerlas en correspondencia adecuadamente. Los ejemplos aquí presentados muestran que este procedimiento puede ser aplicado en el apareamiento estéreo entre cámaras, útil en el área de visión en robótica.



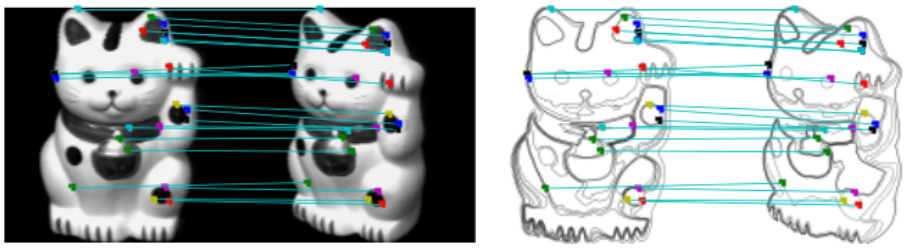
(a) (b)

Figura 4.9: (a) Imagen original. (b) Imagen obtenida luego de una rotación de la cámara 25 grados en sentido horario. Imágenes extraídas de la base de datos COIL-20 [NNM96].



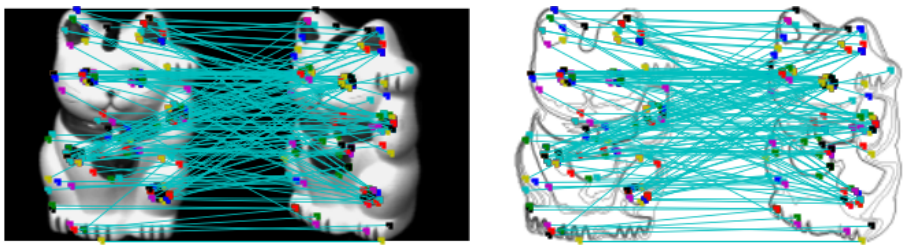
(a) (b)

Figura 4.10: Método SCM_{FME}. En la figura (a), se muestran las correspondencias mediante puntos de colores y unidas por líneas rectas entre las dos imágenes de la figura 4.9. En la figura (b), se superponen las correspondencias sobre los contornos de las formas máximamente estables. Existen 32 pares de correspondencias entre las dos imágenes.



(a) (b)

Figura 4.11: Método SCM_{CS}. En la figura (a), se muestran las correspondencias mediante puntos de colores y unidas por líneas rectas entre las dos imágenes de la figura 4.9. En la figura (b), se superponen las correspondencias sobre los contornos significativos. Existen 23 pares de correspondencias entre las dos imágenes.



(a) (b)

Figura 4.12: Cantidad total de correspondencias (152 pares) entre las imágenes de la figura 4.9 luego del proceso de *matching* utilizando SCM_{CS}. En la figura (a), se muestran las correspondencias sobre las imágenes y en (b), sobre los contornos significativos.

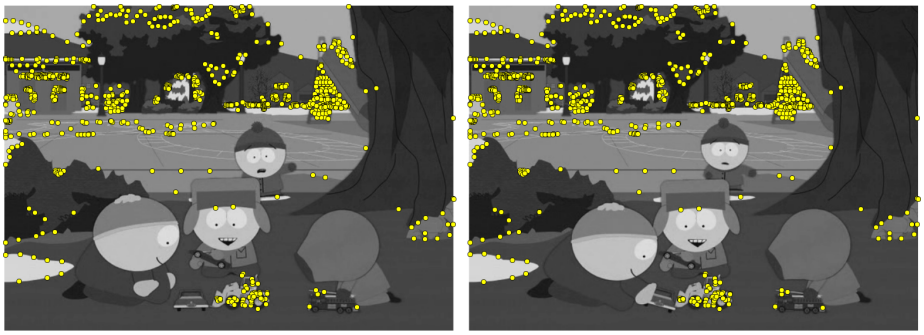
4.3.2. Correspondencias entre cuadros de video

En este ejemplo, aplicamos el proceso a una secuencia de video y analizamos las correspondencias significativas entre dos cuadros con objetos planos y sin texturas, compuestos solamente por contornos. En la figura 4.13, las correspondencias significativas son graficadas sobre las imágenes. Mediante círculos amarillos se representan los centros de los *shape context morfológico* (se decidió no incluir las rectas entre las correspondencias en favor de la claridad del ejemplo) obtenidos del proceso de *matching*. Los pares homólogos de correspondencias se encuentran en la misma coordenada en las dos imágenes.

