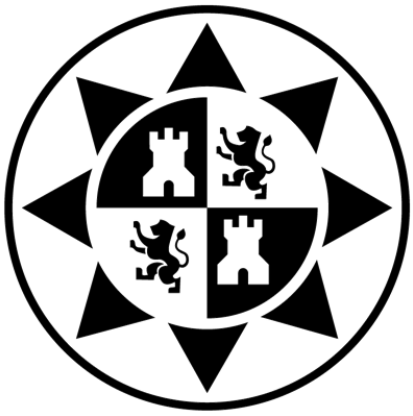
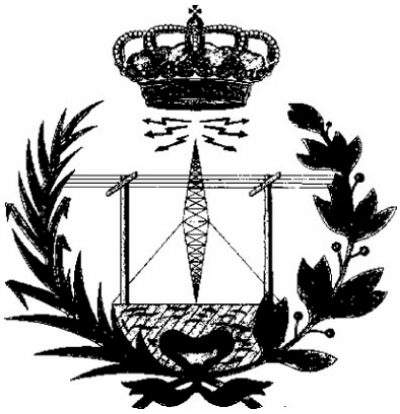
ESCUELA TÉCNICA Y SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA



PROYECTO FIN DE CARRERA

Programación de una interfaz gráfica con diferentes módulos para el tratamiento digital de imágenes en entornos industriales



Autor: Juan Andrés Campillo Fuentes
Director: Juan Carlos Trillo Moya

Cartagena, Julio de 2012

Autor	Juan Andrés Campillo Fuentes
E-mail del Autor	juanandres_upct@hotmail.com
Director	Juan Carlos Trillo Moya
E-mail del Director	jctrillo@upct.es
Título del PFC	Programación de una interfaz gráfica con diferentes módulos para el tratamiento digital de imágenes en entornos industriales
Descritores	Operaciones y filtros mofológicos, descriptores de regiones, segmentación de imágenes digitales basadas en umbralización(Otsu), en detección de bordes (Roberts, Sobel, Prewitt) y en regiones.

Resúmen

En la actualidad, mediante el procesamiento digital de imágenes se disponen de las herramientas necesarias que permiten poder implementar un amplio espectro de utilidades en diversos campos de aplicación como la robótica, medicina, telecomunicaciones, etc.

El estudio de los posibles tipos de relaciones existentes entre los píxeles de una imagen, las distintas operaciones morfológicas (dilatación, erosión, apertura, cierre, coincidencia estructural), filtros morfológicos y elementos estructurantes que se pueden aplicar a una imagen digital, así como el estudio de los diversos tipos de segmentación (basada en umbralización, en la detección de bordes y en las regiones) que posibilitan la extracción o aislamiento de objetos considerados de interés o de importancia de acuerdo al problema planteado, permiten el desarrollo de un amplio conjunto de aplicaciones concretas que logran satisfacer ciertas necesidades que son de utilidad en el ámbito de los entornos industriales.

Para ello se implementa a su vez, una interfaz gráfica intuitiva por medio de la que el usuario pueda tener acceso a todos los desarrollos implementados, así como un grupo de imágenes elaboradas para comprobar las funcionalidades desarrolladas.

Titulación	Ingeniero de Telecomunicación
Departamento	Matemática Aplicada y Estadística
Fecha de Presentación	Julio de 2012

Mi más sincero agradecimiento:

A los verdaderos artífices de esta obra, mis padres Pedro y Lali, por creer siempre en mí, animarme día tras día, y enseñarme que la base de cualquier proyecto exitoso se cimenta en el esfuerzo y las ganas de superación.

A toda mi familia, acordándome especialmente de mis herman@s Carmen y Raúl, y de mi abuelo Juan Ros, que me inspira en la rúbrica de esta dedicatoria.

A mi pareja Julia, por su apoyo incondicional a la hora de recorrer este camino, asi como todas aquellas personas que me han acompañado y contribuido, de una forma u otra en esta aventura universitaria, como Juan Carlos con su ayuda y predisposición ilimitada.

¡Muchas gracias a tod@s!

Índice de Contenidos

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1. Antecedentes

Mediante este capítulo, se permite al lector introducirse en el desarrollo de las técnicas matemáticas y las tecnologías de la información, que posibilitan el tratamiento de imágenes mediante el uso de sistemas digitales. Esta perspectiva histórica provee de la información necesaria para comprender el estado actual, en cuanto a la elevada capacidad de las técnicas de **Procesamiento Digital de Imágenes** [1].

El procesamiento digital de imágenes engloba al conjunto de técnicas que se aplican a las imágenes digitales con el objetivo de mejorar la calidad o facilitar la búsqueda de información en las mismas. El origen de este tratamiento de imágenes puede ser datado hacia principios de la década de los sesenta, ya que fue en este tiempo cuando la NASA daba seguimiento al programa de ciencia lunar, con el que se intentaba caracterizar la superficie de la luna para apoyar el posterior programa Apollo. En este proyecto, se tomaron grabaciones de la superficie lunar, y tras varios intentos fallidos de enviarlas a la Tierra, se lograron enviar dichas grabaciones a la Tierra, para que una vez recepcionadas de forma correcta fueran convertidas de su forma analógica a digital.

Una vez realizadas las tareas comentadas y dada la baja calidad de las imágenes recibidas, para poder evaluar dichas capturas se hizo necesario realizar ciertas operaciones en el tratamiento de las mismas, como por ejemplo la eliminación de distorsiones geométricas y de respuesta. Es por ello, que se estima conveniente vincular el inicio del tratamiento digital de imágenes con el desarrollo del citado proyecto.

Con el programa espacial comentado se comenzaron a utilizar las computadoras para el procesamiento de imágenes, y aunque inicialmente este procedimiento se llevó a cabo únicamente en aplicaciones astronómicas, o en campos de investigación relacionados con las matemáticas y las ciencias de la computación, factores como la disminución en el precio así como el aumento de la accesibilidad a sistemas digitales para la obtención y manejo de imágenes, sumada a la revolución de las microcomputadoras, han logrado convertir el procesamiento digital de imágenes en parte de la vida cotidiana, llegando a ser una herramienta

indispensable en diversos campos de aplicación, como por ejemplo en la robótica, la medicina, las telecomunicaciones y el control industrial entre otros, tal y como se hace en el presente proyecto final de carrera.

1.2. Conceptos Básicos

1.2.1. Imágenes digitales. Tipos

La imagen digital, bien sea generada por el ordenador o bien creada a través de algún instrumento de captura, tal como una cámara o un escáner, supone la traducción de los valores de luminosidad y color a lenguaje digital. La principal ventaja aportada por este lenguaje es la estabilidad: mientras que la emulsión de una imagen fotográfica clásica sufren una degradación química con el paso del tiempo, que repercute en la calidad de dicha reproducción, los ceros y unos que componen una imagen digital permanecen estables, con lo que la imagen no variará a lo largo del tiempo.

La clasificación de las imágenes digitales [7], es una tarea que puede realizarse basándose en múltiples criterios, mientras que en el caso que nos ocupa, nos interesa exclusivamente la forma en que esta imagen se encuentra descrita en el ordenador. En base a esta premisa, podemos distinguir dos grandes grupos de imágenes digitales:

- Las **imágenes vectoriales**, en las que la información de cada uno de los puntos se recoge en forma de ecuación matemática que lo relaciona con el resto de los puntos que forman la imagen. Ofrece la gran ventaja de que la calidad de la imagen no varía al modificar el tamaño, ya que la información de cada punto no es absoluta sino relativa al resto de la imagen. Además, debido a su definición matemática apenas ocupa espacio, ya que una fórmula que represente su forma es suficiente para representar todos los puntos que la componen.

- Las **imágenes de mapa de bits o bitmap**, se construyen describiendo cada uno de los puntos que componen la imagen y llevan, por tanto, información acerca de la posición absoluta y el color de cada uno de ellos. Se podría decir que las imágenes de mapa de bits están construidas mediante una gran cantidad de cuadraditos, llamados *píxel*, y que cada uno de estos cuadraditos está relleno de un color uniforme. La ventaja que presenta este formato es la posibilidad de recoger una amplísima gama tonal, por lo que es el tipo adecuado para representar imágenes captadas de la realidad. Dado que esta tecnología de imagen digital es la más usada (cámaras y escáneres digitales funcionan de esta manera), se hace referencia a la misma en el resto del trabajo.

  Las imágenes bitmap, a diferencia de las imágenes vectoriales, no pueden escalarse sin consecuencias que alteren su aspecto. A la hora de escalar, tanto a mayor como a menor escala, hay que tener en cuenta muchos factores, entre los cuales se encuen-

tran la resolución, el modo de color, la profundidad de bits y el formato de compresión.

1.2.2. Resolución de una imagen digital

La resolución de una imagen digital de mapa de bits es la característica que le permite tener mayor o menor nitidez o calidad visual, y apreciar mayor o menor detalle en la imagen [7]. Las dimensiones del área que ocupa una imagen digital (alto y ancho, ya que siempre son cuadrangulares) se miden en píxeles. Las dimensiones de una imagen están íntimamente relacionadas con la resolución y el peso de la imagen.

1.2.3. Profundidad de color

Una forma muy importante de clasificar las imágenes de mapa de bits es según la cantidad y tipo de información que se asigne a cada píxel [7]. La **Profundidad de Bits** viene determinada por la cantidad de bits utilizados para definir cada píxel. Mientras mayor sea la profundidad de bits, mayor será la cantidad de tonos (escala de grises o color) que se pueden representar. Las imágenes digitales se pueden producir en blanco y negro (en forma *monocromo*), a *escala de grises* o a *color RGB*.

- Una **imamen monocromo (binarias)** está representada por píxeles que constan de 1 bit cada uno, que pueden representar dos tonos (típicamente negro y blanco), utilizando los valores 0 para el negro y 1 para el blanco o viceversa.

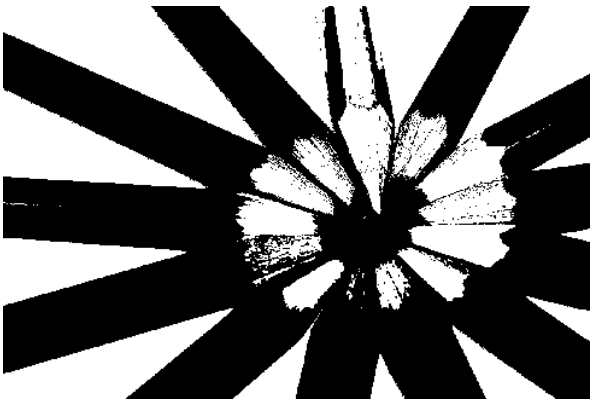


Figura 1.1: Imagen monocromática, 1 bit por píxel.

- Una **imagen en escala de grises** está compuesta por píxeles representados por múltiples bits de información, que típicamente varían entre 2 a 8 bits (un byte), por lo que cada píxel puede tener 256 valores diferentes (tonos de gris distintos).
- Una **imagen a color RGB** está típicamente representada por una profundidad de bits entre 8 y 24. En una imagen de 24 bits, los bits están divididos en tres grupos: 8 para el rojo, 8 para el verde, y 8 para el azul. Para representar otros colores se utilizan

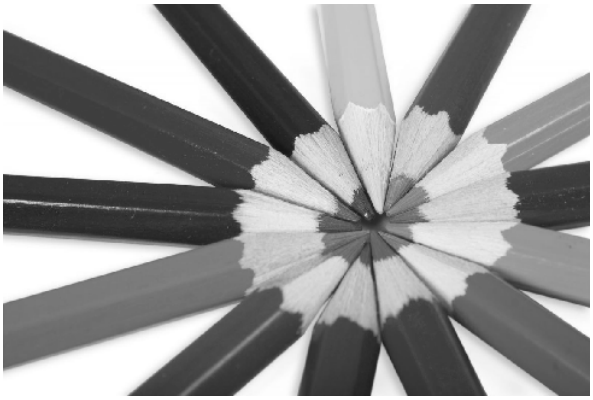


Figura 1.2: Imagen en escala de grises, 8 bits por píxel

combinaciones de esos bits. Una imagen de 24 bits ofrece 16,7 millones (2^{24}) tonos de color (256 Rojo x 256 Verde x 256 Azul).

Figura 1.3: Imagen a color RGB, 24 bits por píxel

1.3. Motivación y objetivos

En la exposición de este capítulo introductorio, a modo general se han citado algunos campos de aplicación (robótica, medicina, telecomunicaciones, etc) del procesamiento digital de imágenes, pero va a ser en el ámbito industrial en el que se va concretar el desarrollo del presente trabajo.

A partir de la premisa anterior, se puede advertir que el objeto del proyecto fin de carrera que nos atañe, consiste en la implementación de herramientas software específicas para su uso en entornos industriales, que permitan facilitar la realización de diversos procesos de control de calidad, producción y diseño, haciendo uso para ello de las técnicas de procesado digital de imágenes y las posibilidades de análisis del contenido de las mismas.

En cada uno de los desarrollos llevados a cabo, el mayor reto a satisfacer consistió en dotar de aplicaciones concretas que lograran adaptar la tecnología de las imágenes a un problema específico, por lo que previo al desarrollo software de cada aplicativo fue necesario realizar el siguiente proceso:

- Reconocer el problema y determinar si se puede solucionar con procesamiento de imágenes.
- Comprender las capacidades y limitaciones del procesamiento de imágenes.
- Utilizar la información anterior en técnicas fundamentales de algoritmos matemáticos y su correspondiente implementación.

Para el desarrollo de los aplicativos software se opta por utilizar *Matlab*® [3], al ser un lenguaje de alto nivel que dispone de un entorno interactivo que permite realizar ciertas tareas de cálculo complejas de forma más rápida en comparación con otros lenguajes de programación tradicionales (C, C++, etc). *Matlab*® está muy extendido en el área de la formulación matemática y dispone de una herramienta de procesamiento de imágenes que ayuda en la tarea de construir una aplicación específica, favoreciendo la manipulación y modificación de imágenes digitales.

Además de las bondades de *Matlab*® al ser un software óptimo para procesar matrices y, por tanto imágenes digitalizadas, también fue seleccionado por disponer de la herramienta *GUIDE* [2]. Dicha herramienta se utiliza para implementar la interfaz gráfica, mediante la que el usuario puede hacer uso, mediante un manejo sencillo, de todas las aplicaciones desarrolladas, sin necesidad de conocer en detalle la teoría relacionada con los algoritmos implementados.

1.4. Estructura

Por medio de esta sección, se puede obtener una breve visión de los principales contenidos documentados en los siguientes capítulos:

- En el *capítulo 2* se desarrollan los fundamentos teóricos en los que se basan las implementaciones de todos los algoritmos desarrollados mediante el presente proyecto final de carrera, es decir, en el citado capítulo se dan a conocer las técnicas que permiten analizar el contenido de imágenes digitales, la separación y extracción de información sobre los objetos que las componen, así como la realización de distintos tipos de operaciones y medidas sobre los objetos principales de la imagen, entre otras tareas.
- En el *capítulo 3* se documentan todos los aspectos relacionados con la interfaz gráfica de usuario principal de la aplicación, que van desde aquella información que debe conocer el usuario antes de iniciar dicha interfaz gráfica, hasta un completo y detallado

manual que permite al usuario de la misma, su correcta utilización sin necesidad de poseer conocimientos avanzados de *Matlab*®.

- En el *capítulo 4* se informa al lector de las etapas de procesamiento que se han llevado a cabo para el desarrollo de todos los algoritmos implementados, así como la descripción pormenorizada de los detalles más relevantes en cuanto a las técnicas aplicadas para lograr la consecución de los objetivos marcados para cada funcionalidad desarrollada, así como el funcionamiento de las mismas y los resultados obtenidos.
- En el *capítulo 5* se presentan las conclusiones que se obtienen a partir de los trabajos realizados, así como posibles líneas futuras que se pueden seguir para lograr la ampliación y mejora de los sistemas diseñados.
- En el *apéndice A* se ilustran los códigos *Matlab* de las principales funciones implementadas, eludiendo exponer los códigos de aquellas funciones relacionadas con el desarrollo de las interfaces gráficas implementadas, dada su extraordinaria longitud en cuanto a líneas de código generadas.
- La memoria finaliza con la presentación de las *referencias bibliográficas* consultadas.

Capítulo 2

Procesamiento de imágenes digitales

2.1. Representación de imágenes digitales

Comenzaremos indicando que una imagen digital a niveles de gris es una función bidimensional de la intensidad de luz de:

$$f : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}, \tag{2.1}$$

cuyos valores se han obtenido muestreando la intensidad sobre una retícula rectangular. Por lo tanto, una imagen digital la vamos a expresar como $f(x, y)$, donde x e y son las coordenadas espaciales y el valor de f en cada punto (x, y) es proporcional a la intensidad de luz (nivel de gris) de ese punto.

Podemos decir que una imagen $f(x, y)$ está formada por dos componentes: una es la cantidad de luz incidente en la escena y la otra es la cantidad de luz reflejada por los objetos. Estas dos componentes se denominan: **iluminación**, que expresaremos como $i(x, y)$ y **reflectancia**, que denotaremos por $r(x, y)$. Entonces;

$$f(x, y) = i(x, y)r(x, y), \tag{2.2}$$

con $0 < i(x, y) < \infty$ y $0 < r(x, y) < 1$. La iluminación $i(x, y)$ está determinada por las características de la fuente que emite la luz y la reflectancia $r(x, y)$ por las características del objeto. Por ejemplo, tenemos valores normales como $i(x, y) = 100$ para una oficina comercial, $i(x, y) = 0{,}01$ para una noche clara, y $r(x, y) = 0{,}01$ para el terciopelo negro y $r(x, y) = 0{,}93$ para la nieve.

Llamaremos **nivel de gris**, l, a la intensidad de luz de una imagen f. El rango de variación de l será:

$$L_{min} \leqslant l \leqslant L_{max}. \tag{2.3}$$

En la práctica $L_{min} = i_{min}r_{min} \sim 0{,}005$ y $L_{max} = i_{max}r_{max} \sim 100$ en la mayoría de las aplicaciones de procesamiento de imágenes. Llamaremos al intervalo $[L_{min}, L_{max}]$ **escala de grises** y normalmente se desplaza al intervalo $[0, L]$, donde $l = 0$ se considera **negro** y

$l = L$ se considera **blanco**. El resto de valores son variaciones de grises que varían de forma continua desde el negro hasta el blanco.

En Matlab [15] una imagen a escala de grises es representada por medio de una matriz bidimensional de $m \times n$ elementos, donde n representa el numero de píxeles de ancho y m el numero de píxeles de largo. El elemento V_{11} corresponde al elemento de la esquina superior izquierda (ver figura 2.1), donde cada elemento de la matriz de la imagen tiene un valor de 0 (negro) a 255 (blanco).

$$f(x,y) = \begin{bmatrix} V_{11} & V_{12} & \cdots & V_{1n} \\ V_{21} & V_{22} & \cdots & V_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ V_{m1} & V_{m2} & \cdots & V_{mn} \end{bmatrix} \tag{2.4}$$

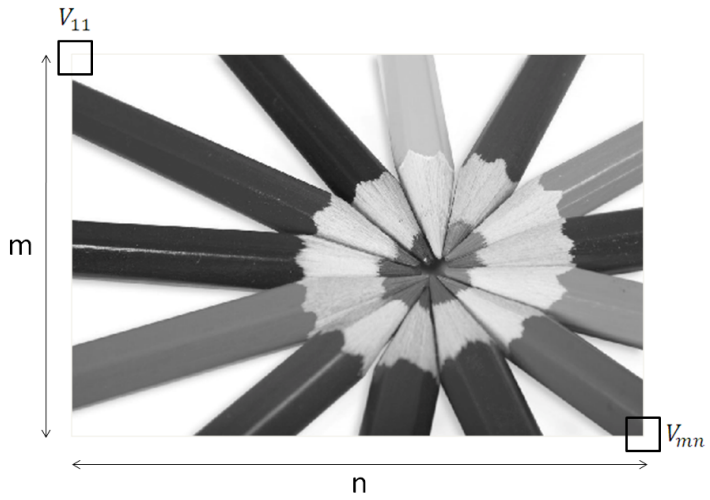


Figura 2.1: Representación de una imagen a escala de grises en Matlab

Por otro lado una imagen de color RGB es representada por una matriz tridimensional $m \times n \times p$, donde m y n tienen la misma significación que para el caso de las imágenes de escala de grises, mientras que p representa el plano, que para RGB puede ser 1 para el rojo, 2 para el verde y 3 para el azul. La figura 2.2 muestra los detalles de estos conceptos.

$$f_R(m,n,1) = \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ r_{21} & r_{22} & \cdots & r_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ r_{m1} & r_{m2} & \cdots & r_{mn} \end{bmatrix} \tag{2.5}$$

$$f_G(m,n,2) = \begin{bmatrix} g_{11} & g_{12} & \cdots & g_{1n} \\ g_{21} & g_{22} & \cdots & g_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ g_{m1} & g_{m2} & \cdots & g_{mn} \end{bmatrix} \tag{2.6}$$

$$f_B(m,n,3) = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{bmatrix} \tag{2.7}$$

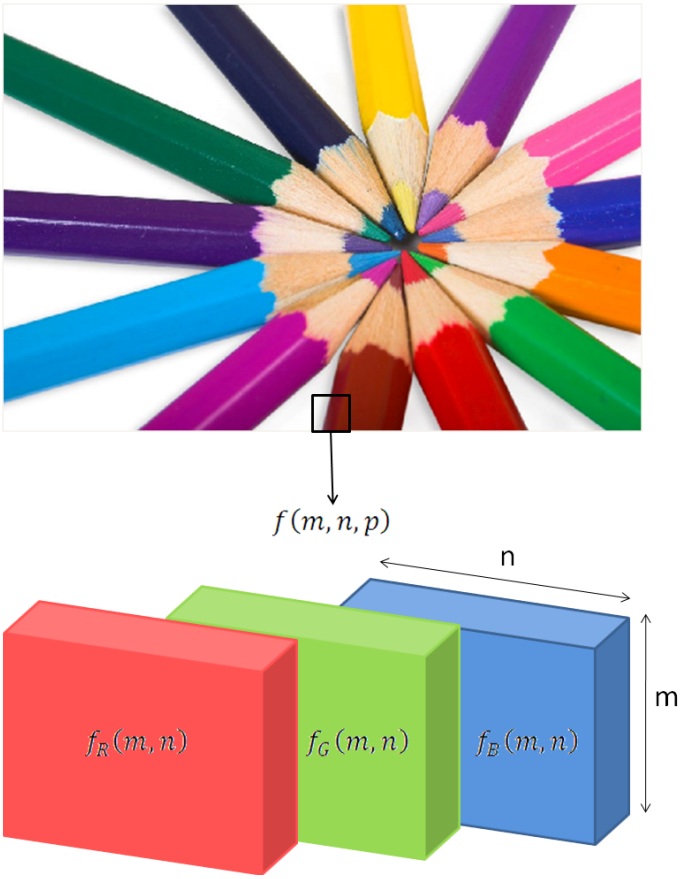


Figura 2.2: Representación de una imagen a color RGB en Matlab

2.2. Relaciones básicas entre píxeles

En esta sección se consideran algunas relaciones básicas pero de gran relevancia entre los píxels de una imagen digital. Como se menciona con anterioridad, una imagen digital la denotaremos por $f(x, y)$, y a partir de ahora, cuando se haga referencia a un píxel en particular, vamos a emplear las letras minúsculas p y q, mientras que un subconjunto de píxels de $f(x, y)$ se indica mediante S.

2.2.1. Vecindad

Un punto p de coordenadas (x, y) tiene 4 vecinos verticales y horizontales cuyas coordenadas son:

$$(x+1, y), (x-1, y), (x, y+1), (x, y-1). \tag{2.8}$$

A este conjunto de puntos se le llama los 4-vecinos de p y lo denotaremos como $N_4(p)$.

	$(x, y-1)$	
$(x-1, y)$	(x, y)	$(x+1, y)$
	$(x, y+1)$	

Tabla 2.1: Coordenadas de los 4-vecinos de p perpendiculares

Los 4 vecinos diagonales de p tienen como coordenadas:

$$(x+1, y+1), (x+1, y-1), (x-1, y+1), (x-1, y-1), \tag{2.9}$$

y los denotaremos como $N_D(p)$.

$(x-1, y-1)$		$(x+1, y-1)$
	(x, y)	
$(x-1, y+1)$		$(x+1, y+1)$

Tabla 2.2: Coordenadas de los 4-vecinos de p diagonales

A estos 4 vecinos diagonales $N_D(p)$ junto con $N_4(p)$ le llamaremos los 8-vecinos de p, denotado como $N_8(p)$. Sus coordenadas son:

$(x-1, y-1)$	$(x, y-1)$	$(x+1, y-1)$
$(x-1, y)$	(x, y)	$(x+1, y)$
$(x-1, y+1)$	$(x, y+1)$	$(x+1, y+1)$

Tabla 2.3: Coordenadas de los 8-vecinos de p

2.2.2. Conectividad

La conectividad entre píxels [10] es un concepto importante empleado para establecer los límites de los objetos y los componentes de áreas en una imagen. Para determinar si dos puntos (píxels) están conectados debemos determinar si son adyacentes en algún sentido (por ejemplo ser 4-vecinos) y si sus niveles de grises satisfacen algún criterio especificado de similitud (como ser iguales). Por ejemplo, en una imagen binaria con valores 0 y 1, dos

píxels pueden ser 4-vecinos pero no estarán conectados a menos que tengan el mismo valor.

Sea V el conjunto de valores de niveles de gris empleados para definir la conectividad; por ejemplo, en una imagen binaria, se tendrá $V = 1$ para la conectividad entre puntos (píxels) con valor 1. En una imagen con escala de grises, para la conectividad entre puntos con un rango de valores de intensidad de, por ejemplo, 1 a 100, se tiene $V = 1, 2, ..., 99, 100$. Se consideran dos tipos de conectividad:

- **4-Conectividad**: dos puntos p y q con valores de niveles de gris dentro de V están 4-conectados si q pertenece al conjunto $N_4(p)$.
- **8-Conectividad**: dos puntos p y q con valores de niveles de gris dentro de V están 8-conectados si q pertenece al conjunto $N_8(p)$.

0	0	0	0	0	0
0	13	43	0	34	22
0	12	0	54	98	67
0	21	45	0	86	0
0	0	0	0	76	0

Tabla 2.4: Ejemplo valores intensidad imagen para cálculo de conectividades

Usando 4-conectividad tenemos dos objetos; usando 8-conectividad tenemos sólo un objeto.

2.2.3. Etiquetado de componentes conectados

El etiquetado de componentes conectados [27], asigna una etiqueta única a un subconjunto de puntos llamados comúnmente objetos. La condición para pertenecer a cierta región es que el pixel en cuestión esté conectado con la región.

La definición de conectividad es clave para determinar que conjuntos van a ser considerados como conectados, ya que si se realiza un barrido de una imagen (de derecha a izquierda y de arriba a abajo), dependiendo del tipo de conectividad que se asuma, se puede obtener una descripción u otra de la imagen en términos de objetos, que permita simplificar en mayor o menor medida su análisis y aumentar la eficiencia en el procesamiento de la misma.

2.2.4. Medidas de distancia

Dados los puntos p, q, con coordenadas (x, y), (s, t), respectivamente:

La **distancia Euclídea** entre dos puntos p y q se define como:

$$D_e(p,q) = \sqrt{(x-s)^2 + (y-t)^2}. \tag{2.10}$$

Los puntos que están a una distancia D menor o igual a un determinado valor r de (x,y) son los puntos contenidos en un disco (círculo) de radio r y centrado en (x,y).

La **distancia** D_1 entre dos puntos p y q se define como:

$$D_1(p,q) = |x-s| + |y-t| \tag{2.11}$$

Los puntos que están a una distancia D_1 de (x,y) menor o igual a un determinado valor de r forma un diamante(rombo) centrado en (x,y). Por ejemplo, los puntos con una distancia $D_4 \leq 2$ desde (x,y) (el punto central) forman los siguientes contornos de distancia constante (el número que aparece en la celda es el valor de la distancia):

		2		
	2	1	2	
2	1	0	1	2
	2	1	2	
		2		

Tabla 2.5: Contornos de distancia constante para $D_1 \leq 2$

Los píxels con distancia $D_1 = 1$ son los 4-vecinos de (x,y).

La **distancia** D_∞ distancia entre dos puntos p y q se define como:

$$D_\infty(p,q) = max(|x-s|, |y-t|). \tag{2.12}$$

En este caso los puntos que están a una distancia D_∞ de (x,y) menor o igual que un cierto valor r forman un cuadrado centrado en (x,y). Por ejemplo, los puntos a una distancia $D_\infty \leq 2$ de (x,y) (el punto central) forman los siguientes contornos de distancia constante:

2	2	2	2	2
2	1	1	1	2
2	1	0	1	2
2	1	1	1	2
2	2	2	2	2

Tabla 2.6: Contornos de distancia constante para $D_\infty \leq 2$

Los píxels con distancia $D_\infty = 1$ son los 8-vecinos de (x,y).

2.3. Operaciones morfológicas

Clásicamente la morfología ha sido una parte de la biología que estudia la forma de los animales y de las plantas. De la misma forma, la *morfología matemática* consiste en un conjunto de técnicas matemáticas que permiten tratar problemas que involucran formas en una imagen digital.

La morfología matemática tiene su origen en la teoría de conjuntos. Para ella las imágenes binarias son conjuntos de puntos 2D, que representan los puntos activos de una imagen, y las imágenes en niveles de gris son conjuntos de puntos 3D, donde la tercera componente corresponde al nivel de intensidad. En este apartado se procede a tratar detalladamente la morfología sobre imágenes binarias [11], para luego presentar los operadores básicos sobre imágenes en niveles de gris.

2.3.1. Definiciones básicas

Sea A un conjunto (con las operaciones habituales entre conjuntos) de Z^2 (con las aplicaciones habituales entre vectores). Cualquier punto a de A se representa mediante un par (a_1, a_2). A continuación se definen las siguientes operaciones sobre A:

Traslación de A por $X = (x_1, x_2)$, como:

$$(A)_x = \{c/c = a + x, \forall a \in A\}. \tag{2.13}$$

Reflexión de A como:

$$\widehat{A} = \{x/x = -a, \forall a \in A\}. \tag{2.14}$$

Complementario de A como:

$$A^c = \{x/x \notin A\}. \tag{2.15}$$

También se define la operación *diferencia* entre dos conjuntos A y B como:

$$A - B = \{x/x \in A \quad y \quad x \notin B\}. \tag{2.16}$$

Una propiedad interesante que se deriva de las operaciones anteriores es:

$$A - B = A \cap B_c. \tag{2.17}$$

2.3.2. Elemento estructurante

Si bien los conjuntos A y B pueden ser considerados como una imagen (u objeto), generalmente se considera que A es la imagen y B es el *elemento estructural*. El elemento estructural es en morfología matemática lo que la máscara (o núcleo) de convolución es en los filtros lineales.

