Tesis de Licenciatura en Ciencias de la Computación

Segmentación de imágenes texturadas

Alumnos

Mariano Andrés Tribuj L.U. 129/96
Ariel David Waisbaum L.U. 135/96

Directores

Dra. Marta Mejail
Dr. Julio Jacobo Berllés

Agosto 2006

Resumen

El proceso de segmentación de imágenes texturadas, consiste en realizar una partición de la imagen de entrada, en un conjunto de regiones disjuntas, haciendo que cada región sea homogénea con respecto a la característica de la textura. Muchos trabajos se han realizado sobre distintos métodos para solucionar el problema, los cuales podrían resumirse principalmente en dos categorías:

1. Los métodos que consideran que los valores de cada uno de los pixels de la imagen están regidos por una función de probabilidad, en cuyo caso el problema consiste en estimar los parámetros que la definen, para con ellos obtener la segmentación.
2. Los métodos basados en aplicar un banco de filtros a la imagen, para obtener de cada uno de los filtros un valor por pixel para el vector de características que permiten luego realizar la segmentación

Los métodos de segmentación pueden ser supervisados o sin supervisión, dependiendo de los requerimientos de información a priori de la imagen o de las texturas que la componen. En este trabajo consideramos la segmentación basada en las texturas, sin tener en cuenta la información de color que pudiera incluir la imagen.

El objetivo es evaluar e implementar distintos métodos que representan a cada uno de los enfoques; implementando también mejoras a los mismos, para incrementar la velocidad de procesamiento, como así también obtener información extra sobre los resultados obtenidos, que los métodos evaluados no brindan.

La segmentación de imágenes texturadas posee una gran cantidad de usos en la vida real, los cuales todos tienen como fin común la comprensión del mundo que nos rodea. Algunos ejemplos de aplicaciones pueden ser: reconocimiento automático de regiones en imágenes aéreas, detección de objetos en visión en robótica, reconocimiento de distintas formaciones celulares en análisis histopatológico, y otros.

En esta tesis se analizan e implementan métodos de segmentación supervisados utilizando campos aleatorios de Markov, y bancos de filtros de Gabor. Para el segundo caso, realizamos una implementación que permite el procesamiento distribuido. El tercer método implementado une ambos conceptos, en un algoritmo sin supervisión. Para concluir comparamos los resultados obtenidos en los distintos algoritmos.

Abstract

The textured image segmentation process consists on producing a partition of the input image into a set of separated regions so that each region represents a single texture of the original image.

A lot of work has been done covering different methods and approaches to solve this problem. These approaches can be summarized in two different categories:

1. Methods that consider each pixel value as governed by a certain probability function. In this case the problem consists in estimating the parameters that define this function and then use them in the segmentation process.

2. Methods based on filter banks. The filters are applied to extract features from the original image. These features are the key of the segmentation process.

The segmentation methods can be supervised or unsupervised, depending on the a priori information requirements for the image or the involved textures.

In this work, we focus on the segmentation process based on the textures, regardless of the color information that the images might include.

The main goal is to evaluate and implement different methods representing each of the approaches; making improvements to increment the processing speed and to obtain additional information about the results.

The textured image segmentation process has a large variety of uses in real life. All of them have as an objective the understanding of the surrounding world. Some examples of applications may be: automatic region recognition in aerial images, object detection in robotic vision, recognition of different cellular formations in histopathologic analysis, and others.

In this thesis we analyze and implement supervised segmentation methods using Markov Random Fields, and Gabor filters. For the second case, we developed a framework for distributed processing. The third method that we implemented combines both concepts into an unsupervised algorithm. Finally, we compare the results of the different methods.

Índice general

Índice de figuras

Capítulo 1

Introducción

Una definición de **segmentación de imagenes texturadas** podría ser "*el proceso por el cual se particiona una imagen en regiones, donde cada una de ellas contiene sólo una textura, distinta de las regiones vecinas*". Sin embargo, esta definición no logra transmitir las dificultades y limitaciones prácticas del problema.

Comencemos por analizar los términos **segmentación** y **textura** por separado.

El proceso de **segmentación de una imagen** puede definirse sin problemas. En la sección 1.1 lo presentaremos más formalmente, pero una primera definición del problema podría ser la siguiente: *Una imagen está conformada por un arreglo bidimensional de pixels, y el objetivo es asignarle a cada uno de ellos una etiqueta que indique la región a la cual el pixel pertenece. Una región, es el conjunto de pixels (generalmente conectados) que comparten la etiqueta asignada.*

El problema se presenta cuando queremos extender el término a segmentación de imagenes texturadas. ¿Que consideramos una correcta segmentación? Analicemos la imagen 1-1. En ella, alguien podría desear separar (*segmentar*) los pixels en tres regiones: cielo, árbol y pasto; pero otra alternativa posible sería también separar los árboles entre sí, lo que haría que la solución buscada contenga más de 8 regiones. Esto muestra que no existe una segmentación "correcta", sino que la misma depende de los objetivos del observador.

Todavía más difícil resulta definir la noción de **textura**. Las típicas definiciones de diccionario, en general se centran en el aspecto relacionado con el sentido del tacto. En el caso

Figura 1-1: Imagen natural de un paisaje

de las texturas desde el punto de vista del procesamiento de imágenes, una definición podría ser: *uno o más patrones básicos, los cuales son repetidos periodicamente mediante alguna regla de ubicación*. Pero en general las texturas -sobre todo las naturales- prácticamente no respetan esta definición.

La dificultad de definir claramente los términos asociados a la segmentación de imágenes texturadas, nos habla de la complejidad del problema al que nos enfrentamos.

En muchos casos, las texturas que componen la imagen también difieren notoriamente en la información de color asociada. Por ejemplo, en la imagen 1-1 se podría utilizar sólo la información de los colores (prácticamente sin considerar las texturas) para asignar los pixels celestes y blancos a la región *cielo*, los verde claros a *pasto*, y los verde oscuros a *árboles*, obteniendo con esto una segmentación bastante real.

En este trabajo decidimos enfocarnos sólo en la segmentación basándonos en las **texturas**, sin tener en cuenta la información de **color**. Por este motivo utilizamos en todos los algoritmos imágenes de una sola banda, las cuales en general representamos con valores de gris entre 0 y 255.

En la la figura 1-2, podemos ver un resumen de lo que implica la segmentación de imágenes texturadas. En primer lugar vemos una imagen texturada, y a su derecha el mapa real de regiones (en este caso sintético). La solución al problema de segmentación, es encontrar el mapa más "parecido" al mapa real.

El problema de la segmentación de imágenes texturadas es un tema bastante estudiado, pudiendo categorizar a los principales métodos propuestos en dos grupos:

1. Los métodos que consideran que los valores de cada uno de los pixels de la imagen están regidos por una función de probabilidad, en cuyo caso el problema consiste en estimar los parámetros que la definen, para con ellos obtener la segmentación.

2. Los métodos basados en aplicar un banco de filtros a la imagen, para obtener de cada uno de los filtros un valor por pixel para el vector de características que permiten luego realizar la segmentación

Los métodos de segmentación pueden ser supervisados o sin supervisión, dependiendo de los requerimientos de información a priori de la imagen o de las texturas que la componen.

El objetivo del trabajo fue evaluar e implementar distintos algoritmos que representan a ambos grupos de métodos. No sólo los implementamos como estaban explicados en los distintos trabajos de referencia, sino que también ideamos mejoras a los mismos, las cuales nos permitieron incrementar la velocidad de procesamiento, como así también brindarle al usuario información extra sobre los resultados obtenidos, que los métodos evaluados no brindaban.

La segmentación de imágenes texturadas posee una gran cantidad de usos en la vida real, los cuales tienen como fin común la comprensión del mundo que nos rodea. Algunos ejemplos de aplicaciones pueden ser: reconocimiento automático de regiones en imágenes aéreas, detección de objetos en visión en robótica, reconocimiento de distintos grupos celulares en análisis histopatológico, entre otros.

El trabajo se organiza de la siguiente forma: en el capítulo 1 introducimos el tema de la segmentación de imágenes texturadas, presentando un marco formal para su especificación. En el capítulo 2 damos una introducción al tema de los campos aleatorios de Markov y su utilización para la síntesis de texturas y la segmentación de imágenes. En el capítulo 3 mostramos la implementación de un método de segmentación utilizando filtros de Gabor. En el capítulo 4 explicamos un método que puede ubicarse en ambas categorías, ya que utiliza tanto la aplicación de un banco de filtros de Gabor, como los campos aleatorios de Markov. Por último, en el capítulo 5, realizamos una comparación entre los métodos implementados y presentamos las conclusiones del trabajo.

1.1. Sitios y Etiquetas

Un problema de etiquetado puede especificarse en términos de un conjunto de sitios y un conjunto de etiquetas.

Sea S un conjunto de m sitios.

$$S = \{1, ..., m\} \qquad m \in \mathbb{Z} \tag{1.1}$$

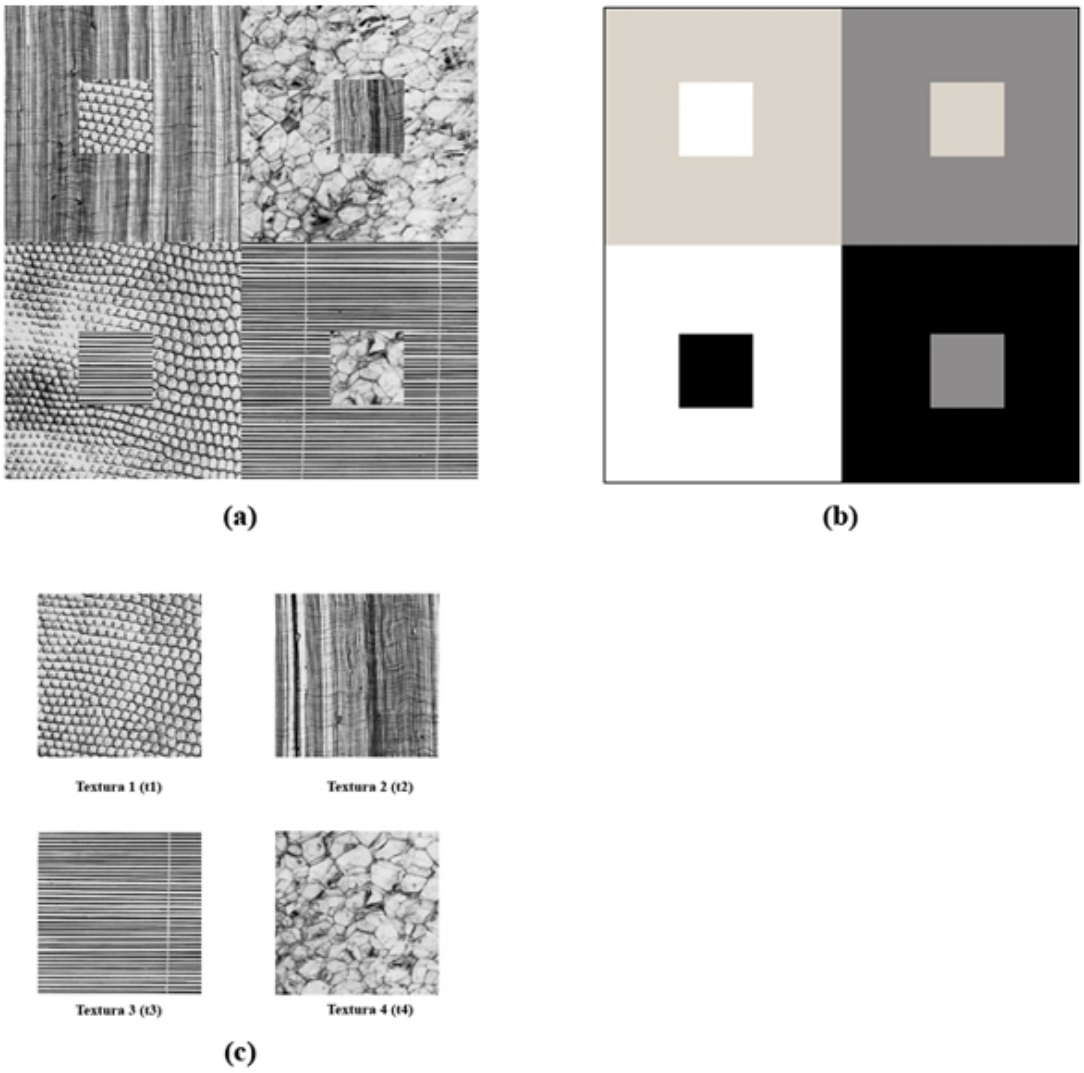


Figura 1-2: (a) Imagen texturada (imagen de entrada), conformada por las texturas t_1 a t_4. (b) Mapa de regiones subyacente de la imagen de entrada (salida ideal del proceso de segmentación). (c) Muestras de las texturas que conforman la imagen de entrada

donde $1, ..., m$ son índices. Un sitio generalmente representa a un punto o a una región en el espacio euclídeo. Por ejemplo un pixel, un vértice, un segmento o una superficie. Un conjunto de sitios puede ser categorizado en términos de su "regularidad". Los sitios en un reticulado son considerados espacialmente regulares. Un reticulado rectangular para una imagen 2D de tamaño n x n puede denotarse como

$$S^n = \{(i,j) \in \mathbb{Z}^2 | 1 \leq i,j \leq n\} \tag{1.2}$$

o en su forma más general, un reticulado rectangular de n x m sitios puede denotarse como

$$S^{nm} = \{(i,j) \in \mathbb{Z}^2 | 1 \leq i \leq n; 1 \leq j \leq m\} \tag{1.3}$$

Sus elementos corresponden a la posición donde la imagen fué muestreada. Los sitios que no presentan regularidad espacial son considerados irregulares. Este es el caso de la representación de características de una imagen a un nivel más abstracto, como bordes, vértices, etc.

Normalmente los sitios son tratados sin un orden particular. Para una imagen de $n\ x\ n$, el pixel (i,j) puede ser convenientemente reindexado por un sólo índice k, donde k toma sus valores entre $\{1, ..., m\}$ y $m = n\ x\ n$. Más allá de esta reindexación, la interrelación entre los sitios se mantiene mediante el "sistema de vecindad". Este sistema o característica de los sitios será explicado en la siguiente sección, en la que se trabajará con Campos Aleatoreos de Markov.

En el enfoque probabilístico que se presenta, una etiqueta es un evento que le puede suceder a un sitio. Un conjunto de etiquetas L puede categorizarse como discreto o continuo. En el caso continuo, podría corresponder a la línea real $\mathbb{R}$ o a un intervalo de ella

$$L_c = [X_{\text{mín}}, X_{\text{máx}}] \subset \mathbb{R} \tag{1.4}$$

siendo $X_{\text{mín}}$ y $X_{\text{máx}}$ los límites inferior y superior del intervalo.

Un ejemplo podría ser el valor analógico de la intensidad de un pixel. También es posible que una etiqueta continua tome como valor a un vector o a una matriz, por ejemplo $L_c = \mathbb{R}^{axb}$ donde a y b son dimensiones.

En el caso discreto, una etiqueta puede tomar un valor discreto de un conjunto de M etiquetas posibles

$$L_d = \{l_1, ..., l_M\} \text{ o simplemente } L_d = \{1, ..., M\} \quad M \in \mathbb{N}$$

En el caso de detección de bordes, por ejemplo, el conjunto de etiquetas discretas es $L = \{borde, no - borde\}$.

Más allá de la continuidad, otra propiedad esencial de un conjunto de etiquetas es el ordenamiento de las mismas. Por ejemplo, los elementos en un conjunto continuo $\mathbb{R}$ (el espacio real) pueden ser ordenados mediante la relación "menor que". Cuando un conjunto discreto $\{0, ..., 255\}$ representa los valores discretizados (o cuantizados) de intensidades, se lo puede considerar un conjunto ordenado debido a que tenemos $0 < 1 < 2 < ... < 255$. Cuando por el contrario el conjunto denota 256 símbolos diferentes, como por ejemplo tipos de texturas, se lo considera un conjunto sin orden, a menos que se imponga una relación de orden artificial.

Para un conjunto ordenado puede establecerse una medida numérica (cuantitativa) de similitud entre dos etiquetas. Para un conjunto sin orden, una medida de similitud es simbólica (cualitativa), típicamente tomando los valores de "igual" y "distinto". El ordenamiento de las etiquetas al igual que la medida de similitud, no sólo categoriza los problemas de "Etiquetado" sino que en gran medida afecta a la elección de los algorítmos, y por ende a la complejidad computacional de los mismos.

1.2. El problema de Etiquetado

El problema de "Etiquetado" consiste en asignarle una etiqueta del conjunto L a cada uno de los sitios de S. La detección de bordes de una imagen, por ejemplo, implica la asignación a cada uno de los sitios $i \in S$ (pixels) de una etiqueta f_i del conjunto $L = \{borde, no - borde\}$. El conjunto

$$f = \{f_1, ..., f_m\} \tag{1.5}$$

es llamado el "Etiquetado" de los sitios de S en términos de las etiquetas L, donde a cada sitio se le asigna una única etiqueta. $f_i = f(i)$ puede ser considerada una función con dominio en S e imagen en L. Debido a que el soporte de la función es todo el dominio S, queda formado un mapeo de S a L

$$f : S \rightarrow L \tag{1.6}$$

En la terminología de los campos aleatorios (que veremos más adelante), una asignación de etiquetas es llamada una configuración. En Visión, una configuración o "Etiquetado" puede corresponder a una imagen, un mapa de bordes, regiones, una interpretación de propiedades de una imagen, etc.

Cuando todos los sitios tienen el mismo conjunto de etiquetas L, el conjunto de todas las posibles asignaciones de etiquetas, es decir, el espacio de configuraciones, es el siguiente producto cartesiano

$$F = \underbrace{L \times ... \times L}_{m \text{ veces}} = L^m \tag{1.7}$$

donde m es el tamaño de S. En restauración de imágenes, por ejemplo, L contiene los valores aceptables para los pixels, los cuales son comunes a todos los sitios en S y F define a todas las imagenes aceptables.

Cuando $L = \mathbb{R}$ es la recta real, $F = \mathbb{R}^m$ es el espacio real m-dimensional. Cuando L es un conjunto discreto, el tamaño de F es combinatorio. Para un problema con m sitios y M etiquetas, por ejemplo, existe un total de M^m posibles configuraciones en F.

1.3. Problemas de etiquetado en Visión

En términos de regularidad y de continuidad, se pueden clasificar a los problemas de "Etiquetado" en las siguiente categorías:

1. Sitios regulares con etiquetas continuas

2. Sitios regulares con etiquetas discretas

3. Sitios irregulares con etiquetas continuas

4. Sitios irregulares con etiquetas discretas

Las dos primeras categorías caracterizan a los problemas de procesamiento de bajo nivel realizado sobre las imagenes observadas, mientras que las otras dos, caracterizan a los problemas de procesamiento de alto nivel sobre información extraída de la imagen en un primer procesamiento (bordes, vértices, etc.).

La segmentación de una imagen texturada en regiones es considerada de tipo 2. En este proceso se particiona una imagen observada en una serie de regiones mutuamente excluyentes, donde los sitios (pixels) pertenecientes a cada una de ellas posee una propiedad uniforme y homogenea, cuyos valores son sensiblemente distintos a los sitios de las regiones vecinas. Esta propiedad puede ser el nivel de gris, el color, o como nuestro caso de estudio: **la textura**. A todos los sitios (pixels) dentro de una región se les asigna la misma etiqueta que los identifica del resto de las regiones. En este caso si hubiera M regiones distintas tendríamos

$$L = \{región_1, región_2, ..., región_M\} \tag{1.8}$$

Para los problemas de Etiquetado discreto de m sitios con M etiquetas, existe un total de M^m posibles configuraciones o Etiquetados. Para el caso de los problemas con etiquetas continuas, en cambio, la cantidad de configuraciones es infinita.

Sin embargo, entre todas las posibles configuraciones, un pequeño subconjunto de ellas son buenas soluciones, y sólo unas pocas se encuentran entre las soluciones óptimas en base al criterio definido en el problema.

1.4. Etiquetado con restricciones contextuales

La información contextual es muy utilizada en los problemas de comprensión de imagenes. En el análisis de imágenes y el reconocimiento de patrones se utiliza desde el año 1962 [1][2].

Se deja de lado la supuesta independencia estadística y se consideran las dependencias entre los sitios vecinos. La información de los vecinos más cercanos es utilizada para obtener la probabilidad condicional de un pixel dado.

Las restricciones contextuales pueden ser expresadas localmente en términos de la probabilidad condicional $P(f_i|\{f_{i\prime}\})$, donde $\{f_{i\prime}\}$ denota el conjunto de etiquetas en todos sitios distintos de i ($i' \neq i$), o globalmente como la probabilidad conjunta $P(f)$.

Debido a que la información local se puede observar de forma más directa, es normal que se realice una inferencia global en base a la información local.

En las situaciones donde las etiquetas son independientes entre sí (sin restricciones contextuales), la probabilidad conjunta es el producto de las probabilidades locales

$$P(f) = \prod_{i \in S} P(f_i) \tag{1.9}$$

Esto implica independencia condicional

$$P(f_i|\{f_{i'}\}) = P(f_i) \qquad i \neq i' \tag{1.10}$$

Con lo cual, un etiquetado global f puede ser computado considerando cada etiqueta f_i localmente. Esto es muy ventajoso para resolver el problema, ya que simplifica su cómputo.

En los casos en que hay presencia de información sobre interacción entre los distintos pixels de la imagen, las etiquetas son mutuamente dependientes. Las relaciones simples que expresamos en (1.9) y (1.10) ya no son válidas. Realizar una inferencia global utilizando información local pasa a ser una tarea no trivial. La teoría de Campos Aleatorios de Markov provee una base matemática para resolver este problema.

Capítulo 2

Campos Aleatorios de Markov y Gibbs

En este capítulo presentamos formalmente los Campos Aleatorios de Markov (Markov Random Fields o MRF) y los Campos Aleatorios de Gibbs (Gibbs Random Fields o GRF), la demostración de su equivalencia y los algoritmos de Gibbs y Metrópolis para la síntesis de texturas. Además se muestran las imágenes resultantes de la implementación de los modelos con ambos algoritmos y los efectos que la variación de parámetros tienen en las imágenes resultantes.

Luego presentamos un algoritmo de segmentación en el cual se conocen a priori los parámetros asociados a cada una de las texturas que componen la imagen.

En la última sección del capítulo mostramos la implementación de un algoritmo que permite la estimación de los parámetros de una textura basándose en una muestra de la misma. Finalmente integramos los algoritmos de estimación de parámetros y de segmentación para obtener resultados con menor información a priori.

2.1. Campos Aleatorios de Markov

Sea $F = \{F_1, ..., F_m\}$ una familia de variables aleatorias definidas en el conjunto S, en el cual cada variable F_i toma un valor f_i en $\mathcal{L}$.

La familia F es llamada Campo Aleatorio. Utilizamos la notación $F_i = f_i$ para denotar el evento en el que F_i toma el valor f_i y la notación $(F_1 = f_1, ..., F_m = f_m)$ para denotar el evento conjunto.

Para simplificar la notación, un evento conjunto se abrevia como

$$F = f \text{ donde } f = \{f_1, ...f_m\}$$

es una configuración de F, correspondiente a una realización (instancia) del campo aleatorio.

Para un conjunto discreto de etiquetas $\mathcal{L}$, la probabilidad de que la variable aleatoria F_i tome el valor f_i se denota como $P(F_i = f_i)$, abreviándose como $P(f_i)$ a menos que haya necesidad de expandir la expresión.

La probabilidad conjunta se denota como $P(F = f) = P(F_1 = f_1, ..., F_m = f_m)$, abreviándose como $P(f)$.

Para un $\mathcal{L}$ continuo, tenemos las funciones de densidad $p(F_i = f_i)$ y $p(F = f)$.

F es un **Campo Aleatorio de Markov** si se satisfacen las siguientes condiciones:

$$\text{Positividad: } P(f) > 0, \forall f \tag{2.1}$$

$$\text{Markovianidad: } P\left(f_i \mid f_{S-\{i\}}\right) = P\left(f_i \mid f_{N_i}\right) \tag{2.2}$$

Donde $S - \{i\}$ es el conjunto de todos los sitios en S menos el sitio i, $f_{S-\{i\}}$ denota el conjunto de etiquetas en los sitios en $S - \{i\}$ y $f_{N_i} = \{f_{i\prime} \mid i\prime \in N_i\}$ el conjunto de etiquetas en los sitios vecinos a i.

La positividad se asume por razones técnicas y generalmente puede ser satisfecha en la práctica. Por ejemplo, si se da la positividad, la probabilidad conjunta $P(f)$ de cualquier campo aleatorio es determinada unívocamente por su probabilidad condicional local (Besag 1974 [3]).

La markovianidad describe la característica local de F. La etiqueta en un sitio depende únicamente de sus sitios vecinos. En otras palabras, sólo los sitios vecinos tienen interacción directa entre ellos.

Siempre es posible elegir un N_i suficientemente grande como para satisfacer la condición de Markovianidad. El vecindario más grande posible se forma con todos los sitios. En tal sistema de vecindario, cualquier F es un campo aleatorio de Markov (MRF).

Un MRF puede tener otras propiedades tales como la homogeneidad e isotropía. Se dice homogeneo si $P(f_i \mid f_{N_i})$ no depende de la posición relativa del sitio i en S. La isotropía será ilustrada en la siguiente sección con los potenciales de cliques.

Existen dos enfoques para especificar un MRF: en términos de la probabilidad condicional $P(f_i \mid f_{N_i})$ y en términos de la probabilidad conjunta $P(f)$. Besag (1974) [3] argumenta a favor del enfoque que utiliza la probabilidad conjunta en vista de las desventajas del que usa la probabilidad condicional:

- No existe un método intuitivo disponible para deducir la probabilidad conjunta a partir de las probabilidades condicionales asociadas.
- Las probabilidades condicionales están sujetas a condiciones de consistencia altamente restrictivas y no obvias.
- La especificación natural de un equilibrio de un proceso estadístico está dada en términos de la probabilidad conjunta en lugar de las probabilidades condicionales de las variables.

Afortunadamente, un resultado teórico acerca de la equivalencia entre los campos aleatorios de Markov y la distribución de Gibbs provee un método "tratable" para la especificación de la probabilidad conjunta de un MRF

2.2. Campos Aleatorios de Gibbs

Un conjunto F de variables aleatorias es considerado un campo aleatorio de Gibbs (GRF) en S con respecto a N si y sólo si su configuración obedece a una distribución de Gibbs.

Una distribución de Gibbs toma la siguiente forma:

$$P(f) = Z^{-1} e^{-\frac{1}{T}U(f)}$$

donde

$$Z = \sum_{f \in F} e^{-\frac{1}{T}U(f)}$$

es una constante normalizadora llamada *función de partición*, T es una constante llamada *temperatura* cuyo valor se asumirá como 1 a menos que se especifique otro valor, y $U(f)$ es la *función de energía*.

$$U(f) = \sum_{c \in C} V_c(f)$$

La energía es la suma de los potenciales de clique sobre todas los cliques posibles C. El valor de $V_c(f)$ depende de la configuración local del clique c. La *distribución Gaussiana* es un caso particular dentro de la familia de distribuciones de Gibbs.

Un GRF se denomina homogéneo si $V_c(f)$ es independiente de la posición relativa de la clique c en S.

Se lo denomina isotrópico si V_c es independiente de la orientación de c.

La especificación de un GRF se simplifica considerablemente si es homogéneo o isotrópico. En la mayor parte de los modelos de visión MRF se asume la homogeneidad por conveniencia matemática y computacional. La condición de isotropía es una propiedad de regiones independientes de la dirección y de formato "blob" (regiones homogéneas grandes).

2.3. Equivalencia Markov-Gibbs

Un MRF es caracterizado por su propiedad local (la Markovianidad) mientras que un GRF se caracteriza por su propiedad global (la distribución de Gibbs). El teorema Hammersley-Clifford (Hammersley y Clifford 1971 [4]) establece la equivalencia de estos dos tipos de propiedades. El teorema establece que F es un MRF en S respecto de N si y sólo si F es un GRF en S respecto de N. (Besag 1974 [3]).

A continuación probaremos que un GRF es un MRF:

Sea $P(f)$ una distribución de Gibbs respecto de un vecindario N. Consideremos la proba-

bilidad condicional

$$P\left(f_i \mid f_{S-\{i\}}\right) = \frac{P\left(f_i, f_{S-\{i\}}\right)}{P\left(f_{S-\{i\}}\right)} = \frac{P\left(f\right)}{\sum\limits_{f_i' \in \mathcal{L}} P\left(f'\right)}$$

Dividiendo C en dos conjuntos A y B,en dónde A está formado por las cliques que contienen a i y B por las que no lo contienen, lo anterior se puede escribir como

$$P\left(f_i \mid f_{S-\{i\}}\right) = \frac{\left[e^{-\sum_{c\in A} V_c(f)}\right]\left[e^{-\sum_{c\in B} V_c(f)}\right]}{\sum_{f_i'}\left\{\left[e^{-\sum_{c\in A} V_c(f')}\right]\left[e^{-\sum_{c\in B} V_c(f')}\right]\right\}}$$

Debido a que $V_c\left(f\right) = V_c\left(f'\right)$ para cualquier clique c en la que no esté contenido i, $e^{-\sum_{c\in B} V_c(f)}$ se cancela en el numerador y denominador. Por lo tanto, esta probabilidad depende sólamente de los potenciales de las cliques que contienen a i,

$$P\left(f_i \mid f_{S-\{i\}}\right) = \frac{e^{-\sum_{c\in A} V_c(f)}}{\sum_{f_i'} e^{-\sum_{c\in A} V_c(f')}}$$

es decir, depende de las etiquetas de los vecinos de i.