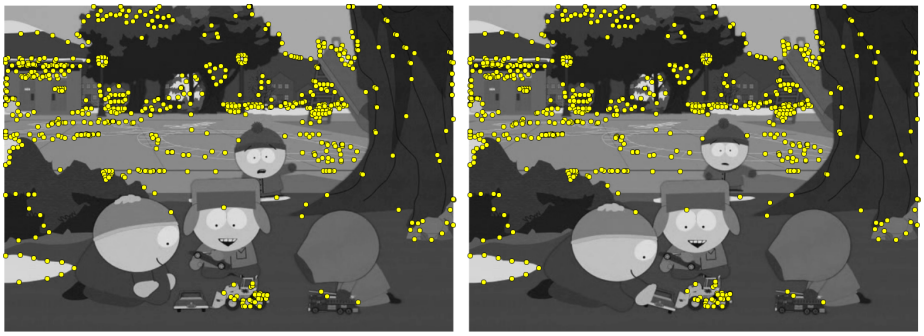
En la figura 4.13c, los dos cuadros fueron superpuestos con transparencia para mostrar las zonas donde hay movimientos. Nótese que en las figuras 4.13a y 4.13b no hay correspondencias en esas áreas.

Algo para destacar de este ejemplo, es la cantidad de coincidencias en las correspondencias obtenidas por los dos métodos SCM_{FME} y SCM_{CS}. Sin bien

hay más correspondencias en SCM_{FME} que en SCM_{CS}, esta diferenencia es mínima: 27 en un total de $\approx$ 888 correspondencias en promedio, i.e. menos del 3.1 %. Este hecho refuerza la idea enunciada en la sección 3.4 –en donde se comparan los dos métodos de extracción de características: formas máximamente estables y contornos significativos– que establece la gran similaridad en la detección de líneas de nivel con buen contraste.



(a) SCM_{FME}



(b) SCM_{CS}

(c)

Figura 4.13: Correspondencias entre cuadros coherentes con una transformación de similaridad. Los puntos homólogos en las imágenes se representan con círculos amarillos. (a) SCM_{FME}: se econtraron en total 901 correspondencias. (b) SCM_{CS}: se econtraron en total 874 correspondencias. (c) Cuadros superpuestos.

4.3.3. Búsqueda en video por contenido

En esta sección se analizó la aplicación de búsqueda de un logo de control parental (ver figura 4.14a) a una secuencia de video de 8434 cuadros. Éste aparece y desaparece difuminándose en tres ocasiones y, en la segunda y tercera aparición, es más pequeño. En la figura 4.14b vemos un cuadro del video y en la figura **??**, se muestra una selección de ellos. En todos los ejemplo presentes en esta sección se utilizó el método SCM_{CS} con la excepción del expuesto en la figura 4.19 como se detalla.

(a) (b)

Figura 4.14: (a) logo de control parental. (b) cuadro de video.

Figura 4.15: Cuadros 0, 5, 43, 55, 74 y 76 correspondientes a la primera aparición del logo de control parental en un video de 8434 cuadros.

En la figura 4.16, graficamos la cantidad de correspondencias por cuadro de

video. El logo aparece en tres intervalos ([0, 76], [2694, 2772] y [4891, 4969]), los cuales coinciden con los tres picos pronunciados en el gráfico. Estos tres picos son claramente mayores que las correspondencias espúreas en el resto del video. El segundo y tercer pico son más chicos que el primero, ya que en esos intervalos el logo aparece a un 66 % de su tamaño original. Vale aclarar que esto ocurre sin ningún tipo de procesamiento multiescala.

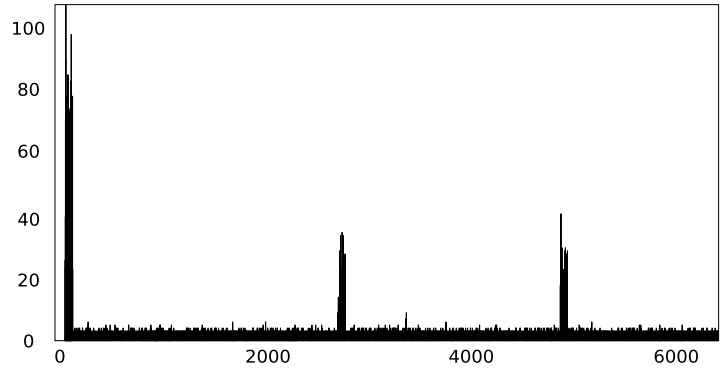
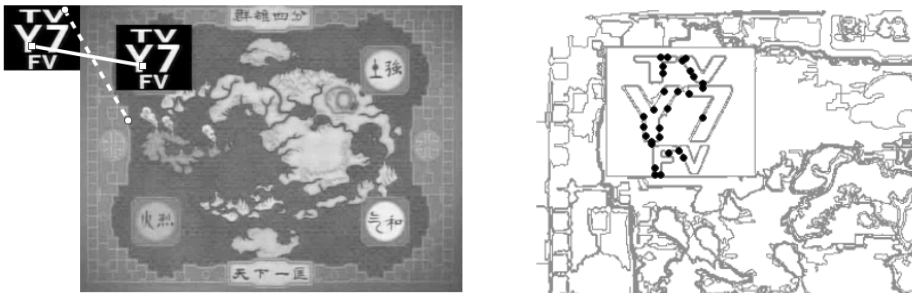


Figura 4.16: Cantidad de correspondencias con la imagen buscada en función del número de cuadro del video.