Los elementos estructurales más comunes son los conjuntos que están 4-conectados, N_4, y 8-conectados, N_8, ilustrados en las siguiente figura:

0	1	0
1	1	1
0	1	0

1	1	1
1	1	1
1	1	1

Figura 2.3: Elementos estructurantes estándar. a)N_4 b)N_8

2.3.3. Dilatación

Siendo A y B dos conjuntos en Z^2, la *dilatación* de A con B, denotada como $A \oplus B$, se define como:

$$A \oplus B = \{x/x = a + b \quad \forall a \in A \;\; y \;\; \forall b \in B\}. \tag{2.18}$$

El elemento B es el elemento que dilata al elemento A, y se conoce como *elemento estructurante* de la dilatación.

Es interesante notar que la dilatación cumple la propiedad conmutativa.

$$A \oplus B = B \oplus A. \tag{2.19}$$

La implementación directa de la dilatación según la definición dada es demasiado costosa. La siguiente formulación, que puede demostrarse que es equivalente, da pistas para una implementación mucho más eficiente.

$$A \oplus B = \{x/(\widehat{B})_x \cap A \neq \phi\}. \tag{2.20}$$

Escrito de otra forma:

$$A \oplus B = \{x/[(\widehat{B})_x \cap A] \subseteq A\}. \tag{2.21}$$

Intuitivamente esta operación produce el efecto de dilatar el aspecto del elemento A usando para ello a B.

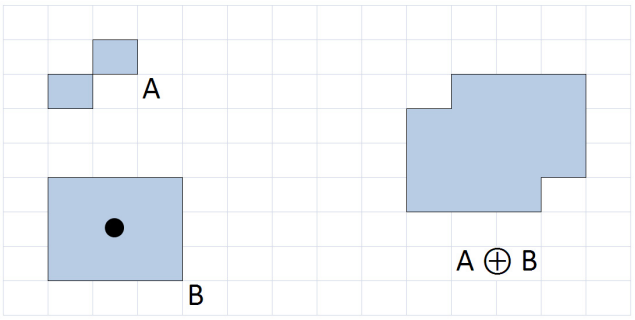


Figura 2.4: Ejemplo de dilatación donde se marca con punto negro el origen del elemento B

En la figura superior se presentan tres objetos conexos sobre un fondo blanco. El primer objeto se etiqueta como A y el segundo como B. El tercero corresponde a la dilatación de A con B. La posición del elemento B respecto del eje de ordenadas es importante, ya que influye en el proceso de dilatación, por ello suele indicarse su centro con un punto.

2.3.4. Erosión

Siendo A y B dos conjuntos en Z^2, la erosión de A con B, denotada como $A \ominus B$, se define:

$$A \ominus B = \{x/x + b \in A \quad \forall b \in B\}. \tag{2.22}$$

Nuevamente puede definirse con otra forma cuyo coste computacional es mucho más reducido.

$$A \ominus B = \{x/(B)_x \subseteq A\}. \tag{2.23}$$

La erosión adelgaza la imagen sobre la que se aplica siendo, en un sentido no estricto, opuesta a la dilatación. Si sobre la figura 2.4 se erosiona $A \oplus B$ con B se obtiene de nuevo A, aunque esto no tiene por qué ocurrir en otro caso distinto. En la figura presentada a continuación, se muestran los resultados obtenidos tras haber erosionado A con B.

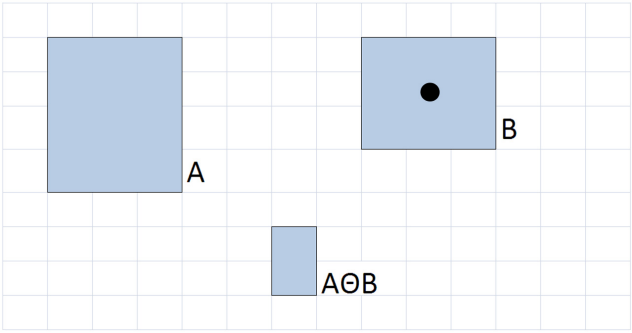


Figura 2.5: Ejemplo de erosión

Se cumple que la dilatación y la erosión son duales respecto al complemento y a la reflexión. Es decir:

$$(A \ominus B)^c = A^c \oplus \widehat{B}. \tag{2.24}$$

Esta propiedad se puede demostrar fácilmente con el siguiente desarrollo:

$$\begin{aligned}(A \ominus B)^c = \{x/(B)_x \subseteq A\}^c = \{x/(B)_x \cap A^c = \phi\}^c = \\ = \{x/(B)_x \cap A^c \neq \phi\} = A^c \oplus \widehat{B}.\end{aligned} \tag{2.25}$$

2.3.5. Apertura

La *apertura* de A con B se define como:

$$A \circ B = (A \ominus B) \oplus B. \tag{2.26}$$

Sus propiedades son:

- $A \circ B$ es un subconjunto de A.
- $(A \circ B) \circ B = A \circ B$.
- Si C es subconjunto de $D \Rightarrow C \circ B$ es subconjunto de $D \circ B$.

Intuitivamente la apertura de A con un elemento estructurante B equivale a determinar los puntos en los que puede situarse alguna parte de B cuando se desplaza por el interior de A (ver figura 2.6). La apertura abre, o agranda, las zonas de píxeles inactivos presentes en una zona de píxeles activos.

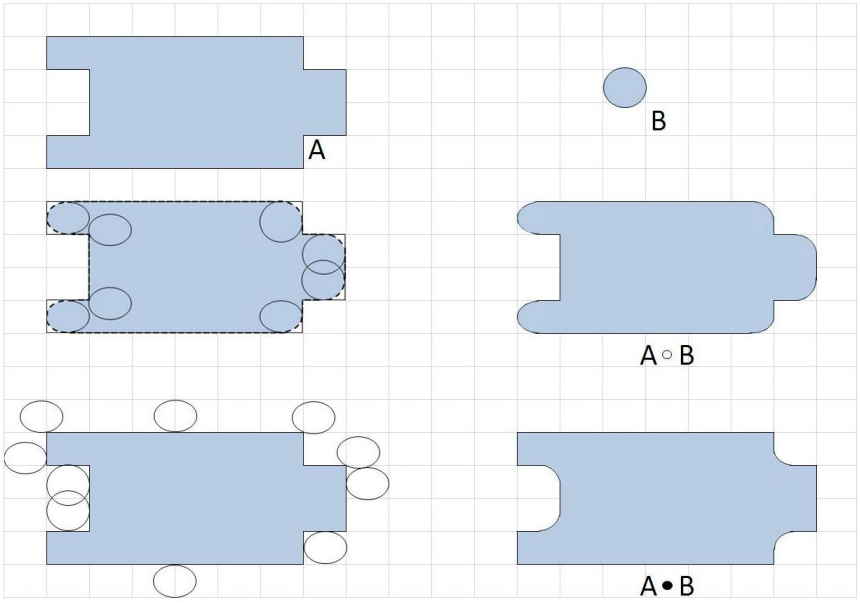


Figura 2.6: Arriba se presenta la figura A y el elemento estructurante B. En medio se presenta la ejecución de la operación de apertura y su resultado. Abajo se presenta la operación de cierre

2.3.6. Cierre

El *cierre* de A con B se define como:

$$A \bullet B = (A \oplus B) \ominus B. \tag{2.27}$$

Sus propiedades son:

- A es un subconjunto de $A \bullet B$.
- $(A \bullet B) \bullet B = A \bullet B$.
- Si C es subconjunto de $D \Rightarrow C \bullet B$ es subconjunto de $D \bullet B$.

Intuitivamente el cierre de A con un elemento estructurante B equivale a los puntos a los que no puede acceder ninguna parte de B cuando se desplaza por el exterior de A (ver figura 2.6). El cierre elimina zonas de píxeles inactivos presentes en el interior de una zona de píxeles activos.

2.3.7. Coincidencia estructural

Si el elemento B se define teniendo en cuenta los puntos a blanco que lo rodean, se tiene la descripción de un objeto B_1 y su entorno B_2. La operación de *coincidencia estructural*, o *Hit or Miss*, busca la parte de la imagen A que cumpla que los puntos activos de B estén en A y los puntos del entorno de B estén en A^c.

$$A * B = (A \ominus B_1) \cap (A^c \ominus B_2). \tag{2.28}$$

El elemento estructurante B suele definirse sobre una cuadrícula donde; un cuadro sombreado indica que el píxel pertenece a B_1, un cuadro blanco indica que el píxel pertenece a B_2, un cuadro con una cruz indica que el cuadro no debe tenerse en cuenta.

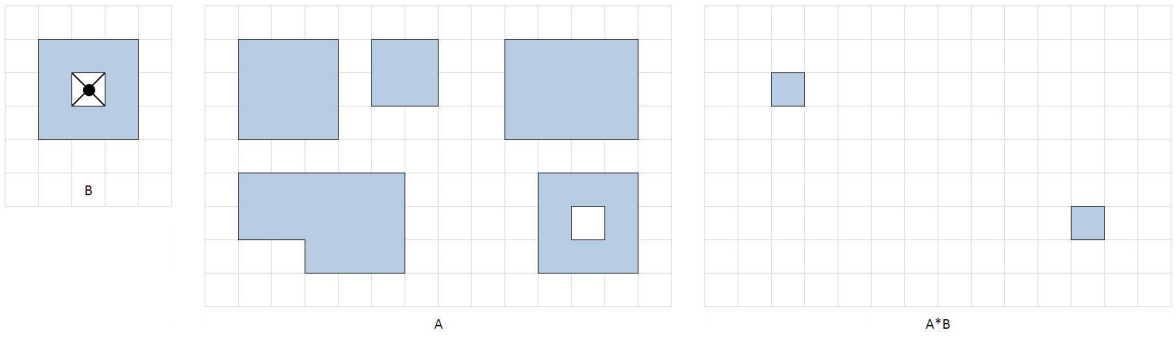


Figura 2.7: Ejemplo de operación de coincidencia estructural

2.3.8. Filtros morfológicos

Los siguientes puntos ilustran cómo emplear morfología para construir filtros.

Eliminación de ruido

Este filtro elimina los objetos de una imagen que tienen un tamaño menor que un elemento estructurante B determinado.

$$Limpiar\ (A) = (A\ {}^{\circ}B) \bullet B. \tag{2.29}$$

En el proceso de apertura elimina los objetos menores que B, luego intenta recuperar la misma forma que antes con el proceso de cierre (aunque los procesos de apertura y cierre no son estrictamente inversos).

Extracción de contornos

Este filtro obtiene los contornos de una figura A restándole su interior.

$$Contornos\ (A) = A - (A \ominus B). \tag{2.30}$$

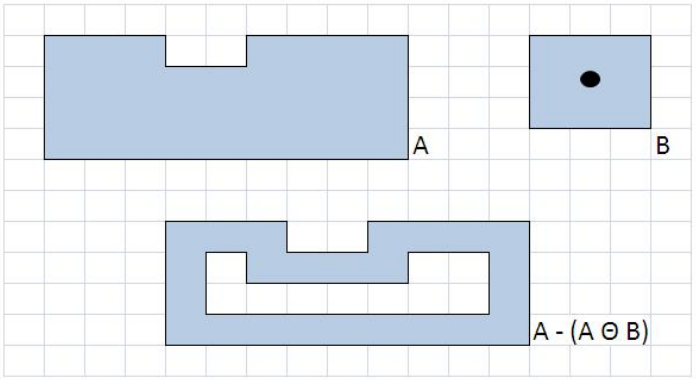


Figura 2.8: Ejemplo de extracción morfológica de contornos

Relleno de agujeros

Este filtro precisa de un proceso iterativo que concluye cuando no se producen más cambios sobre la imagen.

Se parte de X_0 igual a un punto del agujero que se desea rellenar, para luego aplicar de manera iterativa:

$$X_k = (X_{k-1} \oplus B) \cap A^c. \tag{2.31}$$

Adelgazamiento

Esta operación adelgaza los elementos de una imagen hasta que se reducen a un esqueleto interior a la misma.

Para poder realizar este tipo operación es preciso definir previamente la siguiente operación:

$$A \otimes B = A - (A * B). \tag{2.32}$$

Este paso, utilizando la operación de coincidencia estructural, elimina un punto de A cuando tanto B como su entorno encajan en A. Eligiendo cuidadosamente un conjunto de elementos B, que sólo deben erosionar los bordes de A, y repitiendo esta operación sucesivamente se obtiene el resultado deseado.

$$A \bigtriangledown \{B\} = (...((((A \otimes B1) \otimes B2) \otimes B3) \otimes B4)... \otimes Bn). \tag{2.33}$$

Por medio de la siguiente figura se ilustra el proceso de adelgazamiento de un objeto mediante esta operativa.

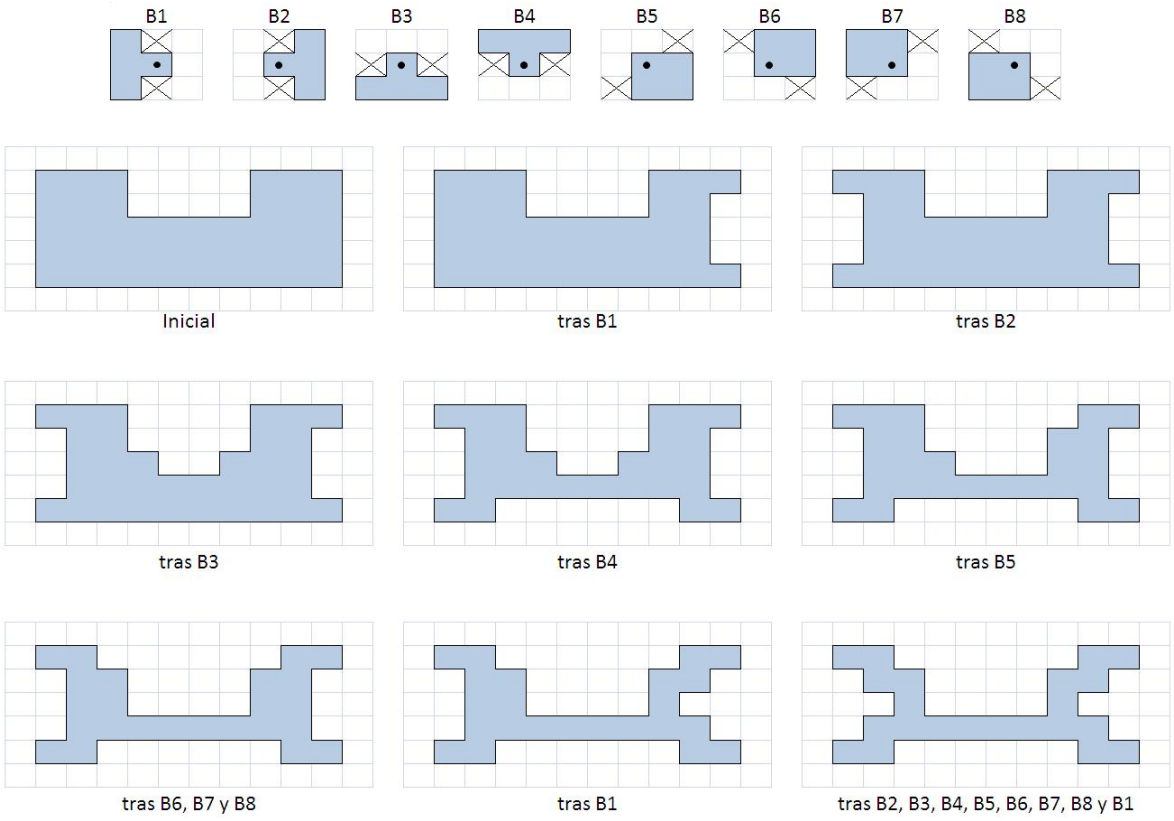


Figura 2.9: Ejemplo de aplicación del algoritmo de adelgazamiento. Las máscaras definidas van erosionando por el borde la figura original, en iteraciones sucesivas

2.3.9. Operaciones morfológicas básicas en imágenes de niveles de gris

Considerando las imágenes en niveles de gris como superficies $I(x, y)$ y tomando como elemento estructural una función $b(x, y)$ se definen las operaciones de erosión y dilatación como:

$$I(x,y) \oplus b = max\{I(x-i, y-j) + b(i,j) / (x-i, y-j) \in Dom(I) \;\; y \;\; (i,j) \in Dom(b)\}. \tag{2.34}$$

$$I(x,y) \ominus b = min\{I(x-i, y-j) - b(i,j) / (x-i, y-j) \in Dom(I) \;\; y \;\; (i,j) \in Dom(b)\}. \tag{2.35}$$

Las operaciones de apertura y cierre se definen igual que para el caso bitonal. Intuitivamente estas operaciones se corresponden con el desplazamiento del elemento estructurante b sobre la superficie $I(x, y)$ en la apertura y bajo la misma en el cierre.

2.4. Segmentación

La segmentación de una imagen digital es una de las tareas que entraña mayores dificultades en el procesamiento de imágenes. Consiste en extraer o aislar objetos considerados de interés o de importancia de acuerdo al problema que se plantee.

Es un proceso que subdivide la imagen en partes constitutivas, agrupando pixeles en regiones homogéneas respecto a una o más características. Este proceso finalizará cuando los objetos de interés sean aislados del resto.

Las técnicas de segmentación están basadas fundamentalmente en dos propiedades de los valores de intensidad, estos son:

- Discontinuidad.- Mediante esta propiedad se realiza la división de una imagen en objetos teniendo en cuenta los cambios bruscos de niveles de gris, para ello se utilizan técnicas de segmentación basadas en la extracción de bordes.
- Similitud.- Esta propiedad permite realizar la división de una imagen en objetos teniendo en cuenta la similitud que presentan ciertas regiones adyacentes de la imagen. Los métodos de segmentación que utilizan esta propiedad son los basados en la umbralización del histograma y extracción de regiones.

Considerando las propiedades citadas con anterioridad, a continuación se describen brevemente las técnicas que permiten segmentar una imagen, y que se agrupan en diversos tipos como los siguientes:

- Segmentación basada en umbralización [12].
- Segmentación basada en la detección de bordes [17], [1].
- Segmentación basada en regiones [16].

2.4.1. Segmentación basada en umbralización

Esta técnica permite clasificar las distintas partes de la imagen, en puntos de objeto y puntos de fondo, mediante el cálculo de un valor límite que realizará dicha separación o binarización.

La mayoría de las técnicas de umbralización [12], se basan en estadísticas sobre el histograma de una imagen. El histograma de una imagen considera solamente la distribución de grises en la imagen, dejando de lado la información espacial.

Si el histograma de una imagen posee picos, podemos separar dos zonas y el umbral es aquel valor que se encuentra en el valle entre ambas, es decir, el umbral que se obtiene se basa en el histograma de la imagen, mediante la búsqueda de sus mínimos locales.

En un caso ideal, el histograma de intensidad de una imagen tendría bien marcados los dos picos para zonas claras y zonas oscuras (ver figura 2.10. En este caso diríamos que el umbral óptimo es aquel valor T que separa ambas regiones.

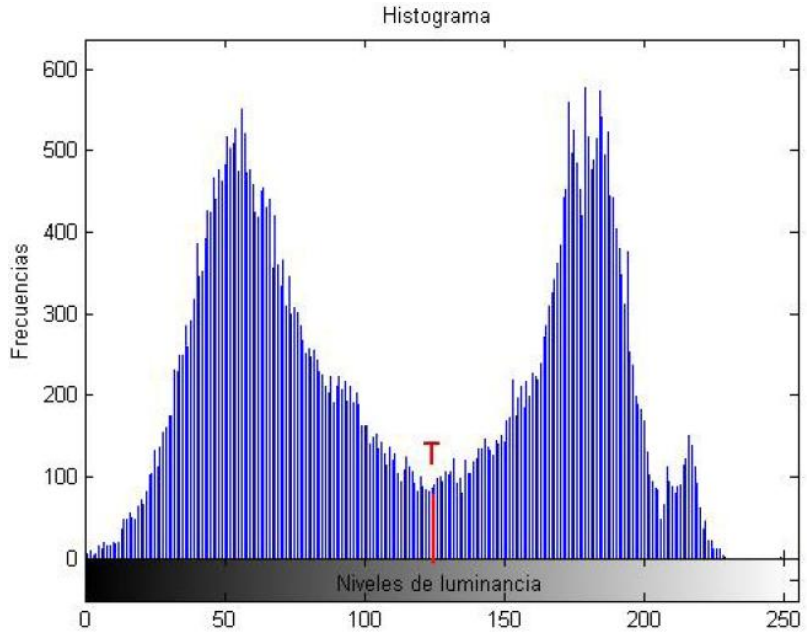


Figura 2.10: Histograma ideal

La umbralización se aplica sobre una imagen en escala de grises, la cual es binarizada consiguiendo un umbral óptimo T, y con ese valor se separan los píxeles en dos regiones, una de zonas claras y otra de zonas oscuras.

Al aplicar un umbral T, la imagen en escala de grises, $f(x, y)$, quedará binarizada; etiquetando con '1' los píxeles correspondientes al objeto y con '0' aquellos que son del fondo. Por ejemplo, si los objetos son claros respecto al fondo, se aplicará:

$$g(x,y) = \begin{bmatrix} 1 & \text{si} & f(x,y) > T \\ 0 & \text{si} & f(x,y) \leq T \end{bmatrix} \tag{2.36}$$

En el caso de que los objetos sean oscuros respecto al fondo, la asignación sería a la inversa:

$$g(x,y) = \begin{bmatrix} 1 & \text{si} & f(x,y) < T \\ 0 & \text{si} & f(x,y) \geq T \end{bmatrix} \tag{2.37}$$

El umbral puede depender de $f(x, y)$, de alguna propiedad local del píxel, $p(x, y)$, y hasta de su propia posición:

$$T = T(f(x,y), p(x,y), x, y). \tag{2.38}$$

Los umbrales elegidos pueden ser de varios tipos, dependiendo de las características tenidas en cuenta para su elección. Si el umbral sólo depende de $f(x, y)$ se dice que es un

umbral global o único; en el caso de que además dependa de $p(x, y)$, por ejemplo, el valor medio de los píxeles vecinos, el umbral es denominado local; y si depende también de la posición (x, y) del píxel, se denominará dinámico.

A modo de ejemplo, sobre la imagen que aparece en la figura 2.11, aplicamos esta técnica mediante la definición de un umbral global, y a partir de dicho valor separamos los objetos de interés del fondo de la imagen, tal y como se observa en al figura 2.12.

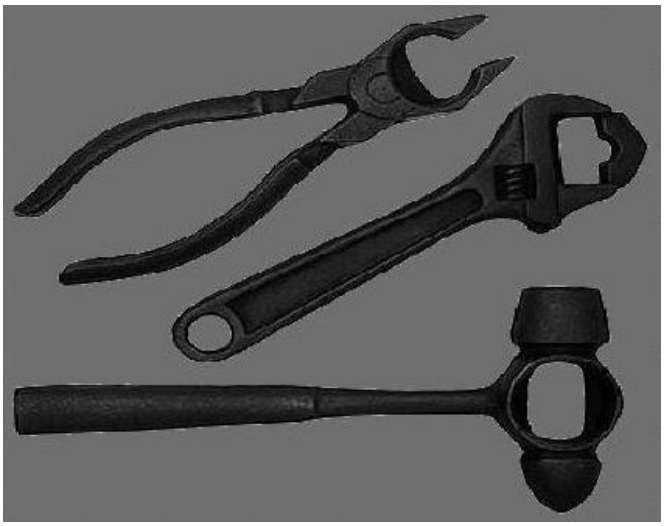


Figura 2.11: Diferencia entre los objetos y el fondo

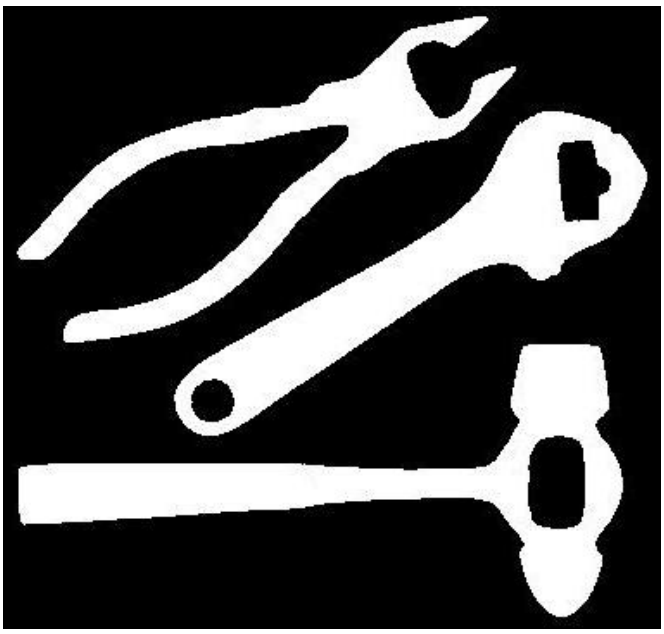


Figura 2.12: Imagen segmentada mediante umbralización

Sin embargo, para la mayoría de las imágenes reales, no existe una diferenciación clara entre los objetos y el fondo de la imagen, con lo que no es tan sencillo detectar el umbral desde el gráfico del histograma, ya que dicha representación no presentará un histograma claramente bimodal.

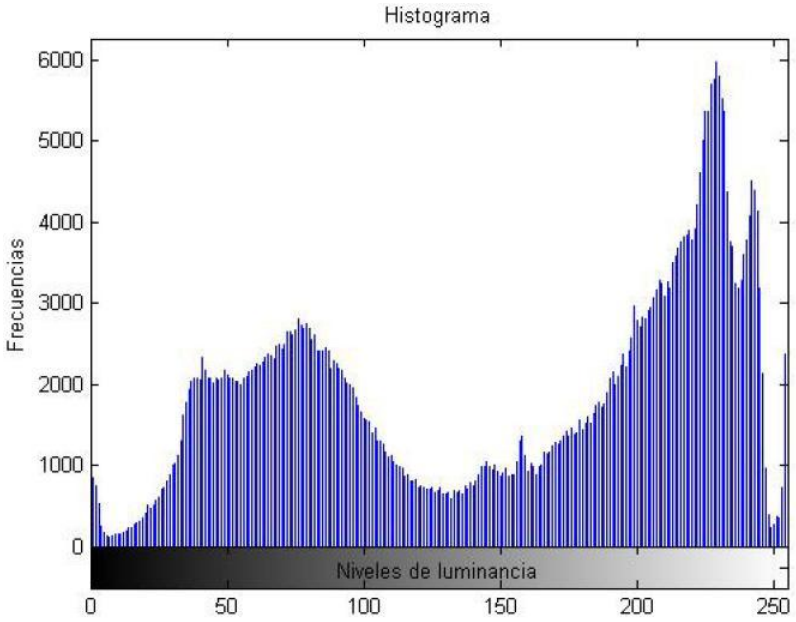


Figura 2.13: Ejemplo de histograma que no es claramente bimodal

En este tipo de imágenes debemos hacer uso de otros métodos de umbralización, con los que obtener umbrales óptimos, que permitan en una etapa posterior, distinguir los objetos a evaluar. En nuestro caso, nos centraremos en el uso del **método de Otsu** [18], para el cálculo de umbrales óptimos.

2.4.1.1. Método de Otsu para cálculo de umbral óptimo

El método de Otsu [18], calcula el umbral óptimo maximizando la varianza entre clases mediante una búsqueda exhaustiva. La importancia de este método radica en que es automático, es decir, no necesita supervisión humana ni información previa de la imagen antes de su procesamiento.

Definiendo una imagen como una función bidimensional de la intensidad del nivel de gris, y que contiene N píxeles cuyos niveles de gris se encuentran entre 1 y L. El número de píxeles con nivel de gris i se denota como f_i, y la probabilidad de ocurrencia del nivel de gris i en la imagen está dada por:

$$p_i = \frac{f_i}{N}. \tag{2.39}$$

En el caso de la umbralización en dos niveles de una imagen (a veces llamada binarización), los píxeles son divididos en dos clases: C_1, con niveles de gris $[1, ..., t]$; y C_2, con niveles de gris $[t+1, \ldots, L]$.

Entonces, la distribución de probabilidad de los niveles de gris para las dos clases son:

$$C_1 : \quad \frac{p_1}{w_1(t)}, \ldots, \frac{p_t}{w_1(t)}, \tag{2.40}$$

$$C_2 : \quad \frac{p_{t+1}}{w_2(t)}, \frac{p_{t+2}}{w_2(t)}, \ldots, \frac{p_L}{w_2(t)}, \tag{2.41}$$

donde

$$w_1(t) = \sum_{i=1}^{t} p_i \qquad\qquad w_2(t) = \sum_{i=t+1}^{L} p_i.$$

También, la media para la clase C_1 y la clase C_2 es:

$$\mu_1 = \sum_{i=1}^{t} \frac{i.p_i}{w_1(t)} \qquad\qquad \mu_2 = \sum_{i=t+1}^{L} \frac{i.p_i}{w_2(t)}.$$

Sea μ_T la intensidad de toda la imagen. Es fácil demostrar que:

$$w_1.\mu_1 + w_2.\mu_2 = \mu_T \qquad\qquad w_1 + w_2 = 1.$$

En este momento, se considera oportuno para tener bien claro lo comentado, ilustrarlo con un ejemplo. Supongamos una imagen de $N = 100$ píxeles con cuatro niveles de gris comprendidos en $[1, 4]$(1 el negro, 4 el blanco), y supongamos a su vez que el número de píxeles con nivel de gris 1 es 10, con nivel de gris 2 es 20, con nivel de gris 3 es 30, y con nivel de gris 4 es 40, es decir, $f_1 = 10$, $f_2 = 20$, $f_3 = 30$ y $f_4 = 40$. Luego, $p_1 = \frac{f_1}{N} = 0{,}1$, $p_2 = 0{,}2$, $p_3 = 0{,}3$ y $p_4 = 0{,}4$.