La demostración de que un MRF es un GRF es mucho más compleja y se obtiene a través de un resultado de Griffeath (1976) [12].

El valor práctico del teorema es que provee una manera simple de especificar la probabilidad conjunta. Uno puede especificar $P\left(F = f\right)$ determinando las funciones de potenciales de cliques $V_c\left(f\right)$. De esta manera se codifica el conocimiento *a priori* de la preferencia acerca de las interacciones entre etiquetas.

Uno de los tópicos principales en el modelado de un MRF es de qué manera seleccionar las formas y parámetros de las funciones de potencial para la codificación de restricciones. La forma de las funciones de potenciales determina la forma de la distribución de Gibbs. Cuando son especificados todos los parámetros involucrados en las funciones de potencial, la distribución de Gibbs está definida completamente.

Para calcular la probabilidad conjunta de un MRF, el cual es una distribución de Gibbs, es necesario evaluar la función de partición. Debido a que es la suma sobre un número combinatorio

de configuraciones en F, el cálculo es usualmente intratable. La evaluación explícita puede ser evitada en modelos de vision MRF de *máxima probabilidad* cuando $U(f)$ no contiene parámetros desconocidos. Sin embargo, esto no es cierto cuando la estimación de parámetros es parte del problema. En el segundo caso, la función de energía $U(f) = U(f \mid \theta)$ también es función de los parámetros θ ,como también sucede con la función de partición $Z = Z(\theta)$. La evaluación de $Z(\theta)$ es necesaria. Para evitar la gran dificultad que esto representa, en la práctica la función de probabilidad conjunta es a menudo aproximada.

2.4. Modelado de Texturas con MRF

Un modelo MRF de textura puede ser definido por la función de probabilidad conjunta $P(f)$. La probabilidad determina la frecuencia de ocurrencia del patrón de textura f.

Se han utilizado diferentes modelos de Campos Aleatorios de Markov (MRF) para modelar texturas, entre ellos el multi-level logistic model (MLL) (Derin y Elliott 1987 [13])

En esta sección presentamos un caso particular del MLL que permite modelar un amplio rango de texturas. Utilizamos estas texturas como caso de estudio para la síntesis de las mismas.

El objetivo es generar imágenes sintéticas basándonos en el modelo de MRF elegido y en dos métodos de muestreo diferentes.

En el modelo MLL, la probabilidad de una textura determinada está dada por la función de potencial de cliques. Esta función es la que le dará peso a las diferentes configuraciones que forman la textura. En la figura 2-1 se ven las configuraciones de cliques posibles y sus respectivos parámetros asociados.

Si se consideran sólo cliques de dos elementos, se deben determinar cuatro parámetros para la función de potencial. Estos parámetros se ilustran en la figura 2-2.

La función de potencial se define de la siguiente manera:

$$V(f_i, f_{i\prime}) = \begin{cases} \beta_c & \text{Si los sitios en } \{i, i\prime\} = c \in C^2 \text{ tienen la misma etiqueta.} \\ -\beta_c & \text{Si no.} \end{cases} \tag{2.3}$$

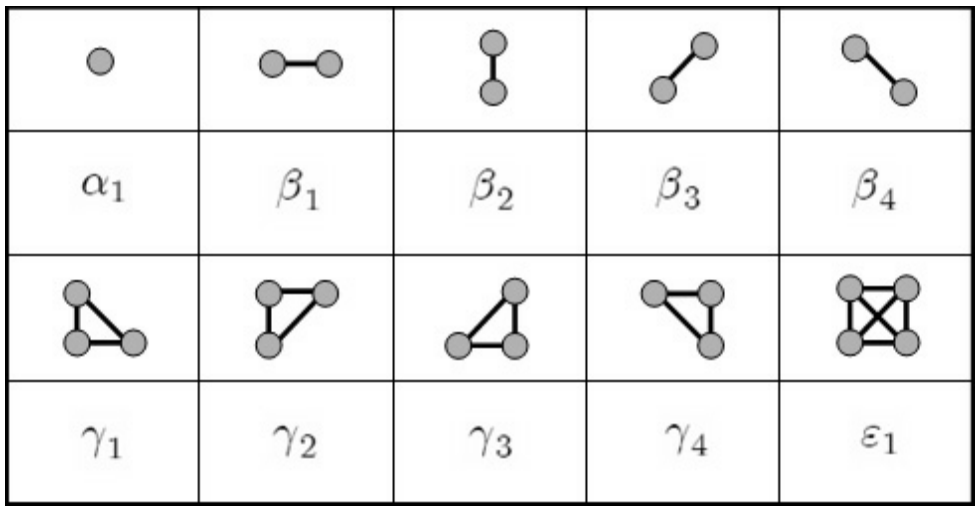


Figura 2-1: Cliques existentes en un sistema de vecindad de segundo grado (ocho vecinos), y sus parámetreos de potencial asociados. Para las cliques de un solo sitio, se especifica un parámetro α_i para cada etiqueta $l_i \in \mathcal{L}$.

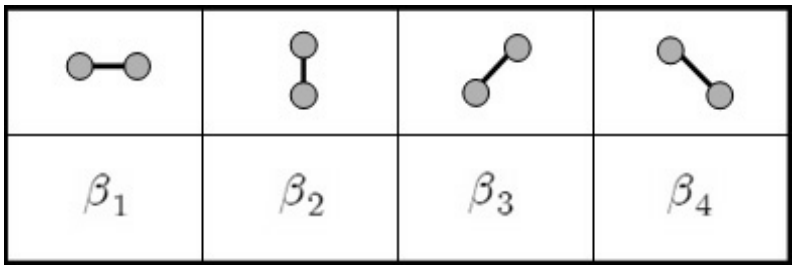


Figura 2-2: Cliques utilizadas en nuestra implementación.

donde $f_i \in \{1, ..., M\}$, C^2 es el conjunto de Cliques de dos elementos y β_c es el parámetro asociado al tipo de clique c.

De esta forma se determina cuán fuertes serán las tendencias a formar patrones horizontales, verticales o diagonales (manteniendo el color) en la textura modelada.

La probabilidad $P(f)$ puede calcularse a partir de $V(f_i, f_{i\prime})$ usando la fórmula de Gibbs:

$$P(f) = \frac{1}{Z} e^{-U(f)}$$

La imagen texturada puede obtenerse muestreando la función de distribución.

A continuación se presentan dos métodos para realizar esta tarea:

Algoritmo 1 *Sintesis de textura utilizando el algoritmo Metropolis*

Comienzo

Inicializar f *con valores random en* $\mathcal{L}^S$

Repetir N *veces*

Para cada $i \in S$

$f\prime_{i\prime} = f_{i\prime}$ *para todo* $i\prime \neq i$

Elegir $f_i \in \mathcal{L}$ *al azar.*

$p = min\left\{1, \frac{P(f\prime)}{P(f)}\right\}$ *donde* P *es la distribución de Gibbs dada.*

Remplazar f *por* $f\prime$ *con probabilidad* p

Siguiente i

Fin Repetir

Fin

Algoritmo 2 *Síntesis de textura, utilizando el algoritmo Gibbs*

Comienzo

Inicializar f *con valores random en* $\mathcal{L}^S$

Repetir N *veces*

Para cada $i \in S$

$p_l = P\left(f_i = l \mid f_{Ni}\right)$ *para todo* $l \in \mathcal{L}$

donde f_{Ni} *son los valores de los pixeles vecinos.*

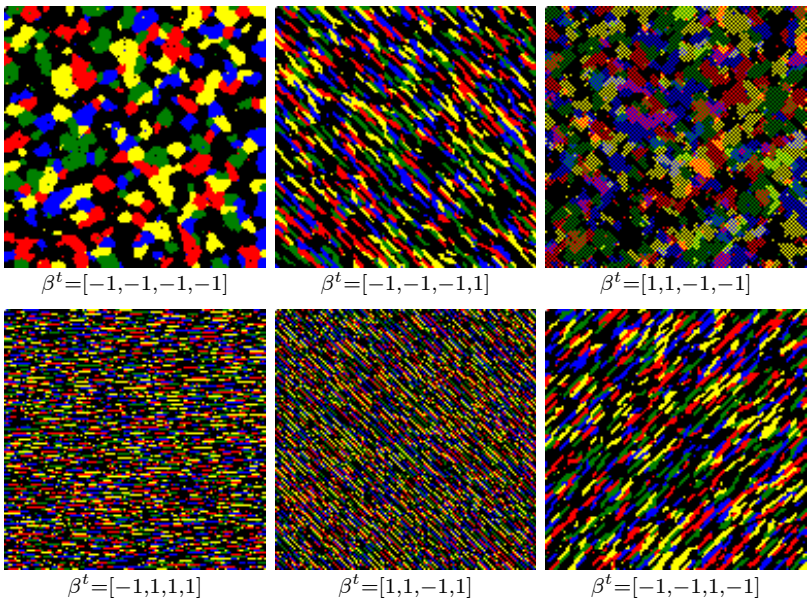
Reemplazar fi *por* l *con probabilidad* p_l

Siguiente i

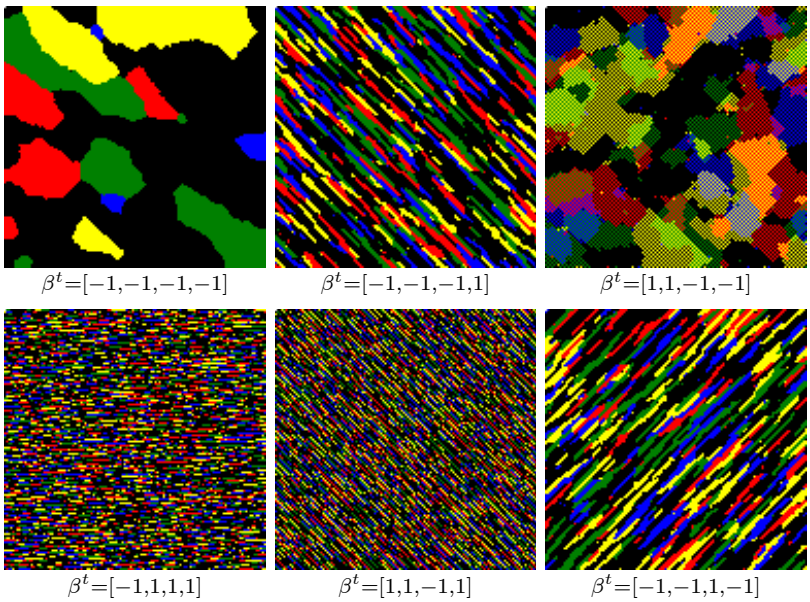
Fin Repetir

Fin

Las siguientes imagenes fueron generadas con el algoritmo Metropolis con los parámetros indicados



Las siguientes imagenes fueron generadas con el algoritmo Gibbs con los parámetros indicados



En todos los casos se trabajó con 4 intensidades diferentes. Por claridad, se utilizaron los colores negro, rojo, amarillo y azul para graficar las imágenes. En el tercer caso ($\beta^t = [1, 1, -1, -1]$)

las cliques que se forman son diagonales, por eso es que se ven zonas de tipo blob, pero no sólidas como en el caso de la primer imágen.

Las diferencias entre los dos algoritmos se dan en la forma de calcular el nuevo valor de pixel. El primer algoritmo sólo necesita evaluar una función exponencial dado que la actualización se basa en la relación $P(f\prime)/P(f)$. El segundo algoritmo debe calcular M funciones exponenciales. Además, cuando los exponentes son grandes, el cálculo se vuelve impreciso.

En cuanto al comportamiento de los dos algoritmos, se puede observar lo siguiente:

1. Para los dos algoritmos 50 iteraciones son suficientes para generar texturas que respondan al modelo.

2. El primer algoritmo es más rápido que el segundo.

3. El segundo algoritmo tiende a generar realizaciones f con menor energía que el primero.

2.5. Segmentación supervisada

En el algoritmo de segmentación de texturas supervisada que presentamos a continuación, se asume que todos los parametros de las texturas, y del ruido si estuviera presente, están especificados (Geman y Geman 1984 [15]); y la segmentación consiste en particionar la imagen en términos de las texturas cuyas funciones de distribución fueron especificadas completamente.

Sea el siguiente conjunto de etiquetas $\mathcal{L} \in \{1, ..., M\}$ y f una segmentación en la que $f_i \in \mathcal{L}$ es el indicador del tipo de textura para el pixel i. Denotamos el conjunto de todos los sitios con etiqueta I de la siguiente manera

$$S^{(I)}(f) = \{i \in S \mid I = f_i\}$$

entonces

$$S = \bigcup_{I \in \mathcal{L}} S^{(I)}(f)$$

y

$$S^{(I)}(f) \cap S^{(J)}(f) = \varnothing, I \neq J$$

La función de energía de la verosimilitud puede expresarse como

$$U(d \mid f) = \sum_{c \in C} V_c(d \mid f) = \sum_{I \in \mathcal{L}} \sum_{c \subset S^{(I)}} V_c^{(I)}(d \mid f)$$

donde $V_c^{(I)}(d \mid f)$ es la función de potencial para el dato d en c marcado con etiqueta I.

Supongamos que la textura de tipo I es modelada como una MLL con parámetros

$$\theta^{(I)} = \left\{ \alpha^{(I)}, \beta^{(I)}, ... \right\}$$

Entonces los potenciales de cliques de un sólo sitio son

$$V_c^{(I)}(d \mid f) = \alpha^{(I)}$$

donde $I = f_i$ y los potenciales de cliques de múltiples sitios

$$V_c^{(I)}(d \mid f) = \begin{cases} \beta_c^{(I)} & \text{si todos los } d_i,\ i \in c, \text{ son iguales} \\ -\beta_c^{(I)} & \text{si no} \end{cases}$$

Lo anterior se aplica a las cliques en el interior de $S^{(I)}$. En los bordes, una clique podría montarse sobre dos o más $S^{(I)}$. En ese caso se usará la siguiente regla para determinar el tipo de textura para generar el dato: Si c "se sienta" mayormente en un $S^{(I)}$ en particular entonces se eligen los parámetros $\theta^{(I)}$; si no, se eligen un $S^{(I)}$ al azar de las etiquetas involucradas. Cuando los niveles de gris para la textura I son conocidos $D_I = \left\{ l_1^{(I)}, ..., l_{M_I}^{(I)} \right\}$ donde M_I es el número de valores de gris de la textura I , se establecen más restricciones en la segmentación de texturas y se pueden esperar mejores resultados. Cuando los niveles de gris también son expuestos a ruido, las restricciones se vuelven inexactas.

Luego de definir $U(f)$ para el proceso de regiones, la energía posterior se puede obtener

como

$$U(f \mid d) = U(f) + U(d \mid f)$$

Se realizan ciertas asunciones de independencia para simplificar las fórmulas (Derin y Elliott 1987 [13]) y se utiliza un algoritmo de programación dinámica.

2.5.1. Algoritmo de Segmentación Supervisada

El algoritmo de segmentación supervisado se puede resumir de la siguiente forma:

Algoritmo 3 *Segmentación supervisada*

Comienzo

Para cada $i \in S$

indiceMax = 0

probMax = calcularProbabilidadTextura(imagen, i, textura[0]*)*

Para cada textura[j] *tal que* $j > 0$

probActual = calcularProbabilidadTextura(imagen, i, textura[j]*)*

Si probActual > probMax Entonces

probMax=probActual

indiceMax=j

Fin Si

Fin Para

segmentacion[i]=indiceMax

Fin Para

Donde la función calcularProbabilidadTextura se evalúa mediante la siguiente fórmula:

$$\text{calcularProbabilidadTextura(imagen, i, f)} = \frac{e^{-\sum_{c \in A} V_c(f)}}{\sum_{f_i'} e^{-\sum_{c \in A} V_c(f')}} \tag{2.4}$$

con A vecindad de i y f etiqueta.

Como se puede ver en 2.4, el denominador es independiente del sitio en el que se esté evaluando la fórmula, con lo cual se puede precalcular antes de comenzar el algoritmo para cada una de las distintas texturas.

Para obtener mejores resultados, le aplicamos a la salida del algoritmo un postprocesamiento, en el cual asignamos a cada pixel el valor medio de sus vecinos (en un vecindario de siete por siete):

Algoritmo 4 *Postprocesamiento de la salida de la segmentación*

Comienzo
Para cada x ∈ [0..anchoImagen]
Para cada y ∈ [0..altoImagen]
suma= 0
Para cada j ∈ [-3..3]
Para cada k ∈ [-3 .. 3]
// Me aseguro no tomar en cuenta
//el valor que estoy modificando
Si $(j \neq 0 \vee k \neq 0)$ *Entonces*
suma = suma + segmentacion[x+j, y+k]
Fin Si
Fin Para
Fin Para
// Como la ventana tomada es de 7x7, divido por 48
segmentacion[x,y]=round(suma / 48)
Fin Para

2.5.2. Resultados - Segmentación supervisada

A continuación mostramos un ejemplo de la ejecución del algoritmo de síntesis y segmentación de texturas utilizando campos aleatorios de Markov.

La imagen 2-3 está conformada por cuatro texturas sintéticas, generadas utilizando el algoritmo de muestreo de Gibbs. Los parámetros de cada una de las texturas son,

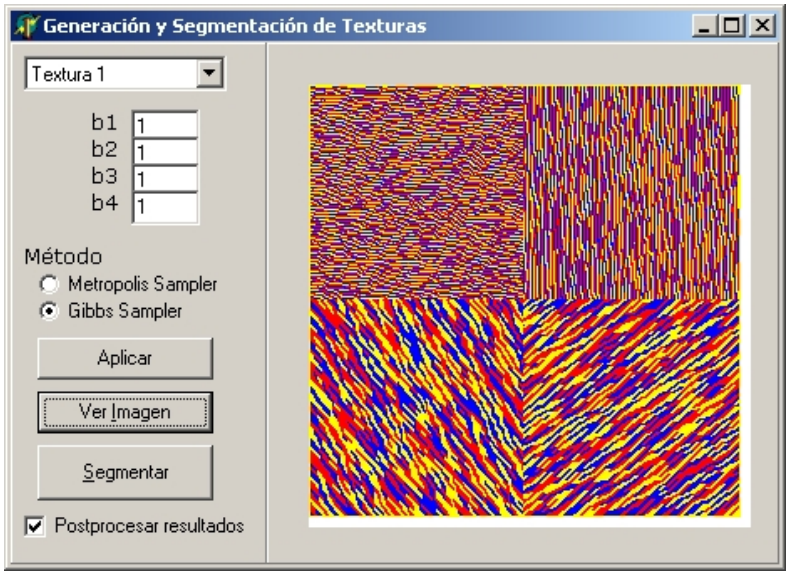
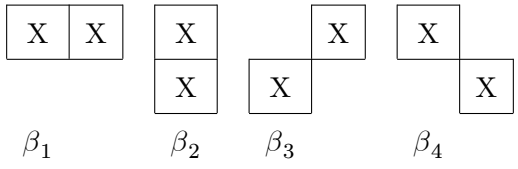


Figura 2-3: Imagen texturada, conformada por 4 texturas sintéticas generadas mediante el algoritmo de Gibbs. Parámetros $(\beta_1, \beta_2, \beta_3, \beta_4)$: t_1(arriba, izquierda) $= (-1, 3, -1, 3)$; t_2(arriba, derecha) $= (1, -3, -1, 1)$; t_3(abajo, izquierda) $= (1, -3, -3, 1)$; t_4(abajo, derecha) $= (-3, 1, 1, -3)$

$$
\begin{aligned}
t_1 &: \beta_1 = -1; \beta_2 = 3; \beta_3 = -1; \beta_4 = 3 \\
t_2 &: \beta_1 = 1; \beta_2 = -3; \beta_3 = -1; \beta_4 = 1 \\
t_3 &: \beta_1 = 1; \beta_2 = -3; \beta_3 = -3; \beta_4 = 1 \\
t_4 &: \beta_1 = -3; \beta_2 = 1; \beta_3 = 1; \beta_4 = -3
\end{aligned}
$$

con los β_i representando las siguientes cliques,



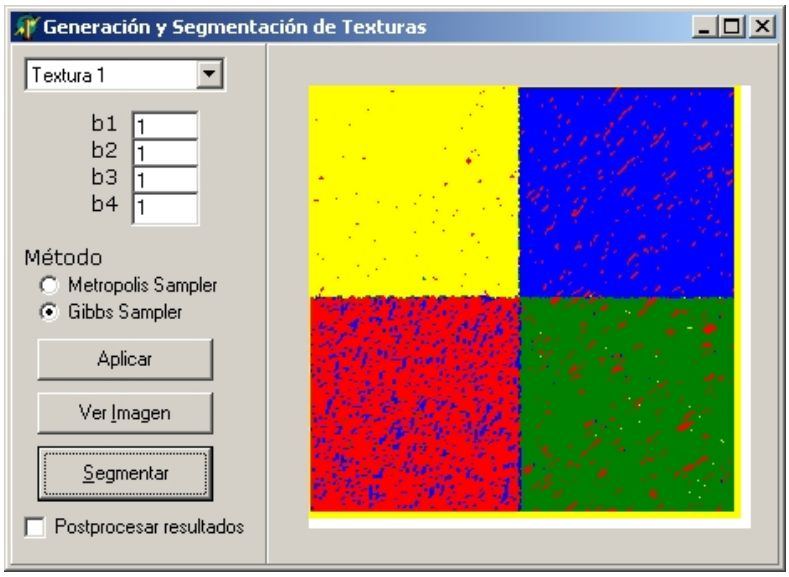


Figura 2-4: Resultado de la segmentacion: Pixels correctamente identificados 87,9 % (error 12,1 %). Aunque se pueden ver una gran cantidad de pixels erroneamente clasificados.

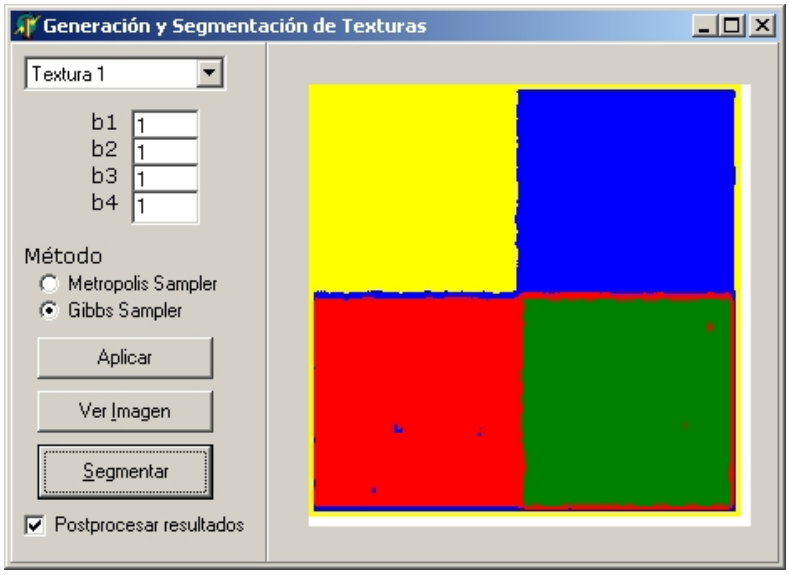


Figura 2-5: Segmentación mejorada, utilizando el algoritmo de postprocesamiento. Pixels correctamente clasificados 91.3 % (Error: 8.7 %)

2.6. Estimación de Parámetros

En el algoritmo de segmentación supervisada presentado en la sección anterior se asumen conocidos los parámetros que definen a cada una de las texturas involucradas. Esto limita la potencia del algoritmo ya que en la práctica se presentan casos en los que se cuenta con muestras de las texturas pero no de los parámetros asociados a cada una de ellas. Por esto resulta útil contar con un método que dada una muestra de una textura permita estimar los parámetros del modelo MRF que mejor la aproxime.

Existen varios métodos de estimación de parámetros. El "método de codificación" propuesto por Besag es básicamente un método de máxima verosimilitud y requiere la solución de un sistema de ecuaciones no lineales, lo cual resulta bastante engorroso de implementar y las implementaciones no son del todo confiables. El método propuesto por Derin y Elliott utiliza sistemas de ecuaciones lineales que pueden ser resueltos a través del método de mínimos cuadrados clásico. Los modelos de Gibbs utilizados consideran un vecindario de segundo grado comúnmente llamado "Ocho Vecinos".

El método de estimación de parámetros implementado deriva del algoritmo propuesto por Derin y Elliott [13].

2.6.1. Estimación utilizando mínimos cuadrados

Sea F un GRF (Gibbs Random Field) en un espacio discreto definido en S; L el conjunto de etiquetas

$$L = \{l_1 ... l_M\}$$

y N_i el sistema de vecindad de ocho vecinos definido como

v_1	u_2	v_2
u_1	i	u_3
v_4	u_4	v_3

(2.5)

Llamemos f a una instancia (realización o muestra) de F.

Sea i un sitio en S y $t\prime$ un vector con los ocho vecinos de i.

$$t\prime = [u_1, u_2, u_3, u_4, v_1, v_2, v_3, v_4]^t$$

donde $u_1..u_4, v_1..v_4$ corresponden a la figura 2.5.

Se definen las funciones indicadoras de la siguiente manera:

$$I(z_1, ..., z_k) = \begin{cases} -1 & \text{si } \ f_{z_1} = f_{z_2} = f_{z_3} = ... = f_{z_k} \\ 0 & \text{si no} \end{cases}$$

$$J\,(i, l) = \begin{cases} 1 & \text{si } f_i = l \\ 0 & \text{si no} \end{cases}$$

Sea $V\,(i, t\prime, \theta)$ la suma de las funciones de potencial de todas las cliques en S que contienen a i, es decir:

$$V\,(f_i, f_{t\prime}, \theta) = \sum_{c \in C_i} V_c\,(f)$$

C_i es el conjunto de todas las cliques de S que incluyen a i .

donde θ es el vector de parámetros de la distribución $[\alpha_1..\alpha_M, \beta_1..\beta_4, \gamma_1..\gamma_4, \xi_1]^t$

El modelo que implementamos sólo considera los parámetros β_i para la síntesis de texturas, la segmentación y también para la estimación de parámetros. De todas maneras mostramos su forma general y luego la implementada por nosotros.

Usando los potenciales de cliques podemos escribir $V(f_i, f_{t\prime}, \theta)$ como $\phi^t(i, t\prime)\,\theta$ donde

$$\phi^t(i,t\prime) = \begin{array}{l} [J(i,1), J(i,2), ..., J(i,M), \\ I(i,u_1) + I(i,u_3), \\ I(i,u_2) + I(i,u_4), \\ I(i,v_2) + I(i,v_4), \\ I(i,v_1) + I(i,v_3), \\ I(i,u_2,v_2) + I(i,u_4,u_3) + I(i,u_1,v_4), \\ I(i,u_4,v_3) + I(i,u_2,u_3) + I(i,u_1,v_1), \\ I(i,u_2,v_1) + I(i,u_1,u_4) + I(i,u_3,v_3), \\ I(i,u_1,v_2) + I(i,u_4,u_4) + I(i,u_3,v_2), \\ I(i,u2,v2,u3) + I(i,u3,v3,u4) + I(i,u1,v1,u2) + I(i,u4,v4,u1)] \end{array}$$

En nuestro caso definimos θ y ϕ de la siguiente manera

$$\theta^t = [\beta_1, \beta_2, \beta_3, \beta_4]$$

$$\phi^t(i,t\prime) = \begin{array}{l} [I(i,u_1) + I(i,u_3), \\ I(i,u_2) + I(i,u_4), \\ I(i,v_2) + I(i,v_4), \\ I(i,v_1) + I(i,v_3)] \end{array}$$

Supongamos ahora que $P(f_i, f_{t\prime})$ es la función de distribución conjunta de las variables aleatorias en la matriz de 3x3 centrada en i y $P(f_{t\prime})$ es la función de distribución conjunta de las variables en el vecindario de i.

La distribución condicional $P(f_i \mid f_{t\prime})$ está dada por

$$\frac{P(f_i, f_{t\prime})}{P(f_t)}$$

A partir de la equivalencia entre MRF y GRF y la característica de localidad resultante

tenemos:

$$\frac{P\left(f_i, f_{t\prime}\right)}{P\left(f_{t\prime}\right)} = P\left(f_i \mid f_{t\prime}\right) = \frac{e^{-V\left(f_i, f_{t\prime}, \theta\right)}}{W\left(f_{t\prime}, \theta\right)}$$

Donde

$$W\left(f_{t\prime}, \theta\right) \triangleq \sum_{l \in L} e^{-V\left(l, f_{t\prime}, \theta\right)}$$

Reagrupando

$$\frac{e^{-V\left(f_i, f_{t\prime}, \theta\right)}}{P\left(f_i, f_{t\prime}\right)} = \frac{W\left(f_{t\prime}, \theta\right)}{P\left(f_{t\prime}\right)}$$

El lado derecho de esta ecuación es independiente de i, por lo tanto también lo es el lado izquierdo.

Dados j y k dos valores diferentes de f_i obtenemos que

$$e^{-V\left(j, f_{t\prime}, \theta\right) + V\left(k, f_{t\prime}, \theta\right)} = \frac{P\left(j, f_{t\prime}\right)}{P\left(k, f_{t\prime}\right)}$$

Aplicando logaritmo natural

$$-V\left(j, f_{t\prime}, \theta\right) + V\left(k, f_{t\prime}, \theta\right) = ln\left(\frac{P\left(j, f_{t\prime}\right)}{P\left(k, f_{t\prime}\right)}\right)$$

Sustituyendo

$$\left(\phi\left(k, t\prime\right) - \phi\left(j, t\prime\right)\right)^t \theta = ln\left(\frac{P\left(j, f_{t\prime}\right)}{P\left(k, f_{t\prime}\right)}\right)$$

Asumiendo que el lado derecho puede ser determinado o estimado, la única incógnita es θ.