En la figura 4.17a, se detalla la mejor y peor correspondencia obtenida enter el logo y el cuadro número 43 de la secuencia. La mejor correspondencia (la correcta) tiene un NFA de $2{,}45 \cdot 10^{-9}$, y la peor correspondencia tiene un NFA de $9{,}99 \cdot 10^{-3}$. Por lo tanto, con un $\varepsilon = 10^{-4}$, todas las correspondencias obtenidas son correctas. Además, en la figura 4.17b, se muestran sobre los contornos significativos del mismo cuadro las correspondencias obtenidas con el logo.

Para el mismo experimento que en la figura 4.17a, si se utiliza el método original de *shape context* [BMP02], se obtienen solamenten 3 correspondencias en vez de las 29 obtenidas usando el *shape context morfológico* (figura 4.17b). Todas las correspondencia del *shape context morfológico* son correctas, mientras que las de *shape context* no: el enfoque global de *shape context* es incapaz de corresponder formas semi-localmente.

En la figura 4.19, vemos las correspondencias obtenidas usando el método SCM_{FME} para el cuadro número 5 de la secuencia de video. Se encontraron un total de 65 correspondencias, y fuera de la zona detallada no existen correspondencias. Este ejemplo nos induce a concluir que el método SCM_{FME} también es adecuado para aplicaciones de búsqueda en video por contenido.



(a) (b)

Figura 4.17: (a) Mejor correspondencia (línea continua) y peor correspondencia (línea punteada) para el cuadro número 43 del video. (b) Detalle del área donde está el logo en el cuadro número 43. Las correspondencias se ven como puntos negros.

Figura 4.18: Ubicación de las correspondencias (en colores) con el logo sobre el cuadro número 55 de la secuencia.

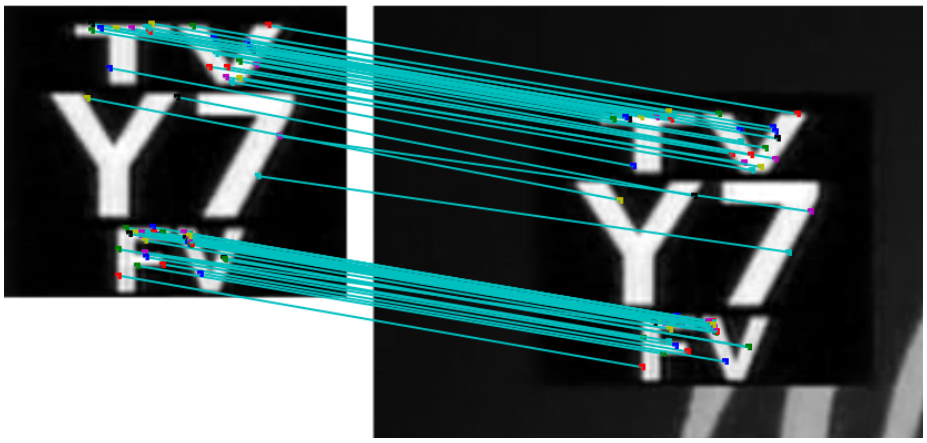


Figura 4.19: Correspondencias entre el logo de control parental y el cuadro número 5 usando el método SCM_{FME}. No existen otras correspondecias fuera del área detallada.

4.3.4. Robustez frente a oclusiones

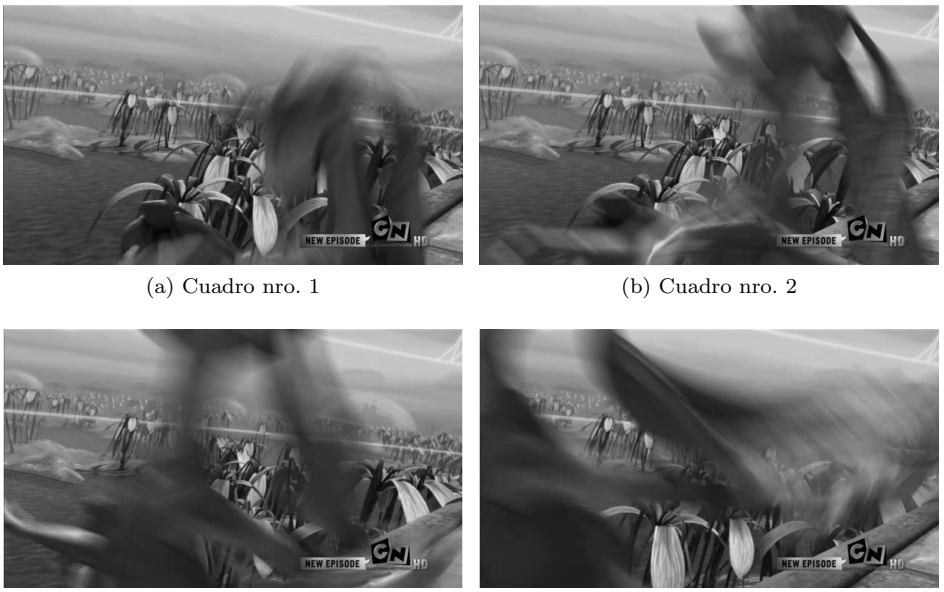
Este ejemplo está muy relacionado con el visto en la sección 4.3.2. La diferencia es que aquí sí hay textura y, que en la secuencia de video (ver figura 4.20), se ve un personaje moviéndose no rígidamente[5] en el primer plano.