Entonces, para una umbralización en dos niveles de esta imagen tomamos $t = 2$ de manera que la clase C_1 consista en los tonos de gris 1 y 2, y la clase C_2 posea los tonos 3 y 4. De esta manera, $w_1(t) = 0{,}1 + 0{,}2 = 0{,}3$ y $w_2(t) = 0{,}3 + 0{,}4 = 0{,}7$, y se comprueba que $w_1(t) + w_2(t) = 1$. Por último, la media para la clase C_1 y para la clase C_2 estará dada por:

$$\mu_1 = \sum_{i=1}^{2} \frac{i.p_i}{w_1(t)} = \frac{1*0{,}1+2*0{,}2}{0{,}3} \approx 1,667.$$

$$\mu_2 = \sum_{i=3}^{4} \frac{i.p_i}{w_2(t)} = \frac{3*0,3+4*0,4}{0,7} \approx 3,57,$$

y $$\mu_T = w_1.\mu_1 + w_2.\mu_2 = 0,3 * 1,667 + 0,7 * 3,57 \approx 3.$$

Sigamos con el método. Usando análisis discriminante, Otsu definió la varianza entre clases de una imagen umbralizada como:

$$\sigma_B^2 = w_1.(\mu_1 - \mu_T)^2 + w_2.(\mu_2 - \mu_T)^2. \tag{2.42}$$

Para la umbralización de dos niveles, Otsu verificó que el umbral óptimo t^* se elige de manera que σ_B^2 sea máxima, esto es:

$$t^* = \underset{t}{Max} \left\{\sigma_B^2(t)\right\} \qquad 1 \leq t \leq L. \tag{2.43}$$

El método de *Otsu* puede extenderse fácilmente a múltiples umbrales. Para ello, asumiendo que hay $M-1$ umbrales, $\{t_1, t_2, \ldots, t_{M-1}\}$, los cuales dividen a la imagen en M clases: C_1 para $[1, \ldots, t_1]$, C_2 para $[t_1+1, \ldots, t_2], \ldots, C_i$ para $[t_{i-1}+1, \ldots, t_i], \ldots,$ y C_M para $[t_{M-1}, \ldots, L]$, los umbrales óptimos $\{t_1^*, t_2^*, \ldots, t_{M-1}^*\}$ se eligen maximizando σ_B^2 como sigue:

$$\{t_1^*, t_2^*, \ldots, t_{M-1}^*\} = \underset{t_1,t_2,\ldots,t_{M-1}}{Max} \left\{\sigma_B^2(t_1, t_2, \ldots, t_{M-1})\right\}, \tag{2.44}$$
$$1 \leq t_1 < \ldots < t_{M-1} < L.$$

Donde,

$$\sigma_B^2 = \sum_{k=1}^{M} w_k.(\mu_k - \mu_T)^2. \tag{2.45}$$

Con

$$w_k = \sum_{i \in C_k} p_i \qquad \mu_k = \sum_{i \in C_k} \frac{i.p_i}{w_k}.$$

w_k es conocido como momento acumulado de orden cero de la k-ésima clase C_k, y el numerador de la última expresión es conocido como momento acumulado de primer orden de la k-ésima clase C_k. Con lo que tenemos:

$$\mu(k) = \sum_{i \in C_k} i.p_i.$$

2.4.2. Segmentación basada en la detección de bordes

El objetivo de estas técnicas es detectar discontinuidades significativas en los valores de intensidad como por ejemplo: discontinuidades en el fondo, discontinuidades en orientación superficial, cambios de propiedades materiales, variaciones en la iluminación de la escena, etc. Tales discontinuidades marcarán los bordes, es decir los lugares donde la intensidad de la imagen cambia rápidamente. Se emplean diferentes algoritmos usando la primera y segunda derivada.

En lo sucesivo se asumirá que las regiones objeto de análisis, son lo suficientemente homogéneas como para que la transición entre dos regiones pueda ser determinada únicamente en base a las discontinuidades de sus niveles de gris. Si no se cumple la hipótesis anterior se ha de recurrir a técnicas de umbralización o segmentación orientada a regiones.

A continuación se presentan los conceptos y definiciones involucrados en el tipo de segmentación que atañe a la presente sección:

- Punto de borde: Es un punto de la imagen con coordenadas (x, y) localizados en un cambio significativo de intensidad de la imagen (figura 2.14).
- Contorno: Es una lista de bordes o la curva matemática que permite modelar la lista de bordes.
- Unión de bordes: Es el proceso de formar una lista ordenada de bordes a partir de una lista no ordenada.

La segmentación de bordes es el proceso de buscar *píxeles* de igual nivel de gris en la imagen para determinar contornos y poder realizar procesos de segmentación. Los métodos empleados para generar detectores diferenciales involucran el cálculo del gradiente en dos direcciones sobre la imagen.

El gradiente de una imagen $f(x, y)$ en un punto (x, y) viene dado por el vector:

$$\nabla f(x,y) = \begin{bmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{bmatrix},$$

MAGNITUD $$|\nabla f(x,y)| = \sqrt{(\frac{\partial f(x,y)}{\partial x})^2 + (\frac{\partial f(x,y)}{\partial y})^2} \cong |G_x| + |G_y|,$$

DIRECCIÓN $$\alpha(x,y) = arctang(\frac{\frac{\partial f(x,y)}{\partial x}}{\frac{\partial f(x,y)}{\partial y}}).$$

La magnitud del gradiente $f(x, y)$ es independiente de la dirección de la discontinuidad. El vector gradiente, se puede calcular de varias formas diferentes para imágenes digitales, siendo la aproximación más sencilla el cálculo de las diferencias.

$$G_x \cong f[i, j+1] - f[i, j], \tag{2.46}$$

$$G_y \cong f[i, j] - f[i+1, j]. \tag{2.47}$$

Esta aproximación puede ser implementada utilizando las máscaras de convolución.

G_x =	-1	1	G_y =	1	
				-1	
G_x =	-1	1	G_y =	1	1
	-1	1		-1	-1

Tabla 2.7: Máscaras de gradiente de una imagen

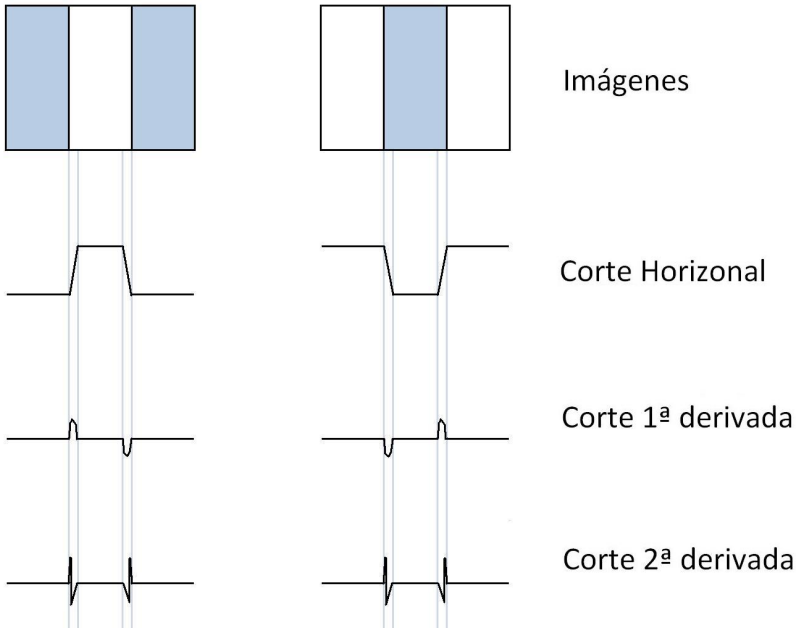


Figura 2.14: Bordes en una imagen

El valor de la primera derivada puede usarse para determinar la presencia de un borde. La primera derivada de cualquier punto en la imagen puede ser obtenida usando el valor del gradiente de dicho punto.

El signo de la segunda derivada determina el lado de dicho borde y viene dado por el operador Laplaciano.

2.4.2.1. Máscara de convolución

Para la utilización de los algoritmos de detección de bordes y de filtrado de imágenes se debe tener en cuenta la convolución que se va a efectuar entre la imagen a procesar y la máscara que determine la operación a realizarse.

La máscara se va posicionando en cada uno de los *píxeles* de la imagen y se calcula la suma de los productos de cada uno de los coeficientes, con el nivel de gris contenido en la región englobada por dicha máscara (ventana). Por tanto, la respuesta a cada uno de los puntos de la imagen viene dada por:

$$R = w_1 I_1 + w_2 I_2 + ... + w_9 I_9 = \sum_{i=1}^{9} w_i I_i. \tag{2.48}$$

Donde I_i es el nivel de gris del píxel asociado con el coeficiente de máscara w_i.

Si la máscara se encuentra centrada en uno de los píxeles de la primera fila, primera columna, última fila o última columna de la imagen, la respuesta R se calcula usando la vecindad parcial apropiada y considerando dichas filas y columnas se encuentran duplicadas para tener vecindades completas.

Esquema general de detección de discontinuidades de una imagen con máscaras de convolución:

w1	w2	w3
w4	w5	w6
w7	w8	w9

$$R = w_1 I_1 + w_2 I_2 + ... + w_9 I_9 = \sum_{i=1}^{9} w_i I_i$$

Figura 2.15: Esquema general de detección de discontinuidades de una imagen con máscaras de convolución

La detección de puntos, consiste en tratar de detectar puntos (*píxeles*) en la imagen, que desentonan su nivel de gris respecto al nivel de gris de sus vecinos (el número de vecinos depende del tamaño de la máscara).

La máscara básica utilizada en este tipo de detección es:

-1	-1	-1
-1	8	-1
-1	-1	-1

$$R = -I_1 - I_2 - I_3 - I_4 + 8I_5 - I_6 - I_7 - I_8 - I_9$$

Figura 2.16: Máscara básica utilizada en este tipo de detección

Se dice que un punto ha sido detectado si $|R| > T$, siento T un umbral(grado de destaque). Si $R = 0$ zona homogénea de la imagen, igual nivel de gris. Esta máscara también se aplica para eliminación de ruido.

2.4.2.2. Operador de Roberts

El operador de Roberts es una aproximación a la magnitud del gradiente, en la cual se devuelve un dato acerca de si hay o no un punto de borde, pero no da la orientación, su funcionamiento es mejor con imágenes binarias.

$$\nabla[f[x,y]] = |f[x,y] - f[x+1,y+1]| + |f[x+1,y] - f[x,y+1]|. \tag{2.49}$$

A continuación se presentan las máscaras usadas. Donde G_x y G_y son calculados usando las siguientes máscaras.

G_x =

1	0
0	-1

G_y =

0	-1
1	0

Tabla 2.8: Máscara del Operador de Roberts

Con el operador gradiente de $2x2$, las diferencias son calculadas en el punto de interpolación $(i + 1/2, \quad j + 1/2)$.

2.4.2.3. Operador de Sobel

Calcular el operador gradiente en el eje x e y, es equivalente a aplicar sobre la imagen las siguientes máscaras, conocidas como operador de Sobel [14]:

G_x =

-1	0	1
-2	0	2
-1	0	1

G_y =

1	2	1
0	0	0
-1	-2	-1

Tabla 2.9: Máscara del Operador de Sobel

La detección se realiza con un ángulo de 90º sobre la dirección que aparentemente tiene el borde. Enfatiza el píxel central de la máscara (lo multiplica por 2).

Los operadores de Sobel son rápidos y efectivos. Sin embargo, no proporcionan el valor real del gradiente, sino una imagen sobre la que se pueden realizar cálculos referentes al contorno.

Este método da relieve al contraste (marca bordes) entre regiones homogéneas, al mismo tiempo que produce un adelgazamiento.

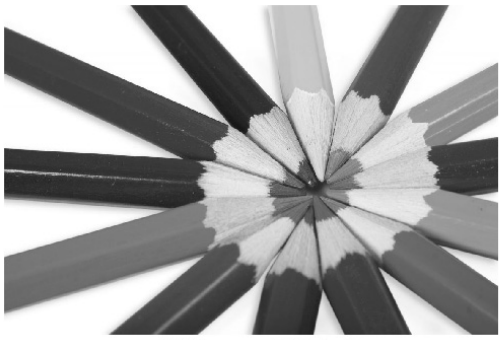
2.4.2.4. Operador de Prewitt

El operador de Prewitt [1] es similar al de Sobel con diferentes coeficientes (c=1). Las máscaras se muestran a continuación:

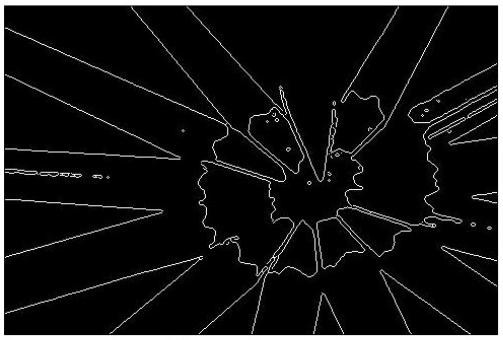
$$G_x = \begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline \end{array} \qquad G_y = \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -1 & -1 \\ \hline \end{array}$$

Tabla 2.10: Máscara del Operador de Prewitt

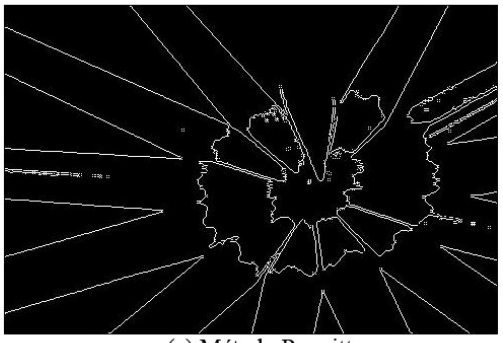
No se enfatiza el píxel central de la máscara. La detección se realiza con un ángulo de 90º, sobre la dirección que aparentemente tiene el borde, en la figura 2.17 se tiene el procesamiento en la obtención de bordes utilizando dos diferentes métodos como son el de Canny ([13]) figura 2.17(b) y el método de Prewitt figura 2.17 (c).



(a) Imagen Original



(b) Método Canny



(c) Método Prewitt

Figura 2.17: Detección de bordes de una imagen

2.4.3. Segmentación basada en regiones

Este tipo de segmentación tiene por objeto dividir una imagen en regiones bien delimitadas [16], evitando que exista superposición entre ellas, de tal manera que se construyen las regiones directamente usando información de la conectividad para crearlas.

Una región es un conjunto de píxeles que son adyacentes, que siguen ciertas reglas de conectividad y que en general se definen entre los cuatro o los ocho vecinos más cercanos. Comúnmente se emplea la conectividad con los ocho vecinos más cercanos, ya que ésta se aproxima bien a la clasificación intuitiva que los seres humanos efectúan cotidianamente. Por lo tanto, esta va a ser el tipo de conectividad empleada en los algoritmos de segmentación basados en regiones que se exponen en el capítulo 4, de *experimentos y aplicaciones desarrolladas*.

La forma de un objeto se puede describir por medio de sus límites (bordes) o en términos de la región que éste ocupa. Los dos enfoques son complementarios. En el caso de regiones se desea definir características que distingan una sección de la imagen de otras y que todas ellas sean homogéneas dentro de la región de interés. Algunos elementos de este tipo son la textura y la intensidad de la imagen.

Antes de presentar los diferentes métodos de segmentación por regiones, se definen algunas propiedades útiles.

- La segmentación es un proceso que divide una imagen R en n subregiones $R_1, R_2, \ldots, R_n$, tal que se deben cumplir las siguientes condiciones:

  a) $U_{i=1}^{n} R_i = R$. La segmentación debe ser completa, donde cada píxel debe estar en una región.

  b) R_i es una región conexa, para $i = 1, 2, \ldots, n$. Los puntos de una región deben estar conectados.

  c) $R_i \cap R_j = \phi$ para todo i y j, $i \neq j$. Las regiones deben ser disjuntas.

  d) $P(R_i) = Verdadero$ para $i = 1, 2, \ldots, n$. Los píxeles de una región segmentada deben cumplir una propiedad (condición), por ejemplo $P(R_i) = Verdadero$ si todos los píxeles de R_i tienen el mismo nivel de gris.

  e) $P(R_i \cup R_j) = Falso$ para $i \neq j$. Las regiones R_i y R_j son diferentes bajo la condición P.

$P(R_i)$ es un predicado definido en R_i, y ϕ es el conjunto vacío. El predicado es una condición que se verifica en todos los puntos pertenecientes a una región y que es falsa en las demás (niveles de gris, gradiente, textura, etc).

La segmentación basada en regiones se divide en dos grupos de métodos, como son los **métodos de crecimiento de regiones** y los **procedimientos de separación y unión** [16], que se describen en las siguientes secciones.

2.4.3.1. Métodos de crecimiento de regiones

Estos métodos se basan en realizar un procedimiento iterativo que consiste en agrupar píxeles o regiones adyacentes que cumplan criterios tales como: nivel de intensidad, varianza de los niveles de la región, y textura de la región, para lo cual se establecen puntos semilla, que son una colección de píxeles iniciales en las regiones que crecen. Es decir, a estos píxeles semillas se les agregan otros píxeles adyacentes y con alguna propiedad similar a la semilla, esto quiere decir que pasarán a pertenecer a la misma región y a tener los mismos valores que los puntos semilla.

Por lo general el criterio de agregación que se utiliza consiste en en que un píxel adyacente a una región se agregará a ésta si su intensidad es similar a la de los píxeles de la región. El algoritmo finaliza cuando dejan de haber regiones adyacentes que cumplan los criterios.

El principio de operación por el que se rige este tipo de métodos, se va a exponer a continuación mediante la exposición de un ejemplo concreto. Para ello, a partir de la subimagen siguiente, representada por una matriz, donde los números dentro de las celdas representan niveles de gris, y los puntos de coordenadas $(3, 2)$ y $(3, 4)$ son las semillas consideradas:

$$\begin{array}{ccccc} 0 & 0 & 5 & 6 & 7 \\ 1 & 1 & 5 & 8 & 7 \\ 0 & [1] & 6 & [7] & 7 \\ 2 & 0 & 7 & 6 & 6 \\ 0 & 1 & 5 & 6 & 5 \end{array}$$

Utilizando los dos puntos semilla indicados, se efectúa una partición en dos regiones, A y B, siguiendo la propiedad:

$$|f(x, y) - Fsemilla| < T.$$

Es decir, un umbral con respecto a los niveles de gris, por lo tanto, si tomamos como umbral a $T = 3$, tenemos:

$$\begin{array}{ccccc} A & A & B & B & B \\ A & A & B & B & B \\ A & A & B & B & B \\ A & A & B & B & B \\ A & A & B & B & B \end{array}$$

En este caso no importa la selección de las raíces para ninguna de las regiones. Sin embargo, resulta más importante la selección del valor de umbral T. Si $T = 8$, se obtiene una sola región:

$$\begin{matrix} A & A & A & A & A \\ A & A & A & A & A \\ A & A & A & A & A \\ A & A & A & A & A \\ A & A & A & A & A \end{matrix}$$

Las características importantes para este método son la determinación de las semillas iniciales y la naturaleza de la propiedad de agrupación (criterio de similitud). En el caso de imágenes a color, se emplean los componentes RGB y reglas de segmentación de la forma,

$$P(R, x, t) : (fR(k, l) < TR)\&\&(fG(k, l) < TG)\&\&(fB(k, l) < TB).$$

Las siguientes figuras presentan un ejemplo sintético de la segmentación por crecimiento de regiones. En este caso la propiedad de agregación es el color idéntico:

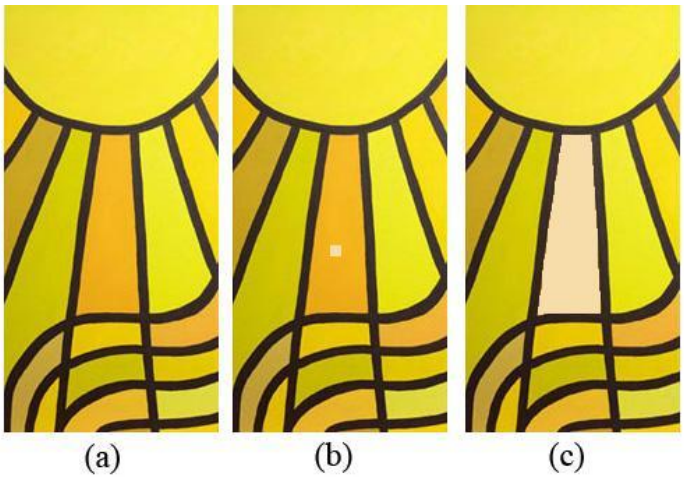


Figura 2.18: Ejemplo de crecimiento de regiones: imagen a tratar (a), punto inicial de crecimiento (b) y resultado final (c)

2.4.3.2. Procedimientos de separación y unión (*split and merge*)

Este procedimiento utiliza un algoritmo que tiene como entrada una imagen y como salida devuelve una imagen formada por regiones homogéneas. Esta técnica se divide en dos fases: la primera consiste en dividir la imagen en sub-imágenes o regiones disjuntas, y la segunda fase consiste en unificar las sub-imágenes similares.

Para poder efectuar el proceso indicado con anterioridad, se hace necesario utilizar una medida de similitud de regiones, para lo cual se puede utilizar los valores de la media y la desviación típica de los niveles de gris de los píxeles de las regiones.

Los píxeles que se encuentran en las mismas vecindades tienden a tener valores estadísticos similares y a encontrarse dentro de las mismas regiones. Una manera fácil de comenzar a hacer segmentaciones es a partir de píxeles que se denominarán *semilla* efectuando un *crecimiento* de las regiones a su alrededor.

Normalmente estas semillas son definidas por el usuario (modo supervisado), y hay una de ellas en cada región. Para describir el mecanismo del crecimiento se definen reglas de pertenencia o de homogeneidad de las regiones adyacentes. En cada etapa y para cada región se revisa para ver si hay píxeles sin clasificar en su vecindad (8 vecinos más próximos). Antes de efectuar la asignación de un píxel a una región, se verifica que cumpla con los criterios de homogeneidad de la región.

$$P(R_i^{(k)} \cup \{x\}) = Verdadero.$$

Para efectuar la unión entre dos regiones se debe determinar primero si éstas son suficientemente parecidas desde el punto de vista estadístico [4]. Normalmente se emplea la media aritmética y la media y desviación estándar como criterio:

$$m_i = \frac{1}{n}. \sum_{(k,l)\in R_i} f(k,l), \tag{2.50}$$

$$\sigma_i = \sqrt{\frac{1}{n}. \sum_{(k,l)\in R_i} (f(k,l) - m_i)^2}. \tag{2.51}$$

De forma que para decidir si dos regiones R_1 y R_2 deben incorporarse en una única región, comprobaremos si sus medias son similares, es decir, se considera la siguiente ecuación:

$$|m_1 - m_2| < k.\sigma_i \quad i = 1, 2. \tag{2.52}$$

A diferencia de la unión de píxeles, en el caso de las operaciones de separación o división, éstas se basan en un procedimiento que se inicia bajo la suposición de que toda la imagen es homogénea, y que si no se cumplen las condiciones, la imagen se divide en cuatro subregiones. Este procedimiento continúa hasta que se encuentren regiones homogéneas.

Por lo tanto, dado R como una imagen entera y P una propiedad (predicado) de homogeneidad, suponiendo que se tiene una imagen cuadrada, una aproximación para segmentar R es subdividir la imagen sucesivamente en cuadrantes menores, de forma que se cumpla para cada región R_i, $P(R_i) = Verdadero$. Esto es, si $P(R) = Falso$, se procede a dividir el cuadrante en subcuadrantes, y se vuelve a repetir el proceso con los subcuadrantes obtenidos.

Esta técnica de división puede representarse muy bien a través de los llamados *árboles cuaternarios (quad-trees)*, esto es, un árbol en el que cada nodo tiene cuatro descendientes (ver figura 2.19).

Obsérvese que la raíz del árbol corresponde a la imagen original y que cada nodo es una subdivisión. En este caso, sólo R_4 se ha dividido más.

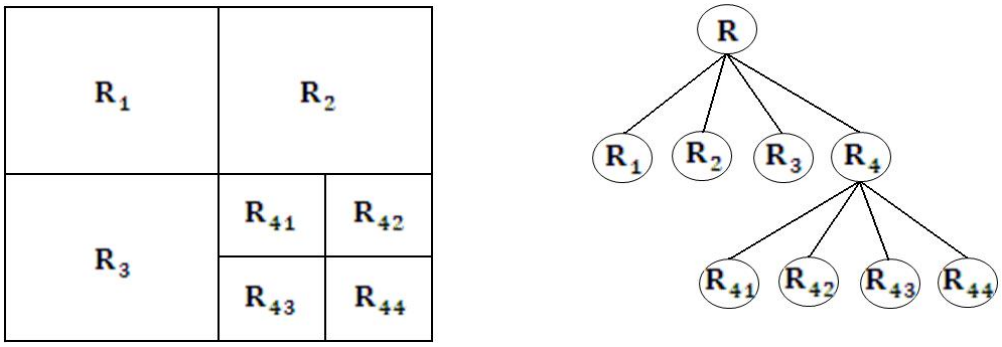


Figura 2.19: Ejemplo de árbol cuaternario

Si sólo se realizaron procesos de separación se podría terminar con regiones idénticas que fueran adyacentes pero no identificadas como una misma región. Este inconveniente puede resolverse si además de la separación se permite la unión. La idea es unir regiones adyacentes cuyos píxeles combinados satisfagan el predicado P, esto es dos regiones adyacentes R_i y R_k son unidas cuando $P(R_i \cap R_k) = Verdadero$.

Lo comentado con anterioridad se puede resumir en el procedimiento siguiente; Dividir en cuatro regiones disjuntas cualquier región R_i para la que $P(R_i) = Falso$, unir cualquiera dos regiones adyacentes R_i y R_k para la que $P(R_i \cap R_k) = Verdadero$, y parar cuando no sea posible realizar más uniones o divisiones.

Gráficamente, a modo de ejemplo este proceso se puede observar por medio de la siguientes figuras:

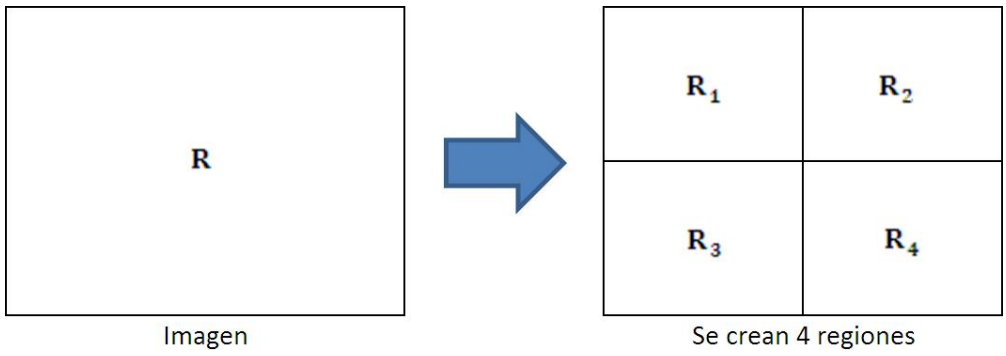


Figura 2.20: Ejemplo procedimiento de división y unión de regiones (1)

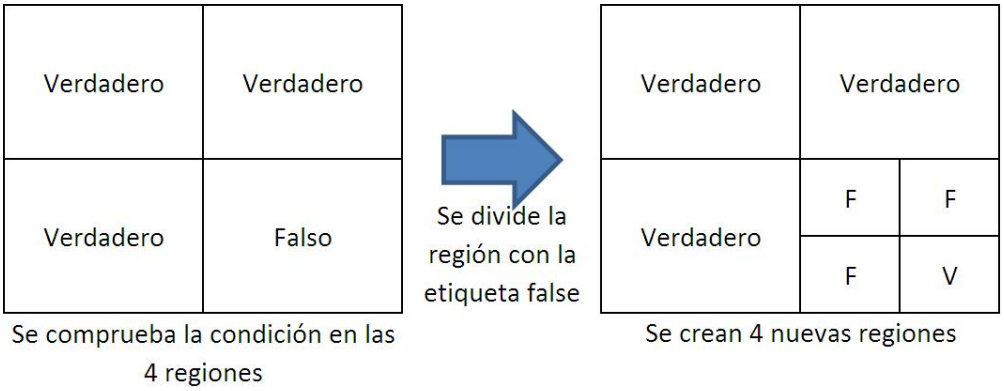


Figura 2.21: Ejemplo procedimiento de división y unión de regiones (2)

Figura 2.22: Ejemplo procedimiento de división y unión de regiones (3)

2.4.3.3. Descriptores basados en regiones

Los *descriptores basados en regiones* [16], se utilizan cuando el interés principal es describir las propiedades de la región, y se pueden obtener del análisis del propio contorno del objeto o por características internas del mismo.

Existen varios descriptores que pueden ser calculados sobre las regiones, que pueden ser útiles en muchas tareas de descripción, siempre que las formas a describir no sean excesivamente complejas. Entre los descriptores de mayor utilidad podemos destacar los siguientes:

- **Área**: Número de píxeles que pertenecen a la región. En la representación habitual de una imagen en matriz, el cálculo de área se reduce a una cuenta de píxeles.

$$Area = \sum_{y=0}^{R-1} \sum_{x=0}^{C-1} p(x,y), \tag{2.53}$$

donde

$p(x, y) =$ Valor gris del píxel (valores 0 o 1 en imágenes binarias).
$R =$ Número de filas.
$C =$ Número de columnas.

Para calcular el área real del objeto se debe tener en cuenta la resolución del píxel en el mundo real.

- **Perímetro**: El cálculo del *perímetro normal* de una región, consiste en el recuento de los píxeles que constituyen el borde de la región en cuestión. Se determina sumando los píxeles del contorno y multiplicando por un factor. Los puntos horizontales y verticales tienen un factor de 1, mientras que en los diagonales su factor de es $\sqrt{1^2 + 1^2} = \sqrt{2}$ correspondiente a la longitud de la diagonal del píxel.

De esta forma el *perímetro normal* se obtiene por medio de la siguiente fórmula:

$$P = \sum v + \sum h + \sum d.\sqrt{2}, \tag{2.54}$$

con

$v =$ píxeles verticales.
$h =$ píxeles horizontales.
$d =$ píxeles diagonales.

Otra medida relacionada con este parámetro es el *perímetro convexo*, que básicamente consiste en calcular el perímetro que tendría la región, considerando que no presenta ninguna concavidad en su contorno, es decir, se define como la longitud de una línea convexa que circunscribe el objeto a medir.

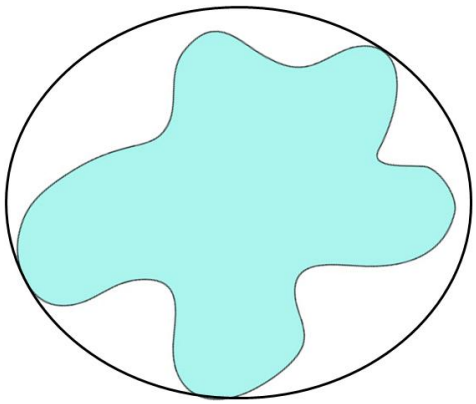


Figura 2.23: Ejemplo de perímetro convexo sobre región

- **Número de Euler**: El número de huecos (H) y el número de componentes conectadas (C), es decir, el número de partes contiguas en el objeto, nos permiten calcular el número de Euler, por medio de la siguiente expresión:

$$E = C - H. \tag{2.55}$$

Este número es invariante frente a traslaciones, rotaciones y cambios de escala, y nos permite de forma sencilla discriminar entre ciertas clases de objetos.

Por medio de la figura 2.24, podemos advertir unos ejemplos de cálculo del número de Euler. Para ello, observamos que la letra *A* tiene un número de Euler igual a 0, ya que el número de componentes conectadas (conexas) es 1 y el número de huecos es 1. Mientras que para la letra *B* su valor es de -1 ($C = 1$, $H = 2$).