Tanto $\phi^t\left(k, t\prime\right)$ como $\phi^t\left(j, t\prime\right)$ pueden ser calculados fácilmente para cualquier j y k.

Por cada par (j, k) y cada valor posible de $t\prime$ queda determinada una ecuación lineal. Se pueden obtener de esta forma un gran número de ecuaciones lineales.

La cantidad de ecuaciones será muy superior a la cantidad de incógnitas (dimensión de θ), por lo tanto podemos utilizar el método de mínimos cuadrados para encontrar una solución al sistema.

Para estimar $P(f_i, f_{t\prime})$ utilizamos técnicas de histogramas.

Llamemos K a la cantidad de bloques de 3x3 existentes en la imagen y $N(f_i, f_{t\prime})$ a la cantidad de veces que el bloque de 3x3 definido por $(f_i, f_{t\prime})$ se presenta en la misma, es posible estimar

$$P(f_i, f_{t\prime}) \cong \frac{N(f_i, f_{t\prime})}{K}$$

Cabe aclarar que los pares $(f_i, f_{t\prime})$ para los cuales $N(f_i, f_{t\prime}) = 0$ no serán considerados debido al cociente y el logaritmo natural del lado derecho de las ecuaciones lineales.

2.6.2. Resultados de la estimación de parámetros

	Parámetros reales ($\beta_1, \beta_2, \beta_3, \beta_4$)	**Parámetros estimados** ($\beta_1, \beta_2, \beta_3, \beta_4$)
Textura 1	$[-1, -1, -1, 1]$	$[-0{,}77, -0{,}79, -0{,}84, 0{,}71]$
Textura 2	$[-1, -1, 1, 1]$	$[-0{,}75, -0{,}75, 0{,}62, 0{,}64]$
Textura 3	$[-1, -1, 1, -1]$	$[-0{,}74, -0{,}67, 0{,}63, -0{,}85]$
Textura 4	$[-1, 1, 1, 1]$	$[-0{,}84, 0{,}37, 0{,}54, 0{,}7]$

(2.6)

En la tabla 2.6 vemos los resultados de la estimación de los parámetros. Los valores estimados siempre respetaron el signo, aunque no lo hicieron en magnitud. Sin embargo, estos valores nos permitieron obtener una segmentación bastante aceptable (Error de segmentación: 12 %)

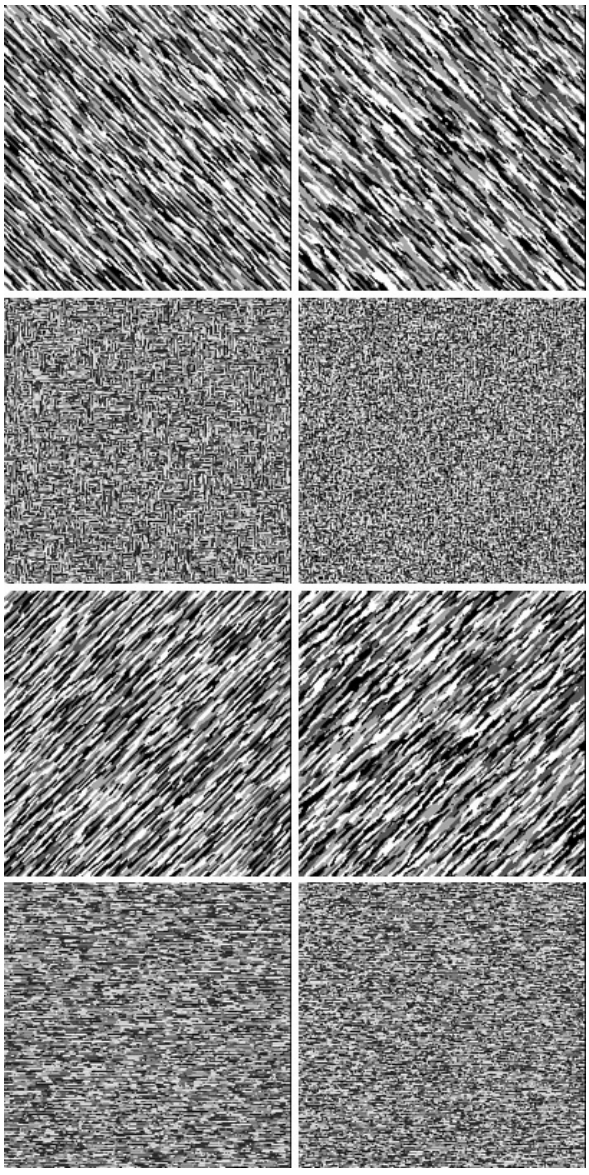


Figura 2-6: **Síntesis de texturas con parámetros estimados.** (Columna izquierda) *Texturas sintetizadas con parámetros conocidos* (Columna derecha) *Texturas sintetizadas con parámetros estimados.*

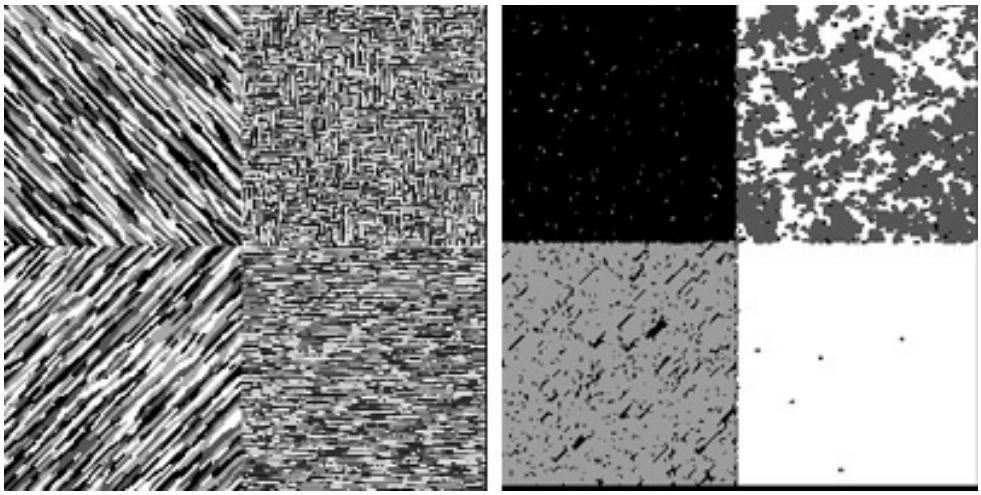


Figura 2-7: **Segmentación utilizando parámetros estimados**: La imagen de la izquierda está formada por texturas sintetizadas con parámetros conocidos. La segmentación de la derecha se realizó utilizando los parámetros estimados.

Capítulo 3

Segmentación utilizando filtros de Gabor

En este capítulo, analizaremos otro enfoque al problema de la segmentación de imágenes texturadas propuesto por Weldon y Higgins [7]. La idea principal es aplicarle a la imagen texturada uno o más filtros que permitan extraer de ella características representativas de las texturas que la componen; para luego clasificar las salidas filtradas mediante un clasificador vectorial.

En la sección (3.1) introducimos los filtros de Gabor; en (3.2) presentamos el modelo estadístico utilizado; en (3.3) explicamos el cálculo del error de segmentación a priori; en (3.4) describimos el proceso de generación del banco de filtros; en (3.5) mostramos los algoritmos implementados; en (3.7) presentamos una mejora al algoritmo original desarrollando un marco de trabajo para procesamiento distribuido; por último en (3.8) mostramos algunos resultados.

3.1. Filtros de Gabor

Los filtros de Gabor recibieron mucha atención, debido a que se demostró que son una muy buena aproximación del comportamiento de ciertas celulas de la corteza visual de algunos mamiferos, las cuales tienen la función de procesar texturas [6]. Además, estos filtros mostraron poseer propiedades de localización óptima en el dominio espacial, siendo esto algo deseable en

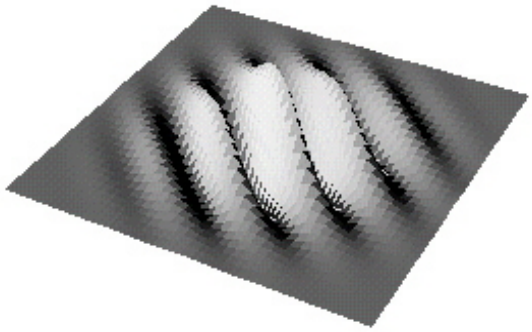


Figura 3-1: Filtro de Gabor (parte real)

los problemas de segmentación de imágenes texturadas. [6]

Sea $I(x,y)$ la imagen de entrada, la cual está formada por N texturas $t_1, t_2, ..., t_N$, con $N \geq 2$. Cuando decimos que la imagen está formada por varias texturas, nos referimos a que la imagen de entrada posee un mapa de texturas subyacente, el que se podría definir como $mapa(x,y) = i$, si en la posición (x,y) de la imagen hay un pixel proveniente de la textura t_i.

A la imagen de entrada se le aplican k filtros, donde cada uno de ellos se compone de un prefiltro de Gabor $h_j(x,y)$, un operador de magnitud $|\cdot|$, y un postfiltro Gaussiano $g_{p_j}(x,y)$. Típicamente, el número de filtros (k) es menor al de texturas (N). El prefiltro de Gabor j tiene como impulso de salida $h_j(x,y)$(3.1). Luego de esto, se utiliza un clasificador para obtener la segmentación definitiva. En la figura 3-2 podemos ver un resumen del proceso.

La fórmula que define al filtro $h_j(x,y)$ es

$$h_j(x,y) = \frac{1}{2\pi\sigma_{g_j}^2} e^{-\frac{\left(x^2+y^2\right)}{2\sigma_{g_j}^2}} e^{-i2\pi(u_j x + v_j y)} \tag{3.1}$$

donde (u_j, v_j) es la frecuencia central del filtro y $\sigma_{g_j}^2$ es el desvío standard de la campana gaussiana que compone el filtro. El prefiltro de Gabor $h_j(x,y)$ tiene una respuesta en frecuencias

$$H_j(u,v) = e^{-2\pi^2\sigma_{g_j}^2\left[(u-u_j)^2+(v-v_j)^2\right]}$$

donde (u,v) es frecuencia espacial.

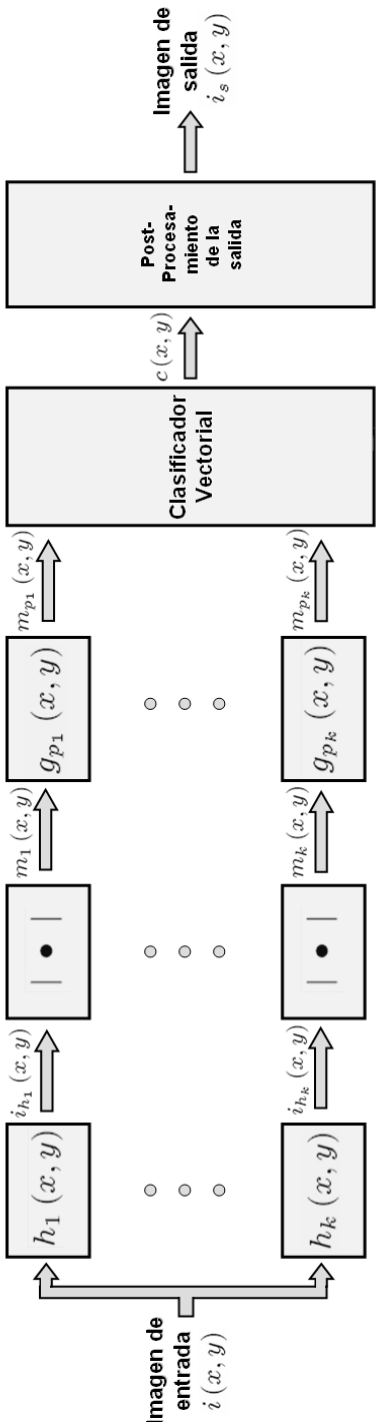


Figura 3-2: Esquema del proceso de segmentación propuesto por Weldon y Higgins. A la imagen de entrada $i(x, y)$, se le aplican k prefiltros $h_j(x, y)$, se toma el módulo de la salida, luego se aplica un postfiltro $g_{pj}(x, y)$, para finalmente correr un clasificador vectorial (para mejorar los resultados, se puede realizar un postprocesamiento de la salida).

La salida del prefiltro $I_{h_j}(x,y)$ es la convolución de la imagen con el prefiltro de Gabor

$$I_{h_j}(x,y) = h_j(x,y) * I(x,y)$$

donde $*$ denota al operador de convolución.

Luego se toma la magnitud (módulo) de la salida del prefiltro de Gabor (el cual es complejo)

$$m_j(x,y) = \left| I_{h_j}(x,y) \right|$$

donde $m_j(x,y)$ tiene, como se muestra en [8],[9] aproximadamente una estadística de Rician para texturas filtradas.

Un postfiltro Gausiano pasabajos $g_{p_j}(x,y)$ es aplicado a $m_j(x,y)$, obteniendo de esta manera la respuesta del j-ésimo filtro, $j \leq k$

$$m_{p_j}(x,y) = m_j(x,y) * g_{p_j}(x,y)$$

con

$$g_{p_j}(x,y) = \frac{1}{2\pi\sigma_{p_j}^2} e^{-\frac{\left(x^2+y^2\right)}{2\sigma_{p_j}^2}}$$

y donde σ_{p_j}, determina el postfiltro Gausiando en el j-ésimo filtro.

En resumen, el proceso toma como entrada la imagen $I(x,y)$, a esta imagen se le aplica por medio de una convolución el filtro de Gabor correspondiente $h_j(x,y)$, se toma módulo a la respuesta para finalmente aplicar un filtro Gaussiano $g_{p_j}(x,y)$, obteniendo

$$m_{p_j}(x,y) = \left| h_j(x,y) * I(x,y) \right| * g_{p_j}(x,y)$$

Consideremos al filtro j como el proceso completo de aplicar sucesivamente el filtro de Gabor $h_j(x,y)$, el operador de módulo, y luego el postfiltro $g_{p_j}(x,y)$. Podemos decir que el vector θ_j formado de la siguiente manera

$$\theta_j = \left(u_j, v_j, \sigma_{g_j}, \sigma_{p_j}\right)$$

define por completo al j-ésimo filtro, ya que:

- u_j, v_j, y σ_{g_j} definen por completo al prefiltro $h_j(x, y)$
- σ_{p_j} defne por completo al postfiltro $g_{p_j}(x, y)$

Sea la siguiente matriz

$$\Theta_k = [\theta_1, \theta_2, ..., \theta_k]^T$$

Θ_k determina los parámetros de los k filtros, donde Θ_k es una matriz de k filas y 4 columnas.

3.2. El modelo estadístico

En estudios previos sobre el diseño de filtros, Weldon presentó un modelo estadístico Gaussiano para la salida del postfiltro $m_{p_j}(x, y)$ [9]. Este modelo estadístico mostró grandes cualidades tanto para predecir el error de segmentación de texturas utilizando un sólo filtro, como para utilizarse como base para el desarrollo de algoritmos de segmentación de un sólo filtro. Este motivo lleva a considerar un modelo Gaussiano multivariado para el caso de la estadística de la salida de los k filtros.

Siguiendo el desarrollo planteado para el caso de un único filtro en [7], el envolvente $m_j(x, y)$ de la imagen prefiltrada tiene una distribución aproximada de Rician. La operación de postfiltrado de cada uno de los k filtros realiza un promedio espacial de la salida del prefiltro, llevandonos a distribuciones Gaussianas para las salidas de los postfiltros $m_{p_j}(x, y)$. La función de densidad Gaussiana multivariada de las salidas de los filtros para la textura de entrada t_i, es entonces

$$p_i(m_p, \Theta_k) = \frac{1}{(2\pi)^{\frac{k}{2}} |C_i|^{\frac{1}{2}}} e^{-\left(\frac{(m_p-\mu_i)^T C_i^{-1}(m_p-\mu_i)}{2}\right)}$$

donde m_p es un vector que representa una muestra k-dimensional de la salida, μ_i es la media y C_i es la matriz de covarianza. Los parámetros para cada filtro j son $\theta_j = (u_j, v_j, \sigma_{g_j}, \sigma_{p_j})$, los cuales corresponden al prefiltro $h_j(x, y)$ y al postfiltro $g_{p_j}(x, y)$.

La determinación tanto de los componentes $\mu_i(j)$ del vector media μ_i, como de la matriz de covarianza C_i se mostrará más adelante en las fórmulas 3.2 y 3.3.

3.3. La medición del error de segmentación

En la sección anterior 3.2, se estableció que el modelo estadístico de la salida del filtro j es Gaussiano multivariado. Según lo explicado en [10], esta distribución sugiere que la distancia de Bhattacharyya proveerá una buena medida sobre la separación de las salidas de los filtros para distintas texturas.

La distancia de Bhattacharyya [10] (o distancia-B), nos permite obtener una medida de "separación" entre dos conjuntos de puntos. En nuestro caso la utilizaremos para obtener una medida de la "separabilidad" (lo que luego se traducirá en una mejor segmentación) de las nubes de puntos obtenidas al aplicarles a las muestras de las texturas, un banco de filtros. En la imagen 3-3 podemos ver las respuestas de tres muestras de texturas a dos filtros que brindan una "buena" separabilidad.

Sean

$$t_i \text{ imagen de muestra de la textura } i$$

$$t_i x \text{ , } t_i y \text{ las dimensiones en filas y columnas de } t_i$$

$$\Theta_k \text{ es el banco de } k \text{ filtros}$$

donde cada posición $\Theta_k(j)$ $1 \leq j \leq k$ es de la forma $(u_j, v_j, \sigma_{g_j}, \sigma_{p_j})$

$$\mu_i(j) = \frac{1}{(t_i x)(t_i y)} \sum_{x=1}^{t_i x} \sum_{y=1}^{t_i y} t_i(x, y) \tag{3.2}$$

es la media de la textura i, con el filtro $j \in \{1..k\}$

$$\sigma_i^2(j) = \frac{1}{(t_i x)(t_i y)} \sum_{x=1}^{t_i x} \sum_{y=1}^{t_i y} (t_i(x, y) - \mu_i(j))^2 \tag{3.3}$$

es el desvío standard de la textura i, con el filtro j

Para el cálculo de la distancia de Bhattacharyya, necesitamos también la matriz de covarianza

$$C_i = \begin{bmatrix} \sigma_i^2(1) & \sigma_i(2)\sigma_i(1) & ... & \sigma_i(k)\sigma_i(1) \\ \sigma_i(1)\sigma_i(2) & \sigma_i^2(2) & ... & ... \\ ... & ... & ... & ... \\ \sigma_i(1)\sigma_i(k) & ... & ... & \sigma_i^2(k) \end{bmatrix}$$

El cálculo de la matriz de covarianza, desde el punto de vista de la implementación, presenta un problema importante en cuanto a su complejidad. Por este motivo se mostró en [9] que si se restringe el conjunto de filtros candidatos de manera que la frecuencia espacial de las respuestas de estos no se superponga significativamente, el efecto combinado o correlación entre ellos puede no tenerse en cuenta, considerando cero para todas las celdas fuera de la diagonal.

Llamemos C_i^* a esta nueva matriz,

$$C_i^* = \begin{bmatrix} \sigma_i^2(1) & 0 & ... & 0 \\ 0 & \sigma_i^2(2) & ... & ... \\ ... & ... & ... & ... \\ 0 & ... & ... & \sigma_i^2(k) \end{bmatrix}$$

Considerando dos texturas t_α y t_β, la distancia de Bhattacharyya entre ellas queda expresada como

$$B\left(t_\alpha, t_\beta, \Theta_k\right) = \frac{1}{8}\left(\mu_\alpha - \mu_\beta\right)^T \left[\frac{C_\alpha^* + C_\beta^*}{2}\right]^{-1} \left(\mu_\alpha - \mu_\beta\right) + \frac{1}{2}\ln\left(\frac{\left|\frac{1}{2}\left(C_\alpha^* + C_\beta^*\right)\right|}{\left|C_\alpha^*\right|^{1/2}\left|C_\beta^*\right|^{1/2}}\right) \tag{3.4}$$

La distancia de Bhattacharyya puede utilizarse para encontrar un límite superior para el error de segmentación de N clases gaussianas multivariadas.

Sea $\varepsilon_c\left(\Theta_k\right)$ el error total de segmentación para todas las N texturas, dado el banco de filtros Θ_k.

$$\varepsilon_c\left(\Theta_k\right) < \sum_{\alpha=1}^{N-1} \sum_{\beta=\alpha+1}^{N} \left(P_\alpha P_\beta\right)^{1/2} \rho_{\alpha\beta}, \text{ con } \rho_{\alpha\beta} = e^{-B\left(t_\alpha, t_\beta, \Theta_k\right)} \tag{3.5}$$

Las ecuaciones 3.4 y 3.5 nos brindan una relación entre el error de segmentación y la distribución Gaussiana multivariada de los vectores de salida de los k filtros. P_α y P_β son las probabilidades a priori de las texturas t_α y t_β. En caso de no contar con información a priori, estas probabilidades pueden tomarse como iguales

$$P_i = \frac{1}{N}, \text{ para } 1 \leqslant i \leqslant N$$

En la práctica se pudo ver que la medida del error en (3.5) es efectiva para el diseño de múltiples filtros cuando el número de texturas es pequeño (< 4). Sin embargo, la performance del banco de filtros diseñado se deteriora rápidamente, a medida que crece la cantidad de texturas. Para mitigar este problema, la ecuación 3.5 es modificada de la siguiente manera:

$$\varepsilon_c\left(\Theta_k\right) \approx \frac{1}{N-1} \sum_{\alpha=1}^{N-1} \sum_{\beta=\alpha+1}^{N} \left(P_\alpha P_\beta\right)^{1/2} \rho_{\alpha\beta} \tag{3.6}$$

donde el límite superor para el peor caso de ε_c es $\frac{1}{2}$.

El error de Bhattacharyya ε_c brinda una medida para la clasificación correcta de las texturas "dentro" de las regiones, pero no brinda solución a la localización de bordes entre regiones.

La segmentación de texturas implica la separación de una imagen en distintas regiones con distintas texturas, con lo cual los problemas en la detección correcta de los bordes de las regiones derivarán en un mayor error de segmentación global.

La función principal de la medida del error de localización es generar un término que favorezca a los filtros con respuestas cuyo componente espacial sea pequeño.

$$\varepsilon_l\left(\Theta_k\right) = \frac{1}{k} \sum_{j=1}^{k} \frac{2N\left(\sigma_{g_j}^2 + \sigma_{p_j}^2\right)}{M^2} \tag{3.7}$$

El término $\left(\sigma_{g_j}^2 + \sigma_{p_j}^2\right)$ de (3.7), aproxima los efectos combinados de localización del prefiltro de Gabor y del postfiltro Gausiano.

Uniendo ambas medidas del error, llegamos a una expresión del error general de seg-

mentación de textura

$$\begin{aligned}\varepsilon_t(\Theta_k) &= \varepsilon_c(\Theta_k) + \varepsilon_l(\Theta_k) \\ &= \sum_{\alpha=1}^{N-1}\sum_{\beta=\alpha+1}^{N}\frac{(P_\alpha P_\beta)^{1/2}}{N-1}e^{-B\left(t_\alpha,t_\beta,\Theta_k\right)} + \frac{1}{k}\sum_{j=1}^{k}\frac{2N\left(\sigma_{g_j}^2+\sigma_{p_j}^2\right)}{M^2}\end{aligned} \tag{3.8}$$

donde P_α y P_β son las probabilidades a priori de ocurrencia de las texturas t_α y t_β en la imagen. La imagen está formada por N texturas, posee dimensiones de $M \times M$ pixels, y $B(t_\alpha, t_\beta, \Theta_k)$ es la distancia de Bhattacharyya entre las texturas t_α y t_β, dado un conjunto de filtros Θ_k.

El error total de segmentación $\varepsilon_t(\Theta_k)$ será utilizado por el algoritmo de selección del banco filtros, que mostraremos en la próxima sección (3.4).

3.4. El Banco de Filtros

Básandonos en el error de segmentación $\varepsilon_t(\Theta_k)$, definido en la ecuación (3.8), procedemos ahora al armado del banco de filtros. En primer lugar, se construye una colección inicial de filtros candidatos Ψ, de los cuales será obtenido el subconjunto de k filtros que se utilizará como banco de filtros para la segmentación:

$$\Psi = \{\theta\} = \{(u, v, \sigma_g, \sigma_p)\} \tag{3.9}$$

tal que:

$$\sigma_g \in \Sigma$$

$$\sigma_p \in \{\lambda\sigma_g | \lambda \in \Lambda\}$$

$$(u, v) \in \left\{\left(\frac{\eta_1}{\sqrt{8\pi^2\sigma_g^2}}, \frac{\eta_2}{\sqrt{8\pi^2\sigma_g^2}}\right)\right\}$$

donde

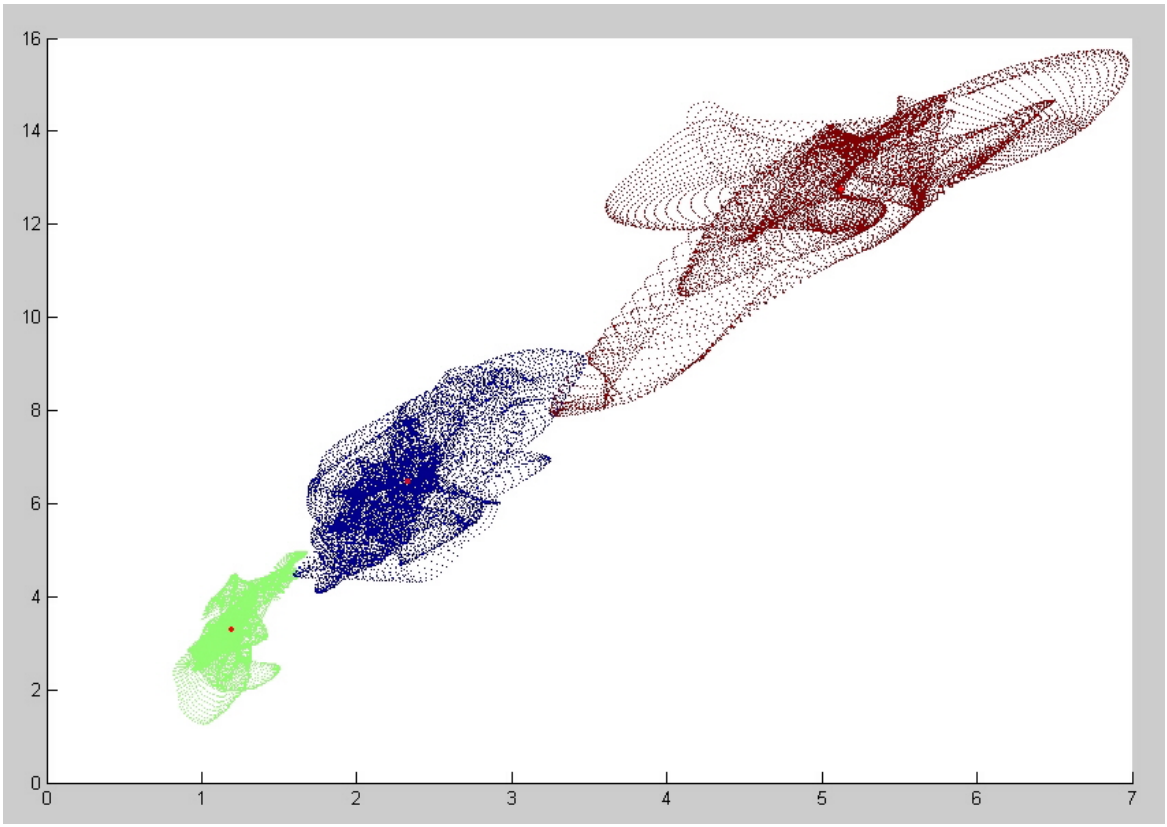


Figura 3-3: **Separabilidad de respuestas a los filtros:** Diagrama de dispersión de las respuestas de tres texturas a los dos filtros con menor error de segmentación a priori. Los puntos rojos son las medias de cada nube (textura).

$$\begin{aligned} \eta_1 &\in \mathbb{Z} \\ \eta_2 &\in \mathbb{N}_0 \end{aligned}$$

$$-0,5 \leq u < 0,5$$

$$0 \leq v < 0,5$$

Σ es el conjunto de valores σ_g para los prefiltros de Gabor, y Λ es un conjunto de constantes que determinan los valores σ_p de los postfiltros, relativos al valor de σ_g. Para cada combinación σ_g y σ_p, la colección de filtros en (3.9), representa un mosaico superpuesto de filtros (overlapping tesselation).

El número total de posibles combinaciones de k filtros, obtenidos del conjunto de candidatos Ψ, $\binom{\#(\Psi)}{k}$, lo que hace prohibitivo su procesamiento para buscar la combinación de filtros con menor error de segmentación estimado. Con lo cual se busca una alternativa que brinde una solución sub-óptima, en un tiempo razonable. Para esto, se utiliza un algoritmo goloso, el cual va seleccionando de manera secuencial, el filtro que brinda el menor error de segmentación estimado, basado en la selección de filtros hasta ese momento.