Al igual que en la figura 4.13, los círculos amarillos representan los centros de los descriptores *shape context* obtenidos del proceso de *matching*. Análogamente, se realizó el proceso de reconocimento de formas tanto para SCM_{FME} (figura 4.21a) como para SCM_{CS} (figura 4.21b).

En este ejemplo, la diferencia en la cantidad de correspondencias obtenidas por los dos métodos es apreciable. Esto se debe a que el método SCM_{CS} encuentra en la imagen de la figura 4.20c 118 contornos significativos y en la imagen de la figura 4.20d 77, ocasiando que la cantidad de descriptores puestos en correspondencias por el métodos, sean muchos menos para la figura 4.20d, perjudicando así la etapa de *matching*.

En la figura 4.21, se muestran las correspondencias entre los cuadros 3 y 4 de la secuencia. Se puede apreciar cómo se pone en correspondencia el logo de canal entre los cuadros, y también algunos objetos estáticos del fondo, concluyendo que la consistencia del *matching* se conserva.

[5] Un movimiento no rígido es aquel no puede ser caracterizado mediante una transformación afín o proyectiva.



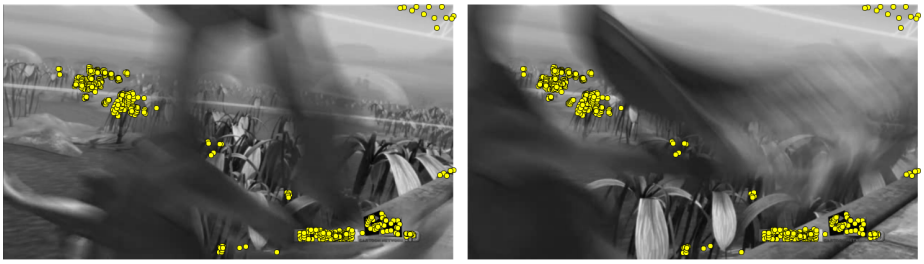
(a) Cuadro nro. 1

(b) Cuadro nro. 2

(c) Cuadro nro. 3

(d) Cuadro nro. 4

Figura 4.20: Secuencia de video con un personaje moviéndose no rígidamente en primer plano. El logo del canal está ubicado abajo a la derecha.



(a) SCM_{FME}

(b) SCM_{CS}

Figura 4.21: Correspondencias entre los cuadros 3 y 4 de la secuencia en la figura 4.20, coherentes con una transformación de similaridad. Los puntos homólogos en las imágenes se representan con círculos amarillos. (a) SCM_{FME}: se econtraron en total 551 correspondencias. (b) SCM_{CS}: se econtraron en total 193 correspondencias.

Capítulo 5

Conclusiones y trabajo futuro

En este trabajo, además de las formas máximamente estables, hemos considerado el criterio de selección de curvas denominado contornos significativos, el cual también tiene por objetivo detectar objetos contrastados en la imagen. Este detector tiene mucho puntos en común con las formas máximamente estables y está muy bien condicionado en la detección de forma perceptualmente interesantes. Como hemos visto a lo largo del trabajo, si bien los contornos de las formas máximamente estables y los contornos significativos no siempre coinciden, ambos están muy bien adaptados para seleccionar un conjunto de características saliente, repetible y robusto de la imagen de entrada.

Dada la característica del enfoque semi-local del proceso de reconocimiento de forrmas presentado, el shape context morfológico, hemos observado que no es indispensable un proceso de suavizado de curvas posterior a la detección de las mismas, ya que, gracias a las propiedades del shape context, éste no tiene en cuenta pequeñas perturbaciones de la forma y no pondera los outliers presentes. Adicionalmente, en la etapa de matching se permiten filtrar las correspondencias aberrantes para conserva un conjunto refinado de correspondencias.

Como trabajo futuro, podemos mencionar varias modificiaciones posibles al descriptor shape context y al proceso de reconocimiento de formas presentado. Por ejemplo, podemos mencionar la inclusión de la propiedad invariante a rotación, considerar otros tipos de distancias para la elaboración de las distribuciones de los shape context, definir otros criterios de comparación entre ellos, entre otras.