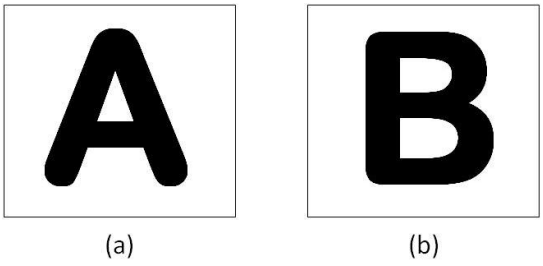


Figura 2.24: Regiones con el número de Euler igual a 0 (a) y -1 (b)

- **Alargamiento**: El *alargamiento o elongación* mide la relación entre el ancho (L_a) y el alto (L_b), del menor rectángulo que envuelve completamente la región (ver figura 2.25).

$$Alargamiento = \frac{L_b}{L_a}. \tag{2.56}$$

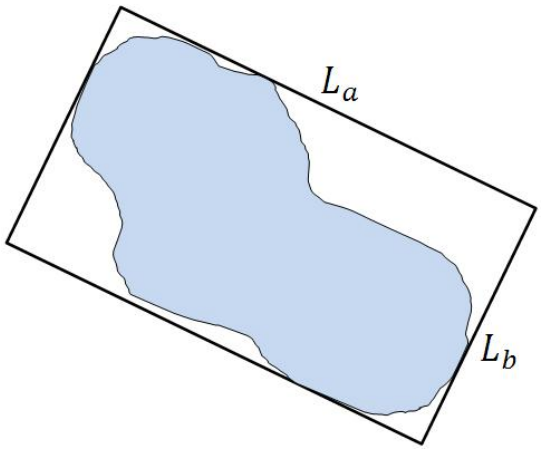


Figura 2.25: Ejemplo identificación lados del menor rectángulo que envuelve la región

- **Excentricidad**: Mide la relación entre los tamaños mayor y menor del objeto. La forma más simple de medirlo es calcular la relación entre la longitud de la cuerda mayor, respecto a la longitud de la cuerda mayor perpendicular a la cuerda mayor, es decir:

$$Excentricidad = \frac{B}{A}, \tag{2.57}$$

con

$A =$ Longitud cuerda mayor perpendicular a la cuerda mayor.
$B =$ Longitud de la cuerda mayor.

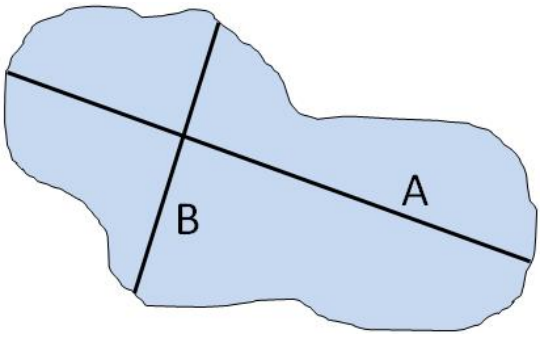


Figura 2.26: Ejemplo de excentricidad

- **Rectangularidad**: Sea F_k la relación entre el área de la región y el área del mínimo rectángulo que la envuelve, cuando el rectángulo tiene dirección k.

$$Rectangularidad = \underset{k}{Max}(F_k), \tag{2.58}$$

donde

$$F_k = \frac{Area}{L_a.L_b}.$$

La **dirección** es una propiedad que sólo tiene sentido para regiones con un valor de alargamiento significativo, y viene dada por la dirección del lado largo del rectángulo que la envuelve.

- **Caja contenedora (*bounding box*)**: Corresponde al rectángulo más pequeño que contiene completamente la región (alineado con los ejes de coordenadas), es decir, el rectángulo más pequeño que se ajusta al contorno.

Está formado por las coordenadas X_{min}, Y_{min}, X_{max} e Y_{max}, tal y como se puede observar por medio la figura 2.27.

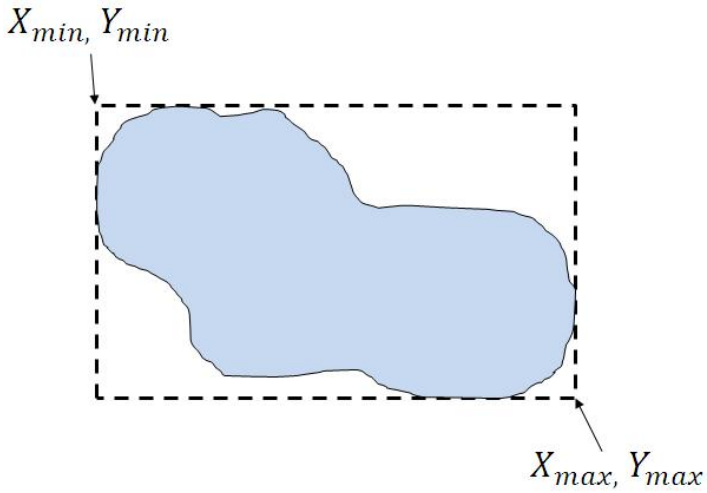


Figura 2.27: Ejemplo de caja contenedora

- **Compacidad**: La compacidad, a veces también llamada *índice de circularidad* es un tipo de descripción muy usada que relaciona la longitud del perímetro con el área, viene definida por:

$$Compacidad = \frac{Perimetro^2}{Area}. \tag{2.59}$$

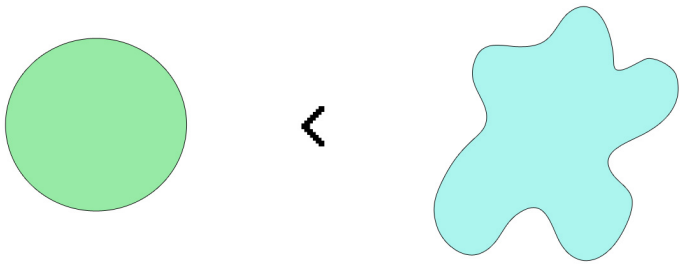


Figura 2.28: Ejemplo de compacidad en regiones

- **Centro de gravedad**: El centro de gravedad o mejor llamado *Centroide*, se puede calcular de dos formas: como la media de todas las coordenadas de los puntos del contorno, o como la media de todas las coordenadas de los puntos internos del objeto. Este segundo método, aunque mucho más lento, es más preciso.

  Dado un objeto formado por un conjunto de puntos etiquetados con un valor determinado, el cálculo de la coordenada X_c del centro de gravedad (el de la coordenada Y_c se calcula de forma análoga) se obtiene mediante:

$$X_c = \frac{m_{10}}{m_{00}}, \tag{2.60}$$

con

$$m_{10} = \sum_{y=0}^{R-1} \sum_{x=0}^{C-1} x.p(x,y) \qquad m_{00} = \sum_{y=0}^{R-1} \sum_{x=0}^{C-1} p(x,y),$$

donde

$X_c =$ Posición x del centro de gravedad.
$x =$ Coordenada x del píxel.
$p(x,y) =$ Valor gris del píxel(valores 0 o 1 en imágenes binarias).
$R =$ Número de filas.
$C =$ Número de columnas.

Momentos invariantes de Hu

Una región dada por sus puntos interiores se puede describir usando *momentos invariantes a escala, rotación y traslación* (*Momentos invariantes de Hu*).

Los *Momentos Invariantes de Hu* se describen a partir de los *Momentos Centrales* y los *Momentos Estadísticos*.

Sea $f(x,y)$ el nivel de gris de un punto (x,y) perteneciente a la región, los *Momentos Estadísticos* se definen como:

$$m_{pq} = \sum_{x=-\infty}^{\infty} \sum_{y=-\infty}^{\infty} x^p y^q f(x,y). \tag{2.61}$$

Los momentos centrales son definidos por:

$$\mu_{pq} = \sum_{x=-\infty}^{\infty} \sum_{y=-\infty}^{\infty} (x-\bar{x})^p (y-\bar{y})^q f(x,y), \tag{2.62}$$

donde

$$\bar{x} = \frac{m_{10}}{m_{10}} \qquad \bar{y} = \frac{m_{01}}{m_{00}}.$$

Los momentos normalizados no escalados, η_{pq}, se definen por:

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}}, \tag{2.63}$$

donde

$$\gamma = 1 + \frac{p+q}{2}.$$

Y entonces, usando los *Momentos normalizados* de órdenes 2 y 3, se generan el siguiente conjunto de siete momentos invariante a traslación, rotación y escalado (*Momentos de* H_u):

$$\phi_1 = \eta_{20} + \eta_{02},$$

$$\phi_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2,$$

$$\phi_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2,$$

$$\phi_4 = (\eta_{30} - \eta_{12})^2 + (\eta_{21} + \eta_{03})^2,$$

$$\phi_5 = (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} - \eta_{03})^2],$$

$$\phi_6 = (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}),$$

$$\phi_7 = (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + (3\eta_{12} - \eta_{30})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2].$$

Estos 7 momentos (*Momentos de* H_u) dan 7 valores numéricos que no son sensibles a transformaciones de las imágenes como cambios de escala, traslación o rotación.

Capítulo 3

Interfaz Gráfica de Usuario

3.1. Introducción

Con objeto de facilitar la interacción entre el usuario y el sistema, se desarrolla una interfaz gráfica principal. De esta manera, no es necesario que el usuario deba poseer nociones sobre la programación de la aplicación, y se puede centrar exclusivamente en las funcionalidades de la misma.

Dado que todos los desarrollos software llevados a cabo para implementar los diferentes algoritmos de procesamiento de imágenes, objeto del presente proyecto fin de carrera, se desarrollan con *Matlab*® [3], y para evitar posibles problemas de interoperabilidad entre lenguajes, se opta por utilizar este mismo lenguaje (*Matlab*®) en la programación de la interfaz gráfica de usuario.

3.2. Acerca de Matlab

MATLAB [3], nombre abreviado de "*MATrix LABoratory*", es un programa muy potente para realizar cálculos numéricos con vectores y matrices. Como caso particular puede trabajar también con números escalares, tanto reales como complejos. Una de las capacidades más atractivas que ofrece, es la posibilidad de realizar una amplia variedad de gráficos en dos y tres dimensiones (aunque este último no se va a explotar para la realización de este proyecto). *Matlab*® se utiliza ampliamente en:

- Cálculos numéricos.
- Desarrollo de algoritmos.
- Modelado, simulación y prueba de prototipos.
- Análisis de datos, exploración y visualización.
- Graficación de datos con fines científicos o de ingeniería.

- Desarrollo de aplicaciones que requieran de una interfaz gráfica de usuario (*GUI, Graphical User Interface*) [2].

3.2.1. Implementación de interfaz gráfica

Matlab® permite desarrollar fácilmente un conjunto de pantallas (paneles) con botones, menús, ventanas, etc., que permiten utilizar de manera muy simple programas realizados dentro de este entorno. Este conjunto de herramientas se denomina interfaz gráfica de usuario (*GUI*).

Las posibilidades que ofrece MATLAB no son muy amplias, en comparación con otras aplicaciones de Windows como Visual Basic, Visual C. La elaboración de *GUIs* puede llevarse a cabo de dos formas, la primera de ellas consiste en escribir un programa que genere la *GUI* (script), y la segunda opción consiste en utilizar la herramienta de diseño de *GUIs*, incluida en *Matlab*®, llamada **GUIDE** [2].

En el presente proyecto final de carrera, se opta por la segunda opción para desarrollar la *GUI*. Por lo tanto, en este caso la elaboración de la interfaz gráfica comienza con la ejecución del comando *guide* desde la línea de comandos de *Matlab*®, que al ser invocado despliega la siguiente ventana:

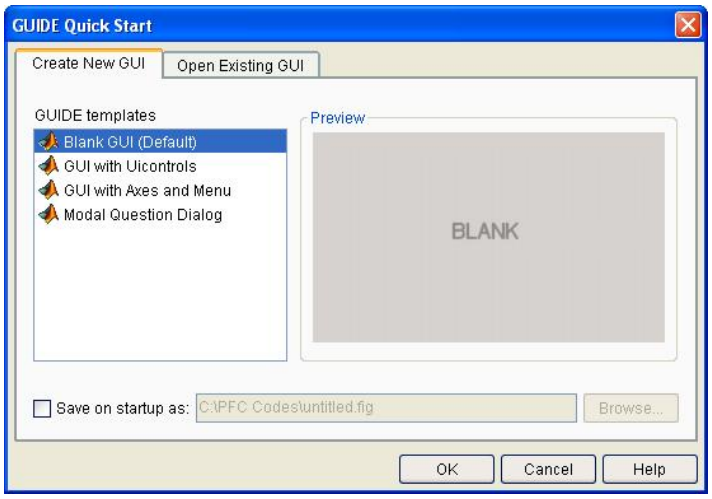


Figura 3.1: Ventana *GUIDE Quick Start*

Por medio de la ventana que se presenta en la figura 3.1, se puede seleccionar entre una *GUI* nueva o una ya existente. En este caso se selecciona la *GUI* de *default*, que aparece en blanco. Al seleccionarla aparece la ventana de la figura 3.5, en la cual se tienen los diferentes tipos de posibles botones, así como los menús necesarios para nombrarlos, editarlos en tamaño, color, alineación, etc. y definir los *callbacks* asociados para cada uno de ellos.

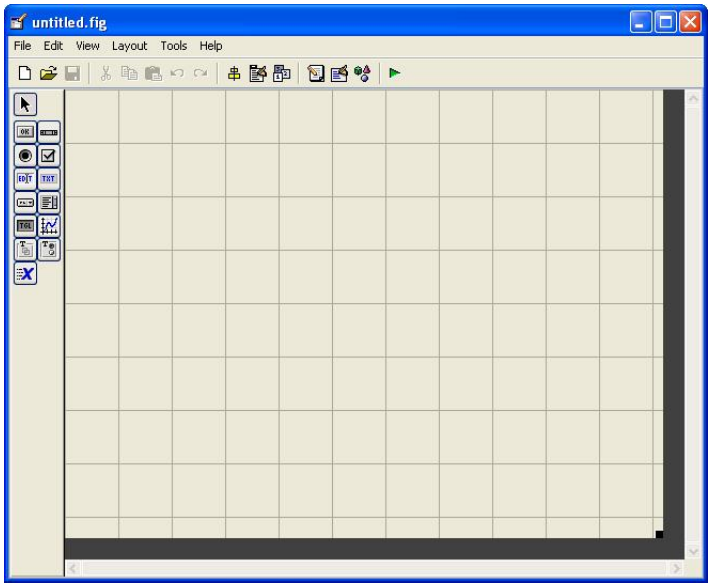


Figura 3.2: Ventana de creación de la Interfaz Gráfica de Usuario

La interfaz gráfica de usuario se diseña de la forma más sencilla posible, que permita al usuario final de la misma, hacer uso de ésta de forma intuitiva, sin necesidad de poseer conocimientos avanzados en *Matlab*.

Una vez se concluyen las fases de diseño gráfico e implementación, se salva la interfaz gráfica con el nombre de *GUI.fig*. Al guardarla, *Matlab*® genera automáticamente un código denominado *GUI.m* con las características de dicha figura, presentando esta el siguiente aspecto:

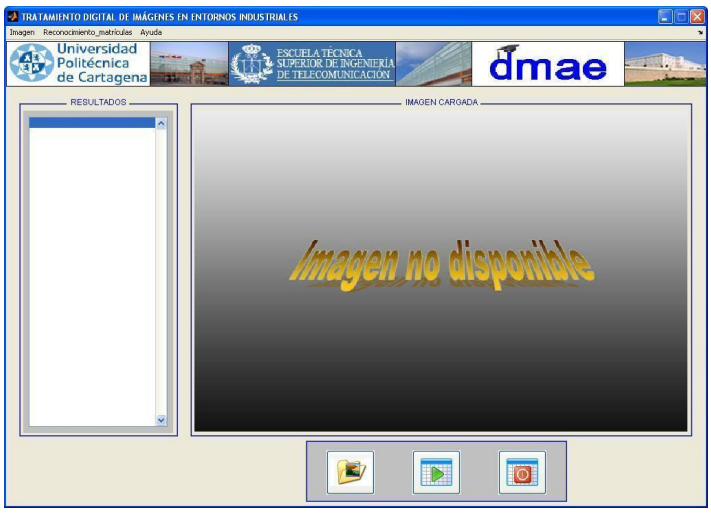


Figura 3.3: Interfaz Gráfica de Usuario (*G.U.I.*)

3.3. Información que debe conocer el usuario antes de iniciar la interfaz gráfica

3.3.1. Ubicación de los archivos y versiones de Matlab

Con la documentación del presente proyecto final de carrera se entrega de forma adjunta un CD-ROM de datos que contiene todo lo necesario para que cualquier usuario pueda ejecutar la interfaz gráfica "*GUI*". Lo único que se necesita para ello, es el programa informático *Matlab* en la versión:

- R14 (*Matlab® 7.0*) [8]

Para versiones anteriores no se garantiza el correcto funcionamiento del programa, debido a que para la creación de la interfaz gráfica se han utilizado librerías que podrían no estar presentes.

3.3.2. Organización del CD-ROM de datos

La organización del CD-ROM de datos, se muestra en la siguiente estructura en árbol con los directorios del mismo:

- Directorio de nombre *Memoria PFC*: En este directorio se ubica el conjunto de códigos desarrollados en *LaTeX*, asi como el documento final generado en formato *'*.pdf'* con la memoria del proyecto, *PFC.pdf*.

- Directorio de nombre *PFC Codes*: El contenido de los archivos y subdirectorios que nos encontramos al acceder a la carpeta indicada son los siguientes:

  a) Varios *ficheros Matlab*(extensión *.m*) con los códigos de los programas y funciones implementadas. El usuario debe tener constancia que con la modificación de cualquiera de los archivos especificados, se corre el riesgo de que la aplicación no funcione correctamente y presente errores inesperados.

  b) Directorio de nombre *GUI images*: Contiene todas las imágenes que se utilizan para presentar la interfaz gráfica de usuario, así como todas las figuras necesarias para presentar algunos resultados en la interfaz gráfica, cuando se ejecutan ciertos algoritmos concretos.

  c) Directorio de nombre *Apps images*: Contiene todas las imágenes que se han utilizado para estudiar y analizar el conjunto de algoritmos implementados, mediante el desarrollo del presente proyecto final de carrera.

  d) Directorio de nombre *BBDD*: Contiene un registro a modo de base de datos, con los nombres y apellidos de las personas autorizadas, y las matrículas que tienen asociadas dichos usuarios, en un aplicativo implementado para el control de accesos realizado a partir de la lectura de las matrículas de vehículos.

El usuario debe copiar el directorio(*PFC Codes)* en un directorio local seleccionado por el usuario. Por ejemplo, el usuario podría copiar este directorio dentro de la raíz de la unidad '*Disco local (C:)*' de su ordenador.

3.3.3. Localización de las carpetas desde el Current Directory de Matlab

Para poder ejecutar la interfaz gráfica de la aplicación, el usuario debe seleccionar desde *Matlab* como directorio principal, el directorio de nombre *PFC Codes*. Por ejemplo, siguiendo la recomendación de copiar este directorio en la unidad '*Disco local (C:)* de *Windows*, debería aparecer el Current Directory de *Matlab*, tal y como se presenta en la sección destacada de la figura 3.4.

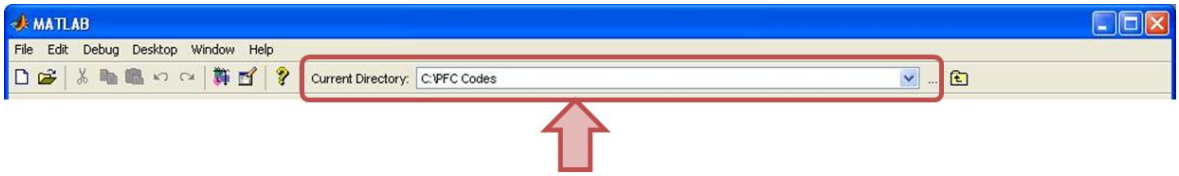


Figura 3.4: Directorio Principal de *Matlab* para ejecutar la *GUI*

3.4. Manual de usuario

La ejecución de la aplicación se inicia introduciendo en la línea de comandos de *Matlab* el nombre del programa principal, *GUI*, con ello se despliega la ventana principal de la aplicación(figura 3.5).

La ventana principal se puede dividir en cuatro partes diferenciadas, tal y como se puede observar en la figura 3.5. El recuadro de color rojo se corresponde con la barra de menús de la aplicación, la parte recuadrada en color naranja representa la sección de la aplicación donde se muestra la imagen que hay cargada en la misma para su procesamiento, en color verde podemos advertir el área donde se presentan los resultados e información adicional de las acciones llevadas a cabo mediante la interfaz gráfica, y en la parte inferior (color amarillo) tenemos los controles de tipo botón.

En el próximo apartado de la presente memoria, se comenta por separado de forma detallada, la utilidad y funcionalidades de cada una de las partes principales que conforman la interfaz gráfica de la aplicación. Sin embargo, antes de entrar en detalle con todas estas características, se estima apropiado indicar el procedimiento general que se suele seguir para hacer uso de los algoritmos implementados en la *GUI*. Para ello se debe adoptar el siguiente proceso:

a) Cargar la imagen a tratar, mediante el botón *Abrir imagen*.

b) Seleccionar una opción de la barra de menús.

c) Pulsar el botón *Ejecutar algoritmo*.

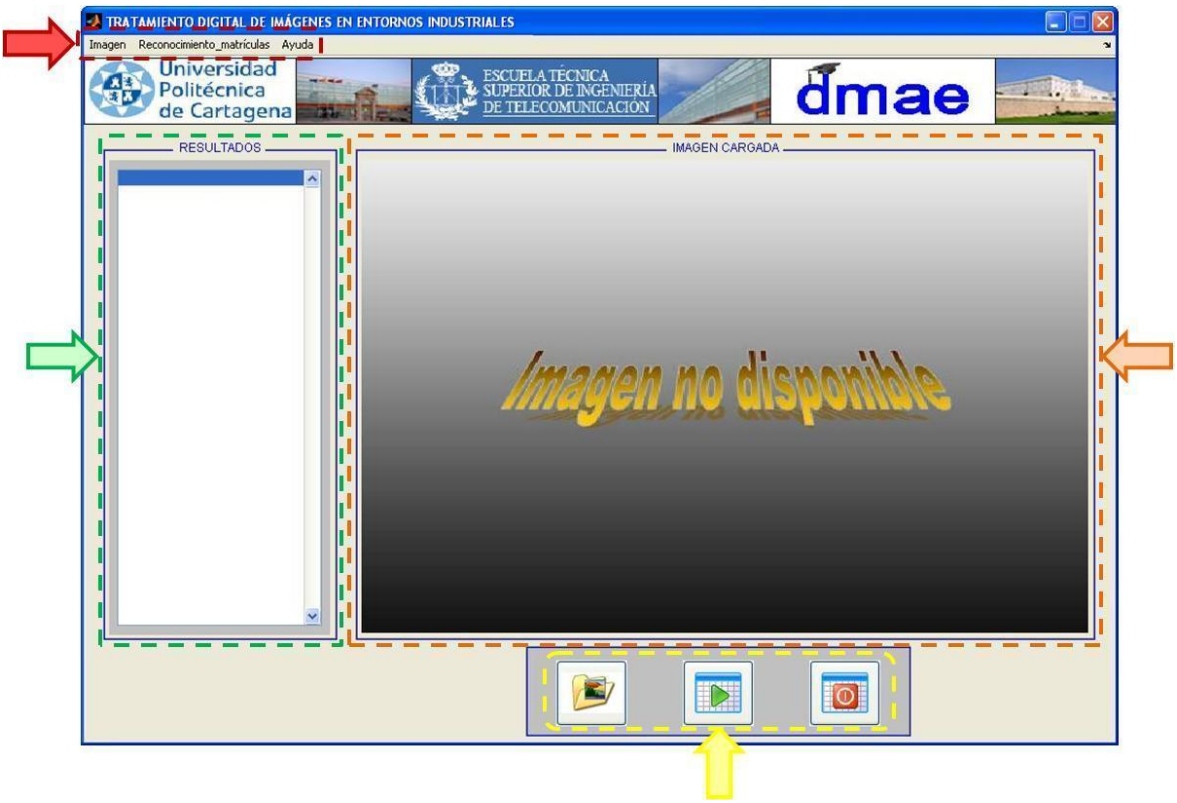


Figura 3.5: Ventana principal de la aplicación

3.4.1. Principales componentes de la interfaz gráfica

3.4.1.1. Barra de menús

El desarrollo de una interfaz gráfica para una aplicación, suele considerar con mucha frecuencia la inclusión de una barra de menús. Al ser una abstracción tan útil, se ha adoptado casi de forma universal en el diseño de interfaces gráficas, fundamentalmente porque permite que el usuario ejecute comandos sin tenerlos que recordar de memoria, evitando la posibilidad de que se equivoque en la sintaxis del comando, y además se pueden deshabilitar ciertas entradas en los menús, en el caso de que en el estado que se encuentre la aplicación no tengan cabida (sentido) la ejecución de dichos comandos.

Por las razones comentadas, y con objeto de facilitar el uso de todos los algoritmos implementados sin necesidad de conocimientos avanzados de *Matlab*, la interfaz gráfica dispone de una barra de menús (área recuadrada de color rojo en la figura 3.5), desde la que se pueden seleccionar las diferentes opciones que albergan los grupos de menús principales (3), como

son *Imagen*, *Reconocimiento_matrículas*, y *Ayuda*.

Menú Imagen

Al acceder a este menú se despliegan los submenús presentados en la siguiente figura:

Figura 3.6: Menú Imagen

Acerca de estos submenús, a grandes rasgos se podría indicar que su selección permitiría realizar dos tipos de acciones diferenciadas. Por un lado, se puede abandonar la aplicación, parando los algoritmos en curso y cerrando todos los componentes y figuras que se pudieran haber desplegado mediante el uso de la misma, todo ello mediante la opción de menú presentada en la imagen 3.7 (menú *Salir*).

Figura 3.7: Menú Imagen - Salir

Por otro lado, mediante las otras opciones del menú *Imagen* que se pueden ver en la figura 3.8, se pueden seleccionar diferentes tipos de aplicaciones, cuyas implementaciones se basan en los diferentes algoritmos y métodos de procesamiento de imágenes digitales objeto del presente proyecto final de carrera.

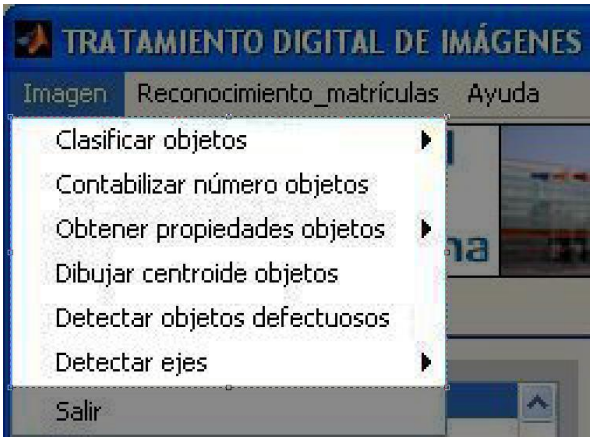


Figura 3.8: Aplicaciones del menú Imagen

A su vez, atendiendo a la similitud del tipo de proceso desarrollado en la implementación de ciertos algoritmos, se realizan algunas agrupaciones por medio de tres submenús adicionales, tal y como se presenta en las siguientes figuras:

Figura 3.9: Menú Imagen - Clasificar objetos

Figura 3.10: Menú Imagen - Obtener propiedades objetos

Figura 3.11: Menú Imagen - Detectar ejes

Menú Reconocimiento_matrículas

Desde este menú se pueden seleccionar los algoritmos identificados con las aplicaciones implementadas en relación a las tareas de reconocimiento y tratamiento de carácteres alfanuméricos de matrículas de vehículos, que permiten desarrollar un completo sistema de control de accesos.

Figura 3.12: Menú Reconocimiento_matrículas

Menú Ayuda

Al seleccionar el usuario desde el menú *Ayuda*, la opción *Acerca de...*, se presenta una nueva ventana donde se informa de la versión de *Matlab* recomendada para hacer uso de la aplicación, así como una breve descripción del proceso a seguir para poder utilizar la interfaz gráfica, y practicar con las distintas aplicaciones implementadas, obteniendo conclusiones acerca de los resultados que presentan las mismas.

Figura 3.13: Menú Ayuda - Acerca de...

Figura 3.14: Información menú Ayuda

3.4.1.2. Sección para representación de imágenes

En el área indicada en la imagen que se presentaba con anterioridad (figura 3.5) mediante el recuadro de color naranja, se puebe observar de forma clara el área al que se hace referencia.

En la sección que estamos abordando, se muestra la imagen sobre la que se van a aplicar los algoritmos seleccionados mediante los menús correspondientes, es decir, la imagen sobre la que se va a realizar el tratamiento correspondiente una vez se pulse el botón de *ejecutar algoritmo*.

También se hace interesante advertir, que cuando se despliega la interfaz gráfica, inicialmente aparece una imagen cargada por defecto, donde se indica mediante el texto '*Imagen no disponible*', que no hay preparada ninguna imagen válida para ejecutar método de procesamiento alguno (ver figura 3.15), y que para ello previamente se debería desplegar (cargar) una imagen.

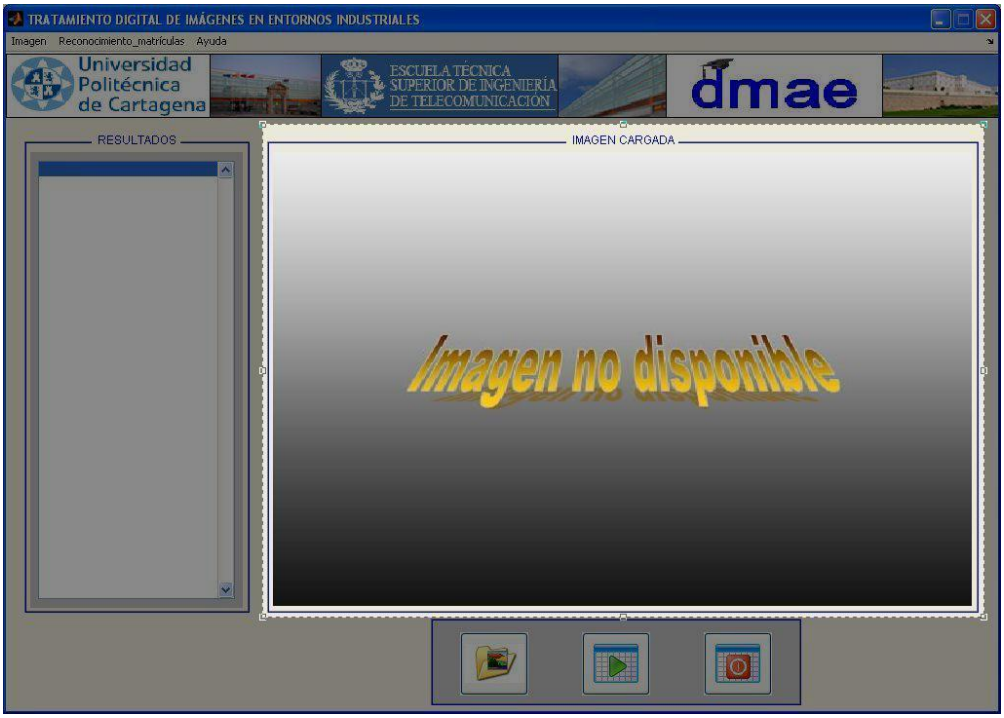


Figura 3.15: Área de representación por defecto (sin imagen cargada)

Figura 3.16: Ejemplo de área de representación con imagen cargada

3.4.1.3. Área de información y resultados

El área al que se va a hacer referencia, se puede visualizar con claridad en la figura 3.5, por medio del recuadro en color verde.