En este método secuencial, el primer filtro en ser agregado θ_1 al conjunto de seleccionados Θ, es el que posea el menor error de segmentación $\varepsilon_t(\theta_1) \leq \varepsilon_t(\theta_\xi), \forall \theta_\xi \in \Psi$, donde $\theta_1 \in \Psi$. Para explicar como continua el algoritmo, definimos el estado del conjunto de filtros seleccionados en el paso de iteración δ como

$$\Theta_\delta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \dots \\ \theta_\delta \end{bmatrix} = \begin{bmatrix} u_1 & v_1 & \sigma_{g_1} & \sigma_{p_1} \\ u_2 & v_2 & \sigma_{g_2} & \sigma_{p_2} \\ \dots & \dots & \dots & \dots \\ u_\delta & v_\delta & \sigma_{g_\delta} & \sigma_{p_\delta} \end{bmatrix} \tag{3.10}$$

donde $\delta \leq k$, y k es el número de filtros que se desean incorporar al banco de filtros buscado.

Utilizando 3.10 podemos formular el algoritmo de selección de filtros de manera recursiva. El conjunto de filtros seleccionados en el δ-ésimo paso Θ_δ, se puede escribir en función del estado del conjunto hasta el paso anterior $\delta - 1$, es decir $\Theta_{\delta-1}$:

$$\Theta_\delta = \begin{bmatrix} \Theta_{\delta-1} \\ \theta_\delta \end{bmatrix}$$

tal que

$$\varepsilon_t \left(\begin{bmatrix} \Theta_{\delta-1} \\ \theta_\delta \end{bmatrix} \right) \leq \varepsilon_t \left(\begin{bmatrix} \Theta_{\delta-1} \\ \theta_\xi \end{bmatrix} \right), \forall \theta_\xi \in \Psi - \{\Theta_{\delta-1}\} \tag{3.11}$$

donde Θ_δ es función de θ_δ, y donde $\Theta_{\delta-1}$ es una **matriz fija**, la cual fue calculada en el paso $\delta - 1$. El algoritmo finaliza al alcanzar la cantidad de filtros deseados ($\delta = k$) o cuando el error obtenido con el último filtro agregado es menor al error deseado $\varepsilon_t(\Theta) < \varepsilon_{deseado}$. Cabe notar que los filtros se seleccionan sólo una vez, por ese motivo en 3.11, hacemos $\Psi - \{\Theta_{\delta-1}\}$, para eliminar de Ψ los filtros ya elegidos hasta el paso $\delta - 1$.

3.5. El Método

El algoritmo comienza recibiendo como parámetros de ejecución:

I : imagen texturada, formada por N texturas distintas

t_i, con $1 < i < N$, muestras de las texturas que conforman I

$\Sigma \subset \mathbb{N}$ y $\Lambda \subset \mathbb{R}$, parametros para el armado del conjunto de filtros candidatos Ψ

En primer lugar, se arma el conjunto Ψ de filtros candidatos, teniendo en cuenta las restricciones para el armado del mismo. La cantidad de filtros a incluir en Ψ queda a criterio del usuario, dependiendo del tamaño de Σ y Λ.

Las restricciones que deben cumplirse en el armado de Ψ, vistas en la sección 3.4, nos asegurarán la cobertura del espacio de filtros, como así también la baja correlación entre las respuestas (para poder despreciar el efecto combinado) [7].

El método no presenta ninguna restricción al momento de seleccionar los conjuntos Σ y Λ.

Algoritmo 5 *Armado del conjunto de filtros candidatos.*

Parámetros de entrada Σ *y* Λ

$\Psi = \emptyset$

Para cada $\sigma_g \in \Sigma$

Para cada $\lambda \in \Lambda$

$d = \sqrt{8\pi^2\sigma_g^2}$

$\eta_1 = -\frac{d}{2}$

Mientras $\eta_1 \leq \frac{d}{2}$

$\eta_2 = 0$

Mientras $\eta_2 \leq \frac{d}{2}$

$\Psi = \Psi \cup \left\{\left(\sigma_g, \lambda\sigma_g, \frac{\eta_1}{d}, \frac{\eta_2}{d}\right)\right\}$

Fin Mientras

Fin Mientras

Fin Para

Fin Para

Una vez obtenido el conjunto de filtros candidatos, procedemos a calcular para cada combinación de filtro $f_i \in \Psi$ y muestra de textura t_j

$$It_{ij} = f_i\left(t_j\right)$$

donde It_{ij} es el resultado de aplicarle a la muestra de textura t_j el filtro f_i.

Luego necesitamos calcular la media y varianza de It_{ij}.

Sean

$It_{ij}X$ el rango de x (ancho) de la imagen It_{ij}

$It_{ij}Y$ el rango de y (alto) de la imagen It_{ij}

Algoritmo 6 *Cálculo de* μ_{ij} *y* σ_{ij}^2 *para todas las texturas y filtros candidatos*

Para cada muestra de textura t_j

Para cada filtro $f_i \in \Psi$

Obtener It_{ij} *como resultado de aplicarle a* t_j *el filtro* f_i

Obtener $\mu_{ij} = \frac{1}{(It_{ij}X)(It_{ij}Y)} \sum_{x=1}^{It_{ij}X} \sum_{y=1}^{It_{ij}Y} It_{ij}(x,y)$

Obtener $\sigma_{ij}^2 = \frac{1}{(It_{ij}X)(It_{ij}Y)} \sum_{x=1}^{It_{ij}X} \sum_{y=1}^{It_{ij}Y} \left(It_{ij}(x,y) - \mu_{ij}\right)$

Fin Para

Fin Para

La tarea de calcular μ_{ij} y σ_{ij}^2 es muy costosa debido, principalmente, al tamaño del conjunto de filtros candidatos Ψ. Observando en detalle las fórmulas para el cálculo de los mismos detalladas en el algoritmo 6, podemos ver que lo único que se necesita en cada paso del ciclo es conocer al filtro f_i y a la textura t_j. De esto deducimos que esta sección del algoritmo es paralelizable. Por esto (y como explicaremos en la próxima sección 3.7) desarrollamos un marco de trabajo que nos permitió implementar el procesamiento distribuido.

Algoritmo 7 *Armado de banco de filtros*

$\Theta_k = \emptyset$

$FiltrosBuscados = CantidadDeTexturas$

$ErrorBuscado = CotaErrorBuscado$

Mientras $\#(\Theta_k) < FiltrosBuscados$ *y* $MinPredErrSeg > CotaErrorBuscado$

$MinPredErrSeg = MaxInt$ *(inicializo en el mayor valor entero)*

Para cada filtro $f_j \in \Psi,\ f_j \notin \Theta_k$

$PredErrSeg = \sum_{\alpha=1}^{N-1} \sum_{\beta=\alpha+1}^{N} \frac{\left(P_\alpha P_\beta\right)^{1/2}}{N-1} e^{-B\left(t_\alpha, t_\beta, \Theta_k\right)} + \frac{1}{k} \sum_{j=1}^{k} \frac{2N\left(\sigma_{g_j}^2 + \sigma_{p_j}^2\right)}{M^2}$

Si $PredErrSeg < MinPredErrSeg$ *Entonces*

$MejorFiltro = f_j$

$MinPredErrSeg = PredErrSeg$

Fin Si

Fin Para

$\Theta_k = \Theta_k \cup \{MejorFiltro\}$

Fin Mientras

Una vez obtenido el banco de filtros, procedemos a la segmentación propiamente dicha. Para realizar esta tarea podemos utilizar varias técnicas de segmentación las cuales varían en su complejidad.

En primer lugar mostramos el algoritmo de segmentación más simple, el cual asigna a cada posición (x, y) de la imagen, la textura t_j cuyo centro (μ_j) minimiza la distancia euclídea.

Algoritmo 8 *Segmentación de la imagen utilizando la minimización de la distancia euclidea*

Para cada muestra de las texturas $t_j(x, y)$

Para cada filtro $f \in \Theta_k$

$RtaTextura(x, y, f) = AplicarFiltro(f, t_j(x, y))$

Fin Para

$\mu_j = \frac{1}{\#((x,y)\in I)} \sum_{(x,y)\in dom(I)} I(x, y)$

Fin Para

Para cada filtro $f \in \Theta_k$

$Rta(x, y, f) = AplicarFiltro(f, I(x, y))$

Fin Para

Para cada $(x, y) \in dom(I)$

$Segmentacion(x, y) = \arg\min_j distEuclidea\left(Rta(x, y), \mu_j\right)$

Fin Para

Algoritmo 9 *Cálculo de la distancia euclidea* $distEuclidea(X, Y)$

Para cada dimensión i *de* X

$distEuclidea = distEuclidea + (X[i] - Y[i])^2$

Fin Para

$distEuclidea = \sqrt{distEuclidea}$

El segundo algoritmo de segmentación que utilizamos utiliza en lugar de la distancia euclídea, la distancia de Mahalanobis.

Algoritmo 10 *Cálculo de la distancia de Mahalanobis* $distMahalanobis(X, Y, C)$

$distMahalanobis = (X - Y)^T C^{-1} (X - Y)$

Implementamos, aunque sólo para introducirnos brevemente en el tema, un algoritmo simple de postprocesamiento de la salida de la segmentación, el cual básicamente, asigna a cada pixel la etiqueta que más aparece dentro de una ventana cuyo tamaño es especificado por el usuario.

Como la clasificación de los pixels se realiza comparando las distancias del vector de respuestas de cada uno de ellos, hacia las distintas texturas que componen la imagen, consideramos incluir dentro del postprocesamiento el concepto de "*confianza*" de un pixel. Esta confianza se mide como la diferencia que encontró el clasificador para cada pixel entre las distancias a la textura "ganadora", y la distancia hasta la segunda textura. De esta forma podemos saber cuan seguro estuvo el clasificador al momento de asignar una etiqueta. Cuanto mayor sea la diferencia, mayor será la confianza que tendremos a esta asignación de etiqueta.

Decidimos aplicar el algoritmo de postprocesamiento sólo a los pixels cuya confianza fuera menor que cierto umbral definible por el usuario, y tomando en cuenta para la reclasificación sólo las etiquetas de sus vecinos (dentro de la ventana definida por el usuario) de confianza alta.

El tercer método de segmentación que implementamos es el algoritmo de clasificación conocido como K-Means.

3.6. El algoritmo de clustering K-Means

K-Means (o K-medias), es un algoritmo para particionar N puntos del conjunto de entrada en K subconjuntos disjuntos S_j conteniendo N_j puntos cada uno, tal que se minimice el criterio de suma de cuadrados:

$$J = \sum_{j=1}^{K} \sum_{n \in S_j} \left(x_n - \mu_j\right)^2$$

donde x_n es un vector representando el n-esimo punto, y μ_j es el centroide geométrico de los puntos del subconjunto S_j. En general, el algoritmo no alcanza el mínimo global J. Más allá de esta limitación, es utilizado extensivamente ya que brinda resultados aceptables.

K-Means consiste en un procedimiento de reestimación iterativo:

- En el primer paso del algoritmo, los puntos se asignan aleatoriamente a los K conjuntos (o clusters) iniciales.
- El segundo paso es calcular el centroide μ_j para cada uno de los S_j subconjuntos.
- El tercer paso es reasignar a cada punto al conjunto (o cluster) cuyo centroide se encuentra más cerca.

  Los pasos dos y tres se van alternando hasta que se cumple del criterio deseado. Por ejemplo que entre una iteración y la otra la cantidad de puntos reasignados (que cambian de cluster) sea menor a un valor predefinido.

Las principales diferencias con los otros dos métodos presentados son:

- No requiere muestras de las texturas que componen la imagen.
- Sólo requiere conocer de antemano la cantidad de texturas.
- No es determinístico.
- Es iterativo, pudiendo el usuario especificar el criterio de parada.

Algoritmo 11 *Segmentación de la imagen utilizando K-Means*

Para cada filtro $f \in \Theta_k$

$Rta\,(x, y, f) = AplicarFiltro(f, I\,(x, y))$

Fin Para

$cantCiclos = 0$

Para cada $i \in \{1..cantTexturas\}$

$centros[i] = Rta(i, i, 0)$

Fin Para

Mientras $cantCiclos < CiclosMáximos$

// Calculo los Clusters, en base a los centros

Para cada $(x, y) \in dom\,(I)$

$Clusters\,(x, y) = \arg\min_{c \in centros} (Rta\,(x, y) - c)^2$

Fin Para

// Actualizo los centros

Para cada $c \in centros$

$$\text{// Sea } PertCluster\,(x,y,c) = \begin{cases} 1 \text{ si } Clusters\,(x,y) = c \\ 0 \text{ si } Clusters\,(x,y) \neq c \end{cases}$$

$$cantCluster = \sum_{(x,y)\in dom(I)} PertCluster\,(x,y,c)$$

$$c = \frac{1}{cantCluster} \left(\sum_{(x,y)\in dom(I)} PertCluster\,(x,y,c)\, Rta\,(x,y) \right)$$

Fin Para

cantCiclos++

Fin Mientras

3.7. El marco de trabajo distribuido

3.7.1. Generalidades

El procesamiento paralelo es el método de dividir un problema de gran escala en muchos problemas pequeños, procesar cada uno de ellos en un procesador diferente, para luego combinar los resultados obtenidos por cada uno. Año tras año ha ido aumentando la necesidad del paralelismo para proyectos científicos y aplicaciones generales por la demanda de alta performance, bajo costo y gran productividad.

La aceptación del procesamiento paralelo ha sido facilitada por dos grandes desarrollos: los procesadores masivamente paralelos (MPPs) y el uso extendido de computación distribuida.

Actualmente los MPPs son las computadoras más poderosas del mundo. Estas máquinas combinan cientos de CPUs en una gran cabina conectada a cientos de gigabytes de memoria. Los MPPs tienen un enorme y poderoso uso computacional y solucionan grandes problemas, tales como modelos de clima global, o diseño de medicamentos. Este poder de procesamiento crece rápidamente.

El otro gran desarrollo para resolver problemas de gran escala es la computación distribuida. La computación distribuida es un proceso por el cual un conjunto de computadoras conectadas

por una red son usadas conjuntamente para solucionar un gran problema. Como cada vez son más las organizaciones que poseen redes locales de alta velocidad, interconectando una gran cantidad de estaciones de trabajo, la combinación de recursos computacionales pueden exceder el poder de una computadora de alta performance.

A los fines prácticos, una diferencia importante entre los MPPs y el procesamiento distribuido es el costo y simplicidad de implementación. Con el desarrollo tan extendido de las redes de computadoras (y en particular de Internet), la computación distribuida brinda sobre los MPPs una ventaja comparativa muy importante, ya que permite utilizar un conjunto arbitrariamente grande de computadoras "hogareñas" heterogéneas de bajo costo para un fin de procesamiento común.

3.7.2. Nuestra implementación

Como se explica en [7], el tamaño del conjunto de filtros candidatos requeridos, a partir de los cuales se obtiene el banco de filtros debe estar en el orden de los 14000 filtros para obtener una buena calidad de resultados.

Al implementar el algoritmo, pudimos ver que el mayor tiempo de procesamiento se consume en la sección que procesa los filtros cantidatos, calculando para cada uno de ellos y cada una de las texturas el valor de μ_{ij} y σ_{ij}. (Algoritmo 6 de la sección 3.5).

El mismo toma como entrada el conjunto de filtros candidatos expresados como una tupla f_i de cuatro componentes de la forma

$$f_i : (u_i, v_i, \sigma_{gi}, \sigma_{pi})$$

y un conjunto de muestras de las texturas intervinientes

$$t_j, \text{ con } 1 \leq j \leq cantTexturas$$

Este algoritmo presenta una oportunidad importante de distribuir el procesamiento. Para cada uno de los filtros, el procesamiento requerido es totalmente independiente de los otros.

Esto se puede ver en el ciclo interior del algoritmo 6 de la sección 3.5, el cual a grandes rasgos implica:

- Aplicar el filtro f_i a la muestra de la textura t_j, obteniendo $respuesta_{ij}$
- Calcular μ_{ij} de $respuesta_{ij}$
- Obtener σ_{ij} de $respuesta_{ij}$

Esta independencia entre un ciclo y otro presenta la posibilidad de procesar los 14000 filtros requeridos en 14000 equipos distintos en paralelo. Obviamente la idea no es llegar a un nivel de distribución tan grande, pero nos permite ver que cualquier elección de división del conjunto de filtros candidatos nos llevará a resultados correctos.

La arquitectura elegida para implementar el procesamiento distribuido distingue dos tipos de nodos:

- El **Nodo Central** es el encargado de ejecutar el programa principal, pudiendo resumir sus funciones en los siguientes puntos:

  1. generar el conjunto de filtros candidatos
  2. gestionar las conexiones con los **Nodos Remotos** y enviarles la información necesaria (muestras de las texturas y subconjunto de filtros candidatos).
  3. consolidar los resultados de los **Nodos Remotos**
  4. obtener el banco de filtros
  5. segmentar la imagen

- Los **Nodos Remotos** son los encargados de procesar un subconjunto de filtros candidatos, pudiendo resumir sus funciones en:

  1. Conectarse con el **Nodo Central**
  2. Recibir las muestras de las texturas y el subconjunto de filtros candidatos

3. Para cada textura t_j y cada filtro f_i, obtener μ_{ij} y σ_{ij}.

4. Devolver los resultados al **Nodo Central**

La comunicación entre el **Nodo Central** y los **Nodos Remotos** se realiza a traves de sockets, utilizando TCP/IP. Esto permite que los **Nodos Remotos** puedan encontrarse tanto en la misma red local del **Nodo Central**, o bien en cualquier ubicación de Internet.

El protocolo de comunicación elegido para el traspaso de información entre las aplicaciones utiliza XML, el cual tiene la siguiente estructura:

Ejemplo 12 *Mensaje de inicialización desde* ***Nodo Central*** *a* ***Nodo Remoto***

```
<bancoFiltros>
    <filtros>
        <filtro id="0" u="-0,45015823841095" v="0" sg="1" sp="1,5" >
            <textura id="0" mu="" sigma="" />
            <textura id="1" mu="" sigma="" />
        </filtro>
        <filtro id="1" u="-0,45015823841095" v="0" sg="1" sp="2" >
            <textura id="0" mu="" sigma="" />
            <textura id="1" mu="" sigma="" />
        </filtro>
        <filtro id="2" u="-0,45015823841095" v="0" sg="1" sp="2,5" >
            <textura  id="0"  mu=""  sigma="" />
            <textura id="1" mu="" sigma="" />
        </filtro>
        ...
    </filtros>
    <texturas>
        <textura id="0" ancho="128" alto="128" >
            <imagen>[Imagen codificada con Base64]</imagen>
```

```
        </textura>
        <textura id="1" ancho="128" alto="128" >
            <imagen>[Imagen codificada en Base64]</imagen>
        </textura>
    </texturas>
</bancoFiltros>
```

Ejemplo 13 *Mensaje de respuesta del* ***Nodo Remoto*** *a* ***Nodo Central***

```
<bancoFiltros>
    <filtros>
        <filtro id="0" u="-0,45015823841095" v="0"  sg="1" sp="1,5" >
            <textura id="0" mu="2,5" sigma="0,095" />
            <textura id="1" mu="4,3" sigma="1,6" />
        </filtro>
        <filtro id="1" u="-0,45015823841095" v="0"  sg="1" sp="2" >
            <textura id="0" mu="2,5" sigma="0,088" />
            <textura id="1" mu="4,3" sigma="1,098" />
        </filtro>
        <filtro id="2" u="-0,45015823841095" v="0" sg="1" sp="2,5" >
            <textura id="0" mu="2,499" sigma="0,084" />
            <textura id="1" mu="4,302" sigma="0,807" />
        </filtro>
        ...
    </filtros>
    <texturas>
        <textura id="0" ancho="128" alto="128"  />
        <textura id="1" ancho="128" alto="128" />
    </texturas>
</bancoFiltros>
```

Los atributos “id” son los identificadores tanto de las texturas como de los filtros dentro del programa principal. El alto y ancho de las texturas se especifica en pixels.

Como se puede ver en los ejemplos 12 y 13, el mensaje devuelto desde el **Nodo Remoto** es prácticamente igual al de inicialización, completando los valores para los atributos “mu” (μ) y “sigma” (σ) (tampoco se devuelven las muestras de las texturas propiamente dichas, ya que obviamente el **Nodo Central** las conoce).

Como explicamos antes, cuando el **Nodo Central** recibe una conexion de un **Nodo Remoto** (llamémoslo X), le envía un subconjunto de filtros pendientes de procesar, y marca internamente esos filtros indicando que los está procesando el nodo X. Si luego de un tiempo estipulado (timeout configurable) no recibe la respuesta del procesamiento del nodo X, da por finalizada la conexión y vuelve a marcar a los filtros como pendientes de procesamiento, quedando a la espera de recibir una nueva conexión.

En particular el **Nodo Central**, no solo cumple la función de administrar las conexiones con los **Nodos Remotos**, sino que también procesa filtros pendientes.

Para simplificar la implementación inicial, así como el mantenimiento y las mejoras, desarrollamos un sólo programa ejecutable que cumple tanto las funciones de **Nodo Central**, como las de **Nodo Remoto**. Al iniciar el programa, se indica si cumple el rol de Nodo Central o bien se solicita que se indique la dirección IP o URL del nodo al cual se desea conectar.

3.8. Resultados

En las figuras 3-4 a 3-11 mostramos algunos ejemplos de ejecución del algoritmo de segmentación explicado en la sección anterior. Podemos ver que los resultados de la segmentación son mejores en los casos en los que la imagen está formada por menos texturas. Igualmente las segmentaciones con hasta cinco texturas resultaron muy aceptables. Especialmente si se considera que prácticamente no realizamos postprocesamiento.

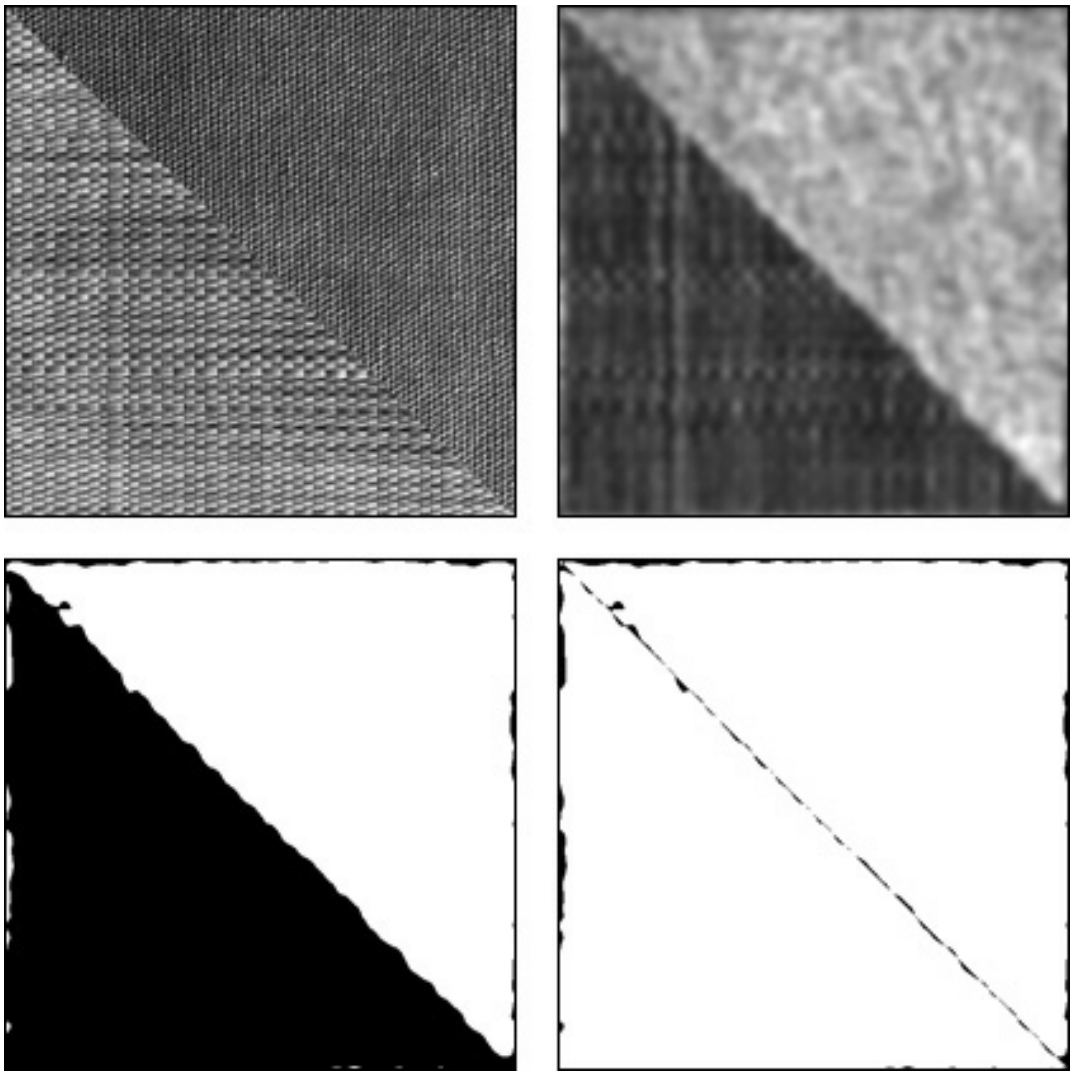


Figura 3-4: **D55_D77_triangulos.raw**: **Fila superior**: (izquierda) Imagen Texturada - Texturas Brodatz D55 y D77. (derecha) Respuesta al filtro seleccionado. **Fila inferior**: (izquierda) Resultado de la segmentación. (derecha) Error de segmentación: 1.4 %

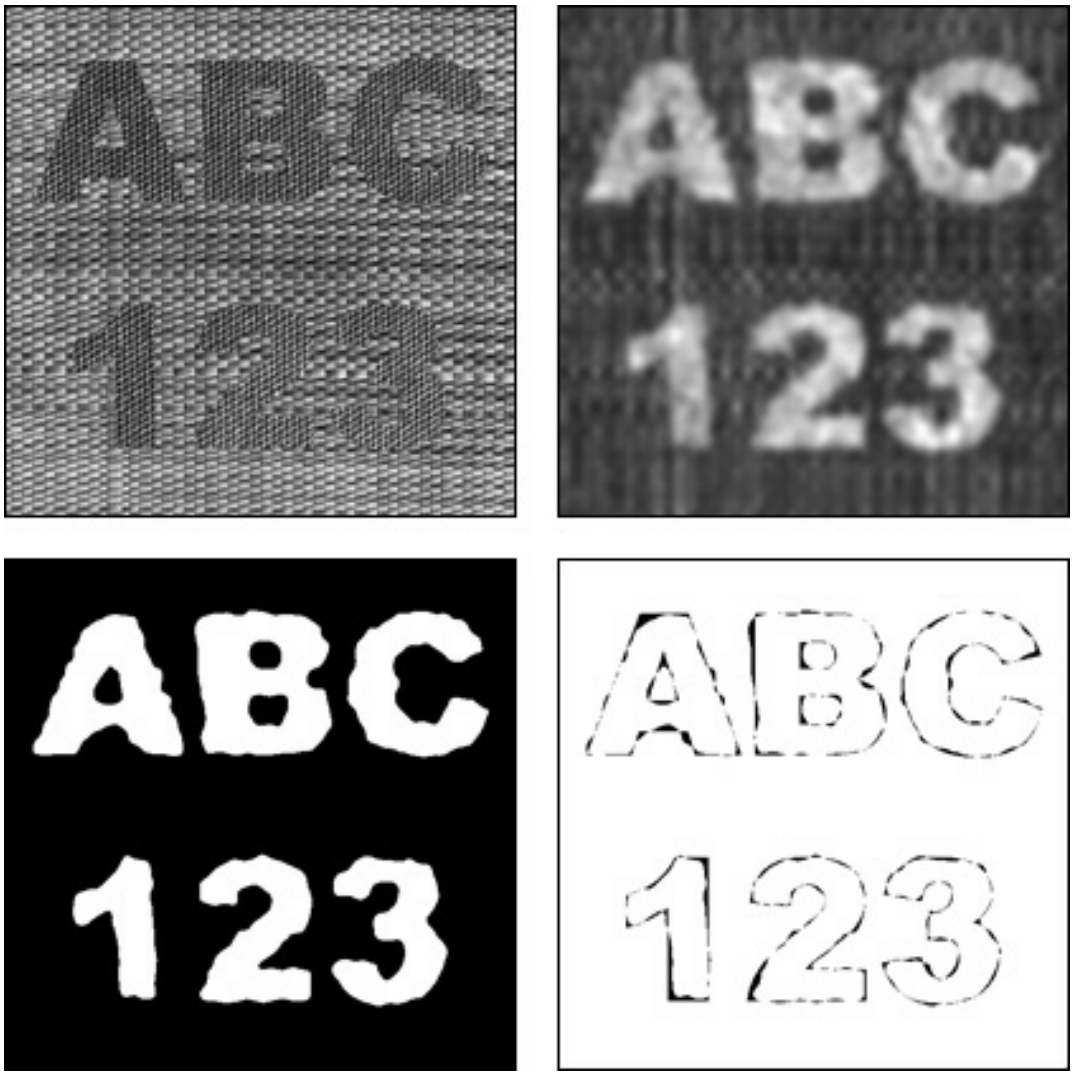


Figura 3-5: **D55_D77_ABC123.raw**: **Fila superior**: (izquierda) Imagen Texturada - Texturas Brodatz D55 y D77. (derecha) Respuesta al filtro seleccionado. **Fila inferior**: (izquierda) Resultado de la segmentación. (derecha) Error de segmentación: 2.7 %