También, una rama de exploración consiste en calcular el área de las formas de una imagen interpolada bilinealmente en base a su contorno para poder obtener un cálculo de variación de área más ajustado en la etapa de detección de formas máximamente estables. Otra vertiente completamente opuesta sería utilizar un tipo de descriptor distinto a shape context que considere las formas como un todo y no solamente por su contorno.

Otro punto el cual posee instancias de mejora, es la etapa de matching en la cual se puede incorporar la comparación multigrupo al considerar la agrupación de varias formas como una nueva forma.

Apéndice A

Árboles

A.1. Definiciones y propiedades

En la presente sección, daremos las definiciones y nociones básicas sobre árboles, necesarias para permitir la comprensión adecuada de los términos usados a lo largo del trabajo[1].

Definición A.1. Un **árbol** se define como un grafo $G = (V, E)$ conexo y sin circuitos. El conjunto V representa los vértices o nodos del grafo, y E es el conjunto de ejes o aristas.

Si el grafo G es dirigido entonces el árbol también es dirigido.

Definición A.2. Un **camino** desde un vértice v a otro v_n es una secuencia alternada

$$C = \langle v_0, e_1, v_1, e_2, \ \ldots, v_{n-1}, e_n, v_n \rangle$$

de vértices y ejes, sin repeticiones, tal que $e_i = \{v_{i-1}, v_i\}$, para $i = 1, \ \ldots, n$.

La **longitud** del camino es la cantidad de ejes que posee.

Nota. Por conveniencia de notación, se definirá el camino C indicando solamente la secuencia de vértices.

Definición A.3. Un **árbol con raíz** es un árbol dirigido que tiene un vértice distinguido llamado *raíz*, tal que para cada otro vértice v del árbol existe un único camino dirigido desde la raíz a v.

De aquí en adelante, cuando hablemos de árboles, nos referiremos a **árboles con raíz**.

Definición A.4. El **nivel** o profundidad de un vértice v en un árbol es la distancia desde la raíz al vértice v. La raíz tiene altura 0.

Definición A.5. La **altura** de un árbol es la longitud del camino más largo desde la raíz.

Definición A.6. Si un vértice v precede inmediatamente a un vértice w en el camino desde la raíz hasta w, entonces v es el **padre** de w, y w es el **hijo** de v.

[1] Para una lectura en profundidad consultar [GY05].

Definición A.7. Una **hoja** es un vértice que no tiene hijos.

Definición A.8. Un **sub-árbol** T de R es un árbol que tiene todos sus vértices y ejes en R.

Definición A.9. Una **rama** de un árbol es un camino desde la raíz a una hoja.

Definición A.10. Una **sub-rama** de un árbol es un camino incluído en una rama del árbol.

Nota. Mientras no queden ambigüedades, haremos referencia a una sub-rama llamándola "rama".

Definición A.11. Un **recorrido en profundidad** o en **preorden** sobre un árbol, se puede definir recursivamente como:

- Visitar la raíz.
- Recorrer en profundidad cada sub-árbol hijo.

A.2. Estructura de representación

A continuación presentamos la estructura de representación de los árboles utilizados a lo largo del trabajo.

A un árbol $\mathcal{T}$ lo representaremos mediante una estructura enlazada de nodos, tal que, conociendo el nodo raíz, tendremos acceso a todo el árbol.

Cada nodo es representado por la siguiente estructura:

Node
- *children* ▷ *lista de nodos hijo.*
- *parent* ▷ *referencia al nodo padre.*
- *data* ▷ *información del nodo.*

La raíz del árbol se representa como el nodo que tiene *parent* definido como la constante NULL. Las hojas del árbol son nodos que tienen la lista *children* vacía.

Luego, para que una estructura enlazada de nodos represente un árbol, debe cumplir las siguiente propiedades invariantes:

- Debe existir un único nodo raíz.
- Todos los nodos de la lista *children* deben apuntar al mismo padre.

Además de la lista de hijos y de la referencia al padre, cada nodo consta con un atributo *data*. Éste será el atributo de información del nodo n, accesible como: $n.data$. Dependiendo del contexto de trabajo, puede haber más de un atributo de información, y también, el nombre del atributo puede variar. Por ejemplo, para el árbol de componente, cada nodo n tiene un atributo *area* y otro *level*, accesibles como $n.area$ y $n.level$.

La estructura de representación de la lista *children* es sencillamente una lista simplemente encadenada con dos operaciones básicas: INSERT y ERASE, que agregan y elminan un nodo a lista, respectivamente. También, la lista cuenta con un método que permite iterarla nodo por nodo.

```
RemoveNode(Node n)
1  parent ← n.parent
2  parent.children.Erase(n)
3  for each n_i ∈ n.children
4        parent.AddChild(n_i)
```

Algoritmo A.1: Remoción de un nodo del árbol asociado al nodo n.

```
AddChild(Node n, Node child)
1  child.parent ← n
2  n.children.Insert(child)
```

Algoritmo A.2: Agregado de un nodo aal árbol asociado a n.