Esta sección de la GUI, va a ser de gran utilidad para el usuario al mostrar diferentes tipos de información, como por ejemplo:

Opción de menú seleccionada: Cada vez que el usuario seleccione cualquier opción de la barra de menús, queda recogida esta acción en el área de resultados, con la facilidad que supone al usuario tener siempre constancia de la aplicación seleccionada, antes de pulsar el botón de *ejecutar algoritmo*.

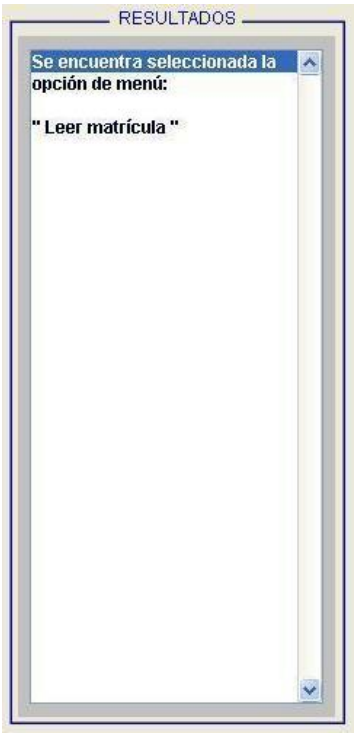


Figura 3.17: Ejemplo de información sobre la opción de menú seleccionada

Resumen de resultados del algoritmo ejecutado: Cuando se da por finalizada la ejecución de un algoritmo, en este área se muestra información acerca de la correcta finalización del mismo (exitosa o no satisfactoria si ha habido alguna incidencia), y en caso de que el algoritmo ejecutado devuelva algún tipo de resultado textual y/o numérico, esta información queda representada de forma resumida en dicho apartado.

Se debe indicar, que cuando se da la situación en la que la información a mostrar en el área de resultados, es de mayor tamaño (ancho y/o alto) que la dimensión del área de presentación de los resultados, para que el usuario pueda visualizar toda la información, de forma automática se agregan una o varias barras de desplazamiento (hotizontales y/o

verticales) según la necesidad, tal y como se puede observar en la figura 3.19.

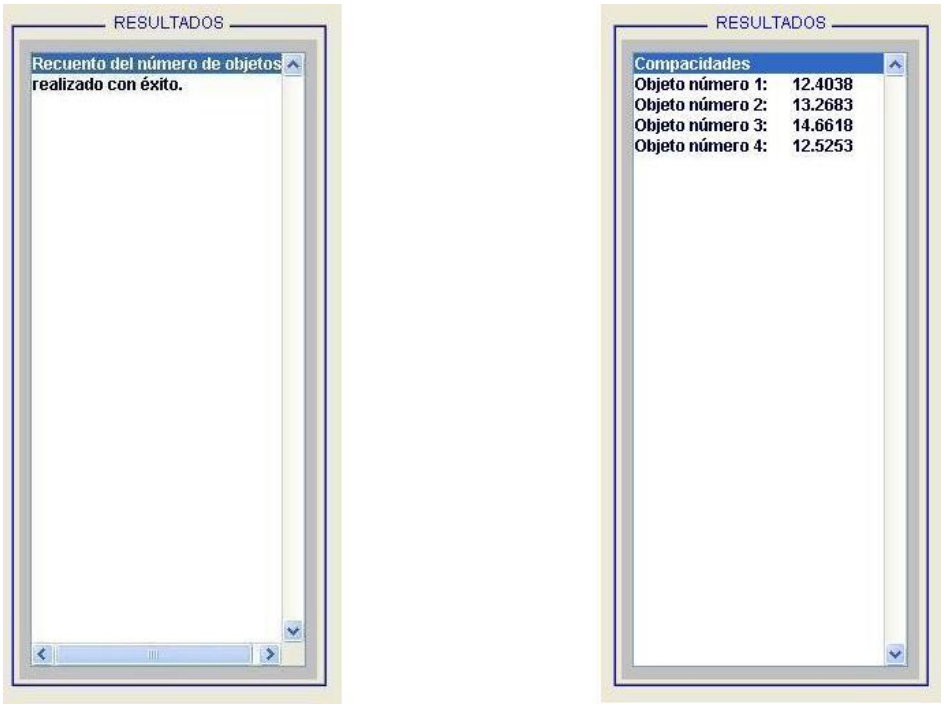


Figura 3.18: Ejemplo de diversos tipos de información en el área de resultados

Figura 3.19: Ejemplo barra de desplazamiento horizontal en el área de resultados

3.4.1.4. Conjunto de controles tipo botón

Por medio de la figura 3.5 se puede observar con claridad, mediante el recuadro de color amarillo, la sección de la interfaz gráfica donde se agrupan el conjunto de botones principales (3) de la aplicación.

A continuación se indica de forma detallada, la funcionalidad de cada uno de los botones:

- **Botón *Abrir Imagen***: Este botón se encuentra situado en la parte izquierda del área que los agrupa, y cuando el usuario sitúa el cursor del ratón sobre este elemento gráfico, se muestra un texto emergente (*Abrir Imagen*) que informa al usuario de la funcionalidad de dicho elemento, tal y como se puede ver en la siguiente figura:

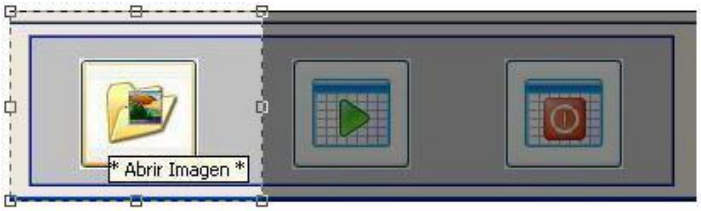


Figura 3.20: Botón *Abrir imagen*

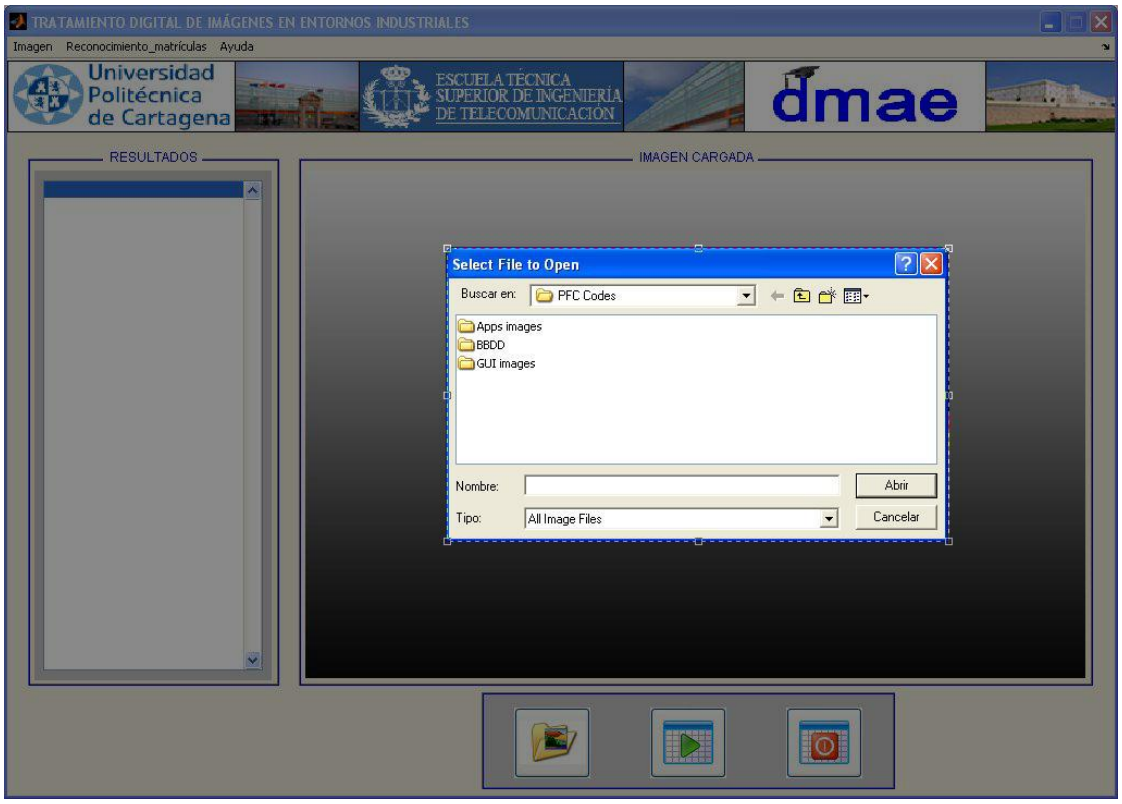


Figura 3.21: Ventana desplegada (por defecto) al pulsar el botón *Abrir imagen*

La funcionalidad de este botón consiste en permitir al usuario abrir cualquier imagen que tenga almacenada en su equipo (o un medio extraíble), pasando el archivo de imagen seleccionado a formar parte de la aplicación, mediante la carga de esta en el área correspondiente para la representación de la misma. Por lo tanto, este botón se debe utilizar como mínimo una vez, antes de poder ejecutar los algoritmos de procesamiento accesibles desde los menús de la *GUI*.

Mediante este botón sólo se permiten seleccionar archivos con formato de imagen, es decir, únicamente se pueden seleccionar y desplegar los tipos de formatos de imágenes más extendidos como son, **.jpeg*, **.jpg*, **.tiff*, **.gif*, **.bmp*, **.png*, **.hdf*, **.pcx*, **.xwd*, **.ico*, **.cur*, **.ras*, **.pbm*, **.pgm*, y **.ppm*.

- **Botón *Ejecutar Algoritmo***: Este control se encuentra ubicado en la posición central del conjunto de elementos de este tipo, y cuando el usuario sitúa el cursor del ratón sobre este botón, se muestra un texto emergente (*Ejecutar Algoritmo*) que informa al usuario de la funcionalidad de dicho elemento, tal y como se puede ver en la figura 3.22.

Figura 3.22: Botón *Ejecutar Algoritmo*

La acción que se despliega cuando el usuario pulsa sobre este botón, es la inmediata ejecución de la aplicación que previamente debe haber seleccionado el usuario por medio de la barra de menús.

- **Botón *Cerrar Ventana***: Este botón se encuentra ubicado en la parte derecha del área que agrupa este tipo de controles, y cuando el usuario sitúa el cursor del ratón sobre dicho componente, se muestra un texto emergente (*Cerrar Ventana*) que informa al usuario de la funcionalidad del dicho elemento, tal y como se puede ver en la figura 3.23.

Este botón implementa la funcionalidad de cerrar todos los elementos de la aplicación, es decir, cuando el usuario selecciona este botón, se finaliza la ejecución de la interfaz gráfica principal, así como todas aquellas ventanas y figuras que se encuentren desplegadas debido a la ejecución de otros algoritmos con anterioridad a la selección de este botón. Además, es relevante informar al lector, de que esta funcionalidad se

encuentra duplicada en la interfaz gráfica de la aplicación, ya que también se encuentra accesible desde el menú *Imagen*, opción *Salir*, tal y como se presentaba en la figura 3.7.

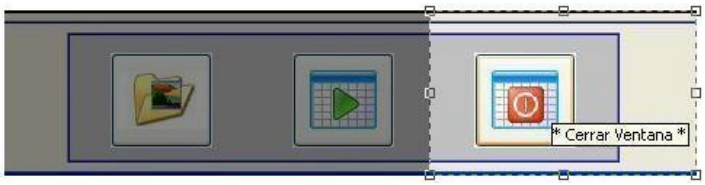


Figura 3.23: Botón *Cerrar Ventana*

3.4.1.5. Ventanas de diálogo

Con objeto de facilitar el manejo de la aplicación, el diseño e implementación de la misma se fundamenta en intentar lograr una apariencia intuitiva, que permita el manejo de la misma de la forma lo más guiada posible. Para ello, con frecuencia se mantiene informado al usuario de las acciones de relevancia llevadas a cabo, así como se presentan diversos tipos de advertencias y errores que se hayan podido producir, mediante distintos modelos de ventanas emergentes.

A continuación se comentan los diversos tipos de ventanas de diálogo que se han implementado para que se muestren en primera instancia, y permitan a su vez interactuar con el usuario de la aplicación.

- **Ventanas de ayuda**: Este tipo de ventanas se utilizan principalmente para presentar al usuario información de resultados, tanto parciales como finales, que se van generando mediante el proceso de ejecución del algoritmo en cuestión. Este tipo de ventanas, requieren interactuar con el usuario, ya que se necesita que éste pulse sobre el botón *OK* para que pueda continuar el proceso de ejecución del algoritmo, o se de por finalizado el mismo(ver figura 3.24).

- **Ventanas de advertencia**: Al igual que las ventanas de ayuda, estas ventanas son también modales con lo que necesitan que el usuario pulse el botón *OK*, para poder volver a interactuar con la aplicación, sin embargo, cuando aparecen este tipo de ventanas requieren que el usuario realice alguna actuación adicional (no sólo pulsar *OK*), ya que el algoritmo no va a continuar su ejecución hasta que se realice dicha tarea sugerida por la ventana, o no van a lograr obtener los resultados esperados.

  A modo de ejemplo, se pueden observar por medio de la figura 3.25, algunas ventanas implementadas de este tipo.

Figura 3.24: Ejemplos de ventanas de ayuda desarrolladas

Figura 3.25: Ejemplos de ventanas de advertencia desarrolladas

- **Ventanas de error**: Al igual que los otros tipos de ventanas de diálogo se requiere que el usuario pulse el botón *OK*, pero en este caso se advierte a éste de manera inequívoca que se ha producido algún error en la tarea que intentaba llevar a cabo. Por lo tanto, este tipo de ventanas informa al usuario del error incurrido para que proceda de la forma adecuada.

  Este tipo de ventanas se pueden visualizar por ejemplo, cuando se intenta ejecutar cualquier algoritmo desde las opciones de menú, pero no se ha procedido previamente a cargar una imagen en la interfaz gráfica sobre la que realizar el procesamiento. En este caso, la ventana de diálogo advierte del error cometido, para que el usuario pueda subsanarlo.

Figura 3.26: Ejemplo de ventana de error implementada

3.4.2. Otras funcionalidades de la GUI

En esta sección se van a documentar algunas funcionalidades de interés que se han implementado, y que no han sido comentadas con anterioridad.

Menús chequeables

Esta opción permite al usuario acceder desde la barra de menús, y visualizar el menú que tiene chequeado, es decir, que siempre (por ejemplo antes pulsar el botón *ejecutar algoritmo*) se puede acceder a la barra de menús para comprobar qué opción de menú se tiene seleccionada en ese momento.

Figura 3.27: Ejemplo de menú seleccionado

Notar también que esta funcionalidad se complementa con la funcionalidad del área de información y resultados, donde se muestra en modo texto la opción de menú chequeada.

Cierre de ventanas y figuras

La ejecución de cualquier algoritmo implica que en primera instancia, y siempre que sea posible, se vayan a tomar las distintas áreas de la ventana principal de la interfaz gráfica, para presentar el proceso y/o los resultados de la ejecución del algoritmo seleccionado. Sin embargo, cuando esto no sea posible dado el alto volumen de información que genera el algoritmo, se utilizan tantas ventanas y figuras auxiliares como sea necesario.

Atendiendo a la premisa anterior y para hacer tanto más rápidas, como menos engorrosas, las pruebas de las diferentes opciones de menú desarrolladas, se ha implementado de forma automática el cerrado de todas y cada una de las ventanas o figuras que difieran de la ventana principal. Con ello, el usuario sólo tiene que preocuparse de ejecutar los algoritmos que desee, ya que en el momento que decida ejecutar un nuevo método, se finalizan automáticamente todos los resultados obtenidos anteriormente, evitando así posibles errores en cuanto a la interpretación de resultados provenientes de diferentes aplicaciones ejecutadas.

Capítulo 4

Experimentos y aplicaciones desarrolladas

Una vez expuesta la parte teórica que envuelve el *procesamiento de imágenes digitales*, habiendo descrito las principales operaciones morfológicas [11], así como las diversas técnicas de segmentación [16] que se pueden aplicar a una imagen digital, y tras haber presentado a su vez, la documentación correspondiente para permitir conocer el funcionamiento de la interfaz gráfica principal de la aplicación, estamos en disposición de descubrir y poder analizar todos los módulos desarrollados para el tratamiento digital de imágenes en entornos industriales.

Por medio de este capítulo se ilustra al lector acerca de cada uno de los módulos desarrollados, habiéndose implementado éstos a partir de la teoría y algoritmos expuestos en el capítulo 2, con objeto de destacar la importancia y las diversas aplicaciones que se pueden construir, a partir de un requerimiento concreto y la aplicación de las técnicas de procesamiento de imágenes digitales adecuadas.

4.1. Etapas del sistema de procesamiento

A partir de la figura 4.1 se pueden distinguir las pautas de procesamiento, que se han empleado como orientación en las fases de desarrollo aplicadas para todos y cada uno de los módulos implementados, y que se corresponden con las etapas típicas de cualquier sistema de procesamiento de imágenes digitales, con la particularidad de que las 3 primeras etapas se han implementado de forma que guarden un alto grado de similitud en todas las aplicaciones desarrolladas, mientras que las 2 etapas siguientes se implementan de forma que cumplan los requisitos de mayor especificación para el aplicativo en cuestión.

A partir de ahora, a las tres primeras etapas las denominaremos como *etapas comunes*, las cuáles se van a caracterizar gráfica (mediante diagrama de bloques) y textualmente de la siguiente forma:

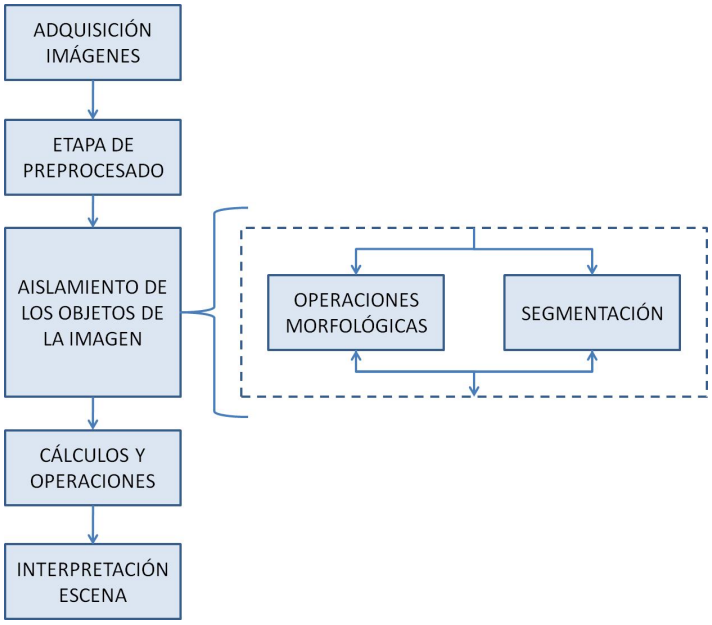


Figura 4.1: Diagrama de bloques etapas de procesamiento

- **Etapa de adquisición de imágenes**. Esta etapa consiste básicamente en leer la imagen que hay cargada en la aplicación cuando el usuario procede a ejecutar cualquier método de la interfaz gráfica. Esta lectura de la imagen cargada se realiza de forma que si dicha imagen se encuentra en escala de grises se traduce la misma a una matriz bidimensional mediante la que poder continuar operando en la siguiente etapa, y en caso de que la imagen sea RGB entonces se trabaja con un arreglo tridimensional.

- **Etapa de preprocesado**. Este preprocesado se basa en la binarización y obtención del negativo de la imagen leída en la etapa anterior. Con la binarización se convierte a escala de grises la imagen leída, y a continuación se transforma esta imagen a binario con una *umbralización* (*thresholding*), con lo que ésta pasa a estar compuesta de valores '*1*' (negro) para todos los píxeles en la imagen de entrada con menor luminancia y '*0*' (blanco) para todos los demás píxeles (indicados a partir de un parámetro de nivel).
  La obtención del negativo (complementación), estriba en invertir el valor de los píxeles de la imagen binarizada, con lo que ahora los blancos se convierten en negros y los negros a blancos, con lo que se destaca el fondo de la imagen (en color negro) de los componentes principales de la misma (color blanco). Esto se consigue mediante la utilización de la operación lógica *NOT*.

- **Etapa de aislamiento de los objetos de la imagen**. Al finalizar esta etapa, tenemos los elementos necesarios para acceder a cada uno de los componentes principales de la imagen, distinguiéndolos del fondo de la misma. Por lo tanto, esta etapa se va a caracterizar por la identificación y etiquetación [27] de los componentes de interés, de la imagen obtenida a partir de la etapa anterior.
  Para alcanzar de forma satisfactoria los objetivos de esta etapa, se hacen necesarias

la aplicación de diversos tipos de operaciones morfológicas y de segmentación, de las presentadas en las secciones correspondientes del capítulo 2. Cabe destacar que el tipo de operaciones de segmentación implementadas en cada módulo, comprenden principalmente operaciones de *segmentación basada en regiones* utilizando para ello *conectividad - 8*, sin embargo, el tipo de operaciones morfológicas aplicadas no tiene la posibilidad de implementarse de forma genérica, sino que este tipo de operaciones dependen de forma específica del módulo que se encuentre implementando, por lo que en los próximos apartados se documenta el tipo de operaciones morfológicas desarrolladas en cada algoritmo.

A modo de ejemplo, en la próxima figura se pueden adivinar los resultados obtenidos a partir de la aplicación de las etapas comunes descritas para una imagen RGB dada.

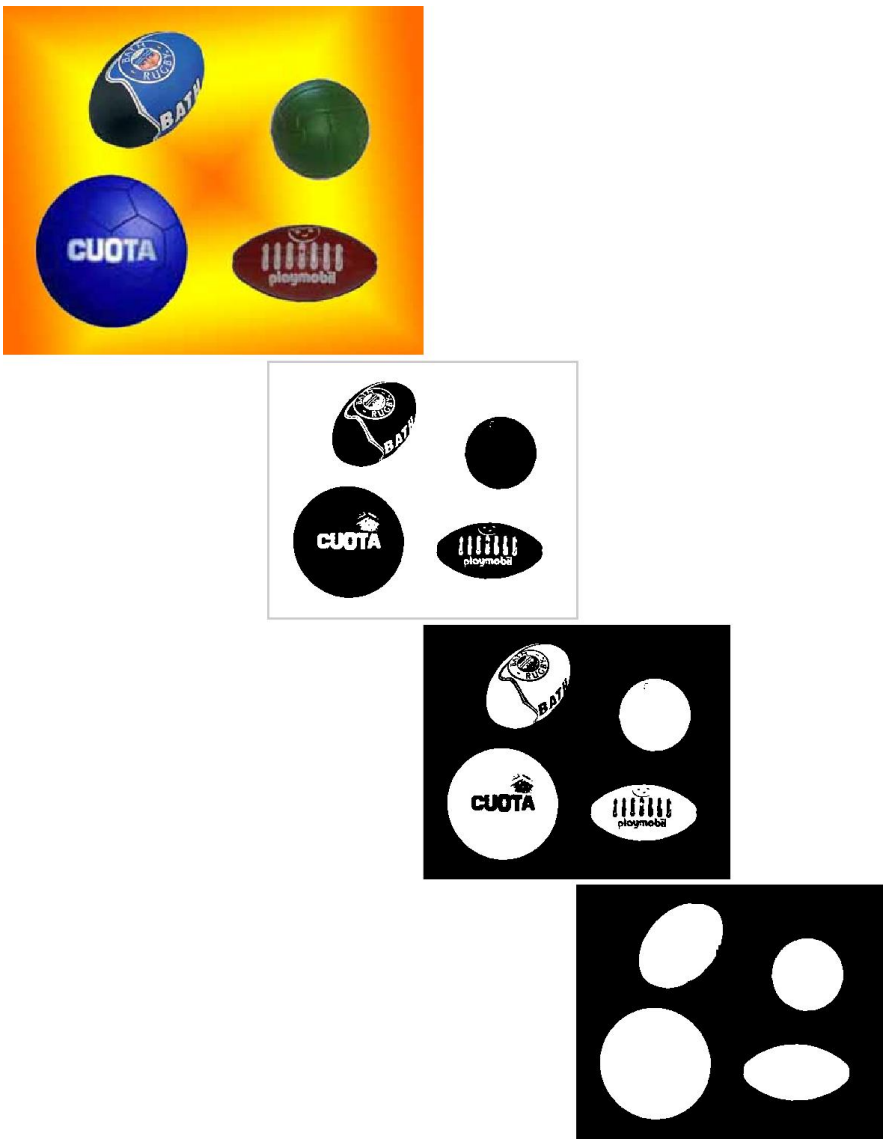


Figura 4.2: Ejemplo de procesamiento de etapas comunes en una imagen RGB

4.2. Extracción de propiedades

Este módulo se encarga de obtener y presentar la propiedad seleccionada por el usuario mediante la interfaz gráfica de la aplicación, para cada uno de los principales objetos que componen la imagen que se encuentre desplegada en la *GUI*.

Las diversas opciones de menú en relación al módulo presentado, permiten seleccionar diversas propiedades tales como, **Área**, **Perímetro**, **Excentricidad**, **Compacidad** y **Centroide**.

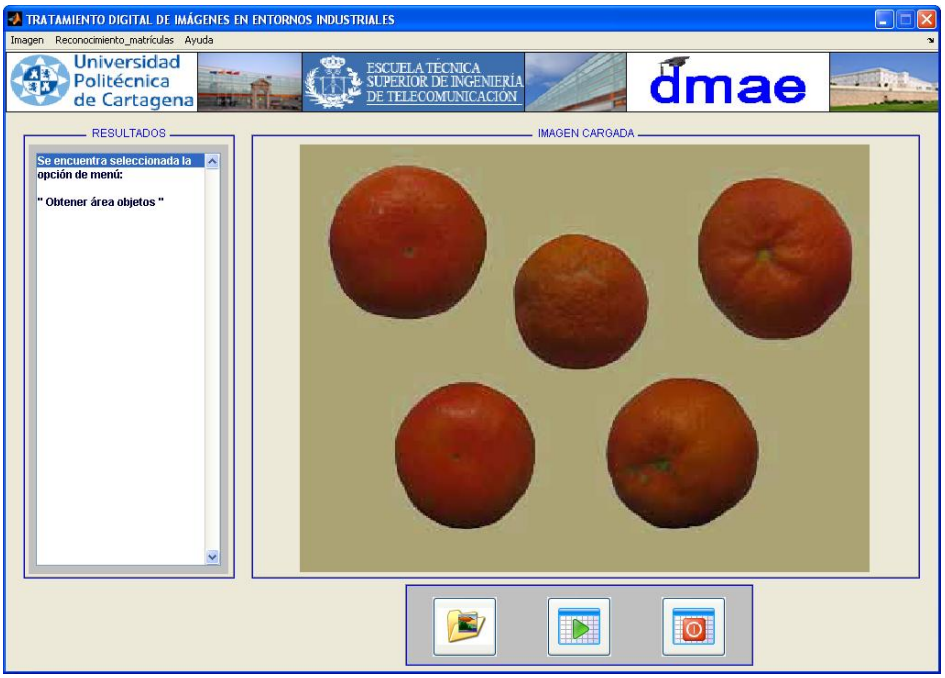


Figura 4.3: Estado GUI previo a la obtención del *área* de los objetos

De forma semejante al resto de módulos implementados, se comienza el procesamiento ejecutando las *etapas comunes*, con la particularidad de que en la tercera etapa, se deben aplicar operaciones morfológicas de dilatación y erosión (en este orden), previas al proceso de etiquetación, para que éste produzca los resultados esperados.

Se debe hacer hincapié en el orden de aplicación de las operaciones morfológicas descritas, al ser de especial relevancia, debido a que en caso de emplearlas en orden inverso al indicado, se incurren en posteriores errores a la hora de proceder con la identificación de las principales componentes de la imagen. En la sección *2.3* del capítulo 2 se puede ampliar información acerca de este tipo de operaciones, así como de otras operaciones morfológicas de relevancia.

En la etapa de *cálculos y operaciones*, nos encargamos de obtener las propiedades de las regiones definidas mediante el proceso de segmentación de regiones llevado a cabo, y

que permite trabajar con las regiones obtenidas. Por lo que, se procede con el recorrido de todas las regiones identificadas, y se aplican las operaciones correspondientes que permiten obtener la propiedad solicitada por el usuario, según se presentaba en la sección del capítulo 2 enunciada como *Descriptores basados en regiones*.

Con la finalización de la última fase (*interpretación de escena*), se facilita al usuario mediante el apartado correspondiente de la *GUI* principal, un resumen con la información de todas las propiedades obtenidas de los todos los objetos de la imagen.

A su vez, cabe reseñar que durante el proceso de ejecución de este algoritmo, se van mostrando diferentes ventanas de diálogo, donde se presenta la información de la propiedad obtenida para el objeto en cuestión. Cada una de estas ventanas requiere la interacción con el usuario (pulsar botón *OK* de la ventana o tecla *intro* del teclado) para proseguir con la ejecución de la función.

También es interesante destacar, que la propiedad seleccionada por el usuario, determina la forma en la que se van presentando los resultados parciales durante el proceso de ejecución del algoritmo, ya que se va a ir seleccionando para cada región que se encuentre analizando, la propiedad seleccionada, es decir, si el usuario selecciona la propiedad área, se va mostrando de forma gráfica(y textual) el área de cada objeto de la imagen. De igual forma se procede con el perímetro, y con el resto de propiedades.