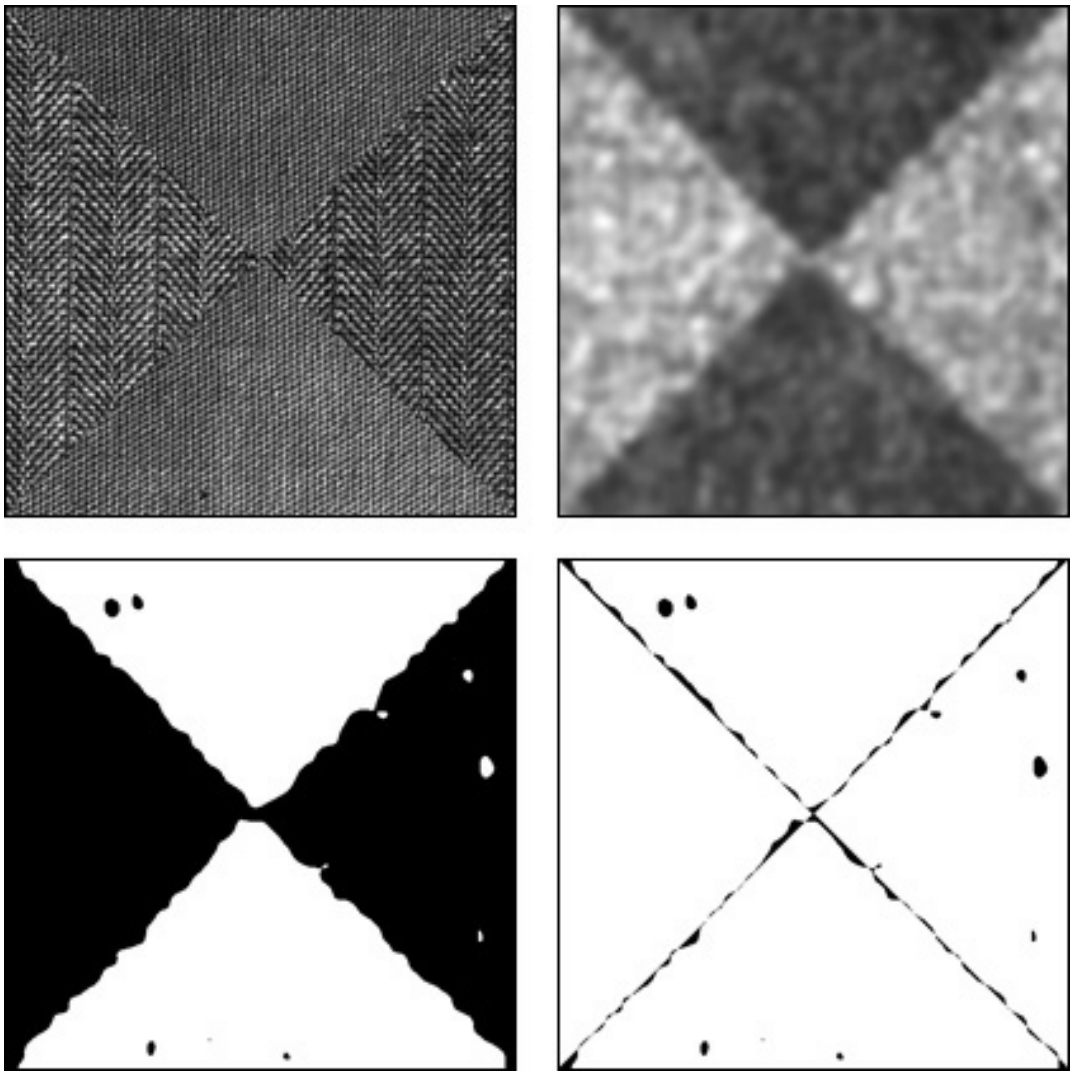


Figura 3-6: **D17_D77_4triangulos.raw**: **Fila superior**: (izquierda) Imagen Texturada - Texturas Brodatz D17 y D77. (derecha) Respuesta al filtro seleccionado. **Fila inferior**: (izquierda) Resultado de la segmentación. (derecha) Error de segmentación: 2.02 %

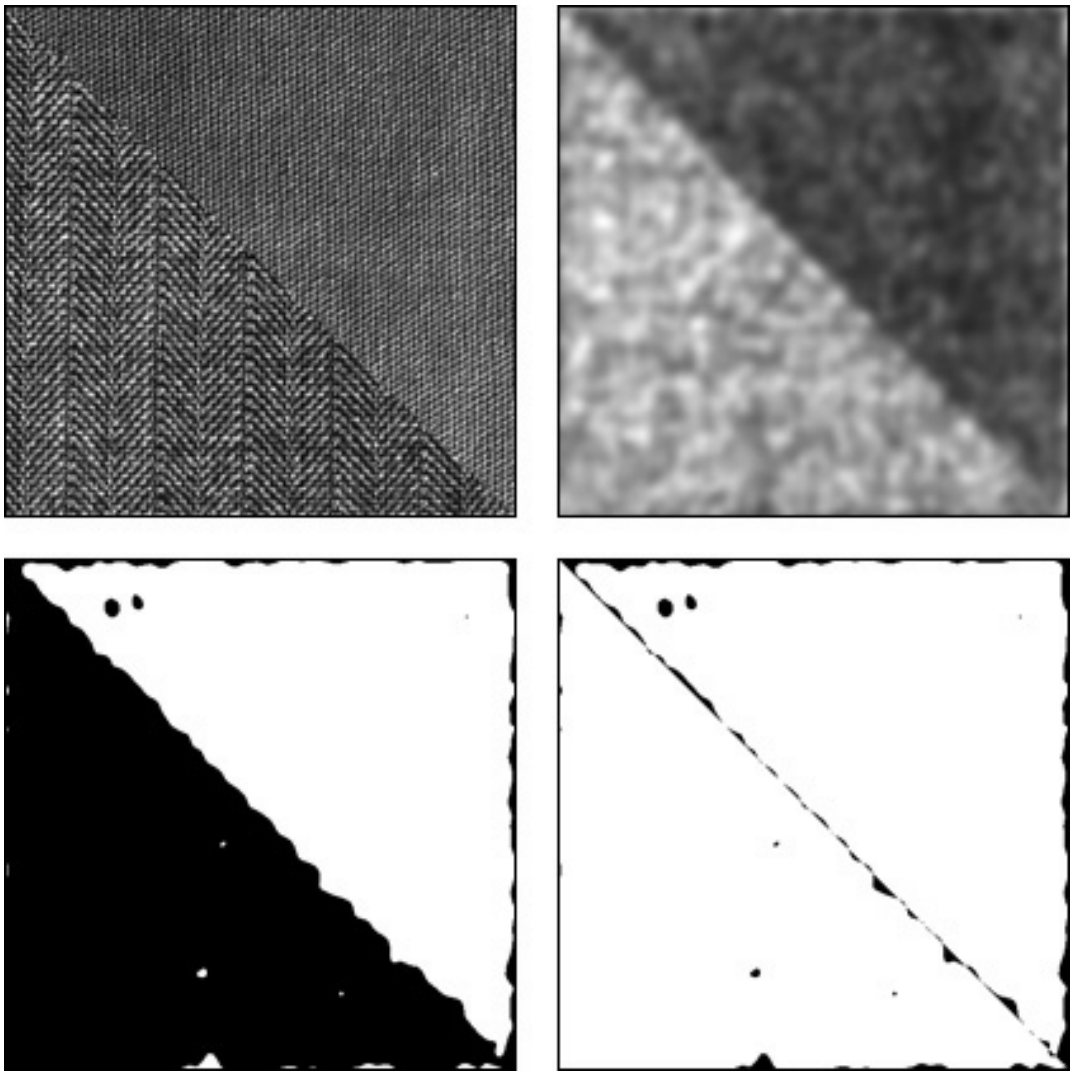


Figura 3-7: **D17_D77_triangulos.raw**: **Fila superior**: (izquierda) Imagen Texturada - Texturas Brodatz D17 y D77. (derecha) Respuesta al filtro seleccionado. **Fila inferior**: (izquierda) Resultado de la segmentación. (derecha) Error de segmentación: 2.2 %

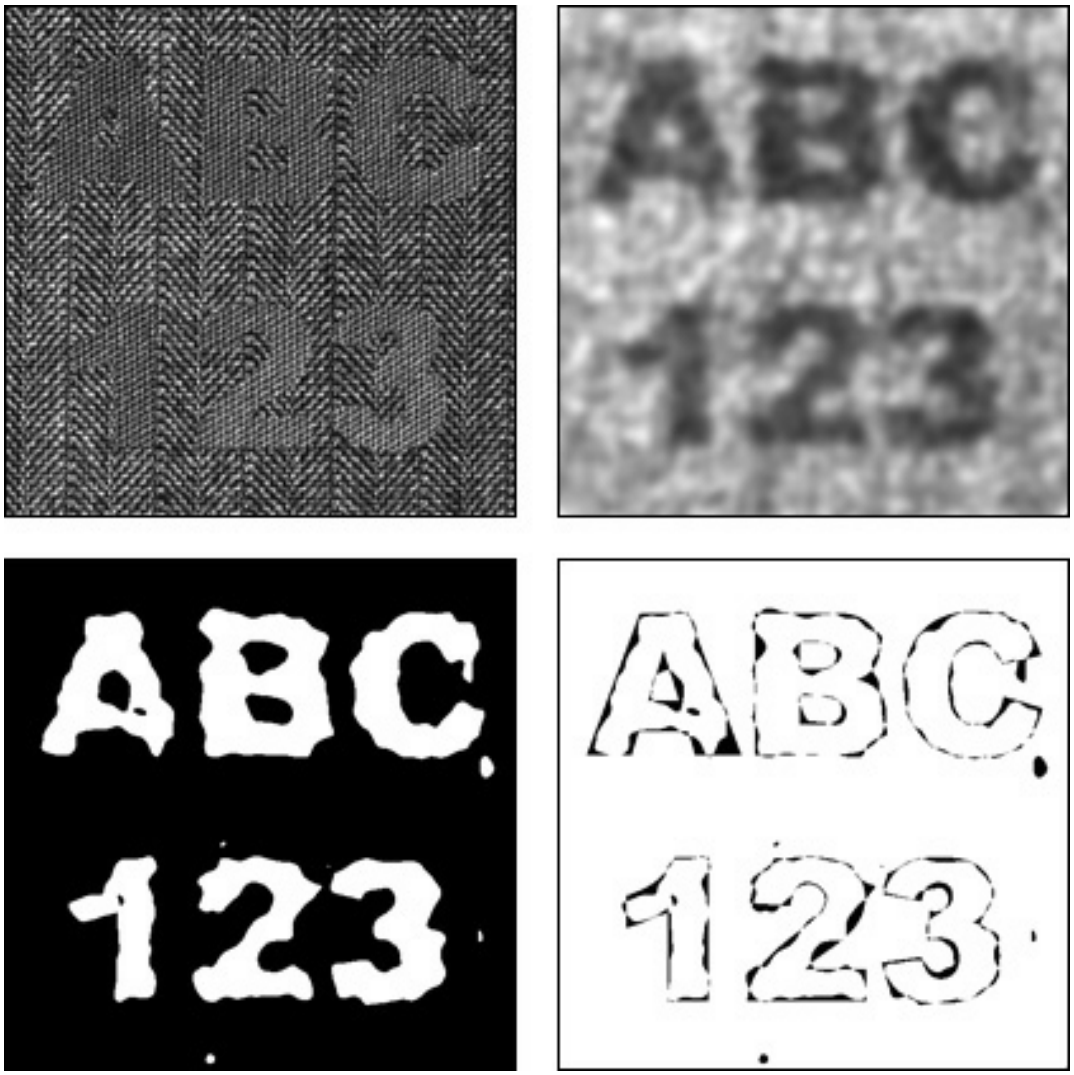


Figura 3-8: **D17_D77_ABC123.raw**: **Fila superior**: (izquierda) Imagen Texturada - Texturas Brodatz D17 y D77. (derecha) Respuesta al filtro seleccionado. **Fila inferior**: (izquierda) Resultado de la segmentación. (derecha) Error de segmentación: 4.6 %

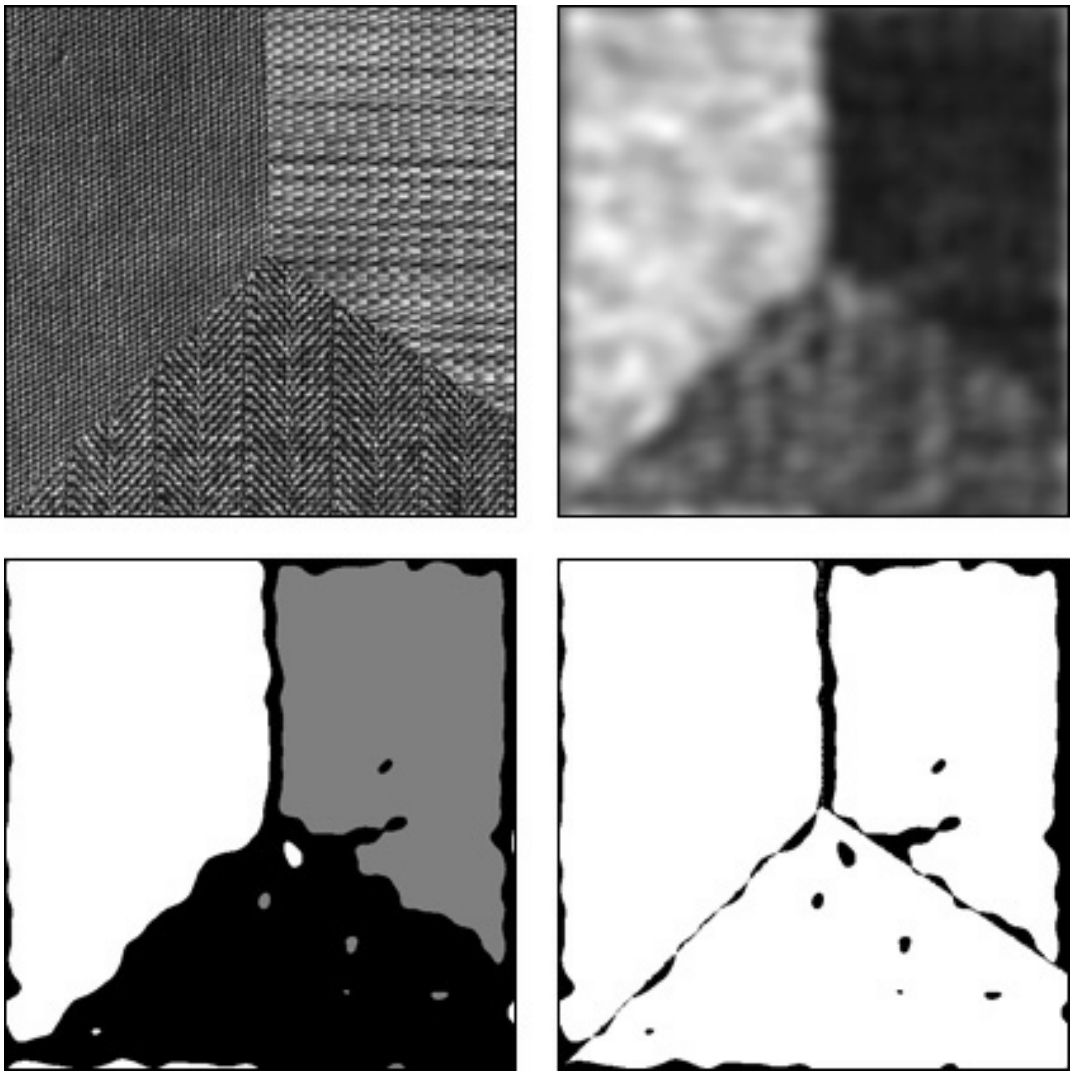


Figura 3-9: **D17_D55_D77_poligono.raw**: **Fila superior**: (izquierda) Imagen Texturada - Texturas Brodatz D17, D55 y D77. (derecha) Respuesta al filtro seleccionado. **Fila inferior**: (izquierda) Resultado de la segmentación. (derecha) Error de segmentación: 6.9 %

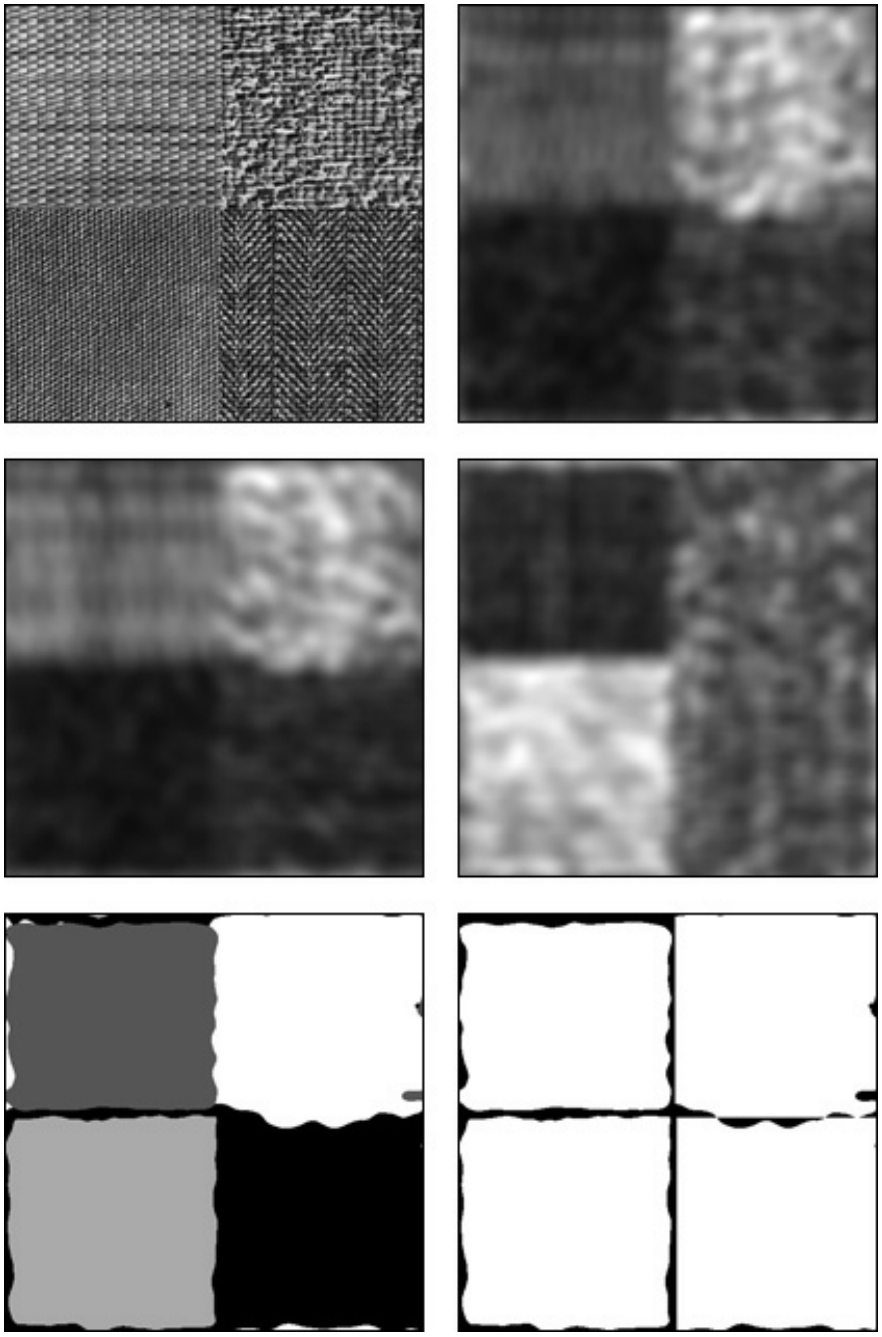


Figura 3-10: **D17_D55_D77_D84_cuadrados.raw**: **Fila superior**: (izquierda) Imagen Texturada - Texturas Brodatz D17, D55, D77 y D84. (derecha) Respuesta al primer filtro seleccionado. **Fila Central**: (izquierda) Respuesta al segundo filtro. (derecha) Respuesta al tercer filtro. **Fila inferior**: (izquierda) Resultado de la segmentación. (derecha) Error de segmentación: 6 %

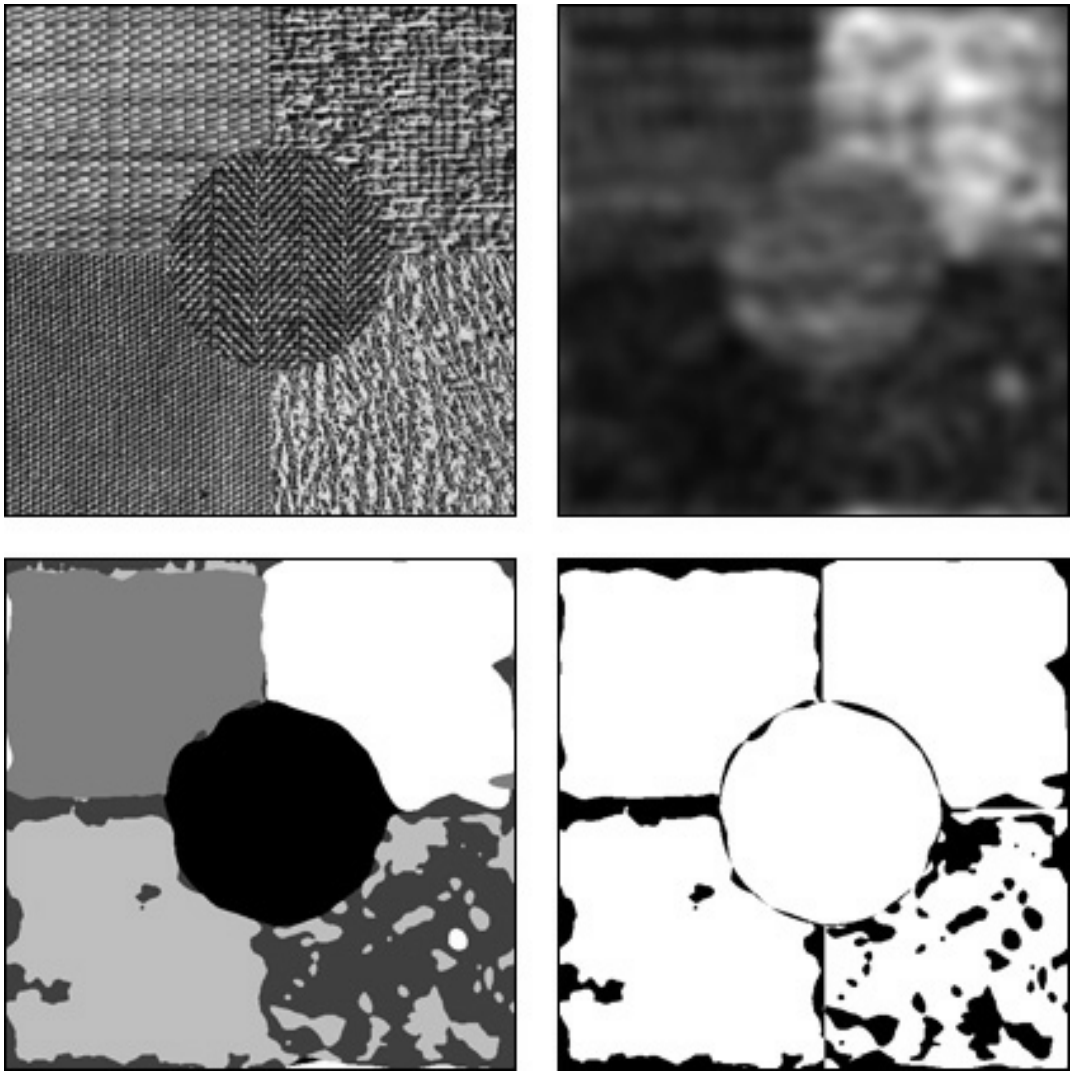


Figura 3-11: **D17_D24_D55_D77_D84_5txt.raw**: **Fila superior**: (izquierda) Imagen Texturada - Texturas Brodatz D17, D24, D55, D77 y D84. (derecha) Respuesta al primer filtro seleccionado. **Fila inferior**: (izquierda) Resultado de la segmentación. (derecha) Error de segmentación: 11 %

3.9. Método con síntesis de Mapa

Como hemos explicado en la sección anterior, la primera parte del algoritmo de segmentación se concentra en el armado del banco de filtros. Esto se realiza seleccionando los mejores filtros en un algoritmo "goloso", de un conjunto de filtros candidatos sobre el espacio de los filtros de Gabor. El criterio de selección de los "mejores" filtros está basado en la minimización, en cada iteración del algoritmo, del error de segmentación *a priori*:

$$\varepsilon_t\left(\Theta_k\right) \approx \sum_{\alpha=1}^{N-1} \sum_{\beta=\alpha+1}^{N} \frac{\left(P_\alpha P_\beta\right)^{1/2}}{N-1} e^{-B\left(t_\alpha, t_\beta, \Theta_k\right)} + \frac{1}{k} \sum_{j=1}^{k} \frac{2N\left(\sigma_{g_j}^2 + \sigma_{p_j}^2\right)}{M^2} \tag{3.12}$$

Como muestra la fórmula 3.12, el cálculo del error está principalmente basado en la distancia de Bhattacharyya entre las respuestas de las texturas al aplicarle los filtros; es decir (como hemos mencionado antes) una estimación del error de segmentación.

Al momento de la implementación del algoritmo, y para poder realizar las pruebas y mediciones, creamos imágenes texturadas, donde el mapa de regiones era conocido por nosotros para poder evaluar la segmentación obtenida. La aplicación tomaba una serie de texturas y mapas, y generaba un conjunto de imágenes combinando las texturas.

Como contábamos con los mapas, pudimos obtener no sólo el error de segmentación como suma de pixels mal clasificados, sino un detalle más pormenorizado de los distintos errores asociados a cada una de las texturas que conformaban las imágenes. Por ejemplo pudimos ver que en una imagen dada, formada por tres texturas, cierto banco de filtros segmentaba muy bien a dos texturas, y tenía problemas con la tercera, cuantificando los errores para cada una de las texturas.

Básicamente los errores que medimos eran, para cada textura t, la cantidad de pixels de la imagen que correspondían a ella y fueron mal clasificados (asignados a otra textura), y la cantidad de pixels que no correspondían a ella que eran asignados a la textura t.

Basados en esta idea, elaboramos un nuevo proceso de selección de filtros, en el cual la clave estaba en hacer que el algoritmo arme (en base a las texturas especificadas por el usuario) una

o más imágenes texturadas, con el mapa subyacente conocido, el cual permitía la evaluación de la segmentación del banco de filtros obtenido en cada paso, mediante la comparación de la segmentación obtenida y el mapa subyacente.

El proceso se podría explicar de la siguiente manera:

- En primer lugar (y de igual manera que en el método anterior), el usuario debe indicar cuales son las texturas incluidas en la imagen texturada y cuales son los porcentajes de aceptación de filtros (se explicará más adelante).

- Como segundo paso (y clave del método) es la generación de los nuevos mapas de texturas en el cual aparezcan todas las texturas indicadas por el usuario.

- Una vez generadas las nuevas imágenes texturadas, comenzamos la búsqueda exahustiva sobre los filtros. Pero a diferencia del método anterior, en el que calculabamos el error *a priori,* en la evaluación de cada filtro, ejecutamos la segmentación de la imagen sintética, aplicando sólo este filtro. De esta manera calculamos el error de segmentación **real** de la misma (ya que el mapa subyacente de la misma es conocido) para cada una de las texturas. Para cada textura, calculamos el porcentaje de pixels de la misma bien clasificados $p_{t,f}^{in}$, y el porcentaje de pixels fuera de la textura bien identificados $p_{t,f}^{out}$:

$$p_{t,f}^{in} = \frac{p_{t,f}^{inOk}}{p_{t,f}^{inTot}}$$

donde $p_{t,f}^{inOk}$ es la cantidad de pixels clasificados como pertenecientes a la textura t, utilizando el filtro f, que efectivamente pertenecen a ella y $p_{t,f}^{inTot}$ es el total de pixels que el mapa sintético asignó a la textura t;

$$p_{t,f}^{out} = \frac{p_{t,f}^{outOk}}{p_{t,f}^{outTot}}$$

$p_{t,f}^{outOk}$ es la cantidad de pixels clasificados como **no** pertenecientes a la textura t, utilizando el filtro f, que efectivamente **no** pertenecen a ella y $p_{t,f}^{outTot}$ es el total de pixels que el mapa sintético asignó al resto de las texturas excluyendo a la textura t.

- Una vez calculados estos valores, los comparamos con los obtenidos hasta el momento y también con los niveles de aceptación indicados por el usuario. Por ejemplo, el usuario podría decir "Acepto un 95 % de $p_{t,f}^{in}$ y $p_{t,f}^{out}$". Con lo cual el algoritmo buscará mejorar en cada paso el filtro seleccionado para clasificar cada textura. Al encontrar uno que satisfaga los parámetros indicados, dejará de buscar.

- El algoritmo se detiene luego de alcanzar los niveles de error deseados para cada una de las texturas, o bien luego de cumplir la evaluación de la cantidad máxima de filtros.

La solución encontrada por este algoritmo, se encuentra en el conjunto de las subóptimas, ya que busca un filtro para cada una de las texturas intervinientes.