Apéndice B

Conjuntos disjuntos

El problema de los conjuntos disjuntos consiste en mantener una colección $\mathcal{Q}$ de subconjuntos disjuntos de un conjunto D. Es decir, dado $\mathcal{Q} = \{Q_1, \ldots, Q_n\}$ se cumple que $Q_i \cap Q_j = \emptyset$ con $1 \leq i \neq j \leq n$

Para poder identificarlos, cada conjunto Q_i de $\mathcal{Q}$ está representado por un único elemento de Q_i denominado *elemento canónico*. Por lo tanto, dos elementos que poseen el mismo representante están en el mismo conjunto.

Las operaciones principales que realizaremos sobre los conjuntos disjuntos son las siguientes:

- FIND(x): devuelve el elemento canónico del conjunto de $\mathcal{Q}$ que contiene a x. Resulta útil para saber si dos elementos están en el mismo conjunto.
- LINK(x, y): llamamos X e Y a los conjuntos representados por los elementos canónicos x e y respectivamente. Esta operación elimina X e Y de $\mathcal{Q}$ y luego agrega el nuevo conjunto $Z = X \cup Y$.
- MAKESET(x): agrega el conjunto $\{x\}$ a $\mathcal{Q}$ asumiendo que $\{x\}$ no está ya en $\mathcal{Q}$.

A la operación LINK, también se la puede nombrar como *Union*, que es la denominación dada por Tarjan [TVL84] en su procedimiento *union-find*.

La estructura de representación adoptada para la colección de conjuntos disjuntos es un conjunto de árboles también denominado *bosque*. Cada árbol del bosque representa un conjunto de la colección, donde los nodos del árbol representan elementos del conjunto.

Cada nodo x del árbol tiene asociado un padre, denominado Par(x), que es otro nodo del árbol; excepto la raíz, que es el elemento canónico del conjunto, y, por convención, se asocia como padre a sí mismo. Además, todo nodo tiene asociado un *ranking*, denotado como Rnk(x), que es un valor que da una medida de la profundidad del sub-árbol con raíz x. Si no se utilizara la técnica *path-compression* (explicada a continuación) sería exactamente la altura.

El procedimiento *union-find* [TVL84] implementa dos heurísticas para reducir el orden de complejidad de las operaciones: *path-compression* y *union-by-rank*. La primera se incluye en el algoritmo B.1 (Find) y, la segunda, en el algoritmo B.2 (Link).

- ***path-compression*:** el objetivo de esta técnica es reducir a largo plazo el costo de la operación FIND. Consiste en que, luego de encontrar la raíz r del árbol que contiene a x, se define como padre a r de cada elemento y del camino que va de x a r. Ver figura B.1.
- ***union-by-rank*:** consiste en elegir siempre el representante de mayor ranking, en el momento de efectuar la operación LINK. Ver figuras B.2 y B.3. Esta técnica evita que se creen árboles degenerados, es decir, con ramas extremadamente largas.

Ambas heurísticas contribuyen a generar árboles de poca profundidad, para reducir el órden de complejidad de las operaciones FIND y LINK.

```
Find(elemento x)
1  if Par(x) = x
2        Par(x) ← Find(Par(x))
3  return Par(x)
```

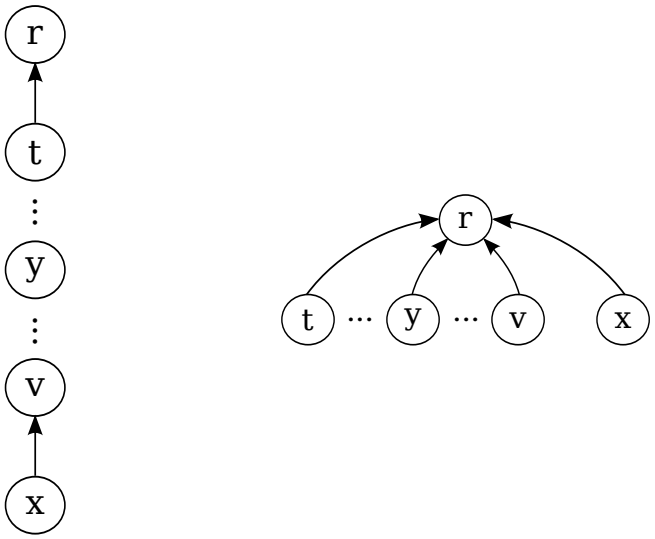
Algoritmo B.1: Procedimento que devuelve el representante del conjunto al cual pertence el elemento x.

```
Link(elemento x, elemento y)
1  if Rnk(x) > Rnk(y)
2        exchange(x, y)
3  if Rnk(x) = Rnk(y)
4        Rnk(y) ← Rnk(y) + 1
5  Par(x) ← y
6  return y
```