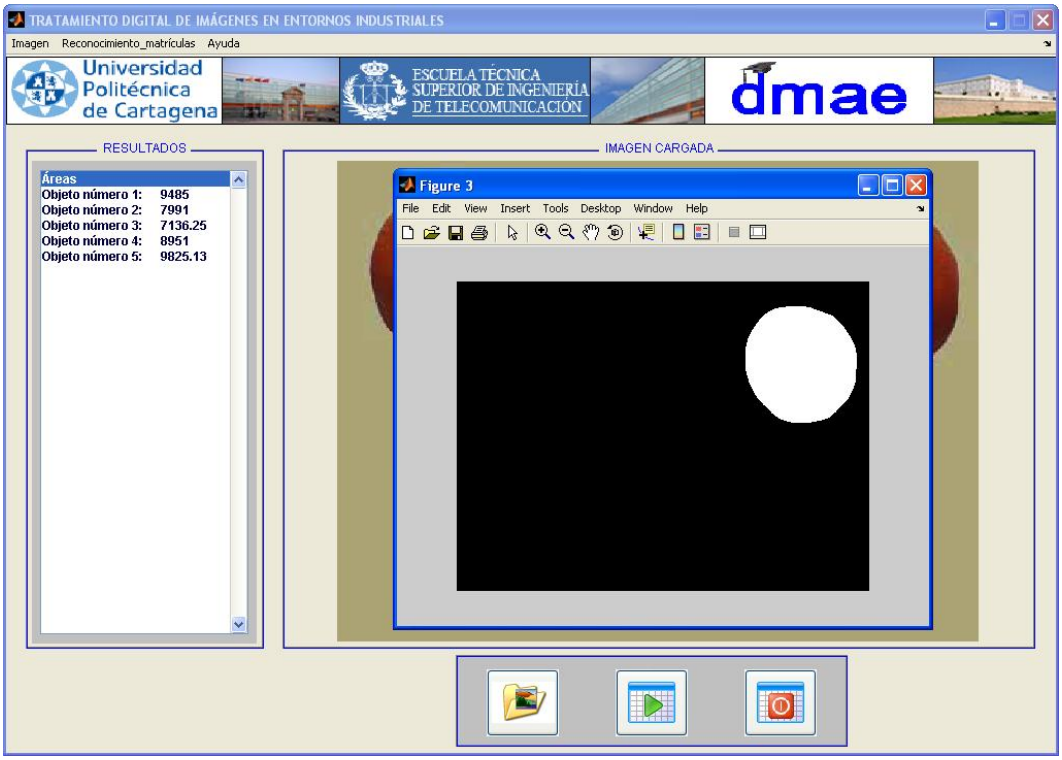


Figura 4.4: Estado GUI tras la obtención del *área* de los objetos

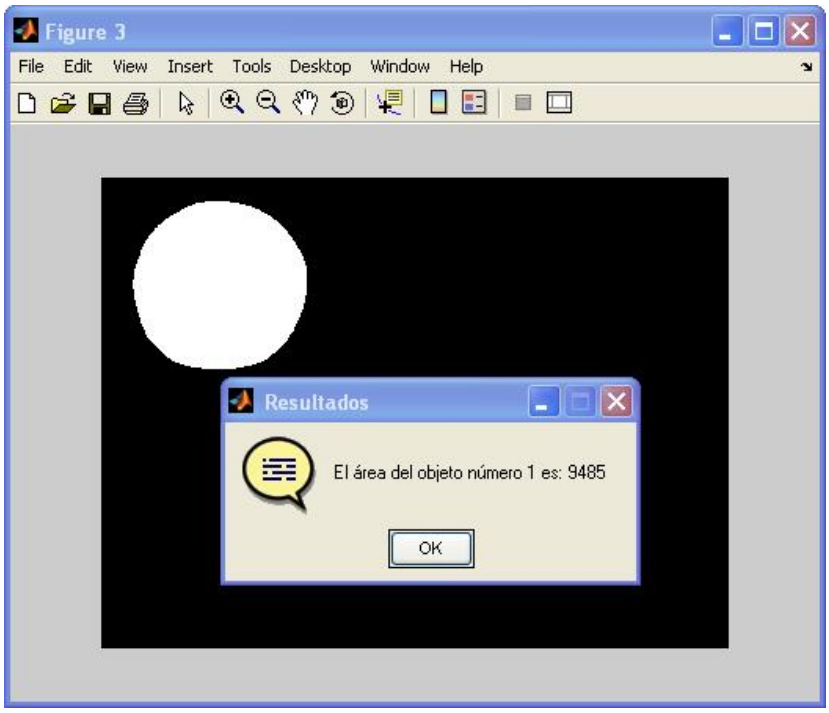


Figura 4.5: Ejemplo ventana de resultados parciales para el cálculo de *áreas*

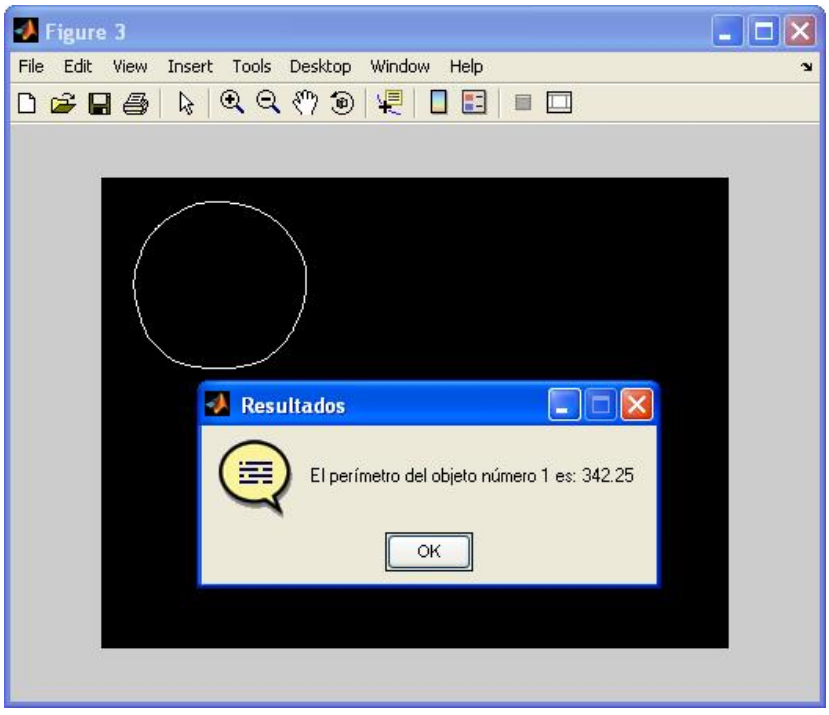


Figura 4.6: Ejemplo ventana de resultados parciales para el cálculo de *perímetros*

4.3. Clasificación de objetos

La funcionalidad principal que se compromete mediante este módulo, es la de efectuar una labor de clasificación en dos clases principales, atendiendo como elemento de clasificación la propiedad seleccionada por el usuario mediante la interfaz gráfica de la aplicación.

A partir de la propiedad seleccionada por el usuario, se define un *sesgo* que permite identificar los objetos de la imagen en una u otra clase, para que una vez se calcule el valor de la propiedad seleccionada para cada elemento, pueda ser comparada con el valor del sesgo en cuestión.

Atendiendo a las posibles opciones de clasificación que ofrece la interfaz gráfica de la aplicación, se han definido los siguientes sesgos:

- Clasificación de objetos por área: El sesgo que se utiliza para esta clasificación, se obtiene como la media aritmética de las áreas de todos los objetos de la imagen.
- Clasificación de objetos por perímetro: En este caso, el sesgo que se implementa para esta clasificación, se obtiene como la media aritmética de los perímetros de todos los componentes de interés en la imagen.
- Clasificación de objetos por excentricidad: Para este tipo de clasificación, la determinación del sesgo no requiere de cálculo alguno, ya que se escoge para éste un valor fijo de $0,5$, dado que esta cifra es el valor medio de los posibles datos de excentricidad que se pueden obtener de los objetos de la imagen, al estar la excentricidad definida en el intervalo $[0, 1]$.
- Clasificación de objetos por compacidad: El sesgo que se implementa para este tipo de clasificación, se obtiene como la media aritmética de los valores de compacidad que se calculan a partir del conjunto de objetos pertenecientes a la imagen a procesar.

Al igual que en el resto de módulos desarrollados, las tareas de procesamiento comienzan con la ejecución de las *etapas comunes*, con la particularidad de que en la tercera etapa, se deben aplicar operaciones morfológicas de dilatación y erosión (en este orden), previas al proceso de etiquetación, para que éste produzca los resultados esperados.

La importancia de aplicar las operaciones morfológicas en el orden indicado, obtiene su trancendencia, en el momento de etiquetación de los objetos que componen la imagen a tratar, ya que no se podría efectuar esta actividad de forma correcta si se hubieran aplicado las operaciones en orden inverso, ya que se estaría simulando una operación de *apertura*, cuando lo que se necesita es simular un *cierre morfológico*. En la sección *2.3* del capítulo 2 se puede ampliar información acerca de estas y otros tipos de operaciones morfológicas aplicadas en algunos de los módulos implementados.

En la etapa de *cálculos y operaciones*, nos encargamos de obtener las propiedades de las regiones definidas mediante el proceso de segmentación de regiones llevado a cabo, y que

permite trabajar con las regiones obtenidas. Por lo que, se procede con el avance sobre cada una de las regiones identificadas, para aplicar las operaciones correspondientes que permitan obtener la propiedad que determina el tipo de clasificación solicitada por el usuario, según se presentaba en la sección del capítulo 2 enunciada como *Descriptores basados en regiones*.

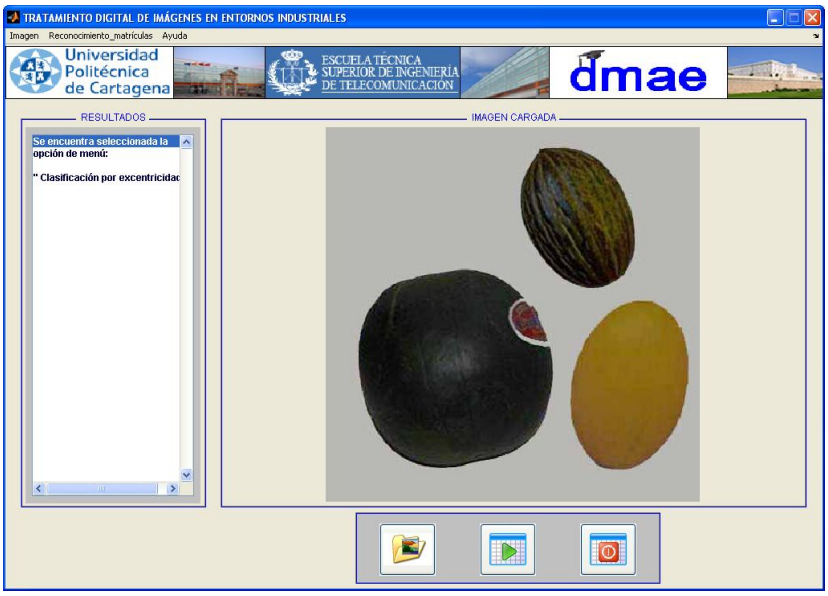


Figura 4.7: Estado GUI previo a proceso de clasificación utilizando propiedad *excentricidad*

La ejecución de la última fase (*interpretación de escena*), presenta al usuario dos tipos de figuras diferentes, en las que se pueden visualizar de forma nítida, por medio de una figura los objetos que se clasifican en la clase uno, y mediante la otra figura los objetos clasificados en la clase dos (ver figura 4.8).

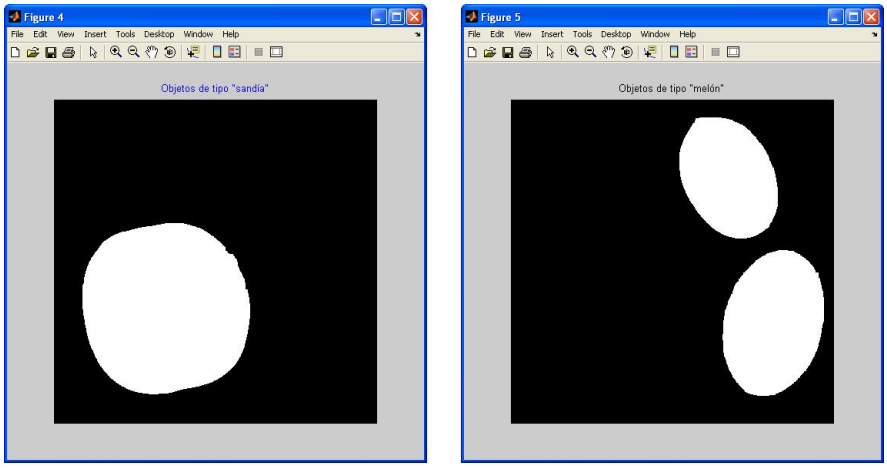


Figura 4.8: Ventanas de resultados tras proceso de clasificación utilizando propiedad *excentricidad*

Se debe destacar a su vez, que el proceso de ejecución del algoritmo, representa de forma gráfica diferentes estados intermedios, atendiendo al tipo de clasificación solicitada por el usuario, ya que se van presentando sobre cada uno de los objetos que se analizan la propiedad que a posteriori va a determinar la pertenencia de esta región a una clase u otra, es decir, que en caso de que el usuario seleccione la clasificación por área, se mostrarían de forma gráfica el área de cada objeto de la imagen, y de igual forma se procedería con la clasificación por perímetro, etc.

Sería conveniente advertir, que aunque al inicio del apartado se indicara de forma expresa como objetivo del mismo, la clasificación de objetos en dos tipos de clases diferenciadas, esto se va a cumplir siempre que el usuario seleccione una imagen cualquiera, distinta a las imágenes facilitadas en el directorio '*Apps images*' que se encuentra en el CD-ROM de datos presentado con la documentación del presente Proyecto Fin Carrera. Por lo tanto, esa va a ser la forma genérica de actuación para este módulo, sin embargo, se anima al lector a utilizar también las imágenes facilitadas en el directorio indicado, ya que con ello se pueden observar diferentes tipos de clasificaciones, que pueden ser de aplicación real y para las que se hacen necesarias un mayor número de clases. Notar también que todo ello se realiza de forma transparente al usuario de la aplicación.

4.4. Contabilización de objetos

En este módulo para la implementación de las *etapas comunes*, concretamente para la etapa de *aislamiento de los objetos de la imagen*, se tiene la particularidad de que en dicha etapa se hacen uso de operaciones de tipo *cierre morfológico*, ya que se requieren suavizar los contornos, rellenar vacíos en los mismos, así como eliminar pequeños huecos. Este *cierre morfológico* se implementa con la aplicación de operaciones de dilatación y erosión, aplicadas en el orden indicado, y utilizando en ambas como elemento estructurante un cuadrado. Se puede ampliar información acerca de este tipo de operación morfológica, mediante la consulta de la sección *2.3.6* del capítulo 2.

Respecto al resto de etapas desarrolladas en este algoritmo, la mayor relevancia de las mismas se puede caracterizar por el tipo de representación gráfica de los resultados del método, ya que el número de objetos contabilizados, se presentan al usuario por medio de una figura con un diseño específico.

La creación de la ventana de resultados, de título '*NÚMERO DE OBJETOS QUE COMPONEN LA IMAGEN*', viene dada por medio de la composición de tres imágenes diferentes, cuyo objeto es la representación de la solución obtenida por el algoritmo mediante tres dígitos. Por lo tanto, si se atiende al método de representación de los resultados, se puede revelar el número total de objetos que se permite contabilizar con el uso de este módulo, y que viene determinado por el intervalo $[0, 999]$.

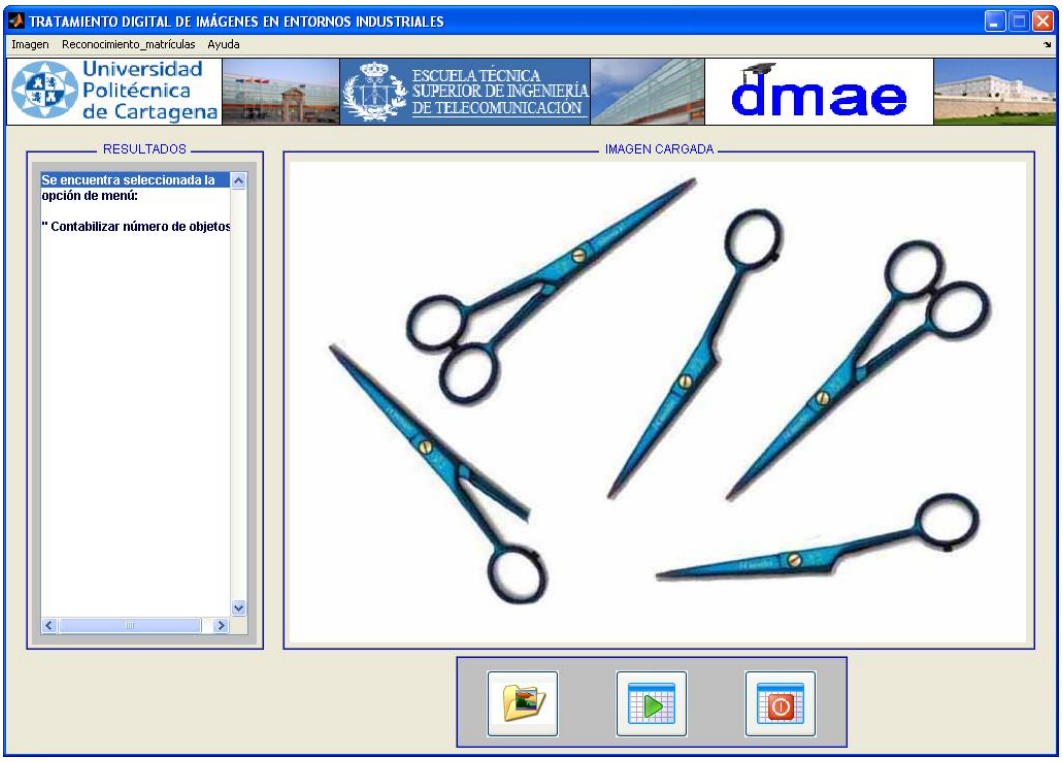


Figura 4.9: Estado GUI previo a proceso de *contabilización del número de objetos*

Figura 4.10: Ventana para la representación gráfica de resultados

4.5. Ilustración de centroides

El algoritmo implementado con este módulo, permite visualizar sobre la imagen cargada por el usuario en la interfaz gráfica principal, la posición del *centro de gravedad* en el interior de cada uno de los objetos que componen la imagen.

En este caso, al igual que en el módulo anterior, para el desarrollo de las *etapas comunes* y más concretamente para la etapa de *aislamiento de los objetos de la imagen*, se requiere hacer uso en dicha etapa de operaciones de tipo *cierre morfológico* mediante la aplicación de operaciones de dilatación y erosión, aplicadas en dicho orden, y empleando en ambas como elemento estructurante un cuadrado. Para ampliar información acerca de este y otros tipos de operaciones morfológicas se puede consultar la sección *2.3.6* del capítulo 2.

La etapa de *cálculos y operaciones* en este módulo, consiste principalmente en obtener los valores de las coordenadas que identifican la posición del *centroide* (o centro de gravedad) para cada uno de los objetos de la imagen, haciendo uso de la teoría comentada en la sección del capítulo 2 enunciada como *Descriptores basados en regiones*.

Figura 4.11: Estado GUI previo a proceso de *ilustración de centroides*

Respecto a la etapa de *interpretación de escena* que da por finalizado el algoritmo, cabe destacar que se ha elaborado la ilustración de los centroides en las coordenadas correspondientes de la imagen, mediante el símbolo '*' en color rojo.

Figura 4.12: Estado GUI tras proceso de *ilustración de centroides*

4.6. Detección de objetos defectuosos

El objetivo con el que se desarrolla la presente aplicación, consiste en la implementación de un algoritmo capaz de detectar objetos que han tenido algún tipo de incidencia (deficiencia) en el proceso de fabricación de los mismos, con lo que se lograrían detectar este tipo de objetos y se procedería a calificarlos como defectuosos, desarrollando para ello un método de clasificación que finaliza cuando se presentan mediante dos figuras distintas, por un lado los objetos que están correctamente formados, y por otro aquellos que presentan algún tipo de deficiencia, que no les permitiría ser admitidos en el proceso de control de calidad efectuado.

Acerca de la caracterización de las fases de procesamiento llevada a cabo para lograr confeccionar este algoritmo, se debe prestar especial atención a la fase de *aislamiento de los objetos de la imagen*, ya que en dicha fase se han implementado diferentes operaciones de *cierre morfológico*, con distintos tipos y tamaños de elementos estructurantes.

Como se indicaba en la sección 2.3.2 del capítulo 2 el elemento estructurante define el tamaño y la forma de la vecindad del píxel que va a ser analizado, para posteriormente alterar su valor, es decir, donde se aplica la operación morfológica. En esta fase se crearon elementos estructurantes de tamaños (radios, longitud y ángulos) determinados, y de tipos que van desde el disco plano a las líneas, para poder implementar correctamente la clasificación para

diversas imágenes que pueda tener cargadas en la aplicación el usuario cuando solicita ejecutar el algoritmo.

Para satisfacer el requerimiento anterior, se presta especial atención a la hora de determinar los tamaños y formas de los elementos estructurantes, intentando que no se alcance el *cierre morfológico total*, es decir, que el procedimiento de selección de estos parámetros se realiza de forma que se evite que la clasificación efectuada (área o perímetro) contenga alguna errata.

En la fase de *cálculos y operaciones* se debe destacar que para lograr la detección de componentes defectuosas, se hace uso en la misma de una propiedad de los objetos como es el área. A su vez, también se prepara dicha fase de forma que obtenga los resultados oportunos en caso de que resulte más apropiado desarrollar la misma, apoyándose en la propiedad perímetro de los objetos de la imagen.

Para observar algunas de las bondades que se obtienen mediante el uso de este módulo, sólo hay que visualizar las figuras que se presenta a continuación (4.13 y 4.14).

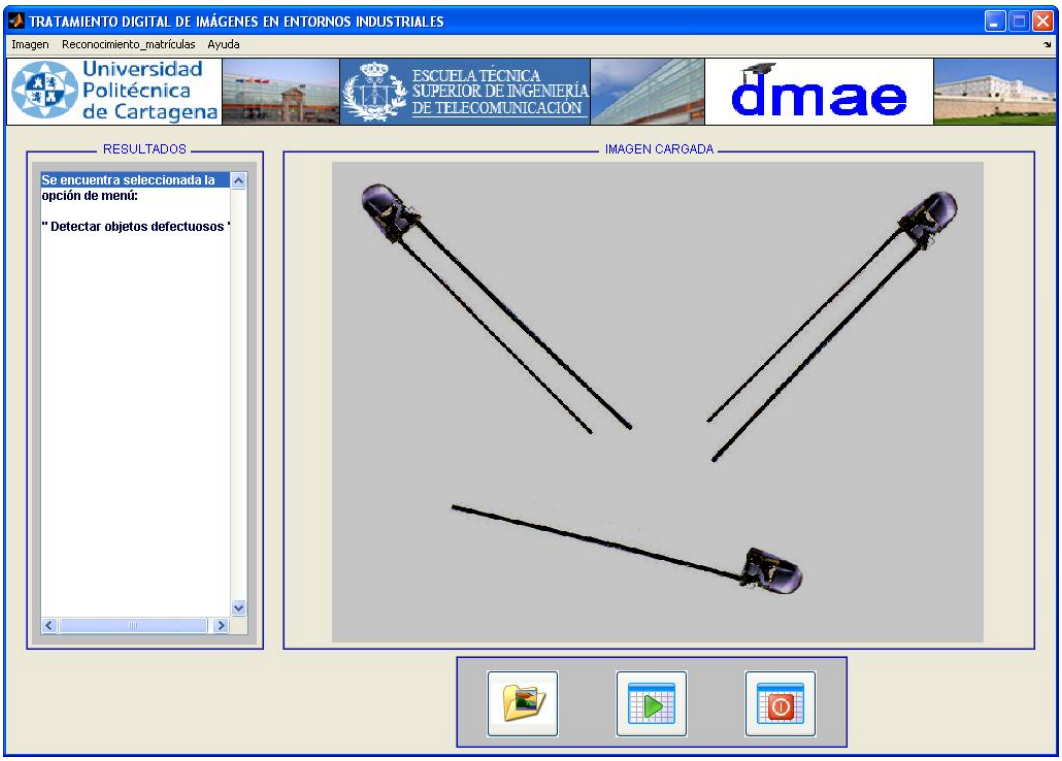


Figura 4.13: Estado GUI previo a proceso de *detección de objetos defectuosos*

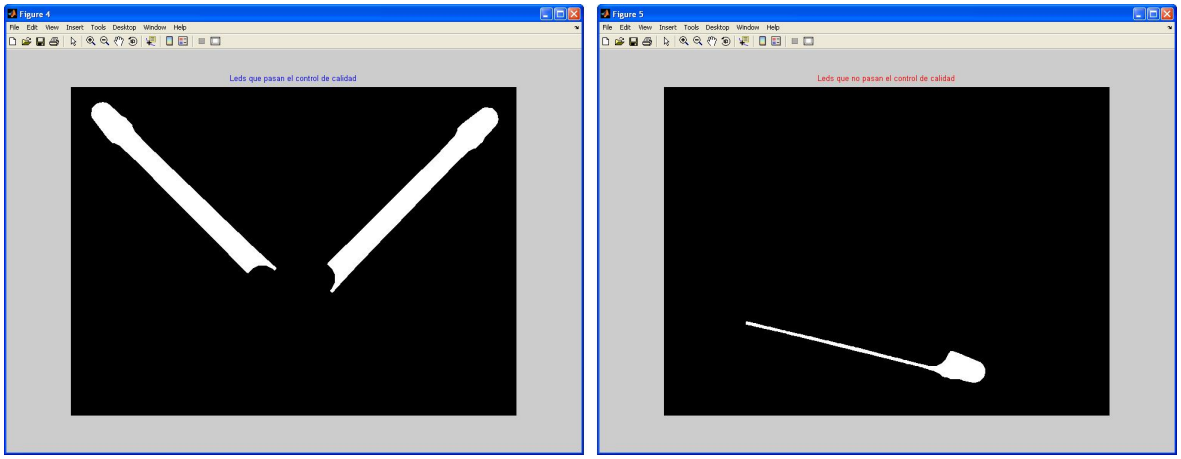


Figura 4.14: Ventanas de resultados tras proceso de *detección de objetos defectuosos*

4.7. Detección de ejes

El cometido de este módulo radica en la detección de ejes en la imagen que tenga desplegada el usuario en la interfaz gráfica de la aplicación, cuando éste selecciona alguna de las opciones disponibles en el menú *Detectar ejes*.

En caso de que el usuario optara por la opción *Mostrar bordes* se visualizarían en una figura los bordes principales de los objetos presentes en la imagen cargada, sin embargo, si seleccionase la opción *Mostrar contorno* el aplicativo presentaría una figura en la que estarían presentes únicamente los ejes de la imagen que se corresponden con el contorno de cada uno de los objetos que componen la misma.

En referencia a las etapas de procesamiento llevadas a cabo para implementar las dos variedades de detección de ejes, se debe destacar que las *etapas comunes*, se han implementado de forma semejante, con la salvedad de que se hace necesario un tipo de procesamiento específico para la opción de *Mostrar contorno* respecto a la opción de menú *Mostrar bordes*. Concretamente el tipo de procesamiento adicional implementado para el aplicativo comentado, consiste en hacer uso de una operación morfológica que no ha sido utilizada en ningún algoritmo previo, como es la operación de *apertura*. Con ello, logramos eliminar pequeñas protuberancias y romper conexiones débiles (brillo, texturas, etc) que nos permiten en las etapas posteriores obtener los resultados deseados.

La etapa de *cálculos y operaciones*, ha sido implementada de forma idéntica en ambas opciones del menú de detección de ejes, y se basa en la detección de bordes empleando para ello el *Operador de Sobel*, expuesto en la apartado correspondiente de la sección de *Segmentación* del capítulo 2.

La última fase (*interpretación de escena*), al igual que la etapa anterior, ha sido desarrollada de forma semejante para ambas opciones de menú, ya que esta fase se

encarga básicamente de mostrar al usuario mediante una figura específica, los resultados del procesamiento obtenido mediante la extracción de bordes, llevada a cabo mediante el *Operador de Sobel* en la etapa anterior.

Figura 4.15: Estado GUI previo a proceso de *detección de ejes* (opción *mostrar bordes*)

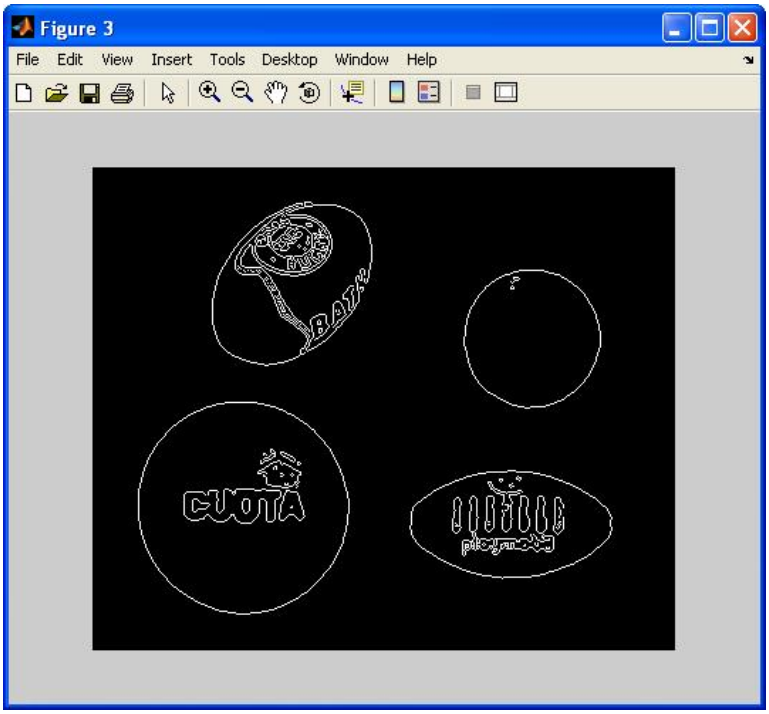


Figura 4.16: Ventana resultante tras proceso de *detección de ejes* (opción *mostrar bordes*)

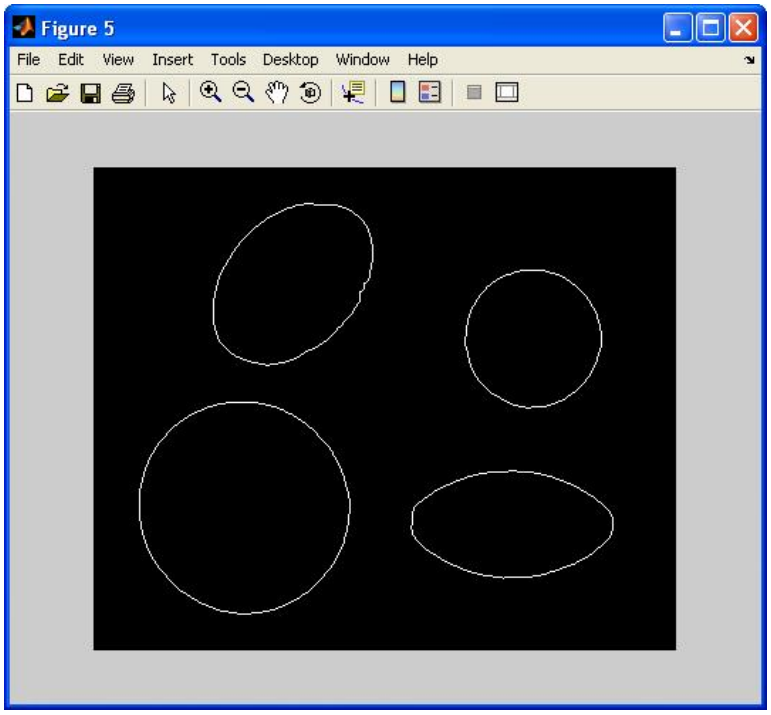


Figura 4.17: Ventana resultante tras proceso de *detección de ejes* (opción *mostrar contorno*)

4.8. Reconocimiento y tratamiento de carácteres alfanuméricos

Mediante las opciones disponibles en la entrada de menú *Reconocimiento_matrículas* de la interfaz gráfica de usuario, se pueden simular todos los elementos que conformarían un sistema de lectura y control de matrículas para parkings. Este sistema se implementa mediante las opciones de menú *Obtener matrícula*, *Comprobar acceso* y *Visualizar BBDD* que a continuación se proceden a analizar de forma pormenorizada.