También puede verse que la evaluación del mismo para su aceptación está limitada a la clasificación entre la textura que analiza y el resto, es decir que no se están evaluando las interacciones de los mismos (que podrían mejorar su performance).

El algoritmo presenta la posibilidad de "paralelización" de manera relativamente sencilla, ya que perfectamente se puede asignar una tarea de búsqueda a cada nodo de procesamiento. Bastaría con indicarle las texturas, el mapa sintético, el subespacio de búsqueda de filtros (es decir el conjunto de filtros que debería procesar), y los parámetros de aceptación. Luego simplemente deberiamos unir los resultados de los nodos intervinientes para obtener la solución al problema.

Realizamos la implementación de este algoritmo, tanto en monoprocesador, como trabajando en forma distribuida. Con un banco de filtros pequeño, el algoritmo nos brindó resultados aceptables que mostraban que la idea funcionaba, aunque los tiempos de ejecución de la misma eran demasiado extensos como para pensar en procesar un conjunto razonablemente grande de filtros candidatos.

En las primeras versiones de la aplicación, se le informaba al usuario el error a priori de segmentación utilizando la fórmula planteada en 3.8. Idealmente este valor permitiría al usuario obtener una cuantificacion del error de la segmentación obtenida.

Cuando comenzamos a comparar este valor con los errores reales de segmentación, nos dimos cuenta que no eran valores comparables. Es decir, si bien cuando este error era utilizado para comparar los bancos de filtros al momento de seleccionar los mejores se comportaba perfectamente, su valor no representaba el porcentaje de pixels mal clasificados.

Por este motivo, decidimos utilizar lo implementado del mapa sintético como una herramienta de estimación del error de segmentación, cuando no se conoce el mapa real de texturas. Lo cual nos brindó valores muy cercanos a los errores de segmentación reales que obtuvimos.

Capítulo 4

Segmentación sin supervisión

4.1. Introducción

En las primeras secciones de este trabajo estudiamos los campos aleatorios de Markov (MRF). Nos enfocamos en sus propiedades para modelar, sintetizar y finalmente segmentar texturas. Analizamos las características de las texturas con las que se puede trabajar con este tipo de modelos y las diferentes alternativas para estimación de parámetros y segmentación de imágenes. La característica de localidad de estos modelos resultó fundamental en el tratamiento de texturas.

Luego presentamos los filtros de Gabor. Comprobamos que son capaces de captar muy efectivamente características de texturas. Según la bibliografía, estos filtros modelan muy bien el comportamiento del sistema de visión humana encargado de identificar texturas. Por eso es que si se cuenta con el banco de filtros apropiado, se pueden capturar las características de las imágenes que hacen que en nuestra visión se vean diferentes áreas o texturas. El problema en este caso consiste en dar con el o los filtros "apropiados".

El algoritmo de Weldon presentado propone un método de segmentación basado en grandes bancos de filtros de Gabor y en la selección de los mismos en forma ortogonal. Es decir, cada filtro candidato es analizado y se elige el que mejor clasifica o divide el espacio generado por los filtros anteriores. Si bien la implementación de este método puede demandar mucho tiempo de cómputo y recursos de memoria, es paralelizable. En la implementación que presentamos también se

propusieron mejoras y alternativas al método original lográndose muy buenos resultados. Por otro lado se paralelizaron las secciones más complejas del algoritmo permitiendo trabajar con bancos de filtros grandes sin mayores dificultades.

Finalmente analizamos un método que combina ambos enfoques. Gouchol Pok y Jyh-Charn Liu proponen un esquema de segmentación sin supervisión en el cuál se integran los filtros o transformaciones de Gabor y los Campos Aleatorios de Markov (MRF) [16].

4.2. El proceso

El método utiliza, al igual que los antes mencionados, un banco de filtros de Gabor. Pero la dimensión del banco es mucho menor. Los filtros se generan a partir un conjunto de parámetros de frecuencia y otro de orientaciones. En nuestra implementación usamos los siguientes:

$$\begin{aligned} \lambda &\in \left\{\sqrt{2}; 2\sqrt{2}; 4\sqrt{2}; 8\sqrt{2}; 16\sqrt{2}; 32\sqrt{2}; 64\sqrt{2}; 128\sqrt{2}\right\} \\ \theta &\in \{0\,^\circ; 30\,^\circ; 60\,^\circ; 90\,^\circ; 120\,^\circ; 150\,^\circ\} \\ \mu &= 128 \end{aligned}$$

La combinación de estos parámetros genera un banco de 48 filtros. Tanto los valores como la cantidad de parámetros utilizados puede variarse para intentar mejorar los resultados. No encontramos en la documentación un criterio definido para seleccionar los mismos, pero con los presentados anteriormente se obtuvieron resultados razonablemente buenos.

Por cada combinación de λ y θ se calculan u y v para generar el filtro de Gabor

$$\begin{aligned} \arctan(\frac{u}{v}) &= \theta \\ \frac{u}{v} &= \tan(\theta) \\ u &= \tan(\theta)v \end{aligned}$$

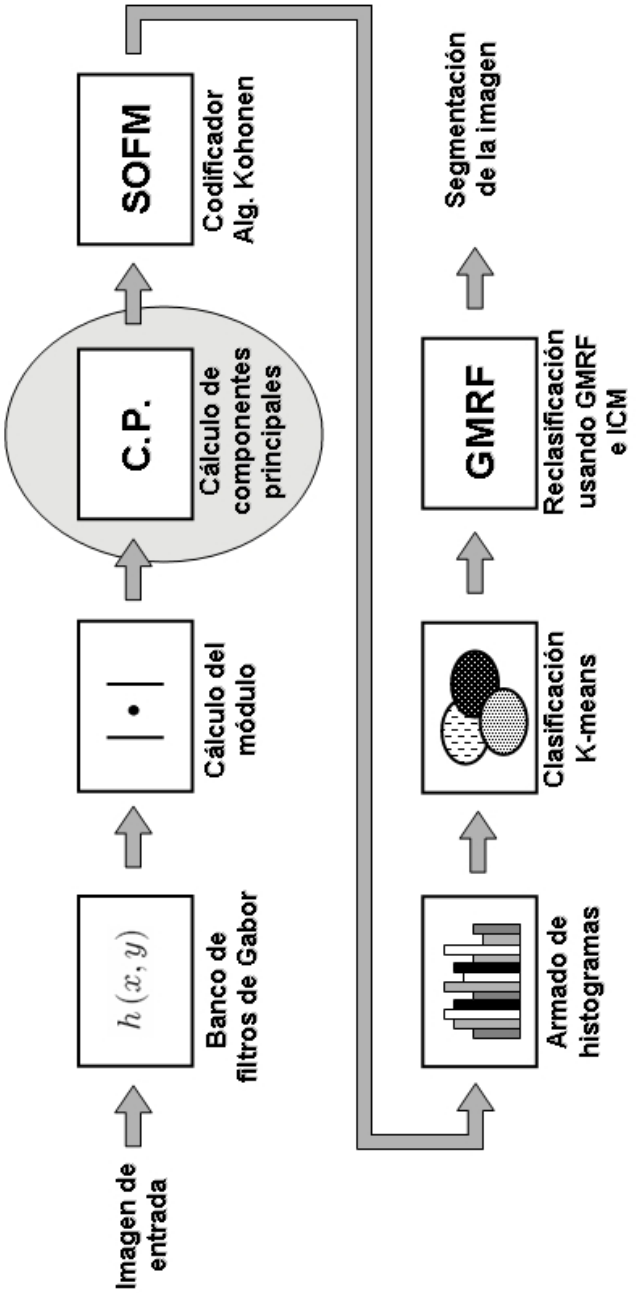


Figura 4-1: Segmentación sin supervisión. Diagrama en bloques del proceso completo

$$\begin{aligned} u^2 + v^2 &= 1 \\ v^2 + (\tan(\theta)v)^2 &= 1 \\ v &= \frac{1}{\sqrt{1 + \tan(\theta)^2}} \end{aligned}$$

$$\sigma = \frac{\mu}{\lambda}$$

La fórmula del filtro de Gabor es la siguiente

$$h(x, y) = \frac{1}{2\pi\sigma^2} e^{2\pi j(ux+vy)}$$

Llamemos N a la cantidad de filtros generada (en este caso 48)

Los filtros de Gabor se utilizan para obtener características de las imágenes o de sus texturas que permitan segmentarlas. Una vez obtenido el banco de filtros, se realiza la convolución de cada uno de ellos con la imagen original

$$V_{\lambda\theta}(x, y) = |I(x, y) * h(x, y)|$$

donde $V_{\lambda\theta}$ es la respuesta de la aplicación del filtro h a la imagen original I.

De esta manera se obtiene un vector de N imágenes de respuesta a los filtros. Estas respuestas podrían usarse directamente para segmentar la imagen original, pero el proceso puede demandar mucho tiempo de procesamiento. A diferencia del algoritmo anterior, los filtros no fueron seleccionados teniendo en cuenta su capacidad de separabilidad, ni se partió de un conjunto grande de los mismos.

El algoritmo se basa en la clasificación de los histogramas de las respuestas de los vecinos de cada pixel, considerando un vecindario de 30x30. Pero antes de generar los Histogramas es necesario codificar de alguna manera las N respuestas que se obtuvieron para cada filtro.

El algoritomo SOFM (Self Organizing Feature Maps) permite codificar las respuestas bajando la dimensión de N a 1 mientras conserva las características espaciales de las respuestas obtenidas.

Implementamos el algoritmo de Kohonen (Implementa SOFM) pero encontramos que la codificación demoraba mucho para las 48 respuestas con las que contamos y por eso decidimos aplicar antes el proceso de obtención de componentes principales.

Este proceso es explicado en la sección 4.3. A través del mismo se logra obtener la información relevante de las N respuestas a los filtros en unas pocas respuestas. Normalmente con dos o tres se logra considerar el 95 % de la información.

El algoritmo de Kohonen (SOFM) se aplica sobre los resultados del proceso de obtención de componentes principales. Como resultado del algoritmo de Kohonen se obtiene una codificación para cada pixel de la imagen. Esta codificación es un valor entero que va de 0 a 127.

En la sección de resultados se muestran las imágenes obtenidas luego del proceso de codificación SOFM.

El siguiente paso es la segmentación de la codificación del algoritmo de Kohonen (SOFM) Para usar la característica de la dependencia local de los sitios de las diferentes texturas, se calculan los histogramas de ventanas de 30x30 centradas en cada sitio. Luego de esto se realiza la segmentación usando los histogramas mediante el algoritmo K-Means.

El input del algoritmo de K-Means puede verse como una imagen en la que el valor de cada pixel es un vector de 128 posiciones. Este vector representa el histograma de las codificaciones de los vecinos.

Finalmente se estiman los parámetros del modelo GMRF (Gaussian Markov Random Field) específico de cada textura y se utilizan para mejorar la clasificación obtenida utilizando el algoritmo ICM.

4.3. Componentes Principales

El Análisis de Componentes Principales (ACP) es una técnica estadística de síntesis de la información, o reducción de la dimensión (número de variables). Es decir, ante un banco de datos con muchas variables, el objetivo será reducirlas a un menor número, perdiendo la menor cantidad de información posible.

Los nuevos componentes principales o factores serán una combinación lineal de las variables originales, y además serán independientes entre sí.

Las principales ventajas del ACP son:

1. Permite representar óptimamente en un espacio de dimensión pequeña observaciones de un espacio general p-dimensional. En este sentido, componentes principales es el primer paso para identificar las posibles variables latentes, o no observadas que generan los datos.
2. Permite transformar las variables originales, en general correlacionadas, en nuevas variables no correlacionadas, facilitando la interpretación de los datos.

En nuestra implementación utilizamos ACP para reducir la dimensión del array de 48 respuestas a los filtros, transformándolas de manera tal que con las dos o tres primeras se contemple el 95 % de la información o variabilidad.

Para el análisis se emplea la *transformada de Hotelling* [17] que se explica a continuación .

4.3.1. La transformada de Hotelling

Consideremos una poblaciónde vectores aleatorios de la forma

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{bmatrix} \tag{4.1}$$

El *vector medio* de esta población se define como

$$m_x = E\{x\} \tag{4.2}$$

donde $E\{\arg\}$ es el valor esperado del argumento, y el subíndice indica que m está asociado con la población de vectores x. Recordemos que el valor esperado de un vector o de una matriz se obtiene tomando el valor esperado de cada elemento.

La *matriz de covarianza* de la población de vectores se define como

$$C_x = E\{(x - m_x)(x - m_x)^T\} \tag{4.3}$$

Dado que x es n dimensional, C_x y $(x - m_x)(x - m_x)^T$ son matrices de orden nxn. El elemento c_{ii} de C_x es la varianza de x_i, la i-ésima componente de los vectores x de la población; y el elemento c_{ij} de C_x es la covarianza entre los elementos x_i y x_j de esos vectores. La matriz C_x es real y simétrica. Si los elementos x_i y x_j no están correlacionados, su covarianza es cero y, por lo tanto, $c_{ij} = c_{ji} = 0$ $(i \neq j)$.

Para M muestras vectoriales de una población aleatoria, el vector medio y la matriz de covarianza se puede aproximar por:

$$m_x = \frac{1}{M}\sum_{k=1}^{M} x_k \tag{4.4}$$

y

$$C_x = \frac{1}{M}\sum_{k=1}^{M} x_k x_k^T - m_x m_x^T \tag{4.5}$$

Como C_x es real y simétrica, siempre es posible hallar un conjunto de n autovalores ortonormales. Sean e_i y $\lambda_i, i = 1, 2, ..., n$ los autovectores y sus autovalores correspondientes de C_x, ordenados en orden decreciente de forma que $\lambda_j \geq \lambda_{j+1}$, para $j = 1, 2, ..., n - 1$.

Sea A una matriz cuyas filas están formadas por los autovectores de C_x, ordenados de forma que la primera fila de A sea el autovector correspondiente al mayor autovalor, y la última fila sea el autovector correspondiente al menor autovalor.

Supongamos que A es una matriz de transformación que transforma las x en otros vectores, a los que llamaremos y, de la forma

$$y = A(x - m_x) \tag{4.6}$$

Esta ecuación se denomina la *transformada de Hotelling*. La media de los vectores y resultantes de esta transformación es cero; es decir:

$$m_y = 0 \tag{4.7}$$

y la matriz de covarianza de las y puede obtenerse en términos de A y C_x por

$$C_y = AC_xA^T \tag{4.8}$$

Además, C_y es una matriz diagonal cuyos elementos de la diagonal principal son los autovalores de C_x; es decir:

$$C_y = \begin{bmatrix} \lambda_1 & & & 0 \\ & \lambda_2 & & \\ & & . & \\ & & & . \\ 0 & & & \lambda n \end{bmatrix} \tag{4.9}$$

Los elementos que están fuera de la diagonal de la matriz de covarianza son 0, por tanto los elementos de los vectores y no están correlacionados. Recordemos que las λ_j son los autovalores de C_x y que los elementos de la diagonal principal de una matriz diagonal son sus autovalores. Por lo tanto, C_x y C_y tienen los mismos autovalores. Lo mismo es válido para los autovectores.

El efecto neto de emplear la *transformada de Hotelling* es establecer un nuevo sistema de coordenadas cuyo origen quede en el centroide de la población y cuyos ejes estén en las direcciones de los autovectores de C_x.

Otra importante propiedad consiste en la reconstrucción de x a partir de y. Debido a que las filas de A son vectores ortonormales, $A^{-1} = A^T$, y todo vector x puede ser reconstruido a

partir de su correspondiente y, empleando la relación:

$$x = A^T y + m_x \tag{4.10}$$

Supongamos, sin embargo, que en lugar de emplear todos los autovectores de C_x se forma una matriz A_k a partir de los K autovectores correspondientes a los K mayores autovalores, dando una matriz de transformación de orden Kxn. Los vectores y serían por lo tanto $K-$dimensionales, y la reconstrucción ya no sería exacta. El vector reconstruido empleando A_k es:

$$\hat{x} = A_K^T y + m_x \tag{4.11}$$

Puede demostrarse que el error cuadrático medio entre x y $\hat{x}$ está dado por la expresión:

$$\begin{aligned} e_{ms} &= \sum_{j=1}^{n} \lambda_j - \sum_{j=1}^{n} \lambda_j \\ &= \sum_{j=K+1}^{n} \lambda_j \end{aligned} \tag{4.12}$$

La ecuación 4.12 indica que el error es cero si $K = n$ (es decir, si se emplean en la transformación todos los autovectores). Dado que los λ_j decrecen monótonamente, la ecuación también demuestra que el error puede ser minimizado seleccionando los K autovectores asociados con los mayores autovalores.

4.4. Self Organizing Feature Map (Mapa de características autoorganizable) - Kohonen

4.4.1. Mapeo de características

Las SOFMs (Self-organizing feature maps; Kohonen Networks) son Redes Neuronales basadas en propiedades topologicas. Tambien son conocidas como Redes Kohonen (Kohonen, 1982;

Fausett, 1994; Haykin, 1994; Patterson, 1996).

El algoritmo de Kohonen consiste en crear una red flexible (de las dimensiones deseadas) con nodos interconectados que se estira y acomoda para cubrir por completo el conjunto de datos que se desea clasificar. Cada uno de los datos a clasificar se identifica con el nodo de la red más cercano a él.

Una caracterítica importante es que la clasificación respeta el orden topológico de los datos. Cada dato es codificado con un valor que identifica al nodo más cercano a él. Luego si dos datos de entrada son codificados con valores diferentes pero cercanos, entonces eran también lo eran en la representación del conjunto de datos. En los ejemplos de la implementación veremos imágenes que describirán claramente esta característica.

En todos los casos la codificación mantiene la estructura subyacente del espacio de entrada reduciendo la dimensión del mismo.

Si por ejemplo, colocamos los datos en una linea (1D) o en un plano (2D), podríamos considerar las redes en las cuales la ubicación de los nodos o unidades ganadoras "*cercanas*", correspondan a datos de entrada "*cercanos*". Si ξ^1 y ξ^2 son dos vectores de entrada, y r^1 y r^2 son las ubicaciones que corresponden a las unidades ganadoras correspondientes, entonces r^1 y r^2 se deberían "acercar" hasta eventualmente coincidir, si ξ^1 y ξ^2 son cada vez más "*parecidos*". Adicionalmente, no deberíamos encontrar $r^1 = r^2$, a menos que ξ^1 y ξ^2 sean similares. Una red que presenta estas propiedades de mapeo es llamada un *feature map*.

Técnicamente lo que estamos buscando es un mapa que preserve la topología del conjunto de entrada. Es decir, un mapeo que preserve las relaciones de vecindad. La idea puede parecer trivial, pero no lo es. Esta dificultad se ve claramente al comprender que este mapeo, no siempre puede encontrarse entre dos espacios. Por ejemplo, no hay forma de mapear un conjunto de datos bidimensionales que forman un círculo, en un espacio unidimensional, sin tener en algún lugar un problema de continuidad.

4.4.2. Algoritmo de Kohonen

El algoritmo de Kohonen brinda una solución al problema del *feature mapping* con un costo computacional relativamente bajo. Este algoritmo normalmente se aplica a las arquitecturas en las que hay entradas con N valores continuos, ξ_1 a ξ_N, los que definen un punto ξ en un espacio continuo N-dimensional. Las unidades de salida O_i estan organizados en un array (generalmente uni o bidimensional), y está totalmente conectado mediante los w_i a las entradas. Es utilizada una regla de aprendizaje competitivo, eligiendo a la unidad ganadora i^* como la unidad de salida cuyo vector de peso es el que se encuentra más cerca de la unidad de entrada actual ξ:

$$|w_{i^*} - \xi| \leq |w_i - \xi| \text{ para todo } i \tag{4.13}$$

La regla de aprendizaje utilizada es

$$\Delta w_i = \eta \Lambda (i, i^*) \left(\xi_j - w_i\right) \tag{4.14}$$

para todo i y todo j. La **función de vecindad** $\Lambda (i, i^*)$ es 1 para $i = i^*$ y va decreciendo con distancia $|r_i - r_{i^*}|$ entre las unidades i e i^* del vector de salida.

Por lo tanto, las unidades cercanas al ganador, así como el ganador i^*, cambian su peso de manera apreciable, mientras que las que se encuentran más lejos, donde $\Lambda (i, i^*)$ es pequeño, sufren un mínimo cambio. Es aquí donde la información topológica es "transferida"; las unidades "cercanas" reciben actualizaciones similares, con lo cual terminan respondiendo como los valores cercanos de la entrada.

La regla 4.14 arrastra al vector de pesos w_{i^*} perteneciente al ganador hacia ξ. Pero también arrastra a los w_i de las unidades cercanas a él. Con lo cual, podríamos pensar en una especie de red elástica en el espacio de entrada, que intenta "estirarse" para cubrir o "parecerse" lo más posible a las entradas; la red tiene la topología del array de salida (1D, 2D, etc.) y los puntos de la red tienen sus pesos como coordenadas.

Para la construcción práctica del algoritmo debemos especificar $\Lambda (i, i^*)$ y η. Resulta útil (y generalmente necesario para lograr convergencia) cambiar estos valores dinámicamente durante

el aprendizaje. Comenzamos con un rango grande para $\Lambda(i, i^*)$ y un η grande, y luego vamos reduciendo gradualmente, tanto el rango "de vecindad" alcanzado por $\Lambda(i, i^*)$, como η. Esto permite a la red organizar su "red elástica" de forma rápida, y luego refinarla lentamente con respecto a los patrones de entrada. Haciendo que η se haga 0, eventualmente detiene el proceso de aprendizaje.

Una típica elección de $\Lambda(i, i^*)$ es

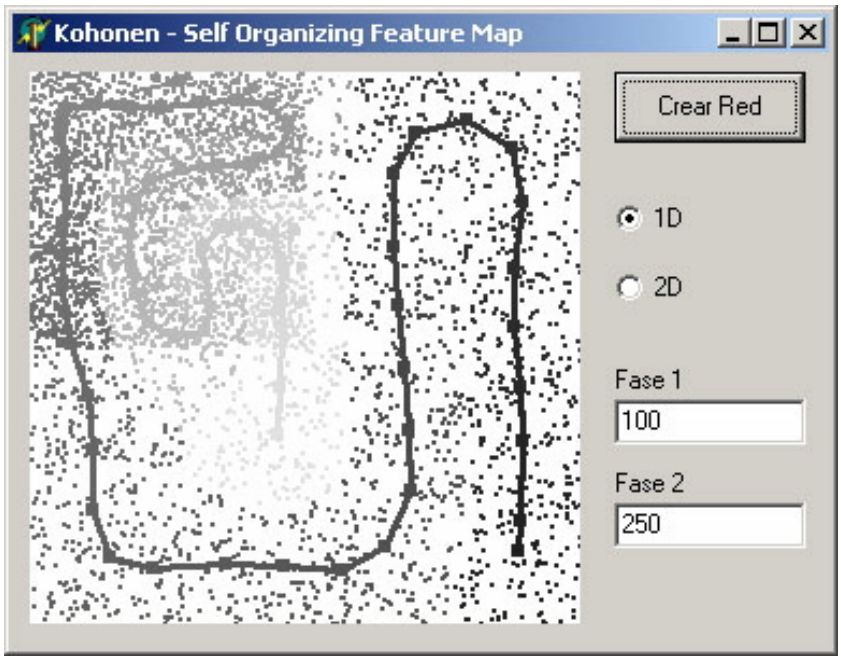
$$\Lambda(i, i^*) = e^{\frac{-|r_i - r_{i^*}|^2}{2\sigma^2}} \tag{4.15}$$

donde σ es el parámetro que maneja el ancho, el cual gradualmente es reducido. La dependencia con el tiempo de $\eta(t)$ y $\sigma(t)$ puede tomar varias formas, incluyendo $\frac{1}{t}$ y $a - bt$, aunque existen razones teóricas para preferir $\eta \propto t^{-\alpha}$ con $0 < \alpha \leq 1$.

4.4.3. Algunos ejemplos de ejecución del algoritmo de Kohonen

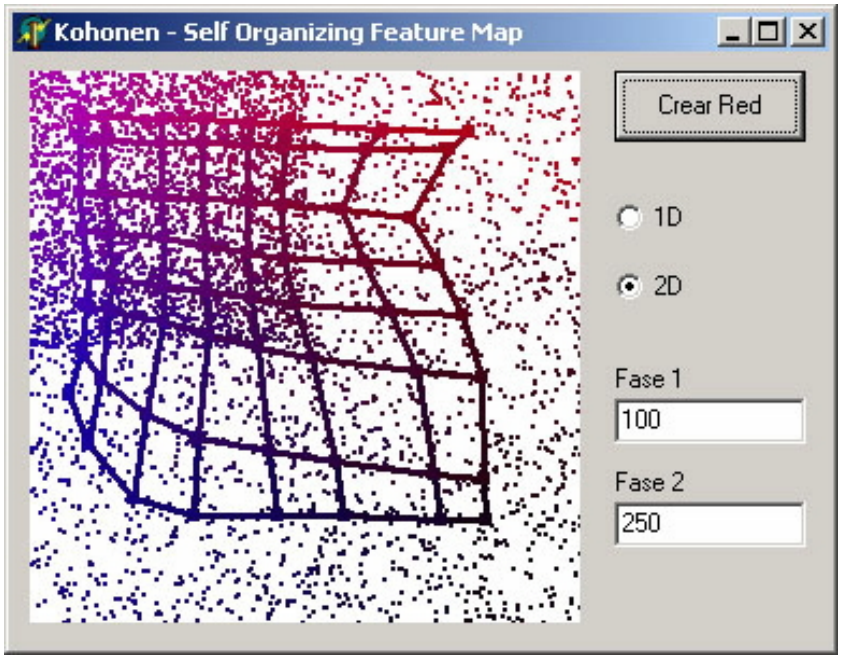
A continuación, y para poder visualizar gráficamente el funcionamiento del algoritmo de Kohonen, mostramos dos ejecuciones del mismo. La primera ejecución muestra como resultado una red unidimensional que mantiene la topología de la muestra expresada con puntos. El segundo ejemplo, muestra una red bidimensional.

La línea representa la unión de los puntos dentro de la red unidimensional obtenida a partir de la muestra de puntos, mediante el algoritmo de Kohonen.



La grilla representa la unión de los puntos dentro de la red bidimensional obtenida a partir de

la muestra de puntos, mediante el algoritmo de Kohonen.



4.5. Ajuste y reclasificación

Para mejorar la segmentación de texturas, se utiliza un modelo de campo aleatorio de Markov Gausiano (GMRF). En el modelo GMRF, la distribución de intensidad para cada clase de textura c está definida como,

$$P\left(X_s = x_s | L_s = c, X_r, r \in N_s\right) = \tag{4.16}$$

$$\frac{1}{\sqrt{2\pi\sigma^2(c)}} \exp\left[-\frac{1}{2\sigma^2(c)}\left[x_s - \sum_{r\in N_s} \theta_r(c) x_{s+r}\right]^2\right]$$

La distribución de las etiquetas también es modelada por un MRF,

$$P\left(L_s = c | L_r, r \in N_s\right) = \frac{1}{Z} \exp\left[\gamma U(c)\right] \tag{4.17}$$

donde $U(c)$ es el número de vecinos cuya etiqueta coincide con c, Z y γ son constantes de normalización.

Estimamos los parametros θ y σ^2para cada textura con sus estimadores de máxima verosimilitud

$$P(X_s = x_s \mid L_s = c, X_r, r \in N_s) = \frac{1}{\sqrt{2\pi\sigma^2(c)}} \exp\left[-\frac{1}{2\sigma^2(c)}\left[x_s - \sum_{r\in N_s} \theta_r(c) x_{s+r}\right]^2\right]$$

Definimos la función de Verosimilitud asumiendo que los sitios de cada textura o clase son i.i.d.

$$L\left(x_1, x_2, ..., x_n, f\left(x_s\right)\right) = \prod_{s\in S_c} f\left(x_s\right)$$

$$\ln L = \sum_{s\in S_c} \ln\left(f\left(x_s\right)\right) =$$

$$\sum_{s\in S_c}\left[-\frac{1}{2}\ln 2\pi - \ln\sigma(c) - \frac{1}{2\sigma^2(c)}\left[x_s - \sum_{r\in N_s}\theta_r(c)x_{s+r}\right]^2\right]$$

$$\theta\left(c\right)^* = \arg_{\theta(c)} \text{máx}\, L$$

$$\sigma^2\left(c\right)^* = \arg_{\sigma^2(c)} \text{máx}\, L$$

Calculamos el estimador de $\theta_r(c)$ como

$$\frac{\partial \sum\limits_{s\in S_c} \ln\left(f\left(x_s\right)\right)}{\partial\theta_r(c)} = 0 = \sum_{s\in S_c} -\frac{1}{2\sigma^2(c)} 2\left[x_s - \sum_{r\in N_s}\theta_r(c)x_{s+r}\right]\left[-\sum_{r\in N_s} x_{s+r}\right]$$

Asumiendo $\sigma^2(c) \neq 0$, reescribiendo θ como vector columna, q_s como el vector de ocho vecinos de s y llamando $v = \sum\limits_{s\in S_c}\left[q_s q_s^T \mathbf{1}\right]$

$$\sum_{s\in S_c}\left[x_s\sum_{r\in N_s}x_{s+r}\right]v^T\left[vv^T\right]^{-1}=\theta^T(c) \tag{4.18}$$

Calculamos el estimador de $\sigma^2(c)$ como

$$L=\sum_{s\in S_c}\left[-\frac{1}{2}\ln 2\pi-\ln\sigma(c)-\frac{1}{2\sigma^2(c)}\left[x_s-\sum_{r\in N_s}\theta_r(c)x_{s+r}\right]^2\right]$$

$$\frac{\partial}{\partial\sigma^2(c)}=0=\sum_{s\in S_c}\left[-\frac{1}{2}\frac{1}{\sigma^2(c)}-\frac{1}{2}\left[x_s-\sum_{r\in N_s}\theta_r(c)x_{s+r}\right]\right]$$

$$\sigma^2(c)=\frac{1}{M}\sum_{s\in S_c}\left[x_s-\sum_{r\in N_s}\theta_r(c)x_{s+r}\right]^2 \tag{4.19}$$

$$\text{donde } M=\#\left(S_c\right)$$

El desarrollo completo de la obtención de ambos estimadores puede verse en el apéndice A.