Algoritmo B.2: Función Link(x, y) que une dos conjuntos disjuntos de la colección representados por los elementos x e y. En este procedimiento se implementa la heurística *union-by-rank*

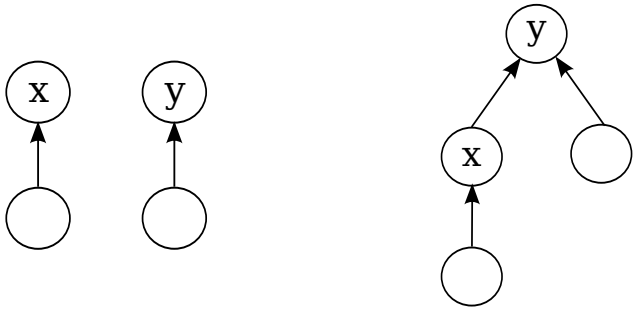
```
MakeSet(elemento x)
1  Par(x) ← x
2  Rnk(x) ← 0
```

Algoritmo B.3: Procedimiento que agrega un nuevo conjunto disjunto $\{x\}$ a la colección asumiendo que no pertenece a ésta.



(a) (b)

Figura B.1: Esquema de la técnica *path-compression* implementada en la función FIND(x) sobre una rama del árbol de un conjunto disjunto. La rama de la figura (a) se obtuvo al ascender por el árbol desde x hasta la raíz r. Al aplicar *path-compression* se ve en la figura (b) como se aplana la rama del árbol de la figura (a) para que en la próxima llamada a FIND el camino desde x hasta la raíz r sea menor.



(a) (b)

Figura B.2: Esquema de la técnica *union-by-rank* implementada en la operación LINK(x, y) en el caso que Rnk(x) = Rnk(y).

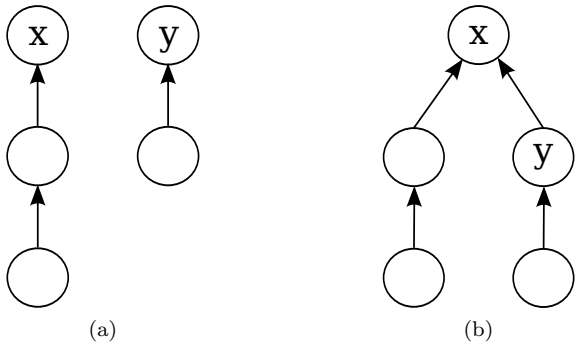


Figura B.3: Esquema de la técnica *union-by-rank* implementada en la operación Link(x, y) en el caso que Rnk$(x) >$ Rnk(y).

B.1. Complejidad

Puede probarse, siguiendo un análisis amortizado de la complejidad, que el orden de complejidad de la combinación de las heurísticas *path-compression* y *union-by-rank* es $O(m \times \alpha(n))$ para m operaciones del tipo *union-find* sobre n elementos [TVL84]. Allí la función α es la inversa de la función diagonal de Ackerman, que es una función que crece muy lentamente y se asimila a una función constante para casos prácticos: $\alpha(n)$ tiene un valor aproximado de 4 para entradas de orden de magnitud de 10^{80} (cercano a la cantidad de átomos en el universo).

Bibliografía

[Ade93] E. H. Adelson. Perceptual organization and the judgment of brightness. *Science*, 262(5142):2042–2044, December 1993.

[Att54] Fred Attneave. Some informational aspects of visual perception. *Psychological Review*, 61(3):183–193, May 1954.

[Bha43] A Bhattacharyya. On a measure of divergence between two statistical populations defined by probability distributions. *Bulletin of the Calcutta Mathematical Society*, (35):109, 1943.

[BMP02] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(4):509–522, 2002.

[Can86] J. Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, November 1986.

[CCM99] Vicent Caselles, Bartomeu Coll, and Jean-Michel Morel. Topographic maps and local contrast changes in natural images. *International Journal of Computer Vision*, 33(1):5–27, 1999.

[CH67] T. Cover and P. Hart. Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 13(1):21–27, 1967.

[CK98] J.L. Cox and D.B. Karron. Digital morse theory. 1998.

[CLM+08] F. Cao, J. L. Lisani, J. M. Morel, P. Musé, and F. Sur. *A Theory of Shape Identification*, volume 1948 of *Lecture Notes in Mathematics*. Springer, 2008.

[CLRS03] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. McGraw-Hill Science / Engineering / Math, 2nd edition, December 2003.

[CMS05] F. Cao, P. Musé, and F. Sur. Extracting meaningful curves from images. *J. Math. Imaging Vis.*, 22(2-3):159–181, 2005.