Obtener matrícula

Este aplicativo se encarga de la lectura y reconocimiento (identificación) de carácteres alfanuméricos, a partir de una imagen digital que contiene la matrícula de un vehículo.

Para el entrenamiento de este módulo, se puede comenzar por utilizar cualquiera de las *10 imágenes* de matrículas de vehículos, que se pueden encontrar en el CD-ROM de datos presentado con la documentación del presente proyecto final de carrera.

En la implementación de las *etapas comunes* presentes en este módulo, cabe destacar la necesidad de calcular un *umbral global* de la escala de grises de la imagen, que permite ser utilizado posteriormente en el proceso de binarización de la imagen de la matricula (RGB) en cuestión. Este umbral óptimo de binarización de la imagen se obtiene mediante la apli-

cación del *método Otsu*, que básicamente se encarga de calcular dicho umbral mediante la maximización de la varianza entre clases por medio de un búsqueda exhaustiva. Se puede ampliar información acerca de éste método en el apartado de *Segmentación basada en umbralización* del capítulo 2.

Figura 4.18: Estado GUI previo a proceso de *obtener matrícula*

En cuanto a las etapas de *cálculos y operaciones* cabe destacar que el análisis e identificación de los carácteres numéricos de la imagen se realiza de forma diferente al de los carácteres alfabéticos, sin embargo, hay un proceso previo a ambos análisis que consiste en eliminar aquellas partes de la imagen que no nos ofrecen información de relevancia para el proceso en cuestión, es decir, antes de analizar los carácteres alfanuméricos se procede a eliminar de la imagen original ciertos elementos innecesarios como son el borde metálico de la matrícula, así como la sección de la franja de color azúl donde se encuentra el logotipo de la Unión Europea.

El procedimiento desarrollado para eliminar las componentes indicadas anteriomente, y así evitar tener que analizarlas, consiste en obtener (recortar) un subconjunto rectangular de la imagen de entrada que excluya estas componentes y que contenga únicamente los elementos de interés [9]. Aún así también se podría haber implementado este requerimiento mediante la aplicación de la propiedad área de las regiones, provocando que sólo permanecieran en la imagen aquellas regiones cuya área fuera superior a un cierto % del total.

Figura 4.19: Figuras intermedias hasta alcanzar imagen final a procesar

La etapa de *interpretación de escena* se caracteriza principalmente por una ventana de diálogo de título *Resultados*, donde se indica al usuario los carácteres alfanuméricos que componen la imagen de la matrícula que había cargada con anterioridad a la ejecución de éste módulo.

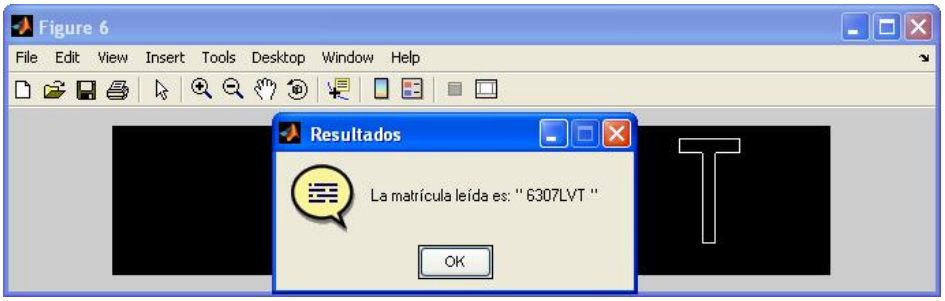


Figura 4.20: Ventana resultante tras proceso de *obtener matrícula*

Por otro lado, también cabe destacar que el procedimiento hasta lograr mostrar la ventana de resultados expuesta en la figura 4.20, se desarrolla mediante una interfaz gráfica amiga-

ble, mediante la que se permite ir presentando en una figura de forma automática, tanto el perímetro como el área del caracter(numérico o alfabético) que va analizando el algoritmo.

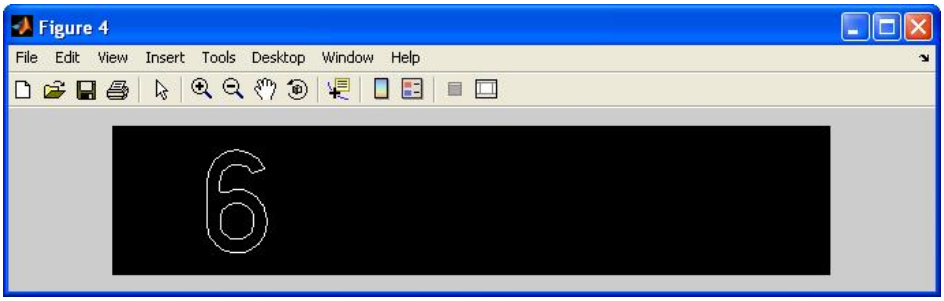


Figura 4.21: Ejemplo etapa intermedia de análisis de carácteres

Comprobar acceso

Este módulo introduce una funcionalidad con respecto al menú de *Obtener matrícula*, ya que permite comprobar en un fichero a modo de base de datos si la matrícula identificada se encuentra o no almacenada en la misma, y en caso afirmativo, a que persona se encuentra asignada la misma.

Por lo tanto, en el supuesto de que se deseara implantar este sistema de control de acceso en un parking dado, se podría determinar para un vehículo que se posicionara a la entrada del parking, si se le deja acceder al mismo (levantar barrera) o no, atendiendo a si previamente el conductor del vehículo ha sido dado de alta en dicha base de datos. A su vez, este sistema permitiría controlar ciertos aspectos como el tiempo que ha permanecido en el parking, los días y horas concretos de estacionamiento, etc.

En este caso para la implementación de las fases de *etapas comunes* y *cálculos y operaciones* se reutilizan los desarrollados llevados a cabo por medio del método anterior, agregando a la finalización de éstas, el chequeo de la base de datos de usuarios, es decir, a partir de los carácteres alfanuméricos leídos se procede a contrastar los mismos, mediante la comprobación de la existencia de éstos en la base de datos (fichero extensión **.dat*).

Para la etapa de (*interpretación de escena*) se pueden producir dos variantes de salida principales:

- **Matrícula no se encuentra en base de datos**: En caso de que la matrícula leída no se halle almacenada en la BBDD de usuarios (no corresponde a ninguna persona registrada en el sistema), se indica dicho evento mediante la ventana de diálogo correspondiente de título *Resultados*. A su vez, dicha información queda recogida también en la lista de resultados de la interfaz gráfica principal.
- **Matrícula se encuentra en base de datos**: En caso de que la matrícula identificada se halle almacenada en la BBDD de usuarios, se indica dicho evento mediante la ventana

de diálogo correspondiente de título *Resultados*. Además una vez se pulse el botón *OK* de la ventana indicada, en la lista de resultados de la interfaz gráfica principal, se muestra la información mediante nombre y apellidos del usuario (o usuarios) que hay asociados en la base de datos a la matrícula del vehículo en cuestión.

Figura 4.22: Ventana resultante si respuesta negativa tras proceso de *comprobar acceso*

Figura 4.23: Ventana resultante si respuesta positiva tras proceso de *comprobar acceso*

Figura 4.24: Estado GUI si respuesta positiva tras proceso de *comprobar acceso*

Visualizar BBDD

La labor que desarrolla este aplicativo, consiste principalmente en mostrar el contenido de la base de datos de usuarios y matrículas registradas en el sistema. La presentación de la información comentada se ilustra mediante el área de resultados de la interfaz gráfica de usuario principal, y se puede tener acceso a toda la información de la BBDD por medio de las barras desplegables (horizontales y verticales) presentes en el apartado indicado de la *GUI*.

Figura 4.25: Estado GUI tras proceso de *visualizar BBDD*

Capítulo 5

Conclusiones y líneas futuras

Con este capítulo se alcanza el cierre del presente trabajo, y por medio del mismo se van a presentar tanto las conclusiones que se han obtenido a partir de la elaboración del proyecto final de carrera, como las líneas de trabajo futuras para la mejora y continuación del sistema desarrollado.

5.1. Conclusiones

Mediante el procesamiento digital de imágenes, se permite poder implementar un amplio espectro de aplicaciones en diversos campos como la robótica, medicina, telecomunicaciones, etc. Así que el objetivo final que se esbozaba en el primer capítulo del presente trabajo, era el desarrollo de aplicaciones concretas que lograran satisfacer ciertas necesidades que pudieran ser de utilidad en el ámbito de los entornos industriales.

Por lo tanto, haciendo uso de las técnicas de tratamiento digital de imágenes y las posibilidades sobre el análisis del contenido de las mismas, se puede indicar que se ha alcanzado el objetivo principal expuesto en la premisa anterior, por medio del desarrollo de diversas herramientas software claramente orientadas para su uso en la industria, y disponibles todas ellas a través de una interfaz gráfica de usuario principal.

La interfaz gráfica de la aplicación cumple con los requisitos impuestos, en cuanto a la presentación de la misma mediante un diseño claro, que permita al usuario final, hacer uso de la misma de forma intuitiva sin necesidad de poseer conocimientos avanzados en *Matlab*®. Este objetivo se ha logrado en cada una de las pantallas y ventanas de diálogo de la interfaz, ya que el usuario es siempre consciente del estado de ejecución del algoritmo en cuestión, y de la opción de menú seleccionada.

A su vez, esta *GUI* principal se ha complementado con la implementación de un menú de ayuda, y diversas ventanas informativas que indican al usuario el procedimiento a seguir en caso de error, u otro tipo de advertencias o incidencias que fuera conveniente considerar.

Los módulos desarrollados para su aplicación en la industria, se podrían organizar atendiendo a dos vertientes principales, como son el **reconocimiento y clasificación**, así como la **inspección y control de calidad**.

En las áreas de aplicación indicadas se han desarrollado diversas herramientas que permiten, a partir de la selección del menú correspondiente en la interfaz gráfica de usuario y la imagen desplegada en la misma:

- *Contabilizar y clasificar objetos*: Esta funcionalidad se encuentra implementada por medio de diversas opciones de menú de la *GUI* principal, que permiten tanto contabilizar el número de objetos totales presentes en una imagen, como clasificar los mismos en diferentes clases. Esta clasificación se ha desarrollado de forma adaptativa al tipo de característica o características diferenciables en los objetos de la imagen, de forma que se permitan realizar clasificaciones tan diversas, como la discriminación entre frutas tales como melones y sandías, así como productos del tipo balones de rugby y fútbol.

- *Control de accesos por reconocimiento de matriculas*: Este aplicativo permite mediante el reconocimiento y tratamiento de carácteres alfanuméricos, que se ha elaborado por medio de un conjunto de algoritmos y funciones, la lectura e identificación de matrículas de vehículos, así como la consulta de la información obtenida a partir de las mismas en una base de datos de usuarios, que permite en primera instancia controlar si el vehículo se encuentra o no asociado a alguna persona registrada en el sistema.

- *Obtener características de objetos*: Los procedimiento de inspección se aplican ampliamente y con gran asiduidad en la industria de la agricultura(con frecuencia los llevan a cabo personas), y se refiere en el sentido más amplio a la verificación de si un objeto cumple con determinados criterios. Por lo tanto, se ha desarrollado un algoritmo cuyo foco de aplicación sería la industria de la agricultura y la alimentación, que permite automatizar esta tarea de inspección.
  Hacer uso de éste algoritmo permite obtener diversas características (perímetro, área, excentricidad, compacidad, centroide) de los objetos presentes en la imagen, y a partir del sesgo introducido realizar la inspección y clasificación oportuna. A modo de ejemplo, se ha utilizado este algoritmo para la clasificación de naranjas y calabacines, atendiendo a su área o perímetro.

- *Detección de defectos de producción*: Los defectos en los procesos de producción están a la orden del día y pueden provenir de muchas fuentes, incluyendo los equipos no fiables o defectuosos, procesos de manufactura erróneos tales como tiempo y temperatura de preparación incorrectos, etc. Algunos de estos defectos, podrían solucionarse con inspección geométrica, tal y como el algoritmo implementado que permite detectar, identificar y clasificar, en dos clases diferentes los objetos formados correctamente y los que presentan defectos en su producción. Esta funcionalidad se puede utilizar para detectar deficiencias tanto en objetos de tipo tijeras como leds, por poner algunos ejemplos.

- *Detección de contornos*: Por medio de la investigación y la utilización de diversas técnicas de detección de ejes, se pudo lograr desarrollar un aplicativo que permite detectar los ejes de los objetos de la imagen, y lo que es más importante, también podría determinar que ejes son los que representan únicamente el contorno del objeto. Esta aplicación podría servir de base para el reconocimiento de objetos parcialmente ocluídos mediante sus bordes, de gran utilidad en la inspección de equipajes en aeropuertos o aplicaciones militares.

- *Centro de gravedad*: Este aplicativo se utiliza principalmente para la ilustración del centro de gravedad de todos y cada uno de los objetos de la imagen, lo que puede ser utilizado como un elemento de inspección y detección de elementos defectuosos en cualquier proceso de producción industrial.

Como se puede vislumbrar, mediante la implantación en diversos sectores de la industria de las herramientas software elaboradas, se permitiría mejorar los controles de calidad, lo que provocaría que se redujera la cantidad de productos con defectos o en condiciones inapropiadas que llegan al mercado, aumentando la eficiencia y productividad, llegando a obtener productos de mejor calidad, con el correspondiente aumento de la competitividad, siendo éste un elemento tan relevante y ansiado en nuestro sistema productivo actual.

5.2. Líneas futuras

Los objetivos impuestos con la puesta en marcha de este trabajo han sido alcanzados de forma satisfactoria. Aún así la herramienta final facilitada está abierta a posibles ampliaciones o mejoras. Por lo tanto, a modo de sugerencia se pueden consultar a continuación algunas de las líneas a seguir en futuros trabajos de temáticas semejantes:

a) En general todos los procesos implementados parten de la base de tener la imagen a analizar en formato digital, tanto es así que para poder utilizar las funcionalidades de la *GUI*, se debieron realizar previamente las fotografías y composiciones que aparecen en la memoria para permitir probar las bondades de las aplicaciones desarrolladas. Por lo tanto, una posible y deseable ampliación futura, podría ser la inclusión de una opción en la interfaz gráfica de usuario que permita la automatización de la etapa de adquisición de imágenes ([5]).
Esta funcionalidad debería permitir capturar imágenes por medio de la cámara a la que tendría acceso el equipo donde se ejecute el aplicativo, así como configurar los principales parámetros de adquisición de la misma. Poner en marcha esta funcionalidad también implicaría tener en cuenta múltiples factores para automatizar la etapa de adquisición de imágenes, como serían tanto el sistema hardware de visión artificial (cámara, óptica, etc), como el entorno y posicionamiento de los elementos a capturar (iluminación, fondo, posición correcta de la cámara, etc).

b) En cuanto a las técnicas de inspección y clasificación llevadas a cabo en el presente trabajo, una posible ampliación de las mismas, podría basarse en la explotación del

reconocimiento de la propiedad *color* [6], de los objetos que compongan la imagen.
Las características cromáticas, por medio del aplicativo correspondiente podrían llegar a solucionar los problemas derivados de defectos en la producción (color inapropiado), como en la inspección de envasado de productos.
Un ejemplo de aplicación podría ser el caso en el que se necesitara detectar si un determinado envase de cápsulas de farmacia ha sido correctamente rellenado con las cápsulas correctas, identificadas por su color, y además también serviría esta característica para ampliar el algoritmo de forma que verificara si todos los compartimentos poseen una cápsula.

Apéndice A

Códigos Matlab

En este apéndice se presentan los códigos *Matlab* de las principales funciones implementadas. Se debe indicar también, que se han eludido exponer los códigos de las funciones desarrolladas para implementar la interfaz gráfica, dada su extraordinaria longitud en cuanto a líneas de código generadas. Dichas funciones, '***GUI.m***' y '***Ayuda.m***', pueden ser consultadas en el CD-ROM de datos que se incluye con la documentación del presente proyecto final de carrera.

A.1. Función *apli_ matriculas.m*

```
function resultados = apli_matriculas(rutaImagen, option)

I = imread(rutaImagen);

level = graythresh(I);

 % Se realiza la binarización de la imagen 'I'(RGB) por umbral 'level'
 % (thresholding).
Ibin = im2bw(I,level);
 % Para mostrar la imagen binarizada (en blanco y negro).
figure(1);
imshow(Ibin,2);

Iaux= Ibin;
figure(2);
imshow(Iaux, 2);

 % Se obtiene un vector de cuatro elementos que almacenamos en 'numeros_mat',
 % y que contiene los cuatro números que conforman la imagen de la matricula
 % en cuestión.
numeros_mat = getNumeros(Iaux, level);
```

```
% Se obtiene un vector de tres elementos que almacenamos en 'letras_mat', y
% que contiene las tres letras que conforman la imagen de la matricula en
% cuestión.
letras_mat = getLetras(Iaux, level);

matricula = [[numeros_mat] [letras_mat]];

% cell array donde se va almacenando toda la información a imprimir.
resultados = {};

% Si se ha seleccionado la opción de menú 'Obtener matricula'
if(option == 1)
    uiwait(helpdlg(sprintf('La matrícula leída es: %s ', matricula),...
    ' Resultados '));
    resultados{1}= sprintf('Matrícula leída: %s ', matricula);

else  % Si se ha seleccionado la opción de menú 'Comprobar acceso'
    usuario = checkBBDD(matricula);
    if(usuario == '0')
        uiwait(warndlg('Usuario no almacenado en la base de datos',...
        ' Resultados '));
        resultados{1} = sprintf('La matrícula %s', matricula);
        resultados{2} = 'No corresponde a ningún usuario';
        resultados{3} = 'almacenado en la base de datos';
    else
        uiwait(helpdlg('Usuario almacenado en la base de datos',...
        ' Resultados '));
        resultados{1} = sprintf('La matrícula %s', matricula);
        resultados{2} = 'Pertenece al usuario de nombre:';
        resultados{3} = sprintf(' %s', usuario);
    end
end
```

A.2. Función *getAllBBDD.m*

```
function Lineas = getAllBBDD()

numLineas = 0;

% Abrir archivo para leerlo.
id = fopen('BBDD\ \registro.dat','r');
if (id == -1)
        error(sprintf('El archivo %s no pudo abrirse para lectura.',...
```

```
        'nombre_archivo'))
end

% Mientras no se alcance el final del archivo se continúa leyendo
Lineas={ }; % cell array que contendrá las líneas

while  feof(id)
    numLineas = numLineas + 1;

    % Para que no almacene nada en los índices pares del cell array de
    % nombre 'Lineas'. Con ello, cuando se imprima el contenido de
    % 'Lineas' por medio del listbox de resultados, se irá imprimiendo
    % una línea en blanco entre cada matrícula-usuario leída de la base
    % de datos.
    Lineas{2*numLineas - 1} = fgetl(id); % Lee toda la línea.
end

% Cierra el archivo leído.
fclose(id);
end
```

A.3. Función *checkBBDD.m*

```
function usuario = checkBBDD(matricula)

count = 0;
usuario = '0';

% Se abre archivo para leerlo.
id = fopen('BBDD\ \registro.dat','r');
if (id == -1)
    error(sprintf('El archivo " %s"no pudo abrirse para lectura.',...
    'nombre_archivo'))
end

% Mientras no se alcance el final del archivo se continúa leyendo.
Lineas={ }; % cell array que contendrá las líneas.
while  feof(id)
    % Lee toda la línea.
    linea = fgetl(id);

    for r=1:length(matricula)
        if(matricula(r)==linea(r))
```

```
            count = count + 1;
        end
    end

    if(count == length(matricula))
        % 'ENCONTRADO'
        for t=8:length(linea)
            usuario(t-7) = linea(t);
        end
        break;
    else
        % 'NO ENCONTRADO'
        count = 0;
        Lineas{end+1,1}=linea;
    end
end

% Cierra el archivo leído.
fclose(id);
end
```

A.4. Función *getNumeros.m*

```
function numeros_matricula = getNumeros(Iaux, level)

Idil = bwmorph(Iaux,'dilate',3);
Icierre = bwmorph(Idil,'erode',3);
fig = figure(3);
imshow(Icierre, 2);

% Para poner el título de la ventana.
set(fig, 'Name', ' Imágen a tratar para obtener números matrícula');
% Para que no salga como título de la ventana 'Figure X'.
set(fig, 'NumberTitle', 'off');

% Etiquetamos las componentes existentes en la imagen binaria 'Icierre'.
[Ilabel,nobj] = bwlabel(Icierre);

figure(4);

%Para no analizar las letras, ni el marco de la matrícula.
for j=2:(nobj-3)
```

```
    %Se obtienen las coordenadas de cada objeto de la imagen.
    [n,m] = find(Ilabel==j);
    %Se crea una matriz de ceros del tamaño de la imágen original.
    G = zeros(size(Iaux));

    for i=1:length(n)
        G(n(i),m(i))=1;
    end

    imshow(G);

    a = find(G);
    x = size(a);

    numeros = '0123456789';
    num = zeros(1, length(numeros));

    for k = 0:9
        rutaIm = sprintf('Apps images\ \Chars&Nums\ \ %d.jpg', k);
        Imn = imread(rutaIm);
        Ibinn = im2bw(Imn,level);
        Iauxn= Ibinn;

        Idiln = bwmorph(Iauxn,'dilate',3);
        Icierren = bwmorph(Idiln,'erode',3);

        b = find(Icierren);
        y = size(b);

        m = x - y;
        num(k+1) = m(1);
    end

    numeros_matricula(j-1) = numeros(getMinIndex(num));

    Iperim = bwperim(G);
    imshow(Iperim,2);
    pause(1);

end
end
```

A.5. Función *getLetras.m*

```
function letras_matricula = getLetras(Icierre, level)

 % Para no tratar la parte de la matrícula donde aparece el logotipo de
 % la unión europea, con el fondo de color azúl y la letra E. Se recorta
 % la imagen de la matrícula, evitando recortar la parte de interés.
cropsize = [52 8 468 94];  % Determina sección a cortar.
Ic = imcrop(Icierre,cropsize);  % Recorta la imagen para eliminar
                                % fondo innecesario.

 % Imagen a tratar para obtener las letras de la matrícula.
fi = figure(5);
imshow(Ic);

 % Para poner el título de la ventana.
set(fi, 'Name', ' Imágen a tratar para obtener letras matrícula');
 % Para que no salga como título de la ventana 'Figure X'.
set(fi, 'NumberTitle', 'off');

[Ilabel,nobj] = bwlabel(Ic);

figure(6);

 % Para no analizar los números.
for j=5:nobj
     % Se obtienen las coordenadas de cada objeto de la imagen.
    [n,m] = find(Ilabel==j);
     % Se crea una matriz de ceros del tamaño de la imagen original
    G = zeros(size(Icierre));

    for i=1:length(n)
        G(n(i),m(i)) = 1;
    end

    imshow(G);

    a = find(G);
    x = size(a);

    letras = 'BCDFGHJKLMNPRSTVWXYZ';
    numeros = zeros(1, length(letras));

    for q=1:length(letras)
```

```
        s = getResult(letras(q), level, x);
        numeros(q) = s(1);
    end

    letras_matricula(j-4) = letras(getMinIndex(numeros));

    Iperim = bwperim(G);
    imshow(Iperim,2);
    pause(1);
end
end
```

A.6. Función *getMinIndex.m*

```
function sol = getMinIndex(numeros)

sol = 1;
n1 = abs(numeros(1));

for k=1:(length(numeros)-1)
    n2 = abs(numeros(k+1));

    if((n1 - n2) >= 0)
        n1 = n2;
        sol = k+1;
    end
end
end
```

A.7. Función *printNumObjects.m*

```
function printNumObjects(rutaImagen)

I = imread(rutaImagen);

 % Se realiza la binarización de la imagen 'I'(RGB) por umbral (thresholding).
Ibin=im2bw(I,0.5);

Iaux= Ibin;
Idilate=bwmorph(Iaux,'dilate',8);
Icierre=bwmorph(Idilate,'erode',8);

nobjs = numObjects(Icierre);
```

A.8. Función *printCentroid.m*

```
function printCentroid(rutaImagen)

bw = imread(rutaImagen);

 % Se realiza la binarización de la imagen 'bw'(RGB) por umbral (thresholding).
Ibin = im2bw(bw,0.5);

Iaux= Ibin;

 % Se hace necesario dilatar y erosionar, para eliminar brillos y otro tipo
 % de imperfecciones, ya que sino el programa interpretaría que hay más
 % objetos de los que hay en realidad, y se calcularían más centroides de los
 % adecuados y en ubicaciones incorrectas.
Idilate=bwmorph(Iaux,'dilate',8);
Icierre=bwmorph(Idilate,'erode',8);

 % Etiquetamos las componentes existentes en la imagen binaria 'Icierre'.
Ilabel = bwlabel(Icierre);

imshow(bw);

s = regionprops(Ilabel, 'centroid');
centroids = cat(1, s.Centroid);
hold on;
 % Pintamos centroide de color magenta ('m') y con el símbolo '*'.
 % Otras opciones que quedan bien son el 's'(cuadrado) y el 'o'(círculo).
 % Se estima como la opción más apropiada a la hora de dibujar los centroides,
 % que se ilustren todos sin pausa alguna, ni necesidad de que deba interactuar
 % con el usuario, para cada centroide que se dibuje.
plot(centroids(:,1), centroids(:,2), 'm*');
hold off;
```

A.9. Función *getImag.m*

```
function imagen = getImag(num)

imagen = imread(sprintf('Apps images\ \Results\ \num_ %d.jpg', num));
```

A.10. Función *numObjects.m*

```
function num_objects = numObjects(imag)

 % Se etiquetan las componentes existentes en la imagen binaria 'imag'.
[Ilabel,num_objects] = bwlabel(imag);

 % El número de objetos se convierte en un string que se almacena en la
 % variable denominada como 'string_num'.
string_num = num2str(num_objects);

 % El siguiente bucle es el encargado de abrir las 3 imágenes almacenadas en
 % disco, necesarias para poder formar y mostrar posteriormente por pantalla
 % los 3 dígitos del número de objetos obtenidos. Dichas imágenes quedan
 % almacenadas en las variables I1, I2 e I3, respectivamente.
if(num_objects >= 10)
     % Si el número de objetos es un número de 3 dígitos.
    if(num_objects >= 100)
        I1 = getImag(str2num(string_num(1)));
        I2 = getImag(str2num(string_num(2)));
        I3 = getImag(str2num(string_num(3)));
     % Si el número de objetos es un número de 2 dígitos.
    else
        I1 = getImag(0);
        I2 = getImag(str2num(string_num(1)));
        I3 = getImag(str2num(string_num(2)));
    end
 % Si el número de objetos es un número de 1 dígito
else
    I1 = getImag(0);
    I2 = getImag(0);
    I3 = getImag(str2num(string_num(1)));
end


fig = figure;

 % Para colorear de color blanco todo el fondo de la ventana.
set(fig, 'color', 'w');

 % Para poner el título de la ventana.
set(fig, 'Name', 'NÚMERO DE OBJETOS QUE COMPONEN LA IMÁGEN');

 % Cambiamos el estilo de la ventana para que no aparezca todo el menú de la
```

```
% ventana de figura, así se ve el texto de la ventana mejor.
% Esto también hace que hasta que no se cierre esta ventana no se pueda
% acceder a otro elemento de Matlab, ni ventanas que hayan aparecido con
% anterioridad.
set(fig, 'WindowStyle', 'modal');

% Para que no salga como título de la ventana 'Figure X'.
set(fig, 'NumberTitle', 'off');

subplot(1,3,1), imshow(I1);
subplot(1,3,2), imshow(I2);
subplot(1,3,3), imshow(I3);
```

A.11. Función *getResult.m*

```
function distancia = getResult(name, levels, sizeIm)

name = name;

rutaImag = sprintf(['Apps images\ \Chars&Nums\ \' name '.jpg']);
name = imread(rutaImag);

Ibin = im2bw(name,levels);
Iaux = Ibin;

Icierre = Iaux;
b = find(Icierre);
y = size(b);

distancia = sizeIm - y;
```

A.12. Función *getProperties.m*

```
function resultados = getProperties(rutaImagen, option)

I = imread(rutaImagen);

% Se realiza la binarización de la imagen 'I'(RGB) por umbral (thresholding).
Ibin=im2bw(I,0.5);
% Para mostrar la imagen binarizada (en blanco y negro).
figure(1);
imshow(Ibin,2);
```

```
% ' ' => Equivale a aplicar la función lógica NOT.
% Invertimos los colores blanco y negro.
Iaux= Ibin;
figure(2);
imshow(Iaux, 2);

% Prestar atención a que primero se dilata y después se erosiona.
Idilate=bwmorph(Iaux,'dilate',8);
Icierre=bwmorph(Idilate,'erode',8);
figure(3);
imshow(Icierre,2);

% Etiquetamos las componentes existentes en la imagen binaria 'Icierre'.
[Ilabel,nobj]=bwlabel(Icierre);

if(option == 1 | option == 4)
    areas=zeros(1,nobj);
end

if(option == 2 || option == 4)
    perim=zeros(1,nobj);
end

if(option == 3)
    excentricidades = regionprops(Ilabel,'Eccentricity');
end

if(option == 5)
    centroides = regionprops(Ilabel, 'centroid');
end

% Se recorren todos los objetos de la imagen.
for j=1:nobj
    % Se obtienen las coordenadas de cada objeto de la imagen.
    [n,m]=find(Ilabel==j);
    % Se crea una matriz de ceros del tamaño de la imagen original.
    G=zeros(size(Iaux));

    for i=1:length(n)
        G(n(i),m(i))=1;
    end

    imshow(G);
```

```
if(option == 1 || option == 4)
    areas(j)=bwarea(G);
end

Iperim=bwperim(G);
imshow(Iperim,2);
if(option == 2 || option == 4)
    perim(j)=bwarea(Iperim);
end

 % Para ir pintando el centroide sobre el elemento en cuestión.
if(option == 5)
    centroids = cat(1, centroides(j).Centroid);
    hold on;
     % Pintamos centroide de color verde ('g') y con el símbolo '*'.
     % Otras opciones que quedan bien son el 's'(cuadrado) y el 'o'
     % (círculo).
    plot(centroids(:,1), centroids(:,2), 'g*');
    hold off;
end

 % Mostramos un cuadro de diálogo informando del objeto seleccionado y
 % el valor de la propiedad requerida. El uiwait se utiliza para que el
 % usuario deba de cliquear con el ratón sobre el botón 'OK' del cuadro
 % de diálogo ó bien pulsar el botón intro, para que el programa siga
 % calculando las propiedades del siguiente objeto de la imágen.
 % uiwait(msgbox(sprintf('El área del objeto número %d es: %g', j, ...
 % areas(j)),'Resultados','modal'));
if(option == 1)
    uiwait(helpdlg(sprintf('El área del objeto número %d es: %g', j,...
    areas(j)),' Resultados '));
    else if(option == 2)
        uiwait(helpdlg(sprintf('El perímetro del objeto número %d es: %g'...
        , j, perim(j)),' Resultados '));
        else if(option == 3)
            uiwait(helpdlg(sprintf('La excentricidad del objeto número %d es: %g'...
            , j, excentricidades(j).Eccentricity),' Resultados '));
            else if(option == 4)
                uiwait(helpdlg(sprintf('La compacidad del objeto número %d es: %g'...
                , j, (perim(j).^2)./areas(j)),' Resultados '));
                else if(option == 5)
                    aux = centroides(j).Centroid;
                     % Cuadro de diálogo con varias líneas.
                    uiwait(helpdlg({sprintf(...
```

```
                    'El centroide del objeto número %d está en las coordenadas:'...
                    , j), sprintf(' x= %g, y= %g' , aux(1),...
                    aux(2))}, ' Resultados '));
                end
            end
        end
    end
end

if(option == 1)
    resultados = areas;
    else if(option == 2)
        resultados = perim;
        else if(option == 3)
            resultados = zeros(1, nobj);
            for k=1:nobj
                resultados(k) = excentricidades(k).Eccentricity;
            end
            else if(option == 4)
                resultados = (perim.^2)./areas;  % COMPACIDADES
                else if(option == 5)
                    resultados = centroides;
                end
            end
        end
    end
end
```