Estos parámetros son estimados sobre los pixels cuyos ocho vecinos tienen todos la misma etiqueta que ellos. Cabe notar que esto es realizable, ya que el primer proceso de "etiquetado" nos da como resultado una imagen inicial, con regiones bastante compactos.

4.5.1. El proceso de ajuste

Ahora debemos integrar los dos procesos definidos por las equaciones 4.16 y 4.17. Para esto utilizamos el método "Iterative Condition Mode" (ICM), como un estimado de la solución MAP de la distribución de etiquetas,

$$\arg\underset{L_s}{\text{máx}}P\left(L_s|L_r,X_s,X_r\right)=\arg\underset{L_s}{\text{máx}}P\left(X_s|L_r,L_s,X_r\right)P\left(L_s|L_r\right) \tag{4.20}$$

Maximizar la probabilidad condicional en 4.20 es equivalente a minimizar la función de

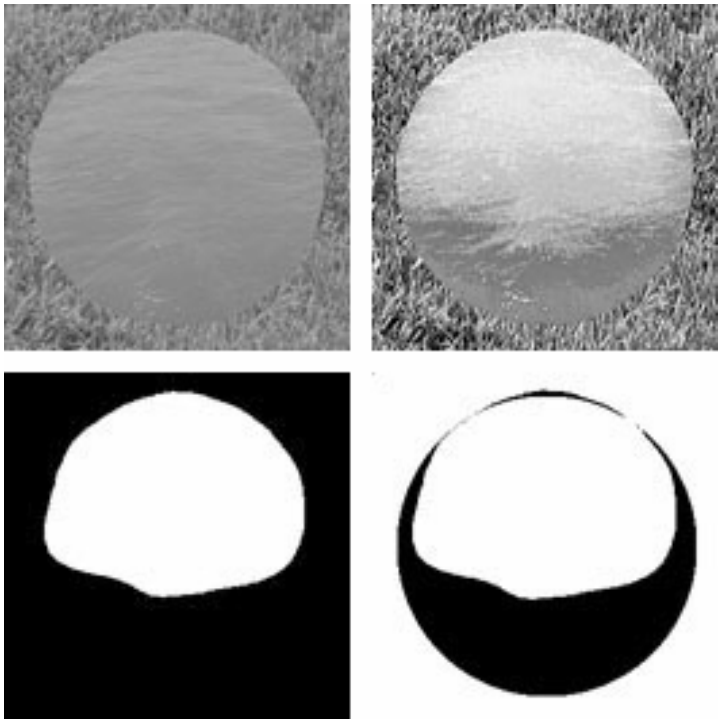


Figura 4-2: PastoAgua_Circulo.raw. **Arriba**. Izquierda: Imagen Original. Derecha: Resultado SOFM. **Abajo**. Izquierda: Segmentación. Derecha: Error de Segmentación.

energía obtenida al tomar logaritmo de las ecuaciones 4.17 y 4.16,

$$\arg\min_{L_s}\left[\log\left[\sigma\left(c\right)\right]-\gamma U\left(L_s=c\right)+\frac{1}{2\sigma^2\left(c\right)}\left[x_s-\sum_{r\in N_s}\theta_r\left(c\right)x_{s+r}\right]^2\right]$$

4.6. Algunos Resultados

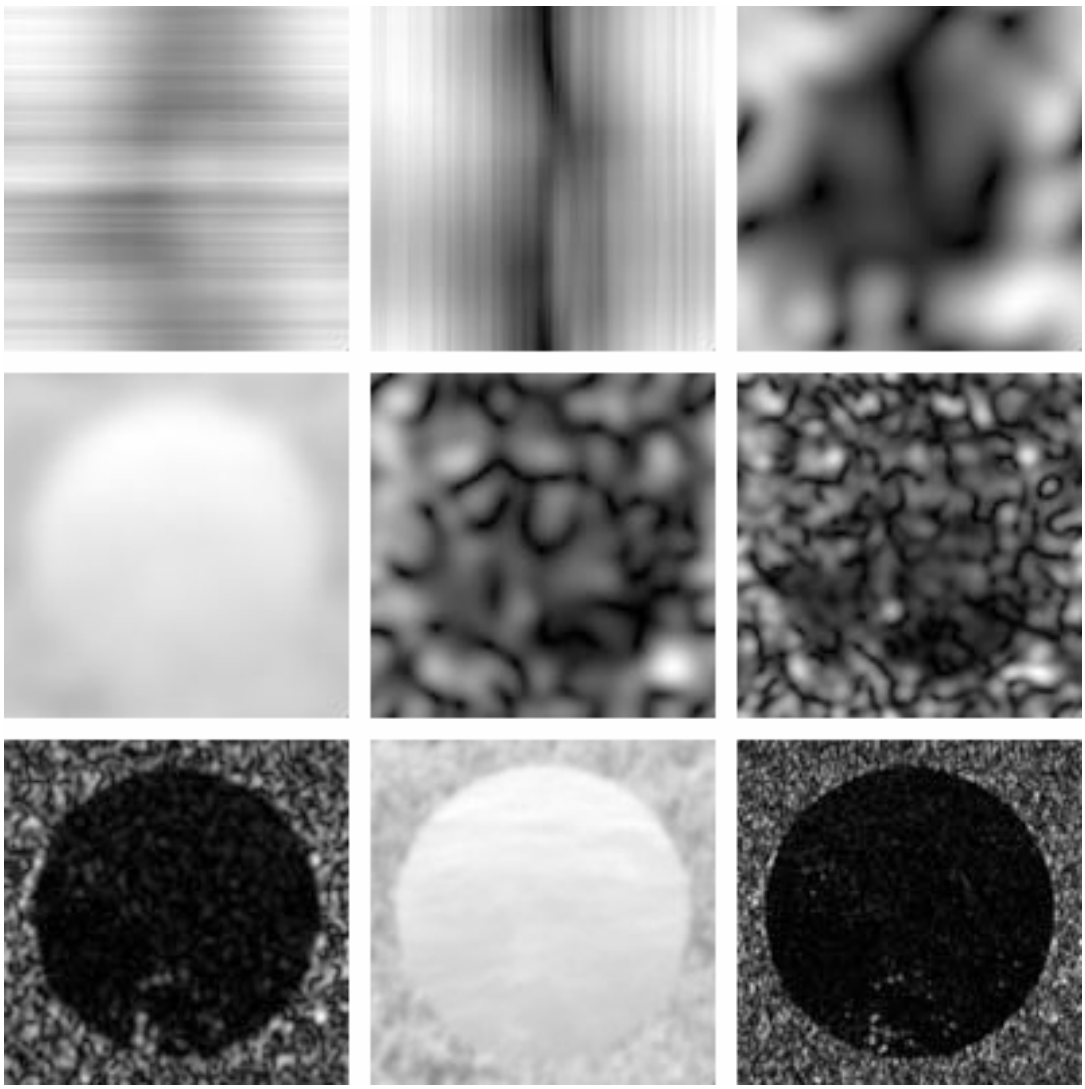


Figura 4-3: PastoAgua_Circulo. Algunas respuestas a los filtros.

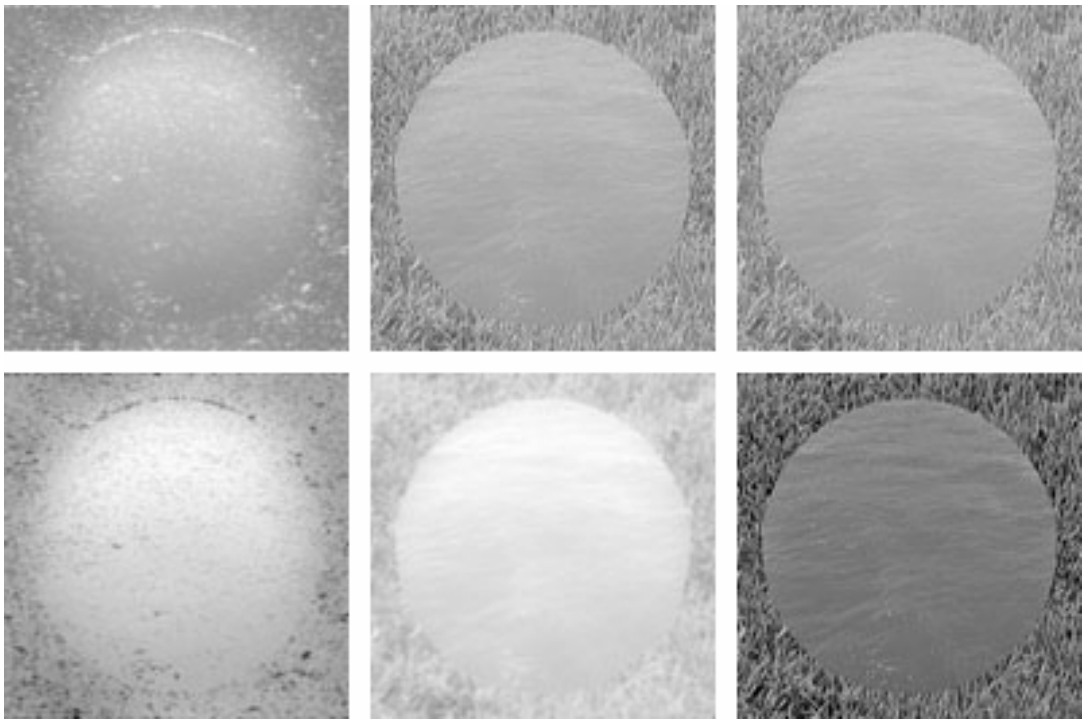


Figura 4-4: PastoAgua_Circulo. Componentes Principales - **Arriba de izquierda a derecha:** λ = 2307,058 88,148 %; λ = 206,613 7,894 %; λ = 54,428 2,079 %. **Abajo de izquierda a derecha:** λ = 20,873 0,797 %; λ = 13,425 0,512 %; λ = 7,284 0,278 %

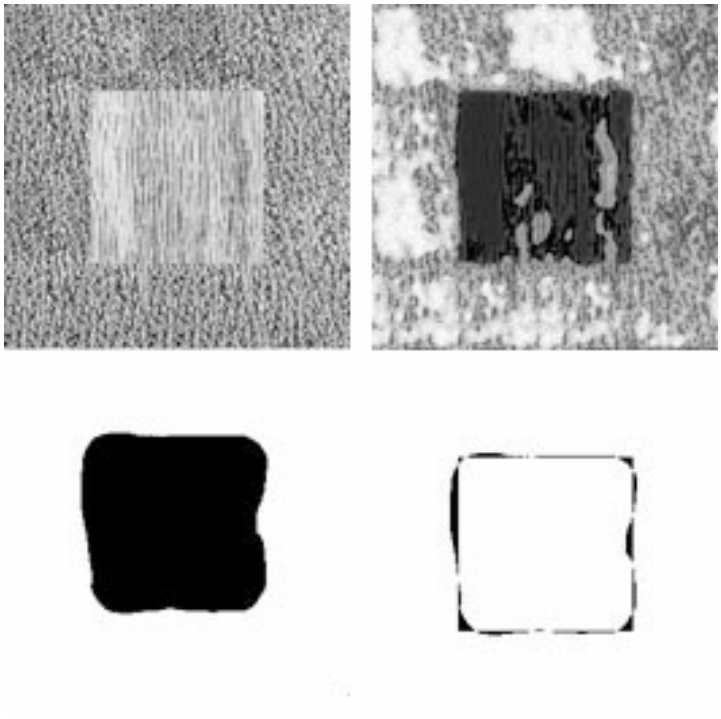


Figura 4-5: **Arriba**. Izquierda: Imagen Original. Derecha: Resultado SOFM. **Abajo**. Izquierda: Segmentación. Derecha: Error de Segmentación.

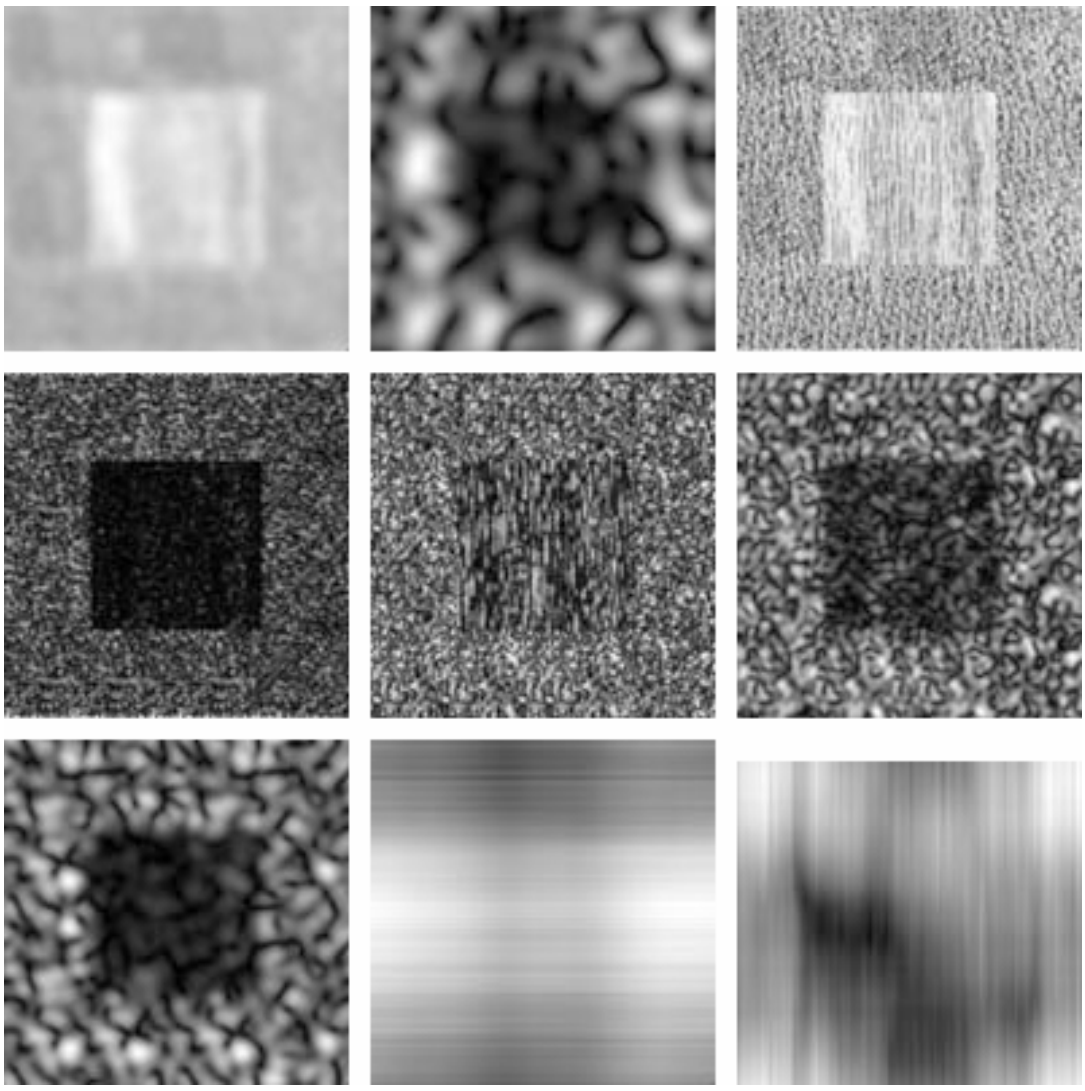


Figura 4-6: Algunas respuestas a los filtros

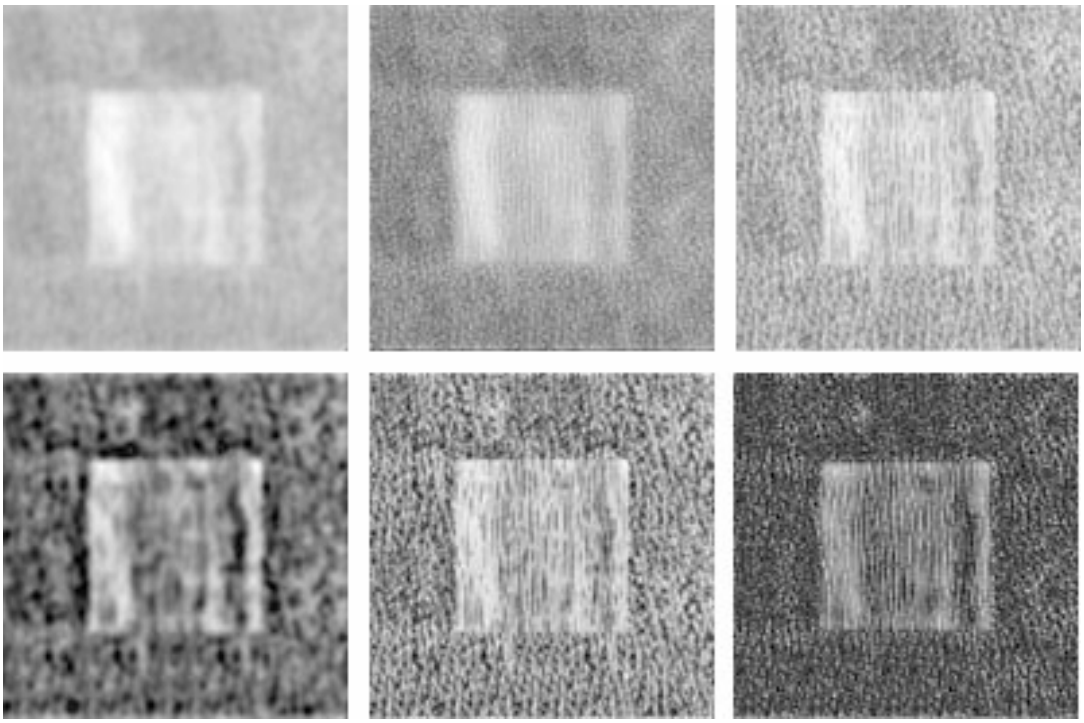


Figura 4-7: Componentes Principales - **Arriba de izquierda a derecha:** λ = 6014,925 71,238 %; λ = 1358,449 16,088 %; λ = 478,820 5,670 %. **Abajo de izquierda a derecha:** λ = 252,209 2,987 %; λ = 155,237 1,838 %; λ = 67,755 0,802 %

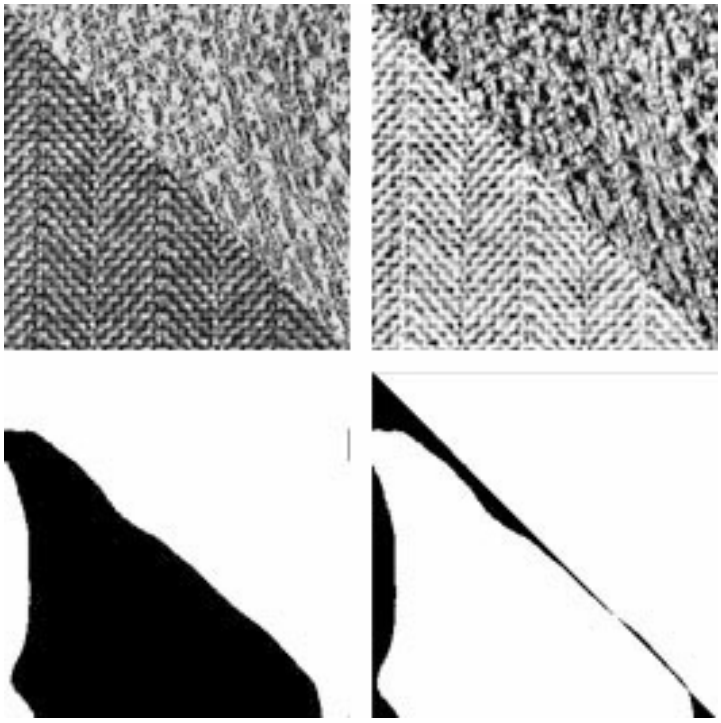


Figura 4-8: **Arriba**. Izquierda: Imagen Original. Derecha: Resultado SOFM. **Abajo**. Izquierda: Segmentación. Derecha: Error de Segmentación.

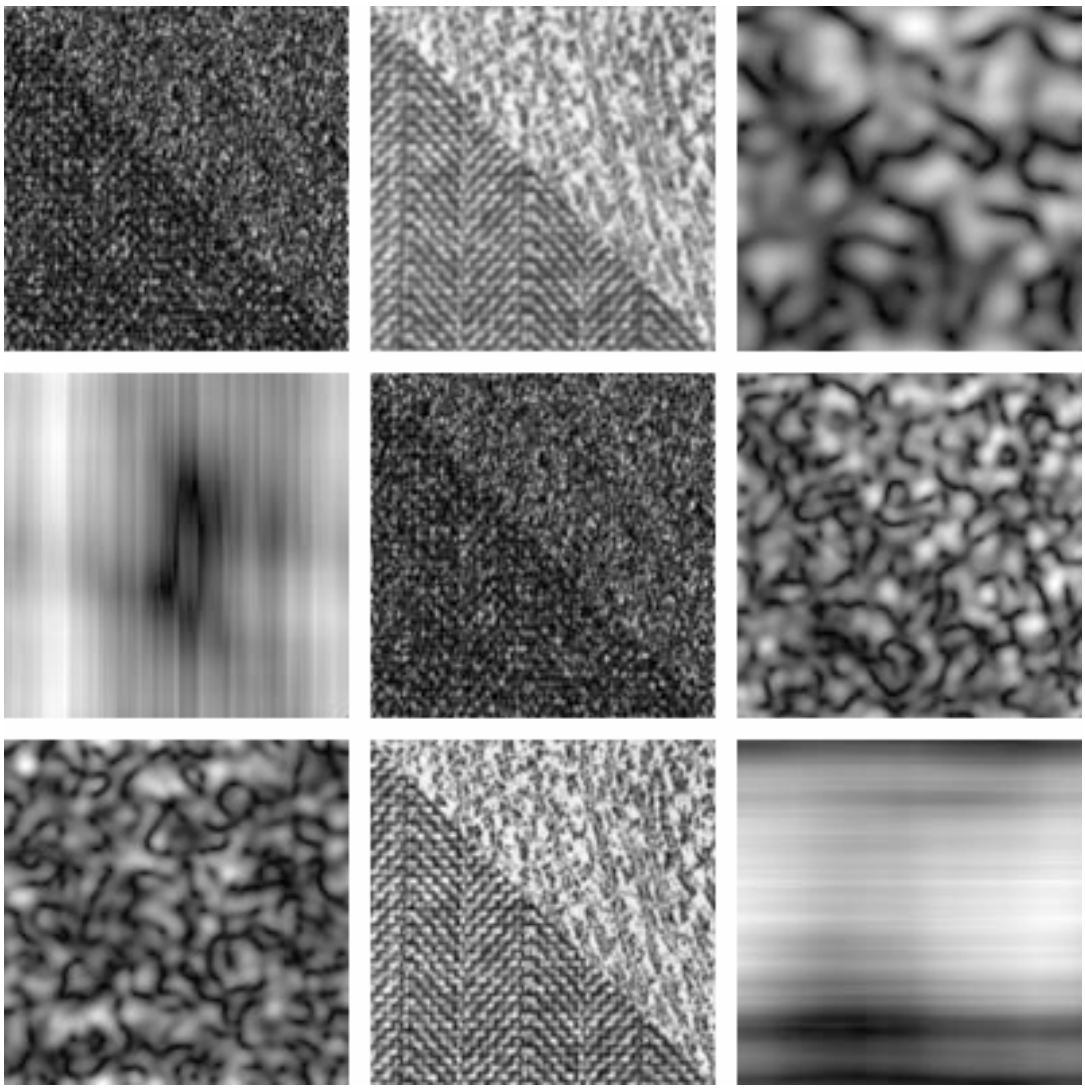


Figura 4-9: Algunas respuestas a los filtros

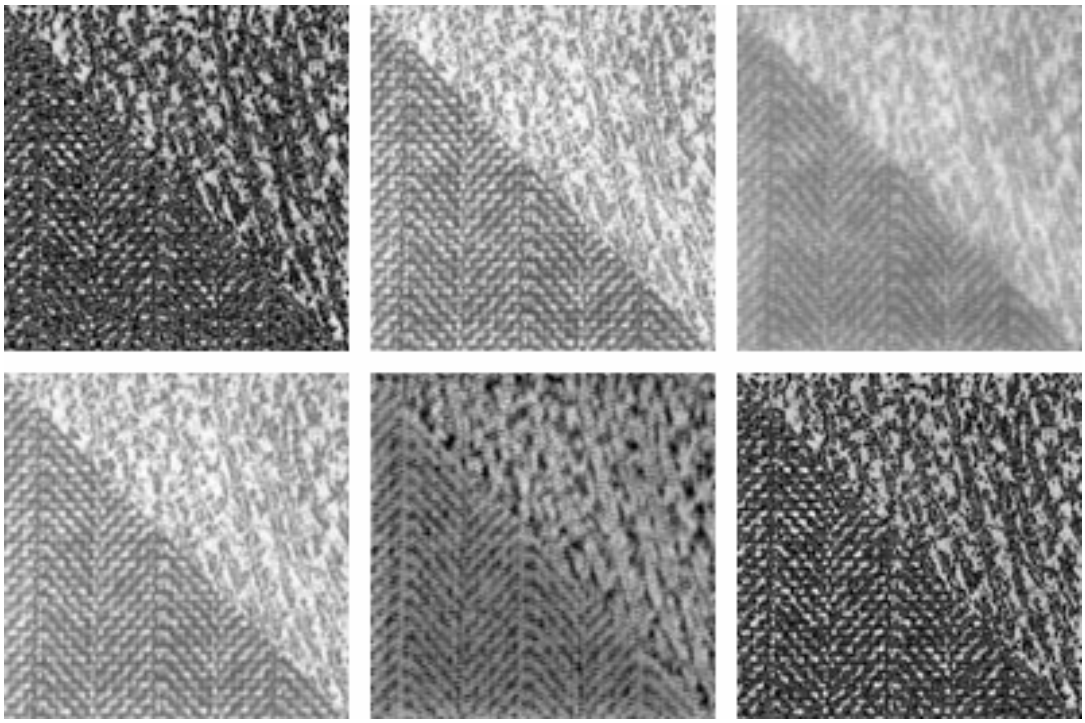


Figura 4-10: Componentes Principales - **Arriba de izquierda a derecha:** λ = 10269,299 81,113 %; λ = 1743,960 13,774 %; λ = 336,062 2,654 %. **Abajo de izquierda a derecha:** λ = 121,884 0,962 %; λ = 91,961 0,726 %; λ = 46,548 0,367 %

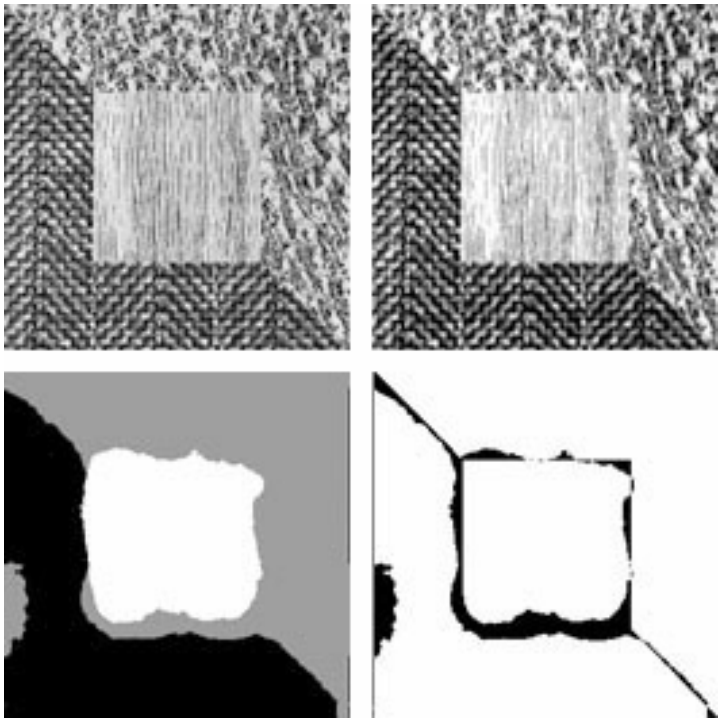


Figura 4-11: **Arriba**. Izquierda: Imagen Original. Derecha: Resultado SOFM. **Abajo**. Izquierda: Segmentación. Derecha: Error de Segmentación.