[DB06] Michael Donoser and Horst Bischof. Efficient maximally stable extremal region (mser) tracking. In *CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 553–560, Washington, DC, USA, 2006. IEEE Computer Society.

[DMM01] A. Desolneux, L. Moisan, and J. M. Morel. Edge detection by helmholtz principle. *Journal of Mathematical Imaging and Vision*, 14(3):271–284, 2001.

[DMM08] A. Desolneux, L. Moisan, and J. M. Morel. *From Gestalt Theory to Image Analysis.*, volume 34. Springer-Verlag, 2008.

[FB81] M. Fischler and R. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.

[for07] *Shape Descriptors for Maximally Stable Extremal Regions*, 2007.

[GY05] Jonathan L. Gross and Jay Yellen. *Graph Theory and Its Applications, Second Edition (Discrete Mathematics and Its Applications)*. Chapman & Hall/CRC, September 2005.

[Kan79] G. Kanizsa. *Organization in Vision: Essays on Gestalt Perception*. Praeger, 1979.

[KH90] A. Khotanzad and Y. H. Hong. Invariant image recognition by zernike moments. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 12(5):489–497, 1990.

[LJ07] Haibin Ling and David W. Jacobs. Shape classification using the inner-distance. *IEEE Trans. Pattern Anal. Mach. Intell*, 29:286–299, 2007.

[LMR01] J. L. Lisani, P. Monasse, and L. Rudin. Fast shape extraction and applications. Technical Report 16, CMLA, 2001.

[Mat75] G. Matheron. *Random Sets and Integral Geometry*. John Wiley & Sons, NY, USA, 1975.

[MCMP02] J. Matas, O. Chum, U. Martin, and T. Pajdla. Robust wide baseline stereo from maximally stable extremal regions. In *Proceedings of the British Machine Vision Conference*, volume 1, pages 384–393, London, 2002.

[MG00] P. Monasse and F. Guichard. Fast computation of a contrast invariant image representation. *Image Processing, IEEE Transactions on*, 9(5):860–872, May 2000.

[Mon00] Pascal Monasse. *Morphological representation of digital images and application to registration*. PhD thesis, Université Paris IX-Dauphine, June 2000.

[MZFC08] Enric Meinhardt, Ernesto Zacur, Alejandro Frangi, and Vicent Caselles. 3d edge detection by selection of level surface patches. *Journal of Mathematical Imaging and Vision*, 2008.

[NC06] L. Najman and M. Couprie. Building the component tree in quasi-linear time. *Image Processing, IEEE Transactions on*, 15(11):3531–3539, 2006.

[NNM96] Sameer Nene, Shree K. Nayar, and Hiroshi Murase. Columbia object image library (coil-20). Technical report, Department of Computer Science, Columbia University, New York, EE. UU., 1996.

[PS98] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization : Algorithms and Complexity*. Dover Publications, July 1998.

[RC96] Srinivasa B. Reddy and B. N. Chatterji. An fft-based technique for translation, rotation, and scale-invariant image registration. *IEEE Transactions on Image Processing*, 5(8):1266–1271, August 1996.

[RGT97] Yossi Rubner, Leonidas Guibas, and Carlo Tomasi. The earth mover‘s distance, multi-dimensional scaling, and color-based image retrieval. In *in Proceedings of the ARPA Image Understanding Workshop*, pages 661–668, 1997.

[SB91] Michael J. Swain and Dana H. Ballard. Color indexing. *International Journal of Computer Vision*, 7:11–32, 1991.

[Ser83] J. Serra. *Image Analysis and Mathematical Morphology*. Academic Press, Inc, Orlando, FL, USA, 1983.

[Son07] Yuqing Song. A topdown algorithm for computation of level line trees. *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society*, 16(8):2107–2116, August 2007.

[SZ03] Josef Sivic and Andrew Zisserman. Video google: A text retrieval approach to object matching in videos. In *ICCV '03: Proceedings of the Ninth IEEE International Conference on Computer Vision*, Washington, DC, USA, 2003. IEEE Computer Society.

[TAG+09] M. Tepper, D. Acevedo, N. Goussies, J. Jacobo, and M. Mejail. A decision step for shape context matching. *International Conference on Image Processing*, 2009. Submitted.

[TGFP+09] M. Tepper, F. Gómez Fernández, Musé P., Almansa A., and M. Mejail. Morphological shape context: Semi-locality and robust matching in shape recognition. *14th Iberoamerican Congress on Pattern Recognition. CIARP*, 2009.

[Tho17] D‘Arcy Wentworth Thompson. *On Growth and Form*. Cambridge Univ. Press, 1917.

[TVL84] R. Tarjan and J. Van Leeuwen. Worst-case analysis of set union algorithms. *Journal of the ACM*, 31(2):245–281, April 1984.

[Wer38] M. Wertheimer. *Laws of organization in perceptual forms*, pages 71–88. Routledge and Kegan Paul, 1938.