A.13. Función *buildResults.m*

```
function imprimir = buildResults(option, results)

 % cell array que va a contener toda la información a imprimir
imprimir = {};

[t, numObjs] = size(results);

if(option == 1)
    imprimir{1} = sprintf('Áreas');
    else if(option == 2)
        imprimir{1} = sprintf('Perímetros');
        else if(option == 3)
```

```
            imprimir{1} = sprintf('Excentricidades');
            else if(option == 4)
                imprimir{1} = sprintf('Compacidades');
                else if(option == 5)
                    imprimir{1} = sprintf('Coordenadas Centroides');
                end
            end
        end
    end
end

if(option  = 5)
    for i=1:numObjs
        imprimir{i+1}= [sprintf('Objeto número %d: ', i)...
        sprintf(' %g \n', results(i))];
    end
else
    for i=1:t
        aux = results(i).Centroid;
        imprimir{i+1}= [sprintf('Objeto número %d: ', i)...
        sprintf(' x= %g, y= %g\n', aux(1), aux(2))];
    end
end
```

A.14. Función *getEdges.m*

```
function getEdges(rutaImagen)

I = imread(rutaImagen);

% Se realiza la binarización de la imagen 'I'(RGB) por umbral (thresholding).
Ibin = im2bw(I,0.5);
% Para mostrar la imagen binarizada (en blanco y negro).
figure(1);
imshow(Ibin,2);

% ' ' => Equivale a aplicar la función lógica NOT.
% Invertimos los colores blanco y negro.
Iaux= Ibin;
figure(2);
imshow(Iaux, 2);

% Extraemos bordes (prewitt/sobel/canny).
```

```
ImageB = edge(Iaux, 'sobel');
figure(3)
imshow(ImageB, 2);
```

A.15. Función *getContour.m*

```
function getContour(rutaImagen)

I = imread(rutaImagen);

 % Se realiza la binarización de la imagen 'I'(RGB) por umbral (thresholding).
Ibin=im2bw(I,0.5);
 % Para mostrar la imagen binarizada (en blanco y negro).
figure(1);
imshow(Ibin,2);

 % ' ' => Equivale a aplicar la función lógica NOT.
 % Invertimos los colores blanco y negro.
Iaux= Ibin;
figure(2);
imshow(Iaux, 2);

 % Se aplican operaciones de dilatación y erosión, para eliminar algunos de
 % los contenidos de los objetos, ya que no nos interesan (brillo, letras,
 % texturas...), aunque no se deben suprimir en su totalidad.
Idilate=bwmorph(Iaux,'dilate',2);
Icierre=bwmorph(Idilate,'erode',2);
figure(3);
imshow(Icierre,2);

figure(4);
 % bwareaopen(I, N); Esta función elimina los objetos de la imagen 'I' que
 % tengas menos de 'N' píxels (Suprimimos el resto de brillo, texturas,
 % letras, etc.), que contegan los objetos de la imagen.
I2 = bwareaopen( Icierre, 3000);
imshow(I2,2);

 % Extraemos bordes (prewitt/sobel/canny).
ImageB = edge(I2, 'sobel');
figure(5)
imshow(ImageB, 2);
```

A.16. Función *clasifAreaPerim.m*

```
function clasifAreaPerim(rutaImagen, clasificacion, typeImg)

I = imread(rutaImagen);

 % Se realiza la binarización de la imagen 'I'(RGB) por umbral (thresholding).
Ibin=im2bw(I,0.5);
 % Para mostrar la imagen binarizada (en blanco y negro).
figure(1);
imshow(Ibin,2);

 % ' ' => Equivale a aplicar la función lógica NOT.
 % Invertimos los colores blanco y negro.
Iaux= Ibin;
figure(2);
imshow(Iaux, 2);

 % bworph(BW, OPERATION, N); Aplica la operación morfológica seleccionada
 % ('OPERATION') sobre la imagen binaria 'BW' y la operación es respetida 'N'
 % veces.
 % 'dilate': Para dilatación de imsgen binaria.
 % 'erode': Para erosión de imsgen binaria.

 % Prestar atención a que primero se dilata y después se erosiona.
Idilate=bwmorph(Iaux,'dilate',8);
Icierre=bwmorph(Idilate,'erode',8);
figure(3);
imshow(Icierre,2);
pause;

 % Etiquetamos las componentes existentes en la imagen binaria 'Icierre'.
[Ilabel,nobj]=bwlabel(Icierre);

areas=zeros(1,nobj);
perim=zeros(1,nobj);
ObjectsCal1 = zeros(size(Iaux));
ObjectsCal2 = zeros(size(Iaux));
ObjectsCal3 = zeros(size(Iaux));

 % Se recorren todos los objetos de la imagen.
for j=1:nobj
    % Se obtienen las coordenadas de cada objeto de la imagen.
    [n,m]=find(Ilabel==j);
```

```
 % Se crea una matriz de ceros del tamaño de la imagen original.
G=zeros(size(Iaux));

for i=1:length(n)
    G(n(i),m(i))=1;
end

 % Si se desea hacer claficicación por área.
if(clasificacion == 1)
    imshow(G);
     % bwarea(BW); Estima el área de los objetos de la imagen binaria BW.
     % Devuelve un escalar cuyo valor corresponde aproximadamente
     % (podrían no ser los mismos) con el número total de píxeles de la
     % imagen.
    areas(j)=bwarea(G);

     % Realizando la clasificación atendiendo a la propiedad 'Área'.
    if(typeImg == 1)
        if(areas(j) > 7500)
            if(areas(j) > 9000)
                for i=1:length(n)
                    ObjectsCal1(n(i),m(i))=1;
                end
            else
                for i=1:length(n)
                    ObjectsCal2(n(i),m(i))=1;
                end
            end
        else
            for i=1:length(n)
                ObjectsCal3(n(i),m(i))=1;
            end
        end
    else
        if(areas(j) > 10000)
            for i=1:length(n)
                ObjectsCal1(n(i),m(i))=1;
            end
        else
            for i=1:length(n)
                ObjectsCal2(n(i),m(i))=1;
            end
        end
    end
```

```
else % Si se desea hacer claficicación por perímetro

    % bwperim (BW); Devuelve una imagen binaria que contiene sólo los
    % píxeles del perímetro de los objetos en la imagen de entrada BW.
    % Un píxel es parte del perímetro, si es distinto de cero y está
    % conectado a él, si al menos tiene un píxel con valor cero. La
    % conectividad por defecto para dos dimensiones es 4, y 6 para tres
    % dimensiones.
    % Para ir mostrando el perímetro del objeto.
    Iperim=bwperim(G);
    imshow(Iperim,2);
    perim(j)=bwarea(Iperim);

    % Realizando la clasificación atendiendo a la propiedad 'Perímetro'.
    if(typeImg == 1)
        if(perim(j) > 300)
            if(perim(j) > 340)
                for i=1:length(n)
                    ObjectsCal1(n(i),m(i))=1;
                end
            else
                for i=1:length(n)
                    ObjectsCal2(n(i),m(i))=1;
                end
            end
        else
            for i=1:length(n)
                ObjectsCal3(n(i),m(i))=1;
            end
        end
    else
        if(perim(j) > 600 & perim(j) < 700)
            for i=1:length(n)
                ObjectsCal1(n(i),m(i))=1;
            end
        else
            for i=1:length(n)
                ObjectsCal2(n(i),m(i))=1;
            end
        end
    end
end

 % Para no tener que estar pulsando una tecla, las pausas se realizan de
```

```
    % un segundo
    pause(1);
end

if(typeImg == 1)
    figure(4);
    imshow(ObjectsCal1,2);
    title('Naranjas mandarinas de Primera Calidad', 'color', 'b');

    figure(5);
    imshow(ObjectsCal2,2);
    title('Naranjas mandarinas de Segunda Calidad', 'color', 'k');

    figure(6);
    imshow(ObjectsCal3,2);
    title('Naranjas mandarinas de Baja Calidad', 'color', 'r');
else
    figure(4);
    imshow(ObjectsCal1,2);
    title('Calabacines que pasan a ser manufacturados', 'color', 'b');

    figure(5);
    imshow(ObjectsCal2,2);
    title('Calabacines que están defectuosos (se desechan)', 'color', 'r');
end
```

A.17. Función *clasifExcentric.m*

```
function clasifExcentric(rutaImagen, typeImg)

I = imread(rutaImagen);

% Se realiza la binarización de la imagen 'I'(RGB) por umbral (thresholding).
Ibin=im2bw(I,0.5);
% Para mostrar la imagen binarizada (en blanco y negro).
figure(1);
imshow(Ibin,2);

% ' ' => Equivale a aplicar la función lógica NOT.
% Invertimos los colores blanco y negro.
Iaux= Ibin;
figure(2);
imshow(Iaux, 2);
```

```
% Prestar atención a que primero se dilata y después se erosiona.
Idilate=bwmorph(Iaux,'dilate',8);
Icierre=bwmorph(Idilate,'erode',8);
figure(3);
imshow(Icierre,2);
pause;

% Etiquetamos las componentes existentes en la imagen binaria 'Icierre'.
[Ilabel,nobj]=bwlabel(Icierre);

areas=zeros(1,nobj);
perim=zeros(1,nobj);
ObjectsExc1 = zeros(size(Iaux));
ObjectsExc2 = zeros(size(Iaux));

% Almacenamos en el vector 'excValues', los valores de excentricidad de cada
% uno de los objetos que conforman la imagen.
excValues = regionprops(Ilabel,'Eccentricity');

% Se recorren todos los objetos de la imagen.
for j=1:nobj
    % Se obtienen las coordenadas de cada objeto de la imagen.
    [n,m]=find(Ilabel==j);
    % Se crea una matriz de ceros del tamaño de la imagen original.
    G=zeros(size(Iaux));

    for i=1:length(n)
        G(n(i),m(i))=1;
    end

    imshow(G);

    % bwarea(BW); Estima el área de los objetos de la imagen binaria BW.
    % Devuelve un escalar cuyo valor corresponde aproximadamente (podrían no
    % ser los mismos) con el número total de píxeles de la imagen.
    areas(j)=bwarea(G);

    % Se realiza la clasificación por excentricidad.
    if(excValues(j).Eccentricity < 0.5)
        for i=1:length(n)
            ObjectsExc1(n(i),m(i))=1;
        end
    else
```

```
        for i=1:length(n)
            ObjectsExc2(n(i),m(i))=1;
        end
    end
    % Para ir mostrando el perímetro del objeto.
    Iperim=bwperim(G);
    imshow(Iperim,2);
    perim(j)=bwarea(Iperim);

    % Para no tener que estar pulsando una tecla, las pausas se realizan de
    % un segundo.
    pause(1);
end

if (typeImg == 1)
    figure(4);
    imshow(ObjectsExc1,2);
    % title('Objetos con valor alto de excentricidad', 'color', 'b');
    title('Balones con forma esférica', 'color', 'b');

    figure(5);
    imshow(ObjectsExc2,2);
    % title('Objetos con valor bajo de excentricidad', 'color', 'k');
    title('Balones con forma oval', 'color', 'k');
else
    figure(4);
    imshow(ObjectsExc1,2);
    title('Objetos de tipo sandía', 'color', 'b');

    figure(5);
    imshow(ObjectsExc2,2);
    title('Objetos de tipo melón', 'color', 'k');
end
```

A.18. Función *clasifCompacidad.m*

```
function clasifCompacidad(rutaImagen)

I = imread(rutaImagen);

% Se realiza la binarización de la imagen 'I'(RGB) por umbral (thresholding).
Ibin=im2bw(I,0.5);
% Para mostrar la imagen binarizada (en blanco y negro).
```

```
figure(1);
imshow(Ibin,2);

% ' ' => Equivale a aplicar la función lógica NOT.
% Invertimos los colores blanco y negro.
Iaux= Ibin;
figure(2);
imshow(Iaux, 2);

% Prestar atención a que primero se dilata y después se erosiona.
Idilate=bwmorph(Iaux,'dilate',8);
Icierre=bwmorph(Idilate,'erode',8);
figure(3);
imshow(Icierre,2);
pause;

% Etiquetamos las componentes existentes en la imagen binaria 'Icierre'.
[Ilabel,nobj]=bwlabel(Icierre);

areas = zeros(1,nobj);
perim = zeros(1,nobj);
ObjectsTipo1 = zeros(size(Iaux));
ObjectsTipo2 = zeros(size(Iaux));

for j=1:nobj
    [n,m]=find(Ilabel==j);
    G=zeros(size(Iaux));

    for i=1:length(n)
        G(n(i),m(i))=1;
    end

    imshow(G);
    areas(j)=bwarea(G);

    % Para ir mostrando el perímetro del objeto.
    Iperim=bwperim(G);
    imshow(Iperim,2);
    perim(j)=bwarea(Iperim);
    pause(1);
end

compacidades = (perim.^2)./areas;
% Para clasificar los objetos en dos tipos, atendiendo a la proopiedad de
```

```
% compacidad, utilizamos como sesgo la media de dicha propiedad de los
% objetos de la imagen.
sesgo = mean(compacidades);

for j=1:nobj
    [n,m]=find(Ilabel==j);

    if(compacidades(j) > sesgo)
        for i=1:length(n)
            ObjectsTipo1(n(i),m(i)) = 1;
        end
    else
        for i=1:length(n)
            ObjectsTipo2(n(i),m(i)) = 1;
        end
    end
end

figure(4);
imshow(ObjectsTipo1,2);
title('Objetos tipo 1', 'color', 'b');

figure(5);
imshow(ObjectsTipo2,2);
title('Objetos tipo 2', 'color', 'k');
```

A.19. Función *clasifObjects.m*

```
function clasifObjects(rutaImagen, option)

I = imread(rutaImagen);

% Se realiza la binarización de la imagen 'I'(RGB) por umbral (thresholding).
Ibin=im2bw(I,0.5);
% Para mostrar la imagen binarizada (en blanco y negro).
figure(1);
imshow(Ibin,2);

% ' ' => Equivale a aplicar la función lógica NOT.
% Invertimos los colores blanco y negro.
Iaux= Ibin;
figure(2);
imshow(Iaux, 2);
```

```
% Prestar atención a que primero se dilata y después se erosiona.
Idilate=bwmorph(Iaux,'dilate',8);
Icierre=bwmorph(Idilate,'erode',8);
figure(3);
imshow(Icierre,2);
pause;

% Etiquetamos las componentes existentes en la imagen binaria 'Icierre'.
[Ilabel,nobj]=bwlabel(Icierre);

% Dependiendo de si la opción de menú seleccionada, es la de clasificación
% por área, perímetro ó excentricidad, en la variable 'propiedades' se van
% a ir almacenando las áreas, perímetros ó excentricidades de los objetos
% de la imagen, respectivamente.
if(option == 3)  % Opción excentricidad.
    propiedades = regionprops(Ilabel,'Eccentricity');
else  % Opciones área y perímetro
    propiedades = zeros(1,nobj);
end

ObjectsTipo1 = zeros(size(Iaux));
ObjectsTipo2 = zeros(size(Iaux));

% Si no estamos en clasificación por excentricidad.
if (option  = 3)
    % Se recorren todos los objetos de la imagen.
    for j=1:nobj
        % Se obtienen las coordenadas de cada objeto de la imagen.
        [n,m]=find(Ilabel==j);
        % Se crea una matriz de ceros del tamaño de la imagen original.
        G=zeros(size(Iaux));

        for i=1:length(n)
            G(n(i),m(i))=1;
        end

        if(option == 1)
            % Si se ha seleccionado la opción de clasificación por área, se
            % va a ir mostrando visualmente como se obtiene el área de cada
            % objeto.
            imshow(G);
            propiedades(j)=bwarea(G);
        else if(option == 2)
```

```
            Iperim=bwperim(G);
             % Si se ha seleccionado la opción de clasificación por
             % perimetro, se va a ir mostrando visualmente como se obtiene
             % el perímetro de cada objeto.
            imshow(Iperim,2);
            propiedades(j)=bwarea(Iperim);
            end
        end
         % Para no tener que estar pulsando ninguna tecla del equipo, las
         % pausas se realizan de un segundo.
        pause(1);
    end
end

% Para clasificar los objetos en dos tipos, ya sea atendiendo a su área o
% perímetro, utilizamos como sesgo la media de dicha propiedad (área ó
% perímetro) en todos los elementos de la imágen. Sin embargo para la
% excentricidad escogeremos el sesgo = 0.5, ya que es el valor medio que
% puede tomar la excentriciad ([0,1]).
if(option  = 3)
    sesgo = mean(propiedades);
     % Se recorren todos los objetos de la imagen.
    for j=1:nobj
         % Se obtienen las coordenadas de cada objeto de la imagen.
        [n,m]=find(Ilabel==j);

        if(propiedades(j) > sesgo)
            for i=1:length(n)
                ObjectsTipo1(n(i),m(i)) = 1;
            end
        else
            for i=1:length(n)
                ObjectsTipo2(n(i),m(i)) = 1;
            end
        end
    end

else
    sesgo = 0.5;

     % Se recorren todos los objetos de la imagen.
    for j=1:nobj
         % Se obtienen las coordenadas de cada objeto de la imagen.
        [n,m]=find(Ilabel==j);
```

```
        if(propiedades(j).Eccentricity > sesgo)
            for i=1:length(n)
                ObjectsTipo1(n(i),m(i)) = 1;
            end
        else
            for i=1:length(n)
                ObjectsTipo2(n(i),m(i)) = 1;
            end
        end
    end
end

figure(4);
imshow(ObjectsTipo1,2);
title('Objetos tipo 1', 'color', 'b');

figure(5);
imshow(ObjectsTipo2,2);
title('Objetos tipo 2', 'color', 'k');
```

A.20. Función *aplicDefectA.m*

```
function aplicDefectA(rutaImagen)

I = imread(rutaImagen);

% Se realiza la binarización de la imagen 'I'(RGB) por umbral (thresholding).
Ibin=im2bw(I,0.5);
% Para mostrar la imagen binarizada (en blanco y negro).
figure(1);
imshow(Ibin,2);

% ' ' => Equivale a aplicar la función lógica NOT.
% Invertimos los colores blanco y negro.
Iaux= Ibin;
figure(2);
imshow(Iaux, 2);

% strel('line',R,N); Crea una línea en forma de elemento estructurante
% de longitud, 'R' y cuyo ángulo en grados viende dado por 'N'.
se=strel('line', 50, 0);
```

```
Icierre = imclose(Iaux,se);
figure(3)
imshow(Icierre,2);
pause;

 % Etiquetamos las componentes existentes en la imagen binaria 'Icierre'.
[Ilabel,nobj]=bwlabel(Icierre);

areas=zeros(1,nobj);
perim=zeros(1,nobj);
ObjectsOk = zeros(size(Iaux));
ObjectsNotOk = zeros(size(Iaux));

 % Se recorren todos los objetos de la imagen.
for j=1:nobj
     % Se obtienen las coordenadas de cada objeto de la imagen.
    [n,m]=find(Ilabel==j);
     % Se crea una matriz de ceros del tamaño de la imagen original.
    G = zeros(size(Iaux));

    for i=1:length(n)
        G(n(i),m(i))=1;
    end
    imshow(G);

     % bwarea(BW); Estima el área de los objetos de la imagen binaria BW.
     % Devuelve un escalar cuyo valor corresponde aproximadamente (podrían
     % no ser los mismos) con el número total de píxeles de la imagen.
    areas(j)=bwarea(G);

     % Realizando la detección atendiendo a la propiedad 'Área'.
    if(areas(j) > 10000)
        for i=1:length(n)
            ObjectsOk(n(i),m(i))=1;
        end
    else
        for i=1:length(n)
            ObjectsNotOk(n(i),m(i))=1;
        end
    end

     % bwperim (BW); Devuelve una imagen binaria que contiene sólo los
     % píxeles del perímetro de los objetos en la imagen de entrada BW. Un
     % píxel es parte del perímetro, si es distinto de cero y está conectado
```

```
    % a él, si al menos tiene un píxel con valor cero. La conectividad por
    % defecto para dos dimensiones es 4, y 6 para tres dimensiones.
    % Para ir mostrando el perímetro del objeto.
    Iperim = bwperim(G);
    imshow(Iperim,2);
    perim(j)=bwarea(Iperim);

%Realizando la detección atendiendo a la propiedad 'Perímetro'.
%    if(perim(j) > 750)
%        for i=1:length(n)
%            ObjectsOk(n(i),m(i))=1;
%        end
%    else
%        for i=1:length(n)
%            ObjectsNotOk(n(i),m(i))=1;
%        end
%    end

    % Para no tener que estar pulsando una tecla, las pausas se realizan de
    % un segundo.
    pause(1);
end

figure(5);
imshow(ObjectsOk,2);
title('Tijeras que pasan el control de calidad', 'color', 'b');

figure(6);
imshow(ObjectsNotOk,2);
title('Tijeras que no pasan el control de calidad (defectuosas)', 'color', 'r');
```

A.21. Función *aplicDefectB.m*

```
function aplicDefectB(rutaImagen)

I = imread(rutaImagen);

% Se realiza la binarización de la imagen 'I'(RGB) por umbral (thresholding).
Ibin=im2bw(I,0.5);
% Para mostrar la imagen en blanco y negro.
figure(1);
imshow(Ibin,2);
```

```
% ' ' => Equivale a aplicar la función lógica NOT.
% Invertimos los colores blanco y negro.
Iaux= Ibin;
figure(2);
imshow(Iaux, 2);

% strel('disk',R,N); Crea un disco plano en forma de elemento estructurante
% con radio, R. Si N es omitido, toma su valor por defecto -> 4.
% Dado que el valor por defecto, para el segundo parámetro de la función
% strel cuando se usa como elemento estructurante 'disk' es 4, y no quedan
% muy bien los resultados, se seleccina el valor '8' para dicho parámetro,
% y se suaviza la solución.
se = strel('disk',26, 8);

Icierre = imclose(Iaux,se);
figure(3);
imshow(Icierre,2);
pause;

% Etiquetamos las componentes existentes en la imagen binaria 'Icierre'.
[Ilabel,nobj]=bwlabel(Icierre);

areas=zeros(1,nobj);
perim=zeros(1,nobj);
ObjectsOk = zeros(size(Iaux));
ObjectsNotOk = zeros(size(Iaux));

% Se recorren todos los objetos de la imagen.
for j=1:nobj
    % Se obtienen las coordenadas de cada objeto de la imagen.
    [n,m]=find(Ilabel==j);
    % Se crea una matriz de ceros del tamaño de la imagen original.
    G=zeros(size(Iaux));

    for i=1:length(n)
        G(n(i),m(i))=1;
    end
    imshow(G);

    % bwarea(BW); Estima el área de los objetos de la imagen binaria BW.
    % Devuelve un escalar cuyo valor corresponde aproximadamente (podrían
    % no ser los mismos) con el número total de píxeles de la imagen.
    areas(j)=bwarea(G);
```

```
    % Realizando la detección atendiendo a la propiedad 'Área'.
    if(areas(j) > 10000)
        for i=1:length(n)
            ObjectsOk(n(i),m(i))=1;
        end
    else
        for i=1:length(n)
            ObjectsNotOk(n(i),m(i))=1;
        end
    end

    % bwperim (BW); Devuelve una imagen binaria que contiene sólo los
    % píxeles del perímetro de los objetos en la imagen de entrada BW. Un
    % píxel es parte del perímetro, si es distinto de cero y está conectado
    % a él, si al menos tiene un píxel con valor cero. La conectividad por
    % defecto para dos dimensiones es 4, y 6 para tres dimensiones.
    % Para ir mostrando el perímetro del objeto.
    Iperim = bwperim(G);
    imshow(Iperim,2);
    perim(j)=bwarea(Iperim);

% Realizando la detección atendiendo a la propiedad 'Perímetro'.
%     if(perim(j) < 800)
%         for i=1:length(n)
%             ObjectsOk(n(i),m(i))=1;
%         end
%     else
%         for i=1:length(n)
%             ObjectsNotOk(n(i),m(i))=1;
%         end
%     end

    % Para no tener que estar pulsando una tecla, las pausas se realizan de
    % un segundo.
    pause(1);
end

figure(4);
imshow(ObjectsOk,2);
title('Leds que pasan el control de calidad', 'color', 'b');

figure(5);
imshow(ObjectsNotOk,2);
title('Leds que no pasan el control de calidad', 'color', 'r');
```

Referencias Bibliográficas

[1] Rafael C. Gonzalez y Richard E. Woods, "Digital Image Processing", Third edition, New Yersey: Pearson Prentice Hall, 2008.

[2] Scott T. Smith, "Matlab Advanced GUI Development", Indianápolis: Dog-Ear Publishing, 2006.

[3] Stephen J. Chapman "Matlab Programming for Engineers", Fourth Edition. Canadá: Thomson Learning, 2008.

[4] Ioannis Pitas, "Digital Image Processing Algorithms", Prentice Hall: New York, 1993.

[5] Ana González Marcos, Francisco Javier Martínez de Pisón Ascacíbar, Alpha Verónica Pernía Espinoza, Fernando Alba Elías, Manuel Castejón Limas, Joaquín Ordieres Meré y Eliseo Vergara González, "Técnicas y algoritmos básicos de visión artificial", Logroño: Universidad de La Rioja, Servicio de Publicaciones, 2006.

[6] Antonio Peralta Sáez, Proyecto Fin de Carrera "Reconocimiento de imágenes a través de su contenido", Madrid: Universidad Pontificia Comillas, Junio 2009.

[7] José Ramón Alcalá Mellado y Guillermo Navarro Oltra, "Una introducción a la imagen digital y su tratamiento", Cuenca: MIDECIANT, 2008.
`http://www.uclm.es/profesorado/gnoltra/publicaciones/mideciantdidact1.html`

[8] B. L. Littlefield y D. C. Hanselman, "Mastering Matlab 7", Prentice-Hall, 2004.

[9] "Image Processing Toolbox",
`http://www.mathworks.es/products/datasheets/pdf/image-processing-toolbox.pdf`

[10] Roshan Dharshana Yapa y Koichi Harada, "A connected component labeling algorithms for grayscale images and application of the algorithm on digital mammograms", New York: ACM Press, 2007.

[11] Rafael C. Gonzalez, Richard E. Woods y Steven L.Eddins, "Digital Image Processing Using Matlab", New Yersey: Pearson Prentice Hall, 2004.

[12] Mehmet Sezgin y Bülent Sankur, “Survey over Image Thresholding Techniques and Quantitative Performance Evaluation”, Publicación en revista de tratamiento digital de imágenes, Enero 2004.

[13] Miguel A. Jaramillo, J. Alvarado Fernández y E. Martínez de Salazar, “Implementación del detector de Bordes de Canny sobre Redes Neuronales Celulares”, Universidad de Extremadura.

[14] Roger Boyle y Richard C. Thomas, “Computer Vision: A First Course”, Blackwell Scientific Publications, 1990.

[15] Javier García Galón. “Aprenda MatLab 7.0 como si estuvieras en primero”, Universidad Politécnica Madrid Ediciones, 2007.

[16] Milan Sonka, Vaclav Hlavac y Roger Boyle, “Image Processing, Analisis, and Machine Vision” Second edition, PWS Publishing, 1999.

[17] William K. Pratt, “Digital Image Processing”, Third edition, New York: John Wiley & Sons, 2001.

[18] N. Otsu, “A Threshold Selection Method from Gray-Level Histograms”, IEEE Transactions on Systems, Man, and Cybernetics, 1979

[19] Tao Wen-Bin, Tian Jin-Wen y Jian Liu, “Image segmentation by three-level thresholding based on maximum fuzzy entropy and genetic algorithm”, Institute for Pattern Recognition and Artificial Intelligence, China, 2003.

[20] James C. Bezdek, James Keller, Raghu Krisnapuram y Nikhil R. Pal, “Fuzzy Models and Algorithms for Pattern Recognition and Image Processing”, New York: Springer, 2005.

[21] Stuart Hoggar, “Mathematics of Digital Images”, Cambridge University Press, Septiembre 2006.

[22] Liyuan Li, Weimin Huang, Yu-HuA Gu y Qi Tian, “Statistical modeling of complex back-ground for foreground object detection”, IEEE Transactions on Image Processing, Noviembre 2004.

[23] John C. Russ y J. Christian Russ, “Introduction to Image Processing and Analysis”, CRC Press, Octubre 2007.

[24] Qivind Due Trier y Anil K. Jain, “Goal-Directed Evaluation of Binarization Methods”, IEEE Pattern Analysis and Machine Intelligence, Diciembre 1995.

[25] J. Zheng, Yinhai Wang, Nancy L. Nihan, y Mark E. Hallenbeck, “Extracting Roadway Background Image: A Mode Based Approach”, Revista de investigación, Marzo 2006.

[26] Kenneth R. Castleman, “Digital Image Processing”, Prentice Hall, 1996.

[27] Milan Sonka, Vaclav Hlavac y Roger Boyle, “Image Processing, Analisis, and Machine Vision” Third edition, Thompson Learning, 2008.

[28] S. Cheung y Chandrika Kamath, “Robust techniques background subtraction with foreground validation for urban traffic video”, Revista de procesamiento digital de señales especial sistemas de visión inteligente, 2005.

[29] Michael Seul, Lawrence O’Gorman y Michael J. Sammon, “Practical Algorithms for Image Analysis”, Cambridge University Press, 2000.

[30] Gérard Blanchet y Maurice Charbit, “Digital Signal and Image Processing Using Matlab”, Great Britain, 2006.

[31] E. Roy Davies “Machine Vision: Theory, Algorithms and Practicalities”, Third edition, London: Elsevier, 2005.