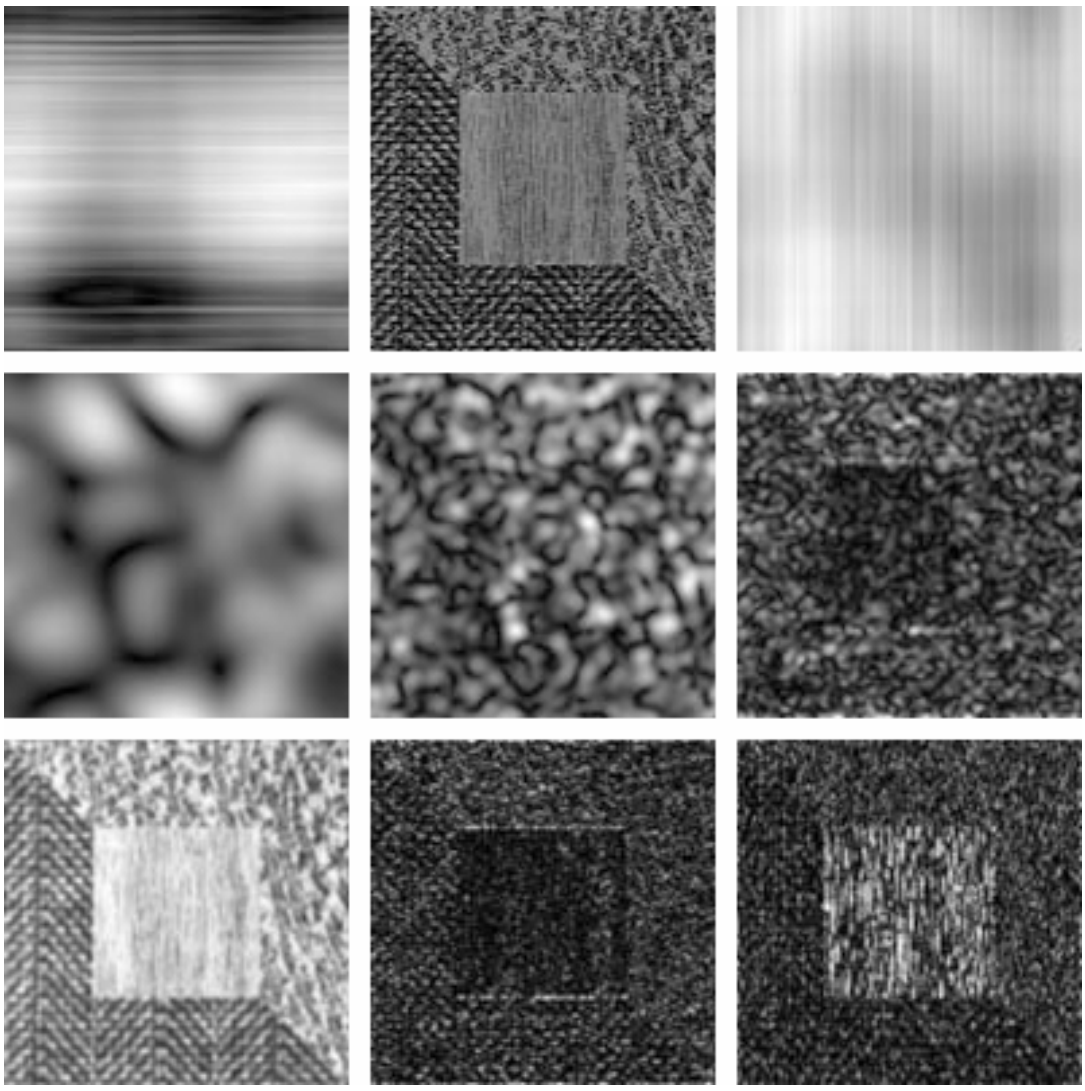


Figura 4-12: Algunas respuestas a los filtros

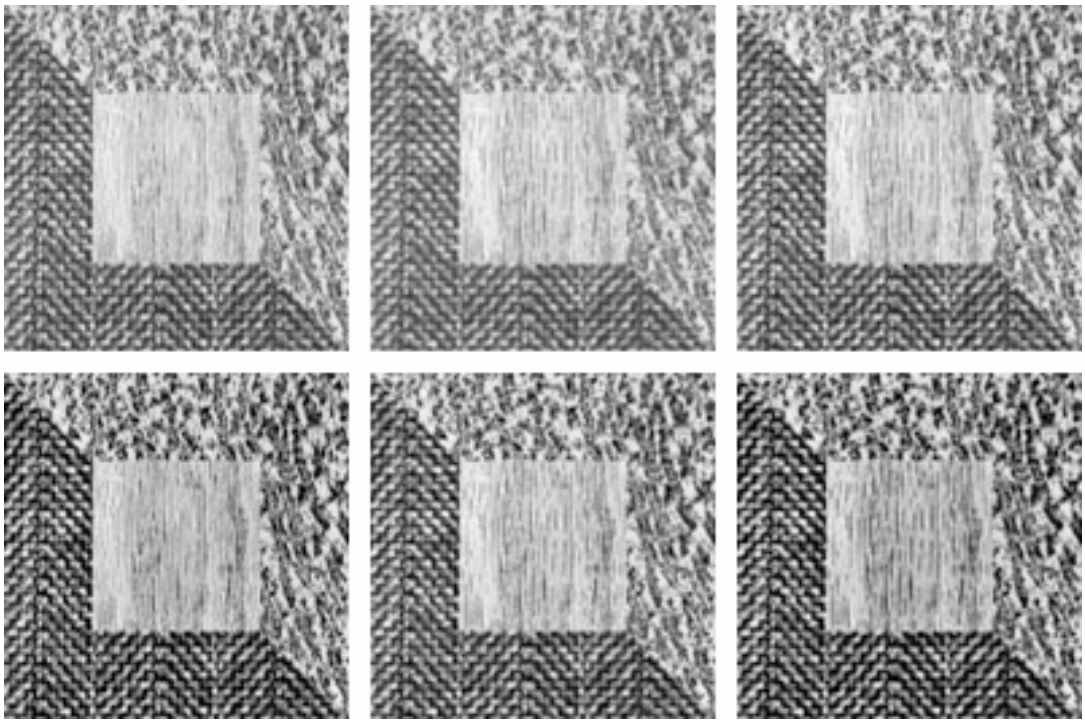


Figura 4-13: Componentes Principales - **Arriba de izquierda a derecha:** λ = 14877,379 82,871 %; λ = 2219,253 12,361 %; λ = 411,797 2,293 %. **Abajo de izquierda a derecha:** λ = 195,938 1,091 %; λ = 114,428 0,637 %; λ = 61,091 0,340 %

Capítulo 5

Conclusiones

La utilización de campos aleatorios de Markov en el modelado de texturas nos permitió generar imágenes texturadas que luego pudieron ser segmentadas con un error aceptable. Sin embargo, el modelo utilizado nos brindó un limitado "poder expresivo" en cuanto a las texturas generables. El uso sólamente de los parámetros β_i (cliques de dos sitios) nos permitió que los algoritmos se ejecuten en tiempos cortos. Igualmente las implementaciones que realizamos tanto para la síntesis de texturas, la estimación de parámetros, como segmentación de imágenes texturadas podrían extenderse sin gran dificultad a modelos más complejos que incluyan mayor cantidad de parámetros (es decir, cliques de mayor tamaño), obteniendo así un mayor poder expresivo.

Los algoritmos derivados del método propuesto por Weldon, los cuales se basan en la premisa de contar con un banco de filtros de Gabor, nos permitieron obtener resultados con errores de segmentación muy bajos.

Las precondiciones del método son el conocimiento *a priori* de las texturas intervinientes en la imagen. En la práctica esto podría lograrse básicamente de dos formas:

1. Solicitando de forma interactiva al usuario de la aplicación que *marque* en la imagen texturada (utilizando el mouse) regiones representantes de cada una de las texturas.

2. Obteniendo antes de comenzar a procesar las imágenes texturadas, muestras de cada una de las texturas que se desean diferenciar (segmentar). Un ejemplo de esto es el proce-

samiento de imágenes aereas, donde uno podría definir de antemano las *texturas* que le interesa detectar. Por ejemplo considerar sólo cuatro posibles etiquetas - texturas - : *campo, bosque, poblado con poca densidad edilicia y poblado con alta densidad edilicia*. Una vez obtenido el banco de filtros candidatos, se pueden procesar de forma automática una gran cantidad de imágenes texturadas buscando identificar estos patrones.

El algoritmo también requiere de antemano especificar ciertos parámetros que definirán cuales serán los filtros incluidos en el banco de candidatos. Sobre los conjuntos Σ y Λ no se menciona en ninguna de las referencias alguna restricción en cuanto a su conformación, más allá de *informar* los conjuntos utilizados en los ejemplos presentados. Consideramos que puede ser de interés profundizar en el estudio de la relación entre los conjuntos de parámetros elegidos y las texturas que se desean segmentar.

Los valores de error *empíricos* basados en la utilización de imágenes texturadas sintetizadas resultaron una medida muy cercana a la realidad, ya que no sólo toman en cuenta para el cálculo del error de segmentación estimado a los filtros seleccionados, sino que también incluyen al clasificador vectorial utilizado al segmentar.

El mayor tiempo de cómputo del algoritmo se encuentra en la primera parte: el procesamiento del banco de filtros candidatos. Una vez obtenidos los valores de μ y σ para cada una de las combinaciones filtro - textura para un banco de filtros considerablemente grande (más de 10000 filtros), el algoritmo realiza la selección de los mejores filtros y la segmentación de forma rápida.

La implementación del algoritmo en forma distribuida nos permitió acotar fuertemente el tiempo de procesamiento del banco de filtros candidatos; y la decisión de implementar computación distribuida en lugar de MPP hizo que podamos ejecutar el algoritmo en más de 10 procesadores a la vez, utilizando computadoras *hogareñas* simplemente conectadas entre sí a través de Internet.

El método de segmentación sin supervisión podría verse como el más ambicioso de los métodos propuestos hasta ahora. Al mismo tiempo es una combinación de ellos. Utiliza un

pequeño banco de filtros de Gabor que son generados en base a parámetros de frecuencia y dispersión predeterminados. A diferencia del método de Weldon, se aplica la totalidad de los filtros y se usan las respuestas como entrada del proceso de clasificación. En el proceso intervienen el Algoritmo Kohonen (SOFM), técnicas de histogramas y K-Means. Finalmente se hace un ajuste de la segmentación estimando los parámetros GMRF de cada textura y ajustando mediante ICM.

La única información a priori requerida es la cantidad de texturas. Considerando esto, los resultados son más que aceptables, aunque no logran la calidad de segmentación de la implementación basada en el algoritmo de Weldon. Por otro lado, los resultados no son comparables con los de MRF, ya que las texturas que pueden tratarse en cada caso son muy diferentes.

Finalmente, se deben analizar y determinar las características de cada problema de segmentación antes de seleccionar el método a implementar. Dependiendo de las variables de tiempo de cómputo, velocidad, capacidad de paralelizar, información a priori, necesidad de exactitud en la segmentación; se podrá optar por un método u otro (o una combinación). Lo que pudimos corroborar en este trabajo es que teniendo en cuenta estos parámetros, todos los métodos analizados logran resultados más que aceptables.

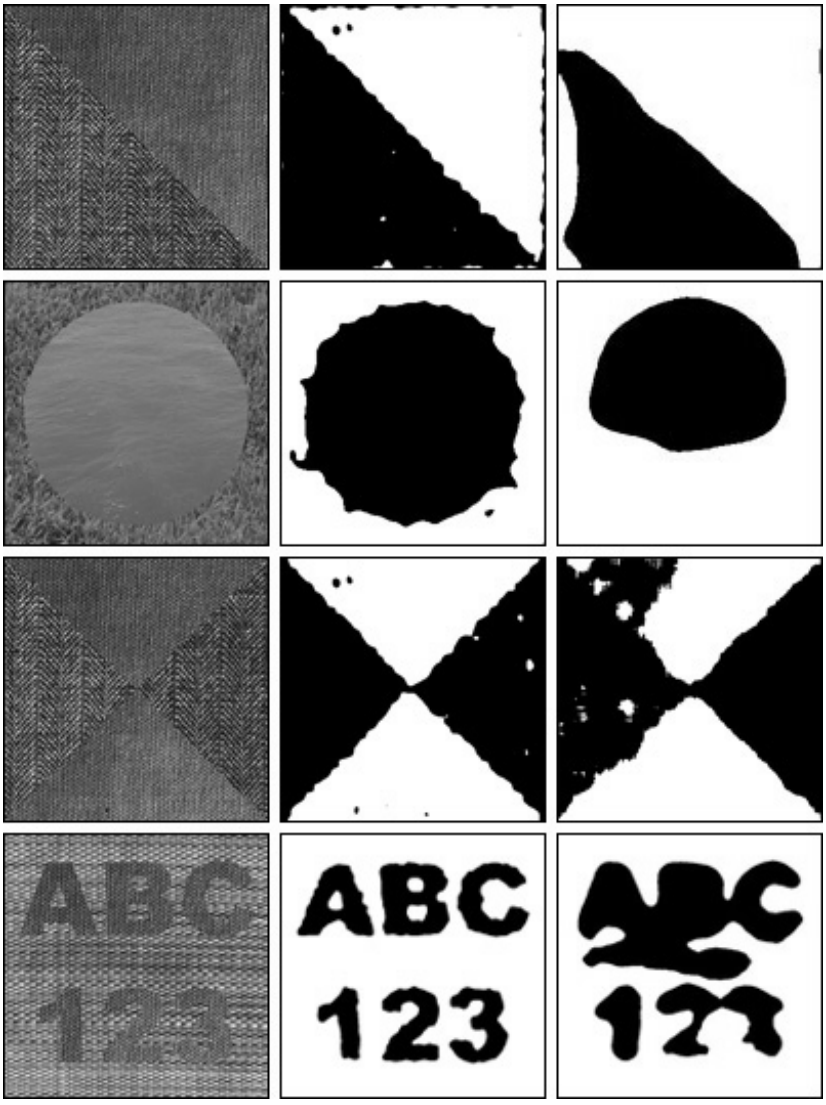


Figura 5-1: **Comparación de resultados de segmentación en imagenes conformadas por dos texturas:** (columna izquierda) Imagen texturada de entrada; (columna central) Segmentación obtenida con el algoritmo supervisado, basado en el método presentado por Weldon; (columna derecha) Segmentación obtenida con el algoritmo sin supervisión.

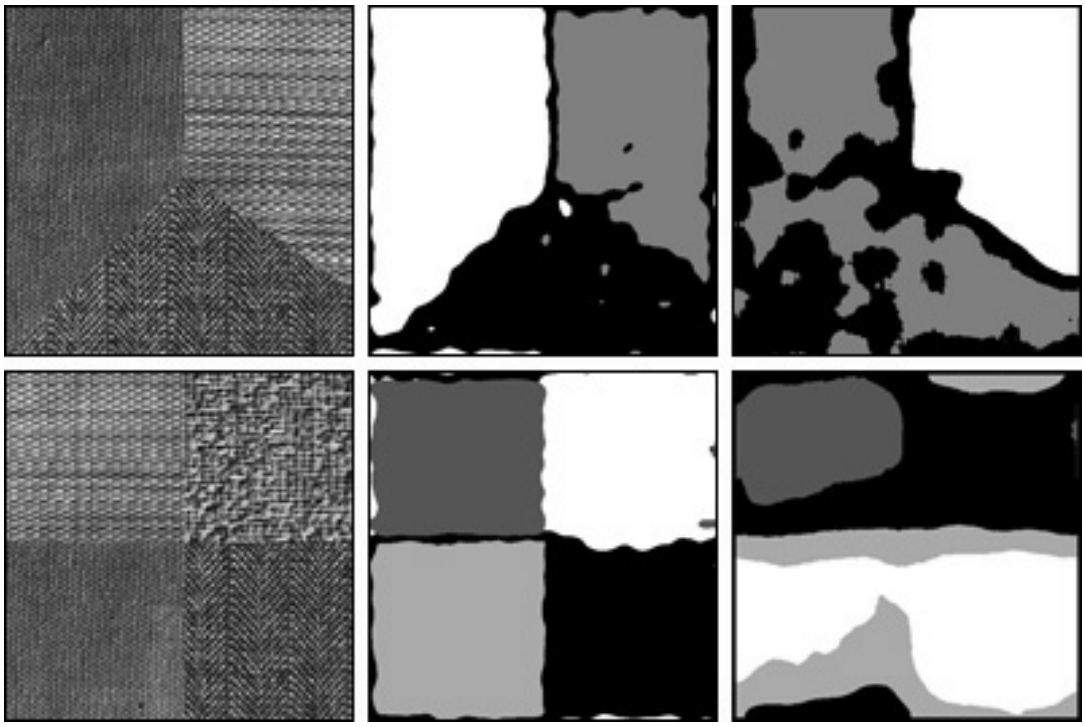


Figura 5-2: **Comparación de resultados de segmentación en imagenes conformadas por tres y cuatro texturas:** (columna izquierda) Imagen texturada de entrada; (columna central) Segmentación obtenida con el algoritmo supervisado, basado en el método presentado por Weldon; (columna derecha) Segmentación obtenida con el algoritmo sin supervisión.

Apéndice A

Programas

La implementación de los distintos programas de la tesis se realizaron utilizando Borland Delphi 6.

A continuación explicamos brevemente el alcance de cada uno de los programas desarrollados.

A.1. Generación de Texturas MRF

En este primer programa implementamos los algoritmos de muestreo Metrópolis y Gibbs para generar texturas utilizando campos aleatorios de Markov. La aplicación permite ingresar los parámetros β_i, seleccionar el algoritmo de muestreo e indicar la cantidad de iteraciones deseadas. Luego nos muestra la textura sintetizada.

Esta aplicación nos permitió comprender los campos de Markov y ver como afectan los valores de los parámetros a la textura generada.

También pudimos ver que frente a un mismo conjunto de parámetros, las realizaciones diferían visualmente, ya que obviamente el algoritmo de muestreo posee un gran factor aleatorio. Por este motivo, le incluimos a la aplicación la funcionalidad de guardar las imágenes, para su posterior utilización.

A.2. Segmentación de Texturas MRF

Esta aplicación implementa el algoritmo supervisado de segmentación utilizando campos aleatorios de Markov. En primer lugar se especifican los parámetros β_i deseados para cada una de las texturas que conformarán la imagen texturada. La aplicación sintetiza las texturas, crea una imagen con todas las texturas y realiza la segmentación. En el proceso de segmentación utiliza los parámetros β_i de cada una de las texturas.

A.3. Estimación de Parámetros con MRF

Esta aplicación permite interactivamente diseñar distintas texturas, por medio del ingreso de los parámetros β_i. Una vez ingresados los cuatro valores, la aplicación sintetiza la textura, utiliza el algoritmo de estimación de parámetros para estimarlos en base a la textura generada, y a su vez sintetiza una nueva textura pero con los parámetros estimados. Esto le permite al usuario no sólo ver los valores de los parámetros estimados, sino cuanto difieren visualmente la imágen creada con los parámetros reales, de la creada con los calculados.

Luego de finalizar el armado de las texturas, se ejecuta el algoritmo de segmentación sobre una imagen texturada creada con las texturas reales, pero utilizando los parámetros estimados.

La aplicación permite luego guardar las texturas sintetizadas con parámetros reales y estimados, la imagen texturada y el resultado de la segmentación.

A.4. Algoritmo de Weldon

Esta aplicación resume todas las funciones que fuimos implementando a medida que abordamos los temas asociados con la implementación del algoritmo de Weldon y los filtros de Gabor.

Las principales funciones con que cuenta son:

- Generación de banco de filtros candidatos, incluyendo las funciones para poder tanto cargarlo, como guardarlo a disco.

- Armado del banco de texturas, permitiendo cargar imágenes en formato .raw.
- Aplicación mediante FFT de filtros de Gabor a las imagenes de entrada.
- Posibilidad de cargar de disco las imágenes texturadas, o bien crear dinámicamente en base al banco de texturas existente.
- Funcionalidad de procesamiento distribuido del banco de filtros, tanto las funciones del nodo central, como la de los nodos remotos. Esto incluye la utilización de comunicación TCP-IP, como así también del procesamiento de documentos XML.
- Algoritmo de selección del banco de filtros, a partir del banco de filtros candidatos procesado.
- Algoritmos de segmentación de imágenes, utilizando como clasificadores finales el método de distancia euclídea, el método de distancia de Mahalanobis, como también la implementación del algoritmo de K-Means.
- Cálculo del error de segmentación real, como así también los errores de segmentación mediante mapas sintetizados.
- Implementación de un método de postprocesamiento sobre los pixels a los que el algoritmo (en base a los valores ingresados por el usuario) les asignó un valor de confianza alto.
- Posiblidad de guardar a disco toda la información sobre la segmentación, incluyendo las imágenes resultantes, intermedias, como así también un reporte del banco de filtros utilizado y los errores obtenidos.

A.5. Kohonen

Implementación del algoritmo de Kohonen de SOFM. El mismo nos permitió entender el funcionamiento de las redes de Kohonen.

La aplicación permite seleccionar los parámetros que determinan el comportamiento de la primera y segunda fase. También permite seleccionar si que quiere trabajar con una red en

una o dos dimensiones. El resultado de la ejecución es un gráfico en el que se ven los puntos generados al azar y la adaptación de la red para cubrir todo el espacio de datos.

A.6. Método sin supervisión

Esta aplicación permite segmentar una imagen texturada utilizando un método sin supervisión.

Se permite la carga tanto del archivo de la imagen texturada como del mapa de texturas real.

Luego de ejecutar el proceso de segmentación se puede visualizar la siguiente información:

- Imágen texturada.
- Mapa real.
- Resultado de la segmentación K-Means.
- Resultado de la segmentación mejorada.
- Diferencias entre los mapas incluyendo el porcentaje de efectividad.
- Respuestas de la aplicación de los filtros.
- Imágenes resultantes del proceso de Análisis de Componentes Principales.

  Además se cuenta con la posibilidad de guardar los archivos correspondientes a todas las imágenes mencionadas.

En esta aplicación se encuentran implementados los siguientes algoritmos:

- K-Means
- Componentes Principales
- SOFM (Kohonen)
- ICM

Apéndice A

Cálculo de estimadores de $\theta_r(c)$ y $\sigma_r^2(c)$

A.1. Estimador de $\theta_r(c)$

$$\frac{\partial \sum_{s\in S_c} \ln\left(f\left(x_s\right)\right)}{\partial \theta_r(c)} = 0 = \sum_{s\in S_c} -\frac{1}{2\sigma^2(c)} 2\left[x_s - \sum_{r\in N_s} \theta_r(c) x_{s+r}\right]\left[-\sum_{r\in N_s} x_{s+r}\right] =$$

$$0 = -\frac{1}{\sigma^2(c)} \sum_{s\in S_c} \left[\left[x_s - \sum_{r\in N_s} \theta_r(c) x_{s+r}\right]\left[-\sum_{r\in N_s} x_{s+r}\right]\right]$$

$$0 = -\frac{1}{\sigma^2(c)} \sum_{s\in S_c} \left[-x_s \sum_{r\in N_s} x_{s+r} + \sum_{r\in N_s} \theta_r(c) x_{s+r} \sum_{r\in N_s} x_{s+r}\right]$$

Asumiendo $\sigma^2(c) \neq 0$

$$0 = -\frac{1}{\sigma^2(c)} \sum_{s\in S_c} \left[-x_s \sum_{r\in N_s} x_{s+r} + \sum_{r\in N_s} \theta_r(c) x_{s+r} \sum_{r\in N_s} x_{s+r}\right]$$

$$0 = \sum_{s\in S_c} \left[-x_s \sum_{r\in N_s} x_{s+r}\right] + \sum_{s\in S_c} \left[\sum_{r\in N_s} \theta_r(c) x_{s+r} \sum_{r\in N_s} x_{s+r}\right]$$

$$-\sum_{s\in S_c}\left[-x_s\sum_{r\in N_s}x_{s+r}\right]=\sum_{s\in S_c}\left[\sum_{r\in N_s}\theta_r(c)x_{s+r}\sum_{r\in N_s}x_{s+r}\right]$$

$$\sum_{s\in S_c}\left[x_s\sum_{r\in N_s}x_{s+r}\right]=\sum_{s\in S_c}\left[\sum_{r\in N_s}\theta_r(c)x_{s+r}\sum_{r\in N_s}x_{s+r}\right]$$

Reescribiendo θ como vector columna y q_s como el vector de ocho vecinos de s

$$\sum_{s\in S_c}\left[x_s\sum_{r\in N_s}x_{s+r}\right]=\sum_{s\in S_c}\left[\theta^T(c)q_s\sum_{r\in N_s}x_{s+r}\right]$$

$$\sum_{s\in S_c}\left[x_s\sum_{r\in N_s}x_{s+r}\right]-\theta^T(c)\sum_{s\in S_c}\left[q_s\sum_{r\in N_s}x_{s+r}\right]=0$$

$$\sum_{s\in S_c}\left[x_sq_s^T\mathbf{1}\right]=\theta^T(c)\sum_{s\in S_c}\left[q_sq_s^T\mathbf{1}\right]$$

Llamando $v=\sum_{s\in S_c}\left[q_sq_s^T\mathbf{1}\right]$

$$\sum_{s\in S_c}\left[x_s\sum_{r\in N_s}x_{s+r}\right]v^T=\theta^T(c)vv^T$$

$$\sum_{s\in S_c}\left[x_s\sum_{r\in N_s}x_{s+r}\right]v^T\left[vv^T\right]^{-1}=\theta^T(c)vv^T\left[vv^T\right]^{-1}$$

Finalmente

$$\sum_{s\in S_c}\left[x_s\sum_{r\in N_s}x_{s+r}\right]v^T\left[vv^T\right]^{-1}=\theta^T(c)$$

A.2. Estimador de $\sigma^2(c)$

$$L=\sum_{s\in S_c}\left[-\frac{1}{2}\ln 2\pi-\ln\sigma(c)-\frac{1}{2\sigma^2(c)}\left[x_s-\sum_{r\in N_s}\theta_r(c)x_{s+r}\right]^2\right]$$

$$\frac{\partial}{\partial \sigma^2(c)} = 0 = \sum_{s \in S_c} \left[-\frac{1}{2} \frac{1}{\sigma^2(c)} - \frac{1}{2} \left[x_s - \sum_{r \in N_s} \theta_r(c) x_{s+r} \right] \right]$$

$$\frac{\partial}{\partial \sigma(c)} = 0 = \sum_{s \in S_c} \left[-\frac{1}{\sigma(c)} + \frac{2}{2\sigma^3(c)} \left[x_s - \sum_{r \in N_s} \theta_r(c) x_{s+r} \right]^2 \right]$$

$$0 = \sum_{s \in S_c} \left[-\frac{1}{\sigma(c)} + \frac{2}{2\sigma^3(c)} \left[x_s - \sum_{r \in N_s} \theta_r(c) x_{s+r} \right]^2 \right]$$

$$0 = \sum_{s \in S_c} \left[-\sigma^2(c) + \left[x_s - \sum_{r \in N_s} \theta_r(c) x_{s+r} \right]^2 \right]$$

$$0 = -\sigma^2(c) M + \sum_{s \in S_c} \left[x_s - \sum_{r \in N_s} \theta_r(c) x_{s+r} \right]^2$$

$$\sigma^2(c) = \frac{1}{M} \sum_{s \in S_c} \left[x_s - \sum_{r \in N_s} \theta_r(c) x_{s+r} \right]^2$$

donde $M = \#\left(S_c\right)$

Bibliografía

[1] Chow, C. K., *A recognition method using neighbor dependence.* IRE Transactions on Electronic Computer. 1962

[2] Abend, K. et al, *Classification of binary random patterns.* IEEE Transactions on Information Theory. 1965

[3] Besag, J., *Spatial interaction and the statistical analysis of lattice systems (with discussions).* Journal of the Royal Statistical Society. 1974

[4] Hammersley, J.M. y Clifford, P. *Markov field on finite graphs and lattices.* No publicado. 1971

[5] S.Z. Li, *Markov Random Field Modeling in Computer Vision.* Springer.1995.

[6] J. G. Daugman, *Uncertainty relation for resolution in space, spatial frecuency, and orientation optimized by two dimensional visual cortical filters.* J. Opt. Soc. Amer. A, vol. 2, no. 7, pp. 1160-1169, Julio 1985

[7] T. Weldon, W. Higgins, *Algorithm for designing multiple Gabor filters for segmenting multitextured images.* 1997.

[8] T. Weldon, W. Higgins, D. Dunn, *Efficient Gabor filter design for texture segmentation.* 1996

[9] T. Weldon, W. Higgins, *Multiscale Rician approacho to Gabor filter design for texture segmentation.* 1996

[10] W. K. Pratt, *Digital Image Processing.* John Wiley and Sons, second ed., 1991.

[11] P. Brodatz, *Textures: A Photographic Album for Artists and Designers.* New York, NY: Dover, 1966

[12] Griffeath, D. *Introduction to random fields.* En Kemeny, Snell, Knapp, *Denumerable Markov chains* (2nd ed.) Springer-Verlag. 1976

[13] Derin, H. y Elliott, H. *Modeling and segmentation of noisy and textured images using Gibbs random fields.* IEEE Transactions on Pattern Analysis and Machine Intelligence. 1987

[14] Geman, D. Geman, S. Graffigne, C. Dong, P. *Boundary detection by constrained optimization.* IEEE Transactions on Pattern Analysis and Machine Intelligence. 1990

[15] Geman, D. Geman, S. *Stochastic relaxation, Gibbs distribution and the Bayesian restoration of images.* IEEE Transactions on Pattern Analysis and Machine Intelligence. 1984

[16] Pok, G. y Liu J..C. *Unsupervised texture segmentation based on histogram of encoded Gabor features and MRF model.* 1999

[17] Gonzalez, R.C. y Woods, R.E. *Tratamiento digital de imágenes* Addison-Wesley/Diaz De Santos. 1996